计算机组成原理

SHUJI JISUANJI ZUCHENG YUANLI

包 健
冯建文 编著
章复嘉

世纪高等教育精品大系

浙江科学技术出版社

浙江省高等教育重点教材

计算机组成原理

包　健　冯建文　章复嘉　编著

21世纪高等教育精品大系

浙江科学技术出版社

内容简介

本书系统地介绍了单机系统计算机的组成结构与基本工作原理。内容包括计算机系统组成及其层次结构、计算机硬件基础、信息编码与数据表示、运算方法与运算器、存储体系、指令系统、控制器及输入输出系统。

本书配有相应的 Yy-z02 计算机组成原理实验系统及其实验 CAI 软件，使理论教学与实践环节相结合，提高学生的学习兴趣及动手能力。本书每章后面都附有习题，并开发了与本书配套的多媒体教学软件和习题解答。如需要 CAI 课件可发邮件到baojian@hzcnc.com索取。

本书为高等学校计算机专业本科生教材，也可供从事计算机专业的科技人员、计算机爱好者及各类自学人员参考。

图书在版编目（CIP）数据

计算机组成原理/包健，冯建文，章复嘉编著. —杭州：浙江科学技术出版社，2004.8
（世纪高等教育精品大系）
ISBN 7-5341-2432-8

Ⅰ.计… Ⅱ.①包…②冯…③章… Ⅲ.计算机体系结构—高等学校-教材 Ⅳ.TP303

中国版本图书馆 CIP 数据核字（2004）第 064781 号

丛 书 名	世纪高等教育精品大系
书　　名	计算机组成原理
编　　著	包　健　冯建文　章复嘉
出版发行	浙江科学技术出版社
联系电话	（0571）85152486
印　　刷	杭州大众美术印刷厂
开　　本	787×1092 1/16
印　　张	16.5
字　　数	408 000
版　　次	2004 年 8 月第 1 版
印　　次	2004 年 8 月第 1 次印刷
书　　号	ISBN 7-5341-2432-8
定　　价	30.00 元
责任编辑	刘丽丽
封面设计	孙　菁

前　言

本书是浙江省高等教育重点教材，是依据 CCC2002 教程的思想，参照 2004 年 5 月于杭州电子科技大学召开的“中国计算机科学与技术学科教程研讨会”有关课程设置以及对“计算机组成原理”课程的要求来编写的。重点讲述单机系统和串行工作为主的计算机系统组成，主线围绕 Von Neumann 计算机展开，着重于基本概念、基本原理的阐述，但也体现计算机技术发展的最新成果和最新动向。

“计算机组成原理”是计算机科学与技术学科的一门核心专业基础课，为了提高本课程的教学质量，必须安排实验来加深理解课程内容。所以本教材的作者在多年从事“计算机组成原理”课程理论教学和实践教学的基础上，结合本课程的教学特点、难点和要点，对教学内容、教学方法、实验方法和手段进行了重大改革；开发了具有创新实验思想的计算机组成原理实验系统、实验教学 CAI 软件、理论教学 CAI 软件等，使授课内容、教材、实验、教学方法综合配套，以改进教师的授课手段，加强学生的学习理解；充分提高学生的学习兴趣、学习效率及实践动手能力。

计算机组成原理全书共有 8 章，系统地介绍了计算机的组成结构和工作原理。第 1 章概论简要介绍了计算机的分类、计算机系统的层次结构以及“计算机组成原理”课程所研究的范围；第 2 章计算机硬件基础介绍了基本逻辑操作、逻辑函数化简方法、逻辑电路设计基本方法和计算机组成中常用的组合及时序逻辑电路，这一章主要是帮助没有学过数字电路的学生了解数字逻辑电路的基本知识，有助于后面章节的理解；第 3 章信息编码和数据表示介绍了数值数据和非数值数据在计算机中的表示方法，包括原码、反码、补码、移码等编码方法，及定点和浮点表示。另外，还介绍了几种常用的校验码和汉字、字符的表示方法。第 4 章运算方法和运算器则以算法为基础，硬件实现为目标，全面介绍了计算机中如何实现加、减、乘、除算术运算，既阐明了运算器的基本设计方法，又将其与 CPU 结构联系起来，放在整个计算机硬件系统的大环境中来考虑；第 5 章存储系统，从存储器的存储结构入手，介绍了存储器的分类及特点、主存的主要性能；并重点讲述了主存的扩展及与 CPU 的连接，并结合当前存储器技术发展，介绍了高性能主存；介绍了提高主存速度的方法；详细介绍了高速缓存 Cache 的原理；最后介绍了虚拟存储器的原理及存储保护的方法。第 6 章指令系统从设计角度介绍了指令的组成、操作码设计方法、地址码及其寻址方法；结合 Pentium 机指令系统举例，并结合 Yy-z02 计算机组成原理实验系统的指令设计，使学生充分理解和掌握指令系统的设计方法；还简要介绍了 RISC 指令；第 7 章控制器介绍了控制器的两种设计方法：微程序设计控制器和硬布线设计控制器，重点介绍了微程序设计的原理、步骤，并结合 Yy-z02 计算机组成原理实验系统的微程序设计实验，让学生真正掌握其设计方法。第 8 章输入输出系统概要地介绍了输入输出系统的构成和工作方式，

既满足了 CPU 及指令系统的设计需要，同时也为其后续课程作铺垫。

本教材参考学时为 66 学时。各校可根据自己的培养方向和教学时数对内容进行取舍。

本教材由包健、冯建文、章复嘉编写。参加编写工作的还有：戴钧、戴国俊、吴国华、赵辽英、王林泽、吴迎来、俞岳军、王景丽。浙江大学王泽滨教授、杭州电子科技大学王小华教授、严义教授等为教材编写提供了许多宝贵意见，特别是严义教授为教材开发了配套的 Yy-z02 计算机组成原理实验系统，特此表示感谢。

计算机科学与技术是一个发展非常快的专业，编写教材也存在着时效性问题，选材及探讨研究的广度和深度问题等等，因此本书作者期盼广大同行专家提出中肯意见。

感谢浙江省教育厅、浙江科学技术出版社对本书的支持。感谢对本书提出各种意见的专家、教师、学者以及广大的读者。

编著者

2004 年 5 月

目 录

第1章 概 论 …… 1
1.1 计算机的分类 …… 1
1.1.1 按信息处理特性分类 …… 1
1.1.2 按计算机使用范围分类 …… 3
1.1.3 按计算机的规模和处理能力分类 …… 3
1.2 计算机的系统组成 …… 4
1.2.1 计算机硬件系统 …… 6
1.2.2 计算机软件系统 …… 7
1.3 计算机系统层次结构 …… 8
1.4 计算机主要技术指标 …… 10
1.5 计算机的发展 …… 11
小结 …… 13
习题 …… 14
第2章 计算机硬件基础 …… 15
2.1 组合逻辑电路 …… 15
2.1.1 逻辑门 …… 15
2.1.2 逻辑代数的基本公式 …… 15
2.1.3 逻辑函数的化简 …… 17
2.2 组合逻辑电路实例 …… 19
2.2.1 加法器 …… 19
2.2.2 算数逻辑运算单元 ALU …… 22
2.2.3 译码器 …… 25
2.2.4 多路选择器 …… 26
2.3 时序逻辑电路 …… 27
2.3.1 触发器 …… 27
2.3.2 寄存器 …… 28
2.3.3 移位寄存器 …… 29
2.3.4 计数器 …… 30
小结 …… 34
习题 …… 35
第3章 信息编码与数据表示 …… 36
3.1 数值数据的表示 …… 36
3.1.1 进位计数制 …… 36
3.1.2 数据格式 …… 40
3.1.3 定点机器数表示方法 …… 41

3.1.4 浮点机器数表示方法 47
3.2 非数值数据的表示 50
3.2.1 字符编码 51
3.2.2 汉字编码 52
3.3 校验码 54
3.3.1 奇偶校验码 54
3.3.2 海明码 55
3.3.3 循环冗余校验码（CRC） 58
小结 60
习题 61
第4章 运算方法与运算器 63
4.1 定点数的加减运算及实现 63
4.1.1 补码加减运算及运算器 63
4.1.2 机器数的移位运算 67
4.1.3 移码加减运算与判溢 68
4.1.4 十进制加法运算 69
4.2 定点数的乘法运算及实现 70
4.2.1 原码乘法及实现 71
4.2.2 补码乘法及实现 77
4.2.3 阵列乘法器 83
4.3 定点数除法运算及实现 85
4.3.1 原码除法及实现 86
4.3.2 补码除法及实现 90
4.3.3 阵列除法器 94
4.4 定点运算器的组成与结构 95
4.4.1 定点运算器的组成 95
4.4.2 定点运算器的内部总线结构与通路 96
4.5 浮点运算及运算器 99
4.5.1 浮点加减运算 99
4.5.2 浮点乘除运算 102
4.5.3 浮点运算器 106
小结 107
习题 107
第5章 存储体系 109
5.1 存储体系概述 109
5.1.1 存储器分类 109
5.1.2 主存储器性能指标 110
5.1.3 存储器的层次结构 112
5.2 主存储器 112
5.2.1 随机读写存储器 114
5.2.2 只读存储器 124
5.2.3 高性能的主存储器 126

5.3 主存储器与 CPU 的连接 …… 129
5.3.1 存储器芯片介绍 …… 129
5.3.2 存储容量的扩展 …… 130
5.3.3 主存储器与 CPU 的连接方法 …… 132
5.4 高速存储器 …… 138
5.4.1 双端口存储器 …… 139
5.4.2 多体交叉存储器 …… 141
5.4.3 相联存储器 …… 143
5.5 高速缓冲存储器 Cache …… 145
5.5.1 Cache 的基本原理 …… 145
5.5.2 主存与 Cache 的地址映射方式 …… 147
5.5.3 替换算法 …… 150
5.5.4 写策略 …… 151
5.5.5 Cache 的多层次设计 …… 152
5.5.6 Pentium II 的 Cache …… 154
5.6 虚拟存储器 …… 156
5.7 外存储器 …… 159
5.7.1 磁盘存储器 …… 159
5.7.2 光盘存储器 …… 161
5.7.3 闪存盘 …… 162
5.8 存储保护 …… 165
小结 …… 166
习题 …… 167
第 6 章 指令系统 …… 169
6.1 指令格式 …… 169
6.2 寻址方式 …… 172
6.2.1 指令寻址 …… 172
6.2.2 数据寻址 …… 173
6.3 指令类型 …… 177
6.4 指令系统的设计技术 …… 179
6.4.1 指令系统的要求 …… 180
6.4.2 操作码扩展技术 …… 180
6.5 指令系统举例 …… 181
6.5.1 Pentium 指令系统 …… 181
6.5.2 一种 8 位字长的指令系统设计 …… 183
6.6 指令系统的发展 …… 188
6.6.1 指令系统的发展演变 …… 188
6.6.2 RISC 的特点 …… 189
小结 …… 190
习题 …… 190
第 7 章 控制器 …… 192
7.1 控制器的组成及指令的执行 …… 192

7.1.1 控制器的组成 192
7.1.2 指令的执行过程 194
7.1.3 指令周期 199
7.2 控制方式和时序的产生 200
7.2.1 控制方式 200
7.2.2 时序脉冲发生器和启停控制 201
7.3 微程序控制器 202
7.3.1 微程序控制的基本概念 203
7.3.2 微程序控制的基本原理 203
7.3.3 微程序控制器的组成 208
7.3.4 微程序设计技术 209
7.4 微程序控制器及其微程序设计举例 220
7.4.1 微程序控制器组成实例 221
7.4.2 模型机微程序设计 226
7.5 硬布线控制器 231
7.5.1 时序系统 232
7.5.2 硬布线控制器的结构 234
7.5.3 硬布线控制器的设计方法 235
7.5.4 硬布线控制器与微程序控制器的比较 237
7.6 流水线基本工作原理 237
7.7 Pentium Ⅱ CPU 239
7.7.1 Pentium Ⅱ CPU 的技术性能 239
7.7.2 Pentium Ⅱ的内部结构及工作原理 240
小结 242
习题 243
第 8 章 输入输出系统 246
8.1 概 述 246
8.1.1 输入输出系统的构成 246
8.1.2 外设与 CPU 的连接 247
8.1.3 I/O 指令格式 248
8.2 输入输出接口 249
8.2.1 I/O 接口的功能 249
8.2.2 I/O 接口的组成 250
8.3 主机与外设交换信息的方式 251
8.3.1 程序查询方式 251
8.3.2 程序中断方式 252
8.3.3 直接存储器访问（DMA）方式 253
8.3.4 通道方式 254
8.3.5 输入输出处理机（IOP）方式 254
小结 254
习题 255

第7章 控制器

控制器是根据指令指挥和协调计算机各部件进行有条不紊地工作，它是计算机的核心部件。控制器根据其组成及工作原理分为硬布线控制器和微程序控制器。本章重点介绍控制器的组成及工作原理，并结合模型机控制器实例加深理解和掌握微程序控制器的设计方法。

7.1 控制器的组成及指令的执行

计算机上电后，如果没有程序，它是不会工作的；计算机的软件必须编译成计算机所能识别的机器指令装入内存，就可以由计算机来自动完成取出指令并执行指令的功能。能识别机器指令的计算机硬件就是控制器。因此，计算机的工作过程，就是其控制器指令译码和指令执行的过程。本节就介绍控制器的组成，以及控制器从取出指令到执行指令的过程。

7.1.1 控制器的组成

简单计算机的主机包括运算器、控制器和主存 3 部分，而控制器基本组成如下：

1．程序计数器（PC）

程序计数器即指令地址寄存器，用来存放当前正在执行的指令地址或者下一条指令的地址。一种情况是，当指令顺序执行时，由程序计数器本身的递增功能来产生下一条指令的地址：如果存储器按字节编址，而指令字长是 1 字节，则程序计数器加 1；若指令字长是 2 个字节，则程序计数器加 2。另一种情况是，当遇到转移指令时，由指令直接提供转移地址，或者在控制器控制下由运算器形成转移地址，转移地址送往程序计数器 PC 作为下一条指令的地址。

2．指令寄存器（IR）

控制器从内存中取出指令存放在指令寄存器中，以便控制器对指令进行译码、执行。

3．指令译码器

指令划分为操作码和地址码字段，为了执行指令寄存器中的指令，必须对操作码进行译码以识别该指令所要求的操作。指令寄存器中操作码字段的输出就是指令译码器的输入，操作码经过译码后的信号与操作控制信号形成部件一起产生该指令所需要的有一定时序关系的操作控制信号序列。

4．操作控制信号形成部件

根据设计方法及组成的不同，该部件分为：一种是采用时序逻辑电路即硬布线设计的操作控制信号形成部件，另一种是采用微程序设计的操作控制信号形成部件。该部件的功能就是根据指令的操作码以及时序信号，产生取出指令和执行这条指令所需的各种操作控制信号，以便正确地建立数据通路，完成取出指令和执行指令的控制。

5．时序信号产生器

取指令和执行指令所需的各种操作控制信号是有一定时序关系的，因为计算机是一个高速运行的机器，每一个操作的时间要求是非常严格的，不能有任何误差，时序信号产生器就是负责提供时钟信号和机器周期信号，以规定每个操作的时间。该部件还包括启停线路，负责控制时钟脉冲的送出与封锁，从而实现计算机的启动与停止。

图 7.1 是计算机的基本组成框图。该图包括了计算机中的 3 大部件：存储器、运算器和控制器。控制器和运算器一起也称为 CPU，它们之间通过内部总线传递信息；而 CPU 与内存及 I/O 接口之间交换信息是通过外部总线（包括地址总线、数据总线和控制总线）。计算机所有的控制信号都是由控制器中的操作控制信号形成部件发出的，图中省略了许多控制信号。

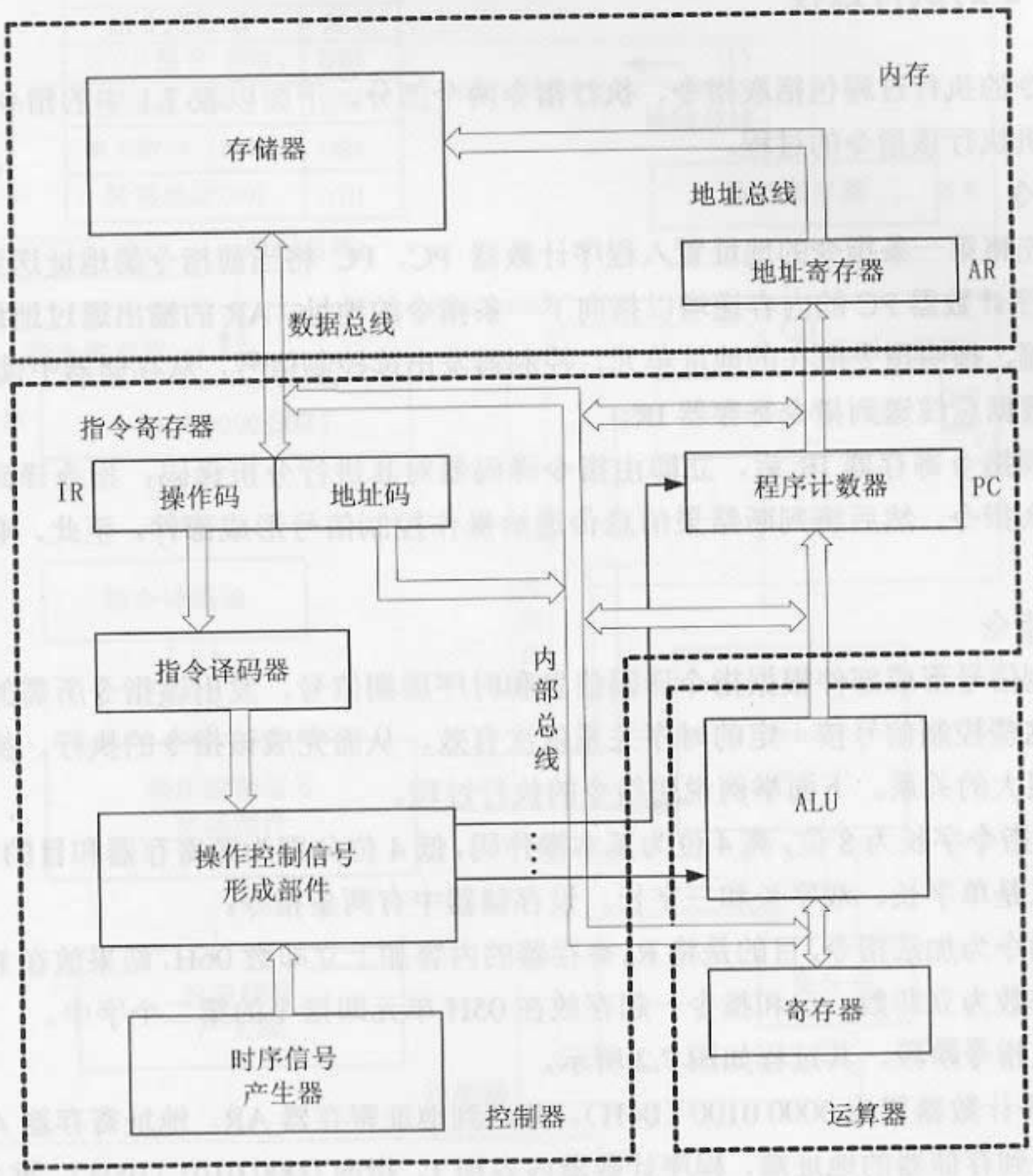


图7.1　计算机基本组成框图

图中假设，操作数的地址和转移地址可以由 IR 的地址码字段直接给出，或者由运算器经过计算后给出，因此 IR 中地址码可以流通到程序计数器 PC、运算器和地址寄存器；即如果 IR 中的地址码为操作数在内存的直接地址，则地址码直接送地址寄存器，并从内存取出操作数；若 IR 中的地址码要经过运算器运算后再送地址寄存器，则地址码送运算器；若 IR 中地址码为直接转移地址，则地址码送程序计数器 PC；若 IR 中地址码为相对转移地址，则地址码还要送运算器计算出转移地址后再送程序计数器 PC。

图 7.1 的各部件都是最基本的组成框图，实际上近年来计算机 CPU 及大规模集成电路 VLSI 的快速发展，使计算机体系结构有了很大的发展，硬件内部结构越来越复杂，比如控制器中要有指令预取队列，可以取出若干条指令存放在指令队列中；也就是将 CPU 设计成流水线的各部件，各个部件可以同时工作。例如还在执行当前指令时，指令部件就可以取下一条指令，这样采用流水线技术可以提高计算机的工作效率和速度，这在 7.6 节中进行详细讲解。此处，为了有利于读者掌握控制器的基本原理，对实际机器进行了简化、提炼。

7.1.2 指令的执行过程

一条指令的执行过程包括取指令、执行指令两个部分。下面以表 7.1 中的指令为例，图 7.2 所示的计算机执行该指令的过程。

1. 取指令

控制器先将第一条指令的地址置入程序计数器 PC，PC 将当前指令的地址送到地址寄存器 AR，同时程序计数器 PC 的内容递增以指向下一条指令的地址；AR 的输出通过地址总线送到存储器的地址端，指明指令所在的地址单元，控制器发出读控制信号，从存储器中读出这条指令；该指令通过数据总线送到指令寄存器 IR。

指令取到指令寄存器 IR 后，立即由指令译码器对其进行分析译码；指令译码器首先判断该指令是什么指令，然后将判断结果信息传递给操作控制信号形成部件，至此，取指令的过程完成。

2. 执行指令

操作控制信号形成部件根据指令译码信息和时序周期信号，发出该指令所需的所有部件的控制信号，这些控制信号按一定的时序关系依次有效，从而完成该指令的执行。执行指令与指令的内容有很大的关系。下面举例说明指令的执行过程。

表 7.1 中指令字长为 8 位，高 4 位为基本操作码，低 4 位分别为源寄存器和目的寄存器地址，一条指令可以是单字长、双字长和三字长。设存储器中有两条指令：

第一条指令为加法指令，目的是将 R_0 寄存器的内容加上立即数 06H，结果放在 R_0 寄存器中；该指令的操作数为立即数，已和指令一起存放在 05H 单元即指令的第二个字中。

首先是取指令阶段，其过程如图 7.2 所示。

（1）程序计数器置为 0000 0100（04H），并送到地址寄存器 AR，地址寄存器 AR 的地址通过地址总线送到存储器的地址端，程序计数器内容加 1，指向 0000 0101（05H），准备取立即数；

（2）控制器发出读信号，将该地址单元的内容 0101 0000（50H）即指令读出，并通过数据

总线送到指令寄存器 IR。

（3）指令寄存器中的指令送到指令译码器进行译码分析判断，并将结果信息送到操作控制信号形成部件。

表 7.1　存放在存储器中的两条指令内容

指令地址	指令机器码	助记符
0000 0100	0101 0000	ADD R_0, 06H
0000 0101	0000 0110（立即数）	
0000 0110	1000 0000	JMP 04H
0000 0111	0000 0100（转移地址）	

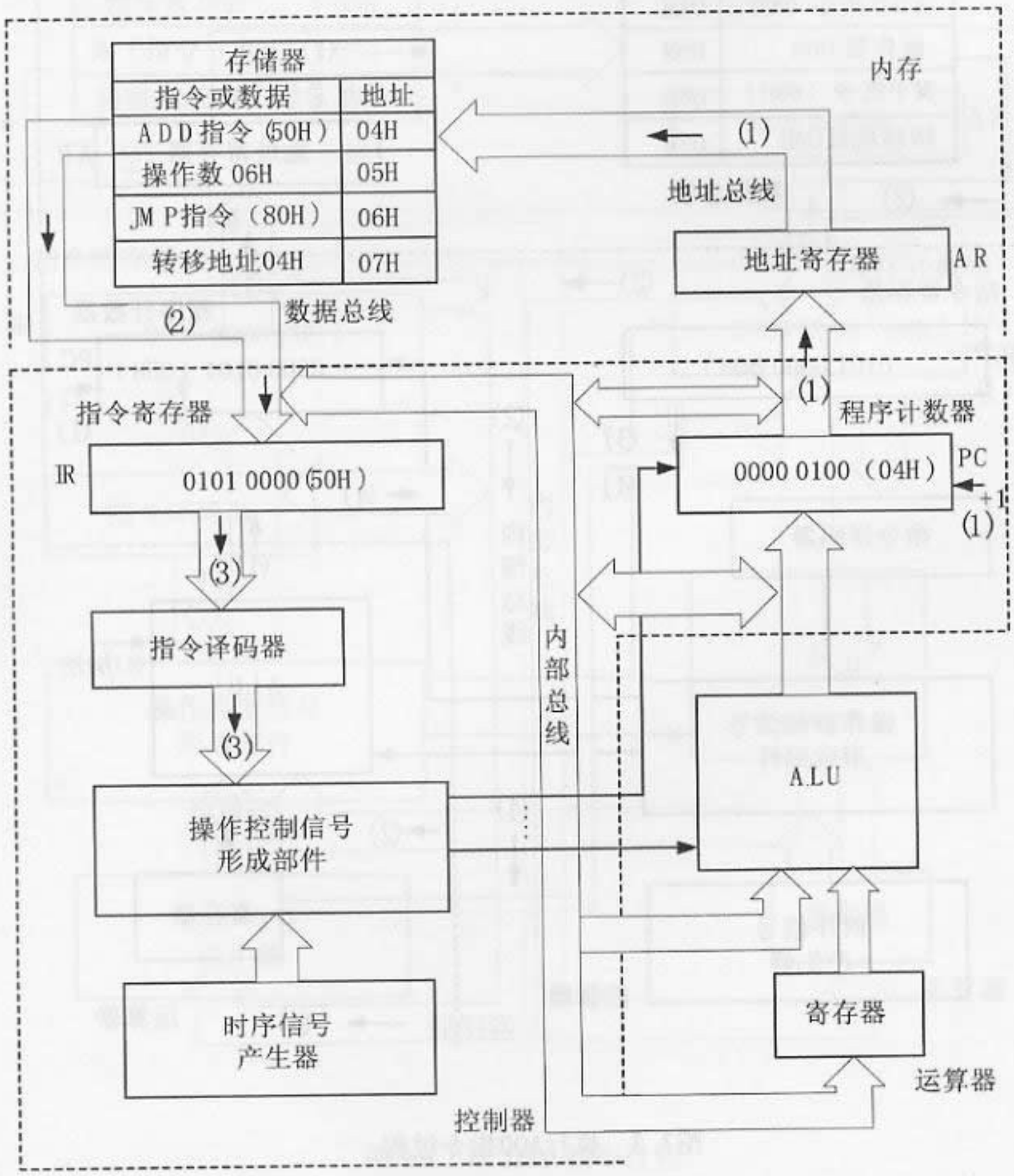


图7.2　取ADD指令过程

其次是执行指令阶段，其过程如图 7.3 所示。

（1）将程序计数器 PC 的内容 0000 0101（05H）送到地址寄存器 AR，同时 PC+1 指向下一条指令的地址 0000 0110（06H），为取下一条指令做好准备。

（2）控制器发读信号，从存储器 05H 单元中读出操作数，并通过数据总线和内部总线送到运算器 ALU。

（3）根据指令寄存器 IR 中的低 4 位（源寄存器和目的寄存器地址），由寄存器地址译码得源操作数寄存器为 R_0，因此，从寄存器 R_0 中取出另一操作数，并送运算器 ALU。

