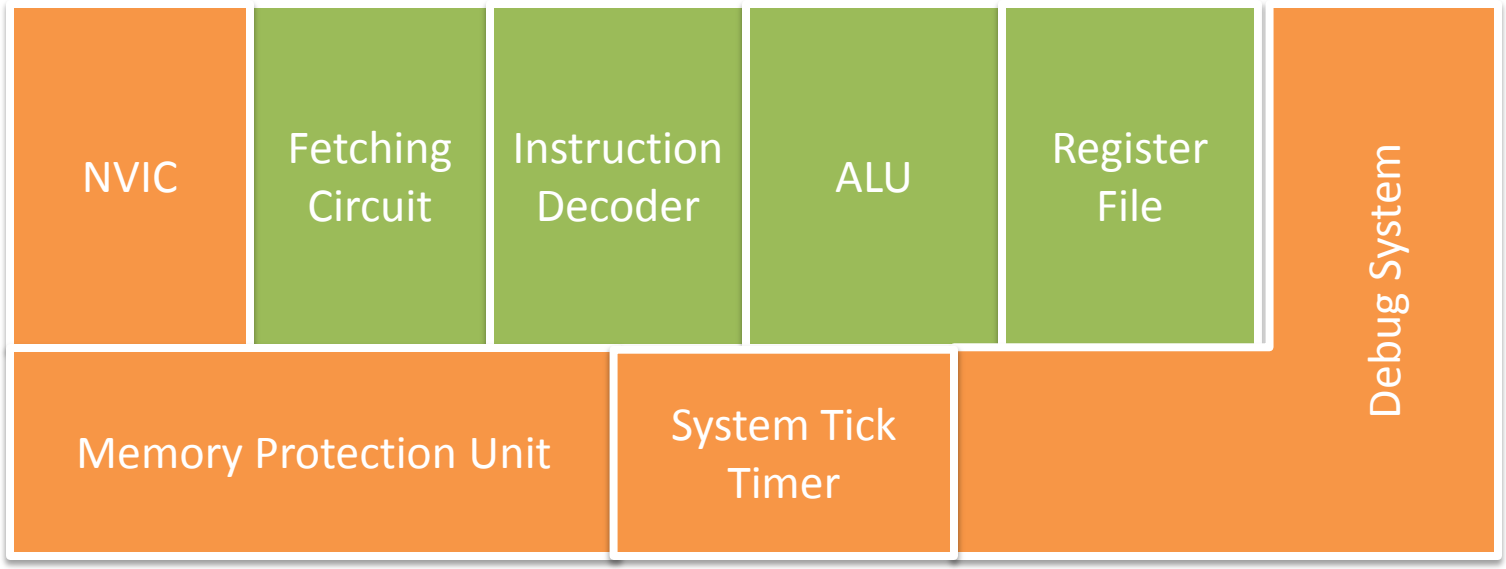
ARM Processor Model

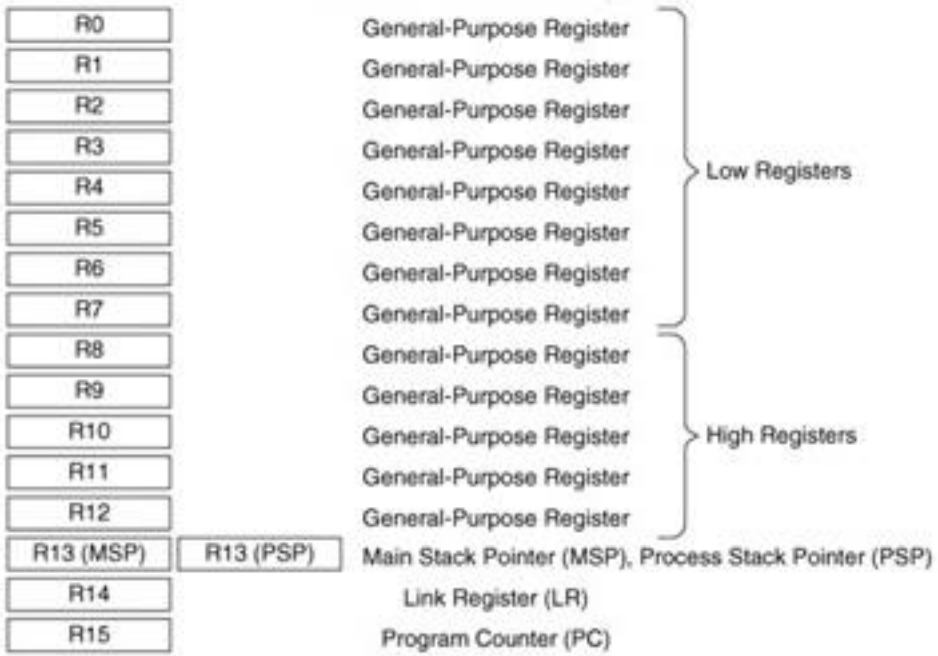
Lecture 2

Introduction to ARM Processor

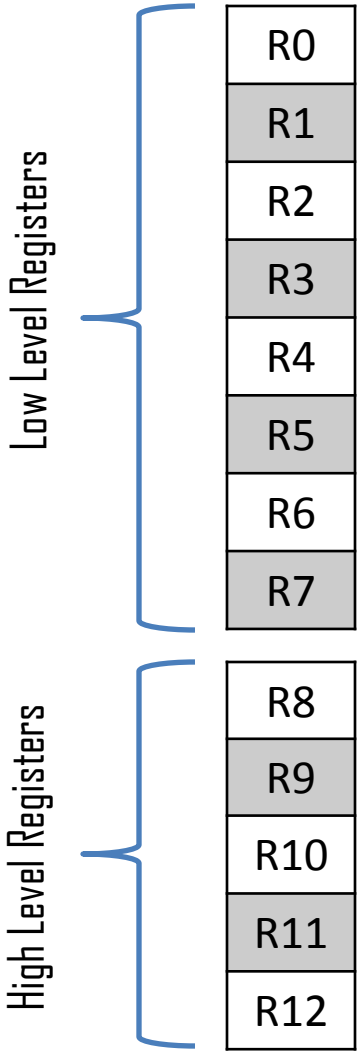
ARM Cortex M3/M4 Processor Model



Register File



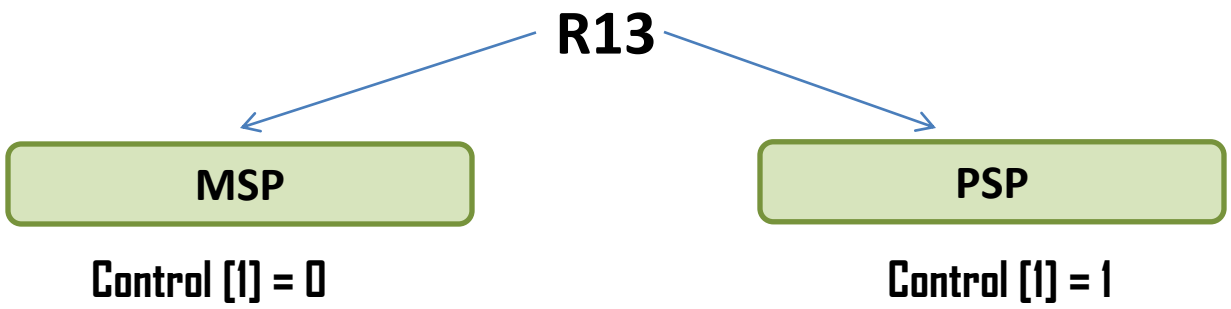
General Purpose Registers



- These registers are used for data operations
- Registers from R0 to R7 are called ***Low Level Registers*** because they are represented only by 3 bits (0 To 7). Therefor they are used mainly by the 16 bit instructions
- Registers from R8 to R12 are called ***High Level Registers*** because they are represented by 4 bits. All of the 32 bit instructions can access these registers while Few of the 16 bit instructions can access these registers.
- The initial value of these registers are not defined !

Stack Pointer R13

Physically there are 2 stack pointers, ***Main Stack Pointer*** **MSP** and ***Process Stack Pointer*** **PSP**. The R13 is pointing to one of them.



- **MSP** is the default stack pointer, the processor uses it by default after reset.
- Changing the stack the pointer is done by changing bit 1 in the **control** register.
- The processor uses the **MSP** always when executing the exception handler.

