==========
==========
==========《嵌入式技术 基础课程》之 ARM 汇编==========
==========
==========
==========

出品：佳嵌工作室

“佳嵌工作室”致力于物联网、嵌入式产品以及‘物联网、嵌入式技术课程’的研发，希望所推出的技术课程，能够帮助到正在努力

学习物联网、嵌入式的同学，欢迎大家和我们一起共同学习，共同进步。

第一部分：ARM 相关概念

0、为什么要推出 ARM 汇编

0.1 对比 X86，8051 汇编，从更底层的角度去理解相关的知识的

0.2 为后续嵌入式课程作准备(S3C2440,STM32F103,EXYNOS4412)

0.3 学习目标：了解 ARM 的体系结构，能看懂 ARM 汇编

1、ARM 分类

参考官方： http://infocenter.arm.com/

1.1 版本号分类

ARM7 ARM9 ARM11

Cortex 系列

Cortex-R

Cortex-M M0 M3 M4 M7

Cortex-A A8 A9 A15 A72 A53

1.2 指令集分类(指令集原则上是向下兼容的)

ARMV4 ARMV7 ARMV8

2、ARM 商业模式及半导体公司

2.1 IP 核(知识产权)

ARM MIPS X86 PowerPC(IBM)

ARM 出卖 IP 而不卖芯片的策略

2.2 半导体公司

三星、高通、苹果、NXP(Freescale)、TI、ST、ATMEL、Ambarella、Amlogic

MTK、海思、全志、瑞芯微、展讯(基带)、炬力、联咏

例如：S3C2440(ARM9)　EXYNOS4412(Cortex-A9)
STM32F103(Cortex-M3)
i.MX.RT1050(Cortex-M7)
i.MX.283(ARM9)
AT89C51(8051 核)
AR9331(MIPS)
HI3518(ARM9)
NT96650(MIPS)
A33(Cortex-A7)
RK3399(Cortex-A72 A53)

2.3 生态系统：核公司&半导体公司&半导体生产&产品研发&产品生产
作为个体：半导体行业，产品应用行业
方案选型-调试 Demo 板-打样生产
(通用方案与专用方案)

3、嵌入式处理器
CPU、MCU、GPU、PLC、DSP、SOC、ASIC、FPGA 的关系
CPU 包含运算器和控制器
MCU 最简单的理解就是一个芯片就是一个微型计算机
GPU 包含很多个运算器，专门解决图形计算
PLC 半成品工控设备
SOC 片上系统,专用/通用软件和硬件系统(ESP8266)
ASIC 专用集成电路,IP 核设计(软核/硬核/固核)
DSP 处理大量数据运算，一般用于音视频计算上。
FPGA 解决程序滞后性问题，并行计算
比如：一个产品上可以同时包含 MCU，GPU，SOC，ASIC，DSP，FPGA

第二部分：ARM 体系结构

4、哈佛结构与冯式结构
4.1 两者的区别
独立的存储架构和信号通道（代码与数据是否统一编址？）
与是否统一编址没有关系，也与 CPU 没有关系，与计算机整体设计有关
CACHE 的引入(CPU 内部哈佛结构)
8086 冯式结构　　相同存储(RAM)相同的通道　　(统一编址)　区别：运行态与存储态
STM32F103 哈弗结构　　不同的存储(ROM/RAM)不同的通道　(统一编址)
8051 改进型的哈弗结构　不同的存储(ROM/RAM)相同的通道　(独立编址)
ARM9 改进型的冯式结构　相同的存储(RAM)/不同的通道　(统一编址)
总结：高性能单片机：冯式结构　单片机：哈弗结构

4.2 总线与 IO 访问
ARM 总线结构（AHB，APB，AXI）与 内部外设

(系统)总线访问与 IO 访问（独立与统一） --区别于单片机中的 GPIO 通信访问,GPIO 本身是一个内部外设

IO 就是指 CPU 的各种内部与外部外设

IO 接口的编址方式：IO 接口中能被 CPU 访问的寄存器称为端口/寄存器

端口与存储器统一编址（ARM）uart gpio

端口与存储器独立编址（X86）8259A 8255(并口) 8253(定时器)

