**PIC(Programmable interrupt controller):**

Each hardware device controller capable of issuing interrupt requests usually has a single output line designated as the Interrupt ReQuest (IRQ) line. All existing IRQ lines are connected to the input pins of a hardware circuit called the Programmable Interrupt Controller, which performs the following actions:

1. Monitors the IRQ lines, checking for raised signals. If two or more IRQ lines are raised, selects the one having the lower pin number.
2. If a raised signal occurs on an IRQ line:
   1. Converts the raised signal received into a corresponding vector.
   2. Stores the vector in an Interrupt Controller I/O port, thus allowing the CPU to read it via the data bus.
   3. Sends a raised signal to the processor INTR pin—that is, issues an interrupt.
3. On the ARM core side PC(R15) with fetch the value from from data bus and decode it. After Decoding PC will Jump to the Vector number using the PC + n .

The IRQ lines are sequentially numbered starting from 0; therefore, the first IRQ line is usually denoted as IRQ 0. Intel's default vector associated with IRQ n is n+32. As mentioned before, the mapping between IRQs and vectors can be modified by issuing suitable I/O instructions to the Interrupt Controller ports.

Selective enabling/disabling of IRQs is not the same as global masking/unmasking of maskable interrupts. When the IF flag of the eflags register is clear, each maskable interrupt issued by the PIC is temporarily ignored by the CPU. The cli and sti assembly language instructions, respectively, clear and set that flag.

**The ARM Processor Operating Modes :**

The ARM processor has seven processor operating modes, as shown in Table 1. Each operating mode is used for a particular purpose; only one mode is in use at any one time:

|  |  |  |
| --- | --- | --- |
| **Mode** | **Privileged** | **Purpose** |
| User | No | Normal operating mode for most programs (tasks) |
| Fast Interrupt (FIQ) | Yes | Used to handle a high-priority (fast) interrupt |
| Interrupt (IRQ) | Yes | Used to handle a low-priority (normal) interrupt |
| Supervisor | Yes | Used when the processor is reset, and to handle the software interrupt instruction swi |
| Abort | Yes | Used to handle memory access violations |
| Undefined | Yes | Used to handle undefined or unimplemented instructions |
| System | Yes | Uses the same registers as User mode |

The operating modes of processor shown in table define the set of map registers that can

be that can be used at operating privilege level.

The ARM processor has a simple privilege model:

* All modes are privileged apart from User mode .
* Privilege is the ability to perform certain tasks that cannot be done from User mode.

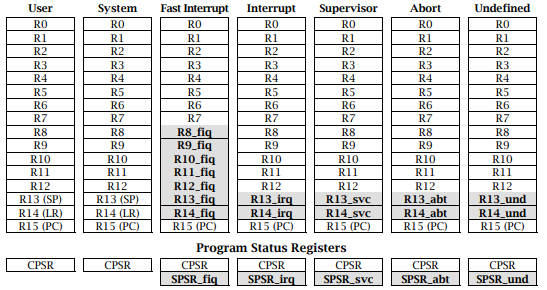
In a system with memory management, only privileged modes have access to certain areas of the address space, such as memory used by the operating system, or to I/O devices. User programs are then run from User mode, which does not have such privileges.

The ARM processor has a total of 37 registers:

* 31 general-purpose registers
* 6 status registers.

**ARM Register set :**

1. Register structure in ARM depends on the mode of operation. For example we have 16 (32- bit) registers named from R0 to R15 in ARM mode (usr).
   1. Registers **R0 to R12** are **general purpose registers**
   2. Register **R13** is **stack pointer (SP)**
   3. Register **R14** is **subroutine link register** and
   4. Register **R15** is **program counter (PC)**.
   5. Register **R16** is the **current program status register (CPSR)** .) This register is shared between all modes and it is used by the ARM core all the time and it plays a main role in the process of switching between modes.



So according to the above picture seven privileged mode contain some general proposed

Register and some special purpose.Using special register they configured in that particular mode .Now we going to consider each one by one :

1. **User Mode :**

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 **(GPR)**

R13 **(SP)**

R14 **(LR)**

R15 **(PC)**

User mode contain 16 general purpose register ,some of them shared with other privileged mode .

**2.Fast Interrupt Mode(FIQ) :**

R8\_fiq R9\_fiq R10\_fiq R11\_fiq R12\_fiq R13\_fiq R14\_fiq

In the FIQ we used the 7 Special register that design for FIQ mode .

