目　录

第一章　数字信号处理器发展概述

数字信号处理中的卷积、相关、窗口及FFT等，都是频繁进行大量数据的乘法和加法运算。通用微处理器（例如，8086、68000等）因适用目的不同，在运算速度上难以适应信号实时处理的要求。随着大规模集成技术的发展，开发了一种集成有高速度乘法器硬件、能够快速进行乘法和加法运算、适用于高速数字信号处理的单芯片大规模集成电路，这就是数字信号处理器（DSP）。近年来，DSP在功能、处理速度和处理能力方面，都取得了划时代的突破，并广泛应用在数据通信、语声信号处理、智能化仪器、自动控制等技术领域中，展示了其独特的应用潜力。

1.1　第一代DSP

早期的DSP器件是1979年Intel公司开发的2920和AMI公司的S2811。但典型DSP还是1980年NEC公司开发的μPD7720和Bell研究所开发的DSP20。由于DSP器件在数字信号处理中展示了独特的优点，随之于1982年日立公司开发了61810，TI公司开发了TMS32010等DSP器件。这个时期开发的DSP器件，称为第一代DSP，其主要指标如表1-1所示。

表1-1　第一代DSP

型号 / 项目	μPD7720 (NEC)	TMS32010 (TI)	HD61810 (日立)	MB8764 (富士通)
硬件工艺	NMOS	NMOS	CMOS	CMOS
引脚数	28(DIP)	40(DIP)	40(DIP)	88(PGA)
命令周期	250ns	200ns	250ns	100ns
数据字长	16位	16位	16位	16位
在片程序存储器	512×23位	1.5K×16位 可外接扩至4K字	512×22位	1K×24位
在片数据ROM 在片数据RAM	512×13位 128×16位	与程序ROM并用 144×16位	128×16位 200×16位	与程序ROM并用 256×16位
乘法器	16×16→31	16×16→32	(12+4)×(12+4)	16×16→26
输入输出	总线8位	总线16位	总线16位	总线16位
I/O接口	串行I/O	16位I/O	串行I/O	16位I/O

第一代DSP的构成特点为：

(1)片装乘法器硬件，具有将乘法器和累加器以流水线方式连接的总线，能高速进行连续的乘法和累加运算；

(2)采用哈佛结构，数据总线和程序总线分离，可同时进行指令的读取和数据运算；

(3)在片程序存储器和数据存储器；

(4)备有与A/D、D/A变换器等外围设备相接的接口，乘法器和累加器的位数在16位以上，能实现高精度的数据运算；

(5)指令基本上在一个机器周期内进行处理；

(6)可进行数据的浮动小数点运算，动态范围大。

表 1－2 为 μPD7720 和通用 8086 微处理器运算性能比较。从表中可以看到：μPD7720 和 8086 虽采用相同的 MOS 工艺和时钟，但同样完成 16 节 FIR 数字滤波器的运算，μPD7720 要比 8086 快几倍。

表 1－2 μPD7720 和 8086 性能比较

型号 / 项目	μPD7720	8086
制造技术	NMOS	NMOS
时钟	8MHz	8MHz
元件数	40000	24000
机器周期	250ns	最小 250ns
16 节 FIR 滤波器	50μs	270μs

1.2 第二代 DSP

1985 年 TI 公司开发的 TMS32020，1986 年日本 NEC 公司开发的 μPD77230 等通用 DSP，与第一代 DSP 相比，在功能上、速度上及内存容量方面，都取得了划时代的突破：

(1)运算速度更高，机器周期减少到 100ns，运算能力达 8～40MFLOPS；

(2)和大型计算机一样，能进行 32 位的浮点运算，运算精度更高；

(3)片装大容量数据存储器和程序存储器，并大大扩展外部存储器空间(达 64K)；

(4)强化和完善了指令功能及寻址方式。

表 1－3 为第二代 DSP 性能一览表。表 1－4 是第二代 DSP 典型器件 μPD77230 处理能力测试结果。

表 1－3 第二代 DSP

型号 / 项目	TMS32020/C25 (TI)	μPD77230 (NEC)	ADSP2100 (AD)	DSP56000 (MOTOROLA)
机器周期	200/100ns	150ns	125ns	97.6ns
乘·加执行时间	200ns/100ns	150ns	125ns	97.6ns
数据字长	16 位	32 位浮点	16 位	24 位
指令字长	16 位	32 位	24 位	24 位
在片程序 ROM	－/4K 字	2K 字	—	2K 字
在片程序 RAM	256 字	—	16 字 (CACHE)	—
在片数据 ROM	与程序 ROM 并用	1K 字	—	512 字
在片数据 RAM	544 字	1K 字	—	512 字
程序存储器空间	64K 字	4K 字	16K 字	64K 字
数据存储器空间	64K 字	8K 字	32K 字	128K 字
ALU	32 位	55 位	16 位	56 位
ACC	32 位	55 位	40 位	56 位

输入·输出接口	16 位并行串行	32 位并行串行	存储器屏蔽	24 位可编程
DMA 接口	有	无	有	有
制造技术	NMOS/CMOS	CMOS	CMOS	CMOS
消耗功率	1.2W/0.6W	0.8W	0.5W	<1.0W
开发年月	1985/1988	1986	1986	1987

表 1-4 μPD77230 运算能力测试

数值运算	除算	4.8μs
	平方根	9.0μs
	sin	10.8μs
	cos	10.8μs
	ATAN	40μs
数字信号处理	2 节 FIR	0.9μs
	32 节 FIR	5.25μs
	32 点 FFT	150μs
	512 点 FFT	4.7ms
	1024 点 FFT	12.3ms

1.3 第三代 DSP 及 DSP 的发展动向

从 1980 年第一代 DSP 付诸实用，随着半导体集成电路技术的急速发展和高速实时信号处理技术的需求，在短短的十几年中，单片数字信号处理器取得了划时代的发展。1987 年，TI 公司开发出高速、高性能、高内存、并且可使用高级语言的第三代数据字信号处理器 TMS320C30，1991 年，又推出了支持 32 位浮点运算的、速为 275MIPS、具有 340MB/秒数据传输能力的、真正支持并行操作并与 TMS320C30 原代码兼容的 TMS320C40。表 1-5 为近年来开发的 32 位浮点数信号处理器功能一览表：

表 1-5 最新 32 位浮点 DSP

型号 / 项目	TMS320C30 (TI)	DSP96002 (摩托罗拉)	ZR34325 (ZORAN)	μPD77240 (NEC)	MB86232 (富士通)	DSP32C (AT&T)
指令周期	60ns	60ns	80ns	90ns	75ns	60ns
浮点形式	2 的补码	IEEE	IEEE	2 的补码	IEEE	
存储器空间	16M 字	12G 字	16M 字	16M 字	1M 字	4M 字
在片 RAM	2×1K	3×512	2×64	2×512	512	1K/1536
在片 ROM	4K	2×512	256	2K+1K	1K	2K
在片高速	64	—	4	—	—	—
ALU	32E8	32E11	44 位	47E8	24E8	32E8
中断 (内/外)	7/4	0/3	22/1		/4	/3

I/O	DMA SIO×2 定时器×2	DMA	DMA	输入2位 输出2位	DMA PIO SIO×2 定时器	PIO
封　装	181端 PGA	195端 PGA	84端 PGA	132端 PGA	208端 PGA	195端 PGA
功　耗	<1W	<1W	<1W	<2.3W	<1W	<1W

由表1-5中可看出，在短短的几年中，通用DSP的发展取得了划时代的突破。人们期待着性能更高、运算速度更快的新DSP产品。从其开发动向来看，为满足复杂信号处理的需要，例如：运动图像处理、连续发声非特定说话人语音识别等，高速化、高性能化、片装存储器大容量化仍然是DSP的开发方向。近年来开发的DSP，大多采用32位浮点运算机构，具有和大型计算机相同的浮点运算能力和精度，更加适合于数字信号处理的要求。

(1) 浮点运算的DSP容易导入高级计算机语言。采用高级语言能缩短程序的开发时间，提高开发效益，因此，用于数字信号处理器的高级语言研究正蓬勃展开。使用高级语言，已成为第三代DSP的重要特征。

(2) 目前使用的DSP其浮点数据的表示格式有两种：一是2的补码形式，另一种是IEEE格式，为便于DSP和高档微计算机相匹配以构成实用系统，采用IEEE格式将为主流。

为充分发挥一般CPU所不具有的DSP的长处，扩大DSP的应用范围，各厂家均致力于开发和完善DSP支援系统，特别是软件开发支援系统。值得指出的是，最近国外已研究出利用通用微型计算机和DSP板组成的DSP开发系统。这个系统主机侧使用高级语言C，PASCAL，DSP板侧采用汇编语言，用微计算机开发DSP软件，直接生成目标代码。如果再配置A/D、D/A板，则可构成一个比较完善数字信号处理系统，完成数字滤波器设计、图像处理、声音处理等。由于这种DSP开发系统的硬件成本远远低于DSP的专用开发支持系统，特别是随着通用32位微型计算机的普及，可用最小的开发成本获得最大的开发效益，因而有可能成为当前DSP应用软件开发系统的主流。

第二章　TMS32010 系列处理器

2.1 TMS32010 概述

TMS32010 是 1983 年开发的第一代数字信号处理器。TMS32010 采用 2．4 μmNMOS 工艺，使用单 +5V 电源，时钟为 20MHz，片内带有时钟电路，指令周期为 200ns。片内数据 RAM 为 144 字，片外程序存储器可扩展到 4K 字。

TMS32010 在运算中采用带符号的 2 的补码定点运算，并且有一个 32 位累加器/算术逻辑单元。除了用于信号定标的 0～15 位定标移位器外，还有一个对累加器输出进行 0、1、4 位移位的行数据移位器。TMS32010 有 8 个输入口和 8 个输出口，允许 40 兆位/秒速率的 16 位双向数据传送。它的中断可采用堆栈进行全部上下文保护。

TMS32010 特性如下：

- 运算速度 5MIPS（20MHz 工作时钟）
- 指令周期 200ns
- 片内数据 RAM144 字
- 外部程序存储器可扩展至 4K 字并全速运行（5MIPS）
- 16 位指令/数据字
- 32 位算术运算单元/累加器
- 16 位乘 16 位运算用时 200ns
- 0～15 位桶形移位器
- 16 位双向数据传输
- 全部上下文保护中断
- 40 引脚双列直插封装件

TMS32010 自 1983 年出品以来，在许多领域得到了广泛的应用，同时根据各自应用范围不同，又开发了一些新的品种。

TMS320C10 与 TMS320C10－25 是 1985 年开发的新产品。TMS320C10 采用了2.0 μmCMOS 工艺，功耗是 NMOS 芯片的 1/6，为 165mW，指令周期为 200ns，并与 TMS32010 目标代码及硬件兼容。通常用于需要低耗的场合。TMS320C10－25 是 TMS320C10 的变通产品，主时钟为 25MHz，指令周期是 160ns，具有低功耗、高速度的特点。

TMS320C15/E15 是 1986 年开发的产品。TMS320C15/E15 采用了2.0 μmCMOS 工艺，指令周期为 200ns，与 TMS32010 的目标代码和硬件完全兼容。但片内 RAM 扩展到 256 字，并增加了 4K 字 ROM（TMS320E15 型式）。

TMS320C17/E17 是 1987 年开发的产品。TMS320C17/E17 采用了2.0 μmCMOS 工艺，指令周期为 200ns，是一种专用的微处理器，片内程序 ROM（TMS320C17）或 EPROM（TMS320E17）为 4K 字，具有一个双工串行接口，片内压缩扩展硬件（μ 律/A 律）和一个串行口定时器。程序和 TMS32010 目标代码兼容。

TMS32010 系列管脚配置如图 2－1 所示。各管脚功能如表 2－1。

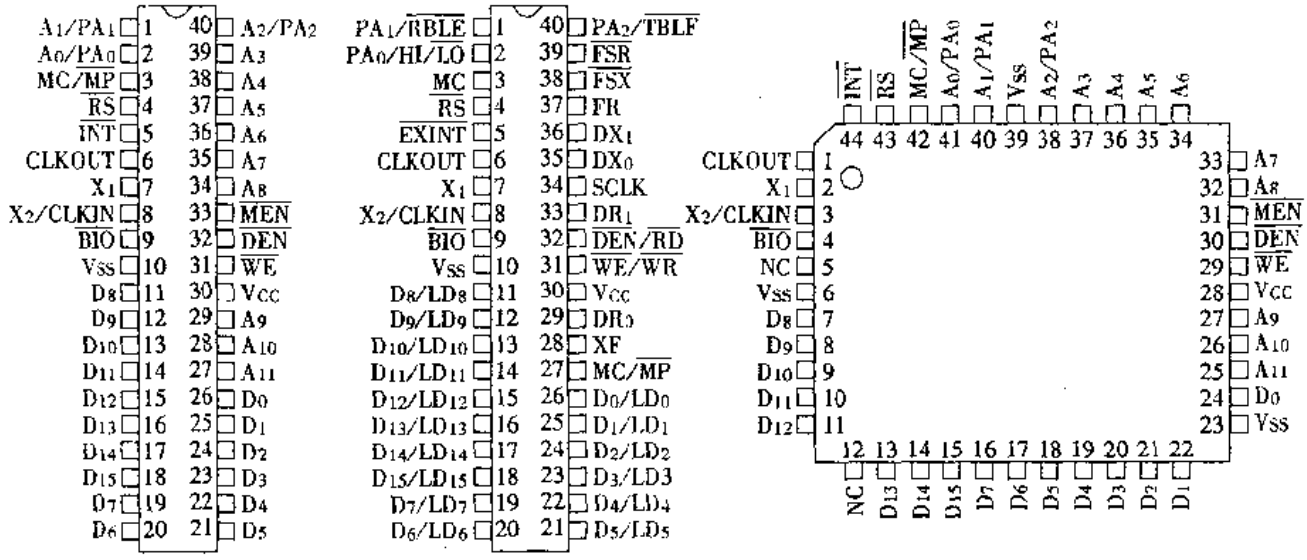


图 2－1　TMS32010 系列引脚图配置

表 2－1　TMS32010－320E15 引脚功能

信　号	引脚序号	输入输出	功　能
V_{cc}	30	输入	电源电压 +5V
V_{ss}	10	输入	接地
X2/CLKIN	8	输入	晶振输入或外部时钟输入
X1	7	输出	内部时钟输出
CLKOUT	6	输出	时钟输出信号。CLKOUT 的频率为外部时钟输入或内部晶振频率的 1/4，占空比为 0.5
$\overline{WE}$	31	输出	在“L”有效时，表示来自 TMS32010 数据总线上的数据有效。仅在 OUT 指令的第 1 周期和 TBLW 指令的第 2 周期有效。$\overline{WE}$有效时，$\overline{MEN}$及$\overline{DEN}$常为“H”电平。
$\overline{DEN}$	32	输出	“L”有效，表示 TMS32010 处于从数据总线接收数据的状态。仅在 IN 指令的第一周期有效，此时$\overline{MEN}$和$\overline{WE}$常为“H”。
$\overline{MEN}$	33	输出	“L”有效，$\overline{WE}$和$\overline{DEN}$无效时，对在片离片程序程序存储器，取指时有效。
$\overline{RS}$	4	输入	复位。
$\overline{INT}$	5	输入	中断。在$\overline{INT}$引脚加下降沿或电平作为中断产生信号。“L”电平时允许中断。
$\overline{BIO}$	9	输入	I/O 分支控制。在执行 BIOZ 指令时，$\overline{BIO}$引脚置“L”电平时，则转向指令指定的地址。

D15	18		D15（MSB）～D0（LSB）
↓	↓	输入/	数据总线当$\overline{WE}$为“L”电平时有效，此外常为高阻。
D8	11	输出/	
D7	19	高阻	
↓	↓		
D0	26		
A11	27	输出	程序存储器地址 A11～A0 及通道
A10	28		地址 PA2（MSB）～PA0（LSB）、A11～A0 非高阻。
A9	29		执行 IN/OUT 指令时，通道地址为 PA2～PA0。
A8	34		
↓	↓		
A3	39		
A2/PA2	40		
A1/PA1	1		
A0/PA0	2		
MC/$\overline{MP}$	3	输入	设置微计算机方式或微处理器方式。 当 MC/$\overline{MP}$为 0 时，为微处理器方式，程序存储器只能对外部存储器进行存取，而对片内 EPROM 或掩模 ROM 不能进行存取。当 MC/$\overline{MP}$为 1 时，为微计算机方式。可使用内部掩模 ROM 和 EPROM。

2.2 TMS32010 的硬件结构

TMS32010 信号处理器的硬件结构框图如图 2－2 所示。

一、运算单元

运算单元主要由 16 位×16 位的补码乘法器和累加器组成。

乘法运算时，先将数据存储器的数据（用 LT 指令）置入 T 寄存器，再用乘算指令（MPY）将数据存储器的数据和 T 寄存器的数据相乘，并将结果存入 P 寄存器。

P 寄存器的数据可用传送指令（PAC 指令）置入累加器。实际上，TMS32010 有 LTA、LTD 等复合指令，可以高效率地进行 T 寄存器的数据置入以及送入累加器的加算数据的传送等操作。累加器长 32 位，可直接取入乘算结果。另外还设有将数据存储器的数据从 0 位到 15 位进行任意移位并置入累加器的定标移位器。累加器输出的高 16 位可进行 0 位、1 位、4 位的移位操作，以便在固定小数点运算中对齐小数点。

二、数据存储器，DP，AR

TMS32010 片内存储器（RAM）为 144 字，分为两页，即 0 页和 1 页。0 页容量为 128 字，1 页容量为 16 字，字长 16 位。TMS320C15/E15 及 TMS320C17/E17 片内 RAM 有 256 字，其中 0 页和 1 页各 128 字。寻址时由页面指针 DP 进行页选择。

数据存储器寻址有两种方式，即直接寻址方式和间接寻址方式。

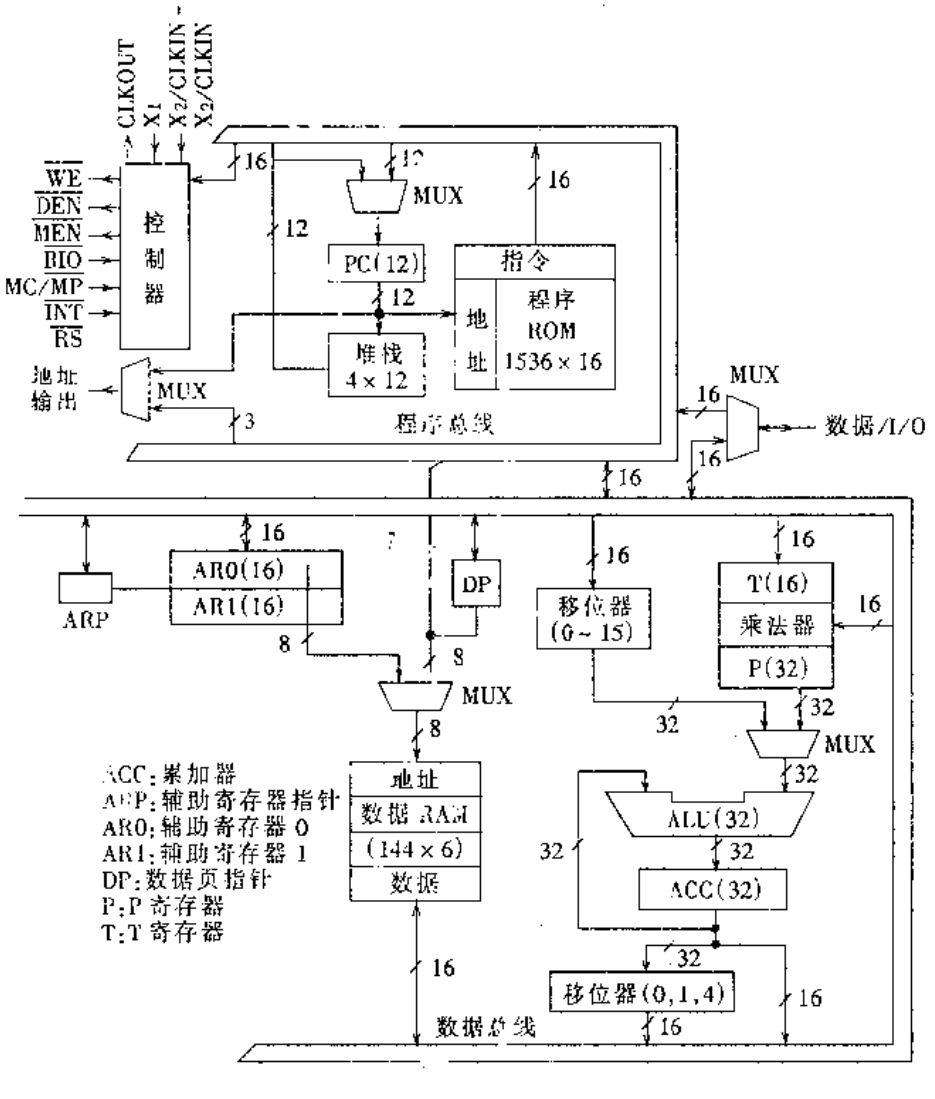


图 2－2　TMS32010 构成框图

直接寻址方式如图 2－3 所示。在 8 位地址操作数中，1 位是数据页面指针，其余 7 位嵌在操作码中。由 DP 和操作码实现直接寻址。

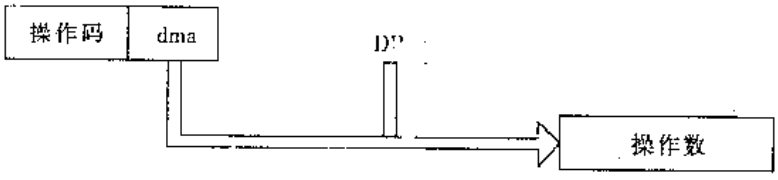


图 2－3　直接寻址方式

间接寻址方式时，由辅助寄存器指针 ARP 选择辅助寄存器 AR0 和 AR1，如图 2－4 所示。指令执行后 AR0 或 AR1 的内容自动进行加减。间接寻址方式时的指令方式如表 2－2 所示。

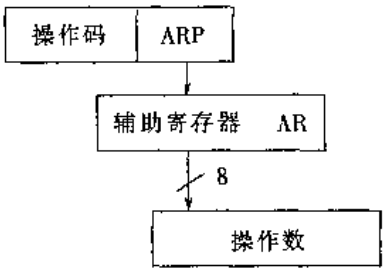


图 2－4 间接寻址方式

辅助寄存器 AR0 和 AR1 不仅可以用作存储器指针，在 BANZ 指令中还可用作循环计数器，作为暂时保留数据的临时寄存器。

表 2－2 间接寻址的指令方式

寻址方式	操　　作
OP＊（，NAPR）	AR 不变化
OP＊＋（，NAPR）	用当前 AR 内容指定的数据存储器地址进行存取后，AR 的内容加 1
OP＊－（，NAPR）	用当前 AR 内容指定的数据存储器地址进行存取后，AR 内容减 1

三、程序总线

程序总线挂接程序计数器（PC）、堆栈（12 位×4 级）、掩模 ROM（1.5 K 字）及控制单元等。

程序存储器空间为 4K 字，12 根地址线（A11～A0）；I/O 通道 PA0～PA2 与地址总线多路复用。

与通用微处理器一样，堆栈不是外部存储器的指针，而是一个 4 级堆栈。中断产生或调用子程序时，（PC＋1）的值进入堆栈；恢复中断或调用结束时，由 RET 指令从堆栈弹出，其值返回到 PC 内。此外，表读、写指令也可以使用堆栈的一个级，因而在使用表读、表写指令产生中断或调用子程序时，要注意堆栈的溢出。

根据需要，也可由程序将堆栈扩展到数据存储器或程序存储器，或用 TRLW 指令在外部存储器中建立极大的堆栈。

控制单元输出控制外部存储器和 I/O 的信号及各种接口信号。

四、状态寄存器

如图 2－5 所示，状态寄存器由 5 个状态位组成。其中仅 OV 是由运算结果引起内容变化的状态位。OV 状态位在累加器溢出时是“H”电平，经过指令 BV 测试后可清除。其他各状态位由图中所示指令进行置位或复位。

累加器溢出标志寄存器 OV	溢出方式位 OVM	中断屏蔽位 INTM	辅助寄存器指针 ARP	数据存储器页指针 DP
	SOVM/ROVM	$\overline{EINT}$ $\overline{DINT}$	MAR/LARP 或间接寻址方式	LDPK/LDP

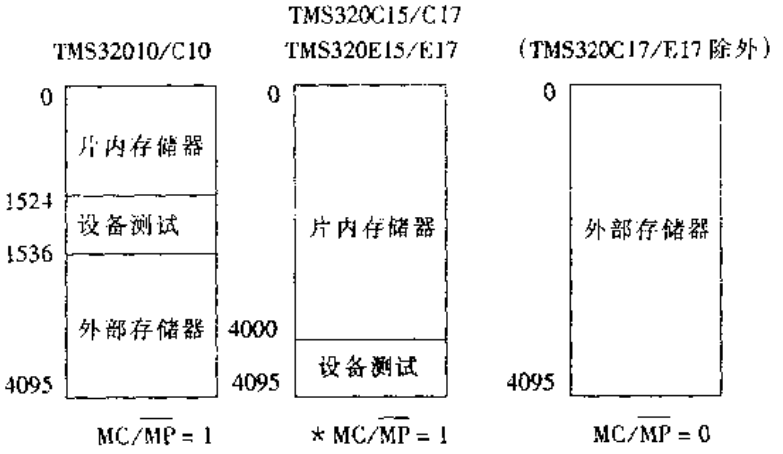
图 2-5 状态寄存器

使用 SST/LST 指令对状态寄存器进行存储和装入。直接寻址方式时，状态寄存器内容存入 1 页数据存储器。由于 TMS32010 的数据存储器有 144 字，因此地址需用 0～15 的数值。用间接寻址方式存储时不存在任何问题。

将数据置入状态寄存器使用 LST 指令。但该指令对中断屏蔽位 INTM 无效。中断屏蔽位由 EINT/DINT 指令进行置位和复位。

五、存储器

TMS32010 系列具有片内 ROM 或 EPROM，同时，通过 MC/MP 切换引脚可进行外部存储器和在片掩模 ROM/EPROM 之间的切换。如图 2-6 所示。



(a) 微计算机方式　　(b) 微处理器方式

图 2-6 存储器分配图（程序存储器）

一般而言，TMS32010/320C15/320C17 多不用掩模 ROM 而用外部存储器作为程序存储器。但 320E15/320E17 有在片 EPROM，在微计算机方式时用作程序存储器。TMS320C17/320E17 无外部地址线，在微计算机方式工作时，不能配外部存储器。

六、I/O 接口

I/O 接口可使用 8 个输入、输出通道。通道地址 PA0～PA2 与总线多路复用，A2～A0 为通道地址。

七、$\overline{BIO}$与中断输入$\overline{INT}$

$\overline{BIO}$和$\overline{INT}$是输入与外部系统接口信号的引脚。

当输入外部标志置$\overline{BIO}$引脚为“L”电平时，使用 BIOZ 指令可进行分支。$\overline{BIO}$引脚还可用于采样频率和程序的同步（如图 2-7）及 I/O 接口的状态监视。当$\overline{BIO}$从“H”变为“L”电平时，DSP 从等待状态转移到执行状态，直至整体处理结束、下一个采样数据有效。

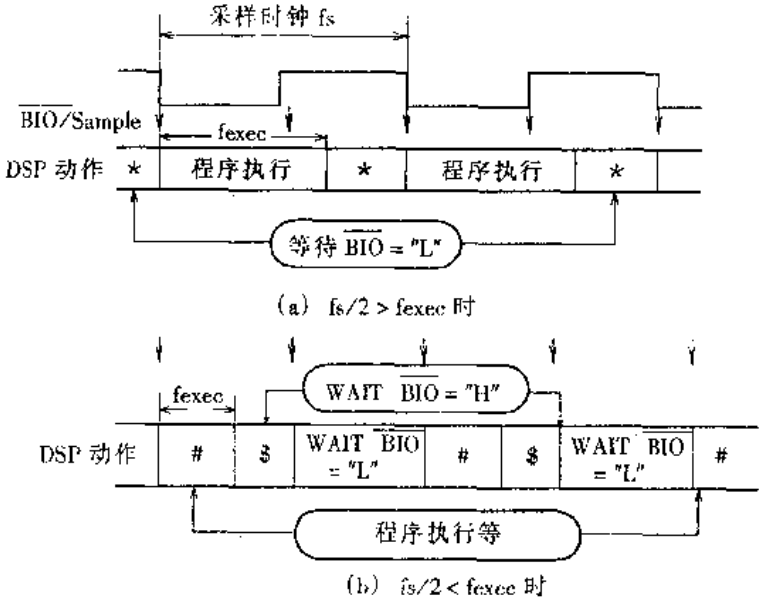


图 2-7 使用$\overline{BIO}$引脚的采样频率的同步

当 DSP 的实时处理时间比采样周期小时，可按图 2-7（b）进行设定，检测出$\overline{BIO}$的“H”，“L”电平，则可完全与采样时钟同步工作。

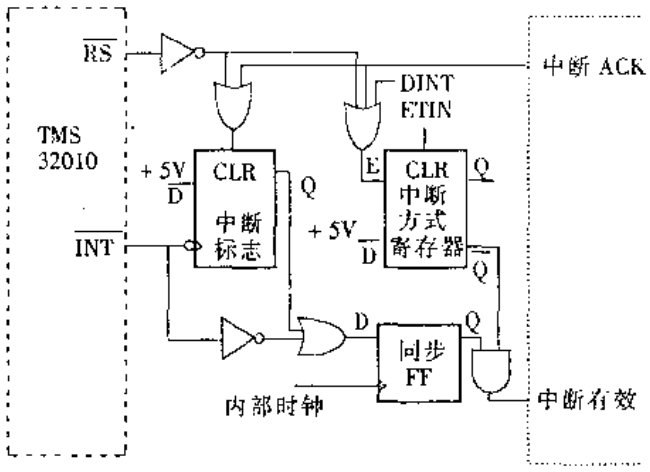


图 2-8 中断输入的内部结构

$\overline{INT}$是中断信号输入引脚。中断输入的内部结构如图 2－8 所示。允许中断时需在$\overline{INT}$引脚加“L”电平信号，在脉冲的下降沿产生中断并转移到中断矢量 002 地址，同时 PC 的值压入堆栈。但是，在下述三种情况下，中断处理要延迟到处理结束：

（1）乘算全周期结束；

（2）乘算指令下一条指令执行完毕；

（3）$\overline{EINT}$指令的下一条指令执行完毕。

图 2－9 表示中断发生时的指令顺序。中断信号加入时从 N 地址和 N＋1 地址取出的指令，在中断前执行；但不执行从 N＋2 地址伪取的内容。待中断处理子程序执行完毕后，再取 N＋2 地址的内容执行之。

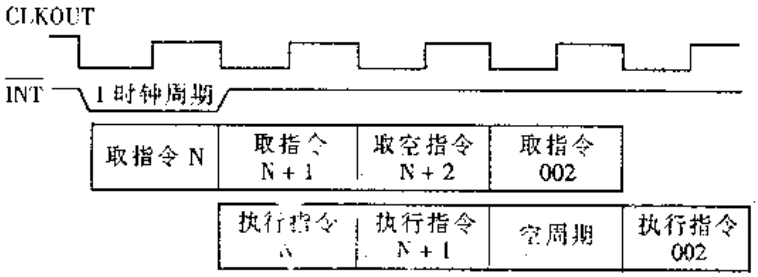


图 2－9　中断时指令顺序

八、复位

在 RS 引脚加上长达 5 个时钟周期以上的“L”电平的脉冲时，TMS32010 则复位。复位时 TMS32010 各引脚信号如图 2－10 所示。

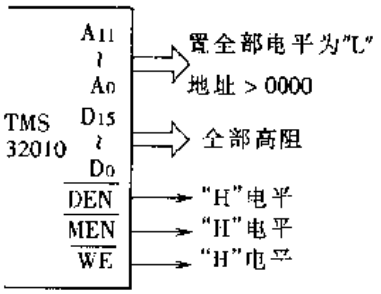


图 2－10　复位时引脚信号

因此，当在主处理器和 32010 之间设置共用存储器时（如图 2－11），DSP 地址总线和存储器之间，必需设置一个三态缓冲器。

TMS32010 的 RS 引脚一旦设置为“H”电平，则复位状态结束。

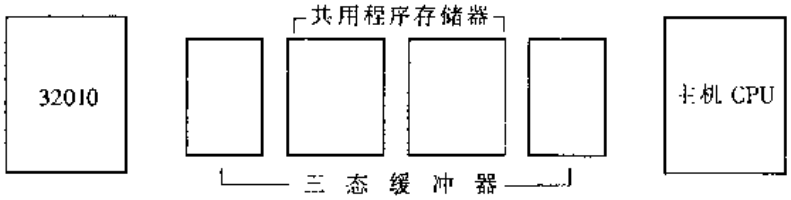


图 2－11　TMS32010 与主机接口

2.3 寻址方式和指令系统

一、TMS32010 寻址方式

TMS32010 有直接寻址、间接寻址、立即寻址等三种寻址方式。立即寻址又有指令助记符直接寻址和指令助记符间接寻址。此外，数据存储器和程序存储器间数据传送指令由累加器指定程序存储器空间的地址；分支指令时程序存储器仅有直接寻址；使用调用子程序指令时，程序存储器可直接寻址或由累加器寻址。

1．直接寻址

直接寻址方式中操作数值本身就是地址。这种方式下，由数据页面指针（DP）决定对包含 128×16 位字的 0 页进行存取或对包含 16×16 位字的 1 页进行存取。如图 2-12 所示，1 位的数据页面指针和程序总线上指令字中 7 位操作数连接生成 8 位数据存储器地址。

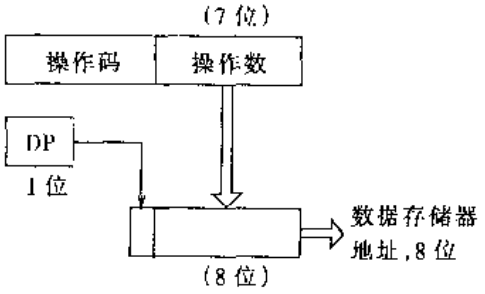


图 2-12　直接寻址方式

2．间接寻址

这种方式是由两个 16 位辅助寄存器 AR0 和 AR1 中的一个来完成的。该寄存器的低 8 位对数据 RAM 寻址。辅助寄存器指针（ARP）是状态寄存器中的一位，用来确定选择哪一个辅助寄存器。间接寻址方式时数据存储器地址生成如图 2-13 所示。

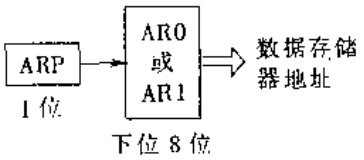


图 2-13　间接寻址数据存储器地址生成

间接寻址方式有几种不同的形式。一是指令执行后，辅助寄存器能自动增量或自动减量；二是相同的指令执行后，用以后的指令可变更辅助寄存器指针的值；在自动增量、减量或变更辅助寄存器指针值时，指令的执行时间不变。

用汇编语言表示间接寻址操作数的方法如表 2-3 所示。在同时书写操作数移位数、通

道地址和指定下一个有效辅助寄存器指针时，应先写移位数和通道地址，如图 2－14 所示。

表 2－3　间接寻址的形式

符　号	操　作
*	指令执行后，AR、ARP 的当前内容不变
* +	指令执行后，AR 内容增 1，ARP 的内容不变
* －	指令执行后，AR 内容减 1，ARP 内容不变
*，ARm	指令执行后，ARP 的值变为由 ARm 指定的值
* +，ARm	指令执行后，AR 的当前值增 1，ARP 值变为由 ARm 指定的值
* －，ARm	指令执行后，AR 的当前值减 1，ARP 值变为由 ARm 指定的值

* 表中 ARm 表示 AR0 或 AR1

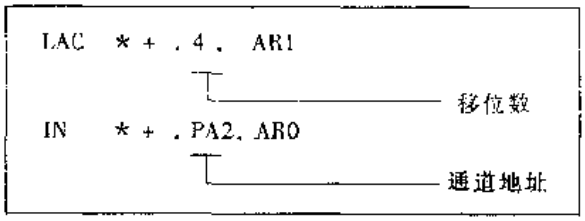


图 2－14　间接寻址时的书写顺序

3．立即寻址

有 5 条指令（LACK，LARK，LARP，LDRK，MPYK）可将立即操作数嵌入操作码中。

二、TMS32010 指令集

TMS32010 的指令按功能分类如表 2－4 所示。

表 2－4　TMS32010 指令

助记符	说明	周期数	字数	操作码指令寄存器 15	14	13	12	11	10	9 8 7 6 5 4 3 2 1 0
				累加器指令						
ABS	累加器的绝对值	1	1	0	1	1	1	1	1	1 1 1 0 0 0 1 0 0 0
ADD	加到累加器带移位	1	1	0	0	0	0	←—S—→		1 ←— D —→
ADDH	加到高位累加器	1	1	0	1	1	0	0	0	0 0 1 ←— D —→
ADDS	加到累加器无符号扩展	1	1	0	1	1	0	0	0	0 1 1 ←— D —→
AND	与累加器“与”	1	1	0	1	1	1	1	0	0 1 1 ←— D —→
LAC	送累加器带移位	1	1	0	0	1	0	←—S—→		1 ←— D —→
LACK	送累加器立即数	1	1	0	1	1	1	1	1	1 0 ←——K——→
OR	与累加器“或”	1	1	0	1	1	1	1	0	1 0 1 ←—D—→
SACH	储存高位累加器位带移位	1	1	0	1	0	1	1	←X→	1 ←— D —→

SACL	储存低位累加器位	1	1	0	1	0	1	0	0	001 ←— D —→
SUB	从累加器中减带移位	1	1	0	0	0	1	←—S—→		1←— D —→
SUBC	条件减法（用于除法）	1	1	0	1	1	0	0	1	001 ←— D —→
SUBH	从高位累加器位减	1	1	0	1	1	0	0	0	101 ←— D —→
SUBS	从累加器中减带无符号扩展	1	1	0	1	1	0	0	0	111 ←— D —→
XOR	与累加器“异或”	1	1	0	1	1	1	1	0	001 ←— D —→
LAC	累加器回零	1	1	0	1	1	1	1	1	1110001001
ZALH	累加器回零并送入高位位	1	1	0	1	1	0	0	1	011 ←— D —→
ZALS	累加器回零并送入带无符号扩展的低位位	1	1	0	1	1	0	0	1	101 ←— D —→
	T寄存器，P寄存器和乘法指令									
APAC	P寄存器加到累加器	1	1	0	1	1	1	1	1	1110001111
LT	送入T寄存器	1	1	0	1	1	0	1	0	101 ←— D —→
LTA	LTA把LT和APAC组合成一条	1	1	0	1	1	0	1	1	001 ←— D —→
LTD	LTD把LT，APAC和DMOV组合一条指令	1	1	0	1	1	0	1	0	111 ←— D —→
MPY	与T寄存器相乘；乘积存入P	1	1	0	1	1	0	1	1	011 ←— D —→
MPYK	T寄存器与立即数相乘；乘寄存器	1	1	1	0	0	←————			K————→
PAC	从P寄存器送入累加器	1	1	0	1	1	1	1	1	1110001110
SPAC	累加器减P寄存器	1	1	0	1	1	1	1	1	1110010000
	控制指令									
DINT	禁止中断	1	1	0	1	1	1	1	1	1110000000
EINT	允许中断	1	1	0	1	1	1	1	1	1110000010
LST	送入状态寄存器	1	1	0	1	1	1	1	0	111 ←— D —→
NOP	空操作	1	1	0	1	1	1	1	1	1110000000
POP	弹出堆栈送累加器	2	1	0	1	1	1	1	1	1110011101
PUSH	从累加器压入堆栈	2	1	1	1	1	1	1	1	1110011100
ROVM	复位溢出方式	1	1	0	1	1	1	1	1	1110001010
SOVM	置溢出方式	1	1	0	1	1	1	1	1	1110001011
SST	贮存状态寄存器	1	1	0	1	1	1	1	1	001 ←— D —→
	I/O和数据存储器操作									
DMOV	数据存储器地址单元内容重写在下一个地址单元	1	1	0	1	1	0	1	0	011 ←— D —→
IN	从端口输入数据	2	1	0	1	0	0	0	P	A 1 ←—D—→
OUT	数据输出到端口	2	1	0	1	0	0	1	P	A 1 ←—D—→
TBLR	表格从程序存储器读到数据RAM	3	1	0	1	1	0	0	1	111 ←— D —→
TBLW	表格从数据RAM写到程序存储器	3	1	0	1	1	1	1	1	011 ←— D —→
	辅助寄存器和数据页面指针指令									
LAR	送入辅助寄存器	1	1	0	0	1	1	1	0	0 R 1 ←— D —→
LARK	送入辅助寄存器立即数	1	1	0	1	1	1	0	0	0 R←———D———→

LARP	送入辅助寄存器指针立即数	1	1	0	1	1	0	1	0	0 0 1 0 0 0 0 0 0 K
LDP	送入数据存储器页面指针	1	1	0	1	1	0	1	1	1 1 1 ←— D —→
LDPK	送入数据存储器页面指针立即数	1	1	0	1	1	0	1	1	1 0 0 0 0 0 0 0 0 K
MAR	修改辅助寄存器和指针	1	1	0	1	1	0	1	1	1 1 0 ←— D —→
SAR	储存辅助寄存器	1	1	0	0	1	1	0	0	0 R 1 ←— D —→
				转移指令						
B	无条件转移	2	2	1	1	1	1	1	0	0 1 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BANZ	辅助寄存器非零转移	2	2	1	1	1	1	0	1	0 0 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BGEZ	如果累加器≥0则转移	2	2	1	1	1	1	1	1	0 1 0 0 0 0 1 0 0 0
				0	0	0	0	←——转移地址 —→		
BGZ	如果累加器>0则转移	2	2	1	1	1	1	1	1	0 0 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BIOG	BIO=0则转移	2	2	1	1	1	1	0	1	1 0 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BLEZ	如果累加器≤0则转移	2	2	1	1	1	1	1	0	1 1 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BLZ	如果累加器<0则转移	2	2	1	1	1	1	1	0	1 0 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BNZ	如果加器≠0则转移	2	2	1	1	1	1	1	1	1 0 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BV	溢出转移则转移	2	2	1	1	1	1	0	1	0 1 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
BZ	如果累加器=0则转移	2	2	1	1	1	1	1	1	1 1 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
CALA	根据累加器调用子程序	2	2	0	1	1	1	1	1	1 1 1 0 0 0 1 1 0 0
CALL	立即调用子程序	2	2	1	1	1	1	1	0	0 0 0 0 0 0 0 0 0 0
				0	0	0	0	←——转移地址 —→		
RET	从子程序返回	2	1	1	1	1	1	1	1	1 1 0 0 0 1 1 0 1

TMS32010 支持交叉汇编语言的伪指令如表 2－5 所示。

表 2－5　TMS32010 伪指令

伪 指 令	功　　能
AORG	指定程序存储器的首地址
BSS	预约数据存储器内空区域
DORG	指定数据存储器的首地址
END	指示源程序结束
EQU	定义常数

三、汇编语言语法

1．数据表示法

(1) 10 进制数（整数）

例：1234，8752，－5322

(2) 16 进制数

数据打头用“$”，例如，$5678

(3) 10 进制数（小数）

用10 进制数书写的小数，附加 Qn，n 表示小数点后的位数。例如，0.5Q15，－1.72Q12 等。这种格式称为 Q 格式。

2. 指令行的构成

汇编语言源语句的各行构成例如下：

LOOP	LAC	data1，4	ACC←左移 4 位
（标号）	（指令助记符）	（操作数）	（注　　释）

一般由标号、指令助记符、操作数和注释四部分组成，各字段的定界符为一个以上的空格或标记码。各字段书写内容如下：

(1) 标号字段

标号或符号书写在每行的行首。字符个数虽无限制，但只有前 8 个字符有效。在该字段内不写任何东西时，应打入一个以上的空格或写入标记。

若第一个字符为 * 时，该行为注释行。

(2) 指令字段

在该字段内书写指令或伪指令的助记符且必须用大写字符书写。

(3) 操作数字段

操作数的个数超过一个时，各操作数间用“,”号分开。操作数也可以是用 + 、－运算符组成的表达式，例如，A + B，X1 + X2 － $10 等。

2.4 TMS32010 程序设计

一、程序结构

TMS32010 的程序如图 2－15 所示那样，主要由三部分组成：

·DSP 内部初始化

·设定常数

·程序体

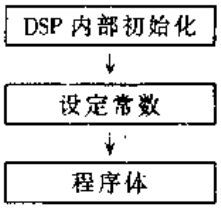


图 2－15　基本程序流程

1. 初始化

将 TMS32010 复位而处于解除状态时，除状态寄存器的中断位处于中断禁止外，寄存器的内容处于不定状态，因此必需重新设定各寄存器的值。

2．设定常数

由于数据存储器是 RAM，必需将必要的数据从程序 ROM 传送到数据 RAM 中去。

3．程序本体

在初始化和设定常数之后，开始进行实际处理的程序部分。

二、初始化程序

初始化程序如 LIST1 所示。TMS32010 复位解除后，从 0 地址开始进行处理。一旦进入中断，则从 2 地址开始处理。因此，在初始化程序中，在 0 地址置无条件分支指令（B）。

LIST1：初始化程序

```
1                0000                AORG    0          解除复位，从0地址开始处理
2       0000     F900                B       INITIAL
3                0100
4
5                0100       AORG     $100
6       0100     7F8B       INITIAL  SOVM                设定溢出方式
7       0101     F500                BV      OVRST      溢出标志复位
8                0103
9       0103     6880       OVRST    LARP    0          选择 AR0
10      0104     6E00                LDPK    0          选择数据存储器的0页
```

指令 SOVM 设定 DSP 的溢出方式。在溢出方式下，累加器溢出后被置为正数 $7FFFFFFF，负数为 $800000000。

指令 BV 用于将溢出标志复位。

LARP 将辅助寄存器指针（ARP）置 0，确定 AR0 为辅助寄存器。

LDPK 置数据页面指针（DP）为 0，直接寻址时在 0 页数据存储器进行存取。

三、循环程序结构的控制

带循环结构的程序如 LIST2 所示。这个程序是从 I/O 的 0 通道读入 8 个数据，并依次存储在从 $10 开始的数据存储器内。

LIST2：使用 BANZ 指令的循环程序控制

```
1                0010               DORG    $10
2       0010                block1  BSS     8          预约8字数据存储区
3                0100               AORG    $100
4       0100     7010               LARK    AR0, block1    AR0←存储区首地址
5       0101     7107               LARK    AR1, 7         计数器置位
6       0102     6880       LOOP    LARP    0
7       0103     40A1               IN      *+, PA0, AR1   由I/O输入
8       0104     F400               BANZ    LOOP           计数器监视
9                0102
```

伪指令 BSS 预约数据存储器的存储区，辅助寄存器 AR0 为指示数据存储目的地址指针，辅助寄存器 AR1 为计算循环次数的减 1 计数器。

BANZ 为辅助寄存器非零转移指令，其操作如图 2－16 所示。首先判别当前辅助寄存器指针指向的辅助寄存器（AR）的值是否为零，不为零时，辅助寄存器减 1，程序转移到由操作数指定的地址；当值为零时，则依次执行下一个地址的指令。这样，使用 BANZ 指令控制有循环结构的程序时，作为减 1 计数器的辅助寄存器的值为循环次数减 1。例如，本程序例中循环次数为 8，因此在第 5 行中 AR1 内的值为 7。

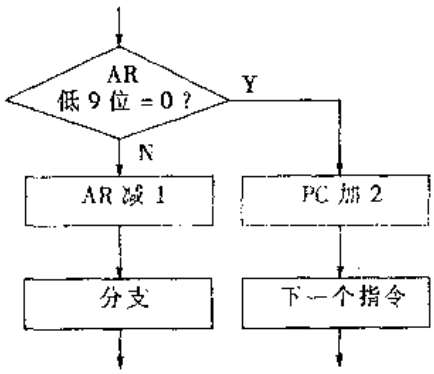


图 2－16　BANZ 的操作功能

四、设定常数的程序

LIST3 为将程序存储器 $20～$24 地址内的常数传送到数据存储器 $10～$14 地址内的程序例。程序存储器内的常数为 Q 格式。

该程序的要点是 16 行 TBLR 指令的用法。TBLR 指令将程序存储器的数据传送到数据存储器。该指令使用累加器为指示程序存储地址的指针。第 16 行中的 AR0 指定数据存储器的地址。

LIST3：设定程序存储器分配器的常数和传送

```
1                0010    0       DORG    $10
2       0010             blk1    BSS     5       预约 5 字存储区
3       0015             ONE     BSS     1       常数 1
4                0020            AORG    $205
5       0020     607D    const   DATA    0．7538Q15      程序存储器上的常数
6       0021     58EC            DATA    0．6947Q15
7       0022     938F            DATA    －0．8472Q15
8       0023     0ADE            DATA    0．0849Q15
```

```
9   0024   CD1B          DATA   -0.3976Q15
10
11          0110          AORG   $110
12  0110   7010          LARK   AR0, blk1     AR0←数据存储器首地址
13  0111   7104          LARK   AR1, 4        计数器置位
14  0112   7E20          LACK   const         Acc←程序存储器首地址
15  0113   6880   LOOP   LARP   0
16  0114   67A1          TBLR   *+, AR1
17  0115   0015          ADD    ONE
18  0116   F400          BANZ   LOOP          监视计数器的次数
19          0113
```

每传送完一个数据，则指向程序存储器和数据存储器的指针均加 1，做好下一个数据传送的准备。

本程序例中 AR0 为数据存储器指针，其自动增 1 表现为操作数自动增 1；而累加器虽为程序存储器的指针，但因无增 1 指令，其增 1 操作由第 17 行的相加指令来完成。因此，在程序存储器的 $15 地址（标号为 ONE），在程序执行前设定了常数 1。

五、移位操作程序

TMS32010 有两种移位器。一种是将数据存储器的数据进行移位后传送到 ALU 的桶形移位器；一种是将累加器的内容移位后送到数据存储器的并行移位器。这两种移位器均只能进行数据的左移位操作。

这两种移位器和一般 CPU 移位的差别在于，它们不是将累加器内容进行移位的移位器，因而在 TMS32010 中无移位操作指令。

桶形移位器可左移 0～15 位，用于 LAC、ADD、SUB 等指令；并行移位器与指令 SACH 一起使用，可左移 0 位、1 位或 4 位。使用桶形移位器使数据进行算术右移的程序如 LIST4 所示。

LIST4：使用定标移位器移位操作

```
1           0010          DORG   $10
2   0010                  data1  BSS    1
3   0011                  data2  BSS    1
4                         *
5   0000   2A10          LAC    data1, 10     左移 10 位后置入 ACC
6   0001   5811          SACH   data2         存储 ACC 的上位 16 位
```

程序中第五行 LAC 指令是将数据 data1 送入累加器，由于 LAC 的第二个操作数为 10，则 data1 的内容左移 10 位后送入累加器。

由于 TMS32010 数据存储器字长为 16 位，累加器长 32 位，因此，累加器的上位 6 位和下 10 位虽是空位，但上位 6 位可进行符号扩展，下位 10 位可填入 0。

第 6 行是将 data2 存入累加器的上位 16 位，结果是将 data1 的内容向右移 4 位后以存放 data2。

六、使用子程序的程序

TMS32010 有两种调用子程序的指令：

CALL：直接调用子程序

CALA：间接调用子程序

从子程序返回主程序的指令是 RET，该指令也可用于由中断处理返回主程序。

子程序例如 LIST5 所示。

LIST5：

```
1                0010                DORG    $ 10
2       0010                 blk1    BSS     5
3       0015                 ONE     BSS     1
4                0020                AORG    $ 20
5       0020     607D        const   DATA    0. 7538Q15
6       0021     58EC                DATA    0. 6947Q15
7       0022     938F                DATA    -0. 8472Q15
8       0023     0ADE                DATA    0. 0849Q15
9       0024     CD1B                DATA    -0. 3976Q15
10
11               0110                AORG    $ 110
12      0110     7010                LARK    AR0, blk1
13      0111     7104                LARK    AR1, 4
14      0112     7E20                LACK    const
15      0113     F800                CALL    MOVE
16               0180
17
18
19                          * * * * * * * * * * * * * * *
20                          *     子     程     序     *
21                          * * * * * * * * * * * * * * *
22               0180                AORG    $ 180
23      0180     6880        MOVE    LARP    0
24      0181     67A1                TBLR    * + , AR1
25      0182     0015                ADD     ONE
26      0183     F400                BANZ    MOVE
27               0180
28      0185     7E8D                RET
```

使用 CALL 和 CALA 指令时，返回地址先压入堆栈，然后执行子程序。产生中断或执行 TBLR、TBLW 指令时，亦使用堆栈。

七、乘法程序

两个小数相乘时应注意小数点的位置。使用固定小数点运算的处理器时，在数字信号处理中其系数的绝对值大多数小于 1 并通常采用 Q15 格式表示之。Q15 格式系指小数点后有 15 位的表示形式。

本例的乘法运算，是求用 Q15 格式表示的数和任意 Q 格式表示的数的乘积，例如和 Q12 表示的小数的乘积，其示意图如图 2－17。16 位长的两个数相乘，其积长 32 位。如将其积的上位 16 位（第 31～16 位）取出并作为结果存储时，则积是用 Q11 格式表示的。但这样以来，在进行乘算时移动小数点位置，在小数处理是非常不方便的；因而希望小数点位置不变。为此，常取第 30 位～15 位作为积存储起来，仍为 Q12 格式，这就需要对小数点位置进行修正，修正的方法是将 32 位数左移 1 位，然后取其上位 16 位作为积。

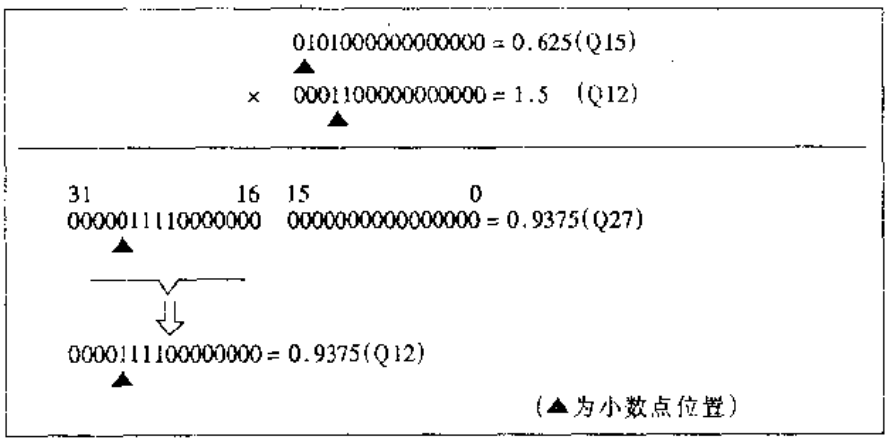


图 2－17 与 Q15 格式的数相乘

LIST6 为任意格式的数和 Q15 格式的数相乘其结果仍用 Q 格式表示，并将其存入数据存储器的程序。这个程序分为两部分，前半部分为不进行舍入，后半部分则进行舍入操作。

LIST6：

```
1                 * * * * *   无舍入时   * * * * *
2                   0020                  DORG      $ 20P
3       0020        x           BSS       1
4       0021        y           BSS       1
5       0022        z           BSS       1
6       0023        ONE         BSS       1
7                   *
8       0200                    AORG      $ 200
9       0200        6A20        LT        x         T←x
10      0201        6D21        MPY       y         T×y
11      0202        7F8E        PAC                 Acc←P
```

```
12      0203    5922        SACH        z, 1    存储结果
13              *
14              * * * * *   有舍入时    * * * * *
15                          *
16      0204    6A20        LT          x
17      0205    6D21        MPY         y
18      0206    7F8E        PAC
19      0207    0E23        ADD         ONE, 14 舍入操作
20      0208    5922        SACH        z, 1
```

程序中由第 12 行（或 20 行）的指令 SACH 进行小数点修正。该指令原是将累加器上位 16 位进行存储，但因第二个操作数是 1，因而成为将累加器内容左移 1 位后，将其上位 16 位进行存储的操作。

舍入操作是将存放在数据存储器内的 16 位的第 1 位右侧加 1。在本程序例中，由于存储的是第 30～15 位，因而在第 14 位进行加 1 操作。在第 19 行中将 1 左移 14 位再与累加器内容相加，从而实现舍入操作。

八、积加运算程序

在数字信号处理中，常进行数字滤波器的运算，其基本运算是积加运算。LIST7 为进行多项式

$$y = a_0b_0 + a_1b_1 + a_2b_2 + a_3b_3 + \cdots$$

运算的程序例。

LIST7：积加程序

```
1               0030        DORG    $ 30
2       0030    a0          BSS     4       a0～a3
3       0034    b0          BSS     4       b0～b3
4       0038    y           BSS     1       卷积结果
5       0039    ONE         BSS     1
6
7               0200        AORG0   $ 200
8       0200    7F89        ZAC             Acc 清 0
9       0201    6A34        LT      b0      T←b0
10      0202    6D30        MPY     a0      T × a0
11      0203    6C35        LTA     b0 + 1  T←b1；Acc←Acc + b0 × a0
12      0204    6D31        MPY     a0 + 1  T × a1
13      0205    6C36        LTA     b0 + 2  T←b2；Acc←Acc + b1 × a1
14      0206    6D32        MPY     a0 + 2  T × a2
15      0207    6C37        LTA     b0 + 3  T←b3；Acc←Acc + b2 × a2
16      0208    6D33        MPY     a0 + 3  T × a3
17      0209    7F8F        APAC            Acc←Acc + b3 × a3
18      020A    0E39        ADD     ONE, 14 舍入
19      020B    5938        SACH    y, 1    存储结果
```

程序中第 11 行、13 行、15 行的指令 LTA 可并行完成两种操作。一是将由操作数指定地址的数据存储器的内容送 T 寄存器，二是将当前 P 寄存器的内容与累加器内容相加。因此，在反复进行积加运算的场合，可同时进行将前次的积加到累加器及将下一个数据送入 T 寄存器的操作。每个积加运算执行时间为 2 个机器周期（400ns）。

九、除法程序

TMS32010 中无进行除算的指令。但使用条件减算指令 SUBC，可编成比较简单的除法程序，如 LIST8 所示。

LIST8：除算程序

```
1                0000                DORG    $0
2       0000                numer    BSS     1        被除数
3       0001                denom    BSS     1        除数
4       0002                quot     BSS     1        商
5                *
6                0200                AORG    $200
7       0200     6500                ZALH    numer
8       0201     700E                LARK    AR0, 14  置计数器
9       0202     6401       LOOP     SUBC    denom    条件减
10      0203     F400                BANZ    LOOP
11               0202
12      0205     5002                SACL    quot
```

除算方法有恢复法和不恢复法。本例程序是采用 SUBC 指令的恢复法除算程序。

程序中第 9 行为 SUBC 指令。除算程序的主体是重复 15 次执行 SUBC 指令。该程序执行时，必需满足下述条件：

·被除数 < 除数

·被除数 ≥ 0；除数 > 0

·除数、被除数的小数点位置相同，即采用相同的 Q 格式的数。此时，商的值为 Q15 格式表示的数。

2.5 TMS32010 外部电路设计

一、TMS32010 电路设计上的注意点

使用 TMS32010 系列处理器时，在电路设计上必需注意解决以下问题。

1．执行 OUT，TBLW 指令后的数据干扰

从 TBLW 指令和 OUT 指令时序可知，当由 MEN 信号控制存储器输出时，在一个最大为 35ns 的间隔内，DSP 和程序存储器之间，会引起总线冲突。为防止出现这种情况，可使用 CLKOUT 信号和 MEN 信号，如图 2－18 那样，对存储器的输出进行控制。

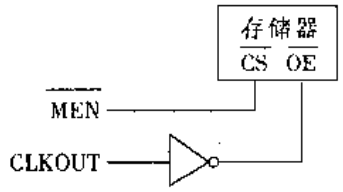


图 2－18　防止数据干扰的电路

TBLW 指令和 OUT 指令时序的非区别性如图 2－19 所示那样，在 $000～$007 地址内，无法区别 TBLW 和 OUT 指令。当然，如果程序存储器全部为 ROM 且不使用 TBLW 指令时，则无需进行地址译码。问题是程序存储器使用 RAM 且使用 TBLW 指令时，如果没有地址译码器，则在全地址空间无法区别这两个指令。

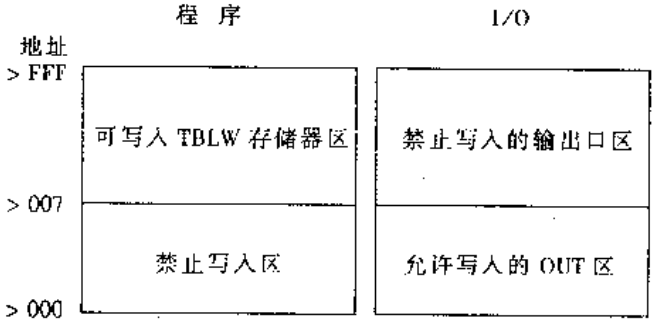


图 2－19　有译码器时地址分配图

为了区别这两条指令，可使用图 2－20 的译码电路。使 $000～$007 地址仅对 OUT 指令有效，而不向存储器写入数据。这样，在设计存储器电路和 I/O 电路时，应综合考虑，要特别注意 TBLW 和 OUT 都使用$\overline{WE}$信号这一特点。

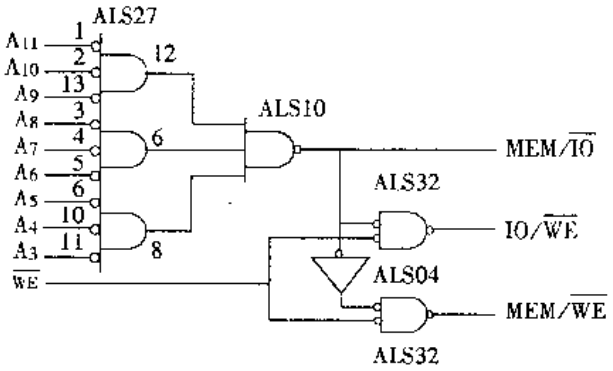


图 2－20　译码电路

2．使用控制信号时的注意点

在设计存储器、I/O 电路时，为防止数据干扰，$\overline{MEN}$信号一定要由 CLKOUT 进行译码。

$\overline{DEN}$和$\overline{WE}$的时序如图 2－21 所示。如图（a）所示那样，$\overline{DEN}$信号的上升沿迟于 CLKOUT 信号的下降沿时，在 CLKOUT 下降沿后呈现不确定地址，因而在数据总线上有可能出现不确定的数据。

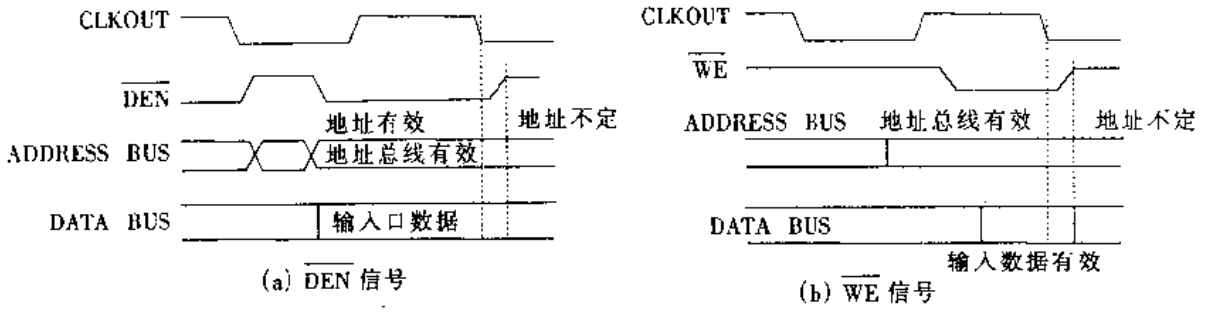


图 2－21　DEN 和 WE 时序

同样的，如图（b）那样，地址总线直到 CLKOUT 下降沿前有效。但当$\overline{WE}$的上升沿较 CLKOUT 的下降沿滞后时，则在 CLKOUT 的下降沿后，会出现将数据写入不确定的地址之内的情况。因而，$\overline{WE}$也必需用 CLKOUT 进行译码。

3．TMS32010 BIO、INT 引脚处理

在使用 TMS32010 时，如图 2－22 那样，$\overline{BIO}$引脚和$\overline{INT}$引脚应接入同步双稳态电路以防止误操作。

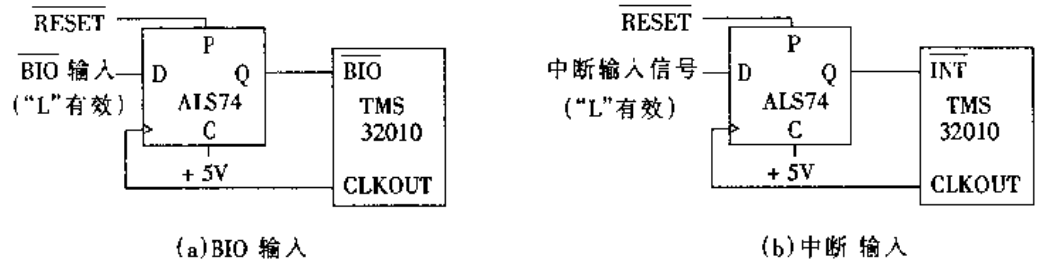


图 2－22　TMS32010 中 BIO、INT 引脚处理

TMS320C10，320C15/E15 是改良型的信号处理单片机，不必使用双稳态电路。

4．X2/CLKIN 引脚的连接

在 TMS320C10 引脚中，许多引脚和 TTL 兼容，X2/CLKIN 引脚为 V1H = 3V。因此，当用 HC 型 CMOS 驱动或用 TTL 驱动时，接一个前置电阻，只要满足 V1H 为 3V 即可。

二、TMS32010 接口设计

1．主时钟周期和指令周期

TMS32010 的主时钟周期和周期时间的关系则如图 2－23 所示。

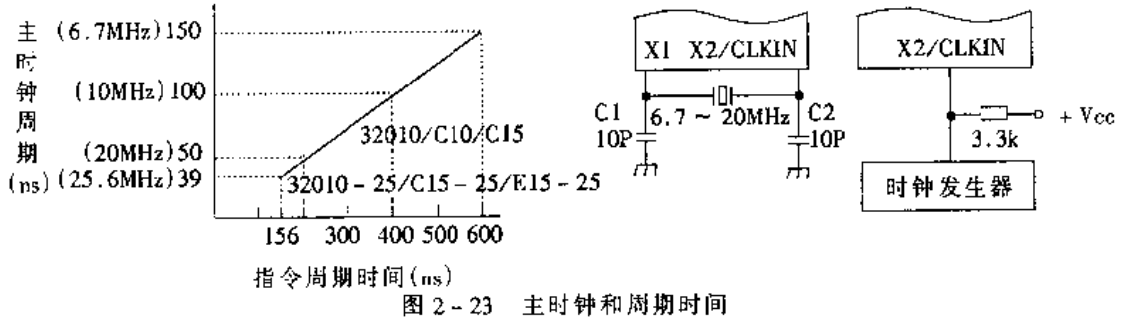


图 2－23　主时钟和周期时间

即主时钟在4分频后作为CLKOUT信号输出到外部，周期时间为主时钟的4倍。主时钟的频率范围为6.7MHz～20MHz（32010/C10/C15/E15），因而指令周期时间在600ns～200ns范围内。

2．存储器、I/O的存取时序

图2－24为存储器的读出时序。从存储器的读出周期来看，有三种存取时间：

（1）t_{ac}从地址有效开始的存取时间。从图2－23中看出，t_{ac}为：

$$t_{ac} = t_c\ (c) - t_{d1} - t_{su}\ (0) = t_c\ (c) - 100 = 100\ (ns)\ (20MHz)$$

为避免总线上数据冲突，多不使用。

（2）tac（CLK）从CLKOUT有效开始的存取时间：

$$t_{ac}\ (CLK) = \frac{1}{2}t_c\ (c) - 20 - 50 = 30\ (ns)$$

这时虽有总线冲突，但由于tac（CLK）=30ns，接口时需用高速存储器。

（3）t_{ac}（$\overline{MEN}$）从$\overline{MEN}$信号有效开始的存取时间：

$$t_{ac}\ (\overline{MEN}) = t_c\ (c) - t_{d2} - t_{su}\ (d) = 85ns$$

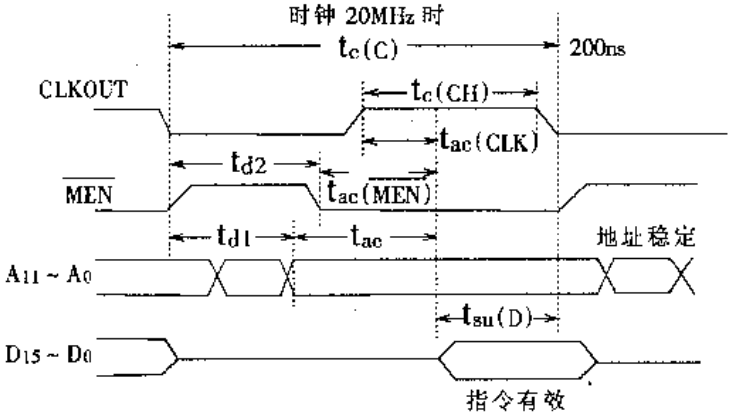


图2－24　存储器读出时序

此时虽可能引起总线冲突，但如果用CLKOUT信号控制输入输出，对数据总线进行离线控制，可适用于比t_{ac}（CLK）速度稍慢的存储器。例如，使用CLKOUT信号进行控制时，用图2－25所示电路。

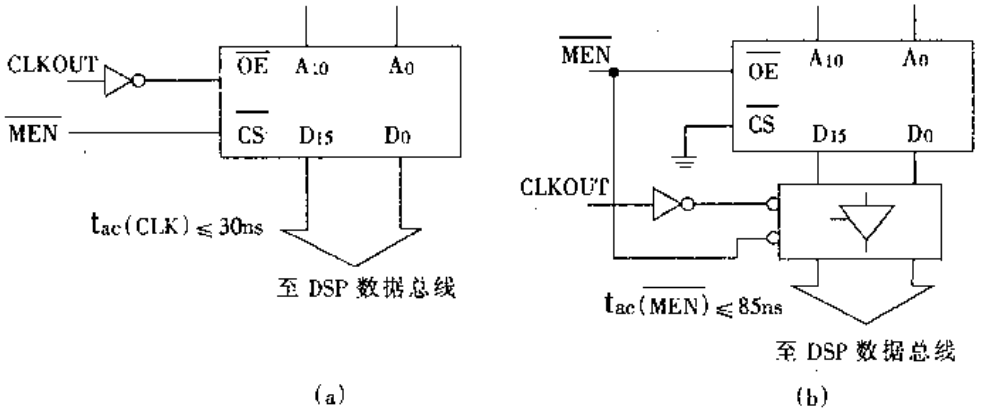
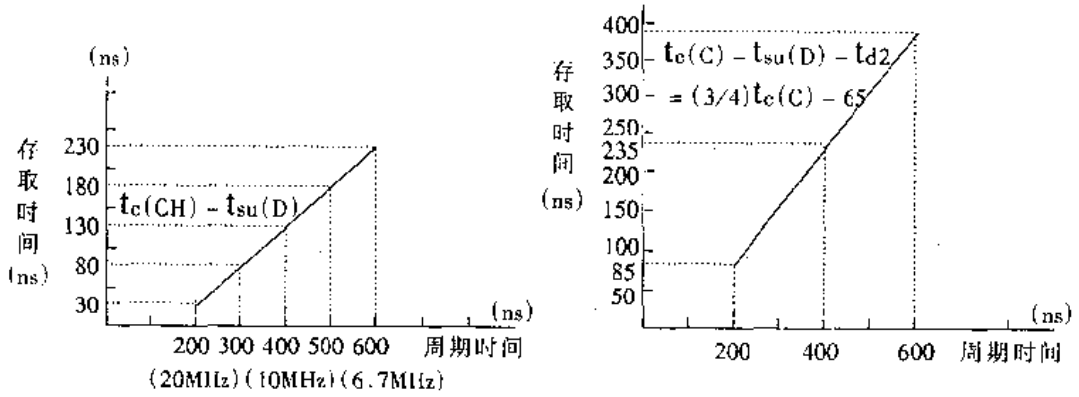


图2－25　由CLKOUT信号控制输入输出

图 2－25（a）为从 CLKOUT 有效开始的存取时间为 30ns 以下时存储器译码电路，图 2－25（b）为从$\overline{MEN}$有效开始的存取时间在 85ns 以下时的存储器译码电路。

上述存取时间 t_{ac}（CLK）和 t_{ac}（$\overline{MEN}$）与周期时间的关系如图 2－26 所示。但这些数值中不包含总线缓冲器和译码器的延迟时间，因而在设计时必须考虑这一点。



(a) CLKOUT 选通时的 t_{ac}(CLK)　　(b)控制信号选通时的 $t_{ac}(\overline{MEN})$

图 2－26 周期时间与存储器存取时间

三、TMS32010 与存储器接口

1．TMS32010 与高速 EPROM 接口

图 2－27 为 TMS32010 与高速 EPROM TMS27C291/29235 接口的实际电路图。

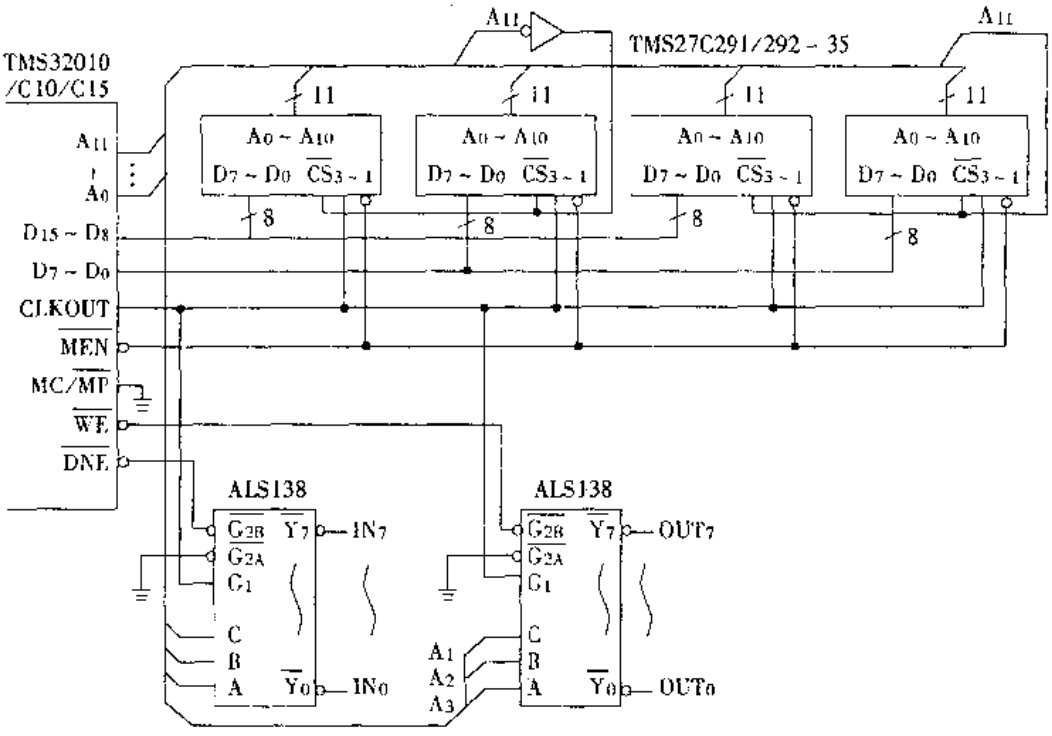


图 2－27　TMS32010 与高速 EPROM 接口

EPROM 从接受片选信号开始的存取时间是 25ns，满足图 2－24 中的存取时间 t_{ac}（CLK）。因此，电路中将 CLKOUT 引脚与 EPROM 的片选引脚直接相接。同时，用最上位地址 A_{11} 进行存储器译码，以构成容量为 2K×8 位的存储器。

本例中全部程序空间均由 ROM 组成，所有地址均不能使用 TBLW 指令。

2．TMS32010 和高速 EPROM 和高速 SRAM 接口（1）

图 2－28 为 TMS32010 与 2K 字 EPROM 和 2K 字高速 SRAM 接口电路图。

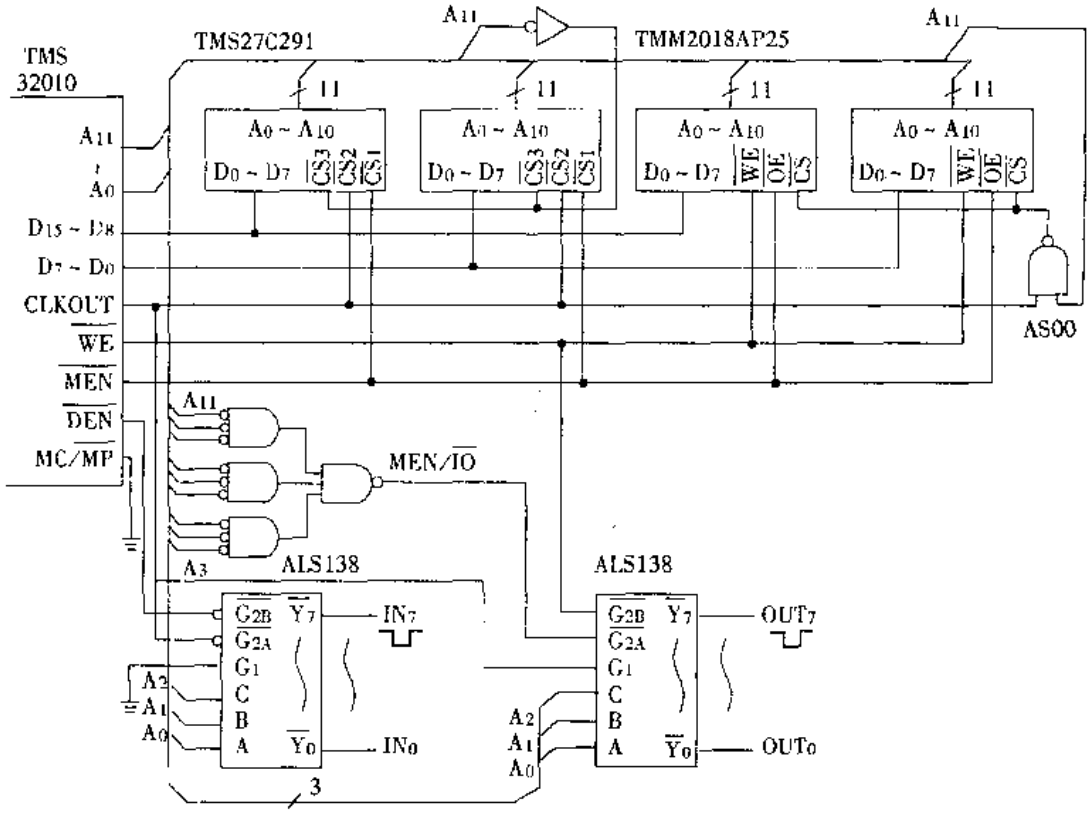


图 2－28　TMS32010 与 EPROM/SRAM 接口

电路中使用了高速 RAM（存取时间 25ns）作为程序存储器，因而附加了译码器电路，译码器电路生成$\overline{MEN}/\overline{IO}$信号，以区别 TBLW 和 OUT 指令。

同样，在 $000～$007H 地址内，硬件电路也不能对 TBLW 和 OUT 进行区别，因而，在这个区域内，不能使用 TBLW 指令。

另外，由于 EPROM 和 SRAM 有相同的存取时间（25ns），小于 t_{ac}（CLK），可以用 CLKOUT 信号直接进行译码。

3．TMS32010 与 EPROM 和高速 SRAM 接口（2）

前两例中介绍的都是使用存取时间为 25ns 的集成电路存储器，图 2－29 所示的是时钟仍为 20MHz，但使用存取时间稍长一些的集成电路存储器时的接口电路。

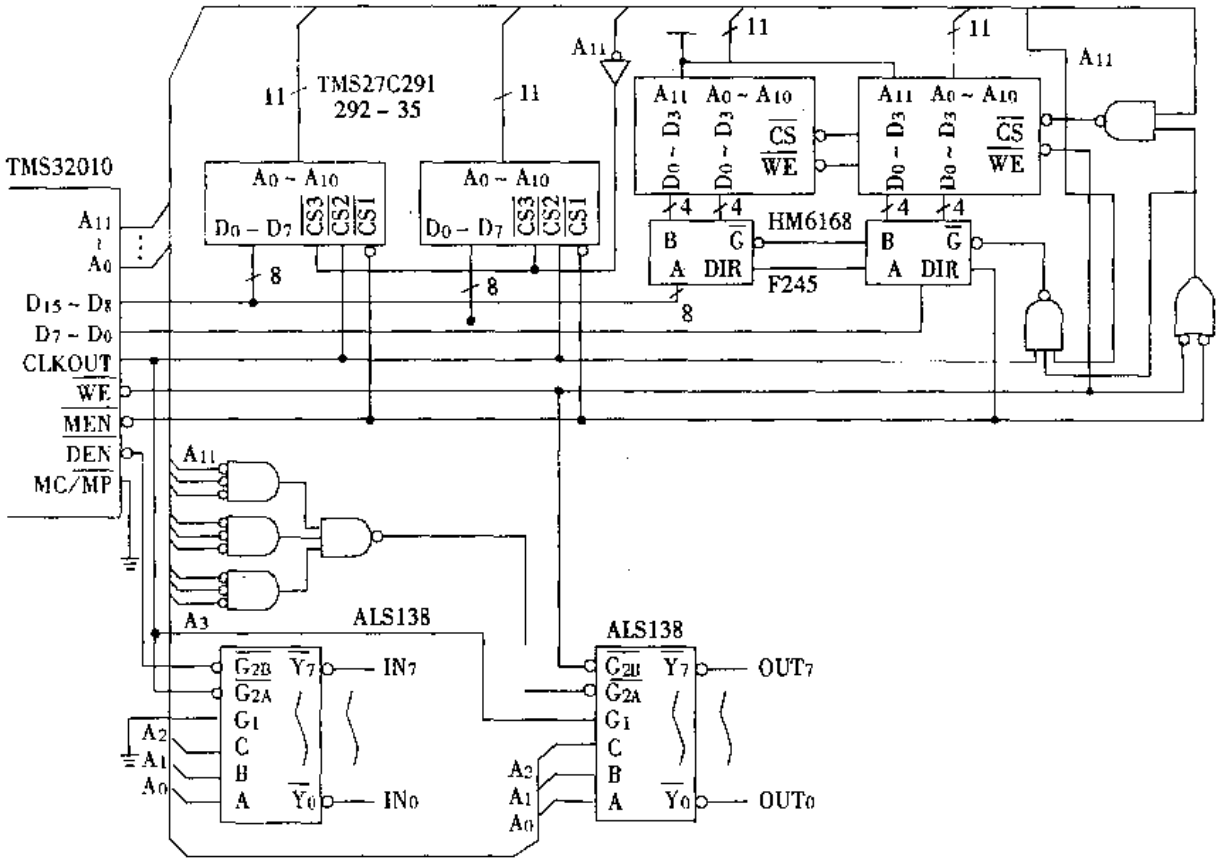


图 2 - 29　TMS32010 与 EPROM/SRAM 接口（2）

正如在存储器存取时序中叙述的那样，在存储器一侧不用 CLKOUT 信号而仅用控制信号进行译码。为避免总线冲突，在 RAM 和 DSP 之间，介入双向数据总线缓冲器并由 CLKOUT 进行控制。

这时，存储器的存取时间变为 t_{ac}（$\overline{MEN}$）（85ns），但尚需从 t_{ac}（$\overline{MEN}$）中减去缓冲器的延迟时间，因而，实际的存储器的存取时间为 t_{ac}（$\overline{MEN}$）－15ns = 70ns，因而可以使用存取时间为 70ns 左右的存储器。

4．TMS32010 和通用 EPROM 的接口

在允许降低执行速度的场合，可以降低时钟频率而和通用存储器接口。图 2 - 30 为使用高速型（存取时间在 150ns 左右）存储器 2764 接口的电路图。

设从地址有效开始的存取时间为 150ns。为避免总线冲突，电路中附加外部总线缓冲器，并由控制信号$\overline{MEN}$和 CLKOUT 信号进行控制。总线缓冲器和反相器的延迟时间约为 20ns，因而需时钟周期 t_c（c）为 270ns，即主时钟频为 14．8MHz。

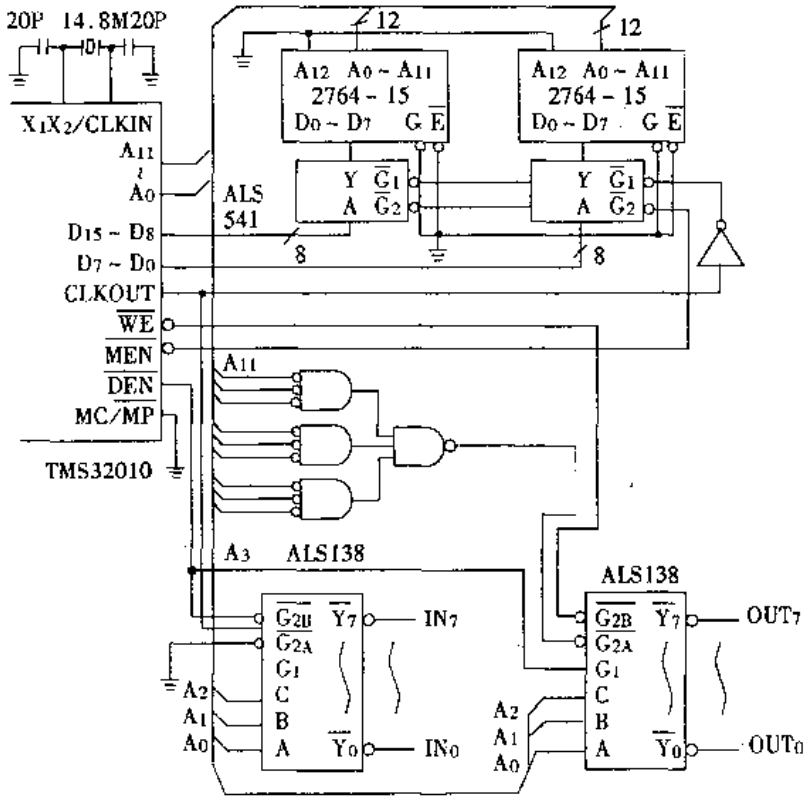


图 2 - 30 TMS32010 与 2764 接口

四、TMS32010 和 I/O 接口的实际电路

1．I/O 存取时间

执行 IN 指令时的存取时间从 $\overline{DEN}$ 脉冲开始，为 t_c（c）$-t_{d4}-t_{su}$（D），在主时钟为 20MHz 时为 85ns。

