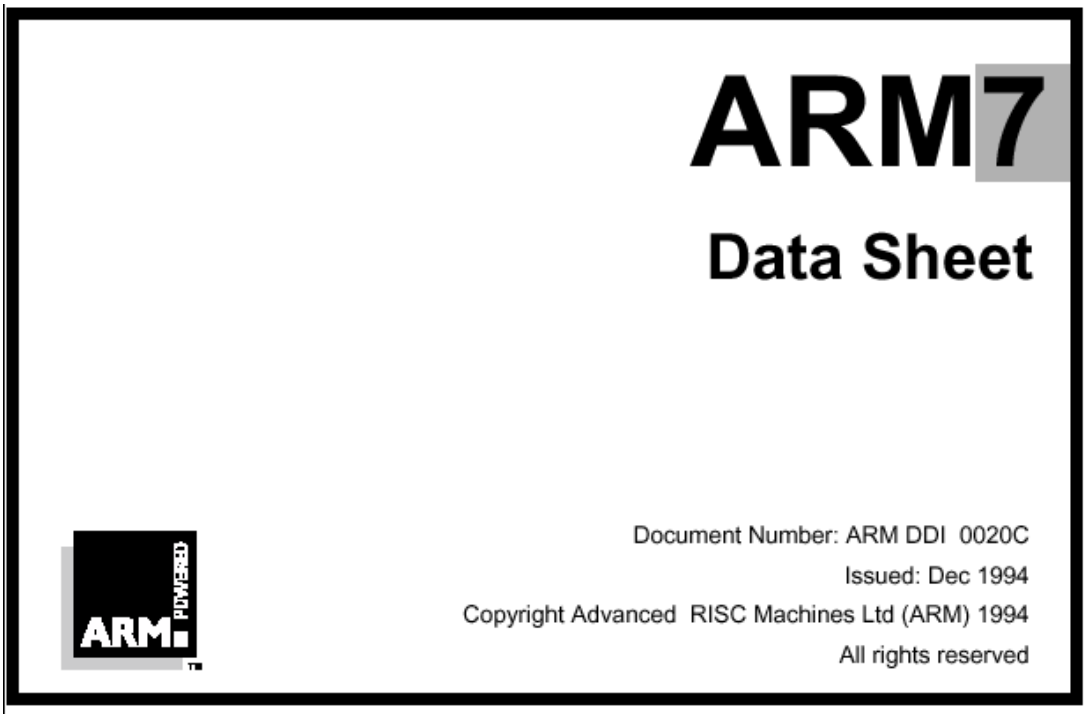


ARM7 数据手册

http://diyclub.nease.net
翻译：aufan

序言：

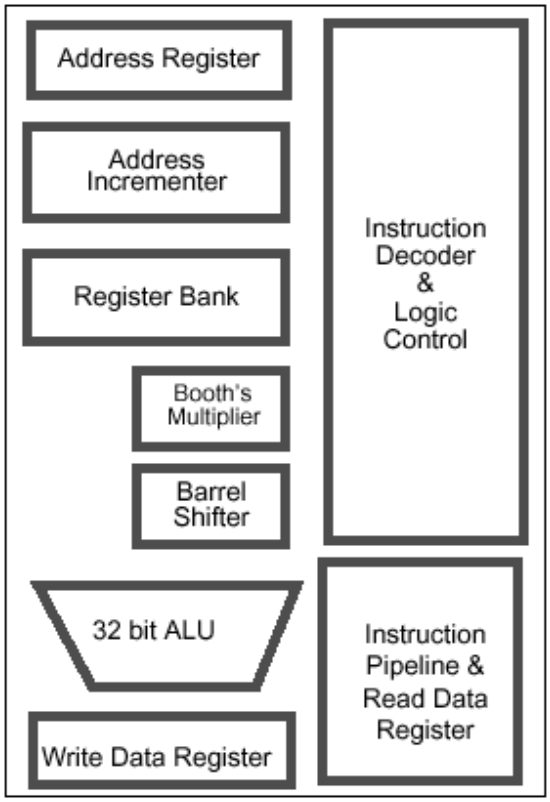
ARM7 是一种低电压，通用 32 位 RISC 微处理器单元，可作一般应用或嵌入到 ASIC 或 CSIC 中，其简洁一流的设计特别适用于电源敏感的应用中。ARM7 的小尺寸使它特别适合集成到比较大的客户芯片中，此芯片中也可以包含 RAM, ROM, DSP，逻辑控制和其他代码。

增强特性：

ARM7 和 ARM6 有相似性，但增加了以下功能：
基于亚微米的制程，增加了速度，减少了电源消耗
3V 操作，很小的电源消耗，并同 5V 系统兼容
较高的时钟对所以程序执行较快。

特性总结：

- 32 位的 RISC 结构处理器（包括 32 位地址线和数据线）；
- Little/Big Endian 操作模式；
- 高性能 RISC
  17 MIPS sustained @ 25 MHz (25 MIPS peak) @ 3V
- 较低的电压损耗
  0.6mA/MHz @ 3V fabricated in .8 m CMOS全静态操作
- 适用于对电源比较敏感的应用中
- 快速中断响应
- 适用于实时系统
- 支持虚拟内存
- 支持高级语言
- 简单但功能强大的指令系统



应用
ARM7适用于那些需要紧凑且功能强大的RISC处理器系统

电讯	GSM终端控制
数据通信	协议转换
便携式计算机	掌上电脑
自动控制系统	发动机管理单元
信息存贮系统	存储卡
图像处理	JOEG控制器

目录

1.0 简介

ARM7是32 位通用微处理器ARM(Advanced RISC Machines)家族中的一员，具有比较低的电源消耗和良好的性价比，基于（精简指令）RISC结构，指令集和相关的译码机制与微程序控制的复杂指令系统的计算机相比要相对简单，这使得它拥有比较高的指令处理能力和实时中断响应能力。

指令集包含11种基本类型：

两种类型用于偏上算术逻辑单元，桶式移位器和乘法器，在31个寄存器（32位）间执行高速操作；

三种类型的指令控制数据在存储器和寄存器之间传送，一种用于弹性地址，一种用于高速内容切换，一种用于交换数据；

三种类型的指令用于控制流程和特权级执行；

三种专门用于控制外部的协处理器，此种协处理器允许指令集的功能以开放和统一的格式扩展到片外。

ARM指令集对不同高级语言的编译器来讲都比较适用，需要临界的代码段，汇编语言的编程也很简单，不像其它的微处理器，需要依靠复杂的编译器来管理指令。

由于应用了流水线技术，所以指令处理和存储系统的各个部分都可以连续运行。典型的例子，一条指令正在执行，下一条指令正被译码，第三条指令同时从存储器中取出。

在存储系统中，存储接口的设计可以最大限度地发挥性能潜力而不需要花费很大的代价，速度敏感信号做成流水线方式，允许系统控制功能应用于标准的低电压逻辑，这些控制信号推动了由工业DRAM提供的快速局部访问模式。

ARM7有32位的地址总线，所有的ARM处理器共享同一个指令集，ARM7可以配制成26位地址线，向下兼容其它早期的处理器。

ARM7是全静态CMOS，允许时钟在周期的任意部分被停止，但不丢失状态。

注意：

0x　　--十六进制

BOLD　　--外部信号用黑体字显示

1.1 ARM7 Block diagram

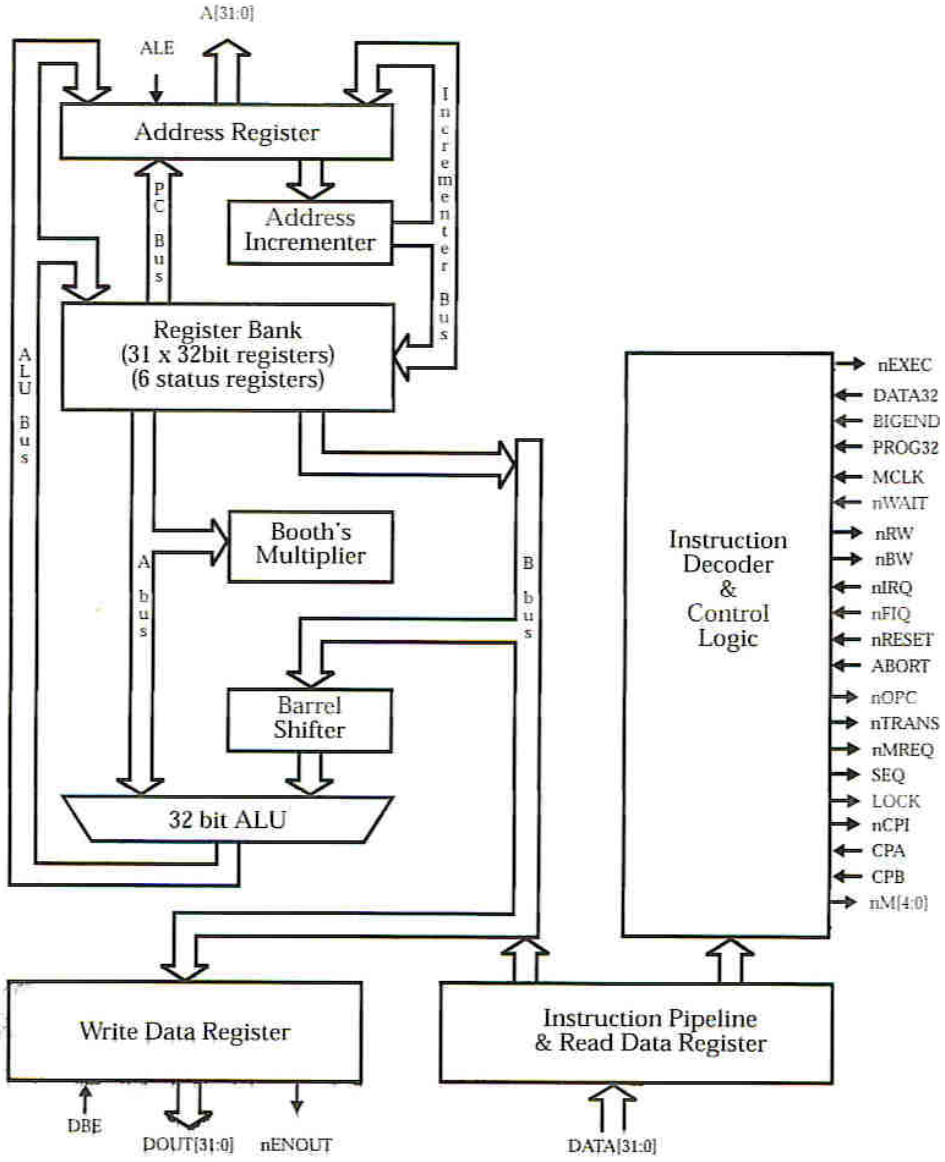


Figure 1: ARM7 Block Diagram

1.2 ARM7 Functional Diagram

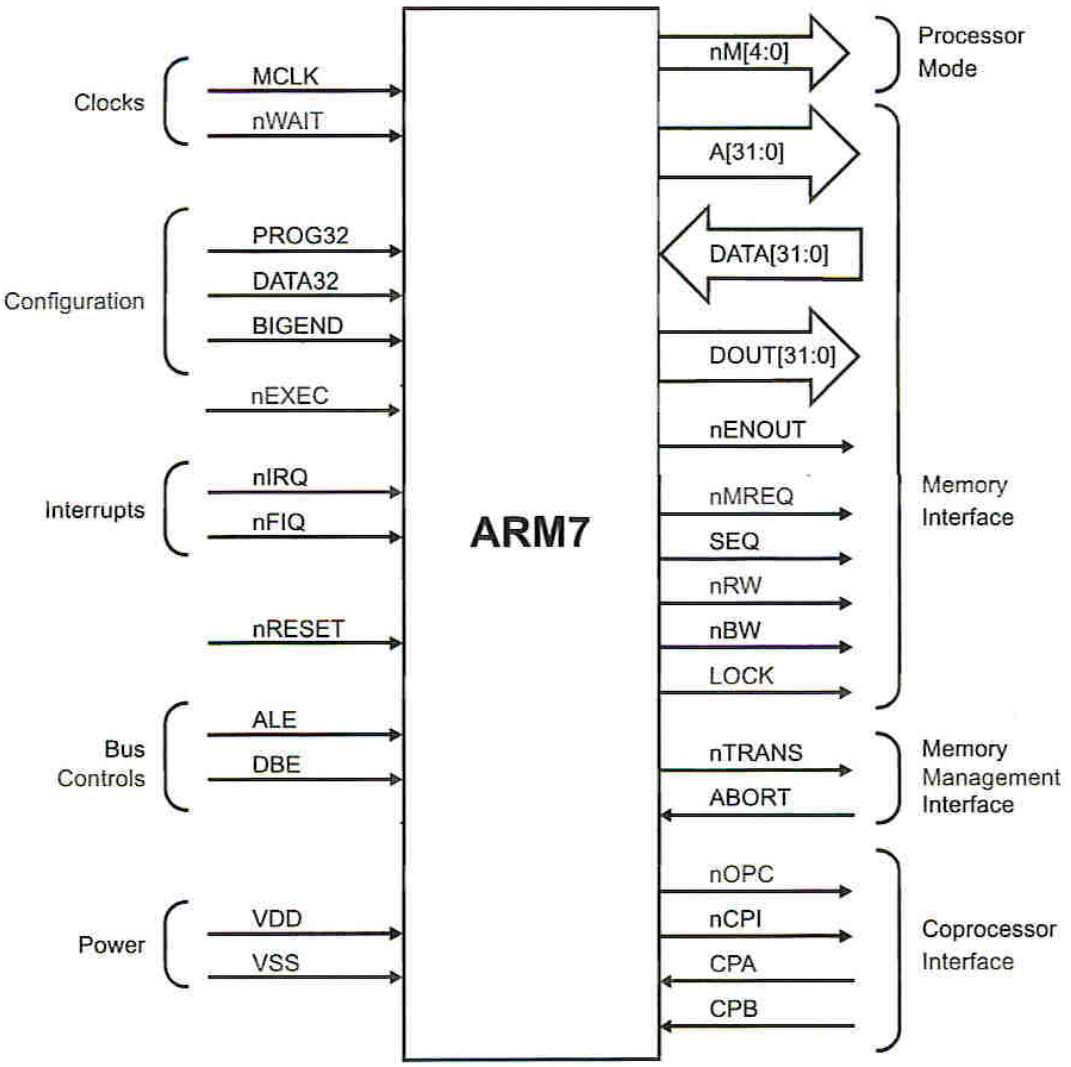


Figure 2: ARM7 Functional Diagram

2.0 信号描述

NAME	TYPE	DESCRIPTION
A[31:0]	O	处理器的地址总线，如果**ALE**(地址所存使能)为高，在要寻址的前一个周期的Phase2地址有效并保持至寻址周期的Phase1，保持时间由ALE控制。
ABORT	I	存储器异常。这是一个输入信号，存储器系统发出此信号告诉处理器请求的访问不被允许。
ALE	I	地址锁存使能，用来控制地址输出锁存，通常地址在周期的Phase2变成下一个周期需要的值，但是对ROM的直接接口来讲，地址需要保持稳定至Phase2结束，所以ALE需要保持低电平至Phase2。此信号对以下的信号有相同的影响，**nBW**, **nRW**, **LOCK**, **Nopc**和**nTRANS**。如果一个系统不需要地址线以这种方式工作，ALE必须拉高，地址锁存是静态的，所以ALE必须保持长时间低电平，以锁存地址。
BIGEND	I	Big Endian 配置。为高电平时，处理器认为存储器中数据为 Big Endian模式，为低电平时，认为存储器中数据为Little Endian模式。那些不可以选择Endianism的处理器（ARM2As,ARM3,ARM61）为Little Endian模式。
CPA	I	协处理器Absent。可以处理ARM7请求的协处理器（通过**nCPI**）必须立即将**CPA**置为低电平。如果在**nCPI**为低电平的周期的Phase1结束时，**CPA**仍为高电平，ARM7将中止与协处理器握手，启动无定义指令陷阱。如果**CPA**为低电平，并且继续保持，ARM7将等待，直到**CPB**为低电平，然后完成协处理器指令。
CPB		协处理器忙。可以处理ARM7请求的协处理器（通过**Ncpi**），但不能立即开始响应，应该通过将**CPB**置高表明此情况。当协处理器准备好时，置**CPB**为低电平。当**nCPI**为低电平时，ARM7在每个周期的Phase1结尾采样CPB。
DATA[31:0]	I	数据输入总线。在读周期（nRW =0），此数据输入线必须保证在传输周期的Phase2结尾之前保持有效。
DATA32	I	32位数据配置。当此信号为高时，处理器可以访问32位地址空间（**A[31:0]**）的数据，当此信号为低时，处理器可以访问26位地址空间（**A[25:0]**）的数据。在后一种配置中，地址线**A[31:26]**没有用到。在改变此信号电平之前，要确保处理器在下一个周期不会访问到大于0X3FFFFFF的空间。
DBE	I	数据总线使能。当**DBE**为低电平时，写数据BUFFER被禁止。当**DBE**为高电平时，写数据BUFFER在下一个真正的写周期时可以被使能，DBE促进了数据总线共享（DMA 或其它）。

DOUT[31:0]	O	数据输出总线，在写周期(when **nRW** = 1)的Phase1数据有效，并保持至传输周期的Phase2结束。
LOCK	O	Lock操作。当**LOCK**为高时，处理器执行一个 Locked 的MEM访问，MEM控制器必须等到LOCK为低时，才允许其它设备访问MEM。当MCLK为高电平时，LOCK信号改变，并在被锁的MEM访问周期一直保持高电平。只有执行数据交换指令(SWP)时，才被激活。此信号的时序可以通过**ALE**改变，就像ALE改变地址一样。驱动**ABE**为低电平，LOCK为高阻态。
MCLK	I	MEM时钟输入。此信号定时ARM7所有MEM访问和内部操作。此信号有两个不同的相位。Phase1:MCLK为低，Phase2：MCLK（和Nwait）为高。此信号可以在任何一相被任意延伸，用于访问比较慢的外设或MEM。与此相似, Nwait输入也可以和一个正常运行的MCLK相与取得相同的效果。
nBW	O	字节或者字使能。处理器输出此信号告诉外部MEM系统，当前数据传输为字节长度。在读或者写周期，高电平表示字传输，低点平表示字节传输。此信号在数据传输的前一个周期的Phase2有效，并在本次数据传输周期的整个Phase1有效。此信号的时序像地址线一样可以通过ALE信号来改变。当ABE信号位低时，此信号为高阻态。
nCPI	O	协处理器指令。当ARM7执行一条协处理指令，此信号将被置为低电平，并等待协处理器响应。ARM7将依据协处理器的CPA 和 CPB的不同状态，做出相应的反应。
nENOUT	O	数据输出使能。处理器通过此信号表明写周期正在发生，DOUT[31:0]被送到外部的MEM系统。如果系统需要双向数据总线，此信号可以使能三态Buffer， DOUT[31:0]通过三态Buffer送至DATA[31:0]
nENIN	I	输入使能。在写周期，此信号和nENOUT一起控制数据总线。请参考第五章：MEM接口。
nFIQ	I	快速中断请求。对处理器来讲，为异步中断请求，当此信号为低电平，并且处理器内部的相应的使能信号有效时，处理器被中断。此信号为电平敏感，并且在处理器做出响应之前应保持低电平。
nIRQ	I	中断请求。像nFIQ，但优先级较低。当使能时，产生异步中断。
Nm[4:0]	O	处理器模式。输出信号为内部状态位的反向，表示处理器的操作模式。

NMREQ	O	MEM请求。此信号位低电平时，表示下一个周期，处理器请求MEM访问。此信号在 Phase1时有效，至访问周期的整个 Phase2。
NOPC	O	OP代码取。此信号位低时，表示处理器正从MEM中预取指令；为高时，表示数据正在传输（如果总线上有数据），此信号在前一个周期的 Phase2有效，至访问周期的整个 Phase1。此信号的时序像地址线一样可以通过ALE信号来改变。当ABE信号位低时，此信号位高阻态。
nRESET	I	复位。此为电平敏感输入信号。从一个已知地址启动处理器。低电平将会导致正在执行的指令被异常中止。nRESET为高电平至少一个周期之后，处理器将从地址0 重新启动。nRESET必须保持低电平（nWAIT必须为高）至少两个周期。在低电平周期，处理器会执行虚拟的指令预取，地址为复位被激活时的地址增加，如果 nRESET继续保持，地址超过最大限制时，溢出为0。
nRW	O	读/写。高电平表示处理器写周期，地电平表示读周期。此信号应该在前一个周期的Phase2 有效，并保持到访问周期的Phase1结束。此信号的时序像地址线一样可以通过ALE信号来改变。当ABE信号位低时，此信号为高阻态。
nTRANS	O	MEM传输。低电平时表示处理器在用户模式，告诉MEM管理硬件，何时应该打开调用地址，或者非用户模式激活的指示器。此信号的时序像地址线一样可以通过ALE信号来改变。当ABE信号位低时，此信号位高阻态。
nWAIT	O	等待。当访问较慢外设时，通过驱动nWAIT为低，使ARM7可以进入长达数个MCLK周期的等待状态。在ARM7内部，Nwait和MCLK相与，并只有当MCLK为低时nWAIT状态才改变。如果nWAIT不用，必须上拉。
PROG32	I	32 位的程序配置。当此信号为高时，处理器可以从32位的地址空间取指，用地址线A[31:0];为低时，处理器从26位的地址空间取指，用地址线A[25:0]。在后一种配置中，指令预取没有用到地址线A[31:26]。在改变**PROG3**2,之前，必须保证处理器在26位模式，并且下一个周期不会写地址空间0 到0X1F(包括)。

SEQ	O	顺序地址。当下一个MEM周期的地址同上一个周期的地址相关联时，此信号为高。新地址将和旧地址一样或者在当一个地址的基础上加四。 在地址可以预料的周期之前，此信号在 phase1 有效，并在 phase2 保持。此信号可以和地位地址线一起应用，表示下一个周期为快速MEM模式（DRAM 页模式），或者旁路地址转换系统。
VDD	P	电源。
VSS	P	GND

表1:信号描述

I: INPUT
O : OUTPUT
P: POWER

3.0 编程模式

ARM7支持不同的操作配置。一种为硬件配置，由输入信号控制。其他为软件控制，称为操作模式。

3.1 硬件配置信号

ARM7提供了三个硬件配置信号，处理器正在运行的时候可以改变，具体如下：

3.1.1 Big and Little Endian(bigend 位)

ARM7 依据BIGNED 信号为高或者低，将MEM中的数据当作Big Endian 或 Little Endian 格式，MEM 被看作字节线性存储，序号从0向上排列，字节0到3为第一个存储字，4 到 7为第二个，以此类推。

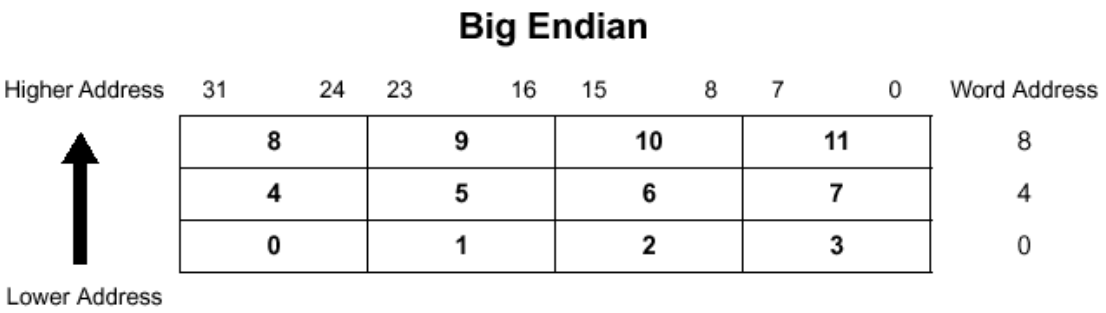
在 Little Endian格式，一个字的最小号字节被当作最低字节，最大号字节被当作最高字节。字节0将被连结到数据线D[7：0]。

最低字节在最低地址

字通过最低字节的地址寻址

图3：Little Endian 格式的地址

在Big Endian 格式下，最高字节被存在MEM中的最低地址，最低字节被存在MEM的最高地址。MEM中字节0对应数据线D[31:24]，endianism只影响加载和存储指令。参考4.7.3 第37页。



最高字节在最低地址

字通过最高字节的地址寻址

图4：Big Endian 地址格式

3.1.2 向下兼容的配置位

其它两个输入PROG32, DATA32用于和早期的ARM处理器兼容（参考第十章），但通常应该置 1 ，这种配置扩展地址空间到32位，编程模式的主要改变如下面所介绍，在32位的环境下支持26位程序。推荐用32 位模式编码，以便和将来的ARM处理器兼容。

为了适应32位操作，原始的ARM指令集已被改变，程序员必须注意一些附加的限制条件，这些限制条件在下面的文章里有提示。也可参考ARM应用注意事项 *“Rules for ARM Code Writers ，” “Notes for ARM Code Writers* 。

3.2 操作模式选择

ARM7有32位地址和32位数据线。处理器支持字节（8位），字（32位）数据类型，字必须四个字节对齐。指令按字执行，数据操作（比如加法）以字为单位执行，加载和存储操作可以为字或字节传送。

