第 3 章　微处理器

引言

本章主要讲述微处理器的基本构成，并以 MIPS 微处理器为例，介绍一个简单的基于给定指令集的类 MIPS 微处理器数据通路以及控制器的设计。同时介绍微处理器流水线技术、超标量技术和异常处理机制，最后介绍微处理器的外部接口以及一个具体的微处理器——MicroBlaze 微处理器的结构。

目标

学习完本章知识，要求掌握以下内容：

(1) 了解微处理器的基本构成；

(2) 能设计执行简单 MIPS 指令集的微处理器；

(3) 了解现代微处理器的流水线技术和超标量技术原理；

(4) 理解微处理器异常处理机制；

(5) 掌握 MicroBlaze 微处理器的应用。

3.1　微处理器基本构成

微处理器是构成计算机系统的核心部件，完成计算机的运算和控制功能。随着计算机技术的不断发展，微处理器的设计也越来越复杂。

微处理器执行一组机器指令，这组指令可向处理器告知应执行哪些操作。通常，微处理器执行三种基本工作：

(1) 通过使用 ALU(算术/逻辑单元)，微处理器执行数学计算，例如加法、减法、乘法和除法。现代微处理器包含完整的浮点处理器，它可以对很大的浮点数执行非常复杂的浮点运算。

(2) 微处理器可以将数据从一个内存位置移动到另一个位置。

(3) 微处理器可以做出决定，并根据这些决定跳转到一组新指令。

微处理器能够执行许多非常复杂的工作，但是所有工作都属于这三种基本操作的范畴。图 3-1 显示了一个能够执行上述三种基本操作的简单微处理器构成。

(1) 算术逻辑运算单元 ALU，实现各种算术逻辑运算。

(2) 寄存器组包括通用寄存器、指令寄存器、程序指针计数器等。通用寄存器用来暂存参与 ALU 单元运算的数据和中间结果，指令寄存器用来暂存从内存读入的指令，程序计数器用来指示下一条要执行的指令在内存中的存放地址。

(3) 指令译码器实现指令的译码，并根据译码结果控制 CPU 完成相关操作。

此微处理器与外部组件的基本接口包括：

(1) 一组地址总线(总线宽度可以 8 位、16 位或 32 位)，用于向内存发送地址。

(2) 一组数据总线(总线宽度可以是 8 位、16 位或 32 位),能够将数据发送到内存或从内存取得数据。

(3) 一条 RD(读)和 WR(写)控制信号,告诉内存它是希望将数据写入某个地址位置还是从某个地址位置获得内容。

(4) 时钟信号,将时钟脉冲序列发送到处理器,控制微处理器进行工作。

(5) 复位信号,用于将程序计数器重置为零(或者其他内容)并重新开始执行。

这是一个微处理器通常应该具有的外部接口。

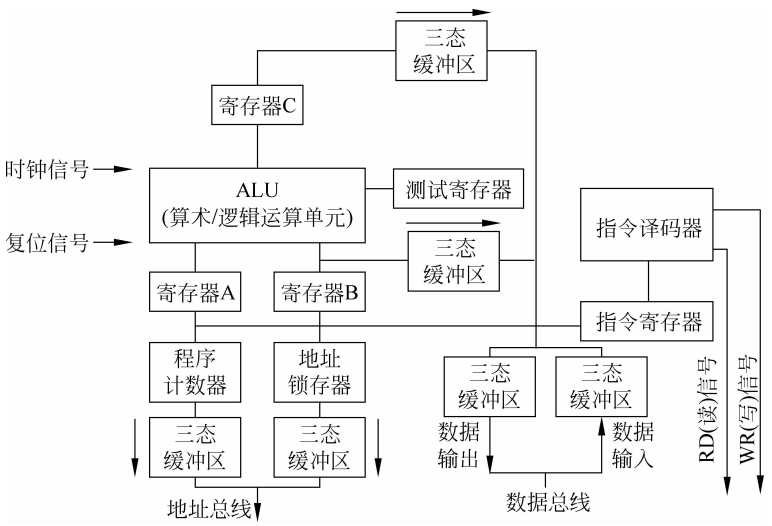

图 3-1 简单微处理器的基本构成

3.2 简单 MIPS 指令集微处理器基本构成

首先来看一个具有简单 MIPS 指令集的微处理器如何构成,同时也了解指令集与微处理器设计之间的关系。假设这个 MIPS 微处理器支持以下指令:

(1) 基本的内存操作如 lw,sw 指令;

(2) 基本的算术逻辑运算如 add,sub,and,or,slt 指令;

(3) 基本的程序控制如 beq,j 指令。

在这个指令集中没有包含乘除法指令,也不包含浮点运算类指令,但包含了作为微处理器必须含有的三类基本操作。本章将通过以上指令集来说明 MIPS 微处理器的基本构成。

在第 2 章 MIPS 汇编指令中,已知大多数指令都会用到算术逻辑运算单元,而且 MIPS 指令中大多数指令都会用到两个寄存器来进行运算,并且把计算结果保存在寄存器中。因此该 MIPS 简单指令集微处理器的基本结构可以用图 3-2 来表示。

在这个简化结构中包含了以下三类模块:

(1) 程序控制模块:顺序执行程序时自动将 PC 值加 4 获取下一条指令的地址,若存在相对跳转则将 PC 的值加上指令中的立即数偏移量形成新的指令地址。

(2) 运算模块:ALU 算术逻辑运算单元从寄存器组中的某两个寄存器或一个寄存器

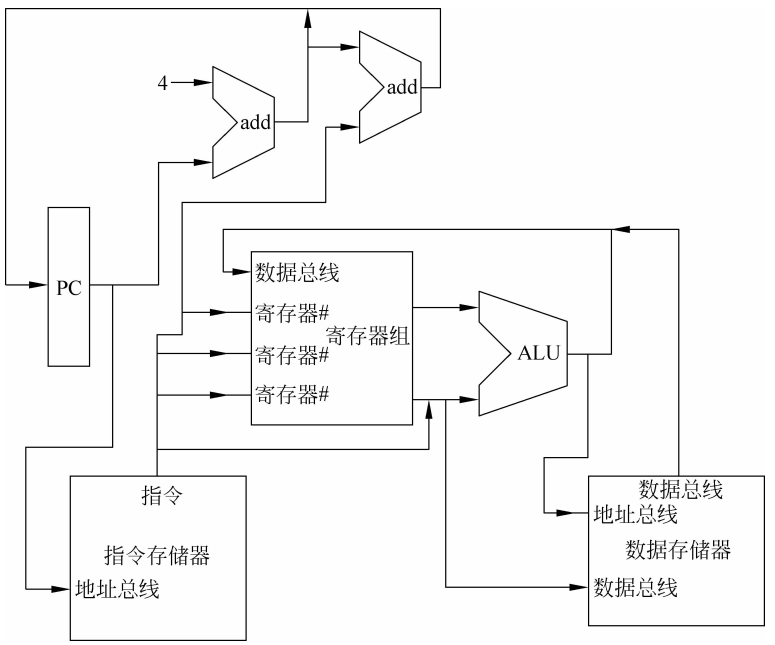

图 3-2　简单指令集 MIPS 微处理器数据通路简化结构图

和指令中的立即数中获取参与运算的数据，并将计算结果保存到寄存器组中。

(3) 数据传输模块：利用 ALU 单元计算数据在内存中的存储地址，实现寄存器与数据存储单元之间的数据拷贝。

这个简化结构虽然包含了 MIPS 处理器的基本功能，但是比较抽象。那么具体如何通过数字逻辑电路来实现这个基本结构呢？若考虑到数字逻辑电路的实现，则以上基本结构将变为图 3-3 所示的逻辑电路图。

在图 3-3 中有 3 个复用器，其中 MUX1 实现顺序程序和分支程序控制时 PC 值的选择；MUX2 实现 ALU 运算结果保存到寄存器还是内存数据存储到寄存器的选择；MUX3 实现是指令中的立即数还是寄存器作为第二个源操作数的选择。它们都由控制器具体控制选择哪路输入信号。同时控制器还提供寄存器读控制信号，存储器读、写控制信号以及 ALU 运算控制信号的产生。

具体如何产生这些信号则需要根据当前指令的译码结果，下面分别解释这个微处理器如何具体实现三类不同的指令：

(1) 算术运算类指令：由控制器产生 ALU 操作控制有效信号，同时使得 MUX2 选择 ALU 的运算结果保存到寄存器中；若操作数中含有立即数，则使得 MUX3 选择指令中的立即数作为输入，否则选择寄存器作为输入。

(2) 存储器读写类指令：若执行的是 lw 指令，则首先由控制器产生 ALU 操作控制信号控制 ALU 计算存储单元地址，然后再产生存储器读有效控制信号，同时使得 MUX2 选择存储器的数据存储到寄存器中；若为 sw 指令，则同样首先产生 ALU 操作控制信号控制 ALU 计算存储单元地址，然后再产生寄存器读有效控制信号和存储器写控制信号同时有效，使得寄存器的数据存入到数据存储器内。

(3) 程序控制类指令：若为 beq 指令首先产生 ALU 操作控制信号，利用 ALU 比较两个操作数的大小，然后根据比较结果以及产生的分支控制有效信号经过与门控制 MUX1 选择 PC+4 的值还是 PC+4 以及指令中的立即数的和作为下一条指令的地址。

