

2. Programmers' model

NXSOL-OJT-2024

Exported on 02/27/2024

Table of Contents

1	Programmers' Model	4
1.1	Data Types.....	4
1.2	Processor Modes.....	4
1.2.1	Exception level	5
1.3	Registers.....	6
1.3.1	General-purpose registers	7
1.3.2	Special Uses of these registers	7
1.4	Program Status Registers	7
1.4.1	Types of PSR bits.....	8
1.4.2	The condition code flags.....	8
1.4.3	The Q flag	8
1.4.4	The GE[3:0] bits	9
1.4.5	The E bit.....	9
1.4.6	The interrupt disable bits.....	9
1.4.7	The mode bits	9
1.4.8	The T and J bits	10
1.4.9	Other bits	10
1.5	Exceptions.....	10
1.5.1	Exception process	10
1.5.2	Types of exceptions.....	10
1.5.2.1	Reset.....	11
1.5.2.2	Undefined Instruction exception.....	11
1.5.2.3	Software Interrupt exception.....	11
1.5.2.4	Prefetch Abort (Instruction fetch memory abort)	11
1.5.2.5	Data Abort (Data access memory abort)	11
1.5.2.6	Interrupt request (IRQ) exception	11
1.5.2.7	Fast interrupt request (FIQ) exception.....	11

1.5.3	Exception priorities	12
1.6	Endian	12
1.7	Unaligned access support	13
1.7.1	Changes with ARMv6:.....	13
1.8	Synchronization primitives	13
1.8.1	LDREX (Load-Exclusive)	13
1.8.2	STREX (Store-Exclusive).....	14
1.8.3	Typical Usage in a Loop:.....	14
1.9	Jazelle extension	14
1.10	Saturated integer arithmetic	14

1 Programmers' Model

1.1 Data Types

- **Byte (8 bits):** Smallest data type.
- **Halfword (16 bits):** Introduced in ARM version 4.
- **Word (32 bits):** Main data type for operations.
- **Support for Unaligned Data:** ARMv6 introduced unaligned data support for words and halfwords.
- **Signed and Unsigned Types:** Represent integers using normal binary format or two's complement format.

1.2 Processor Modes

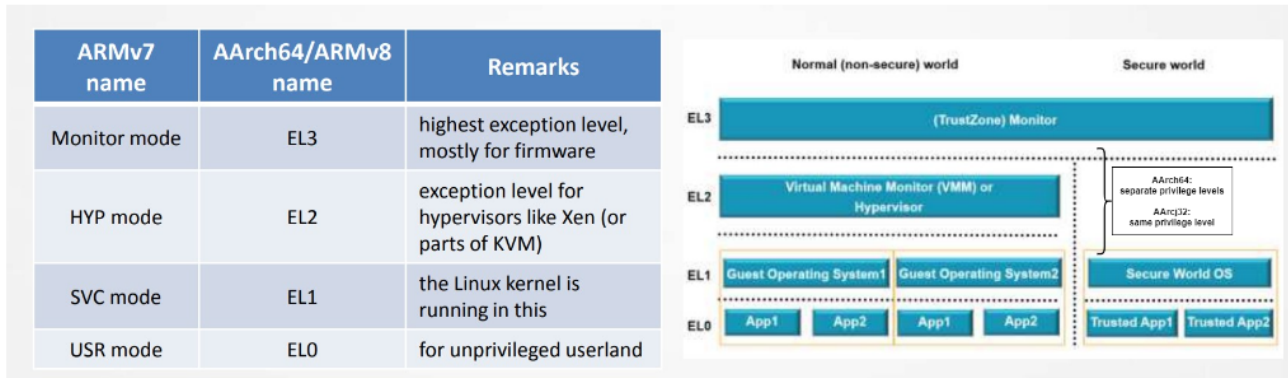
Table A2-1 ARM processor modes

Processor mode		Mode number	Description
User	usr	0b10000	Normal program execution mode
FIQ	fiq	0b10001	Supports a high-speed data transfer or channel process
IRQ	irq	0b10010	Used for general-purpose interrupt handling
Supervisor	svc	0b10011	A protected mode for the operating system
Abort	abt	0b10111	Implements virtual memory and/or memory protection
Undefined	und	0b11011	Supports software emulation of hardware coprocessors
System	sys	0b11111	Runs privileged operating system tasks (ARMv4 and above)

- **Mode changes:** Can be made under software control, or can be caused by external interrupts or exceptions.
- **User mode:** Most application programs run in User mode, which cannot access protected system resources or to change mode, other than triggering an exception.
- **Privileged modes:** The modes other than User mode are known as privileged modes. They have full access to system resources and can change mode freely.
Five of them are known as exception modes: FIQ, IRQ, Supervisor, Abort, Undefined. These are entered when specific exceptions occur. Each of them has some additional registers to avoid corrupting User mode state when the exception occurs.
- **System mode:** System mode is one of the privileged modes, and it shares the same register set as the User mode. It has full access to the system's privileged resources.