ARM7支持六种操作模式：

（1） 用户模式（usr）：正常的程序执行状态
（2） FIQ模式(fiq)： 支持数据传送或通道处理
（3） IRQ模式(irq)： 用于通用的中断处理
（4） 管理模式(svc)：用于操作系统的保护模式
（5） 异常模式(abt)：数据或者指令预取异常时进入
（6） 无定义模式(und)：当无定义指令被执行时进入
（7）
软件控制，外部中断，异常处理都可以改变操作模式。大部分的应用程序在用户模式下执行。其他模式，比如管理模式，在中断、异常服务、或者访问被保护资源时进入。

3.3 寄存器

ARM7处理器共有37个寄存器，包括31个(32位)通用寄存器，6个状态寄存器。在任意时候，16个通用寄存器（R0-R15）和一个或者两个状态寄存器对处理器来讲是可见的。可见寄存器依据处理器模式和其它寄存器（banked registers）将被切换去支持IRQ, FIQ,管理，异常，无定义模式的处理。异常模式，非定义模式。图5显示了寄存器Bank组成，banked 的寄存器在图中用阴影显示。

在所有的模式中，16个寄存器（R0-R15）都可以直接访问。除过R15,其他的寄存器都是通用寄存器，可以用来保存数据和地址。R15保存可编程计数器PC的值，当R15被读时，位[1：0]为'0'，位[31：2]保存计数器的值。第十七个寄存器（CPSR-当前程序状态寄存器）也是可以访问的。它包括了条件代码标志位和当前模式位，可以当作是PC 的扩展。

R14可以当作子程序连接寄存器，当执行一个分支或者连接指令时，可以保存R15的值。在其它情况下，可以当作通用寄存器用。R14_svc, R14_irq, R14_fiq, R14_abt, R14_und与R14相似，当中断或者异常发生时，或者是中断和异常程序中，分支和分支连接指令被执行时，可以保存R15 的返回值。

通用寄存器和可编程计数器模式

User32	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

可编程状态寄存器

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

图5：寄存器组织

FIQ模式有7个Banked 寄存器，映射到R8-14(R8_fiq- R14_fiq)。有些FIQ程序不需要存储任何寄存器。用户模式，IRQ模式，管理模式，无定义模式都有两个Banked寄存器，映射到R13和R14。这两个Banked寄存器允许每个模式有自己的堆栈指针和l链接寄存器。管理模式，IRQ模式，异常模式和无定义模式的程序（需要不止两个Banked寄存器）能够在它们自己的堆栈中存储部分或者全部的调用程序的寄存器（R0-R12），这样就可以无限制地用这些寄存器（R0-R12），这些寄存器在调用程序返回之前被恢复。另外，有5 个SPSRs寄存器（存储可编程状态寄存器），当异常发生时拷贝 CPSR。每一个特权级模式有一个SPSR。

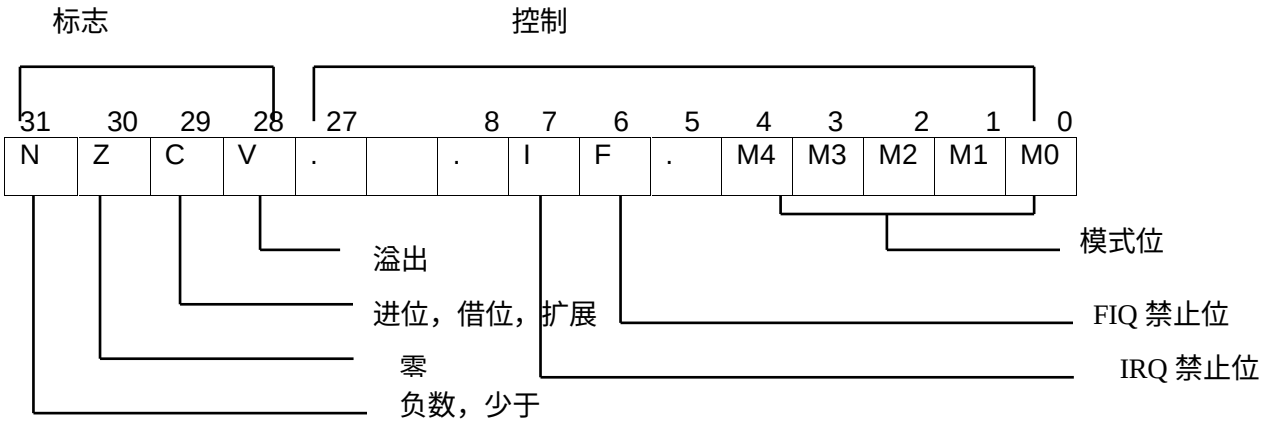


图6 ： 可编程状态寄存器格式

可编程状态寄存器的格式在图6中显示，N,Z,C,和V位条件代码标志位。当作处理器的算术或者逻辑运算结束时这些条件代码标志位可能被改变，也可以在程序中以此作为条件来判断指令是否被执行。

I 和 F 位为中断禁止位。当I置'1'时，禁止IRQ中断，当F置'1'时，禁止FIQ中断。M[4:0]为模式位，这些位决定了处理器的操作模式。表2解释了模式位M[4:0]。并不是所有的组合都表示了有效的操作，只有那些明确定义的组合才有效。

PSR寄存器的低28位（包括I,F,M[4：0]）全部作为控制位。如果有异常产生这些控制位将改变，或者在管理模式，通过软件改变控制位。没有用到的位为保留位，它们的状态在在控制位和标志位改变时可以保留。当检查PSR状态时，程序并不需要依靠保留位的值来判断，因为它们在将来的处理器中中可能为'1'或者'0'。

M[4:0]	模式	可访问的寄存器	
10000	用户	PC,R14…R0	CPSR
10001	FIQ	PC,R14_fiq…R8_fiq,R7…R0	CPSR,SPSR_fiq
10010	IRQ	PC,R14_irq…R13_irq,R12…R0	CPSR,SPSR_irq
10011	管理	PC,R14_svc…R13_svc,R12…R0	CPSR,SPSR_svc
10111	异常	PC, R14_abt..R13_abt, R12..R0	CPSR, SPSR_abt
11011	无定义	PC, R14_und..R13_und, R12..R0	CPSR, SPSR_und

表2 模式位定义

3．4 异常

当正常的程序流程需要被中断时，异常产生，使得处理器可以执行外设产生（例）的中断。进入异常处理之前，处理器的状态必须保存，以便异常处理完之后源程序可以继续执行。可能会有几个异常同时产生。

ARM7处理异常时，通过banked寄存器保存状态。PC和CPSR寄存器中的内容被拷贝到相应的R14和SPSR中，PC 和CPSR中的模式位M[0:4]根据异常类型被强制改变。在异常断处理中，如果要禁止其它难以管理异常的嵌套，可以置位中断禁止标志位。在需要重复进入中断处理程序的情况下，R14和SPSR应该在中断被使能之前保存到主MEM的堆栈中；当传送SPSR寄存器到堆栈或者从堆栈传送时，需要注意的是必须传送整个字（32位），而不只是标志位或者控制位。当多个异常同时产生时，固定的优先级决定异常响应的顺序。优先级将在以后章节中说明。

3.4.1 FIQ

FIQ（快速中断请求）异常是由外部产生，将**Nfiq**输入拉低。此输入可以接受异步转换，在影响处理器的执行流程之前，会被延迟一个时钟周期，以便与处理器保持同步。此设计支持数据传输或者通道处理，有足够的私有寄存器转移在某些应用中需要保存的值（最小化切换成本）。可以置位在CPSR中的F标志位屏蔽FIQ异常（除过用户模式），如果F位清零，ARM7在每一条指令结束时会检查FIQ同步器的输出是否为低电平。

当检测到有FIQ低电平输入时，ARM7执行以下步骤：

（1） 将下一条将要执行的指令地址+4，保存到R14_fiq，保存CPSR的值到SPSR_fiq
（2） 强制M[4:0]=10001(FIQ模式)，置位CPSR中的F,I
（3） 强制PC从地址0X1C取下一条指令

为了从FIQ返回，执行 SUBS PC,R14_fiq，#4 将恢复PC(从R14)和CPSR(从SPSR_fiq)并继续执行的被中断的程序。

3.4.2 IRQ

IRQ（中断请求）为普通异常，当 **nIRQ** 输入位低电平时产生IRQ。它的优先级低于FIQ，并且当进入FIQ异常处理程序时，它被屏蔽。在任何时候将CPSR中I 位置1，都可以屏蔽此中断（用户模式除外）。当I 位清零时，，ARM7在每一条指令结束时会检查IRQ同步器的输出是否有低电平。当检测到有IRQ输入时ARM7执行以下步骤：

(1) 将下一条将要执行的指令地址+4，保存到R14_fiq，保存CPSR的值到SPSR_fiq
(2) 强制M[4:0]=10010(IRQ模式)，置位CPSR中的I
(3) 强制PC从地址0X18取下一条指令

为了从IRQ状态恢复正常，执行SUBS PC,R14_fiq，#4 将恢复PC(从R14)和CPSR(从SPSR_fiq)并继续执行的被中断的程序。

3.4.3 异常中断

异常中断由外部的ABORT输入产生。ABORT表示当前存储器访问没有完成。举例来讲，在虚拟存储器系统中，当前地址上相应的数据可能已经从存储器被移到disc，在访问完成之前，可能要求处理器动作以恢复这些数据。Arm7在存储器访问周期检查ABORT信号，如果有异常产生,ARM7将执行以下两种步骤之一：

（1） 如果异常在指令预取时产生（预取异常），预取的指令被标志为无效，但是异常中断并不立即产生。如果指令没有被执行，举例来讲，在流水线中的一个分支指令，将不会产生异常，如果指令到流水线的出口并将要被执行时，异常产生。

（2） 如果异常在数据访问时产生（数据异常），ARM7的动作将依靠指令类型

（A） 单次数据传输指令（LDR,STR）将回写改变过的基址寄存器，异常处理程序必须注意到这一点。
（B） 交换指令异常（SWP），虽然外部的读写访问可能发生，但就好像此指令没有被执行一样
（C） 块数据传输指令完成（LDM,STM），如果回写位置1，基址将改变。如果正常情况下，指令回写基址寄存器（LDM传输列表中的基址），则回写被禁止。当异常产生时，所有寄存器回写都被禁止。这就意味着当LDM指令异常时，R15（总是最后一个被传送）将保留。

在指令预取异常或数据异常发生，ARM7进行以下操作：

（1） 保存地址（异常指令地址 加4 for 指令预取异常，加8 for 数据异常）到R14_abt,保存CPSR到SPSR_abt
（2） 强制M[4:0]=10111(异常模式)，置位CPSR中的I
（3） 强制PC从地址0X0C(预取异常)或0X10(数据异常)取下一条指令

为了从异常返回，执行SUBS PC,R14_abt，#4 （预取异常）或SUBS PC,R14_abt，#8 (数据异常)，恢复PC和CPSR并继续执行被异常的指令。

当适当的MEM管理软件有效时，异常机制允许执行 需要的页式虚拟MEM系统 。处理器可以产生任意地址，如果地址中的数据无效时，MMU产生异常。处理器进入系统软件陷阱，系统软件必须找出异常原因，使请求的数据有效，重新执行被异常的指令。应用程序不需要知道它用到的有效MEM的总量，也不需要异常时的状态。

3.4.4 软件中断

软件中断指令（SWI）从其它模式进入管理模式。通常情况下，请求一个特殊的管理函数。当SWI指令被执行时，ARM7执行以下动作：

（1） 将SWI指令地址加4保存到R14_svc，保存CPSR的值到SPSR_svc
（2） 强制M[4:0]=10011(管理模式)，置位CPSR中的 I
（3） 强制PC在地址0X80取下一条指令

为了从SWI返回，执行指令MOVS PC,R14_svc，恢复PC和CPSR的值，返回到SWI的下一条指令。

3.4.5 无定义的指令陷阱

当ARM7遇到自己没有办法处理的指令（第四章：指令集），它将此指令提供给现场的协处理器。如果协处理器可以执行此指令，但当时又比较忙，ARM7将进入等待状态，直到协处理器准备好或者中断产生。如果没有协处理器可以处理此指令，ARM7将进入无定义指令陷阱。

当系统中没有协处理器硬件时，陷阱可以用来做协处理器的软件仿真，或者通过软件仿真用于通用指令集扩展。

当ARM7接收到无定义指令陷阱时，执行以下步骤：
（1） 将无定义指令或协处理器指令加4保存到R14_und;保存CPSR到SPSR_und.
（2） 强制M[4:0]=11011(无定义模式)，置位CPSR中的 I
（3） 强制PC在地址0X04取下一条指令

在仿真失败的指令之后，为了从陷阱返回，执行MOVS PC,R14_und,恢复CPSR并返回到无定义指令的下一条指令。

3.4.6 向量总结

地址	异常	进入模式
0X00000000	复位	管理模式
0X00000004	无定义指令	无定义
0X00000008	软件中断	管理
0X0000000C	异常（预取）	异常
0X00000010	异常（数据）	异常
0X00000014	保留	--
0X00000018	IRQ	IRQ
0X0000001C	FIQ	FIQ

表3： 向量总结

这些是字节地址，通常包括了指向相关程序的分支指令。

FIQ程序处于0X1C，从而不需要分支指令（和执行时间）。

3.4.7 异常优先级

当多个异常时产生时，固定的优先级系统决定异常处理的顺序：

（1） 复位（最高优先级）
（2） 数据异常中断
（3） FIQ
（4） IRQ
（5） 预取异常
（6） 无定义异常，软件中断（最低优先级）

注意，不是所有的异常都会立即产生，无定义指令和软件中断互为排斥，因为它们都依靠当前指令的特定译码（不能重叠）。

如果数据异常和FIQ同时发生，且FIQ被使能（CPSR中F标志位为0），ARM7将进入数据异常处理，然后立即进入FIQ中断向量处理，从FIQ的正常返回将使数据异常处理器继续运行。数据异常优先级高于FIQ，保证能够检测到传输错误，计算数据异常响应时间应该加上最差的情况下FIQ处理的时间。

3.4.8 中断响应时间

假设FIQ使能，对FIQ来讲，最长的延时，包括请求被传送到同步器的最长时间，（Tsyncmax）加上最长的指令完成需要的时间（Tidm,最长的指令是LDM，它加载所有的寄存器，包括PC），加上数据异常响应时间，（Texc），加上FIQ入口时间（Tfiq）。在这些时间之后，ARM7将从0X1C取指执行。

Tsyncmax为3个机器周期，Tidm为20 个机器周期，Texc为3个机器周期，Tfiq为2个机器周期,所以总的时间加起来为28个周期。在用20MHz的处理器时钟的系统中，花费1.4Us。最大的IRQ延时时间计算方法类似，但是，必须考虑到这样一个事实，FIQ优先级比RIQ要高，进入RIQ的处理程序入口可能被延长任意时间，对FIQ,RIQ来讲最小的延迟由经过最少时间请求由同步器送到（Tsyncmax），加上FIQ入口时间（Tfiq），大约为四个机器周期。

3.5 Reset

当复位信号位低电平时，ARM7放弃正在执行的指令，继续从字地址加一取指。

当nRESET再次位高时，ARM7执行以下步骤：

（1） 用当前PC,CPSR中的值覆盖R14_svc和SPSR_svc，被保存的PC,CPSR的值没有定义。
（2） 强制M[4:0]=10011(管理模式)，将CPSR中的F,I 值位。
（3） 强制PC从地址0X00取出下一条指令。

4.0 指令集

4.1 指令集总述

图7显示了ARM7的指令集。

注意：一些指令代码没有定义，但不会导致无定义的指令陷阱产生，这些指令没有用到，因为在将来的ARM应用中，它们的含义可能会改变。

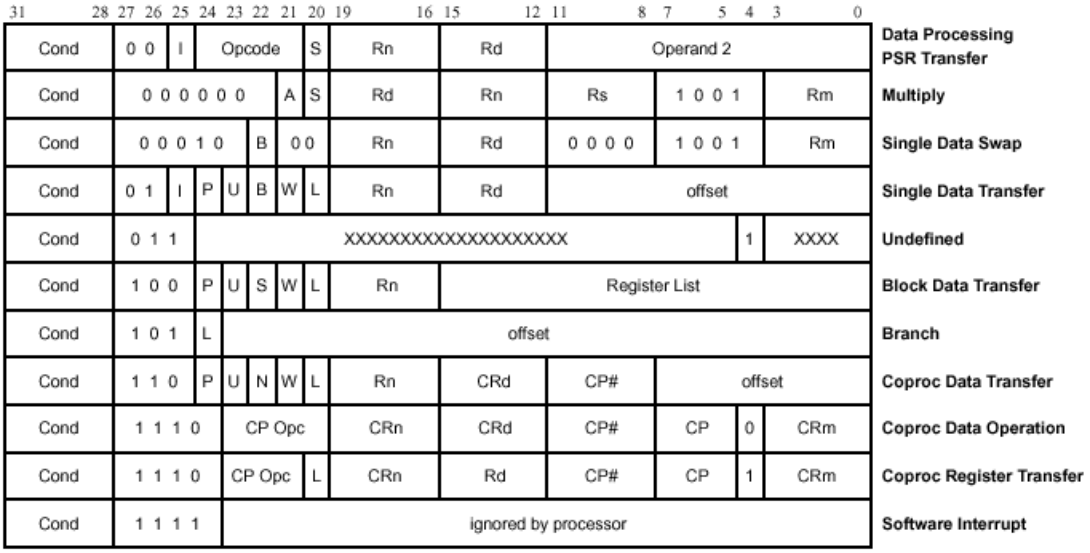


Figure 7: Instruction Set Summary

4.2 条件代码

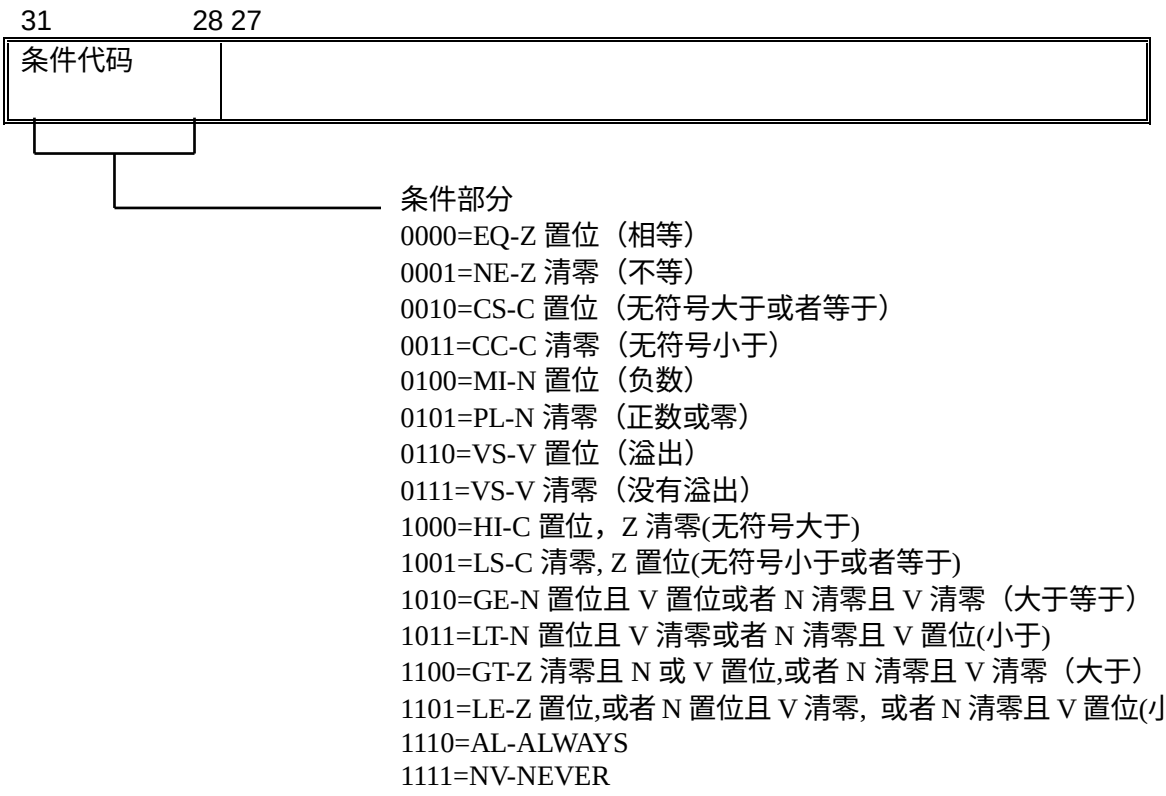


图8：条件代码

所有ARM7的指令都是条件满足时才执行，意味着指令的执行或不执行跟CPSR中的N,Z,C,V标志位有关系，图8 显示了这些条件代码。

如果指定Always（AL），指令将会无条件执行，而不管标志位的值如何。NEVER（NV）没有用到，因为在将来的ARM结构中会被重新定义。如果需要延时，建议用MOV R0，R0，如果没有指定条件代码，编译器默认为Always（AL）。

其它的条件代码已经在图8中详细说明。比如代码0000（Equal）表示只有在标志位Z置位时，指令才会被执行。相当这样一种情况，比较指令（CMP）执行后，两个操作数如果相等，则置位Z，如果不相等，比较指令清零Z标志位，并且指令不会执行（条件不满足）。

4.3 分枝和分支连接 (B，BL)

此指令只有当条件为真时，才会被执行。在本章开始时定义了不同的条件。分支指令代码在图9显示。

分支指令包含了有符号的24位偏移量。左移两位，然后将符号位扩展到32位，加到PC。因此，此指令可以指定+/-32M字节的地址。分支偏移必须考虑预取操作，因为指令预取将导致PC为当前指令地址前两个字（8字接）。

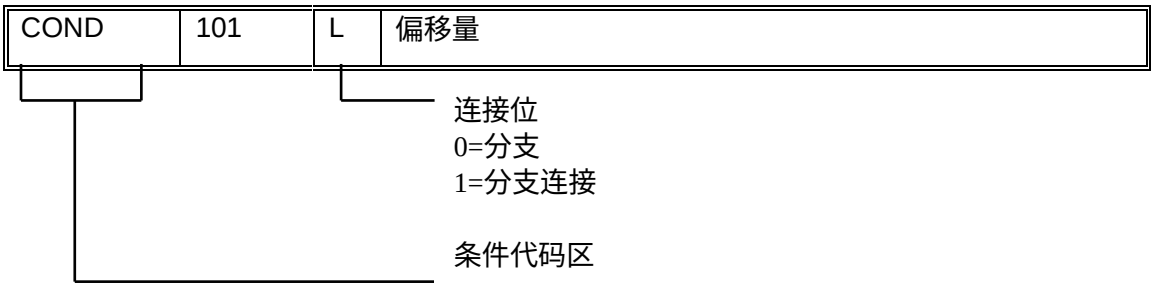


图9 分支指令

分支地址超过+/-32M字节，必须用到偏移量或者已经被预先载入寄存器的绝对目的地址，在这种情况下如果需要一个分支连接操作，PC应该手动保存到R14。

4.3.1 连接位

分支联接(BL)写旧的PC的值到当前bank的连接寄存器R14中。考虑到指令预取，写到R14的PC值是可以调节的，并包含分支联接指令的下一条指令的地址。CPSR没有被PC保存。

为了从被调用的分支联接程序返回，用指令 MOV PC,R14（如果连接寄存器一直有效），或者LDM Rn!,{..PC} （如果连接寄存器已经被保存到由Rn指定的堆栈中）。

4.3.2 指令周期

分支和分支联接指令会花费2S+1N incremental 周期，S,N将在5.1周期类型定义。

4.3.3 汇编语法

B{L}{条件}<表达式>

{L} 用于请求指令的分支联接格式，如果此项缺，R14将不受此指令影响。

{Cond} 如图8 所示，（EQ,NE,VS 等），如果此项缺，编译器会认为AL(Always)无条件执行指令。

<表达式> 目的地址，编译器计算偏移量。

{}为可选项，<>为必须项。

4.3.4 举例

```
here   BAL     here ;      AL-Always
       B       there ;     默认ALways
       CMP     R1,#0 ;     比较R1和0，如果相等，转倒fred，否则，继续
       BEQ     fred ;      执行下一条指令
       BL      sub+ROM ;   调用子程序，在地址sub+ROM
       ADDS    R1,#1 ;     寄存器R1加1， 根据结果设置CPSR的标志位，如果
                           C被清零，调用子程序。
       BLCC    sub ;
```

4.4 数据处理指令

指令只有在条件满足时才会执行，图10显示指令编码。

指令在两个或一个操作数之间执行指定的算术或逻辑运算，第一个操作数一般为寄存器(Rn),第二个操作数根据指令I 位的值，可以是可移位寄存器，也可以是循环的8 位立即数。根据指令中的S位， CPSR中的条件代码可以被保留或者更新（依据指令执行的结果）。一些操作（TST,TEQ,CMP,CMN）不会写结果到Rd寄存器中，它们只是用于测试，并根据指令执行结果设置条件代码，并且s位总是置'1'。这些指令和执行结果在表4 中列出。

