第 1 章　Cortex–A8 处理器

Cortex-A8 是一款成功的 ARM 内核。作为嵌入式应用系统的关键组成部分，目前已被广泛应用于移动终端、掌上电脑和许多其他日常便携式消费电子设备。

本章主要内容:

（1）Cortex-A8 处理器内部组成结构;

（2）Cortex-A8 处理器的编程模型;

（3）Cortex-A8 处理器的时钟、复位和电源控制等功能单元的管理机制。

通过本章的介绍，对 Cortex-A8 处理器内部结构和各部分功能可以有一个初步了解，为后续章节的学习打下基础。

1.1　概　　述

Cortex-A8 处理器是一款高性能、低功耗、完整虚拟内存管理能力并具有高速缓存的应用处理器。

1．处理器特性

（1）支持 ARM 体系结构的 v7-A 指令集。

（2）使用内部 AXI（Advanced Extensible Interface）接口，可将主存配置为 64 位或 128 位高速 AMBA（Advanced Microprocessor Bus Architecture）模式，来支持多种数据传输行为。

（3）一条用于执行整数指令的流水线。

（4）一条用于执行 SIMD 和 VFP 指令集的 NEON 流水线，为多媒体应用提供硬件加速。

（5）使用分支目标地址缓存和全局历史缓冲区实现动态分支预测。

（6）MMU（Memory Management Unit）。用于内存单元管理。

（7）L1 级指令缓存和数据缓存可配置为 16KB 或 32KB。

（8）L2 缓存大小可配置为 0KB 或 128KB～1MB。

（9）L2 缓存可配置奇偶校验模式和纠错码（Error Correction Code，ECC）。

（10）嵌入式跟踪宏单元（Embedded Trace Macrocell，ETM）支持在线调试。

（11）动态和静态的电源管理方案（Intelligent Energy Management，IEM）。

（12）用于调试的探针和断点寄存器，支持片上调试。

2．ARMv7 架构指令集

（1）采用 ARM Thumb-2 指令集，兼容 Thumb 和 ARM 指令。

（2）使用 Thumb-2（Thumb-2EE）技术，提高运行速度。

（3）增强型的安全功能（NEON），有利于安全应用程序的开发。

（4）高级 SIMD（Single Instruction Multiple Data）架构扩展，用于多媒体三维图形和图像处理的加速技术。

（5）用于浮点计算的向量浮点 v3（Vector Floating Point v3，VFPv3）架构，兼容 IEEE 754 标准。

1.2 处理器组成结构

Cortex-A8 处理器的主要组成部分包含 指令取指单元、指令解码单元、指令执行单元、Load/Store 单元、L2 缓存（上述 5 部分统称为内核）、NEON 和 ETM 单元。其内部结构如图 1.1 所示。

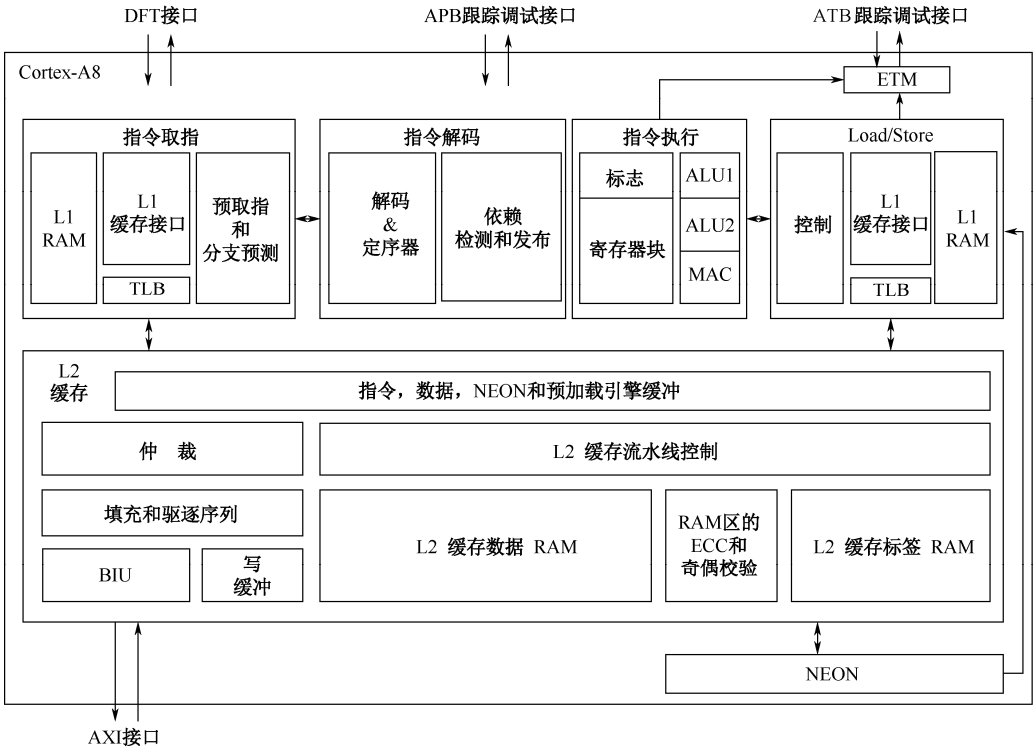

图 1.1 Cortex-A8 处理器结构图

1.2.1 内部功能单元

1．指令取指单元

指令取指单元负责完成指令流预测，从 L1 指令缓存中取出指令代码，并将所获取指令送到解码流水线进行解码。指令取指单元还包括 L1 指令缓存。

2．指令解码单元

指令解码单元依次对 ARM 和 Thumb-2 指令完成解码工作。这些指令可以来自调试控制协处理器（CP14）、指令和系统控制协处理器（CP15）。

当存在多条指令需要解码时，指令解码单元的执行顺序是：异常、调试事件、复位初始化、内存内置自测试（MBIST）、等待中断处理和其他异常事件。

3．指令执行单元

指令执行单元包含两条对称的算术逻辑单元（ALU）流水线、一个用于加载和存储指令的地址发生器和多条专用功能流水线，其中执行流水线同时负责寄存器回写。

指令执行单元的作用：

（1）执行所有整数 ALU 和乘法运算指令，包括产生标志位；

（2）为 Load/Store 指令生成虚拟地址和基地址；

（3）为 Load/Store 指令提供格式化数据；

（4）处理分支和其他变化的指令流，评估指令条件码。

4．Load/Store 单元

Load/Store 单元含有完整的 L1 数据存储系统和整数 Load/Store 指令流水线，包括 L1 数据缓存、数据 TLB、整数存储缓冲区、NEON 存储缓冲区、整数数据加载地址边沿对齐和格式化以及整数数据存储数据地址边沿对齐和格式化。

