CORTEX-M4/-M3 PROCESSORS(1)

마이크로프로세서응용

Handong Global University

References

- Joseph Yiu, The definitive Guide to the ARM Cortex-M3 and Cortex-M4 Processors. 3rd Ed., Elsevier Inc., 2014.
- ARM Cortex-M4 Devices: Generic User Guide, ARM, 2010.
- ARM Cortex-M4 Processor Technical User's Manual, ARM, 2013.
- ARMv7-M Architecture Reference Manual, ARM, 2010.

Fundamentals

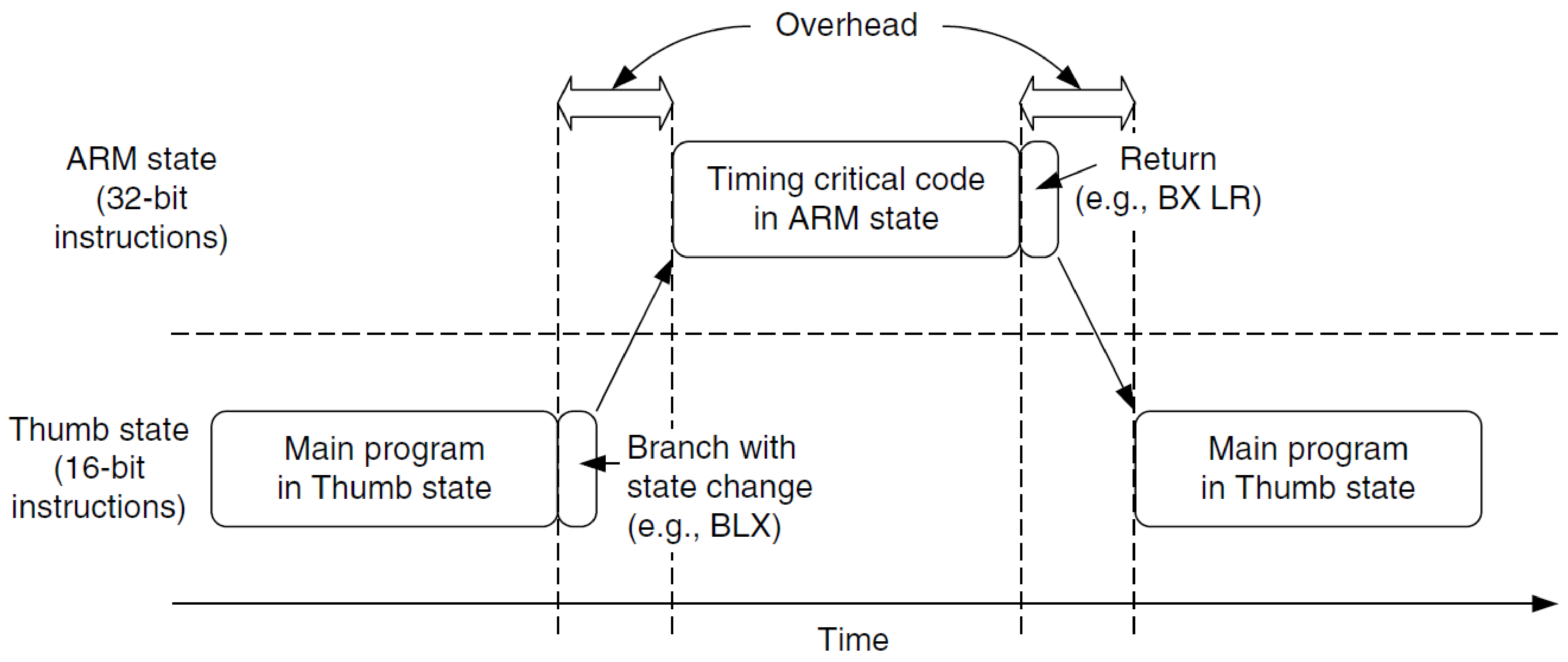
- ARMv7-M architecture
- A 32-bit microprocessor
  - ✓32-bit registers, a 32-bit data bus, and 32-bit memory interfaces
- A Harvard architecture
  - ✓A separate instruction bus and data bus
  - ✓However, a unified memory system
    - the instruction and data buses share the same memory space
- Supports both little endian and big endian memory system

