



中国科学技术大学
University of Science and Technology of China
信息科学技术学院

第7章 ARM程序设计

7.1 ARM程序开发环境

7.2 ARM汇编程序中的伪指令

7.3 ARM汇编语言程序设计

7.4 ARM汇编语言与C/C++的混合编程

本章内容

▣ 7.1 ARM程序开发环境

- 7.1.1 常用ARM程序开发环境简介
- 7.1.2 MDK开发环境简介

▣ 7.2 ARM汇编程序中的伪指令

- 7.2.1 符号定义(Symbol Definition)伪指令
- 7.2.2 数据定义(Data Definition)伪指令
- 7.2.3 汇编控制(Assembly Control)伪指令
- 7.2.4 其他常用的伪指令
- 7.2.5 汇编语言中常用的符号
- 7.2.6 汇编语言中常用运算符和表达式

▣ 7.3 ARM汇编语言程序设计

- 7.3.1 ARM汇编语言的语句格式
- 7.3.2 ARM汇编语言程序结构
- 7.3.3 ARM汇编程序设计实例

▣ 7.4 ARM汇编语言与C/C++的混合编程

- 7.4.1 C语言与汇编语言之间的函数调用
- 7.4.2 C/C++语言与汇编语言的混合编程
- 7.4.3 C编程中访问特殊寄存器的指令

ARM程序设计

- 目前基于**ARM**处理器的程序大多采用**C**语言开发
 - 无操作系统(while语句无限循环体内功能实现+中断)
 - 有操作系统 (μC/OS、Linux、Android 进行任务调度)

- 特定场合下必须使用汇编语言 (系统启动程序)
 - 汇编指令可以直接对ARM处理器中的寄存器进行操作, 掌握必要的汇编程序设计知识, 就能更全面、更深入地理解ARM处理器的工作原理

ARM程序开发环境

- ❑ 用户选用ARM处理器开发嵌入式系统时，选择合适的开发工具可以加快开发进度，节省开发成本。
- ❑ 一套含有编辑软件、编译软件、汇编软件、链接软件、调试软件、工程管理及函数库的集成开发环境（**IDE**）一般来说是必不可少的。
 - 集成开发环境（**Integrated Development Environment**）是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具。集成了代码编写功能、分析功能、编译功能、调试功能等**一体化的开发软件套件**。如微软的Visual Studio系列，谷歌的Android Studio等。
 - 本章仅涉及针对ARM处理器的IDE。
- ❑ 嵌入式实时操作系统、评估板等其他开发工具则可以根据应用软件规模和开发计划选用。

本章内容

□ 7.1 ARM程序开发环境

○ 7.1.1 常用ARM程序开发环境简介

○ 7.1.2 MDK开发环境简介

□ 7.2 ARM汇编程序中的伪指令

○ 7.2.1 符号定义(Symbol Definition)伪指令

○ 7.2.2 数据定义(Data Definition)伪指令

○ 7.2.3 汇编控制(Assembly Control)伪指令

○ 7.2.4 其他常用的伪指令

○ 7.2.5 汇编语言中常用的符号

○ 7.2.6 汇编语言中常用运算符和表达式

□ 7.3 ARM汇编语言程序设计

□ 7.4 ARM汇编语言与C/C++的混合编程

7.1.1 常用ARM程序开发环境

□ 基于Windows平台

目前已停止更新

- SDT (ARM Software Development Kit)是ARM公司最早推出的开发工具
- ADS (ARM Developer Suite)由ARM公司约1999年推出，用来代替SDT
- RealView Developer Suite (RVDS) 是ARM公司继ADS之后推出的IDE
- RealView MDK (Microcontroller Development Kit)
- ARM Development Studio 5

□ 基于Linux平台

- ARM-Linux-GCC

□ 其他的分类方法

- 开源软件，典型如ARM-Linux-GCC
- 商业软件，ARM公司推出的软件工具

ARM-Linux-GCC

- **GNU Compiler Collection (GCC)** 是一套由**GNU**开发的编译器集，不仅支持C语言编译，还支持C++、Ada、Object C等许多语言。GCC还支持多种处理器架构，包括X86、ARM、和MIPS等处理器架构，是在Linux平台下被广泛使用的软件开发工具。
- **GNU**：是“GNU is Not Unix”的递归缩写，是一个自由软件工程项目。这些软件在GNU通用公共许可的保护下允许任何人免费使用和传播（但必须同时提供源程序），GNU软件许可相当宽松，有很多公司利用GNU软件进行商业活动。

题外话：开源许可证

开源许可证教程（阮一峰）

www.ruanyifeng.com/blog/2017/10/open-source-license-tutorial.html

开源许可证都有什么区别,一般开源项目用什么许可证?

<https://www.zhihu.com/question/28292322>



- ❑ “free software”中 “free” 关乎自由，而不是价格。您可以付费或不付费得到GNU软件。
- ❑ 无论如何，一旦得到了软件，您便拥有了使用它的四项特定自由。
- ❑ 有自由按照自己的意愿运行该软件；
- ❑ 有自由复制软件并将其送给您的朋友和同事；
- ❑ 有自由通过对源代码的完全控制而改进程序；
- ❑ 有自由发布改进的版本从而帮助社区建设。

GPL, GNU General Public License



ARM-Linux-GCC

- ❑ **ARM-Linux-GCC**是基于ARM目标机的交叉编译软件，所谓交叉编译简单来说就是在一个平台上生成另一个平台上的可执行代码。
 - 平台实际上包含两个概念：体系结构（Architecture）和操作系统（OS）
 - 同一个体系结构可以运行不同的操作系统；同样，同一个操作系统也可以在不同的体系结构上运行。
 - 一个常见的例子是：嵌入式软件开发人员通常在个人计算机上为运行在基于ARM、PowerPC或MIPS的目标机编译软件。
- ❑ **ARM-Linux-GCC**使用命令行来调用命令执行
 - 相比于RVDS和MDK等IDE而言上手较难。但由于ARM-Linux-GCC不需要授权费用，因而受到使用Linux开发嵌入式系统应用工程师的欢迎。

目前ARM官方提供的软件工具

Embedded Tools and Software

❑ Keil MDK

- Software development package for Arm-based microcontrollers

❑ Arm Development Studio

- Software development tool suite for any Arm-based project

❑ Compiler

- Embedded C/C++ toolchain, from Armv6 M to Armv8-A 64-bit

❑ Keil RTX5

- Real-time operating system implementing the CMSIS-RTOS APIv2

❑ Software Test Libraries

- Efficient processor specific test suites enabling on-line processor testing

❑ FuSa RTS

- Run-time system for embedded functional safety applications

} IDE

MDK (Microcontroller Development Kit)

□ MDK由Keil公司推出

- Keil公司是一家业界领先的MCU软件开发工具的独立供应商，最流行的单片机开发工具Keil C51就是Keil公司出品的。

□ MDK主要特点：

- （1）支持内核： ARM7， ARM9， Cortex-M4/M3/M1， Cortex-R0/R3/R4等ARM微控制器内核，后续可能变化；
- （2）IDE： μ Vision IDE；
- （3）编译器： ARM C/C++ 编译器(armcc)；
- （4）仿真器： μ Vision CPU & Peripheral Simulation；
- （5）硬件调试单元： ULink /JLink。

MDK和DS-5

- ❑ 2005年Keil公司被ARM公司收购之后，ARM公司的开发工具从此分为两大分支：**MDK**系列和**RVDS**系列。
- ❑ MDK系列是ARM 公司推荐的针对微控制器，或者基于单核ARMTDMI、Cortex-M或者Cortex-R处理器的开发工具链，基于Keil公司一直使用的 μ Vision集成开发环境
- ❑ RVDS系列（后升级为DS-5）包含全部功能，支持所有ARM内核。

ARM Development Studio 5 (DS-5)

□ DS-5是一款支持开发所有ARM内核芯片的IDE，主要特点如下：

- (1) 支持内核： 全部；
- (2) 定制的Eclipse IDE；
- (3) 编译器： ARM Compiler 6、ARM Compiler 5、GCC（Linaro GNU GCC Compiler for Linux）；
- (4) 调试器： DS-5调试器支持ETM 指令和数据跟踪、PTM程序跟踪；
- (5) 仿真器： DS-5支持ULINK2、ULINKPRO和DSTREAM仿真器。