Link Register R14

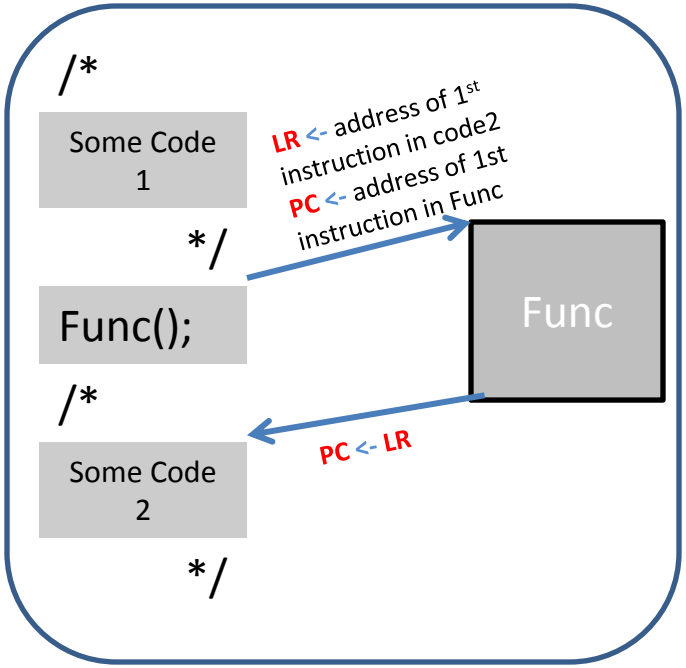
It stores the return information for function calls and exceptions. On reset, the processor sets the LR value to ***0xFFFFFFFF***.

Branch(B) vs Branch With Link(BL) :

The Cortex-M architecture supports two similar instructions: B and BL.

- ***B Instruction*** allows the user to branch to an instruction that does not immediately follow the current instruction without saving any information about the current address.
- ***BL Instruction*** saves the address of the instruction immediately following the BL command into the Link Register (R14). By storing the address of the next instruction into the link register, a function can "return" by moving the contents of the LR back into the PC using the BX (Branch and Exchange) command.

Example



Assembly Example

void *main* (void)

```
; main assembly program
;*********************
__main  PROC

; Initialize R0
MOV   R0, #0x01

; Call the function
BL    __func

;Loop forever
B     __main

ENDP
```

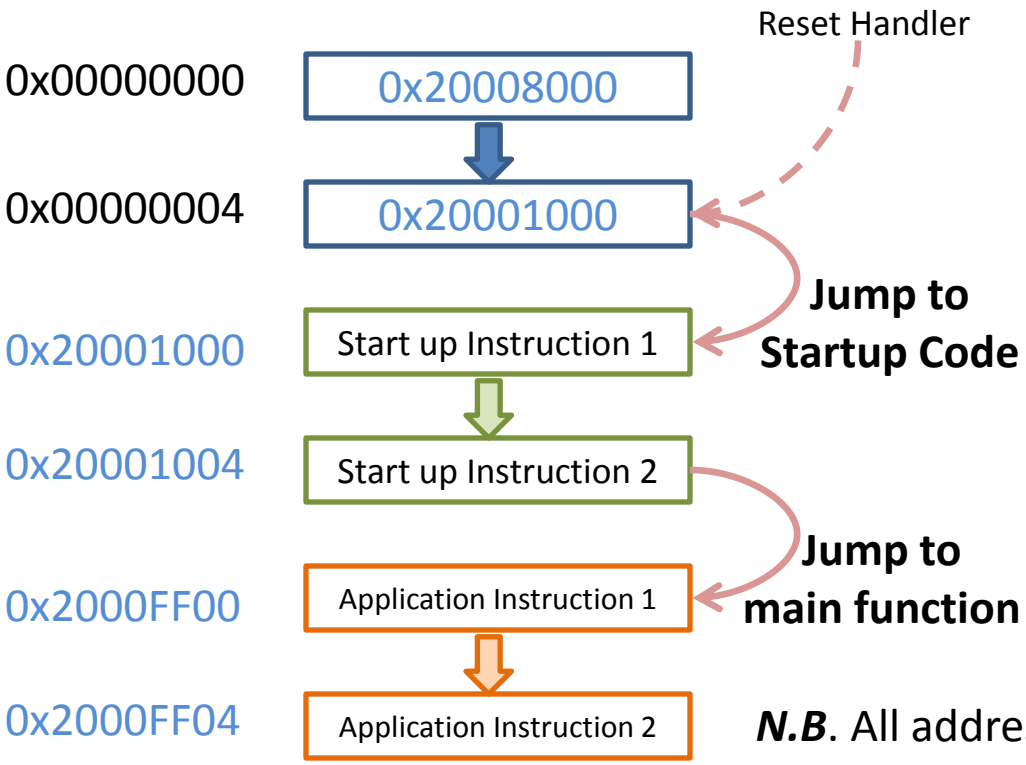
void *func* (void)

```
; func assembly program
;*********************
__func  PROC

;Check if R0 equals 1 or not
;and save 0 or 1 to R1
TST       R0, #0x01
MOVEQ     R1, #0
MOVNE     R1, #1

; Return from the function
BX    LR

ENDP
```

Program Counter R15

The ***Program Counter*** (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004.

