第 5 章 中央处理器

5.1 教学目标和内容安排

主要教学目标：使学生了解 CPU 的主要功能、CPU 的内部结构、指令的执行过程、数据通路的基本组成、数据通路的定时、数据通路中信息的流动过程、控制器的实现方式、硬连线路控制器的设计、微程序控制器的设计、异常和中断的概念等，为进一步深入理解流水线 CPU 的设计原理和高级流水线技术打下基础。

基本学习要求：

(1) 理解 CPU 的功能。

(2) 了解 CPU 的基本结构。

(3) 理解 CPU 中通用寄存器和专用寄存器的作用。

(4) 理解一条指令的基本执行过程。

(5) 理解指令周期、机器周期、时钟周期的概念。

(6) 了解数据通路的基本组成。

(7) 了解数据通路中哪些是组合逻辑部件，哪些是时序逻辑部件。

(8) 了解数据通路中的组合逻辑部件和时序逻辑部件的差别。

(9) 了解寄存器堆(通用寄存器组)的作用与工作原理。

(10) 了解多路选择器的作用与工作原理。

(11) 了解 ALU 在数据通路中的功能。

(12) 了解加法器和 ALU 的差别。

(13) 了解指令存储器和数据存储器之间的差别。

(14) 了解取指阶段的数据流动过程。

(15) 了解从通用寄存器中取数的过程。

(16) 了解数据运算过程。

(17) 了解存储器取数时的数据流动过程。

(18) 了解向通用寄存器中存数时的数据流动过程。

(19) 了解如何实现条件转移的数据通路。

(20) 了解如何实现无条件转移的数据通路。

(21) 了解“0”扩展和“符号”扩展的含义和实现方式。

(22) 理解如何确定单周期数据通路的时钟周期。
(23) 理解如何确定多周期数据通路的时钟周期。
(24) 理解单周期数据通路和多周期数据通路的差别。
(25) 理解为什么很少有机器采用单周期数据通路。
(26) 理解数据通路的设计和 CPI 之间的关系。
(27) 理解指令格式的规整性对数据通路设计的影响。
(28) 理解各个控制信号的含义、控制点以及在各指令中的取值。
(29) 了解控制器的设计步骤和实现方式。
(30) 掌握如何用组合逻辑设计方式实现硬布线控制器。
(31) 了解利用微程序设计方式实现微程序控制器的基本原理。
(32) 理解内部异常和外部中断的概念。
(33) 理解为什么在设计处理器时必须考虑异常和中断的处理。
(34) 了解如何在数据通路设计中考虑异常和中断的处理。
(35) 理解内部异常和外部中断的区别。
(36) 理解常见异常事件的含义和处理方式。
(37) 理解带中断的指令执行过程。

本章是本课程的核心内容,主要介绍 CPU 中执行指令的数据通路及其控制逻辑电路的设计。其重点内容包括数据通路的定时、单周期数据通路、单周期控制器、微程序概念和带异常和中断处理的处理器实现。

主教材分 CPU 概述、单周期处理器设计、多周期处理器设计、微程序控制器设计、异常和中断处理这 5 个部分进行了介绍。

在 CPU 概述中,给出了 CPU 设计涉及的基本问题、基本概念和 CPU 设计的基本思路。这部分内容是最基础的部分,需要让学生很好地掌握。不过,对其中一些概念和知识的理解,还需要在后面具体的数据通路设计和控制器设计的学习过程中得到深化。因此,在后面单周期处理器设计和多周期处理器设计的内容介绍过程中,可以通过对具体情况的分析,强化在 CPU 概述中介绍的内容。

为了全面清楚地说明 CPU 数据通路结构的发展过程,主教材在 CPU 概述中简单介绍了早期单总线数据通路和三总线数据通路,与后面介绍的单周期数据通路、多周期数据通路以及第 7 章介绍的流水线数据通路和高级流水线数据通路串接起来,就形成了数据通路结构发展演变的技术路线图,建议教学过程中要有意识地帮助学生理解这个技术演变过程,并分析这种演变过程的原因所在。在这部分内容中,可对有关单总线数据通路的内容进行较为详细的介绍。

在介绍单周期处理器和多周期处理器设计内容时,主要以 MIPS 指令系统中有代表性的几条指令作为实现目标。其中,对单周期处理器设计内容介绍较为详细,这样做的原因有两个,第一,因为单周期处理器的结构与流水线处理器的结构比较类似,掌握单周期数据通路及其控制逻辑电路的设计方法,能更好地理解流水线处理器的设计方法。第二,因为单周期处理器设计过程比较简单,便于学生理解 CPU 设计的原理性内容。因此,建议在课堂教学中对单周期处理器的设计内容进行较为详细的介绍。在课时有限的情况下,对于多周期处理器的设计,就无须详细展开讲解,只要简单说明一下基本设计思想和基本实现原理