在这个实现中需要用到数字逻辑电路中的组合逻辑电路以及时序逻辑电路。要实现各个部件协调工作，必须为整个处理器提供统一的时钟信号。通常在微处理器实现中采用边沿触发时钟信号来实现各个部件的协调控制。

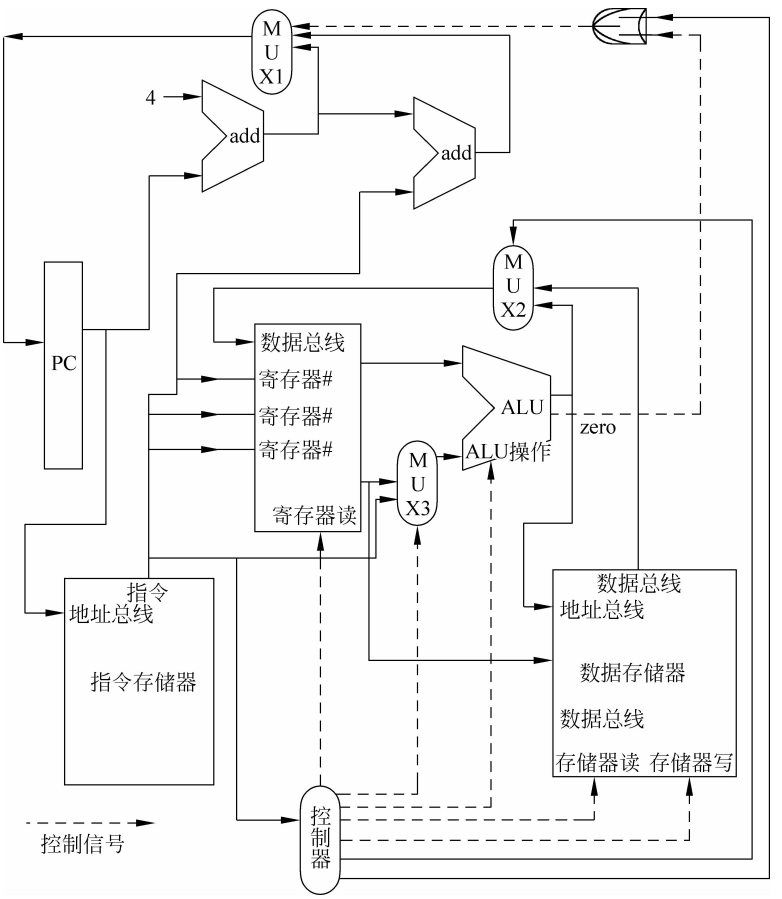

图 3-3 简单指令集 MIPS 处理器的逻辑实现

3.3 数据通路实现原理

下面根据简化的 MIPS 微处理器基本构成来分析各个部件的具体实现原理。在这一节里首先根据不同的指令类型来分别分析数据通路中各个部件的具体实现原理，然后再把各个部件合并形成总的数据通路。

3.3.1 指令获取部件

指令地址在微处理器中采用专门的寄存器程序计数器 PC 来保存，当程序顺序执行时，

PC 的值自动加 4，因此该部件的基本构成如图 3-4 所示。

3.3.2　R 型指令实现部件

R 型指令都是首先读取两个寄存器操作数，然后执行算术逻辑运算，最后再将结果保存到寄存器中。在微处理器中所有的寄存器构成了寄存器组，MIPS 微处理器提供了 32 个不同的通用寄存器，因此要实现对这 32 个不同的通用寄存器实现寻址，需要提供 5 位寄存器选择地址线($32=2^5$)。在一条 R 操作指令中，需同时提供 3 个寄存器地址信号：读寄存器 1 的地址、读寄存器 2 的地址，写寄存器的地址以及 3 组数据总线：寄存器 1 的数据输出、寄存器 2 的数据输出、寄存器 3 的数据输入。并且需要将读寄存器 1 的数据和读寄存器 2 的数据进行运算之后送到写寄存器的数据线上，因此 R 型指令的实现如图 3-5 所示。

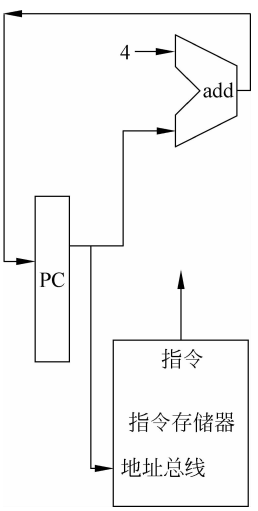

图 3-4　微处理器指令获取部件

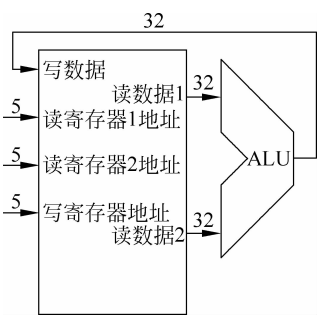

图 3-5　R 格式指令实现部件

3.3.3　存储器数据存取部件

MIPS 处理器仅支持一种存储器寻址即基址寻址。在基址寻址中，数据存储器的地址由寄存器与指令中的立即数偏移量之和构成，因此需要 ALU 运算单元来计算存储单元地址，由于 ALU 单元仅支持 32 位数据的运算，而指令中的立即数偏移量仅为 16 位，因此需要通过符号扩展部件将其扩展为 32 位立即数。同时数据存储器中的数据支持数据存储器到寄存器组之间的双向传输，因此需要提供存储器读写控制信号来控制数据的传输方向。其基本构成如图 3-6 所示。

当执行 lw 指令时，首先从基址寄存器中获取基地址，并通过 ALU 运算单元以及符号扩展单元获得存储单元地址，再由存储器读控制信号使得存储器输出数据到数据总线，并将数据保存到数据寄存器中；当执行 sw 指令时，同样首先从基址寄存器中获取基地址，并通过 ALU 运算单元以及符号扩展单元获得存储单元地址，再由存储器写控制信号以及寄存器读控制信号，控制数据寄存器中的数据输出到数据总线并保存到存储器中。

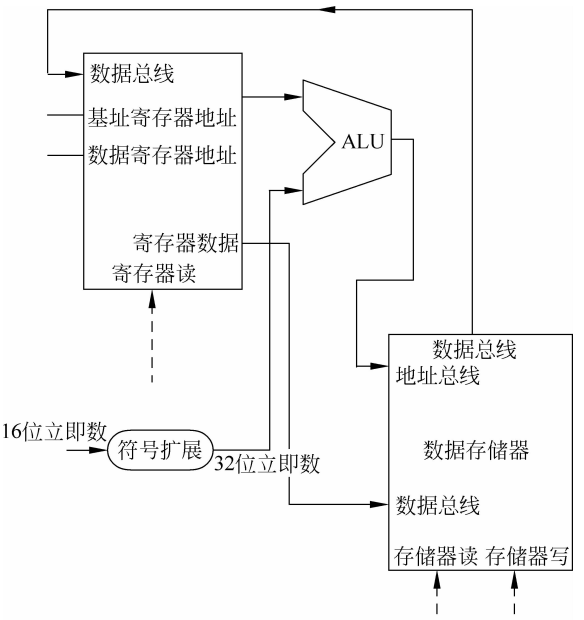

图 3-6 存储器数据传输部件

3.3.4 条件跳转控制

条件跳转控制指令 beq 需要首先比较两个寄存器的值，然后根据比较结果决定是否跳转。因此针对该指令需要利用 ALU 单元实现两个寄存器的值比较，而且还需要地址加法器实现跳转地址计算。在 MIPS beq 指令中，跳转地址由 PC 的值与指令中的立即数的和构成，由于该立即数在指令中仅 16 位，因此需要通过符号扩展部件扩展为 32 位，由于每条指令 4 个字节，因此需要进一步左移 2 位。因此需要用到符号扩展以及移位单元。其构成如图 3-7 所示。当执行 beq 指令且条件成立时，与门输出 1 从而控制复用器采用右边的加法器产生的跳转地址输出到 PC 寄存器。

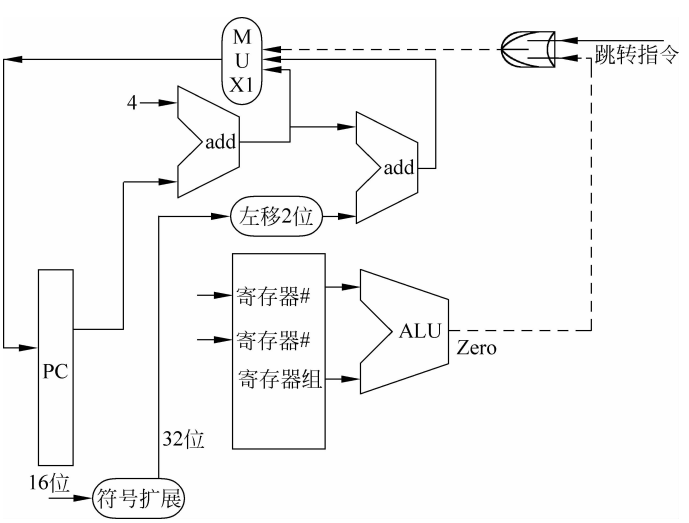

图 3-7 条件跳转控制部件

3.3.5　无条件伪直接寻址部件

无条件伪直接寻址数据属于 J 指令，在该指令中除了 6 位操作码之外，其余位为跳转伪直接地址的立即数，指令的实际地址由 PC 的高 4 位，指令中的 26 位立即数以及在低 2 位补充 0 构成。其构成如图 3-8 所示。

