

《计算机组成原理》 实 验 指 导 书

张宇华 编写

广东工业大学计算机学院

二〇一九年十二月

目 录

第一章 TEC-XP+计算机组成原理实验系统技术说明..... 1

§1.1 TEC-XP+计算机组成原理实验系统概述	1
§1.2 TEC-XP+指令系统	10
§1.3 TEC-XP+运算器部件	14
§1.4 TEC-XP+内存储器部件	16
§1.5 TEC-XP+控制器部件	18
§1.6 TEC-XP+输入输出及中断	22
§1.7 TEC-XP (FPGA) 计算机硬件系统	23
§1.8 教学机的 PC 机仿真终端程序	26

第二章 TEC-XP+实验系统实验内容..... 27

实验一 基础汇编语言程序设计	27
实验二 脱机运算器实验	41
实验三 存储器部件教学实验	45
实验四 组合逻辑控制器部件教学实验	51
实验五 微程序控制器部件教学实验	61
实验六 输入/输出接口扩展实验	67
实验七 中断实验	69
实验八 8 位模型机的设计与实现(综合实验)	75
实验九 FPGA 芯片实现非流水线的 CPU 系统(综合实验)	77

附录..... 80

附录 1 TEC-XP (FPGA) 计算机硬件系统及其实验内容	80
A.1 TEC-XP (FPGA) 教学计算机系统组成概述	80
A.2 VHDL 语言概述	83
A.3 在 FPGA 芯片中实现的非流水线的 CPU 系统	85
附录 2 教学计算机的软件模拟系统	109
附录 3 联机通讯指南	130
附录 4 TEC-XP+计算机组成原理实验系统简明操作卡	133
附录 5 微程序入口地址映射表	134
附录 6 指令流程框图	136
附录 7 指令流程表	138
附录 8 书写实验报告的一般格式	142

第一章 TEC-XP+计算机组成原理实验系统 技术说明

§1.1 TEC-XP+计算机组成原理实验系统概述

一、教学计算机系统的实现方案和硬软件资源概述

TEC-XP+由清华大学科教仪器厂和清华大学计算机系联合研制并通过了教育部主持的鉴定。是在原有TEC系列教学计算机系统的基础上，重新设计的新一代产品，进一步增加了用单片FPGA门阵列器件实现的CPU系统。该实验系统重点用于计算机组成原理和计算机系统结构等课程的硬件教学实验，还支持监控程序、汇编语言程序设计、BASIC高级语言程序设计等软件方面的教学实验。它的功能设计和实现技术，都紧紧地围绕着对课程教学内容的覆盖程度和所能完成的教学实验项目的质量与水平来进行安排。其突出特点有二，一是硬、软件基本配置比较完整，能覆盖相关课程主要教学内容，支持的教学实验项目多且水平高，文字与图纸资料相对齐全。二是既有用不同集成度的半导体器件实现的真实“硬件”计算机系统，同时还有在PC计算机上用软件实现的功能完全相同的教学计算机的“软件”模拟系统。其组成和实现的功能如图1.1.1、图1.1.2所示。

软件：解释 BASIC 语言 汇编语言支持 监控程序 硬件：运算器，控制器（多种实现： （微程序或硬布线控制器， 中小规模器件或 FPGA 器件实现） 主存储器，总线，接口 输入设备，输出设备
硬件与电路：逻辑器件和设备

图1.1.1 硬件实现的实际计算机系统图

软件：解释 BASIC 语言 汇编语言支持 监控程序（指令）级模拟 教学机模拟：运算器、控制器模拟 （微程序级或硬布线控制器级模拟） 主存储器模拟，总线、接口模拟 输入设备/ 输出设备模拟
运行环境：PC 机，Windows 系统

图1.1.2 软件实现的模拟计算机系统

从图1.1.1可以看到，该计算机硬件系统组成中，功能部件是完整齐备的，运算器、控制器、存储器、输入输出接口、计算机总线等配备齐全，还可以接通PC机仿真终端执行输入输出操作，同时实现了微程序方案的和硬连线方案的2种控制器。

从计算机组成原理课程教学需求的角度看，该计算机软件系统的组成也是完整的，支持

简单的高级语言（包括浮点运算指令和基本函数运算功能），汇编语言（支持基本伪指令功能）和二进制的机器语言，配有自己的监控程序，以及PC机仿真终端程序等。

从图 1.1.2 可以看到，软件实现的计算机指令级模拟系统，可以使实验人员脱离实际的教学计算机系统，在 PC 机上执行教学计算机软件系统的全部功能；微程序和硬连线这一级别的模拟软件，可以通过 PC 机屏幕查看在教学计算机内部数据、指令的流动过程，并显示每一步的运行结果，为设计、调试教学机新的软件或硬件功能提供重要的辅助作用。

二、教学计算机指令系统的设计目标和指令格式

合理地确定一台计算机的指令系统，无论对计算机厂家还是对最终用户来说都是十分重要的事情，它密切关系到计算机设计与实现的复杂程度和生产成本，计算机使用的难易程度和运行效率。对主要用于教学和教学实验目的的计算机，特别是对于一台16位字长的教学计算机来说，确定其指令系统，更多地应关注它在教学过程中的作用和使用方法，至少应解决好以下几个问题：

- 1、指令格式和功能的典型性，即选择DLX指令集结构，适当靠拢RISC机的指令格式，包括尽可能小的指令集，简化的寻址方式。这样做不仅可以简化教学计算机的结构，实现简单，易于实现指令流水，重要的是选用有良好典型性的指令格式和功能，讲课时更容易完整地讲解清楚这套指令系统和控制器设计，有利于教学内容的整体安排。
- 2、指令系统要有一定的完备程度，给出的指令格式适当规范，指令分类合理，指令执行步骤容易理解，符合人们通常的编程使用习惯，有较好的易学易用性，确保选用这套指令系统，能方便地设计教学计算机的配套软件。
- 3、更高的可扩充性，即为学生添加各种新的指令留下比较充足的余地，为此可以把完整系统中的指令划分为必备的基本指令（由设计者实现）和待扩展的保留指令（由学生设计实现）2大类；在扩展新的指令时，实现手段要适当简单，但要有比较多的设计内容和选择余地，以便更好地培养学生的创新意识和开创能力，有利于深化教学内容。
- 4、符合教学计算机的特定要求。对16位字长的计算机，指令的操作码部分可以选择为固定长度；再结合我们所选用的运算器器件Am2901芯片内含16个通用寄存器的特点，寄存器寻址方式需要使用4位的形式地址。如果需要，还可以指定16个累加器中的几个为专用的寄存器，以便最大程度地简化教学机硬件组成，简化指令执行流程设计。

遵照上述思路，最终确定了教学计算机的指令格式，如图1.1.3所示。这套指令系统支持单字指令和双字指令，第一个指令字的高8位是指令操作码字段，低8位和双字指令的第二个指令字是操作数地址字段，分别有3种用法。

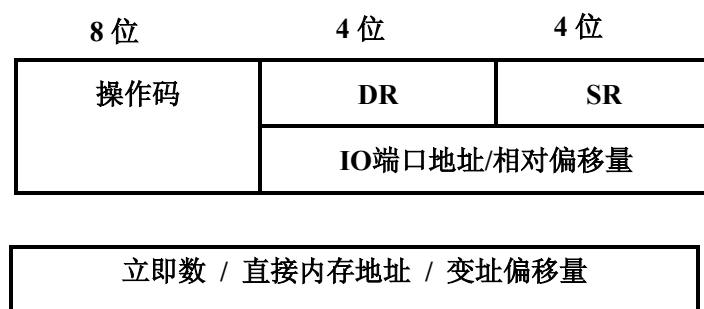


图1.1.3 教学机的指令格式

8位指令操作码（记作“IR15~IR8”），各位的含义如下：

IR15、IR14 用于区分指令组：0×表示A组，10表示B组，11表示C、D组；

IR13 用于区分基本和扩展指令：0表示基本指令，1表示扩展指令；

IR12 用于简化控制器的实现，暂定该位的值为0；

IR11~IR8 用于区分同一指令组中的不同指令（最多16条）；

IR11 还用于区分C、D组指令（每组最多8条）：0表示C组，1表示D组。

第一个指令字中的操作数地址字段可以给出：4位的通用寄存器编号（DR代表目的寄存器，SR代表源寄存器），8位的IO端口地址，8位的相对变址偏移量。第二个指令字中的操作数地址字段用于给出16位的立即数，16位的直接内存地址，或者16位的变址偏移量。

三、教学计算机的硬件组成和设计概述

TEC-XP+教学计算机系统由3个功能子系统组成：1. 利用中小规模电路设计实现的16位字长的教学计算机系统；2. 用中小规模电路设计实现的8位字长的教学计算机系统；3. 用FPGA门阵列器件设计实现的单芯片CPU组成的教学计算机系统。

进一步把 TEC-XP+的三个功能子系统各自设计成相对独立的产品，分别命名为 TEC-XP16、TEC-XP8、TEC-XP(FPGA))。

TEC-XP+是一个是双 CPU 的教学计算机系统。第 1 个 CPU 主要是选用中小规模集成电路实现的，分别支持 16 位和 8 位两种字长，都支持微程序和组合逻辑两种控制器方案；第 2 个 CPU 则是选用高集成度的门阵列 FPGA 芯片实现的，两个 CPU 运行相同的指令系统，以便保证软件系统的兼容性。使用者能够方便地利用在设备上的 FPGA 芯片设计实现另外一个全新的 CPU 系统。

TEC-XP+教学计算机系统外观图如图 1. 1. 4 所示，TEC-XP+教学计算机系统组成如图 1. 1. 5 所示。

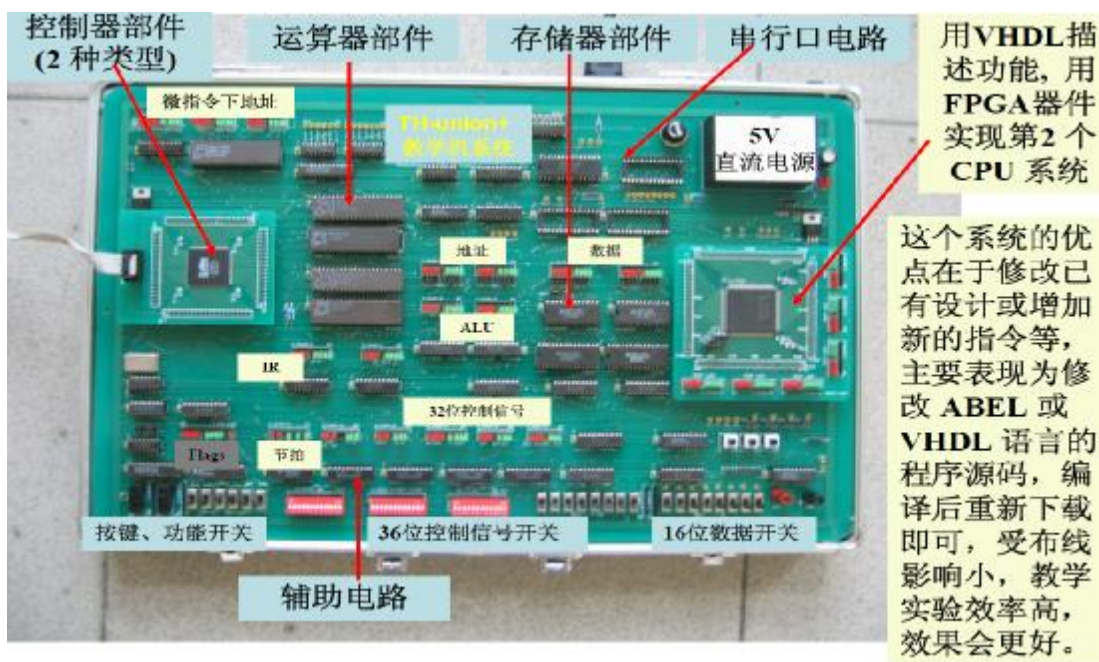


图 1.1.4 教学计算机系统外观图

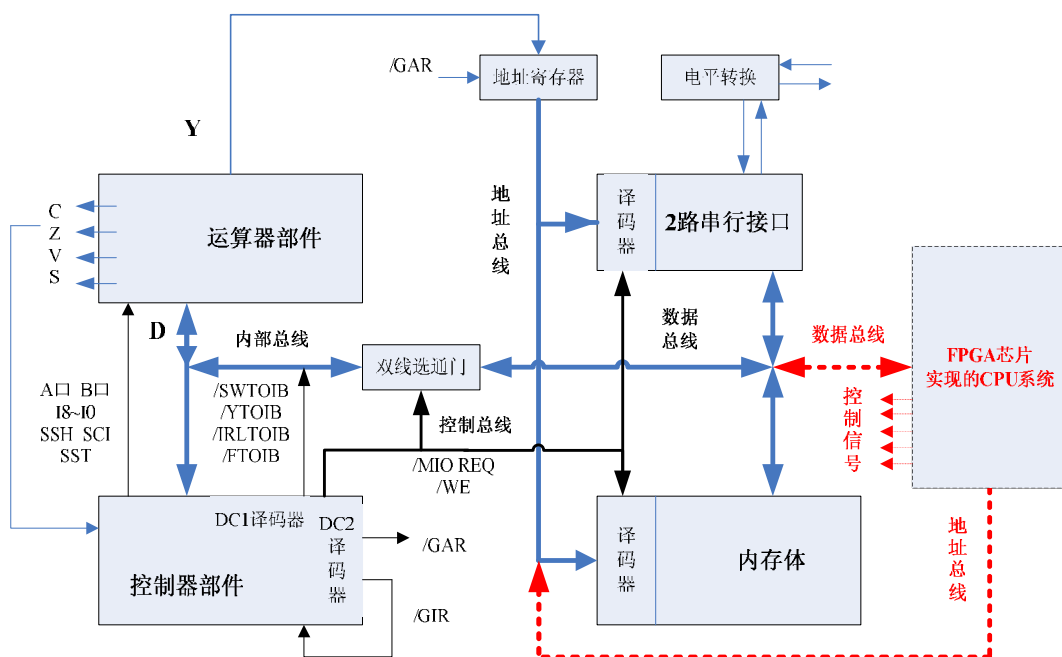


图 1.1.5 双CPU的TEC-XP+ 硬件组成框图

在图的左部所表示的是选用中小规模器件实现的第1个 CPU 系统，由独立的运算器、控制器部件组成。图的中间部分所表示的是内存储器、串行接口线路的组成。图的右部虚线部分所表示的是选用 FPGA 门阵列器件实现的单个芯片的第2个 CPU 系统。这两个 CPU 系统都可以

通过数据总线、地址总线和控制总线连接内存储器、串行接口线路，从而构成一台完整的计算机硬件系统，安装上必要的软件就可以正常运行，作为计算机组成原理课程内容实例和教学实验设备具有很好的典型性。两个 CPU 系统需要通过分时或者独占的方式使用同一套存储器部件和串行接口线路。

四、TEC-XP 16 系统的硬件系统

TEC-XP 16系统是TEC-XP+的一个重要组成部分。作为教学和教学实验使用的计算机，其硬件结构和组成设计，要比较好地体现出尽可能多的主要教学内容，包括功能部件划分清晰，设计合理，它们之间连接关系适当规范等。

TEC-XP16的硬件系统由以下几个基本部分组成：运算器部件、控制器部件、内存储器系统和串行接口线路；此外还设置了辅助电路和扩展电路两个辅助部分，各个部分被划分在电路板的不同区域，基本组成部分的线路逻辑框图如图1.1.6所示。

从图1.1.6中我们可以看到，教学计算机运算器部件是选用4片位片结构的4位长度的运算器Am2901芯片实现的。该芯片包含完成算术和逻辑运算功能的ALU，双端口控制读出、单端口控制写入的16个累加器和完成乘除法运算的乘商寄存器等功能部件，从功能和组成两个方面都比较好地体现了运算器部件的教学内容。从图1.1.6可以看到运算器和其它部件的连接关系，它只能接收教学机内部总线IB送来的16位数据，其运算结果直接送到地址寄存器AR的输入端，或者经过2个8位的开关门电路送到内部总线IB。运算结果的标志位信息送到标志位寄存器FLAG，FLAG的输出可以经过一个8位的开关门送到内部总线IB。

在教学计算机控制器部件设计中，同时实现了微程序和硬连线的两种控制器，并可以通过拨动一个开关完成两种控制器之间的切换。两种控制器主要都由一片高集成度MACH器件实现，这一实现方案为简化修改与扩展控制器功能的操作，改善教学实验效果有重要作用，是本教学计算机系统非常显著的一个特色。在MACH芯片之外，还用到了确定微指令执行次序的一片Am2910芯片，用作指令寄存器IR的2片8位的寄存器电路，1片传送IR低位字节内容到内部总线IB的开关门电路。指令寄存器接收从内存储器读出并传送到内部总线IB的指令，其全部16位输出送到MACH芯片的输入引脚，其低8位内容还要经一个开关门送到内部总线IB。

在教学计算机存储器部件设计中，出于简化和容易实现的目的，选用静态存储器芯片实现内存储器系统，包括了唯读存储区（ROM，存放监控程序等）和随读写存储区（RAM）两部分，ROM存储区选用4片长度8位、容量8KB的芯片实现，RAM存储区选用2片长度8位、容量2KB的芯片实现，每2个8位的芯片合成一组用于组成16位长度的内存字，6个芯片被分成3

组，其地址空间分配关系是：0-1777h用于第一组ROM，固化监控程序，2000-2777h用于RAM，保存用户程序和用户数据，其高端的一些单元作为监控程序的数据区，第二组ROM的地址范围可以由用户选择，主要用于完成扩展内存容量的教学实验。

关于计算机中的接口线路，教学计算机提供了2路串行接口（INTEL 8251），以支持接入PC机作为教学计算机的仿真终端完成输入输出操作；第一个串口的端口地址分配80h/81h，第二个串口的端口地址可以由用户选择。作为扩展实验内容，也可以通过在一个40芯的器件插座上插上其他标准接口线路（例如INTEL 的8255、8253 等）并适当接线，完成常用接口线路的输入输出操作。在教学计算机总线部件设计中，选用单总线结构，数据总线、地址总线和控制总线都比较简单，保证教学机的正常运行并体现出总线设计的基本原理。图1.1.7给出了各个部件如何通过总线相互连接在一起，从而构成一台能够正常运行的计算机系统。

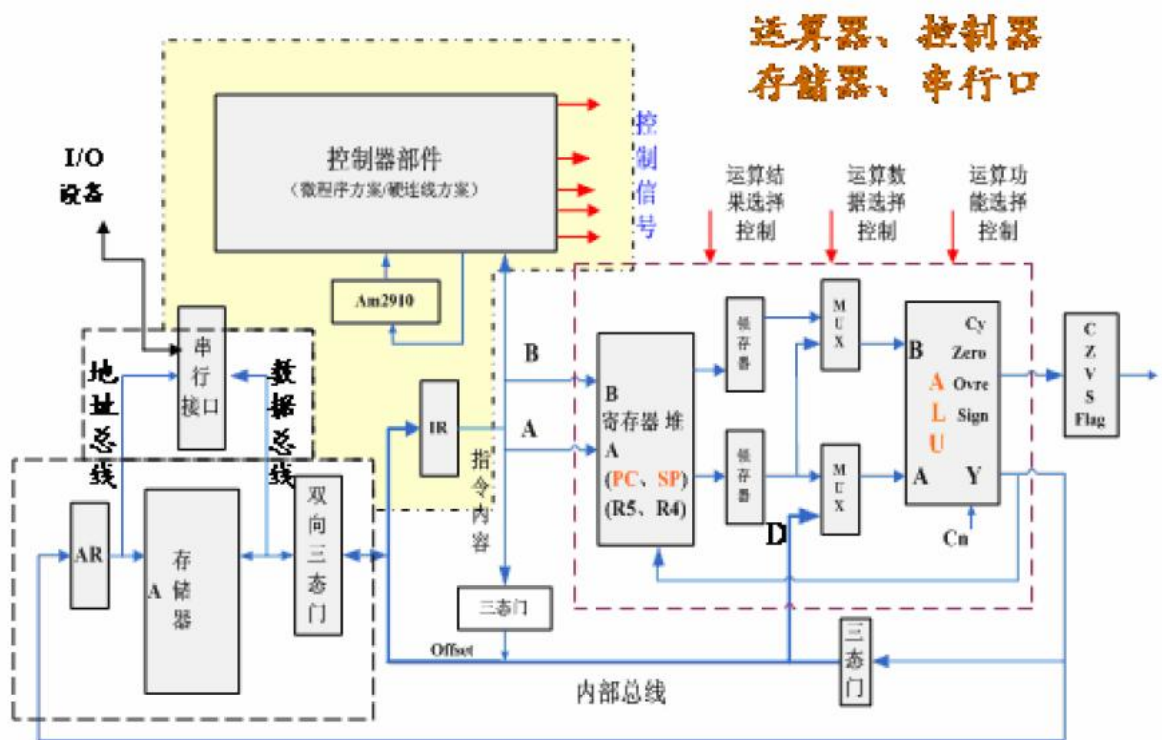
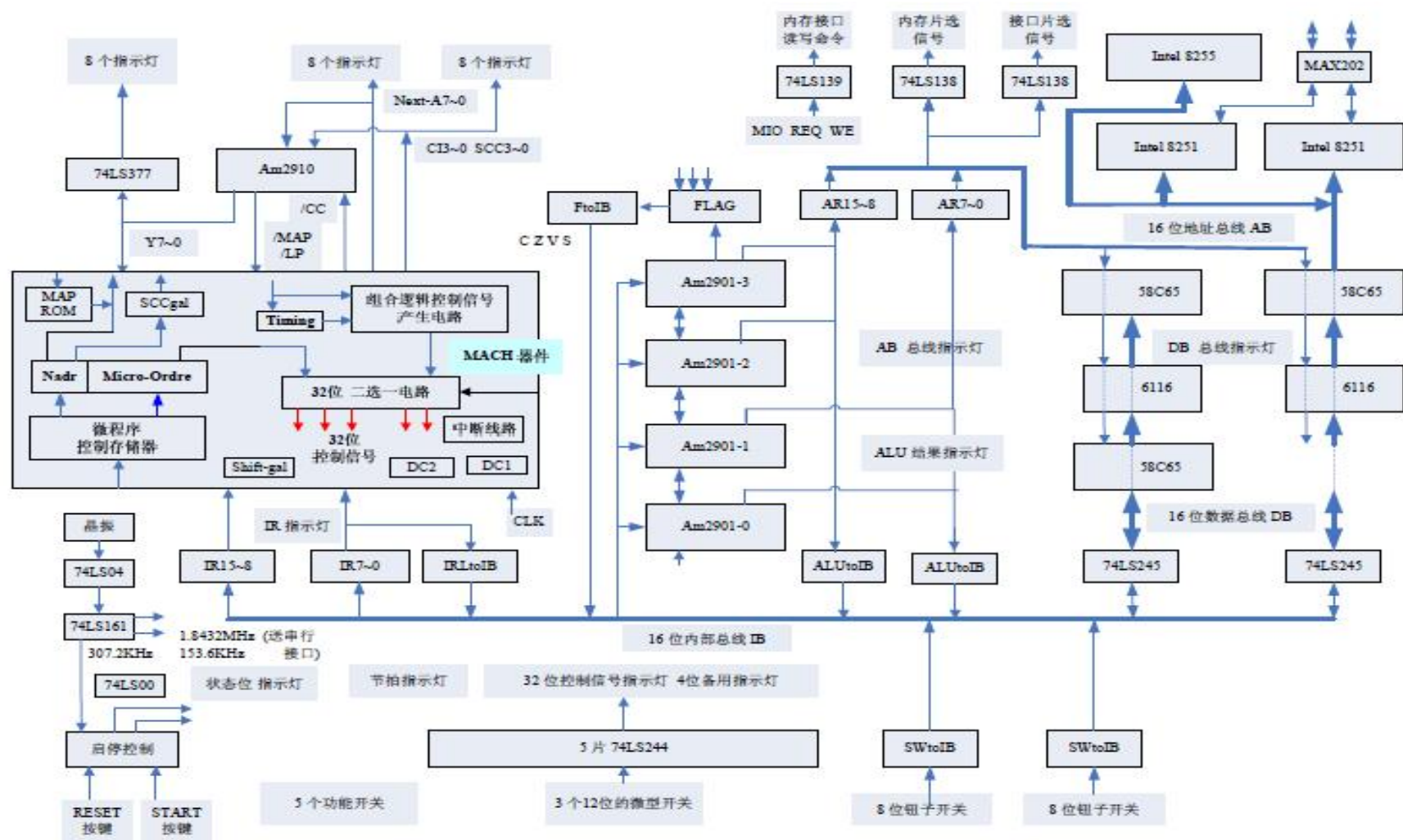


图1.1.7 部件之间的连接关系和信息传送路径



通过图1.1.7简单地看一下教学计算机每个部件的运行环境和相关功能的执行过程。

数据总线被划分成内部总线IB（在CPU一侧）和外部总线DB（在存储器和串行接口一侧）两部分，它们之间通过2片8位的双向三态门电路连接在一起。3组存储器芯片的数据输入输出引脚都直接连接在一起，连接到16位的数据总线DB上，串行接口芯片的8位数据输入输出引脚只与数据总线DB的低8位（DB7-DB0这8位）相连接。

地址总线的构成比较特殊，仅地址寄存器AR一个来源，AR又只接收ALU一路输入。内存储器和接口电路的地址都来自于地址总线AB（地址寄存器AR 的输出），地址总线的最高3位送到1片3-8译码器，地址总线的低位字节中的高4位（规定最高一位必定为1）送到另外1片3-8译码器，分别产生存储器芯片的8个片选信号和接口电路的8个片选信号。

控制总线提供内存和串口的读写命令，是把控制器提供的3位控制信号送1片双2-4译码器得到的，以决定有无内存或接口读写，若有，是和内存还是接口工作，执行的是读还是写操作。内存和CPU选用同步方式运行，串行接口和CPU选用状态查询方式工作。

运算器部件中的ALU可以对两路输入数据A和B执行3种算术或5种逻辑运算功能，其两路输入可来自芯片内部的寄存器堆送出来的数据（是由指令寄存器IR的两个寄存器编号选定的寄存器的内容，还设置有锁存线路），或来自芯片外的内部总线IB的数据，其运算结果可以在芯片内部被直接写入寄存器堆，或送到芯片外被直接连接到地址寄存器AR、或经过支持三态逻辑的开关门送到内部总线IB。ALU运算产生的4个标志位的值被保存进芯片外部的Flag 寄存器。运算器部件中的寄存器堆暂存用于ALU运算的数据和运算的中间结果。

控制器部件将依据指令内容和指令执行步骤信息来提供管理计算机各个部件运行所必需的控制信号，指令寄存器IR接收从内存储器读出来的指令内容，其输出被送到MACH芯片的输入引脚，由MACH（也包括Am2910芯片，图中未画）产生指令执行步骤信号，并为各个部件提供每一个执行步骤要用到的全部控制信号。指令寄存器低位字节的内容可以经过带三态逻辑的开关门送到内部总线，用作为运算器部件的外部输入数据D的一个来源。

内存储器用于保存运行中的程序和数据，可读可写。读写操作的第一步是为其提供内存单元的地址，即把ALU的输出内容写进地址寄存器AR，第二步是执行读或者写操作。为读操作时，若读出的是指令则经过数据总线DB和内部总线IB写进指令寄存器IR，若读出的是数据应经过数据总线DB和内部总线IB、经过运算器的D输入引脚写到运算器寄存器堆中一个寄存器。这个寄存器由指令字中的目的寄存器字段指定。为写操作时，把由指令字的一个字段指定的寄存器堆中的一个寄存器的内容经内部总线IB和数据总线DB写进存储器的一个存储单元。

串行接口用于执行数据的输入输出操作。输入输出操作的第一步是为接口芯片提供入出端口地址，即把指令寄存器低位字节的内容（IO端口地址）经过内部总线和运算器部件写进地址寄存器AR，第二步是执行输入或者输出操作，若执行输入指令IN，则应从接口芯片读出一个8位的数据并经过数据总线DB和内部总线IB写进寄存器堆中的R0寄存器，若执行输出指令OUT，则需要把寄存器堆中的R0寄存器的内容经过内部总线IB和数据总线DB写入接口芯片。接口芯片与输入输出设备之间的数据传送过程无需另外管理，会自动完成。

在教学计算机系统中，实现并提供了简明、常规的中断处理能力，在支持多级的中断嵌套实验方面，这个系列的教学计算机有非常明显的特色。

五、主要技术指标

1. 教学机系统配置了两个不同实现方案的 CPU 系统，一个 CPU 沿袭传统的设计思路，和当前主流的教材配套，由中小规模的器件组成；另一个 CPU 参考国外著名大学的设计思路用大规模的 FPGA 器件设计实现。
2. 教学机的机器字长 16 位，即运算器、主存、数据总线、地址总线都是 16 位。
3. 指令系统支持多种基本寻址方式。其中一部分指令已实现，用于设计监控程序和用户的常规汇编程序，尚保留多条指令供实验者自己实现。
4. 主存最大寻址空间是 18K 字，由基本容量为 8K（字节或 16 位的字）的 ROM 和 2K（字节或 16 位的字）的 RAM 存储区域组成。还可以进一步完成存储器扩展的教学实验。
5. 原理上讲，主时钟脉冲的频率可在几百 KHz~近 2MHz 之间选择。
6. 运算器由 4 片位片结构器件级联而成，片间用串行进位方式传递进位信号。ALU 实现 8 种算术与逻辑运算功能，内部包括 16 个双端口读出、单端口写入的通用寄存器，和一个能自行移位的乘商寄存器。设置 C（进位）、Z（结果为 0）、V（溢出）和 S（符号位）四个状态标志位。
7. 控制器采用微程序和硬布线两种控制方案实现，可由实验者自由选择。实验人员可方便地修改已有设计，或加进若干条自己设计与实现的新指令，新老指令同时运行。
8. 主机上安装有两路 INTEL8251 串行接口，一路出厂时已经实现，可直接接计算机终端，或接入一台 PC 机作为自己的仿真终端；另一路保留学生扩展实现。选用了 MAX202 倍压线路，以避免使用+12V 和-12V 电源。
9. 在主板的右下方，配置了完成中断教学实验的全套线路，可以实现三级中断和中断嵌套。
10. 系统实现多种运行方式，可以单步/连续运行主存储器的指令或程序，也可以执行一条或若干条通过数据开关手动置入的指令。
11. 主板上设置数据开关和微型开关、按键和指示灯，支持最低层的手工操作方式的输入/输出，通过指示灯来显示重要的数据或控制信号的状态，可以完成机器调试和故障诊断。板上还支持教学实验用的一定数量的跳线夹。
12. 实验机硬件系统，全部功能部件分区域划分在大一些的水平放置的一块印制电路板的不同区域，所有器件都用插座插接在印制板上，便于更换器件。
13. 实验计算机使用单一的 5V、最大电流 3A 的直流模块电源，所耗电流在 1.5~2.5A 之间。电源模块安装在水平电路板右上角位置，交流 220V 通过电源接线插到机箱后侧板，经保险丝、开关连接到电路板上，开关安放在机箱右侧靠后位置，方便操作且比较安全。
14. 两路的串行接口的接插座安放在机箱后侧板以方便接线插拔和机箱盖的打开关闭。

§1.2 TEC-XP+指令系统

一、 教学计算机的指令系统概述

在字长为16位的教学计算机系统中，规定指令的位数也是16位的倍数，支持单字指令和双字指令，指令格式要规范和简单，尽量向DLX指令集结构靠拢，使其有一定的完备性和更好的典型性。从有利于教学实验考虑，把指令划分为基本指令（已经由设计者和制作厂家实现）和扩展指令（留待进行教学实验的同学实现）两大类。

对指令的格式说明如图1.2.1所示：

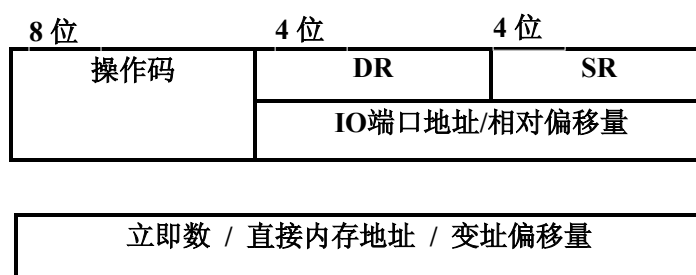


图1.2.1 指令格式

指令操作码由8位组成（记作“IR15~IR8”），各位的控制作用有所不同：

(1) IR15、IR14用于区分指令组：0X表示A组，10表示B组，11表示C、D组；

还要用IR11区分C、D组：IRH11=0为C组，IRH11=1为D组。

(2) IR13用于区分基本和扩展指令：IRH13=0为基本指令，IRH13=1为扩展指令。

(3) IR12用于简化控制器实现，暂定该位的值均为0。

(4) IR11~IR8用于区分同一指令组中的不同指令。

按不同的分类标准，可以把16位机的指令划分成不同的指令组，例如：

从指令长度区分，有单字指令和双字指令，也允许定义与使用3字指令。

从操作数的个数区分，有双操作数指令、单操作数指令和无操作数指令。

从使用的寻址方式区分，有采用寄存器寻址、寄存器间接寻址、立即数寻址、直接寻址、变址寻址、相对寻址、堆栈寻址等多种基本寻址方式的不同类别指令。

从指令功能区分，有算术和逻辑运算类指令、读写内存类指令、输入/输出类指令、转移指令、子程序调用和返回类指令，还有传送、移位、置进位标志和清进位标志等指令。

依照指令的执行步骤，可以把教学计算机的指令划分为如下4 组。

A组：基本指令ADD、SUB、AND、OR、XOR、CMP、TEST、MVRR、DEC、INC、SHL、SHR、JR、JRC、JRPC、JRZ、JRNZ

扩展指令ADC、SBB、RCL、RCR、ASR、NOT、CLC、STC、EI、CI、JRS、JRNS、JMPR

B组：基本指令JMPA、LDRR、STRR、PUSH、POP、PSHF、POPF、MVRD、IN、OUT、RET

C组：扩展指令CALR、LDRA、STRA、LDRX、STRX

D组：基本指令CALA

扩展指令IRET

这种分类办法，是为了突出指令执行步骤的划分结果，有利于讲解控制器设计技术。

A组指令完成的是通用寄存器之间的数据运算或传送，或其它几项特殊的操作，在取指之后可一步完成。

B组指令完成的是一次内存或I/O读、写操作，在取指之后可两步完成，第一步把要使用的地址传送到地址寄存器AR中，第二步执行内存或I/O读、写操作。

C组指令在取指之后可三步完成，其中CALR指令在用两步完成一次写内存之后，第三步执行寄存器之间的数据传送；而其它指令在第一步置地址寄存器AR，第二步读内存（即取得一个内存单元的地址）并传送到地址寄存器AR，第三步执行另外一次读、写内存的操作。

D组指令完成的是两次读、写内存操作，在取指之后可四步完成。

十六位的教学机系统，实现了上面4组中的29条基本指令，用于支持教学机的监控程序和简单的汇编语言程序设计。保留了其余19条扩展指令，供学生在教学实验中进行扩展，即完成对这些指令的设计与调试，当然，还可以扩展另外一些指令。

为了支持汇编语言程序设计，每一条指令分配了一个汇编语句名，其命名规则是：

用一个英文单词或其缩写形式（2~4个字母）给出一个汇编语句名，例如ADD、SUB、MVRR、MVRD、JR、JMPA、STRX 等。

其中的1~2个字母可能涉及到操作数寻址方式，具体规定如下：

用R代表寄存器寻址，例如ADD R0,R1 语句表示 $R0 \leftarrow R0 + R1$ ；MVRR R0,R1 语句表示把寄存器R1的内容传送到寄存器R0；在R字母两侧加上方括号，代表寄存器间接寻址，例如STRR [R8],R9 语句表示把R9的内容传送到以寄存器R8的内容为地址的内存单元之中；

用D表示立即数寻址，例如MVRD R3,1234语句表示 $R0 \leftarrow$ 立即数1234；

用X表示变址寻址，例如LDRX R1,12[R2]语句表示把变址寄存器R2的内容与变址偏移量12 相加作为内存地址，进行读操作，读出的数据传送的寄存器R1；

用A表示直接地址寻址，例如JMPA 2008语句表示转移到2008 单元之处， STRA [2000], R2 语句表示把R2 的内容写入到地址为2000 的内存单元之中。

二、基本指令汇总表

表1.2.1 基本指令汇总表

指令格式	汇编语句	操作数 个数	CZVS	类型	功能说明
00000000 DRSR	ADD DR, SR	2	****	A 组 指 令	DR←DR+SR
00000001 DRSR	SUB DR, SR	2	****		DR←DR-SR
00000010 DRSR	AND DR, SR	2	• * • *		DR←DR and SR
00000011 DRSR	CMP DR, SR	2	****		DR-SR
00000100 DRSR	XOR DR, SR	2	• * • *		DR←DR xor SR
00000101 DRSR	TEST DR, SR	2	****		DR and SR
00000110 DRSR	OR DR, SR	2	• * • *		DR←DR or SR
00000111 DRSR	MVRR DR, SR	2	****		DR←SR
00001000 DR0000	DEC DR	1	****		DR←DR-1
00001001 DR0000	INC DR	1	****		DR←DR+1
00001010 DR0000	SHL DR	1	* • • •		DR, C←DR*2
00001011 DR0000	SHR DR	1	* • • •		DR, C←DR/2
01000001 OFFSET	JR ADR	1	****		无条件跳转到 ADR
01000100 OFFSET	JRC ADR	1	****		C=1 时跳转到 ADR
01000101 OFFSET	JRNC ADR	1	****		C=0 时跳转到 ADR
01000110 OFFSET	JRZ ADR	1	****		Z=1 时跳转到 ADR
01000111 OFFSET	JRNZ ADR	1	****		Z=0 时跳转到 ADR
10000000 00000000 ADR (16 位)	JMPA ADR	1	****	B 组 指 令	无条件跳到 ADR
10000001 DRSR	LDRR DR, [SR]	2	****		DR←[SR]
10000010 I/O PORT	IN I/O PORT	1	****		RO←[I/O PORT]
10000011 DRSR	STRR [DR], SR	2	****		[DR]←SR
10000100 00000000	PSHF	0	****		FLAG 入栈
10000101 0000SR	PUSH SR	1	****		SR 入栈
10000110 I/O PORT	OUT I/O PORT	1	****		[I/O PORT]←RO
10000111 DR0000	POP DR	1	****		DR←出栈
10001000 DR0000	MVRD DR, DATA	2	****		DR←DATA
10001100 00000000	POPF	0	****		FLAG←出栈
10001111 00000000	RET	0	****		子程序返回
11001110 00000000	CALA ADR	1	****	D 组	调用首地址为 ADR 的子程序

注：① 表中CZVS 一列，* 表示对应的状态位在该指令执行后会被重置；

• 表示对应状态位在该指令执行后不会被修改。

② 运算器芯片中有16 个通用寄存器（累加器）R0~R15，其中：

R4 用作16 位的堆栈指针SP； R5 用作16 位的程序计数器PC；

其余寄存器用作通用寄存器，即多数双操作数指令和单操作数指令中的DR、SR。

三、 扩展指令汇总表

表1.2.2 扩展指令汇总表

指令格式	汇编语句	操作数个数	CZVS	类型	功能说明
00100000 DRSR	ADC DR, SR	2	****	A组指令	DR←DR+SR+C
00100001 DRSR	SBB DR, SR	2	****		DR←DR-SR-C
00101010 DR0000	RCL DR	1	*...		DR带进位C循环左移
00101011 DR0000	RCR DR	1	*...		DR带进位C循环右移
00101100 DR0000	ASR DR	1	*...		DR←DR算术右移
00101101 DR0000	NOT DR	1	****		DR←/DR
01100000 0000SR	JMPR SR	1		跳转到SR指明的地址
01100100 OFFSET	JRS ADR	1		S=1时跳转到ADR
01100101 OFFSET	JRNS ADR	1		S=0时跳转到ADR
01101100 00000000	CLC	0	0...		C=0
01101101 00000000	STC	0	1...		C=1
01101110 00000000	EI	0		开中断, INTE←1
01101111 00000000	DI	0		关中断, INTE←0
11100000 0000SR	CALR SR	1	C组指令	调用SR指明的子程序
11100100 DR0000	LDRA DR, [ADR]	2		DR←[ADR]
11100101 DRSR ADR (16位)	LDRX DR, OFFSET[SR]	2		DR←[DATA+SR]
11100110 DRSR ADR (16位)	STRX DR, OFFSET[SR]	2		[DATA+SR]←SR
11100111 0000SR ADR (16位)	STRA [ADR], SR	1		[ADR]←SR
11101111 00000000	IRET	0	D组	中断返回

注：① 表中CZVS 一列，* 表示对应的状态位在该指令执行后会被重置；
· 表示对应状态位在该指令执行后不会被修改。

② 扩展指令的功能、格式、操作码和操作数地址字段的确定，留给同学自己设计。表中给出的只是可能的一种选择，但同学们一定要认识到，这里的基本指令和扩展指令共同构成教学计算机的完整的指令系统，彼此需要协调，至少不能有冲突。

§1.3 TEC-XP+运算器部件

运算器是计算机硬件系统传统的5大功能部件之一，承担执行运算和暂存运算数据的功能，通常由执行算术逻辑运算功能的ALU线路、暂存参加ALU运算的数据和中间运算结果的通用寄存器组、支持乘除法运算的专用寄存器三部分组成，三个部分之间通过多路选择器线路实现连接，从而构成一个完整的运算器部件。

TEC-XP+教学计算机的运算器部件，主体部分由4片4位长度的位片结构的运算器芯片Am2901组成，每片Am2901可以接收来自内部总线IB的4位输入数据，其4位输出都直接送到地址寄存器AR的不同字段（AR不属于运算器的组成部分，图中用虚线框表示），并且经过支持三态功能的开关门电路送到内部总线IB。还要使用MACH芯片内部的部分电路提供ALU最低位的进位输入信号和最高、最低位的移位输入信号，使用一片GAL20V8实现4位的标志位寄存器FLAG，接收ALU输出的4个标志位信号和来自内存堆栈区的4位数据（用于恢复现场状态信息），FLAG的4位输出可以经过一片带支持三态功能的开关门电路送到内部总线IB，用于保存现场状态信息到堆栈区。教学机运算器部件的组成线路和信息连接关系如图1.3.1所示。

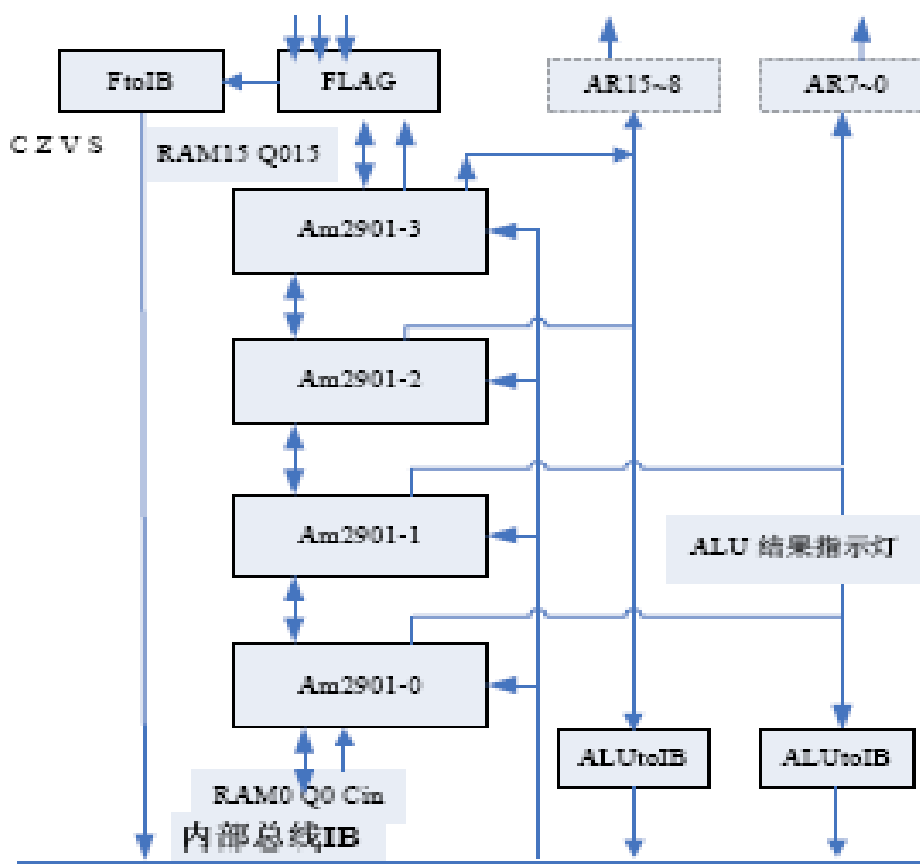


图1.3.1 运算器组成线路和信息连接关系

运算器部件的教学实验，要在教学计算机主板上进行，既可以在脱机方式下完成，也可以在联机方式下完成。

脱机运算器实验方式是指使运算器部件完全脱离与计算机主机其他部件正常的连接关系，在完全孤立出来的运算器上进行的实验。此时，只能通过数据开关拨入参加运算的数据，通过微型开关提供操作运算器运行所必需的控制信号，通过信号指示灯观察运算结果，操作简单，实验结果清晰易

理解。图1.3.2为16位运算器脱机实验的环境，在计原16系统中，运算器最低位的进位输入信号Cin和左右移位输入信号RAM0、Q0、RAM15、Q15是由MACH芯片内部的SHIFT线路提供的，图左侧的长方形部分是MACH内部的线路。16个开关拨入的数据经开关门电路送到内部总线，内部总线与运算器的输入端D15~D0已经连接。23位的微型开关的不同组合完成不同的控制，其各编码对应的控制功能给出在表1.3.1。

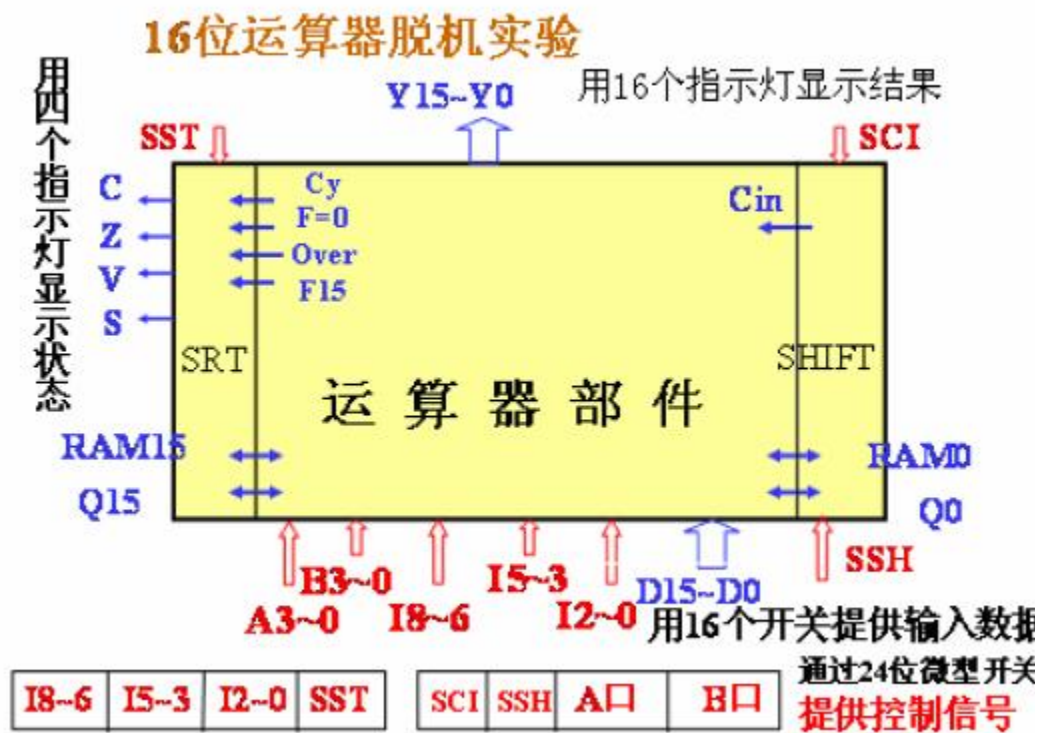


图1.3.2 脱机运算器实验的环境

联机实验方式是指在运算器部件与计算机主机保持正常连接关系，教学计算机可以正常执行指令的情况下进行的以运算器为重点的教学实验。此时，可以通过指令提供参加运算的数据，通过控制器提供操作运算器运行所必需的控制信号，通过信号指示灯或者通过运行监控程序观察运算结果，操作略显复杂，涉及到目前尚未讲解到的如何让控制器提供运算器实验所要求的控制信号的办法，有一定难度，当然完成实验后的收获也会更大，提前接触到控制器部件的一些内容。

表1.3.1 微型开关各编码对应的控制功能表

	I8~6			I5~3	I2~0	
	REG	Q	Y	功能	R	S
000		$F \rightarrow Q$	F	$R+S$	A	Q
001			F	$S-R$	A	B
010	$F \rightarrow B$		A	$R-S$	0	Q
011	$F \rightarrow B$		F	$R \vee S$	0	B
100	$F/2 \rightarrow B$	$Q/2 \rightarrow Q$	F	$R \wedge S$	0	A
101	$F/2 \rightarrow B$		F	$\neg R \wedge S$	D	A
110	$2F \rightarrow B$	$2Q \rightarrow Q$	F	$R \oplus S$	D	Q
111	$2F \rightarrow B$		F	$\neg(R \oplus S)$	D	0

