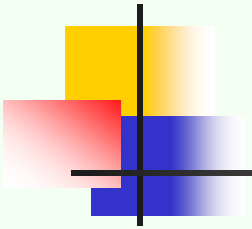
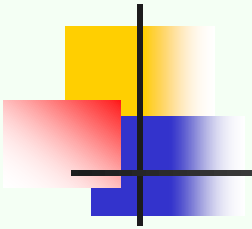
ARM的体系结构与编程



- 计算机体系结构
- 复杂指令集计算机和精简指令集计算机
- **ARM**的发展历程
- **ARM**技术的应用领域及特点
- **ARM**微处理器系列
- **ARM**处理器的工作状态
- **ARM**处理器的工作模式

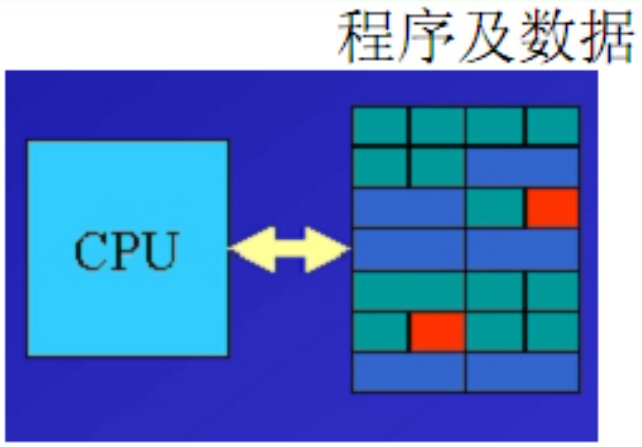


- ARM微处理器的存储器格式
- ARM 处理器的寄存器格式
- ARM 异常处理
- ARM 微处理器指令的分类
- ARM 微处理器指令的格式
- ARM 微处理器指令的条件域
- ARM 微处理器指令的寻址方式

计算机体系结构

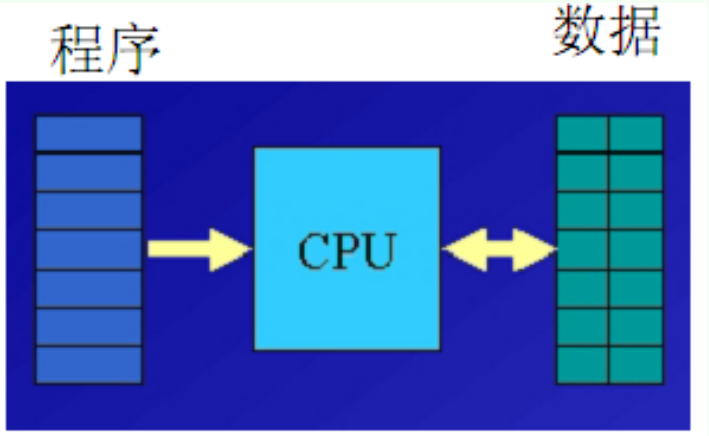
- 冯.诺依曼结构（von Neumann architecture）
  - 五大部件：ALU，Controller，Memory，Input，Output
  - 程序指令和数据放在同一存储器的不同物理位置，CPU通过同一条总线访问程序和数据，程序指令和数据的宽度是相同的
  - 存储程序(stored program)：程序以数字形式存在，可以与数据一样被读写

约翰·冯·诺依曼（**John von Neumann，1903－1957**），美籍匈牙利人“现代电子计算机之父”提出**2**进制思想与程序内存思想



计算机体系结构

- 哈佛体系结构（Harvard architecture）
  - 程序与数据有单独的存储器，它们通过不同的总线来访问
  - 首先到程序指令存储器中读取程序指令，解码后得到数据地址，再到相应的数据存储器中读取数据，并执行。在执行一条指令的同时可以预先读取下一条指令
  - 指令和数据可以有不同的数据宽度（如PIC的程序指令是14位，而数据是8位）



复杂指令集计算机和精简指令集计算机

复杂指令集计算机CISC

（Complex Instruction Set Computer）

- 背景：
  - 存储资源紧缺，强调编译优化
  - 增强指令功能，设置一些功能复杂的指令以减少完成一个任务所需要的指令数目
  - 通过减少指令数达到提高运行速度的目的

复杂指令集计算机和精简指令集计算机

复杂指令集计算机CISC

- 特点：
  - 指令格式不固定，指令可长可短，操作数可多可少
  - 寻址方式复杂多样，操作数可来自寄存器，也可来自存储器
  - 使用微代码。指令集可以直接在微代码记忆体（比主记忆体的速度快很多）里执行
  - 允许设计师实现CISC体系机器的向上相容。新的系统可以使用一个包含早期系统的指令超集合
  - 微程式指令的格式与高阶语言相匹配，因而编译器的设计较简单
  - CPI > 5，指令越复杂，CPI越大

CPI—执行每条指令所需的平均时钟周期数

复杂指令集计算机和精简指令集计算机

- **复杂指令集计算机CISC**

缺陷：

- 指令使用频度不均衡
- 大量复杂指令的控制逻辑不规整，不适于VLSI工艺
- CISC指令的格式长短不一，需要不同的时钟周期来完成。

典型指令使用频度

指令类型	使用指令使用频度
数据传送类	43%
转/跳控制类	23%
算术运算类	15%
比较类	13%
逻辑运算类	5%
其他	1%

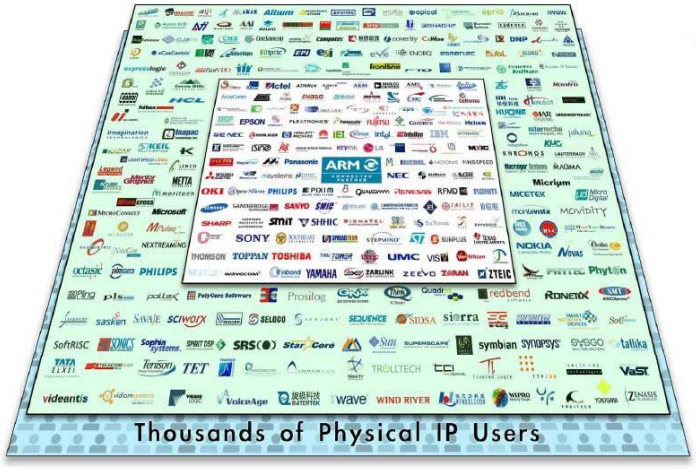
复杂指令集计算机和精简指令集计算机

- **精简指令集计算机RISC**

（reduced instruction set computer）

- 精简指令集:保留最基本的,去掉复杂、使用频度不高的指令
- 复杂指令可通过对简单基本的指令组合而成
- 每条指令的长度都是相同的，大部分指令可以在一个机器周期里完成
- 采用多级指令流水线结构,处理器在同一时间内可执行多条指令
- 采用加载(Load)、存储(Store)结构，统一存储器访问方式，只允许Load和Store指令执行存储器操作，其余指令均对寄存器操作。
- 大大增加通用寄存器的数量，ALU只与寄存器文件直接连接。
- 采用高速缓存(cache)结构

ARM的发展历程



- ARM—Advanced RISC Machines

❑ **ARM（Advanced RISC Machines），既可以认为是一个公司的名字，也可以认为是对一类微处理器的通称，还可以认为是一种技术的名字。**

❑ **1990年ARM公司成立于英国剑桥，主要出售芯片设计技术的授权。**

ARM的发展历程

- ARM = Advanced RISC Machines
  - 1985年4月26日，第一个ARM原型在英国剑桥的Acorn计算机有限公司诞生，由美国加州San Jose VLSI技术公司制造。
  - 20世纪80年代后期，ARM很快开发成Acorn的台式机产品，形成英国的计算机教育基础。
- 1990年成立ARM公司
- 20世纪90年代，ARM32位嵌入式处理器占据了低功耗、低成本和高性能的嵌入式系统应用领域的领先地位。
  - 1999年因移动电话火爆市场，其32位RISC处理器占市场份额超过了50%
  - 2001年初，ARM公司的32位RISC处理器市场占有率超过了75%。
- ARM公司是知识产权供应商，是设计公司。由合作伙伴公司来生产各具特色的芯片。

ARM的发展历程


ATAP Partners
Tools Partners
RTOS Partners
Software Partners
ARM
in Partnership
WIN-FINITY
BARCO SILEX
SOTA
DNP
Infinite Technology Corporation
SIEMENS
NSW
STEPMIND
Think
MacroTech Research, Inc.
COMIT SYSTEMS
YOGITECH
ARCADIA
SIDSA
TOPPAN
WIPRO
SEODU INCHIF
ZAIQ
HOYA
INICORE
MAZ
SCIworx
FARADAY
nordic
TALITY
parthus
SYNOPSYS
FLEXTRONICS
Semiconductor
QUALCOMM
ZTEIC
GOODRICH
MITSUBISHI ELECTRIC
FUJITSU
ERICSSON
OKI
MOTOROLA
ADMtek
NEC
NEOMAGIC
TOSHIBA
intersil
AMI
MICRONAS
PHILIPS
UMC
EPSON
Agilent Technologies
Chartered
SANYO
ROHM
ANALOG DEVICES
ALCATEL
CIRRUS LOGIC
TEXAS INSTRUMENTS
AKM
agere systems
Panasonic
3Com
HYUNDAI
BROADCOM
SHARP
ZEEVO
intel
YAMAHA
OAK
IBM
Virata
SONY
Triscend.
ALTERA
cogency
PRAIRIECOMM
Infineon
ZARLINK
ArrayComm
KAWASAKI MICRO
Silicon Wave
RESONEXT
National Semiconductor
EPI
ASHLING
CoWare
virtio
Green Hills
INNOVEDA.
Computex
YOKOGAWA
ADS
Tektronix
WindRiver
Sophia systems
Axis systems
Verisity
Aptix
Agilent Technologies
COGENT
Aliant
ALCATEL
IKOS SYSTEMS
SYNOPSYS
LAUTERBACH
cadence
FIRMWARE SYSTEMS
realogy
eSOL
jmi
QNX
LINEO
ACCESS
PRECISE
GEOWORKS
US Software
Tao Systems
KADAK Products Ltd.
Microsoft
JAVA
OSE systems
AXE
Eonic
symbian
FirmWorks
MICROWARE
Sun
ETNOTEAM
WindRiver
Embedded System Products
LYNUXWORKS
CMX COMPANY
interniche technologies, inc.
Microsoft
EMBLAZE Cellular
FRONTIER
Packet Video
INTERTRUST
ERICSSON
ZI corporation
liquid audio
Bluetooth
symbian
QNX
Dolby
Symmetricom
CPS
JAVA
ASAHI CHEMICAL INDUSTRY CO.,LTD.