5．L2 缓存

L2 高速缓存单元包括 L2 缓存和缓冲接口单元（BIU）。它服务于被 L1 缓存错过的预取指和 Load/Store 指令部分。

6．NEON

NEON 单元包括完整 10 条流水线，用于解码和执行 SIMD 媒体指令集。NEON 单元包括 NEON 指令队列、NEON 加载数据队列、两条 NEON 解码逻辑流水线、三条用于 SIMD 整数指令的执行流水线、两条用于 SIMD 浮点指令的执行流水线、一条用于 SIMD 和 VFP 的 Load/Store 指令执行流水线和完全执行 VFPv3 的数据处理指令集 VFP 引擎。

7．非侵入式跟踪宏单元（ETM）

ETM 过滤和压缩那些用于系统调试和系统分析而需要跟踪的指令和数据。ETM 单元在处理器外部有一个专用 ATB 接口。

1.2.2 处理器外部接口

处理器含有以下扩展接口与外部相连：与 AMBA 总线相连的 AXI 接口、APB 跟踪调试接口、ATB 跟踪调试接口和 DFT 接口。

（1）AXI 接口是系统总线主要接口，为指令和数据完成 L2 缓存的填充和非缓存类访问。AXI 接口支持 64 位或 128 位的输入和输出数据总线宽度。AXI 总线上支持多个未处理请求。AXI 信号与时钟输入同步，使用时钟允许信号 ACLKEN 可以获得一个较大总线占用比。

（2）APB 跟踪调试接口可以访问 ETM、CTI 和用于寄存器的调试。

（3）ATB 跟踪调试接口可以输出跟踪信息用于调试。

（4）DFT 接口为制造商提供内存内置自测试和自动测试方案。

1.2.3 可配置的操作

对于内部单元, Cortex-A8 处理器提供了灵活的配置方案。处理器内核可配置选项参见表 1.1。

表 1.1 内核可配置选项

状 态	可 配 置 项
AXI 总线宽度	64bit/128bit
L1 RAM 容量	16KB/32KB
L2 RAM 容量	0KB/128KB/256KB/512KB/1MB
L2 Parity/ECC 校验	Yes/No
ETM	Yes/No
IEM	支持内核所有组成部分的电源域管理
NEON	Yes/No。当处理器配置为 No 模式时，所有 SIMD 和 VFP 的指令将尝试进入未定义模式

1.3 编 程 模 型

Cortex-A8 处理器通过设置可以工作在 ARM 或 Thumb 状态。工作在 ARM 状态时，支持 32 位 ARM 指令集；工作在 Thumb 状态时，支持 16/32 位的 Thumb 指令集。通过实现 SIMD 架构，

增加了主要针对音频、视频、3D 图形、图像和语音处理的指令。在介绍指令集前，需要了解处理器的编程模型（Programmer's Model）。

1.3.1 内核数据流模型

Cortex-A8 内核采用冯·诺依曼结构，这是一种将程序指令存储器和数据存储器合并在一起的存储器结构。程序指令存储地址和数据存储地址指向同一个存储器的不同物理位置，由于公用数据总线，因此程序指令总线和数据总线的宽度相同，总线宽度为 32bit。

Cortex-A8 内核支持的 ARM 指令集为 RISC 集。其特点是所有数据都需要在内核的寄存器中处理，不支持在内存中直接操作数据。通过 Load/Store 结构，使用 load 指令将数据从内存中复制到寄存器中，使用 store 指令将寄存器中的数据复制到内存中。

从编程角度来看，ARM 内核由指令解码器（Instruction Decoder）、桶形移位器（Barrel Shifter）、符号扩展单元（Sign Extend）、R 寄存器组（Register File）、ALU（算术逻辑单元）、MAC（乘累加单元）、地址寄存器（Address Register）、地址加法器（Incrementer）等部分组成。各部分通过内部数据总线相连，其连接结构如图 1.2（参考来自 ARM System Developer's Guide）所示。

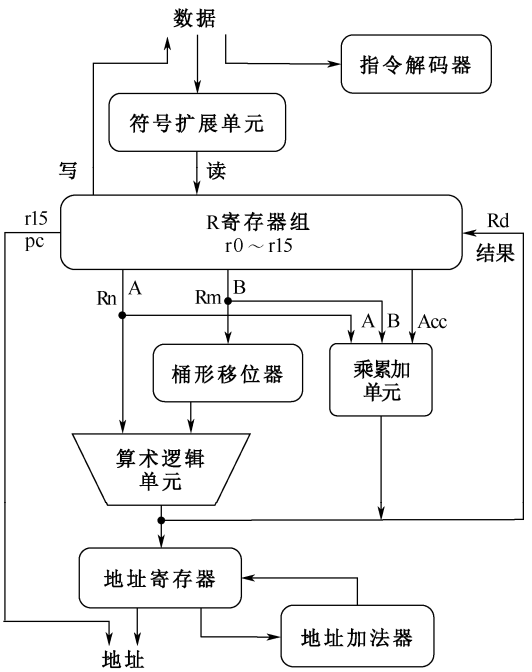

图 1.2 ARM 内核数据流模型

符号扩展单元：存储单元中定义的有符号 8/16 位数据类型的数据在这里被扩展到 32 位后送入寄存器。

R 寄存器组：由 16 个 32 位寄存器组成。可以提供 Rn 和 Rm 两个寄存器，用于存储 ALU 或 MAC 的源操作数；一个 Rd 寄存器，用于记录操作结果（目的操作数）。其中，Rm 支持算术逻辑运算前的移位操作，可用于位操作运算或灵活的寻址方案。指定 r15 寄存器作为 PC（程序计数器寄存器），它为地址总线提供指令代码存放位置的 32 位地址信息。

算术逻辑单元（Arithmetic Logic Unit）：针对来自 Rn 和 Rm（或 N）两个寄存器的源操作数，实现指令规定的运算。运算结果若是数据则送到 Rd 寄存器中，若是地址则送到地址寄存器中，并广播到地址总线中。