Processor Reset Sequence:

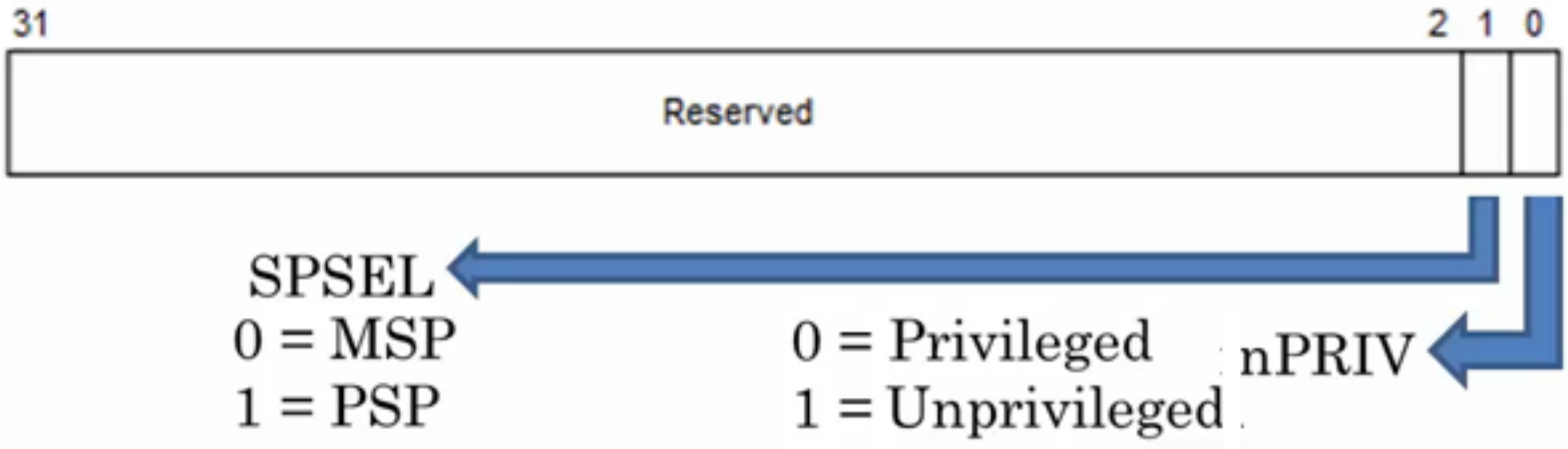


- After reset, the PC is loaded by ***0x00000000*** by hardware
- Processor saves the value of the address ***0x00000000*** to the ***MSP*** then jump to the reset handler
- The instruction in the reset handler is a jump instruction to the ***startup*** code