MDK和DS-5对比

□ 不同的应用领域

- MDK可满足开发者基于ARM7/9, ARM Cortex-M处理器的开发需求, 包括它自带的RTX实时操作系统和中间库, 都是属于MCU应用领域的。
- DS-5是用于创建Linux/Android的复杂嵌入式系统应用和系统平台驱动接口, DS-5支持设备添加, 包括多核调试和支持, 主要针对复杂的多核调试、片上系统开发而推出的。

□ 使用经验

- 如果需要做MCU应用, 推荐用KEIL MDK;
- 如果需要做片上系统、Linux/Android驱动和应用开发, 推荐使用DS-5+DSTREAM。
- 用户可以根据自己的功能需求、使用习惯(比如很多从单片机开发转到嵌入式开发的开发者更习惯使用MDK)、开发用途等选择不同的开发环境。

本章内容

□ 7.1 ARM程序开发环境

- 7.1.1 常用ARM程序开发环境简介

- 7.1.2 MDK开发环境简介

□ 7.2 ARM汇编程序中的伪指令

- 7.2.1 符号定义(Symbol Definition)伪指令

- 7.2.2 数据定义(Data Definition)伪指令

- 7.2.3 汇编控制(Assembly Control)伪指令

- 7.2.4 其他常用的伪指令

- 7.2.5 汇编语言中常用的符号

- 7.2.6 汇编语言中常用运算符和表达式

□ 7.3 ARM汇编语言程序设计

□ 7.4 ARM汇编语言与C/C++的混合编程

MDK-ARM

- ❑ **MDK-ARM**, KEIL提供了包括C编译器、宏汇编、连接器、库管理和一个功能强大的仿真调试器在内的完整开发方案, 通过一个**IDE (μVision)**将这些功能组合在一起, 界面和常用的微软VC++的界面相似, 在调试程序、软件仿真方面也有很强大的功能。

MDK-Lite

Product evaluation, small projects, and education. Code size restricted to 32 Kbyte.

[Learn more >](#)


 [Download & Install](#)

MDK-Essential

For Arm Cortex-M based microcontroller projects.

[Learn more >](#)

 [Request a Quote](#)


 [Buy online](#)

MDK-Plus

For Cortex-M, Arm7, Arm9.
Includes middleware (IPv4 Networking, USB Device, File System, Graphics).

[Learn more >](#)

 [Request a Quote](#)


 [Buy online](#)

MDK-Professional

For Cortex-M, Arm7, Arm9.
Includes middleware (IPv4/IPv6 Networking, USB Host & Device, File System, Graphics, mbed components).

[Learn more >](#)

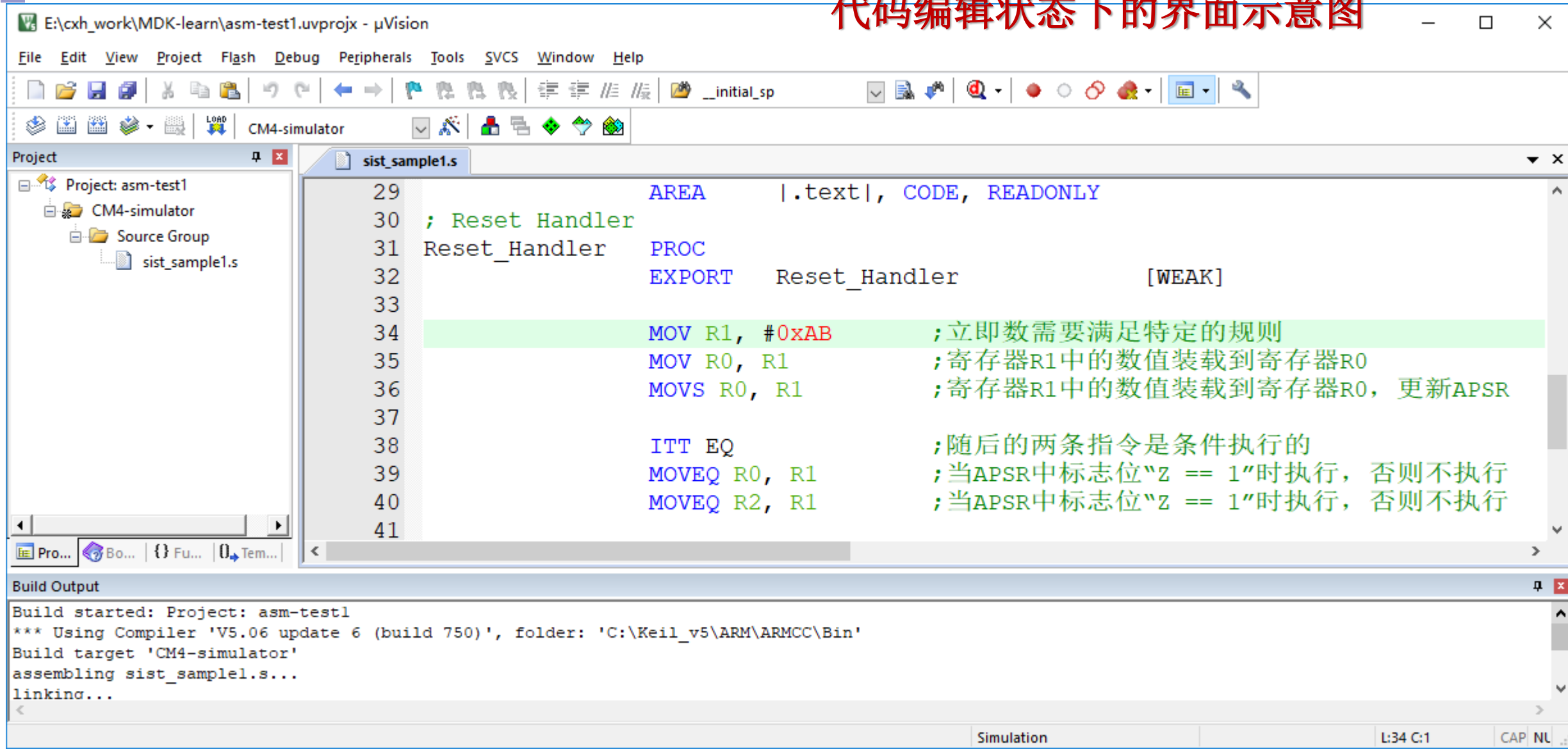
 [Request a Quote](#)

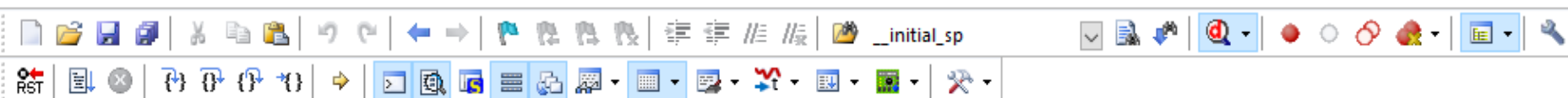
 [Buy online](#)

µVision IDE

µVision IDE是一款集编辑，编译和项目管理于一身的可视化开发环境，集成了C语言编译器，宏编译，链接/定位，以及HEX文件产生器。

代码编辑状态下的界面示意图





Registers

Register	Value
Core	
R0	0x000000AB
R1	0x000000AB
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000406
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	2

Disassembly

```
0x00000404 4608 MOV r0,r1          MOVs R0, R1          ;寄存器R1中的数值装载到寄存器R0, 更新APSR
36:
37:
→ 0x00000406 0008 MOVs r0,r1
38: ITT EQ          ;随后的两条指令是条件执行的
0x00000408 BF04 ITT EQ          ;当APSR中标志位"Z == 1"时执行, 否则不执行
```

sist_sample1.s

```
32 EXPORT Reset_Handler [WEAK]
33
34 MOV R1, #0xAB          ;立即数需要满足特定的规则
35 MOV R0, R1             ;寄存器R1中的数值装载到寄存器R0
36 MOVS R0, R1            ;寄存器R1中的数值装载到寄存器R0, 更新APSR
37
38 ITT EQ                 ;随后的两条指令是条件执行的
39 MOVEQ R0, R1           ;当APSR中标志位"Z == 1"时执行, 否则不执行
40 MOVEQ R2, R1           ;当APSR中标志位"Z == 1"时执行, 否则不执行
41
--- --                  ;随后的两条指令是条件执行的
```

Command

Load "E:\cxh_work\MDK-learn\Objects\asm-test1.axf"