通过 MEMR/MEMW 和 IOR/IOW 两组控制信号来实现对 I/O 端口和存储器的不同寻址

注意：内存(memory map)与内存条

5、ARM 的处理器状态和处理器模式

注意：在不同的体系下一些概念有所不同，在这里我们使用 S3C2440 手册进行参考

5.1 处理器状态

		ARMV4	ARMV7
ARM 状态	执行效率高	32bit	32bit
THUMB 状态	代码密度好	16bit	16/32bit (THUMBEE 状态)

处理器状态的切换 BX BLX

各种指令：ARM 指令(32 位)，THUMB 指令(16 位)，THUMB2 指令(32/16 位)

CORTEX-M 只是指令 THUMB2 指令子集

5.2 处理器模式

usr,fiq,irq,svc,abt,sys,und

usr: The normal ARM program execution state

fiq: Designed to support a data transfer or channel process

irq: Used for general-purpose interrupt handling

svc: Protected mode for the operating system

abt: Entered after a data or instruction prefetch abort

sys: A privileged user mode for the operating system

und: Entered when an undefined instruction is executed

设计的目的：提高响应速度，注意：保护模式/特权模式，需要 MMU 和代码支持

5.3 ARM 流水线设计：五级流水线(ARM9)

每条指令五个工位：取指-译码－执行－访存－回写

	t1	t2	t3	t4	t5	t6
指令 1	取指	译码	执行	防存	回写	
指令 2		取指	译码	执行	防存	回写
指令 3			取指	译码	执行	防存 回写
指令 4				取指	译码	执行 防存 回写
指令 5					取指	译码 执行 防存 回写

结论：程序指针 PC = 执行指令地址 + 8 字节(2 条指令)

6、ARM 内部寄存器及 SFR

6.1 ARM 内部寄存器:设计在 CPU 内部，特点：速度快

一共 37 个寄存器：31 个通用寄存器+6 个状态寄存器

通用寄存器(R0-R12) R13 R14 R15 R16

R13(SP):栈指针,不同模式下栈空间是不一样

R14(LR):链接寄存器 （存放断点），硬件自动完成

R15(PC):程序指针，取指令的位置

PSR---CPSR SPSR 程序状态寄存器

条件状态(NZCV) 保留位 I F T 模式位(M4-M0)

6.2 SFR(Special Function Register)

IO 端口/寄存器-它属于外设的组成部分!ARM 是采用与存储器统一编址的方式

使用软件编程控制某一硬件，其实就是编程读写该硬件的寄存器。

7、ARM 异常处理

7.1 模式与异常

异常的类型：Reset undefined instruction swi prefetch/bort data/abort Reserved IRQ FIQ

处理器的模式：每一个异常对应一种模式，但不是一一对应关系

RESET：

1， Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit

2，Forces the PC to fetch the next instruction from address 0x00

3，Execution resumes in ARM state

7.2 ARM 异常处理过程

进入异常硬件完成下面动作

(1), '断点'---->LR

(2), CPSR---->SPSR

(3), 修改 CPSR 中的模式位

(4), PC 跳到中断向量

软件完成：

(1), 中断散转,二级中断

(2), 保护现场

(3)，中断服务程序

(4), 恢复现场 ---- LR--->PC , SPSR-->CPSR

注意：If the processor is in THUMB state when an exception occurs,

it will automatically switch into ARM state when the PC is loaded with the exception vector address

7.3 中断向量表

Address	Exception	Mode in Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software Interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort

0x00000014 Reserved　　Reserved

0x00000018 IRQ　　IRQ

0x0000001C FIQ　　FIQ

7.4 异常返回

BL　MOV PC, R14　　；用于子程序调用

SWI　MOVS PC, R14_svc　　；用于 OS 的系统调用

UDEF MOVS PC, R14_und　　；执行的的指令不合法

FIQ　SUBS PC, R14_fiq,#4　　；用于快速中断

IRQ　SUBS PC, R14_irq,#4　　；用于中断

PABT SUBS PC, R14_abt,#4　　；取指令的位置不合法(没有权限或者地址不存在),通常是 BL 跳转的位置

DABT SUBS PC, R14_abt,#8　　；访问不合法的地址(没有权限或者地址不存在)

