# 版权声明



唐都公司网址：http://www.tangdu.com/

# 前 言

根据“计算机系统结构”的主要教学内容，西安唐都科教仪器公司开发生产了 TDX-CMX“计算机组成原理与系统结构教学实验系统”。TDX-CMX 是一套具有完全开放性、可扩展性的通用开发平台，该设备与以往的计算机系统结构实验产品相比，具有以下主要优点：

. 采用了更为先进和完全开放的多端口、多总线部件电路单元。

. 可组态构建从单总线到多总线多种先进结构的模型计算机内部数据通路。

.开放式的部件电路结合 VHDL 语言程序设计和实现，可以深入细致地开展关于 CISC 与 RISC 指令系统、先进存储器、具有指令预取功能的模型计算机、三级流水 RISC 模型计算机、超标量结构的模型计算机等的实验研究，全面支持计算机系统结构的实验教学。

. 为各种先进结构的计算机提供了示教效果更为出色的数据通路图实时动态图形调试界面，支持单拍、单周期、连续等调试功能；数据通路图的调试过程也可以保存及回放。

. 更为先进和完善的监测和保护电路设计，使实验系统更易于维护和使用。

该设备的推出为计算机系统结构的教学和计算机系统设计提供了功能强大的软、硬件支持，可使学生通过实验来更有效的理解并掌握当今先进的计算机的结构，为进一步开展具有实用价值的计算机系统的设计打下良好的基础。

本书是为 TDX-CMX “计算机组成原理与系统结构教学实验系统”配套的实验教程。该书以标量机为目标，详细的讲述了 CACHE、RISC、流水、超标量等目前主流的技术及设计实现方法。全书共分为五章，其中：第一章为运算器和寄存器堆的结构，阐述了从计算机系统结构的角度设计先进的运算器和寄存器堆结构的理论和方法；第二章介绍计算机的指令系统，通过基于 CISC、RISC 指令系统的模型机设计实验，详细的讲述了 CISC 和 RISC 的特点和设计方法；第三章介绍计算机的存储系统，通过实验介绍目前主流计算机存储系统及设计实现方法；第四章以时间并行性为特征介绍计算机系统的设计方法，着重体现重叠、流水等技术及设计方案；第五章以奔腾处理机为蓝本设计具有两条流水线的超标量模型机，从指令并行性的角度研究计算机系统的设计方法。

由于编者水平有限，加上计算机技术飞速发展，新的理念和技术层出不穷，在本书中难免会存在一些问题和错误，恳请广大读者批评指正。

编 者

2020 年 2 月

# 目　　录

**第 1 章　多通路的运算器和寄存器堆 ..... 1**

1.1　运算器与寄存器堆的结构 ..... 1
1.2　多通路的运算器与寄存器堆设计实验 ..... 3

**第 2 章　指令系统 ..... 7**

2.1　计算机系统的指令系统 ..... 7
2.2　基于 CISC 指令系统的模型机设计实验 ..... 10
2.3　基于 RISC 技术的模型机设计实验 ..... 25

**第 3 章　存储系统 ..... 31**

3.1　计算机的存储系统 ..... 31
3.2　FIFO 先进先出存储器实验 ..... 33
3.3　CACHE 控制器设计实验 ..... 36

**第 4 章　时间并行性为特征的计算机系统 ..... 41**

4.1　重叠和流水 ..... 41
4.2　具有指令预取功能的模型机设计实验 ..... 45
4.3　具有三级流水的模型机设计实验 ..... 54

**第 5 章　指令并行性为特征的计算机系统 ..... 62**

5.1　超标量处理机 ..... 62
5.2　具有两条流水线的超标量模型机设计实验 ..... 64

**附录 1　控制器单元和扩展单元对应 FPGA 引脚配置说明 ..... 74**

# 第 1 章　多通路的运算器和寄存器堆

在计算机组成原理的学习中，我们了解到计算机的一个重要的功能是处理各种算术和逻辑运算，这个功能要由 CPU 中的运算器来完成。现代的计算机中，参加运算的操作数和运算的结果大多都直接存储在寄存器堆中，研究运算器和寄存器堆的结构，有助于设计合理高效的计算机，从而提高计算机指令执行的效率。

## 1.1　运算器与寄存器堆的结构

运算器包括 ALU、暂存器、三态缓冲器、数据总线等部件。用于实现算术和逻辑运算功能。在现代的计算机中，运算器总是和通用寄存器连接在一起完成算术逻辑类指令的执行。所以运算器的设计,主要是围绕 ALU 和寄存器同数据总线之间如何传送操作数和运算结果进行的。传统的运算器与寄存器堆以及其他周边部件的连接方式有三种：

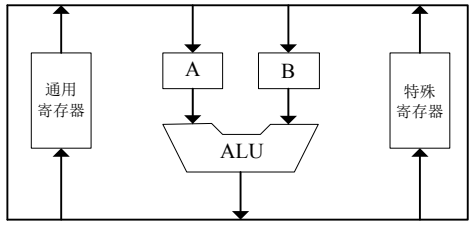



**图 1-1-1 单总线的运算器结构**

单总线结构的运算器如图 1-1-1 所示，所有部件都接到同一总线上。这种结构的运算器控制电路比较简单，在同一时间内,只能有一个操作数放在单总线上。为了把两个操作数输入到 ALU,需要分两次来做,而且还需要 A,B 两个缓冲寄存器。这种结构的主要缺点是操作速度较慢。

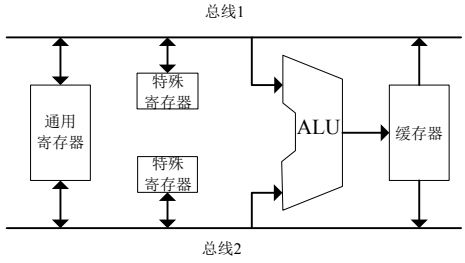



**图 1-1-2 双总线的运算器结构**

双总线结构的运算器如图 1-1-2 所示。在这种结构中,两个操作数同时加到 ALU 进行运算,只需一次操作控制,而且马上就可以得到运算结果。但 ALU 的输出不能直接加到总线上去。这

是因为,当形成操作结果的输出时,两条总线都被输入数占据,因而必须在 ALU 输出端设置缓冲寄存器，等到下一周期再输出运算器的结果到总线上。

三总线结构的运算器如图 1-1-3 所示。在三总线结构中,ALU 的两个输入端分别由两条总线供给,而 ALU 的输出则与第三条总线相连。这样,算术逻辑操作就可以在一步的控制之内完成。由于 ALU 本身有时间延迟,所以打入输出结果的选通脉冲必须考虑到包括这个延迟。另外,设置了一个总线旁路器。如果一个操作数不需要修改,而直接从总线 1 传送到总线 3,那么可以通过控制总线旁路器把数据传出;如果一个操作数传送时需要修改,那么就借助于 ALU。很显然,三总线结构的运算器的特点是操作速度快。

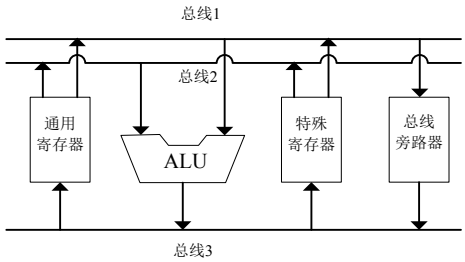



**图 1-1-3 三总线的运算器结构**

在上边的三种典型的运算器结构依次通过增加一定的硬件代价来提高运算器的执行效率。当前计算机发展的主流是标量机，指令的执行从串行执行方式发展到了并行执行方式，因而对于运算器和寄存器堆通路的设计提出了许多新的要求，通过后面的学习，会进一步了解到：在并行指令执行方式中会遇到各种相关问题，一个先进的运算器与寄存器堆的结构在有助于提高指令的执行效率的同时还有助于解决系统设计中的各种相关问题。

以上述的三种运算器的典型结构为蓝本，结合在并行系统设计中所遇到的各种相关问题，我们设计了多通路、多端口的结构灵活的运算器与寄存器堆。基本上涵盖了并行计算机体系中运算器的典型结构，为先进计算机系统的设计提供了良好的实验平台。

# 1.2 多通路的运算器与寄存器堆设计实验

## 1.2.1 实验目的

1． 了解多通路的运算器与寄存器堆的组成结构。
2． 掌握多通路的运算器与寄存器堆的工作原理及设计方法。

## 1.2.2 实验设备

PC 机一台， TDX-CMX 实验系统一套。

## 1.2.3 实验原理

1．ALU&REG 单元的结构

ALU&REG 单元由运算器和双端口寄存器堆构成，通过不同的控制信号 SEL1、SEL0 产生不同结构的运算器。运算器内部含有三个独立运算部件，分别为算术、逻辑和移位运算部件，要处理的数据存于暂存器 A 和暂存器 B。

SEL0 和 SEL1 用于选择运算器和寄存器堆的通路：

(1)当 SEL1=0、SEL0=0，ALU 的输出 D7…D0、REG（右口）的输出 OUT7…OUT0 和 ALU 与 REG 的输入 IN7…IN0 接到 CPU 内总线上时，如图 1-2-1 所示，寄存器堆只能从右口进行操作，相当于只有一组控制线的单端口寄存器堆，一般计算机组成原理实验涉及到的运算器和寄存器就是采用这种结构。

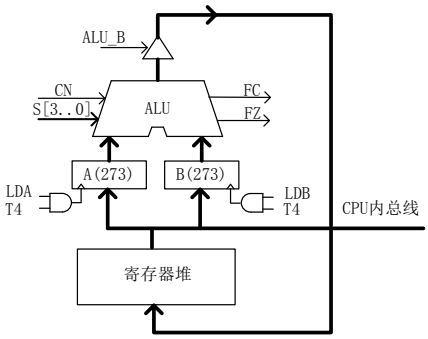



**图 1-2-1 单通道单端口的运算器和寄存器堆的结构**

(2)当 SEL1=1、SEL0=0，REG（右口）的输出 OUT7…OUT0 和 ALU 与 REG（右口）的输入 IN7…IN0 接到 CPU 内总线上时，运算器和双端口寄存器堆的结构如图 1-2-2 所示，寄存器堆由两组控制信号来分别进行控制，每组控制信号都可以相对独立的对寄存器堆进行读写操作，同时增加了执行专用通道 A 总线，以利于提高指令执行的效率。

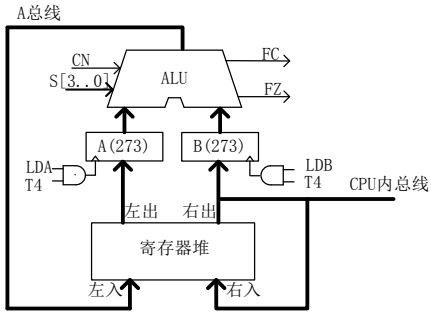



**图 1-2-2 双通道双端口的运算器和寄存器堆的结构**

(3)当 SEL1=1、SEL0=1，REG(右口)的输出 OUT7…OUT0 和 ALU 与 REG(右口)的输入 IN7…IN0 接到 CPU 内总线上时，运算器和双端口寄存器堆的结构如图 1-2-3 所示，在双通道双端口运算器和寄存器堆的基础上增加了暂存器旁路，把运算结果写回到寄存器堆的同时也可以写到暂存器 A、暂存器 B 中。由于在运算型指令中把运算的结果写到通用寄存器中的指令很多，占运算型指令的大多数，发生通用寄存器数据相关的概率相当高，因此，可以用硬件设置专用路径来解决这种通用寄存器数据相关问题。

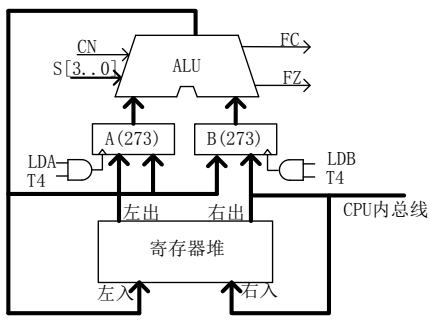



**图 1-2-3 增加暂存器旁路的双通道双端口的运算器和寄存器堆的结构**

上面介绍了运算器和寄存器堆的三种典型的数据通路图，在计算机组成原理这门课程中我们已经对运算器有了初步的了解，明白运算器的主要功能是完成算术和逻辑类运算。在系统结构这门课程中经过进一步的研究，还会了解到运算器与寄存器堆的结构对于计算机系统的设计有着重要的作用，对于计算机性能的优劣有着很大的影响。

2．ALU&REG 单元的应用

在了解运算器与寄存器堆结构的基础上，基于如图 1-2-3 所示的双通道双端口运算器和双端口寄存器堆的结构可以设计一段程序：从 IN 单元读入一个数据，存入 R0；从 IN 单元读入另一个数据，存于 R1；将 R0 和 R1 相加，结果存于 R0；将 R0 和 R1 相加，结果存于 R3，同时打入暂存器 A 中；再将 R0 的值送 OUT 单元显示。

根据指令要求，得出用时钟进行驱动的状态机描述，即得出其有限状态机，如图 1-2-4 所示。

下面分析每个状态中的基本操作：

S0：空操作，系统复位后的状态

S1：IN->R0;从 IN 单元往 R0 中打一个数

S2：IN->R1; 从 IN 单元往 R1 中打一个数

S3：R0 ->A, R1 ->B;同时把 R0、R1 中的数打入暂存器 A、B 中

S4：A+B->R0;将 A+B 的结果送往 R0

S5：A+B->R3，A+B->A; 增加暂存器旁路，将 A+B 的结果送往 R3 的同时打入暂存器 A 中

S6：R0->OUT;把 R0 中的数送入输出单元显示。

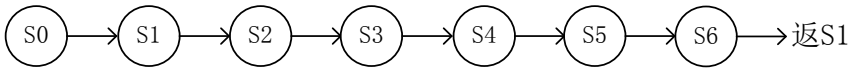



**图 1-2-4 状态机描述**

### 1.2.4 实验步骤

(1) 分析每个状态所需的控制信号，并汇总成表，如表 1-2-1 所示。控制信号由左至右，依次为：SEL1，SEL0，WR，RD，IOM，S3，S2，S1，S0，LDA，LDB，LDR0，LDR1，LDR2，LDR3，R0_B，R1_B，R2_B，R3_B，LLDR0，LLDR1，LLDR2，LLDR3，LR0_B，LR1_B，LR2_B，LR3_B。

**表 1-2-1 控制信号表**

| 状态号 | 控制信号 |
|---|---|
| S0 | 10000000000000001111100001111 |
| S1 | 10011000000100011111000011111 |
| S2 | 10011000000010011111000011111 |
| S3 | 10000000011000010110000011111 |
| S4 | 10000100100000011111000111111 |
| S5 | 11000100110000011111000111111 |
| S6 | 10101000000000000111000011111 |

(2) 本实验在“控制器单元”中的 FPGA 中设计实现。“控制器单元”中 FPGA 与微程序控制器的信号复用，其定义如附录 2 所示。

(3) 用 VHDL 语言来设计本实验的状态机，FPGA 中对应的引脚如图 1-2-5 所示。使用 Quartus 软件编辑 VHDL 文件并进行编译。

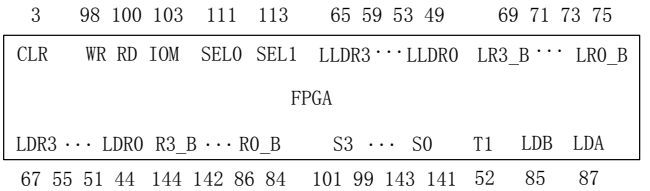



**图 1-2-5 引脚电路图**

(4) 关闭实验系统电源，把时序与操作台单元的“MODE”短路块插上，使系统工作在四节拍模式，JP1，JP2 短路块为 1、2 短接，按图 1-2-6 连接实验电路。

(5) 打开实验系统电源，将下载电缆插入控制器单元的 C_JTAG 口，把生成的 SOF 文件下载到 FPGA 单元中去。

(6) 在 PC 机上运行 TDX-CMX，进入联机软件界面，选择菜单命令“【实验】—【ALU&REG 实验】”，打开数据通路图，按动 CON 单元的总清按钮 CLR，使程序计数器 PC 地址清零，状态机回到 S0，程序从头开始运行，选择相应的功能命令，即可联机调试、运行程序。

(7) 当模型机执行完一遍后，检查 OUT 单元显示的数是否正确，按下 CON 单元的总清按钮 CLR，改变 IN 单元的值，再次执行机器程序，从 OUT 单元显示的数判别程序执行是否正确。

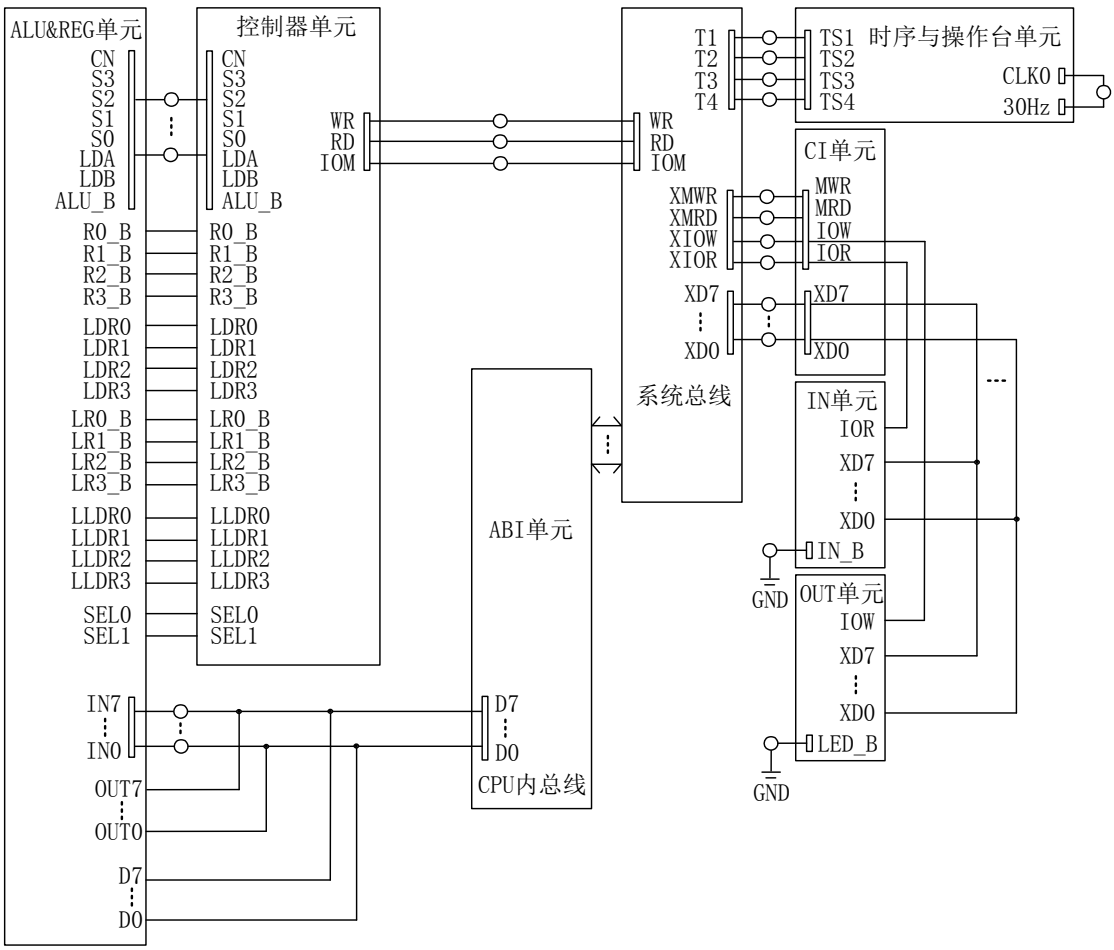



**图 1-2-6 实验接线图**

# 第 2 章　指令系统

指令系统是计算机系统的主要组成部分之一，是计算机系统设计中软件设计与硬件设计的一个主要分界面，也是两者之间沟通的桥梁。与软件设计主要考虑如何采用指令系统来实现既定的功能不同，硬件设计主要针对如何实现指令系统来进行。

## 2.1　计算机系统的指令系统

如果把计算机系统所要实现的功能分为一些基本的功能，那么在这些基本的功能中只有很少的一部分必须由硬件指令系统来实现。绝大多数功能既可以用硬件指令系统实现，也可以用软件的一段子程序来实现。对于指令系统的设计者而言，决定一个功能该如何实现时，要考虑到三个因素：速度、价格和灵活性。用硬件指令系统实现：速度快、价格高、灵活性差；用软件指令系统实现：速度慢、价格低、灵活性好。

设计通用计算机时，要保证指令系统的完整性。对于以下的五类指令要有足够的硬件指令系统支持：数据传送类指令、运算类指令、程序控制类指令、输入输出指令、处理机控制指令和调试指令。

对于计算机指令系统的设计有两种截然不同的思路：CISC（复杂指令系统）和 RISC（精简指令系统）。

采用 CISC 结构设计的计算机包含大量指令的指令系统和各种各样的寻址方式，期望使编译器设计者的任务变的容易；提供更复杂、更精致的高级语言的支持。但这样做就会使指令系统变的越来越庞大。总体来说，CISC 具有如下的特点：

(1) 指令系统复杂。具体表现在指令数多、寻址方式多、指令格式多。

(2) 绝大多数指令需要多个时钟周期才能执行完成。

(3) 各种指令都可访问存储器。

(4) 采用微程序控制。

(5) 设置专用的寄存器。

(6) 难以通过优化编译生成高效的目标代码程序。

CISC 结构是早期指令系统的代表，期望通过提供更复杂、更精致的高级语言的支持来提高计算机的性能。1975 年，IBM 公司率先组织技术力量研究指令系统的合理性问题，这是在指令系统方面的一次有益的探索。从 1979 年开始，美国加州大学伯克利分校的研究小组开展这方面的研究工作，经过细致的研究，他们指出 CISC 的结构和思路存在如下一些问题：

(1) 大量的统计数字表明，大约有 80%的指令只有在 20%的处理机运行时间内才被用到。所以对操作繁杂的指令，不仅增加机器设计人员的负担，也降低了系统的性能价格比。

(2) VLSI（超大规模集成电路）技术的飞速发展，VLSI 工艺要求规整性，而 CISC 处理机中，为了实现大量的复杂指令，控制逻辑极不规整，给 VLSI 工艺造成很大的困难。

(3) 由于许多指令的操作繁杂，使得执行速度很低，甚至比用几条简单的指令来组合实现还要慢。而且由于庞大的指令系统，使得难以优化编译生成真正高效率的机器语言程序，也使编译程序本身太长、太复杂。

针对 CISC 结构存在的这些问题，人们提出了 RISC 的思想：

(1) 确定指令系统时，选取使用频率最高的一些简单指令，以及很有用但不复杂的指令。

(2) 指令长度固定，指令格式简单而统一，限制在 1～2 种之内。大大减少指令系统的寻址方式，寻址方式简单，一般不超过 2 种。

(3) 大部分指令在一个机器周期内完成。

(4) 只有取（LOAD）、存（STORE）指令可以访问存储器，其他指令的操作一律在寄存器间进行，大大增加寄存器的数量。

(5) 以硬布线控制为主，很少或不用微程序控制。

(6) 特别重视编译优化工作，支持高级语言的实现。

进入 20 世纪 80 年代以来，VLSI（超大规模集成电路）技术的迅速发展对于指令系统的发展产生了深远的影响。CISC 由于指令不规整，不利于大规模的集成，而 RISC 由于规整的指令结构、简单的控制逻辑和大量相同的通用寄存器适合 VLSI 的实现，逐渐成为主流的现代计算机指令系统。

目前在 RISC 处理机中采用如下几种技术：

（1）延时转移技术

在 RISC 处理机中，指令一般采用流水线方式工作。取指令和执行指令并行进行。如果取指令和执行指令各需要一个周期，那么，在正常情况下，每个周期就能执行完一条指令。然而，在遇到转移指令时，流水就有可能断流。由于转移的目的地址要在指令执行完后才能产生，这时下一条指令已经取出来了，因此，必须把取出来的指令作废，并按照转移地址重新取出正确指令。为解决上述问题，可以使编译器自动调整指令序列，在转移指令后插入一条有效的指令，而转移指令好象被延迟执行了，这种技术称为延迟转移技术。

然而必须注意，调整指令序列时一定不能改变原程序的数据相关关系，如果找不到合适的指令调整程序中的指令序列，编译程序可以在转移指令后插入一条空操作指令。

（2）在处理器中设置数量较大的寄存器组，并采用重叠寄存器窗口技术

由于在 RISC 程序中有很多的 CALL 和 RETURN 指令。在执行 CALL 指令时，必须保存现场，另外，还要把执行子程序的参数从主程序中传送出去。在执行 RETURN 指令时，要把保存的结果传送回主程序。为了尽量减少访问存储器，在 RISC 处理器中采用重叠寄存器窗口技术。

（3）硬布线实现为主微程序固件实现为辅

主要采用硬布线逻辑来实现指令系统，对于那些必须的少量的复杂指令，可以采用微程序实现。微程序便于实现复杂指令，便于修改指令系统，增加了机器的灵活性和适应性，但执行速度低。