2．I/O 接口

I/O 接口和存储器接口时一样，当使用微处理器的外围集成电路时，必须降低时钟频率。在主时钟为 20MHz 时，可使用 ALS/LS 型 TTL，或 HC/HCT 型标准逻辑器件作为外部设备。如果考虑到扇出等因素，与数据总线有关设备，在速度允许的情况下也可以使 HC 型逻辑电路。但是，译码电路中总是有延迟的，因而要使用 ALS、AS 或 F 型的器件。

3．输入通道和输出通道

输入通道和输出通道的电路例如图 2 - 31。输入和输出通道都使用 HCT574 型锁存器。如果在 CLK 引脚加上正向脉冲时，数据置位，直到下一个上升沿数据被锁存。

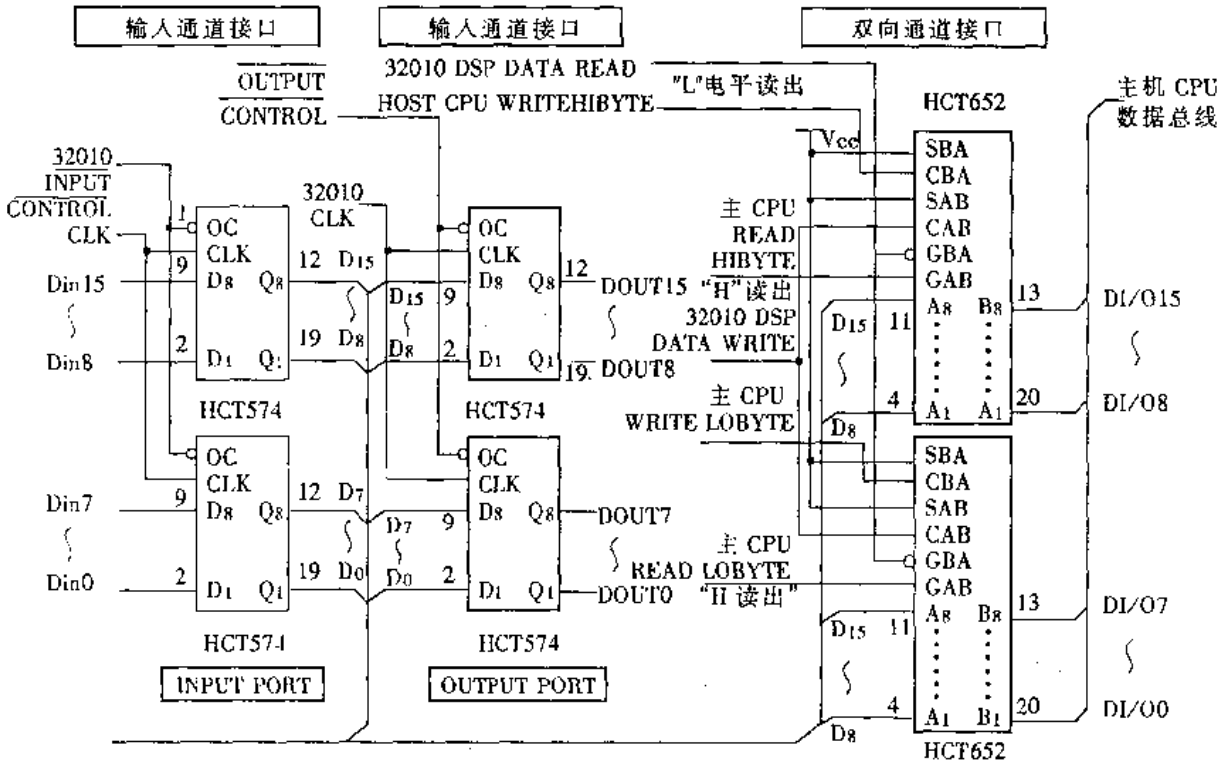


图 2－31　输入通道与输出通道

数据读出时在 OC 引脚置“L”电平。

五、TMS32010 与 A/D 转换器接口

图 2－32 为 TMS32010 与 AD7572 接口的线路。

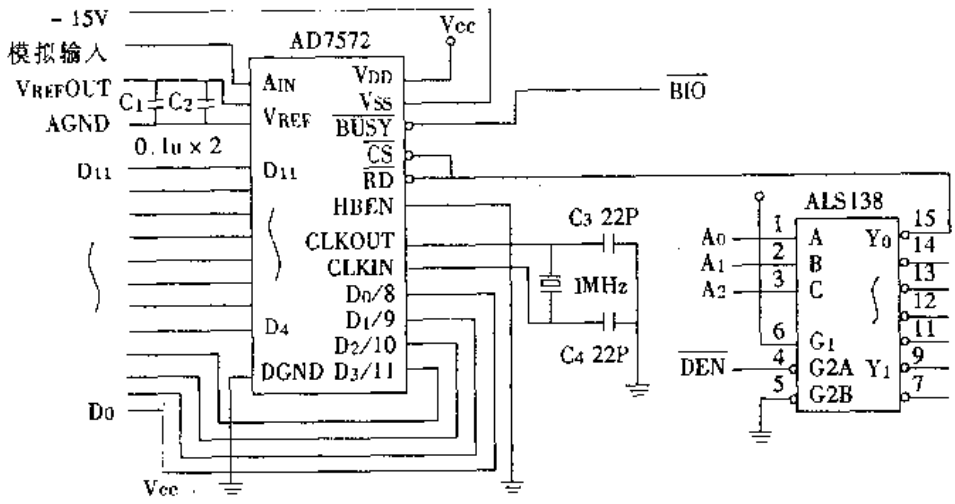


图 2－32　TMS32010 与高速 A/D 接口

图中，DSP 的数据总线直接和 AD7572 相连。AD7572 的使用方式，称为 ROM 并行读出方式。即由一个 READ 操作使转换开始，前次转换结果（12 位数据）出现在数据输出引脚 D11 ~ D0。该数据如不需要可以舍去；然后由第二个 READ 操作读取新的数据（D11 ~ D0）并开始下一个转换。

在两次 READ 操作之间，最少有 AD7572 转换时间的延迟。这时的时序如图 2 - 33 所示。$\overline{DEN}$信号和地址译码之后，生成$\overline{RD}$信号。即用两次 READ 操作得到新的变换数据。

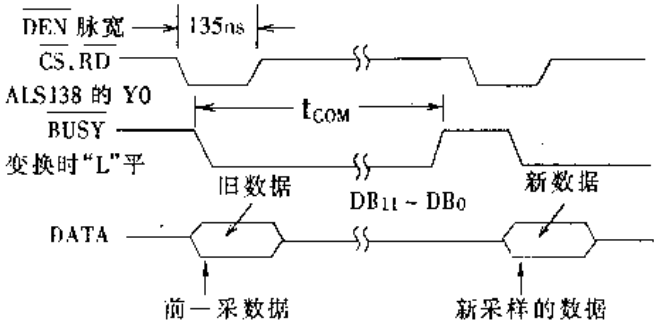


图 2 - 33　TMS32010 与 A/D 接口时序

六、TMS32010 与串行型 A/D 转换器接口

一般的 TMS32010 无串行口，因而在与具有串行型输入输出的 A/D 转换器接口时，需介入移位寄存器，DSP 对进入移位寄存器的数据进行存取。

图 2 - 34 为 8 位 A/D 转换器 TLC549 与 TMS32010 接口的线路图。

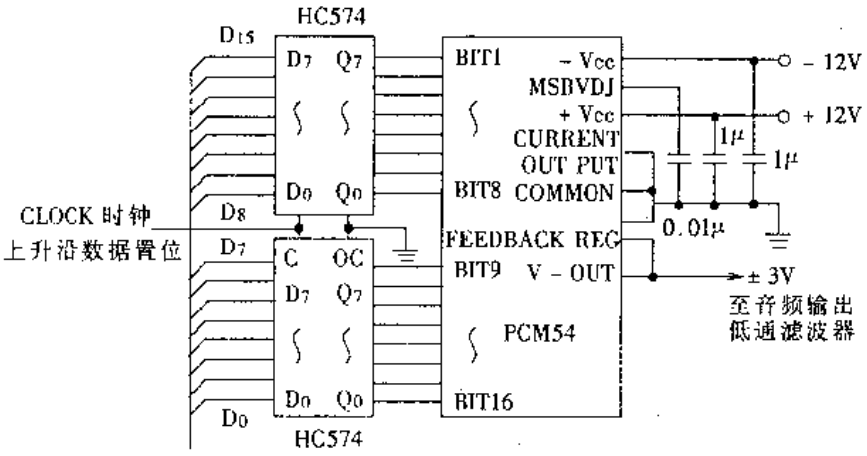


图 2 - 34　TMS32010 与 D/A 转换器接口

七、TMS32010 与 D/A 转换器接口

图 2 - 35 为 D/A 转换器通过锁存器与 DSP 的接口电路。

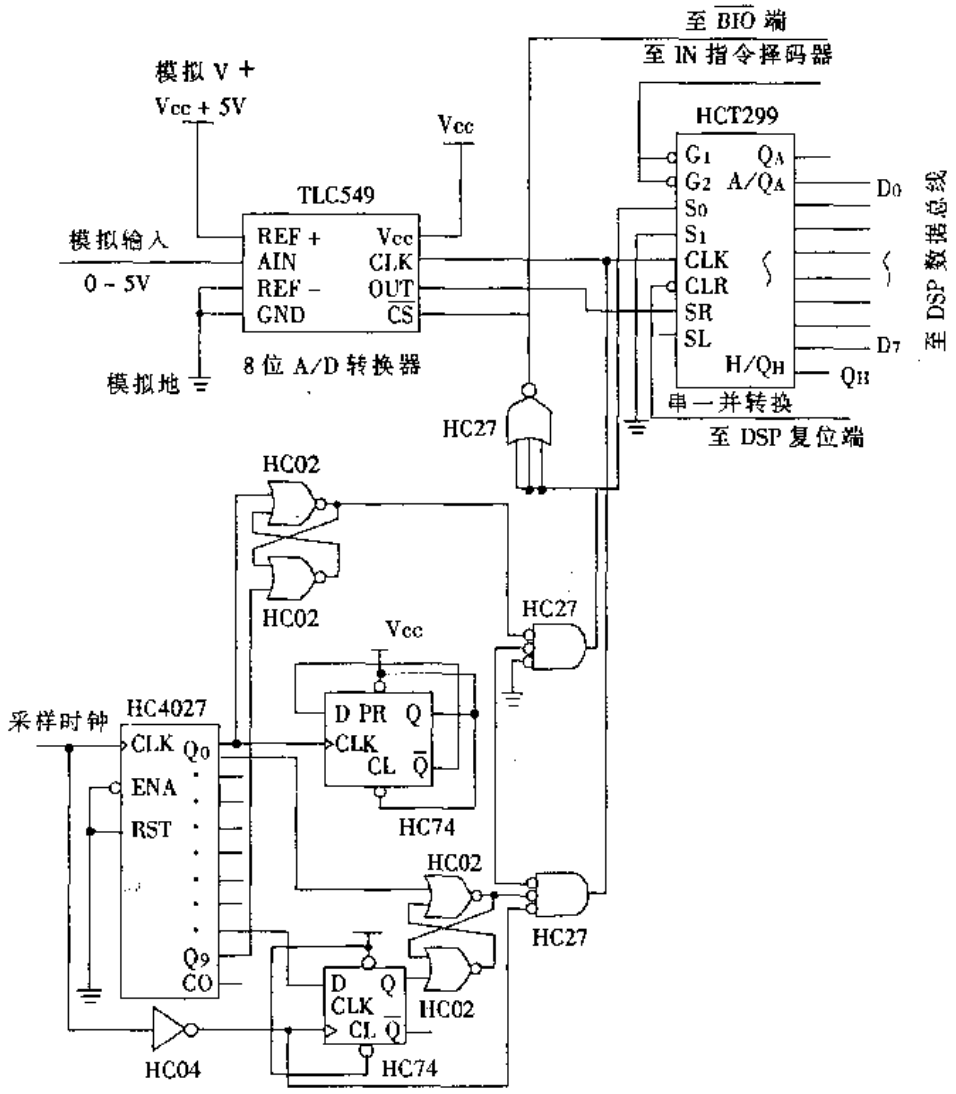


图 2 - 35　TMSA32010 与 D/A 转换器接口

八、TMS32010 与扩展存储器接口

TMS32010 可外接程序存储器和 I/O，但不能外接数据存储器。但是，在 I/O 空间设置锁存器，并以此为地址使用数据输入输出指令而对数据进行存取，虽然需要一定的时间，但终究可作为数据存储器。

图 2 - 36 为扩展存储器的接口电路。

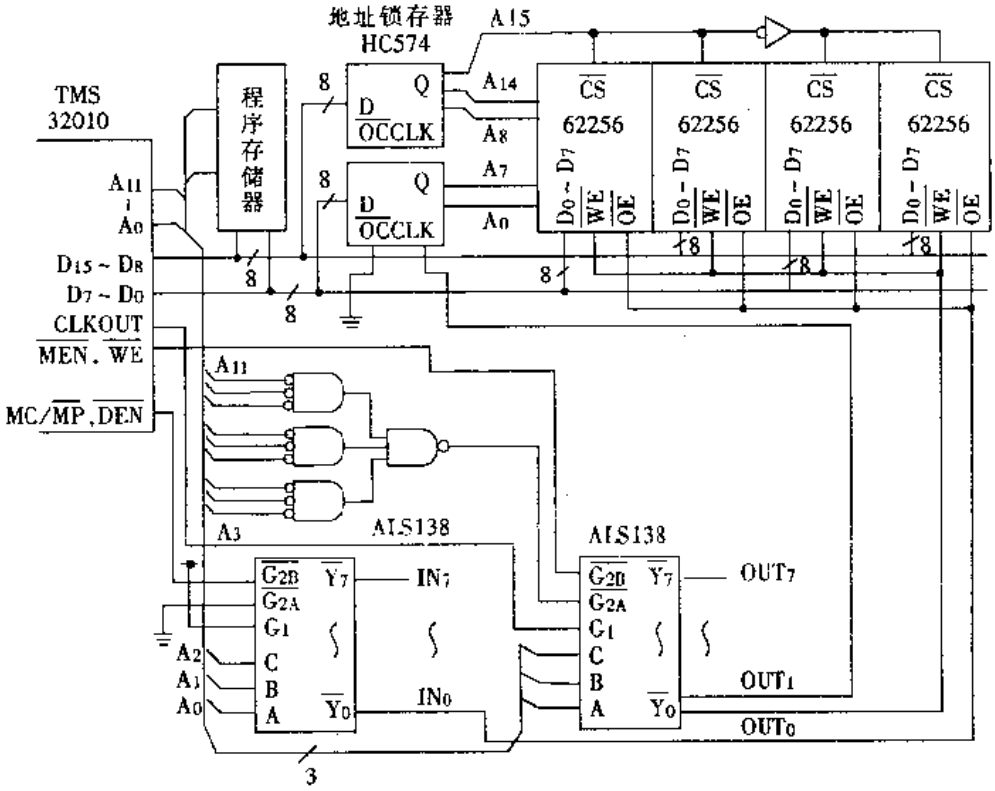


图 2 – 36　TMS32010 与扩展存储接口（64K 字）

第三章　TMS32020 系列处理器

3.1　TMS32030 概述

TMS32020 系列是美国 TI 公司开发的第二代数字信号处理器。它主要有两个产品：TMS32020 和 TMS320C25。

一、TMS32020 特性

TMS32020 是美国 TI 公司 1985 年开发的产品。TMS32020 在保持 TMS32010 优点的基础上又增加和扩充了许多功能。

TMS32020 芯片采用了特殊的系统结构(哈佛结构)，程序存储器和数据存储器分离，这使得指令与数据存取可以同时进行，从而提高了执行速度。32020 的时钟频率为 20MHz，指令执行速度达 5MIPS。它的指令集是针对信号处理工作的。寻址方式有直接和间接寻址方式。对不同的工作要求，可以构成不同配置的各种系统，如最小配置的独立系统、局部存储器的主从系统、以及全局存储器的多处理器系统等。芯片的功能适用于数字信号处理、图形图像处理、语音处理、测量与控制以及通信等领域。

TMS32020 的主要特性有：

- 可执行片内 RAM(B0 块)中的程序；
- 片内数据 RAM544 字，其中 256 字可任意地配置为程序或数据存储器；
- 外部存储器扩展空间 128K 字(64K 字程序存储器和 64K 字数据存储器)；
- 单周期乘法/累加指令；
- 指令系统支持浮点操作；
- 具有多处理器和全局数据存储器接口；
- 同步多处理器配置的能力；
- 可实现数据　/程序管理的块传送；
- 有与慢速外存储器及外围设备通信的等待状态；
- 具有片内定时器和串行口；
- 5 个辅助寄存器带有他们自己的算术单元操作指令；
- 有 3 个外部可屏蔽用户中断；
- 68 脚 PGA 阵列封装；
- 具有 DMA 操作功能；
- 当允许流水线操作时，重复指令能充分利用程序空间；
- 具有 16 个输入和 16 个输出通道；
- 采用 2.4μm NMOS 工艺。
- 200ns 的指令周期。

二、TMS32020 引脚及信号

1. TMS32020 引脚及信号

TMS32020 是 68 引脚 PGA 阵列封装。图 3－1 为 TMS32020 微处理器的引脚排列，表 3－1 是引脚/信号位置表。表 3－2 列出了 TMS32020 的信号状态及功能。

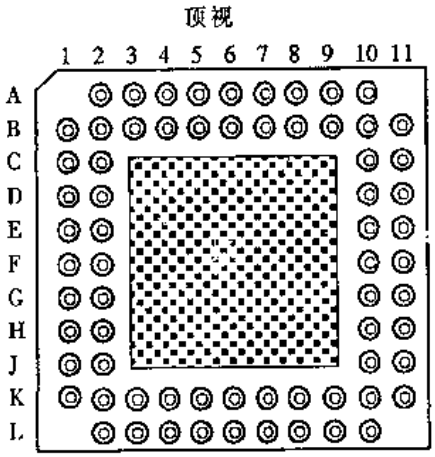


图 3－1　TMS32020 引脚排列

表 3－1　引脚信号

引脚	信号	引脚	信号	引脚	信号	引脚	信号	引脚	信号
A2	D8	B6	D15	E1	D2	H10	$\overline{STRB}$	K9	A14
A3	D10	B7	$\overline{BIO}$	E2	D1	H12	R/$\overline{W}$	K10	$\overline{DS}$
A4	D12	B8	READY	E10	$\overline{HOLDA}$	J1	DR	K11	Vss
A5	D15	B9	CLKR	E11	DX	J2	FSR	L2	Vss
A6	Vcc	B10	Vcc	F1	D0	J10	$\overline{PS}$	L3	A2
A7	$\overline{HOLD}$	B11	$\overline{IACK}$	F2	$\overline{SYNC}$	J11	$\overline{IS}$	L4	A4
A8	$\overline{RS}$	C1	D6	F10	FSX	K1	A0	L5	A6
A9	CLKX	C2	D5	F11	X2/CLKIN	K2	A1	L6	Vcc
A10	Vcc	C10	$\overline{MSC}$	G1	$\overline{INT0}$	K3	A3	L7	A9
B1	Vss	C11	CLKOUT1	G2	$\overline{INT1}$	K4	A5	L8	A11
B2	D7	D1	D4	G10	X1	K5	A7	L9	A13
B3	D9	D2	D3	G11	$\overline{BR}$	K6	A8	L10	A15
B4	D11	D10	CLKOUT2	H1	$\overline{INT2}$	K7	A10		
B5	D13	D1	XF	H2	Vcc	K8	A12		

表 3-2 TMS32020 引脚信号说明

	信号	I/O/Z	说明
地址/数据总线	A15(MSB) A0(LSB)	O/Z	并行地址总线。多路复用寻址外部数据/程序存储器或 I/O。保持方式时呈高阻抗状态。
	D15(MSB) D0(LSB)	I/O/Z	并行数据总线。多路复用,在 TMS32020 和外部数据/程序存储器或 I/O 器件之间传送数据。当不输出或者当$\overline{RS}$或$\overline{HOLD}$信号有效时,呈高阻抗状态。
端口控制信号	$\overline{DS}$ $\overline{PS}$ IS	O/Z	数据、程序和 I/O 空间选择信号。通常呈高电平状态。保持方式呈高阻抗状态。
	READY	I	数据准备好输入。若外部器件未准备好,则 TMS32020 等待一个周期并再次检测 READY,在$\overline{BR}$信号之后,READY 向一个外部器件表示总线许可
	R/$\overline{W}$	O/Z	读/写信号。当与外部器件通讯时指定传送方向。通常在读方式(高电平),只有在执行写操作时才呈低电平。在保持方式时呈高阻抗状态。
	$\overline{STRB}$	O/Z	选通信号。通常呈现高电平。在保持方式时处于高阻抗状态。
多值处理信号	$\overline{BR}$	O	总线请求信号。在 TMS32020 要求存取一个外部全局数据存储器空间时建立。
	$\overline{HOLD}$	I	保持输入。当有效时 TMS32020 进入闲置方式并数据、地址和控制总线呈高阻抗状态。
	$\overline{HOLDA}$	O	保持响应信号。表示 TMS32020 已经进入保持方式。
	$\overline{SYNC}$	I	同步输入。允许两个或多个 TMS32020 的时钟同步。低电平有效,必须在 CLKIN 上升沿时建立。
中断和其他信号	$\overline{BIO}$	I	转移控制输入。由 IBOZ 指令查询。若为低,执行转移操作。该信号必须在 BIOZ 指令读取期间保持有效
	$\overline{IACK}$	O	中断响应信号。仅当 CLKOUT1 为低时,其输出有效。
	$\overline{INT2}$ $\overline{INT1}$ $\overline{INT0}$	I	外部的用户中断输入
	$\overline{MSC}$	O	微状态完成信号。当刚完成一个存储器操作时变为低电平,且仅当 CLKOUT1 呈低电平期间有效。
	$\overline{RS}$	I	复位输入。
	$\overline{XF}$	O	外部标志输出(由软件转接的锁存信号)

电源/振荡器信号	CLKOUT1	O	主时钟输出信号
	CLKOUT2	O	第二个时钟输出信号
	V_{cc}	I	5 个 5 伏电源引脚，在外面连接在一起
	V_{ss}	I	3 个接地引脚，在外面连接在一起
	X1	O	内部振荡器到晶体的输出引脚，如不用晶体，引脚空置
	X2/CLKIN	I	从晶体到内部振荡器的输入引脚。如果不用晶体，在该引脚向器件输入时钟
串行口信号	CLKR	I	接收时钟输入
	CLKX	I	发送时钟输入。为记录从 DXR 到 DX 引脚的为数据的外部时钟信号。当使用串行口时，必须出现
	DR	I	数据接收输入端。串行数据经 DR 引脚接收到 DRR 中
	DX	O/Z	数据发送输出端。串行数据从 DXR 经 DX 引脚发出。不发送时呈高阻状态
	FSR	I	接收帧同步脉冲输入端。
	FSX	I/O	发送帧同步脉冲输入/输出端。

(I/O/Z，表示输入/输出/高阻状态)

3.2 TMS32020 硬件结构

TMS32020 芯片的功能框图如图 3－2 所示。

TMS32020 的结构是围绕着两条主要的总线建立的。程序总线传送来自程序存储器的指令代码和立即操作数。数据总线将中央算术逻辑单元（CALU)和辅助寄存器堆等部件互连到数据 RAM。程序总线和数据总线一起可在单个周期内将来自两个片内 RAM 块的数据传送给乘法器进行乘法/累加操作。

TMS32020 具有高度并行性，当数据正由 CALU 执行的同时，还可以在辅助寄存器算术单元(ARUA)中实现算术运算。这种并行性使算术、逻辑和位操作运算均可在单个机器周期内完成。

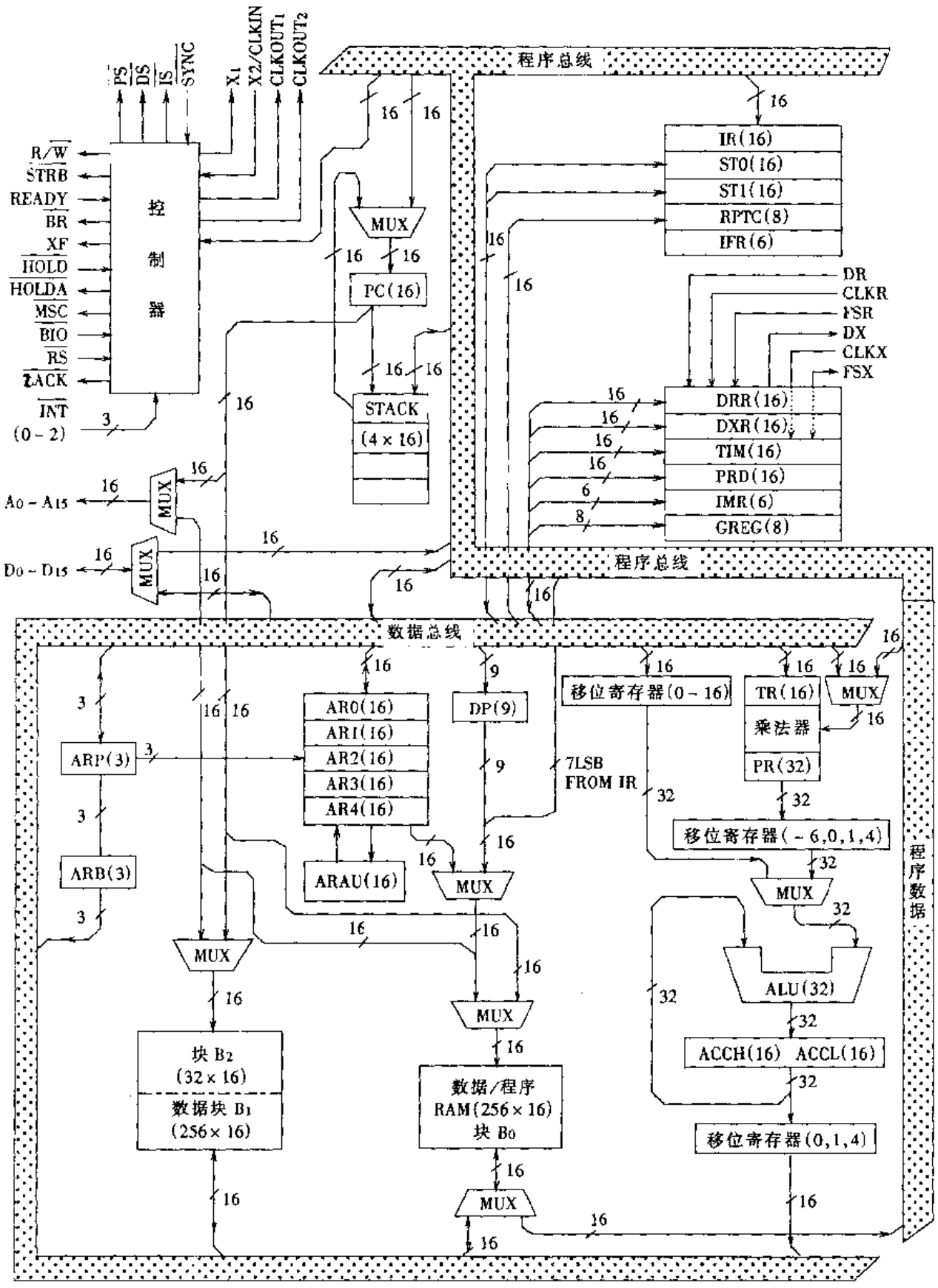


图 3-2　TMS32020 功能框图

图 3-2 中各符号说明如下：

ACCH	累加器高 16 位；	ACCL	累加器低 16 位；
ARAU	辅助寄存器算术单元；	ARB	辅助寄存器指针缓冲器；
ARP	辅助寄存器指针；	DP	数据存储器页指针；
DRR	串行口数据接收寄存器；	DXR	串行口数据发送寄存器；
GREG	全局存储器定位寄存器；	IFR	中断标志寄存器；
IMR	中断屏蔽寄存器；	RPTC	重复指令计数器；
IR	指令寄存器；	PR	P 寄存器；
PRD	关于定时器的周期寄存器；	TIM	定时器；
TR	T 暂时寄存器；	ST0、ST1	状态寄存器

一、内部硬件及功能

TMS32020 用内部硬件实现其他处理器通常用软件或微码完成的功能。例如，它包含用于 16 × 16 位单周期乘法、数据移位和地址操作的硬件。TMS32020 内部硬件的概要如表 3 - 3。

表 3 - 3　内部硬件功能

	单　元	符　号	功　能
处理部件	算术逻辑单元	ALU	32 位补码算术逻辑单元，有两个 32 位输入口，一个 32 位输出口馈给累加器
	中央算术逻辑单元	CALU	ALU、乘法器、累加器和定标器的集合
	乘法器	MUL	16 × 16 位并行乘法器
	周期寄存器	PRD	用于重新装入定时器的存储映射 16 位寄存器
	程序计数器	PC	一个 16 位程序计数器，用于寻址程序存储器，当用块传送和乘法/累加指令时，还用于寻址数据存储
	随机存取存储器(数据或程序)	RAM(B0)	256 字 × 16 位的 RAM 块，可配置为数据或程序存储器
	随机存取存储器(仅数据)	RAM(B1)	数据 RAM 块
	随机存取存储器(仅数据)	RAM(B2)	数据 RAM 块
	辅助寄存器算术单元	ARAU	16 位无符号算术单元，用来对辅助寄存器数据进行运算
	重复计数器	RPTC	8 位计数器，用于控制单一指令的重复执行
	移位器	SFL,SFR	移位器 SFL(左移)和 SFR(右移)置于 ALU 输入端，累加器输出端和乘积寄存器输出端。累加器内还有一个就地移位器
	定时器	TIM	存储器映射 16 位定时器(计数器)，用于定时控制
	累加器	ACCH (31 ~ 16) ACCL(15 ~ 0)	32 位累加器分为两半，ACCH 和 ACCL，用于存储 ALU 的输出

寄存器	辅助寄存器指针	ARP(2~0)	一个3位寄存器,用于选择五个辅助寄存器中的一个
	辅助寄存器指针缓冲器	ARB(2~0)	用于缓冲ARP的3位寄存器。除了在执行LST指令周期外,每次装入ARP,老的值就被写入ARB。当用LST1指令装入ARB时,相同的值也被复制到ARP中
	数据存储器页指针	DP(8~0)	9位寄存器,指向当前页地址。
	全局存储器定位寄存器	GREG(7~0)	存储器映射的8位寄存器,用来确定全局存储器空间的容量
	指令寄存器	IR(15~0)	16位寄存器,用来存储目前正执行的指令
	中断标志寄存器	IFR(5~0)	6位标志寄存器,用于锁存以低有效的外部用户中断$\overline{\text{INT}}$(2~0)及内部中断XINT/RINT和TINT。IFR不能通过软件存取
	中断屏蔽寄存器	IMR(5~0)	存储器映射的6位寄存器,用于屏蔽中断
	乘积寄存器	PR(31~0)	32位寄存器,容纳乘法器的乘积
	堆栈	STACK	4×16硬堆栈,在中断或调用期间用来存储PC.ACCL和数据存储器值也可以压入或弹出堆栈
	串行口数据接收寄存器	DRR(15~0)	存储器映射的16位串行口数据接收寄存器。字节方式时只使用低8位
	串行口发送寄存器	DXR(15~0)	存储器映射的16位串行口数据发送寄存器。在字节方式时只用低8位
	状态寄存器	ST0,ST1	两个16位状态寄存器,包含状态位和控制位
	暂存寄存器	TR(15~0)	16位寄存器,保存乘法器的一个操作数或关于定标移位器的移位代码
总线	辅助寄存器堆总线	AFB(15~0)	16位总线,传送由ARP指定的AR来的数据
	数据总线	DB(15~0)	流通数据的16位总线
	数据存储器地址总线	DAB(15~0)	16位总线,传送数据存储器地址
	直接数据存储器地址总线	DRB(15~0)	16位总线,为数据存储器传送"直接"地址,它是DP寄存器与指令字低7位的级联
	程序总线	P(15~0)	流通指令的16位总线
	程序存储器地址总线	PAB(15~0)	16位总线,传送程序储器地址

二、存储器、寄存器、堆栈

TMS32020 内部配置有 544 字的片内数据存储器 RAM，并分成三个独立的块（B0、B1 和 B2），如图 3－3 所示。块 B0(256 字)可以用作数据或程序存储器，块 B1 和 B2 只用做数据存储器。数据存储器容量允许 TMS32020 处理 512 字的数组（如果片内 RAM 用作程序存储器，则是 256 字的数组），留下的 32 个单元作中间存储。当块 B0 用作程序存储器时，可用 RPTK 和 BLKP 指令将程序从外部程序存储器装入 B0 块。

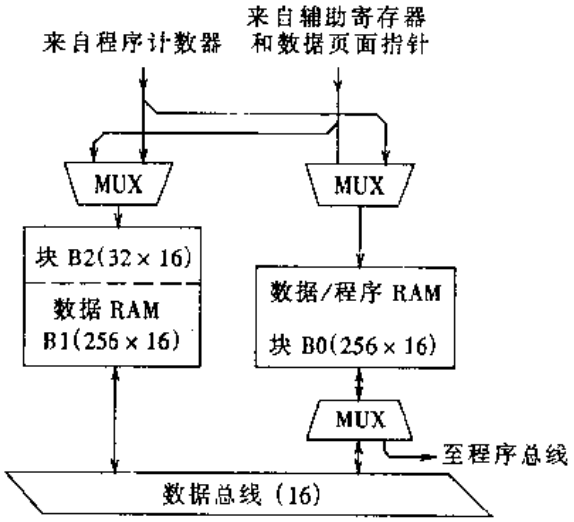


图 3－3　片内数据存储器

TMS32020 提供一个包含 5 个辅助寄存器(AR0～AR4)的寄存器堆。辅助寄存器可用于数据寄存器的间接寻址，或作为数据暂存器。间接辅助寄存器寻址允许将指令操作数据的数据存储器地址放在某个辅助寄存器。这些寄存器由一个 3 位辅助寄存器指针(ARP)指定。该指针可装入 0,1,2,3 或 4，分别指定 AR0～AR4。可以任意地将数据存储器中的数或指令中定义的立即操作数装入辅助寄存器和 ARP。这些寄存器的内容也可以存到数据存储器中。

辅助寄存器堆(AR0－AR4)连接到 ARAU(辅助寄存器算术单元)，如图 3－4 所示。

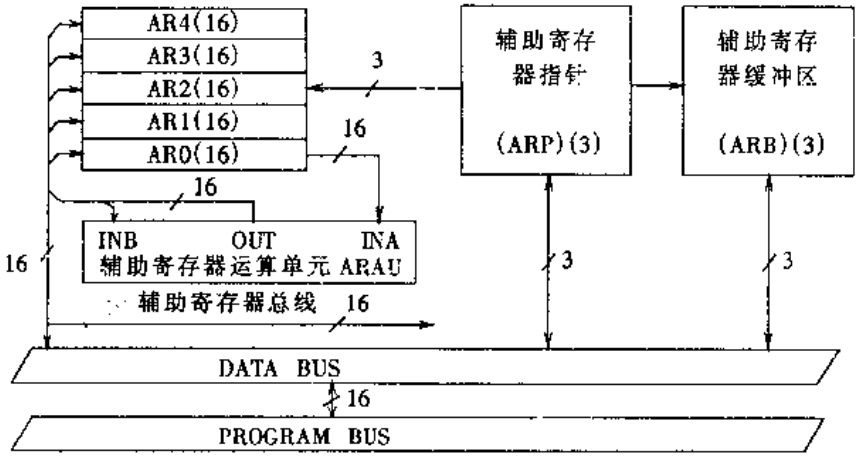
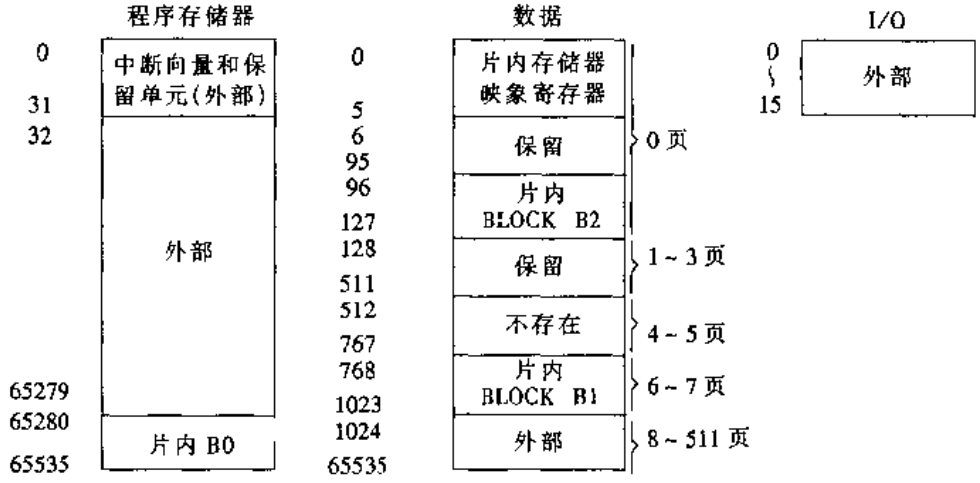
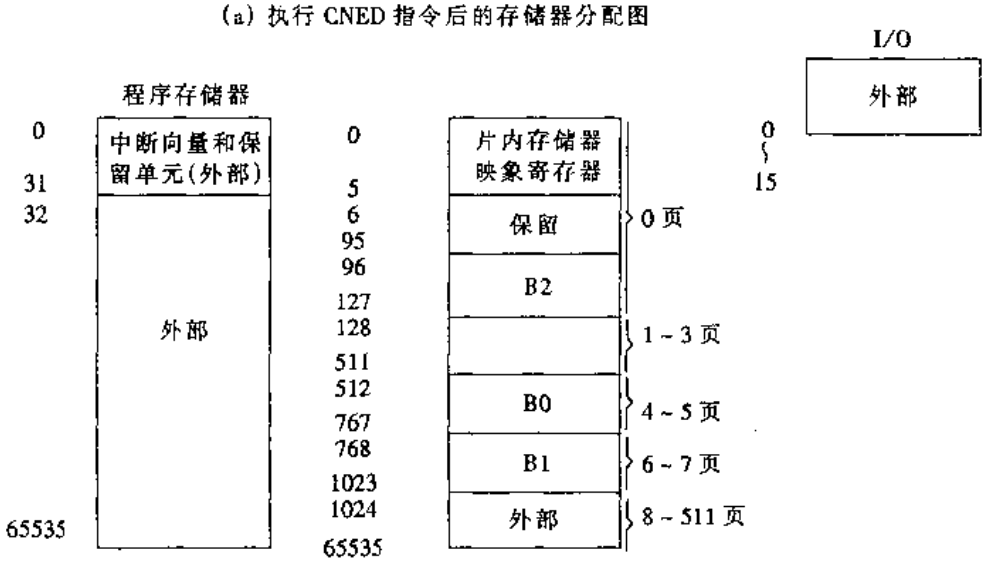


图 3－4　辅助寄存器堆

当数据存储器单元正被寻址时，ARAC 可自动变址当前辅助寄存器。可以 ± 1 变址，或者相对于 AR0 的内容变址。TMS32030 为程序存储器、数据存储器和 I/O 提供了三个独立的地址空间(图 3 - 5)。并在外部由$\overline{PS}$、$\overline{DS}$和$\overline{IS}$信号进行空间选择。TMS32020 提供了可实现数据块和程序块的存储器—存储器传送功能的指令，以进行片内或片外存储器的块传送。



(a) 执行 CNED 指令后的存储器分配图



(b) 执行 CNFP 指令后的存储器分配图

图 3 - 5　存储器分配图

TMS32020 有一个 16 位程序计数器(PC)和一个用来存储 PC 的 4 个单元的硬堆栈(图 3 - 6)。在取指时程序计数器寻址内部和外部程序存储器；在中断、子程序调用以及某些有两地址操作的专用指令期间使用堆栈。

程序计数器经过程序地址总线(PAB)寻址片内或片外程序存储器。通过 PAB 从程序存储器取出指令并装入指令寄存器(IR)后。PC 就准备开始下一个取指周期。当块 B0 用作程序存

储器时，PC 可寻址块 B0，或者通过地址总线 A15～A0 和外部数据总线 D15～D0 寻址片外程序存储器。

程序计数器在 BLKD 指令期间还寻址数据存储器，该指令从数据存储器的某一段向另一段传送数据块。

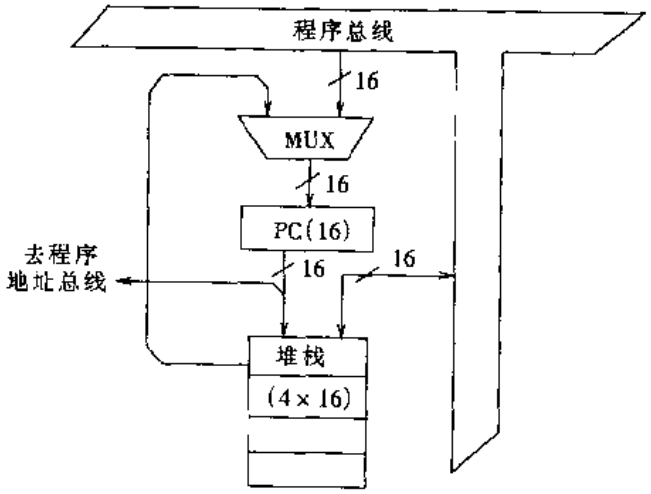


图 3－6　程序计数器和堆栈

三、中央算术逻辑单元(CALU)

TMS32020 的中央算术逻辑处理单元(CALU)包含一个 16 位定标移位器、一个 16×16 位并联乘法器、一个 32 位算术逻辑单元(ALU)、一个 32 位累加器(ACC)和累加器与乘法器输出端附带的一些移位器。如图 3－7 所示。典型的 ALU 指令执行算术操作，结果送到累加器。

ALU 的一个输入总是来自累加器，而另一个输入则由乘积寄存器(PR)或定标移位器馈给，定标移位器的输入来自数据存储器。TMS32020 定标移位器的 16 位输入连接到数据总线，32 位输出连接到 ALU。定标移位器依照指令产生 0 到 16 位左移。TMS32020 还包含另外几个移位器，允许它执行数字定标、位抽出、扩展算术运算和防止溢出。这些移位器连接到乘法器和累加器的输出端。TMS32020 有 32 位算术逻辑单元(ALU)和累加器，可以执行算术和逻辑指令，大多数指令是在单个时钟周期内执行的。输入 ALU 的数据可由定标移位器定标。TMS32020 执行由 ALU 状态确定的转移指令支持浮点操作。为把数据存到数据存储器中，32 位累加器分成两段，每段 16 位，ACCH(累加器高 16 位)和 ACCL(累加器低 16 位)。在累加器的输出端附加了一个可左移 0，1 或 4 位的移位器。这种移位是在数据向数据总线传送准备存储时完成的。累加器还有一个就地 1 位移位，用来使累加器的内容左移或右移 1 位。TMS32020 有一个补码 16×16 位硬件乘法器，它能够在单个机器周期内计算得到 32 位乘积。两个寄存器是和乘法器联系在一起；一是 16 位暂存寄存器(TR)，保存乘法器的一个操作数。另一个是 32 位乘积寄存器(PR)，保存乘积。

通常由 LT(装入 T 寄存器)指令将数据总线上的数据装入 TR，为乘法器提供一个操作数，而 MPY(乘法)指令提供另一个操作数(也来自数据总线)。在这种情况下，每两个周期得到一个乘积。

两个 16 位补码数相乘以后,32 位乘积装进 32 位乘积寄存器(PR)。P 中的乘积可以直接传送给 ALU,或者也可以在送到 ALU 之前先经过移位。

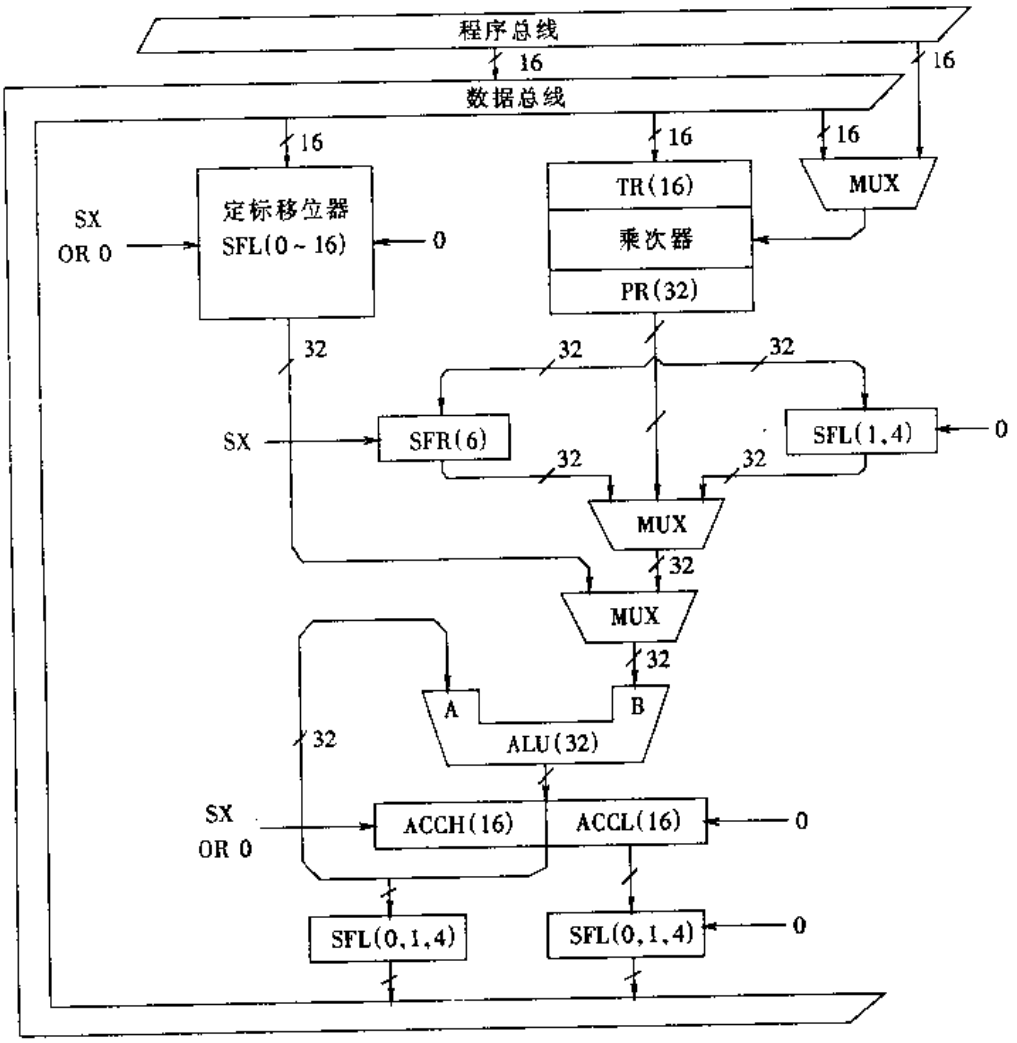


图 3-7 中央算术逻辑单元

乘积移位方式(PM)有 4 种。状态寄存器 ST1 的 PM 字段规定 PM 移位方式,如表 3-4 所示。

表 3-4 PM 移位方式

PM 值	结　　果
00	不移位
01	左移 1 位
10	左移 4 位
11	右移 6 位

T 寄存器(TR)的低 4 位为 LACT/ADDT/SUBT(装入累加器/加到累加器/从累加器减,包含

由 TR 规定的移位)指令规定通过定标移位器的移位。这些指令用在浮点算术运算中。位测试指令(BITT)允许根据 TR 中低 4 位的值测试数据存储器中一个字的某一位。

四、系统控制

TMS32020 通过片内定时器、重复计数器、3 个外部可屏蔽中断、由串行口操作或定时器产生的内部中断以及外部复位信号提供控制操作。

TMS32020 提供一个存储器映射 16 位定时器和一个片内定时寄存器(TIM)。它是一个减计数器,由一个内部时钟连续减 1。这个内部时钟是对 CLKOUT1 4 分频得到的。复位将定时器置为最大值(> FFFF)但并不对周期(PRD)寄存器初始化。复位信号释放后,定时器开始减量。以后,在程序控制下可重新装入定时器或周期寄存器。每当定时器减到零,就产生定时中断。在定时器减到零的同一周期内,可用周期寄存器(PRD)中的值重新装入定时器。这样可以通过编程使中断以 4×(PRD)×CLKOUT1 为周期等间隔地出现。这种情况对于同步采样或向外围设备同步输出是很有用的,可以在任何时候对定时寄存器(TIM)和周期寄存器(PRD)进行存取。周期寄存器的值不能为零。如果不用定时器,定时器中断应被屏蔽,或者用 DINT 指令禁止所有的可屏蔽中断。

TMS32020 的设计包括一种重复特性,它允许一条指令最多可重复执行 256 次。它可将数据存储器的值(用 RPT 指令)或一个立即数(用 RPTK)装入重复计数器(RPTC)。该操作数之值应比下一条指令被执行的次数小 1。重复特性可在乘法/累加、块传送、I/O 传送和表格读/写这样一些指令中使用。

TMS32020 提供 3 个外部的可屏蔽用户中断($\overline{INT2}$ ~ $\overline{INT0}$),由外部器件用来中断处理器。内部中断由串行口、定时器和软件中断指令产生。中断有优先级,复位有最高优先级,而串行口发送中断有最低优先级。所有的中断单元都在两字的边界,如果需要可在那些单元中安排转移指令。

$\overline{RS}$信号使 TMS32020 中止执行并强制使程序计数器为零。$\overline{RS}$影响各种寄存器和状态位。当接通电源时,处理器的状态是不确定的。为了通电后的系统操作,必须建立复位($\overline{RS}$)信号至少 3 个时钟周期以确保器件复位。处理器从 0 单元开始执行。0 单元中通常包含一条转移(B)指令,把程序执行转向系统初始化程序。

TMS32020 有两个状态寄存器 ST0 和 ST1,包含各种条件和方式的状态。用 SST 和 SST1 指令把状态寄存器的内容存进数据存储器中。而用 LST 和 LST1 指令从数据存储器装入状态寄存器。使用这种方法可在中断和子程序调用时保存器件的当前状态。

五、外部接口

TMS32020 三个独立的数据、程序和 I/O 地址空间提供了到存储器和 I/O 的接口,局部存储器接口包括:

·16 位并行数据总线(D15 ~ D0)。

·16 位地址总线(A15 ~ A0)。

·数据、程序和 I/O 空间选择信号($\overline{DS}$、$\overline{PS}$和$\overline{IS}$)。

·各种系统控制信号。R/$\overline{W}$ 信号控制传送方向,而$\overline{STRB}$提供定时信号来控制传送。

TMS32020 用和存储器相同的方法处理 I/O 器件,利用处理器的外部地址和数据总线把 I/O 器件映射到 I/O 地址空间。

用 READY 线来完成与各种不同速度的存储器和 I/O 器件接口。当与较慢的器件通信时，TMS32020 处理器进行等待，直到那个器件完成它的功能并经过 READY 线向处理器发出信号为止。然后 TMS32020 继续工作。

串行口用来与串行器件直接通讯，如编码译码器、串行 A/D 转换器和其他串行系统。接口信号是与编码译码器和许多其他串行器件兼容的。只需要很少的外部硬件，串行口还可在多道处理应用中用于处理器之间的交互通信。

TMS32020 的灵活性使它能满足广泛的系统需要。使用 TMS32020 的一些系统配置如下：

·独立自主的系统(单个处理器)。

·共享全局数据存储器的主/从式或并行多道处理系统。

·使用接口控制信号的主/外围式协同处理器配置。

在多道处理应用中，TMS32020 能定位全局数据存储器空间，并通过 $\overline{BR}$(总线请求)和 READY 控制信号与该空间通信。全局存储器是由一个以上处理器共享的数据存储器。全局数据存储器存取必须经过仲裁。

在多道处理器环境中，使用$\overline{SYNC}$输入可极大地简化处理器间的接口。该输入用来使系统中每个 TMS32020 的内部时钟同步，因此允许多个处理器同步地运行。

TMS32020 用$\overline{HOLD}$和$\overline{HOLDA}$(保持响应)信号支持它的外部程序/数据存储器的 DMA(直接存储器存取)。另一个处理器通过向$\overline{HOLD}$送一个"低"，可以完全控制 TMS32020 的外部存储器。这使 TMS32020 的地址、数据和控制总线处于高阻抗状态。

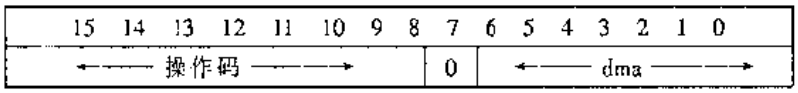
3.3 TMS32020 的指令系统

一、存储器寻址方法

TMS32020 指令系统支持三种存储器寻址方式，即直接寻址方式、间接寻址方式和立即寻址方式。

1. 直接寻址方式

在直接寻址方式中，指令字包含数据存储器地址的低 7 位，该字段与 9 位数据存储器页面指针(DP)寄存器衔接，形成完整的 16 位数据存储器地址。除了 CALL 指令、转移指令、立即数指令和无操作数指令外，直接寻址适用于所有指令。直接寻址指令格式如下：



2. 间接寻址方式

TMS32020 通过辅助寄存器 AR0—AR4 能够灵活方便地完成间接寻址。为了选中一个辅助寄存器，将 0—4 中指定的数字装入辅助寄存器指针(ARP)即可。

在间接寻址中，64K 字数据存储器空间中的任何单元都可以用辅助寄存器内容的 16 位地址进行存取。用 LAR、LARK 或 LRLK 指令可以装入需要的数据存储器地址。通过 MAR 指令或指令字中的修改辅助寄存器指针内容来改变辅助寄存器指针。

间接寻址方式中常用下列符号：

*　　AR(ARP)　　当前内容作为数据存储器地址。

* -	AR(ARP)	当前内容作为数据存储器地址，在数据存储器存取后其内容减 1。
* +	AR(ARP)	当前内容作为数据存储器地址，在数据存储器存取后其内容加 1。
*0-	AR(ARP)	当前内容作为数据存储器地址，存储器存取后，其内容减 AR0 内容。
*0+	AR(ARP)	当前内容作为数据存储器地址，数据存储器存取后，AR 内容加上 AR0 的内容。

在间接寻址方式中包含两种变址型间接寻址，即加 1 和减 1 的简单变址型间接寻址和基于 AR0 值的变址型间接寻址。在两种情况中，由 ARP 寄存器指定的辅助寄存器内容用作数据存储器操作的地址。随后 ARPU 对指定的辅助寄存器进行算术操作。另外，间接寻址方式的指令可以装入一个新的 ARP 值。间接寻址可以和除了立即数指令、无操作数指令之外的所有指令一起使用。间接寻址字段如表 3-5 所示。间接寻址格式如下：

15 14 13 12 11 10 9 8	7	6	5	4	3	2 1 0
←—— 操作码 ——→	0	IDV	INC	DEC	NAR	← dma→

位 0～位 2	辅助寄存器指针值(ARP)。
位 3(NAR)	决定 ARP 是否要装入新值，
位 3 = 1	则将位 0～位 2 的内容装入 ARP。
位 3 = 0	则 ARP 指针的内容保持不变。
位 4(DEC)	控制辅助寄存器减 1。当位 4 = 1 时，ARP 指定的辅助寄存器减 1，当位 4 = 0 时不减。
位 5(INC)	控制辅助寄存器增 1。当位 5 = 1 时，ARP 指定的辅助寄存器增 1，当位 5 = 0 时不增。
位 6(IDV)	决定变址的类型。位 6 = 0 时，当前辅助寄存器加 1 或减 1，位 6 = 1 时，当前辅助寄存器加或减 AR0。
位 7	位 7～1 表示指令为间接寻址的方式。
位 8～位 15	指令操作码。

表 3-5 间接寻址字段表

指令字段位 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	符号	操作
←— OPCODE —→ 1 0 0 0 0 ← Y →	*	NOmanipulation of ARS/ARP
←— OPCODE —→ 1 0 0 0 1 ← Y →	*,Y	Y→ARP
←— OPCODE —→ 1 0 0 1 0 ← Y →	* -	AR(ARP) - 1→AR(ARP)
←— OPCODE —→ 1 0 0 1 1 ← Y →	* -,Y	AR(ARP) - 1→AR(ARP) Y→ARP
←— OPCODE —→ 1 0 1 0 0 ← Y →	* +	AR(ARP) + 1→AR(ARP)
←— OPCODE —→ 1 0 1 0 1 ← Y →	* +,Y	AR(ARP) + 1→AR(ARP) Y→ARP
←— OPCODE —→ 1 1 0 1 0 ← Y →	*0-	AR(ARP) - AR0→AR(ARP)
←— OPCODE —→ 1 1 0 1 1 ← Y →	*0-,Y	AR(ARP) - AR0→AR(ARP) Y→ARP
←— OPCODE —→ 1 1 1 0 0 ← Y →	*0+	AR(ARP) + AR0→AR(ARP)
←— OPCODE —→ 1 1 1 0 1 ← Y →	*0+,Y	AR(ARP) + AR0→AR(ARP) Y→ARP

3.立即寻址方式

在立即寻址中,指令字包含立即数的值。立即数可以包含在指令字内或者在紧接操作码后面的一个字中。下列指令的指令字中包含立即数,并在单指令周期内执行。其操作数的长度由指令决定。

LACK　将短立即数装入累加器(8位绝对数)。

LARK　将短立即数装入辅助寄存器(8位绝对常数)。

LARP　装入辅助寄存器指针(3位常数)。

LDPK　将立即数装入数据存储器页面指针(9位常数)。

MPYK　乘立即数(13位二进制补码常数)。

RPTK　由立即数指定重复次数的重复指令(8位常数)。

下列立即数指令中,紧接指令操作码后面的一个字中包含16位常数值。16位值可以选择作为绝对常数或二进制的补码数使用。

ADLK　长立即数经移位加到累加器(绝对的或二进制补码值)。

ANDK　立即数同累加器"与",带移位。

LALK　长立即数经移位装入累加器(绝对的或二进制补码值)。

LRLK　将长立即数装入辅助寄存器。

ORK　立即数与累加器"或",带移位。

SBLK　累加器减经移位的长立即数(绝对的或二进制补码值)。

XORK　立即数与累加器"异或",带移位。

二、TMS32020指令系统

从指令的功能上看,可以把32020的指令划分为6类:即设置累加器Acc和数据存储器DRAM操作指令;有关辅助寄存器AR和页指针DP的指令;关于TR、FR和乘法器的指令;转移和调用的指令;有关控制、调试的指令;I/O和DRAM传送的指令。实际上,这些指令完成的功能与通用处理器指令功能相似,也是包括运算逻辑、传送、判断和转移以及I/O功能等。

TMS32020的Acc是算术逻辑运算的中心,甚至转移地址也用Acc内容。对于两个操作数的运算,一个隐含于Acc,另一个为指令中的显式操作数。

显式操作数有二种可能情形。一种为立即数,有长短之分。长短立即数分别为16位和8位。另一种为DRAM单元。分为直接、间接寻址两种。另外,TMS32020的一些指令也很特别,如对片外RAM B0块的设置、数据块传送等。

指令系统、使用符号和缩写说明见附录中的一。

1.累加器、存储器指令

在这类指令中,含有加、减或异或等通用算逻指令。操作数一个为Acc,另一个是立即数或DRAM单元,结果在Acc中,规格化指令NORM对于浮点运算很有用。它把不规格化的数转变为尾数和阶码的形式。例如:

```
LARP  1        ;AR1 存指数
LARK  1,0      ;AR1 清 0
RPTK  14       ;15 位规格化
```

```
NORM          ;规格化
```

2.辅助寄存器和页指针指令

这类指令为寻址寄存器和指针的指令。LARP、LAR 和 LDP 分别对辅助寄存器指针、辅助寄存器和 DRAM 页指针装入。

3.关于乘法的指令

这类指令中包括对 TR、PR 和 PM 的装入指令 LT、LPH 和 SPM。还有积寄存器 PR 移入 Acc 以及作积和、积差的指令 PAC 和 APAC、SPAC。乘积功能指令包括操作数在 DRAM 中的乘积 MPY、平方累加 SQRA 等以及 DRAM 与单元间的积累加 MAC 和 MACD 等。例如作积累加的程序如下：

```
SPM  3              ;PR 累加前右移 6 位
LARP  1             ;
LRLK  1,>300        ;AR1 指向 B1 块
CNFP                ;B0 块配置为 PRAM
PR7K  >FF           ;重复
MAC >FF00,*+        ;积加
```

4.关于转移和调用指令

这类指令用于控制程序执行流程，包括无条件转移和条件转移以及调用子程序指令等。控制条件有辅助寄存器内容和位控制以及 Acc 内容等。例如 AR2 作循环控制向一块单元中填 0 的程序如下：

```
LARP  1
ZAC
LARK  1,BEGIN          置 AR1 填区首址
LARK  2,NUM            ;重复次数
LOOP  SACL*+,0,2       ;填 0
BANZ  LOOP,*-,1        ;判断结束
```

调用子程序时要注意堆栈空间的限制，嵌套调用不能超过 4 层。

5.控制和测试指令

这类指令是控制和测试指令。位测试指令常用于转移指令，此类指令中还包括对处理状态的设置。用 CNFP 可把片内 B0 块 RAM 设置成程序存储器 PRAM 地址在 >FF00 至 FFFF；用 CNFD 设置 B0 块为 DRAM，地址在 >200 至 >2FF。重复指令 RPT 与多周期指令连用可提高效率。例如，

```
RPTK  >7F           ;重复 128 次
BLKD  >2000,*+      ;源首址 >2000,目的地址
                    ;由 AR 指出
```

6.I/O 和数据存储器操作、指令

这一类指令是 I/O 和 DRAM 块操作指令。输入指令 IN 将输入口（共有 16 个）的数据读入。通常开机后，第一次是起初始化作用，因而首先读两次取得可靠值。输出指令 OUT 把数据写入到输出口，共有 16 个。读表指令 TBLR 把 PRAM 中数据读入 DRAM 中。PRAM 地址由 Acc 提供，DRAM 地址在指令中给出。写表指令 TBLM 方向与 TBLR 相反。

块传送指令 BLKP 和 BLKD 分别把 PRAM 和 DRAM 中数据送到 DRAM 中。它们常与 RPT 指令合用传送一块数据。值得注意的是表读写和块传送指令执行都用到堆栈。执行后会破坏原栈底数据。数据下移指令 DMOV 只能在片内 DRAM 中使用，在片外无效。

3.4 TMS32020 与外部设备接口

一、TMS32020 与外部设备接口

TMS32020 对外部存储器或 I/O 装置的响应速度的要求为：

读周期：当存储器或 I/O 装置被$\overline{PS}$、$\overline{DS}$或$\overline{IS}$选中后（通常是只读器件），必须在 90ns 以内给予数据响应。故所用的外部存储器或 I/O 装置必须有较快的寻址时间，否则为完成读操作需插入适当的等待状态。当用$\overline{PS}$、$\overline{DS}$或$\overline{IS}$同$\overline{STRB}$一起作为控制信号时（通常适用于读写器件），存储器或 I/O 装置必须在 STRB 变低的 50ns 之内作出响应。

写周期：TMS32020 提供了 50ns 写数据的建立时间，若有必要，可用$\overline{STRB}$信号产生等待状态。

可以看出：TMS32020 同外部存储器或 I/O 装置连接时，如何解决快速处理芯片与低速的被连接器件之间的时间上配合（或速度匹配）问题，是设计一个合理的接口电路的关键所在。

1. TMS32020 与 PROM 的连接

TMS32020 与 PROM 的连接图如图 3-8 所示。

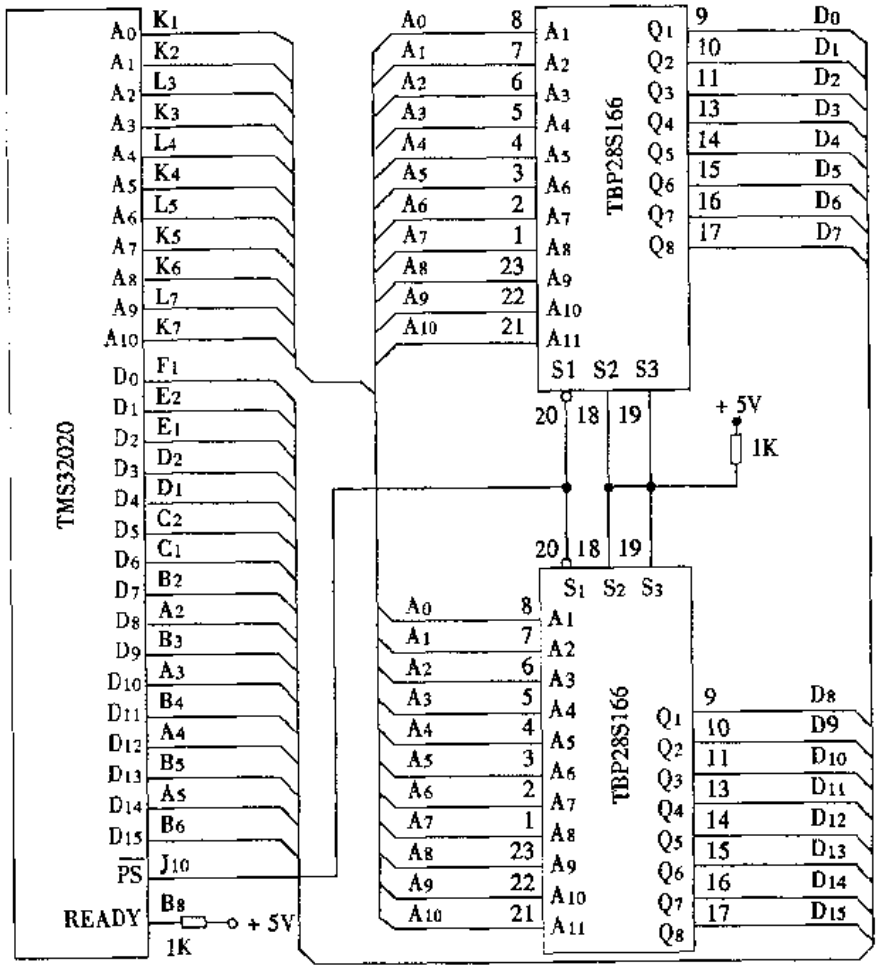


图 3-8 TMS32020 与 TBP28S166 接口

TBP28S166 是 2048 × 8 的 PROM，输出有三态，寻址时间为 75ns（小于 90ns），能满足 TMS32020 的定时要求。这里用$\overline{PS}$作片选信号。若一个系统中有多组存储器，可用$\overline{PS}$作为译码器的控制信号，用几位高地址经译码器译码输出实现组的寻址，如图 3－9 所示，定时关系如图 3－10。

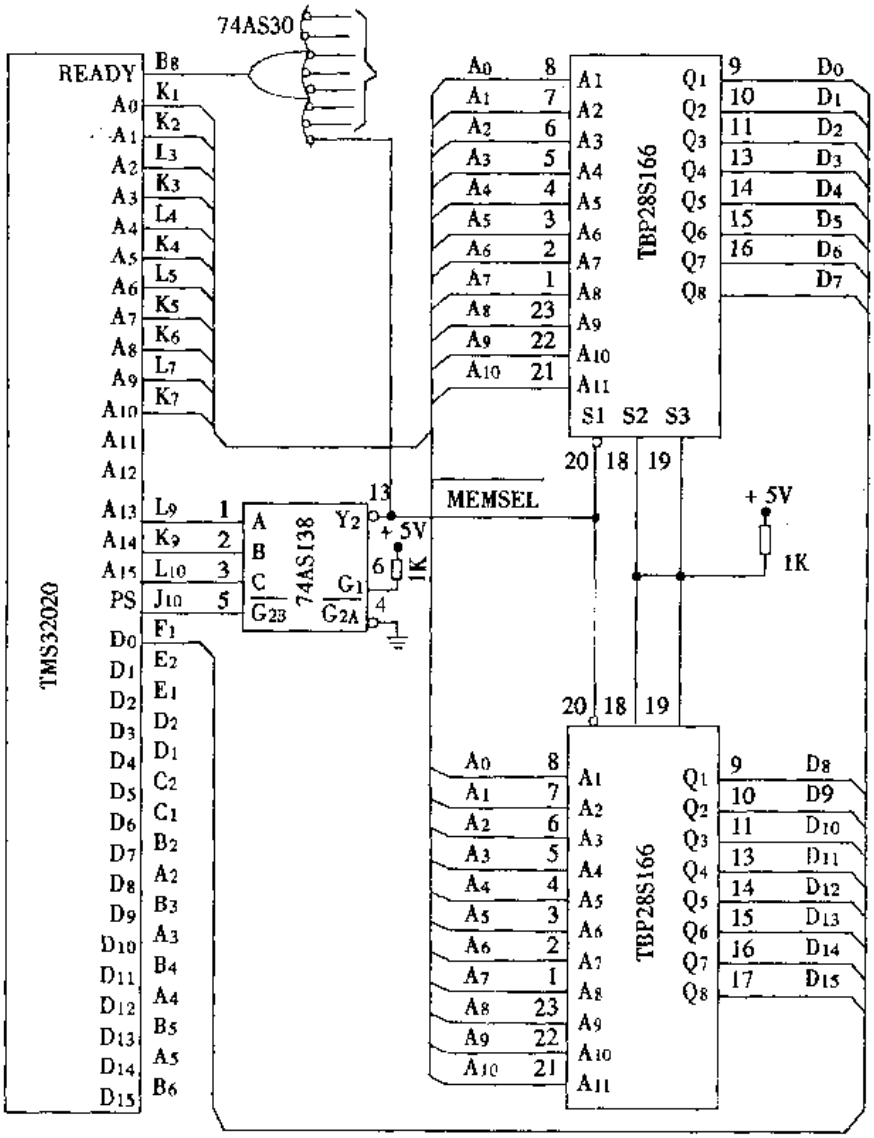


图 3－9　带译码器的 TBP28S166 与 TMS32020 连接

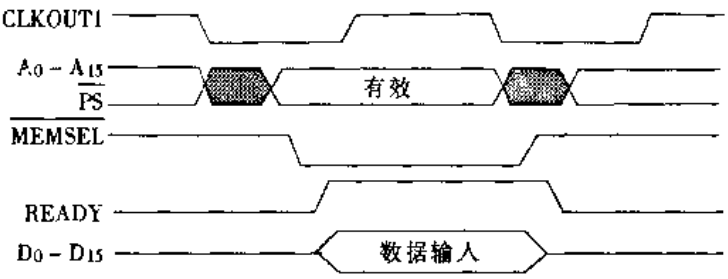


图 3－10　TBP28S166 与 TMS32020 接口时序

2. TMS32020 与 EPROM 的连接

TMS2764 - 35 EPROM(8192 × 8)的寻址时间是 350ns(> 90ns),不能直接满足 TMS32020 定时要求,需要一个等待状态产生器提供两个 200ns 等待状态。图 3 - 11 是等待状态发生器电路,在 A 处输入时,产生两个等待状态的输出,在 B 处输入时,产生一个等待状态的输出,定时关系如图 3 - 12 所示。74AS138 译码器输出作为控制信号。考虑到 EPROM 有较长的截止时间,为了避免数据总线冲突,在 TMS2764 - 35 输出端增加了 74AS244 作为缓冲器。图 3 - 13 表示定时关系。图 3 - 14 是 TMS32020 与 TMS2764 - 35 连接图。

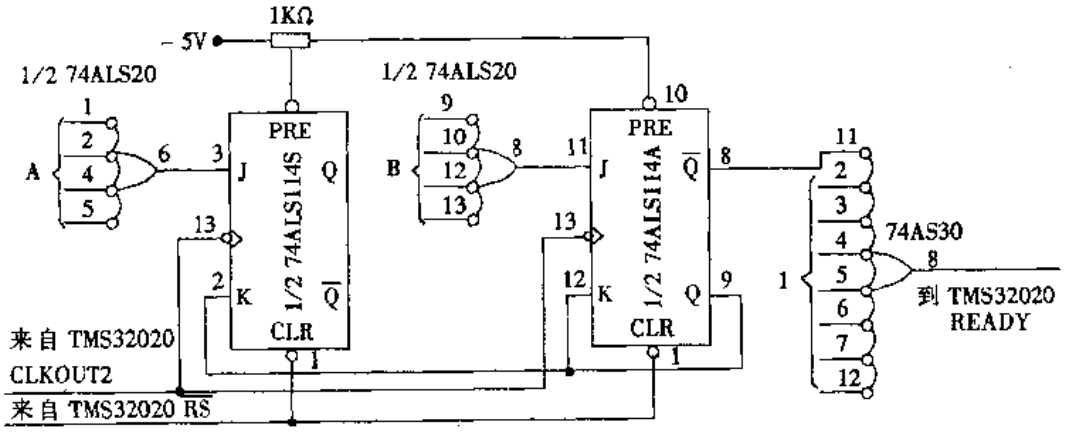


图 3 - 11 等待状态发生器

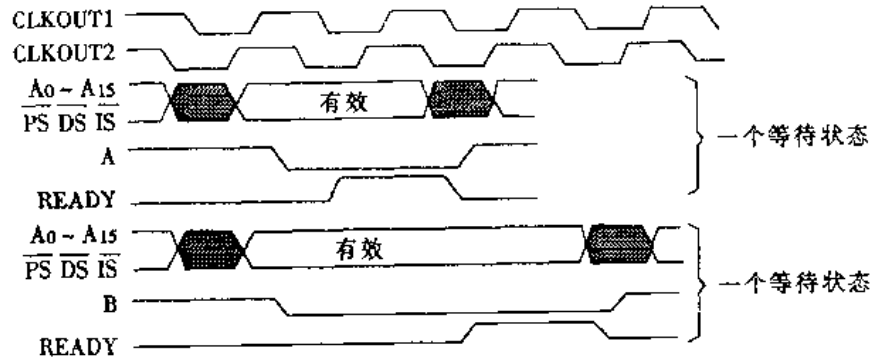


图 3 - 12 等待状态发生器时序图

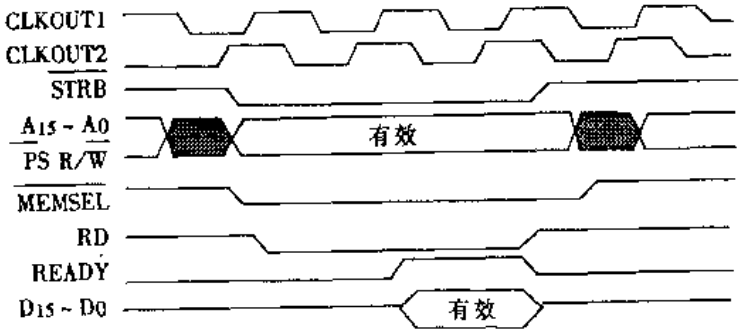


图 3 - 13 TMS2764 与 TMS32020 接口定时

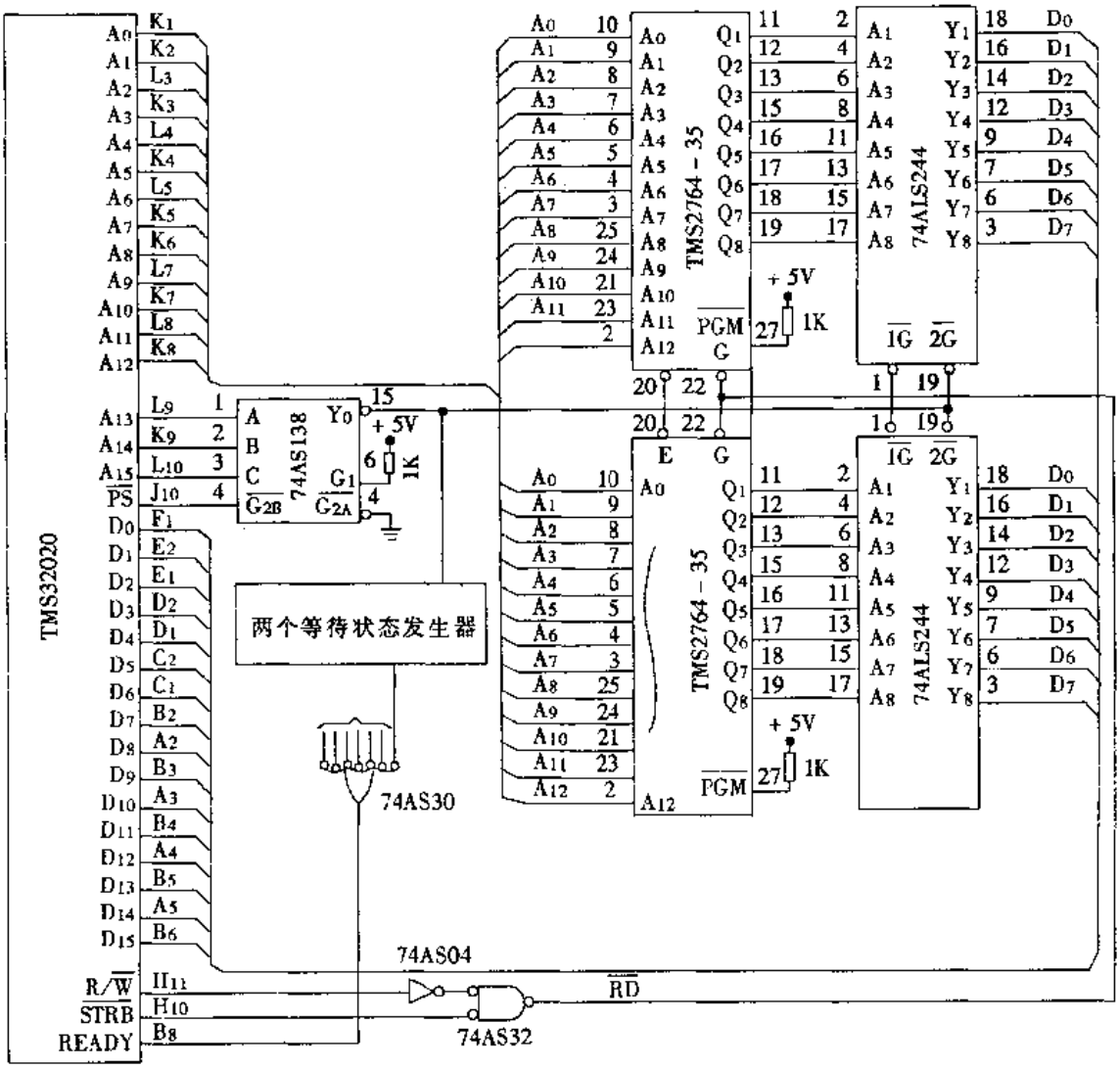


图 3 - 14　TMS2764 与 TMS32020 连接图

3. TMS32020 与静态 RAM 的连接

图 3 - 15 为 INMOS IMS1421 - 50 静态 RAM(4096 × 4)同 TMS32020 的连接，图中 RAM 为程序存储器，$\overline{PS}$为控制信号，定时关系如图 3 - 18。在读周期，IMS 1421 - 50 在被选中后 40ns 内给予数据响应，能满足 TMS32020 的要求，在写周期，$\overline{MEMSEL}$形成写时间，为保证可靠写入并避免数据总线发生冲突，$\overline{W}$(写信号)应在数据出现在数据总线前就已经稳定，并在数据消失后仍保持一段时间。

若将 IMS1421 设计成数据存储器，只需将$\overline{PS}$改为$\overline{DS}$，其它完全一致。

若选择其它存储器，不能满足 TMS32020 的要求，可以用等待状态产生器插入适当的等待状态来驱动 TMS32020 READY 线。

若存储器的截止时间过长，可增加输出缓冲器，防止总线冲突。

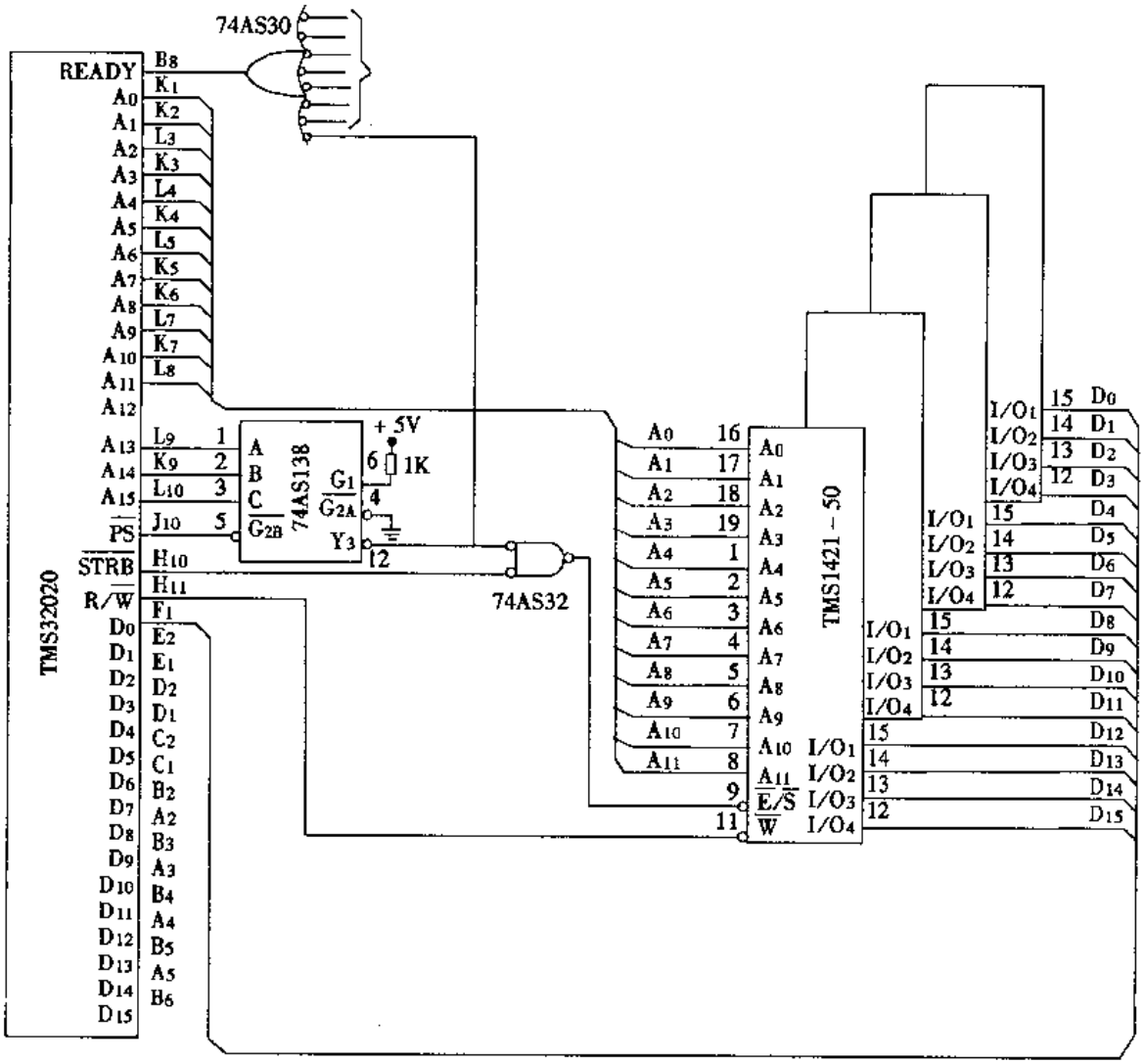


图 3－15　IMS1421－50 与 TMS32020 连接

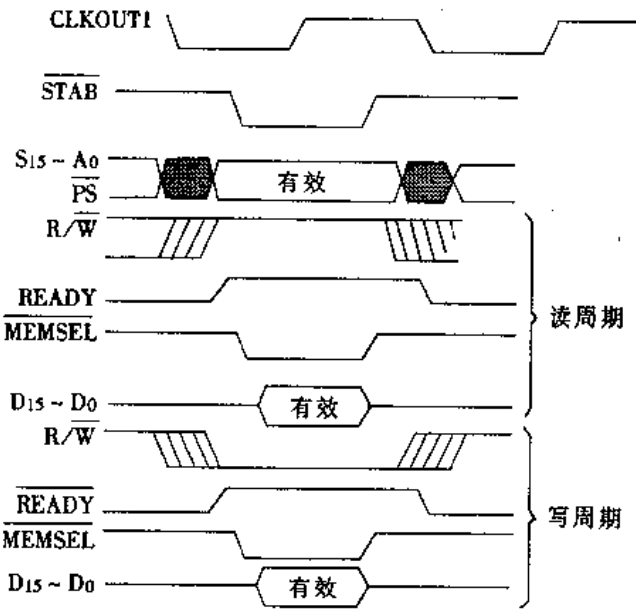


图 3－16　IMS1421－50 与 TMS32020 接口定时

4.TMS32020 与 D/A 转换器接口

8 位 D/A 转换器 TLC 7524 可以以最小的外部电路请求与 TMS32020 相连，如图 3－17 所示，接口电路包括有一个 SN74LS138 的 3－8 线译码器，用于对外围设备地址译码。

