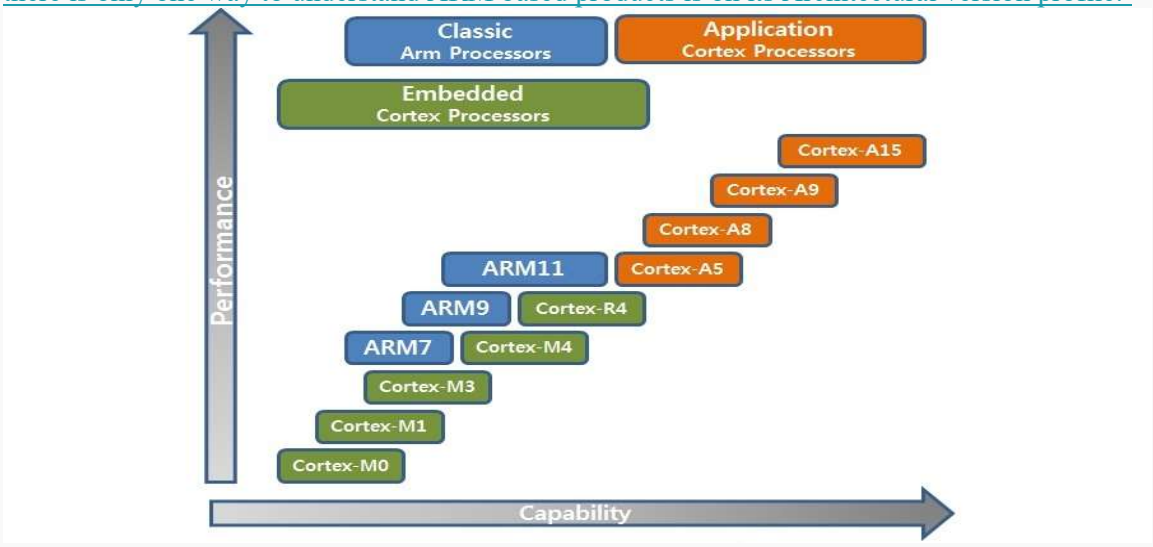
ARM Processors Architecture Overview

This post is specifically designed to ARM Processors Architecture Overview which is not very brief but give you complete overview of strong processor architecture

Development of the ARM Architecture

ARM has introduced many processors. Each set or groups of processors are having different core and different Features. Development of the ARM Architecture is started with 26 bit processors and nowadays it reach upto 64 bit. we can not classify general development of ARM products on any particular fact, there is only one way to understand ARM based products is on its Architectural version profile.



- **ARM Classic** series

The classical ARM series refers to processors starting from ARM7 to ARM11. This is the series which gives market boost to ARM because of its core features like Data Tightly Coupled memory, cache, MMU, MPU, etc. Typical examples of this series are ARM7TDMI, ARM926EJ-S, ARM11 MPCore, etc.

- **Cortex-A series:** A-profile, the "Application" profile

In this Cortex architecture, we port different embedded OS and design embedded system by OS system programming. The core feature of this profile is Highest performance at low power, TrustZone and Jazelle-RCT for a safe and extensible system. Practical development platform of this types of profile is FriendlyARM, Raspberry Pi, etc.