（4）强调优化编译系统设计

编译器必须努力优化寄存器的分配和使用，提高寄存器的使用效率，减少访问存储器的次数。为了使 RISC 处理机中的流水线高效率的工作，尽量不断流，编译器还必须分析程序的数据流和控制流，当发现有可能断流时，要调整指令序列。对有些可以通过变量重新命名来消除数据相关，要尽量消除。这样，可以提高流水线的执行效率，缩短程序的执行时间。

然而，相比于 CISC，RISC 在解决了 CISC 的问题的同时，引入了一些新的问题：指令的优化编译变得困难，在考虑功能实现的同时要考虑各种相关问题，要设计复杂的子程序库等。所以

现代计算机的指令系统以性价比为基准，并不拘泥于单一的指令系统。现代计算机处理器的设计主要遵循下述的基本思想：

(1) 所有指令由硬件直接执行（而不再由微指令解释的方式执行）。
(2) 最大限度的提高指令启动速度。
(3) 指令应易于译码。
(4) 只允许少数指令访问内存（从内存中读取指令是执行速度的瓶颈）。
(5) 提供了足够多的寄存器（寄存器的存取速度远远大于存储器）。

# 2.2 基于 CISC 指令系统的模型机设计实验

## 2.2.1 实验目的

综合运用所学计算机组成原理知识，设计 CISC 指令系统的计算机。

## 2.2.2 实验设备

PC 机一台， TDX-CMX 实验系统一套。

## 2.2.3 实验原理

1．指令系统设计

采用 CISC 思想，本模型机设计了包括运算、控制转移，数据传送共三大类十五条指令。其中，运算类指令包含算术运算、逻辑运算和移位运算，设计有 6 条指令，分别为 ADD、AND、INC、SUB、OR、RR，所有运算类指令都为单字节指令，寻址方式采用寄存器直接寻址；控制转移类指令有三条 HLT、JMP、BZC，用以控制程序的分支和转移，其中 HLT 为单字节指令，JMP 和 BZC 为双字节指令；数据传送类指令有 IN、OUT、MOV、LDI、LAD、STA 共 6 条，用以完成寄存器和寄存器、寄存器和 I/O、寄存器和存储器之间的数据交换，除 MOV 指令为单字节指令外，其余均为双字节指令。模型机的指令格式可定义如下：

（1）所有单字节指令（ADD、AND、INC、SUB、OR、RR、HLT 和 MOV）的指令格式

| 7 6 5 4 | 3 2 | 1 0 |
|---|---|---|
| OP-CODE | RS | RD |

其中 OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，并规定：

| RS 或 RD | 选定的寄存器 |
|---|---|
| 00 | R0 |
| 01 | R1 |
| 10 | R2 |
| 11 | R3 |

（2）IN 和 OUT 的指令格式为：

| 7 6 5 4 （1） | 3 2 （1） | 1 0 （1） | 7—0 （2） |
|---|---|---|---|
| OP-CODE | RS | RD | P |

其中括号中的 1 表示指令的第一字节，2 表示指令的第二字节，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，P 为 I/O 端口号，占用一个字节。

（3）LDI 的指令格式为：

| 7 6 5 4 （1） | 3 2 （1） | 1 0 （1） | 7—0 （2） |
|---|---|---|---|
| OP-CODE | RS | RD | data |

其中 OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器 data 为立即数。

（4）LAD、STA、JMP 和 BZC 指令格式如下。

| 7 6 5 4 （1） | 3 2 （1） | 1 0 （1） | 7—0 （2） |
|---|---|---|---|
| OP-CODE | M | RD | D |

其中 OP-CODE 为操作码，RD 为寄存器，D 为操作数，M 为寻址模式，具体见表 2-2-1。（以 R2 做为变址寄存器 RI）

**表 2-2-1 寻址方式**

| 寻址模式 M | 有效地址 E | 说 明 |
|---|---|---|
| 00 | E = D | 直接寻址 |
| 01 | E =（D） | 间接寻址 |
| 10 | E =（RI）+ D | RI 变址寻址 |
| 11 | E =（PC）+ D | 相对寻址 |

由此可知，指令系统设计了五种数据寻址方式，即立即、直接、间接、变址和相对寻址，LDI 指令为立即寻址，LAD、STA、JMP 和 BZC 指令均具备直接、间接、变址和相对寻址能力。

另外，本模型机也规定一律采用定点补码表示法表示数据，字长为 8 位，8 位全用来表示数据（最高位不表示符号），数值表示范围是： $0 \leqslant X \leqslant 2^8 - 1$。

为了便于使用，表 2-2-2 列出了各条指令的格式、汇编符号和指令功能。

**表 2-2-2 指令描述**

| 助记符号 | 指令格式 | | | | 指令功能 |
|---|---|---|---|---|---|
| MOV RD,RS | 0100 | RS | RD | | RS → RD |
| ADD RD,RS | 0000 | RS | RD | | RD + RS → RD |
| SUB RD,RS | 1000 | RS | RD | | RD - RS → RD |
| AND RD,RS | 0001 | RS | RD | | RD∧RS → RD |
| OR RD,RS | 1001 | RS | RD | | RD∨RS → RD |
| RR RD,RS | 1010 | RS | RD | | RS右环移 → RD |
| INC RD | 0111 | ** | RD | | RD+1 → RD |
| LAD M D,RD | 1100 | M | RD | D | E → RD |
| STA M D,RS | 1101 | M | RD | D | RD → E |
| JMP M D | 1110 | M | ** | D | E → PC |
| BZC M D | 1111 | M | ** | D | 当FC或FZ=1时, E → PC |
| IN RD,P | 0010 | ** | RD | P | [P] → RD |
| OUT P,RS | 0011 | RS | ** | P | RS → [P] |
| LDI RD,D | 0110 | ** | RD | D | D → RD |
| HALT | 0101 | ** | ** | | 停机 |

**2．模型计算机系统结构的设计**

**根据上面的指令系统设计要求，本模型机的数据通路图可设计如下：**

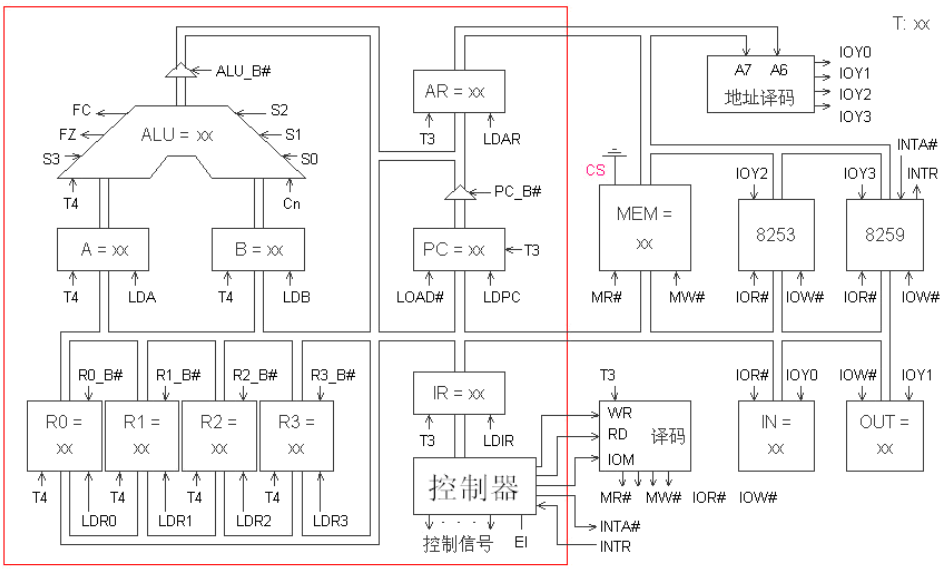



**图 2-2-1　数据通路框图**

这里，我们采用单总线、微程序控制器方案来构建本模型机。机器涉及到的各主要功能部件，由于实验系统都已经以部件单元电路形式给出，直接应用就可以。下面仅就指令的译码逻辑设计、系统的 I/O 地址译码、微程序控制器设计这几方面的情况来做一说明。

（1）指令的译码逻辑设计

图 2-2-2 是本模型机的指令译码电路原理图，该电路已经在实验系统的“控制器单元”的 INS_DEC 中实现。

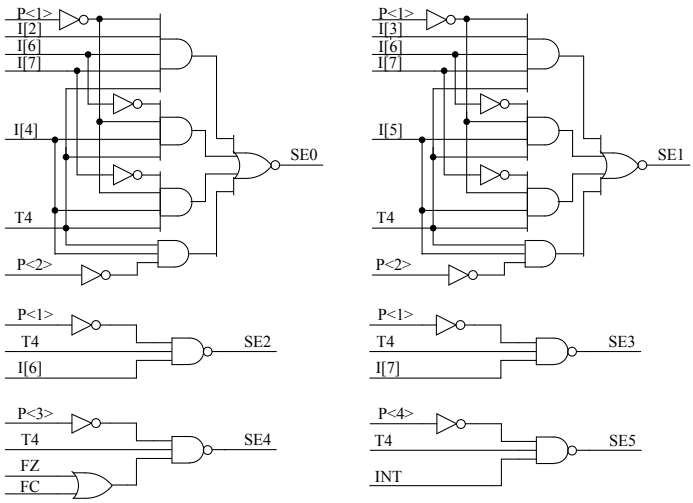



**图 2-2-2　指令译码原理图**

图 2-2-3 是本模型机的寄存器译码电路，该电路也已经在实验系统的“控制器单元”的 REG_DEC 中实现。

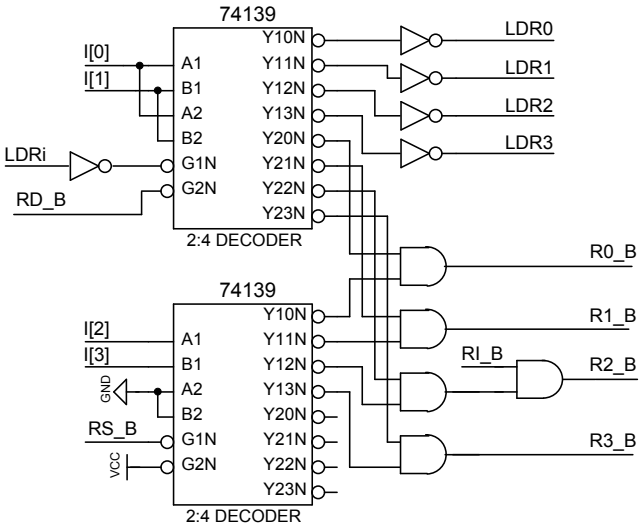



**图 2-2-3 寄存器译码原理图**

（2）系统的 I/O 地址译码

图 2-2-4 是模型机地址总线的 I/O 地址译码（在实验系统的地址总线单元）。

由于用的是地址总线的高两位进行译码，I/O 地址空间被分为四个区，如表 2-2-3 所示：

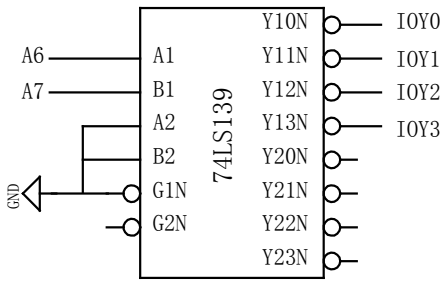



**图 2-2-4 I/O 地址译码原理图**

**表 2-2-3 I/O 地址空间分配**

| A7 A6 | 选定 | 地址空间 |
|---|---|---|
| 00 | IOY0 | 00-3F |
| 01 | IOY1 | 40-7F |
| 10 | IOY2 | 80-BF |
| 11 | IOY3 | C0-FF |

（3）微程序控制器设计

根据模型计算机的指令系统要求，可设计如图 2-2-5 所示的微指令流程图。

按照系统建议的微指令格式，见表 2-2-4，参照微指令流程图，可将每条微指令代码化，译成二进制代码表，见表 2-2-5，并将二进制代码表转换为联机操作时的十六进制格式文件。

**表 2-2-4 微指令格式**

| 23 | 22 | 21 | 20 | 19 | 18-15 | 14-12 | 11-9 | 8-6 | 5-0 |
|---|---|---|---|---|---|---|---|---|---|
| M23 | CN | WR | RD | IOM | S3-S0 | A字段 | B字段 | C字段 | UA5-UA0 |

A字段

| 14 | 13 | 12 | 选择 |
|---|---|---|---|
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | LDA |
| 0 | 1 | 0 | LDB |
| 0 | 1 | 1 | LDRi |
| 1 | 0 | 0 | 保留 |
| 1 | 0 | 1 | LOAD |
| 1 | 1 | 0 | LDAR |
| 1 | 1 | 1 | LDIR |

B字段

| 11 | 10 | 9 | 选择 |
|---|---|---|---|
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | ALU_B |
| 0 | 1 | 0 | RS_B |
| 0 | 1 | 1 | RD_B |
| 1 | 0 | 0 | RI_B |
| 1 | 0 | 1 | 保留 |
| 1 | 1 | 0 | PC_B |
| 1 | 1 | 1 | 保留 |

C字段

| 8 | 7 | 6 | 选择 |
|---|---|---|---|
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | P<1> |
| 0 | 1 | 0 | P<2> |
| 0 | 1 | 1 | P<3> |
| 1 | 0 | 0 | 保留 |
| 1 | 0 | 1 | LDPC |
| 1 | 1 | 0 | 保留 |
| 1 | 1 | 1 | 保留 |

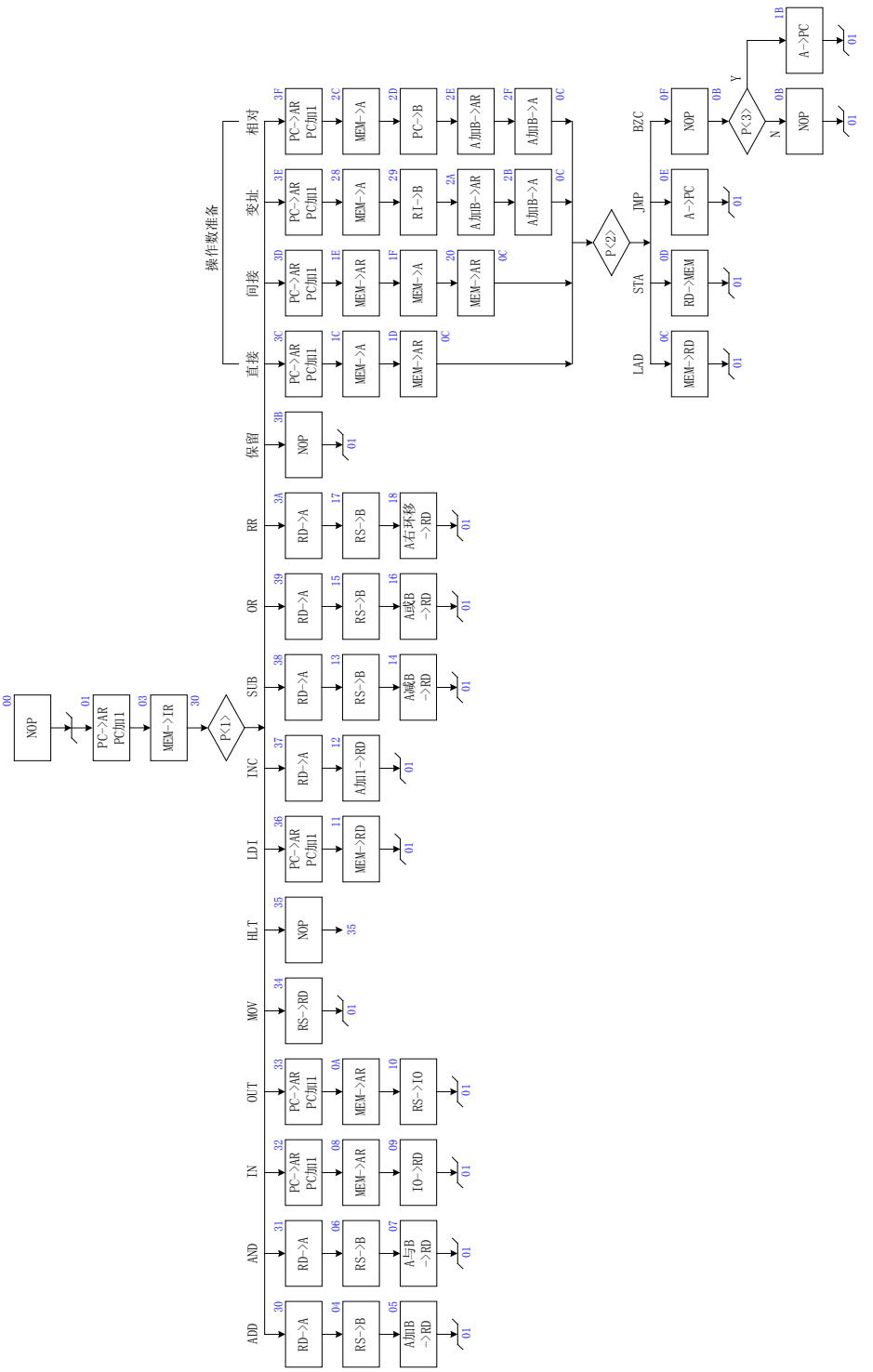

NOP
PC->AR PC加1
MEM->IR
P<1>
操作数准备
直接
间接
变址
相对
保留
RR
OR
SUB
INC
LDI
HLT
MOV
OUT
IN
AND
ADD
P<2>
LAD
STA
JMP
BZC
P<3>
Y
N
A->PC
RD->A
RS->B
A加B ->RD
A与B ->RD
IO->RD
RS->IO
MEM->AR
RS->RD
MEM->RD
A加1->RD
A减B ->RD
A或B ->RD
A右环移 ->RD
MEM->A
A加B->AR
A加B->A
RI->B
RD->MEM
A->PC

**图 2-2-5 微程序流程图**

**表 2-2-5 二进制代码表**

| 地址 | 十六进制表示 | 高五位 | S3-S0 | A 字段 | B 字段 | C 字段 | UA5-UA0 |
|---|---|---|---|---|---|---|---|
| 00 | 00 00 01 | 00000 | 0000 | 000 | 000 | 000 | 000001 |
| 01 | 00 6D 43 | 00000 | 0000 | 110 | 110 | 101 | 000011 |
| 03 | 10 70 70 | 00010 | 0000 | 111 | 000 | 001 | 110000 |
| 04 | 00 24 05 | 00000 | 0000 | 010 | 011 | 000 | 000101 |
| 05 | 04 B2 01 | 00000 | 1001 | 011 | 001 | 000 | 000001 |
| 06 | 00 24 07 | 00000 | 0000 | 010 | 011 | 000 | 000111 |
| 07 | 01 32 01 | 00000 | 0010 | 011 | 001 | 000 | 000001 |
| 08 | 10 60 09 | 00010 | 0000 | 110 | 000 | 000 | 001001 |
| 09 | 18 30 01 | 00011 | 0000 | 011 | 000 | 000 | 000001 |
| 0A | 10 60 10 | 00010 | 0000 | 110 | 000 | 000 | 010000 |
| 0B | 00 00 01 | 00000 | 0000 | 000 | 000 | 000 | 000001 |
| 0C | 10 30 01 | 00010 | 0000 | 011 | 000 | 000 | 000001 |
| 0D | 20 06 01 | 00100 | 0000 | 000 | 001 | 100 | 000001 |
| 0E | 00 53 41 | 00000 | 0000 | 101 | 001 | 101 | 000001 |
| 0F | 00 00 CB | 00000 | 0000 | 000 | 000 | 011 | 001011 |
| 10 | 28 04 01 | 00101 | 0000 | 000 | 010 | 000 | 000001 |
| 11 | 10 30 01 | 00010 | 0000 | 011 | 000 | 000 | 000001 |
| 12 | 06 B2 01 | 00000 | 1101 | 011 | 001 | 000 | 000001 |
| 13 | 00 24 14 | 00000 | 0000 | 010 | 011 | 000 | 010100 |
| 14 | 05 B2 01 | 00000 | 1011 | 011 | 001 | 000 | 000001 |
| 15 | 00 24 16 | 00000 | 0000 | 010 | 011 | 000 | 010110 |
| 16 | 01 B2 01 | 00000 | 0011 | 011 | 001 | 000 | 000001 |
| 17 | 00 24 18 | 00000 | 0000 | 010 | 011 | 000 | 011000 |
| 18 | 02 B2 01 | 00000 | 0101 | 011 | 001 | 000 | 000001 |
| 1B | 00 53 41 | 00000 | 0000 | 101 | 001 | 101 | 000001 |
| 1C | 10 10 1D | 00010 | 0000 | 001 | 000 | 000 | 011101 |
| 1D | 10 60 8C | 00010 | 0000 | 110 | 000 | 010 | 001100 |
| 1E | 10 60 1F | 00010 | 0000 | 110 | 000 | 000 | 011111 |
| 1F | 10 10 20 | 00010 | 0000 | 001 | 000 | 000 | 100000 |
| 20 | 10 60 8C | 00010 | 0000 | 110 | 000 | 010 | 001100 |
| 28 | 10 10 29 | 00010 | 0000 | 001 | 000 | 000 | 101001 |
| 29 | 00 28 2A | 00000 | 0000 | 010 | 100 | 000 | 101010 |
| 2A | 04 E2 2B | 00000 | 1001 | 110 | 001 | 000 | 101011 |
| 2B | 04 92 8C | 00000 | 1001 | 001 | 001 | 010 | 001100 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2C | 10 10 2D | 00010 | 0000 | 001 | 000 | 000 | 101101 |
| 2D | 00 2C 2E | 00000 | 0000 | 010 | 110 | 000 | 101110 |
| 2E | 04 E2 2F | 00000 | 1001 | 110 | 001 | 000 | 101111 |
| 2F | 04 92 8C | 00000 | 1001 | 001 | 001 | 010 | 001100 |
| 30 | 00 16 04 | 00000 | 0000 | 001 | 011 | 000 | 000100 |
| 31 | 00 16 06 | 00000 | 0000 | 001 | 011 | 000 | 000110 |
| 32 | 00 6D 48 | 00000 | 0000 | 110 | 110 | 101 | 001000 |
| 33 | 00 6D 4A | 00000 | 0000 | 110 | 110 | 101 | 001010 |
| 34 | 00 34 01 | 00000 | 0000 | 011 | 010 | 000 | 000001 |
| 35 | 00 00 35 | 00000 | 0000 | 000 | 000 | 000 | 110101 |
| 36 | 00 6D 51 | 00000 | 0000 | 110 | 110 | 101 | 010001 |
| 37 | 00 16 12 | 00000 | 0000 | 001 | 011 | 000 | 010010 |
| 38 | 00 16 13 | 00000 | 0000 | 001 | 011 | 000 | 010011 |
| 39 | 00 16 15 | 00000 | 0000 | 001 | 011 | 000 | 010101 |
| 3A | 00 16 17 | 00000 | 0000 | 001 | 011 | 000 | 010111 |
| 3B | 00 00 01 | 00000 | 0000 | 000 | 000 | 000 | 000001 |
| 3C | 00 6D 5C | 00000 | 0000 | 110 | 110 | 101 | 011100 |
| 3D | 00 6D 5E | 00000 | 0000 | 110 | 110 | 101 | 011110 |
| 3E | 00 6D 68 | 00000 | 0000 | 110 | 110 | 101 | 101000 |
| 3F | 00 6D 6C | 00000 | 0000 | 110 | 110 | 101 | 101100 |

3．模型计算机测试

应用上面定义的指令系统，在模型机上编程实现以下的操作：

从 IN 单元读入一个数据，根据读入数据的低 4 位值 X，求 1+2+…+X 的累加和，最后将结果存放到 70H 地址单元并在 OUT 单元输出显示。

根据要求可以设计如下程序。(地址和内容均为二进制数)

| 地 址 | 内 容 | 助记符 | 说 明 |
|---|---|---|---|
| 00000000 | 00100000 | ; START: IN R0,00H | 从 IN 单元读入计数初值 |
| 00000001 | 00000000 | | |
| 00000010 | 01100001 | ; LDI R1,0FH | 立即数 0FH 送 R1 |
| 00000011 | 00001111 | | |
| 00000100 | 00010100 | ; AND R0,R1 | 得到 R0 低四位 |
| 00000101 | 01100001 | ; LDI R1,00H | 装入和初值 00H |
| 00000110 | 00000000 | | |
| 00000111 | 11110000 | ; BZC RESULT | 计数值为 0 则跳转 |
| 00001000 | 00010110 | | |
| 00001001 | 01100010 | ; LDI R2,60H | 读入数据始地址 |
| 00001010 | 01100000 | | |
| 00001011 | 11001011 | ; LOOP:LAD R3,[RI],00H | 从 MEM 读入数据送 R3，变址寻址，偏移量为 00H |

```
00001100   00000000
00001101   00001101    ; ADD  R1,R3              累加求和
00001110   01110010    ; INC  RI                 变址寄存加 1，指向下一数据
00001111   01100011    ; LDI  R3,01H             装入比较值
00010000   00000001
00010001   10001100    ; SUB  R0,R3
00010010   11110000    ; BZC  RESULT             相减为 0，表示求和完毕
00010011   00010110
00010100   11100000    ; JMP  LOOP               未完则继续
00010101   00001011
00010110   11010001    ; RESULT: STA  70H,R1     和存于 MEM 的 70H 单元
00010111   01110000
00011000   00110100    ; OUT  40H,R1             和在 OUT 单元显示
00011001   01000000
00011010   11100000    ; JMP START               跳转至 START
00011011   00000000
00011100   01010000    ; HLT                     停机
01100000   00000001    ;                         数据
01100001   00000010
01100010   00000011
01100011   00000100
01100100   00000101
01100101   00000110
01100110   00000111
01100111   00001000
01101000   00001001
01101001   00001010
01101010   00001011
01101011   00001100
01101100   00001101
01101101   00001110
01101110   00001111
```