**3. Interrupt Mode(IRQ) :**

R13\_irq R14\_irq

In the IRQ we used the 2 Special register that design for IRQ mode .

**4. Supervisor Mode(SVC) :**

R13\_svc R14\_svc

In the svc we used the 2 Special register that design for supervisor mode .

**5. abort Mode(abt) :**

R13\_abt R14\_abt

In the abt we used the 2 Special register that design for abort mode .

**6. Undefined Mode(und) :**

R13\_und R14\_und

In the und we used the 2 Special register that design for Undefined mode .

So Total Number of Register : User Mode(16)

+Fast Interrupt Mode(7)

+Interrupt Mode(2)

+Supervisor Mode(2)

+abort Mode(2)

+Undefined Mode(2) = 16+7+8 =31

Apart from the general purpose register ARM Processor Also contain the 6 Status register that is as **CPSR** ( **Current Program Status Register**).The CPSR register is used in only USER and SYSTEM MODE in the rest mode they are modified CPSR to SPCRs.

The **SPCRs** is called  **Saved Program Status Registers**

Total No of Reg in ARM CORE : General-purpose registers +CPSR

: 31 + 6

: 37

Each processor mode has its own **R13 and R14 registers**. This allows each mode to maintain its own stack pointer and return address. In addition, the Fast Interrupt (FIQ) mode has additional registers: **R8–R12.** This means that when the ARM processor switches into FIQ mode, the software does not need to save the normal R8–R12 registers, as FIQ mode has its own set that can be modified.

The **Current Program Status Register** (CPSR) is used to store condition code flags, interrupt disable bits, the current processor mode and other status and control information. This register is depicted in Figure 2:

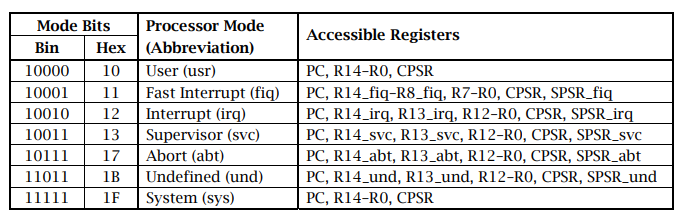


The Current Program Status Register is defined in the following way:

* Bits 24–31 can be modified in any mode, and are used to store the condition code flags.Only four condition code flags are available: N for Negative, Z for Zero, C for Carry and V for Overflow; the other bits are undefined. The condition code flags are set or cleared as a by-product of certain arithmetic instructions. For example,

“cmp r0,r1” ; Sets the Z (Zero) flag if R0 and R1 are equal .

* Bits 6 and 7 (F and I ) are the interrupt disable bits: setting one of these bits to 1 disables that interrupt; bit 6 disables the Fast Interrupt (FIQ), bit 7 disables the normal Interrupt (IRQ). These bits can only be modified in a privileged mode.
* Bit 5 (the T bit) determines whether the processor runs in ARM state or in Thumb state. Thumb state uses a different, more compact, instruction set when compared to ARM. You must never set this bit; doing so will make the processor enter an unpredictable state. This bit can only be modified in a privileged mode.
* Bits 0–4 set the processor mode; Table 2 shows the individual bit patterns needed to use a particular mode. These bits can only be modified in a privileged mode and Bits 8–27 are undefined and reserved for future or more advanced ARM processors.



Please note that the five Saved Program Status Registers (SPSRs) have the same format as the Current Program Status Register; these registers save the contents of CPSR when an exception occurs.

**System Initialisation :**

When an ARM-based system is switched on, a large amount of its state is undefined. In other words, the contents of volatile memory and, more importantly, most of the registers in the processor itself, will have random (undefined) values. However, only two registers are well-defined at reset: the **Program Counter(R15)** and the **Current Program Status Register**:

* The **Program Counter PC** (also called R15) is set to **0x00000000**. This allows the processor to execute instructions at that address; most systems are designed in such a way that the **Flash ROM** resides at address **0x00000000** at reset.
* The **Current Program Status Register(CPSR)** is set to 0x000000D3.This disables the Fast and normal Interrupts,and selects the normal ARM state (instead of Thumb state) and switches to Supervisor mode.

Once the ARM processor **resets PC and CPSR**, it usually begins executing code at the new address in register PC (0x00000000). This code (usually called the **boot code**) performs any further initialisation as required.

