目　　次

第一篇　DSP56000 系列数字信号处理器及其开发系统

第二篇 DSP96002 浮点双端口处理器用户手册

第一篇　DSP56000 系列数字信号处理器及其开发系统

第一章　DSP56000 和 DSP56001 数字信号处理器及其应用开发系统 DSP56000 ADS 导论

DSP56000 和 DSP56001 是通用的单片数字信号处理器（DSP）。DSP56000ADS（Application Development System）是硬件开发工具，它用 DSP56001 来设计实时 DSP 系统。

本章概述 DSP56000 和 DSP56001 处理器以及 ADS 硬件。还给出了建立 ADS、设置 ADS56000 用户接口软件程序以及设置 DSP56000/1 示范软件程序的步进指令（DSP56000/1 示范软件程序通过屏幕菜单有选择地提供本书所选定的题目）。然后描述 ADS 用户接口命令，并用于汇编 执行和测试 DSP56000/1 目的码程序。

1.1　DSP56000 和 DSP56001 处理器概述

DSP56000 和 DSP56001 处理器是高速、通用的 DSP，它采用高密度低功耗和 5V HCMOS 技术实现。DSP56000 和 DSP56001 处理器都是双哈佛结构。包括一个片上程序存贮器和两个独立的片上数据存贮器，三者均可在外部扩展到 64 千字（总和为 192 千字），并可用零等待状态访问。DSP56000 处理器有可保密的 3.75 千字片上程序 ROM，而 DSP56001 处理器有 512 字由片上引导装入程序支持的片上程序 RAM。DSP56000 和 DSP56001 处理器都具有丰富的特性，它们共同使高性能 DSP 或微控制系统得以有效地实现。这些特性包括：

1.1.1　高性能结构

DSP56000 和 DSP56001 处理器都有三个多模的片上外围，它们可增强性能、有助于使系统布置紧凑并降低总价格。DSP56000 和 DSP56001 处理器都用无阻塞三级指令取数/译码/执行流水线，由 7 条独立总线连接到 3 个独立的片上存贮器块的 3 个独立执行单元组成，以提供高性能数字信号处理所需的并行性。例如，这并行性使得 4 系数的无限脉冲响应（IIR）滤波器节的实现仅需 4 个指令周期，这对单乘法器结构而言是理论最小值。

DSP56000 和 DSP56001 处理器的主要结构分别示于图 1-1 和图 1-2，这些结构包括：

1.3 个独立的执行单元

（1）数据 ALU；

（2）地址产生单元；

（3）程序控制器。

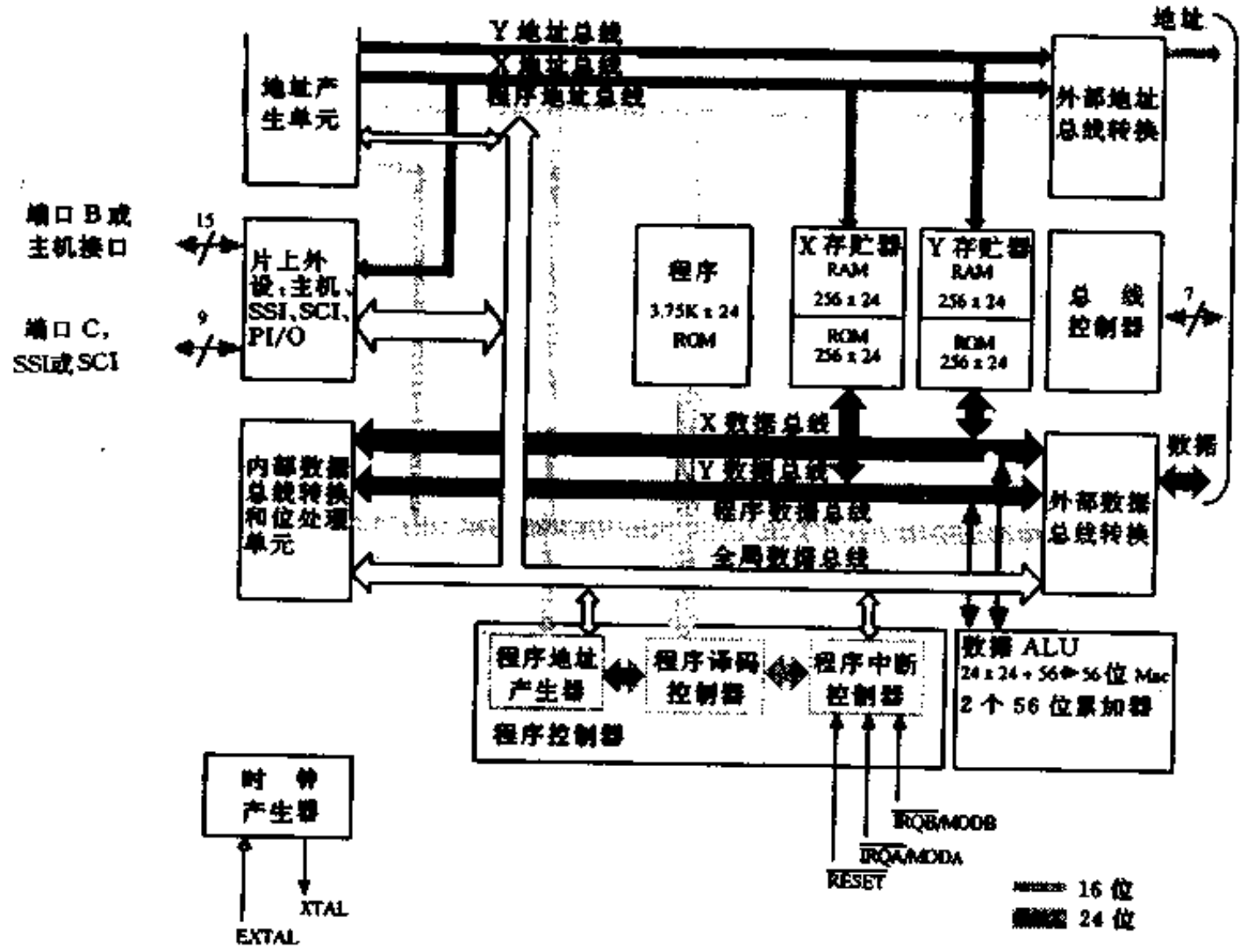


图 1－1　DSP56000 框图

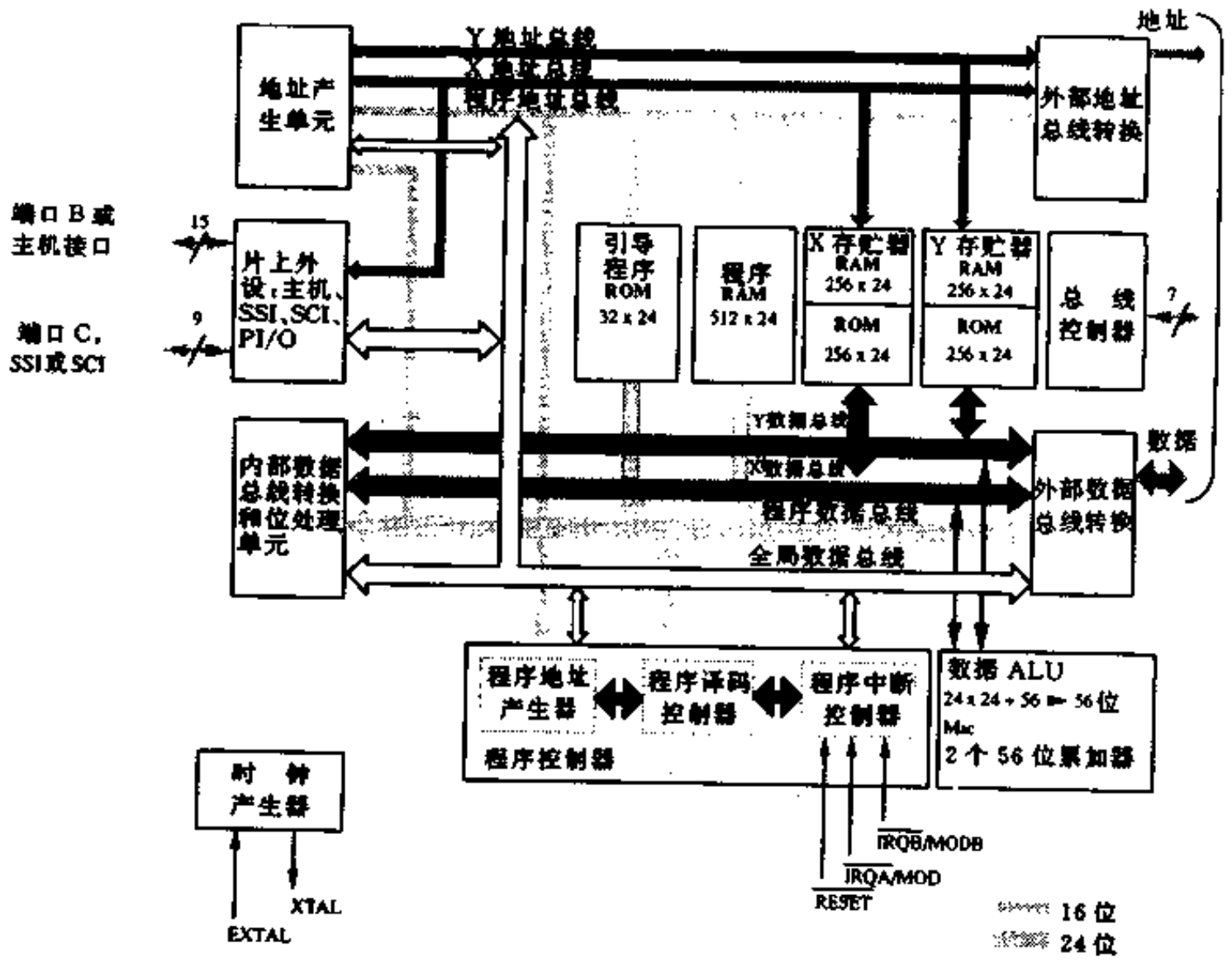


图 1－2　DSP56001 框图

2.6 个片上存贮器

DSP56000：	DSP56001
1 个 3.75 千字程序 ROM	1 个 512 字程序 RAM
1 个 256 字 X -数据 RAM	1 个 256 字 X -数据 RAM
1 个 256 字 Y -数据 RAM	1 个 256 字 Y -数据 RAM
2 个 256 字用户定义的 ROM	2 个 256 字已编程 ROM
1 个 32 字 ROM（未用）	1 个 32 字引导程序 ROM

3.4 条独立的 24 位数据总线

(1) XD 数据总线；

(2) YD 数据总线；

(3) PD 程序数据总线；

(4) GD 全数据总线。

4.3 条独立的 16 位地址总线

(1) XA 地址总线；

(2) YA 地址总线；

(3) PA 程序地址总线。

5. 一个存贮器扩展端口 A，该端口具有：

(1) 为 24 位数据字设置的 24 个引出端；

(2) 为 16 位存贮块线性寻址 64 千字设置的 16 个引出端；

(3) 为分段选择 64 千字程序存贮器块、64 千字 X -数据存贮器块或 64 千字 Y -数据存贮器块设置的 3 个引出端；

(4) 为握手功能设置的 4 个引出端。

6.3 个多模的片上外设

(1) 1 个串行通信接口或通用 I/O 端口 C

(2) 1 个同步串行接口或通用 I/O 端口 C

(3) 1 个并行主机接口或通用 I/O 端口 B

7.1 个时钟产生器

1.1.2 高速率

在 20.48MHz 时钟速率时，DSP56000 和 DSP56001 处理器均可执行 10.24MIPS (million instructions per second)，包括高达每秒 30.72 百万次的并发运算和双数据的传送操作 (MOPS)。在 27MHz 时钟速率时，性能分别变成 13.5MIPS 和 40.5MOPS。例如，在 20.48MHz 的时钟速率时，DSP56000/1 可在 3.39ms 内用 24 位定点运算执行 1024 点的复 FFT (Fast Fourier Transform)。在 27MHz 时，可在 2.57ms 内完成同样的 FFT 运算。

1.1.3 高精度

在 DSP56000 和 DSP56001 处理器中，单精度 24 位数据传送发生在 24 位总线上，双精度 48 位数据传送发生在链接的 24 位总线上。24 位数据总线提供 144dB 数据动态范围，48 位链接数据总线提供 288dB 数据动态范围。这种数据动态范围比 16 位 DSP 提供的大 50%。数据

动态范围计算如下：

$$\text{以 dB 表示的动态范围} = 20\log_{10}(2^{\text{位数}})$$

$$144\text{dB} = 20\log_{10}(2^{24})$$

$$188\text{dB} = 20\log_{10}(2^{48})$$

$$96\text{dB} = 20\log_{10}(2^{16})$$

DSP56000 和 DSP56001 处理器的数据算术和逻辑单元（Data ALU）都用 24 位乘 24 位的硬件乘法器，和两个 56 位累加器之一连接。例如，这样配置可以执行高达 256 条乘-累加指令，而不会由于截断或舍入而损失任何精度。这种能力可直接用于 FIR（Finite Impulse Response）滤波器高精度和高效率的实现，两个 56 位累加器都提供 336dB 的数据动态范围。

1.1.4 高性能指令集

DSP56000 和 DSP56001 处理器执行共用的指令集。指令集包含 62 个类似微处理器的助记符，它能很容易地从微处理器编程转换为 DSP56000/1 处理器编程。此指令集提供的高性能充分利用了并行结构的优点，同时产生紧致码。例如重复下一指令（REP）的指令可重复 N 次。跟随一条乘-累加（MACR）指令的 REP（n）指令，可使 n 抽头的 FIR 滤波算法紧缩为只有两条指令，并且仅执行 2（n+1）个时钟周期。

1.2 DSP56000ADS 应用开发系统概述

DSP56000ADS 应用开发系统（ADS）是设计、调试和评价基于 DSP56000 系统的硬件工具。如图 1-3 所示，ADS 由三部分组成：

（1）应用开发模板（ADM），它包含 DSP56000 处理器、片外扩展存贮器、接口与控制电路，以及为了联系专用板的几个连接器。

（2）主机接口板插入一个用户所提供主机平台的底板，并经带状电缆接到 ADM。

（3）ADS56000 用户接口软件程序在主机平台上运行并控制 ADM。

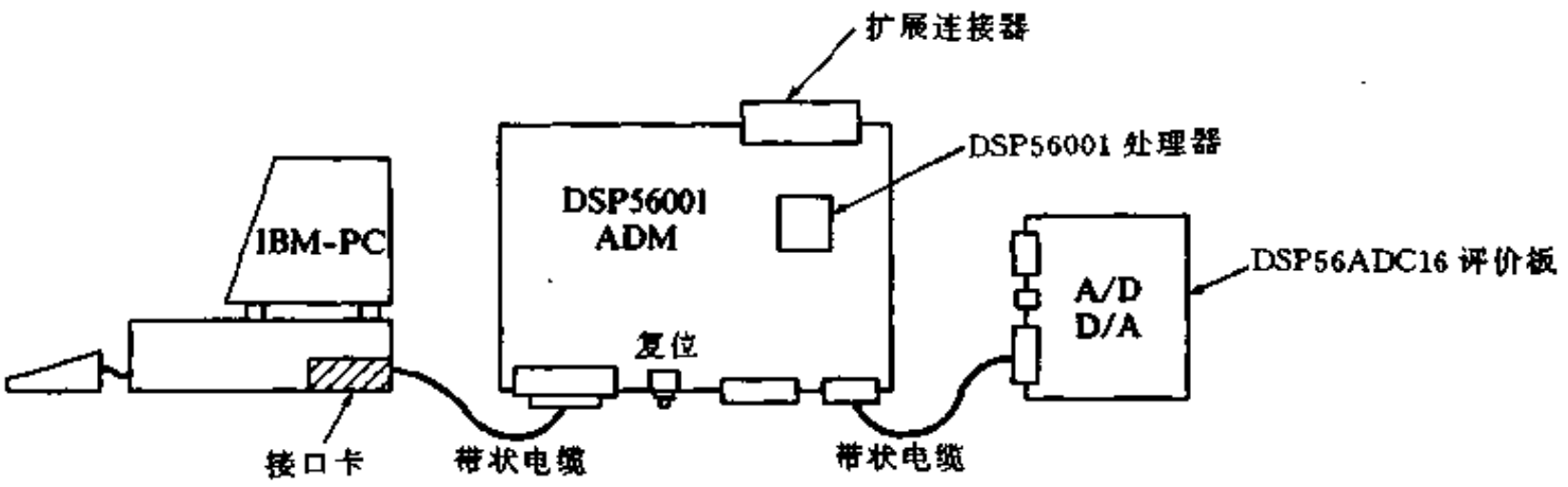


图 1-3 应用开发系统部件

本书中用 IBMPC™在用户和 ADM 之间作为主机平台。所以，要求主机接口板和 IBM PC 的底板兼容，并要求 ADS56000 用户接口软件程序和 MS-DOS 兼容。Motorola 公司可以供应能与 Macintosh™IIPC 或 SUN-3™工作站相兼容的主机接口板和 ADS56000 用户接口软件

程序的产品。

1.3 ADS的建立

1. 关掉PC机电源并拔掉电源插头。

2. 关掉PC机外设的电源（监视器、打印机等），并拔去它们的电源。

3. 在主机接口板上检查跳线JG1和JG2。PC机的I/O地址映像用于外设访问，此外设对标准PC结构是未定义的。主机接口板是工厂设定去对PC I/O地址映像单元从100到200的16进制译码。若那些地址单元已被其他外围板占用，可从地址单元200或300（16进制）处开始改变主机接口板地址译码器。若要从16进制200处开始重新设置地址译码器，只需拔出JG1，去掉JG1的1端和2端上的短路夹，把它改放到JG1的3端和4端上，再牢固地插回JG1即可。若要从16进制300处开始译码，则应把JG1 1端和2端上的短路夹拿掉，并妥为保存以备将来再用。

4. 把“主机接口”板装入PC机的底板。

5. 在“主机接口”板和ADM之间连接带状电缆。

6. 把PC机及其外设插入相应的电源，而后置其开关于通位。

1.4 装配“ADS56000用户接口软件”程序

ADS56000用户接口软件程序可让用户经键盘选择和控制ADS资源，在监视器屏幕上观察菜单并显示ADS所产生的结果。

1.4.1 带两个软盘驱动器和没有硬盘驱动器的PC机

1. 将“DOS”软盘放入驱动器“A”，并引导PC机。

2. 将含有“ADS56000用户接口软件”的“DSP56000ADS用户接口程序放入驱动器“B”。

3. 键入b：ads56000<return>进入“ADS56000用户接口软件”程序。

下面的菜单应显示在屏幕下方：

```
ADMO>
asm break change copy diveice disassemble <space>=more
```

4. 键入pc<return>显示起始地址单元。

屏幕上应显示：

```
PC
PC Interface Card uses PC I/O address100
```

5. 按应用进行第6或第7步。

6. 若在1.3节的第3步中，已改变了“主机接口”板上的地址译码器，在地址单元$200或$300处开始地址译码，则键入pcio $200<return>或pcio $300<return>，以通知

"ADS56000 用户接口软件"程序关于这一改变。进行第 8 步。

7. 如在 1.3 节的第 3 步中，没有改变地址译码器从地址单元 $200 或 $300 处开始地址译码，进行第 8 步。

8. 键入 quit<return>，退出"ADS56000 用户接口软件"程序。

1.4.2 带 1 个硬盘驱动器和 1 个或多个软盘驱动器

为了简化，为硬盘设置驱动器"C"，而 DOS 驻留在 c：\dos 中。

1. 引导 PC 机。

2. 把含有"ADS56000 用户接口软件"程序的"DSP56000ADS 用户接口程序"软盘放驱动器"A"。

3. 键入 md c：\dsptools<return>，在驱动器"C"上产生 dsptools 目录。

4. 键入 md c：\dsptools\ads56000<return>在驱动器"C"上产生 dsptools\ads56000 目录

5. 键入 copy a：c：\dsptools\ads56000<return>把"DSP56000ADS 用户接口程序"软盘的内容复制到驱动器"C"上指定的目录中。

6. 键入 cd c：\dsptools\ads56000<return>，进入此目录。

7. 键入 ads56000<return>，进入"ADS56000 用户接口软件"程序。

以下菜单将出现在屏幕下方：

```
ADMO>
asm break change copy device disassemble<space>=more
```

8. 按需要进行第 9 或第 10 步。

9. 同 1.4.1 节中的第 6 步。进行第 11 步。

10. 同 1.4.1 节中的第 7 步。进行第 11 步。

11. 键入 quit<return>，退出"ADS56000 用户接口软件"程序。

1.5 装配"DSP56000/1 示范软件"程序

DSP56000/1 示范软件程序是一种易于按屏幕菜单提示进行的方法。例如，用户可直接键入本章 1.7 节和 1.8 节提出的命令；或者换一种方法，运行本章所命名的 DSP56000/1 示范软件程序文件，在各种屏幕菜单作出选择，通过 1.7 节和 1.8 节所述命令，按屏幕提示逐步操作。

1.5.1 PC 机带有两个软盘驱动器而没有硬盘

1. 将"DOS"软盘放入驱动器"A"并引导 PC 机。

2. 若第 3、4、5、6 步前面已经完成，直接进行第 4 和第 7 步；否则，进行第 3 步。

3. 从驱动器"A"取出"DOS"软盘并放入驱动器"B"。

4. 将"DSP56000/1 说明软件#N"软盘放入驱动器"A"。

5. 键入 copy b：command.com a：<return>把指定的"DOS"文件拷贝到"DSP56000/1 示范软件#N"软盘上。

6. 对所有“DSP56000/1 示范软件”软盘重复第 4 和第 5 步。

7. 键入 a：chapter1 < return > 或 a：chapter2 < return > 等等，进入所要求的“CHAPTERx”文件。

8. 退出“DSP56000/1 示范软件”程序，由菜单选择“EXIT”命令。

1.5.2 PC 机有一个硬盘驱动器和一个或更多的软盘驱动器

1. 打开 PC 机。

2. 把“DSP56000/1 示范软件#N”软盘 放入驱动器“A”。

3. 键入 a：chapter1 < return > 或 a：chapter2 < return > 等，进入所要求的“CHAPTERx”文件。

4. 退出“DSP56000/1 示范软件”程序，从菜单选择“EXIT”命令。

1.6 “ADS56000 用户接口命令”综合

这里仅包括了本书必须说明的那些命令，关于 ADS56000 命令的详细情况可参阅 Motorola DSP56000ADS 应用开发系统用户手册。ADS56000 用户接口命令（没有任选参数）归纳如表 1－1 所示。

表 1－1 ADS56000 用户接口命令

命 令	说 明	命 令	说 明
ASM	单线装配	LOAD	目的文件装入 ADM 存贮器
BREAK	设置断点	OUTPUT	为 ADM 程序输出打开文件
CHANGE	修正寄存器/存贮器值	LOG	保留命令/全部对话归档
COPY	把存贮块拷贝到其他地方	PATH	为 ADM 设置目录通道
DEVICE	选择缺省的 ADM	PC	设置 PC 接口地址/中断
DISPLAY	显示寄存器/存贮器值	QUIT	退出用户接口程序
DISASSEMBLE	反汇编存贮	RADIX	为命令入口设置缺省基数
EVALUATE	计算	SAVE	保存 ADM 状态/存贮器归档
FORCE	强迫硬件复位或断开	STEP	ADM 的多指令步骤
GO	实时执行 ADM 程序	SYSTEM	执行操作系统命令
HELP	在线帮助文本	TRACE	ADM 的单步指令
INPUT	为 ADM 程序输入打开文件	WAIT	继续工作前的等待

1.7 某些“ADS56000 用户接口命令”的基础

1. 如果想用 DSP56000 示范软件程序包括 1.7 节的题目，按照 1.5.1 节的 1、4 和 7 步或 1.5.2 节的第 1～3 步装入软件。然后由示范软件的菜单选择包括 1.7 节的题目。

2. 如不想用 DSP56000 示范软件程序，则用 1.4.1 节的 1～3 步 或 1.4.2 节的第 1、6 和 7 步装入 ADS56000 用户接口软件程序。

1.7.1 “Help”命令

Help 命令允许用户观察所有的 ADS56000 用户接口命令或在特定命令上得到信息。

1. 键入 Help<return>显示“ADS56000 用户接口命令”全部清单。

显示在屏幕上的应是：

```
--DSP56001 APPLICATION DEVELOPMENT SYSTEM COMMANDS--

ASM  [Dx] [(beginning at) addr] [I(interactive)]
          [assembler_mnemonic]
BREAK  [D0..7] [#bn] [addr|OFF] T [expr]    [H(halt)
                          /In(inc. CNTn)/N(note)/S(show)]
CHANGE  [D0..7][reg[_block]/addr[_block]      (to)
                  [expression]
COPY(from)  [Dx]addr[_block](to)[D0..7]addr
DISPLAY  [D0..7][ON/OFF/ALL/IO][reg[_block/_group]
                   /addr[_block]]...
DISPLAY W(active ADMs)/V(version)
DEVICE D0..7[ON/OFF]
DISASSEMBLE  [D0..7][addr[_block]]
EVALUATE  [Dx][B(binary)/D(decimal)/F(fractional)
                  /H(hexadecimal)]expression
FORCE  [D0..7]R(hardware reset)|B(Return to monitor)
GO  [D0..7][(from)addr/R(reset)][(to break #)#bn]
          [(occurrence) : count]
HELP  [Dx][command|register]
INPUT  [D0..7][#number]OFF/TERM/filename
                  [-rd|-rf|-rh]
LOAD  [D0..7|S(state)] (from) filename
LOG  [D0..7] [OFF] [S(commands)/S(session) [filename]]
OUTPUT  [D0..7] [#unmber] OFF/TERM/filename
                  [-rd|-rf|-rh]
PATH  [D0..7][pathname]
PC(address)  [IO[$100|$200|$300]]
QUIT
RADIX  [D0..7][B(bin)/D(dec)/H(hex)/F(frac)]
                  [reg[_block]/addr[_block]
SAVE  [D0..7]S(state)/addr_block... filename
STEP  [D0..7][count]
SYSTEM  [system_command[parameter_list]]
TRACE  [D0..7][count]
WAIT[count(seconds)]
Macro filename
;comment string entry
Note:Trace/Step/Go commands cause ADM(s) Service
     Requests
```

2. 用“UP ARROW”和“DOWN ARROW”键观察“ADS56000 用户接口命令”。

3. 键入 help display<return>显示全部显示命令清单。

屏幕上应显示：

————————DISPLAY: Display Register or Memory————————

DISPLAY [D0.. 7] [ON/OFF/ALL/IO] [reg [-block/-group]
/addr [-block]]...
DISPLAY W (active ADMs) /V (version)

display
Display all currently enabled registers and memory.

display w
Display the active AMDs.

display v
Display ADS56000 rev. number, date of release, and default ADM monitor rev.

display d2 v
Display ADS56000 rev. number, data of release, and ADM # 2 monitor rev.

————————Register and Memory Display Modes————————

ON Always display the following registers and memory locations.
OFF Never display the following registers and memory locations.

————————Examples————————

display on
Enable all registers and stacked levels for display

display on p: 0.. 20 x: 30.. 40 y: $100
Display enable p memory address block 0 to 20, x memory address block 30 to 40 and y memory address hexadecimal 100.

display off
Disable all display of registers, memory and stacked levels.

display on dsp host scl
Enable display of the DSP56000 programming model registers, the HOST peripheral registers, and the SCI peripheral registers.

————————Register Group Names————————

ALL	all registers.
DSP	all of the DSP56000 programming model registers.
IO	all peripheral register names.
HOST	all registers of the Host peripheral interface.

PORTB	all registers of the Port B interface.
PORTC	all registers of the Port C interface.
SCI	all registers of the SCI peripheral interface.
SSI	all registers of the SSI peripheral interface.
STACK	all current stacked levels.

display p: 0.. 300
Immediate display of p memory addresses 0 through 300.

display on host
Immediate display of all host port registers plus currently enabled registers and memory locations.

display stack
Immediate display of all stacked levels up to the Current stack pointer value. Example: sp=5 shows stacked values from 1 to 5.

4. 用“UP ARROW”和“DOWN ARROW”键观察 display 命令和举例。

5. 键入 help asm<return>，显示全部 asm 命令清单和举例。

屏幕上应显示：

```
--------ASM: Single Line Assembler--------

ASM [Dx] [ (beginning at ) addr] [I (interactive)]
          [assembler_mnemonic]

asm p: $50
Start interactive assembler at program memory address
50 (hex) of default ADM.

asm x: 0 move r0, a1
Assemble single instruction at x memory address 0 of
default ADM.

asm
Start assembler at current program counter value of
default ADM.
asm d3 p: 100
Start assembler at program memory address 100
(decimal) of ADM 3.

asm i
Start assembler at current program counter value of
default ADM.
After instruction is assembled and placed in memory,
trace the instruction.

asm p: $50 i move #1, x0
Assemble a move #1, x0 instruction, place the opcode at
program address 50 hex, and immediately execute the
instruction.
```

6. 用 help 命令考察 go 、break 和 trace 命令，即 help go<return>，help break<return>，help trace<return>。

1.7.2 “Radix” 命令

16 进制常数可在常数前加符号（$）来表示。同样地，10 进制常数前面加（ ‘）符号；2 进制数前面加（%）符号。当 ADS56000 用户接口软件程序进入时，缺省的基数是 10 进制。也就是 10 进制常数可直接进入，前面不用加符号。radix 命令允许改变基数，从而送入常数时就不需加指定的符号。例如，若需送入大量 16 进制常数，radix 命令可用来改基数为 16 进制。但是，这样做以后，需在 10 进制数前加（ ‘），2 进制数前加（%）符号。

1. 键入 radix<return>，显示缺省的基数。

屏幕应显示：

```
The current default radix is decimal
```

2. 键入 radix h<return>，改变缺省基数为 16 进制。

3. 键入 radix<return>，显示新的缺省基数。

屏幕应显示：

```
The current default radix is hexadecimal
```

4. 键入 radix b<return>，基数改为 2 进制。

5. 键入 radix d<return>，基数返回 10 进制。

1.7.3 “display” 命令

display 命令可用来定义要显示的 DSP56000/1 和 ADS 寄存器和存贮器的单元。单元仅受屏幕显示面积的限制. Motorola 定义寄存器组为了显示可用以下显示命令：

all	全部寄存器
dsp	全部编程器的模型寄存器
io	片上外围的全部寄存器
host	片上主机外围的全部寄存器
port b	通用端口 B 的全部寄存器
port c	通用端口 C 的全部寄存器
sci	片上 SCI 外围的全部寄存器
ssi	片上 SSI 外围的全部寄存器

例如，执行命令 display on host ssi，仅显示 DSP56000/1 主机和 SSI 片上外围寄存器。

1. 键入 display<return>，显示所有能显示的有关 DSP56000/1 和 ADS 全部寄存器和/或存贮器单元。

屏幕上显示寄存器符号是：

(1) DSP56000/1 数据 ALU 寄存器

x，x1，x0，y，y1，y0 为输入寄存器

a，a2，a1，a0，b，b2，b1，b0 为累加器寄存器

(2) DSP56000/1 地址产生单元寄存器

r0～r7 为地址寄存器

n0～n7 为偏置寄存器

m0～m7 为修正寄存器

(3) DSP56000/1 程序控制器寄存器

pc 为程序计数寄存器

sr 为状态寄存器

omr 为操作模式寄存器

la 为循环地址寄存器

lc 为循环计数寄存器

sp 为堆栈指针寄存器

ssh 为系统堆栈高寄存器

ssl 为系统堆栈低寄存器

(4) ADS56000 用户接口程序

cnt1～cnt4 为软件计数器寄存器

显示的 DSP56000/1 和 ADS 寄存器和或存贮器单元中的值，在每次执行 display 命令或 DSP56000/1 示范软件时可能不同。最后一次值以黑体字显示。屏幕上应相似于：

```
x= $ 000000000000        y= $ 000000000000
a= $ 00000000000000      b= $ 00000000000000
x1= $ 000000    x0= $ 000000              r7= $ FFFF    n7 = $ FFFF
                                                        m7= $ FFFF
y1= $ 000000    y0= $ 000000              r6= $ FFFF    n6 = $ FFFF
                                                        m6= $ FFFF
a2= $ 00 a1= $ 000000 a0= $ 000000        r5= $ FFFF    n5 = $ FFFF
                                                        m5= $ FFFF
b2= $ 00 b1= $ 000000 b0= $ 000000        r4= $ FFFF    n4 = $ FFFF
                                                        m4= $ FFFF
                                          r3= $ FFFF    n3 = $ FFFF
                                                        m3= $ FFFF
pc= $ 0040   sr= $ 0300   omr= $ 02       r2= $ FFFF    n2 = $ FFFF
                                                        m2= $ FFFF
la= $ 0000   lc= $ FDFD                   r1= $ FFFF    n1 = $ FFFF
                                                        m1= $ FFFF
ssh= $ FFFF   ssl= $ FFFF   sp= $ 0000    r0= $ FFFF    n0 = $ FFFF
                                                        m0= $ FFFF
cnt1=000000   cnt2=000000   cnt3=000000   cnt4=000000
ADM#0   P: $ 0040 FDFFFF   =MACR-Y1, X1, B X: (R7) +, B Y: (R3) +, Y1
```

键入 help lc<return>，决定缩写 lc 的意义。

屏幕应显示：

```
Program Controller Loop Count Register
```

键入 help la<return>，决定缩写 la 的意义。

屏幕应显示：

```
Prongram Controller Loop Address Register
```

2. 键入 display p：0.8<return>，显示 P 存贮器单元 0～8 的值。

屏幕显示应类似于：

```
P：$0000  $FFFFFF  $FDFFDF  $FFFFFF  $FAE3F7
P：$0004  $0BF080  $00E1D6  $0BF080  $00E1D6
P：$0008  $777EFF
ADM#0 P：$0040  FDFFFF      =MACR - Y1，X1，B X：(R7) +，B Y：(R3) +，Y1
```

3. 键入 display p：0..8x：30..40 y：$100<return>，显示 P 存贮器单元 0～8，X 存贮器单元 30～40 和 Y 存贮器单元 $100 的值。

屏幕应显示类似于：

```
P：$0000  $FFFFFF  $FDFFDF  $FFFFFF  $FAE3F7
P：$0004  $0BF080  $00E1D6  $0BF080  $00E1D6
P：$0008  $777EFF
X：$001E  $000000  $000000
X：$0020  $000000  $001001  $000100  $240808
X：$0024  $000000  $000000  $400000  $000800
X：$0028  $000000
Y：$0100  $FA3EDE
ADM#0 P：$0040  FDFFFF      =MACR - Y1，X1，B X：(R7) +，B Y：(R3) +，Y1
```

4. 键入 display on p：0..8x：30..40 y：$100<return>，增加指定的存贮器单元到显示上。

5. 键入 display<return>，表示所有被显示的 DSP56000/1 和 ADS 寄存器和/或存贮器单元的值。

屏幕显示应类似于：

```
x= $000000000000          y= $000000000000
a= $00000000000000        b= $00000000000000
x1= $000000    x0= $000000          r7= $FFFF    n7 = $FFFF
                                                 m7= $FFFF
y1= $000000    y0= $000000          r6= $FFFF    n6 = $FFFF
                                                 m6= $FFFF
a2= $00 a1= $000000   a0= $000000   r5= $FFFF    n5 = $FFFF
                                                 m5= $FFFF
b2= $00 b1= $000000   b0= $000000   r4= $FFFF    n4 = $FFFF
                                                 m4= $FFFF
                                    r3= $FFFF    n3 = $FFFF
                                                 m3= $FFFF
pc= $0040  sr= $0300  omr= $02      r2= $FFFF    n2 = $FFFF
                                                 m2= $FFFF
la= $0000  lc= $FDFD                r1= $FFFF    n1 = $FFFF
                                                 m1= $FFFF
ssh= $FFFF  ssl= $FFFF  sp= $0000   r0= $FFFF    n0 = $FFFF
                                                 m0= $FFFF
cnt1=000000  cnt2=000000  cnt3=000000  cnt4=000000
P：$0000  $FFFFFF  $FDFFDF  $FFFFFF  FAE3F7
P：$0004  $0BF080  00E1D6   $0BF080  00E1D6
P：$0008  $777EFF
```

```
X: $001E  $000000    $000000
X: $0020  $000000    $001001    $000100    $240808
X: $0024  $000000    $000000    $400000    $000800
X: $0028  $FFFFFF    $0100
ADM#0  P: $0040  FDFFFF  =MACR-Y1, X1, B X: (R7)+, B Y: (R3)+, Y1
```

6. 键入 display pc sr<return>，显示 DSP56000/1 的程序计数器（PC）和状态（SR）寄存器的值。

屏幕显示应类似于：

```
pc= $0040    sr=   $0300
ADM#0 P: $0040 FDFFFF  =MACR-Y1, X1, X: (R7)+, B Y: (R3)+, Y1
```

7. 键入 display off<return>，全部显示终止。

8. 键入 display on<return>，使能显示，例如 DS56000/1 的编程器的模型寄存器。

1.7.4 “Change”命令

Change 命令可用以考察或修正寄存器和/或存贮器单元的值。

1. 键入 change x: $55<return>，显示 X 存贮器单元 $55 的值，并得到一新值的提示。

2. 键入 $8<return>，改变 X 存贮器中单元 $55 的值到单元 $8。

屏幕显示应为：

```
change X: $0055 $000008 ; Dec: 8 Fract: 0.0000010
```

3. 键入 $12<return>，改变下一个 X 存贮器单元（$56）的值到 $12。

屏幕显示应为：

```
change X: $0056 $000012 ; Dec: 18 Fract: 0.0000021
```

4. 键入<escape>，退出 change 命令。

5. 键入 display x: $55..$56<return>显示 X 存贮器单元为 $55 和 $56 的值。

屏幕显示应类似于：

```
X: $0055 $000008 $000012
ADM#0 P: $0040 FDFFFF    =MACR-Y1, X1, B X: (R7)+, B Y: (R3)+, Y1
```

6. 键入 change pc $10<return>，改变 DSP56000/1 的程序计数器（PC）寄存器的值到 $10。

7. 键入 display pc<return>，显示 PC 的新值（$10）。

屏幕显示应类似于：

```
pc= $0010
ADM#0 P: $0010  EFFFFE        =DC $EFFFFE
```

8. 键入 display p: $14<return>，显示 P 存贮器单元 $14 的值。

屏幕显示应类似于：

```
p：  $0014  $FFFFFF
ADM#0 P：  $0010 EFFFFE          =DC  $EFFFFE
```

9. 键入 change p： $14 $123<return>改变 P 存贮器单元 $14 的值到单元 $123。

10. 键入 display p： $14<return>，显示位于 P 存贮器 $14 的新值（$123)。

屏幕显示应类似于：

```
p：  $0014  $000123
ADM#0 P：  $0010 EFFFFE          =DC  $EFFFFE
```

11. 键入 change r0.. r7 0 x： $08 $134 pc200<return>，改变 DSP56000/1 地址寄存器 R0～R7 的值到 0，改变 $08x 存贮器的值到 $134，以及改变 DSP56000/1 程序计数器 (PC) 的值到 200。

12. 键入 display on x： $08<return>，把 X 存贮器 $08 值加到显示上。

13. 键入 display<return>，表示执行以上 change 命令的结果。

屏幕显示应类似于：

```
x= $000000000000        y= $000000000000
a= $00000000000000      b= $00000000000000
x1= $000000     x0= $000000                r7= $0000     n7 = $FFFF
                                                         m7= $FFFF
y1= $000000     y0= $000000                r6= $0000     n6 = $FFFF
                                                         m6= $FFFF
a2= $00   a1= $000000   a0= $000000        r5= $0000     n5 = $FFFF
                                                         m5= $FFFF
b2= $00   b1= $000000   b0= $000000        r4= $0000     n4 = $FFFF
                                                         m4= $FFFF
                                           r3= $0000     n3 = $FFFF
                                                         m3= $FFFF
pc= $00C8   sr= $0300   omr= $02           r2= $0000     n2 = $FFFF
                                                         m2= $FFFF
la= $0000   lc= $FDFD                      r1= $0000     n1 = $FFFF
                                                         m1= $FFFF
ssh= $FFFF   ssl= $FFFF   sp= $0000        r0= $0000     n0 = $FFFF
                                                         m0= $FFFF
cnt1=000000   cnt2=000000   cnt3=000000   cnt4=000000
X：  $0008  $000134
ADM#0   P：  $00C8   FFFEFF   =DC  $FFFEFF
```

1.7.5 “Copy”命令

Copy 命令可用于将一个存贮器块的值拷贝到另一个存贮器块。

1. 键入 change p：0..10 $ff<return>，改变位于 0～10（$0～$a)P 存贮器的值到 $ff。

2. 键入 display p： 0..10 p： 100..110<return>，显示位于 0～10（$0－$a）和 100～110（$64～$6e）P 存贮器的值。

屏幕显示应类似于：

```
P: $0C00   $0000FF   $0000FF   $0000FF   $0000FF
P: $0004   $0000FF   $0000FF   $0000FF   $0000FF
P: $0008   $0000FF   S0000FF   $0000FF
P: $0064   $FF5FFF   $FF7FFF   $FFFFFF   $7FFFFF
P: $0068   $FFF77F   $7FFFFD   $FBFBEF   $5FFDCF
P: $006C   $FFFDFF   $FFFFFF   $FFFFFF
ADM#0  P: $00C8 FFFEFF          =DC $FFFEFF
```

3. 键入 copy p：0..10　p：100<return>，把位于 0～10（$0～$a）的 P 存贮器的值拷贝到由 100（$64）开始的 11 个连续的 P 存贮器。

4. 键入 display p：0..10　p：100..110<return>，表示 0～10（$0～$a）的 P 存贮器的值（$ff）已拷贝到 100～110（$64～6c）的 P 存贮器。

屏幕显示应类似于：

```
P: $0000   $0000FF   $0000FF   $0000FF   $0000FF
P: $0004   $0000FF   $0000FF   $0000FF   $0000FF
P: $0008   $0000FF   $0000FF   $0000FF
P: $0064   $0000FF   $0000FF   $0000FF   $0000FF
P: $0068   $0000FF   $0000FF   $0000FF   $0000FF
P: $006C   $0000FF   $0000FF   $0000FF
ADM#0  P: $00C8 FFFEFF          =DC $FFFEFF
```

5. 键入 change x：0..10　$aa<return>，改变 0～10（$0～$a）的 X 存贮器的值到 $aa。

6. 键入 display x：0..10　p：0..10<return>，显示 0～10 的 X 存贮器和 0～10 的 P 存贮器的值。

屏幕显示应类似于：

```
P: $0000   $0000FF   $0000FF   $0000FF   $0000FF
P: $0004   $0000FF   $0000FF   $0000FF   $0000FF
P: $0008   $0000FF   $0000FF   $0000FF
X: $0000   $0000AA   $0000AA   $0000AA   $0000AA
X: $0004   $0000AA   $0000AA   $0000AA   $0000AA
X: $0008   $0000AA   $0000AA   $0000AA
ADM#0  P: $00C8 FFFEFF          =DC $FFFEFF
```

7. 键入 copy x：0#11　p：0<return>，把从 0 开始的连续 11 个 X 存贮器的值拷贝到从 0 开始的 11 个连续的 P 存贮器。

8. 键入 display　x：0..10　p：0..10<return>，表示位于 0～10 的 X 存贮器中的值（$aa）已拷贝到 0～10 的 P 存贮器。

屏幕显示应类似于：

```
P: $0000   $0000AA   $0000AA   $0000AA   $0000AA
P: $0004   $0000AA   $0000AA   $0000AA   $0000AA
P: $0008   $0000AA   $0000AA   $0000AA
X: $0000   $0000AA   $0000AA   $0000AA   $0000AA
X: $0004   $0000AA   $0000AA   $0000AA   $0000AA
X: $0008   $0000AA   $0000AA   $0000AA
ADM#0  P: $00C8 FFFEFF          =DC $FFFEFF
```

9. 键入 quit<return>，退出“ADS56000 用户接口软件”程序。

1.8 ADS 用于编程 DSP56000/1

1. 若想用 DSP56000 示范软件程序去包括 1.8 节的内容，则按照 1.5.1 节的第 1、4、7 步或 1.5.2 节的第 1～3 步装入此软件。然后从示范软件的菜单中选择包括 1.8 节的内容。.

2. 若不想用 DSP56000 示范软件程序，则按照 1.4.1 节的第 1～3 步或 1.4.2 节的第 1、6、7 步装入 ADS56000 用户接口软件程序。

1.8.1 输入、汇编和反汇编已封装的 DSP56000/1 程序

ADS56000 用户接口软件程序有一单线的、相互作用的汇编程序，它接受 DSP56000/1 汇编语言助记符。可用来输入或编辑驻留在存贮器中的 DSP56000/1 目的码程序。

1. 键入 asm p:$217<return>，在 P 存贮器单元 217 处开始汇编处理。

2. 键入：move x:0300,b<return>

```
move b,y:(r5)<return>
clr b x:(r0)+,x0 y:(r5)+,y0<return>
mac -x0,y0,b x:(r0)+,x0 y:(r5)+,y0<return>
mac x0,y0,b y:(r5),y0<return>
mac -x0,y0 b x:(r0)+y:(r4)+,y0<return>
macr x0,y0,b<return>
move b,y:(r4)-<return>
move b,x:301<return>
<escape>
```

3. 键入 disassemble p:$217.. $220<return>，反汇编 P 存贮器单元 $217～$220，并显示最后的汇编语言助记符。

屏幕应显示：

```
ADM#0 P: $0217 57F000 00012C =MOVE X:>$12C, B
ADM#0 P: $0219 5F6500        =MOVE B, Y:(R5)
ADM#0 P: $021A F0B81B        =CLR B X:(R0) +, X0 Y:(R5) +, Y0
ADM#0 P: $021B F0B8DE        =MAC -Y0, X0, B X:(R0) +, X0 Y:(R5) +, Y0
ADM#0 P: $021C 4EE5DA        =MAC Y0, X0, B Y:(R5), Y0
ADM#0 P: $021D F098DE        =MAC -Y0, X0, B X:(R0) +, X0 Y:(R4) +, Y0
ADM#0 P: $021E 2000DB        =MACR Y0, X0, B
ADM#0 P: $021F 5F5400        =MOVE B, Y:(R4) -
ADM#0 P: $0220 57F000 00012D =MOVE X:>$12D, B
```

1.8.2 保存和装入或重装入已封装的程序

在 1.8.1 节送入的 DSP56000/1 程序可命名为 prog1，并相应地用 PC 的软、硬盘驱动器存入软、硬盘。然后 prog1 可重装入 ADS。

1. 键入 path a:\<return>设置到软盘驱动器“A”目录的通路。

2. 键入 save p：$217..$220 prog1.lod<return>，存 prog1. 在驱动器“A”的软盘上。

P 存贮块 $217～$220（prog1）存在由 .lod 指示的目的模型格式（OMF）中扩展到文件名 prog1。如 prog1 被用作没有 .lod 扩展的文件名，save 命令将自动附加 .log 扩展到 prog1 文件名。用 OMF 可将 prog1.lod 重装入 ADS。如 prog1.lod 以前已存入所选目的中，则 ADS56000 用户接口软件程序将提示进入 0、1 或 2 单元，以相应选择附加、替换或中断。一旦存入了 prog1. lod，就可用字处理器程序对它调用、编辑和重存。

3. 键入 load prog1. lod<return>，把 prog1.lod 重新装入 ADS。

4. 键入 disassemble p：$217..$220<return>，反汇编和显示 P 存贮器单元 $217～$220 的值。

屏幕应显示：

```
ADM#0 P:$0217 57F000 00012C =MOVE X:>$12C,B
ADM#0 P:$0219 5F6500        =MOVE B,Y:(R5)
ADM#0 P:$021A F0B81B        =CLR B X:(R0)+,X0 Y:(R5)+,Y0
ADM#0 P:$021B F0B8DE        =MAC  -Y0,X0,B X:(R0)+,X0 Y:(R5)+,Y0
ADM#0 P:$021C 4EE5DA        =MAC  Y0,X0,B Y:(R5),Y0
ADM#0 P:$021D F098DE        =MAC  -Y0,X0,B X:(R0)+,X0 Y:(R4)+,Y0
ADM#0 P:$021E 2000DB        =MACR Y0,X0,B
ADM#0 P:$021F 5F5400        =MOVE B,Y:(R4)-
ADM#0 P:$0220 57F000 00012D =MOVE X:>$12D,B
```

1.8.3 测试和执行已封装的 DSP56000/1 程序

为了测试执行中的 DSP56000/1 程序，用户指定的寄存器和/或存贮器的内容可用 break 和 trace 命令显示。

1. 键入 break#1 p：$219 s<return>，当程序执行时，在 $219 P 存贮器的 DSP56000/1 程序指令到达时，表示可显示寄存器和/或存贮器的值。

2. 键入 break#2 p：21f s h<return>，与 1 相同，仅 $219 改为 $21f。

3. 键入 break<return>，显示允许的断点。

屏幕应显示：

```
ADM 0  Break #1 p:  $219 s
ADM 0  Break #2 p:  $21f s h
```

4. 键入 go $217<return>，在 P 存贮器的 $217 开始程序执行。

在程序执行中，当到达断点单元 $219 和 $21f 时，显示由显示器使能的寄存器和/或存贮器单元。屏幕上显示如下所示。记住，UP ARROW 和 DOWN ARROW 键可用来查对。

```
ADM#0 break #1
x= $000000000000      y= $000000000000
a= $00000000000000    b= $FF8BFBAF000000
x1= $000000   x0= $000000          r7= $0000   n7 = $FFFF
                                               m7= $FFFF
y1= $000000   y0= $000000          r6= $0000   n6 = $FFFF
                                               m6= $FFFF
a2= $00  a1= $000000  a0= $000000  r5= $0000   n5 = $FFFF
```

```
                                                        m5= $ FFFF
b2= $ FF   b1= $ 8BFBAF   b0= $ 000000   r4= $ 0000     n4 = $ FFFF
                                                        m4= $ FFFF
                                          r3= $ 0000    n3 = $ FFFF
                                                        m3= $ FFFF
pc= $ 0219   sr= $ 0300   omr= $ 02       r2= $ 0000    n2 = $ FFFF
                                                        m2= $ FFFF
la= $ 0000   lc= $ FDFD                   r1= $ 0000    n1 = $ FFFF
                                                        m1= $ FFFF
ssh= $ FFFF   ssl= $ FFFF   sp= $ 0000    r0= $ 0000    n0 = $ FFFF
                                                        m0= $ FFFF
cnt1=000000   cnt2=000000   cnt3=000000   cnt4=000000
ADM#0 P: $ 0219   5F6500              =MOVE B, Y: (R5)
wait
ADM#0 break #2
x= $ 000000000000        y= $ 000000000000
a= $ 00000000000000      b= $ FF8BFBAF000000
x1= $ 000000      x0= $ 000000            r7= $ 0000    n7 = $ FFFF
                                                        m7= $ FFFF
y1= $ 000000      y0= $ 000000            r6= $ 0000    n6 = $ FFFF
                                                        m6= $ FFFF
a2= $ 00   a1= $ 000000   a0= $ 000000    r5= $ 0000    n5 = $ FFFF
                                                        m5= $ FFFF
b2= $ FF   b1= $ 8BFBAF   b0= $ 000000   r4= $ 0000     n4 = $ FFFF
                                                        m4= $ FFFF
                                          r3= $ 0000    n3 = $ FFFF
                                                        m3= $ FFFF
pc= $ 0219   sr= $ 0300   omr= $ 02       r2= $ 0000    n2 = $ FFFF
                                                        m2= $ FFFF
la= $ 0000   lc= $ FDFD                   r1= $ 0000    n1 = $ FFFF
                                                        m1= $ FFFF
ssh= $ FFFF   ssl= $ FFFF   sp= $ 0000    r0= $ 0000    n0 = $ FFFF
                                                        m0= $ FFFF
cnt1=000000   cnt2=000000   cnt3=000000   cnt4=000000
ADM#0   P: $ 021F   5F5400            =MOVE B, Y: (R4) —
```

5. 键入 break #2 off<return>，除去断点 #2。

6. 键入 go $ 217 #1<return>，在 P 存贮器 $ 217 开始执行程序。当到达断点 #1 时，显示所允许的寄存器和/或存贮器单元，并当到达断点 #1 时，停止程序执行。

屏幕显示应类似于：

```
ADM#0 break #1
x= $ 0000000000AA        y= $ 0000008BFBAF
a= $ 00000000000000      b= $ FF8BFBAF000000
x1= $ 000000      x0= $ 0000AA            r7= $ 0000    n7 = $ FFFF
                                                        m7= $ FFFF
y1= $ 000000      y0= $ 8BFBAF            r6= $ 0000    n6 = $ FFFF
                                                        m6= $ FFFF
a2= $ 00   a1= $ 000000   a0= $ 000000    r5= $ 0002    n5 = $ FFFF
                                                        m5= $ FFFF
```

```
b2= $ FF   b1= $ 8BFBAF   b0= $ 000000    r4= $ 0000    n4 = $ FFFF
                                                        m4= $ FFFF
                                          r3= $ 0000    n3 = $ FFFF
                                                        m3= $ FFFF
pc= $ 0219   sr= $ 0300     omr= $ 02     r2= $ 0000    n2 = $ FFFF
                                                        m2= $ FFFF
la= $ 0000   lc= $ FDFD                   r1= $ 0000    n1 = $ FFFF
                                                        m1= $ FFFF
ssh= $ FFFF   ssl= $ FFFF   sp= $ 0000    r0= $ 0003    n0 = $ FFFF
                                                        m0= $ FFFF
cnt1=000000   cnt2=000000   cnt3=000000   cnt4=000000
ADM#0   P:  $ 0219 000006              =SWI
```

7. 键入 break#1<return>，取消断点#1。

8. 键入 change pc $ 217<rerurn>，把值 $ 217 放入 DSP56000/1 的程序计数器（PC）中。

9. 键入 trace<return>，执行一条指令，并表示可显示寄存器和/或存贮器单元的值。

屏幕显示应类似于：

```
ADM#0 TRACE COUNT=0
x= $ 0000000000AA          y= $ 0000008BFBAF
a= $ 00000000000000        b= $ FF8BFBAF000000
x1= $ 000000     x0= $ 0000AA              r7= $ 0000    n7 = $ FFFF
                                                        m7= $ FFFF
y1= $ 000000     y0= $ 8BFBAF              r6= $ 0000    n6 = $ FFFF
                                                        m6= $ FFFF
a2= $ 00   a1= $ 000000   a0= $ 000000     r5= $ 0002    n5 = $ FFFF
                                                        m5= $ FFFF
b2= $ FF   b1= $ 8BFBAF   b0= $ 000000     r4= $ 0000    n4 = $ FFFF
                                                        m4= $ FFFF
                                           r3= $ 0000    n3 = $ FFFF
                                                        m3= $ FFFF
pc= $ 0219   sr= $ 0220   omr= $ 02        r2= $ 0000    n2 = $ FFFF
                                                        m2= $ FFFF
la= $ 0000   lc= $ FDFD                    r1= $ 0000    n1 = $ FFFF
                                                        m1= $ FFFF
ssh= $ FFFF ssl= $ FFFF sp= $ 0000         r0= $ 0003    n0 = $ FFFF
                                                        m0= $ FFFF
cnt1=000000   cnt2=000000   cnt3=000000   cnt4=000000
ADM#0   P:  $ 0219 000006              =SWI
```

10. 键入 trace 3<return>，执行下 3 条指令，并显示从执行每条指令遇到的使能寄存器和/或存贮器单元的值。

屏幕显示应类似于：

```
ADM#0  TRACE  COUNT=2
x= $ 0000000000AA          y= $ 0000008BFBAF
a= $ 00000000000000        b= $ FF8BFBAF000000
x1= $ 000000     x0= $ 0000AA              r7= $ 0000    n7 = $ FFFF
                                                        m7= $ FFFF
```

```
y1= $ 000000     y0= $ 8BFBAF                r6= $ 0000    n6 = $ FFFF
                                                           m6= $ FFFF
a2= $ 00  a1= $ 000000  a0= $ 000000         r5= $ 0002    n5 = $ FFFF
                                                           m5= $ FFFF
b2= $ FF  b1= $ 8BFBAF  b0= $ 000000         r4= $ 0000    n4 = $ FFFF
                                                           m4= $ FFFF
                                             r3= $ 0000    n3 = $ FFFF
                                                           m3= $ FFFF
pc= $ 021A  sr= $ 0220  omr= $ 02            r2= $ 0000    n2 = $ FFFF
                                                           m2= $ FFFF
la= $ 0000  lc= $ FDFD                       r1= $ 0000    n1 = $ FFFF
                                                           m1= $ FFFF
ssh= $ FFFF  ssl= $ FFFF  sp= $ 0000         r0= $ 0003    n0 = $ FFFF
                                                           m0= $ FFFF
cnt1=000000  cnt2=000000  cnt3=000000  cnt4=000000
ADM#0  P: $ 021A F0B81B          =CLR B X:(R0)+,X0 Y:(R5)+,Y0

wait
ADM#0 TRACE COUNT=1
x= $ 000000000000      y= $ 000000000000
a= $ 00000000000000    b= $ 00000000000000
x1= $ 000000      x0= $ 000000      r7= $ 0000    n7 = $ FFFF
                                                  m7= $ FFFF
y1= $ 000000      y0= $ 000000      r6= $ 0000    n6 = $ FFFF
                                                  m6= $ FFFF
a2= $ 00  a1= $ 000000  a0= $ 000000  r5= $ 0003  n5 = $ FFFF
                                                  m5= $ FFFF
b2= $ 00  b1= $ 000000  b0= $ 000000  r4= $ 0000  n4 = $ FFFF
                                                  m4= $ FFFF
                                      r3= $ 0000  n3 = $ FFFF
                                                  m3= $ FFFF
pc= $ 021B  sr= $ 0214  omr= $ 02     r2= $ 0000  n2 = $ FFFF
                                                  m2= $ FFFF
la= $ 0000  lc= $ FDFD                r1= $ 0000  n1 = $ FFFF
                                                  m1= $ FFFF
ssh= $ FFFF  ssl= $ FFFF  sp= $ 0000  r0= $ 0004  n0 = $ FFFF
                                                  m0= $ FFFF
cnt1=000000  cnt2=000000  cnt3=000000  cnt4=000000
ADM#0 P: $ 021B F0B8DE          =MAC -Y0,X0,B X:(R0)+,X0
                                Y:(R5)+,Y0
wait
ADM#0 TRACE COUNT=0
x= $ 0000000000AA      y= $ 000000000000
a= $ 00000000000000    b= $ 00000000000000
x1= $ 000000      x0= $ 0000AA      r7= $ 0000    n7 = $ FFFF
                                                  m7= $ FFFF
y1= $ 000000      y0= $ 000000      r6= $ 0000    n6 = $ FFFF
                                                  m6= $ FFFF
a2= $ 00  a1= $ 000000  a0= $ 000000  r5= $ 0004  n5 = $ FFFF
                                                  m5= $ FFFF
b2= $ 00  b1= $ 000000  b0= $ 000000  r4= $ 0000  n4 = $ FFFF
                                                  m4= $ FFFF
                                      r3= $ 0000  n3 = $ FFFF
                                                  m3= $ FFFF
```

```
pc= $021C   sr= $0214   omr= $02          r2= $0000     n2 = $FFFF
                                                        m2= $FFFF
la= $0000   lc= $FDFD                     r1= $0000     n1 = $FFFF
                                                        m1= $FFFF
ssh= $FFFF   ssl= $FFFF   sp= $0000       r0= $0005     n0 = $FFFF
                                                        m0= $FFFF
cnt1=000000   cnt2=000000   cnt3=000000   cnt4=000000
ADM#0 P: $021C 4EE5DA              =MAC Y0,X0,B Y:(R5)Y0
```

11. 键入 quit<return>，退出“ADS56000 用户接口软件”程序。

第二章　介绍 DSP56000CLAS 汇编程序/仿真程序软件包

DSP56000CLAS 汇编程序/仿真程序软件包是 DSP56000/1 程序（算法）开发工具，由以下几部分组成：

1. ASM56000 可重置的宏交叉汇编程序。

2. LNK56000 链接程序。

3. LIB56000 库管理程序。

4. SIM56000 多 DSP56000/1 逐步仿真程序。

DSP56000CLAS 软件包由 Motorola 以主机指定形式可在 IMBPC、Macintosh PC、sun－3 工作站或 VAX™/VMS 计算机上运行。

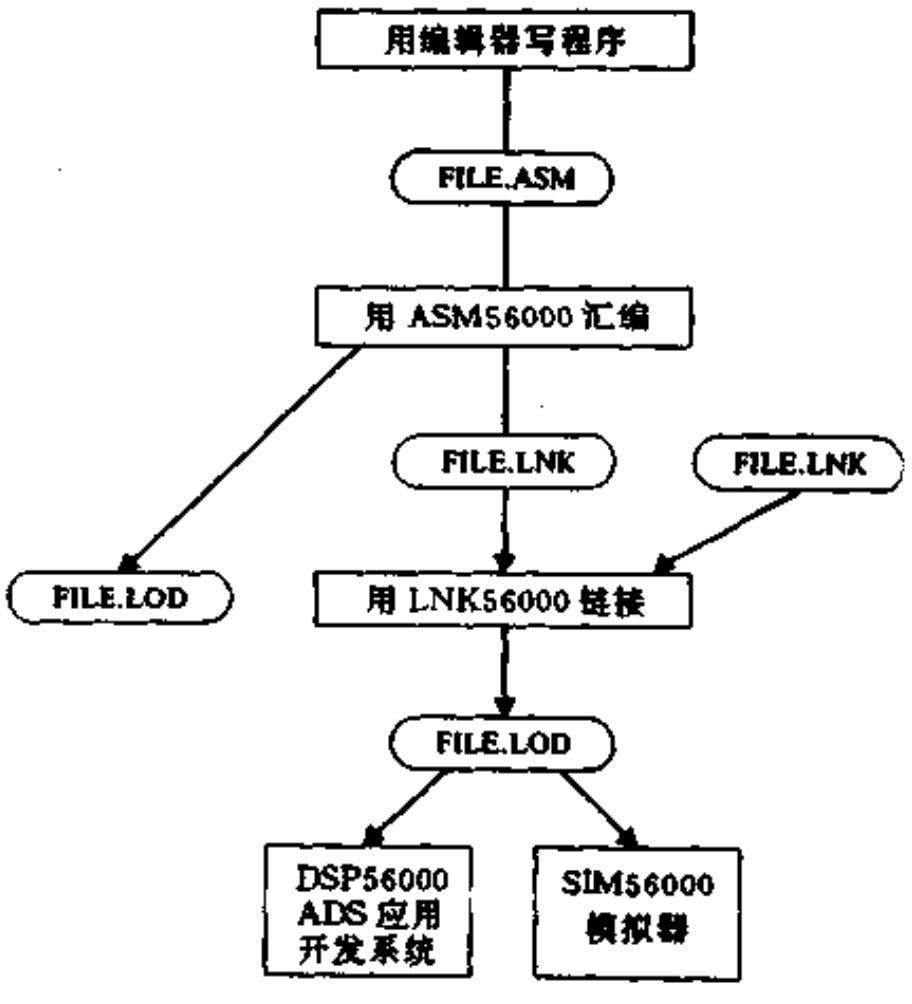


图 2－1　DSP56000/1 程序开发过程

本章中，DSP56000CLAS 软件包和 IBM PC 行编辑（EDLIN）用来开发已封装的 DSP56000/1 程序和宏指令。用 EDLIN 进入已封装的 DSP56000/1 程序和宏指令的 DSP56000/1 汇编语言源语句去产生每个程序或宏指令的汇编源语句文件。其次，ASM56000 浮动宏交叉汇编程序用来对每个程序或宏指令汇编程序源语句的文件转换为 DSP56000/1 目的码语句的浮动文件。LNK56000 链接程序对每个程序或宏指令由汇编程序产生的浮动目的码进行处理，以产生绝对装入文件，此文件可直接装入 ADS 进行仿真执行和测试，也可以送入 SIM56000 程序开发过程。换言之，DSP56000/1 示范软件程序可用于本章所选论题。

2.1　ASM56000 汇编程序软件程序的安装

2.1.1　PC 有二个软盘驱动器而没有硬盘驱动器

1. 把“DOS”软盘放入驱动器“A”并引导 PC 机

2. 把包含“ADS56000 汇编程序软件”程序的“DSP56000 宏交叉汇编程序”软盘放入驱动器“B”。

3. 键入 b：asm56000<return>。

屏幕显示应类似于：

```
Motorola DSP56000 Macro Cross Assembler Version 3.0
(C) Copyright Motorola, Int. 1987, 1988, 1989.
All rights reserved.
Usage: ASM 56000 [-A] [-B [<lodfil>]] [-D<symbol> <string>]
[-F<argfil>] [-I<ipath>] [-L [ldygil>]] [-M<mpath>]
[-O<opt> [, <oppt>...]] [-V] <srcfil>...
where:
    lodfil  - optional load file name
    symbol - user - defined symbol
    string  - string associated with symbol
    ipath   - include file directory path
    lstfil  - optional list file name
    mpath  - macro library directory path
    opt     - assembler option
    srcfil  - assembler source file (s)
```

2.1.2 PC 有一个硬盘驱动器和一个或更多的软盘驱动器。

为简单起见，设驱动器" C" 为硬盘设置，而 DOS 驻留在 C：\DOS 目录中。

1. 引导 PC 机。

2. 把含有“ASM56000 汇编程序软件”程序的“DSP56000 宏交叉汇编程序”软盘放入驱动器“A”。

3. 键入 md c：\dsptools<return>，在驱动器“C”中产生 dsptools 目录。

4. 键入 md c：\dsptools\asm56000<return>在驱动器“C”上产生 dsptool\asm56000 目录。

5. 键入 copy a：c：\dsptools\asm56000<return>，把" DSP56000 宏交叉汇编程序”软盘的内容拷贝到驱动器“C”指定的目录。

6. 键入 cd c：\dsptools\\asm56000<return>，送入目录。

7. 键入 asm56000<return>。

屏幕显示类似 2.1.1 节第三步。

2.2 安装“LNK56000 链接程序软件”和“LIB56000 库管理程序软件”程序

2.2.1 PC 机有两个软盘驱动器而没有硬盘驱动器

1. 把“DOS”软盘放入驱动器“A”并引导 PC 机。

2. 把含有“LNK56000 链接程序软件”和“LIB56000 库管理程序软件”程序的 DSP56000 链接程序/库管理程序”的软盘放入驱动器“B”。

3. 键入 b：lnk56000<return>。

屏幕显示类似于：

```
Motorola DSP56000 Cross Linker Version 3.0
(C) Copyright MOtorola, Inc. 1987, 1988, 1989.
All rights reserved.
Usage: LNK56000 [-B [<lodfil>]] [-D] [-F<argfil>]
[-L<library>] [-M [<map>]] [-O<mem> [<ctr>] [<map>]: <origin>]
[-R [<memfil>]] [-V] [<lnkfil>...
where:
    lodfil  - optional load file name
    argfil  - command line argument file name
    library - library file name
    mapfil  - optional load map file name
    mem     - memory space (X, Y, L, P)
    ctr     - location counter (L, H)
    map     - memory mapping (I, E, B)
    origin  - memory space origin
    memfil  - optional memory map file name
    lnkfil  - link input file name
```

2.2.2 PC 有一个硬盘驱动器和一个或更多的软盘驱动器

1. 引导 PC 机。

2. 把含有“LNK56000 链接程序软件”和“LIB56000 库管理程序软件”程序的 DSP56000 链接程序/库管理程序”软盘放入驱动器“A”。

3. 键入 md c:\dsptools\lnk56000<return>，在驱动器“C”上形成 dsptools\lnk 目录。dsptools 目录在 2.1、2 节中形成。

4. 键入 copy a: c: \dsptools\lnk56000<return>，把" DSP56000" 软盘的内容拷贝到驱动器“C”指定的目录中。

5. 键入 cd c: \dsptools'\lnk56000<return>，进入此目录。

6. 键入 lnk56000<return>。

屏幕显示类似于 2.2.1 节第 3 步。

2.3 安装“SIM56000 仿真程序软件”程序

步骤同 2.2 节（略）。

2.4 程序源语句格式

ASM56000 汇编软件程序允许程序源语句用于 5 种字段。字段被一个或多个空格或标记分开。这 5 种字段是：

1. 标号字段

标号字段表现为源语句第一字段，可取以下格式之一。

(1) 空格或标记作为标号字段的第一个字符，表示此字段是空，而且源语句没有标号。

例：<TAB>CLR<TAB>B<return>

(2) 字母字符作为标号字段的第一个字符，表示源语句包含称为标号的符号。

例：LOOP<TAB>MOVE<TAB>B，L：(RO) <RETURN>

(3) 下划线（_）作为标号字段的第一个字符，表示标号是“局部的”。

例：—ENDP<RETURN>

2. 操作字段

操作字段出现在标号字段之后，并且在其前至少有一空格或标记。操作字段的人口可能是下述三种型式之一：

(1) 操作码：DSP56000/1 指令助记的，它由 ASM56000 汇编程序识别。

例：ENTRY<TAB>ADD B，A<RETURN>

(2) 直接的：由 ASM56000 汇编程序识别的和用于控制汇编程序过程本身的操作码。

例：CNST<TAB>EQU $5<RETURN>

(3) 宏调用：调用一个以前指定的宏指令使其插进来代替宏调用。

3. 操作数字段

操作数字段的说明取决于操作字段的内容。如果有的话，操作数字段必须跟随操作字段，并且必须在其前至少有一空格或标记，操作数字段可能包含符号、说明，或符号和说明的组合并由逗号分开。

例：ENTRY<TAB>ADD B，A<RETURN>

4. 数据变换字段#1 和#2

大多数 DSP56000/1 数据 ALU 操作码可以允许附加一或两种数据变换操作与操作码本身的执行同时进行。数据变换字段#1 指定 一种数据变换操作；数据变换字段#2 指定第二种数据变换操作。数据变换字段操作由被逗点分开（没有嵌人空格）的两个寻址方式操作数所指定。数据变换字段必须至少被 一个空格或标记和前面字段分开。以下例子表示这样的数据变换规格：

例：LOOP<TAB>MOVE<TAB>Y：- (R0) Y0<RETURN>

以下例子表示用数据变换字段#1 和#2 的数据变换规格：

例：<TAB>RND<TAB>A<TAB>X：(R0) +X0<TAB>Y：(R4，+，YO<RETURN>

5. 注解字段

由任何字符组成的注解字段前面加一分号（；）。亦可前面加空格。

例：;PROGRAM TO CLEAR THE RAM IN X and Y MEMORY <RETURN)

<TAB>CLR<TAB>B<TAB>;USED TO CLEAR LONG MEMORY <RETURN>

2.5 送人和编辑已封装的 DSP56000/1 程序

DSP56000/1 程序可用任何有效的字处理程序送人和编辑。这里用 DOS 行编辑进行送人、修改、显示、列表、装人和存放已封装的 DSP56000/1 程序。

1. 引导 PC 机。若没有硬盘进行第 2 步；若有，进行第 3 步。

2. 把“DOS”软盘放人驱动器“B”，并把一格式化软盘放人驱动器“A”。

键人 b：edlin a：.prog2.asm<return>，把文件名 prog2.asm 放在驱动器“A”中的软盘上。屏幕显示如下。进行第 4 步。

```
New file
* —
```

3. 把格式化软盘放入驱动器“A”，并键入 c：\dos\edlin a：\prog2. asm <return>把文件名 prog2. asm 放在驱动器“A”的软盘上。

屏幕显示同上。进行第 4 步。

4. 键入 I<return>，送入“EDLIN”程序的“嵌入模式”。

5. 为“prog 2. ASM”键入以下源语句。

```
; PROGRAM TO CLEAR THE RAM IN X andY MEMORY<RETURN>
<TAB>MOVE<TAB># $100, RO<TAB>; POINTER TO LONG MEMORY<RETURN>
<TAB>CLR<TAB>B<TAB>; USED TO CLEAR LONG MEMORY<RETURN>
LOOP<TAB>MOVE<TAB>B, L:—(RO)<TAB>; DECREMENT THEN CLEAR ONE LOCATION<RETURN
>
<TAB>MOVE<TAB>RO, A<TAB>; IS POINTER ZERO? <RETURN>
<TAB>TST<TAB>A<TAB>, TEST POINTER<RETURN>
<TAB>JNE<TAB>LOOP<TAB>; LOOP IF NOT DONE<RETURN>
^ Z<RETURN>
```

6. 键入 L<return>，在屏幕上列表“PROG2. ASM”

屏幕应显示如下：

```
1: *; PROGRAM TO CLEAR THE RAM IN X and Y MEMORY
2:; *<TAB>MOVE<TAB># $100, RO<TAB>; POINTER TO LONG MEMORY
3: *<TAB>CLR<TAB>; B<TAB>; USED TO CLEAR LONG MEMORY
4: *LOOP<TAB>MOVE<TAB>B, L: - (RO) <TAB>; DECREMENT THEN
   CLEAR ONE LOCATION
5: *<TAB>MOVE<TAB>RO, A<TAB>; IS POINTER ZERO?
6: *<TAB>TST<TAB>A<TAB>; TEST POINTER
7: *<TAB>JNE<TAB>LOOP<TAB>; LOOP IF NOT DONE
8: * ^ Z<RETURN>
```

7. 键入 E<return>，把“PROG2. ASM”存放在第 2 或第 3 步中命名的目的盘中。

2.6 汇编和链接已封装的 DS56000/1 程序

1. 若想用 DS56000/1 示范软件程序去包括 2.6 节的内容，按照第一章 1.5.1 节的第 1 和 6 步或第一章 1.5.2 节的第 1～3 步装入软件。然后从示范软件菜单选择去进行 2.6 节。

2. 若不想用 DSP56000/1 示范软件程序，进行 2.6.1 节。

2.6.1 用“ASM56000 汇编程序软件”程序

ASM56000 汇编软件程序处理 DSP56000/1 汇编语言源语句为 DSP56000/1 目的码，并产生源语句表。

若所用 PC 带两 个软盘驱动器而无硬盘，完成以下 1～4 步。若所用 PC 有一个或更多个软盘和一个硬盘驱动器，完成以下 5～8 步 。

1. 把含有 prog2. asm 的软盘放入驱动器“A”。

2. 把含有“ASM56000 汇编程序软件”程序的“DS56000 宏汇编程序”软盘放入驱动器“B”

3. 键入 b：asm56000<return>表示“ASM56000 汇编程序软件”程序的命令可选项。

屏幕上显示应类似如下，用不带-A 命令的-B [<lodfil>] 命令选择，指示 ASM56000 汇编软件程序工作于它的相对模式，并形成浮动链接（.lnk）文件。一起用-A-B [<lodfil>] 命令指示 ASM56000 汇编程序工作于它的绝对模式，并形成绝对装入（.lod）文件。

```
Motorola DSP56000 Macro Cross Assembler Version 3.0
(C) Copyright Motorola, Int. 1987, 1988, 1989.
All rights reserved.
Usage: ASM56000 [-A] [-B [<lodfil>]] [-D<symbol><string>]
[-F<argfil>] [-I<ipath>] [-L [<lstfil>]] [-M<mpath>]
[-O<opt> [, <oppt>...]] [-V] <srcfil>...
where:
    lodfil  - optional load file name
    symbol - user-defined symbol
    string  - string associated with symbol
    ipath   - include file directory path
    lstfil  - optional list file name
    mpath  - macro library directory path
    opt     - assembler option
    srcfil  - assembler source file (s)
```

4. 键入 b：asm56000 - ba：prog2. lnk a：prog2：asm<return>，把文件 a：prog2. asm 装入相对模式，产生浮动文件 a：prog2. lnk.

5. 把含有 prog2. asm 的软盘放入驱动器“A”。

6. 键入 cd c：\dsptools\asm 56000<return>送入指定的目录。

7. 键入 asm56000<return>，表示“ASM56000 汇编程序软件”程序的命令选择。

显示类似第 3 步所示。

8. 键入 asm56000 - ba：prog2. lnk Q：prog 2. asm<return>把文件 a：prog2. asm 装入相对模式，并产生浮动文件 a：prog2 lnk.

2.6.2 使用“LNK56000 链接程序软件”程序

LNK56000 链接程序软件处理由 ASM56000 汇编程序软件程序产生的浮动链接(.lnk)文件，形成绝对装入（.lod）文件，为在 DSP56000ADS 应用开发系统上执行，或进入 SIM56000 仿真软件程序中执行，此文件可直接装入。

若 PC 机带两个软盘驱动器而无硬盘，完成下面的第 1～4 步，若有一个或更多的软盘和一个硬盘，完成下面的第 5～8 步。

1. 把含有 prog2. lnk 的软盘放入驱动器“A”。

2. 把含有“LNK56000 链接软件”程序的“DSP56000 链接/库管理程序”的软盘放入驱动器“B”。

3. 键入 b：lnk56000<return>，表示“LNK56000 链接软件”程序的命令选择。

屏幕显示类似于：

```
Motorola DSP56000 Cross Linker Version 3.0
(C) Copyright Motorola, Inc. 1987, 1988, 1989.
All rights reserved.
Usage: LNK56000 [-B [<lodfil>]] [-D] [-F<argfil>]
[-L<library>] [-M [<map>]]
[-O<mem> [<ctr>] [<map]: <origin]
[-R [<memfil]] [-V] [<lnkfil>...
where:
    lodfil  - optional load file name
    argfil  - command line argument file name
    library - library file name
    mapfil  - optional load map file name
    mem     - memory space (X, Y, L, P)
    ctr     - location counter (L, H)
    map     - memory mapping (I, E, B)
    origin  - memory space origin
    memfil  - optional memeory map file name
    lnkfil  - link input file name
```

4. 键入 b: lnk56000 - ba: prog2. lod a: prog2. lnk<return>链接单文件 a: prog2. lnk，并产生绝对装入文件 a: prog2. lod。

通常调用 LNK56000 链接软件程序去链接若干 .lnk 文件，并产生单 .lod 文件，例如 lnk56000 - ba: prgm. lod a: prgm1. lnk a: prgm 2. lod。但如上所示，也可调用 LNK56000 链接软件程序去链接单 .lnk 文件和产生单 .lod 文件。

5. 把含有 prog2. lnk 的软盘放入驱动器“A”。

6. 键入 cd c: \dsptools\lnk56000<return>，进入指定的目录。

7. 键入 lnk56000<return>，表示“LNK56000 链接软件”程序的命令选择。

8. 键入 lnk56000 - ba: prog2. lod a: prog2. lnk<return>链接单文件 a: prog2. lnk，并产生绝对装入文件 a: prog2. lod。

2.7 用“SIM56000 仿真软件”程序进行程序开发和执行

SIM56000 仿真软件程序是为开发和执行 DSP56000/1 程序和算法的软件工具。

2.7.1 “SIM56000 仿真程序软件命令”的综合

本书并不企图用全部命令和指定可选参数的组合。而是仅用于说明本书所含概念的那些命令和命令参数。为了详细了解 SIM56000 命令及其可选参数，可参阅 Motorola 数字信号处理器仿真程序参考手册。表 2-1 列出了 ADS56000 仿真软件命令。

2.7.2 “SIM56000 仿真程序软件”程序基础

1. 按照 2.3.1 节第 1～3 步或 2.3.2 节第 1、5 和 6 步送入“SIM56000 仿真程序软件”。
2. 键入 Change r0 $10<return>，把 DSP56000/1 处理器地址寄存器 r0 的值换到 $10。
3. 键入 Change p: $10 $2<return>，把位于 $10 的 P 存贮器的值换到 $2。

4. 键入 evaluate r0+p：$10<return>，计算 DSP56000/1 处理器地址寄存器 ro 和单元 $10 的 P 存贮器中值的和。

表 2-1 引出了 ADS56000 仿真软件命令

命　令	说　明	命　令	说　明
ASM	单行汇编程序	LOAD	目的文件卸载到 ADM 存贮器
BREAK	设置、修正或清除断点	LOG	存命令/全部会话归档
CHANGE	修正寄存器/存贮器值	OUTPUT	为 ADM 程序输出打开文件
COPY	拷贝-存贮器块到另一块	PATH	为 ADM 置目录通道
DISASSEMBLE	反汇编存贮器	QUIT	退出用户接口程序
OISPLAY	显示寄存器/存贮器值	RADIX	为命令入口置缺省基数
EVALUATE	评价	RESET	复位寄存器或全部仿真器
FORCE	强迫硬件重置或断开	SAVE	存 ADM 状态/存贮器归档
GO	实时执行 ADM 程序	STEP	ADM 的多指令步骤
HELP	在线帮助文本	SYSTEM	执行操作系统命令
HISTORY	反汇编和显示以前 20 条已执行的指令	TRACE	ADM 的单指令步骤
INPUT	为 ADM 程序输入打开文件	WAIT	继续进行以前的等待

屏幕应显示：

```
Hex：000012     Dec：000018     Fract：0.0000021
Bin：00000000000000000000010010
```

5. 键入 evaluate b $32<return>，转换 16 进制值 32 为 2 进制值 。

显示如下：

```
Bin：000 000 000 000 000 000 110 010
```

6. 键入 evaluate h%101010&p：r0<return>，按 16 进制计算 2 进制 101010 和由 DS56000/1 处理器地址寄存器 r0 所指 P 存贮器的值之逻辑“和”。

显示如下：

```
Hex：000002
```

7. 键入 Radix h<return>，改变缺省基数为 16 进制。(前面讲过缺省基数可用 radix 命令改变)。

8. 键入 radix d<return>，改变缺省基数为 10 进制。

9. 键入 change x：0.. 10 $10<return>，把单元 0～10（$0～$a）的 X 存贮器的值换到（$10)。

10. 键入 display x：0.. 10<return>，表示单元 0～10（$0～$a）的 X 存贮器的值($10)。

显示如下：

```
X: $0000   $000010   $000010   $000010   $000010
X: $0004   $000010   $000010   $000010   $000010
X: $0008   $000010   $000010   $000010
P: $E000 000000      =NOP
```

11. 键入 radix f x：0..10<return>，改变单元 0～10（$0～$a）X 存贮器中值的缺省基数从 10 进制到“小数”（FRACTIONAL）算法。

12. 键入 display x：0..10<return>，表示单元 0～10（$0～$a）X 存贮器的“FRACTIONAL”算法值。

显示应为：

```
X: $0000   0.0000019   0.0000019   0.0000019
X: $0004   0.0000019   0.0000019   0.0000019
X: $0008   0.0000019   0.0000019
P: $E000   000000      =NOP
```

13. 键入 display c<return>，表示“SIM56000 仿真软件”程序工作在 DSP56001 配置中显示应为：

```
Device Index: 0, Device Type: 56001
```

14. 键入 wait<return>产生暂停状态。

15. 键入 quit<return>退出“SIM56000 仿真软件”程序。

2.7.3 用“SIM56000 仿真软件”程序去装入、执行和测试已封装的 DSP56000/1 程序

1. 按照 2.3.1 节第 1～3 步或 2.3.2 节第 1、5 和 6 步送入“SIM56000 仿真软件”程序。

2. 把含有 prog2.lod 的软盘放入驱动器“A”。

3. 键入 load a：prog2.lod<return>，把 prog2.lod 从驱动器“A”的软盘中拷贝到“SIM56000 仿真软件”程序环境中。

记住，执行 prog2.lod 要清除 DSP56000/1 处理器的片上 X 存贮器和 Y 存贮器。

4. 键入 disassemble p: 0..10<return>，反汇编和显示单元 0～10（$0～$a）P 存贮器。

显示应为：

```
P: $0000   60F400   000100 =MOVE#>$100, R0
P: $0002   20001B          =CLR B
P: $0003   497800          =MOVE B, L: -(R0)
P: $0004   220E00          =MOVE R0, A
P: $0005   200003          =TST A
P: $0006   0AF0A2   000003 =JNE>$3
P: $0008   000000          =NOP
P: $0009   000000          =NOP
P: $000A   000000          =NOP
```

5. 键入 break＃1 pc＞＝＄8<return>，当 DSP56001 处理器的程序计数器（PC）有一个值大于或等于＄8 时，停止 prog2. lod 的仿真。此时，显示出可显示寄存器和存贮器单元的值。

6. 键入 change x：＄e 0.. ＄ff ＄10 y：＄eo.. ＄ff ＄10<return>，把位于＄e0～＄ff X 存贮器和位于＄e0～＄ff Y 存贮器的值改为＄10

7. 键入 display off<return>，断开和复位显示。

8. 键入 display on x：＄e0.. ＄ff y：＄e0.. ＄ff<return>，使能显示出位于＄e0 ＄ff X 存贮器和位于＄eo～＄ff Y 存贮器的值。

9. 键入 display<return>，表示能显示存贮器单元的值。

屏幕显示应为：

```
X：＄00E0    ＄000010    ＄000010    ＄000010    ＄000010
X：＄00E4    ＄000010    ＄000010    ＄000010    ＄000010
X：＄00E8    ＄000010    ＄000010    ＄000010    ＄000010
X：＄00EC    ＄000010    ＄000010    ＄000010    ＄000010
X：＄00F0    ＄000010    ＄000010    ＄000010    ＄000010
X：＄00F4    ＄000010    ＄000010    ＄000010    ＄000010
X：＄00F8    ＄000010    ＄000010    ＄000010    ＄000010
X：＄00FC    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00E0    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00E4    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00E8    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00EC    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00F0    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00F4    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00F8    ＄000010    ＄000010    ＄000010    ＄000010
Y：＄00FC    ＄000010    ＄000010    ＄000010    ＄000010
P：＄0000 60F400 000100＝MOVE ＃＞＄100，R0
```

10. 键入 go＃1<return>，当遇到断点＃1 时，模拟 prog2. lod. 的执行和停止模拟。屏幕显示如下。注意单元＄ff 的 X 存贮器和 Y 存贮器的值已由＄10 改为＄0。

```
Break ＃1 pc＞＝＄8 h ；dev：0 pc：0008 cyc：20
X：＄00E0   ＄000010    ＄000010    ＄000010    ＄00010
X：＄00E4   ＄000010    ＄000010    ＄00010     ＄000010
X：＄00E8   ＄000010    ＄000010    ＄00010     ＄000010
X：＄00EC   ＄000010    ＄000010    ＄00010     ＄000010
X：＄00F4   ＄000010    ＄000010    ＄00010     ＄000010
X：＄00F8   ＄000010    ＄000010    ＄00010     ＄000010
X：＄00FC   ＄000010    ＄000010    ＄00010     ＄000000
Y：＄00E0   ＄000010    ＄000010    ＄00010     ＄000010
Y：＄00E4   ＄000010    ＄000010    ＄00010     ＄000010
Y：＄00E8   ＄000010    ＄000010    ＄00010     ＄000010
Y：＄00EC   ＄000010    ＄000010    ＄00010     ＄000010
Y：＄00F0   ＄000010    ＄000010    ＄00010     ＄000010
Y：＄00F4   ＄000010    ＄000010    ＄00010     ＄000010
Y：＄00F8   ＄000010    ＄000010    ＄00010     ＄000010
Y：＄00FC   ＄000010    ＄000010    ＄00010     ＄000000
P：＄0003 497800        ＝MOVE B，L：－（R0）
```

11. 键入 go#1：10<reurn>继续 prig2.lod 的仿真，从断点#1 执行，并停在断点的第 10 次事件上。

屏幕应显示如下。注意，单元 $fb～$fe 的 X 存贮器和 Y 存贮器的值已从 $10 改为 $0。

```
Breal #1 pc>=$8h ; dev: 0 pc: 0008 cyc: 76
X: $00E0  $000010   $000010   $000010   $000010
X: $00E4  $000010   $000010   $000010   $00010
X: $00E8  $000010   $000010   $000010   $00010
X: $00EC  $000010   $000010   $000010   $00010
X: $00F0  $000010   $000010   $000010   $00010
X: $00F4  $000010   $000010   $000010   $00010
X: $00F8  $000010   $000010   $000010   $00000
X: $00FC  $000000   $000000   $000000   $00000
Y: $00E0  $000010   $000010   $000010   $00010
Y: $00E4  $000010   $000010   $000010   $00010
Y: $00E8  $000010   $000010   $000010   $00010
Y: $00EC  $000010   $000010   $000010   $00010
Y: $00F0  $000010   $000010   $000010   $00010
Y: $00F4  $000010   $000010   $000010   $00010
Y: $00F8  $000010   $000010   $000010   $00000
Y: $00FC  $000000   $000000   $000000   $00000
P: $0003 497800     =MOVE B, L: -(R0)
```

12. 键入 step<return>，模拟下一个程序指令的执行，并显示可存贮的单元。

DSP56001 程序计数器（PC）增量从 $3～$4，单元 $fa 的 X 和 Y 存贮器的值已从 $10 变为 $0。屏幕上应显示：

```
X: $00E0  $000010   $000010   $000010   $000010
X: $00E4  $000010   $000010   $000010   $00010
X: $00E8  $000010   $000010   $000010   $00010
X: $00EC  $000010   $000010   $000010   $00010
X: $00F0  $000010   $000010   $000010   $00010
X: $00F4  $000010   $000010   $000010   $00010
X: $00F8  $000010   $000010   $000000   $00000
X: $00FC  $000000   $000000   $000000   $00000
Y: $00E0  $000010   $000010   $000010   $00010
Y: $00E4  $000010   $000010   $000010   $00010
Y: $00E8  $000010   $000010   $000010   $00010
Y: $00EC  $000010   $000010   $000010   $00010
Y: $00F0  $000010   $000010   $000010   $00010
Y: $00F4  $ 00010   $000010   $000010   $00010
Y: $00F8  $000010   $000010   $000000   $00000
Y: $00FC  $000000   $000000   $000000   $00000
P: $0004 220E00     =MOVE R0, A
```

13. 键入 step 50<return>，模拟下面 59 条程序指令的执行，并在第 15 条指令已执行后显示 X 和 Y 存贮器单元的值。

屏幕应显示如下。注意单元 $ee～$f9 的 XY 存贮器的值已从 $10 变为 $0。

```
X:  $00E0   $000010   $000010   $000010   $000010
X:  $00E4   $000010   $000010   $000010   $00010
X:  $00E8   $000010   $000010   $000010   $00010
X:  $00EC   $000010   $000010   $000000   $00000
X:  $00F0   $000000   $000000   $000000   $00000
X:  $00F4   $000000   $000000   $000000   $00000
X:  $00F8   $000000   $000000   $000000   $00000
X:  $00FC   $000000   $000000   $000000   $00000
Y:  $00E0   $000010   $000010   $000010   $00010
Y:  $00E4   $000010   $000010   $000010   $00010
Y:  $00E8   $000010   $000010   $000010   $00010
Y:  $00EC   $000010   $000010   $000000   $00000
Y:  $00F0   $000000   $000000   $000000   $00000
Y:  $00F4   $000000   $000000   $000000   $00000
Y:  $00F8   $000000   $000000   $000000   $00000
Y:  $00FC   $000000   $000000   $000000   $00000
P:  $0006 OFE0A2 000003=JNE > $3
```

14. 键入 dislpay cyc<return>，表示已模拟的 DSP56001 时钟周期数。

屏幕显示如下：

```
cyc=000256
P:  $0006 0AF0A2 000003 = JNE > $3
```

15. 键入 display on cyc<return>，使能显示 DSP56001 时钟周期数加能显示存贮器单元的值。

16. 键入 step52 cy<return>，模拟下面 52 秒 DSP56001 时钟周期的执行，并在 50 秒时钟周期后显示各项的值。

屏幕显示如下。注意单元 $ea～$ed 的 X 和 Y 存贮器的值已由 $10 变为 $0。

```
cyc=000308
X:  $00E0   $000010   $000010   $000010   $000010
X:  $00E4   $000010   $000010   $000010   $00010
X:  $00E8   $000010   $000010   $000000   $00000
X:  $00EC   $000000   $000000   $000000   $00000
X:  $00F0   $000000   $000000   $000000   $00000
X:  $00F4   $000000   $000000   $000000   $00000
X:  $00F8   $000000   $000000   $000000   $00000
X:  $00FC   $000000   $000000   $000000   $00000
Y:  $00E0   $000010   $000010   $000010   $00010
Y:  $00E4   $000010   $000010   $000010   $00010
Y:  $00E8   $000010   $000010   $000000   $00000
Y:  $00EC   $000000   $000000   $000000   $00000
Y:  $00F0   $000000   $000000   $000000   $00000
Y:  $00F4   $000000   $000000   $000000   $00000
Y:  $00F8   $000000   $000000   $000000   $00000
Y:  $00FC   $000000   $000000   $000000   $00000
P:  $0004 220E00          =MOVE R0, A
```

17. 键入 display cyc<return>，显示已模拟的 DSP56001 时钟周期数。

屏幕显示如下：

```
cyc=000308
P: $0004 220E00          =MOVE R0, A
```

18. 键入 trace 14 cy<return>，模拟下面 14 个 DSP56001 时钟周期的执行，并在 14 个时钟周期后显示各项目。

显示如下。注意路线计数从 13 减到 0，并且 X 和 Y 存贮器单元 $eq 的值已由 $10 改成 $0。

```
Trace cycle count=0
cyc=000322
X: $00E0  $000010   $000010   $000010   $000010
X: $00E4  $000010   $000010   $000010   $00010
X: $00E8  $000010   $000000   $000000   $00000
X: $00EC  $000000   $000000   $000000   $00000
X: $00F0  $000000   $000000   $000000   $00000
X: $00F4  $000000   $000000   $000000   $00000
X: $00F8  $000000   $000000   $000000   $00000
X: $00FC  $000010   $000010   $000010   $00010
Y: $00E0  $000010   $000010   $000010   $00010
Y: $00E4  $000010   $000010   $000010   $00010
Y: $00E8  $000010   $000000   $000010   $00010
Y: $00EC  $000000   $000000   $000000   $00000
Y: $00F0  $000000   $000000   $000000   $00000
Y: $00F4  $000000   $000000   $000000   $00000
Y: $00F8  $000000   $000000   $000000   $00000
Y: $00FC  $000000   $000000   $000000   $00000
P: $0004 220E00          =MOVE R0, A
```

19. 键入 qiut<return>，退出“SIM56000 模拟软件”程序。

2.8 定义和使用宏指令

程序常常重复一组指令。宏指令提供一种速记法，通过它可将一组 DSP56000/1 指令用一个名称来指定。所以当键入程序时，重复的指令组可由它的宏名称指定，而不需重新键入指令本身。

宏指令定义为：标题、主体、终止符。标题给宏指令指定名称并定义哑变量，即当宏指令被调用时以实际变量代替的符号名。主体包含一组 DSP56000/1 源指令。终止符是结束宏（ENDM）的命令。

宏指令由调宏用语句指定在程序中，宏调用语句有三段：标记段、操作段和操作数段。如果用的话，标记段相应于宏扩展开始处的软件单元计数器的值，即代替宏调用语句中相关指令组。操作段包括宏名称。操作数段包括实际变量，它将被用在宏定义标题部分来代替哑变量。

2.8.1 编程已封装的 DSP56000/1 宏指令

DSP56000/1 程序可用任何有效的字处理程序送入和编辑。这里，用 DOS 编辑器来送入、

显示和存放已封装的 DSP56000/1 宏指令。

1. 引导 PC。

若所用 PC 有二个软盘和没有硬盘驱动器，则进入第 2 步。若有一个硬盘和一个或更多软盘驱动器，则跳过第 2 步而进入第 3 步。

2. 把“DOS 软盘放入驱动器 B”，而把格式化软盘放入驱动器“A”。

键入 b：edlin a：\mal. asm <return>，把文件名 mal. asm 放在驱动器“A”上。

屏幕上应有以下消息和提示。进入第 4 步。

```
New file
* —
```

3. 把格式化软盘放入驱动器“A”，并键入 c：\dos\edlin a：\mal. asm<return>把文件名 mal. asm 放在“A”软盘上。

屏幕应显示如下，进入第 4 步。

```
New file
* —
```

4. 键入 I<return>，进入“EDLIN”程序的“插入模式”。

5. 键入 MA1<tab>MACRO<tab>VALUE<return>，定义宏标题。标题指定名称 MA1 并将哑变量 VALUE 赋于宏指令。

6. 键入：

```
; MACRO TO CLEAR THE RAM IN X AND Y MEMORY <return>
 <tab>MOVE<tab>#VALUE, R0<tab>; POINTER TO LONG MEMERY <return>
 <tab>CLR<tab>B<tab>; USED TO CLEAR LONG MEMORY<return>LOOP<tab>MOVE<tab>B, L: -
 (R0) <tab>; DECREMEHT THEN CLEAR ONE LOCATION <return>
 <tab>MOVE<tab>R0, A<tab>; DOES POINTER EQUAL ZERO? <return>
 <tab>TST<tab>A<tab>; TEST POINTER<return>
 <tab>JNE<tab>LOOP<tab>; LOOP IF NOT DONE<return>
```

此时已键入了宏 MA1 的主体。MA1 清除了 DSP56000/1 处理器片上的 X 和 Y 存贮器。现在不再涉及指令本身。

7. 键入<tab>ENDM<return>^Z<return>，进入宏指令的终止符；并退出“EDLIN”的插入模式。

8. 键入 E<return>，把 mal. asm 存入第 2 或第 3 步中命名的目的中。

9. 根据系统配置，键入 b：edlin a：prog3. asm<return>或 c：\dos\edlin a：prog3. asm<return>，把文件名 prog3. asm 放在驱动器“A”的软盘上。

屏幕显示如下：

```
New file
* —
```

10. 键入<tab>INCLUDE<tab>MA1<return>使 mal 宏指令 INCLUDE 到 prog3. asm 中。

ASM56000 汇编程序指令（本书所用）如表 2-2 所示。

表 2-2 ASM56000 汇编程序指令

指　令	说　明	指　令	说　明
DC	定义常数	EQU	使符号等于某值
DS	定义存贮器	INCLUDE	包含二次文件
DSM	定义模存贮器	MACRO	宏定义
DSR	定义反向进住存贮器	ORG	初始化存贮空间和单元计数器
DUP	复制源行序列	SET	设置符号值
END	结束源程序		

举　例

1. 汇编程序指令 ORG p：$100 置运行时间存贮器空间在 100 开始的 P 存贮器。

2. 汇编程序指令 COUNT EQR $12 指定值 $12 到符号 COUNT。

3. 汇编程序指令 ORG x：$100 和 TABLE DC 0.1，0.2，0.3 存 10 进制数 0.1，0.2 和 0.3 在从 $100 开始的连续的 X 存贮器中。

4. 汇编程序指令 ORG x：$100 和 STATES DS 6 为 STATES 变量保留从 100 开始的连续 6 个 X 存贮器单元。

11. 键入 VALUE<tab>EQU<tab>$100<return>，给符号“VALUE”指定值 $100。

12. 键入<tab>MA1<tab>VALUE<return>，进入调用 MA1 宏指令的宏调用语句。

13. 键入<tab>END<retun>^Z，指示 prog3.asm 的逻辑结束。

14. 键入 E<return>，存 prog3.asm 在第 9 步命名的目标中。

2.8.2 汇编已封装的 DSP56000/1 宏指令

1. 引导 PC。

若 PC 无硬盘而只有两个软盘驱动器，则进入第 2 步。若有一个硬盘和一个或更多的软盘驱动器，则跳过第 2、3 步而进入第 4 步。

2. 把含有“ADS56000 汇编程序软件”的“DSP56000 宏交叉汇编程序”软盘放入驱动器“B”。

3. 键入 b：asm56000 - ba：prog3.lnk - ia：prog3.asm<return>，汇编 prog3.asm 和输出可重置位链接文件 prog3.lnk 到驱动器“A”。

进入 2.8.3 节。

4. 键入 cd c：\dsptools\asm56000<return>进入目录。

5. 键入 c：asm56000 - ba：prog3.lnk - ia：prog3.asm<return>

汇编 prog3.asm 并输出可重置位链接文件 prog3.lnk 到驱动器“A”。

进入 2.8.3 节。

2.8.3 链接已封装的 DSP56000/1 宏指令

1. 引导 PC 机（同前）。

2. 把含有“LNK56000LINKER/LIBRARIAN 软件”程序的“DSP56000 宏交叉汇编程序”软盘放入驱动器“B”。

3. 键入 b：lnk56000 - ba：porg3. lod a：prog3. lnk<return>进行 2.8.4 节。

4. 键入 cd c：\dsptools\lnk56000<return>送入目录。

5. 键入 c：lnk56000 - ba：prog3. lod a：prog3. lnk<return>。

进入 2.8.4 节。

2.8.4 装入已封装的 DSP56000/1 宏指令

1. 引导 PC（同前）。

2. 把含有“SIM56000 模拟软件”程序的 DSP56000 模拟软盘放入驱动器“B”。

3. 键入 b：sim56000<return>

屏幕显示如下。进入第 6 步。

```
0>
asm break change copy disassemble display<space> = more
```

4. 键入 cd c：\dsptools\sim56000<return>，进入目录。

5. 键入 sim56000<return>，进入“SIM56000 模拟软件”程序。

显示同 3. 进入第 6 步。

6. 把含有 prog3. lod 的软盘放入驱动器“A”。

7. 键入 load a：prog3. lod<return>，把 prog3. lod 从驱动器“A”的软盘中装入“SIM56000 模拟软件”程序环境中。

8. 键入 disassemble p：0. . 10<return>反汇编并显示位于 0～10 的 P 存贮器的内容（$0～$a）。

屏幕显示如下：

```
P：$0000    60F400    000100 =MOVE#>$100，R0
P：$0002    200018          =CLR B
P：$0003    497800          =MOVE B，L：-(R0)
P：$0004    220E00          =MOVE R0，A
P：$0005    200003          =TST A
P：$0006    0AF0A2    000003 =JNE>3
P：$0008    000000          =NOP
P：$0009    000000          =NOP
P：$000A    000000          =NOP
```

9. 键入 quit<return>，退出“SIM56000 模拟软件”程序。

第三章　DSP56001 结构和寻址方式

本章介绍 DSP56001 处理器的结构、寻址方式和一些机器指令。本章开始先复习 DSP56001 处理器的结构，然后叙述它的三个处理执行单元：数据算术/逻辑单元、程序控制器和地址产生单元。还讨论了 DSP56001 处理器的寻址方式、直接寄存器、专用和间接寄存器。

3.1　DSP56001 结构回顾

DSP56001 处理器结构的主要部件如第一章的图 1-2 所示，现叙述如下。

3.1.1　总线

DSP56001 内部多总线结构由 4 条双向 24 比特数据总线和 3 条单向 16 比特地址总线组成。

数据总线包括：XD 数据总线，YD 数据总线，PD 程序数据总线和 GD 全局数据总线。XD 总线和 YD 总线分别传送数据 ALU 同 X 数据存贮器和 Y 数据存贮器之间的数据。XD 和 YD 总线也可由一定指令进行并置作为一条 48 比特总线。PD 总线传送预取数指令字。GD 总线控制“其他”数据传送，如 I/O 送至和来自外围设备的数据。

地址总线包括：XA 地址总线，YA 地址总线和 PA 程序地址总线。XA 和 YA 总线分别为指定的在 X 和 Y 存贮器中的单元提供地址，PA 总线为 P 程序存贮器中指定的单元提供地址。

3.1.2　内部存贮器

DSP56001 处理器有 6 个片上存贮器，如第一章所述。X 数据 RAM 是 24 比特内部存贮器，它们分别占有 X 和 Y 存贮器地址空间的最低 256 个单元。X 和 Y 数据 ROM 是 24 比特内部存贮器，当操作方式寄存器允许时，分别占据 X 和 Y 存贮器地址空间次最低的 256 个单元。P 程序 RAM 是 24 比特内存贮器，它在 P 存贮器地址空间占最低的 512 个单元。P 程序 RAM 包含机器指令、常数和数据表，表是在汇编时建立的。P RAM、X RAM 和 Y RAM 中不用的单元可用于通用存贮。引导程序 ROM 是 32 单元乘 24 比特的已编程 ROM，它仅用在引导程序方式。引导程序方式提供方便、廉价、自动的方法，通过它可使 DSP56001 机器指令的程序从外部存贮器传送至内部 P 程序存贮器。

3.1.3　执行单元

DSP56001 的核心由 3 个执行单元组成：数据算术/逻辑单元、程序控制器和地址产生单元，将分别在 3.3、3.4 和 3.5 节中叙述。

3.1.4　片上外围设备

DSP56001 处理器的片上外设提供微计算机式的 I/O 能力，片上外设包括：并行主机

MPU/DMA 端口、同步串行通信接口（SCI）端口、同步串行接口（SSI）端口和可编程 I/C 端口。8 比特并行主机 MPU/DMA 端口提供一个途径，通过它主处理器或 DMA 控制器可访问 DSP56001 处理器。SCI 端口通常提供标准串行 I/O 接口（用异步字符协议）以便同 MCU、其他 DSP 片、RS232C 型线路进行通信。SSI 端口通常提供高速同步串行数据接口，与其他 DSP 片或其他同步串行设备通信。可编程 I/O 端口提供通用接口。

3.1.5 存贮器扩展端口

它由以下组成：16 位地址总线、24 位双向数据总线、一组存贮器集选择信号和一组存贮器握于信号。它用于访问外部 X 数据存贮器、Y 数据存贮器、程序存贮器和 I/O 设备。

3.2 如何进行

若要用 DSP 56000/1 示范软件程序包括 3.3 和 3.6 节的内容，则用第一章 1.5.1 节的第 1 和 6 步或 1.5.2 节的第 1～3 步装入软件。然后从该示范软件的菜单中去选择包括 3.3 和 3.6 节的内容。

若不用 DSP56000/1 示范软件程序，用第二章 2.3.1 节的第 1、2、3 步或 2.3.2 节的第 1、5、6 步装入 SIM56000 模拟软件。进入 3.3 节。

3.3 数据 ALU 执行单元

数据 ALU 执行单元示于图 3－1，它完成数据操作数的算术和逻辑操作。主要包括：10 个数据寄存器、1 个 24×24 乘法器和算术/逻辑单元、1 个位变换单元、1 个累加移位器和两个移位器/限幅器。

3.3.1 数据寄存器

数据 ALU 有 4 个输入数据寄存器和 6 个数据累加寄存器，如图 3－2 所示。

1. 数据输入寄存器 X1，X0；Y1，Y0

数据输入寄存器可当作 4 个独立的 24 位寄存器 X1，X0，Y1 和 Y0，或当作 2 个 48 位寄存器 X 和 Y。48 位 X 和 Y 寄存器分别由 X1、X0 和 Y1、Y0 并置形成。

例 3.1：移数据到 24 位数据输入寄存器。

问题：执行指令 MOVE# $1111100，X1 移 $111100 值到 X1 数据寄存器。设指令执行前 X1、X0 和 X 中内容如下：

x= $333333333333

x1= $333333　　x0= $333333

键入：change l $333333333333<return>

display off <return>

display x1 x0 x<return>

asm p: $200 move# $111100, x1<return>

change pc $200<return>

step<return>

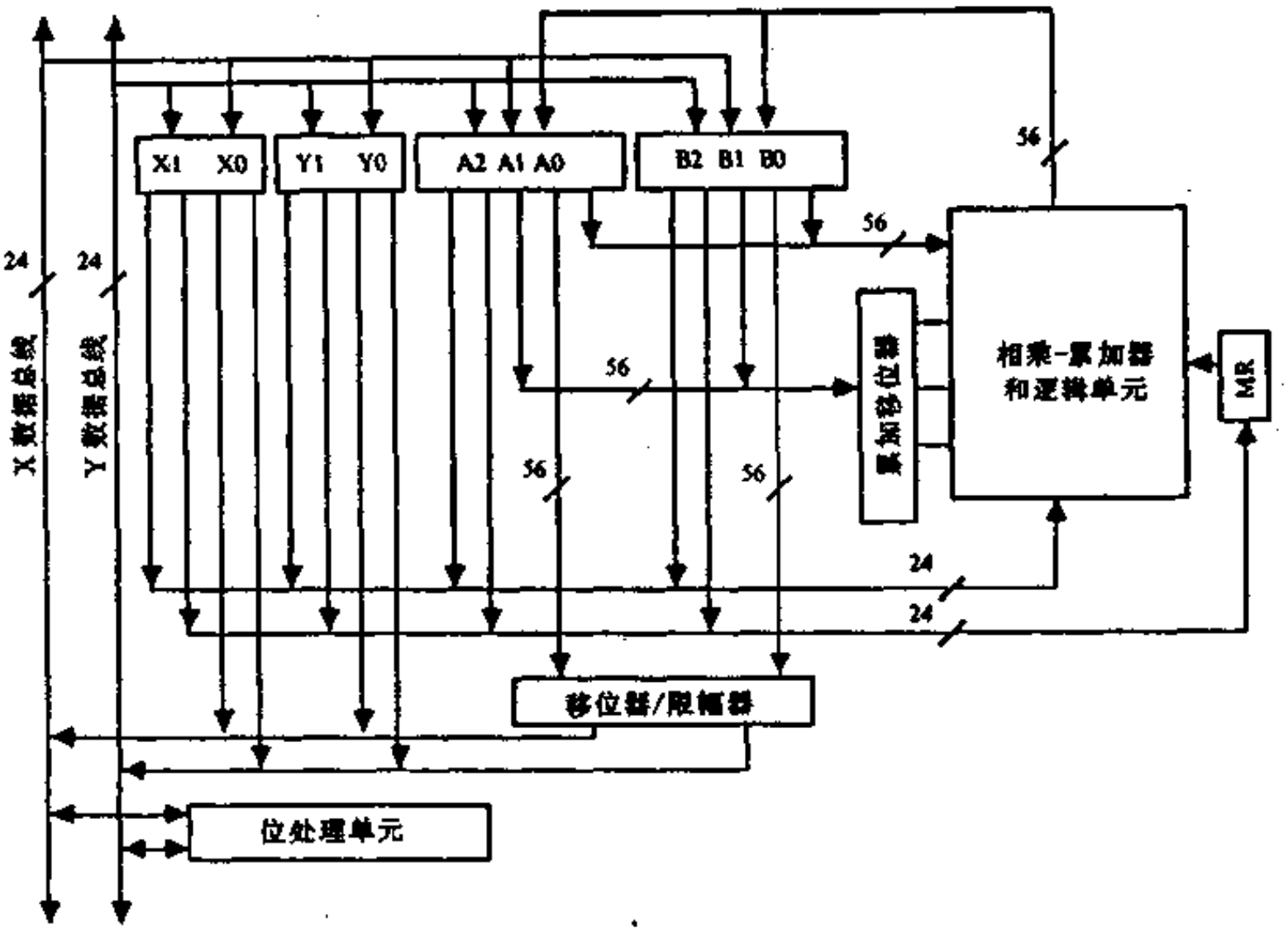


图 3－1　数据 ALU 方框图

数据 ALU 输入寄存器

数据 ALU 累加器

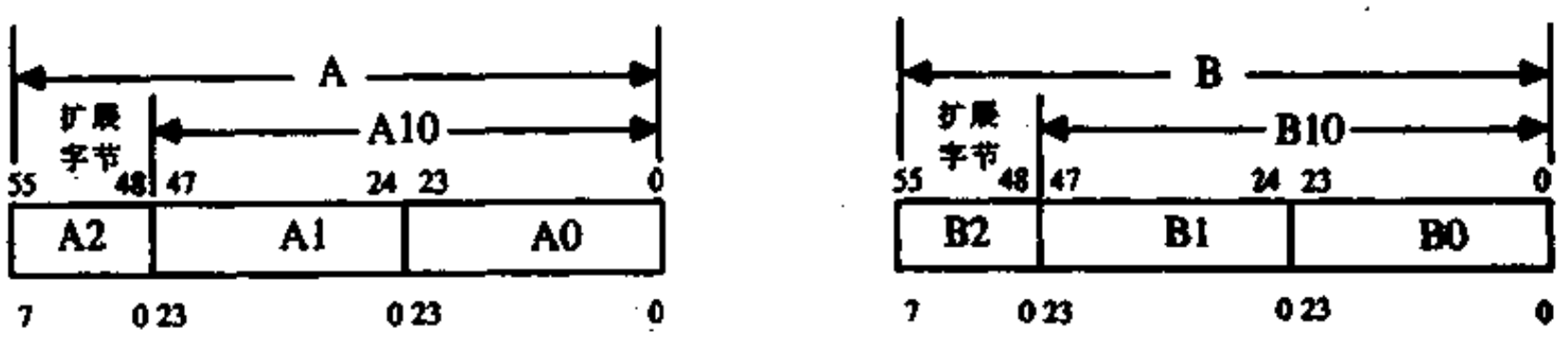


图 3－2　数据寄存器

display x1 x0 x<return>

结果：x＝$111100333333

x1＝$111100　　x0＝$333333

例 3.2：移数据到 48 位数据输入寄存器。

问题：执行指令 MOVE L：$100，X，移单元 $100 的 L 存贮器（单元 $100 的 X 和 Y

存贮器并置）的内容到 X 数据输入寄存器。设指令执行前 X 和 L 中内容分别为：

l：$100　　$333333444444

x=$111111222222

键入：change l：$100 $333333444444 x111111222222<return>

display l：$100 x<return>

asm p：$202 move l：$100，x<return>

change pc $202 <return>

step <return>

display l：$100 x<return>

结果：l：$100　　$333333444444

x=$333333444444

2. 数据累加寄存器：A2，A1，A0；B2，B1，B0

这 6 个数据累加寄存器形成 2 个 56 位累加器 A 和 B，A 由 A2：A1：A0 构成，B 由 B2：B1：B0 构成。A1、A0、B1 和 B0 每个是 24 位宽；A2 和 B2 是 8 位宽，称扩展寄存器。

例 3.3：移 24 位操作数到 56 位累加器。

问题：执行 MOVE X0，A 和 MOVE X1，B 两条指令，说明当移 24 位数据操 作数到 56 位 A 累加器和移 24 位数据操 作数到 56 位 B 累加器时，发生自动的符号位扩展。设指令执行前，各寄存器中的内容如下：

x0=$888888　　x1=$111111

a=$002222222222222

a2=$00　　a1=$222222　　a0=$222222

b=$002222222222222

b2=$00　　b1=$222222　　b0=$222222

键入：change x0 $888888 x1 $111111<return>

change a $002222222222222<return>

change b $002222222222222<return>

display a a2 a1 a0 b b2 b1 b0 x0 x1<return>

asm p：$204 move x0，a<return>

asm p：$206 move x1，b<return>

change pc $204<return>

step 4<return>

display a a2 a1 a0 b b2 b1 b0 x0 x1<return>

结果：x0=$888888　　x1=$111111

a=$ff888888000000

a2=$ff　　a1=$888888　　a0=$000000

b=$001111111000000

b2=$00　　b1=$111111　　b0=$000000

例 3.4：移 16 位操作数到 56 位累加器。

问题：执行指令 MOVE R0，A。设执行前 R0 和 A 的内容为：

r0=$8888　　a=$001111111111111

键入：change r0 $8888 a $001111111111111<return>

```
display r0 a<return>
asm p: $208 move r0, a<return>
change pc $208<return>
step<return>
display r0 a<return>
```

结果：r0＝$8888　　a＝$00008888000000

例 3.5：移 56 位 A 累加器操作数到 16 位和 24 位目的地。

问题：执行指令 MOVE A，R2 和 MOVE A，X1。

设执行前寄存器内容为

a＝$00123456789abc

r2＝$0000　　x1＝$000000

键入：
```
change a $00123456789abc r2 $0000<return>
change x1 $000000<return>
display a r2 x1<return>
asm p: $209 move a, r2<return>
asm p: $20a move a, x1<return>
change pc $209 <return>
step 2<return>
display a r2 x1<return>
```

结果：a＝$00123456789abc

r2＝$3456　　x1＝$123456

例 3.6：移数据操作数到累加器，不带符号自动扩展。

问题：执行指令 MOVE ＃$111111，A1。设指令执行前，寄存器内容为：

a＝$ff888888222222

a2＝$ff　　a1 $888888　　a2＝$222222

键入：
```
change a $ff888888222222<return>
display a a2 a1 a0<return>
asm p: $20b move # $111111, a1<return>
change pc $20b <return>
step<return>
display a a2 a1 a0<return>
```

结果：a＝$ff111111222222

a2＝$ff　　a1＝$111111　　a0＝$222222

3.3.2　乘法器和算术/逻辑单元

它们完成 DSP56001 处理器所有的数据操作数的计算，如相乘、相加、相减、与、或、与或和非。它接受多达 3 个输入操作数并产生 56 位结果。

例 3.7：相加。

问题：执行指令 ADD A，B，将 A 累加器加至 B 并存在 B 中。设执行前

a＝$00111111111111　　b＝$00222222222222

键入：
```
change a $00111111111111 b $00222222222222<return>
display a b<return>
```

```
asm p: $210 add a, b<return>
change pc $210<return>
step<return>
display a b<return>
```

结果：a=$00111111111111　　b=$00333333333333

例 3.8：相减。

问题：执行指令 SUB X，A，从 A 中减去 X 的 48 位操 作数，结果存入 A 累加器。设执行前

x=$111111111111　　a=00333333333333

键入：
```
change x $111111111111 a $00333333333333<return>
display x a>return>
asm p: $211 sub x, a<return>
change pc $211<return>
step <return>
display x a<return>
```

结果：x=$111111111111　　a=$00222222222222

例 3.9：逻辑与。

问题：执行指令 AND X1，A，把 X1 中 24 位操 作数和 A 累加器相与，结果存入 A 累加器，设执行前：

x1=$f0f0f0　　a=$ff888888888888

键入：
```
change x1 $f0f0f0 a $ff888888888888<return>
display x1 a<return>
asm p: $212 and x1, a<return>
change pc $212<return>
step<return>
display x1 a<return>
```

结果：x1=$f0f0f0　　a=$ff808080888888

1. 2 的补码的小数数据表示

这种表示通常用于 DSP 算法中，这里小数是任何大于或等于零且小于 1 的数。图 3-3 表示小数数据操作数的位权重。

例 3.10：正的 2 的补码小数数据。

问题：执行指令 MOVE#0.875，Y0

键入：
```
asm p: $214 move#0.875, y0<return>
change pc $214<return>
step<return>
display y0<return>
```

结果：y0=700000=%011100000000000000000000

$=0*2^{-0}+1*2^{-1}+1*2^{-2}+1*2^{-3}=$ '-0.875

例 3.11：负的 2 的补码小数数据。

问题：执行指令 MOVE #−0.875，X0

键入：
```
asm p: $216 move#-0.875, x0<return>
change pc $216<return>
```

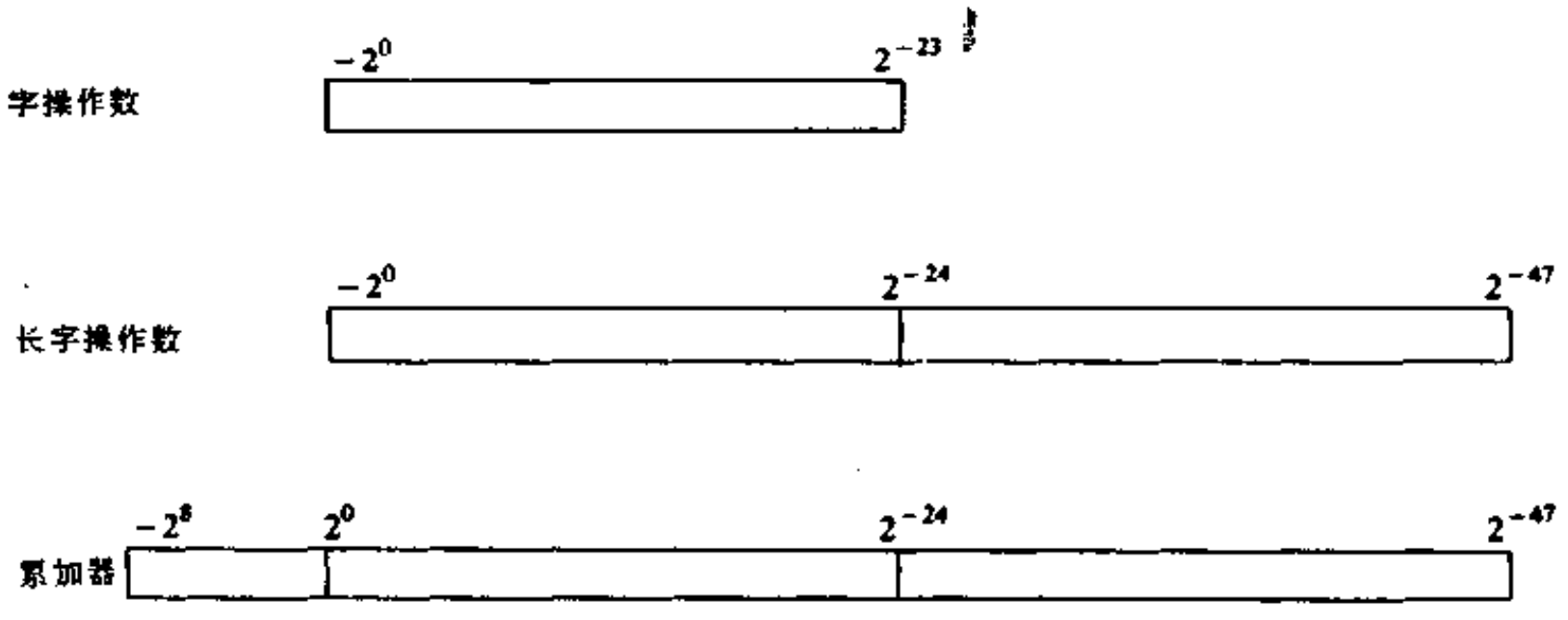


图 3-3 操作数的位权重

step<return>

display x0<return>

结果：x0＝ $900000 ＝%100100000000000000000000＝ ‘_0.875

为了变换以上 2 的补码的 2 进制数 100100……00 为 10 进制数，每位必须互补以形成 1 的补码数 011011……11。然后把 1 加到这 1 的补码数形成 2 进制数 011100……00，这等于 10 进制数 0.875。这 10 进制数的符号为负，因为等效的 2 的补码 2 进制数的符号位是 1。

2. 舍入

相乘器和算术/逻辑单元可以把累加器 A0 或 B0 的最低有效部分舍入到 A1 或 B1 的最高有效部分。

例 3.12：A 累加器寄存器 A0 的值大于 0.5。

问题：执行指令 RND A，表示当 A 累加器寄存器 A0 的值大于 0.5（$800000）时，把 1 加到寄存器 A1。设 A1 和 A0 以及状态寄存器 SR 的原内容为：

sr＝ $0300

a1＝ $111111　　a0＝ $808080

键入：change a1 $111111 a0 $808080<return>

change sr $300<return>

display a1 a0 sr<return>

asm p: $220 rnd a<return>

change pc $220 <return>

step<return>

display a1 a0 sr<return>

结果：sr＝ $0310

a1＝ $111112　　a0＝ $000000

例 3.13：B 累加器寄存器 B0 的值小于 0.5。

问题：执行指令 RND B，表示当 B0 的值小于 0.5（$800000）时，B1 不改变。设执行前 B1 和 B0 的原内容如下：

b1＝ $111111　　b0＝ $777777

键入：change b1 $111111 b0 $777777<return>

display b1 b0<return>

```
asm p: $221 rnd b<return>
change pc $221<return>
step<return>
display b1 b0<return>
```

结果：b1＝$111111　　b0＝000000

当 A0 和 B0 都是 0.5 时，执行指令 RND A 和 RND B，结果 A1 加 1，B1 不变。

例 3.14：有舍入和无舍入相乘。

问题：执行指令 MPYR X1，X0，A 和 MPY X1，X0，B。

(1) X1 的值与 X0 的值相乘，并将舍入结果存入 A 累加器。

(2) X1 的值和 X0 的值相乘，并将无舍入结果存入 B 累加器。

设执行前：

X1＝400000　　X0＝000007

a＝$00000000000000　　b＝$00000000000000

键入：

```
change a 0 b 0<return>
change x1 $400000 x0 $000007<return>
display x1 x0 a b<return>
asm p: $224 mpyr x1, x0, a<return>
asm p: $225 mpy x1, x0, b<return>
change pc $224<return>
step2<return>
display x1 x0 a b<return>
```

结果：x1＝$400000　　x0＝$000007

a＝$00000004000000　　b＝$00000003800000

3.3.3 累加器移位器

它接受 56 位输入和产生 56 位输出。移位器移 1 位输入操作数向左、或移 1 位向右或不位移地通过。

例 3.15：左移。

问题：执行指令 ASL A，使 A 累加器向左移一位。设寄存器原内容为：

a＝$ff888888333333＝ '－0.933333373069765

sr＝$0300

键入：

```
change a $ff888888333333 sr $0300<return>
display a sr<return>
radix f a<return>
display a <return>
radix h a<return>
asm p: $230 asl a<return>
change pc $230<return>
step<return>
display a sr <return>
radix f a<return>
display a<return>
```

radix h a<return>

结果：a＝ $ ff111110666666＝ ‘－1.866666746139529

sr＝ $ 0339

注意：状态寄存器的C位（最低有效位）收到A累加器移出的最高有效位，同时零移入A的最低有效位。还要注意A的内容由2来升标。

例3.16：右移。

问题：执行指令ASR B，使B累加器向右移1位，寄存器原内容为：

b＝ $ ff888888333333＝ ‘－0.933333373069765

sr＝ $ 0300

经过与前例相似过程，得结果：

b＝ $ ffc44444199999＝ ‘－0.466666686534886

sr＝ $ 0319

3.3.4 数据移位/限幅器

两个数据移位/限幅器提供专用的数据后处理，这数据是由累加器移到XD或YD数据总线的。一个56位移位/限幅器与A有关；另一个与B有关。每个移位/限幅器由一限幅器跟随移位器组成。

1. 数据移位器

每个数据移位器通过一个相关的数据限幅器，数据限幅器有24位输出和一个溢出指示器。数据移位器由状态寄存器SR中的S1和S0定标位控制，S1和S0分别为第11和第10位。S1和S0允许定点数据动态定标，而不需要用户修正程序的指令。定标模式如表3－1。

表3－1 定 标 模 式

S1	S0	定 标 模 式
0	0	原标
0	1	降标（算术右移一位）
1	0	升标（算术左移一位）

例3.17：无定标。

问题：执行指令MPY X1，Y1，A和MOVE a，X0。设X0，X1，Y1，A和定标位的原值为：

x1＝ $ 400000　　y1＝ $ 400000

a＝ $ 000000000000　　x0＝ $ 000000

s1＝0，　　s0＝0

键入：change x1 $ 400000 y1 $ 400000<return>

change sr 0 a 0 x0 0<return>

display x1 y1 sr a x0<return>

asm p：$ 235 mpy x1，y1，a<return>

asm p：$ 236 move a，x0<return>

change pc $ 235<renturn>

step 2<return>

display x1 y1 sr a x0<return>

结果：x1＝$400000　　y1＝$400000

sr＝$0010　　a＝$002000000000000

x0＝$200000

例 3.18：降标。

问题：执行指令 MPY X1，Y1，A 和 MOVE a，X0。

设 X0，X1，Y1，A 和 S1，S0 原值为：

x1＝$400000　　y1＝$400000

a＝$000000000000000　　x0＝$000000

s1＝0，s0＝1

键入：change a 0 pc $235 x0 0<return>

change sr $0400<return>

display x0 x1 y1 sr a

asm p：$235 mpy x1，y1，a

asm p：$236 move a，x0

step 2<return>

display x1 y1 sr a x0<return>

结果：x1＝$400000　　y1＝$400000

sr＝$0410　　a＝$002000000000000

x0＝$100000

例 3.19：升标。

问题：执行指令 MPY X1，Y1，A 和 MOVE a，x0。

设各寄存器的原来内容为：

x1＝$400000　　y1＝$400000

a＝$000000000000000　　x0＝$000000

s1＝1　　，s0＝0

键入：change a 0 pc $235 x0<return>

change sy $ 800<return>

display x0 x1 y1 sr a

asm p：$235 mpy x1 y1 a

asm p：$236 move a，x0

step 2<return>

display x1 y1 sr a x0<return>

结果：x1＝$400000　　y1＝$400000

sr＝$0800　　a＝$002000000000000

x0＝$400000

2. 限幅器

每个限幅器可自动完成数据的饱和运算。若所选源累加器内容可以用目的操作数表示而没有溢出，则数据限幅器不用，且操作数不修正。若不用溢出就不能表示，则限幅器将用最大的“受限”数据代替，其符号与源累加器相同。

用两个数据移位/限幅器，可在同一指令周期内独立限制 2 个 24 位操 作数。也可两者并置形成一个 48 位长字操 作数的移位/限幅器。受限数据值示于表 3－2。

表 3－2 受限数据值

目的存贮器参考	受限值（Hex）				访问形式
	源操作数	ACC 符号	XD 总线	YD 总线	
X	X：A X：B	+ －	7FFFFF 800000	… …	1 个 24 位字
Y	Y：A Y：B	+ －	… …	7FFFFF 800000	1 个 24 位字
X 和 Y	X：A Y：A X：A Y：B X：B Y：A X：B Y：B L：AB L：BA	+ －	7FFFFF 800000	7FFFFF 800000	2 个 24 位字
L（X：Y）	L：A L：B	+ －	7FFFFF 800000	7FFFFF 800000	1 个 48 位长字

例 3.20：有限制的传送数据。

问题：执行指令 MOVE A，X0 用 MOVE B，Y1，表示当数据从 56 位 A 和 B 累加器传到 24 位 X0 和 Y1 数据输入寄存器时，自动产生限制。设寄存器原内容为：

a＝ $0100000f800000

a2＝ $01　　a1＝ $00000f　　a0＝ $800000

x0＝ $000000

b＝ $8000000f800004

b2＝ $80　　b1＝ $00000f　　b0＝ $800004

y1＝ $000000

键入：change a $0100000f800000　x0　$000000<return>

change b $8000000f800004　y1 $000000<return>

display a b x0 y1<return>

asm p：$237 move a，x0<return>

asm p：$238 move b，y1<return>

change pc $237<return>

step 2<return>

display a b x0 y1<return>

结果：a＝ $0100000f800000　　x0＝ $7fffff

a2＝ $01　　a1＝00000f　　a2＝ $800000

b2＝ $8000000f800004　　y1＝ $800000

b2＝ $80　　b1＝ $00000f　　a2＝ $800004

例 3.21：无限制传送数据。

问题：执行指令 MOVE　X1，X0，表示当 A1（A2 或 A0）寄存器指定作为数据源时没有限制。设寄存器的原内容是：

a＝＄00800000000000

a2＝＄00　　a1＝＄800000　　a0＝＄000000

x0＝＄000000

键入：change a ＄00800000000000 x0 ＄000000<return>

display a a2 a1 a0 x0<return>

asm p：＄239 move a1，x0<return>

change pc ＄239 <return>

step<return>

display a a2 a1 a0 x0<return>

结果：a＝＄00800000000000

a2＝＄00　　a1＝＄800000　　a0＝＄000000

x0＝＄800000

3.3.5　位变换单元

位变换单元对 X 或 Y 存贮器操 作数完成位变换操作。

例 3.22：位测试和清除。

问题：执行指令 BCLR＃2，X：＄100，测试和清除单元＄100 的 X 存贮器的位 2。设寄存器的原内容为：

x：＄100　　＄00000f　　sr＝＄0300

键入：change x：＄100 ＄00000f sr＄0300<return>

display x：＄100 sr<return>

asm p：＄240 bclr＃2，X：＄100<return>

change pc＄240<return>

step<return>

display x：＄100 sr<return>

结果：x：＄100　　＄00000b　　sr＝＄0301

注意：单元＄100 的 X 存贮器的位 2 状态反映在状态寄存器 C 位（最低有效位）。并注意＄100 的 X 存贮器的位 2 在测试后被清除。

例 3.23：位测试和更改。

问题：执行指令 BCHG＃3，Y：＄200，测试和更改＄200 的 Y 存贮器的位 3。设寄存器的原内容为：

y：＄200　　＄00000　　sr＝＄0300。

键入：change y：＄200 ＄00000f sr ＄0300<return>

display y：＄200 sr<return>

asm p：＄242 bchg＃3，Y：＄200<return>

change pc ＄242<return>

step<return>

display y：＄200 sr<return>

结果：y：$200　　$000007　　sr＝$0301

3.4　程序控制器

程序控制器如图 3－4 所示，是独立的执行单元，它提供标准程序流-控制源，如程序计数器、状态寄存器和系统堆栈。它还包括一操作模式寄存器，还有一个循环地址寄存器和一个循环计数器，以支持 DSP56001 处理器硬件 DO 循环指令。

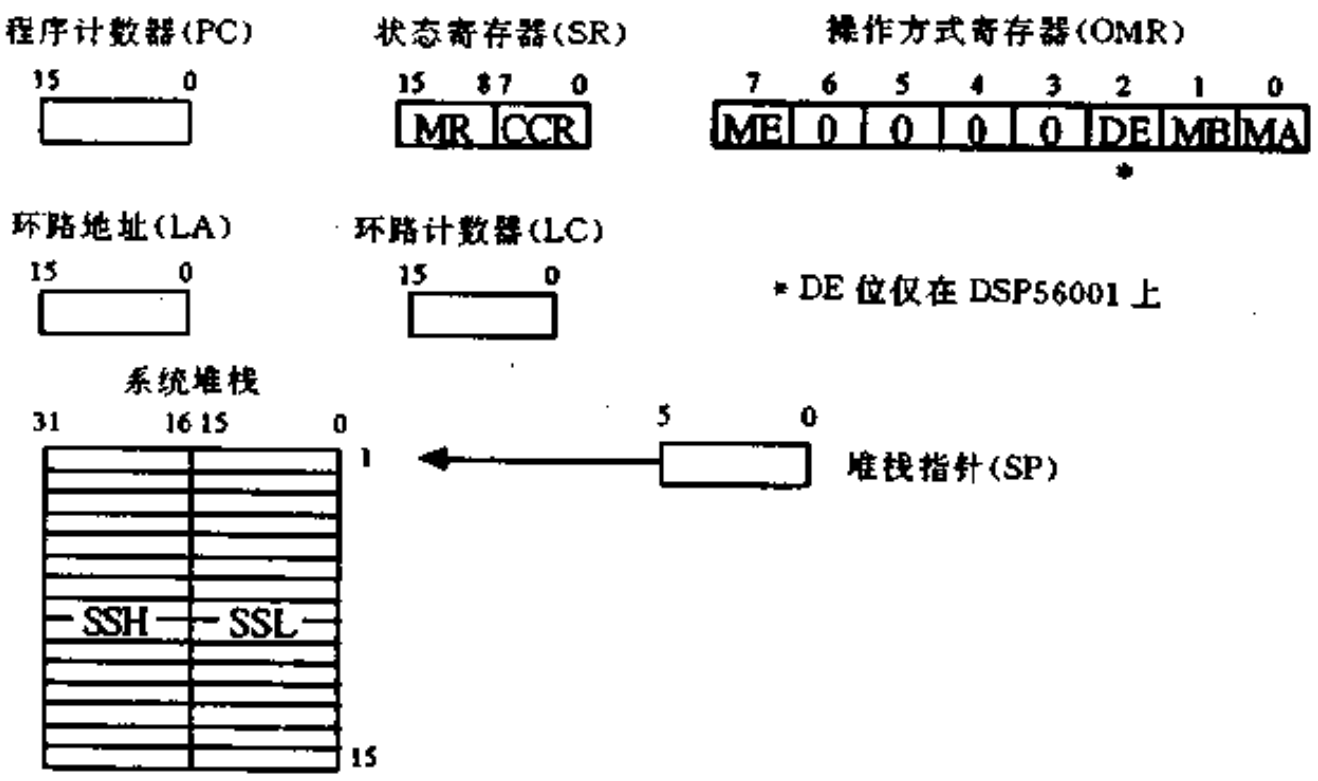


图 3－4　程序控制器

1. 程序计数器寄存器（PC）

16 位 PC 指出下一个指令字、立即数据操作数或立即地址操作数的程序存贮器（P 存贮器）单元。

2. 状态寄存器（SR）

16 位 SR 如图 3－5 所示，由 8 位模式寄存器（MR）和 8 位条件码寄存器（CCR）组成。MR 位于状态寄存器的高 8 位。CCR 位于低 8 位。MR 包含 DSP56001 处理器系统状态信息；CCR 指示运算结果。