乘累加单元（Multiply-Accumulate Unit）：实现硬件乘法或乘-加运算。

地址加法器：当处理器使用 Load/Store 指令访问连续的存储空间时，可以通过地址加法器得到下一个存储单元地址。

1.3.2 工作模式

Cortex-A8 处理器有 8 种工作模式。

（1）系统模式（System）。操作系统的特权用户模式。

（2）用户模式（User）。ARM 程序的正常工作模式，并用于执行大多数应用程序。该模式下，限制内存的直接访问和通过物理地址对硬件设备的读/写操作。

（3）快速中断模式（FIQ）。用于处理快速中断。

（4）中断模式（IRQ）。用于通用中断处理。

（5）超级用户模式（SVC）。ARM 内核上电时处于 SVC 模式，主要用于 SWI（软件中断）和受保护的操作系统模式。

（6）中止模式（Abort）。数据中止或预取中止的异常事件发生后进入中止模式。

（7）未定义模式（Undefined）。当执行一个未定义指令的异常事件发生时，进入未定义模式。

（8）监视模式（Monitor）。是一种安全模式，用于执行安全监视代码。

除用户模式以外的 7 种模式称为特权模式。特权模式用于服务中断或异常，或访问受保护资源，具有对 CPSR 寄存器的读/写控制权。

1.3.3 寄存器结构

Cortex-A8 处理器有 40 个 32 位寄存器, 分为 33 个通用寄存器和 7 个程序状态寄存器。ARM 正常状态下有 16 个数据寄存器和 1～2 个状态寄存器可随时被访问，可被访问的寄存器依赖于处理器当前的工作模式。每种工作模式下处理器可访问的寄存器名称见表 1.2。

表 1.2　基于 ARM 工作模式的寄存器分组

系　统	用　户	快速中断	中　断	超级用户	中　止	未定义	监　视
r0	r0	r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7	r7	r7
r8	r8	r8_fiq	r8	r8	r8	r8	r8
r9	r9	r9_fiq	r9	r9	r9	r9	r9
r10	r10	r10_fiq	r10	r10	r10	r10	r10
r11	r11	r11_fiq	r11	r11	r11	r11	r11
r12	r12	r12_fiq	r12	r12	r12	r12	r12
r13(SP)	r13(SP)	r13_fiq	r13_irq	r13_svc	r13_abt	r13_und	r13_mon
r14(LR)	r14(LR)	r14_fiq	r14_irq	r14_svc	r14_abt	r14_und	r14_mon
r15(PC)	PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_fiq	SPSR_irq	SPSR_svc	SPSR_abt	SPSR_und	SPSR_mon

下面依照寄存器的功能进行划分。

r0～r12 是通用寄存器，可用来保存数据项或表示地址信息的数值。其中，快速中断工作模式具有私有的 r8～r12 寄存器。

r13 是堆栈指针寄存器（Stack Pointer，SP），用于指向堆栈区的栈顶。每种工作模式具有各自私有的堆栈区和堆栈指针寄存器。

r14 是链接寄存器（Link Register，LR），用于存储子程序返回主程序的链接地址。当处理器执行一条调用指令（BL 或 BLX）时，r14 用于存储主程序的断点地址，供子程序返回主程序；其他时间，r14 可以作为一个通用寄存器使用。每种工作模式具有各自私有的链接寄存器。

r15 是程序计数器（Program Counter，PC），用于存放下一条指令所在存储单元的地址。由于 ARM 指令集中的一条指令代码为 4 字节，因此在取指时指令代码的存储地址应满足字对齐，即 r15[1:0]=b00。

CPSR 是当前程序状态寄存器（Current Program Status Register，CPSR）。寄存器中包含条件码标志、状态位和当前工作模式位。除了系统模式外，其他每种特权模式额外拥有一个备份程序状态寄存器（Saved Program Status Register，SPSR），用来保存当异常发生需要进入其他模式时所产生的程序断点处状态信息。通常是将断点的 CPSR 内容复制到 SPSR 中，以便在异常处理程序结束并返回时恢复断点处的程序状态。

8 种工作模式都有各自的 16 个数据寄存器和状态寄存器。依照每种模式的特点，这些寄存器又分为公有和私有两种情况。

公有寄存器：如程序计数器寄存器，8 种工作模式公用一个 r15（PC）。

私有寄存器：8 种工作模式拥有各自独立的寄存器，如堆栈指针 r13（SP）。在寄存器命名过程中，使用添加表示工作模式下标的方法加以区分，如 r13_fiq。此时虽然是表示堆栈指针寄存器（各种模式的此功能寄存器，在编译环境下同为 r13），但是快速中断模式与其他模式有不同的物理空间。在使用过程中，需要通过设置 CPSR 中的模式位来表明处理器当前的工作模式，以确认当前模式下寄存器组的物理位置。8 种工作模式的私有寄存器可参见表 1.2 中加阴影单元中的命名。

1.3.4 程序状态寄存器

Cortex-A8 处理器的程序状态寄存器包含 1 个主程序使用的 CPSR 寄存器和 6 个异常处理程序使用的 SPSR 寄存器。

程序状态寄存器（CPSR 和 SPSR）的主要用途：保存所执行的最后一条逻辑或算术运算指令运行结果的相关信息，控制开启/禁用中断，设置处理器工作模式。程序状态寄存器位定义见表 1.3。

表 1.3 程序状态寄存器（CPSR 和 SPSR）位定义

位	31	30	29	28	27	26～25	24	23～20	19～16	15～10	9	8	7	6	5	4～0
定义	N	Z	C	V	Q	IT[1:0]	J	DNM	GE[3:0]	IT[7:2]	E	A	I	F	T	M[4:0]

表 1.3 中，“ 位 ”一行表示了组成程序状态寄存器的 32 位二进制数排序位置。最右边的规定为第 0 位，依次向左排序，最左边的为程序状态寄存器的最高位第 31 位。“ 定义 ”一行表示了程序状态寄存器中一位二进制数所代表的含义。理解位定义的概念要注意两方面：位在寄存器中的排序位置，位的内容（1/0）所代表意义。

在描述位定义过程中，其位置通常使用其所代表意义的英文缩写字母来表示。例如，程序状态寄存器中第 30 位的内容用来表示当前算术运算指令运行结果是否为零，为了便于描述，将该位定义为“ Z ”标志位。

1．条件代码标志位

N（CPSR[31]）：负数或小于标志位。N=1 表示当前指令运行结果为负数（非正数或零）。

Z（CPSR[30]）：零标志位。Z=1 表示当前指令运行结果为零，常表示两个操作数相等。

C（CPSR[29]）：进位/借位/延伸标志位。C=1 表示两个无符号数相加结果溢出。

V（CPSR[28]）：溢出标志位。V=1 表示两个有符号数相加结果溢出。

Q（CPSR[27]）：若执行乘法和分数算术运算指令（QADD、QDADD、QSUB、QDSUB、SMLAD、SMLAxy、SMLAWy、SMLSD、SMUAD、SSAT、SSAT16、USAT、USAT16）后发生溢出，则 Q 标志位被置 1。此时需要执行 MSR 指令才可清除 Q 标志位，否则后续乘法和小数运算指令不