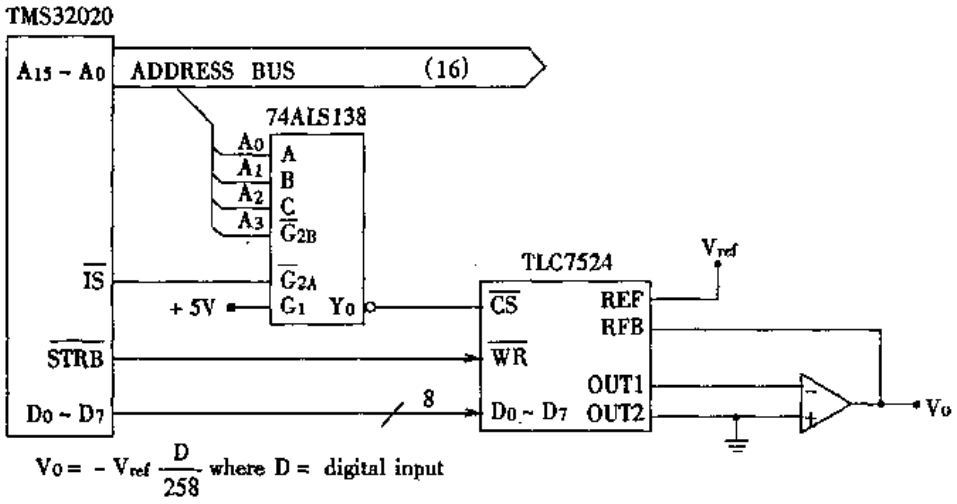


图 3－17　TMS32020 与 TLC7524 接口

当 TMS32020 执行 OUT 指令时(见图 3－18)，外部设备地址置于地址总线，同时$\overline{IS}$先变低电平，指定总线地址相对应的一个 I/O 通道和非外部数据或程序存储器。$\overline{IS}$低电平启动 74LS138 译码器，并使 Y 输出带相应的总线地址并变低电平。当 Y 输出是低电平时，TLC7524 启动，数据出现在数据总线上并由$\overline{STRB}$锁存在 D/A 转换器。

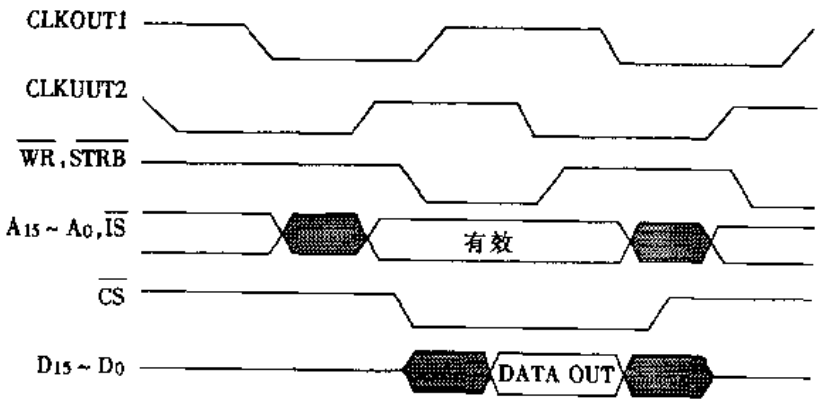


图 3－18　TMS32020 与 TLC7524 接口时序

5.TMS32020 与 A/D 转换器连接

TMS32020 与 8 位 A/D 转换器 TLC0820 连接线路如图 3－19 所示。由于 TLS0820 的控制电路的运行比 TMS32020 稍慢，因而在接口电路中使用下列逻辑电路。

(1)3—8 译码器(SN74ALS138)

(2)两输入与非门(SN74LS00)

(3)反向器(SN74LS04)

(4)两输入 OR 门(SN74LS32)

(5)D 触发器(SN74LS175)

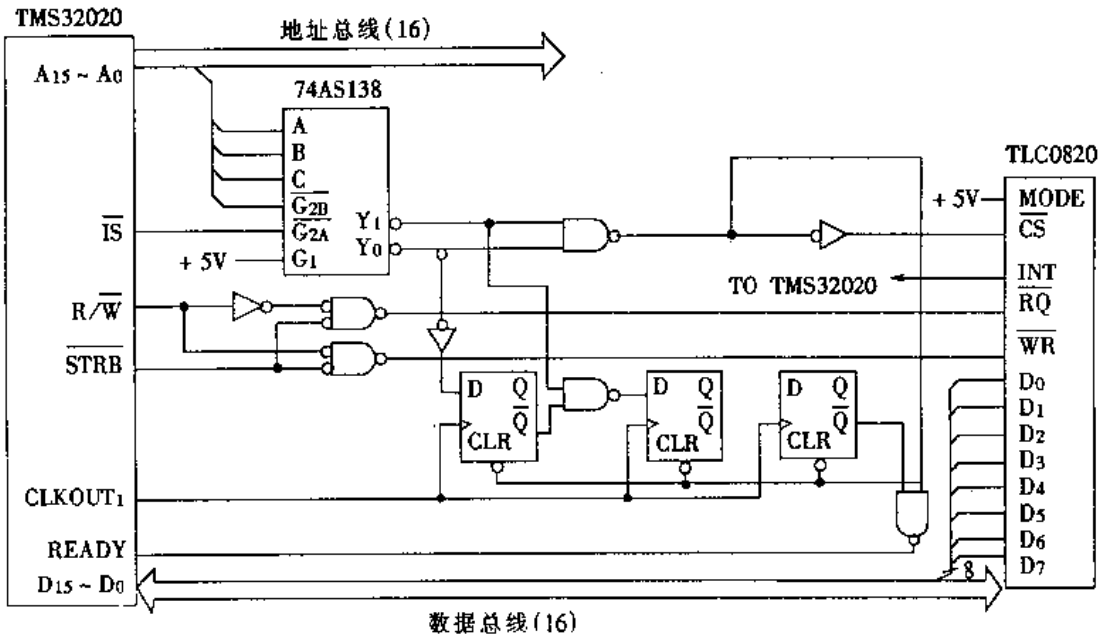


图 3 - 19　TMS32020 与 TLC0820 连接

74LS138 译码地址指定 TLC0820，在执行写操作时，使用这些地址中的一个地址，其余地址用于读操作，这两种不同的地址适宜为读写操作提供等待状态校验。

当 TMS32020 工作在 20MHz 而 TLC0820 配接低速存储器，需 3 个等待状态，以适应长写脉冲的需要。转换开始后，TMS320C20 在转换结果可读之前，需等待长达 600ns，这可由软件提供这种足够长的延时，为读取转换结果，亦需提供长的等待状态容许以 TLC0820 数据存取时间（最小为 320ns）。其指令时序如图 3 - 20，在通道 1 存取时需 2 个等待状态。

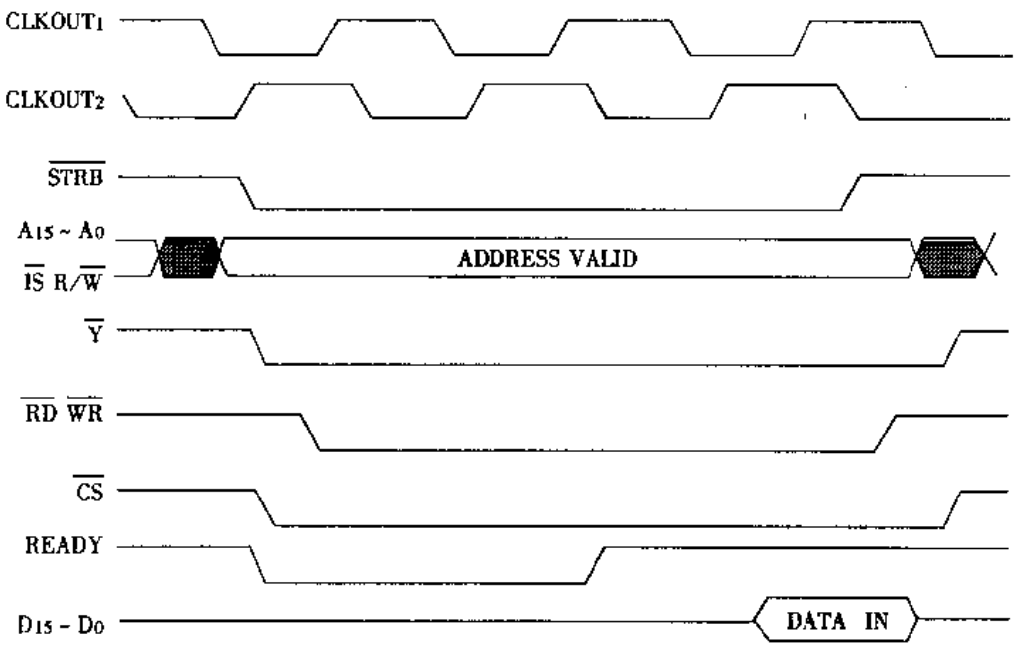


图 3 - 20　TLC0820 与 TMS320C 接口时序

二、TMS32020 组成处理系统

TMS32020 单独一个芯片不能工作。它的数据、程序以及控制等都需要外部的支持，即需要以处理器芯片为基础辅加以一定的外部电路，构成可以完成一定工作的系统。

根据 TMS32020 的功能和应用要求，一般可以构成以下几种配置的系统。

一种是 TMS32020 独立的系统，即一个处理器配以一定量的外部存储器构成一个小配置系统；另一种是采用主从方式的系统，以一个处理器为主处理器，另一个为辅助处理器，两者可以共用数据存储器，但控制权由主处理器掌握；再有一种为并行的多处理器系统，设置全局存储器，它不属于任一处理器，但每个处理器都可使用。

1.独立的系统

构成系统的主要任务是要为处理器配备一定量的外部存储器。对速度要求很高时，可以选择存取时间较小（<80ns）的存储器芯片，使得 32020 不用等待而在一个机器周期内完成存取操作。当存储器存取时间较长时，需要插入几个等待的机器时间。这时要用 READY 信号加入延迟。

当芯片内部数据 RAM 容量已可满足系统处理要求时，则只需扩充外部程序存储器。如图 3－21 所示。

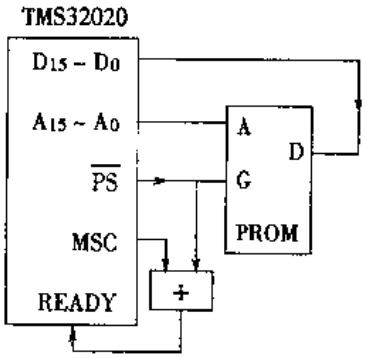


图 3－21　扩展程序存储器的系统

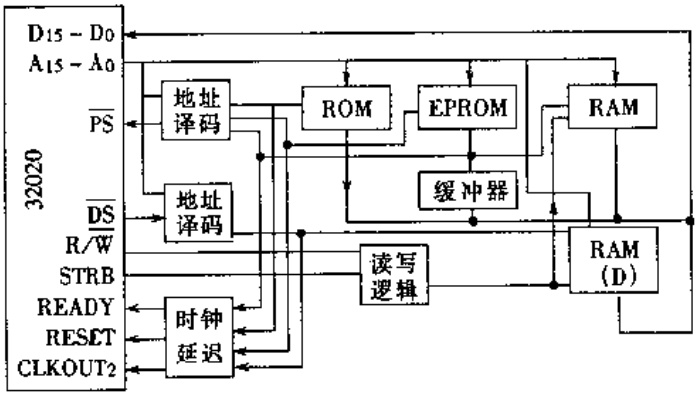


图 3－22　扩展程序和数据存储器的系统

有些处理工作所需数据空间相当大，这时就应扩充外部数据存储器。DRAM 需要读写，所以芯片采用 RAM。程序存储器中存储程序，往往只需读且在执行中不需修改，系统可以使用

ROM。而对于开发系统，可用 EPROM 和 RAM。图 3－22 是具有三种存储器的情况。

根据所用芯片的速度可用 READY 信号延迟一定的周期数。对于 DRAM，由于片内 RAM 占用 0 至 7 页，外部 DRAM 从第 8 页开始编址，即占 >2000 开始的高段空间。

2．主从系统

有些任务需要两个处理器协同工作，这时往往采用主从工作方式。主处理器可以是现有通用微机的 CPU，而从处理器为 TMS32020。这样可以利用通用微机上的系统和软件。通过微机可以使用 TMS32020，例如，可以向 TMS32020 装入程序和对 TMS32020 指令进行交叉汇编等。构成这样一个系统时，从处理器 TMS32020 一般应具有程序存储器和局部数据存储器，主处理器通过$\overline{\text{HOLD}}$信号对 TMS32020 进行控制。构成图如图 3－23 所示。

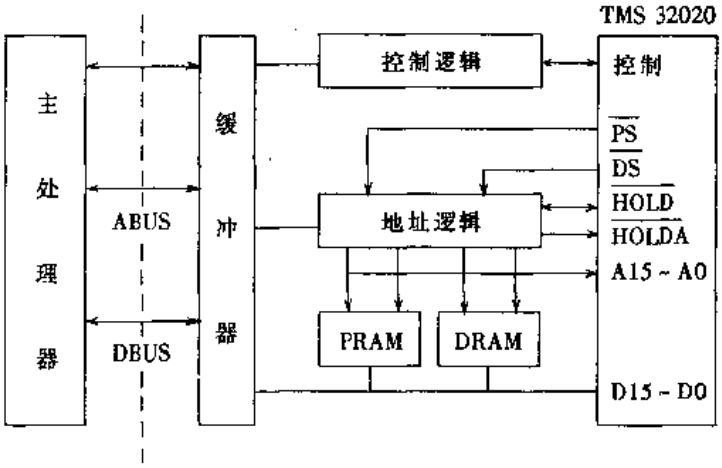


图 3－23 主从系统

3．多处理器并行系统

在一些信号处理工作中，有时需要较高的处理速度。这时，可以采用多个处理器并行工作以提高速度。多个处理器协同工作时，必须进行数据交换，使用全局存储器可以达到这个目的。TMS32020 允许使用的最大空间为 32K 字的全局存储器。每个 TMS32020 都应具有局部存储器。TMS32020 之间可以用$\overline{\text{SYNC}}$信号实现同步工作。当需要使用全局存储器时，TMS32020 发出$\overline{\text{BR}}$信号，并等待仲裁电路裁决。当允许使用时，仲裁电路向 TMS32020 发 READY 信号，使其开始工作。

两个处理器使用全局存储器时的构成如图 3－24 所示。多个处理器的情况也可类似地扩展。

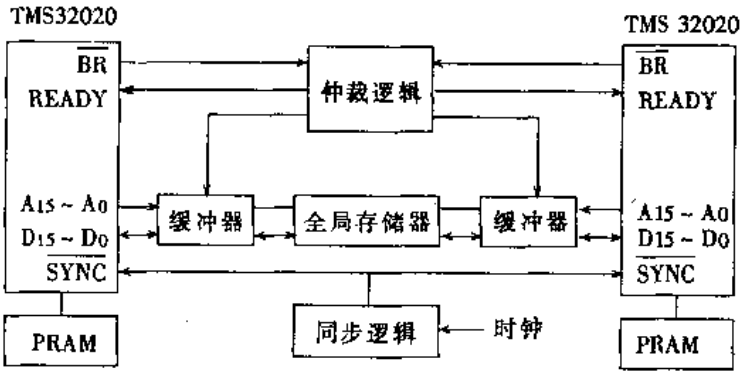


图 3－24 全局存储器构成

3.5 TMS320C20 汇编程序设计

一、汇编程序设计

1.直接寻址和间接寻址方式程序

程序如下:(LIST1)

```
        LIST1:

        IDT     'TEST'
        TITL    'ADDRESSING MODE EXAMPLE'
        AORG    0
        B       START
        AORG    20
START   LARP    AR3
        LRLK    AR0,2
        LRLK    AR3,512
        LALK    00
        SACL    *0
        LALK    200
        SACL
*
* (a) 直接寻址方式
*
        LDPK    4               ;页指针
        LAC     0               ;单元 512 中数据装入 Acc
        SACL    1               ;数据存储在单元 513 中
        LAC     2               ;单元 514 中数据装入 Acc
        SACL    3               ;数据存入 514 单元
*
* (b) 间接寻址方式
*
        LARP    AR1             ;辅助寄存器指针
        LRLK    AR1,512         ;数值 512 装入 AR1
        LAC     *+              ;512 中数据装入 Acc
        SACL    *+              ;数据存入 513
        LAC     *+              ;514 中数据装入 Acc
        SACL    *               ;数据存入 515
```

直接寻址方式时,用指令字中低 7 位的数据存储器地址和数据存储器页面指针指定地址。

例如，访问 B0 块的打头地址时，数据存储器从零地址开始，每 128 字为 1 页，地址 512 恰为第 4 页首地址。这样，将数据存储器 512 地址的数据传送到累加器。

间接寻址方式使用辅助寄存器 AR0 ~ AR5。例如，当访问 512 地址时，可将 512 送入辅助寄存器进行地址指定，而与页数无关。

一般而言，B2 块可用于变量或子程序参数的交换区，数据的存取采用直接寻址方式，其它区域的存取应采用间接寻址方式。

2.初始化程序

主程序首部的程序段多为初始化程序，以对信号处理器进行初始化。初始化程序如下：

LIST2：初始化程序

```
        IDT 'EMIN'                      ;程序名
        TITL 'INITIALIZED EXAMPLE'      ;初始化程序例
        B       START
        RORG    $ +30
*       INITIALIZED SECTION
START   SOVM                            ;设定为溢出方式
        SSXM                            ;扩展符号
        LDPK    0                       ;设定数据页 0
        SPM     1                       ;P 寄存器传送时左移 1 位
        LARP    AR1                     ;使用辅助寄存器 AR1
FIN     B       FIN
        END
```

初始化程序中的 IDT 是说明程序名称的汇编语言伪指令，采用浮动汇编程序时，必须对每一个程序命名，TITL 是说明汇编表题名的伪指令，在程序中可以不用。

B 是无条件转移指令，转到标号 START。RORG 是伪指令。该指令设置的程序地址取决于连接程序，并置入程序计数器。$ +30 因前一转移指令为 2 字，连接时从 0 开始装入，因此 START 的地址为 32。SOVM 指令是当累加器出现上溢或下溢时，分别将正、负最大值置为 7FFFFFFFH 和 80000000H。欲保持上溢、下溢值不变时，用 ROVM 指令。SSXM 置符号扩展方式。在符号扩展方式中，移位时左空位被 MSB 填充，即保持移位前的数据符号。LDPK 是页指针的立即指令。在本程序例中定义为 0 页。SPM 指令当操作数为 0 时不移位，为 1 时左移 1 位，2 时左移 4 位，3 时右移 6 位后将 P 寄存器值传送到累加器。在本程序例中，左移 1 位，但考虑到处理数的动态范围、溢出等因素，必须规定操作数。LARP 指令将辅助寄存器指针定义 AR1，这时操作数可以写 1，也可以写 AR1。

3.子程序及子程序调用

LIST3 为输入子程序 INPUT 例。在子程序中，必须使用 IDT 和外部定义伪指令 DEF。DEF 指定子程序的输入地址，其操作数在程序中必须是标号。子程序最后由 RET 指令返回调用程序。这个子程序是将 1024 个数据由 0 通道输入并转送到数据存储器 400H 地址。重复指令 RPTK 将它的次一条指令重复执行其操作数指定的次数。RPTK 的操作数最大值为 255，因而输入 1024 个数据需用 4 次 RPTK 指令。IN 指令的操作数是端口序号，可写为 0 或 PA0。

LIST3:子程序 INPUT

```
        IDT     'INPUT'
        TITL    'DATA - INPUT'
        DEF     INPUT           ;说明子程序名为 INPUT
INPUT   LARP    AR1             ;使用辅助寄存器 AR1
        LRLK    AR1, >400       ;400H→AR1(>表示 16 进制)
        RPTK    255             ;下面的 IN 指令重复执行 256 次
        IN      * +, PA0        ;从 0 端口输入数据到地址 AR1
        RPTK    255
        IN      * +, PA0
        RPTK    255
        IN      * +, PA0
        RPTK    255
        IN      * +, PA0
        RET
        END
```

LIST4 是调用子程序的程序例。与 LIST3 的程序相比,仅增加了两个语句。应注意,TMS32020 中调用子程序的指令是 CALL,调用子程序名必须和伪指令 REF 定义的子程序名相同。此例中子程序名为 INPUT。

LIST4:子程序调用例

```
        IDT     'EMIN'
        TITL    'INITIALIZED EXAMPLE'
        B       START
        REF     INPUT           ;子程序名 INPUT
        RORG    $ +30
*       INITIALIZED SECTION
START   SOVM                    ;OVERFLOW - MODE
        SSXM                    ;SIGN EXTENSION
        LDPK    0               ;DATA PAGE 0.
        SPM     1               ;ONE BIT SHIFT ON P - REG OUTPUT
        LARP    AR1             ;USE AUX REG
        CALL    INPUT           ;调用子程序 INPUT
FIN     B       FIN
        END
```

4. 存储器间数据块的传送

LIST5 是存储器间数据块传送的子程序例。它将滤波器系数从程序存储器传送到数据存

储器。

一般而言，象滤波器系数这样的常数，先由伪指令 DATA 定义在程序存储器，然后再传送到数据存储器。DATA 指令前使用无条件转移指令 B，将可执行指令无条件转移到标号 START。

LIST5：数据块传送程序：

```
            IDT         'LOWPAS'
            TITL        'LOWPASS - FILTER'
            DEF         LOWPAS
            UNL
LOWPAS      B           START
COEF        DATA        63948
            DATA        1037
            DATA        2173
            DATA        527
            DATA        63038
            DATA        64166
            DATA        6020
            DATA        13871
            DATA        13871
            DATA        6020
            DAT         64166
            DATA        63038
            DATA        527
            DATA        2173
            DATA        1037
DATA        63948
*
*
START       CNFD                    ;B0 块用做数据存储器
            LIST
            PAGE
            LARP    AR1
            LRLK    AR1, >200       ;200H→AR1
            RPTK    15              ;重复 16 次执行 BLKP 指令
            BLKP    COEF, * +       ;程序存储数据→数据存储器
            RET
            END
```

CNFD 指令定义数据存储器 B0 块为数据 RAM，BLKP 将数据从程序存储器块传送到数据

存储器块，这时标号为 COEF 开始的 16 个数据被传送到打头地址为 200H 到 210H 的数据存储器中。

5.卷积运算程序

卷积运算如 LIST6 所示。程序中用伪指令 EQU 定义标号。该程序中由于在主程序内指定页号为零，因而数据地址 96 用于参数变换，地址 97 存储子程序的变量。

```
          LIST6   卷积运算程序
          IDT     'CONVOL'
          TITL    'ANTI-ALAISING.FILTER'
          DEF     CONVOL
CONVOL    EQU     $
NUMBER    EQU     96              ;参数
WORK      EQU     97              ;工作文件
*
          CNFP
*
          LRLK    AR4,1008        ;卷积数据的最小地址
*
          LRLK    AR3,1024-1      ;数据数
          LRLK    AR0,1024        ;输入数据地址
          LRLK    AR2,>800        ;输出数据地址
*
          LARP    AR0
LOOP      LAC     *+,0,AR4        ;输入数据→Acc
          SACL    *               ;送往 1008
          LRLK    AR1,1023
          SAR     AR3,WORK        ;奇偶校验
*
          MPYK    0               ;清除
          ZAC
*
          LARP    AR1             ;卷积运算
          RPTK    15
          MACD    >FF00,*-
*
          APAC                    ;P 寄存器内容→Acc
          BIT     WORK,>F
          BBZ     DEC
          LARP    AR2
          SACH    *+
```

```
*
DEC     LARP    AR3
        BANZ    LOOP,*-,AR0
*
        CNFD
        BV      OVER
        B       FIN
OVER    LALK    >7FFF
        SACL    NUMBER
FIN     RET
        END
```

CNFP 指令说明将数据存储器的 B0 块用做程序存储器，在 B0 块内已存储有滤波器的系数。在积加运算中必须这样对 CNFP 进行说明。

卷积运算的过程如图 3-23 所示，即程序存储器的地址 FF00H 的数与数据存储器地址 3FFH、FF01H 与 3FEH，……的数求其积和。

求得积和后，数据存储器的输入数据移行到下一个地址，地址 3F0H 内输入新的数据后，再求其积和从而实现卷积运算。

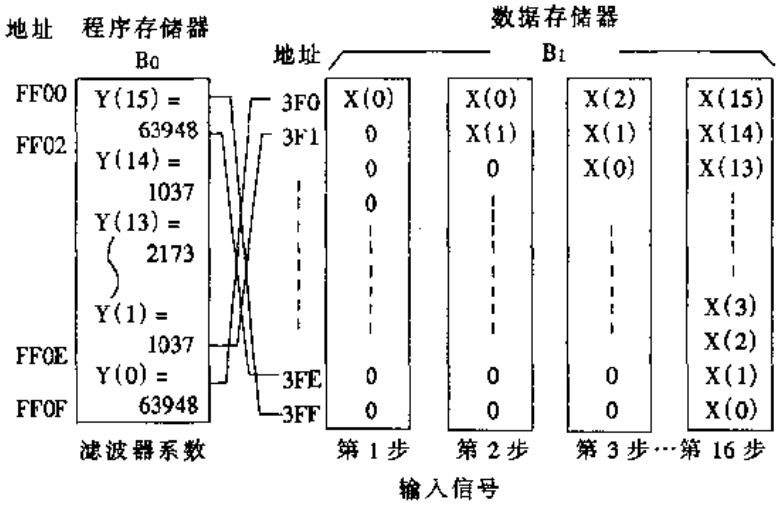


图 3-25 卷积运算

6. FIR 滤波器主程序

LIST7 是 FIR 滤波器的主程序例。这是一个通用程序，即使滤波器的参数变化，也只需修改几行程序便能正常运行。数据个数固定为 1024 个，也可以稍加修改。程序中从 0 号端口输入信号，确认工作在滤波器方式。程序中调用 INPUT, LOWPAS, CONVOL 三个子程序，其中 CONVOL 为卷积子程序。

```
        LIST7:滤波器主程序
    IDT     'EMAIN'
    TITL    'CQ.PUB CO.'
    REF     INPUT,CONVOL,LOWPAS
```

```
        B       START
NUMBER EQU      96
        RORG    $ + 30

*       INITIALIZED SECTION
START   SOVM                    ;OVERFLOW - MODE
        SSXM                    ;SIGN EXTENSION
        LDPK    0               ;DATA PAGE 0.
        SPM     0               ;0 BIT SHIFT ON P - REG OUTPUT
        LARP    AR1             ;USE AUX REG

HIGH    BIOZ    NEXT            ;BIO 为 L 时转入标号 NEXT
        B       HIGH
NEXT    CALL    INPUT
        CALL    LOWPAS
        LALK    0               ;清 96 地址
        SACL    NUMBER
        CALL    CONVOL
        BV      OVERFW          ;如有溢出转标号 OVERFW
FIN     B       FIN
OVERFW  OUT     NUMBER,PAI
        B       FIN
        END
```

该程序中还使用了 BIOZ 指令。当 TMS32020 的外部端子$\overline{BIO}$置"L"前做无限循环。一旦$\overline{BIO}$置"L"时，则无条件转移到 BIOZ 指令操作数为标号的指令并运行之。

3.6 TMS320C25 处理器

TMS320C25 的结构是在 TMS32020 的基础上开发的，它通过更快的指令周期及改进的附加功能增加了 DSP 算法功能。TMS320C25 的目标代码与 TMS32020 的目标代码完全兼容，所以以 TMS32020 有程序不需任何修改就可在 TMS320C25 上运行。

TMS320C25 有 100ns 和 125ns 两种指令周期形式，通常运用 100ns 指令周期方式，这种方式比 TM2020 工作快一倍。TMS320C25 采用 1.8μmCMOS 工艺，时钟频率有 40MH$_z$ 和 32MH$_z$ 两种。

TMS320C25 指令系统有 133 条指令，比 TMS32020 增加了 24 条指令。由于指令中增加了位反转寻址模式，大大加快了 FFT(基 2)运算速度。

硬件方面，与 TMS32020 一样，它的寻址空间依然是 128K 字。但在片内增加了 4K 字的 ROM，同时具有并发 DMA 功能，因而在执行片内程序仅读内数据时，可将外部总线全部交出供 DMA 使用。由于采用了 CMOS 工艺，在芯片不工作时，能以低功能耗状态，维持片内信息不丢失。TMS320C25 的辅助寄存器增加到 8 个，堆栈深度增加到 8 级，除了保持原有的接口外，它

的串行口具有双缓冲功能，在多处理器工作时，它还具有同步功能。

一、硬件结构

MS320C25 同样是一种采用哈佛结构的数字信号处理器(如图 3－26)。它也是围绕两条主要总线即程序总线和数据总线构成的。程序总线传送来自程序存储器的指令码和立即数。数据总线将内部单元联到数据 RAM。程序总线和数据总线把来自片内数据 RAM 和内外程序存储器的数据在一个周期内送到乘法器以进行乘法—累加操作。TMS320C25 有一个高级的并行结构。这种并行结构保证了功能极强的一组算术操作、逻辑操作及位控制操作能够在一个机器周期内完成。

TMS320C25 为 68 脚塑料封装。表 3－6 列出了它的信号、信号状态及其功能。

表 3－6　TMS320C25 信号说明

信　　号	I/O/Z	说　　明
Vcc	I	5V 电源
Vss	I	接地
X1	O	内部振荡器输出
X2/CLKIN	I	对内部振荡器的输入，来自晶体或外时钟
CLKOUT1	O	主时钟输出信号(4 分频晶体或 CLKIN)
CLKOUT2	O	次时钟输出信号
D15—D0	I/O/Z	16 位数据总线，D15(MSB)～D0(LSB)，程序、数据和 I/O 空间均分路传送
A15—A0	O/Z	16 位地址总线，A15(MSB)～D0(LSB)
$\overline{PS}$，$\overline{DS}$，$\overline{IS}$	O/Z	程序、数据和 I/O 空间的选择信号
R/$\overline{W}$	O/Z	读/写信号
$\overline{STRB}$	O/Z	选通信号
$\overline{RS}$	I	复位信号
$\overline{INT2}$～$\overline{INT0}$	I	外部用户的中断输入
MP/$\overline{MC}$	I	微处理机/微计算机方式选择
$\overline{MSC}$	O	微状态完成信号
TACK	O	中断应答信号
READY	I	数据准备输入，由外部逻辑置位，当用慢速设备时，它表示当前总线传输已经完成
$\overline{BR}$	O	总线请求信号，当 TMS320C25 需要对外部全局数据存储器空间进行访问时被置位
$\overline{XF}$	O	外部标志输出(锁存软件可编程信号)
$\overline{HOLD}$	I	占据输入；置位时，TMS320C25 进入空闲方式并将数据线、地址线及控制线均置成高阻状态
$\overline{HOLDA}$	O	占据应答信号
$\overline{BIO}$	I	分支控制输入，由 BIOZ 指令转态
DR	I	串行数据接收输入
CLKR	I	串行口数据输入用的时钟信号
FSR	I	接受输入用的帧同步脉冲
DX	O/Z	串行数据传送输出
FSX	I/O/Z	传输用的帧同步脉冲，既可用于输入，也可用于输出
CLKX	I	串行口传送用的时钟信号

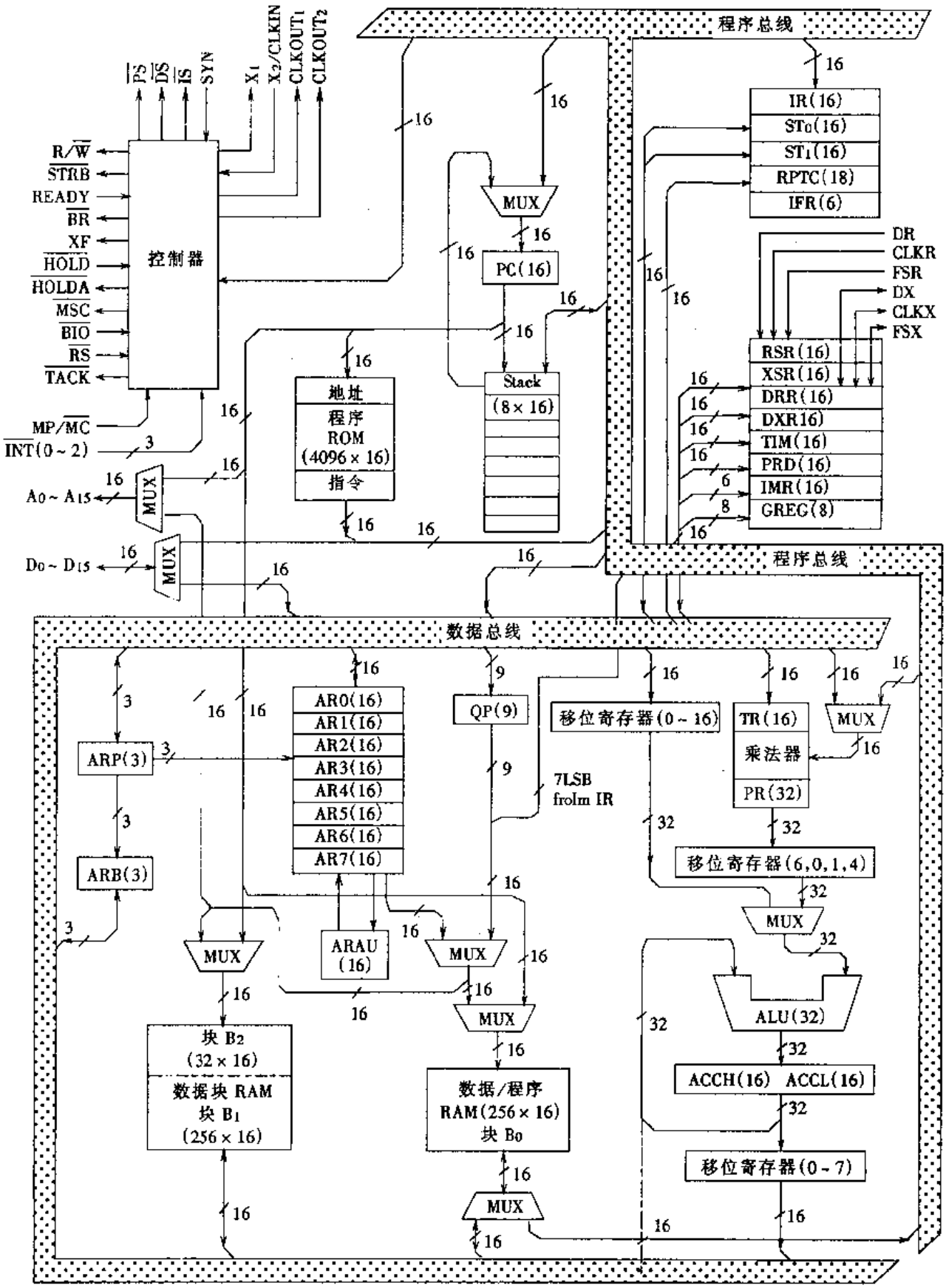


图 3－26　TMS320C25 结构框图

TMS320C25 实现了单个累加器的哈佛结构，程序存储器和地址存储器分配在不同的地址空间。这样取指令和执行指令可同时进行。在外部，程序和数据存储器空间在同一总线上分路以使这两个空间的地址范围最大并使元件的引脚最少；在内部，它拥有程序总线和数据总线两个独立的总线，可使程序和数据全速执行，从而得到最大的处理能力。独立的流水线用来进行指令译码。TMS320C25 大部分指令都在一个机器周期内完成。利用 READY 信号可以同慢速的片外存储器及外围设备进行通信。

二、存储器的控制

TMS320C25 提供了共 544(16 位)字的片内数据存储器 RAM 的分块及控制与 TMS32020 相同，如图 3－27 所示，除此之外，TMS320C25 有一个 4096 字的片内 ROM。用户程序由工厂掩膜在 ROM 里。用微处理机/微计算机选择引脚 MP/MC，TMS320C25 可将 ROM 分配到存储器空间以内或以外。TMS320C25 提供了直接编址 64K 字的片外存储器空间，在这个空间内，高速存储器程序可全速执行，慢速存储器程序需插入等待状态。

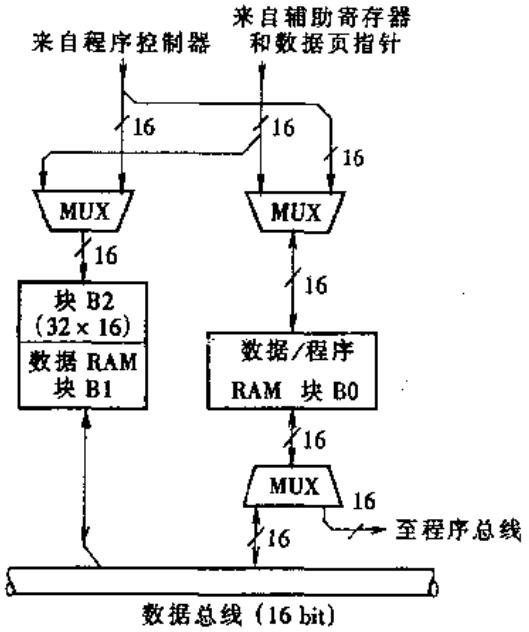


图 3－27 TMS320C25 存储器

TMS320C25 对程序存储器、数据存储器及 I/O 分别提供了独立的地址空间。除了 B0、B1 及 B2 以外，数据存储器的分配还包括分配给存储器的六个外围设备寄存器和备用单元。六个外围设备寄存器是二个串行口寄存器(DRR 及 DXR)、一个中断掩膜寄存器(IMR)和一个全局存储器的分配寄存器(GREG)。备用单元不准存取。

TMS320C25 有一个包括 6 八个辅助(AR0～AR7)寄存器组，用来对数据存储器进行间接寻址，或做为暂时存储。这些寄存器可直接寻址或间接寻址。这些寄存器和 ARP 都可以由数据存储器指令中的立即操作数装入，寄存器的内容也可以存入数据存储器。辅助寄存器与辅助寄存器算术单元(ARAU)相联，如图 3－28。当数据存储器单元寻址时，ARAU 可对当前辅助寄存器进行自动变址。因此，对信息表的访问无需 CALU 进行地址控制。ARAU 还可做为附加的通用算术单元。进行 16 位无符号算术运算，完成 32 位 2 的补码的算术运算。TMS320C25 还

具有几种取决于 ARP 指定的辅助寄存器与 AR0 的比较的分支指令。

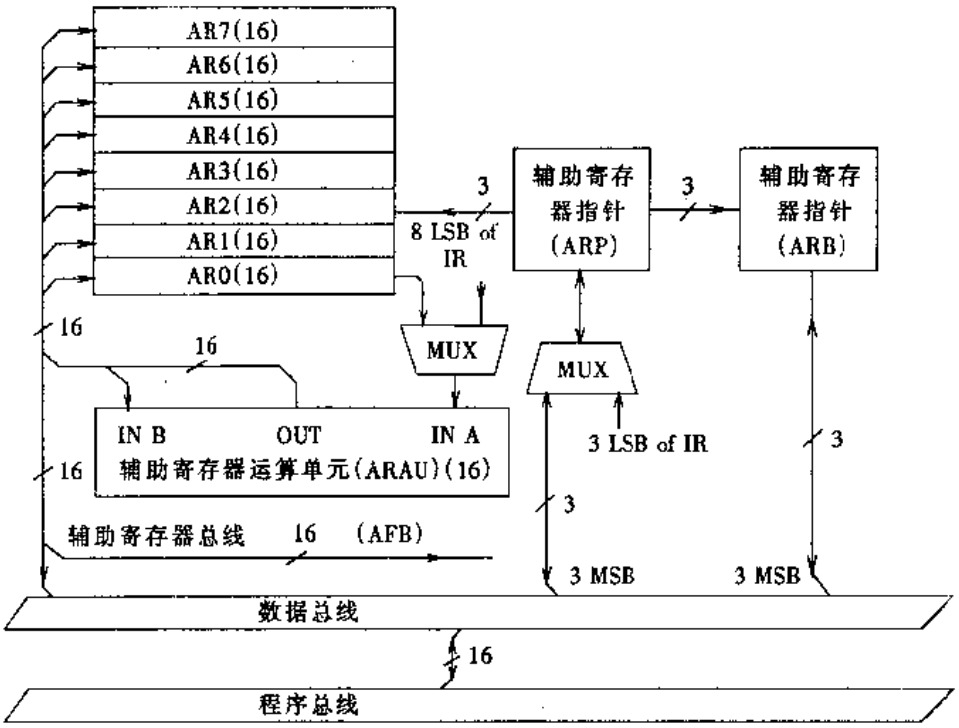


图 3-28 辅助寄存器及其算术单元

TMS320C25 有一个 16 位的程序计数器(PC)和一个用来存放 PC 的八级硬件堆栈。程序计数器通常总是访问程序存储器,(片内或片外)。累加器的内容装入 PC 可完成计算转移操作。为处理子程序嵌套和八级中断,TMS320C25 备有压入和弹出指令,可在数据存储器里建立堆栈,可将栈顶存入数据存储器或把它装入累加器。

本地存储器接口由一个 16 位并行数据总线(D15 ~ D0),一个 16 位程序地址总线(A15 ~ A0)、数据存储器、程序存储器或 I/O 空间的三只选择引脚($\overline{PS}$、$\overline{DS}$及$\overline{IS}$)和各种系统控制信号组成。R/$\overline{W}$ 信号控制数据传输的方向,$\overline{ATRB}$提供控制传输的定时信号。当使用片内程序 RAM、ROM 或高速外部程序存储器时,TMS320C25 可全速无需等待状态。使用片外慢速存储器时要利用 READY 信号产生等待状态。

利用$\overline{HOLD}$和$\overline{HOLDA}$信号,TMS320C25 可实现其对外部程序存储器或数据存储器的 DMA。$\overline{HOLD}$置低电平时,其它处理机可完全控制外部存储器。这时,TMS320C25 把地址线、数据线及控制线全部置成高阻状态。

三、中央算术逻辑单元(CALU)

中央算术逻辑单元中央算术逻辑单元框图如图 3-29 所示。其构成单元和 ALU 指令的执行步骤与 TMS32020 大体相同。同样为便于数据据存入数据存储器,32 位的累加器亦分成两段:ACCH(累加器高段)和 ACCL(累加器低段),每段 16 位。但累加器高段的附加移位器进行出 0 ~ 7 位的左移。左移是在数据送到数据总线进行存储时完成的。这时,累加器的内容保持不变。累加器内部也能左移或右移一位(SFL 或 SFR 指令)并可将进位循环(ROL 和 ROR 指

令)。为了完成高精度算术运算 TMS320C25 设有进位位。

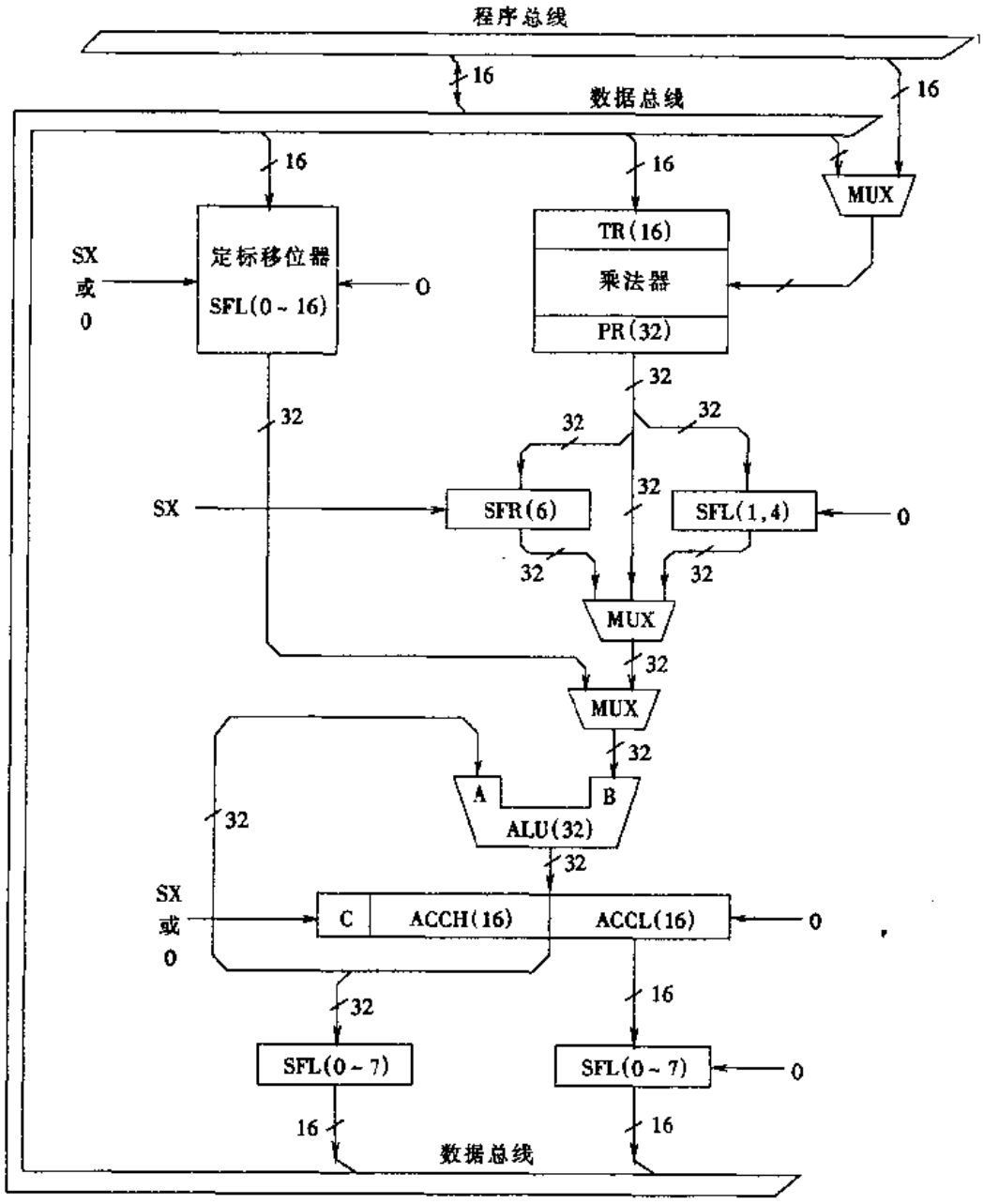


图 3-29 中央逻辑单元

TMS320C25 用一个 2 的补码 16×16 位硬件乘法器,在一个机器周期内计算出 32 位的乘积。与 TMS32020 一样有两个寄存器与乘法器相联。乘积寄存器的输出可左移 1 位或 4 位,主要用于分数的算术运算及验证分数乘积。乘积寄存器的输出还可以右移 6 位,从而可执行 128 次连续乘法——累加操作而不致溢出。无符号乘法器(MPYU)是一种扩展精度的乘法,MAC 和 MACD 两条乘法——累加指令充分利用乘法器的计算宽度,两个操作数可同时处理。利用重复指令(RPT 和 RPTK)可将上述操作变成单周期的乘法——累加。平方加(SQRA)和平方减

(SQRS)指令把同一个值送给乘法器的两个输入端。

与 TMS332020 一样,TMS320C25 支持浮点运算。规格化(NORM)指令通过执行左移使累加器中的定点数进行规格化。按 TR 移位装入累加器(LACT)指令利用移位器将规格化的浮点数变成定点数。ADDT 和 SUBT 指令按 TR 指定的移位可对 16 位尾数和 4 位指数的浮点数进行运算。

TMS320C25 有几条根据 ALU 状态来转移的分支指令。还有位测试(BIT 和 BITT)指令,它们不影响累加器的状态,仅测试数据存储器数据字的某特定位。

四、系统控制

TMS320C25 通过片内定时器、重复计数器、三个用户外部掩膜中断,由串行口操作或定时器产生的内部中断及外部复位信号提供控制操作。分配给存储器的 16 位定时(TIM)寄存器由一个被内部时钟连续钟控的逆计数器组成。定时器一旦减到零就出现定时中断(TINT)。在定时器到零后的下一个周期内,周期(PRD)寄存器的内容重新装入定时器,所以中断是可编程的,而且中断的出现是以规则的(PRD + 1) CLKOUT1 个周期为间隔。TMS320C25 设有重复指令可使一条指令重复最多达 256 次。重复计数器(RPTC)在 RPT 指令时由数据存储器的内容决定,在 RPTK 指令时由立即数值决定。正常时是多周期的指令,使用重复指令时流水作业,并且变成了高效率的单周期指令。TMS320C25 设置了三个外部可屏蔽用户中断($\overline{INT2}$ ~ $\overline{INT0}$),功能与 TMS32020 相同。ST0 和 ST1 两个状态寄存器的内容可用指令存到数据存储器或由数据存储器装入。这些操作可在中断期间或子程序调用期间完成。

五、I/O 接口

与 TMS32020 类似,TMS320C25 中的 I/O 处理与存储器处理完全相同。片内串行口用来与串行装置及串行系统进行直接通讯。两只串行口寄存器(数据传送/接收移位寄存器)已分配给存储器,可以 8 位字节或 16 位字节两种方式工作。帧传送同步脉冲可在内部产生也可在外部产生。串行口的最大速度为 5MHz。TMS320C25 串行口设两个缓冲区,可同时进行接收和传送。CLKR/CLKX 的最小频率为零。设有帧同步方式位(FSM)。

六、指令系统

TMS320C25 指令系统的码完全与 TMS32020 兼容。为获得最大处理能力,当前指令正在执行时就提前取下一条指令。

TMS320C25 指令系统有三种存储器寻址方式:直接寻址、间接寻址和立即寻址。直接寻址和间接寻址可用来访问数据存储器。直接寻址时,7 位的指令字与 9 位的数据存储器页指针(DP)的内容相联构成 16 位的数据存储器地址。用 128 字的页长,DP 寄存器指向数据存储器 512 页中的一页,从而得到 64K 的数据存储空间。指令中的 7 位地址指向数据存储页内的特定单元。直接寻址可用于除 CALL、分支指令、立即操作数指令和无操作数指令以外的全部指令。八个辅助寄存器 AR0 ~ AR7 用于间接寻址式。指令中所用的数据地址放在这八个辅助寄存器中的一个内。辅助寄存器指示器(ARP)中分别装以 1、2……7 来对应 AR0 ~ AR7,用来选择指定的辅助寄存器。ARAU 在一个机器周期里完成 16 位无符号数的算术运算。

间接寻址有七种型式如表 3 - 7。全部变址操作均以原指令相同的周期在当前辅助寄存器里完成,但要给 ARP 一个新的值。位反转变址寻址方式对于基 2 FFT 程序很有用。

表 3－7　寻址方式

寻　址　方　式	操　　作
OPA	直接寻址
OP＊(,NARP)	间接寻址,AR 不变
OP＊＋(,NARP)	间接寻址,当前 AR 增加
OP＊－(,NARP)	间接寻址,当前 AR 减少
OP＊0＋(,NARP)	间接寻址,当前 AR 加上 AR0
OP＊0－(,NARP)	间接寻址,当前 AR 减去 AR0
OP＊BR0＋(,NARP)	间接寻址,当前 AR(求反进位扩展)加上 AR0
OP＊BR0－(,NARP)	间接寻址,当前 AR(求反进位扩展)减去 AR0

立即寻址方式中包括立即操作数。TMS320C25 有单字(8 位及 13 位常数)立即短指令和双字(16 位常数)的立即长指令。在立即长指令中指令操作码后面的字就是立即操作数。

七、TMS320C25 与外设连接

TMS320C25 可以和 PROM、EPROM 和静态 RAM 连接。根据速度、价格和功耗的限制,可以构成各种不同的电路。当速度和最大吞吐量均满足时,TMS320C25 可在无等待状态下直接与存储器接口。这时,存储器的存取在单周期内完成。对于速度较慢的存储器,可通过引入适当数量的等待状态,等待状态的个数,取决于存储器的存取时间。当需要一个或几个等待状态时,则 READY 输入在插入状态的周期内被强制为低电平。

TMS320C25 使用两个分离的存储器空间,即程序存储器空间和数据存储器空间。两者差别仅是一个使用$\overline{PS}$引脚,一个使用$\overline{DS}$引脚。而第三个空间是 I/O 空间,使用引脚$\overline{IS}$与外围设备相接。TMS320C25 和 PROM 直接连接的例如图 3－31 所示。图中,PROM 使用 TI 公司的 TBP 38 L－165－35,它是低功率 2K×8 位 PROM,接口时序如图 3－30 所示。

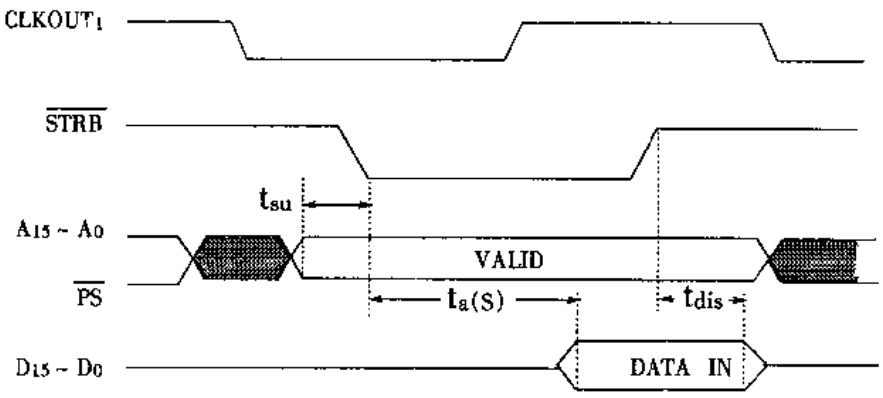


图 3－30　连接时序图

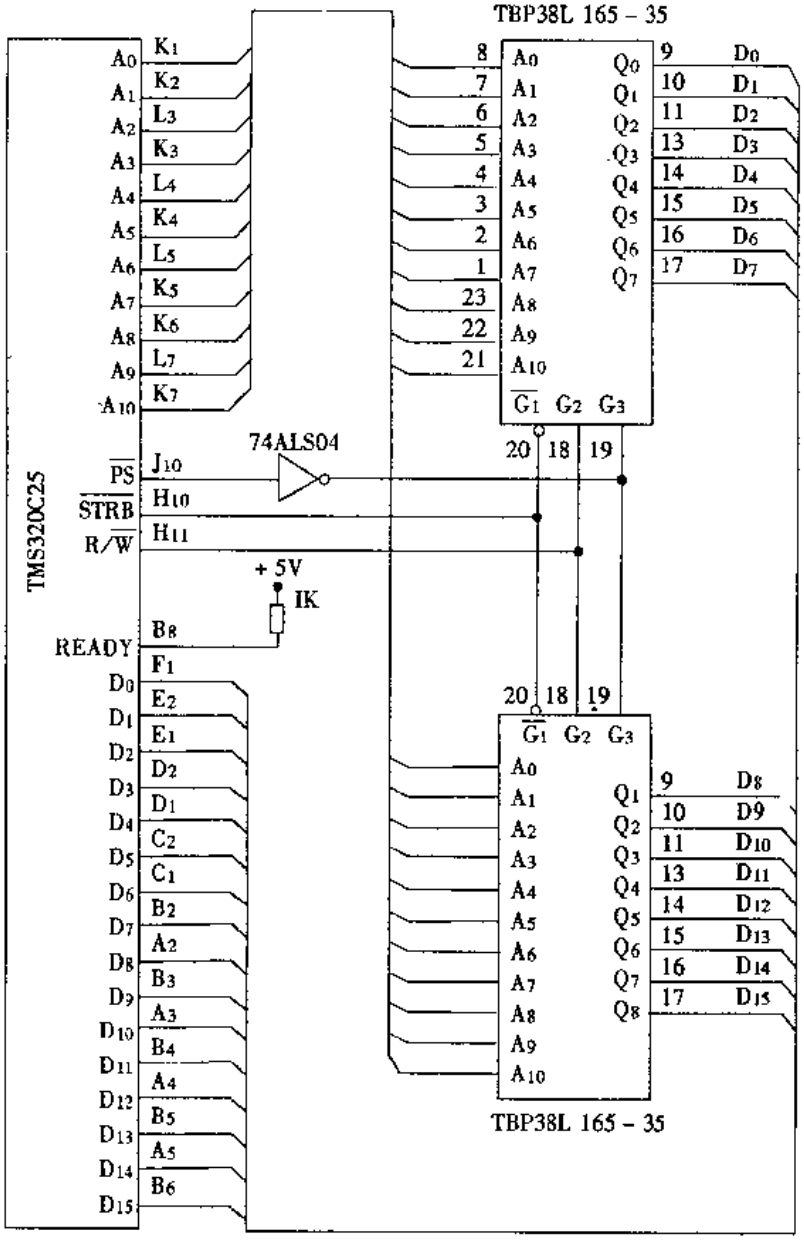


图 3 – 31　TMS320C25 与 PROM 连接

TMS320C25 要求数据在 STRB 变低不迟于 27ns 有效，TBP38L – 165 – 35 对地址有效的存取时间为 35ns，最大片选启动时间为 20ns，如表 3 – 8 所示。

表 3 – 8　为 TBL 38L – 165 – 35 直接与 TMS320C25 接口时的时间参数

项　　目	使用符号	数值
地址建立时间	tsu	13ns
TMP38L165 – 35 片选启动存取时间	ta(s)	20ns
TMP38L165 – 35 禁止时间	tdis	15ns

而 TMS320C25 在 STRB 变低之前，地址有效最小时间为是 13ns，因而在 STRB 变低之后，数

据在数据总线上的时间在 27ns 之内，完全满足 TMS320C25 的要求。

使用地址译码器的 PROM 和 TMS320C25 接口例如图 3－32 所示。采用这种结构，可满足 READY 定时的要求。本电路中，由地址译码器生成 READY 信号，而另一个地址译码器用于启动不同的存储单元。在该设计中，程序存储器空间的上半部份(32K)设置为无等待状态存储单元，下半部分存储单元是有一个或多个等待状态的存储器。其译码电路由 74AS20 四输入 NAND 门电路组成。

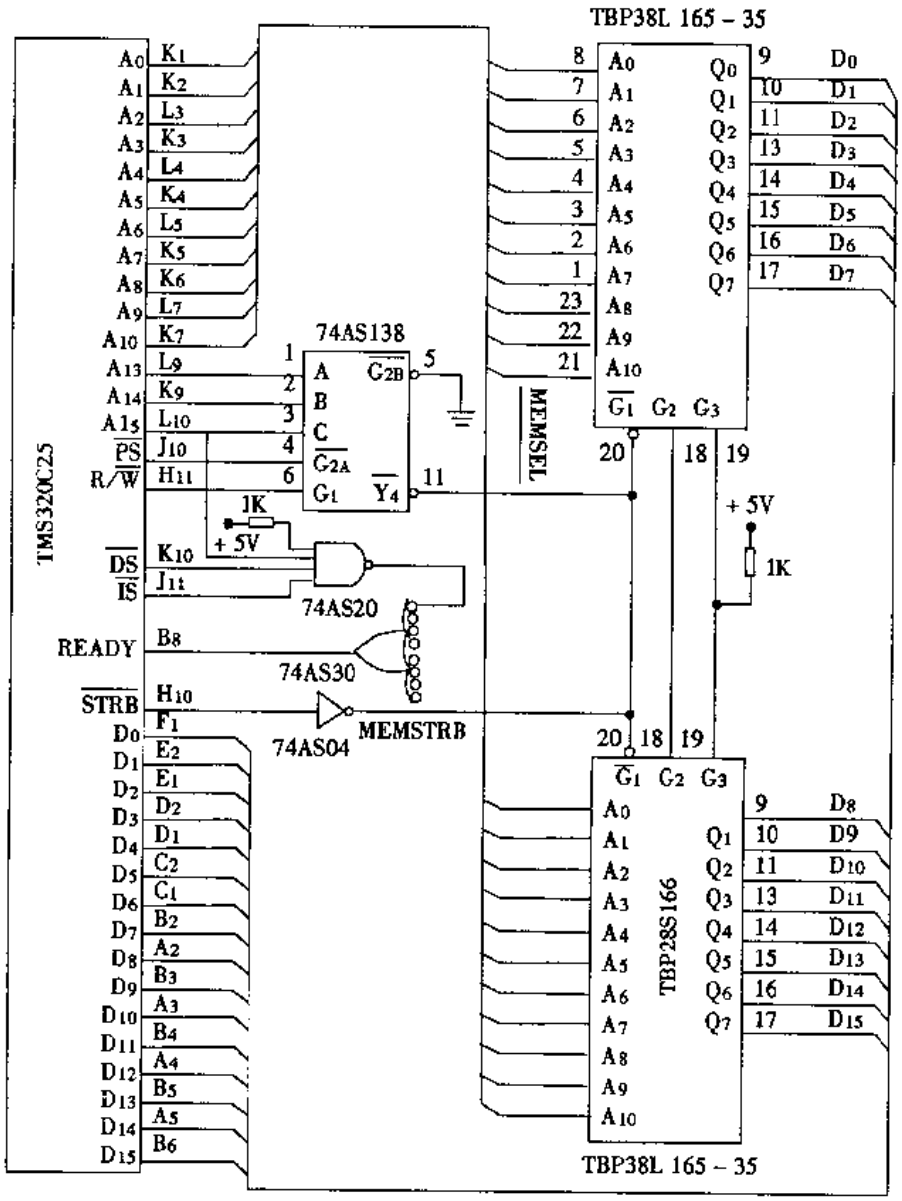


图 3－32　使用地址译码器的接口电路

地址译码器由 74L138 组成。该译码器将程序存储器分成 8 段，每段 8K 字，前四段(地址空间的低 32K)由 74AS138 的输出$\overline{Y1}$、$\overline{Y2}$、$\overline{Y3}$和$\overline{Y4}$启动，并做为具有一个或几个等待状态的存储单元，其余四段选择为无等待存储单元(第 5 段为 TBP38L165，起始地址＞8000)。应该注意，在图 3－32 中，R/W 信号用于启动 74AS138，以避免在 PROM 写入时出现总线冲突。图 3－33

为图 3－32 电路的时序，在地址有效之后 10nsREADY 信号变高电平。

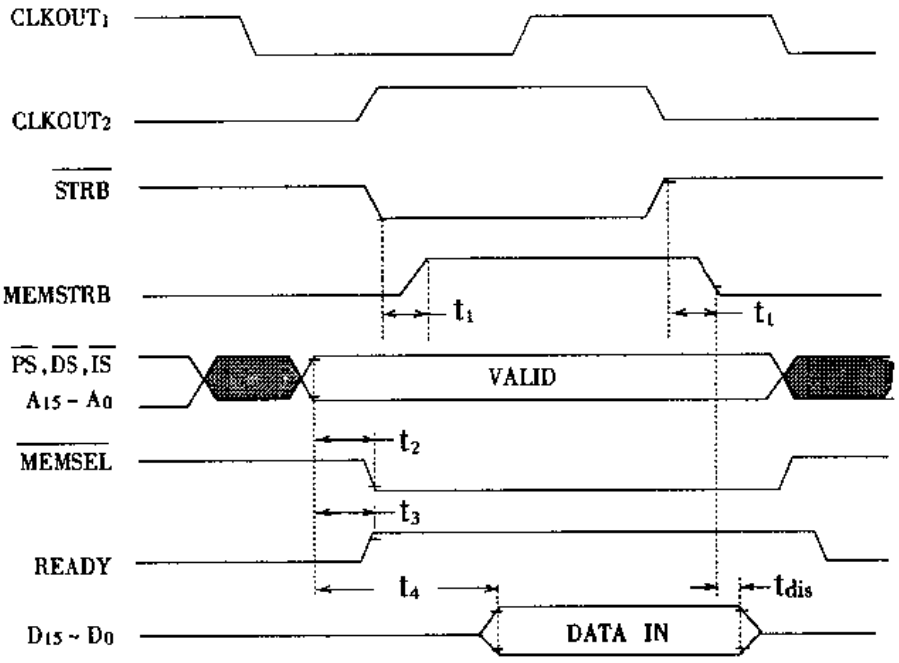


图 3－33 TMS320C25 与 PROM 连接时序

八、系统配置

1.单处理器系统

TMS320C25 的灵活性使得它可以适用于各种系统：

(1)单独系统，即单处理器，用 4K 字片内 ROM 和 544 字片内 RAM。

(2)共享全局数据存储器的并行多处理器系统。

(3)采用接口控制信号的主机——外设协处理系统，通常成为主从系统。

一个单独的硬件系统包括 16 位并行数据总线、16 位程序地址总线、3 条存储空间选择线、以及各种系统控制信号。图 3－34 在最小处理系统的基础上扩充了外部数据 RAM 和 PROM 或 EPROM。READY 信号允许产生等待状态以便于与慢速的外存储器通讯。所有存储器和 I/O 设备均由 TMS320C25 直接控制，折衷情况下所需的外部硬件最少。折衷系统主要利用单处理器内部强化的并行措施来提高速度，它的优点是硬件简单，成本较低。

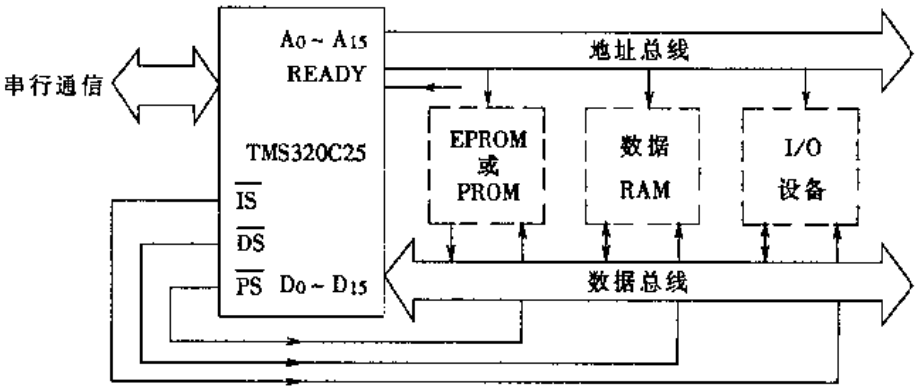


图 3－34 单处理器系统

TMS320C25 的串行口可以用来与串行设备(如编码译码器、串行 ADC 接口),以及在处理器之间进行串行通信,例如将两个最小系统通过串行口连接而成为一个双处理器系统,所表系统处理器间通信较慢,且只在数据一级相互作用,不与程序的正常执行冲突,构成所谓松散耦合系统或间接耦合系统。

在多处理器应用中,TMS320C25 能够分配全局存储空间并通过 BR(总线请求)和 READY 控制信号与存储空间通信。8 位的存储器映象全局存储器分配寄存器(GREG)把多达 32K 字的 TMS320C25 数据存储器设定为全局外部存储器,寄存器的内容决定全局存储空间的大小。如果当前指令对该空间的操作数寻址,BR 有效以请求总线控制。存储器周期的长度由 READY 线控制。

2.并行处理系统

图 3-35 示出了一个采用全局存储器的并行处理系统。两片 TMS320C25 在执行各自程序存储器中的程序的同时,共享全局数据存储器。全局存储器的优先权通过 XF 和 BIO 脚由软件管理。XF 脚作为一个外部标志,BIO 脚则可以用决定于 BIO 状态的转移指令(BIOZ)查询。

这种系统的相互作用虽仍可能在数据一级,但已具备在任务级或作业级并行的条件,是非常典型的紧密耦合系统,或又叫直接耦合系统。

介于直接耦合系统与间接耦合系统之间的是主/从式系统。

应用 TMS320C25 提供的 DMA 方式,通过 HOLD/HOLDA(保持/保持响应)和 INT/INTA(中断/中断响应)信号可以构成一种主/从式并行系统。

由于 TMS320C25 在保持期间,DSP 芯片内可利用片内程序/数据 RAM 正常运行,从而使程序并行性得到大大加强。所以,这种结构更接近于直接耦合系统。

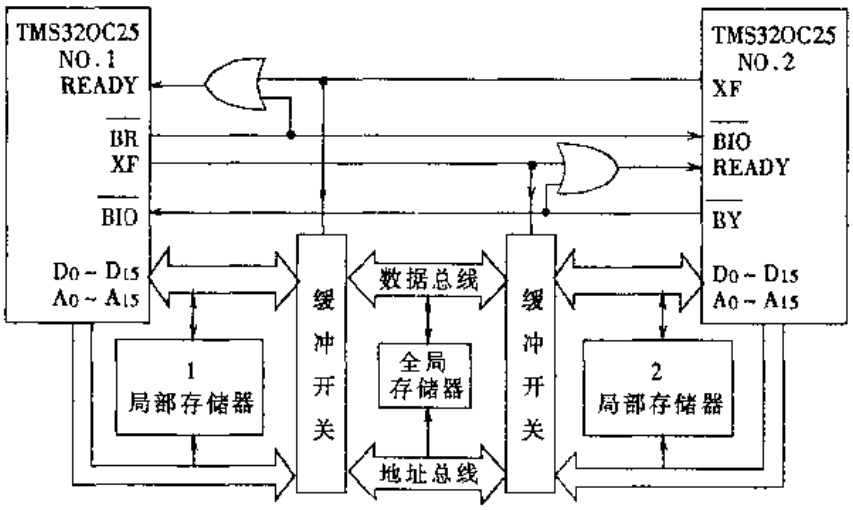


图 3-35 全局存储器并行处理系统

3. 主从系统

图 3 - 36 示出了一种主从处理系统，它使用了接口控制信号 $\overline{\text{HOLD}}/\overline{\text{HOLDA}}$。将 TMS320C25 用于多处理器系统的一个方便之处在于它能够与外部信号同步。

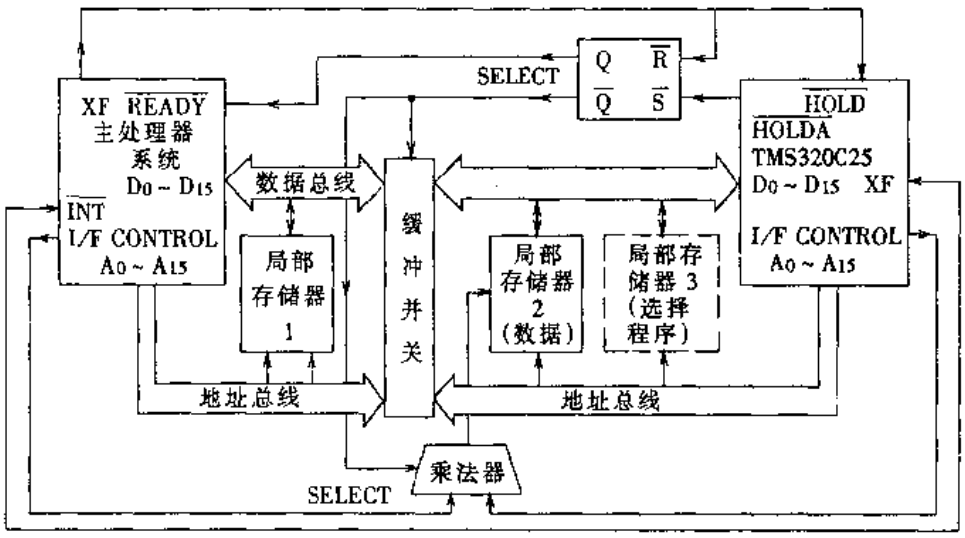


图 3 - 36 采用接口控制信号的主从处理系统

$\overline{\text{SYNC}}$引脚可以使两片或更多的 TMS320C25 的内部时钟相同步，由于各处理器按同一内部时钟相位工作，于是所有的外部信号也被同步，从而不需要外部逻辑用于同步处理之间的信号。

第四章　TMS320C50 及其它常用 DSP

4.1　TMS320C5X 处理器

一、TMS320C5X 概述

TMS320C5X 是 TI 公司开发的支持 16 位定点运算的数字信号处理器，该系列产品包括'C50、'C51、'C53，工艺设计基于'C25，采用静态 CMOS 集成技术。TMS320C5X 有大的片内存储器及一个高速的专用指令集，执行速度可达 28.6MIPS。TMS320C5X 处理器的主要性能如下：

- 单周期定点运算执行时间为 35/50ns(28.6/20MIPS)
- 向上兼容'C1X 和'C2X 的源代码
- 基于 RAM 的指令('C50)
- 基于 ROM 的指令('C51)
- 9K × 16 位单周期在片程序/数据 RAM('C50)
- 1K × 16 位单周期在片程序/数据 RAM('C51)
- 3K × 16 位单周期在片程序/数据 RAM('C53)
- 2K × 16 位单周期在片引导 ROM('C50)
- 8K × 16 位单周期在片程序 ROM('C51)
- 16K × 16 位单周期在片程序 ROM('C53)
- 1056 × 16 位双向可存取在片数据的 RAM
- 224K × 16 位的最大外部可寻址空间
- 32 位 ALU(算术逻辑单元)、32 位累加器(ACC)、32 位累加缓冲器(ACCB)
- 16 位并行逻辑单元 PLU
- 16 × 16 位并行乘法器
- 单周期乘法/加法指令
- 8 个辅助寄存器
- 8 级硬件堆栈
- 0 ~ 16 位的左右数据移位器和 64 位增量数据移位器
- 面向程序代码的单指令循环和块循环指令
- 两个间接寻址的循环缓冲器
- 块存储器移动指令
- 全双工同步串行口
- 分时多路存取(TDM)串行口
- 内部定时器
- 64K 并行 I/O 接口
- 16 个软件可编程等待周期产生器
- 面向当前外部 DMA 的保持指令(HOLD)

· 面向延时分支、调用和返回指令的四级流水线指令
· 变址寻址方式
· 位反转(码位倒置)寻址方式
· 1 比 1 分频时钟操作
· 在片时钟产生器
· 带有两种掉电方式的 5V CMOS 技术

二、TMS320C5X 硬件结构

TMS320C5X 芯片共有 132 个引脚,如图 4－1 所示。

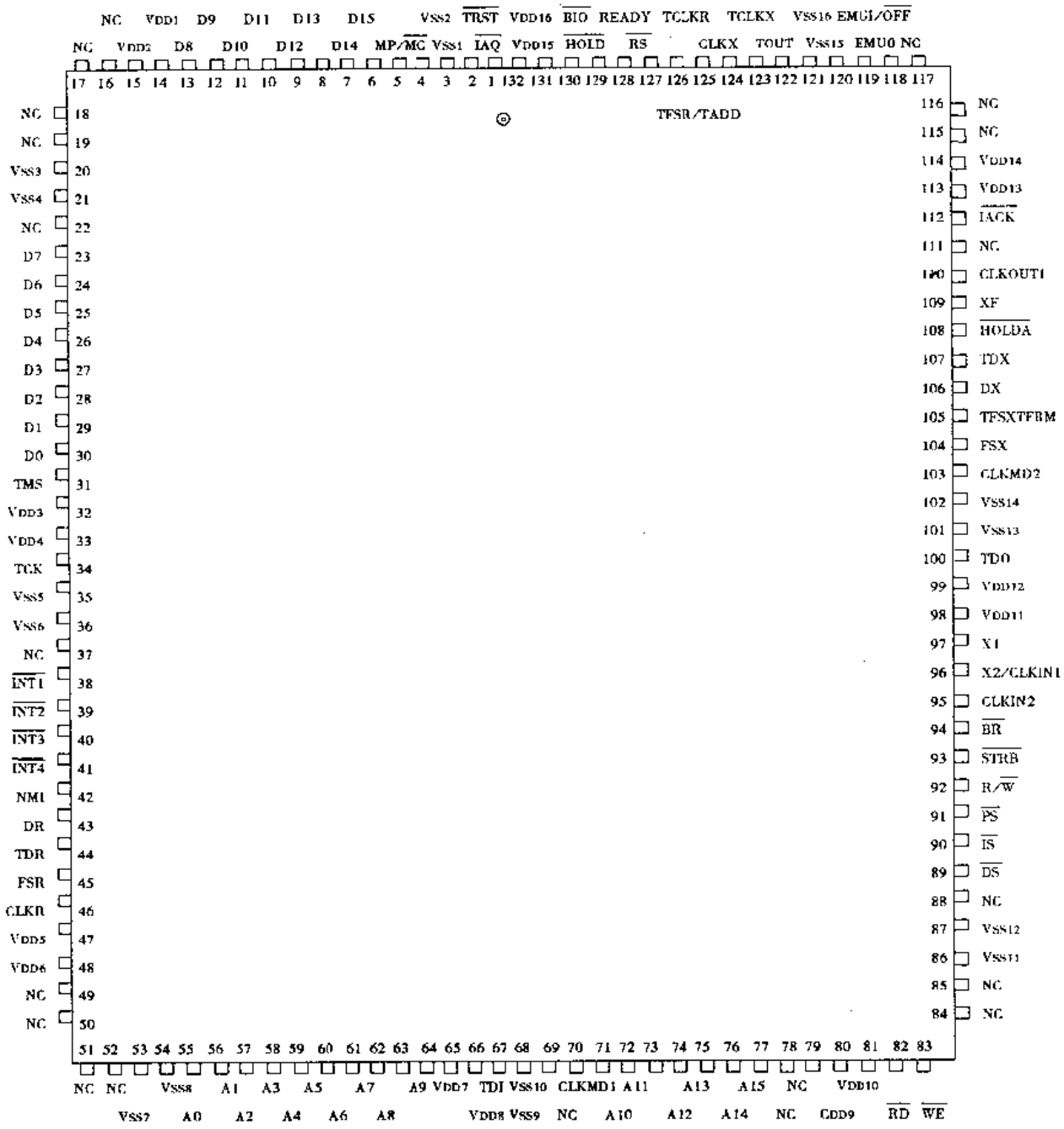


图 4－1　TMS320C5X 引脚排列

表 4－1　TMS320C5X 引脚信号说明

信　号	引脚数	类型	功　能
地 址 和 数 据 总 线			
A15～0	16	I/O/Z	并行地址总线 A15(MSB)到 A0(LSB)
D15～0	16	I/O/Z	并行数据总线 D15(MSB)到 D0(LSB)
存 储 器 控 制 信 号			
$\overline{DS}$	1	O/Z	数据空间选择信号
$\overline{PS}$	1	O/Z	程序空间选择信号
$\overline{IS}$	1	O/Z	I/O 空间选择信号
READY	1	I	数据准备就绪输入
R/$\overline{W}$	1	I/O/Z	读/写信号
$\overline{STRB}$	1	I/O/Z	选通信号
$\overline{RD}$	1	O/Z	读允许
$\overline{WE}$	1	O/Z	写允许
多 路 处 理 信 号			
$\overline{HOLD}$	1	I	保持输入
$\overline{HOLDA}$	1	O/Z	保持响应信号
$\overline{BR}$	1	I/O/Z	总线请求信号
$\overline{IAQ}$	1	O/Z	指令采集信号
$\overline{BIO}$	1	I	分支控制输入
XF	1	O/Z	外部标志输出
$\overline{IACK}$	1	O/Z	中断响应信号
初始化、中断和复位信号			
$\overline{INT4}$～$\overline{INT1}$	4	I	外部用户中断输入
$\overline{NMI}$	1	I	非屏蔽中断
$\overline{RS}$	1	I	复位输入
MP/$\overline{MC}$	1	I	微处理器/微计算机方式选择
晶 振 时 钟 信 号			
CLKOUT1	1	O/Z	主时钟输出信号
CLKMD1－2	2	I	时钟方式
X2/CLKIN1	1	I	晶体输入到内部振荡器
X1	1	O	内部振荡器输出到晶体
CLKIN2	1	I	1 比 1 分频输入时钟
TOUT	1	O	定时器输出
串 行 口 信 号			

CLKR TCLKR	1 1	I I	接收时钟输入，记录从 DR/TDR 引脚到 RSR 的数据的外部时钟信号
CLKX TCLKX	1 1	I/O/Z I/O/Z	发送时钟，记录从 DR/TDR 到 DX/TDX 的时钟信号
DR TDR	1 1	I I	串行数据接收输入，串行数据经 DR/TDR 引脚接收到 RSR
DX TDX	1 1	O/Z O/Z	串行口发送输出，串行数据从 XSR 经 DX/TDX 引脚发送
FSR TFSR/TADD	1 1	I I/O/Z	接收输入的帧同步脉冲
FSX TFSX/TFRM	1 1	I/O/Z I/O/Z	发送输入/输出的帧同步脉冲
检 测 信 号			
TCK	1	I	JTAG 检测时钟
TDI	1	I	JTAG 检测数据输入
TDO	1	O/Z	JTAG 检测数据输出
TMS	1	I	JTAG 检测方式选择
$\overline{\text{TRST}}$	1	I	JTAG 检测复位
EMUO	1	I/O/Z	仿真引脚 0
EMU1/OFF	1	I/O/Z	仿真引脚 1/禁止所有输出
保留	20	NC	保留引脚，不连接
电 源 引 脚			
V_{DD1-4}	4	S	数据总线电源
V_{DD5-6}	2	S	地址总线电源
V_{DD7-8}	2	S	输入和内部逻辑电源
V_{DD9-10}	2	S	地址总线电源
$V_{DD11-12}$	2	S	存储器控制信号电源
$V_{DD13-14}$	2	S	输入和内部逻辑电源
$V_{DD15-16}$	2	S	存储器控制信号电源
V_{SS1-2}	2	S	存储器控制信号地
V_{SS3-6}	4	S	数据总线地
V_{SS7-10}	4	S	地址总线地
$V_{SS11-12}$	2	S	存储器控制信号地
$V_{SS13-16}$	4	S	输入和内部逻辑地

各引脚的输入输出信号说明如表 4－1。图 4－2 为 TMS320C5X 的结构框图。

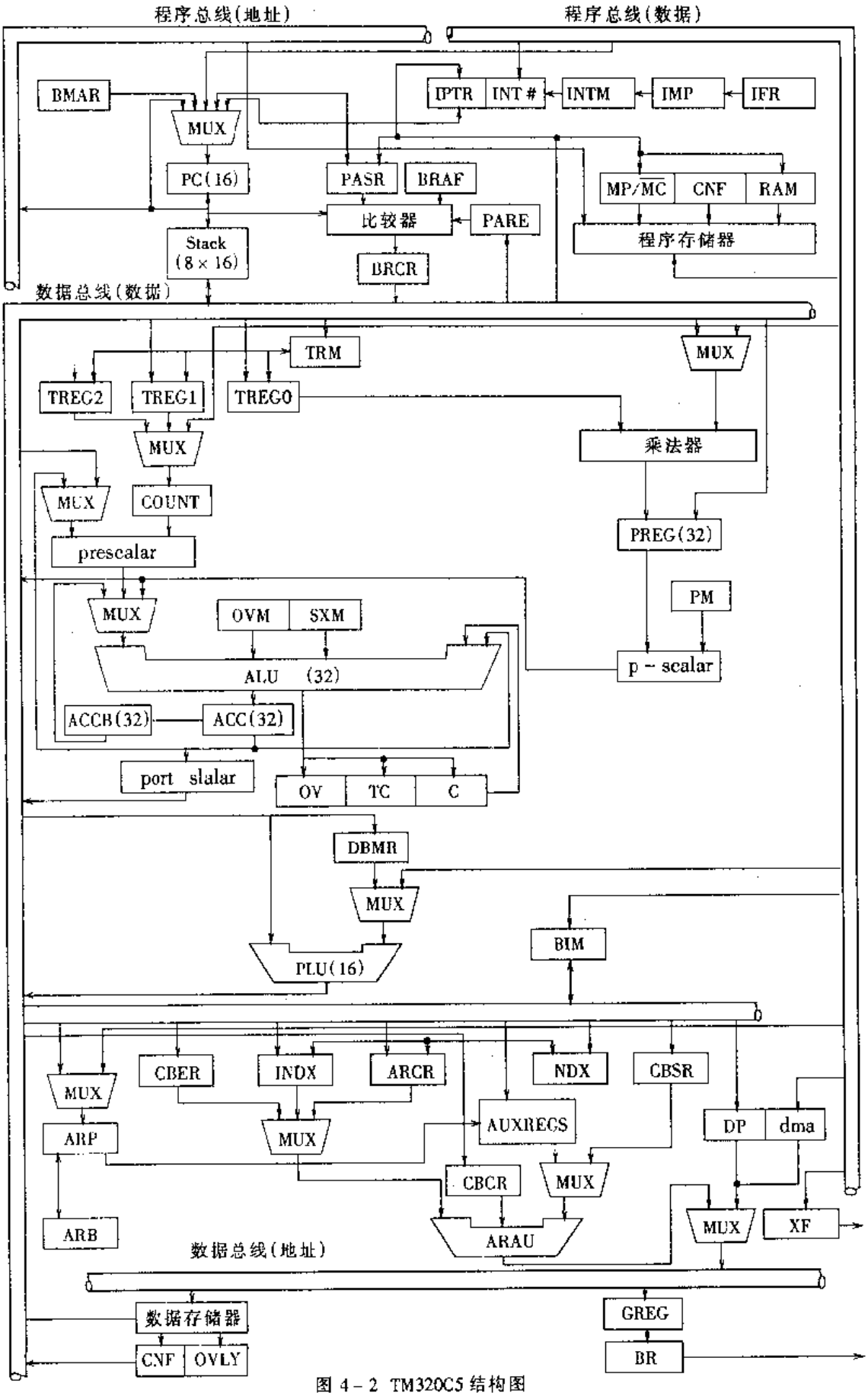


图 4 - 2 TM320C5 结构图

1. 存储器分配

TMS320C5X 是以增强型的哈佛结构为基础设计的，这种结构的多路存储空间可被三条并行总线存取，即同时存取程序和数据。这三条并行总线是：程序读/写总线(PAB)、数据读总线(DAB1)、数据写总线(DAB2)。'C5X 的全部存储器空间为 224K × 16 位字，分为 64K 程序、64K 局部数据、32K 全局数据和 64K 输入/输出接口四个专用的存储段，其中局部数据存储器存储指令所用的数据，全局数据存储器可以和系统内的其它处理器分享数据或用来提供附加数据，I/O 空间与片外存储器操作：取指令、读操作和写操作。

