ECE254 Lab1 Tutorial

ARM RL-RTX Task Management

Irene Huang

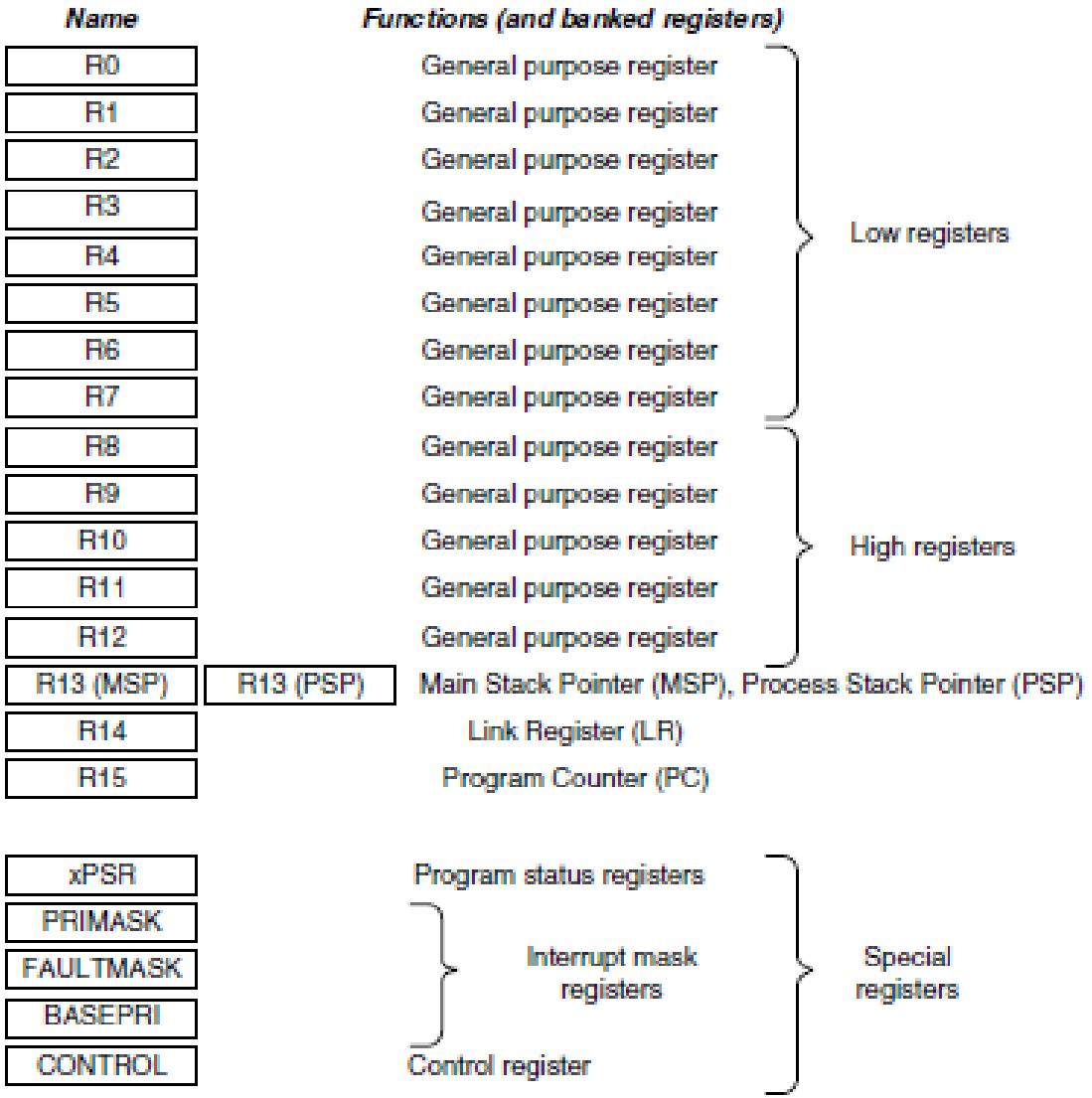
(last updated: 2013/09/30)