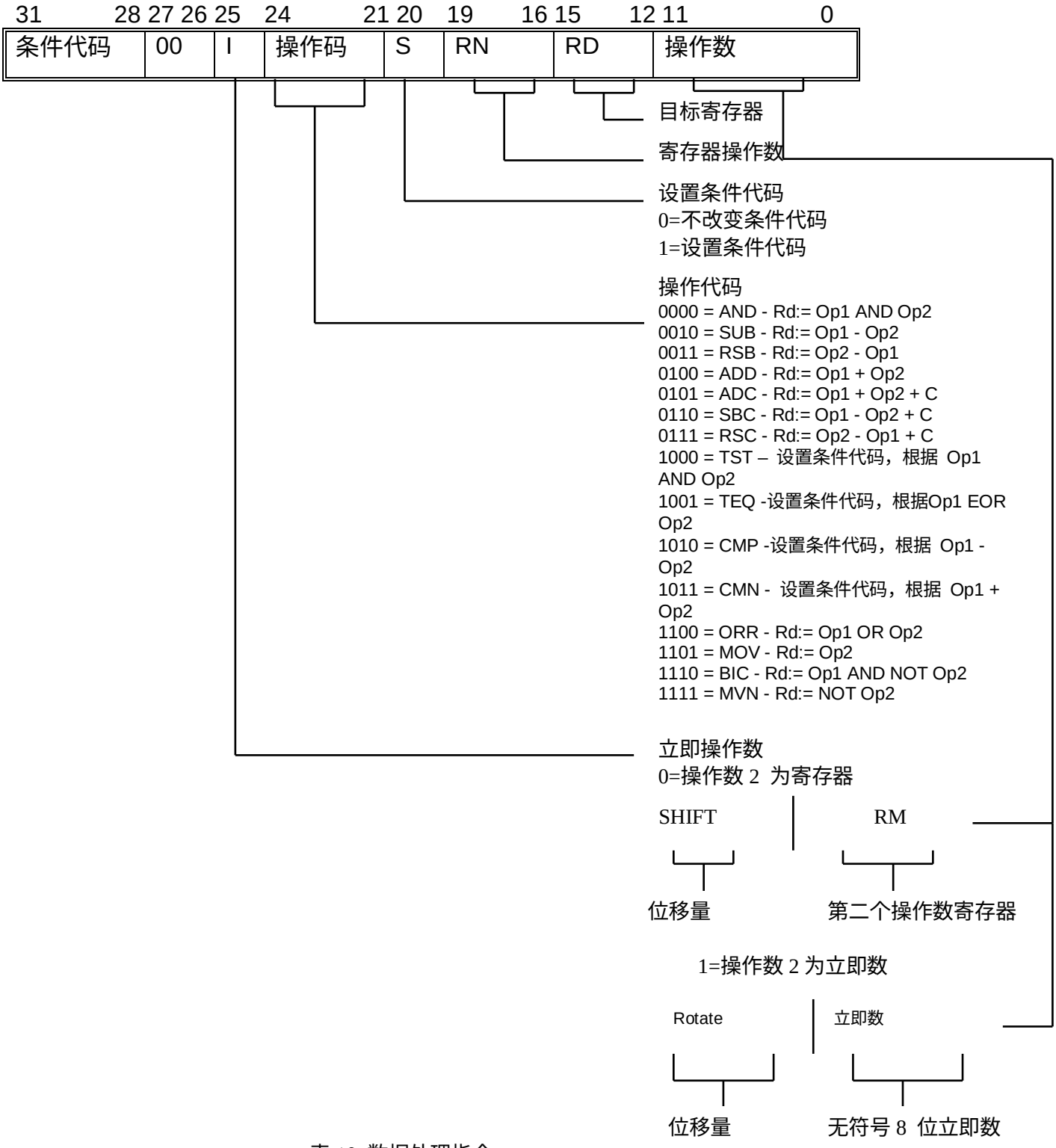


表 10 数据处理指令

4.4.1 CPSR标志位

数据处理操作可以归纳为逻辑和算术运算，逻辑运算（AND,EOR,TST,TEQ,ORR,MOV,BIC,MVN）对操作数相应的位或者整个操作数进行逻辑运算，如果S位置'1'（Rd不是R15）,CPSR中的标志位V不会受影响， C 标志位为桶形移位器的进位输出（或者保持不变，执行指令LSL #0），当且仅当执行结果所有的位为0，Z标志被置 1 ， N 标志为运算结果的第31位。

指令	操作码	
AND	0000	操作数1与操作数2
EOR	0001	操作数1EOR与操作数2
SUB	0010	操作数1-与操作数2
RSB	0011	操作数2-与操作数1
ADD	0100	操作数1+与操作数2
ADC	0101	操作数1+与操作数2+进位
SBC	0110	操作数1-与操作数2+进位-1
RSC	0111	操作数2-与操作数1+进位-1
TST	1000	和AND操作相似，结果不写入
TEQ	1001	和EOR操作相似，结果不写入
CMP	1010	和SUB操作相似，结果不写入
CMN	1011	和ADD操作相似，结果不写入
ORR	1100	操作数一或操作数2
MOV	1101	操作数2（操作数一被忽略）
BIC	1110	操作数一与操作数2 的非（BIT CLEAR）
MVN	1111	操作数2 取反（操作数一被忽略）

表4：ARM数据处理指令

ARM算术运算指令（SUB,RSB,ADD,ADC,SBC,RSC,CMP,CMN）将每一个操作数当作32位的整数(有符号和无符号结果一样)。如果S 位置 1 (且Rd不是R15)，运算结果的第31位有溢出产生，则CPSR中的V标志位将会被置 1 ，如果两个操作数为无符号数，则可以忽略，但如果操作数为有符号数，编译器会报出 可能产当两个操作数都为无符号数生错误 的警告。C 标志位为 ALU 运算结果的第31的进位输出，当且仅当执行结果的所有位为0，Z标志被置 1 ，N 标志为运算结果的第31位（如果操作数为有符号数，表明运算结果为负数）。

4.4.2 移位

如果第二个操作数指定为可移位寄存器，桶形移位器的操作由指令中的移位区域控制。此区域说明了被执行的移位操作的类型（逻辑左或右，算术右或循环右）。寄存器的位移量在指令的立即数区表明，或者是另一寄存器（不包含R15）的低字节，图11 显示了不同类型移位的编码。

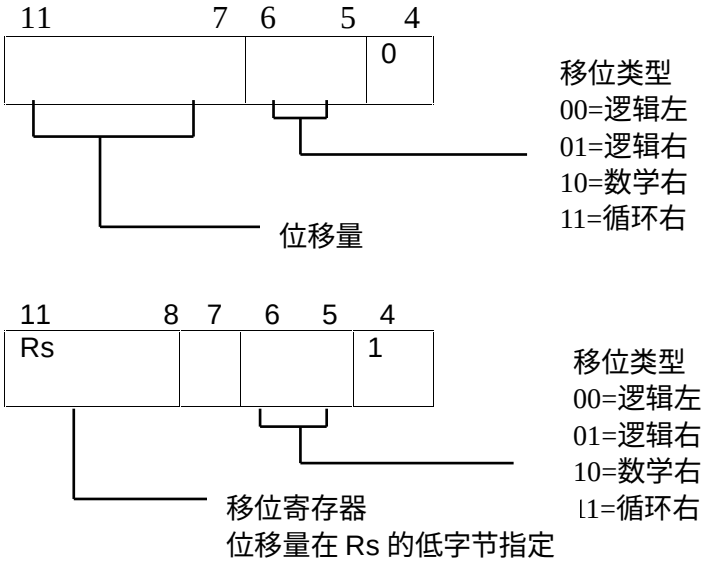


图11 ARM移位操作

指令指定的位移量

当位移量在指令中指定，它包含5位，可以是从0 到31的任意数。逻辑左移（LSL）将Rm中的数值左移指定的位数，右边空位补0，左边丢掉多余的位，最后一位将要被丢掉的位为移位寄存器的进位输出，当ALU操作为逻辑类型时，输出位被锁存到CPSR寄存器的C标志。例：LSL #5 的运行结果在图12中显示。

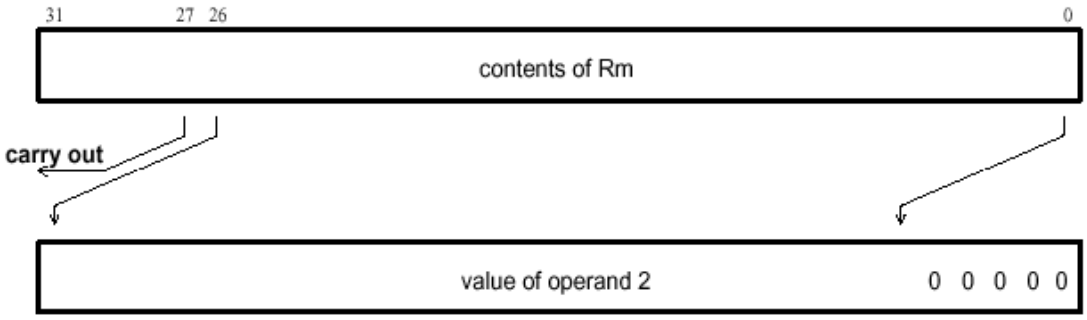
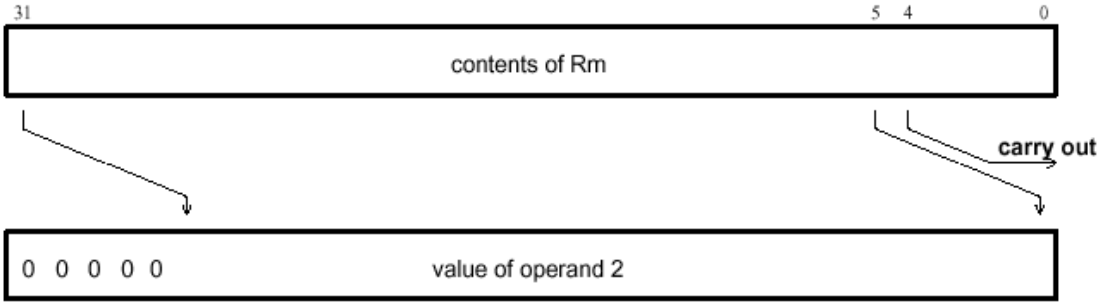


图12：逻辑左移

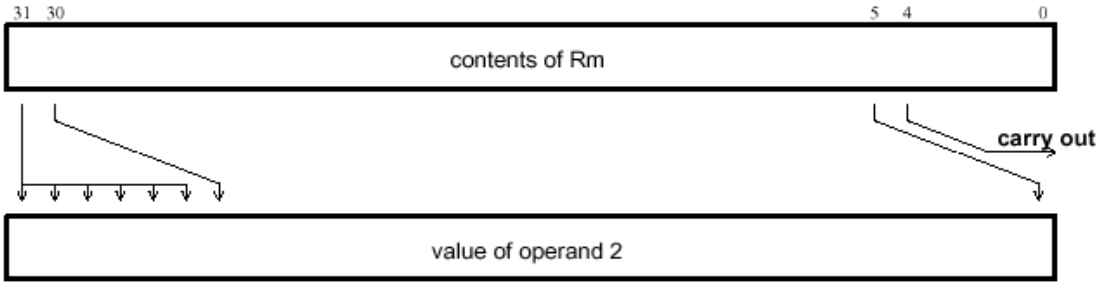
注意：LSL #0 是一种特殊情况，移位寄存器的进位输出是CPSR中原有的C标志，Rm中的值直接作为第二操作数。

逻辑右移指令也如此，只是寄存器Rm中的值向右移，LSR #5运行结果如图13。



LSR #32 移位区域的格式同 LSR #0一样（结果为0，Rm的31位作为进位输出）。逻辑右移0位和逻辑左移0一样是多余的指令，编译器将LSR #0（ASR #0,ROR #0）当作LSL #0，允许操作LSR #32。

算术右移（ASR）同逻辑右移相似，只是是高位将由 Rm 的第31位填补而不是0，这样就保持符号位不变。例：ASR #5 的运行结果在图14显示。



算术右移

ASR #0 移位区域的格式同样用于ASR #32，Rm的31位用于进位输出，操作数2 的每一位等于Rm的第31位，结果根据Rm31位为全0或全1。

循环右移操作将逻辑右移操作中废弃的位重新应用，放到左边最高位。ROR #5运行结果如图15。

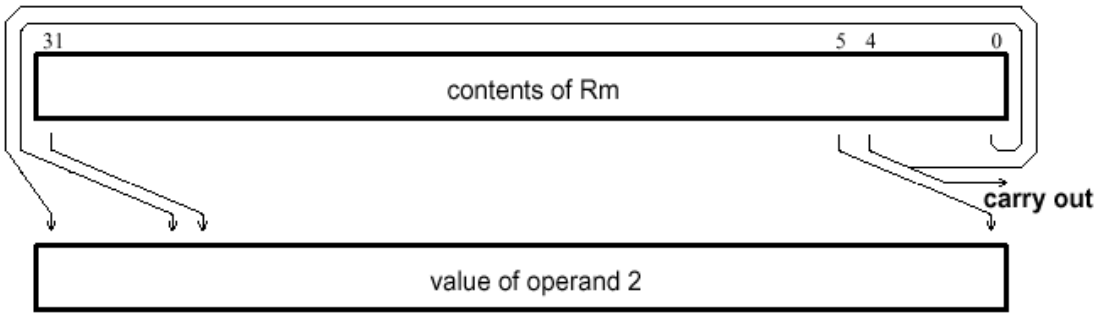
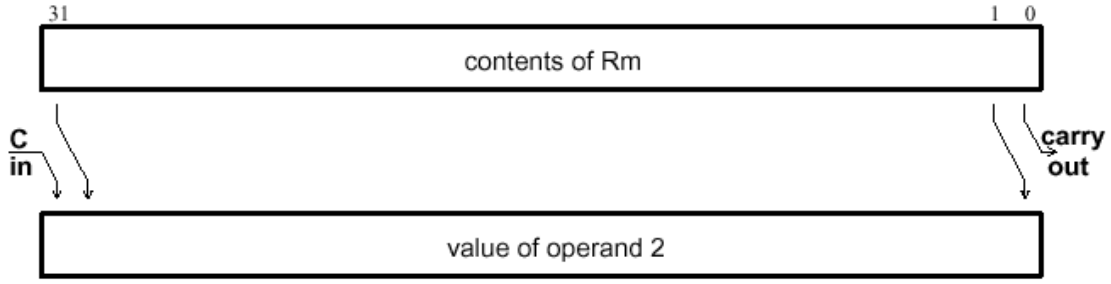


图15：循环右移

ROR #0 的移位区域的格式用于编码桶形移位器的特殊功能，带扩展的循环右移（RRX）。它将CPRS的C标志位当作最高位，和Rm中的值组成一个33位的操作数。



寄存器指定的位移量

Rs寄存器的最低字节指定了位移量，Rs为任意通用寄存器（不包含R15）。

如果此字节为0，Rm中的内容将毫无改变地应用到第二个操作数，CPSR 中原来的C标志值作为移位寄存器的进位输出。

如果这个字节的值在0到31之间，移位结果将同指令指定位移量（相同的移位操作，相同的位移量）的操作结果一样。

如果这个值为32或者更大的数，结果为上面描述的移位操作的逻辑扩展：

（1） LSL #32 结果为0，进位输出为Rm的最低位。
（2） LSL # 大于32的值，结果为0，进位输出为0。
（3） LSR #32 结果为0，进位输出为Rm的最高位。
（4） LSR #大于32，结果为0，进位输出为0。
（5） ASR 大于等于32，结果和进位输出为都为Rm的最高位。
（6） ROR #32 结果为Rm，进位输出为Rm的最高位。
（7） ROR #n，结果将同ROR #n-32 一样，重复地减去32，直到操作数结果在32同0 之间，运算结果同ROR #m 相同，m=n-x32(0<m<32)。

要注意在寄存器控制位移的指令中，指令第7位为0时非常必要的，如果为 1 ，将导致指令为乘法或者为无定义指令。

4.4.3 立即操作数循环

立即操作数的循环量是五位的无符号整数，它指定了8 位立即数的位移量，这个值可以从零扩展的到32，如果循环量为m,则立即数循环的结果为 立即数*2 m。

4.4.4 写R15

当Rd（目的寄存器）不包含R15,CPSR中的条件标志位就可以由ALU标志位修改，如上所述。

当Rd是R15，且指令中的S 位没有被置位，操作结果会被放到R15并且CPSR不受影响。

当Rd是R15，且指令中的S 位被置位，操作结果会被放到R15并且当前模式下SPSR保存到CPSR。这就允许状态改变（PC,CPSR的值自动恢复），这种指令格式在用户模式中不被用到。

4.4.5 R15作为操作数

如果R15（PC）在数据处理指令中被当作操作数，那么这个寄存器可以直接应用。

PC的值是指令的地址，依据指令预取加8 或者12字节。如果位移在指令中指定，PC是当前指令地址8个字节之前，如果位移量在寄存器中指定，PC将是当前指令地址之前12个字节。

4.4.6 TEQ,TST,CMP和CMN操作码

这些指令不写操作结果，但是修改CPSR的标志位，编译器自动置位S（对这些指令）即使指令中没有置位S。

以前处理器用到的TEQP指令格式没有用在32位模式，取而代之的是PSR传输指令。如果在这些模式中用到，处理器处于管理模式它的作用相当于保存 SPSR_<mode> 到CPSR，处于用户模式，则没有影响。

4.4.7指令时间周期

数据处理指令依照incremental周期数不同时间也不同，如下所示：

正常数据处理	1S
寄存器指定位移量的数据处理	1S+1IData Processing with PC written2S + 1N
寄存器指定位移量的数据处理和PC写指令	2S + 1N + 1I

4.4.8 汇编语法

（1） MOV,MVN- 单操作数指令
<操作码>{条件代码}{S}Rd,<操作数2>

（2） CMP,CMN,TEQ,TST 不改变执行结果
<操作码>{条件代码}{S}Rd,Rn,<操作数2>

（3） AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
<操作码>{条件代码}{S}Rd,Rn,<操作数2>

操作数2可以是Rm，<shift>或者<#expression>

{条件代码}-参考图8

{S}-如果S存在，则需要设置条件代码（用于CMP,CMN,TEQ,TST）

如果用到<表达式>，编译器将会尝试产生一个移位的8 位立即数与表达式匹配，如果尝试失败，将报错。

<shift> is <移位类型> <寄存器> or <移位类型> #expression, or RRX (带扩展的循环右移) 。

<移位类型>是ASL,LSL,LSR,ASR,ROR.(ASL 和LSL同义，它们被编译为同一代码)

4.4.9 举例

```
ADDEQ    R2,R4,R5          ；如果Z标志位为1，R2:=R4+R5;
TEQS     R4,#3             ；测试R4是否等于3（编译器自动加入S,所以S为冗余）
SUB R4,R5,R7,LSR R2        ；将R7中的值逻辑右移，位移量包含在R2的低字节，从R15
                           ；中减去移位结果，并送入R4
MOV      PC,R14            ；从子程序返回
MOVS     PC,R14            ；从异常返回，从SPSR模式释放CPSR
```

4.5 PSR传输（MRS,MSR）

此指令只有在条件为真时才执行。不同条件已经在本章开始时定义。

MSR和MRS指令由数据处理操作的子集构成，经常和没有设置S标志位的指令关联（TEQ,TST,CMN,CMP），编码在图17显示。

此指令允许访问CPSR,SPSR。MRS指令允许转移CPSR 或SPSR到通用寄存器中，同样MSR允许转移通用寄存器的内容到CPSR或SPSR.。

MSR指令允许立即数或寄存器的值传送到CPSR或SPSR的条件代码标志位（N,V,Z,C），但不影响控制位。在这种情况下，指定寄存器的高4位或32位立即数被写入到相关的PSR的高4位。

4.5.1操作数制

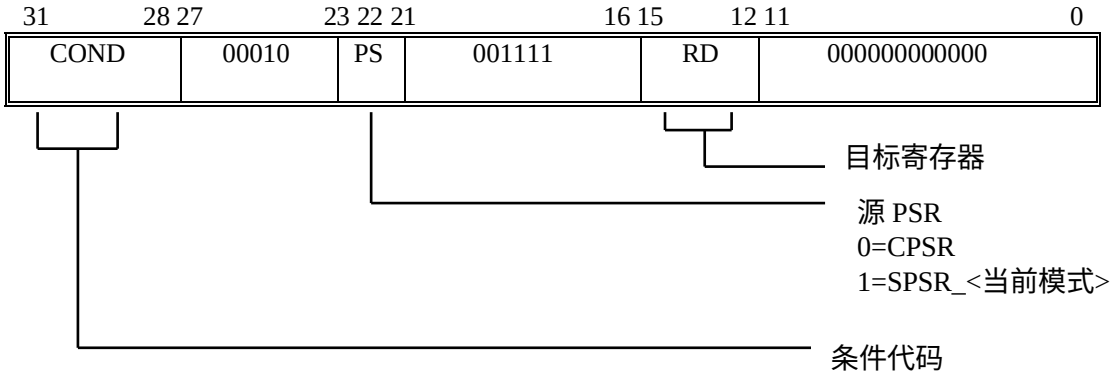
在用户模式，CPSR的控制位被保护而不能改变，所以只有CPSR的条件代码标志位可以改变，在其它的模式（管理模式），整个CPSR都可以改变。

SPSR可否访问要看正在执行的模式，比如只有在FIQ模式下SPSR_fiq才可以访问。

R15不能被指定为源或者目的寄存器。

在用户模式下不要试图访问SPSR,因为此寄存器根本不存在。

MRS（传递PSR的内容到寄存器）



MSR(传送寄存器的值到PSR)

MSR（仅传送寄存器的值或立即数到PSR的标志位）

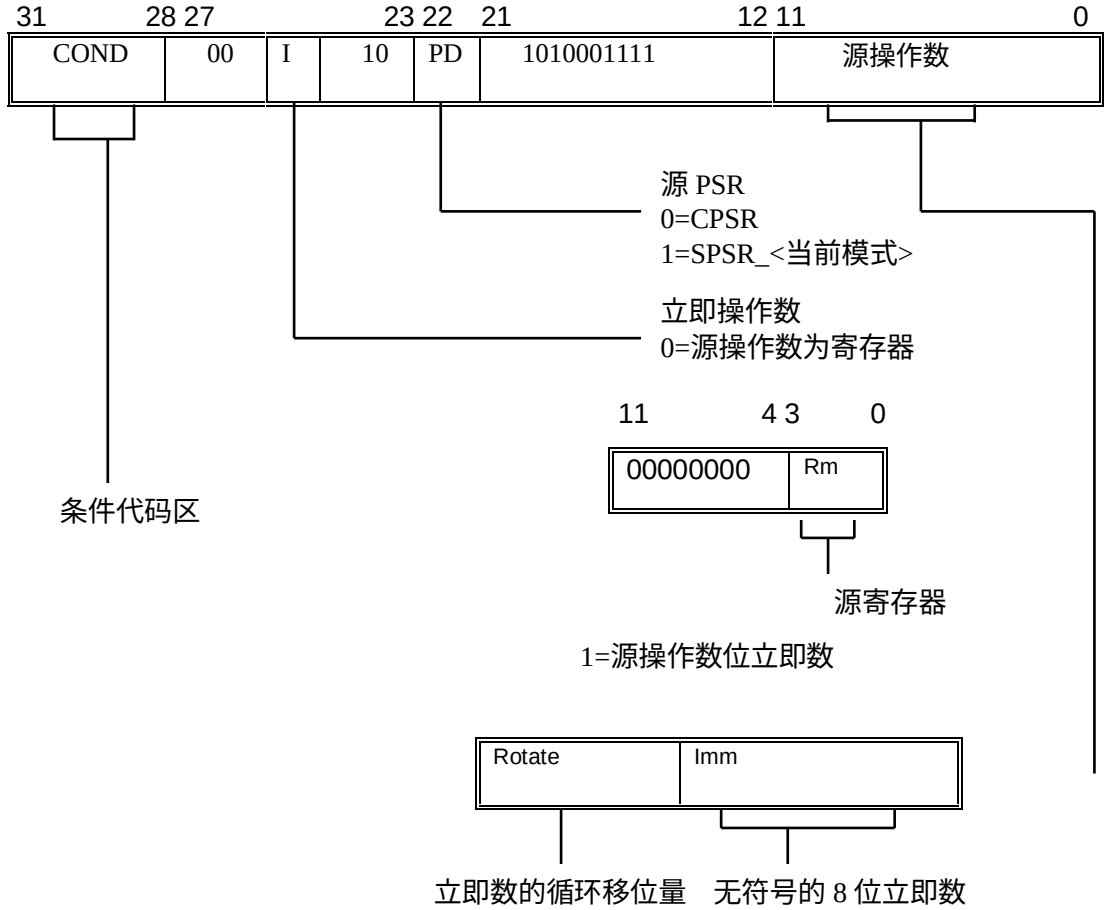


图17：PSR传输

4.5.2 保留位

在ARM7的PSR寄存器中，只定义了11位（N,Z,C,V,I,F和M[4:0]）；其余的位(=PSR[27:8,5])保留，以备以后版本的ARM处理器应用，为了保证处理器最大限度地兼容，必须遵守以下规则：