Switching between these modes is generally controlled by the ARM processor itself in response to certain events (like interrupts). However, software can also change modes explicitly, usually done by system-level code. This is achieved by modifying specific bits in the CPSR.

1.2.1 Exception level























Exception Levels (ELs): ARMv8 replaces the above modes with a hierarchy of exception levels:


- **EL0**: The least privileged level, typically for application code (similar to User mode in ARMv7).
- **EL1**: For operating system kernel code, similar to the Supervisor mode.
- **EL2**: A new level, typically used by hypervisors in virtualized environments.
- **EL3**: The most privileged level, used for secure boot and trusted execution environments, like TrustZone.

1.3 Registers

Registers

Modes						
<div>Privileged modes</div>						
<div>Exception modes</div>						
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	 R8_fiq
R9	R9	R9	R9	R9	R9	 R9_fiq
R10	R10	R10	R10	R10	R10	 R10_fiq
R11	R11	R11	R11	R11	R11	 R11_fiq
R12	R12	R12	R12	R12	R12	 R12_fiq
R13	R13	 R13_svc	 R13_abt	 R13_und	 R13_irq	 R13_fiq
R14	R14	 R14_svc	 R14_abt	 R14_und	 R14_irq	 R14_fiq
PC	PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		 SPSR_svc	 SPSR_abt	 SPSR_und	 SPSR_irq	 SPSR_fiq

 indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

- Total of 37 registers(Sum of the registers in the picture above is 37)
 - 31 general-purpose 32-bit registers, including a program counter. R0 ~ R15 visible at any time. (The PC is no longer treated as a general-purpose register in ARMv8 or later)
 - 6 status registers

1.3.1 General-purpose registers

- **Unbanked Registers (R0 to R7 and R15):** Each of them refers to the same 32-bit physical register in all processor modes. They are completely general-purpose registers, with no special uses implied by the architecture
- **Banked Registers (R8 to R14):** Different physical registers depending on the processor mode. A specific name is used to point to a particular physical register.
 - Almost all instructions allow the banked registers to be used wherever a general-purpose register is allowed.
 - Some of the banked registers are unique to the mode, while others are shared (or overlapped) with other modes:
 - Registers R8 to R12 have two banked physical registers each. One is used in all processor modes other than FIQ mode, and the other is used in FIQ mode.
 - Registers R13 and R14 have six banked physical registers each. One is used in User and System modes, and each of the remaining five is used in one of the five exception modes.

1.3.2 Special Uses of these registers

- R13 (Stack Pointer): Used for stack operations.
- R14 (Link Register): Holds return address after a subroutine call. This is more efficient than using a call stack for every function call, which INTEL uses.
- R15 (Program Counter)(deprecated): R15 can be used in place of other general-purpose registers for certain special-case effects.

By default, R15 operates as a program counter, used for storing the address of the next's next instruction. Storing the next's next instruction is due to the pipeline, where instructions are pre-fetched.

1.4 Program Status Registers

- **Current Program Status Register(CPSR):** Holds processor status, condition code flags, interrupt disable bits, processor mode, etc.
- **Saved Program Status Register(SPSR):** Holds the CPSR of the task before an exception occurred. Each exception mode has a SPSR.

Table A1-1 Status register summary

Field	Description	Architecture
N Z C V	Condition code flags	All
J	Jazelle state flag	5TEJ and above
GE[3:0]	SIMD condition flags	6
E	Endian Load/Store	6
A	Imprecise Abort Mask	6
I	IRQ Interrupt Mask	All
F	FIQ Interrupt Mask	All
T	Thumb state flag	4T and above
Mode[4:0]	Processor mode	All

1.4.1 Types of PSR bits

31	30	29	28	27	26	25	24	23	20	19	16	15	10	9	8	7	6	5	4	0
N	Z	C	V	Q	Res	J	RESERVED	GE[3:0]	RESERVED	E	A	I	F	T	M[4:0]					

Permissions

- **Reserved bits:** Reserved for future expansion. Implementations must read these bits as 0 and ignore writes to them.
- **User-writable bits(N, Z, C, V, Q, GE[3:0], E):** Can be written from any mode.
- **Privileged bits(A, I, F, and M[4:0]):** Can be written from any privileged mode. Writes to privileged bits in User mode are ignored.
- **Execution state bits(J, T):** Can be written from any privileged mode. Writes to execution state bits in User mode are ignored.

1.4.2 The condition code flags

The N, Z, C, and V (Negative, Zero, Carry and oVerflow) bits are collectively known as the condition code flags, often referred to as flags.