能被执行。需要判断 Q 标志位状态时，可将 CPSR 寄存器内容读入一个寄存器中，再从这个寄存器中提取 Q 标志位的内容。

GE[3:0]（CPSR[19～16]）：大于或等于标志位。某些 SIMD 指令，运算结果的大于或等于信息将影响 GE[3:0]标志位。

在执行算术/逻辑运算类或 MSR 和 LDM 等指令时，可以通过在这些指令助记符字段附加后缀“ S ”的方式，使得指令执行结果影响上述条件码标志位内容。

可以通过判断标志的内容作为执行一条指令的条件。大多数 ARM 指令可以在指令格式中的条件域附加条件属性，指定 N、Z、C、V 或 Q 等标志位的内容作为指令执行条件。在执行附加有条件域的指令之前，处理器会将指令的条件属性和 CPSR 中条件代码标志位的内容进行比较，如果匹配，则该指令执行，否则该指令被忽略。条件属性与所判断的标志位之间的对应关系可以参见本书第 2 章表 2.1。

2．状态控制位

IT（CPSR[15～10，26～25]）：条件语句执行控制位。

J（CPSR[24]）：当 T=1 且 J=0 时，设定处理器工作于 Thumb 状态。

当 T=1 且 J=0 时，设定处理器工作于 ThumbEE 状态。

当 T=0（工作于 ARM 状态）时，J 位自动清零，此时不可将 J 位置 1。

E（CPSR[9]）：决定 Load/Store 指令操作数据的端模式（大端/小端）。处理器可以通过指令将 E 位内容置 1 或清 0，也可以通过设置 CFGEND0 引脚电平来硬件配置 E 的状态。

A（CPSR[8]）：被自动设置，用来禁用不精确的数据中止。

3．控制位

I （CPSR[7]）：中断禁止位。I=1 时禁止 IRQ 中断。

F（CPSR[6]）：中断禁止位。F=1 时禁止 FIQ 中断。

T（CPSR[5]）：决定处理器的工作状态。T= 0 时处理器工作在 ARM 状态，T=1 时依据 J 位的内容决定处理器工作在 Thumb 状态或 ThumbEE 状态。

M[4:0]（CPSR[4～0]）：工作模式位。通过设定程序状态寄存器 M[4:0]的内容，可以用来决定处理器当前的工作模式，其内容与所设定处理器当前的工作模式对应关系见表 1.4。

表 1.4　模式位内容与工作模式的设定

M[4:0]	工 作 模 式
b10000	用户（User）
b10001	快速中断（FIQ）
b10010	中断（IRQ）
b10011	超级用户（SVC）
b10111	中止（Abort）
b11011	未定义（Undefined）
b11111	系统（System）
b10110	监视（Monitor）

4．使用 MSR 指令修改 CPSR 寄存器

在 ARMv6 以前架构版本中，MSR 指令可在所有模式下修改标志位字节，即 CPSR[31～24]。 但 CPSR 中其他三个字节内容只有在特权模式下可以修改。

ARMv6 的改进之处有以下几个方面。

（1）CPSR 寄存器中指定标志位在任何模式下可自由修改，通过 MSR 指令或借助其他指令执行结果，来改写或直接修改整个 CPSR 寄存器内容或其中指定标志位内容。这些位包含 N、Z、C、V、Q、GE[3:0]、E。

（2）J 和 T 位的内容不可以使用 MSR 指令，仅能依据其他指令执行结果来修改。如果使用 MSR 指令并尝试修改这些位内容，结果不可预知。

（3）I、F 和 M[4:0]等位的内容在用户模式下受到保护，仅可在特权模式下改写。用户模式下可以通过执行指令进入处理器的异常模式，在异常模式下来改写这些位的内容。

（4）只有在安全特权模式下，才可以通过直接写 CPSR 模式标志位来进入监视模式。如果内

核目前处于安全用户模式、非安全用户模式或非安全特权模式，此时设置进入监视模式的修改将被忽略。内核不可以将 SPSR 寄存器中在非安全模式下发生变化的模式标志位内容复制到 CPSR。

（5）DNM（CPSR[23～20]）为保留位。当改变 CPSR 标志或控制位时，确保不改变这些保留标志位的内容以便和未来高版本处理器兼容。

1.3.5 流水线

流水线（Pipeline）技术是指在程序执行时多条指令重叠进行操作的一种准并行处理实现技术，应用于 RISC 处理器执行指令的机制。

从图 1.1 所示的微处理器内部结构图可知，微处理器指令的执行过程分为取指（Fetch）、解码（Decode）和执行（Execute）3 个阶段。早期的微处理器顺序执行这 3 个阶段来完成指令的执行过程。当引入了流水线技术后，以 3 条流水线为例，每条流水线仅负责完成取指、解码和执行 3 个阶段中的一个环节，将指令 3 个阶段顺序操作转为在 3 条流水线上并行操作。

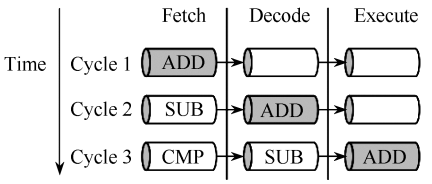

图 1.3　流水线上指令执行顺序

图 1.3 说明了流水线使用的一个简单例子，该例中微处理器需要顺序执行 3 条指令：ADD、SUB、CMP。

图 1.3 所示基于流水线的指令执行过程中，将一条指令执行过程分解为 Fetch、Decode 和 Execute 三个部分，使用 3 条流水线，每条流水线各自负责完成其中一部分，并将处理结果移送到下一条流水线。

在一个时间周期（Cycle）内，3 条流水线同时完成自己所承担的任务。当微处理器开始指令执行过程后，在每个时间周期内，各流水线所承担的任务如下：

Cycle1：流水线 1 完成取指（ADD）。

Cycle2：流水线 1 完成取指（SUB），流水线 2 完成解码（ADD）。

Cycle3：流水线 1 完成取指（CMP），流水线 2 完成解码（SUB），流水线 3 完成指令（ADD）的执行。

在接下来的时间周期中，

Cycle4：流水线 1 完成取指（CMP 指令后需要执行的第 1 条指令），流水线 2 完成解码（CMP），流水线 3 完成指令（SUB）的执行。

Cycle5：流水线 1 完成取指（CMP 指令后需要执行的第 2 条指令），流水线 2 完成解码（CMP 指令后需要执行的第 1 条指令），流水线 3 完成指令（CMP）的执行。

……

由上述分析可知，利用流水线技术在每个时间周期内，微处理器都在不同的流水线上同时处理着取指、解码和执行 3 个部分，因此极大提高了单位时间内执行指令的数量。流水线技术适合处理顺序结构的程序流程，对于分支结构和异常/中断服务程序所带来的问题以及相应解决办法可以参考相关书籍。