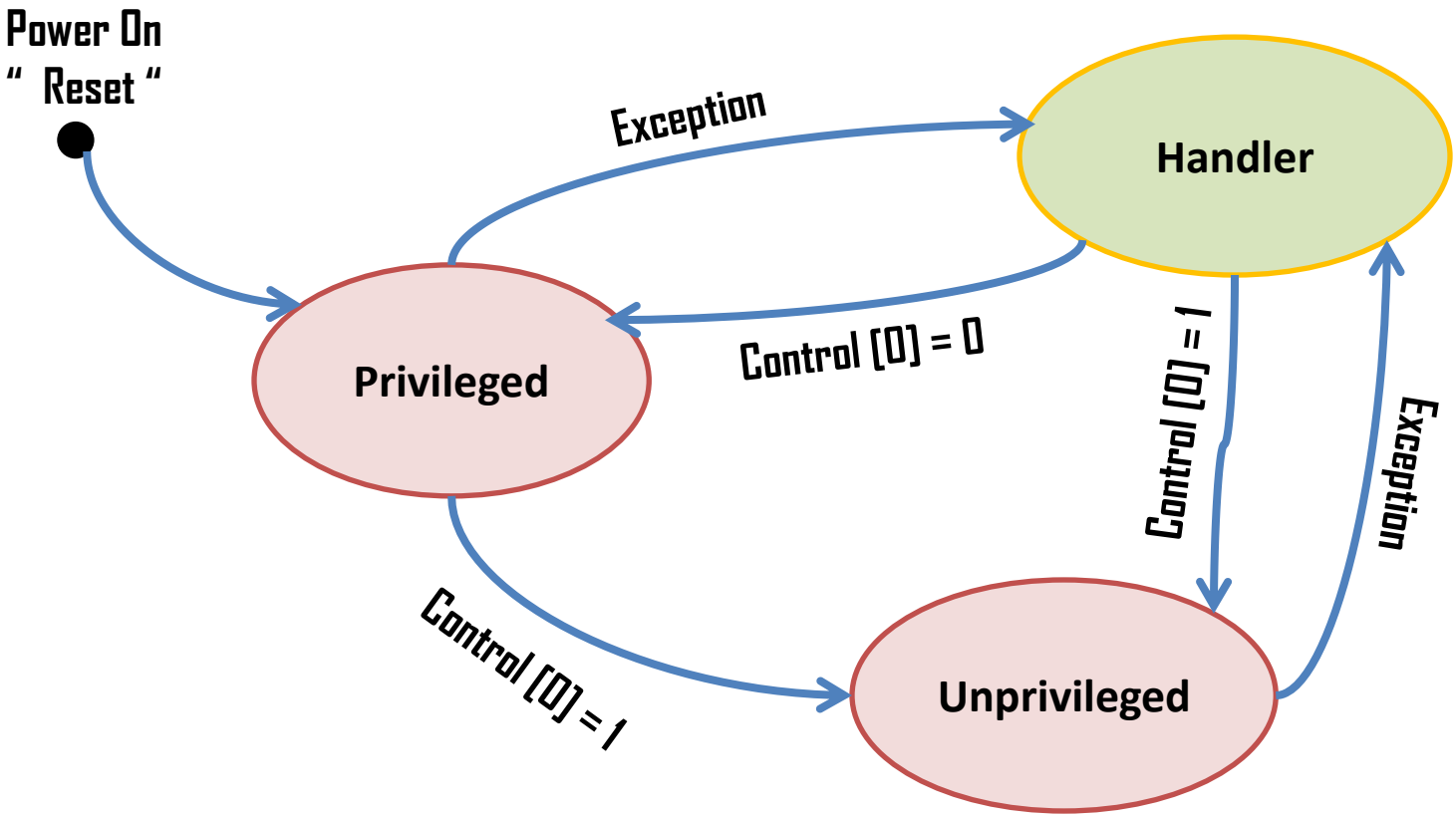
N.B. All addresses in ***Blue*** are for example

Control Register

The ***CONTROL*** register controls the stack used and the privilege level for software execution when the processor is in Thread mode.



Processor Operating Modes



Assembly Examples

Example 1

```
; Write 1 to Control register to enable all interrupts
MOV R0, #1
MSR CONTROL, r0
```

Example 2

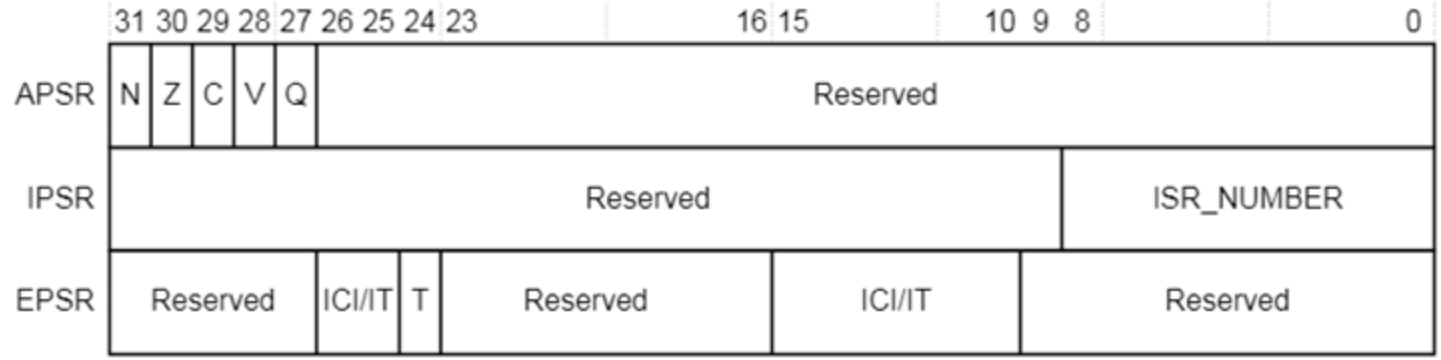
```
; Write 0 to Control register to enable all interrupts
MOV R0, #0
MSR CONTROL, r0
```