注意：

(0)，PC 在不同情况含义不一样，比如：PC(excute)=PC(fetch) - 8，程序进入异常时 LR 值(LR= PC-4)

(1)，异常会清除处理器的流水线。

(2)，每种异常发生的时机不一样，对于中断而言，指令还未被执行就产生了，对于 BL/SWI/UNEF 而言，指令是执行过了的。

当发生预取中止，将预取的指令标记为无效，在指令达到流水线执行阶段执行才进入异常，如果指令在流水线中因

为分支而没有被执行，异常不会发生。

当发生数据中止异常时，异常会在访存阶段发生。

(3)，PABT 和 DABT 常结合 MMU 和 OS 来使用，abort 的处理程序就必须找出异常中断的原因，使要求的数据可用，

并重试被中止掉的指令。

http://www.ethernut.de/en/documents/arm-exceptions.html

8、ARM 的存储系统

8.1 存储器 与 存储控制器

存储器单元：存放数据的位置，地址是挂在系统总线。

存储控制器：初始化存储器器件，有对应的寄存器配置，寄存器是挂在系统总线上。

8.2 存储器存储数据格式

大端格式：8051　　ARM 处理器将最高位字节保存在最低地址，最低字节保存在最高地址

小端格式：ARM,X86　一个字当中最低地址的字节被看作是最低位字节，最高地址的字节被看作是最高位字节

8.3 对齐访问与非对齐访问

PC 中的 bit[1:0]为 0　（32bit 对齐）

0x40000000　0x40000001 0x40000002 0x40000003　　0x33221100

0x00　　0x11　　0x22　　0x33

非对齐访问处理：

(1) 不可预测

(2) 忽略造成访问不对齐的低地址位

(3) 先忽略最低两位，然后使用最低两位地址位控制装载数据循环右移（0x00332211）

第三部分：ARM 指令系统

9、CISC 与 RISC

9.1 CISC 与 RISC

9.1.1 CPU 模型：运算器和控制器

复杂指令集和精简指令集(取决于 N)

N=111（8051） 复杂指令集

N=34 （ARM） 精简指令集

SWAP（1） <---> MOV （3） 2/8 定律

9.1.2 程序语言：编译型 与 解释型

编译型(本地语言)：直接生成机器码

解释型：JAVA：一次编译到处运行（JVM 本身是一个进程，去分配内存空间，将字节码转成机器码，用户程序包含在此进程中）

源文件--编译器--字节码---JVM（解释）--- 机器码

9.2 RISC 架构特点：

9.2.1 采用固定长度的指令格式

9.2.2 使用单周期指令，便于流水线操作

9.2.3 大量使用寄存器，采用 LS 结构访问在存储器

采用 CISC 架构的处理器具有相反的特征，不过功能更强大。采用 CISC 架构的处理器有 X86

10、开发工具

10.1 工具介绍：

10.1.1 交叉编译工具链及提供商

交叉编译工具链的命名规则为：arch [-vendor] [-os] [-(gnu)eabi]

arm/mips 提供商 目标 OS Embedded Application Binary Interface

比如：1，arm-none-eabi-gcc 用于编译 ARM 裸机(uboot,kernel)，不能编译与 OS 相关的应用程序，由 GNU 提供

2，arm-none-linux-gnueabi-gcc 基于 ARM 架构的 Linux 系统的程序，包含 uboot,kernel,Linux 应用程序，由 Codesourcery 公司提供

3，armcc ARM 官方的编译工具链，功能和 arm-none-eabi 类似

10.1.2 ARM 官方开发工具：ADS，RVDS，KEIL(MDK-arm),DS-5 Development

KEIL(MDK):编译器用的是 armcc，开发环境：μVision IDE，只支持 ARM7/9 和全系 Cortex-M，不支持 Cortex-A 系列。

DS5:编译器用的是 armcc 或者 arm-none-eabi-gcc，开发环境：Eclipse，支持 ARM 全系列内核。

10.1.3 KEIL MDK-ARM C51 uVision IDE

KEIL:公司名，被 ARM 收购

MDK-ARM(Microcontroller Development Kit):专门针对 ARM 核的开发工具包

C51:针对 8051 的开发工具包

uVision IDE：用于工程管理，源代码编辑

10.2 MDK-ARM 的使用(软件安装/工程建立)

(1) Project---New uVision Project

(2) 选择 CPU(Sumsung-S3C2440),不添加启动代码

(3) Project--Options-Target 设置(ROM1 0x00 0x1000)

