

EE445M/EE360L.6

Embedded and Real-Time Systems/ Real-Time Operating Systems

Lecture 3: RTOS, Threads, OS Kernel, Context Switch

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

1

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.

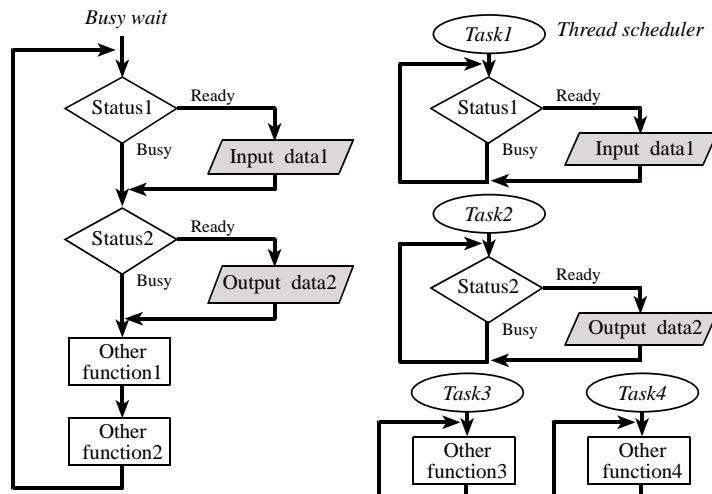
Embedded Microcomputer Systems: Real Time Operating Systems for ARM Cortex-M Microcontrollers, Jonathan W. Valvano (Ch. 3 & 4)

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

2

Single- vs. Multi-Thread



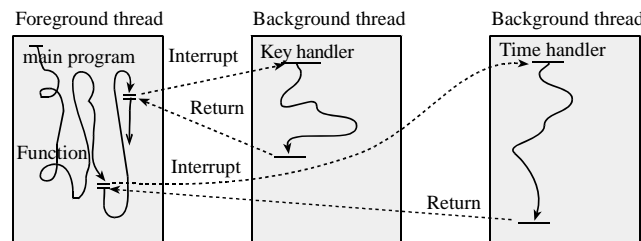
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

3

Interrupt-Based Threading

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



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

4

Threads and Tasks

```
void Producer(void) {
    unsigned short data;
    for(;;) {
        data = ADC_In();
        if(OS_Fifo_Put(data) == 0)
            DataLost++;
    }
}
```

```
void Display(void) {
    unsigned long data,voltage;
    for(;;){
        data = OS_MailBox_Recv();
        voltage = 31*data/64;
        LCD_Message(0,"v(mV) =",voltage);
    }
}
```

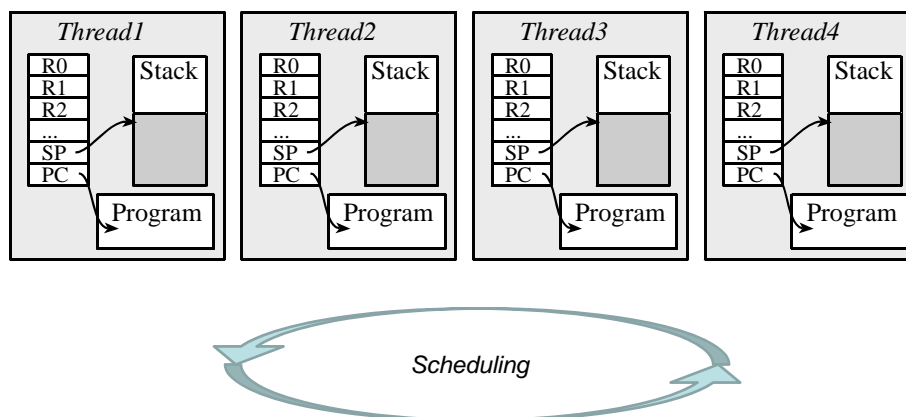
```
void Consumer(void) {
    unsigned short data,average;
    unsigned long sum;
    unsigned short n;
    for(;;) {
        sum = 0;
        for(n = 0; n < LENGTH; n++) {
            data = OS_Fifo_Get();
            sum = sum + data;
        }
        average = sum/LENGTH;
        OS_MailBox_Send(average);
    }
}
```

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

5

Multi-Tasking



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

6

Real-Time Operating System (RTOS)

- Thread management & scheduling
- Thread communication & synchronization
- Time management

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

7

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

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

8

Real-Time

- RT threads have deadlines
 - Hard real-time
 - Guaranteed bounded latency
 - Soft real-time
 - Occasional deadline miss can be tolerated
 - Not real-time
 - Best effort, no deadlines whatsoever