- The condition code flags in the CPSR can be tested by most instructions to determine whether the instruction is to be executed.(if else)
- They are usually modified by the execution of some arithmetic, logical, move instruction, or comparison instruction (CMN, CMP, TEQ or TST).

1.4.3 The Q flag

bit[27] of the CPSR is known as the Q flag and is used to indicate whether overflow and/or saturation has occurred in some DSP-oriented instructions.

1.4.4 The GE[3:0] bits

The SIMD instructions use bits[19:16] as Greater than or Equal (GE) flags for individual bytes or halfwords of the result.
You can use these flags to control a later SEL instruction

1.4.5 The E bit

Controls load and store endianness for data handling.

1.4.6 The interrupt disable bits

- **A bit:** Disables imprecise data aborts when it is set. This is available only in ARMv6 and above. In earlier versions, bit[8] of CPSR and SPSRs must be treated as a reserved bit, as described in Types of PSR bits on page A2-11. **I bit** Disables IRQ interrupts when it is set. **F bit** Disables FIQ interrupts when it is set.

1.4.7 The mode bits

M[4:0] are the mode bits. These determine the mode in which the processor operates.

Table A2-2 The mode bits

M[4:0]	Mode	Accessible registers
0b10000	User	PC, R14 to R0, CPSR
0b10001	FIQ	PC, R14_fiq to R8_fiq, R7 to R0, CPSR, SPSR_fiq
0b10010	IRQ	PC, R14_irq, R13_irq, R12 to R0, CPSR, SPSR_irq
0b10011	Supervisor	PC, R14_svc, R13_svc, R12 to R0, CPSR, SPSR_svc
0b10111	Abort	PC, R14_abt, R13_abt, R12 to R0, CPSR, SPSR_abt
0b11011	Undefined	PC, R14_und, R13_und, R12 to R0, CPSR, SPSR_und
0b11111	System	PC, R14 to R0, CPSR (ARMv4 and above)

1.4.8 The T and J bits

Table A2-3 The T and J bits

J	T	Instruction set
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	RESERVED

The T and J bits select the current instruction set, as shown in Table A2-3.

1.4.9 Other bits

Other bits in the program status registers are reserved for future expansion.

In general, programmers must take care to write code in such a way that these bits are never modified.

Failure to do this might result in code that has unexpected side effects on future versions of the architecture.

1.5 Exceptions

Exceptions are generated by internal and external sources to cause the processor to handle an event, such as an externally generated interrupt.

More than one exception can arise at the same time.

All exception modes have replacement banked registers. When an exception occurs, standard registers are replaced with registers specific to the exception mode.

1.5.1 Exception process

- When an exception occurs, execution is forced from a fixed memory address corresponding to the type of exception. These fixed addresses are called the **exception vectors**. There is a separate vector location for each exception.
- The banked versions of R14 and the SPSR are used to save state so that the original program can be resumed when the exception routine has completed.
- To return after handling the exception, the SPSR is moved into the CPSR, and R14 is moved to the PC.

1.5.2 Types of exceptions

The ARM architecture supports seven types of exception. Table A2-4 lists the types of exception and the processor mode that is used to process each type.



1.5.2.1 Reset

When the Reset input is asserted on the processor, the ARM processor immediately stops execution of the current instruction.

1.5.2.2 Undefined Instruction exception

Occurs when the processor encounters an instruction that is either **not defined** in the ARM instruction set or is **not valid** for the current state of the processor

1.5.2.3 Software Interrupt exception

The Software Interrupt instruction (SWI) enters Supervisor mode to request a particular supervisor function. (OS system call)

1.5.2.4 Prefetch Abort (Instruction fetch memory abort)

- **Precondition to occur:** A memory abort is signaled by the memory system. Attempting to fetch an instruction from a memory location that it cannot access correctly marks the data as invalid.
- **Occurs when:** A Prefetch Abort exception is generated if the processor tries to execute the invalid instruction.
- **Does not occur when:** If the instruction is not executed (for example, as a result of a branch being taken while it is in the pipeline), no Prefetch Abort occurs.

1.5.2.5 Data Abort (Data access memory abort)

A data abort occurs when a problem arises during a data access operation, such as reading from or writing to memory.

1.5.2.6 Interrupt request (IRQ) exception

The IRQ exception is generated externally by asserting the IRQ input on the processor.

1.5.2.7 Fast interrupt request (FIQ) exception

The FIQ exception is generated externally by asserting the FIQ input on the processor. FIQ is designed to support a data transfer or channel process.

- Why fast interrupt is faster
 - **Dedicated Registers:** FIQ has additional banked registers(private) to remove the need for register saving in such applications, therefore minimizing the overhead of context switching.