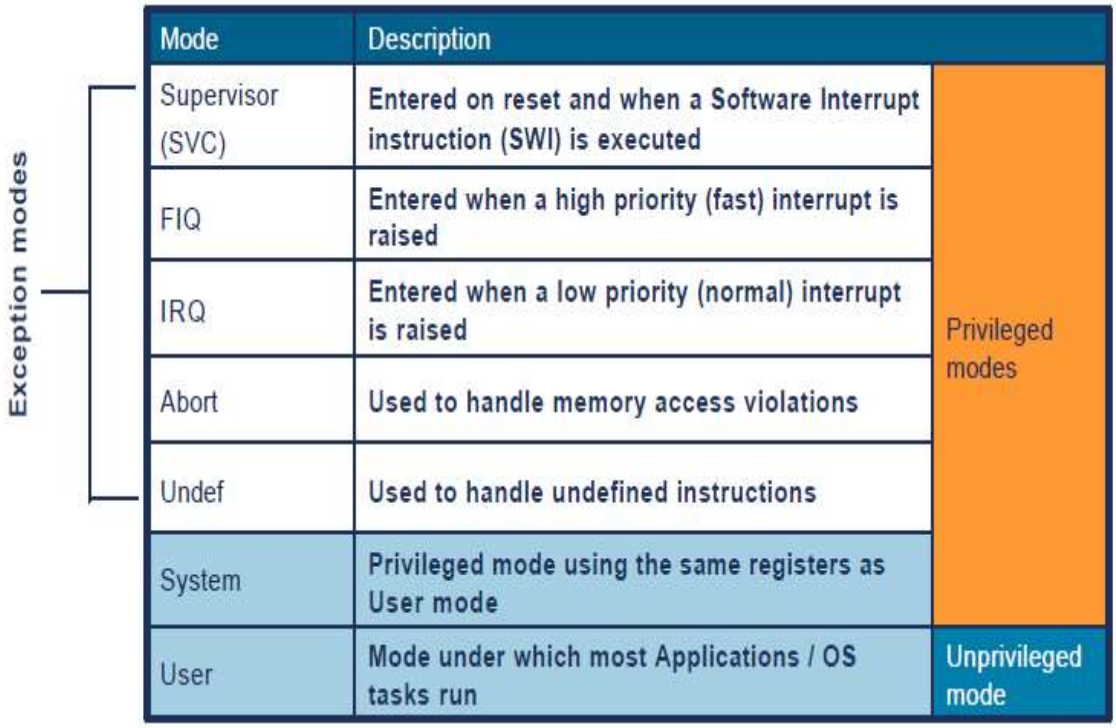
- **Cortex-R series:** R-profile, the "Real-time" profile:

This is Cortex architecture which mostly used for real time purpose where application abort is critical situation contain core features like Protected memory (MPU), Low latency and predictability 'real-time' needs.

- **Cortex-M series:** M-profile, the "Microcontroller" profile:

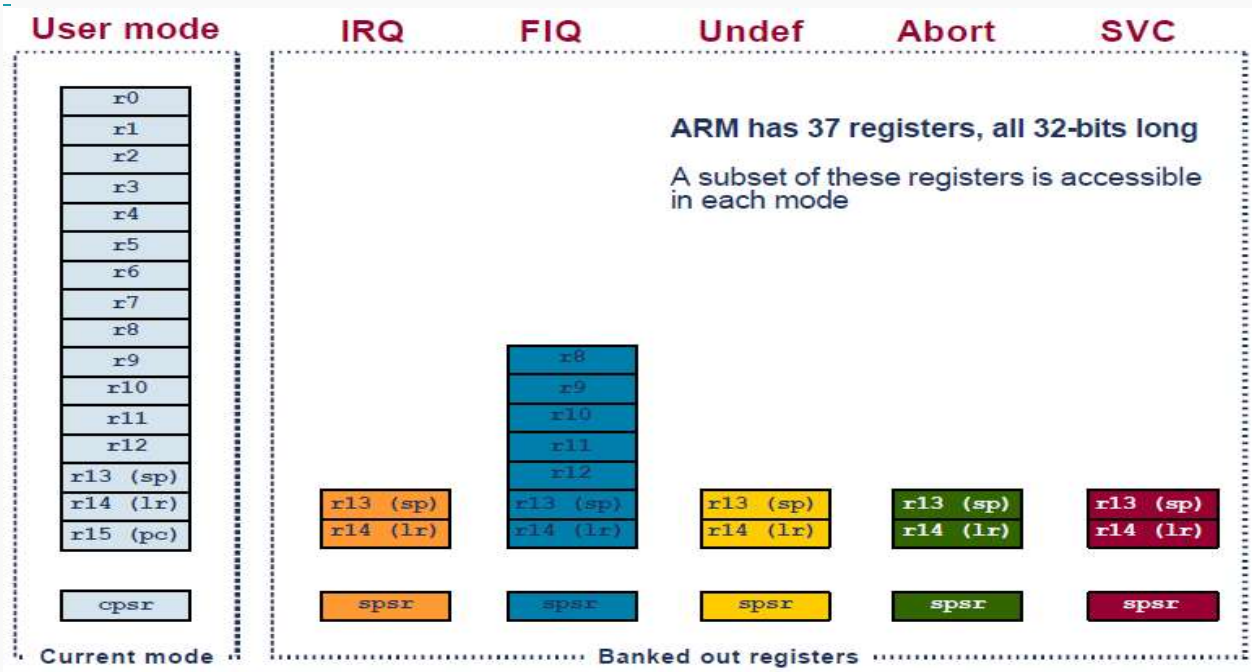
This profile is specially dedicated for microcontroller purpose only. The core feature of this profile is like Lowest gate count entry point, Deterministic and predictable behavior a key priority, Deeply embedded use. Typical example of this kind of profile architecture is Hercules TMS470M, STM32F429, etc.