3. 系统堆栈（SS）

SS 是一单独的 32×15 内部存贮器，它存贮程序计数器（PC）和状态寄存器（SR）是为了子程序调用和长中断。SS 也可存循环计数器和循环地址寄存器。

4. 堆栈指针（SP）

6 位 SP 如图 3－6 所示，它指示系统堆栈的顶端和指示系统堆栈状态条件，如下溢、空、满和溢出。

5. 循环计数器（LC）

16 位 LC 指定硬件 DO 循环指令或硬件 REPEAT 指令要重复的次数。

6. 循环地址寄存器（LA）

16 位 LA 指出硬件 DO 循环中最低指令字的置位。

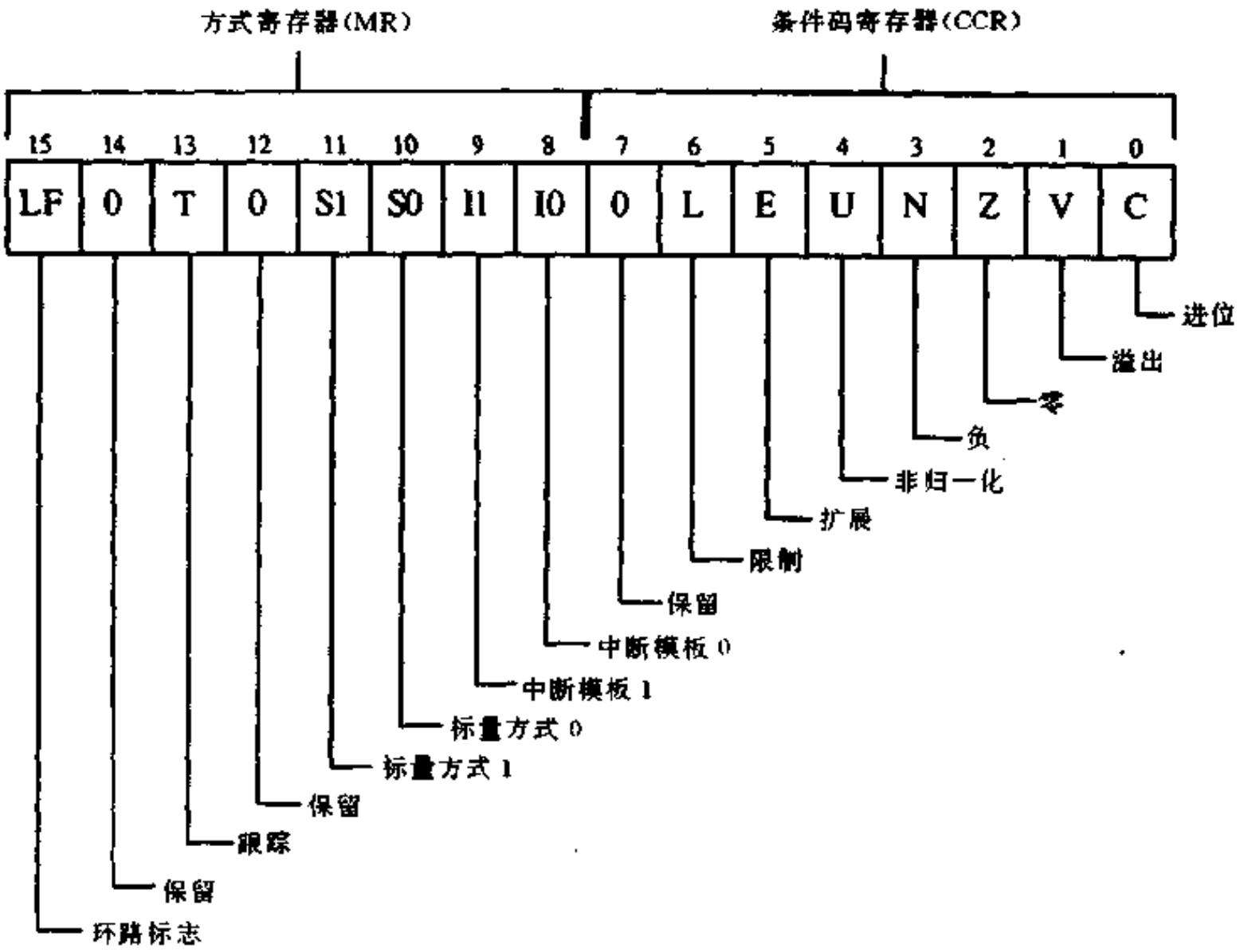


图 3-5 状态寄存器

7. 操作模式寄存器（OMR）

8 位 OMR 定义 DSP56001 处理器当前操作模式。它定义各种存贮器如何映射和开始运行。OMR 格式示于图 3-7。表 3-3 综合了 DSP56001 处理器的操作模式。表 3-4、表 3-5 和图 3-8 表示程序和数据存贮空间。

表 3-3 操作模式综合

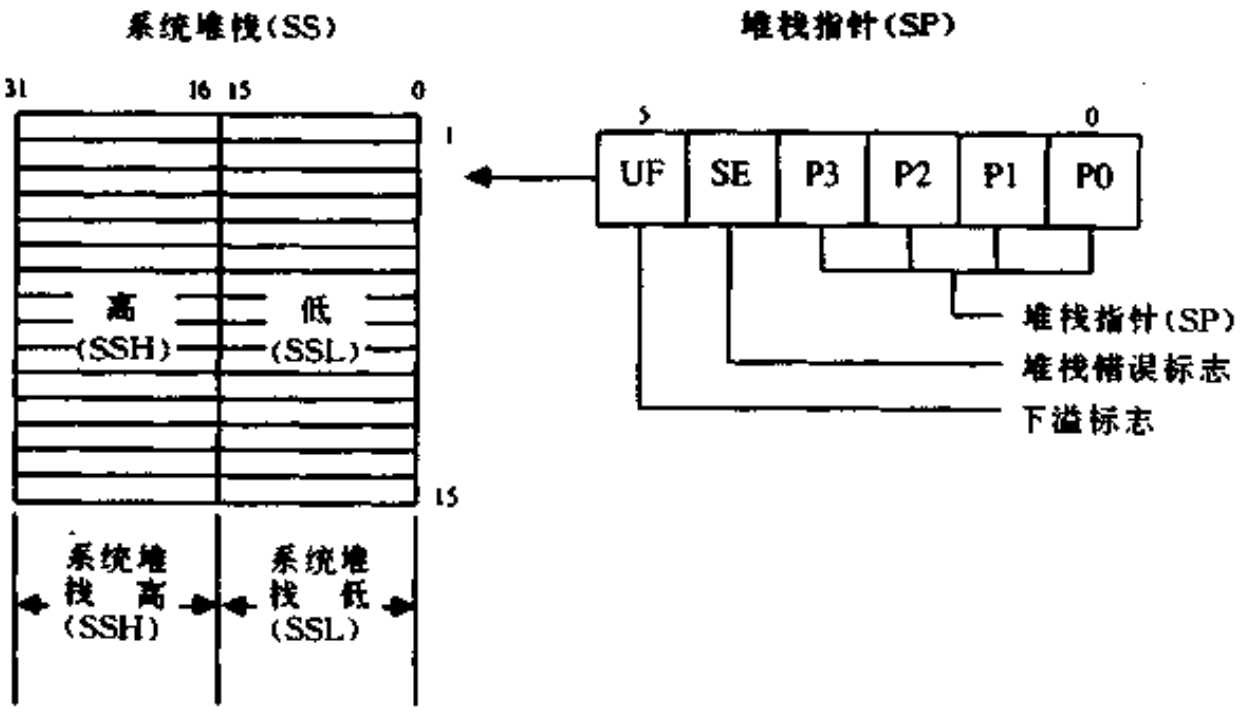
操作模式	MA	MB	说　明
0	0	0	PRAM 允许，在 $0000 复位（内部）
1	0	1	特定引导程序模式，PRAM 装入后模式 2 自运被选
2	1	0	PRAM 允许，在 $E000 复位（外部）
3	1	1	PRAM 允许，在 $0000 复位（外部）

表 3-4 程序存贮空间

操作模式	MB	MA	说　明
0 和 2	X	0	内部 RAM：$0000～$01FF
			外部：$0200～$FFFF
3	1	1	外部：$0000～$FFFF

表 3-5　数据存贮空间

OROM 允许 DE	Y 数据存贮空间映像	X 数据存贮空间映像
0	内部 RAM：$0000～$00FF 外部：$0100～$FFFF	内部 RAM：$0000～$00FF 外部：$0100～$FFBF 片上外围：$FFC0～$FFFF
1	内部 RAM：$0000～$00FF 内部 ROM：$0100～$01FF 外部：$0200～$FFFF	内部 RAM：$0000～$00FF 内部 ROM：$0100～$01FF 外部：$0200～$FFBF 片上外围：$FFC0～$FFFF



使用：

- 子程序
- 长中断
- 硬件"DO LOOPS"
- 不为数据存贮推荐

UF SE P3 P2 P1 P0

1	1	1	1	1	0	← 重拉后堆栈下溢条件
1	1	1	1	1	1	← 堆栈下溢条件
0	0	0	0	0	0	← 堆栈空(复位)，拉升引起下溢
0	0	0	0	0	1	← 堆栈单元 1
·	·	·	·	·	·	·
·	·	·	·	·	·	·
·	·	·	·	·	·	·
0	0	1	1	1	0	← 堆栈单元 14
0	0	1	1	1	1	← 堆栈单元 15，推引起溢出
0	1	0	0	0	0	← 堆栈溢出条件
0	1	0	0	0	1	← 重推后堆栈溢出条件

堆栈错误例外

图 3-6　堆栈指针

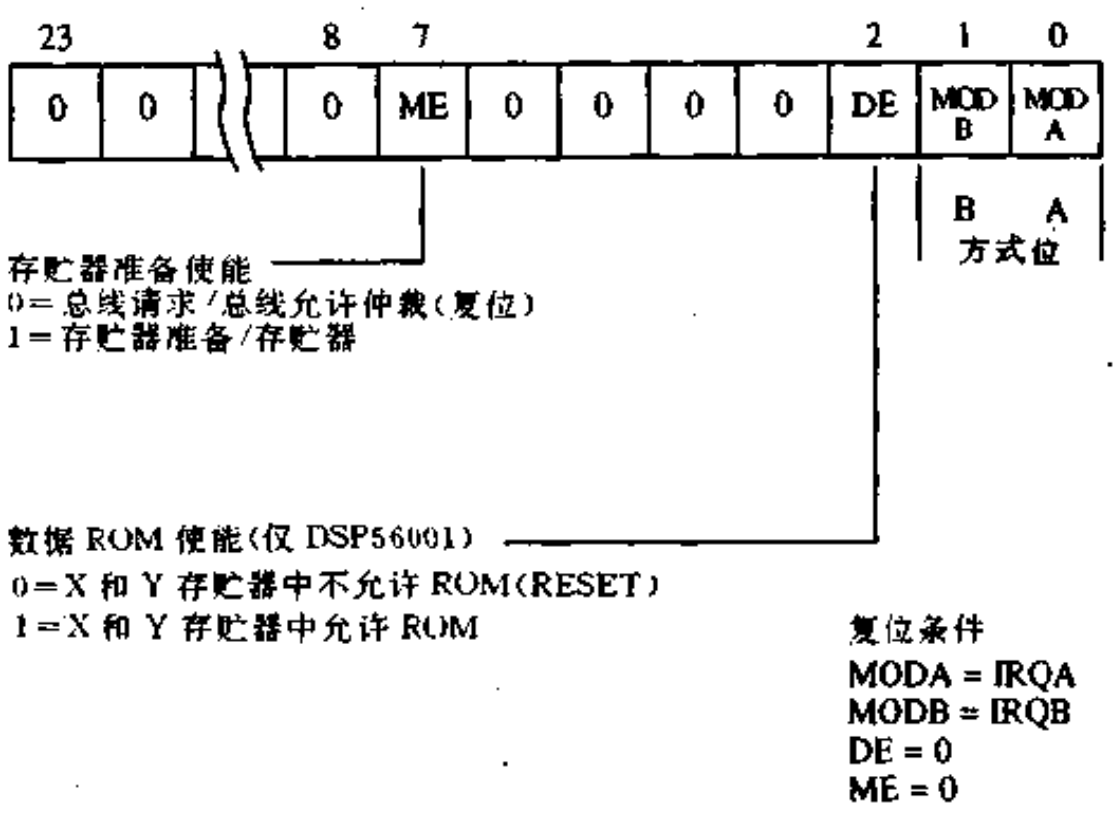


图 3-7　操作模式寄存器

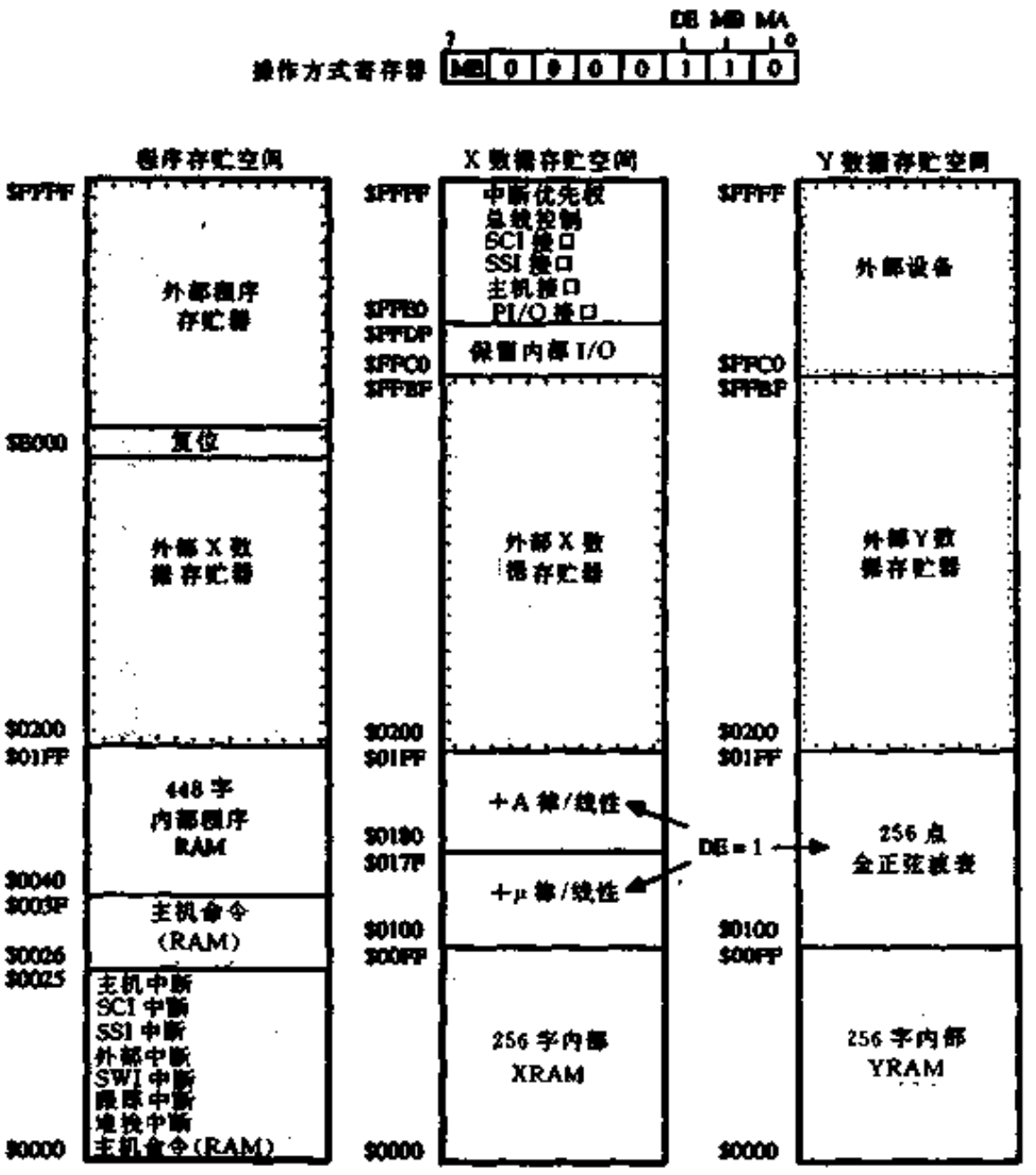


图 3-8 (a)　DSP56001 存贮器映像模式 0—正常扩展和内部复位

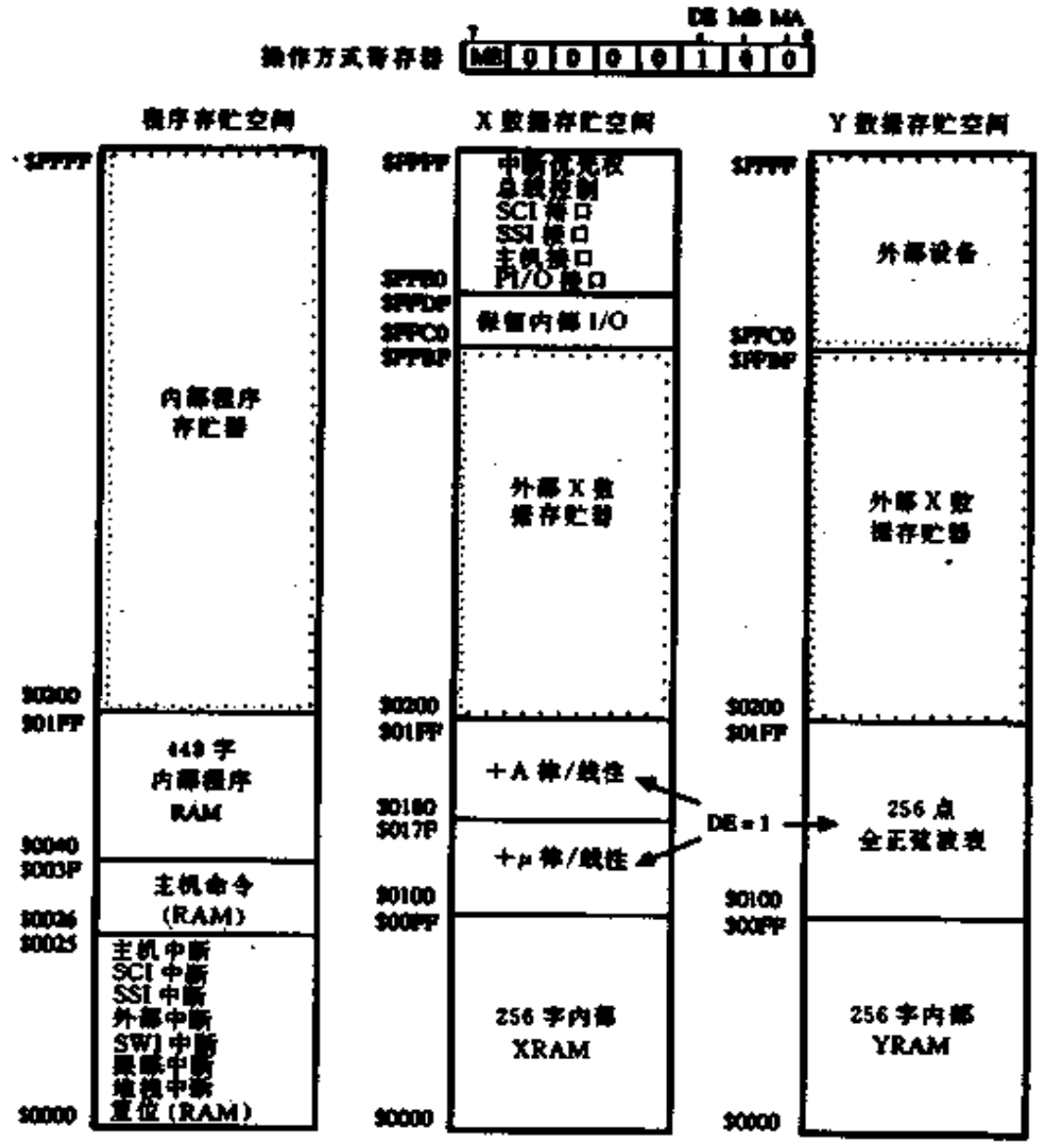


图 3－8（b）　DSP56001 存贮器映像模式 2—正常扩展和外部复位

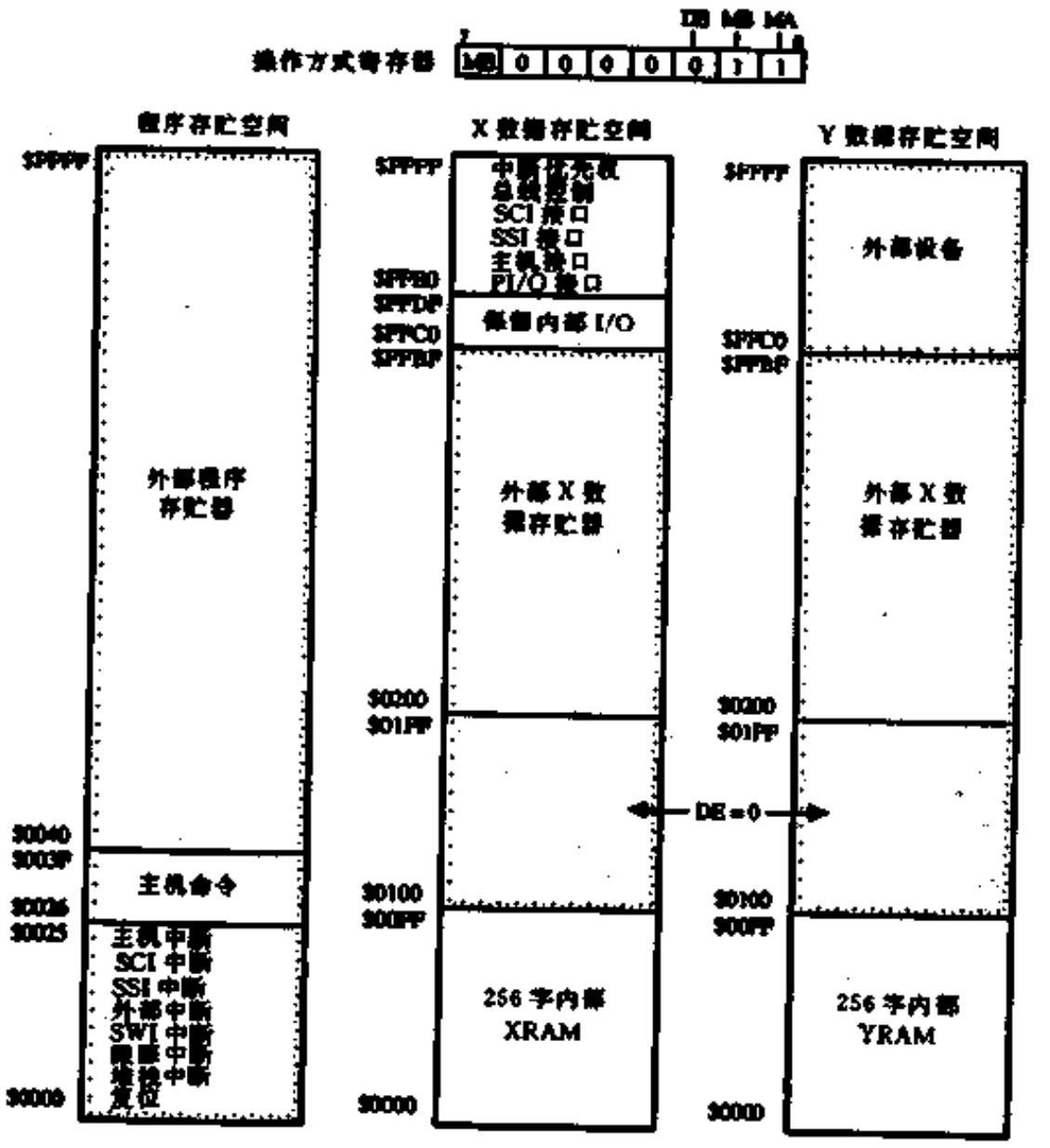


图 3－8（c）　DSP56001 存贮器映像模式 3—开发模式外部程序存贮器

3.5 地址产生单元

地址产生单元是一独立执行单元，它产生地址以在X、Y或P存贮器中定位数据操作数。它提供14种寻址方式并用3种地址产生算法。其主要部件如图3-9所示，包括24个16位地址寄存器，2个16位地址算术单元和3个地址输出复用器。

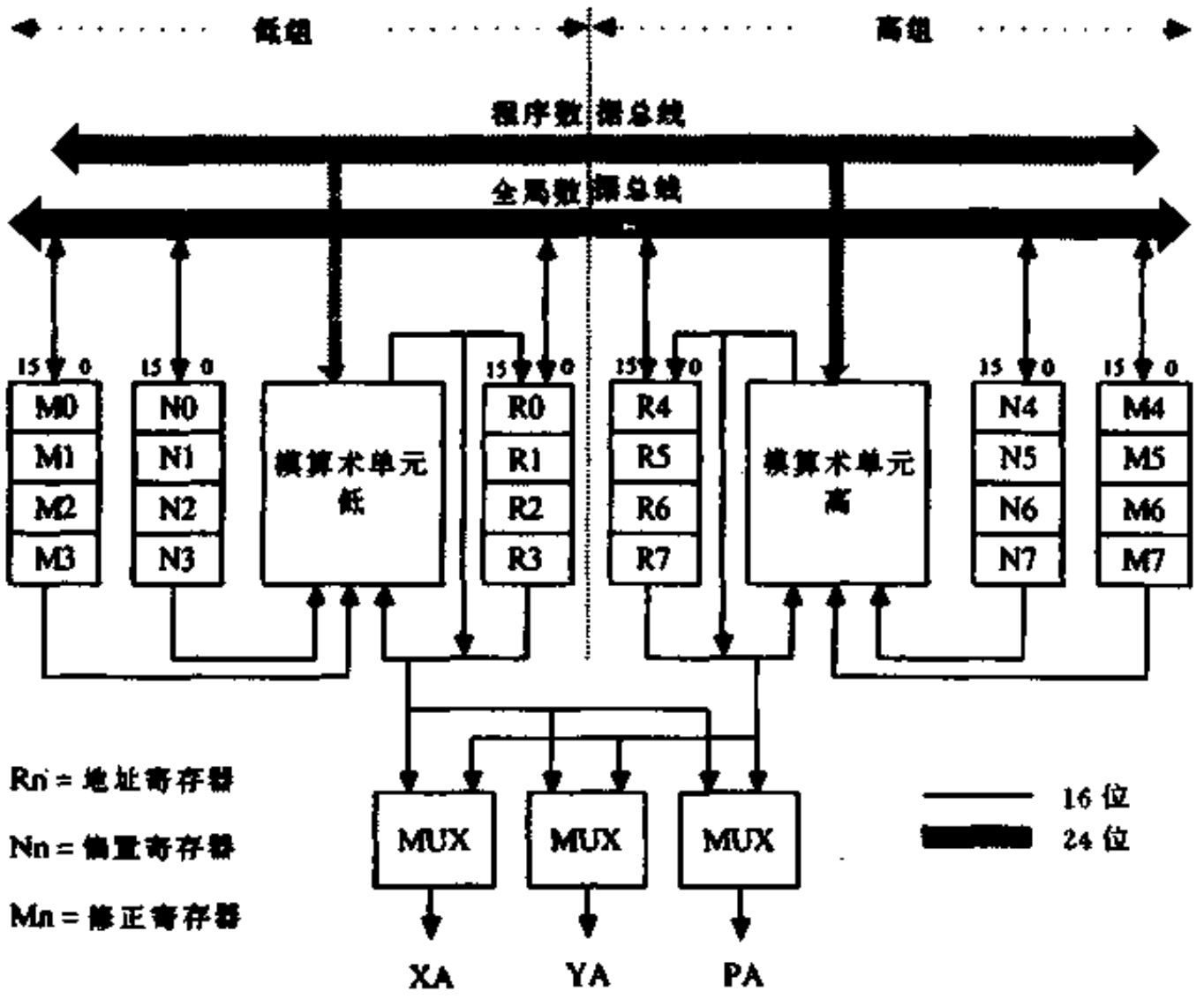


图3-9 地址算术单元

1. 地址寄存器

24个寻址寄存器如图3-10所示，组成3组8个寄存器：

地址寄存器 Rn n=0，1，…，7

偏置寄存器 Nn n=0，1，…，7

变址寄存器 Mn n=0，1，…，7

每个地址寄存器Rn有相应的偏置寄存器Nn，和相应的变址寄存器Mn，三者数量n相同，Rn作为地址指针，指示操作数在存贮器中的单元。Nn为地址寄存器的偏置更新提供偏置值。Mn选择更新地址寄存器时执行的地址算法方式。

2. 地址算术单元

二个地址算术单元完成DSP56000/1处理器寻址方式的地址运算。用3种地址算法：线性、模数、反向进位。线性算法用于标准的MPU型寻址。模数算法对第六章中将讲到的循环

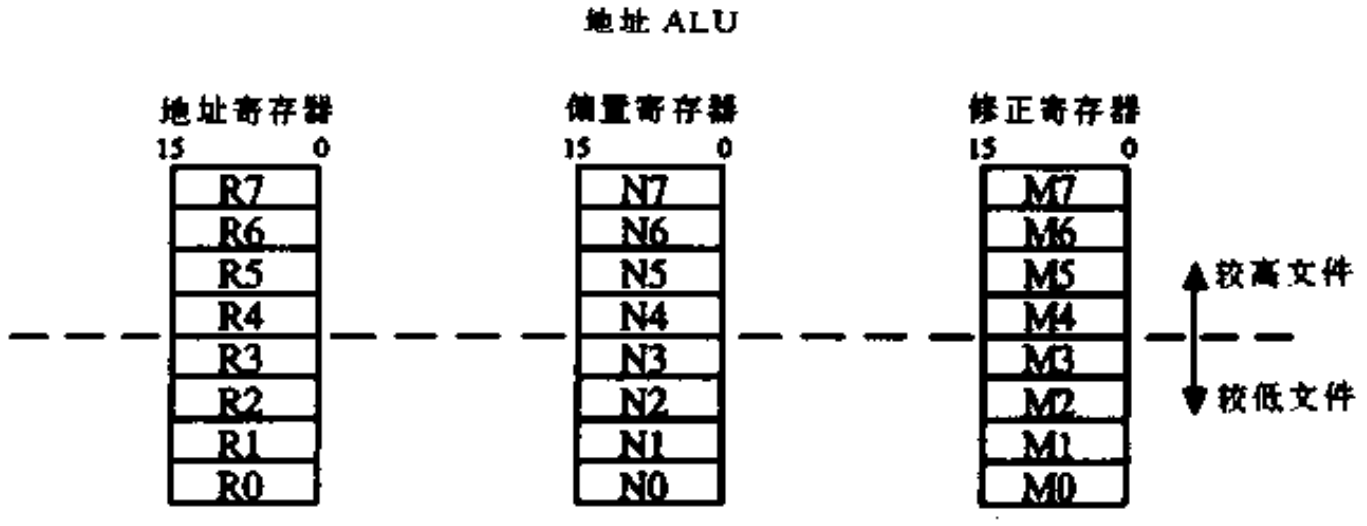


图 3-10 地址寄存器

贮存器有用。反向进位法对快速傅里叶变换的排序数据有用。表 3-6 表示偏置寄存器如何选择要完成的地址算法方式。

表 3-6 变址型编码综合

变址器	地址算法型式	变址器	地址算法型式
0000000000000000	反向进位（位反转更新）	0111111111111111	模 32768
0000000000000001	模 2	1000000000000000	无定义
0000000000000010	模 3	⋮	⋮
0000000000000011	模 4	1111111111111110	⋮
⋮	⋮	1111111111111111	线性（模 65536）
0111111111111110	模 32767		

3. 地址输出复用器

3 个地址输出复用器允许任何地址寄存器 Rn 用来指示任何存贮空间。

3.6 寻 址 方 式

DSP56001 处理器指令由一个或二个 24 位字组成：操作字和扩展字。操作字包含一个 8 位操作码段和 16 位数据总线传送段。操作码段包含指令操作码及其源和目的寄存器操作数。数据总线传送段提供在 XD 和 YD 总线上数据传送的方向和有效地址。有效地址指定寻址方式，对某些寻址方式，有效地址还将进一步指定地址寄存器 Rn。

寻址方式分为三类：寄存器直接、专用、寄存器间接。寄存器间接方式要求附加的变址信息，这在变址寄存器中指定。寄存器直接和间接方式都与单字指令相联。专用寻址方式与一字或二字指令相联。

3.6.1 寄存器直接寻址方式

这种方式指定操作数在一个（或更多个）数据输入寄存器、地址寄存器或控制寄存器中。

例 3.24：传送寄存器数据到 24 位累加器寄存器。

问题：执行指令 MOVE X1，A0，用寄存器直接寻址方式把 X1 数据输入寄存器的内容移

人 A0。设执行前寄存器内容为：

x1＝$1111111　　a0＝$000000

键入：change x1 $111111 a0 0<return>

display x1 a0<return>

asm p：$100 move x1、a0<return>

change pc $100<return>

step <return>

display x1 a0＝$111111

结果：x1＝$111111　　a0＝$111111

3.6.2 专用寻址方式

这种方式指定操作数或操作数地址在一指令段中，或它们隐含地参考一操作数。专用寻址方式包括：立即数据、立即短数据，绝对地址、绝对短地址、I/O 短地址、短跳地址和隐含。绝对短、I/O 短和短跳、立即短寻址方式与单字指令相联。绝对地址和立即数据寻址方式与双字指令相联。

1. 立即数据寻址方式

它指出在指令扩展字中的 24 位操作数。

例 3.25：传送立即数据到 24 位累加器寄存器。

问题 1：执行指令 MOVE# $818181，A0，传送即时值 $818181 到 A0，设寄存器原内容为：

a＝$00000000000000

a2＝$00　　a1＝$000000　　a0＝$000000

键入：change a 0<return>

display a a2 a1 a0<return>

asm p：$101 move # $818181，a0<return>

change pc $101<return>

step<return>

display a a2 a1 a0<return>

结果：a＝$00000000818181

a2＝$00　　a1＝$000000　　a0＝$818181

例 3.26：传送立即数据到 56 位累加器。

问题：执行指令 MOVE* $818181，A 和 MOVE* $121212，B。设寄存器原内容为：

a＝$00000000000000

a2＝$00　　a1＝$000000　　a0＝$000000

b2＝$00　　b1＝$000000　　b0＝$000000

键入：change a 0 b 0<return>

display a a2 a1 a0 b b2 b1 b0<return>

asm p：$103 move # $818181，A<return>

asm p：$105 move# $121212，b，<return>

change pc $103<return>

step 2<return>

display a a2 a1 a0 b b2 b1 b0<return>

结果：a＝$ff818181000000

a2＝$ff　　a1＝$ $818181　　a0＝$000000

b＝$00121212000000

b2＝$00　　b1＝$121212　　b0＝$000000

2. 立即短寻址方式

这种方式在指令操作字中指出8位或12位立即数据操作数。若目的寄存器是A2、A1、A0、B1、B0、R0～R7、N0～N7寄存器之一，则立即数据解释为无符号整数。立即数据送至最高有效位为零的目的的最低有效位。若目的寄存器是X1、X0、Y1、Y0或A、B累加之一，则立即数据解释为有符号小数。

例3.27：传送立即短数据到24位寄存器。

问题：执行指令MOVE＃$81，A1和MOVE＃481，X1。

设寄存器原内容为：

a＝$000000000000

a2＝$00　　a1＝$000000　　$0＝$000000

x1＝$000000　　x0＝$000000

键入：change a 0 x 0<return>

display a a2 a1 a0 x1 x0<return>

asm p：$107 move＃$81，a1<return>

asm p：$108 move＃$81，x1<return>

change pc $107<return>

step2<return>

display a a2 a1 a0 x1 x0<return>

结果：a＝$000000810000

a2＝$00　　a1＝000081　　a0＝000000

x1＝$810000　　x0＝$000000

例3.28：传送立即短数据到累加器。

问题：执行二个指令MOVE＃12，A和MOVE＃81，B设累加器原内容全为0。

键入：change a 0 b 0 <return>

display a a2 a1 a0 b b2 b1 b0<return>

asm p：$109 move＃ $12，a<return>

asm p：$10a move＃ $81，b <return>

change pc $109<return>

step 2<return>

display a a2 a1 a0 b b2 b1 b0<return>

结果：a＝$00120000000000

a2＝$00　　a1＝$120000　　a0＝$000000

b＝$ff810000000000

b2＝$ff　　b1＝$810000　　b0＝$000000

3. 绝对地址寻址方式

这种方式用于指令扩展字中的16位地址操作数作为数据操作数单元的指针。

例 3.29：把由绝对地址操作数指示的数据操作数传送到 A 累加器。

问题：执行指令 MOVE X：$2000，A0。设 A 和位于 $2000 的 X 存贮器的原内容为：

a= $ 00000000000000

a2= $ 00　　a1= $ 000000　　a0= $ 000000

x：$2000　　$121212

键入：change x：$2000 $121212 a 0<return>

display x：$2000 a a2 a1 a0<return>

asm p：$10b move x：$2000，a0<return>

change pc $10b<return>

step<return>

display x：$2000 a a2 a1 a0<return>

结果：a= $ 00000000121212

a2= $ 00　　a1= $ 000000　　a0= $ 121212

x：$2000 $121212

4. 绝对短寻址方式

此方式用于指令操作字中的 6 位立即地址操作数，对数据操作数形成 16 位指针。6 位立即地址操作数由扩展零形成 16 位指针。

例 3.30：把累加器寄存器传送到由绝对短地址操作数指定的存贮器中。

问题：执行指令 MOVE A1，X：$2 把 A1 的内容传送到 6 位绝对短地址 $02 指示的位于 $0002 的 X 存贮器中。设寄存器原内容：

a= $ 00818181000000

a2= $ 00　　a1= $ 818181　　a0= $ 00000

x：$0002　　$000000

键入：change x：$2 0 a 0 a1 $818181<return>

display x：$2 a a2 a1 a0<return>

asm p：$10d move a1，x：$2<return>

step<return>

display x：$2 a a2 a1 a0<return>

结果：a= $ 00818181000000

a2= $ 00　　a1= $ 818181　　a0= $ 000000

x：$0002　　$818181

5. I/O 短寻址方式

它与绝对短寻址方式相似。绝对短寻址方式中的 6 位立即地址操作数的 1 扩展分别形成 X 或 Y 存贮器的 I/O 段（$FFC0～$FFFF）的 16 位指针。它使用 bit manipulation 和 Move peripheral data 指令。

例 3.31：把累加器寄存器传送至由 I/O 短地址操作数指示的 I/O 存贮器中。

问题：执行指令 MOVEP A1，X：$FFFE 把 A1 的内容送到由 6 位 I/O 短地址 $3E 指示的位于 $FFFE 的 X 存贮器 I/O 中。设寄存器原内容为：

a= $ 00112233000000

a2= $ 00　　a1= $ 112233　　a0= $ 000000

x：$fffe　　$000000

键入：change　x：$fffe 0 a 0 a1 $112233<return>

display x：$fffe a a2 a1 a0 <return>

asm p：$10e move p a1，x：$fffe<return>

change pc $10e <return>

step<return>

display x：$fffe a a2 a1 a0<return>

结果：a＝$00112233000000

a2＝$00　　a1＝$112233　　a0＝$000000

x：$fffe $002233

6. 短跳寻址方式

此方式用位于指令操作字中的12位立即跳操作数，形成16位“跳”操作数。此12位跳操作数是扩展0～16位，并代替程序计数器（PC）的内容。

例3.32：短跳到P存贮器单元。

问题：执行指令JMP $222跳到单元$0222的P存贮器。设PC的原内容为：

pc＝$011f

键入：change pc $10f <return>

display pc<return>

asm p：$10f jmp $222<return>

change pc $10f<return>

step<return>

display pc <return>

结果：pc＝$0223

注意指令JMP$222不仅跳到单元$0222的P存贮器，而且还对包含在此单元的指令取数。所以执行此指令后，PC增加1（PC＝$0222+1＝$0223）指出下一个从P存贮器要取数的指令。

7. 隐含寻址方式

此方式被某些指令用来隐含访问程序控制器寄存器。程序控制器寄存器隐含在指令的助记符和操作码中。

3.6.3　地址寄存器间接寻址方式

在此方式中，指令操作字指定地址寄存器R_n去指示存贮器中的操作数。指令操作字也可在指令执行前后指定要完成的地址计算，例如以1记入后递增、以偏置记入后递减。

每个地址寄存器Rn同偏置寄存器Nn和变址寄存器Mn相联系。偏置寄存器Nn包含可加到Rn上以更新其内容的偏置值。变址寄存器Mn当地址寄存器Rn更新时指定要完成的地址算术型式。每个变址寄存器Mn在处理器复位时置为$FFFF，以指定线性寻址算法作为缺省地址算法。

1. 不更新地址寄存器间接寻址方式

在此地址寄存器间接寻址方式中，地址寄存器Rn指出存贮器中的操作数。Rn中的内容不变。

例 3.33：把累加器寄存器传送到由地址寄存器指定的存贮器。地址寄存器的内容不被改变。

问题：执行指令 MOVE B1，Y：(R0)。设原寄存器内容为：

b1＝$010101

r0＝$1200　　n0＝$0000　　m0＝$ffff

y：$1200　　$000000

键入：change b1 $010101 y：$1200 0<return>

change r0 $1200 n0 0 m0 $ffff<return>

display b1 r0 n0 m0 y：$1200<return>

asm p：$110 move b1，y：(r0) <return>

change pc $110<return>

step<return>

display b1 r0 n0 m0 y：$1200<return>

结果：b1＝$010101

r0＝$1200　　m0＝$0000　　m0＝$ffff

y：$1200 $010101

2. 后递增 1 的地址寄存器间接寻址方式

在此方式中，地址寄存器指出存贮器中的操作数。操作数地址使用后，地址寄存器 Rn 的内容加 1 并将结果存入 Rn。

例 3.34：把累加器寄存器传送到地址寄存器所指示的存贮器，地址寄存器内容加 1。

问题：执行指令 MOVE B0,Y：(R1) ＋ 。设寄存器原内容为：

b0＝$010101

r1＝$1000　　n1＝$0000　　m1＝$ffff

y：$1000　　$000000

键入：change b0 $01010ly：$1000 0<return>

change r1 $1000 n1 0 m1 $ffff<return>

display b0 r1 n1 m1 y：$1000<return>

asm p：$111 move b0，y：(r1) ＋< return>

change pc $111<return>

display b0 r1 n1 m1 y：$1000<return>

结果：b0＝$010101

r1＝$1001　　n1＝$0000　　m1＝$ffff

y：$1000　　$010101

3. 后递减 1 的地址寄存器间接寻址方式

此方式中地址寄存器 Rn 指出存贮器中的操作数。操作数地址使用后，Rn 的内容减 1 并将结果存于 Rn 中。

例 3.35：把数据输入寄存器传送到地址寄存器所指示的存贮器。地址寄存器内容减 1。

问题：执行指令 MOVE Y0,X：(R2) － ，把 Y0 内容传送到 R2 所指示的 X 存贮器中。传送完成后，R2 被减 1。设原内容为：

y0＝$010101

r2＝$1001　n2＝$0000　m2＝$ffff

x：$1001 $000000

键入：change y0 $010101 0<return>

change r2 $1001 n2 0 m2 $ffff<return>

display y0 r2 n2 m2 x：$1001<return>

asm p：$112 move y0，x：(r2) <return>

change pc $112<return>

step<return>

display y0 r2 n2 m2 x：$1001<return>

结果：y0= $010101

r2= $1000 n2= $0000 m2= $ffff

x：$1001 $010101

4. 后递增偏置 Nn 的地址寄存器间接寻址方式

此方式中，地址寄存器 Rn 指示存贮器中的操作数。使用操作数地址后，Rn 加上包含在偏置寄存器 Nn 中的偏置成为 Rn 的内容。偏置寄存器内容 Nn 不变。

例 3.36：把数据输入寄存器传送到地址寄存器指示的存贮器。地址寄存器内容加上相关偏置寄存器中的偏置值。

问题：执行指令 MOVE X0,Y:(R3) + N3 。设数据输入寄存器 X0、地址寄存器 R3、偏置寄存器 N3、变址寄存器 M3 和 $1000 及 $1003 的存贮器原内容为：

x0= $010101

r3= $1000 n3= $0003 m3= $ffff

y：$1000 $000000 Y：$1003 $000000

键入：change x0 $010101<return>

change y：$1000 0 y：$1003 0<return>

change r3 $1000 n3 3 m3 $ffff<return>

display x0 r3 n3 m3 y：$1000 y：$1003<rituin>

asm p：$113 move x0，y：(r3) + n3 < return>

change pc $113<return>

step<return>

display x0 r3 n3 m3 y:$1000 y：$1003<return>

结果：x0= $010101

r3= $1003 n3= $0003 m3= $ffff

y：$1000 $010101 y：$1003 $000000

5. 后递减偏置 Nn 的地址寄存器间接寻址方式

此方式中，Rn 指示存贮器中的操作数。使用操作数地址后，自 Rn 内容减去包含在偏置寄存器 Nn 中的偏置。偏置寄存器内容 Nn 不变。

例 3.37：把由地址寄存器指示的存贮器传送到累加器寄存器。地址寄存器内容由减去相关偏置寄存器中偏置值来更新。

问题：执行指令 MOVE Y:(R4) — N4 ，A0 设有关寄存器原内容为：

a0= $000000

r4= $1004 n4= $0004 m4= $ffff

y：$1000 $222222 y:$1004 $111111

键入：change a0 0<return>

change y：$1004 $111111 y：$1000 $222222<return>

change r4 $1004 n4 4 m4 $ffff，<return>

display a0 r4 n4 m4 y:1000 y:$1004 < return >

asm p：$114 move y:(r4) − n4,a0 < return >

change pc $114<return>

step<return>

display a0 r4 n4 m4 y:$1000 y:$1004 < return >

结果：a0＝$111111

r4＝$1000 n4＝$0004 m4＝$ffff

y：$1000 $222222 y：$1004 $111111

6. 由偏置 Nn 定位的地址寄存器间接寻址方式

在此方式中，Rn 被加到偏置寄存器 Nn 以形成在存贮器中操作数的指针。Rn 和 Nn 内容不变。

例 3.38：将数据输入寄存器传送到由地址寄存器及其相关偏置寄存器之和指向的存贮单元，地址寄存器和偏置寄存器的内容不变。

问题：执行指令 MOVE X1，Y：(R5+N5)。设 X1、R5、N5、M5 和 $1300 与 $1306 的 Y 存贮器原内容为：

x1＝$010101

r5＝$1300 n5＝$0006 m5＝$ffff

y：$1300 $000000 y：$1306 $000000

键入：change x1 $010101 y:$1300 0 y:$1306 0 < return >

change r5 $1300 n5 $0006 m5 $ffff < return >

display x1 r5 n5 m5 y:$1300 y:$1306 < return >

asm p:$115 move x1,y:(r5 + n5) < return >

change pc $115 < return >

display x1 r5 n5 m5 y:$1300 y:$1306 < return >

结果：x1＝$010101

r5＝$1300 n5＝$0006 m5＝$ffff

y：1300 $000000 y：1306 $010101

7. 先递减 1 的地址寄存器间接寻址方式

此方式中，地址寄存器 Rn 指示存贮器中的操作数。但操作数被访问前，地址寄存器内容 Rn 减 1，即 Rn＝Rn−1。

例 3.39：把由先递减的地址寄存器所指示的存贮器传送到累加器寄存器。地址寄存器内容不变。

问题：执行指令 MOVE X:(R6)，A1。设寄存器原内容为：

a＝$00000000000000

a2＝$00 a1＝$000000 a0＝$000000

r6＝$1001 n6＝$0000 m6＝$ffff

x：$1000 $121212 x：$1001 $343434

键入：change a 0 x:$1000 $121212 x:$1001 $343434 < return >

```
change r6 $1001 n6 0 m6 $ffff < return >
display a a2 a1 a0 r6 n6 m6 x: $1000.. $1001 < return >
asm p: $116 move x: -(r6),a1 < return >
change pc $116 < return >
step < return >
display a a2 a1 a0 r6 n6 m6 x: $1000.. $1001 < return >
```

结果：a＝$00121212000000

a2＝$00　　a1＝$121212　　a0＝$000000

r6＝$1000　　n6＝$0000　　m6＝$ffff

x：$1000　　$121212

3.6.4 DSP56000/1 寻址方式总结

DSP56000/1 处理器寻址方式总结如表 3－7。

表 3－7　DSP56000/1 寻址方式总结

寻址方式	汇　编	举　例
寄存器直接		move x1，a0
专用		
立即数据	#×××××× ×	move # $818181，a0
立即短数据	#××	move # $81，a1
		move # $81，x1
		move # $12，a
		move # $81，b
绝对地址	××××	move x：$2000，a0
绝对短地址	aa	move a1，x：$2
I/O 短地址	pp	movep a1，x：$ffe
短跳地址	×××	jmp $222
隐含		
专用寄存器间接		
不更新	(Rn)	move b1，y：(r0)
后递增 1	(Rn)＋	move b0，y：(r1)＋
后递减 1	(Rn)－	move y0，x：(r2)－
后递增偏置 Nn	(Rn)＋Nn	move x0，y：(r3)＋n3
后递减偏置 Nn	(Rn)－Nn	move y：(r4)－n4，a0
偏置 Nn 定位	(Rn＋Nn)	move x1，y：(r5＋n5)
先递减 1	－(Rn)	move x：－(r6)，a1

第四章　DSP56001 指令集

本章详细描述 DSP56001 处理器的指令集。它包括 62 条指令。每条指令的长度是一或二个字长。用 30 条指令可指定一或二个要执行的数据转移，且并行执行指令的操作码。这些数据转移称作并行传送操作。指令集分为以下几组：传送、算术、逻辑、位变换、循环和程序控制。指令集使很多信号处理的修正指令很有特色，如 CMPM，NORM，RND，MACR，SUBL，SUBR，ADDL，ADDR，DO 和 REP。

本章首先描述指令格式和并行移动操作，然后提出指令集组。

4.1　指 令 格 式

每条 DSP56001 指令由一或二个字组成。第一个字指定指令及其长度。第二个字可包含一绝对地址或立即数据。

4.1.1　一字指令

所有 DSP56001 处理器的寻址方式，除立即数据和绝对地址寻址方式外，均有一字指令可供使用。对典型的一字指令汇编语言源码可分成 4 段：操作码段、操作数段、X 总线数据转移段和 Y 总线数据转移段。

操作码：操作码段典型地指示要完成的数据 ALU 操作，它亦可指定传送、地址产生或程序控制操作。

操作数段：它指定操作码要用的操作数。

数据转移段：它指定通过 X 和 Y 总线选取的数据转移和相联系的寻址方式。

例 4.1：表 4-1 是一字指令的例子。

表 4-1　一字指令实例

操作码	操作数	X 总线数据转移	Y 总线数据转移
MOVE	X1,A		
MOVE		X0,X:(R2)−	
MOVE			X0,Y:(R3)+N3
RND	A	X:(R0)+,R0	Y:(R4)+,Y0
MAC	X0,Y0,B	X:(R0)+,R0	Y:(R4)+,Y0
ADD	X,B	X:(R6)+,X1	Y,Y:(R7)−

4.1.2　二字指令

在二字指令中，有绝对地址和立即数据寻址方式可供使用以提供 16 位绝对地址或 24 位立即数据值。

例 4.2：以下为二字指令举例：

```
MOVE   #$123456，A0
MOVE   X：$2000，A0
```

24 位立即数据操作数（$123456）和 16 位地址操作数（$2000）分别在指令的第二字中编码。

4.2 并行传送操作

30 个 DSP56001 指令在一个指令周期中，以指令操作码可并行确定一或二个平行数据传送操作。数据可从寄存器到寄存器、寄存器到存贮器、存贮器到寄存器间传送。若 A 或 B 累加器是源寄存器，则随数据转移可能发生位移或限幅。若 A 或 B 累加器是目的寄存器，随着数据转移会自动产生符号扩展和最低有效位填零。当一个指令中执行二个数据转移时，允许重复的源，但不允许重复的目的。

例 4.3：

(1) 允许有相同源的指令转移到几个目的：

```
SUBL    B，A    B，X：(R0)    B，Y：(R4)
```

(2) 不允许几个源的指令转移到相同目的。

```
ADD     B，A    X1，B    Y1，A（不允许）
```

4.3 并行传送型式

并行传送型式是：

(1) 立即短数据传送。

(2) 寄存器到寄存器传送。

(3) 地址寄存器更新。

(4) X 或 Y 存贮器传送。

(5) X 或 Y 存贮器和寄存器传送。

(6) L 存贮器传送。

(7) XY 存贮器传送。

若想用 DSP56000/1 示范软件包含以下各节内容，则可用第一章 1.5.1 节第 1 和 6 步或 1.5.2 节的第 1～3 步装入软件。然后从示范软件菜单选择内容。

若不想用该软件，则用第二章 2.3.1 节第 1、2 和 3 步或 2.3.2 节第 1、5 和 6 步装入 SIM56000 模拟软件。

4.3.1 立即短数据传送

并行传送操作转移 8 位立即短操作数到目的寄存器。8 位操作数可解释为无符号整数或带符号小数，决定于目的寄存器。

1. 无符号整数立即短操作数

若目的寄存器是 A2，A1，A0，B2，B1，B0，R0..R7，或 N0..N7，则立即短操作数解

释为无符号整数。8 位数据存入目的寄存器的最低有效位，而目的寄存器的高位自动复位到 0。

例 4.4：指令 ADD B，A 和立即短数据传送 #81，B0 并行执行。设 A 和 B 原内容为：

a= $ 00111111000000

a2= $ 00　　a1= $ 111111　　a0= $ 000000

b= $ 00222222ff0000

b2= $ 00　　b1= $ 222222　　b0= $ ff0000

键入：
```
change a $ 00111111000000<return>
change b $ 00222222ff0000<return>
display a a2 a1 a0 b b2 b1 b0<return>
asm p: $ 300 add b, a # $ 81, b0<return>
change pc $ 300<return>
step<return>
display a a2 a1 a0 b b2 b1 b0 <return>
```

结果：a= $ 00333333ff0000

a2= $ 00　　a1= $ 333333　　a0= $ ff0000

b= $ 00222222000081

b2= $ 00　　b1= $ 222222　　b0= $ 000081

2. 带符号小数立即短操作数

若目的寄存器是 X0，X1，Y0，Y1，A 或 B，则 8 位立即短操作数解释为带符号小数。8 位操作数存入目的寄存器最高有效位，而其中低位自动复位到 0。

例 4.5：指令 ADD B，A 和立即短数据传送 #81，B 并行执行。设 A 和 B 原内容为：

a= $ 00111111000000

b= $ 00222222222222

键入：
```
change a $ 00111111000000<return>
change b $ 00222222222222<return>
display a a2 a1 a0 b b2 b1 b0<return>
asm p: $ 301 add b, a # $ 81, b<return>
change pc $ 301<return>
step<return>
display a a2 a1 a0 b b2 b1 b0 <return>
```

结果：a= $ 00333333222222

a2= $ 00　　a1= $ 333333　　a0= $ 222222

b= $ ff810000000000

b2= $ ff　　b1= $ 810000　　b0= $ 000000

4.3.2　寄存器到寄存器数据传送

此并行传送操作转移源寄存器的内容到目的寄存器。

例 4.6：指令 ADD B，A 和寄存器到寄存器数据传送操作 X1，B 并行执行。寄存器原内容为：

a= $ 00111111000000

b= $ 00123456111111

X1＝$900000

键入：change a $00111111000000 b $0012345611111<return>

change x1 $900000<return>

display a a2 a1 a0 b b2 b1 b0 x1 <return>

asm p: $302 add b, a x1 b<return>

change pc $302<return>

step<return>

display a a2 a1 a0 b b2 b1 b0 x1<return>

结果：a＝$00234567111111

a2＝$00　　a1＝$234567　　a0＝$111111

b＝$ff900000000000

b2＝$ff　　b1＝$900000　　b0＝$000000

x1＝$900000

例 4.7：两个指令 ADD B，A 和 ADD X1，A 分别与寄存器到寄存器数据传送操作 B，X1 和 B，R0 一起执行。设寄存器原内容为：

a＝$00111111000000

b＝$00800001000000

x1＝$000000　　r0＝$0000

键入：change a $00111111000000 b $00800001000000<return>

change x1 0 r0 0<return>

display a a2 a1 a0 b b2 b1 b0 x1 r0<return>

asm p: $303 add b, a b, x1<return>

asm p: $304 add x1 a b, r0<return>

change pc $303<return>

step 2<return>

display a a2 a1 a0 b b2 b1 b0 x1 r0<return>

结果：a＝$00911112000000

a2＝$01　　a1＝$111111　　a0＝$000000

b＝$00800001000000

b2＝$00　　b1＝$800001　　b0＝$000000

x1＝$7fffff

r0＝$ffff

注意限幅在 B 的输出上完成。目的寄存器 X1 和 R0 的限制值分别是 $7fffff 和 $ffff。

4.3.3 地址寄存器更新

按寻址方式并行传送操作更新地址寄存器 Rn。

例 4.8：指令 ADD B，A 和地址寄存器更新并行传送操作（R1）＋N1 同时执行，这里用后递增偏置 Nn 的间接寻址方式更新 R1。设寄存器原内容为：

a＝$00111111000000　　b＝$00222222000000

r1＝$1000　　n1＝$0006

键入：change a $00111111000000 b $00222222000000<return>

```
change r1 $1000 n1 $0006<return>
display a b r1 n1<return>
asm p: $305 add b, a (r1) +n1<return>
change pc $305<return>
step<return>
display a b r1 n1<return>
```

结果：a=$00333333000000　　b=$00222222000000

r1=$1006　　n1=$0006

4.3.4 X或Y存贮器数据传送

并行传送操作转移去/来自X或Y存贮器的一字操作数。

例4.9：寄存器到X存贮器。指令ADD A，B和X存贮器数据传送操作A，X：$1000并行执行，这里用绝对地址寻址方式指定X存贮器地址$1000。设寄存器原内容为：

a=$00123456000000　　b=$00111111000000

x：$1000 $000000

键入：
```
change a $00123456000000 b $00111111000000<return>
change x: $1000 0<return>
display a a2 a1 a0 b b2 b1 b0 x: $1000<return>
asm p: $306 ADD A, B A, x: $1000<return>
change pc $306<return>
step<return>
display a a2 a1 a0 b b2 b1 b0 x: $1000<return>
```

结果：a=$00123456000000

a2=$00　　a1=$123456　　a0=$000000

b=$00234567000000

b2=$00　　b1=$234567　　b0=$000000

x：$1000　　$123456

例4.10：Y存贮器到寄存器。指令ADD A，B和Y存贮器数据传送操作Y：－(R3)，A并行执行，这里采用先递减1的寻址方式指定Y存贮器地址。设各寄存器原内容为：

a=$00123456000000

b=$00111111000000

y：$1000 $999999

r3=$1001

键入：
```
change a $00123456000000 b $00111111000000<return>
change r3 $1001 y: $1000 $999999 <return>
display a a2 a1 a0 b b2 b1 b0 y: $1000 r3 <return>
asm p: $307 ADD A, B y: - (r3), a <return>
change pc $307<return>
step<return>
display a a2 a1 a0 b b2 b1 b0 y: $1000 r3<return>
```

结果：a=$ff999999000000

a2=$ff　　a1=$999999　　a0=$000000

b= $00234567000000

b2= $00　　b1= $234567　　b0= $000000

y: $1000 $999999

r3= $1000

4.3.5 X或Y存贮器和寄存器数据传送

并行传送操作转移一字操作数去/来自X或Y存贮器和从寄存器到寄存器。

例4.11：指令ADD X，A与X存贮器数据传送操作A，X:(R3+N3)和寄存器数据传送操作A，Y1并行执行。然后执行带符号乘指令MPY Y1，X1，A。设寄存器原内容为：

x= $111111111111

a= $00123456000000

y1= $000000

r3= $1000　　n3= $0006

x: $1006　　$000000

键入：change x $111111111111 a $00123456000000<return>

change r3 $1000 n3 $6 x: $1006 0 y1 0<return>

display x a y1 r3 n3 x: $1006<return>

asm p: $308 add x: a a, x: (r3+n3) a, yi<return>

asm p: $309 mpy y1, x1, a<return>

change pc $308 <return>

step 2<return>

display x a y1 r3 n3 x: $1006<return>

结果：x= $111111111111

a= $00026D60CA5F6C

y1= $123456

r3= $1000　　n3= $0006

x: $1006　　$123456

4.3.6 长存贮器数据传送

并行传送操作转移一长字操作数去/来自L(X: Y)存贮器。二个数据ALU寄存器并置以形长字操作数。这允许传送双精度(高：低)数据值或复(实：虚)数据值去/来自L存贮器位置。

例4.12：指令ADD Y，B和长存贮器传送操作B10，L: $1000并行执行，这里采用绝对地址寻址方式去指定L存贮器地址$1000。设有关寄存器原内容为：

y= $111111222222

b= $001111111111111

l: $1000　　$000000000000

x: $1000　　$000000

y: $1000　　$000000

键入：change y $111111222222　　b $001111111111111<return>

```
change l:  $1000  $000000000000<return>
display y b l: $1000 x: $1000 y: $1000<return>
asm p: $30A add y.b b10,l: $1000<return>
change pc  $30A<return>
step<return>
display y   b   l: $1000   x: $1000   y: $1000<return>
```

结果：y= $111111222222

b= $002222222333333

l: $1000 $111111111111

x: $1000 $111111

y: $1000 $111111

4.3.7 XY 存贮器数据传送

并行传送操作转移二个单字操作数去/来自 X 和 Y 存贮器。

例 4.13：指令 ADDX，B 和 XY 存贮器传送操作 X:(R0)+,X1 和 Y0,Y:(RT)− 同时执行，这里分别用后递增 1 和后递减 1 的寻址方式指定 X 和 Y 存贮器单元。设各寄存器原内容为：

x= $111111222222　　y0= $333333

b= $00111111000000

r0= $1000　　r7= $1500

x: $1000 $444444　　y: $1500 $000000

键入：
```
change x  $111111222222     b  $00111111000000<return>
change r0  $1000 r7  $1500 y0  $333333 <return>
change x:  $1000  $444444 y:  $1500 0 <return>
display x y0 b r0 r7 x: $1000 y: $1500 <return>
asm p: $30B add x,b x:(r0)+,x1 y0,y:(r7)−<return>
change pc  $30B <return>
step <return>
display x y0 b r0 r7 x: $1000 y: $1500 <return>
```

结果：x= $444444222222　　y0= $333333

b= $00222222222222

r0= $1001　　r7= $14ff

x: $1000 $444444　　y: $1500 $333333

4.3.8 并行传送总结

表 4-2 总结了并行传送操作。

可以与并行传送操作一起用的寻址方式总结在表 4-3 中，其中

U=地址寄存器更新

X=X 存贮器传送

Y=Y 存贮器传送

X+R=X 存贮器和寄存器传送

Y+R=Y 存贮器和寄存器传送

L=L 存贮器传送

XY=XY 存贮器传送

表 4-2 并行传送操作总结

并行传送操作	操作码操作数	举　例
立即短数据	ADD B，A	#$81，B
	ADD B，A	#$81，B
寄存器到寄器	ADD B，A	X1，B
	ADD B，A	B，X1
	ADD X1，A	B，R0
地址寄存器更新	ADD B，A	(R1) +N1
X 或 Y 存贮器	ADD A，B	A，X：$1000
	ADD A，B	Y：(−R3)，A
X 或 Y 存贮器加寄存器	ADD X，A	A，X：(R3+N3) A，Y1
L 存贮器	ADD Y，B	B10，L：$1000
XY 存贮器	ADD X,B	X:(R0)+,X1 Y0,Y:(R7)−

表 4-3 对 U，X，Y，X+R，Y+R，L 和 XY 并行传送的寻址方式

寻　址　方　式	并　行　传　送				
	U	X Y	X+R Y+R	L	XY
寄存器直接	否	否	否	否	否
地址寄存器间接					
不更新	否	是	是	是	是
后递增 1	是	是	是	是	是
后递减 1	是	是	是	是	是
后递增偏置 Nn	是	是	是	是	是
后递减偏置 Nn	是	是	是	是	否
由偏置 Nn 定位	否	是	是	是	否
先递减 1	否	是	是	是	否
专用					
立即数据	否	是	是	否	否
绝对地址	否	是	是	是	否
立即短数据	否	否	否	否	否
短跳地址	否	否	否	否	否
绝对短地址	否	是	否	是	否
I/O 短地址	否	否	否	否	否
隐含	否	否	否	否	否

4.4 DSP56001 指令集

DSP56001 指令集分为以下几组：

(1) 传送
(2) 算术
(3) 逻辑
(4) 位变换
(5) 循环
(6) 程序控制

4.4.1 传送指令

传送指令通过 XD 和 YD 数据总线、全局数据总线和程序数据总线传送数据。传送指令可看作是具有完成并行传送操作能力的数据 ALU NOP（无操作）。传送指令仅影响条件码寄存器 CCR 中的极限位，这里极限位是第 6 位。CCR 占据状态寄存器（SR）的低 8 位。传送指令是：

LUA	装人更新的地址
MOVE	传送数据
MOVEC or Move	传送控制寄存器
MOVEM or Move	传送程序存贮器
MOVEP or Move	传送外围数据

例 4.14：装人更新的地址（LUA）。

问题：执行指令 LUA (R0) +N0，N1，更新地址寄存器 R0 并把更新的地址存入偏置寄存器 N1 中。设在执行指令前，地址寄存器 R0 和偏置寄存器 N0 和 N1 的内容是：

r0= $1000　　n0= $0006　　n1= $0000

键人：
```
change r0 $1000 n0 $6 n1 0 <return>
display r0 n0 n1 <return>
asm p: $400 lua (r0) +n0, n1<return>
change pc $400 <return>
step <return>
display r0 n0 n1 <return>
```

结果：r0= $1000　　n0= $0006　　n1= $1006

例 4.15：传送控制寄存器（MOVEC）。

问题：执行指令 MOVEC A，LC 传送 A 累加器到循环控制（LC）寄存器。设 A 累加器、循环控制寄存器和状态寄存器（SR）的原内容是：

a= $.00800001000000　　lc= $0000　　sr= $0000

键人：
```
change a $00800001000000 lc $0 sr $0<return>
display a lc sr<return>
asm p: $401 movec a, lc<return>
change pc $401<return>
```

step<return>

display a lc sr<return>

结果：a＝$00800001000000　　lc＝$ffff　　sr＝$0040

注意，在A累加器的输出上完成限制。传送到循环控制寄存器的极限值是$FFFF。设置状态寄存器中的限制位，这里限制位是位6。

例4.16：传送程序存贮器（MOVEM）。

问题：执行指令MOVEM R3，P：(R2)－N2和MOVEM P：$0000，LC传送地址寄存器R3的内容到地址寄存器R2所指示的P存贮器单元；和传送P存贮器单元$0000的内容到循环计数器（LC）寄存器。设寄存器中原内容是：

r3＝$1000　　r2＝$0500　　n2＝$0002

p：$0000 $123456

p：$0500 $000000

lc＝$0000

键入：change r3 $1000 r2 $500 n2 $2<return>

change p：0500　0 p：$0 $123456 lc 0<return>

display r3 r2 n2 p：$0500 p：$0 lc<return>

asm p：$402 movem r3，p：(r2) －n2<return>

asm p：$403 movem p：$0000，lc<return>

change pc $402 <return>

step 2<return>

display r3 r2 n2 p：$500　p：$0 lc<return>

结果：r3＝$1000　　r2＝$04fe　　n2＝$0002

p：$0000 $123456

p：$0500 $001000

lc＝$3456

例4.17：传送外围数据（MOVEP）。

问题：执行指令MOVEP X：$FFFF，A，传送X存贮器I/O外围单元$FFFF的内容到A累加器。设各寄存器原内容为：

a＝$00000000000000

x：$ffff $001234

键入：change x：$ffff $001234 a 0<return>

display x：$ffff a a2 a1 a0<return>

asm p：$404 Movep x：$ffff，a<return>

change pc $404<return>

step<return>

display x：$ffff a a2 a1 a0 <return>

结果：a＝$00001234000000

a2＝$00　　a1＝$001234　　a0＝$000000

x：$ffff $001234

4.4.2 算术指令

算术指令用数据ALU去完成所有算术型操作。对算术指令，源操作数包含在数据ALU

的输入寄存器或累加器中；执行算术指令所产生的结果的目的地是 A 或 B 累加器。算术指令允许二个要执行的并行传送操作与指令操作码一起执行。并行传送操作用 XD 和 YD 数据总线。并行传送操作可传送从 X 或 Y 存贮器来的预取数据到数据 ALU，为后面的指令使用；并可传送从数据 ALU 来的执行前面指令产生的结果到 X 或 Y 存贮器。

算术指令有：

ABS	绝对值	MPY	乘
ADC	进位加	MPYR	乘和舍入
ADD	加	NEG	负
ADDL	左移加	NORM	归一化迭代
ADDR	右移加	RND	舍入
ASL	算术左移	SBC	进位减
ASR	算术右移	SUB	减
CLR	清除	SUBL	左移减
CMP	比较	SUBR	右移减
CMPM	比较大小	TCC	条件传送
DIV	除迭代	TFR	传送
MAC	乘/累加	TST	测试
MACR	乘/累加和舍入		

在以下算术指令举例中，采用缺省原标方式，即定标方式位 S1 和 S0 被清除，这里 S1 和 S0 分别是方式寄存器（MR）中第 4 和第 3 位，其中 MR 占据状态寄存器（SR）的高 8 位。为了清除 S0 和 S1，.DSP56000ADS 用户接口软件程序的 CHANGE 命令可用来修正 MR 寄存器的内容（例 4.18）。

1. 加和减算术指令

加和减算术指令综合于表 4－4 中。

例 4.18：左移减。执行指令 SUBL B，A ，A 累加器左移一位并以 0 移入最低有效位，再减去 B 累加器，结果（2A－B）存于 A。设原内容为：

a＝ $ 00222222222222

b＝ $ 00111111111111

键入：change sr $ 0300<return>

```
change b $ 00111111111111 a $ 00222222222222<return>
display a a2 a1 a0 b b2 b1 b0<return>
asm p: $ 500 subl b, a<return>
change pc $ 500 <return>
step <return>
display a a2 a1 a0 b b2 b1 b0<return>
```

结果：a＝ $ 00333333333333

a2＝ $ 00　　a1＝ $ 333333　　a0＝ $ 333333

b＝ $ 00111111111111

b2＝ $ 00　　b1＝ $ 111111　　b0＝ $ 111111

表 4-4 加和减算术指令

助记符	操　作	汇编符号	源 S	目的 D
ADC	S+D+C*→D	ADC S, D (PM)**	X, Y	A, B
ADD	S+D→D	ADD S, D (PM)	A	B
			B	A
			X, X1, X0	A, B
			Y, Y1, Y0	A, B
ADDL	S+2D→D	ADDL S, D (PM)	A	B
			B	A
ADDR	S+D/2→D	ADDR S, D (PM)	A	B
			B	A
SBC	D−S−C→D	SBC S, D (PM)	X, Y	A, B
SUB	D−S→D	SUB S, D (PM)	A	B
			B	A
			X, X1, X0	A, B
			Y, Y1, Y0	A, B
SUBL	2D−S→D	SUBL S, D (PM)	A	B
			B	A
SUBR	D/2−S→D	SUBR S, D (PM)	A	B
			B	A

* C=进位位。

* * PM 并行方式。

例 4.19：带进位加长。执行指令 ADC Y, A，把 48 位 Y 数据输入寄存器加进位位 C 加到 A，并将结果存于 A。进位位 C 是状态寄存器中最低有效位。设寄存器原内容为：

y= $ 222222000000

a= $ 00555555000000

sr= $ 0001

键入：change y $ 222222000000 a $ 00555555000000<return>

change sr $ 0001<return>

display y a sr<return>

asm p: $ 501 adc y, a<return>

change pc $ 501 <return>

step<return>

display y a sr<return>

结果：y= $ 222222000000

a= $ 00777777000001

sr= $ 0000

例 4.20：右移加，执行指令 ADDR B, A，A 累加器算术右移一位以最高有效位保持常数，把 B 加到已移的 A，结果（A/2+B）存于 A。设寄存器原内容为：

a= $ 00555555555555

b= $ 00222222222222

键入：change a $ 00555555555555 b $ 00222222222222<return>

display a a2 a1 a0 b b2 b1 b0<return>

asm p: $ 502 addr b, a<return>

change pc $ 502<return>

step<return>

display a a2 a1 a0 b b2 b1 b0<return>

结果：a= $ 004ccccccccccc

a2= $ 00　　a1= $ 4ccccc

a0= $ cccccc

b= $ 00222222222222

b2= $ 00　　b1= $ 222222　　b0= $ 222222

2. 乘法算术指令（见表 4-5，表中 D 是 A 或 B）

例 4.21：有和没有舍入的符号乘。执行指令 MPY X1，Y1，A 和 MPYR X1，Y1，B。分别按有和没有舍入乘 X1、Y1 两个带符号操作数，结果分别存于 A 和 B 中。设寄存器原内容为：

X1= $ 000007　　Y1= $ 400000

a= $ 000000000000

b= $ 000000000000

键入：change x1 $ 000007 y1 $ 400000 a 0 b 0<return>

display x1 y1 a a2 a1 a0 b b2 b1 b0<return>

asm p: $ 503 mpy x1, y1, a<return>

asm p: $ 504 mpyr x1, y1, b<return>

change pc $ 503<return>

step 2<return>

display x1 y1 a a2 a1 a0 b b2 b1 b0<return>

结果：x1= $ 000007 y1= $ 400000

a= $ 000000003800000

b= $ 000000004000000

注意 B 中的乘积是 A 中乘积已舍入的数。

例 4.22：有和没有舍入的带符号乘-累加器。执行指令 MAC－X1，Y1，A 和 MACR－X1，Y1，B。设有关寄存器原内容为：

x1= $ 400000　　y1= $ 000007

a= $ 000000005100000

b= $ 000000005100000

键入：change x1 $ 400000 y1 $ 000007<return>

change a $ 000000005100000 b $ 000000005100000<return>

display x1, y1 a b<return>

asm p: $ 505 mac－x1, y1, a<return>

asm p: $ 506 macr－x1, y1, b<return>

```
change pc $505<return>
step 2<return>
display x1 y1 a b <return>
```

结果：x1＝$400000　　y1＝$000007

a＝$00000001900000

b＝$00000002000000

3. 其他算术指令（见表 4－6）

表 4－5　乘法算术指令

助记符	操　作	汇编符号	源	
			S1	S2
MAC	D±(S1・ms2)→D	MAC±S1,S2,D(PM)**	X0	X0
			Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1
MACR	D±(S1・S2)+r*→D	MACR±S1,S2,D(PM)	X0	X0
			Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1
MPY	±(S1・S2)→D	MPY±S1,S2,D(PM)	X0	X0
			Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1
MPYR	±(S1・S2)+r→D	MPYR±S1,S2,D(PM)	X0	X0
			Y0	Y0
			X1	X0
			Y1	Y0
			X0	Y1
			Y0	X0
			X1	Y0
			Y1	X1

*r＝舍入。

**PM＝并行方式。

表 4-6 其他算术指令

助记符	操　作	汇编符号	源 S	目的 D
TFR	S→D	TFR S,D[PM]	A	B
			B	A
			X1,X0	A,B
			Y1,Y0	A,B
NEG	0−D→D	NEG D [PM]		A,B
ABS	|D|→D	D[PM]		A,B
NORM*	if $\overline{E}.U.\overline{Z}=1$ then	NORM Rn,D		A,B
	ASL→D			
	Rn−1→Rn			
	Else if E=1 then			
	ASR→D			
	Rn+1→Rn			
	Else			
	Nop			
CLR	0→D	CLR D[PM]		A,B

*Rn=地址寄存器 R0～R7

E=扩展位(状态寄存器的第 5 位)

U=未归一化位(状态寄存器的第 4 位)

Z=零位(状态寄存器的第 2 位)

例 4.23:传送数据 ALU 寄存器。执行指令 TFR A,B,从 A 累加器传送数据到 B 累加器。设 A、B 原内容为:

a= $00111111222222

b= $00000000000000

键入:chamge a $00111111222222 b 0<return>

display a b<return>

asm p: $507 tfr a,b<return>

change pc $507<return>

step<return>

display a b<return>

结果:a= $00111111222222

b= $00111111222222

例 4.24:负累加器。执行指令 NEGA,从 0 减去 A,并将结果存于 A。设 A 的原内容为:

a= $00333333333333

键入:change a $00333333333333<return>

display a<return>

asm p: $508 neg a<return>

change pc $508<return>

step<return>

display a<return>

结果：a＝ $ ffcccccccccccd

例 4.25：绝对值。执行指令 ABS A，取 A 中的绝对值并存于 A 中。设 A 中原内容为：

a＝ $ ff900000000000

键入：change a $ ff900000000000<return>

display a<return>

asm p: $ 509 abs a<return>

change pc $ 509<return>

step<return>

display a<return>

结果：a＝ $ 007000000000000

注意 10 进制数 － 0.875（$ ff900000000000）的绝对值是 10 进制数 0.875（$ 007000000000000）。

例 4.26：归一化累加器迭代。执行指令 NORM R0,B，完成 B 累加器的一次归一化迭代。

(1)累加器扩展位被置位。设寄存器原内容为：

b＝ $ 22888888000000

r0＝ $ fffa　　sr＝ $ 0020

若 SR 的扩展位(位 5)在指令执行前被置定，则 B 算术右移一位，且 R0 递增 1。

键入：change b $ 22888888000000 r0 $ fffa sr $ 0020<return>

display b b2 b1 b0 r0 sr<return>

asm p: $ 50a norm r0,b<return>

change pc $ 50a<return>

step<return>

display b b2 b1 b0 r0 sr<return>

结果：b＝ $ 11444444000000

r0＝ $ fffb　　sr＝ $ 0020

(2)不用累加器扩展。设

b＝ $ 00111111000000

r0＝ $ ffff　　sr＝ $ 0010

若 SR 的扩展位被清除，非归一化位(位 4)被置定，则 B 算术左移一位，且地址寄存器 R0 递减 1。

键入：change r0 $ ffff b $ 00111111000000 sr $ 0010<return>

display b r0 sr<return>

asm p: $ 50b norm r0,b<return>

change pc $ 50b<return>

step<return>

display b r0 sr<return>

结果：b＝ $ 00222222000000　　r0＝ $ fffe　　sr＝ $ 0010

例 4.27：清除累加器。执行指令 CLR A。设

a＝ $ 112222223333333

执行程序后，得 a＝ $ 000000000000000

4.4.3 逻辑指令

逻辑指令用数据 ALU 完成全部逻辑型操作。逻辑指令的源操作数，除 ANDI 和 ORI 以外，都包含在数据 ALU 的输入寄存器或累加器中；指令执行的结果，除 ANDI 和 ORI 外，也在 A 或 B 累加器中。ANDI 或 ORI 指令执行结果的目的地是方式寄存器 MR、条件码寄存器 CCR 或操作方式寄存器 OMR。除 ANDI 和 ORI，逻辑指令可与指令操作码一起执行双并行传送操作。并行传送操作用 XD 和 YD 数据总线。

逻辑指令在一个指令周期中执行，并可影响条件码寄存器中的所有位。逻辑指令有：

AND	逻辑与	NOT	补
ANDI	AND 立即控制寄存器	OR	逻辑或
EOR	逻辑“异”	ORI	OR 立即控制寄存器
LSL	逻辑左移	ROL	旋转左移
LDR	逻辑右移	ROR	旋转右移

1. 逻辑指令

4 种逻辑指令列于表 4－7。这些指令是 24 位操作，是在 A 或 B 累加器的 24～47 位上完成的。

表 4－7 逻辑指令

助记符	操　作	汇编符号	源 S	目的 D
AND	S.D→D	AND S,D(PM)	X1,X0	A,B
			Y1,Y0	A,B
OR	S+D→D	OR S,D(PM)	X1,X0	A,B
			Y1,Y0	A,B
EOR	S+D→D	EOR S,D(PM)	X1,X0	A,B
			Y1,Y0	A,B
NOT	$\overline{D}$→D	NOT D(PM)		A,B

2. 移位和旋转逻辑指令

这些指令列于表 4－8。

表 4－8 移位和旋转逻辑指令

助记符	操　作	汇编符号	目的 D
LDL	逻辑左移	LSL D(PM)	A,B
ROL	旋转左移	ROL D(PM)	A,B
LSR	逻辑右移	LSR D(PM)	A,B
ROR	旋转右移	ROR D(PM)	A,B

例 4.28：逻辑右移。执行指令 LSR A，逻辑右移一位 A 累加器的 24～47 位，并存入 A。不

影响其他位。SR 的进位位(位 0)接收移出 A 的最低有效位(位 24)。寄存器原内容：

a＝ $ 00222222444444

sr＝ $ 0001

键入：change a $ 00222222444444<return>

change sr $ 0001<return>

display a sr<return>

asm p: $ 600 lsr a<return>

change pc $ 600<return>

step<return>

display a dr<return>

结果：a＝ $ 00111111444444

sr＝ $ 0000

例 4.29：累加器旋转左移，执行指令 ROL A。向左旋转一位 A 的 24～47 位并存于 A。SR 的进位位接收移出 A 的 47 位。进位位前面的值移入 A 的 24 位。寄存器原内容：

a＝ $ 00222222000000

sr＝ $ 0001

键入：change a $ 00222222000000<return>

display a sr<return>

asm p: $ 601 rol a<return>

change pc $ 601<return>

step<return>

display a sr<return>

结果：a＝ $ 00444445000000

sr＝ $ 0000

3. 立即逻辑指令

这些指令列于表 4－9。

表 4－9 立即逻辑指令

助记符	操　作	汇编符号	目的 D
ANDI	#××.D→D	AND(I)#××.D	MR,CCR,OMR
ORI	#××+D→D	OR(I)#××,D	MR,CCR,OMR

例，4.30：AND 立即控制寄存器。执行指令 ANDI# $ f3,MR。MR 的内容和 8 位立即操作数 $ f3 相与并存于 MR。设：

sr＝ $ af7f

键入：change sr $ af7f<return>

display sr<return>

asm p: $ 602 andi # $ f3,mr<return>

change pc $ 602<return>

step<return>

display sr<return>

结果：sr＝ $ a37f

4.4.4 位变换指令

有二个基本的位变换指令组。

1. 第一组

BCLR　　位测试与清除

BSET　　位测试与置定

BCHG　　位测试与变换

BTST　　存贮器位测试

条件码寄存器的进位码将反映位测试结果。第一组位变换指令列于表 4－10。

表 4－10　第一组位变换指令

助记符	操作	汇编符号
BCLR*	D(n)→C	BCLR＃n,x:<ea>
	0→D(n)	BCLR＃n,y:<ea>
BST*	D(n)→C	BSET＃n,x:<ea>
	1→D(n)	BSET＃n,y:<ea>
BCHG*	D(n)→C	BCHG＃n,x:<ea>
	$\overline{D(n)}$→D(n)	BCHG＃n,y:<ea>
BTST*	D(n)→C	BTST＃n,x:<ea>
		BTST＃n,y:<ea>

*D(n)＝24 位目的操作数数第 n 位。

0≤n≤23。

例 4.31：存贮器位测试。执行指令 BTST＃2,X:＄200。测试位于＄200 的 X 存贮器第位。设：

x:＄200　＄000004

sr＝＄0000

键入：change x:＄200　＄000004 sr　＄0000<return>

display x:＄200 sr<return>

asm p:＄603 btst ＃2,x:＄200<return>

change pc　＄603<return>

step<return>

display x:＄200 sr<return>

结果：x:＄200　＄000004

sr＝＄0001

2. 第二组

JCLR　　若位清除则跳

JSET　　若位置定则跳

JSCLR　　若位清除则跳到子程序

JSSET　　若位置定则跳到子程序

第二组位变换指令列于表 4－11。

表 4-11　第二组位变换指令

助记符	操　作	汇编符号
JSET	If S(n)* =1	JSET #n,x:<ea>,×××× JSET#n,Y:<ea>××××
	Then××××→pc	
	Else pc+1→pc	
JCLR	If S(n)* =0	JCLR #n,X:<ea>×××× JCLR#n,Y:<ea>,××××
	Then××××→pc	
	Else pc+1→pc	
JSSET	If S(n)* =1	JSSET #n,X:<ea>,×××× JSSET *n,Y<ea>,××××
	Then sp+1→sp	
	pc→SSH	
	SR→SSL	
	××××→pc	
	Else pc+1→pc.	
JSCLR	If S(n)* =0	JSCLR #n,X:<ea>×××× JSCLR #n,Y:<ea>,××××
	Then sp+1→sp	
	pc→SSH	
	SR→SSL	
	JSSET #n,Y:<ea>,××××→pc	
	Else pc+1→pc	

*S(n)=源操作数中第 n 位。

例 4.32：位置定则跳。执行指令 JSET#2，X:(R0)，$1000，测试 (R0) 指示的 X 存贮器的第 2 位。若位被置定，取位于 $1000 P 存贮器的指令。否则程序计数器递增 1。设：

x: $200　　$000004

pc= $604　　r0= $0200

键入：change X: $200 $000004 r0 $200<return>

asm p: $604 jset #2, X: (r0), $1000<return>

change pc $604<return>

display x: $200 pc r0<return>

step<return>

display x: $200 pc r0<return>

结果：x: $200　　$000004

pc= $1001　　r0= $0200

若 $200 X 存贮器的位 2 被“清除”，则 PC 将递增 1 从 $604 到 $605。

4.4.5　循环指令

循环指令有 DO 和 ENDDO。可中断 DO 指令引入硬件 DO 循环的开始。ENDDO 指令在循环完成前可终止硬件 DO 循环。循环计数器 (LC) 的内容确定 DO 循环重复的次数。循环地址 (LA) 寄存器的内容指出 DO 循环中最后指令字的单元。LC 和 LA 可以推入系统堆栈，因此允许 DO 循环中断或嵌套；它们可以用 RET 指令或 ENDDO 指令将其弹出堆栈。

例 4.33：执行指令 DOLOOP.LOD。带符号小数 A (i) 乘 B (i)，i=1、2 和 3。设有关

寄存器原内容为：

X：	$1000	$400000	$400000	$400000
X：	$1500	$000000	$000000	$000000
Y：	$2000	$300000	$300000	$300000

数据A（1）、A（2）、A（3）和B（1）、B（2）、B（3）分别存在$1000～$1002的X存贮器和$2000～$2002 Y存贮器中，相乘结果存入$1500～1502 X存贮器中。DOLOOP.ASM汇编程序如下：

```
move #$1000, r0
move #$2000, r4
move #$1500, r1
move                x:(r0), x0
move                                y:(r4), y0
do# 3, .end
mpyr x0, y0, a      x:(r0)+, x0     y:(r4)+, y0
move                a, x:(r1)+
.end
```

过程：

（1）引导PC。

（2）把DSP56000/1示范软件#2软盘插入驱动器“A”。

（3）键入：

```
display off<return>
change x: $1000…$1002 $400000<return>
change x: $1500…$1502 o<return>
change y: $2000…$2002 $300000<return>
display on x: $1000…$1002 y: $2000…$2002<return>
display on x: $1500…$1502<return>
load a: doloop<return>
step 6<return>
display la lc<return>
```

循环开始时：la=$000b

lc=$0002

循环结束时：la=$0000　　lc=$0000

x：	$1000	$400000	$400000	$400000
y：	$2000	$300000	$300000	$300000
x：	$1500	$180000	$180000	$180000

注意1/2（$400000）乘3/8（$300000）的结果是3/16（$180000）。DO指令以程序循环（$000B）中最后指令字的单元装入LA寄存器。

例4.34：执行程序DOLOOP1.LOD。带符号整数A（i）乘B（i），i=1，2，3。数据分别存在$1000～$1002 X存贮器和$2000～2002 Y存贮器。乘的结果存入$1500～$1502 L存贮器。设存贮器内容为：

x：	$1000～$1002	$000002
y：	$2000～$2002	$000138
l：	$1500～$1502	$000000000000

键入：change x：$1000..$1002 $000002<return>

change l：$1500..$1502 0<return>

change y：$2000..$2002 $000138<return>

display on x：$1000..$1002 y：$2000..$2002<return>

display on l：$1500..1502<return>

display<return>

load a：doloop 1<return>

break #1 pc>=$e

go #1<return>

结果：x、y 内容不变

l：$1500…$1502 $000000000270

4.4.6 程序控制指令

程序控制指令包括跳、条件跳和其他影响 PC 与系统堆栈的指令。这些指令不允许并行传送操作。它们有：

Jcc	条件跳	RESET	复位片上外围设备
JMP	跳	RTI	由中断返回
JScc	有条件地跳到子程序	RTS	由子程序返回
JSR	跳到子程序	STOP	停止指令处理
NOP	不操作	SWI	软件中断
REP	重复指令	WAIT	等待中断

例 4.35：重复下一指令。执行两条指令 REP#3 和 ASR B 。重复执行 ASR B 指令三次。重复下一指令（REP）列于表 4-12。设寄存器原内容为：

b= $00888888000000　　lc= $0300

表 4-12 REP 指令

助记符	操　作	汇编符号
REP	LC→TEMP；#×××→LC Repeat Next Instruction Until LC=1 TEMP→LC	REP#××× REP X：<ea> REP Y：<ea> REP S

键入：change b $00888888000000 lc $0300<return>

display b lc<return>

asm p： $700<return>

rep #3<return>

asr b<return>

<esc>

change pc $700<return>

step<return>

display b lc<return>

```
step<return>
display b lc<return>
step<return>
display b lc<return>
step<return>
display b lc<return>
```

结果：第一次 REP 后　　b＝$ 008888888000000　　lc＝$ 0002
　　第二次 REP 后　　b＝$ 004444444000000　　lc＝$ 0001
　　第三次 REP 后　　b＝$ 002222222000000　　lc＝$ 0300
　　程序完　　b＝$ 001111111000000　　lc＝$ 0300

例 4.36：执行程序 PR.LOD。测试位于 $ 100 X 存贮器的第 7 位。若 CCR 进位位被清除则跳。PR.LOD。的汇编程序为：

```
        org p:  $ 200
wait    btst #7, x:  $ 100
        jcc wait
        end
```

(1) 键入：
```
display off<return>
change x:  $ 100  $ 40<return>
load a:  pr<return>
disassemble<return>
```

```
p:  $ 0200    0B7027    000100    =BTST # $ 7, x:  > $ 100
p:  $ 0202    0E0200              =JCC< $ 200
p:  $ 0203    000000              =NOP
p:  $ 0204    000000              =NOP
```

```
display on sr<return>
step 2<return>
```

结果：sr＝$ 0310
　　p: $ 0200 0B7027 000100=BTST $ 7, X: > $ 100
注意：CCR 的进位位被清除，所以程序继续在 P 存贮器 $ 200 执行。

(2) 键入：
```
change x:  $ 100  $ 80<return>
change pc  $ 200<return>
step 2<return>
```

结果：sr＝$ 0311
　　p: $ 0203 000000=NOP
注意：CCR 的进位位被置定，所以程序在 P 存贮器 $ 203 继续执行。

4.4.7　DSP56000/1 指令集综合（见表 4-13）

表 4－13　指令集综合

指令类型	助记符	说　明	指令类型	助记符	说　明
传送指令	LUA	装入更新地址	逻辑指令	AND	逻辑与
	MOVE	传送数据		ANDI	AND 立即控制寄存器
	MOVEC	传送控制寄存器		EOR	逻辑异
	MOVEM	传送程序存贮器		LSL	逻辑左移
	MOVEP	传送外围数据		LSR	逻辑右移
				NOT	补
				OR	逻辑或
				ORI	OR 立即控制寄存器
				ROL	左转
				ROR	右转
算术指令	ABS	绝对值	位变换指令	BCLR	位测试和清除
	ADC	带进位加		BSET	位测试和置定
	ADD	加		BCHG	位测试和变换
	ADDL	左移加		BTST	存贮器上位测试
	ADDR	右移加		JCLR	位清除则跳
	ASL	算术左移		JSET	位置定则跳
	ASR	算术右移		JSCLR	位清除则跳到子程序
	CLR	清除		JSSET	位置定则跳到子程序
	CMP	比较	循环指令	DO	开始硬件循环
	CMPM	比较大小		ENDDO	从硬件循环退出
	DIV	除迭代	程序控制指令	JCC	条件跳
	MAC	乘/累加		JMP	跳
	MACR	乘/累加和舍入		JSCC	条件跳到子程序
	MPY	乘		JSR	跳到子程序
	MPYR	乘和舍入		NOP	不操作
	BEG	负		REP	重复指令
	NORM	归一化迭代		RESET	复位片上外围设备
	RND	舍入		RTI	从中断返回
	SBC	带进位减		RTS	从子程序返回
	SUB	减		STOP	停止指令处理
	SUBL	左移减		SWI	软件中断
	SUBR	右移减		WAIT	等待中断
	TCC	条件传送			
	TFR	传送			
	TST	测试			

第五章　数字信号处理系统概论

本章中，实现由模数（A/D）转换器、DSP 处理器和数模（D/A）转换器组成的简单 DSP 系统。DSP56000ADS 应用开发系统（ADS）用来提供和控制 DSP56001 处理器。DSP56ADC16EVB 评价板（EVB）用来提供和控制 A/D 和 D/A 转换器。DSP56001 处理器的同步串行接口（SSI）端口用来调节从 A/D 转换器到 DSP56001 处理器和由处理器到 D/A 转换器的串行数据传送。

首先，说明 DSP56ADC16EVB 评价板的设备，随后讨论 DSP56001 处理器指令。最后说明取样速率低于 Nyquist 速率时产生的“混叠”现象。

5.1　DSP 系统

在以上定义的 DSP 系统中，如图 5－1 和 5－2 所示，模拟输入信号在 EVB 上由 A/D 转

图 5－1　DSP 系统

换器进行数字化，数字化信号由 A/D 转换器输出，此信号是由帧同步输出（FSO）信号定界的 16 位串行数据流。经 DSP56001 处理器的 SSI 端口的接收信道，数字化信号由 ADS 的应用开发模块上的 DSP56001 处理器接收。DSP56001 处理器引导数字化信号从 SSI 端口的接收信道到 SSI 端口的发送信道。经过 DSP56001 处理器 SSI 端口的发送信道，数字化信号作为 16 位串行数据流从 DSP56001 处理器输出，并到 EVB 上的 D/A 转换器的串行输入。D/A 转换器形成模拟输出。恢复滤波器起平滑作用。

为了 D/A 转换器能恢复原模拟信号，Nyquist 采样定理必须首先满足。即采样速率应大于模拟信号最高频率分量的两倍。

实际上，没有实际的模拟信号能严格地带限在一定范围内，那些高于 Nyquist 采样率一半的频率分量会给数字化信号引进误差，使之不可能准确恢复原模拟信号。通常，模拟信号带限越好，那些高频分量的能量越低，模拟信号恢复越好。

另一个误差分量是由 D/A 转换器引入，因为在信号恢复过程中用了无限宽度的 sinc 函数。由于各种误差分量积累的影响，原模拟信号的恢复只能是近似的。若采样速率低于 Nyquist 速率，将产生“混叠”。

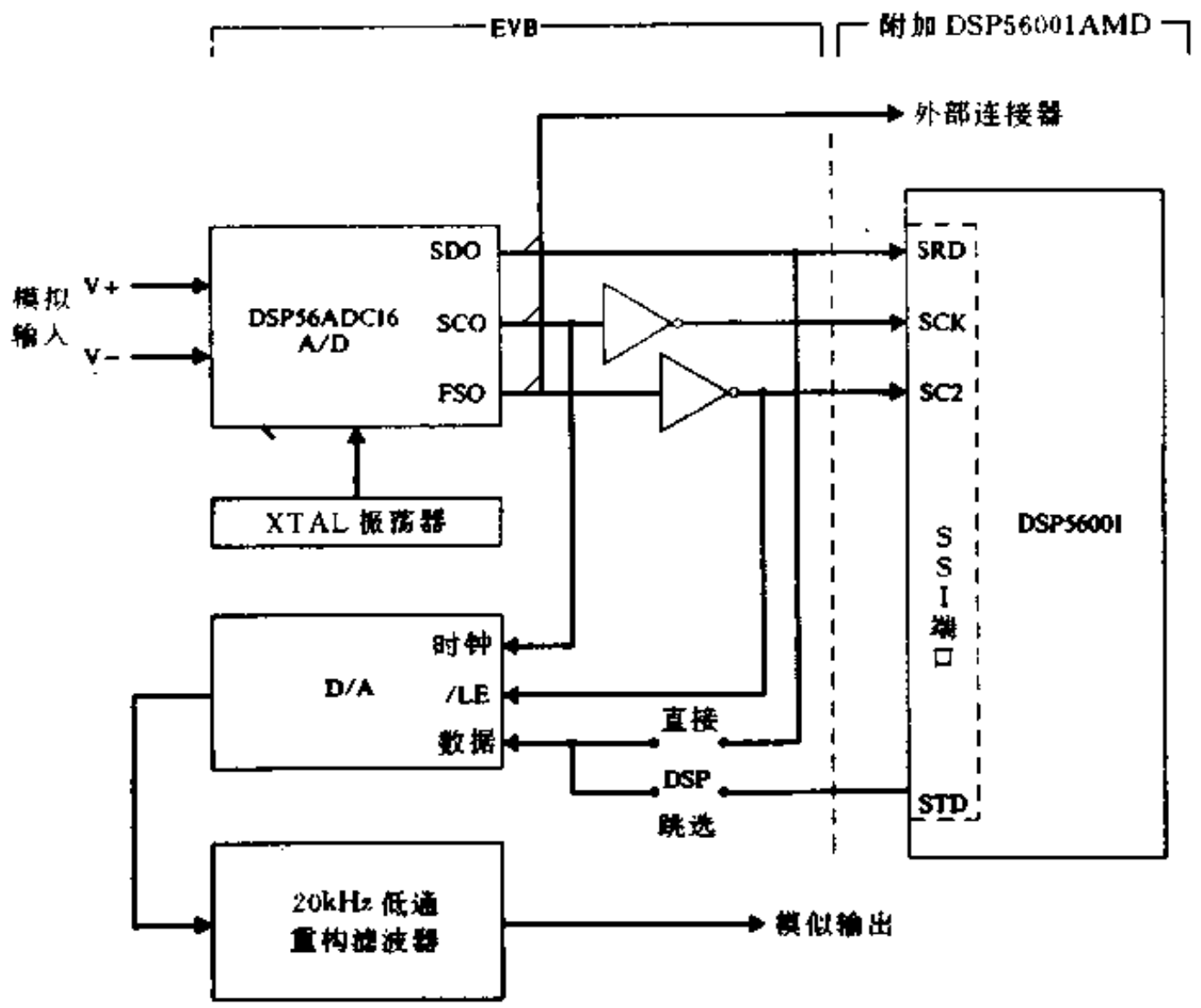


图 5-2　EVB 框图

5.2　DSP56ADC16 EVB 评价板（EVB）

5.2.1　EVB 一般说明

EVB 是 A/D 和 D/A 转换系统，可用于和 ADS 连接。EVB 的主要部件如图 5-2 所示，有：

1. 一个 DSP56ADC16 16 位、过取样、Σ-Δ A/D 转换器，它可在高达 100 kHz/16 位取样的群速率下串行输出 16 位数据样值。

2. 一个 16 位 D/A 转换器。

3. 截止频率为 20kHz 的低通恢复滤波器。

5.2.2　EVB 的建立

1. 如图 5-3，连接带状电缆一端到 EVB 的 J2；连接另一端到 DSP56001 应用开发模块（ADM）的 J5。若电缆不好插，则反 180°再试。

2. EVB 可从 ADM 拉电源。如图 5-3，连接电源线（pigtail）的黑线到地。连接红线到 DIG +5V。连接黑线另一端插头到 J3 地。连接红线另一端插头到 J3 +5V。

3. 在 EVB 上，装上跨线 JP2、JP4、JP6、JP8、JP9、JP11 和 JP13，并去掉 JP1、JP3、JP5、JP7、JP10、JP12 和 JP14。这些跨线使 EVB 作以下操作：

(1) 6.144MHz 时钟供给 DSP56ADC16 A/D 转换器时钟输入 (CLKIN) (A/D 转换器串行输出 16 位采样，以供给 A/D 时钟输入除以 128 的速率，也就是 6.144MHz 时钟输入频率使 A/D 转换器以 48kHz/16 位采样的群速率串行输出 16 位采样)。

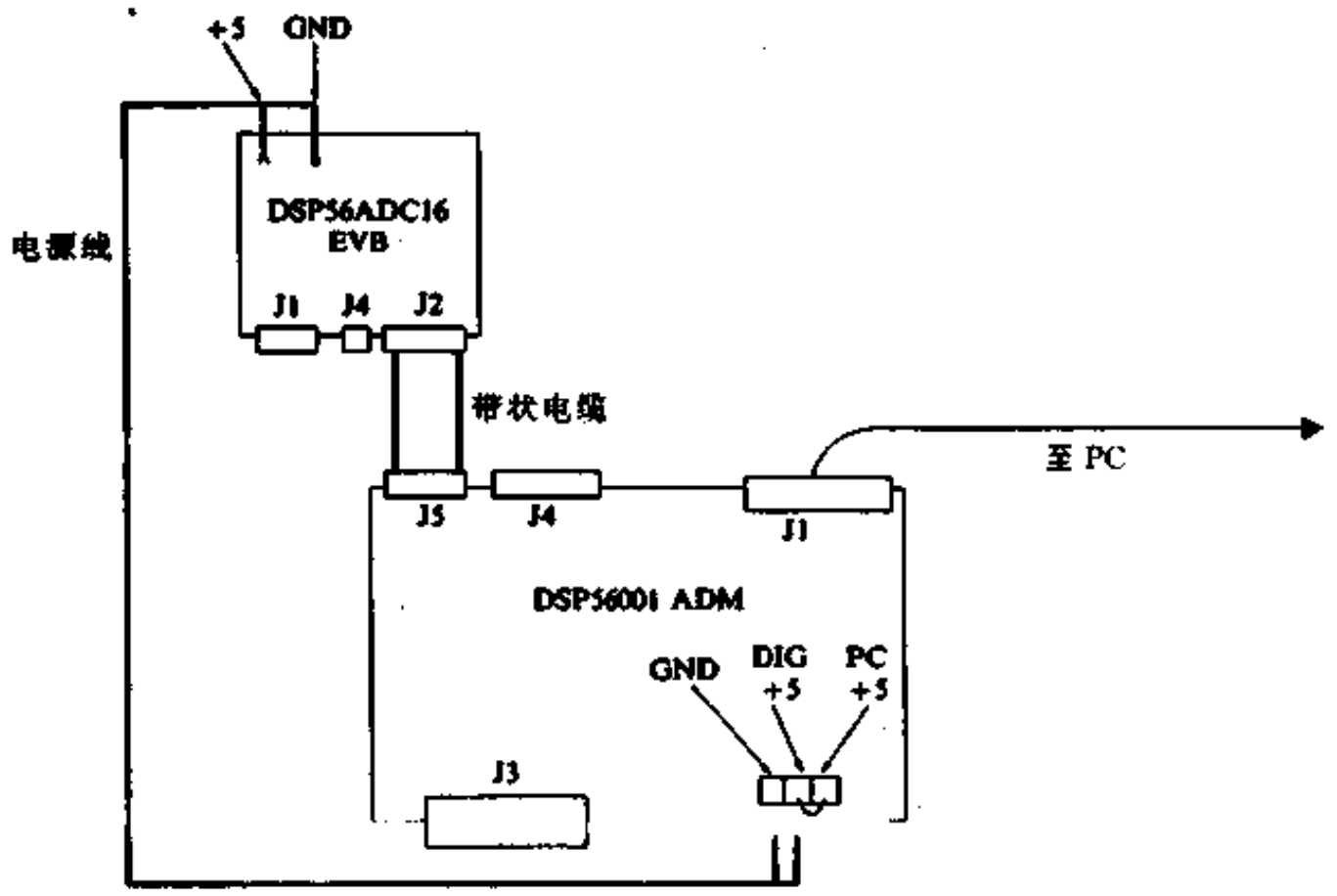


图 5-3 EVB 与 DSP56001 ADM 的连接

(2) 16 位 宽的帧同步输出 (FSO)，来自 DSP56ADC16 A/D 转换器。

(3) 单端模拟输入信号经 BNC2 加到 DSP56ADC16 A/D 转换器模拟+输入 (Vin+)。

(4) DSP56001 处理器 SSI 端口发送数据经过连接器/插脚 J2-1 到 D/A 转换器的数据输入端。

(5) 一低通恢复滤波器在 D/A 转换器的模拟输出处。

5.3 DSP56001 处理器 SSI 口引出端

SSI 端口有 6 条专用引出端，为图 5-4 所示。

(1) 串行发送数据 (STD)。

(2) 串行接收数据 (SRD)。

(3) 串行时钟 (SCK)。

(4) 串行控制 (SC0)。

(5) 串行控制 (SC1)。

(6) 串行控制 (SC2)。

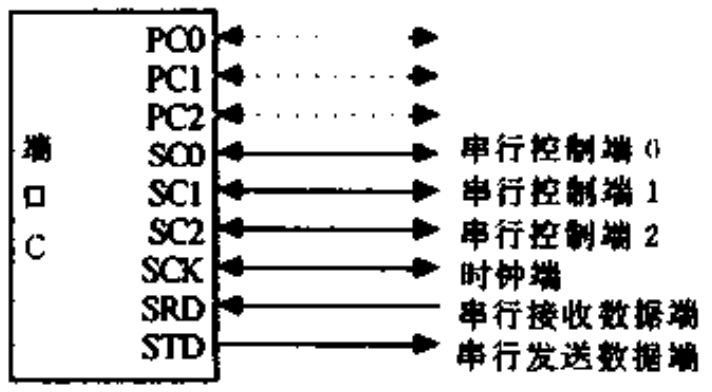


图 5-4 同步串行接口插脚

如图 5-2，SSI 端口的 STD、SRD、SCK 和 SC2、用来连接 ADM 上的 DSP56001 处理器与 EVB 上的 A/D 和 D/A 转换器。STD 端用来从 SSI 端口的串行发送移位寄存器输出数据。此数据送到 D/A 转换器的输入。SRD 端用来接收数据并送入 SSI 端口的接收数据移位寄存器。此数据来自 A/D 转换器。SCK 端用来接收外时钟。此外时钟的源是 A/D 转换器的串

行时钟输出端。外时钟也是 D/A 转换器的时钟。SC2 端用来接受外帧同步。此外帧同步的源是 A/D 转换器的帧同步输出端。外帧同步也要求用 D/A 转换器的使能锁定输入端。

5.4 设定 DSP56001 处理器 SSI 端口

如图 5－5 所示，DSP56001 处理器有：

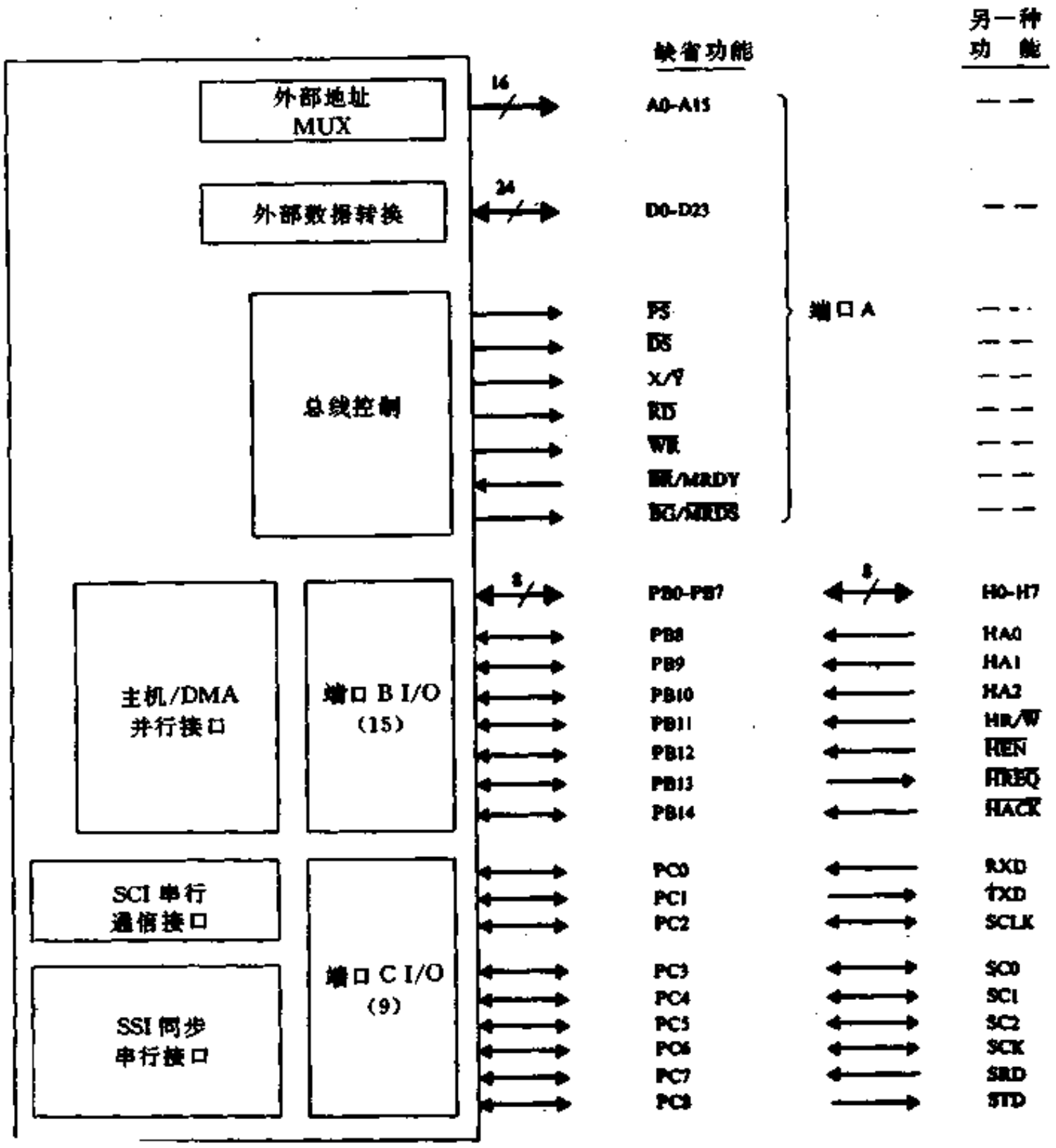


图 5－5 DSP56001 输入/输出图

1. 47 条引出端的存贮器扩展端口 A。

2. 15 条引出端的端口 B，可作为一般 I/O 端口或并行主机 MPU/DMA 接口的片上外围端口。

3. 9 条引出端的端口 C，其中每端可选 用作为一般 I/O 端，或片上外围功能端。片上外围包括串行通信接口（SCI）端口和同步串行接口（SSI）端口。

每个主机 SCI 和 SSI 的片上外围都有它们自己的控制、状态、数据等寄存器。DSP56001 处理器访问这些寄器如同它们是存贮器映射的 I/O。这些片上存贮器映射的寄存器在 X 存贮

器单元 $FFC0～FFFF 处被访问，如图 5－6。

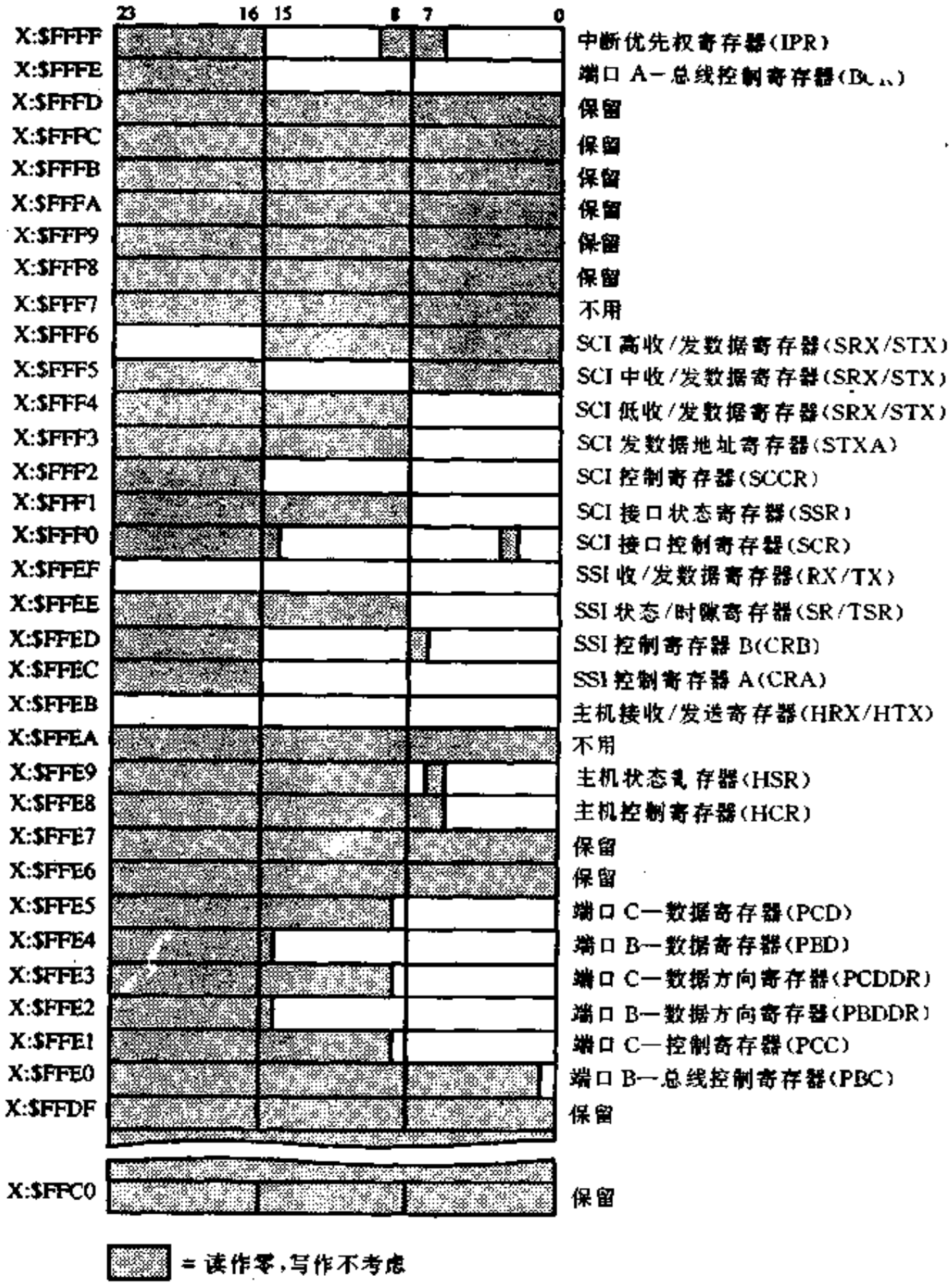


图 5－6　DSP56001 外围存贮器映像

5.4.1　SSI 端口选择

在 DSP 系统中，SSI 端口用来接收从 EVB 上 A/D 转换器来的串行数据，和发送串行数据到 EVB 上的 D/A 转换器。DSP56001 处理器端口 C 控制寄存器（PPC）有 9 位，它们一对一地选择端口 C 的 9 条引出端，作为一般的 I/O 端，或作为 SCI 或 SSI 外围功能端。通过清除相应 PCC 寄存器位，端口 C 引出端可构成一般的 I/O 端。通过置相应 PCC 寄存器位，端口 C 引出端可构成 SCI 或 SSI 功能端。可在 X 存贮器 $FFE1 处访问 PCC 寄存器。下面的

DSP56001 处理器指令选择端口 C 作为 SCI 和 SSI 端口：

```
pcc     equ     $0001ff
        movep   #pcc, x: $FFE1;     write   pcc
```

5.4.2 SSI 端口控制寄存器 A（CRA）

SSI 端口操作由两个 16 位 读/写控制寄存器 CRA 和 CRB 来决定。SSI 寄存器示于图 5-7。CRA 控制 SSI 端口的以下功能：

（1）内部位时钟发生器速率。

（2）内部帧分割速率。

（3）数据字长度。

数据字长度可以是 8、12、16 或 24 位。CRA 的字长控制位 13（WL0）和 14（WL1）按表 5-1 选择字长。

表 5-1 SSI 端口字长

WL1	WL0	位数/字
0	0	8
0	1	12
1	0	16
1	1	24

在 DSP 系统中，不用 SSI 端口的内部时钟和帧发生器。下述 DSP56001 指令置定 CRA 选择 16 位字长。CRA 占用 X 存贮器 $FFEC 单元。

```
cra     equ     $004000
        movep   #cra, x: $FFEC; CRA  pattetn  for  word
                                ; length=16  bits
```

5.4.3 SSI 端口控制寄存器 B（CRB）

SSI 端口的操作由二个 16 位读/写控制寄存器 CRA 和 CRB 决定。CRB 控制 SSI 端口的如下功能：

（1）多功能引出端 SC2，SC1T SC0。

（2）串行输出标志控制位。

（3）引出端 SCK、SC2、SC1 和 SC0 输入对输出的方向。

（4）操作方式；

（5）发送器和接收器使能。

（6）发送器和接收器中断使能，

在图 5-1 和图 5-2 所示的 DSP 系统中，不用SSI 端口的 SC1 和 SC0 引出端。CRB 占据 X 存贮器单元 $FFED。以下所示的 DSP56001 指令置定 CRB。其中：

（1）CRB 位 4 被清除以选择 SC2 端作为输入（外部帧同步）。

（2）CRB 位 5 被清除以选择 SCK 端的方向作为输入（对发送和接收移位寄存器为外部时钟）。

（3）CRB 位 6 被清除以选择发送移位寄存器首先发送 MSB，和接收移位寄存器首先接收 MSB。

（4）CRB 位 7 和位 8 被清除以选择字长（16 位）帧同步。

（5）CRB 位 9 被置定以选择发送器和接收器工作在同步方式，并共用时钟（SCR 端）和共用帧同步（SC2 端）。

（6）CRB 位 10 被清除以选择“连续”时钟工作方式。

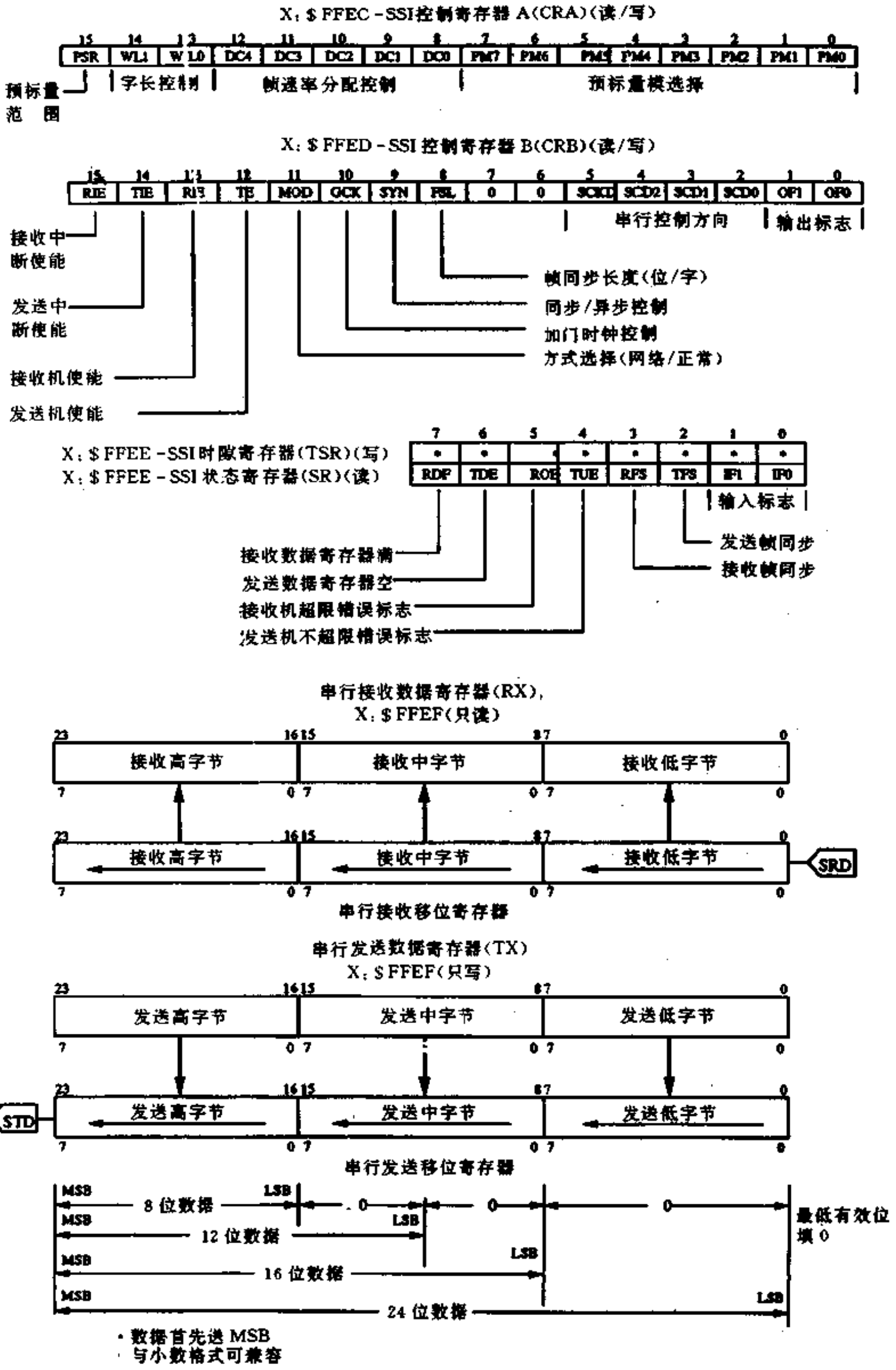


图 5－7　同步串行接口寄存器

(7) CRB 位 11 被清除以选择“正常”工作方式，即每帧发送或接收一个数据字（16 位）。

(8) CRB 位 12 和 13 都被置定，分别允许发送器和接收器工作。

(9) CRB 位 14 和 15 都被清除，分别使发送和接收不能中断。

(10) CRB 位 3～0 不属此特定的 SSI 端口，但它们也被清除。

```
crb   equ   $003200
      movep   #crb, x: $FFED   ; crb  patters  for
                               ; continuous  ck,
                               ; Synch, normal  mode
                               ; word  long  frame  sync.
                               ; FSL=0; ext  ck/fs
```

5.4.4 SSI 端口状态寄存器（SSISR）

SSISR 是 8 位只读寄存器，由 DSP56001 处理器用来询问状态和 SSI 端口的串行输入标志。状态寄存器占用 X 存贮器的 $FFEE 单元。

5.4.5 SSI 端口接收寄存器

SSI 端口的接收移位寄存器是 24 位寄存器，接收由串行接收数据（SRD）端来的数据。接收数据寄存器（RX）是 24 位寄存器，当接收移位寄存器装满时，它并行接收由接收移位寄存器来的数据。RX 占据 X 存贮器单元 $FFEF。当接收移位寄存器的内容传送到 RX 时，SSI 端口的状态寄存器（位 7）的接收数据寄存器满标志（RDF）被设置。当 DSP56001 处理器读接收数据寄存器时或当 DSP56001 处理器复位时，RDF 被清除。

在 DSP 系统中，当外部帧同步要求串行控制端（SC2）时，接收移位寄存器经串行接收数据端（SRD）接收串行输入数据。接收移位寄存器经串行时钟（SCK）端接外时钟。

5.4.6 SSI 端口发送寄存器

SSI 端口的发送移位寄存器是 24 位寄存器，经串行发送数据（STD）端发送串行数据。发送数据寄存器（TX）是 24 位寄存器，并行提供数据到发送移位寄存器。要发送的数据写入 TX 寄存器，并当要求帧同步时，自动传送到发送移位寄存器。TX 占据 X 存贮器单元 $FFEF。当 TX 的内容传送到发送移位寄存器时，SSI 端口的状态 寄存器（位 6）的发送数据寄存器空标志（TDE）被置定。当 DSP56001 处理 器写新数据到 TX 或当 DSP56001 处理器复位时，TDE 被清除。

在 DSP 系统中，当外部帧同步要求串行控制端（SC2）时，发送移位寄存器经串行发送数据端（STD）发送串行输出数据。发送移位寄存器经串行时钟端（SCK）接外时钟。

5.4.7 读 RX 并写入 TX

对 DSP 系统，DSP56001 处理器读 RX，并不加修正地将其内容传送到 TX。DSP56001 指令如下：

1. 查询 SSI 端口的状态寄存器（位 7）中的 RDF。

2. 当 RDF 置定时读 RX（RX 包含由 DSP56ADC16 A/D 转换器在 48kHz 采样率提供的数据）。

3. 传送 RX 内容到 TX（TX 包含要提供给 D/A 转换器的数据）。

4. 返回第 1 步。

```
poll  btst    #7, x: $FFEE   ; test for A/D data
      jcc     poll           ; loop until RDF bit=1
      movep   x: $FFE, a     ; move A/D data to "a"
      move    a, x: $FFEFF   ; send A/D data to D/A
      jmp     poll           ; loop indefinitely
```

5.5 试验“DSP 系统”

为了试验 DSP 系统，如 5.1 节所述，由函数发生器提供模拟输入信号到 EVB 的模拟输入。模拟输入信号由双踪示波器的一路监视。模拟输出信号由示波器的另一路监视，如图 5－8 所示。

ADM 上的 DSP56001 处理器执行绝对装入文件 evb.lod 以发送模拟信号的数字化形式。文件 evb.lod 被汇编并与图 5－9 所示和 5.4 节研究的汇编结果链接。

1. 去掉 PC 机和函数发生器电源。

2. 连接函数发生器输出到 EVB 的“BNC2”输入（A/D 输入）端。

3. 连接函数发生器输出到示波器通道 1 输入端。

4. 连接 EVB 的“BNC1”输出（D/A 输出）到示波器的通道 2 输入端。

5. 加电至函数发生器。

6. 置函数发生器到输出 2V（峰-峰值），5kHz 正弦波。

7. 加电并引导 PC 机。

8. 插“DSP56000/1 示范软件＃1”软盘到驱动器“A”。

9. 从驱动器 B 或硬盘（取决于 PC 机结构）调用“ADS56000 用户接口软件”程序。

10. 键入 load a:evb.lod<return>，装入图 5－9 所示汇编程序的汇编和链接版本。

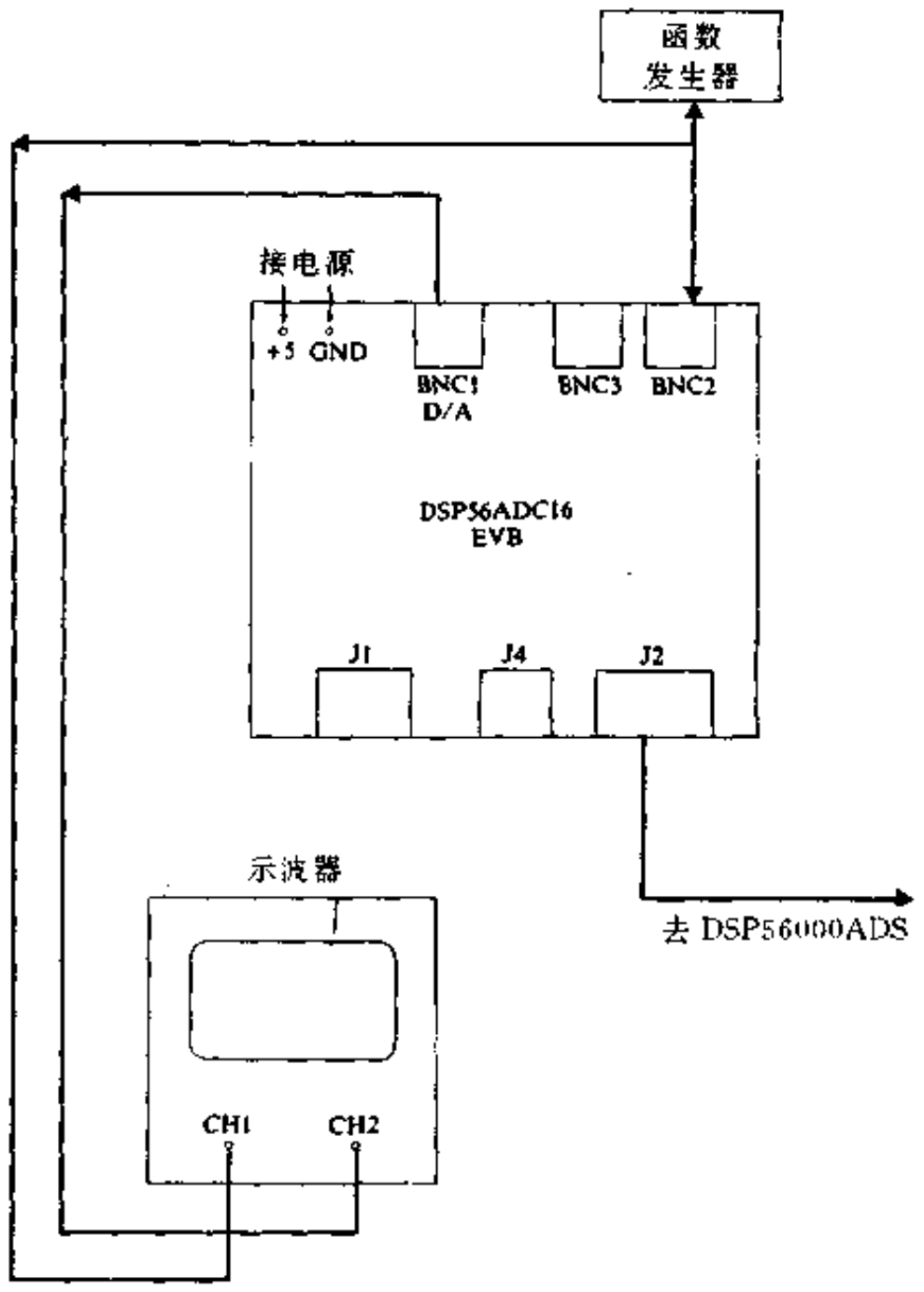


图 5－8　5.5 节中 DSP 连接图

11. 键入 go<return>，开始执行程序。

12. 观察示波器上输入和输出信号。注意输出是输入信号很好的恢复。

```
; Program start address

     org     p:$40

; Set up ADS board in case of force break instead of force
; reset

          movep #0,x:$FFFE     ;set bcr to zero
          movec #0,sp          ;init stack pointer
          movec #0,sr          ;clear loop flag

; Set up the SSI for operation with the EVB
; The following code sets Port C to function as SCI/SSI

        move    #$0,a0              ;zero PCC to cycle it
        movep   a0,x:$FFE1

pcc     equ     $0001ff
        movep   #pcc,x:$FFE1        ;write PCC

; The following code sets the SSI port's CRA and CRB
; control registers for external continuous clock,
; synchronous, normal mode.


cra     equ     $004000
        movep   #cra,x:$FFEC  ;CRA pattern for word
                              ;length=16 bits

crb     equ     $003200
        movep   #crb,x:$FFED  ;CRB pattern for continuous
                              ;ck,synch,normal mode
                              ;word long frame sync:
                              ;FSL=0;ext ck/fs

;*************************************************************
; Actual read A/D and write D/A
;*************************************************************

; The following code polls the RDF flag in the SSI port's
; SR  and waits for RDF=1, then reads the RX register to
; retrieve data from the A/D converter.
; Sample rate is controlled by EVB.

poll      btst    #7,x:$FFEE
          jcc     poll          ;loop until RDF bit = 1

          movep   x:$FFEF,a     ;get A/D converter data

; Write DSP56ADC16 A/D converter data to the D/A converter

          move    a,x:$FFEF     ;write the D/A via SSI
                                ; TX reg.
          jmp     poll          ;loop indefinitely
          end
```

图 5-9　为读 A/D 和写 D/A 的 EVB.ASM 汇编程序

13. 将函数发生器输出频率（f）从 0.5 kHz 调到 30 kHz。

14. 观察示波器上输入和输出信号。注意在频率从 0.5 kHz 到低于 20 kHz 输出是输入信号很好的恢复，因为以下两条被（必须）满足：

（1）为 A/D 和 D/A 转换器所选采样速率 48 kHz，满足对输入频率高达 24 kHz 的 Nyquist 采样定理，即 2 * 24 kHz＝48 kHz。

（2）输入频率低于 20 kHz 是小于 D/A 转换器输出处的低通滤波器的截止频率。

15. 将函数发生器调到输出 1V5 kHz 方波。

16. 观察示波器输入和输出信号。

17. 将方波输入信号频率从 0.5 kHz 调到 20 kHz。

18. 观察示波器输入和输出信号。

19. 键入 force b<return>，停止程序执行。

20. 键入 quit<return>，退出“ADS56000 用户接口软件”程序。

5.6 混叠对模拟输出信号的影响

5.5 节的 DSP 系统建立可用于表示混叠对输出信号的影响，即为 D/A 转换器提供的数字化采样要低于 Nyquist 采样率。若 DSP56001 程序对从 A/D 转换器读的每 N 个采样写一个采样到 D/A 转换器，则输出 采样率可降到 48/N kHz。

图 5－10 所示的 DSP56001 指令是 5.5 节中所用码的修正版本。注意下述码段

```
start
        do      # $2f,_end        ; 48 iteration loop
poll1   btst    #7, x: $FFEE
        jcc     poll1             ; loop until RDF=1
end     novep   x: $FFEF, a       ; get A/D data

poll2   btst    #7, x: $FFEE
        jcc     poll2             ; loop until RDF=1
        movep   x: $FFEF, a       ; get A/D data
```

用以降低输出采样率从 48 kHz 到 1 kHz。

在 ADM 上的 DSP56001 处理器执行 绝对装入 文件 evbm.lod 传送模拟信号的数字化形式。文件 evbm.lod 是图 5－10 所示汇编程序汇编和链接的结果。

1. 去掉 PC 机和函数发生器电源。

2. 连接函数发生器输出到 EVB 的“BNC2”输入（A/D 输入）端。

3. 连接函数发生器输出到示波器通道 1 输入端。

4. 连接 EVB 的“BNC1”输出（D/A 输出）到示波器通道 2 输入端。

5. 函数发生器加上电源。

6. 置定函数发生器输出为 2V 5kHz 正弦波。

7. 加电并导引 PC 机。

8. 将“DSP56000/1 示范软件＃1”软盘插入到驱动器“A”。

9. 从驱动器 B 或硬 盘（取决于 PC 机结构）调用“ADS56000 用户接口软件”程序。

```
; Program start address

     org     p:$40

; Set up ADS board in case of force break instead of force
; reset

          movep #0,x:$FFFE     ;set bcr to zero
          movec #0,sp          ;init stack pointer
          movec #0,sr          ;clear loop flag

; Set up the SSI for operation with the EVB
; The following code sets port C to function as SCI/SSI

        move    #$0,a0              ;zero PCC to cycle it
        movep   a0,x:$FFE1

pcc     equ     $0001ff
        movep   #pcc,x:$FFE1        ;write PCC

; The following code sets the SSI CRA and CRB control
; registers for external continuous clock, synchronous,
; normal mode.

cra     equ     $004000
        movep   #cra,x:$FFEC  ;CRA pattern for word
                              ;length=16 bits

crb     equ     $003200
        movep   #crb,x:$FFED  ;CRB pattern for continuous
                              ;ck,synch,normal mode
                              ;word long frame Sync:
                              ;FSL=0;ext ck/fs

;*************************************************************
; Actual read A/D and write D/A
;*************************************************************

; The following code writes one sample to the D/A
; converter for every 48 samples it reads from the A/D
; converter.

start

          do      #$2f,end
poll1     btst    #7,x:$FFEE
          jcc     poll1         ;loop until RDF bit = 1

          movep   x:$FFEF,a     ;get A/D converter data
end

poll2     btst    #7,x:$FFEE
          jcc     poll2         ;loop until RDF bit = 1

          movep   x:$FFEF,a     ;get A/D converter data

; Write DSP56ADC16 A/D converter data to the D/A converter

          move    a,x:$FFEF     ;write the D/A via SSI
                                ; TX reg.
          jmp     start         ;loop indefinitely

          end
```

图 5－10　为读 A/D 和写 D/A 的 EVBM.ASM 汇编程序

10. 键入 load a；evbm.lod<return>，装入图 5－10 所示汇编程序的汇编和链接版本。

11. 键入 go<return>，开始执行程序。

12. 观察示波器上的输入输出信号。注意因 1 kHz 输出采样率低于 10 kHz Nyquist 采样率（2＊5 kHz＝10 kHz），输出信号产生混叠，因此模拟输入信号不能恢复。

13. 键入 force b<return>以停止程序执行。

14. 键入 quit<return>退出“ADS56000 用户接口软件”程序。

5.7 听混叠的影响

若将一键盘、放大器和扬声器加到 DSP 系统上，如图 5－11，混叠的影响可以听见。

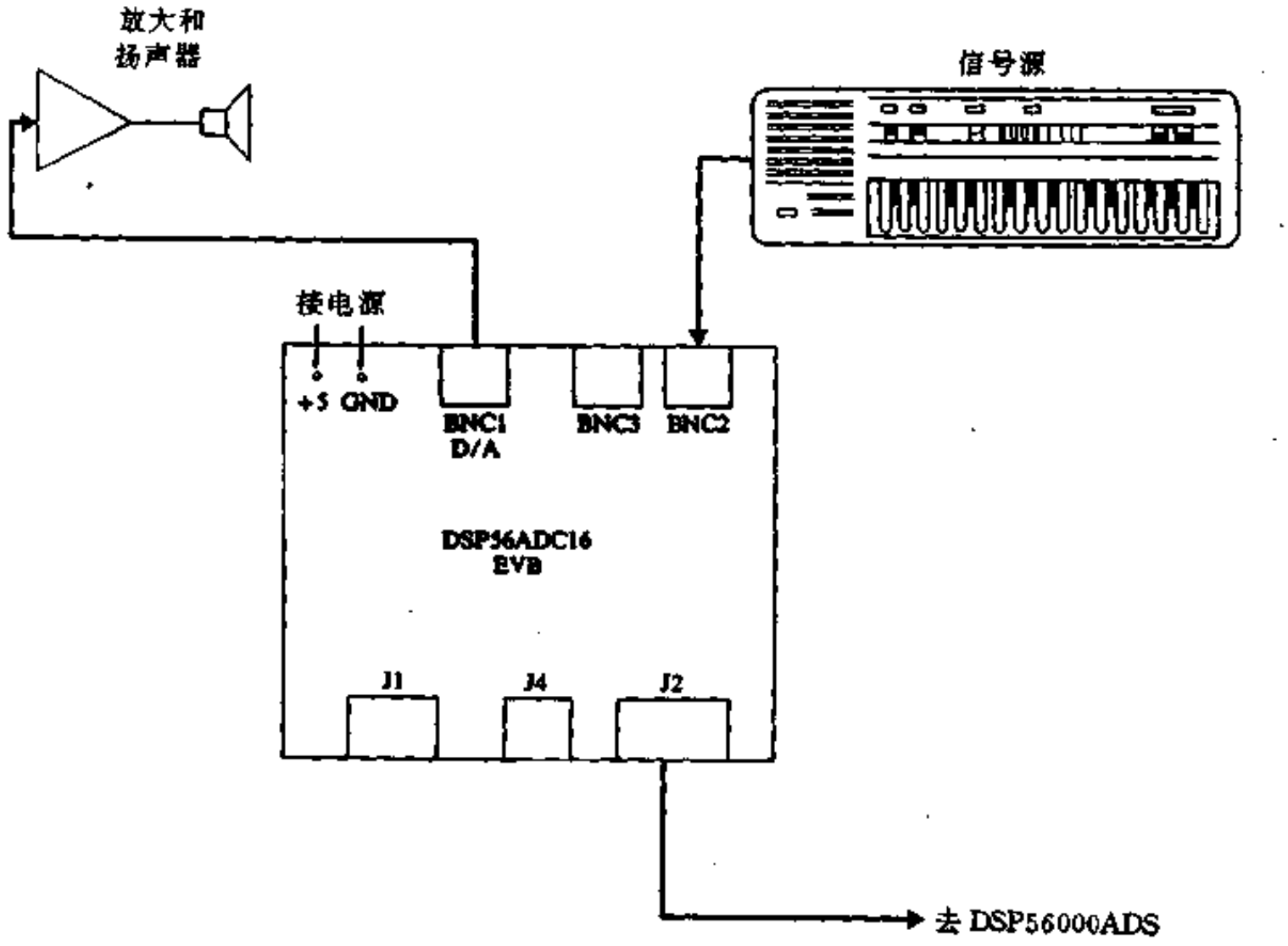


图 5－11 因不适当采样而造成混叠的影响

1. 去掉 PC 机电源。

2. 连接键盘到 EVB 的“BNC2”输入端。

3. 连接放大器和扬声器到 EVB“BNC1”输出端。

4. 加电并导引 PC 机。

5. 把“DSP 56000/1 示范软件＃1”软盘插入驱动器“A”。

6. 从驱动器 B 或硬盘（取决于 PC 机结构）调用“ADS 56000 用户接口软件”程序。

7. 键入 load a；evb.lod<return>，装入图 5－8 所示汇编程序（48 kHz 采样率）的汇编和链接版本。

8. 键入 go<return>，开始执行程序。

9. 由键盘送入音乐，并听扬声器的声音。此时声音应很好。

10. 键入 force b<return>以停止程序执行。

11. 键入 load a：evbm.lod<return>装入图 5－10 所示的汇编程序（1kHz 采样率）。

12. 由键盘送入音乐，并听声音。音频低于 500 Hz，即中音“A”是 440 Hz，可以无混叠地恢复。音频高于 500 Hz 时，会产生混叠。

13. 键入 force b<return>以停止执行程序。

14. 键入 quit<return>，退出“ADS 56000 用户接口软件”程序。

第六章　FIR 滤波器设计及在 DSP56001 处理器上的实现

本章设计了有限脉冲响应（FIR）滤波器，并在DSP56001 处理器上实现。FIR 滤波器有三种独特的性质：

（1）常稳定的。

（2）常可靠的。

（3）有线性相位响应。

本章先定义了 FIR 滤波器的传输函数和脉冲响应，随之描述如何由频率响应得到理想 FIR 滤波器的理想脉冲响应。因 FIR 滤波器的理想脉冲响应序列是无限长，故设计和使用了适当“窗”函数以产生实际的、有限长度脉冲响应序列。