ARM技术的应用领域及特点

ARM 公司的Chip less模式

- 世界各大半导体生产商从ARM公司购买其设计的ARM微处理器IP核（软核、硬核），根据各自不同的应用领域，加入适当的外围电路，从而形成自己的ARM微处理器芯片进入市场。
- 基于ARM技术的微处理器应用约占据了32位RISC微处理器75％以上的市场份额，ARM技术正在逐步渗入到我们生活的各个方面。
- 我国的华为、中兴、大唐电讯、中芯国际和上海华虹，以及国外的一些公司如德州仪器、意法半导体、Philips、Intel、Samsung等都推出了自己设计的基于ARM核的处理器。

ARM技术的应用领域及特点

ARM微处理器的应用领域

- 到目前为止，ARM微处理器及技术的应用已经广泛深入到国民经济的各个领域
  - 工业控制领域：作为32位的RISC架构，ARM微控制器的低功耗、高性价比，向传统的8位/16位微控制器提出了挑战。

ARM技术的应用领域及特点

ARM微处理器的应用领域

- 网络应用：采用ARM技术的ADSL芯片，ARM在语音及视频处理上进行了优化，获得广泛支持。
- 消费类电子产品：ARM技术在目前流行的数字音频播放器、数字机顶盒和游戏机中得到广泛采用。
- 成像和安全产品：现在流行的数码相机和打印机中绝大部分采用ARM技术，手机中的32位SIM智能卡也采用了ARM技术。
- ……

ARM技术的应用领域及特点

ARM微处理器的特点—低功耗、低成本、高性能

- 低功耗、低成本、高性能
  - 采用RISC指令集
  - 使用大量的寄存器
  - ARM/THUMB指令支持
  - 三/五级流水线

ARM技术的应用领域及特点

ARM微处理器的特点—采用RISC体系结构

- 采用RISC架构的ARM处理器一般具有如下特点：
  - 固定长度的指令格式，指令归整、简单、基本寻址方式有2～3种；
  - 使用单周期指令，便于流水线操作执行；
  - 大量使用寄存器，数据处理指令只对寄存器进行操作，只有加载/存储指令可以访问存储器，以提高指令的执行效率。

ARM技术的应用领域及特点

ARM微处理器的特点—大量使用寄存器

- ARM 处理器共有37个寄存器，被分为若干个组，这些寄存器包括：
  - 31个通用寄存器，包括程序计数器（PC 指针），均为32位的寄存器；
  - 6个状态寄存器，用以标识CPU的工作状态及程序的运行状态，均为32位。

ARM技术的应用领域及特点

ARM微处理器的特点—高效的指令系统

- ARM微处理器支持两种指令集：ARM指令集和Thumb指令集。

- ARM指令为32位的长度，Thumb指令为16位长度。Thumb指令集为ARM指令集的功能子集，但与等价的ARM代码相比较，可节省30％～40％以上的存储空间，同时具备32位代码的所有优点。

ARM技术的应用领域及特点

ARM微处理器的特点—其他技术

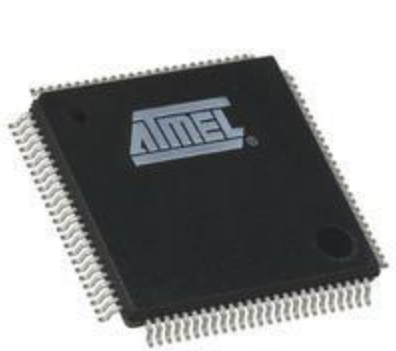
- 除此以外，ARM体系结构还采用了一些特别的技术，在保证高性能的前提下尽量缩小芯片的面积，并降低功耗：
- 所有的指令都可根据前面的执行结果决定是否被执行，从而提高指令的执行效率。
- 可用加载/存储指令批量传输数据，以提高数据的传输效率。
- 可在一条数据处理指令中同时完成逻辑处理和移位处理。
- 在循环处理中使用地址的自动增减来提高运行效率。

ARM微处理器系列

- ARM7系列
- ARM9系列
- ARM9E系列
- ARM10E系列
- ARM11系列
- SecurCore系列
- Intel的Xscale
- ARM Cortex系列
- 其中，ARM7、ARM9、ARM9E和ARM10、ARM11为通用处理器系列，每一个系列提供一套相对独特的性能来满足不同应用领域的需求。SecurCore系列专门为安全要求较高的应用而设计。

ARM微处理器系列

ARM7微处理器系列



Atmel公司生产的ARM7芯片

- ARM7系列是为低功耗的32位RISC处理器，最适合用于对价位和功耗要求较高的消费类应用。ARM7系列有如下特点：
  - 具有嵌入式ICE－RT逻辑，调试开发方便；
  - 极低的功耗，适合对功耗要求较高的应用，如便携式产品；
  - 能够提供0.9MIPS/MHz的三级流水线冯.诺伊曼结构；
  - 代码密度高，并兼容16位的Thumb指令集；
  - 对操作系统的支持广泛，如Windows CE、Linux、Palm OS等；
  - 指令系统与ARM9系列、ARM9E系列和ARM10E系列兼容；
  - 主频最高可达130M，高速的运算处理能力能胜任绝大多数的复杂应用。

ARM微处理器系列

ARM7微处理器系列

- 主要应用领域：工业控制、Internet设备、网络和调制解调器设备、移动电话等多种多媒体和嵌入式应用。

- ARM7系列微处理器包括如下几种类型的核：ARM7TDMI、ARM7TDMI-S、ARM720T、ARM7EJ。其中，ARM7TMDI是使用广泛的32位嵌入式RISC处理器，属低端ARM处理器核。TDMI的基本含义为：

  T：支持16位压缩指令集Thumb；

  D：支持片上Debug；

  M：内嵌硬件乘法器（Multiplier）

  I：嵌入式ICE，支持片上断点和调试点；

ARM微处理器系列

ARM9微处理器系列

SAMSUNG S3C2440

- ARM9系列微处理器在高性能和低功耗特性方面提供最佳的表现。具有以下特点：
  - 5级整数流水线，指令执行效率更高。
  - 提供1.1MIPS/MHz的哈佛结构。
  - 支持32位ARM指令集和16位Thumb指令集。
  - 支持32位的高速AMBA总线接口。
  - 全性能的MMU，支持Windows CE、Linux、Palm OS等多种嵌入式操作系统。
  - MPU支持实时操作系统。
  - 支持数据Cache和指令Cache，具有更高的指令和数据处理能力。

ARM微处理器系列

ARM9微处理器系列

- ARM9系列微处理器主要应用于无线设备、仪器仪表、安全系统、机顶盒、高端打印机、数字照相机和数字摄像机等。

- ARM9系列微处理器包含ARM920T、ARM922T和ARM940T三种类型，以适用于不同的应用场合。

ARM微处理器系列

ARM9E微处理器系列

内嵌arm9e的cpu核

- ARM9E系列微处理器的主要特点如下：
  - 支持DSP指令集，适合于需要高速数字信号处理的场合。
  - 5级整数流水线，指令执行效率更高。
  - 支持32位ARM指令集和16位Thumb指令集。
  - 支持32位的高速AMBA总线接口。
  - 支持VFP9浮点处理协处理器。
  - 全性能的MMU，支持众多嵌入式操作系统。
  - 支持数据Cache和指令Cache，具有更高的处理能力。
  - 主频最高可达300M。

ARM微处理器系列

ARM9E微处理器系列

- ARM9E系列微处理器主要应用于下一代无线设备、数字消费品、成像设备、工业控制、存储设备和网络设备等领域。
- ARM9E系列微处理器包含ARM926EJ-S、ARM946E-S和ARM966E-S三种类型，以适用于不同的应用场合。

ARM微处理器系列

ARM10E微处理器系列

- ARM10E系列微处理器的主要特点如下：
  - 支持DSP指令集，适合于需要高速数字信号处理的场合。
  - 6级整数流水线，指令执行效率更高。
  - 支持32位ARM指令集和16位Thumb指令集。
  - 支持32位的高速AMBA总线接口。
  - 支持VFP10浮点处理协处理器。
  - 全性能的MMU，支持众多嵌入式操作系统。
  - 支持数据Cache和指令Cache，具有更高的处理能力
  - 主频最高可达400M。
  - 内嵌并行读/写操作部件。

ARM微处理器系列

ARM10E微处理器系列

- ARM10E系列微处理器主要应用于下一代无线设备、数字消费品、成像设备、工业控制、通信和信息系统等领域。

- ARM10E系列微处理器包含ARM1020E、ARM1022E和ARM1026EJ-S三种类型，以适用于不同的应用场合。

ARM微处理器系列

Samsung S5P6440 ARM11

ARM11微处理器系列

- **ARM11**系列微处理器是**ARM**公司**2003**年推出的新一代**RISC**处理器，它是**ARM**新指令架构——**ARMv6**的第一代设计实现。该系列主要有**ARM1136J**，**ARM1156T2**和**ARM1176JZ**三个内核型号，分别针对不同应用领域。

- **ARMv6**架构是根据下一代的消费类电子、无线设备、网络应用和汽车电子产品等需求而制定的。**ARM11**的媒体处理能力和低功耗特点，特别适用于无线和消费类电子产品；其高数据吞吐量和高性能的结合非常适合网络处理应用；另外，在实时性能和浮点处理等方面**ARM11**可以满足汽车电子应用的需求。

