# 第二章 ARM体系结构

# 2.1 ARM体系结构概述

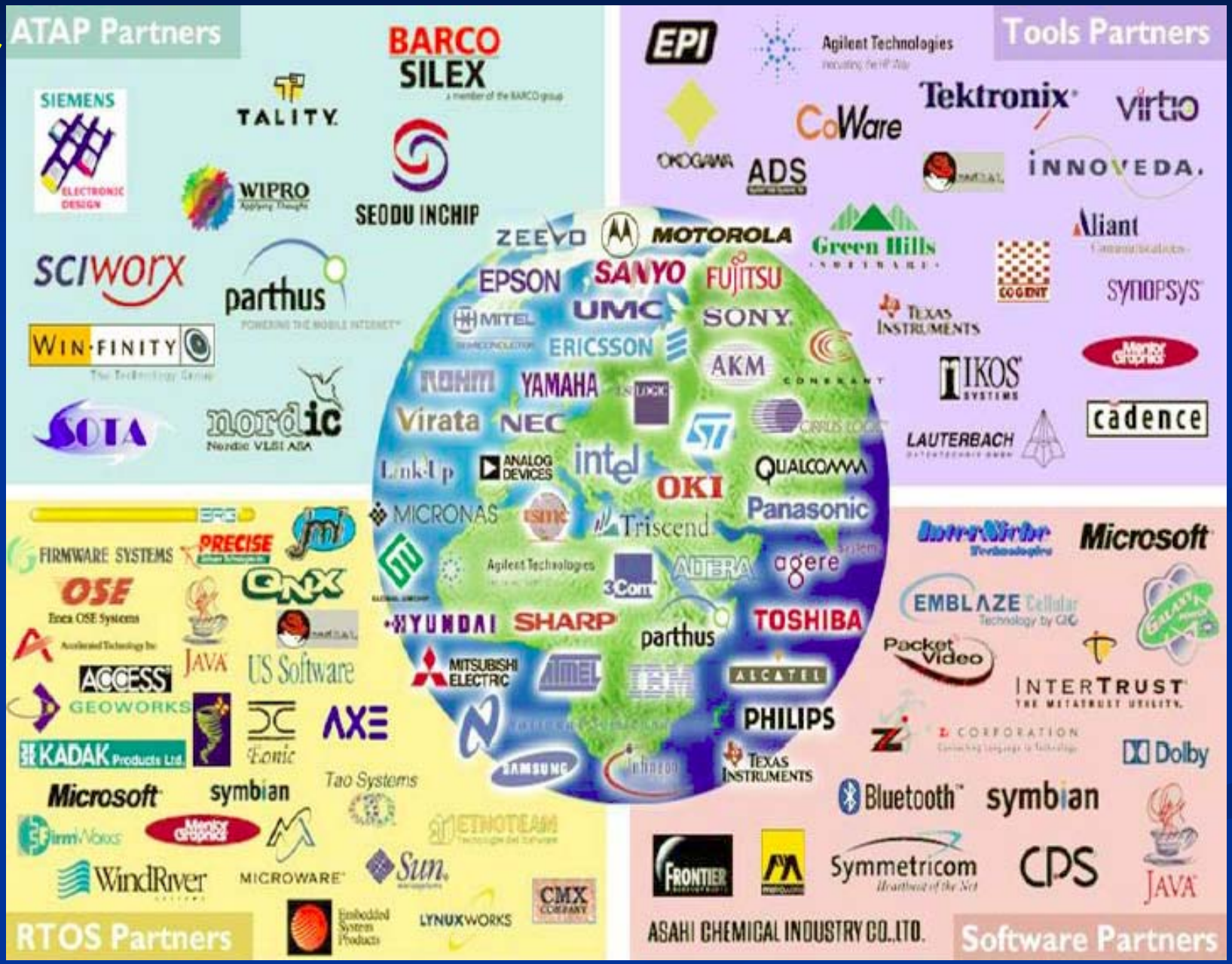
ARM：Advanced RISC Machines

- 1985 年4月26日：第一个ARM原型在英国剑桥诞生
- 1990年11月：成立了Advanced RISC Machines Limited（ARM公司）

