Table of Contents

Processor Mode

- The ARM has seven basic operating modes:
  - Each mode has access to own stack and a different subset of registers
  - Some operations can only be carried out in a privileged mode

Mode	Description		
Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes	
FIQ	Entered when a high priority (fast) interrupt is raised		
IRQ	Entered when a low priority (normal) interrupt is raised		
Abort	Used to handle memory access violations		
Undef	Used to handle undefined instructions		
System	Privileged mode using the same registers as User mode		
User	Mode under which most Applications / OS tasks run	Unprivileged mode	

Figure 1

Operation Mode

Exception	Mode	Priority	IV Address
Reset	Supervisor	1	0x00000000
Undefined Instruction	Undefined	6	0x00000004
Software Interrupt	Supervisor	6	0x00000008
Prefetch Abort	Abort	5	0x0000000C
Data Abort	Abort	5	0x00000010
Interrupt	IRQ	4	0x00000018
Fast Interrupt	FIQ	3	0x0000001C

CPSR

```
       31 30 29 28                           7 6  5  4  3  2  1  0

       N Z C V                               I F T M4 M3 M2 M1 M0
```

ALU Flags

IRQ disable
FIQ disable

Thumb/Arm
mode

Processor Mode**
10000 – User
10001 – FIQ
10010 – IRQ
10011 – Supervisor (SWI)
10111 – Abort D/I mem'y
11001 – Undefined instr.
11111 - System