SST	C	Z	V	S		
000	C	Z	V	S	SH SCI	Cin / Shift
001	CY	F=0	OVR	FIS	000	Cin=0
010	内部总线				001	Cin=1
011	0	Z	V	S	010	Cin=C
100	1	Z	V	S		
101	RAM0	Z	V	S	100	逻辑移位
110	RAM15	Z	V	S	101	循环移位
111	Q0	Z	V	S		

§1.4 TEC-XP+内存储器部件

内存储器是计算机中存放正在运行中的程序和相关数据的部件。在教学计算机存储器部件设计中，出于简化和容易实现的目的，选用静态存储器芯片实现内存存储器的存储体，包括唯读存储区（ROM，存放监控程序等）和随读写存储区（RAM）两部分，ROM存储区选用4片长度8位、容量8KB的58C65芯片实现，RAM存储区选用2片长度8位、容量2KB的6116芯片实现，每2个8位的芯片合成一组用于组成16位长度的内存字，6个芯片被分成3组，其地址空间分配关系是：0-1777h用于第一组ROM，固化监控程序，2000-2777h用于RAM，保存用户程序 and 用户数据，其高端的一些单元作为监控程序的数据区，第二组ROM的地址范围可以由用户选择，主要用于完成扩展内存容量（存储器的字、位扩展）的教学实验。内存储器和串行接口线路的组成如图1.4.1所示。

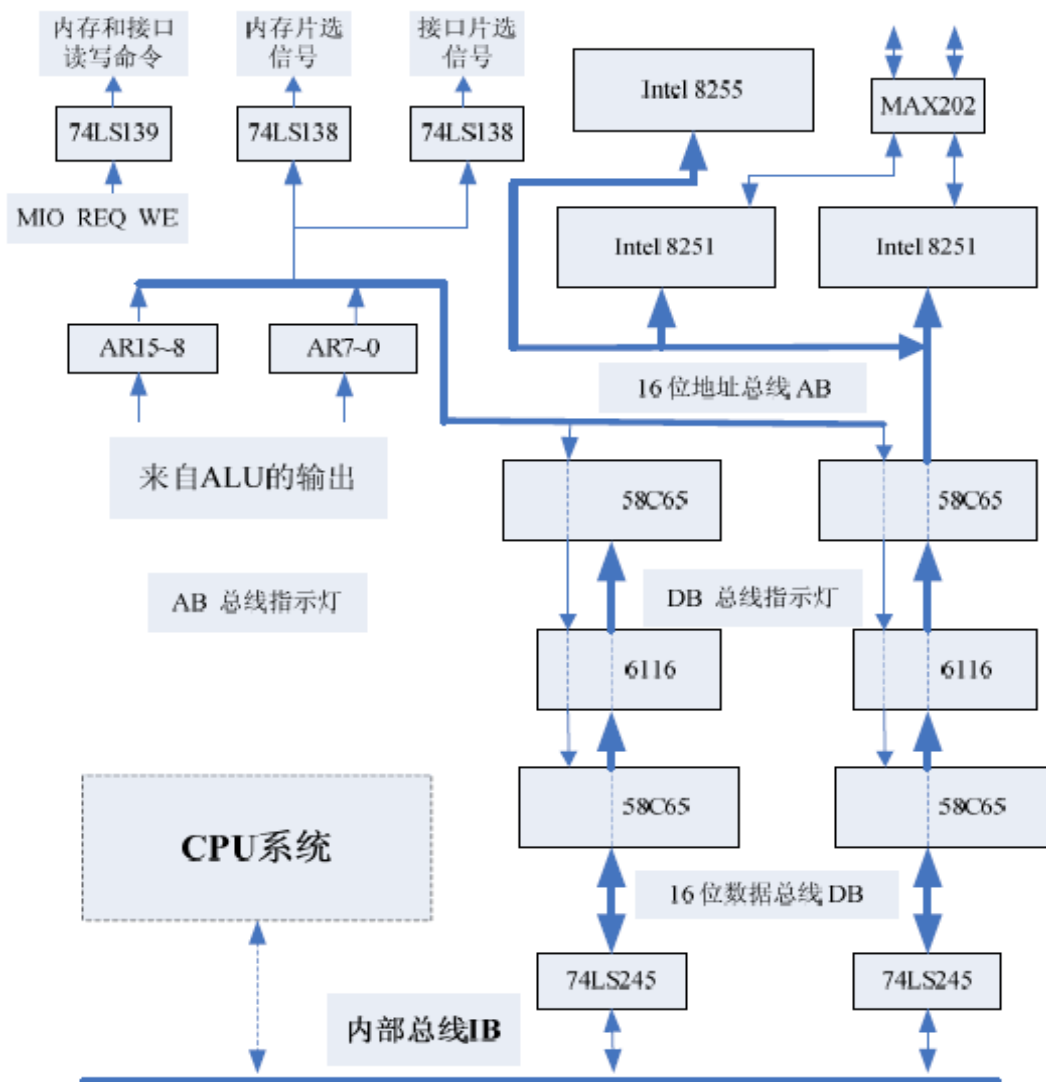


图1.4.1 内存储器和串行接口电路

地址总线的低13位送到ROM芯片的地址线引脚（RAM芯片只使用地址总线的低11位），用于选择芯片内的一个存储字。用于实现存储字的高位字节的3个芯片的数据线引脚、实现低位字节的3个芯片的数据线引脚分别连接在一起接到数据总线的高、低位字节，是实现存储器数据读写的信息通路。数据总线要通过一个双向三态门电路与CPU一侧的内部总线IB相连接，已完成存储器、接口电路和CPU之间的数据通讯，如图1.4.1中的虚线部分所示。

这里用到3个译码器电路，其中一片74LS138译码器芯片接收地址总线最高的3位地址信息，当需要内存工作时，由这片译码器产生内存芯片的8个片选信号，以选择哪一个空间范围的内存区可以读写。另外一片74LS138译码器芯片接收地址总线低位字节的最高4位地址信息（最高一位恒定为1），当需要接口电路工作时，由这片译码器产生接口芯片的8个片选信号，以选择哪一个接口电路可以读写。一片74LS139双二-四译码器芯片接收控制器送来的3位控制信号MIO（有无内存或者接口电路要读写）、REQ（是内存还是接口电路要读写）、WE（是读操作还是写操作），当这3位控制信号的组合为1××、000、001、010、011时，译码器将产生读内存操作、写内存操作、读接口操作、写接口操作、内存和接口芯片都无读写操作的控制信号。

在这里还要说明如下两个问题。第一，要扩展8K字的存储空间，需要使用2片（每一片有8KB容量，

即芯片内由8192个单元、每个单元由8个二进制位组成）存储器芯片实现。第二，当存储器选用58C65ROM芯片时，它属于电可擦出的EPROM器件，可以通过专用的编程器软件和设备向芯片的写入相应的内容，这是正常的操作方式。也可以通过写内存的指令向芯片的指定单元写入16位的数据，只是每一次的这种写操作需要占用长得多写入时间，例如几百个微秒，可以通过运行完成等待功能的子程序来加以保证。对58C65 ROM 芯片执行读操作时，需要保证正确的片选信号（/CE）为低点平，使能控制信号（/OE）为低电平，读写命令信号（/WE）为高电平，读58C65 ROM 芯片的读出时间与读RAM 芯片的读出时间相同，无特殊要求；对58C65 ROM 芯片执行写操作时，需要保证正确的片选信号（/CE）为低电平，使能控制信号（/OE）为高电平，读写命令信号（/WE）为低电平，写58C65 ROM 芯片的维持时间要比写RAM 芯片的操作时间长得多。为了防止对58C65 ROM芯片执行误写操作，可通过把芯片的使能控制引脚（/OE）接地来保证，或者确保读写命令信号（/WE）恒为高电平。

串行接口芯片的8位数据线引脚连接到数据总线DB的低位字节，它与CPU之间每次交换8位信息，属于并行操作关系。串行接口芯片和设备之间的连接，是通过连接到设备端的另外一个串行接口芯片完成的，在两端的接口芯片之间以串行方式实现通讯，即遵从一定的通讯协议，对8位的数据采用逐位传送的方案处理，并把信号的电平从TTL电路的0~4V左右提高到正负12V或者0~12V左右，以增强信号传送过程中的抗干扰能力，图1.4.1中的MAX202芯片就是采用倍压方案完成电平转换功能的，有了这个芯片就可以不再使用直流+12V和-12V两路电源了。

§1.5 TEC-XP+控制器部件

控制器部件是计算机系统传统的5大功能部件之一，其作用是依据指令内容和指令的执行步骤信号等向计算机的各个部件提供它们每一个步骤协同运行所需要的控制信号。教学计算机系统同时实现了微程序和硬连线方案的两种控制器，其总体组成如图1.5.1所示。从两种控制器的组成来看，都包括：程序计数器PC，是选用在运算器的通用寄存器组中的R5实现的，图中用虚线表示；指令寄存器IR，是选用2片8位的寄存器电路实现的，还用到一片传送IR低位字节内容到内部总线IB的开关门电路；最重要的是一片有130多个输入输出管脚、由一万个门电路组成的CPLD器件MACH芯片，其内部的线路组成和实现的功能都比较复杂，微程序控制器还包括确定微指令执行次序的一片Am2910 芯片。

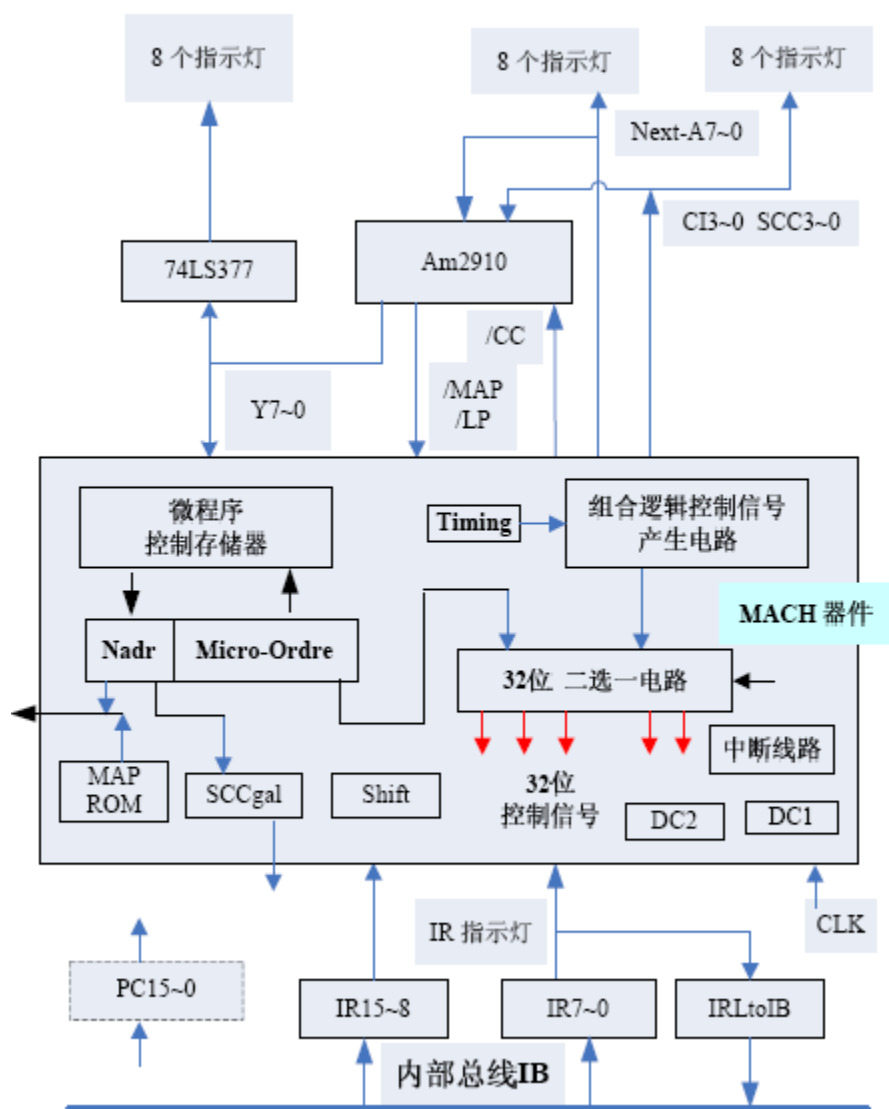


图1.5.1 教学机计算机控制器的总体组成

程序计数器PC可以在运算器内部完成加1运算功能，产生下一条相邻指令的地址；或者完成加一个相对转移的偏移量，产生相对转移指令的转移地址；或者直接接收一个16位的指令地址，用在子程序调用或者跳转指令中。

指令寄存器IR接收从内存储器读出并传送到内部总线IB的指令内容，其全部16位输出送MACH芯片的输入引脚，其低8位内容还经一个有三态功能的开关门送到内部总线IB。

Am2910芯片用在微程序控制器中，用于以多种方式向控制存储器提供下一条微指令的地址，控制微指令的执行次序，解决的是指令执行步骤的衔接问题。

两种控制器的主要功能线路都集中到一片高集成度CPLD器件（MACH芯片）中实现，在完成扩展指令的教学实验中，可以不对MACH芯片之外的电路作任何变动，全部操作都集中到修改描述MACH芯片功能的ABEL或VHDL语言的程序代码，执行编译和下载操作，以及运行调试这样几步工作。这是本教学计算机系统的一项显著特色，对降低学生的学习难度，简化修改与扩展控制器功能的操作，改善教学实验效果将起到重要作用。

从图1.5.1可以看到，MACH 器件内部包含有微程序控制器中的控制存储器、微指令寄存器、映射指令操作码为微指令地址的MAPROM、产生微指令是否转移信号CC的SCCgal、实现微指令字中的下

条微指令地址和MAPROM输出的下条微指令地址二者选其一的功能几个部分的功能线路；包含有硬连线控制器的节拍发生器Timing、控制信号产生部件两部分线路；此外还实现了把两种控制器各自提供的32位控制信号进行二者选其一的功能。

与早期产品比较，还把用于中断实验的电路、对控制器控制信号译码的DC2和DC1电路、为运算器提供最低位进位输入信号、左右移位时提供给最高最低位的移位输入信号的电路也放到了MACH芯片之内实现，以尽量减少在主板上用到的中小规模电路数量。

MACH器件内部实现的电路，都是通过设计描述其功能的逻辑表达式体现出来的，这些逻辑表达式的分段清晰，实现的功能彼此独立，可读性比较强，设计与变更设计容易，不必过多处理功能线路之间的布线连接，可以保证更高的实验效率。这一设计结果也为更好地支持8位字长的教学计算机、完成8位字长计算机的教学实验带来极大方便。设计描述MACH芯片所实现功能的程序源代码是设计控制器的主要工作，下面将详细讲解这些内容。

在图1.5.2中给出了MACH器件的管脚编号和对应的信号名称。这些I/O管脚与外部其他线路的连接关系已经通过印制电路板的布线固定下来，不能试图去变更它。MACH的I/O管脚编号与信号名的对应关系，在程序的说明段中指定的，在程序的表达式段中，必须通过信号名称来表示和使用功能电路的输入输出信号。

MACH芯片被焊接在一块小的电路板上，使用的电源是3.3V(不是5V)，是5V电源经过一个二极管降压后得到的。输入输出信号通过4个40针的接插头与大板上的4个40孔的接插座实现连接，通常情况下，这块小板不必拔下来。如果要插拔这块小电路板，请一定要仔细，方向要正确，小板上的文字方向不能朝左或者朝右，也不能倒过来；位置要对准，每一个插针与对应一个插孔对正，不能错位；插拔时尽量保持小板保持水平状态，不能有太大的倾斜角度；插拔时必须用力适度，力量太小无法完成插拔，也不能使用暴力，以防对电路板或者接插元件造成损坏。

有多个描述MACH器件所实现的线路与运行功能的ABEL语言的程序代码文件，不同的控制器方案 and 不同的指令集合可以随意组合方案，对16位的教学计算系统，有用硬连线方案实现的基本指令集合与用微程序方案实现的全指令集合的组合方案，有用硬连线方案实现的全指令集合与用微程序方案实现的基本指令集合的组合方案，有用两种控制器实现的基本指令集合的组合方案，当然也有只实现一种控制器的方案。当然还有实现8位字长的教学计算机系统的控制器的源代码文件，下载不同的文件就能够得到不同的教学机系统。

完成控制器的教学实验，需要很好地掌握指令格式、指令编码、寻址方式和指令功能等内容，需要熟悉教学计算机的总体组成和各个部件的运行原理。为此，适当地花费一点时间学懂已有指令执行步骤的划分方案，各部件要求使用的控制信号。对微程序控制器还要懂得微指令在控制存储器中的地址分配和形成微指令地址的方案；对硬连线控制器还要懂得节拍发生器的设计和运行原理，控制信号产生部件的设计和提供控制信号的方案等内容。一句话，就是需要初步懂得控制器的组成和运行原理的知识，教学计算机控制器线路的实际组成。教学实验的目的，是在初步理解的基础上，通过自己动手，在已有指令的基础上再扩展若干条指令，把控制器设计的整个过程亲自经历一遍，把尚未完全理解的内容设法理解正确，对认识尚肤浅的、甚至于拿不太准的知识掌握得更深刻，在学习书本知识的同时，有意识地通过实践来获取知识是人类成长进步的重要环节。

新的大MACH I/O PIN

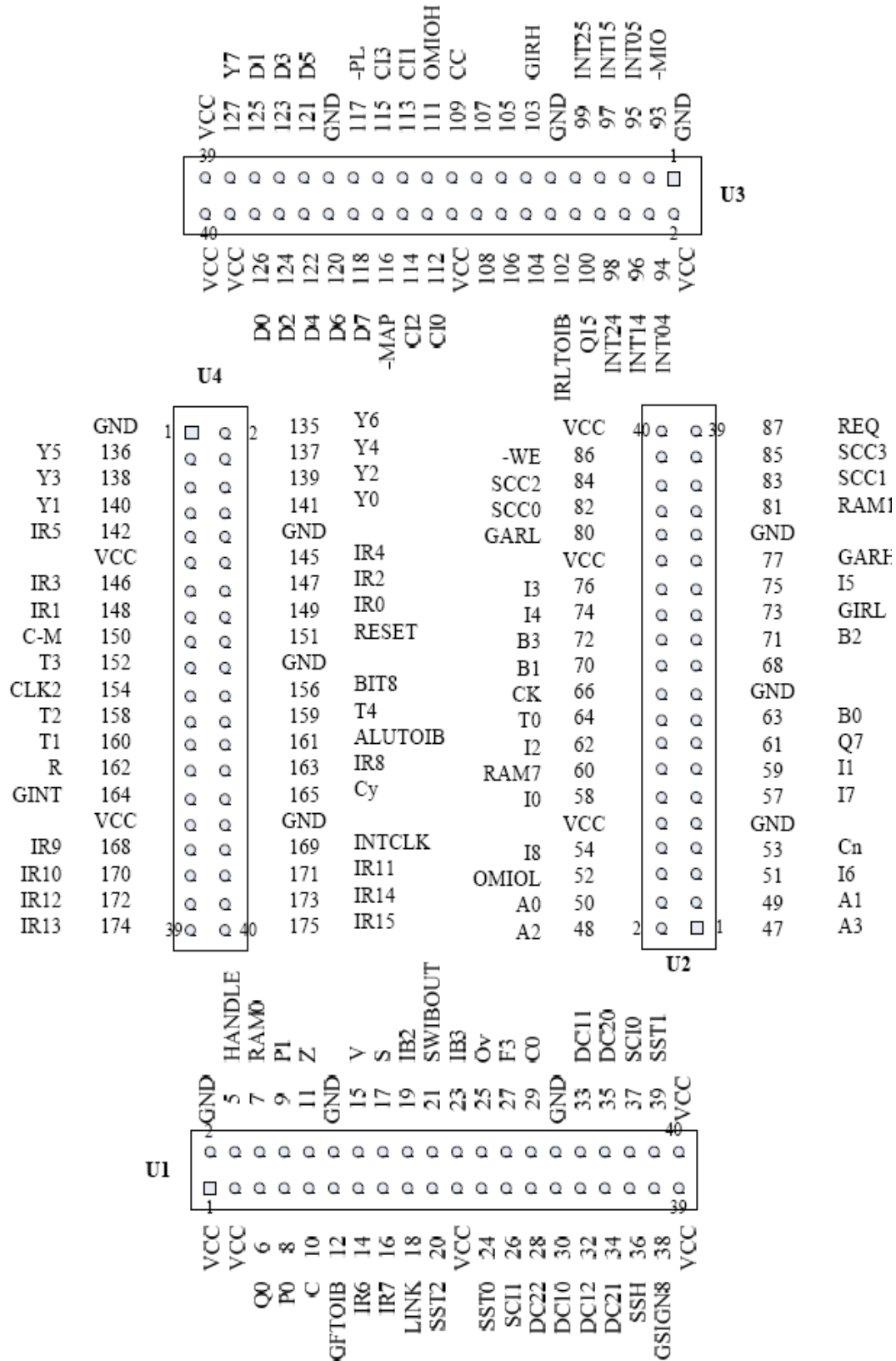


图 1.5.2 MACH 器件的管脚位置、编号和对应的信号名称

§1.6 TEC-XP+输入输出及中断

一、串行I/O接口

串行接口是计算机主机和某些设备之间实现通信，硬件造价比较廉价、标准化程度比较高的一种输入输出接口线路，缺点是通信的速度比较低。从在程序中使用串行接口芯片的角度看，接口芯片内有用户可以访问的4个寄存器，分别是接收CPU送来数据的输出数据缓冲寄存器，向CPU提供数据的输入数据缓冲寄存器，接收CPU发来的控制命令的控制寄存器，向CPU提供接口运行状态的状态寄存器，必须有办法区分这4个寄存器。接口芯片中还有执行数据串行和并行转换的电路，接口识别（片选）电路等。

在教学计算机系统中，配置了两路串行接口，其线路组成和信息传送路径已在图1.1.6中给出。串行接口芯片的8位数据线引脚连接到数据总线DB的低位字节，它与CPU之间每次交换8位信息，属于并行操作关系。串行接口芯片和设备之间的连接，是通过连接到设备端的另外一个串行接口芯片完成的，在两端的接口芯片之间以串行方式实现通讯，即遵从一定的通讯协议，对8位的数据采用逐位传送的方案处理，并把信号的电平从TTL电路的0~4V左右提高到正负12V或者0~12V左右，以增强信号传送过程中的抗干扰能力，通常可以选用MAX202芯片使用倍压技术完成电平转换功能，就可以不再使用直流+12V和-12V两路电源。

串行接口用于执行数据的输入输出操作。一次输入或输出操作通常需要用两个操作步骤完成，第一步是为接口芯片提供入出端口地址，即把指令寄存器低位字节的内容（8位的IO端口地址）经过内部总线和运算器部件写进地址寄存器AR，第二步是执行输入或者输出操作，若执行输入指令IN，则应从接口芯片读出一个8位的数据并经过数据总线DB和内部总线IB写进寄存器堆中的R0寄存器，若执行输出指令OUT，则需要把寄存器堆中的R0寄存器的内容经过内部总线IB和数据总线DB写入接口芯片。接口芯片与输入输出设备之间的数据传送过程无需另外管理，会自动完成。

教学计算机使用8位的IO端口地址，安排在IN和OUT指令的低位字节，指令的高8位用作指令操作码，16位的指令编码全部占满，已经不能再指定要使用的通用寄存器，最终决定对IN和OUT指令默认使用运算器中的R0完成输入输出操作。IO地址端口的高4位（最高一位的值一定为1）用于通过译码电路产生接口芯片的8个片选信号，低4位用于选择一个芯片内最多16个寄存器。教学计算机中，只为每个串行口芯片分配了两个地址，第一路串行接口的端口地址为16进制的80/81，第二路串行接口的端口地址可以由用户从90/91-F0/F1这8对中选择，把译码器的一个输出连接到接口芯片的片选信号引脚。两个端口地址如何能够选择接口芯片内的4个寄存器呢？请注意，4个寄存器中的两个只用于输入，仅对IN指令有用，另外两个只用于输出，仅对OUT指令有用。2个端口地址和2条输入输出指令有如下4种组合，分别实现如下4项功能：

IN 80 完成从接口芯片输入数据缓冲器读出8位数据并传送到R0寄存器低位字节；

OUT 80 完成把R0寄存器低位字节的8位数据写入到接口芯片的输出数据缓冲器；

IN 81 完成从接口芯片状态寄存器读出8位接口状态信息并送R0寄存器低位字节；

OUT 81 完成把R0寄存器低位字节的8位命令信息写入到接口芯片的命令寄存器。

可以看到，偶数地址用于输入输出数据，奇数地址用于输入输出状态或命令信息。

在教学计算机系统中的两路串行接口，其中的一路全部连接线已经连接好，运行之前的初始化操作也由监控程序完成，已经处于正常的运行状态，作为教学机与PC机仿真终端相连接的接口使用。另一路串行接口更多的是用于扩展的教学实验目的，它的一部分连接线已经连接好，还有少数连接线需要由实验者通过跳接线进行连通，并且在自己的程序中执行对此串行口的初始化操作，这之后才能使用这一路串行接口执行输入输出操作。这样做的目的，是为了加深同学对串行口的线路连接、初始化、执行输入输出操作过程的理解程度，否则虽然在程序中通过IN和OUT指令完成了串行接口的输入输出操作的实验，但是对串行接口线路的认识并不全面，把应该由同学执行的一些操作保留给同学完成是必要的。

二、中断

在计算机系统中，中断处理是非常重要的一项功能。在教学计算机系统中，也提供了中断处理的能力。用3个按钮开关作为中断源，在MACH器件内部实现了保存中断请求、中断优先级编码、中断响应前的优先级比较、中断响应与返回过程中的中断优先级的切换和现场信息恢复等全部硬件实现的功能。还设置了处理中断向量的硬件支持。这样就可以完成有3级中断嵌套的教学实验。教学计算机中提供了完成中断处理的全部硬件线路和实现方案，有利于帮助同学准确深入地掌握完成中断处理的原理和过程。

§1.7 TEC-XP（FPGA）计算机硬件系统

随着半导体集成电路技术的迅猛发展，为进行大规模系统的设计和实现带来了新的方法和手段。以HDL（Hardware Description Language, HDL）语言表达设计意图、FPGA作为硬件载体、计算机为设计开发工具、EDA软件为开发环境的现代电子设计方法日趋完善。为了让学生了解科技发展的前沿技术，感受科学技术的成果，为了顺应最新的计算机设计趋势和教学实验的更高需求，在TEC-XP+教学计算机上设置了超大规模集成电路FPGA器件，以满足学生对专用集成电路(ASIC)设计的需求。

一、TEC-XP（FPGA）教学计算机系统组成概述

TEC-XP（FPGA）是TEC-XP+的一个重要组成部分。它和TEC-XP16系统相对独立，软件互相兼容。TEC-XP（FPGA）和TEC-XP16系统各自的CPU通过总线连接TEC-XP+主板上的内存、接口等构筑了双CPU的EC-XP+系统。

TEC-XP（FPGA）教学机系统的主要技术指标是：

1. 机器字长16位（也可设计成8位字长的另外一个新的系统），即运算器、主存、数据总线、地址总线、指令等都是16位。
2. 完整的指令系统被划分为基本指令和扩展指令两部分，支持多种基本寻址方式。其中的基本指令已经实现，用于设计监控程序和用户的常规汇编程序，保留的多条扩展指令供实验者自己实现。
3. 主存最大寻址空间是18K字(16位)，基本容量为8K字的ROM和2K字的RAM存储区域。另外的8K字用于完成存储器容量扩展的教学实验。FPGA芯片和存储器芯片之间可以通过分开的地址总线和分开的数据总线实现连接，这在实现分开的指令存储器和数据存储器的方案中是必要的。
4. 运算器是参照Am2901芯片的组成和功能来设计的，ALU实现8种算术与逻辑运算功能，

内部包括 16 个双端口读出、单端口写入的通用寄存器，和一个能自行移位的乘商寄存器。设置 C（进位）、Z（结果为 0）、V（溢出）和 S（符号位）四个状态标志位。

5. 控制器采用硬连线控制器方案实现，也可修改成微程序控制器。实验人员可方便地修改已有设计，或加进若干条自己设计与实现的新指令，新老指令同时运行。
 6. 主机上安装有二路 INTEL8251 串行接口，可直接接计算机终端，或接入一台 PC 机作为自己的仿真终端。选用了 MAX202 倍压线路，以避免使用+12V 和-12V 电源。
 7. 两路的串行接口的接插座安放在机箱后侧板以方便接线插拔和机箱盖的打开关闭。
 8. 在主板上设置有一些拨数的开关和微型开关、按键和指示灯，支持最低层的手工操作方式的输入/输出和机器调试。
 9. 实验机硬件系统，全部功能部件分区域划分在大一些的水平放置的一块印制电路板的不同区域，所有器件都用插座插接在印制板上，便于更换器件。
 10. 实验计算机使用单一的 5V、最大电流 3A 的直流模块电源，所耗电流在 1.5~2.5A 之间。电源模块安装在水平电路板右上角位置，交流 220V 通过电源接线插到机箱后侧板，经保险丝、开关连接到电路板上，开关安放在机箱右侧靠后位置，方便操作且比较安全。
 11. 板上安装了很多发光二极管指示灯，用于显示重要的数据或控制信号的状态。
 12. 机箱和电路板之间的全部接线都经过接插座，便于整机的生产、调试和维修。
- 此外还设置了辅助电路和扩展电路两个辅助部分，各个部分被划分在电路板的不同区域。

TEC-XP（FPGA）系统的总体组成如图 1.7.1 所示。由图 1.7.1 可以看到 TEC-XP（FPGA）教学计算机系统也是一个完整的计算机系统，由以下几个基本部分组成：运算器部件、控制器部件、内存存储器系统和串行接口线路。

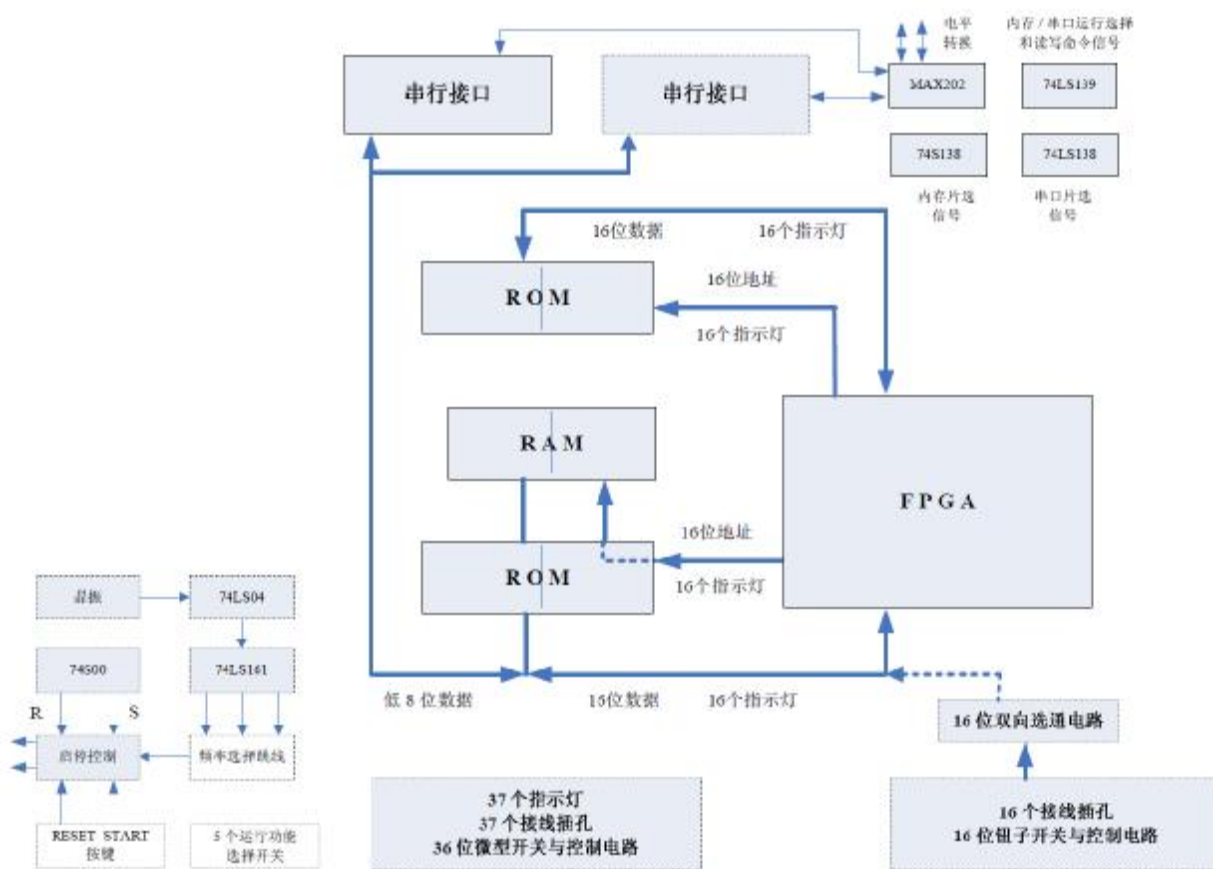


图1.7.1 TH-union (FPGA) 实验机系统逻辑框图

TEC-XP (FPGA) 中 CPU 系统的实现, 选择了 xilinx 公司的 SPARTAN—II 系列的芯片 (型号是 XC2S200)。该器件容量为 20 万门, 内部有 2352 个 CLB, 14 个 4Kb 的 RAM 块, 208 脚 PQFP 封装形式, 支持在系统编程 (in-system programmable), 实现了 TEC-XP 16 系统中 CPU 的全部功能。串行接口接通用 PC 机实现的仿真终端, 把监控程序保存到内存的 ROM 存储区, 则得到一台硬软件组成相对完整的、真正能够正常运行的教学计算机系统。

二、FPGA 芯片的外特性和内部结构

现场可编程门阵列 (FPGA) 是近十多年来出现并开始被广泛应用的大规模集成电路器件。尽管它与 GAL20V8 和 MACH 类型的 PLD 同属现场可编程器件, 但是其内部结构却完全不同, 也有着不同的特性和性能。GAL20V8 和 MACH 都是采用“与—或”两级逻辑阵列加上输出逻辑单元的内部结构, 而 FPGA 的内部结构则是采用许多个独立的可编程的逻辑模块 CLB (configurable logic block)、输入输出模块 IOB (I/O block) 和互连资源 (interconnect resource) 3 部分组成。

§1.8 教学机的 PC 机仿真终端程序

PCEC 是用 IBM-PC 机的汇编语言编制而成的,它可把 IBM-PC 作为教学机的终端完成数据的输入及显示,更重要的是它能实现 PC 机与教学机之间的文件传送,支持交叉汇编程序的使用。PCEC 还具有拷贝屏幕功能。

1. PCEC 的运行过程

在 IBM-PC 机上打入程序名 PCEC16 (8 位机为 PCEC8) 并回车,接下来按程序的提示分别选择连接教学机的 PC 机仿真终端的串行接口号(1 或 2)和通讯参数。

程序中默认的通讯参数为:波特率 9600, 8 位字长, 无奇偶校验和 1 位停止位, 没有特殊要求时一般不需修改这些参数。IBM-PC 机作为教学机的终端, 可以执行教学机监控程序的各种命令。

2. 文件传送过程

PC 机作为教学机的控制台之后, 按 F10 键, 显示菜单:

0--Return to CRT Monitor

1--Send a file to TEC-2

2--Receieve a file from TEC-2

3--Return to IBM-PC MSDOS

选择 0 不执行任何操作, 直接返回教学机的监控状态;

选择 3 退出通讯程序, 返回 IBM-PC 的操作系统;

选择 1 将执行往教学机发送文件, 即把指定的文件从磁盘中取出, 通过串行口传送给教学机并存储于该机的主存中。**要求:** 被发送的文件一定是经交叉汇编程序 ASEC8 (16 位机为 ASEC16) 汇编后生成的类型为 .COD 文件, 该文件包含有由 ORG 定义的程序或数据的首地址和文件长度, 传送时就此地址作为目的地址, 目标地址指向的主存应为 RAM 区, 文件长度控制传送过程的结束。

选择 2 将执行从教学机接收文件, 并作为文件存于 PC 的磁盘中。选择 2 之后, 首先打入 IBM-PC 中将用的文件名, 就可以把教学机主存中某一区域中的数据传送给 PC 机。**要求:** 在执行文件传送之前, 必须先用监控命令 E 在教学机主存 27CFH、27CEH 单元输入被传数据在教学机内存区的起始地址的高 8 位和低 8 位, 在 27CDH、27CCH 单元输入被传数据字节长度的高 8 位和低 8 位。

3. PCEC 的拷贝屏幕功能

按 SHIFT/F10 之后, 屏幕上显示的所有信息均存储于 SCR.TMP 文件中, 直到再次按 SHIFT/F10 或退出 PCEC。可用于打印屏幕内容。

第二章 TEC-XP+实验系统实验内容

实验一 基础汇编语言程序设计

实验目的：

1. 学习和了解 TEC-XP+教学实验系统监控命令的用法；
2. 学习和了解 TEC-XP+教学实验系统的指令系统；
3. 学习简单的 TEC-XP+教学实验系统汇编程序设计。

实验内容：

1. 学习联机使用 TEC-XP+教学实验系统和仿真终端软件 PCEC；
2. 学习使用 WINDOWS 界面的串口通讯软件。
3. 使用监控程序的 R 命令显示/修改寄存器内容、D 命令显示存储器内容、E 命令修改存储器内容；
4. 使用 A 命令写一小段汇编程序，U 命令反汇编刚输入的程序，用 G 命令连续运行该程序，用 T、P 命令单步运行并观察程序单步执行情况；

实验要求

在使用该教学机之前，应先熟悉教学机的各个组成部分，及其使用方法。

实验步骤

一. 实验具体操作步骤：

- (1) 准备一台串口工作良好的 PC 机；
- (2) 将 TEC-XP 放在实验台上，打开实验箱的盖子，确定电源处于断开状态；
- (3) 将黑色的电源线一端接 220V 交流电源，另一端插在 TEC-XP 实验箱的电源插座里；
- (4) 取出通讯线，将通讯线的 9 芯插头接在 TEC-XP 实验箱上的串口“COM1”或“COM2”上，另一端接到 PC 机的串口上；
- (5) 将 TEC-XP 实验系统左下方的六个黑色的控制机器运行状态的开关置于正确的位置，在找个实验中开关应置为 001100（连续、内存读指令、组合逻辑、联机、16 位、MACH），控制开关的功能在开关上、下方有标识；开关拨向上方表示“1”，拨向下方表示“0”，“X”表示任意，其它实验相同；
- (6) 打开电源，船形开关和 5V 电源指示灯亮。
- (7) 在 PC 机上运行 PCEC16.EXE 文件，根据连接的 PC 机的串口设置所用 PC 机的串口为“1”或“2”，其它的设置一般不用改动，直接回车即可。（具体步骤附后）
- (8) 按一下“RESET”按键，再按一下“START”按键，主机上显示：

TEC-2000 CRT MONITOR

Version 1.0 April 2001

Computer Architectur Lab., Tsinghua University

Programmed by He Jia

>

二、实验注意事项：

1. 连接电源线和通讯线前 TEC-XP+实验系统的电源开关一定要处于断开状态，否则可能会对 TEC-XP+实验系统上的芯片和 PC 机的串口造成损害；

2. 六个黑色控制开关的功能示意图如下：

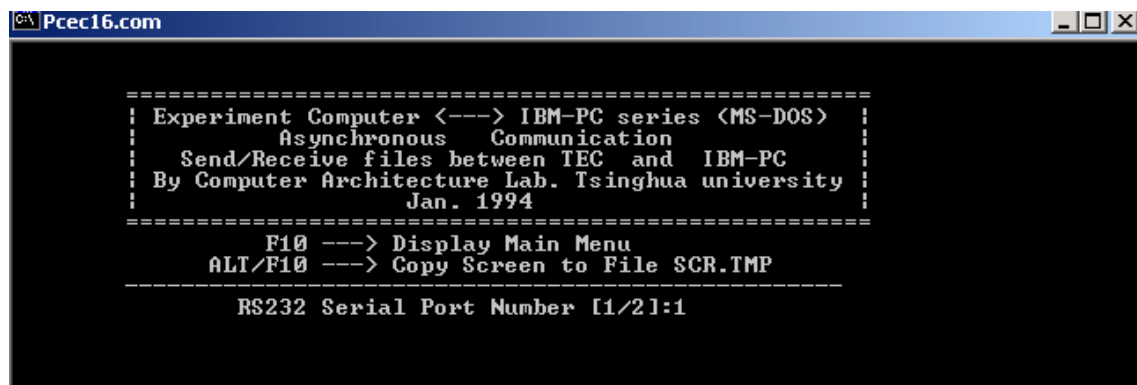
单步	手动置指令	组合逻辑	联机	8 位	FPGA	
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	上面
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	下面
连续	从内存读指令	微程序	脱机	16 位	MACH	

3. 几种常用的工作方式（开关拨到上方表示为 1，拨到下方为 0；）

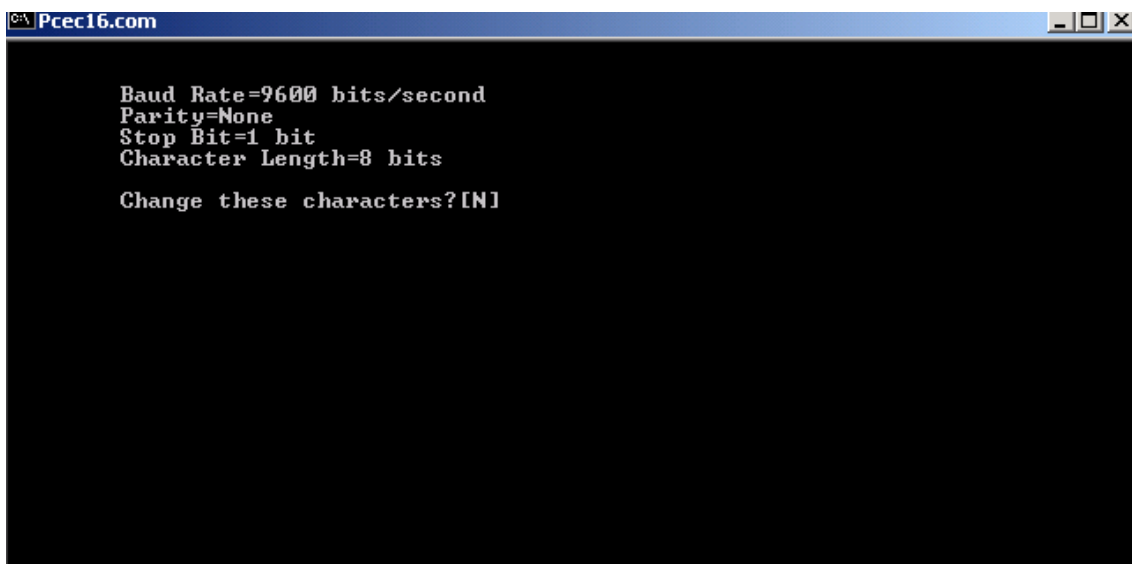
工作方式	六个拨动开关
连续运行程序、组合逻辑控制器、联机、16 位机、MACH	001100
连续运行程序、微程序控制器、联机、16 位机、MACH	000100
单步、手动置指令、组合逻辑控制器、联机、16 位机、MACH	111100
单步、手动置指令、微程序控制器、联机、16 位机、MACH	110100
16 位机、脱机运算器实验、MACH	1XX000
连续运行程序、联机、16 位机、FPGA	00X101

三、仿真终端软件的操作步骤：

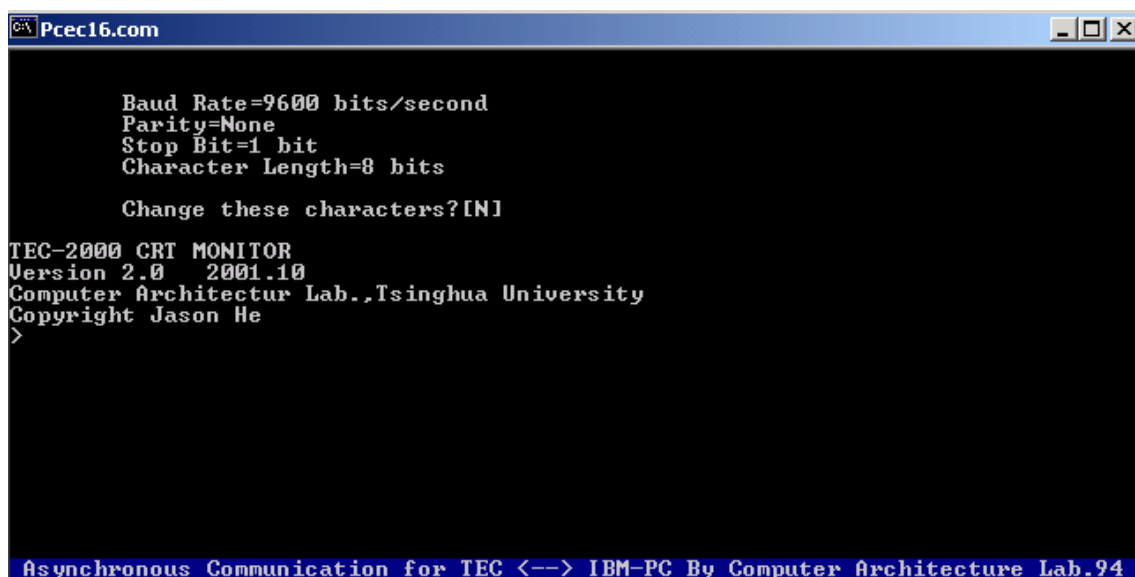
1. 在 PC 机上建一个文件夹 TEC-XP+；
2. 取出配套的用户盘，将应用程序 PCEC16 拷贝到用户机器硬盘上该文件夹里；
3. 双击 PCEC16 图标，出现如图所示的界面：



4. 系统默认选择串口 1，用户可根据实际情况选择串口 1 或是串口 2（这里的串口指的是和 TEC-XP+教学实验系统相连的 PC 机的串口），按回车后出现如图界面：



5. 图中是系统设定的一些传输参数，建议用户不要改动，直接回车。按一下“RESET”按钮放开后再按一下“START”按钮，出现界面如图所示：



6. 此时表明 TEC-XP+机器联机通讯正常。

四. 软件操作注意事项：

1. 用户在选择串口时，选定的是 PC 机的串口 1 或串口 2，而不是 TEC-XP+实验系统上的串口。即选定的是用户实验时通讯线接的 PC 机的端口；
2. 如果在运行到第五步时没有出现应该出现的界面，用户需要检查是不是打开了两个软件界面，若是，关掉其中一个再试；
3. 有时若 TEC-XP+实验系统不通讯，也可以重新启动软件或是重新启动 PC 再试；
4. 在打开该应用软件时，其它的同样会用到该串口的应用软件要先关掉。

五. 联机通讯失败自检：

如果上述的硬件和软件的操作都正确，联机却依旧失败，用户可以进行如下测试：

1. 测试 PC 机的串口是否能正常工作，或是换一台 PC 或换同一台 PC 的另一个串口再试，在换串口时要将 TEC-XP+实验系统断电，换完后重新启动实验系统和软件；
2. 检查机器上的元器件插接是否正确（建议用户对照能够正常通讯的实验系统进行详细检查），有没有被学生动过，尤其是扩展内存和扩展 I/O 接口时，芯片方向是否插对，片选信号有没有连接；
3. 检查相应的短路子是否连接正确；
4. 建议教师预留一台运行正常的 TEC-XP+实验系统备用，机器出问题后可以对照检查。

六. 实验示例：

1. 用 R 命令查看寄存器内容或修改寄存器的内容

1) 在命令行提示符状态下输入：

R✓ ; 显示寄存器的内容

注：寄存器的内容在运行程序或执行命令后会发生变化。

2) 在命令行提示符状态下输入：

R R0✓ ; 修改寄存器 R0 的内容，被修改的寄存器与所赋值之间可以无空格,也可有一个或数个空格

主机显示：

寄存器原值：_

在该提示符下输入新的值 0036

再用 R 命令显示寄存器内容，则 R0 的内容变为 0036。

2. 用 D 命令显示存储器内容

在命令行提示符状态下输入：

D 2000✓

会显示从 2000H 地址开始的连续 128 个字的内容；

连续使用不带参数的 D 命令，起始地址会自动加 128（即 80H）。

3. 用 E 命令修改存储器内容

在命令行提示符状态下输入：

E 2000✓

屏幕显示：

2000 地址单元的原有内容:光标闪烁等待输入

输入 0000

依次改变地址单元 2001~2005 的内容为:1111 2222 3333 4444 5555

注意：用 E 命令连续修改内存单元的值时，每修改完一个，按一下空格键，系统会自动给出下一个内存单元的值，等待修改；按回车键则退出 E 命令。

4. 用 D 命令显示这几个单元的内容

D 2000✓

可以看到这六个地址单元的内容变为 0000 1111 2222 3333 4444 5555。

5. 用 A 命令键入一段汇编源程序，主要是向累加器送入数据和进行运算，执行程序并观察运行结果。

- 1) 在命令行提示符状态下输入：

A 2000↵ ; 表示该程序从 2000H (内存 RAM 区的起始地址) 地址开始

屏幕将显示：

2000:

输入如下形式的程序：

2000: MVRD R0, AAAA ; MVRD 与 R0 之间有且只有一个空格，其他指令相同

2002: MVRD R1, 5555

2004: ADD R0, R1

2005: AND R0, R1

2006: RET ; 程序的最后一个语句，必须为 RET 指令

2007: (直接敲回车键，结束 A 命令输入程序的操作过程)