Processor Modes



All these mode indicates its special work and execute under certain condition with its own stack and a different subset of registers. Modes other than User mode are collectively known as privileged modes. your program or application generally runs under User mode. Privileged modes are used to service interrupts or exceptions, or to access protected resources.

The ARM Register Set



CPSC – Current program status register

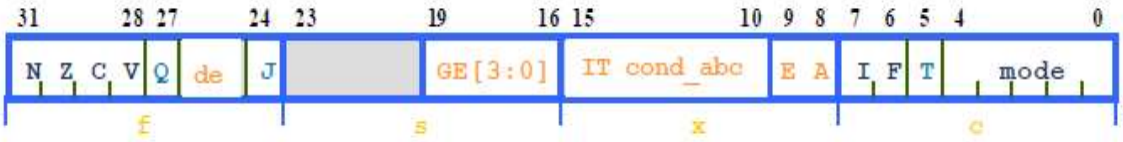
SPSR – Saved program status register

LR – Link register

SP – Stack Pointer

PC – Program Counter

Current/State Program Status Register (CPSR / SPSR)



Condition code flags (Bit 28 to 31)
- V = ALU operation oVerflowed
- C = ALU operation Carried out
- Z = Zero result from ALU
- N = Negative result from ALU

Sticky Overflow flag – Q flag (Bit 27)
- Architecture 5TE and later only
- Indicates if saturation has occurred

J bit (Bit 24)
- Architecture 5TEJ and later only
- J = 1: Processor in Jazelle state

T Bit (Bit 5)
- T = 0: Processor in ARM state
- T = 1: Processor in Thumb state
- Introduced in Architecture 4T

Mode bits (Bit 0 to 4)
- Specify the processor mode

New bits in V6 (Bit 10 to 19)
- GE[3:0] used by some SIMD instructions
- E bit controls load/store endianness
- A bit disables imprecise data aborts
- IT [abcde] IF THEN conditional

Interrupt Disable bits (Bit 6 Bit 7) • I = 1: Disables IRQ**F = 1: Disables FIQ**	• execution of Thumb-2 instruction groups

ARM Instruction Set

- All instructions are 32 bits long / many execute in a single cycle
- Instructions are conditionally executed
- A load / store architecture
- Example **data processing instructions**

SUB r0,r1,#5	r0 = r1 – 5
ADD r2,r3,r3,LSL #2	r2 = r3 + (r3 * 4)
ADDEQ r5,r5,r6	IF EQ condition true r5 = r5 + r6