以 TMS320C50 为例，它有 2K 字的引导 ROM、9K 字的程序/数据 SARAM 和 1056 字的 DARAM。引导 ROM 位于程序空间的 0 地址，包括设备检验和引导代码，9K 字的 SARAM 块可被映射成程序或数据空间，并位于其它空间的 0800H 处，1056 字 DARAM 配置成三块：块 0(B0)占 512 字，位于局部数据存储空间的 0100H ~ 02FFH 或程序空间的 0FF00H ~ 0FFFFH 处，块 1(B1)占 512 个字，位于局部数据存储器的 0300H ~ 04FFH 处，块 2(B2)占 32 个字，位于局部数据存储器的 060H 处，如图 4 - 3 所示。

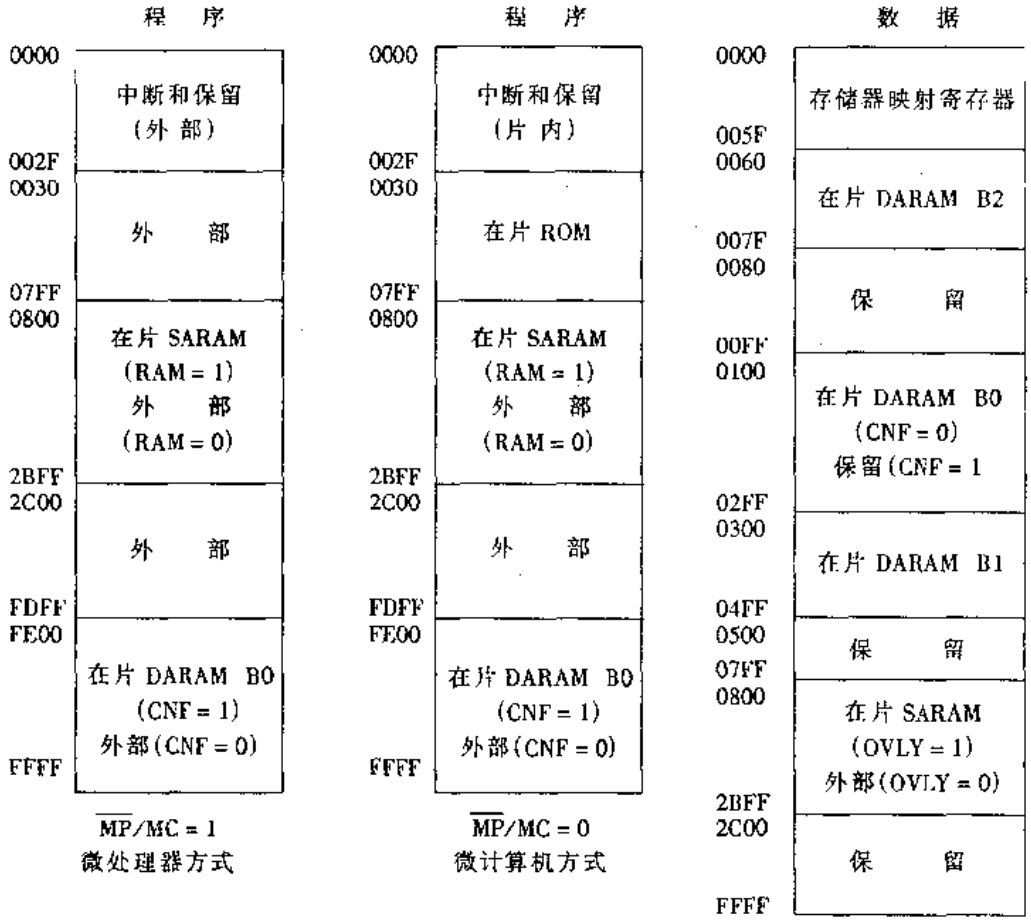
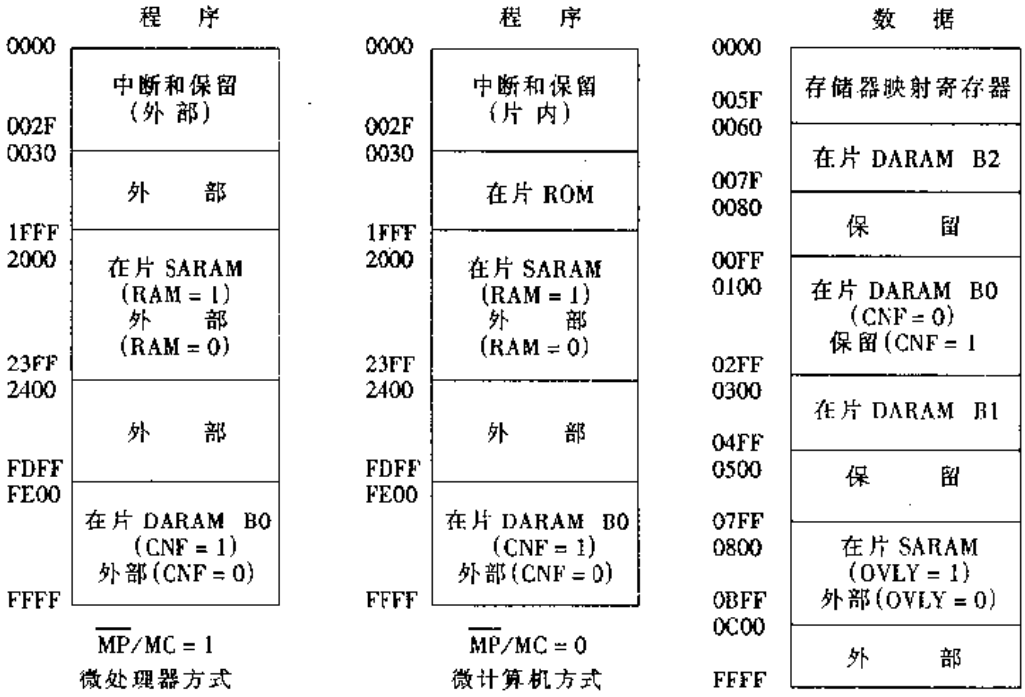
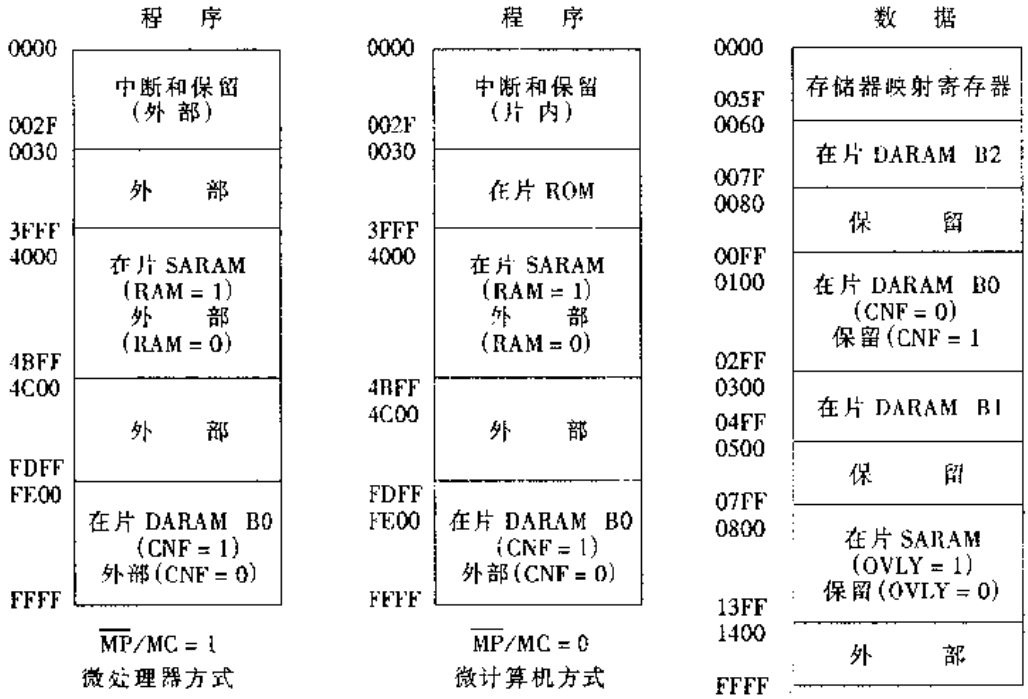


图 4 - 3 TMS320C50 存储器分配

图 4－4 为'C51 和'C53 的存储器分配图。



(a) TMS320C51 存储器分配



(b) TMS320C53 存储器分配

图 4－4 TMS320C51，'C53 存储器分配

2. CALU

TMS320C5X 的中央算术逻辑单元(CALU)包括一个 16 位定标移位器、一个 16×16 位并行乘法器、一个 32 位累加器(ACC)、一个 32 位累加缓冲器(ACCB)以及在累加器和乘法器输出中都要用到的附加移位器。CALU 的组成如图 4-5 所示。一个典型的 ALU 指令要有以下步骤:

(1)从存储器取数据到数据总线。

(2)数据通过定标移位器和 ALU,在那里执行数学运算。

(3)结果存入累加器。

ALU 的一个输入通常由累加器提供,另一个输入可以由乘法器的乘积寄存器(PREG)、累加缓冲器(ACCB)以及被装入存储器或累加器(ACC)的定标移位器传来。

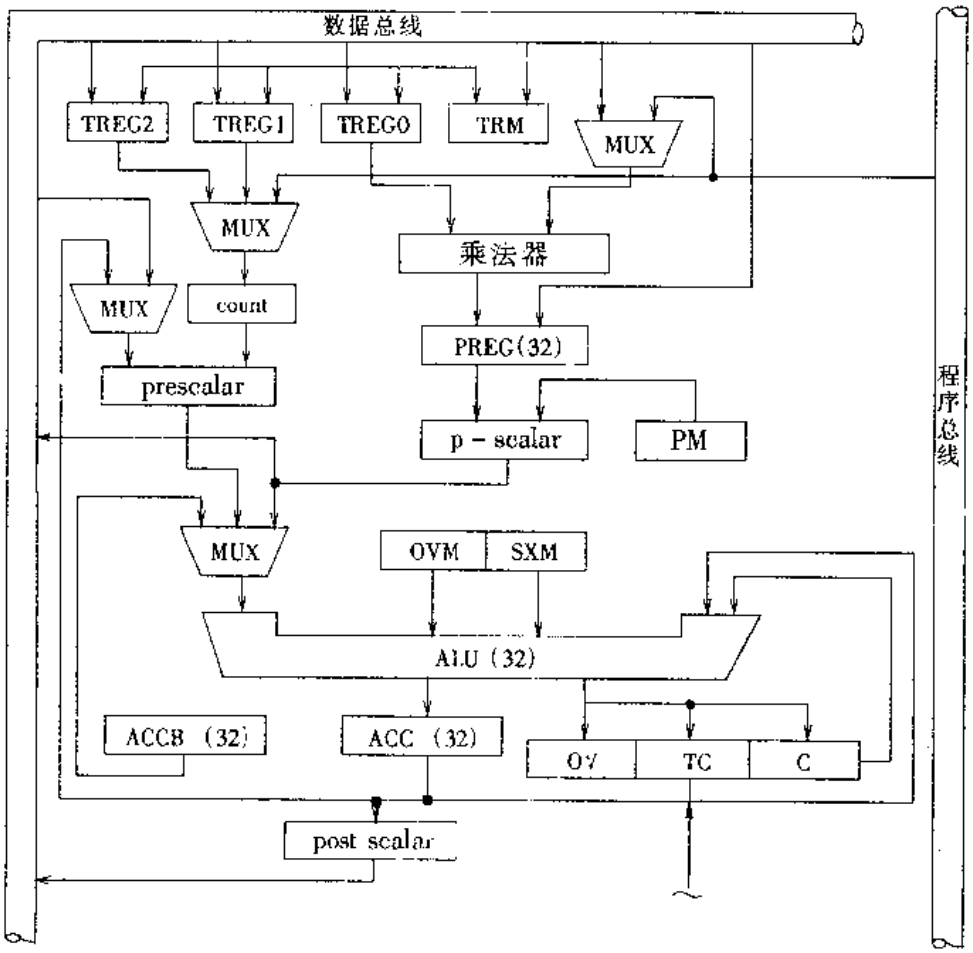


图 4-5 TMS320C5X CALU 结构框图

3. 中断

TMS320C5X 的 CPU 单元提供了 16 个用户可屏蔽中断($\overline{INT16}$ ~ $\overline{INT1}$),然而,每个'C5X DSP 并没有使用全部 16 个中断,'C50、'C51、'C53 仅使用了这些中断中的 9 个。串行口(RINT, XINT, TRNT, TXNT)、定时器(TINT)、软件中断指令(TRAP 和 INTR)都能够产生中断。复位($\overline{RS}$)中断拥有最高的中断权,$\overline{INT16}$的中断权最低。

4. 并行逻辑单元(PLU)

并行逻辑单元可在控制/状态寄存器以及任何数据存储器位置直接设置、清除、检验或反

转乘法器，PLU 提供了一个不影响累加器或乘积寄存器内容和数据寄存器值的直接逻辑操作途径，它能被用来设置或清除控制寄存器中的乘法位或检测标志寄存器中的乘法位，PLU 结构框图如图 4－6 所示。

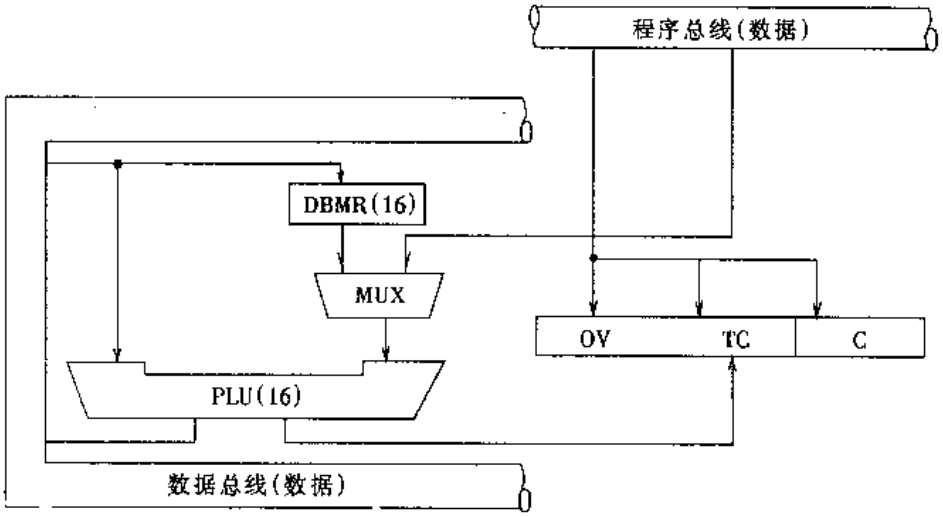


图 4－6　TMS320C5X PLU 结构框图

5.DMA

'C5X 使用在片 SARAM 或外存的 DMA 来支持多处理器设计，它通过暂停一个或多个处理器的执行来允许其它的处理器读取或写入'C5X 的局部片外存储器或片内 RAM。通过$\overline{\text{HOLD}}$/$\overline{\text{HOLDA}}$信号来控制外部存储器存取，'C5X 内部 RAM 的 DAM 存取通过$\overline{\text{HOLD}}$、$\overline{\text{HOLDA}}$、R/$\overline{\text{W}}$、$\overline{\text{STRB}}$、$\overline{\text{BT}}$、$\overline{\text{IAQ}}$引脚来控制。

三、外围设备接口

与'C5X 的 CPU 相联系的 7 个外设接口是串行口、TDM 串行口、定时器、软件可编程等待周期产生器、I/O 接口、1 比 1 分频时钟以及 XF 和$\overline{\text{BIO}}$引脚，外设通过存储映射寄存器来控制，串行口和定时器通过中断与 CPU 同步。

1. 外设控制

外围电路通过对存储器映射控制。设置和清零位可以允许、禁止或初始化外围设备，数据通过存储器映射数据寄存器和外设间进行传递。

2. 并行输入/输出端口

'C5X 有 64K 并行输入/输出端口，对 I/O 端口的存取由 I/O 空间选择信号($\overline{\text{IS}}$)来控制。64K 端口中的 16 个被映射到数据存储器空间，全部这 64K 端口可以用 IN 和 OUT 指令来存取，16 个存储器映射 I/O 端口(50H－5FH)还能通过读出或写入数据空间的一个位置来存取。

3. 软件可编程等待周期产生器

软件可编程等待周期生成器可用来把外部总线周期扩展 7 个机器周期，当外部存取配置成 0 等待周期时，等待周期生成器的内部时钟关闭，允许机器以低耗操作方式运行。软件可编程等待周期生成器由两个 16 位等待周期寄存器(PDWSR 和 IOWSR1)和一个 5 位控制寄存器(CWSR)来控制。

4. 串行口

全双工在片串行口提供了与串行设备间的直接联系，接口信号与串行设备兼容，串行口可在多处理器场合中用作处理器间的内部通信。'C5X 的接收和发送操作都有双缓冲，这样就允许使用连续的通信流。使用内部时钟时，最大操作频率是 CLKOUT1/4(50ns 方式下 5M 位/秒，35ns 方式下 7.14M 位/秒)。当串行口被复位时，机器能被设置成关闭串行口内部时钟而允许以掉电方式运行。

5.TDM 串行口

'C5X 有一个 TDM(多路分时)串行口，允许设备与其它 7 个'C5X 设备进行串行通信。TDM 端口为多处理应用提供了一个简单而有效的接口。通过 TSPC 控制寄存器中 TDM 的设置，端口可被配置成多处理方式(TDM = 1)或单机方式(TDM = 0)

6. 定时器

定时器是一个在片减 1 计数器，可以用它来产生 CPU 中断。定时器每个 CLKOUT1 周期减 1，当计数器减到 0 时，定时器产生中断(TINT)。定时器的结构框图 4 - 7 所示。当定时器停止时，定时器内部时钟关闭，允许机器以掉电方式操作。

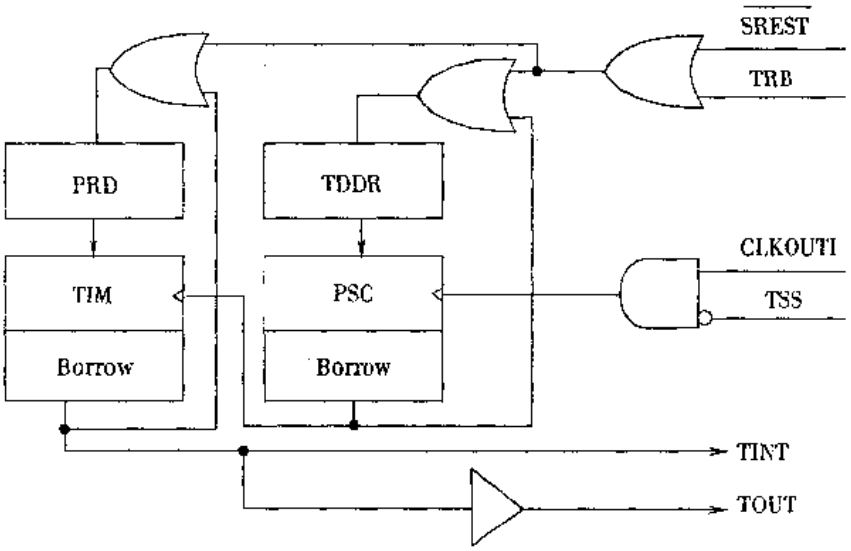


图 4 - 7　定时器结构框图

四、TMS320C5X 寻址方式和指令系统

.1. 寻址方式

TMS320C5X 指令集提供了六种基本的存储器寻址方式：

- 直接寻址方式
- 间接寻址方式
- 立即寻址方式
- 特定寄存器寻址方式
- 存储映射寄存器寻址方式
- 循环寻址方式

(1)直接寻址方式

直接寻址方式中，指令包含着数据存储器地址的低7位，它和数据存储器页指针(DP)的9位一起形成16位数据存储器地址。这样，DP寄存器指向512个可能的128个字数据之一，指令中的7位地址指向包含数据页的特定位置。使用LDP(装入数据页指针)或LST #0(装入状态存器ST0)指令来装入DP寄存器，直接寻址的格式如下表示：

15 14 13 12 11 10 9 8	7	6 5 4 3 2 1 0
操 作 码	0	数据存储器地址

其中第8位到第15位是操作码，第7位为0表明直接寻址方式，第0到6位是数据存储器地址。

(2)间接寻址方式

TMS320C5X的8个辅助寄存器提供了强有力的间接寻址方式，为了选择一个特定的辅助寄存器，通过从0到7的值来装入辅助寄存器指针(ARP)，辅助寄存器的内容可以通过辅助寄存器数学单元(ARAU)来向上操作ARAU实现16位无浮点数学运算，它在时序中的译码段执行辅助寄存器数学运算，这样允许在下一条指令译码前产生地址，AR在当前指令后减1。间接寻址中，64K数据空间中的任何地方都可以通过辅助寄存器中的16位地址来存取，LAR指令把地址的指令来初始化，AR(ARP)标志着辅助寄存器被ARP选中。辅助寄存器还能由数据总线装入，这通过使用存储器映射写入辅助寄存器来实现，如下指令可以写入存储器映射辅助寄存器：APL，BLDD，LMMR，OPL，SACH，SACL，SAMM，SMMR，SPLK，XPL。

下列符号在间接寻址，包括位反转寻址中使用。

*	AR(ARP)内容被用作数据存储器地址
* -	AR(ARP)内容被用作数据存储器地址，存取后减1
* +	AR(ARP)内容被用作数据存储器地址，存取后加1
* 0 -	AR(ARP)内容被用作数据存储器地址，存取后减去INDX内容
* 0 +	AR(ARP)内容被用作数据存储器地址，存取后加上INDX内容
* BR0 -	AR(ARP)内容被用作数据存储器地址，存取后减去带反转进位值的INDX内容
* BR0 +	AR(ARP)内容被用作数据存储器地址，存取后加上带反转进位值的INDX内容

有两种主要类型的变址间接寻址：

①带增1或减1的一般间接寻址。

②基于INDX值的变址间接寻址：

· 通过加或减去INDX内容变址

· 通过加或减去带反转进位值的INDX内容变址(用于'C5X的FFT运算)

间接寻址可被除立即操作和空操作之外的所有指令使用，其寻址格式如下：

15 14 13 12 11 10 9 8	7	6	5	4	3	2 1 0
操 作 码	1	IDV	INC	DEC	NAR	Y

第8到15位是操作码，第7位的1表示间接寻址，第0到6位是间接寻址控制位，第6位表示增值或减值，第4、5位通过AR(ARP)和INDX寄存器来控制数学操作，第5位置1表示增

值操作第4位置1表示减值操作，第3位和第0到第2位控制辅助寄存器指针(ARP)，第3位决定了下一个值是否装入到ARP中，如果第3位为1，则第0到第2位内容装入ARP，否则其内容不变。如果ARP装入新值，旧值装入辅助寄存器缓冲器(ARB)。表4-2表示了间接寻址的位段、符号和相应操作。

表4-2 间接寻址字段表

15 … 8	7	6	5	4	3	2	1	0	符号	操作
←Opcode→	1	0	0	0	0	←	Y	→	*	对 AR_X/ARP 无操作
←Opcode→	1	0	0	0	0	←	Y	→	*,Y	Y→ARP
←Opcode→	1	0	0	0	0	←	Y	→	*-	AR(ARP)-1→AR(ARP)
←Opcode→	1	0	0	0	0	←	Y	→	*-,Y	AR(ARP)-1→AR(ARP) Y→ARP
←Opcode→	1	0	1	0	0	←	Y	→	*+	AR(ARP)+1→AR(ARP)
←Opcode→	1	0	1	0	0	←	Y	→	*+,Y	AR(ARP)+1→AR(ARP) Y→ARP
←Opcode→	1	1	0	0	0	←	Y	→	*BR0-	AR(ARP)-rcINDX→AR(ARP)
←Opcode→	1	1	0	0	0	←	Y	→	*BR0-,Y	AR(ARP)-rcINDX→AR(ARP) Y→ARP
←Opcode→	1	1	0	1	0	←	Y	→	*0-	AR(ARP)-INDX→AR(ARP)
←Opcode→	1	1	0	1	0	←	Y	→	*0-,Y	AR(ARP)-INDX→AR(ARP) Y→ARP
←Opcode→	1	1	1	0	0	←	Y	→	*0+	AR(ARP)+INDX→AR(ARP)
←Opcode→	1	1	1	0	1	←	Y	→	*0+,Y	AR(ARP)+INDX→AR(ARP) Y→ARP
←Opcode→	1	1	1	1	0	←	Y	→	*BR0+	AR(ARP)+rcINDX→AR(ARP) Y→ARP
←Opcode→	1	1	1	1	1	←	Y	→	*BR0+,Y	AR(ARP)+rcINDX→AR(ARP) Y→ARP

(3)立即寻址方式

立即寻址中，指令字是立即操作数的值，TMS320C5X有单字(8位，9位和13位内容)短立即指令和双字(16位内容)长立即指令。在短立即指令中，立即操作数的本身就是指令字。长立即指令中，指令字的下一个字被用作立即操作数。

支持立即寻址的指令有：

8位立即寻址：ADD，ADRK，LACL，LAR，RPT，SBRK，SUB

9位立即寻址：LDP

13位立即寻址：MPY

16位立即寻址：ADD，AND，APL，CPL，LACC，LAR，MPY，OPL，OR，RPT，RPTZ，SPLK，SUB，XOR，XPL

(4)特定寄存器寻址

'C5X指令集中有9条指令可使用CPU中两个专用存储映射寄存器之一，这两个寄存器是块移动地址寄存器(DMAR)和动态位操作寄存器(DBMR)。当一个立即数不被当作操作数

时 APL,OPL,CPL,XPL 并行逻辑单元(PLU)指令使用 DBMR 寄存器内容。BLDD,BLDP 和 BLPD 指令利用 DMAR 寄存器指出块移动的源地址和目的地址。MADD 和 MADS 也使用 BMAR 寄存器在程序存储器中寻址操作码。

(5)存储映射寄存器寻址

存储映射寄存器寻址被用来初始化存储映射寄存器而不影响当前数据页指针的值。另外,任何暂存 RAM 或数据页 0 都可用这种寻址来初始化,因为数据页指针不需在操作前后被初始化,这样所涉及到的存储映射寄存器的执行操作大大减少。下面指令使用存储映射寄存器寻址方式操作:

LAMM	把存储映射寄存器内容装入到累加器
SAMM	把累加器内容存入到存储映射寄存器
LMMR	把数据装入到存储映射寄存器
SMMR	从存储映射寄存器存入数据

(6)循环寻址

许多算法都需要利用存储器中的循环缓冲区,在这些运算中,循环缓冲区被用来实现一窗口,它保存了需要处理的最新数据。'C5X 通过辅助寄存器来支持两种循环缓冲区操作,下面的 5 个存储映射寄存器控制循环缓冲区操作:

CBSR1	循环缓冲区 1 始端寄存器
CBSR2	循环缓冲区 2 始端寄存器
CBER1	循环缓冲区 1 末端寄存器
CBER2	循环缓冲区 2 末端寄存器
CBCR	循环缓冲区控制寄存器

8 位循环缓冲区控制寄存器允许和禁止循环缓冲区操作,CBCR 定义如下:

第 0~2 位,CAR1,指出哪个辅助寄器被映射到循环缓冲区 1。

第 3 位,CENB1,循环缓冲器 1,允许 = 1,禁止 = 0,复位后置 0。

第 4~6 位,CAR2,指出哪个辅助寄器被映射到循环缓冲区 2。

第 7 位,CENB2,循环缓冲器 2,允许 = 1,禁止 = 0,复位后置 0。

为定义循环缓冲区,始端和末端地址应当先被装入到响应缓冲区寄存器,接着,循环缓冲区始端和末端间的一个值被装入到辅助寄存器。辅助寄存器值被装入,在控制寄存器 B 中设置响应循环缓冲区允许位,同一辅助寄存器不能用来允许两循环缓冲区,否则可能会产生不可预料的结果。

2. 指令系统

TMS320C5X 汇编语言指令集支持专用 DSP 和一般应用,如同接收'C5X 的指令一样,TI 公司的'C5X 编译器也可接收'C2X 的指令,表 4-3 给出了'C5X 和'C2X 的指令对照,表 4-4 为'C5X 增加的指令。

表 4－3 'C2X 和'C5X 指令对照

累加器存储器指令	
'C2X 助记符	'C5X 助记符
ABS	ABS
ADD	ADD
ADDC	ADDC
ADDH	ADDH
ADDK	ADDK
ADDS	ADDS
ADDT	ADDT
ADLK	ADLK
AND	AND
ANDK	ANDK
CMPL	CMPL
LAC	LAC
LACK	LACK
LACT	LACT
LALK	LACC
NEG	NEG
NORM	NORM
OR	OR
ORK	ORK
ROL	ROL
ROR	ROR
SACH	SACH
SACL	SACL
SBLK	SBLK
SFL	SFL
SFR	SFR
SUB	SUB
SUBB	SUBB
SUBC	SUBC
SUBH	SUBH
SUBK	SUBK
SUBS	SUBS
SUBT	SUBT
XOR	XOR
XORK	XORK

累加器存储器指令	
'C2X 助记符	'C5X 助记符
ZAC	LACL
ZALH	LACC
ZALR	ZALR
ZALS	LACL
辅助寄存器和数据页指针指令	
'C2X 助记符	'C5X 助记符
ADRK	ADRK
CMPR	CMPR
LAR	LAR
LARK	LAR
LARP	MAR
LDP	LDP
LDPK	LDP
LRLK	LAR
MAR	MAR
SAR	SAR
SBRK	SBRK
I/O 和数据存储器指令	
'C2X 助记符	'C5X 助记符
BLKD	BLDD
BLKP	BLPD
DMOV	DMOV
FORT	OPL
	APL
IN	IN
OUT	OUT
RFSM	APL
RTXM	APL
RXF	CLRC
SFSM	OPL
STXM	OPL
SXF	SETC
TBLR	TBLR
TBLW	TBLW

T寄存器P寄存器和乘法指令	
'C2X 助记符	'C5X 助记符
APAC	APAC
LPH	LPH
LT	LT
LTA	LTA
LTD	LTD
LTP	LTP
LTS	LTS
MAC	MAC
MACD	MACD
MPY	MPY
MPYA	MPYA
MPYK	MPYK
MPYS	MPYS
MPYU	MPYU
PAC	PAC
SPAC	SPAC
SPH	SPH
SPL	SPL
SQRA	SQRA
SQRS	SQRS
分支/调用指令	
'C2X 助记符	'C5X 助记符
B	B
BACC	BACC
BANZ	BANZ
BBNZ	BBNZ
BBZ	BCND
BC	BCND
BGEZ	BCND
BV	BCND
BZ	BCND
CALA	CALA
CALL	CALL
RET	RET
TRAP	TRAP

分支调用指令	
'C2X 助记符	'C5X 助记符
BGZ	BCND
BIOZ	BCND
BLEZ	BCND
BLZ	BCND
BNC	BCND
BNV	BCND
BNZ	BCND
控制指令	
'C2X 助记符	'C5X 助记符
BIT	BIT
BITT	BITT
CNFD	CLRC
CNFP	SETC
DINT	SETC
EINT	CLRC
IDLE	IDLE
LST	LST
LST1	LST
NOP	NOP
POP	POP
POPD	POPD
PSHD	PSHD
PUSH	PUSH
RC	CLRC
RHM	CLRC
ROVM	CLRC
RPT	RPT
RPTK	RPT
RSXM	CLRC
RTC	CLRC
SC	SETC
SHM	SETC
SOVM	SETC
SST	SST
SST1	SST
SSXM	SETC
STC	SETC

表 4－4　TMS320C5X 增加的指令

助记符	说　　明	字数	周期数
累加器存储器指令			
ADCB	带进位加 ACCB 到 ACC	1	1
ADDB	加 ACCB 到 ACC	1	1
ANDB	ACCB 与 ACC 相与	1	1
BSAR	ACC 桶形右移位	1	1
CRGT	检测 ACC > ACCB	1	1
CRLT	检测 ACC < ACCB	1	1
EXAR	ACC 和 ACCB 内容相交换	1	1
LACB	ACCB 内容送入 ACC	1	1
LAMM	存储映射寄存器内容送入 ACC	1	1(2)
ORB	ACCB 与 ACC 相或	1	1
ROLB	ACCB 和 ACC 循环左移	1	1
RORB	ACCB 和 ACC 循环右移	1	1
SACB	ACC 内容存入 ACCB	1	1
SAMM	ACC 内容存入存储映射寄存器	1	1(2)
SATH	按 TREG1 指定桶形右移 16 位或不移动	1	1
SATL	按 TREG1 指定桶形右移 0 到 15 位	1	1
SBB	ACC 减去 ACCB	1	1
SBBB	ACC 带借位减 ACCB	1	1
SFLB	ACC 和 ACCB 左移	1	1
SFRB	ACC 和 ACCB 右移	1	1
XORB	ACCB 和 ACC 相异或	1	1
ZAP	ACC 和 PREG 置 0	1	1
并行逻辑单元指令			
SPLK	存长立即数到数据存储器	2	2
XPL	DBMR 或指定数与数据存储器值异或	1/2	1(2)
APL	DBMR 或指定数与数据存储器值相与	1/2	1(2)
CPL	DBMR 或指定数与数据存储器值比较	1/2	1(2)
OPL	DBMR 或指定数与数据存储器值相或	1/2	1(2)
T 寄存器、P 寄存器和乘法指令			
MADD	带 BMAR 指向的源操作数乘且累加	1	3

MADS	带 BMAR 指向的源操作数和块移动乘且累加	1	3
ZPA	乘积寄存器清 0	1	1
	分 支 指 令		
BR	无条件延迟分支	2	2
BACCD	延迟分支到 ACC 指定地址	1	2
BANZD	延迟分支到 $AR_{n(n=0)}$	2	2
BCNDD	条件延迟分支	2	2
CALAD	延迟直接调用字程序	1	2
CALLD	延迟调用子程序	2	2
CCD	条件延迟调用	2	2
INTR	软件中断	1	4
NMI	非屏蔽中断	1	4
RETD	从子程序延迟返回	1	2
RETCD	条件延迟回	1	2
RETE	带相应开关和全局中断允许返回	1	4
RETI	带相应开关返回	1	4
XC	有条件地执行下一条指令	1	1
	I/O 和 数 据 存 取 指 令		
BLDP	从数据存储器到程序存储器块移动	1	2
LMMR	把数据装入到存储映射寄存器	2	2(3)
SMMR	从存储映射寄存器装入数据	2	2(3)
	控 制 指 令		
IDLE	空闲状态测中断	1	1
IDLE2	空闲状态测中断,掉电方式	1	1
RPTB	块循环	2	2
RPTZ	清 ACC 和 PREG,循环下一指令	2	2

4.2 μPD77230 处理器

μPD77230 是日本 NEC 公司生产的第二代信号处理器。它的第一代产品是 μPD7720。后继产品是 μPD77240。

μPD77230 和其他通用信号处理器一样,芯片内装有高速乘法器,乘法器和累加器由专用总线相连,进行数据的流水线处理。乘数和被乘数可同时由存储器装入乘法器,并有专门的地址生成电路使存储器能够连续地向运算单元提供数据。同时备有专门的接口可直接与外部存

储器及 A/D、D/A 转换器等外部设备相连。

μPD77230 性能参数如下：

硬件技术	CMOS
元件数	370K
芯片面积	112mm
消耗功率	800mV
运算能力	13.4MFLOPS
浮点乘法器	32×32→55 位
浮点加法器	55 位
片内 RAM	30×512 位×2
片内 ROM	数据 32×1024 位
	指令 32×2048 位
外部存储器	32×8K 位

一、μPD77230 硬件构成

μPD77230 硬件构成方块图如图 4－8 所示。基本上分为存储器、程序控制器、数据运算及输入输出等四个部分。

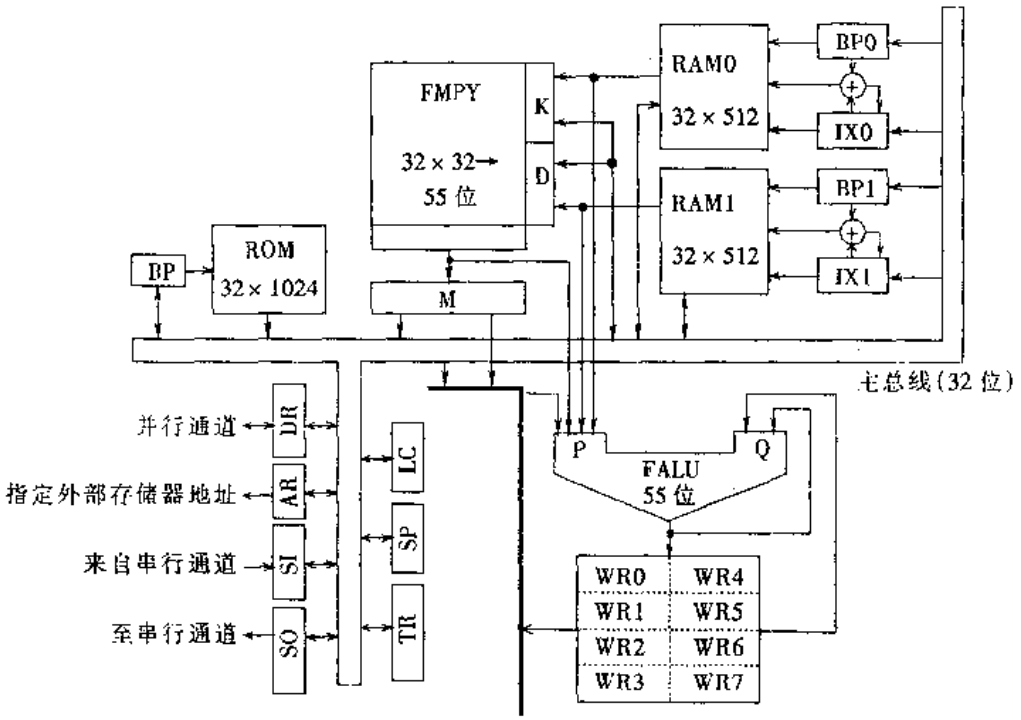


图 4－8　μPD77230 构成方块图

数据运算部分有浮点乘法器(FMPY)、浮点累加器 FALU、乘法器输入寄存器 K、L、输出寄存器 M 及运算结果标志寄存器(FLAG)等，浮点乘法器和累加器由专用总线相连，以流水线方

式进行数据运算处理。FMPY 可在一个机器周期内(150ns)求得分别在 K、L 寄存器内的两个 32 位数据之积,并将结果(55 位:阶码 8 位,尾数 47 位)送入 FALU 与工作寄存器(WR)内的 55 位的浮点数据进行累加,并将累加结果的上位 32 位输出到数据总线。

为了提高运算速度,μPD77230 片内有数据只读存储器 ROM 和两个随机数据存储器 RAM0 和 RAM1,RAM 和 FMPY 间有专门的数据总线相连,当同一个数据传送指令指定乘算对象时,可以实现(RAM)*(ROM)、(RAM)*(RAM)、(RAM)*(寄存器)等方式的选择。通过对乘算内容的两次数据传送,亦可以进行(ROM)*(寄存器)的运算。

μPD77230 除通过数据口 DP、地址口 AP 对外部存储器进行数据存取外,还通过串行口 SI 及输出口 SO 与 A/D、D/A 变换器等相接,以处理模拟信号。

全部硬件通过微程序进行控制。32 位的指令存储在程序存储器 INST ROM 内。PC 是程序计数器,STACK 是子程序栈,LC 是循环次数计数器,可同时进行命令读取和数据处理。

1.指令存储器

μPD77230 指令字长 32 位。指令存储器 ROM 容量为 2K 字(2048 字),还可以外接 4K 字的指令存储器。此外,它还配有和程序计数器连接的堆栈。

2.处理单元(PU)

处理单元完成除乘法运算以外的所有浮点运算。其组成部分包括:24 位尾数运算单元(ALU)、8 位阶码运算单元(EAU)、用于存储运算结果的 8 个工作寄存器(WR)及标志、移位单元。

3.乘法器

乘法器是一个 32 位×32 位的高速并行处理单元。在一个指令周期内,完成 K、L 寄存器内数据的乘算,并在下一个指令周期将 55 位的运算结果送入 M 寄存器 。

4.总线

除 32 位主总线外,还具有 55 位的用于 PU 的内部总线。另外,RAM 和乘法器之间、RAM 和 PU 之间有数据总线连接,因而可实现运算与数据传送同时进行。

5.数据存储器

数据存储器由 32 位的 RAM 和 ROM 组成,另外,尚可外接 8K 的外部存储器。

(1) ROM

在片内 ROM 为 1K 字,由 RP 指针指定地址。

(2) RAM

在片 2 个 512×32 位 RAM,分别称为 RAM0 和 RAM1。其地址由 BP (Base Pointer)和 IX (Index Pointer)进行指定。不使用总线可以将数据直接传送到乘法器。

6.接口

μPD77230 通过接口进行数据的输入和输出,这部分电路由保存数据的寄存器、各种接口及辅助电路构成。各寄存器功能说明如下:

(1) DR (Data Register)

在由并行数据口和外部设备进行数据输入输出时,由 DR 保存处理的数据;在使用外部存储器时,或工作在从机方式下与主机 CPU 交换数据时,使用该数据寄存器。

(2) SI (Serial Input)

SI 是一个 32 位的寄存器,用以保持串行口从外部输入的数据。

(3) SO (Serial Output)

So 是一个 32 位寄存器，用以保持由串行口输出到外部的数据。

(4) SR (Status Register)

显示错误状态、保持并行、串行输入输出或中断操作方式，是一个 20 位寄存器。可以从程序中自由读/写。在进行 A/D 转换器或 D/A 转换器的数据传递时，必须事先进行设定。

(5) TR (Temporary Register)

临时数据寄存器，是一个 32 位通用寄存器。

(6) LC (Loop Conter)

设定程序循环次数的 10 位寄存器。用于控制程序的循环。

μPD77230 基本运算能力测试结果如下：

· 数值运算：

除算	4.8μs
平方根	9.0μs
sin	10.8μs
cos	10.8μs

· 信号处理：

2 节 FIR	0.9μs
32 节 FIR	5.25μs
32 点 FFT	150μs
512 点 FFT	4.7ms
1024 点 FFT	12.3ms

二、 μPD77230 指令系统

μPD77230 指令字长 32 位。根据功能可分三类，即运算指令、分支指令和置数据指令，其指令构成形式如图 4-9 所示。

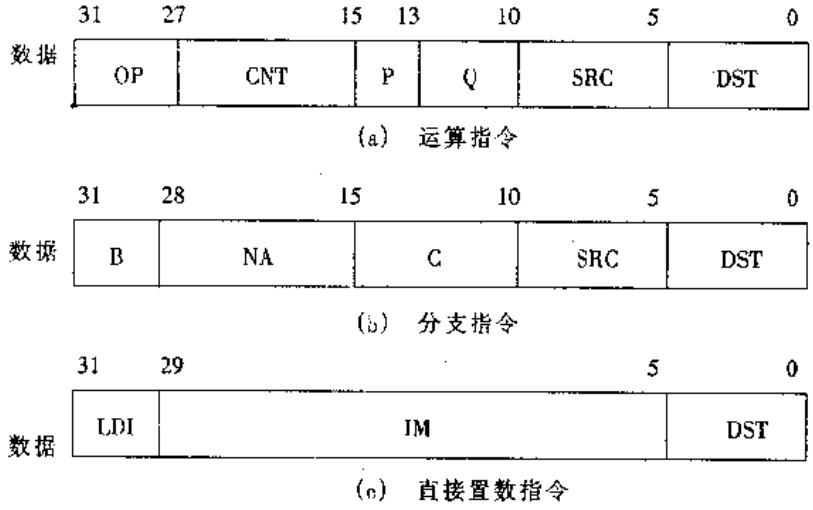


图 4-9 指令的种类及其字段

1. 运算指令

运算指令以使用处理单元的运算指令为中心，包括利用总线的各种数据传送指令和各种指针操作指令。

如图 4－9(a)所示，运算指令由六个字段组成。

(1)OP 字段

运算字段有表 4－5 所示的运算指令，应注意指令执行时受流水线的影响。

表 4－5　由运算字段指定的内容

助记符	运算字段					运算功能
	D31	D30	D29	D28	D27	
NOP	0	0	0	0	0	无操作
INC	0	0	0	0	1	(WR)＝(WR)＋1
DEC	0	0	0	1	0	(WR)＝(WR)－1
ABS	0	0	0	1	1	(WR)＝|(WR)|
NOT	0	0	1	0	0	(WR)＝(WR)
NEG	0	0	1	0	1	(WR)＝－(WR)
SHLC	0	0	1	1	0	带进位，左移
SHRC	0	0	1	1	1	带进位，右移
ROL	0	1	0	0	0	向右循环移 1 位
ROR	0	1	0	0	1	向左循环移 1 位
SHLM	0	1	0	1	0	左移不带进位
SHRM	0	1	0	1	1	右移不带进位
SHRAM	0	1	1	0	0	算术右移
CLR	0	1	1	0	1	(WR)＝0
NORM	0	1	1	1	0	规格化处理
CVT	0	1	1	1	1	数的格式变换
ADD	1	0	0	0	0	定点加，无进位
SUB	1	0	0	0	1	定点减，无进位
ADDC	1	0	0	1	0	带进位定点加
SUBC	1	0	0	1	1	带进位定点减
CMP	1	0	1	0	0	浮点两数值比较
AND	1	0	1	0	1	(WR)＝(WR)∧(P)
OR	1	0	1	1	0	(WR)＝(WR)∨(P)
XOR	1	0	1	1	1	(WR)＝(WR)∨(P)
ADDF	1	1	0	0	0	浮点加
SUBF	1	1	0	0	1	浮点减

(2)P 字段

P 字段选择 RAM0、RAM1、M 寄存器或内部总线上的一个数据。如表 4－6 所示。

表 4－6　P 字段指定的内容

助记符	P 字段		操作对象
	D13	D14	
IB	0	0	内部总线值
M	0	1	乘法器输出

RAM0	1	0	RAM0 值
AM1	1	1	RAM1 值

(3)Q 字段

指定 WR0 ~ WR7 中的一个做为运算单元的工作寄存器,并在指定的寄存器中存放运算结果,如表 4 - 7 所示。

表 4 - 7 Q 字段指定的内容

助记符	Q 字段	操作对象
	D12 D11 D10	
WRn	n 为二进制数	工作寄存器

(4)SRC 字段

SRC 字段指定数据传送的源地址。如表 4 - 8 所示。

表 4 - 8 SRC 字段指定的内容

助记符	SRC 字段					传送原寄存器
	D9	D8	D7	D6	D5	
NON	0	0	0	0	0	无操作
RP	0	0	0	0	1	ROM 指针
PSW0	0	0	0	1	0	算术运算状态 0
PSW1	0	0	0	1	1	算术运算状态 1
SVR	0	0	1	0	0	位移量寄存器
SR	0	0	1	0	1	状态寄存器
LC	0	0	1	1	0	循环计数器
STK	0	0	1	1	1	堆栈存储器
M	0	1	0	0	0	M 寄存器(乘算结果)
ML	0	1	0	0	1	M 寄存器下位 24 位(乘算结果)
ROM	0	1	0	1	0	ROM
TR	0	1	0	1	1	TR 寄存器
AR	0	1	1	0	0	地址寄存器
SI	0	1	1	0	1	串行输入寄存器
DR	0	1	1	1	0	并行输入输出寄存器
DRS	0	1	1	1	1	并行输入输出寄存器 *
WR0	1	0	0	0	0	工作寄存器 0
WR1	1	0	0	0	1	工作寄存器 1
WR2	1	0	0	1	0	工作寄存器 2
WR3	1	0	0	1	1	工作寄存器 3
WR4	1	0	1	0	0	工作寄存器 4
WR5	1	0	1	0	1	工作寄存器 5
WR6	1	0	1	1	0	工作寄存器 6
WR7	1	0	1	1	1	工作寄存器 7
RAM0	1	1	0	0	0	RAM0
RAM1	1	1	0	0	1	RAM1
BP0	1	1	0	1	0	基础指针 0(RAM0 用)
BP1	1	1	0	1	1	基础指针 1(RAM1 用)

IX0	1	1	1	0	0	变址寄存器指针 0(RAM0 用)
IX1	1	1	1	0	1	变址指针 1(RAM1 用)
L	1	1	1	1	1	L 寄存器(乘法器输入)

* 仅从机方式时有效

(5)DST 字段

DST 字段指定数据传送的目的地址,如表 4-9。

表 4-9 DST 字段指定的内容

助记符	DST 字段					传送原寄存器
	D4	D3	D2	D1	D0	
NON	0	0	0	0	0	无操作
RP	0	0	0	0	1	ROM 指针
PSW0	0	0	0	1	0	算术运算状态 0
PSW1	0	0	0	1	1	算术运算状态 1
SVR	0	0	1	0	0	位移量寄存器
SR	0	0	1	0	1	状态寄存器
LC	0	0	1	1	0	循环寄存器
STK	0	0	1	1	1	堆栈寄存器
LKR0	0	1	0	0	0	L←(SRC),K←(RAM0)
KLR1	0	1	0	0	1	K←(SRC),L←(RAM1)
TRE	0	1	0	1	0	TR 寄存器阶码(上 8 位)
TR	0	1	0	1	1	TR 寄存器
AR	0	1	1	0	0	地址寄存器
SO	0	1	1	0	1	串行输入口寄存器
DR	0	1	1	1	0	并行口输出寄存器
DRS	0	1	1	1	1	并行口输入输出寄存器
WR0	1	0	0	0	0	工作寄存器 0
WR1	1	0	0	0	1	工作寄存器 1
WR2	1	0	0	1	0	工作寄存器 2
WR3	1	0	0	1	1	工作寄存器 3
WR4	1	0	1	0	0	工作寄存器 4
WR5	1	0	1	0	1	工作寄存器 5
WR6	1	0	1	1	0	工作寄存器 6
WR7	1	0	1	1	1	工作寄存器 7
RAM0	1	1	0	0	0	RAM0
RAM1	1	1	0	0	1	RAM1
BP0	1	1	0	1	0	基本指针 0(RAM0)
BP1	1	1	0	1	1	基本指针 1(RAM1)
IX0	1	1	1	0	0	变址指针 0(RAM0)
IX1	1	1	1	0	1	变址指针 1(RAM1)
K	1	1	1	1	0	K 寄存器(乘法器输入)
L	1	1	1	1	1	L 寄存器(乘法器输入)

(6)CNT 字段

CNT 字段指定各种指针操作、中断设置、数据传送格式及移位、规格化指令种类等。

CNT 字段依据指定内容又可分为 24 种子字段,如表 4-10 所示。这些子字段的组合,又

可进行 15 类指定如表 4－11。

表 4－10　CNT 字段的子字段一览表

组	子字段	功　　能	有效指令
中断	EM	指定可屏蔽中断允许(允许,不允许)	次一个
	BM	指定可屏蔽中断存储(存储/不存储)	次一个
算术运算标志	FIS	PSW 控制	
	FS	由 PSW0 和 PSW1 切换有效标志	
ROM 指针	RP	指定 ROM 的寻址方式	次一个
	RPC	指定由 ROM 指针 2 进制计数的 n 值	次一个
	RPS	直接指定 ROM 指针的低 9 位	次一个
RAM0 指针	M0	指定 RAM0 的寻址	次一个
	DP0	指定 BP,IX 指针的操作	次一个
	BASE0	指定 BP 计数操作时的上限	次一个
数据格式变换	FD	指定 CVT 指令进行的格式变换操作	
	WI	指定 BP、IX 指针的操作	
	WT	制定从工作寄存器传送数据的格式	
规格化	NF	指定规格化处理(NORM 指令)操作	
位移量	SHV	指定尾数部分 47 位的移位量	
外部存储器存取	RW	外部存储器输入或输出指定	
	EA	指定地址寄存器的增、减	
通用输出口	P2	P2 引脚信号输出控制(从机方式)	次一个
	P3	P3 引脚信号输出控制(从机方式)	次一个
循环计数器	L	指定循环计数量的减量	
转移	NAL	指定无条件转移地址(低 9 位)	

表 4－11　CNT 字段的子字段组合

D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16	D15
0	0	M0		M1		DP0			DP1		
0	1	0	0	EA		DP0			DP1		
0	1	0	1	RP		M0		DP0			FC
0	1	1	0	RP		M1		DP1			FC
0	1	1	1	RP		M0		MI		L	FC
1	0	0	0	0	BASE0			BASE1			FC
1	0	0	0	1	RPC					L	FC
1	0	0	1	0	P3	P2	EM	BM		L	FC
1	0	0	1	1	RW					L	FC
1	0	1	0	0	WT					L	FC
1	0	1	0	1	NF			WI		L	FC
1	0	1	1	0	FIS			FD		L	
1	0	1	1	1	SHV						
1	1	0	RPS								
1	1	1	NAI								

在上述子字段中,对在数字滤波器实现中常用的子字段说明如下:

① M0,M1

指定 RAM0 和 RAM1 的寻址种类，如表 4－12 所示。

表 4－12　M0、M1 子字段指定内容

助记符		0，M1	功能
RAM0	RAM1	字段	
(NON)	(NON)	0　0	不进行指定
SPCBP0	SPCBP1	0　1	RAM 地址 = (BP)
SPCIX0	SPCIX1	1　0	RAM 地址 = (IX)
SPCBI0	SPCBI0	1　1	RAM 地址 = (BP) + (IX)

μPD77230 RAM 的寻址方式说明如下：

μPD77230 具有两个片内 RAM，通过 BP(基准指针)和 IX(变址指针)这两个指针寄存器进行寻址，如图 4－10 所示。从图中可以看出：寻址有三种方式：一是用 BP 寻址，二是只用 IX 寻址，三是用 BP 和 IX 内容之和寻址。同时，指针寄存器 BP 和 IX 均可进行加 1、减 1 操作。

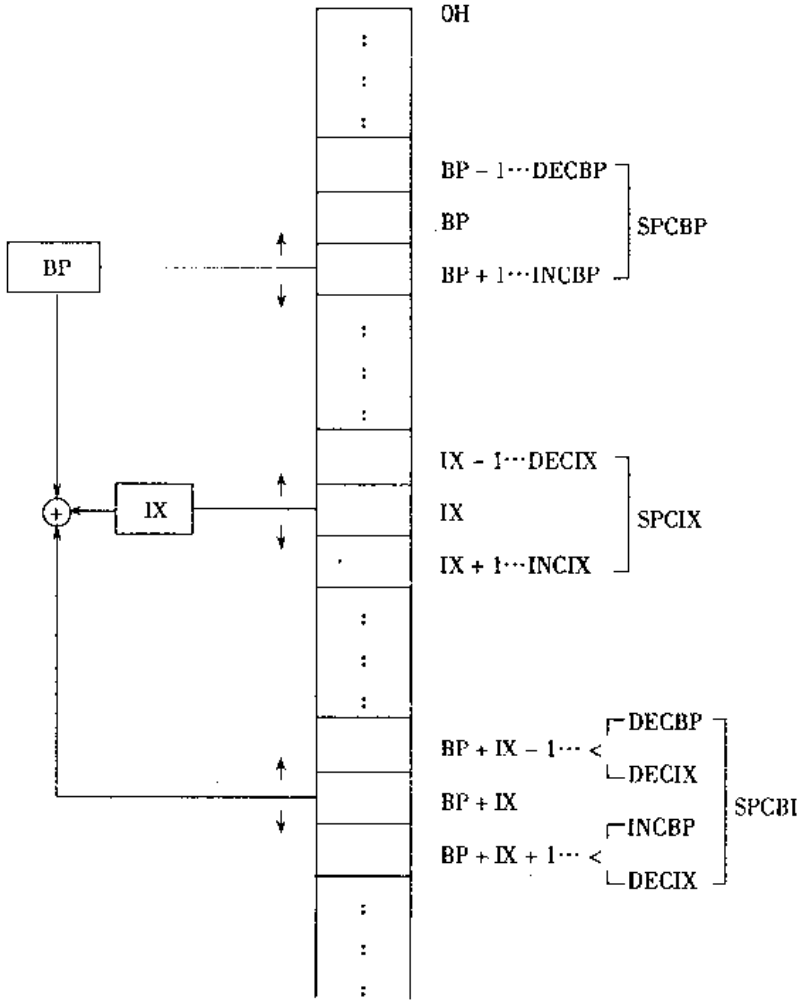


图 4－10　μPD77230 RAM 寻址

② DP0,DP1

RAM0,RAM1 寻址指针(BP 和 IX)的操作指定,如表 4 - 13 所示。

表 4 - 13 DP0,DP1 子字段指定内容

助记符		P0,DP1	功能
RAM0	RAM1	字段	
(NOP)	(NOP)	0 0 0	无操作
INCBP0	INCBP1	0 0 1	(BP) = (BP) + 1
DECBP0	DECBP1	0 1 0	(BP) = (BP) - 1
CLRBP0	CLRBP1	0 1 1	(BP) = 0
STIX0	STIX1	1 0 0	(IX) = (IX) + (BP)
INCIX0	INCIX1	1 0 1	(IX) = (IX) + 1
DECIX0	DECIX1	1 1 0	(IX) = (IX) - 1
CLRIX0	CLRIX1	1 1 1	(IX) = 0

③ RP

指定 ROM 的指针的操作。如表 4 - 14。

表 4 - 14 RP 的子字段指定内容

助记符	RP 字段	功能
	D22 D21	
(NOP)	0 0	无操作
INCRP	0 1	(ROM 指针) = (ROM 指针) + 1
DECRP	1 0	(ROM 指针) = (ROM 指针) - 1
INCBRP	1 1	(ROM 指针) = (ROM 指针) + 2^n

说明:表中 2^n 的 n 由 RPC 子字段进行指定。

④ NF

指定 OP 字段的 NORM 指令的种类。如表 4 - 15 所示。

表 4 - 15 NF 子字段指定内容

助记符	NF 字段	功能
	D21 D20 D19	
(NON)	0 0 0	无操作
TRNORM	0 1 0	浮点数规格化
RDNORM	1 0 0	浮点数上位 32 位舍入规格化
FSTFIX	1 1 0	将浮点数变为 24 位定点数
FIXMA	1 1 1	将 47 位尾数算术左移 N 位

说明:位移量置入 SVR 寄存器。

⑤ NAL

在运算的同时实现无条件转移指令时,以 9 位指定转移源地址。如表 4 - 16 所示。

表 4－16　NAL字段指定内容

助　记　符	NAL字段	功　　能
	D23……D15	
JBLKX	n为二进制数	无条件地转移到地址 n，0≤n≤511

⑥　L

指定循环计数器减1。如表4－17。

表 4－17　L字段指定内容

助　记　符	L字段	功　　能
	D23	
(NOP)	0	无操作
DECLC	1	(循环计数器)=(循环计数器)－1

⑦　EM，BM

指定中断允许及中断存储的有无，如表4－18所示。

表 4－18　EM，BM子字段指定内容

助　记　符		EM	BM	功　　能
EM	BM	D19	D18 D17	
(NOP)	(NOP)	0	0　0	无操作，保持当前状态
(NOP)	CLRBM	0	0　1	不记忆中断
(NOP)	BETBM	0	1　0	记忆中断
DI	(NOP)	0	1　1	中断禁止
BI	(NOP)	1	0　0	中断许可
EI	CLRBM	1	0　1	中断许可，不记忆
EI	SETBM	1	1　0	中断许可，记忆
		1	1　1	禁用！

2. 直接置数指令

将IM字段指定的数直接置入由DST字段指定的寄存器或指针中。IM字段长24位。

3. 转移指令

转移到或调用由NA字段指示的指令的地址。或者从中断处理或子程序返回到堆栈存储器指示的地址。

与运算指令一样，用SRC字段和DST字段可同时进行数据传送。转移种类由C字段指定如表4－19所示。

表 4－19　C字段指定内容

助　记　符	C　字　段					转移种类(条件)
	D14	D13	D12	D11	D10	
JMP	0	0	0	0	0	无条件转移
CALL	0	0	0	0	1	调用子程序
RET	0	0	0	1	0	从调用子程序或中断返回
JNZRP	0	0	0	1	1	ROM指针≠0时转移
JZ0	0	0	1	0	0	PSW0的Z标志为1时转移

JNZ0	0	0	1	0	1	PSW0 的 Z 标志为 0 时转移
JZ1	0	0	1	1	0	PSW1 的 Z 标志为 1 时转移
JNZ1	0	0	1	1	1	PSW1 的 Z 标志为 0 时转移
JC0	0	1	0	0	0	PSW0 的 C 标志为 1 时转移
JNC0	0	1	0	0	1	PSW0 的 C 标志为 0 时转移
JC1	0	1	0	1	0	PSW1 的 C 标志为 1 时转移
JNC1	0	1	0	1	1	PSW1 的 C 标志为 0 时转移
JS0	0	1	1	0	0	PSW0 的 S 标志为 1 时转移
JNS0	0	1	1	0	1	PSW0 的 S 标志为 0 时转移
JS1	0	1	1	1	0	PSW1 的 S 标志为 1 时转移
JNS1	0	1	1	1	1	PSW1 的 S 标志为 0 时转移
JV0	1	0	0	0	0	PSW0 的 OVFM 标志为 1 时转移
JNV0	1	0	0	0	1	PSW0 的 OVFM 标志为 0 时转移
JV1	1	0	0	1	0	PSW1 的 OVFM 标志为 1 时转移
JNV1	1	0	0	1	1	PSW1 的 OVM 标志为 0 时转移
JEV0	1	0	1	0	0	PSW0 的 OVFE 标志为 1 时转移
JEV1	1	0	1	0	1	PSW1 的 OVFE 标志为 1 时转移
JNFS1	1	0	1	1	0	SI 寄存器数据输入未结束时
JNFS0	1	0	1	1	1	SO 寄存器的数据输出未结束时
JIP0	1	1	0	0	0	输入口 P0 为 1 时转移
JIP1	1	1	0	0	1	输入口 P1 为 1 时转移
JNZIX0	1	1	0	1	0	(IX0) = 0 时转移
JNZIX1	1	1	0	1	1	(IX1) = 0 时转移
JNZBP0	1	1	1	0	0	(BR0) = 0 时转移
JNZBP1	1	1	1	0	1	(BR1) = 0 时转移
JRDY	1	1	1	1	0	与外部存储器数据输出结束时
JRQM	1	1	1	1	1	屏蔽操作的 DRS 数据输入输出结束时

为使各类指令均可在一个机器周期内完成，可对字段指定的控制进行并行操作。例如，在计算：

$$\sum_{i=1}^{N-1} a(i) * x(j-i)$$

时，若各字段按：

(1) OP 字段指定为浮点加；

(2) P 字段指定为 M、Q 字段指定 WRn；

(3) SRC 指定系数存储在 DATAROM，DST 指定由 RAM 向 FMPY 寄存器传送数据；

(4) CON 字段指定 DATAROM 和 RAM 地址；进行指定时，可同时进行累加 FMPY 的输出、FMPY 对 K、L 寄存器的数据进行乘算、DATAROM 向 FMPY 送运算数据、RANM0 和 RAM1 向 FMPY 送数据、DATAROM 更新地址、RAM0、RAM1 更新地址等 6 种不同的操作。

三、 μPD77230 应用

作为 μPD77230 的应用，这里举用 μPD77230 汇编语言实现数字滤波器的汇编程序例。为通用起见，选择数字滤波器的基本二阶节，其信号流图如图 4 - 11 所示。图中的 C1 ~ C8 为信号流图中的节点。

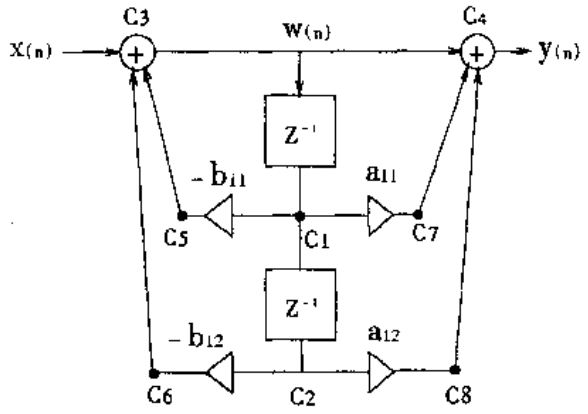


图 4-11　数字滤波器 2 阶节

1. 优先格式法

在编写汇编语言程序时，常采用优先格式法安排计算顺序。优先格式法的说明如下：

(1) 从图 4-11 的信号流图中拿掉延迟单元(Z^{-1}单元)，如图 4-12 所所示。

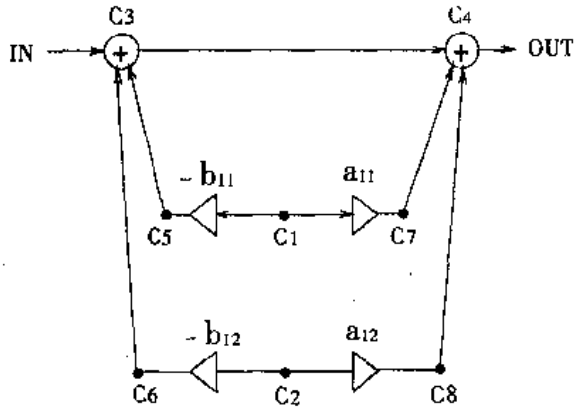


图 4-12　优先格式示意图

(2)设集合{N1}为信号流图中输入节点和延迟器输出节点的集合。计算这些节点的值并保存在存储器或寄存器内。

(3)取由{N1}包含节点可计算值的节点的集合，并设定为集合{N2}。

(4)取由{N1}∪{N2}所含节点可计算值的节点并设定为集合{N3}。

(5)取出{N1}∪……∪{Nn}所含节点中可计算值的节点并设为集合{Nn}。

(6)重复(5)的操作，直到集合{Nk}中无节点为止。

例如，图 4-11，可进行：

第一步：求集合{N1}，由设定得出：

{N1} = IN, C1, C2

第二步：仅使用包含在{N1}中节点的值可计算值的节点为：

C5 = (-b11) × C1

C6 = (-b12) × C2

$C7 = (a11) \times C1$

$C8 = (a12) \times C2$

因而{N2}为：

$\{N2\} = C5, C6, C7, C8$

第三步：由$\{N1\} \cup \{N2\} = IN, C1, C2, C5, C6, C7, C8$的值可求得的节点为：

$C3 = IN + C5 + C6$

因而 $\{N3\} = C3$

第四步：从$\{N1\} \cup \{N2\} \cup \{N3\}$的值，寻找可求得值的节点为：

$C4(out) = C3 + C7 + C8$

因而

$\{N4\} = C4 \qquad (K = 4)$

C4 为 OUT，即信号输出。

根据图 4－11，节点集合{N1}～{N4}分布如图 4－13 所示。

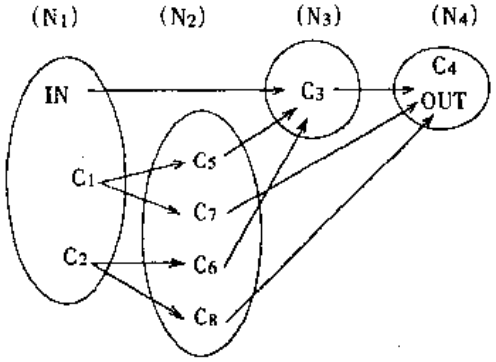


图 4－13 优先格式例

2. 程序编写

编写程序时，首先要考虑乘算系数的处理。在 μPD77230 中，RAM 和乘法器之间连有专用的数据总线，如图 4－14 所示。

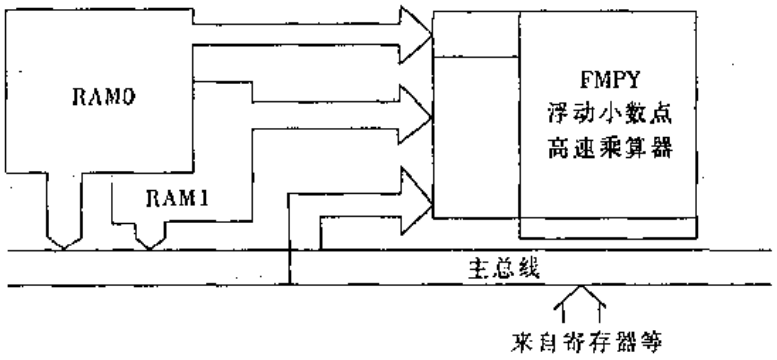


图 4－14 μPD77230 乘法器输入

当用一条传送指令指定乘算对象时，可选择(RAM)×(ROM)、(RAM)×(寄存器)或(RAM)×(RAM)等形式。在下述的程序例中仅使用了(RAM)×(ROM)型乘法运算，但也可以使用(ROM)×(寄存器)型运算。

这样，将乘法系数和延迟器的值，首先存储在存储器内，当如图 4－15 分配地址空间时，用

汇编语言编写的程序可如 LIST1 和 LIST2 所示。

程序和乘算系数在交叉汇编输入形成的源文件内，在个类存储器中是分段记述的，记述格式如图 4－16 所示。

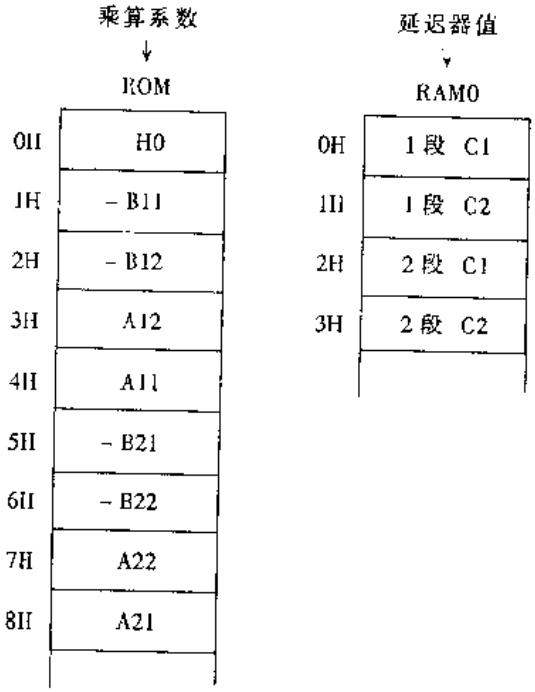


图 4－15　内存分配图

LIST1：汇编语言程序

```
        NAME BIQUAD2
H0      EQU     6.40146E－2          ;滤波器系数
B11     EQU     8.08283978E－1
B12     EQU     －7.94035032E－1
A11     EQU     －1.32808321E－1
A12     EQU     1.00000000E0
B21     EQU     9.30053038E－1
B22     EQU     －3.39527195E－1
A21     EQU     1.22533478E－0
A22     EQU     1.00000000E0

        ROMSEG AT 0H
        DW      H0
        DW      B11
        DW      B12
        DW      A12
        DW      A11                  ;ROM 内容
        DW      B21
```

```
        DW      B22
        DW      A22
        DW      A21
BIQUAD MACRO INP OUT                                        ;C3 = INPUT
        CLR   WR&OUT     MOV  LKR0,ROM   INCRP INCBP0   ;C4 = 0,C5 = ( - b11) * C1
        ADDF  WR&INP,M   MOV  LKR0,ROM   INCRP          ;C3 = C4 + C5 C6=(-b12)*C2
        ADDF  WR&INP,M   MOV  LKR0,ROM   INCRP DECBP0   ;C3 = C3 + C6 C8 = (a12) * C2
        NORM  WR&INP                                TRNORM ;C3 规一化
        ADDF  WR&OUT,M   MOV  LKR0,ROM   INCRP          ;C4 = C4 + C8 C7 = (a11) * C1
        ADDF  WR&OUT,IB  MOV  RAM0,WR&INP    INCBP0     ;C4 = C4 + C3 C1 < = C1
        ADDF  WR&OUT,M   MOV  RAM0,K         INCBP0     ;C4 = C4 + C7 C2 < = C1 (k)
        ENDM

        IMSEG       AT 0H
        LDI         SR,001A0H     ;状态寄存器设定
        EI CLRBM
WAIT:JMP        WAIT
        NOP
        IMSEG       AT 100H     中断开始地址
                    MOV     L,SI        SPCBP0      CLRBP0      ;串行输入
        LDI         BP,000
        CLR         WR0     MOV     K,ROM       INCRP
        ADDF        WR0,M
        BIQUAD 0 1
        BIQUAD 1 0
        NORM WR0                                    FLTFIX      ;浮点→定点
        RET;
                    MOV     S0,WR0  EI          CLRBM       ;串行输出
        END;

LIST2:
                NAME BIQUAD2
H0      EQU         6.40146E - 2
B11     EQU         8.08283978E - 1
B12     EQU         - 7.94095032E - 1
A11     EQU         - 1.32906321E - 1       ;滤波器系数
A12     EQU         1.00000000E0
B21     EQU         9.30053038E - 1
B22     EQU         - 3.39527195E - 1
A21     EQU         1.22533478E - 0
```

```
A22     EQU     1.00000000E0

        ROMSEG  AT 0H
        DW      H0
        DW      B11
        DW      B12
        DW      A12
        DW      A11             ;ROM 内容
        DW      B21
        DW      B22
        DW      A22
        DW      A21
        IMSEG   AT 0H           ;设定状态寄存器
        MOV     SR,001A0H
        SET     EI
        CLR     BM
WAIT:JMP        WAIT
        IMSEG   AT 100H         ;中断开始地址
        SPC     BP0             ;[RAM0] = C1
        CLR     BP0             ;[ROM] = H0
        MOV     RP,000          ;串行输入
        MOV     L,SI            ;
        MOV     K,ROM+          ;INPUT < = [H0] * INPUT [ROM] = B11
        MOV     WR0,M           ;C3 < = INPUT
        MOV     LKR0,ROM+       ;C5 < = ( - B11) * C1 [ROM] = - B12
        INC     BP0             ;[RAM] = C2
        ADDF    WR0,M           ;C3 < = C3 + C5
        MOV     LKR0,ROM+       ;C6 < = ( - B12) * C2 [ROM] = A12
        ADDF    WR0,M           ;C2 < = C3 + C6
        MOV     WR1,WR0         ;C4 < = C3
        MOV     LKR0,ROM-       ;C8 < = (A12) * C2
        DEC     BP0             ;
        ADDF    WR1,M           ;C4 < = C4 + C8
        MOV     LKR0,ROM+       ;C7 < = (A11) * C1 [RAM] = - B21
        INC     BP0             ;[RAM] = C1
        MOV     RAM0,K          ;C2 < = C1
        DEC     BP0             ;[RAM] = C1
        ADDF    WR1,M           ;C4 < = C4 + C7
        MORM    WR0,IR          ;C3 规格化
        MOV     RAM0,WR0        ;C1 < = C3
```

```
INC     BP0             ;
INC     BP0             ;[RAM] = C1
MOV     LKR0,ROM +      ;C6 < = ( - B21) * C1 [ROM] = - B22
INC     BP0             ;[RAM] = C2
ADDF    WR1,M           ;C3 < = C3 + C5
MOV     LKR0,ROM +      ;C6 < = ( - B22) * C2 [ROM] = A22
ADDF    WR1,M           ;C3 < = C3 + C6
MOV     WR0,WR1         ;C4 < = C3
MOV     LKR0,ROM +      ;C8 < = (A22) * C2 [ROM] = A21
DEC     BP0             ;[RAM] = C1
ADDF    WR0,M           ;C4 < = C4 + C8
MOV     LKR0,ROM        ;C7 < = (A21) * C1
INC     BP0             ;[RAM] = C2
MOV     RAM0,K          ;C2 < = C1
DEC     BP0             ;[RAM] = C1
ADDF    WR0,M           ;C4 < = C4 + C7
MORM    WR1,TR          ;规一化 C3
MOV     RAM0,WR1        ;C1 < = C3
NORM    WR0,FF          ;浮点→定点
MOV     S0,WR0          ;串行输出
SET     EI
CLR     BM
RET
END
```

```
NAME   滤波器名……必须书写
HO     EQU   6.40146E - 2
         :
         ;
……定义系统常数

ROMSEG  AT  0H ……开始写 ROM 内容
        DW    H0
         :
         ;
……按 ROM 数值地址顺序书写

IMSEG  AT  0H ……开始指令存储器
WAIT:  JNFSI  WAIT
        NOP
        MOV   WR0,SI
         :
         ;
……写入编写的程序
```

图 4 - 16　源文件记述格式

3.μPD77230 软件开发环境

日本电气提供的用在 CP/M－86 上的 μPD77230 汇编程序库。

(1)交叉汇编程序(含预汇编程序)

由汇编源程序或预汇编源程序生成浮动目标模块,同时输出显示代码状况的列表文件。预汇编时,能输出汇编源程序。

源程序可用宏描述。

(2)链接

连接浮动目标模块生成装入模块,并能连接在库文件的目标模块,同时输出地址分配状况的列表文件。

(3)程序库管理文件

将浮动目标模块群同一连接生成参考库文件。在库文件内,以模块为单位进行追加、删除等同一管理。

(4)目标转换器

将装入模块变换成标准的 HEX 形式的文件,同时可输出符号表文件。这些文件可用做硬件仿真器(EVAKIT－77230)的调试输入。标准 HEX 形式的文件在制造淹膜 ROM 时使用。

4. 调试环境

(1)硬件仿真器

日本电气推出了 EVAKIT－μPD77230 仿真器。EVAKIT－77230 做为主机可以和 VAX 等连接。同时,使用日本电气制作的 PROM 写入器,可将调试完的程序写入 μPD77230 的 EPROM 内。这时,程序(指令码)和乘算系数均以 HEX 形式的文件传送。

使用这种仿真器时应特别注意的是,仿真器每一步都可以执行指令,但由于流水线不能中断,最好进行继续调试。

(2)软件仿真器

日本电气推出了 VAX/Unix 库,其指令体系与 EVAKIT－77230 类似。这时,程序可以用装入模块形式输入和输出,操作非常方便。

另外,硬件的输入输出控制信号也可以用软件进行仿真。

4.3 其它数字信号处理器简介

一、MSM6992

MSM6992 是第二代 DSP,芯片构成方框图如图 4－17。

1.硬件特征

(1)浮点数形式

数据采用 22 位浮点数,阶码 5 位,尾数 16 位。动态范围 470dB。

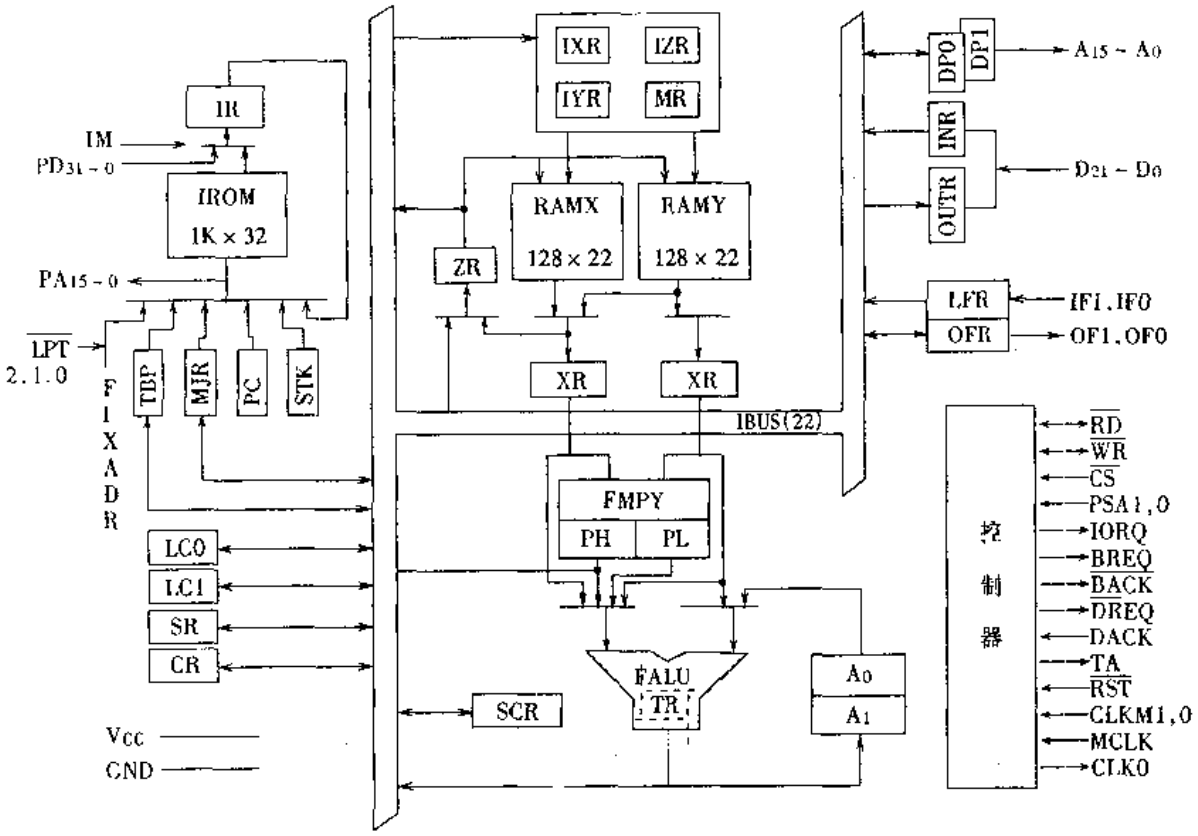


图 4 - 17　MSM6992 构成方框图

(2)高处理能力

机器周期 100ns,最大运算能力 20MFLOPS。内部存储器、乘法器和 ALU 采用流水线结构。1024 点复数 FFT 的执行时间为 6.8ms。

(3)大容量存储器

片内数据存储器 RAM 为 2 × 128 字 × 22 位,程序 ROM 为 1K × 32 位。

(4)扩展性

扩展性强,可构成大规模处理系统。除片内存储器外,可扩展外部 64K × 32 位的程序存储器和 64K × 22 位的数据存储器。

可构成多处理器系统。通常由 1 台主 DSP 和多台从 DSP 连接形成主从系统。微处理器的主存储器和 DSP 之间可进行 DMA 高速数据传送。

(5)低消耗功率

MSM6992 采用 2μm CMOS 技术,消耗功率 400mW。

2.开发支援工具

MSM6992 开发支援系统特征如下:

(1)备有语声处理系统用的高功能交叉宏汇编程序;

(2)用于大规模程序开发的连接和库管理程序；

(3)具有大存储器容量的程序调试用仿真器；

(4)上述开发工具适用于 MS－DOS，可在 IBM PC 机上使用。

二、MB 8764

MB 8764 是富士通公司推出第二代 DSP 产品，构成方框图如图 4－18。

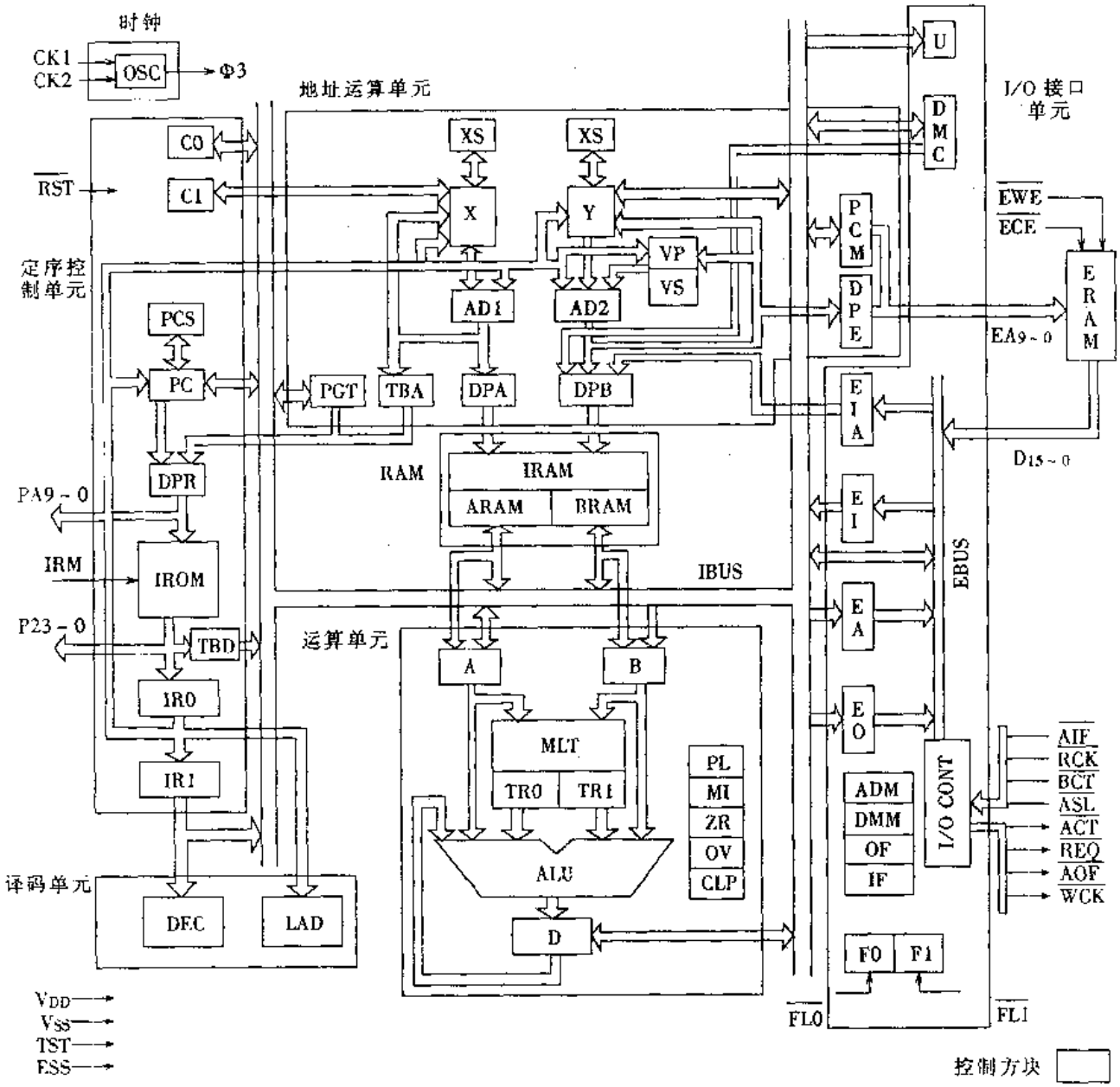


图 4－18　MB8764 方框图

1.MB 8764 的特征

(1)高速运算处理

MB8764 内装 16 位并行乘法器和 26 位 ALU，通过流水线连接，可在 100ns 指令周期内完成一组乘、累加运算。

(2)RAM

内装两组数据存储器，即具有专用地址运算电路的 128×16 位的 RAM(ARAM 和 BRAM)。这两个 RAM 可合在一起用做 256 字的 RAM，称为 IRAM。