Fundamentals

- Supports Thumb-2 instruction set
  - ✓A superset of 16-bit Thumb instructions that contains both 16-bit and 32-bit instructions
  - ✓Can mix 32-bit instructions with 16-bit instructions without switching state, getting high code density and high performance with no extra complexity
- HW divide and single cycle Multiply
- Advanced debugging components
- Optional MPU & ETM (Embedded Trace Macrocell)

Switching Between Two States

- Switching between ARM and Thumb in traditional ARM processors


Overhead
ARM state (32-bit instructions)
Timing critical code in ARM state
Return (e.g., BX LR)
Thumb state (16-bit instructions)
Main program in Thumb state
Branch with state change (e.g., BLX)
Main program in Thumb state
Time

Processor Modes

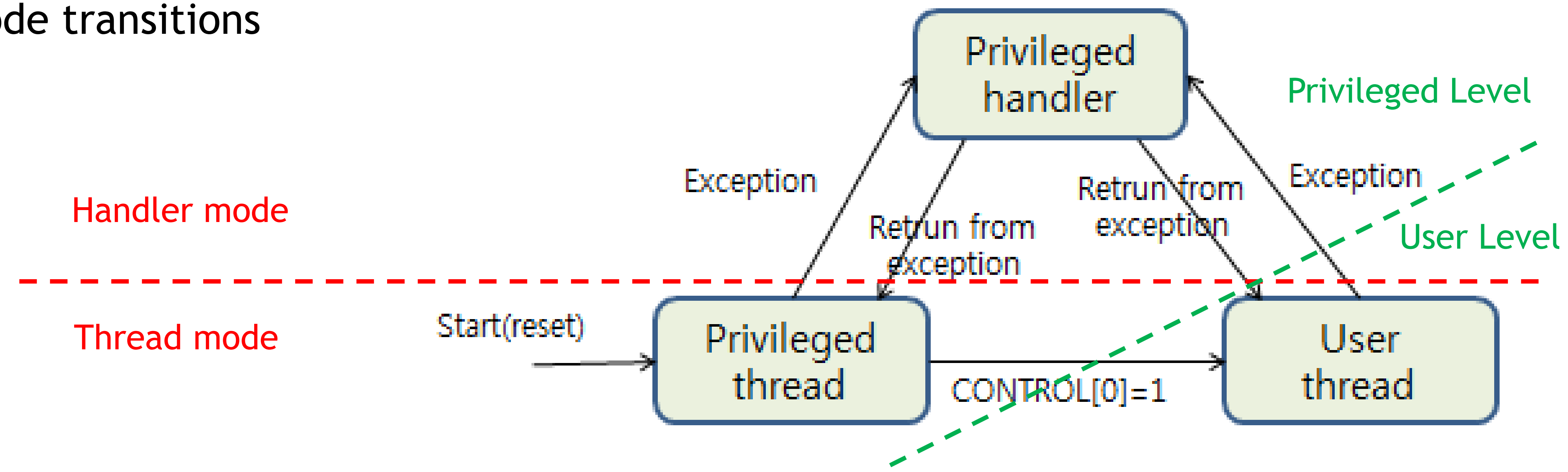
- Processor modes
  - Thread mode (Privileged or User)
    - ✓Used to execute Application Software
  - Handler mode (Privileged only)
    - ✓Used to handle Exceptions

Privilege Levels

- Privilege Levels
  - Privileged level
    - ✓can use all the instructions and access all memory ranges except for the ranges prohibited by MPU settings
  - Unprivileged (User) level
    - ✓Limited access to the MSR and MRS instructions, and cannot use the CPS(Change Processor State) instruction
    - ✓Cannot access the system timer, NVIC, or system control block
    - ✓Might have restricted access to memory or peripherals.