- Example **branching instruction**

B <Label>	Branch forwards or backwards relative to current PC (+/-32MB range)

- Example **memory access instructions**

LDR r0,[r1]	Load word at address r1 into r0
STRNEB r2,[r3,r4]	IF NE condition true, store bottom byte of r2 to address r3+r4
STMFD sp!,{r4-r8,lr}	Store registers r4 to r8 and lr on stack. Then update stack pointer

Thumb Instruction Set

- Thumb is a 16-bit instruction set
- Optimized for code density from C code (~65% of ARM code size)
- Improved performance from narrow memory
- Subset of the functionality of the ARM instruction set
- Thumb is not a “regular” instruction set because that targeted at compiler generation, not hand coding.
- An application code compiled in Thumb is 30% smaller on average than the same code compiled in ARM and normally 30% faster when using narrow 16-bit memory systems.

Thumb-2 Instruction Set

- Thumb-2 is a major extension to the Thumb ISA
- Adds 32-bit instructions to implement almost all of the ARM ISA functionality

- Retains the complete 16-bit Thumb instruction set
- Compiler automatically selects mix of 16 and 32 bit instructions

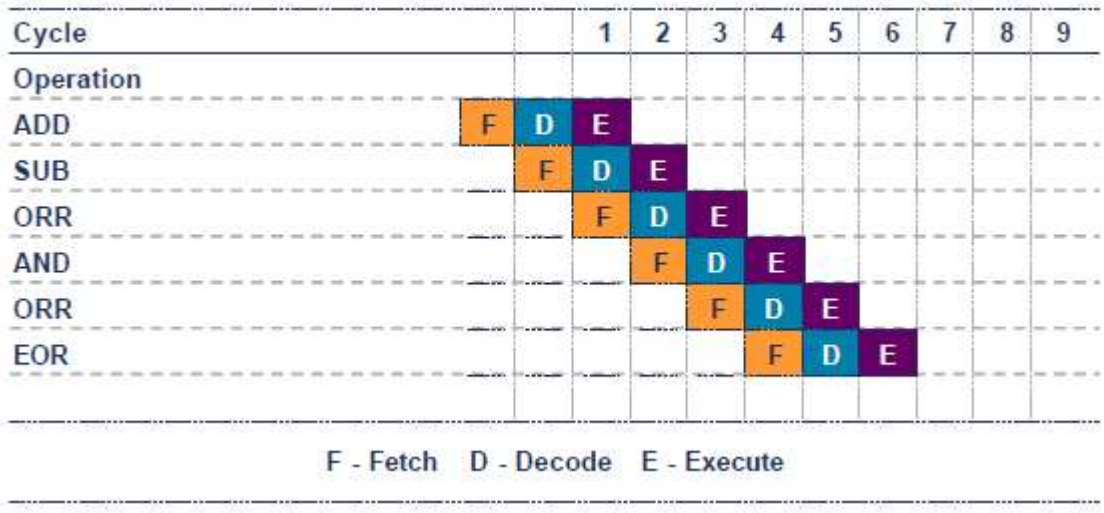
The Instruction Pipeline

The ARM7TDMI uses a 3-stage pipeline in order to increase the speed of the flow of instructions to the processor which allows several operations to be performed simultaneously, rather than sequentially. The basic steps of any operation is