Memory 1

Address: 0x00000400

```
0x00000400: 4F F0 AB 01 08 46 08 00 04 BF 08 46 0A 46 0C BF 08 46 1A 46 00 BF F3
0x00000417: E7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000042E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000445: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE

Call Stack + Locals Memory 1

Simulation

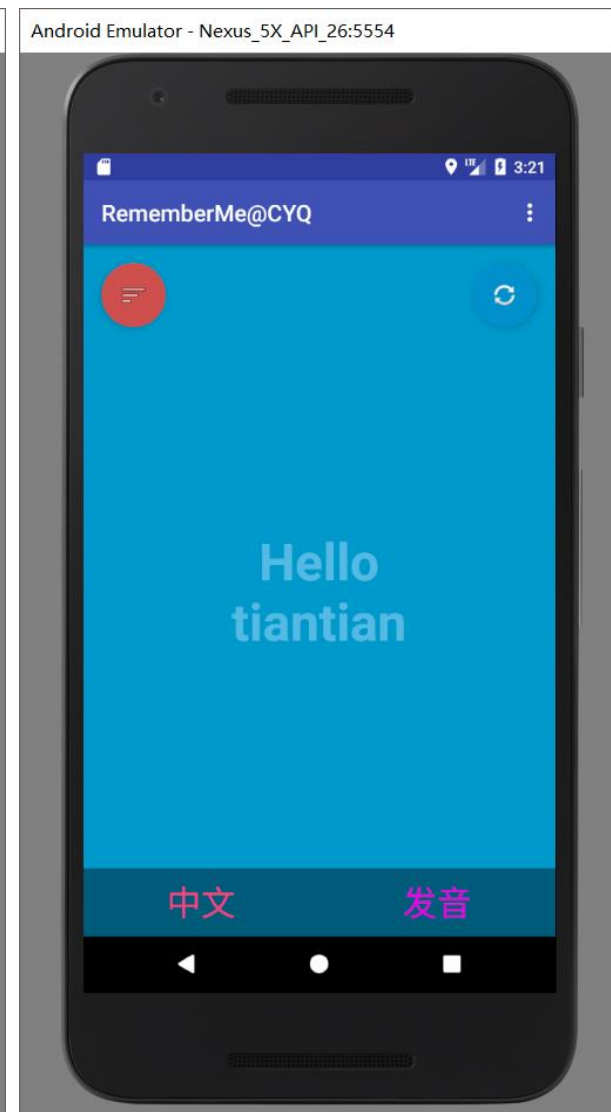
t1: 0.00000017 sec

L:38 C:1

CAP NUM SCRL OVR R/W

- 调试是任何项目开发过程中必不可少的一部分，特别是在软硬件结合非常紧密的嵌入式系统开发中。一般来说，大多数的调试工作是在**RAM**中进行的，只有当程序完成并能运行后才切换到**ROM**上。
- 嵌入式系统的调试有多种方法，可分为
 - 模拟器方式
 - ICE（In-Circuit Emulator，在线仿真器）方式
 - ICD（In-Circuit Debugger，在线调试器）方式
 - 监控器方式

模拟器方式示例：Android Studio的手机模拟器



- 主机和目标板通过某种接口（通常是串口）连接，主机上提供调试界面，被调程序下载到目标板上运行
 - 监控程序是一段运行于目标机上的可执行程序，主要负责监控目标机上被调试程序的运行情况，与宿主机端的调试器一起完成对应用程序的调试。监控程序包含基本功能的启动代码，并完成必要的硬件初始化，等待宿主机的命令。被调试程序通过监控程序下载到目标机，就可以开始进行调试。
 - 监控器调试主要用于调试运行在目标机操作系统上的应用程序，不适宜用来调试目标操作系统。
- 例：单片机与上位机的通信
 - 采用单片机实现的心电信号的采集和处理；上位机（PC机）接收AD数据并进行图形化显示。

- 在线仿真器ICE是一种**完全仿造调试目标CPU设计的仪器**，目标系统对用户来说是完全透明的、可控的。仿真器与目标板通过仿真头连接，与主机有串口、并口、以太网口或USB口等连接方式。该仿真器可以真正地运行所有的CPU动作，并且可以在其使用的内存中设置非常多的硬件中断点，可以**实时查看所有需要的数据**，从而给调试过程带来很多便利。由于仿真器自成体系，调试时可以连接目标板，也可以不接目标板。
- 使用ICE同使用一般的目标硬件一样，只是在ICE上完成调试后，需要把调试好的程序重新下载到目标系统上而已。由于ICE价格昂贵，而且**每种CPU都需要一种与之对应的ICE**，使得开发成本非常高。

- 使用ICD和目标板的调试端口连接，发送调试命令和接收调试信息，可以完成必要的调试功能。一般情况下，在**ARM芯片的开发板上采用JTAG边界扫描口进行调试**。摩托罗拉公司采用专用的BDM调试接口。
- 使用合适的开发工具可以利用这些接口。例如，ARM开发板，可以将JTAG调试器接在开发板的JTAG口上，通过JTAG口与ARM处理器核进行通信。由于JTAG调试的目标程序是在目标板上执行，仿真更接近于目标硬件，因此许多接口问题，如高频操作限制、电线长度的限制等被最小化了。**该方式是目前采用最多的一种调试方式。**

ARM调试器（中文资料习惯译作仿真器）

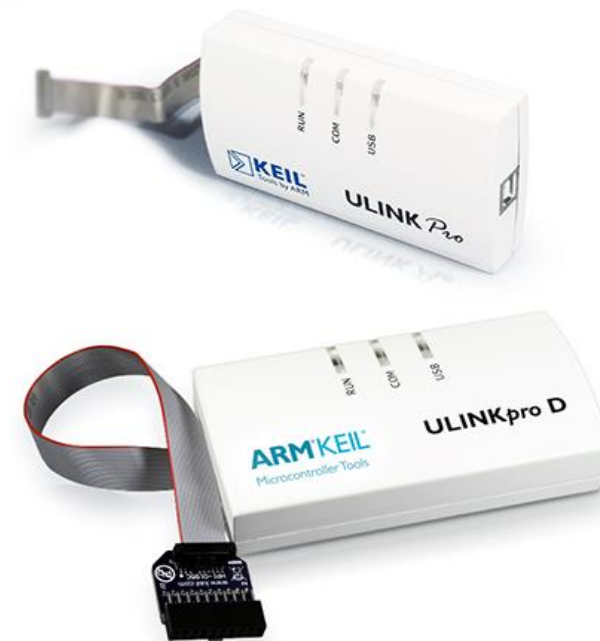


DSTREAM Family

Maximum visibility into Arm processors, with up to **19.2 Gbps** parallel trace over 32 pins, streams trace data directly to host PC, system autodetection with **Development Studio** and a range of target connectors including JTAG, MICTOR, and CoreSight.

ULINK Family

JTAG or SWD debug connections to Arm-based microcontrollers running at clock speeds of up to 200 MHz. ULINKpro allows ETM trace capture at up to **800 Mbps** and streaming trace direct to **Keil MDK**.

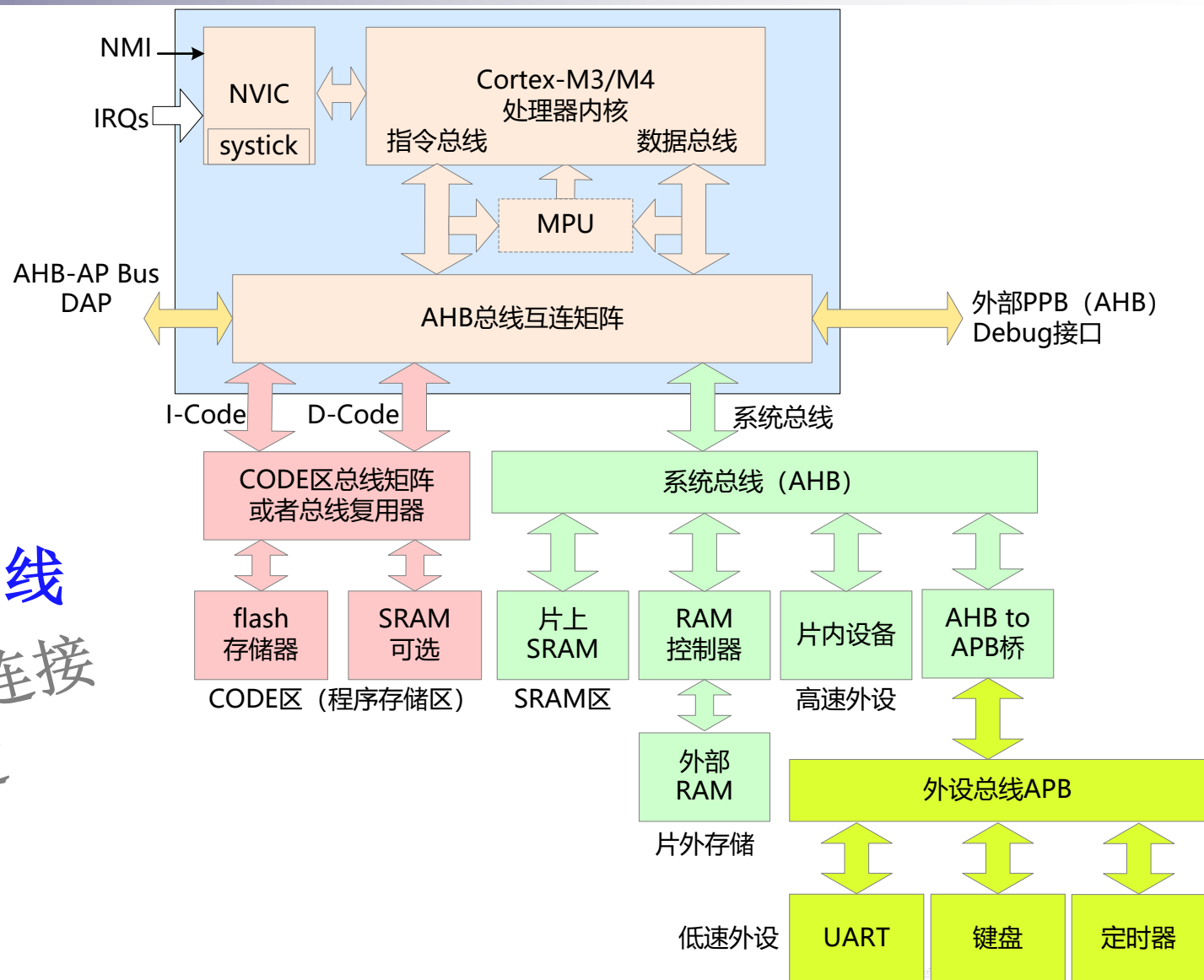


CMSIS-DAP

For **simple debug over driverless USB**, CMSIS-DAP provides an abstraction of the CoreSight Debug Access Port command set. The JTAG/SWD connection is incorporated into the board-level layout.

换个视角：不同功能模块的连接总线

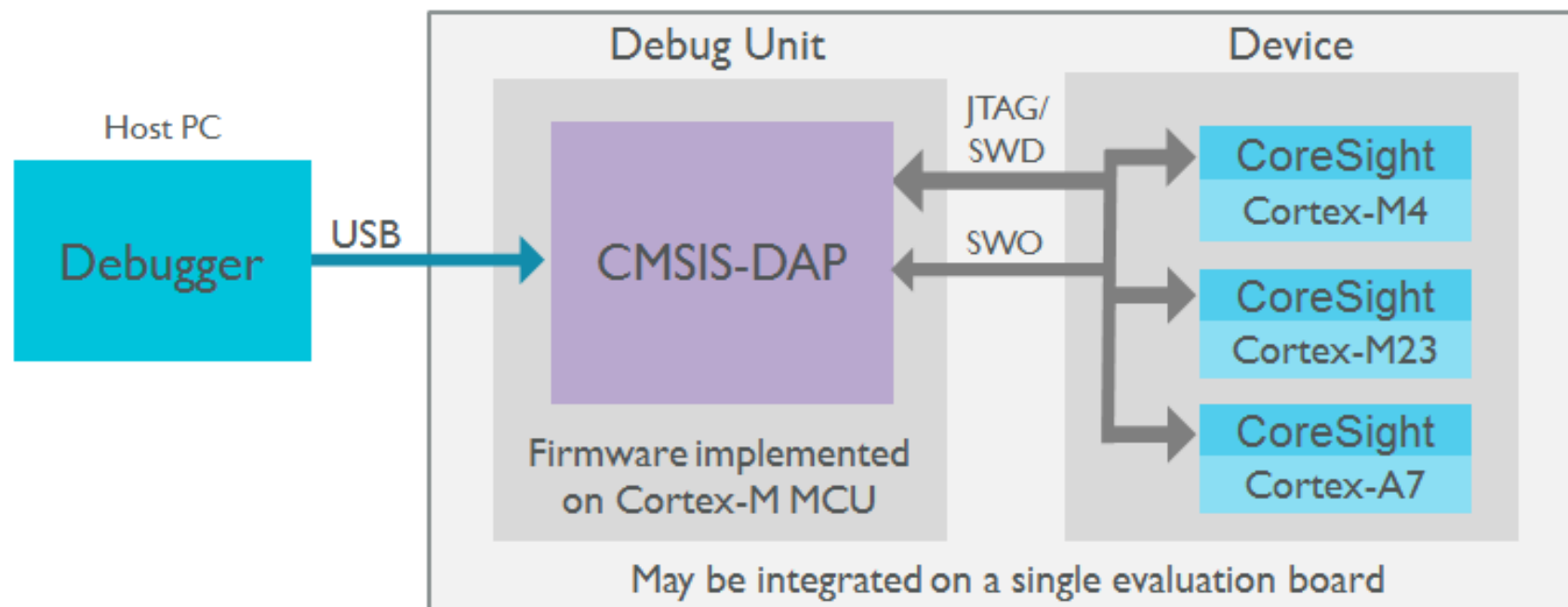
- I-Code总线
- D-Code总线
- 系统总线AHB Lite
- 外设总线APB
- 私有外设总线PPB
- 调试访问端口（DAP）总线



此页PPT在第5章，5.4.2 连接
存储器和外设 部分出现过

最简单的调试场景 CMSIS-DAP

- A Device provides a Debug Access Port (DAP) typically either with a 5-pin JTAG or with a 2-pin Serial Wired Debug (SWD) interface that connects to a Debug Unit. CMSIS-DAP is the interface firmware for a Debug Unit that connects the Debug Port to USB. Debuggers that execute on a host computer connect via USB and the Debug Unit to the Device which runs the application software.