若输入有误，系统会给出提示并显示出错地址，用户只需在该地址重新输入正确的指令即可。

- 2) 用 U 命令反汇编刚输入的程序

在命令行提示符状态下输入：

U 2000↵

在相应的地址会得到输入的指令及其操作码

注：连续使用不带参数的 U 命令时，将从上一次反汇编的最后一条语句之后接着继续反汇编。

- 3) 用 G 命令运行前面键入的源程序

G 2000↵

程序运行结束后，可以看到程序的运行结果，屏幕显示各寄存器的值，其中 R0 和 R1 的值均为 5555H，说明程序运行正确。

- 4) 用 P 或 T 命令，单步执行这段程序，观察指令执行结果

在命令行提示符状态下输入：

T 2000↵

寄存器 R0 被赋值为 AAAAH

T↵

寄存器 R1 被赋值为 5555H

T↵

做加法运算，和放在 R0，R0 的值变为 FFFFH

T↵

做与运算，结果放在 R0，R0 的值变为 5555H

用 P 命令执行过程同上。

注：T 总是执行单条指令，但执行 P 命令时，则把每一个 CALL 语句连同被调用的子程序一次执行完成。T、P 命令每次执行后均显示所有通用寄存器及状态寄存器的内容，并反汇编出下一条将要执行的指令。

6. 举例编写汇编程序，用“A”命令输入，运行并观察结果

1) 例 1：设计一个小程序，从键盘上接收一个字符并在屏幕上输出显示该字符。

<1> 在命令行提示符状态下输入：

A 2000✓ ；

屏幕将显示：

2000:

输入如下形式的程序：

2000: IN 81 ; 判键盘上是否按了一个键

2001: SHR R0 ; 即串行口是否有了输入的字符

2002: SHR R0

2003: JRNC 2000 ; 未输入完则循环测试

2004: IN 80 ; 接收该字符

2005: OUT 80✓ ; 在屏幕上输出显示字符‘6’

2006: RET✓ ; 每个用户程序都必须用 RET 指令结束

2007: ✓ ; (按回车键即结束输入过程)

注：在十六位机中，基本 I/O 接口的地址是确定的，数据口的地址为 80，状态口的地址为 81。

<2> 用“G”命令运行程序

在命令行提示符状态下输入：

G 2000✓

执行上面输入的程序

光标闪烁等待输入，用户从键盘键入字符后，屏幕会显示该字符。

该例建立了一个从主存 2000H 地址开始的小程序。在这种方式下，所有的数字都约定使用 16 进制数，故数字后不用跟字符 H。每个用户程序的最后一个语句一定为 RET 汇编语句。因为监控程序是选用类似子程序调用方式使实验者的程序投入运行的，用户程序只有用 RET 语句结束，才能保证程序运行结束时能正确返回到监控程序的断点，保证监控程序能继续控制教学机的运行过程。

2) 例 2：设计一个小程序，用次数控制在终端屏幕上输出‘0’到‘9’十个数字符。

<1> 在命令行提示符状态下输入：

A 2020✓

屏幕将显示:

2020:

从地址 2020H 开始输入下列程序:

2020:MVRD R2,000A ; 送入输出字符个数

2022:MVRD R0,0030 ; “0” 字符的 ASCII 码送寄存器 R0

2024:OUT 80 ; 输出保存在 R0 低位字节的字符

2025:DEC R2 ; 输出字符个数减 1

2026:JRZ 202E ; 判 10 个字符输出完否, 已完, 则转到程序结束处

2027:PUSH R0 ; 未完, 保存 R0 的值到堆栈中

2028:IN 81 ; 查询接口状态, 判字符串行输出完成否,

2029:SHR R0 ;

202A:JRN 2028 ; 未完成, 则循环等待

202B:POP R0 ; 已完成, 准备输出下一字符并从堆栈恢复 R0 的值

202C:INC R0 ; 得到下一个要输出的字符

202D:JR 2024 ; 转去输出字符

202E:RET

202F: ✓

该程序的执行码放在 2020H 起始的连续内存区中。若送入源码的过程中有错, 系统会进行提示, 等待重新输入正确汇编语句。在输入过程中, 在应输入语句的位置直接打回车则结束输入过程。

<2> 用 “G” 命令运行程序

在命令行提示符状态下输入:

G 2020✓

执行结果为:

0123456789

思考题: 若把 IN 81, SHR R0, JRN 2028 三个语句换成 3 个 MVR R0, R0 语句, 该程序执行过程会出现什么现象? 试分析并实际执行一次。

提示: 该程序改变这三条语句后, 若用 T 命令单条执行, 会依次显示 0~9 十个数字。若用 G 命令运行程序, 程序执行速度快, 端口输出速度慢, 这样就会跳跃输出。

在命令行提示符状态下输 G 2020, 屏幕显示 09。

类似的, 若要求在终端屏幕上输出 ‘A’ 到 ‘Z’ 共 26 个英文字母, 应如何修改例 1 中给出的程序? 请验证之。

参考答案:

在命令行提示符状态下输入:

A 2100✓

屏幕将显示: 2100:

从地址 2100H 开始输入下列程序：

```
(2100) MVRD R2, 001A      ; 循环次数为 26
      MVRD R0, 0041 ; 字符“A”的值
(2104) OUT 80             ; 输出保存在 R0 低位字节的字符
      DEC R2              ; 输出字符个数减 1
      JRZ 210E            ; 判 26 个字符输出完否, 已完, 则转移到程序结束处
      PUSH R0             ; 未完, 保存 R0 的值到堆栈中
(2108) IN 81              ; 查询接口状态, 判字符串行输出完成否
      SHR R0
      JRNC 2108; 未完成, 则循环等待
      POP R0              ; 已完成, 准备输出下一字符, 从堆栈恢复 R0 的值
      INC R0              ; 得到下一个要输出的字符
      JR 2104             ; 转去输出字符
(210E) RET
```

用 G 命令执行该程序, 屏幕上显示“A”~“Z”26 个英文字母。

例 3： 从键盘上连续打入多个属于‘0’到‘9’的数字字符并在屏幕上显示, 遇到非数字字符结束输入过程。

<1> 在命令行提示符状态下输入：

A 2040✓

屏幕将显示：

2040:

从地址 2040H 开始输入下列程序：

```
(2040) MVRD R2, 0030      ; 用于判数字字符的下界值
      MVRD R3, 0039      ; 用于判数字字符的上界值
(2044) IN 81              ; 判键盘上是否按了一个键,
      SHR R0              ; 即串行口是否有了输入的字符
      SHR R0
      JRNC 2044           ; 没有输入则循环测试
      IN 80               ; 输入字符到 R0
      MVRD R1, 00FF
      AND R0, R1          ; 清零 R0 的高位字节内容
      CMP R0, R2          ; 判输入字符 ≥ 字符‘0’ 否
      JRNC 2053           ; 为否, 则转到程序结束处
      CMP R3, R0          ; 判输入字符 ≤ 字符‘9’ 否
      JRNC 2053           ; 为否, 则转到程序结束处
      OUT 80              ; 输出刚输入的数字字符
```

JMPA 2044 ; 转去程序前边 2044 处等待输入下一个字符

(2053) RET

<2> 在命令行提示符状态下输入:

G 2040↵

光标闪烁等待键盘输入, 若输入 0-9 十个数字字符, 则在屏幕上回显; 若输入非数字字符, 则屏幕不再显示该字符, 出现命令提示符, 等待新命令。

思考题: 本程序中为什么不必判别串行口输出完成否? 设计打入 'A' ~ 'Z' 和 '0' ~ '9' 的程序, 遇到其它字符结束输入过程。

例子 4: 计算 1 到 10 的累加和。

<1> 在命令行提示符状态下输入:

A 2060↵

屏幕将显示:

2060:

从地址 2060H 开始输入下列程序:

(2060) MVRD R1, 0000; 置累加和的初值为 0

MVRD R2, 000A; 最大的加数

MVRD R3, 0000

(2066) INC R3 ; 得到下一个参加累加的数

ADD R1, R3 ; 累加计算

CMP R3, R2 ; 判是否累加完

JRNZ 2066 ; 未完, 开始下一轮累加

RET

<2> 在命令行提示符状态下输入:

G 2060↵

运行过后, 可以用 R 命令观察累加器的内容。R1 的内容为累加和。

结果为: R1=0037 R2=000A R3=000A

例子 5: 设计一个有读写内存和子程序调用指令的程序, 功能是读出内存中的字符, 将其显示到显示器的屏幕上, 转换为小写字母后再写回存储器原存储区域。

<1> 将被显示的 6 个字符 'A' ~ 'F' 送入到内存 20F0H 开始的存储区域中。

在命令行提示符状态下输入:

E 20F0↵

屏幕将显示:

20F0 内存单元原值:

按下列格式输入:

20F0 内存原值: 0041 内存原值: 0042 内存原值: 0043

内存原值: 0044 内存原值: 0045 内存原值: 0046✓

在命令行提示符状态下输入:

从地址 2080H 开始输入下列程序:

```
(2080) MVRD R3, 0006    ; 指定被读数据的个数
      MVRD R2, 20F0     ; 指定被读、写数据内存区首地址
(2084) LDRR R0, [R2]    ; 读内存中的一个字符到 R0 寄存器
      CALA 2100         ; 指定子程序地址为 2100, 调用子程序, 完成显示、
                        ; 转换并写回的功能
      DEC R3            ; 检查输出的字符个数
      JRZ 208B          ; 完成输出则结束程序的执行过程
      INC R2            ; 未完成, 修改内存地址
      JR 2084           ; 转移到程序的 2086 处, 循环执行规定的处理
(208B) RET
```

从地址 2100H 开始输入下列程序:

```
(2100) OUT 80           ; 输出保存在 R0 寄存器中的字符
      MVRD R1, 0020
      ADD R0, R1        ; 将保存在 R0 中的大写字母转换为小写字母
      STRR [R2], R0     ; 写 R0 中的字符到内存, 地址同 LOD 所用的地址
(2105) IN 81            ; 测试串行接口是否完成输出过程
      SHR R0
      JRNC 2105         ; 未完成输出过程则循环测试
      RET              ; 结束子程序执行过程, 返回主程序
```

<2> 在命令行提示符状态下输入:

G 2080✓

屏幕显示运行结果为:

ABCDEF

<3> 在命令行提示符状态下输入:

D 20F0✓

20F0H~20F5H 内存单元的内容为:

0061 0062 0063 0064 0065 0066

例子 6: 设计一个程序在显示器屏幕上循环显示 95 个 (包括空格字符) 可打印字符。

<1>在命令行提示符状态下输入:

A 20A0✓

屏幕将显示:

20A0:

从地址 20A0H 开始输入下列程序:

```

A 20A0          ; 从内存的 20A0 单元开始建立用户的第一个程序
20A0: MVRD R1, 7E ; 向寄存器传送立即数
20A2: MVRD R0, 20 ;
20A4: OUT 80      ; 通过串行接口输出 R0 低位字节内容到显示器屏幕
20A5: PUSH R0     ; 保存 R0 寄存器的内容到堆栈中
20A6: IN 81       ; 读串行接口的状态寄存器的内容
20A7: SHR R0      ; R0 寄存器的内容右移一位, 最低位的值移入标志位 C
20A8: JRNC 20A6   ; 条件转移指令, 当标志位 C 不是 1 时就转到 20A6 地址
20A9: POP R0      ; 从堆栈中恢复 R0 寄存器的原内容
20AA: CMP R0, R1  ; 比较两个寄存器的内容是否相同, 相同则标志位 Z=1
20AB: JRZ 20A0    ; 条件转移指令, 当标志位 Z 为 1 时转到 20A0 地址
20AC: INC R0      ; 把 R0 寄存器的内容增加 1
20AE: JR 20A4     ; 无条件转移指令, 一定转移到 20A4 地址
20AF: RET        ; 子程序返回指令, 程序结束

```

<2> 在命令行提示符状态下输入:

```
G 20A0✓
```

运行过后, 可以观察到显示器上会显示出所有可打印的字符。

上述例子, 都是用监控程序的 A 命令完成输入源汇编程序的。在涉及到汇编语句标号的地方, 不能用符号表示, 只能在指令中使用绝对地址。使用内存中的数据, 也由程序员给出数据在内存中的绝对地址。显而易见, 对这样的极短小程序矛盾并不突出, 但很容易想到, 对很大的程序, 一定会有较大的困难。

在用 A 命令输入汇编源语句的过程中, 有一定用机经验的人, 常常抱怨 A 命令中未提供适当的编辑功能, 这并不是设计者的疏漏, 因为我们并不准备在这种操作方式下支持设计较长的程序, 这种工作应转到提供了交叉汇编程序的 PC 机上去完成。相反的情况是, 输入上述一些小程序, 用监控程序的 A 命令完成, 往往比用交叉汇编完成更简捷。

七. 教学计算机的WINDOWS界面的集成开发环境

上述的实验除了在 PCEC 仿真终端的界面下完成外, 也可以在 WINDOWS 界面的集成开发环境下完成。用户可以将配套光盘中的 TEC2KIDE 的文件夹复制到硬盘建立的 TEC-XP16 的文件夹中。然后选中文件夹中的可执行文件 TEC2KIDE 单击鼠标右键, 在弹出的菜单中选择“发送到”项中的“桌面快捷方式”一项, 建立一个放到桌面上的快捷方式。

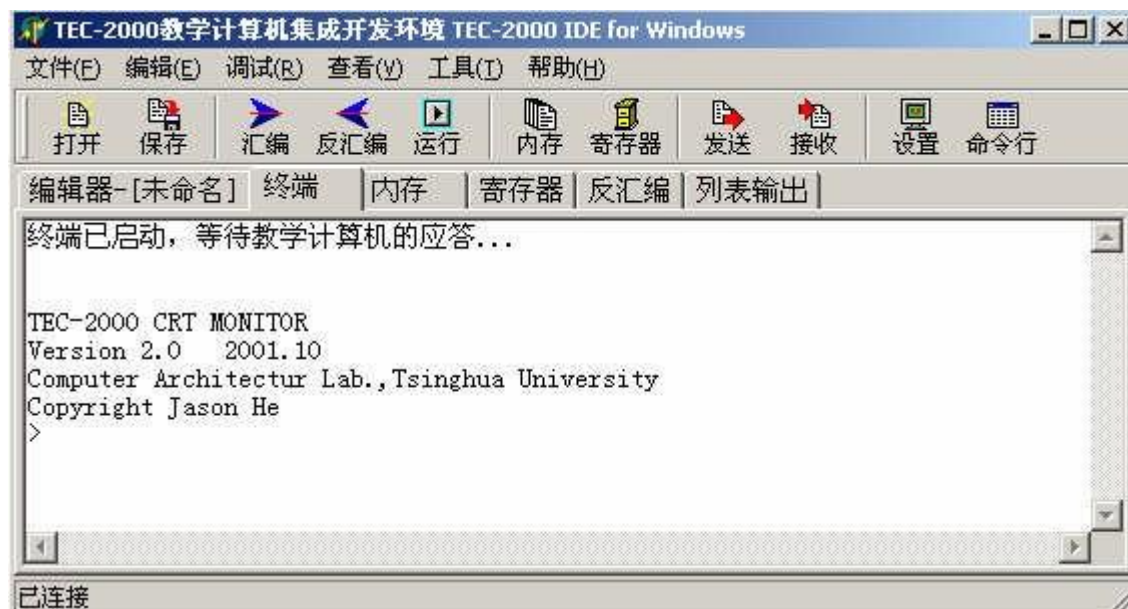
1. 双击桌面上的TEC2KIDE图标, 会出现如下的软件界面:



2. 选中菜单栏中的“终端”选项，打开连接好的教学机的电源，会出现如下通讯界面：



3. 按一次“RESET”按键，再按一次“START”按键，出现如图所示界面：

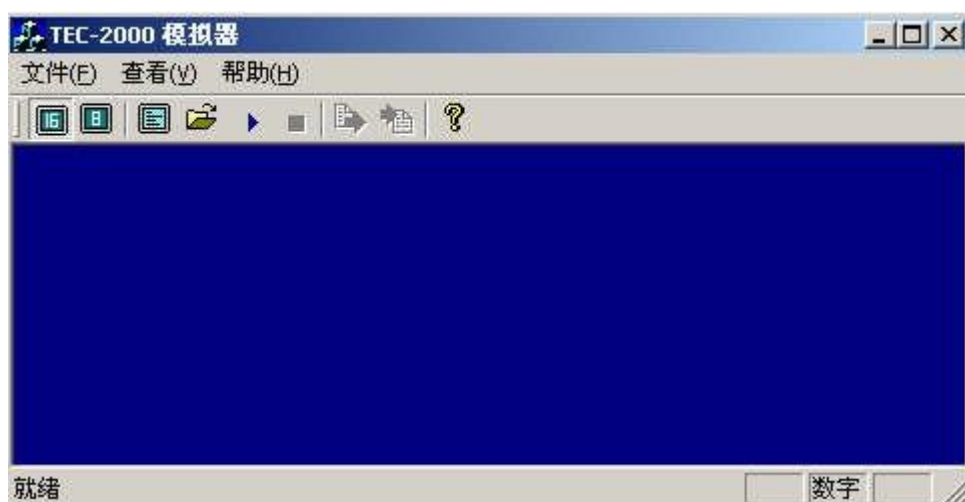


4. 按照第六个实验项目中给出的示例输入程序，并执行，可以观察到同样的实验结果。
5. 建立一个汇编的源文件，并进行编译，下载。选中菜单栏中的“编辑器”选项，可以建立汇编的源文件，也可以打开硬盘上已经建立好的源文件。选中菜单栏中的“汇编”选项，对源文件进行汇编，然后可以选择“发送”选项将文件传送到教学机上运行。
6. 观察内存和寄存器的内容。

八. 软件模拟器

为了让学生更好的理解机器的指令系统，还配套了软件模拟器，联机通讯所能实现的功能在软件模拟器上也一样能实现，这个软件可以让学生拷贝回去，在自己的机器上运行汇编语言的程序，能更好的理解机器的监控程序的功能以及指令系统，对学生的学习有很好的促进作用。

1. 双击桌面上的TEC2KIDE图标，会出现如下的软件界面：



2. 单击标有“16”的图标，选中 16 位机的运行模式；再单击第三项“启动监控”的图标，会出现如下的界面：



3. 在这个界面中可以使用监控程序支持的 DEBUG 命令，输入并且执行程序，也可以观察修改地址单元内容和寄存器的内容。

实验二 脱机运算器实验

一、实验目的：

- 1、深入了解AM2901运算器的功能与具体用法；
- 2、深化运算器部件的组成、设计、控制与使用等知识。

二、实验内容：

在脱机方式下，对于给定指令分析其执行过程中运算的步骤，通过对AM2901运算器所需控制信号的设置，使之完成运算，并核对运算结果。

三、实验要求：

1. 实验前，认真了解AM2901运算器的基本结构，预习所需实验的内容，对于实验数据和实验结果进行预期性的分析，以提高实验效率；
2. 实验过程中，要按正确流程操作，防止损坏设备，分析可能遇到的各种现象，判断结果是否正确，记录运行结果。
3. 实验之后，认真写出实验报告，包括对遇到的各种现象的分析，实验步骤和实验结果。

四、实验说明

脱机运算器实验，是指让运算器从教学计算机整机中脱离出来，此时，它的全部控制与操作均需通过两个 12 位的微型开关来完成，这就谈不上执行指令，只能通过开关、按键控制教学机的运算器完成指定的运算功能，并通过指示灯观察运算结果。

下面先把前边讲过的、与该实验直接有关的结论性内容汇总如下：

1、12位微型开关的具体控制功能分配如下：

A口、B口地址：送给AM2901器件用于选择源与目的操作数的寄存器编号。

I8-I0：选择操作数来源、运算操作功能、选择操作数处理结果和运算器输出内容的3组3位的控制码。

SCi、SSH和SST：用于确定运算器最低位的进位输入、移位信号的入/出和怎样处理AM2901产生的状态标志位的结果。

2、开关位置说明：

做脱机运算器实验时，要用到提供24位控制信号的微动开关和提供16位数据的拨动开关。微动开关是红色的，一共有三个，一个微动开关可以提供12位的控制信号，三个开关分别标有SW1 micro switch、SW2 micro switch 和SW3 micro switch，他们对应的控制信号见表2.2.1；数据开关是黑色的，左边的标有SWH的是高8位，右边的标有SWL的是低8位。微动开关与控制信号对应关系见表(由左到右)：

表 2.2.1 微动开关与控制信号对应关系表

SW1 Micro switch				SW2 Micro switch			SW3 Micro switch		
T3-T0	REQ/MIO/WE	I2-I0	I8-I7	I6-I3	B PORT	A PORT	SST SSH SCI	DC2	DC1

3、开关检测

红色微动开关是该实验系统使用寿命最短的器件，开关好坏的检测方法比较简单，用户将六个控制机器工作方式的开关置于“1XX000”，从左面起第二个和第三个的开关处于任意位置，然后将两个微动开关上的24个小组子依次置为1（开关拨到上方为1），看对应的指示灯是否亮，如果有一个或数个指示

灯不亮，则一般是开关出了问题。

4、微型开关各字段的功能

下面的表格分别给出了 **I₂~I₀**、**I₅~I₃**、**I₈~I₆**、**SST**、**SCI**、**SSH** 的控制码的功能，以方便同学查阅

表 2.2.2 运算数据来源的控制表

编 码			运算数据来源控制	
I ₂	I ₁	I ₀	R	S
0	0	0	A	Q
0	0	1	A	B
0	1	0	0	Q
0	1	1	0	B
1	0	0	0	A
1	0	1	D	A
1	1	0	D	Q
1	1	1	D	0

表 2.2.3 运算功能的控制表

编 码			运算功能控制
I ₅	I ₄	I ₃	
0	0	0	R+S
0	0	1	S-R
0	1	0	R-S
0	1	1	R∨S
1	0	0	R∧S
1	0	1	$\overline{R} \wedge S$
1	1	0	R⊕S
1	1	1	$\overline{R} \oplus S$

表 2.2.4 运算结果的控制表

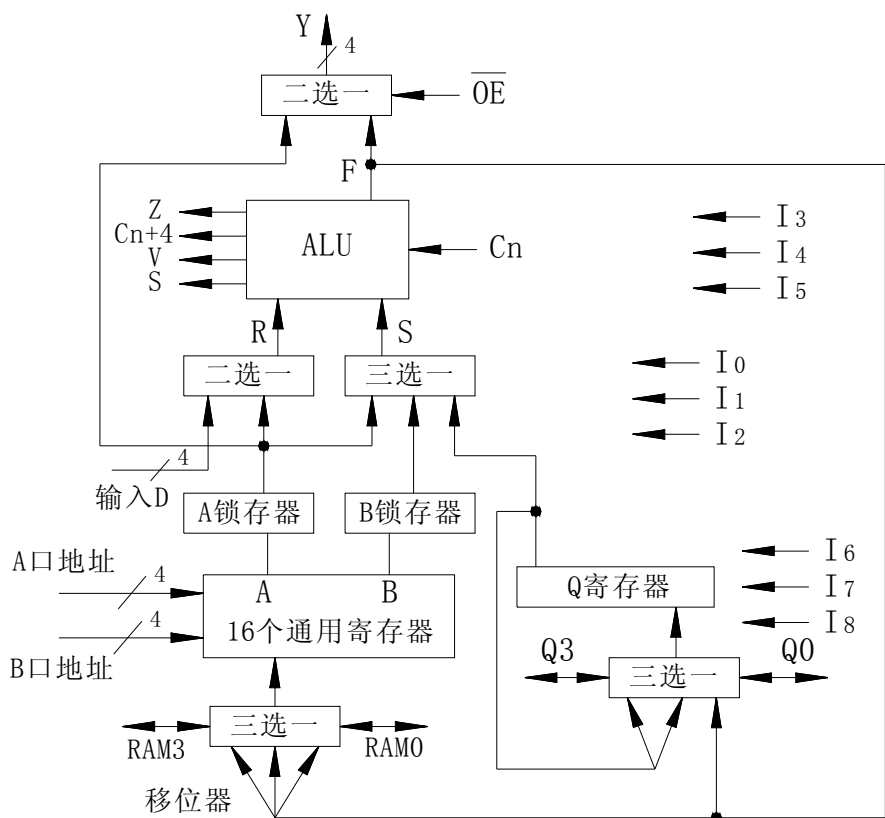
编 码			运算结果控制		
I ₈	I ₇	I ₆	寄存器组	Q 寄存器	Y 输出
0	0	0	--	F→Q	F
0	0	1	--	--	F
0	1	0	F→B	--	A
0	1	1	F→B	--	F
1	0	0	F/2→B	Q/2→Q	F
1	0	1	F/2→B	--	F
1	1	0	2F→B	2Q→Q	F
1	1	1	2F→B	--	F

表 2.2.5 状态寄存器的接收与保持 SST 表

选择码 SST	状态位				说明
	C	Z	V	S	
0 0 0	C	Z	V	S	4 个标志位的值不变
0 0 1	CY	F=0	OVR	F3	接收 ALU 的标志位输出
0 1 0	内部总线对应的一位				恢复标志位原现场值
0 1 1	0	Z	V	S	置 C 为 0，另 3 个标志位不变
1 0 0	1	Z	V	S	置 C 为 1，另 3 个标志位不变
1 0 1	RAM0	Z	V	S	右移操作，另 3 个标志位不变
1 1 0	RAM3	Z	V	S	左移操作，另 3 个标志位不变
1 1 1	Q0	Z	V	S	联合右移，另 3 个标志位不变

SCI 编码	Cn 取值
0 0	0
0 1	1
1 0	C
1 1	

SSH 1 位编码	左 移		右 移		说明
	RAM0	Q0	RAM3	Q3	
0	0	X	0	X	通用寄存器逻辑移位
1	C	X	C	X	通用寄存器与 C 循环移位



五、实验步骤

1、按照表 2.2.8 中的微码和数据开关，对运算器的功能进行设置。操作步骤如下：

微动开关								数据开关
I8-I6	I5-I3	I2-I0	SST	SSH	SCI	B	A	D15-D0
011	000	111	001	0	00	0001	不用	AAAA H

- 43

键，再按一下START按键，进行初始化。

(2) 通过16个数据开关设置立即数AAAA H。

(3) 通过SW1、SW2、SW3设置各微码。

2、按一次START键，立即数XXXX H置入R1，通过显示灯察看按START键后的输出。

若要进行其他操作：

(1) 重新设置SW1、SW2、SW3、数据开关，通过显示灯观察输出。

(2) 然后按START键执行操作，通过显示灯观察按下START键后的输出，检查运算结果是否正确。

3、完成表2.2.9中的各种运算，记录按START键前和按START键后的ALU输出及标志位C、Z、V、S的值。

问题：分析比较各指令按START键前和按START键后的值，是否有不同并解释。

表2.2.9 指令微码及输出状态表

运算	I8-I6	I5-I3	I2-I0	SST	SSH	SCi	B	A	压START前		压START后	
									ALU输出	CZVS	ALU输出	CZVS
0101→R0												
1010→R1												
R0+R1→R0												
R0-R1→R0												
R1-R0→R1												
R0∨R1→R0												
R0∧R1→R0												
R0⊕R1→R0												
¬(R0⊕R1) →R0												
2*R0→R0												
R0/2→R0												

六、实验注意事项：

1、连接电源线和通讯线前TEC-XP16实验系统的电源开关一定要处于断开状态，否则可能会对TEC-XP16实验系统上的芯片和PC机的串口造成损害。

2. 六个黑色控制开关的功能示意图如下：

单步	手动置指令	组合逻辑	联机	8 位	FPGA	
○	○	○	○	○	○	上面
○	○	○	○	○	○	下面
连续	从内存读指令	微程序	脱机	16 位	MACH	

3. 几种常用的工作方式（开关拨到上方表示为 1，拨到下方为 0;）

工作方式	6个拨动开关
连续运行程序、组合逻辑控制器、联机、16位机、MACH	001100
连续运行程序、微程序控制器、联机、16位机、MACH	000100
单步、手动置指令、组合逻辑控制器、联机、16位机、MACH	111100
单步、手动置指令、微程序控制器、联机、16位机、MACH	110100
16位机、脱机运算器实验、MACH	1XX000
连续运行程序、联机、16位机、FPGA	00X101

实验三 存储器部件教学实验

一、实验目的：

- 1、熟悉ROM芯片和RAM芯片在功能和使用方法等方面的相同和差异之处；学习用编程器设备向EEPROM芯片内写入一批数据的过程和方法。
- 2、理解并熟悉通过字、位扩展技术实现扩展存储器系统容量的方案。
- 3、了解静态存储器系统使用的各种控制信号之间正常的时序关系。
- 4、了解如何通过读、写存储器的指令实现对58C65 ROM芯片的读、写操作。
- 5、加深理解存储器部件在计算机整机系统中的作用。

二、预习要求：

教学计算机存储器系统由ROM和RAM两个存储区组成，分别由EPROM芯片（或EEPROM芯片）和RAM芯片构成。TEC-XP+教学计算机中还安排了另外几个存储器器件插座，可以插上相应存储器芯片以完成存储器容量扩展的教学实验，为此必须比较清楚地了解：

- 1、TEC-XP+教学机的存储器系统的总体组成及其连接关系。
- 2、TEC-XP+教学机的有关存储器芯片、I/O接口芯片的片选信号控制和读写命令的给出和具体使用办法。
- 3、RAM和EPROM、EEPROM存储器芯片在读写控制、写入时间等方面的同异之处，并正确建立连线关系和在程序中完成正确的读写过程。
- 4、如何在TEC-XP+教学机中使用扩展的存储器空间并检查其运行的正确性。

三、实验说明：

（1）内存储器原理

内存储器是计算机中存放正在运行中的程序和相关数据的部件。在教学计算机存储器部件设计中，出于简化和容易实现的目的，选用静态存储器芯片实现内存存储器的存储体，包括唯读存储区（ROM，存放监控程序等）和随读写存储区（RAM）两部分，ROM存储区选用4片长度8位、容量8KB的58C65芯片实现，RAM存储区选用2片长度8位、容量2KB的6116芯片实现，每2个8位的芯片合成一组用于组成16位长度的内存字，6个芯片被分成3组，其地址空间分配关系是：0-1777h用于第一组ROM，固化监控程序，2000-2777h用于RAM，保存用户程序和用户数据，其高端的一些单元作为监控程序的

数据区，第二组 ROM 的地址范围可以由用户选择，主要用于完成扩展内存容量（存储器的字、位扩展）的教学实验。内存储器和串行接口线路的组成如图 2.3.1 所示。

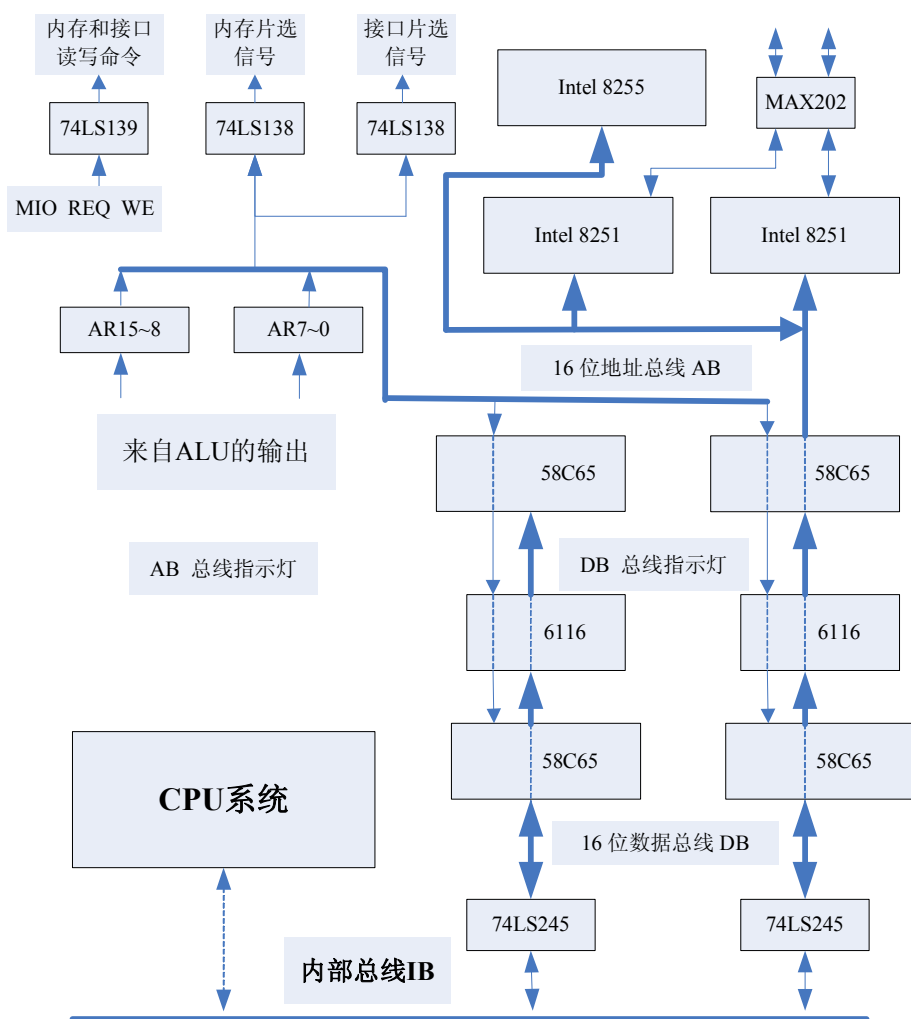


图 2.3.1 内存储器与串行接口电路

地址总线的低 13 位送到 ROM 芯片的地址线引脚（RAM 芯片只使用地址总线的低 11 位），用于选择芯片内的一个存储字。用于实现存储字的高位字节的 3 个芯片的数据线引脚、实现低位字节的 3 个芯片的数据线引脚分别连接在一起接到数据总线的高、低位字节，是实现存储器数据读写的信息通路。数据总线要通过一个双向三态门电路与 CPU 一侧的内部总线 IB 相连接，已完成存储器、接口电路和 CPU 之间的数据通讯，如图 2.3.1 中的虚线部分所示。

这里用到 3 个译码器电路，其中一片 74LS138 译码器芯片接收地址总线最高的 3 位地址信息，当需要内存工作时，由这片译码器产生内存芯片的 8 个片选信号，以选择哪一个空间范围的内存区可以读写。另外一片 74LS138 译码器芯片接收地址总线低位字节的最高 4 位地址信息（最高一位恒定为 1），当需要接口电路工作时，由这片译码器产生接口芯片的 8 个片选信号，以选择哪一个接口电路可以读写。一片 74LS139 双二-四译码器芯片接收控制器送来的 3 位控制信号 MIO（有无内存或者接口电路要读写）、REQ（是内存还是接口电路要读写）、WE（是读操作还是写操作），当这 3 位控制信号的组合为 1××、000、001、010、011 时，译码器将产生读内存操作、写内存操作、读接口操作、写接口操作、内存和接口芯片都无读写操作的控制信号。

在这里还要说明如下两个问题。第一，要扩展 8K 字的存储空间，需要使用 2 片（每一片有 8KB 容

量，即芯片内由 8192 个单元、每个单元由 8 个二进制位组成）存储器芯片实现。第二，当存储器选用 58C65 ROM 芯片时，它属于电可擦出的 EPROM 器件，可以通过专用的编程器软件和设备向芯片的写入相应的内容，这是正常的操作方式。也可以通过写内存的指令向芯片的指定单元写入 16 位的数据，只是每一次的这种写操作需要占用长得多写入时间，例如几百个微秒，可以通过运行完成等待功能的子程序来加以保证。对 58C65 ROM 芯片执行读操作时，需要保证正确的片选信号（/CE）为低点平，使能控制信号（/OE）为低电平，读写命令信号（/WE）为高电平，读 58C65 ROM 芯片的读出时间与读 RAM 芯片的读出时间相同，无特殊要求；对 58C65 ROM 芯片执行写操作时，需要保证正确的片选信号（/CE）为低电平，使能控制信号（/OE）为高电平，读写命令信号（/WE）为低电平，写 58C65 ROM 芯片的维持时间要比写 RAM 芯片的操作时间长得多。为了防止对 58C65 ROM 芯片执行误写操作，可通过把芯片的使能控制引脚（/OE）接地来保证，或者确保读写命令信号（/WE）恒为高电平。

串行接口芯片的 8 位数据线引脚连接到数据总线 DB 的低位字节，它与 CPU 之间每次交换 8 位信息，属于并行操作关系。串行接口芯片和设备之间的连接，是通过连接到设备端的另外一个串行接口芯片完成的，在两端的接口芯片之间以串行方式实现通讯，即遵从一定的通讯协议，对 8 位的数据采用逐位传送的方案处理，并把信号的电平从 TTL 电路的 0~4V 左右提高到正负 12V 或者 0~12V 左右，以增强信号传送过程中的抗干扰能力，图中的 MAX202 芯片就是采用倍压方案完成电平转换功能的，有了这个芯片就可以不再使用直流+12V 和-12V 两路电源了。

2、扩展教学机的存储器空间

教学机的主板上，预留了两片存储器芯片的 28 个引脚的器件插座，可以插上 28 个引脚的 58C65ROM（8KB 容量）器件，也可以插上 24 个引脚的 6161 RAM（2KB 容量）器件。每片器件都只有 8 位数据线，为构建 16 位的字长的存储体，需要使用 2 片存储器器件，这就是通常说的存储器的位数（字长）扩展技术。为了保证新插上去的芯片与原有存储器芯片有正确的连接关系，需要为这 2 片存储器芯片分配地址空间范围，这就是通常说的存储器的容量（字数）扩展技术，通过把对内存高位地址译码产生的不同信号连接到这两个芯片的片选信号引脚完成。除此之外，还要向这 2 片存储器芯片提供正确的读写控制信号、使能控制信号（/OE）。再考虑到 58C65 ROM 器件和 6161 RAM 器件的引脚数不同，使用的控制信号也不完全相同，可能还有几个跳接线需要连通或者断开。对于用 2 片 58C65 ROM 器件扩展 8K 字的 ROM 存储区的实验，可以在使用编程器设备向该存储器芯片内写入程序和数后，再把芯片插到器件插座中，在唯读的操作方式运行其中的程序或 使用其中的数据，检查结果的正确性。通过这个实验可以学习编程器的使用方法和向器件内写入信息的操作步骤。也可以直接用教学机的指令向 58C65 ROM 器件内写入信息，例如用教学机的监控命令 E 向器件内输入某些信息，此时需要保证向器件提供正确的读写控制信号和使能控制信号，并确保每一次写操作要维持约 1 毫秒左右的时间，此方式下的 58C65 ROM 器件提供了类似于 RAM 器件的功能，这只能算是一种变通的用法。

用 2 片 6161 RAM 器件扩展 2K 字的 RAM 存储区的实验。此时需要把 24 脚的 RAM 芯片按照与器件插座右对齐的方式查到插座中，并适当的进行必要的跳接线连通或者断开操作。全部接线都连接正确之后，就可以对这个存储区执行写入或者读出数据的操作功能。

四、实验内容：

1、要完成存储器容量扩展的教学实验，需为扩展存储器选择一个地址，并注意读写和OE等控制信号的

正确状态。

2、用监控程序的D、E命令对存储器进行读写，比较RAM（6116）、EEPROM（28系列芯片）、EPROM（27系列芯片）在读写上的异同。

3、用监控程序的A命令编写一段程序，对RAM（6116）进行读写，用D命令查看结果是否正确。

4、用监控程序的A命令编写一段程序，对扩展存储器EEPROM（28 系列芯片）进行读写，用D命令查看结果是否正确；如不正确，分析原因，改写程序，重新运行。

五、实验步骤：

1、接通教学机电源。

2、将教学机左下方的6个拨动开关置为001100（连续、从内存读指令、组合、联机、16 位、MACH）。

3、按一下“RESET”按键，接着按“START”按键。

4、RAM（6116）支持即时读写，可直接用A、E 命令向扩展的存储器输入程序或改变内存单元的值。

RAM 中的内容在断电后会消失，重新启动实验机后会发现内存单元的值发生了改变。

(1) 用E命令改变内存单元的值并用D命令观察结果。

1) 在命令行提示符状态下输入：

E 2020 ✓

屏幕将显示： 2020 内存单元原值：

按如下形式键入：

2020 原值：2222 （空格）原值：3333（空格）原值：4444（空格）原值：5555 ✓

2) 在命令行提示符状态下输入：

D 2020 ✓

屏幕将显示从2020内存单元开始的值，其中2020H~2023H的值为：

2222 3333 4444 5555

问题：断电后重新启动教学实验机，用D命令观察内存单元2020~2023 的值。会发现什么问题，为什么？

(2) 用A 命令输入一段程序，执行并观察结果。

在命令行提示符状态下输入：

A 2000 ✓

屏幕将显示： 2000:

按如下形式键入：

2000: MVRD R0, AAAA

MVRD R1, 5555

AND R0, R1

RET

问题：采用单步和连续两种方式执行这段程序，察看结果，断电后发生什么情况？

5、将扩展芯片上方标有EXTROMH和EXTROML的“/CS”信号用自锁紧线短接，然后短接到MEMDC 138 芯片的上方的标有“4000—5fff”地址单元。

注意：标有/CS 的圆孔针与标有MEM/CS 的一排圆孔针中的任意一个都可以用导线相连；连接的地址范围是多少，用户可用的地址空间就是多少。

下面以58C65芯片为例，进行扩展EEPROM实验。

6、58C65的读操作和一般的RAM一样，而其写操作，需要一定的时间，大约为1 毫秒。因此，需要编写一延迟子程序，在对EEPROM进行写操作时，调用该子程序，以完成正确的读写。

(1) 用E 命令改变内存单元的值并用D命令观察结果。

1) 在命令行提示符状态下输入：

E 5000✓

屏幕将显示： 5000 内存单元原值：

按如下形式键入：

5000 原值：2424（按空格）原值：3636（按空格）原值：4848（按空格）原值：5050✓

2) 在命令行提示符状态下输入：

D 5000✓

屏幕将显示5000H~507FH 内存单元的值，从5000 开始的连续四个内存单元的值依次为2424 3636 4848 5050。

3) 断电后重新启动，用D命令察看内存单元5000~5003的值，会发现这几个单元的值没有发生改变，说明EEPROM的内容断电后可保存。

(2) 58C65存储器不能直接用A 命令输入程序，单字节的指令可能会写进去，双字节指令的低位会出错（建议试一试），本实验将程序放到RAM（6116）中，调用延时子程序，访问58C65中的内存地址。

下面给出的程序，在5000H~500FH 单元中依次写入数据0000H、0001H、...000FH。

从2000H单元开始输入主程序：

（2000）MVRD R0, 0000

MVRD R2, 0010 ; R2记录循环次数

MVRD R3, 5000 ; R3的内容为16 位内存地址

（2006）STRR [R3], R0 ; 将R0寄存器的内容放到R3 给出的内存单元中

CALA 2200 ; 调用程序地址为2200的延时子程序

INC R0 ; R0加1

INC R3 ; R3加1

DEC R2 ; R2减1

JRNZ 2006 ; R2不为0跳转到2006H

RET

从2200H 单元开始输入延时子程序：

（2200）PUSH R3

MVRD R3, FFFF

（2203）DEC R3

JRNZ 2203

POP R3

RET

运行主程序，在命令提示符下输入：G 2000✓。

程序执行结束后，在命令提示符下输入：D 5000↵；

可看到从5000H开始的内存单元的值变为

5000: 0000 0001 0002 0003 0004 0005 0006 0007

5008: 0008 0009 000A 000B 000C 000D 000E 000F。

思考：

- 1) 为何能用E 命令直接写58C65的存储单元，而A命令则有时不正确；
- 2) 修改延时子程序，将其延时改短，可将延时子程序中R3的内容赋成00FF或0FFF等，再看运行结果。

实验四 组合逻辑控制器部件教学实验

一、实验目的：

通过看懂教学计算机中已经设计好并正常运行的几条典型指令（例如，ADD、SHR、OUT、MVRD、JRC、RET、CALA等指令）的功能、格式和执行流程，然后自己设计几条指令的功能、格式和执行流程，并在教学计算机上实现、调试正确。其最终要达到的目的是：

- 1、深入理解计算机控制器的功能、组成知识。
- 2、深入地学习计算机各类典型指令的执行流程。
- 3、对指令格式、寻址方式、指令系统、指令分类等建立具体的总体概念。
- 4、学习组合逻辑控制器的设计过程和相关技术。

二、实验说明：

控制器设计是学习计算机总体组成和设计的最重要的部分。要在TEC-XP16教学计算机完成这项实验，必须比较清楚地懂得：

- 1、TEC-XP16教学机的组合逻辑控制器主要由MACH器件组成。
- 2、TEC-XP16教学机上已实现了29条基本指令。
- 3、应了解监控程序的A命令只支持基本指令，扩展指令应用E命令将指令代码写入到相应的存储单元中；不能用T、P命令单步调试扩展指令，只能用G命令执行有扩展指令的程序。
- 4、要明白TEC-XP16教学机支持的指令格式及指令执行流程分组情况；理解TEC-XP教学机中已经设计好并正常运行的各类指令的功能、格式和执行流程，也包括控制器设计与实现中的具体线路和控制信号的组成。
- 5、要明确自己要实现的指令格式、功能、执行流程设计中必须遵从的约束条件。

为了完成自己设计几条指令的功能、格式和执行流程，并在教学计算机上实现、调试正确的实验内容，具体过程包括：

- （1）、确定指令格式和功能，要受到教学机已有硬件的约束，应尽量与已实现指令的格式和分类办法保持一致。
- （2）、划分指令执行步骤并设计每一步的执行功能，设计节拍状态的取值，应参照已实现指令的处理办法来完成，特别要注意的是，读取指令的节拍只能用原来已实现的，其他节拍的节拍状态也应尽可能地与原用节拍的状态保持一致和相近。
- （3）、在指令流程表中填写每一个控制信号的状态值，基本上是个查表填数的过程，应该特别仔细，并有意识地体会这些信号的控制作用。
- （4）、在给定的mach的源文件中添加扩展指令的控制信号的逻辑表达式，编译适配后下载到MACH器件中。
- （5）、写一个包含你设计的指令的程序，通过运行该程序检查执行结果的正确性，初步判断你的设计是否正确；如果有问题，通过几种办法查出错误并改正，（比如手动置指令，单步调试每个节拍对应的控制信号）继续调试，直到完全正确。

三、实验内容：

- 1、完成控制器部件的教学实验，主要内容是由学生自己设计几条指令的功能、格式和执行流程，并

在教学计算机上实现、调试正确。

2、首先是看懂TEC-XP+教学计算机的功能部件组成和线路逻辑关系，然后分析教学计算机中已经设计好并正常运行的几条典型指令（例如，ADD、SHR、OUT、MVRD、JRC、CALA、RET 等指令）的功能、格式和执行流程。

3、设计几条指令的功能、格式和执行流程，并在教学计算机上实现、调试正确。例如ADC、JRS、JRNS、LDRA、STOR、JMPR 等指令，当然也可以设计与实现其它的指令，包括原来已经实现的基本指令（要变换为另外一个指令操作码）或自己确定的指令。在原来提供的MACH 程序的基础上按照ABEL 语言的要求添加新指令的控制信号，编译产生JED 文件并下载到MACH 芯片里。软件的使用和下载参见光盘中的附录。

4、单条运行指令，查看指令的功能、格式和执行流程。

先将教学机左下方的5个拨动开关置为11110，再按一下“RESET”按键，然后通过16 位的数据开关（SWH、SWL）置入指令，按“START”按键单步送脉冲，通过指示灯观察控制信号的变化。

5、用监控程序的A、E（扩展指令必须用E 命令置入）命令编写一段小程序，观察运行结果。实验时将教学机左下方的5个拨动开关置为00110，运行编写的小程序。观察终端显示的结果，检验设计的指令是是否正确。若与预定结果不符，可查看指令的功能、格式、执行、流程设计的是否正确。

四、实验步骤：

1、接通教学机电源。

2、将教学机左下方的6个拨动开关置为111100（单步、手动置指令、组合、16 位、联机、MACH）。

3、按一下“RESET”按键。

4、通过16位的数据开关SWH、SWL置入16位的指令操作码。

5、在单步方式下，通过指示灯观察各类基本指令的节拍。

(1) 选择基本指令的A组指令中的ADD指令，观察其节拍流程：

1) 置拨动开关SW=00000000 00000001；（表示指令ADD R0, R1 ）

2) 按RESET按键；节拍指示灯T3~T0显示1000；（本拍在第1次复位后才会执行）

3) 按START按键；节拍指示灯T3~T0显示0000；（以上两拍,为公共节拍，在手动置指令方式下无意义）

4) 按START按键；节拍指示灯T3~T0显示0010；（本拍也是公共节拍，将指令编码写入指令寄存器IRH、IRL）

5) 按START 按键；节拍指示灯T3~T0 显示0011；（本拍执行ADD指令， $R0 \leftarrow R0 + R1$ 操作）

可以看到，A组指令（包括ADD、SUB、CMP、AND、XOR、SHR、SHL、INC、DEC、TEST、OR、MVRD、JR、JRC、JRNC、JRZ、JRNZ）的执行除公共节拍外，只需一步完成。

(2) 选择基本指令的B组指令中的PUSH指令，观察其节拍流程：

1) 置拨动开关SW=10000101 00000000；（表示指令PUSH ）

2) 按RESET按键；节拍指示灯T3~T0显示1000；（本拍在第1次复位后才会执行）

3) 按START按键；节拍指示灯T3~T0显示0000；（以上两拍,为公共节拍，在手动置指令方式下无意义）

4) 按START按键；节拍指示灯T3~T0显示0010；（本拍也是公共节拍，将指令编码写入指令寄存器IRH、IRL）

5) 按START按键；节拍指示灯T3~T0显示0110（本拍执行PUSH指令的第一步，修改地址寄存器和堆

栈的值，即 $AR, SP \leftarrow SP-1$ ，使其指向堆栈空间）

6) 按START 按键；节拍指示灯T3~T0 显示0100；（本拍执行PUSH 指令的第二步， $MEM \leftarrow SR$ ）

可以看到，B组指令（包括JMPA、LDRR、IN、STRR、PSHF、PUSH、OUT、POP、MVRD、POP、RET）的执行除公共节拍外,需两步完成。

(3) 选择基本指令的D组指令中的CALA指令，观察其节拍流程：

1) 置拨动开关 $SW=11001110\ 00000000$ ；（表示指令CALA）

2) 按RESET按键；节拍指示灯T3~T0显示1000；(本拍在第1次复位后才会执行)