1.3.6 异常/中断

在执行主程序过程中，一些特殊状态的发生导致微处理器必须进行处理，称此时发生了一个异常事件。Cortex-A8 处理器目前定义的异常事件有：复位、未定义指令、软件中断、预取指令中止、数据中止、外部中断请求 IRQ、快速中断请求 FIQ 等。当然，研发人员也可以自定义一些异常事件让微处理器来处理。

1.3.6.1 异常向量表

当异常事件发生时，需要一段对应程序来处理，异常事件的处理程序称为异常服务子程序。中断事件被归类为异常事件，中断事件处理程序称为中断服务子程序。这些服务子程序在编译之后产生一个表示函数入口的编译地址，该地址会赋值给在程序中定义的表示函数名称的符号。通过将函数名称所包含的编译地址赋值给 PC，可以使得微处理器找到并运行服务子程序。

由于异常事件发生的随机性，微处理器为每个异常事件指定了一个内存地址（连续 4 个存储单元），地址中存放的内容是一条跳转到这个异常服务子程序编译地址的指令代码，这个被指定的内存地址称为异常服务子程序入口地址。由于跳转指令指明了找到服务子程序的方向，所以异常/中断服务子程序入口地址中存储的内容也称为异常/中断向量。

每个异常/中断事件都有自己的异常/中断向量，这些异常/中断向量统一存放在一个规定的内存空间中，该存储空间称为向量表。Cortex-A8 处理器的向量表见表 1.5。

表 1.5　ARM 异常向量表

入 口 地 址	异　　常	进 入 模 式	进入异常条件
0x00000000	复位 reset	管理模式	复位电平有效时
0x00000004	未定义指令 undefined_instruction	未定义模式	遇到不能处理的指令
0x00000008	软件中断 software_interrupt	管理模式	执行 SWI 指令
0x0000000c	预取指令中止 prefetch_abort	中止模式	处理器预取指令的地址不存在，或该地址不允许当前指令访问
0x00000010	数据操作中止 data_abort	中止模式	处理器数据访问指令的地址不存在，或该地址不允许当前指令访问
0x00000014	未使用 not_used	未使用	未使用
0x00000018	外部中断请求 IRQ	IRQ	外部中断请求有效，且 CPSR 中的 I 位为 0
0x0000001c	快速中断请求 FIQ	FIQ	快速中断请求引脚有效，且 CPSR 中的 F 位为 0

复位事件被定义为异常事件。复位异常事件的服务子程序入口地址是 0x00000000，此处需要放置一条跳转指令，跳转到复位事件处理子程序。地址 0x00000000 也是处理器所有程序的入口地址。

在第 4 章 start.s 文件分析中有以下指令：

```
38  global _start                /*声明全局符号*/
39 _start: b reset               /*程序入口跳转到复位程序执行*/
....
138  reset:                      /*reset函数名称，复位后首先执行该程序*/
139  bl   save_boot_params       /*可通过U-boot-spl.map文件找到该函数的定义出处*/
```

功能释义：

38 行中的全局符号“ _start ”，由编译器指定其内容为 0x00000000，用来表示处理器上电复位后开始执行第一条指令的存放地址。

39 行是一条跳转指令，要求跳转到 139 行处执行新的指令。本行中的“ _start: ”字段指明跳转指令的指令代码存放于地址号为 0x00000000 的存储单元中。

138 行定义函数名为“ reset ”。

139 行定义函数体，用于复位异常事件的处理。

其中 39 行所描述跳转指令的指令代码：复位事件的中断向量，由于一条 ARM 指令的指令代码长度为定长 4 字节，所以中断向量需要占用 4 个存储单元。

地址 0x00000000：中断向量地址。

“ reset ”函数：复位事件处理子程序。

当复位事件发生时, CPU 自动检索中断向量表将复位事件入口地址 0x00000000 赋值给 PC, 将该地址中存放的中断向量（跳转指令）取出，完成解码和执行后跳转到“ reset ”函数入口，开始执行复位事件服务子程序，来处理复位事件。

1.3.6.2 异常事件分类

1．复位（RESET）

在复位信号有效期间，处理器停止执行指令进入复位过程。在复位信号变为高电平后，复位过程结束。在复位过程期间，通过向微处理器内部寄存器赋初值的方法，确定微处理器的工作状态为：

（1）工作于超级用户模式；

（2）禁止中断；

（3）工作于 ARM 状态，需要外部信号配合；

（4）从中断向量表的复位入口地址开始执行程序。

2．快速中断请求（FIQ）

快速中断请求（FIQ）异常事件支持快速中断。在 ARM 状态下，FIQ 模式有 8 个私有寄存器（参见表 1.2），以减少甚至消除寄存器备份存储的要求，这可以极大限度地减少上下文切换的开销。

外部 nFIQ 信号由高电平变为低电平, 产生 FIQ 异常申请。无论异常入口是处于 ARM、Thumb 或 Java 状态，FIQ 的处理过程和返回都要在中断服务程序中完成，返回指令参见表 1.6。

可以在特权模式下将 F（CPSR[6]）标志位置 1，来禁用 FIQ 异常中断申请。当 F=0 时，处理器会在执行完每条指令时，通过 nFIQ 寄存器来检测 nFIQ 信号是否为低电平。

SCR 寄存器中的 FW 和 FIQ 位可以将 FIQ 配置为：

（1）在非安全状态不可屏蔽。

（2）可以分流进入其他异常源 FIQ 或监视模式。

（3）当 FIQ 发生时，FIQs 和 IRQs 申请被禁止。此时可以使用中断嵌套，但需要在处理子程序的入口处保护可能被重复用到的寄存器，并在出口处恢复其原值，重新使能 FIQs 和中断申请。

3．外部中断请求（IRQ）

nIRQ 输入信号由高电平变为低电平，产生 IRQ 异常申请。ARM 状态支持多个外部事件借助 nIRQ 输入信号提出的中断申请。IRQ 的优先级低于 FIQ，在进入 FIQ 异常处理过程中，IRQ 被屏蔽。nIRQ 的处理过程和返回都要在中断服务程序中完成，返回指令参见表 1.6。

可以通过将 CPSR[7]置 1，来禁用 IRQ 异常中断申请。

4．中止

当尝试访问无效的指令时会产生预存指令中止异常，当访问无效的数据存储器时通常会导致一个数据中止异常的发生。

标志位 A（CPSR[8]）为不精确数据中止异常屏蔽位。A=0 时允许异常申请，A=1 时不精确数据中止异常被屏蔽。当不精确数据中止异常发生时，会持续保持不精确数据中止异常挂起的存在，直到屏蔽位被清除，随之不精确数据中止异常事件被处理。A 标志位在处理器进入中止模式、IRQ 和 FIQ 模式以及在复位状态时会被自动置 1。