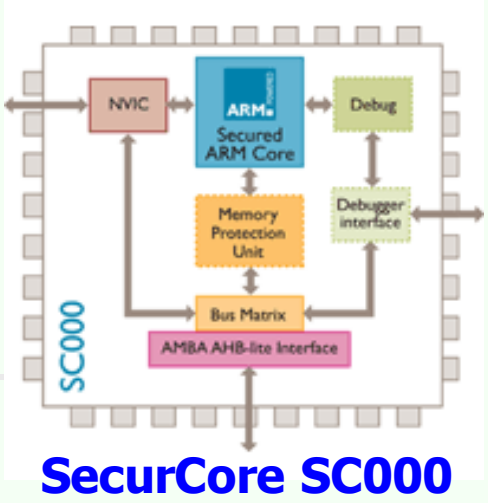
ARM微处理器系列

ARM11微处理器系列

- ARM11系列微处理器的主要特点如下：
  - **ARM11**处理器由**8**级流水线组成，比以前的**ARM**内核提高了至少**40%**的吞吐量，可以使**8**条指令同时被执行。
  - **ARM11**处理器以**32**位处理器的成本，提供**64**位处理器性能的解决方案。内核和**Cache**，及协处理器之间的数据通路是**64**位的。可以每周期读入两条指令或存放两个连续的数据，大大提高了数据访问和处理的速度。经过评测，这已经和**64**位处理器的性能相差无几。
  - **ARM11**处理器将浮点运算当成一个可供用户选择的设计，用户可以在向**ARM**要求授权的时候选择是否包括浮点处理器的内核。
  - **ARM11**系列处理器展示了在性能上的巨大提升，首先推出**350M~500MHz**频率的内核，以后也可以上升到**1GHz**频率。

ARM微处理器系列

SecurCore微处理器系列

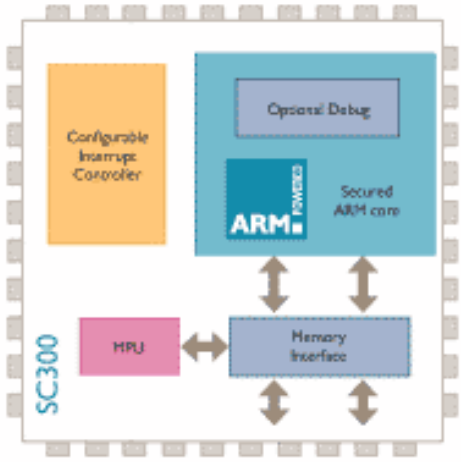


SecurCore SC000

- SecurCore系列微处理器除了具有ARM体系结构各种主要特点外，还在系统安全方面具有如下的特点：
  - 带有灵活的保护单元，确保操作系统和应用数据的安全。
  - 采用软内核技术，防止外部对其进行扫描探测。
  - 可集成用户自己的安全特性和其他协处理器。

ARM微处理器系列

SecurCore微处理器系列



SecurCore SC300处理器

- SecurCore系列微处理器主要应用于一些对安全性要求较高的应用产品及应用系统，如电子商务、电子政务、电子银行业务、网络和认证系统等领域。
- SecurCore系列微处理器包含SecurCore SC100、SecurCore SC110、SecurCore SC200和SecurCore SC300等类型，以适用于不同的应用场合。

ARM微处理器系列

（StrongARM SA-110 微处理器）

StrongARM微处理器系列

- 早期Intel StrongARM SA-1100处理器是采用ARM体系结构高度集成的32位RISC微处理器。它融合了Intel公司的设计和处理技术以及ARM体系结构的电源效率，采用在软件上兼容ARMv4体系结构、同时采用具有Intel技术优点的体系结构。
- **Intel StrongARM**处理器是便携式通讯产品和消费类电子产品的理想选择，已成功应用于多家公司的掌上电脑系列产品。

ARM微处理器系列

Xscale处理器

Intel PXA27x Processor

- Xscale 处理器是基于ARMv5TE体系结构的解决方案，是一款全性能、高性价比、低功耗的处理器。它支持16位的Thumb指令和DSP指令集，已使用在数字移动电话、个人数字助理和网络产品等场合。
- Xscale是Marvell公司（之前是Intel）针对嵌入式产品的核心，是ARM架构v5TE指令集的CPU。2006年6月，Intel将其通信及应用处理器业务出售给Marvell公司。

Toradex Colibri
- XScale Monahans PXA320

ARM微处理器系列

ARM Cortex-M3

ARM Cortex处理器系列

- 包括了**ARMv7**架构的所有系列，含有面向复杂操作系统、实时的和微控制器应用的多种处理器。**ARM Cortex-M**系列支持**Thumb-2**指令集，它是**Thumb**指令集的扩展集，可以执行所有已存的为早期的处理器编写的代码。
- 在命名方式上，基于**ARMv7**架构的**ARM**处理器已经不再延用过去的数字命名方式，而是冠以**Cortex**的代号。基于**v7A**的称为"**Cortex-A系列**"，基于**v7R**的称为"**Cortex-R系列**"，基于**v7M**的称为"**Cortex-M3**"。
- **ARM Cortex-A**系列是针对日益增长的，运行包括**IOS**、**Android**、**Linux**、**Windows CE**和**Symbian**等操作系统的消费者娱乐和无线产品设计的；**ARM Cortex-R**系列针对的是需要运行实时操作系统来进行控制应用的系统，包括有汽车电子、网络和影像系统；**ARM Cortex-M**系列则是为那些对开发费用非常敏感同时对性能要求不断增加的嵌入式应用所设计的，如意法半导体的**stm32**系列。

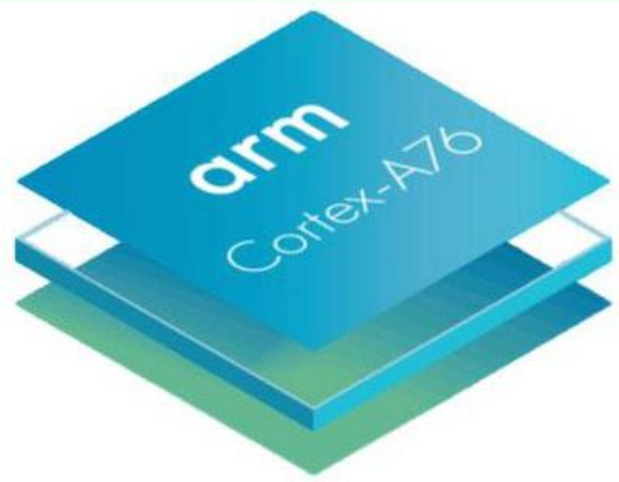
ARM微处理器系列

（Cortex A8核心处理器的苹果iPad）

ARM Cortex处理器系列

- ARM Cortex A8移动芯片工作频率600M～1GHz以上，支持低耗电技术，单核处理器解决方案。
- ARM Cortex-A8处理器是针对高端市场（Iphone4、IPAD），而ARM11针对的是中低端市场，在不到ARM11一半功耗的情况下可提供比基于ARM11处理器最多达到三倍的性能增益 。
- 全新的Cortex-A9处理器让手机、平板电脑这些便携式移动设备迈入了双核CPU的纪元。

ARM微处理器系列



ARM Cortex A76处理器

- Arm在2018年发表最新的Cortex-A76处理器具有13级流水线，以强化性能、机器学习以及省电为目标，同时也能满足移动装置对芯片尺寸的严格限制。
- Cortex-A76支持双指令管线原生16B向量与浮点运算单元，在机器学习领域能创造4倍性能提升，并提升40%能源效率。
- Windows笔记本电脑充分展现了最新Arm处理器比x86架构处理器省电的特性，在使用10.5小时之后，仍能保有35%的剩余电力。Arm处理器相当适合应用于省电、连网的笔记本电脑，这或许意味着在未来我们可以看到更多类似Asus NovaGo等搭载Arm处理器的Windows笔记本电脑产品。
- ARM一直强调新CPU的笔记本级性能，架构师Mike Filippo表示， Cortex A76相当于i5-7300，如果IP厂商缓存设计得更好，那么可以媲美i7。演示中3.3GHz的A76功耗超过了5W，这对于手机来说虽然是不可接受的，但是笔记本倒还好。

ARM处理器的工作状态

- 从编程的角度看，ARM微处理器的工作状态一般有两种，并可在两种状态之间切换：
  - **ARM状态**，此时处理器执行**32位**的字对齐的**ARM**指令；
  - **Thumb状态**，此时处理器执行**16位**的、半字对齐的**Thumb**指令。

ARM处理器的工作状态

ARM与THUMB

- THUMB指令是ARM指令的子集
- 可以相互转换，只要遵循一定的转换规则
- Thumb指令与ARM指令的时间效率和空间效率关系为:
  - 存储空间约为ARM代码的60%～70%
  - 指令数比ARM代码多约30%～40%
  - 存储器为32位时ARM代码比Thumb代码快约40%
  - 存储器为16位时Thumb比ARM代码快约40～50%
  - 使用Thumb代码，存储器的功耗会降低约30%

ARM处理器的工作状态

状态切换方法

- ARM指令集和Thumb指令集均有切换处理器状态的指令，并可在两种工作状态之间切换，
- 在开始执行代码时，应该处于ARM状态。

ARM处理器的工作状态

进入Thumb状态

- 当操作数寄存器的状态位（位0）为1时，可以采用执行BX指令的方法，使微处理器从ARM状态切换到Thumb状态。
- 当处理器处于Thumb状态时发生异常（如IRQ、FIQ、Undef、Abort、SWI等），则异常处理返回时，自动切换到Thumb状态。

ARM处理器的工作状态

切换到ARM状态

- 当操作数寄存器的状态位为0时，执行BX指令时可以使微处理器从Thumb状态切换到ARM状态。
- 在处理器进行异常处理时，把PC指针放入异常模式链接寄存器中，并从异常向量地址开始执行程序，也可以使处理器切换到ARM状态。

