Cortex-M3 programming

Texas Instruments, www.ti.com
CortexM3InstructionSet.pdf

STMicroelectronics, www.st.com
CortexM3Programmer.pdf
PM0056

Modes and Levels

Thread mode Used to execute application software. The processor enters Thread mode when it comes out of reset.

Handler mode Used to handle exceptions. The processor returns to Thread mode when it has finished exception processing.

The *privilege levels* for software execution are:

Unprivileged The software:

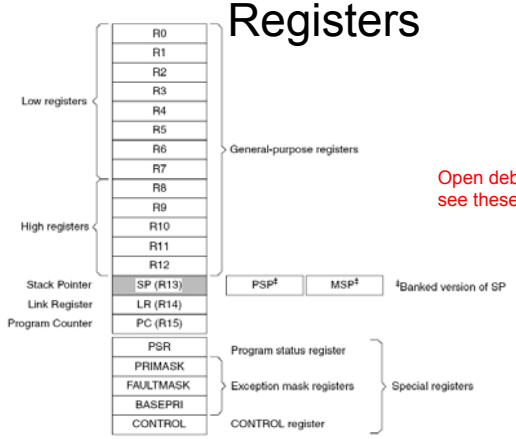
- Has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
- Cannot access the system timer, NVIC, or system control block
- Might have restricted access to memory or peripherals.

Unprivileged software executes at the unprivileged level.

Privileged The software can use all the instructions and has access to all resources.

Privileged software executes at the privileged level.

Registers



Open debugger to see these registers

Thread
- Main stack
- Process stack

Handler
- Main stack

Registers

R0-R3 parameters

R4-R11 must be saved

General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations.

Stack pointer

The *Stack Pointer* (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = *Main Stack Pointer* (MSP). This is the reset value.
- 1 = *Process Stack Pointer* (PSP).

Which SP is active?

On reset, the processor loads the MSP with the value from address 0x00000000.

Link register

The *Link Register* (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor loads the LR value 0xFFFFFFFF.

Program counter

L14 is important

The *Program Counter* (PC) is register R15. It contains the current program address. Bit[0] is always 0 because instruction fetches must be halfword aligned. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004.

Function calls

```
void delay (int cnt){
  while (cnt--);
}


delay(10);
```

```
delay
    SUB     r0,r0,#0x01
    BNE     delay
    BX      lr

    MOV     r0,#0x0A
    BL      delay
```

Follow the link register LR

Function calls

```
void function1 (void)
{
  output(0x01);
}


int main (void) {
…
  function1();
…
}
```

```
function1
        PUSH    {r4-r6,lr}
        MOV     r0,#0x01
        BL      output
        POP     {r4-r6,pc}

main
...
        BL      function1
...
```

R4-R11 must be saved

Draw a stack picture

The accesses happen in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest number register using the highest memory address

Program Status Register

Figure 3. APSR, IPSR and EPSR bit assignments

Q = saturation

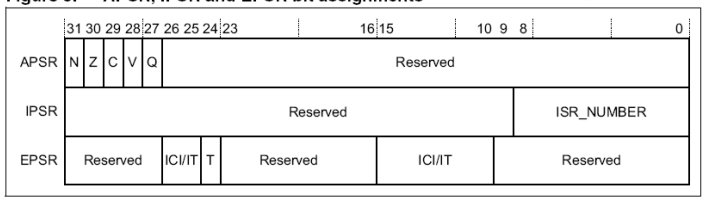
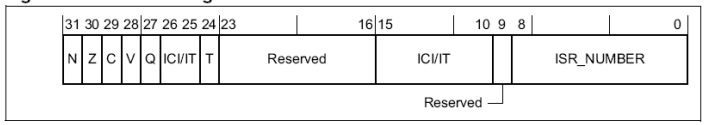


Figure 4. PSR bit assignments



T = Thumb bit

Interrupt Program Status Register

Bits	Description
Bits 31:9	Reserved
Bits 8:0	**ISR_NUMBER:** This is the number of the current exception: 0: Thread mode 1: Reserved 2: NMI 3: Hard fault 4: Memory management fault 5: Bus fault 6: Usage fault 7: Reserved .... 10: Reserved 11: SVCall 12: Reserved for Debug 13: Reserved 14: PendSV 15: SysTick 16: IRQ0[1] ....