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

9

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
EE445M/EE380L.6

10

Time Management

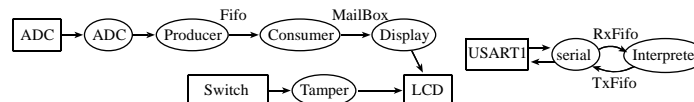
- System time
- Time stamps
 - When did it occur?
 - Performance measures
- Thread sleeping
 - Yield and wakeup after certain delay
 - Run other tasks instead of busy waiting
- Measurements
 - Input capture period -> wheel RPM
 - Input capture PW -> ultrasonic distance

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

11

Thread Communication



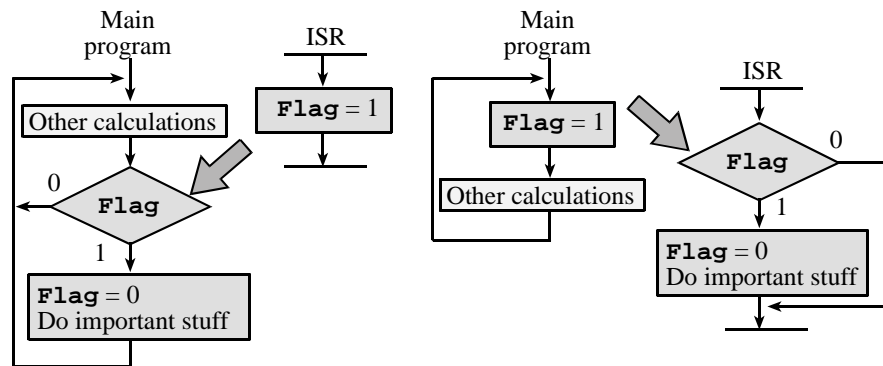
- Types
 - Data sharing (global variable)
 - Pipes=FIFO (one to one, buffered, ordered)
 - Mailbox (one to one, unbuffered)
 - Messages (many to many)
- Performance measures
 - Latency
 - Bandwidth
 - Error rate

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

12

Flag Semaphore

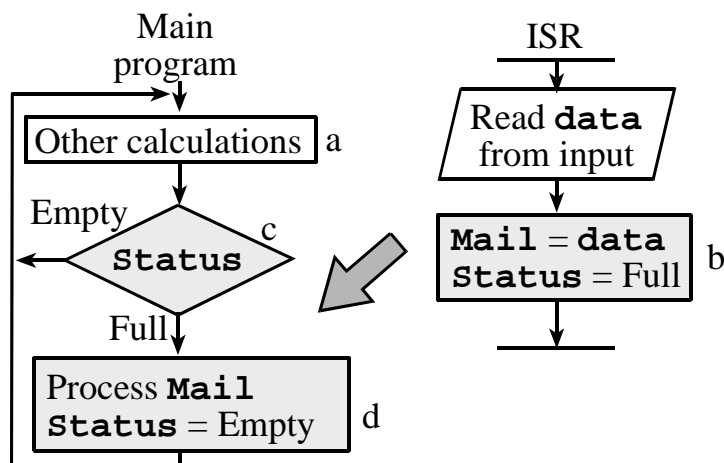


Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

13

Mailbox

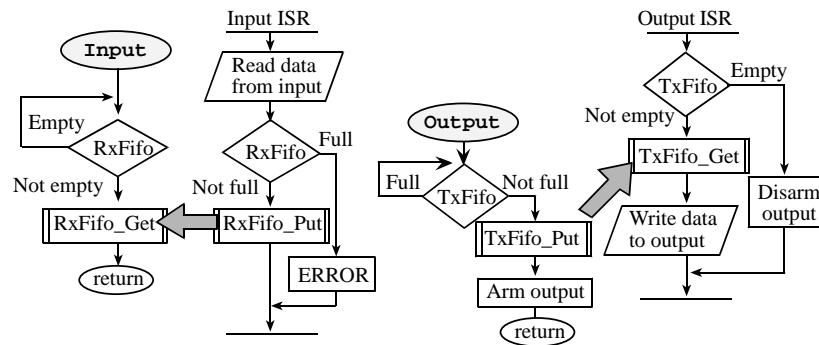


Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

14

FIFO



UARTInts_4C123.zip

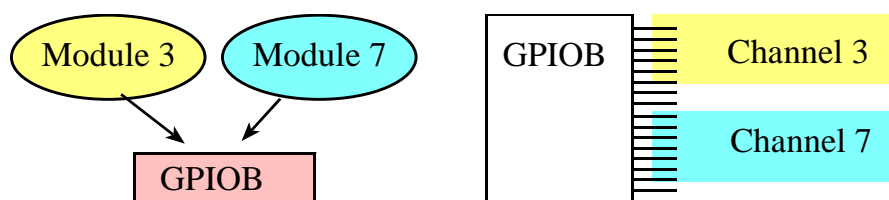
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