### 2.2.4 实验步骤

1. 关闭电源，把时序与操作台单元的“MODE”短路块插上，使系统工作在四节拍模式，JP1、JP2 用短路块将 1、2 短接，按图 2-2-6 连接实验线路，检查无误后打开实验箱电源。

2. 写入实验程序，并进行校验，分两种方式，手动写入和联机写入。

1) 手动写入和校验

(1) 手动写入微程序

① 将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘编程’档，KK4 置为‘控存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD15——SD10 给出微地址，IN 单元给出低 8 位应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元的低 8 位。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ IN 单元给出中 8 位应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元的中 8 位。IN 单元给出高 8 位应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元的高 8 位。

⑤ 重复①、②、③、④四步，将表 2-2-5 的微代码写入 2816 芯片中。

(2) 手动校验微程序

① 将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘校验’档，KK4 置为‘控存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD15——SD10 给出微地址，连续两次按动时序与操作台的开关 ST，MC 单元的数据指示灯 M7——M0 显示该单元的低 8 位。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ 连续两次按动时序与操作台的开关 ST，MC 单元的数据指示灯 M15——M8 显示该单元的中 8 位，再连续两次按动时序与操作台的开关 ST，则 MC 单元的数据指示灯 M23——M16 显示该单元的高 8 位。

⑤ 重复①、②、③、④四步，完成对微代码的校验。如果校验出微代码写入错误，重新写入、校验，直至确认微指令的输入无误为止。

(3)手动写入机器程序

① 将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘编程’档，KK4 置为‘主存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD17——SD10 给出地址，IN 单元给出该单元应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该存储器单元。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ IN 单元给出下一地址（地址自动加 1）应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元中。然后地址会自加 1，只需在 IN 单元输入后续地址的数据，连续两次按动时序与操作台的开关 ST，即可完成对该单元的写入。

⑤ 重复①、②两步，将所有机器指令写入主存芯片中。

(4)手动校验机器程序

① 将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘校验’档，KK4 置为‘主存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD17——SD10 给出地址，连续两次按动时序与操作台的开关 ST，CPU 内总线的数据指示灯 D7——D0 显示该单元的数据。

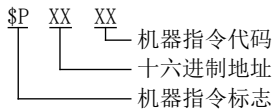
③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ 连续两次按动时序与操作台的开关 ST，地址自动加 1，CPU 内总线的数据指示灯 D7——D0 显示该单元的数据。此后每两次按动时序与操作台的开关 ST，地址自动加 1，CPU 内总线的指数据指示灯 D7——D0 显示该单元的数据，继续进行该操作，直至完成校验。如发现错误，则返回写入，然后校验，直至确认输入的所有指令准确无误。

2) 联机写入和校验

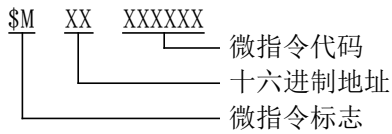
联上 PC 机，可运行 TDX-CMX 联机软件。联机软件提供了微程序和机器程序编辑和下载的功能，以代替手动读/写微程序和机器程序。机器程序以及微程序均以指定的格式写入到以 TXT 为后缀的文件中，机器指令的格式如下：

机器指令格式说明：




如$P 1F 11，表示机器指令的地址为 1FH，指令值为 11H；而微程序的格式如下：

微指令格式说明：




如$M 1F 112233，表示微指令的地址为 1FH，微指令值为 11H（高）、22H（中）、33H（低）。

本次实验程序可编辑如下，其中的分号‘;’为注释符，分号后面的内容在下载时将被忽略掉。

```
; //*************************************//
; //       复杂模型机实验指令文件          //
; //                                       //
; //       By TangDu CO.,LTD               /
; //*************************************//
; //****** Start Of Main Memory Data ******//
  $P 00 20    ; START: IN R0,00H      从 IN 单元读入计数初值
  $P 01 00
  $P 02 61    ; LDI  R1,0FH           立即数 0FH 送 R1
  $P 03 0F
  $P 04 14    ; AND  R0,R1            得到 R0 低四位
  $P 05 61    ; LDI  R1,00H           装入和初值 00H
  $P 06 00
  $P 07 F0    ; BZC  RESULT           计数值为 0 则跳转
  $P 08 16
  $P 09 62    ; LDI  R2,60H           读入数据始地址
  $P 0A 60
  $P 0B CB    ; LOOP:LAD R3,[RI],00H 从 MEM 读入数据送 R3，
  $P 0C 00                            变址寻址，偏移量为 00H
  $P 0D 0D    ; ADD  R1,R3            累加求和
  $P 0E 72    ; INC  RI               变址寄存加 1，指向下一数据
  $P 0F 63    ; LDI  R3,01H           装入比较值
  $P 10 01
  $P 11 8C    ; SUB  R0,R3
  $P 12 F0    ; BZC  RESULT           相减为 0，表示求和完毕
  $P 13 16
  $P 14 E0    ; JMP  LOOP             未完则继续
  $P 15 0B
  $P 16 D1    ; RESULT: STA  70H,R1   和存于 MEM 的 70H 单元
  $P 17 70
  $P 18 34    ; OUT  40H,R1           和在 OUT 单元显示
  $P 19 40
```

```
  $P 1A E0    ; JMP START             跳转至 START
  $P 1B 00
  $P 1C 50    ; HLT                   停机
  $P 60 01    ; 数据
  $P 61 02
  $P 62 03
  $P 63 04
  $P 64 05
  $P 65 06
  $P 66 07
  $P 67 08
  $P 68 09
  $P 69 0A
  $P 6A 0B
  $P 6B 0C
  $P 6C 0D
  $P 6D 0E
  $P 6E 0F
; //***** End Of Main Memory Data *****//

; //** Start Of MicroController Data **//
  $M 00 000001    ; NOP
  $M 01 006D43    ; PC->AR, PC 加 1
  $M 03 107070    ; MEM->IR, P<1>
  $M 04 002405    ; RS->B
  $M 05 04B201    ; A 加 B->RD
  $M 06 002407    ; RS->B
  $M 07 013201    ; A 与 B->RD
  $M 08 106009    ; MEM->AR
  $M 09 183001    ; IO->RD
  $M 0A 106010    ; MEM->AR
  $M 0B 000001    ; NOP
  $M 0C 103001    ; MEM->RD
  $M 0D 200601    ; RD->MEM
  $M 0E 005341    ; A->PC
  $M 0F 0000CB    ; NOP, P<3>
  $M 10 280401    ; RS->IO
  $M 11 103001    ; MEM->RD
  $M 12 06B201    ; A 加 1->RD
  $M 13 002414    ; RS->B
  $M 14 05B201    ; A 减 B->RD
  $M 15 002416    ; RS->B
  $M 16 01B201    ; A 或 B->RD
  $M 17 002418    ; RS->B
  $M 18 02B201    ; A 右环移->RD
  $M 1B 005341    ; A->PC
  $M 1C 10101D    ; MEM->A
  $M 1D 10608C    ; MEM->AR, P<2>
  $M 1E 10601F    ; MEM->AR
  $M 1F 101020    ; MEM->A
  $M 20 10608C    ; MEM->AR, P<2>
  $M 28 101029    ; MEM->A
  $M 29 00282A    ; RI->B
  $M 2A 04E22B    ; A 加 B->AR
  $M 2B 04928C    ; A 加 B->A, P<2>
```

```
$M 2C 10102D    ; MEM->A
$M 2D 002C2E    ; PC->B
$M 2E 04E22F    ; A 加 B->AR
$M 2F 04928C    ; A 加 B->A, P<2>
$M 30 001604    ; RD->A
$M 31 001606    ; RD->A
$M 32 006D48    ; PC->AR, PC 加 1
$M 33 006D4A    ; PC->AR, PC 加 1
$M 34 003401    ; RS->RD
$M 35 000035    ; NOP
$M 36 006D51    ; PC->AR, PC 加 1
$M 37 001612    ; RD->A
$M 38 001613    ; RD->A
$M 39 001615    ; RD->A
$M 3A 001617    ; RD->A
$M 3B 000001    ; NOP
$M 3C 006D5C    ; PC->AR, PC 加 1
$M 3D 006D5E    ; PC->AR, PC 加 1
$M 3E 006D68    ; PC->AR, PC 加 1
$M 3F 006D6C    ; PC->AR, PC 加 1
; //** End Of MicroController Data **//
```

选择联机软件的“【转储】—【装载】”功能，在打开文件对话框中选择上面所保存的文件，软件自动将机器程序和微程序写入指定单元。

选择联机软件的“【转储】—【刷新指令区】”可以读出模型机的所有的机器指令和微指令，并在指令区显示，对照文件检查微程序和机器程序是否正确，如果不正确，则说明写入操作失败，应重新写入，可以通过联机软件单独修改某个单元的指令，以修改微指令为例，先用鼠标左键单击指令区的‘微存’TAB 按钮，然后再单击需修改单元的数据，此时该单元变为编辑框，输入 6 位数据并回车，编辑框消失，并以红色显示写入的数据。

3. 运行程序

方法一：本机运行

将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

将时序与操作台单元的开关 KK2 置为‘单步’档，每按动一次 ST 按钮，即可单步运行一条微指令，对照微程序流程图，观察微地址显示灯是否和流程一致。每运行完一条微指令，观测一次数据总线和地址总线，对照数据通路图，分析总线上的数据是否正确。

当模型机执行完 OUT 指令后，检查 OUT 单元显示的数是否正确，按下 CON 单元的总清按钮 CLR，改变 IN 单元的值，再次执行机器程序，从 OUT 单元显示的数判别程序执行是否正确。

方法二：联机运行（软件使用说明请看附录 1）

在联机软件正常运行的情况下，选择菜单命令“【实验】—【CISC 模型机】”，打开 CISC 模型机数据通路图，选择相应的功能命令，就可联机运行、调试模型机的实验程序。

按动 CON 单元的总清按钮 CLR，然后在 PC 上运行模型机实验程序，当模型机执行完 OUT 指令后，检查 OUT 单元显示的数是否正确。我们可以在 PC 微机上以数据通路图或微程序流程图调试方式来观测指令的执行过程，也可以同时观测实验系统中的地址总线、数据总线以及微

指令显示情况。

### 2.2.5 性能评测

1．指令系统庞大，寻址方式也比较复杂，导致指令格式不规整，控制器的译码和执行的硬件设计复杂，不利于超大规模集成电路实现，同时降低了系统的可靠性。

2．指令操作复杂，导致指令执行效率低下，有的指令甚至低于用几条简单的指令的组合实现相同功能所耗费的时间。

3．指令系统庞大，使高级语言编译程序选择目标指令的范围很大，不利于编译的优化；同时庞大的指令系统使各种指令的使用频率都不高，加重设计人员的负担，降低系统的性价比。

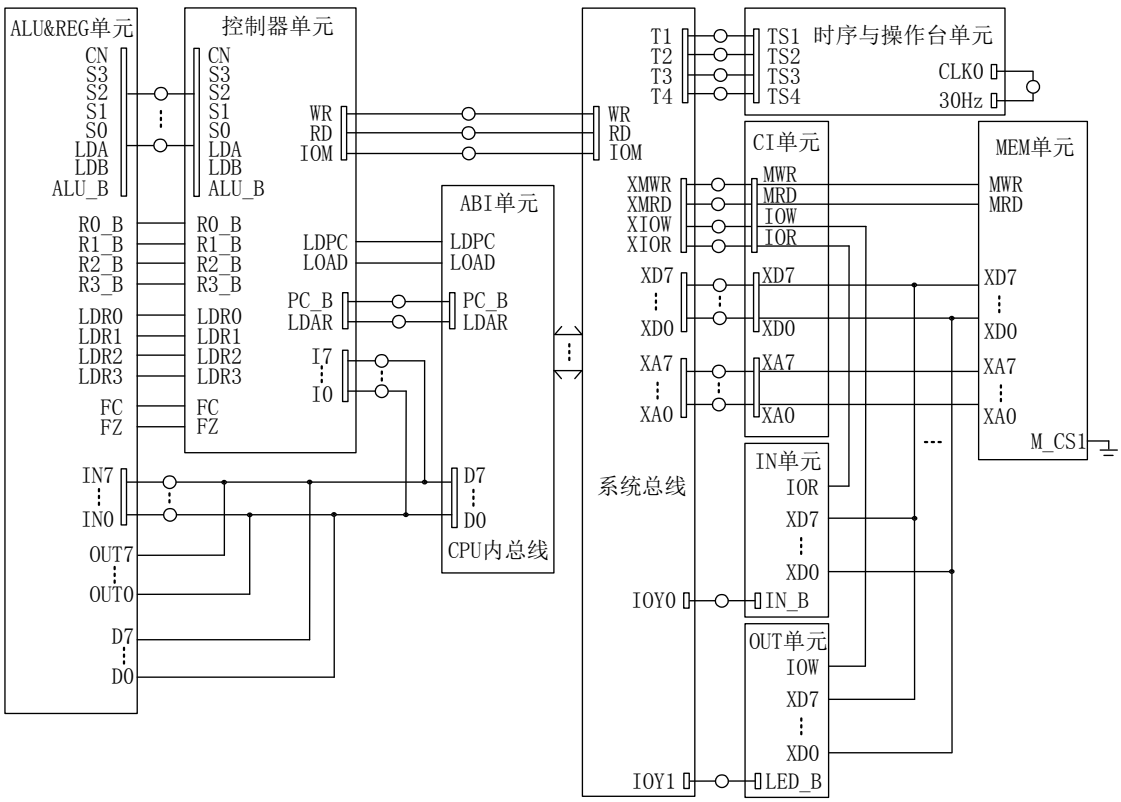



**图 2-2-6 实验接线图**

# 2.3 基于 RISC 技术的模型机设计实验

## 2.3.1 实验目的

1． 了解精简指令系统计算机（RISC）和复杂指令系统计算机（CISC）的系统结构特点和区别。

2． 掌握 RISC 处理器的指令系统特征和一般设计原则。

## 2.3.2 实验设备

PC 机一台， TDX-CMX 实验系统一套。

## 2.3.3 实验原理

1．指令系统设计

本实验采用 RISC 思想设计的模型机选用常用的八条指令：MOV、ADD、NOT、AND、OR、 LOAD、SAVE 和 JMP 作为指令系统，寻址方式采用寄存器寻址及直接寻址两种方式。指令格式采用单字节及双字节两种格式：

单字节指令（MOV、ADD、NOT、AND、OR、JMP）格式如下：

| 7 6 5 4 | 3 2 | 1 0 |
|---|---|---|
| OP-CODE | RS | RD |

其中，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，并规定：

| RS 或 RD | 选定的寄存器 |
|---|---|
| 00 | R0 |
| 01 | R1 |
| 10 | R2 |
| 11 | R3 |

双字节指令（LOAD、SAVE）格式如下：

| 7 6 5 4 （1） | 3 2 （1） | 1 0 （1） | 7—0 （2） |
|---|---|---|---|
| OP-CODE | RS | RD | P |

其中括号中的 1 表示指令的第一字节，2 表示指令的第二字节，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，P 为操作数地址，占用一个字节。

根据上述指令格式，表 2-3-1 列出了本模型机的八条机器指令的具体格式、汇编符号和指令功能：

其中 LOAD 和 SAVE 指令中的 M 位用来判断操作的对象，当 M=0 时 LOAD 和 SAVE 指令是对 IO 进行操作，当 M=1 时 LOAD 和 SAVE 指令是对存储器进行操作。

**表 2-3-1 指令描述**

| 助记符号 | 指令格式 | 指令功能 |
|---|---|---|
| MOV RS RD | 0000 \| RS \| RD | RS → RD |
| ADD RS RD | 0001 \| RS \| RD | RD + RS → RD |
| NOT RD | 0010 \| ** \| RD | /RD → RD |
| AND RS RD | 0011 \| RS \| RD | RD∧RS → RD |
| OR RS RD | 0100 \| RS \| RD | RD∨RS → RD |
| JMP | 0111 \| RS \| | RS → PC |
| LOAD RD | 0101 \| M \| * \| RD \| P | [P] → RD |
| SAVE RS | 0110 \| RS \| M \| * \| P | RS → [P] |

系统采用外设和主存储器各自独立编码的编址方式，I/O 译码单元由采用地址总线高两位作二四译码来实现，原理图如图 2-3-1 所示。

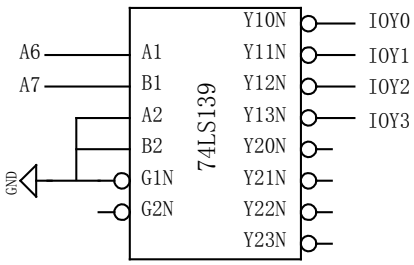



**图 2-3-1 I/O 地址译码原理图**

由于用的是地址总线的高两位进行译码，I/O 地址空间被分为四个区，如表 2-3-2 所示：

**表 2-3-2 I/O 地址空间分配**

| A7 A6 | 选定 | 地址空间 |
|---|---|---|
| 00 | IOY0 | 00-3F |
| 01 | IOY1 | 40-7F |
| 10 | IOY2 | 80-BF |
| 11 | IOY3 | C0-FF |

2．RISC 处理器的模型计算机系统设计

本处理器的时钟及节拍电位如图 2-3-2 所示，数据通路图如图 2-3-3 所示，是采用双总线结构来构建 RISC 处理器的，其指令周期流程图可设计如图 2-3-4 所示，在通路中除控制器由本实

验系统的“控制器单元”中的 FPGA 来设计实现外，其它单元全是由这里实验系统上的单元电路来实现的。

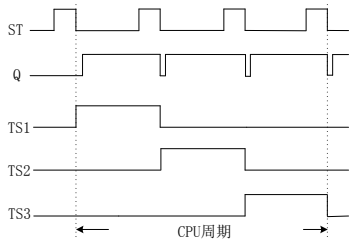



**图 2-3-2 时序电路图**

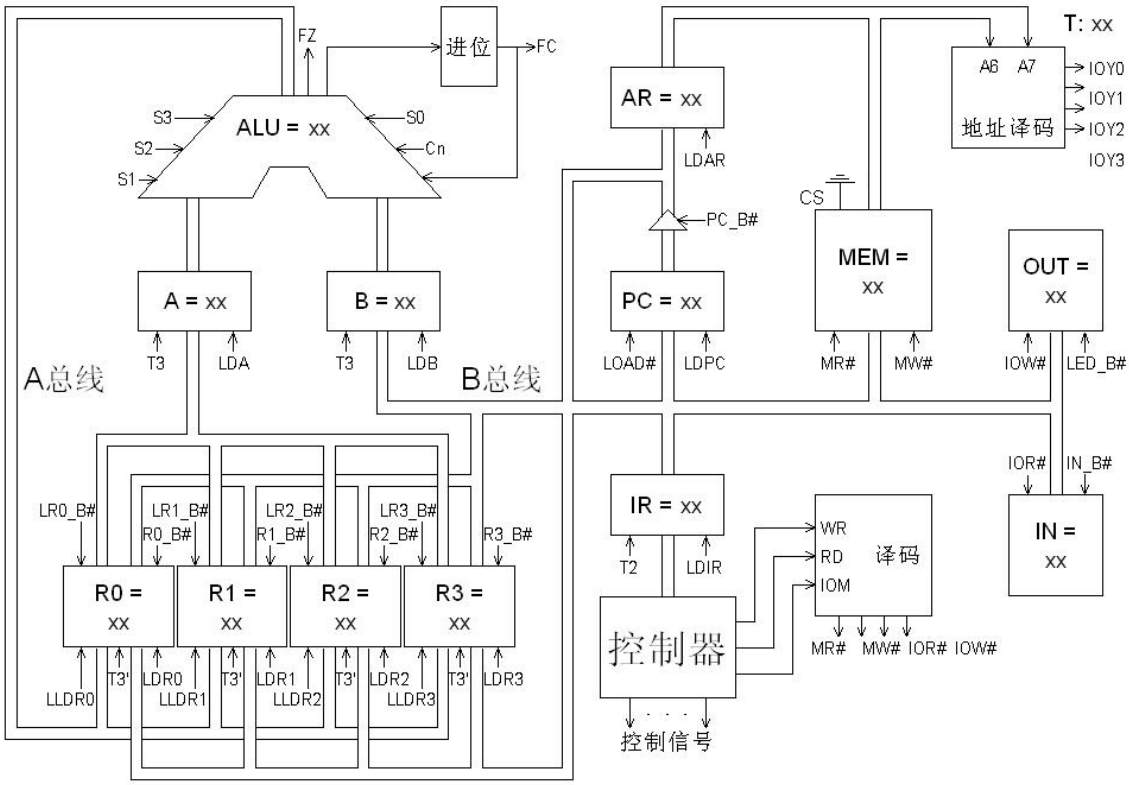



**图 2-3-3 数据通路图**

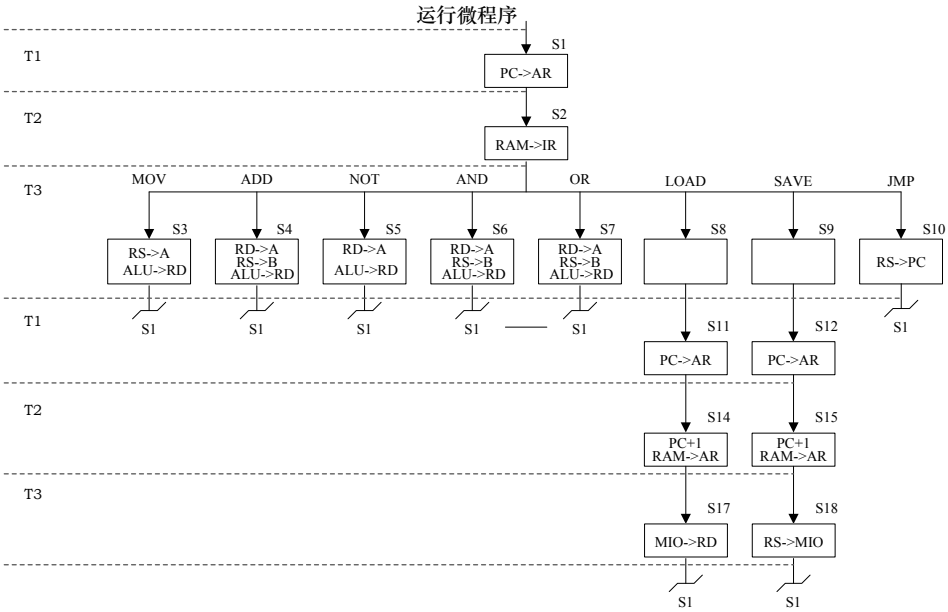



**图 2-3-4 指令周期流程图**

3． 控制器设计

（1）数据通路图中的控制器部分需要在“控制器单元”的 FPGA 中设计。

（2）用 VHDL 语言设计 RISC 子模块的功能描述程序，顶层原理图如图 2-3-5：

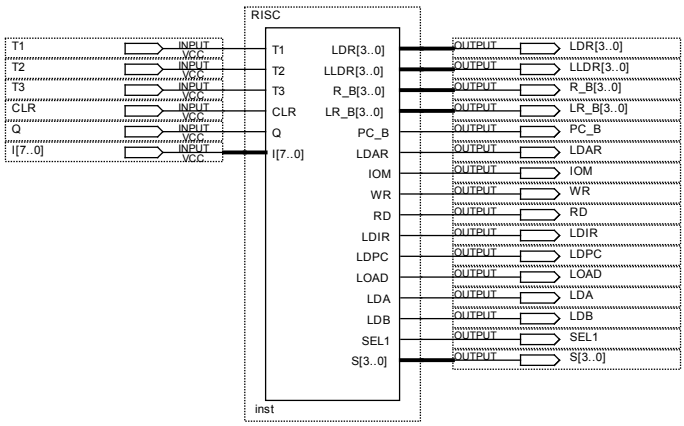



**图 2-3-5 顶层模块图**

## 2.3.4 实验步骤

1.）本实验在“控制器单元”的 FPGA 中编辑、编译所设计的程序，其引脚电路图如图 2-3-6 所示。

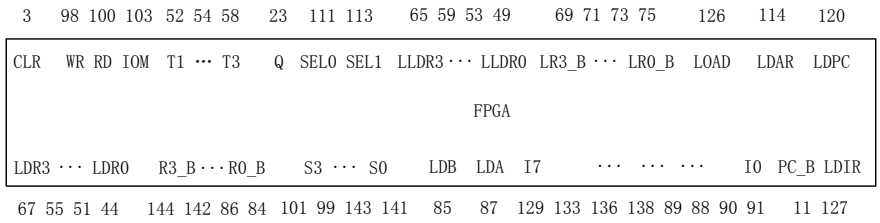



**图 2-3-6 引脚电路图**

2．关闭实验系统电源，把时序与操作台单元的“MODE”短路块拔开，使系统工作在三节拍模式，JP1，JP2 短路块改为 2、3 短接。

3．打开实验系统电源，将下载电缆插入控制器单元的 C_JTAG 口，把生成的 SOF 文件下载到控制器单元中去。

4．编写一段机器指令

| 地址（H） | 内容（H） | 助记符 | 说明 |
|---|---|---|---|
| 00 | 50 | LOAD | IN—>R0 |
| 01 | 40 | | |
| 02 | 51 | LOAD | IN—>R1 |
| 03 | 40 | | |
| 04 | 06 | MOV | R1—>R2 |
| 05 | 18 | ADD | R0+R2—>R0 |
| 06 | 60 | SAVE | R0—>OUT |
| 07 | 80 | | |
| 08 | 58 | LOAD | [10]—>R0 |
| 09 | 10 | | |
| 0A | 70 | JMP | R0—>PC |