3) 按START按键；节拍指示灯T3~T0显示0000；(以上两拍为公共节拍，在手动置指令方式下无意义)

4) 按START按键；节拍指示灯T3~T0显示0010；（本拍也是公共节拍，将指令编码写入指令寄存器IRH、IRL）

5) 按START按键；节拍指示灯T3~T0显示0110；（本拍 $PC \rightarrow AR, PC+1 \rightarrow PC$ ）

6) 按START按键；节拍指示灯T3~T0显示0100；（本拍 $(AR) \rightarrow Q$ ）

7) 按START按键；节拍指示灯T3~T0显示0111；（本拍 $SP-1 \rightarrow SP、AR$ ）

8) 按START按键；节拍指示灯T3~T0显示0101；（本拍 $PC \rightarrow MEM, Q \rightarrow PC$ ）

可以看到，D 组指令CALA 除公共节拍外，需四步完成；

6、单步方式下，通过指示灯观察各类基本指令的控制信号。

(1) 选择基本指令的A 组指令中的SHR 指令，观察其执行过程中控制信号的变化，分析其作用。

1) 置拨动开关 $SW=00001011\ 00010000$ ；(表示指令SHR R1)

2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号如下表。

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0011	SHR	0000 1011	1	0	0	0000	0001	00	0	101	000	011	101	000	000

(2) 选择基本指令的B组指令中的JMPA指令，观察其执行过程中控制信号的变化，分析其作用。

1) 置拨动开关 $SW=10000000\ 00000000$ ；(表示指令JMPA)

2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号如下表。

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0110	JMPA	1000 0000	1	0	0	0101	0101	01	0	010	000	011	000	000	011
0100	JMPA	1000 0000	0	0	1	0000	0101	00	0	011	000	111	000	000	000

(3) 选择基本指令的D组指令中的CALA指令，观察其执行过程中控制信号的变化，分析其作用。

1) 置拨动开关 $SW=11001110\ 00000000$ ；（表示指令CALA）

2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号如下表：

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010	CALA	1100 1110	0	0	1	0000	0000	00	0	001	000	000	000	000	001
0110	CALA	1100 1110	1	0	0	0101	0101	01	0	010	000	011	000	000	011
0100	CALA	1100 1110	0	0	1	0000	0000	00	0	000	000	111	000	000	000
0111	CALA	1100 1110	1	0	0	0000	0100	00	0	011	001	011	000	000	011
0101	CALA	1100 1110	0	0	0	0101	0101	00	0	010	000	010	000	001	000

7、在以上几步实验的基础上，选择几条扩展指令，设计出扩展指令的节拍和每拍对应的控制信号。（节拍的设计参加节拍的流程图，扩展指令的节拍，在出厂时的TIMING GAL中已实现，学生可以不用设计，只需看懂节拍GAL 的逻辑表达式即可,但其控制信号需用户来扩展实现，这一步，只是来观察扩展指令的节拍。）

单步方式下，通过指示灯观察各类扩展指令的节拍

(1) 选择扩展指令的A组指令中的RCR指令，观察其节拍流程：

- 1) 置拨动开关SW=00101011 00010000；（表示指令RCR R1）
- 2) 按RESET按键；节拍指示灯T3~T0显示1000；（本拍在第1次复位后才会执行）
- 3) 按START按键；节拍指示灯T3~T0显示0000；（以上两拍为公共节拍，在手动置指令方式下无意义）
- 4) 按START按键；节拍指示灯T3~T0显示0010；（本拍也是公共节拍，将指令编码写入指令寄存器 IRH、IRL）
- 5) 按START按键；节拍指示灯T3~T0显示0011；（本拍完成循环右移操作，RCR DR）

可以看到，A组扩展指令（包括ADC、SBB、RCL、RCR、NOT、JMPR、ASR、JRS、JRNS、CLC、STC、EI、DI）除公共节拍外，只需一步完成。

(2) 选择扩展指令的C组指令中的LDRA指令，观察其节拍流程：

- 1) 置拨动开关SW=11100100 00000000；（表示指令LDRA）
- 2) 按RESET按键；节拍指示灯T3~T0显示1000；（本拍在第1次复位后才会执行）
- 3) 按START按键；节拍指示灯T3~T0显示0000；（以上两拍为公共节拍，在手动置指令方式下无意义）
- 4) 按START按键；节拍指示灯T3~T0显示0010；（本拍也是公共节拍，将指令编码写入指令寄存器 IRH、IRL）
- 5) 按START按键；节拍指示灯T3~T0显示0110；（本拍完成操作PC→AR，PC+1→PC）
- 6) 按START按键；节拍指示灯T3~T0显示0111；（本拍完成操作MEM→AR）
- 7) 按START按键；节拍指示灯T3~T0显示0101；（本拍完成操作MEM→DR）

可以看到，C组扩展指令（包括CALR、LDRA、LDOR、STOR、STAR）除公共节拍外，需三步完成。

8、设计几条扩展指令的控制信号如下表：

(1) 选择扩展指令ADC、STC、JRS、LDRX、STRX 和JMPR，其节拍和设计的控制信号为：

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0011	ADC	0010 0000	1	0	0	SR	DR	10	0	011	000	001	001	000	000
	JRS	0110 0100	1	0	0	0101	0101	00	0	0S1	000	101	000	010	000
	STC	0110 1101	1	0	0	0000	0000	00	0	001	000	000	100	000	000
	JMPR	0110 0000	1	0	0	SR	0101	00	0	011	000	100	000	000	000
0110	LDRX	1110 0101	1	0	0	0101	0101	01	0	010	000	011	000	000	011
	STRX	1110 0110	1	0	0	0101	0101	01	0	010	000	011	000	000	011
0111	LDRX	1110 0101	0	0	1	SR	0000	00	0	001	000	101	000	000	011
	STRX	1110 0110	0	0	1	SR	0000	00	0	001	000	101	000	000	011
0101	LDRX	1110 0101	0	0	1	0000	DR	00	0	011	000	111	000	000	000
	STRX	1110 0110	0	0	0	0000	DR	00	0	001	000	011	000	001	000

(2) 根据设计的控制信号的表格用ABEL语言编写MACH的逻辑表达式，老师可以参考提供的组合逻辑全指令的MACH程序的逻辑表达式MACHC.JED。

(3) 将编译好的程序MACHC.JED下载到MACH芯片内。

9、单步方式下，通过指示灯观察上面扩展的几条扩展指令的控制信号是否与设计的一致。

(1) 观察A组指令中的ADC指令：

- 1) 置拨动开关SW=00100000 00010000；
- 2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号如下表。

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0011	ADC	0010 0000	1	0	0	SR	DR	10	0	011	000	001	001	000	000

(2) 观察A组指令中的JRS指令：

- 1) 置拨动开关SW=01100100 00000000；
- 2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号如下表。

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0011	JRS	0110 0100	1	0	0	0101	0101	00	0	0S1	000	101	000	010	000

(3) 观察A组指令中的STC指令：

- 1) 置拨动开关SW=01101101 00000000；

2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号如下表：

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0011	STC	0110 0100	1	0	0	0000	0000	00	0	001	000	000	100	000	000

(4) 观察A组指令中的JMPR指令：

1) 置拨动开关SW=01100000 00000001；

2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号如下表。

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0011	JMPR	0110 0000	1	0	0	0001	0101	00	0	011	000	100	000	000	000

(5) 观察C组指令中的LDRX指令：

1) 置拨动开关SW=11100101 00000000；

2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号。

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0110	LDRX	1110 0101	1	0	0	0101	0101	01	0	010	000	011	000	000	011
0111	LDRX	1110 0101	0	0	1	SR	0000	00	0	001	000	101	000	000	011
0101	LDRX	1110 0101	0	0	1	0000	DR	00	0	011	000	111	000	000	000

6) 观察C组指令中的STRX指令：

1) 置拨动开关SW=11100110 00000000；

2) 先按“RESET”按键；再连续按“START”按键，观察每一步的节拍及控制信号。

节拍	指令	编码	/MIO	REQ	/WE	A	B	Sci	SSH	I8-6	I5-3	I2-0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	0	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	0	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	0	001	000	000	000	000	001
0110	STRX	1110 0110	1	0	0	0101	0101	01	0	010	000	011	000	000	011
0111	STRX	1110 0110	0	0	1	SR	0000	00	0	001	000	101	000	000	011
0101	STRX	1110 0110	0	0	0	0000	DR	00	0	001	000	011	000	001	000

10、用教学机已实现的基本指令和扩展的几条指令编写程序并运行，测试扩展的几条指令是否正确。

(1) 测试ADC指令。

在命令行提示符状态下输入：

A 2000✓

屏幕将显示：

2000:

从地址2000H 开始输入下列程序：

2000: MVRD R0,0101 ； 给R0赋值0101

2002: MVRD R1,1010 ； 给R1赋值1010

2004: ✓

在命令行提示符状态下输入：

A 2006✓

2006: RET

2007: ✓

扩展指令STC、ADC不能用A命令键入，必须用E命令在相应的内存地址键入操作码所有扩展指令都必须用E命令键入。

用E命令输入STC、ADC R0,R1的代码，在命令行提示符状态下输入：

E 2004✓

2004: 6D00

2005: 2001

2006: ✓

用G 命令运行前面刚键入源程序，在命令行提示符状态下输入：

G 2000✓

用R 命令察看寄存器的内容，在命令行提示符状态下输入

R✓

运行结果应为R0=1112 R1=1010。

(2) 测试JMPR指令：

在命令行提示符状态下输入：

A 2020✓

屏幕将显示：

2020:

从地址2020开始输入下列程序：

2020: MVRD R2,000D ； 给R2赋值000D，000D为回车键的ASCII码值

2022: IN 81 ； 判键盘上是否按了一个键，

2023: SHR R0 ； 即串行口是否有了输入的字符

2024: SHR R0

2025: JRNC 2022 ； 没有输入则循环测试

2026: IN 80 ； 输入字符到R0低位字节

2027: MVRD R1, 00FF

2029: AND R0, R1 ； 清零R0的高位字节内容

202A: CMP R0,R2 ； 判断输入字符是否为回车

202B: JRZ 2030 ; 若是转向程序结束地址

202C: OUT 80 ; 若否输出键入字符

202D: MVRD R3, 2022

202F: ✓

在命令行提示符状态下输入:

A 2030 ✓

2030: RET

2031: ✓

用E命令输入JMP R3的代码, 在命令行提示符状态下输入:

E 202F ✓

202F: 6003

2030: ✓

用G命令运行前面刚键入源程序, 在命令行提示符状态下输入:

G 2020 ✓

光标闪烁等待键盘输入, 若输入非回车字符, 则在屏幕上回显; 若输入回车字符, 则程序执行结束。

(3) 测试JRS指令:

在命令行提示符状态下输入:

A 2100 ✓

屏幕将显示:

2100:

从地址2100H开始输入下列程序:

2100: MVRD R1, 0000 ; 给R1赋值0000

2102: MVRD R2, 4040 ; 给R2赋值4040

2104: MVRD R3, 01FF ; 给R3赋值01FF

2106: ADD R2, R3 ; R2 和R3相加

*2107: JRS 210E ; 判第一位, 若为1, 向后跳6个单元

2108: MVRD R0, 0030 ; 给R0赋字符“0”

210A: OUT 80 ; 输出该字符

210B: INC R3 ; R3加1

210C: INC R1 ; R1加1

210D: JR 2106 ; 跳到2106循环执行

210E: MVRD R0, 0031 ; 给R0赋字符“1”

2110: OUT 80 ; 输出该字符

2111: RET

注意: *表示扩展指令JRS应用E命令键入, 在命令行提示符状态下输入:

E 2107 ✓

2107: 6406 ; 06为偏移量, 该值是要转向的地址值减去JRS下一条指令的地址得出的。

用G命令运行前面刚键入源程序, 在命令行提示符状态下输入:

G 2100 ✓

屏幕显示字符0001。

用R命令看寄存器的内容，在命令行提示符状态下输入：

R ✓

屏幕回显15个寄存器的值，其中R1的值表示R3加1的次数。

可改变R2、R3的值观察程序运行结果。以加强对该条指令的理解。

(4) 测LDRX、STRX指令

例1：测LDRX指令

1) 在命令行提示符状态下输入：

A 2080

屏幕将显示：

2080:

从地址2080H开始输入下列程序：

2080: MVRD R2, 2000 ; 给寄存器R2赋值2000

*2082: LDRX R1, 0020[R2] ; 将寄存器R2的内容与偏移量相加，相加的和为内存单元2020，将该单元的内容赋给R1

*2084: JMPR R1 ; 跳转到寄存器R1所示的内存单元

2085: MVRD R0, 0030 ; 将字符‘0’的ASCII码值赋给R0

2087: OUT 80 ; 输出该字符

2088: RET

2089: ✓

注意：扩展指令LDRX、JMPR 必须用E命令键入，形式为：E 2082

2082 原值：E512 （空格） 原值：0020（空格）原值：6001 ✓

2) 在命令行提示符状态下输入：

E 2020

屏幕将显示：

2020 内存单元原值：-

在光标处输入2100

3) 在命令行提示符状态下输入：

A 2100

屏幕将显示：

2100:

从地址2100H 开始输入下列程序：

2100: MVRD R0, 0036 ; 将字符‘6’的ASCII码值赋给R0

2102: OUT 80 ; 输出该字符

2103: RET

2104: ✓

4) 在命令行提示符状态下输入：

G 2080 ✓

屏幕回显数字6。

例2：测STRX指令

- 1) 在命令行提示符状态下输入：

A 2000

屏幕将显示：

2000:

从地址2000H 开始输入下列程序：

2000:MVRD R1,6666

2002:MVRD R2,2000

*2004:STRX R1, 0080[R2]

2006:RET

2007: ✓

扩展指令STRX是按如下格式输入的：

在命令行提示符状态下输入：

E 2004 ✓

2004 内存单元原值：E612（空格） 内存单元原值：0060 ✓

- 2) 在命令行提示符状态下输入：

G 2000 ✓

执行输入的程序。

- 3) 在命令行提示符状态下输入：

D 2060 ✓

可以观察到2060内存单元的值为6666，表明寄存器R1中的内容已放入该内存单元,内存单元的值是由寄存器R2中的内容和偏移量0060相加得到的。

五、实验要求与实验报告内容：

- 1、实验之前认真预习，明确实验的目的和具体实验内容，认真地学懂几条典型指令的在执行步骤划分和每一个执行步骤（每一个节拍）使用的控制信号的取值等有关内容，对硬连线控制器的组成和运行原理有一个初步的理解。
- 2、参照对已有指令的执行步骤和各组控制信号的设计结果，设计待扩展的几条指令的相关内容，要特别注意属于同一组的指令在执行步骤划分、每一个节拍中所用到的控制信号取值的相似性和差异之处，有意识地加深硬连线控制器设计方法的理解，做好实验准备。
- 3、想一想实验的操作步骤，明确可以通过实验学习得到哪些知识，想一想怎么样有意识地提高教学实验的真正效果；在教学实验过程中，要爱护教学实验设备和用到的辅助仪表，记录实验步骤中的数据和运行结果，仔细分析遇到的现象与问题，找出解决问题的办法，有意识地提高自己创新思维能力。
- 4、实验之后认真写出实验报告，重点在于预习时准备的内容，最终正确的节拍发生器的逻辑表达式，控制信号产生部件的逻辑表达式，实验过程、遇到的现象和解决问题的办法，自己的收获体会，对改进教学实验安排的建议等。

实验五 微程序控制器部件教学实验

一、实验目的：

- 1、全面掌握计算机各部件组成及相互联接关系。
- 2、深入理解计算机微程序控制器的功能、组成知识。
- 3、深入地学习计算机各类典型指令的执行流程。
- 4、对指令格式、寻址方式、指令系统、指令分类等建立具体的总体概念。
- 5、学习微程序控制器的设计过程和相关技术。

二、实验内容：

- 1、完成控制器部件的教学实验，主要内容是自己设计几条指令的功能、格式和执行流程，并在教学计算机上实现、调试正确。
- 2、首先是看懂TEC-XP+教学计算机的功能部件组成和线路逻辑关系，然后分析教学计算机中已经设计好并正常运行的几条典型指令（例如，ADD、MVRR、OUT、MVRD、JRC、CALA、RET等指令）的功能、格式和执行流程，注意各操作功能所对应的控制信号的作用。
- 3、设计几条指令的功能、格式和执行流程，并在教学计算机上实现、调试正确。例如ADC、JRS、JRNS、LDRA、STAR、CALR等指令，可以从给出的19条扩展指令中任意选择，当然也可以设计与实现其它的指令，包括原来已经实现的基本指令（要变换为另外一个指令操作码）或自己确定的指令。
- 4、单条运行指令，查看指令的功能、格式和执行流程。
- 5、用监控程序的A、E（扩展指令必须用E 命令置入）命令编写一段小程序，观察运行结果。

三、预习要求：

- 1、确定指令格式和操作码，找出其微程序入口地址。
- 2、确定实现方案，确定微程序流程图。
- 3、编写微码，用16进制数表示。
- 4、确定测试方案，编写微码装入和新指令测试程序，准备测试数据、写出测试步骤。

四、实验说明：

控制器设计是学习计算机总体组成和设计的最重要的部分。要在TEC-XP+教学计算机完成这项实验，必须比较清楚地懂得：

- 1、TEC-XP+教学机的微程序控制器主要由微程序定序器AM2910、产生当前微地址和下地址的微控存和MACH 器件组成。
- 2、TEC-XP+教学机上已实现的全部基本指令和留给用户实现的19条扩展指令的控制信号都是由微控存和MACH给出的。
- 3、应了解监控程序的A命令只支持基本指令，扩展指令应用E命令将指令代码写入到相应的存储单元中；不能用T、P命令单步调试扩展指令，只能用G命令执行扩展指令。
- 4、要明白TEC-XP+教学机支持的指令格式及指令执行流程分组情况；理解TEC-XP+教学机中已经设计好并正常运行的各类指令的功能、格式和执行流程，也包括控制器设计与实现中的具体线路和控制信号的组成。
- 5、要明确自己要实现的指令格式、功能、执行流程设计中必须遵从的约束条件。

为了完成自己设计几条指令的功能、格式和执行流程，并在教学计算机上实现、调试正确的实验内容，具体过程包括：

(1)、确定指令格式和功能，包括确定要用的操作码，指令中其它字段的内容分配与使用。指令中字段的使用和分配要受教学机已有硬件的约束，应尽量与已实现指令的格式和分类办法保持一致。

(2)、按新指令的功能和格式，设计指令的执行流程。划分指令执行步骤并设计每一步的执行功能，设计微地址和下地址的取值，应参照已实现指令的处理办法来完成。

(3)、在指令流程表中填写每一个控制信号的状态值，基本上是个查表填数的过程，应该特别仔细，并有意识地体会这些信号的控制作用。

(4)、将设计好的微码，装入控制存储器的相应单元。

(5)、写一个包含你设计的指令的程序，通过运行该程序检查执行结果的正确性，来初步判断你的设计是否正确；如果有问题，通过几种办法查出错误并改正，继续调试，直到完全正确。

五、实验步骤：

1、接通教学机电源。

2、将教学机左下方的六个拨动开关置为110100（单步、手动置指令、微程序、联机、16位、MACH）。

3、按一下“RESET”按键。

4、通过16 位的数据开关SWH、SWL置入指令操作码。

5、在单步方式下，通过指示灯观察各类基本指令的微码。

(1) 选择基本指令的A组指令中的ADD指令，观察其节拍流程

1) 置拨动开关SW=00000000 00000001；（表示指令ADD R0, R1 ）

2) 按RESET按键； 指示灯Microp亮（只要选择微程序，该灯在指令执行过程中一直亮），其它灯全灭；

3) 按START按键； 指示灯CI3~0、SCC3~0显示1110 0000，微址和下址的指示灯全灭；
（本拍完成公共操作0→PC、DI#=0）

4) 按START按键； 指示灯CI3~0、SCC3~0显示1110 0000，微址指示灯显示0000 0001，下址的指示灯全灭；（本拍完成公共操作PC→AR、PC+1→PC）

5) 按START按键； 指示灯CI3~0、SCC3~0显示1110 0000，微址指示灯显示0000 0010，下址的指示灯全灭；（本拍完成公共操作MEM→IR）

6) 以上三步为公共操作，其它指令同；

7) 按START按键； 指示灯CI3~0、SCC3~0显示0010 0000，微址指示灯显示0000 0011，下址的指示灯显示0000 0100；（本拍完成/MAP操作功能）

8) 按START按键； 指示灯CI3~0、SCC3~0显示0011 0000，微址指示灯显示0000 0100，下址的指示灯显示0011 0000 (本拍执行ADD指令，DR←DR+SR 操作)。

9) 按START按键； 指示灯CI3~0、SCC3~0显示0011 0010，微址指示灯显示0011 0000，下址的指示灯显示0011 1010；（本拍完成STR→Q、CC#=INT#公共操作功能）

10) 按START按键； 指示灯CI3~0、SCC3~0 显示0011 0000，微址指示灯显示0011 0001，下址的指示灯显示0000 0010；（本拍完成PC→AR、PC+1→PC、CC#=0的公共操作）

(2) 选择基本指令的B组指令中的MVRD指令，观察其节拍流程

(3) 选择基本指令的D组指令中的CALA指令，观察其节拍流程

这里(2)和(3)部分的节拍流程由学生自己完成，不再列出。

6、在连续方式下，用A命令键入程序并运行（程序由基本指令组成，可直接用A命令键入）。

（程序自选，可选择实验一中的程序）

7、设计几条指令的功能、格式和执行流程，设计每条微指令各字段的具体编码值，包括控制码的各字段、下地址字段、形成下址用到的条件码。

可以从给出的19条扩展指令中任意选择，例如ADC、STC、LDRA、CALR等指令，当然也可以设计与实现其它的指令，包括原来已经实现的基本指令（要变换为另外一个指令操作码）或自己确定的指令。

(1) 扩展几条指令，确定各步的控制信号。

指令	操作功能	微址	下址	CI3~0	SCC3~0	MRW	I2~0	I8~6	I5~3	B口	A口	SST	SSH Sci	DC2	DC1
ADC	DR+SR+ CF→DR	50	30	0011	0000	100	001	011	000	0000	0000	001	010	000	000
STC	STC	57	30	0011	0000	100	000	001	000	0000	0000	100	000	000	000
LDRA	PC→AR PC+1→ PC	5B	00	1110	0000	100	011	010	000	0101	0101	000	001	011	000
	MEM→ AR	5C	1C	0011	0000	001	111	001	000	0000	0000	000	000	011	000
CALR	SP-1 → SP、AR	64	00	1110	0000	100	011	011	001	0100	0000	000	000	011	000
	PC→ MEM	65	00	1110	0000	000	100	001	000	0000	0101	000	000	000	001
	SR→PC	66	30	0011	0000	100	100	011	000	0101	0000	000	000	000	000

注意：在做扩展指令时，控制信号要由MACH产生。

(2) 将扩展好的控制信号添加到给出的MACH程序中，编译生成JED的熔丝图文件，写入MACH内的寄存器中。(出厂时默认的程序就是微程序全指令的，所以这个步骤可以省略，直接进行下面的步骤)

8、在单步方式下，通过指示灯观察各类扩展指令的微码。

(1) 选择扩展指令的A 组指令中的ADC 指令，观察其节拍流程

- 1) 置拨动开关SW=00100000 00000000；（表示指令ADC ）
- 2) 按RESET 按键； 指示灯Microp 亮（只要选择微程序，该灯在指令执行过程中一直亮），其它灯全灭；
- 3) 按START 按键； 指示灯CI3~0、SCC3~0 显示1110 0000，微址和下址的指示灯全灭；
- 4) 按START 按键； 指示灯CI3~0、SCC3~0 显示1110 0000，微址指示灯显示0000 0001，下址的指示灯全灭；
- 5) 按START 按键； 指示灯CI3~0、SCC3~0 显示1110 0000，微址指示灯显示0000 0010，下址的指示灯全灭；
- 6) 以上三步为公共操作，其它指令同。
- 7) 按START 按键； 指示灯CI3~0、SCC3~0 显示0010 0000，微址指示灯显示0000 0011，下址的指示灯显示0101 0000；
- 8) 按START 按键； 指示灯CI3~0、SCC3~0 显示0011 0000，微址指示灯显示0101 0000，下址的指示灯显示0011 0000；（本拍完成DR+SR+CF→DR 操作）
- 9) 按START 按键； 指示灯CI3~0、SCC3~0 显示0011 0010，微址指示灯显示0011 0000，

下址的指示灯显示0011 1010；（本拍完成STR→Q、CC#=INT#操作）

- 10) 按START 按键； 指示灯CI3~0、SCC3~0 显示0011 0000，微址指示灯显示0011 0001，下址的指示灯显示0000 0010；（本拍完成PC→AR、PC+1→PC 操作）

(2) 选择扩展指令的C 组指令中的LDRA 指令，观察其节拍流程

(3) 选择扩展指令的C 组指令中的CALR 指令，观察其节拍流程

这里(2)和(3)部分的节拍流程由学生自己完成，不再列出。

9、用A、E键入程序连续运行（扩展指令用E命令键入）。

将教学机左下方的六个拨动开关置为000100（连续、从内存读指令、微程序、联机、16位、MACH）。

(1) 测试ADC 指令。

1) 在命令行提示符状态下输入：

A 2000✓

屏幕将显示：

2000:

从地址2000H 开始输入下列程序：

2000: MVRD R0,0101 ； 给R0 赋值0101

2002: MVRD R1,1010 ； 给R1 赋值1010

2004: ✓

在命令行提示符状态下输入：

A 2006✓

2006: RET

2007: ✓

用E 命令输入STC、ADC R0,R1 的代码，在命令行提示符状态下输入：

E 2004✓

2004: 6D00

2005: 2001

2006: ✓

2) 用G 命令运行前面刚键入源程序，在命令行提示符状态下输入：

G 2000✓

3) 用R 命令察看寄存器的内容，在命令行提示符状态下输入

R✓

运行结果应为R0=1112 R1=1010。

(2) 测试CALR 指令。

设计一个有读写内存和子程序调用指令的程序,功能是读出内存中的字符，将其显示到显示器的屏幕上，转换为小写字母后再写回存储器原存储区域。

1) 将被显示的6个字符‘A’~‘F’送入到内存20F0H开始的存储区域中。

在命令行提示符状态下输入：

E 20F0✓

屏幕将显示：

20F0 内存单元原值:

按下列格式输入:

20F0 内存原值: 0041 内存原值: 0042 内存原值: 0043

内存原值: 0044 内存原值: 0045 内存原值: 0046✓

2) 在命令行提示符状态下输入:

A 2080✓

从地址2080H 开始输入下列程序:

(2080) MVRD R3, 0006 ; 指定被读数据的个数

MVRD R2, 20F0 ; 指定被读、写数据内存区首地址

(2084) LDRR R0,[R2] ; 读内存中的一个字符到R0寄存器

MVRD R8, 2100 ; 指定子程序地址为2100

*CALR R8 ; 调用子程序, 完成显示、转换并写回的功能

DEC R3 ; 检查输出的字符个数

JRZ 208C ; 完成输出则结束程序的执行过程

INC R2 ; 未完成, 修改内存地址

JR 2084 ; 转移到程序的2086处, 循环执行规定的处理

(208C) RET

从地址2100H 开始输入下列程序:

(2100) OUT 80 ; 输出保存在R0寄存器中的字符

MVRD R1, 0020

ADD R0, R1 ; 将保存在R0中的大写字母转换为小写字母

STRR [R2],R0 ; 写R0 中的字符到内存, 地址同LOD所用的地址

(2105) IN 81 ; 测试串行接口是否完成输出过程

SHR R0

JRNC 2105 ; 未完成输出过程则循环测试

RET ; 结束子程序执行过程, 返回主程序

3) 在命令行提示符状态下输入:

G 2080✓

屏幕显示运行结果为:

ABCDEF

4) 在命令行提示符状态下输入:

D 20F0✓

20F0H~20F5H 内存单元的内容为:

0061 0062 0063 0064 0065 0066

(3) 测试指令LDRA.

编写一程序, 将存放在地址单元2100~2101的内容通过LDRA指令转到寄存器R0输出。

1) 将要输出的字符存放在地址单元2100。

在命令行提示符状态下输入:

E 2100✓

屏幕将显示:

2100 内存单元原值:

按下列格式输入:

2100 内存原值: 0036 内存原值: 0038✓

2) 用A、E 命令键入程序。

在命令行提示符状态下输入:

A 2000✓

从地址2000H 开始输入下列程序:

2000: *LDRA R0, [2100] ; 将内存单元2100 的内容赋给寄存器R0

2002: OUT 80 ; 输出R0 的内容

2003: IN 81 ; 判是否输出完

2004: SHR R0

2005: JRNC 2003 ; 未完, 则循环测试

2006: *LDRA R0, [2101] ; 将内存单元2101 的内容赋给寄存器R0

2008: OUT 80 ; 输出R0 的内容

2009: RET

带*的两条指令为扩展指令, 必须用E 命令键入, 形式如下:

E 2000✓

2000 内存单元原值: E400 内存单元原值: 2100✓

E 2006✓

2006 内存单元原值: E400 内存单元原值: 2101✓

3) 在命令行提示符状态下输入:

G 2000✓

屏幕将回显字符‘6’‘8’

问题: 在“A”命令下能否直接输入新指令? 为什么?

要求学生参照以上过程从给出的19条扩展指令中任意选择几条指令, 包括原来已经实现的基本指令(要变换为另外一个指令操作码)或自己确定的指令。

六、本次实验报告要求:

本实验为综合设计实验, 实验报告的内容要全面。实验报告要求如下:

- 1、写出实验目的、实验器材。
- 2、实验内容包括新指令的格式及相应的微程序。
- 3、实验步骤要写出新指令的装入、运行及测试过程, 特别要说明测试过程。

实验六 输入/输出接口扩展实验

一、实验目的：

- 1、了解串行接口与计算机主机之间的数据传送方式。
- 2、理解串行接口芯片的内部组成和传送数据的运行过程。
- 3、了解串行接口在投入运行之前必须执行的初始化操作的作用及完成初始化操作的具体方案。

二、预习要求：

- 1、查阅有关书籍，了解串行通信接口芯片8251的工作原理。
- 2、了解8251复位、初始化、数据传输的过程。
- 3、阅读实验中的汇编语言程序。
- 4、拟订好实验步骤和测试手段。

三、实验内容：

- 1、为扩展I/O口选择一个地址，即将与COM2口相连的8251的/CS与标有I/O /CS的一排插孔中的一个相连。
- 2、将COM2口与终端或另一台运行有PCEC16的PC机的串口相连。
- 3、用监控程序的A命令，编写一段小程序，先初始化COM2口，再向COM2口发送一些字符，也可从COM2口接收一些字符，或实现两个串口的通信。

四、实验步骤：

- 1、为扩展I/O口选择一个地址：将与COM2口相连的8251的/CS与标有I/O /CS的插孔中地址为A0~AF的一个相连。
- 2、将一台教学机COM1口与一台PC机相连，在PC机上启动PCEC16.EXE。
- 3、将另一台教学机COM1口与另一台PC机相连，同样启动PCEC16.EXE。
- 4、用一根串口线将第一台的教学机的COM2口和另一台教学机的COM2口相连。
- 5、接通教学机电源。
- 6、将教学机左下方的6个拨动开关置为001100（连续、从内存读指令、组合、联机、16 位、MACH）。
- 7、按一下“RESET”按键，接着按“START”按键。
- 8、在两台PC 机对应的PCEC 上分别输入一下程序：

从2000H 单元开始输入下面的程序

2000: MVRD R0, 004E; 给R0赋值004E

2002: OUT A1 ; 将R0 的值输出到COM2口的8251中的寄存器中

2003: MVRD R0, 0037; 给R0 赋值0037

2005: OUT A1 ; 将R0 的值输出到COM2口的8251中的寄存器中

2006: IN 81 ; 检查本机键盘是否按了一个键,

2007: SHR R0 ; 即串行口是否有了输入的字符

2008: SHR RO

2009: JRNC 200D ; 没有, 则转去检查扩展接口的键盘有没有输入

200A: IN 80 ; 若本机键盘有输入则接收该字符

200B: OUT 80 ; 将键盘输入的字符在本机输出

200C: OUT A0 ; 将从键盘输入的字符输出经扩展串口送到另一台教学机输出

200D: IN A1 ; 检查扩展串口相连的另一台教学机对应的PC键盘上是否按键,

200E: SHR RO ; 即串行口是否有了输入的字符

200F: SHR R0

2010: JRNC 2006 ; 没有, 则转去判本机键盘是否有输入

2011: IN A0 ; 若有, 则接收

2012: OUT 80 ; 在本机输出

2013: JR 2006

2014: RET

该程序完成两台教学计算机的第2个串行接口扩展操作并完成该串口初始化, 启动两台教学机, 都运行这个程序, 则两个键盘的输入同时显示在两个屏幕上, 实现的是双机的双向通讯功能。

注意: 每台教学机都只能检查与操作自己的串行口, 管不了另外那台教学机。

问题: 请画出该程序的流程图。

五、实验要求与实验报告内容:

1、实验之前认真预习, 明确实验的目的和具体实验内容, 设计好实验用到的有关程序, 做好实验之前必要的准备工作。

2、查阅教学计算机的图纸, 看清串行接口线路的正确的连接关系, 想清楚需要实现跳线的连接关系。

3、想一想实验的操作步骤, 明确可以通过实验学习到哪些知识, 想一想怎么样有意识地提高教学实验的真正效果; 在教学实验过程中, 要爱护教学实验设备和用到的辅助仪表, 记录实验步骤中的数据和运行结果, 仔细分析遇到的现象与问题, 找出解决问题的办法, 有意识地提高自己创新思维能力。

4、实验之后认真写出实验报告, 重点在于预习时准备的内容, 实验中用到的程序和它的运行结果, 实验过程、遇到的现象和解决问题的办法, 自己的收获体会, 对改进教学实验安排的建议等。

实验七 中断实验

一、实验目的：

- 1、加深理解计算机系统中断的工作原理及处理过程。
- 2、学习和掌握中断产生、响应、处理等技术。
- 3、掌握中断服务子程序的编写要点，进行一次硬、软件的综合调试。

二、实验内容：

- 1、扩展中断隐指令，为中断隐指令分配节拍。中断隐指令用到12个节拍，为了和一般指令相区别，应将其节拍T3设计为1。

注意：在扩展中断隐指令时要用到DC1、DC2的译码信号。

- 2、扩展开中断指令EI、关中断指令DI、中断返回指令IRET。
- 3、确定中断向量表地址。中断向量的高12位由开关确定为（0010 0100 0000）。三级中断对应的中断向量为2404H、2408H、240CH。当有中断请求且被响应后，将执行存放在该中断的中断向量所指向的内存区的指令。
- 4、填写中断向量表。在上述的2404H、2408H、240CH地址写入三条JR转移指令，JR指令的OFFSET是偏移量，其值是要转向的地址的值减去该条转移指令的下一条指令的地址的值得到的，该值的范围在-128~+127之间。但在PCEC16中输入时，用户不需要计算偏移量，直接输入要转向的绝对地址即可。
- 5、编写中断服务程序。中断服务程序可以放在中断向量表之后，中断服务程序可实现在程序正常运行时在计算机屏幕上显示与优先级相对应的不同字符。
- 6、写主程序。可编写一死循环程序，要求先开中断。

三、实验步骤：

1. 扩展中断隐指令和开、关中断指令、中断返回指令，为他们分配节拍并给出各节拍对应的控制信号。

(1) 中断隐指令的节拍和控制信号

节拍	功能说明	/M I O	R E Q	/W E	A	B	Sci	SSH	I8- 6	I5- 3	I2- 0	SST	DC1	DC2
1010	STR→Q. DI#=0	1	0	0	0000	0000	00	00	000	000	111	000	011	111
1110	SP-1→ AR、SP	1	0	0	0000	0000	00	00	011	001	011	000	000	011
1100	PC→MEM、 INTN#	0	0	0	0101	0000	00	00	001	000	100	000	001 1	101
1111	SP-1→ AR、SP	1	0	0	0000	0100	00	00	011	001	011	000	000	011
1101	Q→MEM	0	0	0	0000	0000	00	00	001	000	010	000	001	000
1011	INTVL#、 IB→PC	1	0	0	0000	0101	00	00	011	000	101	000	101	000

(2) EI、DI、IRET 指令的节拍和控制信号

节拍	指令	编码	/M IO	RE Q	/W E	A	B	Sc i	SS H	I8- 6	I5- 3	I2- 0	SST	DC1	DC2
1000			1	0	0	0101	0101	01	00	011	001	001	000	000	111
0000			1	0	0	0101	0101	01	00	010	000	011	000	000	011
0010			0	0	1	0000	0000	00	00	001	000	000	000	000	001
0011	EI	0110 1110	1	0	0	0000	0000	00	00	001	000	000	000	110	000
	DI	0110 1111	1	0	0	0000	0000	00	00	001	000	000	000	111	000
0110	IR ET	1110 1111	1	0	0	0100	0100	01	00	010	000	011	000	000	011
0100	IR ET	1110 1111	0	0	1	0000	0000	00	00	001	000	000	010	000	000
0111	IR ET	1110 1111	1	0	0	0100	0100	01	00	010	000	011	000	000	011
0101	IR ET	1110 1111	0	0	1	0000	0101	00	00	011	000	111	000	000	000

2、任意选择几条指令观察指令执行及转中断执行的节拍和各节拍对应的控制信号。

置控制开关为111100（单步、手动置指令、组合逻辑、联机、16位、MACH）

(1)选择基本指令的A组指令中的SHR 指令，观察其节拍流程：

- 1) 置拨动开关SW=00001011 00000000；（表示指令SHR R0 ）
- 2) 按RESET按键；节拍指示灯T3～T0显示1000；(本拍在第1次复位后才会执行)
- 3) 按START按键；节拍指示灯T3～T0显示0000；(以上两拍,为公共节拍，在手动置指令方式下无意义)
- 4) 按START按键；节拍指示灯T3～T0显示0010；(本拍也是公共节拍，将指令编码写入指令寄存器IRH、IRL)
- 5) 按START按键；节拍指示灯T3～T0显示0011；(本拍执行SHR指令，完成SHR R0操作)
- 6) 按START按键；节拍指示灯T3～T0显示0000；(本拍转中断处理,完成DI#操作)
- 7) 按START按键；节拍指示灯T3～T0显示0010；(本拍完成中断处理操作STR→Q)
- 8) 按START按键；节拍指示灯T3～T0显示0110；(本拍完成中断处理操作SP-1→AR、SP)
- 9) 按START按键；节拍指示灯T3～T0显示0100；(本拍完成中断处理操作PC→（AR），INTN#)
- 10) 按START按键；节拍指示灯T3～T0显示0111；(本拍完成中断处理操作SP-1→AR、SP)
- 11) 按START按键；节拍指示灯T3～T0显示0101；(本拍完成中断处理操作Q→（AR）)
- 12) 按START按键；节拍指示灯T3～T0显示1000；(本拍完成中断处理操作INTV#，IB→PC)

可以看到，A组指令（包括ADD、SUB、CMP、AND、XOR、SHR、SHL、INC、DEC、TEST、OR、MVRR、JR、JRC、JRNc、JRZ、JRNZ）的执行共11拍，包括3拍公共节拍，一拍完成指令操作，另有7拍转中断处理。

(2) 选择基本指令的B组指令中的IN 指令，观察其节拍流程：

- 1) 置拨动开关SW=10000010 00000000；（表示指令IN 00 ）
- 2) 按RESET按键；节拍指示灯T3～T0显示1000；(本拍在第1次复位后才会执行)
- 3) 按START按键；节拍指示灯T3～T0显示0000；(以上两拍,为公共节拍，在手动置指令方式下无意义)
- 4) 按START按键；节拍指示灯T3～T0显示0010；(本拍也是公共节拍，将指令编码写入指令寄存器IRH、IRL)
- 5) 按START按键；节拍指示灯T3～T0显示0110；（本拍执行IN指令的第一步IRL→IB→AR）
- 6) 按START按键；节拍指示灯T3～T0显示0100；（本拍执行IN指令的第二步，（PORT）→R0）

7) 以下7拍转中断处理, 节拍流程与A组指令的相同。

可以看到, B组指令(包括JMPA、LDRR、IN、STRR、PSHF、PUSH、OUT、POP、MVRD、POP、RET)的执行除3拍公共节拍外,指令完成需两步, 转中断处理后执行7 步。

(3) 选择基本指令的D组指令中的CALA指令, 观察其节拍流程:

- 1) 置拨动开关SW=11001110 00000000; (表示指令CALA)
- 2) 按RESET按键; 节拍指示灯T3~T0显示1000; (本拍在第1次复位后才会执行)
- 3) 按START按键; 节拍指示灯T3~T0显示0000; (以上两拍,为公共节拍, 在手动置指令方式下无意义)
- 4) 按START按键; 节拍指示灯T3~T0显示0010; (本拍也是公共节拍, 将指令编码写入指令寄存器IRH、IRL)
- 5) 按START按键; 节拍指示灯T3~T0显示0110 (本拍执行CALA指令的第一步, PC→AR、PC+1→PC)
- 6) 按START按键; 节拍指示灯T3~T0显示0100; (本拍执行CALA指令的第二步, (AR)→Q)
- 7) 按START按键; 节拍指示灯T3~T0显示0111; (本拍执行CALA指令的第三步, SP-1→SP、AR)
- 8) 按START按键; 节拍指示灯T3~T0显示0101; (本拍执行CALA指令的第四步, PC→(AR), Q→PC)
- 9) 以下7拍转中断处理, 节拍流程与A、B组指令的相同。

(4) 选择扩展指令的A组指令中的SBB指令, 观察其节拍流程:

- 1) 置拨动开关SW=00100001 00000000; (表示指令SBB)
- 2) 按RESET按键; 节拍指示灯T3~T0显示1000; (本拍在第1次复位后才会执行)
- 3) 按START按键; 节拍指示灯T3~T0显示0000; (以上两拍,为公共节拍, 在手动置指令方式下无意义)
- 4) 按START按键; 节拍指示灯T3~T0显示0010; (本拍也是公共节拍, 将指令编码写入指令寄存器IRH、IRL)
- 5) 按START按键; 节拍指示灯T3~T0显示0011; (本拍执行SBB指令, DR-SR-/C→DR)
- 6) 以下7拍转中断处理, 节拍流程与基本指令的A、B组指令相同

(5) 选择扩展指令的C组指令中的LDRA指令, 观察其节拍流程:

- 1) 置拨动开关SW=11100100 00000000; (表示指令LDRA)
- 2) 按RESET按键; 节拍指示灯T3~T0显示1000; (本拍在第1次复位后才会执行)
- 3) 按START按键; 节拍指示灯T3~T0显示0000; (以上两拍,为公共节拍, 在手动置指令方式下无意义)
- 4) 按START按键; 节拍指示灯T3~T0显示0010; (本拍也是公共节拍, 将指令编码写入指令寄存器IRH、IRL)
- 5) 按START按键; 节拍指示灯T3~T0显示0110; (本拍执行LDRA指令第一步, 完成操作PC→AR, PC+1→PC)
- 6) 按START按键; 节拍指示灯T3~T0显示0111; (本拍执行LDRA指令第二步, 完成操作(AR)→AR)
- 7) 按START按键; 节拍指示灯T3~T0显示0101; (本拍执行LDRA指令第三步, 完成操作(AR)→AR)
- 8) 以下7拍转中断处理, 节拍流程与扩展指令的A组指令相同

(6) 选择扩展指令的D组指令中的IRET 指令, 观察其节拍流程:

- 1) 置拨动开关SW=11101111 00000000; (表示指令IRET)
 - 2) 按RESET按键; 节拍指示灯T3~T0显示1000; (本拍在第1次复位后才会执行)
 - 3) 按START按键; 节拍指示灯T3~T0显示0000; (以上两拍,为公共节拍,在手动置指令方式下无意义)
 - 4) 按START按键; 节拍指示灯T3~T0显示0010; (本拍也是公共节拍,将指令编码写入指令寄存器IRH、IRL)
 - 5) 按START按键; 节拍指示灯T3~T0显示0110; (本拍执行IRET指令第一步,完成操作 $SP \rightarrow AR$, $SP+1 \rightarrow SP$)
 - 6) 按START按键; 节拍指示灯T3~T0显示0100; (本拍执行IRET指令第二步,完成操作 $(AR) \rightarrow FLAG$)
 - 7) 按START按键; 节拍指示灯T3~T0显示0111; (本拍执行IRET指令第三步,完成操作 $SP \rightarrow AR$, $SP+1 \rightarrow SP$)
 - 8) 按START按键; 节拍指示灯T3~T0显示0101; (本拍执行IRET指令第四步,完成操作 $(AR) \rightarrow PC$)
 - 9) 以下7拍转中断处理,节拍流程与扩展指令的A、D组指令相同。
- (7) 选择几条指令INC、JRS、MOVRD、LDRX、STRX、CALA、IRET,单步运行,观察其各节拍对应的控制信号。

3、将教学机左下方的6个拨动开关置为000100(连续、从内存读指令、微程序、联机、16位、MACH)。接着,按一下“RESET”按键,接着按“START”按键。

4、填写中断向量表。

- (1) 将数据开关的高12位设置成: 0010 0100 0000,即选择3级中断的中断向量为2404H、2408H、240CH。
- (2) 中断向量一共有16位,高12位由数据开关的SWH 7-0以及SWL 7-4决定;后四位为 P_1P_000 , P_1P_0 由按下的无锁按键(中断源)决定,分别为01、10、11,所以中断向量的16位为2404、2408、240C。
- (3) 填写中断向量表:

从2404H单元开始输入下面的程序

(2404) JR 2420 ; 跳转到中断服务程序

(2408) JR 2430 ; 跳转到中断服务程序

(240C) JR 2440 ; 跳转到中断服务程序

5、编写中断服务程序。

该中断服务程序,先开中断,显示字符“BI”和对应的中断优先级“1”、“2”或“3”后,等待从键盘输入一个字符。在键盘输入一个字符后,显示该字符和字符“EI”,然后退出当前中断服务程序,返回中断断点,继续执行。

用A、E命令从2420H单元开始输入下面的程序(标有*的语句表示要用E命令输入)

2420: PUSH R0 ; R0进栈

2421: PUSH R3 ; R3进栈

2422: MVRD R3, 31 ; 将字符‘1’的ASCII码送寄存器R3

2424: JR 2450

2430: PUSH R0 ; R0进栈

2431: PUSH R3 ; R3进栈

2432: MVRD R3, 32 ; 将字符‘2’的ASCII码送寄存器R3

2434: JR 2450

2440: PUSH R0 ; R0 进栈

2441: PUSH R3 ; R3 进栈

2442: MVRD R3, 33 ; 将字符‘3’的ASCII码送寄存器R3

2444: JR 2450

*2450:EI ; 开中断（指令编码：6E00）

2451:MVRD R0,0042 ; 将字符“B”赋值给R0, B 即Begin的缩写。

2453:CALA 2200 ; 调用子程序，完成显示

2455:MVRD R0,0049 ; 将字符“T”赋值给R0, I 即Interrupt的缩写。

2457:CALA 2200 ; 调用子程序，完成显示

2459:MVRR R0,R3 ; 将R3的内容送R1

245A:CALA 2200 ; 调用子程序，完成显示

245C:IN 81 ; 判键盘上是否按了一个键

245D:SHR R0 ; 即串口是否有了输入字符

245E:SHR R0

245F:JRNC 245C ; 若没有，等待

2460:IN 80 ; 输入字符到R0

2461:MVRD R0,0045 ; 将字符“E”赋值给R0, E 即End的缩写。

2463:CALA 2200 ; 调用子程序，完成显示

2465:MVRD R0,0049 ; 将字符“T”赋值给R0, I 即Interrupt的缩写

2467:CALA 2200 ; 调用子程序，完成显示

2469:MVRR R0,R3 ; 将R3 的内容送R1

246A:CALA 2200 ; 调用子程序，完成显示

246C:POP R3 ; R3 出栈

246D:POP R0 ; R0 出栈

*246E:IRET ; 中断返回（指令编码：EF00）

6、用A命令从2200H 单元开始输入下面的子程序

2200:PUSH R0 ; R0进栈

2201:IN 81 ; 查询接口状态，判字符输出完成否

2202:SHR R0

2203:JRNC 2201 ; 未完，循环等待

2204:POP R0 ; R0出栈

2205:OUT 80 ; 输出R0的值

2206:RET

7、编写主程序。

从地址2000H 开始输入下列程序：

*2000:EI

2001:MVRD R0, 0036 ; 将字符‘6’的ASCII码送寄存器R0

2003:CALA 2200 ; 输出该字符

2005:MVRD R0,4000 ; 延时子程序

2007:DEC R0

2008:JRNZ 2007

2009:JR 2001 ; 跳到2001循环执行该程序

200A:RET

8、运行主程序，等待、响应中断。

在命令行提示符状态下输入：

G 2000✓

屏幕将连续显示“6”。在程序执行过程中按下教学机右下方任意一个无锁按键。此时，教学机转向执行本级中断服务程序，在屏幕上显示BI以及按下的键对应的中断优先级。在接收键盘一个字符后，退出当前级的中断服务程序，恢复中断现场，接着执行断点处的程序。若在接收字符之前，又有更高一级的中断请求，则教学机转向执行高一级的中断服务程序，执行完后接着执行低级中断，然后退出执行主程序。需要注意的是若当前中断为高级的中断，则不会响应低级中断简单的中断服务程序。

