第 5 章 中央处理器

计算机所有功能通过执行程序完成，程序由指令序列构成。计算机采用“存储程序”的工作方式，即计算机必须能够自动地从主存取出一条条指令执行，而专门用来执行指令的部件就是中央处理器(central processing unit，CPU)。在 CPU 中控制指令执行的部件是控制器，控制器可采用硬连线路方式实现，也可采用微程序设计方式实现，也有一些 CPU 采用硬连线路和微程序控制相结合的方式实现。

本章主要介绍 CPU 的基本功能和基本组成，以及单周期和多周期处理器的工作原理和设计方法。有关流水线处理器的基本设计原理在第 6 章介绍。

5.1 CPU 概述

5.1.1 CPU 的基本功能

CPU 的基本职能是周而复始地执行指令。CPU 在执行指令过程中可能会遇到一些异常情况和外部中断。例如，对指令操作码译码时，可能会发现有不存在的“非法操作码”；在访问指令或数据时可能发现“缺页”(即要访问的信息不在主存)；外部设备可能会请求中断 CPU 的执行等。因此，CPU 除了执行指令外，还要能够发现和处理异常情况和中断请求。

程序由指令序列和所处理的数据组成。指令按顺序存放在内存连续单元中，将要执行的指令的地址由 PC 给出。CPU 取出并执行一条指令的时间称为指令周期，不同指令的指令周期可能不同。

通常，CPU 执行一条指令的大致过程如下。

(1) 取指令。从 PC 指出的内存单元中取出指令送到指令寄存器(IR)。

(2) 对 IR 中的指令操作码译码并计算下条指令地址。不同指令的功能不同，即指令涉及的操作过程不同，因而需要不同的操作控制信号。

(3) 计算源操作数地址并取源操作数。根据寻址方式确定源操作数地址计算方式，若源操作数是存储器数据，则需要一次或多次访存，例如，对于间接寻址或两个操作数都在存储器的指令，需要多次访存；若源操作数是寄存器数据，则直接从寄存器取数，无须访存。

(4) 对操作数进行相应的运算。在 ALU 或加法器等运算部件中对取出的操作数进行运算。

(5) 目的操作数地址计算并存结果。根据寻址方式确定目的操作数的地址计算方式，

将运算结果存入存储单元中,或存入通用寄存器中。

对于上述过程的第(1)步和第(2)步,所有指令的操作都一样,都是取指令、指令译码并修改PC;而对于第(3)～(5)步,不同指令的操作可能不同,它们完全由第(2)步译码得到的控制信号控制,即每条指令的功能由第(2)步译码得到的控制信号决定。

上述这些基本操作可以用形式化的方式来描述,所用的描述语言称为寄存器传送级(register transfer level,RTL)语言。本书RTL语言规定:R[r]表示通用寄存器r的内容,M[addr]表示存储单元addr的内容;M[R[r]]表示寄存器r的内容所指存储单元的内容;PC表示PC的内容,M[PC]表示PC所指存储单元的内容;SEXT[imm]表示对imm进行符号扩展,ZEXT[imm]表示对imm进行零扩展;传送方向用←表示,即传送源在右,传送目的在左。

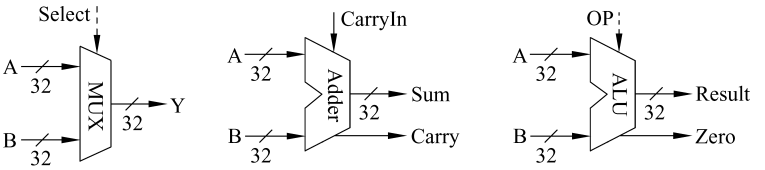
5.1.2 CPU的基本组成

随着超大规模集成电路技术的发展,更多的功能模块被集成到CPU芯片中,包括cache、MMU、浮点运算逻辑、异常和中断处理逻辑等,因而CPU的内部组成越来越复杂,甚至在一个CPU芯片中集成了多个处理器核。但是,不管CPU多复杂,数据通路(datapath)和控制器(control unit)是其两大基本组成部分。控制器也称为控制部件。