（4）在 ALU 中进行加法运算，并将结果送到目的寄存器 R_0 中存放（由 IR 的低 2 位寄存器地址译码选择目的寄存器为 R_0）。

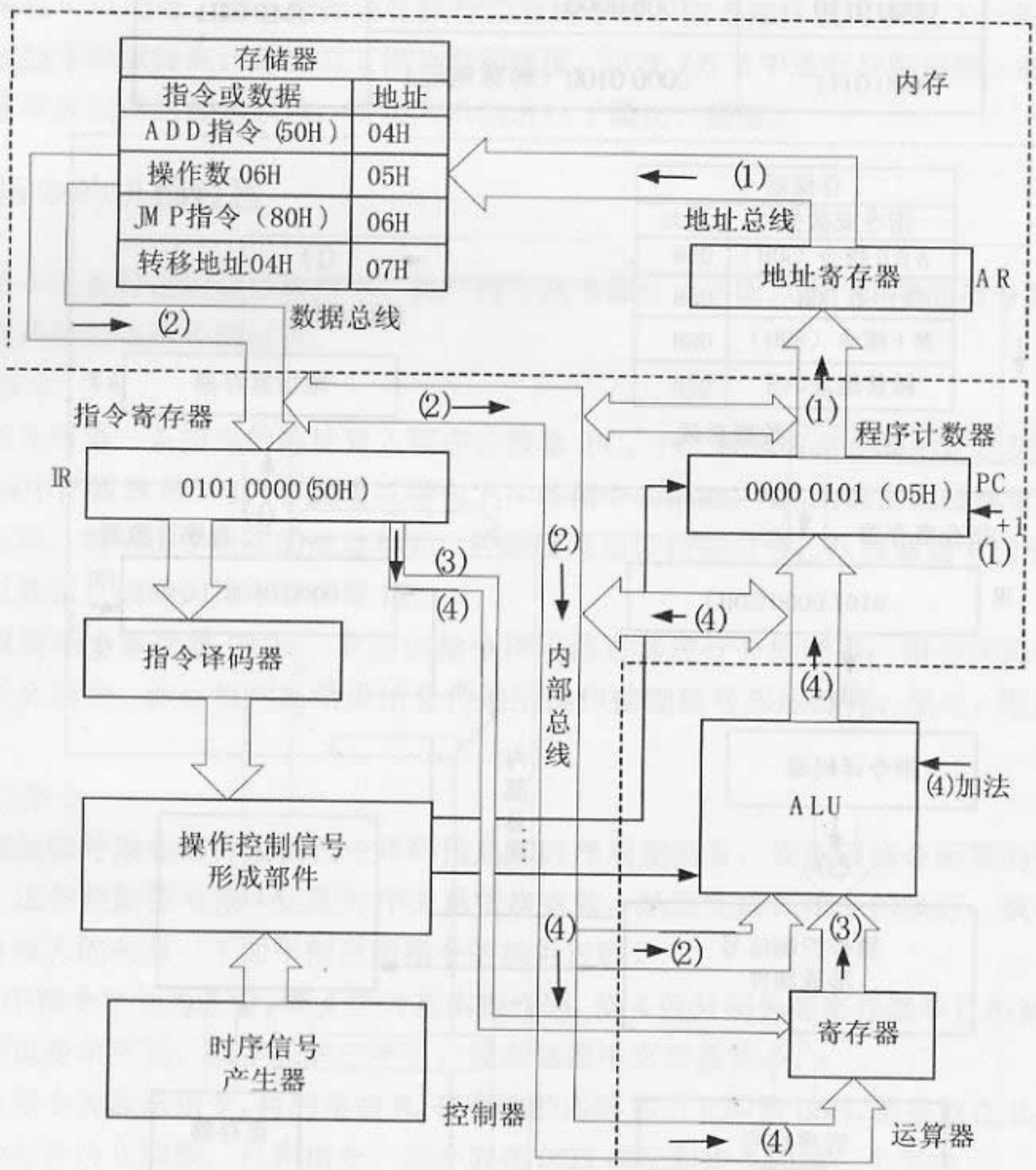


图7.3　执行ADD指令过程

第二条指令是转移指令，目的是将程序转移到 04H 地址的指令去执行。转移地址存放在指

令的第二个字即 07H 地址单元中，可进行直接寻址方式转移。

首先也是取指令阶段，其过程如图 7.4 所示。

（1）程序计数器 PC 的内容 0000 0110（06H）送到地址寄存器 AR，程序计数器 PC 内容加一，指向 0000 0111（07H），准备取转移地址。

（2）地址寄存器 AR 中的内容通过地址总线送到存储器的地址端，控制器发出读信号，将该地址单元的内容即指令读出，并通过数据总线送到指令寄存器 IR。

（3）指令寄存器 IR 中的指令送到指令译码器进行译码分析判断，并将结果信息送到操作控制信号形成部件。

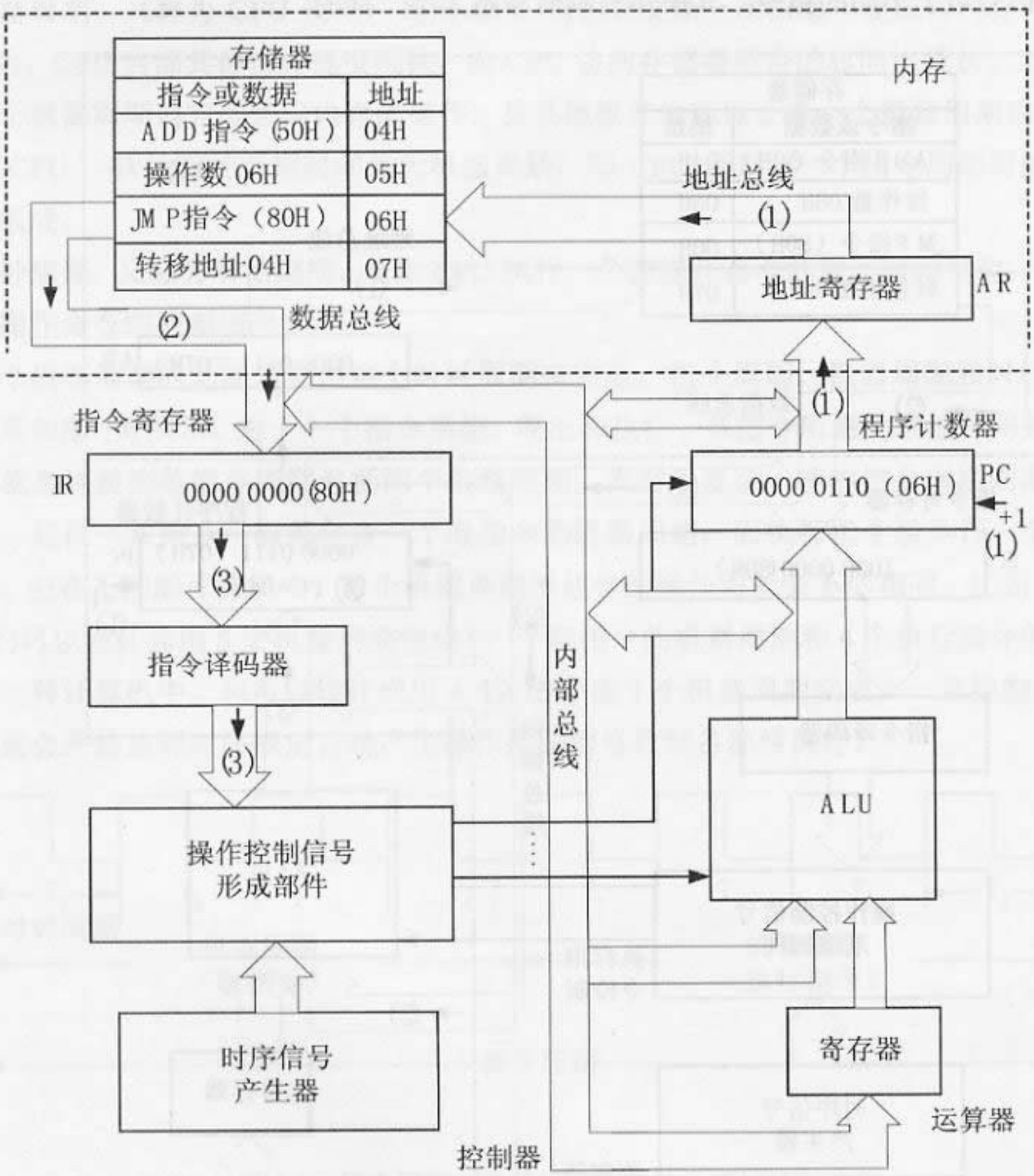


图 7.4　取 JMP 指令过程

其次也是执行指令阶段，其过程如图 7.5 所示。

（1）将程序计数器 PC 的内容 0000 0111（07H）送到地址寄存器 AR，同时 PC+1 指向下一条指令的地址 0000 1000（08H），为取下一条指令做好准备。

（2）控制器发读信号，从存储器 07H 单元中读出转移地址，并通过数据总线和内部总线送到运算器 ALU。

（3）因为该地址为直接转移地址，不需要做转移地址的运算，因此将该地址从 ALU 中直接送到程序计数器 PC，即实现转移操作。

指令的执行过程包括取指令和执行指令两个阶段，这两个阶段的所有操作控制信号均由控制器产生，只是取指令阶段对于所有指令来说，其操作控制信号都是相同的；而执行指令阶段的操作控制信号对于不同的指令，其操作控制信号是大不一样的。

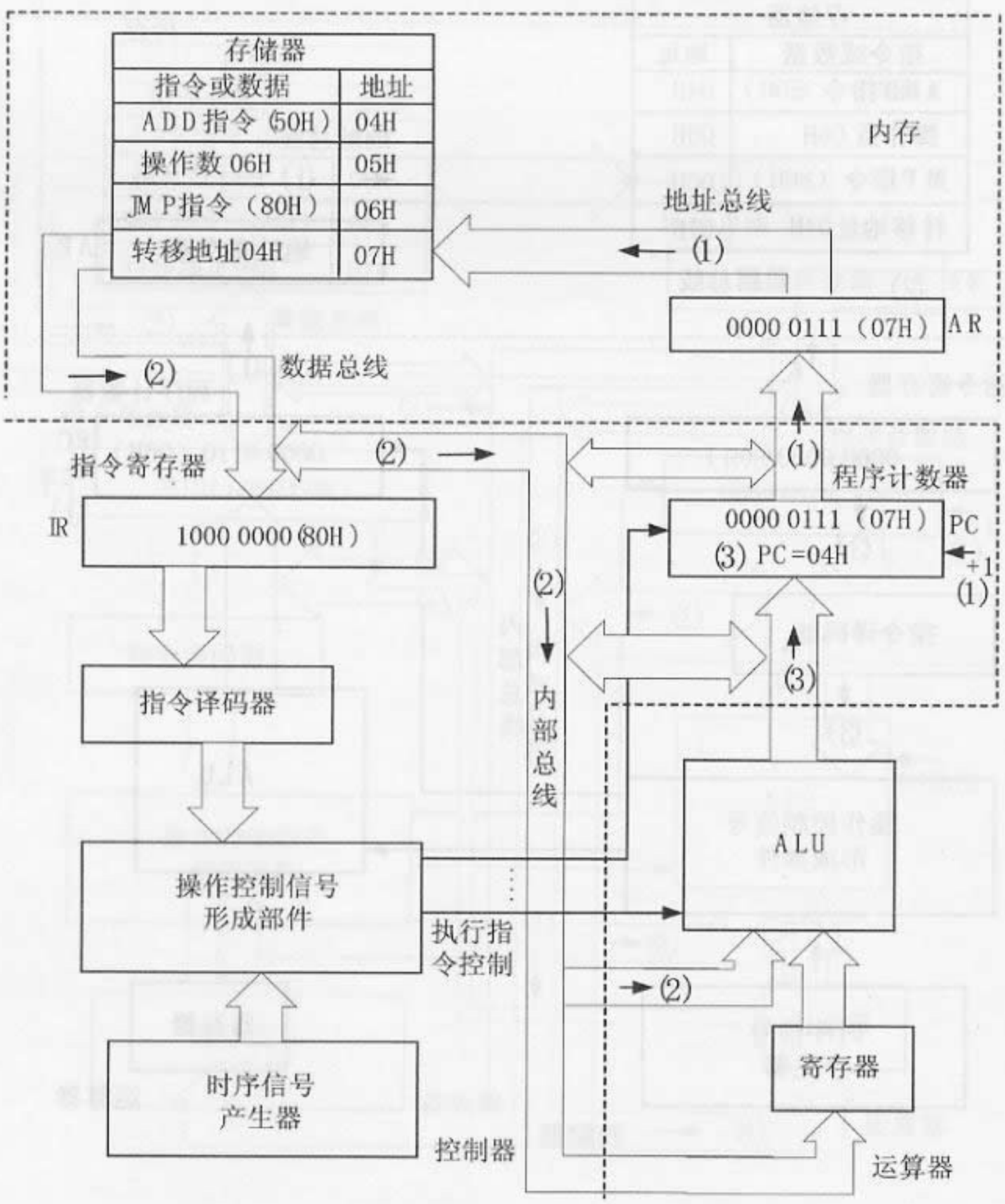


图7.5　执行JMP指令过程

7.1.3　指令周期

计算机在执行每一条指令过程中必须严格地按照一定的时间关系，这种时间关系是由计算机的时序系统来控制的，计算机有 3 级时序系统，即指令周期、机器周期、时钟周期。

指令周期：是指计算机从取出一条指令并完成该指令的执行所需要的时间。也就是说，计算机从取一条指令到执行该指令这两个阶段的所有一系列操作的完成所需要的时间称为一个指令周期。前面已讲过，每条指令的执行阶段的操作控制是不同的，有的简单，有的复杂，因此各种指令的指令周期是不相同的。例如，上述 ADD 指令与 JMP 指令的指令周期一定是不同的。

机器周期：又称为 CPU 周期，是指 CPU 与内存交换一次信息（读或写内存）所需要的时间。因为，CPU 内部其他操作速度很快，而 CPU 访问存储器所需的时间比较长，为了保证 CPU 能在一个机器周期内完成访问内存的操作，且其他操作也能保证在一个机器周期内完成，因此，规定用 CPU 一次访存的最短时间作为机器周期，即 CPU 周期。一个指令周期可能有若干个机器周期组成。

时钟周期：又称为节拍周期，是指 CPU 执行一个微操作命令的最小时间单位，也即 T 周期。所谓微操作命令即控制信号。

一个机器周期的功能需要由多个时钟周期来完成。指令周期、机器周期和时钟周期三者之间的关系如图 7.6 所示。对于一个指令系统，取出和执行一条指令所需的最短时间是两个机器周期，也就是说最简单指令周期包括两个机器周期；而对于复杂一些的指令周期则需要更多的机器周期。任何一条指令一般都包含一个取指令的机器周期，而执行指令最少有一个或更多的机器周期。但在不同的计算机中，每个机器周期所包含的操作可能会不尽相同，比如上述 ADD 指令，我们可以设计为用 5 个机器周期完成（一个取指令的机器周期和 4 个执行指令的机器周期），而在另一种计算机中，也可以设计成用 4 个、6 个或 7 个机器周期完成。一旦控制器设计好了，计算机就会严格按照时序规定自动产生操作控制信号控制各部件执行。

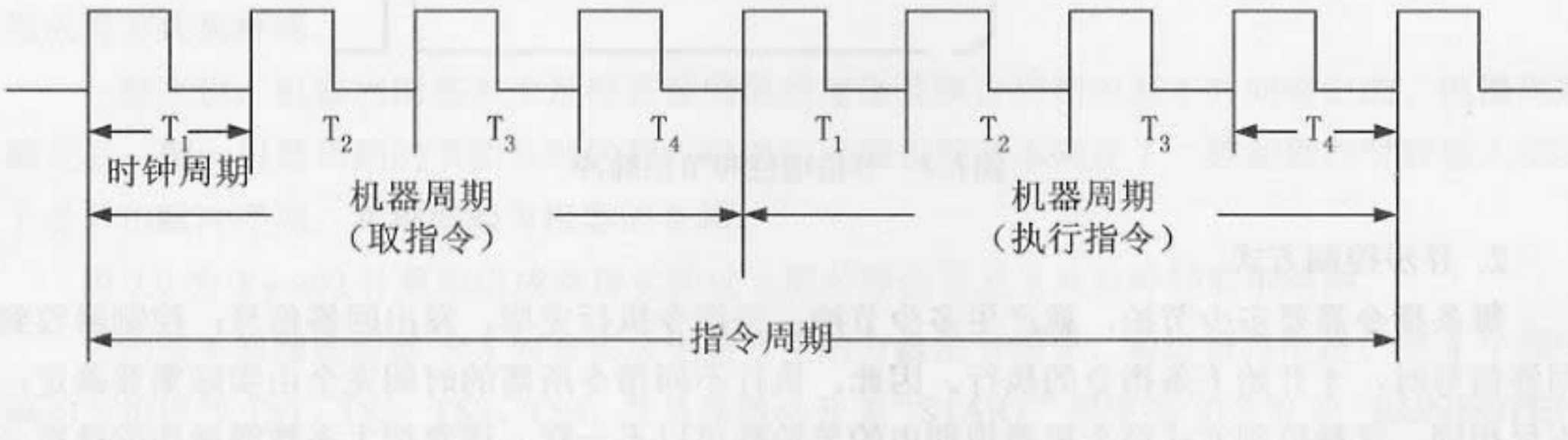


图 7.6　指令周期、机器周期和时钟周期的时序关系

7.2 控制方式和时序的产生

7.2.1 控制方式

由于不同指令所需的微操作控制时序信号不同，每一个微操作的长短、繁简程度及其执行时间也不同，即指令周期是不同的。而对于不同的微操作控制信号序列如何定时及同步，并将信号序列衔接起来，就是控制方式要解决的问题。一般有 3 种控制方式。

1. 同步控制方式

以微操作序列最长的指令为标准，确定控制微操作运行的时钟周期数（节拍数）。控制器产生统一的、顺序固定的、周而复始的节拍电位（机器周期信号）和节拍脉冲（时钟周期信号），如图 7.7 所示。微操作序列短的指令可空着几个节拍不用。也就是说，采用相同的机器周期数和相同的节拍脉冲来形成每条指令的操作控制信号序列，因此每条指令的执行所用的时间都是相同的。

同步控制方式的优点是电路简单，缺点是运行速度慢。

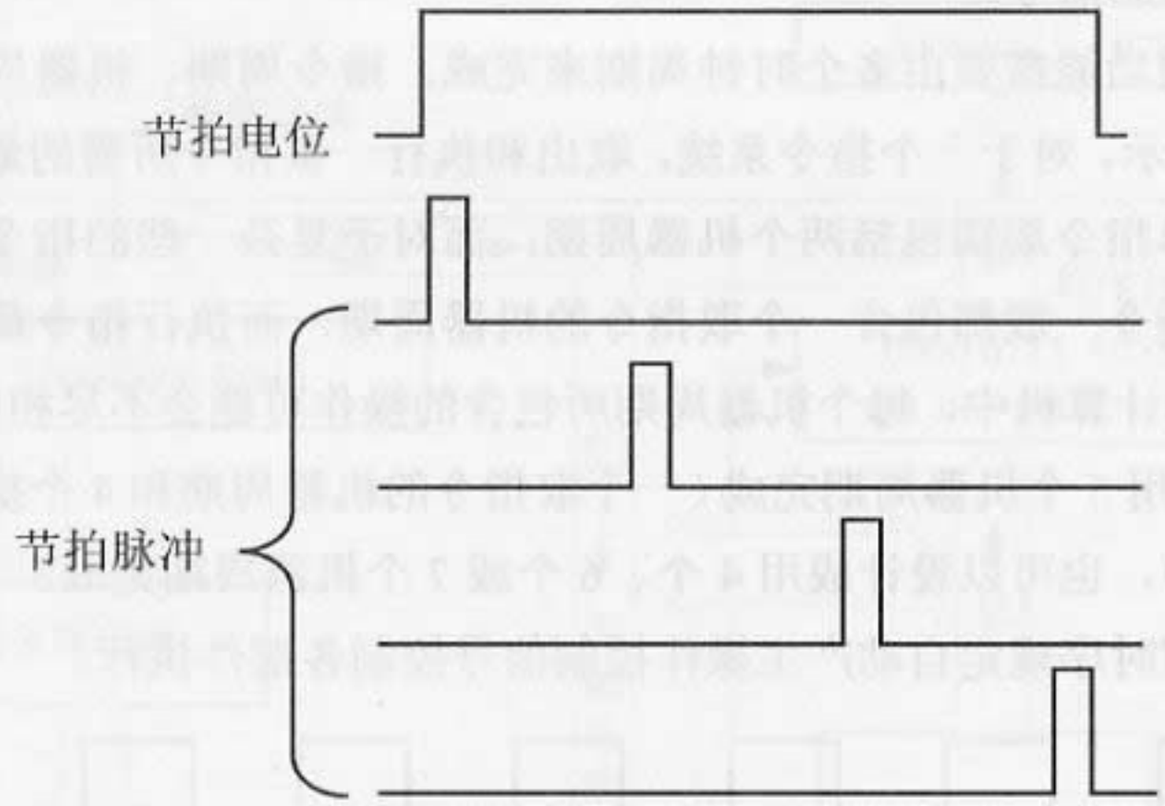


图7.7 节拍电位和节拍脉冲

2. 异步控制方式

每条指令需要多少节拍，就产生多少节拍；当指令执行完毕，发出回答信号；控制器收到回答信号时，才开始下条指令的执行。因此，执行不同指令所需的时间完全由实际需要确定，不尽相同。这种控制方式每个机器周期内的节拍数可以不一样，通常把大多数微操作安排在一个较短的机器周期内完成，而对某些复杂的微操作，采用延长机器周期或增加节拍数的办法来解决，如图 7.8 所示。

异步控制方式的优点是运行速度快，其缺点是控制电路比较复杂。

3. 联合控制方式

这种控制方式是把同步控制方式和异步控制方式结合使用的一种方式。大部分指令安排在统一的机器周期内完成，即同步控制；而将较少数特殊指令，或微操作序列过长或过短，或微操作时间难以确定的，采用异步控制来完成。

联合控制方式的优点是能保证一定的运行速度，其缺点是控制电路设计相对比较复杂。

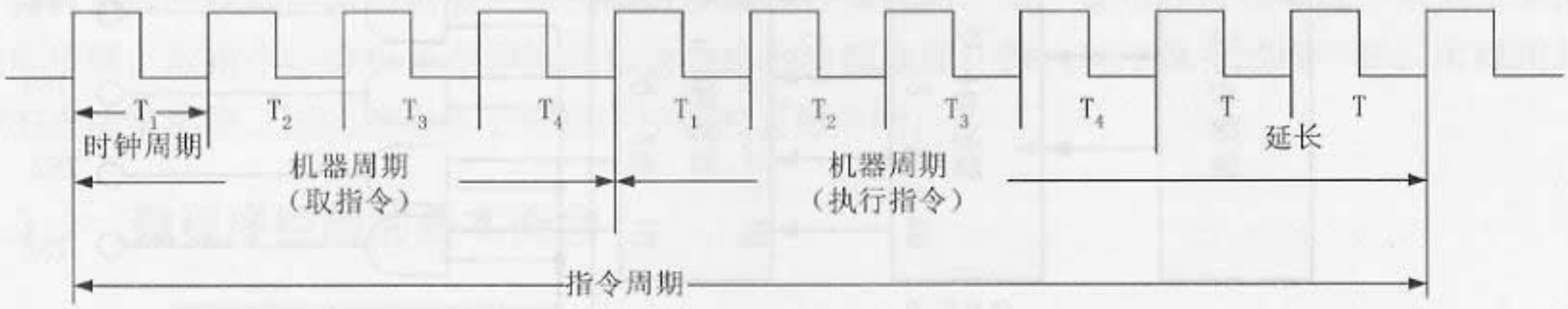


图7.8　延长机器周期的时序图

7.2.2　时序脉冲发生器和启停控制

由于一条指令规定的功能是分解成若干个微操作来实现的，不可能所有的微操作在同一时刻一起进行，它们是依次先后执行的，所以需要时序信号来控制各操作信号按时序关系依次有效，从而保证计算机各部件有节奏地依次执行规定的各种操作。

时序脉冲发生器就是根据时钟产生一定频率的节拍脉冲信号作为整个机器工作的时序信号；启停控制电路是保证在适当的时刻准确可靠地开启或封锁计算机工作时钟，以控制微操作命令序列的产生或停止，从而启动或停止计算机的运行。

一个指令周期由若干个机器周期组成。一般机器周期固定由几个节拍周期或时钟周期组成。通常用访问一次主存取指或取数据的时间来作为机器周期的基本时间，这样可以保证绝大部分指令的每一个操作都能在这一时间内完成。若个别操作不能完成的可采用增加机器周期或者采用应答方式来解决。

一般来说，机器周期基本上是根据存储器的速度及执行周期的基本时间确定的。机器周期确定后，每一机器周期的节拍与时钟数、机器的主频也就基本确定了。控制器的时钟输入实际上是节拍脉冲序列，其频率即为机器的主频。

图 7.9 为 Yy-z02 计算机组成原理实验仪上的时序信号发生及启停控制电路图。

它的基本原理是根据 555 时基集成电路产生的方波信号源Φ，经过消抖电路产生 4 个等间隔的节拍信号 TS1、TS2、TS3、TS4，并且受微动开关“START”和连续/单步开关“RUN#/STEP”的控制。当连续/单步开关“RUN#/STEP”=0（RUN）时，按动微动开关“START”，则产生连续的节拍信号TS1—TS4；当连续/单步开关“RUN#/STEP”=1（STEP）时，每按动微动开关“START”一次，则产生一组时序信号 TS1—TS4。

时序节拍信号 TS1—TS4 和时钟信号源Φ的关系图，如图 7.10 所示。

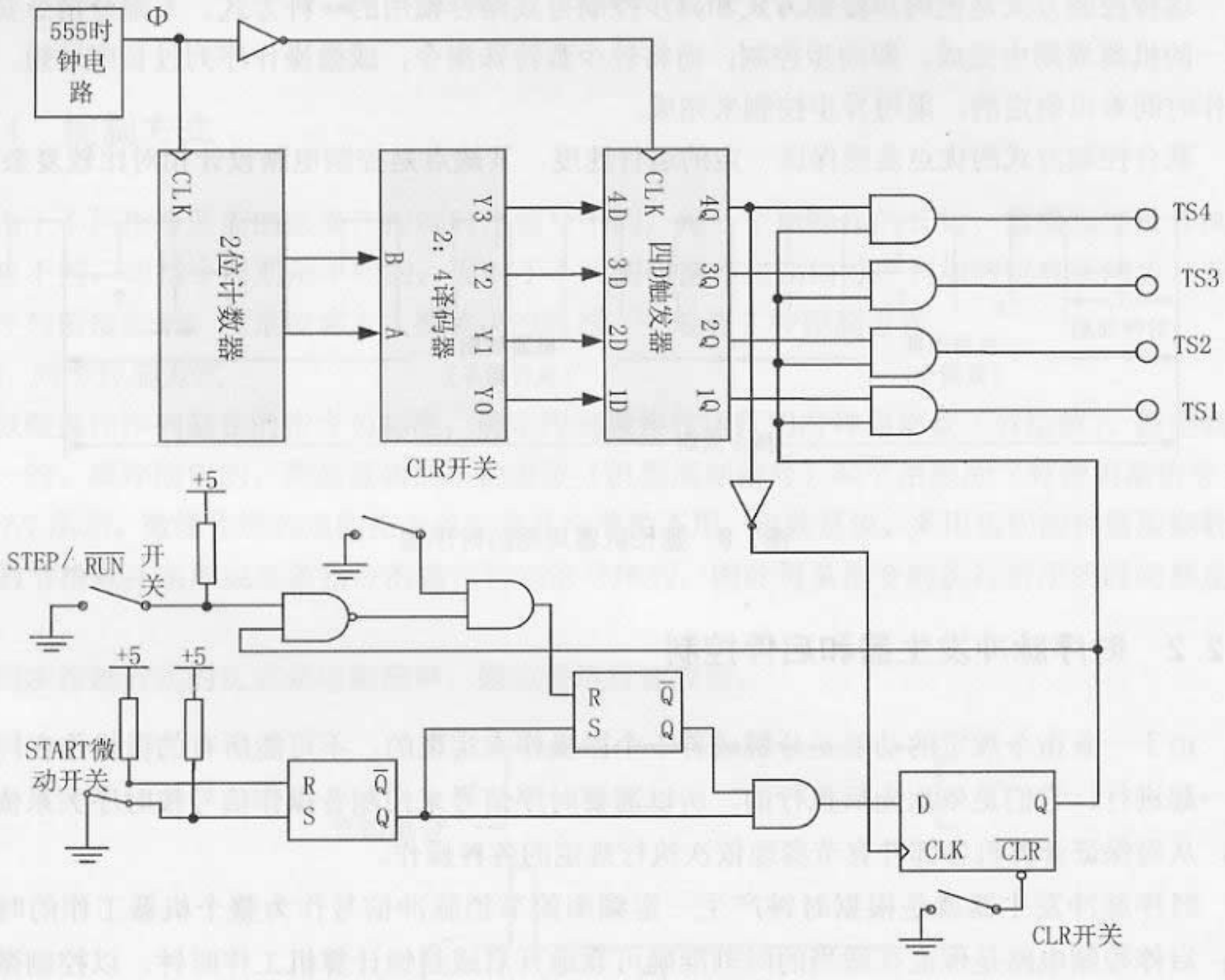


图7.9　时序信号发生及启停电路

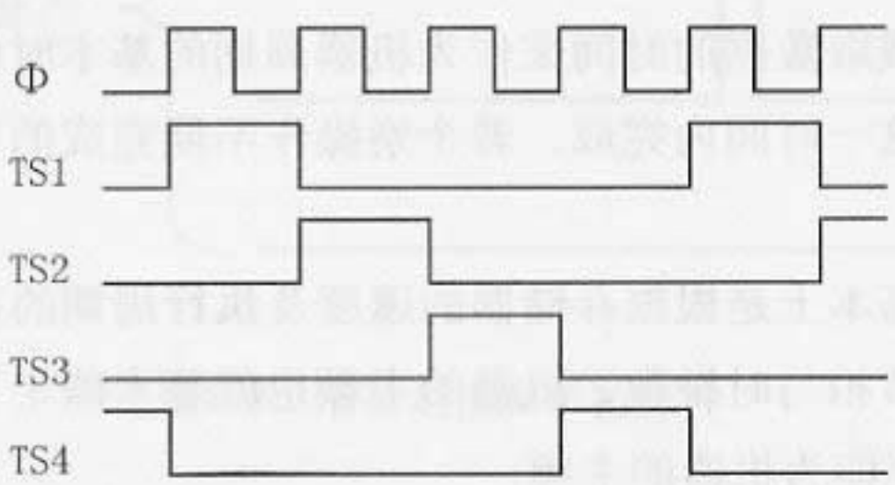


图7.10　时序信号波形图

7.3　微程序控制器

控制器根据其操作控制信号形成部件组成不同分为硬布线控制器和微程序控制器两种。在早期的计算机中，由硬布线即组合逻辑电路来实现指令译码和操作控制信号的形成，但这种组成方式不利于设计指令复杂的计算机系统。一台计算机指令系统越复杂，则其控制器的电路也