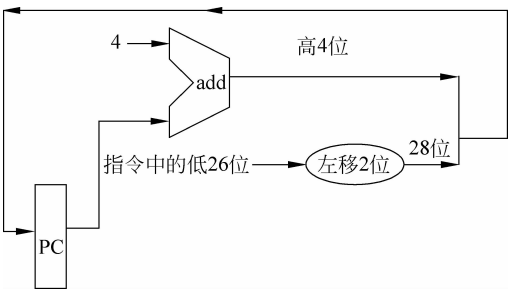

图 3-8　无条件伪直接跳转部件

3.3.6　完整的数据通路构成

要支持以上所有指令，则需要合并这些部件。由于各个功能部件中的部分功能模块可以重复利用，因此合并之后的完整数据通路如图 3-9 所示。

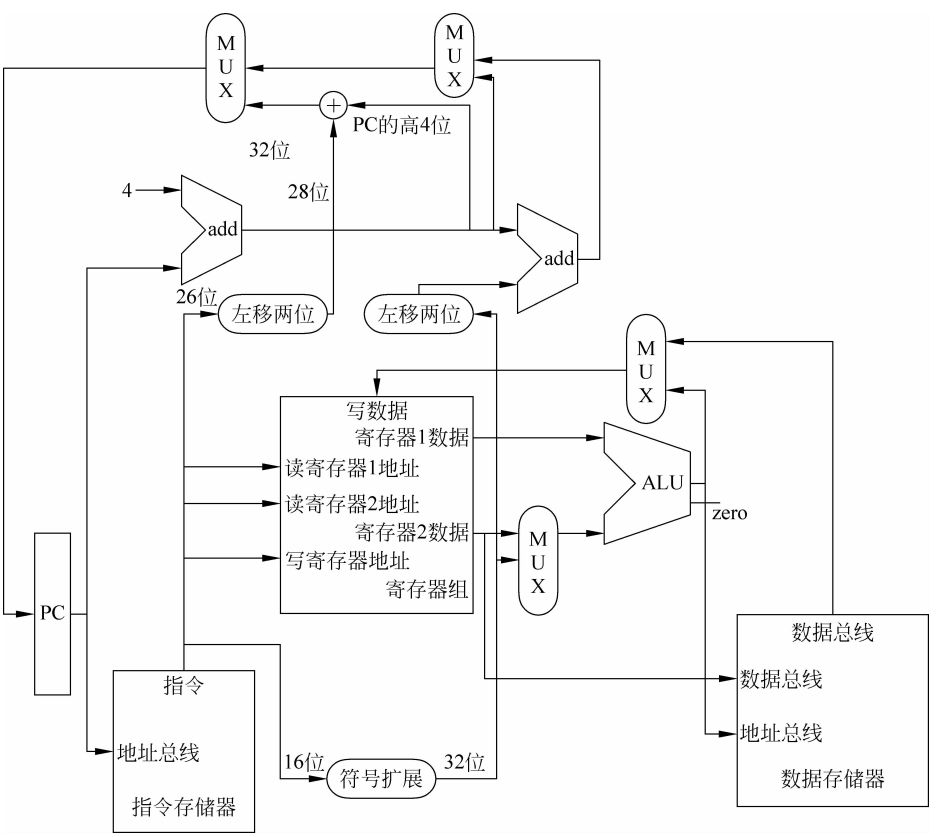

图 3-9　简单 MIPS 微处理器数据通路构成

3.4 控制器实现原理

3.4.1 ALU 控制

已知在这个简单的 MIPS 指令集中，微处理器需支持 add、sub、and、or、slt 运算指令，这些指令需要利用 ALU 单元实现运算，同时数据存储指令 sw、lw 也需要通过 ALU 单元计算存储器地址，条件跳转指令 beq 需要 ALU 来比较两个寄存器是否相等。所有这些指令包含的操作为加、减、与、或、小于设置 5 种不同的操作。

实际的 MIPS 处理器中，ALU 单元利用 4 根输入控制线的译码决定执行何种操作，对应以上 5 种操作的编码如表 3-1 所示。

表 3-1 ALU 操作控制信号编码

输入信号	操作类型	输入信号	操作类型
0000	与	0110	减
0001	或	0111	小于设置
0010	加		

MIPS 指令中具有 6 位操作码，如果为 R 型指令，进一步采用 6 位功能码来表示 R 型指令的具体操作。由于设计的微处理器支持的运算指令全部为 R 型指令，因此可以通过对 R 型指令的 6 位功能码编码产生 ALU 的 4 位控制信号。但是 sw，lw 以及 beq 指令没有功能码，因此需首先区分指令的类型。在这个例子中指令分为 3 类，因此采用 2 位进行编码，这两位叫做 ALU 操作类型码。通过 2 位操作类型码以及 6 位指令功能码就可以产生 ALU 单元的 4 位控制信号。它们之间的对应关系如表 3-2 所示。

表 3-2 ALU 输入控制信号与指令的操作码以及功能码之间的关系

指　令	2 位操作码	指令功能	6 位功能码	ALU 的运算	ALU 的控制信号
LW	00	取字	XXXXXX	加	0010
SW	00	存字	XXXXXX	加	0010
BEQ	01	相等跳转	XXXXXX	减	0110
R 型指令	10	加	100000	加	0010
R 型指令	10	减	100010	减	0110
R 型指令	10	与	100100	与	0000
R 型指令	10	或	100101	或	0001
R 型指令	10	小于设置	101010	小于设置	0111

根据表 3-2 我们可以发现 6 位功能码的前 2 位完全相同，因此在译码时可以不考虑，这样就得到 4 位 ALU 单元控制信号的真值表如表 3-3 所示。

已知了真值表，就可以根据真值表利用逻辑门来完成 ALU 控制信号的产生电路。具体实现请参考数字逻辑电路设计相关知识。

表 3-3　4 位 ALU 单元控制信号的真值表

操作码		功能码						ALU 控制信号
位 1	位 0	位 5	位 4	位 3	位 2	位 1	位 0	
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	0	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	0	X	X	1	0	1	0	0111

3.4.2　主控制器

在这个简单 MIPS 指令集中，微处理器支持的三种不同操作可以分为三种类型的指令：

(1) 所有的运算类指令：add，sub，and，or，slt 都为 R 型指令。

(2) 数据存取类指令 lw，sw 以及条件跳转指令 beq 属于 I 型指令。

(3) 无条件跳转指令 J 属于 J 型指令。

它们的格式如图 3-10 所示。

R 型指令

op	rs	rt	rd	shamt	funct
6 位	5 位	5 位	5 位	5 位	6 位

I 型指令

op	rs	rt	constant address
6 位	5 位	5 位	16 位

J 型指令

op	constant address
6 位	26 位

图 3-10　不同指令格式

主控制器需要根据指令的操作码 op 字段产生不同的控制信号来控制数据通路中的各个不同部件协调工作。下面分析这些不同指令的特点：

(1) 操作码都是 6 位，在指令的最高 6 位上，用指令[31:26]或 op[5:0]来表示这些数位。

(2) R 型指令、beq 指令以及 sw 指令第一个源操作数和第二个源操作数都分别为 rs 字段和 rt 字段，即读寄存器的地址分别在指令的[25:21]和[20:16]位上，即用指令的[25:21]位和以及指令的[20:16]位表示两个读寄存器地址。

(3) 存储器寻址中，基址寄存器都在指令的 rs 字段，在指令的位 25:21 上，用指令的[25:21]位表示。

(4) 存储器偏移地址都保存在指令的最低 16 位上,用指令的[15:0]位来表示。

(5) R 型指令,写寄存器地址保存在 rd 字段,但是对于 lw 指令,写寄存器地址保存在 rt 字段,因此写寄存器地址根据指令类型不同将来自指令的不同字段。

因此需要在原来的数据通路上增加一个写寄存器地址复用器,而且所有的复用器都需要相应的控制信号来选择其输入信号。由于数据通路上共有 5 个复用器,因此需要 5 个复用器控制信号,再加上存储器的读写分别需要控制信号,以及由存储器向寄存器传输数据时,需要提供寄存器写控制信号,因此共需要 8 个控制信号。加上控制信号的数据通路如图 3-11 所示。