Fetch :- Instruction fetched from memory

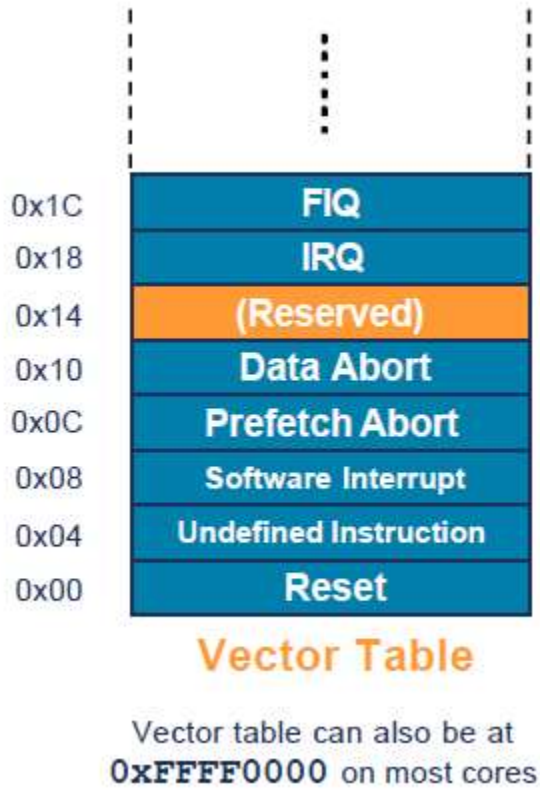
Decode :- Decoding of registers used in instruction

Execute :- Register(s) read from Register Bank, Shift and ALU operation, Write register(s) back to Register Bank.



In ARM architecture, pipe-lining is possible because of its RISC feature in which all instruction size is same so we can take advantage of pipe-lining. As you can see on above figure when ADD instruction is executing we fetched and start decoding next instruction SUB and at the same time we fetched the ORR instruction. So in the time of single instruction execution, we performing 3 instruction. Therefore, this is called three stage pipe-lining.

Exception Handling



When an exception occurs, the core:

- Copies CPSR into SPSR_<mode>
- Sets appropriate CPSR bits
- Change to ARM state
- Change to exception mode
- Disable interrupts (if appropriate)
- Stores the return address in LR_<mode>
- Sets PC to vector address

To return, exception handler needs to:

- Restore CPSR from SPSR_<mode>
- Restore PC from LR_<mode>

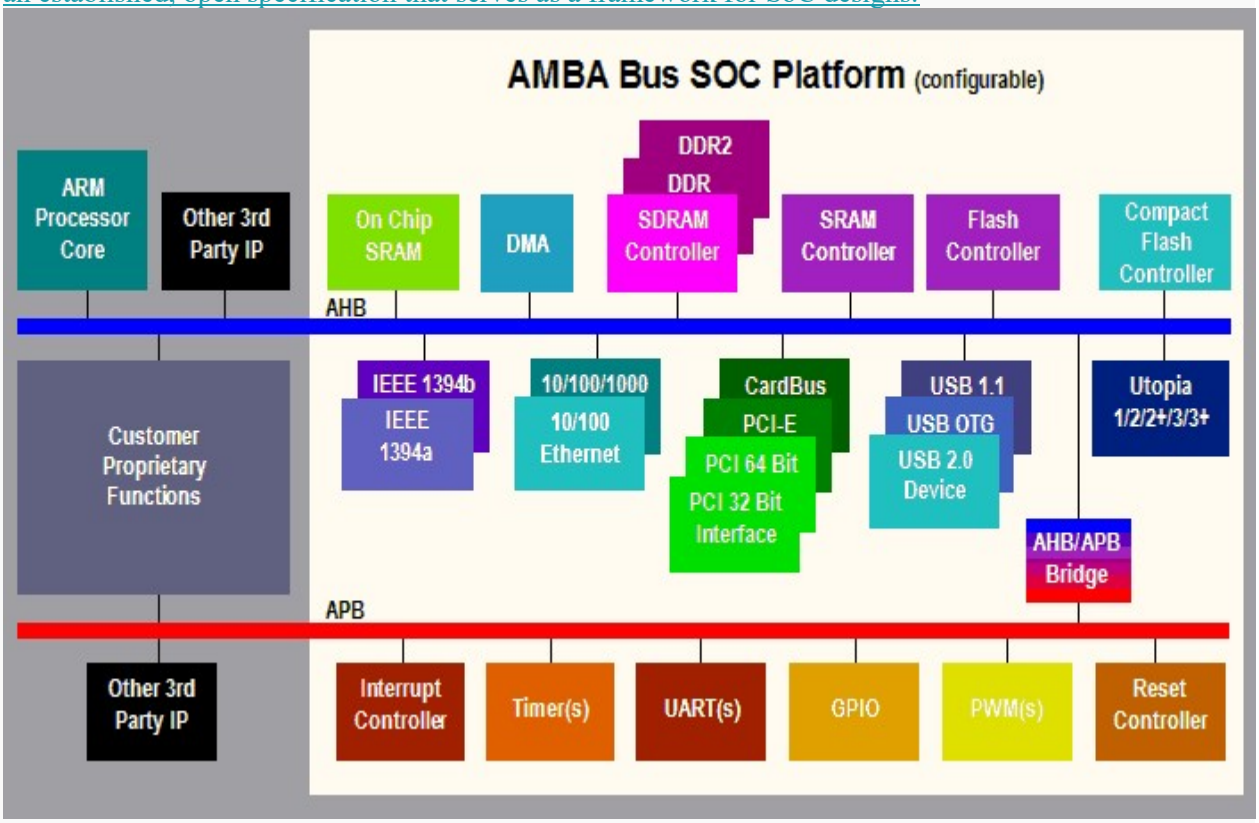
Terms associated with ARM

EmbeddedICE

In order to provide a powerful debugging environment for ARM-based applications the EmbeddedICE logic was developed and integrated into the ARM core architecture. It is a set of registers providing the ability to set hardware breakpoints or watch-points on code or data. The EmbeddedICE logic monitors the ARM core signals every cycle to check if a break-point or watch-point has been hit. Communication with the EmbeddedICE logic from the external world is provided via the test access port or TAP, controller and a standard IEEE 1149.1 JTAG connection. The advantage of on-chip debug solutions is the ability to rapidly debug software, especially when the software resides in ROM or Flash.

AMBA Bus

The "**A**dvanced **M**icro-controller **B**us **A**rchitecture" on-chip bus is freely available from ARM and offers an established, open specification that serves as a framework for SoC designs.



Reference – www.arm.com

The design of the AMBA bus specification is focused on low power consumption and high performance. A typical AMBA-based SoC consists of an advanced high-performance system bus(AHB), and an advanced low power peripheral bus(APB). As seen in above pic

- On the performance critical side of the bus is the ARM core, Memory Controller, Test Interface Controller (TIC), the LCD Controller, on-chip IP, custom logic, and specialized functions.
- On the low power side of the bus is the Smart Card interface, audio codec, UART, PWM, Timers, GPIO, etc.

- This is an excellent example of how the AHB and APB buses work in conjunction to provide a complete system solution.

These bus protocols are independent of the ARM processor and generalized for SoC application. The AMBA test methodology provides a mechanism to give an external tester access to the on-chip AMBA bus. This enables the tester to take control of the bus and check each component separately.

MMU

The **M**emory **M**anagement **U**nit works with the cache memory system to control accesses to and from external memory. The MMU also controls the translation of Virtual Addresses to physical addresses and access permission checks for the instruction and data ports of Processors. An MMU mitigates the problem of fragmentation of memory too.

MPU

A separate **M**emory **P**rotection **U**nit feature has been introduced in ARM by Cortex-M3 cores which provide a way to control memory access rights to applications. This prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception, generally causing termination of the process. In ARM core, separate registers are provided by which you can configure certain portion of memory and its access rights.

Sources

1. www.homepages.thm.de
2. en.wikipedia.org
3. www.arm.com
4. www.infocenter.arm.com

Suggested Reading

1. Introduction of ARM(LPC21XX)
2. Introduction to Arm Cortex-M Microcontrollers (STM32F4 Discovery Board)
3. Understanding Development Environment of ARM Cortex-Mx
4. Difference between RISC and CISC architecture