（1） 在改变PSR的值时，保留位保持不变。

（2） 程序不应该依靠保留位的值来查检查PSR的状态，因为这些保留位在以后版本的处理其中，可能被读为'1'，也可能被读为'0'。

当改变任何PSR的控制位时，应该用到读-改-写指令，如下：用MRS指令传送相应的PSR寄存器到通用寄存器中，只改变相关联的位，然后用MSR指令传送已经改变的值到PSR寄存器中。

例 下面一系列语句执行模式改变：

```
MRS     R0,CPSR              ；复制CPSR到R0
BIC     R0,R0,#0X1F          ；清除模式位
ORR     R0,R0,#new_mode      ；选择新模式
MSR     CPSR,R0              ；回写
```

如果只是简单地改变PSR中条件代码标志位的值，数据可以直接写到标志位，不会改变控制位。下面的指令设置N,Z,C和V 标志位。

```
MSR   CPSR_flg,#0XF0000000     ；设定所有的标志位而不管他们之前出于什么状态
                               ；（不会影响控制位）
```

不要尝试写一个8位的立即数到PSR，因为这样操作会改变保留位。

4.5.3 指令周期

PSR传输花费1S incremental周期，S在5.1指令类型中定义。（65页）

4.5.4 编译器语法

（1） MRS-传送PSR的值到寄存器
MRS{cond} Rd,<psr>

（2） MSR-传送寄存器的值到PSR
MSR{cond} <psr>,Rm

（3） MSR-只传送寄存器的值到PSR的标志位
MSR{cond} <psrf>,Rm
寄存器最高4位分别被写入N,Z,C,和V标志位。

（4） MSR-只传送立即数到PSR的标志位
MSR{cond} <psrf>,<#expression>
表达式应该计算出一个32位值，其中的最高四位分别被写入N,Z,C,和V标志位。

{cond}-条件代码
Rd and Rm 为寄存器的序列号，0-15
<psr>为 CPSR, CPSR_all, SPSR 或SPSR_all. (CPSR 和 CPSR_all 为同义词，SPSR and SPSR_all 一样)
<psrf> 为 CPSR_flg or SPSR_flg

如果用到<#expression>，编译器将产生一个移位过的8位立即数去和表达匹配。如果不可能则报错。

4.5.5 例子

在用户模式下，指令按以下执行：

```
MSR     CPSR_all, Rm                ; CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg, Rm                ; CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg, #0xA0000000       ; CPSR[31:28] <- 0xA (i.e. set N,C; clear Z,V)
MRS     Rd,CPSR                      ; Rd[31:0] <- CPSR[31:0]
```

在管理模式，执行以下过程：

```
MSR       CPSR_all, Rm                      ; CPSR[31:0] <- Rm[31:0]
MSR       CPSR_flg, Rm                      ; CPSR[31:28] <- Rm[31:28]
MSR       CPSR_flg, #0x50000000             ; CPSR[31:28] <- 0x5; (i.e. set Z,V; clear N,C)
MRS       Rd,CPSR                           ; Rd[31:0] <- CPSR[31:0]
MSR       SPSR_all, Rm                      ; SPSR_<mode>[31:0] <- Rm[31:0]
MSR       SPSR_flg, Rm                      ; SPSR_<mode>[31:28] <- Rm[31:28]
MSR       SPSR_flg, #0xC0000000             ; SPSR_<mode>[31:28] <- 0xC;
                                              ;(i.e. set N,Z;clear C,V)
MRS       Rd,SPSR                           ; Rd[31:0] <- SPSR_<mode>[31:0]
```

4.6 乘法和乘加指令（MUL,MLA）

只有当条件为真的时候，此指令才执行。不同条件的定义在本章开始已经讲过，指令编码如图18所示。

乘法和乘加指令用两位的Booth’s算法执行整数乘法。它们给出两个32位操作数计算结果的低32位，可以用于综合的高精度的乘法。

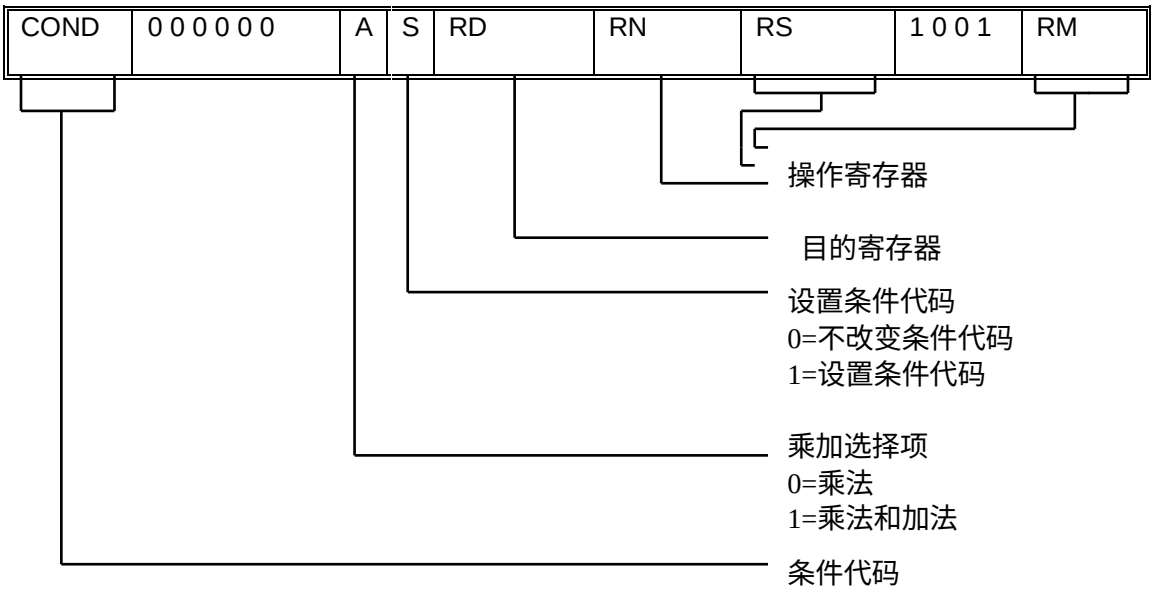


图18：乘法指令

乘法指令使Rd:=Rm*Rs。Rn被忽略，为了保证指令集的向上兼容性，应该设置成0。

乘加指令Rd:=Rm*Rs+Rn，在一些情况下可以节省外在的加法指令。

无符号和有符号的32位操作数乘法指令的结果的不同之处仅仅在高32位，低32位结果是一样的，因为这些指令只计算乘法的低32位，所以有符号和无符号乘法指令可以通用。

比如：乘法操作数：

操作数A	操作数B	结果
0XFFFFFFF6	0X00000014	0XFFFFFF38

如果操作数被认为是有符号，则操作数A为-10，则操作数B为-20，结果位-200，也就是0XFFFFFF38。

如果操作数被认为无符号数，则操作数A为4294967286，则操作数B为20，结果为85899345720，16进制位0X13FFFFFF38,低32位为0XFFFFFF38。

4.6.1 操作限制

由于乘法被应用在其它的ARM处理器中，某种组合的操作数应尽量避免应用。ARM7的先进的乘法器可以处理各种操作数组合，但是留意这些限制可以更好的与其它ARM芯片兼容。（如果忽略这些限制编译器将警告）

目标寄存器Rd不能和操作数Rm相同，R15不能被当作目标寄存器或操作数。

所有其它寄存器组合都可以得出正确的结果，在需要时，Rd,Rn,Rs可以当作同一个寄存器。

4.6.2 CPSR标志

CPSR标志位是的设置可以选择的，CPSR标志位由指令中的S位控制。N (负数) 和 Z (零)标志依据运算结果设置（如果结果为零，Z标志置位，N和结果中的第31位相等），C(Carry)在此指令中无意义，V(Overflow)不受影响。

4.6.3 指令周期时序

乘法指令需要花费1S+ *m* I个 incremental 周期，S和I在5.1部分（65页）周期类型中定义。

m 乘法算法需要的周期数，由Rs的内容决定。乘数在*2^(2m-3)* 和 *2^(2m-1)-1*之间的乘法指令将花费1S+ *m* I 周期（*1<m>16*），由0和1做乘数的乘法指令花费1S+1I 个周期，乘数大于或等于2^(29) 花费1S+16I个周期，乘法指令花费的最大时间为1S+16I个周期。

4.6.4 汇编语言格式

MUL{cond}{S} Rd,Rm,Rs

MLA{cond}{S} Rd,Rm,Rs,Rn

{cond} 条件代码
{S} 如果S出现，设置条件代码
Rd, Rm, Rs 和 Rn 为计算寄存器序号的表达式，不包含R15。

4.6.5 举例

```
MUL R1,R2,R3 ; R1:=R2*R3
MLAEQS R1,R2,R3,R4 ; 有条件地 R1:=R2*R3+R4,设置条件代码
```

4.7 单字节数据传输（LDR,STR）

此指令只有在条件代码为真的时候执行。不同的条件在本章开始时已定义。图19给出指令编码格式。

单字节指令经常用于加载或者存储单个字节或字，传输指令中存储器的地址由基址寄存器的值加上或者减去偏移量，如果需要`auto-indexing'，地址计算结果将被回写到基址寄存器。

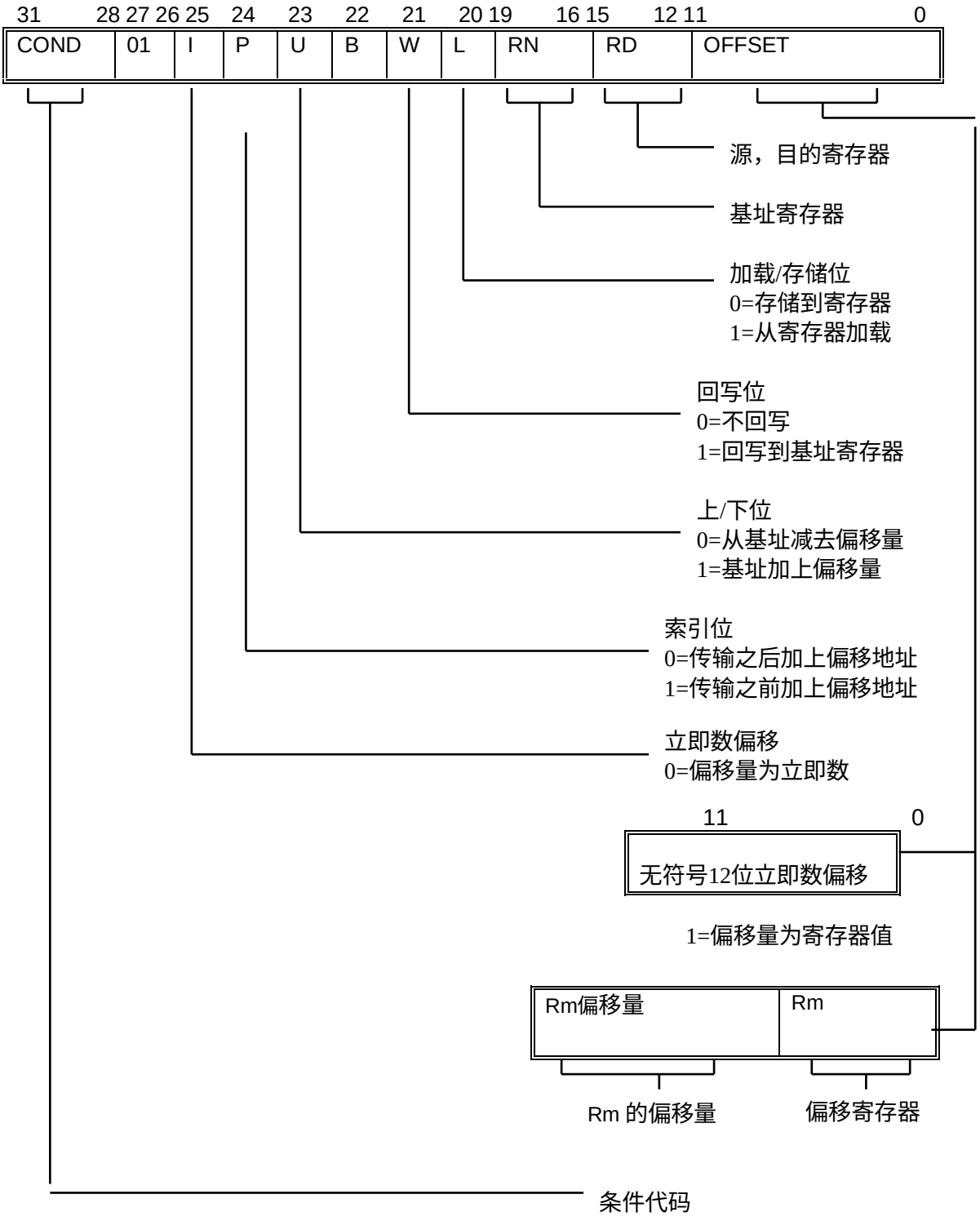


图19　单周期数据传输指令

4.7.1 偏移量和自动索引

基址偏移量可以是指令中的12位无符号二进制立即数，也可以是指令中的第二个寄存器（以某中方式移位）。基址寄存器Rn可以加上(U=1)或减去(U=0)偏移量，偏移量的加减可以在基址作为传输地址之前(前向索引 P=1)或者之后（后向索引, P=0）进行。

W位设置可以选择的自动的加或者减的寻址模式，改变的基址值可以被回写到基址寄存器（W=1）或者原有的基址值保持不变（W=0）。在后向索引（post-indexed）的寻址模式下，回写位是多余的，所以常设为0，后向索引的数据传输总是会回写改变过的基址值，原有的基址值通过设置偏移量为0而得以保持。后向索引的数据传输只有在管理模式下才会用到W位，设置W位强制非管理级模式传输。在用到MMU（存储管理单元）的系统中，允许操作系统产生一个用户地址。

4.7.2 移位寄存器偏移量

在数据处理指令中描述了8位的移位控制位，然而，在这种类型的指令中，寄存器指定的偏移量是无效的，参考4.4.2移位。

4.7.3 字节和字

这种指令类型用于在ARM7寄存器和MEM之间传送字（B=0）或者字节(B=1)。

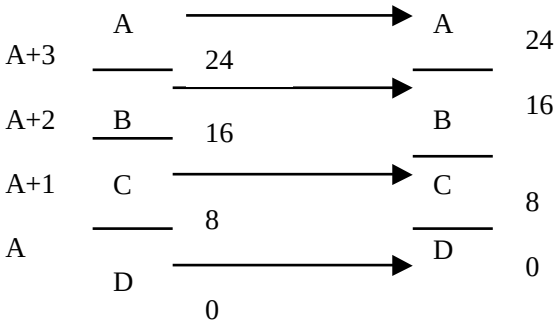
LDR(B)和STR(B)的指令行为受到 Bigend 控制信号的影响，下面描述了两个可能的配置。

Little Endian配置

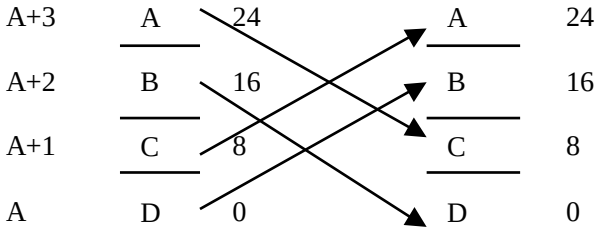
如果提供的地址是字边界，字节加载（LDRB）希望数据线上的数据输入为7到0，如果是字地址加上一个字节地址，数据线上的数据输入应该是15到8。被选中的字节放在目标寄存器的低8位，寄存器余下的位填充位0。请参考图3。

字节存储（STRB）通过四次重复将源寄存器中的低8位放到数据总线上的0到31。外部的MEM系统应该激活相应的字节子系统去存储数据。

字加载（LDR）通常用字对齐地址。然而，字边界的地址偏移将导致数据旋转到寄存器，使寻址的字节占用了位0 到7。这意味着在字边界地址偏移0到2的半字访问可以被正确地加载到寄存器0到15。需要两个移位操作去清除或者标志扩展的高16位。在图25说明 *Little Endian* 偏移地址。



LDR 字对齐地址



LDR 地址偏移2

图20：**Little Endian** 偏移寻址

字存储指令（STR）应该产生一个字对齐地址，如果地址不是字对齐，数据线上的字不会受影响，也就是说，寄存器的第31位通常出现在数据输出的第31位。

Big Endian 配置

如果提供的地址为字边界，字节加载（LDRB）希望在总线输入上的数据为31到24，如果地址为字地址加一，总线输入上的数据为23到16，依此类推。选中的字节被放在目标寄存器的低8位，寄存器的其它位用零填充。参考图4。

字节存储(STRB)通过四次重复将源寄存器中的低8位放到数据总线上的0到31。外部的MEM系统应该激活相应的子系统去存储数据。

字加载（LDR）应产生字对齐地址，字边界地址偏移0 或2将导致数据被旋转到寄存器，所以寻址的字节占用了寄存器31到24位。这意味着在这些偏移的基础上的半字访问将被正确地加载到寄存器的16到31位。需要移位操作去移动（可选的符号扩展）数据到低16位。字边界地址偏移1或3将导致数据被旋转到寄存器15到8 位。

字存储（STR）将产生字对齐地址。如果不是字对齐地址，总线上的字将不受影响。也就是说，存储在寄存器的位31总是总线上的位31。

4.7.4　R15的应用

如果R15被指定为基址寄存器(Rn)，不会指定回写。当R15被用作基址寄存器，必须记住，它包含了的地址为当前指令地址加上的8字节地址。

R15不能作为偏移寄存器（Rm）。

当R15在寄存器存储指令（STR）中作为源寄存器(Rd)，存储的值将是指令地址加12。

4.7.5 基址寄存器的应用限制

当配置过时异常时，下面的代码因为基址寄存器很难解开，在异常处理之前，Rn得到更新，计算初始的值有时是不可能的。

例：

```
LDR R0,[R1],R1
```

<LDR|STR> Rd, [Rn],{+/-}Rn{,<shift>}

4.7.6 数据异常

从合法地址传送或者传送到合法地址对存储器管理系统来说都可能产生问题。例如，在一个用虚拟存储器的系统中，所请求的数据对主存储器来讲可能不存在。存储管理器驱动处理器的ABORT输入为高标志此问题，因此，数据异常陷阱产生。直到系统软件解决这个问题，指令被重新执行源程序继续。

4.4.7 指令周期时序

正常的LDR指令花费1S+1N+1I,LDR PC 花费2S+2N+1I 周期，S,N,I 将在5.1(65)页周期类型中定义。

STR指令花费2N周期。

4.7.8 汇编语法

<LDR|STR>{cond}{B}{T} Rd,<Address>

LDR –从存储器到寄存器

STR – 从寄存器到存储器

{cond} 图8 ：条件代码
{B} 如果有B，则为字节传输，否则为字传输
{T} 如果T存在，在后索引指令中W将被置位，强迫传输周期进入非管理模式。当前索引地址被指定或应用时，T是不允许出现的。

Rd是计算寄存器号的表达式。

<Address>可以是：

（1） 产生地址的表达式
<表达式>
编译器试图产生一个指令，用PC做基址，以表达式计算结果作为地址，此地址中的数据作为立即数偏移量，是一个PC相关，前索引的地址。如果次地址超出边界，报错。

（2） 前索引地址规格

[Rn] 零偏移
[Rn，<#表达式>]{！} 偏移为表达式个字节
[Rn,{+/-}Rm{,<shift>}]{!} 偏移为+/-索引寄存器中的值，移位通过<shift>

（3） 后向索引地址规格

[Rn]<#expression> 偏移为表达式个字节
[Rn],{+/-}Rm{,<shift>}偏移为+/-索引寄存器中的值，移位通过<shift>。

Rn和Rm是计算寄存器序号的表达式，如果Rn为R15，考虑到ARM7的流水线操作，编译器将从偏移量减去8,在这种情况下，不指定回写。

<shift>是通用的移位操作，（请看数据处理指令部分），但要注意，位移量不是通过寄存器指定。

{!} 如果！存在，回写基址寄存器。

4.7.9 举例

```
STR R1,[R2,R4]! ; 存储R1在 R2+R4 (两个都位寄存器)回写地址到R2

STR R1,[R2],R4 ;  存储R1到R2 所包含的有效地址， R2+R4 合成的有效地址写入 R2

LDR R1,[R2,#16] ;以R2+16为有效地址，此地址中的内容加载到R1，不回写

LDR R1,[R2,R3,LSL#2] ; 以R2+R3*4为地址，此地址中的内容加载到R1

LDREQB R1,[R6,#5] ; 以R6+5为地址，有条件地加载此地址中的内容到R1的0到7位，其它
                      位用0填充

STR R1,PLACE ;       产生一个PC相关的偏移地址
```

4.8 数据块传输（LDM,STM）

此指令只有当条件为真的时候才会执行。不同的条件在本章开始时已定义。图12显示了指令编码。

数据块传输指令用于加载（LDM）或者存储（STM）当前有效 寄存器的任意子集。它们支持所有可能的堆栈模式，维持空或者满的堆栈，此堆栈可以向上或者向下，在保存或者恢复内容，移动主存储器的大数据块是非常有效的。

4.8.1 寄存器列表

此指令可以导致当前bank中的任意寄存器传输（非用户模式程序可以传送或者接收用户bank）。在指令中，寄存器列表占用了16位，每一位对应了一个寄存器，如果位0为'1'，将导致R0被传输，为'0 'R0将不被传送，以此类推。

寄存器的任意子集或者所有的寄存器都可以被指定，唯一的限制是寄存器列表不应该为空。

任何时候R15被存储到MEM中，存储的值是SMT指令地址加12。

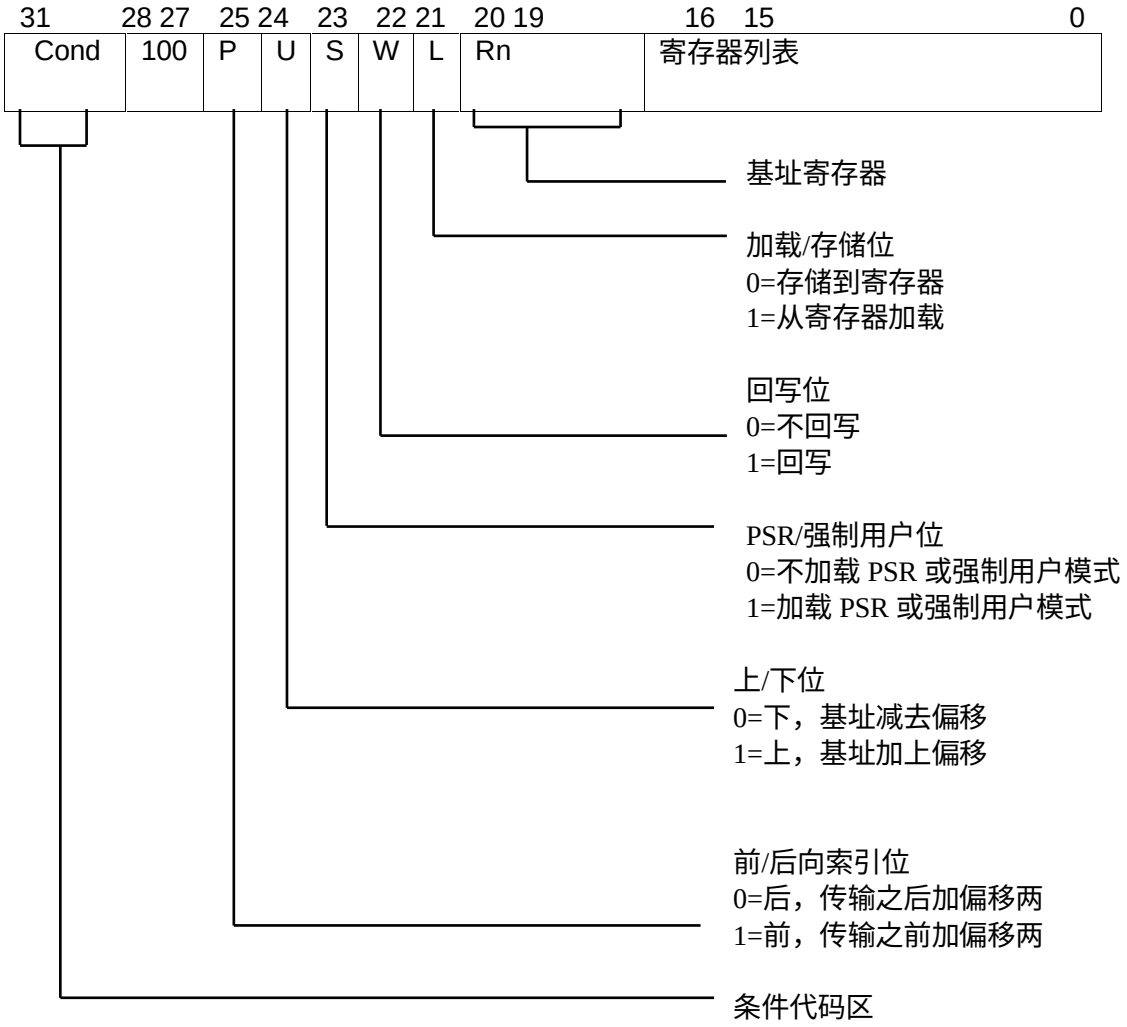


图21：块数据传输指令

4.8.2 地址模式

传输地址是由Rn中的内容和 前/后向索引位，上/下位决定的，寄存器的传输按照从低向高的顺序，如果寄存器列表中有R15,则R15在最后一个被传输。序号低的寄存器对应于存储器的低地址。下面的图例，在Rn=0X1000和需要回写位(W=1)的情况下， R1,R5,R7的传送。