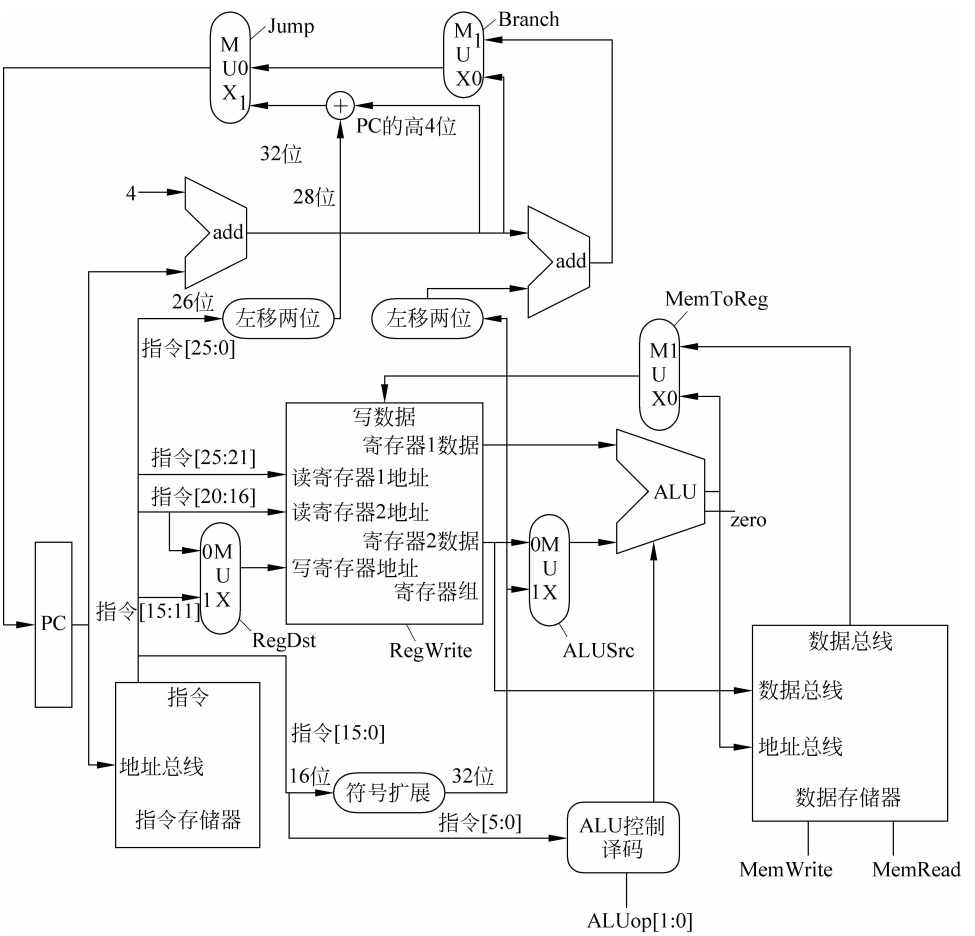

图 3-11 加上写寄存器地址复用器以及 ALU 控制译码部件的数据通路

这些控制信号的作用在表 3-4 中进行了解释。

这 8 个控制信号再加上 ALUop 的 2 个控制信号,共 10 个控制信号除了 Branch 之外都可以直接通过对指令中的[31:26]位进行译码来获得,Branch 控制信号需要经过对指令中的[31:26]位译码以及 ALU 的比较结果 zero 相与来获得,这里暂且把 Branch 信号当作对指令中的[31:26]位译码结果获得。下面首先了解一下微处理器在执行不同指令时,各个控制信号相对应的状态,如表 3-5 所示。

表 3-4　控制信号的种类

控制信号名称	置 1	清 0
RegDst	表示写寄存器的地址来自指令[15:11]	来自指令[20:16]
Jump	表示 PC 的值来自伪直接寻址	来自另一个复用器
Branch	表示下一级复用器的输入来 PC 相对寻址加法器	来自 PC+4
MemToReg	表示写寄存器数据来自存储器数据总线	来自 ALU 结果
ALUSrc	表示 ALU 的第二个数据源来自指令[15:0]	来自读寄存器 2
RegWrite	将写寄存器数据存入写寄存器地址中	无操作
MemWrite	将写数据总线上的数据写入内存地址单元	无操作
MemRead	将内存单元的内容输出到读数据总线上	无操作

表 3-5　控制信号与指令之间的关系

指令	RegDst	Jump	Branch	Mem ToReg	ALUSrc	Reg Write	Mem Write	Mem Read	ALUop1	ALUop0
R 型	1	0	0	0	0	1	0	0	1	0
lw	0	0	0	1	1	1	0	1	0	0
sw	X	0	0	X	1	0	1	0	0	0
beq	X	0	1	X	0	0	0	0	0	1
j	X	1	0	X	X	0	0	0	X	X

如果在数据通路中加入一个控制器实现对操作码的译码，并产生这些控制信号，就形成了一个简单的微处理器，如图 3-12 所示。该处理器各个部件在统一的时钟控制下工作，并在一个时钟周期内完成一条指令。

根据不同指令的 6 位操作码 op[5:0]从而得到指令操作码与控制信号之间逻辑关系真值表，如表 3-6 所示。

表 3-6　指令操作码与控制信号逻辑关系真值表

输入/输出	信号名称	R 型	lw	sw	beq	j
输入	op5	0	1	1	0	0
	op4	0	0	0	0	0
	op3	0	0	1	0	0
	op2	0	0	0	1	0
	op1	0	1	1	0	1
	op0	0	1	1	0	0
输出	ALUop1	1	0	0	0	X
	ALUop0	0	0	0	1	X
	RegDst	1	0	X	X	X
	ALUSrc	0	1	1	0	X
	RegWrite	1	1	0	0	0
	MemWrite	0	0	1	0	0
	MemRead	0	1	0	0	0
	Branch	0	0	0	1	0
	Jump	0	0	0	0	1
	MemToReg	0	1	X	X	X

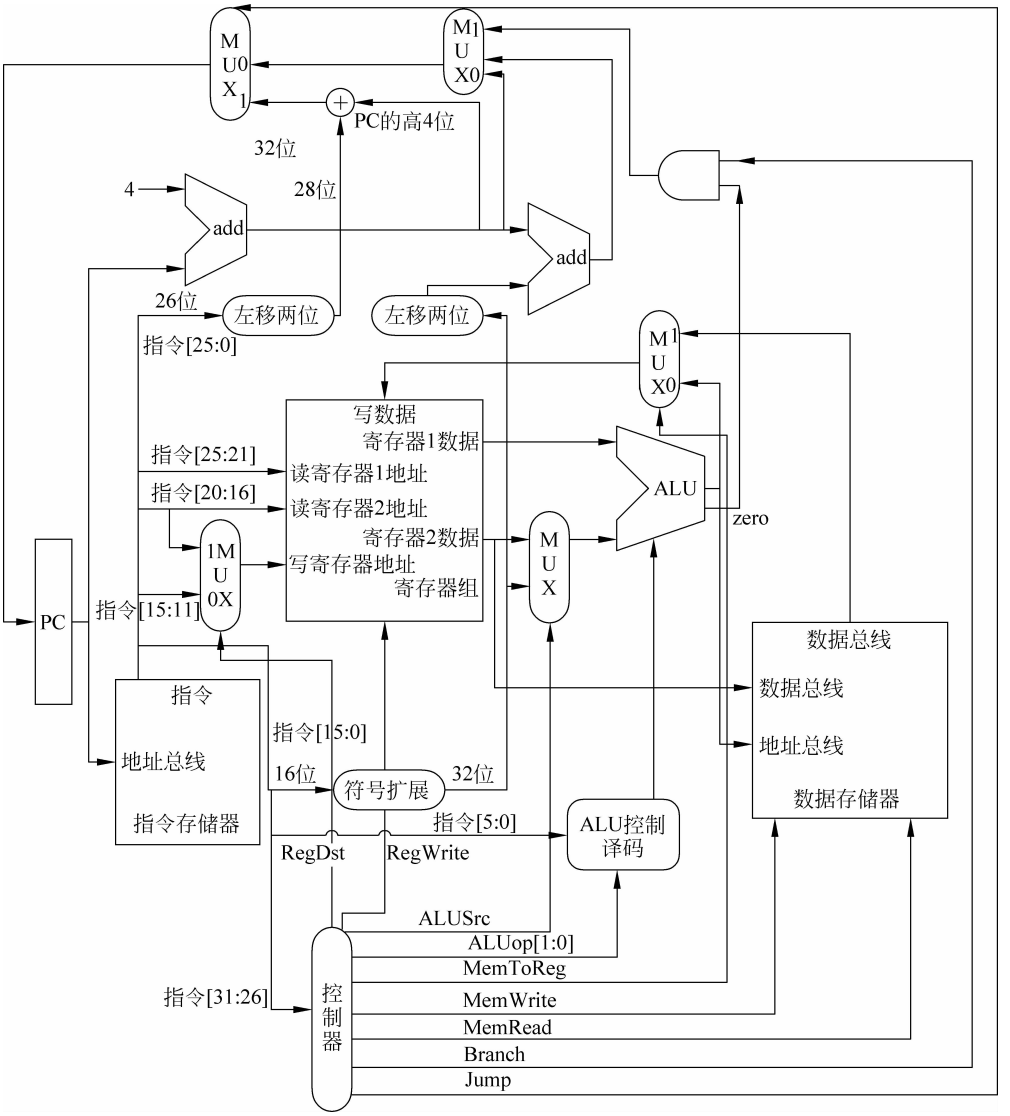

图 3-12 加上控制器的简单指令集 MIPS 微处理器完整框图

根据表 3-6，利用数字逻辑电路中的逻辑门电路就可以设计出控制器的逻辑电路。

3.4.3 不同指令的执行过程描述

下面通过具体的指令实例结合控制信号状态解释几个典型指令在微处理器中的执行过程。

R 型指令 add $t1，$t2，$t3 执行过程如下：

（1）根据 PC 的初始值，从指令存储器中获得指令的机器代码，并将 PC 的值自动加 4。

（2）根据指令的机器代码，使寄存器 $t2 和 $t3 的值从寄存器组中输出，同时控制器对指令的操作码译码产生相应的控制信号，ALUSrc 为 0 使 $t3 的数据进入到 ALU 单元参与运算。