Program Status Word xPSR

The Processor Status Word (***xPSR***) combines 3 registers:

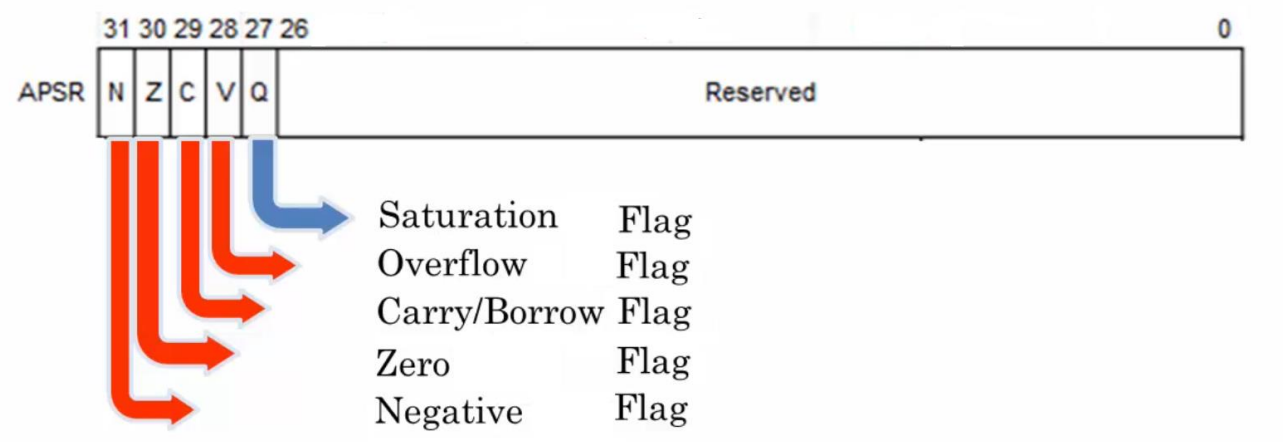
1. Application Program Status Register (***APSR***)
2. Interrupt Program Status Register (***IPSR***)
3. Execution Program Status Register (***EPSR***)

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are:



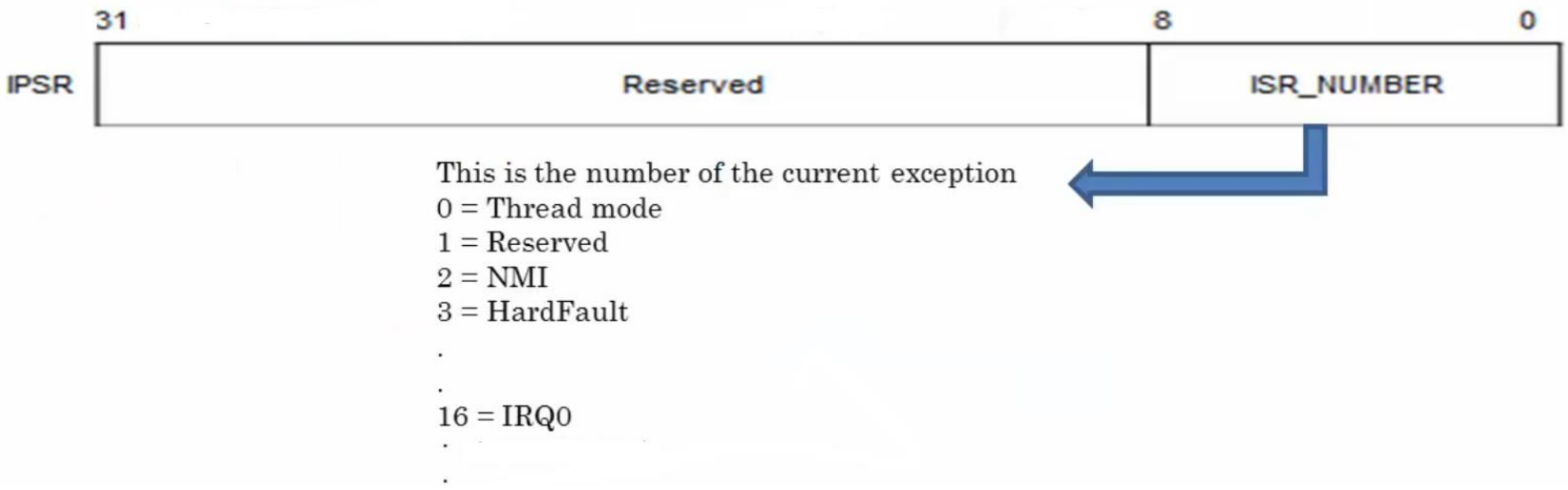
Application Program Status Register

The Application Program Status Register ***APSR*** contains the status flags of the integer operations (N, Z, C, V and Q) resulted from the previous instruction execution.