通常把数据通路中专门进行数据运算的部件称为执行部件(execution unit)。指令执行所用到的元件有两类:组合逻辑元件(也称操作元件)和存储元件(也称状态元件)。连接这些元件的方式有两种:总线方式和分散连接方式。数据通路就是由操作元件和状态元件通过总线或分散方式连接而成的进行数据存储、处理和传送的路径。

1. 操作元件

操作元件属于组合逻辑元件,其输出只取决于当前的输入。如图5.1所示,数据通路中最常用的操作元件有多路选择器(MUX)、加法器(Adder)、算术逻辑部件(ALU)等。



(a) 多路选择器　(b) 加法器　(c) 算术逻辑部件

图5.1　数据通路中的常用组合逻辑元件

图中虚线表示控制信号,多路选择器需要控制信号Select确定选择哪个输入被输出;加法器不需要控制信号控制,因为它的操作是确定的;ALU需要有操作控制信号OP,由它确定ALU进行哪种操作。

2. 状态元件

状态元件属于时序逻辑电路,具有存储功能,输入状态在时钟控制下被写到电路中,并保持电路的输出值不变,直到下一个时钟到达。输入端状态由时钟信号决定何时被写入,输出端状态随时可以读出。最简单的状态元件是D触发器,有时钟输入CLK、状态输入端D和状态输出端Q。图5.2是D触发器的定时示意图。

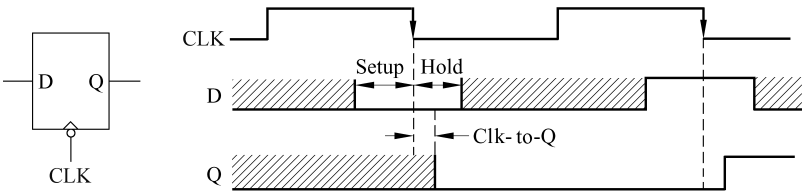


图 5.2　D 触发器定时示意

图 5.2 所示 D 触发器采用下降沿触发，要使输出状态能正确随输入状态改变，必须满足以下时间约束。

(1) 在时钟下降沿到达前一段时间内，输入端 D 必须稳定有效，这段时间称为建立时间(setup time)。

(2) 在时钟下降沿到达后一段时间内，输入端 D 必须继续保持稳定不变，这段时间称为保持时间(hold time)。

在满足上述两个约束条件的情况下，经过时钟下降沿到来后的一段锁存延迟(Clk-to-Q 时间)，输出端 Q 的状态改变为输入端 D 的状态，并一直保持不变，直到下个时钟到来。

数据通路中的寄存器是一种典型的状态存储元件，n 个 D 触发器可构成一个 n 位寄存器。根据功能和实现方式的不同，有各种不同类型的寄存器。例如：①带“写使能”输入的暂存寄存器，可用于实现指令寄存器、通用寄存器组(general purpose register set，GPRs)[①]等；②输出端带一个三态门的寄存器，通常用于与总线相连的寄存器，可通过三态门来控制信息是否打到总线上；③带复位(即清 0)功能的寄存器；④带计数(自增)功能的寄存器；⑤带移位功能的寄存器。

这些不同类型的寄存器都在时钟信号和相应控制信号(如“写使能”“三态门开启”“清 0”“自增”“左移/右移”)的控制下完成信息存储功能。图 5.3 是数据通路中的暂存寄存器和通用寄存器组的外部结构示意图。

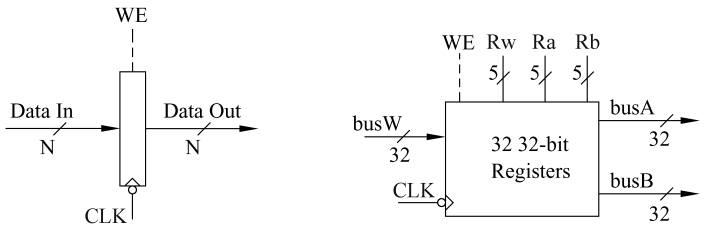


图 5.3　暂存寄存器和通用寄存器组的外部结构

1) 暂存寄存器

如图 5.3(a)所示，暂存寄存器有一个写使能(write enable)信号 WE，当 WE＝0 时，时钟信号(CLK)边沿到来不会改变输出值；当 WE＝1 时，时钟边沿到来后，经过 Clk-to-Q 时