15

Race, Critical Section

- Two or more threads access the same global
 - Permanently allocated shared resource (memory, I/O port, ...)
- At least one access is a write



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

16

Race Condition

- Timing bug
 - Result depends on the sequence of threads
 - E.g. two threads writing to the same global
- Hard to debug
 - Depends on specific order/interleaving
 - Non-deterministic (external events)
 - Hard to reproduce/stabilize (“Heisenbug”)
- Critical or non-critical
 - Final program output affected?

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

17

Critical Section

- Load/store architecture
 - Write access changes official copy
 - Read access creates two copies
 - Original copy in memory
 - Temporary copy in register
- Non-atomic access sequence
 - Begins/ends with access to permanent resource
 - Involves at least one write
 - RMW(+W), WW(+R/W), WR(+W), RR(+W)

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

18

Thread-Safe, Reentrant

- Thread-safe code
 - No global resources
 - Variables in registers, stack
 - No critical section
 - No write access sequence
 - Mutual exclusion
 - Make accesses atomic (no preemption)
 - Prevent other threads from entering critical section
- Reentrant code
 - Multiple threads can (re-)enter same section
 - No non-atomic RMW, WW, WR sequence

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

19

Mutual Exclusion

- Disable all interrupts
 - Make atomic
- Lock the scheduler
 - No other foreground threads can run
 - Background ISR will occur
- Mutex semaphore
 - Blocks other threads trying to access info
 - All nonrelated operations not delayed
 - Thread-safe, but not reentrant

Measure time with interrupts disabled

- Maximum time
- Total time

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

20

Thread Synchronization

- Sequential
- Rendezvous, Barrier
 - Fork/spawn & join
- Trigger, event flags
 - OR, AND
 - I/O event (e.g., I/O edge, RX, TX)
- Time
 - Periodic time triggered (e.g., TATOMIS)
 - Sleep

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

21

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
EE445M/EE380L.6

22

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

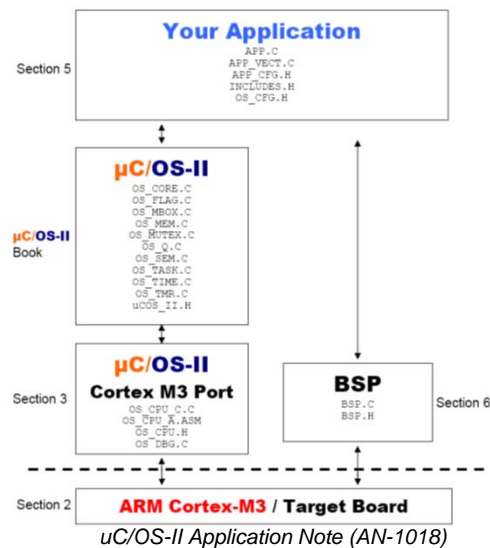
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

23

OS Architecture

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



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

24

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
EE445M/EE380L.6

25

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:

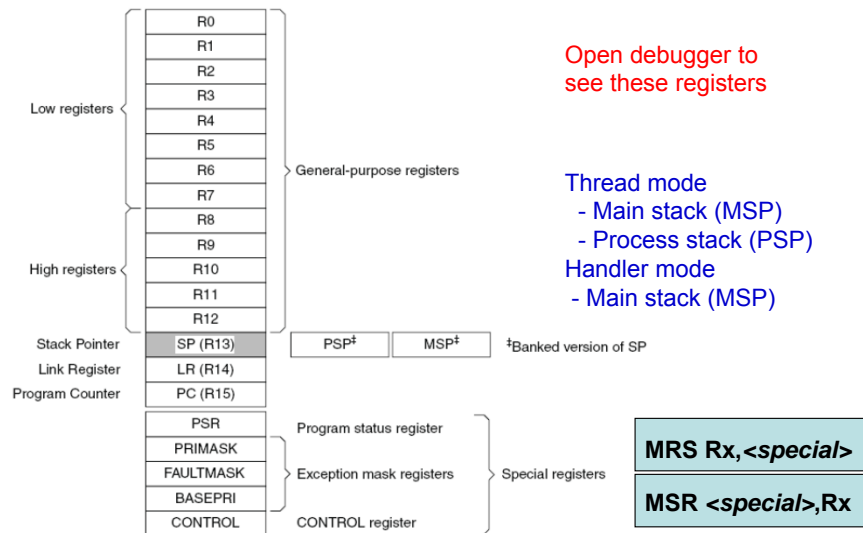
Unprivileged	<p>The software:</p> <ul style="list-style-type: none"> • Has limited access to the MSR and MRS instructions, and cannot use the CPS instruction • Cannot access the system timer, NVIC, or system control block • Might have restricted access to memory or peripherals. <p><i>Unprivileged software</i> executes at the unprivileged level.</p>
Privileged	<p>The software can use all the instructions and has access to all resources.</p> <p><i>Privileged software</i> executes at the privileged level.</p>