四、实验说明：

1、要求中断隐指令中执行关中断功能，如果用户中断服务程序允许被中断，必须在中断服务程序中执行EI开中断命令。

2、教学机的中断系统共支持三级中断，由三个无锁按键确定从右到左依次为一、二、三级中断，对应的P1、P0的编码分别是01、10、11，优先级也依次升高。这决定了它们的中断向量（即中断响应后，转去执行的程序地址）为XXX4、XXX8、XXXC；可以看到，每级中断实际可用的空间只有四个字节，故这个空间一般只存放一条转移指令，而真正的用户中断服务程序则存放在转移指令所指向的地址。

3、用户需扩展中断隐指令、开中断指令、关中断指令、中断返回指令及其节拍。

实验八 8 位模型机的设计与实现(综合实验)

在 16 位字长教学计算机的主板上,设计与实现一台全新指令系统、全新汇编程序、全新监控程序的 8 位字长的计算机系统是一种可行的选择,这比完成扩展几条新指令的教学实验要难得多,具有更大的挑战性,但完成后的学习收获也更大得多。

一、实验环境、设备与必要技术说明:

TEC-XP16教学计算机系统,可以同时实现16位和8位两种字长,可以同时实现微程序控制器方案和硬连线控制器方案。这两种不同字长的计算机在硬件组成和软件实现方面都是完全不同的系统,甚至于都很难在它们之间考虑彼此的兼容性问题。由于8位字长的教学机硬件资源更为不足,实现8位字长的有一定的完备程度的指令系统就有明显的复杂性,为了使用一定容量(至少是10KB)的存储器系统,地址总线要求是16位,这又给设计8位字长的指令的执行步骤增加更多的工作量。8位机中更短的指令格式和更弱的操作运算功能,使得设计8位机的系统软件(例如监控程序)和应用软件变得更困难。从这个角度讲,如果教师能够在课堂上以16位教学计算机系统为例子,把计算机组成原理和教学机实现中的有关技术讲解清楚,并要求学生在这个基础上,自己动手设计并实现8位字长的教学机,还是有相当多的研究性工作可做,有较多的设计与实现、调试任务可供完成。力争达到学习一门课程和设计并实现一台硬软件组成比较完整的计算机的双重目标,为课程的教学模式改革、培养更具有创新思维能力的高素质人才做出贡献。实现这种教学安排必须具备某些必要条件,就实验设备本身来看,要求可以做到:

- 1、可以配置成8位字长,或16位字长的计算机;指令系统可以完全重新设计。
- 2、可以实现硬连线的控制器,也可以实现微程序的控制器。
- 3、内存储器也有多种不同的设计与运行方式;对中断实验的支持更到位。
- 4、有2路串行接口支持,可以连接计算机终端或PC机仿真终端。
- 5、配备了合理的软件系统:交叉汇编程序、监控程序和PC机仿真终端程序。
- 6、只使用一块印制电路板和220V交流电源,方便携带和在实验室之外使用。
- 7、支持学生在更大程度上完成并实现自己的设计。
- 8、还有适当的辅助设计与模拟软件支持,为提高实验效率和设计质量提供帮助。

二、实验目的、内容和操作步骤:

教学实验的具体安排是,每3名同学一组,占用一台教学计算机,初始配置成16 位字长、同时支持硬连线和微程序两种控制器,已处于正常运行的机型;每组同学合作完成一台硬、软件组成相对完整的全新的8位计算机系统的设计、实现和调试任务。包括设计8位机的运算器、存储器,重点实现微程序控制器,也可以实现硬连线控制器,如果需要和可能,实现8位机的软件系统的修改与完善。

全部的实验任务具体包括:

- 1、通过教学来学习、理解16位的教学计算机硬件系统的总体组成;学习16位教学计算机各功能部件(运算器、控制器、内存储器、总线、接口)的组成与设计,熟悉教学计算机主板上元、器件布局和布线情况,在16位机上完成运算器部件、控制器部件的验证性的教学实验。
- 2、独立构思8位教学计算机各功能部件(运算器、控制器、内存储器、总线、接口)的组成线路逻辑图。
- 3、了解8位教学计算机的监控程序的功能设计和软件实现;8位教学计算机的交叉汇编程序的功能和软件实现;8位教学计算机的PC机仿真终端程序的功能设计和软件实现。

- 4、设计与实现8位机的运算器部件、存储器部件。
- 5、设计与定义8位机的指令系统（每组同学实现的指令系统都不会完全相同）。
- 6、学习指令执行过程，设计8位教学计算机的指令执行流程图。
- 7、确定指令执行过程使用的控制信号。
- 8、设计8位机的微指令格式和相关的微程序的内容。
- 9、完成8位教学计算机的微程序控制下的整机硬件系统的组装和调试任务。
- 10、用硬连线控制器方案实现对前述相同指令系统的控制执行过程。
- 11、如果需要和可能，完成8位教学计算机的软件系统的适应性修改或完善。
- 12、用测试软件对完成的计算机系统进行功能和性能测试，评定成绩。

三、 实验总结报告的内容应包括：

- 1、设计与实现8位字长的教学计算机的任务书和工作进度计划。
- 2、8位字长的教学计算机的总体组成框图和每个部件的线路逻辑图纸。提示，可以通过修改、变动16 位字长的教学计算机的电子版的图纸资料来完成。
- 3、8位字长的教学计算机的运算器部件设计说明。
- 4、8位字长的教学计算机的存储器部件设计说明。
- 5、8位字长的教学计算机的指令系统设计说明。
- 6、8位字长的教学计算机的微程序控制器部件设计说明。
- 7、8位字长的教学计算机的硬连线控制器部件设计说明。
- 8、8位字长的教学计算机的硬件系统调试过程、对所遇到和解决的问题的说明。
- 9、对比8位和16位字长的两种教学计算机的硬件系统中运算器部件、存储器部件的组成、两种控制器部件的组成、使用串行接口的方案、指令的执行步骤、控制器提供的控制信号等各个方面的相同和不同之处。说明在已经学过16为教学机实例的基础上设计8位教学机的过程中需要变动和可以继承的各项内容。
- 10、说明对这一教学实验安排的总体看法，提出自己对这一教学实验安排的建议和批评意见。说明自己的学习和实验中的心得体会。

实验九 FPGA 芯片实现非流水线的 CPU 系统(综合实验)

进入到 21 世纪,随着半导体集成电路的迅猛发展,人们对专用集成电路(ASIC)设计的需求与期望值越来越高,希望能够在单个电路芯片上实现一个系统的全部功能。为此,我们利用 VHDL 语言进行描述,通过 FPGA 门阵列器件硬件实现了一个 16 位字长的 CPU 系统。该系统与存储器和输入输出接口线路相连接,共同组成了一台用于硬件课程教学的完整计算机系统。

新设计与实现的 CPU 在指令系统、使用的软件资源等方面与小规模器件构成的左边的教学计算机系统完全兼容;在硬件构成、实现技术等方面也似。这样可以使软件资源得到充分地应用,减轻研制软件系统的负担,又能更好地在两个系统之间得到尽可能高的可比较性,降低授课难度,提高学生的学习效率。所以新的设计与实现的 CPU 系统的外特性是严格限定了的,它与小规模器件构成 cpu 的教学计算机是严格意义上的同一体系结构的计算机,差别仅表现在计算机的具体实现有所不同,包括选用的器件的类型和集成度不同,所用的设计手段、设计过程有所不同,体现出来的设计与实现技术也不尽相同。

该系统选择了 xilinx 公司的 SPARTAN—II 系列的芯片(型号是 XC2S200),20 万门容量,其内部有 2352 个 CLB,14 个 4Kb 的 RAM 块,208 脚的 PQFP 封装形式,支持在系统编程(in-system programmable),实现了 CPU 的全部功能。在完成这项任务时,已经考虑到如何照顾到现有教材和试验指导书内容的稳定性。首先,需要保证新设计的教学机的指令系统,与过去已经使用的 TEC-XP+教学计算机的指令系统有良好的兼容性。其次,在构思新型教学计算机的逻辑结构的过程中,要向原 TEC-XP+教学计算机的实际组成适当地靠拢,尽量地在二者之间有一个平滑的过渡。这对我们的设计,加进了某些限制条件,但是对减轻任课教师的教学负担、保证教学质量是至关重要的。

当把选用 VHDL 语言来描述的 CPU 的源码文件,经过专用工具软件的编译和综合后,下载到这样一个 FPGA 芯片之中,也就得到了能够正常运行的 CPU 系统。

芯片内部的功能结构图如下:

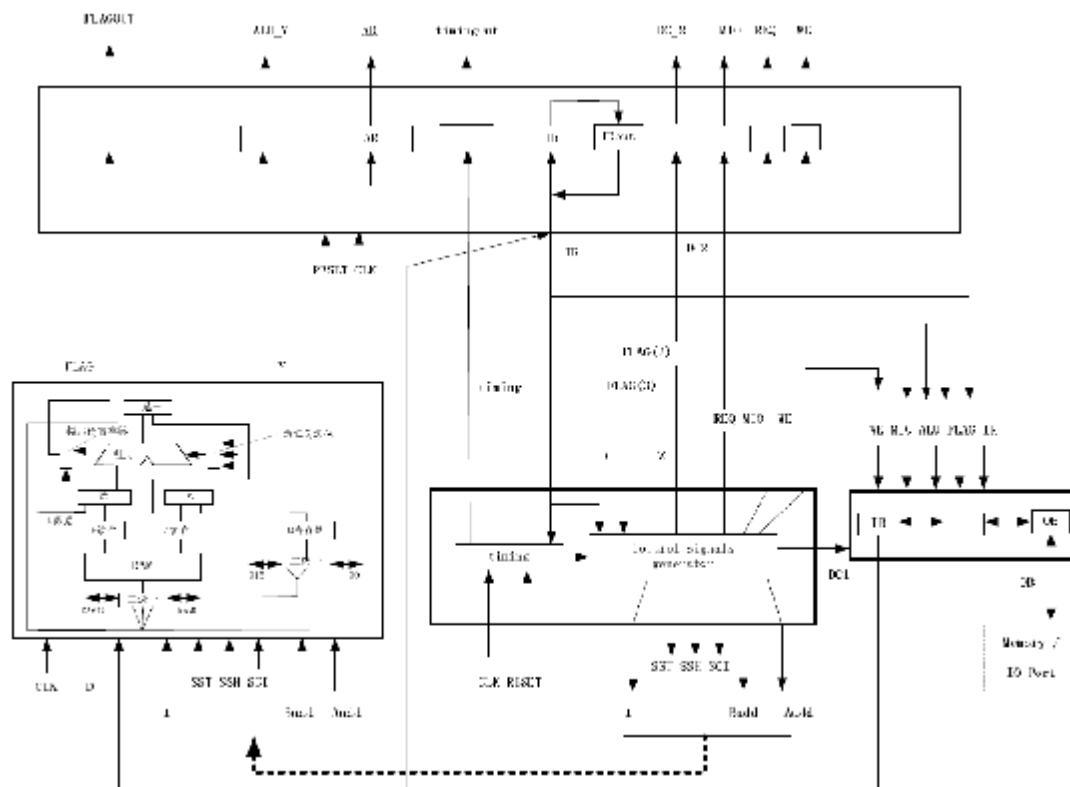


图 2.9.1 FPGA 芯片内部功能结构图

一、实验目的

- 1、进一步熟悉教学计算机的指令格式、指令编码、寻址方式和指令功能等内容；
- 2、进一步熟悉教学计算机的总体组成和各个部件的功能，理解控制器部件在计算机整机中的关键作用；
- 3、进一步理解和熟悉指令执行步骤的划分方案；
- 4、进一步熟悉教学计算机的硬连线控制器各个控制命令（组）的控制功能，学习用 VHDL 语言描述节拍发生器和控制信号产生部件的功能。
- 5、进一步理解与熟悉在 TH-union 教学计算机控制器中处理原有指令和扩展指令的方案，提高对控制器功能描述的理解程度。

二、实验内容

控制器实验可以在两个层次上进行：

第一个层次属于观察、验证性的实验，即通过多种方式，察看教学计算机指令的执行步骤、运行结果、各组控制信号在每一个执行步骤中的状态、指令之间的衔接等有关内容。这个层次的实验，重点在于学懂教学计算机中已有的设计结果，把实现基本指令的 VHDL 语言程序中的语句描述与教学机的运行结果对应清楚。

第二个层次是学生进行自己的设计与实现新的扩展指令的实验，即在教学机系统已有指令的基础上，由学生自己添加若干条（例如 3~5 条）新的指令进去，包括定义指令格式、功能，划分指令执行步骤和确定每一步的功能，确定每一执行步骤使用的全部控制信号的状态值，使用 VHDL 语句把新的设计结果描述正确并添加到已有的源程序代码中去，编译、下载并调试正确，写一个包含已有指令和刚刚实现的指令的小程序，检查运行结果的正确性，若发现错误则找出原因并设法改正，直到全部正确为止。

学生扩展实现哪几条指令，可以由教师指定，也可以由学生根据自己学习情况自选另外一、两条。这些指令最好从教材中给出的扩展指令组中挑选，例如 2 条 A 组指令和 2 条 C 组指令。在设计指令的操

作码编码、指令执行步骤、使用的控制信号等方面，尽可能地参照已有的基本执行的实现办法进行类似的处理，有利于降低实验难度。

三．实验步骤

- 1、按前述的步骤准备好实验机，连接好串口线和电源线，打开 PCEC16.EXE 的仿真界面；
将六个功能开关置为 00X101（连续、内存读指令、联机、16 位、FPGA）；
- 2、确认标有“DataBus 15—8”和“DataBus 7—0”的数据总线的指示灯下方的插针断开；
- 3、确认标有“AdressBus 15-8”和“AdressBus 7-0”的地址总线的指示灯下方的插针断开；
- 4、将提供的带彩线的 FPGA 的下载线并口一端和计算机的并口连接，彩线一端按红色在左边的位置和大板上电源模块下方的一排插针插接好；
- 5、打开实验机的电源；
- 6、在 PC 机上打开 ISE 的软件（软件的具体编译下载使用参见《实验指导和技术说明》）
- 7、打开软件的下载界面，选择 SERIAL 方式，添加器件 CPU.BIT，进行下载；
- 8、下载完成关闭下载界面，启动 PCEC 界面，注意实验机不要断电（FPGA 断电丢失内容）；
- 9、按一下“RESET”按键,再按一下“START”按键，主机上显示：

TEC-2000 CRT MONITOR

Version 1.0 April 2001

Computer Architectur Lab., Tsinghua University

Programmed by He Jia

>

- 10、在 FPGA 构成的 CPU 的控制下将汇编语言程序设计的内容重新作一遍。

附录

附录 1 TEC-XP (FPGA) 计算机硬件系统及其实验内容

随着半导体集成电路技术的迅猛发展,为进行大规模系统的设计和实现带来了新的方法和手段。以 HDL (Hardware Description Language, HDL) 语言表达设计意图、FPGA 作为硬件载体、计算机为设计开发工具、EDA 软件为开发环境的现代电子设计方法日趋完善。为了让学生了解科技发展的前沿技术,感受科学技术的成果,为了顺应最新的计算机设计趋势和教学实验的更高需求,TEC-XP+教学计算机上设置了超大规模集成电路FPGA器件,以满足学生对专用集成电路(ASIC)设计的需求。本章将介绍以TEC-XP+教学计算机为平台、以VHDL语言为设计手段,设计16位字长的CPU系统的基本方法和具体步骤,最后再与存储器 and 输入输出接口线路相连接,共同组成一台用于硬件课程教学目的完整计算机系统的全部过程。

A.1 TEC-XP (FPGA) 教学计算机系统组成概述

TEC-XP(FPGA)是TEC-XP+的一个重要组成部分。它和TEC-XP16系统相对独立,软件互相兼容。TEC-XP (FPGA)和TEC-XP16系统各自的CPU通过总线连接TEC-XP+主板上的内存、接口等构筑了双CPU的TEC-XP+系统。

A. 1. 1 TEC-XP (FPGA) 教学机系统的技术指标与系统配置

TEC-XP (FPGA) 教学机系统的主要技术指标是:

1. 机器字长 16 位 (也可设计成 8 位字长的另外一个新的系统), 即运算器、主存、数据总线、地址总线、指令等都是 16 位。
2. 完整的指令系统被划分为基本指令和扩展指令两部分, 支持多种基本寻址方式。其中的基本指令已经实现, 用于设计监控程序和用户的常规汇编程序, 保留的多条扩展指令供实验者自己实现。
3. 主存最大寻址空间是18K字16位), 基本容量为8K字的ROM和2K字的RAM存储区域。另外的8K字用于完成存储器容量扩展的教学实验。FPGA 芯片和存储器芯片之间可以通过分开的地址总线和分开的数据总线实现连接, 这在实现分开的指令存储器和数据存储器的方案中是必要的。
4. 运算器是参照 Am2901芯片的组成和功能来设计的, ALU实现 8种算术与逻辑运算功能, 内部包括 16个双端口读出、单端口写入的通用寄存器, 和一个能自行移位的乘商寄存器。设置C (进位)、Z (结果为 0)、V (溢出) 和S (符号位) 四个状态标志位。
5. 控制器采用硬连线控制器方案实现, 也可修改成微程序控制器。实验人员可方便地修改已有设计, 或加进若干条自己设计与实现的新指令, 新老指令同时运行。
6. 主机上安装有二路INTEL8251串行接口, 可直接接计算机终端, 或接入一台PC 机作为自己的仿真终端。选用了MAX202倍压线路, 以避免使用+12V 和-12V 电源。
7. 两路的串行接口的接插座安放在机箱后侧板以方便接线插拔和机箱盖的打开关闭。
8. 在主板上设置有一些拨数的开关和微型开关、按键和指示灯, 支持最低层的手工操作方式的 输入/输出和机器调试。

9. 实验机硬件系统,全部功能部件分区域划分在大一些的水平放置的一块印制电路板的不同区域,所有器件都用插座插接在印制板上,便于更换器件。

10. 实验计算机使用单一的5V、最大电流3A 的直流模块电源,所耗电流在1.5~2.5A之间。电 源模块安装在水平电路板右上角位置,交流220V通过电源接线插到机箱后侧板,经保险丝、开关连接到电路板上,开关安放在机箱右侧靠后位置,方便操作且比较安全。

11. 板上安装了很多发光二极管指示灯,用于显示重要的数据或控制信号的状态。

12. 机箱和电路板之间的全部接线都经过接插座,便于整机的生产、调试和维修。

此外还设置了辅助电路和扩展电路两个辅助部分,各个部分被划分在电路板的不同区域。

TEC-XP (FPGA) 系统得总体组成如图A.1所示。由图 A.1 可以看到 TEC-XP (FPGA) 教学 计算机系统也是一个完整的计算机系统,由以下几个基本部分组成:运算器部件、控制器部件、内存储器系统和串行接口线路。

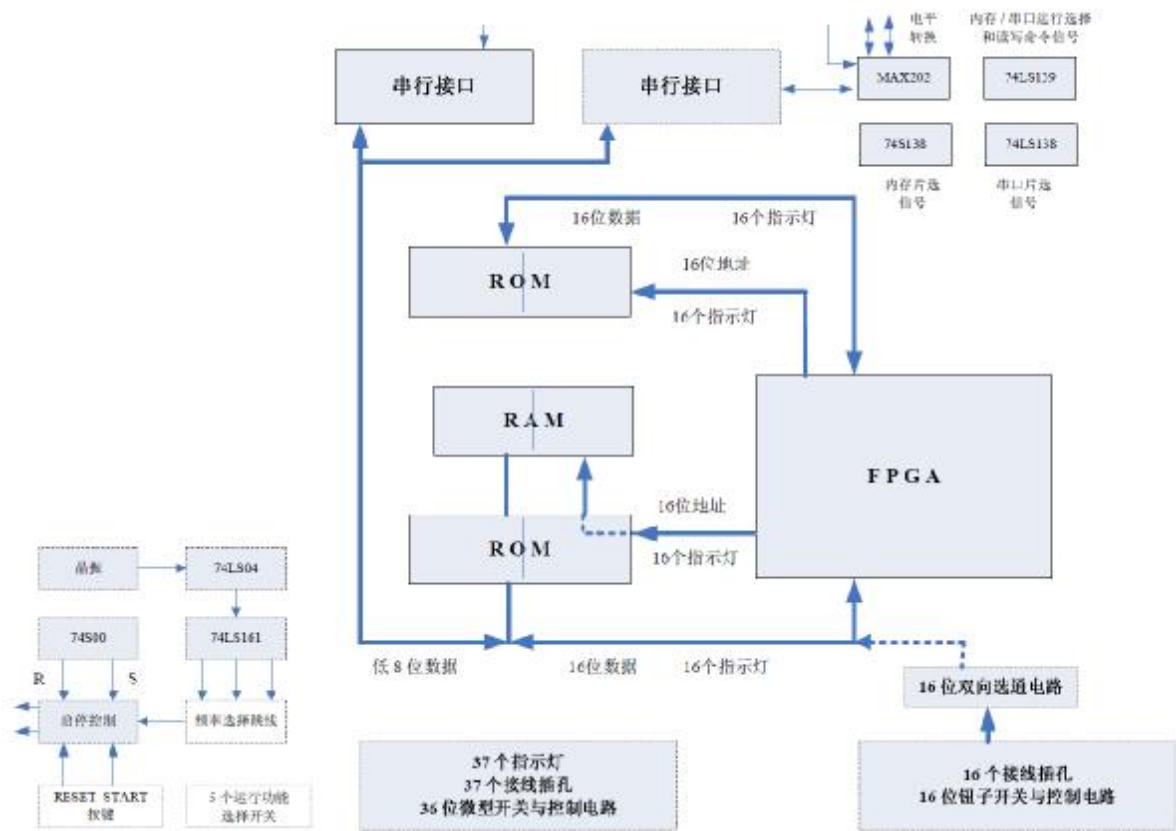


图 A.1 TH-union (FPGA) 实验机系统逻辑图

A. 1. 2 CPU系统设计目标与实现

本小节所说的CPU系统设计,是针对特定的目标、设计特定的CPU系统来展开的。特定的目标,指的是重点针对计算机硬件课程的教学需求,特定的CPU系统,指的是新设计与实现的CPU在指令系统、使用的软件资源等方面尽可能地与TEC-XP 16系统实现完全的兼容;在硬件构成、实现技术等方面也尽可能地与TEC-XP 16系统类似。这样既能使已有的软件资源得到充分地应用,减轻研制软件系统的负担,又能更好地在两个系统之间得到尽可能高的可比较性,保持教材和教学内容的延续性,降低授课难度,提高学生的学习效率。这就意味着,设计与实现的CPU系统的外特性是严格限定在与TEC-XP 16系统是严格意义上的同一体系结构的CPU系统,差别仅表现在CPU具体实现有所不同,包括选用的器件类型和集成度不同、所用设计手段、设计过程有所不同,体现出来的设计与实现技术也不尽相同。

TEC-XP (FPGA) 中 CPU 系统的实现,我们选择了 xilinx 公司的 SPARTAN—II 系列的芯片(型号是 XC2S200)。该器件容量为 20 万门,内部有 2352 个 CLB, 14 个 4Kb 的 RAM 块, 208 脚 PQFP 封装形式,支持在系统编程(in-system programmable),实现了 TEC-XP 16 系统中 CPU 的全部功能。在进行这项任务设计时,我们已经充分考虑到如何照顾到现有教学资源(如教材、系统监控程序、软件等)的稳定性。首先,保证了新设计的教学机的指令系统,与 TEC-XP 16 系统的指令系统有良好的兼容性;其次,在构思新型教学计算机的逻辑结构的过程,适当地向 TEC-XP 16 系统的实际组成功能靠拢,尽量地在二者之间有一个平滑的过渡。这我们的设计加进了某些限制条件,但这对减轻任课教师的教学负担、保证教学质量是极其有益的。

TEC-XP 16 系统有关的指令系统、提供的软、硬件资源等内容,在第二章已做了详细的讲解,这里全部从略,只针对用 VHDL 语言提供的功能和设计手段,直接讲解完成该 CPU 系统的设计的过程和相关技术。

把用 VHDL 语言描述的 CPU 的源码文件,经过专用工具软件的编译和综合后,下载到 FPG 芯片之中,也就得到了能够正常运行的 CPU 系统。然后让这个 CPU 和 TEC-XP 16 系统中的 CPU 共用 TEC-XP (FPGA) 主板上的存储器部件、串行接口电路,就构成了一个新的硬件主机系统,通过串行接口接通用 PC 机实现的仿真终端,把监控程序保存到内存的 ROM 存储区,则得到一台硬软件组成相对完整的、真正能够正常运行的教学计算机系统。

A.1.3 FPGA 芯片的外特性和内部结构

现场可编程门阵列(FPGA)是近十多年来出现并开始被广泛应用的大规模集成电路器件。尽管它与 GAL20V8 和 MACH 类型的 PLD 同属现场可编程器件,但是其内部结构却完全不同,也有着不同的特性和性能。GAL20V8 和 MACH 都是采用“与—或”两级逻辑阵列加上输出逻辑单元的内部结构,而 FPGA 的内部结构则是采用许多个独立的可编程的逻辑模块 CLB (configurable logic block)、输入输出模块 IOB (I/O block) 和互连资源(interconnect resource) 3 部分组成。

FPGA 的输入输出引脚数量更多,并可根据需要设置为输入或输出端。内部的每一个 CLB 的电路中都包含组合逻辑电路、1 或者 2 个触发器电路和一些数据选择器电路,可根据需要实现组合逻辑

辑的功能，也可以实现时序逻辑的功能，即把触发器编程为边沿触发的 D 触发器，或者电平触发的 D 型锁存器，可以使其运行于同步方式（使用公用的 CLK 时钟信号）或异步方式（使用 1 个特定数据输入端作为时钟），触发器接收的数据来自于组合逻辑部分的输出。

内部的互连资源由金属线、开关阵列和可编程连接点 3 部分构成，用于实现把数量很大的 CLB 和 IOB 相互连接起来以构成不同的复杂系统。

FPGA 芯片的工作状态（提供的逻辑功能）由芯片内的编程数据存储器设定，该存储器中的内容在断电后不被保存，因此必须在每次加电时被重新装入，这是在芯片内的一个时序电路控制下自动完成的，被装入的数据通常要存放在芯片之外的一片 EPROM 器件中。或者存放在一个磁盘文件中，通过工具软件将其下载到 FPGA 芯片中。

A.2 VHDL 语言概述

VHDL 是超高速集成电路硬件描述语言的英文缩写（Very High Speed Integrated Circuit Hardware Description Language）。它借鉴高级程序设计语言的功能特性对电路的行为和结构进行高度抽象化、规范化的形式描述，并对设计进行不同层次、不同领域的模拟验证与综合处理。IEEE 已经公布了相应的国际标准，进一步推动了 VHDL 的广泛应用。

VHDL 是一种独立于实现技术的语言，提供了把新技术引进现有设计的潜力，覆盖了逻辑设计的诸多领域和层次，支持众多的硬件模型。

VHDL 语言的硬件描述能力强，支持从系统级到门级电路的描述，支持多层次的混合描述，支持电路的结构描述和行为描述。既支持自底向上（bottom-up）的设计，也支持自顶向下（top-down）的设计，既支持模块化的设计，也支持层次化的设计；支持大规模设计中的分解和设计重用。

VHDL 既支持同步电路，也支持异步电路，既支持同步方式，也支持异步方式，既支持传输延迟，也支持惯性延迟，可以更准确地建立复杂的电路硬件模型。

VHDL 属于强类型语言，数据类型丰富，既支持预定义的数据类型，也支持自定义的数据类型。

VHDL 支持进程和函数的概念，有助于设计者组织描述和对行为功能的进一步分类。

VHDL 语言的类属提供了向设计实体传送环境信息的能力，断言语句可用来描述设计本身的约束信息，支持直接在描述中书写错误条件和特殊约束，便于模拟调试，也为综合化简提供了重要信息。

VHDL 语言的功能特别强大，作为这里设计 CPU 系统的实例，很多的功能未被选用，下面只是简单地综述某些已经用到的概念和术语。

VHDL 语言中的元件（component）是数字硬件结构的“未知方框”的抽象，通常由实体和结构体两个概念共同描述，其中实体（entity）用于描述元件与外部环境的接口，其功能要到结构体（architecture）的单元中定义，规定设计实体的输入和输出之间的关系。一个实体可以存在多个对应的结构体，它们可分别以行为、结构、数据流及各种方式的描述手段予以实现。使用在程序结构体中语句用于描述在电路内信号数据的流动情形，包括向信号赋值的语句，条件式信号赋值语句（when ...else...），选择性信号赋值语句（with ...select...），它们都是并发执行的，即各语句同时执

行，与书写的前后位置无关，体现出硬件运行所固有的特性，这与我们熟悉的高级语言中的语句通常按书写顺序依次执行是完全不同的。

VHDL 语言中的信号（**signal**）的概念是数字电路中连线的抽象，是各元件、各进程之间通信的数据通路，信号的状态可能影响与信号相关的进程的运行，体现数字系统各单元的输入和输出的关系，信号可以是多个进程的全局信号。敏感信号是指其值发生变化时，会引起进程中的语句开始执行的那些信号，即它将激活相应进程。信号表示把元件的输入输出端口连接在一起的互连线，功能是保存变化的数据值和连接子元件，用信号类对象可以把实体连接在一起形成模块，向信号赋值是用字符 '**<=**'，信号赋值可以有延迟，可以有历史信息 and 波形值等，进程对信号敏感，敏感信号会激活相应的进程。

VHDL 语言中的变量（**variable**）和常量（**constant**）与信号是不同类型的对象，变量是用于对中间数据的临时存储，而常量则是固定的数据值，向变量赋值是用字符 '**:=**'，变量只有当前值，进程对变量不敏感，变量只在它的程序之中可见（不能作为全局信号使用）

VHDL 语言中的进程（**process**）用于完成电路的行为描述，由一系列的语句组成，是 VHDL 设计中进行功能描述的基本单元，进程要在程序结构体内部加以说明。

```
label: process    (信号列表)    --标号可有可无，列表中的信号属于敏感信号
    说明部分;                --这里说明的对象和数据类型仅在本进程中有效
    being                --敏感信号的值发生变化时将激活进程运行
    行为描述语句; --敏感信号会出现在行为描述语句中
end process    label; --进程中的语句是按书写次序依次运行
```

使用在 process 中的语句比较多，功能很强，在我们设计的 CPU 系统的实例中，描述电路行为（完成的功能、执行的动作）的程序中只用到很少并且是比较基本的几个语句，包括赋值语句、if 语句、case 语句，这几个语句可以被独立使用，也可以复合使用，其中一个语句作为另一个语句的一部分，再次强调，此处的语句是循序执行，简单综述如下。

赋值语句的功能是把一个对象的值复制给另外一个对象，体现的是器件、线路之间的信息传递，十分常用。

<= 是用于向信号赋值的操作符，例如，DC-2 <= DC2 (2 downto 0) ;

:= 是用于向变量赋值的操作符，

例如，addr_A := CONV_INTEGER (UNSIGNED (addrA));

这里用到了变换无符号整数数据为整数类型数据的函数。

if 语句，用于实现依据某种条件选择两种不同处理、运行的功能。

```
它有两种形式:  if 条件表达式 then 语句块1; else 语句块2;
                  endif;
```

```
if 条件表达式1 then 语句块1;
    elsif 条件表达式2 then 语句块2; else 语句块3; endif;
```

例如, `if result = "0000000000000000" then Zero<='1' else Zero<='0'; end if;`

此语句通过检查16 位的ALU运算结果是否为0值, 来确定标志位Zero的结果。

case语句, 针对几种互斥情形, 实现依据不同情形执行不同处理、运算的功能。是在运算器部件设计中描述多路选择器的基本手段, 也是在控制器部件中, 实现依据指令执行步骤(节拍状态)和指令操作码执行不同的功能、提供不同的控制信号的最主要的手段。

```
例如,      case 信号对象 is
              when 信号对象值1 => 语句1;
              when 信号对象值2 => 语句2;
              ..
              when others => 语句n;
end case;
```

```
例如,      case resultOper is
              when "010" => resultOut <= operant1;
              when others => resultOut <= result;
end case
```

此语句通过检查处置ALU 运算结果的信号resultOper 的当前值, 来选择让运算器输出寄存器组经A 口提供的数值还是ALU 的运算结果。

为了体现不同执行在时间上的同步关系, 引入了delta 延迟和延迟进程两个概念, 在我们设计的CPU 系统中, 没有显示地应用此概念, 故不做更多的说明。

VHDL语言的中支持并调用函数, 通常通过给出函数名和相应的参数即可。可以在源文件开头指出会使用到的设计库(library) 的名字, library语句用于打开指定的设计库, 例如:

```
library ieee; 将打开名字为ieee的库;
```

还可以使用 use 子语句使选定的名字成为可见, 例如:

```
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_arith.all;
```

```
use ieee.std_logic_unsigned.all;
```

将使包含在ieee包中的数据类型连接、算术运算、无符号数处理等有关的定义及函数成为可见, 即在当前的设计中可以直接使用。

以上只是对VHDL语言做一简要概述, 相关技术请参考有关资料。

A.3 在 FPGA 芯片中实现的非流水线的 CPU 系统

本节将以不使用指令流水线技术实现的CPU系统为例, 介绍设计、实现的完整CPU系统的一般过程和相关技术。主要属于计算机系统结构课程的知识。

附1. 3. 1 CPU 系统的层次与模块设计

VHDL 语言支持层次结构，一个完整的系统可以由一个顶层模块和多个部件模块组成。任何一个电路模块，都可以使用 VHDL 为其建立一个电路模型，通常情况下，一个电路模型（亦称设计单元）由一个实体说明（用于描述电路的接口信息）和一个结构体（用于描述电路的行为或结构）组成。

该CPU 系统被分成顶层模块（CPU）和3个部件（Am2901、controllor、data_IB）模块，各自的文件名分别是CPU.vhd、Am2901.vhd、controllor.vhd、data_IB.vhd,模块之间的接口关系如图A.2所示。

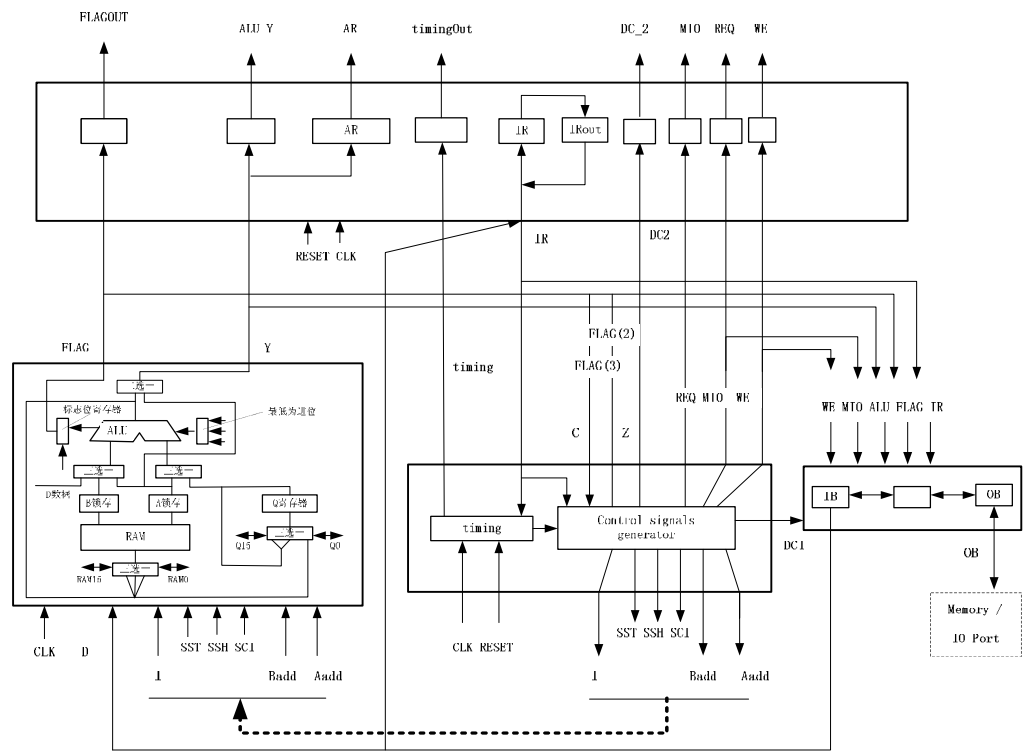


图 A.2 TEC-XP (FPGA) 系统中的单芯片 CPU 的组成结构

一、顶层模块CPU（C P U . v h d）

CPU 系统的顶层模块中,给出该CPU 单元与外部的信号接口,说明用到的3个下层部件的名称和接口参数，还在顶层模块中实现了指令寄存器和地址寄存器。

在该 CPU 系统的顶层模块 CPU 的实体（用 entity 标识）说明中，通过接口 port 给出了 CPU 系统的接口信息（属于公用信息），包括接口信息的名称（需符合标识符的定义，VHDL 不区分标识符中的英文的大小写字母），信息的端口模式（表示信号的流向，输入用 in 标记、输出用 out 标记、输入输出双向用 inout 标记、缓冲用 buffer 标记），信息的取值类型（例如用 std_logic 说明一个二进制位的信号，用 std_logic_vector (15 downto 0) 说明由 16 个二进制位组成的数据）3 部分内容。注意，跟随在 port 之后的右括号之前不应有冒号分隔符。

在顶层模块的结构体（用architecture 标识）部分，首先说明了该CPU 电路模型中使用的3 部件（component）的名称和接口参数，包括运算器部件（名称是Am2901，实现数据运算和暂存）控制器部件（名称是Controllor，提供计算机运行需要的控制信号）数据总线部件（名称是Data_IB，用于实现与存储器和 IO 接口线路通信），接口参数的说明方法和前面在实体中说明接口信息的方法是一样

的；接下来说明了必要的信号。在 `begin` 之后的执行语句中，指出了 3 个部件接口参数的映射关系，执行多个信号的赋值操作，这些赋值语句是并发执行的，体现出硬件电路运行的特性，包括接收运算器的标志位 `FLAG` 输出并传送到输出 `FLAGOUT` 引脚，接收控制器的指令寄存器的输出并传送到 `IR_out` 输出引脚；接收控制器的输出 `MIO_T`、`REQ_T`、`WE_T` 并分别送到输出引脚 `MIO`、`REQ`、`WE`；接收控制器的节拍发生器的节拍编码并传送到输出引脚 `stateOut`。接收控制器的输出信号 `DC2`、`DC1` 并输出到 `DC_2` 和 `DC_1` 等。

通过一个带时钟信号参数 (`clk`) 的进程 (`process`) 描述了指令寄存器和地址寄存器依据 `DC2` 编码值的不同，在时钟脉冲的低电平期间分别接收输入信息的执行过程。即实现了时序逻辑电路指令寄存器 (`IR`) 和地址寄存器 (`AR`) 即在时钟脉冲的低电平期间，当 3 位的 `DC2` 为 001 值时，`IR` 接收输入信号 `IB` 的值，当 3 位的 `DC2` 为 011 值时，`AR` 接收运算器的输出信号 `ALU` 的值。`AR` 的输出送出后用于指定内存的单元地址或者 `IO` 接口芯片的地址，与 `MIO`、`REQ`、`WE` 3 个信号共同控制对内存或者 `IO` 接口芯片的读写操作。

在这里还检查了是联机还是脱机，是手拨指令还是读写内存，以便给出是把手拨开关的内容送到内部总线，还是把从存储器/串行接口读来的信息送到内部总线的控制信号。

二、运算器部件 **Am2901** (`Am2901.vhd`)

`TH_union(FPGA)` 系统中的运算器部件是参照 `TEC-XP16` 系统的 `Am2901` 芯片的功能和组成来设计的。

运算器部件的功能是选择参加运算的数据来源、实现的运算功能、运算结果的处理办法，确定加减运算时 `ALU` 的最低位的进位输入信号，有移位操作时，累加器和 `Q` 寄存器的最高、最低移位输入信号，暂存运算的数据和中间结果，维持与更新 4 位的标志位寄存器内容。

在教材中介绍 `Am2901` 器件时已经讲到该芯片内的运算器内部包括算术与逻辑运算部件 `ALU`、16 个累加器组成的通用寄存器组、用于硬件乘除法指令的 `Q` 寄存器 3 个主体部分，5 组实现多路数据选择的选择器线路和形成 4 个标志位的结果的线路。在 `TEC-XP (FPGA)` 教学机的运算器的设计过程中，比较忠实地实现了 `Am2901` 芯片的组成和功能。

在运算器部件的实体说明中，通过接口 `port` 说明了运算器部件使用的 23 位控制信号，16 位的外部数据输入信号，16 位的结果数据输出信号，4 位的标志位信号，还有一位系统时钟信号，它们都属于敏感信号，其取值发生变化时必将激活运算器模块。这些信号的用法和控制功能在教材其他章节中已有详细讲解，这里不再赘述。

在运算器部件的结构体中，首先指定了 4 位的标志位信号由 `C`、`Z`、`V`、`S` 组成。使用 12 个进程描述了运算器内部的全部运算、处理等功能。

1. 进程 `Control`: `process (funcChoise, RE, S, Cin)` 描述了 `ALU` 完成的运算功能，并用一个 `case` 语句描述 4 位的标志位各自的计算结果。

2. 进程 `process (resultOper, operant1, result)` 描述了 `ALU` 向外输出 16 位数据的来源，是寄存器组通过 `A` 口输出的内容还是 `ALU` 的计算结果。

3. 进程 `process (resultOper, result, RAM0, RAM15)` 描述将写入累加器的数据内容。

4. 进程REG_Result_IN:process (Clk,Result_IN描述累加器在时钟脉冲的低电平期间执行接收输入的操作。
5. 进程Q_Result_IN:process (Clk) 描述在时钟脉冲的上升沿Q 寄存器执行数据写入操作。
6. 进程process (resultOper,Q,Result,Q0,Q15) 描述形成Q寄存器写入内容的方案。
7. 进程 SSHSCi:process (SSHSCi,C_out,C,result (0) ,result (15,Q (15) 描述得到 ALU最低位的进位输入信号、累加器、Q寄存器移位时最高和最低位移位输入信号的方案。
8. 进程 SST_Process:process (Clk) 描述标志位寄存器在不同情况下, 执行的内容维持或更新操作内容。
9. 进程R_Data:process (D,operand1,dataOrigin) 描述ALU 的R操作数来源选择方案。
10. 进程S_DATA:process (operand1,operand2,Q,dataOrigin) 描述ALU的S操作数来源选择方案。
11. 进程A_Locker:process (Clk,A) 描述A锁存器在时钟脉冲的高电平期间、低电平期间和下降沿时刻完成的数据传送与锁存功能。
12. 进程B_Locker:process (Clk,B) 描述B锁存器在时钟脉冲的高电平期间、低电平期间和下降沿时刻完成的数据传送与锁存功能。

三、数据总线部件data_IB (D a t a _ I B . v h d)

数据总线被划分成内部总线IB和外部总线OB两部分, IB 主要用于CPU内部, OB主要用于CPU与计算机的内存储器 and IO接口线路实现通信, 它们都是既用于输入也用于输出的双向传送的电路。

在该模块的实体说明中, 通过接口Port 给出的参数, 属于输入的包括控制信号dc1、mio、we、link, 内部总线的3 组输入数据, 属于输入、输出的是ib 和ob 总线信号。 在该模块的结构体中, 进程process (DC1,MIO,WE) 描述了依据DC1 的编码控制, 把不同的输入信息送到IB 总线这其中也包括在有读内存或者串行接口操作时把OB总线上的信息传送到IB总线。在把指令寄存器的低8 位信息送内部总线时要有符号位扩展处理。

进程 process (MIO,WE) 描述了在有写内存或者写串行接口操作时把IB总线上的信息传送到OB总线, 否则使OB 总线处于高阻态。

四、控制器部件 controllor (c o n t r o l l o r . v h d) 控制器部件的功能是依据所执行的指令(由指令操作码决定)和指令所处的执行步骤(由节拍状态编码确定), 也许还包括必要的控制条件, 来给出这条指令在本节拍所使用的时序控制信号的取值, 保证各个部件协调运行并正确完成确定的运行功能。

在本CPU系统的设计中, 控制器部件模块包括节拍发生器线路和时序控制信号产生线路两部分, 其他的线路在其他模块中给出。控制器是参照TEC-XP 16的组合逻辑控制器实现方案设计的, 指令格式、指令系统、指令执行步骤划分、使用的控制信号等与 TEC-XP 16 相同, 这些内容已经在第二章讲解过了, 在本章不再介绍, 需要时, 请查看第二章相关部分。

在本模块的实体说明中, 通过接口Port 给出全部的输入信号, 包括系统总请信号 RESET、系统时钟信号clk、指令寄存器IR 的16 位信号、标志位进位C、结果为0 位Z 信号; 以及输出的32位的时序控制信号, 4 位的节拍状态信号。

在本模块中，说明了3个进程，分别简单介绍如下。

进程 `instruction:process (IR (15 downto 8))` 描述了29条基本指令的汇编语句名和指令操作码的对应关系，是以向布尔类型信号赋值的形式进行的，在后面就可以使用对应的布尔信号表示某一条指令。

进程 `timeKeeper:process (clk, RESET)` 描述了节拍发生器的运行原理（状态转换过程）这里的节拍发生器的运行状态，要保持与 TEC-XP 16系统的指令执行步骤相同，因此不必讨论节拍发生器的状态编码和状态转换设计，问题就变成如何用 VHDL 语言写出实现节拍状态转换的语句这样一个最简单过程。有两种可行方案，第一种方案是直接写出每一位节拍触发器的状态转换语句，更接近线路描述；第二种方案是直接写出完成状态转换的语句，更接近节拍发生器的功能描述。在这里选用的是第一种方案，给出了对29条基本指令执行过程的设计结果。

回忆一下，指令操作码的最高几位用于区分指令分组，IR15、IR14为0X是A组指令，为10是B组指令，为11是C、D组指令，再用IR11进一步区分，0为C组，1为D组。节拍发生器几位触发器是按照当前的节拍状态和遇到的指令组来变化状态的，用3个if语句设计了3位触发器的状态变化，最高的两位触发器，除了在 RESET 信号到来时 Timing (3) 变为 1 值外，其他情况下都保持为 0 值。进程 `signal_process:process (timing)` 描述产生32位时序控制信号的方案。其基本思路是

把节拍状态编码作为第一级分支，用连续的if语句区分不同的节拍，再用出现在该节拍中的指令或指令组作为第二级分支，来最终确定用到得32位控制信号的取值。要保证在脱机状态下这些信号应处于高阻状态，只在联机状态下处于正常逻辑电平。

对公用于所有指令的公共操作节拍，各个控制信号的取值与指令的操作码无关，可以直接给出这些控制信号的取值，例如，对01000、00000、00010 这3个节拍的處理就是如此。而00111、00101这2个节拍只用于CALA一条指令，其他基本指令不会进入此节拍，也可以直接给出这些控制信号的取值，但是在扩展的新指令也要进入这两个节拍时，仍需要加进指令编码的控制作用。

对于其他各个节拍，则要区分不同的指令，依据具体的操作功能来选择每一个（组）信号的取值。最简单的办法，当然是为每一条出现在本节拍中的指令单独写出向它所使用的控制信号赋值的语句，对于初学者，这是最可行的方案，不足是设计出的 HDML 的源码篇幅更长，但对最终的设计结果并无实质性影响。如果设计者对指令的执行过程理解得很清楚，对每一个节拍中的操作功能也有深入的了解，思维足够快，也可以依据不同指令的组合情况，直接向每一种指令组合可以公用的控制信号进行赋值，下面的例子就是按这一方案完成设计的，例如只有A组指令会进入00011节拍，A组指令都不读写内存或接口，因此可以在00011节拍中直接给出MIO、REQ、WE的值100，而不必涉及有关指令码。这只不过是把可以留给设计工具软件来优化的工作改由设计者本人直接完成了，这种处理办法在实际工作中是否值得提倡，还有待于尝试。

用节拍状态作为第一级分支时，使用的是连续的 if 语句。其具体语句的架构简介如下。

```
if (Timing = "01000") then 该节拍中的语句;
elseif (expandTiming = "0000") then 该节拍中的32位信号赋值语句;
```

```

elseif (expandTiming = "0010") then 该节拍中的32位信号赋值语句;
elseif (expandTiming = "0011") then 该节拍中的32位信号赋值语句;
elseif (expandTiming = "0110") then 该节拍中的32位信号赋值语句;
elseif (expandTiming = "0100") then 该节拍中的32位信号赋值语句;
    elseif (expandTiming = "0111") then 该节拍中的32位信号赋值语句;
        elseif (expandTiming = "0100") then 该节拍中的32位信号赋值语句;
endif;

```

用指令（指令组）在各个节拍中进行第二级分支时，需要根据不同的情况进行不同的处理。为了方便地指明指令或者指令组，在该模块中说明了29个布尔类型的信号，已经在进程instruction中看到。

在节拍编码为 1000、0000、0010 时，是用于所有指令的公用节拍，32 位的控制信号与指令操作码无关，在节拍编码为 0111、0100 时，是单独用于 CALA 一条指令的节拍，也不必区分指令操作码（如果要在本节拍中添加新的指令则另当别论）。