越复杂，而且不同的指令系统，其组合逻辑电路也不同。硬布线控制器电路的设计非常繁琐，很难实现设计自动化，一旦机器的硬布线控制器的组合逻辑电路设计完成后，很难修改和扩充其指令系统的功能。

为了解决以上问题，提出了用微程序控制来实现控制器电路；微程序控制器的基本思想是，计算机系统的控制电路是按照一系列离散步骤进行操作的。即一条指令可以分成一系列基本的操作步骤：取指令、分析指令和执行指令。这些步骤就像计算机程序执行过程一样，可以用软件的方法来实现，因此就形成了微程序控制的思想方法。

7.3.1　微程序控制的基本概念

微程序是实现一条机器指令功能的程序。微程序设计思想就是每条机器指令的功能都用一段相应的微程序来实现。微程序是由若干条微指令组成的。一条微指令包含若干个微命令，一个微命令就完成一个微操作。

微操作：指令执行时必须完成的基本操作。例如，任一条指令在取指令过程中的基本操作也即微操作有：PC→AR，PC+1→PC，RAM→IR。

微命令：是组成微指令的最小单位，也就是微操作的控制信号；一般都是数据通路上控制门的电位，触发器或寄存器的打入、置位、复位脉冲等。

微指令：是一组微命令信息，它存放在微程序控制器中的控制存储器（简称控存）内，其中一个存储单元的内容就是一条微指令。

微周期：执行一条微指令所需要的时间，一般可以作为一个机器周期。

微地址：微指令在控存中的地址。

微程序：是微指令的有序集合。一条机器指令由一段微程序来实现，微程序的控制方法也有微程序转移、微程序循环和、微子程序等。

7.3.2　微程序控制的基本原理

图 7.11 是 Yy-02 模型机逻辑框图，用它作为例子来说明微程序控制的工作原理。该模型机字长 8 位，即运算器、存储器、寄存器和数据总线均为 8 位。地址总线也是 8 位。指令系统，指令字长 8 位，其中基本操作码 4 位；当寄存器地址为两个时，指令操作码为 4 位，源寄存器地址 2 位，目标寄存器地址 2 位；当寄存器地址为单个时，指令操作码为 6 位，寄存器地址 2 位。这样就实现指令操作码的扩展，使指令条数可达 28 条。指令寻址方式有直接寻址、间接寻址、变址寻址和相对寻址。指令类型有传送指令、算术逻辑运算指令、条件转移和无条件转移指令、输入/输出指令、调用子程序指令、返回指令、停机指令。还可以根据硬件组成自己设计各种指令。主存按字节编址，地址线 8 根，可寻访 256 个字节单元；可在 RAM 中设置堆栈区，由堆栈指针 SP 指向。

通过下面几种操作，介绍该模型机的数据通路：

1. 存储器读操作

分成两步：

（1）送地址到总线，并打入地址寄存器 AR。

（2）发送存储器读信号 M-R# =0，启动存储器读操作，并将读出的数据从总线上接收至目的部件（例如某通用寄存器或者暂存器 DA1、DA2）。

例如取指令操作，第一步为 PC→AR，PC 自增 1；第二步，发送存储器读指令，并 RAM→IR。

2. 存储器写操作

分成两步：

（1）送地址到总线，并打入地址寄存器 AR。

（2）送数据到总线，并发送存储器写信号 M-W# =0，启动存储器写操作。

3. 运算器的运算操作

分成 3 步：

（1）送第一个数据到总线，并打入 ALU 暂存器 DA_1（或 DA_2）。

（2）送第二个数据到总线，且打入 ALU 暂存器 DA_2（或 DA_1）。

（3）发送运算器功能选择信号 S_3～S_0、M、Ci，控制 ALU 进行某种运算，并打开 ALU 输出三态门，将总线上运算结果送目的部件（例如某通用寄存器）。

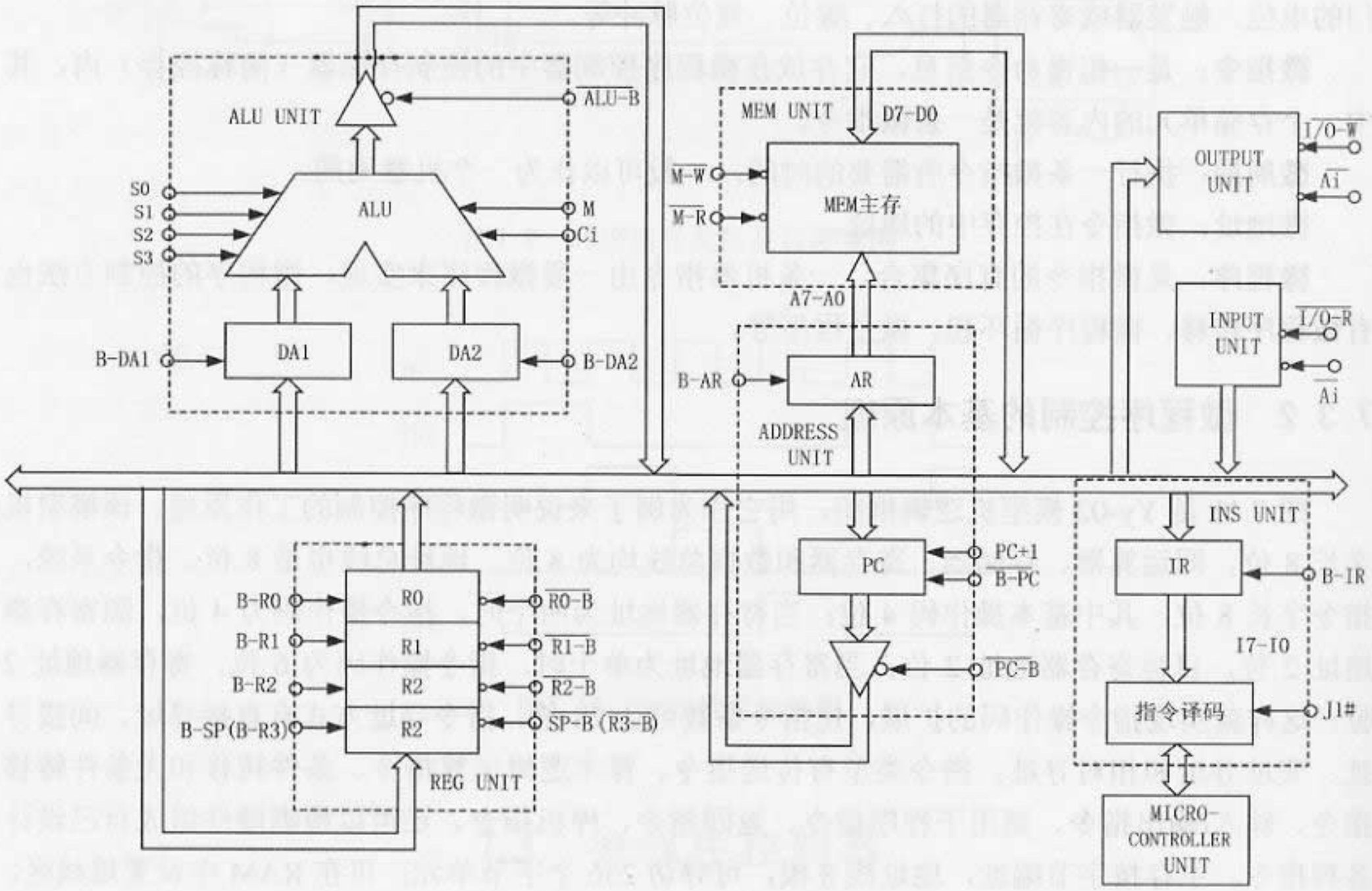


图 7.11　模型机逻辑框图

将图 7.11 的模型机中的控制信号（微命令）列成如表 7.2 所示。现以在该模型机中执行 7.1.2 节所述指令 ADD R_0，06H 为例，该指令是将 R_0 寄存器的内容加上立即数 06H，结果放在 R_0 寄

存器中；该指令的操作数为立即数，属于指令的一部分，存放在指令的第二个字中。根据模型机的数据通路，可以分成以下几条微指令来完成。

取 ADD 指令的微指令：

（1）PC-B＃、B-AR、PC+1 有效，将程序计数器的内容通过总线打入地址寄存器准备取指令，并将程序计数器加 1，为取下一条指令作准备。

（2）M-R#、B-IR 有效，从存储器中读出当前地址的指令，并将指令打入指令寄存器 IR；指令译码信号 J1＃有效，使指令译码器工作，转入当前指令的微程序入口。

表 7.2　模型机控制信号一览表

序号	控制信号	功　能	序号	控制信号	功　能
1	PC-B#	指令地址（程序计数器）送总线	15	ALU-B＃	运算器ALU内容送总线
2	B-AR	总线内容打入地址寄存器	16	Ci	ALU进位输入
3	PC+1	程序计数器内容加一	17	B-R0	总线内容打入R0寄存器
4	B-PC	总线内容打入程序计数器	18	B-R1	总线内容打入R1寄存器
5	B-IR	总线内容打入指令寄存器	19	B-R2	总线内容打入R2寄存器
6	M-W＃	存储器写	20	B-R3(B-SP)	总线内容打入R3寄存器
7	M-R＃	存储器读	21	R0-B＃	R0寄存器内容送总线
8	S_3	S_3- S_0选择ALU16种运算之一	22	R1-B＃	R1寄存器内容送总线
9	S_2	同上	23	R2-B＃	R2寄存器内容送总线
10	S_1	同上	24	R3-B＃（SP-B＃）	R4寄存器内容送总线
11	S_0	同上	25	I/O-W＃	写（输出）I/O端口
12	M	M为“1”选择ALU做逻辑运算，M为“0”选择ALU做算术运算	26	I/O-R＃	读（输入）I/O端口
13	B-DA1	总线内容打入暂存器DA1	27	Ai＃	端口地址线
14	B-DA2	总线内容打入暂存器DA2	28	J1#	指令译码器工作

执行 ADD 指令的微指令：

（1）PC-B、B-AR、PC+1 有效，将程序计数器的内容通过总线打入地址寄存器，准备取指令的第二个字即立即数，并将程序计数器加 1，为取下一条指令做准备。

（2）M-R＃、B-DA1 有效，从存储器中读出立即数通过总线打入暂存器 DA1。

（3）R0-B#、B-DA2 有效，将存放在 R0 寄存器中的数据通过总线打入暂存器 DA2。

（4）M=0，S_3～S_0为 1001 表示 ALU 执行算术加法运算，Ci＝1 表示 ALU 无进位输入，ALU-B#、B-R0 有效，本条微指令实现 DA1 和 DA2 的数据在 ALU 中执行加法，并将结果通过总线打入寄存器 R0。

假设该机所有的控制信号为 28 个，用微指令的一位表示一个微命令（控制信号），该位为 1 表示该微命令即控制信号有效，该微命令发到对应单元控制门即实现该微操作。因此模型机的微指令的控制字段需要 28 位。其中 S_3～S_0、M 和 Ci 信号是用编码来控制 ALU 执行相应的运算。微指令的下址字段是指出下一条微指令的地址，该模型机的控制存储器地址是 7 位，表示最多有 2^7 个单元，每个单元内容的位数即一条微指令字长，也就是最多可存放 128 条微指令。因此微指令的下址字段为 7 位。微指令格式如图 7.12 所示。

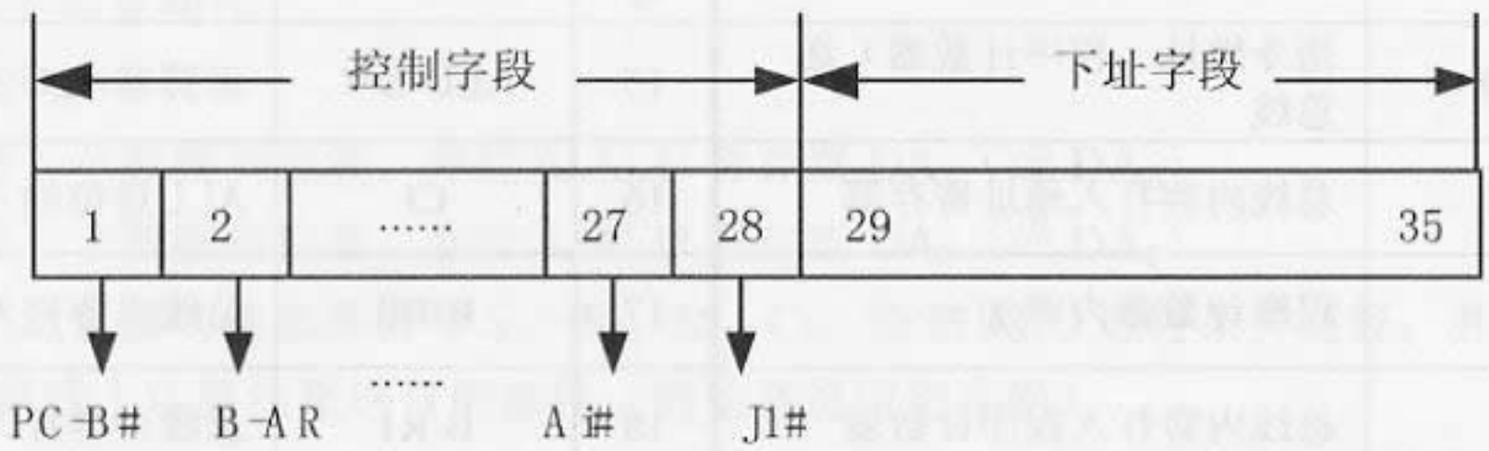


图7.12　微指令格式

控制字段每一位微命令的功能与表 7.2 中所表示的一致。例如第 1 位表示 PC-B#，第 28 位表示 J1#，#表示低电平有效，等同于上划线。根据微指令格式可将解释执行 ADD R_0, 06H 指令的微程序中 6 条微指令表示如图 7.13 所示。

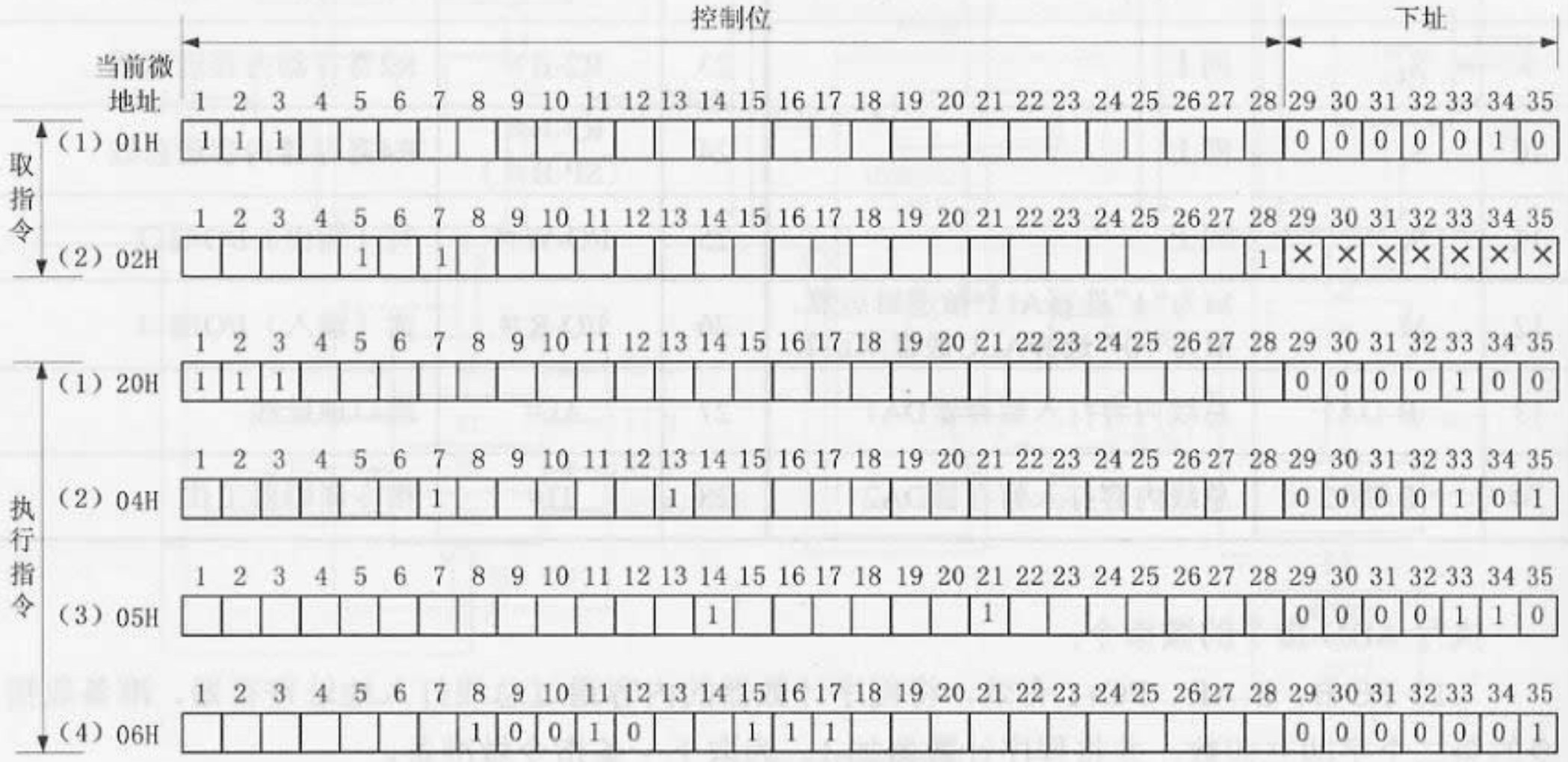


图7.13　ADD指令的微指令

假设取指令的第（1）条微指令的地址是 01H，则第（2）条微指令的地址依次为 02H。这 2 条微指令对于该机所有指令都是必须有的，因此是公共的微指令。取指译码后转入执行指令的微程序，每一条指令的微程序都不一样，因此指令译码的功能就是根据该指令的操作码产生该指令的第一条微指令的微地址（即指令的微程序入口），这样取指令后就转入该指令对应的微程序去执行。这里，假设 ADD R_0，06H 指令译码后产生的微程序入口即第（1）条微指令地址为 20H，则接下去的第（2）、第（3）、第（4）条微指令的微地址就由当前微指令中的下址字段来决定，分别依次设为 04H、05H、06H。

现在我们再来看看转移指令 JMP 04H 的微程序实现，该指令是将程序转移到转移地址的指令去执行。转移地址在内存中指令的第二个字。

首先也是取指令的 2 条微指令，与上述 ADD 指令的取指令微指令相同。

取 JMP 指令的微指令：

（1）PC-B＃、B-AR、PC+1 有效，将程序计数器的内容通过总线打入地址寄存器，准备取指令；并将程序计数器加 1，为取下一条指令作准备。

（2）M-R#、B-IR 有效，从存储器中读出当前地址的指令，并将指令打入指令寄存器 IR；指令译码信号 J1＃有效，使指令译码器工作，转入当前指令的微程序入口。

执行 JMP 指令的微指令：

（1）PC-B、B-AR、PC+1 有效，将程序计数器的内容通过总线打入地址寄存器，准备取指令的第二个字即转移地址，并将程序计数器加 1，为取下一条指令做准备。

（2）M-R#、B-PC 有效，从存储器中读出转移地址，并通过总线打入程序计数器 PC，即实现程序转移。

将 JMP 指令的 4 条微指令列出如图 7.14，假设 JMP 指令取指后的第一条微指令地址，即 JMP 指令的微程序入口为 21H。

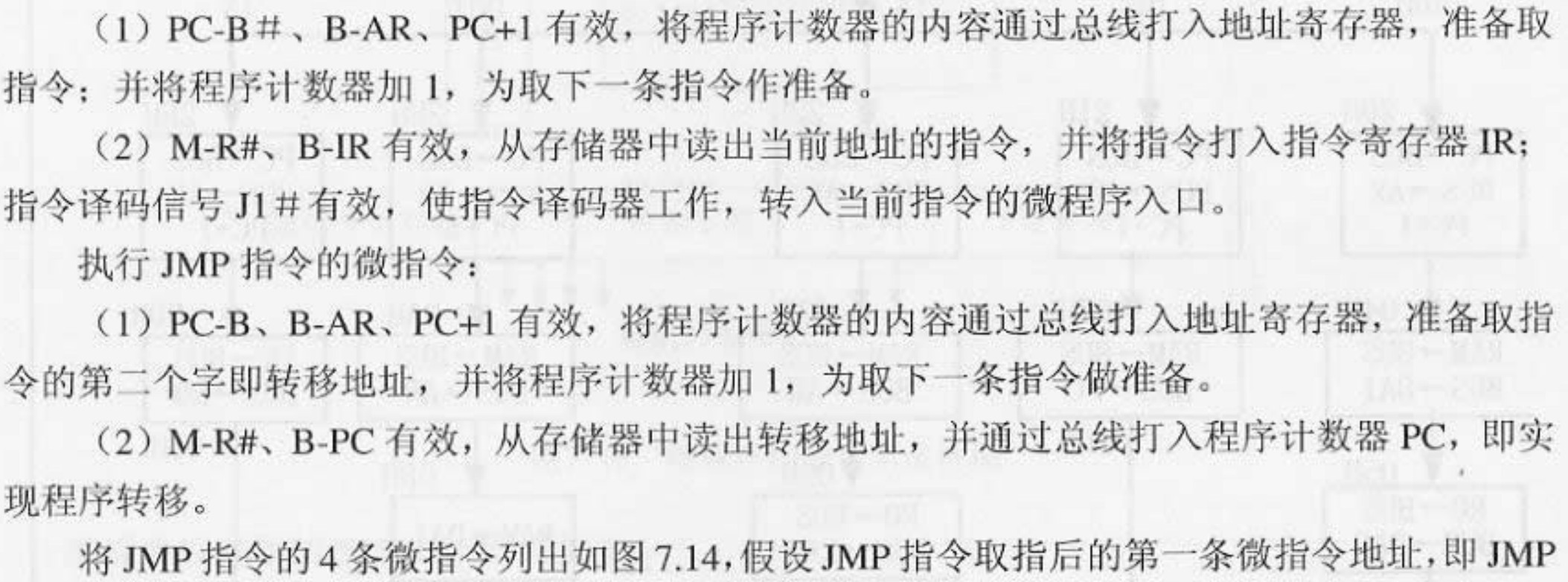


图 7.14　JMP 指令的微指令

微程序也可以用流程图来表示，图 7.15 就是模型机指令系统简化的微程序流程图。图中每一个框表示一条微指令所要实现的操作，框的右上方表示该微指令的微地址，框的右下方表示该微指令的下址字段，框中的内容表示该微指令所执行的微操作。每条指令的最后一条微指令

执行完后，如果没有硬件中断请求，则都转入取指令的第一条微指令，又开始取下一条指令。

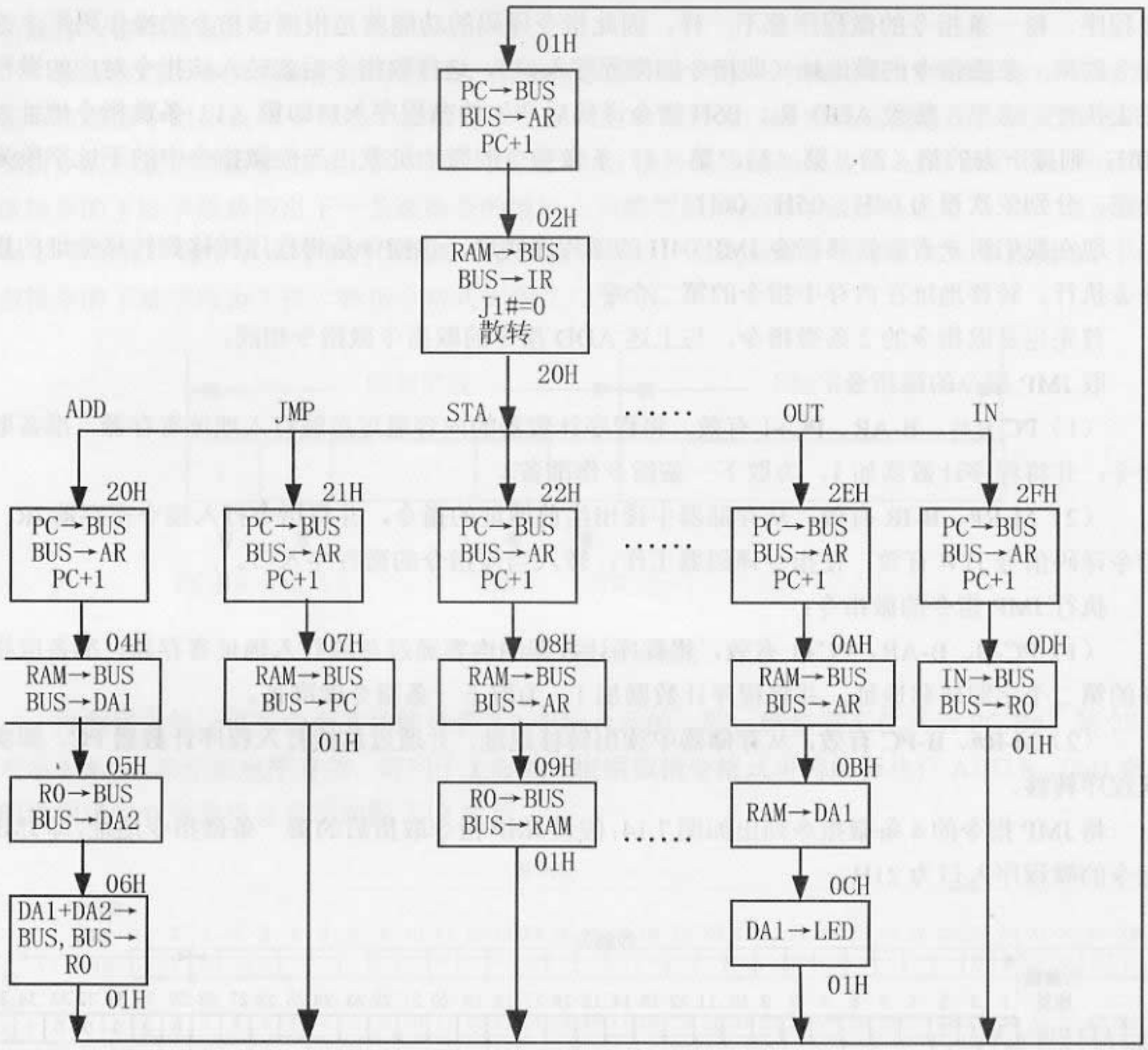


图7.15　简化微程序流程图

7.3.3　微程序控制器的组成

微程序控制器的组成框图如图 7.16 所示。图中微地址形成电路就是指令译码器，其作用是将指令寄存器中的操作码 OP 转换成该指令的微程序入口地址，用该微程序入口地址可以从控制存储器中取出该指令的第一条微指令（执行指令的微指令）。图中的微地址寄存器、控制存储器和微指令寄存器取代了图 7.1 中的操作控制信号形成部件，这就是微程序控制的控制器的组成。

