EE445M/EE380L.12
Embedded and Real-Time Systems/ Real-Time Operating Systems

Lecture 3:

RTOS, OS Kernel,
Operating Modes, Context Switch

Lecture 3   J. Valvano, A. Gerstlauer   1
                              EE445M/EE380L.12

References & Terminology

µC/OS-III, The Real-Time Kernel, or a High Performance, Scalable, ROMable, Preemptive, Multitasking Kernel for Microprocessors, Microcontrollers & DSPs, by Jean J Labrosse, 2009. (there are several versions, with and without a board, including for TI Stellaris MCUs)

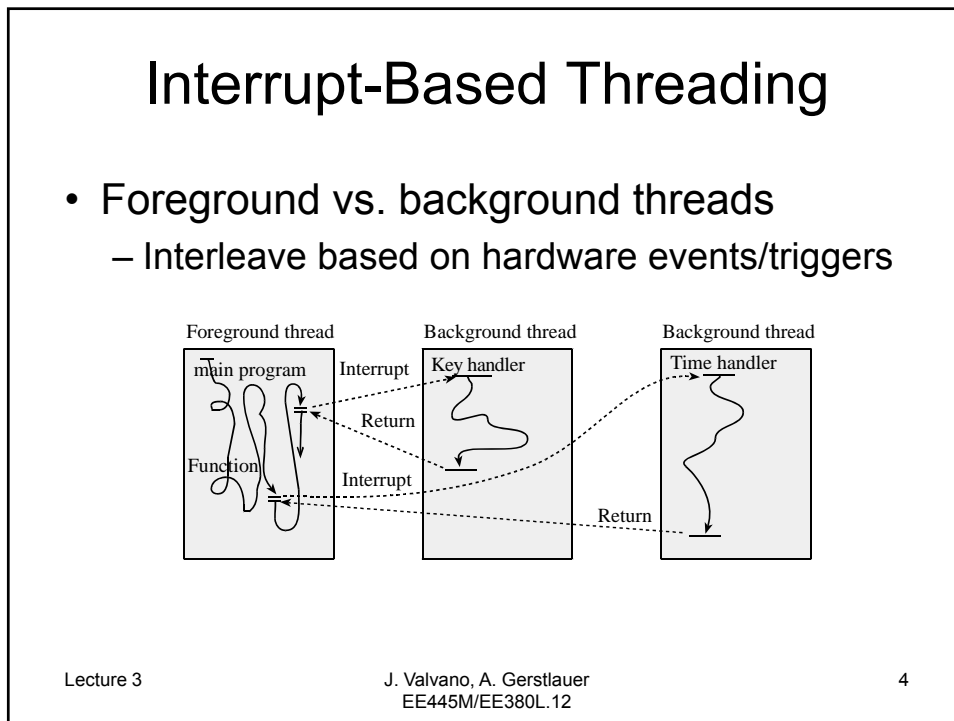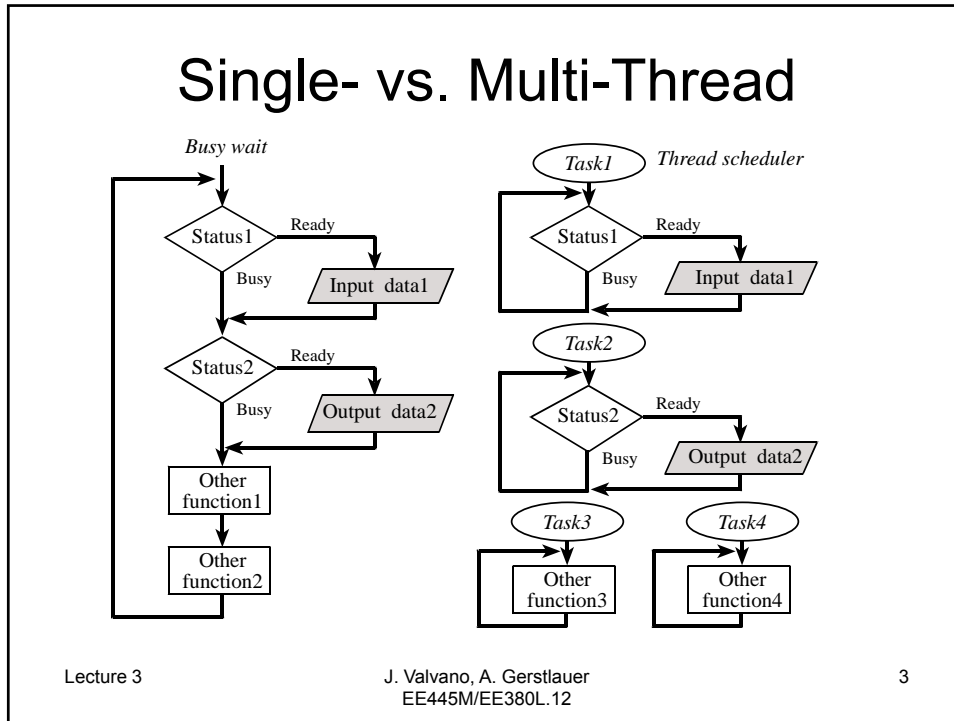**µC/OS-II: The Real Time Kernel**, by Jean J. Labrosse , 2002, ISBN 1-5782-0103-9.