(4) File-New 保存命名 startup.s 并添加文件到工程中

(5) AREA RESET,CODE,READONLY ;注意 TAB

```
ARM
ENTRY
MOV R0,#0x00
B   .
END
```

11、寻址方式

11.1 指令格式

<指令助记符>{<执行条件>}{S} <目标寄存器>,<操作数 1 的寄存器>{,<第 2 操作数>}

注意：< >号内的项是必需的；{}号内的项是可选的，S：是否影响 CPSR 寄存器的值，书写时影响 CPSR

CMP 不需要增加"S"就可改变相应的标志位

例如：SUBS PC,LR,#4

MOV R0,#0x00

LDR R0,[R1]

所有指令都是 32bit,load/store 体系结构(对存储器的访问只能使用加载和存储指令实现)

11.2 条件码与机器码

使用指令条件码可实现高效的逻辑操作，提高代码执行效率。

例如：

```
if(a > b)   a++;          CMP R0,R1
else        b++;          ADDHI R0,R0,#1
                          ADDLS R1,R1,#1
```

MOV R0,#0x00 <--> MOVAL R0,#0x00

注意：CPSR 里面的标志位（不用记） 对应于 条件码助记符（记）

常用的条件码：EQ NE HI LS .....

EQ/NE:等于/不等于(equal / not equal)

HS/LO:无符号数高于或等于/无符号数小于(higher or same/lower)

HI/LS:无符号数高于/无符号数低于或等于(higher/lower or same)

GE/LT:有符号数大于或等于/有符号数小于(greater or equal/less than)

GT/LE:有符号数大于/有符号数小于或等于(greater than/less or equal)

机器码：MOV R0,#0x00 <------> 1110 001 1101 0 0000....000 -->0xE3A00000

11.3 寻址方式(针对源操作数而言)

11.3.1 立即数寻址

MOV R0,#0x300

LDR R0,=0x12345678 (伪指令)

注意：8 位存数据，用 X 表示（0～255），4 位存移位的次数，用 Y 表示（0～15），立即数= X 循环右移 2*Y 个位

立即数 0xF200 是由 0xCF2 间接表示的，即是由 8 位的 0xF2 循环右移 24（2*12）得到 X=0xF2;Y=0xC

11.3.2 寄存器寻址

MOV R0,R1

11.3.3 寄存器移位寻址

MOV R0,R2,LSL #3

11.3.4 寄存器间接寻址-LDR/STR (load/stroe)

LDR R0,[R1]

STR R0,[R1]

11.3.5 基址变址寻址

STR R0,[R1,#-4]

STR R0,[R1,#-4]!

STR R0,[R1],#-4

注意：!表示回写

11.3.6 多寄存器寻址 (Load multiple registers)

STMIA R0!,{R2-R7,R12} ;大括号中的内容表示寄存器中的值，R0 对应的是存储器上的地址

注意组合：I/D(Increase/Decrease) A/B(After/Before)

11.3.7 堆栈寻址

STMFD SP!,{R1-R3} ;压栈，寄存器号大的先入栈

LDMFD SP!,{R1-R3} ;出栈

注意组合：F/E I/D

满堆栈：堆栈指针指向最后压入的有效数据项

空堆栈：堆栈指针指向下一个待压入数据的空位置

11.3.8 相对寻址 Branch(jmp/call)

B BL BLX BX

B 跳转指令

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

12、指令分类

12.1 数据处理指令：数据传送类，算术逻辑运算类 比较指令

12.1.1 数据传送类

MOV 类指令:核内寄存器间的数据传送

加载和存储指令（L/S）：核内寄存器与挂在存储器总线上器件的数据传送

注意：核内寄存器：R0,R1... 外设寄存器：GPA0CON

12.1.2 算术逻辑运算类(+,-,*,/ & | ^)