ARM公司是设计公司，是知识产权（IP）供应商，本身不生产芯片，靠转让设计许可由合作伙伴来生产各具特色的芯片，已有100多个合作伙伴。

# 合作伙伴

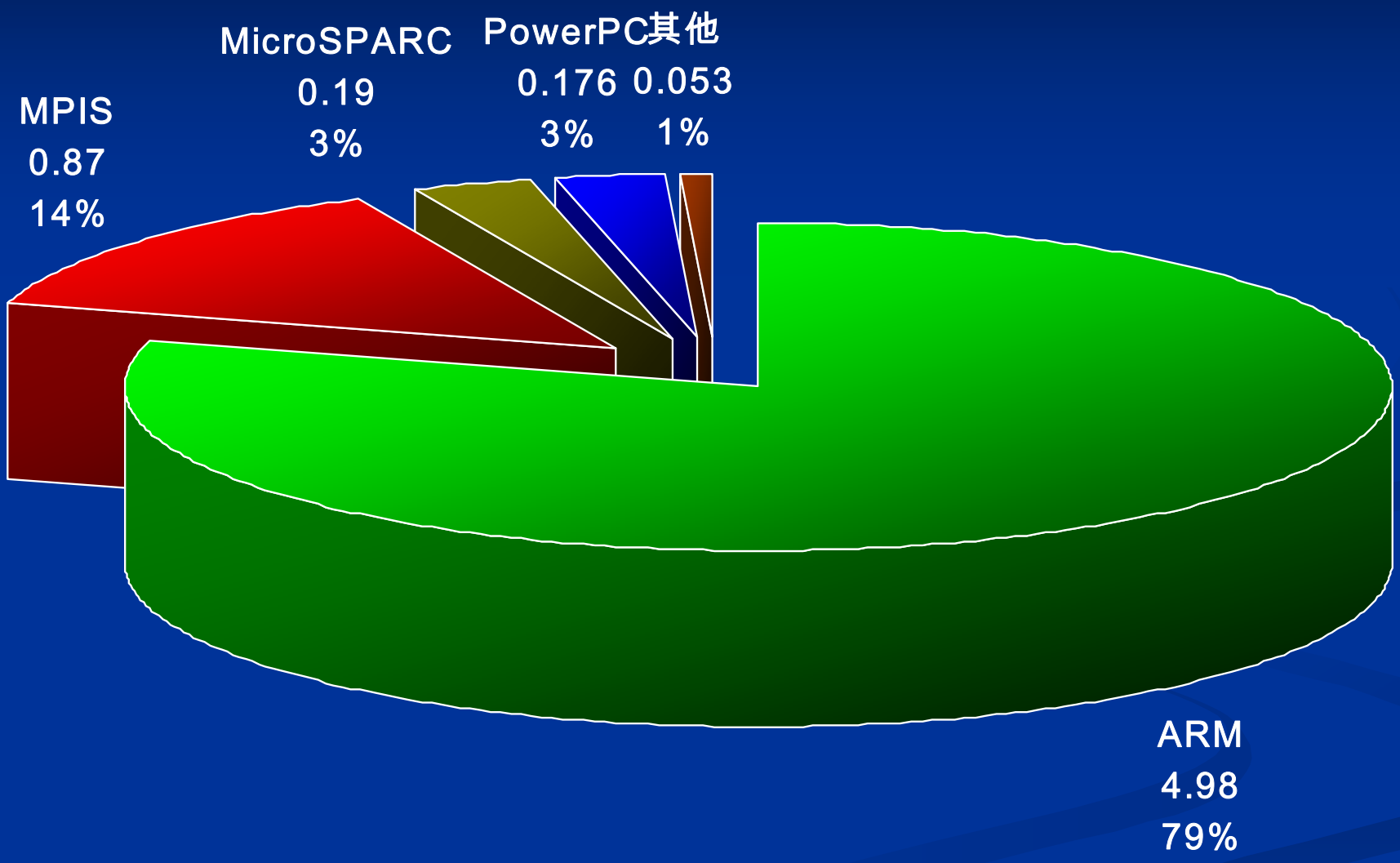
技术共享计划


ATAP Partners
Tools Partners
RTOS Partners
Software Partners
BARCO SILEX
SIEMENS
TALITY
WIPRO
SEODU INCHIP
sciworx
parthus
WIN-FINITY
nordic
SOTA
EPI
Agilent Technologies
Tektronix
virtio
CoWare
ADS
INNOVEDA
Aliant
Green Hills
COGENT
SYNOPSYS
TEXAS INSTRUMENTS
IKOS
cadence
LAUTERBACH
ZEEVO
MOTOROLA
EPSON
SANYO
FUJITSU
UMC
SONY
MITEL
ERICSSON
AKM
ROHM
YAMAHA
Virata
NEC
Link-Up
ANALOG DEVICES
intel
OKI
QUALCOMM
MICRONAS
Triscend
Panasonic
Agilent Technologies
3Com
ALTERA
agere
HYUNDAI
SHARP
parthus
TOSHIBA
MITSUBISHI ELECTRIC
ATMEL
IBM
ALCATEL
PHILIPS
SAMSUNG
Infineon
TEXAS INSTRUMENTS
FIRMWARE SYSTEMS
PRECISE
OSE
Enea OSE Systems
QNX
JAVA
US Software
ACCESS
GEOWORKS
KADAK Products Ltd.
AXE
Tao Systems
Microsoft
symbian
WindRiver
MICROWARE
Sun
CMX
LYNUXWORKS
Microsoft
EMBLAZE
Packet Video
INTERTRUST
Dolby
Bluetooth
symbian
Symmetricom
CPS
JAVA
FRONTIER
ASAHI CHEMICAL INDUSTRY CO.,LTD.

# ARM的应用领域


Samsung ML5100A
Diamond Multimedia Rio 600
JVC "Pixstar" GC-X1
Alba Bush
Internet TV
Nintendo
Gameboy
Advance
Iomega HipZip
Sony MZ-R90 MiniDisc
Lexmark Z52 Color Jetprinter
Ericsson
R380
HP CapShare
HP Jornada 820
Nokia 8810
Nokia Mediamaster