Must be in a "privileged" mode
to change the CPSR

MRS rn,CPSR
MSR CPSR,rn

**2 modes in Cortex:
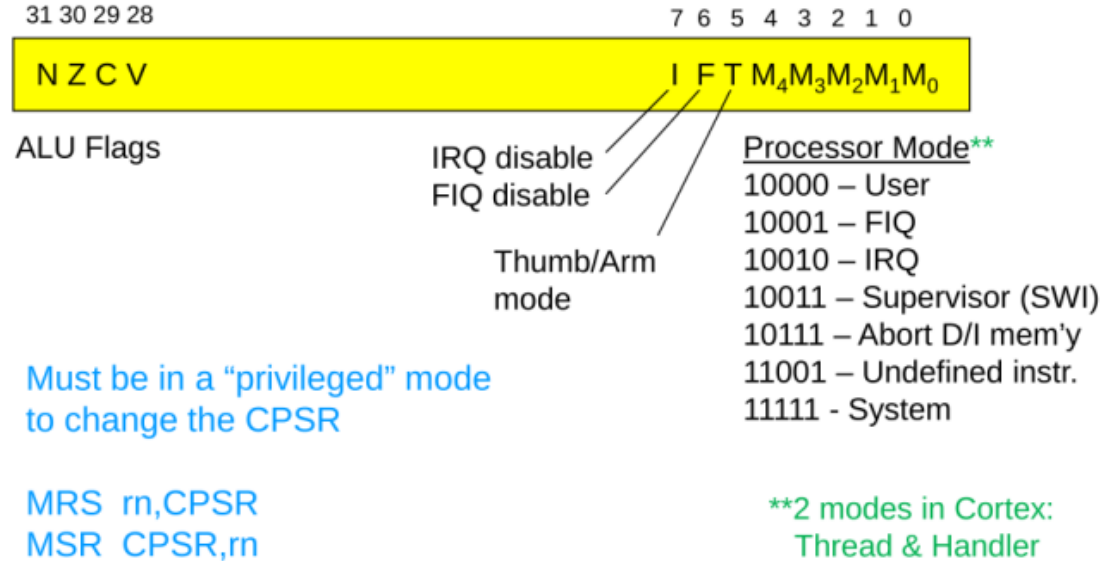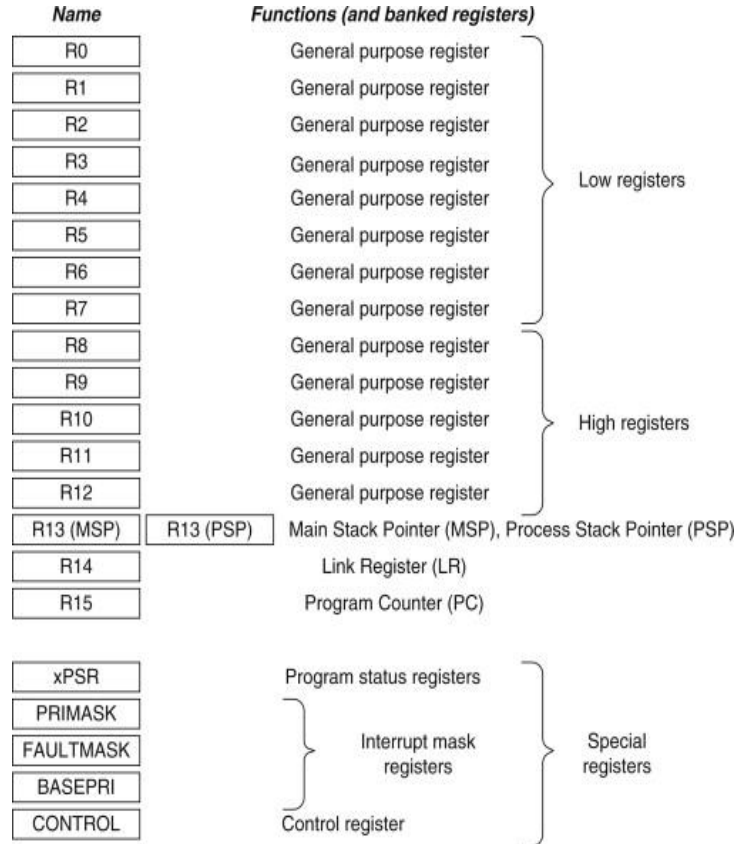   Thread & Handler

Figure 2

Register Sets

Figure 3

Special Registers

- Contains processor status
- Defines operation states and interrupt masking
- Can be accessed using special register access instruction MSR and MRS
- MRS reg, sp_reg ; read sp. register into a general purpose register
- MSR sp_reg, reg ; write to sp. register
- LSB is set zero (default) to be aligned with half-word or word addresses. It is set to 1 to indicate Thumb state.
- Program Status Register (PSR)
  - Application PSR (APSR)
  - Interrupt PSR (IPSR)
  - Execution PSR (EPSR)

PSR

Figure 4

The APSR contains the ==current state of the condition flags from previous instruction executions==.

Table 1: APSR Bit Assignment

Bit	Name	Function
[31]	N	Negative Flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	DSP overflow and saturation flag

Figure 5

IPSR: Interrupt PSR

The IPSR contains the exception type number of the current Interrupt Service Routine (ISR).

Table 2: IPSR Bit Assignment

Bit	Name	Function
[31 : 9]	–	Reserved
[8 : 0]	ISR Number	0 = Thread mode
		1 = Reserved
		2 = NMI
		3 = HardFault
		4 = MemManage
		5 = BusFault
		6 = UsageFault
		7-10 = Reserved
		11 = SVCall
		12 = Reserved for Debug
		13 = Reserved
		14 = PendSV
		15 = SysTick
		16 = IRQ0.

Figure 6

EPSR: Execution PSR

The EPSR contains the Thumb state bit, and the execution state bits for either the:

- If-Then (IT) instruction
- Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instructions

Bit	Name	Function
[31 : 27]	-	Reserved
[26 : 25], [15 : 10]	ICI	Interruptible-continuable instruction bit
[26 : 25], [15 : 10]	IT	Indicates the execution state bits of the IT instruction
[24]	T	Thumb state bit
[23 : 16]	-	Reserved
[9 : 0]	-	Reserved

Figure 7: EPSR Bit Assignment

EPSR

Attempts to read the EPSR directly through application software using the MSR instruction always return zero.

Interruptible-continuable instructions

- When an interrupt occurs during the execution of an LDM, STM, PUSH, or POP instruction, and when an FPU is implemented an VLDM, VSTM, VPUSH, or VPOP 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.

  When the EPSR holds ICI execution state, bits[26:25,11:10] are zero.

Exception Mask Registers

- The exception mask registers disable the handling of exceptions by the processor.
- Disable exceptions where they might impact on timing critical tasks.
- To access the exception mask registers use the MSR and MRS instructions, or the CPS instruction to change the value of PRIMASK or FAULTMASK
- All used for exceptions or interrupt masking based on priority level
- Each exception has a priority level: smaller number means higher priority and vice verse
- These registers only be accessed in privileged access mode
- By default all are zero

PRIMASK

- Interrupt mask register
- When set: blocks all interrupts and exceptions except Non-Maskable Interrupt(NMI) and HandFault exception
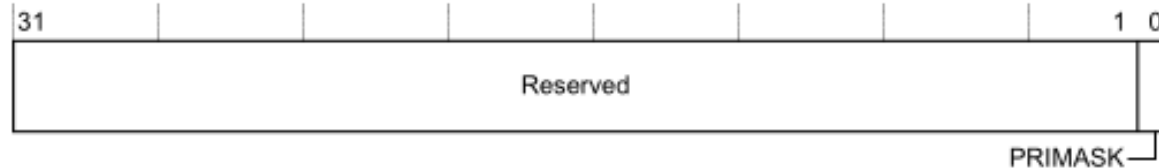- Most common usage: disable all interrupts for a time critical process

Table 2-7 PRIMASK register bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	PRIMASK	0 = no effect
		1 = prevents the activation of all exceptions with configurable priority.