Interrupt Program Status Register

The Interrupt Program Status Register ***IPSR*** contains index of the current executing exception.



Execution Program Status Register

The Execution Program Status Register ***EPSR*** is giving information about the execution state of the processor. It contains the following bits:

1. The Thumb state bit which indicate the current used instruction set. In our case it should be always 1 because we have only one instruction set (Thumb 2).
2. Interruptible Continuable Instruction (ICI).

   When an interrupt occurs during the execution of an LDM or STM instruction, the processor:
   - Stops the load multiple or store multiple instruction operation temporarily
   - Stores the next register operand in the multiple operation to EPSRbits[15:12].

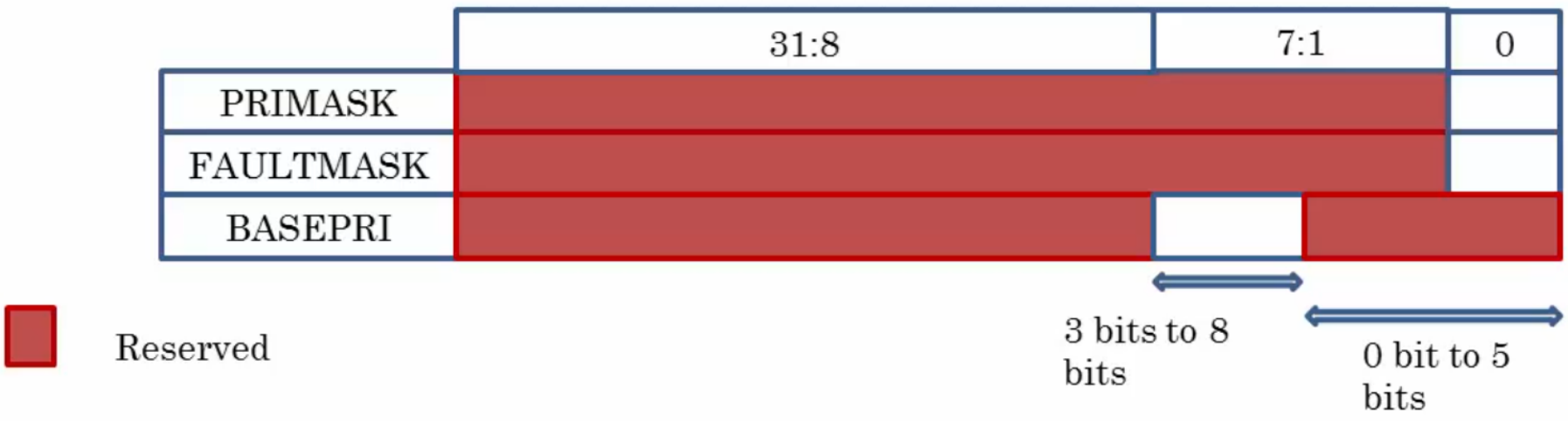
   After servicing the interrupt, the processor:
   - Returns to the register pointed to by bits[15:12]
   - Resumes execution of the multiple load or store instruction.

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

3. IF-Then Instruction

Exception Masking Registers

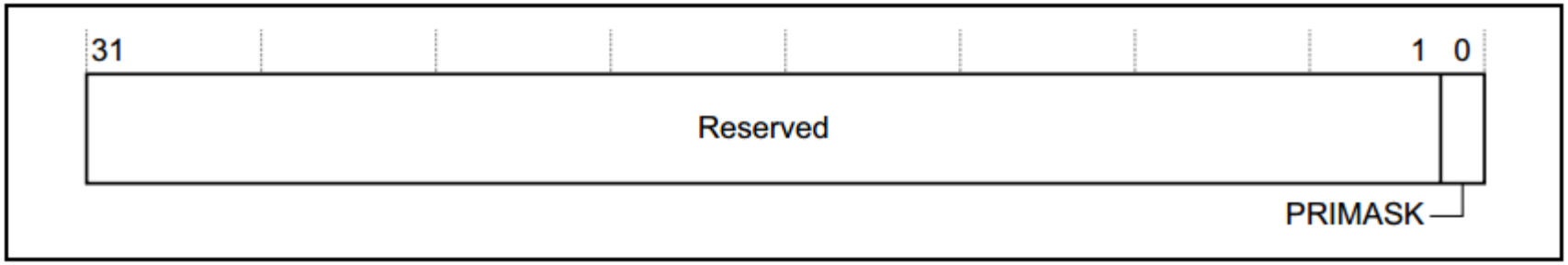
These three registers are used to mask interrupts.