在另外的3个节拍中，则依据不同指令或者指令组对不同的时序控制信号进行相应的赋值操作。例如，在0011 节拍中，需要按A组17条指令的不同情况进行个别处理。A组全部17条指令都不涉及专用寄存器的接收操作，故为DC2赋值000。有关对存储器和IO接口的读写控制，17条指令都无读写操作需求，故为MIO、REQ、WE 三个控制信号分别赋值为1、0、0。

对 8 条双寄存器指令，A、B 口地址都分别选择指令寄存器的 IR (3 downto 0 和 IR (7 downto 4)；对 4 条单寄存器指令，A、B 口地址都分别选择 0000 和 IR (7 downto 4)，对另外的 5 条相对转移指令，A、B 口地址都选择 0101（即程序计数器 PC 的编号）。这里有个 PC 的内容与相对转移偏移量（保存在指令寄存器的低 8 位相加求转移地址的计算过程，此时需要把指令寄存器的低 8 位中的补码值进行符号扩展后送到 ALU 的 D 输入端（见运算器部件）并完成相加操作，至于是否真正进行转移，要根据指令中指定的转移判断条件（C、NC、Z、NZ）和这两个标志位的现行值共同决定，要转移，则把计算得到的转移地址传送到 PC 中（此时应向 I8~I6 信号赋值 010，使 B 口指定的寄存器接收输入数据），否则通过使 I8~I6 的值为 001，使 B 口指定的寄存器不接收计算出的转移地址，则下一条执行的指令必定是跟随在条件相对转移指令之后的相邻指令，既不执行转移操作，这是通过控制使 I7 信号为 0 还是为 1 完成的。对其他几组控制信号的赋值处理也容易看懂，无需多加说明。

请注意，进程signal_process:process (timing) 的内容几乎就是把硬连线控制器的指令流程表中的内容简单地抄写过来，如果想更简单，也可以把32 位控制信号组成一个列表直接书写出来，而不是按控制信号的分组情况，一组一组地分开来写。

在完成扩展指令的教学实验中，只须在3个进程 instruction:process (IR (15 downto 8) 、 timeKeeper:process (clk, RESET) 、 signal_process:process (timing) 中增加对新指令编码、使用的新节拍及其转换关系、新指令在各个节拍中的控制信号的取值的相应语句，重新编译后再把得到的结果文件的内容下载到FPGA芯片中即可,比起在TEC-XP16 教学机中的教学实验要简单许多。

下面我们给出用VHDL 语言描述的TEC-XP（FPGA）系统CPU 功能的源程序代码。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity CPU is
    port( CLK           :in std_logic;           --ALU时钟
          RESET         :in std_logic;           --系统总清控制
          MIO            :out std_logic;          --内存/接口读写控制
          REQ            :out std_logic;
          WE             :out std_logic;
          HndIns         :std_logic;             --手拨指令、数据
          Link           :In std_logic;          --控制联、脱机工作方式
          SWTOIB         :ou std_logic;          --控制手拨指令、数据使能
          OMIO           :out std_logic;         --控制245使能
          ALU_Y          :out std_logic_vector(15 downto 0); --运算结果输出
          FLAGOUT        :out std_logic_vector(3 downto 0); --状态标志位输出
          OB             :inout std_logic_vector(15 downto 0); --数据总线
          ADDR           :buffer std_logic_vector(15 downto 0); --地址总线
          IRout          :buffer std_logic_vector(15 downto 0); --显示指令
          DC_2           :inout std_logic_vector(2 downto 0); --寄存器接受控制
          stateOut       :out std_logic_vector(3 downto 0); --显示指令节拍
          addra          :inout std_logic_vector(3 downto 0); --运算器控制，A口
          addrb          :inout std_logic_vector(3 downto 0); --B口
          SSHSCI         :inout std_logic_vector(2 downto 0); --进位、移位控制
          resultOper     :inout std_logic_vector(2 downto 0); --I8-I6
          funcChoose     :inout std_logic_vector(2 downto 0); --I5-I3
          dataOrigin     :inout std_logic_vector(2 downto 0); --I2-I0
          SST            :inout std_logic_vector(2 downto 0); --标志位控制
          DC_1           :inout std_logic_vector(2 downto 0); --数据送内部总线选择
          GMEM           :out std_logic );        --内存/开关选择
end CPU;

architecture CPU_architecture of CPU is
    component AM2901
    port(
        CLK           :in std_logic;           --时钟脉冲
        dataOrigin    :in std_logic_vector(2 downto 0); -- I2-I0
        funcChoose    :in std_logic_vector(2 downto 0); -- I8-I6
        resultOper    :in std_logic_vector(2 downto 0); -- I8-I6
        SSHSCi       :in std_logic_vector(2 downto 0); --进位/移位信号形成
        SST          :in std_logic_vector(2 downto 0); --状态标志位
        addra        :in std_logic_vector(3 downto 0); --A端口地址
    );
    end component;
end architecture;
```

```

    addrB      :in std_logic_vector(3 downto 0);      --B端口地址
    D          :in std_logic_vector(15 downto 0);      --外部数据输入
    resultOut   :out std_logic_vector(15 downto 0);    --运算结果输出
    FLAG       :out std_logic_vector(3 downto 0)      --4个标志位
);
end component;

component controllor
port(
    IR          :in std_logic_vector(15 downto 0);    --16位指令码
    C,Z         :in std_logic;                        --2个标志位
    state       :out std_logic_vector(3 downto 0);    --节拍状态编码

    RESET       :in std_logic;                        --系统总清信号
    MIO         :out std_logic;                        --32位的控制信号
    REQ         :out std_logic;
    WE          :out
std_logic;
    addrA       :out std_logic_vector(3 downto 0);
    addrB       :out std_logic_vector(3 downto 0);
    SSHSCI      :out std_logic_vector(2 downto 0);
    result0per  :out std_logic_vector(2 downto 0);    --I8-I6
    funcChoose  :out std_logic_vector(2 downto 0);    --I5-I3
    data0rigin  :out std_logic_vector(2 downto 0);    --I2-I0
    SST         :out std_logic_vector(2 downto 0);
    DC1         :out std_logic_vector(2 downto 0);
    DC2         :out std_logic_vector(2 downto 0);
    clk         :in std_logic;                        --系统时
    钟
    Link        :in std_logic                        --联机/脱机信
    号
);
end component;

component data_IB
Port (
    DC1 :in std_logic_vector(2 downto 0);
    ALU :in std_logic_vector(15 downto 0); --ALU输出IB
    FLAG :in std_logic_vector(3 downto 0); --状态标志位送IB
    IR   :in std_logic_vector(15 downto 0); --指令低位字节符号扩展送IB
    WE   :in std_logic;
    MIO  :in std_logic;
    IB   :out std_logic_vector(15 downto 0); --内部总
    线 OB :inout std_logic_vector(15 downto 0); --数据
    总线 Link :in std_logic
);
end component;

signal state :std_logic_vector(3 downto 0); --当前状态
signal ALU, IReg, IB, ADDR_temp :std_logic_vector(15 downto 0); --17组(个)信号量
signal FLAG, PORTA, PORTB :std_logic_vector(3 downto 0);
signal SSHSCi_T :std_logic_vector(2 downto 0);
signal DC2, DC1 :std_logic_vector(2 downto 0);
signal MIO_T, REQ_T, WE_T :std_logic;
signal C_T, Z_T :std_logic;
signal SWIBOUT_T :std_logic_vector(2 downto 0);
signal GMEM_T :std_logic_vector(1 downto 0);

```

```

begin C_T<=FLAG(3);
Z_T<=FLAG(2);
unit1:controllor port
    map(IReg, C_T, Z_T, state, RESET, MIO_T, REQ_T, WE_T, addra, addrb,
        SSHSCI, resultOper, funcChoose, dataOrigin, SST, DC1, DC2, CLK, Link);

unit2:AM2901 port
    map(CLK, dataOrigin, funcChoose, resultOper, SSHSCI, SST, addra, addrb, IB, ALU, FLAG);

unit5:data_IB port map(DC1, ALU, FLAG, Ireg, WE_T, MIO_T, IB, OB, Link);

FLAGOUT    <= FLAG;
IRout       <= IReg;
DC_2        <= DC2(2 downto 0);
ALU_Y       <= ALU;
WE          <= WE_T;
REQ         <= REQ_T;
MIO         <= MIO_T;
stateOut    <= state;
GMEM_T      <=HndIns&Link;
DC_1        <=DC1;

process(clk)
begin
    --依据脱机/联机、读内存/手拨指令开关信号确定内部总线上信息来源
    if (GMEM_T="01") then GMEM<='0'; else GMEM<='1';
    end if;
    if ((HndIns='1' and MIO_T='0' and WE_T='1') or Link='0')
        then SWTOIB<='0'; else SWTOIB<='1';
    end if;

    if (Link='0' or (HndIns='1' and MIO_T='0'))
        then OMIO<='0'; else OMIO<='1';
    end if;

    if(clk='0') then
        --在时钟脉冲的低电平期间，
        case DC2(2 downto 0) is
            --用 DC2 的编码控制
            when "001" => IReg <= IB;    --指令寄存器接收输入的指令，
            when "011" => ADDR <= ALU; --地址寄存器接收输入的地址。when
            others => NULL;
        end case;
    else
    end if;
end process;

end CPU_architecture;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity data_IB is
    Port ( dc1 : in std_logic_vector(2 downto 0); ALU :
        in std_logic_vector(15 downto 0);
        flag: in std_logic_vector(3 downto 0);
        ir  : in std_logic_vector(15 downto 0);
        we  : in std_logic;
        mio : in std_logic;
        ib : inout std_logic_vector(15 downto 0); ob :
        inout std_logic_vector(15 downto 0); Link: in
        std_logic
    );
end data_IB;

architecture Behavioral of data_IB is
begin
    process(DC1, MIO, WE)
    begin
        if (Link='1') then          --联机状态下,
            case DC1 is             --用 DC1 编码选择送到内部总线的数据来源
                when "000" =>
                    if(MIO='0' and WE='1') then IB <= OB; end if; --来自数据总线
                when "001" => IB <= ALU;                          --来自ALU的输出
                when "010" =>
                    case IR(7) is    --来自IR的低8位
                        when '0' => IB <= "00000000"&IR(7 downto 0); --执行符号扩展
                        when '1' => IB <= "11111111"&IR(7 downto 0);
                        when others =>
                            end case;
                    when "011" => IB <= flag&"00000000000000";      --来自标志位寄存器
                    when others =>
                        end case;
                end if;
            end process;

        process(MIO, WE)
        begin
            if (Link='1') then      --在联机状态下,
                if(MIO='0' and WE='0') --在执行写内存或接口操作时,
                    then OB <= IB; else OB <= "ZZZZZZZZZZZZZZZZ";
                end if;              --内部总线信息送数据总线,
            end if;                  --否则使数据总线处于高阻态
        end process;
    end Behavioral;
end

```



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Am2901 is
  port(
    CLK          :in std_logic;           --时钟脉冲
    dataOrigin   :in std_logic_vector(2 downto 0); --选择数据来源
    funcChoose   :in std_logic_vector(2 downto 0); --选择运算功能
    resultOper   :in std_logic_vector(2 downto 0); --选择结果处置方案
    SSHSCi       :in std_logic_vector(2 downto 0); --控制最低位进位和移位输入
    SST          :in std_logic_vector(2 downto 0); --状态标志位
    addrB        :in std_logic_vector(3 downto 0); --B端口地址
    addrA        :in std_logic_vector(3 downto 0); --A端口地址
    D            :in std_logic_vector(15 downto 0); --外部数据输入
    resultOut    :out std_logic_vector(15 downto 0); --运算结果输出
    FLAG         :out std_logic_vector(3 downto 0)  --标志位输出信号
  );
end AM2901;

architecture Behavioral of AM2901 is
  signal cin      :std_logic;           --最低位进位信号
  signal C_out,Z_out,V_out,S_out :std_logic; --状态标志位
  signal RAM0,RAM15,Q0,Q15:std_logic;   --移位信号
  signal Q,Q_temp,Result_IN :std_logic_vector(15 downto 0); --乘商寄存器

  signal A,B,operand1,operand2,operand1_temp,
    operand2_temp,RE,S,result:std_logic_vector(15 downto 0);
  signal C,Zero,Over,Sign:std_logic;    --暂存标志位

  subtype regData is integer range 0 to 65535; --定义子类型
  type regArray is array(0 to 15) of regData; --寄存器类型
  signal RAM :regArray; --16个通用寄存器组

begin
  FLAG<=C_out&Z_out&V_out&S_out; --标志位输出

  Control:process(funcChoose, RE, S, Cin) --ALU运算功能选择并计算标志位的值
  begin
    case funcChoose is --选择运算功能并执行运算
      when "000"=>result<=RE+S+cin; --R+S+CIN
      when "001"=>result<=S-RE - not Cin; --S-R
      when "010"=>result<=RE-S - not Cin; --R-S
      when "011"=>result<=RE or S; --R or S
      when "100"=>result<=RE and S; --R and S
      when "101"=>result<= not(RE) and S; --not R and S
      when "110"=>result<=RE xor S; --R xor S
      when "111"=>result<=not(RE xor S); --not (R xor S)
      when others=>NULL;
    end case;

    case funcChoose is --计算 4个标志位的值

```

```

when "000" => --R+S 运算
    C<=(S(15) and RE(15))or(RE(15) and
        not result(15))or (not result(15) and S(15));
    Over<=(S(15) and RE(15) and not result(15))or
        ( not S(15) and not RE(15) and result(15));
when "001" => --S-R减运算
    C<=(S(15)and not RE(15))or (not RE(15)and
        not result(15))or (not result(15)and S(15));
    Over<=(S(15)and not RE(15)and not result(15))or
        (not S(15)and RE(15)and result(15));
when "010" => --R-S运算
    C<=(RE(15)and not S(15))or (not S(15)and
        not result(15))or (not result(15)and RE(15));
    Over<=(RE(15)and not S(15)and not result(15))or
        (not RE(15)and S(15)and result(15));
when others=>NULL; --逻辑运算不改变C,V
end case;
Sign<=result(15); --最高位为符号位
if result="0000000000000000" --结果是否为0
    then Zero<='1';
    else Zero<='0';
end if;
end process control;

```

```

-----
process(resultOper, operant1, result) --选择ALU的输出内容
begin
    case resultOper is --输出A口内容
        when "010"=>
            resultOut<=operant1;
        when others=>
            resultOut<=result; --输出运算结果
    end case;
end process;

```

```

----- 用于通用寄存器组的两个进程 -----
process(resultOper, result, RAM0, RAM15) --形成寄存器的写入内容
begin
    case resultOper(2 downto 1) is
        when "01"=>result_IN<=result; --ALU运算结果
        when "10"=>result_IN<=RAM15&result(15 downto 1); --右移
        when "11"=>result_IN<=result(14 downto 0)&RAM0; --左移
        when others=>NULL;
    end case;
end process;

```

```

REG_Result_IN:process(Clk, result_IN) --通用寄存器保存结果
variable addr_A, addr_B:integer range 0 to 15;
begin
    addr_A:=CONV_INTEGER(UNSIGNED(addrA));
    addr_B:=CONV_INTEGER(UNSIGNED(addrB));
    IF CLK='0' THEN --低电平接收

```

```

        IF result0per(2)='1' or result0per(1)='1' THEN
            RAM(addr_B)<=CONV_INTEGER(UNSIGNED(result_IN));
        END IF; END IF;
        A<=STD_LOGIC_VECTOR(CONV_UNSIGNED(RAM(addr_A),16));
        B<=STD_LOGIC_VECTOR(CONV_UNSIGNED(RAM(addr_B),16));
    end process REG_Result_IN;

```

```

-----用于Q寄存器的两个进程-----
Q_Result_IN:process(Clk)                                --Q寄存器接收新值或移位
begin
    if Clk'event and Clk='1' then                        --正跳变沿Q寄存
        case result0per is
            when "000"=> Q<=Q_temp;                     --接受
            when "100"=> Q<=Q_temp;                     --右移位
            when "110"=> Q<=Q_temp;                     --左移位
            when others=>NULL;
        end case;
    end if;
end process Q_Result_IN;

process(result0per,Q, result,Q0,Q15)                    --形成Q寄存内容
begin
    case result0per is
        when "000"=>
            Q_temp<=result;                             --ALU运算结果
        when "100"=>Q_temp<=Q15&Q(15 downto 1); --右移
        when "110"=>Q_temp<=Q(14 downto 0)&Q0; --左移
        when others=>NULL;
    end case;
end process;

```

```

-----
SSHSci_Process:process(SSHSci,C_out,C,result(0),result(15),Q(15))
begin
    case SSHSci is
        when "000"=>cin<='0';                            --形成最低位进位信号
        when "001"=>cin<='1';
        when "010"=>cin<=C_out;
        when "100"=>                                     --形成移位输入信号的
            if result0per(2)='1' and result0per(1)='1'
                then RAM0<='0';
            elsif result0per(2)='1' and result0per(1)='0'
                then RAM15<='0';
            end if;
        when "101"=>
            if result0per(2)='1' and result0per(1)='1'
                then RAM0<=C_out;
            elsif result0per(2)='1' and result0per(1)='0'
                then RAM15<=C_out;
            end if;
        when "110"=>
            if result0per(2)='1' and result0per(1)='1'

```

```

        then
            RAM0<=Q(15);
            Q0<=not result(15);
        elsif result0per(2)='1' and result0per(1)='0'
            then
                RAM15<=C;
                Q15<=result(0);
        end if;
    when "111"=>
        if result0per(2)='1' and result0per(1)='1'
            then
                RAM0<='0';
                Q0<='0';
            elsif result0per(2)='1' and result0per(1)='0'
                then
                    RAM15<=result(15) xor Over;
                    Q15<=result(0);
            end if;
        when others=> NULL;
    end case;
end process SSHSCi_Process;

```

```

SST_Process:process(Clk)                --状态寄存器的接收与保存
begin
    if CLK'event and Clk = '1' then      --运算器周期的正跳变用于寄存器接收
        case SST is
            when "000"=>
                C_out<=C_out;
                V_out<=V_out;
                Z_out<=Z_out;
                S_out<=S_out;
            when "001"=>                  --状态寄存
                C_out<=C;
                Z_out<=Zero;
                V_out<=Over;
                S_out<=Sign;
            when "010"=>                  --恢复标志位原现场值
                C_out<=D(15);             --数据总线的高位为FLAG位
                Z_out<=D(14);
                V_out<=D(13);
                S_out<=D(12);
            when "011"=>                  --置'0' C
                C_out<='0';
                Z_out<=Z_out;
                V_out<=V_out;
                S_out<=S_out;
            when "100"=>                  --置'1' C
                C_out<='1';
                Z_out<=Z_out;
                V_out<=V_out;
                S_out<=S_out;
            when "101"=>                  --右移操作

```

```

        C_out<=result(0);
        Z_out<=Z_out;
        V_out<=V_out;
        S_out<=S_out;
    when "110"=>                --左移操作
        C_out<=result(15);
        Z_out<=Z_out;
        V_out<=V_out;
        S_out<=S_out;
    when "111"=>                --联合右移
        C_out<=Q(0);
        Z_out<=Z_out;
        V_out<=V_out;
        S_out<=S_out;

    when others=>null;          --状态寄存器保持不变
end if;
end process SST_Process;

-----

R_Data:process(D, operand1, data0rigin)
Begin                            --用于R操作数的多路选择器
    case data0rigin is
        when "000" => RE<=operand1;
        when "001" => RE<=operand1;
        when "010" => RE<="0000000000000000";
        when "011" => RE<="0000000000000000";
        when "100" => RE<="0000000000000000";
        when "101" => RE<=D;
        when "110" => RE<=D;
        when "111" => RE<=D;
        when others=>NULL;
    end case;
end process R_Data;

S_DATA:process(operand1, operand2, Q, data0rigin)
Begin                            --用于S操作数的多路选择器
    case data0rigin is
        when "000" => S<=Q;
        when "001" => S<=operand2;
        when "010" => S<=Q;
        when "011" => S<=operand2;
        when "100" => S<=operand1;
        when "101" => S<=operand1;
        when "110" => S<=Q;
        when "111" => S<="0000000000000000";
        when others=>NULL;
    end case;
end process S_DATA;

A_LOCKER:process(Clk, A)        --A 锁存器

```

```

begin
    if Clk'event and Clk='0'           --在时钟信号的下降沿,
        then operant1_temp<=A;         --锁存寄存器的输出内容
    end if;

    if Clk='1'
        then operant1<=A;               --在时钟的高电平期间直接送出寄存器的内容
        else operant1<=operant1_temp;   --在时钟的低电平期间送出锁存器保存的内容
    end if;
end process A_LOCKER;

B_Locker:process (Clk,B)               --B 锁存器
begin
    if CLK'event and Clk='0'
        then operant2_temp<=B;
    end if;

    if Clk='1'
        then operant2<=B;
        else operant2<=operant2_temp;
    end if;
end process B_Locker;
end Behavioral;

--*****--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity controllor is
    Port ( IR      : in std_logic_vector(15 downto 0);
          c        : in std_logic;
          z        : in std_logic;
          Link     : in std_logic;
          clk      : in std_logic;
          RESET    : in std_logic;
          state    : out std_logic_vector(3 downto 0);
          MIO      : out std_logic;
          REQ      : out std_logic;
          WE       : out std_logic;
          addra    : out std_logic_vector(3 downto 0); addrb :
          out std_logic_vector(3 downto 0); SSHSCI: out
          std_logic_vector(2 downto 0); resultOper: out
          std_logic_vector(2 downto 0); funcChoose: out
          std_logic_vector(2 downto 0); dataOrigin: out
          std_logic_vector(2 downto 0); SST : out
          std_logic_vector(2 downto 0);
          DC1      : out std_logic_vector(2 downto 0); DC2
                   : out std_logic_vector(2 downto 0) );
end controllor;

```

```

architecture Behavioral of controllor is
    signal timing : std_logic_vector (4 downto 0);
    signal CMDadd, CMDsub, CMDand, CMDcmp, CMDxor, CMDtest, CMDor, CMDmvrr: boolean;
    signal CMDdec, CMDinc, CMDshl, CMDshr, CMDjr, CMDjrc, CMDjrnc, CMDjrz, CMDjrnz: boolean;
    signal CMDjmpa, CMDldrr, CMDin : boolean;
    signal CMDstrr, CMDpshf, CMDpush, CMDout,
           CMDpop, CMDmvr, CMDpopf, CMDret, CMDcala: boolean;
    signal CMDadc, CMDsbb, CMDrcl, CMDrcr, CMDasr, CMDnot, CMDjmp, CMDjrs: boolean;
    signal CMDjrns, CMDclc, CMDstc, CMDei, CMDdi, CMDcalr, CMDldra, CMDldrx, CMDstrx: boolean;
    signal CMDstra, CMDiret: boolean;
    signal I : std_logic_vector (8 downto 0);

begin
    resultOper <= I(8 downto 6);
    funcChoose <= I(5 downto 3);
    dataOrigin <= I(2 downto 0);

    instruction:
        process(IR(15 downto 8))          --指定指令的操作码
        begin
            CMDadd    <= (IR(15 downto 8) = "00000000" );
            CMDsub    <= (IR(15 downto 8) = "00000001" );
            CMDand    <= (IR(15 downto 8) = "00000010" );
            CMDcmp    <= (IR(15 downto 8) = "00000011" );
            CMDxor    <= (IR(15 downto 8) = "00000100" );
            CMDtest   <= (IR(15 downto 8) = "00000101" );
            CMDor     <= (IR(15 downto 8) = "00000110" );
            CMDmvrr   <= (IR(15 downto 8) = "00000111" );
            CMDdec    <= (IR(15 downto 8) = "00001000" );
            CMDinc    <= (IR(15 downto 8) = "00001001" );
            CMDshl    <= (IR(15 downto 8) = "00001010" );
            CMDshr    <= (IR(15 downto 8) = "00001011" );
            CMDjr     <= (IR(15 downto 8) = "01000001" );
            CMDjrc    <= (IR(15 downto 8) = "01000100" );
            CMDjrnc   <= (IR(15 downto 8) = "01000101" );
            CMDjrz    <= (IR(15 downto 8) = "01000110" );
            CMDjrnz   <= (IR(15 downto 8) = "01000111" );
            CMDjmpa   <= (IR(15 downto 8) = "10000000" );
            CMDldrr   <= (IR(15 downto 8) = "10000001" );
            CMDin     <= (IR(15 downto 8) = "10000010" );
            CMDstrr   <= (IR(15 downto 8) = "10000011" );
            CMDpshf   <= (IR(15 downto 8) = "10000100" );
            CMDpush   <= (IR(15 downto 8) = "10000101" );
            CMDout    <= (IR(15 downto 8) = "10000110" );
            CMDpop    <= (IR(15 downto 8) = "10000111" );
            CMDmvr    <= (IR(15 downto 8) = "10001000" );
            CMDpopf   <= (IR(15 downto 8) = "10001100" );
            CMDret    <= (IR(15 downto 8) = "10001111" );
            CMDcala   <= (IR(15 downto 8) = "11001110" );
        end process;

    timeKeeper:process(clk, RESET)          --描述节拍状态转换功能

```

```

begin
    if (Link='1') then timing(4) <= '0';

    if (RESET = '1')
        then timing <= "01000";
        elsif ((clk'event) and (clk = '1'))
            then timing(3) <= '0';

    if (((timing = "00010") and (IR(15) = '1'))      --依据当前节拍状态和指令组
        or ((timing = "00110") and (IR(15) = '1'))  --控制节拍每位触发器状态变化
        or ((timing = "00100") and (IR(15) = '1') and (IR(14) = '1') and (IR(11) = '1'))
    or (timing = "00111")) )
        then timing(2) <= '1';
        else timing(2) <= '0';
    end if;

    if ( ( (timing = "00000")
        or (timing = "00010")
        or ((timing = "00110") and (IR(15 downto 11) = "11100"))
        or ((timing = "00100") and (IR(15 downto 14) = "11") and (IR(11) = '1')) ) )
        then timing(1) <= '1';
        else timing(1) <= '0';
    end if;

    if (((timing = "00010") and (IR(15) = '0'))
        or ((timing = "00110") and (IR(15 downto 11) = "11100"))
        or (timing = "00111")
        or ((timing = "00100") and (IR(15 downto 14) = "11") and (IR(11) = '1')) ) )
        then timing(0) <= '1';
        else timing(0) <= '0';
    end if;
end if;
end if;
end process;

state <= timing(3 downto 0);

signal_produce_Process:process(timing)    --按照节拍状态是实现分支
begin
    if (Link='1') then                    --在联机状态下产生正常电平的控制信号

----- 01000 -----
        if (timing = "01000") then        --在公用所有指令的节拍,
            MIO <= '1'; --一个控制信号的值与具体指令码无关
            REQ <= '0';
            WE <= '0';
            addra <= "0101";
            addrb <= "0101";
            SSHSCI <= "001";
            I <= "011001001";
            SST <= "000";
            DC1 <= "000";

```



```
DC2      <= "111";
```

```
----- 00000 -----
```

```
elseif (timing = "00000") then
```

```
  MIO      <= '1';
  REQ      <= '0';
  WE       <= '0';
  addra    <= "0101";
  addrb    <= "0101";
  SSHSCI   <= "001";
  I        <= "010000011";
  SST      <= "000";
  DC1      <= "000";
  DC2      <= "011";
```

```
----- 00010 -----
```

```
elseif (timing = "00010") then
```

```
  MIO      <= '0';
  REQ      <= '0';
  WE       <= '1';
  addra    <= "0000";
  addrb    <= "0000";
  SSHSCI   <= "000";
  I        <= "001000000";
  SST      <= "000";
  DC1      <= "000";
  DC2      <= "001";
```

```
----- 00011 -----
```

```
elseif (timing = "00011") then  --在选定的节拍中，
                                --在按照指令码进行分支控制
  MIO <= '1';
  REQ <= '0';
  WE  <= '0';
```

```
----- A □ -----
```

```
if (CMDadd or CMDsub or CMDand or CMDcmp or CMDxor or CMDtest or CMDor or CMDmvrr)
then addra <= IR(3 downto 0);
elseif (CMDdec or CMDinc or CMDshl or CMDshr)
then addra <= "0000";
else addra <= "0101";
end if;
```

```
----- B □ -----
```

```
if (CMDadd or CMDsub or CMDand or CMDcmp or CMDxor or CMDtest or
CMDor or CMDmvrr or CMDdec or CMDinc or CMDshl or CMDshr) then
  addrb <= IR(7 downto 4);
else addrb <= "0101";
end if;
```

```
----- SSHSCI -----
```

```
if (CMDsub or CMDcmp or CMDinc)
then SSHSCI <= "001";
else SSHSCI <= "000";
end if;
```

```
----- I (8) -----
```

```
if (CMDshl or CMDshr)
```

```

        then I(8) <= '1';
        else I(8) <= '0';
    end if;
-----I(7)-----
    if (CMDcmp or CMDtest or CMDshr)
    then I(7) <= '0';
        elsif (CMDjrc) then I(7) <= C;
            elsif (CMDjrnc) then I(7) <= not(C);
                elsif (CMDjrz) then I(7) <= Z;
                    elsif (CMDjrnz) then I(7) <= not(Z);
                        else I(7) <= '1';
                    end if;
            end if;
        end if;
-----I(6)-----
    I(6) <= '1';

-----I(5)-I(3)-----
    if (CMDsub or CMDcmp or CMDdec)
    then I(5 downto 3) <= "001";
        elsif (CMDtest or CMDand) then I(5 downto 3) <= "100";
            elsif (CMDxor) then I(5 downto 3) <= "110";
                elsif (CMDor) then I(5 downto 3) <= "011";
                    else I(5 downto 3) <= "000";
                end if;
            end if;
        end if;
-----I(2)-I(0)-----
    if (CMDdec or CMDinc or CMDshr or CMDshl)
    then I(2 downto 0) <= "011";
        elsif (CMDjr or CMDjrc or CMDjrnc or CMDjrz or CMDjrnz)
        then I(2 downto 0) <= "101";
            elsif (CMDmvrr) then I(2 downto 0) <= "100";
                else I(2 downto 0) <= "001";
            end if;
        end if;
-----SST-----
    if (CMDshl) then SST <= "110";
        elsif (CMDshr) then SST <= "101";
            elsif (CMDmvrr or CMDjr or CMDjrc or CMDjrnc or CMDjrz or CMDjrnz)
            then SST <= "000";
                else SST <= "001";
            end if;
        end if;
-----DC1-----
    if (CMDjr or CMDjrc or CMDjrnc or CMDjrz or CMDjrnz)
    then DC1 <= "010";
        else DC1 <= "000";
    end if;
-----DC2-----
    DC2 <= "000";

-----00110-----
    elsif (timing = "00110") then
        MIO <= '1';
        REQ <= '0';
        WE <= '0';
-----A □-----

```

```

if (CMDin or CMDstrr or CMDpshf or CMDpush or CMDout)
then  addra <= "0000";
    elsif (CMDpop or CMDpopf or CMDret) then  addra <= "0100";
    elsif (CMDldrr) then  addra <= IR(3 downto 0);
    else  addra <= "0101";
end if;

-----B □-----
if (CMDldrr or CMDin or CMDout) then  addrb <= "0000";
    elsif (CMDjmpa or CMDmvrdr or CMDcala) then  addrb <= "0101";
    elsif (CMDstrr) then  addrb <= IR(7 downto 4);
    else  addrb <= "0100";
end if;

-----SSHSCI-----
if (CMDjmpa or CMDpop or CMDmvrdr or CMDpopf or CMDret or CMDcala)
then  SSHSCI <= "001";
    else  SSHSCI <= "000";
end if;

-----I(8~6)-----
if (CMDldrr or CMDin or CMDstrr or CMDout) then  I(8 downto 6) <= "001";
    elsif (CMDpshf or CMDpush) then  I(8 downto 6) <= "011";
    else  I(8 downto 6) <= "010";
end if;

-----I(5~3)-----
if (CMDpshf or CMDpush)
then  I(5 downto 3) <= "001";
    else  I(5 downto 3) <= "000";
end if;

-----I(2~0)-----
if (CMDldrr) then  I(2 downto 0) <= "100";
    elsif (CMDin or CMDout)
    then  I(2 downto 0) <= "111";
    else  I(2 downto 0) <= "011";
end if;

-----SST-----
SST <= "000";

-----DC1-----
if (CMDin or CMDout)
then  DC1 <= "010";
    else  DC1 <= "000";
end if;

-----DC2-----
DC2 <= "011";

-----00100 -----
elseif (timing = "00100") then
-----MRW-----
MIO <= '0';
if (CMDin or CMDout)
then  REQ <= '1';
    else  REQ <= '0'; end
if;

if (CMDstrr or CMDpshf or CMDpush or CMDout)

```

```

        then WE <= '0';
        else WE <= '1';
    end if;

-----addrA-----
    if (CMDstrr or CMDpush)
        then addrA <= IR(3 downto 0);
        else addrA <= "0000";
    end if;

-----B 0-----
    if (CMDldrr or CMDpop or CMDmvrD) then addrB <= IR(7 downto 4);
    elsif (CMDret or CMDjmpa)
        then addrB <= "0101";
        else addrB <= "0000";
    end if;

-----SSHSCI-----
    SSHSCI <= "000";

-----I(8^6)-----
    if (CMDcala) then I(8 downto 6) <= "000";
    elsif (CMDjmpa or CMDldrr or CMDin or CMDpop or CMDmvrD or CMDret)
        then I(8 downto 6) <= "011";
        else I(8 downto 6) <= "001";
    end if;

-----I(5^3)-----
    I(5 downto 3) <= "000";

-----I(2^0)-----
    if (CMDpshf or CMDpopf) then I(2 downto 0) <= "000";
    elsif (CMDstrr or CMDpush or CMDout)
        then I(2 downto 0) <= "100";
        else I(2 downto 0) <= "111";
    end if;

-----SST-----
    if (CMDpopf) then
        SST <= "010";
    else
        SST <= "000";
    end if;

-----DC1-----
    if (CMDpshf) then
        DC1 <= "011";
    elsif (CMDstrr or CMDpush or CMDout) then
        DC1 <= "001";
    else
        DC1 <= "000";
    end if;

-----DC2-----
    DC2 <= "000";

-----00111-----
    elsif (timing = "00111") then

```

```

MIO <= '1';
REQ <= '0'; WE
    <= '0';

addra <= "0000";
addrb <= "0100";
SSHSCI <= "000";
I <= "011001011";
SST <= "000";
DC1 <= "000";
DC2 <= "011";

```

----- 00101 -----

```

elsif (timing = "00101") then
    MIO <= '0';
    REQ <= '0';
    WE <= '0';
    addra <= "0101";
    addrb <= "0101";
    SSHSCI <= "000";
    I <= "010000010";
    SST <= "000";
    DC1 <= "001";
    DC2 <= "000";
end if;

else
    --在脱机状态下，使下列控制信号处于高阻态
    addra <= "ZZZZ";
    addrb <= "ZZZZ";
    SSHSCI <= "ZZZ";
    I <= "ZZZZZZZZ";
    SST <= "ZZZ";
    DC1 <= "ZZZ";
    DC2 <= "ZZZ";
end if;
end process;

```

end Behavioral;

A.4 FPGA_CPU 教学实验

用FPGA芯片可以完成两个方面的教学实验,一个是围绕着设计实现CPU系统展开,另一个是围绕着组合逻辑或者时序逻辑的线路展开。

1. 设计实现CPU系统是计算机组成原理课程和计算机系统结构课程的核心教学实验内容。就这台实验设备而言,首要任务是看懂已经实现基本指令的CPU系统的设计过程和相关技术,在此基础上将可以开展添加扩展指令的教学实验,如果实现了全部指令,将可以运行BASIC程序。设计过程中,TEC-XP16的组成与设计内容是重要的参考资料。

用这个芯片也可以实现微程序的控制器,不妨从基本指令系统开始,之后添加扩展指令,最终同样能运行BASIC程序。设计过程中,TEC-XP16的组成与设计内容是重要的参考资料。

用这个芯片设计实现8位字长的教学计算机系统是方便可行的,包括微程序控制器和组合逻辑控制器两种实现方案。设计过程中,TEC-XP16系统同时支持的8位字长的教学计算机系统(TEC-XP16原8)的组成与设计内容是重要的参考资料。

请注意,TEC-XP+产品把TEC-XP16和TEC-XP(FPGA)两个计算机系统合并到同一块印制电路板上实现是一项很有创意的工作,在一些教师对选用FPGA芯片设计实现CPU系统的情况下,参照TEC-XP16系统的组成和设计来完成这项工作变得更为轻松一些。完全仿照TEC-XP16来设计和实现一个CPU系统是建立在特定目标的基础上,诸如保证软件兼容,硬件结构易于理解,教学内容有较好的延续性等。如果完全重新设计是按实现一个全新硬软件系统的计算机也是方便和可行的,例如MIPS16e计算机系统,只是工作量比较大,技术上并没有任何难度。

2. 用FPGA芯片完成EDA方面的教学实验是方便和可行的,包括简单的组合逻辑的电路系统合时序逻辑的电路系统,或者更为复杂的数字电路系统。CPU设计就是一个很好的例子,设计实现其他的数字电路系统同样很方便,既可以使用功能强大的仿真软件完成调试,也有为FPGA芯片准备的几十个指示灯。我们准备在一段时间之后就提供开展EDA教学实验的例子。

附录 2 教学计算机的软件模拟系统

教学计算机的软件模拟系统,包括指令层次与系统层次的模拟、微体系结构层次的组合逻辑控制器模拟、微程序控制器模拟3 部分内容,3 项功能各由1 个程序实现,将在下面的3 节中分别说明它们的启动方式和操作过程。这里给出的是用于原 TEC-2000 计算机的模拟系统,用于 TEC-XP+ 系列教学机的模拟系统仍在改进之中,二者差异很小。

教学计算机指令与系统级模拟程序,是以一条指令的执行功能为最小模拟单位,主要用于学习指令系统和程序设计的知识,熟悉教学计算机监控程序的功能和操作方法。在进入监控程序模拟运行状态之后,可以通过监控命令模拟教学计算机的运行功能,体现出来的运行特性与运行实际教学计算机硬件系统非常接近,能够以连续方式执行程序,也能以单指令方式逐条执行指令,支持汇编语言程序设计,支持 BASIC 高级语言程序设计。微体系结构级的模拟是以指令的一个执行步骤的功能为最小模拟单位,主要用于调试、观察教学计算机的微程序控制器设计和硬连线控制器设计的正确性,帮助实验人员更深入地理解指令的执行过程,给出的是计算机硬件系统内部信息传送的时空关系和运行状态。

B. 1 教学计算机的指令与系统级的模拟

B. 1. 1 指令模拟的概念和在教学中的作用

计算机指令是程序员使用计算机硬件的基本命令,也是计算机本身运行的最小的功能单位。CPU能执行的不同指令的集合就是该计算机的指令系统。

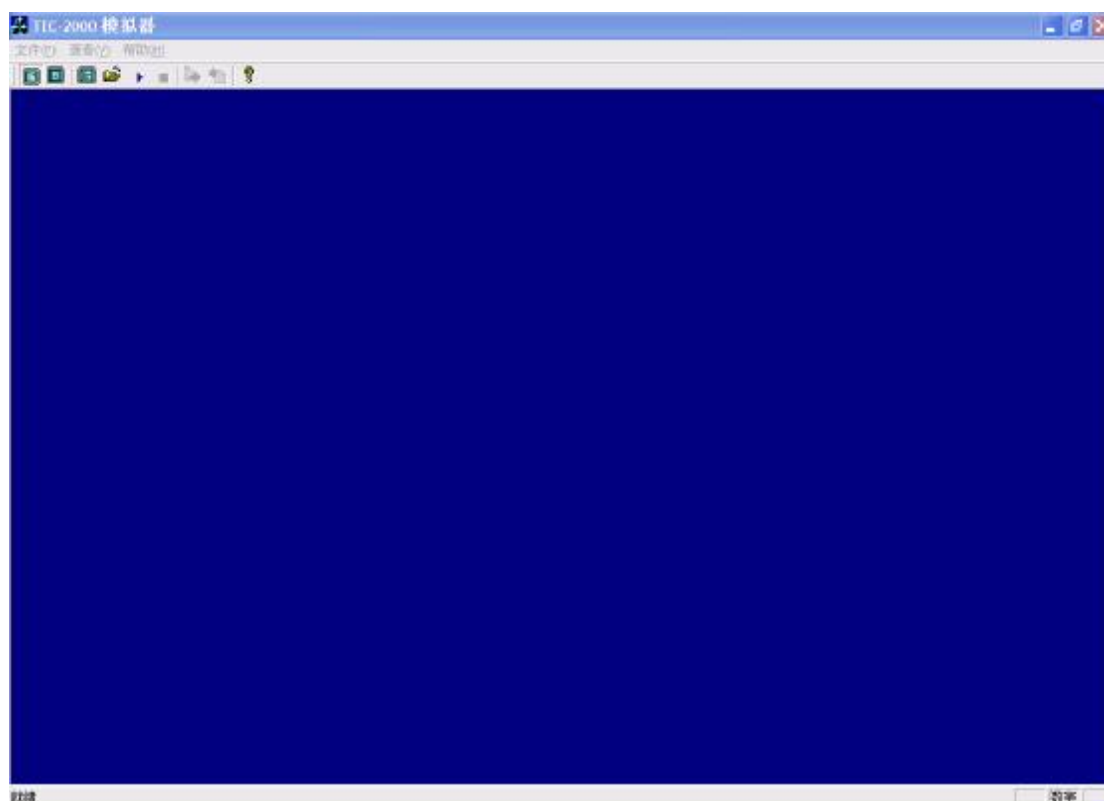
指令系统规定了 CPU 能完成的所有操作以及这些操作的硬件构成,它直接影响到计算机的性能,也决定了底层的硬件结构。因此,指令系统也常被称为计算机硬件和软件的接口。一方面,程序员通过指令系统中的指令编写程序来使用计算机,另一方面,计算机硬件也必须根据指令规定的功能,完成相应的操作。因此,掌握计算机指令系统的设计原则和实现原理,对学习计算机组成课程是十分重要的。

计算机指令由操作码和操作数地址组成。操作码定义了指令的功能,例如是完成加法操作或者是从存储器中读取数据;操作数地址和寻址方式一起指明了如何得到指令的操作数据。学习计算机指令系统,首先需要掌握定义指令操作码的原则和方法,其次,还需要掌握各种寻址方式的功能以及寻址过程。实验是学习和掌握指令系统的重要手段。通过在教学计算机上实际操作计算机,设计一些汇编语言程序并以不同方式运行,观察和记录各种显示数据和指示灯的输出,可以更具体了解指令的功能、寻址方式的实现方法,建立对指令功能和寻址方式的感性认识。

然而,直接在教学计算机上进行程序设计和指令系统的实验,也存在一些困难。首先,由于教学计算机属于专用的教学仪器,很难做到每个同学都有一台,实验要到实验室进行,对大家灵活安排学习时间不利。第二,教学计算机监控程序输出信息有限,许多输出信息需要通过观察指示灯来获得,不很直观。第三,对于扩展原有指令系统,在教学计算机上需要更改硬件设计及其实现,工作量大。

为了解决上述问题,我们开发出一套软件系统,来模拟和仿真 TEC-2000 教学计算机的运行过程。由于仿真程序可在 PC 机上运行,同学可以在自己的计算机上完成硬件模拟实验;同时,由于 PC 机比起教学计算机,具备更完备的输出设备,可直观输出全部寄存器、标志位,甚至是存储单元的内容,方便同学掌握程序运行的全部结果和状态;另外,全软件的设计,使指令的扩展操作只表现为对运行参数的改变,减少了实现工作量。在进行扩展指令的实验中,把已经模拟运行正确的设计结果加载到教学计算机的实际硬件系统中,可以极大地减少对硬件系统的反复调试与修改,明显地提高了工作效率。

要启动并运行指令与系统级模拟软件,可以到 PC 机系统中找到模拟软件所在的目录,运行 Tec2ksim.exe 程序,屏幕上将显示模拟器程序的主界面,如图B. 1 所示。



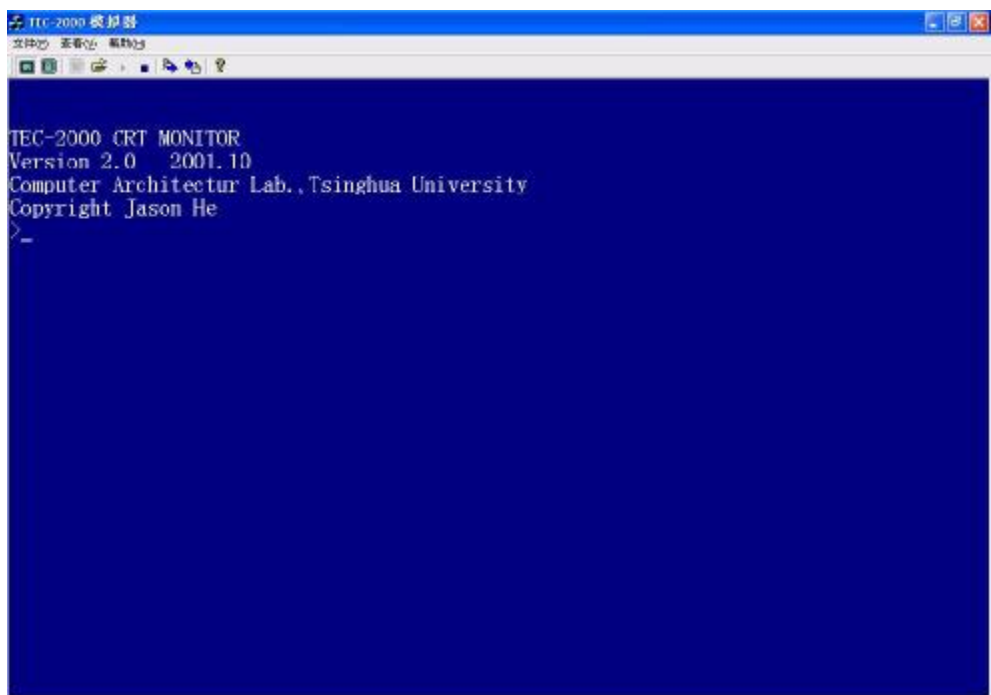
图B. 1 指令与系统级的模拟系统的主界面



图 B. 2 指令与系统级的模拟系统的功能选择界面




- | | |
|-------------------|------------|
| : 选择16 位计算机 | : 选择8 位计算机 |
| : 启动监控程序（16 或8 位） | : 导入程序 |
| : 运行代码 | : 停止 |
| : 发送文件 | : 接收文件 |
| : 帮助文件 | |

在主菜单上选择 和 ，是启动16 位教学机的监控程序，将出现如图B. 3 所示界面：

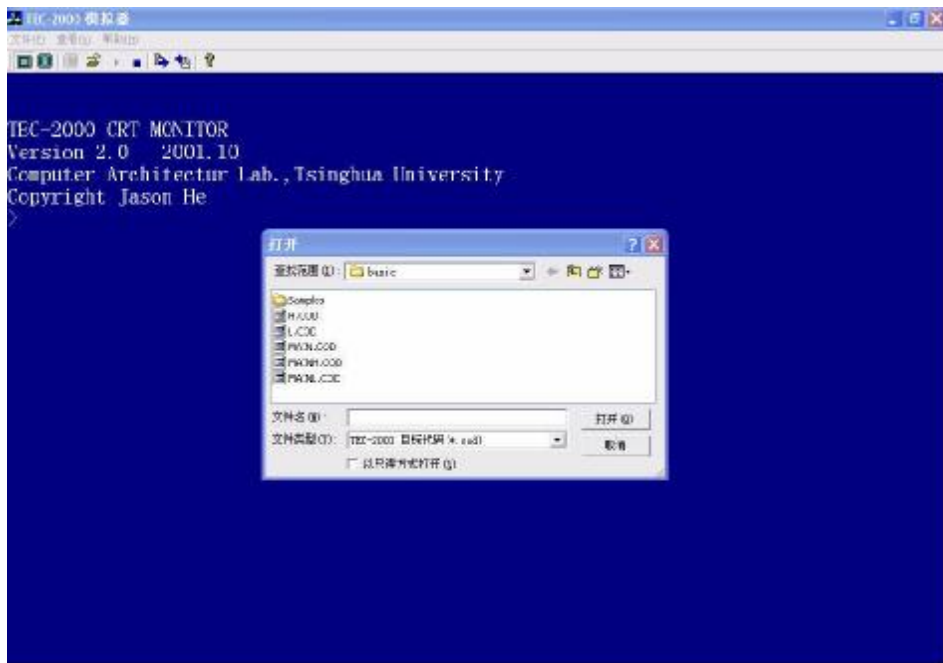


图B. 3 16 位教学计算机的运行界面

这之后就可以通过监控命令控制软件模拟系统的运行过程，和直接操作硬件教学计算机系统几乎是一样的。

可以用 、 这2 个按钮把机器语言程序(.cod 类型的二进制代码)写入到磁盘,把磁盘中已有的.cod 程序文件装入模拟系统。用  按钮可以把BASIC 语言的解释程序调入模拟系统,使系统进入BASIC 语言程序的运行环境,如图 B. 4 所示。

1. 首先需要导入BASIC 的COD 文件,选择MAIN.COD 如图B. 4 所示:



图B. 4 装载程序界面

装载程序成功则显示如图B. 5 界面:



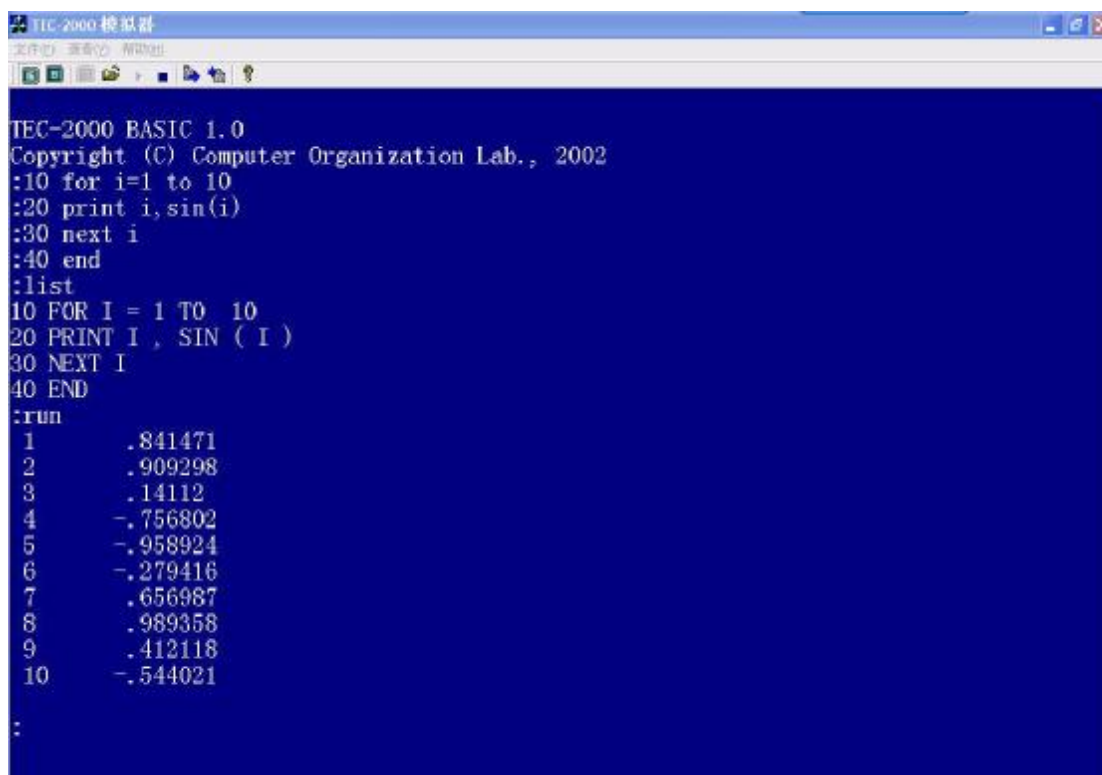
图B. 5 代码装载成功提示

2. 运行BASIC 程序（G0A30）


进入BASIC 的程序界面，可以送入BASIC 的命令或输入如下程序

```
10 FOR I=1 TO 10
20 PRINT I, SIN (I)
30 NEXT I
40 END
```

接下来可以用LIST 命令查看输入的程序，用RUN 运行，用SYSTEM 退出BASIC 等。



图B. 6 BASIC 界面

在 BASIC 的运行环境下，还可以用  按钮把磁盘中的.tba 类型的 BASIC 程序装入到模拟系统并运行之。

B. 1. 2 教学计算机程序设计实验（仿真） 在教学计算机软件模拟系统中进行的运行监控程序和程序设计方面的教学实验，需要在前面提到的实验环境中完成。本小节将说明这些实验的有关内容。

一、实验设备和环境要求

本实验的主要内容是在 PC 机上，使用教学计算机的模拟软件，模拟监控程序的运行过程。在 监控程序控制下，设计并运行一些简短的汇编语言程序，了解 TEC-2000 教学计算机的指令系统和 程序设计方法。因此，实验设备为PC 计算机，主要环境为 Windows 操作系统，安装了教学计算机 指令与系统级模拟软件。

1. 二、实验目的
2. 学习和了解TEC-2000 16 位教学计算机监控命令的用法。
3. 学习和了解TEC-2000 16 位机的指令系统。
4. 学习简单的TEC-2000 16 位机汇编程序设计。

三、实验要求与说明

进行实验之前，应基本了解 TEC-2000 16 位教学计算机的指令系统和每条指令的功能，熟悉程序设计软件的基本使用流程，并预先设计好需要进行实验操作的汇编语言程序。

四、实验步骤

1. 安装软件模拟系统。
2. 启动软件模拟系统，调出程序的主界面。
3. 选择“16 位机”系统，启动监控程序。
4. 输入并执行各监控命令，操作步骤和方法与运行硬件教学计算机系统非常类似。
5. 在监控程序运行界面，使用监控程序的 R 命令显示/修改寄存器内容、D 命令显示存储器内容、E 命令修改存储器内容。
6. 使用 A 命令输入并汇编以下几个汇编程序的例子，用 U 命令反汇编刚输入的程序，用 G 命令连续运行该程序，用 T、P 命令单指令运行并观察其执行情况。
7. 自行设计几个汇编语言程序，使用多种寻址方式，仔细体会这些寻址方式的指令格式，并思考如何在硬件上实现这些指令的功能。

例1：设计一个程序，在屏幕上输出显示字符‘6’。

```
A 2000          ; 地址从16进制的2000（内存RAM区的起始地址）开始
2000: MVRD R0, 0036      ; 把字符‘6’的ASCII码送入R0
2002: OUT 80             ; 在屏幕上输出显示字符‘6’，80为串行接口地址
2003: RET                ; 每个用户程序都必须用RET指令结束
2004: (按回车键即结束源程序的输入过程)
```

这就建立了一个从主存2000h 地址开始的小程序。在这种方式下，所有的数字都约定使用16 进制数，故数字后不用跟字符 h。每个用户程序的最后一个语句一定为 RET 汇编语句。因为监控程序是选用类似子程序调用方式使实验者的程序投入运行的，用户程序只有用 RET 语句结束，才能保证用户程序运行结束时能正确返回到监控程序的断点，保证监控程序能继续控制教学机的运行过程。

下面接着再给出几个程序的例子。

例2：设计一个程序，用次数控制在终端屏幕上输出‘0’到‘9’十个数字符。 A 2020

```
MVRD R2, 000A      ; 送入输出字符个数
MVRD R0, 0030      ; ‘0’ 字符的ASCII码
OUT 80              ; 输出保存在R0低位字节的字符
DEC R2              ; 输出字符个数减1
JRZ 202E            ; 判10个字符输出完否，已完，则转移到程序结束处
PUSH R0              ; 未完，保存R0的值到堆栈中
(2028) IN 81         ; 查询接口状态，判字符串行输出完成否
SHR R0              ;
JRNLC 2028           ; 未完成，则循环等待
POP R0              ; 已完成，准备继续输出下一字符，从堆栈恢复R0的值 INC
R0                  ; 得到下一个要输出的字符
JR 2024              ; 转去输出字符
(202E) RET
```

这个程序只使用基本汇编语句。理解中的一个难点，是程序当中判串行口是否完成一个字符的输出过程并循环等待的三个汇编语句。具体解释见教材《计算机组成与设计》中的有关串行接口的内容。

该程序的执行码放在2020 起始的连续内存区中。若送入源码的过程中有错，系统会进行提示，等待重新输入正确汇编语句。输入过程中，在应键入语句的位置直接输回车则结束输入过程。

接下来可用G 2020 命令运行该程序。

思考题：当把IN 81, SHR R0, JRNC 2028三个语句换成4个MVR R0, R0语句，该程序执行过程会出现什么现象？试分析并实际执行一次。

类似的，若要求在终端屏幕上输出‘A’到‘Z’共26个英文字母，应如何修改例中给出的程序？请验证之。

例3：从键盘上连续键入多个属于‘0’到‘9’的数字并在屏幕上显示，遇非数字结束程序。从地址2040开始输入下列程序：

```
A 2040
    MVRD R2, 0030      ; 用于判数字的下界值
    MVRD R3, 0039      ; 用于判数字的上界值
(2044) IN 81           ; 判键盘上是否按了一个键
    SHR R0             ; 即串行口是否有了输入的字符
    SHR R0
    JRNC 2044          ; 尚没有输入则循环测试
    IN 80              ; 输入字符读到R0低位字节
    MVRD R1, 00FF
    AND R0, R1         ; 将R0的高位字节清0
    CMP R0, R2         ; 判输入字符≥字符‘0’ 否
    JRNC 2053          ; 为否，则转到程序结束处
    CMP R0, R3         ; 判输入字符≤字符‘9’ 否
    JC 2053            ; 为否，则转到程序结束处
    OUT 80             ; 输出刚输入的数字字符
    JMPA 2044          ; 转去程序前边2044处等待输入下一个字符
(2053) RET
```

思考题，本程序中为什么不必判别串行口输出完成否？设计读入‘A’～‘Z’和‘0’～‘9’的程序，遇其它字符结束输入过程。

例4：计算1到10的累加和。 A

```
2060
    MVRD R1, 0000      ; 置累加和的初值为0
    MVRD R2, 000A      ; 最大的加数
    MVRD R3, 0000
(2066) INC R3          ; 得到下一个参加累加的数
    ADD R1, R3         ; 累加计算
    CMP R3, R2         ; 判断是否累加完
    JRNZ 2066          ; 未完，开始新一轮累加
    RET
```

运行过后，可以用R 命令看R1 中的累加结果。

例 5：设计一个有读写内存和子程序调用指令的程序，功能是读出指定内存中的大写字母字符，将其显示到屏幕上，转换为小写字母后再写回存储器原存储区域。

E 20F0 （送入将被显示的6 个字符‘A’～‘F’到内存20F0 开始的存储区域中）

41 42 43 44 45 46

```
A 2080
    MVRD R3, 0006      ; 指定被读数据的个数
    MVRD R9, 20F0      ; 指定被读、写数据内存区首地址
```

```

(2084) LDRR  R0,    [R2]    ; 读内存中的一个字符到R0寄存器
      MVRD  R8,    2100    ; 指定子程序地址为2100
      CALR  R8                ; 调用子程序,完成显示、转换并写回的功能 DEC
      R3                    ; 检查输出的字符个数
      JRZ   208C            ; 完成输出则结束程序的执行过程
      INC  R2                ; 未完成,修改内存地址
      JR    2084            ; 转移到程序的2084处,循环执行规定的处理
(208C) RET

A 2100                                ; 输入用到的子程序到内存2100 开始的存储区
      OUT  80                ; 输出保存在R0寄存器中的字符
      MVRD R1,    0020    ; 转换保存在R0中的大写字母为小写字母
      ADD  R0,    R1
      STRR [R2], R0        ; 写R0中的字符到内存,地址同LDRR所用的地址
(2105) IN   81                ; 测试串行接口是否完成输出过程
      SHR  R0
      JRNC 2105            ; 未完成输出过程则循环测试
      RET                    ; 结束子程序执行过程,返回主程序

```

运行过程中,可以直接看到屏幕上显示的内容,运行过后,再用D 20F0 命令看内存的20F0 区域中保存的运行结果:

```
0061 0062 0063 0064 0065 0066
```

上述5 个例子,都是用监控程序的A 命令完成输入源汇编程序的。在涉及到汇编语句标号的地方,不能用符号表示,只能在指令中使用绝对地址。使用内存中的数据,也由程序员给出数据在内存中的绝对地址。显而易见,对这样的短小程序矛盾并不突出,但很容易想到,当设计很大的程序时,一定会有较大的困难。

五、实验报告要求

实验报告中,大家应对照实验目的和要求,记录实验过程和实验结果,总结在汇编语言中使用的寻址方式和指令格式,指令的执行过程。另外,如有可能,谈谈在实际的 TEC-2000 教学计算机上完成这个实验和在软件模拟环境下完成的实验有什么相同和不同之处,以及你对这种实验手段的看法。

B. 2 教学计算机微体系结构级组合逻辑控制器模拟系统

计算机控制器是Von Neumann 结构计算机的五大功能部件之一,其作用是向计算机的每个功能部件(包括控制器本身)提供协同运行所需要的控制信号。这些控制信号是根据计算机指令生成的,每条指令的执行都需要被划分成几个执行步骤来完成。从这个角度看,控制器的功能是依据当前正在执行的指令和指令的执行步骤,形成并提供当前整机各部件要用到的控制信号。

控制器的实现有两种主要途径,一种是微程序的控制器,另一种为组合逻辑控制器。微程序控制器将全部指令的每个执行步骤所需要的全控制信号保存在微程序的控制存储器中,并在控制信号中明确给出下一条使用的微指令的地址,这样,控制器可通过读取微程序的控制存储器来得到指令的控制信号。组合逻辑控制器则通过节拍状态的转换来完成指令的不同步骤间的衔接,并根据指令操作码和节拍,由逻辑电路直接形成并提供全控制信号。

B. 2. 1 计算机微体系结构级模拟的概念和在教学中的作用

控制器是整台计算机的指挥、协同机构,有着举足轻重的地位,也是学习计算机组成原理的重

点和难点。从难点上看,主要体现在,控制器需要产生的众多控制信号,其产生的机理是什么,应该有比较直观和具体的手段或装置体现出来;另外,控制器如何连续地执行指令步骤,也需要使同学们有具体认识。而这些,只能是通过实验手段来获取,在实验中建立起感性认识,并通过理论学习,提升到理性认识。

作为控制器教学实验,一是在一个已经具备基本指令系统的CPU 上,完成对指令的扩充,设计新的指令并调试和运行;二是在一个部件齐全的教学计算机上,设计全部指令并实现,检验设计正

确定性和可靠性的标志是使教学计算机的监控程序能正确、稳定地运行。当然,如果能让同学们白手起家,完全自行设计出新的CPU 和指令系统,并证明指令系统的可靠性和正确性,则更具有挑战性,但实验负担会更重。

TEC-2000 教学计算机就是这样一个实验装置,能支持上述前两种实验方式。经过适当的改造,也可支持最后一种方式。首先,TEC-2000 教学计算机上本身是一台硬软件配置齐全,功能完备的实验用计算机,已设计好16 位的指令系统和对应的监控程序。同学们实验时,可扩充现有的指令系统,增加自行设计的新指令,并使监控程序能识别这些指令,对新指令进行汇编,使教学计算机的功能得到扩充。也可以在教学计算机上,设计一套新的、指令长度为8 位的指令系统,并使其与之对应的监控程序能正确、稳定地运行。

完全进行硬件上的实验和调试,对刚开始接触计算机硬件实验的同学来说,可能比较复杂。多种情况交织在一起,也给实验增加了难度。同学们对软件系统接触比较多,而且,程序可以采取多种大家熟悉的单步跟踪、断点等方式进行调试,比较直观。因此,如果能先在一台软件模拟的教学计算机上,解决控制器实验的逻辑设计错误,确保设计正确性,而在硬件调试上,主要解决硬件故障等带来的问题,则可简化整个实验。同时,软件模拟中,可及时修改错误,而不牵涉到硬件芯片的烧制,也可缩短实验的进程。另外,软件模拟可跟踪到具体的微操作,对加深同学们的理解也有帮助。

因此,除提供教学计算机硬件外,我们还设计了对TEC-2000 的微体系结构级的软件模拟系统。系统由两个程序组成,可以分别对微程序控制器和组合逻辑控制器进行模拟。同学们设计好自己的指令系统的微程序后,可加载到微程序模拟程序中,进行相应的调试,发现其中的逻辑错误并设法改正。而对组合逻辑控制器,也可在编写完节拍和逻辑信号的表达式后,加载到组合逻辑模拟程序中进行检查和调试。经验证无误后,可基本证明设计的逻辑正确性。对因为种种原因限制而无法进行硬件实验的同学,通过在模拟软件上的设计和实验,也可基本掌握控制器的运行原理,对其有一个基本的认识。

TEC-2000 微体系结构级模拟软件系统由两个程序组成,一个是针对微程序控制器设计的,能对用户自行设计的微程序进行模拟,若设计的微程序符合 TEC-2000 教学计算机硬件运行要求,就可以在模拟器上正确运行 TEC-2000 的机器语言程序。另一个是针对组合逻辑控制器设计的,它装入用户设计的节拍和控制信号产生的表达式后,也可完成对这些设计的检查和仿真运行。两个程序均运行在Windows 环境下。需要指出的是,不管是设计全套新指令还是对指令系统的进行扩展,由于

微体系结构实验需要设计出指令的全部控制信号,因此,必须在实验前设计出全部完整的微程序(微程序控制器)或是组合逻辑的表达式(组合逻辑控制器)。

B. 2. 2 教学计算机组合逻辑控制器软件模拟系统使用说明

TEC-2000 教学计算机的软件模拟系统,是为模拟教学计算机的指令系统(和微体系结构运行状态)而设计的。它运行在Windows 平台上,能模拟教学计算机指令的运行过程。这样,同学们可在脱离教学计算机硬件的情况下,使用本系统来完成教学计算机程序设计方面的实验,熟悉和了解教学计算机的指令格式,寻址方式等,可自行编写教学计算机汇编语言程序,并在系统中进行跟踪调试,给出的运行结果更直观,方便同学们理解实验的目的。

整个系统功能完善,可以进行从汇编语言程序编辑、交叉汇编、机器语言程序调试和完整程序运行的全部的模拟过程,下面介绍系统的使用。

一、组合逻辑控制器软件模拟系统的程序操作三界面

在安装好TEC-2000 教学计算机模拟软件系统的PC 机上,选择TEC-2000 16 位教学计算机组合逻辑仿真软件打开后,将出现如下图5. 7 所示界面。

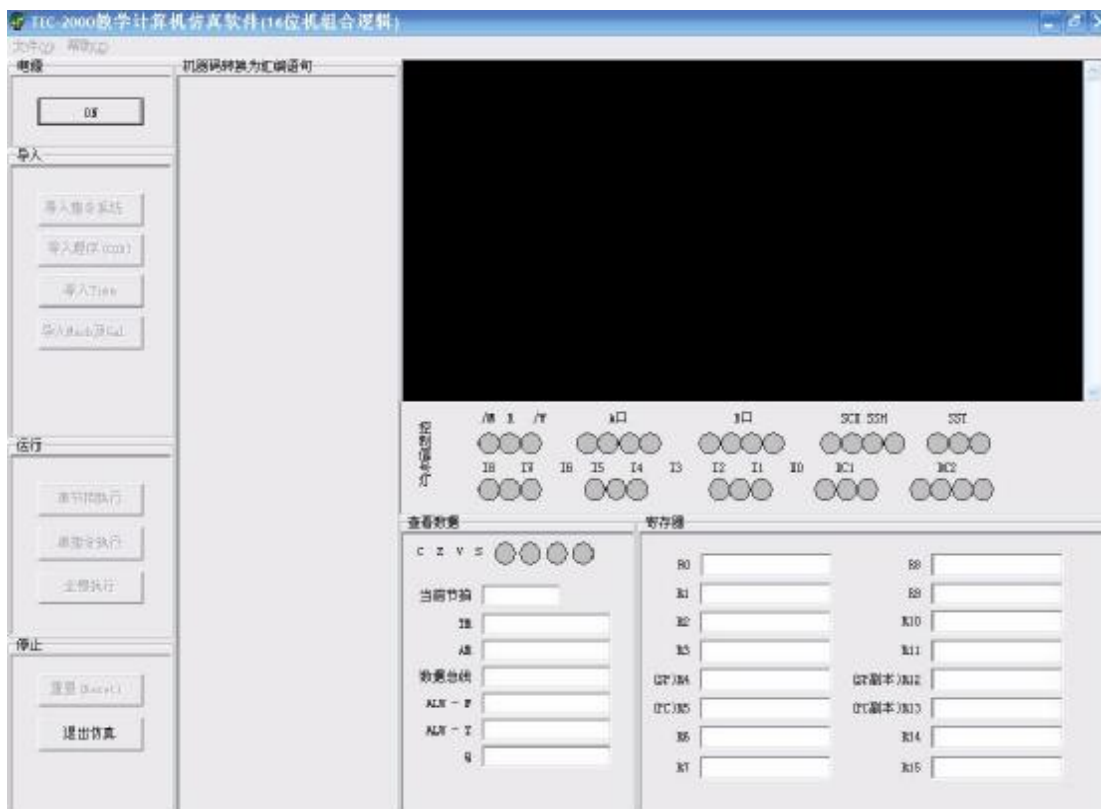


图 B. 7 TEC-2000 16 位组合逻辑控制器模拟主界面

界面的最左边侧是功能区设置了模拟软件系统的主要功能按钮包括启动系统按钮(OFF/ON)，需要导入系统的软件资源选择按钮，这些资源包括指令系统、将要运行的程序、节拍发生器 Time 的表达式、组合逻辑控制信号的表达式（包括MACH 和GAL）指令的执行方式（单节拍执行、单指令执行、全部执行）选择按钮、重置（Reset）和退出仿真按钮等。中间部分是用户程序区，用于显示用户运行和调试的程序，供调试过程中查看。右侧的上半部分是模拟系统的输出屏幕窗口，可看成是 TEC-2000 教学计算机模拟系统的显示器。右侧的中间和下半部分是模拟程序的输出窗口，包括3 部分：①组合逻辑控制器的34 位控制信号的状态；②当前指令（IR）内容、当前节拍状态、ALU 的输出结果、标志位状态、地址总线内容（DR）数据总线内容；③各通用寄存器的值。上述全部信息揭示了计算机系统内部的运行状态。

使用模拟程序的基本目的，是以软件仿真方式运行用户的程序，以检验组合逻辑控制器设计的设计结果是否正确。因此，运行程序之前需要导入相关的资源，通过资源选择按钮来启动这些操作功能。

“导入指令系统”按钮，导入TEC-2000 模拟软件的data 文件内容。“导入程序”按钮，若导入TCE-2000 的指令系统，就可以加载TEC 2000 的监控程序，类似于计算机小的操作系统。

“导入Time”按钮，导入节拍发生器的逻辑表达式。

“导入Mach 及Gal”按钮，导入Mach 控制器的内容及各个gal 的表达式的内容。

运行（控制框）“全部执行”按钮，将连续执行完整的程序，结束后显示程序的最终运行结果。

“单节拍执行”按钮，每次将只执行一个节拍，并在状态窗口中显示该指令执行完成后的状态信息，34 个指示灯将工作。继续按“单步执行”按钮，将继续执行下一个节拍。“单指令执行”按钮，每次将只执行一条指令，并在状态窗口中显示该指令执行完成后的状态信息。继续按“单步执行”按钮，将继续执行下一条指令。

启动和停止（控制框）

启动 (ON /OFF) 按钮，按一下后显示OFF，激活其他的按钮功能。“退出仿真”按钮，退出模拟软件系统。

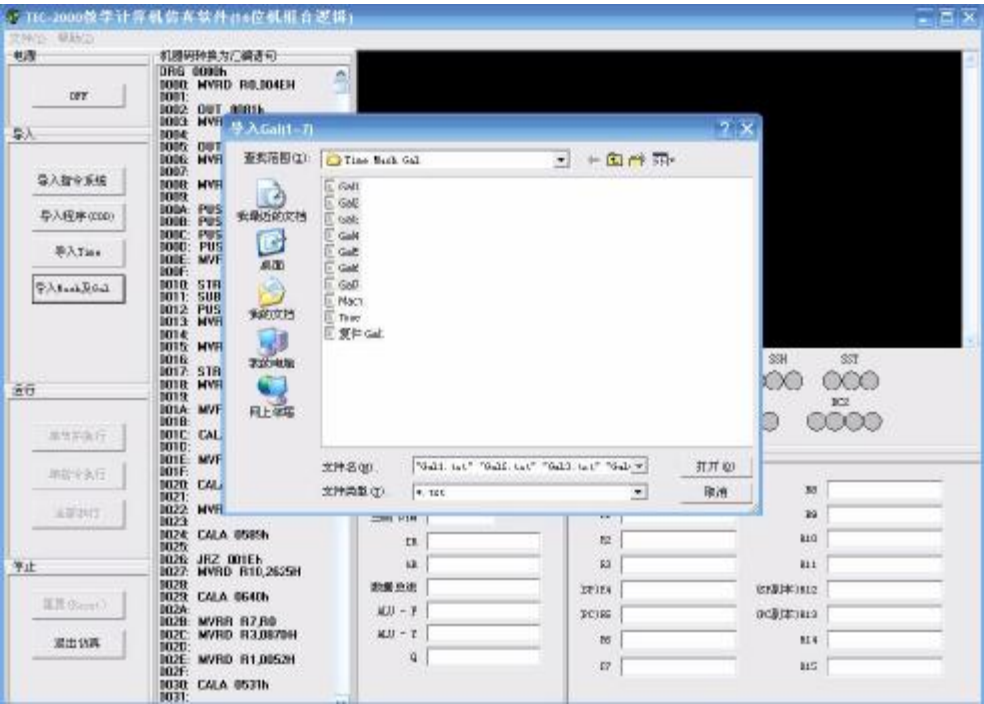
概述基本操作步骤如下：

首先按图5. 7 界面上的“ON”按钮，激活其他的按钮功能。再依次按“导入指令系统”“导入程序”“导入Time”和“导入Mach 及Gal”按钮，可将指定的文件装入到系统中。其中，“导入Mach 及Gal”将先导入Mach 文件，然后将提示继续导入Gal 文件，可选择7 个Gal 文件同时导入。

导入操作举例：

导入Mach 及gal 文件双击导入Mach 及Gal”按钮将出现如下所示界面继续选择gal1—gal7 然后添加则添加完毕。以后就可以单节拍运行，单指令运行及全部指令运行。

这样，运行组合逻辑控制器需要的全部资源已经导入系统，可以进入模拟运行各种机器语言程序的过程。

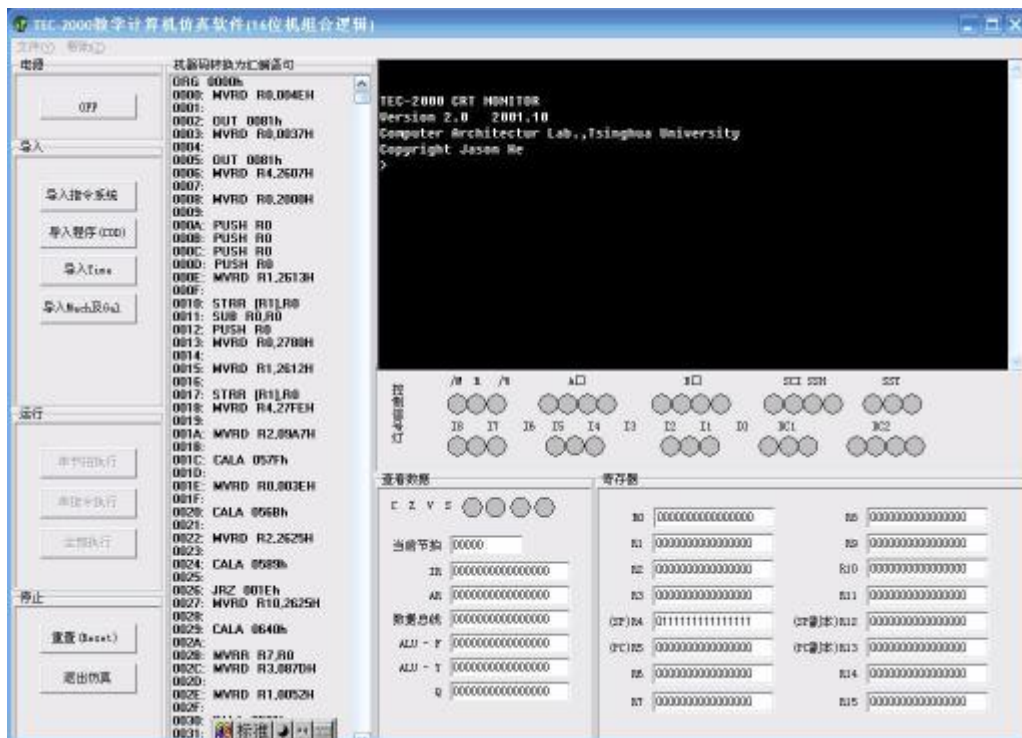


图B. 8 导入Mach 及gal 文件

二、运行程序的操作步骤

可以选择系统中已准备好的程序，使其进入模拟运行过程，当选择监控程序时，系统将进入监控程序的执行过程，此时，PC 机的键盘和模拟系统的屏幕窗口作为仿真系统的输入输出设备使用。

在主菜单上选择运行框内的“全部执行”按钮，将出现如下所示界面：



图B. 9 导入监控程序后显示的提示界面 接下来可以在用键盘输入各种监控命令，监控命令及其执行结果将显示在屏幕区窗口。
若要导入其他的程序，则按“重置 (Reset)”按钮，再选择“导入程序”，将出现选择要运行的机器语言程序如下所示：

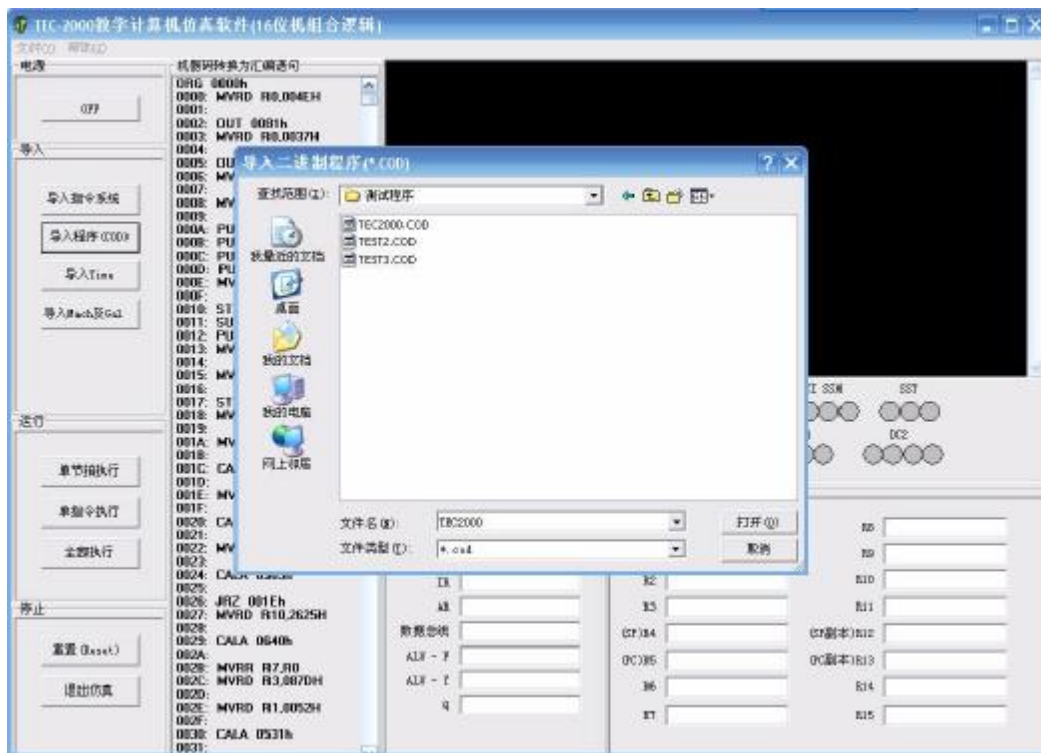


图 B. 10 导入程序界面 选定程序后，可按“运行”框中的按钮，程序开始运行并在屏幕窗口中显示输出信息。

在选择“单节拍执行”功能或“单指令执行”功能时，在寄存器窗口将显示这些寄存器的值，在信号灯窗口将显示相关控制信号的值，在数据框中，将显示节拍状态、指令寄存器、地址寄存器、

ALU 运算结果等内容，显示界面如图B. 11 所示。

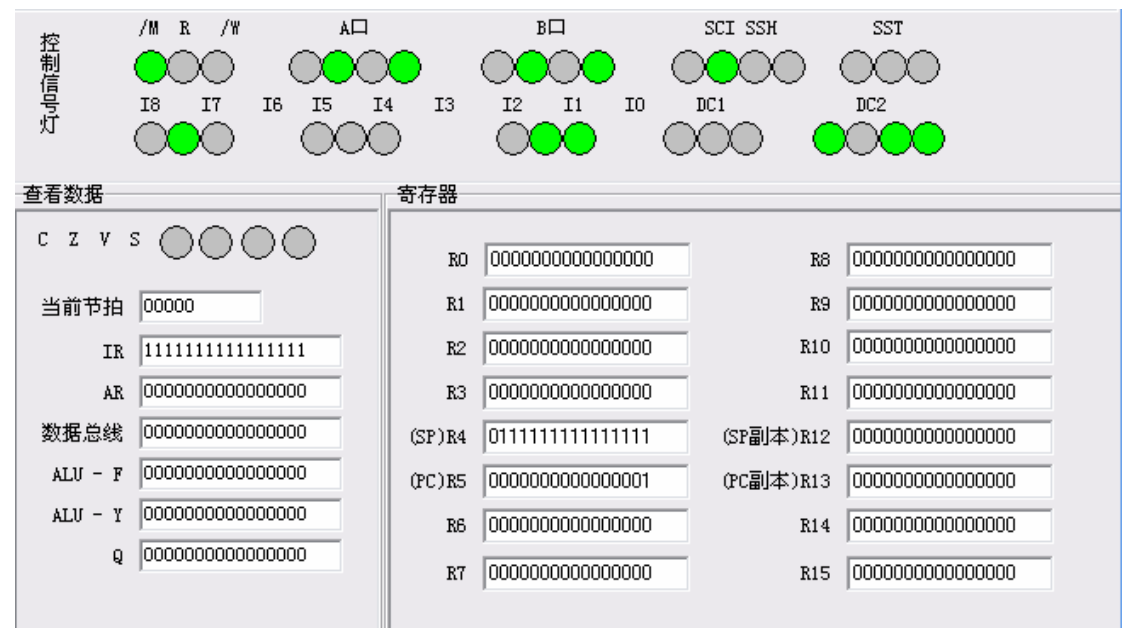
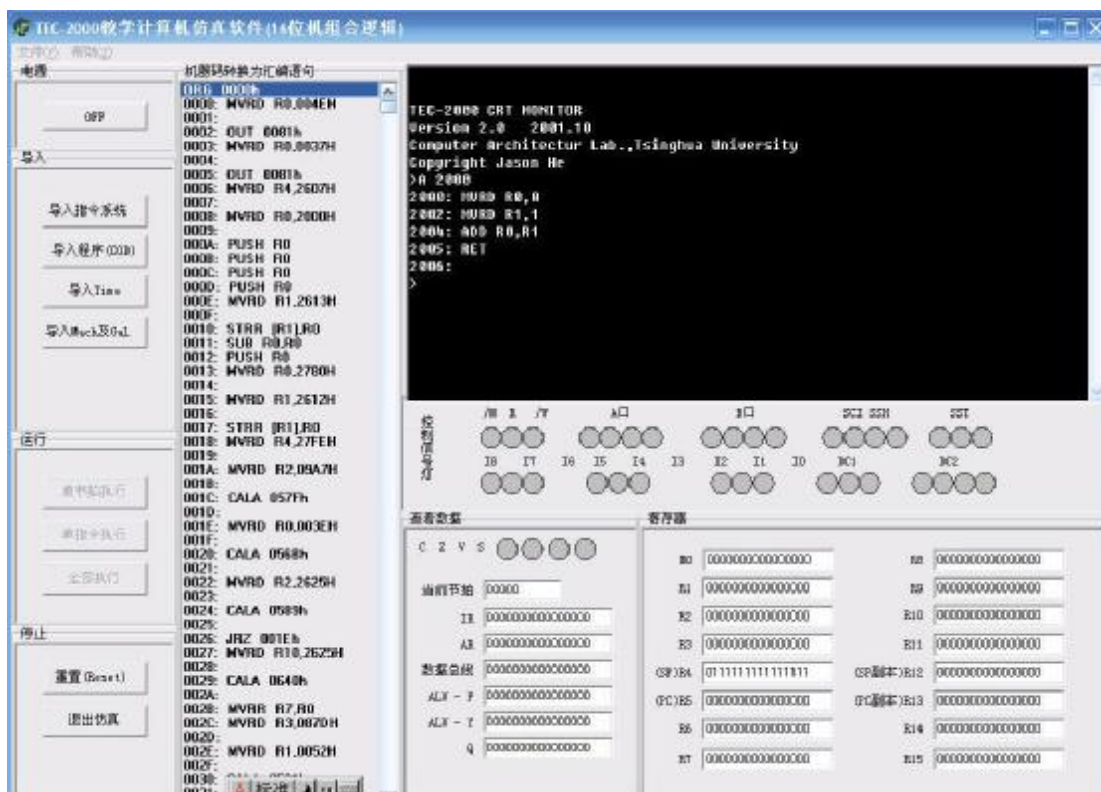


图 B. 11 控制信号灯、数据和寄存器的值

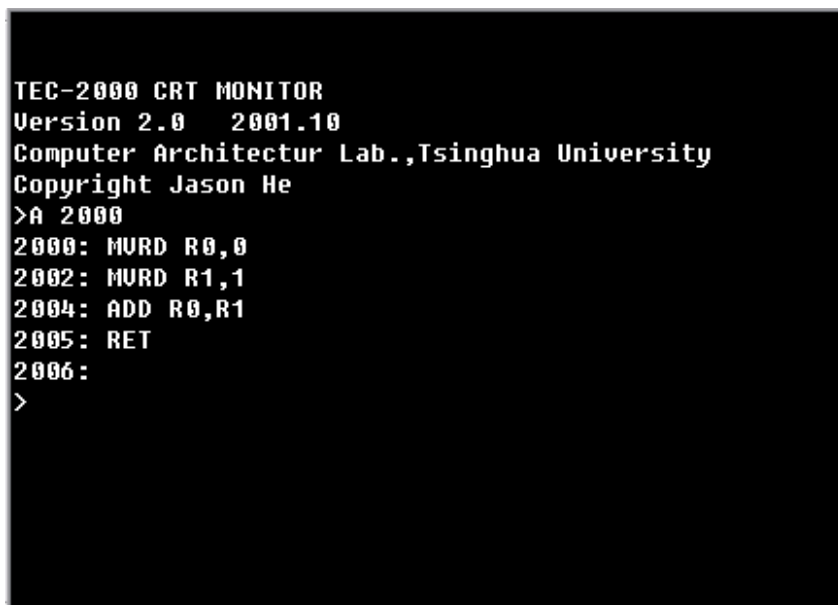
为检查、验证组合逻辑控制器设计方案的正确性，需要对它实现的指令系统中全部指令进行模拟运行，检查运行结果是否正确。这一过程可用简单的机器语言程序来完成，如我们可以用如下程序来验证MVRD、ADD 及RET 三条指令的正确性。

```
MVRD R0, 0
MVRD R1, 1
ADD R0, R1
RET
```

将指令系统中所有指令都验证完后，可得到组合逻辑控制器方案基本设计正确的结论。本系统支持已经汇编好的16 位机的机器语言程序。用户可将设计的汇编语言程序，通过交叉汇编等程序汇编成机器语言程序后进行检测。如有可能，也可直接设计机器语言程序。(如对一些自行设计的指令，则只能直接输入机器语言程序并调试) 按“导入程序”按钮后，将指定的机器语言程序读入到系统中，如图B. 12 所示。图B. 13 给出的是图 B. 12 中的屏幕区窗口的内容，文字大了一点，可以看得更清楚。测试程序装入后，就可以使用“单节拍执行”、“单指令执行”和“全部执行”几个按钮来执行刚装入的机器语言程序。其中，“单节拍执行”仅执行当前指令的当前节拍，输出该节拍运行后的状态。“单指令执行”输出当前指令运行后的程序状态。“全部执行”时，将不再输出这些信息。用户可通过对程序运行过程的模拟，发现设计方案中的错误，并由此修改设计方案。



图B. 12 装入机器语言程序



图B. 13 被装入的机器语言源程序

B. 2. 3 使用模拟程序完成组合逻辑控制器实验

计算机组合逻辑控制器实验,要求在教学计算机已有基本指令系统的基础上,由同学自行添加若干条新的指令,包括定义指令功能、格式,划分指令执行步骤和确定每一步骤的功能,确定每一执行步骤使用的全部控制信号的状态值,然后,根据新的设计结果,重新编写控制信号的逻辑表达式,装入到计算机硬件系统中,并写一个包含有新指令的程序,检查运行结果的正确性。由于硬件环境比较复杂,调试难度大,实验要求比较高。

为降低实验难度,可以先在模拟软件上完成组合逻辑控制器扩展指令的实验,包括设计实现和调试运行两部分工作,得到正确的结果后,再转到硬件系统上完成实验,这可以快速排除设计中的

各种错误，转而集中精力解决硬件故障问题。在硬件实验条件比较欠缺的情况下，也可以通过软件模拟系统中的实验，基本掌握组合逻辑控制器的基本原理，我们提供这种方案，但不推荐此种实验方式，已经有很多单位，也包括清华大学计算机系，都曾吃过硬件课程过分软化的亏。下面介绍使用模拟程序完成组合逻辑控制器实验的基本要求和步骤。

一、实验设备和环境

本实验的主要内容是在 PC 机上，使用教学计算机的模拟软件，模拟组合逻辑控制器的运行过程。因此，实验设备为 PC 计算机，主要环境为 Windows 操作系统，安装教学计算机组合逻辑模拟软件。

如何启动组合逻辑控制器的模拟软件，如何导入系统运行所需要的资源，如何选择运行的程序等，已经在教材 5.2 节进行了必要的说明，不在这里赘述。这里只给出组合逻辑控制器教学实验本身的内容。

a) 实验目的

8. 进一步熟悉教学计算机指令格式、指令编码、寻址方式和指令功能；
9. 进一步熟悉教学计算机的总体组成和各部件的运行原理，理解指令的执行过程；
10. 通过对指令系统的扩展，了解组合逻辑控制器设计和实现的基本过程；
11. 思考和讨论组合逻辑控制器的特点并与微程序控制器进行比较。

二、实验要求

1. 通过预习，熟悉教学计算机组合逻辑控制器模拟系统的使用；
2. 确定实验中要扩展的指令的功能、编码、格式和寻址方式；
3. 设计新指令的节拍状态和控制信号，并将其加入到原有的指令系统表中；
4. 根据新的指令系统表改写控制信号逻辑表达式，将新的逻辑表达式和指令系统表装入到模拟系统中，并验证其正确性，尤其是扩展指令的正确性；
5. 进行实验总结。

三、实验步骤

下面以在 TEC-2000 教学计算机组合逻辑控制器的现有基础上，扩展指令 ADC DR,SR 为例，说明实验的步骤。

5. 指令功能和格式的确定

ADC DR,SR 指令的功能，是将 SR 中的值用作源操作数、DR 的值用作目的操作数，以状态寄存器中 C 标志位的值作为最低位的进位输入完成求和运算，运算结果存入目的寄存器 DR 中，即 $DR \leftarrow DR + SR + C$ 。

指令格式可确定为操作码（8 位）DR（4 位）SR（4 位）由于它的功能在读取指令之后可一步完成，属于 A 组指令，分配操作码为 00100000。指令寻址方式为寄存器寻址。

6. 指令步骤划分

与 ADD 指令类似，ADC 指令也划分为以下 3 个步骤：

(i) $AR \leftarrow PC$

(ii) 读主存， $IR \leftarrow$ 读出内容， $PC \leftarrow PC + 1$

(iii) $DR \leftarrow DR + SR + C$ ，保存状态信息 结束，检测中断请求，无中断请求，进入下一条指令的执行过程。

7. 给出每个步骤的节拍状态和控制信号

我们发现，上面的 (i)(ii) 两步与 A 组指令中其他指令都相同，可照搬其他指令的设计，采用相同的节拍状态。只有 (iii) 用到的控制信号需要设计，其节拍状态也可设定为 0011。根据教学计算机的要求，各部件要求的控制信号设计如下：

CI3~0	SCC3~0	MRW	I8~6	I5~3	B 口	A 口	SST	SSHSCI	DC2	DC1
0011	0000	100	001	011	000	0000	0000	001	0010	000

另外，(iii) 结束后，将和其他指令一样，转到检测中断请求的操作。

8. 根据新的指令系统表，改写控制信号逻辑表达式 根据添加了新指令的指令系统表，并参考原有的控制信号逻辑表达式，进行必要的修改，得到含新扩展指令的表达式。然后，可在模拟程序上进行调试。

具体的操作步骤是,启动模拟程序后,将我们新设计的各文件依次导入到模拟程序中,再设计一段汇编语言程序,包含有ADC 指令。装入到模拟程序中运行调试,通过观察寄存器状态,检查设计正确与否。

9. 硬件调试

如具备条件,可将调试正确的设计结果,装入到TEC-2000 教学计算机的Gal 和MACH 器件中,在实际的教学计算机硬件上检查设计的正确性,增强对计算机硬件的认识。

四、实验报告要求 实验报告的内容应包括以下几个方面的内容:

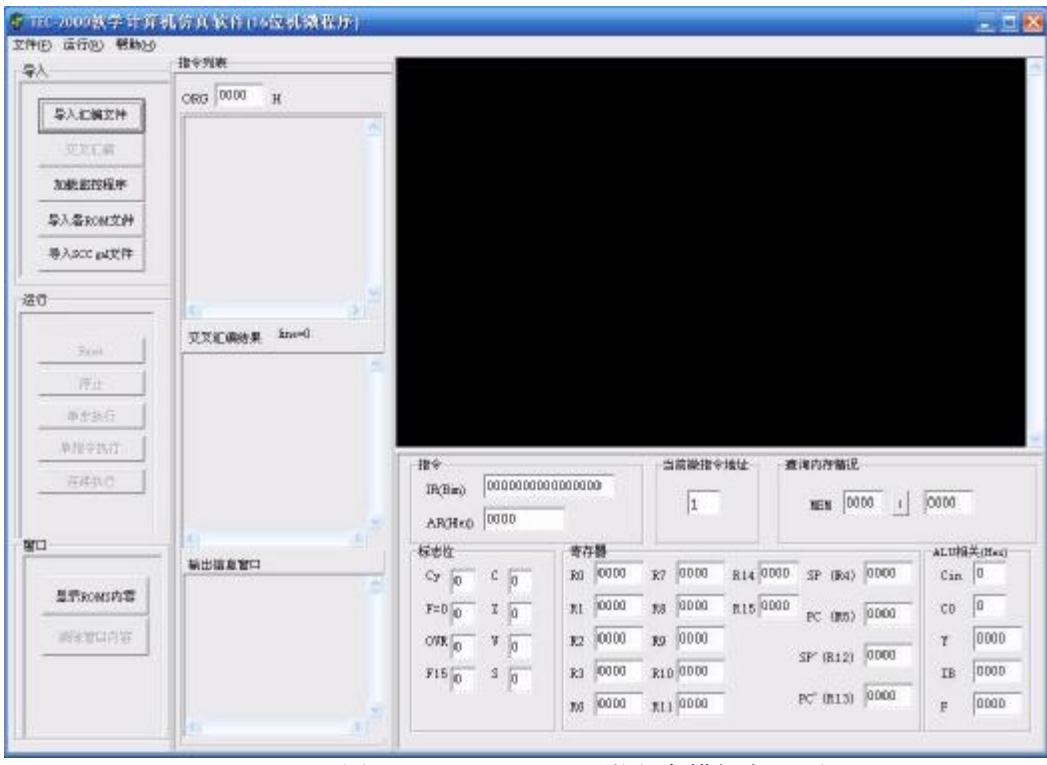
- 3. 实验准备 学习教学计算机组合逻辑控制器模拟程序使用的方法,简要说明你对教学计算机组合逻辑模拟程序的理解,在完成组合逻辑控制器的指令扩展过程中它起什么作用?
- 4. 指令系统 按照教师指定的扩展指令,仔细分析指令的功能、应该采用的寻址方式,设计出指令格式,并合理划分指令执行步骤,将指令进行归类,并为扩展指令指定指令编码。说明你的设计特点以及选用这种设计方案的理由
- 5. 按指令的执行步骤设计出每一步骤的控制信号,完成组合逻辑控制器的指令扩展设计。给出你的设计结果。简要说明节拍状态在组合逻辑控制器中的作用
- 6. 在模拟程序上实现你的设计,并调试。说明你在调试中遇到的问题,以及解决的办法。并给出解释
- 7. 实验总结和体会。

有关监控程序和BASIC 语言解释程序的细节,请参考第4 章有关内容。

B. 3 教学计算机微体系结构级微程序控制器模拟系统

B. 3. 1 启动与运行微程序控制器模拟程序

在安装好TEC-2000 教学计算机软件系统的PC 机上,选择TEC-2000 16 位教学计算机微程序仿真软件并启动运行,将出现图 B. 14 所示界面。



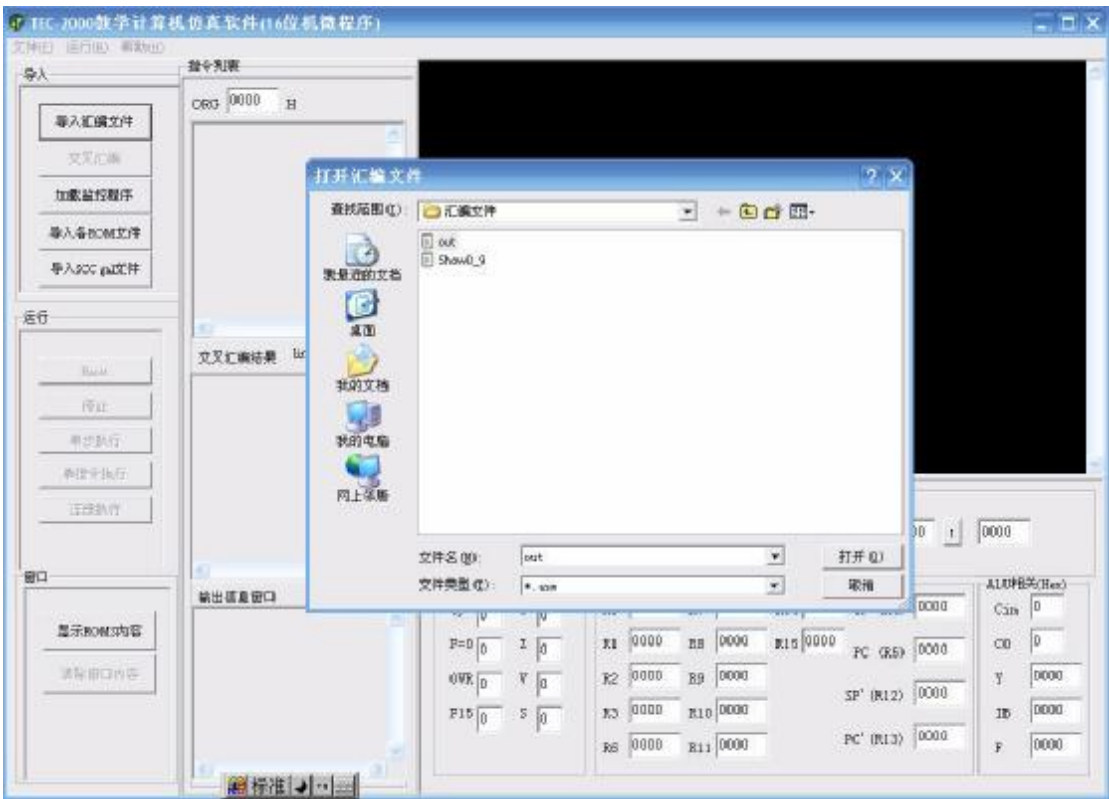
图B. 14 TEC-2000 微程序模拟主界面

界面的最左边是功能区,设置了程序的主要功能按钮,下面将介绍这些按钮的功能和使用方法。

中间部分是用户程序区，用于显示用户的程序，供调试过程中查看。右边的上半部分是输出窗口，可看成是TEC-2000 教学计算机软件模拟系统的显示器，按下“显示ROMS 内容”按钮后，将被切换为显示各 ROM 的二进制数值。下半部分是模拟程序的输出，主要包括当前运行程序的指令字、程序状态字、各通用寄存器的值，以及ALU 的输出结果，这些，可供用户调试程序时观察程序运行的结果。