The Definitive Guide to the ARM Cortex-M3 and Cortex-M4 Processors, Third Edition, by Joseph Yiu, 2013, ISBN 0-1240-8082-0.

Embedded Systems: Real Time Operating Systems for ARM Cortex-M Microcontrollers, Jonathan W. Valvano (Ch. 3, 4 & 5)

Lecture 3   J. Valvano, A. Gerstlauer   2
                              EE445M/EE380L.12

Single- vs. Multi-Thread

Lecture 3   J. Valvano, A. Gerstlauer   3
EE445M/EE380L.12

Interrupt-Based Threading

- Foreground vs. background threads
  - Interleave based on hardware events/triggers

Lecture 3   J. Valvano, A. Gerstlauer   4
EE445M/EE380L.12

Threads and Tasks

```
void Producer(void) {
  uint16_t data;
  while(1) {
    data = ADC_In();
    if(OS_Fifo_Put(data) == 0)
      DataLost++;
  }
}
```

```
void Consumer(void) {
  uint16_t data, average;
  uint32_t sum;
  uint16_t n;
  while(1) {
    sum = 0;
    for(n = 0; n < LENGTH; n++) {
      data = OS_Fifo_Get();
      sum = sum + data;
    }
    average = sum/LENGTH;
    OS_MailBox_Send(average);
  }
}
```

```
void Display(void) {
  uint16_t data, voltage;
  while(1){
    data = OS_MailBox_Recv();
    voltage = 31*data/64;
    LCD_Message(0,"v(mV) =",voltage);
  }
}
```
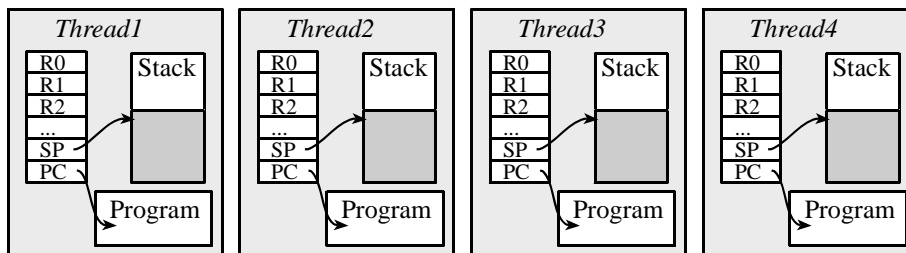
Lecture 3   J. Valvano, A. Gerstlauer   5
                                   EE445M/EE380L.12

Multi-Threading / Multi-Tasking

Thread: Same program & data
Task: Independent program & data (= process**)

Scheduling

** More in Lecture 8

Lecture 3   J. Valvano, A. Gerstlauer   6
                                   EE445M/EE380L.12

Real-Time Operating System (RTOS)

- Thread management & scheduling

- Thread communication & synchronization

- Time management

Thread Classification

- Periodic, execution at regular intervals
  - E.g., ADC, DAC, motor control
  - E.g., Check CO levels
- Aperiodic, execution can not be anticipated
  - Execution is frequent
  - E.g., New position detected as wheel turns
- Sporadic, execution can not be anticipated
  - Execution is infrequent
  - E.g., Faults, errors, catastrophes