即可。

对于微程序控制器设计,着重讲清楚微程序控制器的基本设计思想和基本结构、微指令格式和微命令编码方式,以及微程序执行顺序的控制方式。主教材中例5.2、例5.3和例5.4都是为了便于理解基本原理而给出的,只是为了给学生提供具体示例,以达到通过对概念的具体运用来更好地理解概念的目的,对于这些例子,如果课时有限,课堂上只要大致讲一下字段如何划分,然后对其中的一个字段简单说明一下如何编码,不需要在课堂上详细展开讲解,细节问题可留给学生自学,如果学生自学时不能明白一些具体的细节问题,也没有关系,只要学生通过例子能够掌握基本原理就行了。

本章最后一个内容是异常和中断处理,是本课程和操作系统课程中最重要的概念之一,对学生将来从事处理器设计、操作系统开发和设计、嵌入式软硬件设计等都非常有用。对于这部分内容,学生普遍存在的一些问题是:分不清异常和中断在检测、响应和处理过程中的不同;分不清CPU和I/O接口中各自需要对异常和中断承担哪些职责;分不清哪些由硬件完成,哪些由软件完成,等等。因为CPU设计涉及异常和中断,所以,在本章中应该把CPU、内部异常、外部中断和输入输出这4者的关系交代清楚。主教材对内部异常和外部中断的基本概念以及异常处理过程进行了较为详细的说明,并结合多周期数据通路以及反映指令执行过程的有限状态机,对CPU中涉及异常和中断处理的功能和部件进行了说明。因为是结合具体的数据通路进行说明,学生阅读起来应该比较容易明白。

5.2 主要内容提要

1. CPU的基本功能

CPU总是周而复始地执行指令,并在执行指令过程中检测和处理内部异常事件和外部中断请求。在此过程中,要求CPU具有以下各种功能。

(1) 取指令并译码:从存储器取出指令,并对指令操作码译码,以控制CPU进行相应的操作。

(2) 计算PC的值:通过自动计算PC的值来确定下一条指令地址,以正确控制执行顺序。

(3) 算术逻辑运算:计算操作数地址,或对操作数进行算术或逻辑运算。

(4) 取操作数或写结果:通过控制对存储器或I/O接口的访问来读取操作数或写结果。

(5) 异常或中断处理:检测有无异常事件或中断请求,必要时响应并调出相应处理程序执行。

(6) 时序控制:通过生成时钟信号来控制上述每个操作的先后顺序和操作时间。

2. CPU的基本结构

CPU主要由数据通路(datapath)和控制单元(control unit)组成。

数据通路中包含组合逻辑单元和存储信息的状态单元。组合逻辑单元用于对数据进行处理,如加法器、ALU、扩展器(0扩展或符号扩展)、多路选择器以及总线接口逻辑等;状态单元包括触发器、寄存器等,用于对指令执行的中间状态或最终结果进行保存。

控制单元也称为控制器,主要功能是对取出的指令进行译码,并与指令执行得到的条件

码或当前机器的状态、时序信号等组合，生成对数据通路进行控制的控制信号。

3. CPU 中的寄存器

CPU 中存在大量寄存器，根据对用户程序的透明程度可以分成以下三类。

(1) 用户可见寄存器。

指用户程序中的指令可直接访问或间接修改其值的寄存器。包括通用寄存器、地址寄存器和程序计数器 PC。通用寄存器可用来存放地址或数据；地址寄存器专门用来存放首地址或指针信息，如段寄存器、变址寄存器、基址寄存器、堆栈指针、帧指针等；程序计数器 PC 存放当前或下条指令的地址。

(2) 用户部分可见寄存器。

指用户程序中的指令只能读取部分信息的寄存器，如程序状态字寄存器 PSWR 或标志(条件码)寄存器 FLAG，其内容由 CPU 根据指令执行结果自动设定，用户程序执行过程中可能会隐含读出其部分内容，以确定程序的执行顺序，但不能修改这些寄存器的内容。

(3) 用户不可见寄存器。

指用户程序不能进行任何访问操作的寄存器，这些寄存器大多用于记录控制信息和状态信息，只能由 CPU 硬件或操作系统内核程序访问。例如，指令寄存器 IR 用来存放正在执行的指令，只能被硬件访问；存储器地址寄存器(MAR)和存储器数据寄存器(MDR)分别用来存放将要访问的存储单元的地址和数据，也由硬件直接访问；中断请求寄存器、进程控制块指针、系统堆栈指针、页表基址寄存器等寄存器只能由内核程序访问，因此也都是用户不可见寄存器。

