Architecture

Von Neumann implementation of the ARM

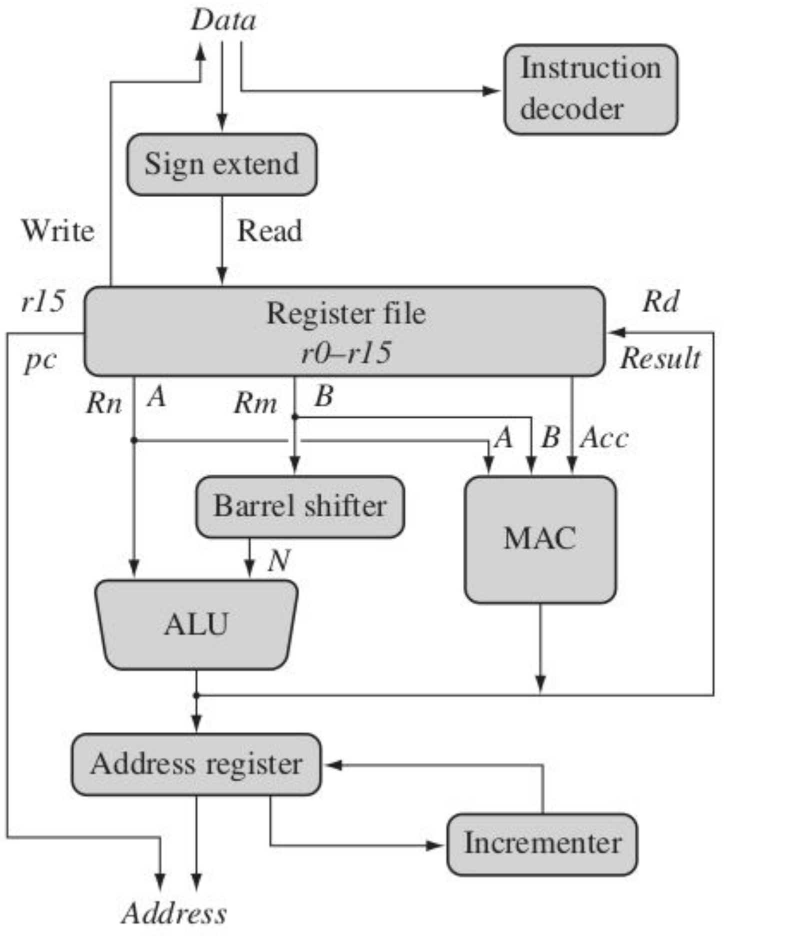


Figure 2.1 ARM core dataflow model.

components that make up an ARM core

functional units connected by data buses

arrows represent the flow of data

lines represent the buses

boxes represent either an operation unit or a storage area

Data enters the processor core through the Data bus.

The data may be an instruction to execute or a data item.

The instruction decoder translates instructions before they are executed.

Each instruction executed belongs to a particular instruction set.

The ARM processor, like all RISC processors, uses a load-store architecture.

It has two instruction types for transferring data in and out of the processor: load instructions copy data from memory to registers in the core, and conversely the store instructions copy data from registers to memory.

There are no data processing instructions that directly manipulate data in memory. Thus, data processing is carried out solely in registers.

Data items are placed in the register file—a storage bank made up of 32-bit registers.

registers can hold signed or unsigned 32-bit values

ARM instructions typically have two source registers, Rn and Rm, and a single result or destination register, Rd.

Source operands are read from the register file using the internal buses A and B, respectively.

The ALU (arithmetic logic unit) or MAC (multiply-accumulate unit) takes the register values Rn and Rm from the A and B buses and computes a result.

Data processing instructions write the result in Rd directly to the register file.

Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the Address bus.

One important feature of the ARM is that register Rm alternatively can be preprocessed in the barrel shifter before it enters the ALU.

Together the barrel shifter and ALU can calculate a wide range of expressions and addresses.

After passing through the functional units, the result in Rd is written back to the register file using the Result bus.