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

26

ARM Registers (1)



Open debugger to see these registers

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

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

27

ARM Registers (2)

General-purpose registers

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

AAPCS:

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

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

R14 is important

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.

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

28

Program Status Register (PSR)

Figure 3. APSR, IPSR and EPSR bit assignments

Q = saturation

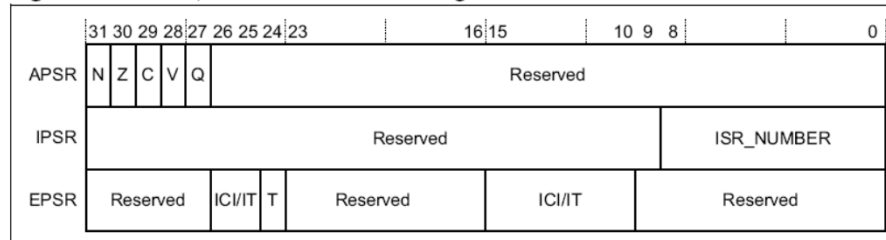
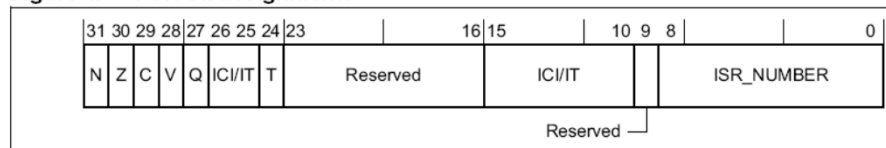


Figure 4. PSR bit assignments

T = Thumb bit



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

29

Interrupt Program Status Register (IPSR)

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




Figure 2-3,

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



Figure 2-3, The IPSR Register.

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

30

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).

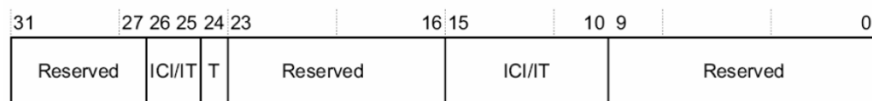


Figure 2-4, The EPSR Register.

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

31

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

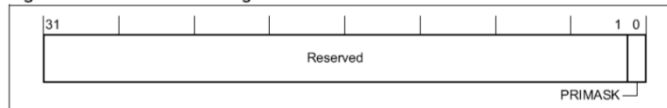


Table 7. PRIMASK register bit definitions

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

Disable interrupts (I=1)

CPSID 1

Enable interrupts (I=0)

CPSIE I

StartCritical():

MRS R0, PRIMASK
CPSID 1

EndCritical():

MRS PRIMASK,R0

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

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
EE445M/EE380L.6

33

CONTROL Register

Figure 8. CONTROL bit assignments

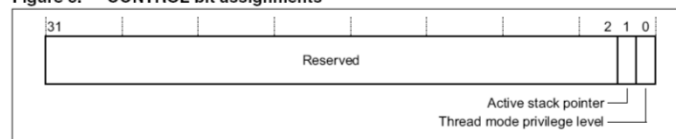


Table 10. CONTROL register bit definitions

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
EE445M/EE380L.6

34

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
EE445M/EE380L.6

35

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	.	.	Reserved
9	.	.	.
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

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

Stack (8 regs):

- R0-R3, R12
- LR
- Return address
- PSR

Run debugger:

- stop in ISR and
- look at LR
- draw stack frame

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

36

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
EE445M/EE380L.6

37

Code from uC/OS-II

```
NVIC_PENDSVSET EQU 0x10000000
NVIC_INT_CTRL EQU 0xE00ED04
```