# 2002年嵌入式内核总量：6.265亿



# 2.1.1 RISC体系结构

1979年美国加州大学伯克利分校的帕特逊等人提出，至今尚未有严格的定义。

比较普遍的认为是，RISC应该是一种计算机设计的基本原则，它的出现标志着计算机体系结构发展上的一个重要里程碑。

**20%-80%定律：**在CISC指令集中只有20%的指令经常用到，约占程序总量的80%；而80%的指令则很少用到，约占程序总量的20%。

# RISC架构的特点

卡内基·梅隆大学：

- 指令集中大多数指令只需执行简单和基本的功能。其执行过程是在一个机器周期内完成的；
- 由于存储器访问指令执行时间长，尽量避免使用，而采用加载和存储（Load/Store）指令。运算指令的操作数都经加载/存储指令，从存储器取出后预先存放在寄存器堆内，以加快执行速度；
- 芯片逻辑不采用或少采用微码技术，而采用硬布线逻辑，以减少指令解释的开销；
- 减少指令数和寻址方式，使控制部件简化，加快执行速度；
- 指令格式固定，指令译码简化；
- 编译开销很大，应尽可能优化。

# 2.1.2 ARM架构的特点

## ARM体系继承了RISC架构的优点

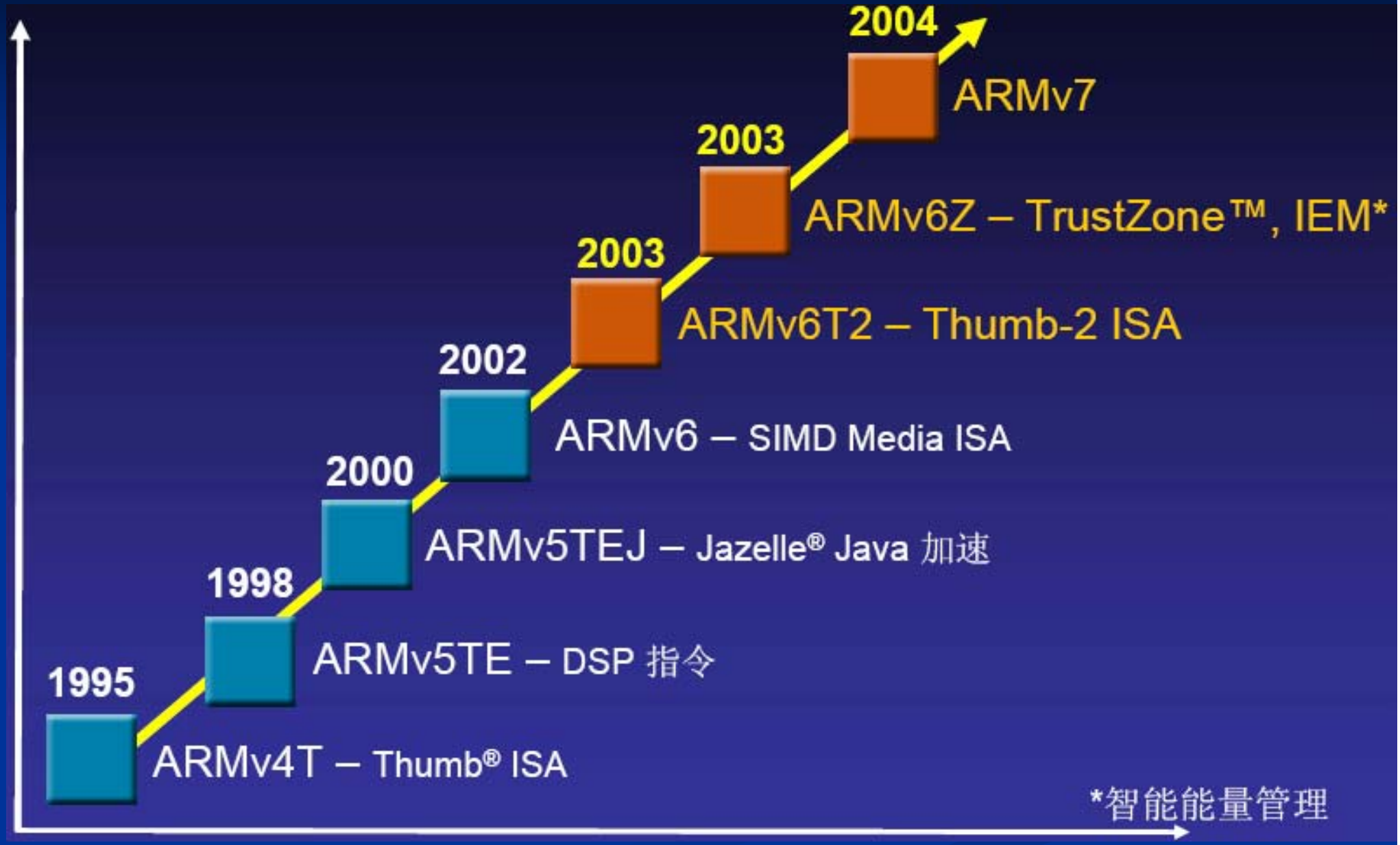
- 大量的寄存器，都可用于多种用途；
- Load/Store体系结构
- 3地址指令（两个源操作数寄存器和结果寄存器独立设定）
- 每条指令都条件执行包含非常强大的多寄存器Load和Store指令
- 能在单时钟周期执行的单条指令内完成一项普通的移位操作和一项普通的ALU操作
- 通过协处理器指令集来扩展ARM指令集，包括在编程模式下增加了新的寄存器和数据类型
- 在Thumb体系结构中以高密度16位压缩形式表示指令集