Processor Modes and Privilege

- Mode transitions



- Privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode.
- Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software

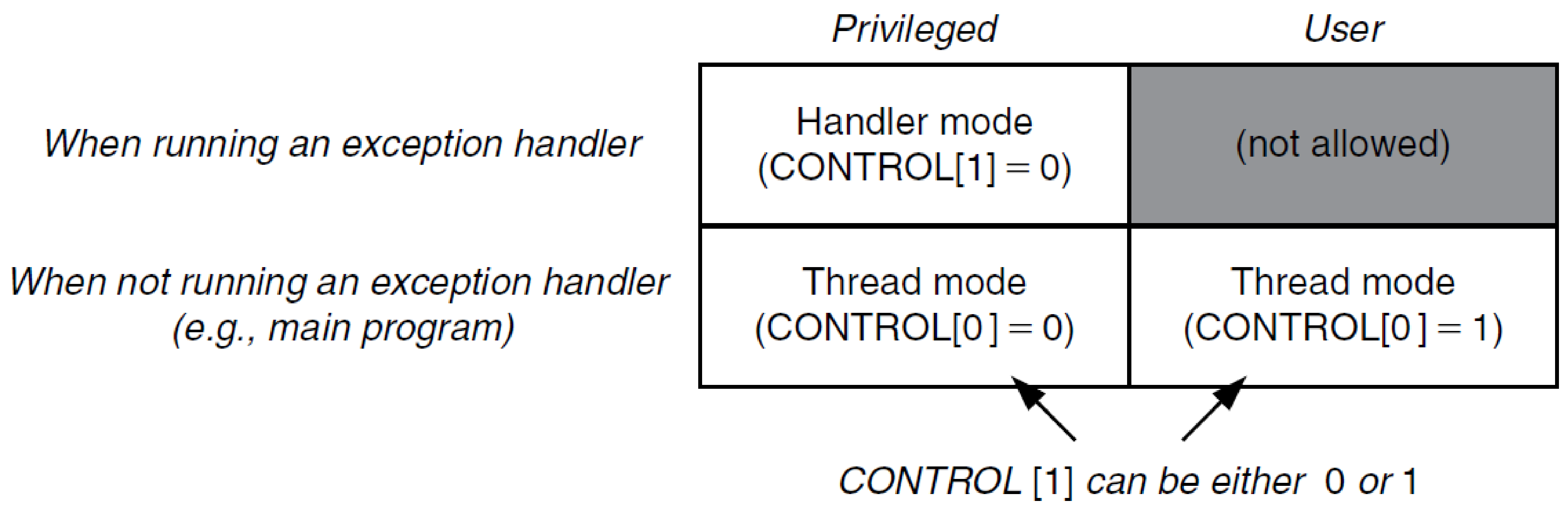
Processor Mode

- CONTROL register
  - A special register that controls the stack used and the privileged level in thread mode.
  - Writable only in privileged thread mode.
    - ✓Not allowed to write in the unprivileged or handler mode

Bits	Name	Description
[2]	FPCA	0=The instructions related to FPU are not used 1=The instructions related to FPU are used
[1]	SPSEL	0=default stack is used (MSP: main stack pointer) 1=process stack is used (PSP: process stack pointer)
[0]	nPRIV	0=privileged in thread mode 1=unprivileged in thread mode