ARM处理器的工作模式

- usr：ARM处理器正常的程序执行状态
- fiq：用于高速数据传输或通道处理
- irq：用于通用的中断处理
- svc：操作系统使用的保护模式
- abt：用于虚拟存储及存储保护
- und：当出现未定义指令终止时进入该模式
- sys：运行具有特权的操作系统任务

ARM处理器的工作模式

用户模式和特权模式

- 除了用户模式之外的其他6种处理器模式称为特权模式
- 特权模式下，程序可以访问所有的系统资源，也可以任意地进行处理器模式的切换。
- 特权模式中，除系统模式外，其他5种模式又称为异常模式
- 大多数的用户程序运行在用户模式下，此时，应用程序不能够访问一些受操作系统保护的系统资源，应用程序也不能直接进行处理器模式的切换。
- 用户模式下，当需要进行处理器模式切换时，应用程序可以产生异常处理，在异常处理中进行处理器模式的切换。

ARM处理器的工作模式

模式切换

- 处理器模式可以通过软件进行切换，也可以通过外部中断或者异常处理过程进行切换。
- 当应用程序发生异常中断时，处理器进入相应的异常模式。在每一种异常模式下都有一组寄存器，可以保证在进入异常模式时，用户模式下的寄存器不被破坏。
- 系统模式并不是通过异常进入的，它和用户模式具有完全一样的寄存器。但是系统模式属于特权模式，可以访问所有的系统资源，也可以直接进行处理器模式切换，它主要供操作系统任务使用。该任务仍然使用用户模式的寄存器组，而不是使用异常模式下相应的寄存器组，这样可以保证当异常中断发生时任务状态不被破坏。

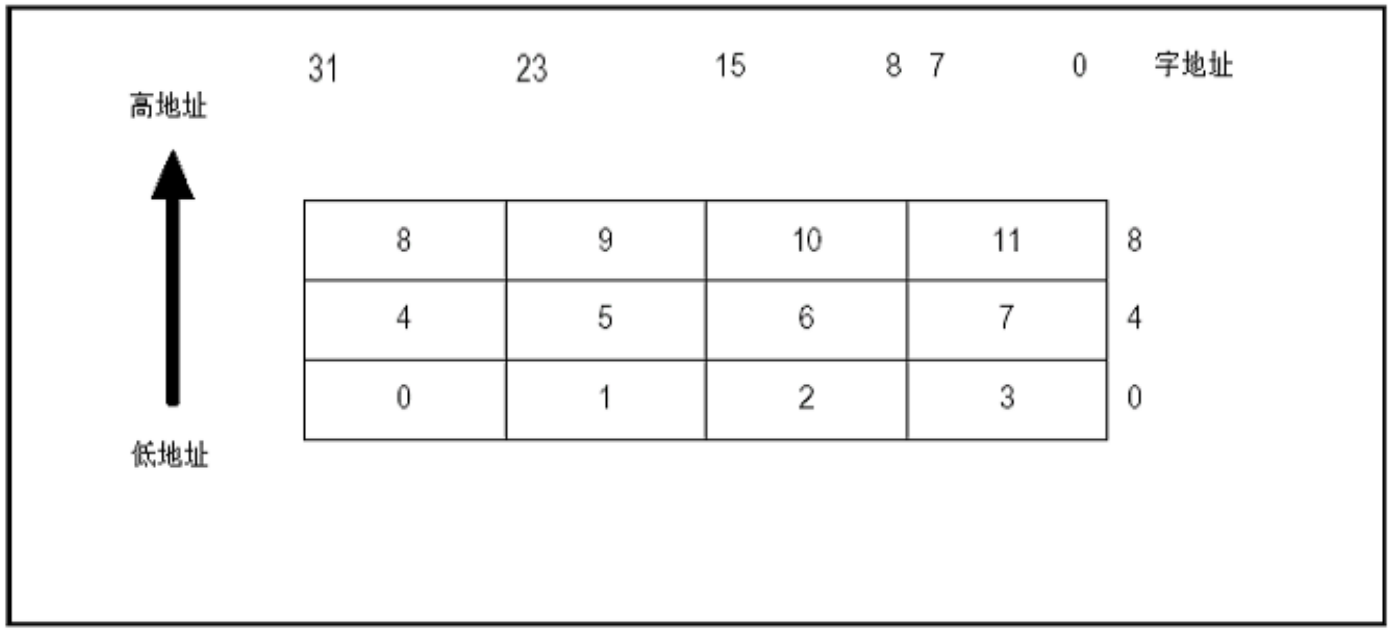
ARM微处理器的存储器格式

- ARM32位体系结构所支持的最大寻址空间为4GB（2^{32}字节）
- ARM体系结构将存储器看作是从零地址开始的字节的线性组合。从零字节到三字节放置第一个存储的字数据，从第四个字节到第七个字节放置第二个存储的字数据，依次排列。
- ARM体系结构可以用两种方法存储字数据，称之为大端格式和小端格式

ARM微处理器的存储器格式

ARM体系结构的存储器格式—大端格式

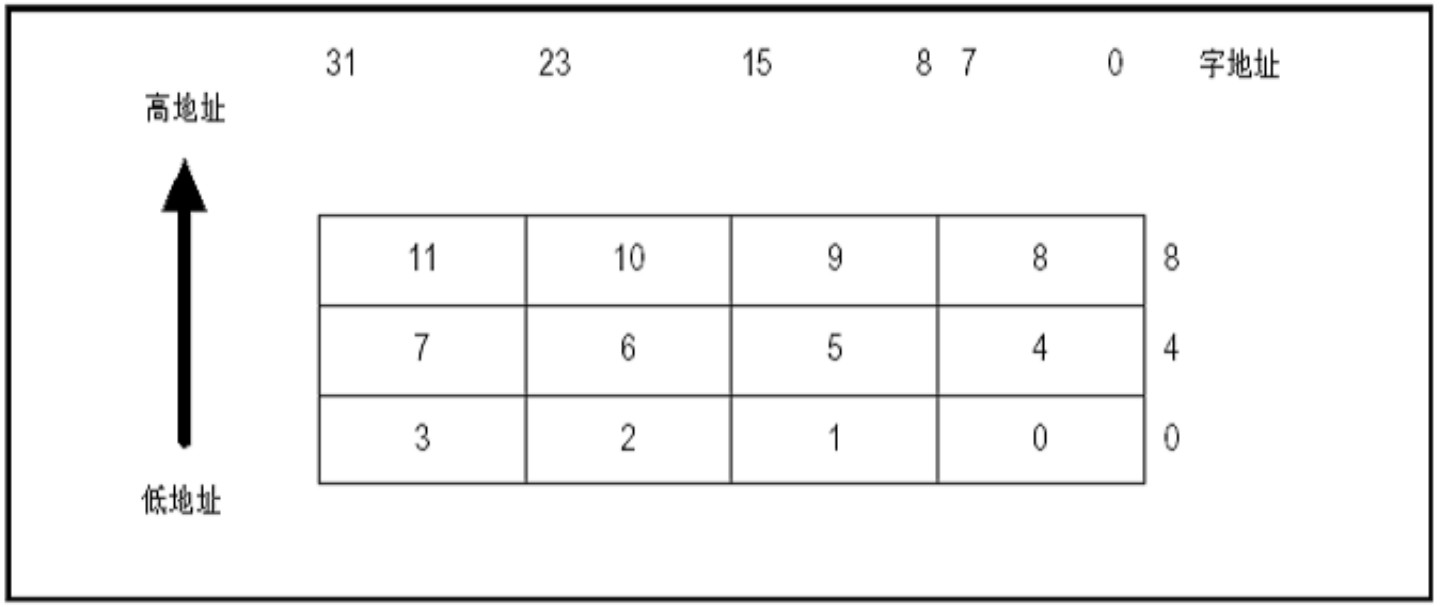
- 在这种格式中，字数据的高字节存储在低地址中，而字数据的低字节则存放在高地址中



ARM微处理器的存储器格式

ARM体系结构的存储器格式—小端格式

- 与大端存储格式相反，在小端存储格式中，低地址中存放的是字数据的低字节，高地址存放的是字数据的高字节



ARM处理器的存储器格式

指令长度及数据类型

- ARM微处理器的指令长度可以是32位（在ARM状态下），也可以为16位（在Thumb状态下）。
- ARM微处理器中支持字节（8位）、半字（16位）、字（32位）三种数据类型，其中，字需要4字节对齐（地址的低两位为0）、半字需要2字节对齐（地址的最低位为0），单字节的没有这个问题，就不用考虑了。

ARM处理器的存储器格式

非对齐的存储访问操作

- 在ARM中，如果存储单元的地址没有遵守对齐规则，则称为非对齐的存储访问操作。
  - 非对齐的指令预取操作
  - 非对齐的数据访问操作

ARM处理器的存储器格式

非对齐的指令预取操作

- 当处理器处于ARM状态期间，如果写入到寄存器PC中的值是非字对齐的(低两位不为0b00)，要么指令执行的结果不可预知，要么地址值中最低两位被忽略。
- 当处理器处于THUMB状态期间，如果写入到寄存器PC中的值是非半字对齐的(最低位不为0b0)，要么指令执行的结果不可预知，要么地址值中最低位被忽略。

ARM处理器的存储器格式

非对齐的数据访问操作

- 对于Load/Store操作，如果是非对齐的数据访问操作，系统定义了三种可能的结果：
  - 执行的结果不可预知
  - 忽略字单元地址的低两位的值，即访问地址为(address AND 0XFFFFFFFC)的字单元；忽略半字单元地址的最低位的值，即访问地址为(address AND 0XFFFFFFFE)的半字单元。
  - 忽略字单元地址的低两位的值；忽略半字单元地址的最低位的值；由存储系统实现这种忽略。也就是说，这时该地址值原封不动地送到存储系统。
- 当发生非对齐的数据访问时，到底采用上述三种方法中的哪一种，是由各指令指定的。