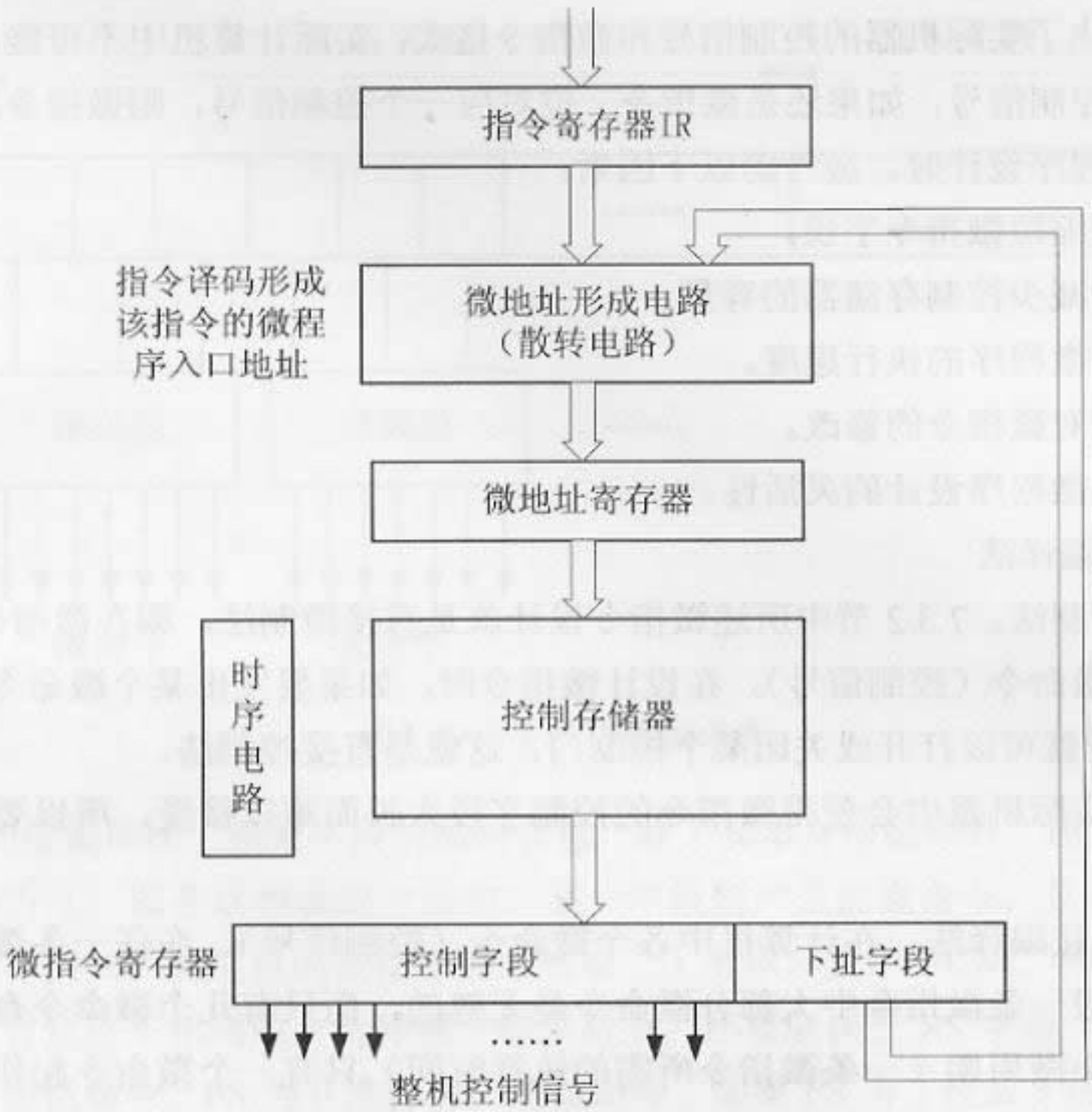


图7.16　微程序控制器组成框图

微程序控制器的基本工作原理如下：

开机后首先使微地址寄存器置为取指令的第一条微指令地址，从控制存储器中取出第一条微指令，完成 PC→AR、PC+1 操作，然后根据微指令的下址字段取出第二条微指令，完成 RAM→IR 从内存中读出指令送指令寄存器 IR，并且发 J1#有效信号使指令译码器工作，即形成该指令的执行指令阶段的微程序入口地址；从控存中取出该指令执行时的第一条微指令送到微指令寄存器，发出控制信号（微命令）实现微操作；然后该指令执行时的其余微指令地址由当前微指令的下址字段确定，依次从控存中取出其余微指令，实现该指令所需的所有微操作，即完成了该指令的执行。每一条指令的最后一条微指令执行完后均会回到取指令的第一条微指令执行，以取下一条指令，如此重复，直至用户要运行的程序指令执行完为止。

在微程序控制器中，时序信号一般采用节拍电位——节拍脉冲二级体制，节拍电位表示一个机器周期（或 CPU 周期），而节拍电位中包含若干个节拍脉冲（即时钟周期），节拍脉冲将机器周期划分成几个较小的时间间隔，使控制信号在各个时间中有效，以保证信息在数据通路中准确顺畅地流通。

7.3.4　微程序设计技术

微程序控制器的设计和微程序的设计是相互依赖的。我们在分析微程序基本原理时，为了

有助于理解而简化了实际机器的控制信号和微指令格式，实际计算机中不可能只有 28 个控制信号，而是上百个控制信号，如果还是微指令一位对应一个控制信号，则微指令的控制字段太长。因此实际进行微程序设计时，应考虑以下因素：

（1）有利于缩短微指令字长。

（2）有利于减少控制存储器的容量。

（3）有利于微程序的执行速度。

（4）有利于对微指令的修改。

（5）有利于微程序设计的灵活性。

1. 微指令的编译法

（1）直接控制法。7.3.2 节中所述微指令设计就是直接控制法，即在微指令的控制字段中，每一位代表一个微命令（控制信号），在设计微指令时，如果要发出某个微命令则将控制字段中对应位置 1，这样就可以打开或关闭某个控制门，这就是直接控制法。

这种方法在实际机器中会使得微指令的控制字段太长而难以接受，所以要采取其他方法来弥补这种缺陷。

（2）字段直接编译法。在计算机中各个微命令（控制信号），在任一条微指令周期内不可能同时发出，一般一条微指令中大部分微命令是无效的，而只有几个微命令有效。如果有一组微命令，在任一个微周期（一条微指令所需的执行时间）只有一个微命令起作用，那么这一组微命令是互斥的；通常将在同一个微周期中不能同时出现的微命令称为相斥性微命令，而将在同一个微周期中可以同时出现的微命令称为相容性微命令。

例如在 Yy-z02 模型机中，总线上的许多部件往总线送信息的控制信号，一次只能一个有效，不能两个以上有效，否则会损坏芯片电路；再如，总线的信息打入各个部件的控制脉冲信号一次也只能一个有效，这些信号相互都是互斥的。我们将这些互斥的微命令编成一组，用编码来表示一个微命令。例如，将 7 个互斥的微命令编成一组，用 3 位二进制编码分别表示每个微命令，则微指令的该控制字段就由原来的 7 位减少为 3 位，从而缩短了微指令长度。这样微指令寄存器中这 3 位输出端加一个 3—8 译码器，该译码器的输出就是这一组微命令，如图 7.17 所示。一般 3 位二进制编码最多只能表示 7 个微命令，因为还要留一个编码（通常为 000 编码）表示这一组微命令没有一个有效。

字段直接编译法的分段原则是：

①相斥性微命令分在同一字段内，相容性微命令分在不同字段内。前者可以提高信息位的利用率，缩短微指令字长；后者有利于实现并行操作，加快指令的执行速度。

②一般将同类操作中互斥的微命令划分在一个字段内，如将控制 ALU 操作的微命令 S_3～S_0 等划分在一个字段内，将控制到总线信息的微命令划分在另一个字段内。这样使微指令结构清晰，易于编制微程序和易于扩充功能。

③每字段的信息位不能太长，一般不超过 6 位，否则将增加译码线路的复杂性和译码时间。

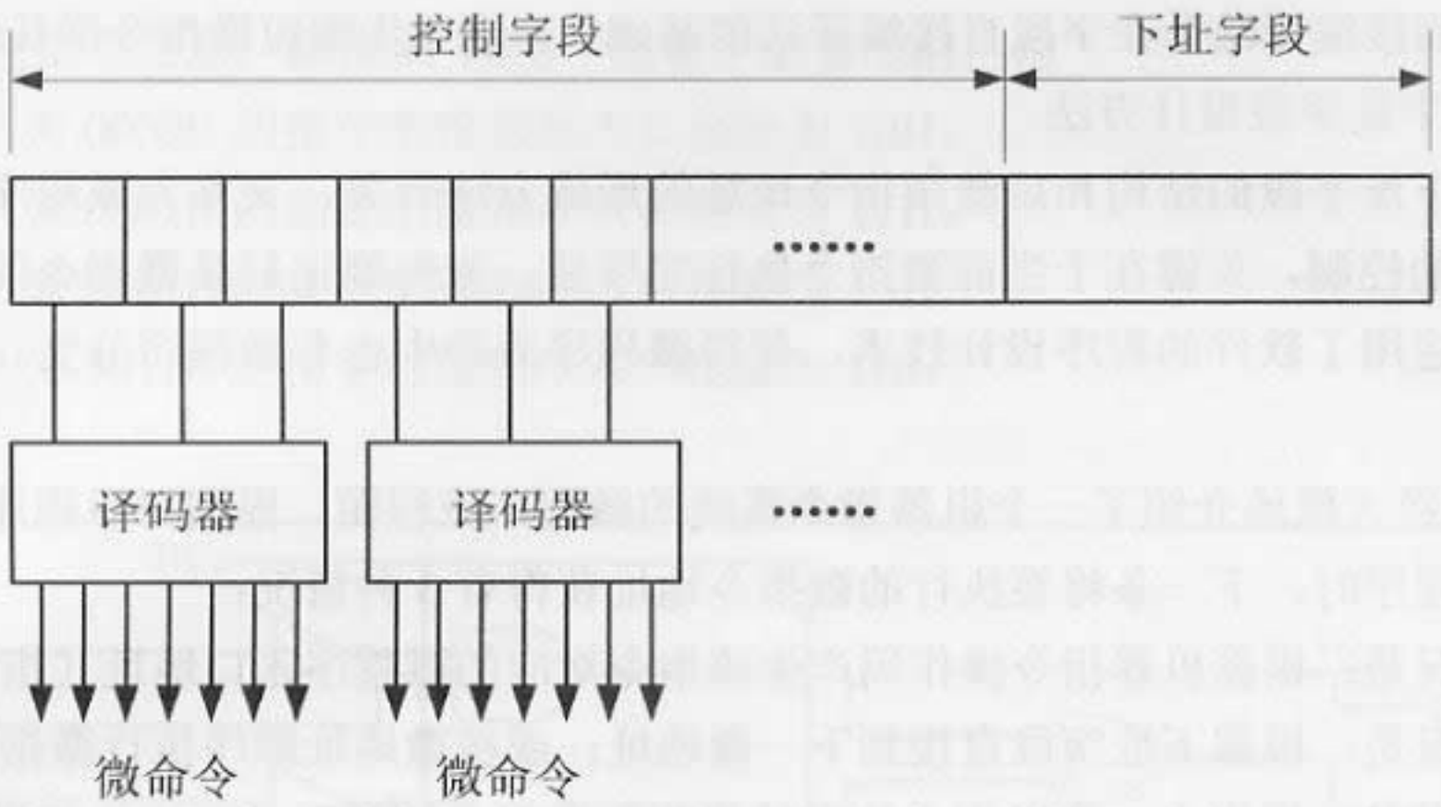


图7.17　字段直接编译法

（3）字段间接编译法。指某字段的编码含意，除了其本身的编码外，还需要由另一字段来加以解释的编码方法，即在这种编码方法中，某一字段所产生的微命令，是和另一字段的代码联合定义出来的。例如，模型机的微指令格式如表 7.3 所示，其中 FUNC 字段有两组微命令，这两组微命令的含意是不相同的，选择哪一组微命令，还要取决于 FS 字段（这里是一位），FS 为 1 选择其中一组微命令，FS 为 0 选择另一组微命令。如果 FS 为 2 位或 3 位，则在 FS 字段输出端还要增加译码器，这样可使 FUNC 字段增加为 4 组或 8 组微命令。

表 7.3　模型机微指令格式

M23M22M21	M20M19M18	M17 M16 M15	M14	M13	M12	M11	M10	M9	M8	M7	M6M5M4M3M2M1M0
BTO(3)	OTB(3)	FUNC(3)	FS	3	S2	S1	S0	M	Ci	空	MA6-MA0(7)

微指令字长 24 位，记为 M_{23}～M_0，各控制字段排列及编码如表 7.4 所示。

表 7.4　微指令字段编码表

编码+译码	BTO	OTB	FS=1	FS=0
			FUNC	FUNC
000			/PC+1	
001	B-DA1(t4)	ALU-B# (t2)	J 1# (t2)	M-W# (t3)
010	B-DA2(t4)	299-B# (t2)	J 2# (t2)	M_R# (t2)
011	B-IR(t3)	SR-B# (t2)	J 3# (t2)	I/O-W# (t3)
100	B-DR(t4)	DR-B# (t2)	J 4# (t2)	I/O_R# (t2)
101	B-SP(t4)	SI-B# (t2)*	J5# (t2)	INT_R# (t2)
110	B-AR(t3)	SP-B# (t2)*	CyCn# (t2)	INT_E# (t2)
111	B-PC# (t4)	PC-B# (t2)	CyNCn# (t2)	

这种字段间接编译法是在字段直接编译法的基础上，进一步缩短微指令字长的一种编译法。

2. 微指令下址字段设计方法

微指令中下址字段的结构和后继微指令地址的形成方法有关，又称为微程序流的控制。要解决微程序流的控制，关键在于当前微指令执行完毕后，如何确定后续微指令的地址。在微程序设计中充分运用了软件的程序设计技术，使得微程序流程中也有微程序分支、微程序循环、微子程序等。

7.3.2 节已经大概地介绍了一个机器指令系统的微程序流程图。根据对微程序流程的分析，得出在执行微程序时，下一条将要执行的微指令地址获得有 3 种情况：

第一种情况是：根据机器指令操作码产生该指令对应的微程序入口地址（指令译码）。

第二种情况是：根据下址字段直接到下一微地址；或按微地址顺序执行微指令。

第三种情况是：根据上一条微指令执行结果判断微指令转移还是顺序执行微指令，即实现微程序分支。

下面从两个方面讨论微指令地址的确定方法，第一个方面是根据机器指令操作码产生对应微程序入口地址，第二个方面就是上述第二种、第三种情况的后继微地址的产生。

（1）微程序入口地址的产生。我们已经知道，每一条机器指令都对应一段微程序，首先由“取机器指令”的微程序段完成将一条机器指令从主存中取出并送到指令寄存器 IR。这段微程序是所有指令公用的，一般安排在控制存储器的特定单元开始，取出机器指令后应根据机器指令的操作码转移到其对应的微程序入口地址。这是一种多分支情况，也就是指令译码，通常采用逻辑电路、PROM 来实现这种转移。

现以 Yy-z02 组成原理实验系统模型机实例来说明采用逻辑电路实现指令译码（即散转到微程序入口）的方法，如图 7.18 所示，它由一片 GAL20V8 实现，该电路是微程序的后继微地址的转移控制逻辑电路，其主要功能是根据输入的指令操作码 I7—I2、微程序的转移方式 J1—J5、运算结果的状态 FC 和 FZ、控制台开关 K1 和 K2 的状态，进行逻辑译码，产生后继微地址的控制信号 SE6#—SE0#，以控制实现机器指令转入微程序入口和微程序的顺序、分支、循环运行。其输入信号为：指令码 I7—I2（来自指令寄存器 IR）、微程序的转移方式 J1—J5（来自微指令寄存器输出的微命令）、运算结果的状态 FC 和 FZ（来自运算器单元 ALU UNIT）、控制台开关 K1K2 的状态（来自手动单元 MANUAL UNIT）；其输出信号为 SE6#—SE0#，送至微地址寄存器。

其中，SE6#—SE0#用于控制修改后继微地址的相应位 MA6—MA0，当 SEi# =0，则微地址 MA_i 被修改（置 1），否则，MA_i 不变，与原微指令寄存器输出的下址字段相同。

由此，可将微程序的后继微地址的控制原理总结如下：

①J1# =0 时，根据机器指令的操作码（OP）I7—I2 进行散转，产生该条指令的微程序入口地址，散转微地址确定如下：

- 若 I7=1 且 I6=1，则 1→MA5，否则 MA5 不变。
- 若 I7 I6= 10 或 I7 I6 I5=111，则 1→MA3，否则 MA3 不变。
- 若 I7 I6= 01 或 I7 I6 I4= 111，则 1→MA2，否则 MA2 不变。
- 若 I7 I6 I5=001 或 011 或 101，或 I7 I6 I3 = 111，则 1→MA1，否则 MA1 不变。
- 若 I7 I6 I4 = 001 或 011 或 101，或 I7 I6 I2 = 111，则 1→MA0，否则 MA0 不变。

例如：取完指令之后，根据 J1 散转，假设下址为 10H，则

操作码 OP 为 0000B 的指令的微程序入口地址为 10H。

操作码 OP 为 0001B 的指令的微程序入口地址为 11H。

操作码 OP 为 0101B 的指令的微程序入口地址为 15H。

操作码 OP 为 1011B 的指令的微程序入口地址为 1BH。

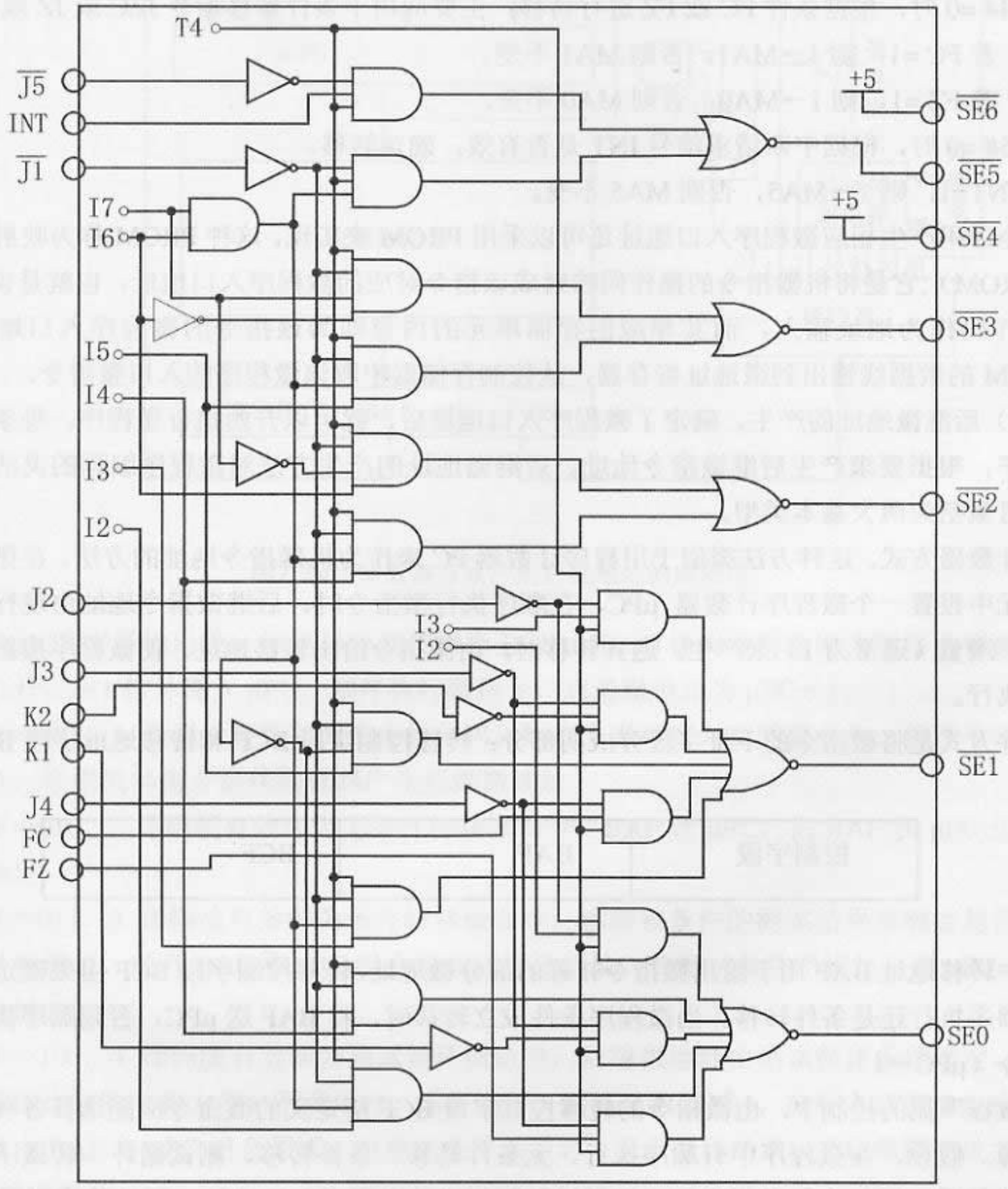


图7.18　指令译码即后继微地址的转移控制逻辑

②J2# =0 时，根据机器指令码 I3I2 进行译码，转移至相应指令的微程序段，主要应用于含寻址方式码（MOD）的机器指令，在这些指令中，I5I4 为寻址方式码（MOD），I7I6 和 I3I2 为

操作码 OP：

- 若 I3=1，则 1→MA1，否则 MA1 不变。
- 若 I2=1，则 1→MA0，否则 MA0 不变。

③J3# =0 时，根据开关 K2、K1 状态进行转移；主要应用于控制台操作指令。

- 若 K2 =0，则 1→MA1，否则 MA1 不变。
- 若 K1 =0，则 1→MA0，否则 MA0 不变。

④J4# =0 时，根据条件 FC 或 FZ 进行转移；主要应用于条件转移指令 JZC 或 JZ 或 JC。

- 若 FC =1，则 1→MA1，否则 MA1 不变。
- 若 FZ =1，则 1→MA0，否则 MA0 不变。

⑤J5# =0 时，根据中断请求信号 INT 是否有效，确定转移。

若 INT=1，则 1→MA5，否则 MA5 不变。

指令译码产生相应微程序入口地址还可以采用 PROM 来实现，这种 PROM 称为映射存储器（MAPROM）,它是将机器指令的操作码映射成该指令对应的微程序入口地址，也就是说，将指令的操作码作为地址输入，而其相应的存储单元的内容即为该指令的微程序入口地址，从 MAPROM 的数据线输出到微地址寄存器，从控制存储器中取该微程序的入口微指令。

（2）后继微地址的产生。确定了微程序入口地址后，就可以开始执行微程序。每条微指令执行完毕，根据要求产生后继微指令地址。后继微地址的产生方法对微程序编程的灵活性影响很大，可概括为两大基本类型。

①计数器方式。这种方法类似于用程序计数器 PC 来作为机器指令地址的方法。在微程序控制器单元中设置一个微程序计数器 μPC，在顺序执行微指令时，后继微指令地址由现行微地址加上一个增量（通常为 1）来产生。遇到转移时，由微指令给出转移地址，使微程序按新的微地址顺序执行。

这种方式是将微指令的下址字段分成两部分：转移控制字段 BCF 和转移地址字段 BAF，格式如下：

控制字段	BAF	BCF

其中转移地址 BAF 用于给出微指令转移的部分微地址，转移控制字段 BCF 用来确定后继微地址是顺序执行还是条件转移。当微程序条件成立转移时，将 BAF 送 μPC，否则顺序执行下一条微指令（μPC+1）。

在微程序流的控制中，由微指令的转移控制字段 BCF 所定义的微命令应能选择各种后继微地址来源。假设，在微程序中有顺序执行、无条件转移、条件转移、测试循环、转微子程序和返回 6 种情况，再加上每条指令取出后要转移到相应的微程序入口地址，则转移控制字段 BCF 用 3 位。转移地址字段 BAF 一般位数少，将它送到 μPC 的若干低位或高位形成后继微地址。图 7.19 为采用计数器方式产生微指令后继地址的原理图。

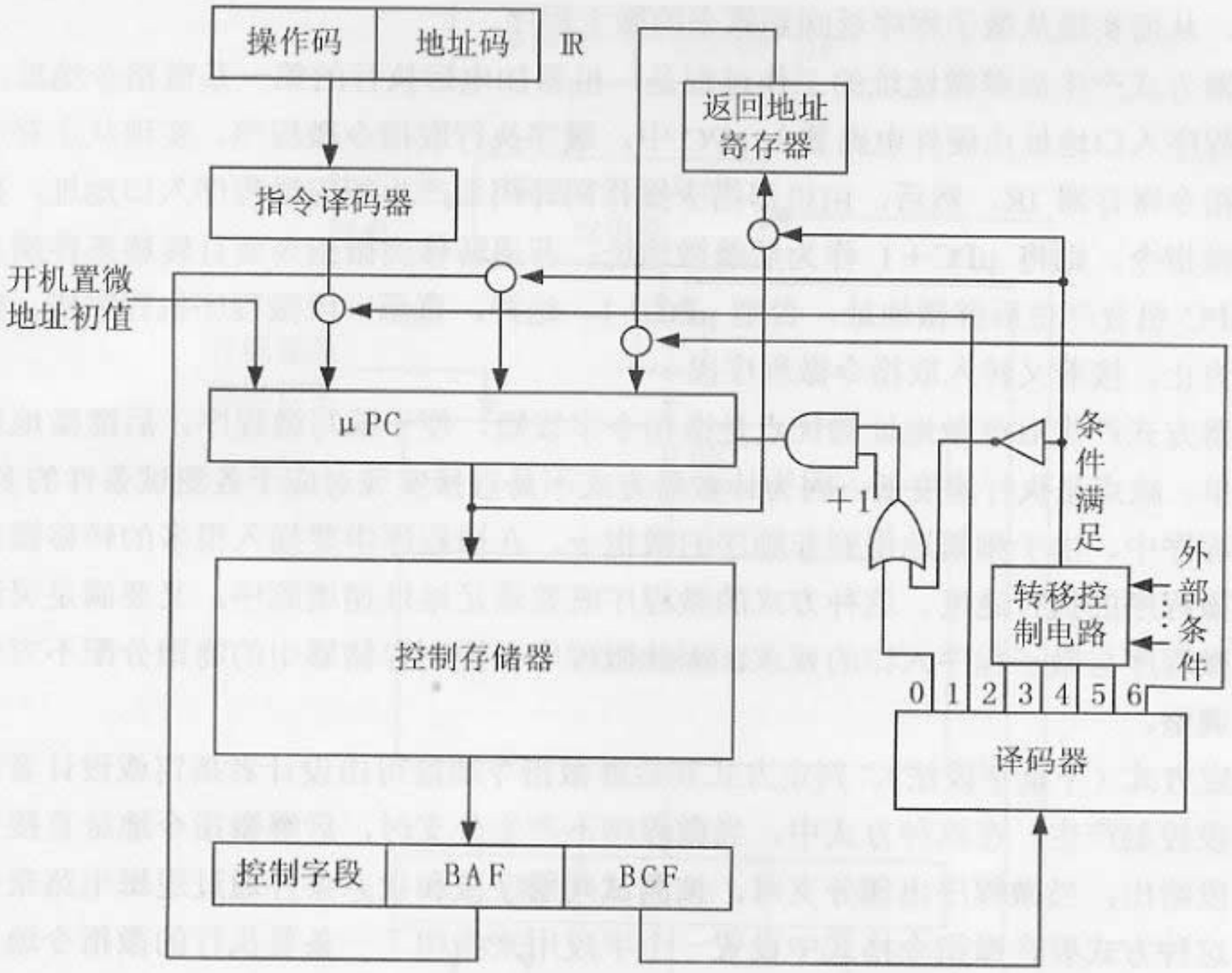


图7.19　计数器方式产生后继地址的原理图

图中 BCF 字段输出设一译码器分别使得当 BCF=000，0 译码线有效即为顺序执行微命令，使与门打开，＋1 信号进入 μPC，顺序执行微指令，后继微地址为 μPC＋1；