Cortex-M3 Registers

32-bit microprocessor
32-bit data path
32-bit register bank
32-bit memory interface
Harvard Architecture
Separate data and memory bus

Low registers: R0-R7
16-bit Thumb instructions
32-bit Thumb-2 instructions
High registers: R9-R12
All Thumb-2 instructions

MSP: default after reset
os kernel, exception handler
Privileged
PSP: base-level application
unprivileged, user-level

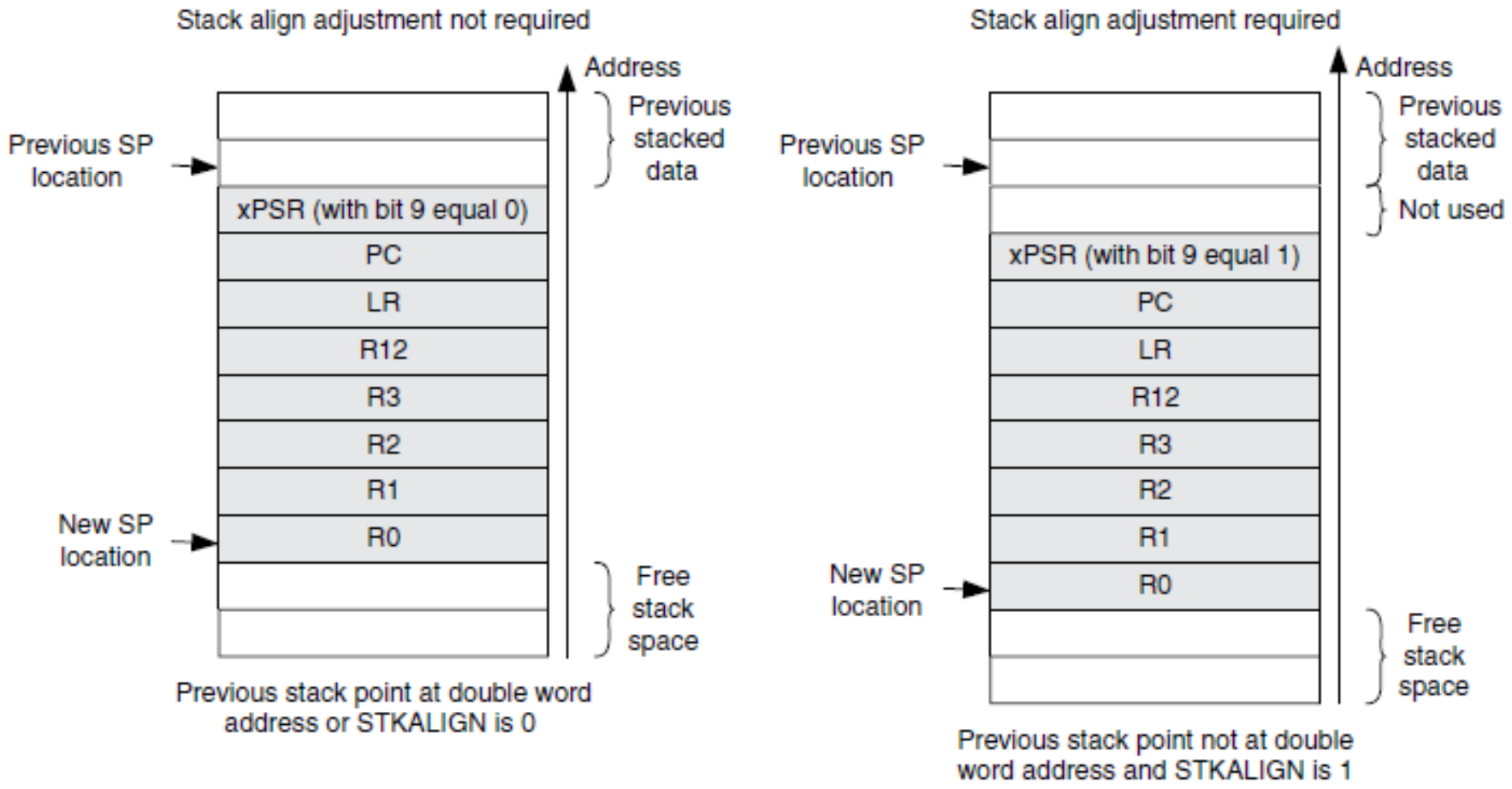


(Image Courtesy of [1])