ARM 处理器的寄存器格式

寄存器组织

- **ARM**微处理器共有**37**个**32**位寄存器，其中**31**个为通用寄存器，**6**个为状态寄存器。但是这些寄存器不能被同时访问，具体哪些寄存器是可编程访问的，取决微处理器的工作状态及具体的运行模式。但在任何时候，通用寄存器**R14～R0**、程序计数器**PC**、一个或两个状态寄存器都是可访问的。

ARM 处理器的寄存器格式

ARM状态下的寄存器组织

- 通用寄存器：通用寄存器包括R0～R15，可以分为三类：
  - 未分组寄存器R0～R7
  - 分组寄存器R8～R14
  - 程序计数器PC(R15)

ARM 处理器的寄存器格式

ARM状态下的寄存器组织

ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
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)

ARM 处理器的寄存器格式

未分组寄存器R0～R7

- 在所有的运行模式下，未分组寄存器都指向同一个物理寄存器，他们未被系统用作特殊的用途，因此，在中断或异常处理进行运行模式转换时，由于不同的处理器运行模式均使用相同的物理寄存器，可能会造成寄存器中数据的破坏，这一点在进行程序设计时应引起注意。

ARM 处理器的寄存器格式

分组寄存器R8～R12

- 每次所访问的物理寄存器与处理器当前的运行模式有关
- R8～R12：每个寄存器对应两个不同的物理寄存器
  - 当使用fiq模式时，访问寄存器R8_fiq～R12_fiq；
  - 当使用除fiq模式以外的其他模式时，访问寄存器R8_usr～R12_usr。

ARM 处理器的寄存器格式

分组寄存器R13～R14

- R13、R14：每个寄存器对应6个不同的物理寄存器
- 其中的一个是用户模式与系统模式共用，另外5个物理寄存器对应于其他5种不同的运行模式
- 采用以下的记号来区分不同的物理寄存器：
  - **R13_<mode>**
  - **R14_<mode>**
- mode为以下几种之一：usr、fiq、irq、svc、abt、und。

ARM 处理器的寄存器格式

堆栈指针—R13

- R13在ARM指令中常用作堆栈指针，但这只是一种习惯用法，用户也可使用其他的寄存器作为堆栈指针。
- 在Thumb指令集中，某些指令强制性的要求使用R13作为堆栈指针。
- 由于处理器的每种运行模式均有自己独立的物理寄存器R13，在初始化部分，都要初始化每种模式下的R13，这样，当程序的运行进入异常模式时，可以将需要保护的寄存器放入R13所指向的堆栈，而当程序从异常模式返回时，则从对应的堆栈中恢复。

ARM 处理器的寄存器格式

子程序连接寄存器（LR）—R14

- R14也称作子程序连接寄存器或连接寄存器LR。当执行BL子程序调用指令时，可以从R14中得到R15（程序计数器PC）的备份。其他情况下，R14用作通用寄存器。
- 在每一种运行模式下，都可用R14保存子程序的返回地址，当用BL或BLX指令调用子程序时，将PC的当前值拷贝给R14，执行完子程序后，又将R14的值拷贝回PC，即可完成子程序的调用返回。

```
        BL   SUB1
                ……
SUB1: STMFD SP! ,{<regs>, LR}     /*将regs、R14存入堆栈*/
                ……
        LDMFD SP! ,{<regs>,PC}   /*完成子程序返回 */
```

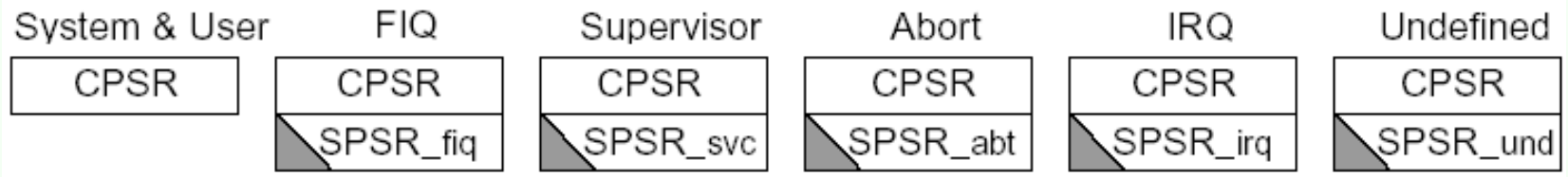
ARM 处理器的寄存器格式

程序计数器PC(R15)

- ARM状态下，位[1:0]为0，位[31:2]用于保存PC；
- Thumb状态下，位[0]为0，位[31:1]用于保存PC；
- R15虽然也可用作通用寄存器，但一般不这么使用，因为对R15的使用有一些特殊的限制，当违反了这些限制时，程序的执行结果是未知的。
- 由于ARM体系结构采用了多级流水线技术，对于ARM指令集而言，PC总是指向当前指令的下两条指令（每个指令占32位4字节）的地址，即PC的值为当前指令的地址值加8个字节。

ARM 处理器的寄存器格式

程序状态寄存器(CPSR/SPSR)



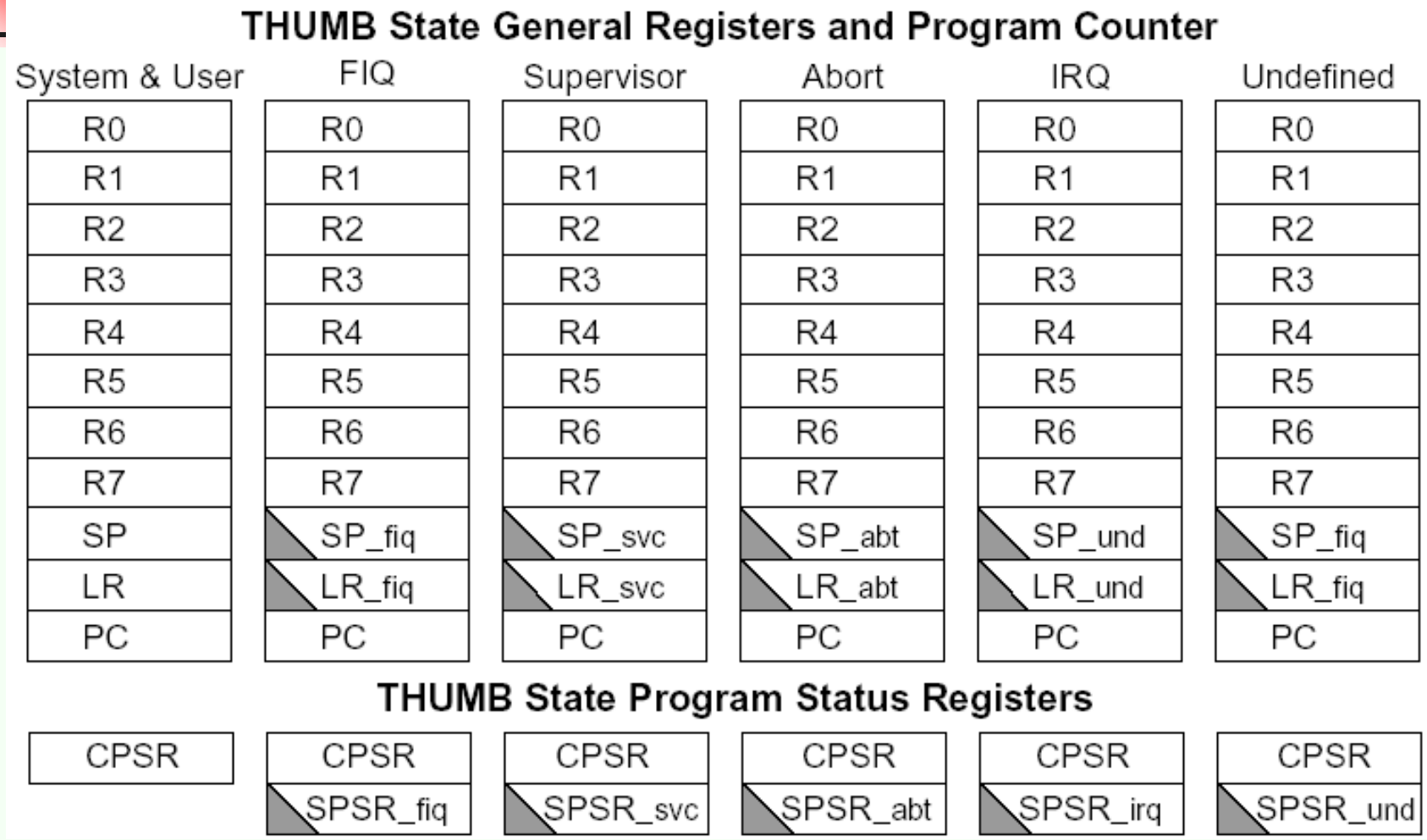
- 寄存器R16用作CPSR(当前程序状态寄存器)，CPSR可在任何运行模式下被访问，它包括条件标志位、中断禁止位、当前处理器模式标志位，以及其他一些相关的控制和状态位。
- 每一种异常模式下又都有一个专用的物理状态寄存器，称为SPSR（备份的程序状态寄存器），异常发生时，SPSR用于保存CPSR的值，从异常退出时则可由SPSR来恢复CPSR。
- 由于用户模式和系统模式不属于异常模式，他们没有SPSR，当在这两种模式下访问SPSR，结果是未知的。

ARM 处理器的寄存器格式

Thumb状态下的寄存器组织

- **Thumb**状态下的寄存器集是**ARM**状态下寄存器集的一个子集
- 程序可以直接访问8个通用寄存器（R7～R0）、程序计数器（PC）、堆栈指针（SP）、连接寄存器（LR）和CPSR。
- 同样，每一种特权模式下都有一组SP、LR和SPSR。