Run debugger:
- stop in ISR and
- look at IPSR

Figure 2-3, The IPSR Register.

Execution Program Status Register

The Execution PSR (**EPSR**) contains two overlapping fields:

- the Interruptible-Continuable Instruction (ICI) field for interrupted load multiple and store multiple instructions

PUSH {r4-r6,lr}

- the execution state field for the If-Then (IT) instruction, and the T-bit (Thumb state bit).

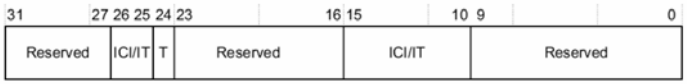


Figure 2-4, The EPSR Register.

Race, Critical Sections

- Shared information
  - Permanently allocated
  - Memory or I/O registers
- Sequences involving at least one write
  - E.g., RMW, WW, WR
- Nonatomic sequence
  - Begins with access to permanent memory
  - Ends with access to permanent memory

LDREX
STREX

Priority Mask Register

Priority mask register

The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in *Table 2 on page 13* for its attributes. *Figure 5* shows the bit assignments.

Figure 5. PRIMASK bit assignments

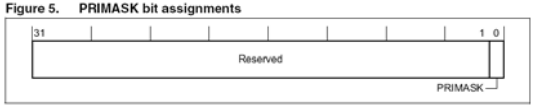


Table 7. PRIMASK register bit definitions

Bits	Description
Bits 31:1	Reserved
Bit 0	**PRIMASK:** 0: No effect 1: Prevents the activation of all exceptions with configurable priority.

CPSID I

CPSIE I

MRS R0, PRIMASK
CPSID I

MRS PRIMASK,R0

Interface latency

Code from uCOS-II

```
SRSave
      MRS    R0, PRIMASK
      CPSID  I
      BX     LR
SRRestore
      MSR    PRIMASK, R0
      BX     LR
```

```
// Prototypes :
long SRSave (void);
void SRRestore(long sr);
```

```
#define OS_ENTERCRITICAL() {sr = SRSave();}
#define OS_EXITCRITICAL() {SRRestore(sr);}

void  Task (void *p_arg)  {
long  sr=0;
  OS_CRITICALENTER();
// critical section
  OS_CRITICALEXIT();
}
```

Where is the I bit saved?

Control Register

Figure 8. CONTROL bit assignments

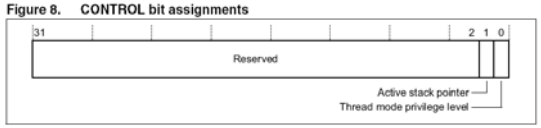


Table 10. CONTROL register bit definitions

Bits	Function
Bits 31:2	Reserved
Bit 1	**ASPSEL:** Active stack pointer selection Selects the current stack: 0: MSP is the current stack pointer 1: PSP is the current stack pointer. In Handler mode this bit reads as zero and ignores writes.
Bit 0	**TPL:** Thread mode privilege level Defines the Thread mode privilege level. 0: Privileged 1: Unprivileged.

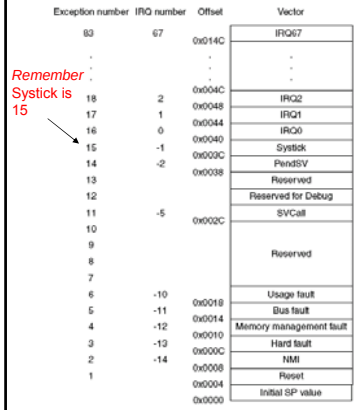
Reset debugger:
- look at CONTROL
- stop in ISR and
- look at CONTROL