① 通用寄存器组(general purpose register set，GPRs)，有的英文原版教材用 register files 表示，可译为寄存器组或寄存器堆。

间的延迟,输出端(DataOut)开始变为输入端(DataIn)的值,表示输入信息被写入寄存器。若数据通路中某个寄存器在每个时钟到来时都需要写入信息,则该寄存器无须 WE 信号。

2) 通用寄存器组

32 个暂存寄存器可以构成一个如图 5.3(b)所示的通用寄存器组,每个寄存器地址是一个 5 位的二进制编码。它有以下两个读口:busA 和 busB,分别由 Ra 和 Rb 给出地址。读操作属于组合逻辑操作,无须时钟信号控制,当地址 Ra 和 Rb 到达后,经过一个"取数时间"的延迟,在 busA 和 busB 上的信息开始有效。它还有一个写口:busW 上的信息写入的地址由 R_W 指定。写操作属于时序逻辑操作,需要时钟信号 CLK 的控制。在 WE 为 1 的情况下,时钟触发边沿到来后经过 Clk-to-Q 时间延迟,从 busW 传来的值开始写入 R_W 指定的寄存器中。

5.1.3 数据通路与时序控制

指令执行过程中的每个操作步骤都有先后顺序,为了使计算机能正确执行指令,CPU 必须按正确的时序产生操作控制信号。由于不同指令对应的操作序列长短不一,序列中各操作执行时间也不相同,因此,需要考虑用怎样的时序方式来控制。

1. 早期计算机的三级时序系统

早期计算机通常采用机器周期、节拍和脉冲三级时序对数据通路操作进行定时控制。一个指令周期可分为取指令、读操作数、执行并写结果等多个基本工作周期,称为机器周期。

一个机器周期内要进行若干步操作。例如,存储器读周期有送主存地址、发送读写命令、检测数据有无准备好、取数据等步骤。因此,有必要将一个机器周期再划分成若干节拍,每个动作在一个节拍内完成。为了产生操作控制信号并使某些操作能在一拍时间内配合工作,常在一个节拍内再设置一个或多个工作脉冲。

2. 现代计算机的时钟信号

现代计算机中,已不再采用三级时序系统,机器周期的概念已逐渐消失。整个数据通路中的定时信号就是时钟信号,一个时钟周期就是一个节拍。

如图 5.4 所示,数据通路可看成由组合逻辑元件和状态元件交替组合而成,即数据通路的基本结构为"… — 状态元件 — 组合逻辑元件 — 状态元件 — …"。

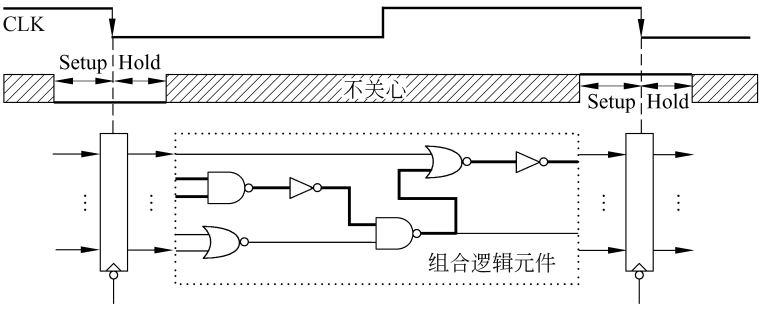


图 5.4 数据通路和时钟周期

图 5.4 所示的数据通路中,虚线框内是组合逻辑元件,所有组合逻辑电路都从状态元件接收输入,并将输出写入状态元件中。所有状态元件在同一时钟信号控制下写入信息。假

定采用下降沿(负跳变)触发,则所有状态元件在时钟下降沿到来时开始写入信息,经过触发器的锁存延迟(即 Clk-to-Q)后输出开始有效。假定每个时钟的下降沿是一个时钟周期的开始时刻,则一个时钟周期内整个处理过程如下:经过 Clk-to-Q 时间,前一个时钟周期内组合逻辑生成的信号被写入状态元件,并输出到随后的组合逻辑电路进行处理,经过若干级门延迟,得到的处理结果被送到下一级状态元件的输入端,然后必须稳定一段时间(setup time)才能开始下个时钟周期,并在时钟信号到达后还要保持一段时间(hold time)。