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
```

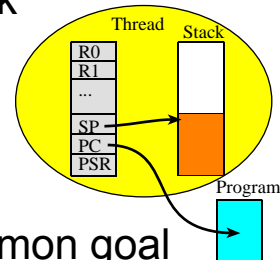
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

38

Thread or Light-Weight Process

- Execution of a software task
- Has its own registers
- Has its own stack
- Local variables are private
- Threads cooperate for common goal
- Private global variables
 - Managed by the OS
 - Allocated in the TCB (e.g., **td**)

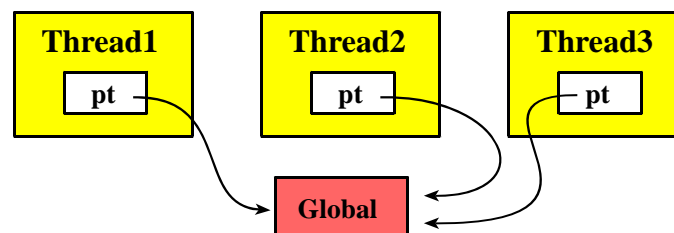


Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

39

Thread Communication/Sharing



- Shared Globals
- Mailbox (Lab 2)
- FIFO queues (Lab 2)
- Message (Lab 6)

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

40

Thread Control Block (TCB)

- Id
- Stack pointer
- Sleep counter
- Blocked pt (Lab 3)
- Priority (Lab 3)
- Next or Next/Previous links

Where are the registers saved?

```
struct TCB {
    // order??. types??
};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
```

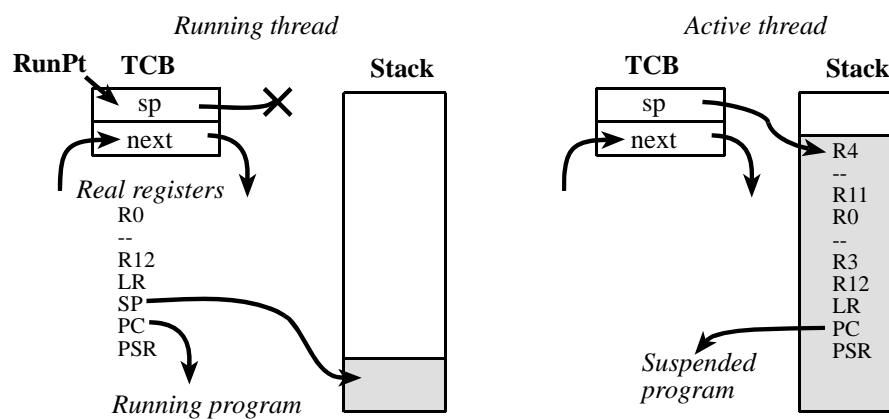
Look at TCB of uC/OS-II, struct `os_tcb` in
Micrium\Software\uCOS-II\Source\ucos_ii.h

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

41

Thread Switch



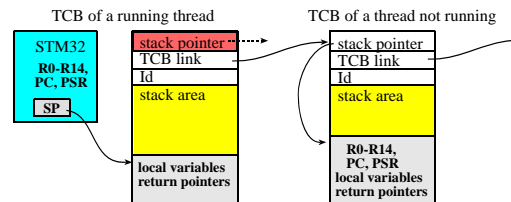
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

42

PendSV Thread Switch (1)

- PendSV handler
 - Give PendSV handler lowest priority
 - Prevent switching out background tasks
 - Use C code to find next thread



- Trigger PendSV

```
NVIC_INT_CTRL_R = 0x10000000;
```

Page 160 of tm4c123gh6pm.pdf

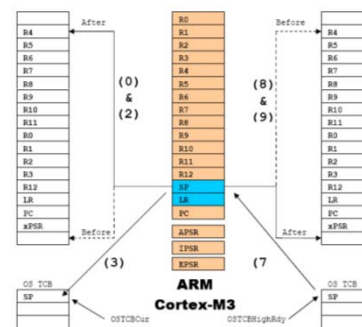
```

NVIC_INT_CTRL EQU 0xE00ED04
NVIC_PENDSVSET EQU 0x10000000
ContextSwitch
    LDR    R0, =NVIC_INT_CTRL
    LDR    R1, =NVIC_PENDSVSET
    STR    R1, [R0]
    BX     LR
  
```

PendSV Thread Switch (2)

- 1) Disable interrupts
- 2) Save registers R4 to R11 on the user stack
- 3) Save stack pointer into TCB
- 4) Choose next thread
- 5) Retrieve new stack pointer
- 6) Restore registers R4 to R11
- 7) Reenable interrupts
- 8) Return from interrupt

Run *Testmain1*
 -Show TCB chain
 -Show stacks
 -Explain switch



Micrium\Software\uCOS-II\Ports\ARM-Cortex-M3\Generic\RealView\os_cpu_a.asm

Assembly Thread Switch