BCF＝001，1 译码线有效即为指令操作码译码产生微程序入口的微命令，将指令译码器输出门打开，根据机器指令操作码译码产生后继微地址。

BCF＝010，2 译码线有效即为无条件转移微命令，BAF 送 μPC，由 BAF 和 μPC 组合形成后继微地址。

BCF＝011，3 译码线有效即为条件转移微命令，由转移条件的测试结果来确定是否转移。若测试条件满足，则有 BAF 和 μPC 组合形成后继微地址；若测试条件不满足，则 μPC＋1 作为后继微地址。

BCF＝100，4 译码线有效即为测试循环微命令，后继微地址由测试循环条件确定。当测试条件（循环计数器内容为零）不满足时，循环继续，循环计数器减 1，循环入口微地址也就是后继微地址由 BAF 与 μPC 组合形成；当测试条件满足时，表示循环结束，其后继微地址为 μPC＋1，以此跳出循环。

BCF＝101，5 译码线有效即转微子程序微命令，其后继微地址即微子程序入口微地址由 BAF 与 μPC 组合形成。在转移之前，要把该条微指令的下一微地址 μPC＋1 送入返回地址寄存器中，以备返回微主程序。

BCF＝110，6 译码线有效即返回微命令，其后继微地址为返回地址寄存器中的内容，将其

送入 μPC，从而实现从微子程序返回到原来的微主程序。

计数器方式产生后继微地址的工作过程是，机器加电后执行的第一条微指令地址，也就是取指令微程序入口地址由硬件电路置入 μPC 中，顺序执行取指令微程序，实现从主存中取出机器指令送指令寄存器 IR，然后，由机器指令操作码译码后产生相应微程序入口地址，接下去若顺序执行微指令，则将 μPC＋1 作为后继微地址；若遇转移类微指令而且转移条件满足，则由 BAF 和 μPC 组合产生后继微地址，否则 μPC＋1。这样，直至一段微程序执行完毕，完成一条机器指令为止。接着又转入取指令微程序段……

计数器方式产生后继微地址的优点是微指令字较短，便于编写微程序，后继微地址产生机构比较简单；缺点是执行速度低，因为计数器方式不易直接实现对应于各测试条件的多路转移，在编写微程序中，由于频繁地用到非顺序的微指令，在微程序中要插入很多的转移微指令，这样会影响微程序的执行速度。这种方式的微程序既要满足地址递增顺序，又要满足灵活地转向各种共用微程序与微子程序入口的要求，因此微程序在控制存储器中的物理分配不方便，需合理安排、调整。

②判定方式（下址字段法）。判定方式其后继微指令地址可由设计者指定或设计者指定的测试判断字段控制产生。在这种方式中，当微程序不产生分支时，后继微指令地址直接由微指令的下址字段给出；当微程序出现分支时，按测试判别字段和状态条件通过逻辑电路来形成后继微地址。这种方式要在微指令格式中设置一个字段用来指明下一条要执行的微指令地址，所以也称为下址字段法。而且，在这种方式中，因为每一条微指令至少都是一条无条件转移微指令，因此不必设置专门的转移微指令。判定方式的原理图如图 7.20 所示。

图中可看出，μAR 代替了 μPC；μAR 的内容可以由微指令的下址字段来装入，也可以由微地址散转及修改电路来装入，微地址散转及修改电路包括由机器指令操作码产生对应微程序入口地址的逻辑电路和条件转移时的微地址修改逻辑电路。当微程序无分支时，后继微指令地址由微指令的下址字段直接给出，这是在设计微程序流程图时由设计者指定。当微程序出现分支时，由微指令的测试字段信号启动微地址修改逻辑电路，根据状态标志位来修改 μAR 的若干位来产生后继微地址，使微程序转移到不同地方去执行，实例电路如图 7.18 所示，图中 FC、FZ 为状态标志，当 FC 或 FZ 为“1”时，使 SE1#或 SE0#有效，从而修改微地址的最低两位 MA1 或 MA0。测试判别字段的位数决定于测试源的个数，微指令中可以用直接控制法表示即一位表示一个测试源，也可用译码法将若干个测试源组合用几位表示，如若测试源少于 4 个，则测试字段用 2 位；若测试源少于 8 个，则测试字段用 3 位，并经过译码后一次只有一个测试源有效。需要修改的 μAR 的位数取决于转移分支的路数。

图 7.20 中下址字段的位数取决于控制存储器的容量，与 μAR 的位数相等。

判定方式的优点是可以实现快速多路分支，以提高微程序的执行速度，微程序在控制存储器的物理分配方便，微程序设计灵活；缺点是微指令字长加长，形成后继微地址的结构比较复杂。

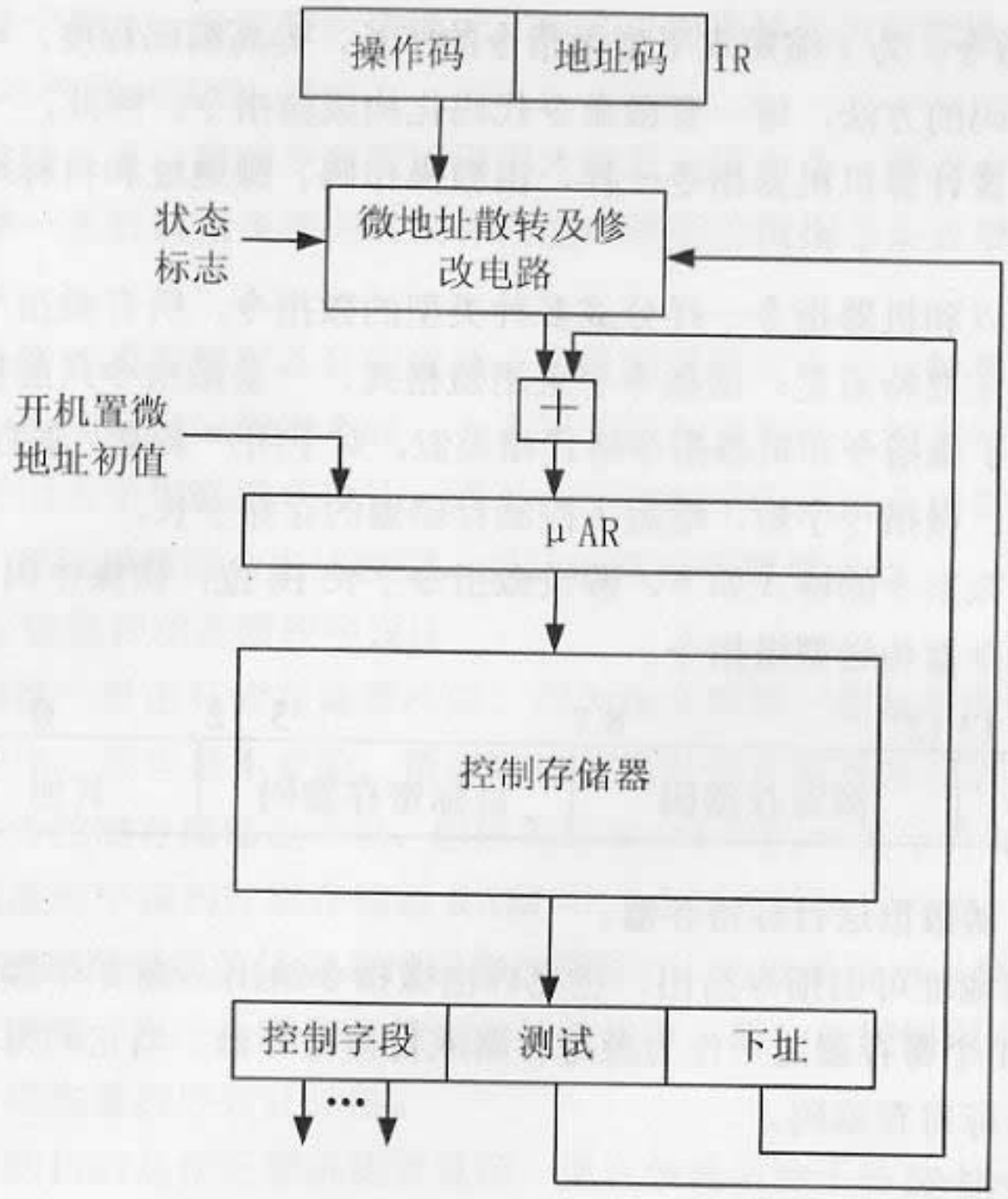


图7.20　判定方式产生后继微地址的原理图

3. 微指令的格式类型

微指令格式的设计直接影响微程序控制单元的结构和微程序的编制，也直接影响计算机的速度和控存的容量。在设计微指令的格式时，要考虑机器硬件的数据通路结构、控制存储器的速度以及微指令的编制等因素，实现计算机的指令系统。不同的计算机有不同的微指令格式，但一般分为水平型微指令和垂直型微指令两种类型。

（1）水平型微指令。其基本特征是：微指令字采用长格式，也就是上述微指令直接控制法的基本思想，一条微指令能控制数据通路中多个功能部件并行操作。控制信息的编码简单，尽可能使微命令与控制门之间有直接对应的关系。在微指令字中，不必给出源部件或目标部件的编址，因为它们已经隐含在有关的并行操作的微命令中。水平型微指令的一般格式为：

控制字段	判别测试字段	下址字段

水平型微指令的优点是：一条微指令可同时发许多个微命令，且微指令控制字段直接控制，不用字段译码，使得微指令执行效率高、速度快、灵活，各部件执行操作的并行能力强；还有，水平型微指令来解释一条指令或完成某种功能所需要的微指令条数少，因此编制的微程序就比较短。

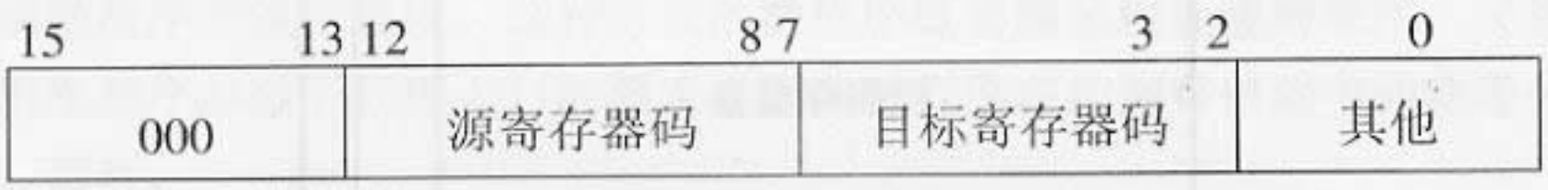
它的主要缺点是：微指令字太长，明显地增加了控制存储器的存储字长。

（2）垂直型微指令。为了缩短水平型微指令的字长，提高编码程度，可采用垂直型微指令设计。它采用完全编码的方法，将一套微命令代码化构成微指令。因此，一条微指令只能控制一两种微操作，它就像计算机机器指令一样，由微操作码、源地址和目标地址以及其他附带信息组成。

垂直型微指令可以和机器指令一样分成多种类型的微指令，所有微指令构成一个微指令系统。垂直型微指令的主要特点是：微指令字采用短格式，一条微指令只能控制一两个微操作，并行控制能力差。由于微指令和机器指令格式相类似，对于用户来说，垂直型微指令比较直观，容易掌握和便于使用。微指令字短，缩短了控制存储器的存储字长。

例如 4 条垂直型微指令的格式如下，假设微指令字长 16 位，微操作码 3 位。

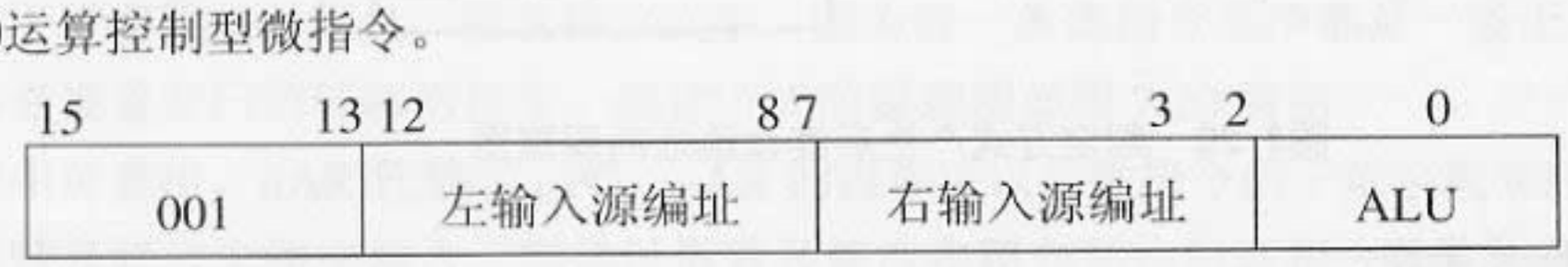
①寄存器——寄存器传送型微指令。



功能：把源寄存器数据送目标寄存器。

参与操作的数据地址可由指令给出，也允许由微指令给出。源寄存器码和目标寄存器码均为 5 位，可以指定 31 个寄存器之一作为源寄存器或目标寄存器；当它们为 00000 时表示由指令给出源寄存器码和目标寄存器码。

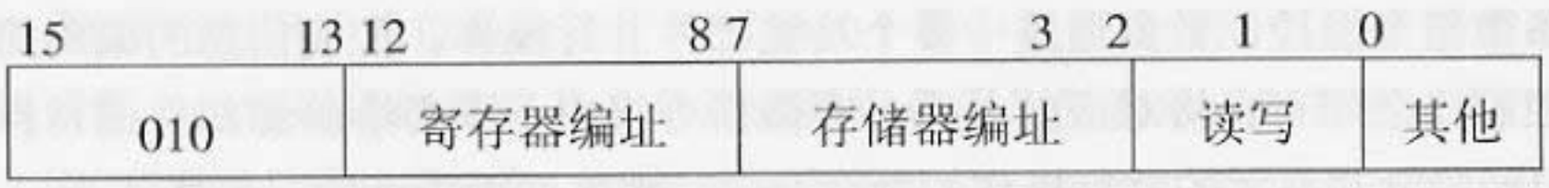
②运算控制型微指令。



功能：选择运算器（ALU）的左、右两输入源信息，按 ALU 字段所指定的运算功能（8 种操作）进行处理，并将结果送入暂存器中。

左、右输入源编址可指定 31 种信息源之一；当其为 00000 码时表示由指令的地址码部分指定左、右输入源的编址。

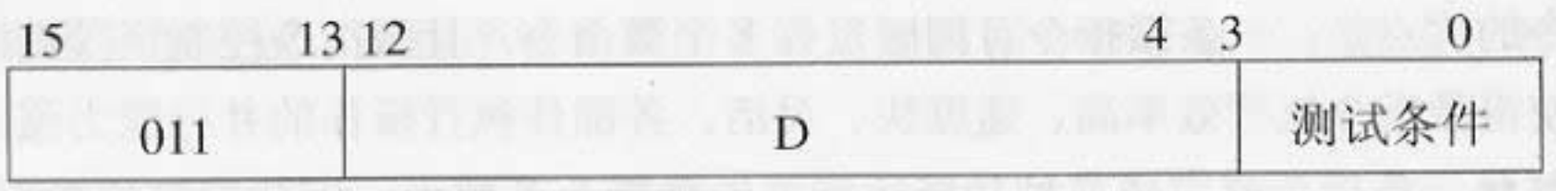
③访问主存微指令。



功能：将主存中一个单元的信息送入寄存器或者将寄存器的数据送入主存。

存储器编址可以按照系统设计所规定的寻址方式进行编址。第 2～1 位规定是读或写操作。

④条件转移微指令。



功能：根据测试对象的状态决定是转移到 D 字段所指定的微地址单元，还是顺序执行下一条微指令。

9 位 D 字段不足于表示一个完整的微地址，但可以用来替代当前微地址 μPC 的低位；4 位测试字段可以规定 16 种测试条件。

垂直型微指令的缺点是：微指令要经过译码才能发出微命令，微指令的执行效率低，并行操作性比较差，解释一条机器指令或完成某种功能所需要的微指令条数增多，增加了纵向微程序容量。

以上两种微指令格式类型都有各自的优缺点，单独采用其中任一种类型的微指令设计都是不合适的，因此，一般实际设计微指令时，大多采用两种类型结合的方法来设计微指令，微指令的一些控制字段采用水平型微指令设计，而另一些控制字段采用垂直型微指令设计，这样使得微指令字长适当，并行操作能力也比较强，设计的微程序容量适当。

4. 微程序控制存储器和动态微程序设计

微程序控制存储器一般由只读存储器构成，因为指令系统一般是不变的，微程序是解释执行指令的，因此微程序一般也是不变的，所以可以使用只读存储器来存放微程序。如果使用可读写的随机存储器作为控制存储器也可以，但停电后 RAM 中的内容会消失，所以，开机后还要首先将外存上存放的微程序调到控制存储器 RAM 中，才能开始执行程序。

采用 RAM 作为控制存储器的优点是可以修改微程序，也就是说可以修改指令系统，可以实现指令系统的扩充或调整。在一台微程序控制的计算机中，假如能根据用户的要求改变微程序，那么这台机器就具有动态微程序设计功能。

动态微程序设计的目的是使计算机能更灵活、更有效地适应于各种不同的应用场合。例如，在不改变硬件结构的前提下，如果计算机配备了两套可供切换的微程序，其中一套用来实现科学计算的指令系统，另一套用来实现数据处理的指令系统，那么该计算机就能高效率地实现科学计算或者数据处理。动态微程序设计也允许用户在原来指令系统的基础上增加一些指令来提高程序的执行效率。总之动态微程序设计需要可读写控制存储器的支持，否则难以改变微程序的内容。用于动态微程序设计的控制存储器称为可写控制存储器或用户控制存储器。

由于动态微程序设计要求用户对计算机硬件组成结构非常熟悉，因此真正由用户自行编写微程序是很困难的，所以尽管设想很好，实事上是难以推广的，一般要由机器设计人员才能设计实现。

5. 毫微程序设计

从上面对两类微指令的分析可以看出，水平型微指令可直接从微指令发出微命令；而垂直型微指令则是要经过译码才能产生微命令。如果把垂直型微指令设计和水平型微指令设计结合起来，采用两级微程序设计。第一级为垂直型微程序，用来解释机器指令，称为微程序，并存放在称为微程序存储器的控存中。第二级为水平型微程序，用来解释垂直微指令，并产生相应微命令，实现数据通路的控制。由于它是解释微程序的微程序，所以称为毫微程序，存放在称为毫微程序存储器的控存中。这样毫微程序的每条水平型微指令就可称之为毫微指令；而使用垂直微指令和毫微指令这两种微指令的程序开发就叫作毫微程序设计。毫微程序设计使得微程序流的控制和微命令发出完全分离，微程序流控制由微程序级实现（垂直型微指令），而微命令则由毫微程序产生。在毫微程序的具体设计中，常遇到以下 3 种情况。

（1）微命令是非常简单的控制信号，可由垂直型微指令直接产生，无需再用毫微指令去

解释。

（2）一条垂直微指令只用一条毫微指令来解释。这种情况可简化垂直微指令的译码器，并且相同的毫微指令在毫微程序存储器中只存放一条即可，能减少控制存储器的容量。毫微程序设计的主要目的就是要减少控制存储器的容量。

（3）一条垂直微指令由一段毫微程序来解释。这是最一般的情况。这时毫微程序与垂直微指令的关系就类似于微程序与机器指令的关系。这种情况，微程序设计更随意，更能有效地控制数据通路。

采用毫微程序设计且能执行毫微程序的控制器称为毫微程序控制器。从毫微程序设计原理，我们可以知道，毫微程序控制器必须有两个存储器和解释装置，如图 7.21 所示。

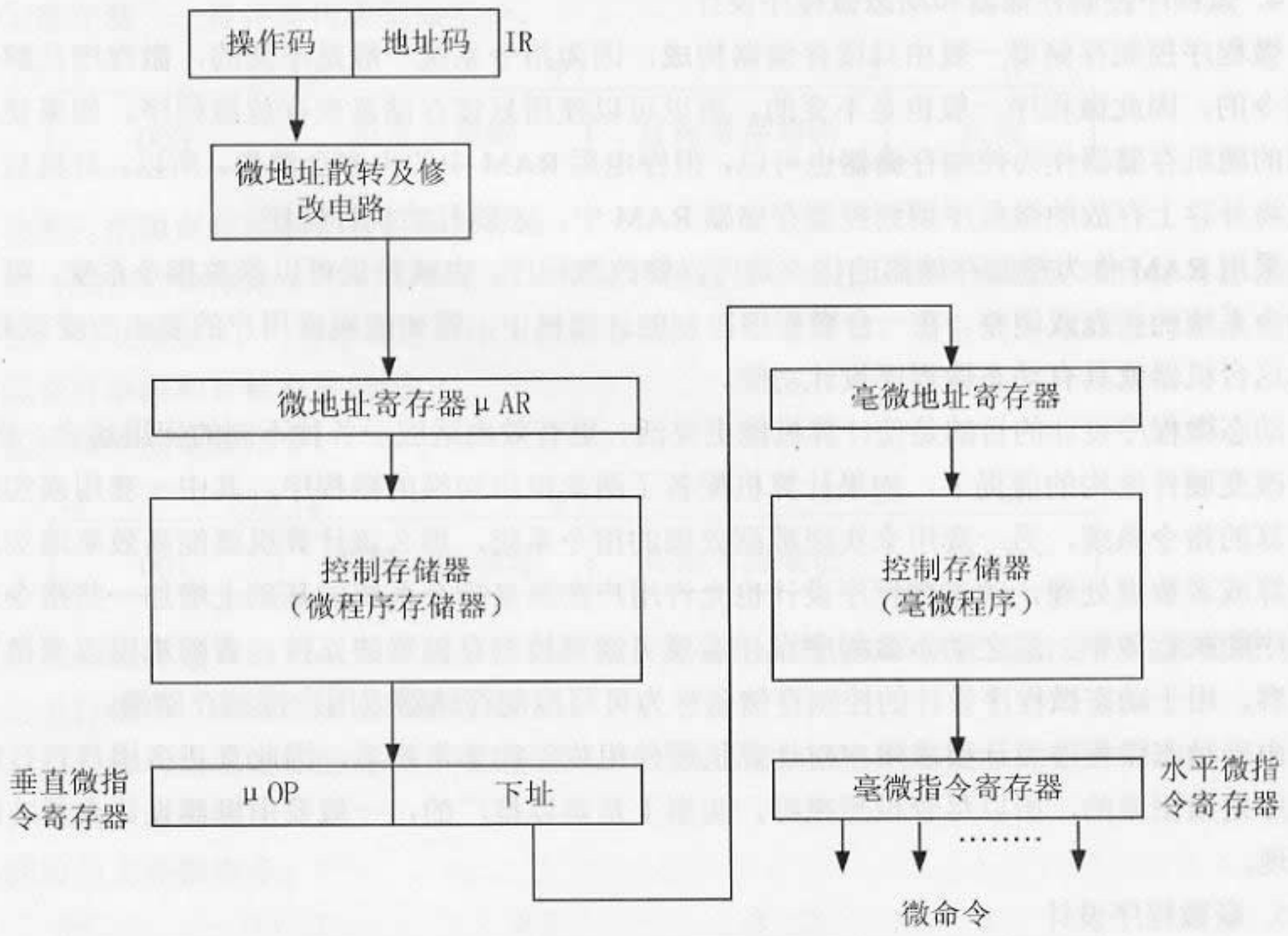


图7.21　毫微程序控制器结构

7.4　微程序控制器及其微程序设计举例

为了加深对微程序控制器原理的理解，我们以 Yy-z02 模型机为例来说明微程序控制器的组成及其微程序设计方法。

7.4.1　微程序控制器组成实例

图 7.22 给出微程序控制器的计算机组成的简化逻辑框图。从框图上可以看到，时钟单元产生整机所需的 T1、T2、T3、T4 时钟信号。各单元的数据传递均通过总线。

机器指令存放在 MEM 主存储器中，存储器的读、写控制信号是：M-R#和 M-W#。程序计数器 PC 由 CLR#信号清零，由 B-PC 信号打入，由 PC+1 信号加 1，由 B-PC#信号将 PC 内容输出到总线。指令的地址由程序计数器 PC 通过总线打入地址寄存器 AR，AR 的输出直接连接主存储器的地址线，从而寻址到指令的地址单元；从主存储器中取出指令通过总线送指令寄存器 IR，IR 中的指令操作码经过指令译码（散转电路）形成该指令的微程序入口地址，从控制存储器中取出微指令，送到微指令寄存器，发出各种微命令控制各部件工作。运算器和移位器的数据输入、输出都是通过总线，通用寄存器的输入输出也是通过总线，通用寄存器的选择信号是由指令寄存器 IR 中的低位部分经过寄存器译码电路而产生的。当 I/O-R# 和地址寄存器 AR 的某一位 A_i 这两个信号同时低有效时（A_i 表示输入单元的端口地址），使输入单元打开连通总线，此时可以通过手动开关向总线输入数据。由 I/O-W# 和地址寄存器 AR 的某一位 A_i 这两个信号同时低有效时（A_i 表示输出单元的端口地址），使总线连通输出单元，即将总线上的数据输出到输出单元发光管显示。

指令的执行过程是：

（1）用开关单元的 CLR 开关将 PC 程序计数器、控存地址寄存器和微指令寄存器清零，使程序从 0 地址单元取指令。

（2）微指令从控存 00 微地址开始执行，经过微程序控制台使在 01 微地址的微指令执行时的 T2 时刻发控制信号，T3 时刻将 PC 程序计数器的内容打入 AR 地址寄存器。

（3）02 微地址的微指令发 M-R#、B-IR 信号，将该地址的指令从存储器中取出送总线，并打入 IR 指令寄存器。

（4）03 微地址的微指令发 J1#信号，将指令寄存器中的指令通过指令译码器（散转电路）转换成微地址寄存器（控存地址寄存器）的输出置 1（SE6-SE0）信号，强制将当前微指令的下址字段中的某些位置 1，从而形成下一条微指令地址（即该指令的微程序入口地址）。

（5）从控存中取出该指令的第一条微指令到微指令寄存 μIR 发出控制信号，并由该微指令的下址字段 MA6～MA0 经微地址寄存器再到控存取下一条微指令。

（6）每一条微指令的控制信号控制各单元（如运算器单元、输入单元、输出单元、寄存器单元等）完成相应的操作，直到一条指令的所有微指令执行完，即完成一条指令的功能；此时，先判测有无中断，若中断信号有效，则将下一条指令的 PC 压入堆栈，并从中断向量单元取出中断子程序入口地址，再回到 01 微地址执行取指令的微指令，即转入中断子程序执行。若中断信号无效，则回到 01 微地址执行取指令的微指令，即继续执行指令。

