**ARM**

The **ARM** is a [32-bit](http://en.wikipedia.org/wiki/32-bit) [reduced instruction set computer](http://en.wikipedia.org/wiki/Reduced_instruction_set_computer) (RISC) [instruction set architecture](http://en.wikipedia.org/wiki/Instruction_set_architecture) (ISA) developed by [ARM Holdings](http://en.wikipedia.org/wiki/ARM_Holdings). It was known as the **Advanced RISC Machine**, and before that as the **Acorn RISC Machine.**

**RISC/CISC(i.e advantages of ARM)**

*Instructions*—RISC processors have a reduced number of instruction classes.

*Pipelines*—The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines.

*Registers*—RISC machines have a large general-purpose register set. Any register can contain either data or an address. In contrast, CISC processors have dedicated registers for specific purposes.

*Load-store architecture*—The processor operates on data held in registers. Separate load and store instructions transfer data between the register bank and external memory.Memory accesses are costly, so separating memory accesses from data processing provides an advantage because you can use data items held in the register bank multiple times without needing multiple memory accesses. In contrast, with a CISC design the data processing operations can act on memory directly.

In contrast, traditional CISC processors are more complex and operate at lower clock frequencies.

**What is load & store**

* **load** To *load* a value from memory, you copy the data from memory into a register.
* **store** To *store* a value to memory, you copy the data from a register to memory.

**Registers**

There are up to 18 active registers: 16 data registers and 2 processor status registers. The data registers are visible to the programmer as *r0* to *r15.* The ARM processor has three registers assigned to a particular task or special function:

*r13*, *r14*, and *r15*.

* Register *r13* is traditionally used as the stack pointer (*sp*) and stores the head of the stack in the current processor mode.
* Register *r14* is called the link register (*lr*) and is where the core puts the return address whenever it calls a subroutine.
* Register *r15* is the program counter (*pc*) and contains the address of the next instruction to be fetched by the processor.

In addition to the 16 data registers, there are two program status registers: *cpsr* and *spsr* (the current and saved program status registers, respectively).

**Processor modes**

Each processor mode is either privileged or nonprivileged: A privileged mode allows full read-write access to the *cpsr*. Conversely, a nonprivileged mode only allows read access to the control field in the *cpsr* but still allows read-write access to the condition flags.

There are seven processor modes in total: six privileged modes (*abort*, *fast interrupt request*, *interrupt request*, *supervisor*, *system*, and *undefined*) and one nonprivileged mode(*user*).

The processor enters *abort* mode when there is a failed attempt to access memory. *Fast interrupt request* and *interrupt request* modes correspond to the two interrupt levels availableon the ARM processor. *Supervisor* mode is the mode that the processor is in after reset andis generally the mode that an operating system kernel operates in. *System* mode is a specialversion of *user* mode that allows full read-write access to the *cpsr*. *Undefined* mode is used

when the processor encounters an instruction that is undefined or not supported by the implementation. *User* mode is used for programs and applications.

**Exceptions, Interrupts, and the Vector Table**

When an exception or interrupt occurs, the processor sets the *pc* to a specific memory address. The address is within a special address range called the *vector table*. The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt.

The memory map address 0x00000000 is reserved for the vector table, a set of 32-bit words. On some processors the vector table can be optionally located at a higher address in memory (starting at the offset 0xffff0000).

When an exception or interrupt occurs, the processor suspends normal execution and starts loading instructions from the exception vector table . Each vector table entry contains a form of branch instruction pointing to the start of a specific routine:

* *Reset vector* is the location of the first instruction executed by the processor when power is applied. This instruction branches to the initialization code.
* *Undefined instruction vector* is used when the processor cannot decode an instruction.
* *Software interrupt vector* is called when you execute a SWI instruction. The SWI instruction is frequently used as the mechanism to invoke an operating system routine.
* *Prefetch abort vector* occurs when the processor attempts to fetch an instruction from an address without the correct access permissions. The actual abort occurs in the decode stage.
* *Data abort vector* is similar to a prefetch abort but is raised when an instruction attempts to access data memory without the correct access permissions.
* *Interrupt request vector* is used by external hardware to interrupt the normal execution flow of the processor. It can only be raised if IRQs are not masked in the *cpsr*.
* *Fast interrupt request vector* is similar to the interrupt request but is reserved for hardware requiring faster response times. It can only be raised if FIQs are not masked in the *cpsr*.

The vector table.

