作业1

STM32F407ZETx

CPU工作原理

CPU = ALU + Control Unit

Register(寄存器)：CPU内部的存储区域，暂时存放参与运算的数据和运算结果。

寄存器:锁存器/触发器实现，只包含存储电路的时序逻辑电路。

创建start.s

代码如下：

AREA mystack, DATA, READWRITE

stack\_start

SPACE 0x200

stack\_end

AREA RESET, DATA, READONLY

vectors

DCD stack\_end ;stack-top

DCD test\_start ;begin code

vectors\_end

AREA mycode, CODE, READONLY,ALIGN=3

test\_start

;test code

B . ;while(1)

END

为什么需要“堆栈”？为了支持过程调用(函数)。

“现场保护”

函数的具体功能的代码

“现场恢复”

Cortex M4有两个堆栈，双堆栈

MSP 主堆栈指针

PSP 进程堆栈指针

为什么需要双堆栈呢？

为了支持操作系统。把操作系统用的堆栈和用户进程用的堆栈分开。

R14(LR): Linked Register 链接寄存器

在执行过程调用的指令的时候，我们需要保存该指令的下一条指针的地址，

因为这个地址，就是需要返回的地址。

有一个专门的寄存器，用来保存过程调用调用的返回地址。->LR(R14)

MOV R0, #3

MOV R1, #4

BL sum ;

// BL:把下一条指令的地址(如下的: (A))存放在LR中

// 跳转是通过把：要跳到的那个地址，直接赋值给PC

// sum -> PC

(A) ADD R0, R1,

sum:

ADD R0,R0,R1

MOV PC, LR ; -> return 函数返回，过程返回。

R15(PC): Program Counter 程序计数器。 保存下一条要执行的指令的地址。

PC会在取指后，会自动增加指令所占的bits位数。

在ARM Cortex M4, PC + 4

在有“指令流水线”情况下，PC的值会有所不同

作业2

应用状态寄存器 APSR: 计算结果的标志

N Z C V Q

N Z C V Q

我们每一条指令的执行都可以影响这些状态标志位。

N: 负数标志。

Z : Zero 。零标志。结果所有bit位都为0,则xPSR.Z == 1

是0不是0

C: Carry 借位或进位标志。bit29

进位： 在做加法运算时，产生了进位。则C == 1,否则 C == 0

借位： 在做减法运算时，没产生借位。则C == 1,否则 C == 0

ADC, ADD, CMN 加法。如果产生了进位，则C == 1,否则 C == 0

SBC, SUB, CMP 减法。如果生生了借位，则C == 0,否则 C == 1

V: oVerflow 溢出标志。bit28

反映有符号数做加减运算所得结果是否溢出，如果运算结果超过当前

运算位数所能表示的范围，则溢出 xPSR.V = 1, 否则为0. 在有符号的运算中，进位(借位,C)与溢出是两个完全不同的概念。

Q: 饱和标志

饱和计算： 通过将数据强制置为最大(或最小)允许值，减小了

数据畸变，当然畸变仍然存在，不过若数据没有超过最大

允许范围太多，就不会有太大的问题。

ICI : Interruptible-Continuable Instrument 可中断－可继续指令位

Cortex M4工作模式

"模式"： 不同环境，不同的角色

ARM cortex M4有两种工作模式:

Thread Mode: 线程模式

Handler Mode: 处理模式(异常中断模式)

异常/中断 是什么？ 打断CPU指令执行顺序的事件，称为中断。

为什么要支持两种模式呢？ 为什么不只用一种模式呢？ Thread Mode

如果只用一种模式，thread mode,为了响应一些外部事件(比如说，用户是否

按下某个按键？):

轮询：轮流询问。 通过轮询，CPU也可能 响应外部事件，但是

轮询天生就有缺陷:

(1) 浪费CPU

(2) 占用总线， Bus is always busy.