该模型机微程序控制器主要有 4 个单元组成：

1. 时序电路单元（CLOCK UNIT）

见 7.2.2 节及图 7.9 所示，介绍了时序电路单元原理。

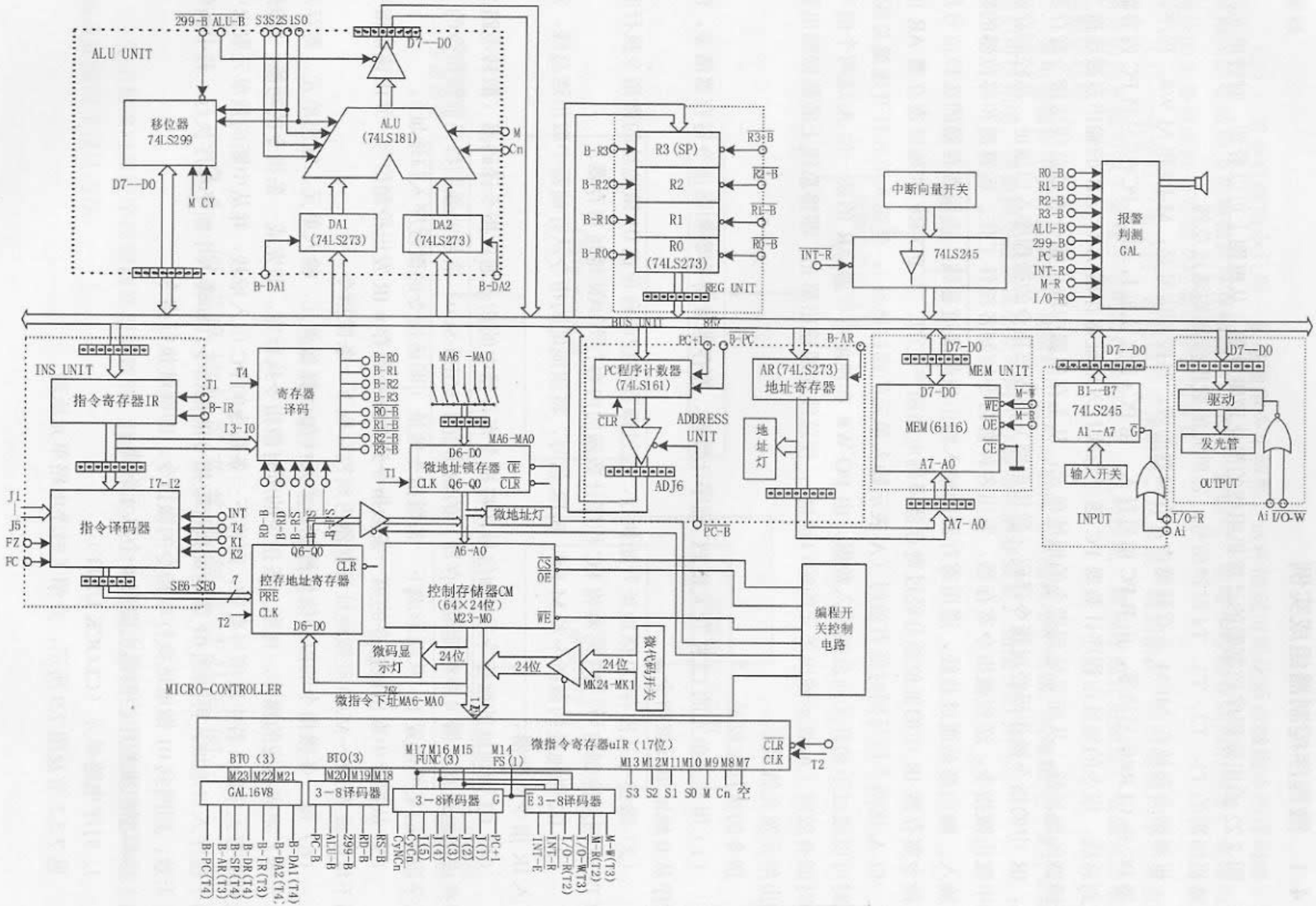


图 7.22　微程序控制器的计算机组成框图

2．计数器与地址寄存器单元（ADDRESS UNIT）

计数器与地址寄存器单元（ADDRESS UNIT），由地址寄存器 AR、程序计数器 PC 及 8 位地址显示灯构成，其电路框图如图 7.22 中 ADDRESS UNIT 所示。

其中，地址寄存器 AR（8 位）由一片 74LS273 构成，其输入端将总线单元（BUS UNIT）的 D7-D0 输入到 AR，输出端接至存储器地址 A7-A0，并用地址灯显示 A7-A0。地址寄存器 AR 的打入脉冲由控制信号 B-AR 控制，B-AR 在 T3 时刻上升沿有效。

程序计数器 PC（8 位）由两片计数器 74LS161（4 位）构成，程序计数器 PC 的输入端接自总线单元（BUS UNIT）的 D7-D0，输出端则通过一三态门输出至总线。CLR 开关的负脉冲将使 PC 清零；置数 PC 则需要控制信号 PC+1 上升沿且 B-PC#=0；加 1 计数则只需要 PC+1 上升沿。

3．指令寄存器单元（INS UNIT）

指令寄存器单元（INS UNIT）中，由一片 74LS273 构成指令寄存器 IR（8 位），其输入端接自总线单元（BUS UNIT）的 D7-D0，输出端为 I7～I0 即指令码，供 INS UNIT 单元的指令译码电路和寄存器译码电路使用。电路框图如图 7.22 中 INS UNIT 所示。

指令译码电路由一片 GAL20V8 实现，其主要功能是根据指令操作码 IR_7-IR_2、微程序的转移方式 J1—J5、运算结果的状态 FC 和 FZ、控制台开关 K1 和 K2 的状态，进行逻辑译码，产生后继微地址的控制信号 SE6#—SE0#，以控制实现机器指令转入微程序入口和微程序的顺序、分支、循环运行；其原理参见 7.3.4 节及图 7.18 所示。

寄存器译码电路由一片 GAL16V8 实现，其原理如图 7.23 所示。

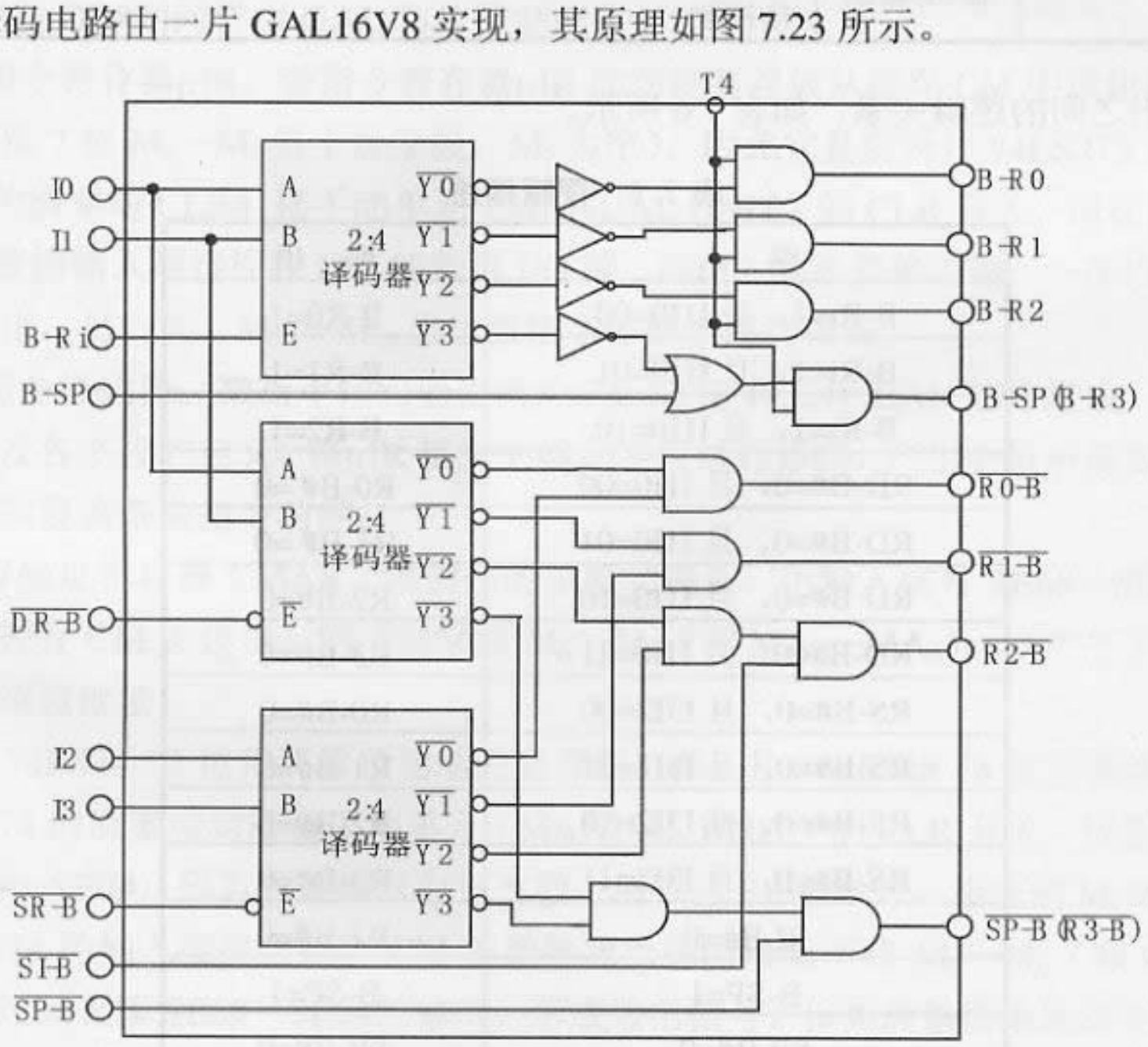


图7.23　寄存器译码电路

图 7.23 中，其输入信号有 B-Ri、DR-B#、SR-B#、SI-B#（来自微控器单元 MAIN CONTROL

UNIT)，指令码 I3-I0（来自指令寄存器），输出信号为 B-R0、B-R1、B-R2、B-SP、R0-B#、R1-B#、R2-B#、SP-B#（送至寄存器单元 REG UNIT)。各信号的解释如表 7.5 所示。

表 7.5　信号的解释

信　号	输入/输出	解　释
I1I0	输入	指令码中目的寄存器 DR 的编码字段
I3I2	输入	指令码中源寄存器 SR 的编码字段
B-Ri	输入	目的寄存器 DR 的装载控制信号，与指令码 I1I0 一起译码产生 B-R0、B-R1、B-R2 信号
DR-B#	输入	目的寄存器 DR 内容送总线的控制信号，与指令码 I1I0 一起译码产生 R0-B#、R1-B#、R2-B#、R3-B#信号
SR-B#	输入	源寄存器 SR 内容送总线的控制信号，与指令码 I3I2 一起译码产生 R0-B#、R1-B#、R2-B#、R3-B#信号
Ri-B#	输入	变址寄存器 SI（(隐含为 R2)内容送总线的控制信号，直接产生 R2-B#信号
B-R0、B-R1、B-R2、B-R3	输出	寄存器 R0、R1、R2、R3 从总线上装入数据的控制信号
R0-B#、R1-B#、R2-B#、R3-B#	输出	将寄存器 R0、R1、R2、R3 的内容送至总线上的控制信号
B-SP 和 SP-B#	输入/输出	堆栈指针寄存器的数据输入和输出由微指令寄存器发出的微码直接控制

输入与输出之间的逻辑关系，如表 7.6 所示。

表 7.6　逻辑描述

输　入	输　出
B-Ri=1，且 I1I0=00	B-R0=1
B-Ri=1，且 I1I0=01	B-R1=1
B-Ri=1，且 I1I0=10	B-R2=1
RD-B#=0，且 I1I0=00	R0-B# =0
RD-B#=0，且 I1I0=01	R1-B# =0
RD-B#=0，且 I1I0=10	R2-B#=0
RD-B#=0，且 I1I0=11	R3-B#=0
RS-B#=0，且 I3I2=00	R0-B#=0
RS-B#=0，且 I3I2=01	R1-B#=0
RS-B#=0，且 I3I2=10	R2-B#=0
RS-B#=0，且 I3I2=11	R3-B#=0
SI-B#=0	R2-B#=0
B-SP=1	B-SP=1
SP-B#=0	SP-B#=0

4．微控器单元（MAIN CONTROL UNIT）

微控器单元（MAIN CONTROL UNIT）的逻辑框图如图 7.24 所示。该单元主要由以下部件组成：

（1）控制存储器 CM。控制存储器 CM 由 3 片 2816（2K×8 位）组成，存放 24 位的微指令。由于 3 片 2816 的高位地址 A_{10}—A_7 均接地，因此，控存的实际容量为 128×24 位，即可以存放 128 条微指令。

控存的片选信号 CS#、输出使能 OE#、写使能 WE#均由 GAL 芯片根据编程开关的状态及联机的情况控制产生并输出。而控存的地址输入 A_6—A_0（即微地址 MA_6—MA_0），可以由微地址锁存器μAR 提供（手动“编程 PROM”或“校验 READ”状态下），也可以由后继微地址修改逻辑来提供（“运行 RUN”状态下），或者由 PC 机控制送出（联机状态下）。控存的地址端接有 7 个微地址显示灯。

控存的数据输入/输出，可以用微代码开关输入（手动“编程 PROM”状态下），并在微代码灯上显示，也可以由 PC 机控制控存的读写数据（联机状态下）。控存中的 24 位微码，高 16 位送微指令寄存器μIR 保存并译码，低 7 位（下址字段）则送后继微地址转移控制逻辑产生后继微地址。

（2）微地址锁存器μAR。μAR 由一片 74LS374（8 位锁存器）构成，用于锁存手动操作时由开关拨入的微地址 MA_6—MA_0，并提供给控存。μAR 的输入端以排针的形式引出，标记为 MA_6—MA_0，用以连接手动单元（MANUAL UNIT）的开关；输出端接至控存的地址输入端；但μAR 的输出使能 OE#则同样由 GAL 芯片控制产生。

（3）微指令寄存器μIR。微指令寄存器μIR 的功能是存放从控存 CM 中读出的高 16 位的微码 M_{23}—M_8（低 7 位 M_6—M_0 是下址字段，M_7 为空），因此它是由两片 74LS273（8 位寄存器）组成的。μIR 的清零端 CLR# 接手动单元（MANUAL UNIT）的 CLR 开关，即在总清时，使μIR 清零。μIR 的数据输入端接控存 CM 的数据 I/O 端；而μIR 的数据输出端，一面以排针 M_{13}—M_8 的形式直接引出，另一面，M_{23}—M_{14} 另送微指令译码器进行译码。

（4）微指令译码器。微指令译码器由两片 74LS138 和二片 GAL 芯片组成，其功能是根据微指令的格式及各字段的定义，将μIR 送来的编码字段进行译码，产生全机所需要的各种微操作控制信号，以实现该条微指令功能。

（5）控存地址寄存器 CMAR。该部件的主要功能是，由输入信号 SE6#—SE0#控制修改当前微指令（从控存 CM 中读出）的下址字段 M_6—M_0（即 MA_6—MA_0），以产生下条微指令的控存地址，即后继微地址。

它由 4 片 74LS74（2 位带清零预置端的寄存器）和一片 74LS245（8 位三态缓冲器）连接而成，4 片 74LS74 的清零端均接自手动单元（MANUAL UNIT）的 CLR 开关，预置端则分别接自输入信号 SE6#—SE0#，因为预置端低电平有效，所以当 SEi#=0 时，相应的 M_i 即 MA_i 被置 1。

4 片 74LS74 的输入端接自控存 CM 的数据输出端的最低 7 位 M_6—M_0（即 MA_6—MA_0），MA_6—MA_0 经过预置端 SE6#—SE0#的修改，形成输出信号，作为后继微地址送至控存的地址输入端，以寻址下条微指令。

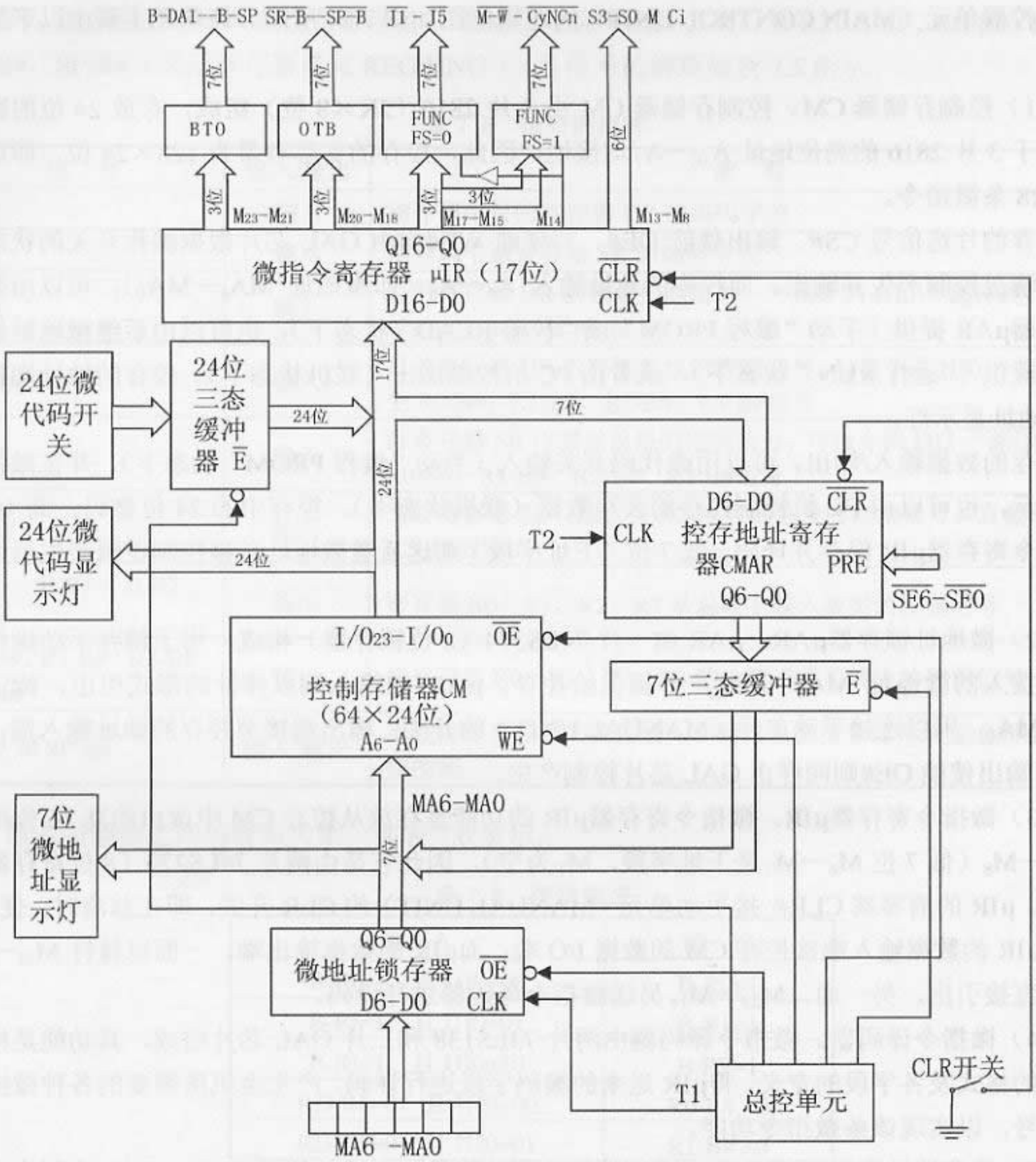


图7.24 微控器逻辑框图

7.4.2 模型机微程序设计

本节主要介绍如何在已经连接好的如图 7.22 所示的模型机上，根据 6.5.2 节所介绍的该机指令的功能、格式、寻址方式，进行微程序设计。

一、自行设计模型机及其新指令的微程序段

设计的步骤为：

（1）设计实验模型机的结构和数据通路。

（2）设计指令的功能、格式（包括指令码）及寻址方式。

（3）在以上的基础上，编写微程序流程图。

（4）根据指令码和转移方式 J1～J5，分配微地址及下址字段。

（5）根据微指令格式，编写微指令代码。

二、微程序流程图的编写

在已经确定模型机的结构和机器指令的情况下，介绍编写微程序流程图的方法及要点。

1. 机器指令的功能由微程序完成，一条机器指令对应着一段微程序。每条指令的微程序都包含 3 部分：取指令微程序段、根据操作码散转至微程序入口的微指令、该机器指令的独立微程序段。但是每一条指令的前两部分都相同，称作公共操作，不同的是第三部分的独立微程序段，取决于该机器指令的寻址方式和功能，用于实现指令规定的特殊功能

例如，取指令及散转的公共微程序段为下面的 3 条微指令：

（1）PC→AR，PC+1。

（2）RAM→IR。

（3）J1=0 实现散转微程序入口。

2. 每条微指令可以实现总线上的一个数据传送（例如 PC→AR），或者进行运算器的一个运算（例如 DA_1+DA_2→DR），或者启动存储器的一个读/写（例如 RAM→DR）

按照每条机器指令的功能，可以将各种操作归纳为以下几种：

（1）通用寄存器之间的传送操作。通过源寄存器数据送总线，而目的寄存器从总线上接收数据来实现。

例如，指令 MOV DR，SR，功能为将源寄存器 SR 的内容送目的寄存器 DR。其微程序段为一条微指令：SR→DR。

（2）存储器访问操作。

①读访问操作，参照模型机的数据通路，通过以下两条微指令实现：

（a）送地址到总线，并打入地址寄存器 AR。

（b）启动存储器读操作，并将读出的数据从总线上接收至目的部件。

例如，指令 MOV DR，@SR 功能为将源寄存器 SR 所指示的存储器地址单元的内容送目的寄存器 DR，即源操作数是寄存器间接寻址。其微程序段为以下两条：

（a）SR→AR。

（b）RAM→DR。

另外，取指令也是一种典型的存储器读访问操作。

②写访问操作，参照模型机的数据通路，通过以下两条微指令实现：

（a）送地址到总线，并打入地址寄存器 AR.

（b）送数据到总线，启动存储器写操作。

例如，指令 MOV @DR，SR，功能为将源寄存器 SR 的内容写至目的寄存器 DR 所指示的存储器地址单元，即目的操作数是寄存器间接寻址。其微程序段为以下两条：

（a）DR→AR。

（b）RAM。

（3）运算器的运算操作。同样，参照模型机的数据通路，通过以下 3 微指令实现：

①送第一个数据到暂存器 DA_1（或者 DA_2）。

②送第二个数据到暂存器 DA_2（或者 DA_1）。

③选择 ALU 运算功能并进行运算，结果送目的部件。

例如，指令 ADD DR，SR，功能为将源寄存器 SR 的内容与目的寄存器 DR 的内容相加，并送 DR。其微程序段为 3 微指令：

①SR→DA_1。

②DR→DA_2。

③DA_1+DA_2→DR。

3. 对于带寻址方式码 MOD 的指令，由于其指令格式不同（I_5I_4 为寻址方式码 MOD，I_7I_6 和 I_3I_2 为指令操作码 OP），这种指令的微程序段至少经过两个散转：第一次为 J1 散转，分辨出寻址方式并计算出有效地址，第二次为 J2 散转，根据 I_3I_2 分辨出指令并转入其微程序入口实现其功能

4. 编写指令的微程序流程图时，不仅数据通路要可行，还要考虑微码编写是否可行

例如，OUT [PORT]，[ADDR]指令，功能为将地址 ADDR 存储器单元中的数据输出至发光管显示，发光管的端口地址为 PORT。其微程序中，要进行存储器的读操作，从模型机框图上看，读出的数据在总线上，可以直接送输出单元显示，即一条微指令 RAM→LED；但其微码的编写却不可行，因为存储器的读操作要求微码 FUNC FS=0100，但输出单元的显示要求微码 FUNC FS=0110，冲突；因此，必须用两条微指令实现：RAM→DA_1；DA_1→LED。

5. 对于同一条指令，可能存在不同的微程序流程图，但均能实现指令的功能

三、微地址及下址字段的分配

在表 7.3 模型机微指令格式中，微指令编码高 17 位是控制字段，微码的最低 7 位是下址字段，其编排与微指令本身的控存地址及机器指令的编码都有关系，方法如下：

（1）指令系统中的所有指令的微程序流程图编制完成后，首先要分配每条机器指令的微程序入口地址。步骤如下：

①确定发送 J1#信号的那条微指令的下址字段。

编排取指令的微程序段中两条微指令的地址，例如第一条微指令“PC→AR，PC+1”的微地址安排为 01H，第二条微指令“RAM→IR”的微地址安排为 02H，第三条微指令“J1# =0”的微地址安排为 03H，则第一条微指令的下址字段则可设置为 000010B=02H（因为是顺序执行），第二条微指令的下址字段可设置为 000011B=03H，第三条微指令的下址字段的设置必须满足 MA_3—MA_0=0000B，如 10H 或者 20H 等，因为 J1# 转移必须要根据指令操作码修改微地址 MA_3—MA_0。

②确定指令的操作码。

对指令系统的所有机器指令进行操作码的编码；注意，对于双操作数都在寄存器中的指令和含寻址方式码的指令及单个操作数在寄存器中的指令的编码有所不同，详见 6.5.2 节。

③根据 J1#转移的规则确定每条指令的微程序入口地址。

例如：假设微指令“J1#散转”的下址字段为 10H；又 MOV 指令操作码为 0000，则其微程序入口地址为 10H；ADD 指令操作码为 0101，则其微程序入口地址为 15H。

（2）如果流程图中还有 J2#、J3#、J4#、J5#转移，则接下来必须确定它们的各分支的入口微地址。

同样，要首先确定发送 J2#、J3#、J4#、J5#信号的微指令的下址字段，其原则是按照 J2#转移的微指令的下址字段必须满足 MA_1MA_0=00B；而按照 J3#转移的微指令的下址字段也必须满足 MA_1MA_0=00B；按照 J4#转移的微指令的下址字段也必须满足 MA_1MA_0=00B；按照 J5#转

移的微指令的下址字段必须满足 MA5=0B。然后，再根据各自转移的规则确定各自的微程序分支的微地址。见 7.3.4 节的“微指令下址字段设计方法”。对 J2#转移，还要根据指令的第二操作码字段 I_3I_2 来修改微地址 MA_1MA_0，确定指令的微程序入口。

例如：假设按照 J2#散转的微指令的下址字段为 3CH，则其对应的 4 条指令的微程序分支微地址为 3CH（I_3I_2=00）、3DH（I_3I_2=01）、3EH（I_3I_2=10）、3FH（I_3I_2=11）。

又如：假设按照 J4#散转的微指令的下址字段为 24H，则其对应的 4 个微程序分支微地址为 24H（若 FZ=0 且 FC=0）、25H（若 FZ=1 且 FC=0）、26H（若 FC=1 且 FZ=0）、27H（若 FZ=1 且 FC=1）。

（3）在确定了上述有特殊要求的微指令的固定地址后，对于其他的顺序执行的微指令，只需直接按前后顺序随意编排微地址，只要微地址不重复即可。

（4）每条指令的微程序段的最后一条微指令的下址字段一定是取指令微程序段的首地址。

四、微指令代码的编写

根据写好的微程序流程图，参照数据通路，首先排列出每条微指令必须发送的微操作控制信号，然后，对照微指令格式，写出这些微操作控制信号对应的微代码。如表 7.7 所示，举例说明微指令的代码化。

表 7.7　微指令的编码方法

序号	微指令	应发送的微操作控制信号	微指令编码（M_{24}～M_7）					
			BTO	OTB	FUNC FS=1	FUNC FS=0	FS	S_3～S_0 MCi
1	PC→AR，PC+1	PC-B#，B-AR，PC+1	000	111	000	000	1	000000
2	RAM→IR	M-R#，B-IR	011	000	000	010	0	000000
3	J1# 方式散转	J1#=0	000	000	001	000	1	000000
4	SR→DR	SR-B#，B-DR	100	011	000	000	0	000000
5	DR→RAM	DR-B#，M-W#	000	100	000	001	0	000000
6	DA_1+DA_2→DR	ALU(F=A 加 B)，不带进位，ALU-B#，B-DR	100	001	111	000	1	100101
7	DA_1→LED	ALU(F=A)，ALU-B#，I/O-W#	000	001	000	011	0	000001
8	RAM→PC	B-PC#，PC+1	111	000	000	000	1	000000

五、微程序设计举例

假设模型机指令系统有 5 条机器指令：

地址	指令字	助记符	操作
00H	11000000	IN　R_0，[PORTAR]	端口数据→R_0
01H	00000000 (端口地址)		
02H	11000100	ADD R_0，[10H]	R_0+[10H]→R_0
03H	00010000（直接地址）		

04H	11001000	STA[10H]，R_0	R_0→[10H]
05H	00010000（直接地址）		
06H	11001100	OUT [PORTAR]，[10H]	[10H]→LED
07H	00010000（主存地址）		
08H	00000000（端口地址）		
09H	11010000	JMP 00H	00→PC
0AH	00000000（转移地址）		