# 2.1.3 ARM体系结构的版本

1. V1（ARM1）：
   - 处理乘法指令之外的基本数据处理指令
   - 基于字节、字和多字的读取和写入指令
   - 包括子程序调用指令BL（带链转移）在内的跳转指令
   - 供操作系统使用的软件中断指令SWI
   - 寻址空间：64MB（$2^{26}$）
2. V2（ARM2和ARM3）
   - 增加了乘法指令和乘加法指令
   - 支持协处理器的指令
   - 对于FIQ模式，提供了额外的两个备份寄存器
   - 增加了SWP及SWPB最基本的存储器和寄存器交换指令
   - 寻址空间：64MB（$2^{26}$）

3. V3（ARM6）：
   - 增加了当前程序状态寄存器CPSR和保存程序状态寄存器SPSR
   - 增加了MRS/MSR指令，以访问CPSR/SPSR
   - 增加了从异常处理返回的指令
   - 寻址空间：4GB（$2^{32}$）
4. V4（ARM7、ARM8、ARM9和StorngARM）
   - 增加了符号化和非符号化半字及符号化字节存/取指令
   - 增加了16位Thumb指令集
   - 完善了软件中断SWI指令的功能
   - 增加了处理器特权方式，在该模式下使用的是用户模式下的寄存器

5. V5（ARM10和XScale）：
   - 增加了带有链接和交换的转移指令BLX
   - 增加了计数前导零CLZ指令，该指令允许更有效的整数除法和中断优先程序
   - 增加了软件断点中断BRK指令
   - 增加了DSP指令（V5TE版）
   - 为协处理器增加了更多可选择的指令
   - 对乘法指令如何设置标志进行了严密的定义
6. V6（ARM11）
   - Thumb：35%代码压缩
   - DSP扩充：高性能定点DSP功能
   - Jazelle：Java性能优化，可提高8倍
   - Media扩充：音/视频性能优化，可提高4倍


2004
ARMv7
2003
ARMv6Z – TrustZone™, IEM*
2003
ARMv6T2 – Thumb-2 ISA
2002
ARMv6 – SIMD Media ISA
2000
ARMv5TEJ – Jazelle® Java 加速
1998
ARMv5TE – DSP 指令
1995
ARMv4T – Thumb® ISA
*智能能量管理

# 2.1.4 ARM体系结构的变量

## 1. T变量（Thumb指令集）

Thumb指令集是将ARM指令集的一个子集重新编码而形成的一个指令集。ARM指令长度为32位，Thumb指令长度为16位。因此，使用Thumb指令集可以得到密度更高的代码。与ARM指令集相比，Thumb指令集具有以下的局限：

- ✓ 完成同样的操作，Thumb代码通常使用更多的指令。因此，在对系统**运行时间要求苛刻**的应用场合，最好还是采用ARM指令；
- ✓ Thumb指令集**不包括异常处理**所需的指令

Thumb有2个版本，Thumb V1用于ARM V4的T变量， Thumb V2用于ARM V5的T变量。

2. M变量（长乘法指令）

   增加了32×32的乘法/乘加指令

3. E变量（增强型DSP指令）

4. J变量（Java加速器）

5. SIMD变种（媒体功能扩展）

# ARM体系版本的命名格式

用下面的字符串连接起来，即形成ARM体系结构版本：

- 字符串ARMv
- 变量字符（除M变量）,V4以上版本中M变量为标配，因而通常不需列出
- 用x表示排除某些功能

如ARMV4T、ARMV5TxM等

# 2.1.5 ARM处理器内核（核）

ARM处理器内核当前有6个系列产品：ARM7, ARM9, ARM9E, ARM10E, SecurCore以及最新的ARM11系列。进一步的产品来自ARM的合作伙伴，例如Intel XScale微体系结构和StrongARM产品。

在ARM内核的基础上，增加Cache、MMU、CP15、AMBA接口等就构成了ARM处理器核。

| ARM7性能特征 | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Cache大小（指令/数据） | 紧密耦合存储器（TCM） | 存储器管理 | AHB总线接口 | Thumb | DSP | Jazelle |
| ARM7TDMI | 无 | 无 | 无 | 有 | 有 | 无 | 无 |
| ARM7TDMI-S | 无 | 无 | 无 | 有 | 有 | 无 | 无 |
| ARM7EJ-S | 无 | 无 | 无 | 有 | 有 | 有 | 有 |
| ARM720T | 8k | 无 | MMU | 有 | 有 | 无 | 无 |

**ARM9 性能特征**

| | Cache 大小（指令/数据） | 紧密耦合存储器（TCM） | 存储器管理 | AHB 总线接口 | Thumb | DSP | Jazelle |
|---|---|---|---|---|---|---|---|
| ARM920T | 16K/16K | 无 | MMU | 有 | 有 | 无 | 无 |
| ARM922T | 8K/8K | 无 | MMU | 有 | 有 | 无 | 无 |
| ARM940T | 4K/4K | 无 | MMU | 有 | 有 | 无 | 无 |