```

SysTick_Handler      ; 1) Saves R0-R3,R12,LR,PC,PSR
    CPSID I           ; 2) Make atomic
    PUSH {R4-R11}     ; 3) Save remaining regs r4-11
    LDR R0, =RunPt     ; 4) R0=pointer to RunPt, old
    LDR R1, [R0]       ; R1 = RunPt
    STR SP, [R1]       ; 5) Save SP into TCB
    LDR R1, [R1,#4]    ; 6) R1 = RunPt->next
    STR R1, [R0]       ; RunPt = R1
    LDR SP, [R1]       ; 7) new thread SP; SP=RunPt->sp;
    POP {R4-R11}      ; 8) restore regs r4-11
    CPSIE I           ; 9) tasks run enabled
    BX LR             ; 10) restore R0-R3,R12,LR,PC,PSR
  
```

Program 4.9

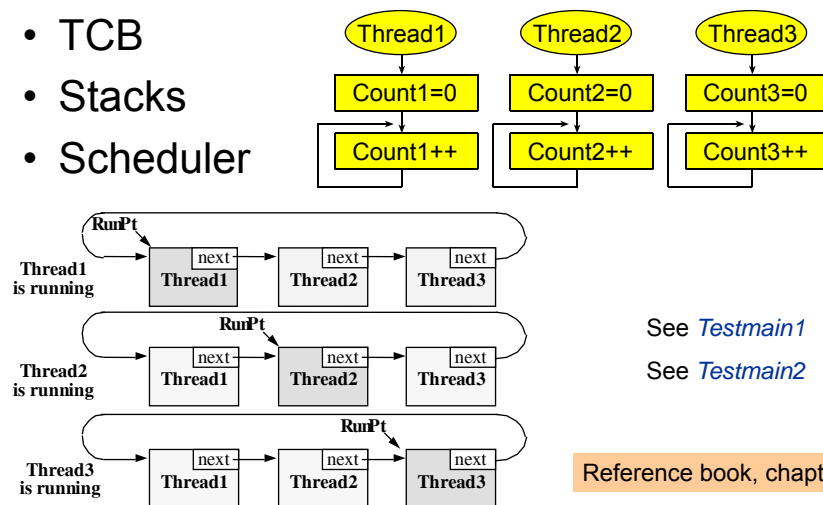
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

45

Thread Management

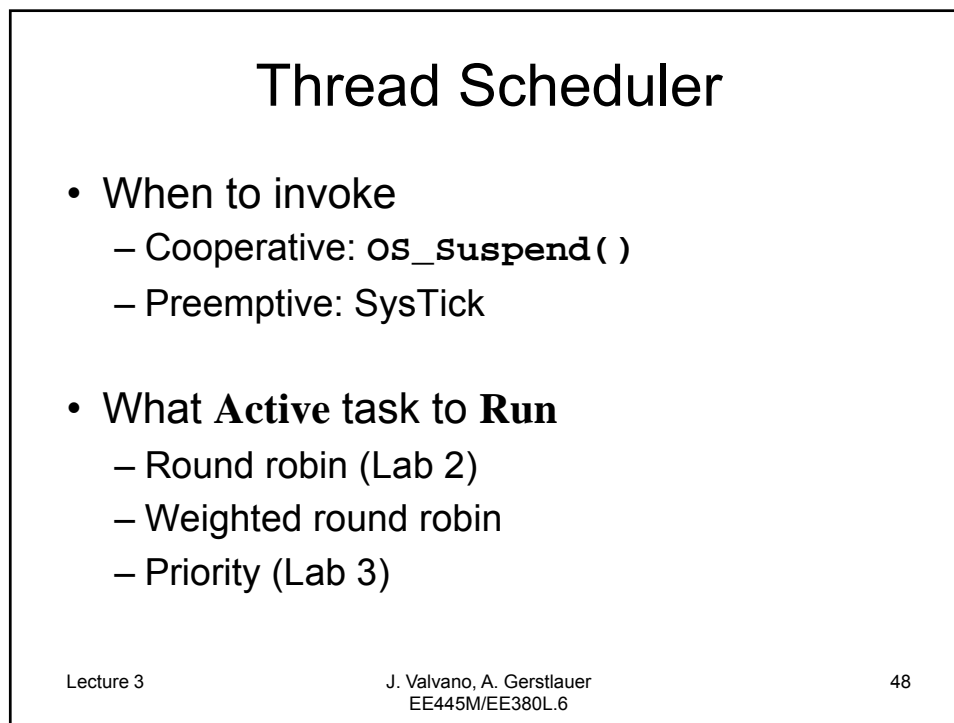
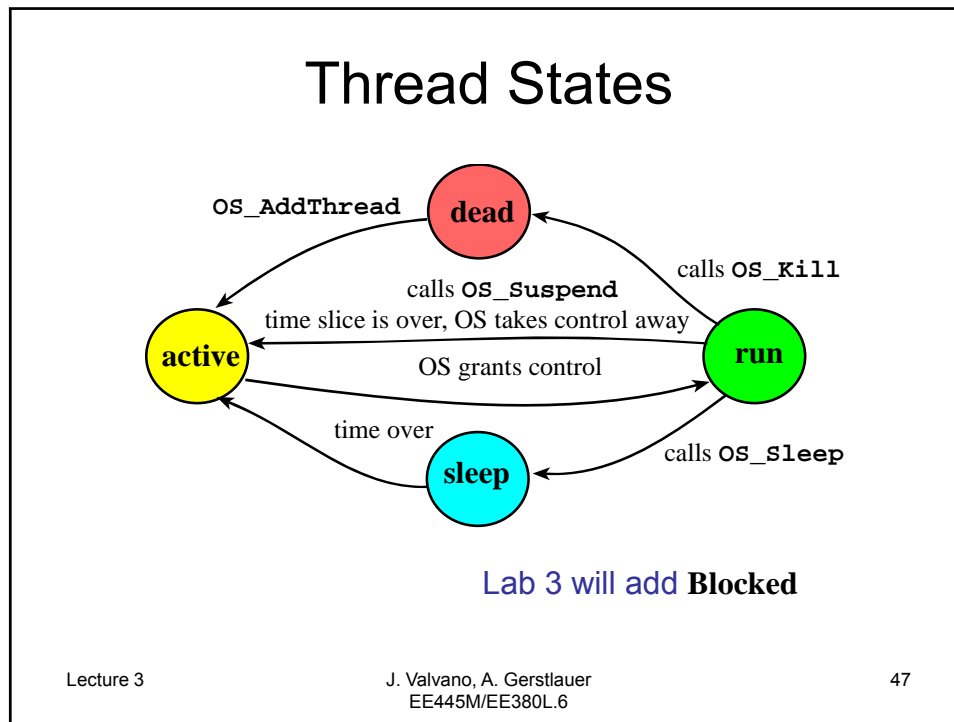
- TCB
- Stacks
- Scheduler



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

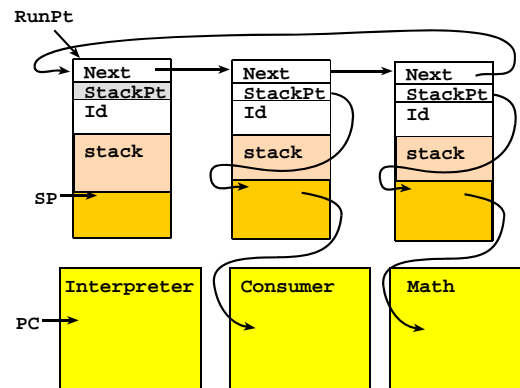
46



Round Robin Scheduler