（3）由于 ALUop1 为 1，ALUop0 为 0，因此 ALU 译码单元进一步对指令的功能进行译码，从而控制 ALU 单元执行加法运算，由于此时 MemToReg 为 0，且 RegWrite 为 1，因

此结果保存到寄存器 $t1。

lw 指令 lw $t1,8($t2)的执行过程如下：

(1) 根据 PC 的初始值，从指令存储器中获得指令的机器代码，并将 PC 的值自动加 4。

(2) 由机器的指令代码使得寄存器 $t2，$t1 的值从寄存器组中输出，同时控制器对指令的操作码译码产生相应的控制信号，此时 ALUSrc 为 1，使得指令中的[15:0]位经符号扩展之后进入到 ALU 单元参与运算，$t1 的数据输出在此指令中没有意义。

(3) 由于 ALUop1 为 0，ALUop0 为 0，因此 ALU 单元执行加法运算，MemToReg 为 1，ALU 单元的计算结果将指示数据内存单元地址。同时 RegWrite 为 1，MemRead 为 1，且 RegDst 为 0，因此特定内存单元内的数据经数据总线存储到寄存器 $t1 中。

sw 指令 sw $t1,8($t2)的执行过程如下：

(1) 根据 PC 的初始值，从指令存储器中获得指令的机器代码，并将 PC 的值自动加 4。

(2) 由机器的指令代码使得寄存器 $t2，$t1 的值从寄存器组中输出，同时控制器对指令的操作码译码产生相应的控制信号，此时 ALUSrc 为 1，使得指令中的[15:0]位经符号扩展之后进入到 ALU 单元参与运算。

(3) 由于 ALUop1 为 0，ALUop0 为 0，因此 ALU 单元执行加法运算，ALU 单元的计算结果将指示数据内存单元地址。同时 MemWrite 为 1，因此从寄存器 $t1 输出的数据将保存到由 ALU 单元计算结果所指示的内存单元中。

beq 指令 beq $t1，$t2，L1 的执行过程如下：

(1) 根据 PC 的初始值，从指令存储器中获得指令的机器代码，并将 PC 的值自动加 4，同时也将指令的低 16 位立即数符号扩展、左移两位之后的和相加。

(2) 由机器的指令代码使得寄存器 $t2，$t1 的值从寄存器组中输出，同时控制器对指令的操作码译码产生相应的控制信号，此时 ALUSrc 为 0，使得 $t1 的值参与 ALU 运算。

(3) 由于 ALUop1 为 0，ALUop0 为 1，因此 ALU 单元执行减法运算，并判断结果是否为 0，若为 0 则使得 zero 输出 1，否则输出 0。

(4) 此时 Branch 为 1，如果 zero 为 1 则经过与门之后控制复用器选择指令中的立即数与 PC＋4 的和赋给 PC，从而实现跳转，若 zero 为 0，则选择 PC＋4 的值赋给 PC，从而顺序执行程序。

j 指令 j L1 的执行过程如下：

(1) 根据 PC 的初始值，从指令存储器中获得指令的机器代码，并将 PC 的值自动加 4，同时也将指令的低 26 位左移两位之后再与 PC＋4 的高 4 位合并形成新的地址。

(2) 此时由机器的指令操作码控制器译码使得 Jump 为 1，因此将新的地址赋给 PC，直接跳转到 L1 处。

这里解释了各类指令的执行流程，必须注意：不管执行什么指令，数据通路上的不受控制的部件一直处于工作状态，如各个地址加法器，只不过在不同的状态下，有些结果是没有意义的。

至此，了解了简单指令集 MIPS 微处理器的设计原理。那么现代实际的微处理器是不是与这里设计的微处理器结构一致呢？实际上，现代微处理器比目前讲解的简单 MIPS 微处理器要复杂得多。在设计的微处理器中，指令是一条一条地执行，任何指令的操作都是在一个时钟周期内完成。事实上不同的指令执行所需要的时间不同，在这个设计中只能采用执行时间最长的指令周期作为微处理器的时钟周期，这样就降低了微处理器的执行效率。

3.5 微处理器设计新技术

3.5.1 流水线技术

流水线的基本原理是把一个重复的过程分解为若干个子过程，前一个子过程为下一个子过程创造执行条件，每一个过程可以与其他子过程同时进行。简而言之，就是“功能分解，空间上顺序依次进行，时间上重叠并行”。流水线处理器把一条指令的操作分为若干级，每一级在一个时钟周期完成，处理器在每个周期发出一条指令。这样，多条指令就可以由处理器并行执行，每条指令处在不同的级。下面来了解一下实现原理。

从 MIPS 微处理器指令的执行过程可知，指令的执行通常可以分为以下 5 个部分：

(1) 从指令存储器中取指令；

(2) 从寄存器中读取数据的同时，对指令进行译码；

(3) 执行指令对应的操作或计算数据存储器地址；

(4) 从数据存储器中存取数据；

(5) 将结果回写入寄存器中。

通常微处理器指令的执行都可以由这 5 个部分来构成。分别把这 5 个部分命名为：取指令(IF)，指令译码(DEC)，计算(EXEC)，存储器操作(MEM)，回写结果(WB)。如果将微处理器时钟周期缩短，每个部件利用一个时钟周期完成其功能，那么微处理器执行指令的过程就可以描述为图 3-13 所示。

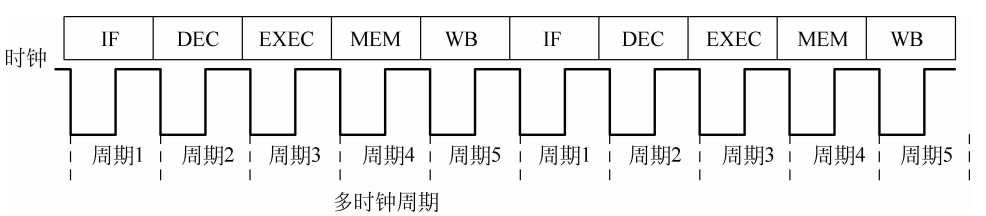

图 3-13 多时钟周期微处理器执行指令过程

如果把这 5 个部分在处理器中设计为相对独立的部件，那么它们就可以同时执行其对应的工作。即在同一条指令中顺序执行，在不同的指令中并行执行，这就是流水线的基本原理，表述如图 3-14 所示。流水线具有多少个不同的执行部件，则称这条流水线具有多少级。

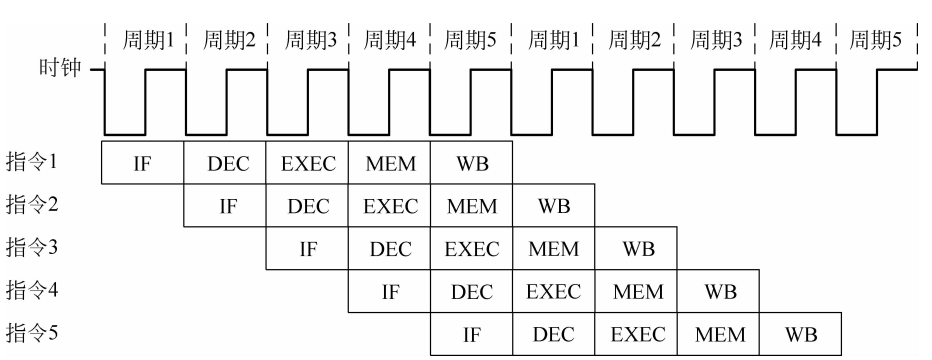

图 3-14 流水线指令执行过程

从图 3-13 和图 3-14 的对比中可以看出，不采用流水线的处理器在 10 个时钟周期内仅完成了两条指令，而采用流水线则可以在 10 个时钟周期内完成 6 条指令。这种指令执行的重叠性（使一条流水线畅通）和同时性（多条流水线同时工作）就称为指令级并行。

如果流水线的每个部件执行时间相同，那么在流水线结构中，每条指令执行的时间间隔可以表示为：

$$\text{指令执行的时间间隔} = \frac{\text{每条指令占有的时钟周期个数}}{\text{流水线级数}}$$

将简单 MIPS 微处理器的数据通路按照这 5 级流水线重新组织，其结构将变为图 3-15 所示。由前面阐述的指令执行过程可知，指令的机器代码在流水线的 IF 部件取出来，在 DEC 部件译码，在 EXEC 部件进行计算，因此当 IF 部件取下一条指令时，必须保存上一次取得的指令以供后面的部件使用，也就是说在每两个流水线部件之间必须增加器件来实现流水线不同部件之间的接口。虽然在流水线处理器中增加少量的器件就较大幅度地提高了微处理器的执行效率。但是它要处理：