For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location.

The processor continues executing instructions until an exception or interrupt changes the normal execution flow.

Registers

General-purpose registers hold either data or an address.

They are identified with the letter r prefixed to the register number.

For example, register 4 is given the label r4.

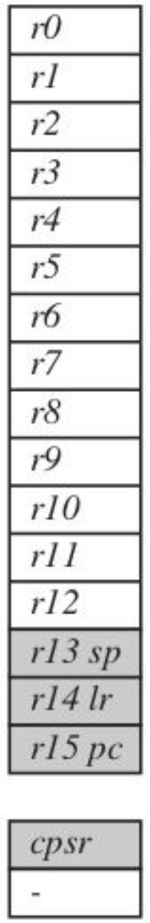


Figure 2.2 Registers available in *user* mode.

User mode - a protected mode normally used when executing applications

All the registers shown are 32 bits in size.

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.

They are frequently given different labels to differentiate them from the other registers.

the shaded registers identify the assigned special-purpose registers:

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.

registers r13 and r14 can also be used as general-purpose registers

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

The register file contains all the registers available to a programmer.

Which registers are visible to the programmer depend upon the current mode of the processor.

Current Program Status Register

The ARM core uses the cpsr to monitor and control internal operations.

The cpsr is a dedicated 32-bit register and resides in the register file.

shaded parts are reserved for future expansion

The cpsr is divided into four fields, each 8 bits wide: flags, status, extension, and control.

In current designs the extension and status fields are reserved for future use.

The control field contains the processor mode, state, and interrupt mask bits. The flags field contains the condition flags.

Some ARM processor cores have extra bits allocated.

For example, the J bit, which can be found in the flags field, is only available on Jazelle-enabled processors, which execute 8-bit instructions.

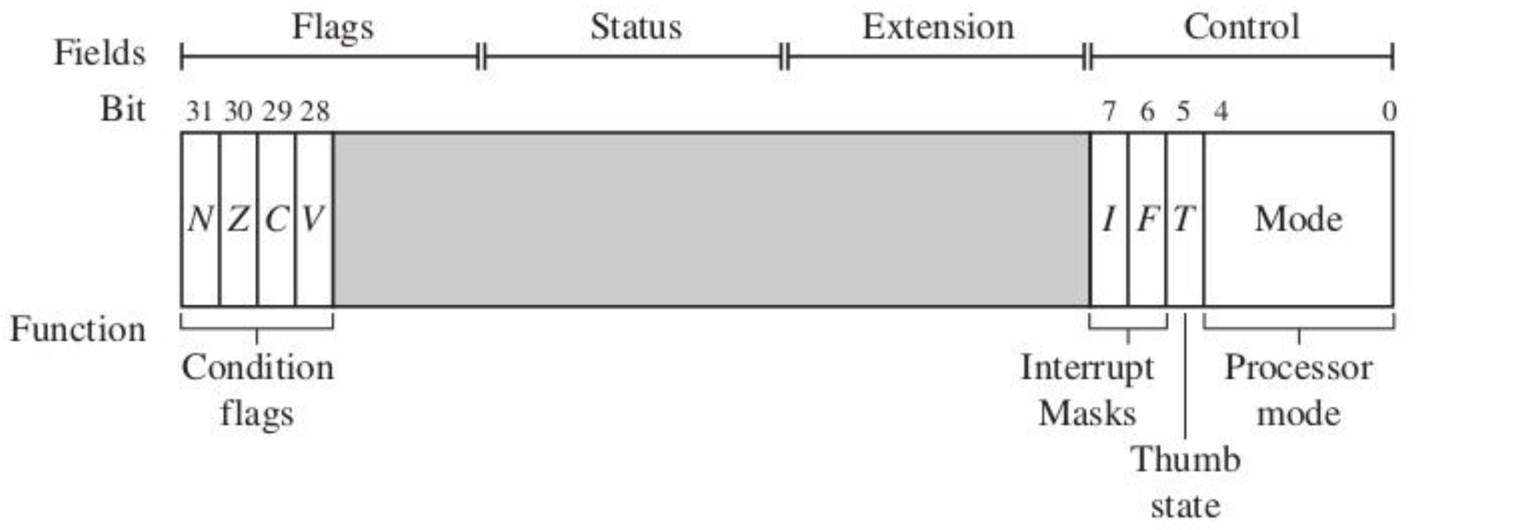


Figure 2.3 A generic program status register (*psr*).