5．联上 PC 机，运行 TDX-CMX 联机软件，将上述程序写入相应的地址单元中或用“【转储】—【装载】”功能将该实验对应的文件载入实验系统上的模型机中。

6．将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

在 IN 输入单元上置一数据，将时序与操作台单元的开关 KK2 置为‘单拍’档，每按动一次 ST 按钮，对照数据通路图，分析数据和控制信号是否正确。

当模型机执行完 JMP 指令后，检查 OUT 单元显示的数是否正确，按下 CON 单元的总清按钮 CLR，改变 IN 单元的值，再次执行机器程序，根据 OUT 单元显示的数可判别程序执行是否正确。

7． 联机运行程序时，进入软件界面，装载机器指令后，选择“【实验】－【RISC 模型机】”功能菜单打开相应动态数据通路图，按相应功能键即可联机运行、监控、调试程序。

### 2.3.5 性能评测

将此 RISC 处理器和前面的 CISC 模型机实验相比较，明显看出以下优点：

1．由于指令条数相对较少，寻址方式简单，指令格式规整，控制器的译码和执行硬件相对简单，适合超大规模集成电路实现。

2．由于运算器设置了专用数据通路，寄存器堆采用双端口寄存器堆，运算类指令在一个周期内完成。

3．机器执行的速度和效率大大提高。如上面的那段机器指令在本处理器中执行完需 11 个机器周期，而前面基于 CISC 指令系统的模型机实验中，需 36 个机器周期才能完成。

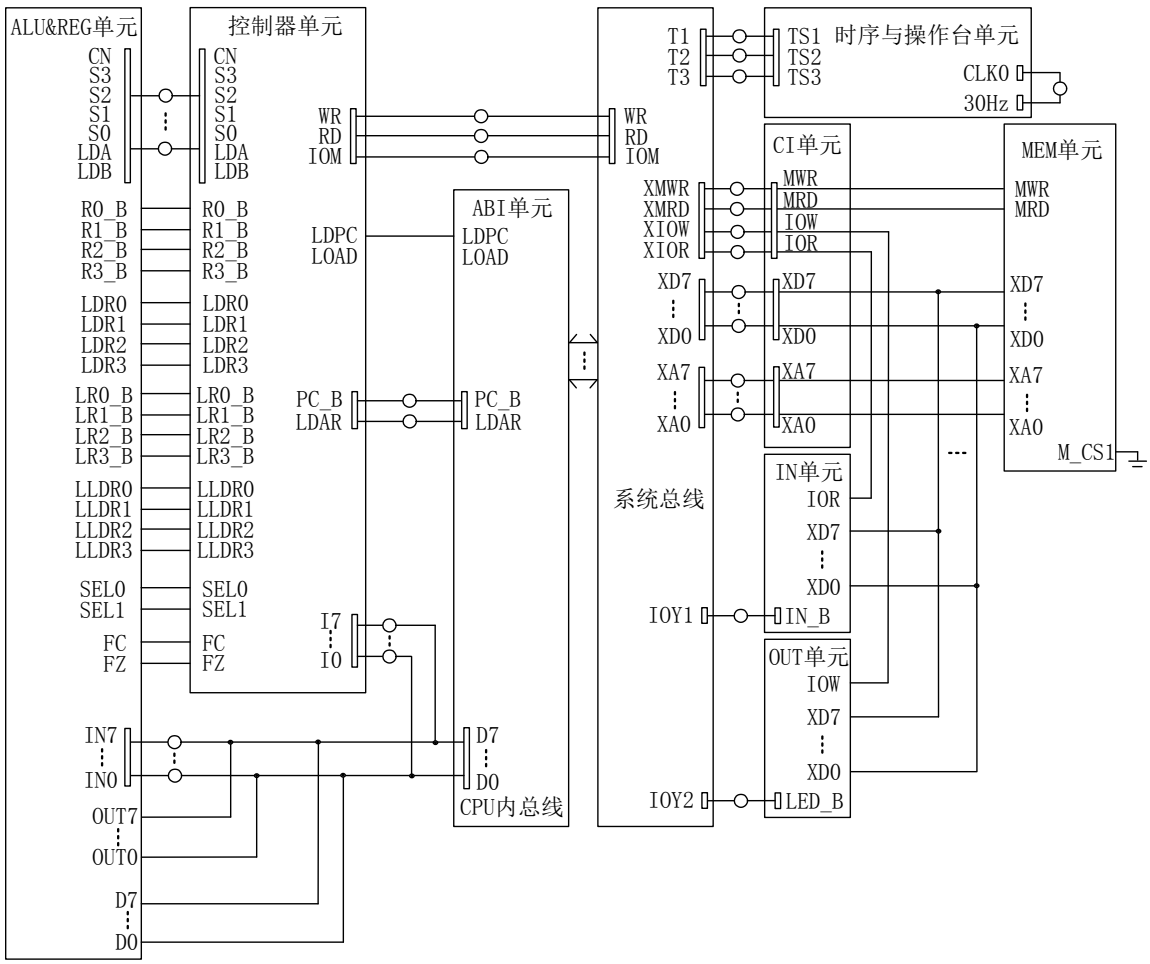



**图 2-3-7 实验接线图**

# 第 3 章　存储系统

存储器是计算机各种信息存储与交换的中心。与传统的以运算器为中心的冯·诺依曼计算机不同，现代的计算机系统以存储器为中心。这里主要从系统结构的角度介绍存储系统的原理，和一些典型的实现方法。

## 3.1　计算机的存储系统

存储器的主要性能指标有速度、容量和价格三个方面。存储系统由两个或两个以上的性能不同的存储器构成，但不是简单的存储器的扩展。存储系统主要研究如何通过软件、硬件或是软件和硬件相结合的方法把多个存储器组合起来，使其等效于一个大的存储器，这个大的存储器的速度接近速度最快的那个存储器、容量接近于容量最大的那个存储器、价格接近于价格最便宜的那个存储器。

如果被操作的数据有着固定的次序，在存储系统中为了简化控制逻辑，通常设有一些特殊的存储器，比如堆栈和队列等。在存取数据时，只需要给出读、写信号就可以了，不需要再给出地址信息。从而简化了操作、提高了数据存取的效率。

高速缓冲存储器（Cache）是插在 CPU 和主存之间的一个快速小容量的存储器，它主要是为解决 CPU 和主存之间的速度匹配问题而设置的。它的速度比主存快，具有和 CPU 相近的速度，有了 Cache，就能高速地向 CPU 提供指令和数据，从而加快了程序的执行速度。它们的关系图如图 3-1-1。

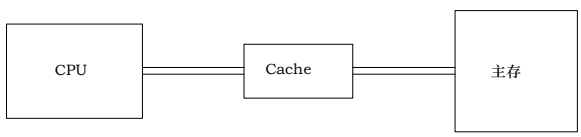



**图 3-1-1　CPU 与 Cache 关系图**

在计算机中增加 Cache 的目的，就是在性能上使主存储器的平均读出时间尽可能的接近于 Cache 的读出时间。要实现这个目的，就是要将 CPU 将要读取的主存单元内容先预先读到 Cache 中。由于程序编制的局部性，即程序总是按顺序编写和编址的，所以指令也往往按地址顺序存放在存储器中；一条指令执行后一般是执行紧接着的那条指令，只有遇到转移指令时才会跳到另一个区域，但跳转之后又会顺序执行；还有就是当执行一个循环程序时，会在一块小的区域内重复循环执行若干指令；对于存储的数据也是如此。因此，按照一定的预读算法，可以实现很高的 Cache 命中率。

当 CPU 要从主存中读取一个字，先把这个字的地址传给 Cache，检查这个字是否在 Cache 中，如果在，就把这个字直接从 Cache 送到 CPU 中，如果不在，则把这个地址传给主存，把这个字附近地址单元的内容以块为单位取到 Cache 中，再从 Cache 中把这个字送往 CPU 这样就有较大的可能把 CPU 下次访问的数据预取到 Cache 中。

为把信息读取到 Cache 中，必须应用某种函数把主存地址映象到 Cache 中定位，称为地址映象。这些函数通常称为映象函数。在信息按这种映象关系装入到 Cache 后，执行程序时，将

主存地址变换成 Cache 地址，这个变换过程称为地址变换。

当要将新的主存中的一块数据装入到 Cache 替换掉原存储的一块数据，必须要有相应的替换算法。常用的替换策略有：

最近最少使用（LRU）策略，即替换掉那些在 Cache 中驻留时间最长且未被引用的块。

先进先出（FIFO）策略，即替换掉那些在 Cache 中停留时间最长的块。

近期最少使用（LFU）策略，即替换掉最近引用次数最少的块。

# 3.2 FIFO 先进先出存储器实验

## 3.2.1 实验目的

掌握 FIFO 存储器的工作特性和读写方法。

## 3.2.2 实验设备

PC 机一台，TDX-CMX 实验系统一套。

## 3.2.3 实验原理

本实验在“扩展单元”上进行，扩展单元由两大部分组成，一是 LED 显示灯，两组 16 只，供调试时观测数据，LED 灯为正逻辑，1 时亮，0 时灭。另外是一片 Intel 公司的 FPGA 芯片及其外围电路。

本实验用 FPGA 芯片来实现一个简单的 8 位×4 的 FIFO，器件的接口信号如图 3-2-2，内部逻辑图如下图 3-2-3。

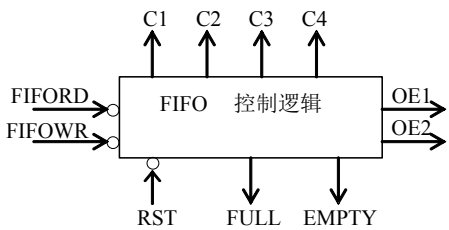



**图 3-2-2 定义 FIFO 器件的接口信号**

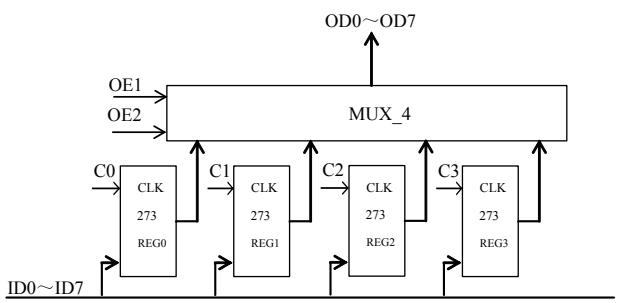



**图 3-2-3 FIFO 内部逻辑图**

其各信号的功能为：

EMPTY：FIFO 存储器空标志，高电平有效。

FULL：FIFO 存储器满标志，高电平有效。

$\overline{RST}$：清 FIFO 存储器为空。

$\overline{FIFOWR}$：FIFO 存储器写入信号，低电平有效。

$\overline{FIFORD}$：FIFO 存储器读信号，低电平有效。

ID0～ID7：FIFO 存储器输入数据线。

OD0～OD7：FIFO 存储器输出数据线。

根据图 3-2-3 所示的内部逻辑图设计的顶层原理图如下：

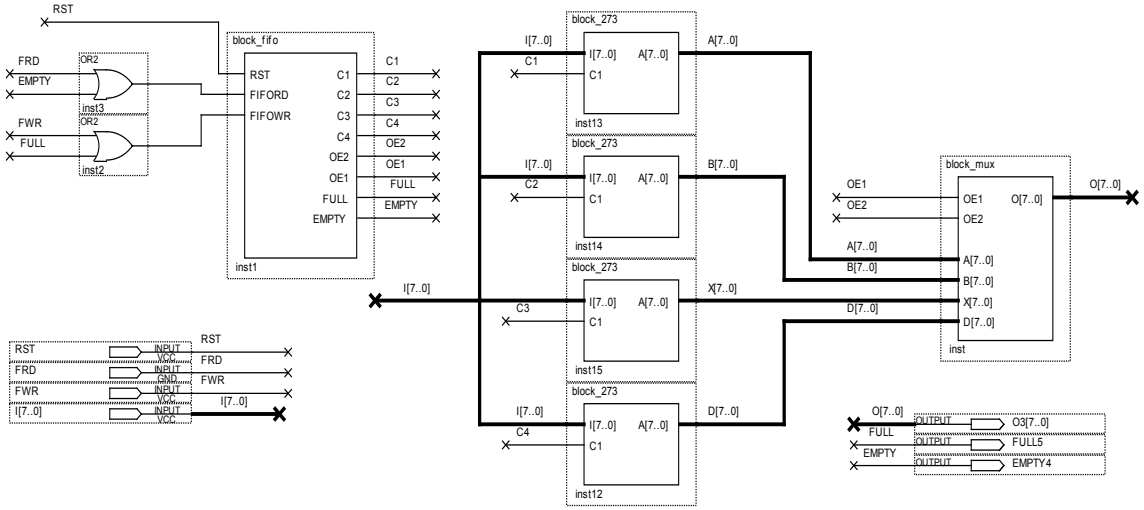



**图 3-2-4 FIFO 顶层原理图**

## 3.2.4 实验步骤

（1）本实验在“扩展单元”的 FPGA 中完成，按照上述功能要求及管脚说明，进行 FPGA 芯片设计，其引脚电路图如图 3-2-5 所示。

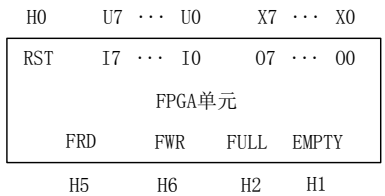



**图 3-2-5 引脚电路图**

（2）关闭电源，按图 3-2-6 实验连线图接线。确保接线正确后打开实验系统的电源。

（3）编辑、编译所设计的程序，打开实验系统电源，将下载电缆插入扩展单元的 E_JTAG 口，把生成的 SOF 文件下载到 FPGA 单元中去。

（4） 接线图中 H1 是 FIFO 空状态、H2 是满状态指示信号，分别通过扩展单元指示灯 H1、

H2 显示，用来反映 FIFO 当前的状态。注意：系统总清后 FIFO 输出的数据是无效的，因为当 FIFO 总清后，读计数器的输出被清零，此时多路开关选择输出 C0 中的数据，而 C0 中的数据是不确定的。当第一次对 FIFO 进行写操作后，FIFO 输出的数据开始有效。简单的说，空标志位无效时，FIFO 的输出有效。每读一次，FIFO 的输出改变一次，指向下一个数据。当 FIFO 满标志有效时，不允许再对 FIFO 进行写操作，否则会引起系统错误。

实验时，按动系统右下脚的 CLR 清零开关可使读、写信号计数清零。这时指示灯 H1 亮，表示 FIFO 为空。使用 CON 单元编号为 SD07 到 SD00 的开关模拟输入总线给出一个数据，按动时序与操作台单元的开关 ST，可将该数写入到 FIFO 中。这时指示灯 H1 灭，表示 FIFO 中已经有数据存在，说明当前 FIFO 的输出是有效的；依次写四次后，FULL 满标志置位，这时指示灯 H2 亮；然后连续按动开关 KK，给出读信号，将顺序读出所存的四个数，扩展单元的 L15 到 L8 显示所读出的数据，四个数全部读出后，EMPTY 空标志置位。检查执行结果是否与理论值一致。

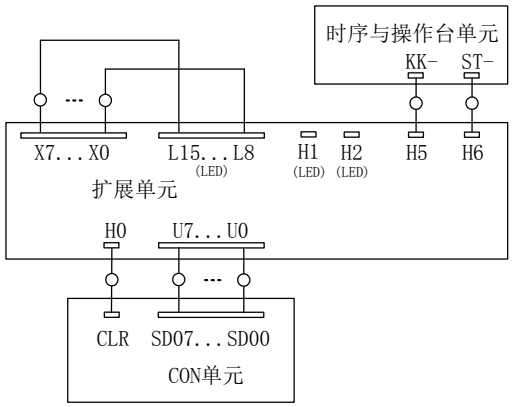



**图 3-2-6 FIFO 实验接线图**

# 3.3 CACHE 控制器设计实验

## 3.3.1 实验目的

掌握 Cache 控制器的原理及其设计方法。

## 3.3.2 实验设备

PC 机一台，TDX-CMX 实验系统一套。

## 3.3.3 实验原理

本实验采用的地址变换是直接映象方式，这种变换方式简单而直接，硬件实现很简单，访问速度也比较快，但是块的冲突率比较高。其主要原则是：主存中一块只能映象到 Cache 的一个特定的块中。

假设主存的块号为 B，Cache 的块号为 b，则它们之间的映象关系可以表示为：

$$b = B \bmod C_b$$

其中，$C_b$ 是 Cache 的块容量。设主存的块容量为 $M_b$，区容量为 $M_e$，则直接映象方法的关系如图 3-3-1 所示。把主存按 Cache 的大小分成区，一般主存容量为 Cache 容量的整数倍，主存每一个分区内的块数与 Cache 的总块数相等。直接映象方式只能把主存各个区中相对块号相同的那些块映象到 Cache 中同一块号的那个特定块中。例如，主存的块 0 只能映象到 Cache 的块 0 中，主存的块 1 只能映象到 Cache 的块 1 中，同样，主存区 1 中的块 $C_b$（在区 1 中的相对块号是 0）

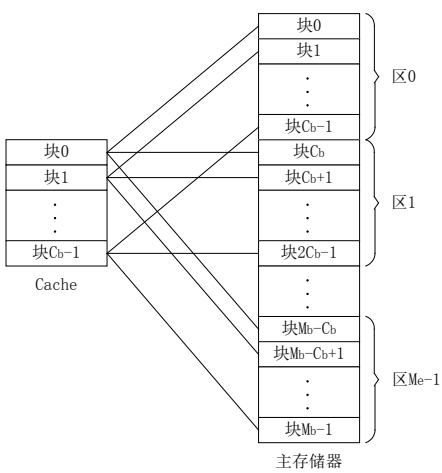



**图 3-3-1 直接相联映象方式**

也只能映象到 Cache 的块 0 中。根据上面给出的地址映象规则，整个 Cache 地址与主存地址的低位部分是完全相同的。

直接映象方式的地址变换过程如图 3-3-2 所示，主存地址中的块号 B 与 Cache 地址中的块

号 b 是完全相同的。同样，主存地址中的块内地址 W 与 Cache 地址中的块内地址 w 也是完全相同的，主存地址比 Cache 地址长出来的部分称为区号 E。

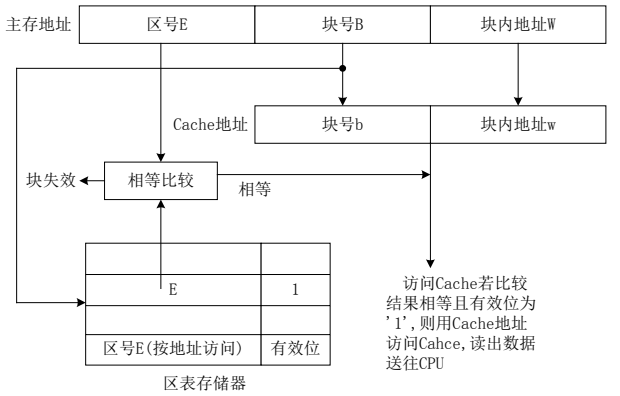



**图 3-3-2 直接相联地址变换**

在程序执行过程中，当要访问 Cache 时，为了实现主存块号到 Cache 块号的变换，需要有一个存放主存区号的小容量存储器，这个存储器的容量与 Cache 的块数相等，字长为主存地址中区号 E 的长度，另外再加一个有效位。

在主存地址到 Cache 地址的变换过程中，首先用主存地址中的块号去访问区号存储器（按地址访问）。把读出来的区号与主存地址中的区号 E 进行比较，根据比较结果和与区号在同一存储字中的有效位情况作出处理。如果区号比较结果相等，有效位为‘1’，则 Cache 命中，表示要访问的那一块已经装入到 Cache 中了，这时 Cache 地址（与主存地址的低位部分完全相同）是正确的。用这个 Cache 地址去访问 Cache，把读出来的数据送往 CPU。其他情况均为 Cache 没有命中，或称为 Cache 失效，表示要访问的那个块还没有装入到 Cache 中，这时，要用主存地址去访问主存储器，先把该地址所在的块读到 Cache 中，然后 CPU 从 Cache 中读取该地址中的数据。

本实验要在 FPGA 中实现 Cache 及其地址变换逻辑（也叫 Cache 控制器），采用直接相联地址变换，只考虑 CPU 从 Cache 读数据，不考虑 CPU 从主存中读数据和写回数据的情况，Cache 和 CPU 以及存储器的关系如图 3-3-3 所示。

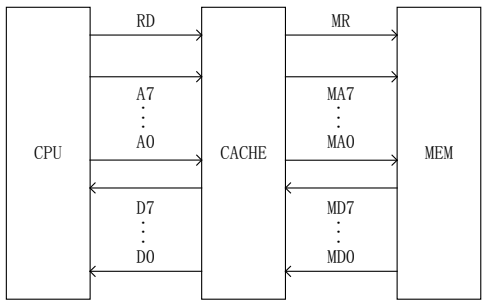



**图 3-3-3 Cache 系统图**

Cache 控制器顶层模块如图 3-3-4 所示，主存地址为 A7…A0,共 8 位，区号 E 取 3 位，这样 Cache 地址还剩 5 位，所以 Cache 容量为 32 个单元，块号 B 取 3 位，那么 Cache 分为 8 块，块

内地址 W 取 2 位，则每块为 4 个单元。图 3-3-4 中，WCT 为写 Cache 块表信号，CLR 为系统总清零信号，A7…A0 为 CPU 访问内存的地址，M 为 Cache 失效信号，CA4…CA0 为 Cache 地址，MD7…MD0 为主存送 Cache 的数据，D7…D0 为 Cache 送 CPU 数据，T2 为系统时钟，RD 为 CPU 访问内存读信号，LA1 和 LA0 为块内地址。

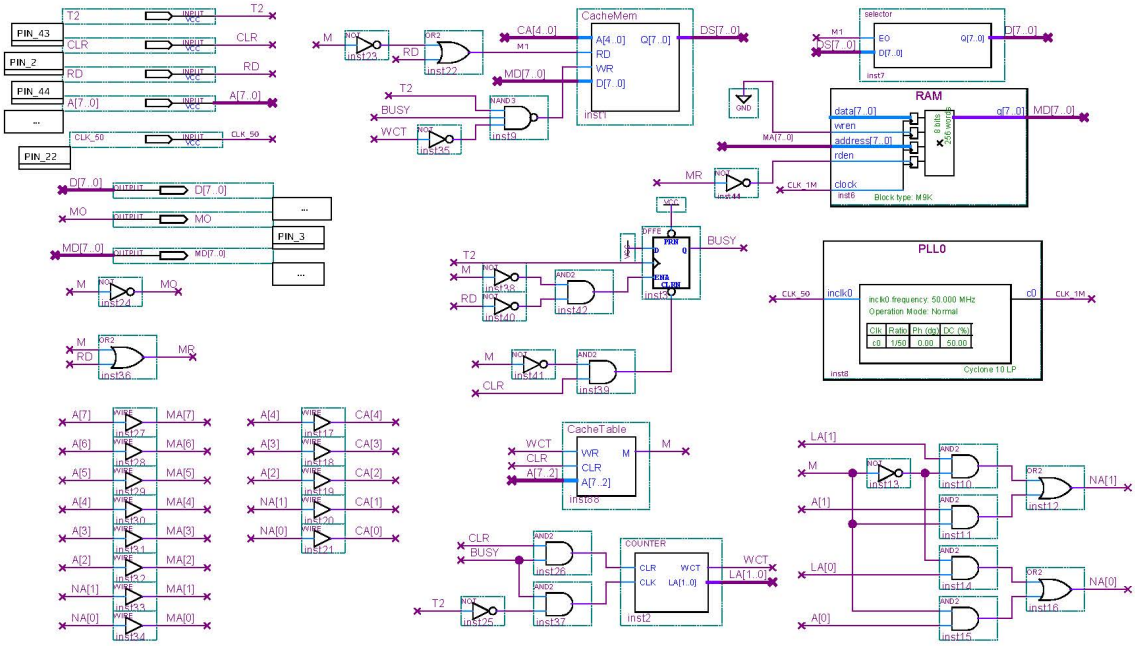



**图 3-3-4　Cache 控制器顶层模块图**

在 Quartus 软件中先调用一个 8 位的 SRAM 的 IP 核，编写一个 MIF 文件用来存储数据，然后实现一个 8 位的存储单元（见例程中的 MemCell.bdf），然后用这个 8 位的存储单元来构成一个 32 X 8 位的 Cache（见例程中的 CacheMem.bdf）,这样就实现了 Cache 的存储体。