ARM 处理器的寄存器格式



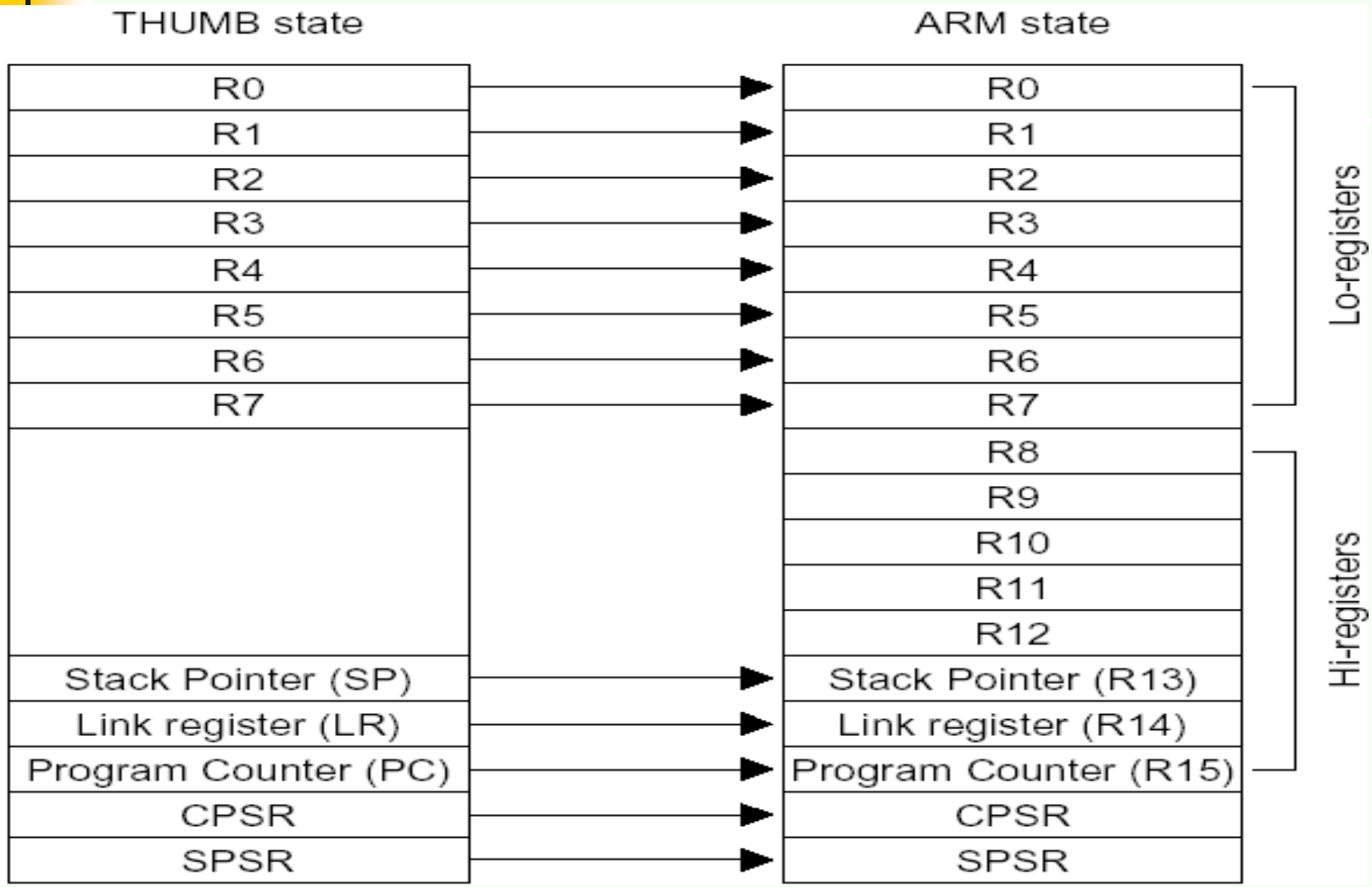
Thumb状态下的寄存器组织图

ARM 处理器的寄存器格式

Thumb状态下的寄存器与ARM状态下的寄存器关系

- Thumb状态下和ARM状态下的R0～R7是相同的。
- Thumb状态下和ARM状态下的CPSR和所有的SPSR是相同的。
- Thumb状态下的SP对应于ARM状态下的R13。
- Thumb状态下的LR对应于ARM状态下的R14。
- Thumb状态下的程序计数器对应于ARM状态下R15。

ARM 处理器的寄存器格式



Thumb状态下的寄存器与ARM状态下的寄存器关系图

ARM 处理器的寄存器格式

访问THUMB状态下的高位寄存器（Hi-registers）

- 在**Thumb**状态下，高位寄存器**R8～R15**并不是标准寄存器集的一部分，但可使用汇编语言程序受限制的访问这些寄存器，将其用作快速的暂存器。
- 使用带特殊变量的MOV指令，数据可以在低位寄存器和高位寄存器之间进行传送；高位寄存器的值可以使用CMP和ADD指令进行比较或加上低位寄存器中的值。

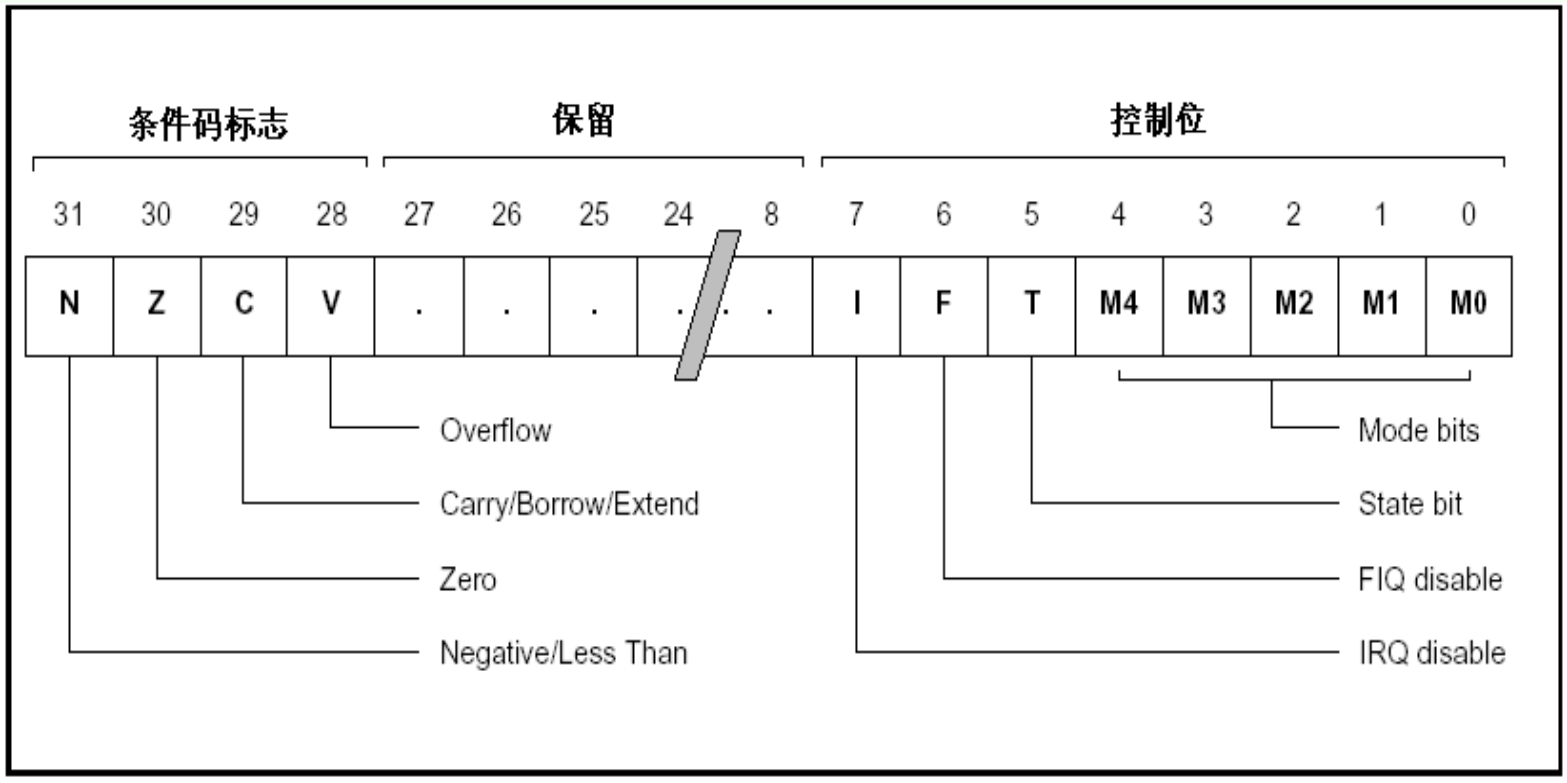
ARM 处理器的寄存器格式

程序状态寄存器

- ARM体系结构包含一个当前程序状态寄存器（CPSR）和五个备份的程序状态寄存器（SPSRs）。备份的程序状态寄存器用来进行异常处理，其功能包括：
  - 保存ALU中的当前操作信息
  - 控制允许和禁止中断
  - 设置处理器的运行模式

ARM 处理器的寄存器格式

程序状态寄存器的每一位的安排



ARM 处理器的寄存器格式

程序状态寄存器的条件码标志

- N、Z、C、V均为条件码标志位。它们的内容可被算术或逻辑运算的结果所改变，并且可以决定某条指令是否被执行
  - 在ARM状态下，绝大多数的指令都是有条件执行的。
  - 在Thumb状态下，仅有分支指令是有条件执行的。

ARM 处理器的寄存器格式

影响标志位的指令

标志位	含　义
N	当用两个补码表示的带符号数进行运算时，N=1 表示运算的结果为负数；N=0 表示运算的结果为正数或零；
Z	Z=1 表示运算的结果为零；Z=0表示运算的结果为非零；
C	加法运算结果进位时，C=1，减法运算借位时，C=0； 移位操作的非加/减运算指令，C为移出的最后一位； 其他的非加/减运算指令，C的值通常不改变。
V	加/减法运算指令，V=1表示符号位溢出。 对于其他的非加/减运算指令，V的值通常不改变。
Q	在ARM v5及以上版本的E系列处理器中，Q标志指示DSP运算指令是否溢出。在其他版本中，Q标志位无定义。