在实际 FIR 滤波器设计过程中，用所要求的频率域指标以适当的窗函数去决定其脉冲响应 h(i)。DSP56000/1 示范软件程序和 Momentum 数据系统公司的专用滤波器设计和分析系统软件程序的专用说明版本，用于说明此设计过程。然后叙述 DSP56001 指令在 DSP56001 处理器上实现 FIR 滤波器的设计。

其后在 DSP 系统上设计和实现低通、高通、带通和带阻 FIR 滤波器，并将 DSP 系统置示于图 6－1。实现这 4 种 FIR 滤波器的 DSP56001 指令包含在 DSP56000/1 示范软件＃2 软盘中。

图 6－1　DSP 系统

6.1　FIR 滤波器频率响应

FIR 滤波器的数字输入序列 x(n)和输出序列 y(n)间的关系可写为：

$$y(n) = \sum_{i=0}^{N-1} b_i x(n-i) \tag{6-1}$$

其中 b_i 是滤波系数，N 是系数的数目。系数 b_i 决定了滤波特性。按定义和设计，FIR 滤波器（为非递归滤波器）的输出序列 y(n) 仅为其当前和过去输入的函数。

相当于式（6－1）的 Z 变换传输函数 H(z) 可表示为：

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{i=0}^{N-1} b_i z^{-i} = \sum_{i=0}^{N-1} h(i) z^{-i} \tag{6-2}$$

其中 FIR 滤波器脉冲响应 h(i) 由系数 b_i 组成。

为了决定 FIR 滤波器的频率响应，变量 z 在单位圆上取值：

$$z = e^{j2\pi f} \tag{6-3}$$

其中 f 为归一化频率，即实际频率被采样频率除。

以（6-3）式代入（6-2）式，FIR 滤波器的频率响应可写为

$$H(e^{j2\pi f}) = \sum_{i=0}^{N-1} h(i)e^{-j2\pi fi} \tag{6-4}$$

6.2 实际 FIR 滤波器与理想滤波器的关系

在实际 FIR 滤波器设计过程中，用其要求的频域指标以适当窗函数决定它的脉冲响应 h(i)。设计（6-2）式中传输函数的系数 h(i)（或 bi）使（6-4）式频率响应满足一定的频域指标。一旦得到脉冲响应，用 b_i 分量在 DSP56001 处理器上实现（6-1）式。

设计过程的第一步是获得理想 FIR 滤波器的频率响应。按定义，理想 FIR 滤波器在通带内的值为 1，其他地方为 0。理想低通、高通、带通和带阻滤 波器的频率响应如图 6-2 所示。

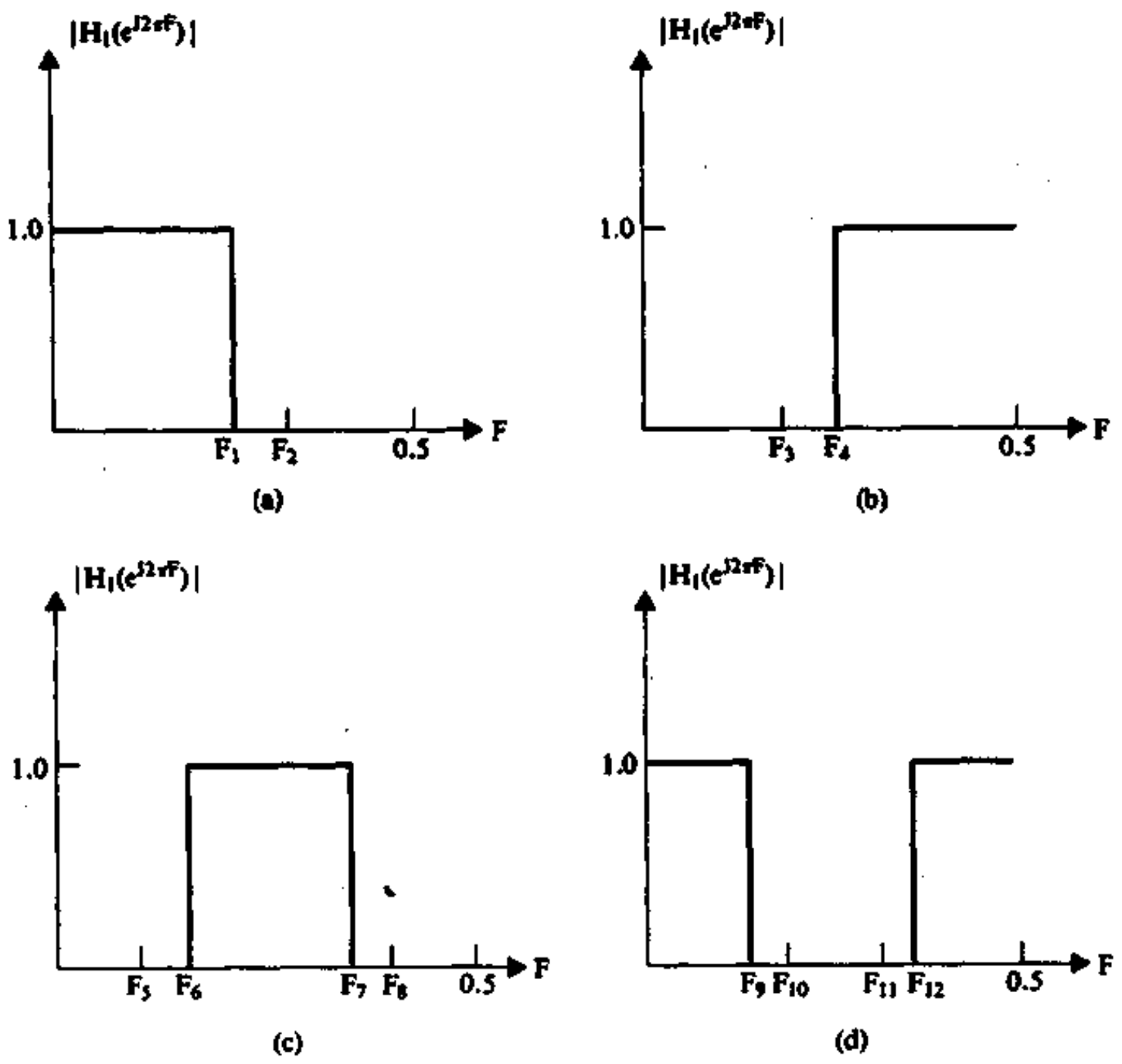


图 6-2 理想滤波器：(a) 低通；(b) 高通；(c) 带通；(d) 带阻

理想 FIR 滤波器的频率响应可写为：

$$H_1(e^{j2\pi f}) = \sum_{i=-\infty}^{\infty} h_1(i)e^{-j2\pi fi} \tag{6-5}$$

其中理想脉冲响应 $h_1(i)$ 可如下计算：

$$h_1(i) = \int_{-0.5}^{0.5} H_1(e^{j2\pi f})e^{j2\pi fi}df \tag{6-6}$$

式中 0.5 是归一化采样频率的一半。

仅当脉冲响应为有限长时，可实现实际的 FIR 滤波器。因为理想脉冲响应 $h_1(i)$ 是无限长序列，它必须被限制到有限长度。为此，$h_1(i)$ 乘以 N 点窗，则 N 点脉冲响应 $h_2(i)$ 是：

$$\begin{aligned} h_2(i) &= h_1(i)w(i) \qquad -(N-1)/2 \leqslant i \leqslant (N-1)/2 \\ &= 0 \qquad\qquad \text{其他} \end{aligned} \tag{6-7}$$

其中 N 是奇整数。一些常用窗型式如图 6-3 所示。

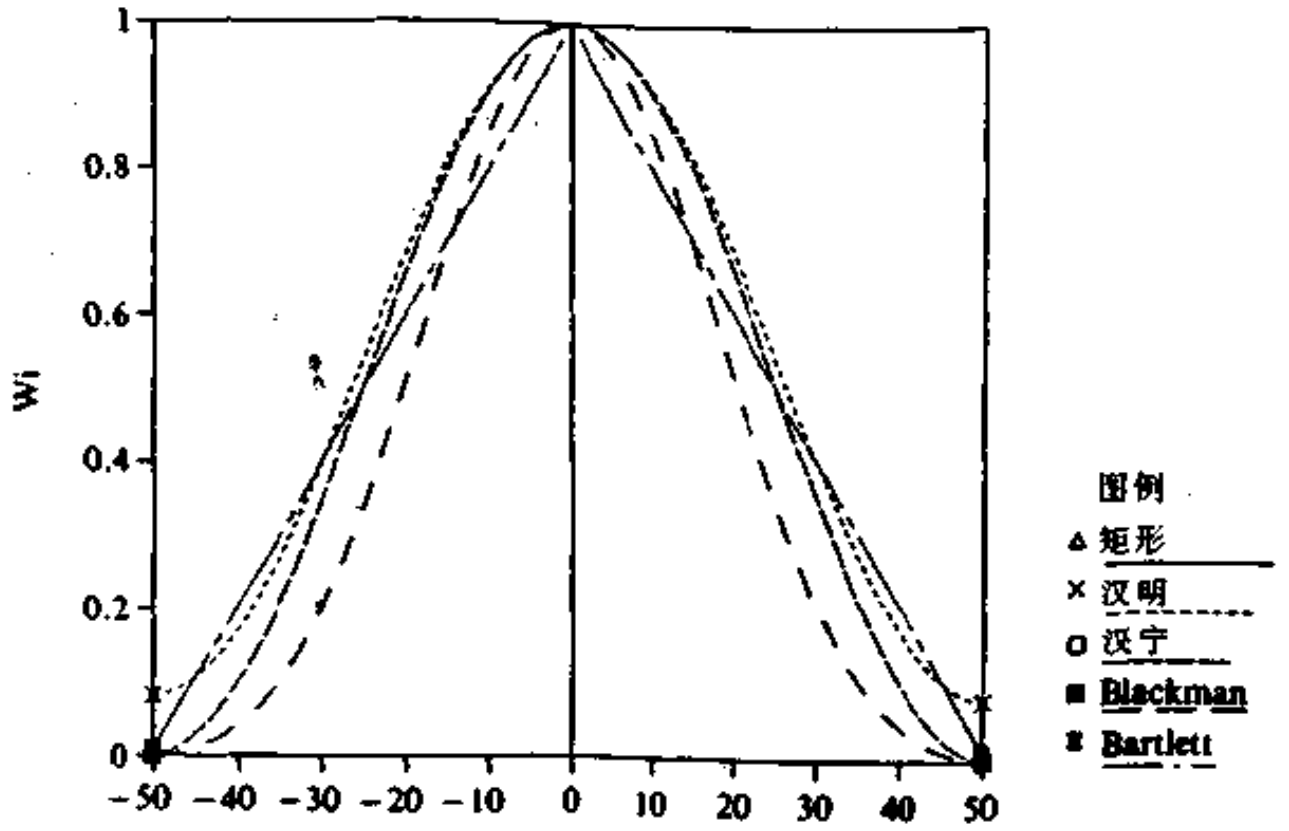


图 6-3 常用窗

因为对 i<0 因果滤波器有零响应，所以 FIR 滤波器可充分延迟其脉冲响应（若必需）做到满足这一条件。也就是说，$h_2(i)$ 可延迟 $(N-1)/2$，使之可得到因果滤波器所要求的脉冲响应 h(i)：

$$h(i) = h_2[i - ((N-1)/2)] \qquad i = 0,1,2,\cdots,N-1 \tag{6-8}$$

由选择 h(i) 等于 h(N-1-i)　$i = 0,\cdots,(N-1)/2$，因果 FIR 滤波器可做到具有线性相位响应。用 h(i) 这一选择，(6-4) 式可写为：

$$H(e^{j2\pi f}) = \sum_{i=0}^{(N-3)/2} h(i)e^{-j2\pi fi} + h(0.5(N-1))e^{-j2\pi f(N-1)/2} + \sum_{i=(N+1)/2}^{(N-1)} h(i)e^{-j2\pi fi} \tag{6-9}$$

令 N-1-i=i，(6-9) 式中最后和项可表示为：

$$\sum_{i=(N+1)/2}^{N-1} h(i)e^{-j2\pi fi} = \sum_{i=(N+1)/2}^{N-1} h(N-1-i)e^{-j2\pi fi} = \sum_{i=0}^{(N-3)/2} h(i)e^{-j2\pi f(N-1-i)} \tag{6-10}$$

由 (6-9) 和 (6-10) 式可得：

$$H(e^{j2\pi f}) = e^{-j\pi f(N-1)}\left[h(0.5(N-1)) + \sum_{i=0}^{(N-3)/2} h(i)(e^{j\pi f(N-1-2i)} + e^{-j\pi f(N-1-2i)})\right]$$

$$= e^{-j\pi f(N-1)}\left[h(0.5(N-1)) + \sum_{i=0}^{(N-3)/2} 2h(i)\cos(\pi f(N-1-2i))\right] \quad (6-11)$$

对实的 h(i)，(6-11) 式中两括号之间的项是 f 的实函数。若此项为正，则 $H(e^{j2\pi f})$ 的相位等于 $-\pi f(N-1)$。若此项为负，则 $H(e^{j2\pi f})$ 的相位等于 $\pi - \pi f(N-1)$。在任一情况下，$H(e^{j2\pi f})$ 的相位都是 f 的线性函数 。

6.3 确定实际 FIR 滤波器

实际 FIR 滤波器的频域指标包括通带和阻带衰减，通带和阻带截止频率。图 6-4 表示低通、高通、带通和带阻 FIR 滤波器的频域指标，其中：

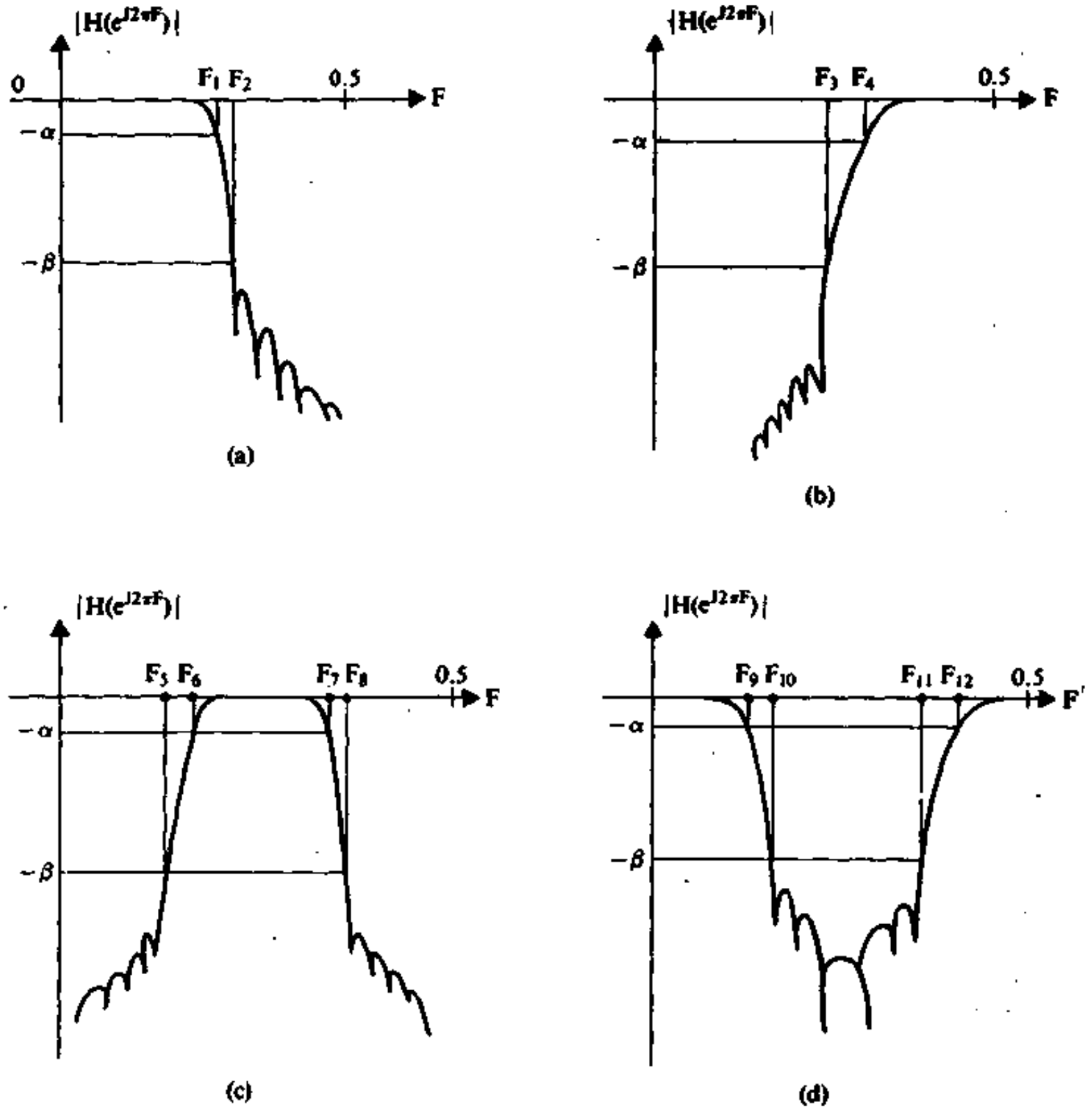


图 6-4 频域指标：(a) 低通，(b) 高通，(c) 带通，(d) 带阻

α=通带衰减 dB。

β=阻带衰减 dB。

f_1=低通滤波器归一化通带截止频率。

f_2=低通滤波器归一化阻带截止频率。

f_3=高通滤波器归一化阻带截止频率。

f_4=高通滤波器归一化通带截止频率。

f_5=带通滤波器归一化低阻带截止频率。

f_6=带通滤波器归一化低通带截止频率。

f_7=带通滤波器归一化高通带截止频率。

f_8=带通滤波器归一化高阻带截止频率。

f_9=带阻滤波器归一化低通带截止频率。

f_{10}=带阻滤波器归一化低阻带截止频率。

f_{11}=带阻滤波器归一化高阻带截止频率。

f_{12}=带阻滤波器归一化高通带截止频率。

6.4 用窗方法设计实际滤波器

现有三种已知方法可设计实际的线性相位响应 FIR 滤波器：

(1) 频率采样法。

(2) “最大最小”方法。

(3) 窗方法。

此书中用窗方法。过程是：

1. 计算理想脉冲响应 $h_1(i)$

使用公式 (6-6)，低通、高通、带通和带阻 FIR 滤波器的理想脉冲响应 $h_1(i)$ 可作如下计算：

低通
$$h_1(i)=\frac{\sin 2\pi f_1 i}{\pi i} \tag{6-12}$$

高通
$$h_1(i)=-\frac{\sin 2\pi f_4 i}{\pi i} \tag{6-13}$$

带通
$$h_1(i)=\frac{\sin 2\pi f_7 i}{\pi i}-\frac{\sin 2\pi f_6 i}{\pi i} \tag{6-14}$$

带阻
$$h_1(i)=\frac{\sin 2\pi f_9 i}{\pi i}-\frac{\sin 2\pi f_{12} i}{\pi i} \tag{6-15}$$

2. 选择窗型式

为了达到要求的阻带衰减水平，必须从表 6-1 选择适当的窗型式。例如，若要求的阻带衰减是 50dB，则从表 6-1 可见，Hamming、Blackman 或 Kaiser 窗都合适。

3. 选择窗长度“N”

下面的关系可用于近似所需的窗长度 N：

$$N \geqslant K/\Delta f \tag{6-16}$$

其中 N 是满足（6－16）式的最小奇整数，K 是由表 6－1 所选的常数，以及

表 6－1 窗型选择表

窗	阻带衰减（dB）	K
Rectangular	21	2.00
Bartlett	25	4.00
Hanning	44	4.00
Hamming	53	4.00
Blackman	74	6.00
Kaiser（$\gamma=2.12$）	30	1.54
Kaiser（$\gamma=4.54$）	50	2.93
Kaiser（$\gamma=7.76$）	70	4.32
Kaiser（$\gamma=8.96$）	90	5.71

Δf ＝以 Hz 表示的过渡带

$=f_2-f_1$ 对低通滤波器

$=f_4-f_3$ 对高通滤波器

$=\min[(f_6-f_5),(f_8-f_7)]$ 对带通滤波器

$=\min[(f_{10}-f_9),(f_{12}-f_{22})]$ 对带阻滤波器

4. 由表 6－2 选窗函数 w（i）

表 6－2 窗函数

窗型式	窗函数 $w(i)\ |i| \leqslant (N-1)/2$
Rectangular	1
Bartlett	$1-2(|i|/N)$
Hanning	$0.5+0.5\cos(2i/N)$
Hamming	$0.54+0.46\cos(2\pi i/N)$
Blackman	$0.42+0.5\cos(2\pi i/N)+0.08\cos(4\pi i/N)$
Kaiser*	$I_0[\gamma(1-2i/(N-1))^2]^{0.5}/I_0(\gamma)$

*I_0 是修正零阶 Bessel 函数。Fortran 子程序“Bessel”示于图 6－5，可用来计算零阶修正 Bessel 函数。

5. 用（6－7）式计算 $h_2(i)$

$h_2(i)$ 的计算是用适当的选自 6－12～6－15 式的 $h_1(i)$ 乘选自表 6－2 的 w(i)。

6. 用（6－8）式计算 h(i)

（6－4）式可用于计算频率响应的大小。频率响应可用快速傅里叶变换（第八章）计算。所设计的 FIR 滤波器可通过测试验证所要求的频域指标。

注意由（6－16）式所得 N 值高于为满足所需频域指标所要求的值。N 可以按偶整数增（减）以增（减）FIR 滤波器的“阶数”，并可重复进行直到滤波器阶数最好地满足频域指标。

```
C*************************************************
C*                                               *
C*  SUBROUTINE BESSEL                            *
C*                                               *
C*                                               *
C*************************************************
C
C        I0(X) = XBESS
C
C
C
         SUBROUTINE BESSEL (X,XBESS)
         INTEGER K
         DOUBLE PRECISION IF
         S=1.0
         K=1
         IF=1
         E1=10000.0
         E2=E1
77       IF=IF*K
         E1=E2
         E2=((X/2.0)**K/IF)**2
         S=S+E2
         ERR1=E2
         ERR2=E2/E1
         IF (ERR1.LT.0.00001) GOTO 88
         IF (ERR2.LT.0.00001) GOTO 88
         IF (K.GT.100) GOTO 99
         K=K+1
         GOTO 77
88       XBESS=S
99       RETURN
         END
```

图 6-5 Fortran 子程序“Bessel”

6.5 用软件程序设计实际 FIR 滤波器

DSP56000/1 示范软件程序可用来设计实际的线性相位响应 FIR 滤波器，过程如下：

1. 引导 PC 机。

2. 把“DSP56000/1 示范软件＃2”程序软盘插入驱动器“A”。

3. 键入 a：chapter6<return>，装入 DSP56000/1 示范软件第六章程序。

4. 从屏幕菜单选择以设计所要求的 FIR 滤波器。

5. 递减（递增）滤波器阶数直到获得所要求的频域指标。

注意：所设计的滤波器 h(i) 存在 DSP56000/1 示范软件＃2 程序软盘的数据文件 FIR-wxx.DAT 中，其中 xx 相应以 LP（低通）、HP（高通）、BP（带通）和 BS（带阻）代替。

6.5.1 低通滤波器设计举例

用户定义的指标：

采样频率＝48kHz

阻带衰减＝50dB

通带边界＝8kHz

阻带边界＝10kHz;

窗 型 式＝Hamming 窗

所设计的通带衰减 0.020dB，阻带衰减 52.745dB。

因果低通滤波器系数是：

h(1) = .11213E-02=h(41)
h(2) = .13362E-02=h(40)
h(3) = .95611E-10=h(39)
h(4) =- .23447E-02=h(38)
h(5) =- .31894E-02=h(37)
h(6) =- .33537E-08=h(36)
h(7) = .57094E-02=h(35)
h(8) = .74605E-02=h(34)
h(9) = .38783E-08=h(33)
h(10) =- .12211E-01=h(32)
h(11) =- .15372E-01=h(31)
h(12) =- .52937E-09=h(30)
h(13) = .23965E-01=h(29)
h(14) = .29919E-01=h(28)
h(15) = .75957E-08=h(27)
h(16) =- .48045E-01=h(26)
h(17) =- .63144E-01=h(25)
h(18) =- .88328E-08=h(24)
h(19) = .13488E+00=h(23)
h(20) = .27418E+00=h(22)
h(21) = .33333E+00=h(21)

6.5.2 高通滤波器设计举例

用户定义的指标：

采样频率＝48kHz

阻带衰减＝50dB

通带边界＝10kHz

阻带边界＝8kHz

窗 型 式＝Blackman 窗

所设计的通带衰减 0.026dB，阻带衰减 50.312dB。

因果高通滤波器系数是：

h(1) =-.19380E-05=h(53)
h(2) =-.35399E-04=h(52)
h(3) = .60145E-10=h(51)
h(4) = .21958E-03=h(50)
h(5) = .20330E-03=h(49)
h(6) =-.46795E-03=h(48)
h(7) =-.87685E-03=h(47)
h(8) = .38364E-03=h(46)
h(9) = .20988E-02=h(45)
h(10) = .74902E-03=h(44)
h(11) =-.33817E-02=h(43)
h(12) =-.36583E-02=h(42)
h(13) = .33748E-02=h(41)
h(14) = .83951E-02=h(40)
h(15) =-.30789E-08=h(39)
h(16) =-.13498E-01=h(38)
h(17) =-.87617E-02=h(37)
h(18) = .15473E-01=h(36)
h(19) = .23634E-01=h(35)
h(20) =-.88315E-02=h(34)
h(21) =-.43002E-01=h(33)
h(22) =-.14249E-01=h(32)
h(23) = .62815E-01=h(31)
h(24) = .71223E-01=h(30)
h(25) =-.77762E-01=h(29)
h(26) =-.30570E+00=h(28)

h(27) = .58333E+00=h(27)

6.5.3 带通滤波器设计举例

用户定义的指标：

采样频率=48 kHz
阻带衰减=50 dB
低通带边界6 kHz
高通带边界12 kHz
低阻带边界4 kHz
高阻带边界14 kHz
窗 型 式 =Blackman 窗

所设计的通带衰减 0.021 dB，阻带衰减 52.223 dB。

因果带通滤波器系数是：

h(1) = .12388E-10=h(57)
h(2) =- .50005E-04=h(56)
h(3) =- .85938E-04=h(55)
h(4) = .52566E-04=h(54)
h(5) = .15195E-10=h(53)
h(6) =- .15064E-03=h(52)
h(7) = .78036E-03=h(51)
h(8) = .19343E-02=h(50)
h(9) = .18979E-09=h(49)
h(10) =- .37179E-02=h(48)
h(11) =- .29173E-02=h(47)
h(12) = .11243E-02=h(46)
h(13) = .17394E-08=h(45)
h(14) =- .18634E-02=h(44)
h(15) = .80464E-02=h(43)
h(16) = .17209E-01=h(42)
h(17) = .89654E-09=h(41)
h(18) =- .26428E-01=h(40)
h(19) =- .19047E-01=h(39)
h(20) = .68508E-02=h(38)
h(21) = .50265E-08=h(37)
h(22) =- .10397E-01=h(36)
h(23) = .44256E-01=h(35)
h(24) = .95863E-01=h(34)
h(25) = .17518E-08=h(33)
h(26) =- .17317E+00=h(32)
h(27) =- .15601E+00=h(31)
h(28) = .92767E-01=h(30)
h(29) = .25000E+00=h(29)

6.5.4 带阻滤波器设计举例

用户定义的指标：

采样频率 =48kHz
阻带衰减 =50dB
低通带边界=4kHz
高通带边界=14kHz
低阻带边界=6kHz
高阻带边界=12kHz
窗 型 式 =Hamming 窗

所设计的通带衰减 0.017dB，阻带衰减 54.043dB。

因果带阻滤波器系数是：

h(1) = .52248E-03=h(47)
h(2) =- .17633E-02=h(46)
h(3) =- .27286E-02=h(45)
h(4) =- .12283E-09=h(44)
h(5) =- .64955E-03=h(43)
h(6) =- .35163E-02=h(42)
h(7) = .34528E-02=h(41)
h(8) = .10078E-01=h(40)
h(9) = .21528E-02=h(39)
h(10) = .33598E-02=h(38)
h(11) = .16636E-01=h(37)
h(12) = .99570E-09=h(36)
h(13) =- .24860E-01=h(35)
h(14) =- .75339E-02=h(34)
h(15) =- .73065E-02=h(33)
h(16) =- .52450E-01=h(32)
h(17) =- .28048E-01=h(31)
h(18) = .45613E-01=h(30)
h(19) = .13834E-01=h(29)
h(20) =- .88802E-09=h(28)
h(21) = .17452E+00=h(27)
h(22) = .21386E+00=h(26)
h(23) =- .14770E+00=h(25)
h(24) = .58333E+00=h(24)

6.6 FIR 滤波器的实现

(6-1) 式用于实现 FIR 滤波器。通常，需完成以下步骤：

1. 存入输入采样值 x(n)。
2. x(n) 乘 b_0，并累加滤波器状态 x(n-i) 和 b_i 的乘积以产生输出样值 y(n)。
3. 位移滤波器状态，得到下一输出样值 y(n+1)。

6.6.1 在 DSP56001 处理器上实现 FIR 滤波器

按以下步骤可在 DSP56001 处理器上有效地实现 6.6 节中的第 1～3 步。

1. 地址指针向模拟 FIFO 位移 RAM 数据。
2. 采用模寻址功能以提供循环数据缓存。
3. 在单指令周期中用乘/累加（MAC）指令去乘两个操作数并加此乘积到第三操作数。
4. 用 MAC 指令的数据并行传送能力保持相乘器按 100%能力运行。
5. 执行 REP 指令以提供紧致滤波器码。

DSP56001 处理器的地址产生单元由 8 个地址寄存器（Rn）、8 个偏置寄存器（Nn）和 8 个模运算寄存器（Mn）组成。可完成模寻址的 DSP56001 处理器允许地址寄存器（Rn）的值增 1（或减 1），并仍保持在大小为 L 的地址范围内，这里 L 由低和高地址界限定义。

对 FIR 滤波器，L 等于系数的数目。L-1 值存在 DSP56001 处理器的修正寄存器（Mn）中。由于实现了 DSP56001 处理器的模寻址机理，L 的低界（基本地址）在其 K LSB 中必须为 0，这里 2**K≥L，即基本地址值必须是 2**K 的倍数。上地址界是基本地址加 L-1。注意上地址界不存在寄存器中。

当用模寻址时，地址寄存器（Rn）在 X 或 Y 存贮器中指定模数据缓存器。地址指针(Rn)不要求指在低地址界上；即可指在所定义的模地址范围 L 内的任何地方。若地址指针增 1 经过上地址界，则将返回到基本地址。

6.6.2 用于实现 FIR 滤波器的 DSP56001 指令

图 6-6 所示的 DSP56001 指令用来实现 FIR 滤波器算法，这些指令包括：

```
;
;  FIR  Filter
;
;  x0 = input sample
;  a = output sample
;  ntaps = number of coefficient taps in the filter
;
        move    #states,r0  ;point to filter states
        move    #ntaps-1,m0 ;mod(ntaps)
        move    #coef,r4    ;point to filter coefficients
        move    #ntaps-1,m4 ;mod(ntaps)

        clr     a             x0,x:(r0)+      y:(r4)+,y0
        rep     #ntapa-1
        mac     x0,y0,a       x:(r0)+,x0      y:(r4)+,y0
        macr    x0,y0,a       (r0)-
```

图 6-6 用于实现 FIR 滤波器算法的 DSP56001 汇编指令

1. 模寄存器 M0 编程为值 NTAPS-1（模 NTAPS）

地址寄存器 R0 编程到指向位于 X 存贮器中的状态可变的模缓存器。模寄存器 M4 编程到值 NTAPS-1。地址寄存器 R4 编程到指向位于 Y 存贮器的系数缓存器。给定 FIR 滤波器算法已执行一些时间，并准备处理在数据 ALU 输入寄存器 X0 中的样值 x(n)，R4 中地址是系数缓存器的基本地址。R0 中的地址是 M，这里 M 大于或等于低界 X 存贮器地址并小于或等于高界 X 存贮器地址。X 存贮器映射滤波器状态，Y 存贮器映射系数，DSP56001 处理器的 A 累加器和数据 ALU 输入寄存器 X0 和 Y0 的内容示于图 6-7。

2. CLR 指令

该指令清除 A 累加器，同时：

(1)将输入样值 x(n) 从数据 ALU 的输入寄存器 X0 传送到由地址寄存器 R0 指示位置的 X 存贮器。

(2)将第一个系数从由地址寄存器 R4 指示单元的 Y 存贮器传送到数据 ALU 的输入寄存器 Y0。R0 和 R4 在 CLR 指令结束时自动增 1（后递增）。

CLR 指令执行后，X 存贮器映射滤波器状态，Y 存贮器映射系数，以及 A 累加器和寄存器 X0 和 Y0 的内容如图 6-8 所示。

3. REP 指令

该指令管理以下指令的迭代执行：

mac x0，y0，a　x:(r0)+,x0　y:(r4)+,y0

4. MAC 指令

该指令将以 Y0 中的系数乘 X0 中的状态变量，并把乘积加到 A 累加器中，同时执行以下操作：

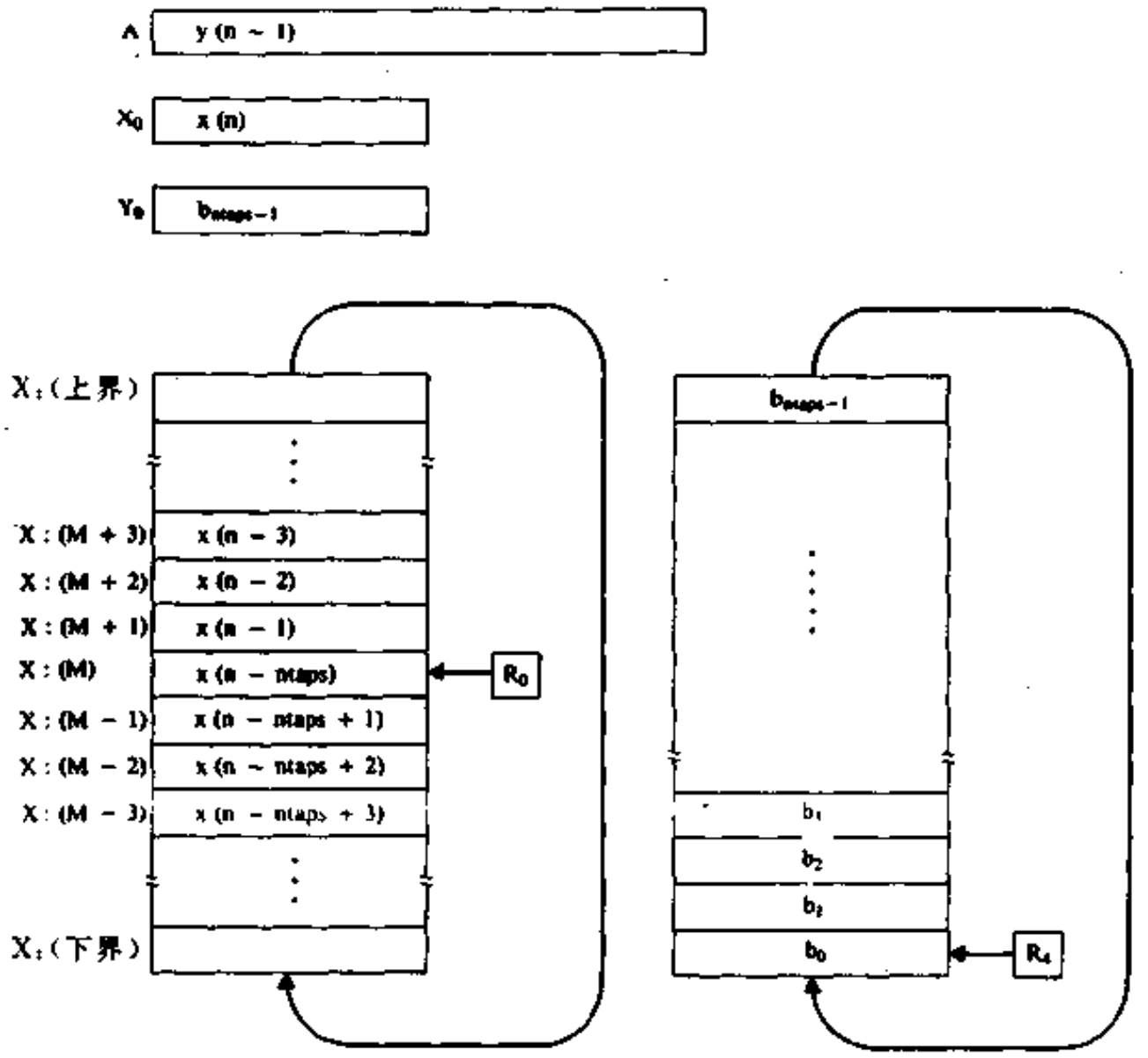


图 6－7　在 n 迭代开始时存贮器映像和数据寄贮器

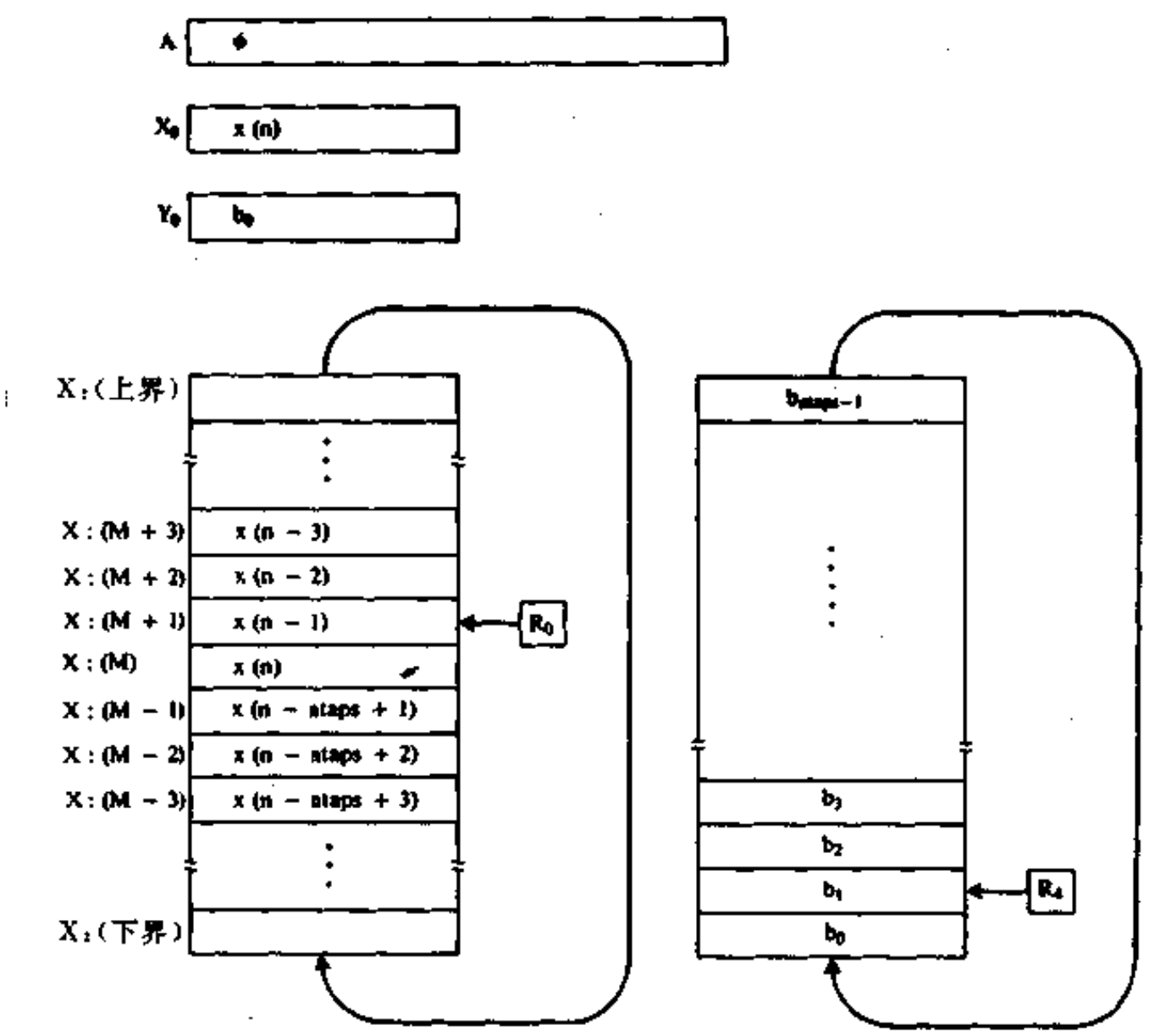


图 6－8　CLR 指令执行后存贮器映射和数据寄存器

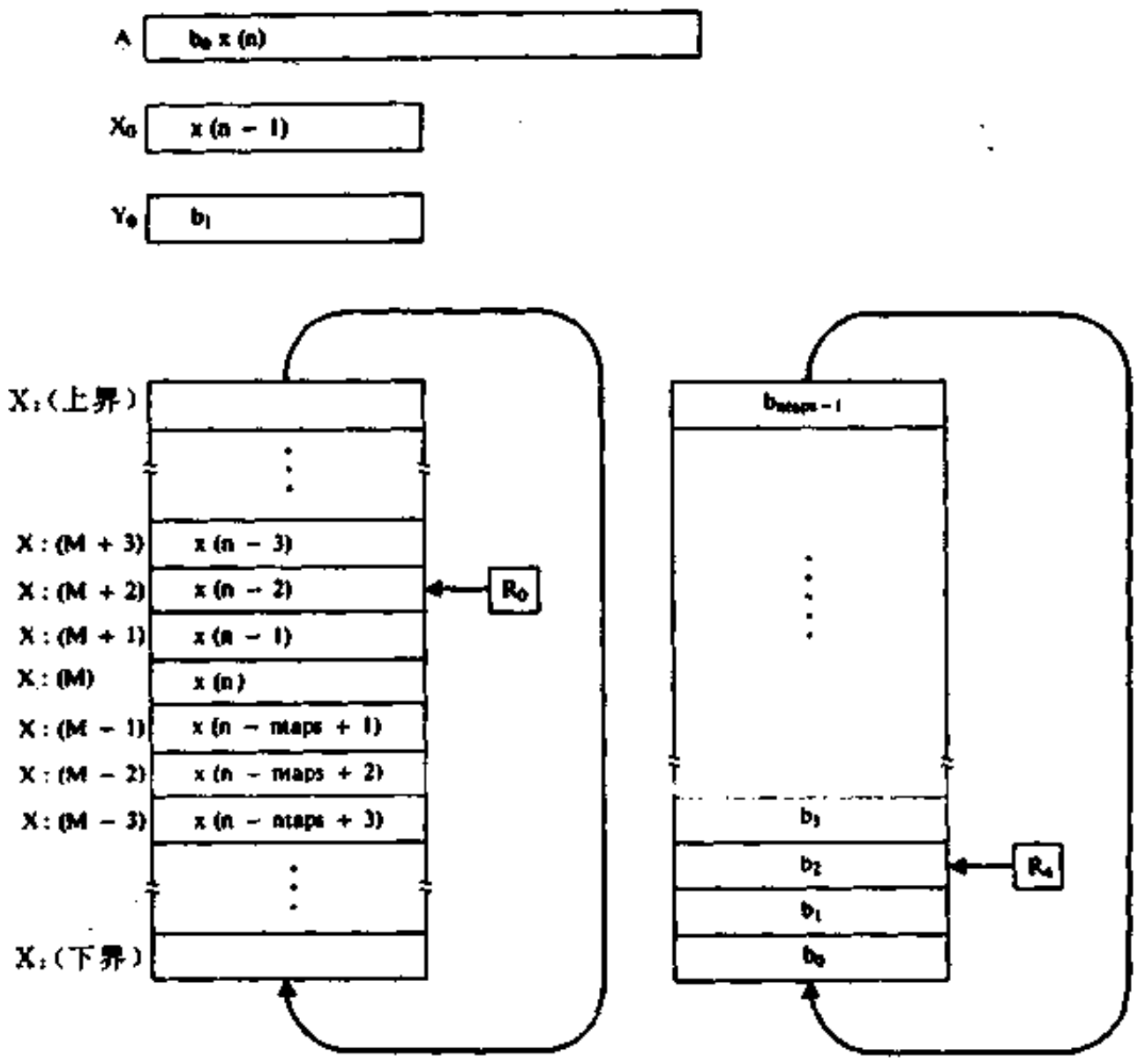


图 6－9　第一个 MAC 指令执行后的存贮器映像和数据寄存器

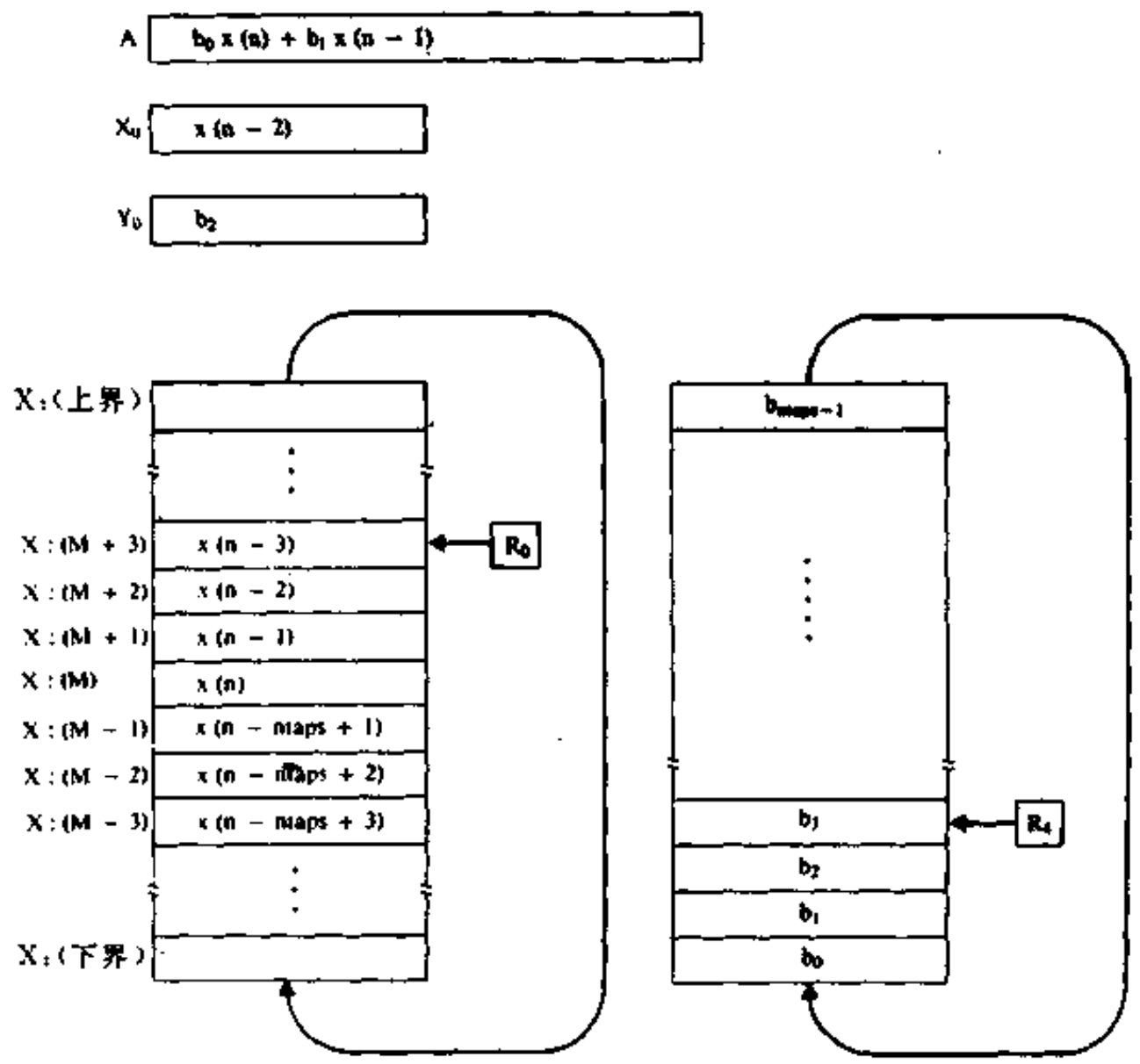


图 6－10　第二个 MAC 指令执行后的存贮器映像和数据寄存器

(1) 将下一个状态变量从 R0 所指的 X 存贮器传送到输入寄存器 X0。

(2) 将下一个系数从 R4 所指的 Y 存贮器传送到输入寄存器 Y0。R0 和 R4 在 MAC 指令结束时自动加 1（后递增）。

第一、第二和最后的 MAC 指令执行后，各存贮器情况分别示于图 6－9～图 6－11。

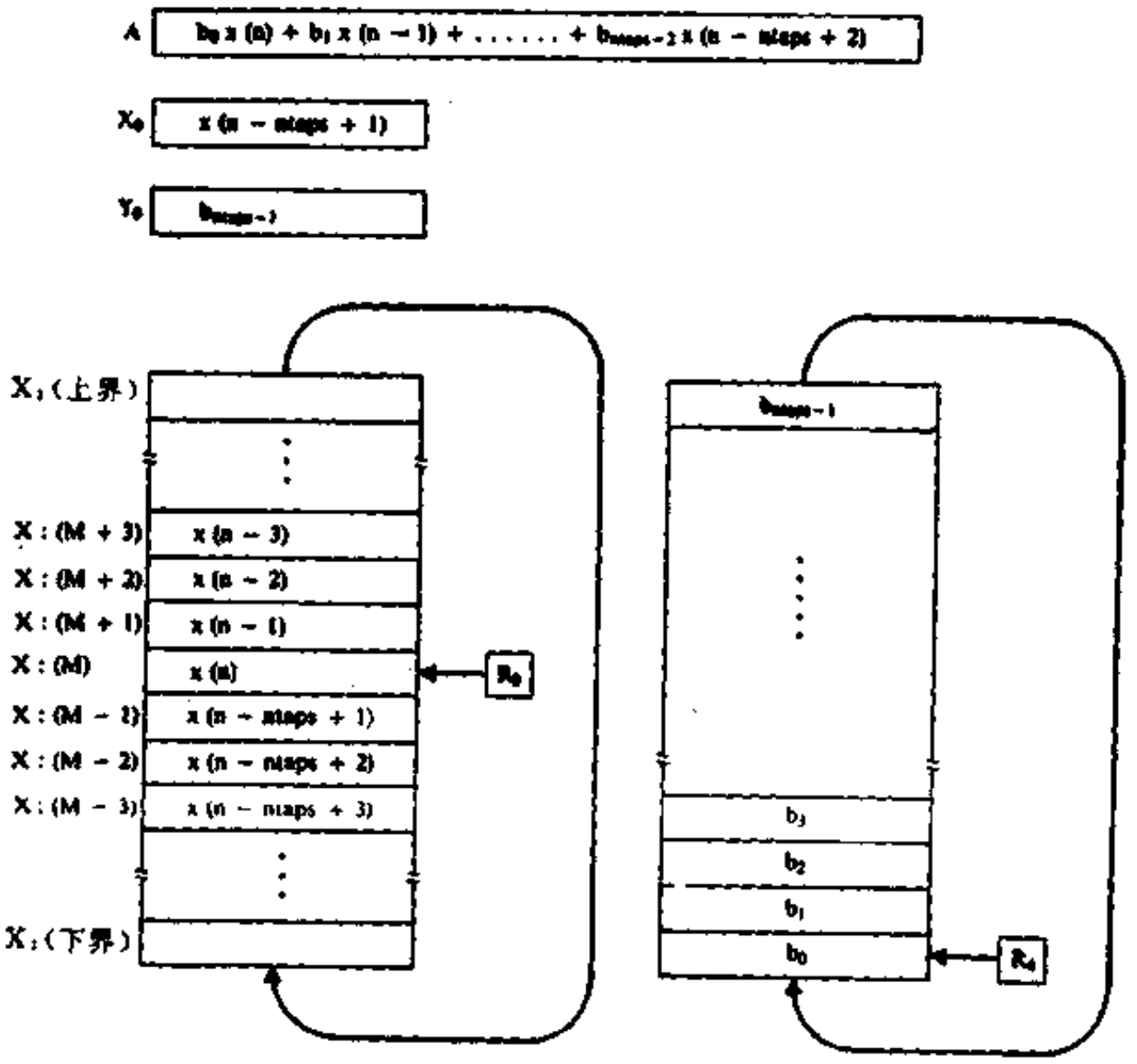


图 6－11　最后的 MAC 指令执行后的存贮器映像和数据寄存器

5. MACR 指令

该指令计算滤波器算法的最后分支，完成结果的舍入，同时地址寄存器减 1。在 MACR 指令结束时，A 累加器已包含滤波器输出样值 y(n)。

注意，当执行滤波器算法时，地址寄存器 R4 后递增总的 NTAPS 次数。因为 R4 的模是 NTAPS 以及 R4 是增加 RTAPS 次，R4 中地址值是循环的并指出系数缓存器的低界地址。

注意当执行滤波器算法时，地址寄存器 R0 后递增总的 NTAPS 次。也需注意在算法开始时，输入样值 x(n) 从数据 ALU 输入寄存器 X0 传送到 R0 指示的 X 存贮器。因为 R0 的模是 NTAPS，且 R0 是增加 NTAPS 次，所以 R0 中的地址值是循环的并指向状态变量缓存器的 X 存贮器单元 M。联系 MACR 指令，R0 是后递增 1。结果，R0 指向状态变量缓存器的 X 存贮器单元 M－1。执行下一次算法，则一个新的输入样值 x(n + 1) 将此值写入 X 存贮器 M－1 处。因此滤波器状态变量的模拟 FIFO 位移由调整 R0 地址指针而完成。MACR 指令执行后，各存贮器内容如图 6－12 所示。

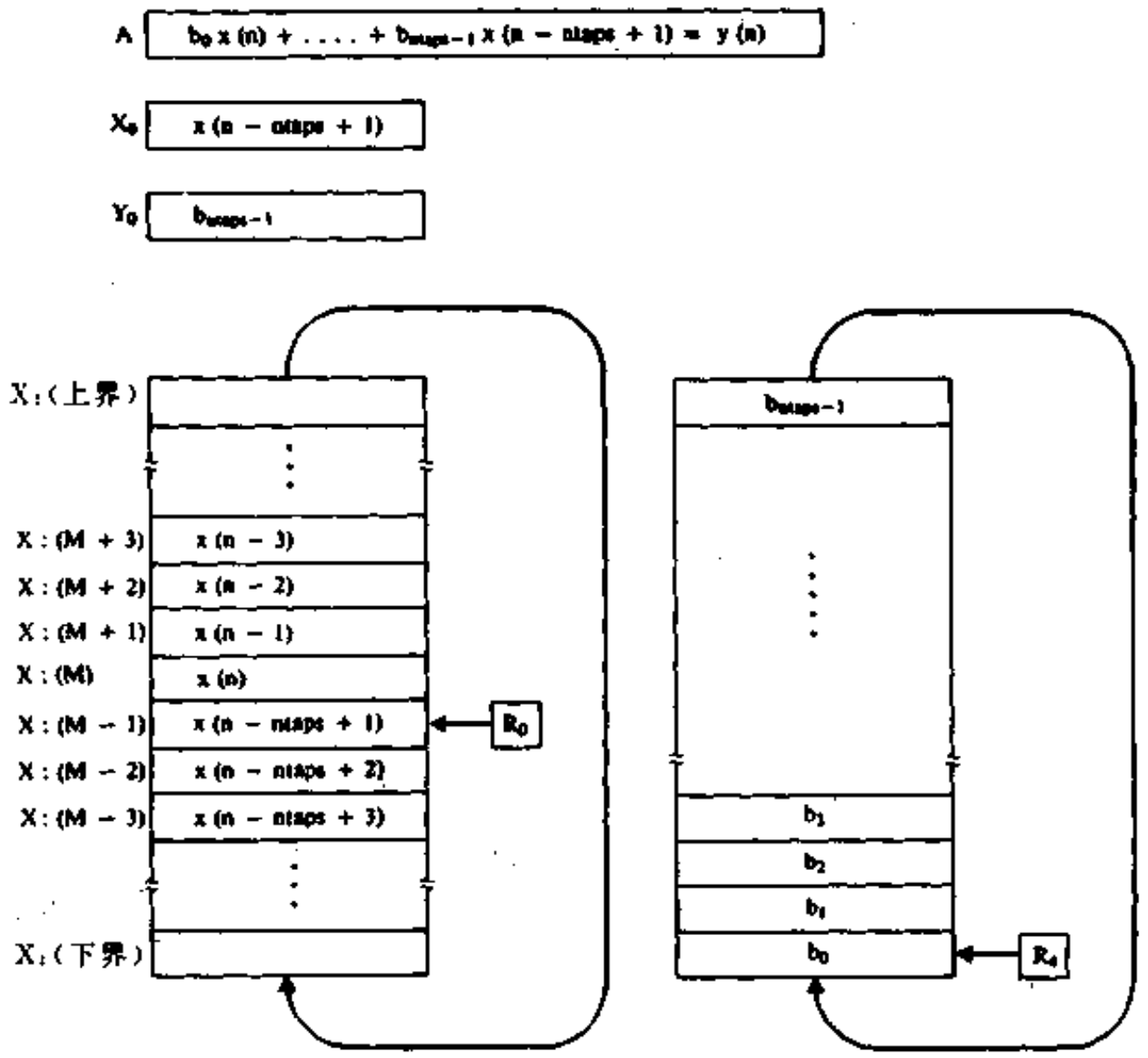


图 6－12　MACR 指令执行后的存贮器映像和数据寄存器

6.7　“滤波器设计和分析系统”软件程序简介

DSP56000/1 示范软件程序可用来设计滤波器和输出已设计的滤波器系数。Momentum 数据系统公司出版的滤波器设计和分析系统（FDAS）的示范版本包括在此书内，不仅可用于设计滤波器，而且还给出频域和时域响应图。FDAS 程序完全是菜单驱动的，很便于使用。

6.8　建立“滤波器设计和分析系统”

为建立滤波器设计和分析系统（FDAS）示范软件程序要求 800 k 字节的硬盘存贮量，为了操作要求 640 k 字节的 RAM。PC 机内虽然不要求有硬件算术协处理器设备，但有了它则可大大加速由 FDAS 软件程序完成的数学计算。

1. 加电和引导 PC 机。

2. 键入 md\dsptools\fdas<return>以产生 FDAS 程序目录。

3. 键入 cd\dsptools\fdas<return>，进入 FDAS 程序目录。

4. 把“FDAS 示范软盘＃1”插入驱动器“A”。

5. 键入 a：run_me c<return>，开始建立 FDAS 程序。

6. 执行屏幕上显示的 FDAS 建立指令。

7. 从主菜单中选择“EXIT TO DOS”以退出 FDAS 程序。

6.9 实现 FIR 滤波器

图 6-6 所示的 DSP56001 指令和图 5-8 所示的 DSP56001 指令一起形成图 6-13 所示的 FIR.ASM 汇 编程序指令。FDAS 程序可用于设计的各种滤波器如以下所述。所设计的滤波器系数和 FIR.ASM 程序宏指令可用于在 DSP 系统上实现 FIR 滤波器。

1. 去掉 PC 机电源。

2. 按第五章 5.2.2 节建立 DSP 系统。

3. 加电和引导 PC 机。

6.9.1 实现低通 FIR 滤波器

1. 键入 cd\dsptools\fdas<return>，进入 FDAS 程序目录。

2. 键入 demo<return>，装入和进入 FDAS 程序。

3. 根据 FDAS 程序菜单选择下面所列滤波器指标，并设计下述低通 FIR 滤波器。

4. 退出 FDAS 程序。

滤波器指标：

滤波器形式：	低通
设计方式：	FIR 设计-汉明窗
采样频率：	48kHz
通带截止频率：	8kHz
阻带截止频率：	10kHz
通带衰减	0.5dB
阻带衰减	50.0dB

设计结果：

抽头数： 75

理想脉冲系数：

H(1) =-.32922140E-02=H(75)	H(10) = .11368210E-01=H(66)
H(2) =-.88419413E-02=H(74)	H(11) = .45115526E-02=H(65)
H(3) =-.34803406E-02=H(73)	H(12) =-.86568877E-02=H(64)
H(4) = .66199729E-02=H(72)	H(13) =-.11763200E-01=H(63)
H(5) = .89115148E-02=H(71)	H(14) = .14624132E-16=H(62)
H(6) =-.14624132E-16=H(70)	H(15) = .12786086E-01=H(61)
H(7) =-.94864513E-02=H(69)	H(16) = .10230867E-01=H(60)
H(8) =-.75026360E-02=H(68)	H(17) =-.58005676E-02=H(59)
H(9) = .42004110E-02=H(67)	H(18) =-.15915494E-01=H(58)

```
fir  macro  states,coef,ntaps
;
;  states  = Location of filter states in X memory
;  coef    = Location of filter coefficients in Y memory
;  ntaps   = Number of taps
;
;*********************************************************
; Initialize routine
;*********************************************************
;
init2    reset
         movep  #0,x:$FFFB     ;set bcr to zero
         movep  #0,x:$FFEF     ;clear the transfer register
         movep  #0,x:$FFFF     ;stop all interrupts

;*********************************************************
; Set up the SSI for operation with the EVB.
; The following code sets port C to function as SSI
;*********************************************************

pcc   equ     $0001e0
      movep   #pcc,x:$FFE1  ;write PCC

;*********************************************************
; The following code sets the SSI CRA and CRB control
; registers for external continuous clock, synchronous,
; normal mode.
;*********************************************************

cra      equ     $004000
         movep   #cra,x:$FFEC  ;CRA pattern for word
                               ;length=16 bits

crb      equ     $003200
         movep   #crb,x:$FFED  ;CRB pattern for continuous
                               ;ck,synch,normal mode
                               ;word long frame sync

         move    #states,r0  ;point to filter states
         move    #ntaps-1,m0 ;mod(ntaps)
         move    #coef,r4    ;point to filter coefficients
         move    #ntaps-1,m4 ;mod(ntaps)

;*********************************************************
; Read A/D
;*********************************************************

; The following code polls the RDF flag in the SSI-SR and
; waits for RDF=1 and then reads the RX register to
; retrieve the data from the A/D converter.
; Sample rate is controlled by EVB.

wait2   nop
wait1   btst    #7,x:$ffee
         jcc     wait1         ;loop  until RDF bit = 1
         nop

         movep   x:$FFEF,x0    ;get A/D converter data

;*********************************************************
;  FIR Filter
;*********************************************************

         clr     a             x0,x:(r0)+      y:(r4)+,y0
         rep     #ntaps-1
         mac     x0,y0,a       x:(r0)+,x0      y:(r4)+,y0
         macr    x0,y0,a       (r0)-

;*********************************************************
; Write data to the D/A converter
;*********************************************************

      move    a,x:$FFEF     ;write D/A via SSI
      nop
      jmp     wait2         ;loop  indefinitely
      nop

      endm
```

图 6-13　FIR.ASM 汇编程序宏指令

H(19) =-.64111537E-02=H(57)
H(20) = .12504393E-01=H(56)
H(21) = .17298823E-01=H(55)
H(22) =-.14624132E-16=H(54)
H(23) =-.19605333E-01=H(53)
H(24) =-.16077077E-01=H(52)
H(25) = .93701477E-02=H(51)
H(26) = .26525824E-01=H(50)
H(27) = .11073811E-01=H(49)
H(28) =-.22507908E-01=H(48)
H(29) =-.32675554E-01=H(47)
H(30) = .14624132E-16=H(46)
H(31) = .42011427E-01=H(45)
H(32) = .37513180E-01=H(44)
H(33) =-.24362384E-01=H(43)
H(34) =-.79577472E-01=H(42)
H(35) =-.40603973E-01=H(41)
H(36) = .11253954E+00=H(40)
H(37) = .29407999E+00=H(39)
H(38) = .37500000E+00=H(38)

窗系数：

W(1) = .00000000E+00=W(75)
W(2) = .18012557E-02=W(74)
W(3) = .71920448E-02=W(73)
W(4) = .16133527E-01=W(72)
W(5) = .28561277E-01=W(71)
W(6) = .44385755E-01=W(70)
W(7) = .63492943E-01=W(69)
W(8) = .85745175E-01=W(68)
W(9) = .11098212E+00=W(67)
W(10) = .13902195E+00=W(66)
W(11) = .16966264E+00=W(65)
W(12) = .20268341E+00=W(64)
W(13) = .23784636E+00=W(63)
W(14) = .27489813E+00=W(62)
W(15) = .31357176E+00=W(61)
W(16) = .35358861E+00=W(60)
W(17) = .39466037E+00=W(59)
W(18) = .43649109E+00=W(58)
W(19) = .47877940E+00=W(57)
W(20) = .52122060E+00=W(56)
W(21) = .56350891E+00=W(55)
W(22) = .60533963E+00=W(54)
W(23) = .64641139E+00=W(53)
W(24) = .68642824E+00=W(52)
W(25) = .72510187E+00=W(51)
W(26) = .76215364E+00=W(50)
W(27) = .79731659E+00=W(49)
W(28) = .83033736E+00=W(48)
W(29) = .86097805E+00=W(47)
W(30) = .88901788E+00=W(46)
W(31) = .91425482E+00=W(45)
W(32) = .93650706E+00=W(44)
W(33) = .95561425E+00=W(43)
W(34) = .97143872E+00=W(42)
W(35) = .98386647E+00=W(41)
W(36) = .99280796E+00=W(40)
W(37) = .99819874E+00=W(39)
W(38) = .10000000E+01=W(38)

由窗系数修正后的滤波器脉冲系数：

H(1)*W(1) = .00000000E+00=H(75)*W(75)
H(2)*W(2) =- .15854836E-04=H(74)*W(74)
H(3)*W(3) =- .24914742E-04=H(73)*W(73)
H(4)*W(4) = .10669231E-03=H(72)*W(72)
H(5)*W(5) = .25451183E-03=H(71)*W(71)
H(6)*W(6) = .00000000E+00=H(70)*W(70)

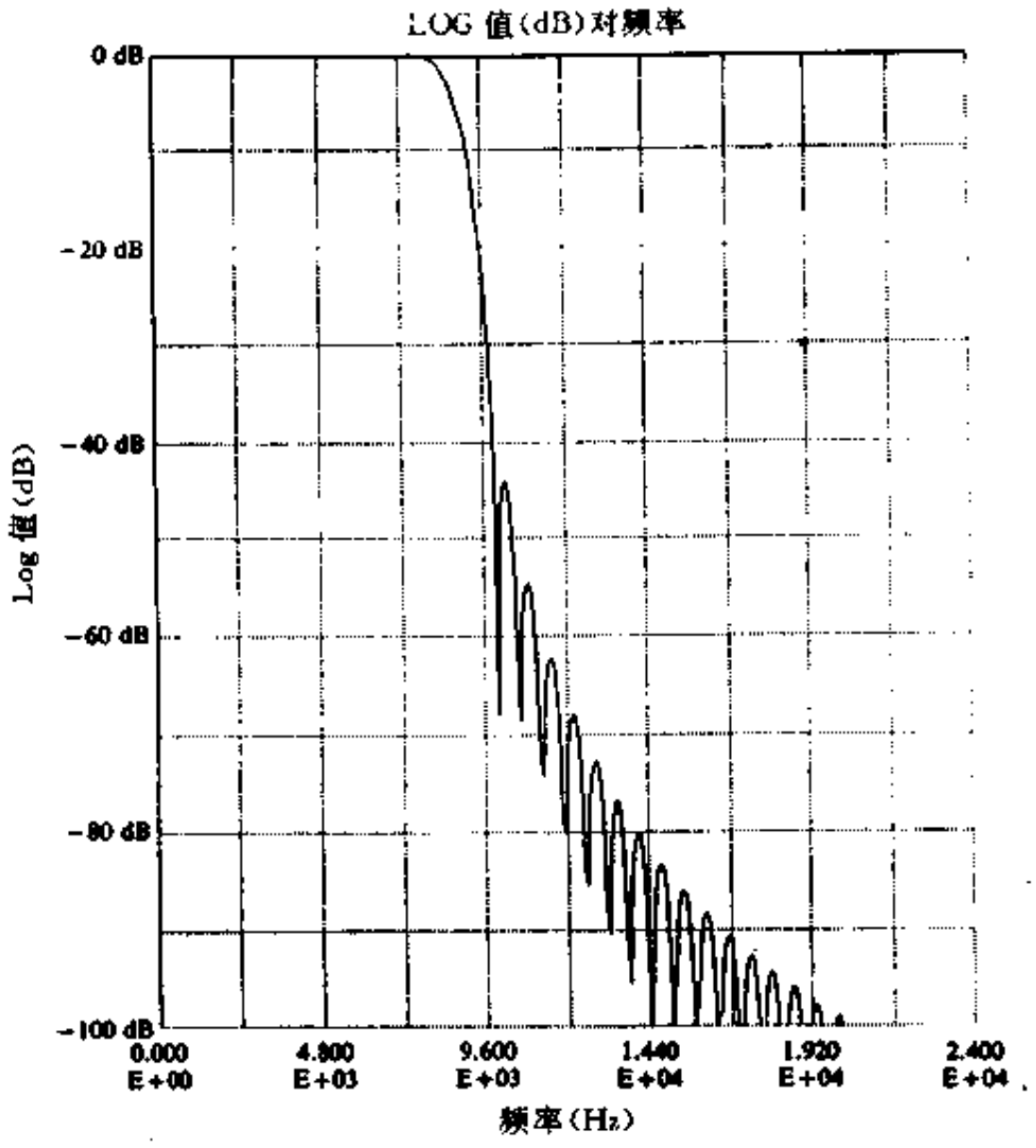
H(7) * W(7) =- .60224533E - 03=H(69) * W(69)
H(8) * W(8) =- .64325333E - 03=H(68) * W(68)
H(9) * W(9) = .46610832E - 03=H(67) * W(67)
H(10) * W(10)= .15803576E - 02=H(66) * W(66)
H(11) * W(11)= .76532364E - 03=H(65) * W(65)
H(12) * W(12)=- .17545223E - 02=H(64) * W(54)
H(13) * W(13)=- .27977228E - 02=H(63) * W(63)
H(14) * W(14)= .00000000E+00=H(62) * W(62)
H(15) * W(15)= .40092468E - 02=H(61) * W(61)
H(16) * W(16)= .36174059E - 02=H(60) * W(60)
H(17) * W(17)=- .22891760E - 02=H(59) * W(59)
H(18) * W(18)=- .69469213E - 02=H(58) * W(58)
H(19) * W(19)=- .30695200E - 02=H(57) * W(57)
H(20) * W(20)= .65175295E - 02=H(56) * W(56)
H(21) * W(21)= .97479820E - 02=H(55) * W(55)
H(22) * W(22)= .00000000E+00=H(54) * W(54)
H(23) * W(23)=- .12673020E - 01=H(53) * W(53)
H(24) * W(24)=- .11035681E - 01=H(52) * W(52)
H(25) * W(25)= .67942142E - 02=H(51) * W(51)
H(26) * W(26)= .20216703E - 01=H(50) * W(50)
H(27) * W(27)= .88292360E - 02=H(49) * W(49)
H(28) * W(28)=- .18689156E - 01=H(48) * W(48)
H(29) * W(29)=- .28132915E - 01=H(47) * W(47)
H(30) * W(30)= .00000000E+00=H(46) * W(46)
H(31) * W(31)= .38409114E - 01=H(45) * W(45)
H(32) * W(32)= .35131335E - 01=H(44) * W(44)
H(33) * W(33)=- .23280978E - 01=H(43) * W(43)
H(34) * W(34)=- .77304602E - 01=H(42) * W(42)
H(35) * W(35)=- .39948821E - 01=H(41) * W(41)
H(36) * W(36)= .11173010E+00=H(40) * W(40)
H(37) * W(37)= .29355025E+00=H(39) * W(39)
H(38) * W(37)= .375000000E+00=H(38) * W(38)

所设计的低通 FIR 滤波器的幅度和脉冲响应示于图 6 - 14。

在 DSP56001 处理器上实现所设计的低通 FIR 滤波器的“FIRLP. ASM” 汇编程序如图 6 - 15 所示。注意此程序有一“include”语句，它参考示于图 6 - 13 的 FIR. ASM 汇编程序宏指令。

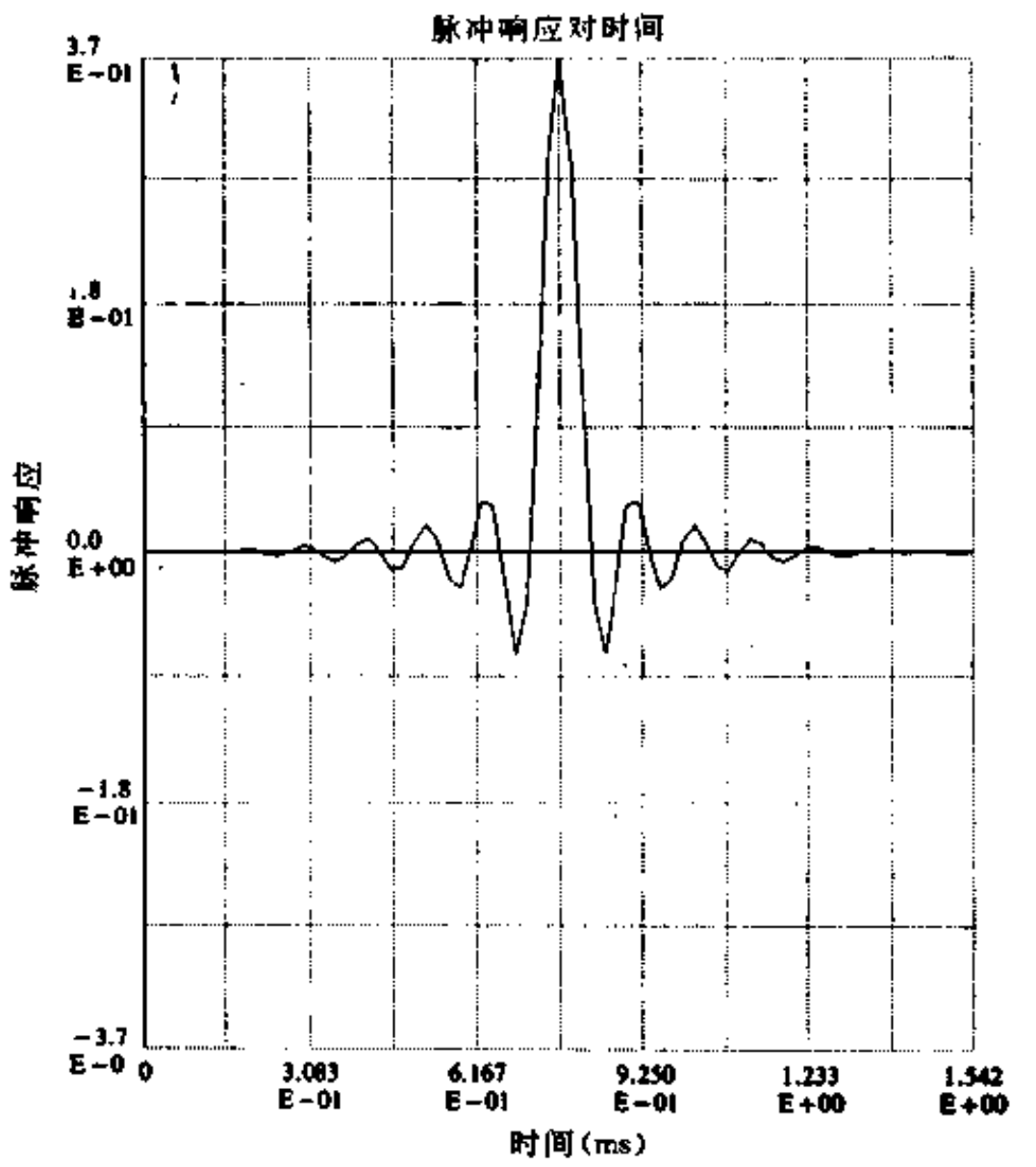
为了说明在 DSP 系统上所设计的 FIR 低通滤波器，应执行下述步骤：

1. 将函数产生器连接到 EVB 上的“BNC2”和双踪示波器的通道 1。
2. 使函数发生器产生一 2V(峰-峰)的正弦波。
3. 将示波器通道 2 连接到 EVB 的“BNC1”。



(a)

图 6-14　低通滤波器幅度响应



(b)

图 6-14　低通滤波器脉冲响应

```
;*****************************************
;
; File: FIRLP.ASM
;
;*****************************************
;
        include 'fir'
;
;*****************************************
; Coefficients of Low-Pass FIR
;*****************************************
;
;
ntaps   equ     75
;
        org     x:$0
states  dsm     ntaps           ;filter states
;
        org     y:$0
coef
        dc          0
        dc       -133
        dc       -209
        dc        895
        dc       2135
        dc          0
        dc      -5052
        dc      -5396
        dc       3910
        dc      13257
        dc       6420
        dc     -14718
        dc     -23469
        dc          0
        dc      33632
        dc      30345
        dc     -19203
        dc     -58275
        dc     -25749
        dc      54673
        dc      81772
        dc          0
        dc    -106309
        dc     -92574
        dc      56994
        dc     169590
        dc      74065
        dc    -156776
        dc    -235996
        dc          0
        dc     322199
        dc     294703
        dc    -195295
        dc    -648478
        dc    -335115
        dc     937260
        dc    2462478
        dc    3145728
        dc    2462478
        dc     937260
        dc    -335115
        dc    -648478
        dc    -195295
        dc     294703
        dc     322199
        dc          0
        dc    -235996
        dc    -156776
        dc      74065
        dc     169590
        dc      56994
        dc     -92574
        dc    -106309
        dc          0
        dc      81772
        dc      54673
        dc     -25749
        dc     -58275
        dc     -19203
        dc      30345
        dc      33632
        dc          0
        dc     -23469
        dc     -14718
        dc       6420
        dc      13257
        dc       3910
        dc      -5396
        dc      -5052
        dc          0
        dc       2135
        dc        895
        dc       -209
        dc       -133
        dc          0
;
; Program start address
;
        org     p:$100
;
        fir     states,coef,ntaps
;
        end
```

图 6-15　FIRLP.ASM 汇编程序

4. 装入“ADS56000 用户接口软件”程序。
5. 从“DSP56001 示范软件＃2”软盘装入文件 firlp.lod。
6. 键入 go<return>执行在 DSP56001 处理器上所设计的低通 FIR 滤波器程序。
7. 将正弦波输入频率从 0.5kHz 改变到 20kHz。在示波器上观察模拟输入和输出信号。注意此 FIR 滤波器通过的频率为 0.5～8kHz，而阻止频率为 10～20kHz。当输入信号频率从 8～10kHz 增加时，输出信号衰减随之增加。
8. 键入 force b<return>停止执行 FIR 滤波器算法，并将 ADM 的控制返回到监控程序。
9. 键入 quit<return>，退出 ADS56000 程序。

6.9.2 实现带通滤波器

1. 键入 cd\dsptool\fdas<return>，进入 FDAS 程序目录。
2. 键入 demo<return>，装入并进入 FDAS 程序。
3. 根据 FDAS 程序菜单选择以下所列滤波器指标，并设计带通滤波器。
4. 退出 FDAS 程序。

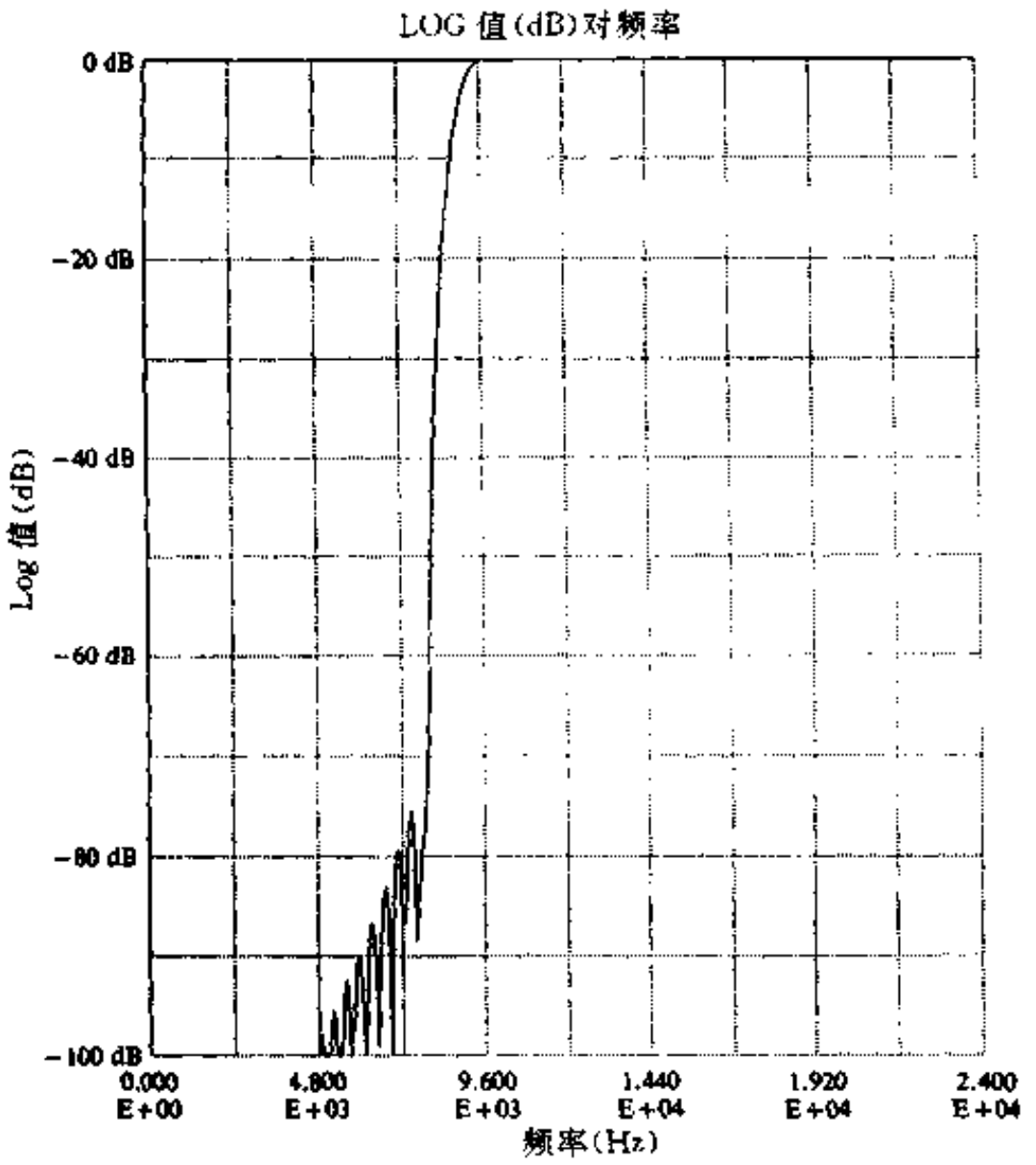
滤波器指标：

滤波器形式：	带通
设计方法：	FIR 设计- Blackman 窗
采样频率：	48kHz
通带截止频率：	6 和 12kHz
阻带截止频率：	4 和 14kHz
通带衰减：	0.5dB
阻带衰减：	50.0dB

设计结果：抽头数 133。

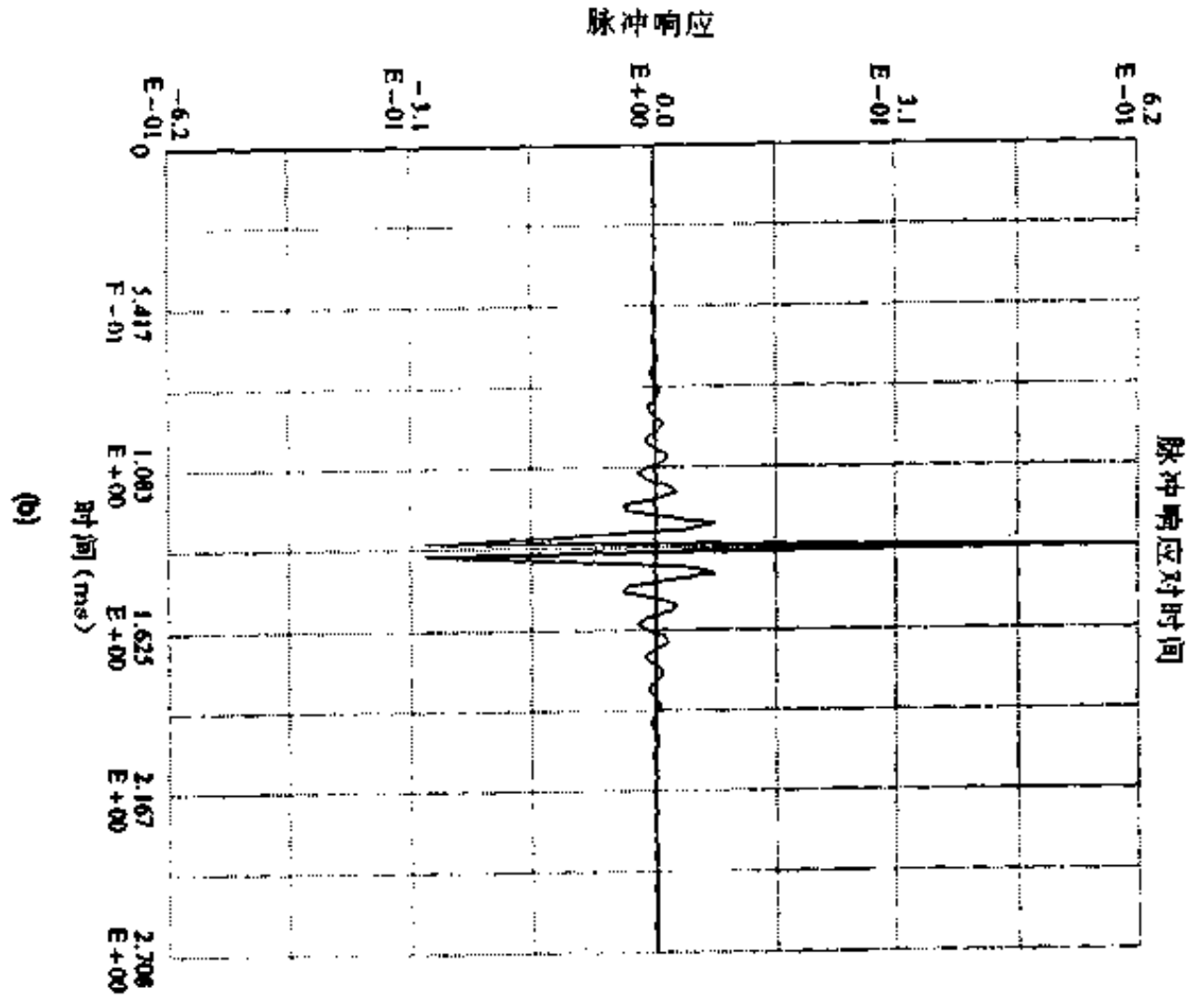
分别求得理想脉冲响应系数 H(1)～H(133)，窗系数 W(1)～W(133)，以及修正后的脉冲系数 H(1)＊W(1)～H(133)＊W(133)（略）

设计所得的带通滤波器幅度响应和脉冲响应如图 6－16 所示。



(a)

图 6-16(a)　带通滤波器幅度响应



(b)

图 6-16(b)　带通滤波器脉冲响应

第七章　在 DSP56001 处理器上设计和实现 IIR 滤波器

本章讨论在 DSP56001 处理器上设计和实现无限脉冲响应（IIR）数字滤波器。IIR 数字滤波器设计包含以 z 为变量的传输函数的确定，以满足一定的频域指标。过去模拟滤波器技术也是基于频域指标。为此，IIR 数字 滤波器通常用模拟滤波器的各种变换来设计。这种方法对设计标准的低通、高通、带通和带阻滤波器最有用，对此有大量模拟滤波知识是有用的。在模拟滤波中，有若干近似可满足设计要求。最常用的近似是 Butterworth 和 Chebyshev 近似。

数字滤波器设计从设计适当的低通模拟滤波器开始。通常用的两种不同方法都是由低通模拟滤波器得到数字滤波器。当用第一种设计方法时，用模拟频率变换得到另一个所要求的模拟滤波器。可用脉冲不变变换或双线性变换从所要求的模拟滤波器确定数字滤波器。当用第二种设计方法时，用脉冲不变变换或双线性变换使低通模拟滤波器转换为低通数字滤波器。最后用数字频率变换得到数字滤波器。最后须检验滤波器的稳定性。本章研究第一种设计方法。

脉冲不变变换指定数字滤波器的脉冲响应为连续时间响应的采样样本，以保持模拟滤波器的脉冲响应。作为结果，数字滤波器的频率响应是模拟滤波器频率响应的混叠。双线性变换消除了混叠问题，因为模拟传输函数转换到数字传输函数，用可比较的频率响应特性和从连续时间 S 平面到离散时间 Z 平面提供极点和零点一对一的映射。因此，双线性变换是设计 IIR 滤波器较好的选择。

在设计过程中，首先用所要求的 IIR 滤波器频域指标设计归一化低通模拟 Butterworth 和 Chebyshev 滤波器。然后用模拟频率变换从归一化低通模拟 Butterworth 和 Chebyshev 滤波器得到所要求的模拟滤波器。再用双线性变换从模拟滤波器得到数字滤波器。

由于假设滤波器阶为偶整数，所以设计过程中最后一步是把 Butterworth 和 Chebyshev 滤波器分解为二阶节的乘积。图 7-1 是本章中用于设

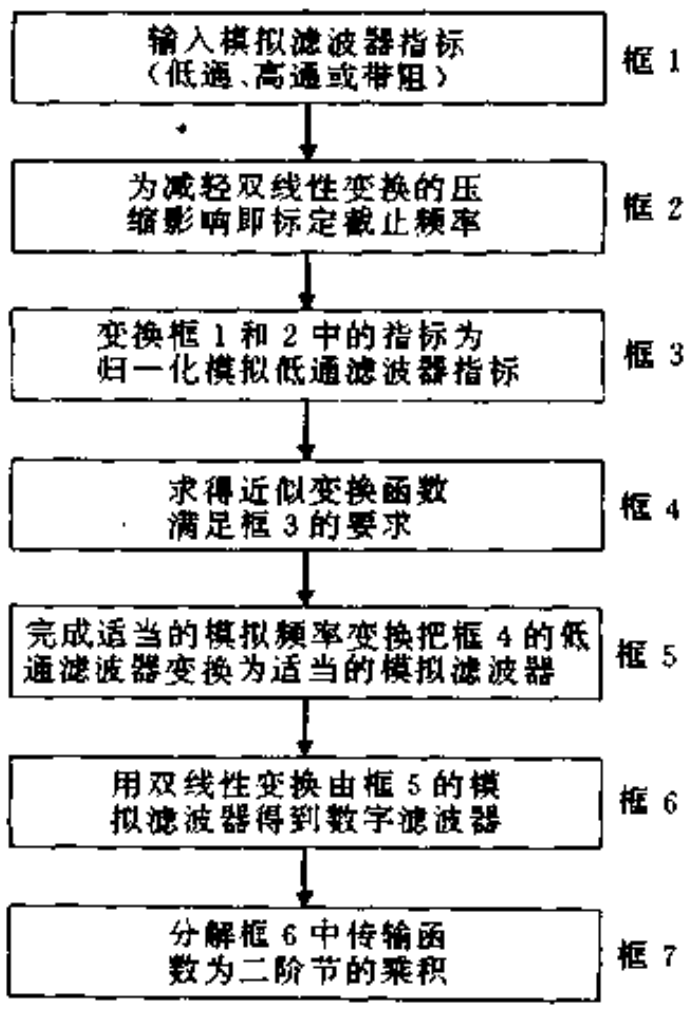


图 7-1　设计过程

图 7-2　DSP 系统

计数字滤波器的过程。本章使用 Momentum Data System 公司的专用 FDAS 软件程序来说明这一设计过程，并介绍了实现二阶节的 DSP56001 指令。第 5 章中所研究的在 DSP 系统上设计和实现各种 IIR 数字滤波器的过程重新示于图 7－2。

7.1 IIR 滤波器传输函数

IIR 滤波器的输入序列 x(n) 和输出序列 y(n) 间的关系可写为以下差分方程：

$$y(n) = \sum_{i=0}^{N} b_i x(n-i) + \sum_{j=1}^{N} a_j y(n-j) \qquad (7-1)$$

其中 b_i 和 a_j 是滤器的系数，且至少有一个 a_i 系数为非零。滤波系 b_i 和 a_j 提供滤波特性，由此使输入序列通过后产生所要求的输出序列 y(n)。由定义和设计可知，IIR 滤波器（也称递归滤波器）的输出序列 y(n) 是过去输出以及当前和过去输入的函数。

与(7－1)式相应的传输函数可表示为：

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 - a_1 z^{-1} - \cdots - a_n z^{-N}} \qquad (7-2)$$

本章的后面，N 都设为偶整数，(7－2)式可分解为二阶节的乘积：

$$H(z) = K \prod_{i=1}^{M} H_i(z) \qquad (7-3)$$

其中

$$H_i(z) = \frac{Y_{i+1}(z)}{Y_i(z)} = \frac{1 + b_{1i} z^{-1} + b_{2i} z^{-2}}{1 - a_{1i} z^{-1} - a_{2i} z^{-2}} \qquad (7-4)$$

$$Y_1(z) = KX(z) \qquad (7-5)$$

$$Y_{M+1}(z) = Y(z) \qquad (7-6)$$

M 等于 0.5N，K 是正增益，第 i 个二阶节输出 $y_{i+1}(n)$ 是第 i＋1 二阶节的输入。图 7－3 是传输函数 H(z) 按二阶节级联而成。为了保证 IIR 数字滤波器的稳定，a_{1i} 和 a_{2i} 必须分别小于 2.0 和 1.0，$H_i(z)$ 可改写成

$$H_i(z) = \frac{Y_{i+1}(z)}{Y_i(z)} = \frac{(1 + b_{1i} z^{-1} + b_{2i} z^{-1}) X_i(z)}{(1 - a_{1i} z^{-1} - a_{2i} z^{-1}) X_i(z)} \qquad (7-7)$$

其中

$$Y_i(z) = (1 - a_{1i} z^{-1} - a_{2i} z^{-2}) X_i(z) \qquad (7-8)$$

$$Y_{i+1}(z) = (1 + b_{1i} z^{-1} + b_{2i} z^{-2}) X_i(z) \qquad (7-9)$$

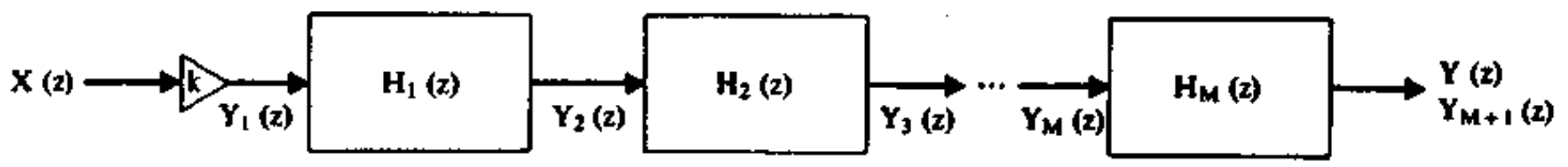


图 7－3　传输函数按二阶节级联而成

(7－8)和(7－9)式的结构示于图 7－4，6 阶数字 IIR 滤波器按三个二阶节级联而成的结构示于图 7－5。

本章的主要目的是设计(7－3)和(7－4)式传输函数，以满足一定的频域指标，然后用(7－4)式的滤波器系数在DSP56001处理器上实现所要求的数字滤波器。

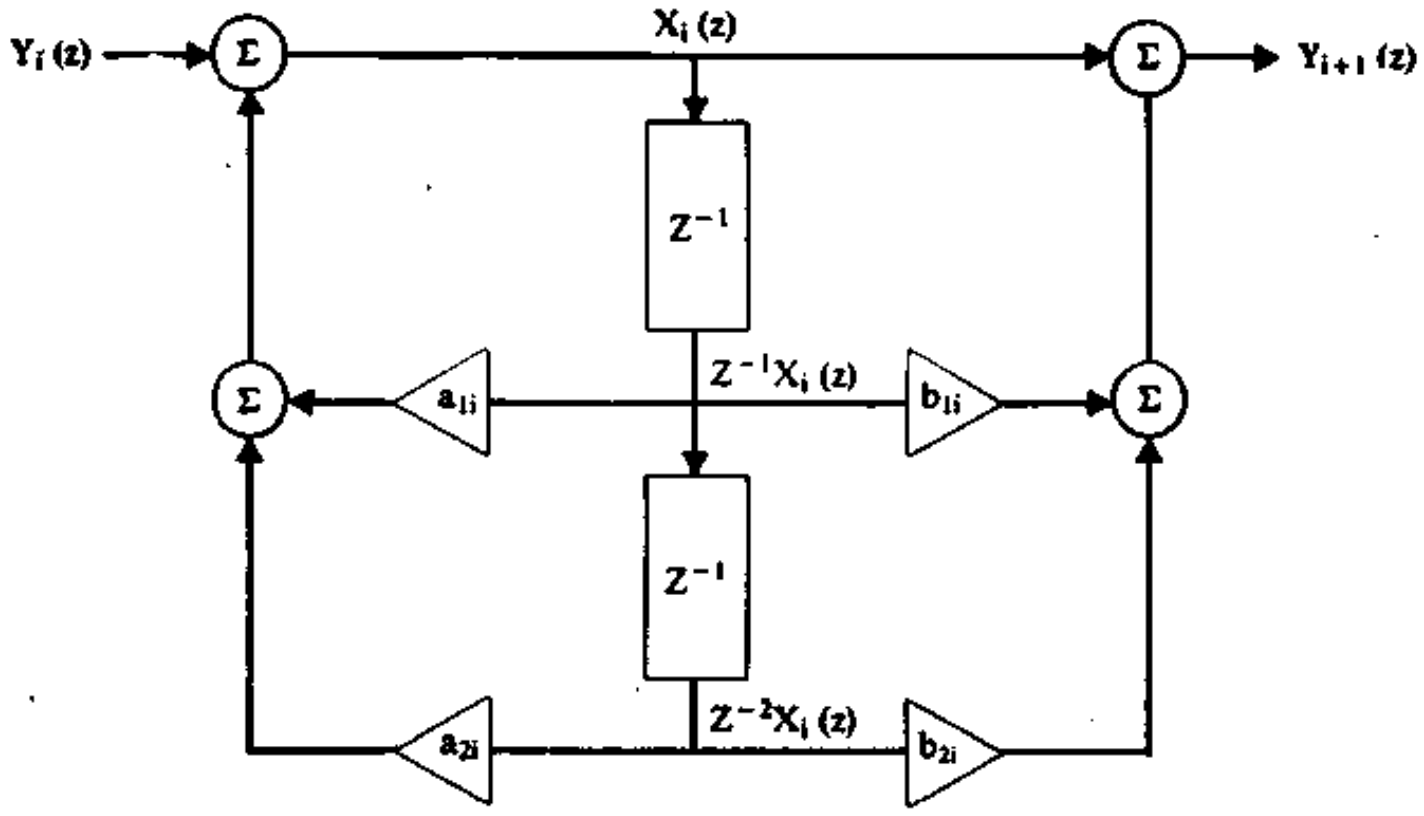


图 7－4　二阶节结构

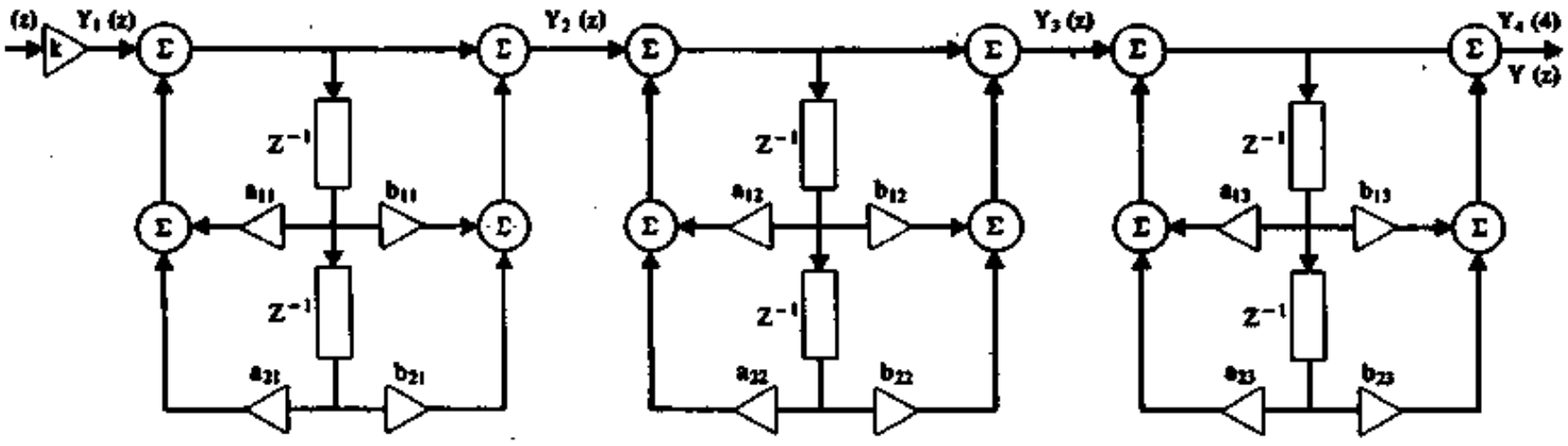


图 7－5　6阶数字滤波器按三组二阶节级联

7.2　双线性变换概述

双线性变换以下式定义：

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{7-10}$$

其中T是采样周期。为了考虑频率响应，变量s在虚轴上计算，z变量在单位圆上计算：

$$s = j\Omega \tag{7-11}$$

$$z = e^{j\omega T} \tag{7-12}$$

其中Ω是模拟频率，ω是数字 频率。二者的关系由(7－8)、(7－9)和(7－10)式得到：

$$\Omega = \frac{2}{jT}\frac{e^{j\omega T}-1}{e^{j\omega T}+1} = \frac{2}{jT}\frac{e^{j\omega T/2}(e^{j\omega T/2}-e^{j\omega T/2})}{e^{j\omega T/2}(e^{j\omega T/2}+e^{j\omega T/2})}$$

$$= \frac{2}{jT}\frac{2j\sin \omega T/2}{2\cos \omega T/2} = \frac{2}{T}\tan \omega T/2$$

(7-13)

模拟频率 Ω 和数字频率 ω 之间的关系示于图 7-6。可以看到 s 平面($-\infty \leqslant \Omega \leqslant +\infty$)中无限虚抽映射为 z 平面($-\pi \leqslant \omega T \leqslant +\pi$)中的单位圆。而且，频率 Ω 和 ω 有非线性关系，在频率的响应特性上有压缩效应。已知此压缩效应是频率缠绕。此压缩使传输函数在高 Ω 频率上变换到 w 域时有很大的失真。在 Ω 频率转换到 Ω^* 频率时引入适当的预缠绕可减少这种压缩，即：

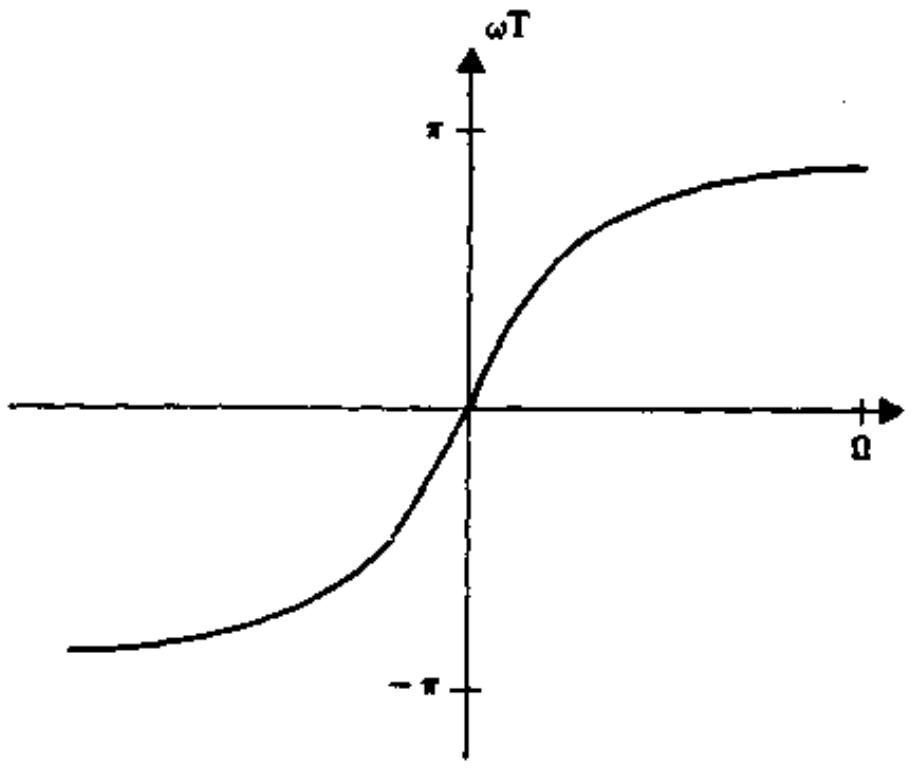


图 7-6　模拟和数字频率间的关系

$$\Omega^* = \frac{2}{T}\tan \Omega T/2 \qquad (7-14)$$

以下各节中，模拟滤波器截止频率都先转换为 Ω^* 频率，然后用于设计模拟滤波器。

7.3　IIR 滤波器指标

标准 IIR 滤波器的频域指标是通带和阻带衰减，以及通带和阻带截止频率。图 7-7 表示低通、高通、带通和带阻滤波器的频域指标，其中

α = 通带衰减(dB)。

β = 阻带衰减(dB)。

Ω_c = 3dB 截止频率。

Ω_1 = 低通滤波器通带截止频率。

Ω_2 = 低通滤波器阻带截止频率。

Ω_3 = 高通滤波器阻带截止频率。

Ω_4 = 高通滤波器通带截止频率

Ω_5 = 带通滤波器阻带低截止频率。

Ω_6 = 带通滤波器通带低截止频率。

Ω_7 = 带通滤波器通带高截止频率。

Ω_8 = 带通滤波器阻带高截止频率。

Ω_9 = 带阻滤波器通带低截止频率。

Ω_{10} = 带阻滤波器阻带低截止频率。

Ω_{11} = 带阻滤波器阻带高截止频率。

Ω_{12} = 带阻滤波器通带高截止频率。

设计过程由所要求的模拟滤波器截止频率的预缠绕开始，得到 Ω^* 标量上相应的截止频率。然后设计模拟滤波器传输函数满足 Ω^* 的指标。用双线性变换，从模拟传输函数得到的数字滤度器传输函数满足 Ω 上的指标。这可由(7-13)和(7-14)式得到 ω 看出：

$$\omega = \frac{2}{T}\text{arc tan } \Omega^* T/2 = \Omega \qquad (7-15)$$

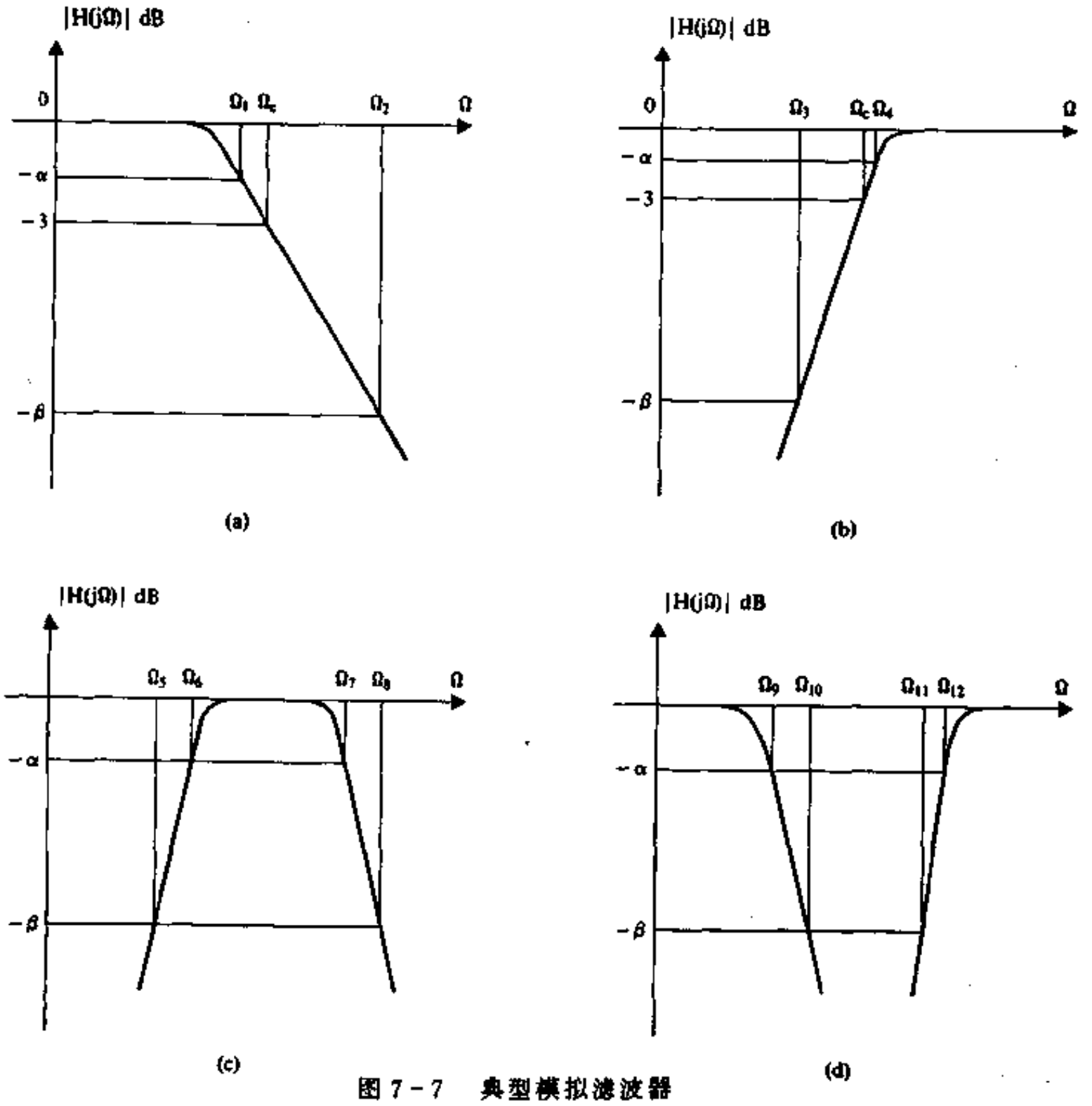


图 7-7　典型模拟滤波器

(a) 低通，(b) 高通，(c) 带通，(d) 带阻

7.4　归一化低通滤波器设计

本节中，用由表 7-1 所选模拟频率变换将 Ω^* 上的模拟滤波器截止频率转换为归一化模拟低通滤波器 $H_n(s)$ 的相应截止频率。$H_n(s)$ 是归一化理想低通滤波器的近似，它在由 0 到 1rad/sec 的频带内增益为 1，对所有高于 1rad/sec 的频率增益为 0。

对于给定的归一化低通模拟滤波器指标，有很多可设计的滤波器型式。最常用的是 Butterworth 和 Chebyshev 滤波器。对于 $w \geqslant 0$，Butterworth 低通滤波器振幅特性是 w 的单调降函数，而 Chebyshev 低通滤波器在通带内是波纹函数，在阻带内是单调降函数。以上二种滤波器示于图 7-8，其中 $\Omega_{\text{通}}^n$ 和 $\Omega_{\text{阻}}^n$ 分别是归一化通带和阻带截止频率。

表 7-1　模拟频率变换

要求的滤波器	模拟频率变换
低通	$s' = s/\Omega^+$
高通	$s' = \Omega^+ /s$
带通	$s' = [(s^2 + \Omega_7^* \Omega_6^*)/(s(\Omega_7^* - \Omega_6^*))]$
带阻	$s' = [s(\Omega_{11}^* - \Omega_{10}^*)\Omega_{\text{阻}}/(s^2 + \Omega_{11}^* \Omega_{10}^*)]$
	$s = j\Omega^n$
	$s' = j\Omega^*$

要求的滤波器	模拟频率变换
	$\Omega^+ = \Omega_c^*$ 对 Butterworth $= \Omega_L^*$ 对 Chebyshev Ω^n = 归一化模拟低通频率

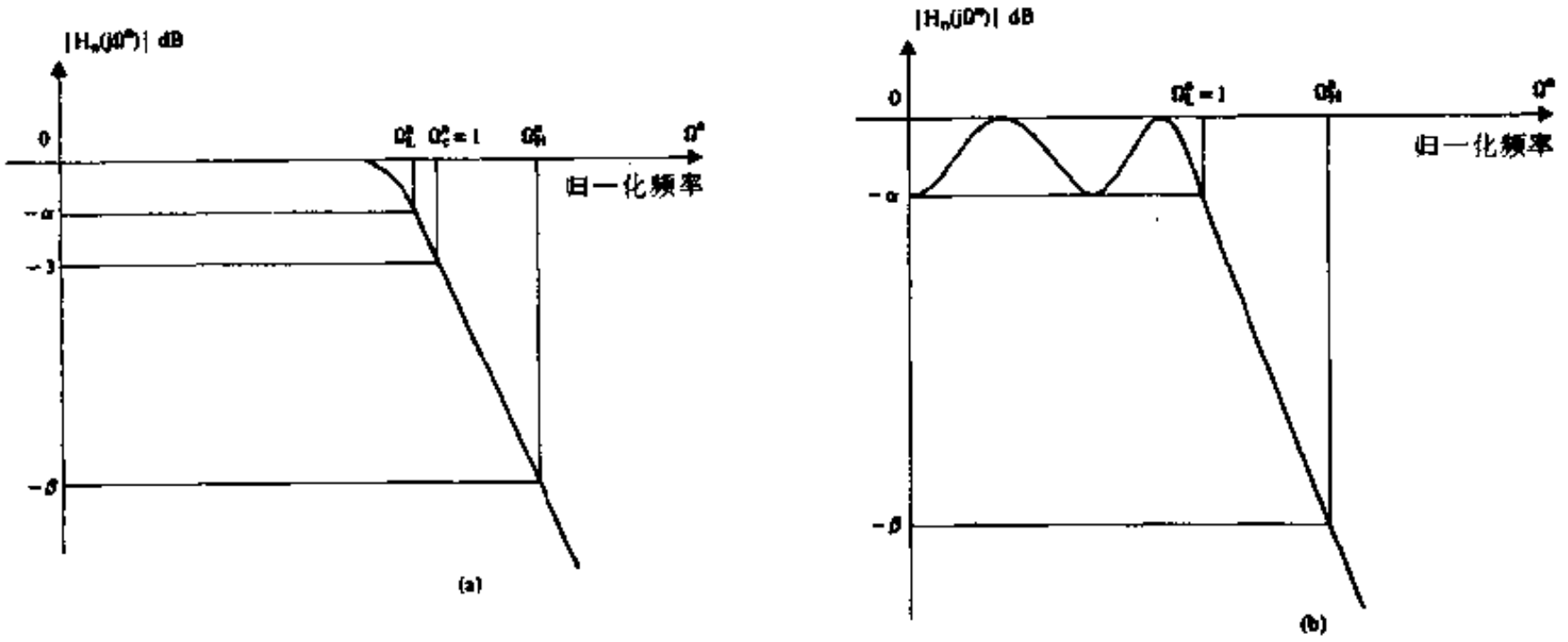


图 7-8 归一化模拟低通滤波器

(a)Butterworth 滤波器,(b)Chebyshev 滤波器

归一化通带截止频率对 Chebyshev 滤波器等于 1,对 Butterworth 滤波器等于通带截止频率被 3dB 频率除。若实际 3dB 截止频率未给出,则需做一假设以计算 Ω_L^n。

Ω_R^n 与 Ω_L^n 之比称作 R,可用模拟频率变换和低通、高通、带通或带阻截止频率进行计算如下:

(1) 低通对归一化低通

由表 7-1,低通对归一化低通变换为:

$$s' = s/\Omega^+ \text{ 或 } \Omega^n = \Omega^* /\Omega^+ \tag{7-16}$$

因此,R 可如下计算:

$$R = \Omega_R^n/\Omega_L^n = (\Omega_2^* /\Omega^+)/(\Omega_1^* /\Omega^+) = \Omega_2^* /\Omega_1^* \tag{7-17}$$

(2) 高通对归一化低通

由表 7-1,高通对归一化低通变换为:

$$s' = \Omega^+ /s \text{ 或 } \Omega^n = \Omega^+ /\Omega^* \tag{7-18}$$

用(7-18)式,R 可如下计算:

$$R = \Omega_R^n/\Omega_L^n = (\Omega^+ /\Omega_3^*)/(\Omega^+ /\Omega_4^*) = \Omega_4^* /\Omega_3^* \tag{7-19}$$

(3) 带通对归一化低通

由表 7-1,带通对归一化低通变换为:

$$s' = \frac{s^2 + \Omega_7^* \Omega_5^*}{s(\Omega_7^* \Omega_6^*)}$$

或

$$\Omega^n = \frac{\Omega^{*2} - \Omega_7^* \Omega_6^*}{\Omega^* (\Omega_7^* - \Omega_6^*)} \tag{7-20}$$

此变换将使 Ω_5^* 和 Ω_7^* 变为 $\Omega_L^n = \pm 1$,以及 Ω_5^* 和 Ω_8^* 变为:

$$\Omega_{H1}^{n}=\frac{\Omega_5^{*2}-\Omega_7^{*}\Omega_6^{*}}{\Omega_5^{*}(\Omega_7^{*}-\Omega_6^{*})} \tag{7-21}$$

$$\Omega_{H2}^{n}=\frac{\Omega_8^{*2}-\Omega_7^{*}\Omega_6^{*}}{\Omega_8^{*}(\Omega_7^{*}-\Omega_6^{*})} \tag{7-22}$$

为了使阻带衰减至少为 β dB，必须选择阻带截止频率 Ω_H^n 为 Ω_{H1}^n 和 Ω_{H2}^n 中较小的。因此，R 可计算如下：

$$R=\min\left[\left|\frac{\Omega_5^{*2}-\Omega_7^{*}\Omega_6^{*}}{\Omega_5^{*}(\Omega_7^{*}-\Omega_6^{*})}\right|,\left|\frac{\Omega_8^{*2}-\Omega_7^{*}\Omega_6^{*}}{\Omega_8^{*}(\Omega_7^{*}-\Omega_6^{*})}\right|\right] \tag{7-23}$$

(4) 带阻对归一化低通

带阻对归一化低通变换是：

$$s'=\frac{s(\Omega_{11}^{*}-\Omega_{10}^{*})\Omega_H^{*}}{s^2+\Omega_{11}^{*}\Omega_{10}^{*}}$$

或

$$\Omega^{n}=\frac{\Omega^{*}(\Omega_{11}^{*}-\Omega_{10}^{*})\Omega_H^{*}}{\Omega_{11}^{*}\Omega_{10}^{*}-\Omega^{*2}} \tag{7-24}$$

此变换将使 Ω_{11}^{*} 和 Ω_{10}^{*} 变为 $\Omega_L^n=\pm\Omega_H^n$，以及 Ω_9^{*} 和 Ω_{12}^{*} 为：

$$\Omega_{L1}^{n}=\frac{\Omega_9^{*}(\Omega_{11}^{*}-\Omega_{10}^{*})\Omega_H^{n}}{\Omega_{11}^{*}\Omega_{10}^{*}-\Omega_9^{*2}} \tag{7-25}$$

$$\Omega_{L2}^{n}=\frac{\Omega_{12}^{*}(\Omega_{11}^{*}-\Omega_{10}^{*})\Omega_H^{n}}{\Omega_{11}^{*}\Omega_{10}^{*}-\Omega_{12}^{*2}} \tag{7-26}$$

为了使通带衰减至少为 α dB，必须选择通带截止频率 Ω_L^n 大于 Ω_{L1}^n 和 Ω_{L2}^n 的绝对值。R 可计算如下：

$$R=\min\left[\left|\frac{\Omega_{11}^{*}\Omega_{10}^{*}-\Omega_9^{*2}}{\Omega_9^{*}(\Omega_{11}^{*}-\Omega_{10}^{*})}\right|,\left|\frac{\Omega_{11}^{*}\Omega_{10}^{*}-\Omega_{12}^{*2}}{\Omega_{12}^{*}(\Omega_{11}^{*}-\Omega_{10}^{*})}\right|\right] \tag{7-27}$$

表 7-2 表示 R 的值，用低通、高通、带通和带阻的截止频率计算。计算 R 之后，归一化阻带截止频率 Ω_H^n 可由 Ω_L^n 乘 R 得到。

表 7-2　阻带与通带截止频率之比

要求的滤波器	归一化低通滤波器的 $R=\Omega_H/\Omega_L$
低通	Ω_2'/Ω_1'
高通	Ω_4'/Ω_3'
带通	$\min\left[\left\|\frac{\Omega_3'^2-\Omega_4'\Omega_1'}{\Omega_3'(\Omega_1'-\Omega_4')}\right\|,\left\|\frac{\Omega_4'^2-\Omega_4'\Omega_1'}{\Omega_4'(\Omega_1'-\Omega_4')}\right\|\right]$
带阻	$\min\left[\left\|\frac{\Omega_{11}^{*}\Omega_{10}^{*}-\Omega_9^{*2}}{\Omega_9^{*}(\Omega_{11}^{*}-\Omega_{10}^{*})}\right\|,\left\|\frac{\Omega_{11}^{*}\Omega_{10}^{*}-\Omega_{12}^{*2}}{\Omega_{12}^{*}(\Omega_{11}^{*}-\Omega_{10}^{*})}\right\|\right]$

7.4.1　Butterworth 滤波器设计

N 阶归一化低通模拟 Butterworth 滤波器的振幅响应可写为：

$$|H_n(j\Omega^n)|^2=\frac{1}{1+(\Omega^n)^{2n}} \tag{7-28}$$

设计的第一步是求滤波器阶数 N，满足归一化低通滤波器指标(α,β,Ω_L^n 和 Ω_H^n)。这可以选择最小偶整数以满足下式来完成：

$$N\geqslant\max(N1,N2) \tag{7-29}$$

其中 N1 和 N2 满足以下不等式：$-10\log\frac{1}{1+(\Omega_L^n)_1^{2N}}<\alpha$ (7-30)

$$-10\log\frac{1}{1+(\Omega_H^n)_2^{2N}}<\beta \tag{7-31}$$

下一步是求归一化传输函数 $H_n(s)$，把 $s'=j\Omega^n$ 代入(7-28)式得到：

$$|H_n(s')|^2=\frac{1}{1+(-1)^N s'^{2N}} \tag{7-32}$$

分母有如下 2N 个根：

$$s'_i=\exp j\pi(2i+N-1)/2N \qquad 1\leqslant i\leqslant 2N \tag{7-33}$$

2N 个根位于 s′ 平面的单位圆上，如图 7-9 所示。为了形成稳定的 N 阶归一化低通滤波器，用 s′ 左半平面中的极点，相应的传输函数是：

$$H_n(s')=\frac{1}{(s'-s'_1)(s'-s'_2)\cdots(s'-s'_N)} \tag{7-34}$$

对偶数 N，所有极点为复共轭形式 s'_i, s'_{N+1-i}，其中 $1\leqslant+i\leqslant N$。用(7-33)式的极点定义，归一化滤波器传输函数为：

$$H_n(s')=\prod_{i=1}^{m=0.5N}H_i(s') \tag{7-35}$$

其中

$$H_i(s')=\frac{1}{(s'-s'_i)(s'-s'_{N+1-i})}=\frac{1}{s'^2-2s'\cos((2i+N-1)\pi/2N)+1} \tag{7-36}$$

7.4.2 Chebyshev 滤波器设计

N 阶 Chebyshev 滤波器振幅响应为：

$$|H_n(j\Omega^n)|^2=\frac{1}{1+\epsilon^2T_N^2(\Omega^n)} \tag{7-37}$$

其中 $T_N(\Omega^n)$ 是 N 阶 Chebyshev 多项式：

$$\begin{aligned}T_N(\Omega^n)&=\cos(N\cos^{-1}\Omega^n) \qquad \Omega^n\leqslant 1\\&=\cosh(N\cosh^{-1}\Omega^n) \quad \Omega^n>1\end{aligned} \tag{7-38}$$

ε 是控制通带波纹的参数。通带幅度在 1 和 $1/(1+\epsilon^2)^{0.5}$ 之间振荡。若 N 为奇数，则在 $\Omega^n=0$ 处幅度为 1，若 N 为偶数，则幅度为 $1/(1+\epsilon^2)^{0.5}$。在归一化截止频率 1rad/sec 处幅度为 $1/(1+\epsilon^2)^{0.5}$，因此通带衰减 α 可写为：

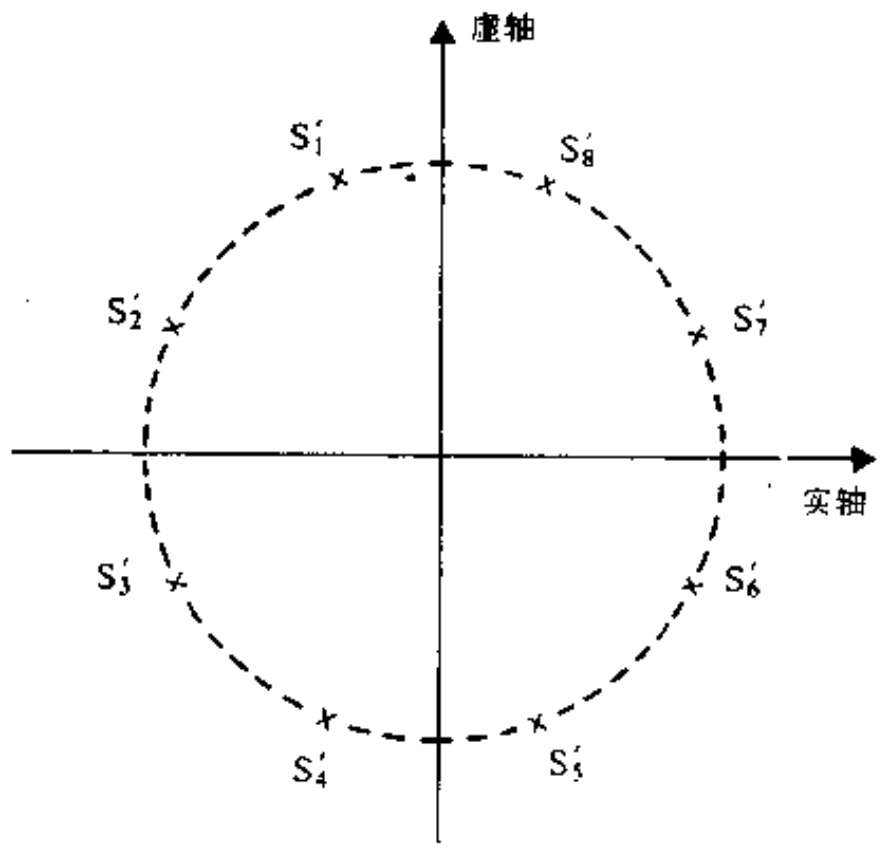


图 7-9 N = 4 的 Butterworth 极点

$$-10\log\frac{1}{(1+\epsilon^2)}=\alpha \tag{7-39}$$

因此，波纹参数 ε 由下式决定：

$$\epsilon=(10^{0.1\alpha}-1)^{0.5} \tag{7-40}$$

类似地，阻带衰减 β 为：

$$-10\log\frac{1}{(1+\epsilon^2 T_N^2(\Omega_H^n))}=\beta \tag{7-41}$$

用(7-38)和(7-41)式,在归一化阻带截止频率Ω_H^n处,Chebyshev多项式$T_N(\Omega_H^n)$可写为:

$$T_N(\Omega_H^n)=[(10^{0.1\beta}-1)/\epsilon]^{0.5}=\cosh(N\cosh^{-1}\Omega_H^n) \tag{7-42}$$

由(7-42)式,Chebyshev滤波器阶数N是满足下式的最小整数:

$$N\geqslant\frac{\cosh^{-1}[(10^{-0.1\beta}-1)/\epsilon]^{0.5}}{\cosh^{-1}\Omega_H^n} \tag{7-43}$$

归一化Chebyshev滤波器与Butterworth滤波器一样,也由s′平面的极点确定,其极点在s′平面的椭圆上如图7-10所示。为了形成稳定的归一化低通滤波器,要选择在稳定域中的极点。对偶数N,稳定极点s'_i由下式给出:

$$s_i'=\mu_i+j\eta_i \tag{7-44}$$

其中

$$\mu_i=-\sinh\gamma\cos\frac{i\pi}{2N}\qquad i=\pm1,\pm3,\cdots,\pm N-1 \tag{7-45}$$

$$=\cosh\gamma\sin\frac{i\pi}{2N}\qquad i=\pm1,\pm3,\cdots,\pm N-1 \tag{7-46}$$

$$\gamma=\frac{1}{N}\sinh^{-1}\frac{1}{\epsilon} \tag{7-47}$$

如前节所述,复共轭根为二阶多项式的根。对偶数N,$H_n(s')$有以下形式:

$$H_n(s')=\pi_i\frac{1}{s_i'^2-2\mu_i s'+(\mu_i^2+\eta_i^2)} \tag{7-48}$$

其中

$$i=1,3,5,\cdots,N-1$$

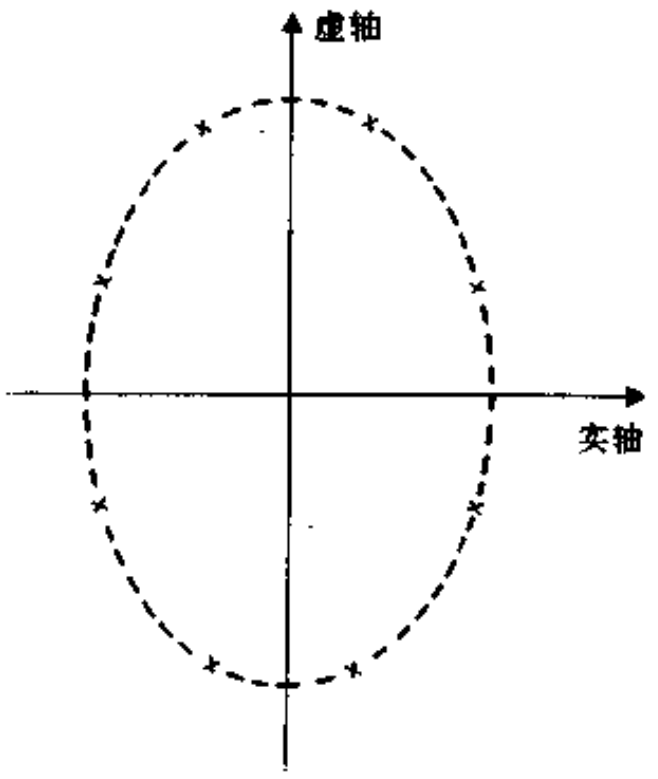


图7-10 N=4时的Chebyshev极点

7.5 模拟滤波器设计

用模拟频率变换把归一化低通模拟滤波器转换为所要求的模拟滤波器,这里模拟低通、高通、带通和带阻滤波器的传输函数可写为:

$$H_{LP}(s)=H_n(s')|_{s'=s/\Omega^+} \tag{7-49}$$

$$H_{HP}(s)=H_n(s')|_{s'=\Omega^+/s} \tag{7-50}$$

$$H_{BP}(s)=H_n(s')|s'=\frac{s^2+\Omega_1^*\Omega_6^*}{s(\Omega_7^*-\Omega_6^*)} \tag{7-51}$$

$$H_{BS}(s)=H_n(s')|s'=\frac{s(\Omega_{11}^*-\Omega_{10}^*)\Omega_H^*}{s^2+\Omega_{22}^*\Omega_{10}^*} \tag{7-52}$$

7.6 数字滤波器设计

设计过程的最后一步是用双线性变换从模拟滤波器得到所要求的数字滤波器:

$$H(z)=H(s)|_{s=(2/T)(Z-\frac{1}{Z}+1)} \tag{7-53}$$

传输函数 H(z) 可分解为二阶节的乘积。以下各节中，用系数 b_{1i}、b_{2i}、a_{1i} 和 a_{2i} 以及总增益 K 在 DSP56001 处理器上实现 IIR 滤波器。

7.7 在 DSP56001 处理器上实现二阶节

对图 7-4 中的第 i 个二阶节，输入采样 $y_i(n)$ 在寄存器 A 中。而输出采样 $y_{i+1}(n)$ 也在寄存器 A 中，对(i + 1) 节，$y_{i+1}(n)$ 是输入采样。

由(7-8) 和(7-9) 式，对第 i 个二阶节的差分方程可写为：

$$x_i(n) = y_i(n) + a_{1i}x_i(n-1) + a_{2i}x_i(n-2) \tag{7-54}$$

$$y_{i+1}(n) = x_i(n) + b_{1i}x_i(n-1) + b_{2i}x_i(n-2) \tag{7-55}$$

因为实际的系数从 −2 到 +2，故可采用定标方式并且系数 a_{1i}、a_{2i}、b_{1i}、b_{2i} 是滤波器实际值除以 2。

图 7-11 中表示第 i 个二阶节执行前对滤波器状态 X 存贮器的映射和对系数 Y 存贮器的映射。R0 初始化后在 X 存储器指向滤波器状态，而 R4 初始化后在 Y 存储器指向滤波器系数。注意在此序列中存着滤波器状态。序列的第一个元素是第二滤波器状态 $x_i(n-2)$，而第二个元素是第一滤波器状态 $x_i(n-1)$。类似地，系数的第一个元素是 a_{2i} 和 a_{1i}，随之是 b_{2i} 和 b_{1i}。R0 和 R4 中的起始值分别假设是 $50 和 $100。