假定各级组合逻辑电路的传输延迟(即最长延迟)为 longest delay,考虑时钟偏移(clock skew)[①],根据上述分析可知,数据通路的时钟周期(cycle time)应为 cycle time=Clk-to-Q+longest delay+setup time+clock skew。假定各级组合逻辑电路中最短延迟为 shortest delay,为了使数据通路能正常工作,则应满足以下时间约束:Clk-to-Q+shortest delay>hold time。

5.2 单周期处理器设计

处理器设计涉及数据通路和控制电路的设计,其设计过程如下。

第 1 步:分析每条指令的功能。

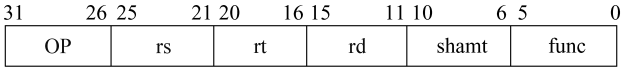
第 2 步:根据指令的功能给出所需的元件,并考虑如何将它们互连。

第 3 步:确定每个元件所需控制信号的取值。

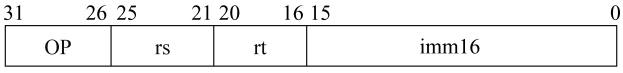
第 4 步:汇总所有指令涉及的控制信号,生成反映指令与控制信号之间关系表。

第 5 步:根据关系表,得到每个控制信号的逻辑表达式,据此设计控制电路。

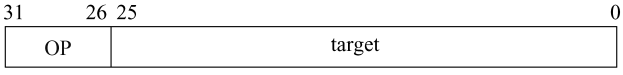
一个指令系统常常有几十到几百条指令,实现一个完整指令系统的处理器是一项非常复杂、烦琐的任务。为了能清楚说明处理器设计过程和基本原理,本节以实际的 MIPS 指令系统为例来说明。有关 MIPS 指令系统参见 4.3.1 节,图 5.5 再次给出了 MIPS 的 3 种指令格式。由于篇幅的限制,不可能介绍所有指令的实现,为此,选择了具有代表性的若干条指令作为实现目标。



(a) R-型指令



(b) I-型指令



(c) J-型指令

图 5.5 MIPS 指令格式

① 时钟偏移(clock skew):由于器件工艺和走线延迟等原因造成的同步系统中时钟信号的偏差,这种时间偏差使得时钟信号不能同时到达不同的状态元件而导致同步定时错误,所以需要在时钟周期中增加时钟偏移时间来避免这种错误。也有人把 clock skew 翻译为时钟扭斜或时钟歪斜。

本节选择以下11条MIPS指令作为实现目标。

5条R-型指令：

```
add    rd, rs, rt
sub    rd, rs, rt
subu   rd, rs, rt
slt    rd, rs, rt
sltu   rd, rs, rt
```

5条I-型指令：

```
ori    rt, rs, imm16
addiu  rt, rs, imm16
lw     rt, rs, imm16
sw     rt, rs, imm16
beq    rs, rt, imm16
```

1条J-型指令：

```
j      target
```

这些指令比较具有代表性，包含了R-型、I-型和J-型3种类型指令；既有算术/逻辑运算指令，又有取数/存数指令；既有条件转移指令，又有无条件转移指令；既有需要考虑溢出判断的指令，又有无须考虑溢出的指令；既有对带符号数判断大小的指令，又有对无符号数判断大小的指令。关于这些指令的介绍内容，涵盖了大部分指令的基本实现技术。

5.2.1 指令功能的描述

设计处理器的第一步先要确认每条指令的功能，表5.1给出了上述11条MIPS指令功能的RTL描述。其RTL描述采用5.1.1节的规定。因为每条指令的第一步都是取指令并PC加4，使PC指向下条指令，所以表中除第一条add指令外，其余指令都省略了对第一步的描述。

表5.1 11条目标指令功能的RTL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0(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0(不进行溢出判断)

续表

指　令	功　能	说　明
ori rt, rs, imm16	R[rt]←R[rs] | ZEXT(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]+SEXT(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]+SEXT(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]+SEXT(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+(SEXT(imm16)×4)	做减法以比较 rs 和 rt 中内容的大小，并计算下条指令地址，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 32767 到后 32768 条指令
j target	PC←PC<31:28>||target<25:0>||00	第一步无须进行 PC+4 而直接计算目标地址，符号||表示“拼接”，PC 最后两位为 00