ARM 处理器的寄存器格式

程序状态寄存器的控制位

- 状态寄存器的低8位（I、F、T和M[4：0]）称为控制位，发生异常时这些位可以被改变。如果处理器运行特权模式，这些位也可以由程序修改。
- 中断禁止位I、F：
  - ✓ I=1　禁止IRQ中断;
  - ✓ F=1　禁止FIQ中断。
- T标志位：该位反映处理器的运行状态
  - ✓ ARM体系结构v5及以上的版本的T系列处理器，当该位为1时，程序运行于Thumb状态，否则运行于ARM状态。
  - ✓ ARM体系结构v5及以上的版本的非T系列处理器，当该位为1时，执行下一条指令以引起未定义的指令异常；当该位为0时，表示运行于ARM状态。
- 运行模式位M[4：0]是模式位，决定处理器的运行模式

ARM 处理器的寄存器格式

处理器运行模式及可以访问的寄存器

M[4：0]	处理器模式	可访问的寄存器
0b10000	用户模式	PC，CPSR,R0-R14
0b10001	FIQ模式	PC，CPSR, SPSR_fiq，R14_fiq-R8_fiq, R7～R0
0b10010	IRQ模式	PC，CPSR, SPSR_irq，R14_irq, R13_irq, R12～R0
0b10011	管理模式	PC，CPSR, SPSR_svc，R14_svc, R13_svc, R12～R0,
0b10111	中止模式	PC，CPSR, SPSR_abt，R14_abt,R13_abt, R12～R0,
0b11011	未定义模式	PC，CPSR, SPSR_und，R14_und, R13_und, R12～R0,
0b11111	系统模式	PC，CPSR（ARMv4及以上版本）, R14～R0

ARM 异常处理

异常（Exceptions）

- 当正常的程序执行流程发生暂时的停止时，称之为异常，例如处理一个外部的中断请求。在处理异常之前，当前处理器的状态必须保留，这样当异常处理完成之后，当前程序可以继续执行。处理器允许多个异常同时发生，它们将会按固定的优先级进行处理。
- ARM体系结构中的异常，与8位/16位体系结构的中断有很大的相似之处，但异常与中断的概念并不完全等同。

ARM 异常处理

ARM体系结构所支持的异常类型

异常类型	具体含义
复位	复位电平有效时，产生复位异常，程序跳转到复位处理程序处执行。
未定义指令	遇到不能处理的指令时，产生未定义指令异常。
软件中断	执行SWI指令产生，用于用户模式下的程序调用特权操作指令。
指令预取中止	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，产生指令预取中止异常。
数据中止	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时，产生数据中止异常。
IRQ	外部中断请求有效，且CPSR中的I位为0时，产生IRQ异常。
FIQ	快速中断请求引脚有效，且CPSR中的F位为0时，产生FIQ异常。

ARM 异常处理

对异常的响应

- 当一个异常出现以后，ARM微处理器会执行以下几步操作
  - 将下一条指令的地址存入相应连接寄存器LR，以便程序在处理异常返回时能从正确的位置重新开始执行。
  - 将CPSR复制到相应的SPSR中。
  - 根据异常类型，强制设置CPSR的运行模式位。
  - 强制PC从相关的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处。

ARM 异常处理

异常响应伪代码

- 处理器处于Thumb状态，则当异常向量地址加载入PC时，处理器自动切换到ARM状态。ARM微处理器对异常的响应过程用伪码可以描述为：

```
R14_<Exception_Mode> = Return Link
SPSR_<Exception_Mode> = CPSR
CPSR[4:0] = Exception Mode Number
CPSR[5] = 0
If <Exception_Mode> == Reset or FIQ then
CPSR[6] = 1
CPSR[7] = 1
PC = Exception Vector Address
```

在ARM状态下执行

禁止快速中断

禁止正常中断

转入异常入口地址

ARM 异常处理

从异常返回

- 异常处理完毕之后，ARM微处理器会执行以下几步操作从异常返回：
  - 将连接寄存器LR的值减去相应的偏移量后送到PC中。
  - 将SPSR复制回CPSR中。
  - 若在进入异常处理时设置了中断禁止位，要在此清除。
  - 可以认为应用程序总是从复位异常处理程序开始执行的，因此复位异常处理程序不需要返回。

ARM 异常处理

FIQ（Fast Interrupt Request）

- **FIQ**异常是为了支持数据传输或者通道处理而设计的。
- 若将CPSR的F位置为1，则会禁止FIQ中断，若将CPSR的F位清零，处理器会在指令执行时检查FIQ的输入。注意只有在特权模式下才能改变F位的状态。
- 可由外部通过对处理器上的nFIQ引脚输入低电平产生FIQ。不管是在ARM状态还是在Thumb状态下进入FIQ模式，FIQ处理程序均可以执行以下指令从FIQ模式返回：

```
SUBS    PC,R14_fiq ,#4
```

ARM 异常处理

IRQ（Interrupt Request）

- **IRQ**异常属于正常的中断请求，可通过对处理器的**nIRQ**引脚输入低电平产生，**IRQ**的优先级低于**FIQ**，当程序执行进入**FIQ**异常时，**IRQ**可能被屏蔽。
- 若将CPSR的I位置为1，则会禁止IRQ中断，若将CPSR的I位清零，处理器会在指令执行完之前检查IRQ的输入。注意只有在特权模式下才能改变I位的状态。
- 不管是在ARM状态还是在Thumb状态下进入IRQ模式，IRQ处理程序均可以执行以下指令从IRQ模式返回：

```
SUBS  PC , R14_irq , #4
```

ARM 异常处理

ABORT（中止）

- 中止异常意味着对存储器的访问失败。**ARM**微处理器在存储器访问周期内检查是否发生中止异常。
- 中止异常包括两种类型：
  - 指令预取中止：发生在指令预取时。
  - 数据中止：发生在数据访问时。
- 当指令预取访问存储器失败时，存储器系统向ARM处理器发出存储器中止（Abort）信号，预取的指令被记为无效，但只有当处理器试图执行无效指令时，指令预取中止异常才会发生，如果指令未被执行，例如在指令流水线中发生了跳转，则预取指令中止不会发生。
- 若数据中止发生，系统的响应与指令的类型有关。
- 当确定了中止的原因后，Abort处理程序均可以执行以下指令从中止模式返回，无论是在ARM状态还是Thumb状态：

```
SUBS PC, R14_abt, #4          ; 指令预取中止 PABT
SUBS PC, R14_abt, #8          ; 数据中止 DABT
```

ARM 异常处理

Software Interrupt(软件中断)

- 软件中断指令（SWI）用于进入管理模式，常用于请求执行特定的管理功能。软件中断处理程序执行以下指令可以从**SWI**模式返回，无论是在**ARM**状态还是**Thumb**状态：

```
MOVS  PC , R14_svc
```

- 以上指令恢复**PC**（从**R14_svc**）和**CPSR**（从**SPSR_svc**）的值，并返回到**SWI**的下一条指令。

ARM 异常处理

Undefined Instruction(未定义指令)

- 当ARM处理器遇到不能处理的指令时，会产生未定义指令异常。采用这种机制，可以通过软件仿真扩展ARM或Thumb指令集。
- 处理器执行以下程序返回，无论是在ARM状态还是Thumb状态：

```
MOVS PC, R14_und
```

- 以上指令恢复PC（从R14_und）和CPSR（从SPSR_und）的值，并返回到未定义指令后的下一条指令。

ARM 异常处理

异常向量表（Exception Vectors）

地 址	异 常	进入模式
0x0000,0000	复位	管理模式
0x0000,0004	未定义指令	未定义模式
0x0000,0008	软件中断	管理模式
0x0000,000C	中止（预取指令）	中止模式 PABT
0x0000,0010	中止（数据）	中止模式 DABT
0x0000,0014	保留	保留
0x0000,0018	IRQ	IRQ
0x0000,001C	FIQ	FIQ

ARM 异常处理

异常优先级（Exception Priorities）

优先级	异　常
1（最高）	复位
2	数据中止 DABT
3	FIQ
4	IRQ
5	预取指令中止 PABT
6（最低）	未定义指令、SWI

ARM 异常处理

应用程序中的异常处理

- 当系统运行时，异常可能会随时发生，为保证在**ARM**处理器发生异常时不至于处于未知状态，在应用程序的设计中，首先要进行异常处理，采用的方式是在异常向量表中的特定位置放置一条跳转指令，跳转到异常处理程序，当**ARM**处理器发生异常时，程序计数器**PC**会被强制设置为对应的异常向量，从而跳转到异常处理程序，当异常处理完成以后，返回到主程序继续执行。
- 我们需要处理所有的异常，尽管我们可以简单的在某些异常处理程序处放置死循环。

ARM 微处理器指令的分类

- **ARM**微处理器的指令集是加载/存储型的，也即指令集仅能处理寄存器中的数据，而且处理结果都要放回寄存器中，而对系统存储器的访问则需要通过专门的加载/存储指令来完成。

- **ARM**微处理器的指令集可以分为以下几类：
  - 跳转指令
  - 数据处理指令
  - 程序状态寄存器（**PSR**）处理指令
  - 加载/存储指令
  - 协处理器指令和异常产生指令

ARM 微处理器指令的分类

助记符	指令功能描述
ADC	带进位加法指令
ADD	加法指令
AND	逻辑与指令
B	跳转指令
BIC	位清零指令
BL	带返回的跳转指令
BLX	带返回和状态切换的跳转指令
BX	带状态切换的跳转指令

ARM 微处理器指令的分类

助记符	指令功能描述
CDP	协处理器数据操作指令
CMN	比较反值指令
CMP	比较指令
EOR	异或指令
LDC	存储器到协处理器的数据传输指令
LDM	加载多个寄存器指令
LDR	存储器到寄存器的数据传输指令
MCR	ARM寄存器到协处理器寄存器数据传输