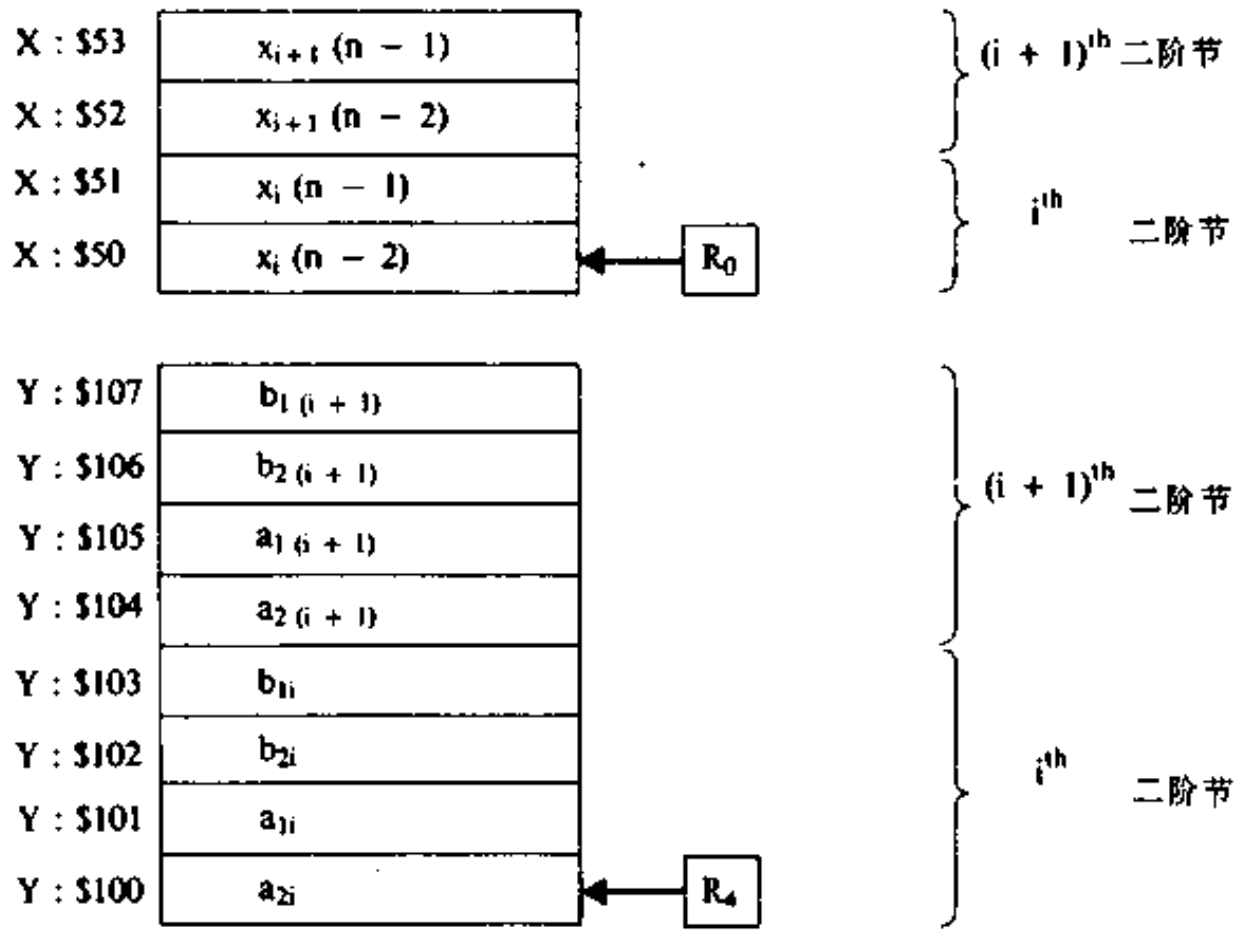


图 7-11 第 i 个二阶节执行前的存贮器映像

以下 DSP56001 指令用来实现第 i 个二阶节：

```
ori     #$8,mr
move                x:(r0)+,x0      y:(r4)+,y0
mac     x0,y0,a     x:(r0)-,x1      y:(r4)+,y0
macr    x1,y0,a     x,x:(r0)+       y:(r4)+,y0
mac     x0,y0,a     a,x:(r0)+       y:(r4)+,y0
mac     x1,y0,a     x:(r0)+,x0      y:(r4)+,y0
```

第 i 个二阶节工作如下：

1. ORI 指令选择 DSP56001 处理器升标方式。

2. MOVE 指令从 X 存储器单元 50(由 R0 指示) 传送 $X_i(n-2)$ 到数据 ALU 输入寄存器 X0，同时从 Y 存储器单元 100(由 R4 指示) 传送 a_{2i} 到数据 ALU 输入寄存器 Y0。地址寄存器 R0 是后递增 1 以指示 X 存储器单元 $51 中的 $x_i(n-1)$。R4 是后递增 1 以指示 Y 存储器单元 $101 中的 a_{1i}。执行 MOVE 指令后各存储器和寄存器内容如图 7-12 所示。

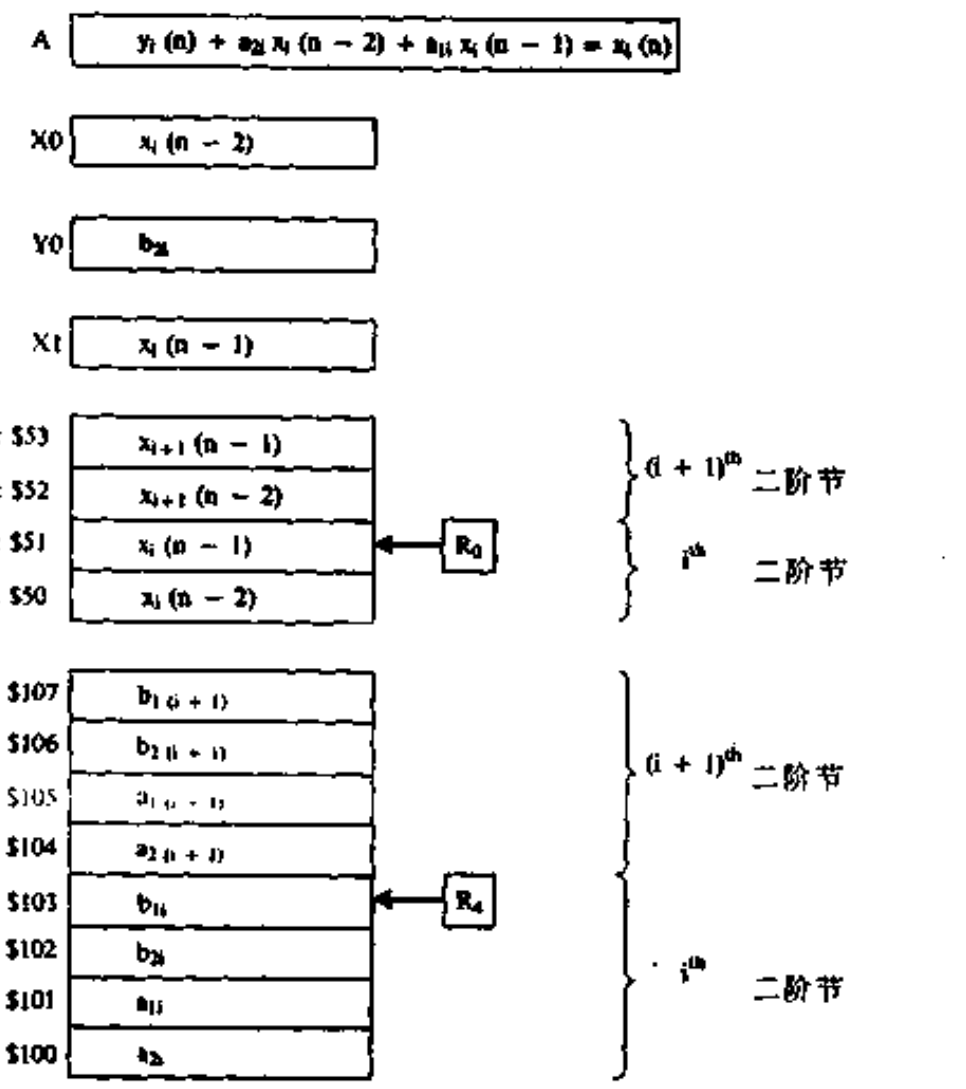


图 7-12 MOVE 指令执行后的存储器映像和数据寄存器

3. 第一条 MAC 指令使来自 X0 的 $x_i(n-2)$ 和来自 Y0 的 a_{2i} 相乘，把乘积加到来自 A 累加器的 $y_i(n)$，即 $[x_i(n-2)a_{2i}] + y_i(n) \rightarrow$ A 累加器。在 MAC 指令和并行传送部分，把来自 X 存储器单元 $51 的 $x_i(n-1)$ 传送到 X1，而把来自 Y 存储器单元 $101 中的下一个系数 a_{1i} 传送到 Y0。R0 后递减 1 以指回 X 存器单元 $50 中的 $X_i(n-2)$，而 R4 后递增 1 以指示 Y 存储器单元 $102 中的 b_{2i}。第一条 MAC 指令执行后，各寄存器和存储器内容如图 7-13 所示。

4. MACR 指令使来自 x1 的 $x_i(n-1)$ 与来自 Y0 的 a_{1i} 相乘，乘积加到来自 A 累加器的 $[y_i(n) + a_{2i}x_i(n-2)]$，并舍入其结果，即 $x_i(n-1)a_{1i} + [y_i(n) + a_{2i}x_i(n-2)]$ 以收敛舍入的结果送入 A 累加器。在 MACR 指令的并行传送部分，把来自 x1 的 $x_i(n-1)$ 传送到 X 存贮器单元 $50，而把下一个系数 b_{2i} 从 Y 存贮器 $102 传送到 Y0。R0 后递增 1 以指示 X 存贮器 $51 中的 $x_i(n-1)$；R4 后递增 1 以指示 Y 存贮器 $103 中的 b_{1i}。注意 $x_i(n-1)$ 写在 X 存贮器 $50 中的 $x_i(n-2)$ 上面。MACR 指令执行后各存贮器和寄存器内容如图 7-14 所示。

5. 第 2 条 MAC 指令使 X0 中的 $x_i(n-2)$ 与 Y0 中的 b_{2i} 相乘，并将乘积加至 A 累加器。在 MAC 指令的并行传送部分，来自 A 累加器的 $x_i(n)$ 放大 2 倍并传送到 R0 所指的 X 存贮器

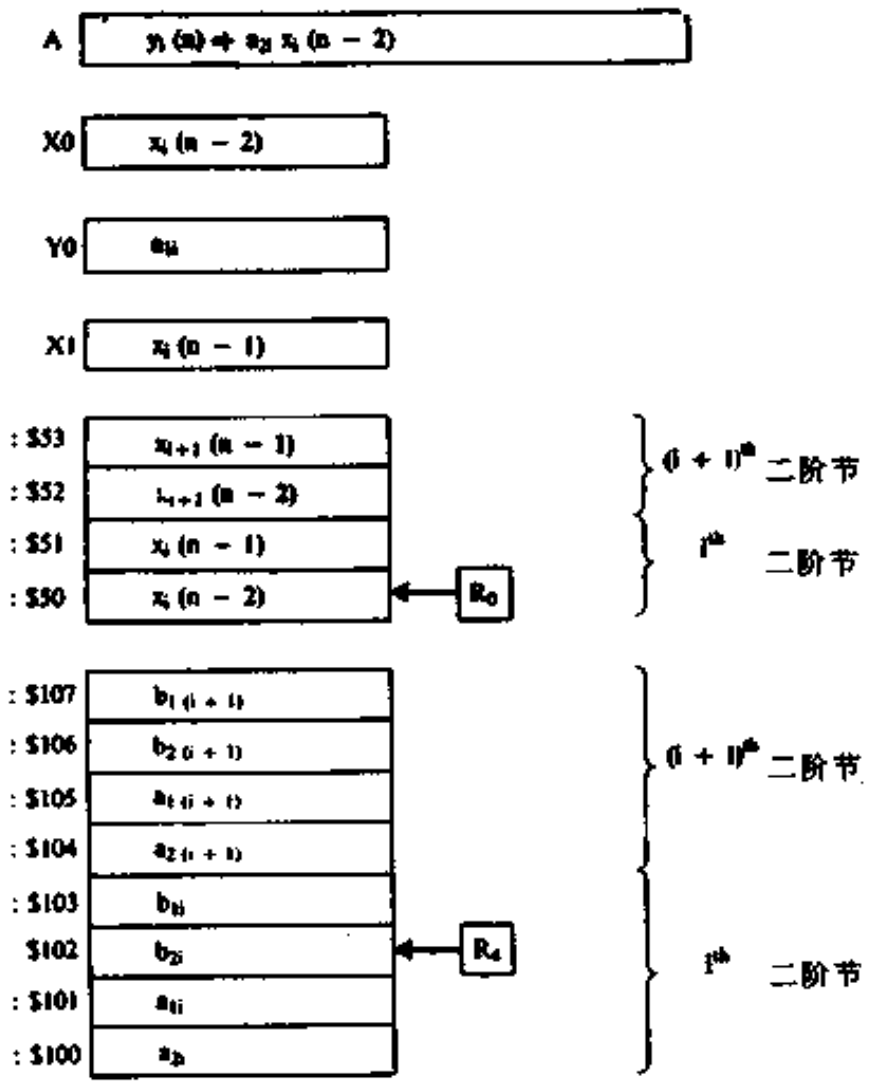


图 7－13　第一条 MAC 指令执行后的存贮器映像和数据寄存器

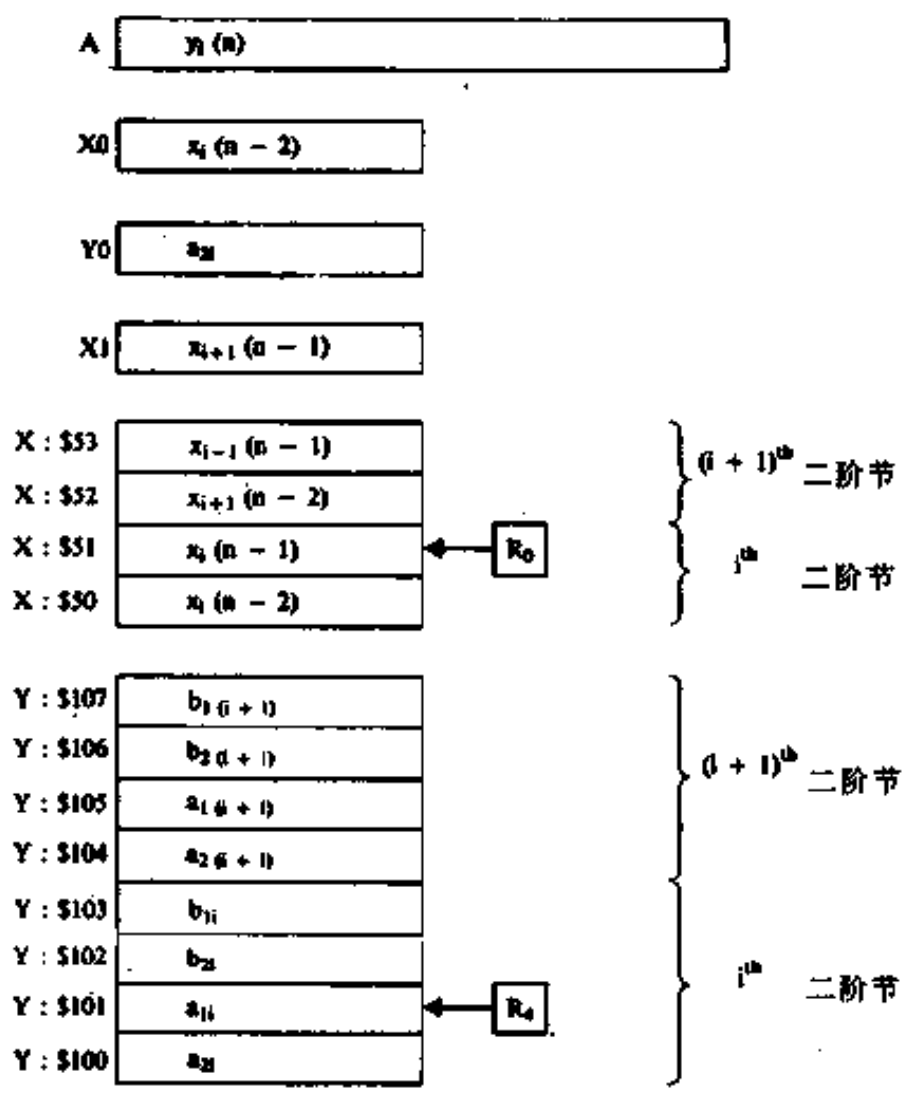


图 7－14　MACR 指令执行后的存贮器映像和数据寄存器

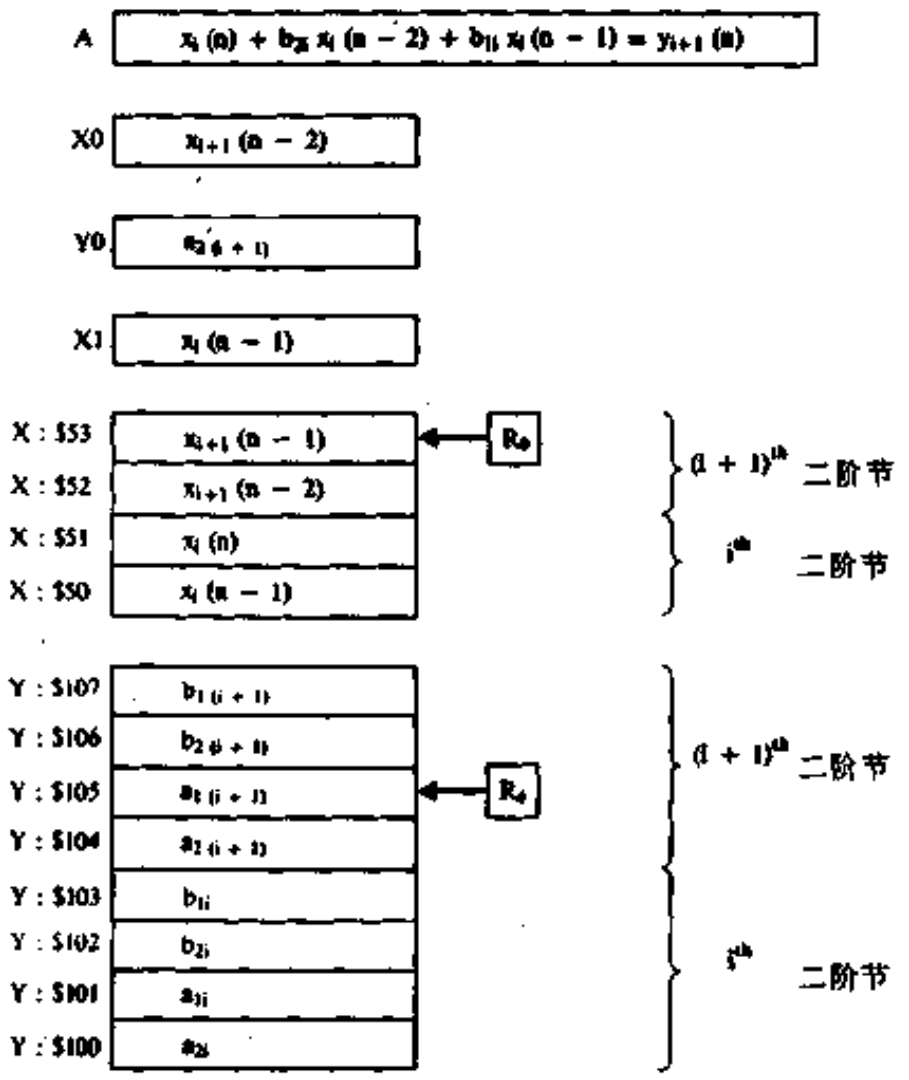


图 7－15　第二个 MAC 指令执行后的存贮器映像和数据寄存器

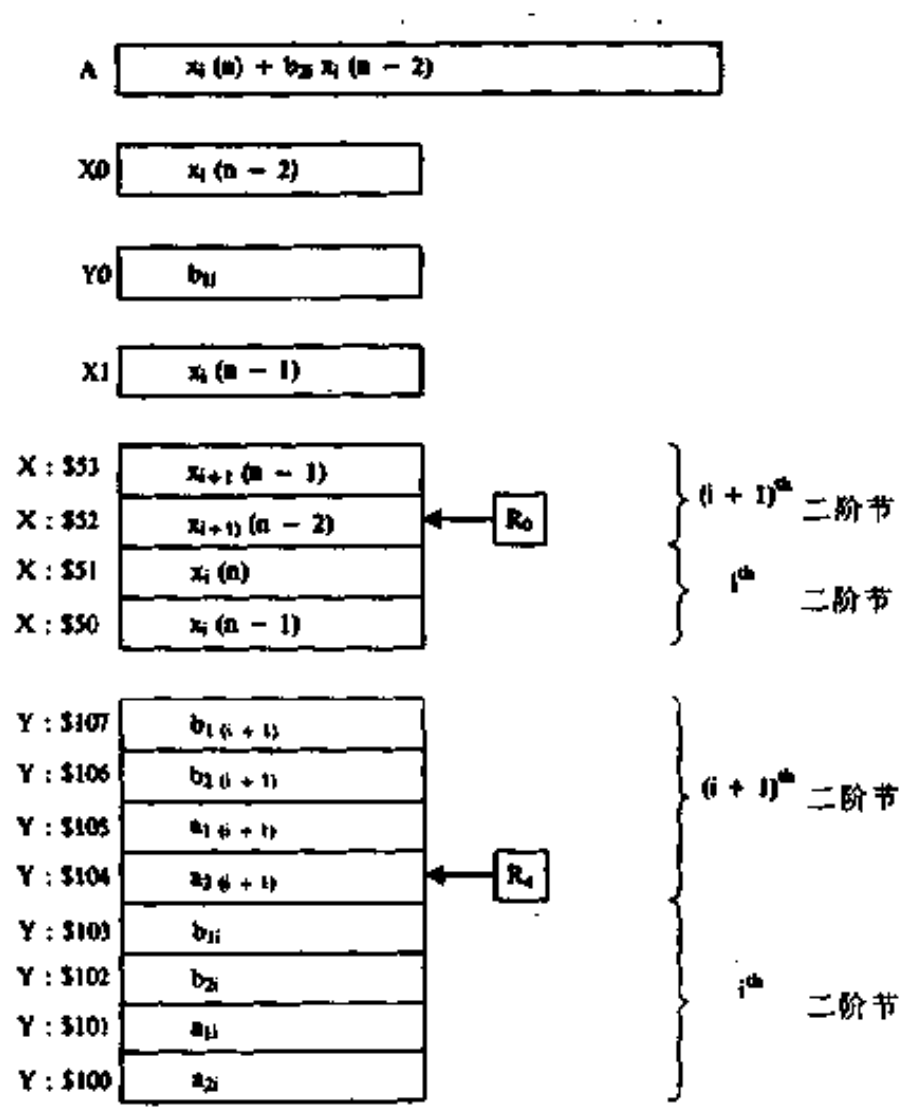


图 7－16　最后的 MAC 指令执行后的存贮器映像和数据寄存器

$51，并将系数 b_{1i} 从 Y 存贮器 $103 传送到 Y0。同样，R0 后递增 1 以指示 X 存贮器 $52 中第 i＋1 个滤波节的 $x_{i+1}(n-2)$。R4 后递增 1 以指示 Y 存贮器 $104 中第 i＋1 节滤波器的系数 $a_{2(i+1)}$。注意 $x_i(n)$ 乘 2 写在 X 存贮器 $51 中的 $x_i(n-1)$ 上面。第二条 MAC 指令执行后，存贮器和寄存器内容如图 7－15 所示。

6. 最后的 MAC 指令使 X1 中的 $x_i(n-1)$ 和 Y0 中 b_{1i} 相乘，并将乘积加至 A 累加器。A 累加器中的结果是二阶节 $y_{i+1}(n)$ 的输出和第 i＋1 节的输入。在 MAC 指令的并行传送部分，把 $x_{i+1}(n-2)$ 从 X 存贮器 $52 传送到 X0，同时把第 i＋1 节的第一个系数 $a_{2(i+1)}$ 从 Y 存贮器 $104 传送到 Y0。R0 后递增 1 以指示 X 存贮器 $53 中的 $x_{i+1}(n-1)$，R4 加 1 以指示 Y 存贮器 $105 中的 $a_{1(i+1)}$。注意第 i＋1 节开始的情况与第 i 节开始的情况完全相同。最后的 MAC 指令执行后存贮器和寄存器内容如图 7－16 所示。

7.8　实现 IIR 滤波器

用二阶节级联以实现 IIR 滤波器如图 7－5 所示。二阶节本身的实现如上所述。

为第一个二阶节产生输入 $y_1(n)$ 的 DSP56001 指令如图 7－17 所示。首先，输入序列 x(n) 的第一个元素在 A 累加器中。第一条 MOVE 指令传送 8 位立即指针值 gk 到 R1。因为 R1 刚好改变，第一条 NOP 指令用以提供足够的访问时间，使由 R1 指示的 X 存贮器数据在第二条 MOVE 指令时可有效使用。第二条 MOVE 指令将“总增益”值从 R1 所指的 X 存贮器单元传送到 Y1。第三条 MOVE 指令将 x(n) 从 A 累加器传送到 X1。在 MPY 指令中，来自 X1 的 x(n) 与来自 Y1 的“总增益”相乘，并将乘积 $y_1(n)$ 放于 A 累加器中。

```
move    #gk,r1
hop
move    X:(r1),y1
move    a,X1
mpy     X1,X0,a
```

图 7－17　DSP56001 汇编指令为第一个二阶第产生 $y_1(n)$ 输入

实现 NSEC 级联二阶节的 DSP56001 指令如图 7－18 所示，这里 NSEC 指定二阶节的数量。注意，核心“二阶节”指令已在 7.7 节中说明。其中，DO 指令使硬件“do loop”执行实现二阶节的指令组(NSEC 次)。RND 指令收敛地舍入 A 累加器内容以产生 IIR 滤波器的最后 24 位输出。ANDI 指令不允许升标方式。

```
        ori     #$8,mr
        nop
        move                    x:(r0)+,x0    y:(r4)+,y0
        do      #NSEC,_iirend
        mac     x0,y0,a         x:(r0)-,x1    y:(r4)+,y0
        macr    x1,y0,a         x1,x:(r0)+    y:(r4)+,y0
        mac     x0,y0,a         a,x:(r0)+     y:(r4)+,y0
        mac     x1,y0,a         x:(r0)+,x0    y:(r4)+,y0
_iirend
        rnd     a
        andi    #$f7,mr
```

图 7－18　DSP56001 汇编指令以实现级联二阶节

7.9 设计和实现 IIR 滤波器

图 7-17 和 7-18 所示的 DSP56001 指令与图 5-8 所示的 DSP56001 指令一起形成的 IIR.ASM 汇编程序宏指令示于图 7-19。FDAS 程序将用於设计各种滤波器。所设计的波器系数和 IIR.ASM 程序宏指令将用於在 DSP 系统上实现各种 IIR 滤波器。

```
iir     macro  states, coef, gk, nsec
;    states =Location of filter states in X memory
;    coef   =Location of filter coefficients in Y memory
;    gk     =Total gain K
;    nsec   =Number of biquad sections

;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Initialize routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
             movep      #0, x: $FFFF
;
init2  reset
;

; set up ads board in case of force break instead of force
; reset
;
             movep      #0, x: $FFFE              ; set bcr to zero
             movec      #0, sp                    ; init stack pointer
             movec      #0, sr                    ; clear loop flag
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Set up the SSI for operation with the EVB
; The following code sets port C to function as SCI/SSI
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
pcc          equ        $0001ff
movep        #pcc, x: $FFE1                       ; write PCC
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; The following code sets the SSI CRA and CRB control
; registers for external continuous clock, synchronous,
; normal mode.
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
cra          equ        $004000
             movep      #cra, x: $FFEC            ; CRA pattern for word
;                                                 ; length=16 bits
;
crb          equ        $003200
             movep      #crb, x: $FFED            ; CRB pattern for continuous
;                                                 ; ck, sych, normal mode
;                                                 ; word long frame SYNC;
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Read A/D
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; The following code polls the RDF flag in the SSI - SR and
; waits for RDF = 1 and then reads the RX register to retrieve
; the data from the A/D converter.
; Sample rate is controlled by the EVB.
;
wait2           btst        #7, x: $FFEE
                jcc         wait2                          ; loop until RDF bit = 1
;
                movep       x: $FFEF, x1                   ; get A/D converter data
                move        #gk, r1
                nop
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Multiply the input by the overall gain
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
                move        x: (r1), x0
                mpy         x1, x0, a
                nop
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Initialize R0, R4, M0 and M4
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
                move        # $ ffff, m0
                move        m0, m4
                move        #states, r0
                move        #coef, r4
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; IIR Filter
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
                ori         # $ 8, mr
                nop
                move                        x: (r0) +, x0   y: (r4) +, y0
;
                do          #nsec, _iirend
                mac         x0, y0, a       x: (r0) -, x1   y: (r4) +, y0
                macr        x1, y0, a       x1, x: (r0) +   y: (r4) +, y0
                mac         x0, y0, a       a, x: (r0) +    y: (r4) +, y0
                mac         x1, y0, a       x: (r0) +, x0   y: (r4) +, y0
_iirend
;
                rnd         a
                andi        # $ f7, mr
                rol         a
                nop
;
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Write D/A
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                move        # $ ffff00, y1
                and         y1, a
                movep       a, x: $ FFEF        ; write D/A via SSI
                nop
                jmp         wait2               ; loop indefinitely
                nop
;
                endm
```

图 7-19　实现 IIR 数字滤波器的 IIR. ASM 汇编宏指令

7.9.1　设计和实现低通 IIR 滤波器

1. 设计所要求的低通 IIR 滤波器

(1) 键入 cd\dsptools\fdas<return>，进入 FDAS 程序目录。

(2) 键入 demo<return>，装入和进入 FDAS 程序。

(3) 根据 FDAS 程序菜单选择下面所列 IIR 滤波器指标。所设计的低通 IIR 滤波器的幅度响应、相位响应、脉冲响应、极点和零点图如图 7-20 所示。

(4) 退出 FDAS 程序。

滤波器指标：

滤波器型式：	低通
模拟滤波器型式：	Butterworth
采样频率：	48kHz
通带截止频率：	8kHz
阻带截止频率：	10kHz
通带衰减：	0.5dB
阻带衰减：	30.0dB
滤波器设计方法：	双线性变换

设计结果：

滤波器阶数：16

归一化模拟传输函数的二阶节
(前行为分子系数，后行为分母系数)

S^2 项	S 项	常数项
.000000E+00	.000000E+00	.100000E+01
.100000E+01	.199037E+01	.100000E+01
.000000E+00	.000000E+00	.100000E+01
.100000E+01	.191388E+01	.100000E+01
.000000E+00	.000000E+00	.100000E+01
.100000E+01	.176384E+01	.100000E+01
.000000E+00	.000000E+00	.100000E+01
.100000E+01	.154602E+01	.100000E+01
.000000E+00	.000000E+00	.100000E+01
.100000E+01	.126897E+01	.100000E+01
.000000E+00	.000000E+00	.100000E+01
.100000E+01	.942793E+00	.100000E+01

.000000E+00	.000000E+00	.100000E+01
.100000E+01	.580569E+00	.100000E+01
.000000E+00	.000000E+00	.100000E+01
.100000E+01	.196034E+00	.100000E+01

起始增益 1.00000000

模拟传输函数的二阶节
（前行为分子系数，后行为分母系数）

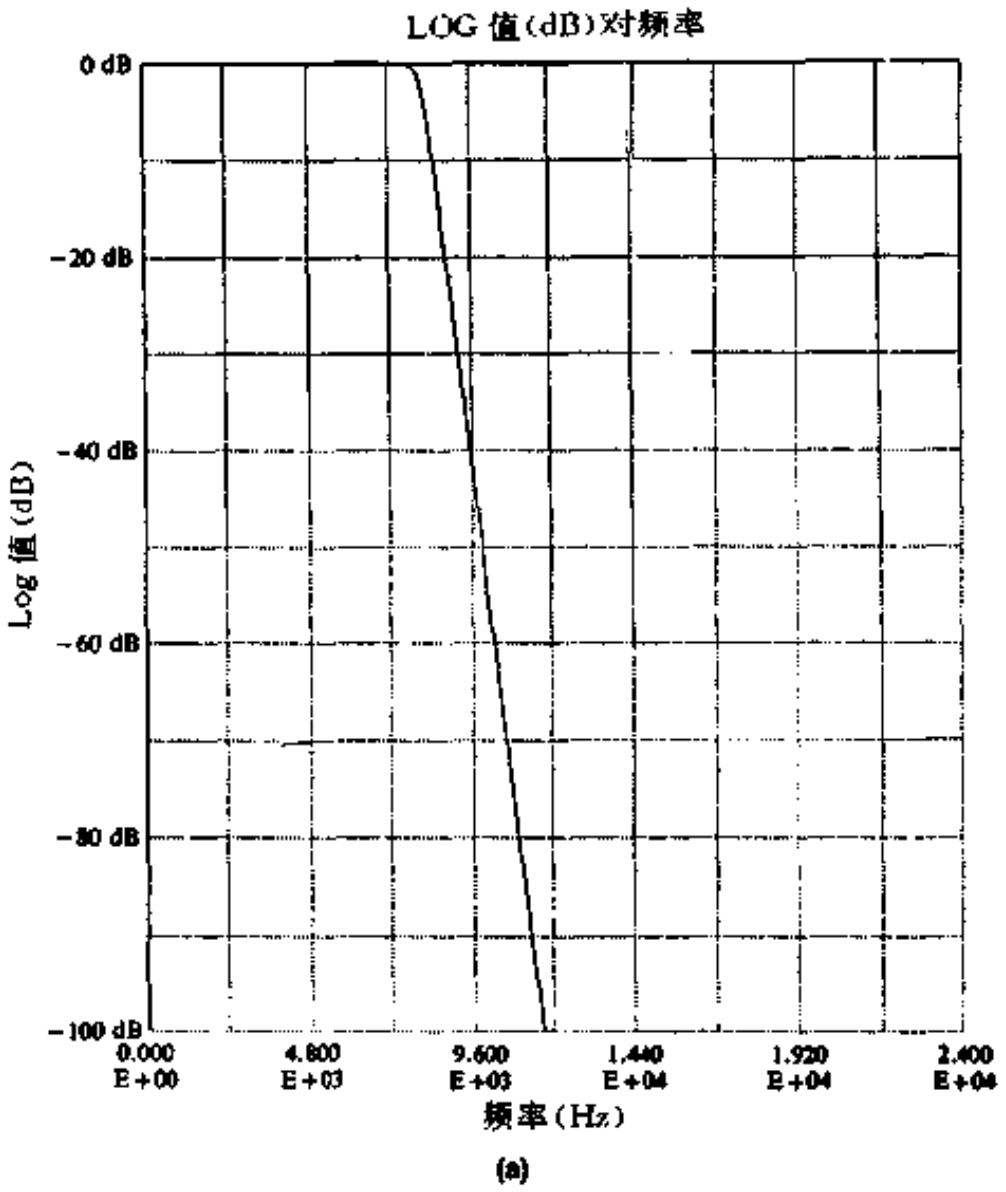
S^2 项	S 项	常数项
.000000E+00	.000000E+00	.350364E+10
.100000E+01	.117813E+06	.350364E+10
.000000E+00	.000000E+00	.350364E+10
.100000E+01	.113286E+06	.350364E+10
.000000E+00	.000000E+00	.350364E+10
.100000E+01	.104405E+06	.350364E+10
.000000E+00	.000000E+00	.350364E+10
.100000E+01	.915114E+05	.350364E+10
.000000E+00	.000000E+00	.350364E+10
.100000E+01	.751015E+05	.350364E+10
.000000E+00	.000000E+00	.350364E+10
.100000E+01	.558054E+05	.350364E+10
.000000E+01	.000000E+00	.350364E+10
.100000E+01	.343648E+05	.350364E+10
.000000E+01	.000000E+00	.350364E+10
.100000E+01	.116036E+05	.350364E+10

起始增益 1.00000000

数字传输函数的二阶节
（前行为分子系数，后行为分母系数）

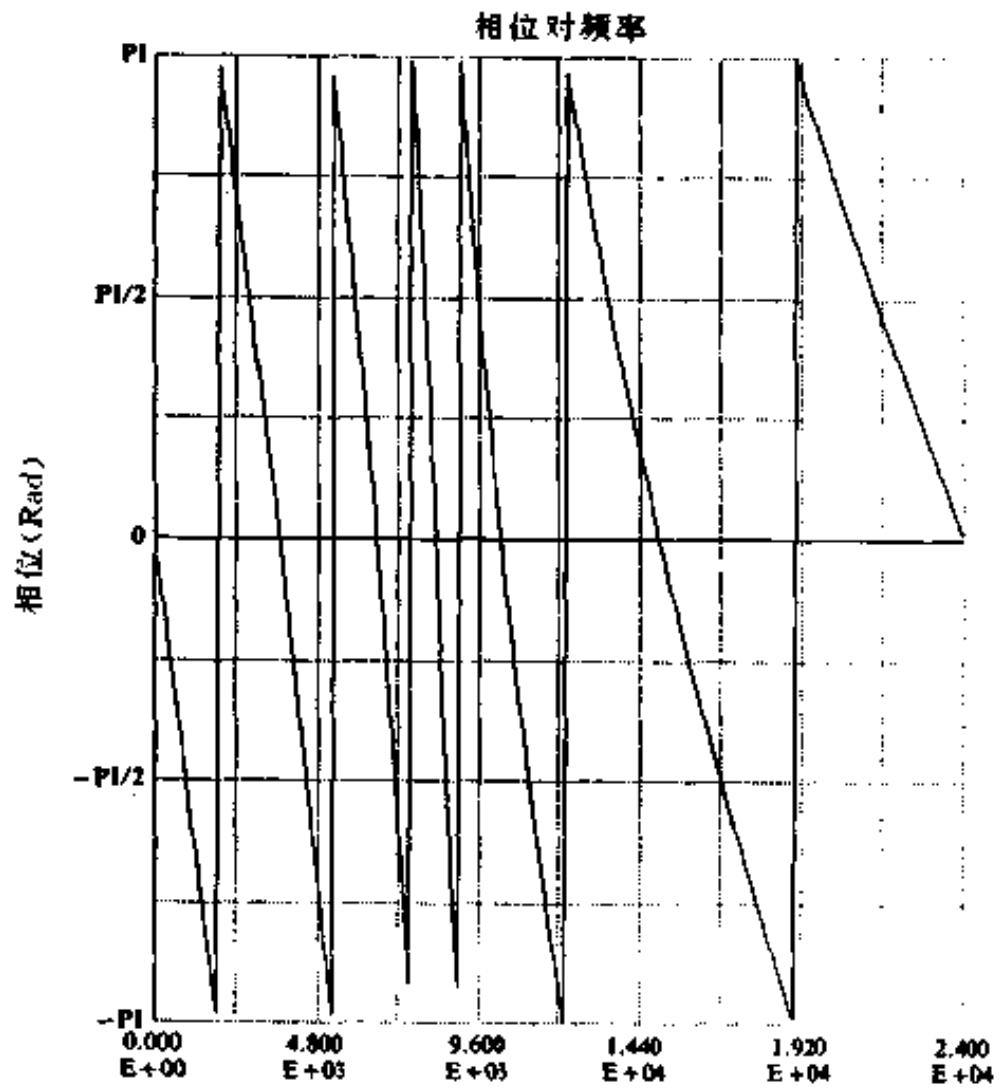
Z^2 项	Z 项	常数项
1.0000000000	2.0000000000	1.0000000000
1.00000	−.4754416049	.0586601458
1.0000000000	2.0000000000	1.0000000000
1.00000	−.4841995835	.0781614780
1.0000000000	2.0000000000	1.0000000000
1.00000	−.5023513436	.1185798347
1.0000000000	2.0000000000	1.0000000000
1.00000	−.5312651992	.1829618961
1.0000000000	2.0000000000	1.0000000000
1.00000	−.5732600093	.2764712274
1.0000000000	2.0000000000	1.0000000000
1.00000	−.6320043206	.4072764516
1.0000000000	2.0000000000	1.0000000000
1.00000	−.7132129073	.5881026983
1.0000000000	2.0000000000	1.0000000000
1.00000	−.8258681893	.8389508724

起始增益 .102573556E−05



(a)

图 7-20 (a)　低通滤波器幅度响应



(b)

图 7-20 (b)　低通滤波器的相位响应

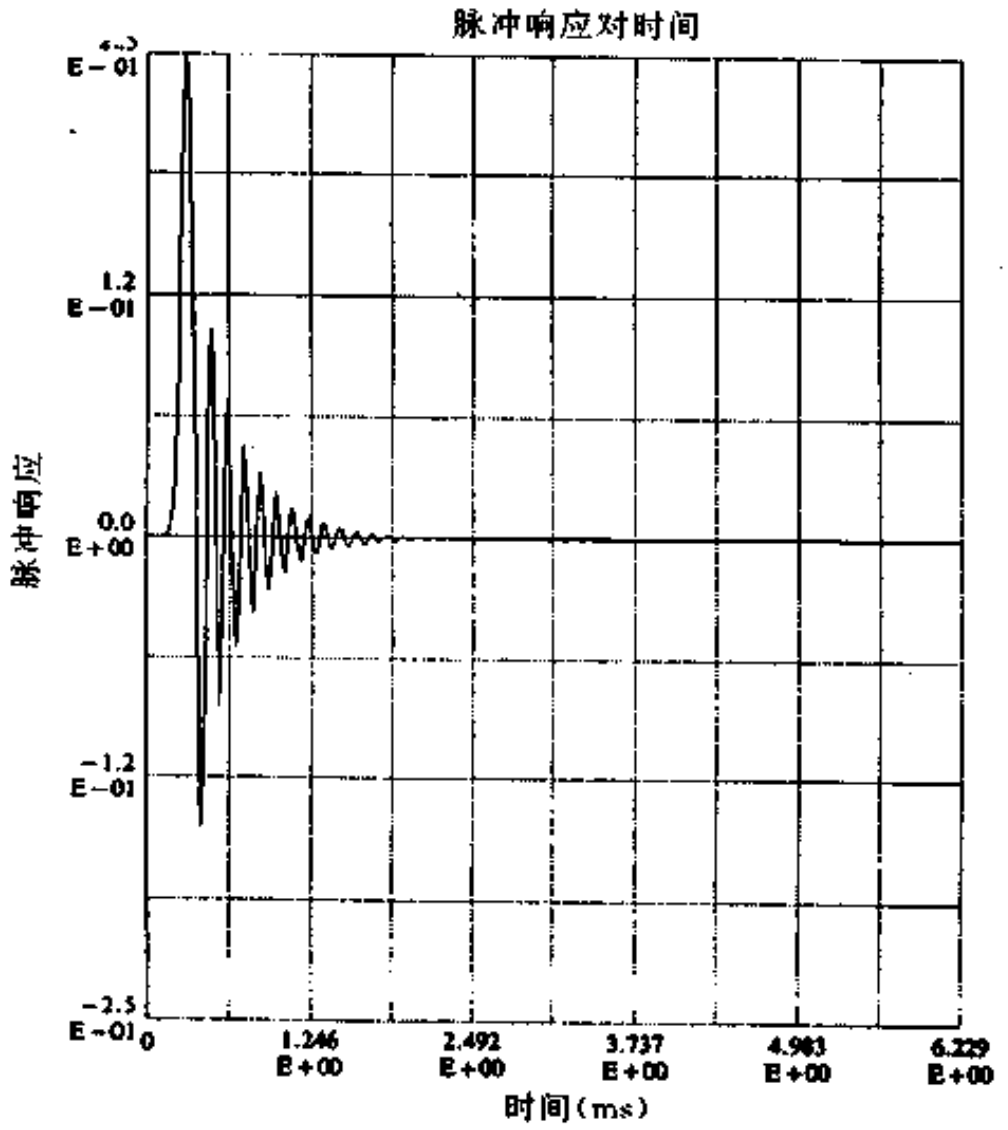


图 7-20（c）　低通滤波器的脉冲响应

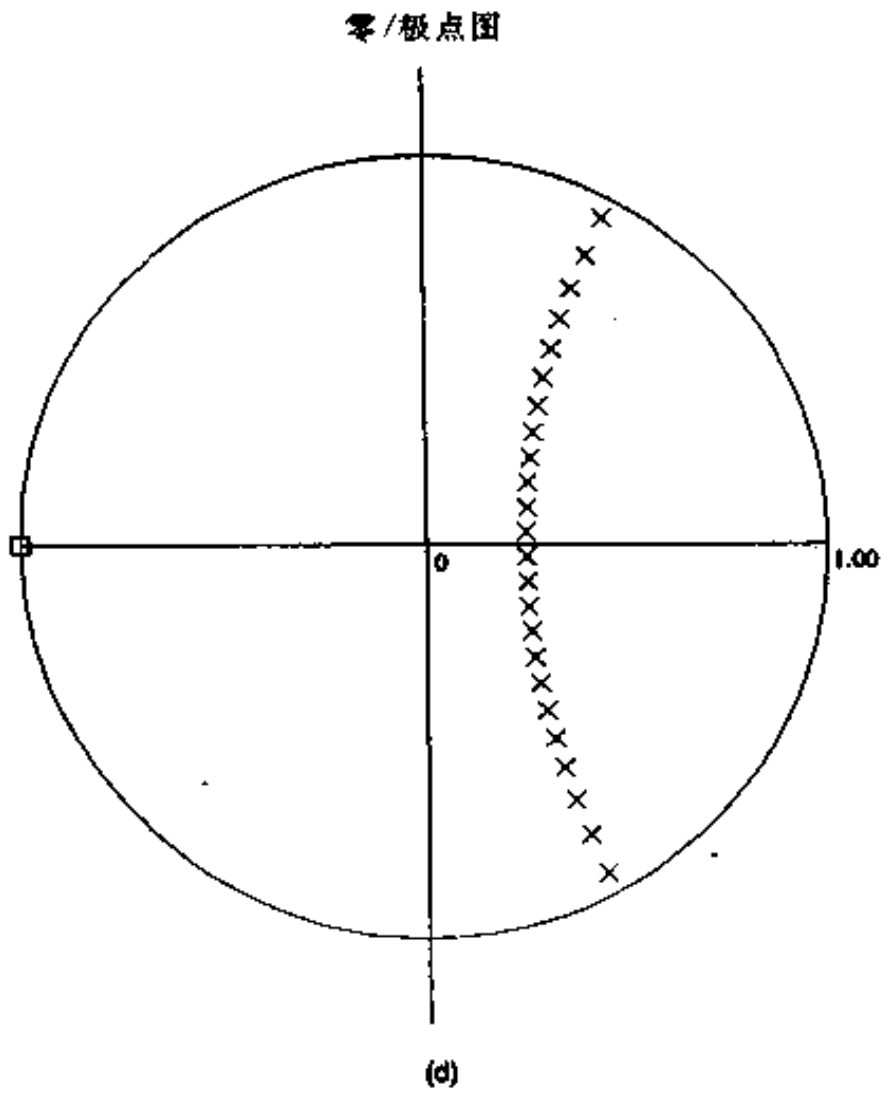


图 7-20（d）　低通滤波器的极点和零点

数字传输函数的零点		数字传输函数的极点	
实部	虚部	实部	虚部
-.100000000E+01	.00000000E+00	.237720802E+00	.463569406E-01
-.100000000E+01	.00000000E+00	.237720802E+00	-.463569406E-01
-.100000000E+01	.00000000E+00	.242099792E+00	.139818342E+00
-.100000000E+01	.00000000E+00	.242099792E+00	-.139818342E+00
-.100000000E+01	.00000000E+00	.251175672E+00	.235564464E+00
-.100000000E+01	.00000000E+00	.251175672E+00	-.235564464E+00
-.100000000E+01	.00000000E+00	.265632600E+00	.335262909E+00
-.100000000E+01	.00000000E+00	.265632600E+00	-.335262909E+00
-.100000000E+01	.00000000E+00	.286630005E+00	.440811148E+00
-.100000000E+01	.00000000E+00	.286630005E+00	-.440811148E+00
-.100000000E+01	.00000000E+00	.316002160E+00	.554453863E+00
-.100000000E+01	.00000000E+00	.316002160E+00	-.554453863E+00
-.100000000E+01	.00000000E+00	.356606454E+00	.678921597E+00
-.100000000E+01	.00000000E+00	.356606454E+00	-.678921597E+00
-.100000000E+01	.00000000E+00	.412934095E+00	.817579541E+00
-.100000000E+01	.00000000E+00	.412934095E+00	-.817579541E+00

直接型数字传输函数：

i) 分子系数（前面为Z的最高价）

.100000000E+01	.160000000E+02	.120000000E+03
.560000000E+03	.182000000E+04	.436800000E+04
.800800000E+04	.114400000E+05	.128700000E+05
.114400000E+05	.800800000E+04	.436800000E+04
.182000000E+04	.560000000E+03	.120000000E+03
.160000000E+02	.100000000E+01	

ii) 分母系数（前面为Z的最高价）

.100000000E+01	-.473760316E+01	.123146660E+02
-.217678558E+02	.287481903E+02	-.296453528E+02
.244823550E+02	-.164083913E+02	.897489810E+01
-.400386702E+01	.144745184E+01	-.418464884E+00
.946350381E-01	-.161528643E-01	.195923087E-02
-.150660767E-03	.552632393E-05	

2. 实现所设计的低通 IIR 滤波器

“IIRLP.ASM”汇编程序示于图 7-21，可在 DSP56001 处理器上实现所设计的低通 IIR 滤波器。

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; File: IIRLP.  ASM
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
                include 'iir'
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Coefficients of Low Pass IIR Filter
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
;
nsec        equ     $8
;
            org     x: $0
states      ds      2*nsec
;
            org     x: $20
gk          dc      .10257356E-05
;
            org     y: $0
coef
            dc -.58660146E-01/2.0
            dc .47544160/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
            dc -.78161478E-01/2.0
            dc .48419958/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
            dc -.11857983/2.0
            dc .50235134/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
            dc -.18296190/2.0
            dc .53126520/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
            dc -.27647123/2.0
            dc .57326001/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
            dc -.40727645/2.0
            dc .63200432/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
            dc -.58810270/2.0
            dc .71321291/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
            dc -.83895087/2.0
            dc .82586819/2.0
            dc 1.0000000/2.0
            dc 1.9999999/2.0
;
; program start
;
            org     p: $200
;
            iir     states, coef, gk, nsec
;
            end
```

图 7-21　实现所设计的低通 IIR 滤波器的 IIRLP.ASM 汇编程序

注意此程序有一“include”语句，可参考图 7-19 所示的 IIR.ASM 汇编程序宏指令。

为在 DSP 系统上证明所设计的低通 IIR 滤波器，应执行下述操作：

(1) 将函数发生器连接到 EVB 上的“BNC2”和双踪示波器的通道 1。

(2) 使函数发生器产生一 2V（峰-峰）的正弦波。

(3) 将示波器的通道 2 连接到 EVB 上的“BNC1”。

(4) 装入“ADS56000 用户接口软件”程序。

(5) 把“DSP56001 示范软件#3”软盘放入驱动器“A”。

(6) 键入 a：iirlp. lod<return>以装入 iirlp. lod 程序。

(7) 键入 go<return>，在 DSP56001 处理器上执行所设计的低通 IIR 滤波器程序。

(8) 将正弦输入频率从 0.5kHz 变化至 20kHz。观察示波器上模拟输入和输出信号。注意，此滤波器通过的频率为 0.5～8kHz，阻带频率为 10～20kHz。

(9) 键入 force b<return>以停止 IIR 滤波器算法的执行，并将 ADM 的控制返回到监视程序。

(10) 键入 quit<return>，退出 ADS56000 程序。

7.9.2 统计和实现带通 IIR 滤波器

1. 带通滤波器的设计

(1) 键入 cd\dsptools\fdas<return>，进入 FDAS 程序目录。

(2) 键入 demo<return>，装入和进入 FDAS 程序。

(3) 根据 FDAS 程序菜单选择以下所列 IIR 滤波器指标。带通滤波器的幅度响应、相位响应、脉冲响应、极点和零点图示于图 7-22。

(4) 退出 FDAS 程序。

滤波器指标：

滤波器型式：	带通
模拟滤波器型式：	Butterworth
采样频率：	48 kHz
通带截止频率：	6 kHz　12 kHz
阻带截止频率：	4 kHz　14 kHz
通带衰减：	0.5 dB
阻带衰减：	50.0 dB
滤波器设计方法：	双线性变换

设计结果：

滤波器阶数：28

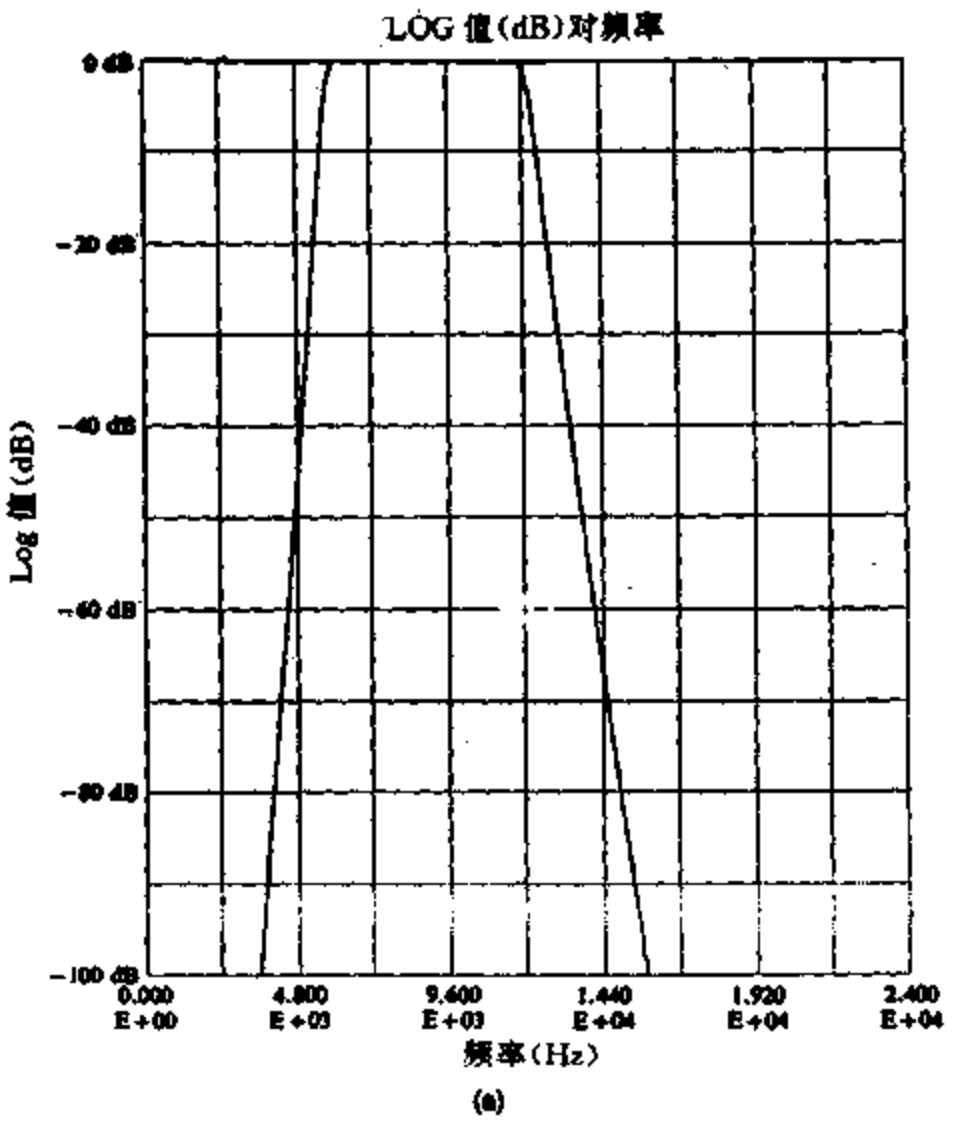


图 7-22 (a)　带通滤波器的幅度响应

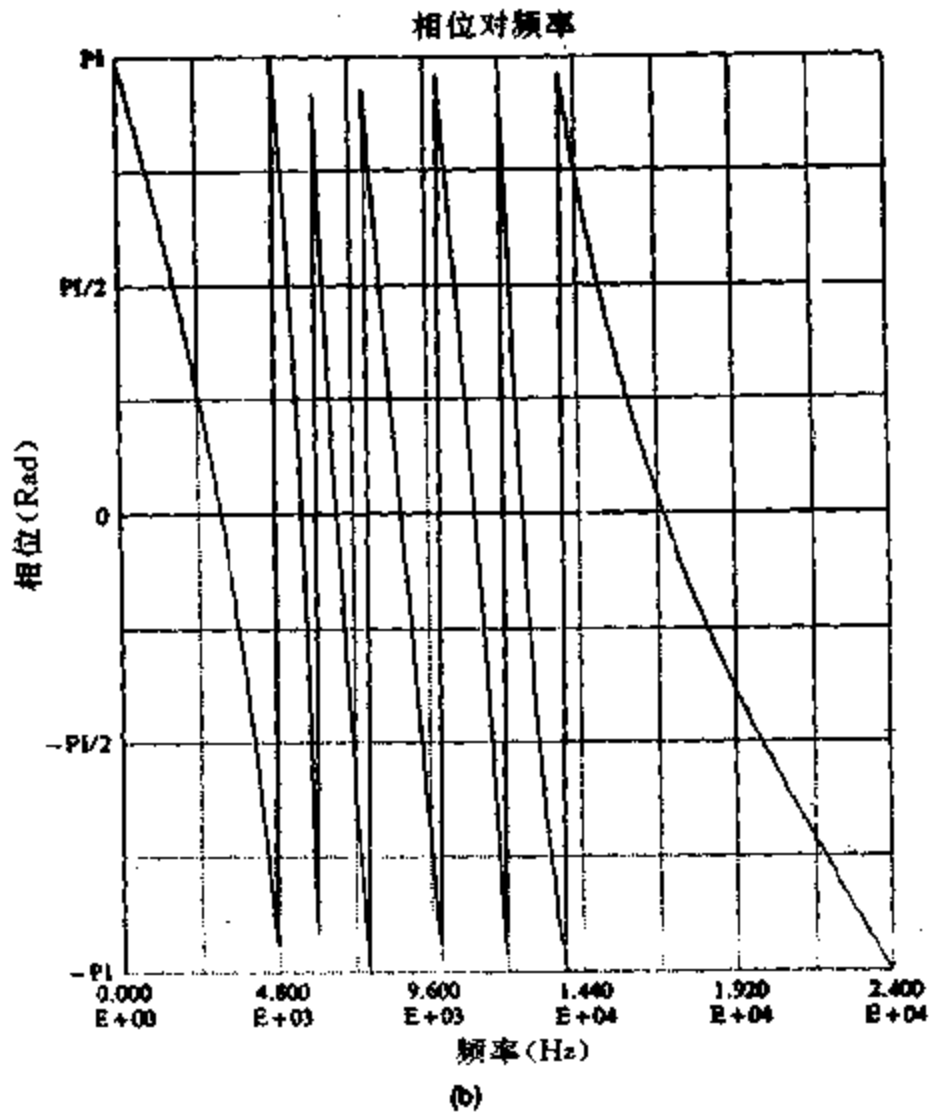


图 7-22 (b)　带通滤波器的相位响应

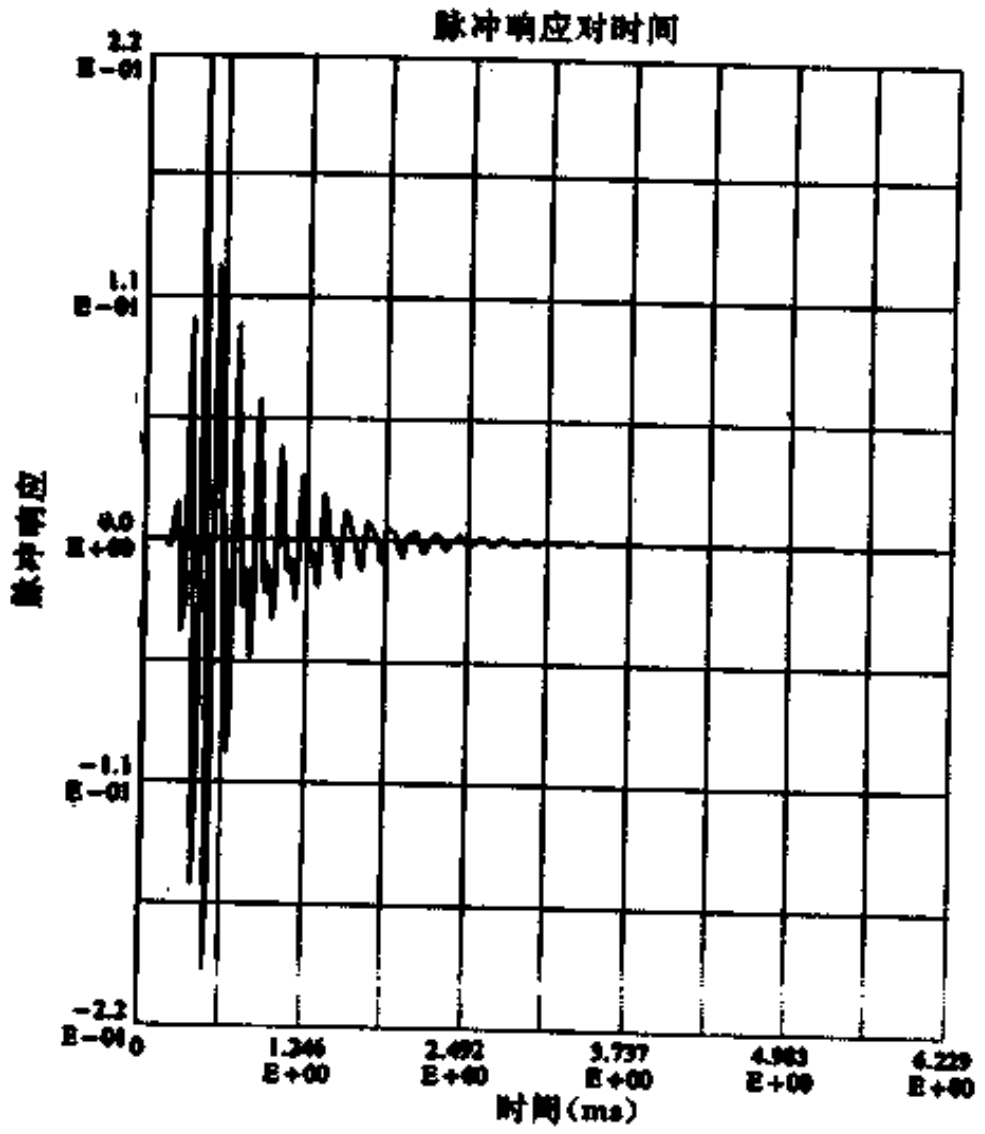


图 7-22 (c)　带通滤波器的脉冲响应

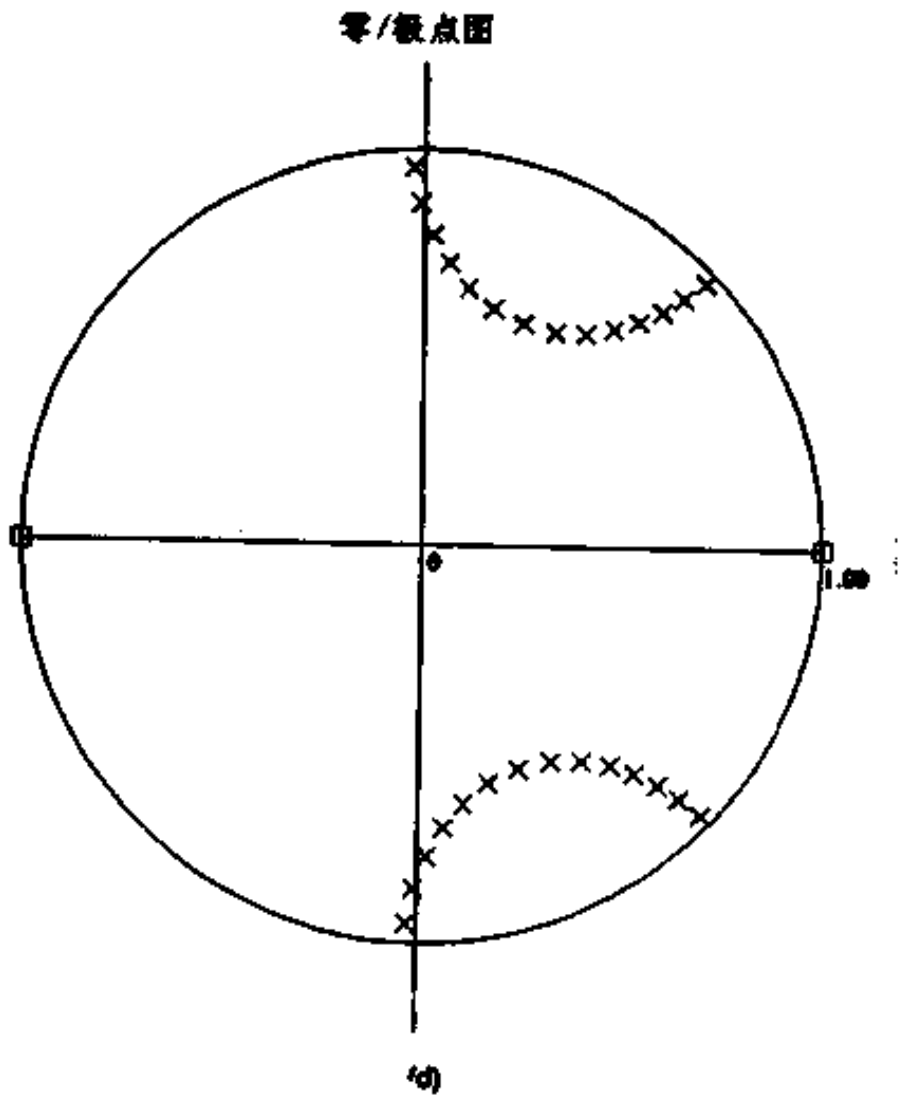


图 7-22 (d)　带通滤波器的极点和零点

归一化模拟传输函数的二阶节

（前行为分子系数，后行为分母系数）

S^2 项	S 项	常数项
. 000000E+00	. 000000E+00	. 100000E+01
. 100000E+01	. 198742E+01	. 100000E+01
. 000000E+00	. 000000E+00	. 100000E+01
. 100000E+01	. 188777E+01	. 100000E+01
. 000000E+00	. 000000E+00	. 100000E+01
. 100000E+01	. 169345E+01	. 100000E+01
. 000000E+00	. 000000E+00	. 100000E+01
. 100000E+01	. 141421E+01	. 100000E+01
. 000000E+00	. 000000E+00	. 100000E+01
. 100000E+01	. 106406E+01	. 100000E+01
. 000000E+00	. 000000E+00	. 100000E+01
. 100000E+01	. 660558E+00	. 100000E+01
. 000000E+00	. 000000E+00	. 100000E+01
. 100000E+01	. 223929E+00	. 100000E+01

超始增益　1.00000000

模拟传输函数的二阶节

（前行分子系数后行分母系数）

S^2 项	S 项	常数项
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 564618E+05	. 336660E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 640221E+05	. 432855E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 469741E+05	. 265782E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 674682E+05	. 548287E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 372745E+05	. 217612E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 653876E+05	. 669653E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 281013E+05	. 186133E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 576327E+05	. 782906E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 195649E+05	. 166186E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 449419E+05	. 876880E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 115335E+05	. 154420E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 285116E+01	. 943691E+10
. 000000E+00	. 606231E+05	. 000000E+00
. 100000E+01	. 381029E+05	. 148954E+10
. 000000E+00	. 606231E+05	. 000000E+00

. 100000E+01	. 976498E+05	. 978320E+10

超始增益 1.00000000

数字传输函数的二阶节系数

Z^2 项	Z 项	常数项
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 4964235425	. 3757326901
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 3525845110	. 3882693052
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 6498274207	. 3978390992
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 2270842493	. 4342242777
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 8005914688	. 4495001435
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 1228588596	. 5098953843
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 9405009747	. 5219451189
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−1. 0678237677	. 6083174348
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−. 0401085988	. 6130425334
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−1. 1843982935	. 7055158019
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	. 0206551403	. 7440753579
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−1. 2929184437	. 8134030700
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	. 0568998940	. 9059581757
1. 0000000000	. 0000000000	−1. 0000000000
1. 00000	−1. 3957597017	. 9339216352

超始增益 .252520644E−06

数字传输函数的零点

实部	虚部
. 100000000E+01	. 000000000E+00
−. 100000000E+01	. 000000000E+00

共 14 次重根

数字传输函数的极点		数字传输函数的极点	
实部	虚部	实部	虚部
. 248211741E+00	. 560467298E+00	. 533911824E+00	. 568555661E+00
. 248211741E+00	−. 560467298E+00	. 533911824E+00	−. 568555661E+00
. 176292241E+00	. 597654040E+00	. 200542808E+01	. 782713421E+00
. 176292241E+00	−. 597654040E+00	. 200542808E+01	−. 782713421E+00
. 324913681E+00	. 540620171E+00	. 592199087E+00	. 595664221E+00

. 324913681E+00	−. 540620171E+00	. 592199087E+00	−. 595664221E+00
. 113542080E+00	. 649101259E+00	−. 103275180E−01	. 862536168E+00
. 113542080E+00	−. 649101259E+00	−. 103275180E−01	−. 862536168E+00
. 400295734E+00	. 537832138E+00	. 646459222E+00	. 628882633E+00
. 400295734E+00	−. 537832138E+00	. 646459222E+00	−. 628882633E+00
. 614293814E−01	. 711422347E+00	−. 284498930E−01	. 951393073E+00
. 614293814E−01	−. 711422347E+00	−. 284498930E−01	−. 951393073E+00
. 470250487E+00	. 548461118E+00	. 697879791E+00	. 668494856E+00
. 470250487E+00	−. 548461118E+00	. 697879791E+00	−. 668494856E+00

直接型数字传输函数

i) 分子系数（前面为 Z 的最高阶）

. 276490596E−07	. 000000000E+00	−. 387086834E−06
. 000000000E+00	. 251606442E−05	. 000000000E+00
−. 100642577E−04	. 000000000E+00	. 276767086E−04
. 000000000E+00	−. 553534172E−04	. 000000000E+00
. 830301258E−04	. 000000000E+00	−. 948915724E−04
. 000000000E+00	. 830301258E−04	. 000000000E+00
−. 553534172E−04	. 000000000E+00	. 276767086E−04
. 000000000E+00	−. 100642577E−04	. 000000000E+00
. 251606442E−05	. 000000000E+00	−. 387086834E−06
. 000000000E+00	. 276490596E−07	

ii) 分母系数（前面为 Z 的最高阶）

. 625000000E−01	−. 530832767E+00	. 251207986E+01
−. 843389559E+01	. 222337880E+02	−. 484798859E+02
. 902932943E+02	−. 146681774E+03	. 210903238E+03
−. 271190807E+03	. 314219049E+03	−. 329838470E+03
. 314876521E+03	−. 274044842E+03	. 217739867E+03
−. 157987192E+03	. 104612494E+03	−. 631065002E+02
. 345835497E+02	−. 171460187E+02	. 764673669E+01
−. 304372247E+01	. 106996685E+01	−. 327313664E+00
. 853465422E−01	−. 183749442E−01	. 310616169E−02
−373261264E−03	. 253873722E−04	

2. 带通滤波器的实现

图 7－23 所示的 IIRBP. ASM 汇编程序在 DSP56001 处理器上实现所设计的带通 IIR 滤波器。注意此程序有一“include”语句，可参考图 7－19 所示的 IIR. ASM 汇编程序宏指令。

为证明在 DSP 系统上设计的带通 IIR 滤波器，应执行以下操作：

（1）将函数发生器连接到 EVB 上的“BNC2”和双踪示波器的通道 1。

（2）使函数发生器产生一 2V（峰到峰）的正弦波。

（3）将示波器通道 2 连接到 EVB 上的“BNC1”。

（4）装入“ADS56000 用户接口软件”程序。

（5）把“DSP56002 示范软件＃3”软盘放入驱动器“A”。

（6）键入 a：iirbp. lod<return>，装入 iirbp. lod 程序。

（7）键入 go<return>，在 DSP56001 处理器上执行所设计的带通 IIR 滤波器程序。

(8) 使正弦波输入频率从 0.5kHz 调到 20kHz。在示波器上观察模拟输入和输出信号。

(9) 键入 force b<return>，停止执行 IIR 滤波器算法，并将 ADM 的控制返回到监视程序。

(10) 键入 quit<return>，退出 ADS56000 程序。

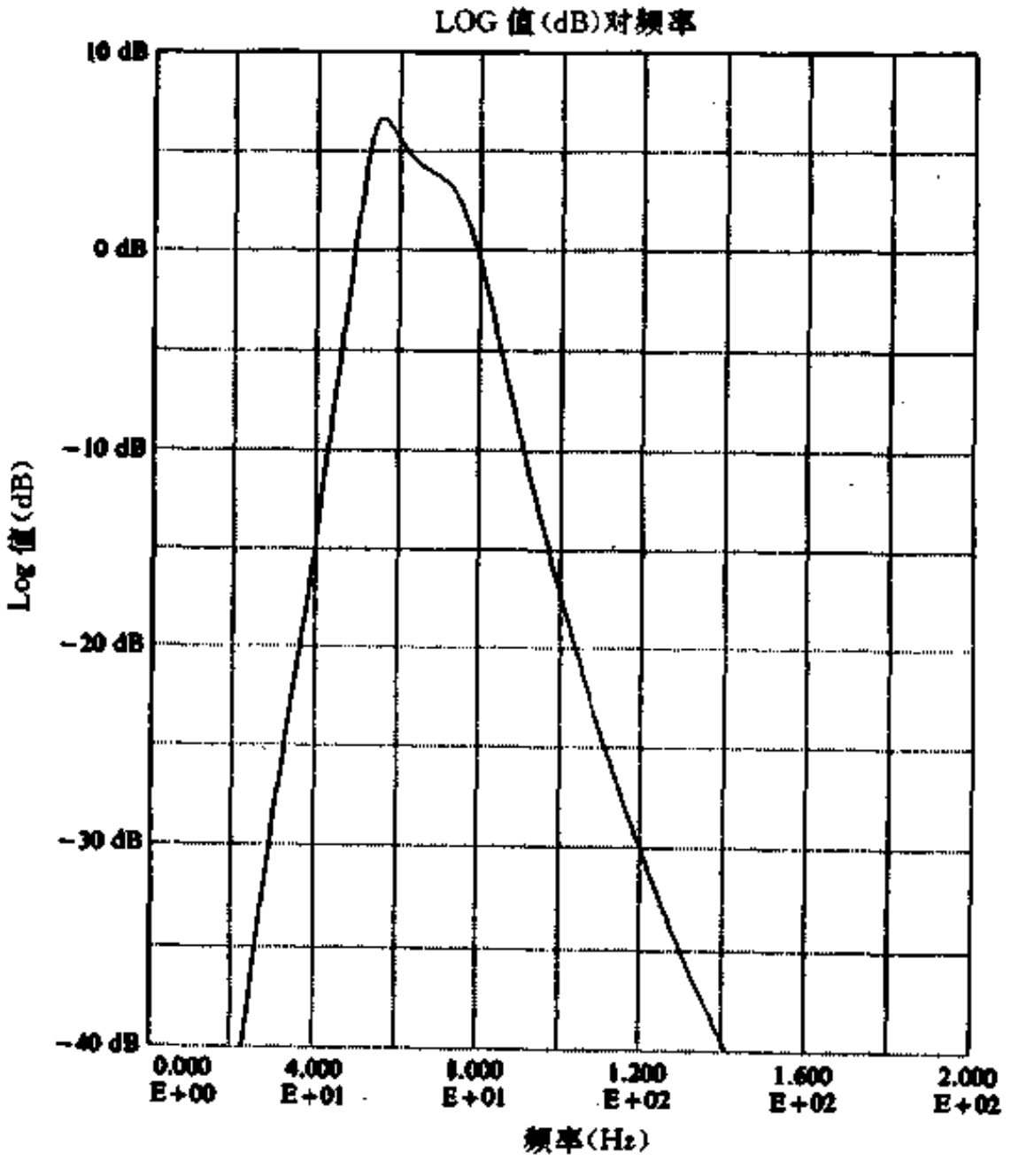
```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; File: IIRBP. ASM
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
        include 'iir'
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Coefficients of Band Pass IIR filter
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
nsec    equ         $e
;
        org         x: $0
states  ds          2*nsec
;
        org         x: $20
gk      dc          .25252064E-06
;
        org         y: $0
coef
        dc -.37573269/2.0
        dc  .49642354/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.38826931/2.0
        dc  .35258451/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.39783910/2.0
        dc  .64982742/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.43422428/2.0
        dc  .22708425/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.44950014/2.0
        dc  .80059147/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
```

```
        dc -.50989538/2.0
        dc  .12285886/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.52194512/2.0
        dc  .94050097/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.60831743/2.0
        dc  1.0678238/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.61304253/2.0
        dc  .40108599E-1/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.70551580/2.0
        dc  1.1843983/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.74407536/2.0
        dc -.20655140E-1/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.81340307/2.0
        dc  1.2929184/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.90595818/2.0
        dc -.56899894E-1/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
        dc -.93392164/2.0
        dc  1.3957597/2.0
        dc -1.0000000/2.0
        dc  .00000000/2.0
;
; program start
;
        org         p:  $200
;
        iir         states, coef, gk, nsec
;
        end
```

图 7-23　实现带通 IIR 滤波器的 IIRBP.ASM 汇编程序

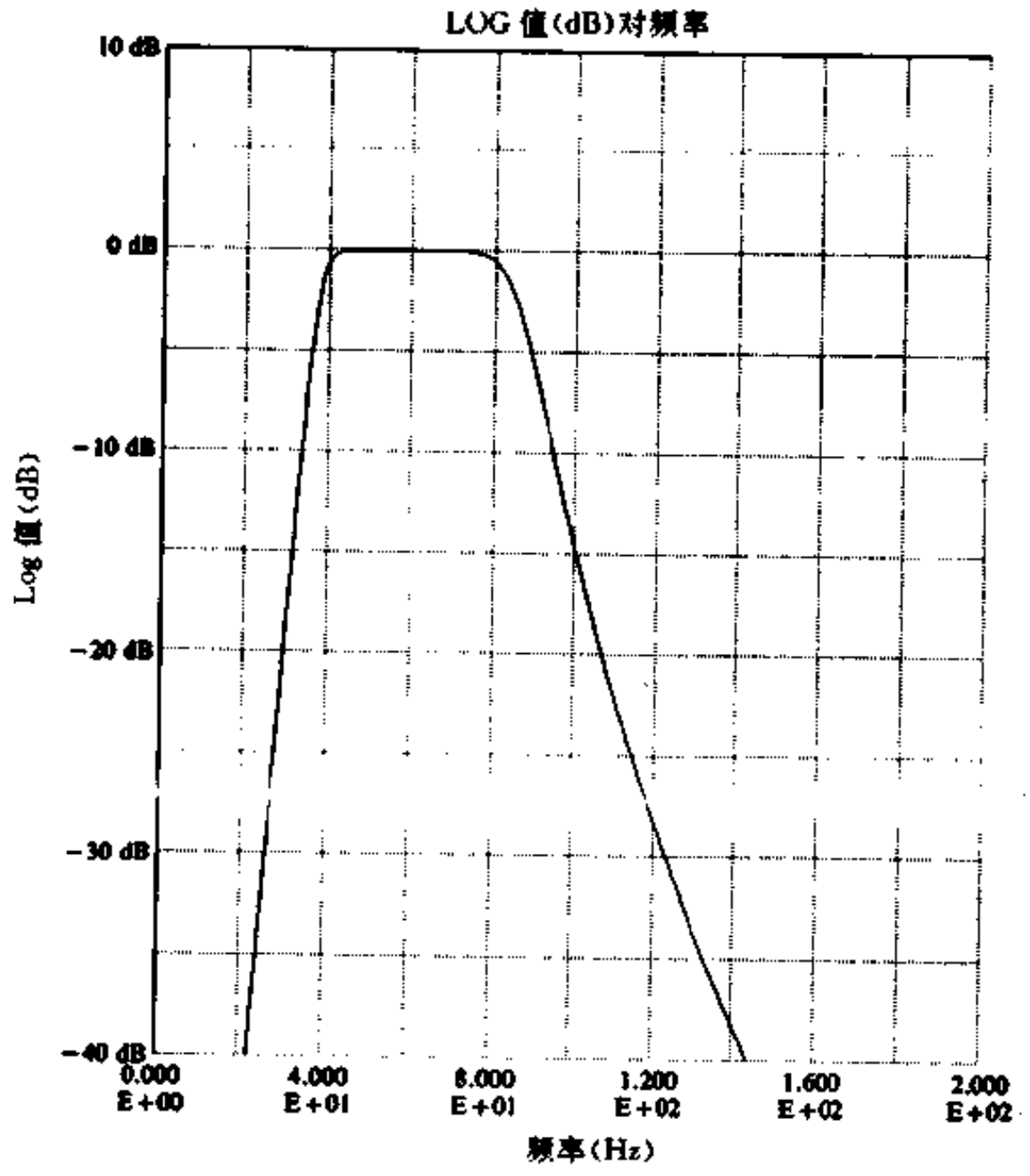
7.10 量化效应

滤波器设计和分析系统(FDAS)亦可用于比较量化效应，即数字信号处理器的字长精度。图 7－24 表示用 24 位 DSP56001 处理器和 16 位处理器实现的 10 阶 50Hz 带通 IIR 滤波器的频率响应。很明显，用 24 位 DSP56001 处理器实现的滤波器比用 16 位处理器实现的滤波器性能为好。



(a)

图 7－24 (a)　16 位处理器实现的滤波器的频率响应



(b)

图 7-24 (b)　24 位处理器实现的滤波器的频率响应

第八章　用 DSP56001 处理器实现快速傅里叶变换

离散傅里叶变换（DFT）广泛用于数字信号处理中，以决定信号的频率谱和完成信号在频域的滤波。快速傅里叶变换（FFT）是计算 DFT 的一种有效快速算法。FFT 达到高速计算是在于按一种有规则的状态，把数据采样集分解为更小的子集，以消除多余的乘法计算。两种常用的分解方法是：

（1）时间抽取 FFT 算法。

（2）频率抽取 FFT 算法。

DSP56000/1 处理器指令集本身对实现时间抽取的 FFT 算法特别好。频率抽取的 FFT 算法也可在 DSP56000/1 处理器上实现，但执行速度比时间抽取法稍慢。

本章开始复习时间抽取 FFT 算法。然后在 DSP56001 处理器上实现。DSP56001 汇编宏指令用于产生“波纹”因子，按比特倒置次序存贮输入序列，实现 FFT 蝶形计算组和系数偏置，并完成全部 N 点 FFT。DSP56001FFT 汇编宏指令可在 DSP56000/1 示范软件 #3 软盘上找到。这些宏指令也可用于实现谱分析算法。

8.1　FFT 的回顾

离散傅里叶变换（DFT）可定义为：

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{8-1}$$

其中

$$W_N = e^{-j2\pi/N} \tag{8-2}$$

和

$$W_N^{nk} = e^{-j2\pi nk/N} \tag{8-3}$$

对长度为 N 的实的或复的离散时间序列 x(n)，(8.1) 式中的和给出长度为 N 的复频率 X(k)。W_N^{nk} 已知为“波纹”因子或叫加权因子。在本章中，N 都认为是 2 的整数幂，即 $N = 2^m$，其中 m 是正数。

FFT 可用时间抽取算法把取样集分解为较小的子集。首先把 DFT 和式中的项分成两组：一组包含 n 为偶数的项，一组包含包含 n 为奇数的项。对偶数项令 n = 2r，对奇数项令 n = 2r + 1，则 DFT 可写为：

$$\begin{aligned} X(k) &= \sum_{r=0}^{(N/2-1)} x(2r) W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x(2r+1) W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x(2r) W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1) W_N^{2rk} \end{aligned} \tag{8-4}$$

因为 N 是偶整数以及 $W_N^2 = W_{N/2} = e^{-j4\pi/N}$，(8-4) 式可写为：

$$X_{l,m}(k) = X_{l,m-1}(k) + W_N^k X_{2,m-1}(k) \tag{8-5}$$

其中

$$X_{l,m}(k) = X(k) \tag{8-6}$$

$$X_{1,m-1}(k) = \sum_{r=0}^{(N/2)-1} X(2r)W_{N/2}^{rk} \tag{8-7}$$

$$X_{2,m-1}(k) = \sum_{r=0}^{(N/2)-1} X(2r+1)W_{N/2}^{rk} \tag{8-8}$$

而其中 $X_{1,m-1}(k)$ 和 $X_{2,m-1}(k)$ 每个都是 N/2 点 DFT。

$X_{1,m-1}(k)$ 和 $X_{2,m-1}(k)$ 可按(8-5)式合并，其信号流图示于图 8-1 以提供 x(n) 的 N 点 DFT。

类似地，二个 N/2 点 DFT 可写成以下形式：

$$X_{1,m-1}(k) = X_{1,m-2}(k) + W_N^{2k}X_{2,m-2}(k) \tag{8-9}$$

$$X_{2,m-1}(k) = X_{3,m-2}(k) + W_N^{2k}X_{4,m-2}(k) \tag{8-10}$$

其中 $X_{1,m-2}(k)$、$X_{2,m-2}(k)$、$X_{3,m-2}(k)$ 和 $X_{4,m-2}(k)$ 每个都是 N/4 点 DFT 。

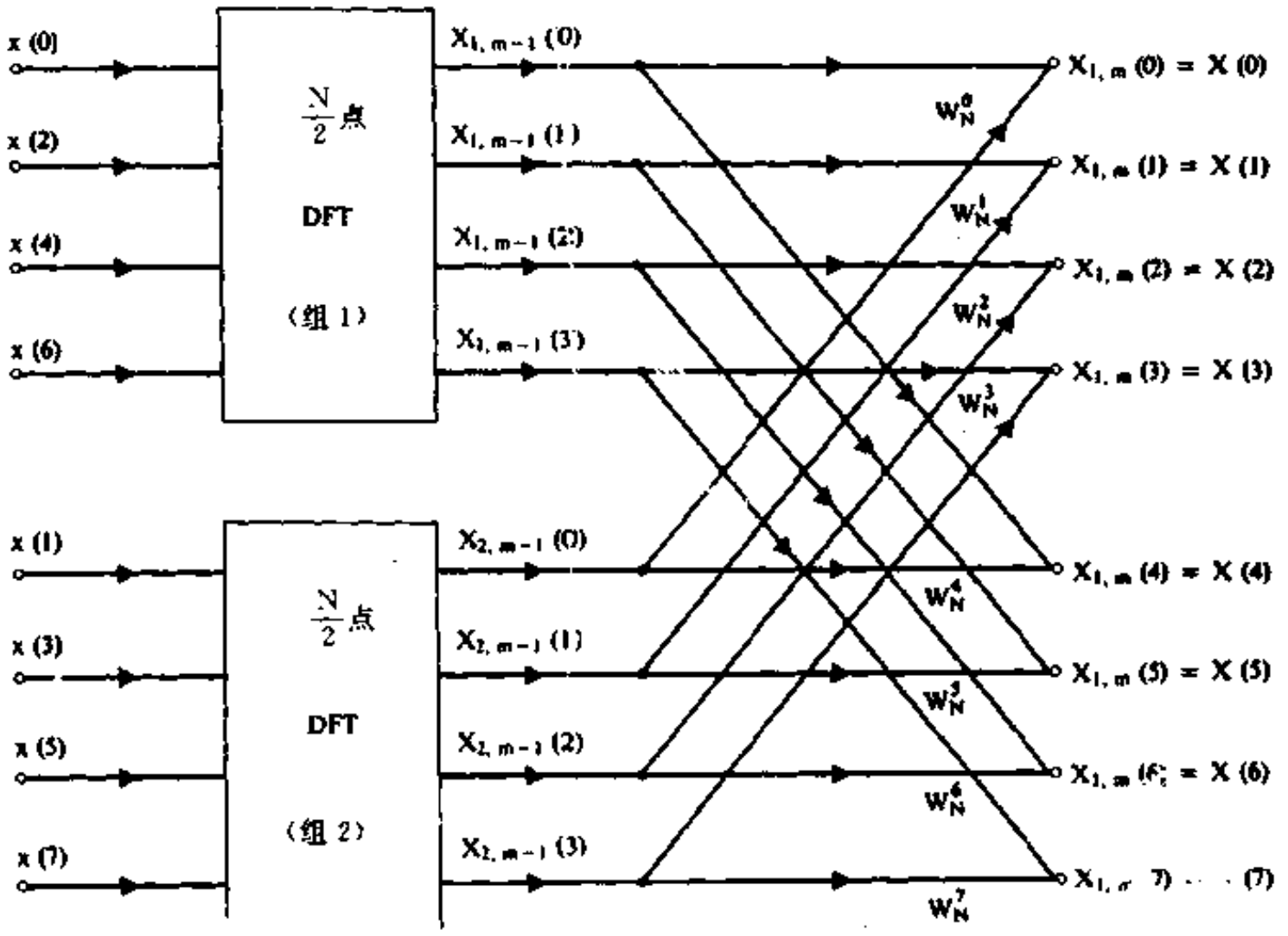


图 8-1 时间抽取法分解 N 点 DFT 为二个 N/2 点 DFT

$X_{1,m-2}(k)$、$X_{2,m-2}(k)$、$X_{3,m-2}(k)$ 和 $X_{4,m-2}(k)$ 可按(8-9)、(8-10)和(8-5)式合并，其信号流图示于图 8-2 以提供 x(n) 的 N 点 DFT。

分解过程继续进行到 N/2 个两点 DFT 组，总的级数 m 等于 $\log_2(N)$，即 $2^m = N$。N = 8 输入样值、N/2 = 4 组和 m = 3 级的流图示于图 8-3，图中两条或更多直线相交处的实心点表示一存贮器单元，用于存贮输入数据样值、中间计算结果，或频域输出样值。

8.1.1 蝶形计算

图 8-3 中所示 DFT 的各级的每个组常用的计算是以图 8-4 所示的基 2 FFT 蝶形为基础。一级中的二个复值由前级中两个复值导出，按以下公式：

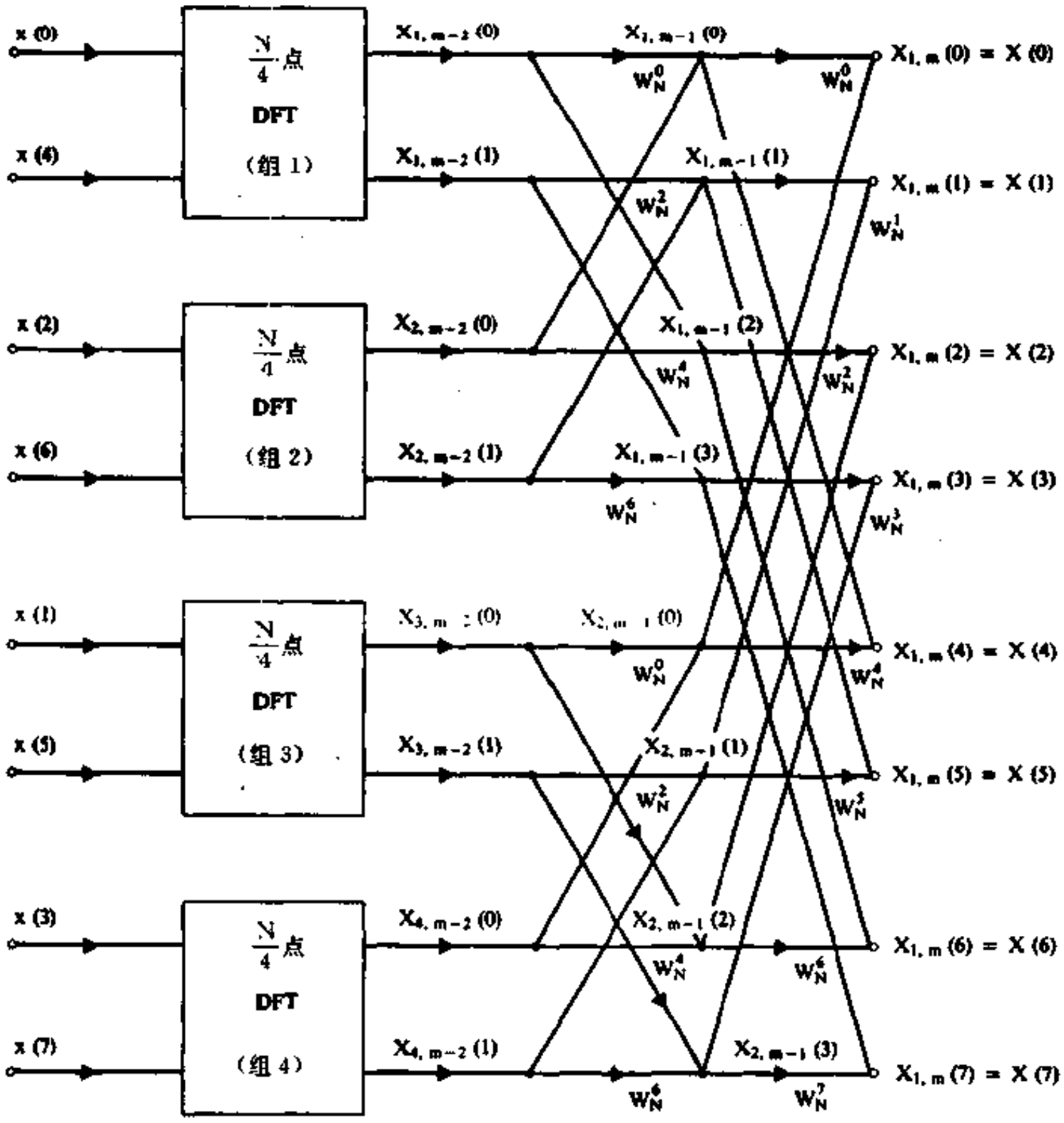


图 8-2　N 点 DFT 分解为 (N/4) DFT 项

$$X_{i,j}[p] = X_{i,j-1}[p] + W_N^r X_{i,j-1}[q] \tag{8-11}$$

$$X_{i,j}[q] = X_{i,j-1}[p] + W_N^{r+N/2} X_{i,j-1}[q] \tag{8-12}$$

其中 i 是组序号，j 是级序号，p 和 q 是对蝶形的 RAM 数据单元。

i、p、q 和 r 的值从级到级的变化如图 8-3 所示。注意 $X_{i,j}[p]$ 和 $X_{i,j}[q]$ 分别存在与 $X_{i,j-1}[p]$ 和 $X_{i,j-1}[q]$ 相同的 RAM 单元。所以，这种“原位”FFT 蝶形计算减少了计算 DFT 所需的 RAM 单元总数。

蝶形系数 W_N 示于图 8-4，它总是 r 或 r+N/2 的幂。因为，$W_N^{(r+N/2)} = W_N^r W_N^{N/2} = e^{-jn} W_N^r = -W_N^r$，图 8-4 的蝶形可简化为图 8-5 所示的蝶形。(8-11) 和 (8-12) 式可写为：

$$X_{i,j}[p] = X_{i,j-1}[p] + W_N^r X_{i,j-1}[q] \tag{8-13}$$

$$X_{i,j}[q] = X_{i,j-1}[p] - W_N^r X_{i,j-1}[q] \tag{8-14}$$

级号 j 从 1 变到 m。如图 8-3 所示，每级 j 有 2^{m-j}组，而每组 i 在每级 j 内有 2^{j-1}个蝶形。在一个蝶形内 RAM 存贮器单元 p 和 q 间的距离称作组的偏离 n_0，n_0 等于 2^{j-1}。级 j 的 i 组中系数指数 r 值之间的差称作系数偏离 n_2，这里 n_2 等于 2^{m-j}。

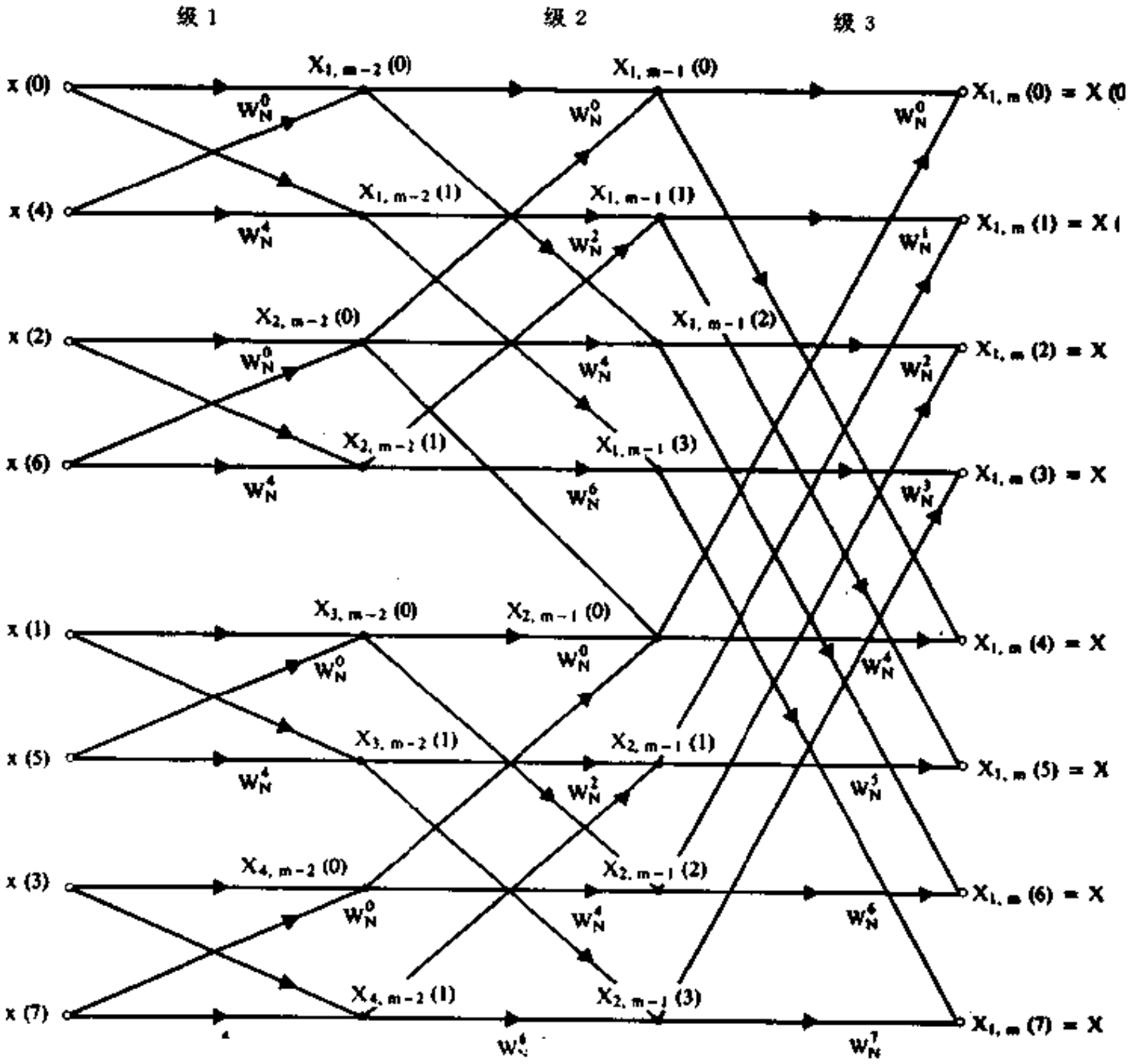


图 8－3　N＝8 点 DFT 的时间抽取分解

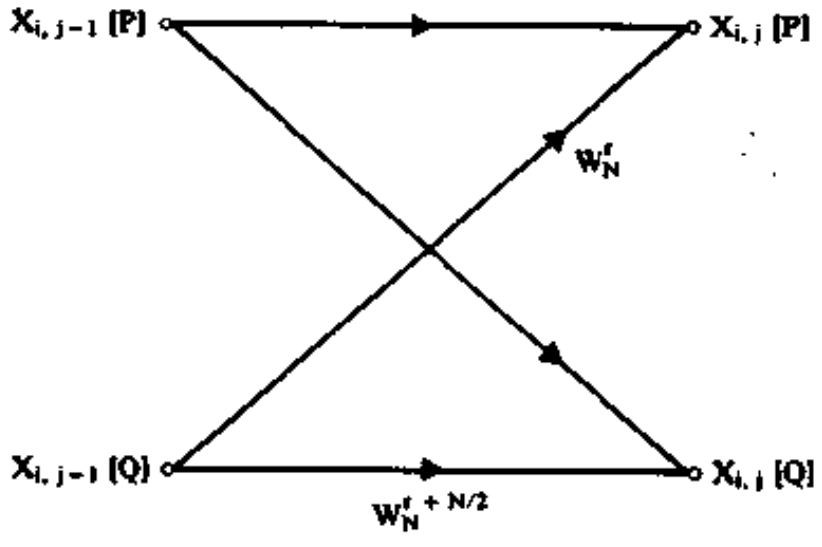


图 8－4　(8－11) 和 (8－12) 式的蝶形表示

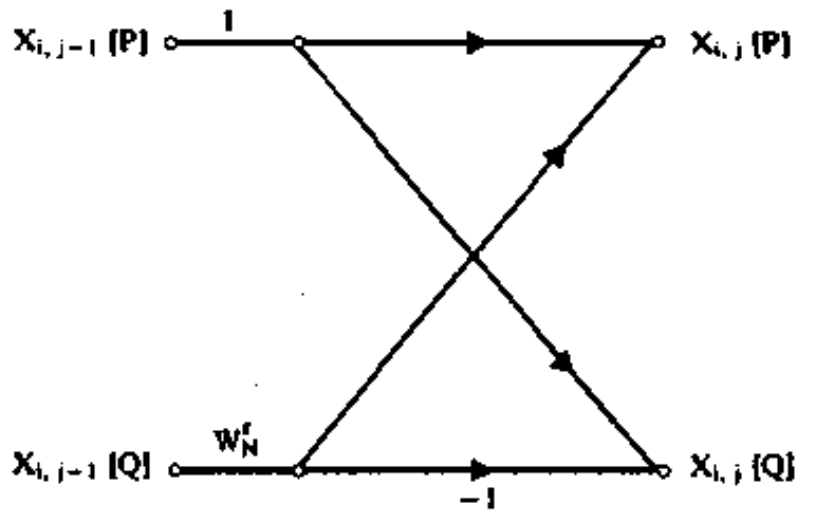


图 8－5　表示(8－13)和(8－14)式的简化蝶形

组偏离　　$n_0=2^{j-1}$　　$j=1, 2, \cdots, m$

系数偏离　　$n_2=i=2^{m-j}$　　$j=1, 2, \cdots, m$　　(8-15)

在计算 DFT 中，j 从 1 递增到 m，i 和 n_2 从 2^{m-1}递减到 2^0，同时 n_0 从 2^0 递增到 2^{m-1}。

8.1.2 输入数据的“位倒置”重新排序

为了完成 8.1.1 节所述的“原位”计算，输入序列 x(n) 的元素必须按特定方式重新排序，并存于连续的RAM单元。为了说明重排技术，考虑输入序列x(n)的每个元素有一个相关的二进制地址，第一个输入元素 x(0) 的地址称为基地址 L，并设 L 的 m 个最低有效位是 0，对 8 点输入序 列：

x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7)

相关的二进制地址的 3 个最低有效位是：

000, 001, 010, 011, 100, 101, 110, 111

为了重排输入序列，每个地址的 m 个最低有效比特必须按表 8-1 进行“位倒置”。

表 8-1　输入序列按位倒置次序

正常次序输入序列	地址的 LSB	地址倒位	从基地址 L 的位移	位倒置输入序列
x (0)	000	000	0	x (0)
x (1)	001	100	4	x (4)
x (2)	010	010	2	x (2)
x (3)	011	110	6	x (6)
x (4)	100	001	1	x (1)
x (5)	101	101	5	x (5)
x (6)	110	011	3	x (3)
x (7)	111	111	7	x (7)

8.2　FFT 的实现

用以下 DSP56001 程序在 DSP56001 处理器上实现 FFT：

1. DSP56001 汇编宏指令产生“波纹”因子。

2. DSP56001 汇编宏指令完成位倒置重排输入序列 x(n) 在 8.2.2 节中研究。

3. 三个不同过程在 DSP56001 处理器上实现 FFT 蝶形运算在 8.2.3 节中研究。

4. 用 DSP56001 处理器计算 FFT 组的偏离和计算系数的偏离在 8.2.4 节中讨论。

5. 8.2.5 节研究了三种不同的 DSP56001 汇编宏指令，每种实现一个完整的 N 点 FFT。

8.2.1　产生“波纹”因子

(8-2) 式中的“波纹”因子可写为：

$$W_N^k = e^{-j2\pi k/N} = \cos(2\pi k/N) - j\sin(2\pi k/N)$$

其中　　$k = 0,1,\cdots,(N-1)/2$　　(8-16)

示于图 8－6 的 DSP56001 汇编宏指令 SINCOS. ASM 产生“波纹”因子的查找表。(8－16) 式所产生的实部值取负 (－cos (2πk/N)) 存在 X 存贮器中；虚部值 [－sin (2πk/N)] 存在 Y 存贮器中。每个表分别包括－cos 和 sine 波形的半周。

建在 ASM56000 汇编软件程序中的 sine 和 cosine 超越函数由汇编程序宏指令用来计算 sine 和 cosine 表值。因为 sine 和 cosine 值的范围从－1 到＋1，并因为 DSP56001 处理器有小数算术范围从－1 到 $1-2^{-23}$ (没有＋1)，故可选用 cos (0) 的负值代替而避免 cos (0) ＝＋1。

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Sine - Cosine Table Generator for FFTs
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
sincos    macro    points, coef
;
;
;
;    points - unmber of points
;    coef   - base address of sine/cosine table
;           - negative cosine value in X memory
;           - negative sine value in Y memory
;
;
pi        equ      3.141592654
freq      equ      2.0 * pi/@cvf (points)    ; freq=2pi/points

          org      x: coef           ; Calculation of
count     set      0                 ; -cos (frep * count) for
          dup      points/2          ; count=0..., (points/2) -1

          dc       -@cos (@cvf (count) * freq)
count     set      count+1
          endm

          org      y: coef           ; Calculation of
count     set      0                 ; -sin (freq * count) for
          dup      points/2          ; count=0..., (points/2) -1

          dc       -@sin (@cvf (count) * freq)
count     set      count+1
          endm

          endm                       ; end of sincos macro
```

图 8－6　产生 sine 和 cosine 查找表的 SINCOS. ASM 汇编程序指令

例 8.1：执行程序“SINEX. LOD”对 8 点 FFT 产生 sine 和 cosine 查找表的值。

此例中，X 和 Y 存贮器中 cosine 和 sine 查找表的基地址选择等於 FFT 的大小 N＝8。所以－cos (2#k/8) 对 k＝0，1，2，3 的表值存入 X 存贮器单元 $8～$b 中，－sin (2πk/8) 对 k＝0，1，2，3，的表值存入 Y 存贮器单元 $8～$b 中。SINEX. ASM 汇编程序示于图 8－7。

1. 装入“SIM56000 示范模拟软件”程序。

2. 把“DSP56000/1 示范软件#3”软盘插入驱动器“A”。

3. 键入 SIM56000 命令：

```
display off<return>
load a: sinex<return>
step<return>
radix f x: $8... $b y: $8.. $b<return>
display x: $8.. $b<return>
display y: $8.. $b<return>
```

```
;
; Twiddle factors for 8 - point FFT
;
;
points          equ          $8
coef            equ          $8

; points - number of points
; coef   - base address of sine/cosine table
; negative cosine value in X memory
; negative sine value in Y memory
;
;
        include 'sincos'
        sincos          points, coef
        end
```

图 8-7　SINEX. ASM 汇编程序

4. 键入 quit<return>，退出 SIM56000 程序。

执行 SINEX. LOD 程序后，在 cosine 和 sine 查找表中的值是：

X：$0008　−1.0000000　−0.7071068　0.0000000　0.7071068

Y：$0008　0.0000000　−0.7071068　−1.0000000　−0.7071068

8.2.2　按位倒置次序存入输入序列 x(n)

DSP56001 处理器地址产生单 元（AGU）有一种自动完成反进位（位倒置）算术的寻址方式。当用此寻址方式来计算 FFT 时，DSP56001 处理器会自动选择适当的输人（或输出）序列元素，所以增加了 FFT 的执行速度。

为了充分说明反进位算术概念，本节将用 DSP56001 处理器的 AGU 以首先重排输入序列 x(n)，然后存入重排结果。反进位可用于完成存贮器地址 m 个最低有效位的倒置修正。这可以加偏移 $N = 2^{m-1}$ 到地址值上，然后以反方向传任何所产生的进位项，即进位项沿最低有效位的方向传输。

示于图 8~8 的 BITREV. ASM 汇编程序宏指令按上述位倒置重排输人序列 x(n)，并存人重排结果。设输人序列原来是正常次序，它的实部存在 X 存贮器单元 L～L＋N－1，而其虚部存在 Y 存贮器单元 L～L＋N－1，这里 L 是基地址。L 的 m 个最低有效位必须是 0。执行汇编宏指令后，重排的输人序列实部值 存在 X 存贮器单元 0～N－1，而其虚部存在 Y 存贮器单元 0～N－1，因为 sine 表存在 X 存贮器单元 N～3N/2－1 处，而 cosine 表存在 Y 存贮器 N～3N/2－1 处，则 L 可大于或等于 3N/2。

```
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Storing Input Data in Bit - Reversed Order
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

bitrev  macro    points, data, olddata

;
;       points   =Number of points
;       olddata  =Input data in normal order
;       data     Input data in bit - reversed order
;
;

        move     #olddata, r1        ; initialize the
        move     r1, r6              ; pointers
        move     #data, r0
        move     r0, r5

        move     #0, m1              ; m1 and m6=bit
        move     m1, m6              ; reverse
        move     #-1, m0             ; m5 and m6=linear
        move     m0, m5

        move     #points/2, n1
        move     n1, n6

        do       #points,_end_reverse
        move     x:(r1)+n1, a    y:(r6)+n6, b
        move     a, x:(r0)+      b, y:(r5)+

_end_reverse

        endm
```

图 8-8　以位倒置次序存贮输入序列的 BITREV. ASM 汇编宏指令

BITREV. ASM 汇编程序宏指令用 DSP56001 处理器以反进位算法偏离 N 后递增地址，因此在地址上完成位倒置修正。输入序列正常次序元素的基地址 L 开始存在 AGU 寄存器 R1 和 R6 中，其中 R1 指示 X 存贮器，R6 指示 Y 存贮器。输出序列重排的元素基地址开始存在 R0 和 R5 中，其中 R0 指示 X 存贮器，R5 指示 Y 存贮器。R1 和 R6 都是用反进位算法后递增偏移 N。寄存器 N1 和 N6 以偏移值 $N=2^{m-1}$ 编程；寄存器 M1 和 M6 以值 $0000 编程以选择反进位算法。R0 和 R5 都是以线性算法后递增 1，线性算法由以 $ffff 编程的 M0 和 5 选择。汇编 程序句法本身定义后递增 1 的条件，因此 N0 和 N5 不需要编程。表 8-2 表示对输入序列元素 0.0，0.1，0.2，0.3，0.4，0.5，0.6，0.7 的反进位地址修正，其中 N=8，L=$18。

例 8.2：执行程序 BITEX. LOD，按位倒置次序存贮 8 点输入序列。

此例中，原始输入序列为正常次序，其实部存在 X 存贮器单元 $18～$1f，而其虚部存在 Y 存贮器单元 $18～$1f 中。执行程序 BITEX. LOD 以后，输入序列按位倒置次序，其实

表 8-2　对 N=8 和 L=$18 的地址修正

正常次序的输入序列	地址修正	位倒置的输入序列
X: $18　0.0	00011000=18	x: $0　0.0
	+----1---	
X: $19　0.1	00011100=1C	X: $1　0.4
	+----1---	
X: $1A　0.2	00011010=1A	X: $2　0.2
	+----1---	
X: $1B　0.3	00011110=1E	X: $3　0.6
	+----1---	
X: $1C　0.4	00011001=19	X: $4　0.1
	+----1---	
X: $1D　0.5	00011101=1D	X: $5　0.5
	+----1---	
X: $1E　0.6	00011011=1B	X: $6　0.3
	+----1---	
X: $1F　0.7	00011111=1F	X: $7　0.7

```
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; 8 - point Bit - Reversed Example
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

            include 'bitrev'

;       Input Data
;       points   = Number of points
;       olddata  = Input data in normal order
;       data     = Input data in bit - reversed order
;

data        equ     $0              ; Begin of data
points      equ     $8              ; FFT size
olddata     equ     $18

            org     x: $18          ; Begin of olddata
olddata                             ; (real part)
            dc      0.0, 0.1, 0.2, 0.3
            dc      0.4, 0.5, 0.6, 0.7

            org     y: $18          ; Begin of olddata
olddataI                            ; (imaginary part)
            dc      0.0, 0.1, 0.2, 0.3
            dc      0.4, 0.5, 0.6, 0.7

; program start
            org     p: $100
            bitrev  points, data, olddata
            end
```

图 8-9　BITEX. ASM 汇编程序

部存在 X 存贮器单元 $0～$7 中，而虚部存在 Y 存贮器单元 $0～$7 中。BITEX. ASM 程序示于图 8-9。其中对 BITREV. ASM 的“include”语句示于图 8-8 中。

1. 装入“SIM56000 模拟软件”程序。

2. 把“DSP56000/1 示范软件＃3”软盘插入驱动器“A”。

3. 键入 SIM56000 命令：

```
display off<return>
load a: bitex<return>
break #1 pc>$10f<return>
go #1<return>
radix f x: $18..$1f<return>
display x: $18..$1f<return>
radix fy: $18..$1f<return>
display: $18..$1f<return>
radix f x: $0 ..$7<return>
display x: $0 ..$7<return>
radix f y: $0 ..$7<return>
display y: $0 ..$7<return>
```

正常次序的输入序列

实部：

X：$0018 0.0000000 0.1000000 0.2000000 0.3000000

X：$001C 0.4000000 0.5000000 0.6000000 0.7000000

虚部：

Y：$0018 0.0000000 0.1000000 0.2000000 0.3000000

Y：$001C 0.4000000 0.5000000 0.6000000 0.7000000

位倒置次序的输入序列

实部：

X：$0000 0.0000000 0.4000000 0.3000000 0.6000000

X：$0004 0.1000000 0.5000000 0.3000000 0.7000000

4. 键入 quit<return>，退出 SIM56000 程序。

8.2.3 蝶形计算

(8－13) 和 (8－14) 式用于计算图 8－5 所示的蝶形。若 $X_{i,j-1}[p]$，$X_{i,j-1}[q]$，$X_{i,j}[p]$，$X_{i,j}[q]$ 和 W_N^r 的实、虚部是：

$$X_{i,j-1}[p]=ar+jai$$

$$X_{i,j-1}[q]=br+jbi$$

$$X_{i,j}[p]=ar'+jai'$$

$$X_{i,j}[q]=br'+jbi'$$

$$W_N^r=wr+jwi \tag{8-17}$$

则蝶形的输出 ar′、ai′、br′和 bi′可由 (8－13)、(8－14) 和 (8－17) 式计算如下：

$$ar'=ar+wr*br-wi*bi$$

$$ai'=ai+wi*br+wr*bi$$

$$br'=ar-wr*br+wi*bi=2ar-ar'$$

$$bi'=ai-wi*br-wr*bi=2ai-ar' \tag{8-18}$$

蝶形计算之后，ar′和 br′存在 X 存贮器中，而 ai′和 bi′存在 Y 存贮器中。

因为（8－13）和（8－14）式所用波纹因子形式为 $e^{-j2\pi r/N}$，其幅值为 1，$X_{i,j}[p]$ 和 $X_{i,j}[q]$ 的幅值每级可增长的最大因子为 2。对 N 点 FFT，其幅值可增长 N。因此，$X_{i,j}[p]$ 和 $X_{i,j}[q]$ 的幅值必须保持小于 1，以使存贮时没有溢出或限幅。这可用以下三个过程之一来完成：

1. 限制输入序列元素的幅值为 1/N，这里的幅值按实部和虚部平方和的平方根计算。

2. FFT 内全部蝶形输出自动缩小 0.5 倍。FFT 输入数据的幅值 小于 1。若要求绝对结果，必须加共用定标因子 N 到 FFT 的结果上。可以选择 DSP56001 处理器自动以 0.5 因子缩小数据。这可分别以它的状态寄存器的位 10 和 11 编程为 0 和 1 来完成。

3. 用方块浮点技术完成有条件定标。这种方法允许全部 FFT 级有选择地定标，视发生在前一级幅值 的增长而定。这种方法可最有效地实现，只需在全 CMOS 的 DSP56001 处理器的状态寄存器（位 7）中用定标位 S。当 DSP56001 处理器传送到存贮器的结果绝对值 大于 0.25 时，设置定标位。FFT 算法在每个 FFT 级完成时测试定标位。若定标位被设置，则 DSP56001 处理器的自动定标方式在下一级之前工作。若没有设置，则没有发生增长且自动定标方式不工作。若要求绝对值，则必须计算定标因子。定标因子等于 2^k。其中 k 是自动加上降标的级的数目。

例 8.3：限制二点蝶形输入值 的大小为 1/N＝0.5。

此例中，执行 BUTER1.LOD 程序以计算二点蝶形的输出值 ar′，ai′，br′和 bi′。输入值的实部和虚部是：ar＝0.4，ai＝0.25，br＝0.1 和 bi＝0.15；“波纹”因子的实和虚部是，wr＝－1 和 wi＝0。实部输入值 ar′和 br′分别存在 X 存贮器单元 $0 和 $1；虚部值 ai′和 bi′分别存在 Y 存贮器单元 $0 和 $1。wr 存在 X 存贮器单元 $2；wi 存在 Y 存贮器单元 $2。

执行 BUTER1.LOD 程序以后，实部输出值 ar 和 br 分别存入 X：$0 和 $1；虚部输出值 ai 和 bi 存入 Y：$0 和 $1。因为在 X 和 Y 存贮器中输出值改写输入值，所以 BUTER1.LOD 程序是原位蝶形算法。BUTER1.ASM 汇编程序示于图 8－10。

1. 装入“SIM56000 模拟软件”程序。

2. 把“DSP56000/1 示范软件＃3”软盘插入驱动器“A”。

3. 键入 SIM56000 命令：

```
change x:0  0.4  y:0  0.25<return>
change x:1  0.1  y:1  0.15<return>
change x:2 -1.0  y:2  0<return>
display off <return>
radix f x:0..1  y:0..1<return>
display x:0..1  y:0..1<return>
load a:buter 1<return>
break #1  pc> $ 11d<return>
go<return>
display x:0..1  y:0..1<return>
```

4. 键入 quit<return>，退出 SIM56000 程序。

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; 2-point FFT
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
passes      equ     $1
points      equ     $2
data        equ     $0
coef        equ     $2

            org     p: $100

            move    #1, n0              ; n0=group offset
            move    #points/2, n2       ; n2=group per pass

            move    #-1, m0             ; n0=m1=m2=m4=m5=m6
            move    m0, m1              ; =linear addressing
            move    m0, m2
            move    m0, m4
            move    m0, m5
            move    m0, m6

            move    n0, n1              ; n0=n1=n4=n5
            move    n0, n4              ; group offset
            move    n0, n5
            move    n2, n6              ; n6=coefficient offset

            move    #data, r0           ; r0=ar, ai input pointer
            move    r0, r4              ; r4=ar', ai' output pointer
            move    #coef, r6           ; r6=wr, wi input pointer

            nop
            lua     (r0) +n0, r1        ; r1=br, bi input pointer
            nop
            move    r1, r5              ; r5=br', bi' output pointer
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; 2-point FFT
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    move                                y: (r0), b      ; b=ai

    move                x: (r1), x1     y: (r6), y0     ; x1=br
                                                        ; y0=wi
    mac x1, y0, b       x: (r6), x0     y: (r1), y1     ; b=ai+br*wi
                                                        ; x0=-wr
                                                        ; y1=bi

    macr -x0, y1, b                     y: (r0), a      ; b=ai+br*wi+bi*wr
                                                        ; =ai'
                                                        ; a=ai

    subl b, a           x: (r0), b      b, y: (r4)      ; a=2ai-ai'=bi'
                                                        ; b=ar
                                                        ; y (r4) =ai'

    mac -x0, x1, b      x: (r0), a      a, y: (r5)      ; b=ar+br*wr
                                                        ; a=ar
                                                        ; y (r5) =bi'

    macr -y0, y1, b                                     ; b=ar+br*wr-bi*wi
                                                        ; =ar'
```

```
subl b, a          b, x: (r4)          ; a=2ar-ar'=br'
                                       ; x: (r4) =ar'

move               a, x: (r5)          ; x: (r5) =br'
end
```

图 8-10　BUTER1.ASM 汇编程序

蝶形输入：

X：$0000　0.4000000　0.1000000

Y：$0000　0.2500000　0.1500000

蝶形输出：

X：$0000　0.5000000　0.3000000

Y：$0000　0.4000000　0.1000000

例 8.4：以因子 0.5 自动减小单级二点蝶形输出。

此例中，执行程序 BUTER2.LOD，计算二点蝶形的输出值 ar′、ai′、br′和 bi′，其中输入值幅度限定小于 1，而输出为一半。输入的实部虚部值是 ar=0.8，ai=0.5，br=0.2 和 bi=0.3。“波纹”因子实虚部分别为 wr=−1 和 wi=0。输入实部存于 X：$0 和 $1，虚部存于 Y：$0 和 $1。wr 存于 X：$2，wi 存于 Y：$2：

执行 BUTER2.LOD 程序后，输出的各相应值仍存在原位。BUTER2.ASM 程序示于图 8-11。注意使用 ORI 指令使之降标工作，用 ANDI 指令使降标不工作。

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; 2-point FFT
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
passes      equ     $1
points      equ     $2
data        equ     $0
coef        equ     $2
            org     p: $100

            move    #1, n0              ; n0=group offset
            move    #points/2, n2       ; n2=group per pass
            move    #-1, m0             ; n0=m1=m2=m4=m5=m6
            move    m0, m1              ; =linear addressing
            move    m0, m2
            move    m0, m4
            move    m0, m5
            move    m0, m6

            move    n0, n1              ; n0=n1=n4=n5
            move    n0, n4              ; group offset
            move    n0, n5
            move    n2, n6              ; n6=coefficient offset

            move    #data, r0           ; r0=ar, ai input pointer
            move    r0, r4              ; r4=ar', ai' output pointer
            move    #coef, r6           ; r6=wr, wi input pointer
```

```
        nop
        lua     (r0) +n0, r1          ; r1=br, bi input pointer
        nop
        move    r1, r5                ; r5=br', bi' output pointer

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; 2-point FFT
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   ori # $4, mr

   move                                y: (r0), b   ; b=ai

   move                x: (r1), x1     y: (r6), y0  ; x1=br
                                                    y0=wi

   mac x1, y0, b       x: (r6), x0     y: (r1), y1  ; b=ai+br*wi
                                                    ; x0=-wr
                                                    ; y1=bi

   macr -x0, y1, b                     y: (r0), a   ; b=ai+br*wi+bi*wr
                                                    ; =ai'
                                                    ; a=ai

   subl b, a           x: (r0), b      b, y: (r4)   ; a=2ai-ai'=bi'
                                                    ; b=ar
                                                    ; y (r4) =ai'

   mac -x0, x1, b      x: (r0), a      a, y: (r5)   ; b=ar+br*wr
                                                    ; a=ar
                                                    ; y (r5) =bi'

   macr -y0, y1, b                                  ; b=ar+br*wr-bi*wi
                                                    ; =ar'

   subl b, a           b, x: (r4)                   ; a=2ar-ar'=br'
                                                    ; x: (r4) =ar'

   move                a, x: (r5)                   x: (r5) =br'
   andi # $fb, mr
   end
```

图 8-11　Buter2.ASM 汇编程序

1. 装入“SIM56000 模拟软件”程序。
2. 把“DSP56000/1 示范软件#3”软盘插入驱动器“A”。
3. 键入 SIM56000 命令：

```
change x:0   0.8   y:0   0.5<return>
change x:0   0.2   y:1   0.3<return>
change x:2   -1.0  y:2   0<return>
diaplay off <return>
radix f x:0..1   y:0..1<return>
display x:0..1   y:0..1<return>
```

load a:buter2<return>

break #1 pc>$11f<return>

go<return>

display x:0..1 y:0..1<return>

4. 键入 quit <return>,退出 SIM56000 程序。

蝶形输入:

X:$0000 0.8000000 0.2000000

Y:$0000 0.5000000 0.3000000

蝶形输出:

X:$0000 0.5000000 0.3000000

Y:$0000 0.4000000 0.1000000

8.2.4 组和系数偏移

图 8-12 的 PASSEX.ASM 汇编程序计算组偏移 n_0 和系数偏移 n_2,其中 n_0 和 n_2 由(8-15)式定义。

```
;
;
;     passes=unmber of passes
;     points=FFT size
;
passes      equ     $3
points      equ     $8

            move    #1,n0               ; n0=1,2,4
            move    #points/2,n2        ; n2=4,2,1

            move    #-1,m0              m0=m1=linear
            move    m0,m2

            do      #passes,_end_pass

            move    n2,b1
            lsr     b                   n0,a1
            lsl     a                   b1,n2
            move    a1,n0

  _end_pass

            end
```

图 8-12 PASSEX.ASM 汇编程序

例 8.5:此例中,PASSEX.ASM 程序用来计算三级 8 点 FFT 的组和系数偏移。

1. 装入"SIM56000 模拟软件"程序。
2. 把"DSP56000/1 示范软件#3"软盘插入驱动器"A"。
3. 键入 SIM56000 命令:

display off <return>

display n0 n2<return>

load a :passex<return>

step 2<return>

```
display n0 n2 <return>
step 7 <return>
display n0 n2 <return>
step 7 <return>
display n0 n2 <return>
step 4<return>
display n0 n2<return>
```

4. 键入 quit<return>，退出 SIM56000 程序。

三级 FFT 的 n_0 和 n_2 分别为：

第一级　　$n_0=1$　　$n_2=4$

第二级　　$n_0=1$　　$n_2=2$

第三级　　$n_0=4$　　$n_2=1$

8.2.5 FFT 宏指令

FFT 的宏指令 FFT1.ASM，FFT2.ASM 和 FFT3.ASM 每个都可在复输入序列上完成全部 FFT。三个宏指令分别示于图 8-13、8-14 和 8-15。宏指令可用以下变量调用：

1. FFT 的点数。

2. 正常次序或倒位次序输入序列的单元。

3. 正-余弦表单元。

所有 DSP56001 寄存器初始化在宏指令内完成，并且输出序列是正常次序。在 FFT1 宏指令中，输入序列的元素幅值限制到 1/N。在 FFT2 宏指令中，FFT 蝶形的全部输出按比例因子 2 下降。在 FFT3 中，输入序列元素幅值限制到 0.5，块浮点方法用来实现有条件定标，并有一存贮器单元用来存贮 DSP56001 处理器状态寄存器的内容。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Decimation in Time FFT
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

fft1 macro points, data, coef, olddata

;  passes     =number of passes =log2(number of points)
;  points     =number of points
;  coef       =base address of sine/cosine table
;  olddata    =Starting address of input data before bit reverse
;  data       =Starting address of input data after bit reverse

;
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT Initialization
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
        move    #1,n0                   ; n0=group offset
        move    #points/2,n2            ; n2=group per pass
        move    #-1,m0                  ; m0=m1=m2=m4=m5=m6
        move    m0,m1                   ; =linear addressing
        move    m0,m2
        move    m0,m4
        move    m0,m5
        move    m0,m6
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT passes
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        do      #@cvi(@log(points)/@log(2)+0.5),_end_pass

        move    #data,r0                ; r0=ar,ai input pointer
        move    r0,r4                   ; r4=ar',ai'output pointer
        move    #coef,r6                ; r6=wr,wi input pointer

        nop
        lua     (r0)+n0,r1              ; r1=br,bi input pointer
        nop
        lua     (r1)-,r5                ; r5=br',bi' input pointer

        move    n0,n1                   ; n0=n1=n4=n5
        move    n0,n4                   ; group offset
        move    n0,n5
        move    n2,n6                   ; n6=coefficient offset

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT Group
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      do    n2,_end_grp

      move                  x:(r5),a        y:(r0),b

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT Butterfly
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      do    n0,_end_bfy

      move                          x:(r1),x1       y:(r6),y0
      mac       x1,y0,b             x:(r6)+n6,x0    y:(r1)+,y1
      macr      -x0,y1,b            a,x:(r5)+       y:(r0),a
      subl      b,a                 x:(r0),b        b,y:(r4)
      mac       -x0,x1,b            x:(r0)+,a       a,y:(r5)
```

```
    macr    -y0,y1,b    x:(r1),x1
    subl    b,a         b,x:(r4)+       y:(r0),b

_end_bfy

    move    #coef,r6
    move                a,x:(r5)+n5     y:(r1)+n1,y1
    move                x:(r0)+n0,x1    y:(r4)+n4,y1

_end_grp

    move    n2,b1
    lsr     b           n0,a1
    lsl     a           b1,n2
    move    a1,n0

_end_pass

    endm
```

图 8-13　FFT1.ASM 汇编指令

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Decimation in Time FFT
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

fft2 macro points, data, coef, olddata

;   passes      =number of passes =log2(number of points)
;   points      =number of points
;   coef        =base address of sine/cosine table
;   olddata     =Starting address of input data before bit reverse
;   data        =Starting address of input data after bit reverse
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT Initialization
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        move    #1,n0              ; n0=group offset
        move    #points/2,n2       ; n2=group per pass

        move    #-1,m0             ; m0=m1=m2=m4=m5=m6
        move    m0,m1              ; =linear addressing
        move    m0,m2
        move    m0,m4
        move    m0,m5
        move    m0,m6
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT passes
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        do      #@cvi(@log(points)/@log(2)+0.5),_end_pass

        move    #data,r0            ; r0=ar,ai input pointer
        move    r0,r4               ; r4=ar',ai' output pointer
        move    #coef,r6            ; r6=wr,wi input pointer

        nop
        lua     (r0)+n0,r1          ; r1=br,bi input pointer
        nop
        lua     (r1)-,r5            ; r5=br',bi' input pointer

        move    n0,n1               ; n0=n1=n4=n5
        move    n0,n4               ; group offset
        move    n0,n5
        move    n2,n6               ; n6=coefficient offset

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT Group
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      ori       #$4,mr

      do      n2,_end_grp

      move                      x:(r5),a      y:(r0),b
      lal     a

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT Butterfly
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      do        n0,_end_bfy

      move                      x:(r1),x1     y:(r6),y0
      mac       x1,y0,b         x:(r6)+n6,x0  y:(r1)+,y1
      macr      -x0,y1,b        a,x:(r5)+     y:(r0),a
      subl      b,a             x:(r0),b      b,y:(r4)
      mac       -x0,x1,b        x:(r0)+,a     a,y:(r5)
      macr      -y0,y1,b        x:(r1),x1
      subl      b,a             b,x:(r4)+     y:(r0),b

_end_bfy

      move      #coef,r6
      move                      a,x:(r5)+n5   y:(r1)+n1,y1
      move                      x:(r0)+n0,x1  y:(r4)+n4,y1
```

```
_end_grp

    andi    # $fb,mr
    move                    n2,b1
    lsr     b               n0,a1
    lsl     a               b1,n2
    mvoe    a1,n0

_end_pass

    endm
```

图 8-14　FFT2.ASM 汇编宏指令

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Decimation in Time FFT
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

fft3    macro    points, data, coef, olddata, accr

;  passes     =number of passes =log2(number of points)
;  points     =number of points
;  coef       =base address of sine/cosine table
;  olddata    =starting address of input data before bit reverse
;  data       =starting address of input data after bit reverse
;  accr       =address used to store the value of the ccr register
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; FFT Initialization
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        move    #0,r7
        move    #1,n0               ; n0=group offset
        move    #points/2,n2        ; n2=group per pass

        move    #-1,m0              ; m0=m1=m2=m4=m5=m6
        move    m0,m1               ; =linear addressing
        move    m0,m2
        move    m0,m4
        move    m0,m5
        move    m0,m6
        move    m0,m7

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; FFT passes
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        do      #@cvi(@log(points)/@log(2)+0.5),_end_pass
```

```
        move    #data,r0            ; r0=ar,ai input pointer
        move    r0,r4               ; r4=ar',ai'output pointer
        move    #coef,r6            ; r6=wr,wi input pointer

        nop
        lua     (r0)+n0,r1          ; r1=br,bi input pointer

        move    #accr, r3

        nop
        lua     (r1)-,r5            ; r5=br',bi' input pointer
        move    n0,n1               ; n0=n1=n4=n5
        move    n0,n4               ; group offset
        move    n0,n5
        move    n0,n6               ; n6=coefficient offset

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; FFT Group
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        move    sr,x:(r3)
        jclr    #7,x:(r3),noscale
        ori     #$4,mr
        move                (r7)+
noscale

        do      n2,_end_grp
        move                x:(r5),a    y:(r0),b

        jclr    #7,x:(r3),Continue
        lsl     a
Contiune
        andi    #$7f,ccr
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; FFT Butterfly
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        do      n0,_end_bfy

        move                        x:(r1),x1       y:(r6),y0
        mac     x1,y0.b             x:(r6)+n6,x0    y:(r1)+,y1
        macr    -x0,y1,b            a,x:(r5)+       y:(r0),a
        subl    b,a                 x:(r0),b        b,y:(r4)
        mac     -x0,x1,b            x:(r0)+,a       a,y:(r5)
        macr    -y0,y1,b            x:(r1),x1
        subl    b,a                 b,x:(r4)+       y:(r0),b

_end_bfy

        move    #coef,r6
        move                        a,x:(r5)+n5     y:(r1)+n1,y1
        move                        x:(r0)+n0,x1    y:(r4)+n4,y1

_end_grp
```

```
        andi    # $fb,mr
        move    n2,b1
        lsr     b               n0,a1
        lsl     a               b1,n2
        mvoe    a1,n0

_end_pass

        endm
```

图 8-15　FFT3.ASM 汇编宏指令

例 8.6：执行 FFTEX1.LOD，FFTEX2.LOD 和 FFTEX3.LOD 程序，每个程序用不同的方法在复输入序列上完成 8 点 FFT。

在 FFTEX1.LOD、FFTEX2.LOD 和 FFTEX3.LOD 程序中，输入序列的实部元素存在 X 存贮器单元 $18～$1f 中，而虚部元素存在 Y 存贮器单元 $18～$1f 中。每个程序调用 SINCOS 宏指令以产生正、余弦查找表，其中余弦表的值存在 X 存贮器单元 $8～$b 中，而正弦表值存在 Y 存贮器单元 $8～$b 中。每个程序也可调用 BITREV 宏指令按倒位次序存贮，实部倒位元素存在 X 存贮器单元 $0～$7 中，虚部倒位元素存在 Y 存贮器单元 $0～$7 中。FFTEX1.ASM、FFTEX2.ASM 和 FFTEX3.ASM 汇编程序分别调用 FFT1、FFT2 和 FFT3，以在倒位输入序列上完成 8 点 FFT，其中正常次序输出序列的实部元素存在 X 存贮器单元 $0～$7 中，而虚部元素存在 Y 存贮器单元 $0～$7 中。

在 FFTEX1.LOD 程序中，输入序列元素幅值限制到 0.125。FFTEX1.ASM 汇编程序示于图 8-16。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; 8-point FFT
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                include 'sincos'
                include 'bitrev'
                include 'fft1'
reset           equ     0
start           equ     100
points          equ     8
data            equ     0
coef            equ     8
olddata         equ     $18

; points    =number of points
; coef      =base address of sine/cosine table
; olddata   =Starting address of input data before bit reverse
; data      =Starting address of input data after bit reverse

                org     x:$18           ; Begin of old data
olddatar                                ; (real part)
                dc      0.1,0.1,0.05,0.1
                dc      0.1,0.05,0.1,0.1
```

```
            org     y:$18               ; Begin of old data
olddataI                                ; (imaginary part)
            dc      0,0,0,0
            dc      0,0,0,0

            sincos  points,coef

            org     p:reset
            jmp     start
            org     p:start

            bitrev  points,data,olddata
            nop

            fft1    points,data,coef,olddata
            end
```

图 8-16　FFTEX1.ASM 汇编程序

1. 装入“SIM56000 模拟软件”程序。
2. 把“DSP56000/1 示范软件＃3”软盘插入驱动器“A”。
3. 键入 SIM56000 命令：

```
display off <return>
load a :fftex1<return>
break #1 pc>$138<return>
go #1<return>
radix f x:$18..$1f<return>
display x:$18..$1f<return>
radix f y:$18..$1f<return>
display y:$18..$1f<return>
radix f x:$0..$7<return>
display x:$0..$7<return>
radix f y:$0..$7<return>
display y:$0..$7<return>
```

4. 键入 quit<return>，退出 SIM56000 程序。

输入序列实部元素：

X:$0018	0.1000000	0.1000000	0.0500000	0.1000000
X:$001C	0.1000000	0.0500000	0.1000000	0.1000000

输入序列虚部元素：

Y:$0018	0.0000000	0.0000000	0.0000000	0.0000000
Y:$001C	0.0000000	0.0000000	0.0000000	0.0000000

输出序列实部元素：

X:$0000	0.7000000	0.0353554	0.0500001	−0.353554
X:$0004	0.0000000	−0.0353554	0.0500001	0.0353554

输出序列虚部元素：

Y:$0000	0.0000000	0.0146446	0.0500001	−0.0853555

Y：$0004　　0.0000000　　0.0853555　　−0.0500001　　−0.0146446

在 FFTEX2.LOD 程序中，所有 FFT 蝶形的输出是缩小比例为 2。当要求绝对值时，共用比例因子为 2^3。FFTEX2.LOD 汇编程序示于图 8-17。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; 8-point FFT
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                include 'sincos'
                include 'bitrev'
                include 'fft2'

reset           equ     0
start           equ     100
points          equ     8
data            equ     0
coef            equ     8
olddata         equ     $18