Processor Modes

- The processor mode determines which registers are active and the access rights to the cpsr register itself.
- Each processor mode is either privileged or nonprivileged.
- A privileged mode allows full read-write access to the cpsr.
- 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 available on the ARM processor.

Supervisor mode is the mode that the processor is in after reset and is generally the mode that an operating system kernel operates in.

System mode is a special version 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.

Banked Registers

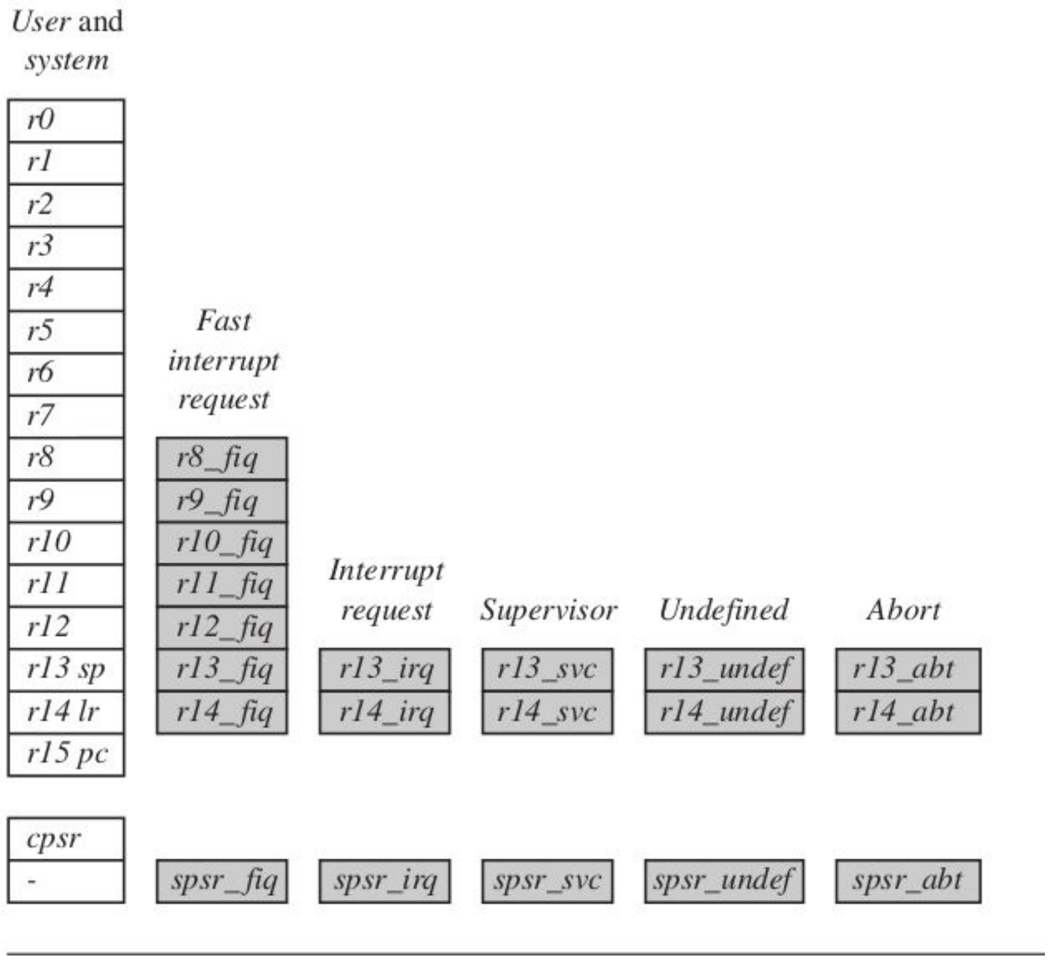


Figure 2.4 Complete ARM register set.

