# ucos ii 简介

* 源码

1. µC/OS-II全部以源代码的方式提供，大约 有5500行。
2. CPU相关的部分使用的是针对Intel80x86微 处理器的代码。
3. µC/OS-II可以很容易地移植到不同架构的 嵌入式微处理器上。
4. 源代码文件介绍
   1. 对函数和环境的定义： PC.C
   2. 与处理器类型无关部分： OS\_CORE.C OS\_FLAG.C OS\_MBOX.C OS\_MEM.C OS\_MUTEX.C OS\_Q.C OS\_SEM.C OS\_TASK.C OS\_TIME.C µCOS-II.C µCOS-II.H
   3. 与处理器类型相关部分： OS\_CPU\_A.S OS\_CPU\_C.C OS\_CPU.H
   4. 给整个内核库提供总体的include文件：INCLUDES.H
   5. 配置文件，定义使用µC/OS-II中的哪些功能： OS\_CFG.H

* 源代码获取

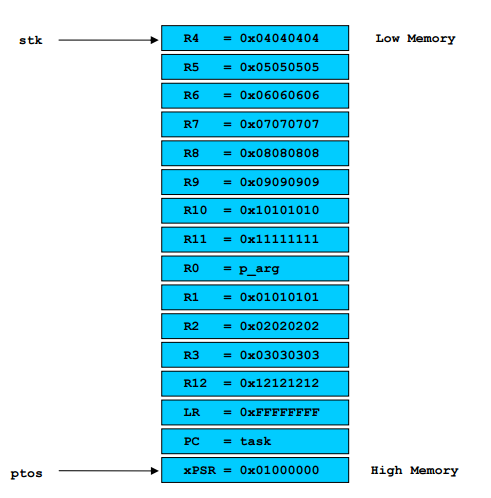
## 流程

OSInit()-->OSInitHookBegin()-->OSInitHookEnd()

## 函数

OStaskCreateHook() 当任务创建时被调用（OSTaskCreate（）或者 OSTaskCreateExt()） 此例中调用的是 App\_TaskCreateHook(ptcb)

OSTaskInit 对一个任务进行初始化



OSTaskSwHook（） 任务切换的时候调用

OSTimeTickHook() 在OSTimeTick()的开始处被调用，这个任务可以用来更新TimerTask

OS\_CPU\_SysTickInit() 初始化系统时钟，BSP必须提供OS\_CPU\_SysTickClkFreq()来获取系统时钟频率

## 任务切换

OS\_CPU\_SR\_Save() 实现了OS\_CRTICAL\_METHOD # 3

OS\_CPU\_CR\_Restore()

OSStartHighRdy() 由OSStart()调用，运行最高优先级的任务

OSCtxSw 切换过程中设置CPU寄存器

OS\_Sched 任务切换

OSPendSV() 处理所有的PendSV，任务切换

## 中断

14号 PendSV 执行上下文切换

15号 SysTick 处理系统时钟中断

# cortex m4简介

32位，具有DSP，MAC的指令，有FPU，使用ARM-v7指令集。

* 维基百科的解释

Conceptually the Cortex-M4 is a Cortex-M3 plus [DSP](https://en.wikipedia.org/wiki/Digital_signal_processor) instructions, and optional floating-point unit (FPU). If a core contains an FPU, it is known as a Cortex-M4F, otherwise it is a Cortex-M4.

Key features of the Cortex-M4 core are:[[5]](https://en.wikipedia.org/wiki/ARM_Cortex-M#cite_note-M4-TRM-5)

* ARMv7E-M architecture[[10]](https://en.wikipedia.org/wiki/ARM_Cortex-M#cite_note-ARMv7-M-Manual-10)
* 3-stage [pipeline](https://en.wikipedia.org/wiki/Pipeline_(computing)) with [branch speculation](https://en.wikipedia.org/wiki/Branch_predictor).
* Instruction sets:
  + Thumb-1 (entire).
  + Thumb-2 (entire).
  + 32-bit hardware integer multiply with 32-bit or 64-bit result, signed or unsigned, add or subtract after the multiply.
  + 32-bit hardware integer divide (2-12 cycles).
  + [Saturation arithmetic](https://en.wikipedia.org/wiki/Saturation_arithmetic) support.
  + DSP extension: Single cycle 16/32-bit [MAC](https://en.wikipedia.org/wiki/Multiply%E2%80%93accumulate_operation), single cycle dual 16-bit MAC, 8/16-bit [SIMD](https://en.wikipedia.org/wiki/SIMD) arithmetic.
* 1 to 240 [interrupts](https://en.wikipedia.org/wiki/Interrupt), plus [NMI](https://en.wikipedia.org/wiki/Non-maskable_interrupt).
* 12 cycle interrupt latency.
* Integrated sleep modes.
* dsp是什么