目前ARM公司的Debug Probes

□ DSTREAM系列，高端的ARM处理器如Cortex-A



DSTREAM



DSTREAM-ST



DSTREAM-PT



DSTREAM-HT

External parallel trace 9.6 Gb/s (up to 16 pins)

Maximum number of supported cores 128

2.4 Gb/s (up to 4 pins)

1022

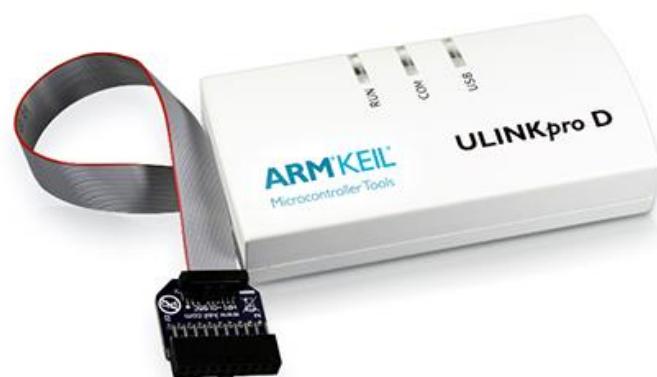
19.2 Gb/s (up to 32 pins)

1022

2.4 Gb/s (up to 4 pins)

1022

□ ULINK系列，用于低端嵌入式微控制器



Debug Probes Comparison

Debug probe family	DSTREAM family	ULINKpro family
Probe types	DSTREAM, DSTREAM-ST, DSTREAM-PT, DSTREAM-HT	ULINKpro, ULINKproD
Functionality	Debug, CoreSight trace	Debug, CoreSight trace
Target type	Hardware	Hardware
Supported target processors	All Arm Cortex and Neoverse processors	Cortex-A5, A8, A9, R4, R5, any Cortex-M
Arm7, Arm9, Arm11	All	Arm 7, Arm9
Maximum number of cores	128 – 1022	64
Debug connectivity	JTAG, SWD	JTAG, SWD
JTAG frequency	60 – 180 Mhz	50
Memory download	2.5 – 12 MB/s	3
External parallel trace	4 – 32 pins	Up to 4 pins
External serial trace	20 – 60 Gb/s	800 Mb/s
Trace buffer	Up to 8 GB or streamed	–
Host connectivity	USB 2.0, Ethernet or USB 3.0, Gigabit Ethernet	USB 2.0
Software requirements	Arm Development Studio (DSTREAM, DSTREAM-ST, DSTREAM-PT DSTREAM-HT) DS-5 (DSTREAM, DSTREAM-ST only)	Arm Development Studio, Keil MDK, DS-5

其他公司的仿真器

□ ARM仿真器

- JTAG仿真器也称为JTAG调试器，是通过ARM芯片的JTAG边界扫描口进行调试的设备。JTAG仿真器比较便宜，连接比较方便，而且JTAG调试的目标程序是在目标板上执行，仿真更接近于目标硬件。
- **JLink**是SEGGER公司支持仿真ARM内核芯片推出的JTAG仿真器。支持所有ARM7/ARM9/ARM11，Cortex M0/M1/M3/M4，Cortex A5/A8/A9等内核芯片的仿真，与Keil MDK等编译环境无缝连接。

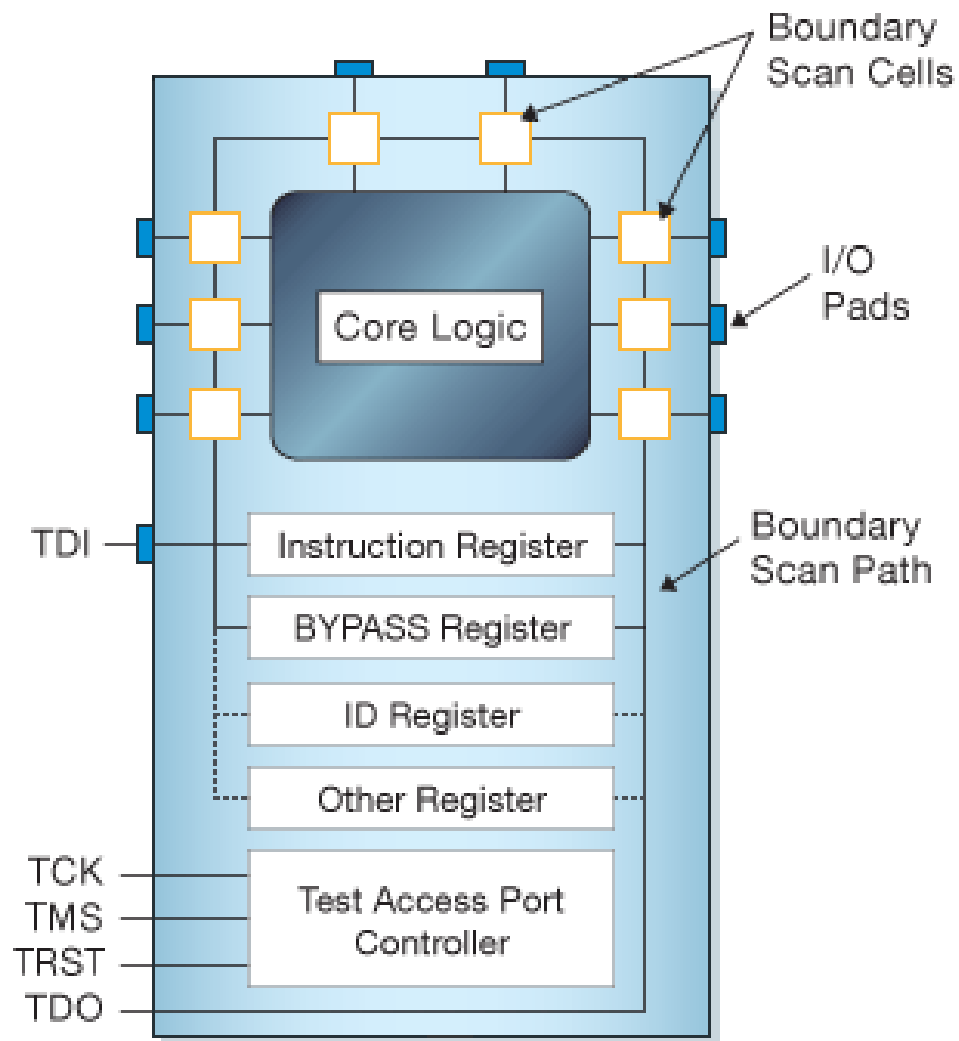


- JTAG(Joint Test Action Group) 组织成立于1985 年，是由几家主要的电子制造商发起制订的PCB 和IC 测试标准。JTAG 建议1990 年被IEEE 批准为**IEEE1149.1** 测试访问端口和边界扫描结构标准。
- 标准的JTAG接口是**4线：TMS、TCK、TDI、TDO**,分别为模式选择、时钟、数据输入和数据输出线。
 - JTAG最初是用来对芯片进行测试的，基本原理是在器件内部定义一个测试访问口，通过专用JTAG测试工具对内部节点进行测试。测试允许多个器件通过JTAG接口串联在一起，形成一个JTAG链，能实现对各个器件分别测试。
 - 现在，JTAG接口还常用于实现ISP（In-System Programmable;在线编程），对FLASH等器件进行编程。

IEEE 1149.1 (1990)

JTAG TAP Interface Signals

Abbreviation	Signal	Description
TCK	Test Clock	Synchronizes the internal state machine operations
TMS	Test Mode State	Sampled at the rising edge of TCK to determine the next state
TDI	Test Data In	Represents the data shifted into the device's test or programming logic. It is sampled at the rising edge of TCK when the internal state machine is in the correct state.
TDO	Test Data Out	Represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal state machine is in the correct state
TRST	Test Reset	An optional pin which, when available, can reset the TAP controller's state machine