; points    =number of points
; coef      =base address of sine/cosine table
; olddata   =Starting address of input data before bit reverse
; data      =Starting address of input data after bit reverse
                org     x:$18                   ; Begin of old data
olddatar                                        ; (real part)
                dc      0.8,0.8,0.4,0.8
                dc      0.8,0.4,0.8,0.8

                org     y:$18                   ; Begin of old data
olddataI                                        ; (imaginary part)
                dc      0,0,0,0
                dc      0,0,0,0

                sincos  points,coef

                org     p:reset
                jmp     start
                org     p:start

                bitrev  points,data,olddata
                nop

                fft2    points,data,coef,olddata
                end
```

图 8-17　FFTEX2.ASM 汇编程序

经过与 FFTEX1.LOD 的相似的过程，可得结果如下：

输入序列实部元素：

X：$0018　　0.8000000　　0.8000000　　0.4000000　　0.8000000

X：$001C　　0.8000000　　0.4000000　　0.8000000　　0.8000000

输入序列虚部元素：

Y：$0018	0.0000000	0.0000000	0.0000000	0.0000000
Y：$001C	0.0000000	0.0000000	0.0000000	0.0000000

FFT 输出序列实部元素：

X：$0000	0.6999999	0.0353553	0.0500000	−0.0353553
X：$0004	0.0000000	−0.0353553	0.0500001	0.0353553

FFT 输出序列虚部元素：

Y：$0000	0.0000000	0.0146446	0.0500000	−0.0853553
Y：$0004	0.0000000	0.0853553	−0.0500000	−0.0146446

在 FFTEX3.LOD 程序中，方块浮点方法用来完成有条件定标。在此情况下，比例因子等于 2^K，其中 K 是缩小比例的级数。对 N=3，K 从 0 变到 2。FFTEX3.ASM 汇编程序示于图 8-18。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; 8-point FFT
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                include 'sincos'
                include 'bitrev'
                include 'fft3'

reset           equ     0
start           equ     100
points          equ     8
data            equ     0
coef            equ     8
olddata         equ     $18
accr            equ     $20
; points   =number of points
; coef     =base address of sine/cosine table
; olddata  =Starting address of input data before bit reverse
; data     =Starting address of input data after bit reverse
; accr     =address for the ccr register

        org     x:$18                   ; Begin of old data
olddatar                                ; (real part)
        dc      0.2,0.2,0.1,0.2
        dc      0.2,0.1,0.2,0.2

        org     y:$18                   ; Begin of old data
olddataI                                ; (imaginary part)
        dc      0,0,0,0
        dc      0,0,0,0

        sincos  points,coef

        org     p:reset
        jmp     start
        org     p:start
```

```
bitrev   .    points,data,olddata
nop

fft3          points,data,coef,olddata accr
end
```

图 8-18　FFTEX3.LOD 汇编程序

经过与前面相似过程，可得结果如下：

输入序列实部元素：

X：$0018　0.2000000　0.2000000　0.1000000　0.2000000

X：$001C　0.2000000　0.1000000　0.2000000　0.2000000

输入序列虚部元素：

Y：$0018　0.0000000　0.0000000　0.000000　0.000000

Y：$001C　0.0000000　0.0000000　0.0000000　0.0000000

比例因子：2^K，K＝2

r7＝$0002

FFT 输出序列实部元素：

X：$0000　0.3500001　0.0176777　0.0250000　−0.0176777

X：$0004　0.0000000　−0.0176777　0.0250000　0.0176777

FFT 输出序列虚部元素：

Y：$0000　0.0000000　0.0073223　0.0250001　−0.0426776

Y：$0004　0.0000000　0.0426776　−0.0250001　−0.0073223

第九章　在 DSP56001 处理器上设计与实现自适应 FIR 滤波器

本章讨论在 DSP56001 处理器上设计和实现有限脉冲响应自适应滤波器。自适应滤波器有能力自动调整它自身的参数（系数）。因此其设计要求很少或不要求系统的输入信号或噪声特性的先验知识。自适应滤波器有两个输入 x(n) 和 d(n)，它们通常以某些方式相关。自适应滤波器示于图 9－1：

1. 用当前参数估计计算的滤波器输出 y(n) 与输入信号 d(n) 比较。

2. 预测误差 e(n) 通过参数自适应算法反馈，产生新的参数估计 。

3. 当下一个输入样值收到时，产生新的预测误差。自适应滤波器的目的就是减小预测误差。

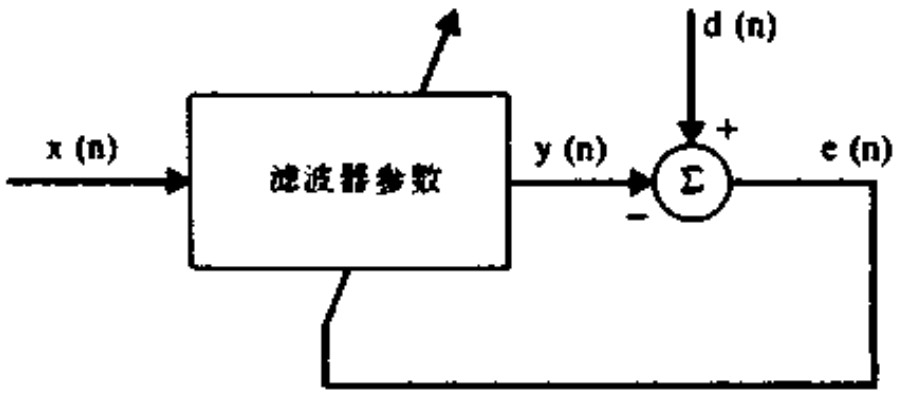


图 9－1 自适应滤波器

自适应滤波器的两个方面是其内部结构和它的自适应算法。它的内部结构可以是非递归（FIR）或递归（IIR）滤波器。其自适应算法可分为二种主要类别：梯度算法和非梯度算法。本章采用梯度算法调整 FIR 滤波器参数。

梯度算法是基于线性和非线性系统数字解所用的最陡下降方法。最小均方算法（LMS）或许是最广泛使用的梯度算法。LMS 算法调整滤波器参数以减小滤波器输出 y(n) 和所要求的响应输入 d(n) 之间的均方误差。

本章先介绍 LMS 算法，然后叙述在 DSP56001 处理器上实现自适应 FIR 滤波器的 DSP56001 指令。

9.1　LMS　算　法

信号 x(n) 被滤波后可按照以下公式估计输入信号 d(n)：

$$y(n) = \sum_{i=0}^{N-1} b_i(n)x(n-i) = B^T(n)X(n) \tag{9-1}$$

其中

$$B^T(n) = [b_0(n)b_1(n)\cdots b_{N-1}(n)]^T \tag{9-2}$$

$$X(n) = [x(n)x(n-1)\cdots x(n-N+1)]^T \tag{9-3}$$

估计中误差可计算如下：

$$e(n) = d(n) - y(n) = d(n) - B^T(n)X(n) \tag{9-4}$$

误差的平方等于：

$$e^2(n) = d^2(n) - 2d(n)X^T(n)B(n) + B^T(n)X(n)X^T(n)B(n) \tag{9-5}$$

均方误差是 $e^2(n)$ 的期望值，可写为：

$$E[e^2(n)] = E[d2(n) - 2R(x,d)B(n) + B^T(n)R(x,x)B(n)] \tag{9-6}$$

其中 x(n) 和 d(n) 之间互相关的矢量定义为：

$$R(x,d) = E[d(n)X(n)^T] \tag{9-7}$$

其中输入信号 x(n) 的相关矩阵是：

$$R(x,x) = E[X(n)X(n)^T] \tag{9-8}$$

对平稳输入信号，式 (9-6) 中的均方误差是参数的二阶函数。最陡下降方法寻找最小均方误差法是使参数矢量中的每个变化正比于均方误差相对于参数矢量的梯度。于是最陡下降法可用下式表示：

$$B(n+1) = B(n) - 0.5K\nabla[E(e^2(n)] \tag{9-9}$$

其中

B (n+1) ＝下一采样周期要用的更新参数矢量。

B (n) ＝当前采样周期的参数矢量。

K＝控制收敛速率和滤波器稳定性的环路增益。K 是正增益。

$\nabla[E(e^2(n)]$ ＝ 均方误差对参数矢量 B(n) 的梯度。

用 (9-6) 式，均方误差梯度可得为：

$$\nabla[E(e^2(n)] = -2R(x,d) + 2R(x,x)B(n) \tag{9-10}$$

令梯度为 0 即可得最小均方误差，其解如下：

$$B_{LMS} = R^{-1}(x,x)R(x,d) \tag{9-11}$$

(9-11) 式通常称为 Wiener-Hopf 方程。

LMS 算法是找(9-11)式近似解常用的方法，此法中不要求对相关函数作明确的的测量，也不要求矩阵变换。LMS 算法可写为：

$$B(n+1) = B(n) - 0.5K\hat{\nabla}[E(e^2(n)] \tag{9-12}$$

其中 $\hat{\nabla}[E(e^2(n)]$ 是均方误差梯度的估计。它可用平方误差的单次样 值得到：

$$\hat{\nabla}[E(e^2(n))] = \nabla[e^2(n)] = 2e(n)\nabla e(n) \tag{9-13}$$

由 (9-4) 式得：

$$\nabla e(n) = \nabla[d(n) - B^T(n)X(n)] = -X(n) \tag{9-14}$$

$$\hat{\nabla}[E(e^2(n))] = -2e(n)X(n) \tag{9-15}$$

用 (9-12) 和 (9-15) 式，LMS 算法可写为：

$$B(n+1) = B(n) + Ke(n)X(n) \tag{9-16}$$

(9-16) 式可写为：

$$b_i(n+1) = b_i(n) + Ke(n)x(n-i) \qquad i = 0,1,\cdots,N-1 \tag{9-17}$$

其中

$b_i(n+1)$ ＝ 下一采样周期要用的更新的第 i 个参数值

$b_i(n)$ ＝ 当前采样周期的第 i 个参数值

在下一采样周期的开始，移二个新输入采样值到系统并且重复此过程，几次迭代后，FIR 参数收敛到与最小均方误差一致。

环路增益 K 控制收敛速度和 FIR 滤波器稳定性。K 的最佳值取决于所用的 FIR 抽头长

度。由经验得知，以下的值是推荐的最大值，并将由小于或等于所选 FIR 相应长度的 K 值开始。

FIR 抽头长度应舍入到与表 9－1 所示最接近的 FIR 抽头长度。

表 9－1　推荐的最大环路增益

FIR 抽头长度	推荐的最大 K
≤32（20Hex）	0.75（600000Hex）
≤64（40Hex）	0.375（300000Hex）
≤128（80Hex）	0.125（100000Hex）
≤192（C0Hex）	0.078125（A0000Hex）
≤256（FFHex）	0.0703125（90000Hex）

在很多数字信号处理应用中，诸如自适应线性增强（ALE），输入信号 x(n) 是输入信号 d(n) 的延迟。ALE 检测和跟踪在宽带噪声中的移动谱线而增强信噪比。下节中假设 d(n) = x(n)。并假设 d(n) 由所要求的信号 s(n) 和白噪声信号 w(n) 组成，

$$d(n) = s(n) + w(n) \tag{9-18}$$

9.2　实现自适应 FIR 滤波器

(9－1)，(9－4) 和 (9－17) 式用来实现自适应 FIR 滤波器。为得到输出样值 y(n)，误差样值 e(n) 和更新的参数 bi(n + 1)，必须完成以下步骤：

1. 存输入样值 x(n) 作为第一个滤波器状态。
2. 以 bo(n) 乘 x(n) 并累加滤波器状态 x(n－i) 和系数 bi(n) 的乘积以产生输出样值 y(n)。
3. 从输入样值 x(n) 中减去输出样值 y(n) 以得到误差样值 e(n)。
4. 更新滤波器参数得到下个采样周期的参数 bi(n + 1)。
5. 移滤波器状态得到下一个输出样值 y(n + 1)

9.2.1　在 DSP56001 处理器上完成自适应 FIR 滤波器

类似于第六章提到的 FIR 滤波器，以下所述各点可使 1～5 步在 DSP56001 处理器上有效实现：

1. 地址指针对 RAM 数据的模拟 FIFO 位移，
2. 模数寻址能力提供循环数据缓存，
3. 乘/累加（MAC）指令使两个操作数相乘，并在单指令周期中把乘积加至第三操作数
4. MAC 指令与数据传送能力并行以保持相乘器以 100%能力运行，
5. 重复下条指令（REP）的指令提供紧致滤波码。

DSP56001 处理器能完成模寻址使地址存贮器（Rn）的值递增（或递减），并仍保持在地址范围 L 以内，这里 L 由下和上地址界定义。

对自适应 FIR 滤波器，L 等于系数（抽头）数。

L－1 值存在 DSP56001 处理器修正寄存器（Mn）中，因为实现了 DSP56001 处理器的模寻址机理，所以 L 的下地址界（基地址）在它的 JLSB 中必须是 0，这里 $2^{**}J \geqslant L$，即基地址值必须是 $2^{**}J$ 的倍数。上地址界是下地址界加模值减 1，即基地址值加 L－1。注意上地址界由 DSP56001 处理器计算，而不存在寄存器中。

当用模寻址时，地址寄存器（Rn）指示在 X 存贮器或 Y 存贮器中的模数据缓存器。地址指针（Rn）不要求指在下地址界上，即它可指在定义的模地址范围 L 的任何地方。若地址指针增量超过上地址界，将循环到基地址。

9.2.2 实现自适应 FIR 滤波器的 DSP56001 指令

图 9－2 所示的 DSP56001 指令可用来实现自适应 FIR 算法，这些指令分别执行：

1. 编程模寄存器 M0 到 NTAPS－1 值（模 NTAPS）编程地址寄存器 R0 指向 X 存贮器中的状态变量模缓存器。编程模寄存器 M4 到 NTAPS－1 值（模 NTAPS）。

```
;
; Adaptive FIR Filter

; x0 =Input sample
; a  =Output sample
; b  =Error sample * loop gain
; ntaps =number of parameter taps in the filter
; lg    =loop gain

;       Initialize routine

        move    #states, r0
        move    #pata, r4
        move    #ntaps－1, m0
        move    m0, m4

;       The following code obtains the output sample y (n);

        clr     a               x0, x: (r0) +       y: (r4) +, y0
        rep     #ntaps－1
        mac     x0, y0, a       x: (r0) +, x0       y: (r4) +, y0
        macr    x0, y0, a       x: (r0), b

;       The following code obtains the product of the error
;       sample and the loop gain:

        sub     a, b

        move    #lg, y1
        move    b, x1
        mpy     x1, y1, b
        move    b, y1

;       The following code updates the parameters:

        do      #ntaps,_update
        move    y: (r4), ax: (r0) +, x0
        mac     x0, y1, a
        move    a, y: (r4) +
—update

;       The following instruction updates the address register R0

        lua         (r0) －, r0
```

图 9－2 实现自适应 FIR 滤波器的汇编程序码

编程地址寄存器 R4 指向 Y 存贮器中的系数缓存器。给定 FIR 滤波器算法已执行一些时间，并准备处理在数据 ALU 输入寄存器 X0 中的样值 x(n)。R4 中的地址是系数缓存器的基地址（低界）。R0 中的地址是 M，这里 M 大于等于 X 存贮器地址的低界，小于或等于 X 存贮器地址的高界。X 存贮器映射滤波器状态，Y 存贮器映射系数，DSP56001 处理器的 A 和 B 累加器和数据 ALU 输入寄存器 X0、X1、Y0、Y1 的内容如图 9－3 所示

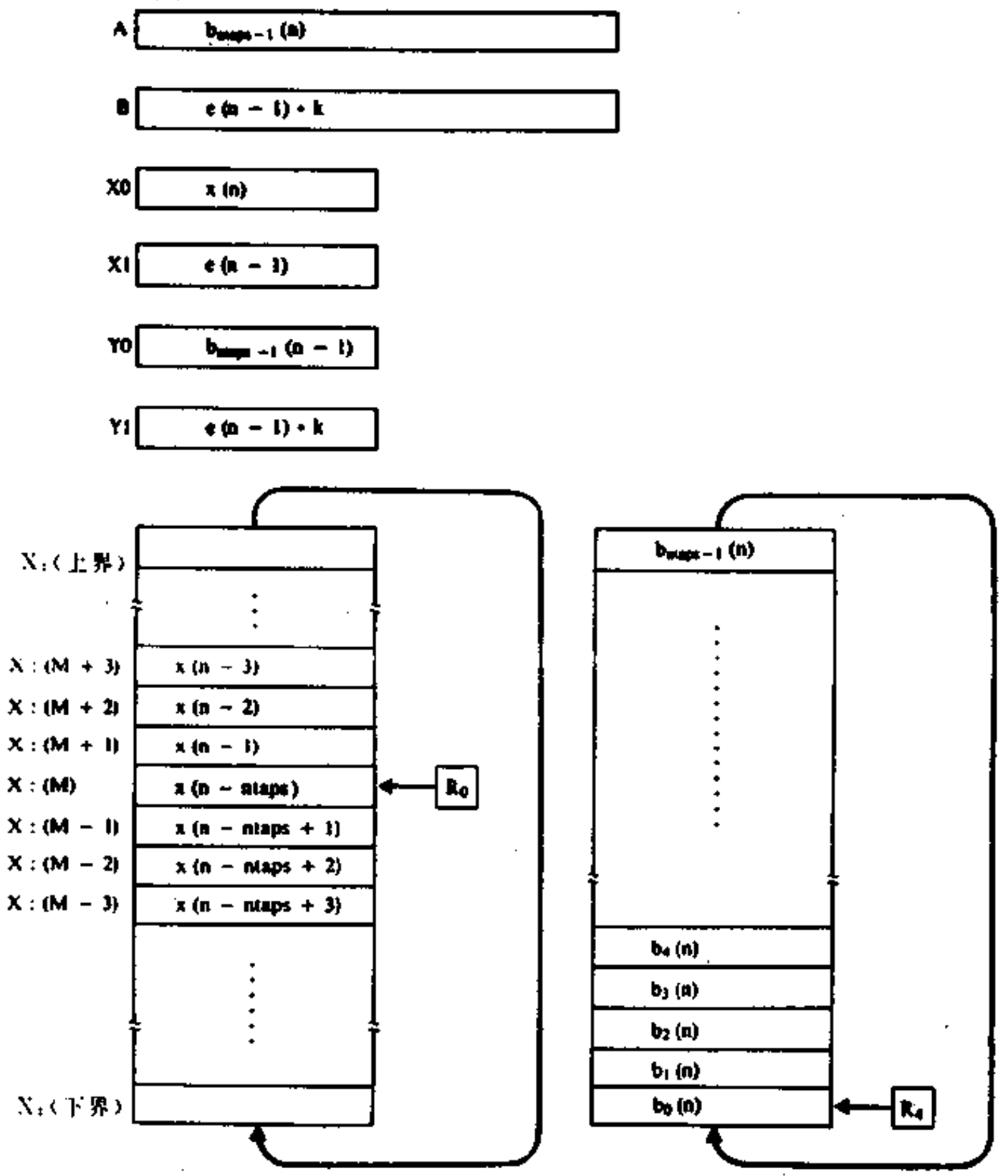


图 9－3　第 n 次迭代开始时的存贮器映像和数据寄存器

2. CLR 指令清除 A 累加器，并同时执行：

(1) 从数据 ALU 的输入寄存器 X0 传送输入样值 x(n) 到 R0 所指的 X 存贮器单元。

(2) 从 R4 所指的 Y 存贮器单元传送第一系数到数据 ALU 的输入寄存器 Y0。

R0 和 R4 在 CLR 指令结束时都自动增 1。有关寄存器内容如图 9－4。

3. REP 指令管理 MAC 指令的 NTAPS－1 次迭代的执行。

```
mac     x0, y0, a     x: (r0) +, x0     y: (r4) +, y0
```

4. MAC 指令使 X0 中的滤波器状态变量与 Y0 中的系数相乘，并将乘积加到 A 累加器，

同时执行：

（1）从地址寄存器 R0 所指的 X 存贮器传送下一个状态变量到输入寄存器 X0；

（2）从地址寄存器 R4 所指的 Y 存贮器传送下一个系数到输入寄存器 Y0。

R0 和 R4 在 MAC 指令结束后都自动增 1。有关寄存器的内容，在执行第一、第二和最后一条 MAC 指令后分别示于图 9－5，9－6 和 9－7。

注意当执行滤波器算法时，地址寄存器 R4 是后递增 NTAPS 次，即 CLR 指令执行一次，而 MAC 指令执行 NTAPS－1 次。因 R4 模是 NTAPS，而 R4 是递增 NTAPS 次，所以 R4 中的地址值循环并指向系数缓存器的低界单元。

注意当执行滤波器算法时，R0 后递增总的 NTAPS 次。还应注意在算法开始，输入样值 x(n) 从数据 ALU 寄存器 X0 传送到 R0 指示的 X 存贮器单元。因 R0 模是 NTAPS，而 R0 是递增 NTAPS 次，所以 R0 中的地址值循环并指示状态变量缓存器的 X 存贮器单元 M。

5. MACR 指令计算滤波器算法的最后抽头，并完成结果的收敛舍入。这条指令的数据传送部分输入样值 x（n）装入到 B 累加器中。在 MACR 指令结束时，A 累加器包含了滤波器输出样值 y（n）。X 存贮器映射滤波器状态，Y 存贮器映射系数，MACR 指令之后各寄存器的内容如图 9－8 所示。

6. SUB 指令计算误差样值 e(n)，即从 B 累加器中的输入样值 x(n) 减去 A 累加器中的输出 y(n)。e(n) 存入 B 中。

7. 两条传送指令 move＃lg，y1 和 move b，xi 分别将环路增益 K 和误差样值 e(n) 传送到数据寄存器 Y1 和数据输入寄存器 XI。

8. MPY 指令用环路增益 K 乘 e(n)，并存结果于 B 累加器中。MPY 指示执行后的各寄存器内容如图 9－9 示。

9. MOVE 指令 move b，y1 将 e(n) 和 K 的乘积传送到数据寄存器 YI。

10. DO 指令设置硬件“do loop”以更新 NTAPS 滤波器的参数，DO 指令后的 3 条指令重复 NTAPS 次。

11. 在“do loop”中的第一条 MOVE 指令传送参数 $b_i(n)$ 到 A 累加器，和传送滤波器状态 x(n－i) 到数据输入寄存器 X0。地址寄存器 R0 增 1 以指示下一个滤波状态。

12. MAC 指令使 X0 中的滤波器状态乘以 YI 中的 K 和 e(n) 的乘积，并加乘积到 A 累加器中。A 中的结果是更新的参数 $b_i(n+1)$。

13. 在“do loop”中的第二条 MOVE 指令传送 $b_i(n+1)$ 到 R4 指示的 Y 存贮器单元。R4 增 1 以指示下一个滤波器参数。第一、第二和最后一次的“do loop”通过以后，各寄存器的内容如图 9－10，9－11 和 9－12 所示。

14. LUA 指令将 R0 减 1。于是 R0 指向状态变量缓存器的 X 存贮器单元 M－1。执行下一次算法后，新的输入 x(n＋1) 将写在 X 存贮器单元 M－1 上。因此滤波器状态变量的似先进先出位移只能直接调整 R0 地址指针来完成。LUA 指令之后寄存器的内容如图 9－13 所示。

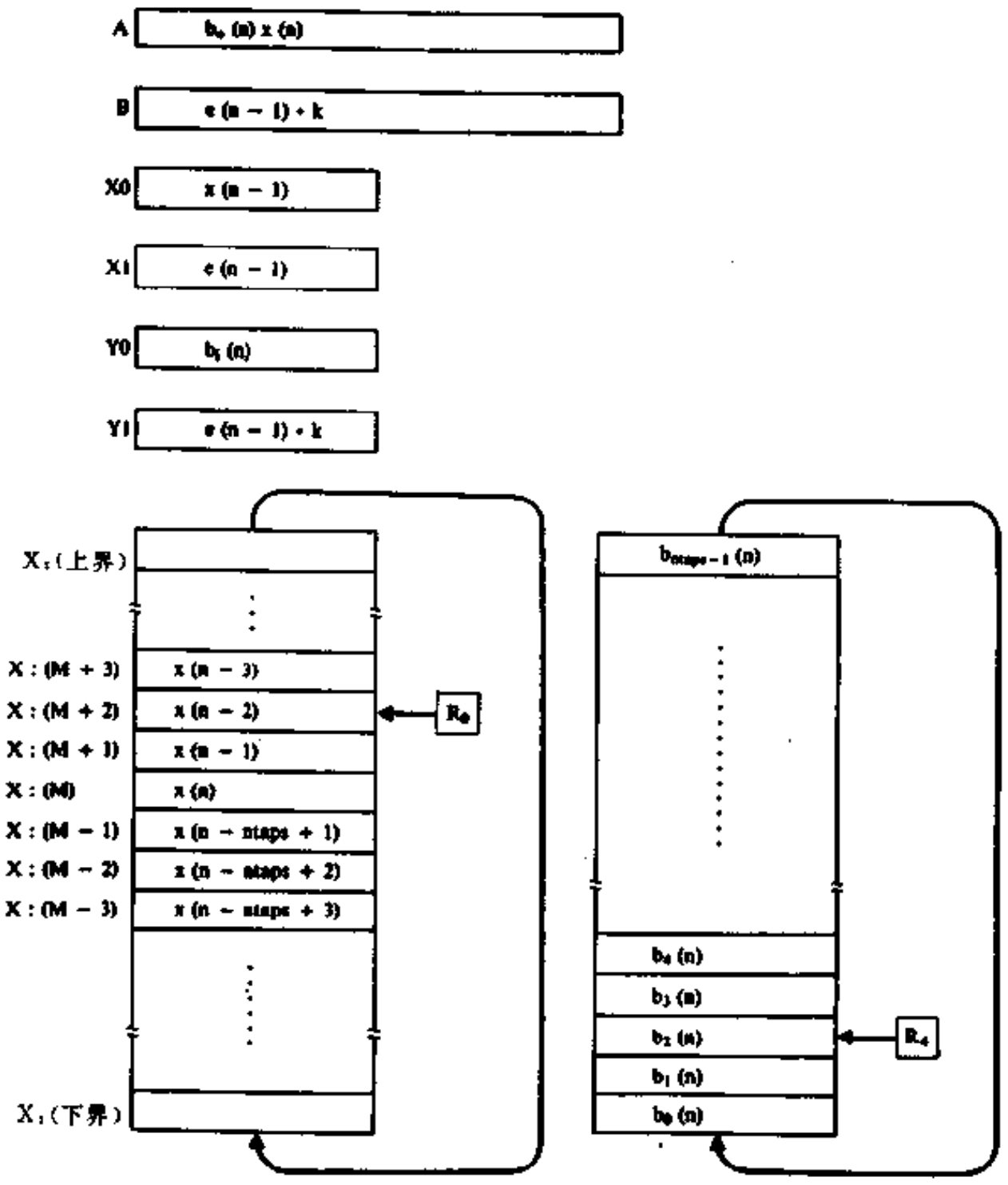


图 9－5　第一条 MAC 指令执行后的存贮器映像和数据寄存器

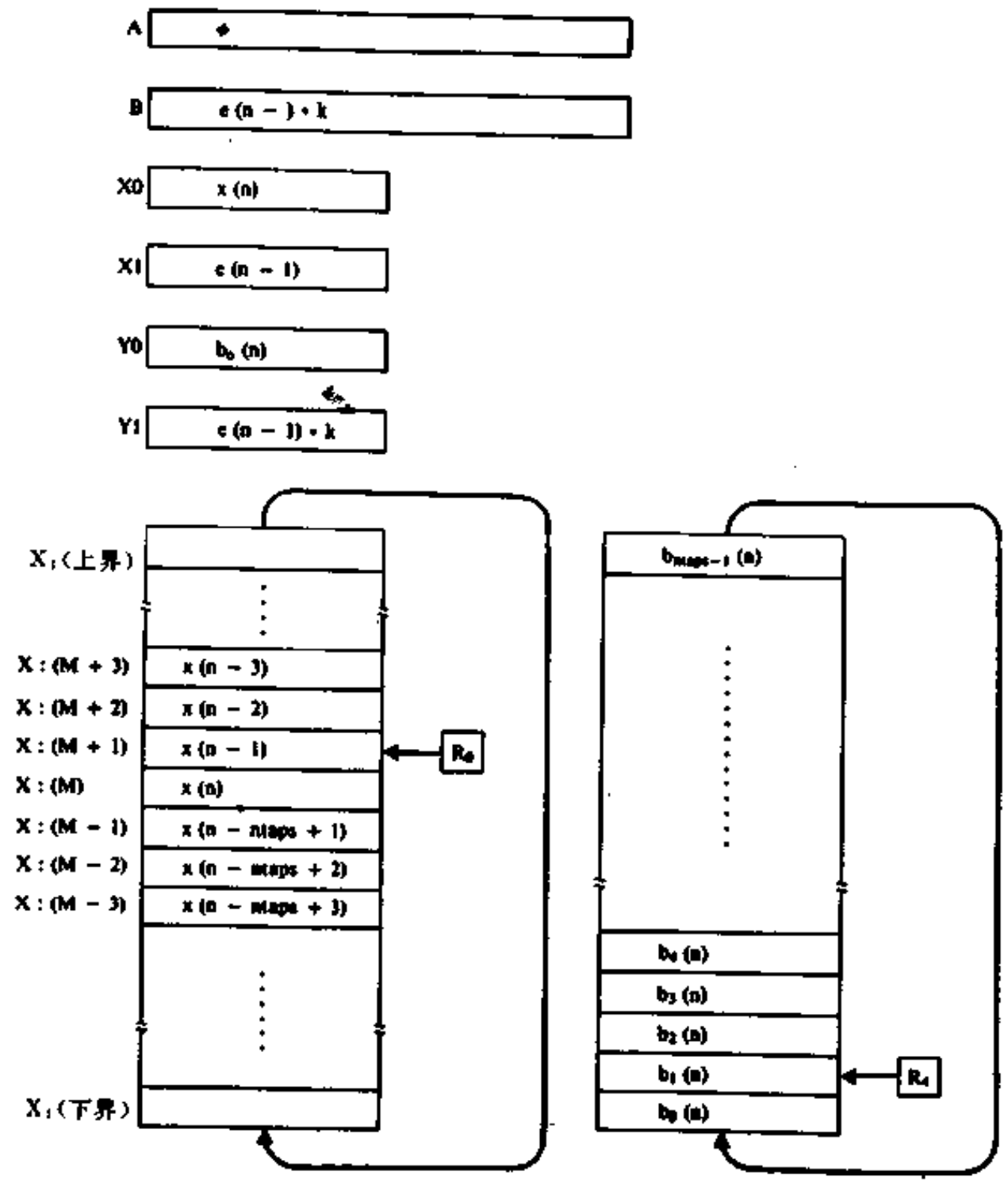


图 9－4　CLR 指令执行后的存贮器映像和数据寄存器

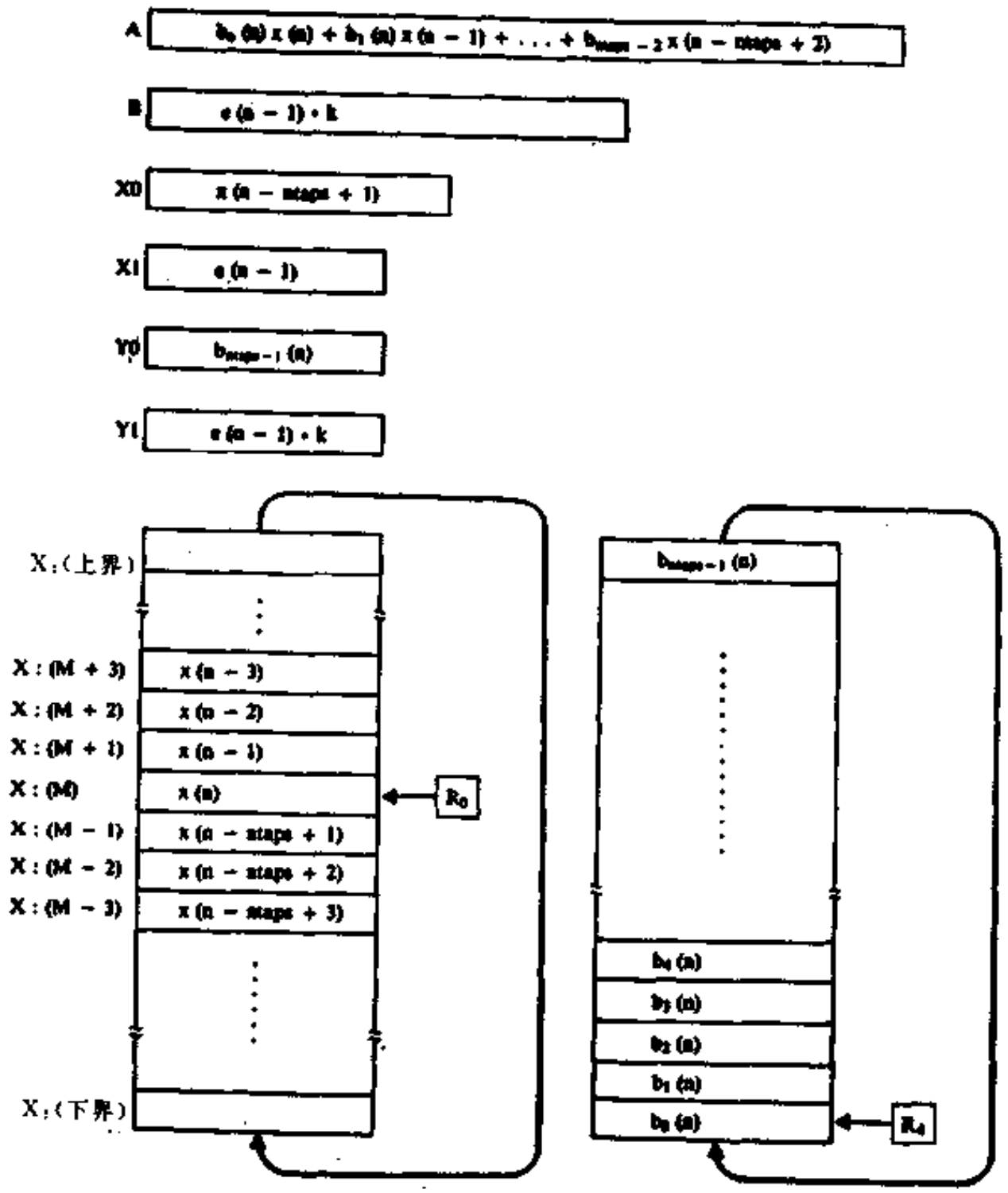


图 9-7　最后一次 MAC 指令后的存贮器映像和数据寄存器

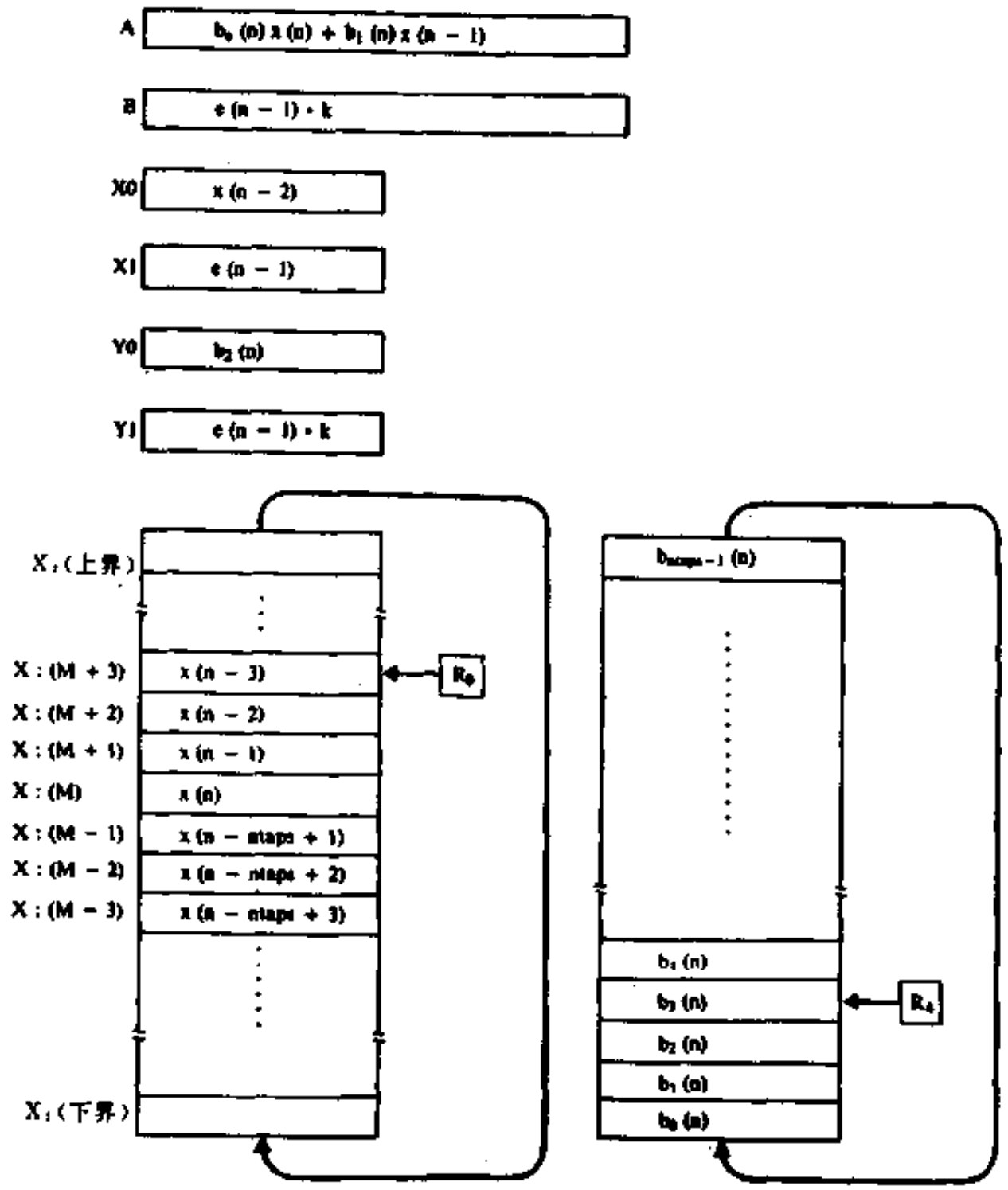


图 9-6　第二次 MAC 指令后的存贮器映像和数据寄存器

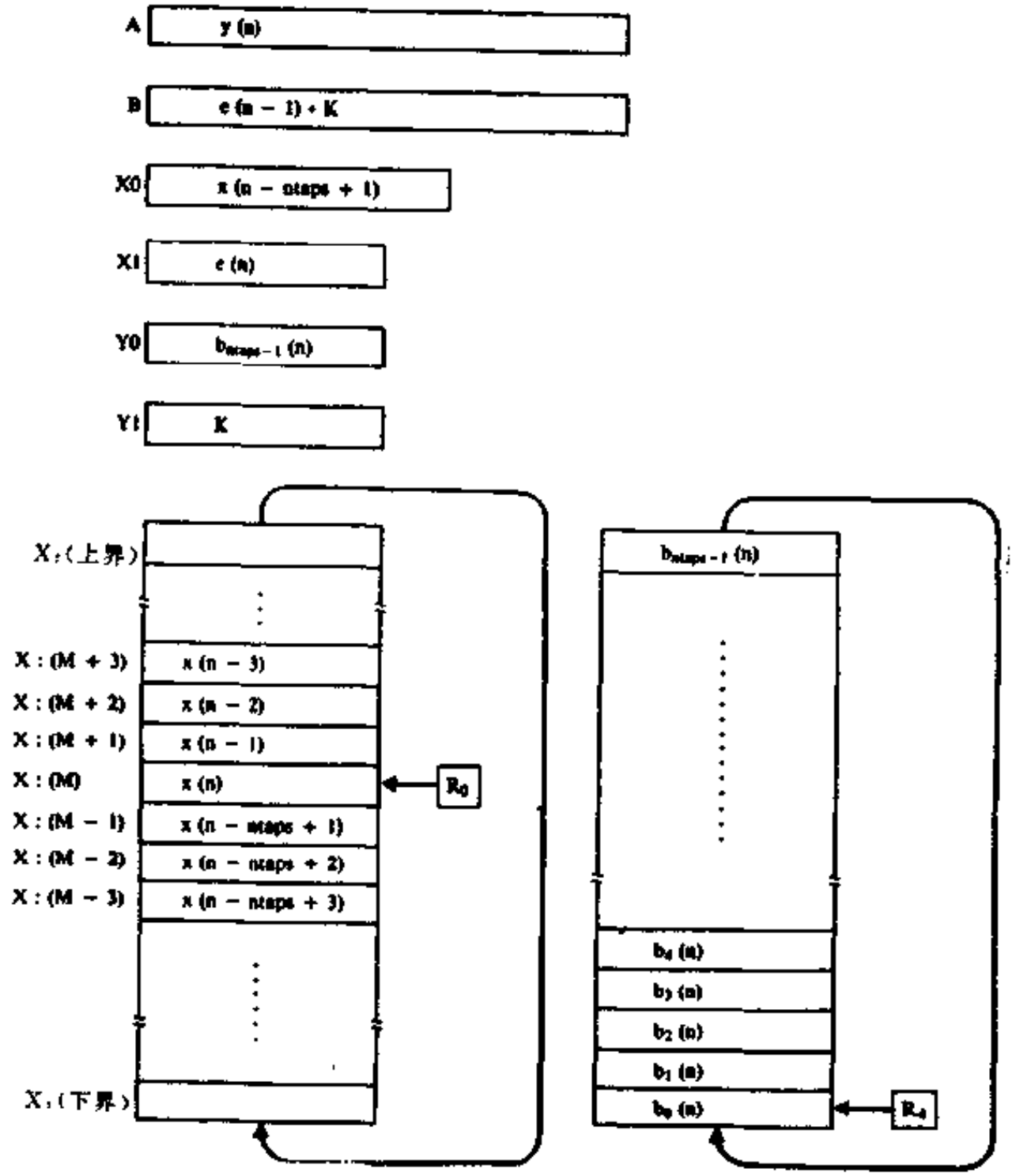


图 9-9 MPY 指令后的存贮器映像和数据寄存器

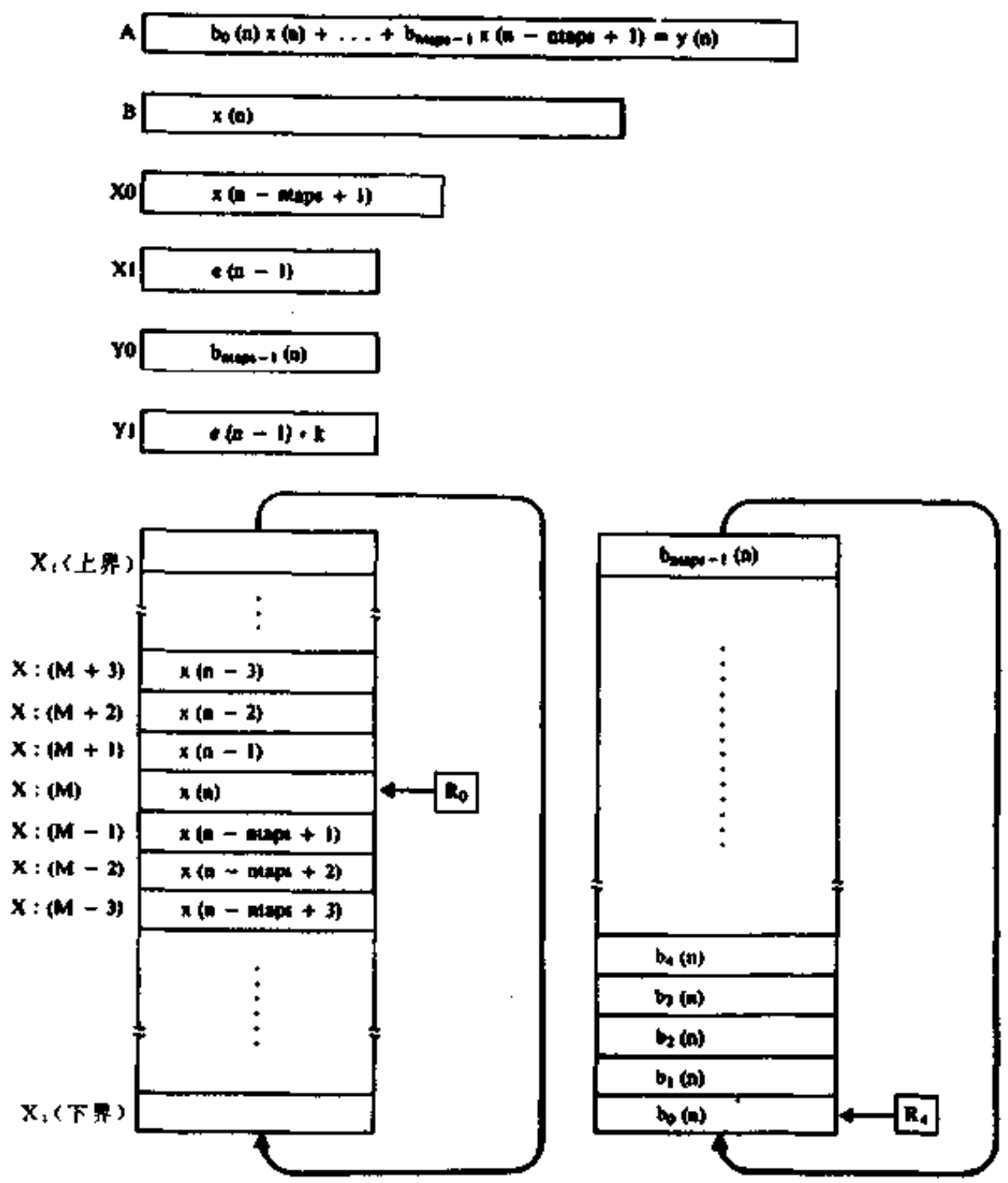


图 9-8 MACR 指令之后的存贮器映像和数据寄存器

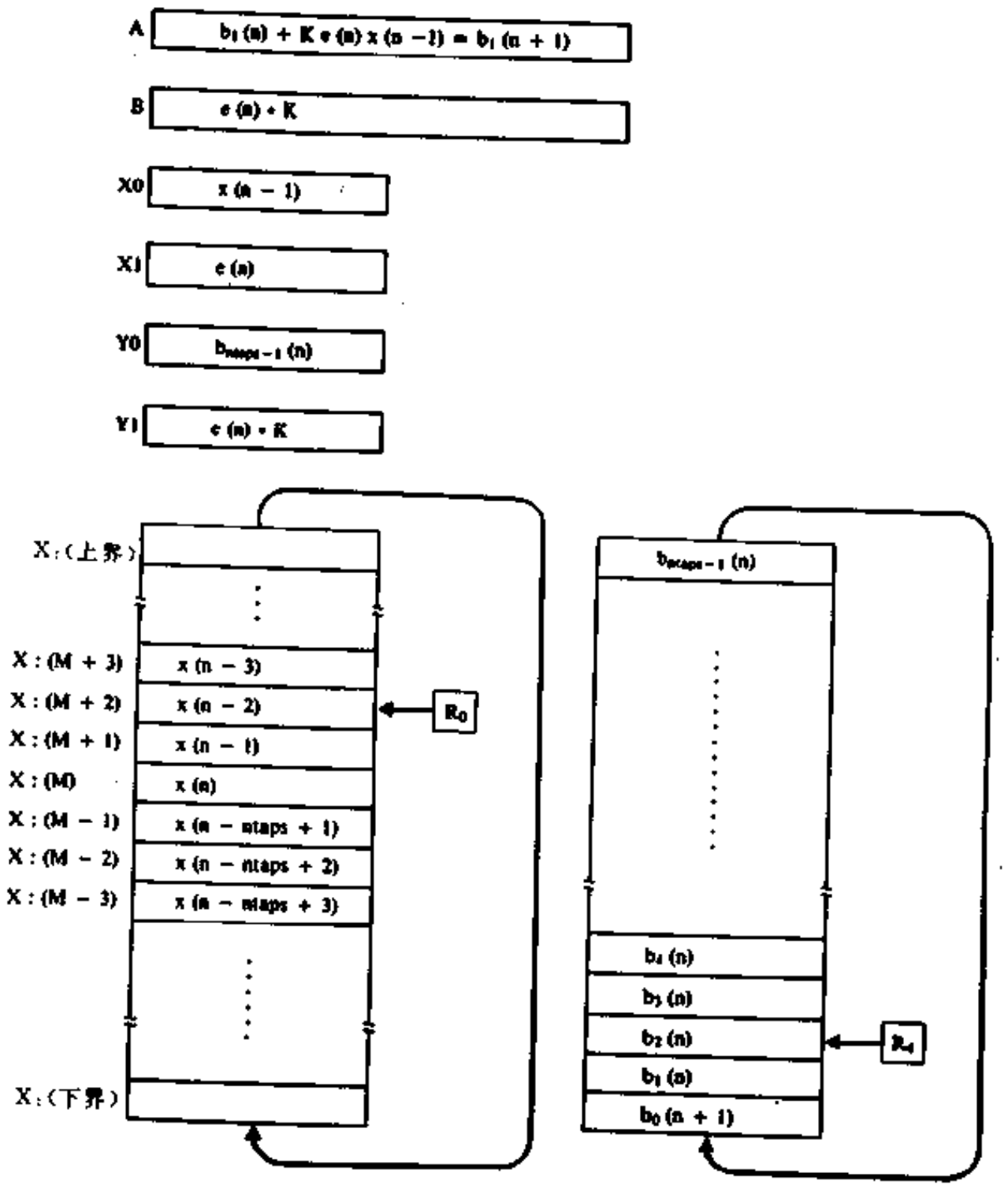


图 9－11　Do Loop 第二次后的存贮器映像和数据寄存器

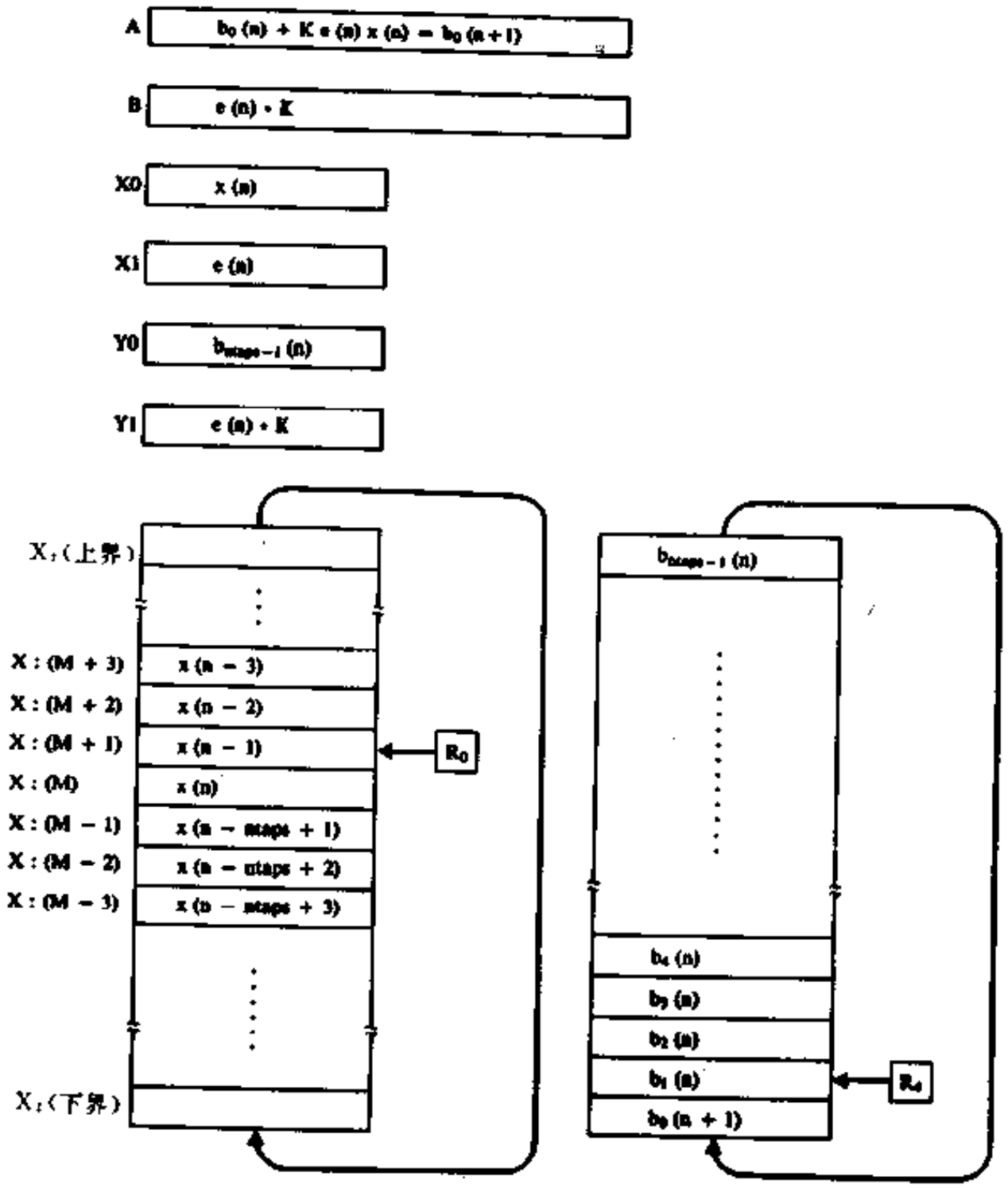


图 9－10　Do Loop 第一次以后的存贮器映像和数据寄存器

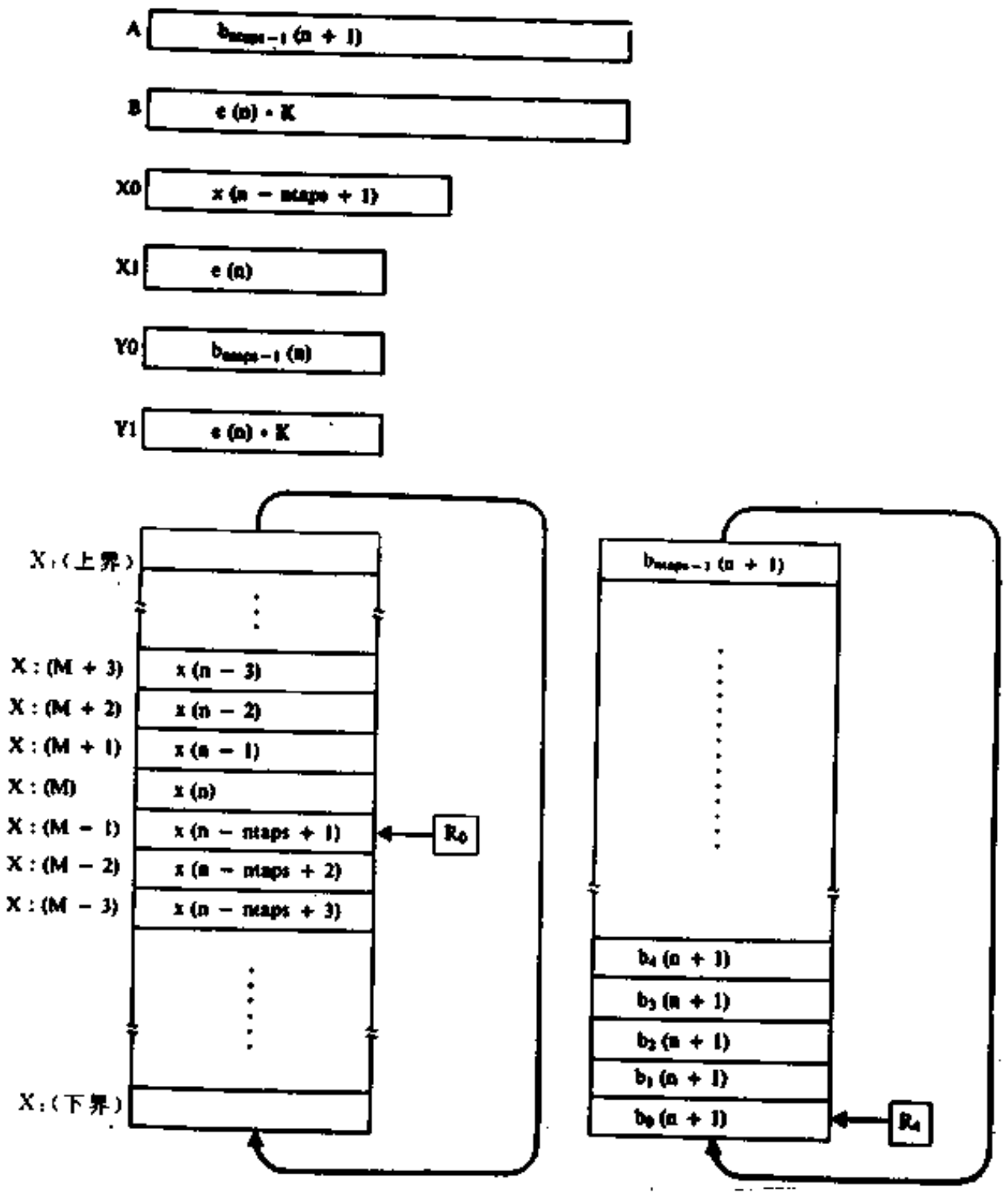


图 9-13　LUA 指令后的存贮器映像和数据寄存器

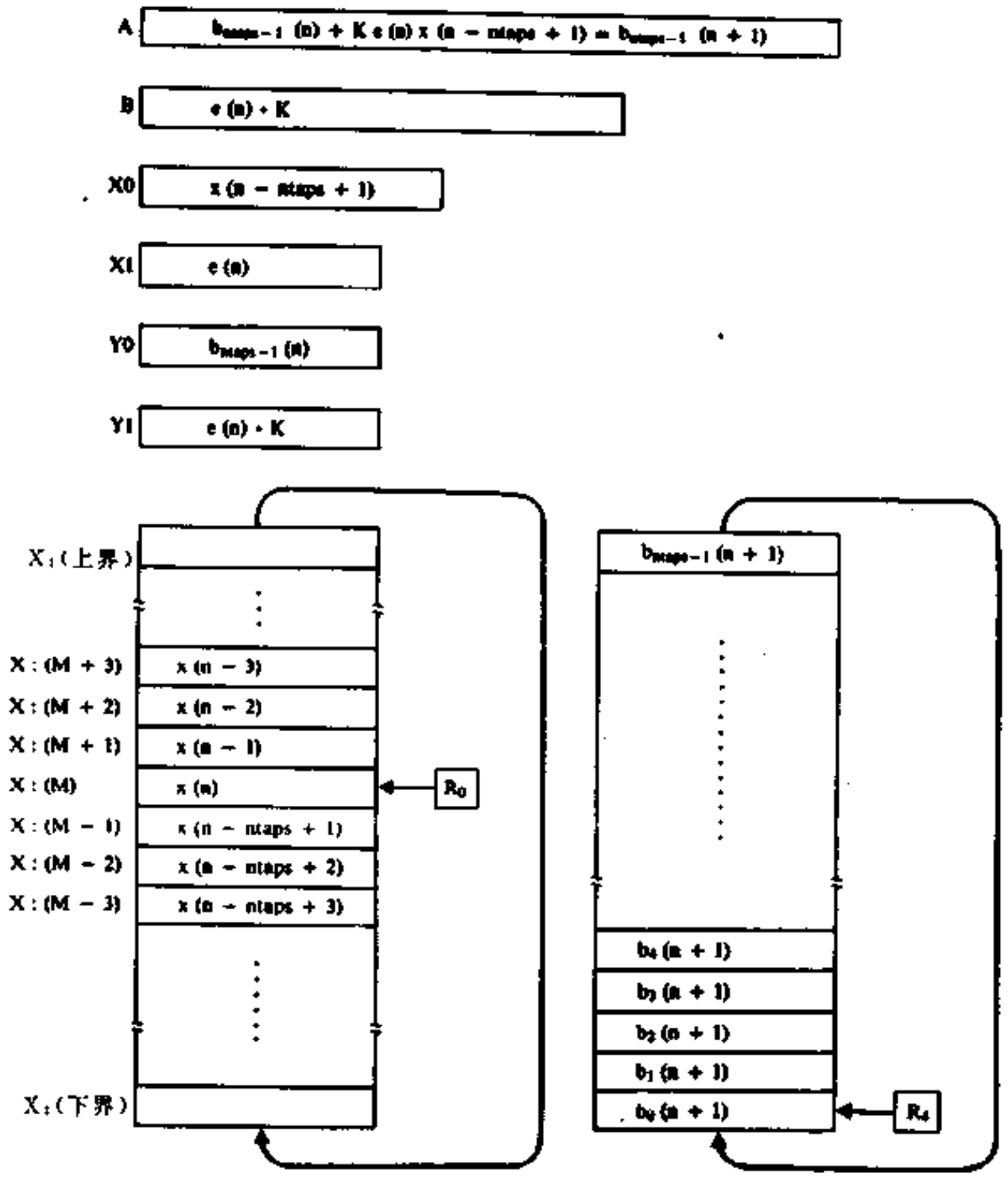


图 9-12　Do Loop 最后一次之后的存贮器映像和数据寄存器

9.3 自适应 FIR 滤波器示范系统

自适应滤波过程可用第五章所述的 DSP 系统来说明。DSP 系统由 DSP56000ADS 应用开发系统(ADS)和 A/D-D/A 评价板(EVB)组成。模拟输入信号 x(t) 由所要求的输入信号 x(n) 加白噪声 w(n) 组成。模拟输入信号 x(t) 首先用 EVB 上的 A/D 转换器数字化，然后，DSP56001 处理器执行自适应滤波算法去处理数字化的输入信号 x(n)，从而产生滤波输出 y(n)。输出 y(n) 是 s(n) 的估计。最后 D/A 变换器把数字化输出 y(n) 变为模拟输出 y(t)。

9.4 在示范系统上实现自适应 FIR 滤波器

图 9-2 所示的 DSP56001 指令和图 5-8 所示的 DSP56001 指令一起形成示于图 9-14 的 AFIR、ASM 编程序宏指令，AFIR、ASM 汇编程序宏指令用来在示范系统上实现自适应 FIR 滤波器。

```
afir macro ntaps, lg
;
;
; ntaps =number of parameter taps in the filter
; lg    =loop gain
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Initialize routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
init2   reset                              ; Reset the system
        movep   #0, x: $ffe                ; set bcr to zero
        movep   #0, x: $ffef               ; clear the transfer register
        movep   #0, x: $ffff               ; stop all the interrupt
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Set up SSI for operation with the EVB
; The following code sets port C to function as SSI
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pcc     equ     $ffe0
        movep   #pcc, x: $ffe1             ; write PCC
cra     equ     $4000
        movep   #cra, x: $ffec             ; CRA pattern for word
                                           ; length=16 bits
cra     equ     $3200                      ; CRA pattern for continuous
        movep   #crb, x: $ffed             ; ck, synch, normal mode
                                           ; word long frame sync;
        move    #states, r0                ; point to filter states
        move    #ntaps-1, m0               ; mod (ntaps)
        move    #para, r4                  ; point to filter parameters
        move    #ntaps-1, m4               ; mod (ntaps)
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Read A/D
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; The following code polls the RDF flag in the SSI-SR and
; waits for RDF=1 and then reads the RX register to
; retrieve the data from the A/D converter.
; Sample rate is controlled by EVB board.
```

```
;
wait2     nop
wait1     btst      #7, x: $ffee
          jcc       wait1
          movep     x: $ffef, x0,          ; Read input signal
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Adaptive FIR Filter
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; x0 = Input sample
; a  = Output sample
; b  = Error sample * loop gain
;    The following code obtains the output sample y (n) and
;    the error sample e (n):
     clr       a          x0, x: (r0) +          y: (r4) +, y0
     rep       #ntaps-1
     mac       x0, y0, a  x: (r0) +, x0          y: (r4) +, y0
     macr      x0, y0, a  x: (r0), b
     move      #$ffff00, x1
     and       x1, a
     sub       a, b
;
; Write to D/A
;
          movep a, x: $ffef
;         The following code obtains the product of the error
;         sample and the loop gain:
          move          #lg, y1
          move          b, x1
          mpy           x1, y1, b
          mopv          b, y1
;         The following code updates the parameters:
          do            #ntaps, _update
          move          y: (r4), a     x: (r0) +, x0
          mac           x0, y1, a
          move          a, y: (r4) +
  _update
;    The following instruction updates the address register R0
          lua       (r0) -, r0
          jmp       wait2          ; Do again
          nop
          endm
```

图 9-14　AFIR、ASM 汇编宏指令实现自适应 FIR 滤波器

例：示于图 9-15 的“AFIREX、ASM”汇编程序在示范系统上实现 7 抽头和环路增益为 0.2 的自适应滤波器。注意此程序有一“include”语句参考了图 9-14 所示的 AFIR. ASM 汇编程序宏指令。

1. 关掉 PC、函数和白噪声产生器的电源。
2. 建立求和运算放大器，使函数发生器输出 s(t) 与白噪声产生器输出 w(t) 相加以形成输入信号 x(t) 。
3. 连接输入信号 x(t) 到 EVB 的“BNC2”输入并连接到示波器通道 1 输入。
4. 连接 EVB 的“BNC1”输出到示波器的通道 2 输入。
5. 加电源并引导 PC。

6. 加电到函数和白噪声产生器。
7. 把“DSP56000/1 示范软件＃4”软盘插入驱动器“A”。
8. 装入“ADS56000 用户口软件”程序。
9. 键入 load a：afirex，lod<return>，装入图 9－15 所示的 AFIREX.ASM 汇编程序的汇编和链接版本。
10. 键入 go〈return>，开始执行程序。
11. 正弦输入频率从 0.5kHz 改变到 20kHz，并观察示波器上输入和输出信号。
12. 键入 force b<return>，停止执行程序。
13. 键入 quit<return>，退出“ADS56000 用户接口软件程序。

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Adaptive FIR Example
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
        include 'afir'
;
ntaps       equ     7
lg          equ     $199999
            org     x: $0
states      dsm     ntaps       ; filter states
            org     y: $0
para        dsm     ntaps       ; patameters
            org     p: $40      ; program start address
            afir    ntaps, lg
            end
```

图 9－15　实现 9.4 节中的自适应 FIR 滤波器的 AFIREX.ASM 汇编程序

第十章　用 DSP56001 处理器的递归自适应线性增强

自适应滤波器的环境可以是确定的或随机的。在确定的环境中，自适应滤波器的输入是可测输入。第九章的自适应算法是在确定环境中自适应非递归算法的一例。在随机的环境下，某些输入是不可测的。在随机环境下自适应递归算法的一种重要应用是滤出被加性噪声污染的窄带信号。在数字信号处理（DSP）文献中，这些滤波器通常在递归自适应线性增强（RALE）的名称下进行讨论。RALE 在宽带噪声中检测和跟踪移动的谱线以增强信噪比。自适应线增强器用于以下领域，如通信、雷达、语声、声纳、控制和谱估计。

自适应随机算法由图 10－1 所示三级组成。这种算法不要求严格的实测或稳定性测试以检验其极点和零点的位置。这使三级自适应随机算法适合于在 DSP56001 处理器上实现 RALE。第一级中，自递归模型的残差用作未知噪声的估计。第二级中，自递归滑动平均（ARMA）模型适合用第一级的残差。从第二级得到的修正的残差被滤波以产生改进的噪声估计。第三级中，改善的估计用来得到较好的 ARMA 模型。

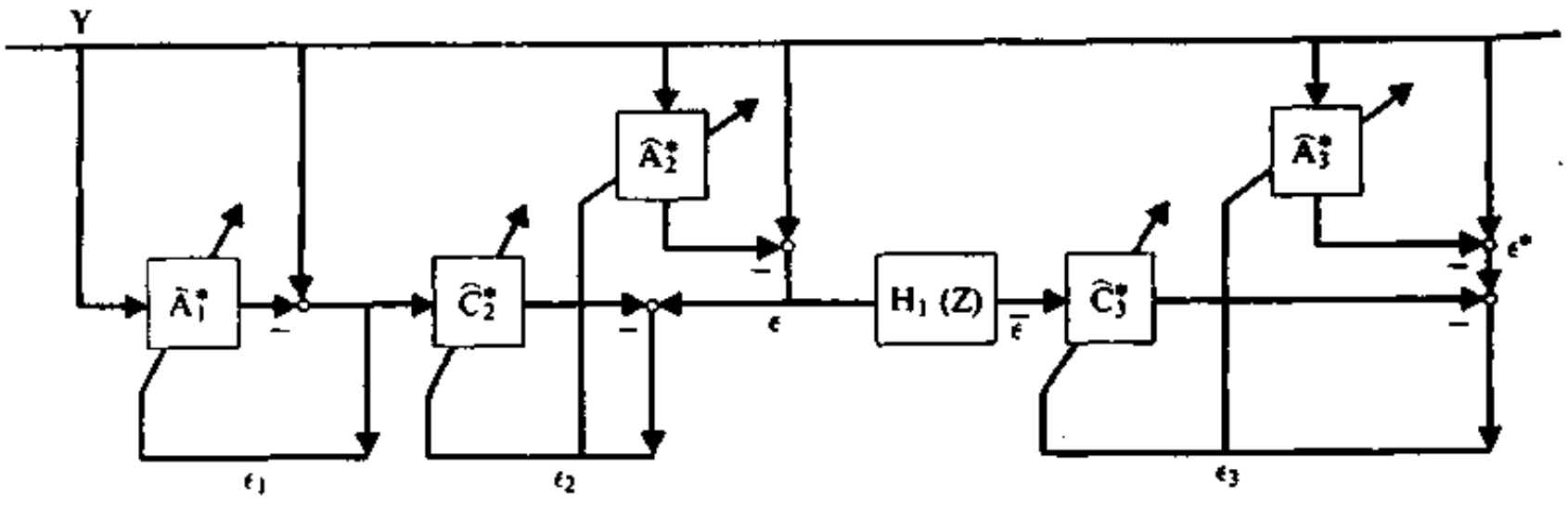


图 10－1　Rale 算法

本章先复习 RALE 算法，然后在 DSP 示范系统上实现 RALE 算法以估计 RALE 参数。所估计的参数用来计算估计谱。

10.1　RALE 算法

本章中特定的 RALE 算法可用以下 ARMA 模型表示：

$$A(q^{-1})y(k) = C(q^{-1})e(k) \tag{10-1}$$

其中 y(k) 是输入数据，e(k) 是白噪声。$A(q^{-1})$ 和 $C(q^{-1})$ 是迟延算子 q^{-1} 的 n 阶多项式：

$$A(q^{-1}) = 1 + a_1q^{-1} + \cdots + a_nq^{-n} \tag{10-2}$$

和

$$C(q^{-1}) = 1 + c_1q^{-1} + \cdots + c_nq^{-n} \tag{10-3}$$

这算法的目的是由输入数据 y(k) 估计参数 a_i 和 C_i。算法由图 10-1 所示三级组成。第一级中，考虑式(10-1) 的以下自递归表示

$$A_1(q^{-1})y(k) = e(k) \tag{10-4}$$

其中

$$A_1(q^{-1}) = A(q^{-1})/C(q^{-1}) \tag{10-5}$$

另一个算子

$$A_1^*(q^{-1}) = 1 - A_1(q^{-1}) \tag{10-6}$$

有时在下面的分析中用以代替 $A(q^{-1})$。

算法第一级的目的是得到白噪声 e(k) 的估计。用 P 次多项式 $\hat{A}_1$ 估计 A_1 来得到，其中

$$\hat{A}_1(q^{-1}) = 1 + \hat{a}_{11}q^{-1} + \cdots + \hat{a}_{1p}q^{-p} \tag{10-7}$$

或等效写为：

$$\hat{A}_1(q^{-1}) = 1 - \hat{A}_1^*(q^{-1}) \tag{10-8}$$

其中

$$\hat{A}_1^*(q^{-1}) = -(\hat{a}_{11}q^{-1} + \cdots + \hat{a}_{1p}q^{-p}) \tag{10-9}$$

以及 $\hat{a}_{1i}$ 是 a_{1i} 的估计。第一级预测输出 $\hat{y}_1(k)$ 可定义为：

$$\hat{y}_1(k) = \hat{A}_1^*(q^{-1})y(k) = \hat{\Theta}_1(k)^T\Phi_1(k-1) \tag{10-10}$$

这里

$$\hat{\Theta}_1(k)^T = [\hat{a}_{11}(k), \cdots, \hat{a}_{1p}(k)]$$

$$\Phi_1(k)^T = [-y(k), \cdots, -y(k-p+1)] \tag{10-11}$$

第一级的残差（误差信号）$\epsilon_1(k)$ 定义为第一级输入数据 y(k) 和预测输出间之差：

$$\epsilon_1(k) = y(k) - y_1(k) \tag{10-12}$$

用 (10-8) 和 (10-10) 式，(10-12) 式可变成：

$$\epsilon_1(k) = \hat{A}_1(q^{-1})y(k) \tag{10-13}$$

比较 (10-13) 和(10-4) 式，表明第一级 $\epsilon_1(k)$ 可用作白噪声 e(k) 的估计。

因为(10-1) 式的自递归表示有无限阶，故引入一偏置到估计噪声中。适当选择 P 可减小此偏置，这里 P = 2n 或 3n 通常就够了。

在算法的第二级中，白噪声 e(k) 的改进的估计用以下方法得到，用输入数据 y(k) 和第一残差 $\epsilon_1(k)$ 按下述方法去估计多项式 $\hat{A}_2$ 和 $\hat{C}_2$，这方法能很好地满足以下模型：

$$A(q^{-1})y(k) = C(q^{-1})\epsilon_1(k) \tag{10-14}$$

多项式 $\hat{A}_2$ 和 $\hat{C}_2$ 可定义为：

$$\hat{A}_2(q^{-1}) = 1 + \hat{a}_{21}q^{-1} + \cdots + \hat{a}_{2n}q^{-n}$$

或

$$\hat{A}_2(q^{-1}) = 1 - \hat{A}_2^*(q^{-1}) \tag{10-15}$$

其中

$$\hat{A}_2^*(q^{-1}) = -(\hat{a}_{21}q^{-1} + \cdots + a_{2n}q^{-n})$$

和

$$\hat{C}_2(q^{-1}) = 1 + \hat{C}_{21}q^{-1} + \cdots + \hat{C}_{2n}q^{-n}$$

或

$$\hat{C}_2(q^{-1}) = 1 - \hat{C}_2^*(q^{-1}) \qquad (10-16)$$

其中

$$\hat{C}_2^*(q^{-1}) = -(\hat{C}_{21}q^{-1} + \cdots + \hat{C}_{2n}q^{-n})$$

第二级预测输出 $y_2(k)$ 可定义为：

$$\hat{y}_2(k) = \hat{\Theta}_2(k)^T\Phi_2(k-1) \qquad (10-17)$$

其中

$$\hat{\Theta}_2(k)^T = [\hat{a}_{21}(k),\cdots,\hat{a}_{2n}(k),\hat{c}_{21}(k),\cdots,\hat{c}_{2n}(k)]$$

$$\hat{\Phi}_2(k)^T = [-y^{(k)},\cdots,-y(k-n+1),\epsilon_1(k),\cdots,\epsilon_1(k-n+1)]$$

第二级误差可定义为：

$$\epsilon_2(k) = y(k) - \hat{y}_2(k) \qquad (10-18)$$

修正误差 $e(k)$ 用新的估计多项式 A_2 定义为：

$$\epsilon(k) = \hat{A}_2(q^{-1})y(k) \qquad (10-19)$$

然后用参数 d 对修正误差滤波以得到改进的噪声估计 $\bar{\epsilon}(k,\delta)$，其中

$$\bar{\epsilon}(k,\delta) = -\sum_{i=1}^{n}\delta^i\hat{a}_{2i}\bar{\epsilon}(k-i,\delta) + \epsilon(k) = H_1(q^{-1},\delta)\epsilon(k) \qquad (10-20)$$

和

$$H_1(q^{-1},\delta) = 1/A_2(\delta q^{-1}) \qquad \text{对 } 0 \leqslant \delta < 1$$

用残差 $\bar{\epsilon}(k,\delta)$ 的特定滤波方法的原因说明如下：

用(10-19)式,(10-20)式可写为：

$$\bar{\epsilon}(k,\delta) = H(q^{-1},\delta)y(k) \qquad (10-21)$$

这里

$$H(q^{-1},\delta) = \hat{A}_2(q^{-1})/\hat{A}_2(\delta q^{-1}) \qquad (10-22)$$

这种“调谐”滤波器 $H(q^{-1},\delta)$ 选择输入数据 $y(k)$ 的噪声分量和抑制信号分量。所以，得到改进的 $e(k)$ 估计。这种滤波器或等效滤波器$[1-H(q^{-1},\delta)]$称为“陷波”滤波器。δ越接近1，此滤波器响应将越平坦。响应越平坦，$e(k)$ 的估计偏移越小。参数 δ 有时称为去偏移参数。

算法的第三级中，滤波修正的残差 $\bar{\epsilon}(k,\delta)$ 和输入数据 $y(k)$ 用来按一定方法得到改进估计 $\hat{A}_3,\hat{C}_3$，这方法适合 模型：

$$A(q^{-1})y(k) = C(q^{-1})\bar{\epsilon}(k,\delta) \qquad (10-23)$$

第三级的预测输出 $\hat{y}_3(k)$ 可定义为：

$$\hat{y}_3(k) = \hat{\Theta}_3(k)^T\Phi_3(k-1) \qquad (10-24)$$

其中

$$\hat{\Theta}_3(k)^T = [\hat{a}_{31}(k),\cdots,\hat{a}_{3n}(k),\hat{C}_{31}(k),\cdots,\hat{C}_{3n}(k)]$$

$$\Theta_2(k)^T = [-y(k),\cdots,-y(k-n+1),\bar{\epsilon}(k,\delta),\cdots,\bar{\epsilon}(k-n+1,\delta)]$$

第三级的误差可定义为：

$$\epsilon_3(k) = y(k) - \hat{y}_3(k) \qquad (10-25)$$

用超弓式接线（supermartingale）法，此三级参数自适应算法可以给出如下：

$$\hat{\Theta}_i(k+1) = \hat{\Theta}_i(k) + F_i(k)\Phi_i(k)\varepsilon_i(k+1) \qquad i = 1,2,3 \qquad (10-26)$$

其中

$$F_i^{-1}(k+1) = \lambda_1(k)F_i^{-1}(k) + \lambda_2(k)\Phi_i(k)\Phi_i(k)^T$$

$$F_i(0) > 0 \qquad 0 \leqslant \lambda_1(k) < 1 \qquad 0 \leqslant \lambda_2(k) < 2$$

不像其他算法，这种三级自适应随机算法不要求严格的实际测试或稳定测试去检验极点和零点位置。

10.2 在 DSP 示范系统上实现 RALE

RALE 可用第五章所述的 DSP 示范系统来说明。DSP 系统由 DSP56000ADS 应用开发系统（ADS）和评价板（EVB）上的 A/D 变换器组成。模拟输入信号 $y(t) = x(t) + v(t)$。

数字化输入信号 $y(k)$ 可写为：

$$y(k) = x(k) + v(k) \qquad (10-27)$$

因为数字正弦输入 $x(k)$ 可表示为：

$$x(k) = u(k)/A(q^{-1}) \qquad (10-28)$$

其中

$$A(q^{-1}) = 1 + a_1q^{-1} + a_2q^{-2}$$

$u(k)$ 是未知的白噪声过程。

用（10-28）式，（10-27）式可写为：

$$A(q^{-1})y(k) = u(k) + A(q^{0-1})v(k) \qquad (10-29)$$

或等效地写为：

$$A(q^{-1})y(k) = C(q^{-1})e(k) \qquad (10-30)$$

其中

$$C(q^{-1})e(k) = u(k) + A(q^{-1})v(k) \qquad (10-31)$$

注意（10-30）式准确地与（10-1）式相同。

用（10-26）式，参数自适应算法可写为：

$$\hat{\theta}_{li}(k+1) = \hat{\theta}_{li}(k) + au_{li}(k)\varepsilon_i(k+1)/bt_i(k) \qquad (10-32)$$

其中

$$au_{li}(k) = \sum_{j=1}^{4} f_{lji}(k)\hat{\theta}_{ji}(k)$$

$$u_{li}(k) = \sum_{j=i}^{4} f_{jli}(k)\hat{\theta}_{ji}(k)$$

$$bt_i(k) = \sum_{l=1}^{4} \hat{\theta}_{li}(k)au_{li}(k)$$

和

$$f_{lji}(k+1) = f_{lji}(k) - au_{li}(k)u_{ji}(k)/bt_i(k) \qquad (10-33)$$

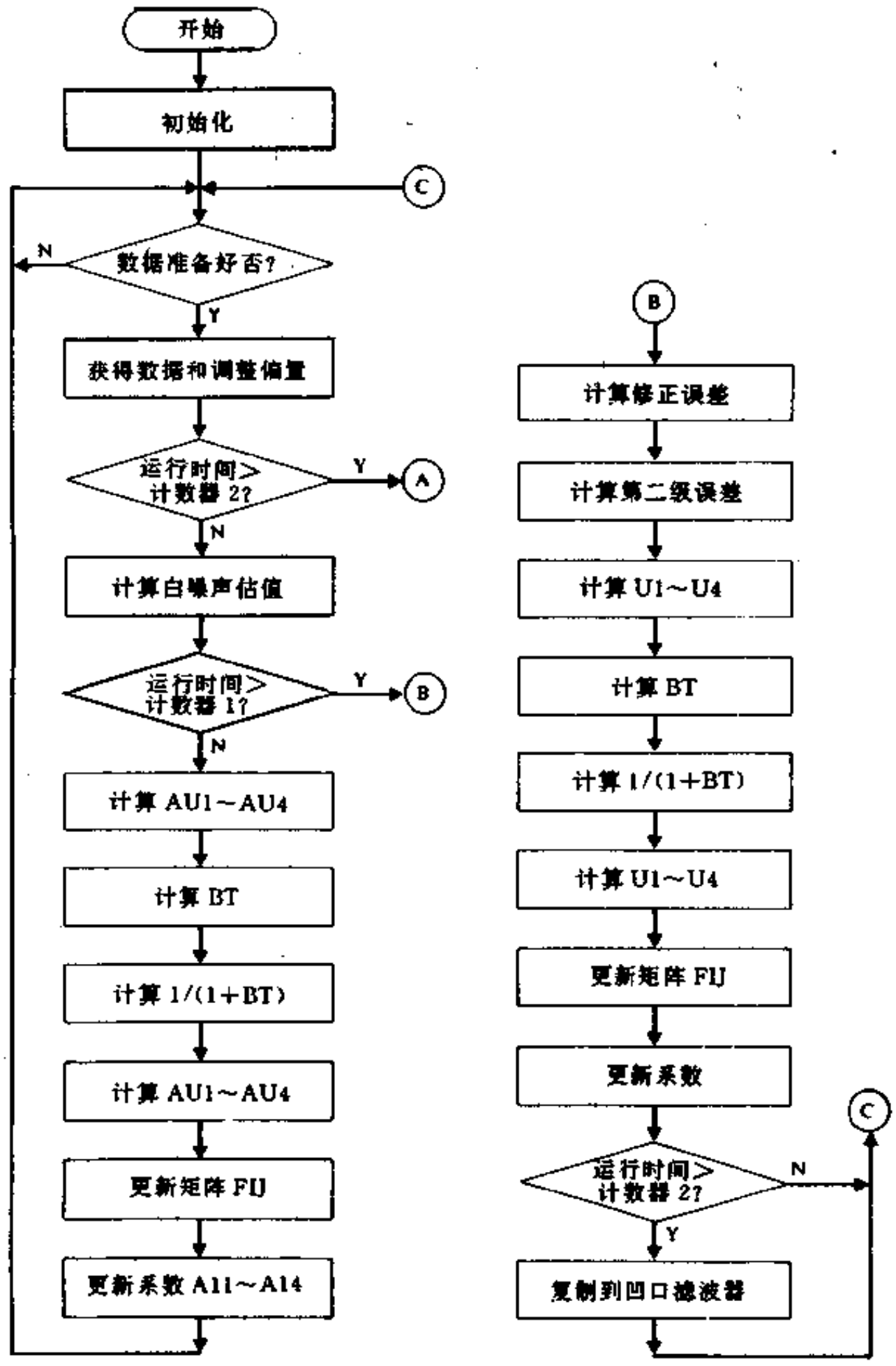
$$l = 1,\cdots,4 \qquad j = 1,\cdots,4 \qquad 和\ i = 1,2,3$$

$\hat{\theta}_{ji}$ 和 f_{lji} 分别是 $\hat{\Theta}_i$ 和 F_i 的元素。RALE 算法的流程图示于图 10-2。

RALE.LOD绝对装入程序可用来从输入数据y(k)估计多项式$A(q^{-1})$和$C(q^{-1})$的系数。然后用估计的参数计算估计谱。

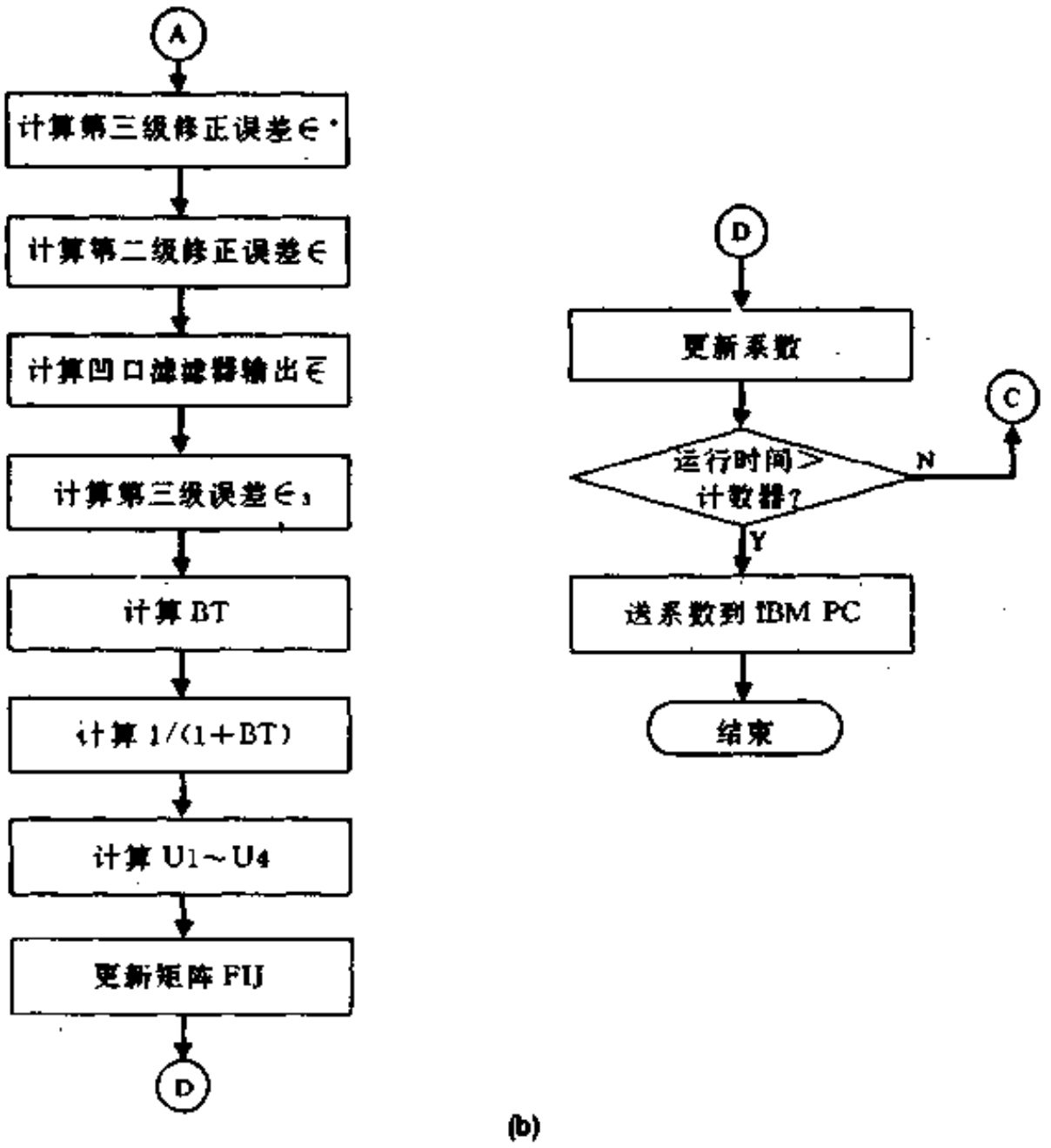
1. 关掉PC机和函数与白噪声产生器的电源。

2. 在EVB上移去跳线JPI，装上跳线JPI和一个2MHz晶体，使EVB在输出采样率15.625kHz上工作。输出采样率等于输入时钟频率被128除（2MHz/128=15.625kHz）。



(a)

图10-2（a） Rale算法流程



(b)

图 10-2 (b)　Rale 算法流程图

3. 函数产生器的输出 X(t) 加到白噪声产生器输出 V(t) 上，产生输入信号 Y(t)。
4. 输入信号连接到 EVB 的“BNC2”输入(A/D 输入)，并连到示波器通道 1 输入。
5. 加电并引导 PC 机。
6. 加电给函数和白噪声产生器。
7. 将正弦输入频率调到 500Hz。
8. 把“DSP56000/1 示范软件#4”软盘插入驱动器“A”。
9. 装入“ADS56000 用户接口软件”程序。
10. 键入 load a：rale. lod<return>，装入 RALE 汇编程序的汇编和链接版本。
11. 键入display off <return>

    break #1 p：$257<retrrn>

    go<return>

    执行 RALE 程序。
12. 用几秒钟收敛 RALE 参数，然后键入：

    force b<return>

    radix f y：$108.. $10b<return>

    display y：$108.. $10b<return>

    显示所估计的系数 a1、a2、c1 和 c2。

13. 键入 quit<return>，退出“ADS56000 用户接口软件”程序。

10.3 在 PC 屏幕上画估计的谱

估计的参数 $\hat{a}_{3i}$ 和 $\hat{c}_{3i}$ 按以下关系用来计算估计谱 Sy(ω,k)：

$$Sy(\omega,k) = \hat{C}_3(q^{-1},k)\hat{C}_3(q,k)/\hat{A}_3(q^{-1},k)\hat{A}_3(q,k) \qquad q = e^{j\omega} \qquad (10-34)$$

用 DSP56000/1 示范软件 #4 软盘上的 SPECT 程序可计算估计谱 Sy(w,k) 并显示在 PC 屏幕上。注意程序要求 VGA 显示适配器。否则，只有估计频率的值显示在屏幕上。

1. 把“DSP56000/1 示范软件 #4 软盘”插入驱动器“A”。

2. 键入 a:'spect<return>。

以下显示将出现在屏幕上

```
Input Sampling Frequency:
```

3. 键入 15625<return>作为输入采样频率。

屏幕显示

```
Input a1, a2, c1, c2:
```

4. 键入 −0.4035406，−0.3607159，0.1752567，−0.0798273<return>分别作为 a1、a2、c1 和 c2 的输入。估计谱将显示于屏幕。

第十一章　用 DSP56001 处理器设计谱分析器和自适应差分脉冲编码调制器（ADPCM）

在第一个设计中，用 DSP56001 处理器实现谱分析器，虽然 EVB 上的 A/D 转换器能以 100kHz 的最大速率产生 16 位数字化采样，但用于实现谱分析器的最大采样精度（位数）与数据采样数 N 有关，这里 N 必须是 2 的幂，N 数据采样被 Hamming 窗乘来改善谱内容。谱用 N 点 FFT 计算，实现谱分析器的 DSP56001 指令已被叙述并包括在 DSP56000/1 示范软件 # 3 软盘中。

在第二个设计中，说明在 ADPCM 计算系统上实现自适应差分脉冲编码调制（ADPCM）系统，这在电话应用中对于减少传送话音信号所需比特数是很有用的。ADPCM 评价板的主要部分是 DSP56001 处理器和 MC145503 PCM 编译码器。ADPCM 评价板的框图示于图 11－6。ADPCM 设计留给学生作为练习。

11.1　用 DSP56001 处理器设计谱分析器

此设计用 DSP56001 处理器计算和显示输入数据 1024 点采样的谱。

11.1.1　用输入数据采样的最大精度

数字化输入样值可被 2^G 的因子除，以限制输出谱幅值小于或等于 1，这里 G 小于或等于 log2（FFT 点数）因为 EVB 上 A/D 转换器提供 15 位精度，而 DSP56001 处理器提供 24 位精度，G 可选 得小于或等于 24－15＝9。在此设计 中 G＝7。

11.1.2　加窗

为了计算谱，谱分析器算法仅用数字化输入信号 N 个样值，这里限制 N 个样值就称为加窗。样值的限制就等效于输入信号被矩形窗序列 w(n) 乘，这里 w(n) ＝ 1 对 $0 \leqslant n \leqslant N-1$，而 w(n) ＝ 0 对其他。但是采样的限制给谱引入了误差。为了减少误差对谱的影响，可用几种窗函数之一。这里考虑 Hamming 窗。因果 Hamming 窗函数可写为：

$$w(n) = 0.54 - 0.46 * \cos(2\pi n/N) \qquad (11-1)$$

Hamming. ASM 汇编宏指令示于图 11－1，用来产生 N 点 Hamming 窗。

11.1.3　用 DSP56001 处理 器实现谱分析器

1. 下面所列指令从 A/D 转换器中读实部数据，并以因子 2^{-} scldw 降低实部数据。因为虚部数据在此应用中不被 A/D 转换器所采用，但第八章所研究的实部和虚部数据 FFT 用于此应用中，故 FFT 所要的虚部数据清除到 0。实部和“零”虚部存在由 R0 所指示的 X 和 Y 存贮器单元。

```
get1        do      #points,end - ld
;read A/D
wait        jclr    # $7,x: $FFEE,wait
            movep   x: $FFEF,a
;scale thereal part op data down
;
            dup     scldw
            asr     a
            endm
            move    al,x:(R0) +
end: ld
;
;clear the imaginary part of data
;
clri        move    # $0,a
            move    #imagm,r0
            do      #points,end - clr
            move    al,y:(r0) +
end - clr   nop
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Hamming Window
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
hamming     macro   points,coef
;
; points - number of points
; coef   - base address for window table
;
;
;
pi2         equ     3.141592654
freq2       equ     2.0*pi2/@cvf(points)      ;freq = 2 pi/points
            org     p:coef
count       set     0
            dup     points              ; count = 0, . . . ,points - 1
            dc      0.54 - 0.46*@cos(@cvf(count)*freq2)
count       set     count + 1
            endm
            endm                        ; end of hamming window macro
```

图 11-1　产生 N 点 Hamming 窗的 HAMMING.ASM 汇编宏指令

2. SINCOS1.ASM 汇编宏指令为 FFT 产生 sine 和 cosine 表。

```
;
; Sine - Cosine Table Generator for FFTs
;
;
sincos1     macro points,coef
;
; points - number of points
; coef   - base address of sine/cosine table
```

```
;       - cosine value in X memory
;       - negative sine value in Y memory
;
;
pi1     equ     3.141592654
        fcpq1   2.0*pi1/@cvf(points)        ; freq = 2 pi/points
        org     x:coef                      ; Calculation of
        setnt   0                           ; cos(freq*count) for
        dup     points/2                    ; count = 0,...,(points/2) - 1
        dc      @cos(@cvf(count)*freq1)
        setnt   count + 1
        endm
        org     y:coef                      ; Calculation of
count   set     0                           ; - sin(freq*count) for
        dup     points/2                    ; count = 0,..,(points/2) - 1
        dc      - @sin(@cvf(count)*freq1)
count   set     count + 1
        endm
        endm                                ; end of sincos macro
```

3. 以下所列指令用 Hamming 窗乘 N 点输入数据样本，Hamming 窗的系数存在地址寄存器 R1 所指示的 P 存贮器单元中。实部数据存在由 R0 所指示的 X 存贮器单元中。

```
ham         move        #realm,r0
            move        #coef,r1
            move        #$FFFFFF,m0
            move        m0,m1
            do          #points,end_ham
            move        x:(r0),x0
            movem       p:(r1) + 1,x1
            mpy         x1,x0,a
            move        a,x:(r0) +
end_ham     nop
```

4. 以下所列指令计算 FFT，其输入序列，是正常序列而输出序列是位倒置序列。

```
stage       do          #nstage,end_stage
            move        #$0,r0
            move        #coef,r5
            move        #$0,r2
            move        n1,n3
            move        n1,n4
tfs         do          n2,end_tfs
            move        r2,r3
            lua         (r2) + n2,r4
tf          do          n0,end_tf
            move        x:(r3),a        y:(r4),y1
            move        x:(r4)x1        y:(r3),b
            sub         x1,a
            sub         y1,b            a,x0
```

```
                move        x:(r3),a        b,y0
                add         x1,a            y:(r3),b
                add         y1,b            a,x:(r3)
                move        x:(r5),x1       b,y:(r3) + n3
                mpy         x0,x1,a         y:(r5),y1
                macr        - y0,y1,a
                mpy         x0,y1,b         a,x:(r4)
                macr        y0,x1,b
                move        b,y:(r4) + n4
end_ tf
                lua         (r0) + n0,r0
                lua         (r2) + ,r2
                move        r0,r5
                nop
                lua         (r5) + n5,r5
end_ tfs
                move        n2,n1
                move        n2,b
                lsr         b               n0,a1
                lsl         a               b1,n2
                move        a1,n0
end_ stage nop
```

5. 下面的指令计算 FFT 幅值的平方，并以 2**sclup 的因子升标。

```
                move        #realm,r0
                move        #imagm,r1
                move        # $ 0,n0
                move        # $ 0,n1
                move        # $ FFFFFF,m0
                move        # $ FFFFFF,m1
mag             do          #points,end_ mag
                move        x:(r0),x0
                mpy         x0,x0,a
                move        y:(r1) + ,y0
                mac         y0,y0,a
                dup         sclup
                asl         a
                endm
                rnd         a
                move        a,x:(r0) +
end_ mag        nop
```

6. 以下指令以正常次序传送 FFT 输出值到 D/A 转换器，DSP56001 处理器的反进位寻址能力用来使输出序列元素自动成为正常次序。

```
write           move        #realm,r0
                move        #points/2,n0
                move        #0,m0
                do          #points,end_ wr
```

```
wait2       jclr        # $6,x:#FFEE,wait2
            movep       x:(r0)+n0,x:$FFEF
            rep         #delay
            nop
end_wr      nop
```

11.1.4 例:1024点谱分析器

用图11-2所示的SPECTRUM.ASM汇编程序实现1024点谱分析器,这里升标因子是4,而降标因子为7。

1. 去掉PC机和函数产生器电源。
2. 连接函数产生器输出 到EVB的"BNC2"输入和示波器通道1输入。
3. 连接EVB的"BNC1"输出到示波器通道2输入。
4. 加电和引导PC机。
5. 加电到函数产生器。
6. 插入"DSP56000/1示范软件 #3"软盘到驱动器"A"。
7. 从驱动器"B"或硬盘调用"ADS56000用户接口软件"程序。
8. 键入load a:spectrum.lod <retun>,装入图11-2所示汇编程序的汇编和链接版本。
9. 键入change PC $800 <return> 和go <retun> 开始执行程序。
10. 将正弦输入频率从0.5kHz调到20kHz,并在示波器上观察输入和输出信号谱。
11. 键入force b <return>,停止执行程序。
12. 键入quit <return>,退出"ADS56000用户接口软件"程序。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
; Spectrum Analyzer
;
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
points      equ     1024        ; fft size
nstage      equ     10          ; number of stages = log2(number of points)
coef        equ     $400        ; base address for sine, cos and hamming
scldw       set     $7          ; scale input down
sclup       set     #4          ; scale output up
delay       equ     $d2         ; delay in output between pts
            org     x:$0
realm       dsm     points
            include 'sincosl.asm'
            include 'hamming.asm'
;Program start address
            org     p:$100
            sincosl points,coef
            hamming points,coef
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Initialize routine
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
START       reset
init2       movep   #0,x:$FFEF
            movep   #0,x:$FFEF
            movep   #0,x:$FFFF
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Set up the SSI for operation with EVB
; The following code sets port C to function as SSI
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pcc         equ       $0001e0
            movep     #pcc,x:$FFE1   ; wite PCC
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; The following code sets the SSI CRA
; and CRB control registers
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
cra         equ       $004000
            movep     #cra,x:$FFEC
crb         equ       $003200
            movep     #crb,x:$FFED
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Read A/D and scale the data down
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; read A/D
get2        move      #realm,r0
            move      #$0,N0
            move      #$FFFFFF,m0
get1        do        #points,end_ld
wait        jclr      #$7,x:$FFEE,wait
            movep     x:$FFEF,a
;
; scale the real part of data down
;
            dup       scldw
            asr       a
            endm
            move      a1,x:(r0)+
end_ld      nop
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Clear the imaginary part of data
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
clri        move      #$0,a
            move      #imagm,r0
            do        #points,end_clr
            move      a1,y:(r0)+
end_clr     nop
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Multiply by hamming windown
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
ham         move      #realm,r0
            move      #coef,r1
            move      #$FFFFFF,m0
            move      m0,m1
            move      #1,n1
            do        #points,end_ham
            move      x:(r0),x0
            movem     p:(r1)+n1,x1
            mpy       x1,x0,a
            move      a,x:(r0)+
end_ham     nop
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Init registers for fft
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            move    # $200,m0
            move    # $FFFFFF,m1
            move    m1,m2
            move    #1023,m3
            move    m3,m4
            move    m1,m5
            move    #coef,n5
            move    #(points/2),n2
            move    #points,n1
            move    # $1,n0
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Compute fft
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
stage       do      #nstage,end_ stage
            move    # $0,r0
            move    #coef,r5
            move    # $0,r2
            move    n1,n3
            move    n1,n4
tfs         do      n2,end_ tfs
            move    r2,r3
            lua     (r2) + n2,r4
tf          do      n0,end_ tf
            move    x:(r3),a        y:(r4),y1
            move    x:(r4),x1       y:(r3),b
            sub     x1,a
            sub     y1,b            a,x0
            move    x:(r3),a        b,y0
            add     x1,a            y:(r3),b
            add     y1,b            a,x:(r3)
            move    x:(r5),x1       b,y:(r3) + n3
            mpy     x0,x1,a         y:(r5),y1
            macr    - y0,y1,a
            mpy     x0,y1,b         a,x:(r4)
            macr    y0,x1,b
            move    b,y:(r4) + n4
end_ tf
            lua     (r0) + n0,r0
            lua     (r2) + ,r2
            move    r0,r5
            nop
            lua     (r5) + n5,r5
end_ tfs
            move    n2,n1
            move    n2,b
            lsr     b               n0,a1
            lsl     a               b1,n2
            move    a1,n0
end_ stage  nop
```