图22：后向增加寻址，图23：前向增加寻址，图24：后向减小寻址，图25：前向减小寻址 。显示出了指令完成后，寄存器传输顺序，用到的地址，Rn值的变化。

在基址回写位没有被请求（W=0）的情况下,Rn将保持它的初始值0X1000，除非Rn也在多寄存器的指令列表中，被加载的值覆盖。

4.8.3 地址对齐

地址通常应该是字对齐，但非字对齐的地址不影响指令。地址的低两位将出现在地址线A[1:0]，由MEM系统解释。

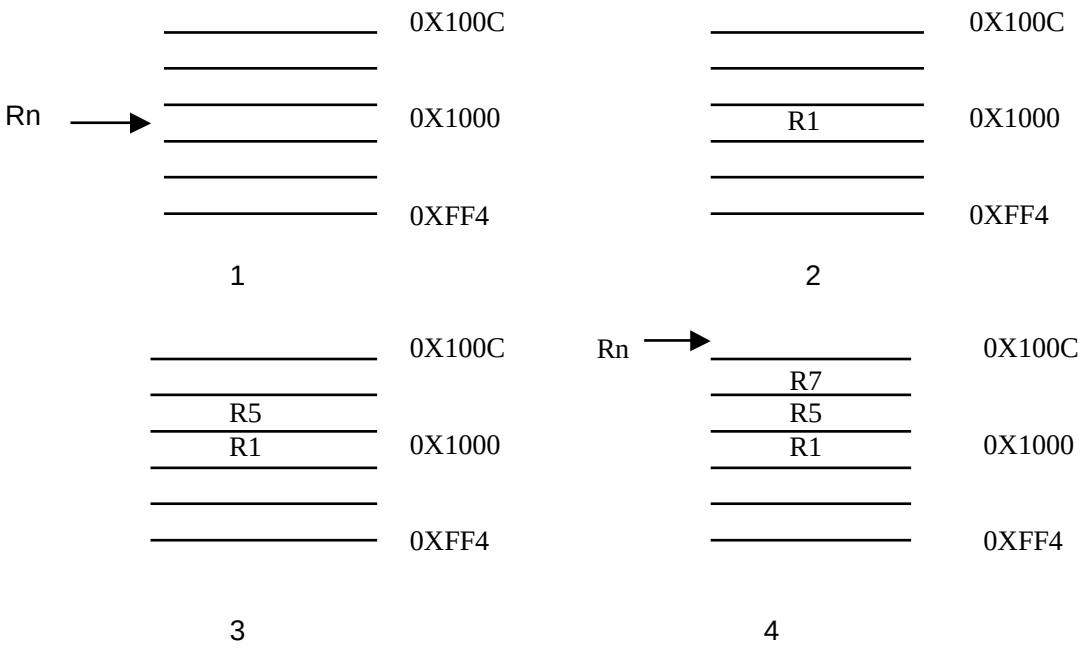


图22：后向增加地址

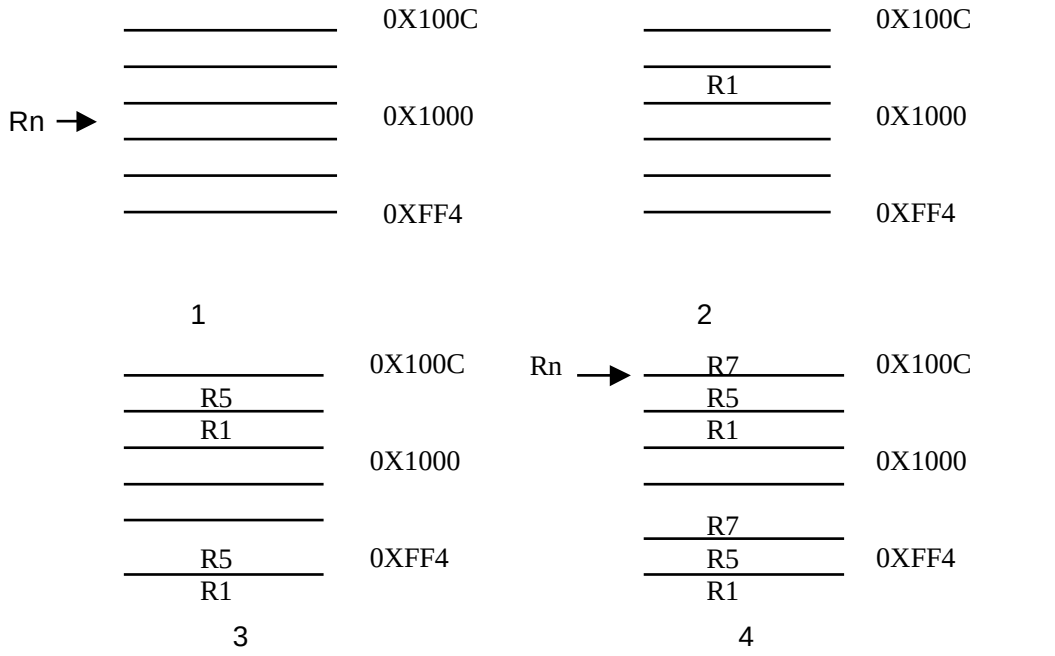


图23：前向加地址

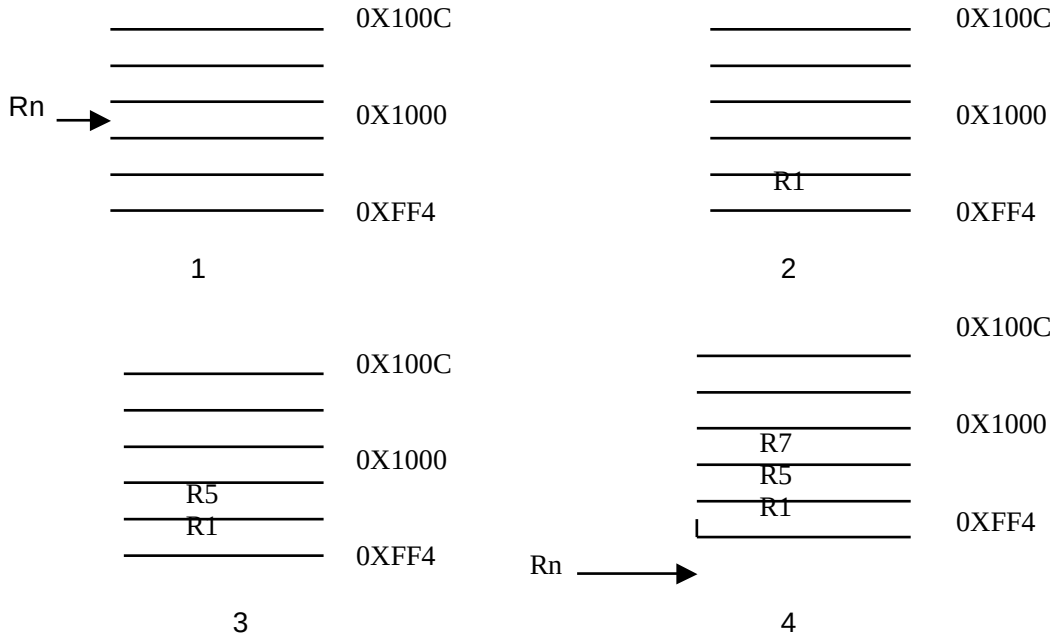


图24：后向减地址

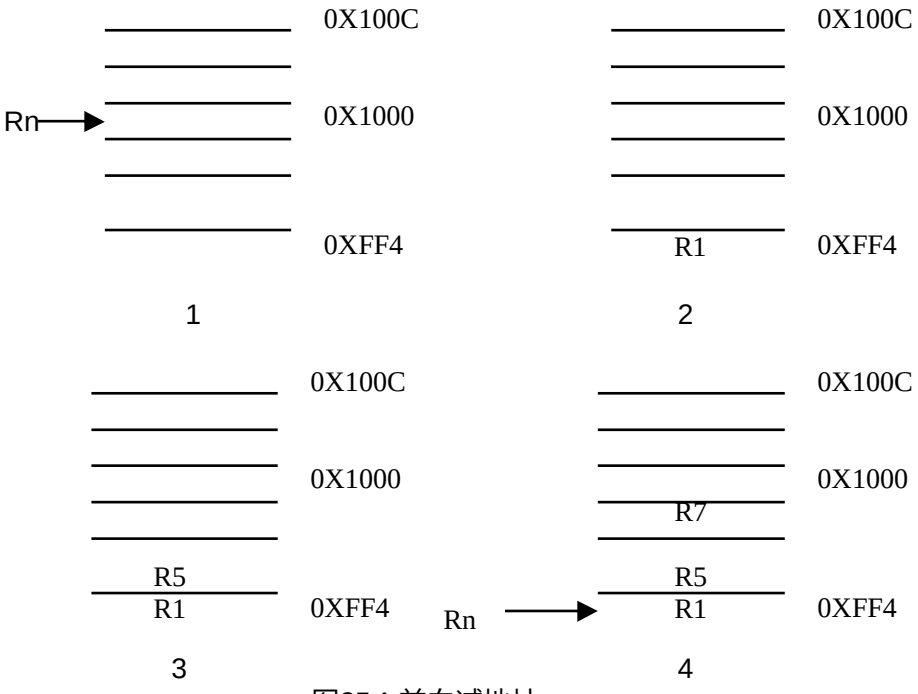


图25：前向减地址

4.8.4 S位的应用

在LDM/STM指令中，若置位S位，它的含义依据R15是否在传输列表中和指令类型的不同而不同。指令只有在管理模式执行时，S才应该置位。

LDM R15在传输列表中，S置位（模式转换）

如果指令是LDM，当R15被加载的同时，SPSR_<mode>被传送到CPSR中。

STM R15在传输列表中，S置位（用户bank传输）

被传输的寄存器来自用户bank，而不是从当前模式相应的bank。这对于在进程切换中节省用户状态是非常有用的。当这个状态启动时，将禁止基址回写。

R15 不在传输列表中，S置位（用户bank传输）

对LDM,STM指令来讲，用户寄存器bank将被传送，而不是当前模式下相应的寄存器bank。这对于在进程切换中节省用户状态是非常有用的。当这个状态启动时，将禁止基址回写。

当指令为LDM，必须注意，在下一周期中不要读bank寄存器（在LDM后加入延迟保证安全）。

4.8.5 用R15作为基址

在LDM 或STM指令中，R15不能作为基址寄存器。

4.8.6 寄存器列表包含基址寄存器

当回写被指定，基址在指令执行的第二个周期结束时回写。在STM指令周期，第一个寄存器在第二个周期的开始被写出。包含存储的基址，并且基址作为第一个寄存器被存储的STM指令将存储不变的基址值，而基址在第二或其它顺序时，则存储的是已经修改过的值。如果基址在列表中，LDM指令将覆盖当前基址。

4.8.7 数据异常

某些合法的地址对存储器管理系统来讲是不可接受的，存储管理器通过将ABORT置高表明一个地址问题。在任何一个加载或者传输多个寄存器的指令中，都可能发生这种情况，如果ARM7用在虚拟存储器系统中，异常必须为可恢复的。

STM指令异常

如果一个异常发生在多个寄存器存储指令执行时，ARM7将保持不动直至此指令完成，然后，它进入一个数据异常陷阱。存储管理器应该阻止错误的写存储器操作。如果回写被指定，处理器内部状态的唯一变化是基址的改变，这个改变在指令重试之前必须由软件恢复。

LDM指令异常

当ARM7在加载多个寄存器指令时检测到数据异常，它改变指令的操作以保证数据恢复。

（1） 当异常产生时，停止正在进行的覆盖寄存器的操作。当前异常加载将不会发生，但是之前有可能产生寄存器覆盖。PC一直是最后一个被写的寄存器，所以将被保留。

（2） 如果回写被请求，对于已经修改过的基址来讲，是可以恢复的。这就保证了在基址寄存器在传输传输列表中或者在异常发生之前被覆盖的可恢复性，。

当加载多个寄存器指令完成时，数据异常陷阱产生，系统软件在重新执行指令之前，必须恢复任何基址改变（并解决异常）。

4.8.8 指令周期

通常LDM指令花费nS + 1N + 1I incremental 周期，LDM PC 花费 (n+1)S + 2N + 1I incremental 周期，S,N,I在65页第五章周期类型中定义。

STM指令花费(n-1)S + 2N incremental cycles。

n 表示需要传输的字数。

4.8.9 汇编语法

<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}

{cond} 条件代码

Rn 计算有效寄存器序号的表达式

<Rlist> 寄存器列表，寄存器范围包含在{}（比如{R0,R2-R7,R10}）

{!} 如果存在，表示请求回写（W=1），否则，W=0

{^}如果存在，置位S，加载CPSR和PC，在管理模式，强制传送用户bank

地址模式命名

对于每一个不同的地址模式有不同的汇编标识方法，依据指令是否用于支持堆栈或者其它用途。不同的指令名称和相应的位的对应关系列于下表：

NAME	STACK	OTHER	L BIT	P BIT	U BIT
pre-increment load	LDMED	LDMIB	1	1	1
post-increment load	LDMFD	LDMIA	1	0	1
pre-decrement load	LDMEA	LDMDB	1	1	0
post-decrement load	LDMFA	LDMDA	1	0	0
pre-increment store	STMFA	STMIB	0	1	1
post-increment store	STMEA	STMIA	00	0	1
pre-decrement store	STMFD	STMDB	0	1	0
post-decrement store	STMED	STMDA	0	0	0

地址模式名称

通过参考堆栈请求格式，FD,ED,FA,EA定义了前/后向索引和上/下位，F,E表示堆栈满或者空。 A 和 D 定义堆栈是递增还是递减，如果递增，STM将向上，LDM向下，如果递减，则相反。
当LDM/STM没有被用于堆栈，而只是简单地表示地址前向增加，后向增加，前向减少，后向减少时，由IA,IB,DA,DB控制。

4.8.10 举例

```
LDMFD SP!,{R0,R1,R2} ; 弹出三个寄存器
STMIA R0,{R0-R15} ; 保存所有的寄存器
LDMFD SP!,{R15} ; R15 <- (SP),CPSR 不改变
LDMFD SP!,{R15}^ ; R15 <- (SP), CPSR <- SPSR_mode (优先级模式)
STMFD R13,{R0-R14}^ ; 在堆栈中保存用户寄存器 (优先级模式)
```

这些指令用于在子程序的入口保存状态，返回调用程序时恢复。

```
STMED SP!,{R0-R3,R14} ;保存R0 到R3，使应用程序可以访问;保存R14用于程序返回
BL somewhere ; 嵌套调用将覆盖R14
LDMED SP!,{R0-R3,R15} ; 恢复空间并返回
```

4.9 单次数据交换

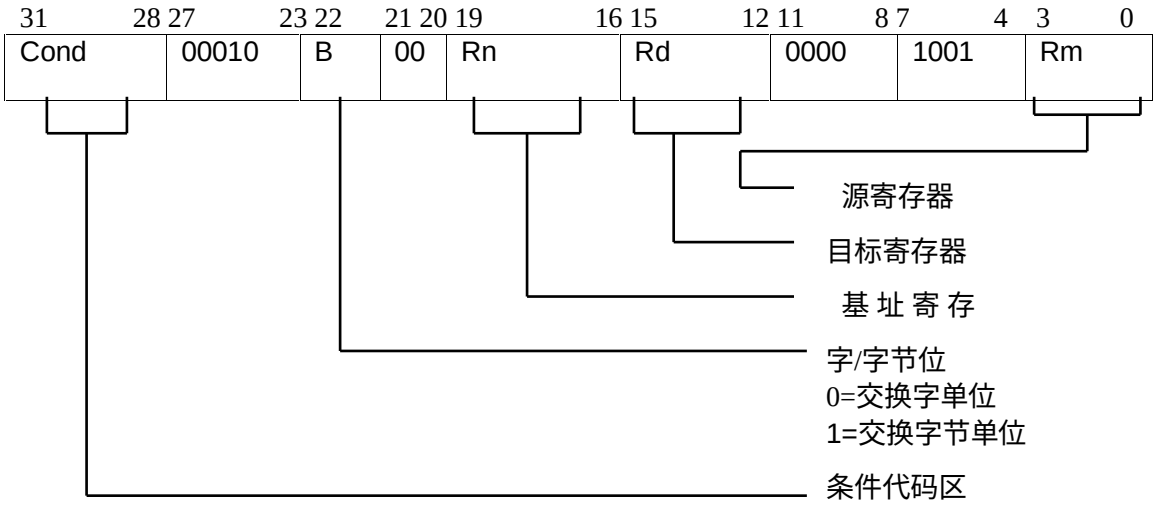


图26：交换指令

此指令只有条件为真的时候才能执行。不同的条件定义在本章开头已经讲过。指令编码显示在图26：交换指令。

数据交换指令用于在寄存器和外部存储器之间交换字或者字节。此指令执行表现为一个存储器读后面紧跟存储器写，两条指令被锁在一起连续执行（处理器在两个操作完成之前，不能被中断，存储管理器被警告不可将这两条指令分开对待），这种类型的指令是非常有用的，可以用做软件semaphores。

交换的地址由基址寄存器(Rn)中的内容来决定，处理器首先读要交换的地址的内容，然后写源寄存器（Rm）中的内容到交换地址，然后存储原存储器内容到目标寄存器。源寄存器和目标寄存器可能为同一个寄存器。

在读和写操作过程中，LOCK信号输出为高，提醒外部的存储管理器读写操作已经被锁在一起，应该没有中断地完成。在多处理器系统中，这是非常重要的，交换指令是惟一的不可分割指令，可以用于semaphores，存储器的控制权一定不能从正在执行锁操作的处理器上移开。

4.9.1 字节和字

这种指令可以用于在ARM7 寄存器和存储器之间交换字（B=0）或者字节(B=1)。SWP 指令执行相当于LDR指令紧跟STR指令，这些指令行为和在单此数据传输部分描述的一样。特别地，Big Endian和 Little Endian 的描述同样适用于SWP指令。

4.9.2 R15的应用

在SWP指令中，R15不能作为操作数使用。

4.9.3 数据异常

如果用于交换的地址对于存储管理系统来讲是不可接受的，内部的MMU或外部的存储管理器将通过驱动ABORT为高来标志这个问题。这可能发生在读，写（读写）周期或其它周期，产生数据异常陷阱。直到系统软件解决这个问题，指令才可以重新执行，源程序继续。

4.9.4 指令周期

交换指令的执行花费1S + 2N +1I 个周期，S,N,I在65页5.1部分指令周期类型中介绍。

4.9.5 汇编语法

<SWP>{cond}{B} Rd,Rm,[Rn]

{Cond} 条件代码

{B}　　如果B存在，则为字节交换，否则，字交换

Rd,Rm,Rn 为计算有效寄存器序号的表达式

4.9.6 例子

```
SWP R0,R1,[R2] ; 加载R0到R2的有效地址,保存R1在R2有效地址

SWPB R2,R3,[R4] ; 加载R2到R4有效字节地址, 存储R3的0到7 到R4有效字节地址

SWPEQ R0,R0,[R1] ,有条件地将R1有效地址中的内容和R0的值交换
```

4.10 软件中断

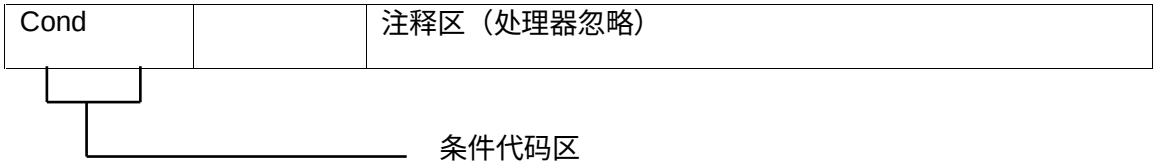


图27：软件中断指令

此指令只有在条件代码为真时执行。不同的条件定义在本章开头已经讲过。图27 显示指令编码。

软件中断指令常以一种可控制的方式进入管理模式。此指令导致软件中断陷阱产生，并且影响到模式切换。PC被强制为固定值0X08,CPSR被存到SPSR_svc。如果保护SWI向量地址，禁止用户改变（通过外部的存储管理硬件），可以建立一个全保护的操作系统。

4.10.1 从管理模式下返回

一旦进入软件中断陷阱，PC被保存到R14_svc，在SWI指令后，调节PC到SWI指令的下一条指令的地址。MOVS PC,R14_svc 将返回到调用程序并恢复CPSR。

要注意联接机制不能重复进入，所以如果管理代码自己想用软件中断，它必须首先保存返回的地址和SPSR。

4.10.2 注释区

指令的低24位被处理器忽略，可以用来同管理代码交换信息。比如，管理员可以检查这个区域并且用它作为执行不同管理功能程序的队列索引。

4.10.3 指令周期

软件中断指令执行将花费2S+N incremental cycles，S,N将在65页5.1部分说明。

4.10.4 汇编语法

SWI{cond} <expression>

{cond}　　条件代码

<expression> 放在注释区域，可以计算（被ARM7忽略）

4.10.5 举例

```
SWI ReadC ; 从读数据流得到下一个字符
SWI WriteI+"k" ; 输出一个  K"到写数据流
SWINE 0 ; 条件调用用户模式，注释区域为0
```

上面的例子加顶相应的超级用户代码存在，例

```
0x08 B Supervisor ; SWI 入口
EntryTable ; 超级用户程序入口
DCD ZeroRtn
DCD ReadCRtn
DCD WriteIRtn
...
Zero EQU 0
ReadC EQU 256
WriteI EQU 512
```

超级用户

```
SWI has routine required in bits 8-23 and data (if any) in bits 0-7.
```

假设R13_svc指向一个相应的堆栈

```
STMFD R13,{R0-R2,R14} ; 保存工作寄存器和返回地址
LDR R0,[R14,#-4] ; 得到SWI指令
BIC R0,R0,#0xFF000000 ;清除高8位
MOV R1,R0,LSR#8 ; 得到程序偏移量
ADR R2,EntryTable ; 得到入口列表的起始地址
LDR R15,[R2,R1,LSL#2] ; 分支到适当的程序
WriteIRtn ; enter with character in R0 bits 0-7
......
LDMFD R13,{R0-R2,R15}^ ; 恢复工作环境并返回
                          恢复处理器模式和标志位
```

4.11 协处理器数据操作(CDP)

此指令只有在条件代码为真的时候才会执行。本章开头讲过不同条件代码的定义。指令编码在图28中显示，图28：协处理器数据操作指令

此种类型的指令告诉协处理器执行一些内部的操作，没有结果返回到ARM7，ARM7也不会等待指令的完成。协处理器可以包含一个等待指令执行的队列，这些指令的执行可以和ARM7交迭，将花费2S +1N incremental cycles, S 和N 在65页5.1部分说明，允许协处理器和ARM7并行工作，执行相互独立的任务。

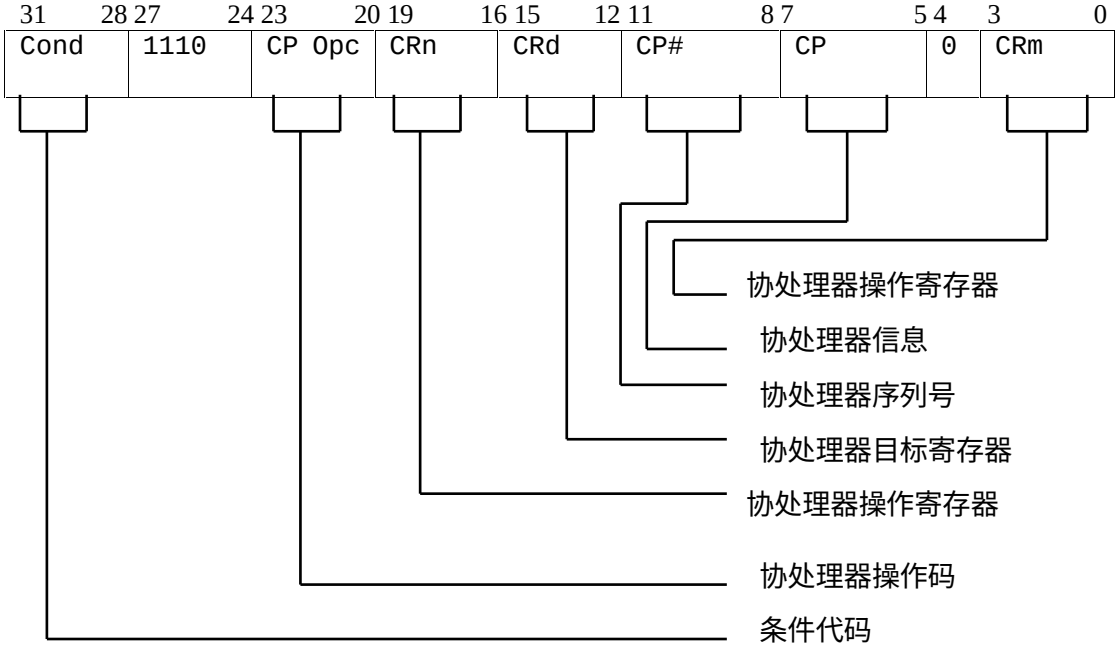


图28:协处理器数据操作指令

4.11.1 协处理器区域

只有位4和位24到31对ARM7来讲有意义；余下的位都由协处理器使用。上面的区域定义都是习惯应用，特殊的协处理器可以重新定义所有域的应用，除过CP#。CP#包含一个协处理器的标识符（从0 到15），一个协处理器可以忽略任何一条CP#域内不包含它的标识符的指令。