遵循IEEE 1149.1标准的芯片，除了芯片本身的功能电路（Core Logic）之外，会在芯片中放置边界扫描单元电路（**Boundary Scan Cells**）。通过TDI引脚向芯片输入指令，TDO引脚将会输出芯片内的相关信息。

小结：ARM程序开发环境

❑ RealView **MDK** (Microcontroller Development Kit)

- μ Vision IDE

❑ ARM Development Studio 5

❑ ARM-Linux-GCC

❑ Debugger/Debug Probes

- DSTREAM Family

- ULINK Family

- CMSIS-DAP

- SEGGER公司的JLink

μVision IDE演示

□ 代码编辑界面

- 新建project
- 设置project的option

□ 调试界面

- 观察寄存器
- 观察存储器
- 观察栈

□ 界面控制

- Edit→Configuration



<https://www.eeo.cn/live.php?lessonKey=610aa3470c0e38a9>

From 11:44 to 42:33

本章内容

□ 7.1 ARM程序开发环境

- 7.1.1 常用ARM程序开发环境简介
- 7.1.2 MDK开发环境简介


□ 7.2 ARM汇编程序中的伪指令

- 7.2.1 符号定义(Symbol Definition)伪指令
- 7.2.2 数据定义(Data Definition)伪指令
- 7.2.3 汇编控制(Assembly Control)伪指令
- 7.2.4 其他常用的伪指令
- 7.2.5 汇编语言中常用的符号
- 7.2.6 汇编语言中常用运算符和表达式


□ 7.3 ARM汇编语言程序设计


□ 7.4 ARM汇编语言与C/C++的混合编程

ARM汇编的资料来源

 D:\cxh_code\MDK\class_show\sample.uvprojx - µVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

 µVision Help

 Open Books Window

□ ARM Compiler 5 User's Guides

○ ARM Compiler 5 toolchain

- ARM and Thumb C and C++ compiler, armcc
- ARM and Thumb assembler, armasm
- ARM linker, armlink
- ARM librarian, armar
- ARM image conversion utility, fromelf
- supporting libraries ←浮点运算的库













□ ARM Compiler 6 User's Guides

○ 提供对ARMv8的支持

 ARM Development Tools

 隐藏  查找  上一步  前进  打印  选项(O)

目录(C) 索引(N) 搜索(S) 收藏夹(I)

-  Arm Development Tools
-  µVision® User's Guide
-  **Arm Compiler 5 User's Guides**
-  Arm Compiler 6 User's Guides
 -  Compiler Getting Started Guide
 -  Migration and Compatibility Guide
 -  Compiler Reference Guide
 -  Errors and Warnings Reference Guide
 -  Libraries and Floating-Point Support User Guide
-  Debugger Adapter User's Guides
-  Getting Assistance
-  Licensing User's Guide

ARM汇编代码行的格式

{symbol} {instruction | directive | pseudo-instruction} {;comment}

□ symbol

- 通常就是标号，在一些directive中有时为常数或者变量 ;代码中要顶格写

□ Instruction | directive | pseudo-instruction

- **Instruction**，即ARM或Thumb**指令**
- **pseudo-instructions**，ARM汇编器支持少数几条**伪指令**，伪指令在编译的时候被翻译为一组ARM或Thumb指令
- **directive**，指示符，指示ARM汇编器行为，中文资料中**习惯翻译为伪指令或伪操作**

□ comment，注释

LDR pseudo-instruction

□ 语法

- LDR{cond}{.W} Rt, =expr
- LDR{cond}{.W} Rt, =label_expr
 - cond 是条件码，.W 是操作数宽度（可选），Rt是标寄存器
 - expr 是拟加载的数值，label_expr 是标号或标号加减一个数值

□ 示例

LDR r1,=0xff ; 加载立即数0xff至R1

□ 被翻译为

LDR r1, [pc,offset_to_litpool]

...

litpool DCD 0xff

E:\cxh_work\MDK-learn\sample2\sample2.uvprojx - μVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Registers

Register	Value
Core	
R0	0x12345678
R1	0x00000001
R2	0x00000003
R3	0x00000003
R4	0x00000001
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000043C
xPSR	0xA1000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	27

Disassembly