4. 指令执行过程

指令的执行过程大致分为取指、译码、取数、运算、存结果、查中断等几个步骤。指令周期是指取出并执行一条指令的时间，它由若干个机器周期或直接由若干个时钟周期组成。

早期的机器因为没有引入 cache，所以每个指令周期都要执行一次或多次总线操作，以访问主存读取指令或进行数据的读写，因而，将指令周期分成若干机器周期，每个机器周期对应 CPU 内部操作或某种总线事务类型，一个总线事务访问一次主存或 I/O 接口。因为一个总线事务中需要送地址和读写命令、等待主存进行读写操作等，需要多个时钟周期才能完成，所以一个机器周期又由多个时钟周期组成。

现代计算机引入 cache 后，大多数情况下都不需要访问主存，而可以直接在 CPU 内的 cache 中读取指令或访问数据，因此，每个指令周期直接由若干个时钟周期组成。时钟是 CPU 中用于控制同步的信号，时钟周期是 CPU 中最小的时间单位。

5. 数据通路的实现

现代计算机都采用时钟信号进行定时，一旦时钟边沿信号到来，数据通路中的状态单元开始写入信息。不同指令其功能不同，所以，每条指令执行时数据在数据通路中所经过的路径及其路径上的部件都可能不同。不过，每条指令在取指令阶段都一样。

根据是否将所有部件通过总线相连，可将数据通路分成总线式数据通路和非总线式数据通路；根据指令执行过程是否按流水线方式进行，可将数据通路分成非流水线数据通路和流水线数据通路。总线式数据通路无法实现指令流水线，所以，一定是非流水线数据通路。现代计算机都采用流水线数据通路。

总线式数据通路中，各个通用寄存器和 ALU 的输入、输出端之间都通过 CPU 内部总

线交换数据,因为总线上某一时刻只能传送一个部件送出的信息,所以,总线式 CPU 中执行指令时,每一步都只能串行进行,速度很慢。若所有部件连接到一个总线上,则是单总线数据通路;也可以将 ALU 的输入和输出端分别连到不同的两个总线上,构成双总线数据通路;还可以将 ALU 的两个输入端和一个输出端分别连接到三个总线上,构成三总线数据通路。

非总线式数据通路可以设计成单周期数据通路、多周期数据通路和流水线数据通路。以下用例子来说明单周期数据通路、多周期数据通路的实现。流水线数据通路的实现在下一章介绍。

6. 数据通路设计举例

以 MIPS 指令系统为例,以下概要介绍单周期数据通路和多周期数据通路的设计原理和设计步骤。

(1) 确定实现目标。为方便起见,假定数据通路实现的目标是表 5.1 所示的 11 条 MIPS 指令。

表 5.1 11 条目标指令及其功能描述

指 令	功 能	说 明
add rd,rs,rt sub rd,rs,rt	M[PC],PC←PC+4 R[rd]←R[rs] ± R[rt]	从 PC 所指的内存单元中取指令,并 PC 加 4 从 rs、rt 中取数后相加减,若溢出则异常处理,否则结果送 rd
subu rd,rs,rt	R[rd]←R[rs]−R[rt]	从 rs、rt 中取数后相减,结果送 rd(不进行溢出判断)

slt rd,rs,rt	if (R[rs]<R[rt])
	R[rd]←1
	else
	R[rd]←0
		从 rs、rt 中取数后按带符号整数来判断两数大小
		小于则 rd 中置 1,否则,rd 中清零
		(不进行溢出判断)

sltu rd,rs,rt	if (R[rs]<R[rt])
	R[rd]←1
	else
	R[rd]←0
		从 rs、rt 中取数后按无符号数来判断两数大小
		小于则 rd 中置 1,否则,rd 中清零
		(不进行溢出判断)