Processor Mode

- The relationship with Control register

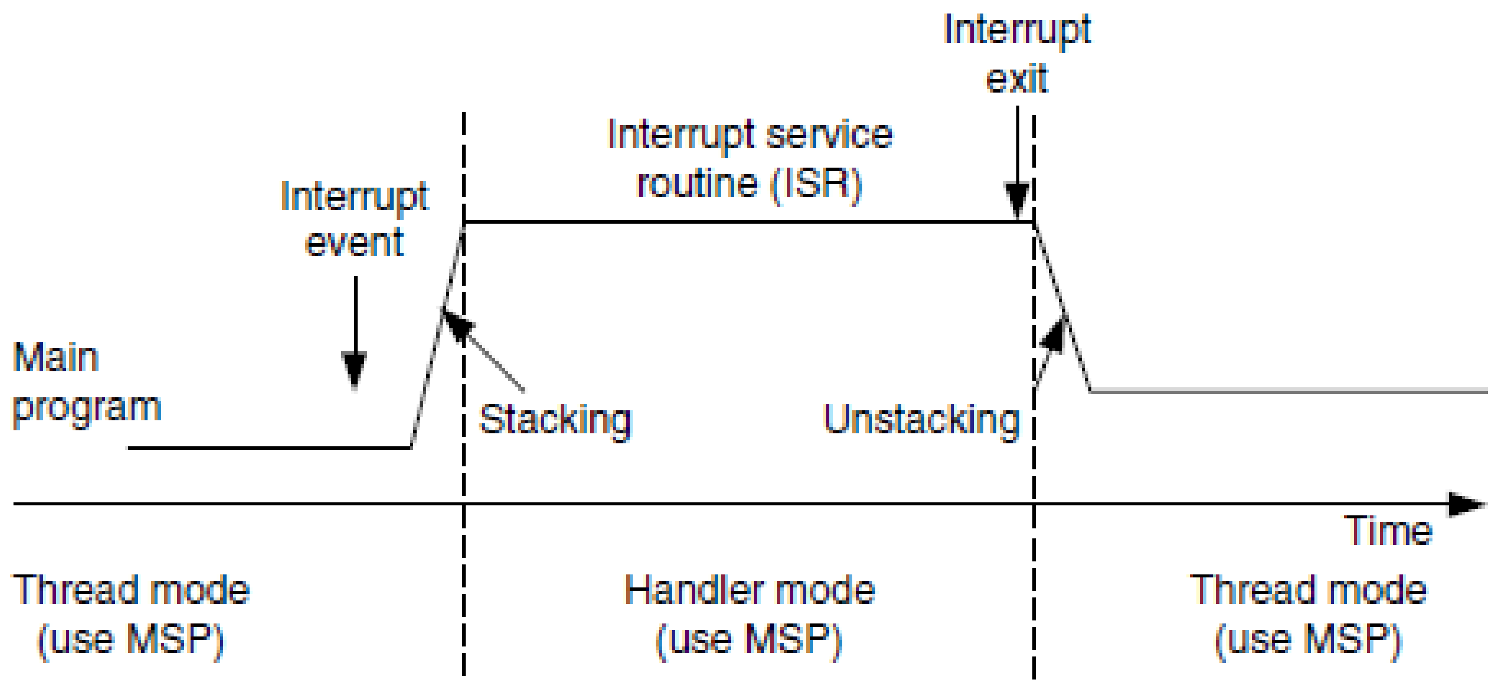

Privileged
User
When running an exception handler
Handler mode (CONTROL[1] = 0)
(not allowed)
When not running an exception handler (e.g., main program)
Thread mode (CONTROL[0] = 0)
Thread mode (CONTROL[0] = 1)
CONTROL [1] can be either 0 or 1

Stacks

- Two stacks: main stack and process stack
  - In thread mode, the processor can use the main stack or process stack
    - ✓By the SPSEL of the CONTROL register bit[1] (0: Main stack, 1: process stack)
  - In handler mode, the processor always use <u>the main stack.(control[1]=0)</u>
- Full descending stack
  - PUSH operation: the Sp decrements and then data is saved in the memory address of Sp.
  - POP operation: the data in the memory address of Sp is retrieved, and then the Sp increases.

Stacks

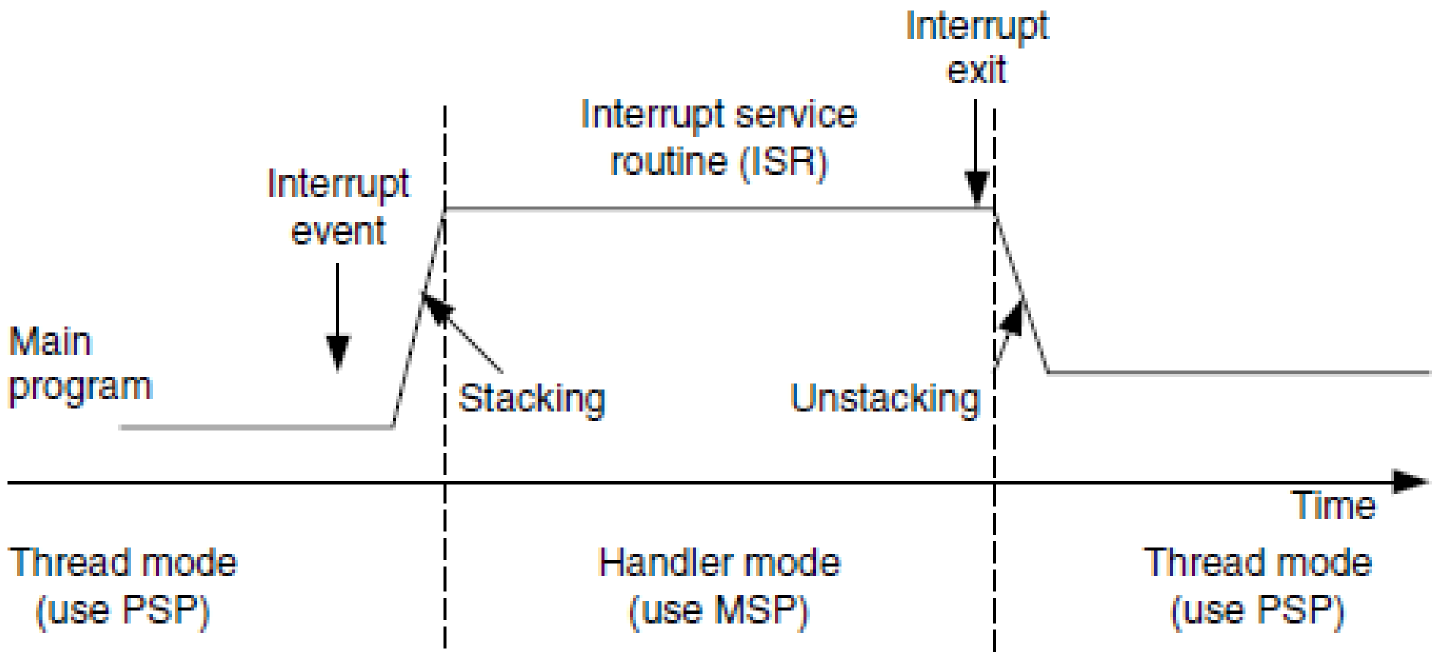
- In case that CONTROL[1] = 0 (main stack)