5．软件中断指令

可以使用软件中断指令（SWI 指令）进入超级用户模式，通常用来请求一个特定的超级用户功能。

6．软件监控指令

使用软件中断指令，可以让内核进入监视模式执行安全监控代码。

7．未定义指令

当遇到处理器或协处理器无法处理的指令，将会进入未定义指令陷阱，触发未定义指令异常事件。软件可以使用这种机制通过仿真未定义的协处理器指令来扩展 ARM 指令集。

8．断点指令

断点指令（BKPT）的操作就如同使用指令产生预取指中止。一个断点指令不会导致处理器指令预取指中止异常，直到指令达到在流水线中的执行阶段。如果此前在流水线上出现一个分支指令，会导致断点指令不被执行。

1.3.6.3 异常事件处理响应机制

当一个异常/中断事件发生时，微处理器会暂停正常操作，自动查找向量表，获得异常/中断事件所对应的中断向量，跳转到异常/中断服务子程序。当服务子程序运行结束后，再返回主程序。

1．响应前期的准备

为了使微处理器能够正确地响应异常/中断事件，前期需要做好准备工作：

（1）开辟有效堆栈区；

（2）编写异常/中断服务子程序，填写异常/中断向量表；

（3）服务子程序入口是否需要参数保护，注意异常/中断服务子程返回类型；

（4）正确了解和设置异常/中断的申请/触发条件。

2．响应异常/中断事件

异常/中断事件发生后，微处理器响应过程如下：

（1）备份寄存器。把 CPSR 和 PC 保存到相应模式下的 SPSR_和 lr_寄存器中。

（2）设置当前模式寄存器组。当 FIQ、IRQ 或外部中止异常发生时，可通过检测安全配置寄存器 SCR[3:1]位的内容来确定所进入的工作模式。设置的寄存器组可参见表 1.2。

（3）读取向量表，设 PC 为相应异常处理程序的入口地址，获得中断向量。

3．执行异常事件服务子程序

依照所编写的服务子程序，完成异常事件的处理过程。

4．异常事件服务子程序的返回

当完成一个异常处理后返回时，异常处理程序需要执行异常返回指令，各种模式下异常入口处保留在 r14 的 PC 值以及退出异常返回前需要执行的指令参见表 1.6。

表 1.6 ARM 异常向量表

异 常 入 口	返 回 指 令	返回前 r14_	描　　述
SVC	MOVS　PC,R14_svc	PC+4	PC：SVC（进入超级用户模式指令），SMC（进入监视模式指令）或 Undefined 指令代码的地址
SMC	MOVS　PC,R14_mon	PC+4	
UNDEF	MOVS　PC,R14_und	PC+4	
PABT	SUBS　PC,R14_abt，#4	PC+4	PC：导致预取指中止的指令地址
FIQ	SUBS　PC,R14_fiq，#4	PC+4	PC：被 FIQ 或 IRQ 异常中断而未被执行的指令地址
IRQ	SUBS　PC,R14_irq，#4	PC+4	
DABT	SUBS　PC,R14_abt，#8	PC+8	PC：导致数据中止的 Load/Store 指令地址
RESET	—	—	复位状态开始执行主程序，不需要返回
BKPT	SUBS　PC,R14_abt，#4	PC+4	软件断点

从异常事件服务子程序返回前完成下面两个操作：

（1）从 spsr_中恢复状态内容到 cpsr 中；

（2）从 lr_中恢复程序断点到 PC 中，从断点处继续执行命令。

1.3.6.4 异常优先级

异常发生时，处理器暂时停止正常程序流来响应异常事件请求。在处理异常之前，处理器会自动保存当前处理器的状态，以便当异常处理程序完成后可以恢复执行原来程序。如果两个或多个异常同时发生，处理器将按照异常事件优先级所固定的顺序来执行异常处理程序，优先级高的异常事件能够中断优先级低的异常事件。表 1.7 中为异常优先级顺序。

表 1.7 异常优先级

优 先 级	异 常 事 件
1（最高）	复位（Reset）
2	精确的数据中止（Precise Data Abort）
3	快速中断请求（FIQ）
4	中断请求（IRQ）
5	预取指中止（Prefetch Abort）
6	不精确的数据中止（Imprecise Data Abort）
7（最低）	断点（BKPT）
	未定义指令（Undefined Instruction）
	软件中断指令（SWI）
	软件监控指令（SMI）

1.3.7 数据类型

Cortex-A8 处理器支持以下数据类型：字（32 位）、半字（16 位）和字节（8 位）。

数值范围：上述 3 种类型的无符号 N 位二进制数值代表一个非负整数，取值范围为 0 至 2^N-1。上述类型的有符号 N 位二进制数值代表一个使用 2 的补码格式表示的有符号整数，取值范围为-2^{N-1}～+2^{N-1}-1。

数据对齐存储：为了获得最佳性能，对字型或半字型数据在内存中开始存放的位置要求边界地址对齐。

字型数据存储位置要求 4 个字节边界对齐（Address[1:0]=b00）。

半字型数据存储位置要求 2 个字节边界对齐（Address[0]=b0）。

字节型数据，存储位置无边界地址要求。

【注意】如果字型数据存储时未按要求 4 个字节边界对齐，不可以使用 LDRD、LDM、LDC、STRD、STM 或 STC 指令来访问内存数据。

1.3.8 存储端模式

内存每个单元存储一个字节，内存单元地址号从零开始按升序排列。一个字（32 位）由 4 个字节组成，存储一个字需要占用 4 个连续存储单元。存放于地址 A 中的字分别存放于地址 A、A+1、A+2、A+3 的 4 个字节中。一个字占用 4 个连续存储单元的存储顺序依照存储端模式（Endianness），可以分为小端模式（Little-endian）和大端模式（Big-endian）两种。

例如，一个字型数据以十六进制格式表示为 0x12345678，该数据由 4 个字节组成，需要占用 4 个连续存储单元。当将该数据存储于 0x00000050 单元时，其数据在内存单元中的存储格式见表 1.8。表中存储单元地址号一栏中为了便于阅读，特将 0x00000050 书写为 0x0000_0050。

表 1.8 存储单元数据存储的端模式

存储单元地址号	小端存储模式（Little-endian）	大端存储模式（Big-endian）
0x0000_0053	0x12	0x78
0x0000_0052	0x34	0x56
0x0000_0051	0x56	0x34
0x0000_0050	0x78	0x12