根据指令的功能，设计每一条指令的微程序，图 7.25 显示了这 5 条指令的微程序流程图，在流程图中给每一条微指令分配微地址，标注在每个微指令框的右上方。

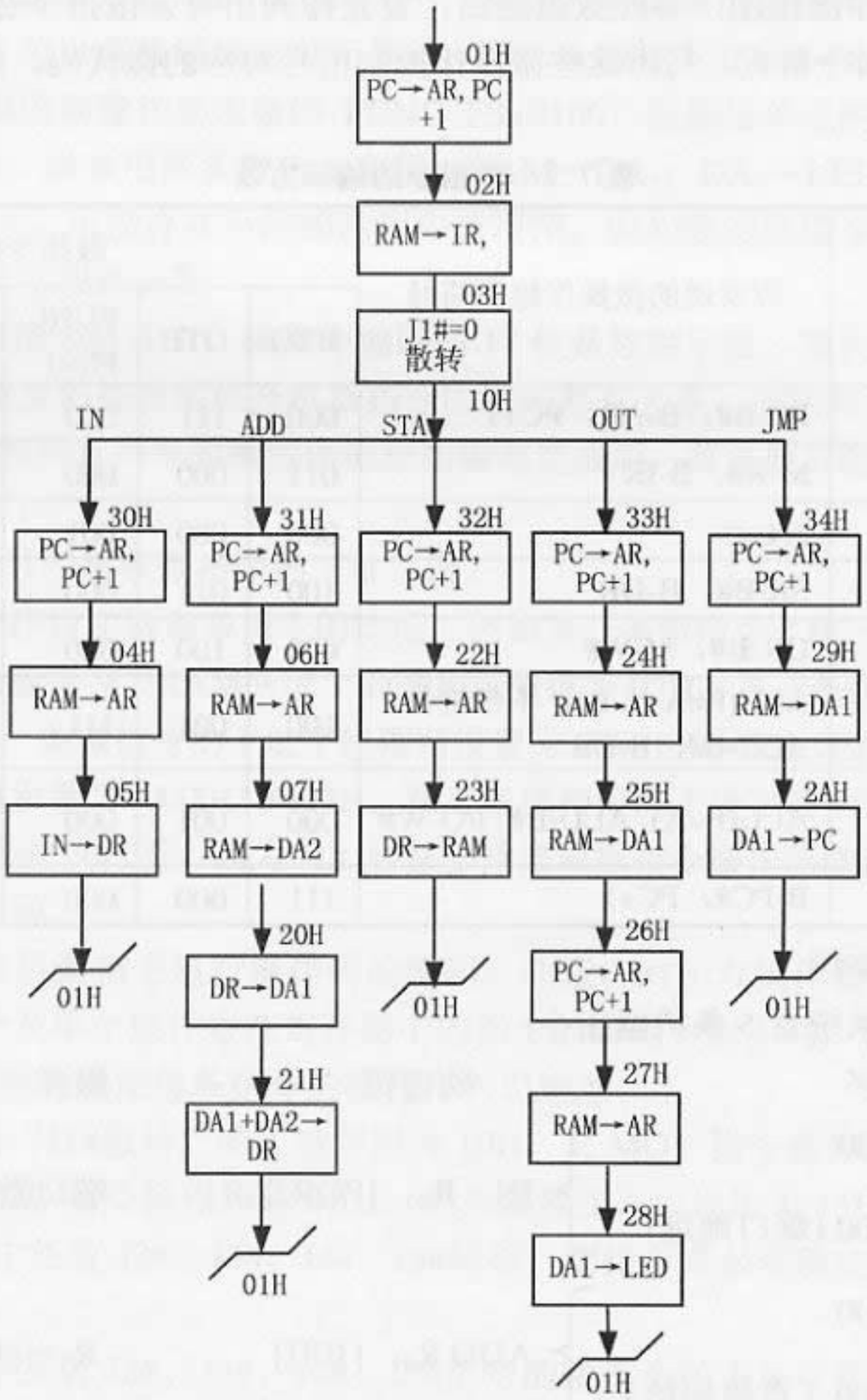


图7.25　5条指令微程序流程图

根据微程序流程图中每个微指令框所表示的微指令功能，写出每条微指令如表 7.8 所示；将机器指令码装入主存，微指令编码装入控制存储器；当执行机器指令时，机器会按照微程序控制器的原理，完成指令的执行。

表 7.8　5 条微指令码

微地址	微指令	微指令编码（M_{23}～M_0）						
		BTO	OTB	FUNC FS=1	FUNC FS=0	FS	S_3～S_0 MCi	下址
01H	PC→AR，PC+1	110	111	000	000	1	000000	0000010
02H	RAM→IR	011	000	000	010	0	000000	0000011
03H	J1# 方式散转	000	000	001	000	1	000000	0010000
04H	RAM→AR	110	000	000	010	0	000000	0000101
05H	IN→DR	100	000	000	100	0	000000	0000001
06H	RAM→AR	110	000	000	010	0	000000	0000111
07H	RAM→DA_2	010	000	000	010	0	000000	0100000
20H	DR→DA_1	001	100	000	000	0	000000	0100001
21H	DA_1 +DA_2→DR,	100	001	111	000	1	100101	0000001
22H	RAM→AR	110	000	000	010	0	000000	0100011
23H	DR→RAM	000	100	000	001	0	000000	0000001
24H	RAM→AR	110	000	000	010	0	000000	0100101
25H	RAM→DA_1	001	000	000	010	0	000000	0100110
26H	PC→AR，PC+1	110	111	000	000	1	000000	0100111
27H	RAM→AR	110	000	000	010	0	000000	0101000
28H	DA_1→OUT	000	001	000	011	0	000001	0000001
29H	RAM→DA_1	001	000	000	010	0	000000	1000000
2AH	DA_1→PC, PC+1	111	001	000	000	1	000001	0000001
30H	PC→AR，PC+1	110	111	000	000	1	000000	0000100
31H	PC→AR，PC+1	110	111	000	000	1	000000	0000110
32H	PC→AR，PC+1	110	111	000	000	1	000000	0100010
33H	PC→AR，PC+1	110	111	000	000	1	000000	0100100
34H	PC→AR，PC+1	110	111	000	000	1	000000	0101001

7.5　硬布线控制器

如前所述，计算机各个部件所需要的微操作控制信号均由控制器的“操作控制信号形成部件”（又称操作控制器）根据指令的要求来产生。而“操作控制信号形成部件”的实现则有两种方式：一种是采用存储逻辑来实现，即前面所述的“微程序控制器”；另一种则是采用组合逻辑

来实现，即本节所要讨论的“硬布线控制器”。硬布线控制器的基本原理是根据指令的要求、当前的时序及外部和内部的状态情况，按时间的顺序发送一系列微操作控制信号。它由复杂的组合逻辑门电路和一些触发器构成，因此又称为组合逻辑控制器，或常规逻辑控制器。硬布线控制器是早期的计算机控制器的一种设计方法，由于它由繁琐的硬件逻辑电路构成，一旦确定，即不可更改和扩充，这一点非常不利于控制器的设计和调试，因此后来被微程序控制器所取代。然而，随着 RISC 机的出现和 VLSI 技术的飞速发展，硬布线控制器由于其高速性能又重新受到青睐，并将它与微程序控制技术共同应用于 CPU 的设计。

7.5.1 时序系统

一条机器指令的执行过程分为取指令和执行指令，而执行指令又根据不同的指令需要不同的机器周期数来完成。在微程序控制器中，从控制存储器中取出一条微指令来实现一个操作步骤。一条机器指令的执行就是由若干条微指令分别完成各个步骤来实现的。而在硬布线控制器中，机器指令执行的各个步骤是在各个机器周期中由指令的操作码和机器周期信号及时钟周期信号组成的时序逻辑电路来完成。一个机器周期完成一个步骤。

一个指令系统，其每条指令所需要的机器周期数可能不相同，有些指令需要两个机器周期，而有些指令需要 4 个机器周期等等。因此要根据指令来产生控制器的机器周期信号，而每个机器周期内则都是 2 个或 4 个固定的时钟周期，如图 7.26 所示为具有 3 个机器周期的一条指令的周期时序。时序电路就要产生指令系统的全部指令的各种机器周期信号和时钟信号，其中机器周期信号与指令译码线有关，而时钟周期则由时序电路产生 4 个固定的 T 周期即可。

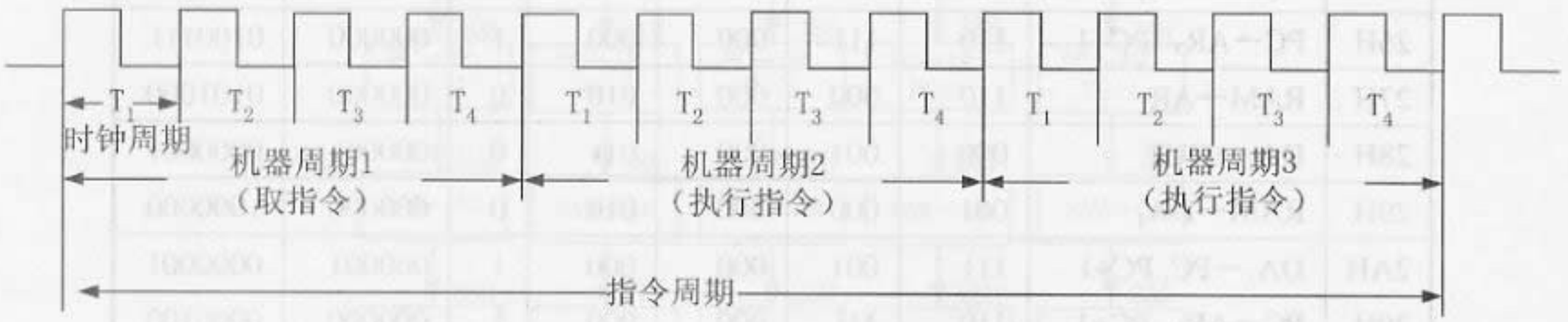


图 7.26　具有 3 个机器周期的指令周期时序图

假设某机器的指令系统有两条指令，执行指令 A 需要 3 个机器周期，而执行指令 B 需要 4 个机器周期，采用计数器输出译码方式产生机器周期信号。指令 A 需要 3 个机器周期，则计数器的计数变化状态如图 7.27 所示。指令 B 需要 4 个机器周期，则计数器的变化状态如图 7.28 所示，其中 01 状态变化至 11 状态，再变化至 10 状态，是为了避免 01 至 10 状态时两个触发器同时翻转造成毛刺。

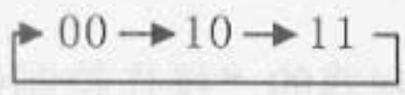


图7.27　3个机器周期计数器状态

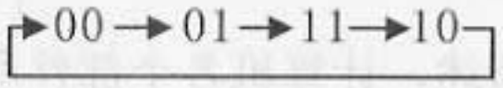


图7.28　4个机器周期计数器状态

表 7.9 列出了产生两条指令所需的机器周期信号时的计数器状态。其中 Q_1，Q_2 表示当前周期计数器状态输出，Q'_1，Q'_2 表示下一个周期计数器状态输出。

表 7.9　指令 A 和 B 各机器周期计数器状态变化

指令 A				指令 B			
Q_1	Q_2	Q'_1	Q'_2	Q_1	Q_2	Q'_1	Q'_2
0	0	1	0	0	0	0	1
1	0	1	1	0	1	1	1
1	1	0	0	1	1	1	0
				1	0	0	0

根据表 7.9 的真值表列出计数器的输出表达式，对于指令 A，其表达式为

$$Q_1' = \overline{Q}_1\overline{Q}_2 + Q_1\overline{Q}_2 = \overline{Q}_2 \tag{7-1}$$

$$Q_2' = Q_1\overline{Q}_2 \tag{7-2}$$

对于指令 B，其表达式为

$$Q_1' = \overline{Q}_1 Q_2 + Q_1 Q_2 = Q_2 \tag{7-3}$$

$$Q_2' = \overline{Q}_1\overline{Q}_2 + \overline{Q}_1 Q_2 = \overline{Q}_1 \tag{7-4}$$

根据表达式画出逻辑电路图，图 7.29 为指令 A 和指令 B 执行时产生所需要的机器周期信号 M_0～M_3 的逻辑电路图。当执行指令 A 时，产生机器周期信号 M_0、M_1、M_2；而当执行指令 B 时，产生机器周期信号 M_0、M_1、M_2、M_3。

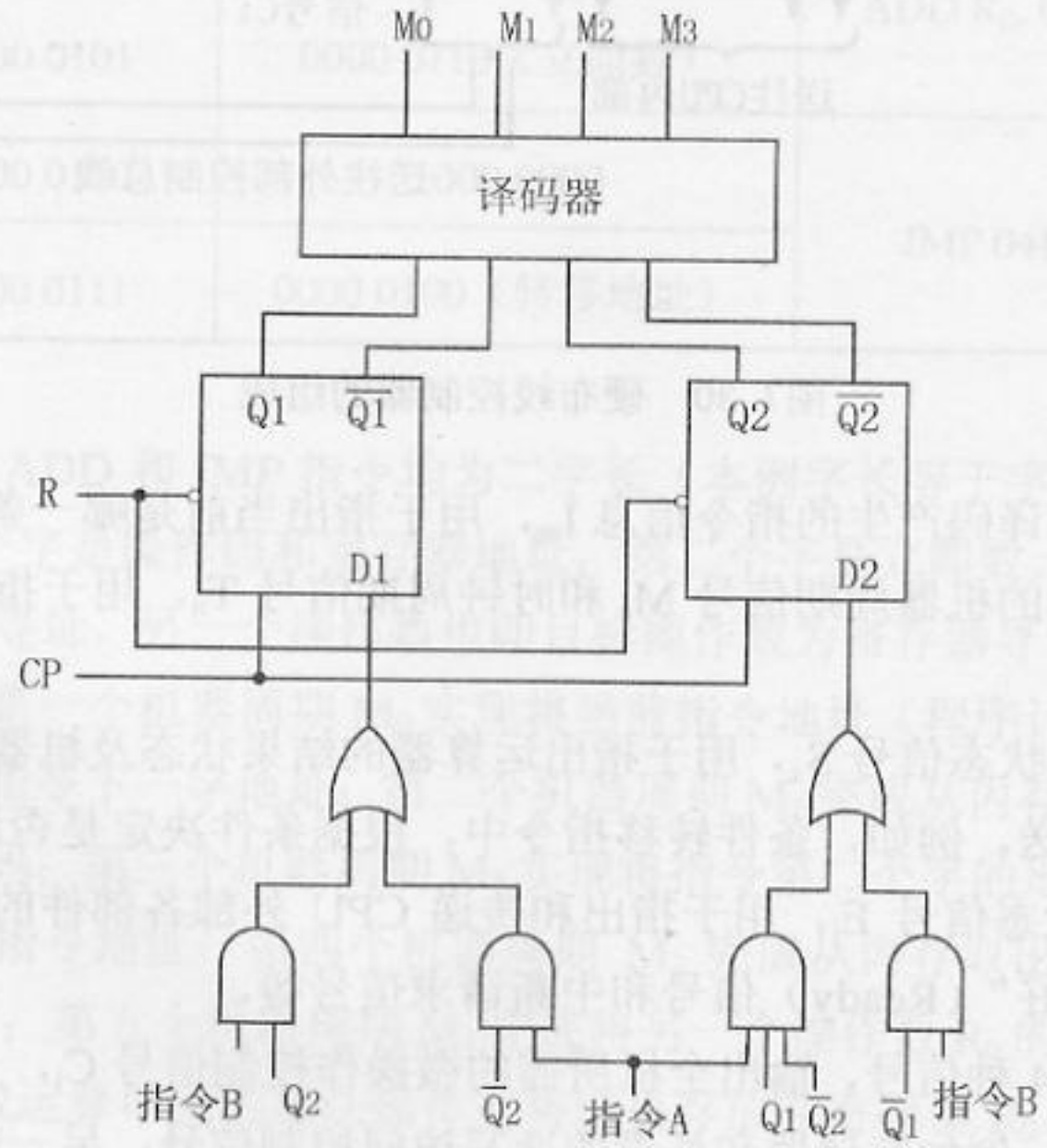


图7. 29　两条指令的机器周期产生电路图

以上只举例了两条指令的机器周期信号如何产生，而实际机器有几十条或上百条指令，则要根据所有指令所需要的机器周期信号，列出每条指令的机器周期变化规律，最好归纳出几种情况，将相同情况的指令归纳为一类，然后像上述举例方法一样列出表达式，画出逻辑电路图，产生整个机器的机器周期信号。

7.5.2 硬布线控制器的结构

图 7.30 是硬布线控制器的结构框图。由图可见，硬布线控制器的核心部件是由组合逻辑电路构成的操作控制器，即操作控制信号形成部件。操作控制器有 4 种信号输入：

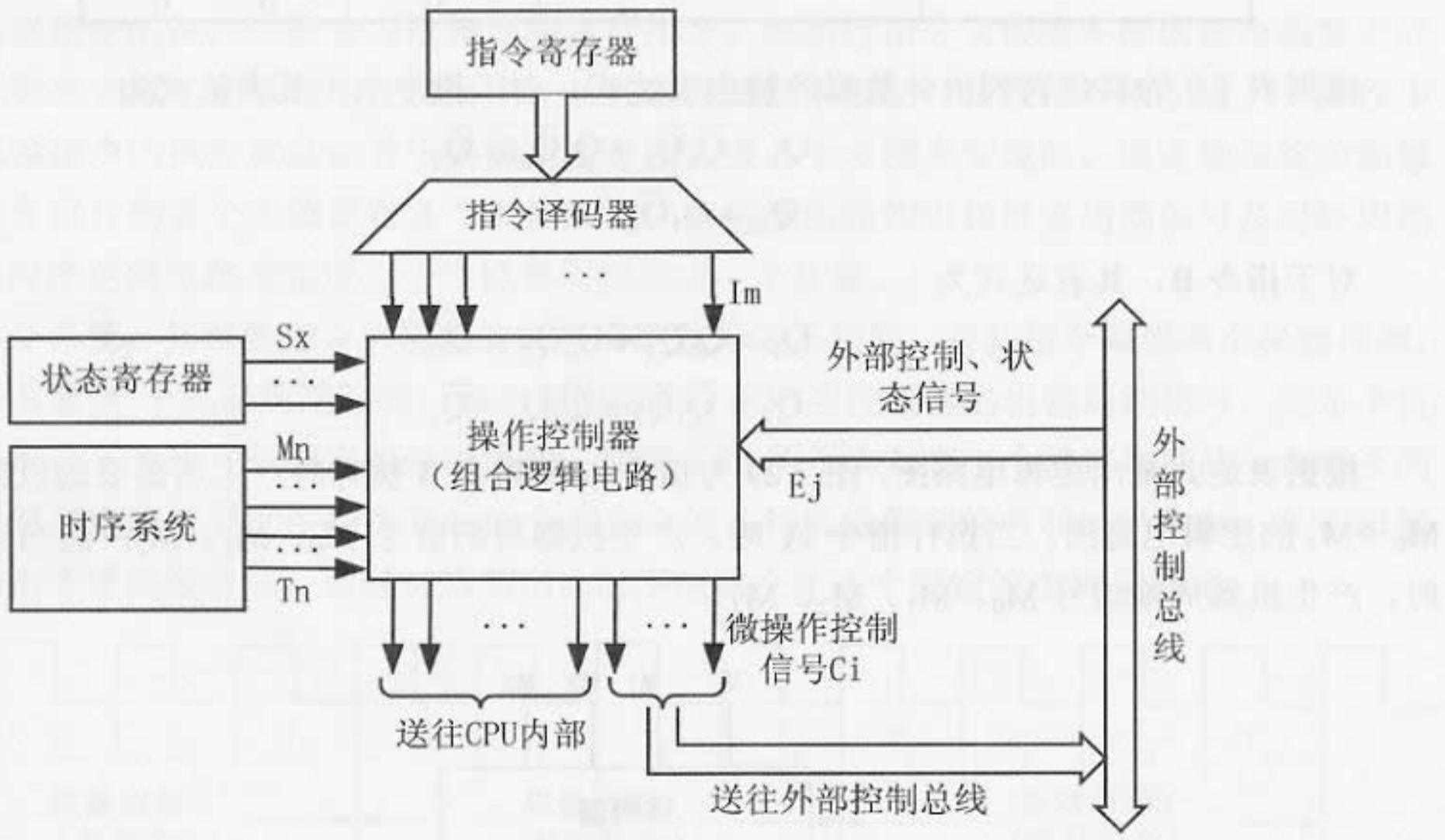


图7.30 硬布线控制器的组成

（1）经指令译码器译码产生的指令信息 I_m，用于指出当前是哪一条指令的指令周期。

（2）时序系统产生的机器周期信号 M_n 和时钟周期信号 T_n，用于指出当前处于哪一个机器周期和哪一个节拍。

（3）状态寄存器的状态信号 S_x，用于指出运算器的结果状态及机器内部的其他状态，以决定某些操作信号是否发送。例如，条件转移指令中，根据条件决定是否进行转移。

（4）外部控制、状态信号 E_j，用于指出和传递 CPU 外部各部件的状态和控制信号，例如存储器和外设的“准备好”（Ready）信号和中断请求信号等。

操作控制器根据这 4 种信号，输出全机所需的微操作控制信号 C_i，一部分送到 CPU 外部构成系统总线的控制总线，例如存储器和外设的读写访问控制信号；另一部分则送到 CPU 内部供使用，例如运算器的功能控制信号和各寄存器的控制信号。因此，从逻辑函数的角度来看，输出微操作控制信号 C_i 是 4 种输入信号的函数：

$$C_i=f_i（I_m，M_n，T_n，S_x，E_j）$$

所以，设计硬布线控制器的过程，也就是求出每个微操作控制信号 C_i 的逻辑函数 f_i 的过程。

7.5.3　硬布线控制器的设计方法

下面从整个 CPU 的设计方法这个角度来讨论硬布线控制器的设计步骤：

1. 确定指令系统，包括指令系统中每条指令的格式、功能和寻址方式

2. 围绕着指令系统的实现，确定 CPU 的内部结构，包括运算器的功能和组成，控制器的组成及它们的连接方式和数据通路，同时也确定时序系统的构成

3. 在以上基础上，分析每条指令的执行过程，按机器周期顺序，写出所必须发送的微操作控制信号序列

4. 综合每个微操作控制信号的逻辑函数，并化简和优化

5. 用逻辑电路实现

根据上述的 5 个步骤，下面仍以本章图 7-1 的计算机组成框图和 7.1.2 节的 ADD 和 JMP 指令执行为例来介绍硬布线控制器的设计方法。

表 7.10　ADD 指令和 JMP 指令的格式及内容

指令地址	指令机器码	助记符
0000 0100	0101 0000	ADD R_0, 06H
0000 0101	0000 0110（立即数）	
0000 0110	1000 0000	JMP 04H
0000 0111	0000 0100（转移地址）	

从表 7.10 中看出，ADD 和 JMP 指令均为二字长（本例字长等于字节）指令。

ADD 指令的第一个字是操作码和寄存器地址，第二个字是立即数。也就是说 ADD 指令的一个源操作数为立即数寻址，另一个操作数也即目标操作数为寄存器寻址。将 ADD 指令执行分为 6 个机器周期完成。第一个机器周期 M_0 实现将当前指令地址（程序计数器内容）送地址寄存器，且程序计数器指向指令下一字地址；第二个机器周期 M_1 完成从内存中取出指令送指令寄存器，并由指令译码器译码；第三个机器周期 M_2 实现将指令第二个字的地址送地址寄存器，且使程序计数器指向下一条指令地址；第四个机器周期 M_3 完成从内存取出指令的第二个字即立即数，并送运算单元 ALU；第五个机器周期 M_4 完成将另一个操作数 R_0 的内容送运算单元 ALU；第六个机器周期 M_5 完成运算单元的加法操作并将结果存放在寄存器 R_0。

JMP 指令第一个字是操作码，第二个字是转移的直接地址。将 JMP 指令执行分为 5 个机器周期完成。第一和第二个机器周期完成与 ADD 指令一样（该机器所有指令都相同）的操作，也

就是取指令的操作。第三个机器周期 M_2 实现将指令的第二个字地址送地址寄存器，且使程序计数器指向下一条指令地址；第四个机器周期 M_3 完成从内存取出指令的第二个字即转移地址暂存 ALU；第五个机器周期 M_4 将转移地址送 PC，实现转移。

假设在该计算机中指令执行为异步控制，执行时间以 CPU 周期（机器周期）M_n 表示，则：

1. 首先列出 ADD 和 JMP 指令的执行过程如下：冒号“：”左边为条件，右边为执行的操作

（1）ADD 指令：

M_0：PC→AR，PC+1→PC；（取指令地址）

M_1：RAM→IR；（取指令并译码）

ADD·M_2：PC→AR，PC+1→PC；（取指令第二字地址）

ADD·M_3：RAM→ALU；（取数据）

ADD·M_4：R_i→ALU；（送寄存器数据）

ADD·M_5：ALU（+）→R_i；（计算并存结果）

（2）JMP 指令：

M_0：PC→AR，PC+1→PC；（取指令地址）

M_1：RAM→IR；（取指令并译码）

JMP·M_2：PC→AR，PC+1→PC；（取指令第二字地址）

JMP·M_3：RAM→ALU；（取转移地址）

JMP·M_4：ALU→PC；（执行转移）

2. 对应的每个机器周期所必须发送的微操作控制信号序列如下

（1）ADD 指令：

M_0：PC→IB，IB→AR，PC+1；

M_1：MEMR#（内存读），IB→IR；

ADD·M_2：PC→IB，IB→AR，PC+1；

ADD·M_3：MEMR#，IB→ALU；

ADD·M_4：R_i→IB，IB→ALU；

ADD·M_5：ALU（F=A+B），IB→R_i；

（2）JMP 指令：

M_0：PC→IB，IB→AR，PC+1；

M_1：MEMR#，IB→IR；

JMP·M_2：PC→IB，IB→AR，PC+1；

JMP·M_3：MEMR#，IB→ALU；

JMP·M_4：ALU（F=A），IB→PC；

3. 对所有的微操作控制信号进行综合：即对于某一个微操作控制信号，将上述列表中，凡是在冒号“：”右边出现该信号的机器周期，把其左边的条件（与项）作为一个或项，全部进行或运算，即得到该微操作控制信号的逻辑函数

PC→IB=（M_0+ ADD·M_2+ JMP·M_2+……）

IB→AR=（M_0+ ADD·M_2+ JMP·M_2+……）

$$\begin{aligned}
PC+1 &= (M_0+ADD\cdot M_2+JMP\cdot M_2+\cdots\cdots)\\
\overline{MEMR} &= \overline{(M_1+ADD\cdot M_3+JMP\cdot M_3+\cdots\cdots)}\\
IB\rightarrow IR &= (M_1+\cdots\cdots)\\
IB\rightarrow ALU &= ADD\cdot M_3+ADD\cdot M_4+JMP\cdot M_3+\cdots\cdots\\
R_i\rightarrow IB &= ADD\cdot M_4+\cdots\cdots\\
RAM\rightarrow ALU &= ADD\cdot M_3+JMP\cdot M_3+\cdots\cdots\\
ALU(+)\rightarrow R_i &= ADD\cdot M_5+\cdots\cdots\\
ALU\rightarrow PC &= JMP\cdot M_4+\cdots\cdots\\
ALU(F=A+B) &= ADD\cdot M_5+\cdots\cdots\\
IB\rightarrow R_i &= ADD\cdot M_5+\cdots\cdots\\
ALU(F=A) &= JMP\cdot M_4+\cdots\cdots\\
IB\rightarrow PC &= JMP\cdot M_4+\cdots\cdots\\
&\cdots\cdots
\end{aligned}$$

注意，由于指令系统中还有其他的指令未在此列出，因此，每个微操作控制信号的逻辑函数中可能还有其他的或项，以省略号代替；若某个微操作控制信号必须在某个机器周期内的 Tn 时刻有效，则该信号表达式还要与上 Tn 时钟周期信号；同时也可能还有其他的微操作控制信号，也必须按照上述方法全部求出其逻辑表达式。

4. 最后，对逻辑函数优化和简化，使得逻辑电路最简，用硬件电路实现即可

7.5.4　硬布线控制器与微程序控制器的比较

如前所述，硬布线控制器和微程序控制器的根本区别在于微操作控制信号的产生方法不同：前者由组合逻辑电路即时产生；后者是从控存读取并送出。因此，硬布线控制器的电路繁琐、不规整，不易修改和扩充；但它执行速度快，多应用于 RISC 系统。而微程序控制器由于控制信号存于控存，电路相对规整，易修改和扩充；执行速度相对硬布线控制器慢，多应用于 CISC 系统。

7.6　流水线基本工作原理

上面已对控制器的组成作了全面的讨论，分析各条指令的执行过程发现，机器的各个部分在某些周期内进行操作，而某些周期内是空闲的。例如，在取指令和取操作数阶段，执行部件运算器处于空闲状态；而在运算器执行运算时，存储器又处于空闲状态。如果控制器能调度恰当，使各个部件连续紧张工作，就可以提高计算机的运算速度。采用流水线工作原理，同时运行两条或多条指令，使这些指令处于指令运行的不同阶段，使各个部件都同时在工作，虽然每条指令的执行时间并未缩短，但 CPU 运行指令的速度却可以成倍提高。

前面讨论的控制器执行程序时是按顺序方式进行的，即程序中各条机器指令是按串行方式执行，如一条指令分取指令、计算操作数地址、取操作数、执行运算 4 个周期来完成，串行方