地址运算电路的结果由引脚 EA9～EA0 输出，用这个信号可外接 1K RAM 实现存取。

RAM 的寻址，可采用由指令进行直接寻址及由指令指定的值及变址寄存器值之和为地址的变址寻址方法。

(3)ROM

MB8764 内有 1K 字的掩膜 ROM，用于存储程序和常数数据。通过地址指定引脚 PA9～PA0 和程序数据输入引脚 P23～P0，可使外部存储器和片内掩膜 ROM 一样使用。

(4)输入输出方法

输入输出可用 16 位串行接口，输入、输出各有三种模式。

(5)微指令

MB8764 有 15 类运算指令，这些指令和传送指令、分支指令组合，可进行并行处理。

三、DSP96002

DSP96002 是 HCMOS 系列芯片，是 32 位通用浮点 DSP。DSP96002 适用于数据密集的快速浮点运算，可对大量的存储器子系统进行存取。DSP96002 工作频率 27MHz，运算能力达 40.5MFLOPS。

DSP96002 在结构上具有双重性：两个独立的数据存储器空间；两个地址运算单元；两个在片 DMA 控制器和双总线结构，如图 4～19 所示。

正是由于这种双重性，使之在数据处理的应用中可容易的写入软件。例如：数据自然的划分为 X 和 Y 坐标，分别应用在图形和图像处理中。为了进行复数运算，又将数据分为实数部分和虚数部分。

DSP96002 在结构上显示出高度的并行性—三浮点运算、两数据传送和在单指令周期内实现两地址指针校正。

CPU 由三个 32 位并行操作执行单元组成。数据 ALU 执行所有算术运算(定点和浮点)和逻辑运算，包括五部分——一个格式转换单元、一个存放 ALU 运算结果的通用寄存器、一个浮点乘法器、一个浮点加/减单元(这个浮点加/减单元包括一个 32 位定标移位器)，一个特殊功能单元。第二个并行单元是地址生成单元(AGU)，它完成存储器的地址存储及运算。第三是程序控制器，它完成存储器之间直接数据传送(无需 DSP96002 中央控制)。该单元由程序地址发生器、程序译码控制器和程序中断控制器组成。

DSP96002 存储器包括 1024 字的数据 RAM(等分为 X、Y 数据存储器)、全速在片程序 RAM 和两个预编程数据 ROM。在片引导 ROM 使用户程序可方便的装入程序 RAM，两个独立的扩展总线的端口便于与 SRAM，DRAM 和 VRAM 相接。

用户目前可使用软件开发程序包 DSP96000CLASX 开发适用于 DSP96002 的汇编语言。联接后可进行全面测试。程序包包括一个仿真器，一个汇编程序，一个连接带和可调用的程序卡。

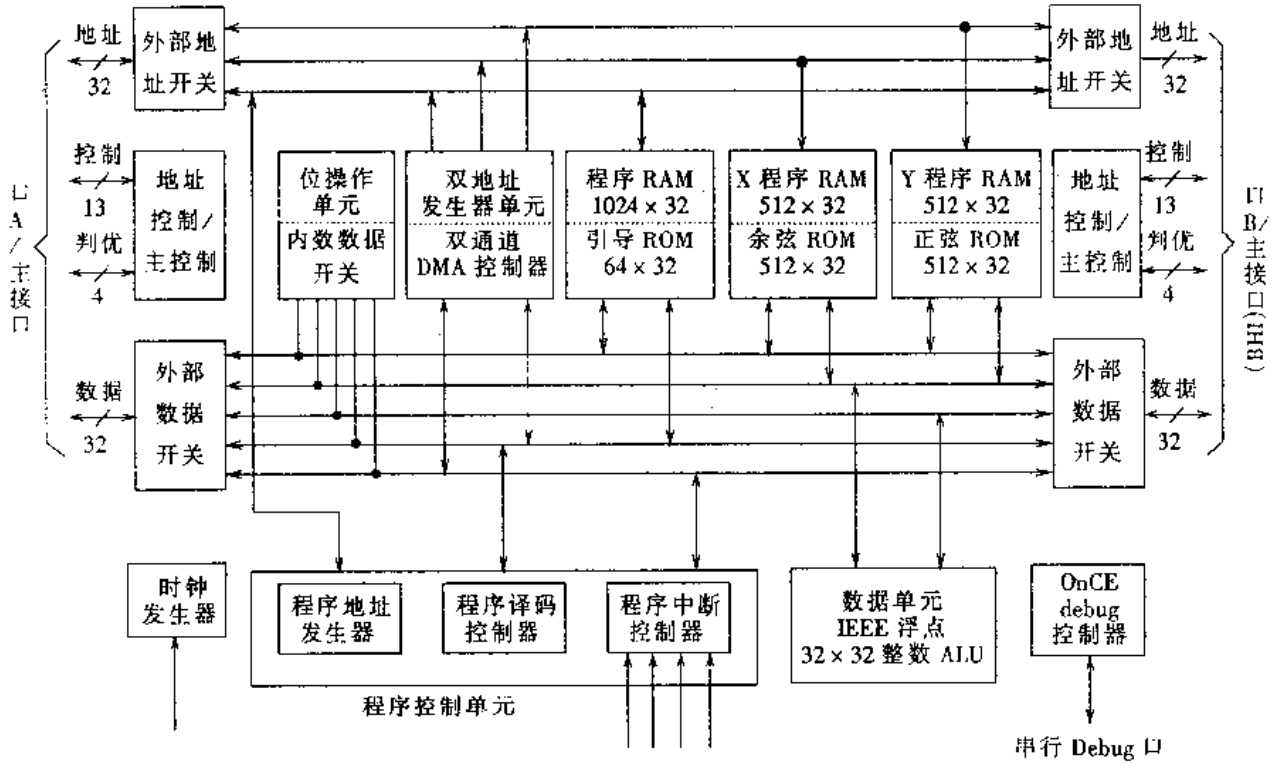


图 4－19　DSP96002 结构框图

四、DSP56000/56001

DSP56000/56001 是一种高速微调节器，也是大功率 DSP(图 4－20)。DSP56001 具有 512×24 位的程序 RAM。可将 2K×8 位的 EPROM 或主处理器的内容送到 RAM 内。DSP56000 提供一个 3.75K×24 位的程序 ROM，并可由用户编程作为专用标准系统。DSP56000/DSP56001 包含两个独立的能同时在一个指令周期内存取数据的数据存储空间。DSP56000/DSP56001 另一特点是具有硬件乘法/累加器，可实现两个 24 位数据的乘法，并将 48 位的结果加到 56 位累加器。两个存储器在同一个指令周期内可同时完成存取操作。为了更快的进行输入/输出(I/O)操作，DSP56000/DSP56001 有三个在片辅助装置，即主机接口(HI)、同步串行接口(SSI)和串行通信接口(SCI)。当 SCI 为非通信用途时，它的波特率发生器可做为通用定时器。这些辅助设备使得 DSP56000/DSP56001 具有高达 24 个通用 I/O 线。因此，DSP56000/DSP56001 可用于许多实时计算控制装置中，例如：电唱机驱动、电机控制、计算机控制的悬挂系统、有源干扰的消除和遥控设备。

这些不同的功能信号使 DSP56000/DSP56001 更适用于数字控制系统。DSP56000/DSP56001 提供了三种在片外围设备接口，即一个 8 位的并行主 MPU/DMA 接口、一个 SCI 接口和一个 SSI 接口。同时，可用 24 个通用 I/O 引线端，指定使用哪一个在外围设备接口。

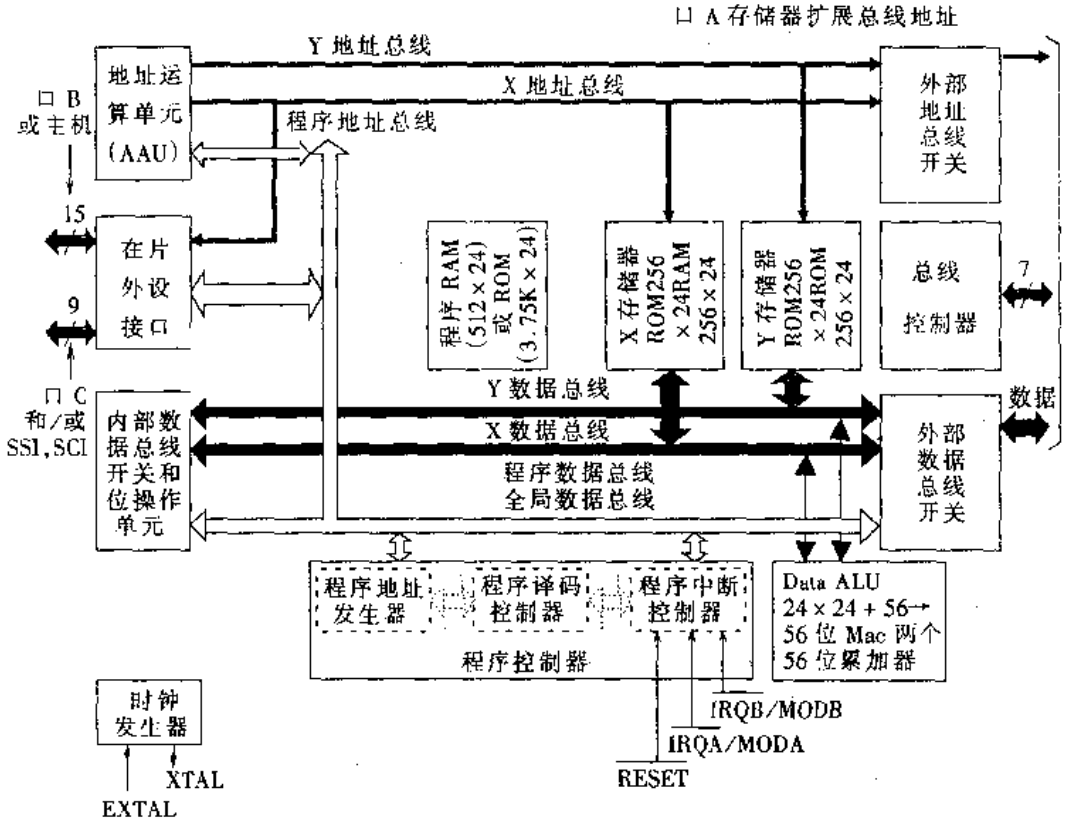


图 4－20　DSP56000/DSP56001 的构成方框图

DSP56000/DSP56001 引脚信号分类如图 4－21 所示。

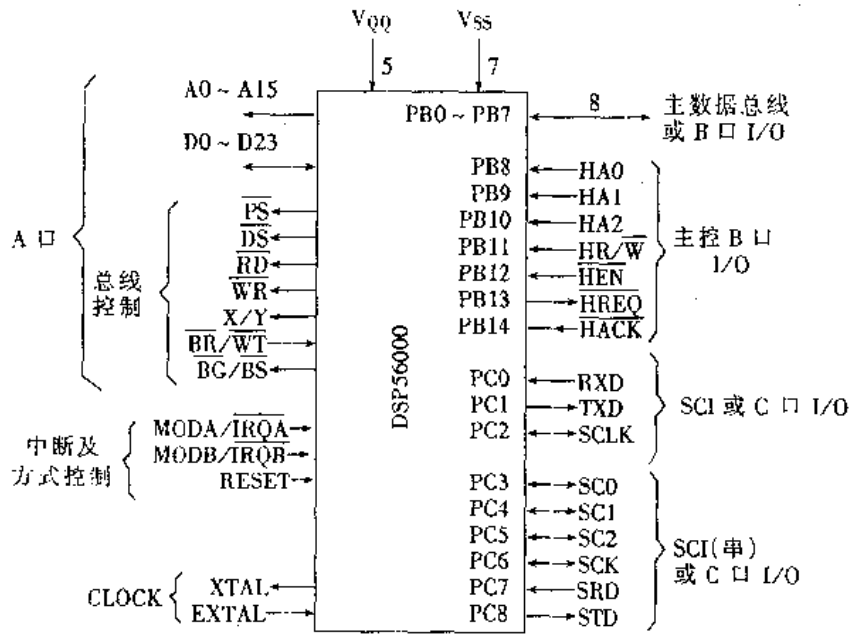


图 4－21　DSP56000/DSP56001 管脚功能

1. 主机接口

56000/56001 主机接口是一个专用 8 位并行口，用于直接主微处理器或 DMA 控制器。图 4－22 和图 4－23 分别表示 DSP56000/DSP56001 主机接口与 MC68000 和 MC68HC11 连接的电路。

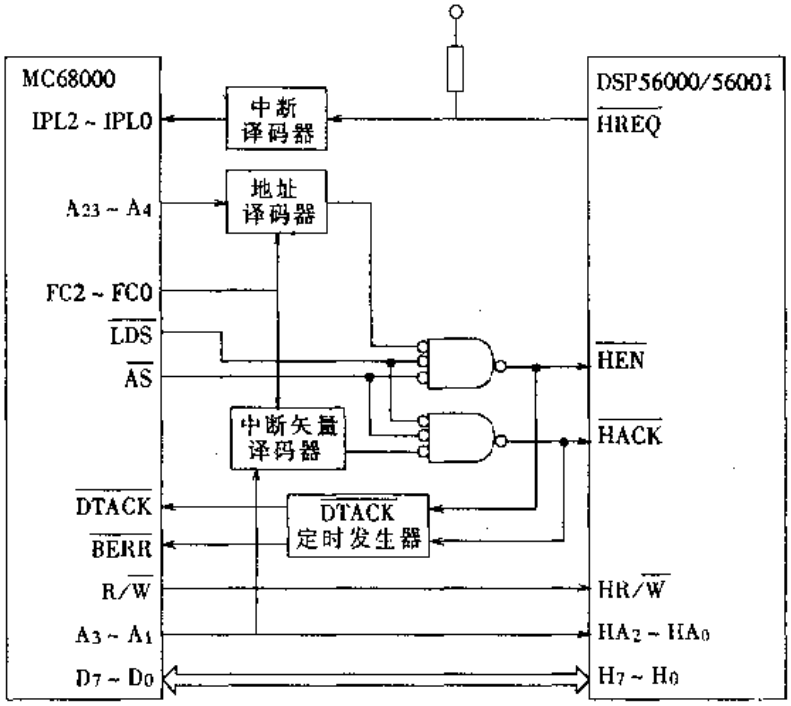


图 4－22　DSP56000/DSP56001 与 MC68000 连接

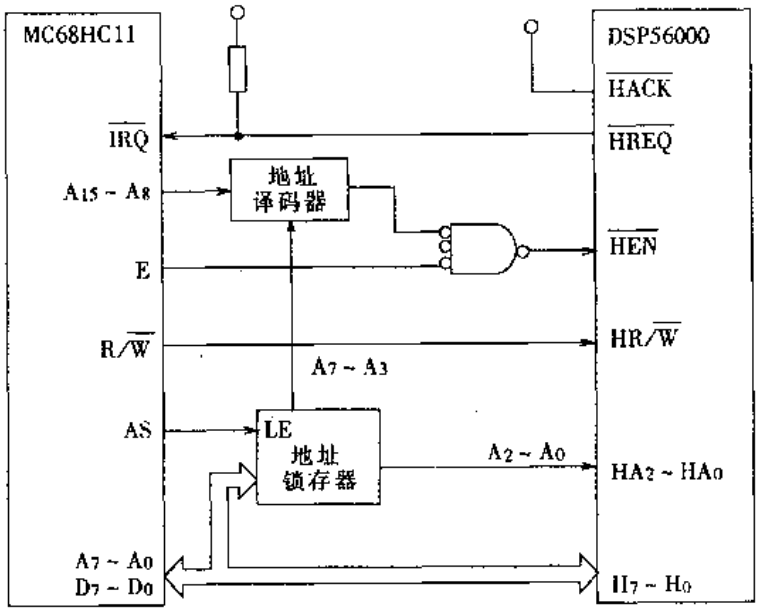


图 4－23　DSP56000/DSP56001 与 MC68HC11 连接

主处理器能够响应诸如监控面板开关和 DSP 内启动各种控制顺序等高级操作。主机接口的一个优点是主机指令选择，如果需要，主处理器可以执行 DSP56000/DSP56001 的 32 个中断矢量的任意一中断矢量。图 4－24 为 DSP56000/DSP56001 块速中断矢量，其中包括主机接口可以实现的功能。

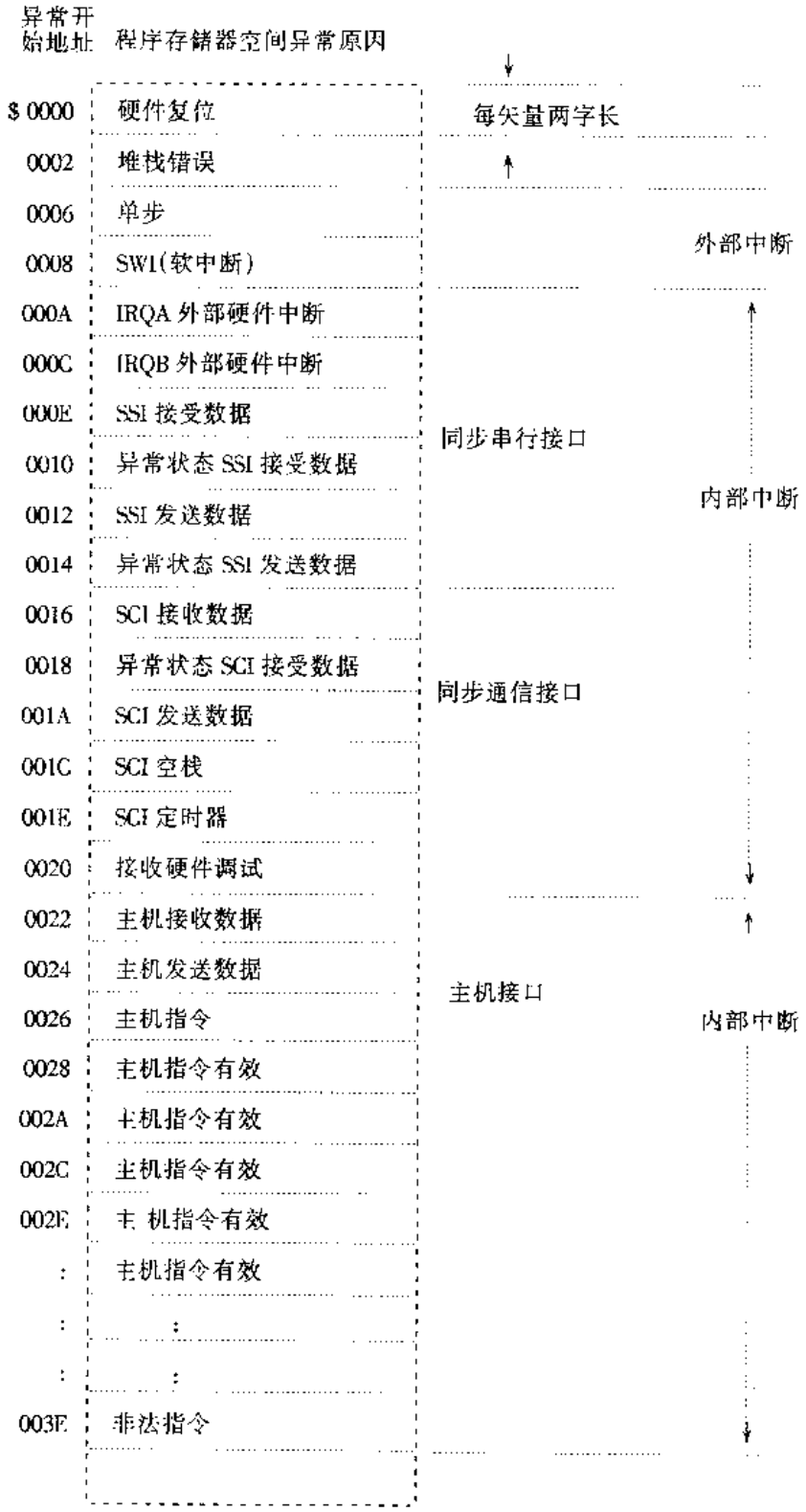


图 4－24 DSP56000/DSP56001 快速中断矢量

置数到指令矢量寄存器将使 DSP 执行其对应的快速中断。十二个主机指令矢量可用于通用的程序设计。如果未使用任一剩余预置中断，如软件中断或 SSI 空闲中断，则还可以使用通有用指令矢量。这些主机指令矢量给系统设计者提供了更多的灵活性。程序 LIST1 为使用主机接口的 PID 调节器程序例：

LIST1：主机接口程序

```
                include 'declare.dat'
input       equ     $fffe
output      equ     $ffff
* * * * * * * * * *
* X 存储器说明 *
* * * * * * * * * *
        org   x:0
b-0     ds   1      ;b(0)系数
b-1     ds   1      ;b(1)系数
b-2     ds   1      ;b(2)系数
a-1     ds   1      ;a(1)系数
a-2     ds   2      ;a(2)系数
* * * * * * * * * *
* 定义快速中断 *
* * * * * * * * * *
            org     p:reset             ;RESET 服务程序单元
            jmp     main                ;主程序开始
            org     p:hc1               ;HC1 服务程序单元
            movep   x:hrx,x:b-0         ;更新系数 b0
            nop                         ;快速中断第二字
            org     p:hc2               ;HC2 服务程序单元
            movep   x:hrx,x:b-1         ;更新系数 b1
            nop                         ;快速中断第二字
            org     p:hc3               ;HC3 服务程序单元
            movep   x:hrx,x:b-2         ;更新系数 b2
            nop                         ;快速中断第二字
            org     p:hc4               ;HC4 服务程序单元
            movep   x:hrx,x:a-1         ;更新系数 a1
            nop                         ;快速中断第二字
            org     p:hc5               ;HC5 服务程序单元
            movep   x:hrx,x:a-2         ;更新系数 a2
            nop                         ;快速中断第二字
* * * * * * * * * *
* 主程序：(Main Routine) *
* * * * * * * * * *
```

```
        org     p:main                  ;主程序单元
        movep    # $0A00,x:ipr           ;启动主接口中断
        movep   #0,x:bcr                ;无等待状态
        movep   # $04,x:hcr              ;启动主指令中断
        movep   #1,x:pbc                ;主接口运行
        andi    # $FC,mr                ;启动全部中断
start
        movep   y:input,x0              ;从A口取采样输入
        jst     filter                  ;滤波器采样
        movep   a,y:iuput
        jmp     start                   ;重复
```

在这个例子中，主机指令矢量用来实时变更 PID 参数而无需重新编译程序或暂停 DSP56000/DSP56001。参数可以从前面板或以软件菜单形式输入到主机。首先，主处理器将新的 PID 参数写入主机接口主处理器侧的 TXH、TXM 或 TXL 寄存器。然后，主机必须将适当的主机指令编号写入 DSP 指令矢量寄存器(CVR)，例如，改变 b0，则数值 $92 必须写入 CVR，同时置 IIC 位，并使 DSP56000/DSP56001 启动主机指令。一旦 DSP 识别主机指令矢量未决，将转移到地址 $24，并执行相应的快速中断。

2. SCI 接口

SCI 提供一个与其他 DSP、微处理器等采用直接通信或通过调制调器进行异步串行通信的接口。SCI 具有使用标准异步位比率和协议通讯的功能，例如进行高速异步数据传送。异步通信口还包含一个带有唤醒空线和唤醒寻址能力的主从操作复合方式，在一些控制系统中，SCI 接口还提供一种廉价的标准通信方法，即使用 DSP56000/DSP56001 的一个终端及一个简单的监控程序。

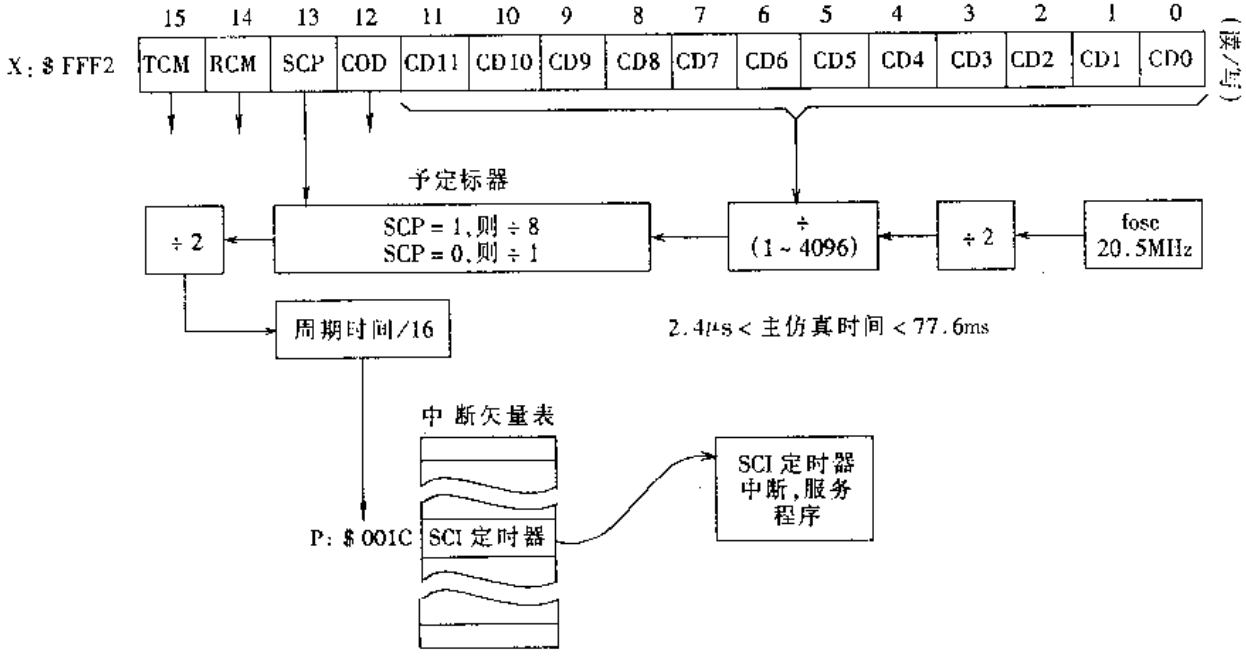


图 4-25　SCI 可编程定时器功能图

如果在异步通信定时器中不使用 SC1 接口波特率发生器，那么，波特率发生器可用做通用的定时器。图 4－25 表示用 SC1 时钟控制寄存器的 SC1 可编程定时器的结构，系统时钟首先被 2 除，然后将相应值存入到几个 CD 位，这个时钟再被 1－4096 的数整除。预置刻度位允许程序员进一步将时钟除以 8。最后，时钟再除以因子 2 和因子 16。SC1 定时器的分辨率△t 将为 $32t_i \leqslant \triangle t_i \leqslant 1048576\triangle t_i$，这里 t_i 是 DSP 的指令周期时间。由于 DSP56000/DSP56001 为 27MHz，因而定时器分辨率为 $2.4\mu s \leqslant \triangle t \leqslant 77.6ms$。对 DSP56000/DSP56001 进一步观察，图 4－25 中最后除以 16 的方框是任选的，进一步降低定时器的分辨率到 $0.6t_i$ 的最小中断速率。这样，在 27MHz 时钟下，定时器分辨率进一步修正为 444ns

3.SSI 接口

SSI 口可以接收和传送串行 A/D、D/A 和 CODE 的杂乱或无关联逻辑。SSI 口是一个全双工六引脚接口，它包括一个串行接收引脚，一个串行时钟引脚和三个帧同步/输出标志引脚，全双工接口和可编程帧同步可用于选通接收数据和传送数据。同时，SSI 接口允许 DSP56000/DSP56001 在一个网络中进行多处理器间串行通信。在联网方式 1，允许在 32 个不同时间段和其他处理器或 I/O 设备进行通信。

4.通用 I/O 引脚

HI、SSI 和 SCI 中任何一个接口均可用做通用 I/O 引脚。通用控制寄存器与 B 通道和 C 通道均有关联，并允许通道引脚被选择为通用 I/O 引脚或者专用外设引脚。B 通道的全部 15 个引脚必须设定为 I/O 引脚或者 HI 引脚。这样，C 通道的所有的专用引脚在接口时均是分离的。当不使用 SSI 的串行发送器时，PC8 可用做 I/O 引脚，PC3－PC7 则完成它们指定的 SSI 功能。通过对相应的通道数据寄存器进行存取来把一个通道引脚选为通用 I/O 引脚，写入通道寄存器的数据被锁存。

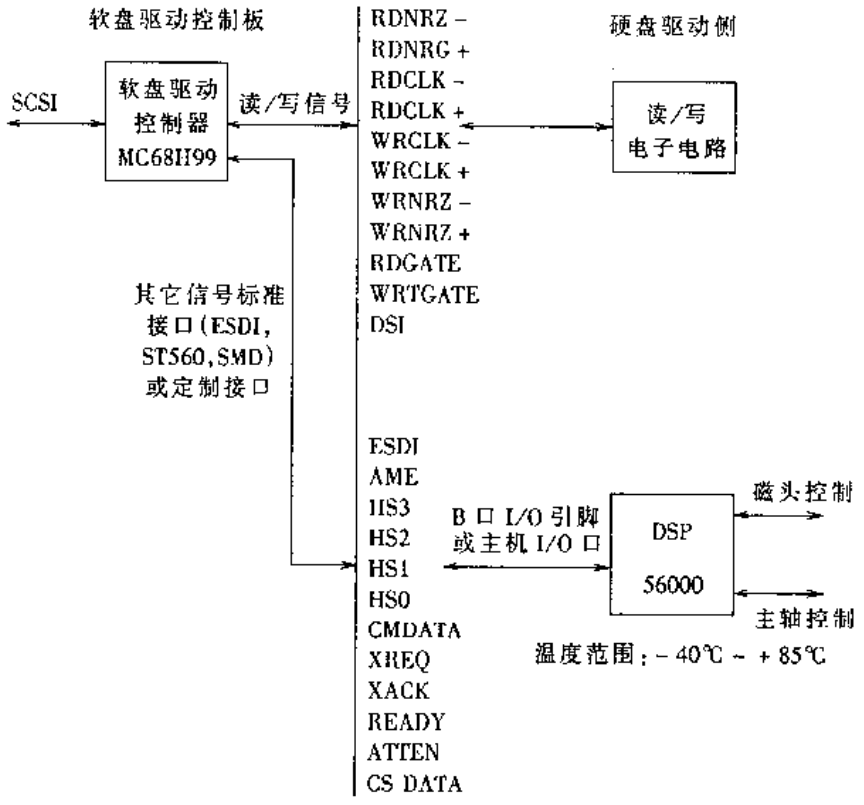
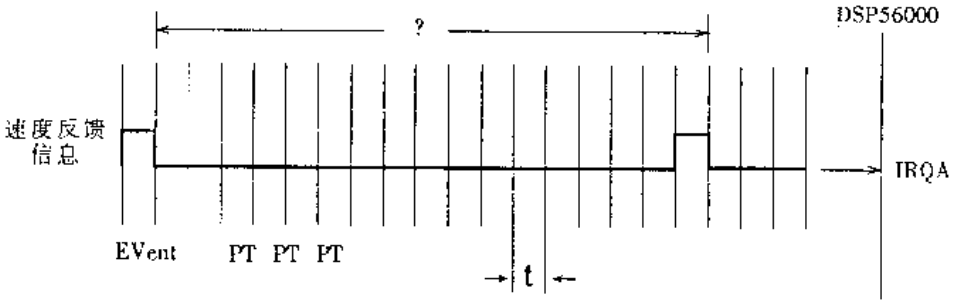


图 4－26 DSP56000/DSP56001 经口 BI/O 线与 MC68HC99 连接

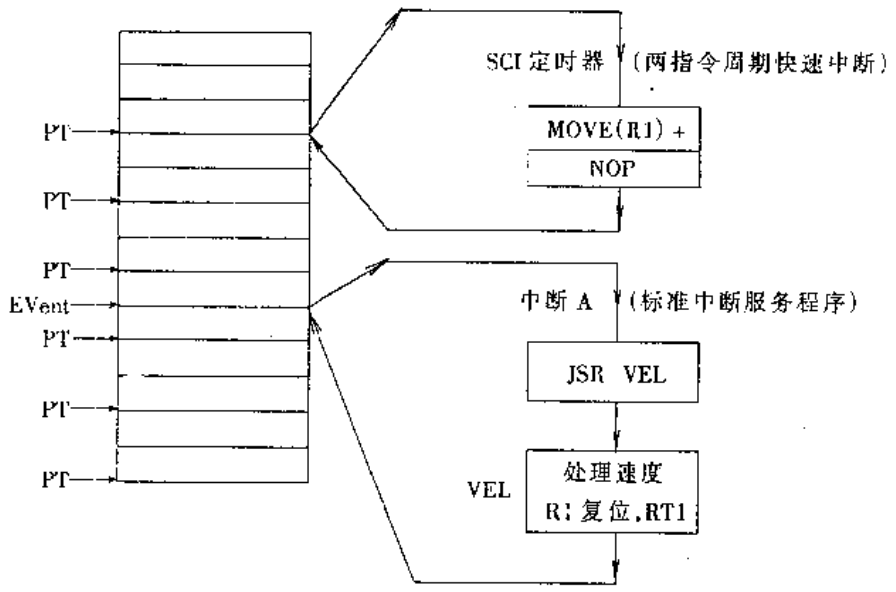
B 通道的 15 个 I/O 引脚可用于磁盘驱动器，图 4－26 表示在磁盘驱动系统中如何使用 DSP56000/DSP56001。从主计算机通过 SCSI 总线或其他格式传送数据。磁盘驱动器（例如用 MC68HC99）对这些数据进行译码、错误检查和修整，然后用标准串行接口 DSDI，ST－506 将数据传送到磁盘驱动器，由模拟电子电路将数据读/写到磁盘，DSP56000/DSP56001 编译出高级地址信息，把磁头修正到所需位置，除了调节磁头外，DSP56000/DSP56001 还可调节主轴的速度，空闲的 I/O 引脚提供磁头和主马达的故障检测信号。DSP56000/DSP56001 的工作温度范围在 －40°C～85°C，因而芯片在封闭的硬磁盘环境下可以正常工作。

5.外部中断

DSP56000/DSP56001 提供两种处部硬件中断功能，并能由程序控制在上升沿触发或下降沿触发。实现速度反馈是一种中断用法。在许多情况下，常用速度信息对马达进行控制。例如，用一个光敏感器和装在磁盘驱动器主轴上的光码凹槽检测主轴转速。用外部反馈和定时实现速度反馈，图 4－27 给出了利用 SSI 定时器和外部中断的速度反馈。



(a) t = 可编程定时器输出



(b)

图 4－27　用外部中断提供速度反馈信息

图中速度反馈信息直接输入到 IRQA。这时确定两个连续脉冲之间的时间是很必要的。如果没有使用 SCI 定时器进行数据通讯;则凹槽之间表示的时间是任意的。使用一个地址寄存器(R1)做为速度计数器。图中,一旦 SCI 定时器超时,则它快速中断。一旦这个中断出现,可读出速度定时器构成一个用软件可读的定时器。SCI 串行时钟引线(SCLK)也可作为一个有效的通用 I/O 引线。程序 LIST2 是 DSP56000/DSP5600 在此种情况下计算速度的软件例子。

```
;LIST2;
include'declare.dat'
* * * * * * * * * *
* 中断服务程序说明 *
* * * * * * * * * *
            org     p:reset             ;RESET 中断服务程序
            jmp     main                ;转主程序
            org     p:irqa              ;IRQA 中断服务程序
            jsr     vectory             ;更新
            org     p:timer             ;SCI 定时器中断服务程序
            move    (r0)+
            nop
* * * * * * * * * *
* 主程序 *
* * * * * * * * * *
            org     p:main              ;主程序
            movep   # $ C007,x:ipr      ;置 SCI、IRQA 为优先级 2

            movep   # $ 0,x:bcr         ;无等待状态
            movep   # $ 0,x:scer        ;
            movep   # $ 2000,x:scr       ;启动定时器中断
            move    # 0,r1
            move    # $ FC,mr           ;全部中断无屏蔽
            jmp     *
velocity
            process velocity            information
            move    # 0,r0
            rti
```

因为 SSI 定时器定时输出后自动复位并连续运行,所以它的快速中断所用的时间不影响整个计算。

6.SCI 定时器产生脉宽调制输出

用 SCI 定时器和模寻址,可以用通用 I/O 引线进行 PWM 输出。LIST3 给出的 DSP56000/DSP56001 汇编语言程序可产生图 4-28 所示的 PWM 输出。

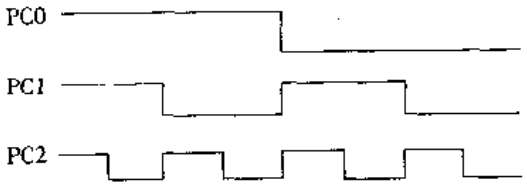


图 4－28　PWM 输出波形

LIST3

```
        include'declare.dat'
        org     x:0                     ;输出波形说明
data     dc     %111                     ;
        de      %011                    ;
        de      %101                    ;
        de      %001                     ;
        de      %110                    ;
        de      %010                    ;
        de      %100                    ;
        de      %000                    ;
        org     p:reset
        jmp      main                   ;upon reset,jmp to main

        org     p:timer                 ;after I/O pin voltages
        movep   x:(r0)+,x:ped            ;unused second instr
        nop

        org     p:main                  ;主程序
        movep   #data,r0                ;初始化数据指针
        move    #7,m0                    ;Cir buffer is modulo 8
        movep   #7,x:pcddr              ;输出口为 PC2－0
        movep   #0,x:secr
        andi     #$FC,mr                ;启动所有中断
        do      #10000,end-do
        nop
end-do
        jmp      here
```

程序对三条通用 I/O 引线 PC2－PC0 脉宽调制它使用一个数量表格控制每个 I/O 引线电平，用此代替对每条引线工作状态周期的编程。使用 SCI 定时器产生周期中断，中断优先权寄存器用以启动 SCI 定时器。定时器定时输出时，执行 SCI 定时器引起的快速中断。此中断服

务程序传送一个新的数据值到口 C 的数据寄存器，形成一个相应的 PWM 输出。通过执行一个无穷空操作循环，可使 PWM 输出同步。象所有中断一样，发出中断请求到运行快速中断的第一条指令之间的等待时间取决于正在读取、译码及运行的是何指令。连续执行 NOP 是为了保证信号输出稳定，如果没有使用 NOP，PWM 输出会在某些指令周期出现跳动。

7.用模数寻址产生三相输出

对于一个三相马达，可使用模寻址方法产生交流信号。图 4－29 给出如何使用三个地址寄存器查所需波形的数值表。在此例中，用正弦波控制电机。但是任意波形都可存储在波形表中。通过改变三个地址寄存器之间距离可以改变电机速度。程序执行到末时，模数地址使每个单独地址寄存器返回到表格起地址。最终得到一个连续的、三相周期化的输出，经调整可使输出适于不同类型电机。程序 LIET4 给出了用 DSP56000/DSP56001 实现此三相输出的软件。

```
        LIST4
            include 'declare.dat'
D-A1        equ     $FFFD           ;第一个 D-A 地址
D-A1        equ     $FFFE           ;第二个 D-A 地址
D-A2        equ     $FFFF           ;第三个 D-A 地址

            org     p:reset         ;由服务程序进行硬件复位
            jmp     main

            org     p:timer         ;SCI 定时器中断服务程序
            jsr     output          ;输出第三相电压到 D－A
            org     p:main          ;主程序单元
            movep   #$FC3C,x:ipr

            movep   #2,x:bcr        ;I/O wait states 2
            ori     #4,omr          ;启动数据 RAM
            move    #$100,r1        ;r1 指向正弦波开始
            move    #255,m1         ;r1 为模 256
            move    #$140,r2        ;r2 是 90 角输出相
            move    #255,m2         ;r2 为模 256
            move    #$180,r3        ;r3 是 180 角输出相
            move    #255,m3         ;r3 为模 256
            andi    #$FC,mr         ;启动全部中断
            jmp     *
output
            movep x:(r1)+,y:D-A1    ;输出第 1 相
            movep y:(r2)+,y:D-A2    ;输出第 2 相
            movep y:(r3)+,y:D-A3    ;输出第 3 相
```

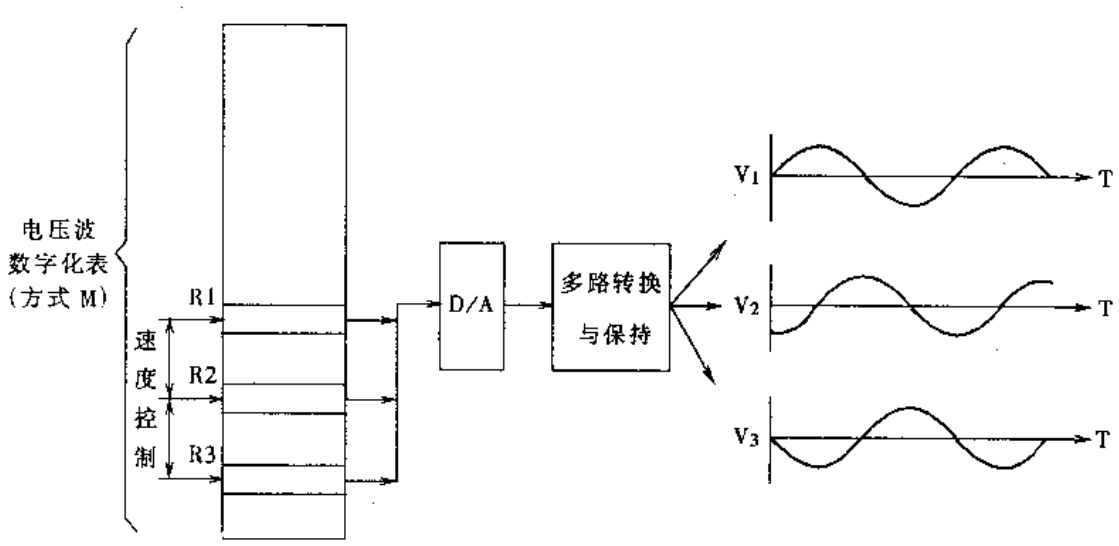


图 4－29　用模数寻址产生三相输出

第五章　TMS320C30 与'C40 处理器

5.1　TMS320C30 处理器

一、TMS320C30 硬件概述

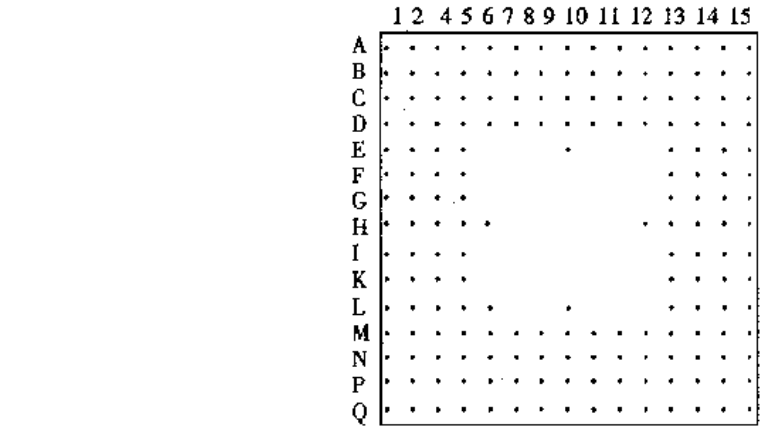
TMS320C30 是 TI 公司 1987 年开发的支持 32 位浮点数的数字信号处理器。TMS320C30 采用 1μm CMOS 技术，集成度为 69 万个晶体管，是第三代数字信号处理器。

TMS320C30 的主要特性如下：

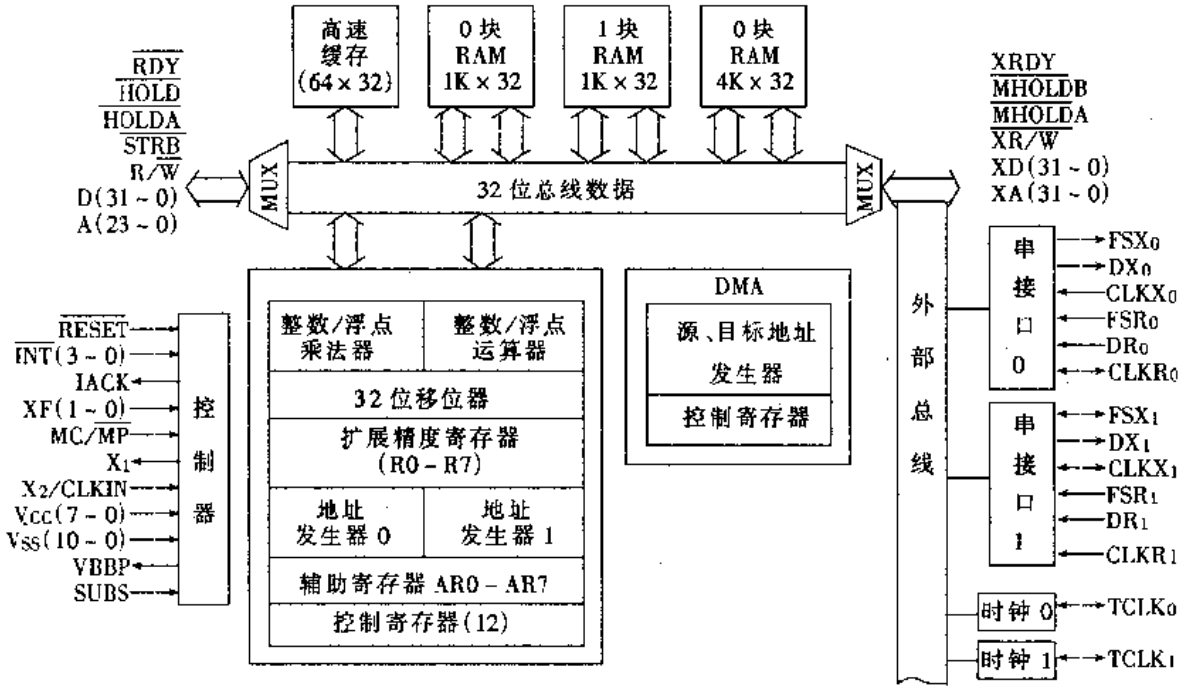
- 指令执行周期为 60ns(16.7MIPS)。
- 单周期实现浮点数乘、加运算(33.3MFLOPS)。
- 数据总线 32 位。
- 地址总线 24 位。
- 存储空间 16M 字(每字 32 位)。
- 64 位高速指令存取存储器。
- 片内 1K 字×2 的双端口 RAM。
- 片内 4K 字的双端口 ROM。
- 32/40 位整数/浮点数乘法器 ALU。
- 32 位桶形移位器。
- 8 个扩展精度寄存器(40 位)。
- 8 个辅助寄存器。
- 具有主总线和扩展总线两个外部总线。
- 联锁多处理器接口。
- 片内 DMA 控制器。
- 两组串行口。
- 两组定时器。
- 并行处理命令。
- 无辅助操作的多命令重复。
- 无辅助操作的延时锁存。
- 多处理器联锁命令。

1. TMS320C30 引脚与结构框图

TMS320C30 采用陶瓷 PGA 式封装，有 181 个引脚，如图 5－1 所示。各引脚的输入输出信号说明如表 5－1 所示。



(a) 引脚排列



(b) 结构框图

图 5-1 TMS320C30 引脚图及结构框图

表 5－1　TMS320C30 信号功能说明

信号	引脚数	I/O/Z	功　　能
D31 ~ D0	32	I/O/Z	32 位主数据总线
A23 ~ A0	24	O/Z	24 位主地址总线
R/$\overline{W}$	1	O/Z	主总线读/写信号。该引脚在并行接口读出时为"H"，写入时为"L"。
$\overline{STRB}$	1	O/Z	主总线外部存取选通。
$\overline{RDY}$	1	I	准备就绪。RDY 为"H"时，与主总线接口的地址总线和数据总线有效。
$\overline{HOLD}$	1	I	主总线接口保持信号。$\overline{HOLD}$为"L"时，在结束当前处理后，A23 ~ 0，D31 ~ 0，STRB 及 R/$\overline{W}$信号呈高阻抗状态；直到$\overline{HOLD}$信号变为"H"前，主总线接口上的所有处理呈保持状态。
$\overline{HOLDA}$	1	O	主总线接口的保持应答信号。在$\overline{HOLD}$信号变"L"时产生应答信号。该信号使 A23 ~ A0，D31 ~ D0，STRB 及 R/W 信号呈高阻抗状态，表示总线上所有数据的收送均被保持。$\overline{HOLDA}$信号当$\overline{HOLD}$变为"H"时，其应答变为"H"。
XD31 ~ XD0	32	I/O/Z	与扩展总线接口的 32 位数据总线。
XA12 ~ XA0	13	O/Z	与扩展总线接口的 13 位地址总线。
XR/$\overline{W}$	1	O/Z	扩展总线接口的读写信号。该引脚读时为"H"，写时为"L"。
$\overline{MSTRB}$	1	O/Z	扩展总线接口的外部存储器存取选通信号。
$\overline{IOSTRB}$	1	O/Z	扩展总线接口外部 I/O 存取选通信号。
XRDY	1	I	准备就绪信号。在 XRDY 为"H"时，扩展总线的地址总线和数据总线有效。
RESET	1	I	复位。当该引脚为"L"时，设备处于复位状态；当变为"H"时，开始进行由复位矢量指定位置的处理。
$\overline{INT3}$ ~ $\overline{INT0}$	4	I	外部中断。
$\overline{IACK}$	1	O	中断应答信号。IACK 指令执行时，IACK 为"L"。该信号用于表示中断子程序开始或结束。
MC/$\overline{MP}$	1	I	微处理器/微计算机方式引脚。
XF1 ~ XF0	2	I/O	外部标志引脚。
CLKX0/ CLKX1	2	I/O	串行发送时钟。该引脚向串行口发送部提供移位时钟。
DX0/DX1	2	O/Z	输出发信数据。串行口从该引脚送出串行数据。
FSX0/ FSX1	2	I/O	发信帧同步脉冲。由 FSX 脉冲开始向 DX 引脚的数据发信。
CLKR0/ CLKR1	2	I/O	串行口收信时钟。该引脚向串行口收信部提供移位时钟。
DR0/DR1	2	I	输入收信数据。串行口由该引脚收信串行数

			据。
FSR0/ FSR1	2	I	收信帧同步脉冲。
TCLK0/ TCLK1	2	I/O	定时时钟。
VBBP	1	I/O	
X1	1	O	内部振荡器输出到水晶振子。不使用晶振时，该引脚空置。
X2/CLKIN	1	I	晶振或时钟输入到内部振荡器。
H1	1	O	外部 H1 时钟。
H3	1	O	外部 H3 时钟，周期为 CLKIN 的 2 倍。

2.存储器分配

TMS320C30 全部存储器空间为 16M 字。它包含有程序、数据、I/O、存储器分配寄存器、中断矢量等区域，但是程序、数据和 I/O 空间不互相分离。

TMS320C30 的存储器有两种组织形式，即微计算机方式和微处理器方式，如图 5－2 所示。

图 5－2 存储器分配

在微计算机方式中，可使用片内 4K 字的 ROM，从 0H 地址开始进行内存分配；微处理器方式不能使用片内 ROM。16M 字的内存空间可分为主总线区、扩展总线区、I/O 区和片内 RAM 区。

3、CPU

TMS320C30 的 CPU 采用通用寄存器方式的结构体系，如图 5－3 所示的那样，CPU 由乘法器、ALU 和 28 个寄存器组成的多端口的寄存器堆组成。

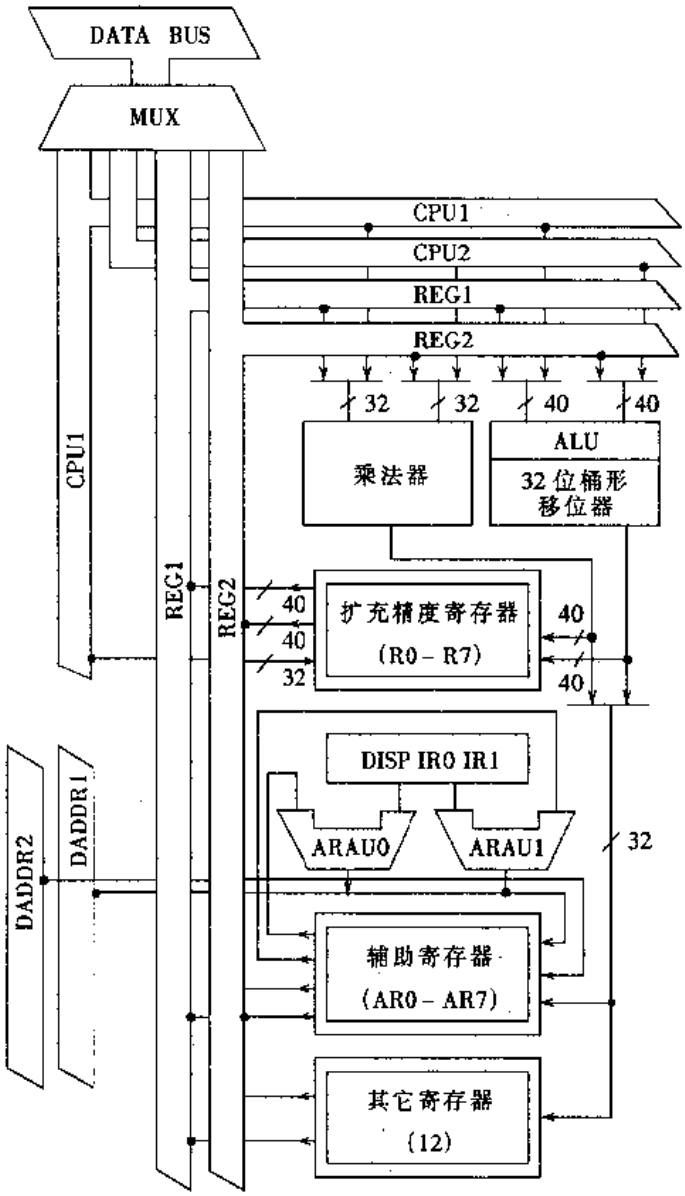


图 5－3 TMS320C30 CPU 结构

(1)乘法器

乘法器在单指令周期内完成整数 24×24 位→32 位、浮点数 32×32 位→40 位精度的乘法运算，同时，由于乘法器和 ALU 可进行并行处理，因而在单指令周期内可完成乘、加运算。

(2)ALU

ALU 可在单指令周期内完成 32 位的整数运算、32 位逻辑运算和 40 位浮点运算。同时，在其运算周期内，可进行整型数和浮点数之间的转换。ALU 的输出一般为 32 位整数或 40 位扩

展精度浮点数。

(3)辅助寄存器运算单元(ARAU0,ARAU1)

CPU 有两个辅助寄存器运算单元 ARAU0 和 ARAU1,可在单指令周期内生成两个不同的地址(合计 4 个地址)。依靠辅助寄存器运算单元生成的地址,可并行进行乘法器及 ALU 的运算。乘法器和 ALU 通过 CPU1/CPU2 及 REG1/REG2 四条总线,根据 ARAU0、ARAU1 生成的 4 个地址,可在单指令周期内提供四个数据。

(4)寄存器堆

CPU 内部有 28 个多端口寄存器,如表 5-2 所示。这些寄存器既担当一定的特殊用途,又可用做通用寄存器。

表 5-2 CPU 寄存器堆

符号	名称	功能
R0~R7	扩展精度寄存器	存储运算结果
AR0~AR7	辅助寄存器	地址指针
DP	数据页寄存器	直接寻址方式数据页指定
IR0,IR1	变址寄存器	间接寻址方式变址
BK	块尺寸	循环寻址时指定缓冲器大小
SP	堆栈指针	堆栈首地址
ST	状态寄存器	CPU 状态
IE	允许中断	允许/禁止 CPU 和 DMA 的中断
IF	中断标志	中断请求标志
IOF	I/O 标志	XF 端口功能指定和输入输出
RS	重复始地址	
RE	重复终地址	重复指令时使用
RC	重复计数器	

4.片内 ROM、RAM 和高速指令缓冲器

TMS320C30 片内有 ROM、RAM 和高速指令缓冲器。如图 5-4 所示,RAM0 和 RAM1 区容量各为 1K×32 位,ROM 容量为 4K×32 位,每个 RAM 和 ROM 区在单周期内可进行两次存取。同时,由于各区均具有相互独立的程序总线、数据总线和 DMA 总线,因而可并行进行读出程序、两数据存取及 DMA 存取操作。在一个单周期内,CPU 可以对一个 RAM 区进行两次数据存取和执行依次读取外部程序。同时,并行地有 DMA 对另一区 RAM 放入数据,内存控制器执行对内存资源和总线的管理。

TMS320C30 有了一个 64×32 位的高速指令存储区,从而以最小的系统代价达到系统的良好性能。DMA、外部存储器读取或系统其它部件都可用外部总线。

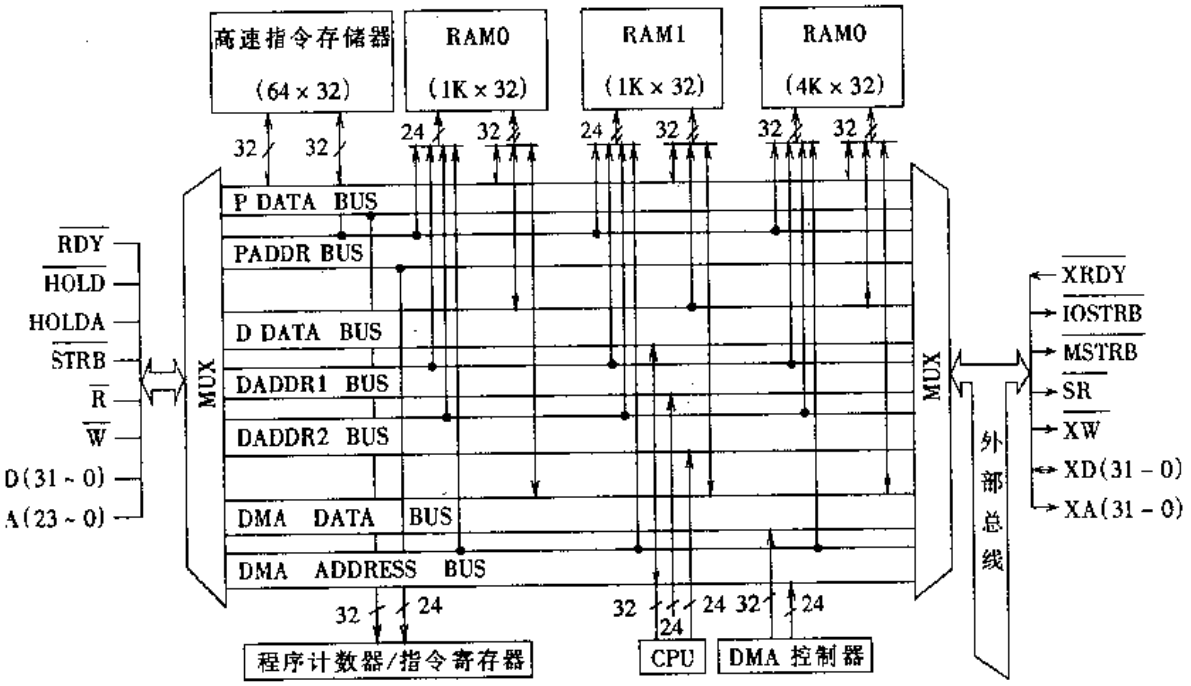


图 5-4 存储器配置

5. DMA 控制器

TMS320C30 片内 DMA 控制器如图 5-5。

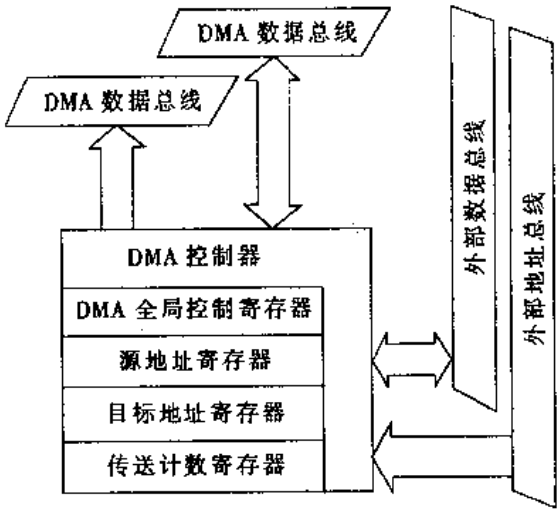


图 5-5 DMA 控制器

DMA 控制器可以不打断 CPU 的工作而对内存任意地址进行读出和写入，故当外部存储器和外设速度较慢时，采用 TMS320C30 可以保持 CPU 的处理能力。

DMA 控制器有它自己的地址生成器、源存储器、目标存储器和变换计数器，还有专用的 DMA 地址总线、数据总线，这样就避免了 DMA 控制器与 CPU 间的相互干扰，依据 DMA 操作可以是从内存输出或向内存输入一个数据块或者一个数据。

与 CPU 相似，DMA 控制器亦可进行中断，这样，DMA 可在相应中断请求下传输数据，故通常由 CPU 执行的输入/输出变换亦可由 DMA 执行。另外，当 DMA 读入或送写操作数据的同时，CPU 可继续进行数据处理。

6、I/O 接口

TMS320C30 有两组定时器/计数器和两组串行口。如图 5－6 所示。

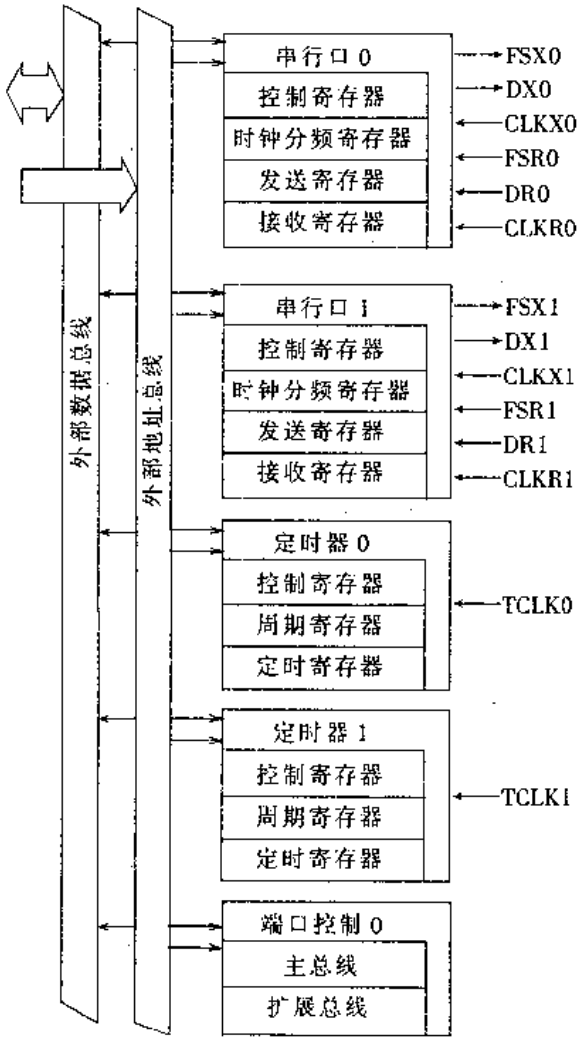


图 5－6　I/O 模块

定时器是一个具有内部或外部时钟源的 32 位通用定时器/事件计数器。该模块可完成 DMA 控制器的启动、定时输出及外部事件测量。另外，定时器的输入、输出端也可以用做通用 I/O。

串行口不仅可直接传送编码数据，而且传输每帧长为 8 位、16 位、24 位和 32 位的数据。传输时钟可设定为内部时钟或外部时钟。串行口的最大传送速度为 8Mbps。另外，串行端口的各输入、输出口可做为通用 I/O 使用。

TMS320C30 还有两个外部接口:并行接口和输入/输出接口。并行接口由一条 32 位的数据总线,一条 24 位的地址总线及一组控制信号组成。输入/输出接口由一条 32 位数据总线,一条 13 位地址总线和一组控制信号组成。这两个端口为等待状态的产生和软件控制等待状态的利用提供准备好信号。充足的内部外部总线使 TMS320C30 能达到其最高性能指标和以最高速度执行程序。四条内部总线执行两次彼此独立的数据读取,程序读取和一次 DMA 读取,并且都是在单周期内同时进行的。两条外部总线可以同时进行程序读取,数据读写和 DMA 读取。TMS320C30 支持四种外部中断,若干种内部中断及一个非屏蔽外部复位信号。两个外部输入/输出标志可构成由软件控制的输入输出引线 XF0 和 XF1。此外,连锁指令会利用这两个引脚支持多处理器间的通信。

二、TMS320C30 数据格式

TMS320C30 支持整数、无符号整数和浮点数三种类型的数据。

1.整数

TMS320C30 支持 16 位的短整数和 32 位单精度整数。整数在扩展精度寄存器中仅使用 31 ~ 0 位,39 ~ 32 位不能使用。

短整数表示为 16 位的 2 的补码。对这种格式进行符号扩展则成为 32 位的单精度整数;如图 5 - 7 所示。图中的 S 为符号位。

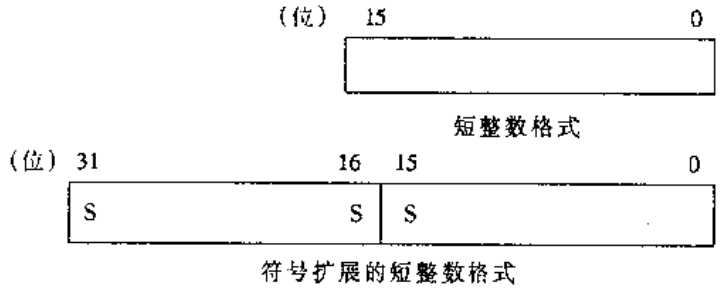


图 5 - 7 短整数格式及其符号扩展

单精度整数格式为 32 位 2 的补码。

2.无符号整数

TMS320C30 中,支持 16 位的无符号短整数和 32 位单精度无符号整数。在扩展精度寄存器中,无符号整数仅使用 31 ~ 0 位。单精度无符号整数用 32 位表示,如图 5 - 8 所示。

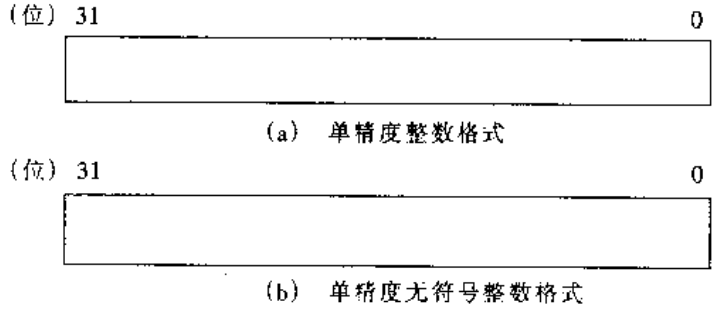


图 5 - 8 单精度整数

无符号短整数用 16 位表示。这种格式中,第 31 ~ 16 位填满 0,从而成为 32 位的单精度无符号整数格式,如图 5 - 9 所示。图中的 X 位是第 15 位,可取 0 或 1。

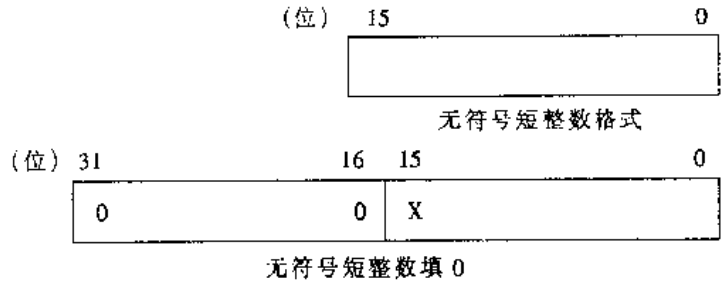


图 5-9 无符号短整数格式

3.浮点数

浮点数格式由阶码(e)、符号位(s)和小数(f)三部分组成。阶码用 2 的补码表示,s 和 f 合在一起称为尾数。尾数也用 2 的补码表示。s 和 f 之间省略 1 位,该位在 s = 0 时为 1,在 s = 1 时为 0。用这种方法表示的浮点数,如图 5-10 所示。

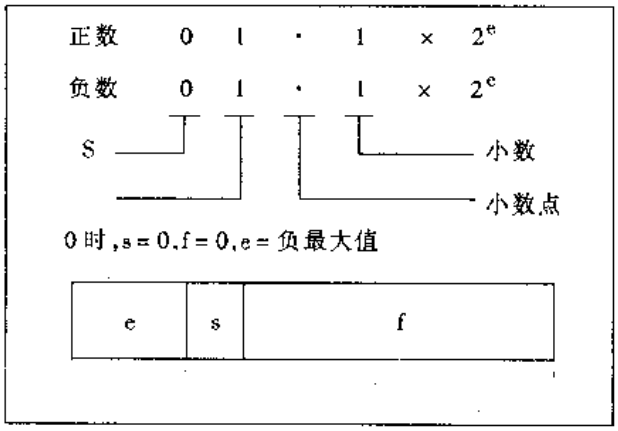


图 5-10 一般浮点数的格式

TMS320C30 支持下述三种浮点格式:

(1)短浮点数据(图 5-11)。阶码 4 位,符号部 1 位,小数部分 11 位。

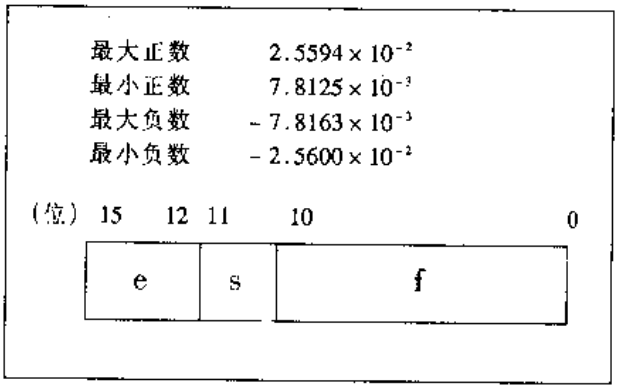


图 5-11 短浮点数格式

(2)单精度浮点数(图 5-12)。阶码 8 位,符号 1 位,小数部分 23 位。

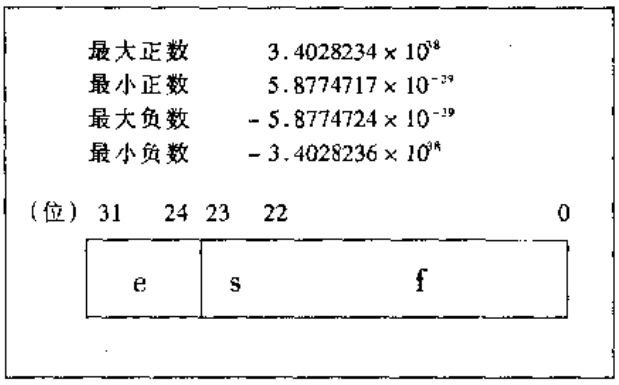


图 5－12　单精度浮点数格式

(3)扩展精度浮点数(图 5－13)。阶码 8 位,符号 1 位,小数部分 31 位。

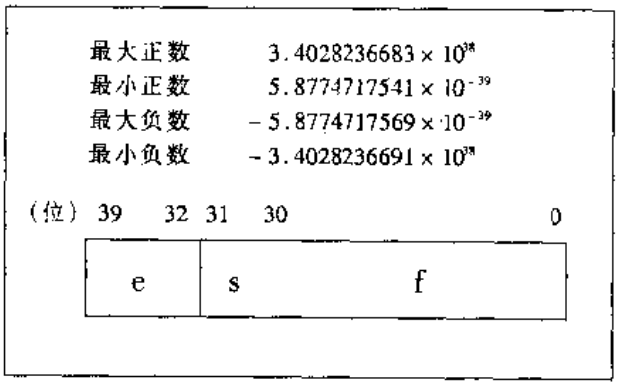
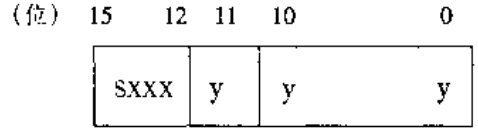


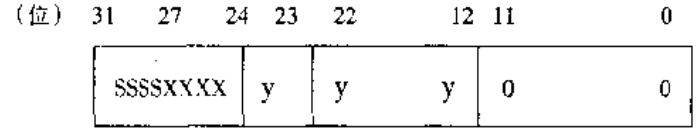
图 5－13　扩展精度浮点数格式

上述三种格式的浮点数之间可互相转换,转换时不用辅助操作而自动进行。如图 5－14 所示。

①短浮点数转换为单精度浮数：



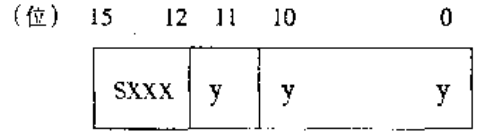
短浮点数格式



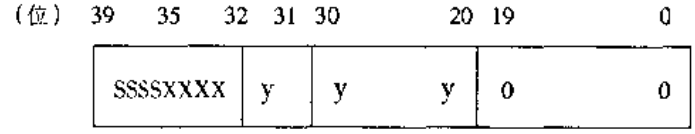
单精度浮点数

这种格式中，阶码进行了符号扩展，小数部分填满 0。

②短浮点数转换为扩展精度浮点数



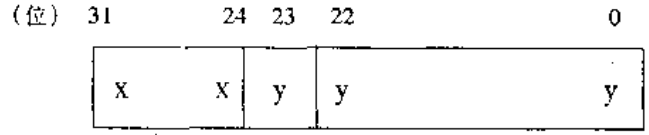
短浮点数



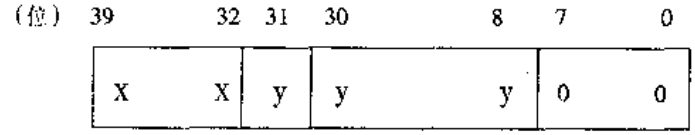
扩展精度浮点数

阶码进行了符号扩展，小数部分填满 0。

③单精度浮点数转换为扩展精度浮点数

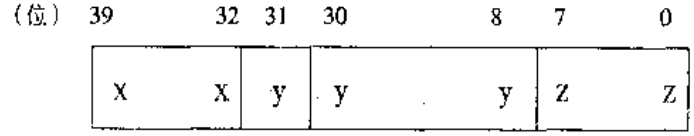


单精度浮点数

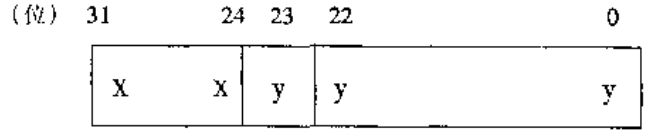


扩展精度浮点数

④由扩展精度浮点数变换为单精度浮点数



扩展精度浮点数



单精度浮点数

小数部分舍去

图 5－14　格式之间转换

三、TMS320C30 指令系统

TMS320C30 有 114 个汇编语言指令。全部指令的字长为一个逻辑字，大部分指令在一个机器周期内完成。TMS320C30 的指令系统可作如下分类：

- 装入和存储指令；
- 两个操作数的算术/逻辑运算指令；
- 三个操作数的算术/逻辑运算指令；
- 并行操作指令；
- 算术/逻辑内存操作指令；
- 程序控制指令；
- 联锁操作指令。

1.装入和存储指令

TMS320C30 中的置数和存储指令如表 5－3 所示。这类指令有 12 条。与其它微处理器的差别是，TMS320C30 有条件置数和条件存储指令。

表 5－3　装入和存储指令

指令助记符	说　　明
LDE	装入点数阶码部分
LDF	装入浮点数
LDF cond	条件装入浮点运算数
LDI	装入整数
LDI cond	条件装入整数
LDM	装入浮点数尾数
POP	从堆栈弹出整数
POPF	从堆栈弹出浮点数
PUSH	整数压入堆栈
PUSHF	浮点数压入堆栈
STF	存储浮点数
STI	存储整数

应用置数、存储、重复指令的程序例如下：

```
        LDI     nn,RC           ;将数据数－1(nn)并置入重复计数器
        LDI     @ADDR,AR0       ;数据首地址→AR0
        LDF     *AR0++,R0       ;最大值初值
        LDF     *AR0,R1         ;最小值初值
                                ;指令执行后 AR0 表示下一个数据
        RPTB    MINMAX          ;块重复到 MINMAX
        CMPF    *AR0,R0         ;当前最大值与 AR0 的数据比较
        LDFLT   *AR0,R0         ;比较结果为小于时更新最大值
        CMPF    *AR0++,R1       ;当前最小值(R1)和 AR0 表示值比较
MINMAX: LDFGT   *-AR0,R1        ;比较结果为大于时更新最小值，AR0
                                ;为－AR0
```

除此之外，在后述的并行操作，联锁操作中还有置数/存储命令。

2.两操作数指令

表 5－4 列出了两操作数指令，用于两操作数的算术和逻辑运算。这些指令大体和一般的 CPU 的指令相同。

表 5－4　两操作数指令

指令助记符	说　　明
ABSF	取浮点数绝对值
ABSI	取整数绝对值
ADDC	整数进位加
ADDF	浮点数加
ADDI	整数加
AND	按位取逻辑“与”
ANDN	按位取补码的逻辑“与”
ASH	算术移位
CMPF	浮点数比较
FIX	浮点数转换为整数
FLOAD	整数转换为浮点数
LSH	逻辑移位
MPYE	浮点数乘
MPYI	整数乘
NEGB	带借位的整数符号取反
NEGF	浮点数符号取反
NEGI	整数符号取反
NORM	浮点数规格化
NOT	按位取逻辑补
OR	按位取逻辑或
RND	浮点数截尾
ROL	左循环移位
ROLC	带进位左循环
ROR	右循环移位
RORC	带进位右循环移位
SUBB	带借位整数减
SUBC	条件整数减
SUBF	浮点数减
SUBI	整数减
SUBRB	带借位整数的反减
SUBRF	浮点数反减
SUBRI	整数反减
TSTB	位字段检查
XOR	按位取 XOR

3.三操作数算术/逻辑运算指令

三操作数算术/逻辑运算指令如表 5－5。使用三操作数指令时，TMS320C30 在一个周期内，可将两个操作数从存储器或寄存器中读出，将结果存储在寄存器。

表 5－5　三操作数指令

指令助记符	说　　明
ADDC3	带进位的整数加
ADDF3	浮点数加
ADDI3	整数加
AND3	按位取逻辑"与"
ANDN3	和补码按位取"与"
ASH3	算术移位
CMPF3	浮点数比较
CMPI3	比较整数
LSH3	逻辑移位
MPYF3	浮点数乘
MPYI3	整数相乘
OR3	按位取逻辑或
SUBB3	带借位的整数减
SUBF3	浮点数相减
SUBI3	整数相减
TSTB3	位字段检查
XOR3	按位取"异或"

4.程序控制指令

程序指令由所有对程序流程发生影响的操作组成，它大致可分为两大类：重复模式和转移，如表 5－6 所示。

表 5－6　程序控制指令

命　令	说　　明
Bcond	标准条件转移
BcondD	延迟条件转移
BR	标准无条件转移
BRD	延迟无条件转移
CALL	调用子程序
CALLcond	条件调用
DBcond	条件减量转移
DBcondD	延迟的条件减量转移
IDLE	空闲状态到中断
NOP	无操作
RETIcond	来自中断的条件返回
RETScond	来自子程序的条件返回
RPTB	循环命令块
RPTS	循环单命令
SWI	软中断
TRAPcond	条件陷阱

* cond 表示可使用表 5－7 的条件。

在很多算法中，都存在着一个指令码的内核(inter kernel of code)，程序执行的大部分时间

花在这上面。TMS320C30 的重复模式允许进行零值以上的循环操作，当采用这个重复模式，即使用 TMS320C30 的块重复指令时，如指令 RPTS(重复单个指令)和 RPTB(重复一块指令)等，TMS320C30 的块的大小不限，执行过程中可中断，而且这段码的内核可以在尽可能短的时间内被执行。

TMS320C30 的转移指令包括两大类：标准转移和延迟转移。标准转移的执行时间为四个周期，包括调用和返回操作；延迟转移的执行时间为一个周期，标准转移和延迟转移都可以是多条件的。

标准转移和延迟转移的程序例如下：

```
        LDI     LABEL       ;① 标准转移
        LDI     xx,xx       ;②
        ADDI    xx,xx       ;③
        STI     xx,xx       ;④
        ......
LABEL;  ANDI    xx,xx       ;⑤
        ......
        BRD     LABELD      ;⑥延迟转移
        MPYF    xx,xx       ;⑦
        ADDF    xx,xx       ;⑧
        NOP                 ;⑨
        ......
LABELD; ORI     xx,xx       ;⑩
        ......
```

在某时刻一旦执行标准转移①，则在流水线处理中舍去某个先读取的指令或译码器中的指令②③④，而执行转移目的地址的指令；但是延迟转移⑥，则不舍去流水线中的指令而是在这些指令后读入转移目的地的指令。也就是说，实际发生转移的地点不是置转移指令的地方，而是在其后三条指令⑦⑧⑨执行后的地方。这样，使用延迟转移指令，流水线不会紊乱，而且不可能产生读入指令无效。指延迟转移实际上用 4 个时钟，由于在延迟转移指令后需执行三条指令，若不满三条指令时，则如⑨那样写上 NOP。

表 5-7　条件码和标志

无条件比较		
U	无条件	无视
LO	较小	C
LS	以下	C 或 Z
HI	较大	~C 及 ~Z
HS	上	Z
EQ	相等	Z
NE	不等	~Z
带符号比较		
LT	较小	N
LE	以下	N 或 E
GT	较大	~N 及 ~Z
GE	以上	~N
EQ	相等	Z

NE	不等	~Z
	与0比较	
Z	零	Z
NZ	非零	~Z
P	正	~N及~Z
N	负	N
NN	非负	~N
	与条件标志比较	
NN	非负	~N
N	负	N
NZ	非零	~Z
Z	零	Z
NV	无溢出	~V
V	溢出	V
NUF	无上溢出	~UF
UF	上溢出	UF
NC	无进位	~C
C	进位	C
NLV	无锁存溢出	~LU
LV	锁存器溢出	LU
NUF	无锁存浮点溢出	~LUF
UF	锁存器浮点溢出	LUF
ZUF	零或浮点数溢出	Z或UF

TMS320C30的流水线结构的五个主要单元及其功能如下：

(1)取指单元(F)：从存储器中取指令字后，刷新程序计数器。

(2)译码单元(D)：指令字译码之后，执行生成的地址，辅助寄存器和堆栈指针的修改，均由该单元控制。

(3)读单元(R)：从存储器读出操作数。

(4)执行单元(E)：根据需要从寄存器文件中读出操作数，执行必要的运算。如果需要，也可将结果写入寄存器文件，或将前一次的运算结果写入存储器。

(5)DMA通道：读/写存储器。

基本指令有取指、译码、读出、实行四阶段，图5-15表示四级流水线的构造。

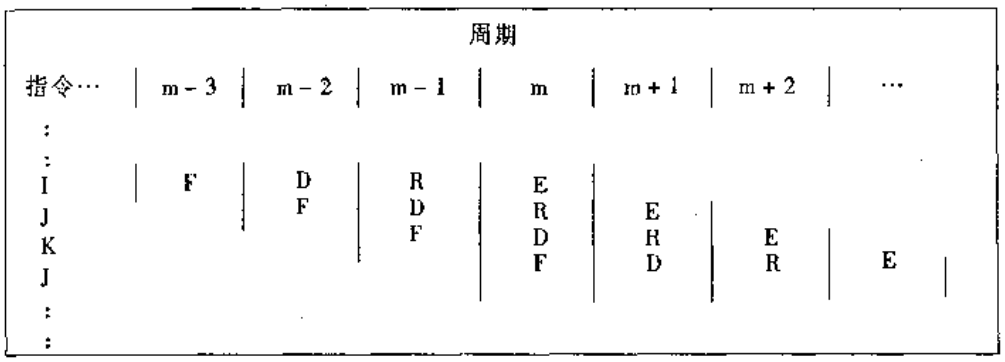
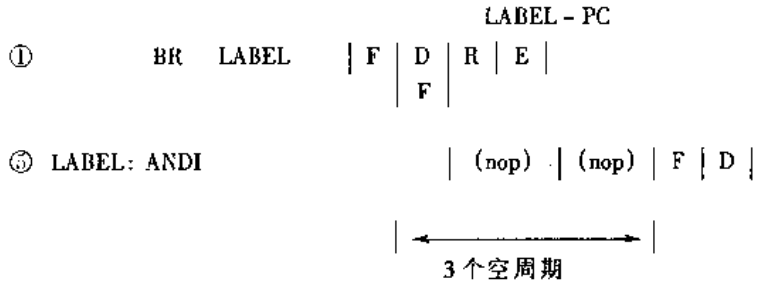
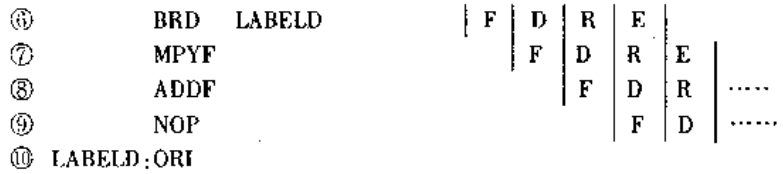