This includes setting up the various **stack pointers** (**R13 registers in each mode**), initialising the **exception handlers** (including those that handle interrupts) and setting up any peripheral devices in the system. After doing all this, the boot code usually enters User mode.

**Components of the Linux Boot Process:**

* **RBL**

Bootrom (or Boot ROM) is a small piece of mask ROM or write-protected flash

embedded inside the processor chip. It contains the very first code which is executed by the processor on power-on or reset.Depending on the configuration of some strap pins or internal fuses it may decide from **where to load the next part**(SPL in the RAM) of the code to be executed and how to verify it for correctness or validity.

ROM BL Reads(SPL) from initialized persistent storage (selected by boot mode) SPL in to internal Ram(SRAM)

* **SPL**

Secondary Program Loader, called many different names depending on processor (UBL,Xloader). SPL provide specific configuration of target board that has the capability to setup the processor to be able to read in the next stage which is U-Boot.

SPL does additional setup (DD3RRAM for u-boot) in the primary RAM and Map and reads from persistent storage for u-boot into DDR.

* **u-boot**

Enables most of the specific processor functionality for the target board and u-boot do the following task :

* Read the uImage/zImage from the external Memory and Map it on the DDR3RAM
* Load the dtb file from from the external memory and give the control to the Kernel to read the Board specific hardware .
* **Kernal image**

Final stage of the boot process. Kernel initialization, MMU enable, Device Initialization, User Init process and finally user level applications.

<https://www.timesys.com/security/trusted-software-development-op-tee/>

<https://arxiv.org/pdf/1506.07367.pdf>

<https://www.semanticscholar.org/paper/SeCReT%3A-Secure-Channel-between-Rich-Execution-and-Jang-Kong/3f4eef59703ff179e2faf08d04156d00acb1b352/figure/0>

# **Resets**

The processor has the following reset inputs:

**nRESET**

This signal is the main processor reset that initializes the majority of the processor logic.

**PRESETDBGn**

This signal resets processor debug logic and CoreSight ETM-R4.

**nSYSPORESET**

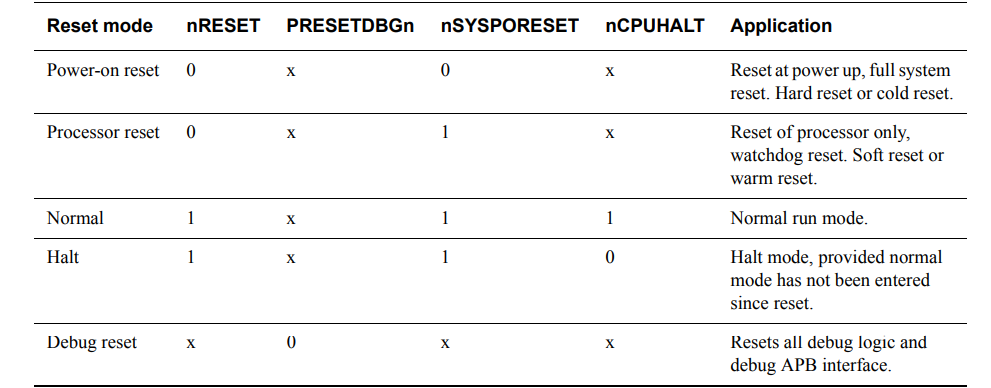
This signal is the reset that initializes the entire processor, including CP14 debug logic and the APB debug logic. See CP14 registers reset on page 11-23 for information.

**nCPUHALT**

This signal stops the processor from fetching instructions after reset.

**Reset modes:**

The reset signals in the processor enable you to reset different parts of the design independently.Table 3-1 shows the reset signals, and the combinations and possible applications that you can use them in.



**Power-on reset :**

You must apply power-on or cold reset to the processor when power is first applied to the system.During the power-on reset, the Rising/falling, edge of the reset signals like nRESET and nSYSPORESET, didn’t no synchronous with CLKIN signal, Because the nRESET and nSYSPORESET signals are synchronized within the processor.

ARM recommends that you assert the reset signals for at least four CLKIN cycles to ensure correct reset behavior.

**Processor reset**

A processor or warm reset initializes the majority of the processor, excluding the EmbeddedICE-RT logic. Processor reset is typically used for resetting a system that has been operating for some time, for example, watchdog reset. Because the nRESET signal is synchronized within the processor, you do not have to synchronize this signal.