```
62:      POP {R4} ;从堆栈弹出至R4
63:
0x00000438 BC10      POP      {r4}
64:      LDR R0, =0x12345678 ;把0x12345678装载到R0
0x0000043A 4803      LDR     r0, [pc, #12] ; @0x00000448
65:      MOV R1, #0xABABABAB ;立即数需要满足特定的规则
0x0000043C F04F31AB      MOV     r1, #0xABABABAB
```

ustc_class_show_directive.s

```
62      POP {R4} ;从堆栈弹出至R4
63
64      LDR R0, =0x12345678 ;把0x12345678装载到R0
65      MOV R1, #0xABABABAB ;立即数需要满足特定的规则
66      STR R1, [R0]
67      LDR R2, [R0]
68
69      NOP
70      B Reset_Handler
71      ENDP
```

Command

```
Load "E:\cxh_work\MDK-learn\sample2\Objects\sample2.axf"
BS \sample2\ustc_class_show_directive.s\64
```

Memory 1

Address: 0x00000400

0x00000400:	4F F0 AB 01 08 46 08 00 04 BF 08 46 0A 46 4F F0 FF 03 4F EA 03
0x00000415:	63 4F EA 43 03 5B 00 4F F0 34 01 4F F0 EF 03 2C BF 08 46 1A 46
0x0000042A:	4F F0 01 01 4F F0 03 03 02 B4 08 B4 04 BC 10 BC 03 48 4F F0 AB
0x0000043F:	31 01 60 02 68 00 BF DB E7 78 56 34 12 00 00 00 00 00 00 00 00
0x00000454:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess

Simulation t1: 0.00000225 sec L:65 C:1 CAP NUM SCRL

观察LDR伪指令解析后的结果

ARM Compiler支持的Pseudo-instructions

- *ADRL pseudo-instruction*
 - 加载PC相对寻址或寄存器间接寻址的地址至目标寄存器
- *MOV32 pseudo--instruction*
 - 加载32-bit的立即数或地址至目标寄存器
- *LDR pseudo-instruction*
 - 加载32-bit的立即数或地址至目标寄存器
- *UND pseudo-instruction*
 - 生成一条未定义指令
- *CPY pseudo-instruction ; ARM Compiler 6*
 - 拷贝一个寄存器内容至另一个寄存器
- *NEG pseudo-instruction ; ARM Compiler 6*
 - 取反

ARM Compiler支持的directives

ALIAS	DCFS/DCFSU	FRAME POP	INCLUDE see GET	REQUIRE8/PRESERVE8
ALIGN	DCI	FRAME PUSH	INFO	RLIST
ARM or CODE32	DCW/DCWU	FRAME REGISTER	KEEP	RN
AREA	DN	FRAME RESTORE	LCLA, LCLL,/LCLS	ROUT
ASSERT	ELIF, ELSE (see IF)	FRAME SAVE	LTORG	SETA, SETL,/SETS
ATTR	END	FRAME STATE REMEMBER	MACRO/MEND	SN
CN	ENDFUNC or ENDP	FRAME STATE RESTORE	MAP	SPACE or FILL
CODE16	ENDIF (see IF)	FRAME UNWIND ON or OFF	MEND (see MACRO)	SUBT
COMMON	ENTRY	FUNCTION or PROC	MEXIT	THUMB
CP	EQU	GBLA, GBLL,/GBLS	NOFP	THUMBX
DATA	EXPORT or GLOBAL	GET or INCLUDE	OPT	TTL
DCB	EXPORTAS	GLOBAL (see EXPORT)	PRESERVE8 (see REQUIRE8)	WHILE/WEND
DCD/DCDU	EXTERN	IF, ELSE, ENDIF,/ELIF	PROC see FUNCTION	
DCDO	FIELD	IMPORT	RELOC	
DCFD/DCFDU	FRAME ADDRESS	INCBIN	REQUIRE	

ARM汇编程序中的伪指令

- 由于ARM Compiler支持的Pseudo-instructions数目非常有限，本节内容仅限于directives，且沿用中文资料习惯翻译“伪指令”
- 在ARM汇编语言程序里，有一些特殊指令助记符，这些助记符与指令系统的助记符不同，没有相对应的操作码，通常称这些特殊指令助记符为伪指令，他们所完成的操作称为伪操作。
- 伪指令不像机器指令那样在处理器运行期间由机器执行，而是汇编程序对源程序汇编期间由汇编程序处理，包括：定义变量、分配数据存储空间、控制汇编过程、定义程序入口等，这些伪指令仅在汇编过程中起作用，一旦汇编结束，伪指令的使命就完成了。

7.2.1 符号定义(Symbol Definition)伪指令

- 符号定义伪指令用于定义ARM汇编程序中的变量、对变量赋值以及定义寄存器的别名等操作。常见的符号定义伪指令有如下几种：
 - 用于定义全局变量的GBLA、GBLL和GBLS
 - GBLA用于定义一个全局的**数字变量**，并初始化为0；
 - GBLL用于定义一个全局的**逻辑变量**，并初始化为F（假）；
 - GBLS用于定义一个全局的**字符串变量**，并初始化为空；
 - 用于定义局部变量的LCLA、LCLL和LCLS
 - 用于对变量赋值的SETA、SETL、SETS
 - 为通用寄存器列表定义名称的RLIST

7.2.2 数据定义(Data Definition)伪指令

- 数据定义伪指令一般用于为特定的数据分配存储单元，同时可完成已分配存储单元的初始化。常见的数据定义伪指令有如下几种：
 - DCB 用于分配一段连续的字节存储单元并用指定的数据初始化
 - DCW (DCWU) 用于分配一段连续的半字存储单元并用指定的数据初始化
 - DCD (DCDU) 用于分配一段连续的字存储单元并用指定的数据初始化
 - DCFD (DCFDU) 用于为双精度的浮点数分配一段连续的字存储单元并用指定数据初始化
 - DCFS (DCFSU) 用于为单精度的浮点数分配一段连续的字存储单元并用指定数据初始化
 - DCQ (DCQU) 用于分配一段以8字节为单位的连续的存储单元并用指定的数据初始化
 - SPACE 用于分配一段连续的存储单元
 - MAP 用于定义一个结构化的内存表首地址
 - FIELD 用于定义一个结构化的内存表的数据域

7.2.3 汇编控制(Assembly Control)伪指令

□ 汇编控制伪指令用于控制汇编程序的执行流程：

○ IF、ELSE、ENDIF

○ WHILE、WEND

○ MACRO、MEND

○ MEXIT

IF logical-expression

...;code

{ELSE

...;code}

ENDIF

IF logical-expression

instructions

ELIF logical-expression2

instructions

ELIF logical-expression3

instructions

ENDIF

WHILE logical-expression

code

WEND

MACRO、MEND、MEXIT

□ C语言中的宏定义

- `#define PI 3.1416` //无参数宏定义
- `#define MAX(a,b) ((a)>(b)?(a):(b))` //带参数的宏定义

□ 汇编语言中宏定义

MACRO

{ \$标号 } 宏名 { \$参数1, \$参数2, }

指令序列

MEND

- 如果在宏定义体中使用到了循环，则可使用 **MEXIT** 提前退出宏

7.2.4 其他常用的伪指令

□ AREA

- 用于定义一个代码段或数据段，例如

AREA RESET, CODE, READONLY

;定义了一个代码段，段名为RESET，属性为只读

□ EXPORT（或GLOBAL）

- 用于在程序中声明一个全局的标号，该标号可在其他的文件中引用

□ IMPORT

- 用于通知编译器要使用的标号在其他的源文件中定义

□ END

- 用于通知编译器已经到了源程序的结尾

7.2.5 汇编语言中常用的符号

在汇编语言程序设计中，经常使用各种符号代替地址、变量和常量等，以增加程序的可读性。符号的命名必须遵循以下约定：

- (1) 符号名区分大小写
- (2) 符号名在其作用范围内必须唯一