ADD R1,R1,#0x01

SUBS R1,R2,#0x01

MUL R1,R2,R3

AND R1,R2,0xff // ORR EOR

BIC R1,R1,#0x0f //最低 4 位清 0

注意：ARM 指令集中没有除法指令，通过通过软件(移位-比较-相减)实现
不过在 ARMV7 指令集中增加了除法指令

12.1.3 比较指令(cmp tst teq)

CMP R0,#0x01

TST R0,#0x01 //判断 R0,最后一位是否为 0，与 EQ，NE 条件码结合使用

TEQ R0,R1 //判断 R0,R1 是否相等，与 EQ，NE 条件码结合使用

注意：它们不需要加后缀 S，会直接影响程序状态寄存器，常用于选择/循环结构中

12.2 跳转指令

B BL BX BLX

BX Rn Rn[0] = 1 THUMB 状态 （在原有的 Rn 上加 1 即可）

Rn[0] = 0 ARM 状态

12.3 协处理器指令

协处理器是协助主 cpu 完成一些特定功能（MMU、Cache 和 TLB 等等）的处理器
功能上和操作系统的虚拟地址映射、cache 管理等有关。

主要的指令有三类：

12.3.1 数据操作指令：ARM 处理器通过 CDP 命令通知协处理器完成一些初始化操作，命令的解析由协处理器完成。

12.3.2 存储器数据传送指令：ARM 处理器通过 LDC/STC 指令将内存单元的数据读取或者写入到协处理器的寄存器中。

12.3.3 寄存器数据传送指令：ARM 处理器通过 MCR/MRC 指令将 ARM 处理器中的寄存器数据与协处理器中的寄存器数据进行数据传送

协处理器有 16 个，一般 CP15 是设置 MMU，Cache，大小端等关于存储器配置的。CP15 包含 16 个 32 位寄存器，c0,c1.....c15

https://blog.csdn.net/gameit/article/details/13169405

MRC p15,0,R0,c0,c0,0; //读取主标示符到寄存器 R0 中

mrc p15,0,r0,c1,c0,0; // 读出 cp15 的 c1 到 r0 中

orr r0, r0, #(1<<7); // bit7 置 1 设置为大端存储

mcr p15,0,r0,c1,c0,0

12.4 杂项指令

软中断：SWI 一般用于系统调用，有两种方式完成

MOV R0,#0x01 ;设置子功能为 0x01

SWI 0x12 ;调用 0x12 号软中断 (open read write ..fork..)

MOV R0,#0x12

MOV R1,#0x01

SWI 0x00 ;产生中断，模式变成 SVC，跳到中断向量位置去。

MRS/MSR：寄存器与 PSR 数据传送

PSR 的控制域(fsxc)：位[31：24]为条件标志位域，用 f 表示；位[23：16]为状态位域，用 s 表示；

位[15：8]为扩展位域，用 x 表示；位[7：0]为控制位域，用 c 表示。

```
mrs  r0,cpsr
bic   r0,r0,#0x1f
orr   r0,r0,#0xd3
msr  cpsr,r0
msr  cpsr_c, #0xd3        @ I & F disable, Mode: 0x13 - SVC
```

12.5 伪指令(ADR ADRL LDR NOP)

伪指令不是 ARM 指令，但可以把它当作指令来使用，主要作用是方便编写程序，最终还是会转换成 ARM 指令

LDR R0,=0x12345678 ;绝对地址 文字池(常量) ---- LDR R0,[PC,#0x0008]

ADR R0,xxx(地址标号) ;相对地址 ---- SUB R0,PC,#0x00000004

ADRL R0,xxx(地址标号) ;相对地址 ---- ADD R0,PC,#0x00000000

ADD R0,R0,#0x00000000

NOP ---- MOV R0,R0

区分：

```
adr r0,_start      ;取运行地址
ldr r1,_start      ;取的运行地址中的值(指令)
ldr r2,=_start     ;取的是链接地址
```

第四部分：ARM 汇编编程

13、ARM 汇编程序格式

源程序中的语句可以分为两种类型：指令性语句，指示性语句。

指示性语句就是一些伪操作，在 MDK 编译环境下的伪操作有下面几种

13.1 符号定义伪操作:用于定义 ARM 汇编程序中的"变量"