Exceptions

Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address or offset[2]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	-
4	-12	Memory management fault	Configurable[3]	0x00000010	Synchronous
5	-11	Bus fault	Configurable[3]	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	Usage fault	Configurable[3]	0x00000018	Synchronous
7-10	-	-	-	Reserved	-
11	-5	SVCall	Configurable[3]	0x0000002C	Synchronous
12-13	-	-	-	Reserved	-
14	-2	PendSV	Configurable[3]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[3]	0x0000003C	Asynchronous
16-83	0-67	Interrupt (IRQ)	Configurable[4]	0x00000040 and above[5]	Asynchronous

Table 16

Interrupt vector table



Remember Systick is 15

Show vector table in project, debugger

Define
- Group priority 0-15
- Subpriority
- Nested exceptions
- Return
- Tail chaining
- Late arrival

Stacking
- 8 Registers
- R0-R3, R12
- LR
- Return address
- PSR

LR=EXC_RETURN
- 0b11110001 Ret to Handler MSP
- 0b11111001 Ret to Thread MSP
- 0b11111101 Ret to Thread PSP

aligned to double-word address

Run debugger:
- stop in ISR and
- look at LR, draw stack frame

Supervisor Call

3.9.10 SVC

Supervisor Call.

Syntax

`SVC{cond} #imm`

where:
- *'cond'* is an optional condition code, see *Conditional execution on page 56*.
- *'imm'* is an expression evaluating to an integer in the range 0-255 (8-bit value).

Operation

The SVC instruction causes the SVC exception.

imm is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

Condition flags

This instruction does not change the flags.

Examples

```
SVC  0x32  ; Supervisor Call (SVC handler can extract the immediate value
           ; by locating it via the stacked PC)
```

Code from uCOS-II

```
OSCtxSw
    LDR     R0, =NVIC_INT_CTRL
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
```

```
#define  OS_TASK_SW()  OSCtxSw()
```

```
    OS_CPU_PendSVHandler
        CPSID  I                ; Prevent interruption during context switch
        MRS    R0, PSP          ; PSP is process stack pointer
    ;....
        MSR    PSP, R0          ; Load PSP with new process SP
        ORR    LR, LR, #0x04    ; exception return uses process stack
        CPSIE  I
        BX     LR
```

```
NVIC_PENDSVSET  EQU    0x10000000
NVIC_INT_CTRL   EQU    0xE000ED04
```


System tick (initialization)

```
void SysTick_Init(unsigned long period){   volatile unsigned long delay;
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOD; // activate port D
  Counts = 0;
  GPIO_PORTD_DIR_R |= 0x01;    // make PD0 output
  GPIO_PORTD_DEN_R |= 0x01;   // enable digital I/O on PD0
  NVIC_ST_CTRL_R = 0;               // disable SysTick during setup
  NVIC_ST_RELOAD_R = period - 1; // reload value
  NVIC_ST_CURRENT_R = 0;       // any write to current clears it
                                    // Systick=priority 2
  NVIC_PRI3_R =  (NVIC_PRI3_R&0x00FFFFFF)|0x40000000;
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC
                   +NVIC_ST_CTRL_INTEN;
  EnableInterrupts();
}
```

PeriodicSysTickInts_8962

LM3S8962.pdf:
- RCGC2 (pg 229)
- STCTRL (pg 109)
- SYS PRI3 (pg 120)
- STRELOAD (pg108)

System tick

```
#define GPIO_PD0  (*((volatile unsigned long *) 0x40007004))

void SysTick_Handler(void) {
  GPIO_PD0 = GPIO_PD0^0x01;
  Counts = Counts + 1;
}

void main(void){
…
 SysTick_Init(50000); // 1msec, assuming 50 MHz bus clock
…
}
```

PeriodicSysTickInts_8962

Reset debugger:
- stop in ISR and
- single step through ISR
- look at assembly code
- draw stack

Summary

- Learn the assembly language
- Study how interrupts are processed
- Determine what goes on the stack
- Develop and deploy your debugging skills
  - TFBF (time to first bus fault)
  - Developing an OS is inherently unstable
  - Control and observability