- **Shorter vector:** There are fewer instructions to process before the ISR code is reached.
- **Higher Priority:** FIQ has a higher priority than IRQ

1.5.3 Exception priorities

Table A2-4 Exception processing modes

Exception type	Mode	VE ^a	Normal address	High vector address
Reset	Supervisor		0x00000000	0xFFFF0000
Undefined instructions	Undefined		0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor		0x00000008	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	Abort		0x0000000C	0xFFFF000C
Data Abort (data access memory abort)	Abort		0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0	0x00000018	0xFFFF0018
		1	IMPLEMENTATION DEFINED	
FIQ (fast interrupt)	FIQ	0	0x0000001C	0xFFFF001C
		1	IMPLEMENTATION DEFINED	

a. VE = vectored interrupt enable (CP15 control); RAZ when not implemented.

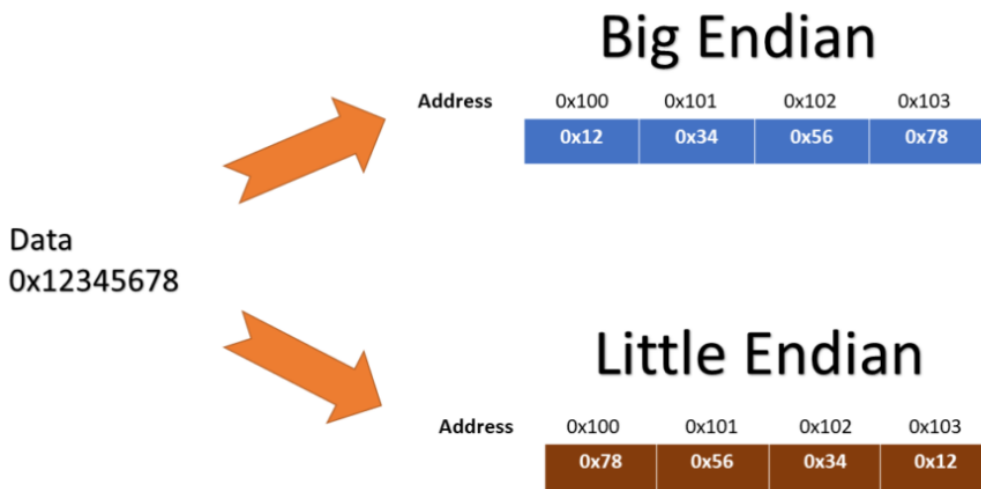
1.6 Endian

Endianness is the order of bytes of digital data.

ARMv6 supports both big-endian and little-endian operation

(Least Significant bit: the bit that has the lowest value)

- In little-endian mode, the least significant bit is stored at the smallest address. (most common)
- In big-endian mode, the most significant bit is stored at the smallest address.



1.7 Unaligned access support

- **Aligned Access:** An access is aligned if the memory address being accessed is a multiple of the size of the data type. For example, accessing a 4-byte integer at memory addresses 0, 4, 8, etc., is considered aligned.
Aligned memory accesses are typically faster because they align with the natural boundaries of memory buses and caches.
- **Unaligned Access:** An access is unaligned if the memory address is not a multiple of the size of the data type. For instance, accessing a 4-byte integer at memory addresses 2, 6, 10, etc., would be unaligned.

1.7.1 Changes with ARMv6:

Traditionally, ARM expects memory accesses to be aligned.
ARMv6 introduced support for unaligned word and halfword data access.

1.8 Synchronization primitives

1.8.1 LDREX (Load-Exclusive)

- When a thread executes an LDREX instruction, it reads data from a memory location and the processor marks this location as having been accessed exclusively by this thread.
- However, this 'exclusive' mark doesn't block other threads or cores from accessing the same memory location. They can still read from and write to this location.

1.8.2 STREX (Store-Exclusive)

- STREX checks if the memory location still has its exclusive access status. If no other thread has written to this location since the LDREX was executed, STREX considers the operation successful and writes the new value.
- If, however, another thread has written to the location, the STREX operation will fail, returning a non-zero value.

1.8.3 Typical Usage in a Loop:

In practice, the thread often repeatedly reads (LDREX) and tries to write (STREX) in a loop until STREX reports success (indicating no other writes occurred in the meantime).

1.9 Jazelle extension

- **Background:** Jazelle was introduced to mitigate the overheads of JVM by allowing for the native execution of Java bytecode.
- **Operation:** When in Jazelle mode, the processor can directly execute Java bytecode instructions. This is achieved by mapping Java bytecodes to equivalent ARM processor instructions or sequences of instructions.
- **Decline of Jazelle:** The advancements in JIT have largely overshadowed the need for direct bytecode execution.

1.10 Saturated integer arithmetic

Saturated arithmetic prevents the overflow/underflow. Instead, if an operation results in a value outside the representable range, it is set to the closest representable value (the maximum or minimum value of the data type).

- **For instance,** in 8-bit saturated arithmetic, if adding two numbers results in a value greater than 255, the result is set to 255. Similarly, if subtraction would result in a negative value, the result is set to 0.