用于定义全局变量的 GBLA 、 GBLL 和 GBLS

用于定义局部变量的 LCLA 、 LCLL 和 LCLS

用于对变量赋值的 SETA 、 SETL 、 SETS

为一个通用寄存器列表定义别名－－RLIST (List RLIST {R0-R3})

为一个协处理器的寄存器定义名称－－CN (Power CN 6)

为一个协处理器定义名称－－CP

为一个双精度的 VFP 寄存器定义名称－－DN

为一个单精度的 VFP 寄存器定义名称－－SN

为一个浮点寄存器定义名称－－FN

例子：GBLA Test1　　LCLS Test2　　GBLL Test3

Test1 SETA 0xaa　　Test2 SETS "Testing"　　Test3 SETL {TRUE}

注意：全部变量用于程序体中，而局部变量用于宏定义中，常用于循环控制、逻辑运算、条件判断中，不会分配空间

这些变量是在编译器的预处理阶段进行的。　MOV R0,#(1+2)

13.2 数据定义伪操作:用于为特定的数据分配存储单元，同时可完成已配存储单元的初始化

DCD DCW DCB 用于分配一片连续的字、半字、字节的存储单元并用指定的数据初始化 （初始化的全局数组）

SPACE 用于分配一片连续的存储单元，并用 0 初始化　　（相当于 C 语言中的未初始化全局数组）

MAP 用于定义一个结构化的内存表首地址，与 FIELD 配合使用　（相当于 C 语言中的结构体）

FIELD 用于定义一个结构化的内存表中的数据域

LTORG 用于声明一个数据缓冲池（文字池）的开始，如果没有使用 LTORG 声明文字池，则汇编器会在程序末尾自动声明

例子：
```
Str DCB    "This is Test!"      Data DCD 1,2,3
DataSpace SPACE 10
MAP 0x40000000
A FIELD 4
B FIELD 4
```

符号描述：DCB(=) DCD(&) SPACE(%) MAP(^) FILED(#)

13.3 汇编控制伪操作:用于控制汇编程序的执行流程,如：条件汇编、宏定义、重复汇编控制等

IF 、 ELSE 、 ENDIF　([|])

WHILE 、 WEND

MACRO 、 MEND 、MEXIT

例子：
```
CONFIG EQU 16                    MACRO
IF {CONFIG}=16                       MAX $data1,$data2
    ADD R0,R0,R1                     .....
ELSE                             MEND
    SUB R0,R0,R1                     MAX 1,2
ENDIF
```

13.4 其他伪操作

AREA－－用于定义一个代码段或数据段

ALIGN－－通过添加填充字节的方式，定义边界对齐方式，默认的情况下，代码段

和数据段是 4 字节对齐的

CODE16、CODE32－－伪操作通知编译器，其后的指令序列通过 16 位的还是 32 位来编译

ENTRY －－用于指定汇编程序的入口

END－－用于通知编译器已经到了源程序的结尾

EQU－－用于为程序中的数字常量、标号等定义一个等效的字符名称，相当于 C 中的 define

EXPORT（或 GLOBAL ）－－用于在程序中声明一个全局的标号

IMPORT（ EXTERN ）－－用于通知编译器要使用一个在其他的源文件中定义的标号

GET （或 INCLUDE ）－－用于将一个源文件包含到当前的源文件中

例子： AREA Buf,DATA,READWRITE ;符号 AREA 和 END 都不能顶格写，只有标号可以而且必须顶格写

注意:伪操作不会产生机器指令，它是辅助编译器工作的，在不同环境下伪代指令的描述是不一样的(对比 GNU)