**ARM9E 性能特征**

| | Cache 大小（指令/数据） | 紧密耦合存储器（TCM） | 存储器管理 | AHB 总线接口 | Thumb | DSP | Jazelle |
|---|---|---|---|---|---|---|---|
| ARM926EJ-S | 4-128K/4-128 | 有 | MMU | 双 AHB | 有 | 有 | 有 |
| ARM946E-S | 4-1MB/4-1MB | 有 | MPU | AHB | 有 | 有 | 无 |
| ARM966E-S | 无 | 有 | 无 | AHB | 有 | 有 | 无 |

| ARM10E 性能特征 | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Cache（指令/数据） | 紧密耦合存储器（TCM） | 存储器管理 | AHB 总线接口 | Thumb | DSP | Jazelle |
| ARM1020E | 32K/32K | 无 | MMU | 双 AHB | 有 | 有 | 无 |
| ARM1022E | 16K/16K | 无 | MMU | 双 AHB | 有 | 有 | 无 |
| ARM1026EJ-S | 可变 | 有 | MMU+MMU | 双 AHB | 有 | 有 | 有 |

| ARM11 性能特征 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Cache（指令/数据） | 紧密耦合存储器（TCM） | 存储器管理 | AHB 总线接口 | DSP | Jazelle | SIMD | 浮点运算 |
| ARM1136J-S | 4-64K | 有 | MMU | 四个 64 位 AHB | 有 | 有 | 有 | 无 |
| ARM1136JF-S | 4-64K | 有 | MMU | 四个 64 位 AHB | 有 | 有 | 有 | 有 |

**ARM SecurCore 性能特征**

| | Cache 大小（指令/数据） | 紧密耦合存储器（TCM） | 存储器管理 | AHB 总线接口 | Thumb | DSP | Jazelle |
|---|---|---|---|---|---|---|---|
| SC100 | 无 | 无 | MPU | 无 | 有 | 无 | 无 |
| SC110 | 无 | 无 | MPU | 无 | 有 | 无 | 无 |
| SC200 | 可选 | 无 | MPU | 无 | 有 | 有 | 有 |
| SC210 | 可选 | 无 | MPU | 无 | 有 | 有 | 有 |

**基于 ARM 的 Intel 微处理器**

| | Cache 大小（指令/数据） | 紧密耦合存储器（TCM） | 存储器管理 | AHB 总线接口 | Thumb | DSP | Jazelle |
|---|---|---|---|---|---|---|---|
| Strong ARM | 16K/8K | 无 | MMU | N/A | 无 | 无 | 无 |
| XScale | 32K/32K | 无 | MMU | N/A | 有 | 有 | 无 |

# 2.1.6 ARM处理器模式

1. Usr模式：正常程序执行的模式
2. fiq模式：用于处理高速数据传输和通道处理
3. irq模式：用于通常的中断处理
4. sve模式：供操作系统使用的一种保护模式
5. abt模式：实现虚拟存储器和/或存储器保护
6. und模式：支持硬件协处理器的软件仿真
7. sys模式：运行特权操作系统任务（v4及以上版本）

# 2.1.6 寄存器组织

ARM共有37个寄存器：

- 31个通用寄存器，32位
  - 不分组寄存器R0~R7
  - 分组寄存器R8~R14（22个）
  - 程序计数器R15：ARM状态，位[1：0]为0，位[31：2]保存PC；Thumb状态，位[0]为0，位[31：1]保存PC
- 6个状态寄存器，32位，但只使用了其中12位

分组寄存器：

2组R8~R12：1组用于FIQ模式，另1组用于除FIQ外其它模式

6组R13和R14：1组用于用户模式和系统模式，令5组用于其余5种模式

# 2.1.7 ARM基本寻址方式

- 寄存器寻址

```
ADD R0，R1，R2          ；R1+R2→R0
```

- 立即寻址

```
ADD R2，R2，#1          ；R2+1 →R2
```

- 寄存器移位寻址

```
ADD R3，R2，R1，LSL #3     ；R2+8×R1 →R3
```

- 寄存器间接寻址

```
LDR R0，[R1]            ；（R1）→R0
```

- 变址寻址

```
LDR R0，[R1+4]          ；（R1+4）→R0
```

- 多寄存器寻址

```
LDMIA R1，{R0，R2，R5}      ；  (R1) →R0
                            ；  (R1+4) →R2
                            ；  (R1+8) →R5
```

- 堆栈寻址

分为向上生长和向下生长2种

堆栈指针指向最后入栈的有效数据项，称为满堆栈；

堆栈指针指向下一个数据项放入的空位置，称为空堆栈；

ARM处理器支持4种类型的堆栈：满递增，空递增，满递减，空递减

- 块拷贝寻址

  与堆栈寻址类似有4种类型

  STMIA R9！，{R0，R1，R5}　　；满递增

  STMIB R0！，{R2 - R9}　　；空递增

- 相对寻址

  与基址寻址类似，由PC提供基址，地址码字段作为偏移量，得到有效地址

# 2.1.8 ARM汇编指令

ARM有三种类型的数据指令

1. 数据处理指令：这类指令只能使用和改变寄存器中的值
2. 数据传送指令：这类指令把存储器中的值拷贝到寄存器（Load）或把寄存器的值拷贝到存储器中（Store）
3. 控制流指令：一般指令在执行时使用存储于连续的存储器地址中的指令。控制流指令使执行切换到不同的地址。切换是永久的或保存返回地址以恢复原来的执行顺序，或者陷入系统代码。

# 1. 数据处理指令

- 功能：完成寄存器的数据的算术和逻辑操作
- 典型特征：需要两个操作数，产生单个结果
- 使用原则：

① 所有的操作数是32位宽，或来自寄存器，或在指令中定义的立即数

② 如果有结果，则结果为32位宽，放在一个寄存器中

③ 每一个操作数寄存器和结果寄存器都在指令中独立的指定，即使用3地址模式

例：ADD r0，r1，r2； r0=r1+r2

# ARM数据处理指令一览

1. 简单的寄存器操作
   ① 算术操作：ADD、ADC、SUB、SBC、RSB、RSC
   ② 按位逻辑与：AND、ORR、EOR、BIC
   ③ 寄存器传送操作：MOV、MVN
   ④ 比较操作：CMP、CMN、TST、TEQ
2. 立即数操作：如ADD r1，r1，#1
3. 寄存器移位操作：LSL、LSR、ASL、ASR、ROR、RRX