ARM 微处理器指令的分类

助记符	指令功能描述
MLA	乘加运算指令
MOV	数据传送指令
MRC	协处理器寄存器到ARM寄存器数据传输
MRS	传送CPSR或SPSR的内容到通用寄存器
MSR	传送通用寄存器到CPSR或SPSR的指令
MUL	32位乘法指令
MLA	32位乘加指令
MVN	数据取反传送指令

ARM 微处理器指令的分类

助记符	指令功能描述
ORR	逻辑或指令
RSB	逆向减法指令
RSC	带借位的逆向减法指令
SBC	带借位减法指令
STC	协处理器寄存器写入存储器指令
STM	批量内存字写入指令
STR	寄存器到存储器的数据传输指令
SUB	减法指令

ARM 微处理器指令的分类

助记符	指令功能描述
SWI	软件中断指令
SWP	交换指令
TEQ	相等测试指令
TST	位测试指令

ARM 微处理器指令的格式

- 典型的**ARM**指令的一般书写格式

<opcode> {<cond>} {s} <Rd>,<Rn>,<shifter_operand>

其中**<>**中的内容是必须的，**{}**中的内容是可选的。**opcode**为指令助记符；**cond**为指令执行的条件编码；**s**决定指令执行结果是否影响**CPSR**寄存器中相应位的值（加**s**则影响）；**Rd**为目标寄存器；**Rn**为存放第一个源操作数的寄存器；**shifter_operand**为第二个源操作数（立即数、寄存器、寄存器移位之一）

ARM 微处理器指令的格式

- 典型的**ARM**指令的一般书写格式

<opcode> {<cond>} {s} <Rd>,<Rn>,<shifter_operand>

例如指令：ADDEQS　R0,R1,R2;

则，该指令的编码格式如下第三行所示：

31~28	27~25	24~21	20	19~16	15~12	11~~~~~~~~~~~~0
cond		opcode	S	Rn	Rd	op2
0000	001	0100	1	0001	0000	000000000010

ARM 微处理器指令的格式

- **ARM指令可选后缀：**

  有**P**，则传送前加上有效地址，否则传送后加上有效地址，**pre/post**即前/后索引偏移；

  有**U**，则基地址+偏移量，否则基地址-偏移量，**up/down**；

  有**B**，则传送**Rd**的最低有效字节；

  有**W**，则回写地址，否则不回写；

  L表示写入存储器还是从存储器读出；

  A表示是否包含加法。

ARM 微处理器指令的格式

- **Thumb**指令为**ARM**指令的一个子集，其中只有分支指令含有条件码；没有条件选择和**S**位；无协处理器相关指令；无寄存器交换指令，无访问**PSR**指令、乘加指令等；指令的第二操作数使用受限；无堆栈操作指令；除分支指令外的指令都是无条件执行的。

ARM 微处理器指令的条件域

- 当处理器工作在**ARM**状态时，几乎所有的指令均能根据**CPSR**中条件码的状态和指令的条件域有条件的执行。当指令的执行条件满足时，指令被执行，否则指令被忽略。

- 每一条**ARM**指令包含**4**位的条件码，位于指令的最高**4**位**[31:28]**。条件码共有**16**种，每种条件码可用两个字符表示，这两个字符可以添加在指令助记符的后面和指令同时使用。例如，跳转指令**B**可以加上后缀**EQ**变为**BEQ**表示“相等则跳转”，即当**CPSR**中的**Z**标志置位时发生跳转。

ARM 微处理器指令的条件域

条件码	后缀	标 志	含 义
0000	EQ	Z置位	相等
0001	NE	Z清零	不相等
0010	CS	C置位	无符号数大于或等于
0011	CC	C清零	无符号数小于
0100	MI	N置位	负数
0101	PL	N清零	正数或零
0110	VS	V置位	溢出
0111	VC	V清零	未溢出

ARM 微处理器指令的条件域

条件码	后缀	标 志	含 义
1001	LS	C清零Z置位	无符号数小于或等于
1010	GE	N等于V	有符号数大于或等于
1011	LT	N不等于V	有符号数小于
1100	GT	Z清零且（N等于V）	有符号数大于
1101	LE	Z置位或（N不等于V）	有符号数小于或等于
1110	AL	总是，忽略	无条件执行
1000	HI	C置位Z清零	无符号数大于或等于
1111	NV	从不，NOP	不执行此指令

ARM 微处理器指令的寻址方式

- **ARM指令系统支持如下几种常见的寻址方式：**

- 立即寻址
- 寄存器寻址
- 寄存器间接寻址
- 基址变址寻址
- 多寄存器寻址
- 相对寻址
- 堆栈寻址

ARM 微处理器指令的寻址方式

立即寻址

- 立即寻址也叫立即数寻址，这是一种特殊的寻址方式，操作数本身就在指令中给出，只要取出指令也就取到了操作数。这个操作数被称为立即数，对应的寻址方式也就叫做立即寻址。例如以下指令：

```
ADD  R0，R0，＃1              /*R0←R0＋1*/
ADD  R0，R0，＃0x3f           /*R0←R0＋0x3f*/
```

- 在以上两条指令中，第二个源操作数即为立即数，要求以“＃”为前缀，对于以十六进制表示的立即数，还要求在“＃”后加上“0x”。

ARM 微处理器指令的寻址方式

寄存器寻址

- 寄存器寻址就是利用寄存器中的数值作为操作数，这种寻址方式是各类微处理器经常采用的一种方式，也是一种执行效率较高的寻址方式。以下指令：

ADD R0，R1，R2　　/*R0←R1＋R2*/

- 该指令的执行效果是将寄存器**R1**和**R2**的内容相加，其结果存放在寄存器**R0**中。

ARM 微处理器指令的寻址方式

寄存器间接寻址

- 寄存器间接寻址就是以寄存器中的值作为操作数的地址，而操作数本身存放在存储器中。例如以下指令：

```
ADD R0，R1，[R2]        /*R0←R1＋[R2]*/
LDRR0，[R1]             /*R0←[R1]*/
STRR0，[R1]             /*[R1]←R0*/
```

- 在第一条指令中，以寄存器**R2**的值作为操作数的地址，在存储器中取得一个操作数后与**R1**相加，结果存入寄存器**R0**中。
- 第二条指令将以**R1**的值为地址的存储器中的数据传送到**R0**中。
- 第三条指令将**R0**的值传送到以**R1**的值为地址的存储器中。

ARM 微处理器指令的寻址方式

基址变址寻址

- 基址变址寻址就是将寄存器（该寄存器一般称作基址寄存器）的内容与指令中给出的地址偏移量相加，从而得到一个操作数的有效地址。变址寻址方式常用于访问某基地址附近的地址单元。采用变址寻址方式的指令常见有以下几种形式，如下所示：

```
LDR R0，[R1，＃4]          ；R0←[R1＋4]
LDR R0，[R1，＃4]!         ；R0←[R1＋4]、R1←R1＋4
LDR R0，[R1]，＃4          ；R0←[R1]、R1←R1＋4
LDR R0，[R1，R2]           ；R0←[R1＋R2]
```

- 在第一条指令中，将寄存器**R1**的内容加上**4**形成操作数的有效地址，从而取得操作数存入寄存器**R0**中。
- 在第二条指令中，将寄存器**R1**的内容加上**4**形成操作数的有效地址，从而取得操作数存入寄存器**R0**中，然后，**R1**的内容自增**4**个字节（符号！的含义：自动更新基地址）。
- 在第三条指令中，以寄存器**R1**的内容作为操作数的有效地址，从而取得操作数存入寄存器**R0**中，然后，**R1**的内容自增**4**个字节。
- 在第四条指令中，将寄存器**R1**的内容加上寄存器**R2**的内容形成操作数的有效地址，从而取得操作数存入寄存器**R0**中。

ARM 微处理器指令的寻址方式

多寄存器寻址

- 采用多寄存器寻址方式，一条指令可以完成多个寄存器值的传送。这种寻址方式可以用一条指令完成传送最多16个通用寄存器的值。以下指令：

```
LDMIA  R0，{R1，R2，R3，R4}      ；R1←[R0]
                                 ；R2←[R0+4]
                                 ；R3←[R0+8]
                                 ；R4←[R0+12]
```

- 该指令的后缀IA表示在每次执行完加载/存储操作后，R0按字长度增加，因此，指令可将连续存储单元的值传送到R1～R4。

ARM 微处理器指令的寻址方式

相对寻址

- 与基址变址寻址方式相类似，相对寻址以程序计数器PC的当前值为基地址，指令中的地址标号作为偏移量，将两者相加之后得到操作数的有效地址。以下程序段完成子程序的调用和返回，跳转指令BL采用了相对寻址方式：

```
BL      NEXT        ；跳转到子程序NEXT处执行
……
NEXT
……
MOV     PC，LR      ；从子程序返回
```

ARM 微处理器指令的寻址方式

堆栈寻址

- 堆栈是一种数据结构，按先进后出（**First In Last Out，FILO**）的方式工作，使用一个称作堆栈指针的专用寄存器指示当前的操作位置，堆栈指针总是指向栈顶。
- 当堆栈指针指向最后压入堆栈的数据时，称为满堆栈（**Full Stack**），而当堆栈指针指向下一个将要放入数据的空位置时，称为空堆栈（**Empty Stack**）。

ARM 微处理器指令的寻址方式

堆栈寻址

- 根据堆栈的生成方式，又可以分为递增堆栈（**Ascending Stack**）和递减堆栈（**Decending Stack**），当堆栈由低地址向高地址生成时，称为递增堆栈，当堆栈由高地址向低地址生成时，称为递减堆栈。这样就有四种类型的堆栈工作方式
  - 满递增堆栈**FA**：堆栈指针指向最后压入的数据，且由低地址向高地址生成。
  - 满递减堆栈**FD**：堆栈指针指向最后压入的数据，且由高地址向低地址生成。
  - 空递增堆栈**EA**：堆栈指针指向下一个将要放入数据的空位置，且由低地址向高地址生成。
  - 空递减堆栈**ED**：堆栈指针指向下一个将要放入数据的空位置，且由高地址向低地址生成