从表 1.8 可知，小端模式的字型数据存储格式为，数据的低位字节存于低端地址号的存储单元中，高位字节存于高端地址号的存储单元中。

通过 CP15 register1[7]可以设置有效的存储端模式。复位后处理器将该位清零，表示 Cortex-A8 处理器默认工作于小端模式。如果要求处理器工作于大端模式，复位后需要首先执行以下指令完成设置：

```
MRC  p15,0,r0,c1,c0          ;r0=CP15 register1
ORR  r0,r0,#0x80             ;r0[7]=1
MCR  p15,0,r0,c1,c0          ;CP15 register1=r0
```

当系统需要访问硬件接口设备或与外界交互数据时，交互双方的数据存储端模式需要保持一致。通常情况下，可采用系统默认的小端存储模式。

1.4 时钟、复位和电源控制

本节介绍处理器的时钟域和复位输入条件。

1.4.1 时钟域

处理器内部含有 3 个主要时钟域（Clock Domains）：CLK、PCLK 和 ATCLK。可以设置 PCLK 和 ATCLK 与 CLK 同步，也可以设 PCLK 和 ATCLK 之间同步。

1．时钟信号

（1）CLK 是高速内核时钟，用于所有主要处理器接口时钟。L1 存储系统使用时钟的上升沿和下降沿。若所设计逻辑需要使用 CLK 信号下降沿，则其占空比需要为 50%，如图 1.4 所示。

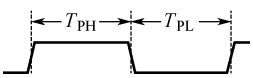

图 1.4 CLK 信号占空比

在处理器内部 CLK 时钟信号控制以下单元：指令取指单元、指令解码单元、指令执行单元、Load/Store 单元、L2 缓存单元（包含 AXI 接口）、NEON 单元、ETM 单元（不含 ATB 接口）和时钟域的调试逻辑。

（2）PCLK 用于 APB 时钟控制处理器的调试接口。PCLK 与 CLK 和 ATCLK 是异步的。PCLK 控制作用域内的调试接口和调试逻辑。

（3）ATCLK 用于 ATB 时钟控制处理器的 ATB 接口。ATCLK 与 CLK 和 ATCLK 是异步的。ATCLK 控制 ATB 接口。

2．AXI 接口时钟的门控信号 ACLKEN

处理器包含一个同步 AXI 接口。AXI 接口时钟（ACLK）可以使用 ACLKEN 作为门控信号，通过 CLK 来产生。AXI 接口时钟频率可以设置为低于内核时钟 CLK 的任意整数倍。借助 ACLKEN 门控信号，利用 CLK 产生 ACLK 的时序关系如图 1.5 所示。

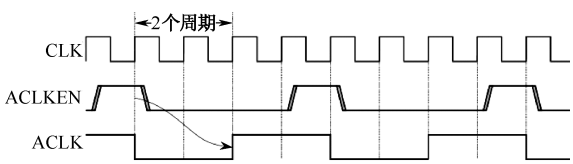

图 1.5 CLK/ACLK 比为 4：1

3．使用 PCLKEN 产生 Debug 时钟

处理器内部的调试逻辑依赖于 PCLK 时钟，PCLK 时钟频率小于或等于 APB 时钟频率。借助 PCLKEN 门控信号，利用 PCLK 产生 Debug 时钟 Internal PCLK 信号的时序关系如图 1.6 所示。

4．使用 ATCLKEN 产生 ATB 时钟

处理器内部的 ATB 逻辑依赖于 ATCLK 时钟，ATCLK 时钟频率小于或等于 ATB 时钟频率。借

助 ATCLKEN 门控信号, 利用 ATCLK 产生 ATB 时钟 Internal ATCLK 信号的时序关系如图 1.7 所示。

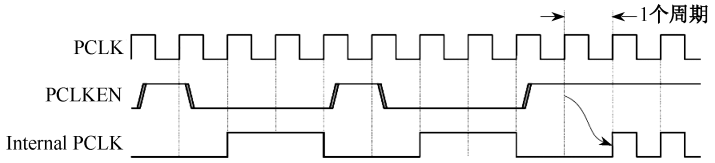

图 1.6　PCLK/Internal PCLK 比为 4：1

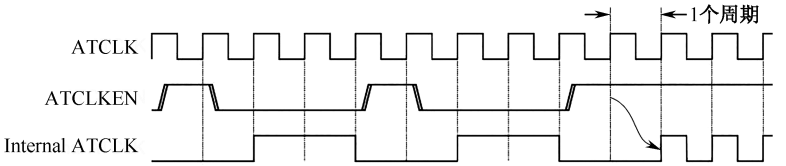

图 1.7　ATCLK/Internal ATCLK 比为 4：1

1.4.2　复位域

处理器的复位域（Reset Domains）支持多个条件域触发复位信号：上电复位、软件复位以及 APB 和 ATB 复位。

所有复位信号低电平有效, 复位过程会影响一个或多个时钟域, 表 1.9 显示了不同复位条件对处理器内部单元的影响。

表 1.9　复位信号作用域

信　　号	Core(CLK)	NEON(CLK)	ETM(CLK)	Debug(CLK)	APB(PCLK)	ATB(ATCLK)
nPORESET	Reset	Reset	Reset	Reset	—	—
ARESETn	Reset	Reset	—	—	—	—
PRESETn	—	—	Reset	Reset	Reset	—
ARESETNEONn	—	Reset	—	—	—	—
ATRESETn	—	—	—	—	—	Reset