（1）结构相关：由于硬件资源不充足而导致流水线不畅通。

（2）数据相关：由于指令的源操作数依赖其他指令的计算结果而导致读数据不正确。

（3）转移相关：由于是流水线操作而导致在转移发生之前，若干条转移指令的后续指令已被取到流水线处理器中。

这些相关问题，并不能从根本上解决微处理器的执行效率问题。

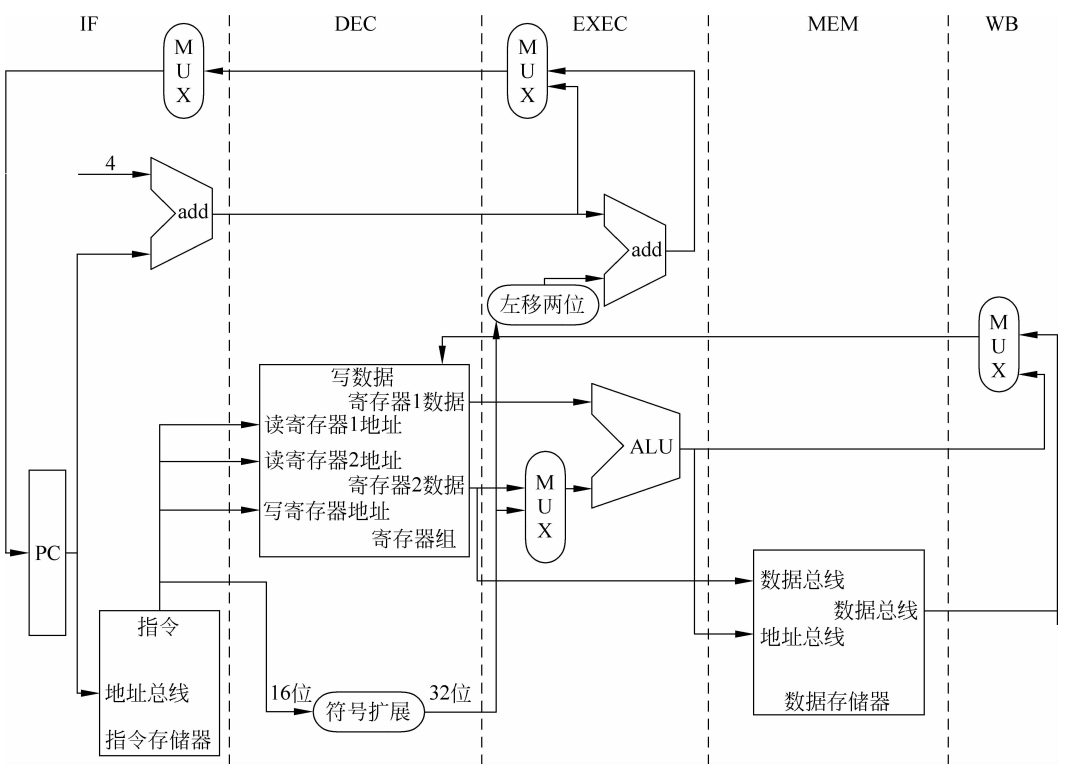

图 3-15　数据通路的 5 级流水线划分

3.5.2 超标量技术

为微处理器集成多个 ALU、多个译码器和多条流水线，以并行处理的方式来提高性能，这就是超标量技术。通过采用超标量技术可以提高指令级并行度。

流水线处理器在每个时钟周期最多发出一条指令，而超标量处理器可以在每个时钟周期发出 1～8 条指令。同时发出的指令必须是不相关的并且不能出现资源冲突。假设处理器有一个整数部件和一个浮点部件，处理器至多能发出两条指令，一条是整数类型的指令，包括整数算术逻辑运算，存储器访问操作和转移指令；另一条必须是浮点类型的指令。如果不是这样，处理器只好发出一条指令。如果处理器的执行部件足够，则资源冲突的可能性会减小。在这样的处理器中，指令的调度策略是一个需要考虑的问题。为实现指令的调度，通常在处理器中会设计一个指令缓冲区，用来存放等待调度的指令。

3.6 微处理器异常处理原理

计算机除了正常顺序执行程序中的指令，分支转移以及跳转之外，往往还有一些特殊的情况需要处理器：如计算结果产生了溢出，碰到了非正确的机器指令以及外部设备需要进行数据的输入/输出等。这些情况微处理器往往不能预测其发生的时间，不受程序控制。微处理器必须提供一种机制来处理这类异常事件。在计算机系统中，通常把这些异常事件叫异常(exception)或者中断(interrupt)。在 MIPS 处理器中中断特指处理器外部事件，而 Intel 处理器将内部异常事件和外部异常事件统称为中断。表 3-7 列举了计算机系统中常见的异常事件。

表 3-7 计算机系统中常见的异常事件

异 常 种 类	来 源	MIPS 处理器命名
I/O 设备	外部	中断
用户程序唤醒操作系统	内部	异常
计算结果溢出	内部	异常
未定义的指令(非法指令)	内部	异常
硬件出错	两者	异常或中断

异常处理是指计算机在正常运行的过程中，由于种种原因，使 CPU 暂时停止当前程序的执行，而转去处理临时发生的异常事件，处理完毕后，再返回去继续执行暂停的程序。也就是说，在程序执行过程中，插入另外一段程序处理异常事件。微处理器要能够实现异常处理需要完成以下几方面的功能：

(1) 记录异常发生的原因；

(2) 记录程序断点处的指令在存储器中的地址，因为处理完异常之后需要继续执行暂停的程序；

(3) 记录不同种类的异常处理程序在内存中的地址；

(4) 建立异常种类与异常处理程序地址之间的对应关系。只有建立了这种对应关系，微处理器才能在遇到某个特定的异常事件时去执行相对应的异常处理程序。

3.6.1 异常事件识别

由于在计算机系统中异常的种类较多，不同种类的异常处理方式不同。因此微处理器

为处理异常事件，首先需要知道异常事件的类型。在计算机系统中，有两种方式来识别异常事件。第一种方式是状态位法：在微处理器中利用一个寄存器对每种异常事件确定一个标志位，当有异常事件发生时，寄存器中对应的位置 1，一个 32 位的寄存器可以表示 32 种不同类型的异常事件。第二种方法是向量法：对不同类型的异常事件进行编码，这个编码叫中断类型码或异常类型码。若产生了异常事件，就产生相对应的异常类型码，微处理器通过解码就可以获知异常发生的原因。如果采用一个 8 位的寄存器记录异常类型码，就可以实现 256 种不同类型的异常事件识别。

MIPS 处理器采用第一种方式状态位法：一个 32 位的寄存器来记录异常产生的原因，该寄存器叫 Cause 寄存器。实际上，MIPS 计算机系统并没有这么多种不同类型的异常，因此有很多位没有用到。

Intel 处理器采用第二种方式向量法来识别不同的中断源。中断类型码的产生采用编码器来实现，将外部异常事件和内部异常事件的中断类型编码器分开，内部异常事件的编码器集成在处理器中，而外部事件的中断类型码采用专门的芯片来实现，当有外部中断发生时，微处理器在接收到外部中断信号发生的同时，通过读取外部器件提供的中断类型码来识别不同的中断源。

3.6.2　断点保存和返回

在异常发生后，CPU 暂时停止当前程序的执行，转去处理临时发生的异常事件，处理完毕后，再返回去继续执行暂停的程序。要能够继续执行暂停的程序，那么就需要保留被中断程序的指令在内存中的存放地址，只有这样，才能处理完中断之后恢复到被中断的程序继续执行。

保留被中断处指令的地址有两种实现方式。第一种方式是在微处理器中设计一个寄存器 EPC，当微处理器出现异常时，就将 PC 的值保存到 EPC 中。异常处理完之后，再把 EPC 的值赋给 PC，这样就可以实现中断的返回。但是采用这种方式，如果计算机正在处理中断还没有返回，就不能处理再次发生的异常事件，就与仅采用 $ra 保存主程序的返回地址类似。因此为实现异常处理的嵌套，微处理器需要在进入异常处理程序之前首先将 EPC 的值压入栈中，然后再保存 PC 的值到 EPC 中。第二种方式是当出现异常时，微处理器直接将 PC 的值压入栈中，异常处理完之后，再从栈顶把值弹出来赋给 PC，这样也就不用担心异常处理的嵌套问题了。

3.6.3　异常处理程序进入方式

异常处理程序根据其处理异常原因的不同而不同，这些异常处理程序都必须在异常发生之前保存在内存中，以便异常发生时执行相应的处理。这些程序应该保存在内存的什么位置与微处理器如何根据异常事件类型寻找异常处理程序的机制有关。微处理器寻找异常处理程序有以下几种方式：

(1) 微处理器分配一块专门的内存区域保存异常处理程序。通常在这块内存区域中为每个异常处理程序分配固定长度的空间如 32 个字节或 8 条指令长度的空间，而且针对每个异常事件其异常处理程序的存放地址是固定的。如果 8 条指令不能完成异常处理，那么在这 32 个字节的内存空间中就只能实现一些简单的处理，然后再通过一条跳转指令跳转到真正的异常处理程序的存放地址处。这种方式适合于大多数异常处理非常简单的应用场景。

(2) 微处理器为所有的异常事件仅提供一个异常处理程序存放地址,发生任何异常事件都首先转移到该地址执行总的异常处理,并在总异常处理程序中分析异常事件的原因,然后再根据异常的原因通过子程序调用的方式去执行相应的异常处理。在这种实现方式中,微处理器需要逐个比较异常状态标志寄存器识别异常事件。硬件实现简单,但是降低了异常处理的效率。