(3) 轮询有一个时间差，轮询的时间间隔。不及时！！！

在CPU内部设计一个 “中断模式”：

为了提高效率和响应速度。

两种模式之间是怎么切换的呢？ 重要。如图thread\_Mode与Handler\_Mode之间的切换

Handler Mode :

中断模式，当一些比较重要的事件，产生时，CPU中止正在做的

事情，切换到Handler Mode下去执行，此时 “特权等级”

中断处理完成后，再返回到断点处，继续Thread Mode运行。

Thread Mode:

线程模式。

特权等级 : 可以跑一些如OS的代码

非特权等级: 可以跑一些如 "用户态"的代码

特权等级 -> 非特权等级

但是:

非特权等级 不可以 切换到 特权事件，除非产生“中断”

作业3

startup.s

stack\_size EQU 0x200 ;512B

vectors\_size EQU 0x400 ;1KB

;define stack

AREA m\_stack, NOINIT, READWRITE

\_stack

SPACE stack\_size

\_stack\_top

;define INT-VEC

AREA RESET, DATA, READONLY

\_vectors

DCD \_stack\_top;first is stack top

DCD test\_code ;second is begin code

SPACE vectors\_size

\_vectors\_end

;define code

AREA m\_code, CODE, READONLY, ALIGN=3

test\_code

;my test code

B . ;while(1)

END

作业4

用汇编语言实现一个函数，判断a是否为b的倍数。

stack\_size EQU 0x200 ;512B

vectors\_size EQU 0x400 ;1KB

;define stack

AREA m\_stack, NOINIT, READWRITE

\_stack

SPACE stack\_size

\_stack\_top

;define data

AREA m\_data, DATA, READWRITE

\_data\_a

SPACE 4

;define INT-VEC

AREA RESET, DATA, READONLY

\_vectors

DCD \_stack\_top ;first is stack top

DCD test\_code ;second is begin code

SPACE vectors\_size

\_vectors\_end

;define code

AREA m\_code, CODE, READONLY, ALIGN=3

test\_code PROC

;my test code

MOV R0,#8;R0 a

MOV R1,#2;b

BL Is\_Multi

LDR R1,=\_data\_a

STR R0,[R1];R0-->data\_a

B .

ENDP

Is\_Multi PROC

PUSH {R2-R3,LR}

MOV R2,R0;R2-a-R0

MOV R0,#0;0 is not multi

MOV R3,R1;i(R3) = b(R1)