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Compute fft Magnitude
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            move    #realm,r0
            move    #imagm,r1
            move    #$0,n0
            move    #$0,n1
            move    #$FFFFFF,m0
            move    #$FFFFFF,m1
mag         do      #points,end_mag
            move    x:(r0),x0
            mpy     x0,x0,a
            move    y:(r1)+,y0
            mac     y0,y0,a
            dup     sclup
            asl     a
            endm
            rnd     a
            move    a,x:(r0)+
end_mag             nop
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Pulse output to trigger scope
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pulse       move    #$0,a
            do      #$4,l_pls
w_pls       jclr    #$6,x:$FFEE,w_pls
            movep   a1,x:$FFEF
l_pls       move    #$27cd00,a
            neg     a
            do      #$4,12pls
w2pls       jclr    #$6,x:$FFEE,w2pls
            movep   a1,x:$FFEF
12pls       nop
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Write to D/A
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
write       move    #realm,r0
            move    #points/2,n0
            move    #0,m0
            do      #points,end_wr
wait2       jclr    #$6,x:$FFEE,wait2
            movep   x:(r0)+n0,x:$FFEF
            rep     #delay
            nop
end_wr      nop
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Clear output
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
clout       move    #$0,a
w_clout     jclr    #$6,x:$FFEE,w_clout
            movep   a1,x:$FFEF
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Loop Indefinitely
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
again       jmp     get2
            reset
            end
```

图 11-2　SPECTRUM.ASM 汇编程序

11.2 用 DSP56001 处理器设计 ADPCM

在此设计中，不像谱分析器设计，而只提出 足够的信息使之能开始设计 ADPCM。ADPCM 的完成留作练习。

ADPCM 是一种去相关的方法，可减少信号中的多余度。因为话音信号通常有很高相关性，用 ADPCM 系统可减小其动态范围。所以减少了比特数，可传输话音信号而不降低分辨水平。例如，用相同数目的电话线，给每路话配置的带宽减少使总的话路数增加。这里所讲的 ADPCM 系统能减少一半话路带宽。

在图 11－3 所示的 ADPCM 评价系统中，MC145503 CODEC 的编码器部分用来数字化“发送的”话音信号，而 MC145503 CODEC 的译码器部分用来恢复“接收的”话音信号。名称“CODEC 是 CODER”和“DECODER”合并而成，MC145503 CODEC 所用的信号编码方法是脉冲编码调制（PCM）。

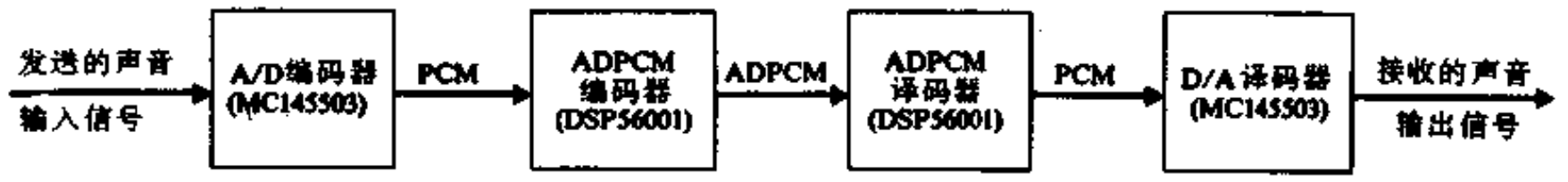


图 11－3　ADPCM 评价系统

为了保持工业标准的话音信号质量，数字化话音信号在 40dB 动态范围，应至少有 30dB 信号 失真比。此指标可用线性 13 比特 A/D 编译码器达到，但在振幅低于最大振幅超过 40dB 时，13 比特 线性编译码远远超过信号失真比指标 。因为过高的性能是不需要的，可用两种数据压缩方法（压扩法）把 13 比特压缩到 8 比特而仍满足指标。Mμ－225 律压扩方法是用于北美的标准。A 律压扩方法是欧洲的标准。MC145503 CODEC 提供 Mμ－255 律和 A 律两种压扩方法。这里考虑 Mμ－255 律压扩方法。样本以比特串行格式从 MCI45503 CODEC 编码器部分传送到 ADPCM 编码器部分，传输速率为 $8f_s$ b/s，这里 f_s 是采样频率。

图 11－4 所示是 ADPCM 编码系统的流程图。所接收的已压扩 PCM 信号，用位于 DSP56001 处理器的片上 X 存贮器中的 Mμ 律扩展表 ROM，首先把信号变为线性数字化信号。然后从线性信号中减去一估价信号以得到误差信号。误差信号被自适应增益提升，其结果量化到 4 比特。ADPCM 编码器的输出比特率降至 $4f_s$。这 4 比特样本的幅度用在自适应增益调整中以防止饱和。自适应滤波器内有一个反量化的反馈环恢复误差的量化形式，由此滤波器能得到输入话音信号的估计。

用于反馈环中的量化器和自适应滤 波器硬件也用在 ADPCM 译码系统中，ADPCM 译码器的流程图示于图 11－5。

ADPCM 评价系统的框图示于图 11－6，其中，MC145503 PCM 编码器用于使输入话音信号数字化及恢复输出话音信号。三个 8K×8b、“快”静态 MCM6264 RAM 用来作为外部 P 存贮器以避免由“慢”EPROM 运行 ADPCM 程序。硬件复位电路调用 DSP56001 处理器的片上引导程序以装入存在 2764 EPROM 之一的监视程序。DSP56001 处理器执行监视程序，使 ADPCM 应用程序从“慢”EPROM 向下加载到“快”外部静态 RAM。然后 DSP56001 处理器执行 ADPCM 应用程序。

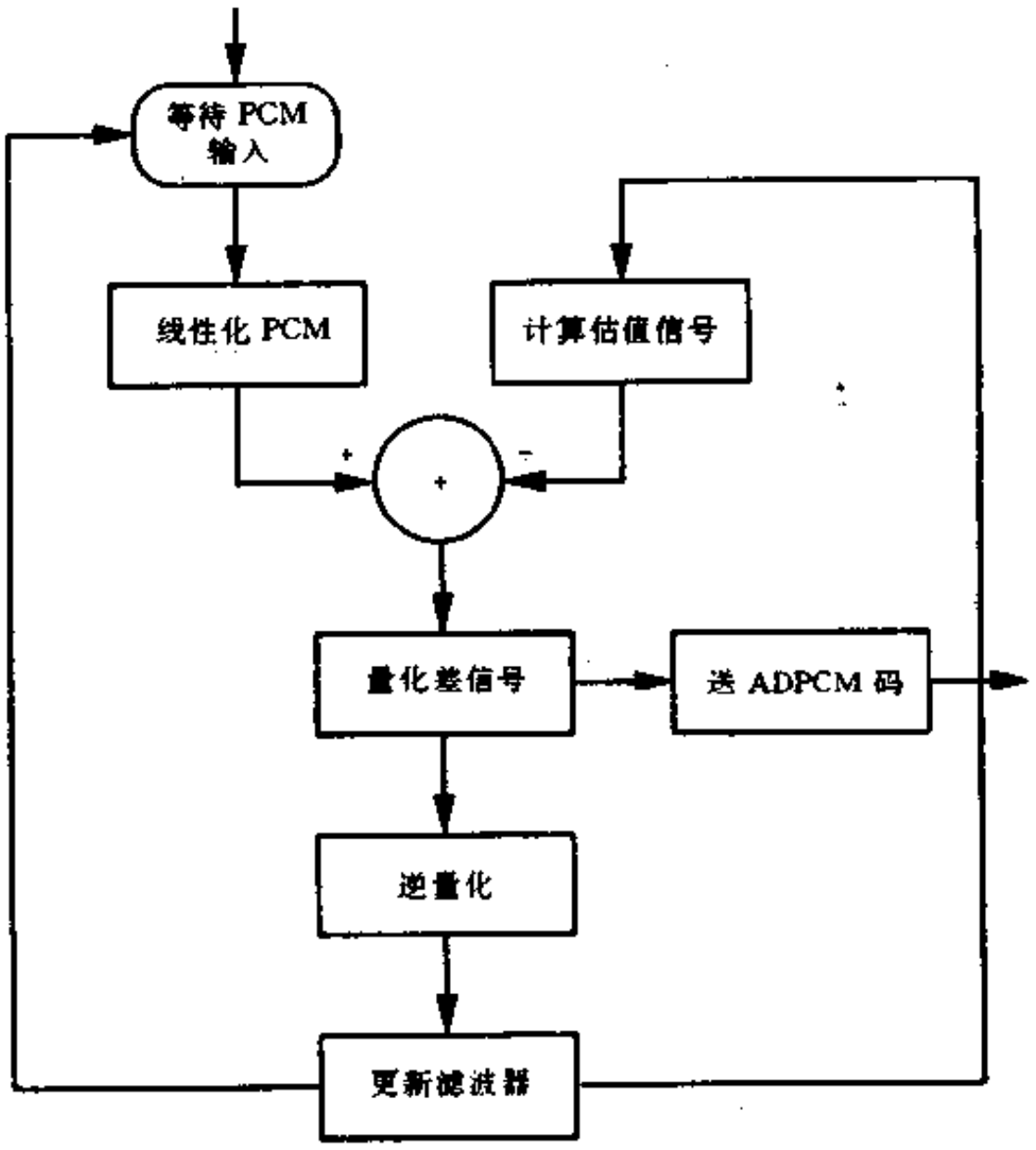


图 11-4　ADPCM 编码器

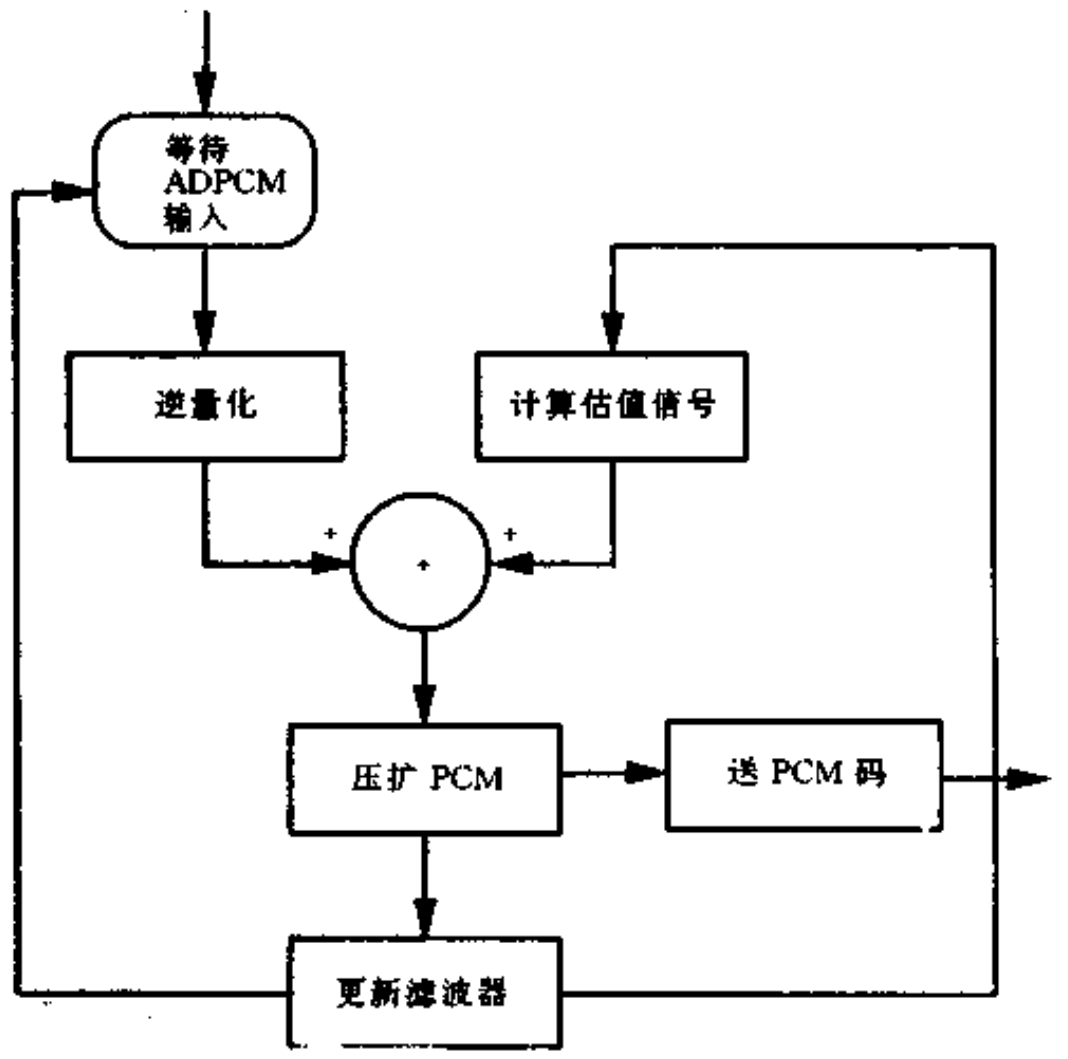


图 11-5　ADPCM 译码器

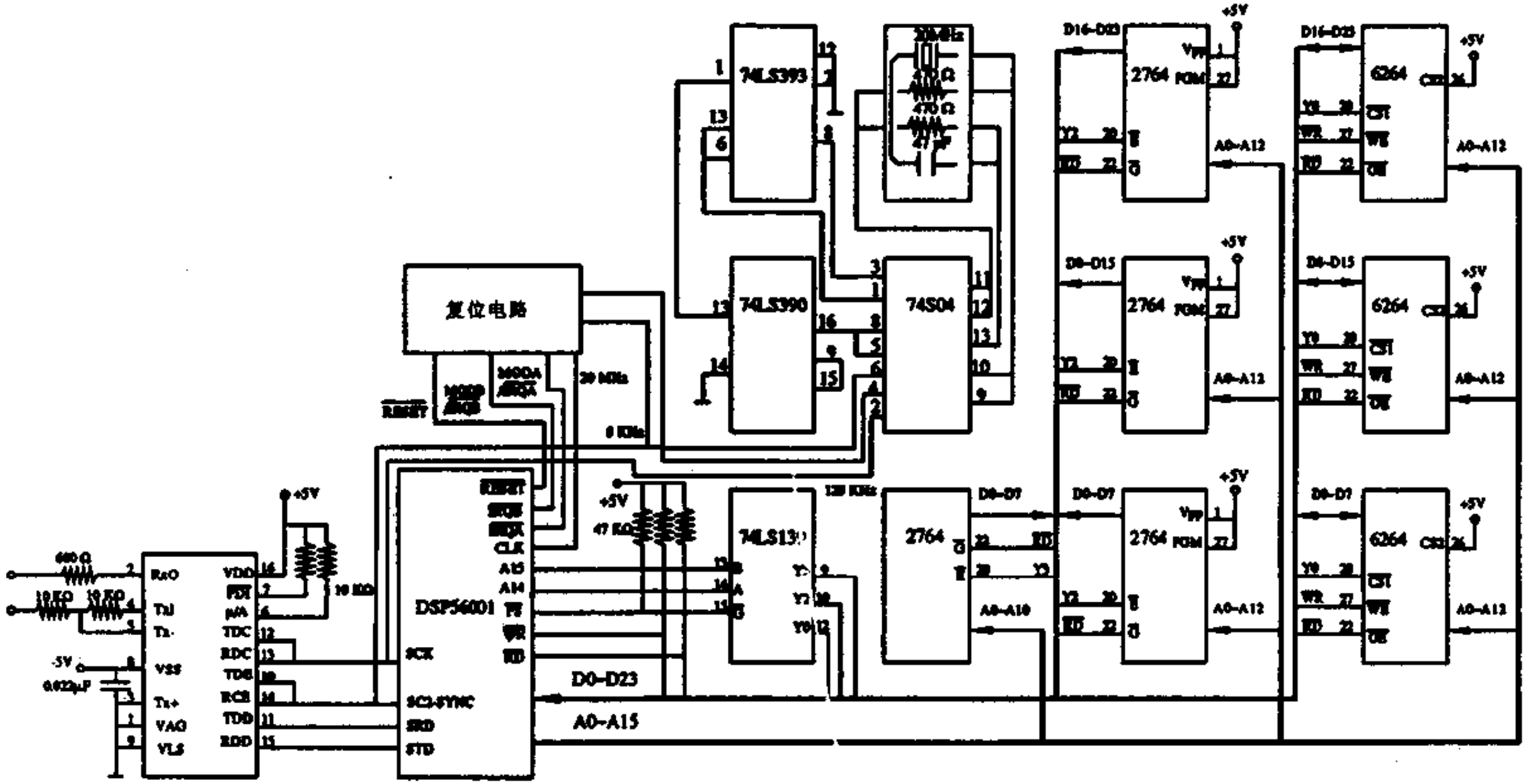


图 11－6　ADPCM 评价板

第二篇　DSP96002 浮点双端口处理器用户手册

第十二章　DSP96002 概述

本篇叙述 IEEE 浮点双端口可编程 CMOS 处理器族中的第一个成员。族的概念确定了一个核心，如数据 ALU、地址产生单元、程序控制器和有关指令集。但片上程序存贮器、数据存贮器和外围支持的很多扩展应用以及最小化系统规模和电源消耗等都未列入核心部分。

DSP96002 支持 IEEE 754 单精度（8 位指数和 24 位尾数）和单扩展精度（11 位指数和 32 位尾数）浮点和 32 位有符号和无符号的定点算法，与 2 个外部存贮器扩展端口耦合。DSP96002 特性如下：

- IEEE 745 标准 SP（32 位）和 SEP（44 位）运算。
- 16.5×10^6 条指令/s（Mips），时钟为 33MHz。
- 49.5×10^6 条浮点指令/s（MFLOPS），时钟为 33MHz。
- 单周期 32×32 位并行乘法器。
- 高度并行指令集有独特的 DSP 寻址方式。
- 嵌套的硬件 DO 循环。
- 快速自动返回中断。
- 2 个独立的片上 512×32 位数据 RAM。
- 2 个独立的片上 1024×32 位数据 ROM。
- 离片数据存贮器扩展到 2×2^{32} 位字。
- 片上 1024×32 位程序 RAM。
- 片上 64×32 位引导程序 ROM。
- 离片程序存贮器扩展到 2×2^{32} 位字。
- 2 个相同的外部存贮器扩展端口。
- 2 个 32 位并行主机 MPU/DMA 接口。
- 片上 2 路 DMA 控制器。
- 片上仿真器。

第十三章　信号说明和总线操作

13.1　引出端说明

DSP96002的各引出端配置的综合示于图13-1，其功能信号组示于图13-2，并在以下各节中说明。

13.1.1　封装

DSP96002按223条引出端封装，包括176条信号端（其中5条空闲），17条电源端和30条接地端。全部封装资料参见数据表。

13.1.2　中断和模式控制（4条引出端）

1. $\overline{\text{RESET}}$（复位）

低电平有效Schmitt触发输入。$\overline{\text{RESET}}$由输入时钟（CLK）内同步。此时片子置于复位状态，且内部相位产生器复位。Schmitt触发输入允许慢上升输入（如电容器充电）以可靠地复位片子。若$\overline{\text{RESET}}$不同步于输入时钟（CLK），则要保证准确起始定时，使多处理器同步起动和一起工作在“锁定状态”。当$\overline{\text{RESET}}$端不确定时，片子起始工作方式由MODA、MODB和MODC端锁定。

2. MODA/$\overline{\text{IRQA}}$（方式选择A/外部中断请求A）

有效低输入，内同步於输入时钟（CLK）。当硬件复位时，MODB/$\overline{\text{IRQA}}$选择片子起始工作方式，并变成一电平敏感的或负边触发器，当正常指令处理时可屏蔽中断请求输入。MODA、MODB和MODC选择8个片子起始工作方式之一，当$\overline{\text{RESET}}$端不确定时，锁定到操作方式寄存器（OMR）。若$\overline{\text{IRQA}}$是确定同步到输入时钟（CLK），用WAIT指令并确定$\overline{\text{IRQA}}$，多处理器可被重新同步以退出等待状态。若处理器在STOP等待状态，且$\overline{\text{IRQA}}$确定，处理器将退出STOP状态。

3. MODB/$\overline{\text{IRQB}}$（方式选择B/外部中断请求B）

有效低输入，对输入时钟内同步。MODB/$\overline{\text{IRQB}}$选择初始的片操作方式（当硬件复位时）并变成电平敏感的或负边触发器，当正常指令处理时可屏蔽中断请求输入。MODA、MODB和MODC选择8种初始的片操作方式之一，当复位端不确定时，锁定在操作方式寄存器（OMR）中。若$\overline{\text{IRQB}}$确定与输入时钟（CLK）同步，用WAIT指令并确定$\overline{\text{IRQB}}$，多处理器可被重新同步以退出等待状态。

4. MODC/$\overline{\text{IRQC}}$（方式选择C/外部中断请求C）

有效低输入，对输入时钟（CLK）内同步。当硬件复位时，MODC/$\overline{\text{IRQC}}$选择初始的片操作方式并变成电平敏感的或负边触发器，当正常指令处理时，可屏蔽中断请求输入。MODA、MODB和MODC选择8种初始片操作方式之一，当$\overline{\text{RESET}}$端不确定时，锁定在操作方式寄存

器（OMR）中。若$\overline{\text{IRQC}}$确定，同步于输入时钟（CLK），用 WAIT 指令并确定$\overline{\text{IRQC}}$，多处理器可被重新同步以退出等待状态。

CPU 引出端	引出端数	端口 A/B	引出端数 每个端口	引出端数 两个端口
复位和 IRQ	4	数据总线	32	64
时钟输入	1	地址总线	32	64
OnCE 端口	4	数据电源	2	4
CPU 备用	1	数据接地	4	8
静噪电源	4	地址电源	2	4
静噪接地	4	地址接地	4	8
合计	18	合计	76	152
电源/接地面		端口 A/B		
封装噪声电源面	2	总线控制信号	17	34
封装噪声接地面	5	总线控制备用	2	4
封装静噪电源面	1	总线控制电源	1	2
封装静噪接地面	1	总线控制接地	2	4
合计	9	合计	22	44
		总计引出端 223		

图 13－1　DSP96002 功能组引出端配置

13.1.3　电源和时钟（39 条引出端）

1. CLK（时钟输入）

有效高输入，高频处理器时钟。频率是指令速率的两倍。内部相位产生器把 CLK 分为 4 相（t0，t1，t2 和 t3），它们是基本的指令执行周期。有选择地产生 tw 相位把等待状态（ws）插入指令执行，将 t2 和 tw 相位配对形成等待状态，见图 13－3。CLK 应以 46～54%的占空因数连续。

2. Quiet Vcc（4）（电源）

对 CPU 逻辑的隔离电源。

3. Quiet Vss（4）（接地）

对 CPU 逻辑的隔离地。

4. Address Bus Vcc（4）（电源）

对地址总线 I/O 驱动器的隔离电源。

5. Address Bus Vss（8）（接地）

地址总线 I/O 驱动器的隔离地。

6. Data Bus Vcc（4）（电源）

数据总线 I/O 驱动器的隔离电源。

7. Data Bus Vss（8）（接地）

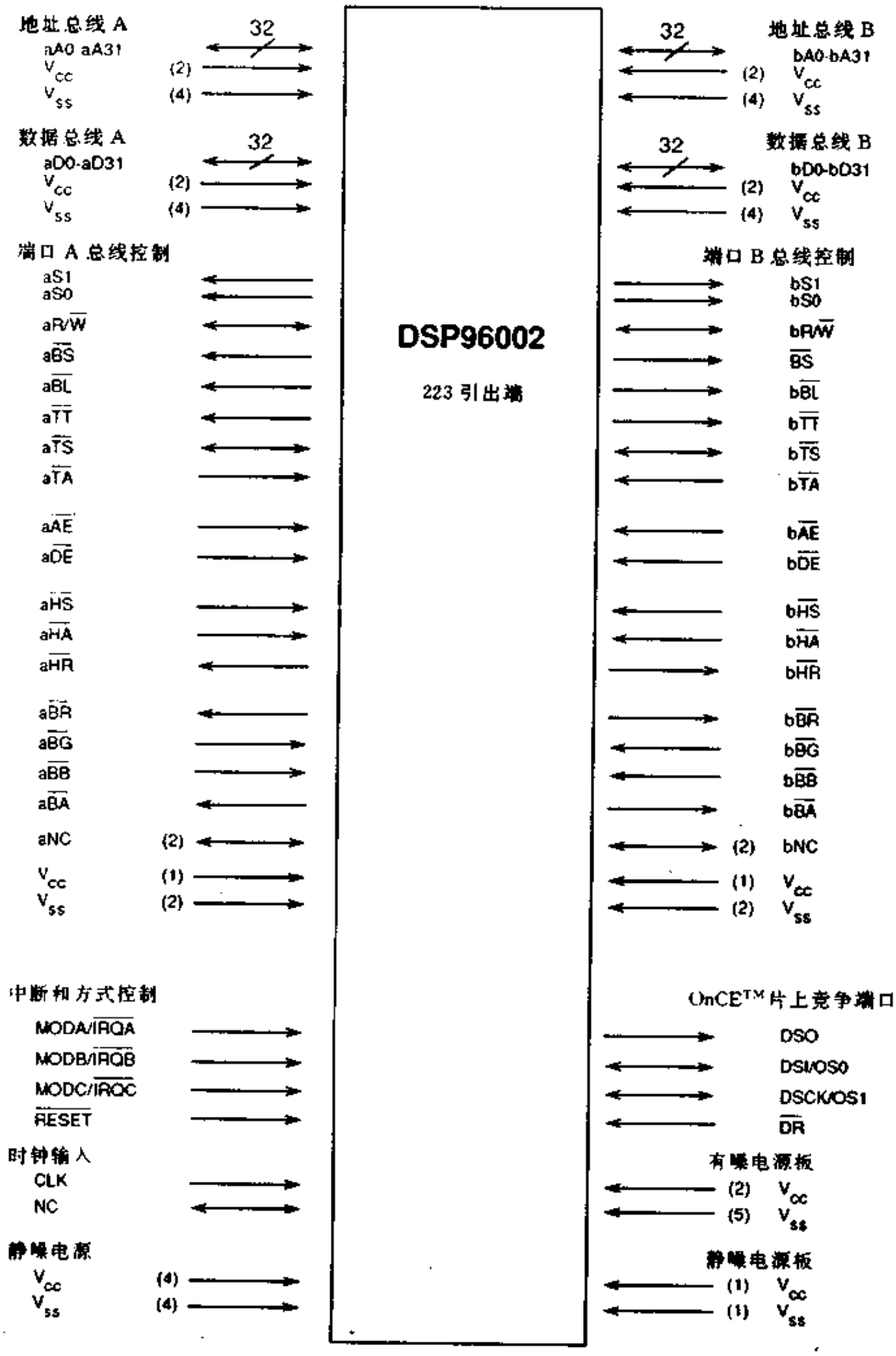


图 13-2 DSP96002 功能信号组

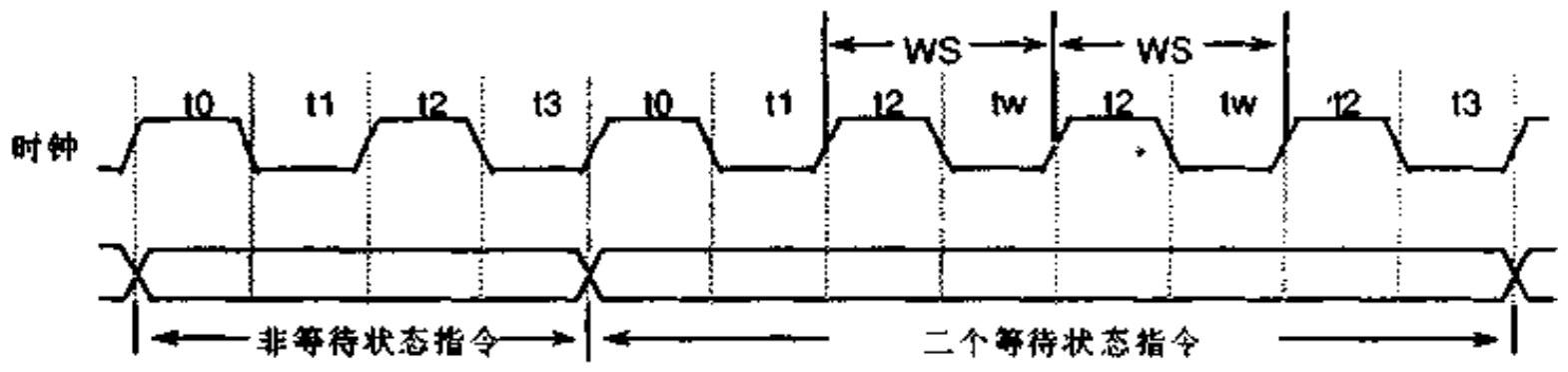


图 13－3　时钟执行周期

数据总线 I/O 驱动器的隔离地。

8. Bus Control Vcc（2）（电源）

总线控制 I/O 驱动器的隔离电源。

9. Bus Control Vss（4）（接地）

总线控制 I/O 驱动器的隔离地。

以上电源和地线都应和外部其他片子的电源和地端相连，并需提供适应的去耦电容。

13.1.4　片上模拟器接口（OnCE）（4 条引出端）

1. $\overline{DR}$（调试请求）

调试使能输入提供从外部命令控制器进入调试操作方式的手段。当确定时，此端使 DSP96002 结束当前正执行的指令，存指令流水线信息，进入调 试方式，并等待将从调试串行输入线进入的命令。

2. DSCK/OS1（调试串行时钟/片状态 1）

DSCK/OS1 端作为输入时，通过此端串行时钟加到 OnCE。串行时钟提供为移数据进出 OnCE 串行端口所要求的脉冲。当输出时（不在调试方式），此端与 OS0 端连接，提供有关片子状态的信息。

3. DSI/OS0（调试串行输入/片状态 0）

DSI/OS0 端作为输入时，串行数据或命令通过此端提供给 OnCE 控制器。DSI 端上接收的数据仅当 DSP96002 进入操作调试方式时会被识别。当作为输出时（不在调试方式），此端与 OS1 端相连，提供有关片子状态的信息。

4. DSO（调试串行输出）

调试串行输出提供包含在 OnCE 控制器寄存器之一的数据，作为从外部命令控制器收到的最后命令的记录。当跟踪或断点发生时，此线将被确定一个 T 周期，指出片子已进入调试方式并等待命令。

13.1.5　端口 A 和 B（162 条引出端）

端口 A 和 B 在引出端和功能上是一致的。下面的引出端说明适合两个端口，每个端口可能是一总线控制器以及每个端口有一主机接口，可按要求访问。

此引出端是 50pF 负载和两个外部 TTL 负载所专用。降额曲线可提供高到 250pF 容性负载时的规定性能。

1. A0～A31（地址总线）

当总线控制时为三态有效高输出。当不是总线控制时，A2～A5 是有效的高输入，A0～A1 和 A6～A31 是三态。作为输入，A2～A5 可能改变对输入时钟（CLK）的非同步关系。A2～A5 是主机接口地址输入，用来选择主机接口寄存器。当总线控制时，A0～A31 指定外部程序和数据存储器访问的地址。若没有外部总线工作，A0～A31 保持它们以前的值。当总线控制时，地址使能（$\overline{AE}$）输入起着对 A0～A31 输出使能控制作用。当总线控制时，在传输选通$\overline{TS}$确定时，A0～A31 是稳定的；在$\overline{TS}$不确定时，才可能改变。A0～A31 在硬件复位时是三态。

2. D0～D31（数据总线）

当总线控制或不是总线控制时为三态有效高双向输入/输出。数据使能（$\overline{DE}$）输入对 D0～D31 起着输出使能的控制作用。作为总线控制，CPU 指令执行或 DMA 控制器控制数据线。D0～D31 也是主机接口数据线。若没有外部总线工作，则 D0～D31 是三态。当硬件复位时，D0～D31 也是三态。

3. S1，S0（空间选择）

当总线控制时为三态有效低输出，当不是总线控制时为三态。定时与地址线 A0～A31 相同。当硬件复位时 S1 和 S0 是三态。

S1	S0	存贮器空间
1	1	不访问
1	0	P 访问
0	1	X 访问
0	0	Y 访问

这些信号可用不同的方法来观察，这取决于外部存贮器如何映射。它们支持把存贮空间分配给各端口的趋向，并映射多存贮空间到同一实际存贮器单元，图 13－4 给出几个例子。编码 S1：S0＝11 可用来将外部存贮器置于低功耗等待方式。

外部存贮器和映像	S1 功能	S0 功能
P	—	$\overline{PS}$
X	$\overline{DS}$	—
Y	$\overline{DS}$	—
X 和 Y 映像为 1 或 2 空间	$\overline{DS}$	X/$\overline{Y}$
P 和 X 映像为 2 空间	$\overline{DS}$	$\overline{PS}$
P 和 X 映像为 1 空间	$\overline{PS}$/$\overline{DS}$	$\overline{PS}$和$\overline{DS}$
P、X 和 Y 映像为 1 空间	$\overline{PS}$/$\overline{DS}$	—

图 13－4　程序和数据存贮器选择编码

4. R/$\overline{W}$（读/写）

当总线控制时为三态有效低输出，当不是总线控制时有效低输入。总线控制定时为和 DSP96002 地址线一样，对 DRAM 接口给一个“早写”信号。对读访问 R/$\overline{W}$ 是高，而对写访问是低。R/$\overline{W}$ 端也是主机接口读/写输入。作为输入，R/$\overline{W}$ 对输入时钟可异步改变。在一个指令周期期间若外部总线不用，则 R/$\overline{W}$ 变高。当硬件复位时，R/$\overline{W}$ 为三态。

5. $\overline{BS}$（总线选通）

当总线控制时为三态有效低输出，不是总线控制时为三态。在总线周期的开始为确定

(对 DRAM 接口提供“提前总线开始”信号)，而在总线周期的结束为不确定。提前非提供“提前总线结束”信号对外部总线控制有用。在一个指令周期期间若不用外部总线，则直到下一个外部总线周期，$\overline{BS}$保持不确定。当硬件复位时$\overline{BS}$是三态。

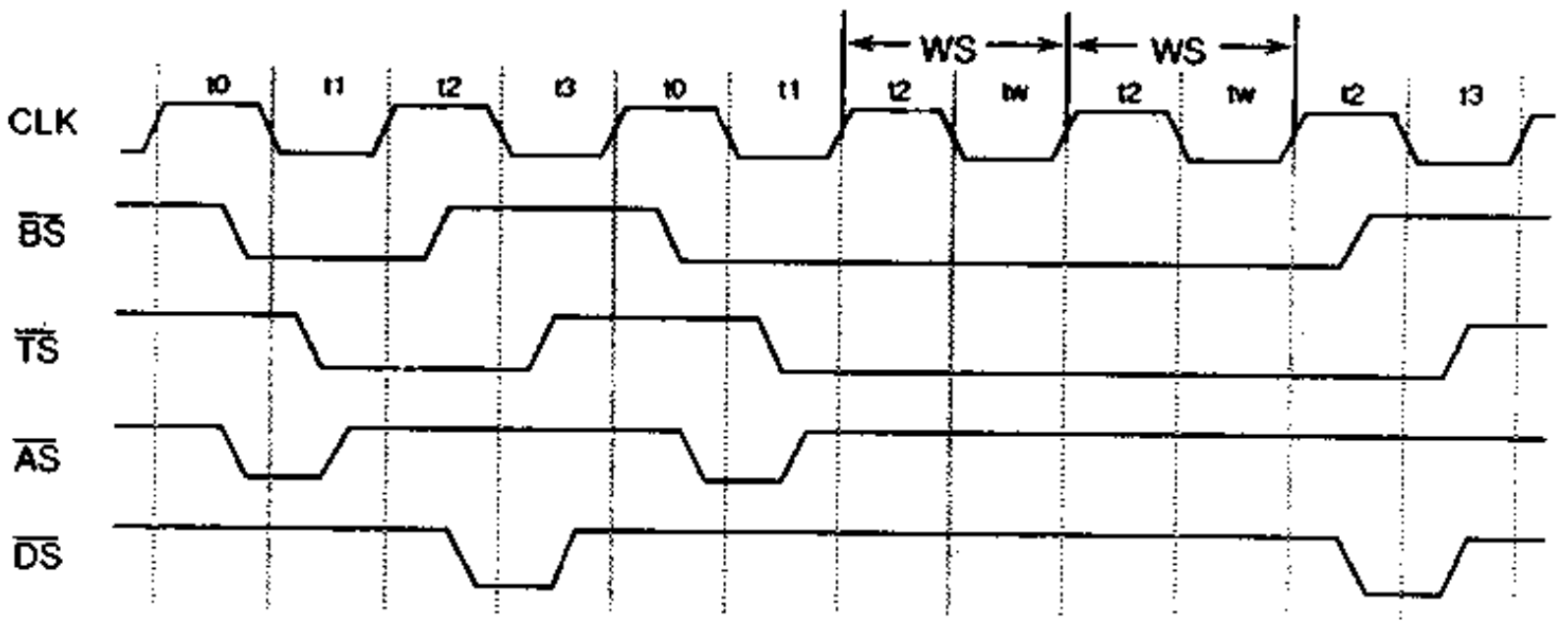
6. $\overline{TT}$ (传送型式)

当总线控制时为三态有效低输出。当不是总线控制时为三态。当总线控制时$\overline{TT}$由片上一页面电路控制。当检测到快访问存贮器方式（页面、静态行、半字节或串行移位寄存器）时，$\overline{TT}$是确定的，在一个指令周期期间若外部总线不用，或当外部访问时若页面电路检测到一个错误，则$\overline{TT}$保持不确定。页面电路错误检测参数是用户可编程的。当硬件复位时$\overline{TT}$是三态。

7. $\overline{TS}$ (传送选通)

当总线控制时为三态有效低输出，当非总线控制时为有效低输入。当总线控制时，$\overline{TS}$确定以指示地址线 A0～A31、S1、S0、$\overline{BS}$、$\overline{BL}$和 R/$\overline{W}$ 是稳定的，并且发生总线读或总线写的传输。当读周期期间，输入数据在$\overline{TS}$上升边被锁在 DSP96002 中。当写周期期间，$\overline{TS}$确定后，输出数据放在数据总线上。所以$\overline{TS}$可用作对外部数据总线缓存（若有的话）的输出使能控制。若当指令周期中外部总线不用，$\overline{TS}$保持不确定直到下一个外部总线周期。对于低速设备或更多地址译码时间，可采用外部触发器来延迟$\overline{TS}$。$\overline{TS}$端也是主机接口传送选通输入，当主机读操作时用于起动数据总线输出驱动器，当主机写操作时用于把数据锁在主机接口中。作为输入，$\overline{TS}$可相对于输入时钟，作非同步改变。写数据在$\overline{TS}$的上升边被锁于主机接口内。当硬件复位时$\overline{TS}$是三态。

当作总线控制时，可一起外部解码$\overline{BS}$和$\overline{TS}$以决定当前总线周期的状态，并产生硬件选通，它对锁住地址和数据信号有用，编码示于图 13－5。



$\overline{BS}$	$\overline{TS}$	总线状态	选通产生应用
1	1	空闲	
0	1	周期开始	地址选通($\overline{AS}$)
0	0	等待	
1	0	周期结束	数据选通($\overline{DS}$)

图 13－5　总线状态编码

8. $\overline{TA}$ (传送确认)

有效低输入。若 DSP96002 是总线控制，并且无外总线工作或 DSP96002 不是总线控制，

则核心不受$\overline{TA}$输入影响。$\overline{TA}$输入是同步“DTACK”功能，它可以无限期地延长外部总线周期。为使$\overline{TA}$能正确操作，其确定和不确定必须对输入时钟（CLK）同步。$\overline{TA}$在输入时钟的下降边取样。任何数目的等待状态（0，1，2…无穷）可由保持$\overline{TA}$不确定来插入。在典型操作中，在总线周期开始时$\overline{TA}$不确定，为使完成总线周期$\overline{TA}$确定，而在下个总线周期前为不确定。$\overline{TA}$确定同步于CLK以后，当前总线周期完成一个时钟周期。等待状态的数目由$\overline{TA}$输入或总线控制寄存器（BCR）二者中的较长者来决定。BCR可用来在外部总线周期中设置等待状态的最小数目。若$\overline{TA}$为低电平且在BCR寄存器中没有指定等待状态，则零等待状态将插入外部总线周期。

9. $\overline{AE}$（地址使能）

有效低输入，要正确地操作其确定和不确定必须对输入时钟同步。若总线控制，$\overline{AE}$确定起动A0～A31地址输出驱动器。若$\overline{AE}$不确定，地址输出驱动器为三态。若非总线控制，不管$\overline{AE}$确定与否地址输出驱动器是三态。$\overline{AE}$的作用是允许复用要实现的总线系统。例子是像用在Macirtosh 11[①]中的NuBus[②]那样的地址/数据复用总线，或者像动态VRAM那样用双端口存贮器的复用地址1/地址2总线。在另一个总线工作以前，恢复用总线允许一条总线到三态之间，必需至少有一个不驱动CLK周期。外部控制负责这个定时。对非复用系统，$\overline{AE}$应当定为低。

10. $\overline{DE}$（数据允许）

有效低输入。若总线控制或主机接口正在读，$\overline{DE}$确定起动D0～D31数据总线输出驱动器。若$\overline{DE}$不确定，数据总线输出驱动器为三态。若没总线控制，则无论$\overline{DE}$确定与否，数据总线输出驱动器均为三态。即使$\overline{DE}$不确定，只读总线周期可以完成。$\overline{DE}$的作用是允许要实现的复用总线系统。例子是像用于Macintosh 11中NuBus那样的复用地址/数据总线，或像以32位存贮器为长字传送所用的复用数据1/数据2总线。对非复用系统，$\overline{DE}$应为低。

11. $\overline{HS}$（主机选择）

有效低输入，可对输入时钟非同步地改变。$\overline{HS}$为低可由地址线A2～A5选择主机接口功能。当$\overline{HS}$为确定时，若$\overline{TS}$确定，将和主机接口发生数据传送。注意，$\overline{HS}$和$\overline{HA}$必须都为高才能阻塞主机接口。当$\overline{HA}$确定时，$\overline{HS}$可忽略。

12. $\overline{HA}$（主机确认）

有效低输入。$\overline{HA}$用来确认对主机接口的中断请求或DMA请求。当主机接口不是在DMA方式时，当$\overline{HA}$和$\overline{HR}$确定时，确定$\overline{TS}$将使主机接口的内容中断矢量寄存器（IVR）到数据总线输出D0～D31上。这提供中断确认能力，可与MC68000族处理器兼容。

若主机接口在DMA方式，$\overline{HA}$用作DMA传送确认输入，且由外部设备确定，以在主机接口寄存器和外部设备之间传送数据。在DMA读方式中，$\overline{HA}$的确定在数据总线输出D0～D31上读主机接口RX寄存器。在DMA写方式中，$\overline{HA}$确定以选通外部数据进入主机接口TX寄存器。写的数据按$\overline{HA}$的上升边锁入TX寄存器。

13. $\overline{HR}$（主机请求）

有效低输出，没有三态。主机请求$\overline{HR}$的确定以指示主机接口正请求服务，从外部设备或

---

① Macintosh11是苹果计算机公司的商标。

② NuBus是Texas仪器公司的商标。

为中断请求或为 DMA 请求。$\overline{HR}$输出可连接到另一 DSP96002 的中断请求输入$\overline{IRQA}$、$\overline{IRQB}$或$\overline{IRQC}$。DSP96002 片上 DMA 控制器通道可选择中断请求输入作为 DMA 传送请求输入。

14. $\overline{BR}$（总线请求）

有效低输出，无三态。当 CPU 或 DMA 正请求总线控制权时，$\overline{BR}$被确定。当 CPU 或 DMA 不再需要总线时$\overline{BR}$不确定。$\overline{BR}$可以是确定或不确定取决于 DSP96002 是总线控制还是总线受控。即使 DSP96002 是总线控制，总线“Parking”允许$\overline{BR}$不确定。典型地$\overline{BR}$被送到外部总线仲裁器，它控制同一外部总线上每个 DSP96002 的优先权、驻留和保存，$\overline{BR}$仅受 CPU 或 DMA 请求外部总线的影响，不受请求内部总线的影响。当硬件复位时，$\overline{BR}$不确定，而仲裁器复位到总线受控状态。

15. $\overline{BG}$（总线开放）

有效低输入。当 DSP96002 可变为下次总线控制时，外部总线仲裁电路确定$\overline{BG}$。当$\overline{BG}$确定时，取得总线控制权以前，DSP96002 必须等待直到$\overline{BB}$不被确定。当$\overline{BG}$不确定时，通常总线控制权在当前总线周期的结束时放弃。这可能发生在指令周期中间，执行这指令所需时间，要大于外部总线周期。注意不可分的读-修改-写指令（BSET，BCLR，BCHG）将不会放弃总线控制权直到当前指令结束。当硬件复位时，$\overline{BG}$不再起作用。

16. $\overline{BA}$（总线确认）

有效低输出。当确定$\overline{BA}$时，DSP96002 在一半 CLK 周期驱动$\overline{BA}$为高，并阻塞有效的提升。在这方面，仅要求小的外部提升电阻以保持线为高。为了得到和 MC68040 $\overline{BB}$端同样的功能，$\overline{BA}$可与$\overline{BB}$直接连接。当$\overline{BG}$确定时，DSP96002 变成不定的总线控制。它直等到$\overline{BB}$为负，指示前面的总线控制脱开总线。未定的总线控制确定$\overline{BA}$以变为当前总线控制。当 CPU 或 DMA 取总线且为总线控制时，$\overline{BA}$被确定。当$\overline{BA}$确定时，DSP96002 是总线的拥有者。当$\overline{BA}$为负时，DSP96002 是总线受控。$\overline{BA}$可以用作三态以控制外部地址、数据和总线控制信号缓存器。当硬件复位时，$\overline{BA}$是三态。

注意在停止总线工作后，不论$\overline{BR}$是确定或不确定，当前总线控制可能保持$\overline{BA}$确定。这叫做“总线停放”并允许当前总线控制去重复使用总线而不用重新仲裁直到某些其他设备要用总线为止。

当不可分的读-修正-写的总线周期时，当前总线控制保持$\overline{BA}$确定，不管$\overline{BG}$是否被外部总线仲裁单元定为不确定。“总线锁定”的形式允许当前总线控制在多任务和多处理器系统中在分享变量上完成极强的操作。完成不可分的读-修正-写总线周期的当前指令是 BCLR，BCHG 和 BSET。

17. $\overline{BB}$（总线忙）

有效低输入。当在外部总线上不是总线控制时，$\overline{BB}$不被确定。在多 DSP96002 系统中，全部$\overline{BB}$输入连在一起，并由所有$\overline{BA}$输出的逻辑与驱动。$\overline{BB}$由未定的总线控制确定以指示它是当前总线控制。$\overline{BB}$由当前总线控制使之不确定以指示它脱离总线，并不再是总线控制。未定的总线控制监视$\overline{BB}$信号直到它不确定。然后未定总线控制确定$\overline{BA}$以变成当前总线控制，直接或间接确定$\overline{BB}$。

18. $\overline{BL}$（总线锁定）

有效低输出，无三态。在外部不可分的读-修正-写（RMW）总线周期开始时确定，而在写总线周期结束时不确定。在 RMW 总线序列的读和写总线周期之间$\overline{BL}$保持确定。$\overline{BL}$可用来

指示特别的存贮器定时（如对 DRAM 的 RMW 定时）可以使用，或用来为安全的信号量更新“锁定”外部多端口存贮器。提前变负提供“提前总线结束”信号对外部总线控制有用。当指令周期内若没用外部总线，则$\overline{BL}$保持不确定直到下一个 RMW 总线周期。若外部总线周期不是不可分的 RMW 总线周期，或若有内部 RMW 总线周期，则$\overline{BL}$也保持不确定。$\overline{BL}$也可用在 BCR 寄存器中设置 LH 位来确定。当硬件复位时$\overline{BL}$不被确定。

13.1.6 保留引出端

有 5 条空闲引出端保留着为以后使用。

13.2 总 线 操 作

外部总线定时由地址总线、数据总线和总线控制的操作所决定。DSP96002 外部端口用于与各种存贮器、外围设备、高速静态 RAM、动态 RAM 和视频 RAM 以及慢存贮器接口。外部总线定时由$\overline{TA}$控制信号和总线控制寄存器（BCR）所控制。BCR 和$\overline{TA}$控制总线接口信号的定时。等待状态的插入由 BCR 控制以提供固定的总线访问定时，还由$\overline{TA}$控制以提供动态总线访问定时。等待状态的数目由$\overline{TA}$输入或 BCR 决定，看那一个更长。

13.2.1 同步总线操作

同步的外总线周期至少由 4 个内时钟相位组成。每个同步的外存贮器访问要求以下过程。

（1）外存贮器地址由地址总线 A0～A31 和存贮器参考选择信号 S1 和 S0 决定。这些信号在外总线周期的第一个位中改变。存贮器参考选择信号与地址总线有相同的定时，并可用作附加地址线。地址和存贮器参考信号也可为相应的存贮器片产生片选信号。这些片选信号改变存贮器片从低功耗等待到有效的方式并开始读访问时间。这允许用慢存贮器，因为片选信号是基于地址而不是基于读或写的允许。

（2）当地址和存贮器参考信号稳定时，数据传送由传送选通$\overline{TS}$信号起动。$\overline{TS}$的确定可“考核”地址和存贮器参考信号已稳定，并完成读和写的数据传送。在总线周期第二相中$\overline{TS}$被确定。

（3）等待状态插入总线周期由等待状态计数器或$\overline{TA}$控制，看那个更长。等待状态计数器由总线寄存器加载。若由此二因素决定的等待状态数目为 0，则没有等待状态插入总线周期，且在第四相$\overline{TA}$不被确定。若等待状态数是 W，则 W 个等待状态插入指令周期。每个等待状态引入一个 Tc 延迟。

（4）当传送选通$\overline{TS}$在总线周期结束时不确定，数据锁在目的设备中。在读周期结束处，DSP96002 内部锁着数据。在写周期结束处，外存贮器锁着数据。地址信号保持稳定直到下个外总线周期第一相以使损耗最小。当无总线作用周期和数据信号三态时，存贮器参考信 S1 和 S0 不被确定。

13.2.2 静态 RAM 支持

静态 RAM 可很容易地与 DSP96002 总线定时接口。有两个基本技术——$\overline{CS}$控制写和$\overline{WE}$控制写。

1. $\overline{CS}$控制写

此静态接口的形式用存贮器片选（$\overline{CS}$）作为写选通。DSP96002R/$\overline{W}$信号用作提前读/写方向指示。适当的数据缓存使能控制RAM不用独立的输出使能（$\overline{OE}$）输入，必须用此形式以避免多数据缓存在数据总线上碰撞。接口的框图示于图13－6。这种方法的缺点是，访问时间是从$\overline{TS}$测量而不是从地址或$\overline{BS}$。因此要求较快的存贮器。

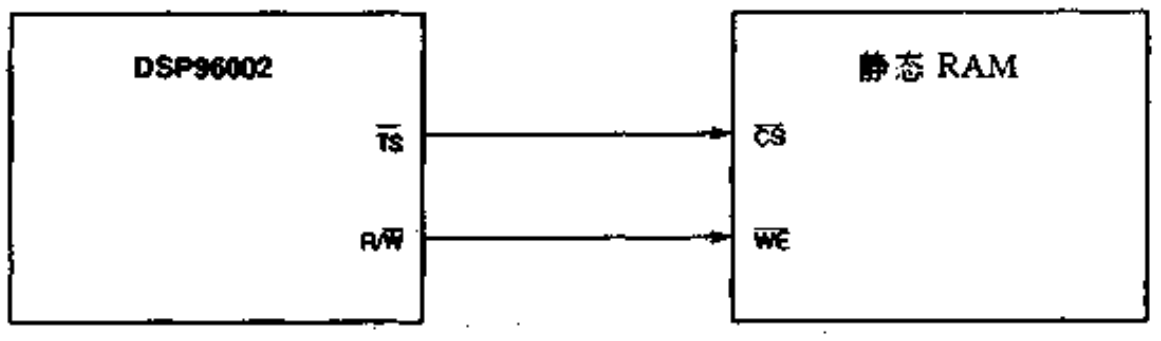


图13－6　$\overline{CS}$控制写接口到静态RAM

2. $\overline{WE}$控制写

此静态接口的形式用存贮器写允许（$\overline{WE}$）作为写选通。DSP96002R/$\overline{W}$信号用来形成迟后的读/写指示。这种形式是56000/1总线接口所用的一种。适当的数据缓存的使能控制要求独立的输出允许（$\overline{OE}$）输入在存贮器上，以避免多数据缓存在数据总线上的碰撞。接口框图如图13－7。

这种方法的优点是访问时间从S1、S0或地址测量而不是从$\overline{TS}$。因此可用较慢的存贮器。缺点是写数据将缩短，因为$\overline{WE}$信号被或门所延迟。

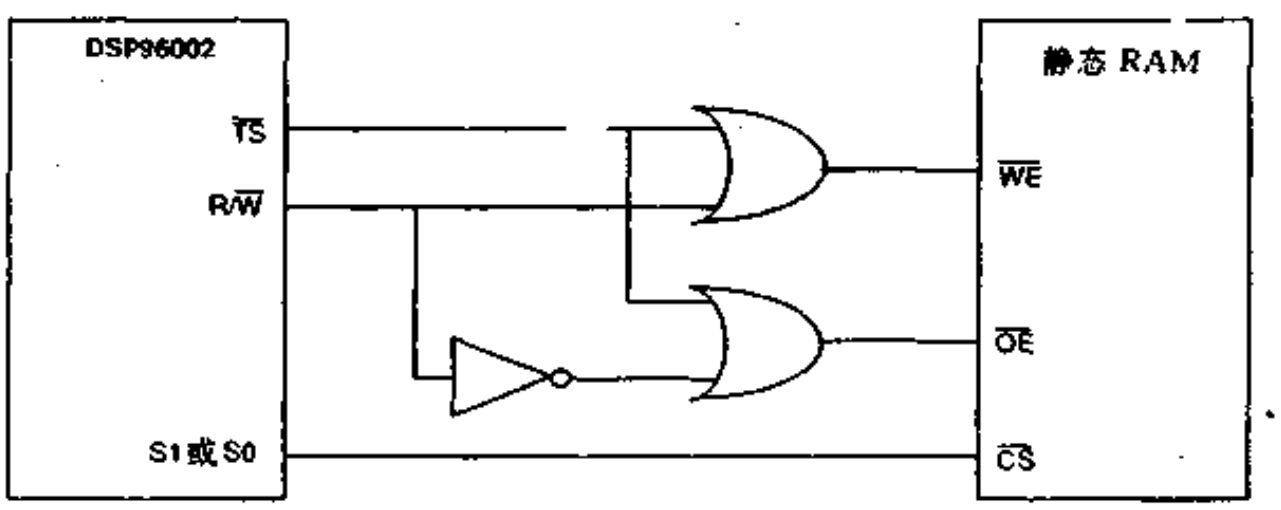


图13－7　$\overline{WE}$控制写接口到静态RAM

13.2.3　动态RAM和视频RAM支持

现代动态存贮器（DRAM）和视频存贮器（VRAM）正变为多种基于以下各点的计算机系统的优先选择：

（1）由于动态存贮单元密度每位的价格。

（2）由于复用地址和控制端的分组密度。

（3）由于快速访问模式〔页、静态行、半字节和串行移位（VRAM）〕相对静态RAM的改进性能。

（4）由于大量生产的商业价格。

设计端口 A/B 总线控制信号是为了在随机读/写周期和快速访问方式中与 DRAM/VRAM 作有效接口。指定总线控制信号定时相对于外时钟（CLK）以外部状态机构实现同步控制。片上页面电路控制$\overline{TT}$端，当慢或快速访问时指示给外部状态机构。

13.3 总线握手和仲裁

总线的交易由单总线控制来管理。总线仲裁决定哪个设备成为总线控制。仲裁逻辑实现与系统有关，结果必须最多使一个设备变成总线控制（即使多个设备要求总线控制权）。

13.3.1 总线仲裁信号

为总线仲裁提供 4 个信号。其中 3 个作为局部仲裁信号，而另一个作为系统仲裁信号。局部仲裁信号运行在潜在总线控制和仲裁逻辑之间。局部信号是$\overline{BR}$、$\overline{BG}$和$\overline{BA}$，$\overline{BB}$是系统仲裁信号。

1. $\overline{BR}$（总线请求）

由请求设备确定以表示要用总线，并保持到不再需要总线。

2. $\overline{BG}$（总线开放）

由总线仲裁控制器确定以通知请求设备总线控制被选。$\overline{BG}$仅当总线不忙时有效。

3. $\overline{BA}$（总线确认）

由从总线仲裁控制器接受总线所有权的设备确定。$\overline{BA}$指示设备是总线控制或总线受控。当确定时，$\overline{BA}$表示此设备是总线控制。$\overline{BA}$可用作对外地址、数据和总线控制信号缓存的三态使能控制。

4. $\overline{BB}$（总线忙）

由所有潜在总线控制监视系统仲裁信号$\overline{BB}$，并由局部总线信号$\overline{BA}$导出。通常$\overline{BB}$是所有总线确认的线或。若总线确认信号由总线控制确定则$\overline{BB}$确定。

13.3.2 仲裁协议

中心总线仲裁器仲裁总线，采用单个的请求/开放线对每个总线控制。

仲裁序列的产生如下：

（1）全部总线所有权的候选者按它们需要总线的情况尽快确定它们各自的$\overline{BR}$信号。

（2）仲裁逻辑按确定的$\overline{BG}$信号指定总线控制选择。

（3）控制选择测试$\overline{BB}$以确保以前的控制已经交出总线。若$\overline{BB}$不确定，则控制选择确定$\overline{BA}$，它指定此设备作为新总线控制。若在$\overline{BB}$信号不确定前发生较高优先权的总线请求，则仲裁逻辑可能以高优先权候选者代替当前控制选择。但一次只能确定一个$\overline{BG}$信号。

（4）新总线控制在$\overline{BA}$确定以后开始总线传送。

（5）仲裁逻辑通知当前总线控制在任何时间以不确定$\overline{BG}$交出总线。完成当前外总线访问后，DSP96002 总线控制释放它的所有权。若指令正在执行读-修正-写外部访问，DSP96002 确定$\overline{BL}$信号，并仅将在完成全部读-修正-写序列之后释放总线。

DSP96002 有 2 个控制位和 1 个状态位（位于总线控制寄存器中），允许$\overline{BR}$和$\overline{BL}$信号的软件控制，当片子是总线控制时加以验证，若 BCR 寄存器中的 RH 位被清除，则 DSP96002 仅

当对总线传送请求是未定时才确定其$\overline{BR}$信号，若 RH 位被设置，则$\overline{BR}$将保持确定。若 BCR 寄存器中的 LH 位被清除，则 DSP96002 仅在读-修正-写总线访问时确定其$\overline{BL}$信号。

13.3.3 仲裁机构

图 13-8 表示实现总线仲裁机构的常用方法的框图。仲裁逻辑决定设备优先权并指定取决于那些优先权的总线所有权。

若没有其他设备请求总线，总线仲裁机构的实现可保持$\overline{BG}$确定当前总线所有权。停止总线工作以后，当前总线控制可以保持$\overline{BA}$确定，不管$\overline{BR}$是确定或不确定。这种情况称为“总线暂停”并允许总线控制重复使用总线，不用重新仲裁，直到其他设备请求总线为止。

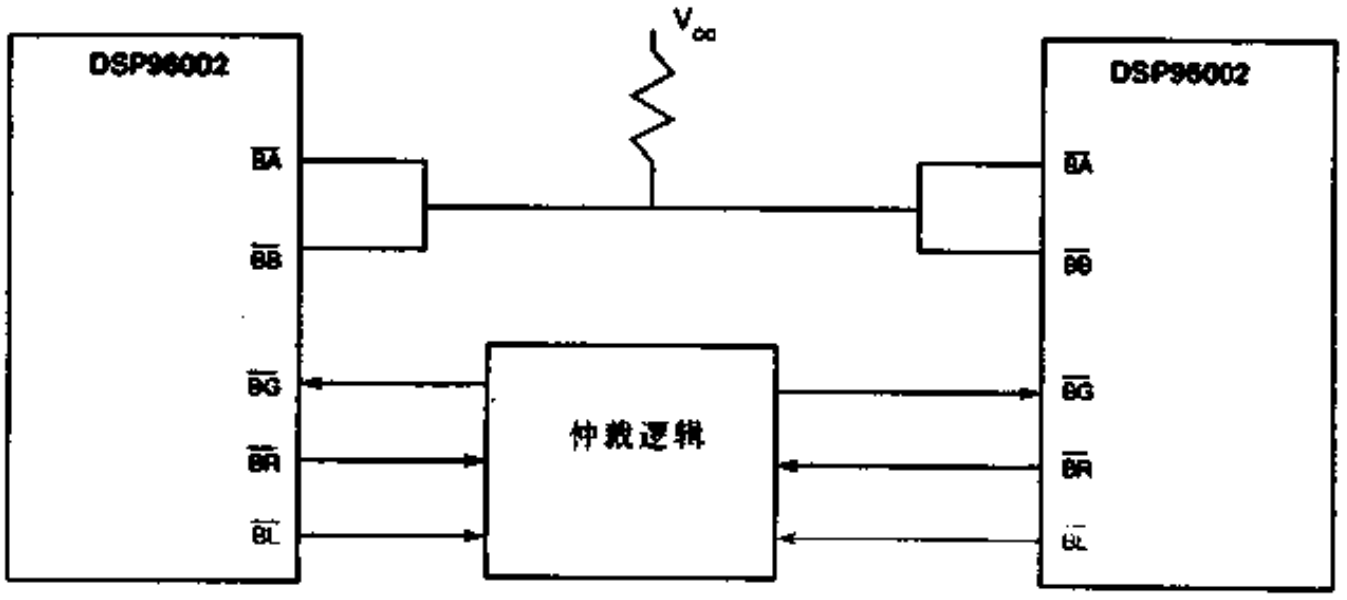


图 13-8 总线仲裁机构

13.3.4 总线握手单元

DSP96002 中的总线握手单元在有限状态机中实现。它由 2 个外输出（$\overline{BR}$、$\overline{BA}$）、2 个外输入（$\overline{BG}$、$\overline{BB}$）和 3 个内输入（ext_acc_req、end_of_sequence、RH）组成（见图 13-9）。当对外总线访问的请求未定时，ext_acc_req 信号被确定，且只要传送在执行就保持确定。在当前序列的最后总线周期时，end_of_sequence 信号被确定。而当执行 RMW 访问的读部分时，end_of_sequence 信号不被确定。当总线传送时若$\overline{BG}$不确定，此信号用来放弃总线所有权。控制总线握手的状态机示于图 13-10。

转移线标有两个字母，它们表示源和目的状态。

转移线方程表示如下：

XX = ^ ext_acc_req & ^ (^ $\overline{BG}$ & $\overline{BB}$)

XY = ext_acc_req & ^ (^ $\overline{BG}$ & $\overline{BB}$)

XZ = ext_acc_req & (^ $\overline{BG}$ & $\overline{BB}$)

XW = ^ ext_acc_req & (^ $\overline{BG}$ & $\overline{BB}$)

YX = ^ ext_acc_req & ^ (^ $\overline{BG}$ & $\overline{BB}$) (注 1)

YY = ext_acc_req & ^ (^ $\overline{BG}$ & $\overline{BB}$)

YZ = ext_acc_req & (^ $\overline{BG}$ & $\overline{BB}$)

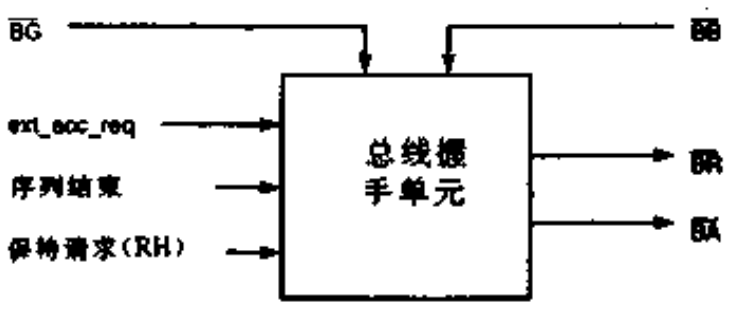


图 13-9 总线握手单元

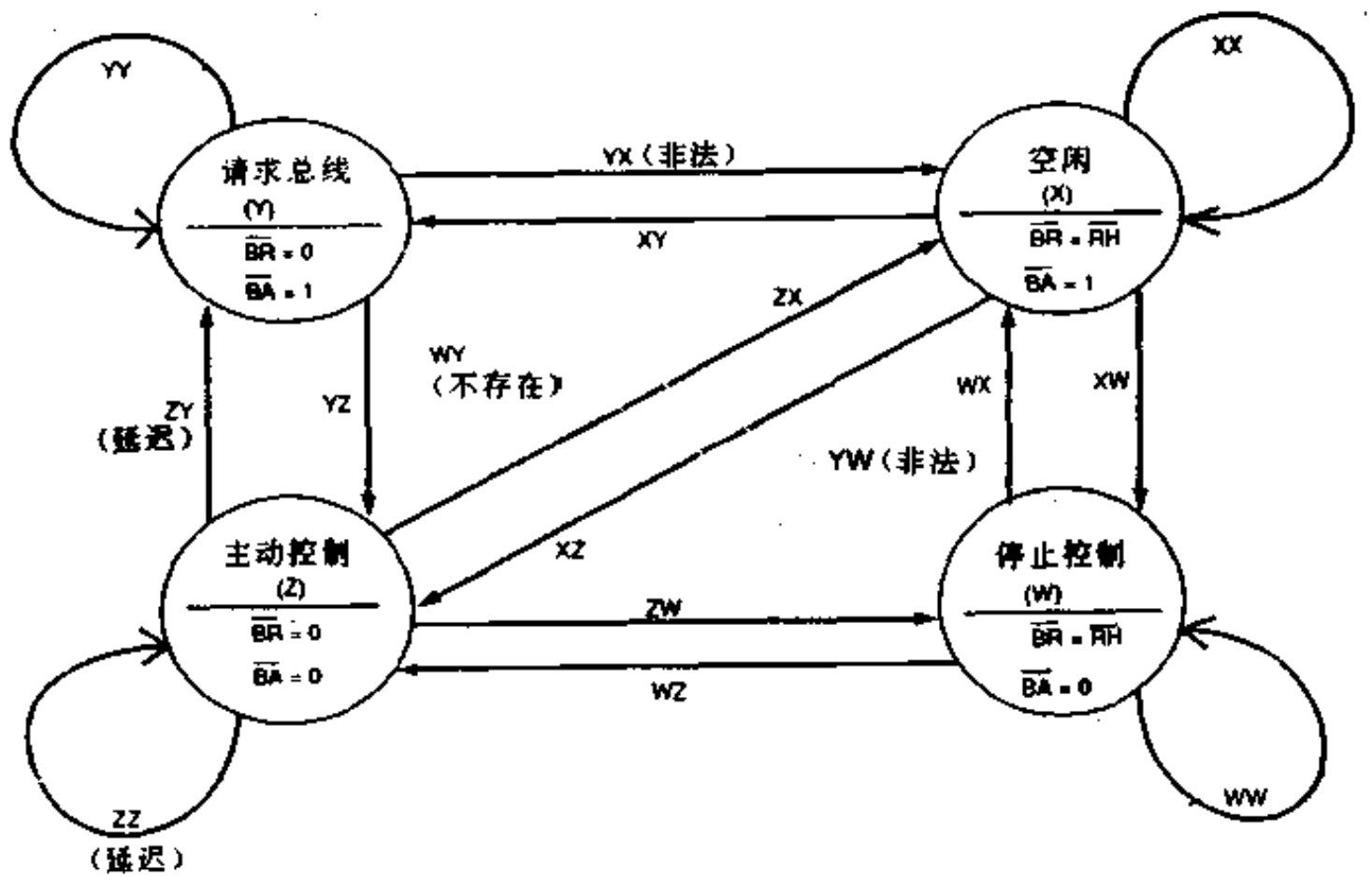


图 13-10　总线握手状态图

YW = ^ ext_acc_req & (^ $\overline{BG}$ & $\overline{BB}$)（注 1）

ZX = ^ ext_acc_req & $\overline{BG}$

ZY = ext_acc_req & $\overline{DBG}$ & end_of_sequence（注 3）

ZZ = ^ end_of_sequence v (ext_acc_req & $\overline{DBG}$)（注 3）

ZW = ^ ext_acc_req & ^ $\overline{BG}$

WX = ^ ext_acc_req & $\overline{BG}$

WY = NON_EXISTENT ARC（注 2）

WZ = ext_acc_req

WW = ^ ext_acc_req & ^ $\overline{BG}$

注：(1) DSP96002 中的不合法线只要总线请求未定，它在访问执行前不会取消。

(2) Non_existent arc 因为若 ext_acc_req 与$\overline{BG}$的非一起到达，设备变成主动控制并开始总线传送。

(3) $\overline{DBG}$是$\overline{BG}$延迟一个相位，这是当它和$\overline{BG}$的非在同样相位处被确定时用来对 ext_acc_req 信号提供响应。

13.3.5　总线仲裁举例

1. 情况 1——正常

若设备请求控制权确定$\overline{BR}$；仲裁器确定请求设备的$\overline{BG}$，且$\overline{BB}$不确定以表示总线不忙。请求设备将确定$\overline{BA}$。

2. 情况 2——总线忙

若设备请求控制权确定$\overline{BR}$；仲裁器由确定请求设备的$\overline{BG}$响应；但是，因$\overline{BB}$确定，总线是忙。请求设备将不确定$\overline{BA}$直到$\overline{BB}$不确定。

3. 情况 3——低优先权

若设备请求控制权确定$\overline{BR}$：仲裁器不允许确定请求设备的$\overline{BG}$，因为较高优先权的设备请求总线。请求设备的$\overline{BA}$将不确定。

4. 情况 4——空缺

若设备不请求总线，且不在总线暂停状态，而在空闲状态：按设计，仲裁器确定$\overline{BG}$。$\overline{BA}$仍保持不确定。

5. 情况 5——当 RMW 时总线闭锁

若设备请求控制权确定$\overline{BR}$，且仲裁器确定请求设备的$\overline{BG}$，以及$\overline{BB}$不确定，则请求设备将确定$\overline{BA}$。若访问外存贮器的 RMW 指令正在执行，且总线仲裁器不确定$\overline{BG}$，则$\overline{BA}$将保持确定直到全部 RMW 指令执行完毕。于是$\overline{BA}$将不确定，因此交出总线。注意当外 RMW 指令执行时，$\overline{BL}$是确定的。通常，$\overline{BL}$信号可用来保证一个多端口存贮器仅被一种控制器在同一时间内写入。参考图 13-11，$\overline{BL}$可从 DSP#1 输入到存贮器控制器，它防止$\overline{TA}$被控制器确定直到 DSP#1 完成它的 RMW 访问为止。

图 13-11　当 RMW 时总线闭锁

6. 情况 6——总线暂停

设备请求控制权确定$\overline{BR}$；仲裁器确定请求设备的$\overline{BG}$，且$\overline{BB}$不确定表示总线不忙——请求设备将确定$\overline{BA}$。当请求设备不再要求总线时，它将不确定$\overline{BR}$；若总线仲裁器因为其他请求已定而保持$\overline{BG}$确定，则$\overline{BA}$将保持确定。这种情况称作总线暂停，并在下次外部访问时消除最后总线控制器对总线再仲裁的需要。

第十四章　片 子 结 构

14.1 引　　言

DSP96002 结构是 32 位高度并行多总线 IEEE 浮点处理器。这种结构适合于各种具有不同存贮器和片上外围要求的 IC 族的成员，而标准的可编程核心保持不变。

14.2 DSP96002 框图

DSP96002 的主要部件是：

- 数据总线。
- 地址总线。
- 数据 ALU。
- 地址产生单元。
- X 数据存贮器。
- Y 数据存贮器。
- 程序控制和系统堆栈。
- 程序存贮器。
- 端口 A 和 B 外总线接口。
- 内总线转换和位操作单元。
- I/O 接口。

总的框图结构示于图 14－1。

14.2.1 数据总线

片上数据传送要经过 5 种 32 位总线：X 数据总线（XDB）、Y 数据总线（YDB）、全局数据总线（GDB）、DMA 数据总线（DDB）和程序数据总线（PDB）。X 和 Y 数据总线也可通过一定指令连结 XDB 和 YDB 而作为 64 位数据总线。在数据 ALU 和 X 数据存贮器和 Y 数据存贮器之间的数据传送通过 XDB 和 YDB。这使片上速度最大和功率最小。直接存贮器访问的数据传送通过 DMA 数据总线。程序存贮器数据传送和指令提取通过 程序数据总线。所有其他数据传送通过全局数据总线。

14.2.2 地址总线

为在两条单向 32 位总线上的内 X 数据存贮器和 Y 数据存贮器指定地址，这两条地址总线是 X 地址总线（XAB）和 Y 地址总线（YAB）。内地址总线的尺寸取决于所用内存贮器的容量，对每个端口 A 和 B 外存贮空间由三个输入 复用器驱动的一条 32 位单 向地址总线寻

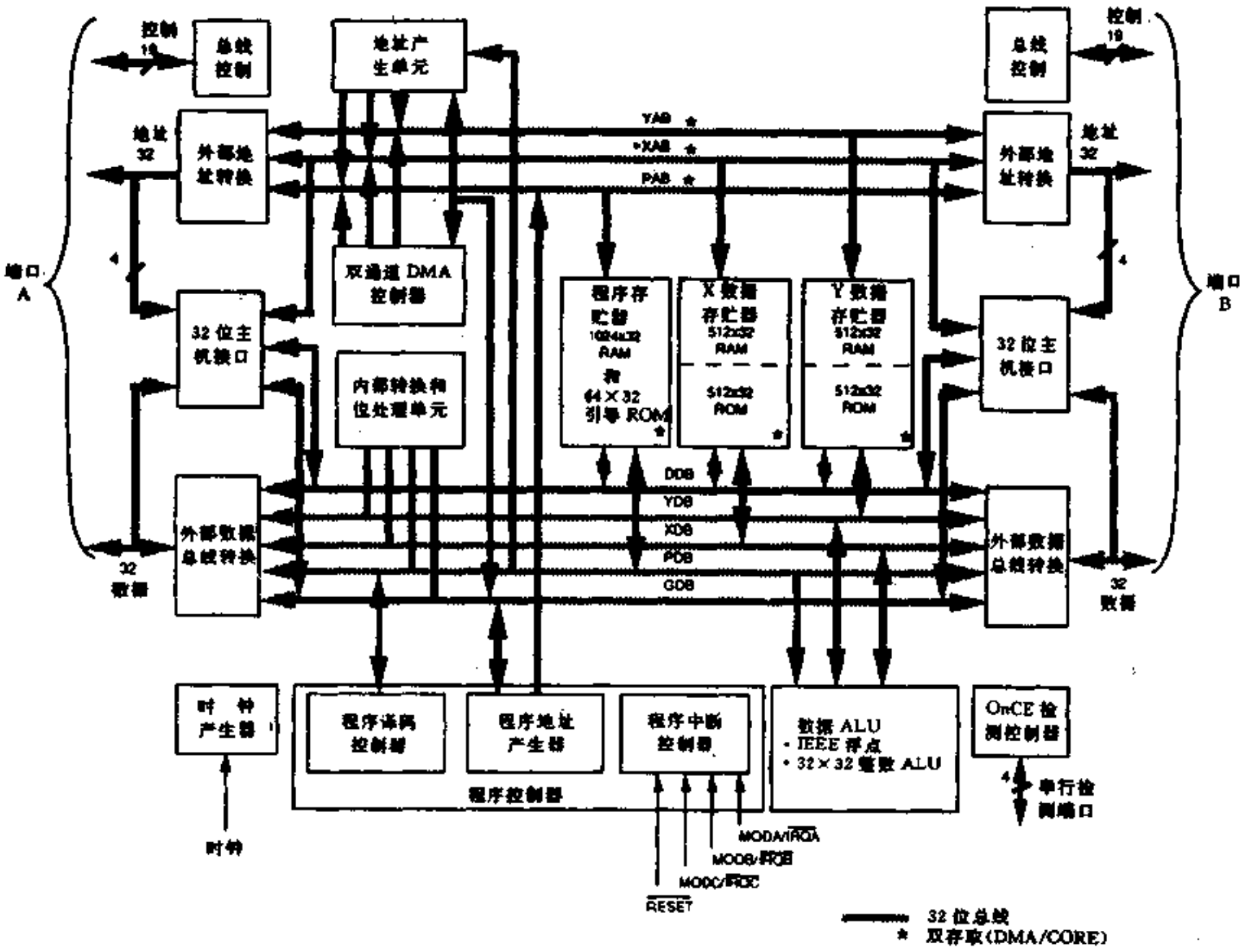


图 14-1 DSP96002 框图

址，复用器可选择 X 地址总线（XAB）、Y 地址总线（YAB）或程序地址总线（PAB）。片上外围和 DMA 控制器是映射在内部 X 存贮空间的存贮器。当使用零等待状态外存贮器时，对每个外存贮器访问需要一个指令周期。

XAB、YAB 和 PAB 是双访问 总线，其意义为，一个指令周期有两个时隙，一个用于片上 DMA 传送，而第二个用于核心传送。

14.2.3 数据 ALU

数据 ALU 完成对数据操作数的全部算术和逻辑操作。数据 ALU 由 10 个 96 位通用寄存器、1 个 32 位桶形移位器、1 个 32 位相加器和 1 个 32 位并行乘法器组成。数据 ALU 寄存器可通过 XDB 和 YDB 作为 32 位或 64 位操作数读或写。数据 ALU 可在一个指令周期内进行乘、加、减、格式变化、移位和逻辑操作 。数据 ALU 源操作 数可能是 32 位或 96 位，并由通用寄存器文件产生。数据 ALU 的结果通常存在通用寄存器之中。浮点数据 ALU 操作常有 96 位结果。整数（定点）数据 ALU 操作 有 32 位或 64 位结果。

数据 ALU 对二进制浮点算术完全实现 IEEE 标准 754。有三种数据格式支持操作：32 位 2 的补数定点、32 位无符号大小定点和 44 位 IEEE 单扩展精度浮点。所有浮点计算用单扩展

精度格式完成，而其结果自动地舍入为单精度或单扩展精度数。所有 4 种 IEEE 舍入模式（舍入到零 、舍入到最近、舍入到正无穷和舍入到 负无穷）都支持所有的浮点操作 和变换。

14.2.4 AGU

AGU 完成对存贮器中数据操作数寻址所需的全部地址存贮和有效地址计算。AGU 和其他片的源并行操作 以减少地址产生的额外开销。AGU 包括 8 个地址寄存器（R0～R7）、8 个偏移寄存器（N0～N7）和 8 个修正寄存器（M0～M7）。地址寄存器是 32 位寄存器，它可包含任何地址或数据。每个地址寄 存器可为输出到 XAB、YAB、和 PAB 而被访问。修正和偏移寄存器是 32 位寄存器，它们通常用于控制更新地址寄存器。

AGU 可通过全局数据总线作为 32 位操作 数读或写 。每个指令周期 AGU 可产生两个 32 位地址，其中之一是为 XAB、YAB 或 PAB 中的任意二个。AGU 可直接访问 XAB 上的 4，294，967，296 个单元，和 YAB 上的 4，294，967，296 个单元，总的有 8，589，934，592 个 32 位数据字的能力。

14.2.5 X 数据存贮器

X 数据存贮器可包含数据 RAM 和 ROM。X 数据 RAM 是 32 位宽的内存，并在 X 存贮器空间占有最低的 512 个单元。X 数据 ROM 也是 32 位内存，在 X 存贮器中占有 1024 个单元。从 XAB 收到地址，并在 XDB 上进行数据传送。X 存贮器是双访问存贮器，在一个周期内它可被访问两次：一次被核心，一次被 DMA。X 存贮器可离片扩展。

14.2.6 Y 数据存贮器

Y 数据存贮器包含数据 RAM 和 ROM。Y 数据 RAM 是 32 位内存，在 Y 存贮器空间占有最低的 512 个单元。Y 数据 ROM 也是 32 位内存，并在 Y 存贮空间占有 1024 个单元。从 YAB 收到地址，并在 YDB 上进行数据传送。Y 存贮器是双重访问存贮器，在一个周期内可两次被访问 ：一次被核心，一次被 DMA。Y 存贮器可离片扩展。

14.2.7 程序控制和系统堆栈

程序控制逻辑完成指令预提取、指令译码和特别处理。32 位程序计数（PC）寄存器在程序存贮空间可访问 4，294，967，296 个单元。

系统堆栈是一分离的内部 RAM，它为子程序调用长中断存贮 PC 与状态寄存器（SR）的内容。堆栈还存贮循环计数器（LC）和循环地址寄存器（LA）。系统堆栈在堆栈存贮器空间它的地址常是固有的，并隐含在当前指令中。堆栈 RAM 是 64 位宽和 15 个单元“深“。当子程序调用或长中断发生时，PC 和 SR 寄存器的内容存在（推 入）系统堆栈“顶“端的内容 复制到 PC 中。当发生中断返回时，“顶“端内容复制到 PC 和 SR 中。

中断将使处理器进入特定状态。进入此状态后，译码中的当前指令将正常地执行，除非它是双字指令的第一字，在此情况下，指令将停止，并在处理完成时重取。其次的二个所取地址由中断控制器提供。当这些地址提取期间，PC 不更新。

若由中断控制所取的字之一跳到子程序，就形成长中断程序，并用堆栈完成转换。若没有中断指令字使控制源改变，则两个所提取中断指令组成一快速中断程序。此时，不用堆栈，

并且中断服务随含二个字在内的指令完成而结束，快速中断程序提供最小额外开销的特定处理。此机理通常用于存贮器和I/O设备之间的传送数据。

系统堆栈也用于实现无额外开销的硬件程序循环。当以DO指令开始程序循环时，发生以下情况：

·当前32位循环计数器（LC）和32位循坏地址寄存器（LA）推入系统堆栈以允许嵌套循环。

·LC和LA寄存器由DO指令中指定的值 初始化。

·程序循环中第一指令的地址和当前状态寄存器内容向系统堆栈上传送。

·状态寄存器中的循环标志位被设置。

当程序循环在进行中，并允许循环检测结束时，设置循环标志。当循环终止并指出终止的循环是否嵌套循环，则从系统堆栈取出循环标志位。

DO指令之后程序循环开始执行，并继续到所取程序地址等于循环地址寄存器内容（程序循环最后地址）。若循环计数器不是1，循环计数器递减且堆栈RAM中的顶点单元读入PC以返回到循环开始。若循环计数器是1，程序循环由递增PC而终止，从堆栈顶点单元读以前的循环标志位到状态寄存器，清除堆栈并使LA和LC寄存器离开堆栈而存入各自的寄存器。当终止一循环时，循环标志、LA和LC寄存器以及系统堆栈指针都复原。

14.2.8 程序存贮器

程序存贮器由1024个单元的32位RAM组成。从程序控制逻辑收到地址。程序存贮器可能包含指令、常数和数据表。程序存贮器是双重访问存贮器，在一个周期内可能被访问二次：一次被 核心，一次被DMA。程序存贮器可以离片扩展。程序RAM可被写出以取出指令。当在引导 程序方式时引导程序ROM也可出现在程序存贮空间。

14.2.9 外总线接口

DSP96002有二个相同的外总线接口。每个接口有一条32位宽地址总线和一条32位宽数据总线，并可用来访问外部数据存贮器、程序存贮器或I/O设备，独立选择线控制访问存贮空间。端口选择控制寄存器允许指定每个存贮空间段到每个外总线接口的端口。

14.2.10 内总线转换位操作单元

内总线转换完成从一条内总线到另一条的数据传送。位操作单元完成XDB、YDB和GDB上对存贮器和寄存器的操作数的操作。

14.2.11 I/O接口

片上I/O接口的目的是减少系统的片数，和在很多应用中“粘住“逻辑。每个I/O接口有它自已的控制、状态和数据寄存器，并作为由DSP96002映射的存贮器I/O对待。每个接口有几个专用的中断矢量地址和控制位以允许/不允许中断。这样可减少与服务设备有关的额外开销，因为每个中断源有它自已的服务程序。

DSP96002提供以下的I/O接口：两个32位并行主机MPU/DMA接口外围在DSP96002上，一个连接外总线的接口A，另一个连接外总线的接口B，一个双通道DMA控制器。

1. 主机接口

DSP96002 为每个外总线接口的端口提供主机 MPU/DMA 接口。每个主机接口（HI）是一个 8 、16、24 或 32 位宽的并行端口，它可直接连接到主机处理器的数据总线。主机处理器可能是任何一个常用微计算机或微处理器、另外的 DSP96002 或 DMA 硬件。HI 犹如在主机处理器地址空间中占 16 个字的映射外围的存贮器。分立的发送和接收数据寄存器是双缓存的，允许 DSP96002 和主机处理器高速有效地传送数据。主机处理器和 HI 的通信是用标准主机处理器数据传送指令和寻址方式来完成的。握手标志是为查询或中断驱动的数据传送而提供的。

2. DMA 控制器

DMA 控制器完成访问 DMA 源和目的操作数所需要的全部地址存贮和有效的地址计算。DMA 控制器与其它片上设备并行工作以减少数据或程序传送的额外开销。DMA 控制器对每个通道包含一个源地址寄存器、一个源偏置寄存器、一个源修正寄存器、一个目的地址寄存器、一个目的偏置寄存器和一个目的修正寄存器。

此外每个通道有二个控制寄存器。每次传送后递减计数器。DMA 控制/状态寄存器控制 DMA 工作并包含 DMA 状态。所有 DMA 寄存器映射到 X 存贮器空间。AGU 是为计算源地址和目的地址，而由 DMA 分享。DMA 寻找方式是：线性、位倒置和模数。

14.3 数据 ALU 框图

数据 ALU 的主要部件是：

- 数据 ALU 寄存器行。
- 乘法单元。
- 加法单元。
- 逻辑单元 。
- 格式转换器。
- 除和平方根单元。
- 控制器和仲裁器。

数据 ALU 结构的框图示于图 14－2。

D0～D9 是 96 位寄存器，它作为数据 ALU 通用的寄存器行。每个寄存器分成三部分：高、中和低，每个 32 位。寄存器对浮点源和/或目的操作数可作为 10 个 96 位寄存器 Dn（Dn.H；Dn.M；Dn.L）n＝0，1，..9。这些寄存器接收来自乘法器、加法器和减法器的输入，并提供同样形式的源数据寄存器。大多数数据 ALU 浮点操作指定 96 位寄存器作为源和/或目的操作数。但是 D8 和 D9 从来不是数据 ALU 操作的目的地址。

数据以双精度浮点格式存在寄存器中。每个寄存器可通过 XDB 或 YDB 作 为浮点操作数读或写。当 Dn 寄存器写入不同浮点格式操作数时，格式转换自动完成，当从 XDB 或 YDB 以单精度浮点 MOVE 的结果写 Dn 时，可能发生格式转换。若单精度操作 数被写入浮点数据寄存器中，则数据寄 存器的中间部分写入字操作数的尾数部分，低的部分是 0，而高的部分写入字操作数的指数部分。

寄存器也可作为 30 个 32 位寄存器 Dn.H，Dn.M，Dn.L，n＝0，1，...9，每个寄存器

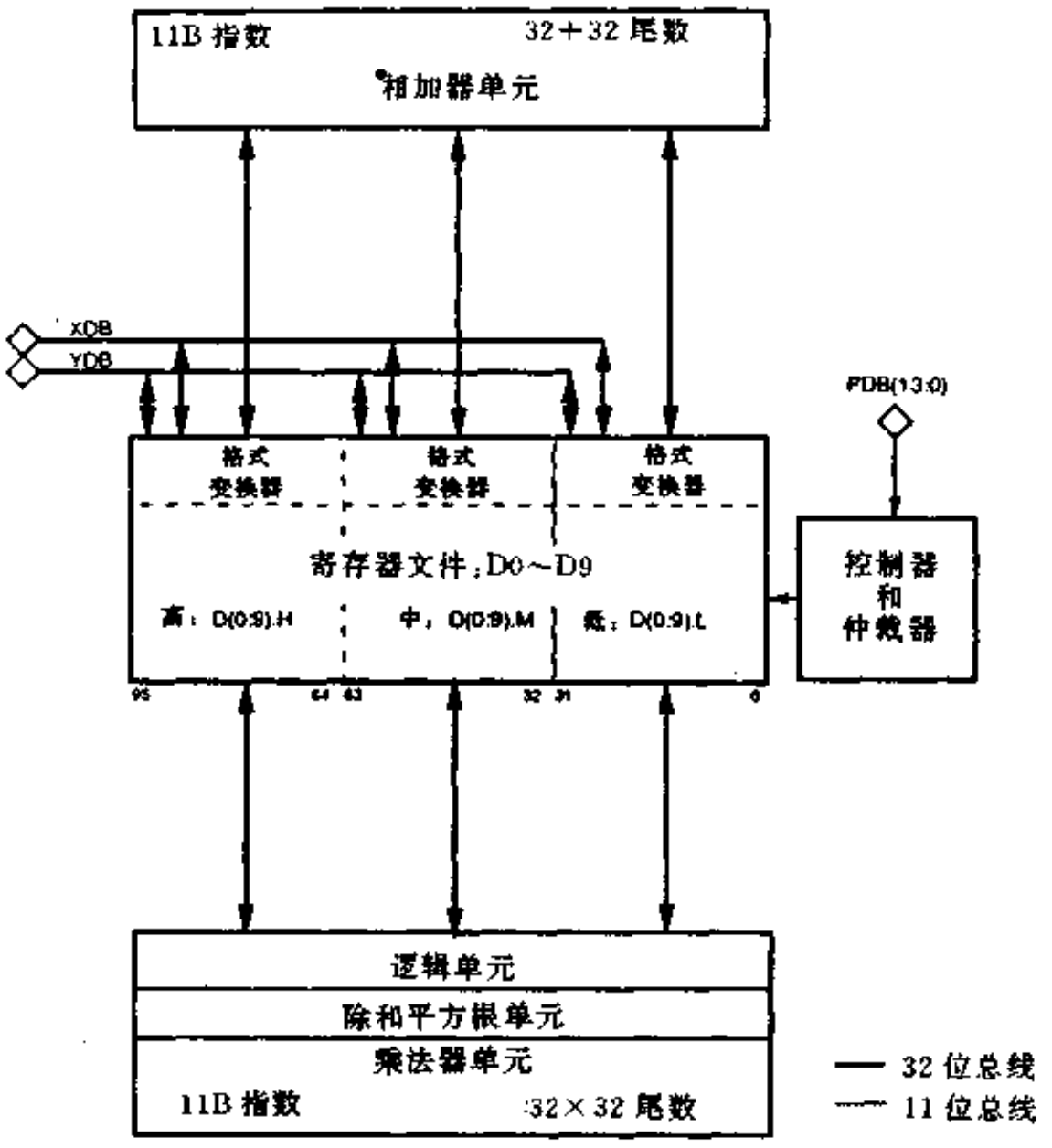


图 14－2　数据 ALU 框图

可通过 XDB 或 YDB 作为字操作 数读或写，当单个的 32 位寄存器通过 XDB 或 YDB 写 入时，不发生格式转换而且仅被指定的寄存器所影响。寄存器的低部分 DN.L 对大多数整数运算用作源和/或目的。此时整数寄存器为乘法器和加法器/减法器提供一操作数，同时从乘法器和加法器/减法器接收一输入。注意在整数相乘时结果将是 64 位，并存在目的寄存器的中间和低的部分。

14.3.1　乘法单元

相乘器是数据 ALU 的二个算术处理单元之一，并完成所有浮点相乘以及有符号/无符号定点（整数）相乘。

对于浮点相乘，相乘器接收二个 44 位输入 操作数和输出一个 44 位结果。浮点相乘的操作独立发生，并与浮点相加器以及 XDB 和 YDB 并行操作。对于定点相乘，相乘器接收二个 32 位输入操作数，并输出一个 64 位结果。定点相乘器的操作独立发生，并与 XDB 和 YDB 并行工作 。数据 ALU 寄存器可由程序员使用以实现数据 ALU 流水线。

相乘器以异步逻辑实现，且所有相乘运算发生在一个指令周期中。在相乘器输入操作 数总线上提供闭锁以避免竞争。相乘单元的主要部件如下：

- 相乘器阵列。
- 相乘器控制记录器。

·指数相加器。

1. 相乘器阵列

相乘器阵列是 32×32 位具有 64 位结果的异步、并行相乘器。相乘器阵列是基于修正的 Booth 算法。阵列完成带符号/无符号定点相乘，用整数数据表示，浮点相乘，用 32 位尾数表示。相乘器阵列完成自动的舍入到 32 位尾数（对浮点相乘）。

2. 相乘器控制记录器

相乘器控制记录器引导相乘器阵列的操作 并完成对修正的 BOOTH 算法相乘的相乘器操作数记录。

3. 指数相加器

指数相加器是 11 位相加器，它用于两 个相乘操作数的指数相加。它实际计算二个输人指数间的和，并减去偏置。最后的指数存在目的寄存器的高位部分。

14.3.2 加法单元

相加器是数据 ALU 的第二个算术处理单元，并完成 所有带符号 /无符号整数定点相加、相减和位移操作 以及浮点加、减和加一减，浮点加一减操作 由在相同输人操作数上同时完成加和减所组成。此种操作对实现 FFT 和其它变换有用。

浮点相加器/相减器的操作独立发生，并与浮点相乘器操作 以及 XDB 和 YDB 工作并行。

定点相加器操作独立发生，并与 XDB 和 YDB 工作 并行。数据 ALU 对数据 ALU 相加器输人和输出提供流水线。

所有相加器内的操作发生在一个指令周期中，在相加器输人操作数总线上提供闭锁以避免竞争。相加器的主要部件是：

·相加单元。

·相减单元。

·桶形位移器和归一化单元。

·指数比较器和更新单元。

·特定功能单元。

1. 相加单元

相加单元是高速 32 位的异步相加器，用在所有浮点非相乘操作中，得出 32 位的结果。相加单元完成自动的舍人到 32 位尾数。若指定舍入到 IEEE 单精度，则结果为 24 位尾数。舍入的形式由 MR 寄存器中舍人方式位指定。

二条内数据总线收到二个输人操作数，在尾数排行的处理之后，32 位尾数的操作数送到相加单元。相加单元的输出送到舍人单元，所得结果存人目的寄存器。

2. 相减单元

相减单元 是高速 32 位异步相加器/相减器，用在所有浮点非乘运算以及定点运算中，给出 32 位结果。相减单元完成自动舍人到 32 位总尾数（对浮点）。若指定 舍人到 IEEE 单精度，则结果是舍人到 24 位尾数。舍人形式由 MR 寄存器中舍人方式位指定 。

相减单元 在浮点运算时，送出结果 在目的寄存器的中部，在整数运算时，结果在目的寄存器的低部位。

3. 桶形移位器和归一化单元

桶形移位器是32位异步并行双向（左一右）多位移位器，用在大多数浮点运算和算术与逻辑移位操作中，送出32位结果。当用在浮点运算中时，它的主要任务是提供操作数行以及最后结果的后归一化。当用在定点移位时，桶形移位器完成以下操作：

- 单位和多位算术左移和右移（ASL*n，ASR*n）
- 单位和多位逻辑左移和右移（LSL*n，LSR*n）

4. 指数比较器和更新单元

EXC是11位相减器，它比较加/减运算的两个操作 数的指数 。它从寄存器高部位收到AEIA和AEIB总线上的输人 ，并送出最大指数和指数之间的差。指数差被送到桶形移位器，它用作浮点运算要求的尾数处理的信息。最大指数送到指数更新单元，它按照后归一化处理结果进行更新，最后结果送到AEOA和/或AEOS总线上，并存在目的寄存器的高部位。

14.3.3 逻辑单元

数据ALU中的逻辑单元完成逻辑运算AND、ANDC、OR、ORC、EOR、NOT、ROR和ROL。它也完成SPLIT、SPLITB、JOIN、JOINB、EXT和EXTB段处理指令。逻辑单元是32位并运算寄存器低部位的数据。

14.3.4 除和平方根单元

除和平方根运算用迭代算法，要求1/X和SQR（1/X）的起始种子（第一次近似）。

14.3.5 控制器和仲裁器

控制器和仲裁器单元（CA）提供数据ALU处理单元和寄存器列所要求的控制信号，并负责全部IEEE标准的实现。后一个任务由SR寄存器中FZ位决定 。在“Flush－to－Zero”方式中，所有去归一化的输入操作 数认为是0，并且所有去归一化结果是“Flush－to－Zero”。去归一化数包括浮点0。

当测定去归一化的数目作为输人操作数时控制器和仲裁器为了进入IEEE方式处理将加一个额外周期，其后将对每个去归一化输入操作加一个额外周期。这些额外周期用来归一化输人操作 数。归一化以后，操作 数以暂时格式存贮，此格式有负的偏置指数（“缠绕格式”），它对用户无效。但源寄存器中的操作 数原始值不受影响。当测定去归一化数目作为输出 结果时，控制器和仲裁器单元将进人IEEE方式处理，并对每个去归一化输出 结果加一个额 外周期。

14.4 AGU（地址产生单元）

AGU的主要部件是：

- 地址寄存器列。
- 偏置寄存器列。
- 修正寄存器列。
- 暂时地址寄存器。
- 模算术单元。

·地址输出复用器。

AGU 的框图示于图 14－3。

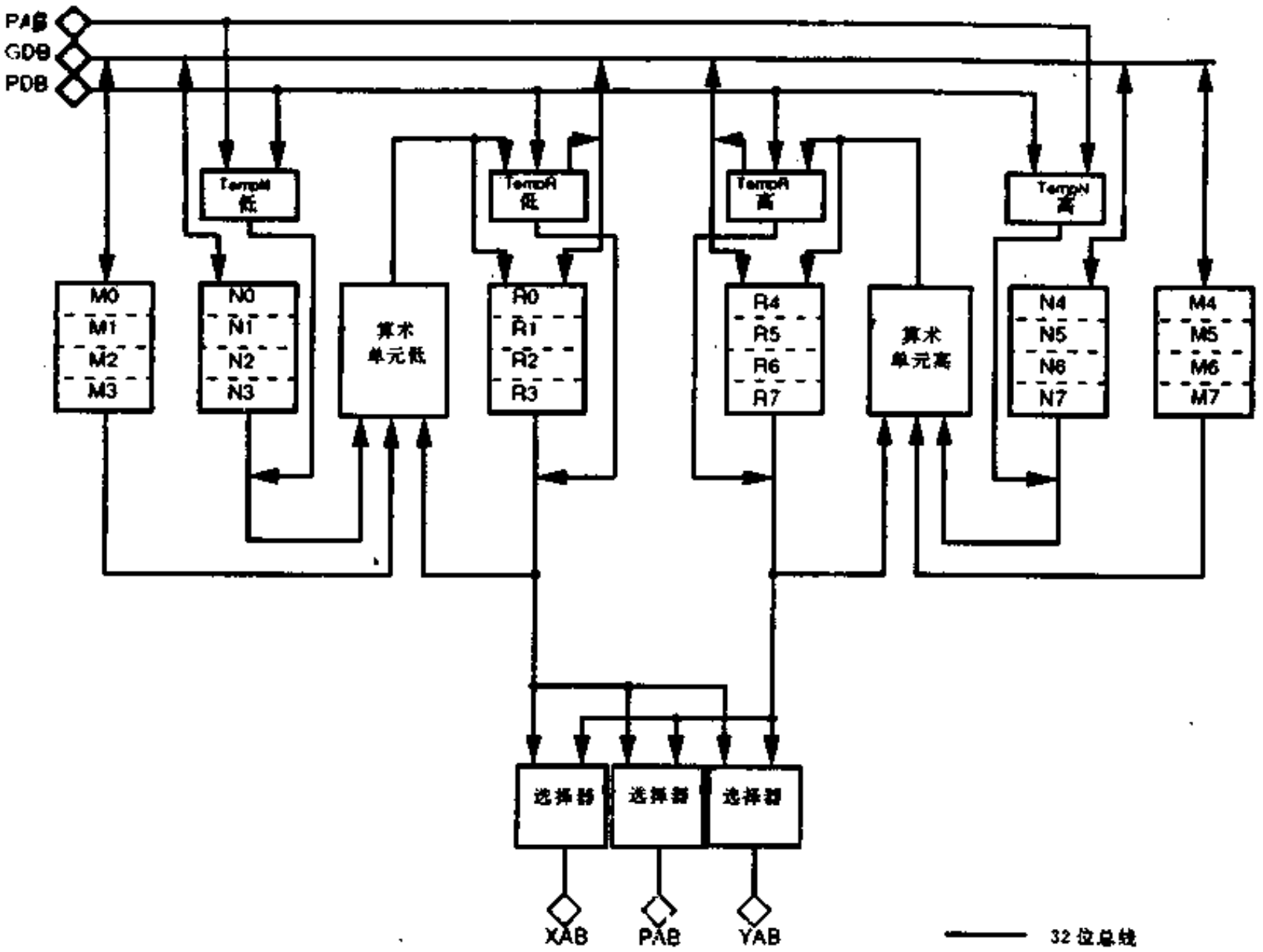


图 14－3　AGU 框图

14.4.1　地址寄存器列

二个地址寄存器列的每一个由 4 个 32 位寄存器组成。二列分别包括地址寄存器 R0～R3 和 R4～R7，它们通常包含用于指示存贮器的地址。每个寄存器可由全局数据总线读或写。要求对 XAB 和 YAB 高速访问以便使对内和外 X 数据存贮器、Y 数据存贮器和程序存贮器有最大访问时间。每个地址寄存器可用作有关的模算术单元的输入以进行寄存器更新计算。每个寄存器可由全局数据总线写入，或由相应的模算术单元输出写入。全局数据总线和模算术单元访问的寄存器不要求相同。每个寄存器可提供独立的写入。

14.4.2　偏置寄存器列

二个偏置寄存器列的每一个由 4 个 32 位寄存器组成。二列分别包括偏置寄存器 N0～N3 和 N4～N7，并通常保持偏置值用以更新地址指针，但可保持数据，每个偏置寄存器可由全局数据总线读和写。当相同数量地址寄存器是读时，每个偏置寄存器是读，并用来输入 到它的有关模算术单元。读地址选择要读的偏置寄存器到模算术单元。全局数据总线访问寄存器。对每个寄存器能提供独立写。

14.4.3 修正寄存器列

两个修正寄存器列的每一个由4个32位寄存器组成。两列分别包括修正寄存器M0～M3和M4～M7。每个修正寄存器可由全局数据总线读或写。当相同目的地址寄存器读时，每个修正寄存器自动读，并作为有关的模算术单元的输人。对每个寄存器提供独立 的写。当处理器复位时每个修正寄存器置到$FFFFFFFF。

14.4.4 暂时地址寄存器

在AGU中有两类暂时寄存器：TempR（高和低）和TempN（高和低）。暂时地址寄存器，TempR低和TempR高，是32位寄存器，它对来自程序数据总线载人的绝对地址提供暂存。当由偏置寻址方式和LEA指令标注的预更新周期期间，模算术单元输出载人TempR寄存器。在上述每一种情况下，由它相应的模算术单元访问和更新地址寄存器。并在一个指令周期内存人TempR。在以下的周期内，TempR的内容用来访问X或Y存贮器。对所有绝对寻址方式，操作数的地址写 人TempR，并用来访问X、Y或P存贮器。

暂时地址寄存器TempN低和TempN高是32位寄存器，它对来自程序地址总线载人的PC提供暂存，并且它用于PC相对寻址方式中。它们在长或短位移寻址方式情况下也可以由程序数据总线载人。

14.4.5 模算术单元

模算术单元框图示于图14－4。二个单元相同。每个包含32位全加器（称偏置相加器）对已选的地址寄存器内容，它可以加1、减1。第二个全加器（称模相加器）把第一个全加器相加结果加到M或－M上，这里M存在相应的修正寄 存器中。第三个全加器（称逆进位相加器）把常数1、－1、偏置N或－N加到反向进位的被选地址寄存器。反向进位就是从最高有效位到最低有效位。偏置相加器和逆进位相加器是并行的并分享共同的输人。它们之间仅有的差别是进位的传送是相反方向。测试逻辑由修正 译码器、二个进位相乘器和一些控制逻辑组成，此逻辑决定全加器的三个相加输出 中哪一个输出 到相关的地址寄存器列或暂存器。

在一个指令周期中，每个模算术单元可从相应的地址寄存器更新一个地址寄 存器。它可以完成线性、逆进位和模等运算。所选修正寄存器内容指定在模算术单元中译码并影响单元操作。模算术单元并行完成三种操作：

（1）偏置相加器的输出给出线性算术结果（例如Rn＋1；Rn＋Nn），并选作为线性算术寻址修正器和PC相对寻址方式的模算术 单元输出。

（2）逆进位相加器完成逆进位要求的操作，并且它的输出选作逆进位寻址修正 器的模算术单元输出。逆进位对2K点基2FFF寻址是有用的。模算术单元将完成函数（Rn＋/－N）的模M，其中N可以是1、－1或偏置寄存器Nn的内容。

（3）若模运算要求缠绕，则模相加器的相加输出将给出正确的更新地址寄存器值，否则，若不需要缠绕，则偏置相加器的输出给出正确结果。

测试逻辑决定哪个输出 地址要选 择。模算术单元由DMA和AGU分享，且它们是时间复用的。

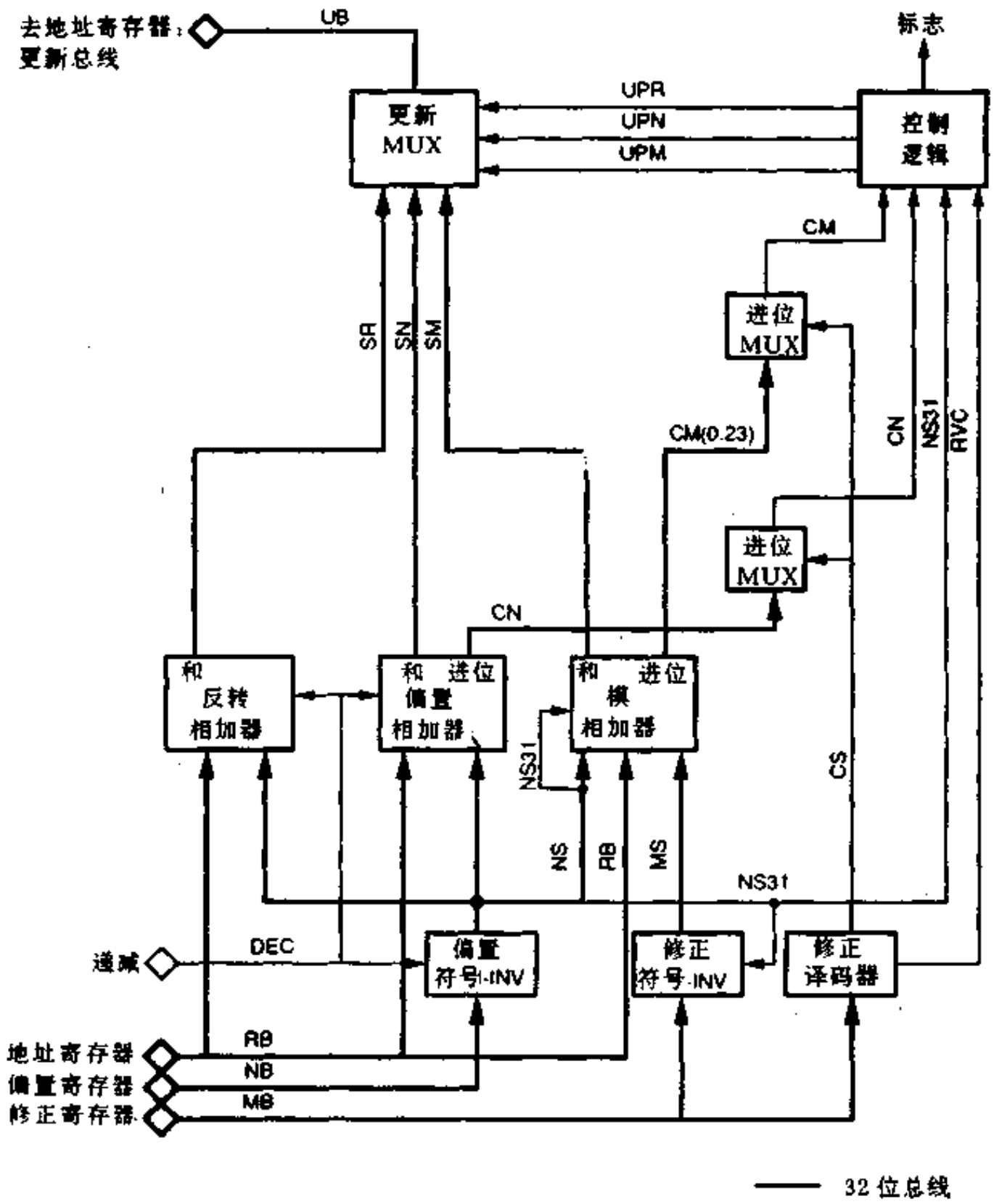


图 14－4　模算术单元框图

14.4.6　地址输出复用器

地址输出复用器由 XAB、YAB 和 PAB 选择源。它们允许 XAB、YAB 或 PAB 的地址输出从 R0～R3，R4～R7 或从 TempR 低或 TempR 高开始。地址输出复用器由 DMA 和 AGU 分享。输出复用器是时间复用的——第一个半指令周期指定为 DMA 传送，而第二个半周期为核心传送。

第十五章　软 件 结 构

15.1　编 程 模 型

编程器可把 DSP96002 结构看成三个并行工作的执行单元。这三个执行单元是：

- 数据 ALU。
- 地址产生单元。
- 程序控制器。

DSP96002 指令集设计得能够对这些并行处理设备进行灵活控制。很多指令使编程器保持每个单元很忙，因此增强了程序执行速度。编程模型示于图 15－1 和 15－2。

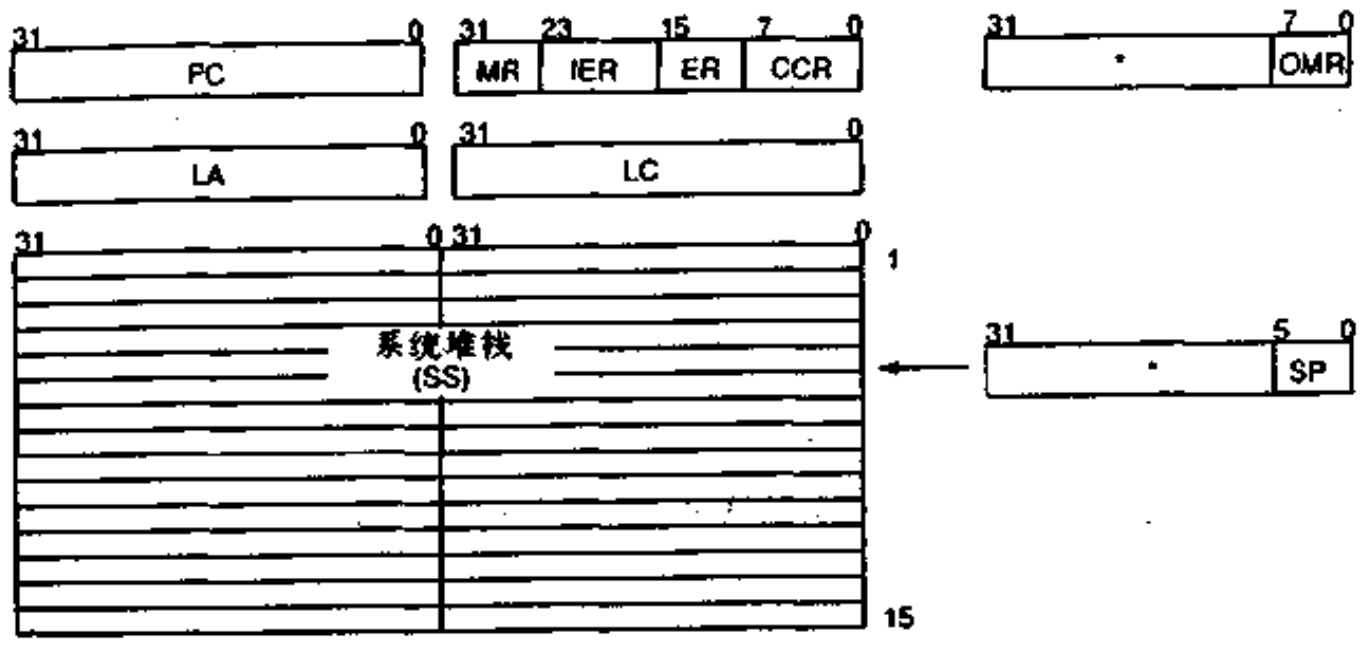


图 15－1　DSP96002 编程模型——程序控制器

15.2　数据 ALU 寄存器组

10 个寄存器 D0～D9 都是 96 位，可作为 30 个独立的 32 位寄存器或 10 个 96 位浮点寄存器。每 96 位寄存器分为三个子寄存器：高、中和低。每个子寄存器可被指定寄存器号和子寄存器名称（如 D0.H，D0.M，D0.L）访问。低的子寄存器用作整数运算的源和目的。当对子寄存器写入或读出时不用进行格式变换。

96 位寄存器 Dn（n＝0，…，9）由 Dn.H：Dn.M：Dn.L 连结而形成浮点数据寄存器。浮点数据寄存器中的数据表示通常在 IEEE 双精度式的表示中。当以单或双精度浮点数写寄存器时，发生内部表示的格式变换。格式变换自动完成，并对用户是透明的。

寄存器用作 XDB 和 YDB 和相乘器和/或相加器之间的输入流水线寄存器。它们也可读回到适当的数据总线以完成存贮器延迟操作。并为中断服务程序存入操作。

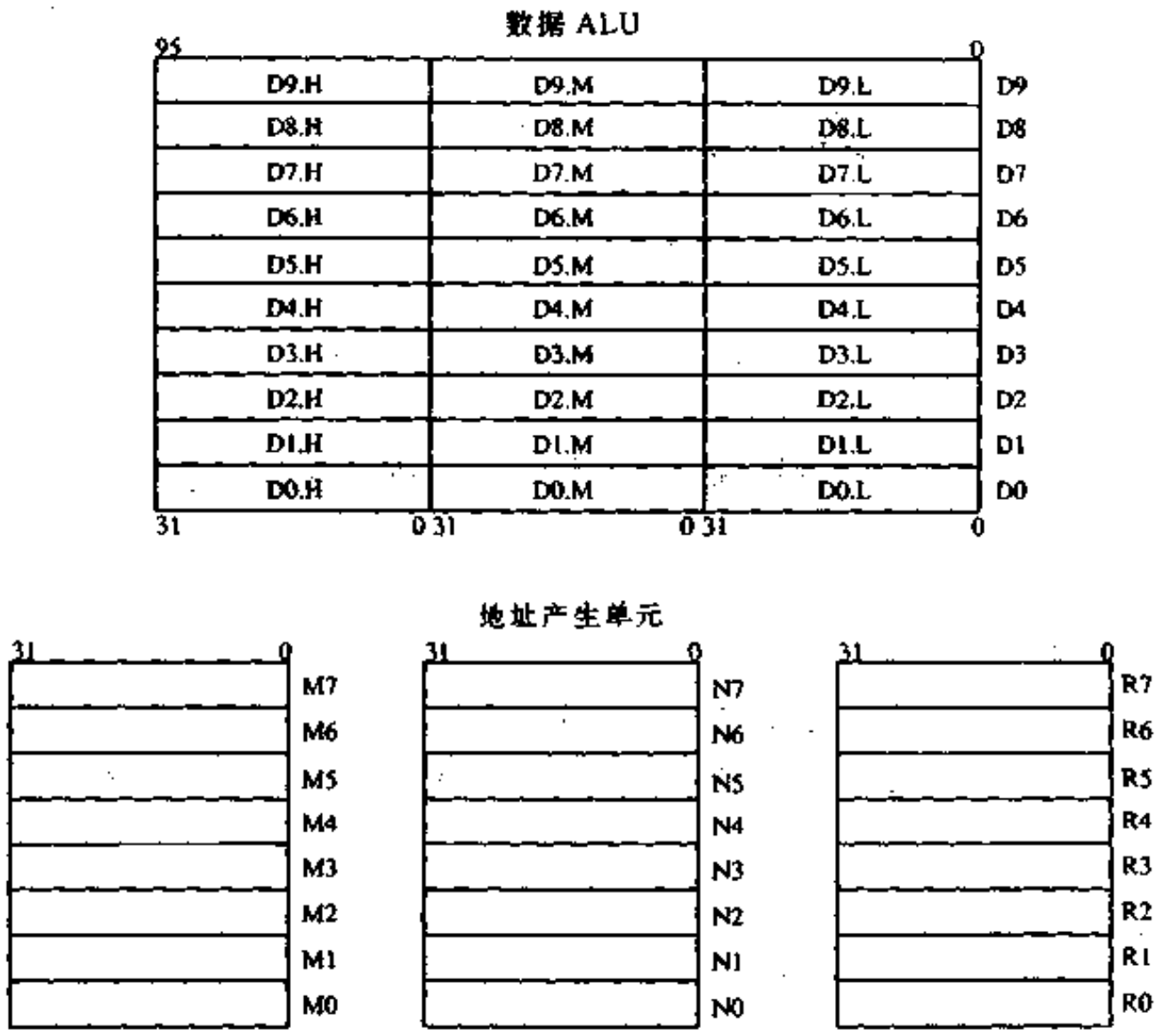


图 15-2　DSP96002 编程模型——数据 ALU 和地址产生单元

15.2.1　数据 ALU 辅助寄存器（D8，D9）

D8 和 D9 是 2 个 96 位寄存器，它们主要用于 4 指令基 2FFT 蝶形运算。它们仅能在相乘运算和在 MOVE 指令中的源或目的操作数中是源操作数。这些寄存器对额外相乘器输入寄存器、流水线寄存器，为编译程序的暂存保持常数有用。

15.2.2　数据 ALU 通用寄存器（D0～D7）

D0～D7 是 8 个通用寄存器，在 MOVE 指令和算术运算中它们之间没有区别。对大多数数据 ALU 指令都作为数据 ALU 的源和目的操作数。

15.3　地址寄存器组（R0～R3 和 R4～R7）

8 个地址寄存器 R0～R7 是 32 位宽，可包含地址和通用数据。在所选地址寄存器中 32 位地址用于操作数有效地址的计算。此地址可直接指向数据或可以由寄存器偏置加以修正。大多数寻址方式以读-修正-写的形式修正所选地址寄存器。典型地，地址寄存器被访问，用作有关模算术单元的输入，被算术单元修正和写回所选寄存器。地址寄存器修正的形式由偏置和修正寄存器内容所控制。地址寄存器的内容可传送到/自暂时地址寄存器保持的有效地址。

15.4 偏置寄存器组（N0～N3 和 N4～N7）

8 个偏置寄存器 N0～N7 是 32 位宽，并包含在地址寄存器更新计算中用于递增和递减地址寄存器的偏置值，或可用于一般目的存贮。此外，偏置寄存器内容可按某些速率通过一个表进行跳步来产生波形，或可指定表的偏置或表的基础。

15.5 修正寄存器组（M0～M3 和 M4～M7）

8 个修正寄存器 M0～M7 是 32 位宽，并包含在地址寄存器更新计算中（即线性逆进位和模数），用于指定地址算术形式的值，或用于一般目的的存贮。当指定模运算时，修正寄存器也将指定要用的模值。为了相同号码的地址寄存器更新计算，修正寄存器将被访问（即当 R0 要更新时，M0 被访问，M1 要更新时 R1 被访问）。每个修正寄存器在处理器复位时置定到 $FFFFFFF，该值指定线性算术寄存器更新计算的缺省值。

15.6 程序计数器（PC）

32 位寄存器包含要从程序存贮器空间提取的下一个单元的地址。PC 可指示指令，数据操作数或操作数地址。对这寄存器访问通常是固有的，并由大多数指令所隐含。当程序开始循环时，这个特定用途的地址寄存器被推入堆栈，并完成跳到子程序，产生中断。

15.7 状态寄存器（SR）

SR 是 32 位寄存器，由 8 位方式寄存器（MR）、8 位 IEEE 异常寄存器（IER）、8 位异常寄存器（ER）和 8 位条件码寄存器（CCR）组成。

MR 位仅受处理器复位、异常处理、DO、DORENDDO、ILLEGAL、RTI、RTR、FTRAPcc 和 TRAPcc 指令以及直接涉及 MR 寄存器的指令所影响。

IER 位受处理器复位，直接涉及 IER 寄存器的指令和数据 ALU 浮点运算的影响。IER 包含 IEEE 舍入方式控制和 5 个由 IEEE 754 标准定义的异常标志。5 个异常标志是“粘的”，清除它们的唯一方法是硬件复位或由用户写 IER 寄存器。使这些位粘住的目的是防止它们被处理之前或其他指令后面用的时候偶然被清除。

ER 位受处理器复位、直接涉及 ER 寄存器的指令和数据 ALU 浮点运算的影响。ER 反映最后一指令执行结果产生的异常。

CCR 位包含反映数据 ALU 指令当前执行产生的状态的标志。CCR 位受数据 ALU 操作和直接涉及 CCR 寄存器的指令的影响。

SR 寄存器在程序循环初始化时装入堆栈，完成跳或分支到子程序。SR 格式示于图 15－3。

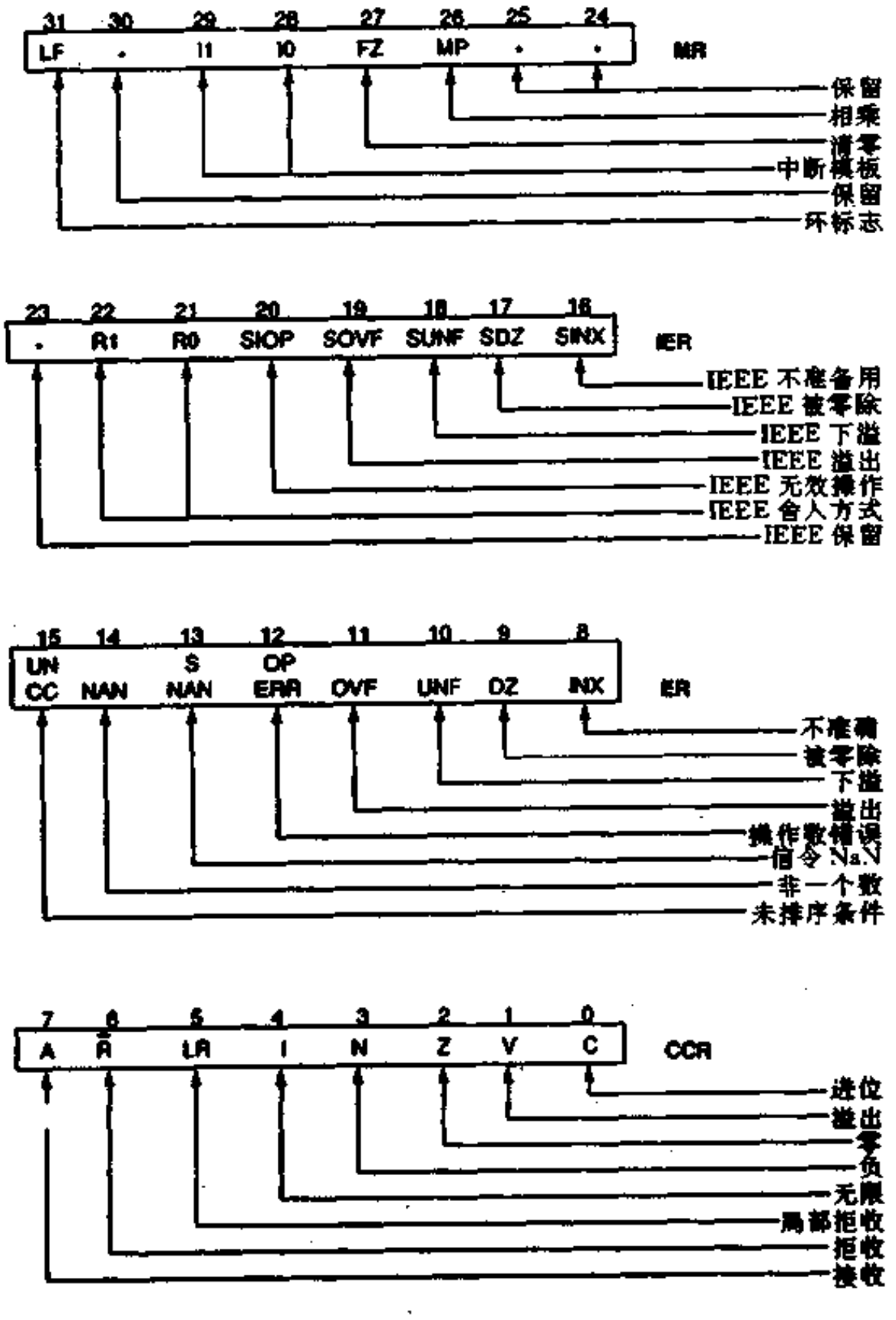


图 15-3 SR 格式

15.7.1 CCR 进位（C）位 0

若在整数相加中产生进位或在整数相减中产生借位，则设置进位位。进位位也被位操作、旋转和移位整数指令以及执行 MOVETA 指令时的地址产生单元操作所修正。进位位不受浮点指令的影响。当处理器复位时 C 位被清除。

15.7.2 CCR 溢出（V）位 1

若在定点运算中发生算术溢出，则设置溢出位。这意味着此结果不能表示在最后长度中。V 位不受浮点运算影响，除非它有定点结果。溢出位在执行 MOVETA 指令时也被地址产生单元操作修正。

15.7.3 CCR 零（Z）位 2

若在浮点中结果等于加或减 0。或在定点运算中结果等于 0，则设置零位。零 位在执行 MOVETA 指令时也被地址产生单元操作修正 。当处理器复位时 Z 位被清除。

15.7.4 CCR 负（N）位 3

若在浮点运算中结果是负或定点运算中结果是 0，则设置负位。在执行 MOVETA 指令时，负位也被地址产生单元的操作修正。当处理器复位时，N 位被清除。

15.7.5 CCR 无限（I）位 4

若浮点运算结果是无限，设置无限位。I 位不受定点运算影响。当处理器复位时，I 位被清除。

15.7.6 CCR 局部拒绝（LR）位 5

局部拒绝位用来拒绝浮点或定点操作数在图形应用中的测试。当处理器复位时，LR 位被清除。

15.7.7 CCR 拒斥（R）位 6

全局拒斥位用来拒绝浮点或定点操作数在图形应用中的测试。当处理器复位时，$\overline{R}$ 位被清除。

15.7.8 CCR 接收（A）位 7

接收位用来接收浮点或定点操作数在图形应用中的测试。当处理器复位时，A 位被清除。

15.7.9 ER 不准确（INX）位 8

若浮点结果不准确，则设置不准确位。当来自数据 ALU 操作的中间结果尾数舍入到指定的精度时此情况发生。若传送到 DN 寄存器的舍入尾数不同于未舍入中间结果的尾数，产生精度的损失，则将设置 INX 位。INX 位不受定点操作影响。当处理器复位时，INX 位被清除。

15.7.10 ER 被 0 除（DZ）位 9

DSP96002 中 DZ 标志可由软件设置作为 FDIV 程序的部分。没有单 DSP96002 指令可设置 DZ 标志。当处理器复位和全浮点指令时 DZ 位被清除。

15.7.11 ER 下溢（UNF）位 10

若在浮点数据寄存器中的浮点运算结果太小（在$\pm 2^{Emin}$之间）无法表示，则设置下溢位。舍入以前在指数上作测试。去归一化结果将设置 UNF 位。UNF 位不受定点操作影响。当处理复位时，UNF 位被清除。

15.7.12 ER 上溢（OVF）位 11

若在以指定舍入精度作为非归一化结果的浮点数据寄存器中浮点结果太大无法表示，则

设置上溢位。舍入尾数后（舍入尾数的结果$\geqslant 1.0\times 2^{Emin+1}$）按指数作测试。根据舍入方式和结果的符号，作出判决返回结果将是什么。这返回结果是最后的舍入结果。例如，不设置OVF的最大正SP结果对所有舍入方式是$7F7FFFFF。

15.7.13 ER操作数错误（OPERR）位12

若在给定操作数时，一个操作没有数学意义，则设置操作数错误。设置OPERR位的操作数例子是（+∞）+（-∞），OX∞和$\sqrt{-n}$。OPERR位不受定点操作影响。当处理器复位时，OPERR被清除。

15.7.14 ER信令NaN（SNAN）位13

当信令NaN（Not-a-Number）包含在算术浮点操作中时，设置信令NaN位。例如，"FABS.S D"（D进SNaN）将设置SNaN位并返回无变化的NaN。SNAN位不受定点操作影响。当处理器复位时SNAN位被清除。信令NAN可用的一例是给未初始化的存贮器一个已知值，它可用于标志用户。

15.7.15 ER Not-a-Number（NAN）位14

若浮点运算结果是NaN，则设置NAN位。例如，DSP96002设置NaN位作为设置OPERR位操作的结果。NAN位不受定点操作影响，但受一些变换指令的影响。例如，"INT D"（D是NAN）将返回定点值$FFFFFFFF，并设置NAN位。当处理器复位时，NAN位被清除。

15.7.16 ER无序条件（UNCC）位15

若当NaN位设置时执行一不知道的浮点条件指令（FBcc、FJcc、FIFcc等）则设置无序条件位。由指令测试的结果取决于实数线上可能出现的操作数。按定义，若操作数是NaN，不可能排序或出现在实数线上，所以要设置UNCC位。

15.7.17 IER IEE不准确标志（SINX）位16

IEEE不准确标志是对因陷阱而不能操作的IEEE标志。当操作的舍入结果不准确或若没有溢出陷阱而溢出时，则设置此标志位（即INX位由当前的或以前的指令设置）。

15.7.18 IER IEEE被零除标志（SDZ）位17

IEEE被零除标志是对因陷阱而不能操作的IEEE标志，若被除数是有限非零数而除数是零则设置比标志。

15.7.19 IER IEEE下溢标志（SUNF）位18

IEEE下溢标志是对因陷阱而不能操作的IEEE标志。当所检测的精度有损失，则设置此标志（即INX位和UNF位在当前或以前指令中同时设置）。

15.7.20 IER IEEE溢出标志（SOVF）位19

IEEE溢出标志是对因陷阱而不可操作的IEEE标志。当目的格式的最大有限数的大小被

已舍入浮点结果所超过，此标志被设置（即 OVF 位由当前或以前的指令设置）。

15.7.21 IER IEEE 无效操作标志（SIOP）位 20

IEEE 无效操作标志是对因陷阱不可操作的 IEEE 标志。若对要完成的操作，一操作数无效，则设置此标志（即 OPERR 位由当前和以前的指令设置）。

15.7.22 IER 舍入方式（R0～R1）位 21，22

舍入方式位 R1 和 R0 指定在浮点运算中不准确结果应当舍入的方法。当处理器复位时舍入方式位被清除。

R1	R0	舍入方式
0	0	舍入到最近的偶数
0	1	舍入向 0
1	0	舍入向－∞
1	1	舍入向＋∞

数据 ALU 完成结果的舍入到指令所指定的精度。DSP96002 仅支持单扩展和单精度结果。DSP96002 实现 4 种由 IEEE 标准指定的舍入方式。这些方式是舍入到最近的（RN），向 0 舍入（RZ），向＋∞舍入（RP）和向－∞舍入（RM）。舍入定义列于下。

RN 舍入到最近的偶数，此方式中，可表示的最接近精确值的值将作为结果送出。若两个最接近值是相等地接近，最低有效位等于 0（偶数）的一个将是结果，例如 1.65 舍入到 1.6，而 1.75 舍入到 1.8。

RZ 向零舍入。此方式中，结果是最靠近值的，而数值不大于无限精度结果。这种方式有时称“截断方式”，因为舍入点右边的位被删除，例如 1.65 舍入为 1.6，而－1.65 舍入到－1.6。

RM 向－∞舍入。此时，结果是最靠近值的，而不大于无限精确结果，例如，1.65 舍入到 1.6，而－1.65 舍入到 －1.7。

RP 向＋∞舍入。例如，1.65 舍入到 1.7,，而－1.65 舍入到 －1.6。

15.7.23 保留状态（位 23，24，25）

这些位为将来的扩展而保留，并在读操作时作为 0 读入。它们为将来的兼容性应当写 0。

15.7.24 MR 相乘精度控制（MP）位 26

相乘精度控制位指定在 FMPY/FADD，FMPY/FADDSUB 和 FMPY/FSUB 指令中相乘操作的输出精度。若 MP 被清除，则相乘操作的输出精度由伴随指令（FADD，FADDSUB，或 FSUB）所决定。若 MP 被设置，则查乘操作输出精度是由硬件支持的最大精度。

例如，若 MP＝0 且伴随指令是 FADD.S，则相乘输出精度是单精度。若 MP＝1 且伴随指令是 FADD.S，则相乘输出精度将是单扩展精度。若伴随指令是 FADD.X，则相乘输出精度是单扩展精度与 MP 状态无关。

MP	相乘精度控制
0	输出精度取决于伴随指令
1	最大输出精度

15.7.25 嵌入 0（FZ）位 27

嵌入 0 位指定控制浮点下溢的两种方式之一，即 IEEE 用去归一化数的逐渐下溢方式和嵌入 0 方法。若 FZ 被清除，则在与 IEEE 754—1985 浮点标准完全一致下处理浮点下溢。若数据 ALU 源操作数或结果是去归一化数，则 IEEE 下溢方式可能对归一化和去归一化分别插入指令周期。若 FZ 被设置，则浮点下溢嵌入 0。任何去归一化源操作数认为是 0，且任何下溢结果嵌入 0。

FZ	说明
0	IEEE 随去归一化数逐渐下溢
1	嵌入 0

15.7.26 MR 中断模板（I1～I0）位 28，29

中断模板位 11 和 10 反映处理器当前优先级，并指出中断源到要中断处理器所需要的中断优先级（IPL）。处理器当前的优先级在软件控制下可以改变。

11	10	异常允许	异常模板
0	0	IPL0，1，2，3	无
0	1	IPL1，2，3	IPL0
1	0	IPL2，3	IPL0，1
1	1	IPL3	IPL1，2

15.7.27 保留状态（位 30）

这一位为将来扩展保留，并当读操作时将读为 1。为将来的兼容性它应写 1。

15.7.28 MR 循环标志（LF）位 31

当程序循环进行中设置循环标志位，并由电路检测循环的结束。循环标志是的当程序循环终止时恢复的唯一 SR 位。当开始和退出程序循环时，分别堆积和恢复循环标志，允许程序循环嵌套。

15.8 循环计数器（LC）

循环计数器是专用的 32 位计数器，用于指定硬件程序循环的重复次数。此寄存器内容由 DO 指令存入堆栈，而由循环处理结束或执行 ENDDO 指令而从堆栈中取出。当硬件程序循环

到达结束时，循环计数器的内容按 1 测试。若循 环计数器是 1，程序循环终止，且以前面存在堆栈中的 LC 内容载入 LC 寄存器。若计数不是 1，则递减 1 并重复程序循环。循 环计数器可在程序控制下读。这使当执行时，已经执行过的循环次数可以决定。LC 也用在 REP 指令中。

15.9 循环地址寄存器（LA）

循环地址寄存器指示程序循环中最后指令字的单元。这个寄存器内容由 DO 指令存入堆栈，而由循 环处理的结束或执行 ENDDD 指令从堆栈中取出。当包含在此寄存器中的地址指令字被锁住时，检验 LC 的内容。若它不是 1，则 LC 递减，且从系统堆栈顶端的地址取下一个指令。否则 PC 递增，循环标志恢复（从堆栈拉出），堆 栈被清除，LA 和 LC 寄存器从堆栈拉出，并恢复指令正常地执行。LA 寄存器是由 D0 指令写入的 32 位读/写寄存器，由系统堆栈读出。

15.10 系统堆栈（SS）

系统堆栈是独立的内部 RAM，有 15 单元“深”，并分成两块：高（SSH）和低（SSL），每个 32 位。SSH 存贮 PC 或 LA 内容，SSL 存贮 LC 或 SR 内容。

PC 和 SR 寄存器为子程序调用和长中断被推入堆栈。这些寄存器可用 RTS 指令返回子程序和用 RTI 指令返回中断从堆栈拉回。系统堆 栈也用于存贮硬件程序循环的起始指令地址以及 SR、LA 和 LC 寄存器恰优先于循环开始的内容。这允许 DO 循环嵌套。

高达 15 长的中断，7 个 D0 循环、15 个 JSR、或它们的组合可由堆栈提供。当达到堆栈极限时必须小心。当超过堆栈极限时，要入堆栈的数据将丢失，并且将发生不可屏蔽的堆栈错误中断。

15.11 堆栈指针（SP）

堆栈指针寄存器（SP）是 32 位寄存器，它指示系统堆栈顶端的单元和堆栈的状态（下溢和溢出的错误情况）。堆栈指针由某些指令所隐含（DO、ENDDO、REP、JSR、RTI 等），或直接涉及（MOVEC、MOVEI、MOVEM、MOVEP 和 MOVES 指令）。堆栈指针寄存器格式示于图 15－4。堆栈指针寄存器按 6 位计数器实现，它选 15 个单元堆栈具有 4 个最低有效位。可能的堆栈值示于图 15－5。

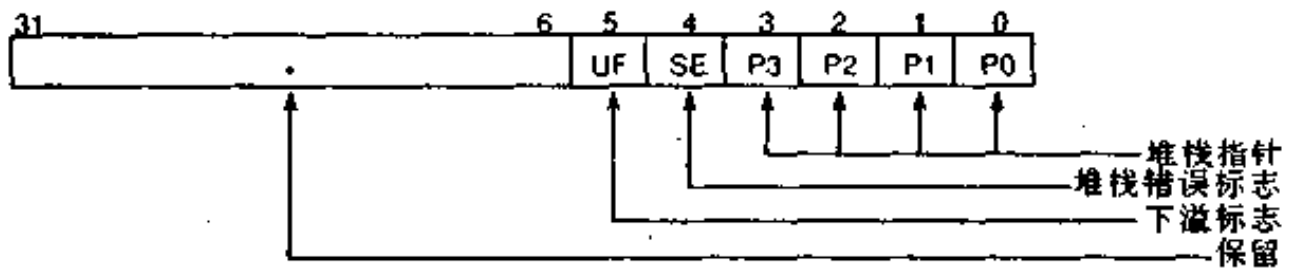


图 15－4 堆栈指针格式

UF	SE	P3	P2	P1	P0	说明
1	1	1	1	1	0	重拉后堆栈下溢条件
1	1	1	1	1	1	堆栈下溢条件
0	0	0	0	0	0	堆栈空(复位)，拉引起下溢
0	0	0	0	0	1	堆栈单元 1，重拉引起下溢
0	0	0	0	1	0	堆栈单元 2.
.	.	.	.	.	.	
.	.	.	.	.	.	
.	.	.	.	.	.	
0	0	1	1	0	1	堆栈单元 13.
0	0	1	1	1	0	堆栈单元 14。重推引起溢出
0	0	1	1	1	1	堆栈单元 15.(堆栈满)，推引起溢出
0	1	0	0	0	0	堆栈溢出条件
0	1	0	0	0	1	重推后堆栈溢出条件

图 15－5　堆栈指针值

15.11.1　堆栈指针（SP）位 0，1，2，3

堆栈指针（SP）指出堆栈上最后所用地方。硬件复位后，这些位立即被清除（SP＝0），表明堆栈是空。

数据推入堆栈时 SP 递增 1，于是在新堆栈单元 SP 处写项目。从单元 SP 复制一个项目，再使 SP 递减 1，就拉出了该项目。传送指令读 SSH 隐含着 SP 递减，而传送指令写 SSH 隐含着 SP 递增。在软件控制下管理堆栈。因堆栈指示的每个单元是 64 位，故必须由两条传送指令访问。第一条传送指令必须去/自 SSL，而第二条传送指令应当去/自 SSH 以自动触发 SP 递增/递减。

15.11.2　堆栈错误标志（SE）位 4

堆栈错误标志（SE）表示发生堆栈错误。SE 从 0 到 1 过渡使优先级 3 堆栈错误异常。

当堆栈全满时，堆栈指针读 001111，这时任何推入数据到堆栈的操作将发生堆栈错误异常，而堆栈寄存器将读 010000（若隐含的双重推入发生则为 010001）。

任何隐含的拉出操作（对 SP＝0）将造成堆栈错误异常，且 SP 将读全 1。如图 15－5 所示，SE 位被置定。

一旦置定，SE 标志一直保留到传送或位指令（它们直接明显地访问堆栈指针）清除 SE 标志为止。SE 标志也由硬件复位清除。当 SP＝0（堆栈空），无堆栈级被选。指令读没有 SP 后递减的堆栈肯定不造成堆栈错误异常，并且所读数据将不确定。指令写没有 SP 前递增的堆栈肯定不造成堆栈错误异常，且没有堆栈寄存器改变。

15.11.3　下溢标志（UF）位 5

当堆栈发生下溢时，下溢标志被 置定。当堆栈发生溢出时，UF 标志被清除。当 SE 保持置定时，UF 标志不随隐含 堆栈指针的指令（如 RTI、RTS、DO、ENDDO、JSR 等）引起的堆栈指针操作而改变。UF 标志可由硬件复位清除。只要 SE 未置定，不产生堆栈错误的隐含堆栈指针操作，总会清除 UF。

15.11.4 不设置堆栈指针寄存器位（6～31）

任何不设置堆栈指针寄存器的位是为将来扩展用的，并在DSP96002读操作时读为0。它们为了将来的兼容性应当写为0。

15.12 操作方式寄存器（OMR）

操作方式寄存器（OMR）是32位寄存器，它定义当前处理器的片操作方式。OMR位仅受处理器复位和直接访问OMR指令的影响。其格式示于图15－6。

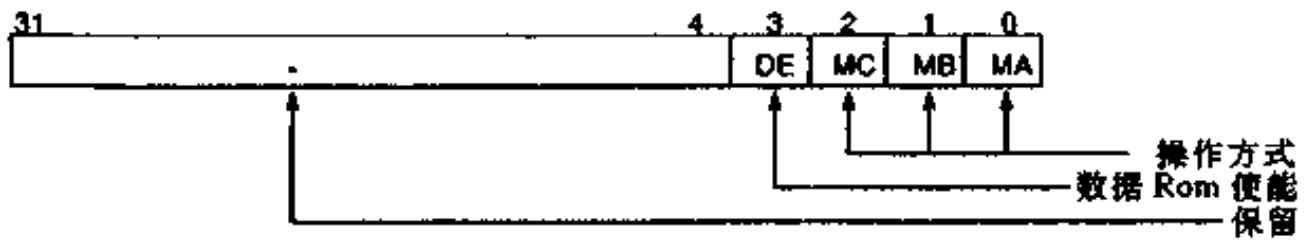


图15－6 操作方式寄存器格式

15.12.1 片操作方式位（0，1，2）

在片脱离RESET状态时，操作方式位MA、MB和MC确定是否起动内部程序RAM和起动步骤。这些位在$\overline{\text{RESET}}$端为负时，分别由外部方式选择端MODC、MODB和MODA载入。在DSP96002脱离RESET状态后，在程序控制下MC、MB和MA可能改变。

15.12.2 数据ROM起动位3

数据ROM起动（DE）位起动两个位于X和Y存贮空间中地址为$00000400～$000007FF的片上512×512数据ROM。当DE被清除时，$00000200到$000007FF空间是外部X和Y数据空间的部分，并且片上数据ROM不能起动。

15.12.3 保留的操作方式寄存器（位4～31）

这些操作方式寄存器位是为将来扩展而保留的，当DSP96002读操作时读为0。为了将来兼容它们应用零写入。

第十六章　数据组织和寻址方式

16.1　操作数大小

操作数大小定义如下：1 字节为 8 位长，1 短字为 16 位长，1 个字为 32 位长，1 长字为 64 位长。对浮点运算操作数大小定义如下：单实是 32 位，双实是 64 位，寄存器操作数是 96 位。每条指令操作数大小或明显地编在指令中，或由指令操作隐含地定义。

16.2　存贮器中的数据组织

程序存贮器是 32 位宽并支持 32 位指令字和指令扩展字。

X 和 Y 数据存贮器每个 32 位宽并支持字和单实操作数。X 和 Y 存贮器可作为单 64 位宽存贮器空间（“L”空间）以支持长字和双实操作数。

16.2.1　整数存贮器数据格式

DSP96002 支持 4 种整数存贮器数据格式：

- 带符号的字整数：32 位宽以 2 的补数表示。
- 带符号的长字整数：64 位宽以 2 的补数表示。
- 不带符号的字整数：32 位宽以不带符号的绝对值表示。
- 不带符号的长字整数：64 位宽以不带符号的绝对值表示。

对带符号整数的位加权表示在图 16－1 中。对不带符号整数的位加权示于图 16－2。

DSP96002 不支持在长字整数上直接操作，但长字整数可成为一些 ALU 操作结果或长传结果。

16.2.2　浮点存贮器数据格式

DSP96002 支持二种浮点存贮器数据格式：单精度（32 位）和双精度（64 位），都符合对二进制浮点运算的 IEEE 标准 754。DSP96002 支持的浮点操作数存贮器格式示于图 16－3。遵从 IEEE 754 标准的单和双实操作数的存贮器格式在下面示出。

注意所存指数（e）是无符号的并且位置在尾数上面的有效位中。无偏指数 E 的范围是在 Emin 和 Emax 之间的每个整数，包含（－Emin≤E≤Emax）。对单精度（SP），Emin＝－126，而 Emax＝＋127；对双精度（DP），Emin＝－1022，而 Emax＝＋1023。对 SP 和 DP，Emin－1 用于编码±0 和去归一化数，而 Emax＋1 用于编码±∞和 NaN。

1. IEEE 单精度实存贮器格式总结

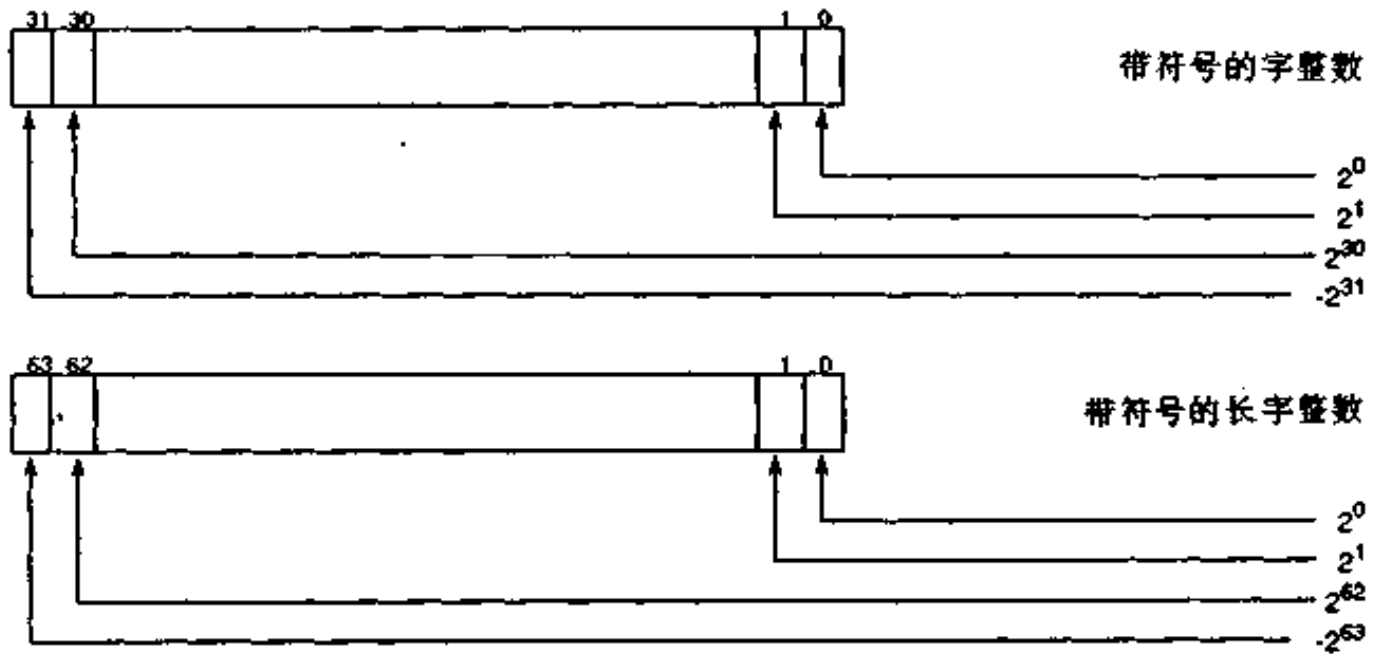


图 16－1　带符号整数操作数的位加权和定位

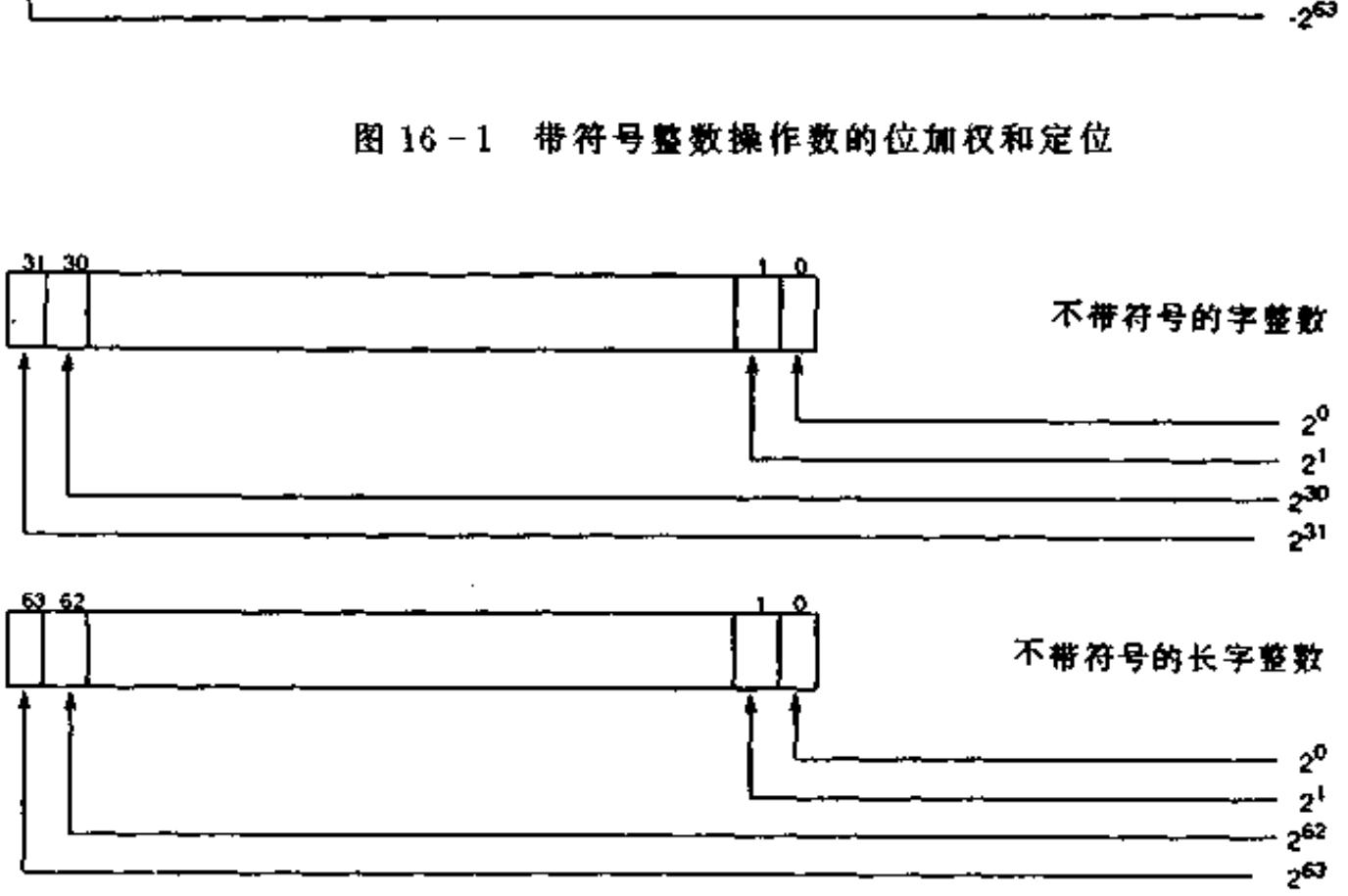


图 16－2　无符号操作数的位加权和定位

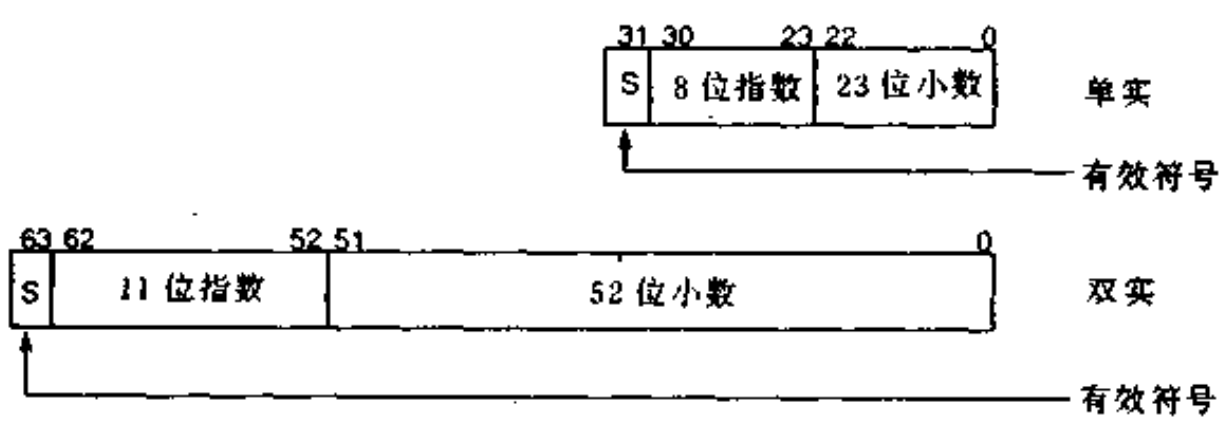
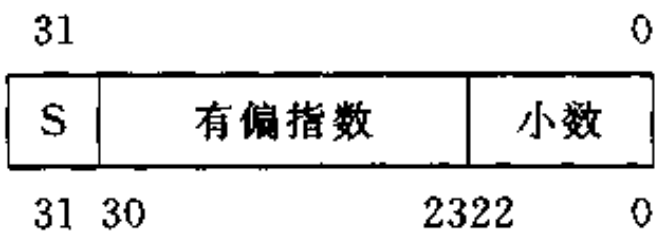


图 16－3　浮点操作数的存贮器格式



字段大小（位）

s＝符号……1

e＝有偏指数……8

f＝小数……23

符号解释：

正尾数： s＝0

负尾数： s＝1

归一化数：

表示实数的形式 $(-1)^S \times 2^{(E+127)} \times 1.f$

E……无偏指数 $-126 \leqslant E \leqslant +127$

e 的偏置……＋127（$7F）

e＝E＋偏置……$0 \leqslant e \leqslant 254$（$FE）

f……零或非零

尾数……1.f

去归一化数：

表示实数的形式 $(-1)^s \times 2^{(Emin-1+127)} \times 0.f$

e 的偏置……＋127（$7E）

e……0（$00）

f……非零

尾数……0.f

带符号的零：

表示实零的形式 $(-1)^s \times 2^{(Emin-1+127)} \times 0.0$

e 的偏置……＋127（$7F）

e……0（$00）

f……零

尾数……0.f＝0.00…00

带符号的无穷大：

表示实无穷大的形式 $(-1)^s \times 2^{(Emax+1+127)} \times 1.0$

e 的偏置……＋127（$7F）

e……255（$FF）

f……零

尾数……1.f＋1.00…00

NaN（Not－a－Number）：

表示 NaN 用 $2^{(Emax+1+127)} \times 1.f$

s……不用管

e 的偏置……n.a.

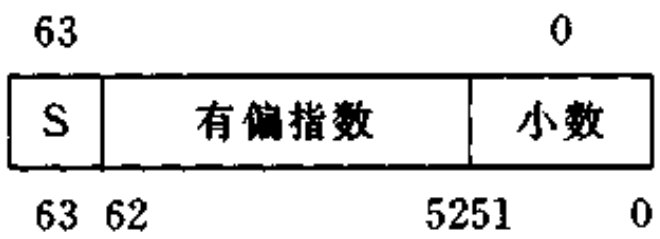
e……255（$FF）

f……非零：11…11　　内部　（合法）QNaN

1×…××　识别的 QNaN

0×…××　SNaN

2. 双精度实存贮器格式总结



字段长度（位）：

s=符号……1

e=有偏指数……11

f=小数……52

符号的解释：

正尾数：s=0

负尾数：s=1

归一化数：

表示实数的形式 $(-1)^s\times 2^{(E+1023)}\times 1.f$

E……无偏指数 $-1022\leqslant E\leqslant +1023$

e 的偏置……+1023（$3FF）

e+E+bias……$0\leqslant e\leqslant 2046$（$7FE）

f……零或非零

尾数……1.f

去归一化数：

表示实数的形式 $(-1)^s\times 2^{(Emin-1+1023)}\times 0.f$

E_{min}……−1022

e 的偏置……+1023（$3FF）

e……0（$000）

f……非零

尾数……0.f

带符号的零：

表示实零的形式 $(-1)^s\times 2^{(Emin-1+1023)}\times 0.0$

e 的偏置……+1023（$3FF）

e……0（$000）

f……零

尾数……0.f=0.00…00

带符号的无穷大：

表示无穷大的形式 $(-1)^s\times 2^{(Emax+1+1023)}\times 1.0$

e 的偏置……n.a.

e……2047（$7FF）

f……零

尾数……1.f=1.00…00

NaNs（Nat－a－Namber）：

表示 NaN 为 $2^{(Emax+1+1023)}\times 1.f$

s……不用管

e 的偏置……n.a.

e……2047（$7FF）

f……非零：11…11　　内部（合法）QNaN

1×…××　识别的 QNaN

0×…××　SNaN

16.3　寄存器中的数据组织

16.3.1　数据 ALU 寄存器

30 个数据 ALU 寄存器是 32 位宽，并可作为字操作数访问。2 个数据 ALU 寄存器组可连接形成 10 个 64 位寄存器，它可作为长字被访问。最低有效位（LSB）是最右位（位 0），而最高有效位（MSB）是位 31 或 63（对整操作数）。

3 个数据 ALU 寄存器组可连接形成 10 个 96 位寄存器，它可作为单实或双实操作数被访问。浮点操作数通常以内部双精度格式表示。

1. 内部浮点数据格式

全部 DSP96002 内部浮点操作都用单扩展精度完成。当写入数据 ALU 寄存器时，所有操作数变换为内部双精度格式。内部双精度浮点格式用于 10 个浮点数据寄存器中，如图 16－4 所示。

S 是尾数的符号。

U 是单精度非归一化标记。

V 是单扩展精度非归一化标记。

Biased Exponent 是 11 位数，它本质上是 11 位双精度偏置指数。

Zero 是通常由浮点运算和浮点传送所清除的位。

I 是尾数的整数部分。

Fraction 是 52 位字段表示尾数的小数部分。

图 16－4　浮点寄存器中数据格式

当内部运算结果写入数据 ALU 寄存器或当把表示在存贮器数据格式之一的单或双精度数写入数据 ALU 寄存器作为 MOVE 操作的结果时，完成自动格式到内部双精度表示的变换。因此隐含地支持混合方式运算。

因为 DSP96002 实现单扩展精度内部计算，所以寄存器中小数部分对单扩展精度结果仅

包含 31 个有效位，或对单精度结果包含 23 个有效位。但是，若双精度 MOVE 完成，52 位小数将写入寄存器，但若同样的寄存器用作浮点操作数，则仅 31 个小数的最高有效位被使用，而其他位被数据 ALU 所忽略，使截断误差趋于 0。所以，为将来的兼容性，仅单扩展精度数据应随双精度数据传送而传送。

2. 内部双精度格式总结

字段大小（位）：

s＝符号……1

e＝偏置指数……11

u＝U 标记……1

v＝V 标记……1

i＝整数部分……1

f＝小数……52

z＝不用位……29

不用位解释：

输入……无关

输出……全 0

符号位解释：

正尾数：s＝0

负尾数：s＝1

归一化数：

表示实数的形式 $(-1)^s \times 2^{(e-1023)} \times 1.f$

e 的偏置……＋1023（$3FF）

e……0＜e＜2047（$7FF）

i……1

f……零或非零

尾数……i.f＝1f

去归一化的数：

表示实数的形式 $(-1)^s \times 2^{(-1022)} \times 0.f$

e 的偏置……＋1022（$3FE）

e……0（$000）

i……0

f……非零

尾数……i.f＝1.f

带符号的零：

e 的偏置……n.a.

e……0（$000）

i……0

f……零

尾数……i.f=0.00…00

带符号的无穷大：

e 的偏置……n.a.

e……2047（$7FF）

i……1

f……零

尾数……i.f=1.00…00

NaN（Not-a-Number）：

s……无关

e 的偏置……n.a.

e……2047（$7FF）

i……1

f……非零

尾数……i.f：1.11…11　　合法 QNaN

1.1×…××　QNaN

1.0×…××　SNaN

16.3.2　地址产生单元（AGU）寄存器

符号 Rn 用于指定 8 个地址寄存器 R0～R7 之一。Nn 用于指定 8 个地址偏置寄存器 N0～N7 之一。Mn 用于指定 8 个地址修正寄存器 M0～M7 之一。8 个 AGU 地址寄存器 R0～R7 支持 32 位的地址或数据操作数。8 个 AGU 偏置寄存器 N0～N7 支持 32 位偏置或 32 位地址或数据操作数。8 个 AGU 修正寄存器 M0～M7 支持 32 位的修正或 32 位地址或数据操作数。

16.3.3　程序控制寄存器

操作方式寄存器（OMR）是 32 位宽，并可作为字节或字操作数访问。状态寄存器（SR）是 32 位，与系统方式寄存器（MR）一起占有高 8 位，IEEE 异常寄存器（IER）占次 8 位，异常寄存器（ER）占据随后的 8 位，用户条件码寄存器（CCR）占据低 8 位。SR 寄存器可作为字操作数访问。MR、IER、ER 和 CCR 可作为字节操作数访问。循环计数寄存器（LC）、循环地址寄存器（LA）、系统堆栈指针（SP）、系统堆栈高（SSH）和系统堆栈低（SSL）是 32 位宽并可作为字操作数访问。

程序计数器寄存器（PC）是特殊的 32 位程序控制寄存器，通常被认为是字操作数。

系统堆栈是 64 位宽并为子程序调用、中断和程序循环支持 PC 和 SR 寄存器连接（PC：SR），同时为程序循环也支持 LA 和 LC 寄存器连接（LA：LC）。

16.4 NaN 的实现

当由 DSP96002 产生时，静 NaN (Quiet Not - a - Number) 表示没有数学意义的运算结果（如 0 乘无穷大），或涉及作为输入的 NaN 操作数的运算结果。

用小数的最高有效位 (MSB) 微分实现 NaN 的两种型式。小数的最高有效位置于 1 的 NaN 是静 NaN (QNaN)，或称无信号 NaN。最高有效小数位等于 0 的 NaN 是有信号 NaN (SNaN)。DSP96002 从不产生 SNaN 作为运算结果。

DSP96002 合法 QNaN 定义如下：

- 对所有精度有相同模式。
- 小数的所有位置于 1。
- 偏置指数置于全 1。
- 清除符号位。
- 在内浮点格式中，I 位通常置于 1；注意若 I 位置于 1，模式不被数据 ALU 硬件识别为合法模式，并且在这些位上操作模式可能产生不希望的结果。

16.5 自动的浮点格式转换

在 DSP96002 中有两种自动浮点格式转换：

(1)任何存贮器数据格式中浮点操作数转换到浮点数据寄存器的双精度内部数据格式。当从外部单元传送数据到数据 ALU 浮点寄存器时作此事。

(2) 在浮点数据寄存器的内部数据格式中浮点操作数转换到任何存贮器数据格式。当从数据 ALU 浮点寄存器传送数据到外部单元时作此事。

16.5.1 转换到双精度内部数据格式

因为 DSP96002 数据 ALU 所用的内部数据格式是双精度的，所以所有外部浮点操作数在写入数据 ALU 浮点寄存器以前转换为双精度值。转换实际是用图 16 - 5 所示的“位重排”操作过程。

当把单精度数转换为内部寄存器数据格式时，隐含位被显露并作为显式位存在寄存器中。若要转换的数是去归一化单精度浮点数，则 U 标记被置定，指示一去归一化数。若这种数用作浮点操作的操作数，两种情况的发生取决于 SR 中 FZ (Flush - to - Zero) 的状态。在 FZ 方式中，操作数在计算中被认为是 0。然而，存在寄存器中的数据不会受影响（除非寄存器也是当前操作的目的）。在 IEEE 方式中，操作数为归一化将首先被加到执行周期的额外周期进行“校正”。但是，存在寄存器中的数据不会受影响。

当双精度数转换到内部寄存器数据格式时，隐含位被显露并以显示位存在寄存器中。若要转换的数是去归一化双精度浮点数，则 V 标记被置定。若这种数要用作浮点操作的操作数，两种情况的发生取决于 FZ 位在 SR 中的状态。在 FZ 方式中，操作数在计算中被认为是 0。但是存在寄存器中的数据不受影响。在 IEEE 方式中，乘操作数为归一化将首先被加到执行周期

单精度→双精度 存贮器格式	内部格式
31→95	S
94	U-若去归一化则设置,否则清除
93	V-清除
92	清除
75	清除
30→74	
73	若 NAN 或不定则设置,若零则清除,否则倒置(位 30)
72	若 NAN 或不定则设置,若零则清除,否则倒置(位 30)
71	若 NAN 或不定则设置,若零则清除,否则倒置(位 30)
29→70	
·→·	
23→64	
63	1-若去归一化或零则清除,否则设置
22→62	
·→·	
0→40	
39	清除
·	·
0	清除

双精度→双精度 存贮器格式	内部格式
63→95	S
94	U-精除
93	V-若去归一化则设置,否则清除
92	清除
·	
75	清除
62→74	
·→·	
52→64	
63	1-若去归一化或零则清除,否则设置
51→62	
·→·	
0→11	
10	清除
·	·
0	清除

图 16-5 转换为双精度内部数据格式

的额外周期所“缠绕”。但存在寄存器中的数据不受影响。DSP96002 不支持双精度,但确支持单扩展精度。

16.5.2 转换到存贮器格式

从内部双精度格式转换到两个存贮器浮点格式之一,是在数据寄存器要存入存贮器时完成,或在存入任何其他数据 ALU 以外的单元时完成。转换实际是由 MOVE 指令自动完成的“位重排”操作,它仅对从寄存器收集要求的位和对构成要存入存贮器的 32 位或 64 位字段负责。仅当寄存器中的数据等于指定的 MOVE 精度时,将产生正确结果。例如,为了单精度 MOVE,数据必须舍入到单精度。

双精度转换到单精度(不是格式转换)由指定适当的舍入操作来完成(可能是显式指令如 FTFR.S 或隐式操作如 FADD.S)。舍入后的结果仍存在内部双精度格式中。但是,由数据 ALU 读出结果的 MOVE 指令不会因位重排而改变此值。图 16-6 表示 MOVE 指令完成的位重排过程。

若双精度值要舍入到单精度,并且舍入结果应产生去归一化数,则由于 SR 中的 FZ 位可完成二种不同作用。在 FZ 方式中,结果将以 0 存入寄存器。在 IEEE 方式中,操作数将首先为去归一化被加到执行周期的额外周期所“校正”。但是,存在寄存器的数据将是内部双精度格式和 U 标记被置定。U 标记表示,若另一数据 ALU 操作用此结果作为操作数,则在用它以前为了操作数归一化应加额外周期。

双精度→单精度 内部格式　存贮器格式	双精度→双精度 内部格式　存贮器格式
95→31	95→63
94	94
.	75
75	74→62
74→30	.→.
73	64→52
72	63
71	62→51
70→29	.→.
.→.	11→0
64→23	10
63	0
62→22	
.→.	
40→0	
39	
.	
0	

图 16-6　从内部格式到存贮器格式的转换

16.6　操作数访问

DSP96002 把操作数访问分为四种：程序、堆栈、寄存器和存贮器访问。对指令所要求的操作数访问型式由操作码字段和指令的数据总线传送字段确定。所有操作数访问型式可能不和所有指令一起用。

16.6.1　程序访问

程序访问（称作 P 访问）是访问 32 位宽的程序存贮器空间，并且通常是指令读。指令或数据操作数可用 MOVE 程序存贮器（MOVEM）、MOVE 外围数据（MOVEP）和 MOVE 绝对短（MOVES）指令从存贮空间读或写。程序访问可能是内部或外部存贮器访问，取决于地址和片子操作方式。

16.6.2　堆栈访问

堆栈访问（称作 S 访问）是访问独立的 64 位内部存贮空间（系统堆栈），它为子程序调用、中断和返回用于存贮 PC 和 SR 寄存器。在 PC 和 SR 寄存器之外，LA 和 LC 寄存器当程序循环开始时存在堆栈上。堆栈空间地址总是由指令所隐含。数据写入堆栈存贮空间以保存处理器状态，并从堆栈读出以恢复处理器状态。

16.6.3　寄存器访问

寄存器访问（称作 R 访问）是访问数据 ALU、地址产生单元和程序控制寄存器。数据可由一个寄存器读出而写入另一个寄存器。

16.6.4 存贮器访问

存贮器访问是访问32位宽的X和Y存贮器空间，并可以是内部或外部存贮器访问取决于指令的数据总线传送字段中操作数的有效地址。数据可从每个存贮器空间中任何地址读或写。

1. X存贮器访问

操作数在X存贮空间并且是字访问。数据可由存贮器读入寄存器或由寄存器读入存贮器。

2. Y存贮器访问

操作数在Y存贮空间并且是字访问。数据可由存贮器读入寄存器或由寄存器读入存贮器。

3. L存贮器访问

L存贮空间以一个操作数地址访问X和Y存贮空间。L存贮空间是由连结X和Y存贮空间而得到。数据操作数是长的字访问。操作数的高位字在X存贮器中，低位字在Y存贮器中。数据可在存贮器和连结的寄存器（如Dn.M：Dn.L）或双精度寄存器（如Dn.D）之间传送。

4. XY存贮器访问

XY存贮空间以二个操作地址访问X和Y存贮空间。一个字操作数在X存贮空间，而一个字操作数在Y存贮空间。

(1) 二个独立地址

二个独立地址用于访问二个字操作数。指令中二个有效地址用于驱动二个独立操作数地址——一个操作数地址可访问X存贮空间或Y存贮空间，而另一个操作数地址必须访问其他存贮空间。指令中指定的两个有效地址之一必须访问地址寄存器R0～R3之一，而另一个有效地址必须访问地址寄存器R4～R7之一。寻址方式被限为不更新和由+1、-1和+N后更新寻址方式。每个有效地址为其存贮空间提供独立的读/写控制，数据可从存贮器读入寄存器或相反。

(2) 一个共用地址

一个共用地址用于访问二个字操作数。指令中一个有效地址用于导出访问X和Y存贮空间的二个相同操作数地址。指令中指定的有效地址访问地址寄存器R0～R7之一。所有地址寄存器间接寻址方式都可以使用。有效地址为二个存贮空间提供共用读/写控制。数据可从存贮器读入寄存器或相反。

16.7 寻 址 方 式

DSP96002指令集包含操作数寻址方式的全部集合。所有地址计算在地址产生单元完成以减小执行时间和循环的额外开销。

寻址方式指定操作数是在寄存器还是存贮器中，并提供确定的操作数地址。指令中的有效地址将指定寻址方式，而对某些寻址方式，有效地址将进一步指定一地址寄存器。此外，地址寄存器间接方式要求附加的地址修正信息，它是不编入指令的。地址修正信息在被选地址修正寄存器中指定。所有存贮器访问要求一地址修正器而XY存贮器访问要求一或二个地址

修正器。一些指令的定义意味着指定寄存器的使用和所用寻址方式。

地址寄存器间接方式为用于地址计算要求一个偏置和一个修正寄存器。这些寄存器被指令字中指定在有效地址中的地址寄存器所隐含。每个偏置寄存器 Nn 和每个修正寄存器 Mn 指定给具有相同寄存器数 n 的寄存器 Rn。因此，被指定的寄存器是 M0；N0；R0、M1；N1；R1、M2；N2；R2、M3；N3；R3、M4；N4；R4，M5；N5；R5、M6；N6；R6、M7；N7；R7。地址寄存器 Rn 用作地址寄存器，偏置寄存器 Nn 用于指定选用的偏置以及修正寄存器 Mn 用于指定寻址方式修正器。

寻址方式分为三类：寄存器直接、地址寄存器间接和专用的。这些寻址方式叙述于下。

16.7.1 寄存器直接方式

这些有效寻址方式指定操作数是在 30 个数据 ALU 寄存器、10 个浮点寄存器、24 个地址寄存器或 7 个控制寄存器的一个（或多个）之中。

1. 数据或控制寄存器直接

操作数是在一、二或三个数据 ALU 寄存器中，作为指令中数据总线传送字段的一部分。寻址方式也用于为专用指令指定控制寄存器操作数。此访问属于寄存器访问。

2. 地址寄存器直接

操作数是在由指令中有效地址指定的 24 个地址寄存器中的一个之中。此访问属于寄存器访问。

16.7.2 地址寄存器间接方式

指令中的有效地址指定地址存器 Rn 和要完成的地址计算。这些寻址方式指定操作数在存贮器中，并提供操作数的指定地址。当地址寄存器用于指出存贮器单元时，此寻址方式称作地址寄存器间接。用词语间接是因为操作数不是地址寄存器本身，而是由地址寄存器指出存贮器单元的内容。指令中数据总线传送字段的一部分指定要完成的存贮器访问。所用地址运算型式由地址修正寄存器 Mn 指定。

1. 不更新 (Rn)

操作数地址在地址寄存器 Rn 中。Rn 寄存器中的内容不变。Mn 和 Nn 寄存器无关。此访问属于存贮器访问。

2. 后递增 1(Rn) +

操作数地址在地址寄存器 Rn 中。操作数地址使用以后，地址递增 1 并存入同一地址寄存器中。所用递增 Rn 型式的算法由 Mn 决定。Nn 寄存器无关。此访问属于存贮器访问。

3. 后递减 1 (Rn) −

除地址递减 1 以外，与上一条相同。

4. 后递增偏置 Nn(Rn) + Nn

操作数地址在地址寄存器 Rn 中。操作数地址使用后，地址递增 Nn 寄存器内容，并存入同一地址寄存器中。Nn 的内容作为 2 的补数处理，所以可解释为带符号或无符号的。Nn 寄存器的内容是不变的。用于递增 Rn 形式的算法取决于 Mn。此访问属于存贮器访问。

5. 后递减偏置 Nn(Rn) − Nn

除递减 Nn 的内容外，其他与上一条相同。

6. 由偏置 Nn 变址 (Rn+Nn)

操作数地址是地址寄存器 Rn 的内容和地址偏置寄存器 Nn 的内容之和。Nn 的内容按 2 的补数处理，所以可解释为带符号或无符号的。Rn 和 Nn 寄存器的内容不变。用于递增 Rn 型式的算法取决于 Mn。此访问属于存贮器访问。

7. 预递减 1 — (Rn)

操作数地址是地址寄存器递减 1 的内容。操作数地址使用以后，地址减 1 并存入同一地址寄存器。此访问属于存贮器访问。

8. 长位移 (Rn+Label)

此寻址方式要求指令扩展一个字 (Label)。操作数地址是地址寄存器 Rn 内容和扩展字之和。Rn 寄存器的内容是不变的。此访问属于存贮器访问。

16.7.3 PC 相对寻址方式

在 PC 相对寻址方式中，操作数地址是把用 2 的补数表示的位移加到程序计数器 (PC) 的值上而得到的。PC 总是指出下一条指令的地址，所以用 0 位移的 PC 相对寻址会产生后随指令的地址。

1. 长位移 PC 相对

此寻址方式要求指令扩展一个字。操作数地址是 PC 的内容和扩展字之和。

2. 短位移 PC 相对

短位移在指令操作字中占 15 位。位移首先带符号扩展到 32 位，然后加 PC 上以得到操作数的地址。

3. 地址寄存器 PC 相对

操作数的地址是地址寄存器和 PC 的内容之和。Mn 和 Nn 寄存器无关。

16.7.4 专用地址方式

专用地址方式在指定有效地址中不用地址寄存器。这些方式指定操作数或操作数地址在指令段中或它们隐含地访问一个操作数。

1. 直接数据

此寻址方式要求指令扩展一个字。直接数据是在指令的扩展字中的字操作数。此访问属于程序访问。

2. 直接短数据

8、16 或 19 位操作数在指令操作字中。8 位操作数用于 ANDI 和 ORI 指令，并且它是零扩展的。16 位操作数用于直接传送到寄存器，而它是带符号扩展的（解释为带符号整数）。19 位操作数用于 DO 和 REP 指令，而它是零扩展的。此访问属于程序访问。

3. 绝对地址

此寻址方式要求指令扩展一个字。操作数地址在扩展字中。此访问属于存贮器访问和程序访问。

4. 绝对短地址

对绝对短寻址方式，操作数地址在指令操作字中占有 7 位并且它是零扩展。此访问属于存贮器访问。

5. 短跳地址

操作数在指令操作字中占 15 位。地址带符号扩展到 32 位以使用相同的跳和相对分支格式。此访问属于程序访问。

6. I/O 短地址

对 I/O 短寻址方式操作数地址在指令操作字中占 7 位并且它是扩展的 1。I/O 短地址与位变换和传送外围数据指令一起用。

7. 隐式访问

某些指令对程序计数器（PC）、系统堆栈（SSH、SSL）、循环地址寄存器（LA）、循环计数器（LC）或状态寄存器（SR）作隐式访问。隐含的寄存器和它们的用途由各个指令说明定义。

16.7.5 寻址方式总结

图 16－7 包含前节讨论的寻址方式的总结。

寻址方式	修正器	操作数访问								
	MMM	P	S	C	D	A	X	Y	L	XY
寄存器直接										
数据和控制寄存器	不是			×	×					
地址寄存器	不是					×				
地址修正寄存器	不是					×				
地址偏置寄存器	不是					×				
地址寄存器间接										
无更新	不是	×					×	×	×	×
后增 1	是	×					×	×	×	×
后减 1	是	×					×	×	×	×
后增偏置 Nn	是	×					×	×	×	×
后减偏置 Nn	是	×					×	×	×	
由偏置值 Nn 变址	是	×					×	×	×	
前减 1	是	×					×	×	×	
长位移	是						×	×	×	
PC 相对										
长位移	不是	×								
短位移	不是	×								
地址寄存器	不是	×								
专用										
立即数据	不是	×								
绝对地址	不是	×					×	×	×	
绝对短地址	不是						×	×	×	
立即短地址	不是	×								
短跳地址	不是	×								
I/O 短地址	不是						×	×		
隐含	不是	×	×	×						

其中 MMM＝地址修正器　　A＝地址产生单元寄存器访问
P＝程序访问　　X＝X 存贮器访问
S＝堆栈访问　　Y＝Y 存贮器访问
C＝程序控制寄存器访问　　L＝L 存贮器访问
D＝数据 ALU 寄存器访问　　XY＝XY 存贮器访问

图 16－7　寻址方式总结

16.8 地址修正器型式

DSP96002 地址产生单元对所有地址寄存器间接方式支持线性、模数和倒位的地址运算。地址修正器决定用于更新地址的运算型式。地址修正器能够对 FIFO（排队）、迟延线、循环缓冲器、堆栈和倒位 FFT 缓存器在存贮器中产生数据结构。数据由更新地址寄存器（指针）变换，而不是由传送大数据块改变。地址修正寄存器 Mn 的内容定义为了寻址方式计算要完成的地址运算型式，并且对模数运算情况，Mn 的内容也指定模数。所有地址寄存器间接方式可与任何地址修正器型式一起使用。每个地址寄存器 Rn 有与它自己相关的修正寄存器 Mn。

16.8.1 线性修正器

用正常 32 位（模 4 294 967 296）线性算法（2 的补）完成地址修正。32 位偏置 Nn，或直接数据（+1、−1 或位移值）可用于地址计算中。值的范围可考虑带符号的（Nn 从 2 147 483 648 到+2 147 483 647）或无符号的（Nn 从 0 到+4 294 967 295）。在这两种数据表示之间没有运算差别。地址一般认为无符号，数据一般认为带符号。

16.8.2 逆进位修正器

地址修正以反方向传递进位来完成，即从 MSB 到 LSB 进位。这等效于倒位 Rn 的内容和偏置值 Nn，相加并且倒位其结果。若（Rn）+Nn 寻址方式与这地址修正器一起用，并且 Nn 包含值 2^{K-1}（2 的幂），则后递增 Nn 等效于倒位 Rn 的 K 个 LSB，Rn 递增 1，并倒位 Rn 的 K 个 LSB。此地址修正对 2^K 点 FFT 寻址是有用的。对 Nn 值的范围是 0 到+4 294 967 295。这允许对 FFT 倒位的寻址高达 8 589 934 592 点。

作为一个例子，考虑 1024 点 FFT，有实数存在 X 存贮器中而虚数存在 Y 存贮器中。Nn 包含数值 512 并后递增+N，会产生地址序列 0、512、256、768、128、640…。这是对序列频率点从 0 到 2 * Pi 的被扰乱的 FFT 数据次序。为了正常操作，逆进位修正器限制倒位数据缓存器的基地址为整数乘 2^K，如 1024、2048、3072 等。使用不是后递增 Nn 的寻址方法是可能的，但不能提供有用的结果。

16.8.3 模数修正器

模 M 完成地址修正，允许 M 的范围从 2 到+16 777 216。模 M 运算使地址寄存器的值保持在由低和高地址界所定义的大小 M 的地址范围内。M−1 值存在修正寄存器 Mn 中，因此允许模数的范围从 2 到 16 777 216。低界（基地址）值必须在 K 个 LSB 中有 0，这里 $2^K >= M$，所以必须是 2 的倍数。高界是低界加模数大小减 1（基地址+M−1）。

例如，为产生 24 级循环缓存器，M 选为 24 且低地址界必须有 5 个 LSB 等于 0（$2^K >= 24$，因此 K>=5）。Mn 寄存器装入数值 23（M−1）。低界可选为 0、32、64、96、128、160 等。缓存器的高界则是低界加 23。

地址指针不要求在低地址界开始而可在所定义的模地址范围内的任何地方开始。事实上，Rn 的单元决定低界和高界。在 DSP96002 上高和低界显然不是必需的，若地址寄存器指针增加通过缓存器高界（基地址加 M−1），则它将缠绕到基地址上。若递减通过低界（基地址），

则它将缠绕到基地址加 M－1。

若偏置 Nn 用在地址计算中，为了正常的模寻址。32 位数值 Nn 必需小于或等于 M，这是因为单模缠绕被检测。若 Nn 大于 M，其结果是数据，并且不可预测，除了特殊情况 Nn＝L＊（2^K），2^K 的倍数，这里 L 是正整数。注意偏置 Nn 必须是正的 2 的补的整数。对此情况指针 Rn 用线性运算将递增到存贮器中同样相对地址前面 L 块。类似地，对（Rn）－Nn 寻址方式，指针 Rn 用线性运算将递减到存贮器中后面 L 块。对正常情况，Nn 小于或等于 M，模运算单元将按要求的数量自动地缠绕地址指针。这种地址修正的型式对 FIFO（排队）、迟延线和高达 16 777 216 字长的采样缓存器产生循环缓存器是有用的。它对十分之一抽取、插值和波形产生也是有用的。（Rn）＋/－Nn 的特殊情况（有 Nn＝L＊（2^K））对在多缓存器上（例如实现并行滤波器组）完成相同算法是有用的。Nn 值的范围是－2 147 483 648～＋2 147 483 647。

16.8.4 多缠绕的模修正器

模 M 完成地址修正，M 可以是任何 2 的幂，范围为 2^1～2^{23}。模 M 运算使地址寄存器值保持在由低和高地址界所定义的大小为 M 的地址范围内。值 M－1 存在修正寄存器 Mn 中的最低有效 24 位，而 8 个最高有效位置于 $FF。低界值（基地址）在 K 个 LSB 中必须有 0，这里 2^K＝M，所以必须是 2^K 的倍数。高界是低界加模数减 1（基地址＋M－1）。

例如，为产生 32 级循环缓存器，M 选为 32，而低地址界必须有 5 个 LSB 等于 0（2^K＝32，因此 K＝5）。Mn 寄存器装入数值 $FF00001F。低界可选为 0、32、64、96、128、160 等。缓存器高界则是低界加 31。

地址指针不要求在低地址界开始，而可在模地址范围定义之内的任何地方开始。若地址寄存器指针递增通过缓存器高界，它将缠绕向基地址。若地址递减通过低界，它将缠绕向基地址加 M－1。若偏置 Nn 用于地址计算中，为了正确的模寻址，32 位值 Nn 不要求小于或等于 M，因为对（Rn）＋Nn、(Rn)－Nn 和(Rn＋Nn)，地址更新支持多次缠绕。Nn 值的范围是－2 147 483 648～＋2 147 483 647。

地址修正的型式对抽取、插值和波形产生是有用的，因为多缠绕的能力可用于减少变量。

16.8.5 地址修正器型式总结

图 16－8 包含前节讨论的地址修正器型式的总结。

修正器								
M	M	M	M	M	M	M	M	地址计算操作
0	0	0	0	0	0	0	0	反进位　　(位反转更新)
0	0	0	0	0	0	0	1	模　2
0	0	0	0	0	0	0	2	模　3
0	0	F	F	F	F	F	E	模　16 777 215　　((2**24)-1)
0	0	F	F	F	F	F	F	模　16 777 216　　(2**24)
0	1	×	×	×	×	×	×	保留
0	2	×	×	×	×	×	×	保留
F	D	×	×	×	×	×	×	保留
F	E	×	×	×	×	×	×	保留
F	F	0	0	0	0	0	0	保留
F	F	0	0	0	0	0	1	多缠绕模 2
F	F	0	0	0	0	0	3	多缠绕模 4
F	F	0	0	0	0	0	7	多缠绕模 8
F	F	3	F	F	F	F	F	多缠绕模 2**22
F	F	7	F	F	F	F	F	多缠绕模 2**23
F	F	F	F	F	F	F	F	线性　(模 2**32)
其中　MMMMMMMM=修正器寄存器内容 (16 进制)								

图 16-8　地址修正器总结

第十七章　指令集和执行

17.1　引　　言

本章介绍 DSP96002 指令集和指令格式。指令功能与第十六章所述的灵活寻址方式一起对数字信号处理和图形算法提供了很强功能的汇编语言。指令集已设计得能对高级语言编译器进行有效编码，并也可用汇编语言很容易地编程。

如第十五章编程模型所指出，DSP96002 结构可看成三个并行操作的执行单元（数据 ALU、地址产生单元和程序控制器）。指令集的目的是保持在指令周期时这些单元的每一个都是忙。从而达到程序存贮器最大的吞吐量和最少的使用。

17.2　指　令　组

指令集分成以下的组：

- 浮点运算　　(38)
- 定点运算　　(30)
- 逻辑　　(13)
- 位变换　　(4)
- 循环　　(4)
- 传送　　(9)
- 程序控制　　(35)

每个指令组在以下各节中叙述。

17.2.1　浮点运算指令

全部浮点运算指令在 96 位数据 ALU 寄存器上操作。浮点运算指令是基于寄存器的（对操作数是寄存器直接寻址方式）并在数据 ALU 以内执行。这意味着 X 数据总线、Y 数据总线和全局数据总线选用并行传送操作是随意的。这允许新数据可在后面的指令中预取，且前面指令计算的结果可存贮。浮点指令总是在 Flush - to - Zero 方式中单指令周期内执行。若去归一化数未被检测到，则在 IEEE 方式中浮点指令在单指令周期中执行，否则要求附加的指令周期。图 17 - 1 是 38 种浮点运算指令。

17.2.2　定点运算指令

定点运算指令完成所有数据 ALU 中的运算。运算指令是基于寄存器的（对操作数用寄存器直接寻址方式）所以由指令指出的数据 ALU 操作不用 X 数据总线、Y 数据总线或全局数据总线。当大多数数据 ALU 操作时，可供在这些总线之间作并行数据传送。这使新数据能在随

后的指令中预取且前面指令所计算的结果可以存贮。定点运算指令在一个指令周期中执行。图17－2是30条定点运算指令。

FABS. S	绝对值(单精度)	FMPY FADD. S	乘和加(单精度)
FABS. X	绝对值(单扩展精度)	FMPY FADD. X	乘和加(单扩展精度)
FADD. S	相加(单精度)	FMPY FADDSUB. S	乘、加和减(单精度)
FADD. X	相加(单扩展精度)	FMPY FADDSUB. X	乘、加和减(单扩展精度)
FADDSUB. S	加和减(单精度)	FMPY FSUB. S	乘和减(单精度)
FADDSUB. X	加和减(单扩展精度)	FMPY FSUB. X	乘和减(单扩展精度)
FCLR	清除浮点操作数	FMPY. S	乘(单精度)
FCMP	比较	FMPY. X	乘(单扩展精度)
FCMPG	图形与 Trivial Accept/Reject Flags 比较	FNEG. S	变符号(单精度)
FCMPM	比较数值	FNEG. X	变符号(单扩展精度)
FCOPYS. S	拷贝符号(单精度)	FSCALE. S	定标浮点操作数(单精度)
FCOPYS. X	拷贝符号(单扩展精度)	FSCALE. X	定标浮点操作数(单扩展精度)
FGETMAN	得尾数	FSEEDD	倒数近似
FINT	转换到浮点整数	FSEEDR	平方根倒数近似
FLOAT. S	整数到 SP 浮点转换	FSUB. S	减(单精度)
FLOAT. X	整数到 SEP 浮点转换	FSUB. X	减(单扩展精度)
FLOATU. S	无符号整数到 SP 浮点转换	FTFR. S	传送浮点寄存器(单精度)
FLOATU. X	无符号整数到 SEP 浮点转换	FTFR. X	传送浮点寄存器(单扩展精度)
FLOOR	变换到浮点整数向无穷大舍入	FTST	测试浮点操作数

图 17－1　浮点运算指令

ABS	绝对值	INTU	浮点到无符号整数转换
ADD	相加	INTURZ	浮点到无符号整数转换向 0 舍入
ADDC	进位加	JOIN	联合二个 16 位整数
ASL	算术左移	JOINB	联合二个 8 位整数
ASR	算术右移	MPYS	带符号乘
CLR	清除一操作数	MPYU	无符号乘
CMPG	比较图形和 Trivial Accept/Rejext Flays	NEG	取负
CMP	比较	NEGC	带进位取负
DEC	递减 1	SETW	设置操作数
EXT	符号扩展 16 位到 32 位	SPLIT	提取 16 位整数
EXTB	符号扩展 8 位到 32 位	SPLITB	提取 8 位整数
GETEXP	变为指数	SUB	相减
INC	递增 1	SUBC	带进位减
INT	浮点到整数转换	TFR	传送数据 ALU 寄存器
INTRZ	浮点到整数转换向 0 舍入	TST	测试操作数

图 17－2　定点运算指令

17.2.3　逻辑指令

逻辑指令在数据 ALU 以内除 ANDI 和 ORI 外完成所有逻辑操作。逻辑指令像前面所讨论的运算指令一样是基于寄存器的。选择的数据传送与大多数逻辑指令并行指定——通过 X 和 Y 数据总线或全局数据总线。这使新数据可以在随后的指令中预取，且前面指令计算的结果可以存贮。这些指令在一个指令周期中执行。图 17－3 是 13 条逻辑指令表。

AND	逻辑与	NOT	逻辑补
ANDC	逻辑与带补数	OR	逻辑异或
ANDI	AND 直接到控制寄存器*	ORC	逻辑异或(带补数)
BFIND	找前导 1	ORI	OR 直接到控制寄存器*
EOR	逻辑异或	ROL	左转
LSL	逻辑左移	ROR	右转
LSR	逻辑右移		
* 这些指令不允许并行数据传送。			

图 17-3 逻辑指令

17.2.4 位变换指令

位变换指令在数据存贮器或寄存器中测试任何一个位的状态,然后有选择地设置、清除或倒置此位。在 CCR 寄存器中进位位将包含位测试结果。不允许和这些指令的任一条并行传送。图 17-4 是 4 条位变换指令表。

BCLR	位测试和清除	BCHG	位测试和改变
BSET	位测试和置定	BTST	位测试

图 17-4 位变换指令

17.2.5 循环指令

循环指令由起动程序循环和设置循环参数,或当终止一循环时"清除"系统堆栈来控制硬件循环。初始化包括保留在系统堆栈上程序循环(LA 和 LC)所用的寄存器,因此程序循环可以嵌套。程序循环中第一条指令的地址也被保存使之无额外的循环。图 17-5 是 4 条循环指令表。

DO	开始硬件循环	ENDDO	从硬件循环退出
DOR	开始 PC 相对硬件循环	REP	重复下一条指令

图 17-5 循环指令

17.2.6 传送指令

传送指令通过 X 和 Y 数据总线、全局数据总线和程序数据总线完成数据传送。地址产生单元指令也包括在下面传送指令中。图 17-6 是 9 条传送指令表。

LEA	装入有效地址	MOVEI	传送立即数
LRA	装入 PC 相对地址	MOVEM	传送程序存贮器
MOVE	传送数据寄存器	MOVEP	传送外围数据
MOVETA	传送数据寄存器和测试地址	MOVES	传送绝对短数据
MOVEC	传送控制寄存器		

图 17-6 传送指令

17.2.7 程序控制指令

程序控制指令包括跳、条件跳、分支、条件分支和其他影响 PC 和系统堆栈的指令。分支指令允许放置独立码需要的 PC 相对位移。图 17-7 是 35 条程序控制指令表。

Bcc	条件分支	ILLEGAL	合法指令中断
BRA	常分支	Jcc	条件跳
BRCLR	若位清除则分支	JCLR	若位清除则跳
BRSET	若位设置则分支	JMP	跳
BScc	条件分支到子程序	JScc	条件跳到子程序
BSCLR	若位清除分支到子程序	JSCLR	若位清除则跳到子程序
BSR	分支到子程序	JSET	若位设置则跳
BSSET	若位设置则分支到子程序	JSR	跳到子程序
DEBUG	进入调试方式	JSSET	若位设置则跳到子程序
FBcc	条件分支	NOP	无操作
FBScc	条件分支到子程序(浮点情况)	RESET	复位外围设备
FFcc	条件的数据 ALU 操作没有 CCR 更新	RTI	从中断返回
FFcc. U	条件的数据 ALU 操作有 CCR 更新	RTR	从子程序返回和恢复状态寄存器
FJcc	条件跳	RTS	从子程序返回
FJScc	条件跳到子程序	STOP	停止处理(低功率等待)
FTRAPcc	条件软件中断	TRAPcc	条件的软件中断
IFcc	条件的数据 ALU 操作没有 CCR 更新	WAIT	等待中断(低功率等待)
IFcc. U	条件的数据 ALU 操作有 CCR 更新		

图 16-7　程序控制指令

17.3　指 令 格 式

因为多数据总线结构以及 DSP96002 的并行性，指令字中可指定高达 3 个数据传送——一个在 X 数据总线上，一个在 Y 数据总线上以及一个在数据 ALU 内。第 4 个数据传送通常是隐含的，并发生在程序控制器中(指令字提取、程序循环控制等)。每个数据传送包含一个源和一个目的。

在一个指令字中，可指定一个或多个“有效地址”。有效地址定义导出操作数单元的途径。有效地址包含寻址方式还可能有所选的寄存器。寻址方式选择要用的地址更新。指定的寄存器可以是操作数的单元或是用于计算操作数地址的地址寄存器。有一些指令隐含指定寄存器的用途，而对这些寄存器不指定有效地址。

DSP96002 指令由 1 或 2 个 32 位字组成——操作字和选用的有效地址扩展字。指令及其长度由指令的第一个字指定。操作字的一般格式示于图 17-8。

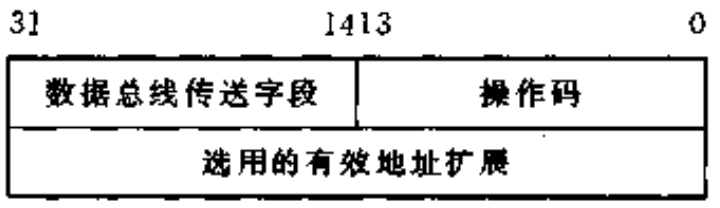


图 17-8　指令字一般格式

大多数指令指定数据在 X 和 Y 数据总线传送和在同一操作字中数据的 ALU 操作。DSP96002 设计得能并行完成每个这些操作。数据总线传送段提供操作数访问型式、传送方向和在 X、Y 数据总线上数据传送的有效地址。操作数访问型式选择存贮器或寄存器要进行的访问型式。数据总线传送段为对某些寻址方式完全指定操作数要求附加的信息。跟随操作字的有效地址扩展字根据要求用于提供立即数据、绝对地址或位移。

操作字的操作码段指定要完成的数据 ALU 操作或程序控制器操作以及任何指令所要求的附加操作数。仅那些能够伴随总线传送的数据 ALU 和程序控制器操作将被指定在指令的操作码字段中。其他数据 ALU 和程序控制器操作和所有地址产生单元操作将在不同格式的指令字中指定。这些包括有短立即数据或短绝对地址的操作字。

典型的一字指令的汇编语言源码示于下面。源码由高达 6 段组成。

（相乘器）		（相加器/相减器）			
操作码	操作数	操作码	操作数	X 总线数据	Y 总线数据
FMPY	D0,D5,D2	FSUB.S	D7,D3	X:(R0)+,D0.S	Y:(R4)+,D5.S

第一个操作码表示数据 ALU、地址产生单元、位变换单元或要完成的程序控制器操作。第一操作数段指定在第一操作码段中指定的操作码要用的操作数。

第二操作码段表示当要求浮点相加器/相减器和相乘器并行操作时，数据 ALU 中浮点相加器/相减器运算。第二操作数段指定相加器/相减器操作码要用的操作数。操作码段之一通常需包括在源码中。

X 总线数据段指定通过 X 总线选用的数据传送和要用的寻址方式。Y 总线数据段指定通过 Y 总线选用的数据传送和要用的寻址方式。地址空间修饰词 X:,Y:和 L:表示哪个地址空间正被访问。

DSP96002 提供数据 ALU、地址产生单元和程序控制器的并行处理。对上述指令字，DSP96002 将完成指定的浮点相乘器运算（数据 ALU），指定的浮点相加器/相减器运算（数据 ALU），以地址寄存器更新（地址产生单元）指定的数据传送和译码下一指令以及从程序存贮器（程序控制器）提取指令都在一个指令周期内。长指令大于一个字长时，要求附加的指令执行周期。

包含数据 ALU 的大多数指令是基于寄存器的（所有操作数在数据 ALU 寄存器中）并允许程序员保持每个并行处理单元为忙。定向存贮器（如位变换指令）或使控制流改变（如跳）的指令，当它执行时防止使用并行处理资源。

17.4 指 令 执 行

指令执行是流水线式，允许大多数指令以每个指令周期一条指令的速率执行。但是某些指令要求附加时间去执行。这些包括长于一个字的指令、用一种寻址方式要求大于一个周期的指令、使用全局数据总线多于一次的指令和使控制流改变的指令。

17.4.1 指令处理

流水线允许当其他指令作取数-译码-执行操作时，进行另一指令的取数-译码-执行操作。当一条指令正在执行时，对要执行的下一条指令译码，同时跟随正被译码指令的指令从程存贮器取数。若指令是 2 个字长，在提取下一条指令之前附加字将被提取。图 17-9 表示流水线。F1、D1 和 E1 分别是第一条指令的提取、译码并执行操作。第三条指令包含一指令扩展字和用 2 周去执行。

	F1	F2	F3	F3e	F4	F5	F6 . . .
		D1	D2	D3	D3e	D4	D5 . . .
			E1	E2	E3	E3e	E4 . . .
指令周期	1	2	3	4	5	6	7 . . .

图 17-9 指令流水线

每条指令要求最小 12 个时钟段去提取、译码和执行。4 个段以后新的指令可以开始。双字指令要求最少 16 段去执行，而 8 个段以后新指令可以开始。

17.4.2 存贮访问处理

当指令执行时，可访问一个或多个 DSP96002 存贮器源(X 数据存贮器、Y 数据存贮器和程序存贮器)。每个这样的存贮器源可在 DSP96002 的内部或外部。3 条地址总线(XAB、YAB 和 PAB)和 4 条数据总线(XDB、YDB、PDB 和 GDB)在一个指令周期内都可供内部存贮器核心(不同于 DMA)访问使用。

DSP96002 有二个外部扩展端口(A 和 B)，用作内部地址的扩展和作为外部存贮器访问的数据总线。若所有存贮器源在 DSP96002 内部，则三个存贮器源之一或一个以上可在一个指令周期中访问(即程序存贮器访问或程序存贮器加 X、Y、XY 或 L 存贮器访问)。但是，当一个或更多的存贮器在 DSP96002 外部，并且外部存贮器位于同一扩展端口时，存贮器访问可能要求附加的指令周期。

若在一个指令周期中，在同一端口要求一个以上的外部访问，则访问将按以下优先权排序：

(1) X 存贮器。

(2) Y 存贮器。

(3) 程序存贮器。

(4) DMA。

第十八章　扩展端口和I/O外围

18.1　引　　言

X和Y数据存贮器的高128单元定义为I/O空间。Y存贮器I/O空间全是外部的，而X存贮器I/O空间是内部的。X存贮器I/O空间用来访问I/O接口寄存器以及总线、端口选择和中断控制寄存器。两种I/O空间都可由X和Y存贮器MOVE指令访问。MOVEP指令提供I/O短寻址以及用I/O映射的寄存器为数据传送的存贮器到存贮器的传送能力。

设置片上I/O外围是要在很多应用中减少系统的片数和“粘结”逻辑。每个I/O接口有它自己的映射到X存贮器I/O空间的控制、状态和数据寄存器。每个接口有几个专用中断矢量地址和控制位以允许/不允许中断。因为每个中断源有它自己的服务程序，这减少了与服务此设备有关的额外开销。

在DSP96002中提供三种片上外围：

- 连接到端口A的32位并行主机MPU/DMA接口。
- 连接到端口B的32位并行主机MPU/DMA接口。
- 二通道DMA控制器。

18.2　扩展端口控制

DSP96002有二个外部扩展端口(A和B)。每个端口有一总线控制寄存器，其中可指定存贮器等待状态，为专用页面电路对DRAM/VRAM存贮器支持配置参数和控制位，以及得到为$\overline{BR}$和$\overline{BL}$的直接软件控制的控制位。

18.2.1　总线控制寄存器（BCRA和BCRB）

有二个相同的BCR寄存器，每个端口一个。在外部存贮器访问时可编程总线控制寄存器(BCRx)以在一个总线周期内插入等待状态。它们也可用于编程页面出错电路以及$\overline{BR}$和$\overline{BL}$端的直接软件控制。

1. BCRx等待控制段（位0～15）

BCRx等待控制段为外部X存贮器、Y存贮器、程序存贮器或I/O的访问在总线周期内指定要插入的等待状态数。对每一种外部存贮器访问型式在控制寄存器中有4位可供使用。每4位字段可确定高达15个等待状态。当硬件复位时，设置等待控制字段为‘$F’(15个等待状态)。参见第十三章中由BCR决定的等待状态和由于$\overline{TA}$端产生的等待状态之间的相互作用的叙述。软件复位和页面电路本身复位都不影响BCRx。

端口 A 总线控制寄存器 (BCRA) X: $FFFFFFFE

端口 B 总线控制寄存器 (BCRB) X_1 = $FFFFFFFD

**保留用，读作 0，为了将来兼容应写 0。

图 18-1 DSP96002 总线控制寄存器 (BCRA 和 BCRB)

2. BCRx 页面大小 (P3～P0) 位 16～19

这些位对页面故障操作定义页面大小。P3～P0 由硬件复位设置为 '1010'。

P3～P0	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010
页面大小	1	2	4	8	16	32	64	128	256	512	1 024(复位值)
P3～P0	1011	1100	1101	1110	1111						
页面大小	2 048	4 096	8 192	16 384	32 768						

3. BCRx 保留位 (20, 21)

这些保留位读作 0 并用 0 写入为将来兼容。

4. BCRx 非顺序故障使能 (NS) 位 22

若 NS 控制位被设置，则允许非顺序的故障检测。若 NS 控制位被清除，则非顺序故障被页面电路忽略。参看 18.2.2 节页面电路原理。此位由硬件复位来清除。

5. BCRx 总线控制权故障使能 (MF) 位 23

若设置了 MF 控制位，则允许总线控制权故障检测。若清除 MF 控制位，则总线控制权故障被页面电路忽略。此位由硬件复位来清除。

6. BCRx 存贮器空间故障使能 (SF1～SF0) 位 24～25

基于 S1 和/或 S0 改变的存贮器空间故障分别由 SF1 和 SF0 控制。若设置了 SF1 (SF0)，则 S1 (S0) 的改变将使一存贮器空间出错。若 SF1 (SF0) 被清除，则 S1 (S0) 的改变被页面电路忽略。SF1 和 SF0 由硬件复位清除。

7. BCRx 程序存贮器故障使能 (PE) 位 26

若设置程序存贮器故障使能位，则页面故障电路将监视程序存贮器总线周期。若设置 PE，而在程序存贮器总线周期内检测到故障，$\overline{TT}$将不确定。若设置 PE 而在程序存贮器总线周期

时没有检测到故障，则$\overline{TT}$将被确定。若清除 PE，则页面故障电路对程序存贮器总线周期将是无效的，而$\overline{TT}$保持不确定。PE 由硬件复位清除。

PE	$\overline{TT}$端对 P 空间作用
0	不确定
1	有效

8. BCRx Y 数据存贮器故障使能（YE）位 27

若 YE 位被设置，则页面故障电路将监视 Y 数据存贮器总线周期。若设置 YE，而在 Y 数据存贮器总线周期检测到一个故障，则$\overline{TT}$将不被确定。若设置 YE，而在 Y 数据存贮器总线周期未检测到故障，则$\overline{TT}$将被确定。若 YE 清除，则页面电路对 Y 数据存贮器总线周期无效。并且$\overline{TT}$将保持不确定。YE 由硬件复位清除。

YE	$\overline{TT}$端对 Y 空间的作用
0	不确定
1	有效

9. BCRx X 数据存贮器故障使能（XE）位 28

若 XE 被设置，则页面故障电路将监视 X 数据存贮器总线周期。若在 X 数据存贮器总线周期 XE 被设置并检测到故障，则$\overline{TT}$将不确定。若在此周期内 XE 被设置而未检测到故障，则$\overline{TT}$将被确定。若 XE 被清除，则页面电路对 X 数据存贮器总线周期无效，且$\overline{TT}$将保持不确定 XE 由硬件复位清除。

XE	$\overline{TT}$端对 X 空间的作用
0	不确定
1	有效

10. BCRx 总线状态（BS）位 29

若 DSP96002 当前是总线控制，则设置只读总线状态的状态位。若 DSP96002 不是总线控制，则 BS 被清除。此位由硬件复位清除。

11. BCRx 总线锁定控制（LH）位 30

若设置 LH 位，则即使未发生读-修改-写的访问，$\overline{BL}$端也是确定的。若 LH 被清除，则$\overline{BL}$只在读-修改-写的外部访问时确定。此位由硬件复位来清除。

12. BCRx 总线请求保持控制（RH）位 31

若设置 RH 位，则即使 CPU 或 DMA 不需要总线，$\overline{BR}$端也是确定的。若 RH 被清除，则$\overline{BR}$端只在外部访问正在试探或未决定时才被确定。此位由硬件复位清除。

18.2.2 页面电路原理

页面电路的目的是使设计者能以低价的动态 RAM 存贮器系统达到静态 RAM 性能。，其内部页面检测电路，DSP96002 用在 DRAM/VRAM 设备上提供的快速访问方式可达到零

等待状态的性能。不用内部页面检测电路，零等待状态的性能不可能得到。典型存贮器是：

设备	大小	方式
MCM514256A	256K×4	页面
MCM51L1000A	1M×1	页面
MCM514258A	256K×4	静止列
MCM511002A	1M×1	静止列

当总线控制时，在 CPU 或 DMA 用 P、X 或 Y 存贮空间（S1：S0=10，01 或 00）访问外部总线时，页面电路有效，页面电路用传送型（$\overline{TT}$）输出端指出外总线访问型式。当外存贮器在当前总线周期情况下可用快速访问方式（页面、静止列、半字节或串行位移）时，页面电路确定传送型式（$\overline{TT}$）端。页面电路必须以允许快速访问方式的外存贮器特性编程。若外存贮器在当前总线周期中不能用快访问方式时，$\overline{TT}$保持不确定。

页面电路有选择地比较基于总线控制器中由用户编程的存贮器参数的当前总线周期 C 和以前锁住的总线周期 C′的属性，包括地址、存贮空间选择和总线控制权。注意以前锁住的总线周期 C′不能立即优先于当前总线周期，这取决于存贮空间映射。当前和以前总线周期的属性定义在图 18－2 中，而页面电路编程参数定义在图 18－3 中。这些参数（或功能等效）是在总线控制寄存器中用户可编程的。硬件、软件或页面电路本身复位将使页面电路复位。

C	C′	总线访问属性
A	A′	地址（A0～A?1）
S	S′	空间选择 S0～S1
M	M′	总线控制权$\overline{BA}$

图 18－2　总线访问属性

名称	存贮器参数	随机端口（D/VRAM）	串行端口（VRAM）
P3～P0	Log2（页面大小）	行数（若半字节方式为 4）	串行寄存器大小
NS	非顺序故障	若半字节方式则是	是
MF	总线控制权故障	取决于系统	取决于系统
SF1	存贮器空间故障 1	取决于系统	取决于系统
SF0	存贮器空间故障 0	取决于系统	取决于系统
PE	P 空间允许	取决于系统	取决于系统
XE	X 空间允许	取决于系统	取决于系统
YE	Y 空间允许	取决于系统	取决于系统

图 18－3　页面电路编程参数

一旦存贮器参数在页面电路编程，$\overline{TT}$端将提供关于当前外总线周期的信息，它是基于以前外总线周期页面电路中锁住的信息。页面电路能检测以下故障：

页面故障：若当前地址 A 不与锁住的地址 A′在同一存贮器页面中，则$\overline{TT}$是不确定的。DRAM 或 VRAM 的随机访问端口的页面大小典型地是行数。页面大小参数 P 等于在行地址选通脉冲确定时，锁入存贮器的行地址线的数目。对页面或静止列方式 RAM，典型页面大小

是 256、1024 等。对半字节方式 RAM，页面大小是 4。

非顺序故障：若当前地址 A 不是锁住的地址 A′加 1（+1），则$\overline{TT}$不确定。若 NS 控制位被置定，则允许非顺序故障，否则不允许。半字节方式访问随机端口，或串行访问串行端，能产生非顺序故障。页面和静止列方式 RAM 不可能有非顺序故障。并且 NS 应清除。页面电路定义页面内的地址校验非顺序故障。

总线控制权故障：若当前总线周期自变成总线控制，是第一个外总线周期，则$\overline{TT}$是不确定的。由任何总线控制的第一个外总线周期通常不是快速访问方式，因为其他总线控制可能访问同一外存贮器。这也保证硬件复位以后第一个外总线周期不确定$\overline{TT}$。若 MF 控制位被设置，则允许总线控制权故障，否则不允许。若外存贮器分配到特定处理器，则某些多处理器系统希望不出现此特性是可能的。

存贮空间（实际的存贮器）故障：若当前总线周期访问与以前锁住的总线周期不同的存贮空间，则$\overline{TT}$不确定。若空间选择端 S1 或 S0 用作到外存贮器的地址线，这是有用的。此时，用户映射不同存贮空间的同一地址到不同的实际存贮器单元。若空间选择端 S1 和 S0 不用作到外存贮器的地址线，则用户映射不同存贮空间的同一地址到同一实际存贮器单元，所以存贮空间的改变应忽略。这是“单存贮空间”在系统中执行高级语言（如 C 语言）的智力优势的一例。

基于 S1 和/或 S0 改变的存贮空间故障分别由 SF1 和 SF0 控制位所控制。若 SF1(SF0)被设置，则 S1（S0）的改变将产生存贮空间故障，并不确定$\overline{TT}$。若 SF1（SF0）被清除，则 S1（S0）的改变被忽略。对每个 SF1 和 SF0 合成的用户存贮器映射和存贮空间变化检测在图 18-4（a）中给出。

注意当前总线周期 C 和以前锁定总线周期 C′都表示访问三个存贮空间之一。S1：S0＝11 一起从不出现当前或锁定的存贮空间值，因为它意味着没有正在进行的访问（S：S0＝00→Y、S1；S0＝01→X、S1：S0＝10→P）。

SF1	SF0	存贮空间映射到同一实际地址	存贮空间变化按故障检测
0	0	PXY 分享相同地址	否
0	1	PY 分享相同地址	P→X，X→P，X→Y，Y→X
1	0	XY 分享相同地址	P→X，X→P，P→Y，Y→P
1	1	否，所有地址是唯一的	P→X，X→P，X→Y，Y→X，P→Y，Y→P

图 18-4（a）　存贮空间变化检测

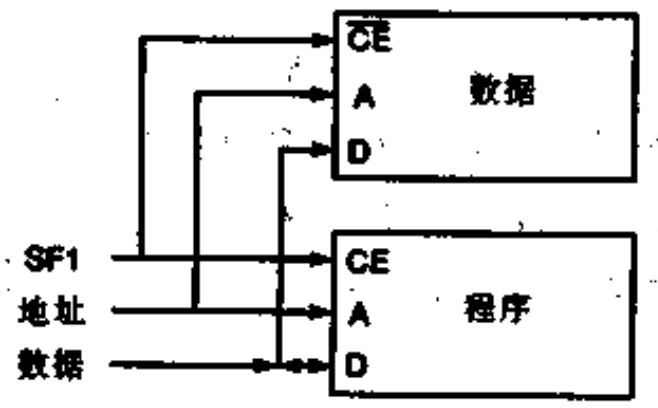


图 18-4（b）　SF1 用于分立的数据和程序空间

有一个组合（PX）在编码中没有，这里 P 和 X 享有相同的地址。因为这个组合不能直接用 S1 或 S0 作为地址线，它的使用不会普遍，并且它的实现要控制“每个空间”基础而不是如上所示的“每个引出端”基础。

这种讨论假设若 S1 和/或 S0 用作地址线，则它们作为页面大小边界之上的高阶地址线引入。若 S1 和/或 S0 作为低于页面大小边界的低阶地址引入，则以调整页面大小可达到适当的页面故障操作，但不能用非顺序故障检测。所以推荐仅用 S1 和 S0 作为页面大小边界之上的高阶地址线。具有 SF1：SF0＝10 的实例系统在程序和数据空间之间检测位移示于图 18－4(b)。

1. 存贮空间使能和页面故障电路专用复位

若当前总线周期在所选存贮空间的用户中，则页面故障电路使能。提供分立的存贮空间使能控制位（PE、XE 和 YE），使用户可选择页面故障电路监视的存贮空间。若设置存贮空间使能位（PE、XE 和/或 YE），如当前总线周期在那个存贮空间，则页面故障电路有效。若存贮空间使能位被清除，则对总线周期页面电路无效，并且$\overline{TT}$仍然不确定。若所有三个存贮空间使能位被设置，则对所有外总线周期页面电路有效。

若当前总线周期是使能存贮空间，则$\overline{TT}$端由对当前总线周期和以前锁住的总线周期的比较来控制，并且当前总线周期信息（A，S）被锁在总线周期末尾。因此当前总线周期信息变成以前被锁的总线周期信息，供在下一个使能外总线周期比较。存贮空间使能的编码示于图 18－5。

$\overline{TT}$端作用						被锁住的当前总线周期		
PE	XE	YE	P 空间	X 空间	Y 空间	P 空间	X 空间	Y 空间
0	0	0	不确定	不确定	不确定	否	否	否
0	0	1	不确定	不确定	有　效	否	否	是
0	1	0	不确定	有效	不确定	否	是	否
0	1	1	不确定	有效	有效	否	是	是
1	0	0	有效	不确定	不确定	是	否	否
1	0	1	有效	不确定	有效	是	否	是
1	1	0	有效	有效	不确定	是	是	否
1	1	1	有效	有效	有效	是	是	是

图 18－5　存贮空间使能编码

页面电路一般监视一个外部实际存贮器设计的地址。但是，若多存贮空间以相同或不同的地址映射到一个实际存贮器，则页面电路必须监视多存贮空间。这些存贮空间使能位允许用户指示哪个存贮空间应当被监视。若多存贮空间也映射到不同的实际存贮器，则一个页面电路可用多于一个的存贮空间服务于多个外部实际存贮器。以多个外部实际存贮器非交错访问是典型的系统，其中主要的外总线作用是有向阻塞的 DMA 传送。

若三个存贮空间使能位都被清除，则页面电路处于专用复位状态。当在专用复位状态时，页面电路无效，对所有外总线周期$\overline{TT}$仍然不确定，并且没有总线周期信息被锁住。重新使能页面电路后的第一个总线周期，$\overline{TT}$总是不确定的，因为没有前面的总线周期信息可用来与之比较。

2. 刷新故障

对刷新计时器、刷新地址计数器或刷新故障没有内部支持。页面电路假设刷新不存在，所

以$\overline{TT}$必须由外存贮器控制器来解释，并基于刷新定时和外总线作用的知识。使用具有相同的外部 DRAM/VRAM 的多处理器指出，存贮器控制器是强制刷新优先权的最好地方。随着刷新技术的多样化，外存贮器控制器状态机对通过刷新定时的全局控制和由多个访问冲突引起的仲裁是最好的地方。在每个外总线周期的末尾，外存贮器控制器应决定是否应开始刷新周期。若刷新，则将使传送确认$\overline{TA}$信号不能保证 DSP96002 等待，若开始一次外部访问。一旦刷新完成，外存贮器控制器必须记住对下一次存贮器周期忽略$\overline{TT}$信号，使之不用快速访问方式。外部状态机应在任何硬件刷新以后的下一个外总线周期内取消$\overline{TT}$信号的作用。注意若用快中断实现软件刷新，则刷新看起来像存贮器读周期，因此不需要对$\overline{TT}$进行特殊处理。

3. $\overline{RAS}$、$\overline{CAS}$和 SC 超时故障

因为 DRAM/VRAM 设备是动态的，故对$\overline{RAS}$和$\overline{CAS}$的低时有最大限制。为了有效地使用 DSP96002 快访问方式，外部状态机必须在页面、半字节和静态列方式的总线周期之间保持$\overline{RAS}$确定。$\overline{CAS}$必须仅在静态列方式的总线周期之间保持确定。但是，若对块访问方式外部状态机准备好以后没有外部访问发生，则有可能$\overline{RAS}$或$\overline{CAS}$“超时”。这是因为为了以 DSP96002 无突发、随机的地址总线周期使用快访问方式，空闲存贮器状态必须是“$\overline{RAS}$有效”。DSP96002 对$\overline{RAS}$或$\overline{CAS}$超时不提供任何内部支持。为了保证不发生$\overline{RAS}$或$\overline{CAS}$超时，外部状态机是有责任的。因为典型的$\overline{RAS}$和$\overline{CAS}$超时是 10～100μs，最简单的解决方法之一是完成确定$\overline{RAS}$和$\overline{CAS}$的硬件刷新。若能经常完成刷新，则$\overline{RAS}$和$\overline{CAS}$超时将不会发生。

VRAM 设备的串行端口由串行时钟 SC 定时。因串行移位寄存器是动态的，所以一个移位寄存器必须有按时刷新其内容的最低频率。该频率典型值是 20kHz（50μs 刷新周期）。DSP96002 不为 SC 超时提供任何内部支持。外部状态机有责任保证不发生 SC 超时。若发生 SC 超时，外部状态机取消下个外总线周期中$\overline{TT}$信号的影响，以强制串行移位寄存器的再装入。很幸运，未来 1Mb VRAM 正以静态移位寄存器被指定，故 SC 超时问题将消失。

4. DMA 访问

外部 DMA 访问 P、X 或 Y 存贮空间是正常总线周期，并且不能和 CPU 读/写周期区分。所以 DMA 访问可用$\overline{TT}$端并且不需要外部硬件的特殊处理。

5. 多个存贮器组

当外部存贮器多于覆盖 32 位数据总线所需的数量时，存在多个存贮器组。此时，外部存贮器控制器以若干行地址选通脉冲（$\overline{RAS}$）信号或列地址选通脉冲（$\overline{CAS}$）信号之一的使能在组之间选择，因为从一个存贮器组到另一个将产生页面故障，允许多个存贮器组，且不要求特殊处理。

6. 多个存贮器控制器

多存贮器控制器可能存在，以用多个外部实际存贮器支持快访问方式。因为页面电路可监视多存贮空间检测或忽略存贮空间的变化，允许多存贮器控制器，并且不要求特殊处理。

18.3 扩展端口选择

每个存贮空间（X、Y 和 P）分为 8 个相等部分。分割是固定的，即每部分大小是 0.5 千兆字，而且地址边界也是固定的。每个存贮空间的每部分可分别指定外部扩展端口之一（A 或 B）。由端口选择寄存器（PSR）控制映射。

18.3.1 端口选择寄存器（PSR）

端口选择寄存器是 32 位宽位于 X1/O 存贮器空间的读/写寄存器。对每个存贮空间的每个部分有一位在端口选择寄存器（PSR）中。若此位被清除，则相应的部分通过端口 A。若此位被设置，则它通过端口 B。

任何定义为内部的存贮器段仍然是内部的。端口选择寄存器格式示于图 18－6，并说明如下。

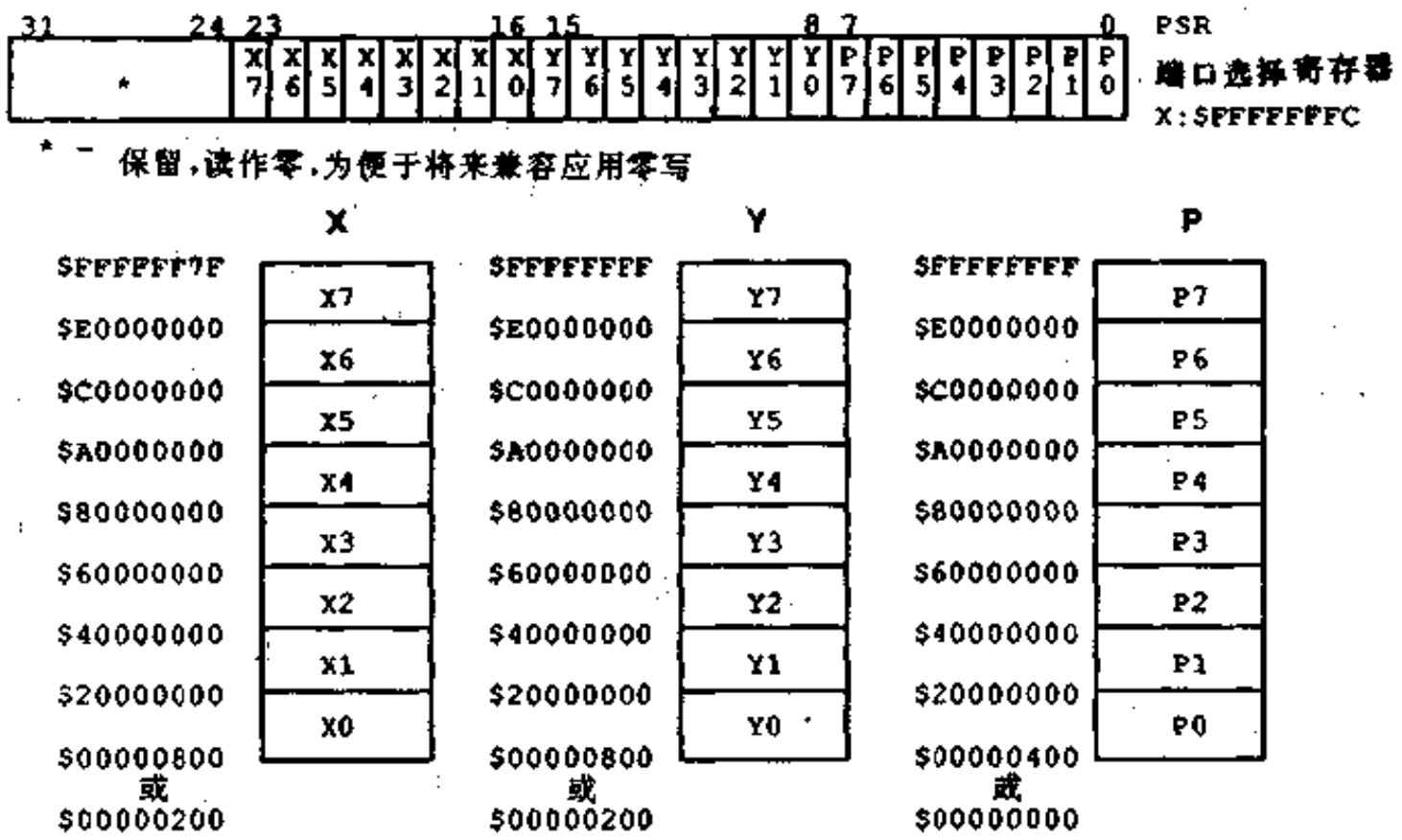


图 18－6 DSP96002 端口选择寄存器（PSR）

1. PSR 程序存贮器端口选择（P0～P7）位 0～7

程序存贮器端口选择控制位（P0～P7）决定 8 个程序存贮器段指定到端口 A 或 B。若段位被清除，则程序存贮器段指定到端口 A。若段位被设置，则到端口 B。存贮器段控制位的关系示于图 18－6 中。例如，若 P4 位被设置，则所有存贮器信息从地址 P：$80000000～P：$9FFFFFFF 将通过端口 B。当硬件复位时，若在取消$\overline{\text{RESET}}$时 MODA 端保持低，则 P0～P7 位被清除。若在取消$\overline{\text{RESET}}$时 MODA 端保持高，则 P0～P7 被设置。

2. PSR Y 数据存贮器端口选择（Y0～Y7）位 8～15

Y 数据存贮器端口选择控制位（Y0～Y7）决定 8 个 Y 数据存贮器段分配给端口 A 或 B。若段位被清除，则 Y 数据存贮器段分配到端口 A。若段位被设置，则到端口 B。存贮段控制位关系示于图 18－6。例如，若 Y4 位被设置，则对地址 Y：$80000000～Y：$9FFFFFFF 的所有存贮器信息将通过端口 B。当硬件复位时，Y0～Y7 位被清除。

3. PSR X 数据存贮器端口选择（X0～X7）位 16～23

X 数据存贮器端口选择控制位（X0～X7）决定 8 个 X 数据存贮器段分配到端口 A 或 B。若段位被清除，则 X 数据存贮器段分配到端口 A。若段位被设置，则到端口 B。存贮器段控制位关系示于图 18－6。例如，若 X4 位设置，则对地址 X：$80000000～X：$9FFFFFFF 的所有信息将通过端口 B。当硬件复位时，X0～X7 位被清除。

4. PSR 保留位（位 24～31）（略）

18.4 主机接口

18.4.1 引言

DSP96002 对每个端口提供 MPU/DMA 主机接口。这些接口提供 32 位并行端口到主处理器或 DMA 控制器。

这些主机接口（HI）的作用是减少在很多计算机作图和其他多处理应用中的系统片数和“粘结”逻辑。每个 HI 有它自己的控制、状态和数据寄存器，并按 DSP96002 映射存贮器 I/O 处理。每个接口有若干专用中断矢量地址和控制位的允许/不允许中断，从而减少了与使用此接口有关的额外开销，因为每个中断源有它自身的服务程序。

HI 以一组“主功能”在多处理器环境中支持操作。调用这些特性的外部设备称作“主处理器”并可能是另一个 DSP96002 处理器或 32 位微处理器，如 68020、68030、68040 或 88000。有 32、24 或 16 位数据总线的主处理器可访问 HI 的所有状态和控制位。有 8 位数据总线的主处理器应外加硬件方能访问所有状态和控制位。

HI 的功能允许：

- 主处理器传送有任意地址的数据去/从不用外部共享存贮器的 DSP96002。
- 主处理器中断用多中断矢量而不用外部共享存贮器的 DSP96002。
- 主处理器（有 DMA 功能）传送数据块去/从不用外部共享存贮器的 DSP96002。
- 外部 DMA 控制器传送数据块去/从不用外部共享存贮器的 DSP96002。
- 有最小外部逻辑的无缓存系统以及大缓存系统。

HI 通过外部扩展端口和一组专用引出端连接到外部世界：

- 32 位双向数据总线 D0～D31。
- 5 条控制线：R/$\overline{W}$，$\overline{HS}$，$\overline{HA}$，$\overline{TS}$，$\overline{HR}$。
- 地址线 A2～A5

HI 表现为在主处理器地址空间占有 16 单元的存贮器映射外围。分开的发送和接收数据寄存器是双缓存的，允许 DSP96002 和主处理器有效地高速传送数据。主处理器和 HI 寄存器的通信是用标准主处理器指令和寻址方式完成的。提供握手标志查询中断驱动与主处理器的数据传送。

外部 DMA 控制器（如 MC68450）能完成 DSP96002 HI 和外部主处理器存贮器间的块数据传送。为此，HI 中提供“DMA 方式”。在此方式中，用$\overline{HA}$端使能访问 HI 中的发送/接收寄存器，不管地址线 A2～A5 的状态如何。

主处理器亦可对有主命令性质的 DSP96002 发出矢量的异常请求。主处理器可用矢量地址寄存器选择执行 256 个 DSP96002 异常程序中的任何一个。这种灵活性使主处理器程序员可执行大量的在 DSP96002 以内的预编程功能。主机异常允许主处理器读或写 DSP96002 寄存器、X、Y 或程序存贮器单位，并且完成控制和调试操作，若在 DSP96002 中为完成这些任务实现了异常程序。

DSP96002 看 HI 如同在 X 数据存贮空间占有 4 个 32 位字的存贮器映射外围。DSP96002

可用 HI 作为用标准查询或中断编程技术的正常存贮器映射的外围。

18.4.2 HI 复位

HI 受以下复位型式影响：

HW/SW 复位：硬件（HW）复位，由插入$\overline{\text{RESET}}$端产生，或软件（SW）复位，由执行 RESET 指令产生。在 HI 中状态和控制位受到的影响如图 18－7 和 18－8 中定义。

Host 复位：HI 专用复位，当 HCR 寄存器中的 HRES 位被设置时产生。仅 HI 状态位受影响如图 18－7 和 18－8。仅 DSP96002 可直接激活 HOST 复位，因为 HRES 位于 DSP96002 一边。注意只要 HRES 位被设置，HI 就保持此状态。HRES 位不是自清除的。

INIT：当 ICS 寄存器中的 INIT 位被设置时，HI 产生专用复位。仅 HI 状态位受影响如图 18－7 和 18－8 中定义。注意 INIT 可按 ICS 寄存器中 TREQ 和 RREQ 控制位的状态有选择地复位发送和/或接收通道。此外：INIT 位是自清除的，它不同于需要显式操作的 HRES 位。

寄存器名称	寄存器内容	HW/SW 复位	主机复位	INIT TREQ=1 RREQ=0	INIT TREQ=0 RREQ=1	INIT TREQ=1 RREQ=1	注释
ICS	HMRC	0	0	0	—	0	
	HRST	1	1	—	—	—	
	DMAE	0	—	—	—	—	
	HF3～HF2	0	—	—	—	—	
	HF1～HF0	0	—	—	—	—	
	HREQ	0	注意①	注意①	注意②	1	
	INIT	0	—	0	0	0	
	TYEQ	0	—	—	—	—	
	TREQ	0	—	1	0	1	
	RREQ	0	—	0	1	1	
	TRDY	1	1	1	—	1	
	TXDE	1	1	1	—	1	
	RXDF	0	0	—	0	0	
CVR	HC	0	—	—	—	—	
	HV7～HV0	$0E	—	—	—	—	端口 A
		$0F	—	—	—	—	端口 B
IVR	IV7～IV0	$0F	—	—	—	—	
SEM	SEM（15～0）	$0000	—	—	—	—	

注意：① HREQ＝TYEQ＋TREQ

② GREQ＝（TYEQ & TRDY）＋（TEWQ & TXDE）

符号：

HW——由插入外部端$\overline{\text{RESET}}$产生的硬件复位。

SW——执行$\overline{\text{RESET}}$指令产生的软件复位。

HOST——当 HRES＝1 时产生的主机专用复位。

INIT——当 INIT＝1 时产生的主机专用复位。

"1"——位设置。

"0"——位清除。

"—"——位不受影响。

"＋"——逻辑或。

"&"——逻辑与。

图 18－7　主机接口复位-主处理器一边

寄存器名称	寄存器内容	HW/SW复位	主机复位	INIT TREQ=1 RREQ=0	INIT TREQ=0 RREQ=1	INIT TREQ=1 RREQ=1	注释
HCR	HTWE	0					
	HYRE	0					
	HXWE	0					
	HXRE	0					
	HPWE	0					
	HPRE	0					
	HRES	1					
	HF3～HF2	0					
	HCIE	0					
	HTIE	0					
	HRIE	0					
HSR	HYWP	0	0	0	—	0	
	HYRP	0	0	0	—	0	
	HXWP	0	0	0	—	0	
	HXRP	0	0	0	—	0	
	HPWP	0	0	0	—	0	
	HPRP	0	0	0	—	0	
	HDMA	0	—	—	—	—	
	HFI～HFO	0	—	—	—	—	
	HCP	0	—	—	—	—	
	HTDE	1	1	—	1	1	
	HRDF	0	0	0	—	0	

图 18-8　主机接口复位-DSP96002 一边

18.4.3　停止时 HI 的原理

主处理器当 DSP96002 在停止状态中能读/写 HI 寄存器。若时钟在主处理器访问中间停止，则通过 HI 的标志设置和数据传送将冻结。时钟重新开始后传送和标志的设置将做完。

若当 DSP96002 在停止状态用$\overline{HR}$和主处理器读 RX 或写 TX，则$\overline{HR}$仅在退出停止状态后为不确定。

18.4.4　HI 编程模式

HI 框图示于图 18-9。HI 有两个编程模型——一个为 DSP96002 程序员，而一个为外部主处理器程序员。在大多数情况下，所用符号反映了 DSP96002 配置。HI-DSP96002 编程模型示于图 18-10。HI-外部主处理器编程模型示于图 18-11。HI 中断结构示于图 18-13。DSP96002 有两个 HI。两个 HI 寄存器除地址外是一样的。它们的名称有 A 或 B 作后缀，说明它们连接的端口。

18.4.5　主机发送数据寄存器（HTX）-DSP96002 一边

主机发送寄存器（HTX）用于 DSP96002 到主处理器的数据传送。DSP96002 把 HTX 寄存器看作 32 位只写寄存器。写 HTX 寄存器清除 HTDE。当 HTDE 被设置时，DSP96002 可编程 HTIE 位使主机发送数据中断。若 HTDE 位和接收数据满 RXDF 状态位被清除，则 HTX 寄存器传送 32 位数据到接收寄存器 RX。这种传送操作置定 RXDF 和 HTDE。

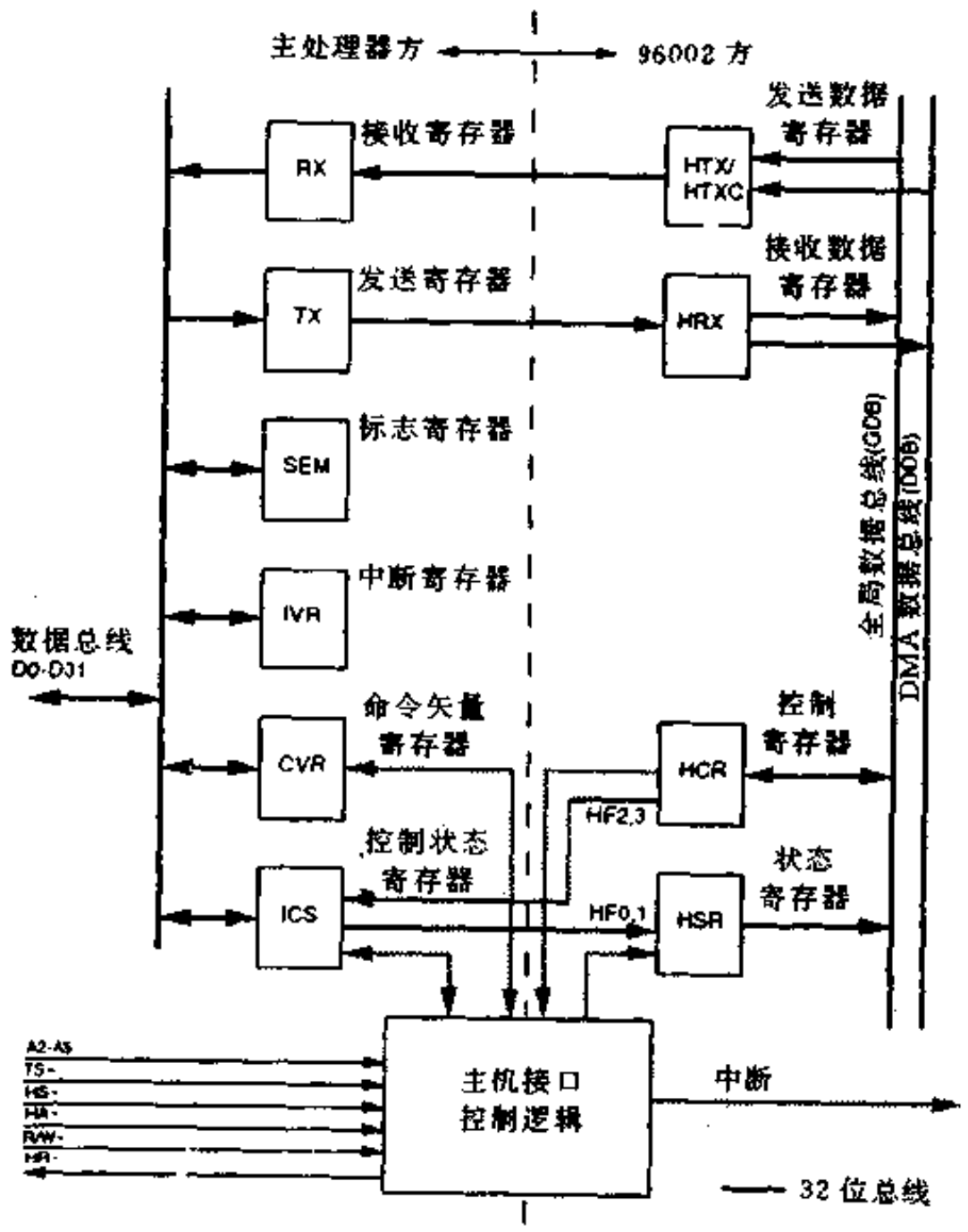


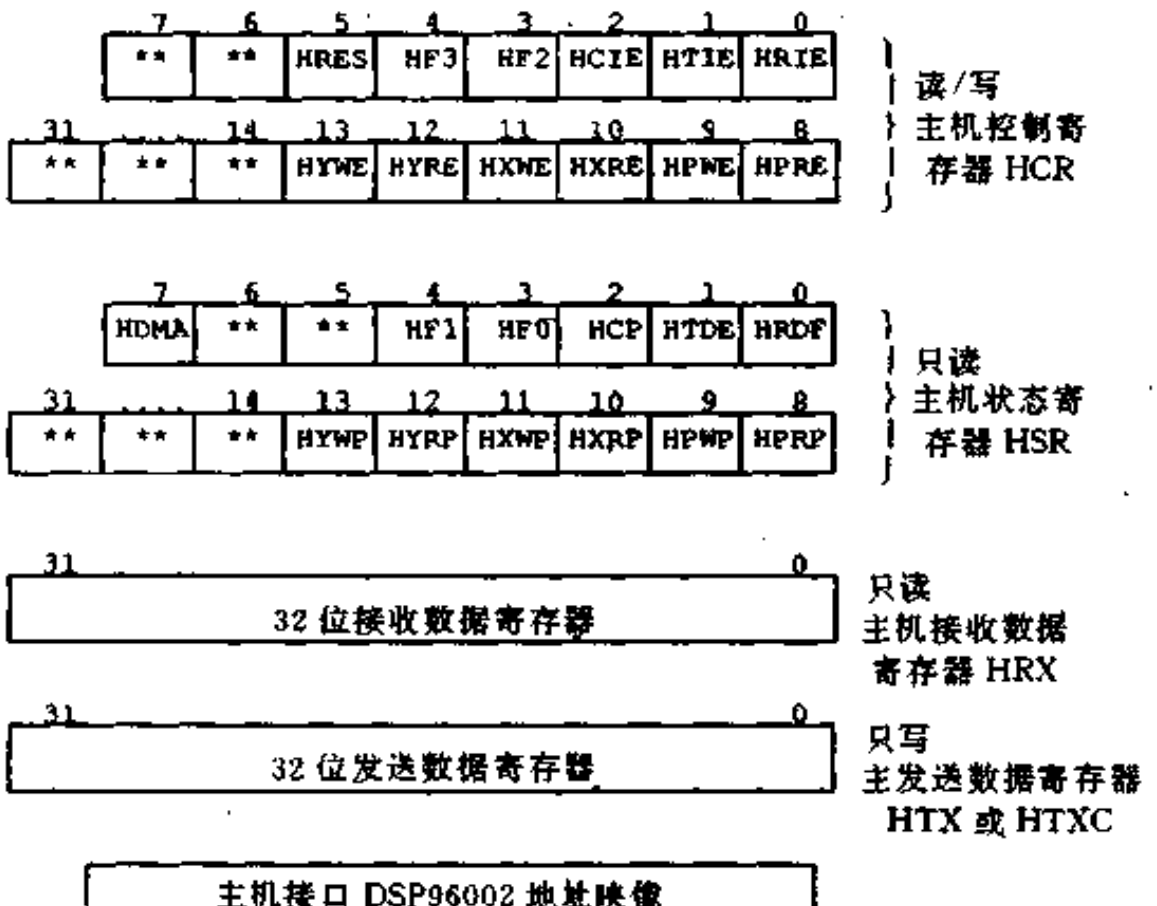
图 18-9 HI 框图（一个端口）

18.4.6 主机发送数据寄存器和 HMRC 清除（HTXC）-DSP96002 一边

HTXC 用于 DSP96002 到主处理器数据传送配合“TX 寄存器写（地址）和 X/Y/P 存贮器读（数据）中断”的主机功能。DSP96002 把 HTXC 寄存器看作 32 位只读寄存器。写 HTXC 寄存器清除 HTDE、HPRP、HXRP、和 HYRP。若 HTDE 位和接收数据满 RXDF 状态位被清除，则 HTXC 寄存器按 32 位数据传送到接收寄存器 RX。此传送操作设置 RXDF 和 HTDE，并清除 HMRC。

18.4.7 主机接收数据寄存器（HRX）-DSP96002 一边

HRX 用于主机处理器到 DSP96002 的数据传送。HRX 寄存器可由 DSP96002 看作 32 位只读寄存器。在发送寄存器空 TXDE 和主机接收数据满 HRDF 位清除时由 TX 寄存器以 32 位数据装入 HRX 寄存器。这传送操作置定 TXDE 和 HRDF。当 HRDF 位被设置时，HRX 寄存器包含有效值。读 HRX 清除 HRDF。DSP96002 可编程 HRIE 位使之在 HRDF 被置定时主机接收机数据中断。



主机接口 DSP96002 地址映像			
ADDR (HEX)		DSP96002 读	DSP96002 写
X:$FFFFFFEC		HCRA	HCRA
X:$FFFFFFED		HSRA	----
X:$FFFFFFEE		----	HTXCA
X:$FFFFFFEF		HRXA	HTXA
X:$FFFFFFE4		HCRB	HCRB
X:$FFFFFFE5		HSRB	----
X:$FFFFFFE6		----	HTXCB
X:$FFFFFFE7		HRXB	HTXB

端口 A

端口 B

** — 保留，读作零，为以后兼容应用零写

图 18-10 HI-DSP96002 编程模型

18.4.8 主机控制寄存器（HCR）~DSP96002 一边

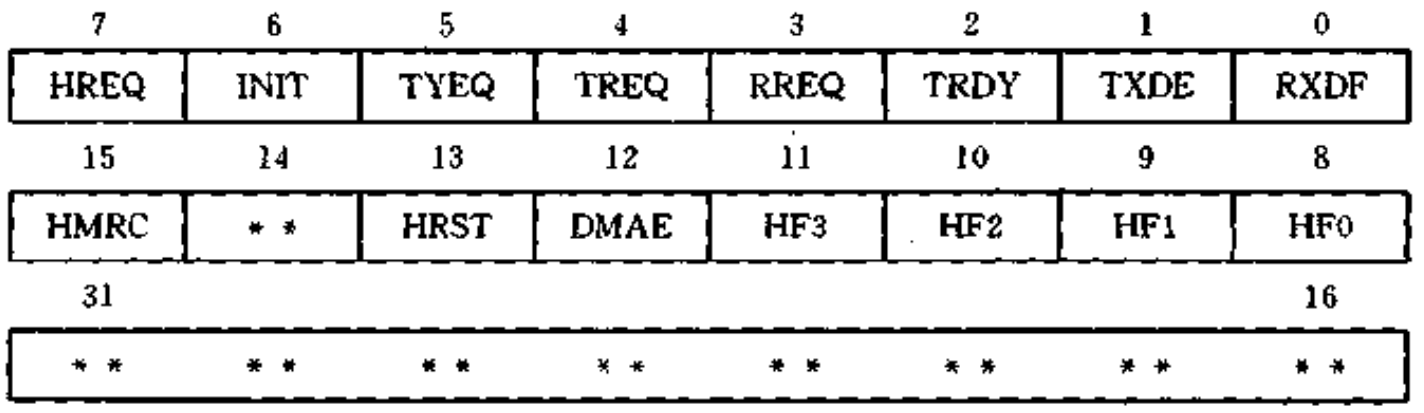
HCR 是 32 位读/写控制寄存器，由 DSP96002 用来控制 HI 中断和标志。HCR 不能被主处理器访问。HCR 是读/写寄存器允许位变换指令用于控制寄存器位。

1. HCR 主机接收中断使能（HRIE）位 0

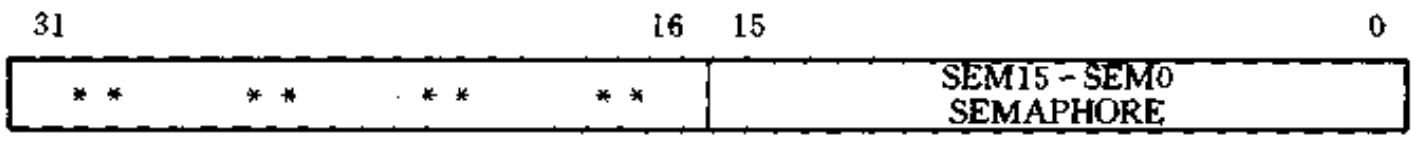
当主机接收数据满（HRDF）状态位在主机状态寄存器（HSR）中被设置时，HRIE 位用于允许主机接收数据中断。当 HRIE 被清除时，禁止 HRDF 中断。当 HRIE 被设置时，若 HRDF 被设置，则主机接收数据中断请求发生。HRIE 由 HW/SW 复位清除。

2. HCR 主机发送中断使能（HTIE）位 1

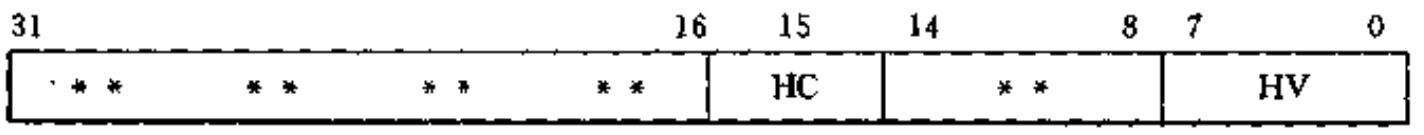
当主机发送数据空（HTDE）状态位在主机状态寄存器中被设置时，HTIE 位用于允许主机发送数据中断。当 HTIE 被清除时，禁止 HTDE 中断。当 HTIE 被设置时，若 HTDE 被设置，则主机发送数据中断请求发生。HTIE 由 HW/SW 复位清除。



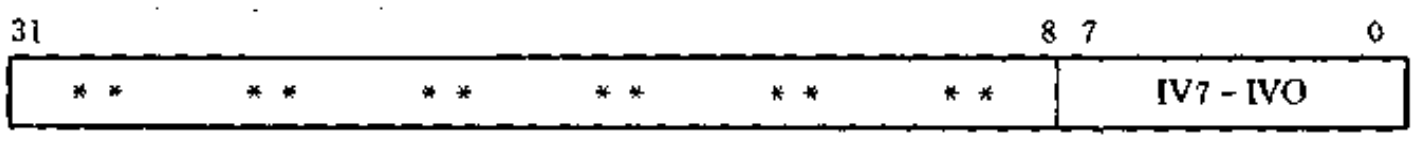
读/写中断控制/状态寄存器 ICS



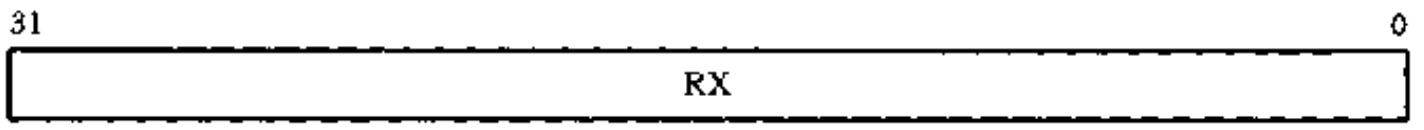
读/写寄存器 SEM



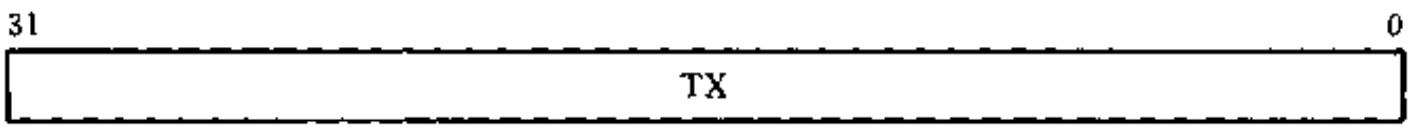
读/写命令矢量寄存器 CVR



读写/中断矢量寄存器 IVR



只读接收数据寄存器 RX



只写发送数据寄存器 TX

"*"保留位，读为 0，为将来兼容应写 0。

图 18-11 HI-主处理器编程模型

3. HCR 主机命令中断使能（HCIE）位 2

当主机命令未定（HCP）状态位在主机状态寄存器中被设置时，HCIE 位用于允许 DSP96002 中断矢量的主机命令。当 HCIE 被清除时，禁止 HCP 中断。当 HCIE 被设置时，若 HCP 被置定，则主机命令中断请求发生。这个中断的起始地址由主机矢量（HV）决定。HCIE 由 HW/SW 复位来清除。

4. HCR 主机标志 2（HF2）位 3

HF2 位为 DSP96002 到主处理器通信的通用标志。HF2 可由 DSP96002 设置或清除。HF2 状态可由主处理器读入 ICS 寄存器。HF2 由 HW/SW 复位来清除。

5. HCR 主机标志 3（HF3）位 4

主机标志 3（HF3）位用作 DSP96002 到主处理器通信的通用标志。HF3 可由 DSP96002 设置或清除。HF3 状态可由主处理器读入 ICS 寄存器。HF3 由 HW/SW 复位来清除。

6. HCR 主机复位（HRES）位 5

HRES 位用于复位 HI 的状态位，并初始化发送/接收通道到由硬件或软件复位产生的相同状态。当 HRES 置定时，主机复位（主机接口专用复位）产生。此位清除后主机接口退出主机复位状态。HRES 由 HW/SW 复位设置。

$\overline{HR}$	HA	$\overline{HS}$	R/$\overline{W}$	A5～A2	主机功能
×	1	1	×	××××	主机接口不能
×	1	0	1	1000	ICS 寄存器读
×	1	0	0	1000	ICS 寄存器写
×	1	0	1	1001	SEM 寄存器读
×	1	0	0	1001	SEM 寄存器写
×	1	0	1	1010	RX 寄存器读
×	1	0	0	1010	TX 寄存器写
×	1	0	×	1011	保留
×	1	0	1	1100	IVR 寄存器读
×	1	0	0	1100	IVR 寄存器写
×	1	0	1	1101	CVR 寄存器读
×	1	0	0	1101	CVR 寄存器写
×	1	0	×	1110	保留
×	1	0	×	1111	保留
×	1	0	0	0000	TX 寄存器写和 Y 存贮器写中断
×	1	0	0	0001	TX 寄存器写和 Y 存贮器读中断
×	1	0	0	0010	TX 寄存器写和 X 存贮器写中断
×	1	0	0	0011	TX 寄存器写和 X 存贮器读中断
×	1	0	0	0100	TX 寄存器写和 P 存贮器写中断
×	1	0	0	0101	TX 寄存器写和 P 存贮器读中断
×	1	0	0	011×	保留
×	1	0	1	0×××	保留
0	0	×	1	××××	IVR 读 (DMAE=0) —68K 中断认可
0	0	×	0	××××	保留 (DMAE=0)
1	0	×	×	××××	保留 (DMAE=0)
×	0	×	×	××××	RX 读 (DMA 方式：DMAE=1，TREQ=0，RREQ=1)
×	0	×	×	××××	TX 写 (DMA 方式：DMAE=1，TREQ=1，RREQ=0)
×	0	×	×	××××	保留 (DMA 方式：DMAE=1，TREQ=0，RREQ=0)
×	0	×	×	××××	保留 (DMA 方式：DMAE=1，TREQ=1，RREQ=1)

图 18－12　HI 功能

HI 中断源 (96002 一边)

中断源	状态	屏蔽	异常端口 A	开始地址端口 B
接收数据满	HRDF	HRIE	$00000020	$00000030
发送数据空	HTDE	HTIE	$00000022	$00000032
X 存贮器读	HXRP	HXRE	$00000024	$00000034
Y 存贮器读	HYRP	HYRE	$00000026	$00000036
P 存贮器读	HPRP	HPRE	$00000028	$00000038
X 存贮器写	HXWP	HXWE	$0000002A	$0000003A
Y 存贮器写	HYWP	HYWE	$0000002C	$0000003C
P 存贮器写	HPWP	HPWE	$0000002E	$0000003E
主机命令	HCP	HCIE	2×HV ($00000000～$000001FE)	

主机处理器 ($\overline{HR}$) 结构

$\overline{HR}$源	状态	屏蔽
接收数据满	RXDF	PREQ
发送数据空	TXDE	TREQ
发送器准备好	TRDY	TYEQ

图 18－13　HI 中断结构

7. HCR 保留位（6，7，14～31）略

8. HCR 主机 P 存贮器读中断使能（HPRE）位 8

当主机 P 存贮器读命令暂挂(HPRP)状态位在主机状态寄存器(HSR)中被设置时，HPRE 位用于允许 P 存贮器读中断。当 HPRE 被清除时，禁止 HPRP 中断。当 HPRE 被设置时，若 HPRP 被设置，则主机 P 存贮器读中断请求将发生。中断起始地址示于图 18－13。HPRE 由 HW/SW 复位来清除。

9. HCR 主机 P 存贮器写中断使能（HPWE）位 9

当主机 P 存贮器写命令暂挂(HPWP)状态位在主机状态寄存器(HSR)中被设置时，HPWE 位用于允许 P 存贮器写中断。当 HPWE 被清除时，禁止 HPWP 中断。当 HPWE 被设置时，若 HPWP 被置定，则主机 P 存贮器写中断请求将发生。此中断的起始地址示于图 18－13。HPWE 由 HW/SW 复位来清除。

10. HCR 主机 X 存贮器读中断使能（HXRE）位 10

当主机 X 存贮器读命令暂挂(HXRP)状态位在主机状态寄存器中被设置时，HXRE 位用来允许 X 存贮器读中断。当 HXRE 被清除时，禁止 HXRP 中断，当 HXRE 被设置时，若 HXRP 被置定，则主机 X 存贮器读中断请求将发生。此中断的起始地址示于图 18－13。HXRE 由 HW/SW 复位来清除。

11. HCR 主机 X 存贮器写中断使能（HXWE）位 11

当主机 X 存贮器写命令暂挂（HXWP）状态位在主机状态寄存器（HSR）中被设置时，HXWE 位用于 X 存贮器写中断允许。当 HXWE 被清除时。禁止 HXWP 中断。当 HXWE 被设置时，若 HXWP 被置定，则主机 X 存贮器写中断请求将发生。此中断的起始地址示于图 18－13。HXWE 由 HW/SW 复位来清除。

12. HCR 主机 Y 存贮器读中断使能（HYRE）位 12

当主机 Y 存贮器读命令暂挂（HYRP）状态位在主机状态寄存器（HSR）中被设置时，HYRE 位用于 Y 存贮器读中断允许。当 HYRE 被清除时，禁止 HYRP 中断。当 HYRE 被设置时，若 HYRP 被置定，则主机 Y 存贮器读中断请求将发生。此中断的起始地址示于图 18－13。HYRE 由 HW/SW 复位来清除。

13. HCR 主机 Y 存贮器写中断使能（HYWE）位 13

当主机 Y 存贮器写命令暂挂(HYWP)状态位在主机状态寄存器(HSR)中被设置时，主机 Y 存贮器写中断使能（HYWE）位用于 Y 存贮器写中断允许。当 HYWE 被清除时，不允许 HYWP 中断。当 HYWE 被设置时，若 HYWP 被置定，则主机 Y 存贮器写中断请求将发生。此中断的起始地址示于图 18－13。HYWE 由 HW/SW 复位来清除。

18.4.9 主机状态寄存器（HSR）－DSP96002 一边

主机状态寄存器(HSR)是 32 位只读状态寄存器，由 DSP96002 用于询问状态和 HI 的标志。它不能直接被主处理器访问。

1. HSR 主机接收数据满（HRDF）位 0

HRDF 位表示主机接收数据寄存器（HRX）包含来自主处理器的数据，仅由主处理器经过主机功能“TX 寄存器写”写入。当数据由 TX 寄存器传送到 HRX 寄存器时 HRDF 被设置。当接收数据寄存器由 DSP96002 读时，HRDF 被清除。HRDF 由 INIT（TREQ＝1）、主机复

位和 HW/SW 复位来清除。

2. HSR 主机发送数据空（HTDE）位 1

HTDE 位指出主机发送数据寄存器（HTX）空并且可由 DSP96002 写。当 HTX 寄存器传送到 RX 寄存器时 HTDE 被设置。当发送数据寄存器 HTX 被 DSP96002 写时 HTDE 被清除。HTDE 由 INIT（RREQ=1）、主机复位、和 HW/SW 复位来清除。

3. HSR 主机命令暂挂（HCP）位 2

HCP 位指出主机处理器已设置 HC 位并且主机命令中断正处于暂挂。HCP 位反映在命令矢量寄存器（CVR）中 HC 位的状态。当取出主机命令中断的第二个矢量单元时，HC 和 HCP 由 DSP96002 特殊硬件清除。HCP 由 HW/SW 复位来清除。

4. HSR 主机标志 0（HF0）位 3

HF0 位指出在中断控制寄存器 ICS 中主机标志 0 的状态。HF0 仅能由主处理器改变，并由 HW/SW 复位来清除。

5. HSR 主机标志 1（HF1）位 4

HF1 位指出在中断控制寄存器 ICS 中主机标志 1 的状态。HF1 仅能由主处理器改变，并由 HW/SW 复位来清除。

6. HSR 保留位（5，6，14～31）

这些位用于将来扩展。

7. HSR DMA 状态（HDMA）位 7

HDMA 位指出主处理器允许 HI 的外部 DMA 握手方式。当 HDMA 被清除时，它表示在中断控制寄存器 ICS 中 DMA 方式不允许（DMAE=0）。当 HDMA 被设置时，它表示 MDA 方式允许（DMAE=1）。由 HW/SW 复位来清除。

8. HSR 主机 P 存贮器读命令暂挂（HPRP）位 8

HPRP 位指出 HRX 寄存器包含来自由主处理器经过主机功能“TX 寄存器写和 P 存贮器读中断”写的主处理器的数据。当数据从 TX 寄存器传送到 HRX 寄存器时 HPRP 被设置。当 HTXC 寄存器由 DSP96002 写入时 HPRP 被清除。HPRP 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

9. HSR 主机 P 存贮器写命令暂挂（HPWP）位 9

HPWP 位指出 HRX 和 TX 寄存器包括来自由主处理器经主机功能“TX 寄存器写和 P 存贮器写中断”写入的主处理器数据。当主处理器用主机功能为连续第二次写 TX 时，HPWP 被设置。当 HRX 寄存器由 DSP96002 连续读二次（一次为地址而一次为数据）时，HPWP 被清除。HPWP 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

10. HSR 主机 X 存贮器读命令暂挂（HXRP）位 10

HXRP 位指出 HRX 寄存器包括来自主处理器的数据，它由主处理器经主机功能“TX 寄存器写和 X 存贮器读中断”写入。当数据从 TX 寄存器传送到 HRX 寄存器时，HXRP 被设置。当 HTXC 寄存器被 DSP96002 写入时 HXRP 被清除。HXRP 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

11. HSR 主机 X 存贮器写命令暂挂（HXWP）位 11

HXWP 位指出 HRX 和 TX 寄存器包含来自主处理器的数据，它由主处理器经过主机功能“TX 寄存器写和 X 存贮器写中断”写入。当主处理器为连续第二次写 TX 用此主机功能时，

HXWP 被设置。当 DSP96002 连续二次读 HRX 寄存器时，HXWP 被清除。HXWP 由 INIT (TREQ=1)、主机复位和 HW/SW 复位来清除。

12. HSR 主机 Y 存贮器读命令暂挂（HYRP）位 12

HYRP 位指出 HRX 寄存器包含来自主处理器的数据，它由主处理器经过主机功能“TX 寄存器写和 Y 存贮器读中断”写入。当数据从 TX 寄存器传送到 HRX 寄存器时设置 HYRP。当 DSP96002 写入 HTXC 寄存器时清除 HYWP。HYRP 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

13. HSR 主机 Y 存贮器写命令暂挂（HYWP）位 13

HYWP 位指出 HRX 和 TX 寄存器包含来自主处理器的数据，它由主处理器经主机功能“TX 寄存器写和 Y 存贮器写中断”写入。当主处理器用主机功能连续第二次写 TX 时设置 HYWP。当 DSP96002 连续二次读 HRX 寄存器时清除 HYWP。HYWP 由 INIT (TREQ=1)、主机复位和 HW/SW 复位来清除。

18.4.10 接收寄存器（RX）-主处理器一边

32 位寄存器接收来自主机发送数据寄存器 HTX 的数据。当 RXDF 位被设置时，RX 寄存器包含有效数据。主处理器可编程接收请求使能位（RREQ），当设置 RXDF 时确定主机请求 $\overline{HR}$端。这通知主处理器接收寄存器 RX 已满。RXDF 位随着读 RX 寄存器而被清除。

外部主处理器把 RX 寄存器看作在其存贮器映像中的地址并可由主处理器读操作来读。当 HI 在 DMA 方式（DMAE=1）时 RX 寄存器也可由外部 DMA 控制器来读（不需 A2～A5 地址）。

18.4.11 发送寄存器（TX）-主处理器一边

32 位寄存器送数据到主机接收数据寄存器 HRX。当 TXDE 位被清除时，TX 寄存器包含有效数据。随着写 TX 寄存器，清除 TXDE 位。主处理器可编程发送请求使能位 (TREQ)，当 TXDE 被设置时，确定主机请求$\overline{HR}$端。这通知主处理器 TX 寄存器是空。

外部主处理器把 TX 看作在其存贮器映像中的地址并可由主处理器存贮器写操作来写入。TX 寄存器在 HI 是 DMA 方式时（DMAE=1）也可由外部 DMA 控制器写入。

18.4.12 命令矢量寄存器（CVR）-主处理器一边

32 位命令矢量寄存器（CVR）被主处理器用来请求来自 DSP96002 的矢量特殊服务。在 DSP96002 可以指定任何特殊程序。主机命令特性与 HI 中任何数据传送机理无关。

1. CVR 主机矢量（HV）位 0～7

8 位主机矢量（HV）间接指定主机命令特殊地址。当 DSP96002 中断控制逻辑识别主机命令异常时，所取异常的起始地址是 2×HV。这允许主处理器对主机命令异常改变异常起始地址。主处理器把异常程序起始地址除以 2 写入 HV 即可选择 DSP96002 中 256 个可能的异常程序起始地址的任何一个。这意味着主处理器可控制任何现有的异常处理程序（IRQA、IRQB 等）并可以使用任何保留的或其他未用的 DSP96002 中已预先编程的起始地址。HV 由 HW/SW 复位对每个端口设置到预定值（看图 18-7）。若 HC 被设置，则主处理器应不改变 HV。

2. CVR 保留位（8～14，16～31）（略）

3. CVR 主机命令（HC）位 15

主处理器用 HC 起始执行主机命令异常。主处理器正常设置 HC 以请求来自 DSP96002 的主机命令异常服务。设置 HC 使 HCP 在 HSR 寄存器中被设置。当主机命令第二矢量单元取出时，HC 位被 HI 硬件（中断确认）清除。HC 由 HW/SW 复位来清除。

18.4.13 中断控制/状态寄存器（ICS）-主处理器一边

ICS 是 32 位读/写控制和状态寄存器，由主处理器用来控制 HI 和验证 HI 的当前状态。ICS 是读/写寄存器，可用位变换指令访问。以下几节说明控制和状态位。

1. ICS 接收数据寄存器满（RXDF）位 0

只读的接收数据寄存器满（RXDF）位指出接收寄存器 RX 包含来自 DSP96002 的数据并可由主处理器来读。当主机发送数据寄存器 HTX 或 HTXC 传送到接收寄存器 RX 时设置 RXDF。当 RX 被主处理器读时清除 RXDF。RXDF 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

若接收请求使能位（RREQ）被设置，则 RXDF 可用于确定主机请求$\overline{HR}$端。不管 RXDF 中断是允许或不允许，FXDF 提供有效状态，因此查询方法可由主处理器使用。

2. ICS 发送数据寄存器空（TXDE）位 1

只读的发送数据寄存器空（TXDE）位指出发送寄存器 TX 是空的并可由主处理器写。当发送寄存器 TX 传送到主接收机数据寄存器（HRX）时设置 TXDE。当主处理器写 TX 时，清除 TXDE。TXDE 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

3. ICS 发送器准备（TRDY）位 2

TRDY 状态位指出发送寄存器 TX（在主处理器一边）和主接收数据寄存器 HRX（在 DSP96002 一边）都是空。若发送准备请求使能位（TYEQ）被设置，则 TRDY 可用来确定主机请求$\overline{HR}$端。不管 TRDY 中断是允许或不允许，TRDY 提供有效状态，因此主处理器可使用查询方法。TRDY 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

4. ICS 接收请求使能（RREQ）位 3

当设置接收数据寄存器满（RXDF）状态位时，用 RREQ 经过外部主机请求$\overline{HR}$端允许主处理器中断/请求。当 RREQ 被清除时，不允许 RXDF 中断。当 RREQ 被设置时，若 RXDF 被设置，则主机请求$\overline{HR}$端将被确定。

在 DMA 方式中，RREQ 必须由软件设置或清除以选择 DMA 传送方向。设置 RREQ，定义 DMA 传送方向是 DSP96002→外部 DMA，并使$\overline{HR}$端能请求这些数据传送。

图 18-14 和图 18-15 是$\overline{HR}$端上 RREQ 影响的总结。RREQ 由 HW/SW 复位来清除。

5. ICS 发送请求使能（TREQ）位 4

当设置发送数据寄存器空（TXDE）状态位时，用 TREQ 经主机请求$\overline{HR}$端允许主处理器中断/请求。当 TREQ 被清除时，不允许 TXDE 中断。当 TREQ 被设置时，若 TXDE 被置定，则主机请求$\overline{HR}$端将被确定。

在 DMA 方式中，TREQ 必须由软件设置或清除以选择 DMA 传送方向。置定 TREQ 定义 DMA 传送方向是从外部 DMA→96002，并允许$\overline{HR}$端请求这些数据传送。

图 18-14 和 18-15 是在$\overline{HR}$端上 TREQ 的影响的总结。TREQ 由 HW/SW 复位来清除。

TREQ	RREQ	TYEQ	HREQ 标志和$\overline{HR}$端
0	0	0	不中断（查询）
0	1	0	RX 满或 HTX 满
1	0	0	TX 空或 HRX 空
1	1	0	RX 满、HTX 满、TX 空或 HRX 空
×	0	1	TX 空和 HRX 空
×	1	1	全中断（不查询）

图 18-14　HREQ 和$\overline{HR}$定义——中断方式（DMAE=0）

TREQ	RREQ	TYEQ	HREQ 标志和$\overline{HR}$端
0	0	0	保留
0	1	0	DSP96002→DMA 请求（RX 满）
1	0	0	DMA→DSP96002 请求（TX 空）
1	1	0	保留
×	×	1	保留

图 18-15　HREQ、$\overline{HR}$和 DMA 传送方向定义——DMA 方式（DMAE=1）

6. ICS 发送器准备请求使能（TYEQ）位 5

当置定发送器准备（TRDY）状态位时，用 TYEQ 经主机请求$\overline{HR}$端允许中断。当 TYEQ 被清除时，不允许 TRDY 中断。当 TYEQ 被置定时，若 TRDY 被置定，则主机请求$\overline{HR}$端将被确定。

图 18-14 是$\overline{HR}$端上 TYEQ 影响的总结。TYEQ 由 HW/SW 复位来清除。在 DMA 方式中，TYEQ 必须清除。

7. ICS 初始化（INIT）位 6

主处理器用 INIT 位强迫 HI 硬件初始化。这是否需要，取决于接口的软件设计。

为了正确地初始化 HI，与 ICS 中决定初始过程的其他控制位一起置定 INIT 位。可能以同样的命令写所有的位。置定 INIT 位以后，HI 开始初始化过程，并在过程结束时 HI 清除 INIT 位。在初始化过程中，主处理器不应去读 RX、写 TX、或写 ICS 寄存器。主处理器首先应采用以下方法之一，保证初始化过程完成。

（1）当用$\overline{HR}$端握手时，等待到$\overline{HR}$确定后开始写/读数据。

（2）当不用$\overline{HR}$端握手时，用查询 ICS 中 INIT 位以确定它已由硬件清除（这意味着已完成 INIT 的执行）。然后开始写/读数据。

（3）若既不用$\overline{HR}$端握手也不查询 INIT 位，则在写/读数据以前，写 ICS 的$\overline{TS}$不确定以后，至少等待 3Tc+Th。这样保证 INIT 完成。看图 18-16。

所做的初始化类型决定于 TREQ 和 RREQ 的状态。若 TREQ 和 RREQ 都被清除，则 INIT 过程将不影响 HI。初始化过程的影响在图 18-7 和 18-8 中说明。INIT 位由 HW/SW 复位来清除。

8. ICS 主机请求（HREQ）位 7

只读主机请求（HREQ）位指出主机请求$\overline{HR}$端的状态。在中断方式（DMAE=0）中：

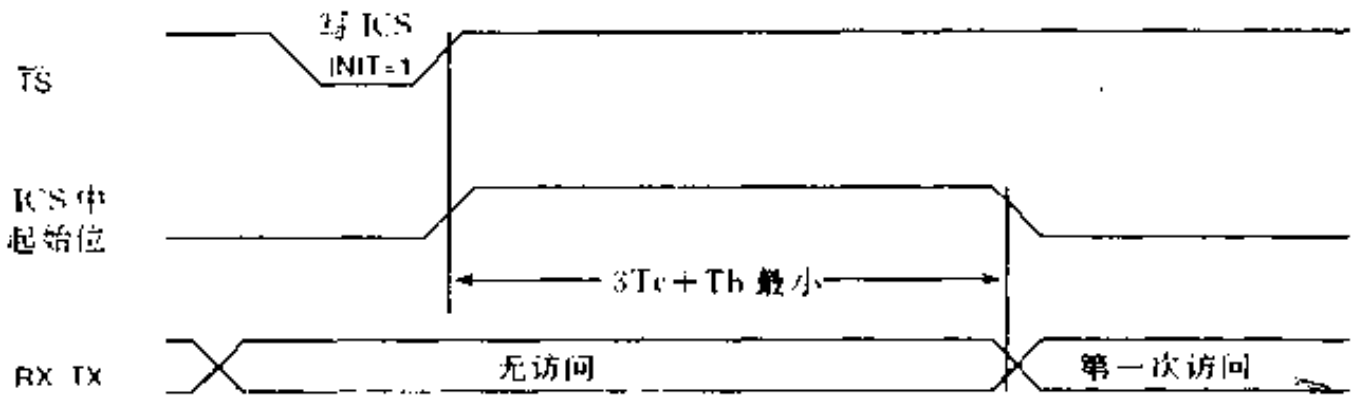


图 18-16　保证正确执行 INIT 的最小迟延

当 HREQ 状态位被清除时，它指出$\overline{HR}$端不确定，且不是正在请求主处理器中断。当 HREQ 状态位被置定时，它指出$\overline{HR}$端确定，表明 DSP96002 正在中断主处理器。HREQ 中断请求可能由 3 个源中之一个或多个开始，由它们的使能位 RREQ、TREQ 和 TYEQ（看图 18-14）来选择：

- RX 寄存器或 HTX 寄存器满。
- TX 寄存器或 HRX 寄存器空。
- TX 寄存器（在主处理器一边）和 HRX 寄存器（在 DSP96002 一边）都空。

在 DMA 方式（DMAE=1）中：

当 HREQ 状态位被清除时，它指出$\overline{HR}$端不确定并且没有请求 DMA 传送。当 HREQ 位被置定时，它指出$\overline{HR}$端确定并且正在进行 DMA 传送的请求。当 DMA 传送方向是 DSP96002→外部 DMA 时，因为接收寄存器（RX）满，或当 DMA 传送方向是外部 DMA→DSP96002 时，因为发送寄存器（TX）空，所以可能发生 DMA 传送请求（看图 18-15）。

RX 满和 TX 空的情况分别由 ICS 寄存器 RXDF 和 TXDE 状态位指出。若中断控制寄存器 ICS 中的有关请求使能位已允许中断源，并若 1 或 2 个使能中断源被置定，则 HREQ 将被置定。HREQ 由 HW/SW 复位来清除。若 TYEQ 和 TREQ 均被清除，则 HREQ 由主机复位清除，否则是设置。INIT 对 HREQ 的影响看图 18-7。

9. ICS 主机标志 0（HF0）位 8

HF0 位用作主处理器到 DSP96002 通信中的通用标志。HF0 可由主处理器来设置或清除。并由 HW/SW 复位来清除。HR0 的状态可在 HSR 位 3 中读。

10. ICS 主机标志 1（HF1）位 9

HF1 位用作主处理器到 DSP96002 通信中的通用标志。HF1 可由主处理器来设置或清除。HF1 由 HW/SW 复位来清除。HF1 的状态可在 HSR 位 4 中读。

11. ICS 主机标志 2（HF2）位 10

HF2 位指出主机控制寄存器 HCR 中主机标志 2 的状态。HF2 仅可由 DSP96002 来改变。它由 HW/SW 复位来清除。

12. ICS 主机标志 3（HF3）位 11

HF3 位指出主机控制寄存器 HCR 中主机标志 3 的状态。HF3 仅可由 DSP96002 来改变。它由 HW/SW 复位来清除。

13. ICS DMA 方式使能（DMAE）位 12

DMAE 选择 HI 的操作方式。当 DMAE 被置定时，HI 以 DMA 方式操作。当 DMAE 被

清除时，不允许 DMA 方式。由 HW/SW 复位来清除。

当 DMAE 被清除时，地址线 A2～A5 选择 HI 寄存器。这种操作方法适合于与外部设备接口，如微处理器，它能提供地址。在此方式中，$\overline{HR}$端可用作对主处理器的中断请求，并且$\overline{HA}$端可用来支持 68K 族的中断确认。

当 DMAE 被置定时，HI 以 DMA 方式操作。当在 DMA 方式时，访问 RX 和 TX 寄存器与地址线 A2～A5 无关，允许数据在诸如不提供地址的 DMA 控制器等外设的控制下传送。$\overline{HR}$端用作为对外部 DMA 控制器的 DMA 传送请求。DMA 传送的方向由 TREQ 和 RREQ 来选择。不支持双向 DMA 传送；用户不能在 DMA 方式中设置 RREQ 和 TREQ。TYEQ 也应保持清除。

14. ICS 主机复位状态（HRST）位 13

HRST 位可由主处理器测试以验证 HRES 控制位的状态。若 HRST 被置定，则 HRES 位被置定并且 HI 在复位状态。若 HRST 被清除，则 HRES 位被清除并且允许 HI 工作。HRST 位由清除 HRES 来清除。HRST 由主机复位和 HW/SW 复位来设置。

15. ICS 保留位（14，16～31）

16. ICS 主存贮器读命令（HMRC）位 15

HMRC 位可由主处理器测试以验证何时写入 HTXC 寄存器（96002 一边）的数据传送到 RX 寄存器（主处理器一边）。

当主处理器用主机功能“TX 寄存器写和 X/Y/P 存贮器读中断”写入 TX 寄存器时，HMRC 被置定。当在 DSP96002 一边 HTX 寄存器已写入的内容通过 HTXC 地址传送到主处理器一边的 RX 寄存器时，HMRC 被清除。HMRC 由 INIT（TREQ=1）、主机复位和 HW/SW 复位来清除。

18.4.14 信标寄存器（SEM）-主处理器一边

信标寄存器（SEM）是 32 位读/写寄存器，由主处理器用来控制多处理器系统中 HI 的分配和表示当前主处理器的标识。

1. SEM 主机信标（SEM0～SEM15）位 0～15

主机信标寄存器位 SEM0～SEM15 由主处理器用于控制 HI 对的软件仲裁。这个寄存器不影响 HI 工作并仅作为读/写信标的容器。要竞争控制权的所有外部主处理器应按相同的软件协议工作，以便从一个主处理器到另一个主处理器交接。

典型地，在访问 HI 以前，主处理器校验信标寄存器看 HI 是否分配到另一个主处理器。若 SEM0～SEM15 未被清除，则 HI 已经分配而主处理器不能访问 HI。若 SEM0～SEM15 被清除，则 HI 空闲而主处理器写 SEM0～SEM15。主处理器既可只设置一位、几位，也可写整个 16 位（如用作主处理器标识）。

主处理器应用读/修正/写的不可中断指令（如 MC88000 中的 XMEM、MC680$_x$0 中的 CAS 或 DSP96002 中的 BEST）并考察哪个主处理器已分配 HI 或由“位测试和设置”指令置定信标位。DSP96002 中的 BEST 是“不可中断”的，因为它测试信标位并在状态寄存器示出结果，然后不释放总线，设置信标位。这组合操作防止另一个处理器在 BEST 测试和设置操作之间读或写。

当前 HI 完成其传送后，它必须随清除信标寄存器位释放 HI。SEM0～SEM15 由 HW/SW

复位来清除。

2. SEM 保留位（16～31）

18.4.15 中断矢量寄存器（IVR）-主处理器一边

中断矢量寄存器 IVR 是 32 位读/写寄存器，它包含和 MC680×0 处理器族矢量中断一起用的异常矢量数。

1. IVR 中断矢量（IVR0～IVR7）位 0～7

当不在 DMA 方式时（DMAE＝0），且当$\overline{HR}$和$\overline{HA}$确定时，IVR 寄存器的内容可由确定$\overline{TS}$读到数据总线。在 HW/SW 复位时，IVR 寄存器的内容初始化到 $0F。这相应于 MC68K 族中的非初始化异常矢量。

IVR 寄存器也可作为用地址线 A2～A5 的常规读/写寄存器由主处理器访问，如图 18-12。

2. IVR 保留位（8～31）

18.4.16 HI 中断

HI 可由 DSP96002 核心或外部主处理器请求中断服务。对 DSP96002 的 HI 中断请求是内部的，因而不要求用外部中断引出端。DSP96002 核心以取适当的中断矢量单元服务于 HI 中断。中断服务程序必须读或写适当的 HI 寄存器以清除中断请求（如读 HRX 以清除 HRDF）。在主命令中断情况下，来自 DSP96002 核心的中断确认（当取到第二个中断矢量单元时产生）将清除暂挂中断情况。

用主机请求$\overline{HR}$端对外部主处理器的 HI 中断请求。$\overline{HR}$通常连接到主处理器中断输入。主处理器以执行中断服务程序确认 HI 中断。当确定$\overline{HR}$和$\overline{HA}$从 HI 的 IVR 寄存器读异常矢量数时，MC680×0 处理器族将确定$\overline{TS}$端。在多 DSP96002 系统中，可测试中断状态寄存器（ICS）中的 HREQ 位来决定哪个 DSP96002 HI 是中断设备，并且在测试 RXDF、TXDE 和 TRDY 位来决定中断源。主处理器中断服务程序必须读或写适当 HI 寄存器以清除中断并不确定$\overline{HR}$。

18.4.17 主处理器程序员考虑事项

1. 读 RX

当读接收寄存器 RX 时，主处理器程序员应使用中断或查询表明有可用数据的 RXDF 标志。这保证 RX 寄存器中的数据是稳定的。

2. 写 RX

主处理器程序员不应写发送寄存器 TX，除非 TXDE 已被置定，指出 TX 寄存器是空。这可保证 HI 将稳定的数据传送到 DSP96002 一边的 HRX 寄存器。

3. 从 DSP96002 到主处理器状态位的同步

由 HI 的 DSP96002 一边，设置和清除 HC、HMRC、HREQ、HF3、HF2、TRDY、TXDE 和 RXDF 并由主处理器读。主处理器可以读这些状态位与 DSP96002 所用时钟速率无关，但在读操作时有一个意外状态位可能改变。这通常不是系统问题，因为若此位改变，读就表示应进行另一次查询，并且此位将在下一个查询程序通过时正确地读。

由主机读任何状态位的不定性造成的唯一的潜在系统问题是在 HF3 和 HF2 用作编码对时。例如，若 DSP96002 改变 HF3 和 HF2 从“00”到“11”，则有很小的概率使主处理器在过渡时读此位并接收到“01”或“10”而不是“11”。若 HF3 和 HF2 的组合有意义，则建议读二次 HF3 和 HF2 并检验其一致性。

4. 写主机矢量寄存器

主处理器程序员仅在主机命令位（HC）清除时，应改变主机矢量寄存器。清除 HC 是 DSP96002 HI 的任务而不应由主程序员来做。这保证 DSP96002 中断控制逻辑接收稳定矢量。

18.4.18 96002 程序员考虑事项

读状态位

HF1、HF0、HCP、HPRP、HPWP、HXRP、HXWP、HYRP、HYWP、HTDE 和 HRDF 状态位由 HI 的主处理器一边设置和清除。这些位每个都与 DSP96002 时钟同步。

若 HF1 和 HF0 成对编码如 4 种组合 00、01、10 和 11 每组都有效，则与读状态有关的唯一的系统问题是 HF1 和 HF0。这是因为有很小的概率使 DSP96002 在过渡时读已同步的状态位。这个潜在问题的解决方法是读两次这些位。

18.4.19 DSP96002 到 DSP96002 数据传送——举例

本节举例说明为在 DSP96002 处理器之间传送数据，HI 和片上 DMA 控制器的使用。主总线用常规的存贮器访问去访问从属的 HI。从属的 HI 寄存器是映射总线主存贮空间的存贮器。注意主总线 HI 是不用的并且从属 HI 不在 DMA 方式中。

1. 用片上 DMA 控制器写数据

此例略述以下步骤，即作用如同主处理器的 DSP96002 主总线通过从属的 HI 传送数据到 DSP96002 从总线，两个 DSP96002 的片上 DMA 控制器都用来传送数据，不妨碍两个片的自身处理。图 18－17 包含用于数据传送的表示数据通路和控制线的图。

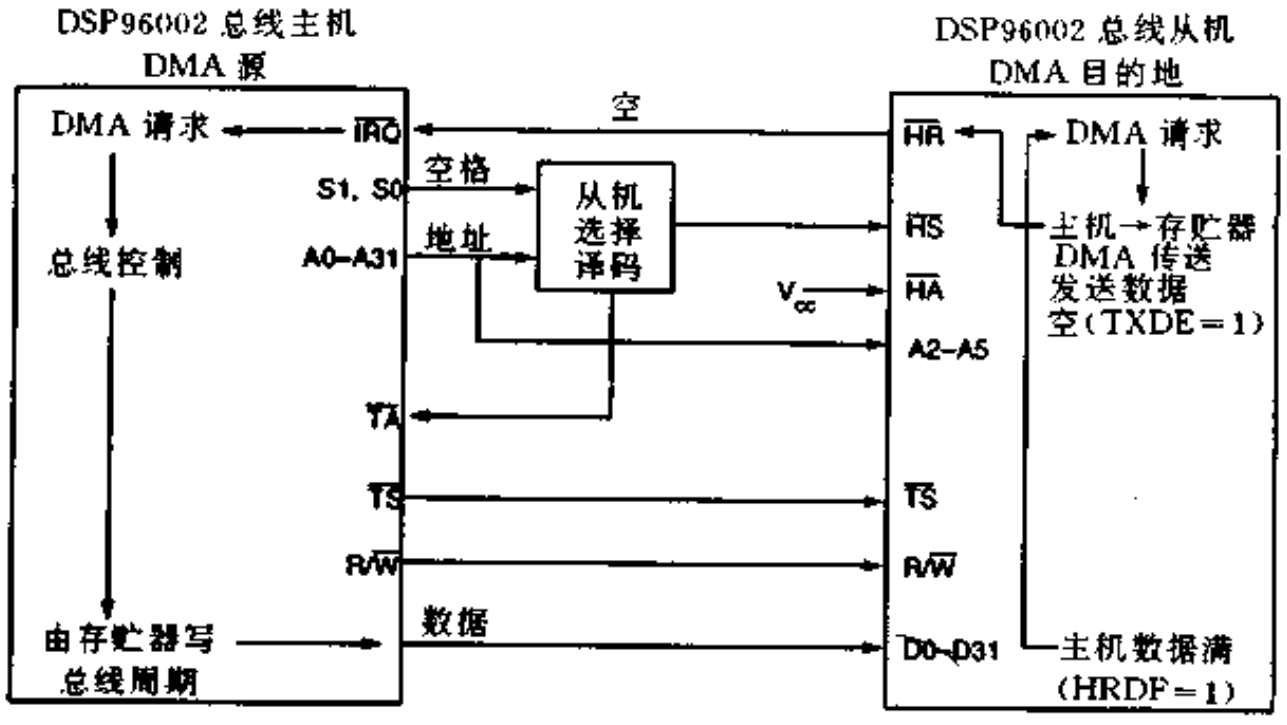


图 18－17　DSP96002 到 DSP96002 数据传

当从属的$\overline{HR}$信号确定时，表示它的 HI TX 寄存器空，并准备从主方接收数据字，则开始

数据写传送。$\overline{HR}$信号连接到主片中的$\overline{IRQ}$端，这里此端定义为 DMA 服务请求输入。当$\overline{HR}$确定时，主 DMA 控制器从主存贮器传送数据字到从属的 HI 中选择 TX 寄存器的外部地址作为目的地。TX 被写以后，数据由 HI 传送到 HRX 寄存器。若 TREQ 被置定，则设置 TXDE 使$\overline{HR}$确定。当 HRDF 被确定时，从属的 DMA 控制器开始从 HRX 传送数据到从属存贮器。

2. 用片上 DMA 控制器的数据读

此例略述以下步骤，DSP96002 主总线自 DSP96002 从属总线经过从属 HI 传送数据。两个 DSP96002 处理器的片上 DMA 控制器用来传送数据不妨碍两片自身的处理。图 18－18 表示用于数据传送的数据通路和控制线。

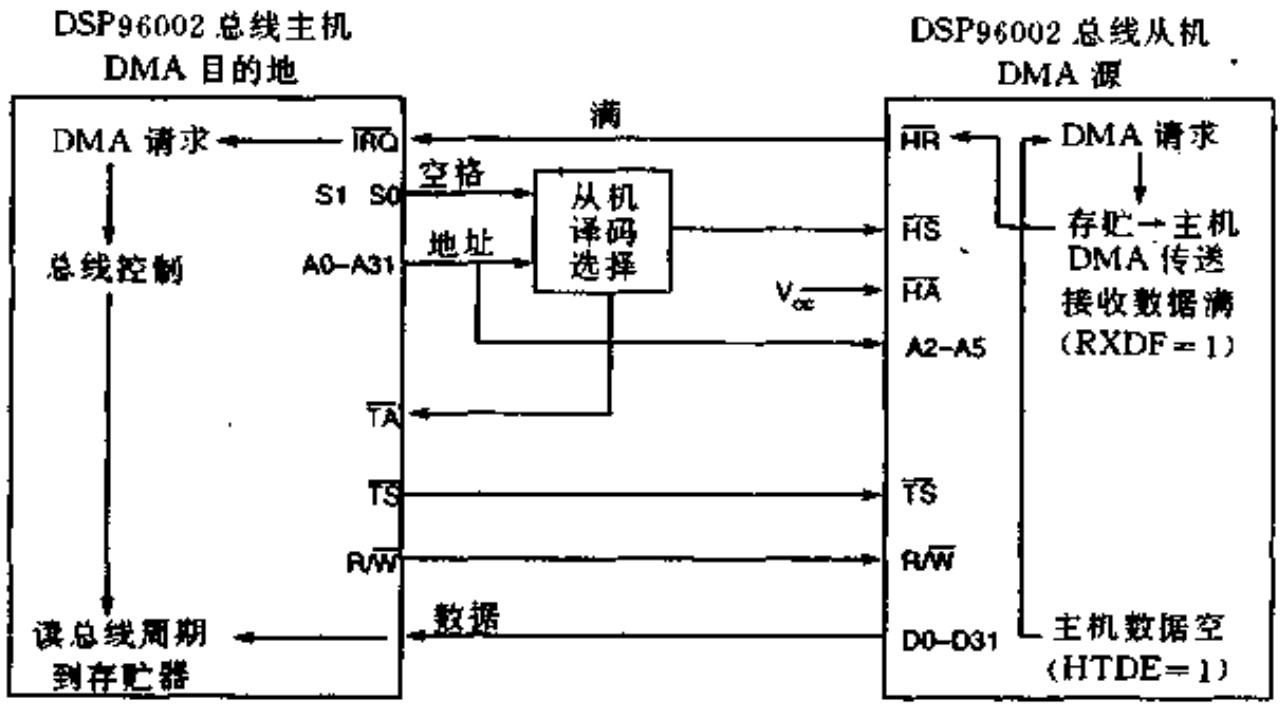


图 18－18　DSP96002 到 DSP96002 数据读

当从属的$\overline{HR}$信号确定时，指出其 HI RX 寄存器满并且数据已准备好，则开始数据读传送。$\overline{HR}$连接到主片中的$\overline{IRQ}$端，此端定义为 DMA 服务请求输入。当$\overline{HR}$确定时，主 DMA 控制器传送数据字由从属 HI 中选择 RX 寄存器的外部地址到主存贮器单元。读 RX 以后，设置 HTDE 和 RXDF 时，HI 可传送来自 HI HTX 寄存器的下一个数据字。若 RREQ 被置定，则设置 RXDF 使$\overline{HR}$确定。在从属的 DMA 控制器中，HTDE 定义为 DMA 服务请求信号。当 HTDE 确定时，从属 DMA 控制器开始由从属存贮器到 HTX 寄存器的数据传送，同时为进一步数据传送保持寄存器满。

18.4.20　外部 DMA 控制器到 DSP96002 数据传送——举例

本节举例说明在 DSP96002 和外部 DMA 控制器之间为传送数据，HI 和片上 DMA 控制器的使用。

外部 DMA 控制器是主总线，而 DSP96002 是从总线。外部 DMA 控制器访问 DSP96002 HI 不提供选择 HI 寄存器的地址。注意编程 HI 工作在 DMA 方式。

1. 数据写用 DSP96002 片上 DMA 控制器

此例勾画以下步骤：外部 DMA 控制器（主总线）经过从属 HI 传送数据到 DSP96002 从总线。DSP96002 的片上 DMA 控制器用于在 HI 和 DSP96002 存贮器之间局部传送数据，不妨碍核心处理。ICS 寄存器中的 TREQ 和 RREQ 位必须编程以规定数据传送方向为从外部 DMA 控制器到 HI（TREQ＝1，RREQ＝0）。ICS 寄存器中的 TYEQ 位应当清除。图 18－19

表示用于数据传送的数据通路和控制线。

当从属$\overline{HR}$信号确定时，表示其 HI TX 寄存器空并准备接收来自主机的数据字，则开始数据写传送。$\overline{HR}$连接到主机中 DMA 服务请求输入的$\overline{REQ}$端。当$\overline{HR}$确定时，外部 DMA 控制器传送数据字从存贮器到 HI 中 TX 寄存器。由确定$\overline{HA}$和 TREQ＝1 以及 RREQ＝0 写 TX 寄存器。写 TX 以后，设置 HRDF 和 TXDE，由 HI 传送数据到 HRX 寄存器。因为 TREQ 被置定，故设置 TXDE 使$\overline{HR}$确定。在从机的片上 DMA 控制器中，HRDF 规定为 DMA 服务请求信号。当 HRDF 置定时，从机片上 DMA 控制器开始数据传送从 HRX 到从机存贮器，完成数据传送。

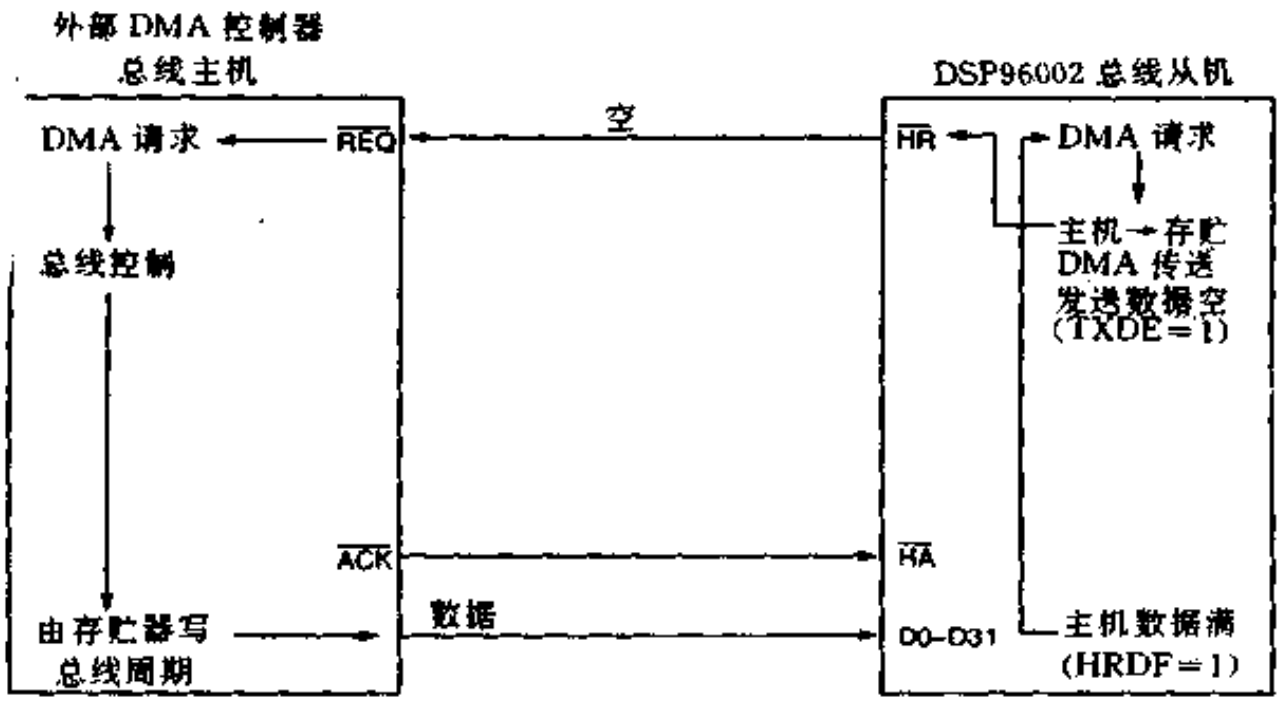


图 18－19　外部 DMA 到 DSP96002 数据写

2. 数据读用 DSP96002 片上 DMA 控制器

此例略述以下步骤：即外部 DMA 控制器（主总线）通过从机的 HI 传送来自 DSP96002 从机总线的数据。DSP96002 的片上 DMA 控制器用来在 HI 和 DSP96002 存贮器之间局部传送数据，不妨碍核心处理。ICS 寄存器中的 TREQ 和 RREQ 位必须编程以规定数据传送的方向为从 HI 到外部 DMA 控制器（TREQ＝0，RREQ＝1）。ICS 寄存器中的 TYEQ 位应当清除。图 18－20 表示用于数据传送的数据通路和控制线。

当从机$\overline{HR}$信号确定时，指出其 HI RX 寄存器是满并且数据准备好由外部 DMA 控制器读，则开始数据读传送。$\overline{HR}$连接到主机中 DMA 服务请求输入的$\overline{REQ}$端。当$\overline{HR}$确定时，外部 DMA 控制器传送数据字由从机 HI 中的 RX 寄存器到存贮器单元。由确定$\overline{HA}$和 TREQ＝0 以及 RREQ＝1 读 RX 寄存器。读 RX 以后，设置 HTDE 和 RXDF，HI 可传送来自 HI HTX 寄存器的下一个数据字。因为 RREQ 被置定，所以设置 RXDF 使$\overline{HR}$确定。在从机片上 DMA 控制器中，HTDE 规定为 DMA 服务请求信号。当 HTDE 确定时，从机片上 DMA 控制器开始数据传送由从机存贮器到 HTX 寄存器，为进一步数据传送保持寄存器满。

18.4.21　HI 性能分析和编程举例

下面主机编程举例表示支持两个 DSP96002 之间的主-从传送所需的软件。估计主处理器负载、最小传送周期和额外开销。这些估计可随寻址方式而变化。在很多情况下可用最快的寻址方式。它也假设主处理器在主机作用的中间不放掉总线。HI 寄存器由主处理器以 0 等待

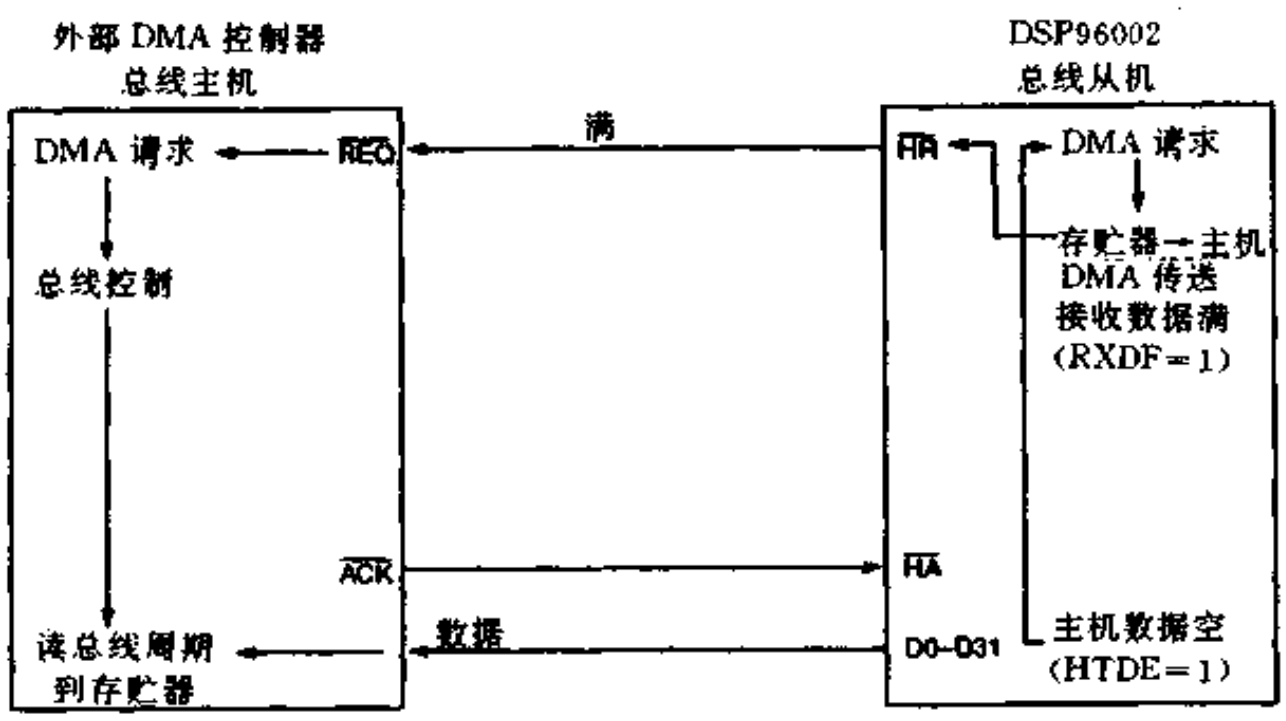


图 18-20　DSP96002 到外部 DMA 数据读

状态访问。

1. 信标控制

无论何时执行主机传送，主处理器必须首先得到从属 HI 的所有权。这由信标控制来完成。以下是由主处理器得到 HI 所有权编码的例子。SEM 寄存器的 LSB 用作信标位。