此指令的习惯的解释是协处理器应该执行一个在CP Opc域中指定的操作（也有可能在CP域中），操作数为CRn 和 CRm中的内容，并将结果放在CRd中。

4.11.2 指令周期

协处理器数据操作将花费1S + bI incremental cycles ，S和 I在65页5.1中介绍。
b 协处理器busy-wait loop 花费的周期数

4.11.3 汇编语法

CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}

{Cond} 两个字符的条件码，参考图8：条件代码
p# 被请求协处理器的惟一标识
<expression1> 计算一个常量放到CP Opc域
cd, cn and cm 计算有效的协处理器寄存器序号，分别对应于CRd,CRn,CRm
<expression2> 计算出一个常量并放到CP域

4.11.4 举例

```
CDP p1,10,c1,c2,c3 ;   请求写处理器1 执行操作10，操作数为CR2，CR3，并将结果放
                       在CR1
CDPEQ p2,5,c1,c2,c3,2 ; 如果 Z 标志被置位，请求协处理器2 执行操作5（类型2），操
                       作数为 CR2 和 CR3,并将结果放在CR1
```

4.12 协处理器数据传输（LDC,STC）

此指令只有在条件代码位真的时候才执行，不同的条件代码已在本章开始讲过。指令编码在图29显示，图29：协处理器数据传输指令。

这类指令用于直接加载（LDC）或存储(STC)一个协处理器寄存器的子集到MEM。ARM7提供MEM的地址，协处理器提供或者接收数据，并控制传输的字数。

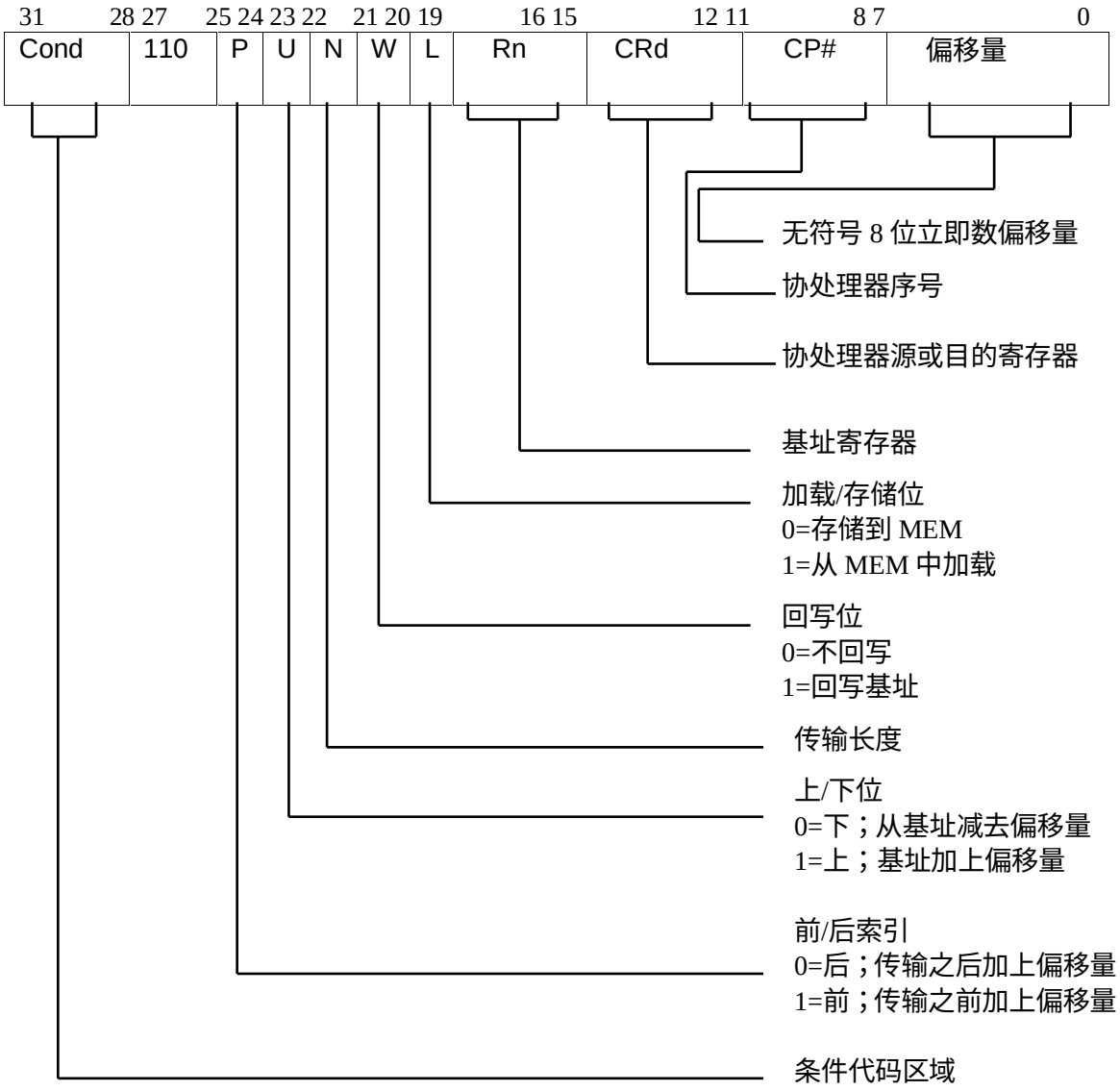


图29：写处理器数据传输指令

4.12.1 协处理器域

CP#域用来标识被请求接收或者发送数据的协处理器，只有当一个协处理器的序号与CP#域中的内容相符，协处理器才会执行指令。

CRd域和N包含了协处理器的信息，这些信息被不同的协处理器用不同的方式解释，但是，通常CRd为被传送的寄存器（当多个寄存器被传送时，CRd为第一个寄存器）。N位用于选择两个传输长度之一。例如，N=0选择单寄存器传输，N=1选择内容切换时所有寄存器的传输。

.

4.12.2 地址模式

ARM7提供存储器系统传输的地址，用到的寻址模式是用于单次数据传输地址模式的子集。注意，对于协处理器数据传输来讲，8位偏移量指定字偏移，但是对于单次数据传输，立即数偏移量为12位，且指定的是字节偏移。

8位无符号立即数偏移量被左移两位，基址加（U=1）或者减(U=0)偏移量，这个计算可以在基址作为传输地址之前(P=1)或者之后(P=0) 进行。改变的基址值可以回写到基址寄存器（W=1），或者老的基址值被保存（W=0）。注意：后向索引地址模式需要清楚地设置W位，不像LDR,STR，当后向索引时总是要回写。

在前向索引指令中被改变的基址寄存器的值，将作为第一个传输字的地址。第二个字（如果多个字传输的话）的地址将是第一个字地址加一个字（四个字节），并且地址将会按此方式依次增加。

4.12.3 地址对齐

基地址通常情况下应该为字对齐。最低两位地址将为A[1:0]，可以由存储器系统解释。

4.12.4 R15的应用

如果Rn为R15，基址应为指令地址加8个字节，不指定基址回写到R15。

4.12.5 数据异常

如果地址是合法的，但是存储器管理系统产生一个异常中断，将产生数据陷阱。被改变的基址回写，但是所有其它处理器的状态将保持不变。协处理器要保证在异常处理完之后，数据传输可以重新启动，必须保证当指令重试时，任何一个由它响应的动作都可以重复。

4.12.6 指令周期

所有的LDC指令将由软件来计算，所花费的周期数将依靠协处理器的支持软件。
协处理器的数据传输指令的执行将花费(n-1)S + 2N + bI incremental cycles，S,N,I在65页5.1部分周期类型说明。
n　传输字的数量
b 协处理器busy-wait loop 所花费的周期数

4.12.7 汇编语法

<LDC|STC>{cond}{L} p#,cd,<Address>

LDC 从存储器加载到协处理器
STC 从协处理器保存到存储器
{L} 如果有L则表示一个长传输（N=1），否则，短传输(N=0)
{Cond} 两个字符的条件代码,参考图8：条件代码
p# 被请求的协处理器唯一的标识
cd 是计算协处理器寄存器序号的表达式，此序号被放在CRd 区域

< Address〉可以是

（1） 可以产生地址的表达式
<expression> 编译器尝试产生一个用PC做基址的指令，和一个正确的偏移量（表达式计算出的地址）。这将是一个PC相关的，前向索引地址，如果地址超出范围，报错。

（2） 指定一个前向地址
[Rn] 零偏移
[Rn,<#expression>]{!} 偏移表达式个字节

(3) 指定一个后向地址
[Rn],<#expression>偏移表达式个字节

Rn 是有效的ARM7寄存器序号的表达式。注意：如果Rn为R15，考虑到ARM7的流水线操作，编译器将从偏移量减去8。

{!} 如果存在，（W=1）回写基址。

4.12.8 举例

```
LDC p1,c2,table          ; 从地址table中加载协处理器1的操作数c2
                         ; 用PC相关地址。

STCEQL p2,c3,[R5,#24]! ; 有条件地存储协处理器2操作数c3到存储器，地址为R5加24
                         个字节长度的地址，回写此地址到R5,用长传输选项（可能存
                         储多字）。
```

注意：地址偏移用字节表示，指令偏移区域用字表示，编译器会适当地调节偏移。

4.13 协处理器寄存器传输（MRC,MCR）

此指令只有在条件为真时执行，不同的条件在本章开头以定义，指令编码在图30 显示。图30 ：协处理器寄存器传输指令。

此种类型的指令用于直接在ARM7和协处理器之间传输信息。一个协处理器到ARM7寄存器传输指令（MCR）是一个 FIX 保存在协处理器中的浮点值的过程，在协处理器中浮点数被转换成32位的整数，结果传送到ARM7的寄存器。从ARM7寄存器中32位的 (FLOAT) 值到协处理器的浮点数(Floating)阐述了ARM7寄存器到协处理器的传输指令（MCR）的应用。

此指令的一个重要应用是直接在协处理器和ARM7 CPSR寄存器之间传输控制信息。例如，在协处理器中比较两个浮点数的值，结果可以传输到CPSR，来控制指令流程。

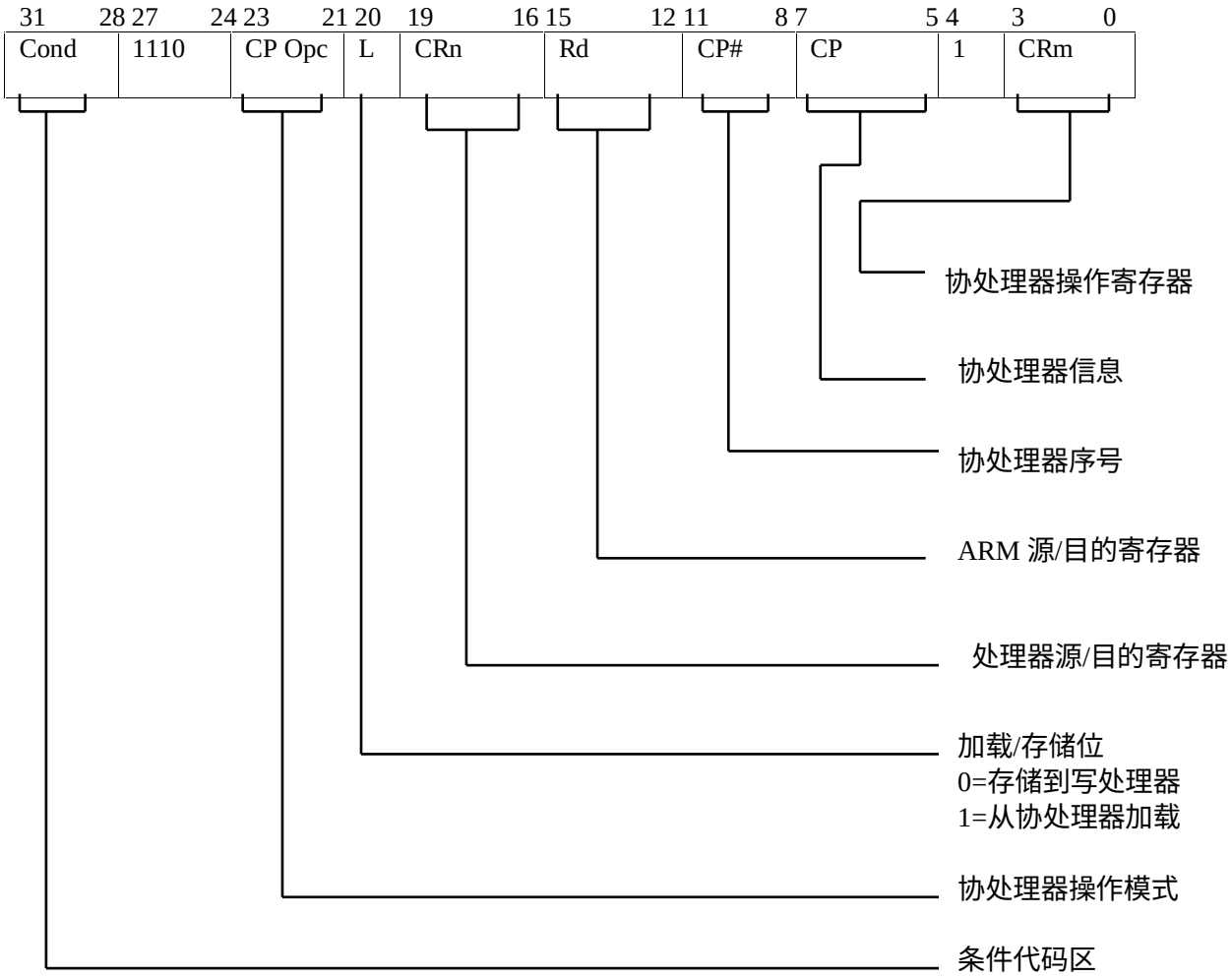


图30：协处理器传输指令

4.13.1 协处理器区域

像所有的协处理器指令一样，CP#指明了哪一个协处理器被访问。

CP Opc, CRn, CP ，CRm 只被协处理器用到，这里的解释来自于惯例应用。当协处理器功能与此不一样时，允许其它解释。通常的解释是CP Opc和CP域指定协处理器被要求的操作，CRn是传送信息的源或者目的协处理器寄存器，CRm是第二个协处理器寄存器，根据指定的特殊操作，被用于不同的场合。

4.13.2 传送到 R15

当一个协处理器传输数据到ARM7，R15作为目的寄存器，被传输的字的31，30，29，28 分别被拷贝到N,Z,C,V标志位。传输字的其它位被忽略，PC 和CPSR 不受传输的影响。

4.13.2 从R15传送

从ARM7传送数据到协处理器寄存器，R15作为源寄存器，协处理器寄存器将存储PC+12。

4.13.4 指令周期时间

MRC 指令执行花费1S + (b+1)I +1C incremental 周期，S,I,C在65页5.1部分说明。
MCR 指令执行花费1S + bI +1C周期。
b 是花费在busy-wait loop 的周期数

4.13.5 汇编语法

<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}

MRC 从协处理器移到ARM7寄存器（L=1）

MCR 从ARM7寄存器移到协处理器（L=0）

{cond} 两个字符的条件代码

p# 被请求得协处理器的惟一标识符

<expression1> 计算一个常量并放到CP Opc域

Rd 是一个表达式计算ARM7有效寄存器序号

cn 和 cm 是计算有效协处理器寄存器CRn CRm序号

<expression2> 计算一个常量，并放到CP域

4.13.6 举例

```
MRC 2,5,R3,c5,c6   ;请求协处理器2 执行操作5 ，操作数为c5和c6，
                   ; 传送结果到R3 (单次32位字)

MCR 6,0,R4,c6       ; 请求协处理器6执行操作0，操作数为R4
                     ;结果送到 c6

MRCEQ 3,9,R3,c5,c6,2 ; 有条件地请求协处理器2执行操作9（类型2）
                       ;操作数为 c5 和 c6，传送结果到R3
```

4.14 无定义指令

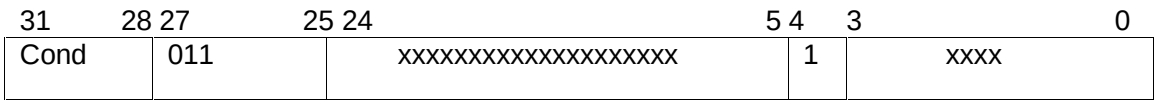


图31：无定义指令

指令只有当条件为真时才执行。不同条件在本章开头以定义，指令编码在图31中显示。
如果条件为真，将产生无定义指令陷阱。

注意：无定义的指令机构提供这条指令给任何一个可能存在的协处理器，所有的协处理器必须通过驱动CPA和CPB为高来拒绝接受这条指令。

4.14.1 汇编语法

目前，编译器没有mnemonics产生这条指令，如果将来某个指定的用途用到它，合适的mnemonics将被加到编译器，在一定的时间内，此指令不会被应用。

5.0 存储器接口

ARM7通过双向的数据总线（D[31:0]）同它的存储器系统通信。单独的32地址总线指定要传输的存储器地址，nRW信号给出传输方向（ARM7到MEM或MEM到ARM7），控制信号给出传输周期需要的其它信息，特别地，它可以使DRAM页模式的应用变得容易。并没有排除基于存储器的静态RAM的接口，通常SDRAM接口比这里描述的DRAM接口要简单。

5.1 周期类型

所有存储器的传输周期都可以被归结到以下四种类型之一：

（1） 不连续周期。ARM请求传输到某个地址或者从某个地址传输，但这个地址跟前一个周期用到的地址没有联系。

（2） 连续周期。ARM请求传输到某个地址或者从某个地址传输，此地址或者同上一个周期的地址相同或者是上一个周期的地址之后一个字。

（3） 内部周期。ARM7不请求一个传输，因为它执行一个内部功能，同时不执行有效的预取。

（4） 协处理器寄存器传输。ARM7希望通过数据总线同协处理器通信，但是不请求任何存储器动作。

对存储器来讲，通过检查nMREQ 和 SEQ 控制线区分这四种周期类型（参考表6：存储器周期类型）。这些控制线在周期的phase1产生，before the cycle whose characteristics they forecast, 控制信息的流水线操作给存储器系统够的时间去决定是否要用页模式访问。

NMREQ	SEQ	周期类型	
0	0	不连续周期	(N-cycle)
0	1	连续周期	(S-cycle)
1	0	内部周期	(I-cycle)
1	1	协处理器寄存器传输	(C-cycle)

表6： 存储器周期类型

图32：ARM存储器周期时序 显示了控制信号的流水线操作，对S周期页模式，建议DRAM地址锁存（**nRAS** 和 **nCAS**)时序。N-cycle 笔其它的周期长，这考虑了DRAM的预冲和行访问时间，并不是 ARM7 的需要。

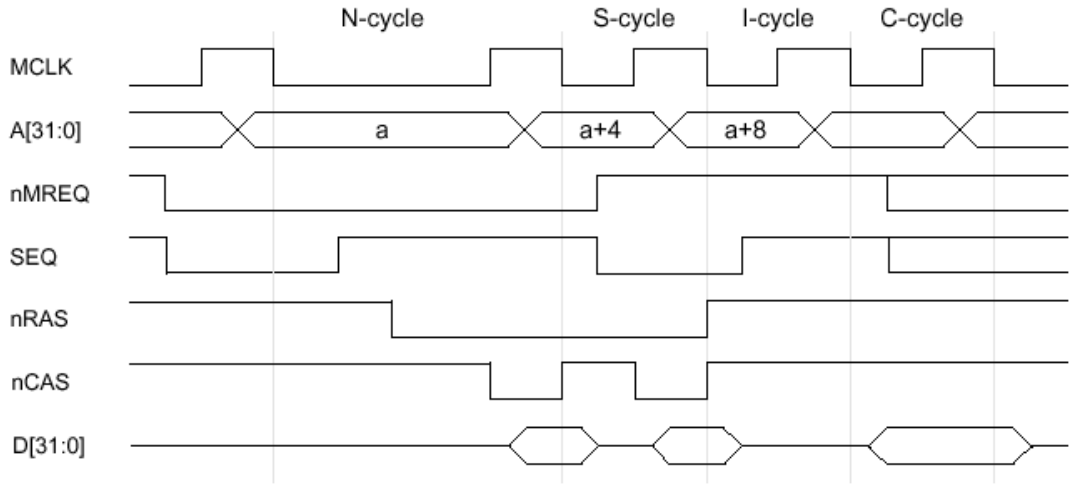


Figure 32: ARM Memory Cycle Timing

当一个S周期跟着一个N周期，地址将是N周期的地址加上一个字。检查该地址（上图中用a标志 ）以保证存储器系统切换到S 周期之前，它不是DRAM页的最后一个地址。如果它是页尾，S周期不能在页模式下执行，存储器系统将不得不进行一个全访问。

处理器的时钟必须是可以延伸的，以适应所有的访问。当一个S周期紧跟着一个I 或 C周期，其地址将和I 或 C周期地址一样。这种情况用于在前一个周期启动DRAM访问，它使能S周期运行在页模式下，同时执行一个全DRAM访问。请参考图33：存储器周期优化。

5.2 字节寻址

处理器地址总线给出了字节地址，但是指令总是字长度（一个字等于四个字节），数据经常以字为单位。但数据传输（LDR,STR）指定以字节为单位。**nBW**控制线用于从存储器系统中请求一个字节，通常情况下，此信号如果为高，表示一个字请求，如果请求字节传输，它在前一个周期的phase2 为变为低。

当处理器从存储器中取出一个指令时，地址线上最低两位A[1:0]时没有定义的。

执行（LDBR）从存储器中读一个字节，存储器系统可以忽略字节请求，而提供整个字。

ARM7将执行内部的字节扩展。或者存储器系统可以激活MEM中任一寻址字节。这对于节约电源，在读，写周期使能通常的解码系统的是非常值得的。

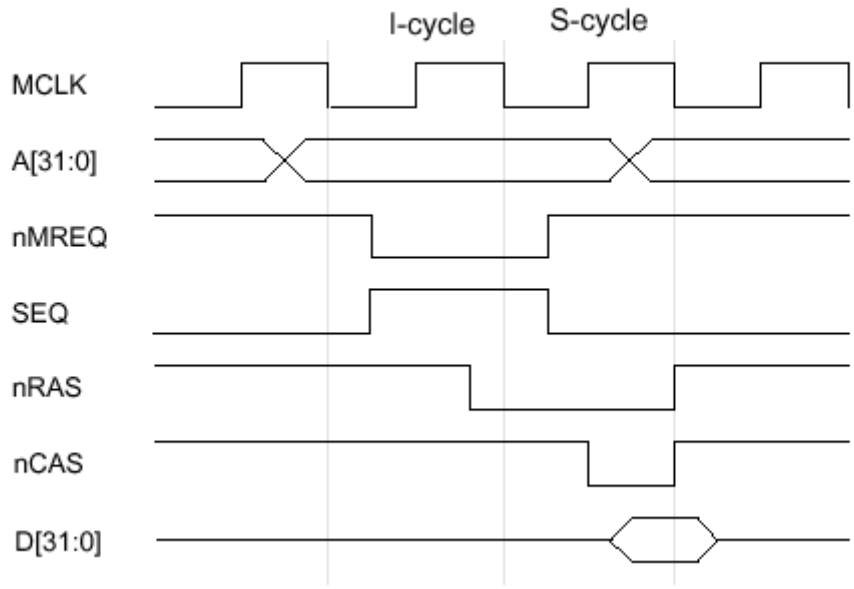


图33：存储周期优化

如果一个字节写被请求（STRB），ARM7将通过数据线广播这个字节值，数据线上的每个字节都被驱动为广播的字节值。存储器系统必须译码A[1:0]去对应唯一的字节地址。

在DRAM系统中应用字节译码的一种方法是将DRAM中32位宽度的BLOCK拆分成四个字节BANK，独立地产生列地址锁存信号。如图34：访问存储器字节译码。

当一个处理器配置成Little Endian，存储器系统的字节0应该对应到数据线D[7:0]，由nCAS0锁存。NCAS1 驱动对应于数据线15到8的bank，依次例推。这个操作具有以下优点：减少每一个锁存驱动的负载，提高关键信号的时间精度。