再实现一个 4 位的存储单元（见例程中的 TableCell.bdf），然后用这个 4 位的存储单元来构成一个 8 X 4 位的区表存储器，用来存放区号和有效位（见例程中的 CacheTable.bdf）,在这个文件中，还实现了一个区号比较器，如果主存地址的区号 E 和区表中相应单元中的区号相等，且有效位为 1，则 Cache 命中，否则 Cache 失效，标志为 M，M 为 0 时表示 Cache 失效。

当 Cache 命中时，就将 Cache 存储体中相应单元的数据送往 CPU，这个过程比较简单。当 Cache 失效时，就将主存中相应块中的数据读出写入 Cache 中，这样 Cache 控制器就要产生访问主存储器的地址和主存储器的读信号，由于每块占四个单元，所以需要连续访问四次主存，这就需要一个低地址发生器，即一个 2 位计数器（见例程中的 Counter.vhd），将低 2 位和 CPU 给出的高 6 位地址组合起来，形成访问主存储器的地址。M 就可以做为主存的读信号，这样，在时钟的控制下，就可以将主存中相应的块写入到 Cache 的相应块中，最后再修改区表（见例程中的（CacheCtrl.bdf）。

### 3.3.4 实验步骤

(1) 使用 Quartus 软件编辑实现相应的逻辑并进行编译，直到编译通过，Cache 控制器在 FPGA 芯片中对应的引脚如图 2-2-5 所示，框外文字表示连接标号，框内文字表示该引脚的含义（本实验例程见‘安装路径\FPGA\CacheCtrl\CacheCtrl.qpf’工程）。

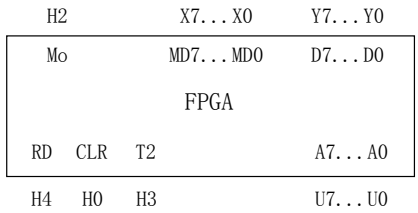



**图 3-3-5 引脚分配图**

(2) 关闭实验系统电源，按图 3-3-6 连接实验电路，并检查无误，图中将用户需要连接的信号用圆圈标明。

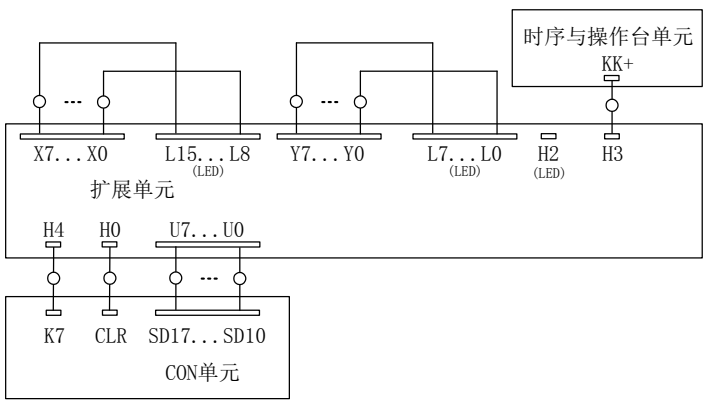



**图 3-3-6 实验接线图**

(3) 打开实验系统电源，将下载电缆插入扩展单元的 E_JTAG 口，把生成的 SOF 文件下载到扩展单元中的 FPGA 去。

(4) 将时序与操作台单元的开关 KK3 置为‘运行’档，CLR 信号由 CON 单元的 CLR 模拟给出，按动 CON 单元的 CLR 按钮，清空区表。

(5) 预先往主存写入数据：存储器已经提前装载好了数据文件（RAM.mif），用户也可以自己改写内容。

(6) CPU 访问主存地址由 CON 单元的 SD17…SD10 模拟给出，如 00000001。CPU 访问主存的读信号由 CON 单元的 K7 模拟给出，置 K7 为低，可以观察到扩展单元上的 H2 指示灯亮，L7…L0 指示灯灭，表示 Cache 失效。此时按动 KK 按钮四次，注意 L15…L8 指示灯的变化情况，地址会依次加一，L15…L8 指示灯上显示的是当前主存数据，按动四次 KK 按钮后，H2 指示灯变灭，L7…L0 上显示的值即为 Cache 送往 CPU 的数据。

(7) 重新给出主存访问地址，如 00000011，H2 指示灯变灭，表示 Cache 命中，说明第 0 块

数据已写入 Cache。

(8) 重新给出大于 03H 地址，体会 Cache 控制器的工作过程。

# 第 4 章　时间并行性为特征的计算机系统

加快机器语言的解释是计算机设计的基本任务。在计算机组成原理中研究的模型机多采用顺序执行方式，在这种执行方式中，取指令和执行指令顺序进行。因而在取指令时执行部件空闲，执行时取指令部件空闲，不利于提高计算机的运行效率。为了提高计算机的效率，让各功能部件在时间上并行工作，提高计算机系统的时间并行性，我们需要引入重叠和流水的相关概念。

## 4.1　重叠和流水

### 4.1.1　重叠的基本思想及实现

一条指令的执行过程可以分为多个阶段，一般可以把它们归并为取指令、分析和执行三个阶段。其中，“取指令”就是按指令计数器的内容访问主存储器，取出一条指令送到指令寄存器。“分析”是指对指令的操作码进行译码，按照给定的寻址方式和地址字段中的内容形成操作数的地址，并用这个地址读取操作数，操作数可能在主存储器中，也可能在寄存器中。“执行”是根据操作码的要求，完成指令规定的功能，在此期间，要把运算结果写到寄存器或主存储器中。下面为了便于分析，把指令的执行过程分为两个步骤：取指令、分析和执行。当多条指令要在处理机中执行时，一般有两种执行方式。

1．顺序执行方式。指各条指令之间顺序串行的执行，执行完一条指令后才取出下一条指令来执行，而且每条指令内部的各个微操作也是顺序串行地执行。

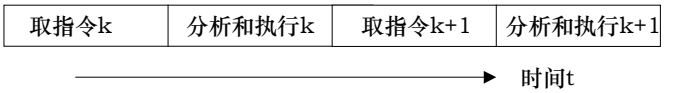



**图 4-1-1　指令的顺序执行方式**

采用顺序控制方式的优点是控制简单，节省设备。但主要缺点有两个，一个是执行指令速度慢，只有当上一条指令完全执行完后才能开始下一条指令的执行。另一个是功能部件的利用率低。例如在取指和取操作数期间，主存储器是忙碌的，但是运算器却是空闲着；而在对操作数执行运算期间，运算器是忙碌的，主存却空闲着。

2．重叠执行方式。指在解释第 k 条指令的操作完成之前，就可开始解释第 k+1 条指令。如图 4-1-2 所示。

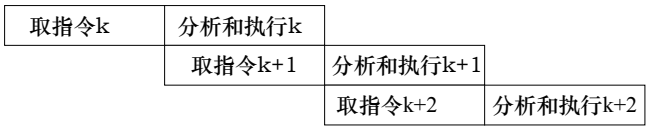



**图 4-1-2　重叠执行方式**

采用重叠控制方式的主要优点有两个，一个是缩短了程序的执行时间，另一个是功能部件的利用率明显提高。主存储器基本上处于忙碌状态。缺点是需要增加一些硬件，控制过程也复

杂一些。

假设取指令操作与分析和执行操作所用的时间相等，都是Δt，则执行 n 条指令若采用顺序执行方案所用的时间为：

$$T=2n\Delta t$$

若改用重叠方案来实现，所用的时间为：

$$T=n\Delta t$$

### 4.1.2 流水线技术

流水可以看作是重叠思想的进一步延伸，指令的执行过程可以划分为指令预取、指令分析和指令执行三个部分，这三个子过程分别在指令预取部件、执行分析部件和指令执行部件中完成。如图 4-1-3 所示。由于在各功能段输出端各有一个锁存器，可以分别保存指令分析和指令执行的结果，因此，各部件可以完全独立并行地工作，而不必等一条指令的“分析”、“执行”子过程都完成之后才送入下一条指令。分析部件在完成一条指令“分析”并将结果送入指令执行部件的同时，就可以开始分析下一条指令。

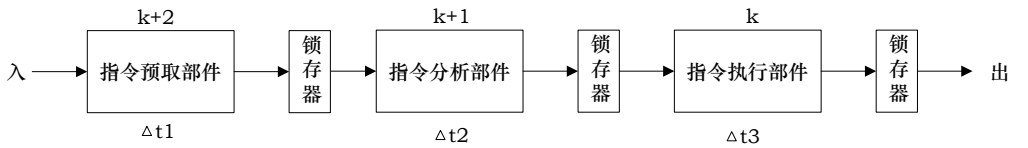



**图 4-1-3 简单的流水**

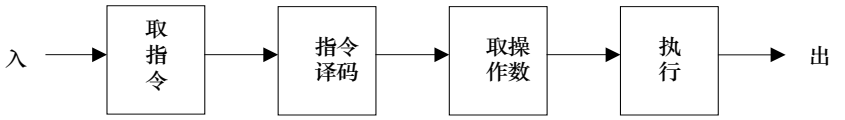
上图中如果指令预取部件预取一条指令所用的时间△t1、指令分析部件分析一条指令所用的时间△t2 与指令执行部件执行一条指令所用的时间△t3 都相等，即△t1=△t2=△t3=△t，就一条指令的解释来看还是需要 3△t，但是从机器的输出来看，每过△t 就有一条指令执行完成。因此，机器执行指令的速度提高了 2 倍。

如果把“分析”子过程再细分成“取指令”、“指令译码”和“取操作数”3 个子过程，并加快“执行”子过程，使 4 个子过程都能独立地工作，且经过的时间都是△t。如图 4-1-4（a）所示，则可以描述出流水的时空图如图 4-1-4（b）。

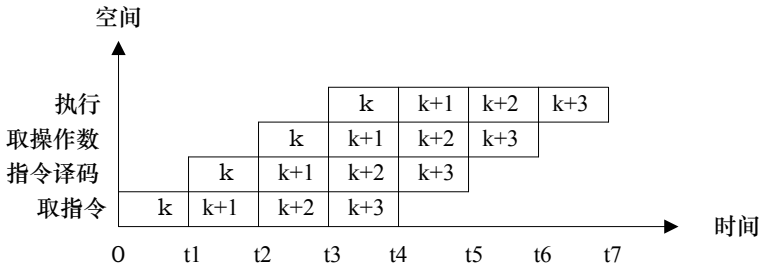
在时空图中，横坐标表示时间，也就是输入到流水线中的各个任务在流水线中所经过的时间。纵坐标表示空间，即流水线的各个子过程。在时空图中，流水线的一个子过程通常成为“功能段”。

从时空图中，可以很清楚的看出各个任务在流水线的各段中的流动的过程。从横坐标方向看，流水线中的各个功能部件逐个连续地完成自己的任务；从纵坐标看，在同一时间段内有多个功能段在同时工作。

在上面的流水线中，对于“取指令”、“指令译码”、“取操作数”、“执行”每个子过程都需要△t 时间完成，这样，虽然完成一条指令所需的时间还是一个 T，但是每隔一个△t（T/4）时间就会一条指令结果输出，这样的执行效率比顺序方式提高了 3 倍。




（a） 指令流水线




（b） 流水处理的时空图

图 4-1-4 流水处理

### 4.1.3 流水线的特点

采用流水线方式的处理机与传统的顺序执行方式相比，具有如下特点：

1．流水线中处理的必须是连续的任务，只有连续不断地提供任务才能发挥流水线的效率。流水线从开始启动到流出第一个结果需要一个“装入时间”,在这段时期内并没有流出任何结果，所以，对第一条指令来说，和顺序执行没有区别。

2．在流水线每个功能部件的后面都要有一个缓冲寄存器，用于保存本段的执行结果，以保证各部件之间速度匹配及各部件独立并行的运行。

3．流水线是把一个大的功能部件分解为多个独立的功能部件，并依靠多个功能部件并行工作来缩短程序执行时间。流水线中各段的执行时间应尽量相等，否则将引起“堵塞”、“断流”等不利状态。执行时间最长的一段将成为整个流水线的“瓶颈”，在流水线中解决“瓶颈”段的执行时间会使流水线的实际效率有较大的提高。

### 4.1.4 相关处理

所谓相关是指在一段程序的相近指令之间有某种关系,这种关系可能影响指令的重叠执行。通常，把相关分为两大类，一类是数据相关，另一类是控制相关。数据相关主要有四种，分别是指令相关、主存操作数相关、通用寄存器相关和变址相关。解决数据相关的方法通常有两种，一种是推后分析法，在遇到数据相关时，推后本条指令的分析，直至所需要的数据写入到相关的存储单元中；另一种方法是设置专用通路，即不必等所需要的数据写入到相关的存储单元中，而是经专门设置的数据通路读取所需要的数据。

设置专用通路解决数据相关的常用方法也有两种。一种是采用 D 型触发器构成通用寄存器，并且在通用寄存器和运算器之间不设置缓冲寄存器或锁存器以避免数据相关，因为在这种结构中，允许在同一节拍中实现寄存器之间的循环传送，如图 4-1-5 所示。由于在执行的同时已经将执行结果写回，因而避免了数据相关的发生。另一种是在运算器和通用寄存器堆之间增

设专用数据通路。在许多处理机中，由于把运算结果写到寄存器堆中需要的时间比较长，运算与结果写回必须分为两个时钟周期，因此，在如图 4-1-6 所示的数据通路中间需要加入锁存器或数据缓冲器。

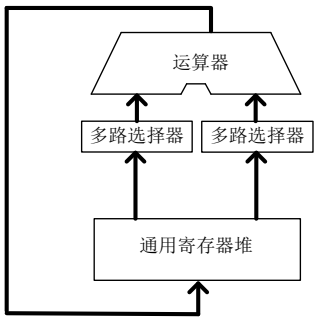



**图 4-1-5 一种典型的运算器结构**

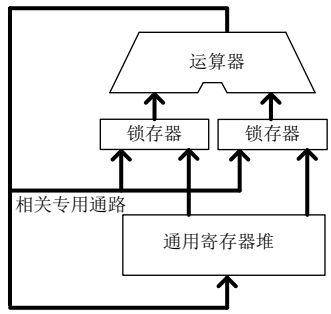



**图 4-1-6 设置数据相关专用通路的运算器结构**

如图 4-1-6 所示，在运算器的输出端到锁存器的输入端之间建立一条专用的数据通路。当发生数据相关时，运算结果在送入寄存器堆的同时，还需要通过这条专用通路把运算结果送入到运算器的锁存器中。这样，下条指令就不必推后执行，就像没有数据相关时一样。

控制相关是指因为程序的执行方向可能改变而引起的相关。可能改变程序执行方向的指令通常有无条件转移、一般条件转移、子程序调用、中断等。下面分别介绍几种转移指令的处理方法。

1．无条件转移指令

无条件转移指令一般能够在指令分析器中执行完成，形成转移地址送到先行程序计数器 PC1 和 PC 中，指令缓冲栈按照 PC1 的指示重新开始向存储控制器申请取指令。当要转移到的指令不在先行指令缓冲栈中，则要将先行指令缓冲栈中所有预取的指令作废，重新取指令。当要转移的指令在先行指令缓冲栈中时，只要把这条指令之前的预取的指令作废，就可以不用停顿的连续工作。

2．一般条件转移

对于一般的条件转移指令，如果转移不成功，只要等待一个周期，就可以继续“分析和执行”，在先行指令缓冲栈中预取的指令也仍然有效。如果转移成功，且转移的距离比较近，指令已被取到先行指令缓冲栈中。这时，只要作废本指令之前的所有指令，接着进行分析就可以了。如果转移距离比较远，指令不在先行缓冲栈中，则要将指令缓冲栈的指令全部作废，相当于串行的开始取指令，分析指令。

# 4.2 具有指令预取功能的模型机设计实验

## 4.2.1 实验目的

1．在 CISC 模型机的基础上，设计一台具有指令预取功能的模型机。
2．通过具体上机调试来掌握处理机重叠操作的原理。

## 4.2.2 实验设备

PC 机一台， TDX-CMX 实验系统一套。

## 4.2.3 实验原理

1．指令系统设计

本实验设计的模型机指令分为两大类，由于所设计的指令格式中操作码有四位，可以设计十六条不同的指令，我们给出其中常用的八条指令的设计，有兴趣的读者可以通过在此模型机的基础上扩充指令来构建自己的模型机。模型机指令格式如下，其中括号中的 1 表示指令的第一字节，2 表示指令的第二字节，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，P 为操作数地址，占用一个字节。

单字节指令（MOV、ADD、NOT、AND、OR）格式如下：

| 7 6 5 4 | 3 2 | 1 0 |
|---|---|---|
| OP-CODE | RS | RD |

其中，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，规定：

| RS 或 RD | 选定的寄存器 |
|---|---|
| 00 | R0 |
| 01 | R1 |
| 10 | R2 |
| 11 | R3 |

双字节指令（IN、OUT、JMP）格式如下：

| 7 6 5 4 （1） | 3 2 （1） | 1 0 （1） | 7—0 （2） |
|---|---|---|---|
| OP-CODE | RS | RD | P |

根据上述指令格式，表 4-2-1 列出了本模型机的八条机器指令的具体格式、汇编符号和指令功能：

**表 4-2-1 指令描述**

| 助记符号 | 指令格式 | | | | 指令功能 |
|---|---|---|---|---|---|
| MOV RS RD | 0000 | RS | RD | | RS → RD |
| ADD RS RD | 0001 | RS | RD | | RD + RS → RD |
| NOT RD | 0010 | ** | RD | | /RD → RD |
| AND RS RD | 0011 | RS | RD | | RD∧RS → RD |
| OR RS RD | 0100 | RS | RD | | RD∨RS → RD |
| IN RD | 0101 | ** | RD | P | [P] → RD |
| OUT RS | 0110 | RS | ** | P | RS → [P] |
| JMP D | 0111 | ** | ** | P | P → PC |

系统采用外设和主存储器各自独立编码的编址方式，I/O 译码单元由采用地址总线高两位作二四译码来实现，原理图如图 4-2-1 所示。

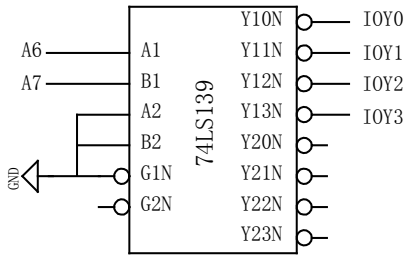



**图 4-2-1 I/O 地址译码原理图**

由于采用地址总线的高两位进行译码，I/O 地址空间被分为四个区，如表 4-2-2 所示：

**表 4-2-2 I/O 地址空间分配**

| A7 A6 | 选定 | 地址空间 |
|---|---|---|
| 00 | IOY0 | 00-3F |
| 01 | IOY1 | 40-7F |
| 10 | IOY2 | 80-BF |
| 11 | IOY3 | C0-FF |

2．有指令预取功能的模型机系统设计

在 CISC 模型机实验过程中，我们已经了解了在微程序控制下可自动产生各部件单元控制信号，实现特定指令的功能。而在本次实验中，引入“指令预取”部件和“总线控制”部件，使指令预取与指令执行的工作重叠进行。

采用重叠方案实现上面指令系统的模型计算机的数据通路框图可设计如图 4-2-2 所示。整体的模型机采用双总线的结构，每个机器周期由三节拍构成。这里，计算机“执行部件”数据通路的控制主要由微程序控制器来完成，而“指令预取”部件的数据通路可以通过设置先进总线接口 ABI 单元来实现。“指令预取”采用四字节的先进先出栈（FIFO）作为指令缓冲栈，在程

序运行过程中，“指令预取”部件将指令从主存储器中取到 FIFO 里，而执行部件则从 FIFO 中取得指令并进行指令的译码，在微程序控制下实现指令的操作。“总线控制”部件根据“执行部件”和“指令预取”部件发出的相应信号来选择总线当前的数据通路，并产生相应的控制信号，以实现对 I/O 设备的读/写操作和指令预取操作。“总线控制”部件的电路可以设计如图 4-2-3，该电路已经在实验系统的“控制器单元”的“指令预取控制部件”中实现。

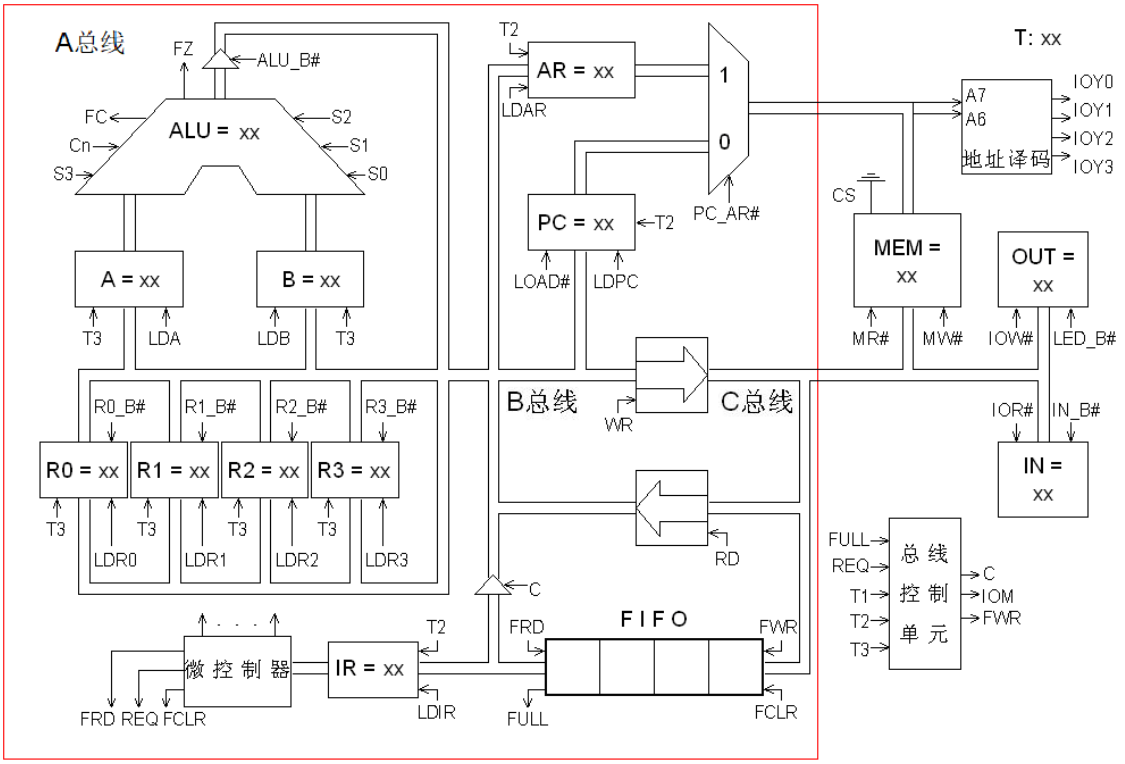



**图 4-2-2　数据通路图**

总线控制部件产生的控制信号有： C、WR、RD、IOM、LDPC、PC_AR 和 FWR、FRD、FCLR，其中信号 C 控制取地址通道，执行双字节指令取地址时有效。

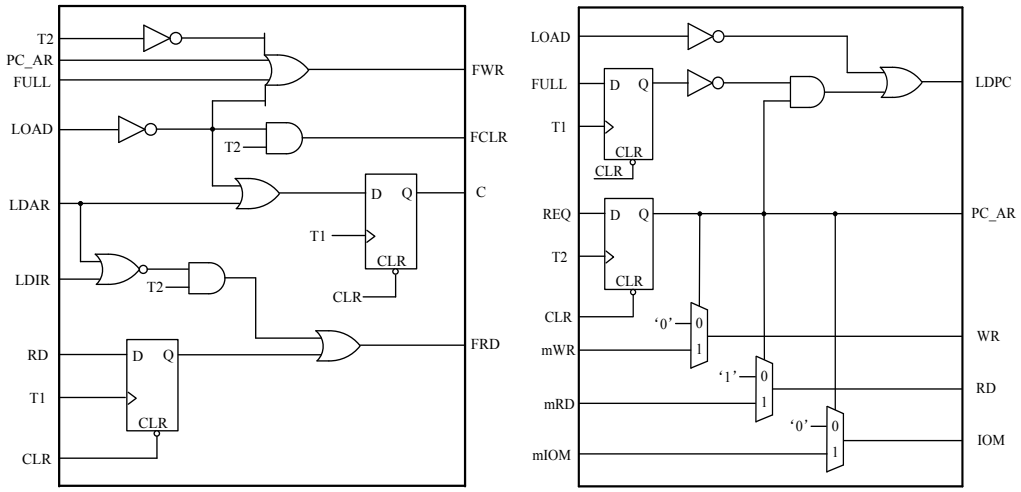



**图 4-2-3　指令预取控制逻辑图**

处理器的时钟及节拍电位由时序电路产生，为每周期 3 节拍，如图 4-2-4 所示。

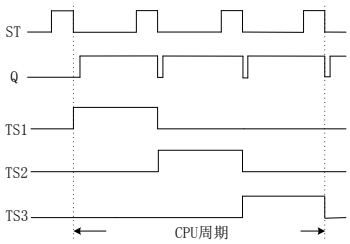



**图 4-2-4　时序电路图**

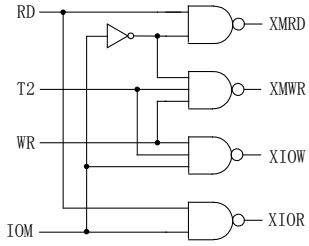



**图 4-2-5　读写控制逻辑**

WR、RD、IOM 为一组用于控制存储器和输入输出设备读写的信号，其控制的具体逻辑如图 4-2-5 所示， IOM=1 时对 I/O 设备进行读写操作，IOM=0 时对 MEM 进行读写操作，RD=1 时为读，WR=1 时为写。在数据通路上面还有一个多路开关，它用来控制地址总线上的地址是来自程序计数器还是 AR 地址寄存器，当 PC_AR=1 时，地址来自 AR 地址寄存器，当 PC_AR=0 时，地址是来自 PC 计数器。