(3) 微处理器分配一块专门的内存区域保存异常处理程序的入口地址,这个入口地址也叫做中断向量。保存异常处理程序的入口地址的内存区域叫做中断向量表。每个中断向量都是固定长度的,如 4 个字节等。针对每个异常事件其异常处理程序的入口地址存放位置是固定的。在这种方式中,异常处理程序可以存放在内存中的任意位置,只需要把该异常处理程序的入口地址保存到中断向量表中正确的地址中,当异常发生时,微处理器就可以通过中断向量表查找到中断服务程序的入口地址。这种方式是间接获取异常处理程序的地址。

方式(1)、(3)应用在采用向量法识别异常事件的处理器中,方式(2)应用在状态位法识别异常事件的处理器中。方式(1)、(2)微处理器都是直接根据内部电路转移到异常处理程序,而方式(3)微处理器需要首先读取内存中断向量表中异常处理程序的存放地址,然后才能转入异常处理程序。

图 3-16 展示了假设产生了 2 号异常事件,各种方式下微处理器如何转移到异常处理程序的过程。

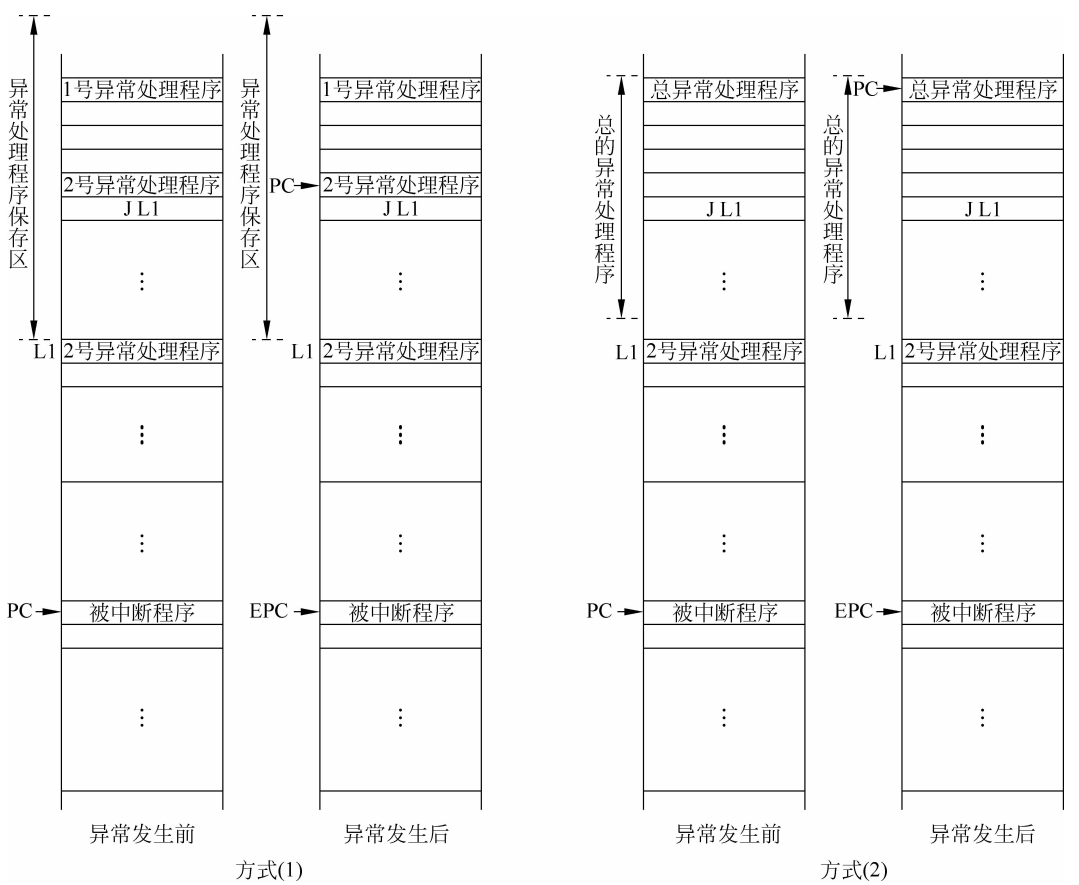

图 3-16 异常发生时 3 种不同方式进入异常处理的过程示意

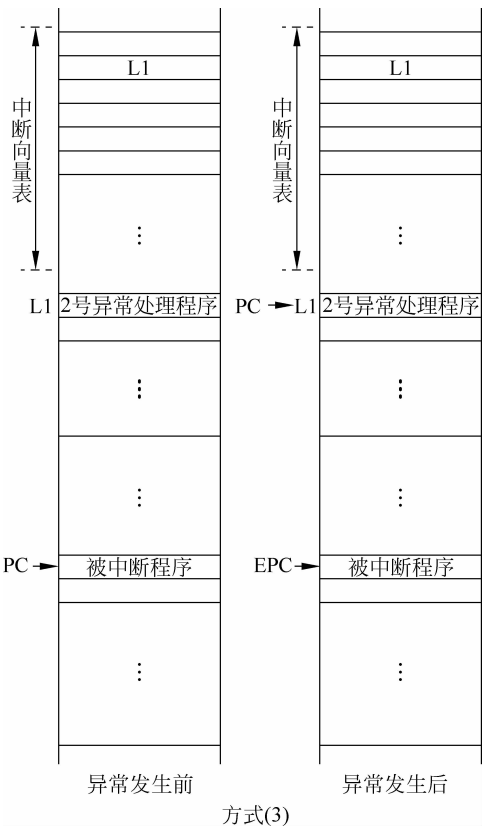

图 3-16　（续）

3.7　微处理器外部接口

在 3.4 节设计的简单 MIPS 微处理器中，我们看到了指令存储器和数据存储器，这样设计的目的只是为了简化微处理器设计。实际上，微处理器内部仅含有少量的存储单元：如 PC 微处理器内的片内 Cache，嵌入式微处理器内的部分指令存储器。也就是说计算机系统内大量的存储单元位于微处理器的外部。微处理器要实现与存储器数据的交互，必须通过外部总线才能访问存储器。另一方面，微处理器作为计算核心，其处理的数据除了来源于存储器之外，还有大量的数据来自于计算机外围各种数据采集设备，如鼠标、键盘、麦克风、摄像头等。要实现与这些外围设备之间的数据交互，微处理器也必须通过总线才能访问这些输入/输出设备。因此微处理器必须提供总线接口，才能访问其外部器件。

由图 1-6 中的典型计算机系统结构可知，微处理器为实现与计算机系统内的其他部件之间的信息交互必须提供地址、数据和控制总线。地址总线实现对不同设备或不同存储单元的寻址，数据总线承载交互的数据，控制总线控制数据的传送方向以及与外设的其他控制信号：中断，总线请求等。Intel X86CPU 的外部接口就是采用这样的接口方式，图 3-17 展示了 8088 微处理器的外部引脚接口。$A/D_0 \sim A/D_7$ 为 8 位地址数据总线复用接口，$A_8 \sim A_{19}$ 为高 12 位地址总线，其余引脚除了电源、时钟引脚之外就是控制总线。

图 3-17　8088 微处理器外部接口

图 3-18 是针对 8088 微处理器的引脚特点设计出的最小组态下的系统总线接口电路。ALE 为 CPU 提供的地址锁存信号，当该信号有效时，$A/D_0 \sim A/D_7$ 以及 $A_{16} \sim A_{19}$ 这些复用引脚输出地址信号，因此可以利用 3 组 74LS373 锁存器在 ALE 地址锁存使能信号的控制下锁存 CPU 输出的地址信号，从而提供分离的地址总线。DEN 为数据有效信号，当该信号有效时，$A/D_0 \sim A/D_7$ 这些复用引脚传输数据信号，数据的传输方向由 DT/R 引脚指示，可以通过一个 74LS245 双向缓冲器在 DEN 以及 DT/R 的控制下分离出数据总线。控制引脚没有复用，直接由 CPU 的引脚提供控制总线。

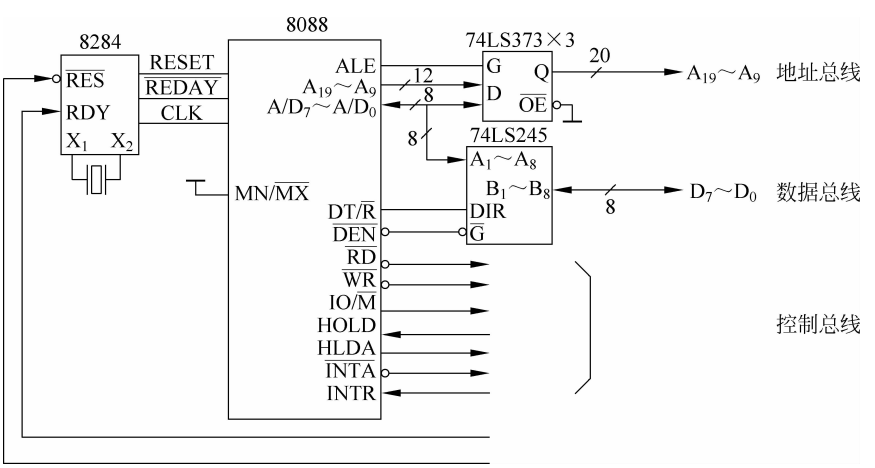

图 3-18　8088 微处理器最小组态系统总线接口电路

微型计算机的微处理器基本上都采用这种系统总线方式。在这种系统总线接口方式，不同类型的存储芯片或不同类型的 I/O 设备与微处理器相连时，都可以通过其自身针对系统总线特有的接口电路连接到系统总线上。一方面为用户设计各种计算机接口电路提供了较大的自由度，另一方面也给用户的接口电路设计增加了难度。