When an interrupt takes place,
a number of registers will be pushed automatically

Stacks

- In case that CONTROL[1] = 1 (process stack)



-By reading the PSP value using MRS instruction, the OS can read data stacked by the user application

Core Registers

Name		Type	Required Privileged	Reset value
R0 ~ R12		RW	Either	Unknown
R13 (SP)	MSP	RW		
	PSP	RW		Unknown
R14 (LR)		RW	Either	0xFFFFFFFF
R15 (PC)		RW	Either	
xPSR		RW	Privileged	
APSR		RW	Either	Unknown
IPSR		RO	Privileged	0x00000000
EPSR		RO	Privileged	0x01000000
PRIMASK		RW	Privileged	0x00000000
FAULTMASK		RW	Privileged	0x00000000
BASEPRI		RW	Privileged	0x00000000
CONTROL		RW	Privileged	0x00000000

Registers

Name	Functions
R0	General purpose
R1	General purpose
R2	General purpose
R3	General purpose
R4	General purpose
R5	General purpose
R6	General purpose
R7	General purpose
R8	General purpose
R9	General purpose
R10	General purpose
R11	General purpose
R12	General purpose
R13 (MSP) R13(PSP)	Stack Pointer (SP)
R14	Link Register (LR)
R15	Program Counter (PC)

Registers

- R0-R12
  - 32-bit general purpose registers
  - Some 16-bit Thumb instructions can only access low registers, R0-R7
    - ✓R0-R7: called low registers
    - ✓R8-R12: called high registers
  - The reset value is unpredictable

Registers

- R13: SP (stack pointer) (MSP or PSP)
  - R13 (SP) means the current SP. The other one can be accessed by special instructions MSR/MRS.
    - ✓MSP (Main SP): Default stack pointer used by the OS kernel and exception handler
    - ✓PSP (Process SP): Used by user applications
  - On reset, the processor loads the MSP with the value from address 0x00000000.
  - The lowest 2 bits of the SP are always 0. (word aligned)
  - Stack operation: full-descending stack

```
subroutine:
    PUSH    {R0-R7, R12, R14}
             …
    POP {R0-R7, R12, R14}
    BX  R14
```

Registers

- R14: LR (link register)
  - When a subroutine is called, the return address is stored in the link register
  - On reset LR=0xFFFFFFFF

```
main:
    …
    BL func1   ; LR=the address of
            ; the next instruction

func1:
    …
    BX R14 ; if the LR[0] is 0, it can imply
            ; trying to switch to the ARM
            ; mode and will result in a fault
            ; exception in the Cortex-M
```