使用模拟程序主要是对用户设计的微程序进行仿真运行，以检验方案的正确性。因此，首先要装入用户设计的微程序，和真正要在硬件的控制存储器中使用用的格式相同，即全部微程序方案由MAPROM、ROM1~ROM7 共8 个二进制文件组成。这些文件可用UltraEdit 软件编写，也可自行编制程序完成由文本格式到二进制数据格式的转换后生成。

准备好MAPROM、ROM1~ROM7 共8 个二进制文件后，将它们存放在一个文件目录下。然后，按图3-8 中界面上的“导入各ROM 文件”按钮，在图B. 15 所示界面上选定这8 个文件（文件名必须按照上述名称给定，并以.bin 为文件后缀）按“打开”按钮，将装入用户设计的微程序方案。以后就可以进入对各机器语言程序模拟运行的过程。



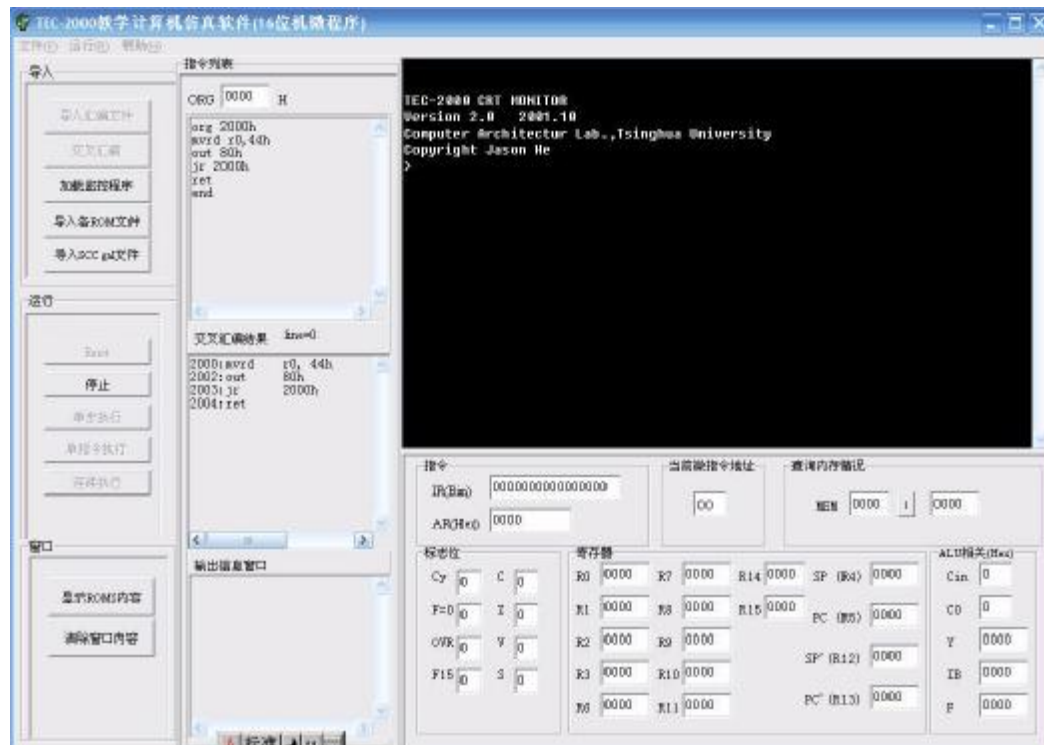
图B. 15 装入微程序方案

微程序方案装入完成后，可点击“显示各ROMS 内容”按钮，进行查看和检查，如图 B. 16 所示。若发现错误，可直接在界面上修改。修改后注意按“Update”按钮保存。



图B. 16 查看或修改微程序界面

按“显示输出窗口”按钮可返回主界面。准备好需要导入的文件后，将他们存放文件目录下，然后，首先按照图 B. 14 中的按钮分别导入其的文件，若出现如下图B. 17 所示界面则表示。



图B. 17 16 位微程序控制器启动界面

为检查、验证微程序方案的正确性，需要对微程序实现的指令系统中全部指令进行模拟运行，检查运行结果是否正确。这一过程可用简单的机器语言程序来完成，如我们可以用如下程序来验证 MVRD、ADD 及 RET 三条指令的正确性。

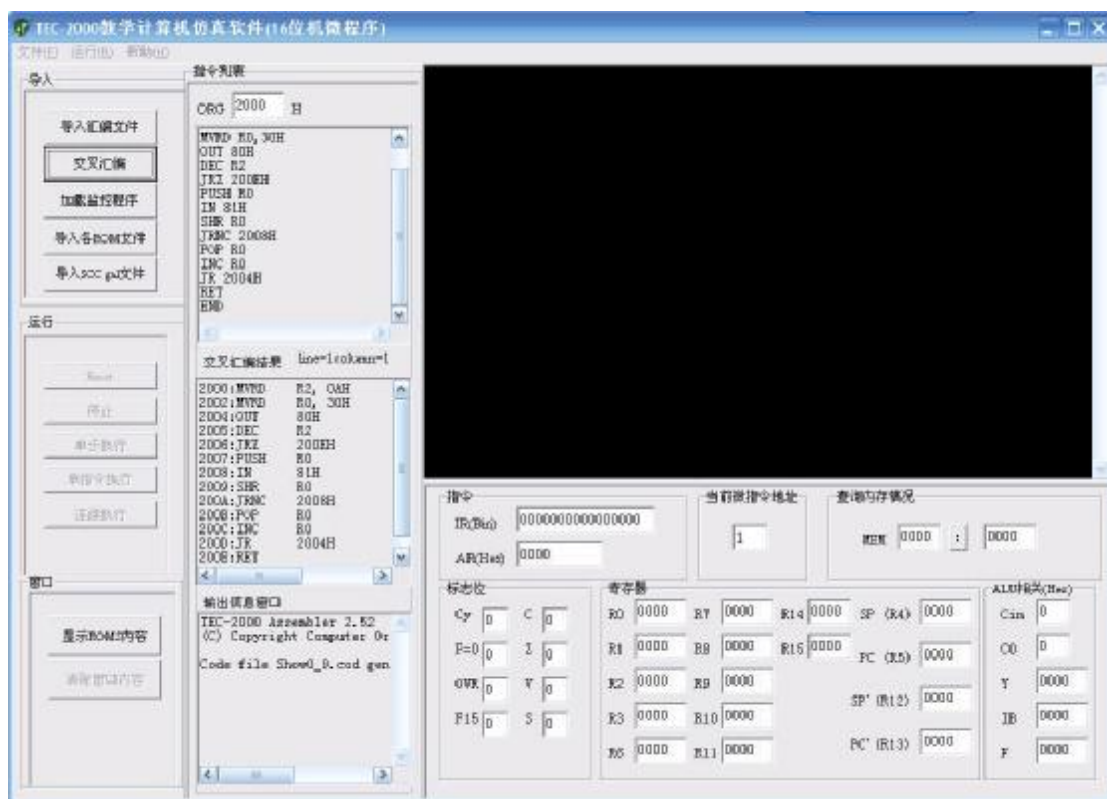
```

MVRD R0, 0
MVRD R1, 1
ADD R0, R1
RET

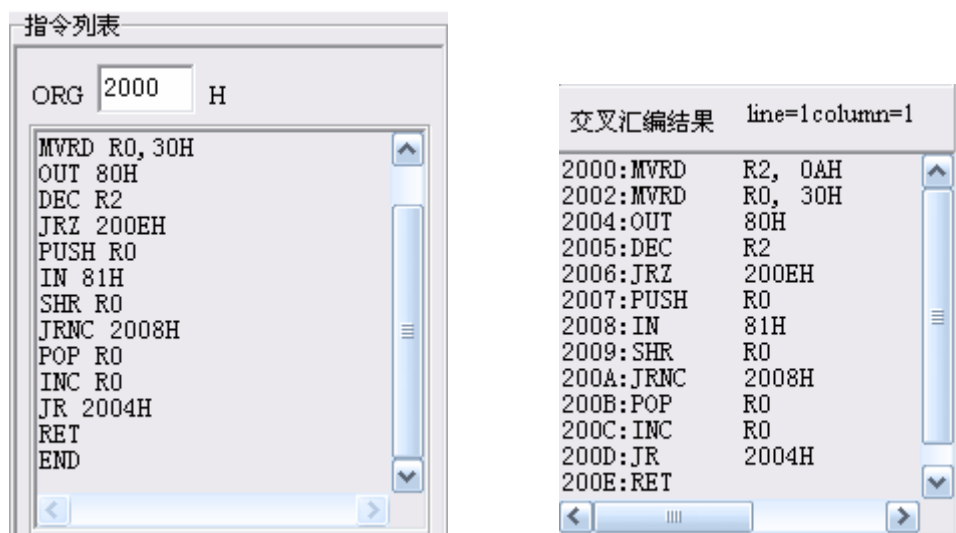
```

将指令系统中所有指令都验证完后，可得到微程序方案基本设计正确的结论。此时，可尝试进行加载监控程序并运行，这样，整台教学计算机将在用户设计的微程序控制器的控制下运行。

为方便用户输入程序，教学计算机模拟系统支持汇编语言（如用户使用 TEC-2000 规定的基本指令集）文本形式的二进制机器语言程序（对一些用户自行扩展的指令，交叉汇编程序不能汇编，此时可采用这种方式）和纯二进制机器指令程序（即真正的机器语言程序）下面分别介绍它们的使用方法。汇编语言程序是符号化的机器语言，与机器语言相比，更方便进行人机交流。通过“导入汇编文件”按钮或“文件”菜单项中的“导入汇编文件”项，将汇编语言程序读入到系统中，然后，再使用“交叉汇编”按钮，对该汇编语言程序进行交叉汇编。此时，系统界面如图 B. 18 所示，可以连续、单指令或单步骤执行方式下运行这个汇编语言程序。



图B. 18 交叉汇编后的文件框



图B. 19 装入汇编语言程序并交叉汇编图示

汇编语言程序虽然比较直观,但受交叉汇编的限制,只能是使用 TEC-2000 的基本指令且是规定的操作码。对用户自行扩展的指令,或者是改变了指令的操作码,则无法直接使用汇编语言程序,只能使用机器语言程序。

机器语言程序一般应是二进制格式,直接装入到主存储器中运行,这样,对用户编写程序就不是很方便,写出 0、1 的代码后,还要事先转换成二进制格式才能装入。为此,教学计算机模拟系统支持用户用 0、1 代码编写的机器语言程序(文本形式)由模拟系统本身进行到二进制的转换。例如,我们可以用下面的机器语言文本文件来替换前面的汇编语言程序。

```
1000100000000000
0000000000000000
1000100000010000
0000000000000001
0000000000000001
1000111100000000
```

这些机器指令由前面的汇编语言程序对应而来,使用的是 TEC-2000 缺省定义的操作码。将这些内容保存为文本文件后(以.txt 为后缀,选择“文件”菜单项中的“导入机器指令的汇编文件”项,可将程序装入到系统中。系统将自动完成该程序从文本文件到二进制机器指令的转换。

当然,本系统也可直接装入二进制机器指令运行,例如把一些已经是机器指令的程序直接装入到系统中运行。程序在模拟系统中运行正确后,也可以用做在实际的教学计算机上,用来进行硬件调试(此类文件一般应以.COD 或.BIN 作为后缀)选择“文件”菜单项中的“导入机器指令的汇编文件”项,可将程序装入到系统中。需要说明的是,监控程序也可以采用这种方式装入。

所有的指令都检查无误后,可以把监控程序作为对微程序设计方案的一个总的测试对象。选择“文件”菜单项中的“加载监控程序”项,可将监控程序装入到系统中。

程序装入完成后,可使用“Reset”按钮,启动程序的运行。可用“连续执行”按钮将程序完整的执行,也可使用“单指令执行”按钮单独执行一条指令,还可以用“单步执行”按钮单独执行一个微指令。“单指令执行”和“单步执行”时,可通过观察各输出结果,判断程序执行的正确性。若不正确,可能是微程序设计方案有错误,可通过前面介绍的查看和修改微程序界面,修改微程序,再进行进一步的调试,直到程序能正常运行。最后,可通过运行监控程序来进行较大规模的检验。

B. 3. 2 使用模拟程序完成微程序控制器实验

在硬件教学计算机 TEC-2000 上,微程序控制器部件实验要求是在已有基本指令的基础上,完成扩展几条新的指令,使新指令能够在教学机上运行。它要求掌握指令格式、指令操作码编码、寻址方式和指令功能等基本内容,熟悉教学计算机总体组成和各部件的运行原理,设计出新扩展指令

的微程序段，并合理安排到已有基本指令的微程序中，并且希望同学们增强硬件调试能力，能熟练地将微程序装入到控制存储器中，发现和排除设计中的错误。实验要求比较高。

为降低实验难度，可以首先先在模拟程序上完成微程序控制器实验的设计部分，将设计的扩展方案在模拟环境下调试通过后，再进行硬件实验，这可排除设计中的各种错误，转而集中精力解决硬件故障问题。在硬件实验条件比较欠缺的情况下，也可以通过模拟程序实验，基本掌握微程序控制器的基本原理，支持这种方案，但不推荐此种实验方式，已经有很多单位，也包括清华大学计算机系，都曾吃过硬件课程软化的亏。下面，我们就介绍使用模拟程序完成微程序控制器实验的基本要求 and 步骤。

一、实验设备和环境

本实验的主要内容是在 PC 机上，使用教学计算机微程序的模拟程序，模拟微程序控制器的运行。因此，实验设备为 PC 计算机，主要环境为 Windows 操作系统，安装教学计算机微程序模拟软件。

二、实验目的

6. 进一步熟悉教学计算机指令格式、指令编码、寻址方式和指令功能；
7. 进一步熟悉教学计算机的总体组成和各部件的运行原理，理解指令的执行过程；
8. 通过对指令系统的扩展，了解微程序控制器设计和实现的基本过程；
9. 思考和讨论微程序控制器的特点并与组合逻辑控制器进行比较。

三、实验要求

10. 通过预习，熟悉教学计算机微程序控制器模拟系统的使用；
11. 确定实验中要扩展的指令的功能、编码、格式和寻址方式；
12. 设计新指令的微程序段，并将其加入到原有的微程序中，组成新的微程序；
13. 将新的微程序装入到模拟系统中，并验证其正确性，尤其是扩展指令的正确性；
14. 进行实验总结。

四、实验步骤

下面，以在现有 TEC-2000 教学计算机微程序控制器基础上，扩展指令 ADC DR，SR 为例，说明实验的步骤。

15. 指令功能和格式的确定

ADC DR, SR 指令的功能是，将 SR 中的值用作源操作数、DR 的值用作目的操作数，并把状态寄存器中 C 标志位的值作为最低位进位输入完成求和运算，结果存入目的寄存器 DR 中，即

$DR \leftarrow DR + SR + C$ 。

指令格式可确定为操作码（8 位）DR（4 位）SR（4 位）由于它的功能在取指之后可一步完成，我们把它归为 A 组指令，分配操作码为 00100000。

指令寻址方式显然为寄存器寻址。

16. 指令步骤划分

与 ADD 指令类似，ADC 指令也划分为以下 3 个步骤：

(i) $AR \leftarrow PC$

(ii) 读主存， $IR \leftarrow$ 读出内容

$PC \leftarrow PC + 1$

(iii) $DR \leftarrow DR + SR + C$ ，保存状态信息

结束，检测中断请求，无中断请求，进入下一条指令的执行过程。

17. 给出每个步骤的控制信号 我们发现，上面的 (i)(ii) 两步都可借用原有微程序的段，只有 (iii) 的控制信号需要设计。

根据教学计算机的要求，各部件要求的控制信号设计如下：

CI3~0	SCC3~0	OMRW	SAI8~6	SBI5~3	B 口	A 口	OSST	DC2	DC1
0011	0000	0100	0001	1011	1000	0000	0001	0010	0000

另外，(iii) 结束后，将和其他指令一样，转到检测中断请求的操作，所以，该条微指令的下地址字段值应为 00110000（也就是 16 进制的 30）

18. 将设计好的微程序段加入到原有微程序中

首先，应确定该段微程序的微地址，我们可以选择一个原有微程序中没有使用的微地址如 50，来存放该段微程序。

具体的操作步骤是,启动模拟程序后,先导入只有基本指令的ROM 文件,进入显示ROMS 内容窗口,在该窗口输入微地址50,按“查找”按钮后,在对应栏目中输入上面设计的微程序控制信号,输入完成后,按“update”按钮修改。注意:因为我们给ADC 设计的操作码是20,还需要修改地址为20 的MPROM 单元的内容,向该单元写入50。如希望保存设计结果,可按“生成新文件”按钮保存。此处的数字均为16 进制的数。

19. 检查并调试设计方案

设计一段汇编语言程序,包含有ADC 指令。装入到模拟程序中运行调试,通过观察寄存器状态,检查设计正确与否。

20. 硬件调试

如具备条件,可将调试正确的微程序,装入到TEC-2000 教学计算机的控制存储器中,在实际的教学计算机硬件上检查设计的正确性,增强对计算机硬件的认识。

五、实验报告要求

实验报告的内容应包括以下几个方面的内容:

21. 实验准备 学习教学计算机微程序模拟程序使用的方法,简要说明你对教学计算机微程序模拟程序的理解,

它在完成微程序控制器方案的指令扩展中的作用有哪些?

22. 指令系统 按照教师指定的扩展指令,仔细分析指令的功能、应该采用的寻址方式,设计出指令格式,并

合理划分指令执行步骤,将指令进行归类,并为扩展指令指定指令操作码编码。说明你的设计的特点以及你选择这种设计方案的理由。

23. 按指令的执行步骤设计出每一步骤的控制信号,完成微程序段的设计。给出你的设计结果。 简要说明下地址字段在微程序中的作用,根据什么来确定下地址字段的内容?

24. 在模拟程序上实现你的设计,并调试。说明你在调试中遇到的问题,以及解决的办法,并给出解释。

25. 实验总结和体会。

附录 3 联机通讯指南

一、联机操作步骤

1、准备

- (1) 准备一台串口工作良好的PC机。
- (2) 将TEC-XP+放在实验台上，打开实验箱的盖子，确定电源处于断开状态。

2、连接电源线

将黑色的电源线一端接220V交流电源，另一端插在TEC-XP+实验箱的电源插座里。

3、连接TEC-XP+和PC

取出通讯线，将通讯线的一端的9芯插头接在TEC-XP+实验箱上的串口“COM1”上，另一端接到PC 机的串口上。

4、TEC-XP+的初始设置

将TEC-XP+实验系统左下方的六个黑色的控制机器运行状态的开关置于正确的位置，在找个实验中开关应置为001100（连续、内存读指令、组合逻辑、联机、16位、MACH），控制开关的功能在开关上、下方有标识；开关拨向上方表示“1”，拨向下方表示“0”，“X”表示任意。

5、开机

打开电源，船形开关和5V电源指示灯亮。

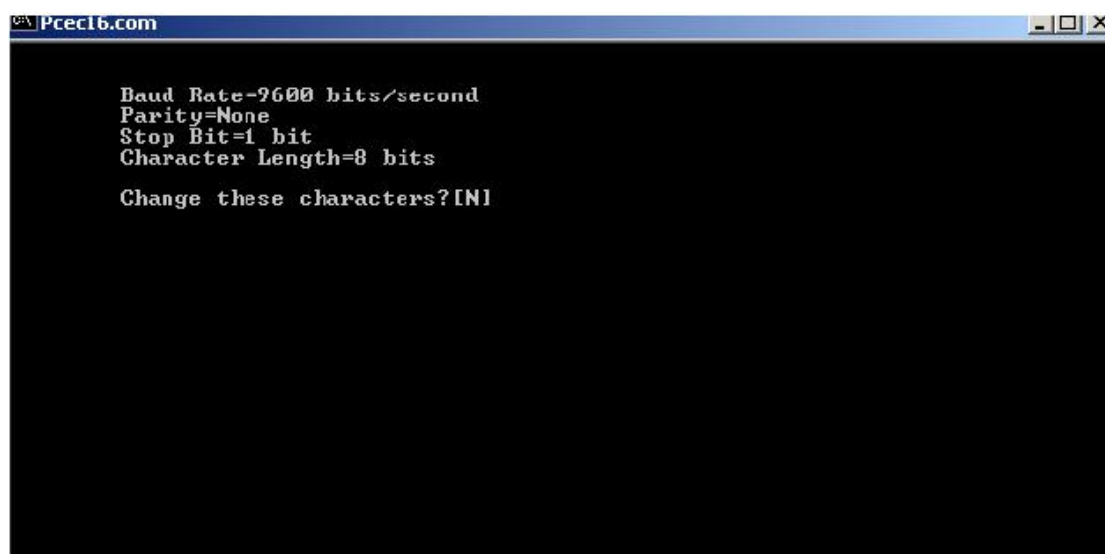
6、加载通讯软件

在PC机上运行数据光盘中配套的PCEC16.EXE文件，根据连接的PC机的串口设置所用PC机的串口为“1”或“2”，其它的设置一般不用改动,直接回车即可。具体步骤如下（仿真终端软件PCEC 的操作步骤）：

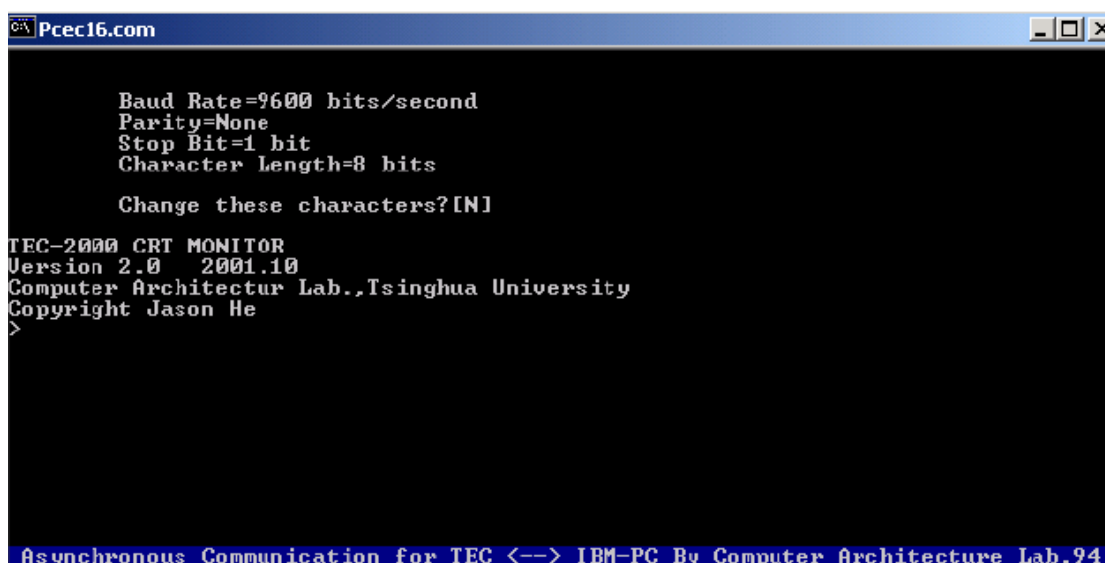
- (1) 在PC机上建一个文件夹TEC-XP+;
- (2) 取出配套的用户盘，将应用程序PCEC16拷贝到用户机器硬盘上该文件夹里；
- (3) 双击PCEC16图标，出现如图所示的界面：



(4) 系统默认选择串口1，用户可根据实际情况选择串口1 或是串口2（这里的串口指的是和TEC-XP+教学实验系统相连的PC机的串口），按回车后出现如图界面：



(5) 图中是系统设定的一些传输参数，建议用户不要改动，直接回车。按一下“RESET”按钮放开后再按一下“START”按钮，出现界面如图所示：



(6) 此时表明TEC-XP+机器联机通讯正常。

二、软件操作注意事项：

- 1、用户在选择串口时，选定的是PC机的串口1或串口2，而不是TEC-XP+实验系统上的串口。即选定的是用户实验时通讯线连接的PC机的端口；
- 2、如果在运行到第五步时没有出现应该出现的界面，用户需要检查是不是打开了两个软件界面，若是，关掉其中一个再试；
- 3、有时若TEC-XP+实验系统不通讯，也可以重新启动软件或是重新启动PC再试；

4、在打开该应用软件时，其它的同样会用到该串口的应用软件要先关掉。

三、联机通讯失败自检：

如果上述的硬件和软件的操作都正确，联机却依旧失败，用户可以进行如下测试：

- 1、测试PC机的串口是否能正常工作，或是换一台PC或换同一台PC的另一个串口再试，在换串口时要将TEC-XP+实验系统断电，换完后重新启动实验系统和软件；
- 2、检查机器上的元器件插接是否正确（建议用户对照能够正常通讯的实验系统进行详细检查），有没有被学生动过，尤其是扩展内存和扩展I/O接口时，芯片方向是否插对，片选信号有没有连接；
- 3、检查相应的短路子是否连接正确。

附录4 TEC-XP+计算机组成原理实验系统简明操作卡

一、微指令格式：

下地址	CI3~CI0	SCC	MRW	I2~I0	I8~I6	I5~I3	B口	A口	SST	SSH SCI	DC2	DC1
8	4	4	3	3	3	3	4	4	3	3	3	3

二、控制信号的控制功能：

/MIO、REQ、/WE

CI3~0	功能	MRW	功能
0000 (0#)	初始化	000	内存写
0010 (2#)	MAPROM	001	内存读
0011 (3#)	条件微转移	010	I/O 写
1110 (14#)	顺序执行	011	I/O 读
		1XX	无读写

编码	I8~6			I5~3	I2~0	
	REG	Q	Y	功能	R	S
000		F→Q	F	R+S	A	Q
001			F	S-R	A	B
010	F→B		A	R-S	0	Q
011	F→B		F	R∨S	0	B
100	F/2→B	Q/2→Q	F	R∧S	0	A
101	F/2→B		F	/R∧S	D	A
110	2F→B	2Q→Q	F	R⊕S	D	Q
111	2F→B		F	/(R⊕S)	D	0

SST	C	Z	V	S
000	C	Z	V	S
001	CY	F=0	OVR	F15
010	内部总线			
011	0	Z	V	S
100	1	Z	V	S
101	RAM0	Z	V	S
110	RAM1 5	Z	V	S
111	Q0	Z	V	S

DC2	译码信号	操作
000	NC	NC
001	/GIR	指令寄存器接收
010	/GARL	AR 低位接收
011	/GARH	AR 高位接收
100	/INTR	恢复中断优先级
101	/INTN	新中断优先级
110	/EI	开中断，置 INTE 为 1
111	/DI	关中断，清 INTE 为 0

DC1	译码信号	操作
000	/SWTOIB	开关到内部总线
001	/RTOIB	ALU 输出到内部总线
010	/ETOIB	16 扩展符号到内部总线
011	/FTOIB	状态到内部总线
100	/STOIB	8 扩展符号 到内部总线
101	/INTVH	中断向量高位到内部总线
110	/INTVL	中断向量低位到内部总线
111	NC	NC

SSH SCI	Cin / Shift
000	Cin = 0
001	Cin = 1
010	Cin = C
100	逻辑移位
101	循环移位

三、指定的专用寄存器

16 位机：PC：R5

SP：R4

IO 默认 R0

T3~0	MIO REQ WE	I2~0	I8~7	I6 I5~3	B 口	A 口
4 位	3 位	3 位	2 位 + 1 位	3 位	4 位	4 位

36 位微型开关的控制功能分配

SST	SSHSCI	DC2	DC1
3 位	3 位	3 位	3 位

附录 5 微程序入口地址映射表

表 1 基本指令微程序入口地址映射表

序号	指令	编码	入口地址
1	ADD DR, SR	0000 0000	04
2	SUB DR, SR	0000 0001	05
3	AND DR, SR	0000 0010	06
4	OR DR, SR	0000 0110	07
5	XOR DR, SR	0000 0100	08
6	CMP DR, SR	0000 0011	09
7	TEST DR, SR	0000 0101	0A
8	MVRR DR, SR	0000 0111	0B
9	INC DR	0000 1001	0C
10	DEC DR	0000 1000	0D
11	SHL DR	0000 1010	0E
12	SHR DR	0000 1011	0F
13	JRC OFFSET	0100 0100	10
14	JRNC OFFSET	0100 0101	10
15	JRZ OFFSET	0100 0110	10
16	JRNZ OFFSET	0100 0111	10
17	JR OFFSET	0100 0001	11
18	IN PORT	1000 0010	12
19	OUT PORT	1000 0110	12
20	PSHF	1000 0100	15
21	PUSH DR	1000 0101	15
22	POP DR	1000 0111	17
23	POPF	1000 1100	17
24	STRR [DR], SR	1000 0011	19
25	LORR DR, [SR]	1000 0001	1B
26	MVRD DR, DATA	1000 1000	1D
27	JMPA ADR	1000 0000	1E
28	CALA ADR	1100 1110	1F
29	RET	1000 1111	23

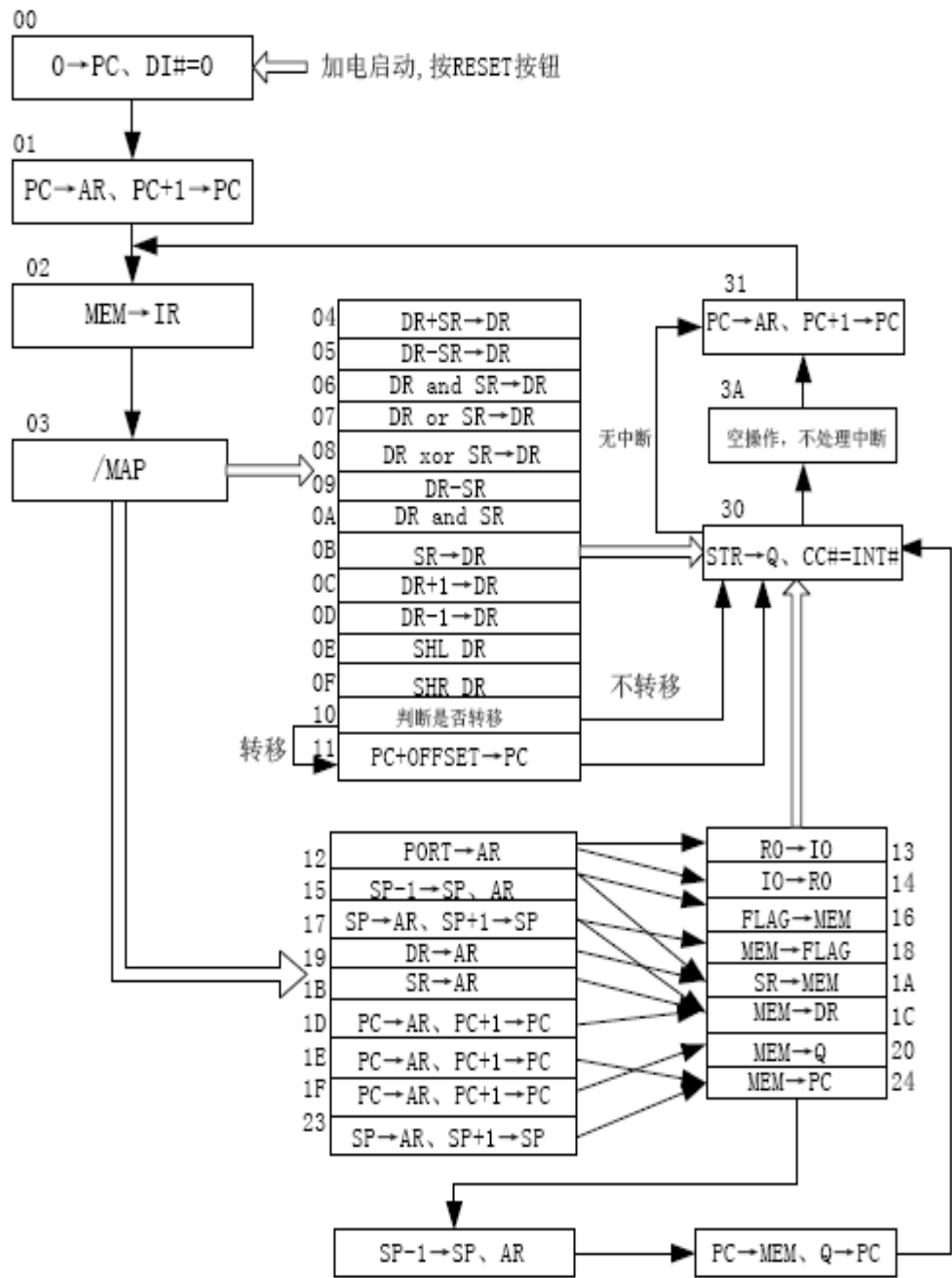
表 2 扩展指令微程序入口地址映射表

序号	指令	编码	入口地址
1	ADC DR, SR	0010 0000	50
2	SBB DR, SR	0010 0001	51
3	NOT DR	0010 1101	52
4	ASR DR	0010 1100	53
5	RCL DR	0010 1010	54
6	RCR DR	0010 1011	55
7	CLC	0110 1100	56
8	STC	0110 1101	57
9	EI	0110 1110	58
10	DI	0110 1111	59
11	JMPR SR	0110 0000	5A
12	LDRA DR, [ADR]	1110 0100	5B
13	LDRX DR, OFFSET	1110 0101	5D
14	STRA [ADR], SR	1110 0111	5F
15	STRX DR, OFFSET[SR]	1110 0110	61
16	CALR SR	1110 0000	64
17	IRET	1110 1111	67
18	JRS OFFSET	0110 0100	69
19	JRNS OFFSET	0110 0101	69

注：该指令入口地址映射表中，前29条指令为基本指令，所有基本指令都已编程到微程序控制器中；后19条为扩展指令，需用户自己确定完成各步操作所需的控制微码，并将微码扩展到MAPROM和七片MPRAM中去。

附录 6 指令流程框图

一、基本指令执行流程框图：



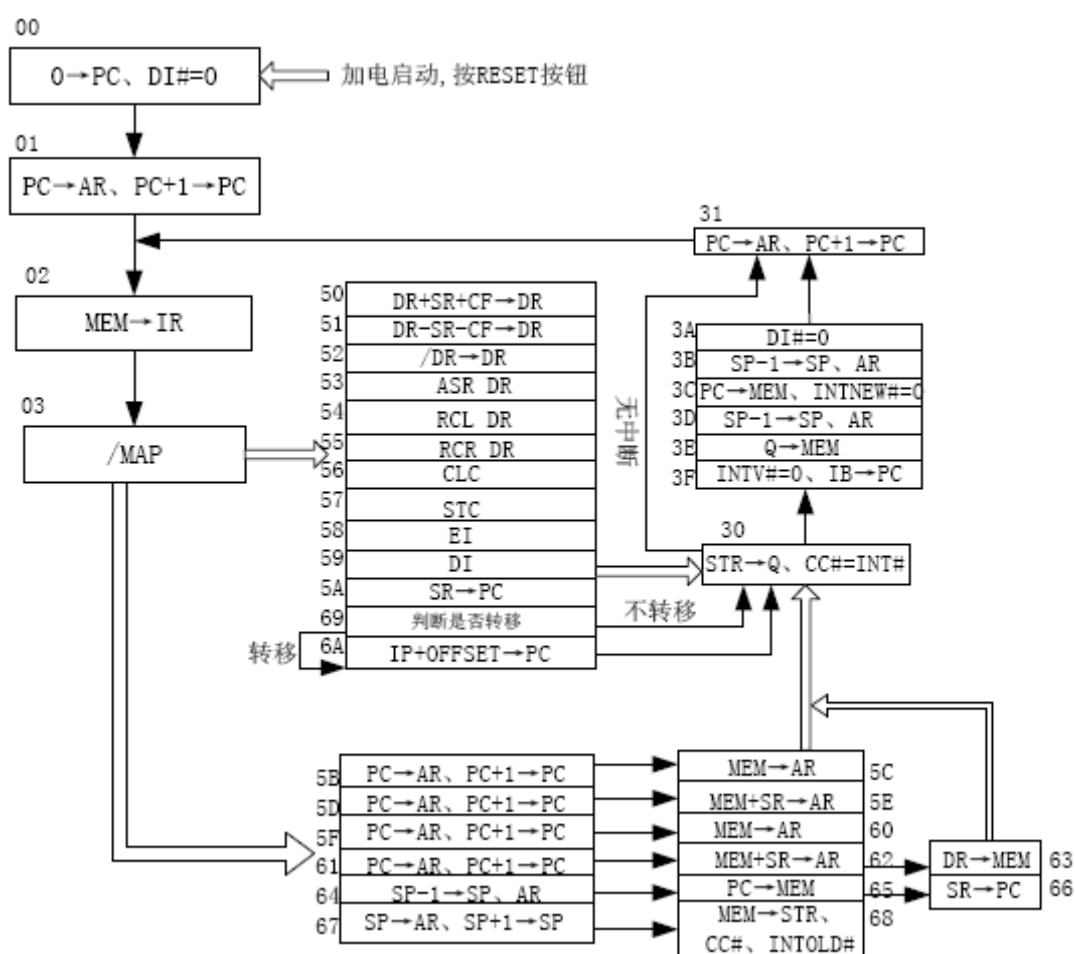
注：12 是IN/OUT两条指令的入口地址，IN指令由12 跳到14，OUT指令由12跳到13。

15 是PUSH/PSHF两条指令的入口地址，PUSH指令由15跳到1A，PSHF 指令由15跳到16。

17 是POP/POPF两条指令的入口地址，POP指令由17跳到1C，POPF 指令由17跳到18。

在地址3A 处放的是一条空操作指令，只起跳转的作用，用户在扩展中断隐指令时，可将该地址的这条指令用中断隐指令代替。

二、扩展指令执行流程框图：



附录 7 指令流程表

指令	操作功能	微址	下址	CI3~0	SCC30	OMRW	0I2~0	SA、 I8~6	SB、 I5~3	B 口	A 口	OSST	SSH、 SCI	DC2	DC1
ALL	0→PC、DI#=0	00	00	1110	0000	0100	0001	0011	0001	0101	0101	0000	0001	0111	0000
ALL	PC→AR、PC+1→PC	01	00	1110	0000	0100	0011	0010	0000	0101	0101	0000	0001	1011	0000
ALL	MEM→IR	02	00	1110	0000	0001	0000	0001	0000	0000	0000	0000	0000	0001	0000
ALL	/MAP	03	00	0010	0000	0100	0000	0001	0000	0000	0000	0000	0000	0000	0000
ADD	DR+SR→DR	04	30	0011	0000	0100	0001	1011	1000	0000	0000	0001	0000	0000	0000
SUB	DR-SR→DR	05	30	0011	0000	0100	0001	1011	1001	0000	0000	0001	0001	0000	0000
AND	DR∧SR→DR	06	30	0011	0000	0100	0001	1011	1100	0000	0000	0001	0000	0000	0000
OR	DR∨SR→DR	07	30	0011	0000	0100	0001	1011	1011	0000	0000	0001	0000	0000	0000
XOR	DR⊕SR→DR	08	30	0011	0000	0100	0001	1011	1110	0000	0000	0001	0000	0000	0000
CMP	DR-SR	09	30	0011	0000	0100	0001	1001	1001	0000	0000	0001	0001	0000	0000
TEST	DR∧SR	0A	30	0011	0000	0100	0001	1001	1100	0000	0000	0001	0000	0000	0000
MVRR	SR→DR	0B	30	0011	0000	0100	0100	1011	1000	0000	0000	0000	0000	0000	0000
INC	DR+1→DR	0C	30	0011	0000	0100	0011	0011	1000	0000	0000	0001	0001	0000	0000
DEC	DR-1→DR	0D	30	0011	0000	0100	0011	0011	1001	0000	0000	0001	0000	0000	0000
SHL	SHLDR	0E	30	0011	0000	0100	0011	0111	1000	0000	0000	0110	0000	0000	0000
SHR	SHRDR	0F	30	0011	0000	0100	0011	0101	1000	0000	0000	0101	0000	0000	0000
JR CND	JRCndOFFSET	10	30	0011	0100	0100	0000	0001	0000	0000	0000	0000	0000	0000	0000
JR	Offset+IP→PC	11	30	0011	0000	0100	0101	0011	0000	0101	0101	0000	0000	0000	0010
IN/OUT	PORT→AR	12	14	0011	0110	0100	0111	0001	0000	0000	0000	0000	0000	0011	0010
	R0→IO	13	30	0011	0000	0010	0011	0001	0000	0000	0000	0000	0000	0000	0001
	IO→R0	14	30	0011	0000	0011	0111	0011	0000	0000	0000	0000	0000	0000	0000
PSH/F	SP-1→SP, AR	15	1A	0011	0111	0100	0011	0011	0001	0100	0000	0000	0000	0011	0000
	FLAG→MEM	16	30	0011	0000	0000	0000	0001	0000	0000	0000	0000	0000	0000	0011
POP/F	SP→AR, SP+1→SP	17	1C	0011	0111	0100	0011	0010	0000	0100	0100	0000	0001	0011	0000
	MEM→FLAG	18	30	0011	0000	0001	0000	0001	0000	0000	0000	0010	0000	0000	0000

STRR	DR→AR	19	00	1110	0000	0100	0011	0001	1000	0000	0000	0000	0000	0011	0000
ALL	SR→MEM、CC#=0	1A	30	0011	0000	0000	0100	1001	0000	0000	0000	0000	0000	0000	0001
LORR	SR→AR	1B	00	1110	0000	0100	0100	1001	0000	0000	0000	0000	0000	0011	0000
ALL	MEM→DR、CC#=0	1C	30	0011	0000	0001	0111	0011	1000	0000	0000	0000	0000	0000	0000
MVRD	PC→AR、PC+1→PC、CC#=0	1D	1C	0011	0000	0100	0011	0010	0000	0101	0101	0000	0001	0011	0000
JMPA	PC→AR、PC+1→PC	1E	24	0011	0000	0100	0011	0010	0000	0101	0101	0000	0001	0011	0000
CALA	PC→AR、PC+1→PC	1F	00	1110	0000	0100	0011	0010	0000	0101	0101	0000	0001	0011	0000
	MEM→Q	20	00	1110	0000	0001	0111	0000	0000	0000	0000	0000	0000	0000	0000
	SP-1→SP、→AR	21	00	1110	0000	0100	0011	0011	0001	0100	0000	0000	0000	0011	0000
	PC→MEM、Q→PC、CC#=0	22	30	0011	0000	0000	0010	0010	0000	0101	0101	0000	0000	0000	0001
RET	SP→AR、SP+1→SP	23	00	1110	0000	0100	0011	0010	0000	0100	0100	0000	0001	0011	0000
	MEM→PC、CC#=0	24	30	0011	0000	0001	0111	0011	0000	0101	0000	0000	0000	0000	0000
ALL	STR→Q、CC#=INT#	30	3A	0011	0010	0100	0111	0000	0000	0000	0000	0000	0000	0000	0011
	PC→AR、PC+1→PC、CC#=0	31	02	0011	0000	0100	0011	0010	0000	0101	0101	0000	0001	1011	0000
	用户做中断实验时，自写中断隐指令代替本行微指令。	3A	31	0011	0000	0100	0000	0001	0000	0000	0000	0000	0000	0000	0000
	DI#=0	3A	00	1110	0000	0100	0000	0001	0000	0000	0000	0000	0000	0111	0000
	SP-1→SP、→AR	3B	00	1110	0000	0100	0011	0011	0001	0100	0000	0000	0000	0011	0000
	PC→MEM、INTNEW#=0	3C	00	1110	0000	0000	0100	0001	0000	0000	0101	0000	0000	0101	0001
	SP-1→SP、→AR	3D	00	1110	0000	0100	0011	0011	0001	0100	0000	0000	0000	0011	0000
	Q→MEM	3E	00	1110	0000	0000	0010	0001	0000	0000	0000	0000	0000	0000	0001
	INTV#=0、IB→PC、CC#=0	3F	31	0011	0000	0100	0111	0011	0000	0101	0000	0000	0000	0000	0101
ADC	DR+SR+CF→DR														
SBB	DR-SR-CF→DR														
NOT	/DR→DR														
ASR	ASR DR														
RCL	RCL DR														
RCR	RCR DR														

CLC	CLC														
STC	STC														
EI	EI														
DI	DI														
JMPR	SR→PC、CC#=0	5A	30	0011	0000	0100	0100	1011	0000	0101	0000	0000	0000	0000	0000
LDRA	PC→AR、PC+1→PC	5B	00	1110	0000	0100	0011	0010	0000	0101	0101	0000	0001	0011	0000
	MEM→AR、CC#=0	5C	1C	0011	0000	0001	0111	0001	0000	0000	0000	0000	0000	0011	0000
LDOR	PC→AR、PC+1→PC	5D	00	1110	0000	0100	0011	0010	0000	0101	0101	0000	0001	0011	0000
	MEM+SR→AR、CC#=0	5E	1C	0011	0000	0001	0101	1001	0000	0000	0000	0000	0000	0011	0000
STAR	PC→AR、PC+1→PC	5F	00	1110	0000	0100	0011	0010	0000	0101	0101	0000	0001	0011	0000
	MEM→AR、CC#=0	60	1A	0011	0000	0001	0111	0001	0000	0000	0000	0000	0000	0011	0000
STOR	PC→AR、PC+1→PC	61	00	1110	0000	0100	0011	0010	0000	0101	0101	0000	0001	0011	0000
	MEM+SR→AR、CC#=0	62	00	1110	0000	0001	0101	1001	0000	0000	0000	0000	0000	0011	0000
	DR→MEM	63	30	0011	0000	0000	0011	1001	0000	0000	0000	0000	0000	0000	0001
CALR	SP-1→SP、→AR	64	00	1110	0000	0100	0011	0011	0001	0100	0000	0000	0000	0011	0000
	PC→MEM	65	00	1110	0000	0000	0100	0001	0000	0000	0101	0000	0000	0000	0001
	SR→PC、CC#=0	66	30	0011	0000	0100	0100	1011	0000	0101	0000	0000	0000	0000	0000
IRET	SP→AR、SP+1→SP	67	00	1110	0000	0100	0011	0010	0000	0100	0100	0000	0001	0011	0000
	MEM→STR、CC#、INTOLD#	68	23	0011	0000	0001	0000	0001	0000	0000	0000	0010	0000	0100	0000
JR CND	JRCnd OFFSET	69	30	0011	0101	0100	0000	0001	0000	0000	0000	0000	0000	0000	0000
	Offset+IP→PC	6A	30	0011	0000	0100	0101	0011	0000	0101	0101	0000	0000	0000	0010
Debug 11111111	执行Debug 前, 置 SW=FFFFH, 控制10001														
	/SWTOIB, D+0→B(R0), Y	70	00	1110	0000	0100	0111	0011	0000	0000	0000	0000	0000	0000	0000
	B(R0)+0→Y, AR, DB, LOOK:Y, AR, DB, IR=FF	71	00	1110	0000	0000	0011	0001	0000	0000	0000	0000	0000	0011	0001
	Cin=1, B(R0)+0+Cin→Y, IR	72	00	1110	0000	0100	0011	0001	0000	0000	0000	0000	0001	0001	0001
	B(R0)+0+Cin→Y, AR, DB,	73	00	1110	0000	0000	0011	0001	0000	0000	0000	0000	0001	0011	0001

	LOOK:Y, AR, DB, IR=00														
	CONTRL=FF	74	00	1110	1111	0111	0111	1111	1111	1111	1111	1111	1111	1111	1111
	CONTRL=00	75	FF	0011	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
		FF	00	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

附录 8 书写实验报告的一般格式



_____学院_____专业_____班_____组、学号_____

姓名_____协作者_____教师评定_____

实验题目_____

一、实验目的：

1、

2、

3、

.....

二、实验设备与器材：

填写所用器材与设备或电路板等。

三、实验说明和原理：

简介实验说明和原理

四、实验内容：

根据具体实验内容及要求去写。

五、实验步骤：

根据本次实验具体要求去写。

六、思考题：

本次实验若有思考题，回答本次实验报告中的思考题。

七、实验心得：

对本次实验进行总结，写出在实验中遇到的问题及所采用的解决方法，对本次实验的改进意见等等。