```
OS_AddThread(&Interpreter);
OS_AddThread(&Consumer);
OS_AddThread(&Math);
OS_Launch(TIMESLICE); // doesn't return
```

RunPt



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

49

Decisions

- MSP/PSP or MSP?
 - Trap or regular function call?
 - How do you link OS to user code?
 - Protection versus speed?
 - Check for stack overflow
 - Check for valid parameters

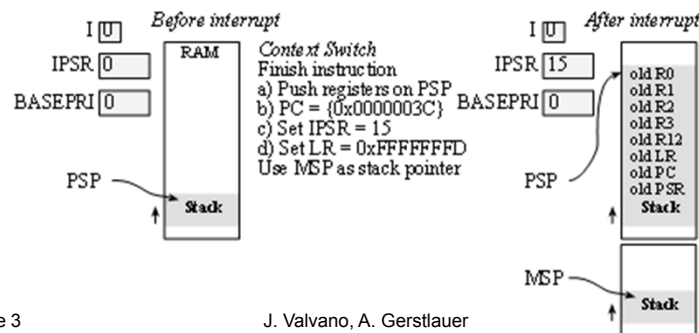
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

50

Thread Switch with PSP (1)

- Bottom 8 bits of LR
 - 0xE1 11110001 Return to Handler mode MSP (using floating point state)
 - 0xE9 11101001 Return to Thread mode MSP (using floating point state)
 - 0xED 11101101 Return to Thread mode PSP (using floating point state)
 - 0xF1 11110001 Return to Handler mode MSP
 - 0xF9 11111001 Return to Thread mode MSP
 - 0xFD 11111101 Return to Thread mode PSP**



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

51

Thread Switch with PSP (2)