   例 ADD r3，r2，r1，LSL #3
4. 设置条件码
5. 乘法：MUL、MLA

# 2. 数据传送指令

- 单寄存器的Load和Store指令
- 多寄存器的Load和Store指令
- 单寄存器交换指令

# 3. 控制流指令

- 转移指令
- 子程序返回指令
- 监控程序调用指令

# 实例

```
                ENTRY                          ; 代码入口
START           ADR     r1,TEXT                ; r1← “Hello World”
LOOP            LDRB    r0,[r1],#1             ;读取下一字节
                CMP     r0,#0                  ; 检查文本终点
                SWINE   SWI_WriteC             ; 若非终点，则打印
                BNE     LOOP                   ; 并返回LOOP
                SWI     SWI_Exit               ; 执行结束
TEXT            =       “Hello World”,&0a,&0d,0
                END                            ; 程序结束
```

# 2.2 ARM存储器结构

ARM架构处理器的存储器寻址空间为4GB，一般片内无RAM/ROM，系统所需的RAM/ROM通过总线外接。但有的带有指令快存（I-Cache）和数据快存（D-Cache）。

ARM存储系统的体系结构适应不同的嵌入式应用系统的需要差别很大。最简单的存储系统如同单片机一样使用物理地址；而复杂的系统将会将会包括一种或多种下面的技术：

- 用于存储管理的系统控制协处理器CP15
- 存储器保护单元MPU（Memory Protect Unit）
- 存储器管理单元MMU（Memory management Unit)
- Cache及Write Buffer技术
- 快速进程上下文切换技术

# 2.2.1 系统控制协处理器CP15

ARM的存储器管理是通过系统控制协处理器CP15完成的。CP15包含16个32位寄存器C0～C15，在系统模式下访问CP15中的寄存器需使用下列2个指令：

MCR：ARM寄存器→CP15寄存器的数据传送指令

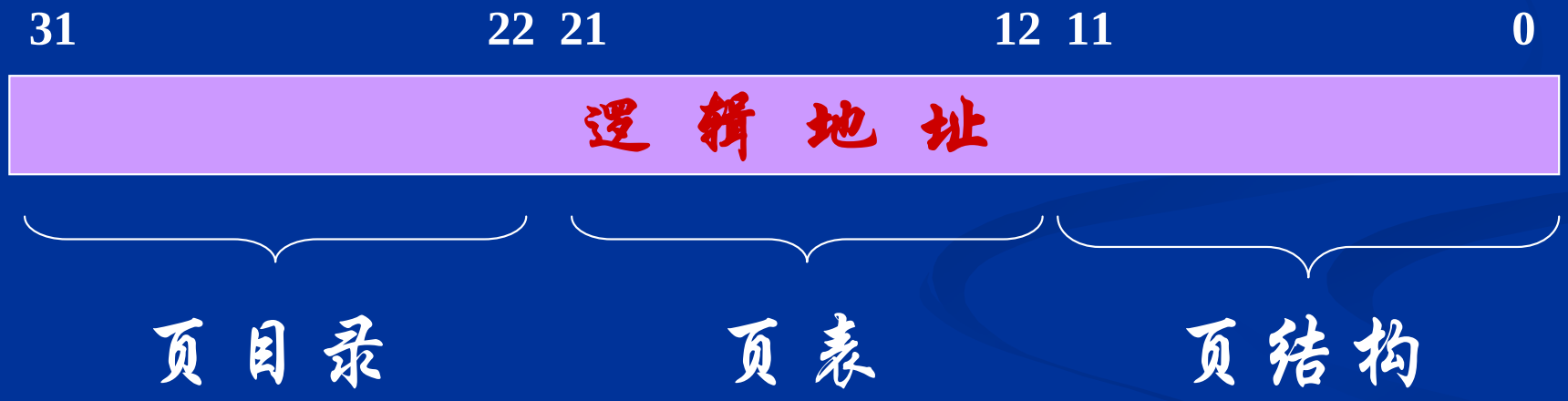
MRC： CP15寄存器→ ARM寄存器的数据传送指令

在用户模式下访问CP15中的寄存器需使用软中断（SWI）调用的方式。

其中，C1是一个控制寄存器，它控制MMU（MPU）的使能、数据Cache或统一Cache的使能、指令Cache的使能、写缓冲使能等。

# 2.2.2 ARM的MMU

ARM架构处理器中存储粒度，根据不同的应用方式，可有：大页（64KB）、小页（4KB）、微小页（1KB）和段（1MB），常用的是小页。



# 2.2.3 ARM Cache结构

常用的Cache有指令和数据统一的Cache及指令和数据分离的Cache。

| Cache结构 | 性能 |
|---|---|
| 无Cache | 1 |
| I-Cache（only） | 1.95 |
| D-Cache（only） | 1.13 |
| I-Cache + D-Cache | 2.5 |

# 2.3 ARM I/O结构

ARM架构的处理器核和处理器内核一般都没有I/O，而是通过AMBA（先进微控制器总线架构）总线来扩充。ARM系统访问I/O功能的标准方法是使用存储器映射I/O，并标识为非Cache和非缓冲。对于数据流量较大的事件，需采用DMA，但大多数ARM处理器都没有DMA。而是采用系统提供的FIQ来处理速率比较高的事件，而对于其余的I/O源仍采用一般中断IRQ。