ori rt,rs,imm16	R[rt]←R[rs]|ZeroExt(imm16)	从 rs 取数,将 imm16 进行零扩展,然后两者按位或,结果送 rt
addiu rt,rs,imm16	R[rt]←R[rs]+SignExt(imm16)	从 rs 取数,将 imm16 进行符号扩展,然后两者相加,结果送 rt(不进行溢出判断)
lw rt,rs,imm16	Addr←R[rs]+SignExt(imm16) R[rt]←M[Addr]	从 rs 取数,将 imm16 进行符号扩展,然后两者相加,结果作为访存地址 Addr,从 Addr 中取数并送 rt
sw rt,rs,imm16	Addr←R[rs]+SignExt(imm16) M[Addr]←R[rt]	从 rs 取数,将 imm16 进行符号扩展,然后两者相加,结果作为访存地址 Addr,将 rt 送 Addr 中
beq rs,rt,imm16	Cond←R[rs]−R[rt] if (Cond eq 0) PC←PC+4+(SignExt(imm16) ×4)	做减法以比较 rs 和 rt 中内容的大小,并计算下一条指令地址,然后根据比较结果修改 PC。转移目标地址采用相对寻址,基准地址为下一条指令地址(即 PC+4),位移量为立即数 imm16 经符号扩展后的值的 4 倍。因此,转移目标指令的范围为相对于当前指令的前 32 767 到后 32 768 条指令
j target	PC<31:2>←PC<31:28>‖ target<25:0>	第一步无须进行 PC+4 而直接计算目标地址,符号‖表示“拼接”

表 5.1 给出了每条指令的 RTL 描述。每条指令的取指阶段功能都一样，都需要从 PC 所指的内存单元中取指令，并 PC 加 4。为了避免重复说明，表中省略了对取指阶段功能的描述。

图 5.1 给出了三种 MIPS 指令格式，表 5.1 中前 5 条是 R-型指令，随后 5 条是 I-型指令，最后一条跳转指令 j target 是 J-型指令。

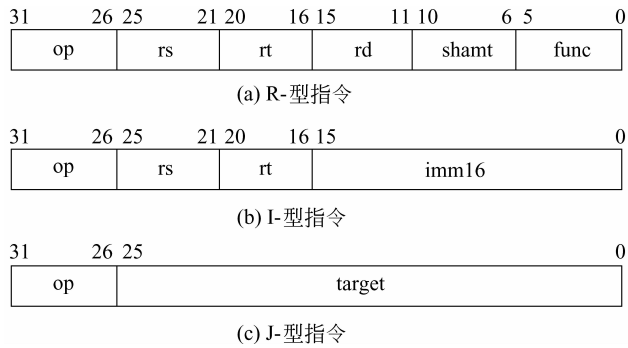

图 5.1 MIPS 指令格式

(2) 设计 ALU 电路。对目标指令中涉及的所有运算进行分析，确定用于这些运算的 ALU 及其控制电路如何设计。从表 5.1 可以看出，这 11 条指令涉及的运算包括带溢出检测的加法和减法、带符号整数大小判断、无符号数大小判断、相等判断以及各种逻辑运算等。图 5.2 给出了实现这些运算的 ALU 电路。