\_loop

CMP R2,R1;a--b

BLT \_loop\_end;< LT-end

MOVEQ R0,#1;1 is multi--R0

BEQ \_loop\_end;RETURN

ADD R1,R1,R3;b = b + i

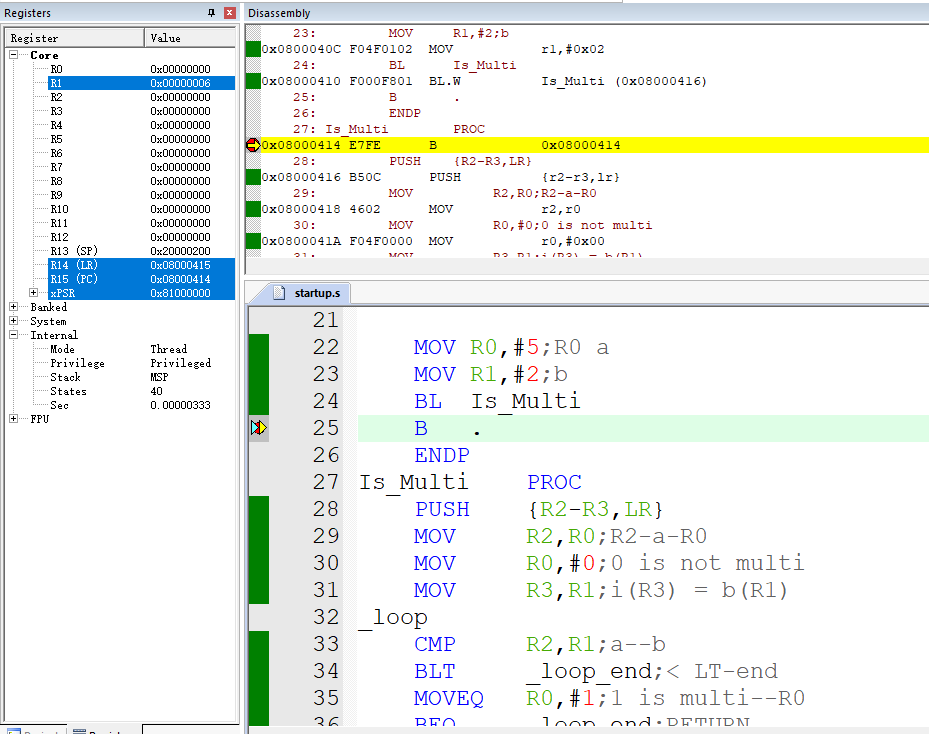
B \_loop

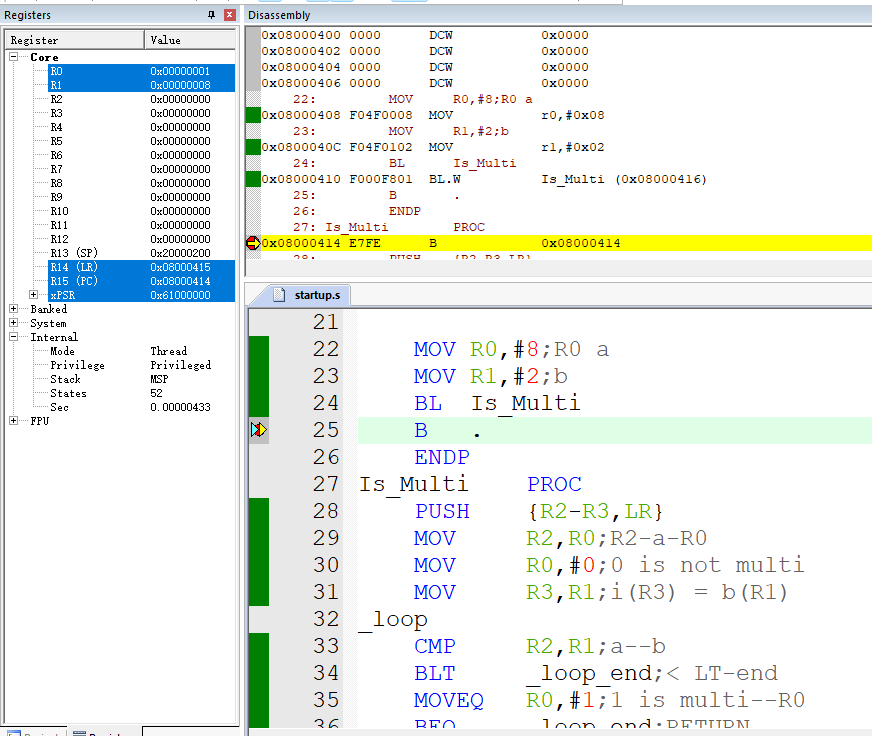
\_loop\_end

POP {R2-R3,PC}

ENDP

END





用汇编语言实现一个函数，判断一个数m是否为质数。

stack\_size EQU 0x200 ;512B

vectors\_size EQU 0x400 ;1KB

;define stack

AREA m\_stack, NOINIT, READWRITE

\_stack

SPACE stack\_size

\_stack\_top

;define data

AREA m\_data, DATA, READWRITE

\_data\_a

SPACE 4

;define INT-VEC

AREA RESET, DATA, READONLY

\_vectors

DCD \_stack\_top ;first is stack top

DCD test\_code ;second is begin code

SPACE vectors\_size

\_vectors\_end

;define code

AREA m\_code, CODE, READONLY, ALIGN=3

test\_code PROC

;my test code

MOV R0,#4;R0 a

BL Is\_Prime

LDR R4,=\_data\_a

STR R0,[R4];R0-->data\_a

B .