5.2.2 数据通路的设计

在对所有指令进行功能分析的基础上，可以进行数据通路的设计。为简化数据通路设计，假定所用的数据存储器和指令存储器皆为一种理想存储器。如图 5.6 所示，理想存储器有一个 32 位数据输入端 DataIn，一个 32 位数据输出端 DataOut，还有一个读写公用的地址输入端 Address。控制信号有一个写使能信号 WE，写操作受时钟信号 CLK 的控制，假定采用下降沿触发，即在时钟下降沿开始写入信息。

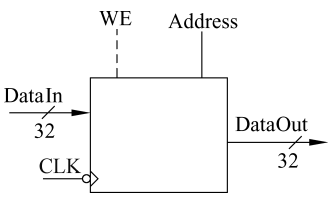


图 5.6　理想存储器外部结构

该理想存储器的读操作是组合逻辑操作，即在地址 Address 有效后，经过一个“取数时间”，数据输出端 DataOut 上数据有效；写操作是时序逻辑操作，即在 WE 为 1 的情况下，当时钟 CLK 边沿到来时，DataIn 开始写入存储单元中。

1. 算术逻辑部件的设计

上述 11 条指令涉及带溢出判断的加法和减法、带符号整数的大小判断、无符号整数的大小判断、相等判断以及各种逻辑运算等。为了支持这 11 条指令包含的运算，ALU 必须具有相应的功能。

图 5.7 给出了一个实现上述 11 条指令中运算的 ALU。该 ALU 的输入为两个 32 位操作数 A 和 B，其中，核心部件是加法器，加法器的输出除两个数的和 Add-Result 以外，还有

进位标志 Add-carry、零标志 Zero、溢出标志 Add-Overflow 和符号标志 Add-Sign。有关加法器的实现可参见第 3 章内容。在操作控制端 ALUctr 的控制下,在 ALU 中执行加、减、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,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。