Figure 8

FAULTMASK

- When set: blocks all interrupts and exceptions except Non-Maskable Interrupt(NMI)
- Most common usage: suppress the triggering of further faults during fault-handling
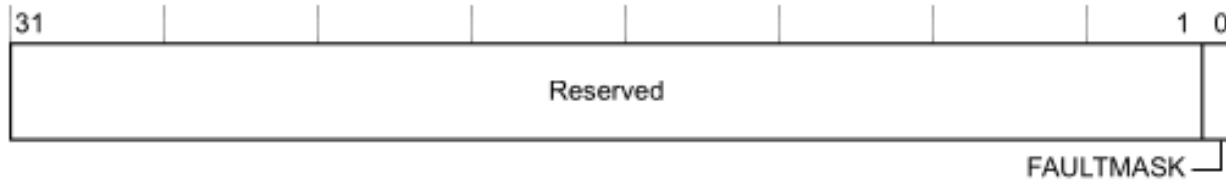

Table 2-8 FAULTMASK register bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	FAULTMASK	0 = no effect 1 = prevents the activation of all exceptions except for NMI.

Figure 9

The processor clears the FAULTMASK bit to 0 on exit from any exception handler except the NMI handler.

BASEPRI

- Masks interrupts and exceptions based on priority level
- The width depends on how many priority levels are implemented in design, defined by vendors
- Cortex M4 has 8 programmable exception priority level (3 bit width)/ 16 (4 bit)
- When set to non-zero value, blocks exception/interrupt that has same or lower priority

31	8	7	0
Reserved		BASEPRI	

Table 2-9 BASEPRI register bit assignments

Bits	Name	Function
[31:8]	-	Reserved
[7:0]	BASEPRI[a]	Priority mask bits: 0x00 = no effect Nonzero = defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.

Figure 10

a. This field is similar to the priority fields in the interrupt priority registers. Register priority value fields are eight bits wide, and non-implemented low-order bits read as zero and ignore writes.

Handling FAULTMASK, PRIMASK, BASEPRI

- MRS R0,PRIMASK ; Read PRIMASK register to R0
- Change Processor State (CPS) instruction: set or clear PRIMASK,FAULTMASK,BASEPRI
- Change Processor State, Enable Interrupts: CPSIE i; Enable interrupts and configurable fault handlers (clear PRIMASK)
- Change Processor State, Disable Interrupts: CPSID i; Disable interrupts and configurable fault handlers (set PRIMASK)
- CPSIE f; Enable interrupts and fault handlers (clear FAULTMASK)
- CPSID f; Disable interrupts and all fault handlers (set FAULTMASK)

This instruction does not change the condition flags.

Control Register (CR)

- Select stack pointer (SP_main/Sp_process)
- Define access level in Thread mode: Privileged/Unprivileged
- One bit of the control register whether the currect execution is using FPU or not
- Can be read in both Privileged/Unprivileged mode
- Can be modified in Privileged mode
- CR is 0: Thread mode uses SP_main+ Privileged access

Bits	Function
Bits 31:3	Reserved
Bit 2	**FPCA:** Indicates whether floating-point context currently active: 0: No floating-point context active 1: Floating-point context active. The Cortex-M4 uses this bit to determine whether to preserve floating-point state when processing an exception.
Bit 1	**SPSEL:** 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. The Cortex-M4 updates this bit automatically on exception return.
Bit 0	**nPRIV:** Thread mode privilege level. Defines the Thread mode privilege level. 0: Privileged 1: Unprivileged.

Figure: 11

Special Instructions for handling Exceptions

- PRIMASK: used to disable all exceptions except NMI and Hard faults
  - To disable all interrupts:
    MOVS R0, #1
    MSR PRIMASK, R0
  - To allow all interrupts:
    MOVS R0, #0
    MSR PRIMASK, R0
- FAULTMASK: similar to the previous one except that it changes the effective current priority level to -1, only the NMI exception handler can works
  - To disable all interrupts:
    MOVS R0, #1
    MSR FAULTMASK, R0
  - To allow all interrupts:
    MOVS R0, #0
    MSR FAULTMASK, R0
- BASEPRI: disable interrupts with priority lower than a certain level
  - To disable all interrupts with priority 0x60-0xFF:
    MOVS R0, #0x60
    MSR BASRPRI, R0
  - To cancel the masking:
    MOVS R0, #0x0
    MSR BASEPRI, R0