图5-15 TMS320C30流水线构造

一般分支和延迟分支的流水线处理如图 5－16 所示。



(a) 普通分支流水线



(b) 延迟分支和延迟分支流水线处理

图 5－16 一般分支和延迟分支流水线处理

5.联锁操作命令

联锁操作指令如表 5－8 所示。联锁操作指令支持多处理器之间的通信,依靠这些指令和外部信号(XF0,1),可实现机构高速同步。图 5－17 为两个 TMS320C30 之间最简单的取同步的方法。

表 5－8 联锁操作命令

指令助记符	说　　明
LDFI	装入浮点数,互锁
LDII	装入整数
SIGI	信号
STFI	存储浮点数
STII	存储整数

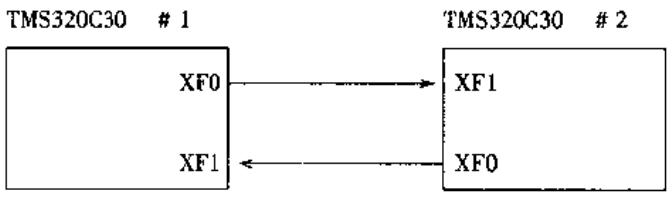


图 5－17 TMS320C30 间简单同步

6.并行操作命令

并行操作命令如表 5－9 所示。在汇编语言中，符号“||”表示同时执行两个指令。并行操作指令主要分三类：

(1)寄存器的并行装入；

(2)并行算术运算；

(3)和存储指令同时使用的算术/逻辑运算指令。

使用并行命令时，在一个周期(60ns)内可完成两个运算。这时，可发挥出 TMS320C30 的最高速度(33MFLOPS)。

表 5－9　并行操作命令

类型	助记符	说　明	操　作
	ABSF‖STF	浮点数绝对值	|src2|→dst1 ‖src3→dst2
	ABSI‖STI	整数绝对值	|src2|→dst1 ‖src3→dst2
	ADDF3‖STF	浮点数加算	src1 + src2→dst1 ‖src3→dst2
并	ADDI3‖STI	整数加	src1 + src2→dst1 ‖src3→dst2
	AND3‖STI	按位取反	src1 AND src2→dst1 ‖src3→dst2
	ASH3‖ST1	算术移位	If count > 0： src2 < < count→dst1 ‖src3→dst2
行			Else src2 > > |count|→dst1 ‖stc3→dst2
	FIX‖STI	浮点数变为整数	Fix(src2) - > dst1 ‖src3 - > dst2
	FLOAT‖STF	整数变浮点数	Float(src2)→dst1 ‖src3→dst2
操	LDF‖STF	置入浮点数	src2→dst1 ‖src3 - > dst2
	LDI‖STI	置入整数	src2→dst1 ‖src3→dst2
	LSH3‖STI	逻辑移位	If count > 0 src2 < < count→dst1 ‖src3→dst2 Else； src2 > > |count|→dst1
作			‖src3→dst2
	MPYF3‖STF	浮点数乘	src1 X src2 →dst1 ‖src3→dst2
	MPYI3‖STI	整数乘	src1 X src2→dst1 ‖src3→dst2
	NEGF‖STF	浮点数符号取反	0 - src2→dst1 ‖src3→dst2

	NEGI ‖STI	整数符号取反	0 - src2→dst3 ‖src3→dst2
	NOT3‖STI	取补码	src→dst1 ‖src3→dst2
	OR3‖STI	按位取或	src1 OR src2→dst1 ‖src3→dst2
	STF‖STF	存储浮点数	src1→dst1 ‖src3→dst2
	STI‖STI	存储整数	src1→dst1 ‖src3→dst2
	SUBF3‖STF	浮点数减	src1 - src2→dst1 ‖src3→dst2
	SUBI3‖STI	整数减	src1 - src2→dst1 ‖src3→dst2
	XOR3‖STI	按位取异或	src1 XOR src2→dst1 ‖src3→dst2
并行置数	LDF‖LDF	置入浮点数	src2→dst1 ‖src4→dst2
	LDI‖LDI	置入整数	src2→dst1 ‖src4→dst2
并行乘、加/减	MPYF3‖ADDF3	浮点数乘、加	op1 × op2→op3 ‖op4 + op5→op6
	MPYF3‖SUBF3	浮点数乘、减	op1 × op2→op3 ‖op4 - op5→op6
	MPYI3‖ADDI3	整数乘、加	op1 × op2→op3 ‖op4 + op5→op6
	MPYI3‖SUBI3	整数乘、减	op1 × op2→op3 ‖op4 - op5→op6

四、TMS320C30 寻址方式及开发支援环境

1.寻址方式

TMS320C30 支持六种强有力的寻址方式:寄存器方式,直接方式,间接方式,短立即数方式,长立即数方式和 PC—关系寻址方式,用户可以从内存或寄存器进行数据存取。在一个周期内,可产生两次独立自主的寻址且进行两次独立自主的数据读取。

(1)寄存器寻址

寄存器寻址方式中,操作数即成为寄存器的内容。

(2)直接寻址方式

直接寻址方式中,数据页寄存器(DP)中的低 8 位和指令字内的低 16 位连接,形成数据的 24 位地址。如图 5 - 18 所示。这样,TMS320C30 的存储空间被分为 256 页(每页 64K 字),各页内的存取时不必变更 DP 的内容。

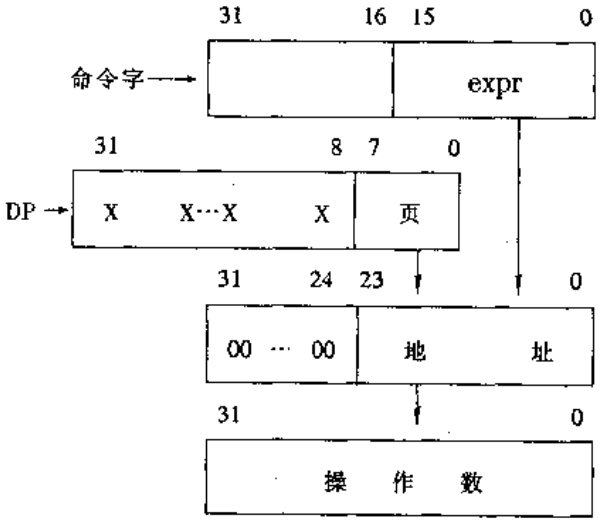


图 5－18 直接寻址方式

(3)间接寻址方式

间接寻址方式中，由辅助寄存器、移位寄存器及变址寄存器的内容合成数据的地址。表 5－10 表示间接寻址方式的语句构成。

表 5－10 间接寻址方式

语 句	操 作
* ARn	ADR = ARn
* ±ARn(x)	ADR = ARn ± x
* ± ±ARn(x)	ADR = ARn ± x，ARn←ARn ± x
* ARn± ±(x)	ADR = ARn，ARn←ARn ± x
* ARn± ±(x)%	ADR = ARn，ARn←CIRC(ARn ± x)
* ARn+ +(IR0)B	ADR = ARn，ARn←B(ARn ± IR0)

表注：①ADR 是操作数的地址；②x 是命令字内的移位，或变址寄存器(IRm，m = 0，1)；③± 表示有 +，－ 时；④± ± 表示有 + +，－ － 的情形；⑤ARn 表示辅助寄存器 AR0～AR7；⑥% 表示循环寻址；⑦B 表示位反转寻址方式；⑧x 缺省时取 1。

①间接寻址方式 1

汇编语言语法：* ARn

方式如图 5－19。

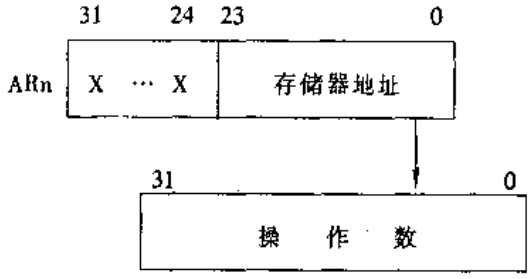


图 5－19 间接寻址方式 1

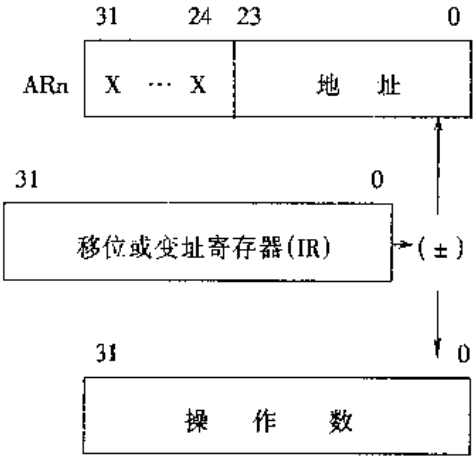
例：　LDF　　* AR0,R1

将 AR0 所示地址的浮点数装入 R1。

②间接寻址方式 2

汇编语言语法：* ± ARn(x)

方式如图 4－20。



移位：低 8 位为无符号整数(省略 1)
IR：低 24 位为无符号整数(IR0,IR1)

图 5－20　间接寻址方式 2

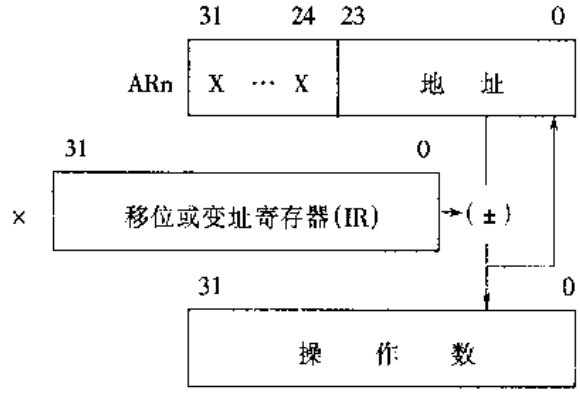
例：

ADDI　　* + AR1(2),R3

AR1 的内容加 2，结果作为操作数地址，其地址的内容与 R3 相加，结果进入 R3，AR1 的内容无变化。

③间接寻址方式 3

汇编语言语法：* ± ± ARn(x)。方式如图 5－21。



移位：低 8 位为无符号整数(省略 1)
IR：低 24 位为无符号整数(IR0,IR1)

图 5－21　间接寻址方式 3

例：

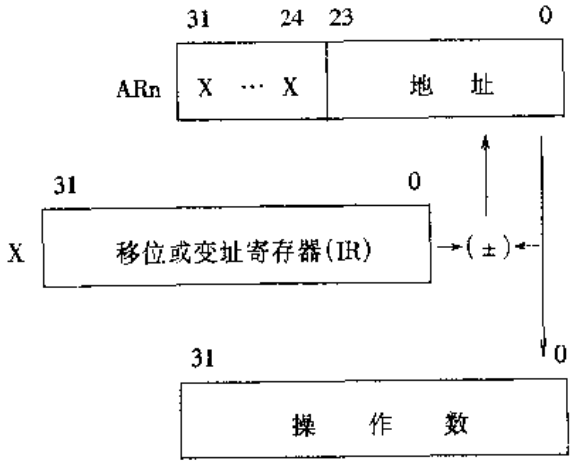
STI　R2，* - - AR3(IR0)

AR3 的内容减去变址寄存器(低 24 位)，其结果作为操作数地址。在该地址存储 R2 的内容。AR3 的内容刷新为刚生成的操作数地址。

④间接寻址方式 4

汇编语言语法：* ARn ± ±(x)

方式如图 5 - 22。



移位：低 8 位为无符号整数(省略 1)
IR：低 24 位为无符号整数(IR0,IR1)

图 5 - 22　间接寻址方式 4

例：

LDI　* AR0 + +(IR1),R2

AR0 的内容为操作数地址。该地址内容置入 R2，指令执行后 AR0 的值加上 IR1 的值。

⑤循环寻址方式

在间接寻址方式中，循环寻址和位反转寻址是 TMS320C30 特有的寻址方式。

循环寻址是在指针指向的地址超过指定范围后，则指针自动地返回首地址。

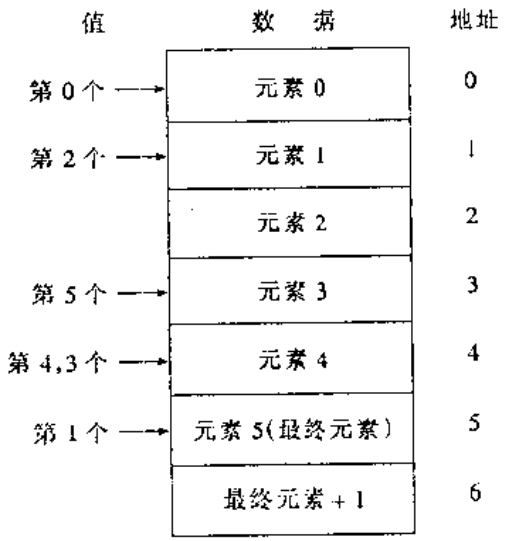
图 5 - 23 为循环寻址方式。

汇编语言语法为：* ARn ± ±(x)%

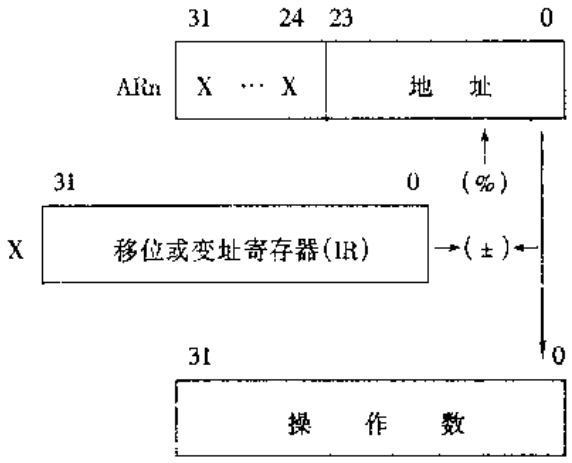
例：初始化设定 (AR0，BK 设定为 4 位)。

```
AR0 = 0000
BK = 0110          (缓冲器大小为 6)
* AR0              ;AR0 = 0
* AR0 + +(5)%      ;AR0 = 5
* AR0 + +(2)%      ;AR0 = 1
* AR0 - -(3)%      ;AR0 = 4
```

```
*AR0++(6)%     ;AR0=4
*AR0--(%)      ;AR0=3
```



循环寻址常用于相关计算和数字滤波器设计的程序中。



移位:低8位为无符号整数
IR:低24位为无符号整数(IR0,IR1)

图 5-23 循环寻址方式

⑪位反转寻址方式

位反转寻址方式是按图5-24所示的顺序(例如FFT的指针数为8时)进行存取的寻址方式。图5-24的汇编语言语法为:

```
*ARn++(IR0)B
```

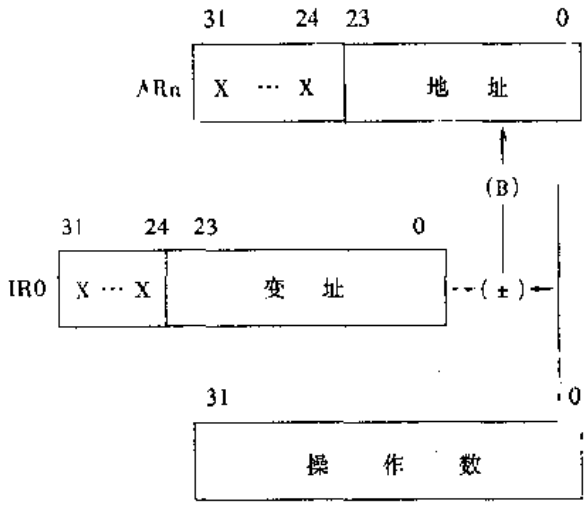


图 5-24 变址加算后位反转寻址

例如:初值

AR2 (基本地址)

0110 0000 (96)

IR0 (FFT 的点数)

0000 1000 (8)

地址		偏置	
* AR2	;AR2 = 0110 0000	0	X(0)
* AR2 + + (IR0)B	;AR2 = 0110 1000	8	X(4)
* AR2 + + (IR0)B	;AR2 = 0110 0100	4	X(2)
* AR2 + + (IR0)B	;AR2 = 0110 1100	12	X(6)
* AR2 + + (IR0)B	;AR2 = 0110 0010	2	X(1)
* AR2 + + (IR0)B	;AR2 = 0110 1010	10	X(5)
* AR2 + + (IR0)B	;AR2 = 0110 0110	6	X(3)
* AR2 + + (IR0)B	;AR2 = 0110 1110	14	X(7)

这种寻址方式在 FFT 运算程序中使用。

(4)短立即数寻址方式

短立即数寻址方式中,操作数为指令字内进入低 16 位内的立即值。依据指令中假定的数据类型,可为 2 的补码整数,无符号整数或浮点数。

(5)长立即数寻址方式

长立即数寻址方式的操作数进入指令字内低 24 位内的立即值。

(6)PC 相对地址方式

PC 相对寻址方式常用做转换,一般用命令字的 16LSB 的内容置换 PC 的值。

2.TMS320C30 的开发环境及支持工具

TMS320C30 的软件开发工具中，比较成功的有在 MS－DOS 上使用的 C 编译器，汇编语言和链接器等。所有软件的开发，全部可以在 MS－DOS 下进行。C 编译器的输出为汇编语言的源码，C 编译器的软件包内，包含有汇编语言和链接器功能。

另外，已开发有 TMS320C30 调试器、仿真器，在 IBM PC 机即可运行。

TMS320C30 软件开发流程如图 5－25 所示。

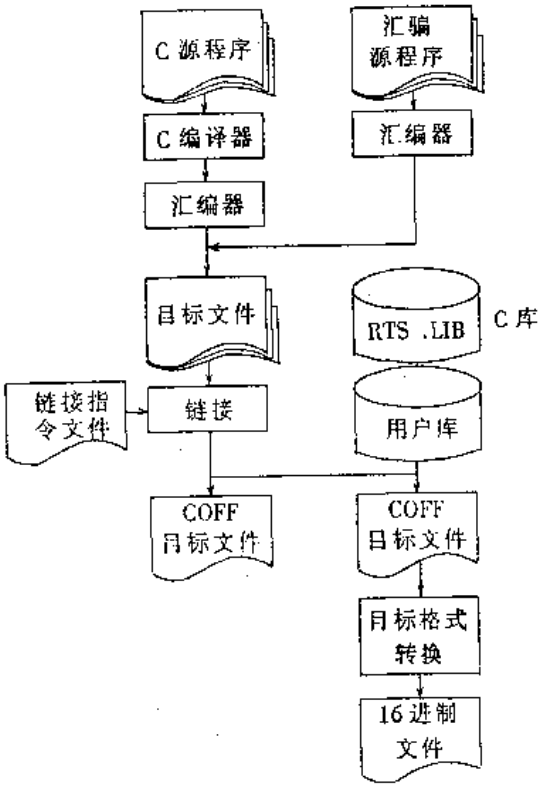


图 5－25　TMS320C30 开发流程

(1)C 编译器

TI 公司开发的 TMS320C30 C 编译器，基本符合标准 K&R，但和 K&R 主要不同点如下：

①标识符，最大 31 字符。

②追加 eunm，void，asm 关键字。

③char，short，int，long，float，double 的大小为 32 位。

④可将结构、联合做为直接函数的参数及返回值。

⑤字符串处理差异较大。

除字符串处理外，在其他扩展功能方面不存在什么问题。另外，字符串处理在数字信号处理中不是很重要的。

利用 C 编译器，用汇编语言书写的程序，可做为 C 函数来调用，也可以在 C 源程序中插入汇编语言源程序。因此，时间的极限部分可用汇编语言记述。同时，由于 C 编译器输出汇编语言源程序，在程序编辑中，可自行实现最佳化。

C 编译器的程序库中包含全部函数。同时，源程序附属在程序库，容易实现自身改善。

存储器模式分为小规模模式和大规模模式。小规模模式下，静态和动态合计必须在 64K 长字以内；大规模模式则没有这种限制。这种限制是由于 TMS320C30 在直接寻址方式下，24 位地址的上 8 位由数据页寄存器指定而产生的，在大规模模式下，变量引用时对 DP 寄存器进行再设定。

C 编译器目前使用 Ver.2 版本，Ver.3 版本可能采用 ANSI 标准。

(2)汇编器

汇编器将 TMS320C30 的汇编源程序译成浮动目标代码文件。源程序中含指令、汇编控制指令和宏指令。汇编控制指令含有源程序格式，数据数组以及预约时间的内容等。

(3)链接器

链接器将程序库和浮动目标代码文件结合在一起，生成一个可执行文件。

链接器分配给对话时间、符号等以特定的存储器地址，将几个文件组合构成对话时间。这些均用链接命令文件来进行指定，通过链接命令文件，定义 TMS320C30 可使用的存储器，分配各对话时间链接时的自由使用的存储器。

链接命令文件是一个测试文件，其典型例如下；

链接命令文件例：

```
/* SPECIFY THE SYSTEM MEMORY MAP */
MEMORY
{
    MYWORK: org = 0xc0        len = 0x000060    /*  工作 RAM     */
    STACK:  org = 0x180       len = 0x000100    /*  堆栈         */
    ERAM1:  org = 0x1500      len = 0x7feb00    /*  外部 RAM     */
    IRAM1:  org = 0x809800    len = 0x000400    /*  片内 RAM1    */
    IRAM2:  org = 0x809c00    len = 0x000400    /*  片内 RAM2    */
}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */
SECTIONS
{
    .text:      {} > IRAM1    /*  C 代码       */
    .data:      {} > IRAM2    /*  C 数据       */
    .bss:       {} > ERAM1    /*   C 的        BSS */
    .workarea:  {} > MYWORK   /*  自工作区     */
    .sysmem:    {} > ERAM1    /*               */
    .cinit:     {} > ERAM1    /*  C 初始部分   */
}
```

MEMORY()表示实装存储器，SECTION()指定将以汇编语言描述的节分配给那一个用 MEMORY()描述的存储区内。

五、TMS320C30 与外部设备接口

TMS320C30 与高速 SRAM 接口

TMS320C30 主总线的读/写时序如图 5－26 所示。无等待存取时，读数据时间为 1 个周

期、写数据需用 2 个周期，和一般 CPU 无多大差别。

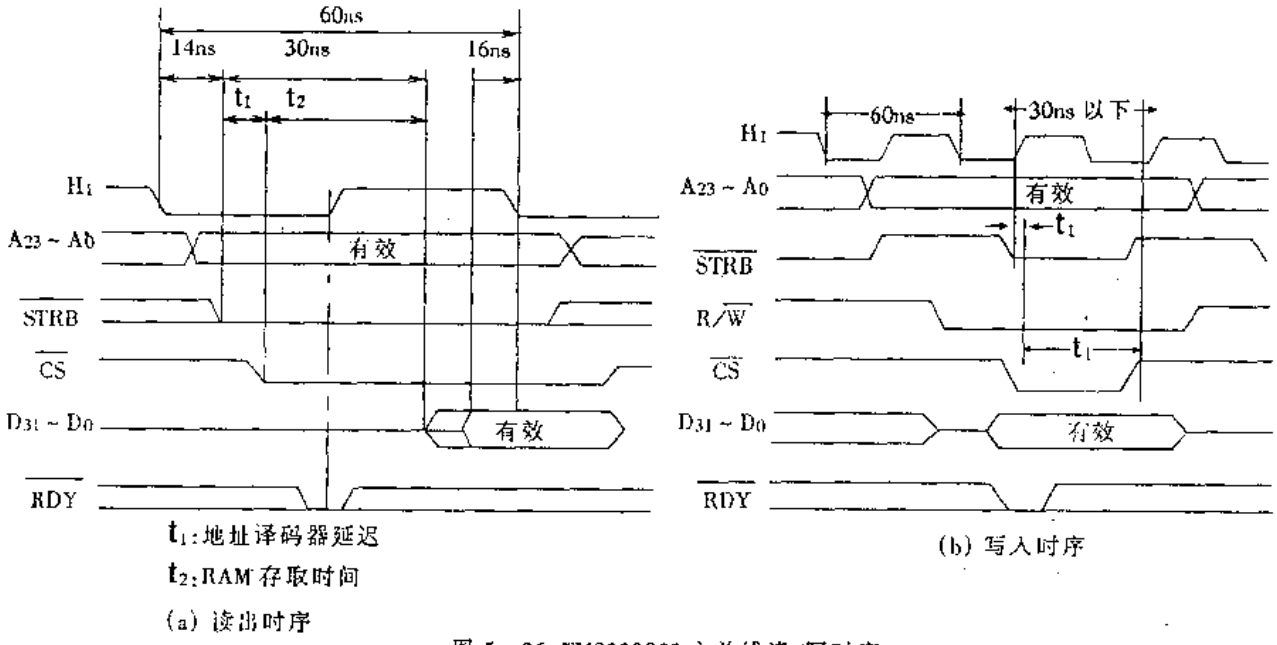


图 5 - 26 TMS320C30 主总线读/写时序

图中，t1 表示地址译码的延迟，t2 表示存取时间。

TMS320C30 主总线对主存储器的存取，采用无等待存取方式也是可能的。一般存取时间必需在 30ns 以下，当地址译码器内使用一级选通门(74AS 延迟时间大约 5ns)时，存储器需的存储时间为 25ns。

图 5 - 27 为 TMS320C30 与高速 SRAM 的接口电路。它与 8 个 64K × 4 容量的 HM6208 相连接。

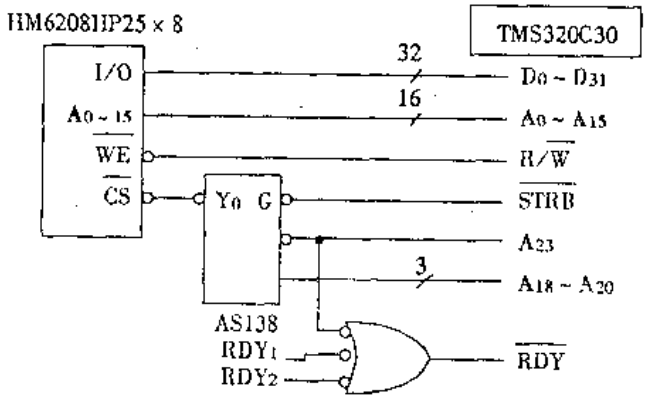
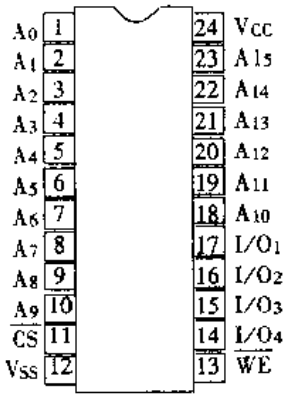


图 5 - 27 TMS320C30 与 SRAM 接口

高速 SRAM HM6208 - 25 的引脚及存取时序图如图 5 - 28 和图 5 - 29 所示。

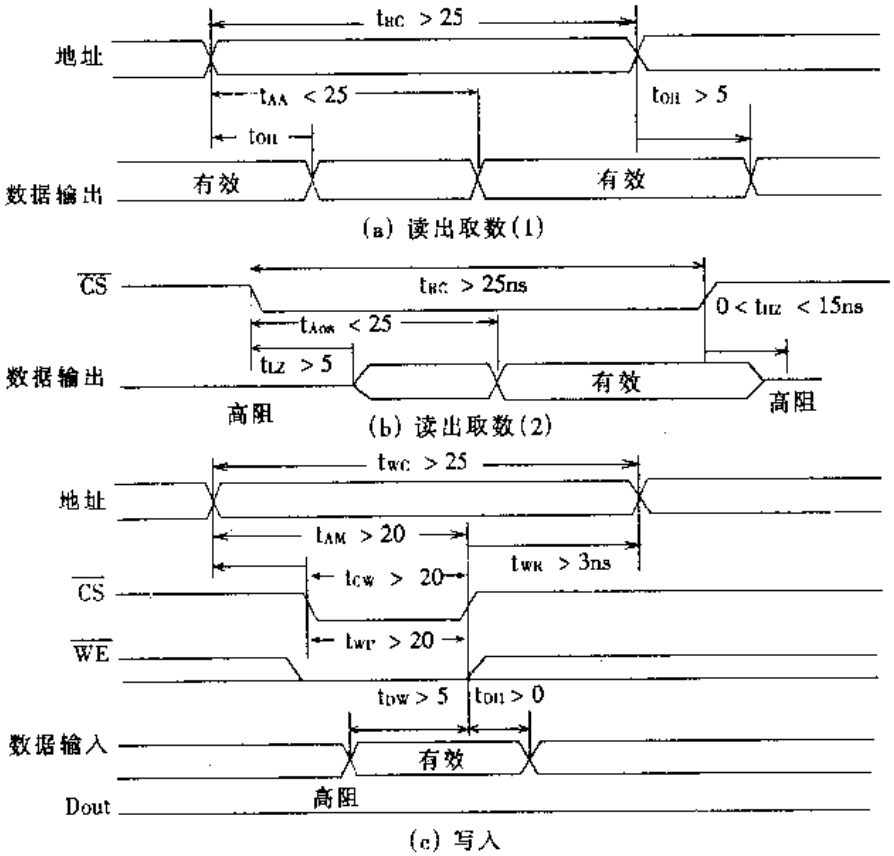


(a) 引脚

记　号	引脚名称
A0 ~ A15	地址输入
I/O1 ~ I/O4	数据输出
CS	片选择
WE	读
Vcc	电源
Vss	地

(b) 引脚说明

图 5 - 28　HM6208 管脚



(a) 读出取数(1)

(b) 读出取数(2)

(c) 写入

图 5 - 29　HM6208 的读/写时序

0 - 7FFFFFH 范围内配置无等待存取状态存储器。对地址空间的存取，将 RDY 信号加到 TMS320C30 则返回。

在图像处理中需大容量存储器时，加入等待使用 SRAM，这时，在 80A000H ~ 0FFFFFH 实装存贮器，为与实装存储器的存取速度相匹配，也可返回 RDY 信号。

在 TMS320C30 中，可通过内部寄存器设定进入无等待存取还是等待存取。这时，可以不

使用 RDY 信号。

5.2 TMS320C40 处理器

一、 TMS320C40 硬件概述

TMS320C40 是 TI 公司 1991 年推出的支持 32 位浮点数的数字信号处理器，其运算速度为 275MOPS(每秒百万次操作)，有着每秒 320M 字节的数据流量，它是真正支持并行操作的数字信号处理器。

TMS320C40 主要特性如下：

(1)服务于高速在片处理器通信的六个通信口

· 最大数据流量时每秒 20M 字节的异步传输速度。

· 处理器对处理器的直接通信。

· 双向传输。

(2)六通道 DMA 协同处理

· 维持 CPU 性能的同时数据传输和 CPU 操作。

· 每条通道的自动初始化能力。

· 可以传来和发送数据去处理器存储映象中的任何地方。

(3)275MOPS 和 320M 字节/秒的高性 DSP CPU

· 每周期 11 次操作。

· 指令周期为 40 或 50ns。

· 40/32 位单周期浮点/整数乘法器。

· 单周期 IEEE 浮点变换。

· 字节和半字节操作能力。

· 支持线性、循环和位反转寻址。

· 单周期分支、调用、返回。

· 单周期桶形移位器。

· 支持高性能的硬件除法和平方根取反。

· 可重定位的复位和中断向量。

· 源代码与 TMS320C3X 兼容。

(4)支持存器分享系统和高速单周期数据传输的两个外部数据和地址总线

· 100M 字节/秒的高速数据传输率。

· 16G 字节的程序/数据/外设寻址空间。

· 四种存储器控制信号可支持不同速度的存储器。

· 相互独立的地址、数据和控制允许引脚。

(5)提高存储器存取性能的在片程序缓冲器和双向单周期可存取 RAM

· 512 字节的指令缓冲器。

· 8K 字节的单周期双向可存取程序或数据 RAM。

· ROM 中固化的引导载入程序。

(6)支持高效并行处理调试技术的在片分析模式

(7)独立的内部程序、数据和 DMA 协同处理器总线

1.TMS320C40 引脚与芯片

TMS320C40 采用 PGA 式封装,有 325 个引脚,引脚及结构框图如图 5－30 所示。各引脚及信号说明如表 5－11 和表 5－12。

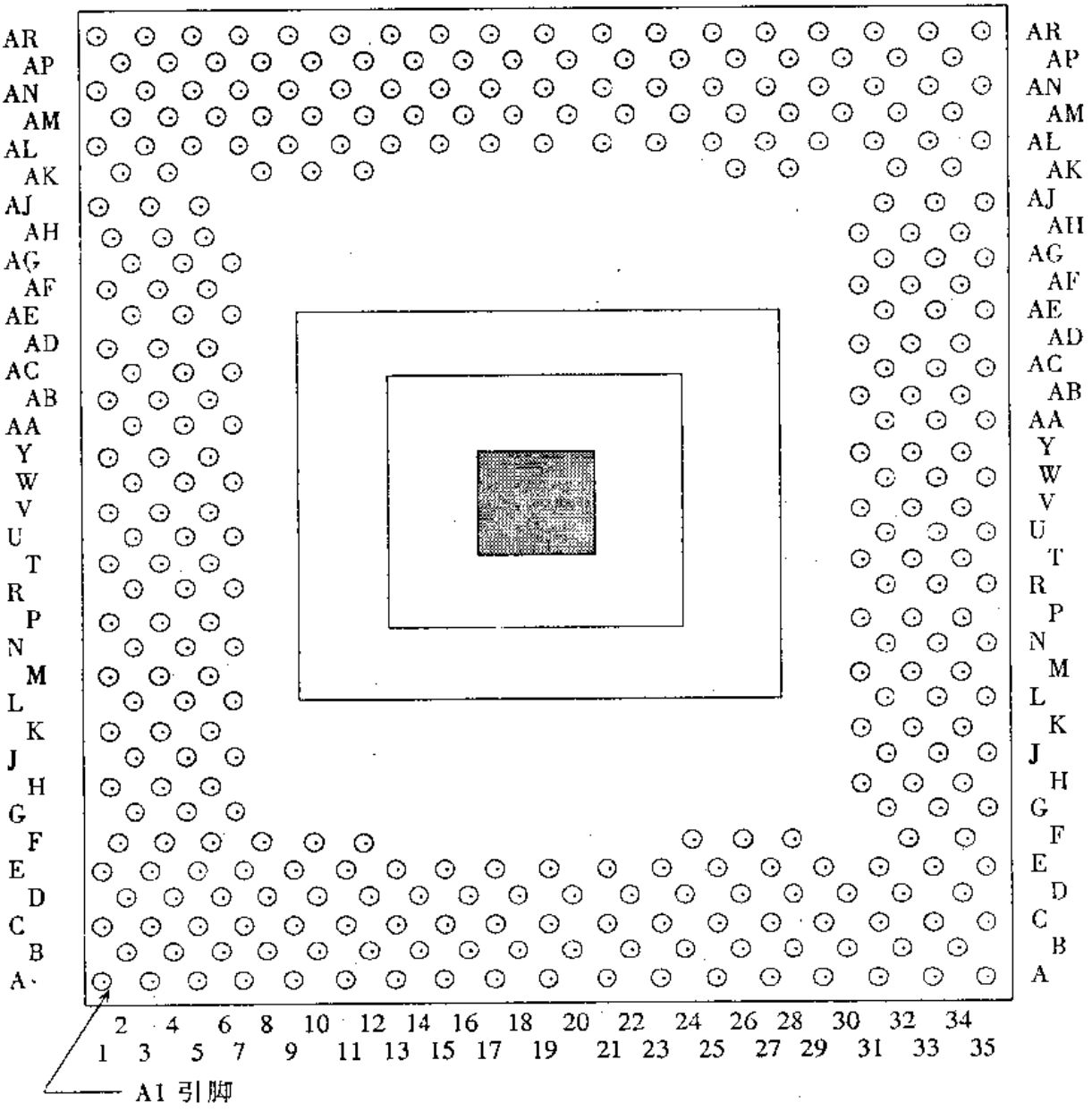


图 5－30 (a) TMS320C40 引脚图

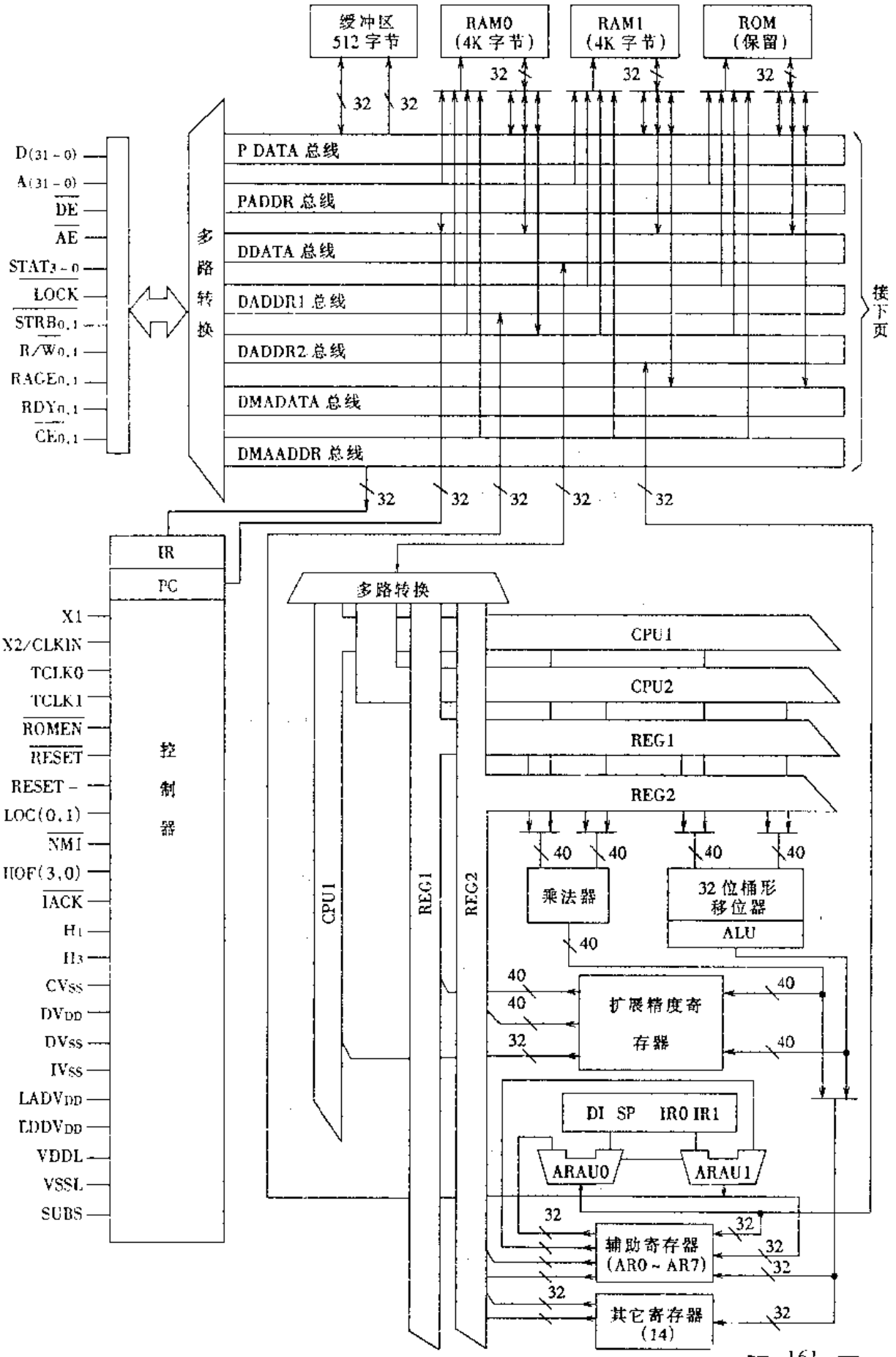
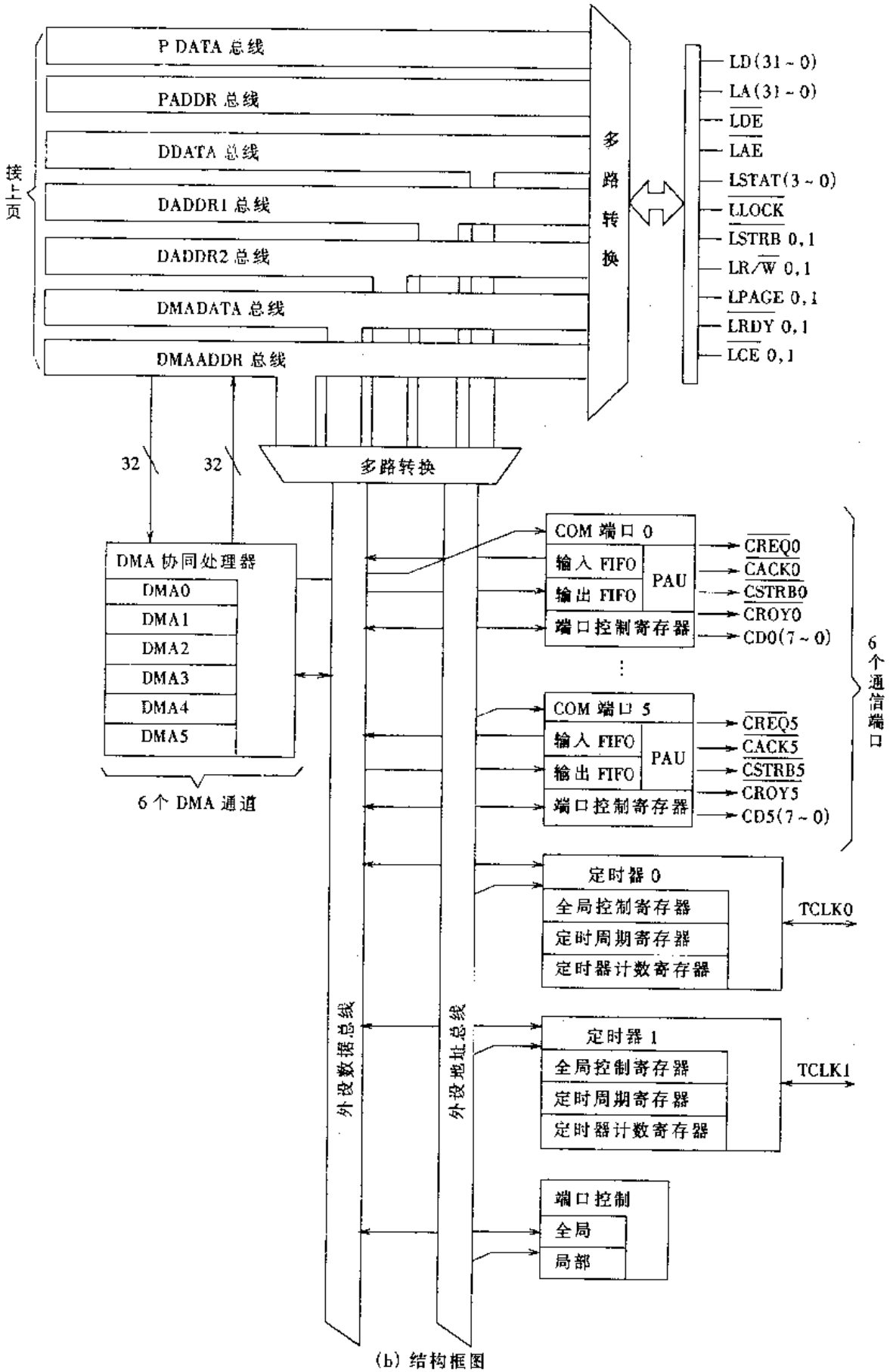

缓冲区
512 字节
RAM0
(4K 字节)
RAM1
(4K 字节)
ROM
(保留)
32
P DATA 总线
PADDR 总线
DDATA 总线
DADDR1 总线
DADDR2 总线
DMADATA 总线
DMAADDR 总线
多路转换
接下页
D(31-0)
A(31-0)
DE
AE
STAT3-0
LOCK
STRB0,1
R/W0,1
RACE0,1
RDY0,1
CE0,1
IR
PC
控制器
X1
X2/CLKIN
TCLK0
TCLK1
ROMEN
RESET
RESET-
LOC(0,1)
NMI
HOF(3,0)
IACK
H1
H3
CVSS
DVDD
DVSS
IVSS
LADVDD
LDDVDD
VDDL
VSSL
SUBS
CPU1
CPU2
REG1
REG2
40
乘法器
32 位桶形
移位器
ALU
扩展精度寄
存器
DI SP IR0 IR1
ARAU0
ARAU1
辅助寄存器
(AR0 ~ AR7)
其它寄存器
(14)

图 5-30 (b)



(b) 结构框图

图 5-30 TMS320C40 引脚及结构框图

表 5－11　TMS320C40 各引脚信号说明

信号	引脚
A0	B32
A1	D32
A2	D30
A3	C29
A4	B30
A5	F28
A6	F24
A7	E29
A8	C27
A9	D28
A10	B28
A11	F26
A12	C25
A13	E27
A14	B26
A15	D26
A16	C23
A17	B24
A18	E25
A19	C21
A20	D24
A21	B22
A22	F23
A23	C19
A24	D22
A25	B20
A26	E21
A27	B18
A28	C17
A29	D20
A30	B16
AE	AC31
C0D0	AP4
C0D1	AL5
C0D2	AN5
C0D3	AM4
C0D4	AP6
C0D5	AM6

信号	引脚
C0D6	AN7
C0D7	AK8
C1D0	AL7
C1D1	AP8
C1D2	AM8
C1D3	AK12
C1D4	AK10
C1D5	AN9
C1D6	AL9
C1D7	AP10
C2D0	AM18
C2D1	AM19
C2D2	AL19
C2D3	AP20
C2D4	AM20
C2D5	AN21
C2D6	A121
C2D7	AP22
C3D0	AM22
C3D1	AM23
C3D2	AL23
C3D3	AP24
C3D4	AM24
C3D5	AN25
C3D6	AL25
C3D7	AP26
C4D0	AN27
C4D1	AM26
C4D2	AK24
C4D3	AL27
C2D4	AP28
C4D5	AK26
C4D6	AN29
C4D7	AM28
C5D0	AL29
C5D1	AP30
C5D2	AK28
C5D3	AN31

信号	引脚
C5D4	AM30
C5D5	AP32
C5D6	AM32
C5D7	AL31
$\overline{\text{CACK0}}$	AN11
$\overline{\text{CACK1}}$	AN13
$\overline{\text{CACK2}}$	AM14
$\overline{\text{CACK3}}$	AM16
$\overline{\text{CACK5}}$	AJ31
$\overline{\text{CE0}}$	AA33
$\overline{\text{CE1}}$	V34
$\overline{\text{CRDY0}}$	AP12
$\overline{\text{CRDY1}}$	AP14
$\overline{\text{CRDY2}}$	AL15
$\overline{\text{CRDY3}}$	AL17
$\overline{\text{CRDY4}}$	AH30
$\overline{\text{CRDY5}}$	AH32
$\overline{\text{CREQ0}}$	AM10
$\overline{\text{CREQ1}}$	AM12
$\overline{\text{CREQ2}}$	AN15
$\overline{\text{CREQ3}}$	AN17
$\overline{\text{CREQ4}}$	AN33
$\overline{\text{CREQ5}}$	AL33
$\overline{\text{CSTRB0}}$	AL11
$\overline{\text{CSTRB1}}$	AL13
$\overline{\text{CSTRB2}}$	AP16
$\overline{\text{CSTRB3}}$	AP18
$\overline{\text{CSTRB4}}$	AM34
$\overline{\text{CSTRB5}}$	AK34
CVss	AR19
CVss	AR7
CVss	N1
CVss	AL35
CVss	A27
CVss	49
CVss	E1
CVss	J35

信号	引脚
CVSS	E35
CVSS	AR25
CVSS	AE1
CVSS	AR13
CVSS	R19
CVSS	R35
CVSS	AL1
D0	U33
D1	V32
D2	T34
D3	U31
D4	R33
D5	P34
D6	T32
D7	N33
D8	R31
D9	M34
D10	P32
D11	L33
D12	N31
D13	K34
D14	M32
D15	J33
D16	L31
D17	M30
D18	K32
D19	H34
D20	J31
D21	G33
D22	K30
D23	F34
D24	H32
D25	E33
D26	D34
D27	G31
D28	C33
D29	H30
D30	E31

信号	引脚
D31	F32
DE	AA31
DVDD	AR11
DVDD	AR29
DVDD	A13
DVDD	A7
DVDD	A17
DVDD	L35
DVDD	AR23
DVDD	A29
DVDD	L1
DVDD	AC1
DVDD	AR17
DVDD	A23
DVDD	AJ1
DVss	AJ35
DVss	A21
DVss	A25
DVss	G35
DVss	A11
DVss	AG1
DVss	AM2
DVss	R1
DVss	AR21
DVss	AR15
DVss	A15
DVss	AR27
DVss	G1
DVss	N35
DVss	AR9
EMU0	AA35
EMU1	AD34
GADVDD	B2
GADVDD	AR1
GADVDD	U35
GDDVDD	V2
GDDVDD	A35
GDDVDD	A1

信号	引脚
H1	AC3
H3	AC5
$\overline{TACK}$	W3
HOF0	AN3
HOF1	AL3
HOF2	AH6
HOF3	AK2
IVSS	AR5
IVSS	AR31
IVSS	AG35
IVSS	A31
IVSS	J1
IVSS	A5
LA	D2
LA1	D4
LA2	E3
LA3	F4
LA4	H6
LA5	F2
LA6	G5
LA7	G3
LA8	H4
LA9	H2
LA10	K6
LA11	M6
LA12	J5
LA13	J3
LA14	K4
LA15	K2
LA16	L3
LA17	L5
LA18	M2
LA19	M4
LA20	N3
LA21	N5
LA22	P2
LA23	P4
LA24	R3

信号	引脚
LA25	R5
LA26	T2
LA27	U3
LA28	T4
LA30	U5
LAVDD	B34
LAVDD	AB2
LAVDD	AP34
LAE	AB4
LCE0	AG5
LCE1	AF2
LD0	E19
LD1	C15
LD2	D18
LD3	B14
LD4	E17
LD5	D16
LD6	D13
LD7	E15
LD8	B12
LD9	D14
LD10	C11
LD11	E13
LD12	B10
LD13	D12
LD14	C9
LD15	E11
LD16	F12
LD17	D10
LD18	B8
LD19	E9
LD20	C7
LD21	F10
LD22	B6
LD23	D8
LD24	C5
LD25	E7

信号	引脚
LD26	B4
LD27	F8
LD28	D6
LD29	C3
LD30	E5
LD31	F6
$LDDV_{DD}$	AR35
$LDDV_{DD}$	AP2
$LDDV_{DD}$	U1
$\overline{LDE}$	AD4
$\overline{LLOCK}$	AA5
$\overline{LOCK}$	W33
LPAGE0	AH2
LPAGE1	AG3
$\overline{LRDY0}$	AF6
$\overline{LRDY1}$	AE5
LR/$\overline{W1}$	AH4
LR/$\overline{W1}$	AF4
LSTAT0	AA3
LSTAT1	Y4
LSTAT2	Y2
LSTAT3	W5
$\overline{LSTRB0}$	AJ3
$\overline{LSTRB1}$	AD6
NMI	AJ5
PAGE0	AG33
PAGE1	AB32
$\overline{RDY0}$	Y32
$\overline{RDY1}$	W31
RESETLOC0	AF30
RESETLOC1	AH34
$\overline{RESET}$	AJ33
ROMEN	AK4
R/$\overline{W0}$	AF32
R/$\overline{W1}$	AC31

信号	引脚
STAT0	AD32
STAT1	AE33
STAT2	AF34
STAT3	AE31
$\overline{STRB0}$	AD30
$\overline{STRB1}$	AC33
SUBS	C31
TCK	Y34
TCLKD	AE3
TCLK1	AD2
TD0	AD34
TD1	AC35
TMS	W35
TRST	AE35
$V_{DD}L$	AN1
$V_{DD}L$	AN35
$V_{DD}L$	C35
$V_{DD}L$	C1
$V_{SS}L$	A3
$V_{SS}L$	AR3
$V_{SS}L$	AR33
$V_{SS}L$	A33
X1	W1
X2/CLKIN	AA1

表 5－12　各引脚信号说明

全局总线外部接口(80 个引脚)

信　号	引脚数	类　型	功　　能
D(31－0)	32	I/O/Z	全局外部接口 32 位数据口
$\overline{DE}$	1	I	全局外部接口数据总线允许信号
A(30－0)	31	O/Z	全局外部接口 31 位地址口
$\overline{AE}$	1	I	全局总线接口地址总线允许信号
STAT(3－0)	4	O	全局总线接口状态信号
$\overline{LOCK}$	1	O	全局总线接口锁信号
$\overline{STRB0}$	1	O/Z	全局总线接口存取选通 0
R/$\overline{W0}$	1	O/Z	$\overline{STRB0}$存取读写信号
PAGE0	1	O/Z	$\overline{STRB0}$存取页信号
$\overline{RDY0}$	1	I	$\overline{STRB0}$存取准备就绪信号
$\overline{CE0}$	1	I	$\overline{STRB0}$,PAGE0 和 R/$\overline{W0}$信号控制允许
$\overline{STRB1}$	1	O/Z	全局总线接口存取选通 1
R/$\overline{W1}$	1	O/Z	$\overline{STRB0}$存取读/写信号
PAGE1	1	O/Z	$\overline{STRB0}$存取页信号
$\overline{RDY1}$	1	I	$\overline{STRB0}$存取准备就绪信号
$\overline{CE1}$	1	I	$\overline{STRB0}$,PAGE0 和 R/$\overline{W0}$信号控制允许

局部总线外部接口(80 个引脚)

信　号	引脚数	类　型	功　　能
LD(31－0)	32	I/O/Z	局部片外接口 32 位数据口
$\overline{LDE}$	1	I	局部片外接口数据总线允许信号
LA(30－0)	31	I	局部片外接口 31 位地址口
$\overline{LAE}$	1	I	局部总线接口地址总线允许信号
LSTAT(0－3)	4	O	部局总线接口状态信号
$\overline{LOCK}$	1	O	局部总线接口锁信号
$\overline{LSTRB0}$	1	O/Z	局部总线接口存取选通 0
LR/$\overline{W0}$	1	O/Z	$\overline{LSTRB0}$存取读写信号
LPAGE0	1	O/Z	$\overline{LSTRB0}$存取页信号
$\overline{LRDY0}$	1	I	$\overline{LSTRB0}$存取准备就绪信号
$\overline{LCE0}$	1	I	$\overline{LSTRB0}$,LPAGE0 和 LR/$\overline{W0}$信号控制允许
$\overline{LSTRB1}$	1	O/Z	局部总线接口存取选通 1

LR/$\overline{W1}$	1	O/Z	$\overline{LSTRB0}$存取读写信号
LPAGE1	1	O/Z	$\overline{LSTRB0}$存取页信号
$\overline{LRDY1}$	1	I	$\overline{LSTRB0}$存取准备就绪信号
$\overline{LCE1}$	1	I	$\overline{LSTRB0}$,LPAGE0 和 LR/$\overline{W0}$信号控制允许

通信端口 0 接口(12 个引脚)

信　号	引脚数	类　型	功　　能
C0D(7-0)	8	I/O	数据总线通信端口 0
$\overline{CREQ0}$	1	I/O	通信端口 0 标志请求信号
$\overline{CACK0}$	1	I/O	通信端口 0 标志请求响应信号
$\overline{CSTRB0}$	1	I/O	通信端口 0 数据选通信号
$\overline{CRDY0}$	1	I/O	通信端口 0 数据准备就绪信号

通信端口 1 接口(12 个引脚)

信　号	引脚数	类　型	功　　能
C1D(7-0)	8	I/O	数据总线通信端口 1
$\overline{CREQ1}$	1	I/O	通信端口 1 标志请求信号
$\overline{CACK1}$	1	I/O	通信端口 1 标志请求响应信号
$\overline{CSTRB1}$	1	I/O	通信端口 1 数据选通信号
$\overline{CRDY1}$	1	I/O	通信端口 1 数据准备就绪信号

通信端口 2 接口(12 个引脚)

信　号	引脚数	类　型	功　　能
C2D(7-0)	8	I/O	数据总线通信端口 2
$\overline{CREQ2}$	1	I/O	通信端口 2 标志请求信号
$\overline{CACK2}$	1	I/O	通信端口 2 标志请求响应信号
$\overline{CSTRB2}$	1	I/O	通信端口 2 数据选通信号
$\overline{CRDY2}$	1	I/O	通信端口 2 数据准备就绪信号

通信端口 3 接口(12 个引脚)

信　号	引脚数	类　型	功　　能
C3D(7-0)	8	I/O	数据总线通信端口 3
$\overline{CREQ3}$	1	I/O	通信端口 3 标志请求信号
$\overline{CACK3}$	1	I/O	通信端口 3 标志请求响应信号

$\overline{CSTRB3}$	1	I/O	通信端口3数据选通信号
$\overline{CRDY3}$	1	I/O	通信端口3数据准备就绪信号

通信端口4接口(12个引脚)

信　号	引脚数	类　型	功　　能
C4D(7-0)	8	I/O	数据总线通信端口4
$\overline{CREQ4}$	1	I/O	通信端口4标志请求信号
$\overline{CACK4}$	1	I/O	通信端口4标志请求响应信号
$\overline{CSTRB4}$	1	I/O	通信端口4数据选通信号
$\overline{CRDY4}$	1	I/O	通信端口4数据准备就绪信号

中断、I/O标志、复位、定时器(12个引脚)

信　号	引脚数	类　型	功　　能
IIOF(3-0)	4	I/O	中断和I/O标志
$\overline{NMI}$	1	I	非屏蔽中断
$\overline{IACK}$	1	O	中断响应
$\overline{RESET}$	1	I	复位信号
RESETLOL(1,0)	2	I	复位向量定位引脚
ROMEN	1	I	在片ROM允许
TCLK0	1	I/O	定时器0引脚
TCLK1	1	I/O	定时器1引脚

时钟和电源(4个引脚)

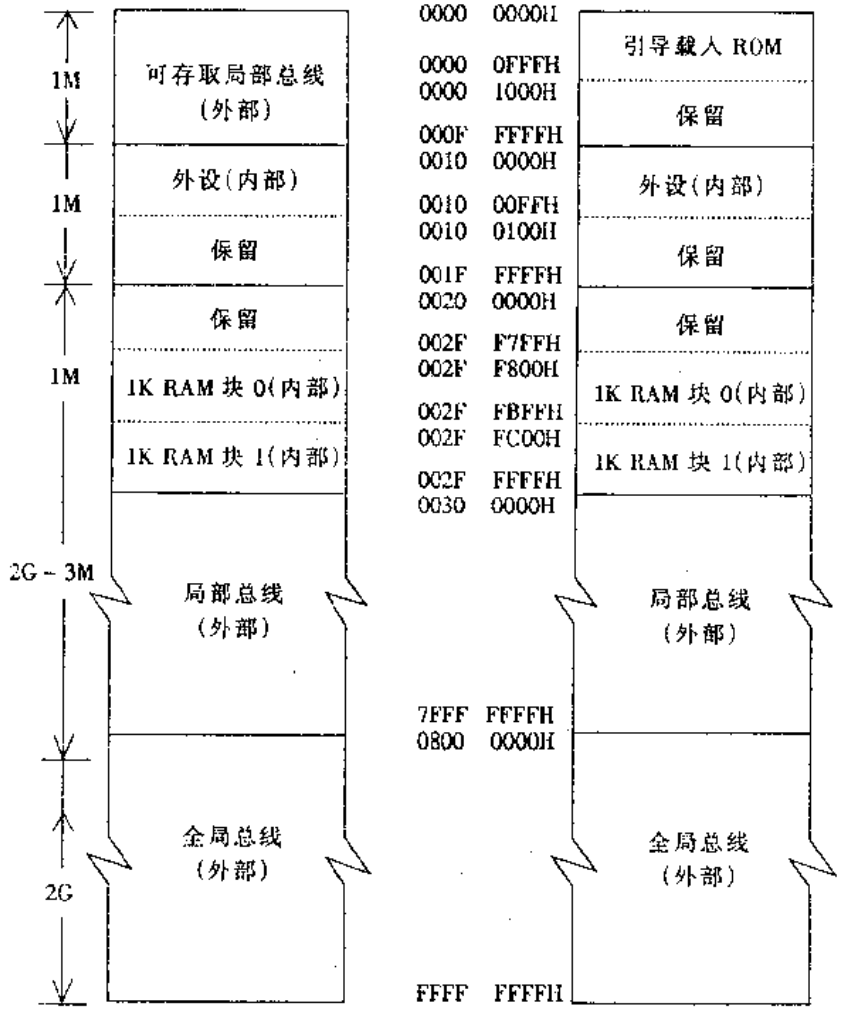
信　号	引脚数	类　型	功　　能
X1	1	O	内部振荡器输出到水晶振子
X2/CLKIN	1	I	晶振或时钟输入到内部振荡器
H1	1	O	H1时钟
H3	1	O	H3时钟

信　号	引脚数	类　型	功　　能
TCK	1	I	JTAG检测端口时钟
TDO	1	I/T	JTAG检测端口数据输出
TDI	1	I	JTAG检测端口数据输入
TMS	1	I	JTAG检测端口方式选择
$\overline{TRST}$	1	I	JTAG检测端口复位
EMU0	1	I	仿真引脚0
EMU1	1	I	仿真引脚1

2. 存储器分配

TMS320C40 的全部可寻址空间为 4G 个 32 位字，包括程、数据、通讯端口、寄存器及 DMA 通道。它允许把程序代码、数据、表格等存入 RAM 或 ROM，其存储器分配如图 5－31 所示。

外部引脚 ROMEN(AK4)置 1 使得内部 ROM 从 0000H 开始的 1M 空间(0000 0000H～000FFFFFH)被保留，置 0 则可作为局部总线而被存取。RAM 块 0 和块 1 各占 4K 字节，相互独立的程序总线，数据总线和 DMA 总线允许并行程序取指，读出或读入数据以及 DMA 操作。保留 ROM 块包括引导载入程序，它在复位时装入程序或数据。128×32 位的指令缓冲区大大减少了外部存取的次数，这使提代码可存入廉价的低速片外存储器。第二个 1M 存储区为外设服务，第三个 1M 存储区包括两个各 1K 的 RAM 块。第一个 2G 空间的其余部分(0030 0000H～7FFF FFFFH)在局部总线上，第二个 2G 空间(8000 0000H～FFFF FFFFH)在全局总线上。从 0010 0000H～0010 00FFH 的 256 个字是外设存储器空间，依次为局部和全局端口控制、分析块寄存器、定时器 0 寄存器、定时器 1 寄存器、通信端口 0～6、DNA 协处理器通道 0～5，它们各占 16 个字。



(a) 片内 ROM 禁止 (ROMEN = 0)　　(b) 片内 ROM 允许 (ROMEN = 1)

图 5－31 TMS320C40 存储空间分配

3. CPU

TMS320C40 采用基于寄存器的 CPU 技术，CPU 包括：

· 浮点数/整数乘法器。

· 执行浮点数，整数和局部操作等运算的 ALU。

· 32 位桶形移位器。

· 内部总线(CPU1/CPU2 和 REG1/REG2)。

· 辅助寄存器运算单元。

· CPU 寄器堆。

①乘法器

乘法器在单指令周期内完成 32 位整数或 40 位浮点数的乘法运算。当执行浮点数运算时，输入和输出都是 40 位浮点数。执行整数运算时，输入是 32 位数，结果是 64 位。TNS320C40 的浮点运算允许浮点操作以定点的速度在 40ns 的指令周期内完成，具有高度的并行性。可使用并行指令执行乘法，ALU 操作在一个周期内 即可完成。

②算术逻辑单元(ALU)

ALU 可在单周期内完成 32 位整数运算、32 逻辑运算和 40 位浮点运算，ALU 的结果通常以 32 位整数或 40 位浮点数形式保存。由于内部总线 CPU1/UPC2 和 REG1/REG2 可执行存储器的两条指令，这就使得并行乘法器和加/减四个整数或浮点数的指令可在一个周期内完成。

③辅助寄存器运算单元(ARAU0 和 ARAU1)

两个辅助寄存器运算单元(ARAU0 和 ARAU1)可在 1 个指令周期内产生两个地址。ARAU 和乘法器及 ALU 一起进行并操作，它支持变址寄存器寻址，循环寻址和位反转寻址。

④CPU 主寄存器堆

TMS320C40 主寄存器堆是一个包含 32 个寄存器的多端口寄存器堆，所有这些寄存器都可以被乘法器或 ALU 操作，也可当作通用寄存器。这些寄存器还具有一些特殊功能，12 个扩展精度寄存器用来保存浮点结果，8 个辅助寄存器用来间接寻址，并可被用作通用 32 位整数或逻辑寄存器，其余的寄存器提供系统控制。和 TMS320C30 相比，C40 多了四个扩展精度寄存器，其它与 C30 的寄存器堆基本相同。

⑤CPU 扩展寄存器推

CPU 扩展寄存器推包括两个专用寄存器：

IVTP 寄存器(指向中断向量表)TVTP 寄存器(指向 TRAP 向量表)。

4. 内部总线操作

TMS320C40 性能高的一个重要原因是它的内部总线并行技术、相互独立的总线允许并行程序读出、数据存取和 DMA 存取。程序总线：PADDR 的 PDATA，数据总线：DADDR1、DADDR2 和 DDATA，DMA 总线：DMAADDR 和 DMADATA，这些总线联系着 TMS320C40 支持的所有物理空间(在片、片外存储器及外设)。程序计数器(PC)联系着 32 位程序地址总线(PADDR)，指令寄存器 IR 联系着 32 位程序数据总线(PDATA)，这些总线可以在一个机器周期内取一个指令字。32 位数据地址总线(DADDR1 和 DADDR2)和 32 位数据数据总线(DDATA)支持每个机器周期内的两次数据存储器存取。DDATA 总线通过 CPU1 和 CPU2 总线把数据送到 CPU，CPU1 和 CPU2 总线可以在一个机器周期内传送两个数据存储器操作数到乘法器、ALU 和寄存器推。同样，REG1 和 REG2 总线可在一个机器周期内从寄存器堆传送两个数值到乘法器和 ALU。32 位地址总线 DMAADDR 和 32 位数据总线 DMADATA 支持 DMA 控制，这两条总线允许 DMA 并行

地执行存储器存取。

5.外部总线操作

TMS320C40提供两个同样的外部接口:全局存储器接口和局部存储器接口,每个接口均包括一条32位数据总线、一条31位地址总线和 两条总线都能被用来寻址外部程序/数据存储器或I/O空间,总线还有外部$\overline{RDY}$信号来产生等待周期。TMS320C40支持4种外部中断(IIOF3-0)、内部中断、非屏蔽中断、外部非屏蔽中断和非屏蔽外部复位信号。

6.外部设备接口

TMS320C40对外设的控制通过外设总线上的存储器映象寄存器完成,外部设备总线由32位数据总线和32位地址总线组成。TMS320C40的外部设备包括两个定时器和两个串行口。

(1)通信端口

六个高速通信端口通过各自的专用通信接口来提供高速的处理器对处理器通信,其技术特性如下:

· 每秒160M位双向数据传输操作。

· 通过8条数据线和4条控制线完成直接处理器对处理器通信。

· 数据传输缓冲器。

· 提供自判定以保证同步通信。

· 通过内部中数据和内部等待信号使得CPU或DMA控制器和六个通信端口同步。

(2)DMA

在片协处理器的六个通道可以不打断CPU的工作而对存储器映象中的任何地址进行读出或写入,这样,当接口到速度较慢的外部存储器或外部设备时可以不减少通过CPU的流量。DMA协处理器包含它自己的地址产生器、源寄存器和目标寄存器,传送计数器、专用的地址总线和数据总线将DMA协处理器与CPU间的冲突降到最小。

(3)定时器

定时器是两个具有内部或外部时钟源的32位通用定时器/事件计数器。每个定时器都有一个I/O引脚,可用来作定时器输入时钟、定时器驱动的输出信号或通用I/O引脚,这与TMS320C30的定时器相同。

二、TMS320C40的指令系统

TMS320C40的数据格式和六种寻址类型与C30完全相同,形成四个寻址方式组:通用寻址方式(G),三操作数寻址方式(T),并行寻址方式(P),条件分支寻址方式(B)。从功能上分,TMS320C40的指令系统与C30同样包括六种指令。

1.装入和存储指令

TMS320C40共有23条置数和存储指令,这些指令可完成:把一个字从存储器装入寄存器,把一个字从寄存器装入存储器,对系统堆栈中数据进行操作。它比TMS320C30的置数和存储指令少一条POP指令,多了如下12条指令:

LDA	装入地址寄存器
LBb	装入有符号字节
LBUb	装入无符号字节
LDEP	装入整数、扩展寄存器堆到主寄存器
LDHI	装入16位无符号立即数到16MSB

LDPE　　装入整数、主寄存器到扩展寄存器堆
LDPK　　装入 DP 寄存器立即数
LHw　　装入半字有符号数
LHUw　　装入半字无符号数
LWLct　　装入字,左移(ct 为移位字节数)
LWRct　　装入字,右移
STIK　　装入整立即数

2.两操作数指令

TMS320C40 共有 43 条两操作数数学和逻辑指令,这两个操作数是源操作数和目标操作数。源操作数可以是一个存储器字、寄存器或寄存器内容,目的操作数是寄存器。这些指令可提供整数、浮点数、逻辑操作和多精度数学运算。它比 TMS320C40 增加的指令如下:

CMPI　　比较整数
FRIEEEE　变 IEEE 浮点格式到双补浮点格式
MBct　　吞入半字,左移
MHct　　吞入半字,左移
MPYSHI　有符号整数乘,32MSB 结果
MPYUHI　无符号整数乘,32MSB 结果
RCPF　　浮点取倒数
RSQRF　　浮点平方根取倒数
TOLKEEE　变双补浮点格式到 IEEE 浮点格式

3.三操作数指令

TMS320C40 有 19 条三操作数指令,比 C30 多了两条;

MPYSHI3　有符号整数乘,32MSB 结果
MPYUHI3　无符号整数乘,32MSB 结果

4.程序控制指令

TMS320C40 控制指令共 23 条,比 C30 多了四条指令:

LAJ　　链接并跳转
LAJcond　链接并条件跳转
LATcond　链接并条件陷阱
RPTBD　　延迟块循环

5.联锁操作指令

TMS320C40 的联锁操作指令与 C30 完成相同

6.并行操作指令

TMS320C40 的并行操作指令比 C30 多两条:

FRIEEE ‖ STF 转变 IEEE 浮点格式并存储
TDIEEE ‖ STF 变成 IEEE 浮点格式并存储

第六章　数字信号处理器的应用

6.1　数字滤波器

一、非递归数字滤波器的实现

非递归数字滤波器可以用线性常系数差分方程来描述：

$$y(n)=\sum_{k=0}^{M}b_k x(n-k) \qquad (6-1)$$

式中 k 是由 0 值开始，(M + 1)为数字滤波器节数。图 6 - 1 表示节数 M + 1 = 5 的滤波器构成图。b_k 是滤波器系数，Z^{-1}支路为延时单元。

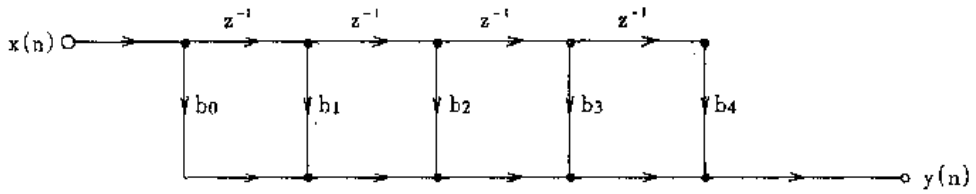


图 6 - 1　直接实现的非递归滤波器(长度为 5)

实时滤波必须考虑运行时间。例如，在语音信号处理中，通常采样速率为 10kHz 左右，相应采样间隔仅为 100μs，该时间间隔是进行实时处理最大允许的间隔时间。TMS32020 的单周期乘法/累加指令与数据传送指令 MACD，以及在片内 RAM，可以在 200ns 内完成每一个滤波节的运算。

TMS32020 的在片 RAM 有 544 字，包括 3 个互相分离的块 B0，B1，B2，其中 B1 与 B2(共 288 字)固定作为数据存储器 DM，B0(256 个字)可根据需要由程序设定为 DM 或 PM，调用指令 CNFD(设置 B0 为 DM，起始地址为 512)及 CNFP(设置 B0 为 PM，起始地址为 65280)可以动态地设置存储器配置。注意使用指令 MACD 前，存储块 B0 必须由指令 CNFP 设置为 PM。MACD 指令仅对片内 RAM 操作，可以加快实时滤波器的处理速度。

用 TMS32020 实现式(6 - 1)时，可以使用重复指令 RPTK 与 MACD 指令对：

```
RPTK   NM1
MACD   pma,dma
```

指令 RPTK 把 1 个 8 位立即数 M 装入重复计数器，从而使下一条指令可以重复执行 M + 1 次(M + 1 为滤波器节数)。

指令 MACD pma，dma 完成下列操作：

(1)把 pma 值装入程序计数器。

(2)用 B1 中 dma 单元的数值乘 B0 块中的程序存储器 pma 单元的内容。

(3)将前次乘积值加到累加器中。

(4)把数据存储器 B0 中的数据复制到高一位地址的对应片内 RAM 内,以实现 Z^{-1}的延时。

(5)每次乘/累加运算后,程序计数器自动增量指向下一个滤波系数 b_k。

图 6-2 为实时滤波器操作时 TMS32020 的存储配置。

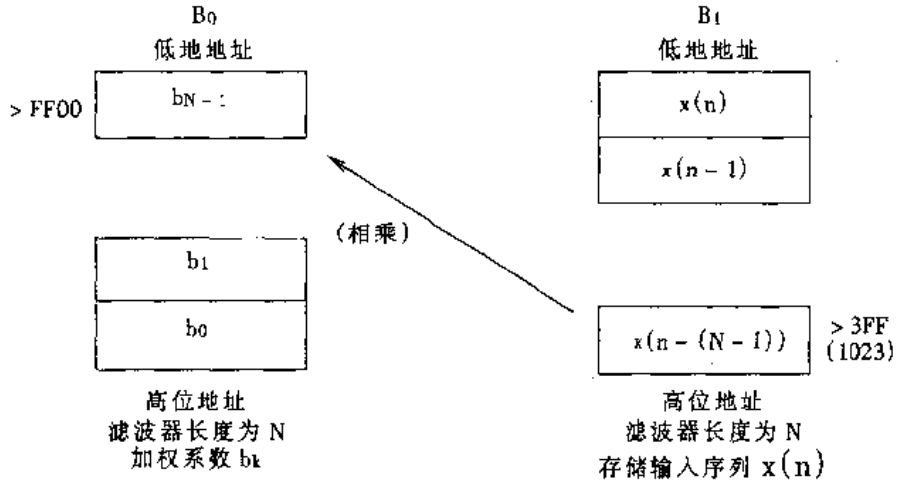


图 6-2 TMS32020 的存储配置

用 TMS32020 汇编语言编写的 5 阶非递归数字滤波器程序如下。程序中用 XN 代表 x(n),XNM1 代表 x(n-1)等等。

```
        CNFP                            ;设置 B0 为程序存储器
NXTPT   IN      XN,PA0                  ;由端口 PA0 取入新样本值
        LRLK    AR1,>3FF                ;设定 B1 底地址
        LARP    AR1
        MPYK    0                       ;设置 P 寄存器为零
        ZAC                             ;清累加器
        RPTK    4                       ;重复 5 次
        MACD    >FF00,*-                ;乘/累加
        APAC
        SACH    YN,1
        OUT     YN,PA1                  ;输出滤波器结果值 y(n)
        B       NXTPT                   ;转入下一小节
```

当执行 MACD 指令时,被操作的数据通过辅助寄存器(AR1)间接寻址。对于低阶(2 阶)数字滤波器,没有必要建立重复循环的操作,通常直接使用指令对 LTD/MPYK。

用 LTD/MPYK 指令对可以实现任何滤波器运算，MPYK 指令用来把 T 寄存器内容乘上 MPYK 指令中给出的 13 位带符号常数。在许多场合下，一个 13 位系数已可以保证滤波器的足够准确度。这样，系数存于 PM 中，而不是 DM。

采用 LTD/MPYK 指令对只要求较少的数据存储空间，但这是以消耗较大程序存储单元为代价的。对于 TMS32020，使用指令对 RPTK/MACD 只需最小的 PM 空间，但为充分发挥指令的潜力，就要使乘法流水线处于满负荷状态，并由存储块 B0 来提供滤波系数 b_k。

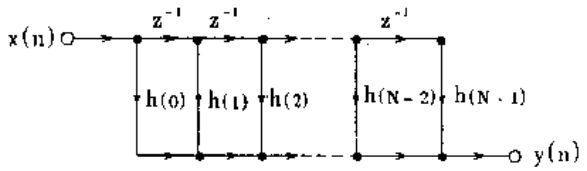
用 TMS32020 实现 80 节 FIR 带通数字滤波器的程序例如下：

线性相位 FIR 80 节带通数字滤波器，采样频率 10kHz。

滤波器参数：

	频带 1	频带 2	频带 3
频带下限	0.0000	1.3750	4.0000
频带上限	1.0000	3.6250	5.0000
标称增益	0.0000	1.0000	0.0000
标称纹波	0.0010	0.0200	0.0010
最大纹波	0.0004	0.0076	0.0004
纹波分贝	-68.3965	0.0657	-68.3997

滤波器结构：



程序如下：

```
        IDT     'FIRBPASS"
YN      EQU     45
MODE    EQU     46
CLOCK   EQU     47
XN      EQU     48
*
        AORG    0
        B       START
*
CTABLE  AORG    32
CH0     DATA    >FFDC       ;滤波器系数
CH1     DATA    >001F
CH2     DATA    >0051
CH3     DATA    >FFE9
```

```
CH4     DATA    > FFE6
CH5     DATA    > FFBA
CH6     DATA    > FFB4
CH7     DATA    > 004B
CH8     DATA    > FFF9
CH9     DATA    > 0069
CH10    DATA    > 00A2
CH11    DATA    > FF6F
CH12    DATA    > FFFE
CH13    DATA    > FF70
CH14    DATA    > FEF4
CH15    DATA    > 00CB
CH16    DATA    > 000B
CH17    DATA    > 00E6
CH18    DATA    > 0187
CH19    DATA    > FEE5
CH20    DATA    > 000B
CH21    DATA    > FE7F
CH22    DATA    > FDBF
CH23    DATA    > 0192
CH24    DATA    > FFB5
CH25    DATA    > 026A
CH26    DATA    > 0368
CH27    DATA    > FDC2
CH28    DATA    > 00C0
CH29    DATA    > FC0A
CH30    DATA    > FAA3
CH31    DATA    > 0347
CH32    DATA    > FE3D
CH33    DATA    > 0747
CH34    DATA    > 09BB
CH35    DATA    > FA3D
CH36    DATA    > 052B
CH37    DATA    > EB59
CH38    DATA    > DC2A
CH39    DATA    > 2D57
CH40    DATA    > 2D57
CH41    DATA    > DC2A
CH42    DATA    > EB59
CH43    DATA    > 052B
```

```
CH44    DATA    > FA3D
CH45    DATA    > 09BB
CH46    DATA    > 0747
CH47    DATA    > FE3D
CH48    DATA    > 0347
CH49    DATA    > FAA3
CH50    DATA    > FC0A
CH51    DATA    > 00C0
CH52    DATA    > FDC2
CH53    DATA    > 0368
CH54    DATA    > 026A
CH55    DATA    > FFB5
CH56    DATA    > 0192
CH57    DATA    > FDBF
CH58    DATA    > FE7F
CH59    DATA    > 000B
CH60    DATA    > FEE5
CH61    DATA    > 0187
CH62    DATA    > 00E6
CH63    DATA    > 000B
CH64    DATA    > 00CB
CH65    DATA    > FEF4
CH66    DATA    > FF70
CH67    DATA    > FFFE
CH68    DATA    > FF6F
CH69    DATA    > 00A2
CH70    DATA    > 0069
CH71    DATA    > FFF9
CH72    DATA    > 004B
CH73    DATA    > FFB4
CH74    DATA    > FFBA
CH75    DATA    > FFE6
CH76    DATA    > FFE9
CH77    DATA    > 0051
CH78    DATA    > 001F
CH79    DATA    > FFDC
MD      DATA    > 000A
SMP     DATA    > 01F3      ;采样频率 10kHz
STAR    EQU     $
*
```

```
* 初始化模拟接口板
        LDPK    7
        LACK    MD
        TBLR    MODE
        OUT     MODE,PA0
        LACK    SMP
        TBLR    CLOCK
        OUT     CLOCK,PA1
* 装入滤波器系数
        LARP    AR0             ;由 AR0 寻址
        LRLK    AR0,>200        ;指向 B0
        RPTK    >4F
        BLKP    CTABLE,*+
        CNFP                    ;B0 为程序区
WAIT    BIOZ    NXTPT
        B       WAIT            ;新样本值有效
NXTPT   IN      XN,PA2          ;得到新样本值 XN
        LRLK    AR1,>3FF        ;指向 B1 末端
        LARP    AR1
        MPYK    0
        ZAC
        RPTK    >4F
        MACD    >FF00,*-
        APAC
        SACH    YN,1
        OUT     YN,PA2          ;输出 y(n)
        B       WAIT            ;指向下一个点
        END
```

从这个程序中可以看出,使用 TMS32020,所占用的程序存储器比使用 TMS32010 要少,但它占用了更多的数据存储器。同时,我们看到,当滤波器的 80 个系数都由 B0 块提供时,滤波器的运算速度相当快。

二、递归数字滤波器

递归型数字滤波器常用线性常系数差分方程来描述:

$$y(n)=\sum_{k=0}^{M}b_k x(n-k)-\sum_{k=1}^{N}a_k y(n-k) \qquad (6-2)$$

其中 M 不一定等于 N。由于滤波器的系数 a_n,b_n 可以从式(6-2)中读出,这种结构形式常称

为滤波器的直接型结构(图 6-3)。

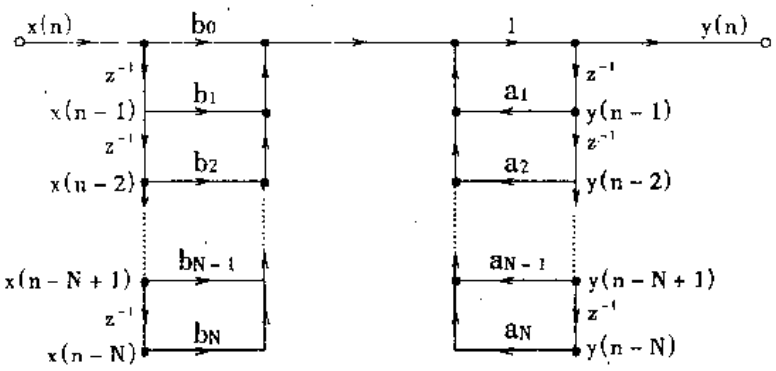


图 6-3 直接 I 型实现的递归滤波器

直接标准型结构如图 6-4 所示,与直接型结构相比,这种结构所需要的延时单元较少,因而计算时占用的存储容量也较少。图 6-4 中,N=M=2,是一个 2 阶递归数字滤波器。

从图 6-4 可以写出其线性差分方程为:

$$\begin{aligned} d(n) &= x(n) + a_1 d(n-1) + a_2 d(n-2) \\ y(n) &= b_0 d(n) + b_1 d(n-1) + b_2 d(n-2) \end{aligned} \qquad (6-3)$$

其中 d(n)表示图 6-4 中延时节点的数值。相应于零延时为 d(n),单位延时值为 d(n-1)等等。

滤波器的每个延迟节点值 d(n)按图 6-5 存放在数据存储器中。实质上,对于递归数字滤波器,实时滤波算法的主要步骤是作一次乘法,然后把乘积加到累加器中,同时将延迟节点值移入下一个较高位地址的数据存储器单元,以便于下一轮操作时将数据准备在正确的地址中。所有这些操作均由下述指令对来完成:

```
LTD     DNM1
MPY     B1
```

这里 DNM1 表示 d(n-1),B1 表示 b_1。当做完最后一次乘法并将乘积加入累加器后,累加器中内容即为实时滤波的结果 y(n)。由式(6-3)可见,延时节点值 d(n)取决于前几个延时节点的数值,这种反馈可由

```
SACH    DN,1
```

以及语句

```
LTD     DNM1
…
LTD     DN
```

来完成。

当 M≥N 时,直接标准型结构递归数字滤波器的延时节点排列如图 6-6 所示。此时,滤波器阶数取决于 M,N 中较大的一个。

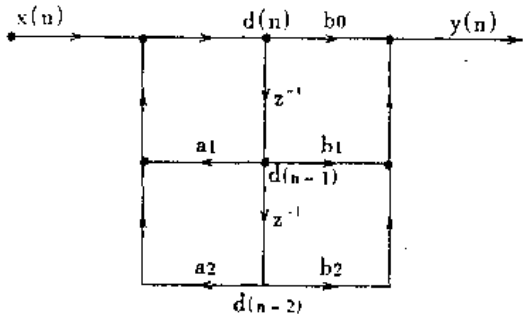


图 6-4　直接标准型结构递归滤波器

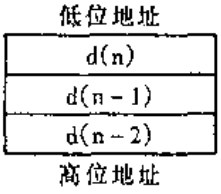


图 6-5　2 阶递归滤波器直接标准型实现的延时节点数值的存放

用 TMS32020 汇编语言编写的 2 阶递归数字滤波器(IIR)直接标准型实现的程序如下。程序中使用了 MACD 指令。与非递归时一样，当滤波器阶数大于等于 3 时，使用 RPTK/MACD 指令效率最高。在此程序中使用 XN 表示 x(n)，YN 表示 y(n)等。