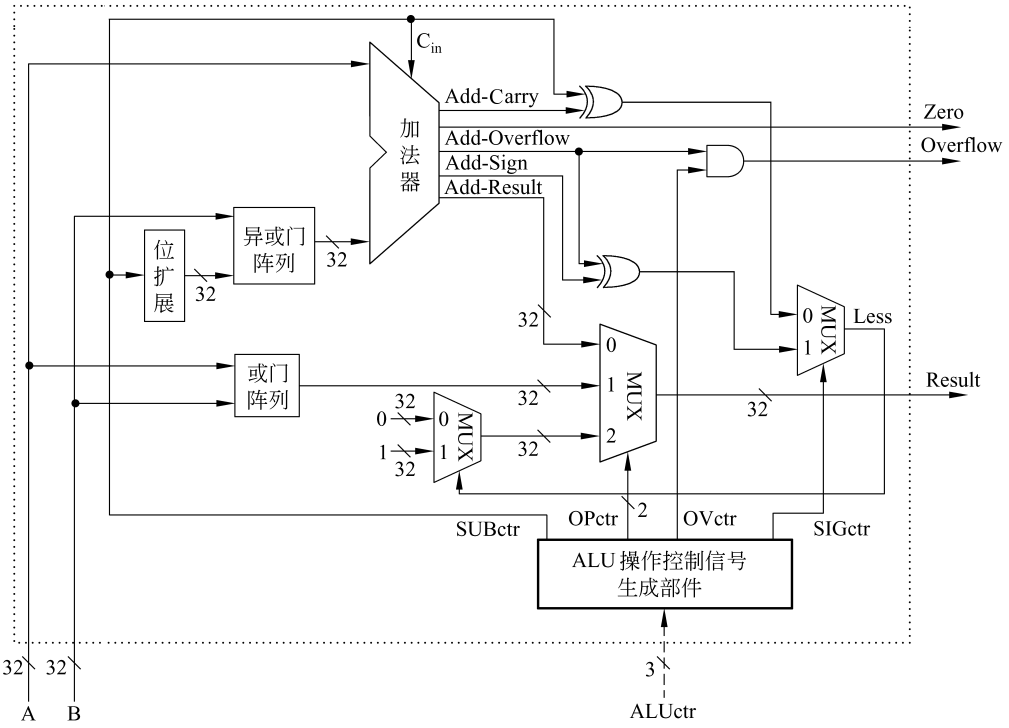


图 5.7　11 条目标指令的 ALU 实现

从图 5.7 可以看出,ALU 的操作由一个“ALU 操作控制信号生成部件”产生的控制信号来控制,其输入是 ALUctr,输出有 4 个控制信号:① SUBctr 用来控制 ALU 执行加法还是减法运算,当 SUBctr=1 时,做减法,当 SUBctr=0 时,做加法;②OPctr 用来控制选择哪种运算的结果作为 Result 输出,因为所实现的 11 条指令中只可能有加/减、按位或、小于置 1 这 3 种运算,所以 OPctr 有两位;③OVctr 用来控制是否要进行溢出判断,当 OVctr=1 时,进行溢出判断,此时,若结果发生溢出,则溢出标志 Overflow 为 1,当 OVctr=0 时,无须溢出判断,此时,即使结果发生溢出,溢出标志 Overflow 也不为 1;④SIGctr 信号控制 ALU 是执行“带符号整数比较小于置 1”还是“无符号数比较小于置 1”功能,当 SIGctr=0,则执行“无符号数比较小于置 1”,当 SIGctr=1 时,则执行“带符号整数比较小于置 1”。

根据表 5.1 列出的每条指令的功能,可以了解到各条指令在 ALU 中所进行的运算,由此可列出各条指令对应的 4 种 ALU 操作控制信号取值,如表 5.2 所示。

从表 5.2 可知,指令 addiu、lw、sw 和 beq 转移目标地址计算的 ALU 控制信号取值一样,都是进行加法运算并不判溢出,记为 addu 操作;指令 subu 和 beq 判 0 操作的 ALU 控制信号可看成一样,都做减法运算并不判溢出,记为 subu 操作。因此,这 11 条指令可以归纳为以下 7 种操作:addu、add、or、subu、sub、sltu、slt,需要 3 位对其进行编码,因而 ALU 的

表 5.2　11 条目标指令对应的 4 种 ALU 操作控制信号取值

指　令	功　能	运算类型	SUBctr	OPctr	OVctr	SIGctr
add rd, rs, rt	R[rd]←R[rs]+R[rt]	加(判溢出)	0	00	1	×
sub rd, rs, rt	R[rd]←R[rs]−R[rt]	减(判溢出)	1	00	1	×
subu rd, rs, rt	R[rd]←R[rs]−R[rt]	减(不判溢出)	1	00	0	×
slt rd, rs, rt	if (R[rs]<R[rt]) R[rd]←1 else R[rd]←0	减(不判溢出) 带符号整数比较大小	1	10	0	1
sltu rd, rs, rt	if (R[rs]<R[rt]) R[rd]←1 else R[rd]←0	减(不判溢出) 无符号数比较大小	1	10	0	0
ori rt, rs, imm16	R[rt]←R[rs]|ZEXT(imm16)	按位或(不判溢出)	×	01	0	×
addiu rt, rs, imm16	R[rt]←R[rs]+SEXT(imm16)	加(不判溢出)	0	00	0	×
lw rt, rs, imm16	Addr←R[rs]+SEXT(imm16) R[rt]←M[Addr]	加(不判溢出)	0	00	0	×
sw rt, rs, imm16	Addr←R[rs]+SEXT(imm16) M[Addr]←R[rt]	加(不判溢出)	0	00	0	×
beq rs, rt, imm16	Cond←R[rs] −R[rt]	减(判 0)	1	××	0	×
	if (Cond eq 0) PC←PC+(SEXT(imm16)×4)	加(不判溢出)	0	00	0	×
j target	PC<31:2>←PC<31:28>|| target<25:0>	无须 ALU 运算	×	××	×	×

注：×表示无论取什么值都不影响运算结果。

操作控制输入端 ALUctr 至少有三位。

在对 ALUctr 进行编码时，可以根据这些 ALU 操作和 4 种 ALU 操作控制信号的对应关系进行优化，例如，把加减控制(SUBctr)、溢出判断(OVctr)和符号控制(SIGctr)等信号分别对应到不同的位来进行控制。表 5.3 给出了 ALUctr 的一种三位编码方案。