# 2.4 ARM AMBA接口

ARM通过先进微控制器总线架构AMBA（Advanced Microcontroller Bus Architecture）来扩展不同体系架构的宏单元和I/O部件。AMBA已成为事实上的片上总线OCB（On Chip Bus）标准。包括：

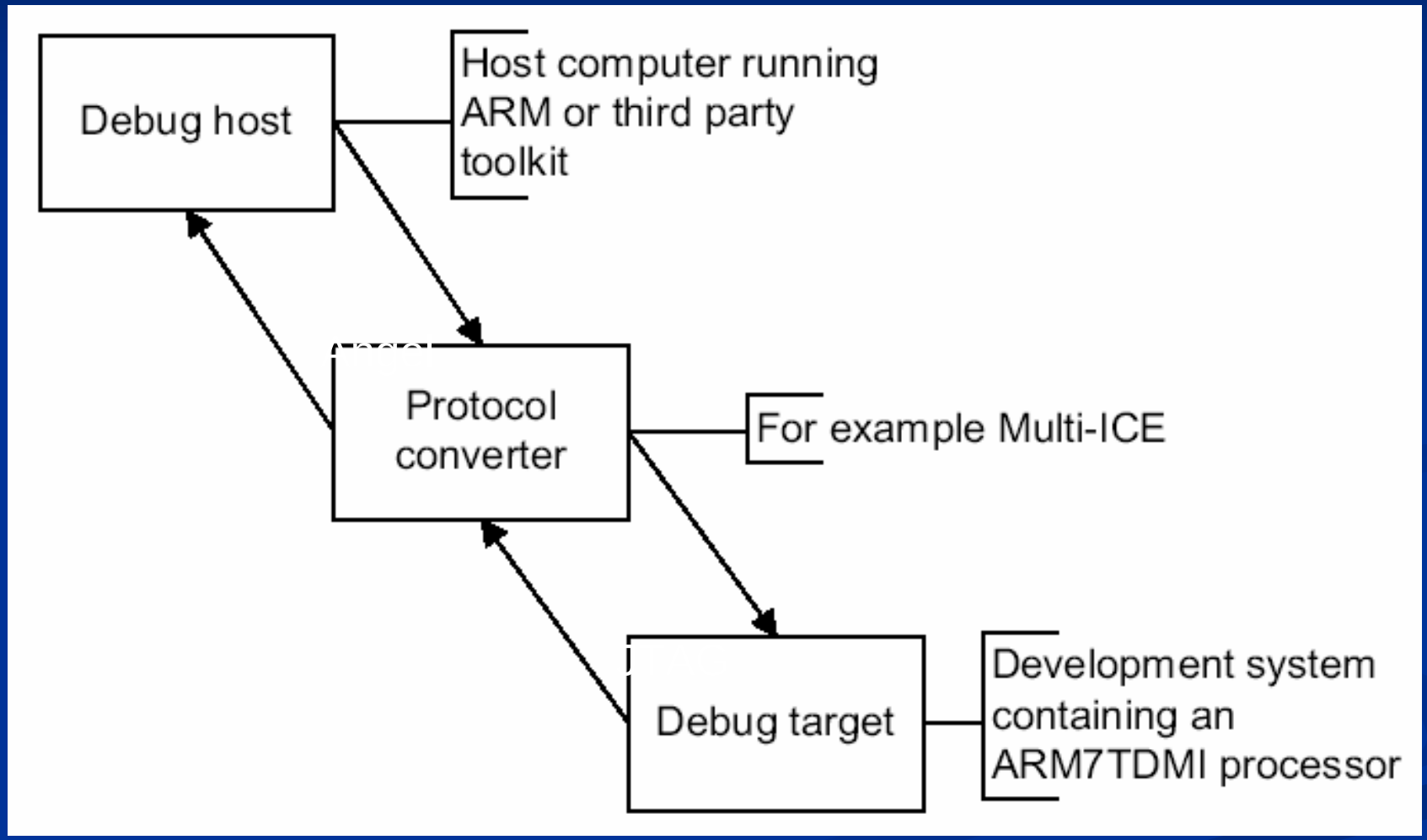
- AHB（Advanced High-performance Bus）：先进高性能总线
- ASB（Advanced System Bus）：先进系统总线
- APB（Advanced Peripheral Bus）：先进外围总线

# 2.5 ARM JTAG调试接口

几种常用的调试方法：

- 指令集模拟器：一种利用PC机端的仿真开发软件模拟调试的方法。
- 驻留监控软件：驻留监控程序运行在目标板上，PC机端调试软件可通过并口、串口、网口与之交互，以完成程序执行、存储器及寄存器读写、断点设置等任务。
- JTAG仿真器：通过ARM芯片的JTAG边界扫描口与ARM核进行通信，不占用目标板的资源，是目前使用最广泛的调试手段。
- 在线仿真器：使用仿真头代替目标板上的CPU，可以完全仿真ARM芯片的行为。但结构较复杂，价格昂贵，通常用于ARM硬件开发中

# ARM的JTAG调试结构



# 宿主机调试器

- 宿主机调试器通过固定的协议控制下位机（协议转换器）。比如，SDT中通过Angel协议或者第三方调试器所提供的协议
- 宿主机调试器只发送宏观的命令，比如：
  程序运行、终止。读写内存、ARM寄存器等
- 通讯的介质可以是串口、并口、以太网、USB等