```

; everyone uses MSP (Program 4.9 from book)
SysTick_Handler
    CPSID    I           ; 1) Saves R0-R3,R12,LR,PC,PSR
    PUSH     {R4-R11}    ; 2) Prevent interrupt during switch
    LDR      R0, =RunPt   ; 3) Save remaining regs r4-11
    LDR      R1, [R0]     ; 4) R0=pointer to RunPt, old thread
    STR      SP, [R1]     ; 5) Save SP into TCB
    LDR      R1, [R1,#4]  ; 6) R1 = RunPt->next
    STR      R1, [R0]     ; RunPt = R1
    LDR      SP, [R1]     ; 7) new thread SP; SP = RunPt->sp;
    POP      {R4-R11}    ; 8) restore regs r4-11
    CPSIE    I           ; 9) run with interrupts enabled
    BX      LR           ; 10) restore R0-R3,R12,LR,PC,PSR
  
```

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

52

Thread Switch with PSP (3)

```

; tasks use PSP, OS/ISR use MSP, Micrium OS-II
SysTick_Handler          ; 1) R0-R3,R12,LR,PC,PSR on PSP
    CPSID    I            ; 2) Prevent interrupt during switch
    MRS      R2, PSP       ; R2=PSP, the process stack pointer
    SUBS     R2, R2, #0x20
    STM      R2, {R4-R11} ; 3) Save remaining regs r4-11
    LDR      R0, =RunPt    ; 4) R0=pointer to RunPt, old thread
    LDR      R1, [R0]      ; R1 = RunPt
    STR      R2, [R1]      ; 5) Save PSP into TCB
    LDR      R1, [R1,#4]   ; 6) R1 = RunPt->next
    STR      R1, [R0]      ; RunPt = R1
    LDR      R2, [R1]      ; 7) new thread PSP in R2
    LDM      R2, {R4-R11} ; 8) restore regs r4-11
    ADDS     R2, R2, #0x20
    MSR      PSP, R2       ; Load PSP with new process SP
    ORR      LR, LR, #0x04 ; 0xFFFFFFFF (return to thread PSP)
    CPSIE    I            ; 9) run with interrupts enabled
    BX       LR           ; 10) restore R0-R3,R12,LR,PC,PSR

```

MSP active,
LR=0xFFFFFFFF

Lecture 3

OS calls implemented with trap (SVC)

53

NVIC

- Set priorities
 - PendSV low
 - Timer1 high
- Trigger PendSV

NVIC_INT_CTRL_R

Page 160 of tm4c123gh6pm.pdf

Launch

- Set SysTick period
- Set PendSV priority
- Using RunPt
 - Pop initialize Reg
- Enable interrupts
- Branch to user

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

54

To do first (1) To do last (2)

- | | |
|--|---|
| <ul style="list-style-type: none"> • Debugging • Interrupts • OS_AddThread • Assembly • NVIC • PendSV • OS_Suspend • OS_Launch | <ul style="list-style-type: none"> • Stack size • FIFO size • Timer1 period • SysTick period • Semaphores • PSP <ul style="list-style-type: none"> – Just use MSP |
|--|---|

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

55

Lab 2 Part 1 (1)

- Debugging
 - How to breakpoint, run to, dump, heartbeat
- Interrupts
 - How to arm, acknowledge, set vectors
 - What does the stack look like? What is in LR?
- OS_AddThread
 - Static allocation of TCBs and Stack
 - Execute 1,2,3 times and look at TCBs and Stack
- Assembly
 - PendSV, push/pull registers, load and store SP
 - Enable, disable interrupts
 - Access global variables like RunPt

Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

56

Lab 2 Part 1 (2)

- NVIC
 - Arm/disarm, priority
- PendSV
 - How to trigger
 - Write a PendSV handler to switch tasks
- OS_Suspend (scheduler and PendSV)
- OS_Launch (*this is hard*)
 - Run to a line at the beginning of the thread
 - Make sure TCB and stack are correct

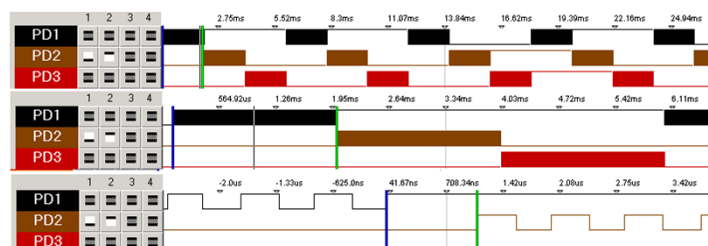
Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

57

Debugging tips

- Visualize the stacks
- Dumps and logs
- Logic analyzer



Lecture 3

J. Valvano, A. Gerstlauer
EE445M/EE380L.6

58

Summary

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