3．相关处理

由于“指令执行”和“指令预取”是以重叠方式运行的，所以系统必然存在一些相关情况。本系统制定如下的控制策略来解决相关问题：

指令预取部件和执行部件可能同时用到 C 总线，因此对取指操作和执行操作设置优先级，当发生竞争时，执行段访内优先。

具体参照数据通路图来讲，就是当执行部件遇到访内指令需要占用外总线时， 微控器发出访内请求 REQ 信号，BIU 在下一机器周期将暂停指令预取，让出总线控制权，由执行部件通过

总线对外部设备进行读/写操作。控制相关的问题相对来讲要简单的多了，当执行段执行程序转移的同时只须由微控器发出 FCLR 信号清除预取指令缓冲栈就可以实现。

4．微程序控制器设计

基于上面的讨论，本系统所涉及的微程序流程可设计如图 4-2-6 所示。当程序准备执行时，前两个机器周期向指令缓冲队列 FIFO 预取两条指令，然后转入微程序运行阶段，后续指令的预取在微程序运行时完成。具体的实现方式是当指令的执行不需要占用 C 总线时，在 T1 时刻完成指令的预取。由于执行 JMP 指令时需要清空指令缓冲队列，所以在 JMP 指令执行后插入两条空操作用来向指令缓冲队列中预取两条指令，以确保执行部件可以从指令缓冲队列中读到正确的指令。

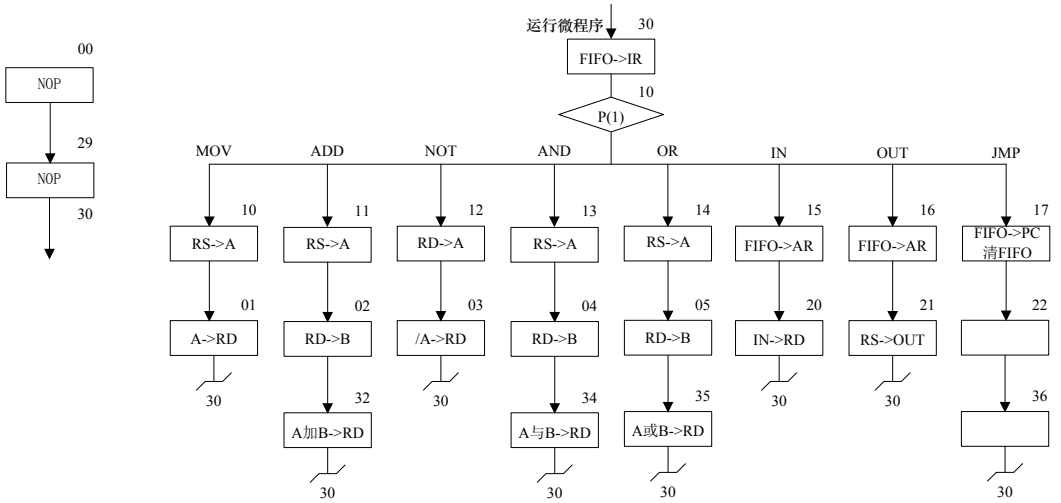



**图 4-2-6　微程序流程图**

**表 4-2-3 微代码的指令格式**

| 23 | 22 | 21 | 20 | 19 | 18-15 | 14-12 | 11-9 | 8-6 | 5-0 |
|---|---|---|---|---|---|---|---|---|---|
| REQ | 保留 | WR | RD | IOM | S3-S0 | A字段 | B字段 | C字段 | UA5-UA0 |

A字段

| 14 | 13 | 12 | 选择 |
|---|---|---|---|
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | LDA |
| 0 | 1 | 0 | LDB |
| 0 | 1 | 1 | LDRi |
| 1 | 0 | 0 | 保留 |
| 1 | 0 | 1 | LOAD |
| 1 | 1 | 0 | LDAR |
| 1 | 1 | 1 | LDIR |

B字段

| 11 | 10 | 9 | 选择 |
|---|---|---|---|
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | ALU_B |
| 0 | 1 | 0 | RS_B |
| 0 | 1 | 1 | RD_B |
| 1 | 0 | 0 | 保留 |
| 1 | 0 | 1 | 保留 |
| 1 | 1 | 0 | 保留 |
| 1 | 1 | 1 | 保留 |

C字段

| 8 | 7 | 6 | 选择 |
|---|---|---|---|
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | P<1> |
| 0 | 1 | 0 | 保留 |
| 0 | 1 | 1 | 保留 |
| 1 | 0 | 0 | 保留 |
| 1 | 0 | 1 | 保留 |
| 1 | 1 | 0 | 保留 |
| 1 | 1 | 1 | 保留 |

当全部微程序设计完成后，应将每条微指令代码化，表 4-2-4 即为将图 4-2-6 的微程序流程图按表 4-2-3 所示微指令格式转化而成的“二进制微代码表”。

它在前面 CISC 模型机的指令格式的基础上，增加了 REQ 信号。REQ 信号在执行 IN、OUT

指令时有效，表示该指令的执行需要占用 C 总线。

**表 4-2-4 二进制代码表**

| 地址 | 十六进制表示 | 高五位 | S3-S0 | A 字段 | B 字段 | C 字段 | UA5-UA0 |
|---|---|---|---|---|---|---|---|
| 00 | 00 00 29 | 00000 | 0000 | 000 | 000 | 000 | 101001 |
| 01 | 00 32 30 | 00000 | 0000 | 011 | 001 | 000 | 110000 |
| 02 | 00 26 32 | 00000 | 0000 | 010 | 011 | 000 | 110010 |
| 03 | 02 32 30 | 00000 | 0100 | 011 | 001 | 000 | 110000 |
| 04 | 00 26 34 | 00000 | 0010 | 010 | 011 | 000 | 110100 |
| 05 | 00 26 35 | 00000 | 0011 | 010 | 011 | 000 | 110101 |
| 10 | 00 14 01 | 00000 | 0000 | 001 | 010 | 000 | 000001 |
| 11 | 00 14 02 | 00000 | 0000 | 001 | 010 | 000 | 000010 |
| 12 | 00 16 03 | 00000 | 0000 | 001 | 011 | 000 | 000011 |
| 13 | 00 14 04 | 00000 | 0000 | 001 | 010 | 000 | 000100 |
| 14 | 00 14 05 | 00000 | 0000 | 001 | 010 | 000 | 000101 |
| 15 | 80 60 20 | 10000 | 0000 | 110 | 000 | 000 | 100000 |
| 16 | 80 60 21 | 10000 | 0000 | 110 | 000 | 000 | 100001 |
| 17 | 00 50 22 | 00000 | 0000 | 101 | 000 | 000 | 100010 |
| 20 | 18 30 30 | 00011 | 0000 | 011 | 000 | 000 | 110000 |
| 21 | 28 04 30 | 00101 | 0000 | 000 | 010 | 000 | 110000 |
| 22 | 00 00 36 | 00000 | 0000 | 000 | 000 | 000 | 110110 |
| 29 | 00 00 30 | 00000 | 0000 | 000 | 000 | 000 | 110000 |
| 30 | 00 70 50 | 00000 | 0000 | 111 | 000 | 001 | 010000 |
| 32 | 04 B2 30 | 00000 | 1001 | 011 | 001 | 000 | 110000 |
| 34 | 01 32 30 | 00000 | 0010 | 011 | 001 | 000 | 110000 |
| 35 | 01 B2 30 | 00000 | 0011 | 011 | 001 | 000 | 110000 |
| 36 | 00 00 30 | 00000 | 0000 | 000 | 000 | 000 | 110000 |

本实验为提高实验的效率和实验的成功率,特别是为了能基于数据通路图方式来调试实验，达到好教好学的效果，处理器中的运算器与 REG 堆、微程序控制器、指令预取控制部件、先进总线接口（支持流水技术)、存储器 RAM、IN 单元、OUT 单元等都是用实验系统上的单元电路来构建的，只需在电路搭建完成后加载设计的微程序，就可构成一具有重叠功能的模型机。.

### 4.2.4 实验步骤

1．关闭实验系统电源，把时序与操作台单元的“MODE”短路块拔开，使系统工作在三节拍模式，JP1、JP2 用短路块均将 1、2 短接，按图 4-2-7 连接实验电路。

2．编写一段机器指令程序

地址（H） 内容（H） 助记符 说明

| | | | |
|---|---|---|---|
| 00 | 50 | IN | IN—>R0 |
| 01 | 40 | | |
| 02 | 51 | IN | IN—>R1 |
| 03 | 40 | | |
| 04 | 06 | MOV | R1—>R2 |
| 05 | 18 | ADD | R0+R2—>R0 |
| 06 | 60 | OUT | R0—>OUT |
| 07 | 80 | | |
| 08 | 70 | JMP | 00—>PC |
| 09 | 00 | | |

3．联上 PC 机，运行 TDX-CMX 联机软件，将上述程序写入到相应的地址单元中或用“【转储】—【装载】”功能将该实验对应的文件载入实验系统。

4．将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

在输入单元上置一数据，将时序与操作台单元的开关 KK1 和 KK3 置为‘运行’档，KK2 置为‘单拍’档，每按动一次 ST 按钮，对照数据通路图，分析数据和控制信号是否正确。

当模型机执行完 JMP 指令后，检查 OUT 单元显示的数是否正确，按下 CON 单元的总清按钮 CLR，改变 IN 单元的值，再次执行机器程序，从 OUT 单元显示的数判别程序执行是否正确。

5． 在联机软件界面下，完成装载机器指令后，选择“【实验】－【指令预取模型机】” 功能菜单打开相应动态数据通路图，按相应功能键即可联机运行、调试模型机实验程序。

为了便于分析，将单拍调试情况罗列如下：

第 1 周期：T1：置下一条微指令码；T2：第一条指令的指令码打入 FIFO 中，PC 加 1；T3：空操作。

第 2 周期：T1：置下一条微指令码；T2：第一条指令的指令码地址打入 FIFO 中，PC 加 1；T3：空操作。

第 3 周期：T1：置下一条微指令码；T2：第二条指令的指令码打入 FIFO 中，PC 加 1，第一条指令的指令码打入指令寄存器 IR；T3：空操作。

第 4 周期：T1：置下一条微指令码；T2：第二条指令的指令码地址打入 FIFO 中，PC 加 1，第一条指令的指令码地址打入地址寄存器 AR 中；T3：空操作。

第 5 周期：T1：置下一条微指令码，将地址寄存器 AR 中的地址输出到地址总线；T2：空操作；T3：把 IN 单元的数据打入到 R0 中。

第 6 周期：T1：置下一条微指令码；T2：第三条指令的指令码打入 FIFO 中，PC 加 1，第二条指令的指令码打入指令寄存器 IR；T3：空操作。

第 7 周期：T1：置下一条微指令码；T2：第四条指令的指令码打入 FIFO 中，PC 加 1，第二条指令的指令码地址打入地址寄存器 AR 中；T3：空操作。

第 8 周期：T1：置下一条微指令码，将地址寄存器 AR 中的地址输出到地址总线；T2：空操作；T3：把 IN 单元的数据打入到 R1 中。

第 9 周期：T1：置下一条微指令码；T2：第五条指令的指令码打入 FIFO 中，PC 加 1，第三条指令的指令码打入指令寄存器 IR；T3：空操作。

第 10 周期：T1：置下一条微指令码；T2：第五条指令的指令码地址打入 FIFO 中，PC 加 1；T3：把 R1 中的数据打入到暂存器 A 中。

后面的机器周期由学生自己分析，并思考以下问题：第 5、第 8 机器周期为什么没有向 FIFO 预取数据？

### 4.2.5 性能评测

1． 本实验重叠方案清晰，易于理解。由于该实验是基于重叠执行方式的原理性实验，故指令系统也比较简单。

2． 本实验在前面 CISC 模型机的基础上以重叠方案实现模型机功能，除第一条指令执行前的指令预取操作需要占用单独的机器周期外，其它每条指令的取指操作都不占用单独的机器指令周期。同时引入地址寄存器专用通路，减少了双字节指令的取地址操作所占用的时间。因此缩短了指令的执行时间，提高了指令的执行效率。

3． 与前面的 CISC 模型机相比，硬件上增加了 FIFO、总线控制器和相应的总线。从而大大地提高了指令的执行效率，比如上面的那段机器指令在 CISC 模型机中执行完需 29 个机器周期，而在本模型机实验中，需 22 个机器周期就能完成。

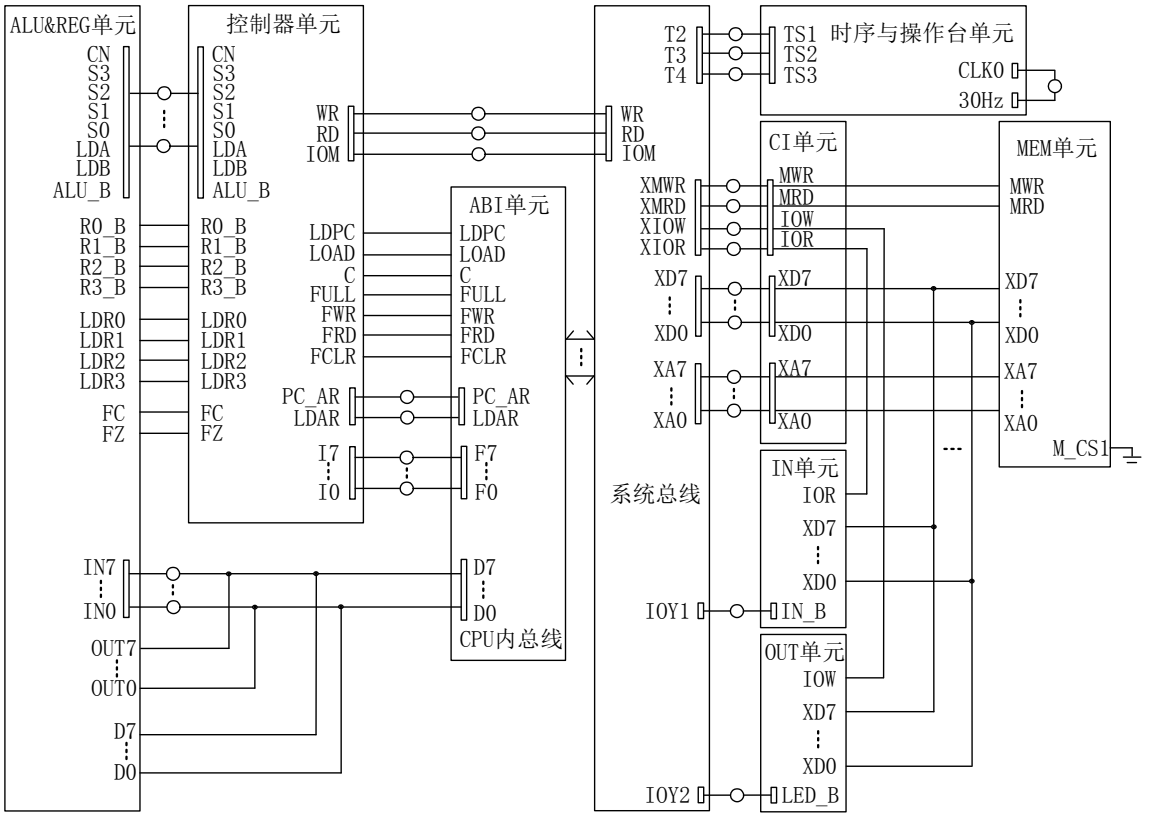

ALU&REG单元
CN
S3
S2
S1
S0
LDA
LDB
ALU_B
R0_B
R1_B
R2_B
R3_B
LDR0
LDR1
LDR2
LDR3
FC
FZ
IN7
IN0
OUT7
OUT0
D7
D0
控制器单元
WR
RD
IOM
LDPC
LOAD
C
FULL
FWR
FRD
FCLR
PC_AR
LDAR
I7
I0
ABI单元
F7
F0
D7
D0
CPU内总线
T2
T3
T4
TS1
TS2
TS3
时序与操作台单元
CLK0
30Hz
XMWR
XMRD
XIOW
XIOR
XD7
XD0
XA7
XA0
系统总线
CI单元
MWR
MRD
IOW
IOR
MEM单元
M_CS1
IN单元
IOY1
IN_B
OUT单元
IOY2
LED_B

**图 4-2-7　实验接线图**

# 4.3 具有三级流水的模型机设计实验

## 4.3.1 实验目的

1． 掌握流水线处理器的一般设计原则和指令系统特征。

2． 在掌握 RISC 处理器构成的模型机实验基础上，进一步将其构成一台具有三级流水功能的模型机。

## 4.3.2 实验设备

PC 机一台， TDX-CMX 实验系统一套。

## 4.3.3 实验原理

1．指令系统设计

本实验采用RISC技术设计的模型机选用常用的八条指令：MOV、ADD、NOT、AND、OR、 LOAD、SAVE 和 JMP 作为指令系统，寻址方式采用寄存器寻址及直接寻址两种方式。指令格式采用单字节及双字节两种格式：

单字节指令（MOV、ADD、NOT、AND、OR）格式如下：

| 7 6 5 4 | 3 2 | 1 0 |
|---|---|---|
| OP-CODE | RS | RD |

其中，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，并规定：

| RS 或 RD | 选定的寄存器 |
|---|---|
| 00 | R0 |
| 01 | R1 |
| 10 | R2 |
| 11 | R3 |

双字节指令（LOAD、SAVE、JMP）格式如下：

| 7 6 5 4 （1） | 3 2 （1） | 1 0 （1） | 7—0 （2） |
|---|---|---|---|
| OP-CODE | RS | RD | P |

其中括号中的 1 表示指令的第一字节，2 表示指令的第二字节，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，P 为操作数地址，占用一个字节。

根据上述指令格式，表 4-3-1 列出了本模型机的八条机器指令的具体格式、汇编符号和指令功能：

其中 LOAD 和 SAVE 指令中的 M 位用来判断操作的对象，当 M=0 时 LOAD 和 SAVE 指令是对 IO 进行操作，当 M=1 时 LOAD 和 SAVE 指令是对存储器进行操作。

**表 4-3-1 指令描述**

| 助记符号 | 指令格式 | 指令功能 |
|---|---|---|
| MOV RS RD | 0000 RS RD | RS → RD |
| ADD RS RD | 0001 RS RD | RD + RS → RD |
| NOT RD | 0010 ** RD | /RD → RD |
| AND RS RD | 0011 RS RD | RD∧RS → RD |
| OR RS RD | 0100 RS RD | RD∨RS → RD |
| LOAD RD | 0101 M * RD P | [P] → RD |
| SAVE RS | 0110 RS M * P | RS → [P] |
| JMP D | 0111 ** ** P | P → PC |

系统采用外设和主存储器各自独立编码的编址方式，I/O 译码单元由采用地址总线高两位作二四译码来实现。

由于用的是地址总线的高两位进行译码，I/O 地址空间被分为四个区，如表 4-3-2 所示：

**表 4-3-2 I/O 地址空间分配**

| A7 A6 | 选定 | 地址空间 |
|---|---|---|
| 00 | IOY0 | 00-3F |
| 01 | IOY1 | 40-7F |
| 10 | IOY2 | 80-BF |
| 11 | IOY3 | C0-FF |

2．流水模型机系统设计

本实验的流水模型机采用三级流水，将系统分为“指令预取部件”、“指令分析部件”和“指令执行部件”，各部件的执行周期均为一个机器周期。

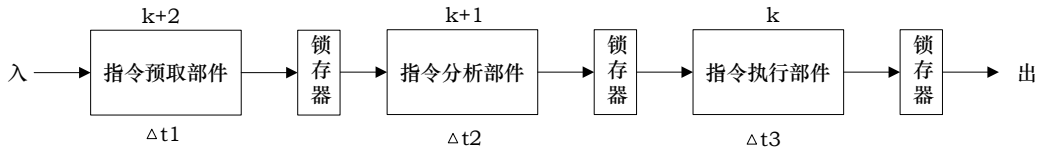



**图 4-3-1 三级流水模型**

“指令预取部件”完成取指操作，将指令和操作数预取到 FIFO 中，“指令预取部件”的设计主要采用了 PC 取指专用通路和 FIFO 技术。“指令分析部件”主要是译码、准备操作数，IR1 将指令码锁存，译码产生出分析部件所需的控制信号，准备操作数，在该机器周期结束时，将

指令码递推到 IR2 锁存，完成指令的分析。“指令执行部件”主要负责执行指令，在 IR2 锁存指令码后，就会译码出执行部件需要的控制信号，完成指令的执行。与此同时分析部件完成了下一条指令的分析。

根据指令系统的设计要求以及三级流水机器指令程序运行的特点，可以设计如图 4-3-2 所示的数据通路图。

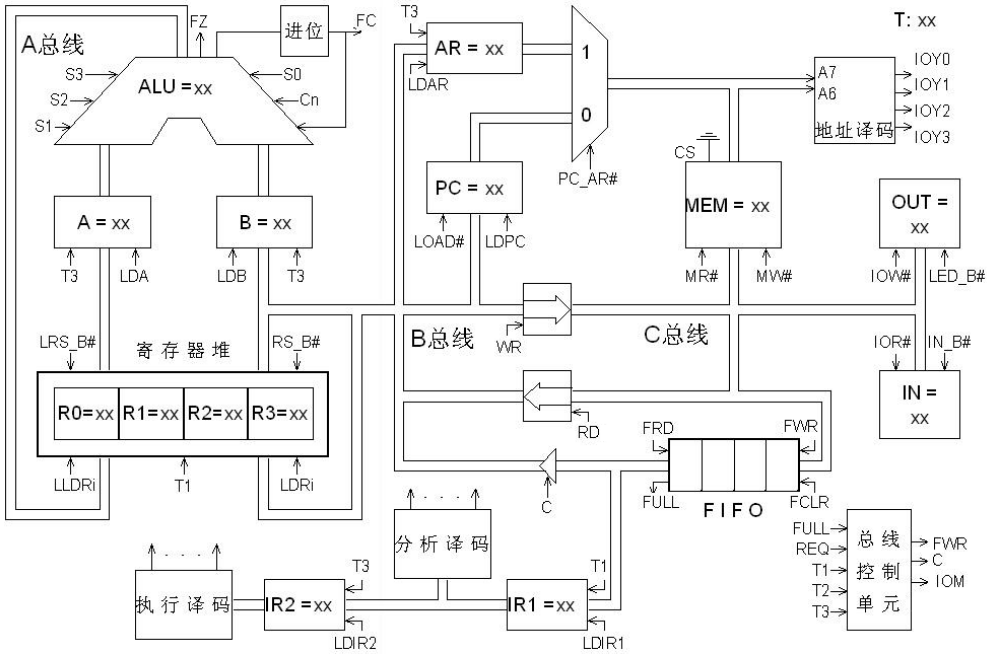



**图 4-3-2　数据通路图**

从上面的数据通路图可以看到，我们在这里是采用多总线方案来构建模型机的，其主要的控制部件是分析译码单元、执行译码单元和总线控制单元。分析译码单元产生分析部件所需的控制信号；执行译码单元产生执行部件所需的控制信号；总线控制单元根据分析段和执行段是否有访内请求 REQ，来自动搭建总线的数据通路并产生相应的控制信号，以实现指令预取或分析和执行段访内的操作。总线控制单元产生的控制信号有： C、WR、RD、IOM 和指令预取信号 FWR，其中信号 C 控制取地址通道，执行双字节指令取地址时有效。

处理器的时钟及节拍电位由时序电路产生。如图 4-3-3 所示，为每机器周期 3 节拍。

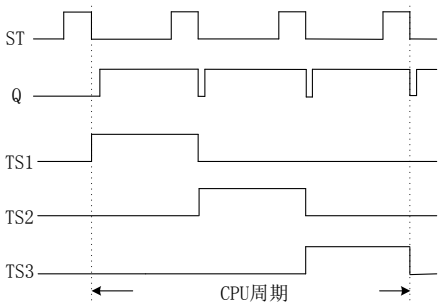



**图 4-3-3　时序电路图**

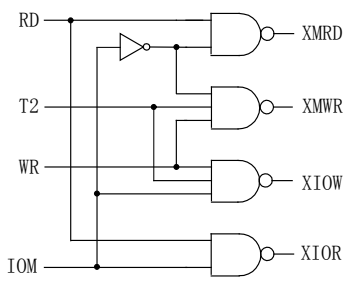



**图 4-3-4　读写控制逻辑**

WR、RD、IOM 为一组信号，用于控制存储器和输入输出设备的读写。进行控制的具体逻辑如图 4-3-4 所示， IOM=1 时对 I/O 设备进行读写操作，IOM=0 时对 MEM 进行读写操作，RD=1 时为读，WR=1 时为写。在数据通路上面还有一个多路开关，它用来控制地址总线上的地址是来自程序计数器还是 AR 地址寄存器，当 PC_AR=1 时，地址来自 AR 地址寄存器，当 PC_AR=0 时，地址是来自 PC 计数器。