符号可以代表地址、数值、变量

当符号代表地址时又称为标号,符号代表某个特定数值时又称为符号常量,符号代表变量时又称为变量名

标号：基于 PC，基于寄存器(MAP R0)，绝对地址

常量：与 EQU 搭配使用

变量：变量名是一个符号地址，系统会给每一个变量名分配一个内存地址

在 MDK 中的伪操作都可以通过 Help - uVision Help（ARM Development Tools）查找

14、程序控制

顺序结构：程序流程无分支，无循环，无转移，以直线方式一条指令接着一条指令顺序执行

选择结构：根据不同的条件，分成若干个分支路，配合条件码使用

循环结构：在顺序和选择控制中，任一语句执行次数最多一次，循环控制中可以多次。

循环结构一般分成四部分：1，初值 2，循环体 3，修改初值 4，判断条件

例子 1：1+2...+100

例子 2：冒泡排序

15、ARM 汇编与 C 的混合编程

15.1 内嵌汇编 __asm __asm("指令") ，例如关闭/打开总中断开关 CPSR

```
__asm                    //使用 C 中变量名代替寄存器
  {
    MOV var,x
    ADD y,var,x/y
  }
```

内联汇编语言中的寄存器名被编译器视为 C 或 C++语言中的变量，

所以内联汇编中出现的寄存器名不一定和同名的物理寄存器相对应。

这些寄存器名在使用前必须声明，否则编译器将提示警告信息

15.2 汇编访问 C 中的全局变量

15.2.1 利用 IMPORT 声明全局变量

15.2.2 利用 LDR 得到其地址

```
AREA    globals，CODE，READONLY
        EXPORT asmadd
        IMPORT   gvar；声明外部变量 gvar
asmadd
        LDR R1，=gvar；装载变量地址
        LDR R0，[R1] ；读出数据
        ADD R0，R0，#1；加 1 操作
        STR R0，[R1]；保存变量值
        MOV PC, LR
        END
```

15.3 ATPCS（ARM-Thumb Produce Call Standard）：ARM 程序和 Thumb 程序中子程序调用的基本规则

子程序调用过程中寄存器的使用规则

利用 R0-R3 来传递参数，R4-R11 用来保存局部变量。

数据栈的使用规则

栈采用的是满递减(FD),当参数超过 4 个时，超过的部分使用栈来传递参数，返回值存于 R0

例子：

15.3.1 用汇编语言定义加法子程序 int SUM(int a, int b)，通过 C 语言调用，实现 23+54，并将结果输出到屏幕上

15.3.2 用 C 语言定义加法子程序 int SUM(int a, int b)，通过汇编语言调用，实现 23+54，并将结果保存在地址为 0x40001000 处

startup.s sum.s xmain.c

16、ARM 汇编 THUMB2 指令集

16.1 ARM THUMB THUMB2

ARM 指令：32 位，支持所有功能，所有指令都可以条件执行

THUMB：16 位，不能访问协处理器，特权指令和特殊功能指令，只有 B 指令才能条件执行

THUMB2：Thumb-2 是 16 位 Thumb 指令集的一个超集，指令是 32 位或者是 16 位。与 ARM 指令 32 位编码格式是不同的。

由汇编器来决定使用 16 位指令还是 32 位指令。

16.2 ARMV8 (64bit 体系结构)

ARMv8 提供 AArch32state 和 AArch64 state 两种 Execution State，不在区分 ARM 状态与 THUMB 状态。

在 AArch32 状态下，提供两个指令集 A32（32bit）、T32（16/32bit），两种指令是通过 BX 切换的。

在 AArch64 状态下，只支持 A64 指令集，固定长度为 32bit.

ARMV8 提供了不同的运行级别：Exception Level 与 Security （EL0(app)，EL1，EL2，EL3）类比于 X86 的 Ring0-ring3(app)

16.3 CORTEX-M3

Cortex-M3 只支持 Thumb-2 指令，在 STM32 单片机课程中会详细讲解 Thumb-2 指令集。

学习目标：能否看懂 ARM 汇编代码

学习建议：

1，知识就是一张网，相互联系的,不管是从哪个方向突破，都会遇到不熟悉的问题。承认没有学习过的知识点。

2，源点(熟悉知识点)+推理----->新知识。完善自己的知识体系结构

3，熟悉知识<---相互-->应用知识

出品：佳嵌工作室

“佳嵌工作室”致力于物联网、嵌入式产品以及‘物联网、嵌入式技术课程’的研发，希望所推出的技术课程，能够帮助到正在努力

学习物联网、嵌入式的同学，欢迎大家和我们一起共同学习，共同进步。