AAPCS (ARM Architecture Procedure Call Standard)

- R0-R3 , R12
  - Input parameters Px of a function. R0=P1, R1=P2, R2=P3 and R3=P4
  - **R0** is used for **return value** of a function
- R12, SP, LR and PC
  - R12 is the Intra-Procedure-call scratch register.
- R4-R11
  - Must be preserved by the called function. C compiler generates push and pop assembly instructions to save and restore them automatically.

Exception Stack Frame



(Image Courtesy of [1])

Lab1 Part A Requirements (1)

- A function to obtain the task information

```
OS_RESULT os_tsk_get(OS_TID task_id, RL_TASK_INFO *buffer)
```

- Task information data structure

```
typedef struct rl_task_info {
  U8    state;          /* Task state                                  */
  U8    prio;           /* Execution priority                          */
  U8    task_id;        /* Task ID value for optimized TCB access      */
  U8    stack_usage;    /* Stack usage percent value. eg.=58 if 58%*/
  void (*ptask)();      /* Task entry address                          */
} RL_TASK_INFO;
```

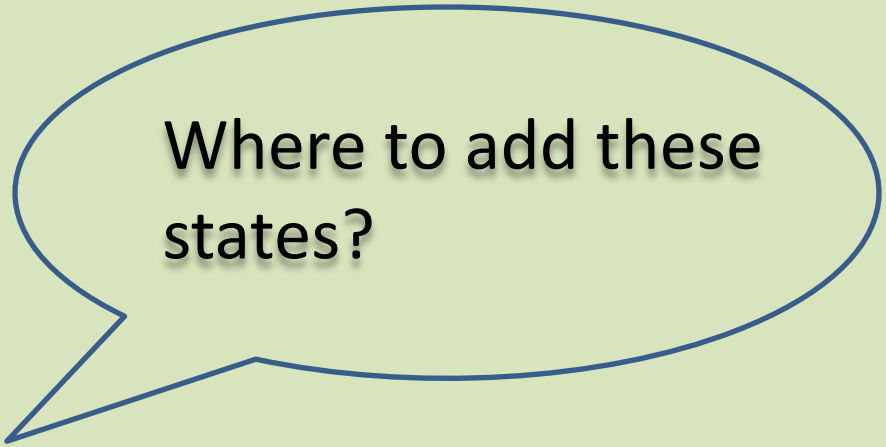
- Return value
  - OS_R_OK
  - OS_R_NOK

Where to define this data structure?

Lab1 Part A Requirements (2)

- Task state symbols

```
#define INACTIVE        0
#define READY           1
#define RUNNING         2
#define WAIT_DLY        3
#define WAIT_ITV        4
#define WAIT_OR         5
#define WAIT_AND        6
#define WAIT_SEM        7
#define WAIT_MBX        8
#define WAIT_MUT        9
#define WAIT_MEM        10
```



- **Assumption**
  - **No user defined stack in any tasks in the system.**

Task Stack

- Size of a task stack
  - **os_stackinfo** in RTX_Lib.c
- Starting address of each task stack
  - Stack initial setup is done by **rt_init_stack()** in HAL_CM3.c
- Stack pointer of a RUNNING task
  - **rt_get_psp()** in HAL_CM3.c
- Stack pointer of a non-RUNNING task
  - **Struct OS_TCB** {

```
    …
    U32   tsk_stack;
    U32   *stack;
    …
}
```

Lab1 Requirements Part B (1)

- **Assumption: only one memory pool defined by the user application**
- A blocking fixed-size memory allocator

```
void *os_mem_alloc(void *box_mem)
```

  - Input
    - Starting address of the memory pool
  - Return
    - A pointer to an available memory block
  - **Q: What if there is no memory block available?**

Lab1 Requirements Part B (2)

- **Assumption: only one memory pool defined by the user application**
- A function to release the memory block

```
OS_RESULT os_mem_free(void *box_mem, void *ptr)
```