```
                                          字      时钟周期
SEMA    BEST      #0，Y：SEMR    1         4
        JCS       SEMA           1         4
        开始主机工作
              ⋮
        主机工作结束
        BCLR      #0，Y：SEMR    1         4
```

BEST 指令测试信标位，然后在释放总线前设置此位。若（1）当测试时此位已被置定，表明从机正由另一主机使用，于是此主机进入循环等待另一主机结束并清除信标位。若（2）当测试时此位是 0，从机可用，于是主机可继续访问从机。用 BEST 指令设置此位，通知其他主机从机现在不可用。完成主机工作以后，当前主机清除信标位以允许其他主机访问从机。一次主机传送的最小额外开销是 3 个程序字和 12 个时钟周期。当仅有一个总线主机时，不需要这个过程。

2. 主机命令寄存器读

此例中，主机和从机都是 DSP96002。HCVR 指出所选从机 CVR 寄存器（$\overline{HS}$=0、$\overline{HA}$=1，A5－A2=1101）的地址。主机执行以下指令：

```
MOVE    Y：HCVR，R0
```

3. 主机命令寄存器写

此例中，主机和从机都是 DSP96002。HCVR 指出从机 CVR 寄存器（$\overline{HS}$=0、$\overline{HA}$=1、A5－A2=1101）。开始主机命令前，推荐验证以前主机命令已被执行（HC 位已清除）。主机执行以下指令：

```
HCMD    BRSET    15，Y：HCDR，HCMD    ；testing of HC
        MOVE     R0，Y：HCVR
```

4. ICS 寄存器读

HICSR 指向从机 ICS 寄存器（$\overline{HS}$=0、$\overline{HA}$=1、A5－A2=1000）。主机执行以下指令

```
MOVE    R0, Y: HICSR
```

5. ICS 寄存器写

HICSR 指向从机 ICS 寄存器（$\overline{HS}$=0、$\overline{HA}$=1、A5－A2=1000）。主机执行以下指令。

```
MOVE    R0, Y: HICSR
```

6. 68K 中断确认序列

MC680×0 中断确认序列如下：

（1）当有暂挂中断时，68K 必须首先决定中断服务程序的起始单元。68K 以中断确认周期获得此信息。

（2）68K 中断控制器产生 IACK 信号，以响应连接到 96K $\overline{HA}$端的中断设备（此时为 96K）。

（3）中断设备把矢量数置于总线上以响应来自中断控制器的 IACK 信号。

图 18－21 表示 68K 中断确认序列流程图。

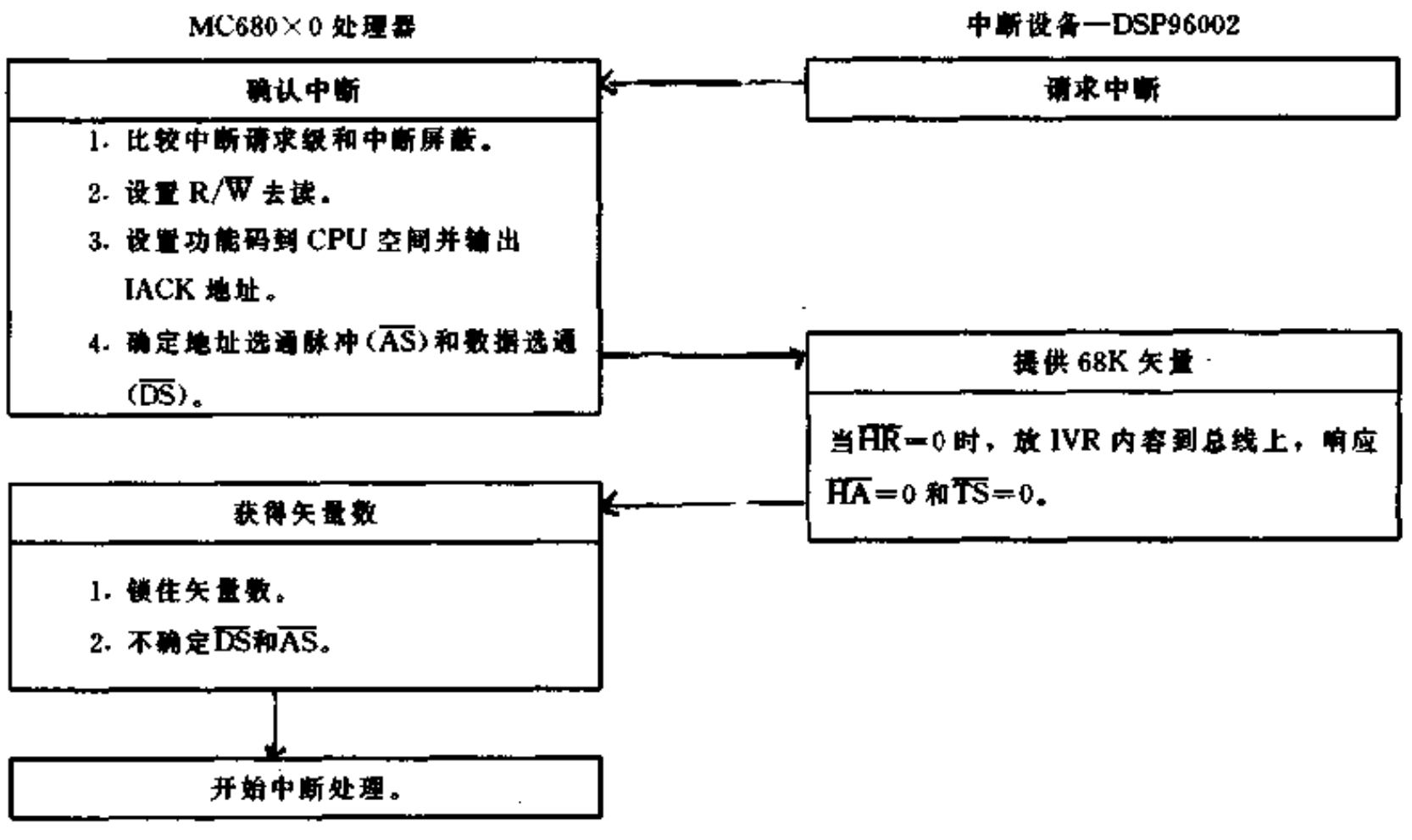


图 18－21　68K 中断确认序列

7. IVR 寄存器读

此例中，主机和从机是两个 DSP96002。HIVR 指向从机 IVR 寄存器（$\overline{HS}$=0、$\overline{HA}$=1、A5－A2=1100）。主机执行以下指令：

```
MOVE    Y: HIVR, R0
```

8. 68K 中断寄存器写

HIVR 指向从机 IVR 寄存器（$\overline{HS}$=0、$\overline{HA}$=1，A5－A2=1100）。主机执行以下指令：