# JTAG与Angel

- JTAG调试：协议转换器解释上位机传送过来的命令，通过JTAG控制ARM执行。
- Angel调试：协议转换器可以直接做为目标板的Firmware的一部分。直接执行从宿主机传送过来的调试命令；并回送相应的数据。
- Angel可以节省专门的JTAG仿真器，但是，它需要软件，或者是嵌入式操作系统的支持，做不到完全的实时仿真。而JTAG仿真是通过硬件和控制ARM的EmbeddedICE实现的，可以做到实时仿真。

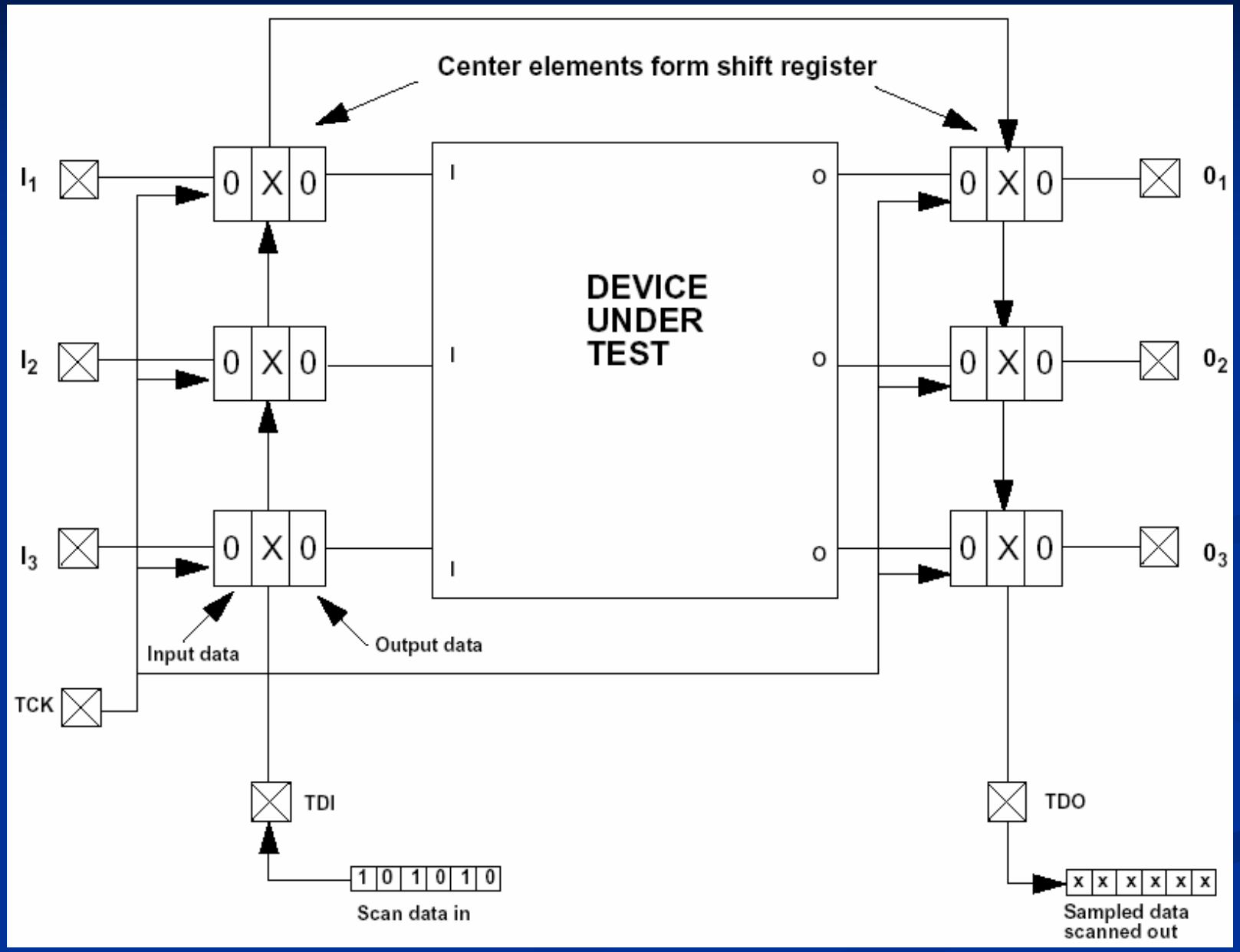
# 什么是JTAG?

- JTAG是Joint Test Action Group的缩写；是IEEE1149.1标准
- JTAG的建立使得集成电路固定在PCB上，只通过边界扫描便可以被测试
- 在ARM处理器中，可以通过JTAG直接控制ARM的内部总线，IO口等信息，从而达到调试的目的

# JTAG的典型接口

- TMS：测试模式选择（Test Mode Select），通过TMS信号控制JTAG状态机的状态
- TCK：JTAG的时钟信号
- TDI：数据输入信号
- TDO：数据输出信号
- nTRST：JTAG复位信号，复位JTAG的状态机和内部的宏单元（Macrocell）

# JTAG链的组成


Center elements form shift register
I1
I2
I3
0 X 0
DEVICE
UNDER
TEST
I
O
01
02
03
Input data
Output data
TCK
TDI
TDO
1 0 1 0 1 0
Scan data in
x x x x x x
Sampled data
scanned out