当一个处理器配置成Big Endian，字节0 对应到数据线D[31:24]。

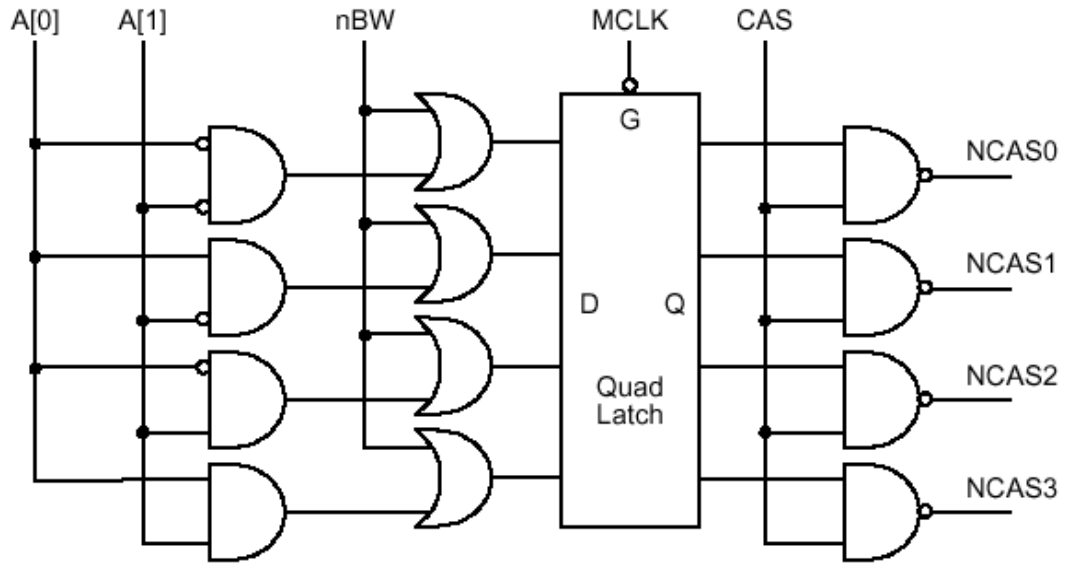


图34：存储器访问的字节译码

5.3 地址时序

通常情况下，处理器地址在phase2改变成下一个周期存储器系统要用到的地址，这给出了驱动地址到大的存储器阵列和需要的地址转换最大时间。动态存储器通常在片上锁存地址，如果锁存时间正确，即使在访问结束之前地址已经改变，存储器也将会正常工作。

静态RAM和ROM在这种情况下将不会正常工作，因为它们要求，在访问的整个过程中地址保持不变。因此，对于我们来讲，如果要用到此类设备，地址转换必须延迟，直到phase2结束。一个由ALE控制的片上地址锁存器，允许地址时序以这种方式改变。在一个用静态和动态存储器的混合系统中（有或者没有地址锁存设备），从一个周期到下一个周期ALE可以动态地改变，依靠存储器系统的判断。

5.4 存储器管理

ARM7 的地址总线在提供给MEM之前可能通过一个地址转换单元处理，ARM7有能力管理一个虚拟存储器。存储管理器通过输入异常到处理器，通知ARM7页错误。其它不同的信号支持不同的页保护级别：

（1） nRW 存储管理器保护页不被写。

（2） nTRANS 表示无论处理器处于用户或者管理模式，用于保护系统页，或者支持对用户和系统来讲完全分离的映射。

（3） nM[4:0] 在处理器模式下，给出存储管理器全部信息。

地址转换通常情况下，只有在N 周期时才是必须的，使得在存储管理模式下可以减少电源消耗，在其它时间避免转换延时。在地址必须需转换的情况下，可以通过跟踪处理器的周期类型减少时间。

如果N 周期要和完全DRAM访问匹配，花费的时间比最小处理器周期要长。Phase1比phase2时间长会给转换系统更多的时间去产生异常中断（必须在Phase1结尾建立）。

5.5 锁操作

ARM7包含数据交换指令，允许存储器中的内容同处理器寄存器中的内容交换。此指令执行像不可中断的一对指令；第一次访问读存储器中的内容，第二次写寄存器中的内容到存储器，这些访问必须被存储控制器当作一个连续的操作，以阻止其它的设备在数据交换完成之前改变存储器相关地址。ARM7在交换操作期间驱动LOCK信号为高警告存储控制器，不要将存储器交给其它设备。

5.6 延长访问时间

所有的存储器周期都由MCLK定义，通过拉长此时钟可以适应较长的访问周期。通常情况下，延长MCLK的低电平，如果访问最终不成功，允许存储管理器中断操作。

延长周期由两种方法，在应用到ARM7之前延长MCLK，或者nWAITE 输入和正常的MCLK与。拉低nWAITE和延长MCLK低电平效果一样，nWAITE只有在MCLK为低电平的时候才可以改变。

ARM7不包含任何根据正常时钟保持内部状态的动态逻辑，所以，MCLK延长或者nWAITE保持低电平的最大周期都没有限制。

6.0 协处理器接口

ARM7指令集的功能可以通过过扩展外部协处理器的方法进一步扩展，外部协处理器最多可以扩展到16个。如果协处理器不存在，预设给它执行的指令产生陷阱，可以安装相应的软件来仿真它的功能。增加协处理器将增加系统在软件兼容方面的性能。注意，一些协处理器序号可能已经被分配。联系ARM Ltd 获得最新消息。

6.1 接口信号

三个专用的信号控制协处理器接口，**nCPI**, **CPA** 和 **CPB** 。**CPA** 和 **CPB** 输入应该驱动为高，除过它们用作握手信号。

6.1.1 协处理Present/Absent

任何时候当ARM7执行一条协处理器（或无定义）指令时，驱动 **nCPI** 为低。（如果因为条件代码不满足，指令不能执行，**nCPI** 不会为低）每一个协处理器会拷贝指令，通过检查CP#，知道此指令是针对哪一个协处理器。系统中每一个协处理器都有一个唯一的序列号（标识），如果CP#域中的内容和协处理器的序列号匹配，协处理器应该驱动CPA为低。如果没有协处理器的序列号同CP#匹配，CPA 和CPB保持高电平，ARM7将产生一个无定义指令陷阱。否则的话，ARM7监视CPA为低，并且一直等到协处理器不忙（CPB为低）。

6.1.2 **Busy-waiting**

如果CPA为低，ARM7将观察CPB（协处理器忙信号），只有驱动CPA为低的协处理器才可以驱动CPB为低。当协处理器准备好执行此指令时，驱动CPB为低。当CPB为高时，ARM7进入Busy-waiting状态，除非一个使能的中断发生，如果中断发生，ARM7将中断和协处理器的握手去处理中断。通常情况下，ARM7从中断返回之后重新执行协处理器指令。

当CPB为低，指令继续执行直到完成，包括协处理器和ARM7或者和存储器之间的数据传输，except in the case of coprocessor data operations which complete immediately the coprocessor ceases to be busy。

ARM7 和协处理器在MCLK的上升沿采样三个接口信号，如果全为低，指令执行。如果需要传输，传输将在下一个周期开始。如果 nCPI 在指令执行之前从低电平之后变为高电平，ARM7将中断Busy-waiting状态进入中断处理程序，指令可能之后会重新执行，若其它协处理器指令很快到来，此指令将被丢弃。

6.1.3 **Pipeline following**

当协处理器指令产生时，为了能够正确地响应，每一个协处理器都会拷贝指令。所有ARM7的指令通过主数据线从存储器终取出，协处理器也连到这条数据总线，所以它们可以在数据进入ARM7时保存拷贝，**nOPC** 信号暗示正在取指，MCLK给出传输时序，这些信号一起应用，在协处理器中加载指令流水线。

6.2 数据传输周期

在数据传输周期，一旦协处理器不忙，它必须以ARM7总线频率（由MCLK定义）提供或者接收数据。它可以检查指令的L位来判断数据传输方向，但是这些操作只有在当DBE为高，允许驱动总线时才可以执行。协处理器要决定传输的字数，ARM7将执行连续传输，每个传输地址为前一个地址加一个字，直到协处理器告诉它传输结束。结束条件为，协处理器驱动CPA和CPB为高。

协处理器依次数据传输可以传输多少个字没有具体的规定，但是通常情况下，在一个指令中没有协处理器允许传输超过16个字。如果超过16个字，将会恶化最坏情况下ARM7的中断响应时间，因为一旦传输开始，此指令时不允许中断。若为16个字，此指令相当于16个寄存器的块传输，因此，不影响响应时间。

6.3 寄存器传输周期

当ARM7在不请求激活存储器的情况下请求数据总线，产生协处理器寄存器传输周期。ARM7通过拉高nMREQ 和 SEQ 通知存储器系统请求总线，当总线空闲时，DBE被拉高允许ARM7或协处理器驱动总线，MCLK时钟定时传输时序。

6.4 特权指令

协处理器可以限制某些指令只用于管理模式。为了做到这些，协处理器必须跟踪 nTRANS 和 nM[4:0] 输出。

应用举例，在多处任务系统中浮点协处理器的应用，在每次任务切换时，操作系统应该保存所有的浮点寄存器，但在一个只有一两个任务用到浮点操作的特殊系统中，这种做法效率是很低的，所以应该有特权指令可以打开或者关上浮点协处理器。当一个任务切换发生时，操作系统可以关掉FPU而不用保存它的寄存器，如果一个新任务尝试FPU操作，FPU将表现为 Absent，导致一个无定义指令陷阱，操作系统就明白有新任务请求FPU，它将重新使能FPU并保存FPU寄存器使任务可以正常地用FPU。如果新任务从来不尝试FPU操作（大多数任务是这样的），将避免状态保存。

6.5 Idempotency

应用协处理器界面和可中断的busy-wait状态的结果是所有的指令在任一点都可能被中断，直到协处理器进入闲状态。如果中断产生，通常在中断处理完成之后指令将从开始重新执行。所以协处理器执行的任何一个动作在它进入闲状态之前必须为Idempotency，它必须是可以重复执行并且有相同的结果。

举例，在浮点协处理器上的FIX操作，它将返回一个整数到ARM7寄存器。当协处理器执行浮点到定点的转换时，它必须保持忙状态，因为ARM7希望浮点协处理器为空闲的下一个周期接收到一个整数值。在整个转换周期协处理器必须保持原始的浮点值不破坏它，因为如果在忙状态有中断产生，浮点数会被重新转换。

协处理器数据操作类指令并不是总是服从 idempotency ，在协处理器进入闲状态后，处理动作可能发生，ARM7不需要hold up，直到结果产生，因为这个结果仅局限在协处理器中。

6.6 无定义指令

ARM7对待无定义指令和协处理器指令一样。当一个无定义类型指令执行时所有的协处理器必须在 Absent 状态（CPA,CPB为高），ARM7将产生一个无定义指令陷阱。注意，协处理器只需要检查指令的第27位来区分是无定义指令（0）还是协处理器指令（1）。

7.0 指令周期操作

在下表中，**nMREQ** 和**SEQ** （为流水线操作，在应用周期的前一个周期出现），所以它们预示了下一个周期的类型，地址，**nB**W, **nR**W, **Nopc**（应用周期的前半个周期）在它们的应用周期显示。

7.1 分支和分支联接

一个分支指令在第一个周期计算分支的目的地址。同时从当前PC执行一个预取。预取在所有情况下都要执行，因为到决定分支是否执行的时候，再阻止预取已经太迟了。

在第二个周期，从计算出来的分支目的地址取指，如果连接位置位，返回个地址被保存在寄存器R14。

第三个周期从目的地址+4 取指，重新加满指令管道，如果为分支连接，R14被修改（-4），简化了SUB PC,R14,#4 和 MOV PC,R14。这使得STM..{R14} LDM..{PC}子程序类型正常工作。操作周期在表7 显示。

Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
1	pc+8	1	0	(pc + 8)	0	0	0
2	alu	1	0	(alu)	0	1	0
3	alu+4	1	0	(alu + 4)	0	1	0
	alu+8						

表7：分支指令周期操作

pc 是分支指令地址
alu 由ARM7计算出的地址
(alu) 是此地址中的内容

7.2 数据操作

数据操作是单周期的数据周期，除过位移量由寄存器中的内容决定的指令。一个寄存器的值被读到A总线，另一个寄存器或者立即数在B总线，ALU根据指令指定的操作，组合A总线上的数据和被移位的B总线上的数据，执行结果（如果需要）写到目的寄存器。比较和测试指令多不会产生结果，只影响ALU状态标志位。

一个指令预取同上面的操作同时产生，程序计数器增加。

当移位量由寄存器指定，在上面的操作之前，产生一个附加周期，拷贝寄存器的低8位并锁存到桶形移位器。指令预取在第一个周期发生，操作周期为内部的（不请求存储器）。存储管理器和并这个内部周期和下一个顺序访问周期，因为这两个周期的地址保持不变。

PC可能是一个或者多个寄存器操作数，当它为目的寄存器时，外部的总线活动将受到影响。如果结果被写到PC，将使指令管道中的内容无效，下一次指令从ALU而不是地址增加器。在任何进一步的动作发生之前，指令管道将被重新加满，在这期间，异常被禁止。

PSR传输操作展示了和数据操作相同的时间特性，只是PC从来没有用作目的或者源寄存器。操作周期在表8中显示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
normal	1	pc+8	1	0	(pc+8)	0	1	0
		pc+12						
dest=pc	1	pc+8	1	0	(pc+8)	0	0	0
	2	alu	1	0	(alu)	0	1	0
	3	alu+4	1	0	(alu+4)	0	1	0
		alu+8						
shift(Rs)	1	pc+8	1	0	(pc+8)	1	0	0
	2	pc+12	1	0	-	0	1	1
		pc+12						
shift(Rs)	1	pc+8	1	0	(pc+8)	1	0	0
dest=pc	2	pc+12	1	0	-	0	0	1
	3	alu	1	0	(alu)	0	1	0
	4	alu+4	1	0	(alu+4)	0	1	0
		alu+8						

表8：数据操作指令周期操作

7.3 乘法和乘加

乘法指令使用特殊的硬件，此硬件执行两位的Booth's算法 with early termination。在第一个周期，加法操作数寄存器被送到ALU，ALU将传送它或者生成零到目标寄存器(依据指令为MLA还是MIL指令)。在同一个周期，乘数（Rs）通过A总线被加载到Booth's移位器中。

数据周期开始循环，加,减被乘数（Rm）或者只是传递它到结果寄存器。在Booth's 算法的控制下，被乘数在第N个周期被移了2N或2N+1位。乘数在每个周期被右移两位，当乘数为零时，指令终止(可能需要一个附加周期去处理的借位)。

除过第一个周期，其它周期都是内部周期，操作周期在下表中显示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
(Rs)=0,1	1	pc+8	1	0	(pc+8)	1	0	0
	2	pc+12	1	0	-	0	1	1
		pc+12			(pc+8)			
(Rs)>1	1	pc+8	1	0	(pc+8)	1	0	0
	2	pc+12	1	0	-	1	0	1
	•	pc+12	1	0	-	1	0	1
	m	pc+12	1	0	-	1	0	1
	m+1	pc+12	1	0	-	0	1	1
		pc+12						

表9：乘法指令周期操作

m是Booth's算法需要的周期数，参考指令速度部分

7.4 加载寄存器

加载寄存器指令的第一个周期执行地址计算，第二个周期数据从存储器中取出，基址寄存器在这个周期被修改（如果需要），第三个周期，数据传送到目的寄存器，外部的存储器没有用到，通常情况下，第三个周期和下一个指令预取周期合并，形成一个存储器N-cycle。下表显示了加载寄存器指令的操作周期。

基址或者目的寄存器（或者两个）可能是PC，如果PC被指令影响，指令预取的顺序将会改变。

如果取数据操作中断，禁止目的寄存器改变。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
normal	1	pc+8	1	0	(pc+8)	0	0	0
	2	alu	b/w	0	(alu)	1	0	1
	3	pc+12	1	0	-	0	1	1
		pc+12						
dest=pc	1	pc+8	1	0	(pc+8)	0	0	0
	2	alu	b/w	0	pc'	1	0	1
	3	pc+12	1	0	-	0	0	1
	4	pc'	1	0	(pc')	0	1	0
	5	pc'+4	1	0	(pc'+4)	0	1	0
		pc'+8						

表10：加载寄存器指令周期操作

7.5 存储寄存器

存储寄存器的第一个周期同加载寄存器的第一个周期相似，在第二个周期，修改基址，同时数据被写入存储器，没有第三个周期。操作周期在下表显示。

Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
1	pc+8	1	0	(pc+8)	0	0	0
2	alu	b/w	1	Rd	0	0	1
	pc+12						

表11：存储寄存器指令周期操作

7.6 加载多个寄存器

LDM的第一个周期计算第一个传输字的地址，同时执行从存储器预取，第二个周期取出第一个字，执行基址修改，第三个周期，第一个传输字被移到相应的目的寄存器，同时第二个字从存储器中取出，修改过的基址在内部锁存，以防中断发生之后，需要恢复基址。第三个周期重复执行，直到最后一个数据字被访问。最后一个周期（内部的）移动最后一个字到它的目的寄存器，操作周期下表显示。

最后一个周期和下一个指令预取周期和并，形成一个单次存储器N-cycle。

如果异常中断产生，指令继续执行直到完成，但是在异常发生之后禁止所有的寄存器写。最后一个周期改变，恢复修改过的基址寄存器（可能被异常产生之前的加载动作改写）。

当PC在要加载的寄存器列表中时，当前的指令管道必须无效。

注意PC总是最后一个被加载的寄存器，所以任何一点产生异常将阻止PC被改写。指令操作周期如下表所示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
1 register	1	pc+8	1	0	(pc+8)	0	0	0
	2	alu	1	0	(alu)	1	0	1
	3	pc+12	1	0	-	0	1	1
		pc+12						
1 register	1	pc+8	1	0	(pc+8)	0	0	0
dest=pc	2	alu	1	0	pc'	1	0	1
	3	pc+12	1	0	-	0	0	1
	4	pc'	1	0	(pc')	0	1	0
	5	pc'+4	1	0	(pc'+4)	0	1	0
		pc'+8						
n registers	1	pc+8	1	0	(pc+8)	0	0	0
(n>1)	2	alu	1	0	(alu)	0	1	1
	*	alu+*	1	0	(alu+*)	0	1	1
	n	alu+*	1	0	(alu+*)	0	1	1
	n+1	alu+*	1	0	(alu+*)	1	0	1
	n+2	pc+12	1	0	-	0	1	1
		pc+12						
n registers	1	pc+8	1	0	(pc+8)	0	0	0
(n>10)	2	alu	1	0	(alu)	0	1	1
incl pc	*	alu+*	1	0	(alu+*)	0	1	1
	n	alu+*	1	0	(alu+*)	0	1	1
	n+1	alu+*	1	0	pc'	1	0	1
	n+2	pc+12	1	0	-	0	0	1
	n+3	pc'	1	0	(pc')	0	1	0
	n+4	pc'+4	1	0	(pc'+4)	0	1	0
		pc'+8						

表12：加载多个寄存器指令操作周期

7.7 存储多个寄存器

存储多个寄存器操作和加载多个寄存器相似，只是没有最后一个周期。重新启动的问题在这里非常简单，没有必要对付大规模的寄存器覆盖。指令操作周期如下表所示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
1 register	1	pc+8	1	0	(pc+8)	0	0	0
	2	alu	1	1	Ra	0	0	1
		pc+12						
n registers	1	pc+8	1	0	(pc+8)	0	0	0
(n>1)	2	alu	1	1	Ra	0	1	1
	•	alu+•	1	1	R•	0	1	1
	n	alu+•	1	1	R•	0	1	1
	n+1	alu+•	1	1	R•	0	0	1
		pc+12						

表13：存储多个寄存器指令操作周期

7.8 数据交换

数据交换和加载，存储寄存器指令相似，但是实际的交换发生在周期2和3。在第二个周期，数据从外部存储器取出，在第三个周期，源寄存器中的内容写到外部的存储器。在周期四，周期二中读出的数据写到目的寄存器。操作周期下表显示。

ARM7在数据交换操作周期应驱动LOCK在（周期2和周期3）为高电平，表明这两个周期执行期间不允许中断。

数据交换可以是8位或者32位操作。

在读或者写周期，数据交换操作可能被异常中断，这两种情况都不影响目的寄存器。

Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	LOCK
1	pc+8	1	0	(pc+8)	0	0	0	0
2	Rn	b/w	0	(Rn)	0	0	1	1
3	Rn	b/w	1	Rm	1	0	1	1
4	pc+12	1	0	-	0	1	1	0
	pc+12							

表14：数据交换指令周期操作

7.9 软件中断和异常入口

异常（和软件中断）强迫PC为特定的值，并从此地址重新加载指令管道。在第一个周期，被强制的地址建立，并产生模式转换。返回地址保存到R14，CPSR保存到SPSR_svc。

在第二个周期，修改返回地址以便返回，即使这个修改比分枝联结中的意义更小。

第三个周期只是为了重新加载指令管道。操作周期在下表显示。

Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	nTRANS	Mode
1	pc+8	1	0	(pc+8)	0	0	0	C	old mode
2	Xn	1	0	(Xn)	0	1	0	1	exception mode
3	Xn+4	1	0	(Xn+4)	0	1	0	1	exception mode
	Xn+8								

表15：软件中断指令操作周期

其中C代表决定当前模式相关的值

对软件中断来讲，pc是SWI指令的地址。对中断和复位来讲，pc是在进入中断处理之前最后一条被执行的指令下一条指令的地址。对于预取异常，pc时异常指令的地址。对于数据异常，pc是产生数据异常的指令的下一条指令的地址。Xn是相应的陷阱地址。

7.10 协处理器操作

协处理器操作是ARM7请求协处理器开始某些操作，这些操作不需要在某个时间段中完成，但是协处理器在驱动CPB为低之前,必须应答去处理这些操作。

如果协处理器从不能执行被请求的任务，它应该驱动CPA,CPB为高，如果它可以完成此任务，但是又不能马上应答，它应该驱动CPA为低而CPB为高直到它可以应答。ARM7将处于busy-wait 状态，直到CPB为低。操作周期在表16中显示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	nCPI	CPA	CPB
ready	1	pc+					0	0	0	0	0
		pc+									
not ready	1	pc+8	1	0	(pc+8)	1	0	0	0	0	1
	2	pc+8	1	0	-	1	0	1	0	0	1
	•	pc+8	1	0	-	1	0	1	0	0	1
	n	pc+8	1	0	-	0	0	1	0	0	0
		pc+12									

表16：协处理器数据操作指令操作周期

7.11 协处理器数据传输（从存储器到协处理器）

当协处理器准备好接收数据时，协处理器响应传输，当CPB为低，ARM7将产生一个地址，希望协处理器在连续周期ｒａｔｅ上取走数据，协处理器决定要传输的字数，通过驱动CPA,CPB为高表明最后一个传输周期。

ARM7在第一个周期（和任何busy-wait周期）产生传输地址，在传输周期执行地址回写。操作周期在表17中显示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	nCPI	CPA	CPB
1 register	1	pc+8	1	0	(pc+8)	0	0	0	0	0	0
ready	2	alu	1	0	(alu)	0	0	1	1	1	1
		pc+12									
1 register	1	pc+8	1	0	(pc+8)	1	0	0	0	0	1
not ready	2	pc+8	1	0	-	1	0	1	0	0	1
	•	pc+8	1	0	-	1	0	1	0	0	1
	n	pc+8	1	0	-	0	0	1	0	0	0
	n+1	alu	1	0	(alu)	0	0	1	1	1	1
		pc+12									
n registers	1	pc+8	1	0	(pc+8)	0	0	0	0	0	0
(n>1)	2	alu	1	0	(alu)	0	1	1	1	0	0
ready	•	alu+•	1	0	(alu+•)	0	1	1	1	0	0
	n	alu+•	1	0	(alu+•)	0	1	1	1	0	0
	n+1	alu+•	1	0	(alu+•)	0	0	1	1	1	1
		pc+12									
m registers	1	pc+8	1	0	(pc+8)	1	0	0	0	0	1
(m>1)	2	pc+8	1	0	-	1	0	1	0	0	1
not ready	•	pc+8	1	0	-	1	0	1	0	0	1
	n	pc+8	1	0	-	0	0	1	0	0	0
	n+1	alu	1	0	(alu)	0	1	1	1	0	0
	•	alu+•	1	0	(alu+•)	0	1	1	1	0	0
	n+m	alu+•	1	0	(alu+•)	0	1	1	1	0	0
	n+m+1	alu+•	1	0	(alu+•)	0	0	1	1	1	1
		pc+12									