Real-Time

- RT threads have deadlines
  - Hard real time
    - Guaranteed bounded latency
  - Firm real time
    - Missed deadline loss of quality
  - Soft real time
    - Delayed response reduces value
  - Not real time
    - Best effort, no deadlines whatsoever

Lecture 3   J. Valvano, A. Gerstlauer   9
                                    EE445M/EE380L.12

Thread Scheduler

- Thread management
  - Thread states
- Scheduling algorithm
  - What? (order of threads) — **Round robin / Weighted round robin / Priority**
  - How? (when to decide) — **Static / Dynamic / Deterministic/fixed**
  - Why? (when to run) — **Cooperative / Preemptive**
- Performance measures
  - Utilization
  - Latency
  - Bandwidth

Lecture 3   J. Valvano, A. Gerstlauer   10
                                    EE445M/EE380L.12

Time Management

- System time
- Time stamps
  - When did it occur?
    - Performance measures
- Thread sleeping
  - Yield and wakeup after certain delay
    - Run other tasks instead of busy waiting
- Measurements
  - Input capture period -> wheel RPM
  - Input capture PW -> ultrasonic distance

Lecture 3   J. Valvano, A. Gerstlauer   11
                            EE445M/EE380L.12

Additional OS Requirements

- Run-time configurable, extensible
  - Priority, stack size, fifo size, time slice
- Reliability, certification
  - Medical, transportation, nuclear, military
- Scalable
  - 10 threads versus 200 threads
- ROMable
  - Runs in ROM

Lecture 3   J. Valvano, A. Gerstlauer   12
                            EE445M/EE380L.12

Hooks

- Run user supplied code at strategic places
- Allows you to
  - Extend the OS
  - Implement debugging
  - Implement performance testing
  - Implement black box recording
- Collect run-time performance data

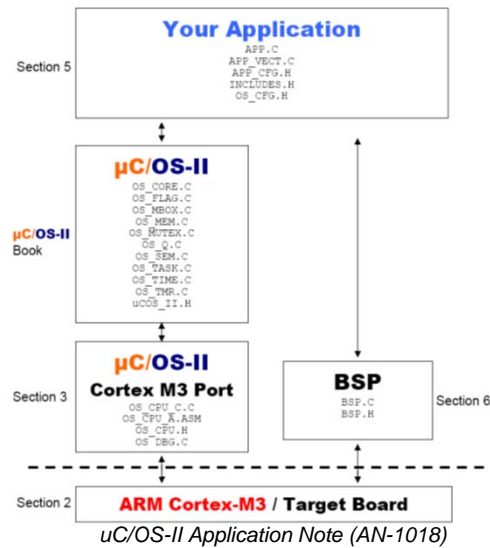Lecture 3   J. Valvano, A. Gerstlauer   13
                                    EE445M/EE380L.12

OS Architecture

- Portability
  - Small kernel
  - Hardware abstraction layer (HAL)
  - Common structure

uC/OS-II Application Note (AN-1018)

Lecture 3   J. Valvano, A. Gerstlauer   14
                                    EE445M/EE380L.12

OS Kernel

- Basic thread management
  - Maintain thread states
    - Running/ready/waiting
  - Context switch
    - Switch running thread
  - Protection
    - OS kernel from threads
    - Threads from each other

Lecture 3   J. Valvano, A. Gerstlauer   15
                                 EE445M/EE380L.12

ARM 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:

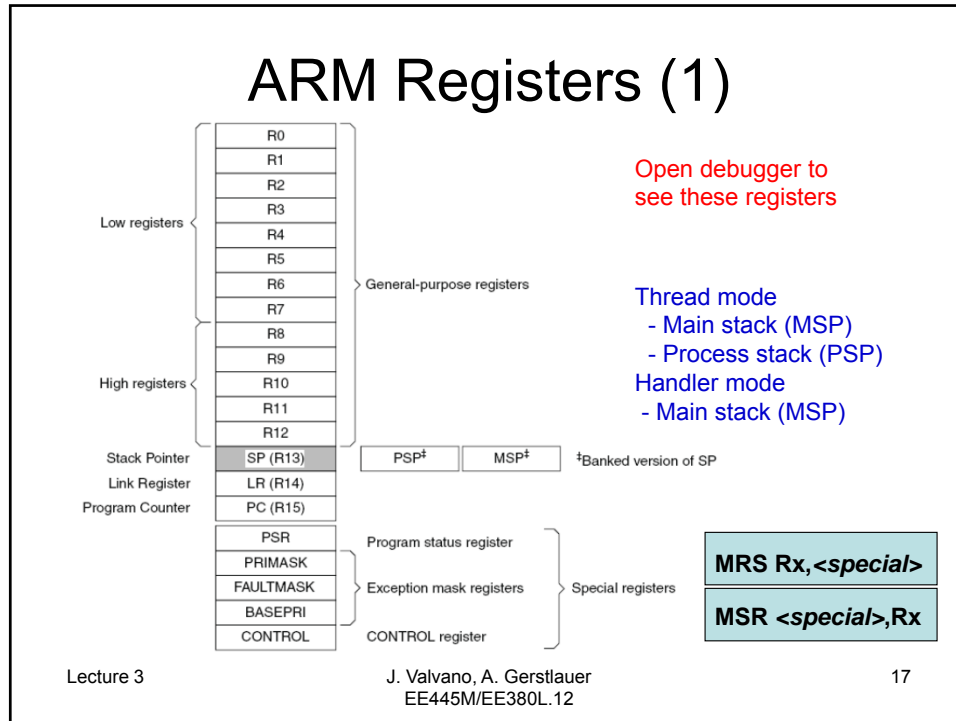|	
Unprivileged	The software:
	● Has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
	● Cannot access the system timer, NVIC, or system control block
	● Might have restricted access to memory or peripherals.
	Unprivileged software executes at the unprivileged level.
Privileged	The software can use all the instructions and has access to all resources.
	Privileged software executes at the privileged level.

Lecture 3   J. Valvano, A. Gerstlauer   16
                                 EE445M/EE380L.12

ARM Registers (1)

Low registers	R0
	R1
	R2
	R3
	R4
	R5
	R6
	R7

High registers	R8
	R9
	R10
	R11
	R12

General-purpose registers

Stack Pointer — SP (R13) | PSP‡ | MSP‡   ‡Banked version of SP
Link Register — LR (R14)
Program Counter — PC (R15)

PSR — Program status register
PRIMASK
FAULTMASK — Exception mask registers — Special registers
BASEPRI
CONTROL — CONTROL register

Open debugger to see these registers

Thread mode
 - Main stack (MSP)
 - Process stack (PSP)
Handler mode
 - Main stack (MSP)

MRS Rx,*<special>*

MSR *<special>*,Rx

Lecture 3 · J. Valvano, A. Gerstlauer · EE445M/EE380L.12 · 17

ARM Registers (2)

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).

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

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.

AAPCS:
R0-R3 parameters/return
R4-R11 must be saved

Which SP is active?

R14 is important

Lecture 3 · J. Valvano, A. Gerstlauer · EE445M/EE380L.12 · 18

Program Status Register (PSR)

Figure 3.   APSR, IPSR and EPSR bit assignments   Q = saturation
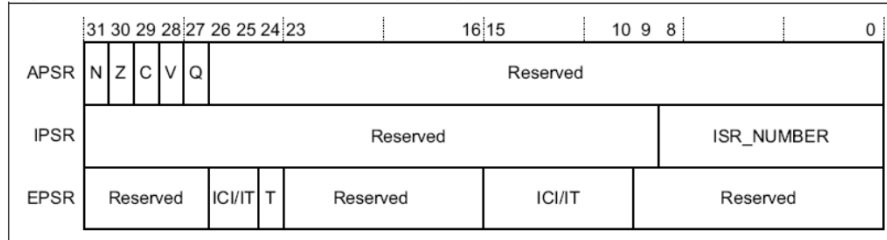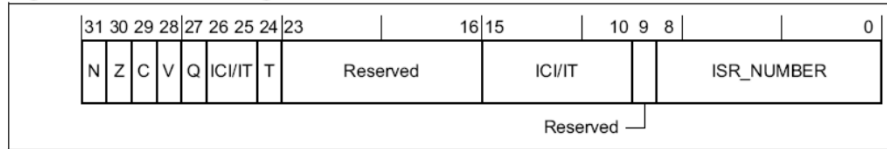
	31 30 29 28	27	26 25 24	23 ... 16	15 ... 10	9	8 ... 0
APSR	N Z C V	Q	Reserved				
IPSR	Reserved					ISR_NUMBER	
EPSR	Reserved	ICI/IT	T	Reserved	ICI/IT	Reserved	