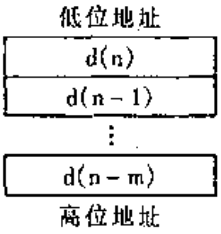


图 6-6　递归滤波器直接标准型实现的延时节点数值的存放

```
NEXT    IN      XN,PA2          ;由端口 PA0 取入新样本值 x(n)
        LAC     XN              ;清 P 寄存器
        MPY     K0
        LARP    AR1
        LRLK    AR1,>03FF
        CNFP                    ;设定 B0 为 PM
*       d(n)=x(n)  +a1d(n-1)+a2d(n-2)
        RPTK    1               ;重复 2 次
        MACD    >FF00,*+
        APAC
        SACH    DN,1            ;*d(n)
```

```
ZAC
MPYK    0                ;清 P 寄存器
MPY     > FF02
RPTK    1
MACD    > FF03, * -
APAC
SACH    YN,1             ;保留滤波器输出值
OUT     YN,PA2           ;y(n)为滤波器的输出
B       MEXT
```

递归数字滤波器经常使用级联型与并联型结构。数字滤波器传递函数 H(Z)可写为：

$$H(z)=\prod_{k=1}^{N/2}\frac{\beta_{0k}+\beta_{1k}z^{-1}+\beta_{2k}z^{-2}}{1-a_{1k}z^{-1}-a_{2k}z^{-2}} \tag{6-4}$$

即递归滤波器(IIR)可以由一系列双二次型(a series of biquads)滤波单元的级联来实现。图 6-7 表示一个 4 阶递归数字滤波器的级联型式的网络，其每一级均是一个 2 阶基本节。

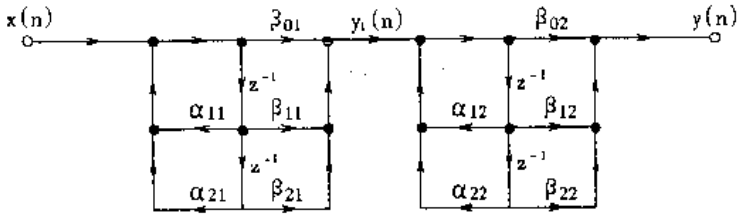


图 6-7　4 阶级联递归滤波器

同样，H(z)也可写为：

$$H(z)=\sum_{k=0}^{M-N}C_kz^{-k}+\sum_{k=1}^{(N+1)/2}\frac{\gamma_{0k}+\gamma_{1k}z^{-1}}{1-\alpha_{1k}z^{-1}-\alpha_{2k}z^{-2}} \tag{6-5}$$

则可画出图 6-8 所示并联滤波器结构，这里设 M = N = 4，输入是否与一常数 c 相乘是无关紧要的。每一个单独的并联支路与一个 2 阶基本节十分相似。显然，将图 6-8 中每一并联支路的输出值相加即得 y(n)。

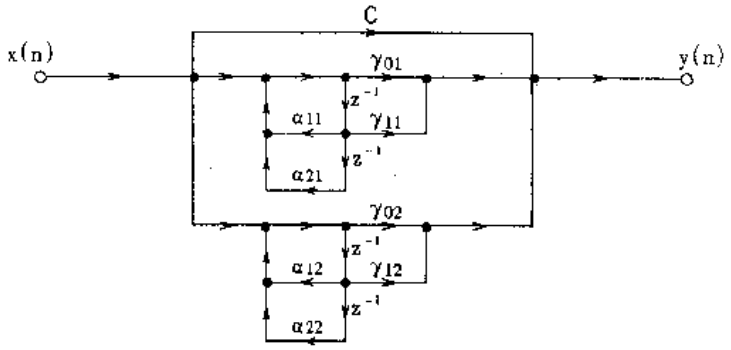


图 6－8　并联型递归滤波器

三、IIR 数字滤波器程序例

串连型 IIR 滤波器

4 阶 IIR 椭圆低通滤波器，基本二阶节级联结构，采样频率 10kHz。

滤波器参数：

	频带 1	频带 2
频带下限	0.00000	2.75000
频带上限	2.50000	5.00000
标称增益	1.00000	0.00000
标称纹波	0.06000	0.06000
最大纹波	0.05617	0.05514
纹波分贝	0.47469	－25.17089

滤波器结构：(参见图 6－7)。

程序如下：

```
        IDT         'IIR4CAS'
D2N                 EQU         0
D2NM1               EQU         1
D2NM2               EQU         2
D1N                 EQU         3
D1NM1               EQU         4
D1NM2               EQU         5
*
B01                 EQU         6
B11                 EQU         7
B21                 EQU         8
A11                 EQU         9
A21                 EQU         10
```

```
B02             EQU     11
B12             EQU     12
B22             EQU     13
A12             EQU     14
A22             EQU     15
MODE            EQU     16
CLOCK           EQU     17
YN              EQU     18
XN              EQU     19
ONE             EQU     20
AORG            0
B               START
* 系数初始化存储到程序存储器
CB01    DATA    >1F05       ;0.242342
CB11    DATA    >2B75       ;0.339521
CB21    DATA    >1EF0       ;0.242117
CA11    DATA    >394D       ;0.447687
CA21    DATA    >D889       ;-0.308310
CB02    DATA    >62F1       ;0.772990
CB12    DATA    >26DB       ;0.303581
CB22    DATA    >62ED       ;0.772587
CA12    DATA    >FEF7       ;-0.008080
CA22    DATA    >90EE       ;-0.867723
MD      DATA    >000A
SMP     DATA    499         ;采样频率 10kHz
*
START   LDPK    0
        LACK    1
        SACL    ONE         ;设定常数 ONE 为 1
        LARK    AR0,CLOCK   ;将滤波器系数及其他值从 PM→DM
        LARK    AR1,11
        LACK    SMP
        LOAD    LARP        AR0
        TBLR    *-,AR1
        SUB     ONE
        BANZ    LOAD
        ZAC
        SACL    D2N         ;置滤波器初始状态到 0
        SACL    D2NM1
        SACL    D2NM2
```

```
        SACL    D1N
        SACL    D1NM1
        SACL    D1NM2
        OUT     MODE,PA0        ;初始化模拟接口板
        OUT     CLOCK,PA1
WAIT    BIOZ    NXTPT
        B       WAIT            ;新样本有效
NXTPT   IN      XN,PA2          ;取新样本 XN
        LAC     XN,15           ;开始第一个基本节
        LT      D1NM1
        MPY     A11             ;d1(n-1)*an
        LTA     D1NM2
        MPY     A21             ;d2(n-1)*a12
        APAC
        SACH    D1N,1
        ZAC
        MPY     B21
        LTD     D1NM1
        MPY     B11
        LTD     D1N
        MPY     B01             ;结束第一个二阶节
* 开始第二个二阶节
        LTA     D2NM1
        MPY     A12
        LTA     D2NM2
        MPY     A22
        APAC
        SACH    D2N,1
        ZAC
        MPY     B22
        LTD     D2NM1
        MPY     B12
        LTD     D2N
        MPY     B02
        APAC
        SACH    YN,1            ;完成第二个二阶节
*                               ;结束滤波
        OUT     YN,PA2          ;输出 y(n)
        END
```

6.2 采用 TMS320C25 实现 FFT

一、FFT 算法概述

离散傅利叶变换将时域信息转换为频域信息，而离散傅利叶反变换则将频域信息转换为时域信息。

序列 x(n)的离散傅利叶变换可写为：

$$X(k)=\sum_{n=0}^{N-1}x(n)\ W_N^{kn} \qquad k=0,1,2,\cdots\cdots,N-1 \qquad (6-6)$$

相应的，离散傅利叶反变换可记为：

$$X(n)=1/N\sum_{n=0}^{N-1}X(k)\ W_N^{-kn} \qquad n=0,1,2,\cdots\cdots,N-1 \qquad (6-7)$$

式中，W_N 为转换因子，并记为：

$$W_N=e^{-j(2\pi/N)} \qquad (6-8)$$

由式(6-6)可以看出，离散傅利叶变换是由大量的乘法、累加等基本运算构成的，从计算时间来看，直接计算 N 个采样值的离散傅利叶变换，需做 N×N 次乘和 N×(N-1)次累加。为减少运算时间，提出了以减少乘法运算次数为主要目标的快速傅利叶算法(FFT)。

通常使用的 FFT 算法有时域抽取法(DIT)和频域抽取法(DFT)。其算法的关键是将 N 点 DFT，分解为两个较小的 DFT 的线性组合，每一个仅包含 N/2 个采样值。当 N 取为 $N=2^r$ 时，分解可进行 r 次，并用一种称为蝶型运算的信号流程图来表示算法。这样，可将乘法的次数从直接计算时的 N×N 次减少到 $N/2\cdot\log_2^N$ 次。当 N 值较大时，其效果极为显著。

基 2-时域抽法(DIT)是将 X(n)按 n 为奇数和偶数分为 X(2r)、X(2r+1) (r=0,1,2,……N/2-1 两个 N/2 长度的序列，式(6-6)可写为：

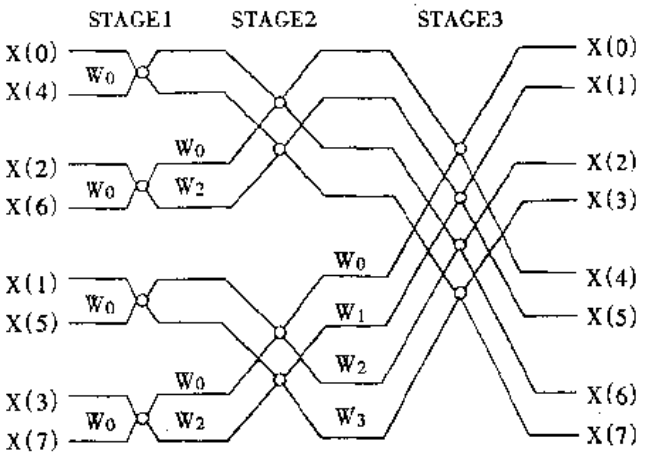
$$\begin{aligned}X(k)&=\sum_{k=0}^{N-1}x(n)W_N^{kn}\\&=\sum_{r=0}^{N/2-1}x(2r)W_N^{2rk}+\sum_{r=0}^{N/2-1}x(2r+1)W_N^{(2r+1)k}\\&=\sum_{r=0}^{N/2-1}x(2r)W_{N/2}^{rk}+W_N^k\sum_{r=0}^{N/2-1}x(2r+1)W_{N/2}^{rk}\\X(K+N/2)&=\sum_{r=0}^{N/2-1}x(2r)W_{N/2}^{rk}-W_N^k\sum_{r=0}^{N/2-1}X(2r+1)W_{N/2}^{rk}\\&\qquad k=0,1,2,\cdots,N-1\end{aligned} \qquad (6-9)$$

基 2-频域抽法(DFT)是将 X(n)按 n=2r 分为两半，每个长度均为 r，这样，式(6-6)可写为：

$$X(k)=\sum_{k=0}^{N-1}x(n)W_N^{kn}$$

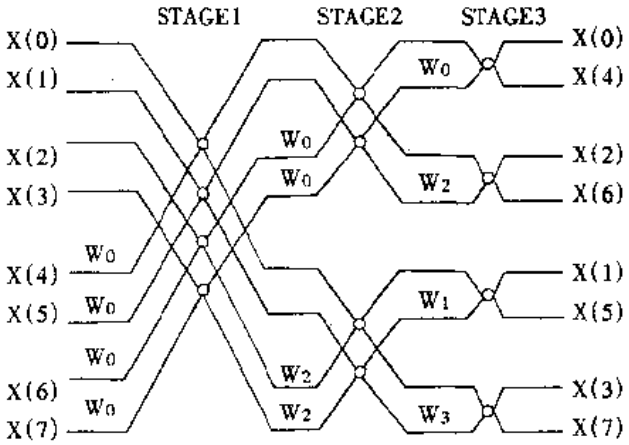
$$
\begin{aligned}
&= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + \sum_{n=N/2}^{N-1} x(n) W_N^{kn} \\
&= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + \sum_{n=0}^{N/2-1} x(n+N/2) W_N^{(n+\frac{N}{2})k} \\
&= \sum_{n=0}^{N/2-1} x(n) W_N^{nk} + (-1)^k \sum_{n=0}^{N/2-1} x(n+N/2) W_N^{nk} \\
&= \sum_{n=0}^{N/2-1} [x(n) + (-1)^k x(n+N/2)] W_N^{nk} \\
&\qquad k = 0,1,2,\cdots,N-1
\end{aligned} \tag{6-10}
$$

例如,8 个采样点的 FFT 运算流程如图 6-9 所示。



LEGEND FOR TWIDDLE FACTOR: $W_0 = W_8^0$ $W_1 = W_8^1$ $W_2 = W_8^2$ $W_3 = W_8^3$

(a) 输入乱序,输出正序



LEGEND FOR TWIDDLE FACTOR: $W_0 = W_8^0$ $W_1 = W_8^1$ $W_2 = W_8^2$ $W_3 = W_8^3$

(b) 输入正序,输出乱序

图 6-9 8 点 DIT FFT 流程图

从图 6-9 中可以看出,FFT 运算的输入和输出不是同序的。即输入"乱序"时,输出正序;输入正序时输出是"乱序"的。因而,在计算 FFT 时,常需采用位反转(码的位置)的方法进行地

址整序。例如 8 个采样值的 FFT 而言，对输入进行整序的(不等位倒置)方法如表 6－1 所示。

表 6－1　8 点基 2 DIT FFT 位反转算法

原序号	位　排　序	码位倒置	码位倒置后序号
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

TMS32020 实现码位倒置的方式是将需要交换的两个点装入累加器，然后将其交换后，存入原来的存储单元中，在数据通过 I/O 口输入或输出时完成整序。这种寻址方式是通过辅助寄存器和算术逻辑单元而实现的间接寻址方式。这时，由 ARP 指向的辅助寄存器中加上(或减去)AR0 的值(即序号)，就可以得到整序后的存取地址。

二、FFT 实例程序

本例是一个利用 TMS320C25 的码位倒置(位反转)寻址功能进行的 8 点 DIT FFT 运算程序。在 TMS320C25 上，使用指令对进行数据输入并将其按位反转顺序存入存储器：

```
RPTK    7
IN      * BR0 + , PA0
```

该指令对完成的功能相当于前面给出的 TMS32020 上进行的输入和码位倒置寻址两部分的组合。它也可把两部分分开进行：

```
RPTK        7
IN          * 0-, PA0
BITREV      X1R, X1T, X4R, X4T
BITREV      X3R, X3T, X6R, X6T
```

程序如下：

```
LIST: TMS320C25 8 点 FFT 运算程序
    X0R         EQU  00
    X0I         EQU  01
    X1R         EQU  02
    X1I         EQU  03
    X2R         EQU  04
    X2I         EQU  05
    X3R         EQU  06
    X3I         EQU  07
    X4R         EQU  08
```

```
X4I         EQU  09
X5R         EQU  10
X5I         EQU  11
X6R         EQU  12
X6I         EQU  13
X7R         EQU  14
X7I         EQU  15
W           EQU  16
WVALUE      EQU  >5A82
 * 初始化
FFT         SPM      0              ;PR 非移位输出
            SSXM
            ROVM
            LDPK     4              ;置数据页指针到 4
            LALK     WVALUE         ;取旋转因子值
            SACL     W              ;存储 SIN(45)或 COS(45)
* 输入样本,按位反序存储
            LARK     AR0,8          ;在 AR0 内装入 FFT 长度
            LRLK     AR1,>200       ;装入 AR1,以页 4 寻址
            LARP     AR1
            RPTK     7
            IN       *bR0+,PA0      ;输入一个实数
* 第一、二级除 4 隔行定标
            COMBO    X0R,X0I,X1R,X1I,X2R,X2I,X3R,X3I
            COMBO    X4R,X4I,X5R,X5I,X6R,X6I,X7R,X7I
* 第 3 级除 2 隔行定标
            ZERO     X0R,X0I,X4R,X4I
            PIBY4    X1R,X1I,X5R,X5I,W
            PIBY2    X2R,X2I,X6R,X6I
            PI3BY4   X3R,X3I,X7R,X7I,W
* 输出样本,正序输出
            LRLK     AR1,>200       ;数据页 4 寻址装入 AR1
            RPTK     15
            OUT      *+,PA0         ;输出复数值
            RET
```

6.3 数据压扩

在 PCM 通信系统中,通常进行数据压扩。数据压扩是将信号以对数因子进行压缩,待处

理时再扩展为原信号的操作。

常用的数据压扩方式如图 6－10 所示。其中，图 6－10(a)(b)是使用编/译码器或编译码器组合以硬件方式实现的。图 6－10(c)为使用数字信号处理器方式。数字信号处理器可用最短的时间和最小的程序实现压扩功能。一般而言，数据压扩软件中常使用两种算法，一是实时运算法，二是查表法。查表法占用存储空间多，但处理速度快。在编写程序时，可以任选其中的一种算法。

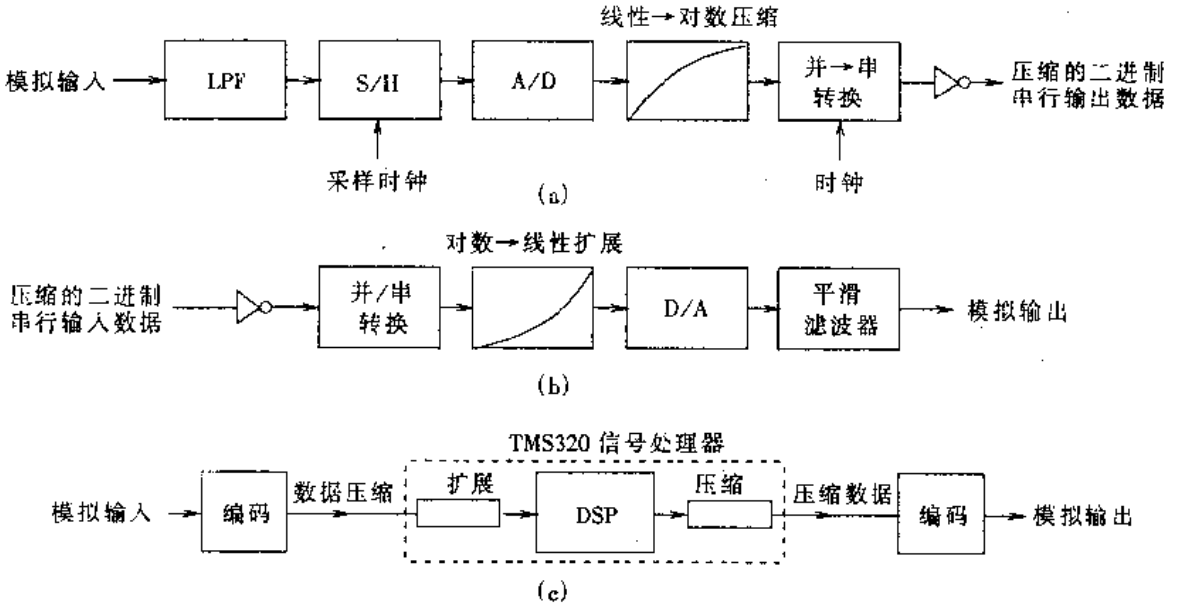


图 6－10　数据压扩方式

一、数据压扩方式

目前，国际标准规定的数据压扩为，对应 13 位的动态压缩到 7 位。美国和日本采用的是 $\mu-255$ 标准。其压扩特性为：

$$F(x)=\operatorname{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \tag{6-11}$$

式中，F(x)是压缩输出值，x 是输入信号(1～－1)，μ 是压缩系数(取为 255)，sgn(x)为符号函数。

欧洲多采用 A 律压缩标准，其特性为：

$$F(x)=\begin{cases}\operatorname{sgn}(x)\dfrac{A|x|}{1+\ln(A)} & 0\leqslant|x|<\dfrac{1}{A}\\[2ex] \operatorname{sgn}(x)\dfrac{1+\ln A|x|}{1+\ln(A)} & \dfrac{1}{A}\leqslant|x|\leqslant 1\end{cases} \tag{6-12}$$

式中，F(x)是压缩输出值，x是输入信号(1～－1)，A是压缩系数(欧洲取87.6)，sgn(x)是符号函数。

二、用TMS32020系列实现数据压扩的程序

1.μ律压扩程序(TMS32020)：

```
*
DRR       EQU     0
DXR       EQU     1
IMR       EQU     4
S         EQU     96          ;μ律段数=33
BIAS      EQU     97
X         EQU     98
SUM       EQU     99
SIGN      EQU     100
NEG7      EQU     101
Q         EQU     102         ;μLAW QUANTIZATION BIN
BIAS2     EQU     103
          AORG    0
RSVECT    B       INIT
          AORG    26
EXPAND    BIT     DRR,8       ;测试DRR符号
LAC       DRR,12
ANDK      >7F00,4             ;符号位及其它MSB位置0
XORK      >7F00,4             ;全位倒置
          SACH    S
          SUBH    S           ;ACCH变0
          ADD     BIAS,11
          SFL
          SACH    SUM,4
          LT      S
          LACT    SUM
          SUB     BIAS
          BBNZ    POSVAL      ;TC=1时为正
          NEG
POSVAL    SACL    X
*
JSTIFY    LAC     X,2         ;左移14位数值
          SACL    X
*
COMPRS    ZALH    X           ;μ律压缩
```

```
BLZ     NEGCMP
        ADDH    BIAS2
        LAR     AR0,NEG7
        RPTK    6               ;寻找 MSB
        NORM
        ANDK    >F000,14        ;高 2 位和所有低位置 0
        SACH    Q
        SAR     AR0,5
        ZALH    S
        ABS
        ADD     Q,2
        XORK    >FF00,4         ;全部倒置
        B       SATCH
*
NEGCMP  ABS
        ADDH    BIAS2
        LAR     AR0,NEG7
        RPTK    6               ;搜索 MSB
        NORM
        ANDK    >F000,14        ;2 个 MSB 位及全部 LSB 位置 0
        SACH    Q
        SAR     AR0,S
        ZALH    S
        ABS
        ADD     Q,2
        XORK    >7F00,4         ;Q 全位倒置
SATCH SACH DXR,4
WAIT FINT
        IDLE                    ;对 RINT 等待
* 初始化程序
        AORG    1024
INIT    LDPK    0               ;DP 指向 B2 和 MMRS
        LACK    >10
        SACL    IMR             ;允许 RINT,禁止其它
*       FORT    1
        SOVM
        SSXM
        SPM     0
        LACK    33
        SACL    BIAS
```

```
        LAC     BIAS,2
        SACL    BIAS2
        LALK    -7
        SACL    NEG7
        LACK    0
        SACL    DRR             ;DRR 的 MSBS 置 0
        SACL    DXR             ;对第 1 个传送操作 DXR 置 0
        LARP    0               ;ARP 及 ARB 置 0
        LARP    0
        EINT
        IDLE    ;等待 RINT
        END
```

2.A 律压扩程序(TMS32020)

```
        *
DRR     EQU     0
DXR     EQU     1
IMR     EQU     4
S       EQU     96
BIAS    EQU     97
X       EQU     98
SUM     EQU     99
SIGN    EQU     100
NEG7    EQU     101
Q       EQU     102             ;A 律量化储存器
ONE     EQU     103
        AORG    0
RSVECT  B       INIT
        AORG    26
EXPAND  BIT     DRR,8           ;DRR 符号测试
        LAC     DRR,12
        ANDK    >7F00,4
        XORK    >5500,4
        SACH    S
        SUBH    S               ;ACCH 为 0
        LAR     AR0,S
        BANZ    SEGNZ,*-        ;测试 0 号段
        ADD     ONE,11
        SFL
        BBZ     PSVAL1
```

```
        NEG
PSVALI  SACH    X,4
        B       JSTIFY
*
SEGNZ   SAR     AR0,S           ;存储减量 S
        ADD     BIAS,11
        SFL
        SACH    SUM,4
        LT      S
        LACT    SUM
        BBZ     POSVAL
        NEG
POSVAL  SACL    X
*
JSTIFY  LAC     X,3
        SACL    X
*
COMPRS  ZALH    X
        BIZ     NEGCMP
        LAR     AR0,NEG7
        RPTK    6               ;寻找 MSB
        NORM
        BANZ    SEGNZ1,*
        SACH    Q               ;段数=0
        LAC     Q,1
        XORK    >5500,4         ;偶次位反转
        B       SATCH
SEGNZ1  ANDK    >F000,14
        SACH    Q
        SAR     AR0,S
        ZALH    S
        ABS
        ADD     Q,2
        XORK    >5500,4
        B       STACH
*
NEGCMP  ABS
        LAR     AR0,NEG7
        RPTK    6               ;FIND MSB
        NORM
```

```
                BANZ    SEGNZ2,*
                SACH    Q               ;段数=0
                LAC     Q,1
                XORK    >D500,4         ;偶数位反转,符号位置1
*
                B       SATCH
SEGNZ2          ANDK    >F000,14
                SACH    Q
                SAR     AR0,S
                ZALH    S
                ABS
                ADD     Q,2
                XORK    >D500,4         ;偶数位反转,符号位置1
*
SATCH           SACH    DXR,4
*
WAIT            EINT                    ;WAIT FOR A RINT
                IDLE
*
*
* 初始化程序
*
*
                AORG    1024
INIT            LDPK    0
                LACK    >10
                SACL    IMR
*
                FORT    1
*
                SOVM
                SSXM
                SPM     0
                LACK    1
                SACL    ONE
                LACK    33
                SACL    DIAS
                LACK    -7
                SACL    NEG7
                LACK    0
```

```
SACL    DRR             ;ZERO MASBS OF DRR
SACL    DXR             ;ZERO DXR FOR FIRST
LARP    0               ;ZERO ARP
LARP    0               ;AND ARB
EINT    ;WAIT FOR RINT
IDLE
END
```

6.4 双 DSP 实时数字相关处理系统

这里介绍一种采用双端口共用存储器的双 TMS320C25 微处理器系统及其在实时数字相关中的应用。该系统可用于检测传输线的噪声以及通过对不同点振动信号的相关计算实现对汽车、飞机的振动测试。

一、相关理论

相关性是指两个信号波之间的相似程度，可用于从噪声中提取周期信号，建立随机信号的相关性及建立信号源等。信号的自相关函数是指信号与其延时信号之间的相似性，它是一个时间函数。无论是随机信号还是周期信号，任何信号的自相关函数都不仅与波形有关，而且还与其频率有关。

信号 a(t)的自相关函数定义为：

$$R_{aa}(\tau)=\lim_{T\to\infty}(1/T)\int_0^T a(t)*a(t-\tau)dt \qquad (6-13)$$

即：信号 a(t)与其延时变换式相乘，乘积在 T 秒内积分的平均值。对数字系统而言，平均是指把信号以△t 秒周期采样，然后将 N 个采样点值相乘并求和，即：

$$R_{aa}(\tau)=(1/N)\sum_{k=1}^{N}a(k\triangle t)a(k\triangle t-\tau) \qquad (6-14)$$

式中可取不同的 τ 值进行计算。τ 值的取值范围取决于信号 a(t)的带宽。例如，1MHz 信号自相关函数的 τ 值可从 0 至 100μs，此时分辨率为 100ns。

两个不同的信号波 a(t)和 b(t)的互相关函数为：

$$R_{ba}(\tau)=(1/N)\sum_{k=1}^{N}a(k\triangle t)b(k\triangle t-\tau) \qquad (6-15)$$

在线相关有利于得到检测结果。下面介绍基于在线数字相关而设计的双 TMS320C25 处

理器系统。

二、双处理器结构

一个用双端口存储器进行通信的双 TMS320C25 实时数字相关分析系统的框图如图 6－11。这种双处理器结构的特点是允许两个分离的处理器在各自的时钟下用同一数据空间进行异步操作。

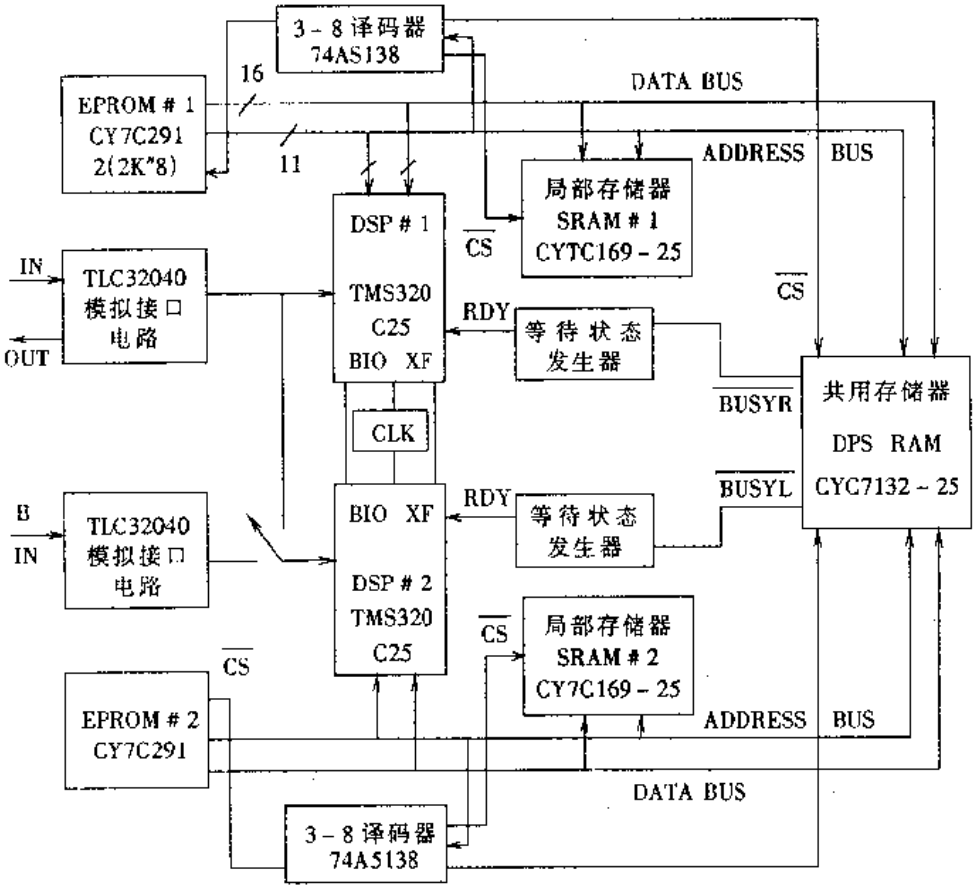


图 6－11　双 DSP 系统方框图

1.存储器系统

2K×16 位程序存储器由两片 TBP38L165－35(或 Cypress CYTC 292－35)CMOS EPROM 构成。该 EPROM 对有效地址的存取时间为 35ns,片选时间为 25ns,可直接与 TMS320C25(钟频 40MHz)连接。4K×16 位的局部数据存储器使用 Cypress CYTC169－25 构成。这些 4K×4 的静态 RAM(SRAM)对有效地址的存取时间为 25ns,片选时间为 15ns,也无需使用等待状态发生器。双端口静态 RAM CYC 132－25 和 CYTC 142－25 构成 2K×16 位全局存储器。

2.双端口存储器

高速双端口静态 RAM 提供并行(重复)地址总线和数据总线(在双处理器中用作通过存储变量分享的存储器或局部存储单元时,其通信带宽几乎是双倍的)。双端口静态 RAM 可很方便地构成全局数据结构,使软件设计者可以在多处理器系统的工作转换中消除因过多的局部

变量引起的分支和多路存取操作。对于两个 DSP 处理器同时对同一单元写入而形成的冲突和竞争，双端口存储器提供了一个仲裁逻辑控制。双端口 RAM 的两个端口都有各自独立读/写、片允许和$\overline{BUSY}$控制引脚。$\overline{BUSY}$信号表示端口所要存取的单元当前正被另一端口存取。

双端口存储器可以接受两个 DSP 处理器来的片选信号$\overline{CS}$。通过左右$\overline{BUSYL}$、$\overline{BUSYR}$信号，只允许其中一个处理器进行存取操作。其$\overline{BUSY}$信号启动等待状态发生器(WSG)。WSG 的输出控制 TMS320C25 处理器 READY 引脚。当该引脚输入是高电平时，处理器对在选的存储单元进行读或写操作；当输入是低电平时，存取操作被延时至下一个机器周期，即插入一个等待状态。双端口存取操作所需产生的等待状态数量取决于双端口存储器的$\overline{BUSY}$信号。

3.等待状态发生器

低速存储器与钟频 40MHz 的 TMS320C25 处理器接口时，需插入等待状态发生器。在 STRB 信号有效(低电平)后 5ns 内，READY 引脚必须达到规定状态。若 READY 脚是高电平，存储器或外设的存取操作在当前机器周期内完成。若 READY 脚是低电平，存取操作延时至下一个操作周期，即插入一个等待状态。所需插入的等待状态数量，取决于各存储器的存取时间 t_a。若 $t_a < 40ns$，则不需等待状态；若 $40ns < t_a < 140ns$，则须插入一个等待状态。一般而言，需要的等待状态数与存取时间 ta 的关系为：

$$[100(N-1)+40]ns < t_a \leqslant [100N+40]ns \qquad (6-16)$$

等待状态发生器的电路如图 6－12。

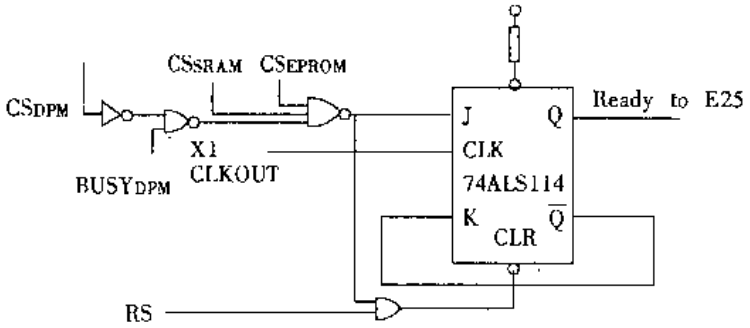


图 6－12　等待状态发生器电路

该发生器仅用于接口入双端口存储器。所需插入的等待状态数取决于双端口存储器的$\overline{BUSY}$信号。图 6－13 表示了 WSG 的时序。当某个 TMS320C25 处理器要从双端口存储器读取数据时，它在地址总线上提供一个指定地址。地址译码器对该地址译码，并在延时 t1 后，向双端口存储器发出片选信号$\overline{CS}$。若第二个 DSP 处理器正访问该存储器，则双端口 RAM 在延时 t2 后发出$\overline{BUSY}$信号。$\overline{BUSY}$信号启动 WSG，把第一处理器的 READY 引脚拉至低电平。READY 信号向高电平的转变发生在时钟 2 信号的下降沿，此时来自双端口存贮器的确定数据保持在数据总线上(D0—D15)，并在时钟 2 信号的上升沿被第一个 TMS320C25 处理器读出。指定地址的传送是在时钟 1 信号的下降沿进行的。写操作也是在时钟 2 和 BUSY 信号协调作用下进行的。

各 DSP 处理器的外部数据存储器(SRAM)和程序存储器可直接接口。当 TMS320C25 对它们进行存取操作时,$\overline{READY}$引脚置为高电平,可无需等待状态直接进行操作。

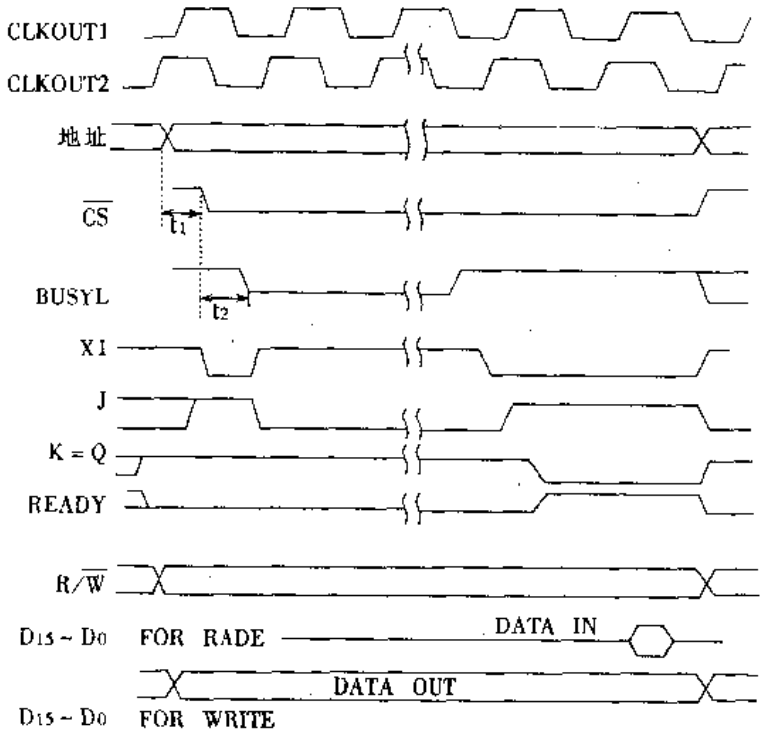


图 6 - 13 等待状态发生器时序

4.地址译码器

存储器分配图如图 6 - 14。这种分配方法设置的地址可以使两个处理器对共用的全局存储器具有相同的地址。地址译码器可向局部 EPROM、局部 SRAM 和全局双端口 SRAM 发送片选信号。从指定地址到片选信号发生时的传输延迟时间是线路设计时所需着重考虑的一个因素。地址译码器一般选用 74AS138,它的传输时延为 10ns(最大)。

5.模拟接口

每个 TMS320C25 处理器都各与一个模拟接口芯片 TLC32040 连接,如图 6 - 15 所示。TLC32040 芯片内集成有一个带通开关电容滤波器(防混淆输入)、一个 14 位分辨率的 A/D 转换器和一个低通开关电容输出平滑滤波器。之所以选用该芯片,是因为它有 14 位 A/D 和 D/A,接口简单,无需别的门电路,造价较低。而选用 14 位并行 A/D 接口则还要使用片选逻辑、三态门电路等。所需元件数多,造价较高。

TLC32040 中 A/D 转换器的数字输出可直接串行连接到 TMS320C25 的串行接收寄存器的输入。TMS320C25 的数字输出以位串行格式送到 TLC32040 内的 D/A 转换器。使用 TLC32040 的 SHIFT CLK 信号,传输和接受同步进行。当然,也可使用无帧同步脉冲的连续操作方式,这样可不用每 8 位或 16 位一个帧同步脉冲,而进行连续位流式传送和接收。用 RFSM 指令可把 FSM 位置 0。此时 FSX 和 FSR 引脚对 TMS320C25 的输入不起作用。传送在每个 CLK X 周期进行,接受则在每个 CLK R 周期进行。

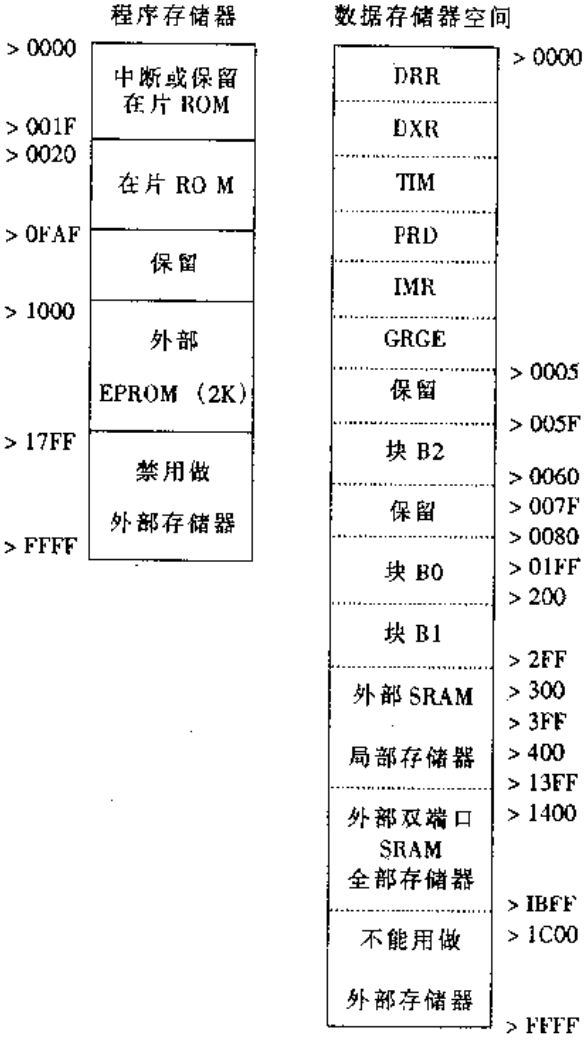


图 6－14　存储器分配

这种结构有很多优点：

· 公用存储器用于两个处理器间通信。

· 高速双端口 RAM 使设计工作简化。

· 两个处理器都可对双端口公用存储器进行存取操作，无需外部控制。

· 处理器间通信无需特别的协议。

· 公用存储器容量相当大，足够最大的程序使用。

但这种结构也有缺点：

· 可分享的信息受双端口公用存储器大小的限制。

· 当处理器 1 在公用存储器中装入处理器 2 需用的信息时，必须通知处理器 2 有新的信息可使用。这可通过一个软件查询的中断来完成。

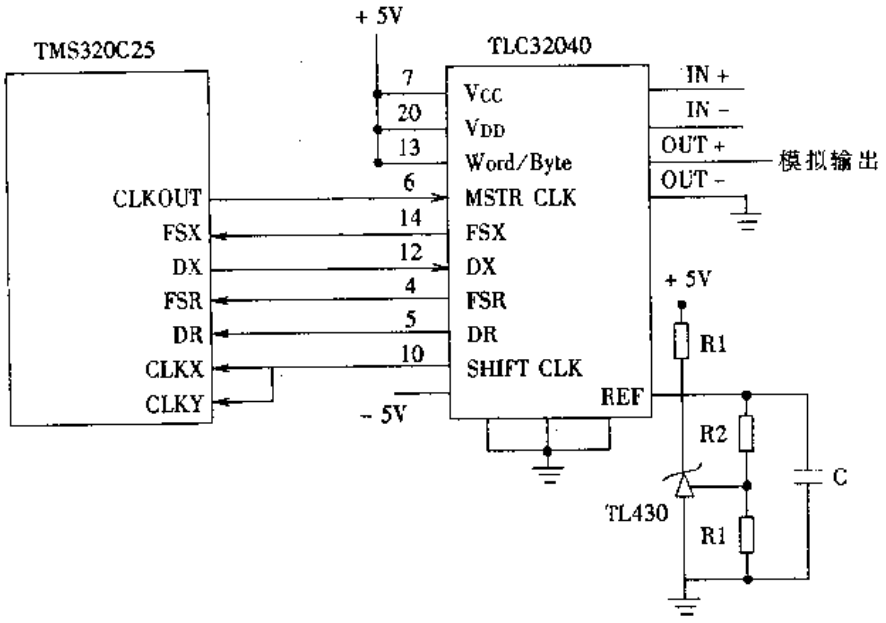


图 6 - 15　AIC TLC32040 和 TMS320C25 接口

三、自相关和互相关函数的计算

下面说明用多处理器系统进行自相关和互相关函数的计算。若信号 a(t)和 b(t)以 △t 秒进行周期采样并延时 m△t,则式 6 - 15 所示的互相关函数可表示为:

$$R_{ab}(m\triangle t) = (1/N)\sum_{k=m}^{N+m-1} a(k\triangle t) * b(k\triangle t - m\triangle t) \qquad (6-17)$$

式中 N 是样本数量,m = 0,1,…,99。也就是对每一个 m 值,把 N 个乘积值求和后再除以 N。同样,式(6 - 14)的自相关函数可表示为:

$$R_{aa}(m\triangle t) = (1/N)\sum_{k=m}^{N+m-1} a(k\triangle t) * a(k\triangle t - m\triangle t) \qquad (6-18)$$

对模拟信号的自相关和互相关函数值,在双 DSP 处理器系统中计算时间延时轴上的 100 个等距离点。输入波形 a(t)和 b(t)如图 6 - 16 中所示那样进行采样。每个处理器以 2△t 周期采样,并把采样值存于各自的 RAM 及双端口 RAM 中。处理器 1 在奇数△t 周期内采样,处理器 2 在偶数周期采样。各处理器以周期 2△t 各自完成相关函数的中间运算。每进行一次新的采样,处理器就把结果存入双端口存储器及其各自的存储器中,同时传送另一处理器存入双端口存储器的最后一个采样值。在运算过程中,a(t)的每个最新采样值都与 b(t)的 100 个采样值相乘,并与以前的积加值取和,这样完成的 100 个 R_{ab} 中间结果存于处理器各自的存储器中。因此,每个处理器需要 100 个局部存储器单元保存 $R_{ab}(m\triangle t)$的中间结果。当 b(t)和 a(t)有新值读入时,用 DMOV 指令传送 b(t)的原值,以使存储的采样值变成当前计算所需的延时采样值。在处理器各自完成 N/2 个样本的处理后,处理器 2 把它的 $R_{ab}(m\triangle t)$结果传送到双端口

存贮器中。处理器 1 读出这些结果，把它们与自己存储器中的对应值相加，生成 R_{ab}输出结果。

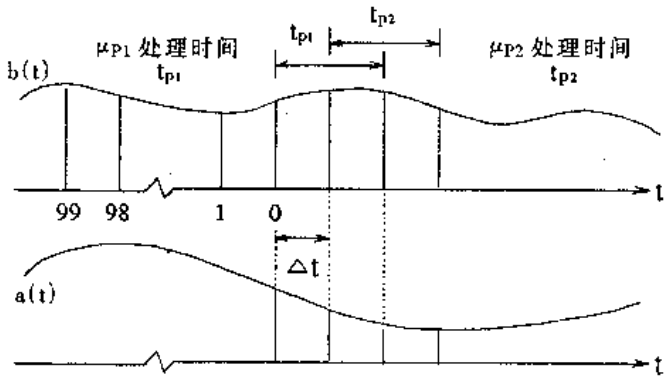


图 6－16　a(t)，b(t)采样

1．软件设计

自相关函数运算的程序流程图如图 6－17。

处理器 1、处理器 2 进行初始化后，处理器 1 采集 a(t)的 100 个样本值(周期△t)，并存入双端口存储器及自存储器。接着，它通过 XF 引脚向处理器 2 发出一个同步信号。当同步信号到达 BIO 输入端时处理器 2 开始操作，它从全局双端口存储器中读出 100 个 b(t)值。处理器 1 对偶数△t 周期采样，处理器 2 对奇数周期采样，其采样周期为 2△t。这一周期是内部计时中断得到的。在上电或复位操作时，处理器 1 和处理器 2 运行初始化程序。

存储器 B0 块 201H 到 264H 的 100 个单元存储 100 个输入样本值。201H 单元存储 A0，264H 单元存储 A99 或最新样本值。A0 是 A99 前 99 个△t 周期的采样值。内部存储单元 300H 至 363H 在 B1 块。双端口存储器的 1400H 至 1463H 单元存储最初的 100 个样本值。在后面的程序中，处理器 1 用 1400H 单元存储最新输入样本值。在所有的运算完成后，处理器 2 在 1400H 至 1463H 单元中存储中间结果。

处理器 1 的操作步骤如下：

(1)禁止中断。

(2)XF 引脚及 XF 的状态位 ST 寄存器置 1。处理器 2 此后一直处于等待状态直至处理器 1 的 XF 为 0。

(3)B0 块置为数据存储器。

(4)产生 2△t 周期中断信号的循环计数寄存器所需的值存入存储器。禁止计数器中断。

(5)串行口寄存器初始化(因为 A/D 需与串行口连接)。

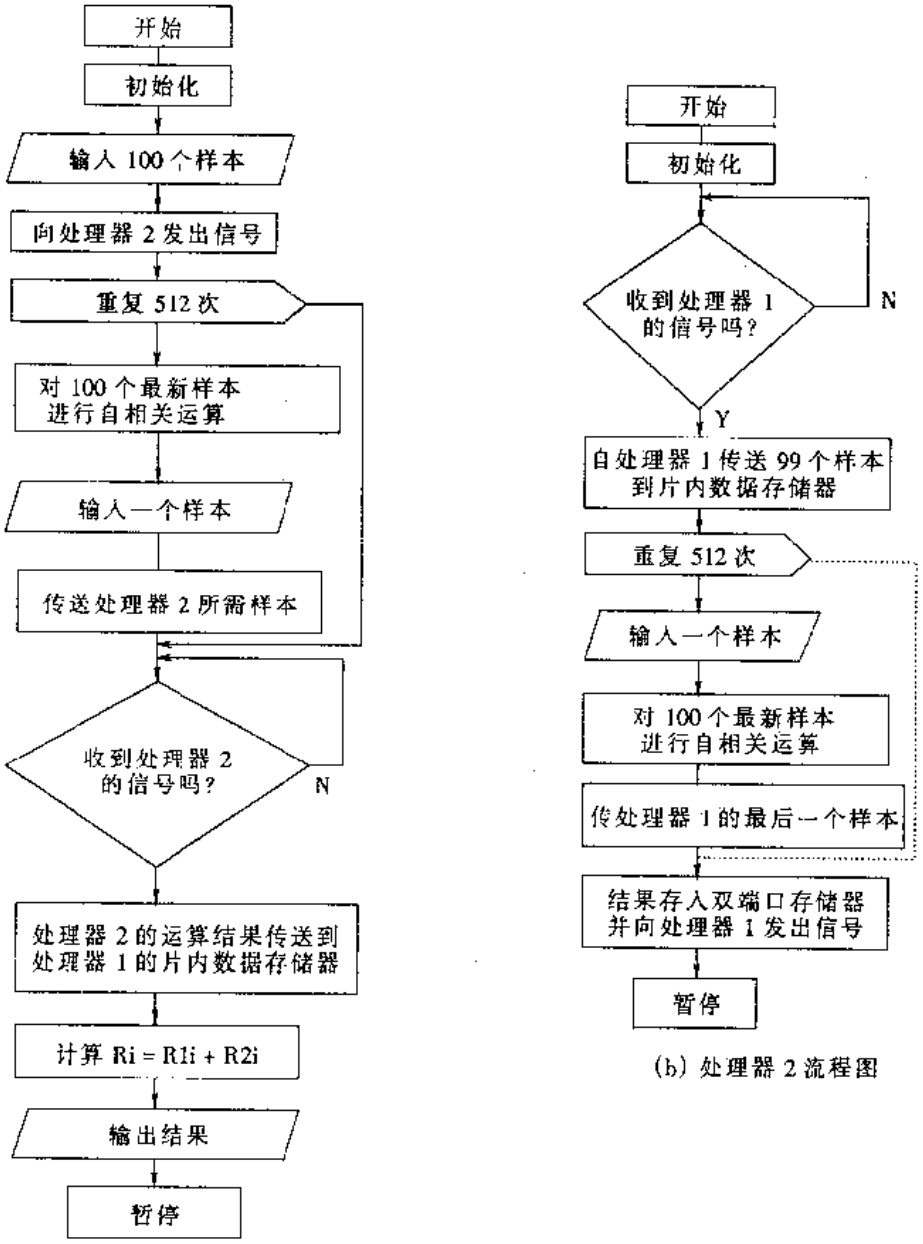
(6)B1 块中的 100 个单元(300H 至 363H)清零，用于存储自相关结果 R0 到 R99。

(7)允许计数器中断。输入周期为△t 的 100 个采样值，并将其存入双端口存储器的 1400H 至 1463H 单元。采样周期△t 用计数器中断得到。

(8)处理器 1 把 XF 引脚复位成 0，即向处理器 2 发出开始操作信号。

(9)处理器 1 等待处理器 2 送到 BIO 引脚的证实信号。收到证实信号时把 XF 引脚置 1。

(10)把前面存入双端口存储器 1400H 至 1463H 单元的输入值传送到片内 RAM 的第 4 页 B0 块的 264H 至 201H 单元。注意：264H 单元保存 A99，201H 单元保存 A0。



(a) 处理器 1 流程图

(b) 处理器 2 流程图

图 6-17 程序流程图

(11)处理器对下式进行 512 次(或 N 次)计算,i = 0 到 511:

$$R0 = R'0 + A1 * a(i)$$

$$R1 = R'1 + A1 * a(i)$$

$$R99 = R'99 + A99 * a(i)$$

式中 a(i)是处理器读入的最新输入样本值。R'表示前一个值。两个采样值间有 $2\triangle t$ 的延时。每隔 $2\triangle t$ 秒,计数器发出中断信号,执行中断服务程序(ISR)。在中断服务程序中,处理器 1 读出输入采样值并把它存入双端口存储器的 1400H 单元。在 $\triangle t$ 秒之前,处理器 2 已在 1401H 单元中存入了一个新的输入样本值。这两个最新样本值被传送到片内 RAM B0 块的

201H 和 202H 单元。在得到新值前，201H 至 264H 单元中的 100 个旧值已用指令 LTS 和 DMOV 移动两次。

(12)完成 512 次运算后，把 300H 至 363H 单元中的 100 个结果除以 512。

(13)收到来自处理器 2 的$\overline{BIO}$信号后，处理器 1 从双端口存储器读出结果 A0 至 A99，和它自己的结果值相加，生成最后的 100 个结果值，并把它们以 $\triangle t$ 周期输出到 D/A 转换器。

除等待处理器 1 来的开始信号和把最后结果存入双端口存储器，处理器 2 的其余操作与处理器 1 相同。这种双处理器系统采用常规布局的印刷线路板。图 6-18 是一个正弦波输入的自相关输出波形。可用一个在双端口存储器中有固定 a(t)和 b(t)值的模拟程序来测试软件。XDS/22 扩展软件开发系统可用来测试硬件。可用作为终端的 IBM PC CLONE 把软盘中的汇编程序送入 XDS/22。XDS/22 可象 TMS320C25 一样执行命令、指令、驱动目标程序。在跟踪方式下存入选择条件数后，XDS/22 能按要求停机，为后面的显示存入程序执行信息。XDS/22 在选择的条件下停机即形成断点。跟踪方式的断点特性可用来检查程序执行情况。

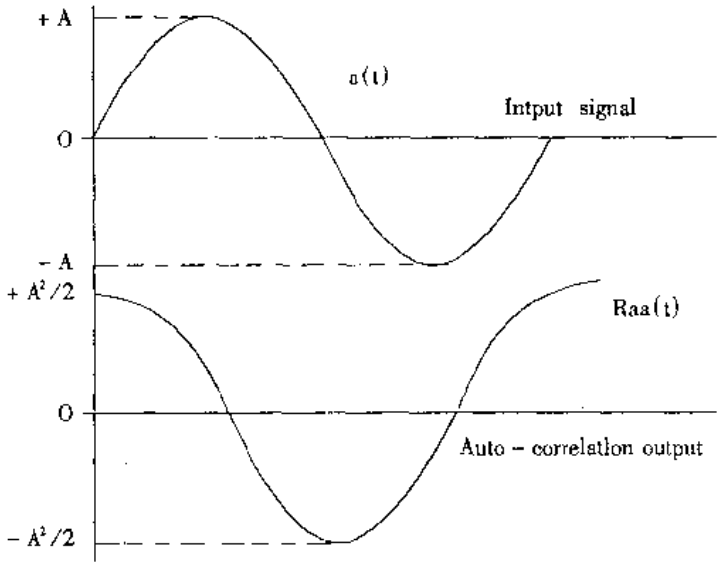


图 6-18 输出图形

2.特性

这种双处理器系统进行的实时相关运算，信号采样的最小周期是 40μs。这一限制是因为该算法中每个处理器的采样、运算时间为 80μs。速度指标 S 定义为一个处理器的全部运算时间和 P 个处理器全部运行时间之比。该双处理器系统进行相关运算的速度指标为 1.998(趋近 2)。

6.5 自适应滤波器及用 TMS32020 实现

一、自适应滤波器原理

K 阶横向滤波器构成方框图如图 6-19 所示。x(n)为输入信号，y(n)为输出信号，并表示为：

$$y(n)=\sum_{k=1}^{k} w_k\, x(n-k+1) \qquad (6-19)$$

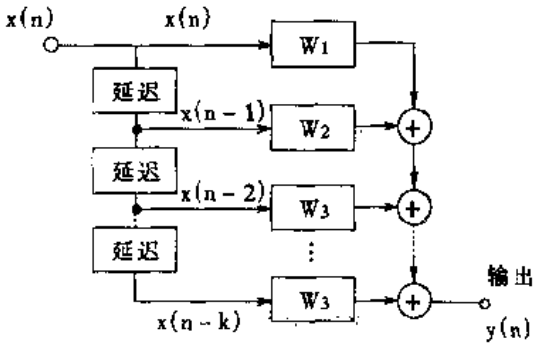


图 6-19 横向滤波器结构

式中，$w_k(k=0,1,2,\cdots,K)$为第 k 个滤波器系数，$x(n-k+1)$ $(k=1,2,\cdots,k)$表示输入信号。

二、最佳滤波器设计

作为滤波器的设计方法之一，如图 6-20 所示的那样，对于输入数据，预先确定一个所希望的理想输出 d(n)，然后以输出尽可能与 d(n)相近为目标，设计滤波器的各个系数。即以图 6-20 中误差 e(n)为最小来设计滤波器系数。这种方法设计的滤波器，称为最佳滤波器。

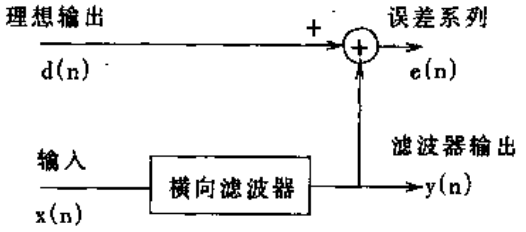


图 6-20 滤波器设计方法

例如，在预测信号 x(k) (k=1,2,3,……)的未来值时，如果预测得 m 个采样点的值，则理想输出 d(n)可为 x(n+m)。图 6-21 为从信号预测开始，按这种方法实现的信号处理例。

设在时点 n 的 K 个输入信号为 $\mathbf{x}_n$，滤波器系数矩阵为 W：

$$\mathbf{x}_n=\begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ \vdots \\ x(n-k) \end{bmatrix} \qquad \mathbf{W}=\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ \vdots \\ w_k \end{bmatrix} \tag{6-20}$$

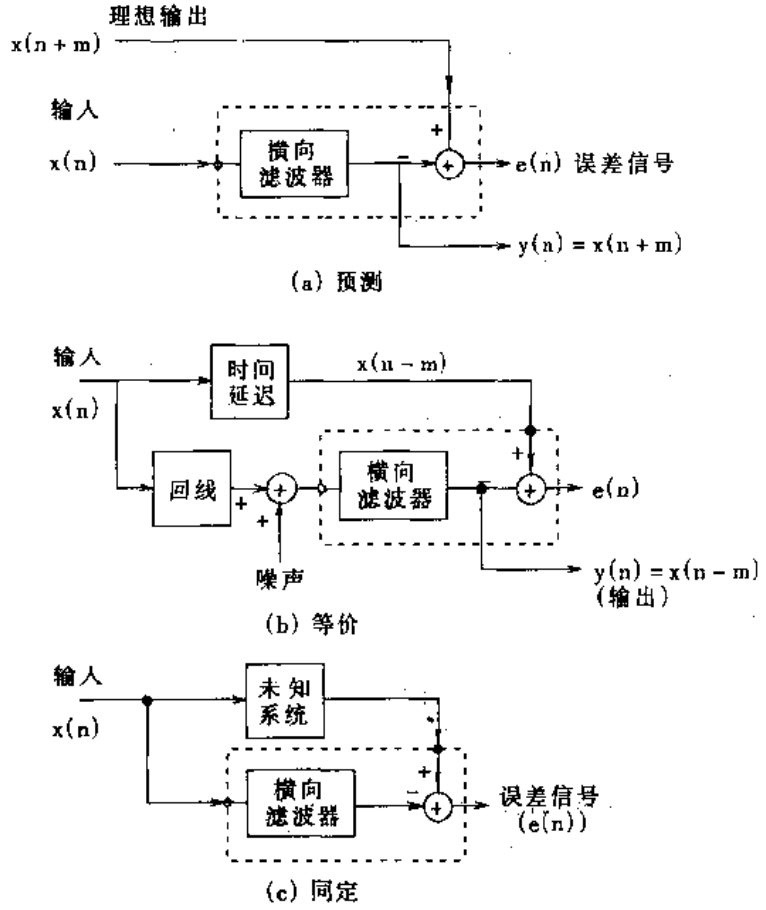


图 6-21　最佳滤波器在信号处理中应用

则式 6-19 用矩阵可描述为：

$$y[n] = w^{T}x_n \tag{6-21}$$

式中，W^T 为 W 的转置矩阵。这时，理想输出 d(n)和 y(n)之差，即 e(n)的平方误差 $E(e^2(n)]$最小时，最佳滤波器的系数 W_k 为

$$\mathbf{w}^* = \mathbf{R}^{-1}\mathbf{P} \tag{6-22}$$

式中，R 为输入信号的相关矩阵，P 为理想输出和输入信号的相关矢量。即：

$$R = E[\mathbf{x}_n\ \mathbf{x}_n^T],\ P = E[d(n)\cdot\mathbf{x}_n]$$

即，R^{-1}是 R 的逆阵，$\mathbf{x}_n^T$ 是矢量 $\mathbf{x}_n$ 的转置。这时，最小平均平方误差为：

$$E[e^2(n)] = E[d^2(n)] - P^{-1}R^{-1}P \qquad (6-23)$$

三、自适应滤波器

在最佳滤波器的设计中，必须以相关函数的形式明确给出信号的性质。然而当信号性质随时间变化时，则缺少相应的柔性。解决这一问题的滤波器是自适应滤波器。

当将输入信号和理想输出信号输入到自适应滤波器后，自适应滤波器自动地进行系数学习，则可得到理想输出和输出之间误差最小的最佳的滤波器系数。一旦学习结束之后，则可自动适应信号性质的变化。

滤波器系数的学习算法——LMS 算法说明如下：

设在时刻 n 时滤波器系数取：

$$w_k(n) \qquad (k = 1,2,3,\cdots\cdots,k)$$

则系数可按式

$$w(n+1) = w(n) + 2\mu e(n) \cdot x_(n) \qquad (6-24)$$

求得。系数 μ 可根据系数稳定性和系数的收敛性来确定：μ 小时，稳定性好但收敛性差，μ 大时收敛性好但稳定性差、输出容易发散。根据经验，取各抽头输入信号总功率值的 10 分之一到百分之一即可。

图 6－22 为自适应滤波器在回波消除器中的应用。在电话线路中，在混合呼叫的 4 线 2 线转换部分因有串音，发话者的声音做为回波而返回，这在长距离线路尤为明显。采用图 6－22 的线路可以消除回波。

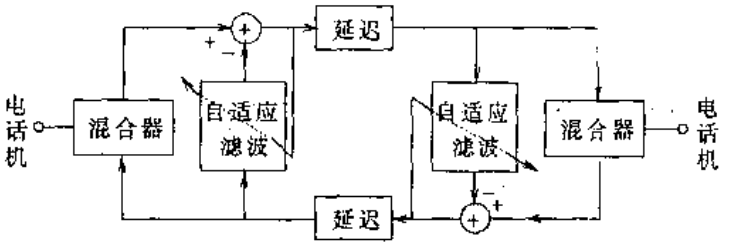


图 6－22　回波消除器示意图

四、用 DSP 芯片实现自适应滤波器

使用 DSP 芯片，可以很容易地实现实时数据处理的自适应滤波器。

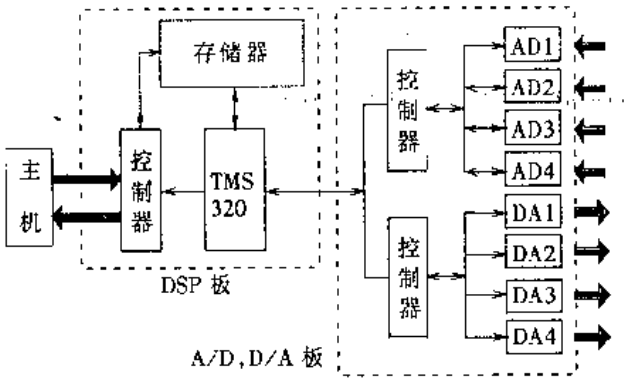


图 6-23　评价系统方框图

开发系统使用系统工程公司的评价开发系统。系统构成如图 6-23 所示。该系统是由 DSP 插件板、模拟输入、输出板和主机(PC-9801)及软件组成。DSP 板插入 PC-9801 的扩展槽内,通过 PC-9801 来的控制信号实现数据的传送。

输入输出板用两根 50 线电缆与 DSP 插件板相连,实装 A/D 为四个通道,D/A 用一个通道,A/D、D/A 转换器与 DSP 的 I/O 口相对应。

程序开发可在 PC-9801 的 MS-DOS 环境下进行,为此,备有 TMS32020 专用的汇编器和链接器。首先作成汇编源程序,然后通过汇编、连接,将生成的装入文件用专用的装入器传送给 DSP 插件板上的存储器,由 PC-9801 发出 DSP 板开始执行的命令。

为使主机(PC-9801)能对 DSP 进行控制和进行数据传送,系统备有 C 语言函数库,可方便地将 DSP 板存储器的内容取入 C 语言中变量或数组,相反地,C 语言中变量和数组数据也可传送给 DSP 板面的存储器。

6.6　PC 和 DSP 之间的 DMA 通信

在设计微处理器系统时,系统结构是一个重要因素。常见的系统结构有:

· 独立 DSP 系统;用于便携式仪表。

· 多处理器并行系统;用于集中、重复数字计算系统。

· 多种处理器芯片混合的主从系统;用于智能仪表。

主从系统是广泛使用的一种结构,它可以根据任务性质把工作合理地分配给两个处理器。一般下位机 DSP(从机)完成数据采集和信号处理,主机 IBM PC 则负责人机交互和更高级的控制,如数据检索和存储、更新图形显示、对下位机的控制和通信等。

实时信号处理系统的数据吞吐量很高,因此系统带宽主要取于主从系统间的数据传送速度。下面介绍以 IBM PC 为主机、DSP TMS320C25 为从机的主从系统中的通信。

一、主从机间信方式

主从微处理器系统中常用的数据传送方式包括:双端口存储器方式、双存储体方式、I/O

端口存储器存取方式等。

1.双端口存储器(DPM)存取

图 6-24 所示为 PC 和 DSP 间双端口存储器通讯方式。这种方式中 DPM 的作用类似于一个双向"邮箱"。例如,当 PC 需向 DSP 传送数据时,它产生一个控制信号选通 DSP 的 HOLD,把 DSP 置入三态状态(如高阻状态)。这时 PC 就可以对"邮箱"进行存取操作而不会引起总线必须缓冲。这种通信方式可以提高菊花链接信号交换协议的速度。

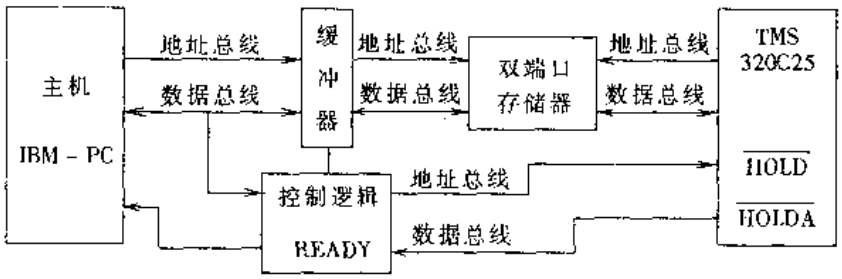


图 6-24 双端口存储器通信

TMS320C2X 系列处理器有一个专用在片电路进行全局数据存储器空间的分配,并使 PC 和 DSP 对双端口存储器能同步进行存取。

DPM 系统的主要缺点是同一时间内仅允许 DSP 或者 PC 对"邮箱"进行存取。另外,在 PC 对 DPM 进行存取期间内,DSP 需暂停运行。对实时系统而言,这将使系统带宽降低到不能使用的程度。

2.双存储体(DBM)存取

图 6-25 是双存储体存储器方式。在这种方式下,同一时间内每个处理器只唯一地对单个存储体进行存取。通过主处理器(PC)的控制,各存储体可在处理器间转接。PC 和 DSP 的地址/数据总线经缓冲三态开关连接到指定的存储体。通过地址/数据总线与存储体的转接,处理器可对相应的存储体进行存取。因处理器间没有直接的数据传送,所以系统带宽取决于存储体容量和地址/数据总线的转接时间。例如,处理器间通信是先把信息或数据存入当前存储体,再进行 DBM 转接操作,之后目标处理器就可以通过标准读/写操作完成数据存取。

DBM 通信方式可以最大限度地发挥每个处理器的作用,使空载时间减到最小。但这种方式增加了存储器,所以造价较高。

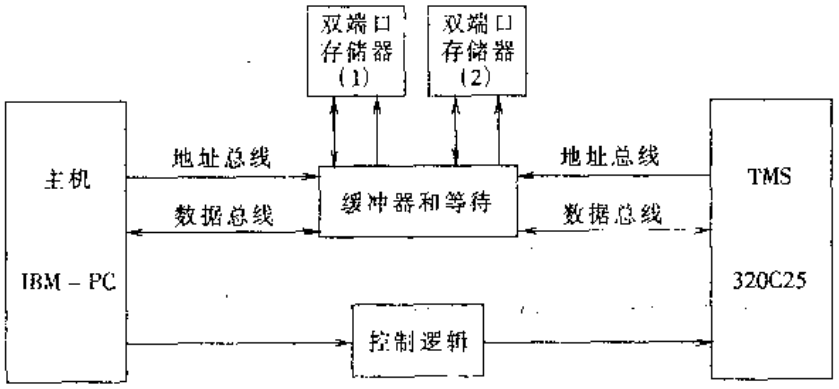


图 6-25 双存储体存储器通信

3.I/O 端口存储器存取

DPM、DBM 存储器存取方式都占用 PC 存储器空间，其主要缺点是降低了系统的扩展性能。例如，当主机增容或增加新的硬件时，所占用的存储器空间可能恰好是 DPM 或 DBM 当前占用的存储器地址，这就会引起冲突。这两种方式减少了主机存储器的空置部分，限制了硬件的扩展。因此，这就要求硬件设计能满足所用结构的各种变形。解决这个问题的一种方法是用 I/O 端口连接 PC 和 DSP。

图 6－26 是 I/O 端口数据传送方式，其接口选用并行口芯片 82C55 PPI。PPI 由三个特殊的 8 位 I/O 口构成。例如，PC 需传送一字节的数据时，它必须向 PPI 送出三条输出指令：高位地址字节、低位地址字节和数据。每个传送都必须按这一顺序，所以这种方式不适用于需大量传送数据的地方。

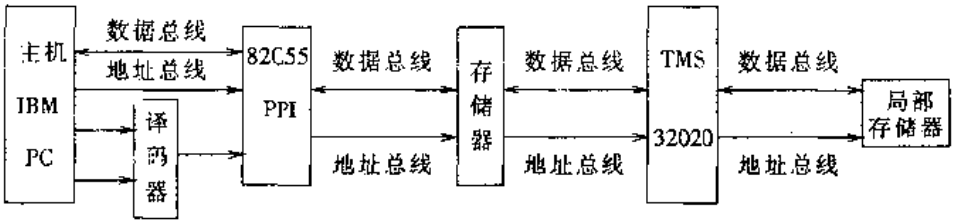


图 6－26　使用 82C25PPI 通信接口

图 6－27 是双端口存储器的另一种方式，PC 通过一个 I/O 端口对 DPM 进行存取，这样可以提高系统性能。因地址计数器的初始化只需一条输出指令，所以使用一个自动增/减地址发生器能提高数据块的传送速度。这个地址发生器可用二进制 up/down 双时钟计数器构成。

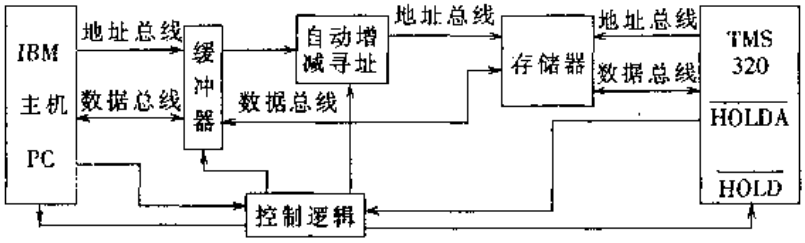


图 6－27　自动增/减寻址双端口存储器存取

二、PC－DSP 间 DMA 通信

上面介绍的 PC 与 DSP 间的常用接口方式中，数据传送速度取决于 PC 对存储器及 I/O 口的访问时间。为防止竞争，另一处理器须处于三态禁止状态，所以系统带宽仅由 PC 与 DSP 间数据传送的情况决定。

直接存储器存取方式可以提高处理器间数据传送的速度和效率。IBM－PC 对 DRAM 及硬盘的高速数据传送是用 Intel 8237A 可编程 DMA 控制器完成的。8237 是一个带优先权的四通道芯片，它在 PC 机上的用法如表 6－2。

表 6－2　DMA 通道和 IRM－PC 的应用

通道	次序	用　　法
0	0	存储器更新
1	1	用户可用
2	2	软盘
3	3	硬盘

DMA 通道 1 用于 PC 与 DSP 间数据传送。全部 DMA 请求信号(DRQ0－DRQ3)及应答信号(DACK0－DACK3)都在 62 引脚 PC 扩展槽上。用 DMA 传送数据时，考虑到存储器刷新时间，通道 1 的最大 DMA 传送速度可达 422KB，较之一般的 I/O 读/写及存储器读/写都快得多。

图 6－28 为用 DMA 接口方式构成的 PC 及 DSP 间的高速数据传送。其高速传送缓冲器是由双向先入先出存储器(FIFO)构成的。

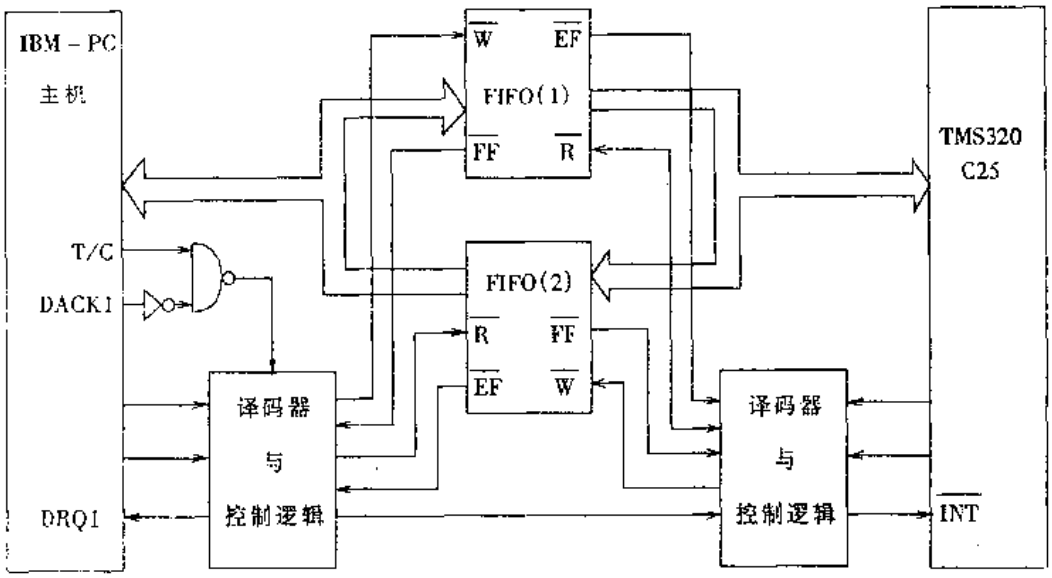


图 6－28　使用 DMA 的 DSP 和 PC 通信原理

例如，DSP 向 FIFO(1)传送数据时，$\overline{EF}$引脚有效，向 PC 发出请求信号(DRQ1)，FIFO 中存有数据。启动 DMA 传送数据直至 FIFO 中数据送完当 PC 向 DSP 传送数据时，它把数据存储在第二个 FIFO(2)中。$\overline{EF}$引脚变成有效，驱动 DSP 的 INT 引脚来启动中断服务服务程序接收 FIFO 中保存的数据。使用 FIFO 的一个优点是各处理器不需要同步。数据装入操作简单，控制电路选通相应的处理器并始读取数据。这种方法的灵活性及效率都相当高。

8237A 仅提供了一个终端计数信号(T/C)，因此，为识别来自 DMA 通道的 T/C 信号，增加了硬件通过 DACK1 信号及 T/C 信号来完成此项工作。

新一代的浮点 DSP，如 TMS320C30，AT&T DSP32C，Motorola DSP96002 等，都带有片内 DMA 控制器。这样它们可以允许 DSP 与高速传送缓冲器(如 FIFO)间进行双向 DMA 通信，大大增

强了数据吞吐能力。

三、DMA 软件开发

在 PC 与 DSP 通信前，必须先对 8237A 编程。这里先介绍一些 8237A 的 PCI/O 端口分配的知识，表 6－3 列出了用于 PC 和 AT 的 I/O 端口。

用 DMA 通道 1 进行 PC 和其他 I/O 芯片间的处理器间通信的编程流程如图 6－29。

下面介绍 DMA 通道初始化的主要编程步骤：

(1)基本地址　I/O 端口地址是 02H。注意基本地址计数器是 16 位的，所以需将其分成两个字节，依次送入同一 I/O 端口。先是高八位，然后是低八位。

(2)字节计数　I/O 端口地址是 03H，它也是 16 位的，所以其操作过程同上。

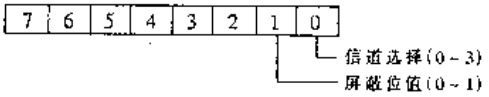
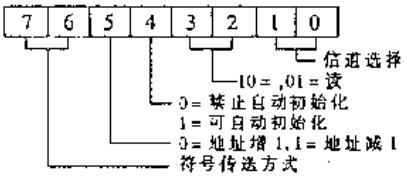
(3)页地址　I/O 端口地址是 83H，表 6－2 中未列出，这是因为系统板上包含有一个用于 3 个通道的 4 位页寄存器。因为所有存储器芯片刷新都是同步进行的，所以通道刷新无需页寄存器。若使用通道 2 或通道 3，则 I/O 口分别为 81H 和 82H。因此若数据传送是跨页的，则需送入两个通道。这即意味着页寄存器不能自动调节，必须对其复位。

(4)传送方式　方式寄存器是 8 位的，每位含义见表 6－3。B7 和 B6 位分别是 0、1 时，传送方式为单字节方式，这种方式允许 8088 和优先权较高的通道在传送期间对总线进行存取操作，其余各位见表 6－3。须记住的是读操作是把数据从存储器送入 I/O 芯片，而写操作是从 I/O 芯片到存储器。本文的 I/O 均为 FIFO。

(5)屏蔽寄存器　屏蔽寄存器也是 8 位的，但仅用低 4 位作为四个通道的屏蔽标志位，高四位未用。有三个 I/O 端口可对屏蔽寄存器进行存取(见表 6－3)。0H 口的每位输出都可清屏蔽标志，而 0FH 口输出的 0－3 位把屏蔽寄存器置位。不过最好是使用 0AH 口，因为该口一次仅写入一位屏蔽标志。例如，对 0AH 口写入 05H，则把通道 1 屏蔽标志置位，在使用 DMA 传送数据时，把屏蔽标志位清零，传送完成后屏蔽标志位置位。

表 6－3　8237A DMA 控制器 I/O 通道地址

PCI/O 地址	ATI/O 地址	寄存器内容说明
0	C0	信道 0 计数
1	C2	信道 0 基本地址
2	C4	信道 1 计数
3	C5	信道 1 基本地址
4	C8	信道 2 计数
5	CA	信道 2 基本地址
6	CC	信道 3 计数
7	CE	信道 3 基本地址
8	D0	指令/状态寄存器
9	D2	请求寄存器
A	D4	符号屏蔽寄存器

		
B	D6	方式寄存器(仅写) 
C	D8	清字指针触发器
D	DA	清主机
E	DC	清屏蔽寄存器
F	DE	写入全部屏蔽寄存器

(6)状态寄存器

I/O 口地址 08H。该寄存器标志各通道是处于未决请求还是达到终端计数(T/C)。其 0 - 3 位分别标志通道 0 - 3。图 6 - 29 中使用为实时处理软件设计的硬件来检测 T/C,例如状态检测时间可用于显示、打印、存储等操作。

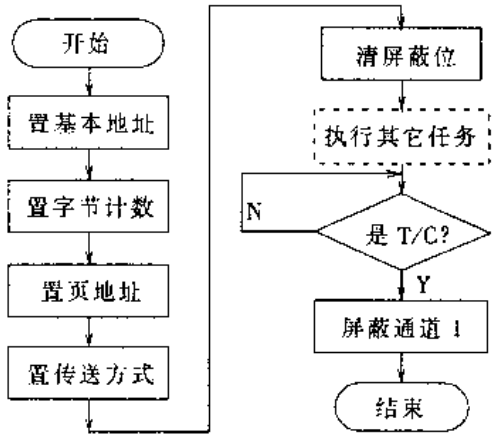


图 6 - 29　8237A 编程顺序

四、应用

上面介绍的 DSP 和 PC 间 DMA 通信方式已应用在血流动监控设备的测量与分析系统中。该系统 DSP 为 TMS320C25,包括一个从 1 次端仪表向 PC 传送测量数据的 DMA 通讯接口。这一接口非常重要,因为 DSP 系统需要访问主机 PC 的大量软件——为后续处理在磁盘中存储数据或在 PC 图形设备上显示处理结果。

图 6 - 30 是用 DMA 通信的医疗信号图形处理系统。该系统造价相对较低,图形复合处理

能力强，它是基于 TMS34010 图形处理芯片(GSP)和 TMS320C25(DSP)构成的。视频信号数字化转化器把照像图形处理后送入图形缓冲器。当前系统用 DMA 在图形缓冲器、DSP 及 PC 间传送数据。使用 PC 总线把图形数据送到 GSP 用于显示；这一操作最终也是用 DMA 进行的。图形缓冲器和 DSP 间有大量图形数据需要传送，以便完成图形处理运算及在 PC 磁盘上存储结果或在 GSP 上显示结果。若不计数据进出 GSP 时的缓冲，系统性能近似于实时处理。

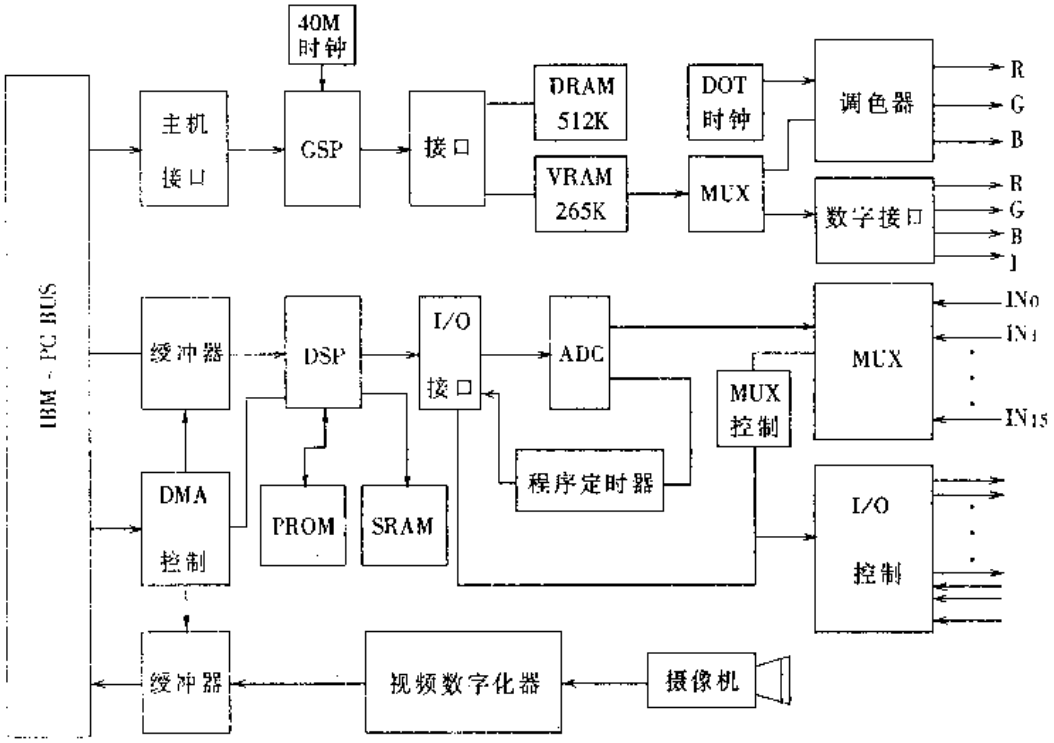


图 6－30 GSP 方框图，DSP 基本信号和图像处理系统

用 DMA 进行 DSP 与 PC 间通信有许多优点，如硬件结构简单、数据传送速度高、软件灵活等，特别适用于实时处理。主从 DSP 应用系统需要核心软件来支持 DSP 与 PC 的通信，该软件需存入 ROM。这种系统的 DSP 软件在执行前需转送到 DSP 存储器，这是与一般主从 DSP 系统不同的。若数据只需从 DSP 送入 PC(图 6－28)，则接口可以相应简化。

6.7 DSP56000/DSP56001 PID 控制系统

模拟实时控制系统有很多优点，但也存在诸如温度系数、元器件老化、电网电压波动及产品难以批量化等问题，此处，模拟调节器和模拟滤波器对低频扰动不敏感，当参数需要改变时，改变模拟调节器参数值也不方便。

使用微处理器、微调节器或 DSP 控制系统可以解决这些问题。数字调节器不存在元件老化或环境温度影响问题，利用软件可进行参数修改。使用单片 DSP，不用增加更多的硬件结构，借助于软件可实现许多复杂的控制系统，其控制算法精度很高。

一、数字调节器

使用 DSP56000/DSP56001 可构成精确的数字调节器。为减少因寄存器有限长而引起的失调噪声，在结构上多采用一阶节和二阶节的级联或并联结构。图 6－31 为采用级联和并联实现 6 阶调节器的示意图。