**Exception/interrupt Shorthand Address High address**

Reset RESET 0x00000000 0xffff0000

Undefined instruction UNDEF 0x00000004 0xffff0004

Software interrupt SWI 0x00000008 0xffff0008

Prefetch abort PABT 0x0000000c 0xffff000c

Data abort DABT 0x00000010 0xffff0010

Reserved — 0x00000014 0xffff0014

Interrupt request IRQ 0x00000018 0xffff0018

Fast interrupt request FIQ 0x0000001c 0xffff001c

**Nomenclature**

**ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{-S}**

x—family

y—memory management/protection unit

z—cache

T—Thumb 16-bit decoder

D—JTAG debug

M—fast multiplier

I—EmbeddedICE macrocell

E—enhanced instructions (assumes TDMI)

J—Jazelle

F—vector floating-point unit

S—synthesizible version

Some points about the nomenclature

All ARM cores after the ARM7TDMI include the *TDMI* features even though they may

not include those letters after the “ARM” label.

* The processor *family* is a group of processor implementations that share the same hardware characteristics. For example, the ARM7TDMI, ARM740T, and ARM720T all share the same family characteristics and belong to the ARM7 family.
* *JTAG* is described by IEEE 1149.1 Standard Test Access Port and boundary scan architecture.It is a serial protocol used by ARM to send and receive debug information between the processor core and test equipment.
* *EmbeddedICE macrocell* is the debug hardware built into the processor that allows breakpoints and watchpoints to be set.
* *Synthesizable* means that the processor core is supplied as source code that can be compiled into a form easily used by EDA tools.

CPSR

31 28 27 8 7 6 5 4 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NZCV | Unused | I | F | T | Mode |

In user programs only the top 4 bits of the CPSR are relevant

* N - the result was negative
* Z - the result was zero
* C - the result produced a carry out
* V - the result generated an arithmetic overflow
* I, F – interrupt enable bits
* T – instruction set (Thumb/ARM)

**Pipelining**

An efficient technique to complete an average of one instruction per cycle.

ARM7 has a 3-stage pipeline.   
ARM9 has a 5-stage pipeline.  
ARM10 has 6 and ARM11 has 8, both with branch prediction to avoid pipeline stall due to branching.

ARM operates only in pipeline mode (3 or 5 pipes).Any instruction can be divided into different cycles which determine the pipe length. For 3 stages pipe (fetch- decode - execute):During fetch memory is accesed for instruction.During decode control logic makes control for the execute cycle.During execute dataflow path is occupied.Without pipelining each instruction will pass by the 3 phases & needs three cycles to execute.

A simple **3-stage pipeline** consists of fetch, decode and execute.  
fetch - to fetch the instruction from the code memory indicated by the program counter.  
decode - interpret the opcode from the instruction.  
execute - based on the opcode, perform required operation on the operand(s).

How a 3-stage pipeline works?  
  
Time Fetch Decode Execute  
------------------------------------------------  
time 0: Fetch 1 - -  
time 1: Fetch 2 Decode 1 -  
time 2: Fetch 3 Decode 2 Execute 1  
time 3: Fetch 4 Decode 3 Execute 2

A 5 stage pipeline consists of

1. Instruction fetch
2. Instruction decode and register fetch
3. Execute
4. Memory access
5. Register write back

**Exception/Interrupt**

*Exception handling.* Exception handling covers the specific details of how the ARM processor handles exceptions. ARM defines an *interrupt* as a special type of exception.

An exception is any condition that needs to halt the normal sequential execution of instructions. Examples are when theARMcore is reset, when an instruction fetch or memory access fails, when an undefined instruction is encountered, when a software interrupt instruction is executed, or when an external interrupt has been raised. Exception handling is the method of processing these exceptions.

**Basic of what happens when interrupt occurs**

1. *Disable interrupt/s*—When the IRQ exception is raised, the ARM processor will disable further IRQ exceptions from occurring. The processor mode is set to the appropriate interrupt request mode, and the previous *cpsr* is copied into the newly available *spsr\_{interrupt request mode}*. The processor will then set the *pc* to point to the correct

entry in the vector table and execute the instruction. This instruction will alter the *pc* to point to the specific interrupt handler.

2. *Save context*—On entry the handler code saves a subset of the current processor mode nonbanked registers.

3. *Interrupt handler*—The handler then identifies the external interrupt source and executes the appropriate interrupt service routine (ISR).

4. *Interrupt service routine*—The ISR services the external interrupt source and resets the interrupt.

5. *Restore context*—The ISR returns back to the interrupt handler, which restores the context.

6. *Enable interrupts*—Finally, to return from the interrupt handler, the *spsr\_{interrupt request mode}* is restored back into the *cpsr*. The *pc* is then set to the next instructionafter the interrupt was raised.

(The above is a case when it is a nonnested internet handler,there are a lot of other cases i.e nested interrupt handler,a reentrant interrupt handler,prioritized simple interrupt handler)

**Harvard/Von Neuman**

**Harvard architecture** has separate data and instruction busses, allowing transfers to be performed simultaneously on both busses. A **vonNeumann architecture** has only one bus which is used for both data transfers and instruction fetches, and therefore data transfers and instruction fetches must be scheduled - they can not be performed at the same time.