在上面数据通路图的基础上采用流水方案可设计本模型机指令系统的指令周期流程如下：

第一周期：向 FIFO 中预取三个数；第二周期：“指令分析部件”T1 打指令，T3 打地址（如果是双字节指令），同时把 IR1 中的指令送到 IR2 中，“指令预取部件”同时预取指令；第三周期以后：“指令执行部件”执行指令，“指令预取部件”、“指令分析部件”也同时工作。

具体情况如下：

第一周期: 取指 T1：PC→AR’,PC+1,RAM→FIFO;
T2：PC→AR’,PC+1,RAM→FIFO;
T3：PC→AR’,PC+1,RAM→FIFO;

第二周期: 取指 T1：PC→AR’,PC+1,RAM→FIFO
T2：PC→AR’,PC+1,RAM→FIFO;
T3：PC→AR’,PC+1,RAM→FIFO;
分析 T1：FIFO→IR1;
T2：
T3：FIFO→AR,IR1 →IR2；

第三周期以后: 取指 T1：PC→AR’,PC+1,RAM→FIFO（如果“执行部件”中不是 LOAD、 STORE 指令）；
T2：PC→AR’,PC+1,RAM→FIFO;
T3：PC→AR’,PC+1,RAM→FIFO;
分析 T1：FIFO→IR1;
T2：
T3：FIFO→AR,IR1 →IR2，向 A、B 中装入操作数；
执行 T1：执行;
T2：清 IR1，IR2，FIFO（如果是 JMP 指令）
T3：

以上的过程反应出了流水技术在“时空”上的并行性。除第一、二两个机器周期外，其它周期三个部件都是同时工作的，每一个周期都会有一个结果输出。

3．相关处理：

由于指令采用并行执行方式，使用流水方案实现，因而相邻指令间的相关问题会影响指令的正确执行，经过分析，将可能遇到的相关问题和对应的解决策略罗列如下：

⑴指令预取部件设置了指令预取专用通路和先入先出栈 FIFO，指令执行部件设置了专用的数据通路和双端口寄存器堆可以解决通用寄存器相关问题。

⑵取指占用 C 总线，分析占用 B 总线，执行占用 A 总线，三总线可以独立运行，避免了由于总线竞争引起的相关问题。

⑶分析、执行可能都用到 B 总线，执行部件 T1 时刻执行，分析部件 T3 时刻打地址，这样可以避免由于 B 总线竞争引起的冲突。

⑷取指、分析、执行可能都用到 C 总线，因此取指、分析、执行设置优先级，当发生竞争时，执行段访内优先，其次是分析段访内，最后是取指。

4．FPGA 芯片设计

在图 4-3-2 数据通路中需用控制器单元的 FPGA 来实现的部分见图 4-3-5。

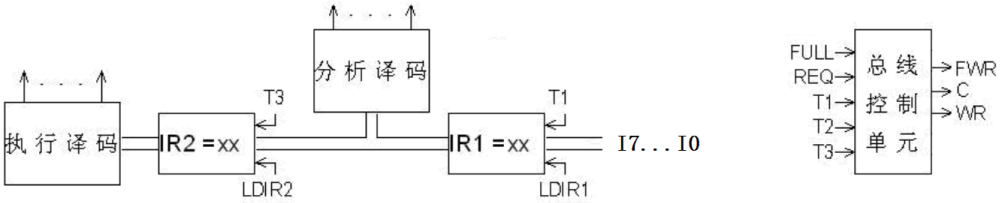



**图 4-3-5 在 FPGA 中实现的数据通路**

其在 FPGA 中的顶层模块电路图见图 4-3-6，而各子模块功能描述的参考程序可参见实验系统随机的光盘文件。

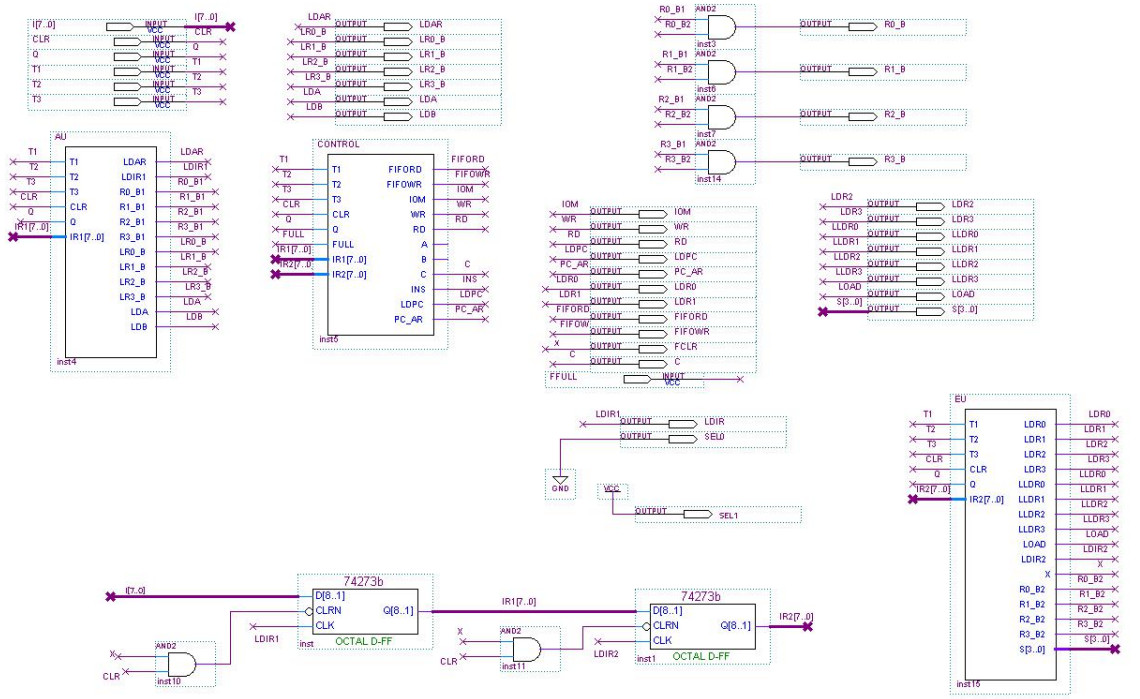



**图 4-3-6 FPGA 顶层模块电路图**

本实验为提高实验的效率和实验的成功率，特别是为了能基于数据通路图方式来调试实验，达到好教好学的效果，处理器中的多端口运算器与 REG 堆、先进总线接口（支持流水技术）、

存储器 RAM、IN 单元、OUT 单元等都是用实验系统上的单元电路来构建的，只将上面图 4-3-5 数据通路中的模块由控制器单元的 FPGA 来实现，即构成一完整的具有三级流水功能的模型机。

### 4.3.4 实验步骤

1．使用 Quartus 软件编辑 “控制器单元”中的 FPGA 逻辑并进行编译，直到编译通过。在 FPGA 中实现的引脚电路图如图 4-3-7 所示：

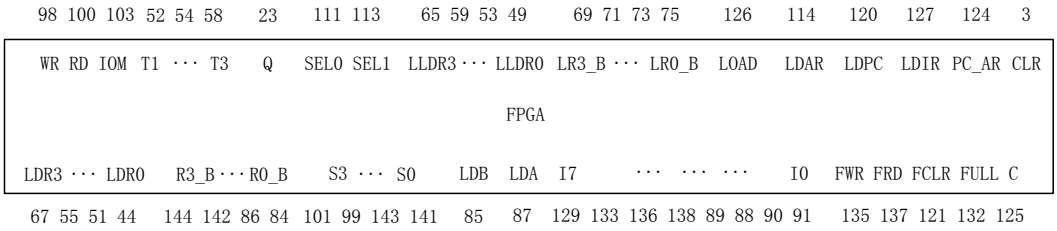



**图 4-3-7 引脚电路图**

2．关闭实验系统电源，把时序与操作台单元的“MODE”短路块拔开，使系统工作在三节拍模式，JP1，JP2 短路块改为 2、3 短接，按图 4-3-8 连接实验电路。

3．打开实验系统电源，将下载电缆插入控制器单元的 C_JTAG 口，把生成的 SOF 文件下载到控制器单元的 FPGA 中。

4．编写一段机器指令程序

| 地址（H） | 内容（H） | 助记符 | 说明 |
|---|---|---|---|
| 00 | 50 | LOAD | IN—>R0 |
| 01 | 40 | | |
| 02 | 51 | LOAD | IN—>R1 |
| 03 | 40 | | |
| 04 | 06 | MOV | R1—>R2 |
| 05 | 18 | ADD | R0+R2—>R0 |
| 06 | 60 | SAVE | R0—>OUT |
| 07 | 80 | | |
| 08 | 70 | JMP | 00—>PC |
| 09 | 00 | | |

5．联上 PC 机，运行 TDX-CMX 联机软件，将上述程序写入相应的地址单元中或用“【转储】—【装载】”功能将该实验对应的文件载入实验系统。

6．将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

在输入单元上置一数据，将时序与操作台单元的开关 KK2 置为‘单拍’档，每按动一次 ST 按钮，对照数据通路图，分析数据和控制信号是否正确。

当模型机执行完 JMP 指令后，检查 OUT 单元显示的数是否正确，按下 CON 单元的总清按

钮 CLR，改变 IN 单元的值，再次执行机器程序，从 OUT 单元显示的数判别程序执行是否正确。

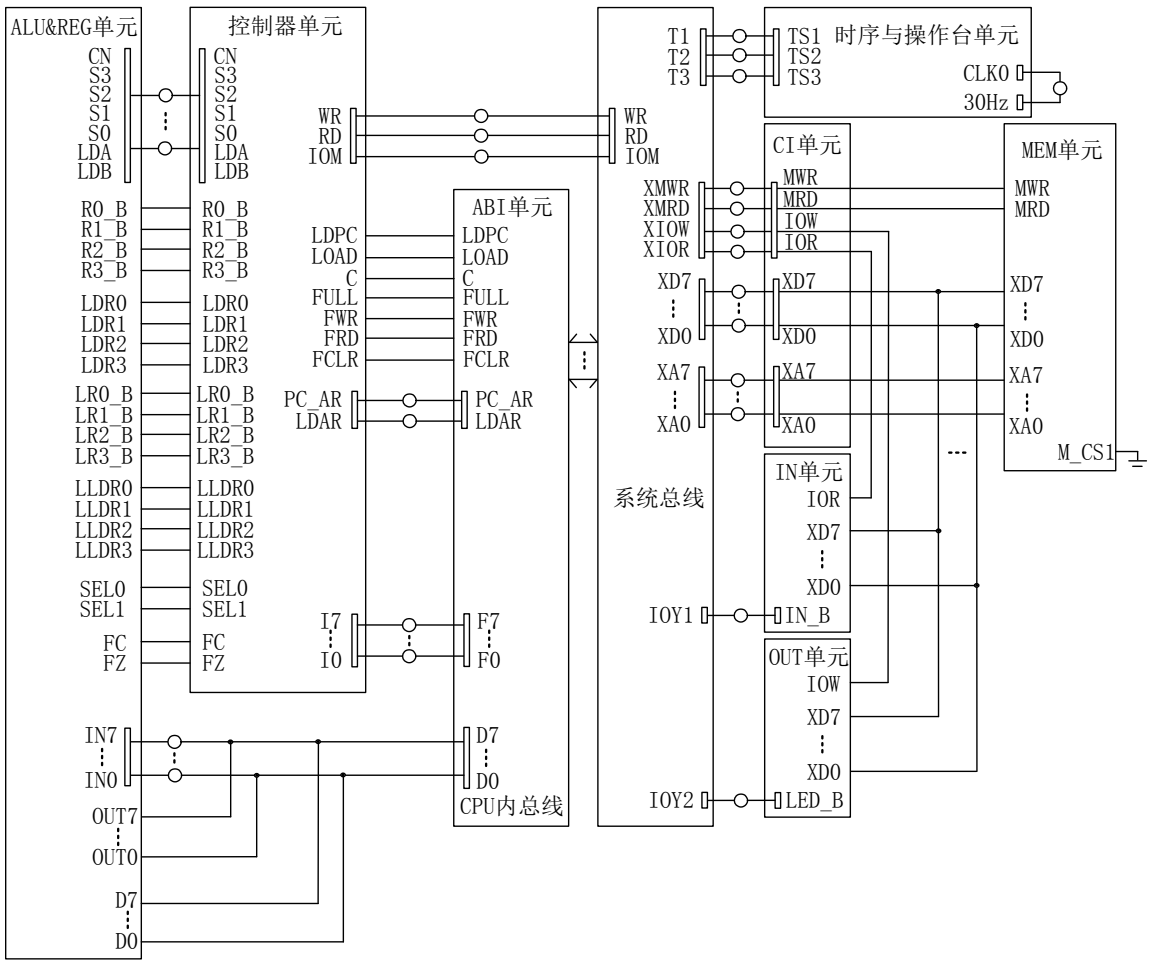



**图 4-3-8 实验接线图**

7． 在联机软件界面下，完成装载机器指令后，选择“【实验】－【三级流水模型机】” 功能菜单打开相应动态数据通路图，按相应功能键即可联机运行、调试模型机的实验程序。

### 4.3.5 性能评测

1． 本实验流水方案清晰，易于理解。由于该实验是基于 RISC 流水的原理性实验，故指令系统也比较简单。

2． 本实验在精简指令处理器的基础上以流水方案实现模型机功能，除第一、二两个机器周期预取指令外，其它每个机器周期都有结果输出。前面基于 RISC 处理器的实验没有指令预取部件和指令执行部件的概念，在遇到访内指令时它需要两个机器周期才能完成。

3． 与前面的基于 RISC 处理器构成的模型机相比，硬件上只增加了一条取指专用通路、先入先出栈 FIFO 和执行指令寄存器 IR2，执行效率就大大提高了，如上面的那段机器指令在本处理器中执行完需 8 个机器周期，而在 RISC 模型机实验中，需 11 个机器周期才能完成。

# 第 5 章　指令并行性为特征的计算机系统

自从 80 年代 RISC 思想成为主流以来，出现了以时间并行性为特征的计算机系统，它们中的处理机只有一条指令流水线，每个机器周期解释一条指令。随后又出现了实现指令级并行的一些新的计算机，让处理机在每个机器周期里可解释多条指令。比较有代表性的是超标量处理机、超长指令字处理机和超流水线处理机。

## 5.1　超标量处理机

### 5.1.1　超标量处理机的概念及基本结构

超标量是指 CPU 内有多条能够并行处理的流水线。在单流水线结构中，指令虽然能够重叠执行，但仍然是顺序的，每个机器周期只能发射一条指令，超标量结构的 CPU 支持指令级并行，每个机器周期可以发射多条指令(2-4 条居多)。

超标量处理机能同时对多条指令进行译码，将可以并行执行的指令送往不同的执行部件，在程序运行期间，由硬件(通常是状态记录部件和调度部件)来完成指令调度。超标量处理机主要是借助硬件资源重复(例如有两套分析、译码器和 ALU 等)来实现空间上指令的并行操作。我们熟知的 pentium 系列、SUN SPARC 系列以及 MIPS 若干型号等都采用了超标量技术。

在单发射处理机中，取指令部件和指令译码部件只设置一套；而操作部件可以只设置一个多功能操作部件，也可以设置多个独立的操作部件。一个有 2 个操作部件组成的单发射处理机如图 5-1-1 所示，它在一个机器周期内只从存储器中取一条指令，并且只对一条指令进行译码，只执行一条指令，只写回一个运算结果。单发射处理机的设计目标是每个机器周期平均执行一条指令。

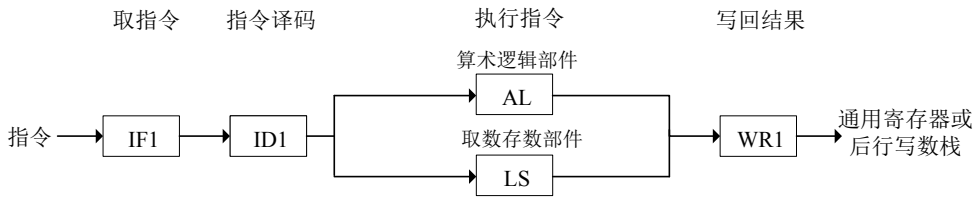



**图 5-1-1　单发射指令流水线**

多发射处理机在一个基本机器周期内同时从指令缓存中读出多条指令，同时对多条指令进行译码。为了实现在一个机器周期同时发射多条指令，通常需要有多个取指部件，多个指令译码部件和多个执行、写回部件。图 5-1-2 是一个同时发射两条指令的多发射处理器的指令流水线，两个取指部件同时从指令缓存中取出两条指令，两个指令译码部件同时对两条指令进行译码，指令的译码结果分别送往 2 个操作部件中执行。

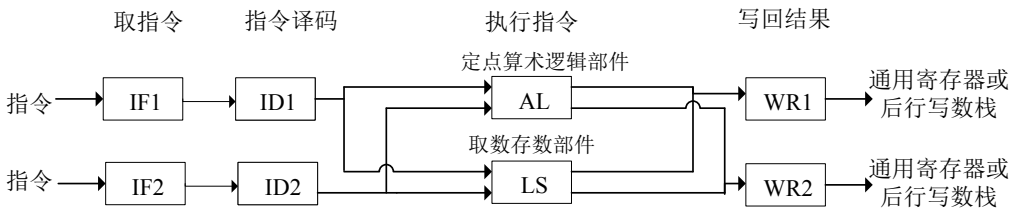



**图 5-1-2 多发射指令流水线**

在超标量处理机中，不仅需要设置多套取指部件和指令译码部件，而且要判断指令之间有无功能部件冲突，有无数据相关和由于条件转移引起的控制相关等。另外，还要通过一套交叉开关把几个指令译码器的输出送到多个操作部件中执行。

在超标量处理机中，有多条指令流水在同时工作，设置有多个能够独立工作的操作部件，为了这些流水线、操作部件能够正确执行给定的程序，必须解决多条流水线的调度问题和操作部件的资源冲突问题。

### 5.1.2 多流水线调度和资源冲突

多条流水线的调度问题非常复杂，通常需要软件（主要是编译器）和硬件的共同结合才能获得比较好的调度效果。在有多条流水线同时工作时，指令的发射顺序和完成顺序对提高超标量处理机的性能非常重要。如果指令的发射顺序是按照程序中的指令排列顺序进行的，称为顺序发射，否则，称为乱序发射。同样，如果指令的完成顺序是按照程序中的指令排列顺序进行的，称为顺序完成，否则，称为乱序完成。

根据多流水线中指令发射顺序和完成顺序的不同组合，多流水线的调度主要有三种，即顺序发射顺序完成，顺序发射乱序完成和乱序发射乱序完成。

在超标量处理机中，通常设置有多个独立的操作部件，由于操作部件的数目通常多于每个机器周期发射的指令条数。这些操作部件可以采用流水线结构，也可以不采用流水线结构。如果采用流水线结构，可以减少发生资源冲突的可能性，因为在流水线结构已经解决了指令并行处理的一些问题。

# 5.2 具有两条流水线的超标量模型机设计实验

## 5.2.1 实验目的

1． 掌握超标量处理器的基本结构、功能特点及设计原则。

2． 在掌握流水线模型机实验基础上，设计并实现一个具有两条流水线的超标量流水模型机。

## 5.2.2 实验设备

PC 机一台， TDX-CMX 实验系统一套。

## 5.2.3 实验原理

1．指令系统设计

采用 RISC 思想设计超标量模型机的指令系统。选用常用的六条指令：MOV、ADD、AND、LOAD、SAVE 和 JMP 作为系统的指令集，寻址方式采用寄存器寻址及直接寻址两种方式。具体的指令格式采用单字节及双字节两种。

单字节指令（MOV、ADD、AND）格式如下：

| 7 6 5 4 | 3 2 | 1 0 |
|---|---|---|
| OP-CODE | RS | RD |

其中，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，并规定：

| RS 或 RD | 选定的寄存器 |
|---|---|
| 00 | R0 |
| 01 | R1 |
| 10 | R2 |
| 11 | R3 |

双字节指令（LOAD、SAVE、JMP）格式如下：

| 7 6 5 4 （1） | 3 2 （1） | 1 0 （1） | 7—0 （2） |
|---|---|---|---|
| OP-CODE | RS | RD | P |

其中括号中的 1 表示指令的第一字节，2 表示指令的第二字节，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，P 为操作数地址，占用一个字节。

根据上述指令格式，表 5-2-1 列出了本模型机的八条机器指令的具体格式、汇编符号和指令功能：

系统采用外设和主存储器各自独立编码的编址方式，I/O 译码单元由采用地址总线高两位作二四译码来实现。

由于用的是地址总线的高两位进行译码，I/O 地址空间被分为四个区，如表 5-2-2 所示：

**表 5-2-1 指令描述**

| 助记符号 | 指令格式 | 指令功能 |
|---|---|---|
| MOV RS RD | 0000 RS RD | RS → RD |
| ADD RS RD | 0001 RS RD | RD + RS → RD |
| AND RS RD | 0011 RS RD | RD∧RS → RD |
| LOAD RD | 0101 ** RD P | [P] → RD |
| SAVE RS | 0110 RS ** P | RS → [P] |
| JMP D | 0111 ** ** P | P → PC |

**表 5-2-2 I/O 地址空间分配**

| A7 A6 | 选定 | 地址空间 |
|---|---|---|
| 00 | IOY0 | 00-3F |
| 01 | IOY1 | 40-7F |
| 10 | IOY2 | 80-BF |
| 11 | IOY3 | C0-FF |

2．超标量流水模型机系统设计

本实验以奔腾处理器为蓝本设计一个简单的超标量流水线处理器，包括两条相对独立的流水线。这两条流水线的取指、分析部件完全相同，设置两个不同功能的执行部件，其中一个执行部件完成运算类指令的功能，另一个执行部件完成访内类指令的功能。指令采用顺序发射顺序执行的调度策略，在指令发射前判断两条分析部件中将要发射的指令是否可以同时发射，若两条指令无功能部件冲突（一条是算术逻辑指令，一条是访内指令）和无数据相关，则可同时发射并执行；否则，采用顺序发射、顺序执行，即先发射Ⅰ指令并执行，Ⅱ指令等待一个机器周期后再发射执行。具体的设计如下：

处理器设计包含两条完整的流水线，流水线分为取指（F）、译码（D）及执行（E）三个部分，其中发射控制判断两条指令能否同时执行，若判断结果是可以同时执行，同时发射 D1、D2 中的指令，否则就顺序发射 D1、D2 中的指令。功能部件的结构图如图 5-2-1 所示：

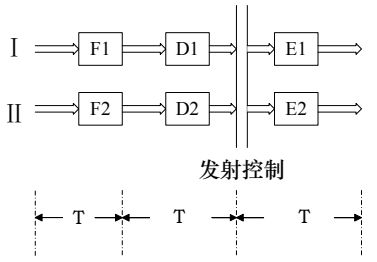



**图 5-2-1 超标量流水线双功能部件结构**

两条指令流水线Ⅰ、Ⅱ的取指 F，译码 D 完全相同，而其执行部件 E1 和 E2 完成不同的功能。如在本实验设计中，E1 完成一般的 ALU 运算指令功能，而 E2 完成装载、存储、转移等访问内存的指令功能，两套指令流水线可同时并行执行。

在每个机器周期按顺序取出两条指令Ⅰ、Ⅱ，然后在下个机器周期进行译码，并由发射控制部件来配对分析并控制发射。若两条指令无功能部件冲突（一条是算术逻辑指令，一条是访内或转移指令）和无数据相关，则可同时发射并执行；否则，采用顺序发射、顺序执行，即先发射Ⅰ指令并执行，Ⅱ指令等待一个机器周期后再发射执行。