表 5.3　ALUctr 的三位编码及其对应的操作类型和 ALU 控制信号

ALUctr<2:0>	操作类型	SUBctr	OVctr	SIGctr	OPctr<1:0>	OPctr 的含义
0 0 0	addu	0	0	×	0 0	选择加法器的结果输出
0 0 1	add	0	1	×	0 0	选择加法器的结果输出
0 1 0	or	×	0	×	0 1	选择“按位或”结果输出
0 1 1	(未用)					
1 0 0	subu	1	0	×	0 0	选择加法器的结果输出
1 0 1	sub	1	1	×	0 0	选择加法器的结果输出
1 1 0	sltu	1	0	0	1 0	选择小于置位结果输出
1 1 1	slt	1	0	1	1 0	选择小于置位结果输出

根据表5.3得到各输出控制信号的逻辑表达式如下：

```
SUBctr=ALUctr<2>
OVctr=!ALUctr<1>&ALUctr<0>
SIGctr=ALUctr<0>
OPctr<1>=ALUctr<2>& ALUctr<1>
OPctr<0>=!ALUctr<2>ALUctr<1>& !ALUctr<0>
```

根据上述逻辑表达式，不难实现图5.7中的“ALU操作控制信号生成部件”。

如果要实现更多指令，则ALU必须支持更多的运算，如取负(neg)、取反(not)、与(and)、异或(xor)、或非(nor)等，如果在ALU中考虑所有这些情况的话，需在图5.7所示的ALU中增加相应的取负、按位取反、按位与、按位异或、按位或非等逻辑电路，同时ALU输出结果选择控制信号OPctr的位数需扩充到至少3位，ALUctr的位数需扩充到4位。

2. 取指令部件的设计

从上述指令功能的RTL描述中，可以看出，每条指令的第一步都是完成取指令并计算下条指令地址的功能。因此，在数据通路中，需要专门设计一个取指令部件来完成上述功能。

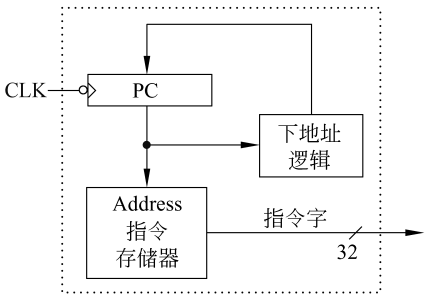


图5.8 取指令部件示意图

图5.8是取指令部件的示意图。假定指令专门存放在指令存储器中，它只有读操作，读指令操作可看成组合逻辑操作，因此无须控制信号的控制，只要给出指令地址，经过一定的“取数时间”后，指令被送出。指令的地址来自PC，有专门的下地址逻辑来计算下条指令的地址，然后送PC。因为是单周期处理器，每个时钟周期执行一条指令，所以每来一个时钟，PC的值都会被更新一次，因而，PC无须“写使能”信号控制。下地址逻辑中，要区分是顺序执行还是转移执行。若是顺序执行，则执行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来计算转移目标地址。

3. R-型指令的数据通路

图5.9是R-型指令相关的数据通路示意图，用它可以完成对两个寄存器Rs和Rt内容的运算并将结果写入Rd寄存器。像add和sub等指令还要判断结果是否溢出，只有不溢出时才写结果到Rd，否则转异常处理程序执行。

指令中Rs和Rt是两个源操作数寄存器编号，Rd是目的寄存器编号，因此，寄存器堆的两个读地址端Ra和Rb应分别与Rs和Rt相连，写地址端Rw与Rd相连。ALU运算结果连到寄存器堆的写数据端busW，控制信号RegWr为“写使能”信号，只有在RegWr信号为1且不溢出的情况下，运算结果才写入寄存器堆，显然R-型指令执行时，RegWr信号应该为1。

11条目标指令中有5条R-型指令：add、sub、subu、slt和sltu，根据表5.2可知，它们分别对应ALU的5种操作：add、sub、subu、slt和sltu，因此，可根据不同的指令，控制将不同的操作编码送到ALU操作控制端ALUctr，以便在ALU中进行不同指令所对应的运算。

当前时钟周期内执行的运算结果总是在下一个时钟到来时，开始写到寄存器堆中。为