- (3) 符号名不能与系统的保留字相同
- (4) 符号名不应与指令或伪指令同名

□ 1. 程序中的变量

LCLS S1 ;定义局部字符串变量S1

S1 SETS "Test!" ;字符串变量S1的值为 "Test!"

□ 2. 程序中的常量

NUM EQU 64

abcd EQU 2 ;定义abcd符号的值为2

abcd EQU label+16 ;定义abcd符号的值为(label+16)

□ 3. 程序中的变量代换

LCLS S1 ;定义局部字符串变量S1和S2

LCLS S2

S1 SETS "Test!"

S2 SETS "This is a \$S1" ;字符串变量S2的值为 "This is a Test! "

7.2.6 汇编语言中常用运算符和表达式

□ 1. 数字表达式及运算符

- (1) “+”、“-”、“×”、“/”及“MOD”算术运算符
- (2) “ROL”、“ROR”、“SHL”及“SHR”移位运算符
- (3) “AND”、“OR”、“NOT”及“EOR”按位逻辑运算符

□ 2. 逻辑表达式及运算符

- (1) “=”、“>”、“<”、“>=”、“<= ”、“/=”、“<>”运算符
- (2) “LAND”、“LOR”、“LNOT”及“LEOR”运算符

□ 3. 字符串表达式及运算符

LEN、CHR、STR、LEFT、RIGHT、CC

□ 4. 与寄存器和程序计数器（PC）相关的表达式及运算符

BASE、INDEX

□ 5. 其他常用运算符

- (1) ? 运算符
- (2) DEF运算符

本章内容

- 7.1 ARM程序开发环境
- 7.2 ARM汇编程序中的伪指令
- 7.3 ARM汇编语言程序设计
 - 7.3.1 ARM汇编语言的语句格式
 - 7.3.2 ARM汇编语言程序结构
 - 7.3.3 ARM汇编程序设计实例
- 7.4 ARM汇编语言与C/C++的混合编程
 - 7.4.1 C语言与汇编语言之间的函数调用
 - 7.4.2 C/C++语言与汇编语言的混合编程
 - 7.4.3 C编程中访问特殊寄存器的指令

ARM汇编语言程序设计

- 由于**C**语言便于理解，有大量的支持库，所以它是当前**ARM**程序设计所使用的主要编程语言。
- 但是对硬件系统的初始化、**CPU**状态设定、中断使能、主频设定以及**RAM**控制参数初始化等**C**程序力所不能及的**底层操作**，还是要由汇编语言程序来完成。
 - 注：**C**编译器不一定能够把**C**代码翻译为最佳的机器指令！特别是在一些指令特别丰富的专用处理器中。
- **ARM**程序在完成上述功能时通常是**C**语言和汇编语言混合编程。

7.3.1 ARM汇编语言的语句格式

此页PPT在前面出现过

{symbol} {instruction | directive | pseudo-instruction} {;comment}

语法错误ARM Compiler 会给出提示

□ symbol

- 通常就是标号，在一些directive中有时为常数或者变量 ← 代码中要顶格写

□ Instruction | directive | pseudo-instruction

- ← 如果前面没有symbol，必须有空格或者TAB
- **Instruction**，即ARM或Thumb指令
- **pseudo-instructions**，ARM汇编器支持少数几条伪指令，伪指令在编译的时候被翻译为一组ARM或Thumb指令
- **directive**，指示符，指示ARM汇编器行为

□ comment，注释 ← 以英文符号“;”起为注释

7.3.2 ARM汇编语言程序结构

- 在ARM（Thumb）汇编语言程序中，通常以段为单位来组织代码段是具有特定名称且功能相对独立的指令或数据序列根据段的内容，分为代码段和数据段。
- 一个汇编程序至少应该有一个代码段，当程序较长时，可以分割为多个代码段和数据段。

- 例如，定义一个可读写属性的数据段

AREA Buf, DATA, READWRITE

Num DCD 0x11

Nums DCD 0x22 ;分配一个字存储单元并初始化

;分配一段连续的存储单元用Space (directive)

示例，一个代码段的定义

<pre> AREA RESET, CODE, READONLY ENTRY START LDR R0, = Num LDR R1, [R0] ADD R1, #0x9A ; R1=R1+0x9A STR R1, [R0] LDR R0, = Nums LDR R2, [R0] ADD R2, #0xAB STR R2, [R0] LOOP B LOOP END </pre>	<p>;只读的代码段，RESET为段名</p> <p>;程序入口点</p> <p>;取Num地址赋给R0</p> <p>;取Num中内容赋给R1</p> <p>;R1内容赋给Num单元</p> <p>;.....</p> <p>;无限循环反复执行</p> <p>;段结束</p>
---	---

7.3.3 ARM汇编程序设计实例

□ 1. 顺序结构，最简单的程序结构

- AREA、DCB、ENTRY、END
- LDR、MOV

□ 2. 分支结构，使用带条件码的指令实现

- MOVLT、MOVHI、MOVLS
- CMP、B、BEQ、BL

□ 3. 循环结构，重复执行某段代码

- B、BLS

□ 4. 子程序调用与返回

- BL
- 用栈传递数据：PUSH、POP、STMFD、LDMFD

cond	Mnemonic extension	Meaning, integer arithmetic
0000	EQ	Equal
0001	NE	Not equal
0010	CS ^b	Carry set
0011	CC ^c	Carry clear
0100	MI	Minus, negative
0101	PL	Plus, positive or zero
0110	VS	Overflow
0111	VC	No overflow
1000	HI	Unsigned higher
1001	LS	Unsigned lower or same
1010	GE	Signed greater than or equal
1011	LT	Signed less than
1100	GT	Signed greater than
1101	LE	Signed less than or equal

条件码可参考第6章表6.6

本章内容

- 7.1 ARM程序开发环境
- 7.2 ARM汇编程序中的伪指令
- 7.3 ARM汇编语言程序设计
 - 7.3.1 ARM汇编语言的语句格式
 - 7.3.2 ARM汇编语言程序结构
 - 7.3.3 ARM汇编程序设计实例
- 7.4 ARM汇编语言与C/C++的混合编程
 - 7.4.1 C语言与汇编语言之间的函数调用
 - 7.4.2 C/C++语言与汇编语言的混合编程
 - 7.4.3 C编程中访问特殊寄存器的指令

ARM汇编语言与C/C++的混合编程

- ❑ 嵌入式软件开发过程中，通常会使用包括ARM汇编语言和C/C++语言在内的多种语言。一般情况下，一个ARM工程(**Project**)应该由多个文件组成，其中有可能包括扩展名为.s的汇编语言源文件、扩展名为.c的C语言源文件、扩展名为cpp的C++源文件，以及扩展名为.h的头文件等。
- ❑ 通常程序会使用汇编完成处理器启动阶段的初始化等工作，有些对处理器运行效率较高的底层算法也会采用汇编语言进行编写和手工优化，而在开发主程序时一般会采用C/C++语言，因此嵌入式软件开发人员必须掌握ARM汇编语言与C/C++语言混合编程技能。

7.4.1 C语言与汇编语言之间的函数调用

1.ATPCS

- 动机：为了使单独编译的C语言程序和汇编程序之间能够相互调用，必须为子程序之间的调用制定一定的规则。
- ATPCS（ARM-Thumb Procedure Call Standard，基于ARM指令集和Thumb指令集过程调用规则）规定了一些不同语言撰写的函数之间相互调用（mix calls）的基本规则，这些基本规则包括子程序调用过程中寄存器的使用规则、数据栈的使用规则、以及参数的传递规则。
 - 参数传递规则：当参数个数不超过4个时，可以使用寄存器R0~R3来传递参数，如果参数多于4个，则将剩余的字数数据通过堆栈传递，入栈的顺序与参数传递顺序相反，即最后一个字数数据先入栈，第一个字数数据最后入栈。

观察printf()输出参数如何传递

59

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

SystemCoreClockUpdate

Registers

Register	Value
Core	
R0	0x0000105C
R1	0x20001170
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x0000102B
R8	0x00000000
R9	0x00000000
R10	0x0000104C
R11	0x0000104C
R12	0x00000000
R13 (SP)	0x20001160
R14 (LR)	0x00000461
R15 (PC)	0x00000FE2
xPSR	0x61000000
Banked	
System	
Internal	
Mode	Thread

Disassembly