To access the exception mask registers use the ***MSR*** and ***MRS*** instructions, or the ***CPS*** instruction to change the value of PRIMASK or FAULTMASK.

Priority Mask Register

The ***PRIMASK*** register prevents activation of all exceptions with configurable priority. It disables all interrupts have a priority more than or equals to 0. It is used to disable all exceptions except ***Reset***, ***NMI*** and ***hard fault***.



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

Assembly Examples

Example 1

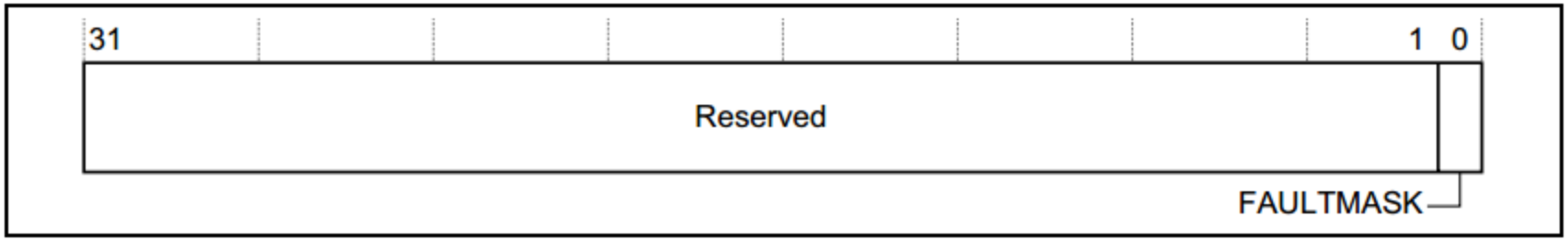
```
; Clear PRIMASK (Enable interrupts)
        CPSIE I
; Set PRIMASK (Disable interrupts)
        CPSID I
```

Example 2

```
; Write 0 to PRIMASK to enable all interrupts
MOV R0, #0
MSR PRIMASK, R0
; Write 1 to PRIMASK to disable all interrupts
MOV R0, #1
MSR PRIMASK, R0
```

Fault Mask Register

The ***FAULTMASK*** disables all interrupts have a priority more than or equals to -1. It is used to disable all exceptions except ***Reset*** and ***NMI***.



Bits	Function
Bits 31:1	Reserved
Bit 0	**FAULTMASK**: 0: No effect 1: Prevents the activation of all exceptions except for NMI.

Assembly Examples

Example 1

```
; Clear FAULTMASK (Enable interrupts)
CPSIE F
; Set FAULTMASK (Disable interrupts)
CPSID F
```

Example 2

```
; Write 0 to FAULTMASK to enable all interrupts
MOV R0, #0
MSR FAULTMASK , R0
; Write 1 to FAULTMASK to disable all interrupts
MOV R0, #1
MSR FAULTMASK , R0
```

Base Priority Mask Register

The ***BASEPRI*** register defines the minimum priority for exception processing. 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	Function
Bits 31:8	Reserved
Bits 7:4	**BASEPRI[7:4]** Priority mask bits(1) 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.
Bits 3:0	Reserved

Assembly Examples

Example 1

```
; Disable interrupts with priority 0x60-0xFF
MOV R0, #0x60
MSR BASEPRI, R0
```

Example 2

```
;Turn off BASEPRI masking
MOV R0, #0x0
MSR BASEPRI, R0
```

Register File Summery

Name	Type	Privilege	Reset Value
R0-R12	RW	Either	Unknown
MSP	RW	Privileged	See description
PSP	RW	Either	Unknown
LR	RW	Either	0xFFFFFFFF
PC	RW	Either	See description
PSR	RW	Privileged	0x01000000
ASPR	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

IMT SCHOOL

www.imtschool.com

ww.facebook.com/imaketechnologyschool/