式的执行过程如下：

取指 1	计算地址 1	取数 1	运算 1	取指 2	计算地址 2	取数 2	运算 2	……

现在把若干条指令在时间上重叠起来进行如图 7.31 所示，则大大提高程序的执行速度。

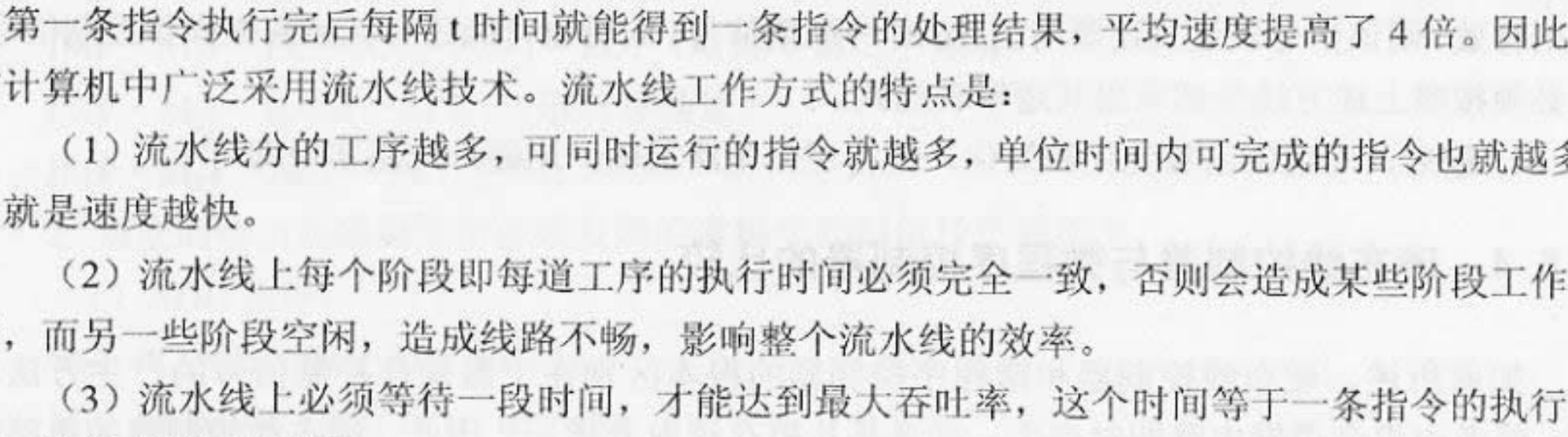


图7.31　3条指令重叠执行

图 7.31 是四级流水线，可以看出，当指令部件完成取指令后，交给运算部件计算操作数地址，同时进行取第二条指令；在第一条指令进行取操作数时，第二条指令在进行计算地址，同时第三条指令又在进行取指令。若每一阶段所需要的时间为 t，则一条指令的执行时间为 4t，但当第一条指令执行完后每隔 t 时间就能得到一条指令的处理结果，平均速度提高了 4 倍。因此目前计算机中广泛采用流水线技术。流水线工作方式的特点是：

（1）流水线分的工序越多，可同时运行的指令就越多，单位时间内可完成的指令也就越多，也就是速度越快。

（2）流水线上每个阶段即每道工序的执行时间必须完全一致，否则会造成某些阶段工作积压，而另一些阶段空闲，造成线路不畅，影响整个流水线的效率。

（3）流水线上必须等待一段时间，才能达到最大吞吐率，这个时间等于一条指令的执行时间，称为“通过时间”。

（4）当编译形成的程序不能发挥流水线的作用，或存储器供应不上流动所需的指令和数据，或遇到程序转移指令等情况时，会造成流水线断流，使效率下降。

从流水线的特点可知，采用流水线会使得指令执行速度明显加快，但也有一些问题需要解决。

（1）访问内存的冲突。从图 7.31 中可以看出，取数 1 和取指令 3 在同一时间进行，就存在访问内存冲突的问题。为了解决这个问题，第一可以采用双端口存储器，使取指令和取操作数可同时进行；第二采用相互独立的指令 Cache 和数据 Cache，使取指令和取操作数在 Cache 这级可以同时进行；第三采用指令队列，将指令从存储器中预取到指令队列中，从而避免与取数据发生冲突。

（2）计算地址与运算的冲突。从图 7.31 中看出，第一条指令在运算 1 时要使用运算器，而第三条指令计算地址 3 也要用到运算器。解决的办法是增设专门的操作数地址计算部件，用来形成操作数地址，避免与指令执行的运算冲突。

（3）操作数相关问题。如果图 7.31 中的第二条指令的取数 2 要取的是第一条指令的运算 1 结果，或者第三条指令的取数要取的操作数是第二条指令的运算 2 结果，这些都称为操作数相关。其解决方法是：

①采用推迟上述第二条指令或第三条指令执行的方法，等待前一条指令运算结果产生。该方法较简单，但降低了指令执行效率。

②采用数据旁路技术，在前一条指令执行完毕，还未存入存储器或寄存器前先由数据旁路转给后一条指令处理。即后一条指令的取数直接从数据旁路获得，而不用通过地址读内存或寄存器得到。这种方法效率高但控制复杂。

（4）转移相关问题。当程序中出现条件转移指令时，一般转移条件要等前一条指令执行完后才能建立。因此产生转移相关问题，条件转移指令必须等到前一条指令执行结束，才能确定下一条指令的地址。造成流水线无法流动。解决方法是：

①加入空操作，或尽量调整指令执行的先后顺序，使转移条件提前建立。前者相当于顺序操作，后者增加了控制难度。

②采用猜测法，先选定转移分支中的一个，按此分支继续取指并处理，假如条件码生成后，说明猜测是正确的，则流水线可以继续进行，效率得到提高；假如猜测错了，则要返回分支点，并要保证分支点后已进行个处理不破坏原有现场。

（5）中断发生时转中断处理。当程序在运行过程中，有外部设备中断请求或机器故障时，要求中止当前程序的执行而转入中断处理。在流水线中存在几条指令，如何断流进行程序切换？可以采用不精确断点法和精确断点法。不精确断点法是一种简化的断流方法，当中断发生时，不允许后续指令进入流水线，但已进入指令流水线的指令继续执行直到完毕，然后将中断处理程序的指令送入指令流水线。精确断点法是，当中断发生时，指令流水线中的指令立即停止执行，将中断处理程序指令送入指令流水线，尽快转入中断处理。目前大多数流水线计算机均采用后者。

7.7　Pentium Ⅱ CPU

7.7.1　Pentium Ⅱ CPU 的技术性能

Pentium Ⅱ CPU 采用了 P6（Pentium Pro）核心结构，继承了 Pentium Pro 优异的 32 位微处理器性能，同时利用 MMX 技术加强对多媒体的支持，并对 16 位代码进行了优化。

MMX 技术的基础是单指令流多数据流（SIMD）技术，可以并行处理 8 个 8 位数据或 4 个 16 位数据或 2 个 32 位数据。这种并行操作技术和 Pentium Ⅱ 的超标量体系结构，极大地提高处理器的性能。MMX 技术新增加了 4 种数据类型：紧缩字节类型——8 个字节打包成一个 64 位长的数据、紧缩字类型、紧缩双字类型和四字类型，还增加了 57 条新指令，另外拥有 8 个 64 位的 MMX 寄存器。

Pentium Ⅱ CPU 借鉴了 RISC 技术来实现传统的 x86 指令系统。它把每一条 x86 操作都转换成简单的微操作，然后用动态执行技术和寄存器重命名等 RISC 类处理器所采用的技术对这些微操作进行处理。

动态执行技术通过预测指令流来调整指令的执行，并且分析程序的数据流来选择指令执行

的最佳顺序。它包括以下几项技术：

1. **多路分支预测**：利用先进的、预测正确率高达90%的分支预测技术，允许程序的几个分支流向同时在处理器中执行。处理器在取指令时，还会在程序中寻找未来要执行的指令，加快了向处理器传递任务的过程，并为指令执行顺序的优化提供了可调度基础。

2. **数据流分析**：处理器读取指令并经过译码后，判断该指令能否与其他指令一起处理，然后处理器分析这些指令的数据相关性和资源可用性，以优化的执行顺序高效率地处理这些指令。

3. **推测执行**：将多个程序流向的指令序列，以调度好的优化顺序送至执行部件执行，尽量使多端口、多功能的执行部件保持“忙”状态。因为程序流向是根据分支预测建立的，因此指令序列的执行结果只能作为“预测结果”保留。一旦确定分支预测正确，已提前建立的“预测结果”立即变成“最终结果”并及时修改机器的状态。显然，推测执行可保证处理器的超标量流水线始终处于忙碌，加快了程序执行速度，从而全面提高处理器的性能。

Pentium II CPU 采用了双独立总线结构，其中前端总线 FBS 主要负责与主存储器的信息传送操作，后端总线连接到 L2 Cache 上。在 Pentium II CPU 中使用了一种 CPU 芯片外的 512KB L2 Cache，它可以在 CPU 一半的时钟频率下运行。Pentium II CPU 片内 L1 Cache 由原来的 16KB 容量扩大到了 32KB（其中 16KB 为代码 Cache，另 16KB 为数据 Cache），从而有效地减少了对 L2 Cache 的调用频率。

Pentium II CPU 也和其他 x86 CPU 一样，使用 IA（Intel 架构）指令和寄存器。但 IA 寄存器只有 16 个通用寄存器，即 8 个 32 位通用寄存器和 8 个浮点寄存器。太少的通用寄存器使相近的两条指令争用同一寄存器的可能性较大，不利于超标量流水线的执行。因此 Pentium II CPU 借鉴 RISC 类寄存器多的特点，配备了 40 个内部寄存器。采用寄存器重命名技术，将 IA 指令使用的 IA 寄存器映射成微操作使用的 Pentium II 内部寄存器，这样可极大地消除指令的数据相关性。

7.7.2 Pentium II 的内部结构及工作原理

图 7.32 是 Pentium II 的内部结构图。从图中可以看到，Pentium II 的核心功能单元包括了总线接口单元（BIU）、指令预取单元（FIU）、分支目标缓冲区（BTB）、x86 指令译码器、微指令序列器（MIS）、寄存器别名表（RAT）、保留站（RS）、指令重排缓冲区（ROB）、L1 指令 Cache、L1 数据 Cache、存储器排序缓冲区（MOB）以及若干个执行单元等。其中 X86 指令译码器包括两个简单指令译码器和一个复杂指令译码器，微指令序列器是将 x86 的 CISC 指令转换成内部微操作码的关键部件。可见，共有 3 个并行的指令单元（超标量结构的特征），可在一个周期内并发执行 3 条简单指令。

保留站有 5 个端口，它们与执行单元相连，并行地处理不同类型的微操作指令流。其中端口 0 连接 5 个执行单元，端口 1 连接 4 个执行单元，端口 2 连接加载地址单元，端口 3 连接存储地址单元，端口 4 连接存储数据单元。

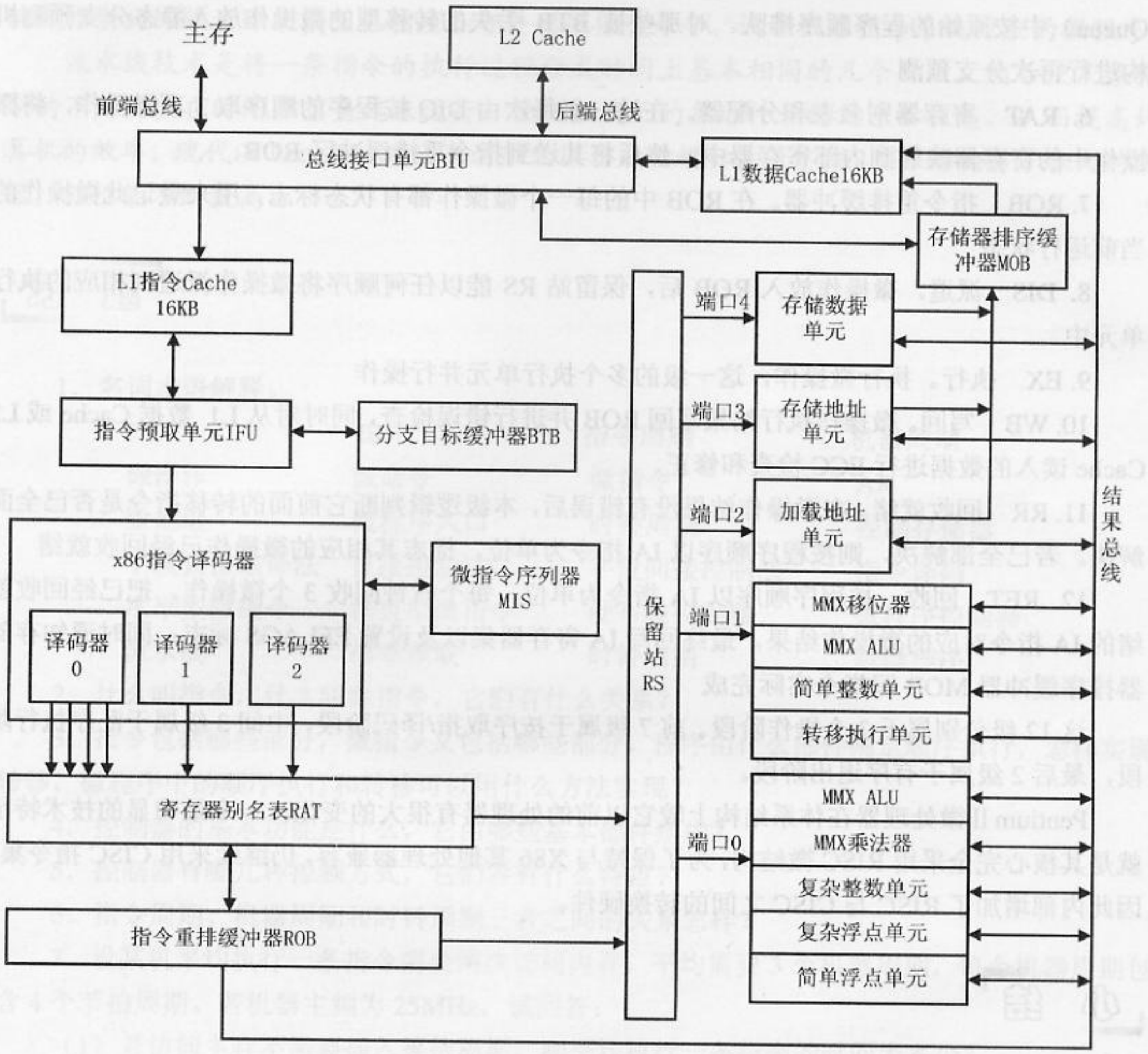


图7.32　Pentium Ⅱ微处理器内部结构框图

Pentium Ⅱ共有 3 条指令流水线，每条指令流水线共有 12 级，它们分别是：

1. IFU1　取指 1，由 L1 代码 Cache 取一个 32 字节的行，装入预取流式缓冲器

2. IFU2　取指 2，预取流式缓冲器中的内容以 16 字节块向前传递，IFU2 在 16 字节块中标志指令边界，并将发现的转移指令地址交给分支目标缓冲区 BTB 进行动态预测

3. IFU3　取指 3，旋转 16 字节块中的 3 条指令，使它们能按照复杂、简单、简单或简单、简单、简单的次序同时递交到下一级的 3 个译码器中

4. DEC1　译码 1，将 IA 指令译码成 RISC 型的微操作。译码器 0 为复杂译码器，它将一条复杂指令译码成多达 4 个微操作，译码器 1 和译码器 2 均为简单译码器，各只能生成一个微操作。DEC1 把这些微操作提交给微指令序列器

5. DEC2　译码 2，从 DEC1 出来的微操作，在本级译码后指令队列 DIQ（Decoded Instruction

Queue）中按原始的程序顺序排队。对那些被 BTB 丢失的转移型的微操作放入静态分支预测机构进行再次分支预测

6. RAT　寄存器别名表和分配器。在这一级每次由 DIQ 按程序的顺序取 3 项微操作，将微操作中的寄存器映射到内部寄存器中，然后将其送到指令重排缓冲区 ROB

7. ROB　指令重排缓冲器。在 ROB 中的每一个微操作都有状态标志，用来登记此微操作的当前运行状态

8. DIS　派遣，微操作放入 ROB 后，保留站 RS 能以任何顺序将微操作派遣到相应的执行单元中

9. EX　执行。执行微操作，这一级的多个执行单元并行操作

10. WB　写回。微操作执行结果写回 ROB 并进行错误检查，同时对从 L1 数据 Cache 或 L2 Cache 读入的数据进行 ECC 检查和修正

11. RR　回收就绪。在微操作结果没有错误后，本级逻辑判断它前面的转移指令是否已全面解决。若已全部解决，则按程序顺序以 IA 指令为单位，标志其相应的微操作已经回收就绪

12. RET　回收。按程序顺序以 IA 指令为单位，每个时钟回收 3 个微操作。把已经回收就绪的 IA 指令对应的微操作结果，最终回写 IA 寄存器集以及设置 EFLAGS 标志；同时通知存储器排序缓冲器 MOB 写指令实际完成

这 12 级分别属于 3 个操作阶段。前 7 级属于按序取指/译码阶段，中间 3 级属于乱序执行阶段，最后 2 级属于有序退出阶段。

Pentium II 微处理器在体系结构上较它以前的处理器有很大的变化，一个最明显的技术特征就是其核心完全采用 RISC 微结构，为了保持与 X86 其他处理器兼容，仍继续采用 CISC 指令集，因此内部增加了 RISC 与 CISC 之间的转换硬件。

小　结

控制器是计算机硬件的核心部件，是根据机器指令产生执行指令时全机所需要的操作控制信号，协调控制计算机各个部件有序工作。指令的执行阶段由其操作码决定，通过分析指令的各种微操作，从而形成控制器的设计思路。

控制器的设计思路有两条：一种是微程序控制器的设计方法，它是将机器指令根据其执行步骤分成若干条微指令，指令执行时从控制存储器中依次取出这些微指令，发出指令所需要的全部微操作控制信号，从而完成指令的执行。微程序控制器设计的关键是指令译码形成微程序入口、微指令格式设计及确定微指令流（后继微指令地址）的方法。另一种是硬布线控制器的设计方法，它是将指令执行时的各个机器周期的微操作信号用时序逻辑电路来实现，硬布线控制器速度快，但设计复杂繁琐，适合于 RISC 结构。微程序控制器相对硬布线控制器速度慢，但设计比较规整，易于实现指令系统修改，适合于 CISC 结构。

本章介绍了模型机的微程序控制器设计方法，可以通过 Yy-02 型计算机组成原理实验系统的实验，了解掌握模型机结构原理，掌握微程序设计技术的具体方法，设计机器指令，

确定指令的操作步骤及模型机相应各部件的微操作命令，设计机器指令系统对应的微程序。

流水线技术是将一条指令的执行过程分成时间上基本相同的几个阶段，然后使几条指令的不同阶段在时间上重叠起来进行，使流水线上的各部件始终处于忙状态，从而提高计算机的效率。现代以 Pentium 为代表的微处理器采用超标量流水线，CPU 结构有了很大的变化，其性能不断提高。

习 题

1. 名词术语解释。

PC	IR	指令周期	机器周期
微操作	微命令	微指令	AR
微地址	微程序入口	时钟周期	控制存储器
字段直接编译法	直接控制法	字段间接控制法	指令译码
水平型微指令	垂直型微指令	硬布线控制器	微程序控制器
流水线	指令预取	时钟周期	毫微程序

2. 什么叫指令，什么叫微指令，它们有什么关系？

3. 指令包括哪些部分，微指令又包括哪些部分，程序由什么部件确定顺序执行，怎样实现转移，微程序中的顺序执行和转移可以用什么方法实现？

4. 控制器的基本功能是什么，它由哪些基本部件组成？

5. 控制器有哪几种控制方式，它们各有什么特点？

6. 指令周期、机器周期和时钟周期三者之间的关系怎样？

7. 设某机平均执行一条指令需要两次访问内存，平均需要 3 个机器周期，每个机器周期包含 4 个节拍周期。若机器主频为 25MHz，试回答：

（1）若访问主存不需要插入等待周期，则平均执行一条指令的时间为多少？

（2）若每次访问内存需要插入 2 个等待节拍周期，则平均执行一条指令的时间又是多少？

8. 设某机主频为 8MHz，每个机器周期包含 4 个节拍周期，该机平均指令执行速度为 1MIPS。试回答：

（1）该机的平均指令周期是多少时间？

（2）平均每条指令周期包含几个机器周期？

9. 根据图 7.11 的数据通路，指令“INC R1”将 R1 寄存器的内容加 1，画出其指令周期微程序流程图，并根据表 7.2 和图 7.12 写出每一条微指令码。

10. 根据图 7.11 所示的模型机结构和数据通路，写出以下指令从取址到执行的全部微操作序列，说明各条指令需要哪几个机器周期，需要几次访问内存及完成什么操作。

（1）SUB A，R 指令完成（A）－R→（A），源操作数一个为寄存器寻址，目标操作数为指令提供的内存有效地址 A。

（2）JMP 偏移量，该指令完成 PC＋偏移量→PC。

11．假设某机器主要部件有：程序计数器 PC、指令寄存器 IR、通用寄存器 R0～R3、暂存器 DD1 和 DD2、ALU、移位器、存储器地址寄存器 MAR 及存储器 M。

（1）要求采用单总线结构，画出包含上述部件的逻辑框图，并注明数据流动方向。

（2）画出 ADD （R1），（R2）指令在取指和执行阶段的操作步骤流程图。R1 寄存器存放目标操作数地址，R2 寄存器存放源操作数地址。

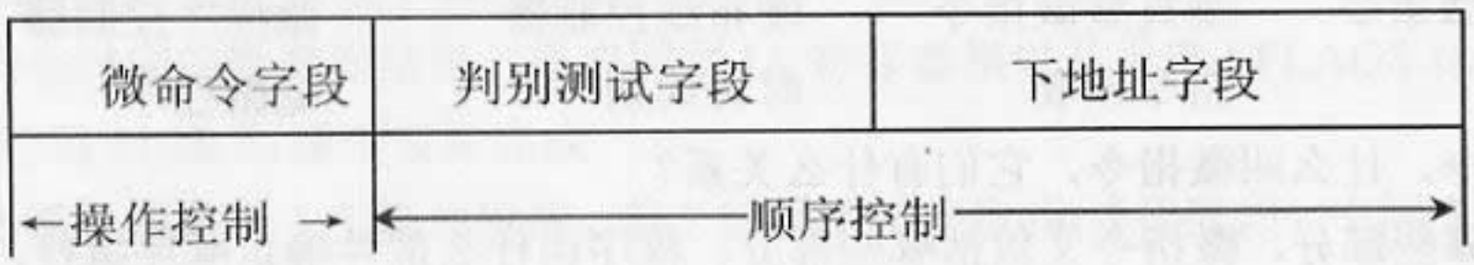
（3）写出各操作步骤所需的全部微操作命令。

12．假设某机共有 76 条指令，平均每一条指令由 11 条微指令组成，其中有一条取指令的微指令是所有指令公共的，该机共有微命令 31 个，微指令的微操作码采用直接控制法。试问：

（1）该机微指令长度为多少？

（2）控制存储器的容量应为多少？

13．某机采用微程序控制方式，其存储器容量为 512×48（位），微程序在整个控制存储器中实现转移，可控制微程序的条件共 4 个，判别测试字段采用编译法。微指令采用水平型格式，后继微指令地址采用判定方式，如下所示：



（1）微指令中的 3 个字段分别应为多少位？

（2）画出对应这种微指令格式的微程序控制器逻辑框图。

14．什么叫互斥微命令？微指令控制字段主要有哪几种编码方法，各有何特点？

15．图 7.33 为某模型机的微程序流程图，图中每一个框表示一条微指令。在点（1）处为指令译码后转入指令的微程序入口的多路分支点，由指令寄存器 IR 的 I_5I_4 两位来决定转入哪一个入口。在点（2）处根据状态条件 F 实现条件转移，微指令中判断测试位为直接控制法。控存容量能容纳图 7.33 所列出的微指令即可。

（1）微指令的判断测试位需要几位二进制？微指令的下址字段需要几位？

（2）在图中标出每条微指令的微地址。

（3）写出每条微指令的下址字段内容及判断测试字段码。

（4）画出微地址转移逻辑电路图。

16．试分别简述硬布线控制器设计和微程序控制器设计的设计步骤和硬件组成，比较其各自的特点。

17．顺序方式与流水线方式的主要区别是什么？流水线结构为什么可以提高计算机系统的性能？

18．Pentium II 处理器内部采用（　　）条指令流水线，每条流水线有（　）级；共有（　　）个执行单元，分别是（　　），每个周期可同时执行（　　）简单指令。

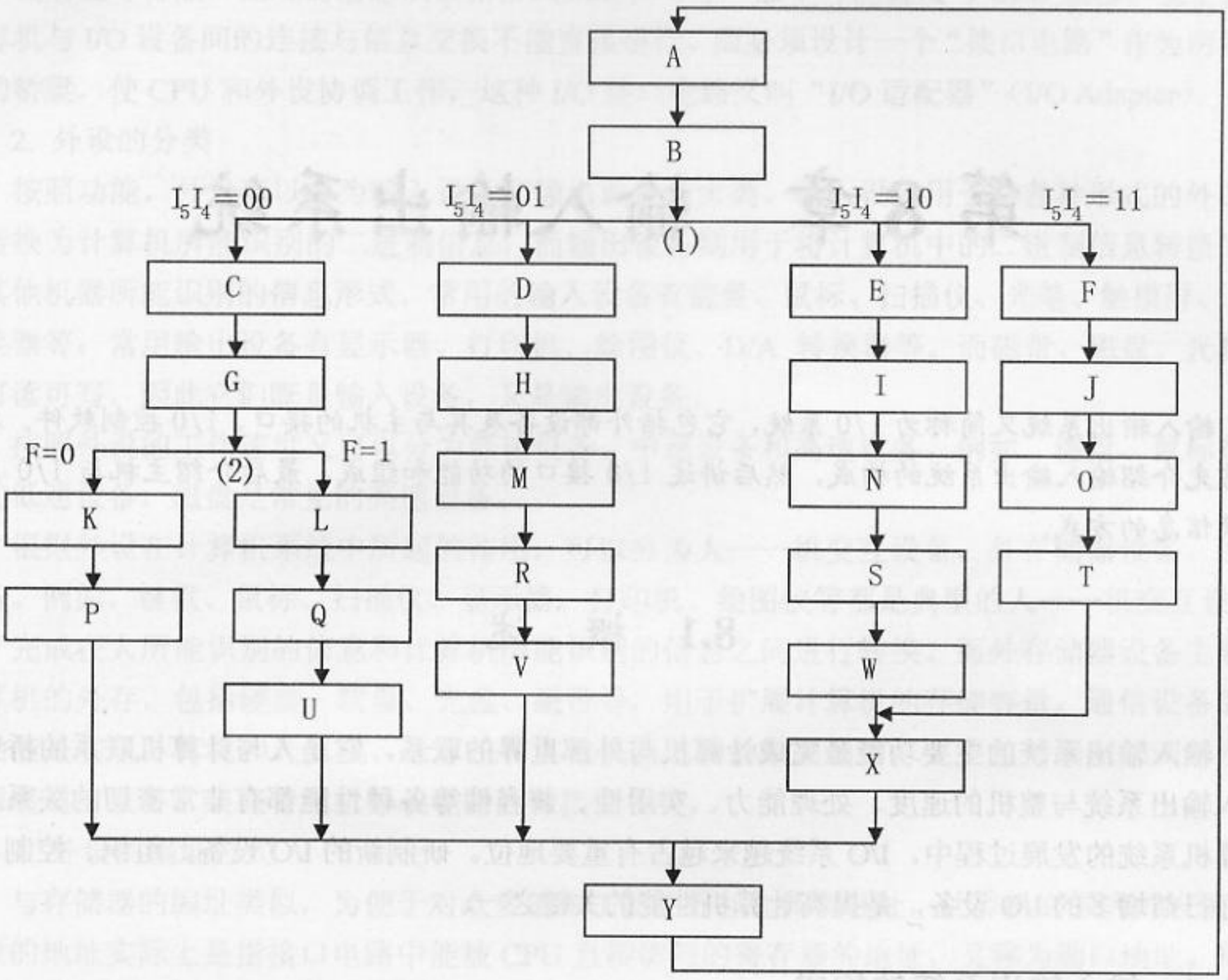


图7.33　模型机微程序流程图

世纪高等教育精品大系

ISBN 7-5341-2432-8

9 787534 124327 >

ISBN 7-5341-2432-8

定 价: 30.00元