```
13: //观察PC, 观察printf()输入参数存放位置
0x00000FE0 B51C PUSH {r2-r4,lr}
14: printf("Hello USTCer\n");
15:
0x00000FE2 A007 ADR r0,{pc}+0x20 ; @0x00001000
0x00000FE4 F7FFFA62 BL.W __2printf(0x000004AC)
16: char * srcstr = "0123456" ;
0x00000FE8 A409 ADR r4,{pc}+0x28 ; @0x00001010
```

ustc_class_show_C.c startup_ARMCM3.s Scopy.s

```
11 int main (void) {
12
13 //观察PC, 观察printf()输入参数存放位置
14 printf("Hello USTCer\n");
15
16 char * srcstr = "0123456" ;
17 char dststr [ ] = "abcdefg" ;
```

Command

Load "E:\cxh_work\MDK-learn\sample3\Objects\sample_for_c.axf"
BS \\sample_for_c\ustc_class_show_C.c\14

Memory 1

Address: 0x00000fc0

0x00000FC0:	00 00 00 20 8A 68 00 2A 01 DC FF F7 B7 BB 52 1E 8A 60 4A 68 C0
0x00000FD5:	B2 53 1C 4B 60 10 70 70 47 00 00 1C B5 07 A0 FF E7 62 FA 09 A4
0x00000FEA:	0B A1 03 C9 CD E9 00 01 21 46 68 46 FF F7 CD FF 00 BF FE E7 00
0x00000FFF:	00 48 65 6C 6C 6F 20 55 53 54 43 65 72 0A 00 00 00 30 31 32 33
0x00001014:	34 35 36 00 61 62 63 64 65 66 67 00 3A 74 74 00 3A 74 74 00 3A
0x00001029:	74 74 00 4C 10 00 00 00 00 00 20 10 00 00 00 FC 03 00 00 5C 10
0x0000103E:	00 00 10 00 00 20 60 11 00 00 18 04 00 00 40 78 7D 01 00 00 00
0x00001053:	00 00

ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess

Simulation t1: 0.00033267 sec L:14 C:1 CAP NUM SCRL

PC偏移寻址, 文本池位置
R0传递第一个参数

7.4.1 C语言与汇编语言之间的函数调用

2. C程序调用汇编函数实例

- 为保证调用时参数传递正确，C程序调用的汇编函数时遵循ATPCS。
- 如果汇编函数和调用函数的C程序不在同一个文件中，则需要用汇编语言中用**EXPORT**声明汇编语言起始处的标号为外部可引用符号，该标号应该为C语言中所调用函数的名称。

[例]调用汇编函数，实现把字符串srcstr复制到字符串dststr中。

//需要调用的汇编函数原型并加extern关键字

```
extern void strcpy (char * d , char * s ) ;
```

```
int main () {  
    char * srcstr = "0123456" ;  
    char dststr [ ] = "abcdefg" ;  
    strcpy(dststr, srcstr);  
    return 0 ;  
}
```

7.4.1 C语言与汇编语言之间的函数调用

2. C程序调用汇编函数实例

;汇编语言源程序*.s源程序*.c文件在同一工程中

```
AREA MyCode, CODE, READONLY
EXPORT strcpy
strcpy PROC; 必须与EXPORT后面标号一致

LOOP    LDRB    R2, [R1], #1 ;R1指向源字符串地址，取出字符内容存入R2
                                ;更新R1=R1+1，第一次调用时R1指向源字符串首地址
        STRB    R2, [R0], #1 ;R0指向目的字符串地址，R2中内容存入R0指向内存单元，
                                ;更新R0=R0+1，第一次调用时R0指向目的字符串首地址

        CMP    R2, #0
        BNE    LOOP ;先执行后判断，源字符串的终止符'\0'也复制到目的字符串

        MOV    PC, LR
        ENDP
```

7.4.1 C语言与汇编语言之间的函数调用

3.汇编程序调用C函数实例

讲义上还有一个传递参数超过4个示例

- 在汇编程序中调用C语言函数，需要在汇编程序中利用**IMPORT**说明对应的C函数名，按照ATPCS的规则保存参数。完成各项准备工作后利用跳转指令跳转到C函数入口处开始执行。跳转指令后所跟标号为C函数的函数名。

```

PRESERVE8      ;根据ATPCS,堆栈数据需要设置为8字节对齐
THUMB
EXPORT call_C
IMPORT MY_C_FUNCTION ;声明MY_C_FUNCTION为外部引用符号
ALIGN
call_C PROC;必须与EXPORT后面标号一致
ENTRY

;LDR    SP, =0x20000460 ;设置堆栈指针
MOV     R0, #1 ;参数1赋R0
MOV     R1, #2 ;参数2赋R1
BL      MY_C_FUNCTION
ENDP

```

```

int MY_C_FUNCTION(int a, int b) {
    return (a+b);
}

```

7.4.2 C/C++语言与汇编语言的混合编程

- ❑ 嵌入式系统开发中，目前使用的**主要**编程语言是**C和汇编**，**C++**已经有相应的编译器，但是现在使用**还比较少**。在稍大规模的嵌入式软件中，大部分的代码都是用C编写的。
- ❑ C语言中内嵌汇编代码，可以在C/C++程序中实现C/C++不能完成的一些操作，同时程序的代码效率也会更高。

❑ 在C程序中嵌入汇编常用两种方法

❑ 内联汇编， **inline assembler**

❑ 内嵌汇编， **Embedded assembler**

```
asm  
(  
    "NOP"  
);
```

```
asm void Test()  
{  
    nop  
    BX lr  
}
```


μVision IDE演示，基于C代码的Project

□ 代码编辑界面

- 新建project的选项
- 分析Startup

□ 调用示例

- C调用ASM
- ASM调用C
- 嵌入汇编
- 访问特殊寄存器



<https://www.eeo.cn/live.php?lessonKey=610aa3470c0e38a9>

From 31:44 to 53:10

在C语言程序中嵌入汇编指令的注意事项

□ 在C语言中内嵌汇编指令存在一些限制：

- (1) 不能直接向PC寄存器赋值，程序跳转要使用B或者BL指令。
- (2) 在使用物理寄存器时，不要使用过于复杂的C表达式，避免物理寄存器冲突。
- (3) R12和R13可能被编译器用来存放中间编译结果，计算表达式值时可能将R0到R3、R12及R14用于子程序调用，因此要避免直接使用这些物理寄存器。
- (4) 一般不要直接指定物理寄存器，而让编译器进行分配。

在汇编中调用C语言定义的函数和全局变量

- 使用内联或者内嵌汇编不用单独编辑汇编语言文件使用起来比较方便，但是有诸多限制。
- 当汇编文件较多的时候就需要使用专门的汇编文件编写汇编程序，在C语言和汇编语言进行数据传递的最简单的形式是使用全局变量。

7.4.3 C编程中访问特殊寄存器

□ 两种不同方式

- CMSIS-CORE提供了访问 Cortex-M3/M4处理器内特殊寄存器的多个函数
- 也可以使用CMSIS定义的“已命名寄存器变量”

//使用CMSIS-core函数

```
unsigned int uiTmp = 0;
```

```
uiTmp = __REV16(1);           //Reverse byte order (16 bit)
```

```
uiTmp = __get_xPSR();         //Get xPSR Register
```

```
uiTmp = __get_MSP();          //Get Main Stack Pointer
```

//使用CMSIS预定义的寄存器（已命名寄存器变量）

```
uiTmp = SCB->VTOR;            //Get Vector Table Offset Register
```



第7章 ARM程序设计

- 7.1 ARM程序开发环境
- 7.2 ARM汇编程序中的伪指令
- 7.3 ARM汇编语言程序设计
- 7.4 ARM汇编语言与C/C++的混合编程