ENDP

Is\_Prime PROC

PUSH {R1-R3,R5,LR}

MOV R2,R0;R2-m(R0)

MOV R5,#1;1 is prime --R5

MOV R3,#2;i=2--R3

\_loop\_m

CMP R3,R2;i--m

BGE \_loop\_m\_end;i>=m GE--end

MOV R0,R2;(R0)a-m(R2)

MOV R1,R3;(R1)b-i(R3)

BL Is\_Multi

CMP R0,#1;if(1 is multi)

MOVEQ R5,#0;R0==1--return 0 is not prime

BEQ \_loop\_m\_end

ADD R3,R3,#1;i++

B \_loop\_m

\_loop\_m\_end

POP {R1-R3,R5,PC}

ENDP

Is\_Multi PROC

PUSH {R2-R3,LR}

MOV R2,R0;R2-a-R0

MOV R0,#0;0 is not multi

MOV R3,R1;i(R3) = b(R1)

\_loop

CMP R2,R1;a--b

BLT \_loop\_end;< LT-end

MOVEQ R0,#1;1 is multi--R0

BEQ \_loop\_end;RETURN

ADD R1,R1,R3;b = b + i

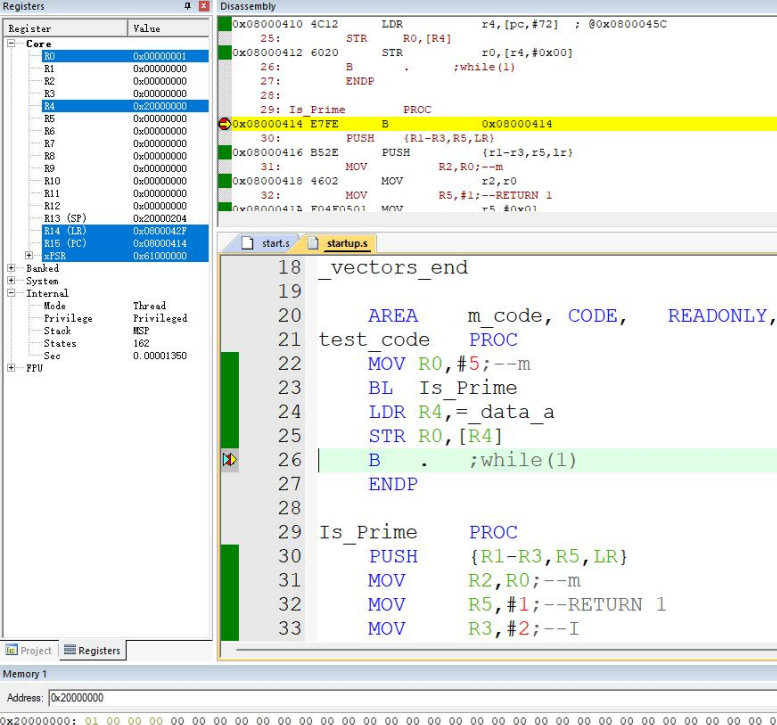
B \_loop

\_loop\_end

POP {R2-R3,PC}

ENDP

END



作业5

判断闰年函数

stack\_size EQU 0x200 ;512B

vectors\_size EQU 0x400 ;1KB

;define stack

AREA m\_stack, NOINIT, READWRITE

\_stack

SPACE stack\_size

\_stack\_top

;define data

AREA m\_data, DATA, READWRITE

\_data\_a

SPACE 4

;define INT-VEC

AREA RESET, DATA, READONLY

\_vectors

DCD \_stack\_top ;first is stack top

DCD test\_code ;second is begin code

SPACE vectors\_size

\_vectors\_end

IMPORT leapyear

PRESERVE8

;define code

AREA m\_code, CODE, READONLY, ALIGN=3

test\_code

;my test code

MOV R0,#7d4;2004

BL leapyear

B .

END

leapyear.c

int leapyear(int n)

{

    int year ,a;

    year = n;

    if(year%400==0)

        a=1;

    else

    {

        if(year%4==0&&year%100!=0)

            a=1;//a = 1 leapyear

        else

            a=0;//a = 0 not

    }

    return a;

}