为降低用户设计复杂接口电路的难度，嵌入式微处理器将大部分接口电路都集成到了微处理器的内部，因此用户看到的大多数嵌入式微处理器提供的接口并非系统总线方式，而

是针对不同的微处理器外部设备或存储器类型提供了多种不同的接口。图 3-19 展示 32 位 Microchip 嵌入式芯片系列 PIC32MX5XX/6XX/7XX 的框图。

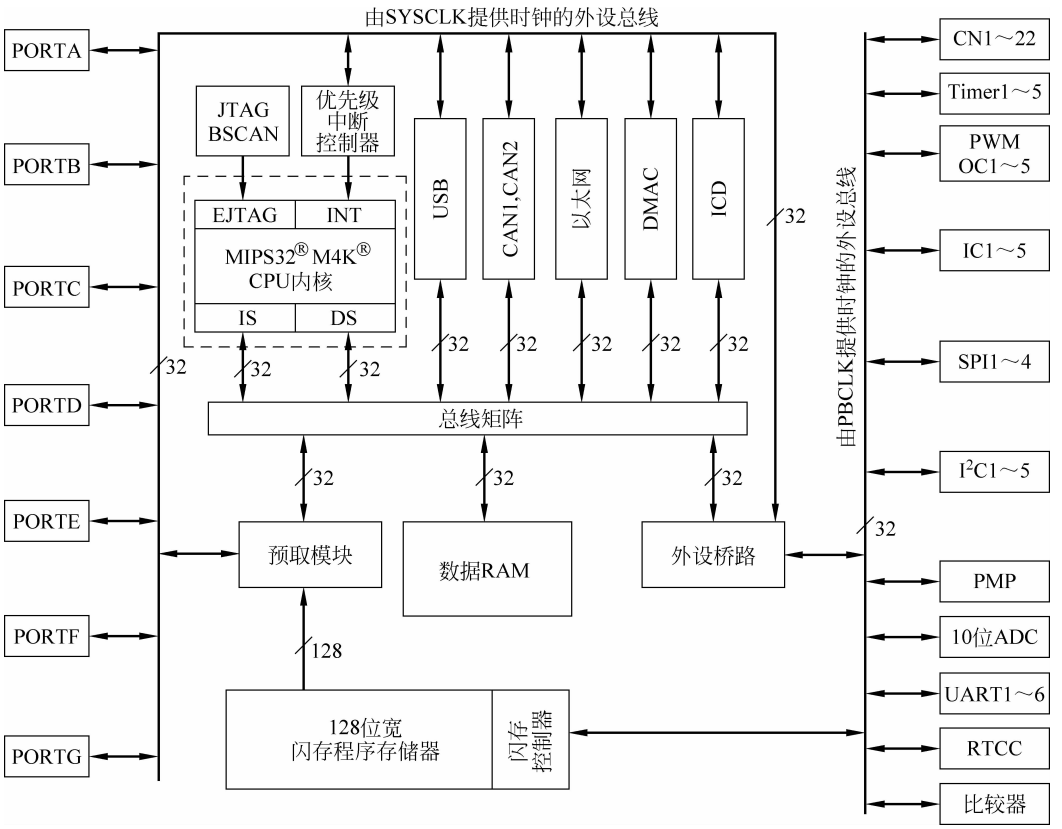

图 3-19　嵌入式芯片 PIC32MX5XX/6XX/7XX 系列框图

从图 3-19 中可以看出在这个芯片中已经集成了相当多的外围接口，如 USB，CAN，ethernet，I^2C，SPI，UART 等多种类型的处理器外围接口。实际上该芯片已经不再是单纯意义上的微处理器，它是一个微缩的集成计算机系统。这类嵌入式微控制器内微处理器仅只是芯片的一部分，如图 3-19 中的虚线框内的 MIPS32 CPU 内核。虽然嵌入式芯片外部引脚结构发生了较大的改变，但是从图 3-19 中可以看出该系统的 CPU 仍然是通过统一的总线结构（总线矩阵）与系统内各个部件相连。

为便于读者学习计算机工作基本原理，本书中的微处理器指采用统一的总线接口方式即三总线方式与外部部件实现数据通信的中央处理器 CPU。

3.8　MicroBlaze 微处理器简介

MicroBlaze 微处理器是 Xilinx 公司基于 FPGA 开发的一个软核类 MIPS 指令集微处理器，它支持 32 位的数据总线和 32 位的地址总线。基本结构如图 3-20 所示。它包含了简单 MIPS 微处理器的各个部件，并且功能更完善，能够支持的指令集更复杂。它将指令存储部件和数据存储部件放在微处理器之外，为实现指令和数据的访问，设计了专门的总线接口

部件，由于采用哈佛结构，因此有专门的指令总线和数据总线接口部件。该微处理器可以配置为大字节序或小字节序，默认情况下当采用 PLB 外部总线时为大字节序，采用 AXI4 外部总线时为小字节序。支持三级或五级流水线，内部具有可选的指令和数据 Cache，同时支持虚拟内存管理。它支持通过 PLB、LMB、AXI 总线与外围接口或部件相连。

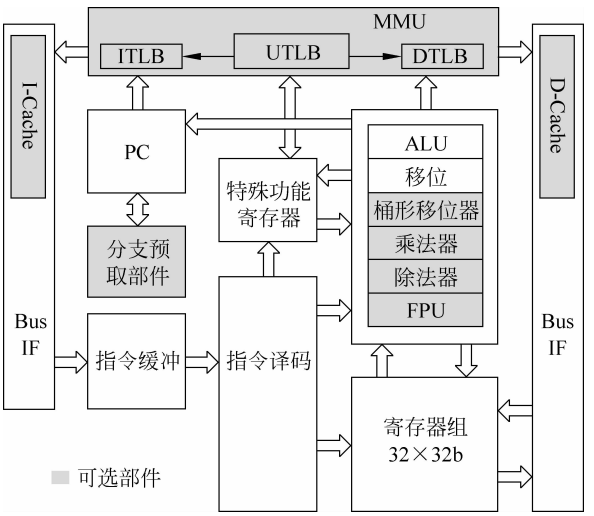

图 3-20 MicroBlaze 微处理器基本结构

该处理器支持两种类型的指令格式：A 型指令和 B 型指令。A 型指令相当于 MIPS 指令中的 R 型指令，其指令格式如图 3-21 所示。

图 3-21 A 型指令编码

B 型指令相当于 MIPS 指令里面的 I 型指令，其指令格式如图 3-22 所示。

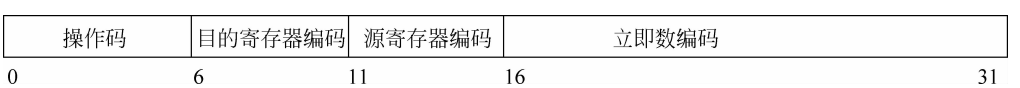

图 3-22 B 型指令编码

它将 MIPS 指令中的 J 型指令合并到 B 型指令中，此时目的寄存器编码为保存返回地址的寄存器编码，源寄存器不再是真正意义的源寄存器编码，各位具体含义如图 3-23 所示。

其中，D 表示是否延时之后再跳转：1 延时数个时钟周期再跳转；0 立即跳转。A 表示是否是绝对跳转：1 立即数符号扩展后为绝对跳转地址；0 立即数为相对跳转偏移，跳转地址为 PC＋IMM(立即数)。L 表示是否链接返回地址：1 保存下一条指令地址到目的寄存器作为返回地址；0 不保存返回地址。

图 3-23 跳转指令中源寄存器编码各位含义

该处理器具有 32 个 32 位的通用寄存器，其使用规则与 MIPS 微处理器的通用寄存器使用规则类似，命名为 R0～R31。其中 R14，R15，R16，R17 又用做异常返回地址，具体使用

规则参考中断技术。另外还具有 18 个 32 位的特殊功能寄存器，包括 PC，MSR 等。这些特殊功能寄存器根据用户对 MicroBlaze 微处理器的不同配置，不一定都存在。具体含义请读者参考其数据手册，本书不再一一介绍。

思考与练习

1. 微处理器的三种基本操作分别是什么？
2. 微处理器的基本构成包括哪些部分？这些部分分别起什么作用？
3. 微处理器的数据通路包括哪些部件？分别完成什么功能？
4. 试对比分析图 3-1 与图 3-12 微处理器构成的异同。
5. 举例分别简述 MIPS 三类指令的执行过程。
6. 采用 Verilog HDL 语言实现一个能执行以下指令集的简单微处理器：

(1) lw，sw 指令；

(2) add，sub，and，or，slt 指令；

(3) beq，j 指令。

7. 在题 6 指令集基础上，增加 addu，addiu 指令，修改图 3-9 简单 MIPS 微处理器数据通路构成，并完成 ALU 控制器和主控制器译码设计。
8. 简述流水线技术与超标量技术的特点。
9. 微处理器识别异常事件的方法有哪些？它们各有什么优缺点？
10. 微处理器保存断点的方法有哪些？它们各有什么优缺点？
11. 微处理器进入异常处理的方法有哪些？它们各有什么优缺点？
12. 参考 MicroBlaze 数据手册，说明 MSR 寄存器各位的具体含义。