Registers

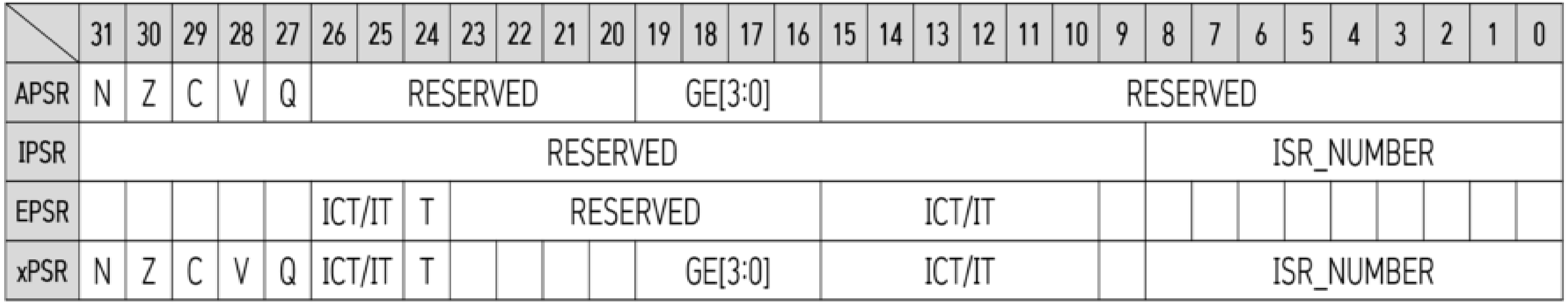
- R15: the program counter
  - Contains the current program address
  - Due to pipelined nature, the value of PC is normally advanced by 4 compared to the location of the executing instruction.
  - On reset, PC=the value of the reset vector, which is at address 0x00000004.

Special registers

Name	Function
xPSR	Arithmetic and logic processing flags, execution status, current interrupt number
PRIMASK	Disable all interrupts except the NMI and hard fault
FAULTMASK	Disable all interrupts except the NMI
BASEPRI	Disable all interrupts of specific priority level or lower priority level
CONTROL	Define privileged status and stack pointer selection

Special Registers (PSR)

- PSR (Program Status Register)
  - xPSR combines
    - ✓APSR (Application PSR)
    - ✓IPSR (Interrupt PSR)
    - ✓EPSR (Execution PSR)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
APSR	N	Z	C	V	Q	RESERVED							GE[3:0]				RESERVED															
IPSR	RESERVED																							ISR_NUMBER								
EPSR						ICT/IT		T	RESERVED								ICT/IT															
xPSR	N	Z	C	V	Q	ICT/IT		T					GE[3:0]				ICT/IT							ISR_NUMBER								

Special Registers (PSR)

- PSR (Program Status Register)
  - These can be accessed individually or a combination of any two or three registers

Name	Combination
xPSR	APSR, EPSR and IPSR
IEPSR	IPSR and EPSR
IAPSR	IPSR and APSR
EAPSR	EPSR and APSR

Special Registers

- Application PSR (APSR)
  - Contains the condition flags that result from a previous instruction execution
  - Possible to read/write it via MRS or MSR instruction

Bits	Name	Description
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry flag
[28]	V	Overflow flag
[27]	Q	Saturation flag
[19:16]	GE	Greater than or Equal flag

Special Registers (IPSR)

- Interrupt PSR (IPSR)
  - Contains the exception number of the current ISR
  - Ignores writes to the IPSR (read only)

Bits	Name	Description
[8:0]	ISR_NUMBER	The number of the current exception (0 = thread mode, 2 = NMI, 3 = Hard fault, 4 = MemManage, 5 = Bus fault, 6 = Usage fault, 7-10: reserved 11 = SVCall, 12-13 : reserved, 14 = PendSV, 15 = SysTick, 16 = external IRQ0, 17 = external IRQ1 …)

Special Registers (EPSR)

- Execution PSR (EPSR)
  - Contains the Thumb state and the execution state for
    - ✓If-Then (IT) instruction
    - ✓Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction
  - Ignores writes to the ESPR and read returns zero

Bits	Name	Description
[26:25][15:10]	ICI/IT	The interrupted position of a continuable instruction or The execution state of IT instruction (ITSTATE)
[24]	T	Thumb state

Special Registers (EPSR)

- Execution PSR (EPSR)
  - Interruptible-Continuable instructions
    - ✓When an interrupt occurs during the execution of an LDM , STM , PUSH, or POP instruction, the processor:
      - stops the load multiple or store multiple instruction operation temporarily
      - stores the next register operand in the multiple operation to EPSR bits[15:12].
    - ✓After servicing the interrupt, the processor:
      - returns to the register pointed to by bits[15:12]
      - resumes execution of the multiple load or store instruction.