1．上电复位

良好的上电复位时序可将系统时钟逻辑设置为正常工作状态。图 1.8 显示了上电复位过程中各信号间的时序关系。

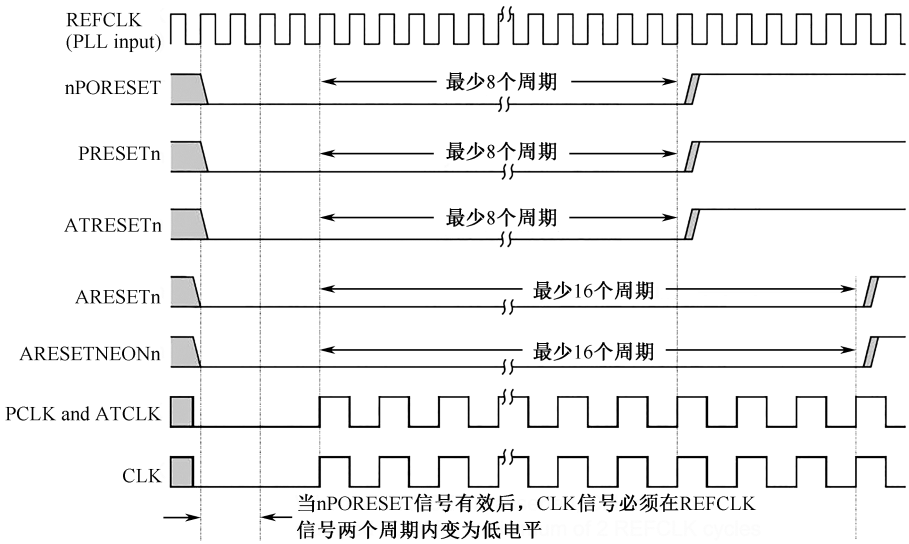

图 1.8　上电复位时序

在图 1.8 所示的上电复位过程中，信号时序关系如下：

（1）时钟（CLK）必须保持 2 个时钟（PLL）周期以上的低电平，用来将处理器内部组件置于安全状态。

（2）nPORESET、PRESETn 和 ATRESETn 复位信号必须保持 8 个时钟周期低电平，确保复位信号已传播到处理器内的正确位置。

（3）ARESETn 和 ARESETNEONn 复位信号在 nPORESET 和 PRESETn 撤销后，必须额外增加 8 个时钟周期低电平，使上述域能够安全退出复位状态。

2．软复位

软复位序列通过复位事件实现使用 ETM 跟踪或调试。通过 ARESETn 和 ARESETNEONn 信号，由 nPORESET 控制复位域实现 ETM 跟踪或调试而不是系统复位，以此在一个软启动序列期间保留断点和观测点信息。图 1.9 显示软复位序列时序。

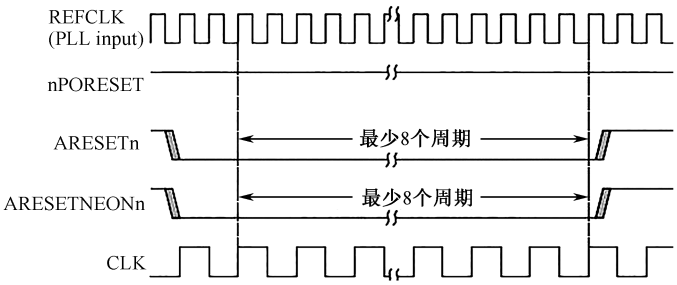

图 1.9　软复位序列时序

处理器提供独立的复位信号（ARESETNEONn）来控制 NEON 单元。ARESETNEONn 必须保持至少 8 个时钟周期，以保证 NEON 单元能够可靠进入复位状态。

3．APB 和 ETM ATB 复位

PRESETn 用于复位处理器内 ETM 时钟域硬件电路。ATRESETn 用来复位 ATB 接口和交叉触发接口（CTI）。为了安全复位，硬件调试单元、ATB 和 CTI 域、PRESETn 和 ATRESETn 信号必须保持 8 个时钟（CLK、PCLK 或 ATCLK 三者中最慢的一个）周期。图 1.10 显示 PRESETn 和 ATRESETn 的时序要求。

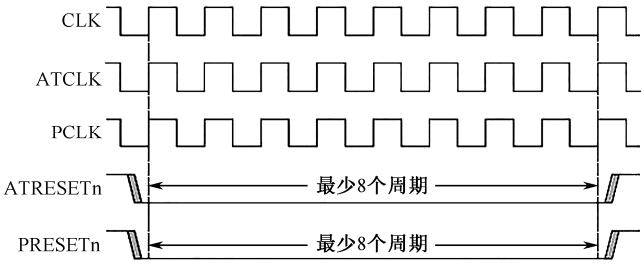

图 1.10　PRESETn 和 ATRESETn 复位时序

4．硬件 RAM 复位

电源复位或软复位时，在默认情况下，处理器将清除 L1 数据缓存和 L2 缓存的有效位。当复位信号解除后，根据 L2 不同容量，这一过程需要占用 1024 个时钟周期。L1 数据和指令缓存复位需要占用 512 个周期，L1 完全复位后，处理器才可以开始工作。L2 硬件复位的发生不影响复位代码。

处理器使用 L1RSTDISABLE 和 L2RSTDISABLE 两个引脚控制硬件复位过程。硬件复位引

脚使用模式如下：

（1）使用 ARESETn 或 nPORESET 信号使 L1 数据缓存和 L2 缓存硬件复位，且 L1RSTDISABLE 和 L2RSTDISABLE 信号必须绑定为低电平。在内核断电序列过程中，应用程序不保留 L1 数据高速缓存和 L2 缓存的内容。

（2）当系统第一次上电时，硬件复位信号 L1RSTDISABLE 和 L2RSTDISABLE 必须绑定为低电平，通过硬件复位机制使 L1 数据缓存和 L2 缓存中的内容无效。如果需要在复位过程保持 L1 和 L2 中的内容，则对应的硬件复位禁止需要保持高电平。

（3）如果硬件阵列复位机制没有使用，L1RSTDISABLE 和 L2RSTDISABLE 引脚电平必须为高电平。在 ARESETn 和 nPORESET 无效后，L1RSTDISABLE 和 L2RSTDISABLE 引脚必须保持最少 16 个时钟周期。

5．存储器阵列复位

处理器复位期间，下面的存储阵列无效：分支预测阵列（BTB 和 GHB）、L1 的指令和数据缓存 TLB。如果 L1RSTDISABLE 绑定为低电平，L1 数据高速缓存为有效内存。如果 L2RSTDISABLE 绑定为低电平，L2 数据高速缓存为有效内存。

1.4.3 电源管理

Cortex-A8 处理器提供了多种方案，实现在内部单元中动态的分级电源管理机制。其中通过对各个时钟作用域的时钟信号使能管理和复位域的复位信号管理，可以对指定单元的供电进行关闭或降功耗等多种方式的管理。

通过对处理器电源管理方案的运用，可有效降低系统功耗。使用方法和应用案例请参考相关手册。

习 题 1

1.1 简述嵌入式微处理器数据存储格式中的大、小端模式。

1.2 Cortex-A8 处理器主要由哪 5 部分组成？

1.3 从编程角度来简述图 1.1 所示 Cortex-A8 内核各组成部分的名称及其作用。

1.4 写出 R 寄存器组中 r13、r14、r15 的名称和作用。

1.5 简述 Cortex-A8 处理器的 8 种工作模式。

1.6 简述 Cortex-A8 处理器的 CPSR 寄存器各功能位名称和功能定义。

1.7 简述 Cortex-A8 处理器可处理的常用异常事件。

1.8 简述使用异常向量表描述异常事件产生条件以及各异常事件处理子程序的入口地址。