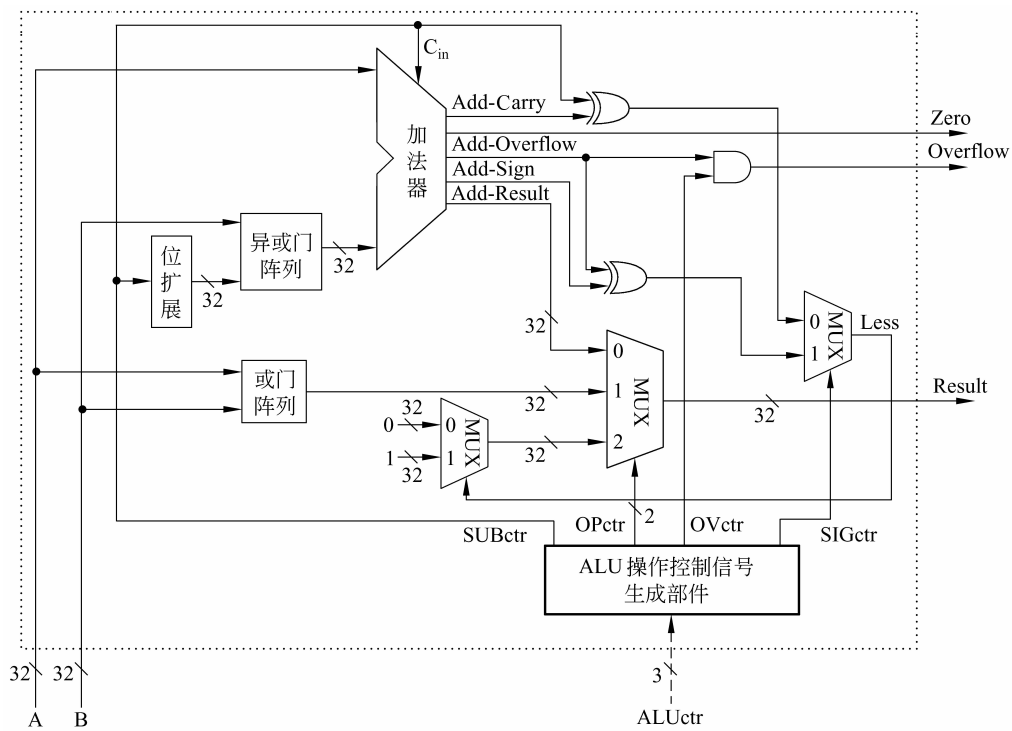

图 5.2 11 条目标指令的 ALU 实现

上述 ALU 中的核心部件是加法器,减法运算也在该加法器中实现,加法器有进位标志 Add-carry、零标志 Zero、溢出标志 Add-Overflow 和符号标志 Add-Sign。该 ALU 的输入为两个 32 位操作数 A 和 B, Result 作为 ALU 运算的结果被输出,同时,零标志 Zero 和溢出标志 Overflow 也被作为 ALU 的结果标志信息输出。

为了实现对 ALU 操作的控制,需要有相应的操作控制信号。在操作控制端 ALUctr 的控制下,图 5.2 中的"ALU 操作控制信号生成部件"用来生成各种操作控制信号,以控制在 ALU 中执行"加法""减法""按位或""带符号整数比较小于置 1"和"无符号数比较小于置 1"等运算。

(3) 设计取指令部件。取指令操作是每条指令的公共操作,其功能是取指令并计算下条指令地址。若是顺序执行,则下条执行指令地址为 PC+4;若是转移执行,则要根据当前指令是分支指令还是跳转指令分别计算转移目标地址。因为指令长度为 32 位,主存按字节编址,所以指令地址总是 4 的倍数,即最后两位总是 00,因此,PC 中只需存放前 30 位地址 PC<31:2>,在取指令时,指令地址=PC<31:2>||"00"。下条指令地址的计算方法如下。

顺序执行指令时: PC<31:2>← PC<31:2>+1。

Branch 指令中条件满足转移执行时: PC<31:2>← PC<31:2>+1+SignExt[Imm16]。

Jump 指令跳转执行时: PC<31:2>←PC<31:28>||target<25:0>。

根据上述指令地址计算方法,图 5.3 给出了完整的取指令部件。

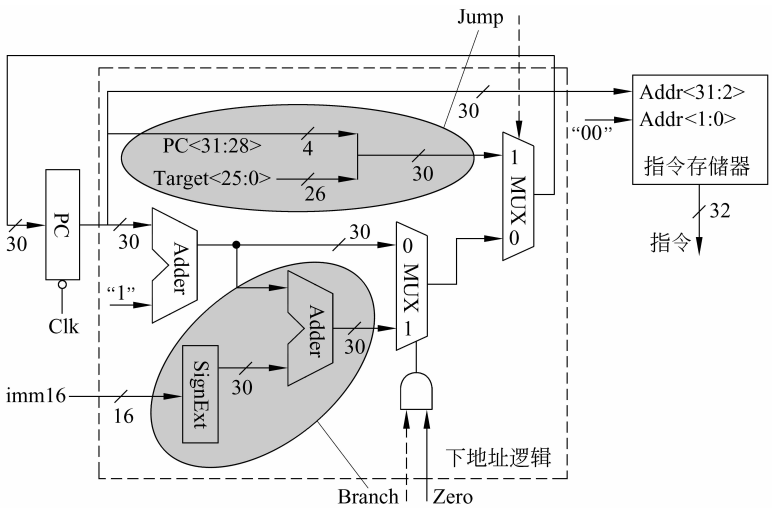

图 5.3 完整的取指令部件

取指令部件的输出是指令,输入有三个: 标志 Zero 和控制信号 Branch、Jump。下地址逻辑中的立即数 imm16 和目标地址 target<25:0>都直接来自取出的指令。分支指令时 Branch=1, Jump=0,跳转指令时 Branch=0, Jump=1,其他指令时 Branch=Jump=0。

(4) 单周期数据通路设计。11 条指令中,lw 指令最复杂,执行 lw 指令过程中数据所经过的部件最多,路径最长,因此,以它为基准设计单周期数据通路。图 5.4 给出了能够执行 11 条目标指令的完整的单周期数据通路。