Figure 4.   PSR bit assignments   T = Thumb bit

	31 30 29 28	27	26 25 24	23 ... 16	15 ... 10	9 8	... 0
	N Z C V	Q ICI/IT	T	Reserved	ICI/IT		ISR_NUMBER

Reserved ⌐

Lecture 3   J. Valvano, A. Gerstlauer   19
                                  EE445M/EE380L.12

Interrupt Program Status Register (IPSR)

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

31 ... 9	8 ... 0
Reserved	ISR NUMBER

Figure 2-3, The IPSR Register.

Lecture 3   J. Valvano, A. Gerstlauer   20
                                  EE445M/EE380L.12

Execution Program Status Register (EPSR)

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}

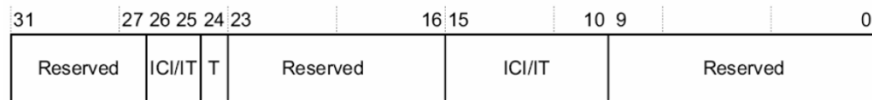• the execution state field for the If-Then (IT) instruction, and the T-bit (Thumb state bit).

31	27 26 25 24 23			16 15	10 9	0
Reserved	ICI/IT	T	Reserved	ICI/IT	Reserved	

Figure 2-4, The EPSR Register.

Lecture 3   J. Valvano, A. Gerstlauer   21
                                              EE445M/EE380L.12

Priority Mask Register

Disable interrupts (I=1)

CPSID   I

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.

Enable interrupts (I=0)

CPSIE   I

Figure 5.   PRIMASK bit assignments

31						1 0
Reserved						

PRIMASK ⎯

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.

StartCritical():

MRS R0, PRIMASK
CPSID   I

EndCritical():

MRS PRIMASK,R0

Lecture 3   J. Valvano, A. Gerstlauer
                                              EE445M/EE380L.12

Code from uC/OS-II

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

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

Where is the I bit saved?

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

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

Lecture 3   J. Valvano, A. Gerstlauer   23
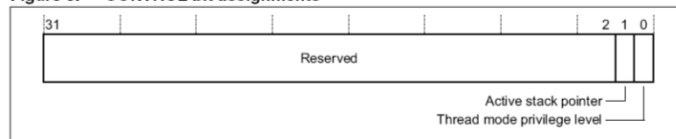   EE445M/EE380L.12

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

Lecture 3   J. Valvano, A. Gerstlauer   24
                                   EE445M/EE380L.12

Exception Processing

Exception number	IRQ number	Offset	Vector
83	67	0x014C	IRQ67
.		.	.
.		.	.
.		.	.
18	2	0x004C	IRQ2
17	1	0x0048	IRQ1
16	0	0x0044	IRQ0
15	-1	0x0040	Systick
14	-2	0x003C	PendSV
13		0x0038	Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			
9			Reserved
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

Remember Systick is 15

Stacking

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

Stack (8 regs):
● 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
  0b1110xxxx means floating point
aligned to double-word address

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

Lecture 3   J. Valvano, A. Gerstlauer   25
EE445M/EE380L.12

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 2-8, Exception Types (TM4C123GH6PM Data Sheet)

Lecture 3   J. Valvano, A. Gerstlauer   26
EE445M/EE380L.12

Supervisor Call (svc)

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)
```

Lecture 3   J. Valvano, A. Gerstlauer   27
                             EE445M/EE380L.12

Code from uC/OS-II

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

```
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         ; not necessary, PSR will be popped
   BX    LR
```

Lecture 3   J. Valvano, A. Gerstlauer   28
                             EE445M/EE380L.12

Summary

- Threads are executing software tasks
- RTOS has unique requirements
  - Reliability
  - Real-Time
  - Priority
  - Certification
  - Runs in ROM

Lecture 3   J. Valvano, A. Gerstlauer   29
                                    EE445M/EE380L.12