EPSR[26:25]	EPSR[15:12]	EPSR[11:10]
ICI[7:6]=00	ICI[5:2]=reg_num	ICI[1:0]=00

Special Registers (EPSR)

- Execution PSR (EPSR)
  - ITSTATE
    - ✓The If-Then block contains up to four instructions following an IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others.
    - ✓IT[7:5] = the Base Condition for the current IT block (the top 3 bits of the condition specified by the IT instruction
    - ✓IT[4:0] = encodes (the size of IT block & the value of LSB of the conditional code)

EPSR[26:25]	EPSR[15:12]	EPSR[11:10]
IT[1:0]	IT[7:4]	IT[3:2]

Special Registers (EPSR)

- Examples of IT instruction

	Examples

One conditional instruction	`IT EQ`
	`ADDEQ R0, R0, R1`

Two conditional instruction	`ITE GE`
	`ADDGE R0, R0, R1`
	`ADDLT R0, R0, R3`

Three conditional instruction	`ITET GT`
	`ADDGT R0, R0, R1`
	`ADDLE R0, R0, R3`
	`ADDGT R2, R4, #1`

Four conditional instruction	`ITETT NE`
	`ADDNE R0, R0, R1`
	`ADDEQ R0, R0, R3`
	`ADDNE R2, R4, #1`
	`MOVNE R5, R3`

Special Registers (EPSR)

- Execution PSR (EPSR)
  - Codes for condition in ITSTATE

Code	condition	meaning	Code	condition	meaning
0000	EQ	Equal	1000	HI	Unsigned higher
0001	NE	Not equal	1001	LS	Unsigned lower or same
0010	CS	Carry set	1010	GE	Signed greater than or equal
0011	CC	Carry clear	1011	LT	Signed less than
0100	MI	minus	1100	GT	Signed greater than
0101	PL	Positive or zero	1101	LE	Signed less than or equal
0110	VS	Overflow	1110	AL	always
0111	VC	No overflow			

Special Registers (EPSR)

- Execution PSR (EPSR)
  - ITSTATE

IT[7:5]	IT[4]	IT[3]	IT[2]	IT[1]	IT[0]	description
Base condition	P1	P2	P3	P4	1	Entry point for 4-instruction IT block
Base condition	P1	P2	P3	1	0	Entry point for 3-instruction IT block
Base condition	P1	P2	1	0	0	Entry point for 2-instruction IT block
Base condition	P1	1	0	0	0	Entry point for 1-instruction IT block
000	0	0	0	0	0	Normal execution (not in an IT block)

Special Registers (PRIMASK)

- PRIMASK register
  - Used to Disable all Exceptions except NMI and Hard Fault.

Bits	Name	Function
[31:1]	-	Reserved
[0]	PRIMASK	0 = no effect 1 = disable all exceptions except NMI and hard fault

```
CPSIE I ; enable interrupt
CPSID I ; disable interrupt
```

```
; disable interrupt
MOV R0, #1
MSR PRIMASK, R0
; enable interrupt
MOV R0, #0
MSR PRIMASK, R0
```

- PRIMASK register
  - In CMSIS

```
uint32_t __get_PRIMASK(void);
void __set_PRIMASK(uint32_t value);
void __enable_irq(void);       //CPSIE I
void __disable_irq(void);      //CPSID I
```

Special Registers (FAULTMASK)

- FAULTMASK register
  - Used to Disable all Exceptions except NMI.
  - The processor clears the FAULTMASK bit on exit from any exception handler except NMI handler

Bits	Name	Function
[31:1]	-	Reserved
[0]	FAULTMASK	0 = no effect 1 = disable all exceptions except NMI

```
CPSIE F ; clear FAULTMASK
CPSID F ; set FAULTMASK
```

```
; disable interrupt
MOV R0, #1
MSR FAULTMASK, R0
; enable interrupt
MOV R0, #0
MSR FAULTMASK, R0
```

- FAULTMASK register
  - In CMSIS

```
uint32_t __get_FAULTMASK(void);
void __set_FAULTMASK(uint32_t value);
void __enable_fault_irq(void);      //CPSIE F
void __disable_fault_irq(void);     //CPSID F
```

Special Registers (BASEPRI)

- BASEPRI (base priority) register
  - When BASEPRI is set to a Nonzero value, it prevents the activation of All Exceptions with the Same or Lower Priority Level as the BASEPRI value.