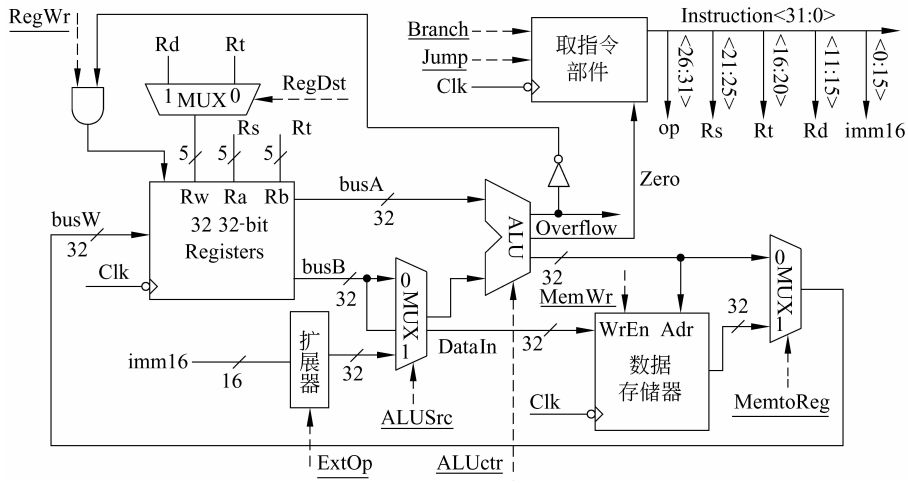

图 5.4　完整的单周期数据通路

图 5.4 中带下划线的是控制信号，用于控制数据通路的执行，由专门的控制逻辑单元根据对当前指令的译码结果生成控制信号。

(5) 时钟周期的确定。单周期处理器每条指令在一个时钟周期内完成，所以 CPI 为 1，时钟周期通常取最复杂指令所用的指令周期。在给出的 11 条指令中，最长的是 lw 指令周期。图 5.5 给出了 lw 指令执行定时过程，从图中可以看出，lw 指令周期所包含的时间为：PC 锁存延迟(Clk-to-Q)＋取指令时间＋寄存器取数时间＋ALU 延迟＋存储器取数时间＋寄存器建立时间＋时钟扭斜。

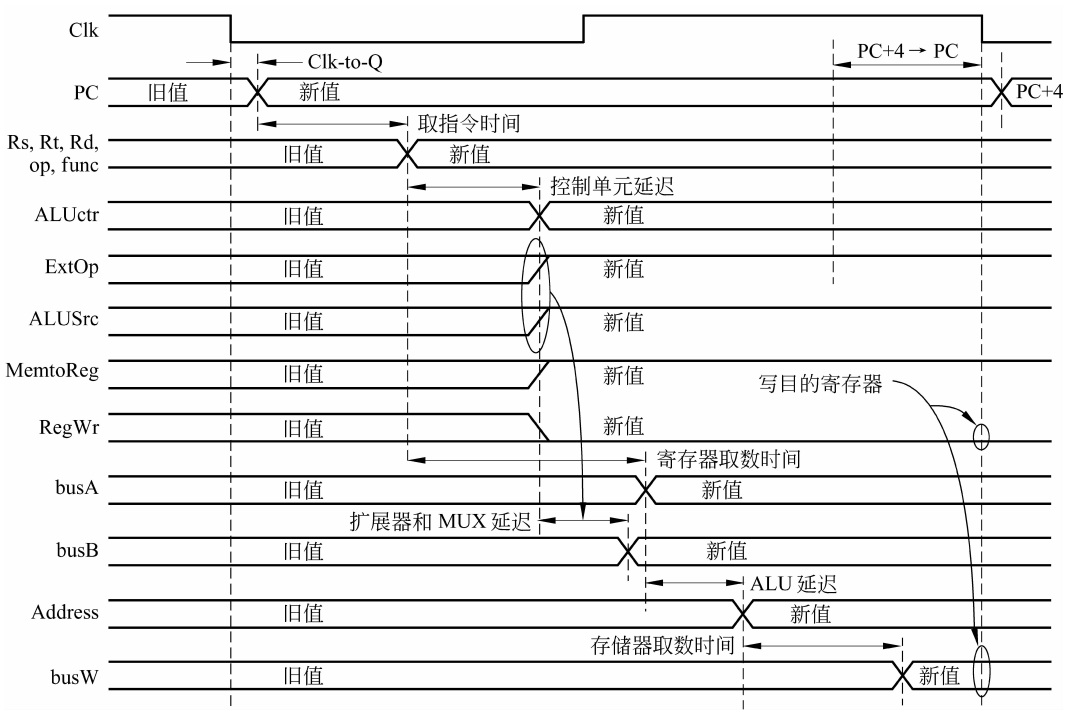

图 5.5　lw 指令执行定时

单周期处理器时钟周期远远大于许多指令实际所需执行时间,例如,R-型指令和立即数运算指令在指令执行过程中都不需要读内存;sw 指令不需要写寄存器;分支指令不需要访问内存和写寄存器;跳转指令不需要 ALU 运算和内存(寄存器)的读写。因而,单周期处理器的效率低下,性能极差,实际上,现在很少用单周期方式设计 CPU。介绍单周期数据通路,只是为了帮助理解实际的多周期和流水线两种方式。