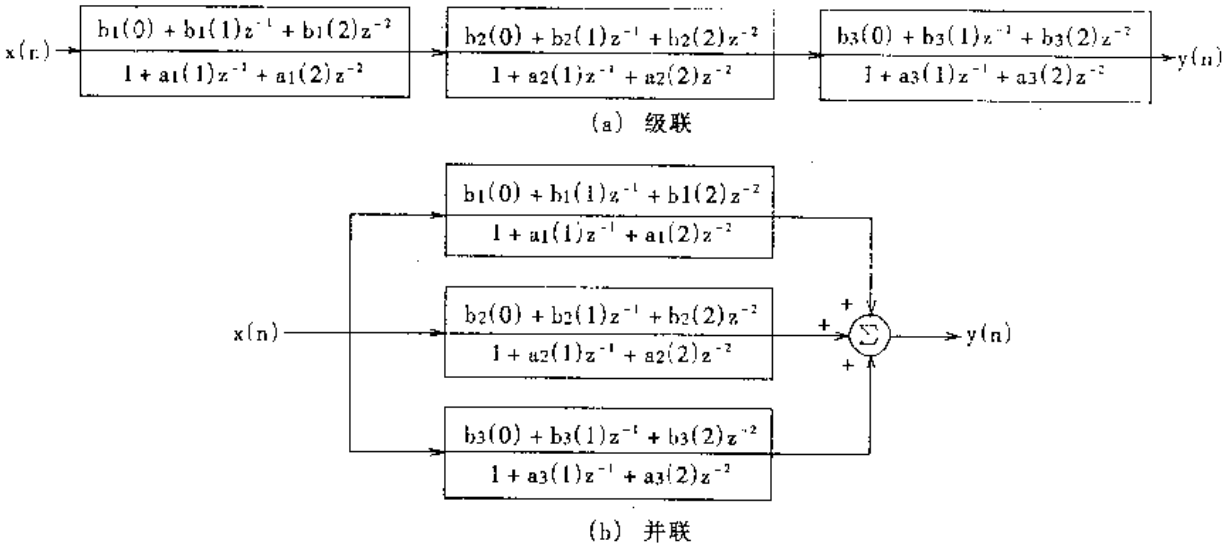


图 6－31　调节器级联和并联方式实现

并联方式适宜于多处理器结构，但这种结构对系统的量化噪声非常敏感。级联时，数字调节器的传递函数可写为：

$$G_c(z)=g\prod_{i=1}^{n}\frac{b_i(0)+b_i(1)Z^{-1}+b_i(2)Z^{-2}}{1+a_i(1)Z^{-1}+a_i(2)Z^{-2}} \qquad (6-25)$$

式中，g 为全系统增益，它包括在一个基本节内。图 6－32 为双二次型基本二阶节结构。

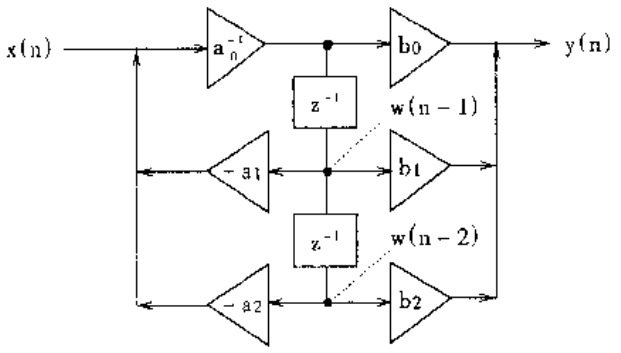


图 6－32　双二次基本二阶节

图 6－33 是由标准型级联构成的 6 阶数字调节器。它的第一级为全零点一阶节，最后一级为全极点一阶节，中间两级为双二次型二阶节。

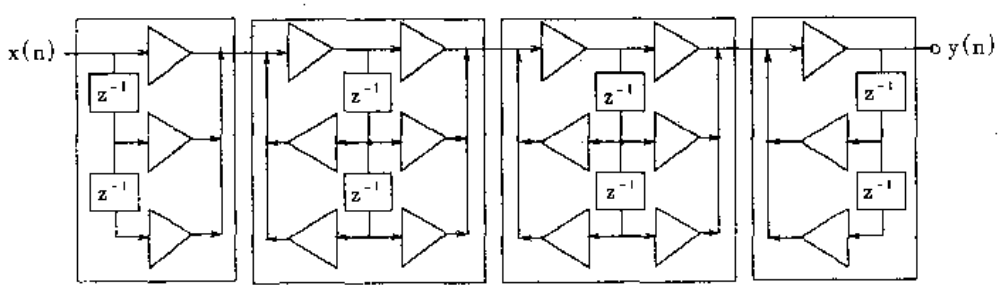


图 6-33 标准型级联 6 阶数字调节器

由于 DSP56000/DSP56001 所有算术运算、逻辑操作要求用小数表示数据，因此，二价节的系数 ai(k)、bi(k)均需表示成小数。如果分子 bi(k)中任一个或多个大于 1，那么，双二次型的所有分子系数均需除以 2 的幂直至所有的系数小于 1。这个 2 的幂即为输入的比例因子，并成为其他二阶节的分子参数或输出比例因子。

由 DSP56000/DSP56001 实现最简单的 PID 调节可使用直接标准型单元，例如，传递函数为：

$$G_c(z)=\left(\frac{0.6+0.4Z^{-1}-0.25Z^{-2}}{1-0.8Z^{-1}+0.6Z^{-2}}\right)\left(\frac{0.8+0.6Z^{-1}+0.2Z^{-2}}{1+0.4Z^{-1}+0.3Z^{-2}}\right)\times \left(\frac{0.8+0.64Z^{-1}+0.9Z^{-2}}{1+0.7Z^{-1}+0.5Z^{-2}}\right) \tag{6-26}$$

根据图 6-32 所示结构，先计算并存储中间值(即内部节点值)，记为 W(n-1)、W(n-2)，等等。递归方程式可写为：

$$W(n)=a_0^{-1}(X(n)-a(1)\times W(n-1)-a(2)\times W(N-2))$$
$$Y(n)=b_0\times W(n)+b(1)\times W(n-1)+b(2)\times W(n-2)$$
$$W(n-2)=W(n-1)$$

$$W(n-1)=W(n)$$

使用 DSP56000/DSP56001 汇编语言编写的实现 6 阶 PID 调节器的程序如下：

LIST1：6 阶 PID 调节器程序

```
            include'declare.dat'
A_D         equ     $FFFE           ;Location of A_D in Y memory
D_A         equ     $FFFF           ;Location of A_D in X memory
numsec      equ     3               ;Number of cascaded biquad sections
* * * * * * * * *
* X 存储单元 *
* * * * * * * * *
            org     X: $0
data        dc      0               ;Cascade Section 1 W(n-2)
```

```
                dc      0                ;Cascade Section 1 W(n-1)
                dc      0                ;Cascade Section 2 W(n-2)
    dc          0                        ;Cascade Section 2 W(n-1)
                dc      0                ;Cascade Section 3 W(n-2)
                dc      0                ;Cascade Section 3 W(n-1)
*********
*Y存储单元*
*********
                org     Y:$0
    coef
                dc      .6               ;a(2) - Cascade Section 1
                dc      .8               ;a(1) - Cascade Section 1
                dc      .25              ;b(2) - Cascade Section 1
                dc      .4               ;b(1) - Cascade Sertion 1
                dc      .6               ;b(0) - Cascade Section 1
                dc      .3               ;a(2) - Cascade Section 2
                de      .4               ;a(1) - Cascade Section 2
                de      .2               ;b(2) - Cascade Section 2
                de      .6               ;b(2) - Cascade Section 2
                dc      .4               ;b(0) - Cascade Section 2
                de      .5               ;a(2) - Cascade Section 3
                de      .7               ;a(1) - Cascade Section 3
                dc      .9               ;b(2) - Cascade Section 3
                de      .64              ;b(1) - Cascade Section 3
                dc      .8               ;b(0) - Cascade Section 3
*********
*快速中断服务程序*
*********
                org     p:reset          ;Reset service routine
                jmp     main
                org     p:irqa
                movep   Y:A_D,a
                nop
*初始化程序*
                org     p:main
                move    #data,r0         ;r0 points to states
                move    #5,m0,           ;r0 is modulo 6
                move    #coef,r4         ;r4points to filter coeffcients
                move    #14,m4           ;r4 is modulo 15
                move    #$007,x:ipr
```

```
                move        #2,x:ber                ;2 wait states for A _ D and D _ A
                move        x:(r0)+,x0y:(r4)+,y0;init data ALU registers
                andi        #$FC,mr                 ;enable all interrupts
* PID 算法 *
start
        wait
        do      # numsec,enddo
        mac     x0,y0,a     x:(r0),x1     y:(r4)+,y0     ;A x(k) a2 * s2
        macr    x1,y0,a     x1,x:(r0)+    y:(r4)+,y0     ;A x(k) a2 * s2 a1 * s1
        mpy     x0,y0,a     a,x:(r0)+     y:(r4)+,y0     ;A b2 * s2
        mac     x1,y0,a     x:(r0)+,x0    y:(r4)+,y0     ;A b2 * s2 + b1 * s1 +
enddo                                                    ;b0(x(n) a2 * s2 a1 * s1)
        movep   a,y:D           ;output result to D _ A
        jmp     start           ;repeat
```

在本程序中，进行并行 A/D 和 D/A 外部寻址，地址分别为 Y:$FFFE 和 Y:$FFFF。图 6-34 表示数据和系数在存储器中的分配图，每个二阶节的内部节点和系数存储在片 X、Y 数据 RAM。这种存储器分配方式允许连续地在 X 总线和 Y 总线上进行数据传送，以便在每次乘法时将数据置入 ALU 寄存器，用于进行下一次乘法。

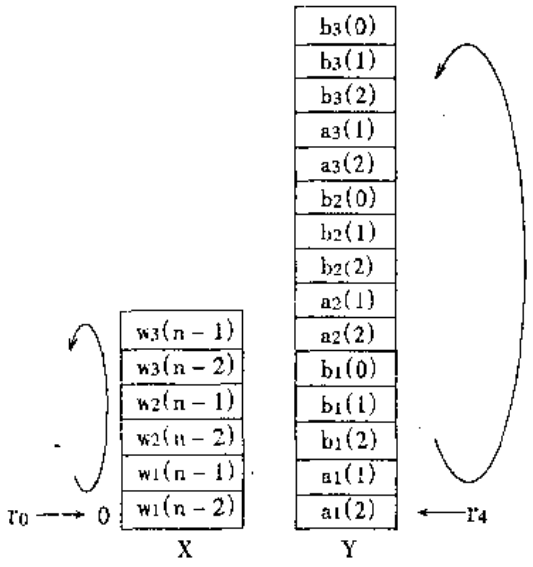


图 6-34　阶 PID 调节器内存分配

二、初始化

硬件复位时，在全操作方式下 DSP56000/DSP56001 开始执行程序存储器 0 单元的指令，但在标准扩展方式下，程序执行程序存储器 $E000 为始地址的指令。复位之后，DSP56000/DSP56001 立即处于主程序的起始地址。头 8 条指令是数字调节器初始化所必须的程序部分。

在汇编语言程序中，地址寄存器 r0 被初始化为指向 6 阶调节器内部节点表的指针，它以模 6 变址来建立一个内部节点表，其功能类似于循环缓冲器。同样，r4 初始化为指向系数表表首地址，以模 15 变址。使用模数寻址方式，可省略重新初始化地址指针的操作。然后是中断优先寄存器的初始化。它设置外部硬件中断 A 的优先级为 2，边缘触发方式。总线控制寄存器的编程组给 A/D、D/A 转换器提供 2 个等待状态。由于滤波器程序可存储在内部程序 RAM，而且仅能使用内部数据 RAM，所以 P、X、Y 存储器不需要外部等待状态。下一个指令取第一节内部节点 W1(n-2)和第一个系数 a1(2)。该指令的作用是为滤波操作初始化数据 ALU 寄存器和地址指针。最后，清除方式寄存器的低 2 位，打开所有中断，即设置优先级大于或等于 0。

三、PID 校正算法

在这个程序中，使用 WAIT 指令使 DSP 和 A/D 同步。假设用并行 A/D 转换结束脉冲触发 DSP 的外部中断 IRQA。通过等待指令，使用 WAIT 指令的优点是 DSP56000/DSP56001 在进行下一次采样之前，处于低功率状态。中断也能完成 A/D 和 DSP 之间的同步。应该注意的问题是使用 WAIT 指令时，至少需要 8 个指令周期完成快速中断的头一条指令。还可使用 SSI 做 I/O 口，用于串行 A/O 和 D/A(如 DSP56ADC16)的通信。

硬件 DO 循环可以实现级联二阶系统调节器或滤波器。DO 循环仅需三个指令周期，且无需执行周期。由于 DSP56000/DSP56001 的并行结构，仅需五个指令周期就可完成每个二阶节。中间结点值存入存储器前，先舍入为 24 位数。这种舍入是在精度要求内进行的。

为了保证调节器和滤波器的稳定性，系统所有复极点必须在 Z 平面单位圆内。二阶系统的分母为：

$$D(z) = 1 + a_1 Z^{-1} + a_2 Z^{-2} = (1 - P_1 Z^{-1})(1 - P_2 Z^{-1}) \qquad (6-27)$$

式中 a1 表示极点和的负值—(P1 + P2)；而 a2 是两极点乘积 P1·P2。如果极点是共扼极点，则 P2 = P1*。由于两个极点中最大的极点绝对值必须小于 1，可得出下式：

$$|a_2| = |P_1 \cdot P_2| < 1 \qquad (6-28)$$

而

$$|P_1 \cdot P_2| = \left|\frac{-a_1 \pm \sqrt{a_1^2 - 4a_2}}{2}\right| < 1 \qquad (6-29)$$

解方程(6-28)得到如下不等式：

$$|a_1| < 1 + a_2 \qquad (6-30)$$

式 6-28 和式 6-30 决定了图 6-35 所示稳定三角形。对于一个稳定的阶系统，a1 和 a2 的值必须在稳定三角形内。

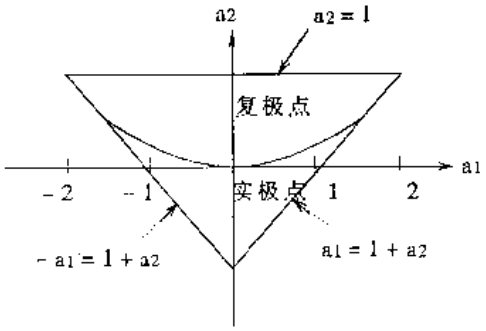


图 6-35 稳定三角形

由图 6-35 可见,a1 的值大于 1 时,系仍然是稳定的。由于 DSP56000/DSP56001 不能支持非小数系统,为了使这种类型的网络稳定,可以提出分母中的公因子。图 6-36 是修改后的这种系统的信号流程图。这时需要 6 个指令周期实现这种双二次型系统。

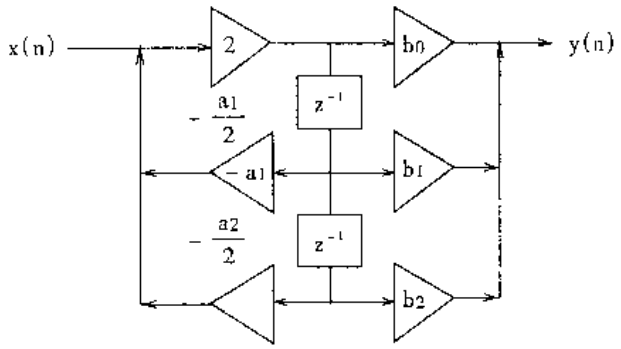


图 6-36 修改后的双二次阶节

当 bi(0)系数是 1,而 bi(1)、bi(2)和 ai(2)系数是小数时,DSP56000/56001 仅需要 4 个指令周期执行双二次型数字系统的操作。

硬件系统构成参见 4.3 节。

6.8 并行实时数字信号处理系统

用 TMS32010 可组成一个高速、通用的数字信号处理系统,并可设计成简单、紧凑、灵活可扩展的结构。在标准系统中,使用 4 个 TMS32010。主计算机通过一个简单的数据总线和共用存储器进行处理器间的数据传送和通信。这种系统可用于 FFT 处理机、数字滤波器、语音滤波器、语音识别及图像处理等。

一、并行处理机结构

标准形式的异步处理结构是一种高度并行的松散结合的多处理器系统。系统使用的四个处理器组件,由公共总线互相连接并排列成一种逻辑循环的结构。各处理器时序相同,处理周期都包含 N 个时隙,每个时隙为输入数据的帧间时间间隔。每一个处理器均依次占据一个新

的数据输入段然后开始处理。在每个周期的末端产生中间输出，因而，处理器用 N 个时间帧处理一个数据段。如果所需处理时间小于 NG 时隙，那么实时处理所能完成的最大数据输入速度将比单处理器系统提高 N 倍。由四个处理器组成的系统的时序如图 6－37 所示。

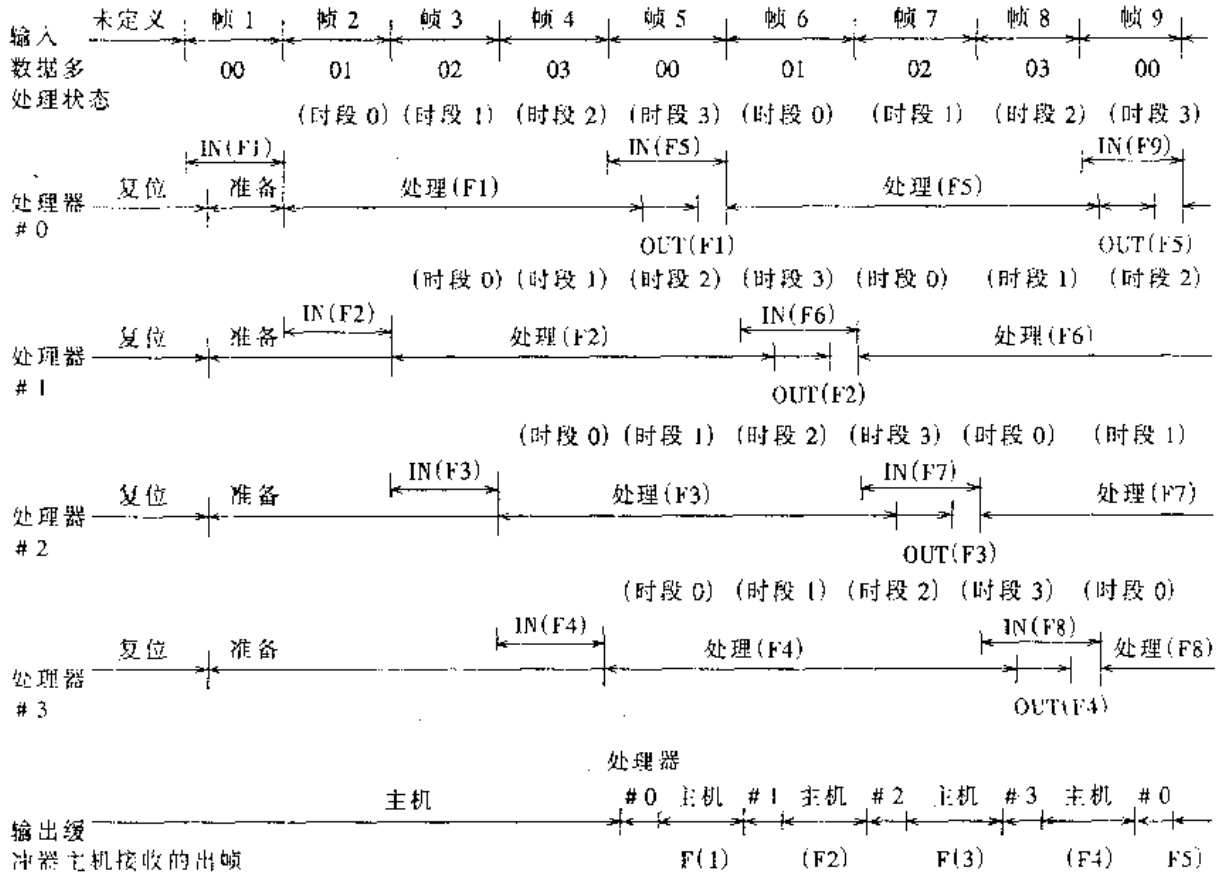


图 6－37　四处理器组件的 DSP 系统时时序图

二、硬件实现

实时信号处理系统由三个子单元(如图 6－38 组成：实时 DSP，包括母控制板和四个处理器板；模拟接口板(AIB)和 AIB 控制器，计算机和主机适配器。

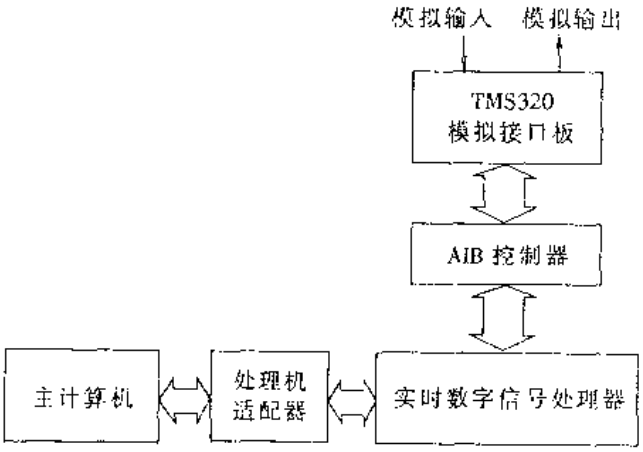


图 6－38　DSP 系统框图

1.实时信号处理器

实时信号处理器包含一个含四个相同处理器单元并可扩充的母控制板。处理系统和标准方框图分别如图 6－39 和图 6－40 所示。

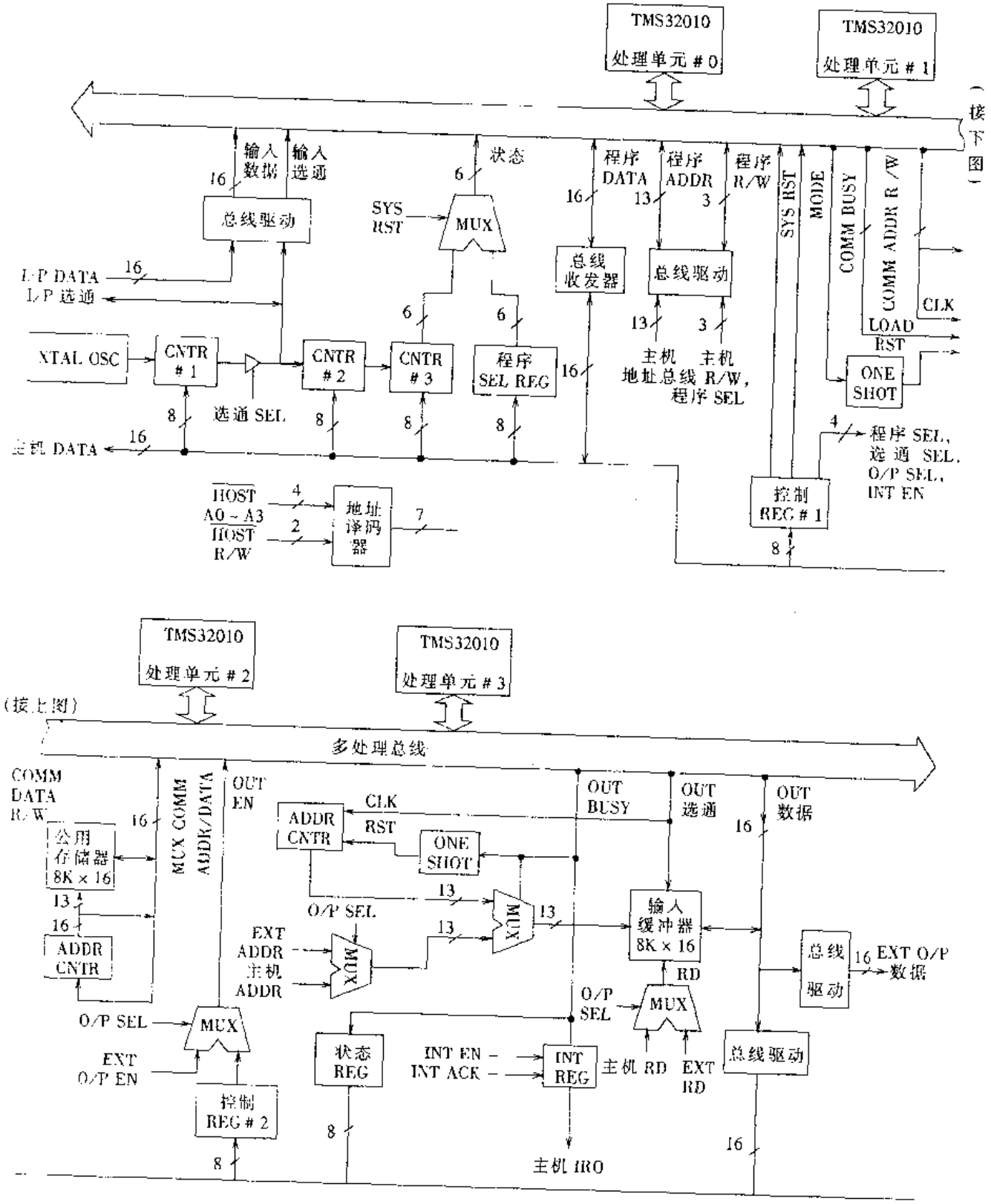


图 6－39 实时 DSP 系统方框图

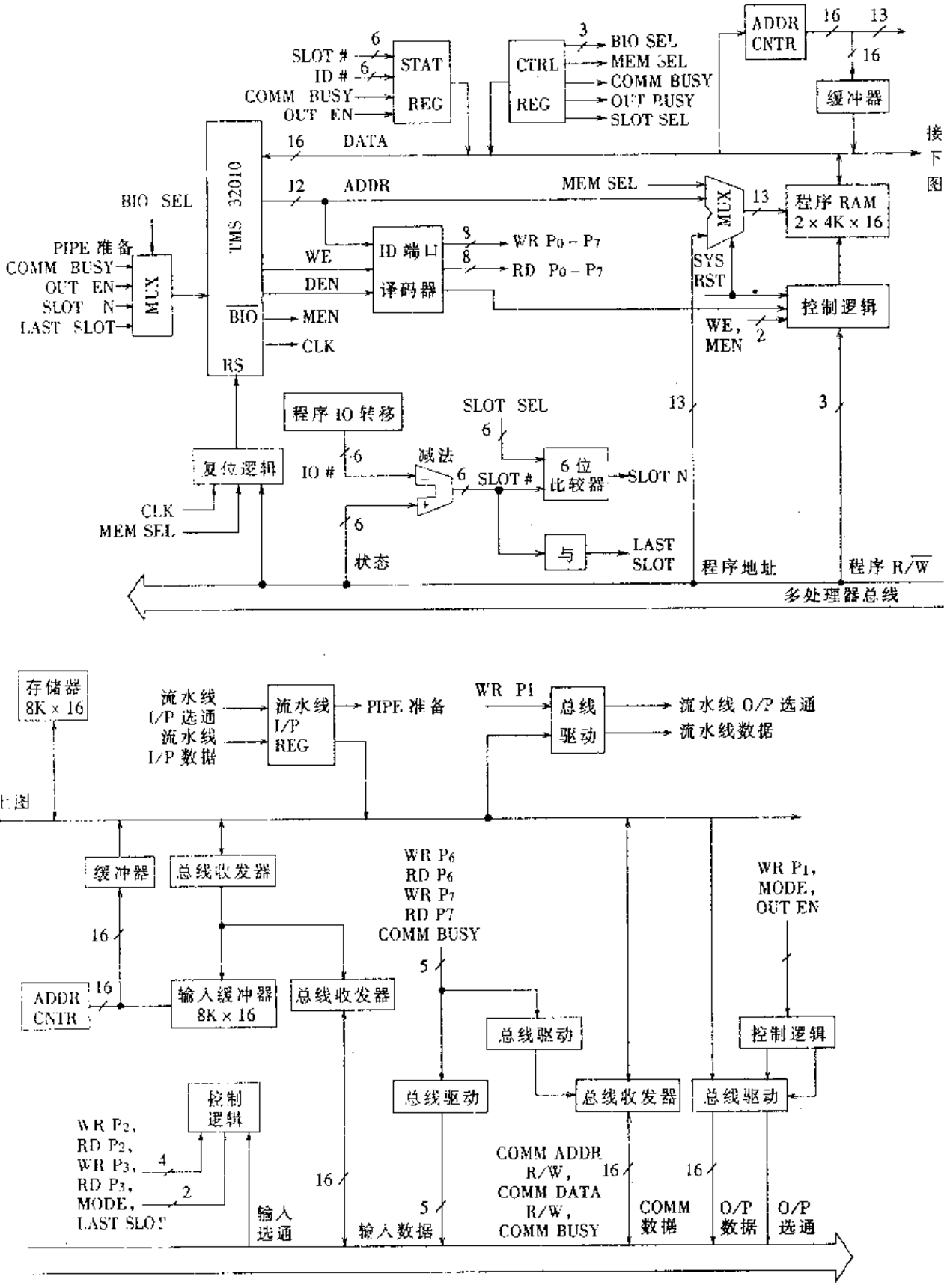

SLOT #
ID #
COMM BUSY
OUT EN
STAT REG
CTRL REG
BIO SEL
MEM SEL
COMM BUSY
OUT BUSY
SLOT SEL
ADDR CNTR
缓冲器
接下图
16 DATA
TMS 32010
J2 ADDR
BIO SEL
PIPE 准备
COMM BUSY
OUT EN
SLOT N
LAST SLOT
MUX
WE
DEN
ID 端口 译码器
WR P0 – P7
RD P0 – P7
MEM SEL
程序 RAM 2 × 4K × 16
SYS RST
控制逻辑
WE, MEN
BIO
MEN
CLK
RS
程序 IO 转移
复位逻辑
IO #
减法
SLOT #
6 位比较器
SLOT N
CLK
MEM SEL
状态
与
LAST SLOT
程序地址
程序 R/W
多处理器总线
存储器 8K × 16
流水线 I/P 选通
流水线 I/P 数据
流水线 I/P REG
PIPE 准备
WR P1
总线驱动
流水线 O/P 选通
流水线数据
接下图
缓冲器
总线收发器
WR P6
RD P6
WR P7
RD P7
COMM BUSY
WR P1,
MODE,
OUT EN
ADDR CNTR
输入缓冲器 8K × 16
总线收发器
总线驱动
控制逻辑
总线驱动
总线收发器
总线驱动
控制逻辑
WR P2,
RD P2,
WR P3,
RD P3,
MODE,
LAST SLOT
输入选通
输入数据
COMM ADDR R/W,
COMM DATA R/W,
COMM BUSY
COMM 数据
O/P 数据
O/P 选通

图 6 – 40 处理器板图

电路的主要部分是由公共多处理器总线连接的控制器板、处理器组件和模拟接口板组成。每个处理器组件的核心部件是一个时钟为 20MHz 的 TMS32010DSP，并有 4K×16 位的高速静态 RAM 为程序存储器。处理器板的方框图如图 6－40 所示。定序器装在母控制器板上，客观存在生成一个多处理器状态字控制所有处理单元，使其同步操作。这个状态字是一个周期序列：

$$0,1,\cdots,(N-1),0,1,\cdots,(N-1),0,1,\cdots$$

它将每个处理器组件的时隙数量定在一个处理周期内，N 是装入系统中的处理器组件数量。在原型设计中，N＝4，即四个器件，用状态字命名为 0，1，2，3。这个序列将根据输入数据帧自动地一位位前移。每个处理器有唯一的状态名，它们各自的时隙位置可以从多处理状态字中减去它的状态名后得到。例如，在状态 2 时，处理器 2 在间隔 0(第 1 个间隔)，处理器 1 在间隔 1，0 在间隔 2，3 在间隔 3(最后一个)。

实时 DSP 的 16 位数字化外部数据受输入选通信号的控制。用德克萨斯仪器公司的模拟接口板(AIB)将模拟输入信号转换为数字数据，这种接口板由带有防混淆输入滤波器和输出低通滤波器的速 ADC 和 DAC 组成。控制器对 AIB 初始化，从 AIB 中读取数据，并将数据送到各处理子系统。

每一组件都装有一个 8K16 位的局部双端口输入数据缓冲器。除最后一个时隙外，TMS32010 在任何时间都可对这个缓冲器进行存取操作。输入采样值通过多处理总线直接通知所有的处理器组件，但每块板只在它的最后时隙接收一帧数据。所以，一个整帧的新输入数据可在每个处理周期的开始时得到，用处理器总线带宽来输入数据可不降低处理器组件的效率。

从每个处理器组件上得到的输出数据，通过多处理总线传送到母板上的一快公共存储器上。输出缓冲器也是一双端口缓冲器，除输出控制器外，它还受处理器单元的控制。输出控制器可以是主计算机，也可以是一外部输出控制器。当输出控制器不从缓冲器中接收数据时，处理器件可在它的最后一个时隙向缓冲器中写入数据。若象本文中那样，用主计算机做为输出控制器，那么输出缓冲器将直接分配在主机的存储器地址空间内。存储器一完成处理周期，主计算机就可读出输出数据。所以，在主机与处理器件间要建立一条快速数据传输线(典型的为 1.2us 每字)，因为这种情况下主机总线在数据传送期间不能受干扰。

在 DSP 的一些应用中，处理过程被限定为除相邻帧边界以外的帧内(例如 FIR 滤波器)。所以需选定一个公共存储器块用作处理器间数据传送。因处理器组件的操作是与输入帧同步的，故不需要总线调解，这样整个系统电路是很简明的。

2、主接口

选择 IBM PC 做为 DSP 系统的主计算机主要功能是：

(1)选择把应用程序装入各 TMS32010 处理器单元。

(2)设置系统参数。

(3)运行应用程序。

(4)完成通用的操作。

(5)提供系统诊断和开发。

主适配器设计成 PC 的插件形式。它通过一对带式电缆连接主机和实时 DSP。适配器对主机地址部分译码，把系统的 16 位数据多路传送到 PC 机 8 位总线上。DSP 的寄存器做为 PC 机的 I/O 端口编址，所以，处理器组件的程序和输出缓冲器的内容可通过存储器编址的 I/O 很方便地进行存取。

三、软件开发

本设计的开发程序分为两部分，即为操作系统和应用程序集。

操作系统主要是一C语言写的监控程序。这个程序在主机上运行，它控制DSP系统。这个程序的基本功能如下：

(1)装配和执行应用程序。TMS32010的应用汇编程序需提前准备，从主机按要求分别装入各处理器组件的程序存储器，然后运行初始化程序。一旦应用程序结束，系统控制就返回操作系统。

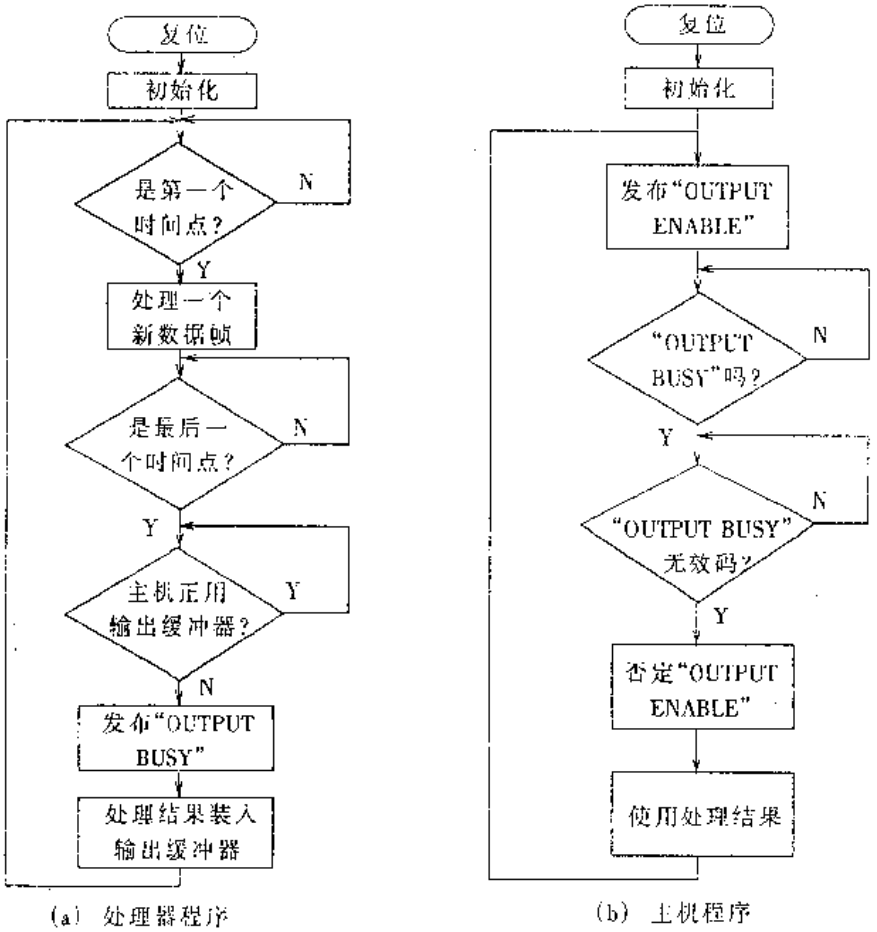
(2)修改系统参数。可调用这个功能来检查实时DSP的寄存器存储内容。在需要时也可修改这些寄存器内容。

(3)实时显示输出数据。输出数据可用图解显示，按每一个新的数据帧适时更新。

(4)状态存储。在某些情况下，需要把输出数据存入缓冲器，以便进行离线分析。所以主机中保留了一个16K字节的存储器用于存储输出样本，以便以后恢复。存储的数据可用图解方式显示，也可用数字格式显示。

(5)实用诊断。各种实用诊断程序提供了对硬件系统进行检查和故障排除的方便工具。

每个应用程序又可分成两部分。第一，汇编的TMS32010的机器码，它可在信号处理器上执行。第二，用高级语言写的控制部分，它在主机上执行。一个典型的应用程序执行的流程图如图6-41所示。



(a) 处理器程序　　(b) 主机程序

图6-41 典型应用程序流程图

本设计的一个主要原则是 TMS32010 的现有应用汇编程序在装入多处理器系统时只需进行很小的修改。另外，当处理器组件的数量改变时，程序结构将不用做修改。图 6－41 所示的流程图，是所有应用程序的标准，在每个处理器件上运行的机器码是相同的。

6.9 SIMD 多 DSP 图像处理系统

图像处理需要运算大量的数据，要求系统具有很高的数据吞吐量。并行处理结构能较好地满足这一要求。图像处理可以分成一系列的子处理（如预处理、边界判定、区域标号、量化及最终合成等），而且整个图像也可分成一系列图像块分别处理。因而并行处理适用做图像处理系统。本节介绍一种 SIMD 并行图像处理系统。该系统具有避免冲突、能连续处理图像数据、处理器间通信及 I/O 部分简单、硬件及软件模块化等优点。

一、SIMD 系统硬件结构

SIMD 并行处理系统构成如图 6－42 所示，包括 SIMD 并行多处理器单元、MC68000 单板控制器、IBM PC 主计算机及帧接收/显示单元等。

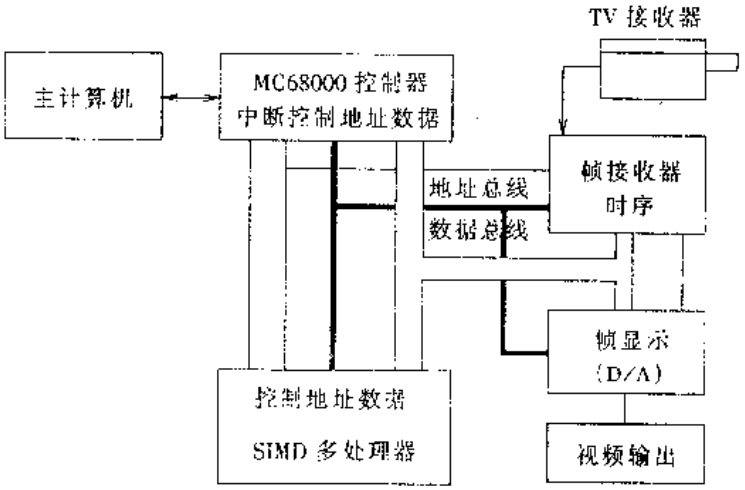


图 6－42 SIMS 并行处理器图象处理系统

控制器由 MC68000 微处理器、68230 外围接口/适配器、ACIA6850、程序存储器及数据存储器等构成，以串行线（a serial line）与其他单元连接。

SIMD 并行多处理器单元结构如图 6－43，由 8 个 DSP、8 个 16K 字双端口存储器（DPR）、8 个数据选择器、8 个局部存储器 SRAM、一个状态选择器及一个地址译码器构成。

SIMD 方式是一种单元指令多数据操作方式。即在控制单元控制下，各处理器对多路数据流进行相同程序指令的处理。本系统处理器选用 TI 公司的产品 TMS320C25。处理程序装在系统程序存储器中，由 TMS320C25 # 1 对其进行存取，并将程序转存入各处理器的局部 RAM 中。TMS320C25 # 1 还负责在处理完成后通知控制器。各处理器的运算及程序的传送均是同步进行的

每个 16K 字的 DPR 是由一个 2L * 8 位 Cypress CYTC132 和一个 2K * 8 位 CYTC142IC 以主从配置组成。其存取时间为 25ns。它的功能是存储原始图像和结果图象，以及和数据选择器

一起完成相邻处理器间通信。MC68000 访问它的 A 口，TMS320C25 访问其 B 口。各 TMS320C25 可读、写自己的 DPR，还可读相邻处理器的 DPR。通过数据选择器挑选要访问的 DPR。当两个处理器同时访问同一 DPR 而产生竞争时，由 DPR 产生一个仲裁逻辑，通过它的 BUSY 信号及等待状态发生器控制两处理器先后完成访问操作。

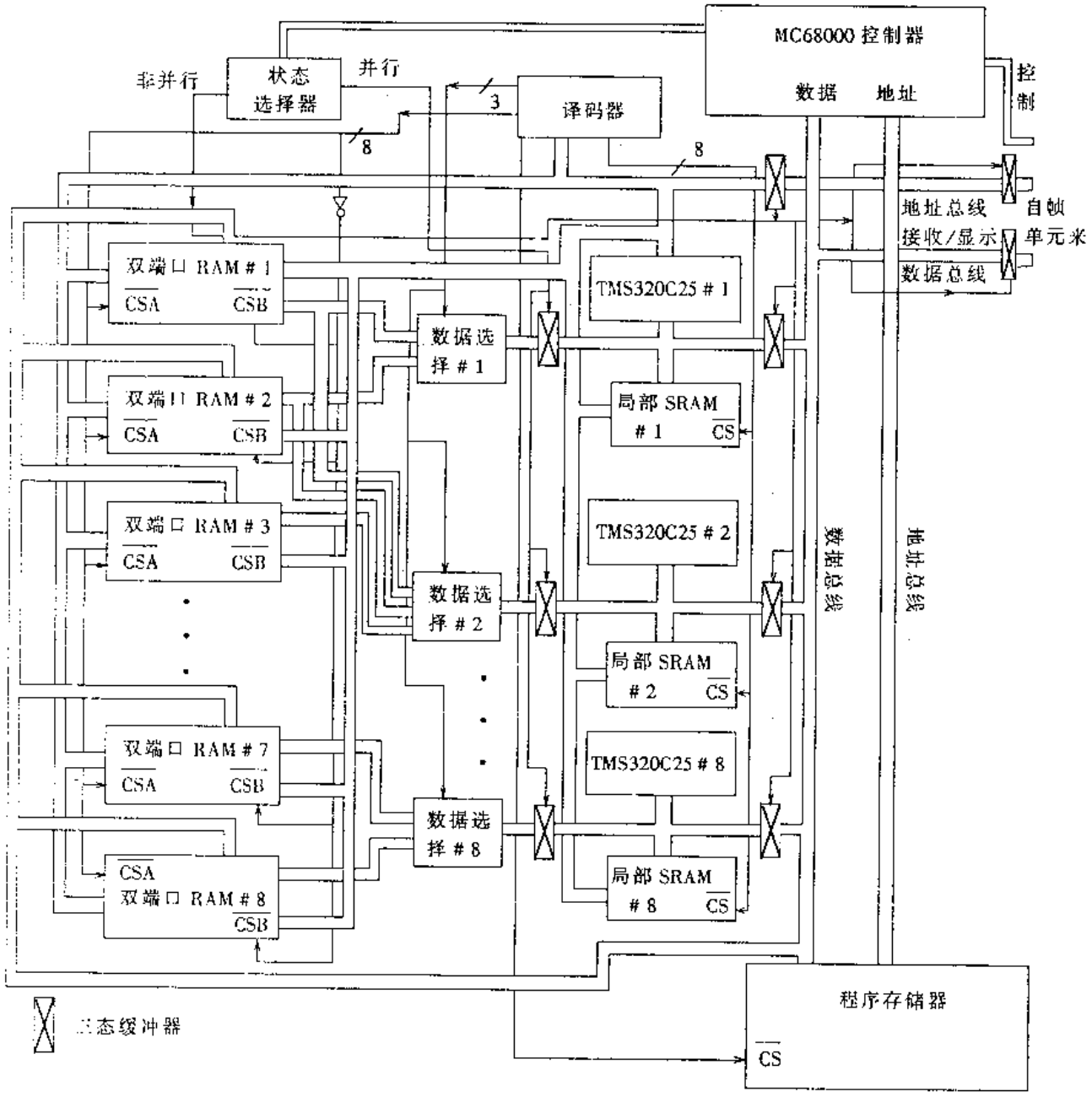


图 6-43　SIMD 并行处理单元结构

数据选择器和状态选择器都是定制的，由触发器和组合逻辑构成。状态选择器可控制系

统选择两种操作状态：并行状态和非并行状态。在非并行状态时，TMS320C25 处理器处于关闭或复位状态，各 DP、局部 SRAM 以及处理器的程序存储器均由 MC68000 以存储器方式存取。局部 SRAM 中装入所需的处理程序及 DPR 中装入相应数据后，控制器把系统初始化为并行状态。

系统程序存储器由 4K 高速静态 RAM 构成，为各处理器公用，设置在它们的直接寻址存储空间内。各处理器各自拥有的局部 SRAM，均用 Cypress 芯片 CYTC169 SRAM 构成，也分别设置在各处理器的直接寻空间内。注意各 SRAM 间是相互隔离的。地址译码器为 SN74AS138 译码器。

系统进行图像处理时，首先处于非并行状态，由控制器把处理程序装入公用程序 RAM，操作指令装令装入各处理器的局部 SRAM，帧接收器缓冲器中的原始数字化图像装入各 DPR（各 DPR 是相互隔离的）。之后，系统进入并行状态，各处理器开始以 SIMD 方式对 DPR 中图像进行同步处理，处理结果存入 DPR 结果单元。处理结束后 TMS320C25 向控制器发出中断信号，把系统重新设置为非并行状态。控制器中断帧显示单元，用 DPR 中的处理结果更新其缓冲器，显示新的处理图像。若结果还需进一步处理，则将其转送到 DPR 的原始图像单元，开始新的处理及显示。

二、图像处理算法

并行图像处理单位，常用到两个算法：用斜率判定边缘的算法及细化算法。

1. 边缘判定

图像分析的一个重要操作是确定边缘。边缘表征目标边界的特征，可用于分段、对位及识别等。图像中灰度突变点即可认为是边缘点。例如在黑白图像处理中，黑色边缘点至少与一个白色象素相邻，即象素单元(m,n)应有：u(m,n = 0,g(m,n) = 1 。此处：

$$g(m,n) = [u(m,n) \oplus u(m \pm 1,n)] .OR. [u(m,n) \oplus u(m,n \pm 1)] \qquad (6-31)$$

式中⊕表示异或操作。对连续图像 f(x,y)，其边缘方向的导数应为局部极大值。所以，可以通过 f 沿 r 的 0 角方向的斜率来进行边缘判定。这里引入斜率算子 H1 和 H2，计算图像 u(m,n)在两个垂直方向上的斜率。对数字化图像，该算子表示为平方向或竖直方向斜率的有限差分近似。算子尺寸为 3×3，其结构如下：

m−1,n−1	m−1,n+0	m−1,n+1
m+0,n−1	m+0,n+0	m+0,n+1
m+1,n−1	m+1,n+0	m+1,n+1

斜率向量的模值和方向为：

$$g(m,n) = (g_1^2(m,n) + g_2^2(m,n)^{\frac{1}{2}} \qquad (6-32)$$

$$\theta(m,n) = \tan^{-1}(g_2(m,n)/g_1(m,n)) \qquad (6-33)$$

$g_1(m,n)$和 $g_2(m-n)$的运算包括算子和象素间的乘法和加法。斜率向量模值也常用下式计算：

$$g(m,n) \triangleq |g_1(m,n)| + |g_2(m,n)| \qquad (6-34)$$

边缘判定常使用 Sobel 算子：

$$H_1 = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} \qquad H_2 = \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$$

Sobel 算子计算水平和垂直方向局部和差值。这样可以减少数据中噪声的影响。对于均匀区域，该算子计算结果为 0。若 g(m,n)超出阈值 t，则可判决定 u(m,n)是边缘点，各边缘点组成边缘线 e(m,n)，它定义为：

$$e(m,n) = \begin{cases} 1 & ,(m,n) \in Ig \\ 0 & ,其它 \end{cases} \tag{6-35}$$

式中：

$$Ig \triangleq \{(m,n); g(m,n) > t\} \tag{6-36}$$

通过边缘线就可追踪目标的边界了。一般地，t 值的选取通过 g(m,n)的累积直方图进行，使具有最大斜率的象素的 5%到 10%成为边缘点。

用 Sobel 算子时，需用到相邻象素。因此硬件中访问相邻存储器的功能对此处理很有用。

2.细化

用 Sobel 算子求出的边缘线线宽常大于一个象素，所以需对其加以细化，使线宽减为一个象素。细化算法大致是沿目标边缘中间轴线将其转换成一系列的简单弧线。所得结果不受原始图像中小和轮廓拐折的影响。一种简单的处理方法是从目标 X 中删除与 X 不只一点相邻的边界点，这种删除还须不断开 X。这里区域不断开的定义是区域内任两点都能由区域内曲线相连。这样，就可保证不删除细化弧线的端点。下面是一个产生连续续弧线而对轮廓噪声不敏感的简单算法。

参照图 6-44(a)，以 Z(0)表示有序集[P2,P3,P4,…P9,P2]中非零变换的零的数量，NZ(P_1)表示 P_1 的非零邻点的数量。当 P_1 满足下列条件时可删除(图 6-44(b))：

$$2 \leqslant NZ(P_1) \leqslant 6 \tag{6-37}$$

$$Z0(P_1) = 1 \tag{6-38}$$

$$P_2 \cdot P_4 \cdot P_8 = 0 \text{ 或 } Z0(P_2) \neq 1 \tag{6-39}$$

$$P_2 \cdot P_4 \cdot P_6 = 0 \text{ 或 } Z0(P_4) \neq 1 \tag{6-40}$$

反复进行这一过程，直至图像不再变化。

P_3	P_2	P_9
P_4	P_1	P_8
P_5	P_6	P_7

(a) 标号点 P1 及其邻点

1	1	0
1	P_1	1
0	0	0

(i)删除 P_1 将导致区域隔裂

0	0	0
1	P_1	0
0	0	0

(ii)删除 P_1 将缩线弧终点

1	0	1
0	P_1	0
1	1	1

(iii)$2 \leqslant NZ(P_1) \leqslant 6$ 但 P_1 不能删除

(b)

图 6-44 细化算法

三、图像数据的分块及分配

帧接收器中存储的图像尺寸可以为：128×128、256×256 或 512×512。在 SIMD 并行方式时，各处理器分别处理图像的一个固定部分，因此图像需相应地均匀分块。

例如，128×128 图像需分成 8 块 64×32 或 32×64。图 6-45 是分成 64×32 块。在并行处理时，块 #1 由处理，块 #2 又处理器 2 处理，等等。各图像块的数据量相同，所以处理时间相同。

块 #1 64×32	块 #2 64×32	块 #3 64×32	块 #4 64×32
块 #5 64×32	块 #6 64×32	块 #7 64×32	块 #8 64×32

图 6-45 128×128 图像分成 8 块

图像分块是系统在非并行状态时，由控制器完成，并分别存入各处理器的 DPR。对于 N×N 图像，分块方式如图 6-46 所示。

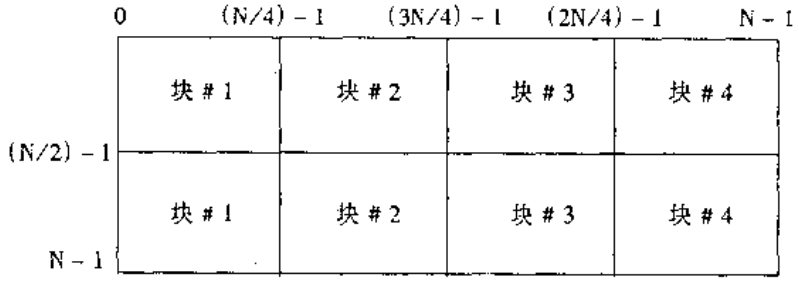


图 6-46 N×N 图像分块

在边缘判定时，处理器对每个象素均用 Sobel 算子处理。因 Sobel 算子是 3-3 核，所以有时要用到相邻处理器 DPR 中的图像象素。例如，当 IM(i,j)为图像块的左边界时，运算就要使用左邻图像块的象素 IM(i,j-1)、IM(i+1,j-1)和 IM(i-1,j-1)。因此，处理器 #1 和处理器 #8 仅需访问两个 DPR，其余处理器均访问三个 DPR。

四、系统性能分析

本系统的结构防止了对存储器及总线的争用。图像块的存入在非并行状态下进行，而其

读取在并行状态进行，这就消除了对 DPR 争用。图像块从帧接收器装入 DPR 在数据总线上进行，而处理结果存入帧接收器是在全部图像处理完成后才进行，这样不会出现对数据总线的争用。

因此，系统总处理时间取决于：图像装入 DPR 的时间 t_1；图像块的处理时间 t_2；处理结果送入帧接收器缓冲器的时间 t_3。系统总处理时间可表示为：

$$P = t_1 + t_2 + t_3 \tag{6-41}$$

t_1 包括对帧接收器的访问及把图像存入 DPR 的时间。由控制器在非并行状态用 MOVE.B Mem1，Mem2 指令依次把图像入 DPR 完成其分块及存入。

各处理器同步并行操作，所以全部图像处理时间 t_2 等于一个图像块的处理时间。它包括象素检索时间、图像处理及存入 DPR 时间、中间结果的读取及处理后再存入 DPR 的时间等。图像的读取用 LAC 指令，处理用 ADD 和 MAC 指令，结果存储用 SACL 和 SACH 指令。因 MAC 指令为单周期，所以处理速度很高。

t_3 包括把 DPR 中的结果依序存入显示缓冲器相应单元的时间。这一操作用 MOVE.B Mem2，Mem1 指令完成。

使用高速直接访问存储器控制器(DMA)也可提高系统速度。DMA 可以大大减少从帧接收器缓冲器到 DPR 及从 DPR 到显示器缓冲器的数据存储时间，使其小于处理时间，即 $t_1 < t_2$，$t_3 < t_2$。

本系统结构可以很容易地扩展更多的处理器及相应的 DPR 的数据选择器，构成更大的处理系统，处理更大的图像。

附录　TMS32020 系列指令系统

一、TMS32020 指令系统

指令系统使用符号和缩写说明如下。

符　号	意　　义
ACC	累加器
ARn	辅助寄存器 r(n = 0 - 4)　(AR0 到 AR4 是汇编默认符)
ARP	辅助寄存器指针
B	决定位码的 4 位字段
CM	决定比较方式的 2 位字段
D	数据存储器地址字段
DATn	赋于数据存储器单元口 n 的标号
dma	数据存储器地址
DP	数据页指针
FO	格式状态位
I	寻址方式位
INTM	中断方式标志位
K	立即操作数字段
> nn	nn 是 16 进制数,没有 > 前缀的数假定为 10 进制数
P	乘积寄存器(P)
PAn	端口地址(PA0 到 PA15 是预先定义的汇编符号)(n = 0 - 15)
PC	程序计数器
pma	程序存储器
PRGn	赋于程序存储器单元 n 的标号
R	规定辅助寄存器的 3 位操作数字段
S	4 位左移代码
T	T 寄存器
TOS	栈顶
X	累加器的 3 位左移字段
→	赋于
| |	表示绝对值
( )	角符号内的项由用户定义
[ ]	方括号内的项是可选择的
( )	表示它的内容
{ }	大括号内各项必须选择其一
)(	背对背的角括号表示不等于
⊔	在所示处插入空格

1 累加器、存储器指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ABS	累加器的绝对值	1	1 1 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 1
ADD	带移位累加器加	1	0 0 0 0 ← S → 1 ← D →
ADDC	带进位累加器加	1	0 1 0 0 0 0 1 1 1 ← D →
ADDH	累加器高位加	1	0 1 0 0 1 0 0 1 ← D →
ADDK	累加器加 8 位立即数	1	1 1 0 0 1 1 0 0 ← K →
ADDS	代抑制符号扩展累加器低位加	1	0 1 0 0 1 0 0 1 1 ← D →
ADDT *	按 T 寄存器移位并加到寄存器	1	0 1 0 0 1 0 1 0 1 ← D →
ADLK *	带移位的 16 位立即数加到累加器	2	1 1 0 1 ← S → 0 0 0 0 0 0 1 0
AND	和累加器逻辑“与”	1	0 1 0 0 1 1 1 0 1 ← D →
ANDK *	累加器和带移位的 16 位立即数逻辑“与”	2	1 1 0 1 ← S → 0 0 0 0 0 1 0 0
CMPL *	累加器求补	1	1 1 0 0 1 1 1 0 0 0 1 0 0 1 1 1
LAC	带移位的装入累加器	1	0 0 1 0 ← S → 1 ← D →
LACK	8 位立即数送累加器	1	1 1 0 0 1 0 1 0 ← K →
LACT *	按 T 寄存器移位并送累加器	1	0 1 0 0 0 0 1 0 1 ← D →
LALK *	带有移位的 16 位立即数送累加器	2	1 1 0 1 ← S → 0 0 0 0 0 0 0 1
NEG *	累加器取负	1	1 1 0 0 1 1 1 0 0 0 1 0 0 0 1 1
NORM *	累加器的内容规格化	1	1 1 0 0 1 1 1 0 1 ← D →
OR	和累加器逻辑“或”	1	0 1 0 0 1 1 0 1 1 ← D →
ORK *	带移位的 16 位立即数和累加器逻辑“或”	2	1 1 0 1 ← S → 0 0 0 0 0 1 0 1
ROL * *	累加器循环左移	1	1 1 0 0 1 1 1 0 0 0 1 1 0 1 0 0
ROR * *	累加器循环右移	1	1 1 0 0 1 1 1 0 0 0 1 1 0 1 0 1
SACH	带移位的存储累加器高位内容	1	0 1 1 0 1 ←X→ 1 ← D →
SACL	带移位的存储累加器低位内容	1	0 1 1 0 0 ←X→ 1 ← D →
SBLK *	累加器减去带有移位的 16 位立即数	2	1 1 0 1 ← S → 0 0 0 0 0 0 1 1
SFL *	累加器左移	1	1 1 0 0 1 1 1 0 0 0 0 1 1 0 0 0
SFR *	累加器右移	1	1 1 0 0 1 1 1 0 0 0 0 1 1 0 0 1
SUB	带有移位的累加器减	1	0 0 0 1 ← S → 1 ← D →

SUBB * *	带有借位的累加器减	1	0 1 0 0 1 1 1 1 1 ← D →
SUBC	条件减	1	0 1 0 0 0 1 1 1 1 ← D →
SUBH	累加器高位减	1	0 1 0 0 0 1 0 0 1 ← D →
SUBK * *	累加器减 8 位立即数	1	1 1 0 0 1 1 0 1 ← K →
SUBS	带有抑制符号扩展的累加器低位减	1	0 1 0 0 01 0 1 1 ← D →
SUBT *	累加器减去按 T 寄存器移位的内容	1	0 1 0 0 0 1 1 0 1 ← D →
XOR	和累加器逻辑"异或"	1	0 1 0 0 1 1 0 0 1 ← D →
XORK *	累加器和带有移位的 16 位立即数进行逻辑"异或"	2	1 1 0 1 ←S→ 0 0 0 0 0 1 1 0
ZAC	累加器置零	1	1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0
ZALH	累加器低位置零并装入累加器高位	1	0 1 0 0 0 0 0 0 1 ← D →
ZALR * *	累加器低位置零并带有舍入地装入累加器高位	1	0 1 1 1 1 0 1 1 1 ← D →
ZALS	累加器低位置零并带有抑制符事情扩展地装入累加器低位	1	0 1 0 0 0 0 0 1 1 ← D →

2. 辅助寄存器和页指针指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ADRK * *	辅助寄存器加 3 位立即数	1	0 1 1 1 1 1 1 0 ← K →
CMPR *	辅助寄存器与辅助寄存器 AR0 进行比较	1	1 1 0 0 1 1 1 0 0 1 0 1 0 0 CM
LAR D →	装入辅助寄存器	1	0 0 1 1 0 ← R → 1
LARK	8 位立即数送到辅助寄存器	1	1 1 0 0 0 ← R → ← K →
LARP	装入辅助寄存器指针	1	0 1 0 1 0 1 0 1 1 0 0 0 1 ← R →
LDP	装入数据存储器页指针	1	0 1 0 1 0 0 1 0 1 ← D →
LDPK	立即数送入数据存储器页指针	1	1 1 0 0 1 0 0 ← DP →
LRLK *	16 位立即数送入辅助寄存器	2	1 1 0 1 0 ←R→ 0 0 0 0 0 0 0 0
MAR	修改辅助寄存器	1	0 1 0 1 0 1 0 1 1 ← D →
SAR	存储辅助寄存器	1	0 1 1 1 0 ← R → 1 ← D →
SBRK * *	辅助寄存器减去 8 位立即数	1	0 1 1 1 1 1 1 1 ← K →

3. 乘法指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
APAC	累加器加 P 寄存器	1	1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1
LPH *	装入 P 寄存器高位	1	0 1 0 1 0 0 1 1 1 ← D →
LT	装入 T 寄存器	1	0 0 1 1 1 1 0 0 1 ← D →
LTA	送 T 寄存器,并累加前面的积	1	0 0 1 1 1 1 0 1 1 ← D →
LTD	送 T 寄存器,累加前面的积并传送数据	1	0 0 1 1 1 1 1 1 1 ← D →
LTP *	送 T 寄存器并把 P 寄存器内容存储到累加器	1	0 0 1 1 1 1 1 0 1 ← D →
LTS *	送 T 寄存器并减去前面的积	1	0 1 0 1 1 0 1 1 1 ← D →
MAC *	乘并且累加	2	0 1 0 1 1 1 0 1 1 ← D →
MACD *	乘并且带有数据传送的累加	2	0 1 0 1 1 1 0 0 1 ← D →
MPY	乘(与 T 寄存器乘,其乘积在 P 寄存器中)	1	0 0 1 1 1 0 0 0 1 ← D →
MPYA * *	乘并累加前面的乘积	1	0 0 1 1 1 0 1 0 1 ← D →
MPYK	乘立即数	1	1 0 1 ← K →
MPYS * *	乘并减去前面的乘积	1	0 0 1 1 1 0 1 1 1 ← D →
MPYU * *	无符号乘	1	1 1 0 0 1 1 1 1 1 ← D →
PAC	P 寄存器内容送累加器	1	1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0
SPAC	累加器减 P 寄存器	1	1 1 0 0 1 1 1 0 0 0 0 1 0 1 1 0
SPH * *	存储 P 寄存器高位	1	0 1 1 1 1 1 0 1 1 ← D →
SPL * *	存储 P 寄存器低位	1	0 1 1 1 1 1 0 0 1 ← D →
SPM *	置 P 寄存器输出移位方式	1	1 1 0 0 1 1 1 0 0 0 0 0 1 0 PM
SQRA *	平方并加累加器内容	1	0 0 1 1 1 0 0 1 1 ← D →
SQRS *	平方并减去前面的乘积	1	0 1 0 1 1 0 1 0 1 ← D →

4. 转移和调用指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
B	无条件转换	2	1 1 1 1 1 1 1 1 1 ← D →
BACC *	按累加器内容转移	2	1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 1

BANZ	辅助寄存器不为零时转移	2	1 1 1 1 1 0 1 1 1 ← D →
BBNZ *	如果 TC 位≠0 转移	2	1 1 1 1 1 0 0 1 1 ← D →
BBZ *	如果 TC 位 = 0 转移	2	1 1 1 1 1 0 0 0 1 ← D →
BC * *	有进位时转移	2	0 1 0 1 1 1 1 0 1 ← D →
BGEZ	如果累加器≥0 转移	2	1 1 1 1 0 1 0 0 1 ← D →
BGZ	如果累加器 > 0 转移	2	1 1 1 1 0 0 0 1 1 ← D →
BIOZ	I/O 状态 = 0 转移	2	1 1 1 1 1 0 1 0 1 ← D →
BLEZ	如果累加器≤0 转移	2	1 1 1 1 0 0 1 1 1 ← D →
BLZ	如果累加器 < 0 转移	2	0 1 0 1 1 1 1 1 1 ← D →
BNC * *	没有进位时转移	2	0 1 0 1 1 1 1 1 1 ← D →
BNV *	没有溢出时转移	2	1 1 1 1 0 1 1 1 1 ← D →
BNZ	如果累加器≠0 转移	2	1 1 1 1 0 1 0 1 1 ← D →
BV	溢出转移	2	1 1 1 1 0 0 0 0 1 ← D →
BZ	累加器 = 0 转移	2	1 1 1 1 0 1 1 0 1 ← D →
CALA	间接调用子程序	2	1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0
CALL	调用程序	2	1 1 1 1 1 1 1 0 1 ← D →
RET	从子程序中返回	2	1 1 0 0 1 1 1 0 0 0 1 0 0 1 1 0

5. 控制和测试指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
BIT *	位测试	1	1 0 0 1 ← B → 1 ← D →
BITT *	测试由 T 寄存器指定的位	1	0 1 0 1 0 1 1 1 1 ← D →
CNFD *	构成数据存储器块	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 0 0
CNFP *	构成程序存储器块	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 0 1
DINT	禁止中断	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1
EINT	允许中断	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0
IDLE *	等待中断	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 1
LST	装入状态寄存器 ST0	1	0 1 0 1 0 0 0 0 1 ← D →
LST1 *	装入状态寄存器 ST1	1	0 1 0 1 0 0 0 1 1 ← D →

NOP	空操作	1	0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0
POP	退栈到累加器低位	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 1
POPD *	退栈到数据存储器	1	0 1 1 1 1 0 1 0 1 ← D →
PSHD *	数据存储器的值值接进栈	1	0 1 0 1 0 1 0 0 1 ← D →
PUSH	累加器低位进栈	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 0
RC * *	复位进位位	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 0
RHM * *	复位保持方式	1	1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 0
RPT *	根据数据存储器所指定的值重复执行指令	1	0 1 0 0 1 0 1 1 1 ← D →
RPTK *	根据立即数值重复执行指令	1	1 1 0 0 1 0 1 1 1 ← D →
RSXM	复位符号扩展方式	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 0
RTC * *	复位测试/控制标志	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0
SC * *	置进位位	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1
SHM * *	置保持方式	1	1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1
SOVM	置溢出方式	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1
SST	存储状态寄存器 ST0	1	0 1 1 1 1 0 0 0 1 ← D →
SST1 *	存储状态寄存器 ST1	1	0 1 1 1 1 0 0 1 1 ← D →
SSXM *	置符号扩展方式	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1
STC * *	置测试/控制标志	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1
TRAP *	软件中断	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0

6. I/O 和数据存储器操作、指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
BLKD *	数据存储器之前的块传送	2	1 1 1 1 1 1 0 1 1 ← D →
BLKP *	把程序存储器的块传送到数据存储器	2	1 1 1 1 1 1 0 0 1 ← D →
DMOV	在数据存储器内进行数据传送	1	0 1 0 1 0 1 1 0 1 ← D →
FORT *	设置串行口寄存器方式	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1 F
IN	从口中读入数据	1	1 0 0 0 ←PA→ 1 ← D →
OUT	把数据输出到口上	1	1 1 1 ← PA → 1 ← D →
RFSM * *	复位串行口帧同步方式	1	1 1 0 0 1 1 1 0 0 0 1 1 0 1 1 0

RTXM *	复位串行口发送方式	1	1 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0
RXF *	复位外部标志	1	1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0
SFSM * *	置串行口帧同步方式	1	1 1 0 0 1 1 1 0 0 0 1 1 0 1 1 1
STXM *	置串行口发送方式	1	1 1 0 0 1 1 1 0 0 0 1 0 0 0 0 1
SXF *	置外部标志	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0
TBLR	表读	1	0 1 0 1 1 0 0 0 1 ← D →
TBLW	表写	1	0 1 0 1 1 0 0 1 1 ← D →

注：* 表示这些指令不包括在 TMS32010 指令集中

* * 表示这些指令仅包括在 TMS320C25 指令集中

二、TMS320C25 指令系统

TMS320C25 指令系统中使用符号和缩写说明如下：

符　号	意　　义
ACC	累加器
ARB	辅助寄存器 n(AR0 到 AR7 是预先定义的汇编语言符号,分别等于 0 到 7)
ARP	辅助寄存器指针
B	决定位码的 4 位字段
BIO	转移控制输入
C	进位位
CM	决定比较方式的 2 位字段
CNF	片内 RAM 构成控制位
D	数据存储器地址字段
DATn	赋于数据存储器单元口 n 的标号
dma	数据存储器地址
DP	数据页指针
FO	格式状态位
FSM	帧同步方式位
HM	保持方式位
I	寻址方式位
INTM	中断方式标志位
K	立即操作数字段
> nn	nn 是 16 进制数(其他数认定为 10 进制数)
OV	溢出方式标志位
OVM	溢出方式位
P	乘积寄存器
PA	端口地址(PA0 到 PA15 是预先定义的汇编符号,分别等于 0 - 15)
PC	程序计数器
PM	规定 P 寄存器输出移位不写的 2 位字符
pma	程序存储器
PRGn	赋于程序存储器单元 n 的标号
R	规定辅助寄存器的 3 位操作数字段

RPTC	重要计数器
S	4 位左移代码
STn	状态寄存器(ST0 或 ST1)
SXM	符号扩展方式位
T	暂存寄存器
TC	测试控制位
TOS	栈顶
TXM	发送方式位
X	3 位累加器左移字段
XF	XF 引脚状态位
|	表示取绝对值
→	赋于
()	括号内的项由用户定义
[]	方括号内的项是可选的
()	表示它的内容
| |	括号内各项必须选择其一
⊔	在所示处,插入空格

1. 累加器存储器访问指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ABS	累加器内容取绝对值	1	1 1 0 0 1 1 1 0 0 0 0 1 1 0 1 1
ADD	带移位加到累加器	1	0 0 0 0 ← S → 1 ← D →
ADDC * *	带进位加到累加器	1	0 1 0 0 0 0 1 1 1 ← D →
ADDH	加到累加器高 16 位	1	0 1 0 0 1 0 0 0 1 ← D →
ADDK * *	短立即数加到累加器	1	1 1 0 0 1 1 0 0 ← D →
ADDS	加到累加器低 16 位,抑制符号扩展	1	0 1 0 0 1 0 0 1 1 ← D →
ADDT *	加到累加器,包含由 T 寄存器确定的移位	1	0 1 0 0 1 0 1 0 1 ← D →
ADLK *	长立即数带移位加到累加器	2	1 1 0 1 ← S → 0 0 0 0 0 0 1 0
AND	同累加器"与"	1	0 1 0 0 1 1 1 0 1 ← D →
ANDK *	立即数同累加器"与",带移位	2	1 1 0 1 ← S → 0 0 0 0 0 1 0 0
CMPL *	累加器取反码	1	1 1 0 0 1 1 1 0 0 0 1 0 0 1 1 1
LAC	带移位装入累加器	1	0 0 1 0 ← S → 1 ← D →
LACK	短立即数装入累加器	1	1 1 0 0 1 0 1 0 ← K →
LACT *	装不加器,包含由 T 寄存确定的移位	1	0 1 0 0 0 0 1 0 1 ← K →
LALK *	长立即数带移位装入累加器	2	1 1 0 1 ← S → 0 0 0 0 0 0 0 1

NEG *	累加器取相反数	1	1 1 0 0 1 1 1 0 0 0 1 0 0 0 1 1
NORM *	累加器内容归一	1	1 1 0 0 1 1 1 0 1 ← D →
OR	同累加器"或"	1	0 1 0 0 1 1 0 1 1 ← D →
ORK *	立即数带移位与累加器"或"	2	1 1 0 1 ← S → 0 0 0 0 0 1 0 1
ROL * *	累加器循环左移	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 0
ROR * *	累加器循环右移	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 1
SACH	存储累加器高 16 位,带移位	1	0 1 1 0 1 ←X →1 ← D →
SACL	存储累加器低 16 位,带移位	1	0 1 1 0 0 ←S →1 ← D →
SBLK *	累加器减去经移位的长立即数	2	1 1 0 1 ← S → 0 0 0 0 0 0 1 1
SFL *	左移累加器	1	1 1 0 0 1 1 1 0 0 0 0 1 1 0 0 0
SFR *	右移累加器	1	1 1 0 0 1 1 1 0 0 0 0 1 1 0 0 1
SUB	累加器减,带移位	1	0 0 0 1 ← S →1 ← D →
SUBB * *	累加器减,带借位	1	0 1 0 0 1 1 1 1 1 ← D →
SUBC	条件减	1	0 1 0 0 0 1 1 1 1 ← D →
SUBH	从累加器高 16 位减	1	0 1 0 0 0 1 0 0 1 ← D →
SUBK * *	累加器减去短立即数	1	1 1 0 0 1 1 0 1 ← K →
SUBS	从累加器低 16 位减,抑制符号扩展	1	0 1 0 0 0 1 0 1 1 ← D →
SUBT *	累加器减,包含由 T 寄存器确定的移位	1	0 1 0 0 0 1 1 0 1 ← D →
XOR	同累加器"异或"	1	0 1 0 0 1 1 0 0 1 ← D →
XORK *	立即数同累加器"异或"带移位	2	1 1 0 1 ← S → 0 0 0 0 0 1 1 0
ZAC	累加器清零	1	1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0
ZALH	累加器低 16 位清零,装入累加器高 16 位	1	0 1 0 0 0 0 0 0 1 ← D →
ZALR *	累加器低 16 位清零,带舍入装累加器高 16 位	1	0 1 1 1 1 0 1 1 1 ← D →
ZALS	累加器清零,装入累加器低 16 位,抑制符号扩展	1	0 1 0 0 0 0 0 1 1 ← D →

2. 辅助寄存器和数据页指针指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ADRK * *	短立即数加入辅助寄存器	1	0 1 1 1 1 1 1 0 ← D →
CMPR *	辅助寄存器与辅助寄存器 ARO 比较	1	1 1 0 0 1 1 1 0 0 1 0 1 0 0 CM

LAR	装入辅助寄存器	1	0 0 1 1 0 ← R → 1 ← D →
LARK	短立即数装入辅助寄存器	1	1 1 0 0 0 ← R → ← K →
LARP	装入辅助寄存器指针	1	0 1 0 1 0 1 0 1 1 0 0 0 1 ← R →
LDP	装入数据存储器页指针	1	0 1 0 1 0 0 1 0 1 ← D →
LDPK	立即数装入数据存储器页指针	1	1 1 0 1 0 0 ← DP →
ORLK *	立即数装入辅助寄存器	2	1 1 0 1 0 ← R → 0 0 0 0 0 0 0 0
MAR	修改辅助寄存器	1	0 1 0 1 0 1 0 1 1 ← D →
SAR	存储辅助寄存器	1	0 1 1 1 0 ←R→ 1 ← D →
SBRK * *	辅助寄存器减去短立即数	1	0 1 1 1 1 1 1 1 ← K →

3. P 寄存器、T 寄存器和乘法指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
APAC	P 寄存器加到累加器	1	1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1
LPH *	装入 P 寄存器高 16 位	1	0 1 0 1 0 0 1 1 1 ← D →
LT	装入 T 寄存器	1	0 0 1 1 1 1 0 0 1 ← D →
LTA	装入 T 寄存器并累加前一次乘积	1	0 0 1 1 1 1 0 1 1 ← D →
LTD	装入 T 寄存器，累加前一次乘积并传送数据	1	0 0 1 1 1 1 1 1 1 ← D →
LTP *	装入 T 寄存器，P 寄存器存入累加器	1	0 0 1 1 1 1 1 0 1 ← D →
LTS *	装入 T 寄存器并减去前一次乘积	1	0 1 0 1 1 0 1 1 1 ← D →
MAC *	相乘并累加	2	0 1 0 1 1 1 0 1 1 ← D →
MACD *	相乘并累加，带数据传送	2	0 1 0 1 1 1 0 0 1 ← D →
MPY	与 T 寄存器乘，乘积存入 P 寄存器	1	0 0 1 1 1 0 0 0 1 ← D →
MPYA * *	乘并累加前一次乘积	1	0 0 1 1 1 0 1 0 1 ← D →
MPYK	乘立即数	1	1 0 1 ← K →
MPYS * *	乘并减去前一次乘积	1	0 0 1 1 1 0 1 1 1 ← D →
MPYU * *	无符号乘	1	1 1 0 0 1 1 1 1 1 ← D →
PAC	P 寄存器装入累加器	1	1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0
SPAC	从累加器减去 P 寄存器	1	1 1 0 0 1 1 1 0 0 0 0 1 0 1 1 0
SPH * *	存 P 寄存器高 16 位	1	0 1 1 1 1 1 0 1 1 ← D →

SPL * *	存 P 寄存器低 16 位	1	0 1 1 1 1 1 0 0 1 ← D →
SPM * *	设置 P 寄存器输出移位方式	1	1 1 0 0 1 1 1 0 0 0 0 0 1 0 PM
SQRA *	平方并累加	1	0 0 1 1 1 0 0 1 1 ← D →
SQRS *	平方并减去前一次乘积	1	0 1 0 1 1 0 1 0 1 ← D →

4. 转移/调用指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
B	无条件转移	2	1 1 1 1 1 1 1 1 1 ← D →
BACC *	转移到由累加器内容指定	1	1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 1
BANZ	辅助寄存器不为零时转移	2	1 1 1 1 1 0 1 1 1 ← D →
BBNZ *	若 TC 位≠0，则转移	2	1 1 1 1 1 0 0 1 1 ← D →
BBZ *	若 TC 位 =0，则转移	2	1 1 1 1 1 0 0 0 1 ← D →
BC * *	有进位则转移	2	0 1 0 1 1 1 1 0 1 ← D →
BGEZ	若累加器 ACC≥0，则转移	2	1 1 1 1 0 1 0 0 1 ← D →
BGZ	若累加器 ACC > 0，则转移	2	1 1 1 1 0 0 0 1 1 ← D →
BIOZ	I/O 状态 = 0，则转移	2	1 1 1 1 1 0 1 0 1 ← D →
BLEZ	若累加 ACC≤0，则转移	2	1 1 1 1 0 0 1 1 1 ← D →
BLZ	若累加器 ACC < 0，则转移	2	0 1 0 1 1 1 1 1 1 ← D →
BNC * *	无进位则转移	2	0 1 0 1 1 1 1 1 1 ← D →
BNV *	无溢出则转移	2	1 1 1 1 0 1 1 1 1 ← D →
BNZ	若累加器 ACC≠0 则转移	2	1 1 1 1 0 1 0 1 1 ← D →
BV	有溢出则转移	2	1 1 1 1 0 0 0 0 1 ← D →
BZ	若累加器 ACC = 0 则转移	2	1 1 1 1 0 1 1 0 1 ← D →
CALA	间接调用子程序	1	1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0
CALL	调用子程序	2	1 1 1 1 1 1 1 0 1 ← D →
RET	从子程序返回	1	1 1 0 0 1 1 1 0 0 0 1 0 0 1 1 0

5. I/O 和数据存储器操作指令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
BLKD *	从数据存储器到数据存储器的块传送	2	1 1 1 1 1 1 0 1 1 ← D →
BLKP *	从程序存储器到数据存储器的块传送	2	1 1 1 1 1 1 0 0 1 ← D →
DMOV	数据存储器中的数据传送	2	0 1 0 1 0 1 1 0 1 ← D →
FORT *	格式化有串行口寄存器	1	1 0 0 1 1 1 0 0 0 0 0 1 J J FO
IN	由端口输入数据	1	1 0 0 0← PA →1 ← D →
OUT	输出数据至端口	1	1 1 1 0← PA →1 ← D →
RFSM * *	串行口帧同步方式复位	1	1 1 0 0 1 1 1 0 0 0 1 1 0 1 1 0
RTXM *	串行口发送方式复位	1	1 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0
RXF *	外部标志复位	1	1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0
SFSM *	串行口帧同步方式复位	1	1 1 0 0 1 1 1 0 0 0 1 1 0 1 1 1
STXM *	串行口发送方式复位	1	1 1 0 0 1 1 1 0 0 0 1 0 0 0 0 1
SXF *	外部标志置位	1	1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 1
TBLR	表读	1	0 1 0 1 1 0 0 0 1 ← D →
TBLW	表写	1	0 1 0 1 1 0 0 1 1 ← D →

6. 控制命令

助记符	说　明	字数	指令位代码
			15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
BIT *	位测试	1	1 0 0 1← B →1 ← D →
BITT *	测试由 T 寄存器指定的位	1	0 1 0 1 0 1 1 1 1 ← D →
CNFD *	BO 块作为数据存储器配置	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 0 0
CNFP *	BO 块作为程序存储器配置	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 0 1
DINT	关中断	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1
EINT	开中断	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0
IDLE *	闲置等待中断	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 1
LST	装入状态寄存器 ST0	1	0 1 0 1 0 0 0 0 1 ← D →
SLTI	装入状态寄存器 ST1	1	0 1 0 1 0 0 0 1 1 ← D →
NOP	空操作	1	0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0

POP	弹出栈顶到累加器低16位	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 1
POPD *	弹出栈顶到数据存储器	1	0 1 1 1 1 0 1 0 1 ← D →
PSHD *	将数据存储器内容压入堆栈	1	0 1 0 1 0 1 0 0 1 ← D →
PUSH	累加器低16位压入堆栈	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 0
RC * *	复位进位位	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 0
RHM * *	复位保持方式	1	1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 0
ROVM	复位溢出方式	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 0
RPT *	由数据存储器内容确定重复次数的重复指令	1	0 1 0 0 1 0 1 1 1 ← D →
RPTK *	由立即数确定的重复指令	1	1 1 0 0 1 0 1 1 ← K →
RSXM	复位符号扩展方式	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 0
RTC * *	复位测试/控制标志	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0
SC * *	进位位置位	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1
SHM * *	保持方式置位	1	1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1
SOVM	溢出方式置位	1	1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1
SST	存储状态寄存器ST0	1	0 1 1 1 1 0 0 0 1 ← D →
SST1 *	存储状态寄存器ST1	1	0 1 1 1 1 0 0 1 1 ← D →
SSXM *	置位符号扩展方式	1	1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1
STC * *	置位测试/控制标志	1	1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1
TRAP *	软中断	1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0

符号 * 表示在 TMS32010 指令系统中不包括这些指令。

符号 * * 表示在 TMS32020 指令系统中不包括这些指令

参考文献

[1] 山下,"TMS32010 の ハ－ドウェア",トランジスタ技术,1989.2

[2] 三上,"TMS32010 の プログラミング技法",トランジスタ技术,1989.2

[3] 山下,"TMS32010の周边回路设计技术",トランジスタ技术,1989.2

[4] 丸田,"信号处理用 VLSIア－キテクチャ",计测と制御,1988.12

[5] 油田,"ディジタルフィルタ设计のための理论",インタフェ－ス,1988.5

[6] 山内,"TMS320C30の概要とハ－ドウェア构成",インタフェ－ス,1991.3

[7] 山内,"TMS320C30 搭载のPC－98 扩张ボ－ドの设计",インタフェ－ス,1991.5

[8] 山内,"TMS320C30の应用",インタフェ－ス,1991.7

[9] 堤松,"ディジタル·ワィルタ设计の基础と实现法",インタフェ－ス,1987.9

[10] 李兰友等,"TMS32020 汇编程序设计",电子仪表,1990.6

[11] 李兰友,"μPD77230 及应用",电子仪表,1990.11

[12] 李兰友,"第三代信号处理器 TMS320C30",电子仪表,1991.2

[13] "TMS320C25 digital signal processor", Microprocessors and Microsystem, Vol 12 No 9 ,1988

[14] Subramaniam Ganesan, "Adual－DSP microprocessor system for real－time digital correlation", Microprocessors and Microsystem, Vol 15 No 7 ,1991

[15] "Implementation of PID Controllers on the Motorola DSP 56000/56001", Microprocessors and Microsystem, Vol 15 No 6 ,1991

[16] TMS32020 用户手册

[17] TEXAS INSTRUMENTS, "Digital Signal Processing Applications" ,1988

[18] 王新扬,陈尚勤,"用于高速信息处理的 TMS32010 微处理系统",微电子学与计算机,1990.2

[19] Paruvachi V R Raja et. "An SIMD multiple DSP Microprocessor System for image processing", Microprocessors and Microsystem, Vol 15 No 9 ,1991

[20] M H Er et. "Design and implementation of an RSA cryptosystem using multiple DSP chips", Microprocessors and Microsystem, Vol 15 No 7 ,1991

[21] J lin et. "DMA－based communications between PC and DSP" ,Microprocessors and Microsystem, Vol 15 No 3 ,1991

[22] E Horn et. "TMS32020 implementation of an adaptive recursive echo canceller", Microprocessors and Microsystem, Vol 12 No 9 ,1988

Powered by xiaoguo’s publishing studio
QQ:8204136