根据本模型机指令系统的定义以及超标量流水线处理器的设计特点，可以设计如图 5-2-2 所示的超标量流水模型机的数据通路图。

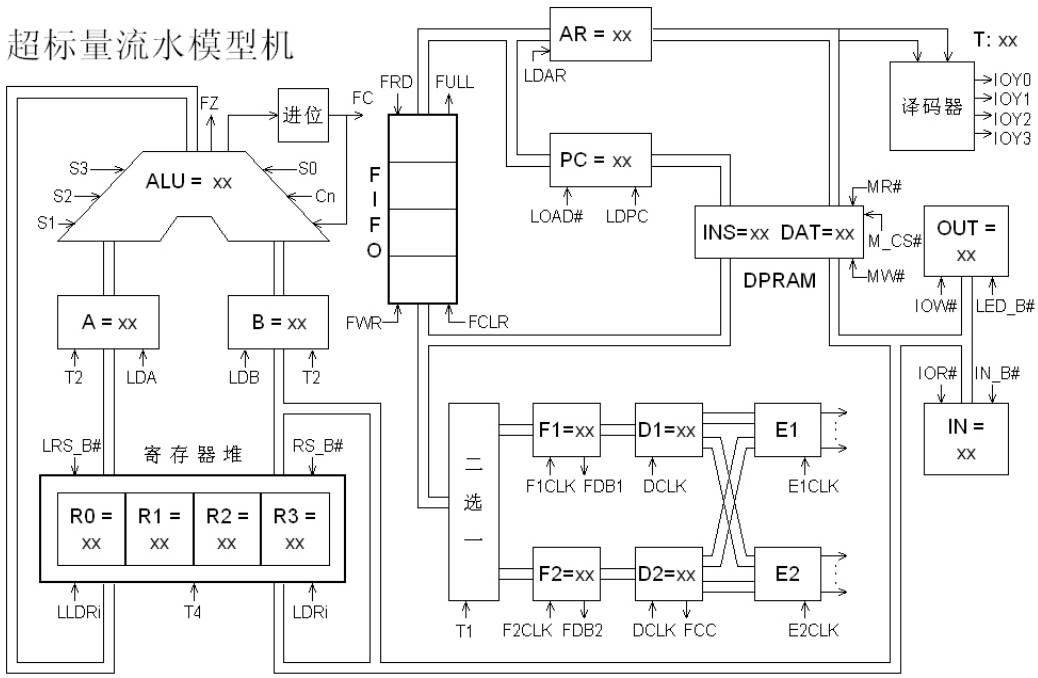



**图 5-2-2　数据通路图**

在数据通路图中，两条指令流水线Ⅰ、Ⅱ的取指（F1、F2），译码（D1、D2）完全相同，而两套执行部件（E1、E2）完成不同的功能。如在本实验设计中，E1 完成一般的 ALU 运算指令功能，而 E2 则借助于一个先进先出的操作数地址缓冲栈（FIFO）来完成装载、存储、转移等访问内存的指令功能。在每个 D（译码）段的末尾进行判断，若装入的两条指令没有功能部件冲突以及无数据相关情况发生，则两条指令可同时发射并往后执行；否则，采用顺序发射，顺序执行，即先发射Ⅰ指令并执行，Ⅱ指令等待一个周期后再发射执行。存储器采用双端口存储器 IDT7130，IDT7130 的引脚分配图如图 5-2-3 所示。

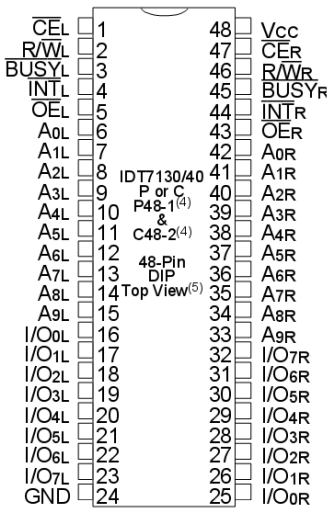



**图 5-2-3 IDT7130 引脚图**

IDT7130 存储器有两组控制线、两组地址线和两组数据线：$CE_L$（左端口片选线）、$OE_L$（左端口输出允许线）、$R/W_L$（左端口读写线）、$A_{0L}$~ $A_{9L}$（左端口地址线）、$I/O_{0L}$~ $I/O_{7L}$（左端口数据线）和 $CE_R$（右端口片选线）、$OE_R$（右端口输出允许线）、$R/W_R$（右端口读写线）、$A_{0R}$~ $A_{9R}$（右端口地址线）、$I/O_{0R}$~ $I/O_{7R}$（右端口数据线）等。两组控制线、地址线和数据线可以分别对存储器进行读写操作。在本实验中，我们把存储器的右端口定义为指令端口 INS，而左端口定义为数据端口 DAT，这样就大大简化了系统的设计，使得取指操作不存在竞争问题。在指令程序存储器数据通路中，程序的地址来自 PC 计数器。每次取指，若是单字节指令则交替地进入 Fi 部件，同时 PC+1，如果是访内或 JMP 指令（双字节指令），则还需在当前机器周期中完成指令操作数的读取和 PC+1 并将该操作数放到一个长度为 4 的单字节操作数地址缓冲栈中（FIFO），待该指令进入执行段时使用。

处理器的时钟及节拍电位由时序电路产生。如图 5-2-4 所示，为每机器周期 4 节拍。

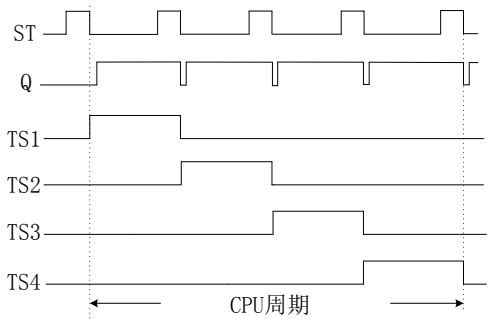



**图 5-2-4 时序电路图**

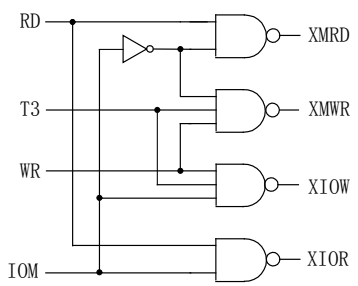



**图 5-2-5 读写控制逻辑**

WR、RD、IOM 为一组用于控制存储器和输入输出设备的读写信号，其具体的控制逻辑如图 5-2-5 所示，其中 IOM 用来选择是对 I/O 设备还是对 MEM 进行读写操作，IOM=1 时

对 I/O 设备进行读写操作，IOM=0 时对 MEM 进行读写操作，RD=1 时为读，WR=1 时为写。

3．相关处理：

由于采用指令并行执行方式，因而相邻指令间的相关问题会影响指令的正确执行，经过分析，将可能遇到的相关问题和对应的解决策略罗列如下：

⑴指令预取部件设置了双端口存储器、指令预取专用通路和先入先出栈 FIFO。

⑵在每个 D（译码）段的末尾进行判断，若装入的两条指令没有功能部件冲突或数据相关，则两条指令可同时继续往后执行，否则，采用顺序发射，顺序执行，即下一个周期的取指（F1、F2）、译码（D1、D2），执行（E2）都延迟一个周期执行。

⑶指令执行部件设置了专用的数据通路和双端口寄存器堆可以解决通用寄存器相关问题。

⑷为访内类指令设置了专用的数据通路。

4． FPGA 程序设计

在图 5-2-2 的数据通路中须用控制器单元的 FPGA 来实现的部分见图 5-2-6。

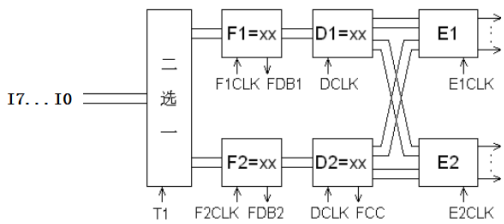



**图 5-2-6　在 FPGA 中实现的数据通路**

本实验为提高实验的效率和实验的成功率，特别是为了能基于数据通路图方式来调试实验，达到好教好学的效果，处理器中的多端口运算器与 REG 堆、先进总线接口（支持超标量）、双端口存储器 RAM、IN 单元、OUT 单元等都是用实验系统上的单元电路来构建的，只将上面图 5-2-6 数据通路中的模块由控制器单元的 FPGA 来实现，既可构成一完整的具有两条流水线的超标量模型机。

FPGA 中的顶层模块电路图见图 5-2-7，而各子模块功能描述的参考程序可参见实验系统随机的光盘文件。

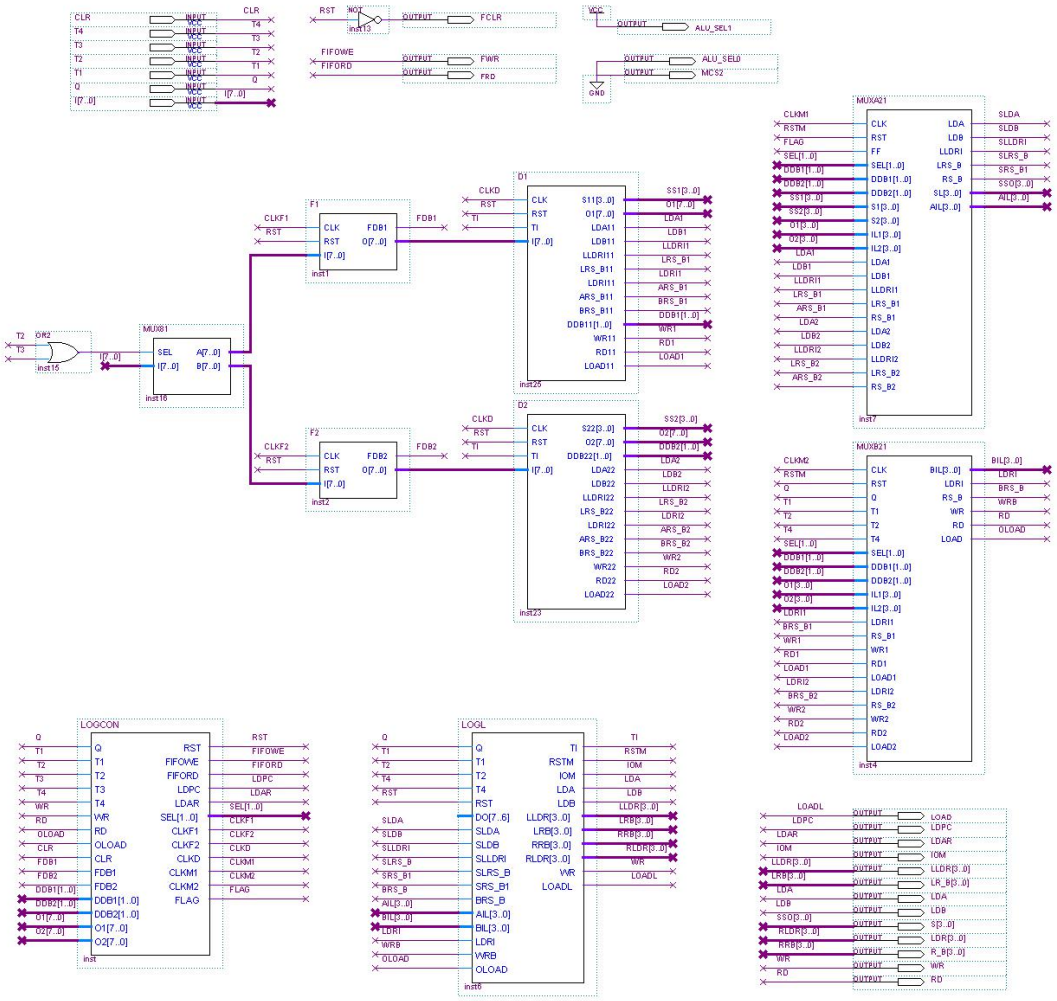



**图 5-2-7 超标量实验 FPGA 顶层模块电路图**

## 5.2.4 实验步骤

1．使用 Quartus 软件编辑 “控制器单元”中的 FPGA 逻辑并进行编译，直到编译通过。在 FPGA 中实现的引脚电路图如图 5-2-8 所示：

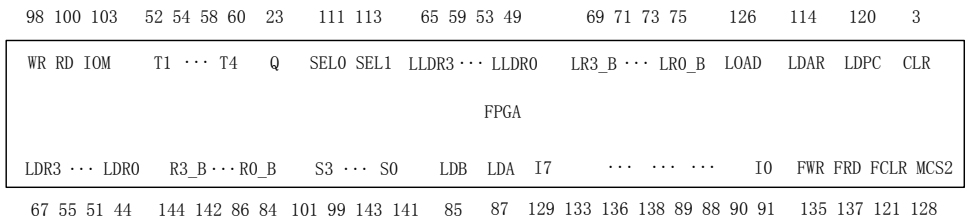



**图 5-2-8 引脚电路图**

2．把时序与操作台单元的“MODE”用短路块短接，使系统工作在四节拍模式， JP1，JP2

短路块改为 2、3 短接，按图 5-2-10 连接实验电路。检查无误后打开实验系统电源，将下载电缆插入控制器单元的 C_JTAG 口，把生成的 SOF 文件下载到控制器单元的 FPGA 中。

3．编写一段机器指令

| 地址（H） | 内容（H） | 助记符 | 说明 |
|---|---|---|---|
| 00 | 50 | LOAD | IN—>R0 |
| 01 | 40 | | |
| 02 | 51 | LOAD | IN—>R1 |
| 03 | 40 | | |
| 04 | 06 | MOV | R1—>R2 |
| 05 | 60 | SAVE | R0—>OUT |
| 06 | 80 | | |
| 07 | 14 | ADD | R0+R1—>R0 |
| 08 | 60 | SAVE | R0—>OUT |
| 09 | 80 | | |
| 0A | 70 | JMP | 00—>PC |
| 0B | 00 | | |

4．联上 PC 机，运行 TDX-CMX 操作软件，将上述程序写入相应的地址单元中或用“【转储】—【装载】”功能将该实验对应的文件载入实验系统。

5．确信指令载入正确后，将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

在输入单元上置一数据，将时序与操作台单元的开关 KK2 置为‘单拍’档，每按动一次 ST 按钮，对照数据通路图，分析数据和控制信号是否正确。

当模型机执行完 JMP 指令后，检查 OUT 单元显示的数是否正确，按下 CON 单元的总清按钮 CLR，改变 IN 单元的值，再次执行机器程序，从 OUT 单元显示的数判别程序执行是否正确。

6． 在联机软件界面下，完成装载机器指令后，选择“【实验】－【超标量流水模型机】”功能菜单打开相应动态数据通路图，按相应功能键即可联机运行、调试模型机上的实验程序。

7、为了便于模型机运行的观测，我们在这里给出上述实验程序的执行时空图如下：

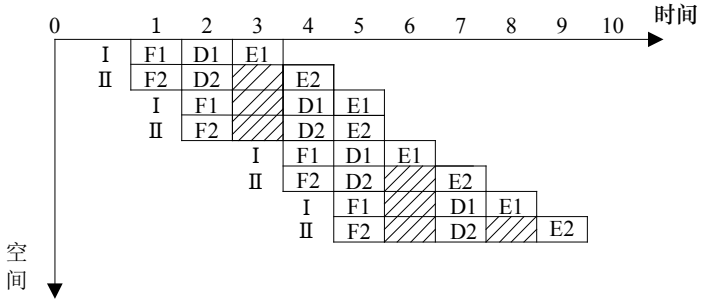



**图 5-2-9 超标量实验程序执行时空图**

具体现执行的情况可解释如下：

按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

第 1 周期：第一节拍，将 00 地址中的指令 50 打入到取指 F1 中，PC 加一。

第二节拍，将 01 地址中的当前指令操作数 40 打入到 FIFO 中，PC 加一。

第三节拍，将 02 地址中的指令 51 打入到取指 F2 中，PC 加一。

第四节拍，将 03 地址中的当前指令操作数 40 打入到 FIFO 中。

第 2 周期：第一节拍，将 04 地址中的指令 06 打入到 F1 中，PC 加一；将 F1 中的数打入到 D1 中，F2 中的数打入到 D2 中。

第二节拍，空操作；

第三节拍，将 05 地址的指令 60 打入到 F2 中，PC 加一。

第四节拍，将 06 地址中的当前指令操作数 80 打入到 FIFO 中，PC 加一。

第 3 周期：第一节拍，由于有功能部件冲突，经过分析将 D1 中的 50 指令　　发射出去。

第二节拍，FIFO 读出 40，经译码选中 IN 单元（IOY1 有效）。

第三节拍，空操作。

第四节拍，IN（输入设备）中的数打入到 R0 中。

第 4 周期：第一节拍，将 07 地址中的指令 14 打入到 F1 中，PC 加一；F1 中的数打入到 D1 中，F2 中的数打入到 D2 中；将 D2 中的 51 指令发射出去。

第二节拍，FIFO 读出 40，经译码选中 IN 单元（IOY1 有效）。

第三节拍，将 08 地址的指令 60 打入到 F2 中， PC 加一；

第四节拍，将 09 地址中的当前指令操作数 80 打入到 FIFO 中，PC 加一；IN（输入设备）中的数打入到 R1 中。

第 5 周期：第一节拍，0A 地址中的 70 打入到 F1 中，PC 加一； F1 中的数打入到 D1 中，F2 中的数打入到 D2 中；将 D1 中的指令 06 和 D2 中的指令 60 同时发射出去。

第二节拍，将 0B 地址中的上指令第二字节 00 打入到 FIFO 中，PC 加一；R1 中的数打入 A，FIFO 读出 80，经译码选中 OUT 单元（IOY2 有效）。

第三节拍，空操作；

第四节拍，将 ALU 中的数打入到 R2 中，同时 R0 中的数输出到 LED。

第 6 周期：第一节拍，由于有功能部件冲突，经过分析将 D1 中的 14 指令发射出去。

第二节拍，R0 中的数打入 A，R1 中的数打入 B；

第三节拍，空操作；

第四节拍，将 ALU 中的数打入到 R0 中。

第 7 周期：第一节拍，将 0C 地址中的指令打入到 F1 中，PC 加一； F1 中的数打入到 D1 中，F2 中的数打入到 D2 中；将 D2 中的 60 指令发射出去。

第二节拍，FIFO 读出 80，经译码选中 OUT 单元（IOY2 有效）。

第三节拍，空操作；

第四节拍，将 R0 中数输出到 LED。

… …

上段程序是一个无条件循环程序，读者可以根据实验现象自己分析。

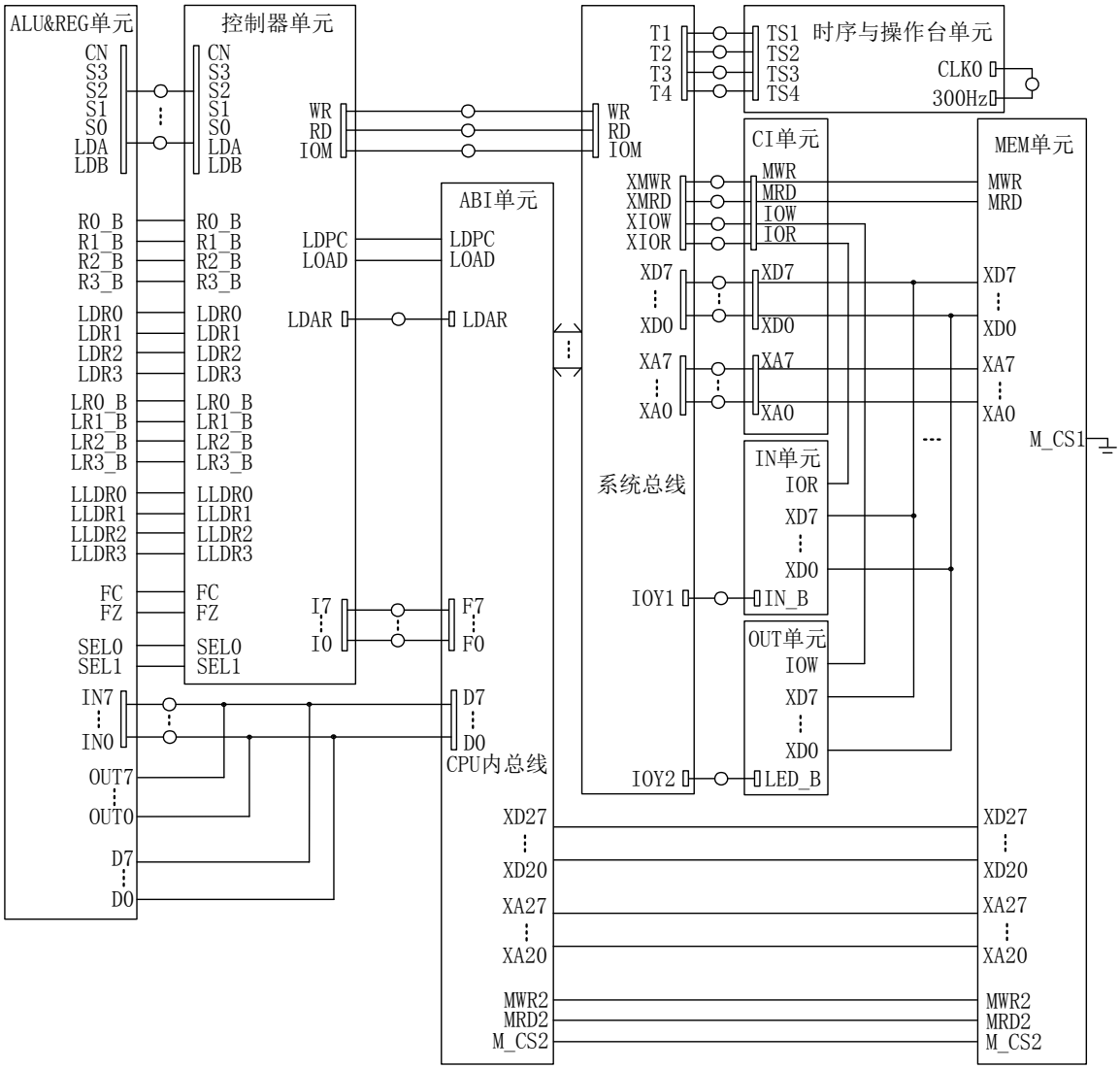



**图 5-2-10 超标量实验接线图**

## 5.2.5 性能评测

1．本实验设计的是一个简单的超标量处理器，与前面的基于流水处理器的模型机相比，硬件上又增加了一条流水线，存储器采用双端口存储器。

2．由于处理器中有两组可独立并行执行的功能部件，指令的发射采用顺序发射顺序完成的调度方法。对于功能部件冲突及数据相关，全部在指令译码阶段通过控制指令发射方式来处理，从而简化了处理器的控制电路。在无功能部件冲突或数据相关的情况下，在一个机器周期就可以直接完成两条指令的执行；在有功能部件冲突或数据相关的情况下，指令的发射策略采

用顺序发射的调度方法，一个机器周期也可输出一条指令的执行结果，所以，此处理器比前边的流水模型机的执行效率又有了很大的提高。

3．由此可知本模型机效率完全取决于指令流的优化，而这主要由系统的编译程序来完成。

# 附录 1　控制器单元和扩展单元对应 FPGA 引脚配置说明

本实验平台提供了两块 FPGA，一块位于控制器单元，一块位于扩展单元，芯片都是 Intel 公司的 FPGA。

1、控制器单元中的 FPGA 引脚分配如下表所示：

| 引脚 | 信号 | 引脚 | 信号 | 引脚 | 信号 | 引脚 | 信号 | 引脚 | 信号 |
|---|---|---|---|---|---|---|---|---|---|
| 3 | CLR | 52 | T1 | 84 | R0_B | 91 | I0* | 137 | FRD |
| 11 | PC_B* | 54 | T2 | 86 | R1_B | 90 | I1* | 135 | FWR |
| 23 | Q | 58 | T3 | 142 | R2_B | 88 | I2* | 121 | FCLR |
| 44 | LDR0 | 60 | T4 | 144 | R3_B | 89 | I3* | 132 | FULL |
| 51 | LDR1 | 49 | LLDR0 | 75 | LR0_B | 138 | I4* | 125 | C |
| 55 | LDR2 | 53 | LLDR1 | 73 | LR1_B | 136 | I5* | 115 | FC |
| 67 | LDR3 | 59 | LLDR2 | 71 | LR2_B | 133 | I6* | 119 | FZ |
| 126 | LOAD | 65 | LLDR3 | 69 | LR3_B | 129 | I7* | 141 | S0* |
| 127 | LDIR | 98 | WR* | 120 | LDPC | 128 | MCS2 | 143 | S1* |
| 114 | LDAR* | 100 | RD* | 87 | LDA* | 111 | SEL0 | 99 | S2* |
| 124 | PC_AR* | 103 | IOM* | 85 | LDB* | 113 | SEL1 | 101 | S3* |
| 106 | EI* | 112 | /INTA* | | | | | 105 | CN* |

注：带*的信号在控制器单元以排针形式开放引出，不带*的信号已连接至各单元。

2、扩展单元中的 FPGA 引脚分配如下表所示：

| 引脚 | 排针 | 引脚 | 排针 | 引脚 | 排针 | 引脚 | 排针 | 引脚 | 排针 |
|---|---|---|---|---|---|---|---|---|---|
| 70 | X0 | 1 | Y0 | 106 | U0 | 24 | Z0 | 2 | H0 |
| 68 | X1 | 28 | Y1 | 103 | U1 | 32 | Z1 | 7 | H1 |
| 66 | X2 | 136 | Y2 | 100 | U2 | 34 | Z2 | 3 | H2 |
| 60 | X3 | 138 | Y3 | 98 | U3 | 39 | Z3(IN) | 43 | H3 |
| 58 | X4 | 46 | Y4 | 144 | U4 | 91 | Z4(IN) | 44 | H4 |
| 54 | X5 | 76 | Y5 | 142 | U5 | 89 | Z5(IN) | 11 | H5 |
| 52 | X6 | 74 | Y6 | 86 | U6 | 88 | Z6(IN) | 77 | H6 |
| 50 | X7 | 72 | Y7 | 84 | U7 | 90 | Z7(IN) | | |

注：排针名称后有（IN）的是只能作为输入的引脚，其他都是输入输出引脚。