(6) 多周期数据通路设计。多周期处理器的基本思想为:把每条指令的执行分成多个大致相等的阶段,每个阶段在一个时钟周期内完成;各阶段内最多完成一次访存或一次寄存器读/写或一次 ALU 操作;各阶段的执行结果在下个时钟到来时保存到相应存储单元或稳定地保持在组合电路中;时钟周期的宽度以最复杂阶段所花时间为准,通常取一次存储器读写的时间。图 5.6 给出了实现 11 条目标指令的多周期数据通路,其中带下划线的是控制信号。

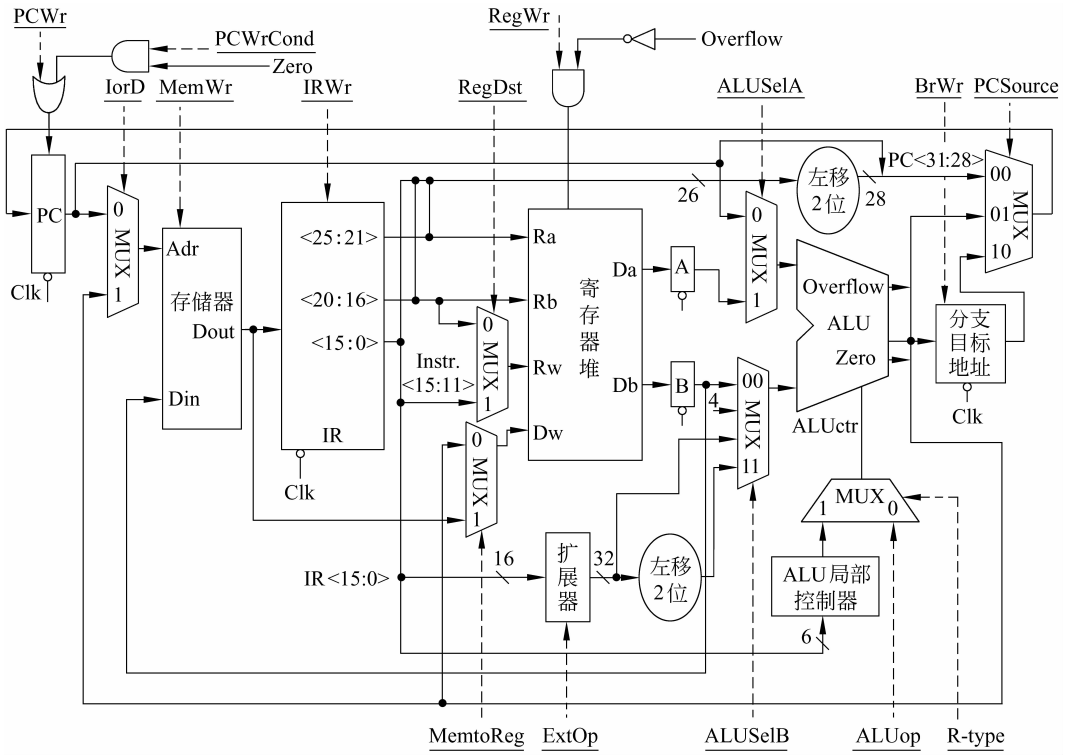

图 5.6　带控制信号的多周期数据通路

(7) 分析每条指令的执行过程,得到指令执行状态转换图。对于取指令周期(IFetch)和取数/译码周期(RFetch/ID),每条指令所进行的操作完全一样。除了取指令和译码/取数两个周期外,在 11 条目标指令中,R-型指令还需要一个执行周期(RExec)和一个结束回写周期(RFinish);ori 指令也还需要一个执行周期(oriExec)和一个结束回写周期(oriFinish);分支指令和跳转指令都是只需要一个周期,分别是 BrFinish 和 JumpFinish;lw 和 sw 共用一个主存地址计算周期(MemAdr),然后根据指令是 lw 还是 sw,确定后面是写主存周期(swFinish)还是取数周期(MemFetch)或写寄存器周期(lwFinish)。11 条指令在图 5.6 所示的多周期数据通路中执行时的状态转换过程如图 5.7 所示。