表17：协处理器数据传输指令操作周期

7.12 协处理器数据传输（从协处理器到存储器）

ARM7严格地控制协处理器到存储器的指令，和从存储器到协处理器传输的指令一样，只是在传输周期nRW反向(表示传输方向)。操作周期在表18中显示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	nCPI	CPA	CPB
1 register	1	pc+8	1	0	(pc+8)	0	0	0	0	0	0
ready	2	alu	1	1	CPdata	0	0	1	1	1	1
		pc+12									
1 register	1	pc+8	1	0	(pc+8)	1	0	0	0	0	1
not ready	2	pc+8	1	0	-	1	0	1	0	0	1
	•	pc+8	1	0	-	1	0	1	0	0	1
	n	pc+8	1	0	-	0	0	1	0	0	0
	n+1	alu	1	1	CPdata	0	0	1	1	1	1
		pc+12									
n registers	1	pc+8	1	0	(pc+8)	0	0	0	0	0	0
(n>1)	2	alu	1	1	CPdata	0	1	1	1	0	0
ready	•	alu+•	1	1	CPdata	0	1	1	1	0	0
	n	alu+•	1	1	CPdata	0	1	1	1	0	0
	n+1	alu+•	1	1	CPdata	0	0	1	1	1	1
		pc+12									
m registers	1	pc+8	1	0	(pc+8)	1	0	0	0	0	1
(m>1)	2	pc+8	1	0	-	1	0	1	0	0	1
not ready	•	pc+8	1	0	-	1	0	1	0	0	1
	n	pc+8	1	0	-	0	0	1	0	0	0
	n+1	alu	1	1	CPdata	0	1	1	1	0	0
	•	alu+•	1	1	CPdata	0	1	1	1	0	0
	n+m	alu+•	1	1	CPdata	0	1	1	1	0	0
	n+m+1	alu+•	1	1	CPdata	0	0	1	1	1	1
		pc+12									

表18：协处理器数据传输指令操作周期

7.13 协处理器寄存器传输（从协处理器加载）

这里busy-wait周期跟上面一样多，传输被限制在一个数据字，在第三个周期ARM7将这个字写到目的寄存器中，向所有ARM7寄存器加载指令一样，第三个周期和下一个值领预取周期合并成一个存储器N-cycle。操作周期在表19中显示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	nCPI	CPA	CPB
ready	1	pc+8	1	0	(pc+8)	1	1	0	0	0	0
	2	pc+12	1	0	CPdata	1	0	1	1	1	1
	3	pc+12	1	0	-	0	1	1	1	-	-
		pc+12									
not ready	1	pc+8	1	0	(pc+8)	1	0	0	0	0	1
	2	pc+8	1	0	CPdata	1	0	1	0	0	1
	•	pc+8	1	0	-	1	0	1	0	0	1
	n	pc+8	1	0	-	1	1	1	0	0	0
	n+1	pc+12	1	0	CPdata	1	0	1	1	1	1
	n+2	pc+12	1	0	-	0	1	1	1	-	-
		pc+12									

表19：协处理器寄存器传输（从协处理器加载）

7.14 协处理器寄存器传输（存储到协处理器中）

像从协处理器加载一样，只是省略了最后一个周期。操作周期时序在表20中显示。

	Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	nCPI	CPA	CPB
ready	1	pc+8	1	0	(pc+8)	1	1	0	0	0	0
	2	pc+12	1	1	Rd	0	0	1	1	1	1
		pc+12									
not ready	1	pc+8	1	0	(pc+8)	1	0	0	0	0	1
	2	pc+8	1	0	-	1	0	1	0	0	1
	•	pc+8	1	0	-	1	0	1	0	0	1
	n	pc+8	1	0	-	1	1	1	0	0	0
	n+1	pc+12	1	1	Rd	0	0	1	1	1	1
		pc+12									

表20 ：协处理器寄存器传输（存储到协处理器）

7.15 无定义指令和协处理器Absent

当一个协处理器检测到一条不能执行的协处理器指令，包括所有的无定义指令，协处理器必须放弃驱动CPA,CPB为低，它们将保持高，导致无定义指令陷阱产生。操作周期时序在表21显示。

Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC	nCPI	CPA	CPB	nTRANS	Mode
1	pc+8	1	0	(pc+8)	1	0	0	0	1	1	Old	Old
2	pc+8	1	0	-	0	0	0	1	1	1	Old	Old
3	Xn	1	0	(Xn)	0	1	0	1	1	1	1	00100
4	Xn+4	1	0	(Xn+4)	0	1	0	1	1	1	1	00100
	Xn+8											

表21：无定义指令操作周期时序

7.16 未执行的指令

任何条件不满足的指令都不会执行，它将增加一个周期to the execution time of 嵌入的代码段。

Cycle	Address	nBW	nRW	Data	nMREQ	SEQ	nOPC
1	pc+8	1	0	(pc+8)	0	1	0
	pc+12						

表22：未执行的指令周期操作

7.17 指令速率总结

由于CPU的流水线特性，指令通常相互交迭。在一个典型周期，一条指令可能正在用到数据通路，下一条指令译码，再下一个指令取指。基于这个原因，下表列出了一个指令需要的 incremental 周期数，而不是指令只用了处理器部分资源的整个周期数。对一个程序来讲，运行的时间（以周期位单位）可以通过表23来计算。这些图表假设指令真正执行，未执行指令花费一个周期。

Instruction	Cycle count	Additional
Data Processing	1S	+ 1I for SHIFT(Rs) + 1S + 1N if R15 written
MSR, MRS	1S	
LDR	1S + 1N + 1I	+ 1S + 1N if R15 loaded
STR	2N	
LDM	nS + 1N + 1I	+ 1S + 1N if R15 loaded
STM	(n-1)S + 2N	
SWP	1S + 2N + 1I	
B,BL	2S + 1N	
SWI, trap	2S + 1N	
CDP	1S + bI	
LDC,STC	(n-1)S + 2N + bI	
MCR	1N + bI + 1C	
MRC	1S + (b+1)I + 1C	

表23：ARM指令速度总结

n　是要传输的字数，m是乘法算法所需要的周期数,由Rs中的内容决定。在*2^(2m-3)*和*2^(2m-1)-1* 之间的任何数，乘法都需花费1S+mI *m* 周期，1<m>16。乘以0或者1花费1S+1I周期，被任何一个大于或等于2^(29)乘法花费1S+16I周期。任何一个乘法花费的最大时间为1S+16I周期。

如果乘数的[32:16]位全为0 或者全为1， m为2
其它情况下m为4。

b　是在协处理器busy-wait loop花费的周期数。

如果条件不满足所有的指令花费一个S-周期。周期类型N,S,I,C在第五章存储器接口中定义。

8.0 DC特性

8.1 绝对最大频率

Symbol	Parameter	Min	Max	Units	Note
VDD	Supply voltage	VSS-0.3	VSS+7.0	V	1
Vin	Input voltage applied to any pin	VSS-0.3	VDD+0.3	V	1
Ts	Storage temperature	-40	125	deg C	1

表24：ARM7 DC最大频率特性

注意：These are stress ratings only。超过最大频率特性将永久伤害器件。在最大频率特性上操作器件超过一段时间将影响器件可靠性。

8.2 DC 操作条件

Symbol	Parameter	Min	Typ	Max	Units	Notes
VDD	Supply voltage	4.5	5.0	5.5	V	
Vihc	Input HIGH voltage	3.5		VDD	V	1
Vilc	Input LOW voltage	0.0		1.5	V	1
Ta	Ambient operating temperature	0		70	deg.C	

表25：ARM7 操作条件

注意：电压测量基于VSS.

9.0 AC 参数

这里给出的时序参数只是一些初步的数据，当器件特性完善，这些数据可以改变。

这部分的时序图假定ARM7的输出部分已经加载了容性负载，如表26测试负载所示。这些负载已经被当作有ARM7单元的系统的典型负载。

ARM7输出单源驱动器为 CMOS inverters。它表明传播延迟时间随着负载电容的增长呈线性增长。这里给出每一个输出驱动器的`Output derating' 图，显示出了随着负载电容的增加输出时间增加的大约比例。

Output Signal	Test Load (pF)	Output derating (ns/pF)
D[31:0]	5	0.5
A[31:0]	5	0.5
LOCK	2	0.5
nCPI	2	0.5
nMREQ	2	0.5
SEQ	2	0.5
nRW	2	0.5
nBW	2	0.5
nOPC	2	0.5
nTRANS	2	0.5

表26：AC 测试负载

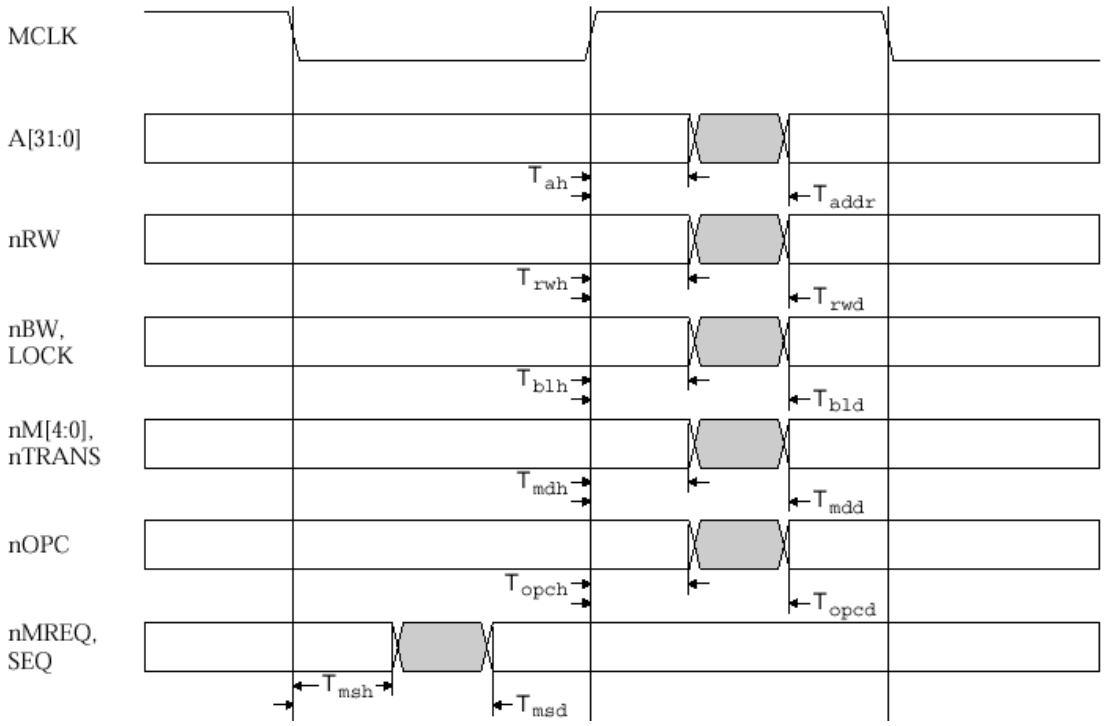


图35：通用时序

注意： 在显示的周期，**nWAIT** 和 **ALE** 都为高电平。

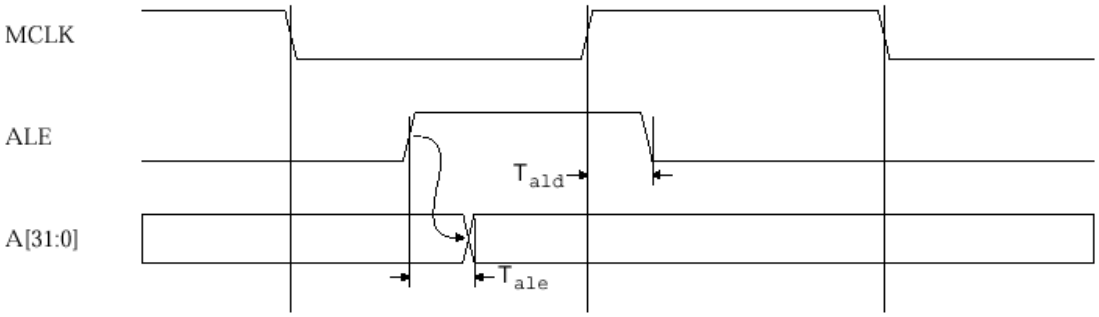


图36：地址时序

注意：在phase2， Tald期间，为了锁存当前地址，ALE驱动为低电平，如果ALE在 Tald 之后驱动为低电平，将锁存一个新地址。

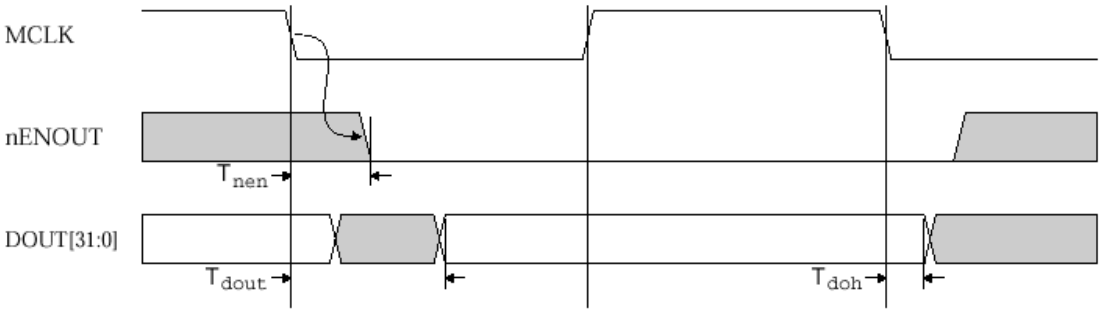


图37：数据写周期

注意：在图所示周期，DBE为高。

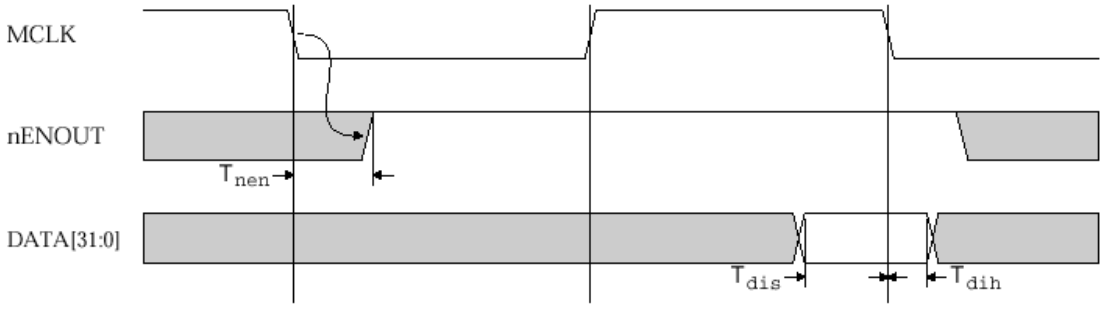


图38：数据读周期

注意：在图所示周期，DBE为高。

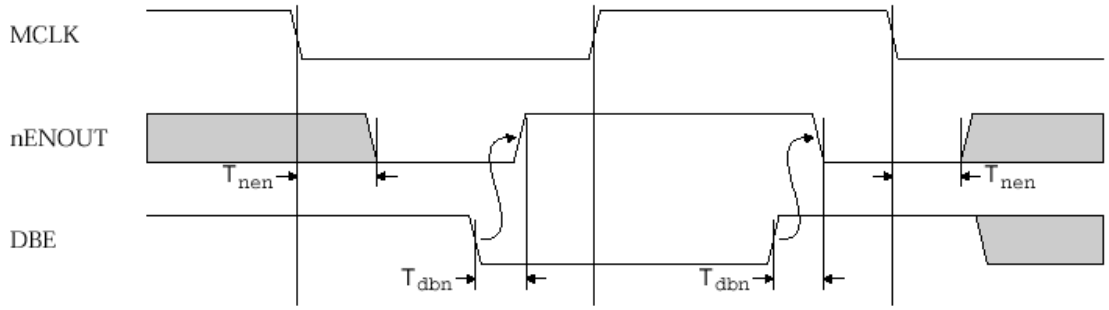


图39：数据总线控制

注意：所示周期为数据写周期，因为 **nENOUT** 在 **phase1** 为低。这里，DBE用于改变**nENOUT**的状态。

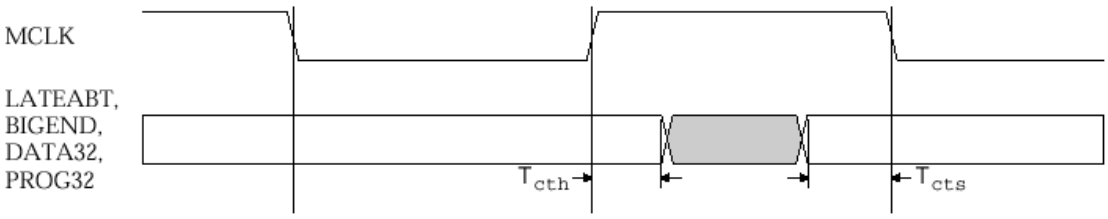


图40：管脚配置时序图

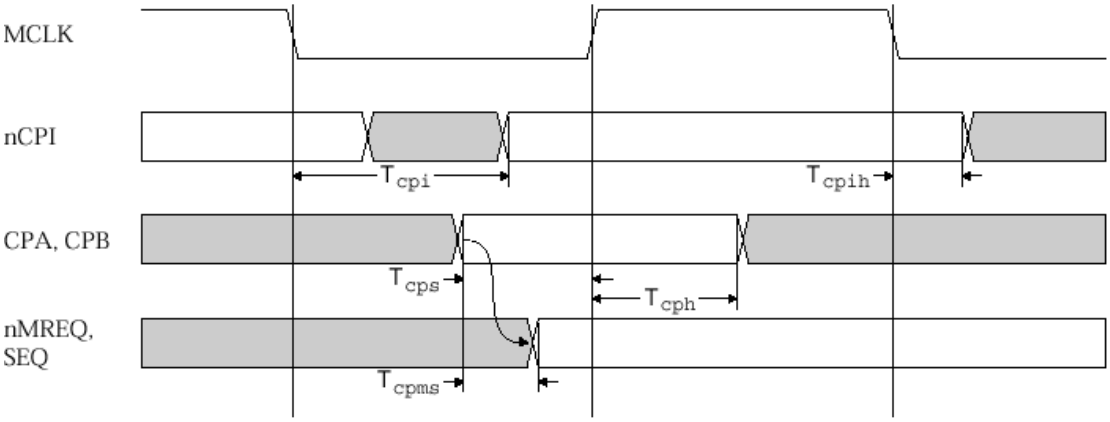


图41：协处理器时序图

注意：通常，在MCLK下降沿之后，**nMREQ 和 SEQ** 有效Tmsd。在这个周期，ARM已经处于busy-waiting状态，等待协处理器完成当前指令。如果CPA和CPB在 phase1 改变，**nMREQ** 和 **SEQ** 的时序依靠Tcpms。大多数系统在 phase2 之前应该能够产生CPA和CPB,所以，**nMREQ** 和 **SEQ** 的延迟时间总为Tmsd。

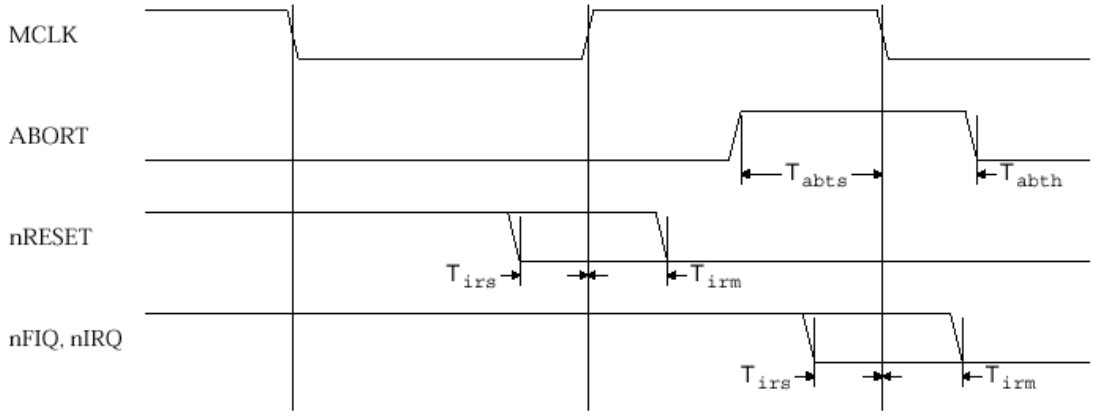


图42：异常时序

注意：Tirs 通过响应的时钟沿保证识别中断或者复位信号。Tirm保证在响应的时钟沿不被识别。guarantees non-recognition by that clock edge. 这些输入可以异步地充分应用于精确识别周期不是很重要的场合。

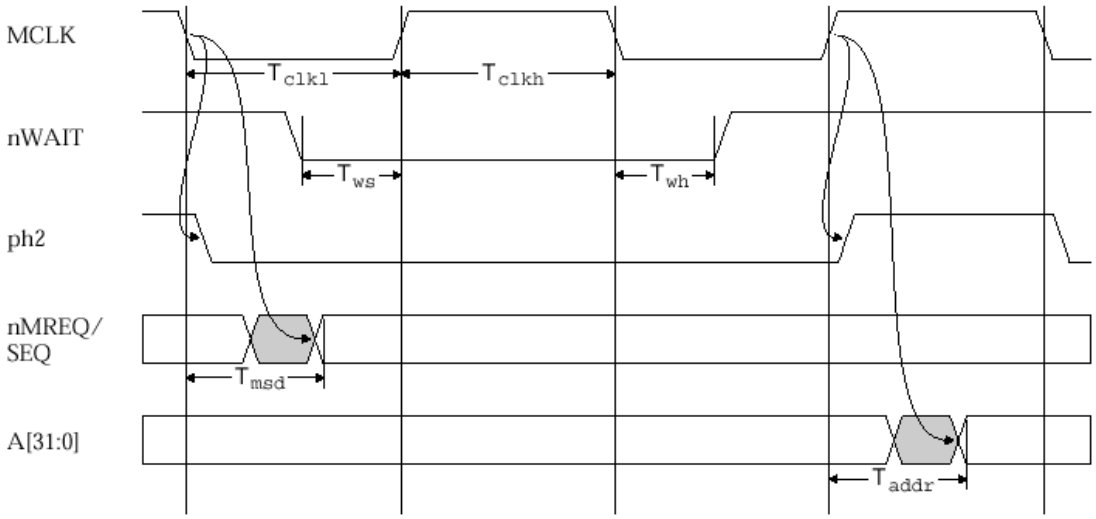


图43：时钟时序

注意：MCLK 的高电平周期并没有定时ARM7 核，ARM7的系统时钟是由MCLK 和 nWAITE 相与组成的。如上所示，在MCLK第一个低相位 **nMREQ** 和 **SEQ** 改变，并且在MCLK第二个高相位 **A[31:0]** 改变。参考ph2，ph2是ARM7 核内部的时钟，定时ARM7的所有动作。

Symbol	Parameter	Min	Max
Tckl	clock LOW time	21	
Tckh	clock HIGH time	21	
Tws	nWAIT setup to CKr	3	
Twh	nWAIT hold from CKf	3	
Tale	address latch open		4
Tald	address latch time		4
Taddr	CKr to address valid		12
Tah	address hold time	5	
Tnen	nENOUT output delay		2
Tdbn	DBE to nENOUT delay		11
Tdout	data out delay		17
Tdoh	data out hold	5	
Tdis	data in setup	0	
Tdih	data in hold	5	
Tabts	ABORT setup time	10	
Tabth	ABORT hold time	5	
Tirs	interrupt setup	6	
Tirm	Interrupt non-recognition time		TBD
Trwd	CKr to nRW valid		21
Trwh	nRW hold time	5	
Tmsd	CKf to nMREQ & SEQ		23
Tmsh	nMREQ & SEQ hold time	5	
Tbld	CKr to nBW & LOCK		21
Tblh	nBW & LOCK hold	5	
Tmdd	CKr to nTRANS/nM[4:0]		21
Tmdh	nTRANS/nM[4:0] hold	5	
Topcd	CKr to nOPC valid		11
Topch	nOPC hold time	5	
Tcps	CPA, CPB setup	7	
Tcph	CPA,CPB hold time	2	
Tcpms	CPA, CPB to nMREQ, SEQ		15
Tcpi	CKf to nCPI delay		11
Tcpih	nCPI hold time	5	
Tcts	Config setup time	10	
Tcth	Config hold time	5	

Table 27: Provisional AC Parameters (units of ns)

9.1 AC参数注意事项

所有的图像都是临时的，假定1微米CMOS技术已经应用到包含ARM7的ASIC制造工艺。

10.0 附录-向下兼容性

两个输入，PROG32 和 DATA32，允许处理器按照以下方式配置：

（1） 26位程序和数据空间-（**PROG32**低, **DATA32**低)。这个配置强制ARM7像以前26位地址空间的ARM一样操作。编程模式也要按照26位地址空间的ARM进行。但是新的指令的操作比如访问CPSR和SPSR寄存器详细情况如本文所述。在这种模式下，选择32位的操作模式是不可能的，所有的异常（包括地址异常）都在相应的26位模式下进入异常处理。

（2） 26位程序空间32位数据空间（**PROG32** 低, **DATA32** 高)。这和26 位程序和数据空间配置相同，but with address exceptions to allow data teansfer operations to access the full 32 bit address space。

（3） 32位程序和数据空间（**PROG32** 高, **DATA32** 高）。这个配置扩展地址空间为32位，下面描述了编程模式比较大的改变，支持在32位的环境下跑26位程序。

第四个处理器配置也是可能的（26位数据32位地址空间），不应该选择。

当配置为26位的程序空间，ARM7被限制在26位地址空间四个模式中的某一个－26位模式。这些模式跟之前的ARM处理器兼容，具体如下：

User26
FIQ26
IRQ26 and
Supervisor26

这些都是在26为地址空间下一般的操作模式，26位地址模式提供前下兼容，允许执行在以前ARM处理器上写的程序。