```
MOVE    R0, Y: HIVR
```

9. X/Y/P 存贮器写过程

X/Y/P 存贮器写过程允许主处理器写数据字 D 到 DSP96002 存贮空间的仲裁地址 A。主

处理器必须执行以下步骤：

（1）验证 TX 是空（TXDE＝1）。

（2）用主机功能“TX 寄存器写和 X/Y/P 存贮器写中断”把 A 写入 TX 寄存器。若 HRX 是空，则 HI 自动地传送 A 到 HRX。

（3）验证 TX 是空（TXDE＝1）。

（4）用主机功能“TX 寄存器写和 X/Y/P 存贮器写中断”把 D 写入 TX 寄存器。第二次写开始 X/Y/P 存贮器写中断。

（5）在 DSP96002 一边，X/Y/P 存贮器写中断矢量应指向首先读 HRX 以得到地址 A 的程序，把 A 存在地址指针 Rn 中，然后再读 HRX 以取回数据 D 并存在由 Rn 指示的 DSP96002 存贮器单元中。

（6）主处理器可测试 TRDY，看 A 和 D 是否都从输入双缓存器（TX/HRX）中取出。

图 18－22　表示 X 存贮器写的流程图。

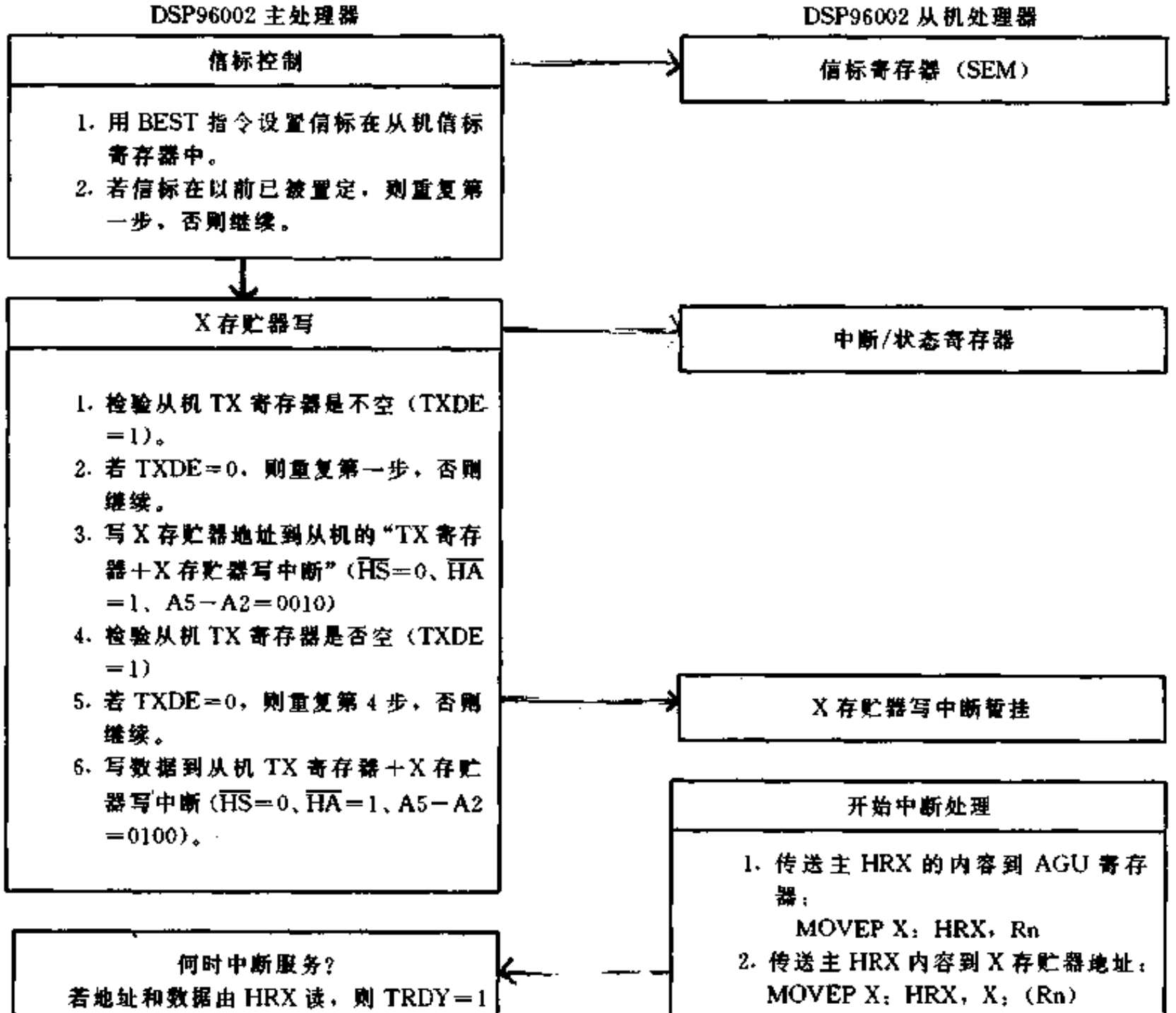


图 18－22　X 存贮器写过程

以下的码由主处理器执行。R3 寄存器包含选择在从机 HI 中“TX 寄存器写和 X 存贮器写中断”主机功能所需地址，如图 18－12 中定义。R4 寄存器包含读从机 HI 的 ICS 寄存器所需的地址。RI 寄存器包含目标 X 存贮器地址。R0 寄存器包含要写到目标 X 存贮器地址的数据。

主机执行以下指令：

			字	时钟周期
_LOOP1	JCLR	#TXDE，X：(R4)，_LOOP1	2	6
	MOVE	R1，X：(R3)	1	2
_LOOP2	JCLR	#TXDE，X：(R4)，_LOOP2	2	6
	MOVE	R0，X：(R3)	$\frac{1}{6}$	$\frac{2}{16}$

最小的存贮器写是 6 个程序字和 16 个时钟周期。第二次传送触发从机中 X 存贮器写中断请求。从机中的中断服务程序用 10～14 个时钟周期去执行。若有更高优先权的其他中断，则对此中断的响应可能延迟。

有一个较快的过程能消除上述的对 TXDE＝1 的测试，这就是要保证在写地址到 TX 以后写数据以前，已经过充分的时间。

			字数	时钟周期
_LOOP	JCLR	#TRDY，X：(R4)，_LOOP	2	6
	MOVE	R1，X：(R3)	1	2
	NOP		1	2
	MOVE	R0，X：(R3)	$\frac{1}{5}$	$\frac{2}{12}$

这一过程要求 5 个程序字和 12 个时钟周期。若主和从处理器用相同时钟频率和占空因数，则 NOP 指令在两个相继的 TX 写之间提供必需的通过时间，否则第二个 NOP 指令应加到以上的码中。

10. X/Y/P 存贮器读过程

X/Y/P 存贮器读过程允许主处理器读来自 DSP96002 存贮空间的任意地址 A 的数据字 D。主处理器必须执行以下步骤：

（1）验证 TX 是空（TXDE＝1）。

（2）用主机功能“TX 寄存器写和 X/Y/P 存贮器读中断”把 A 写入 TX 寄存器，这设置 HMRC。若 HRX 是空，则 HI 自动地传送 A 到 HRX 并开始 X/Y/P 存贮器读中断。

（3）在 DSP96002 一边，X/Y/P 存贮器读中断矢量应指向首先读 HRX 以得到地址 A 的程序，把 A 存在地址指针 Rn 中，读由 Rn 指示的存贮器单元，并用 HTXC 地址把数据 D 存入 HTX 寄存器。数据 D 通过 RX 寄存器（主处理器一边），HMRC 清除而 RXDF 置定。

（4）主处理器查询 ICS 寄存器直到 HMRC 清除，然后读来自 RX 寄存器的数据 D。

图 18－23 表示 X 存贮器读的流程图。

下面是由主处理器执行的码。R3 寄存器包含为选择从机 HI 中“TX 寄存器写和 X 存贮器读中断”主机功能所需的地址，如图 18－12 中定义的。R4 寄存器包含为读从机 HI 的 ICS 寄存器所需的地址。R1 寄存器包含目的 X 存贮器地址。R2 寄存器包含读从机 HI 的 RX 寄存器所需的地址。要求的数据字最后存在 DO. S 中。主机执行以下指令：

			字数	时钟周期
_LOOP1	JCLR	#TXDE，X：(R4)，_LOOP1	2	6
	MOVE	R1，X：(R3)	1	2
_LOOP2	JCLR	#HMRC，X：(R4)，_LOOP2	2	6
	MOVE	X：(R2)，DO.S	$\frac{1}{6}$	$\frac{2}{16}$

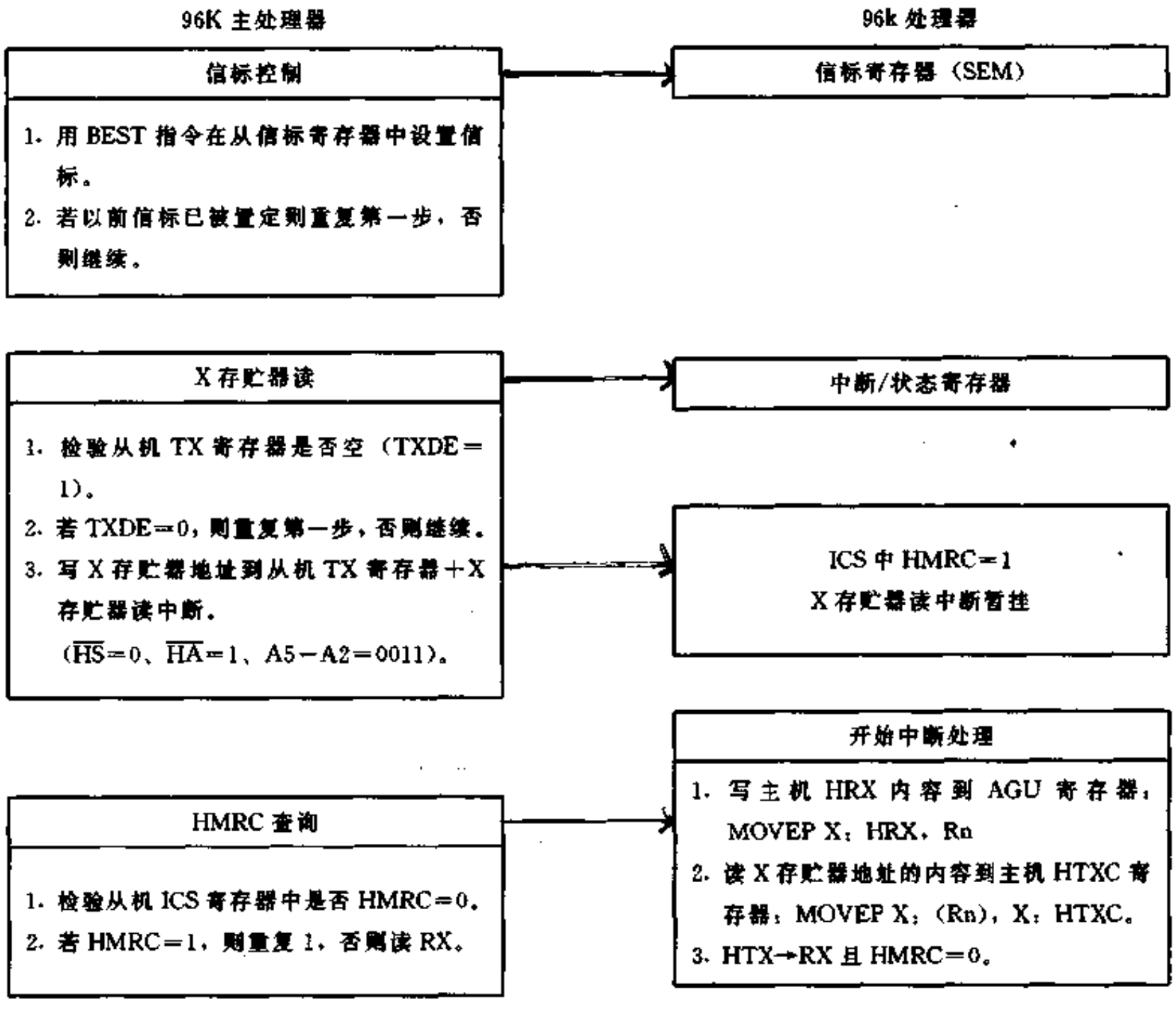


图 18-23 X 存贮器读过程

最小存贮器读的过程是 6 个程序字和 16 个时钟周期。第一次传送触发从机中 X 存贮器读中断请求。从机中的中断服务程序用 8～12 个时钟周期执行。只有这时主机才能以第二次传送继续读数据。

11. DSP96002 到 DSP96002 传送用片上 DMA 控制器

由片上 DMA 控制器进行数据传送不要求核心介入。因为 DMA 有专用内部数据通路和内部存贮器槽，不妨害核心处理的执行时间。但是，对片上 DMA 通路存在与初始化过程有关的额外开销。必须完成以下初始化步骤：

(1) 由读 DMA 通道控制/状态寄存器，主机验证从机 DMA 通道是空闲。这可用 X 存贮器读过程来完成。若 DMA 通道是为传送专用的，则这一步可以跳过。

(2) 主机初始化从机 DMA 通道。这可由 X 存贮器写过程或更有效地用预定义的主机命令完成。若要对预定义地址做数据块的重复 DMA 传送，则主机命令程序恰包含两条指令：允许 DMA 通道和装入 DMA 计数寄存器。

(3) 主机初始化它自身的 DMA 通道。

(4) 主机初始化从机 HI。

全部初始化处理可能用少于 12 周高到多于 100 周。例如，在 96K 线阵列中传送 DMA 块(传送数据仅一个方向到固定的预定义的地址)，初始化过程仅执行一次。每个 DMA 块传送对主和从处理器都只要求 DMA 使能（位设置）和 DMA 计数器装入。这用快中断可在 8～12 周内完成。

对以一个主机和 N 个从机配置的系统，初始化过程不是很长。若主机进行“恒定的”DMA 传送，则它可能有 N 个预定义中断，而每个从机 DMA 有固定的控制寄存器设置。在此情况下，完成初始化可少于 20 周。

18.5 DMA 控制器

18.5.1 引　　言

直接存贮器访问（DMA）控制器是片上设备，它允许任何存贮空间的组合中两个单元之间的数据传送，不介入 DSP96002 核心。由于专用 DMA 总线和双向访问内部存贮器，可实现高度的独立，即 DMA 操作不影响或减慢核心操作。DMA 控制器有两个通路，每个有它自己的寄存器组。DMA 控制器寄存器是在内部 I/O 存贮空间（在 X 存贮器中最高 128 单元）存贮器映射的读/写寄存器。

图 18－24 中的表格表示 DMA 控制器传送数据的能力。图中注释指定的周期数是为用连续块传送的一个通路的情况。

DMA 数据传送	注释
Int. mem←→Int. mem（不同存贮空间）	#1
Int. mem←→Int. mem（相同存贮空间）	#2
Ext. mem←→Int. mem（不同存贮空间）	#1
Ext. mem←→Int. mem（相同存贮空间）	#2
Ext. mem←→Ext. mem	#3
Int. mem←→Int. I/O（不同存贮空间）	#1
Int. mem←→Int. I/O（相同存贮空间）	#2
Ext. mem←→Int. I/O（不同存贮空间）	#1
Ext. mem←→Int. I/O（相同存贮空间）	#2
Int. I/O ←→Int. I/O	#2

注释：(1)每个字 2 个时钟周期。

(2)每个字 4 个时钟周期(对源和目的用相同地址总线)。

(3)每个字 4 个时钟周期。

图 18－24　DMA 数据传送方向

18.5.2 DMA 控制器编程模型

包含 DMA 控制器的寄存器示于图 18－25 和图 18－26 中。

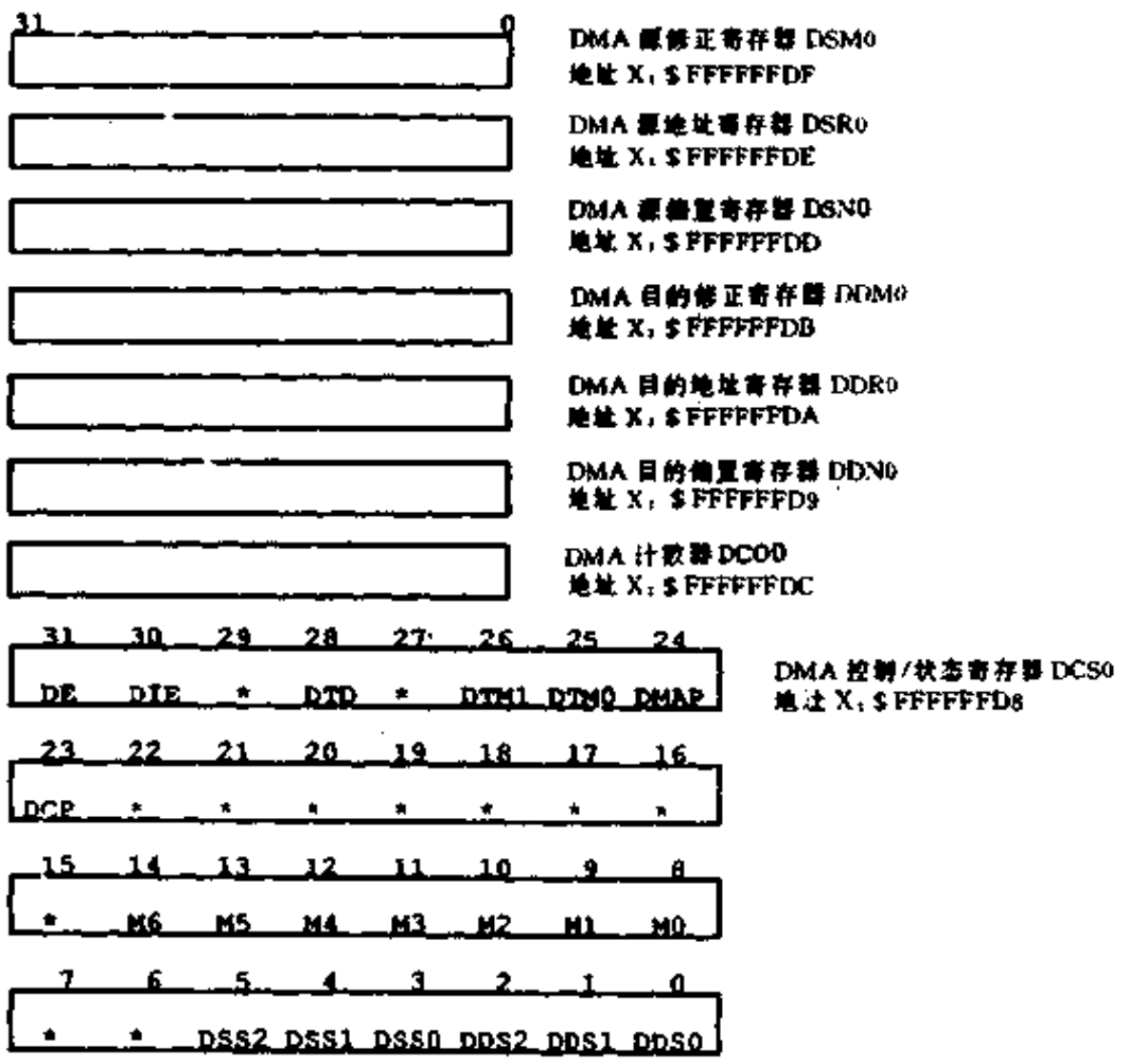


图 18－25　DMA 控制器编程模型——通路 0

18.5.3　DMA 控制/状态寄存器（DCS）

DCS 是控制 DMA 操作的 32 位读/写寄存器。以下各节说明每一位的作用。

1. DCS DMA 目的空间控制（DDS2～DDS0）位 0，1，2

DDS2～DDS0 指定作为目的将由 DMA 访问的存贮器或 I/O 空间。DDS2～DDS0 位由硬件和软件复位来清除。

DDS2	DDS1	DDS0	DMA 目的存贮空间
0	0	0	内部程序存贮器
0	0	1	内部 X 数据存贮器
0	1	0	内部 Y 数据存贮器
0	1	1	内部 I/O（X 存贮空间）
1	0	0	外部程序存贮器
1	0	1	外部 X 数据存贮器
1	1	0	外部 Y 数据存贮器
1	1	1	外部 I/O（Y 存贮空间）

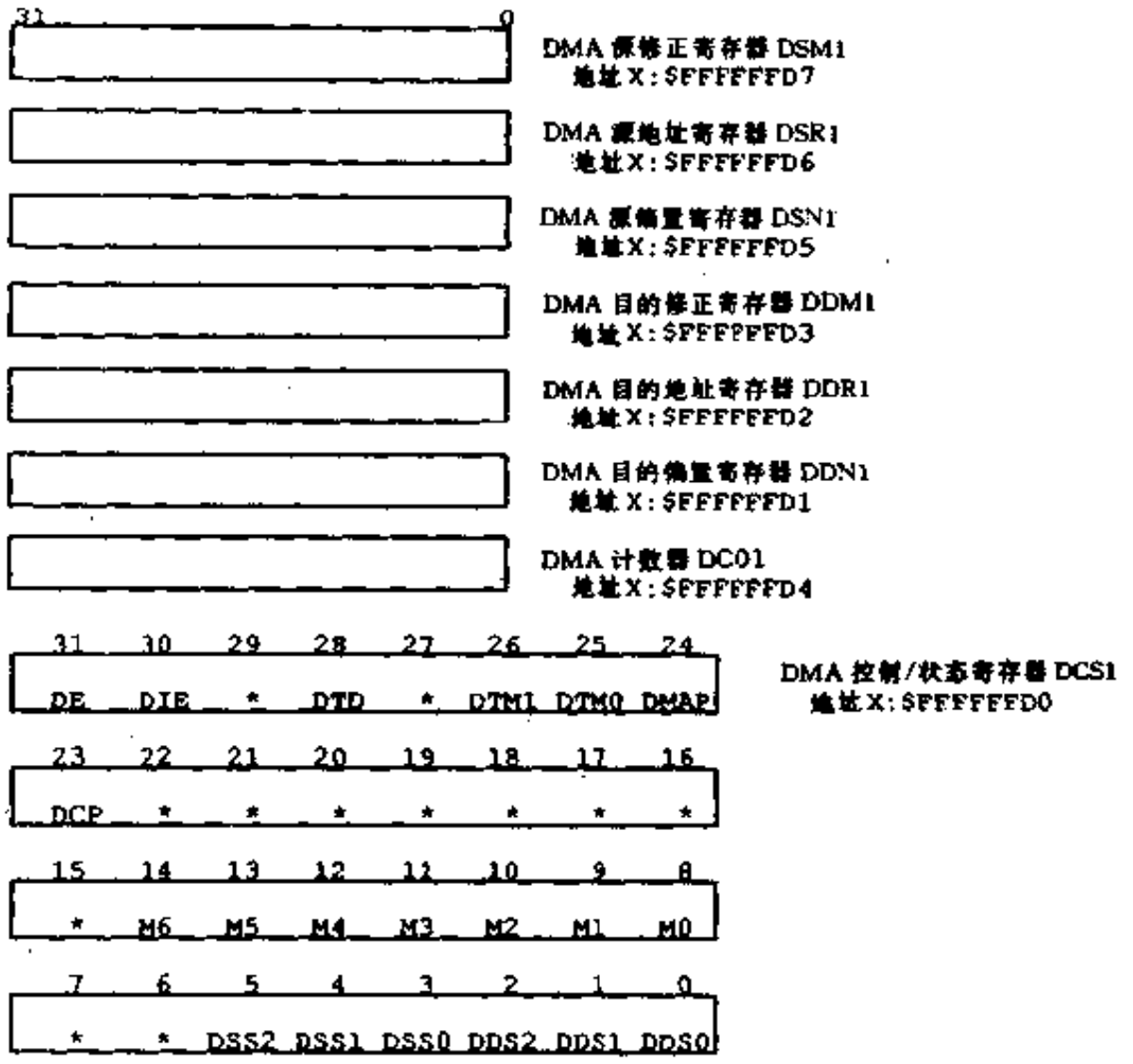


图 18-26 DMA 控制器编程模型——通路 1

2. DCS DMA 源空间控制（DSS2～DSS0）位 3，4，5

DSS2～DSS0 指定作为源将由 DMA 访问的存贮器或 I/O 空间。DSS2～DSS0 位由硬件和软件复位来清除。

DDS2	DDS1	DDS0	DMA 源存贮空间
0	0	0	内部程序存贮器
0	0	1	内部 X 数据存贮器
0	1	0	内部 Y 数据存贮器
0	1	1	内部 I/O（X 存贮空间）
1	0	0	外部程序存贮器
1	0	1	外部 X 数据存贮器
1	1	0	外部 Y 数据存贮器
1	1	1	外部 I/O（Y 存贮空间）

3. DCS 保留位（6，7，15～22，27，29）（略）

4. DCS DMA 请求屏蔽（M0～M6）位 8～14

DMA 请求屏蔽位选择用于启动 DMA 传送的 DMA 请求源。若屏蔽位被设置，则选择相应设备作为 DMA 请求源。若屏蔽位被清除，则此设备被忽略。DMA 请求源可能是经过$\overline{\text{IRQA}}$、$\overline{\text{IRQB}}$和$\overline{\text{IRQC}}$端请求服务的内部外围或外部设备。屏蔽位由硬件和软件复位来清除。内部 DMA 请求源由内部外围状态位和 DE 相“与”产生。

每个请求设备输入首先是单个地和它各自的屏蔽位（M0、M1 等）相“与”，然后所有“与”的输出一起进行“或”。“或”输出进行边缘触发闩锁，其输出启动 DMA 传送。若输入不被屏蔽，则确定输入将设置闩锁并启动 DMA 传送。当访问 DMA 源地址时 DMA 状态机清除闩锁。若允许更多请求设备，则任何输入的第一个边缘被门锁并启动 DMA 传送，闩锁被清除前出现的任何其他边缘都被忽略。

DMA 请求 屏蔽位请求设备	
M0	外部的（$\overline{\text{IRQA}}$端）
M1	外部的（$\overline{\text{IRQB}}$端）
M2	外部的（$\overline{\text{IRQC}}$端）
M3	端口 A 主机接收数据（HRDF＝1）
M4	端口 A 主机发送数据（HTXE＝1）
M5	端口 B 主机接收数据（HRDF＝1）
M6	端口 B 主机发送数据（HTXE＝1）

5. DCS DMA 通路优先级（DCP）位 23

DCP 位包含 DMA 通路相对其他 DMA 通路的优先级。当 DMA 传送暂挂时，两个通路的 DMA 通路优先级进行比较以决定哪个通路开启。因为两个通路用共同的资源如 DMA ALU 和地址总线，所以必须做此判决。DCP 由硬件和软件复位来清除。

若两通路有相同的优先级，则通路将以循环方式工作。通路 0 将启动传送单数据字，后面是通路 1 接上。

若通路优先级不同，最高优先级的通路将开始执行 DMA 传送，并且只要有 DMA 传送暂挂就继续进行。当较高优先级通路接收传送请求时，低优先级通路正执行 DMA 传送，则低优先级通路将做完当前数据字的传送并再一次进行仲裁。

DCP	DMA 通路优先级
0	优先级 0
1	优先级 1

6. DCS DMA 优先级（DMAP）位 24

当请求外总线访问时，此位允许设置相对于核心的优先级。在核心和 DMA 控制器之间竞争的情况下，优先级决定 DMA 是等待还是不等待。若 DMAP 被清除，则 DMA 等待直到外总线上有一空槽。若 DMAP 被置定，则核心周期将扩展并且核心和 DMA 在同一周期内将进行访问。DMAP 由硬件和软件复位来清除。

DMAP	外部访问优先级
0	核心
1	相等

7. DCS DMA 传送方式——（DTM1～DTM0）位 25，26

DTM1～DTM0 位指定 DMA 通路的工作方式。它们由硬件和软件复位来清除。

当 DTM1～DTM0＝00 时，传送单一数据块，块的长度由计数器决定，由设置 DE 位启动传送，当计数器递减到 0 时完成传送。

当 DTM1～DTM0＝01 时，传送单一数据块，块长度由计数器决定，传送由 DE 置 1 后的第一次 DMA 请求来启动，并且当计数器递减到 0 时完成传送。

当 DTM1～DTM0＝10 时，传送单一数据块，块长度由计数器决定，DE＝1 时每个 DMA 请求将传送一个字，当计数器递减到 0 时传送完成。

当 DTM1～DTM0＝11 时，DE＝1 时每次收到 DMA 请求传送一个字。此方式中计数器不用。

DTM1	DTM0	传送方式
0	0	单块、由 DE 位启动，DMA 请求无用
0	1	单块、由第一次 DMA 请求启动
1	0	单块、由 DMA 请求启动字传送
1	1	单字、由 DMA 请求启动

8. DCS DMA 传送进行状态（DTD）位 28

当最后的字在单块传送存入目的停止 DMA 操作时，只读 DMA 传送进行状态位设置。在同一时间，DE 将被清除。最后传送定义为 DMA 计数器达到 0，或者当 DE 位被核心清除时正在进行的传送。若 DIE 被置定（DMA 中断允许），则 DTD＝1 将形成 DMA 中断请求。当不允许 DMA 中断时（DIE＝0），中心可通过查询此位验证 DMA 状态。DTD 由硬件和软件复位来设置，由设置 DE 来清除。

9. DCS DMA 中断允许控制位（DIE）位 30

当 DMA 传送允许（DIE）位被置定时，DMA 中断发生在 DTD 被置定时，当 DIE 被清除时，不允许 DMA 中断。DIE 由硬件和软件复位清除。

DIE	DMA 中断
0	不允许
1	允许

10. DCS DMA 通路允许控制位（DE）位 31

DE 位允许 DMA 控制器工作。设置 DE 将清除 DTD。若 DTM1～DTM0＝00，设置 DE 将启动一块 DMA 传送。设置 DE 将允许以 DMA 方式传送，用请求设备作为触发。DE 由硬件和软件复位来清除，若选择单块传送方式，由 DMA 传送的结束来清除，在 DMA 工作时，只有在当前的 DMA 传送完之后，清除 DE，才能使 DMA 停止。

DE	DMA 工作
0	不允许
1	允许

18.5.4 DMA 计数器（DCO）

DMA 计数器是读/写 32 位寄存器，包含要完成的 DMA 数据传送数。若 DMA 通路设置为单块传送方式，则每次 DMA 数据传送后，DMA 计数器递减 1 并检测是否为 0。当计数器到达 0 时，进行 DMA 块传送而 DMA 通路将停止数据传送。若通路设置为单字方式（DTM1～DTM0=11），则因为每次 DMA 数据传送按要求进行，就不管 DMA 计数器的内容了。

当 DMA 通路工作在块传送方式之一时，DMA 计数器不应当写。仅当通路不允许（DE=0 和 DTD=1）或当用单字方式（DTM1～DTM0=11）时可以写 DMA 计数器。

18.5.5 DMA 地址寄存器（DSR 和 DDR）

DMA 源地址寄存器（DSR）和 DMA 目的地址寄存器（DDR）是二个 32 位寄存器，分别包含为下一次 DMA 传送的源和目的的地址。DMA 地址寄存器在功能上与地址产生单元地址寄存器相同。

18.5.6 DMA 偏置寄存器（DSN 和 DDN）

DMA 源偏置寄存器（DSN）和 DMA 目的偏置寄存器（DDN）是二个 32 位寄存器，它指定用于更新相应的 DMA 地址寄存器所用的偏置值。当读有关地址寄存器并用作输入给模运算单元时，读每个偏置寄存器。DMA 偏置寄存器在功能上与地址产生单元偏置寄存器相同。

18.5.7 DMA 修正寄存器（DSM 和 DDM）

DMA 源修正寄存器（DSM）和 DMA 目的修正寄存器（DDM）是 32 位寄存器，它指定在 DMA 地址寄存器更新计算中，用于更新相应的 DMA 地址寄存器的运算型式。当读有关地址寄存器并用作输入给模运算单元时读每个修正寄存器。DMA 修正寄存器在功能上与地址产生单元修正寄存器相同。当处理器复位或软件复位时，二个 DMA 修正寄存器都被置于 $FFFFFFFF（线性运算）。

18.5.8 DMA ALU

ALU 是对 DMA 和地址产生单元共用的，并且在它们之间是时间复用的。DMA ALU 是在（R）+N 结构中硬件实现的。因此装入 DMA 偏置寄存器用户可递增或递减 1 或 N。例如，用 DSP96002 字可寻址存贮器的 DMA 块传送经常以+1 装入 DMA 偏置寄存器。但是，插入、分样和交换操作可要求任意的地址偏置值 N。用字节可寻址存贮器的 DMA 块传送典型地以+4 装入偏置寄存器，以完成 32 位排队访问。DMA 传送到/从 I/O 外围可以 0 装入偏置寄存器以继续访问同一地址。

18.5.9 DMA 寻址方式

DMA 控制器可按与地址产生单元中的寄存器同样方法对地址计算和更新编程。DMA 修正寄存器与修正寄存器 M0～M7 完全相同。这样，DMA 源和/或目的地址寄存器可用线性、位倒置或模地址计算方法更新。

18.5.10 DMA 限制

以下是对 DMA 工作的一些限制：

(1) 源/目的地址区域必须整个是外部或内部的。DMA 不能处理部分内部和部分外部的数据块。这些块须按二个分立的块处理，一个内部的和一个外部的。

若源/目的地址区定义为内部的，并且一个地址大于由 DMA ALU 产生的最大内部地址，则地址将在内部地址空间中缠绕。

若源/目的地址区定义为外部的，并且一个地址小于由 DMA ALU 产生的最小外部地址，则地址总要访问外部存贮器。注意经常被核心认为是内部的 X 和 Y 数据存贮器单元可能作为外存贮器单元被 DMA 访问。

(2) WAIT 和 STOP 指令将停止 DMA 传送。STOP 和 WAIT 可能不允许内部时钟在 DMA 传送的中间。在执行 STOP 和 WAIT 指令以前，用户应停止 DMA 传送。为了停止 DMA 传送，DE 必须被清除。执行 STOP 或 WAIT 指令前，用户应查询 DTD 位（或当 DTD 置定时接收 DMA 中断）以保证当前 DMA 传送完成。

注意在多处理系统中这些指令的使用要求几种软件管理，因为外部设备没有办法知道片子进入 STOP 或 WAIT 状态。

(3) 当指定源或目的在内部 I/O 空间时，DMA 控制器仅可访问主机发送/接收数据寄存器。

(4) 当任何（内部或外部）读-修正-写核心访问时，DMA 不允许完成或启动任何 DMA 传送。DMA 停止如同它试图去访问外总线而且它不是主总线。

(5) DMA 受影响的情况：

① 若核心通过两个端口同时访问外存贮器，并且一个或二个核心由于存贮器等待使访问延迟，则内部 DMA 传送将延迟，因为芯片时钟产生等待状态，冻结内部动作。

② 若核心正进行一次外部访问，而 DMA 通过另一端口也进行一次外部访问，并且 DMA 访问延迟了（如由于等待状态），则在另一端口由核心的访问不受影响。DMA 有独立的等待机理，此时核心继续正常执行，因为核心时钟不进入等待状态。

③ 若核心作一次外部访问而 DMA 通过另一端口也作一次外部访问，并且核心访问延迟，则在另一端口 DMA 的访问也延迟。这是因为芯片时钟产生等待状态且整个片子停止。若核心冻结，则 DMA 和核心之间的仲裁不能继续。

④ 若 DMA 通路之一正通过一端口访问外存贮器，而且因总线仲裁或存贮器等待使访问延迟，则第二路 DMA 也将停止，因为 DMA 的机理不能对两路区别。

⑤ 若数据 ALU 正执行要求归一化周期的浮点指令，则数据 ALU 可能对包括 DMA 的其他芯片冻结时钟。此时 DMA 操作将变慢。

18.6 I/O 存贮器映像

内部 I/O 外围占据 X 存贮空间顶端 128 个单元。外部 I/O 外围占据 Y 存贮空间顶端 128 个单元。图 18-27 表示对内部 I/O 外围的 I/O 存贮器映像。

X 数据存贮器空间

$FFFFFFFF	IPR - 中断优先权寄存器
$FFFFFFFE	BCRA - 端口 A 总线控制寄存器
$FFFFFFFD	BCRB - 端口 B 总线控制寄存器
$FFFFFFFC	PSR - 端口选择寄存器
	保留
$FFFFFFF0	为 OnCE 操作保留(OGDBR)
$FFFFFFEF	HTXA/HRXA - 主机 A HTX/HRX 寄存器
$FFFFFFEE	HTXCA - 主机 A HTX 寄存器和 HMRC 清除
$FFFFFFED	HSRA - 主机 A 状态寄存器
$FFFFFFEC	HCRA - 主机 A 控制寄存器
	保留
$FFFFFFE7	HTXB/HRXB - 主机 B HTX/HRX 寄存器
$FFFFFFE6	HTXCB - 主机 B HTX 寄存器和 HMRC 清除
$FFFFFFE5	HSRB - 主机 B 状态寄存器
$FFFFFFE4	HCRB - 主机 B 控制寄存器
	保留
$FFFFFFE0	保留
$FFFFFFDF	DSM0 -DMA CH0 源修正寄存器
$FFFFFFDE	DSR0 -DMA CH0 源地址寄存器
$FFFFFFDD	DSN0 -DMA CH0 源偏置寄存器
$FFFFFFDC	DCO0 -DMA CH0 计数器寄存器
$FFFFFFDB	DDM0 -DMA CH0 目的修正寄存器
$FFFFFFDA	DDR0 -DMA CH0 目的地址寄存器
$FFFFFFD9	DDN0 -DMA CH0 目的偏置寄存器
$FFFFFFD8	DCS0 -DMA CH0 控制/状态寄存器
$FFFFFFD7	DSM1 -DMA CH1 源修正寄存器
$FFFFFFD6	DSR1 -DMA CH1 源地址寄存器
$FFFFFFD5	DSN1 -DMA CH1 源偏置寄存器
$FFFFFFD4	DCO1 -DMA CH1 计数器寄存器
$FFFFFFD3	DDM1 -DMA CH1 目的修正寄存器
$FFFFFFD2	DDR1 -DMA CH1 目的地址寄存器
$FFFFFFD1	DDN1 -DMA CH1 目的偏置寄存器
$FFFFFFD0	DCS1 -DMA CH1 控制/状态寄存器
$FFFFFFCF	保留
	保留
$FFFFFF80	保留

图 18 - 27　内部 I/O 存贮器映像

第十九章　异常处理

19.1 引　　言

本章叙述除正常处理有关指令执行以外的DSP96002的动作。阐述了异常情况时DSP96002所引起的一系列动作。也阐述了处理器和中断源的优先级（IPL）。

19.2 处理状态

DSP96002通常处于5种处理状态之一：正常、异常、复位、等待或停止。正常处理状态就是有关指令执行。

19.2.1 异常处理状态

异常处理状态与中断有关。异常处理可能由软件中断指令、由片上外围硬件中断或由错误在内部产生。而外部异常处理可由中断产生，异常处理为服务于I/O的设备提供有益的前后转换。

19.2.2 复位处理状态

为响应已确定的外部$\overline{\text{RESET}}$端而进入复位处理状态。在进入复位状态时发生以下动作：

- 内部外围设备复位和不允许。
- 修正寄存器置于$FFFFFFFF。
- 中断优先级寄存器（IPR）被清除。
- 除MR寄存器中11和10外，所有CCR、ER、IER和MR位被清除。
- MR寄存器中的中断屏蔽位11，10被置定。

DSP96002保持复位状态直到$\overline{\text{RESET}}$不确定。一离开复位状态，就由外部方式选择端（MODA、MODB、MODC）装入操作方法寄存器的芯片操作方式位，并在第二十章所说的单元开始执行程序。

19.2.3 等待处理状态

等待处理状态由于执行WAIT指令而进入低功耗方式。在等待方式中，除内部外围（中断控制器和主机接口）不允许来自所有内部电路的内部时钟。所有内部处理停止直到发生任何非屏蔽中断，DS96002复位或$\overline{\text{DR}}$被确定。若因为确定DR而由等待状态退出，则处理器可能进入调试方式。

19.2.4 停止处理状态

停止处理状态是最低功耗方式并随执行STOP指令而进入此方式。在停止方式中，时钟

振荡器被禁止，与等待状态不同，那时时钟振荡器仍有效。处理器中所有活动都停止直到发生以下动作之一：

（1）给$\overline{\text{IRQA}}$端加低电平（$\overline{\text{IRQA}}$确定）。

（2）给$\overline{\text{RESET}}$端加低电平（$\overline{\text{RRESET}}$确定）。

（3）给$\overline{\text{DR}}$端加低电平。

这些动作中任一个将恢复振荡器，经稳定性延迟后，时钟对处理器和外围重新有效。

当时钟对处理器和外围有效时，若由于$\overline{\text{RESET}}$端上加低电平而退出停止状态，则处理器进入复位处理状态。若由于$\overline{\text{IRQA}}$端上加低电平而退出停止状态，则处理器将服务于最高优先级的暂挂中断。若没有中断暂挂（即在中断仲裁以前$\overline{\text{IRQA}}$不确定），则处理器恢复执行使之进入停止状态的停止指令。

若由$\overline{\text{DR}}$端加低电平使之退出停止状态，则处理器进入调试方式。

19.3 异 常 处 理

在数字信号处理环境中，异常处理主要与 DSP96002 存贮器之间或寄存器和外围设备之间的数据传送相联系。当中断发生时，必须以最小的额外开销完成有限的前后关系转换。

当收到硬件中断时，它同步在指令边界上，因而前两个中断指令字可能插入指令流。在当前指令的取周期假设中断存在中断暂挂闩锁寄存器中。当下一周期时，是现行指令的解码周期，PC 机将更新去取下一条指令。但是，在以后的现行指令执行周期中，位于程序地址总线（PAB）的地址来自适当的中断起始地址，而不是来自 PC 机。注意 PC 机是冻结的，直到异常处理终止。

图 19-1 说明中断控制器的影响，控制器把二个指令字插入处理器指令流。

中断控制 CYC1	i						i*				
中断控制 CYC2		i						i			
提　取	n3	n4	ii1	ii2	n5	n6	n7	n8	ii3	ii4	
译　码	n2	n3	n4	ii1	ii2	n5	n6	n7	n8	ii3	ii4
执　行	n1	n2	n3	n4	ii1	ii2	n5	n6	n7	n8	ii3
i＝中断；　ii＝中断指令字；　n＝正常单字指令；　*＝此时允许连续中断											

图 19-1　中断流水线操作

在第一个中断指令字（图 19-1 中 n4 或 n8）提取之前的周期中提取单字指令时，以下单字指令异常终止：Bcc、BRA、BScc、BCR、FBcc、FBScc、FJcc、FJScc、Jcc、JMP、JScc、JSR、LRA、REP、RESET、RTI、RTR、RTS、STOP 和 WAIT。

当第一个中断指令字的提取代替双字指令（图 19-2 中 n5）的第二个字的提取时，异常终止双字指令。

当程序控制由中断程序返回时，异常终止指令再重新提取。调整 PC 机适当先于异常终止指令译码周期的结束。

中断控制 CYC1	i						i*				
中断控制 CYC2		i									
提　取	n3	n4	ii1	ii2	n4	n5	n6	n7	ii3	ii4	n8
译　码	n2	n3	n4	f1	f2	n4	— —	n6	n7	f3	f4
执　行	n1	n2	n3	NOP	f1	f2	n4	— —	n6	n7	f3

f＝快中断指令字（不控制流改变）　　n4＝2－字指令

i＝中断　　n5＝n4 的第 2 字

ii＝中断指令字　　* 此时允许相随中断

n＝正常单字指令

图 19－2　双字指令快中断失败举例

若第一个中断字提取发生在单字指令（不是以上所列的）提取或双字指令的第二个以后的周期中，则该指令将正常地先于中断程序开始而完成。

以下情况已经验证，中断服务可能产生额外延迟：

(1) 若长的中断程序用于 (F) TRAPcc 中断，则处理器优先级置于 3。因此，除非法指令和堆栈错误外，不允许所有中断直到 (F) TRAPcc 服务程序 RTI 终止（除非 (F) TRAPcc 服务程序软件使处理器优先级下降）。

(2) 当正服务中断时，下一中断将按以下规则延迟：

第一个中断指令字到达指令译码器之后，在译下面第一个中断指令字以前将至少译 4 条以上指令。若任何一对正计数的指令是由要重复的指令跟随的 REP 指令，则整个“程序包”计为两条指令，与已做的重复数无关。

(3) 以下指令是不可中断的：ILLEGAL、(F) TRAPcc、STOP、WAIT 和 RESET。

(4) REP 指令和正在重复的指令不可中断。

19.3.1　中断指令提取

当中断指令提取时，由中断起始地址和 中断起始地址 +1 单元提取指令字。

中断控制器产生中断指令提取地址，它指出双字快中断程序的第一个指令字，这地址用于下一条指令提取，而不是用于 PC 机，以及中断指令提取地址+1 用于相继的指令提取。当中断指令正被提取时，PC 机禁止更新。两个中断字提取后，PC 机用于任何后面的指令提取。

二个中断指令字都被提取后，保证它们是要执行的。这是确实的，即使正在执行的指令是流指令（即 JMP、JSR 等）的改变。这指令正常地忽略流水线中的指令。中断指令被提取后，若中断指令没有被代替，PC 机将指出已经提取的指令。

19.3.2　中断指令执行

两种中断程序可以使用：快和长。快程序仅由两个自动插入的中断指令 字组成。除了一些特殊限制外，这些字可包括一个双字指令或二个单字指令。若没有中断服务程序指令使流指令改变，则中断指令执行认为是快的。在快中断程序内跳到子程序形成长中断。终止长中断程序是以 RTI 指令使 PC 和 SR 由堆栈恢复以及返回正常程序执行，硬件复位是一种特殊的异常，它一般仅在异常起始地址含有一个跳指令。

1. 快中断指令执行

快中断程序执行常常有以下规则：

(1) 当快中断程序时状态不保存，所以应当不用修改状态指令。

(2) 快 中断程序从来不可中断。

(3) 快中断程序可包括任何一个双字指令或二个单字指令。

(4) 若快程序中指令之一跳到子程序，则形成长中断程序。

(5) 当快中断程序 时 PC 机从不更新。

(6) 完成快中断程序后正常指令提取恢复 PC 机的使用。

图 19-3 说明快中断程序对指令流水线的作用。

中断控制 CYC1	i						i*				
中断控制 CYC2		i									
提　取	n3	n4	ii1	ii2	n5	n6	n7	n8	ii3	ii4	n9
译　码	n2	n3	n4	f1	f2	n5	n6	n7	— —	f3	f4
执　行	n1	n2	n3	n4	f1	f2	n5	n6	n7	— —	f3
n＝正常指令字（其他同图 19-2）											

图 19-3　连续矢量间 4 指令情况举例

2. 长中断指令执行

在快中断程序内跳到子程序指令形成长中断程序。单字或双字跳到子程序的指令可用于形成长中断程序。单字跳到子程序可能置于第一或第二个中断矢量单元。若条件单字跳到 子程序置于第一个中断矢量单元，若跳条件是真，则第二个矢量单元的指令将被忽略，若跳条件是假则执行指令。若单字跳到子程序置于第二个中断矢量单元，则在执行跳到子程序以前提取和执行第一个矢量单元中的指令。长中断程序的执行常有以下规则：

(1) 在跳到子程序的指令被执行期间，当它发生在第一或第二个中断矢量单元时，产生以下动作：

① 将 PC 和 SR 的内容推入堆栈。

② 按下述修改状态寄存器 (SR)；更新 MR 中的中断屏蔽位 11，10 以屏蔽相同或更低优先级的中断（除不合法指令，堆栈错误和 (F) TRAPCC 总是可中断以外）。

③ JSR 指令将改变 PC 机，因此指令随着由 JSR 指令指出的地址中的指令继续执行。

(2) 更高优先级中断可中断长中断程序。下一中断服务的第一个指令字，只有在跟随前面中断的第一指令译码之后，至少有 4 条指令译码，才能到 达译码器。

(3) 长中断指令应由 RTI 终止，RTI 把 PC 和 SR 从堆栈中拉出。

图 19-4 示出长中断程序对指令流水线的影响。快 JSR（即单字 JSR 指令）用来形成长中断程序。对此例，长中断程序的字 4 是 RTI。其后的中断说明在长中断程序中前面指令的不可中断性质。

图 19-5 是当收到 内部服务请求的指令是 REP 指令（图 19-5 中 n3）时中断服务的例子。当跟随 REP 指令 (N4) 的指令重复执行时，指令提取停止。这种提取指令仅在循环计数器递减到 1 之后才重新开始。

中断控制 CYC1	i							
中断控制 CYC2		i						
提　取	n3	n4	ii1	ii2	sr1	sr2	sr3	sr4
译　码	n2	n3	n4	JSRf	NOP	sr1	sr2	sr3
执　行	n1	n2	n3	n4	JSRf	NOP	sr1	sr2

中断控制 CYC1	i*							
中断控制 CYC2		i						
提　取	sr5	n5	ii3	ii4	n6	n7	n8	n9
译　码	RTI	NOP	n5	ii3	ii4	n6	n7	n8
执　行	sr3	RTI	NOP	n5	ii3	ii4	n6	n7

i＝中断　　n＝正常指令字

ii＝中断指令字　　sr＝服务程序字

JSRf＝快速 JSR（单字 JSR 指令）　　*＝在此时允许其后的中断

图 19－4　长中断流水线动作

中断控制 CYC1	i				i*						
中断控制 CYC2		i				i					
提　取	n3	n4	n5			n6	ii1	ii2	n7	n8	n9
译　码	n2	REP	NOP	n4	n4	n5	n6	ii1	ii2	n7	n8
执　行	n1	n2	REP	NOP	n4	n4	n5	n6	ii1	ii2	n7

n3＝REP＃2 指令　　＋＝在此时拒绝中断

n4＝指令重复两次　　*＝此时重新允许中断　　（其他同图 19－4）

n5＝在备份指令存贮器中等待的指令

图 19－5　当中断交给 REP 指令时中断服务举例

当执行 n4 时，中断不工作，当 LC 最后达到 1 时，重新开始取指令并且中断可以工作。在图 19－5 中可以看到，在第一个中断矢量前，译码和执行 n5（由备份指令存贮装入指令存贮）以及 n6。

19.4　中　断　源

异常可能由一些中断源产生。图 19－6 给出了 DSP96002 中断源，并表示了每个中断源相应的起始地址。中断起始地址是内部产生的指示快速中断服务程序起始地址的 32 位地址，每个中断的中断起始地址为了最小开销是地址常数。Motorola 预定 128 个中断起始地址单元，而 128 个地址是为用户使用预定的。这些单元占据程序存贮空间最低的 512 个字，为硬件复位的也可占据程序存贮器地址的高区。若有些空间没有用，则可用于程序存贮。

中断起始地址	中断源	中断起始地址	中断源
$FFFFFFFE	Hardware RESET	$00000026	Host A Read Y Memory
$00000000	Hardware RESET	$00000028	Host A Read P Memory
$00000002	Stack Error	$0000002A	Host A Write X Memory
$00000004	Illegal Instruction	$0000002C	Host A Write Y Memory
$00000006	(F) TRAPcc (defaut)	$0000002E	Host A Write P Memory
$00000008	IRQA	$00000030	Host B Receive Data
$0000000A	IRQB	$00000032	Host B Transmit Data
$0000000C	IRQC	$00000034	Host B Read X Memory
$0000000E	Reserved	$00000036	Host B Read Y Memory
$00000010	DMA Channel1	$00000038	Host B Read P Memory
$00000012	DMA Channel2	$0000003A	Host B Write X Memory
$00000014	Reserved	$0000003C	Host B Write Y Memory
$00000016	Reserved	$0000003E	Host B Write P Memory
$00000018	Reserved	$00000040	Reserved
$0000001A	Reserved	⋮	⋮
$0000001C	Host A Command (default)	$000000FE	Reserved
$0000001E	Host B Command (default)	$00000100	User interrupt vector
$00000020	Host A Receive Data	⋮	⋮
$00000022	Host A Transmit Data	$000001FE	User interrupt vector
$00000024	Host A Read X Memory		

注：用户中断矢量单元用于主机命令。

图 19－6　DSP96002 中断源

19.4.1　内部外围中断源

内部外围中断源包括所有的片上外围设备（主机和 DMA）。每个内部中断源是对电平敏感的：即每个中断工作在出现的任何时候并不屏蔽中断。每个内部硬件源有独立的使能控制。

19.4.2　硬件复位

硬件 RESET 中断是对电平敏感的，并是最高优先级 3 中断。它由确定$\overline{\text{RESET}}$端而引起。

19.4.3　外部中断请求 IRQA、IRQB 和 IRQC

IRQA、IRQB 和 IRQC 中断可编程使其对电平敏感或边界敏感，电平敏感的中断当它们工作时不是内部存贮并且不是自动清除；它们必须用其他方法清除以防止多中断。存贮边界敏感的中断当中断工作时如同暂挂在中断输人的高到低过渡处并自动清除。IRQA、IRQB 和 IRQC 可对三种优先级（0、1 或 2 级）之一编程，它们都是可屏蔽的。此外，每个中断有独立的使 能控制。当 IRQA、IRQB 和 IRQC 不允许在中断优先级寄存器中时，将删除暂挂请求，不接收新的请求，并且边界检测锁存仍保留在复位状态。若中断定义为电平敏感的，它的边界检测锁存也保留在复位状态。

在第一个矢量单元提取指令字开始的中断服务，认为是在提取第二个矢量单位指令字时结束。在边界敏感中断情况下，当提取第二个矢量单元量内部锁存自动被清除。第一个矢量单元的提取不保证第二矢量单元被提取。图 19－7 中说明第二矢量单元不被提取的情况。图 19－7，(F) TRAPcc 指令“删除”第一中断矢量的提取以保证提取 (F) TRAPcc 矢量。当 iil

正被提取时译码指令 n4 作为（F）TRAPcc。执行（F）TRAPcc 要求 ii1 删除并且代之以提取两个（F）TRAPcc 矢量（ii3 和 ii4）。

中断控制 CYC1	i				i*						
中断控制 CYC2		i				i					
提　取	n3	n4	ii1				ii3	ii4	tr1	tr2	tr3
译　码	n2	n3	trap	— —	— —	— —	— —	JSR	— —	tr1	tr2
执　行	n1	n2	n3	trap	— —	— —	— —	— —	JSR	— —	tr1

i=中断请求
i* =由(F)TRAPcc 产生的中断请求
ii1=中断 i 的第一矢量
ii3=第一个(F)TRAPcc 矢量(单字 JSR)
ii4=第二个(F)TRAPcc 矢量
n=正常指令字
n4=(F)TRAPcc,cc 条件真
tr=属于(F)TRAPcc 长中断程序的指令

图 19-7　(F)TRAPcc 指令拒绝另一个中断

19.4.4　（F）TRAPCC（有条件的软件中断指令）

(F) TRAPcc 指令形成不可屏蔽中断。若指定的条件是真。立即按照（F）TRAPcc 指令工作。(F) TRAPCC 是优先级 3 的中断。

19.4.5　非法指令中断

非法指令中断是不可屏蔽中断，它按照非法指令中断指令立即工作或在指令锁存器中装入非法指令。非法指令中断是优先级 3 的中断。

19.4.6　堆栈错误中断

堆栈错误中断是优先级 3 的中断。它由堆栈指针寄存器中堆栈错误标志产生，通常由于不正确的堆栈操作引起，堆栈错误标志保持置定直到它由某些写入 SP 寄存器的指令所清除。因为这个中断优先级（3）不屏蔽同一级的其他暂挂中断，建议堆栈错误标志由堆栈错误中断服务程序的第一条指令清除，为了不再得到相同的请求。

19.5　中断优先级结构

共有 4 个中断优先级。编号 0，1 和 2 的中断优先级（IPL）是可屏蔽的。0 级是最低级，3 级是最高级，是不可屏蔽的。级 3 中断是堆栈错误、复位、非法指令和（F）TRAPcc。中断屏蔽位（11、10）在状态寄存器中反映当前处理器优先级，并指出了中断源去中断处理器所需的中断最低优先级。图 19-8 给出了中断屏蔽位的描述。对所有优先级低于当前处理器优先级的中断都被禁止。级 3 中断总能中断处理器。

19.5.1　中断优先级（IPL）

对每个片上外围设备（主机 DMA）和每个外部中断源（IRQA、IRQB 和 IRQC），中断优先级都可在软件控制下编程。每个片上或外部外围设备都可对三个可屏蔽中断级（0、1 或

2）之一编程。中断优先级由写入中断优先级寄存器来设置。

I1	I0	异常允许	异常屏蔽
0	0	IPL 0，1，2，3	None
0	1	IPL 1，2，3	IPL.0
1	0	IPL 2，3	IPL 0，1
1	1	IPL 3	IPL 0，1，2

图 19－8　状态寄存器状态屏蔽位

19.5.2　中断优先权寄存器（IPR）

读/写寄存器对每个中断设备（主机、DMA、IRQA、IRQB、IRQC）指定中断优先级。此外，此寄存器指定每个外部中断源的触发方式和表示外部中断请求的状态。寄存器由硬件复位或 RESET 指令来清除。中断优先级寄存器示于图 19－9。图 19－10 定义了中断优先级的位。图 19－11 定义外部中断触发方式位和状态信息。

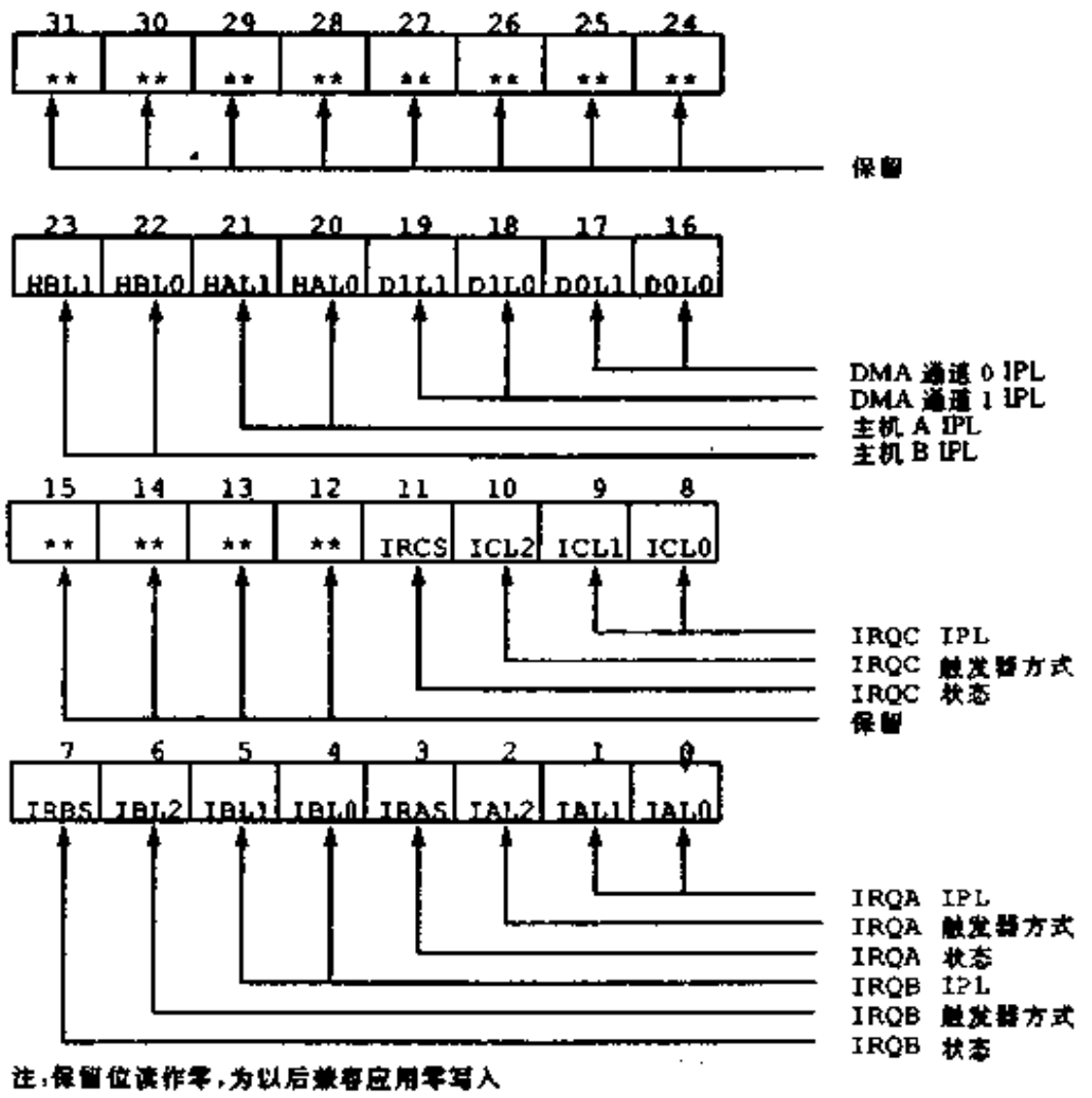


图 19－9　中断优先级寄存器（地址 X：FFFFFFF）

xxL1	xxL0	使能	内部优先权级（IPL）
0	0	n0	—
0	1	yes	0
1	0	yes	1
1	1	yes	2

图 19－10　中断优先级位

IxL2	触发方式
0	电平
1	负边缘

IRxS	状态
0	服务
1	悬挂

图 19-11　外部中断触发方式和状态

1. IRQA 中断优先级——IAL1—IAL0（位 0～1）

IRQA 中断优先级（IAL1～IAL0）位用来对外部中断输入 IRQA 的使能和指定优先级。

IAL1	IAL0	使能	IPL
0	0	否	—
0	1	是	0
1	0	是	1
1	1	是	2

2. IRQA 触发方式——IAL2（位 2）

IRQA 触发方式（IAL2）位指定外部中断输入 IRQA 的触发方式。

IAL2	触发方式
0	电平
1	负边缘

3. IRQA 状态——IRAS（位 3）

只读 IRQA（IRAS）位指出外部中断输入 IRQA 中断请求的状态。若 IRQA 中断定义为边界敏感的并是允许的。则 IRAS 位指出边界检测锁存的状态。若 IRQA 中断定义为电平敏感的或是不允许的，则 IRAS 位指出内同步后 IRQA 端的状态。

IRAS	状态（边界和允许）	IRQA 端（电平或不允许）
0	工作	高
1	暂挂	低

4. IRQB 中断优先级——IBL1～IBL0（位 4～5）

IRQB 中断优先级（IBL1～IBL0）位用来对外部中断输入 IRQB 的使能和指定优先级。

IBL1	IBL0	使能	IPL
0	0	否	—
0	1	是	0
1	0	是	1
1	1	是	2

5. IRQB 触发方式——IBL2（位 6）

IRQB 触发方式（IBL2）位对外部中断输入 IRQB 指定触发方法。

IBL2	触发方式
0	电　平
1	负边缘

6. IRQB 状态——IRBS（位 7）

只读的 IRQB 状态（IRBS）位指出对外部中断输入 IRQB 中断请求的状态。若 IRQB 中断定义为边界敏感的并被允许，则 IRBS 位指出边界检测锁存的状态。若 IRQB 中断

IRBS	状态（边界和允许）	IRAB 端（电平或不允许）
0	工作	高
1	暂挂	低

定义为电平敏感的或不被允许，则 IRBS 位指出在内同步以后 IRQB 端的状态。

7. IRQC 中断优先级——ICL1～ICL0（位 8～9）

IRQC 中断优先级（ICL1～ICL0）位用于外部中断输入 IRQC 的允许和指定优先级。

ICL1	ICL0	允许	IPL
0	0	否	—
0	1	是	0
1	0	是	1
1	1	是	2

8. IRQC 触发方式——ICL2（位 10）

IRQC 触发方式（ICL2）位对外中断输入 IRQC 指定触发方法。

ICL2	触发方式
0	电　平
1	负边（界）

9. IRQC 状态——IRCS（位 11）

只读 IRQC 状态（IRCS）位为外中断输入 IRQC 指出中断请求状态。若 IRQC 中断定义为边界敏感的且被允许，则 IRCS 位指出边界检测锁存的状态。若 IRQC 定义为电平敏感的或不被允许，则 IRCS 位指出内同步后 IRQC 端的状态。

IRCS	状态（边缘和允许）	IRQC 端（电平或不允许）
0	工作	高
1	暂挂	低

10. 保留位——（12～15，24～31）（略）

11. DMA 通道 0 中断优先级——DOL1～DOL0（位 16～17）

DMA 通道 0 中断优先级（DOL1～DOL0）位用于 DMA 通道 0 中断的允许和指定优先级。

DOL1	DOL0	允许	LPL
0	0	否	—
0	1	是	0
1	0	是	1
1	1	是	2

12. DMA 通道 1 中断优先级——DIL1～DIL0（位 18～19）

DIL1～DIL0 位用于 DMA 通道 1 中断的允许和指定其优先级。

DIL1	DIL0	允许	IPL
0	0	否	—
0	1	是	0
1	0	是	1
1	1	是	2

13. 主机 A 中断优先级——HAL1～HAL0（位 20～21）

HAL1～HAL0 位用于对端口 A 主机接口中所有中断源的允许和指定其优先级。

HAL1	HAL0	允许	IPL
0	0	否	—
0	1	是	0
1	0	是	1
1	1	是	2

14. 主机 B 中断优先级——HBL1～HBL0（位 22～23）

HBL1～HBL0 位用于对端口 A 主机接口中所有中断源的允许和指定其优先级。

HBL1	HBL0	允许	IPL
0	0	否	—
0	1	是	0
1	0	是	1
1	1	是	2

19.5.3 在 IPL 中的异常优先级

若当执行一条指令时多于一个异常处于暂挂，则有最高优先级的中断首先工作 。在给定的中断优先级以内，第二优先级决定当具有相同 IPL 的多中断请求暂挂时，哪个中断工作。图 19－12 给出相等 IPL 中断的优先级。还给出所有中断的中断允许位。

优先级	异　常	允许来自	优先级	异　常	允许来自
最高	硬件复位			主机 A 写 P 存贮器中断	(HCR)HPWE
	非法指令			主机 A 传送数据中断	(HCR)HTIE
	堆栈错误			主机 B 命令中断	(HCR)HCIE
	(F)TRAPcc			主机 B 收数据中断	(HCR)HRIE
	IRQA(外中断)	(IPR)IAL1－IAL0		主机 B 读 X 存贮器中断	(HCR)HXRE
	IRQC(外中断)	(IPR)IBL1－IBL0		主机 B 读 Y 存贮器中断	(HCR)HYRE
	IRQC(外中断)	(IPR)ICL1－ICL0		主机 B 读 P 存贮器中断	(HCR)HPRE
	主机 A 命令中断	(HCR)HCIE		主机 B 写 X 存贮器中断	(HCR)HWE
	主机 A 收数据中断	(HCR)HRIE		主机 B 写 Y 存贮器中断	(HCR)HYWE
	主机 A 读 X 存贮器中断	(HCR)HXRE		主机 B 写 P 存贮器中断	(HCR)HPWE
	主机 A 读 Y 存贮器中断	(HCR)HYRE		主机 B 传送数据中断	(HCR)HTIE
	主机 A 读 P 存贮器中断	(HCR)HPRE		DMA 通道 0 中断	(DCSO)DIEO
	主机 A 写 X 存贮器中断	(HCR)HXWE			
	主机 A 写 Y 存贮器中断	(HCR)HYWE	最低	DMA 通道 1 中断	(DCS1)DIEO

图 19－12　在 IPL 内 DSP96002 的异常优先级

第二十章　芯片操作方式和存贮器映像

20.1　操作方式和程序存贮器映像

OMR 寄存器中的操作方式位 MA、MB 和 MC 决定程序存贮器的总线扩展方式和 DSP96002 脱离 RESET 状态时的起始过程。OMR 中的数据 ROM 使能位决定数据存贮器的总线扩展方式。

MODA、MODB 和 MODC 端用于装入有 DSP96002 的起始操作方式的 MA、MB 和 MC。这些引出端按 DSP96002 脱离 RESET 状态时抽样，以后这些端不影响操作方式，并可用于其它功能。芯片操作方式以在操作方式寄存器中写入操作方式位 MA、MB 和 MC 来编程。参考 15.12 节关于操作方式寄存器 OMR 的描述。图 20－1 表示方式的分配。

有 8 种芯片操作方式分为两组：

- 不引导方式——这些方式用于访问已编程的程序存贮器。
- 引导方式——这些方式用于装入在 RAM 中实现的内部程序存贮器。装入内部程序存贮器之后，DSP96002 转换到方式 0 或 1，但在位于片 上程序 存贮地址 $00000000 的地址开始执行。

方式	MC	MB	MA	DSP96002 起始的芯片操作方式
0	0	0	0	PRAM 允许，在 $FFFFFFFE 复位（端口 A）
1	0	0	1	PRAM 允许，在 $FFFFFFFE 复位（端口 B）
2	0	1	0	PRAM 不允许，在 $00000000 复位（端口 A）
3	0	1	1	PRAM 不允许，在 $00000000 复位（端口 B）
4	1	0	0	由字节引导（位 D7～D0）外部存贮器在 $FFFF0000（端口 A）
5	1	0	1	由字节引导（位 D7～D0）外部存贮器在 $FFFF0000（端口 B）
6	1	1	0	通过主机接口引导　　（端口 A）
7	1	1	1	通过主机接口引导　　（端口 B）

图 20－1　DSP96002 起始的芯片操作方式综合

20.1.1　方式 0（内部 PRAM 允许，复位在 $FFFFFFFE，端口 A）

在方式 0 中，内部程序存贮器占据程序存贮空间的较低部分。高于最高内部程序 存贮器单元的地址指向外部程序 存贮器。硬件复位矢量的地址是 $FFFFFFFE，位于端口 A 外部程序 存贮空间。此方式的程序存贮器映像示于图 20－2。

20.1.2　方式 1（内部 PRAM 允许，复位在 $FFFFFFFE，端口 B）

在方式 1 中，内部程序 存贮器占据程序存贮空间的较低部分。高于最高内部程序 存贮器

单元的地址指向外部程序存贮器。硬件复位矢量的地址是$FFFFFFFE，位于端口B外部程序 存贮空间。此方式的程序存贮器映像示于图20-2。

20.1.3 方式2（内部PRAM允许，复位在$00000000，端口A）

在方式2中，内部程序存贮器是不允许的。所有对程序存贮空间的访问是指向外部程序存贮器。硬件复位矢量的地址是$00000000，位于端口A外部程序 存贮空间。此方式的程序存贮器映像示于图20-2。

20.1.4 方式3（内部PRAM不允许，复位在$00000000，端口B）

在方式3中，内部程序存贮器是不允许的。所有对程序存贮空间的访问都指向外部程序存贮器。硬件复位矢量的地址是$00000000，位于端口B外部程序存贮空间。此方式的程序存贮器映像示于图20-2。

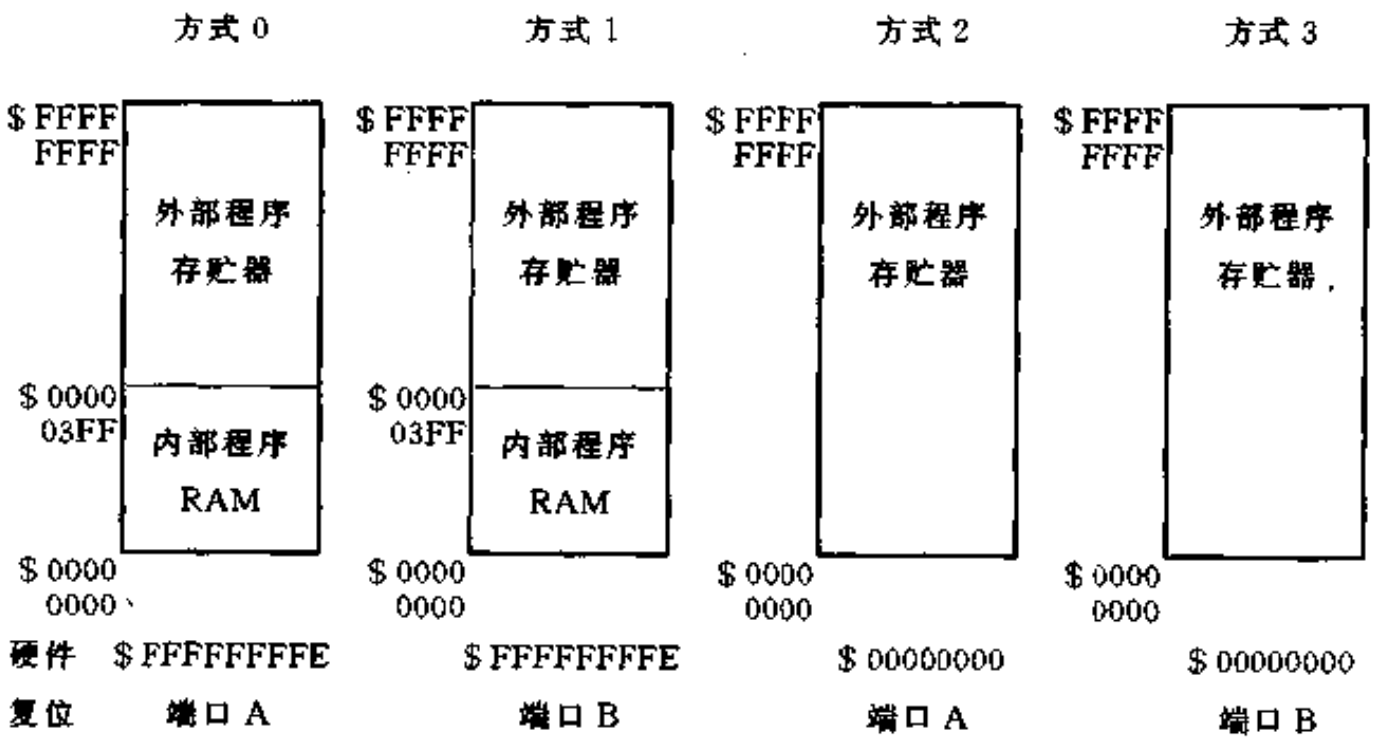


图20-2 DSP96002程序存贮器映像

20.1.5 方式4～7（引导方式）

引导方式由外部源装入内部程序存贮器，按照OMR中MA和MB位的值选择源的形式和单元。装入内部程序 存贮器后，DSP96002在位于片上程序存贮器地址$00000000的地址上开始程序 的执行。

由执行引导程序完成引导，程序在用户不可见的引导程序ROM中，在引导期间ROM映射到程序存贮空间。

当片子在引导方式之一中退出复位状态时发生以下动作：

(1) 片上硬件由32位、用户不可见的ROM映射64字到内部DSP96002程序存贮空间，它开始于$00000000单元。

(2) 在引导装入期间片上硬件使内部程序RAM只写。

(3) 程序执行开始于内部引导ROM的$00000000单元。图20-3列出DSP96002引导程序。

```
; 若 MB：MA＝0X——由 4096 连续的字节 P：存贮器单元
;（在 P：$FFFF0000 开始）装入。
; 若 MB：MA＝10——通过端口 A 中主机接口装入内部 PRAM，
; 若 MB：MA＝11——通过端口 B 中主机接口装入内部 PRAM。
BOOT        EQU         $FFFF0000     ; 此单元在 P：存贮器中
M_HCRA      EQU         $FFFFFFEC     ; 端口 A 主机控制寄存器
M_HSRA      EQU         $FFFFFFED     ; 端口 A 主机状态寄存器
M_HRXA      EQU         $FFFFFFEF     ; 端口 A 主机接收数据寄存器
M_HCRB      EQU         $FFFFFFE4     ; 端口 B 主机控制寄存器
M_HSRB      EQU         $FFFFFFE5     ; 端口 B 主机状态寄存器
M_HRXB      EQU         $FFFFFFE7     ; 端口 B 主机接收数据寄存器
            ORG         PL：$0        ; 引导码开始在 P：$0
START       MOVE        #BOOT，R1     ; R1＝外部 P：引导字节 ROM 的地址
            MOVEI       #0，R0        ; R0＝开始 P：程序开始装入的
                                      ; 内部存贮器的地址
; 若由改变 OMR 到引导方式进入此程序，则应肯定寄存器 M0 和 M1 已置于 $FFFFFFFF。使 BCR 寄存器确实置
; 于 $××××××F× 因为 EPROM 是慢的，使端口选择寄存器设置得允许程序存贮器通过要求的扩展端口
;（A 或 B）访问。
; 第一个程序由外部 P 存贮空间装入 4096 字节在 P：$FFFF0000（位 7～0）开始。这些将聚集到 1024 个 32 位字
; 中并存入由 P：$0 开始的连接的内部 PRAM 存贮单元。注意第一个程序首先装入由 P：$0 的最低有效字节
; 开始的数据。
; 端口选择寄存器不由此程序设置。它由 HW 复位设置。
; 第二个程序装入用主机接口逻辑的内部 PRAM。若 HF1＝0，将由外部主处理器装入 4096 字节。这些将聚集到
; 1024 个 32 位字并存入从 P：$0 开始的连续内部 PRAM 存贮单元。
; 若 HF1＝1，将从外部主处理器装入 1024 个 32 位字。
; 若主处理器只企图装入 P 存贮器的一部分，并开始执行已装入的程序，则主机接口引导程序可能由设置 HF0
; ＝0 而无效。
INLOOP      DO          #1024，_LOOPI         ; 装入 1024 个指令字，这是文本转换
            JSET        #，OMR，_HOSTLD       ; 若 MB＝1，从主机接口完成装入
; 这是第一个程序，由外部 P 存贮器装入。
            DO          #4，_LOOP2            ; 使 4 字节进入 D0.L
            LSR         #8，D0                ; 下移前面字节
            MOVEM       P：(R1)＋，D1，L      ; 从外部 P 存贮器得字节
            LSL         #24，D1               ; 移入高字节
            OR          D1，D0                ; 衔接
_LOOP2      JMP         <_STORE               ; 把字放入 P 存贮器
; 这是第二个程序，通过主机接口装入。
_HOSTLD     JSET        #0，OMR，_HOSTB       ; 端口 A 或 B?
; 通过端口 A 中主机接口引导
_HOSTA      BCLR        #5，X：M_HCRA         ; 使能端口 A 主机接口
            MOVE        #M_HSRA，R2           ; R2 指出 HSRA
            MOVE        #M_HRXA，R3           ; R3 指出 HRXA
            JMP         <_HOSTR               ; 进入主机程序
```

图 20－3（a）　DSP96002 引导程序汇编程序源

```
; 通过端口 B 中主机接口引导
  _HOSTB     BCLR   #5, X: M_HCRB        ; 使能端口 B 主机接口
             MOVE   #M_HSRB, R2          ; R2 指出 HSRB
             MOVE   #M_HRXB, R3          ; R3 指出 HRXB

; 主机装入程序
  _HOSTR
  _LBL11     JCLR   #3, : (R2), _LBL22   ; 若 HF0=1, 停止装数据
             ENDDO                       ; 必须终止 DO 循环
             JMP    <_BOOTEND
  _LBL22     JCLR   #0, X: (R2), _LBL11  ; 等待 HRDF 进入高 (表示数据存在)
             JCLR   #4, X: (R2), _LBL33  ; 8 位源?
             MOVE   X: (R3), D0L         ; 从主机得到 32 位字
             JMP    <_STORE
  _LBL33     DO     #4, _LOOP4           ; 使 4 字节进入 DO.L
             LSR    #8, D0               ; 下移前面的字节
  _LBL1      JCLR   #3, X: (R2), _LBL2   ; 若 HF0=1, 停止装数据
             ENDDO                       ; 必须终止 DO 环
             ENDDO
             JMP    <_BOOTEND
  _LBL2      JCLR   #0, X: (R2), _LBL1   ; 等待 HRDF 进入高
             MOVE   X: (R3), D1.L        ; 从主机得到字节
             LSL    #24, D1              ; 移入高字节
             OR     D1, D0               ; 衔接
  _LOOP4
  _STORE     MOVEM D0.L, P: (R0) +       ; 存 32 位结果在 P 存贮器
  _LOOP                                  ; 得到另外 32 位字
; 这是退出控 制以返回内部 PRAM 执行
  _BOOTEND   ANDI   #$F9, 0MR            ; 置操作方式于 00x (并由引导方式触发退出)
             ANDI   #$1, CCR             ; 清除 CCR 如 HW 复位。也为操作方式改变而必要的延
                                         ; 迟
             JMP    <$1                  ; 从 PRAM 开始取数
; DSP96002 引导程序长=50 字
```

图 20-3 (b)　DSP96002 引导程序汇编程序源

(4) 引导程序读 OMR 位 MA 和 MB 以决定所选的引导方式。

在方式 4 中，引导程序通过端口 A 从 $FFFF0000 开始的 4 096 个连续的字节外部程序存贮单元装入内部程序 RAM。

在方式 5 中，引导程序通过端口 B 从 $FFFF0000 开始的 4 096 个连续的字节外部程序存贮单元装入内部程序 RAM。

在方式 6 中，引导程序通过端口 A 中主机接口从外部主处理器装入内部程序 RAM。若主机接口标志 HF1 被清除，则引导程序假设外部主处理器是 8 位源，它将供给多达 4 096 字节。若主机接口标志 HF1 被置定，则引导程序假设外部主处理器是 32 位源，它将供给多达 1 024 个 32 位字装入程序 RAM。外部主处理器可以设置主机接口标志 HF0 来终止引导程序。

在方式 7 中，引导程序通过端口 B 中主机接口从外部主处理器装入内部程序 RAM。若主机接口标志 HF1 被清除，则引导程序假设外部主处理器是 8 位源，它提供多达 4 096 字节 。若主机接口标志 HF1 被置定，则引导程序假设外部主处理器是 32 位源，它提供 1 024 个 32 位字装入程序 RAM。外部主处理器可由设置主机接口标志 HF0 来终止引导程序。

(5) 由写入 OMR 以进入方式 0 或 1。这动作将开始有定时的延迟来从程序存贮器 映像去掉引导程序 ROM。

(6) 这种定时延迟准确地定时使引导程序执行一次 NOP，然后跳到单元 $ 00000000 并开始执行用户程序。

用户也可以写 入 OMR 来选择引导方式。此方法允许 DSP96002 程序员重新引导他的系统。用户可从任何操作方式编程 OMR 到所要求的引导方式。这时开始定时延迟以便将引导 ROM 映射到程序地址空间。这定时延迟准确地定时以使程序员执行一次 NOP 然后跳到 $ 00000000 单元的引导 ROM，并开始上述 1～6 步的引导过程。

20.2 数据存贮器映像

数据存贮器映像示于图 20－4 和图 20－5。

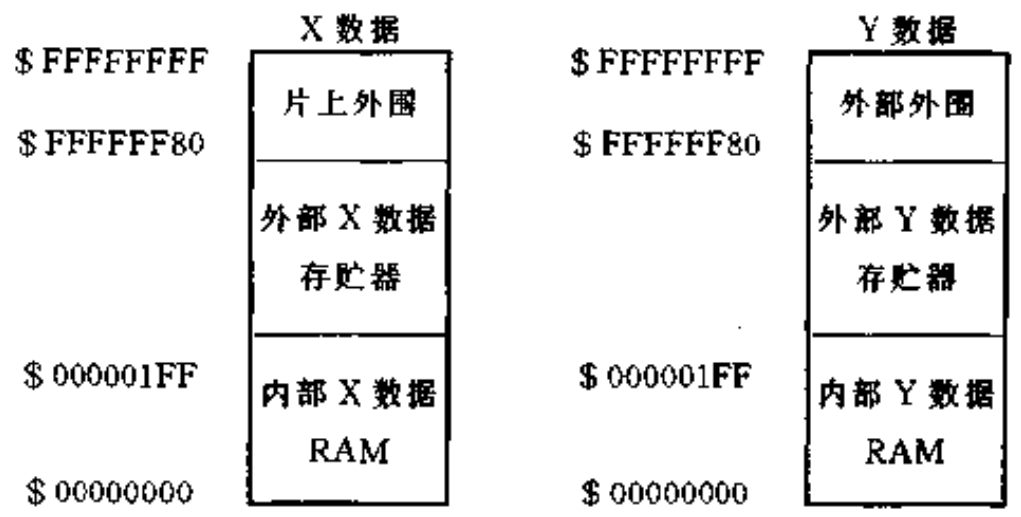


图 20－4 DE＝0 时 DSP96002 数据存贮器映像

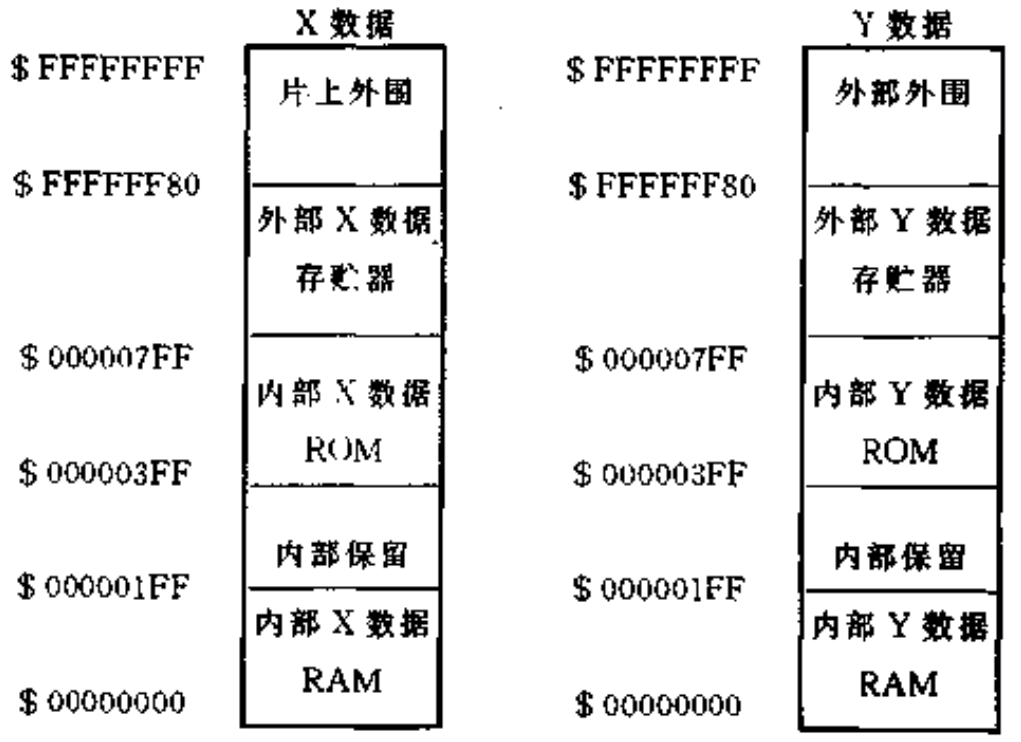


图 20－5 DE＝1 时 DSP96002 数据存贮器映像

20.2.1 内部数据 RAM

片上 X 和 Y 数据 RAM 分别占据 X 和 Y 数据存贮器映像中 $00000000～$000001FF 单元，并且它们总是被允许的。

20.2.2 内部数据 ROM

X 和 Y 数据存贮器扩展方式受位于 OMR 中的 DE 位的影响。当由设置操作方式寄存器中 DE＝1 允许时，片上 X 和 Y 数据 ROM 分别占据 X 和 Y 数据存贮器中 $00000400～$000007FF 单元。若 DE＝0，则片上数据 ROM 不允许，并且它们以前占据的地址范围现在是外部数据寄存器。

X 和 Y 数据 ROM 每个占 1 024 个单元。X 数据 ROM 包含一周余弦值 而 Y 数据 ROM 含一周正弦值。正弦和余弦值用最接近方式舍入到 IEEE 单精度浮点的 MC68881 浮点协处理器产生。

当内部数据 ROM 允许时（DE＝1），地址范围 $00000200～$000003FF 中的 X 和 Y 数据存贮单元定义为内部的。为便于将来扩展，这地址范围不用并保留，当内部数据 ROM 不允许（DE＝0）时，地址范围 $00000200～$000003FF 定义为外部的。

第二十一章　片上仿真器

21.1　概　　述

系统开发的一般方法由驻留在DSP程序存贮器中的程序组成。使用片上资源或附加程序存贮器地址的接口电路与主计算机或终端相联。这种方法是不透明的，会加载DSP总线并且有时会干扰用户系统结构。为在用户目标系统中仿真DSP，必须用电缆以将DSP引出端连到要开发的系统上。

DSP96002片上仿真(OnCE™)①电路提供与DSP96002和它的外围相互作用的方法以使用户可以考察寄存器、存贮器或片上外围。这样使得很容易在DSP96002处理器上进行硬件/软件开发。为此，定义DS96002印模上的特殊电路和专用引出端，以避免牺牲任何用户对片上资源的可访问性。

OnCE™专用引出端的关键是允许用户把DSP96002插入目标系统还保持调试控制。对引出仿真系统上DSP96002引出端所要求的昂贵电缆不需要了，因为对专用OnCE™调试串行端口易于访问。图21-1说明OnCE™串行接口的框图。

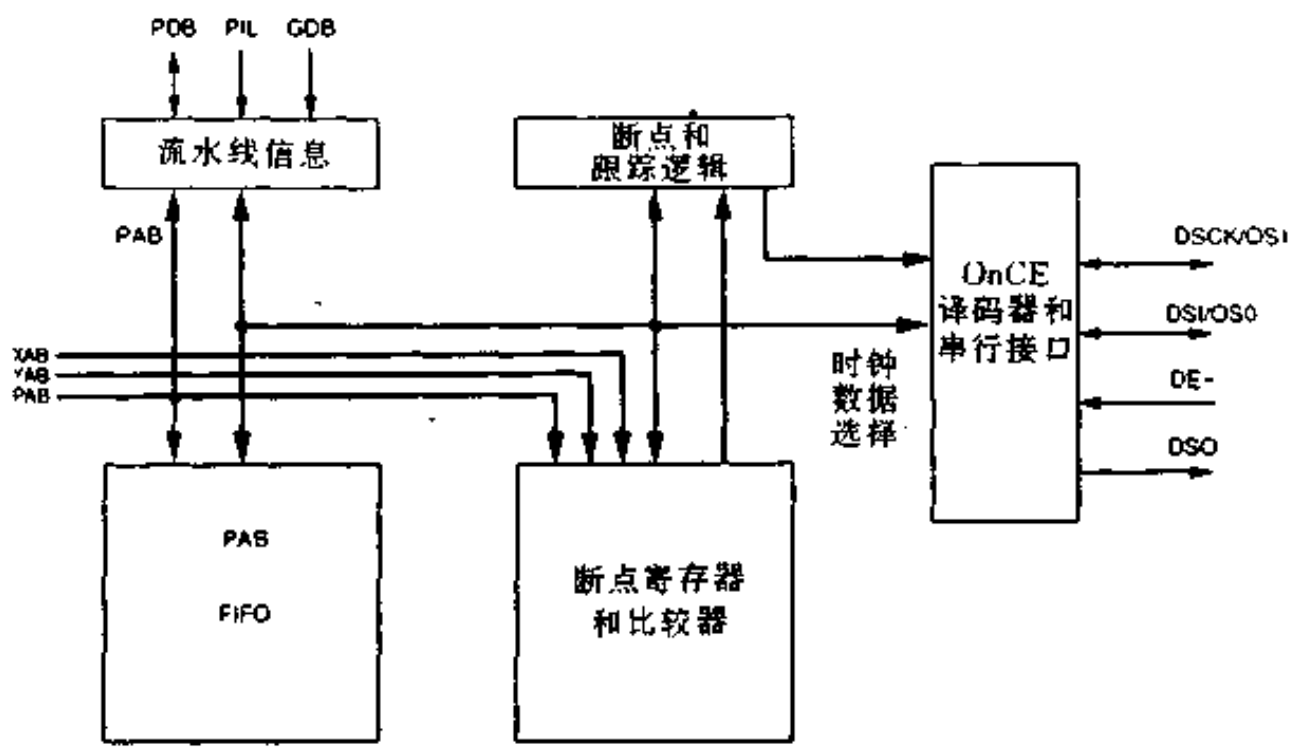


图21-1　OnCE™框图

21.2　片上仿真（OnCE™）输出端

21.2.1　调试串行输入/芯片状态0（DSI/OS0）

当输入时通过DSI/OS0端提供串行数据命令给OnCE™控制器。仅当DSP96002进入调

① OnCE™是Motorola公司的商标。

试操作方式时才识别 DSI 端上的接收数据。当锁在串行时钟的下降边缘上以前，数据必须有有效的 TTL 逻辑电平。数据总是首先由最高有效位（MSB）移入 OnCE™串行端口。当输出时，此端与 OS1 脚相连，提供片上状态信息，指出为什么调试方式不能响应外部请求。当不在调试方式时，DSI/OS0 端是输出（直到发出确认信号给命令控制器）。当从输出转换到输入时，引出端是三态的。为避免任何可能的误操作，应接一外部下拉电阻到此端。当硬件复位时，此端定为输出并驱动为低。

21.2.2 调试串行时钟/芯片状态 1（DSCK/OS1）

当输入时，通过 DSCK/OS1 端提供串行时钟给 OnCE™。串行时钟提供要求的脉冲把数据移入和移出 OnCE™串行端口。数据按时钟在下降边上进入 OnCE™并按时钟在上升边上出自 OnCE™串行端口。当输出时，此端与 OS0 端相连提供片上状态信息，说明为何调试方式不响应外部请求。当不在调试方式时，DSCK/OS1 端是输出（直到发出确认信号到命令控制器）。当从输出转换到输入时，此端是三态的。为避免任何可能的误操作，应接一外部下拉电阻到此端。当硬件复位时，此端定为输出并驱动为低。最大 SCK 频率是系统时钟频率的 1/3。

OS1	OS0	状态
0	0	正常状态
0	1	停止或等待状态
1	0	核心忙状态
1	1	核心或 DMA 忙状态

21.2.3 调试串行输出（DSO）

调试串行输出提供包含 OnCE™控制器寄存器之一中的数据，如由外部命令控制器收到的最后命令所指定。当空闲时，此端保持高电平。当所要求的数据可用时，DSO 线将被确定（负真逻辑）两个 T 周期（2T=DSP96002 控制时钟周期）以指出串行移存器为了释放数据而准备接收时钟。当发生踪迹或断点时，此线确定一个 T 周期以指出（确认）芯片已进入调试方式并正等待命令。数据总是最高有效位（HSB）首先移出 OnCE™串行端口。

21.2.4 调试使能输入（$\overline{DR}$）

调试请求输入提供由外部命令控制器进入调试工作方式的方法。当此端确定时，使 DSP96002 结束当前正在执行的指令，保留指令流水线信息，进入调试方式并等待来自调试串行输入线要进入的命令。

21.3 OnCE™控制器和串行接口

OnCE™控制器和串行接口包含以下块：输入移存器、位计数器、OnCE™译码器和状态/控制寄存器。图 21-2 表示 OnCE™串行接口框图。

21.3.1 OnCE™输入移存器（OISR）

OnCE™输入移存器是 8 位移位寄存器，从 DSI 线接收串行数据。数据在提供 DSCK 端的

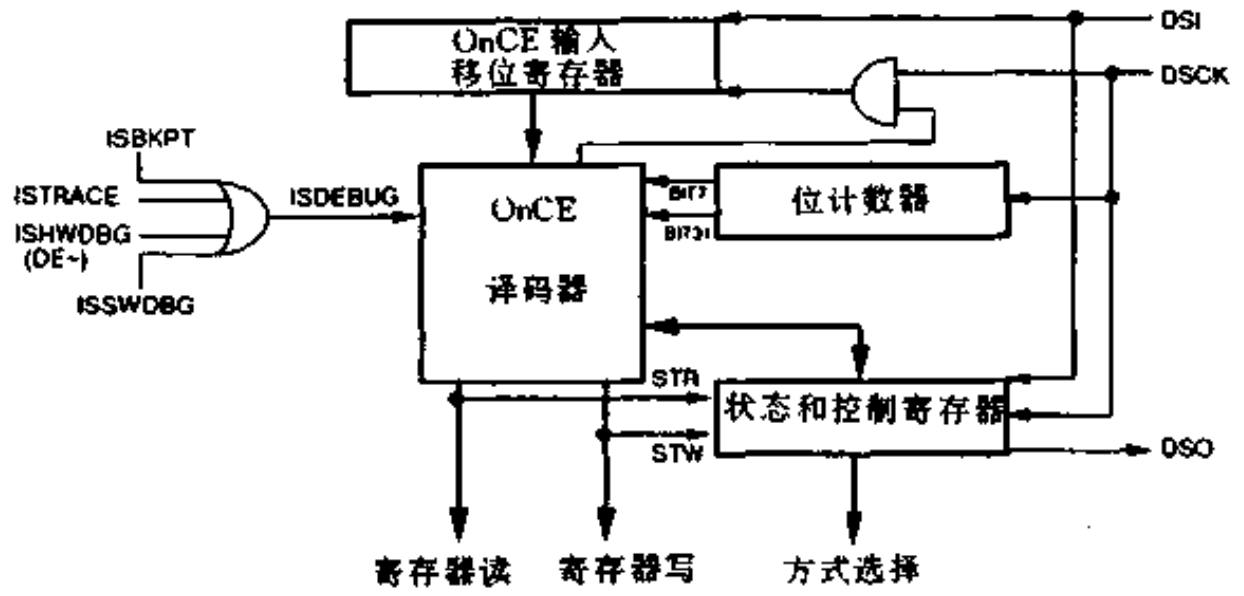


图 21-2 OnCE™控制器和串行接口

时钟的下降边进入寄存器计时，接收到第 8 位以后，OISR 停止移入新数据。被锁住的数据用来作为 OnCE™译码器的输入。数据总是首先移入 OISR 最高有效位。

21.3.2 OnCE™位计数器（OBC）

OnCE™位计数器是与移入和移出数据位有关联的 5 位计数器。OBC 按 DSCK 下降边递增。当硬件复位时和任何时候 DSP96002 确认调试方式已进入时，OBC 被清除。OBC 提供二个信号给 ONCE™译码器：一个指出前 8 位已移入（所以新的命令有效）；而第二个指出 32 位已移入（与此命令有关的数据有效）或 32 位已移出（由读命令要求的数据已移出）。

21.3.3 OnCE™译码器（ODEC）

OnCE™译码器管理全部 OnCE™活动。它接收来自 OISR 的 8 位命令，来自 OBC 的二个信号（一个指出 8 位已收到而另一个指出 32 位已收到），和指出核心和 DMA 停止的二个信号作为输入。ODEC 为读和写所选的寄存器产生所有要求的选通脉冲。

21.3.4 OnCE™状态和控制寄存器（OSCR）

状态和控制寄存器是 32 位寄存器，用于选择把芯片放入调试方式（看图 21-3）。断点在一个或多个存贮器上可能允许或不允许。操作方式的踪迹也选自 OSCR。控制位是读/写，而状态位是只读。

1. 程序存贮器断点允许（PBE0～PBEI）位 0～1

当程序存贮器地址在低和高的程序存贮器地址寄存器以内并选择断点是否为读或写访问进行识别时，这些控制位不屏蔽允许发生断点中断的程序存贮器 断点。这些位由硬件复位清除。

PBE1	PBE0	选　择
0	0	断点不允许
0	1	写访问断点
1	0	读访问断点
1	1	读和写访问断点

31 19	18	17	16	15 9	8	7	6	5	4	3	2	1	0
*	TO	DBO	PBO	*	TME	DBS1	DBS0	DBE1	DBE0	PBS1	PBS0	PBE1	PBE0

* 读作 0，应用 0 写入以便以后兼容。

图 21－3　OnCE™编程模型

2. 程序存贮器断点选择（PBS0～PBS1）位 2～3

这些控制位选择将识别程序存贮器断点是在核心程序存贮器提取、核心程序存贮器访问（MOVEM 或 MOVEP）或是在 DMA 程序存贮器访问上。这些位由硬件复位清除。

PBS1	PBS0	选　择
0	0	断点在核心取数访问上
0	1	断点在核心 P 传送访问上
1	0	断点在核心和 P 传送访问上
1	1	断点在 DMA 访问上

当 PBS1＝0 和 PBS0＝0 时，仅对实际执行指令的第一个指令字的提取允许程序存贮器断点。程序存贮器地址断点发生在取指令执行和断点计数器已减到 0 以后。

当 PBS1＝0 和 PBS0＝1 时，仅对从 MOVEP 和 MOVEM 指令到/从 P：存贮空间（MOVDP P…，…或 MOVE…，P…）导致明显的程序存贮器访问才允许程序存贮器断点。

当 PBS1＝1 和 PBS0＝0 时，对任何对程序空间的访问允许程序存贮器断点。

当 PBS1＝1 和 PBS0＝1 时，仅对 DMA 访问程序存贮空间才允许程序存贮器断点。

3. 数据存贮器断点允许（DBE0～DBE1）位 4～5

当数据存贮器地址在低和高数据存贮器地址寄存器以内并将选择断点识别是读还是写访问时，这些控制位允许数据存贮器断点发生。这些位由硬件复位清除。

DBE1	DBE0	选　择
0	0	断点不允许
0	1	断点在写访问上
1	0	断点在读访问上
1	1	断点在读和写访问上

4. 数据存贮器断点选择（DBS0～DBS1）位 6～7

这些控制位选择将识别数据存贮器断点是对 X 或 Y 数据空间在核心还是在 DMA 数据存贮器上访问。这些位由硬件复位清除。

DBS1	DBS0	选　择
0	0	断点在 X 核心取地址上
0	1	断点在 Y 核心取地址上
1	0	断点在 X DMA 取地址上
1	1	断点在 Y DMA 取地址上

5. 跟踪方式允许（TME）位 8

任何时候完成指令的执行和跟踪计数器是零时，设置此位，允许跟踪方式使芯片进入调试方式。此位由硬件复位清除。

6. 保留位（位 9～15，20～31）（略）

7. 程序存贮器断点发生（PBO）位 16

当程序存贮器断点发生时，设置此只读状态位。它由外部命令控制器用来决定如何进入调试方式。此位由硬件复位和读 OSCR 时来清除。

8. 数据存贮器断点发生（DBO）位 17

当数据存贮器发生断点时，设置此只读状态位。它由外部命令控制器用来决定如何进入调试方式。此位由硬件复位和读 OSCR 时来清除。

9. 跟踪发生（TO）位 18

当由跟踪计数器减为 0 和跟踪方式已经执行而进入操作调试方式时，设置此只读状态位。此位由硬件复位和读 OSCR 时清除。

10. 软件调试发生（SWO）位 19

由于条件真执行（F）DEBUGcc 指令而进入操作调试状态时，设置此状态位。此位由硬件复位和读 OSCR 时清除。

21.4 OnCE™硬件断点逻辑

硬件断点可设置在程序存贮器或数据存贮器单元上。断点也可不在程序流中，而在程序可能正执行的近似地址范围以内。这有效地增强了程序员实时进行监视程序的能力（看 21.3.4 节）

断点逻辑有二个相同的段，一是为程序存贮器断点，而另一为数据存贮器断点。每段包含核心或 DMA 地址的锁存、存贮上、下地址限的寄存器、比较器和一个计数器。图 21－4 表示 OnCE™程序存贮器逻辑的框图。图 21－5 表示 OnCE™数据存贮器断点逻辑的框图。

21.4.1 地址比较器断点寄存器

在决定程序可能在何处受损和数据何时会写入不应当实时写入的区域方面，地址比较器是有用的。在指定的点上停止程序以考察/改变寄存器或存贮器方面它们也是有用的。用地址比较器设置断点允许用户在任何操作方式时在 RAM 和 ROM 中设置断点。

当总线上地址大于或等于低界时，低地址比较器将产生逻辑真信号。当总线上地址小于或等于高界时，高地址比较器将产生逻辑真信号。若低地址比较器和高地址 比较器都发出逻辑真信号，则地址是在地址范围内，并且断点计数的内容如大于 0 则会递减。若内容是 0 则计数器不递减且发生断点异常。

若发生条件跳指令，由指令流水线产生的在被监视的程序块以内的条件跳地址是有效的，否则忽略条件跳地址。执行操作码或操作数以及断点计数器已降到 0 以后发生程序存贮器地址断点。

执行形成数据存贮器地址的指令以及断点计断器已降到 0 以后也发生数据存贮器地址断点。

所有断点寄存器由调试状态的控制寄存器 OSCR 控制。

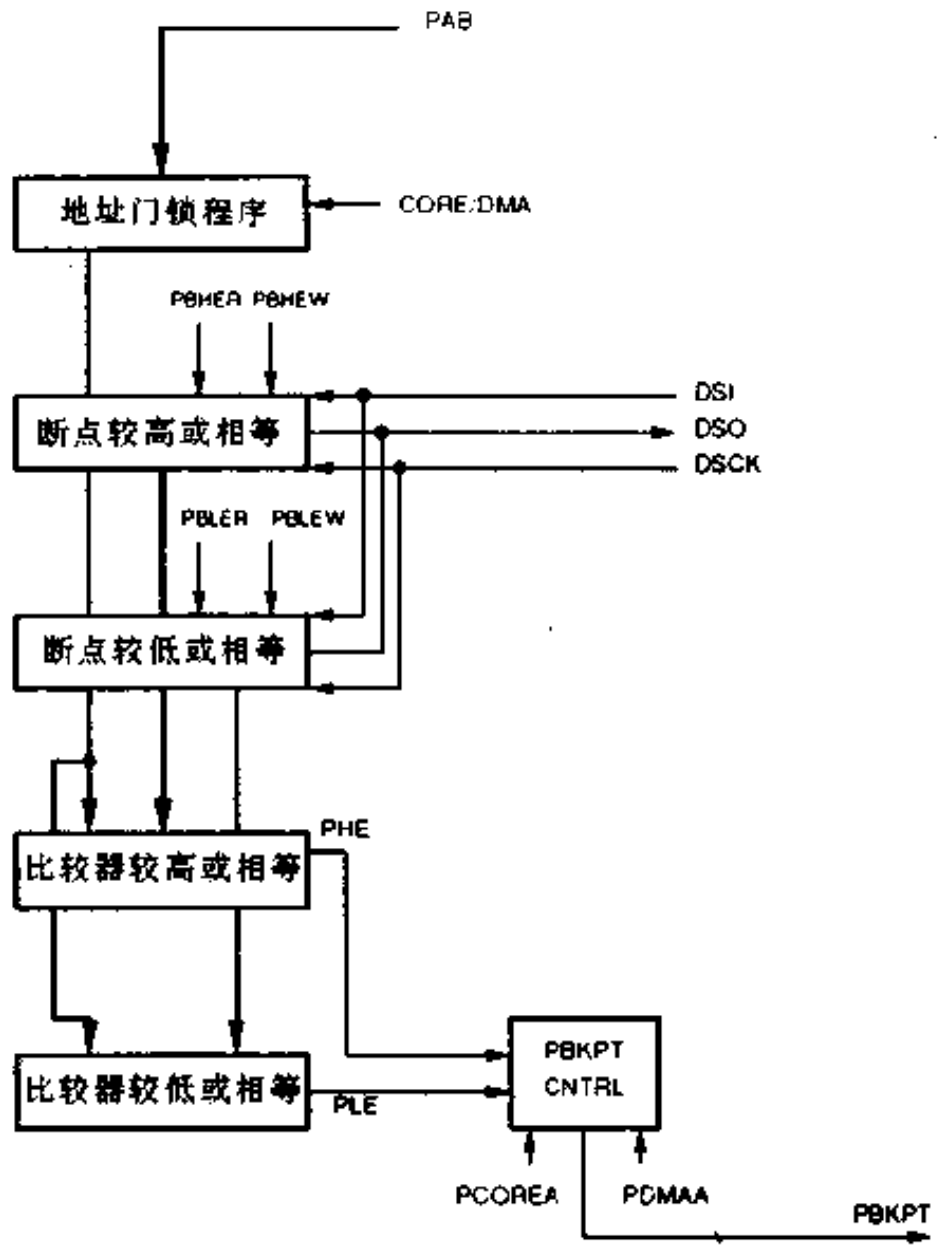


图 21－4　程序存贮器断点逻辑

21.4.2　断点计数器

为停止在程序循环的第 n 次循环或当数据存贮器访问第 n 次发生时，断点计数器是有用的。此信息有效地减少运算调试时间，并提供在程序段中以及外围或数据访问中检验热点的方法。

程序热点可能由以下情况随机评价：设置断点计数器到一个值，在程序地址比较器寄存器中设置程序地址，通过 DSP96002 的控制返回到用户程序以及检验在程序存贮器访问 n 次循环后是否发生断点。

当调试实时快速中断序列如在指定数量的主机传送发生以后用于或停止 A/D 或 D/A 变换时，断点计数器成为有力的工具。

断点计数器由硬件复位清除。

21.4.3　程序存贮器地址锁存器（OPAL）

程序存贮器地址锁存器是 32 位寄存器，在磁芯开槽或 DMA 开槽期间的每个周期内按照 OSCR 中 PBS1～PBS0 位将 PAB 锁存在该寄存器中。

21.4.4　程序存贮器上限寄存器（OPULR）

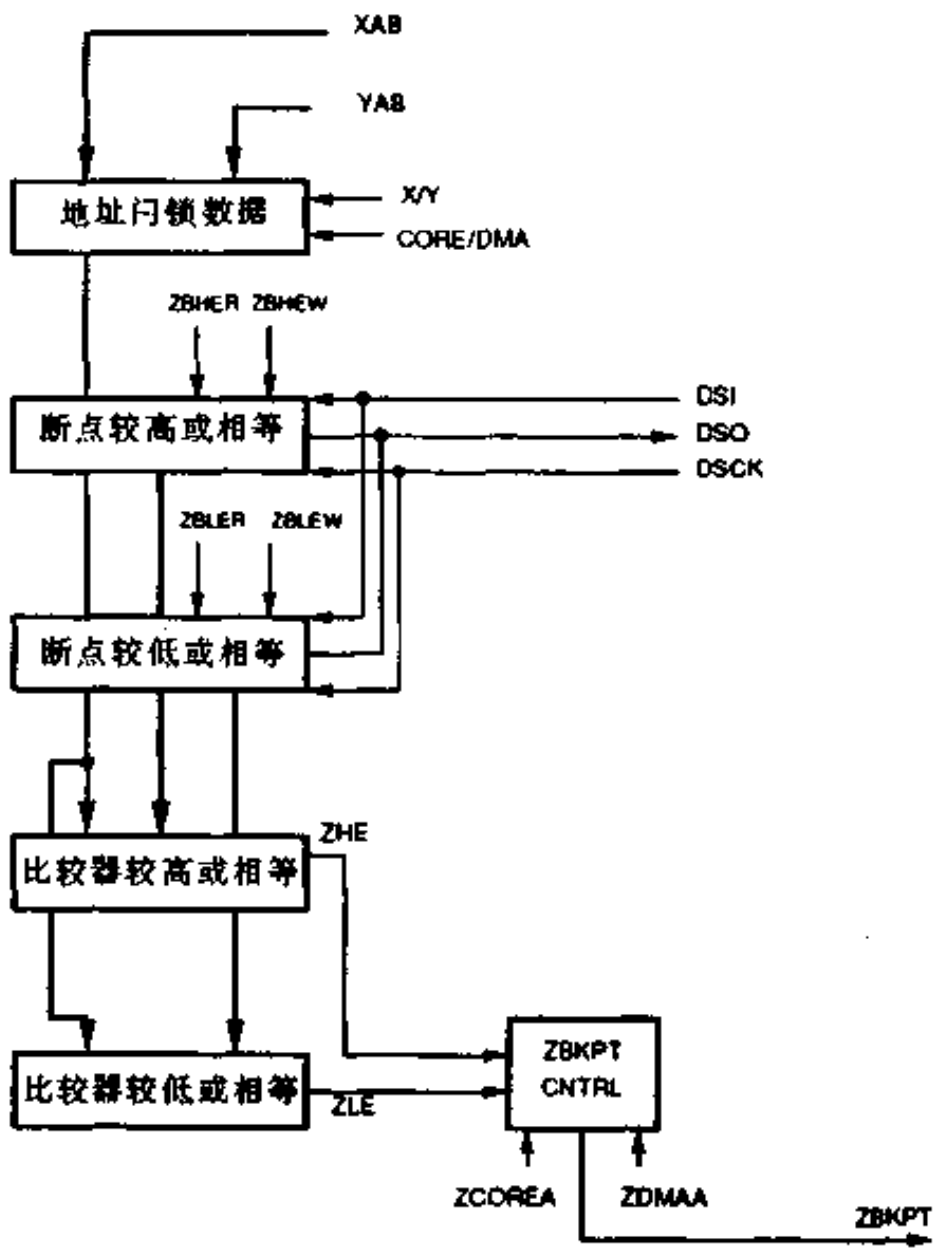


图 21－5　数据存贮器断点逻辑

程序存贮器上限寄存器是 32 位寄存器，它存贮程序存贮器断点的上限。OPULR 仅可通过串行接口读或写。允许断点以前，OPULR 必须装入命令控制器。

21.4.5　程序存贮器下限寄存器（OPLLR）

程序存贮器下限寄存器是 32 位寄存器，它存贮程序存贮器断点的下限。OPLLR 仅可通过串行接口读或写，允许断点以前，OPLLR 必须装入命令控制器。

21.4.6　程序存贮器高地址比较器（OPHC）

程序存贮器高地址比较器比较当前程序存贮器地址（由 OPAL 存入）和 OPULR 的内容。若 OPULR 高于或等于 OPAL，则比较器送出信号指出地址是低于或等于高限。

21.4.7　程序存贮器低地址比较器（OPLC）

程序存贮器低地址比较器比较当前程序存贮器地址（由 OPAL 存入）和 OPLLR 和内容。若 OPLLR 低于或等于 OPAL，则比较器送出信号指出地址是高于或等于低限。

21.4.8　程序存贮器断点计数器（OPBC）

程序存贮器断点计数器是 32 位计数器，它装入等于次数减 1 的值，这是在断点确认以前，

应当访问程序存贮器地址的次数。在每一次发生程序存贮器地址访问时，计数器递减。当计数器达到0值并发生新的事件时。则产生一信号，而且若PBE被置定，则芯片进入调试方式。OPBC仅可通过串行接口读或写。允许程序存贮器断点以前，OPBC必须由命令控制器装入，图21－6表示程序存贮器断点计数器逻辑框图。

21.4.9 数据存贮器地址锁存器（ODAL）

数据存贮器地址锁存器是32位寄存器，在磁芯或DMA按照OSCR中的DBS1～DBS0开槽期间，在每个周期内锁存XAB或YAB。

21.4.10 数据存贮器上限寄存器（ODULR）

数据存贮器上限寄存器是32位寄存器，它存贮数据存贮器断点上限。ODULR仅能通过串行接口读或写。允许断点以前，ODULR必须由命令控制器装入。

21.4.11 数据存贮器下限寄存器（ODLLR）

数据存贮器下限寄存器是32位寄存器，它存贮数据存贮器断点下限。ODLLR仅能通过串行接口读或写。允许断点以前，ODLLR必须由命令控制器装入。

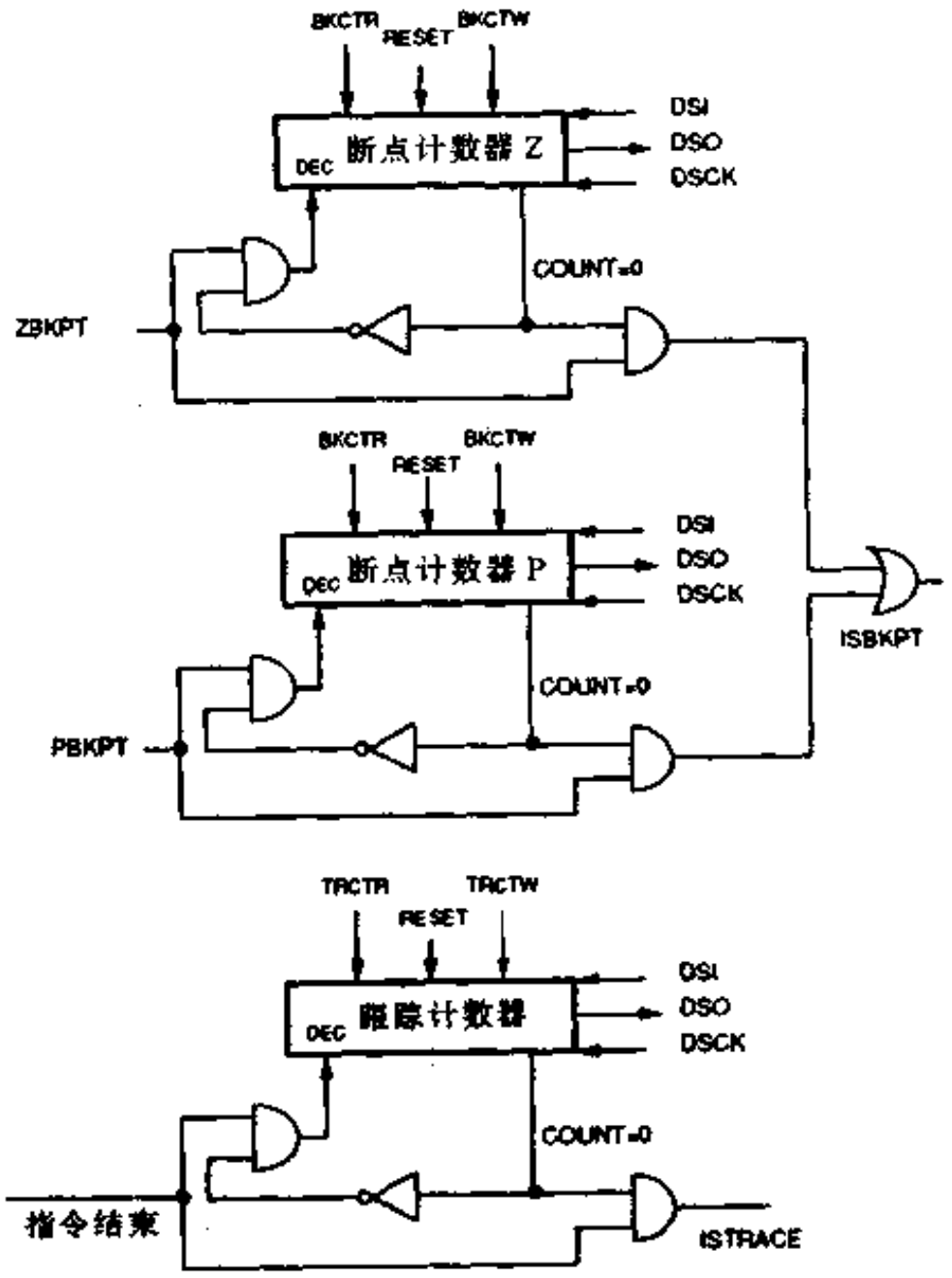


图21－6 断点和跟踪计数器逻辑

21.4.12 数据存贮器高地址比较器（ODHC）

数据存贮器高地址比较器比较当前数据存贮器地址（由 ODAL 存入）和 ODULR 的内容。若 ODULR 高于或等于 ODAL，则比较器送出信号指出地址低于或等于高限。

21.4.13 数据存贮器低地址比较器（ODLC）

数据存贮器低地址比较器比较当前数据存贮器地址（由 ODAL 存入）和 ODLLR 的内容。若 ODLLR 低于或等于 ODAL，则比较器送出信号指出地址高于或等于低限。

21.4.14 数据存贮器断点计数器（ODBC）

数据存贮器断点计数器是 32 位寄存器，以次数减少的值装入，它是在断点确认以前应访问数据存贮器地址的次数。每次数据存贮器访问，计数器递减。当计数器到达 0 并且新的事件发生时，则产生一信号，并若设置 DBE 位，则芯片进入调试方式。ODBC 仅能通过串行接口读或写。允许数据存贮器断点以前，ODBC 必须由命令控制器装入。图 21－6 表示数据存贮器断点计数器逻辑的框图。

21.5 跟踪/跳步方式

为了以单步或多步执行 DSP96002 指令，需要类似于工作在 DSP56001 上的跟踪方式的特殊方式。DSP96002 不形成像 DSP56001 那样的中断异常，而是进入调试工作方式并在每个指令或指令群之后等待进一步来自调试串行端口的指令。

21.5.1 跟踪计数器（OTC）

跟踪方式有一个与之相关的 32 位计数器，使在返回调试工作方式前可执行多于一条的指令。计数器的目的是使用户可用多指令步实时而不受调试方式的干扰。此特性帮助软件开发者调试码段，这码没有正常的数据流或是中止在无限循环中。跟踪计数器亦允许用户调试时间临界的码区。

为了允许跟踪工作方式，以一数值装入计数器，设置程序计数器到要实时执行的指令开始位置。在 OSCR 中选择跟踪方式，并且 DSP96002 由执行外部命令控制器发出的适当命令退出调试方式。

一旦退出调试方式，每次执行一条指令后计数器就递减。中断是可服务的并且所有执行的指令（包括快速中断服务）将递减跟踪计数器。计数器降到 0，DSSP96002 将重新进入调试方式（中断服务断点信号，ISBKPT 设置），DSCR 中的跟踪发生位将被设置并且 DSO 端被置定以指出 DSP96002 已进入调试方式并请求服务。

跟踪计数器由硬件复位或每当进入调试工作方式来清除。图 21－6 表示跟踪计数器逻辑的框图。

21.6 OnCE™串行端口定时

把外部数据以在可变速率上定时的每一位送入串行输入线。最小时钟速率应是 1 MHz，而最大时钟速率应是 10 MHz 串行时钟（建立时间）的下降边之前至少 10 ns 串行输入位必须稳定，并且时钟（保持时间）下降边之后至少 10 ns 必须保持稳定。

串行输出线必须由来自命令控制器进入的最后命令所指定的，所选寄存器定时输出数据。数据位的值在时钟上升边是有效的，并在时钟上升边以后至少 10 ns 仍保持有效。

进入调试工作方式以后串行输出线对至少一个 T 周期将降低以标志 DSP96002 请求断点或跟踪服务的命令控制器。

21.7 进入调试方式的方法

以 1 个 T 周期定 DSO 线由芯片确认进入调试方式。这告诉外部命令控制器芯片已进入调试方式并等待命令。调试方式可以 7 种途径进入。

21.7.1 当$\overline{\text{RESET}}$时外部请求

当$\overline{\text{RESET}}$确定时，保持$\overline{\text{DR}}$线确定使芯片进入调试方式。收到确认以后，命令控制器必须不确定$\overline{\text{DR}}$线。注意此时在进入调试方式以前芯片不进行任何取数或存贮访问。

21.7.2 当正常工作时外部请求

当正常芯片工作时保持$\overline{\text{DR}}$线确定，使芯片结束执行当前指令，然后进入调试方式。收到确认以后，命令控制器必须不确定$\overline{\text{DR}}$线。注意此请况下芯片在新的取数指令进入指令锁存器以后完成并停止当前指令的执行。这一过程与任何新的取指令相同，包括由中断处理提取的指令或将被中断处理取消的指令。

21.7.3 当 STOP 时外部请求

当芯片在 STOP 状态（即已执行 STOP 指令）时确定$\overline{\text{DR}}$，使芯片退出 STOP 状态并进入调试方式。收到确认后，命令控制器必须对$\overline{\text{DR}}$取非。注意此请况下，芯片完成执行 STOP 指令，并在下一指令进入指令锁存器后暂停。

21.7.4 当 WAIT 时外部请求

当芯片处于 WAIT 状态（即已执行等待指令）时确定$\overline{\text{DR}}$，使芯片退出 WAIT 状态并进入调试方式。收到确认后，命令控制器必须对$\overline{\text{DR}}$取非。注意此情况下，芯片完成执行 WAIT 指令，并在下一指令进入指令锁存器以后暂停。

21.7.5 在正常工作时软件请求

在执行（F）DEBUGcc 指令中，当指定的条件是真时，跟随（F）DEBUGcc 指令的指令已进入指令锁存器以后，芯片进入调试方式。

21.7.6 允许跟踪方式

当工作在跟踪方式并且跟踪计数器已到达 0 时，在使最后的跟踪计数器递减的指令完成执行以后芯片进入调试方式。仅指令的实际执行使跟踪计数器递减，即已取消的指令不递减跟踪计数器，并且不使芯片进入调试方式。

21.7.7 允许断点

当工作在跟踪方式或正常方式，以及用断点计数器值为 0 允许断点的机理时，在完成使断点计数器递减的指令执行以后，芯片进入调试方式。在程序存贮地址上断点的情况下，已使指定的地址产生的指令执行以后，将立即确认断点。在数据存贮地址上断点的情况下，在跟随使指定的地址产生的指令完成以后，将确认断点。

21.8 流水线信息

为了恢复流水线使之从调试方式返回时继续正常的芯片工作，一些片上寄存器存贮芯片流水线状态。图 21-7 表示流水线信息寄存器框图，但图 21-8 所示的 PAB 寄存器除外。

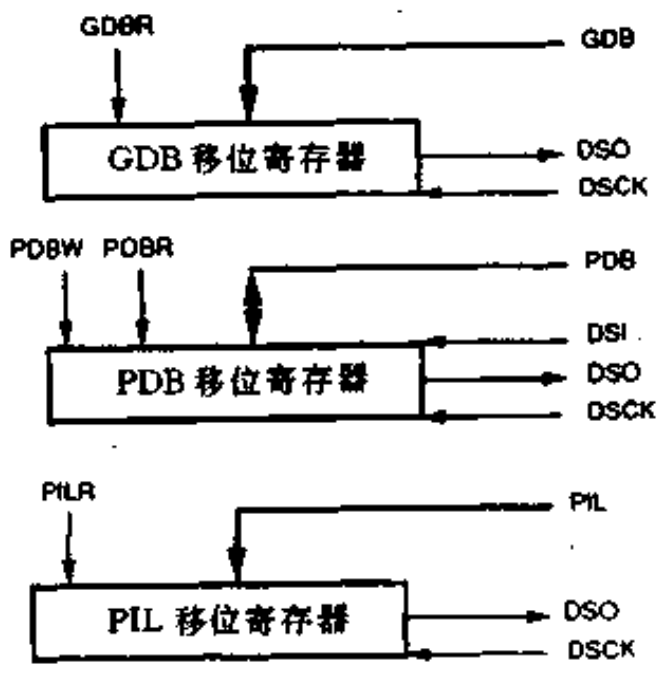


图 21-7 流水线信息寄存器

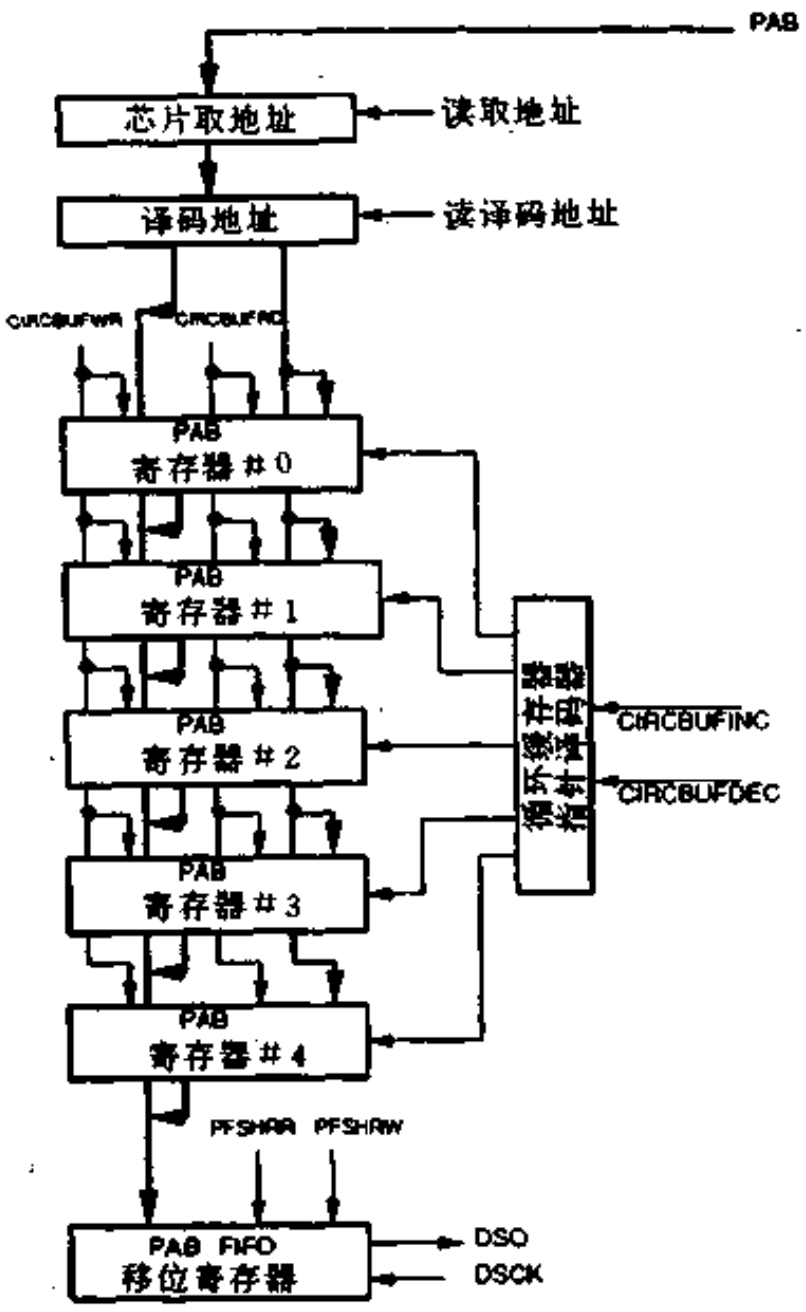


图 21-8 程序地址总线 FIFO

21.8.1 PAB 寄存器（OPABF，OPABD）

当进入调试方式时有二个只读 PAB 寄存器给出流水线信息。OPABF 寄存器指出哪个操作码地址在流水线取数阶段，OPABD 指出哪个操作码在译码阶段。在正常程序流情况下，保存的程序地址将是所取最后指令前面指令的地址，并在进入调试方式以前解码。只能通过串行接口读或写 PAB 寄存器。

21.8.2 PDB 寄存器（OPDBR）

PDB 寄存器是 32 位锁存器，它在进入调试方式以前，存贮由最后程序存贮器访问产生的程序数据总线的值。只能通过串行接口读或写 OPDBR。这个寄存器当调试方式时受已完成的操作的影响，而当返回正常方式时必须由命令控制器恢复。

21.8.3 PIL 寄存器（OPILR）

PIL 寄存器是 32 位锁存器，在进入调试方式以前，它存贮指令锁存器的值。只能通过串行接口读 OPILR。当调试方式时寄存器受所完成操作的影响，当返回正常状态时必须由命令控制器来恢复。因为对寄存器没有直接访问，此任务首先由写 OPDBR 完成，从而来自 OPDBR 的数据锁在 OPILR 中。

21.8.4 GDB 寄存器（OGDBR）

GDB 寄存器是 32 位只能通过串行接口来读的锁存器。从流水线状态恢复的观点看，实际不需要 OGDBR；而作为在芯片和命令控制之间通信的一种方法则需要 OGDBR。在内部 I/O 空间地址 $FFFFFFF0 处映射 OGDBR。任何时候命令控制器需要像寄存器或存贮器值那样的数据字，这将强迫芯片执行带信息到 OGDBR 的指令。然后 OGDBR 的内容将由命令“READ GDB REGISTER”串行送到命令控制器。

21.9 PAB 历史缓存器

为了便于调试工作和保持程序流的跟踪，提供先进先出缓存器，它存贮已执行的最后 5 条指令的地址以及最后提取的指令的地址和在指令锁存器中当前指令的地址。

21.9.1 用于取数的 PAB 寄存器（OPABFR）

用于取数的 PAB 寄存器是 32 位寄存器，它存贮进入调试方式以前已提取的最后指令的地址。只能通过串行接口读 OPABFR。此寄存器不受调试方式期间所完成操作的影响。

21.9.2 用于译码的 PAB 寄存器（OPABDR）

用于译码的 PAB 寄存器是 32 位寄存器，它存贮指令锁存器中当前指令的地址。若芯片不进入调试方式，则它是已经译码的指令。只能通过串行接口读 OPABDR。这寄存器不受调试方式期间所完成操作的影响。

21.9.3 PAB FIFO

为了便于调试工作和保持跟踪程序流，提供先进先出缓存器以存贮已执行的最后5条执令的地址。FIFO作为循环缓存器，包含5个32位寄存器和三个3位计数器。所有寄存器有相同地址，但任何对FIFO地址的读访问都会使计数器递增，从而指向下一个FIFO寄存器。这个寄存器通过它们的公用FIFO地址，对命令控制器是串行有效，图21-8表示程序地址总线FIFO的框图。除了当读FIFO时FIFO指针递增外，调试方式期间所完成的操作不影响FIFO。进入调试方式前，最后执行的指令将在FIFO的底部。

21.10 串行协议说明

为了在命令控制器和DSP96002芯片之间进行有效的通信，采用以下协议。任何调试活动开始之前，命令控制器必须等待芯片已进入调试方式的确认，注意在断点、跟踪或软件(F)DEBUGcc指令情况下，确认本身是调试对话的开始。命令控制器与芯片通信时送出可能伴有32位数据的8位命令，送出命令后，命令处理器等待芯片执行命令的确认。仅在芯片确认前面的命令执行之后，命令处理器才可送出新命令。

OnCE™命令

有两种命令：读命令（当芯片将发送被请求数据时）和写命令（当芯片将接收数据和将写数据到片上资源之一时）。命令是8位长并具有图21-9所示格式。

7	6	5	4	3	2	1	0
R/W	GO	EX	RS4	RS3	RS2	RS1	RS0

图21-9 OnCE™命令格式

1. *寄存器选择*（RS4～RS0）位0～4

寄存器选择位定义哪个寄存器对读（写）操作是源（目的）。

RS4～RS0	选择的寄存器	RS4～RS0	选择的寄存器
00000	调试状态/控制（OSCR）	01101	清除数据断点计数器
00001	断点计数器程序（OPBC）	01110	清除跟踪计数器
00010	断点计数器数据（ODBC）	01111	保留
00011	跟踪计数器（OTC）	10000	保留
00100	断点数据存贮器高等于（ODULR）	10001	程序地址总线FIFO和递增计数器
00101	断点数据存贮器低等于（ODLLR）	10010	用于译码的程序地址总线锁存器（OPABD）
00110	断点程序存贮器高等于（OPULR）	10011	保留
00111	断点程序存贮器低等于（OPLLR）	101XX	保留
01000	传送寄存器（OGDBR）	11XX0	保留
01001	程序数据总线锁存器（OPDBR）	11X0X	保留
01010	用于取数的程序地址总线锁存器（OPABF）	110XX	保留
01011	程序指令锁存器（OPILR）	11111	不选寄存器
01100	清除程序断点计数器		

2. *退出命令*（EX）位5

若设置EX，则脱离调试方式并恢复正常操作。

EX	动作
0	保持调试方式
1	脱离调试方式

3. 继续命令（GO）位 6

若设置 GO 则执行指令。

GO	动作
0	不动作（没有动作发生）
1	执行指令

4. 读/写命令（R/W）位 7

R/W	动作
0	把与命令有关的数据写入由 RS4～RS0 指定的寄存器
1	读由 RS4～RS0 指定的寄存器中的数据

21.11 DSP96002 目标位置调试系统要求

典型的 DSP96002 调试环境由目标系统组成，这里 DSP96002 属于硬件定义的用户。调试串行端口通过由 4 条 OnCE™线、地址和复位线组成的 6 条线与命令转换器接口。复位线是可选的并且仅用于复位 DSP96002 和与其有关的电路。

命令控制器如同 DSP96002 目标系统和主计算机之间的媒介。主计算机到控制器接口采用标准 RS232 三线电缆或 DSP96002 应用开发系统并行总线。命令控制器板上的跳线可选择将采用什么联系方法。这使主计算机与控制电路的联系能有很多不同的方法。

控制电路提供一些重要功能。它可作为 DSP96002 串行调试端口驱动器、主机命令解释器和 DSP96002 控制器。当在调试方式时 DSP96002 作为从机，并仅对请求提供数据。控制器从与用户相联的用户接口程序发出基于主计算机输入的命令。

21.12 使用 OnCE™

用以下符号：

ACK＝等待 DSO 端上的确认

CLK＝发出 32 时钟从所选寄存器读出数据

21.12.1 开始调试工作

多数调试工作有以下的开头：

(1) ACK

(2) 保存流水线信息：

① 送命令 READ　PDB　REGISTER

② ACK

③ CLK

④ 送命令 READ PIL REGISTER（指令锁存器）

⑤ ACK

⑥ CLK

（3）读 PAB FIFO 和提取/译码

① 送命令 READ PAB 为取地址

② ACK

③ CLK

④ 送命令 READ PAB 为译地址

⑤ ACK

⑥ CLK

⑦ 送命令 READ FIFO REGISTER（和递增指针）

⑧ ACK

⑨ CLK

⑩ 送命令 READ FIFO REGISTER（和递增指针）

⑪ ACK

⑫ CLK

⑬ 送命令 READ FIFO REGISTER（和递增指针）

⑭ ACK

⑮ CLK

⑯ 送命令 READ FIFO REGISTER（和递增指针）

⑰ ACK

⑱ CLK

⑲ 送命令 READ FIFO REGISTER（和递增指针）

⑳ ACK

㉑ CLK

21.12.2　显示指定的寄存器

（1）送命令 WRITE PDB REGISTER 和 GO（不是 EX）。

（ODEC 选择 PDB 对串行数据作为目的。）

（2）ACK

（3）送 32 位操作码："MOVE reg，x：OGDB"

（32 位收到后，PDB 寄存器驱动 PDB. ODEC 由暂停状态释放芯片，同时指令中所指定寄存器的内容装入 GDB 寄存器，标有指令结束的信号使芯片返回"暂停"状态，并给命令控制器发出确认）。

（4）ACK

（5）送命令 READ GDB REGISTER

(6) ACK

(7) CLK

21.12.3 从地址××××开始显示 X 存贮区

此命令用 Rn 以减少串行信息。

(1) 送命令 WRITE PDB REGISTER 和 GO (不是 EX)。

(ODEC 选择 PDB 串行数据作为目的。)

(2) ACK

(3) 送 32 位操作码："MOVE R0，x：OGDB"

(4) ACK

(5) 送命令 READ GDB REGISTER

(6) ACK

(7) CLK

(命令控制器产生 32 时钟，它移出 GDB 寄存器的内容。因此保存 R0 的值并在退出调试方式前将得到恢复。)

(8) 送命令 WRITE PDB REGISTER (不是 GO，EX)

(9) ACK

(10) 送 32 位操作码："MOVE #$××××，R0"

(32 位收到之后，PDB 寄存器驱动 PDB。ODEC 使核心装入操作码，发出确认给命令控制器。)

(11) ACK

(12) 送命令 WRITE PDB REGISTER 和 GO (不是 EX)

(13) ACK

(14) 送 "MOVE #$××××，R0" 的 32 位第二个字 (××××字段)

(15) ACK

(16) 送命令 WRITE PDB REGISTER 和 GO (不是 EX)

(17) ACK

(18) 送 32 位操作码："MOVE X：(R0) +，×：OGDB"

(19) ACK

(20) 送命令 READ GDB REGISTER

(21) ACK

(22) CLK

(23) 送命令 NO SELECTION 和 GO (不是 EX)

(24) ACK

(25) 送命令 READ GDB REGISTER

(26) ACK

(27) CLK

(28) 从 23 步重复直到考察过全部存贮区。在处理结束时，R0 必须恢复。

21.12.4 从调试方式返回到正常方式

为从调试方式返回有两种情况：调试起始前控制返回到正在运行的程序或是改变寄存器从而跳到不同的程序。

1. 情况1：返回到以前的程序（返回到正常方式）

（1）送命令 WRITE PDB REGISTER

（ODEC 选择 PDB 作为串行数据的目的，ODEC 也选择片上 PAB 寄存器作为 PAB 总线的源。驱动 PAB 以后，发出确认信号给命令控制器。）

（2）ACK

（3）送保存的 PIL（指令锁存器）值 的 32 位。

（全部 32 位收到后，PDB 寄存器驱动 PDB。ODEC 使核心去装操作码。送出确认信号给命令控制器。）

（4）ACK

（5）送命令 WRITE PDB REGISTER（GO，EX）

（ODEC 选择 PDB 作为串行数据的目的。）

（6）ACK

（7）送保存的 PDB 值的 32 位

（收到 32 位后，PDB 寄存器驱动 PDB。ODEC 从“暂停”状态释放芯片并清除 OSCR 中的调试方式位。芯片继续执行指令直到发生调试方式。）

2. 情况2：跳到新的程序（从地址 $××××××××进行）

（1）送命令 WRITE PDB REGISTER（不是 GO，EX）

（ODEC 选择 PDB 作为串行数据的目的。）

（2）ACK

（3）送双字跳指令（$030c3f80）的操作码的 32 位代替保存的 PIL（指令锁存器）值。

（同情况 1 中 3）

（4）ACK

（5）送命令 WRITE PDB REGISTER（GO，EX）

（6）ACK

（7）送目标绝对地址（$××××××××）的 32 位。芯片将由目标地址（不必担心流水线）恢复取数。注意跟踪计数器将计指令数，所以若跟踪方式在 OSCR 中的使能位已经设置，则当前跟踪计数器必须更正。（如 32 位收到后，PDB 寄存器驱动 PDB。ODEC 从“暂停”状态释放芯片并清除 OSCR 中的调试方式位。芯片首先执行跳指令。然后从目标地址提取指令。芯片继续执行来自该地址的指令直到发生调试方式。）

[General Information]
书名=Motorola集成电路应用技术丛书　　数字信号处理原理及应用
作者=陆心如
页数=349
SS号=0
下载位置=http://192.168.36.205/01/diskdz1/dz24/20/!00001.pdg