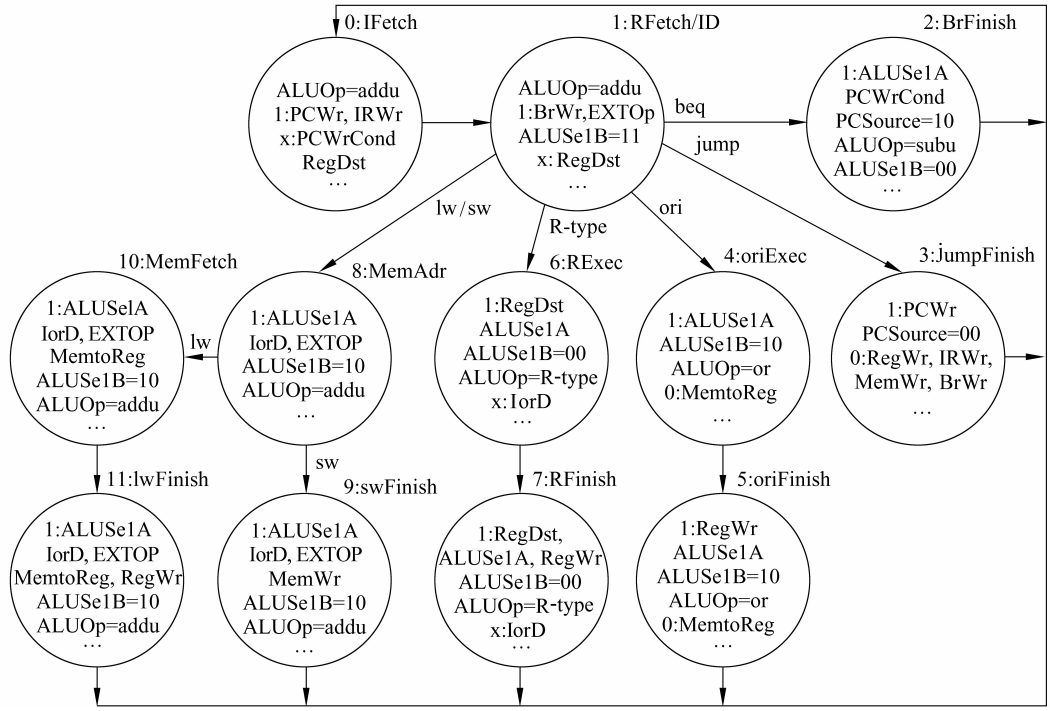

图 5.7　指令执行状态转换图

图 5.7 反映了指令在多周期数据通路中执行时的状态转换过程，每个周期对应一个状态。每来一个时钟，进入下一个执行状态。从图 5.7 可看出，R-型指令、I-型运算类指令和 sw 指令的 CPI 都是 4；lw 指令的 CPI 为 5；分支指令和跳转指令的 CPI 为 3。

7. 控制逻辑单元的实现

根据不同的控制描述方式，可以有硬连线路控制器和微程序控制器两种实现方式。

硬连线路控制器的基本实现思路：将指令执行过程中每个时钟周期所包含的控制信号取值组合看成一个状态，每来一个时钟，控制信号会有一组新的取值，也就是一个新的状态，这样，所有指令的执行过程就可以用一个有限状态转换图来描述。实现时，用一个组合逻辑电路（一般为 PLA 电路）来生成控制信号，用一个状态寄存器实现状态之间的转换。

微程序控制器的基本实现思路：将指令执行过程中每个时钟周期所包含的控制信号取值组合看成是一个 0/1 序列，每个控制信号对应一个微命令，控制信号取不同的值，就发出不同的微命令。这样，若干微命令组合成一个微指令，每条指令所包含的动作就由若干条微指令来完成。指令执行时，先找到对应的第一条微指令，然后按照特定的顺序取出后续的微指令执行。每来一个时钟，执行一条微指令。实现时，每条指令对应的微指令序列（称为微程序）事先存放在一个只读存储器（称为控制存储器，简称控存）中，用一个 PLA 电路或 ROM 来生成每条指令对应的微程序的第一条微指令地址，用相应的微程序定序器来控制微指令执行流程。微程序定序器的实现有计数器法和断定法（即下址字段法）两种。

8. 内部异常和外部中断

在程序正常执行过程中，CPU 会遇到一些特殊情况而无法继续执行当前程序。这种中断程序正常执行的情况主要有两大类：内部异常和外部中断。