数字信号处理的简称

# 附录：ARM汇编

## 工作模式（7种）

svc 管理模式 -- 指令 swi 0x0 调用0号软中断

abt 终止模式

und 未定义指令模式

特权模式 = 除了用户模式之外的6种模式

异常模式 = 除了用户模式和系统模式

模式改变： 通过系统调用，通过外部中断（IRQ，FIQ） ， 通过异常处理

## 工作状态

* ARM指令集 -- 对应ARM工作状态
  + 32位ARM指令
  + 指令要求字对齐
* Thumb -- 对应Thumb工作状态
  + 16位Thumb指令
  + 指令半字对齐

复位后总是处于ARM状态，处理器通过BX命令进入Thumb状态

BX是一条跳转指令，当其最低位为0的时候进入Thumb状态

## 通用寄存器

* CPSR状态寄存器

r0~r15为通用寄存器， r15作为PC寄存器

分为三类：

r0~r7 未分组 -- 在所有的工作模式下，这些寄存器都指向对应的一个物理寄存器（共享）

r8~r14 分组

r8~r12 每个寄存器对应不同的两个物理寄存器 – 分别是 R8\_fiq~R12\_fiq, R8\_usr~R12\_usr

r13~r14 对应6个不同的物理寄存器，其中的一个用户模式与系统模式共用R1x\_mode x=13,14, mode=usr,fiq,irq,svc,abt,und

R13寄存器通常用作SP寄存器（堆栈寄存器），一般都需要初始化每种模式下的R13，使其指向正常的栈空间。

R14 链接寄存器 LR，使用BL子程序调用指令时，R14保存R15的下一条指令

* 常用用法
  + 返回

mov PC,LR

bx LR

或者

LDMFD SP!,{<REGS>,PC}

* + 进入

STMFD SP!, {<Regs>,LR}

r15 PC

ARM状态下，位[1:0]为0 [31:2]用于保存PC

Thumb状态下 位[0]为0，[31:1]保存PC

* 程序访问

可直接访问 R0~R7 PC SP LR CPSR

* 指令条件后缀

ADDHI中的HI表示

## 寻址方式

* 立即数

ADD　R0,R0,#1 ; #表示立即数

* 寄存器寻址

ADD R0,R1,R2

* 寄存器间接寻址

LDR R0,[R1]

STR R0,[R1] ; R0-->[R1]

* 基址寻址

LDR R0,[R1,#4] ；以立即数作为偏移

LDR R0,[R1,R2] ;以寄存器作为偏移

LDR R0,[R1,#4]! ;!后缀表示R1自增4

LDR R0,[R1],#4 ；移动后R1自增4

* 相对寻址

以PC作为基地址来寻址

BL SORT ;跳转到某个标号

…

SORT

…

MOV PC,LR

* 多寄存器寻址

LDM/STM Rleft,Rright

LDM R0, {R1,R2,R3,R4}; 一次最多完成16个通用寄存器的值传送；寄存器按固有次序 [R0]->R1,..,[R0]->R4

LDMIA SP!,{R1,R0} ; 表示如果进行自增，IA后缀暗示增加 [SP]->R0,SP+=4,[SP]->R1,SP+=4

* 堆栈寻址

满堆栈方式：指针指向最后一个数据

空堆栈方式：指针指向下一个位置

递增/递减方式

满递增 {LDMFA,STMFA}

满递减 {LDMFD,STMFD}

空 F换成E

STMFD SP!,{R0-R7,LR} ; LR最先进栈，R0最后

LDMFD SP!,{R0-R7,PC}; R0最先出栈，PC最后

## 编程概述

指令集类型：Load/Store型

指令格式： <opcode> [<cond>][S] <Rd>,<Rn>[,oprand2]

cond如NE,EQ

S:是否影响CPSR的值

Rd：目标寄存器

Rn:第一个操作数寄存器

## 指令

ADC 带进位加法

ADD

AND

B 跳转指令

BL 带返回的跳转

BIC 位清0

BLX 带状态转换的BL

BX 带状态的B

CDP 协处理器数据操作指令

CMN 取负比较指令

CMP 比较指令

EOR 异或

LDC 存储器->协处理器传输

LDM 加载多个寄存器

STM 批量写入内存

LDR 存储器->寄存器

MOV 数据传送指令

SUB

SWI 软中断

TEQ 相等测试

TST 位测试

SWP 交换指令

* 条件码(占据4位，从0b0000~0b1110)

EQ NE Z=1,0

CS CC 无符号大于等于 无符号小于 C=1,0

MI PL 负数 非负数 N=1,0

VS VC 溢出与否 V=1,0

HI C=1 Z=0 无符号大于

LS C=0 Z=1 无符号小于等于

GE N=V 有符号大于或等于（下面都是带符号的）

LT

GT

LE

AI 忽略

* 跳转指令

MOV PC,R14或者B系列指令

* 加载/存储指令

LDR 字加载

LDRB 字节加载

LDRH 半字加载

STR同理

* 异常指令

SWI 0x0 调用中断

BKPT 0x00ff 产生软件断点中断，用于程序调试

## 函数

参数少于5个时，使用R0~R3来传递

5个或者以上借助栈来实现

返回值通过R0