Bits	Name	Function
[31:8]	-	Reserved
[7:0]	BASEPRI	0x00 = no effect Non-zero = define the base priority

```
; disable interrupt
; priority 0x60 – 0xFF
MOV R0, #0x60
MSR BASEPRI, R0
```

- BASEPRI (base priority) register
  - In CMSIS

```
uint32_t __get_BASEPRI(void);
void __set_BASEPRI(uint32_t value);
```

Special Registers (CONTROL)

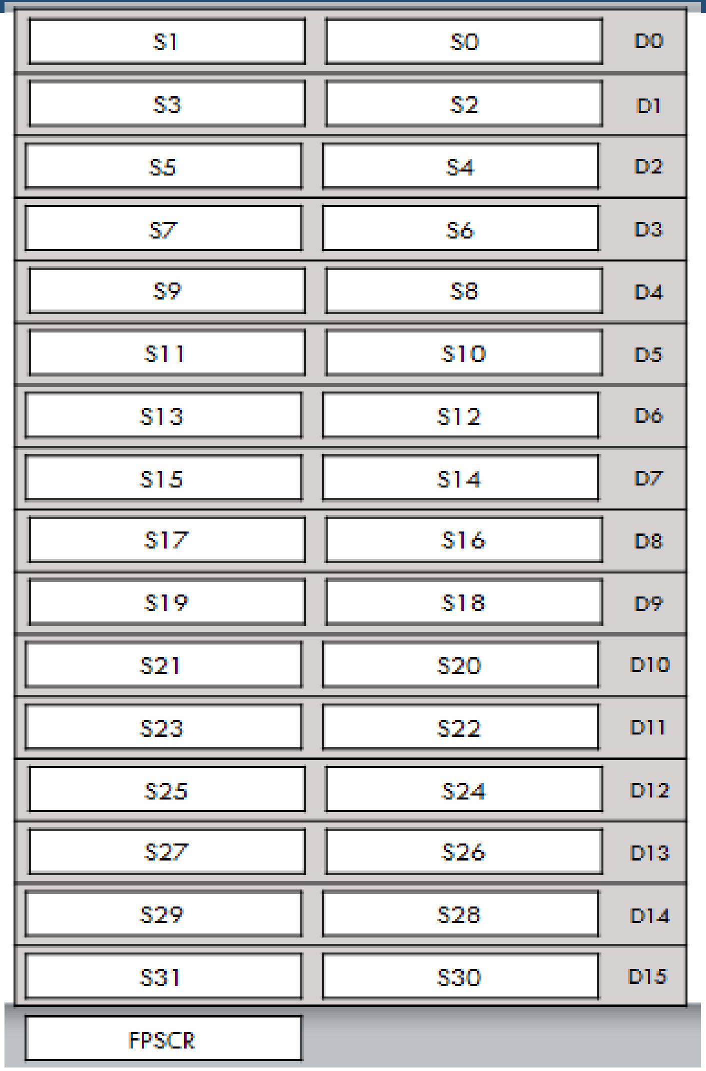
- CONTROL register
  - Controls the stack used and the privileged level in thread mode.
  - Writable only in privileged thread mode.
    - ✓Not allowed to write in the unprivileged or handler mode

Bits	Name	Description
[2]	FPCA	0=The instructions related to FPU are not used 1=The instructions related to FPU are used
[1]	SPSEL	0=default stack is used (MSP: main stack pointer) 1=process stack is used (PSP: process stack pointer)
[0]	nPRIV	0=privileged in thread mode 1=unprivileged in thread mode

- CONTROL register
  - In CMSIS

```
uint32_t __get_CONTROL(void);
void __set_CONTROL(uint32_t value);
```

Registers related to FPU



- S0 ~ S31: 32-bit registers
- D0 ~ D15 : 64-bit registers
  - D0 = S1:S0
- FPSCR: Floating point Status and Control Register