  - Input:
    - Starting address of the memory pool
    - Pointer to the memory block to free
  - Return
    - `OS_R_OK`
    - `OS_R_NOK`
  - **Q: What if there are processes blocked waiting for memory?**

Use Existing Kernel Functions (1)

- In rt_MemBox.c
  - **`void *rt_alloc_box (void *p_mpool)`**
- A blocking memory allocator pseudo code

```
ptr = rt_alloc_box ();
If ptr is null, then
    adding the task to waiting list
    block the calling task
else
    return ptr
endif
```

Use Existing Kernel Functions (2)

- In rt_MemBox.c
  - **`int rt_free_box (void *p_mpool, void *box)`**
- Psedo code of a memory de-allocator that may unblock a task. Assume **box** points to the memory block to be freed.

```
If there are tasks waiting for memory, then
    remove a task from the waiting list
    set its TCB ret_val to box
    let os unblock the task
else
    return the result of rt_free_box()
endif
```

How to Create a Waiting List?

- The rt_List.c file
  - `rt_put_prio()`
  - `rt_get_first()`

- You may want to explore other functions in the file

Context Switching Kernel Functions

- In rt_Task.c
  - rt_block(): change TCB state
  - rt_dispatch(): dispatch the next to run task
- In HAL_CM3.c
  - SVC_Handler:
    - Save the current running task context
    - Restore the newly picked task context

```
__asm void SVC_Handler (void) {
  ;...
  BLX     R12                     ; Call SVC Function

  MRS     R12,PSP                 ; Read PSP
  LDR     R3,=__cpp(&os_tsk)
  LDM     R3,{R1,R2}              ; os_tsk.run, os_tsk.new
  CMP     R1,R2
  BEQ     SVC_Exit                ; no task switch

  CBZ     R1,SVC_Restore          ; Runtask deleted?

  PUSH    {R2,R3}
  MOV     R3,#1
  STRB    R3,[R1,#TCB_RETUPD]     ; os_tsk.run->ret_upd = 1
  STMDB   R12!,{R4-R11}           ; Save Old context
  STR     R12,[R1,#TCB_TSTACK]    ; Update os_tsk.run->tsk_stack
  BL      rt_stk_check            ; Check for Stack overflow
  POP     {R2,R3}
  ; omit the rest of the code below
}
```

```
__asm void SVC_Handler (void) {
  ...
SVC_Restore
  STR     R2,[R3]                ; os_tsk.run = os_tsk.new

  LDR     R12,[R2,#TCB_TSTACK]; os_tsk.new->tsk_stack
  LDMIA   R12!,{R4-R11}          ; Restore New Context
  LDRB    R3,[R2,#TCB_RETUPD] ; Update ret_val?
  MSR     PSP,R12                ; Write PSP

  CBZ     R3,SVC_Return
  LDR     R0,[R2,#TCB_RETVAL] ; Write os_tsk.new->ret_val, U32

SVC_Exit
        STR     R0,[R12]       ; Function return value

SVC_Return
        MVN     LR,#:NOT:0xFFFFFFFD     ; set EXC_RETURN value
        BX      LR
; omit the rest of the code below
}
```

Hints

- RL-RTX Kernel
  - Kernel data structure in `rt_TypeDef.h`
    - `Struct OS_TCB`
    - `Struct OS_XCB`
    - `Struct OS_TSK`
  - Task management in `rt_Task.c`
    - `rt_block(U16, U8)`
    - `rt_dispatch(P_TCB)`
    - `os_active_TCB`
    - `os_idle_TCB`
    - `os_tsk`
    - `os_rdy`
  - Read the `rt_Mailbox.c` file `rt_mbx_send()` function to see how the return value of `rt_mbx_receive()` function is set when a task is found waiting for a message and unblocked.

References

1. Yiu, Joseph, *The Definite Guide to the ARM Cortex-M3*, 2009
2. *RealView® Compilation Tools Version 4.0 Developer Guide*
3. *ARM Software Development Toolkit Version 2.50 Reference Guide*
4. *LPC17xx User's Manual*

Questions