37 registers are in the register file

banked registers - 20 registers are hidden from a program at different times, identified by the shading

They are available only when the processor is in a particular mode

abort mode has banked registers r13_abt, r14_abt and spsr_abt

Banked registers of a particular mode are denoted by an underline character post-fixed to the mode mnemonic or _mode.

Every processor mode except user mode can change mode by writing directly to the mode bits of the cpsr.

All processor modes except system mode have a set of associated banked registers that are a subset of the main 16 registers.

A banked register maps one-to-one onto a user mode register.

If you change processor mode, a banked register from the new mode will replace an existing register.

The processor mode can be changed by a program that writes directly to the cpsr (the processor core has to be in privileged mode) or by hardware when the core responds to an exception or interrupt.

The following exceptions and interrupts cause a mode change:

reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort, and undefined instruction.

Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location.

Saved program status register (spsr) - a new register appearing in interrupt request mode, stores the previous mode’s cpsr

cpsr is not copied into the spsr when a mode change is forced due to a program writing directly to the cpsr. The saving of the cpsr only occurs when an exception or interrupt is raised.

When power is applied to the core,

it starts in supervisor mode, which is privileged.

Starting in a privileged mode is useful since

initialization code can use full access to the

cpsr to set up the stacks for each of the other

modes.

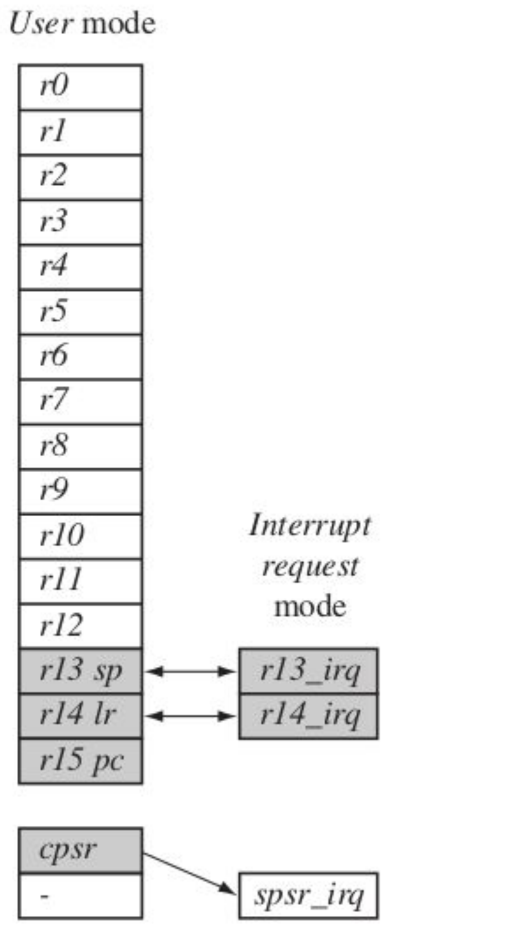


Figure 2.5 Changing mode on an exception.

Table 2.1 Processor mode.

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

State and Instruction Sets

The state of the core determines which instruction set is being executed.

There are three instruction sets: ARM, Thumb, and Jazelle.

The ARM instruction set is only active when the processor is in ARM state.

The Thumb instruction set is only active when the processor is in Thumb state.

In Thumb state the processor is executing purely Thumb 16-bit instructions.

Intermingling of sequential ARM, Thumb, and Jazelle instructions are not allowed.

The Jazelle J and Thumb T bits in the cpsr reflect the state of the processor.

When both J and T bits are 0, the processor is in ARM state and executes ARM instructions (when power is applied to the processor).

When the T bit is 1, then the processor is in Thumb state.

To change states the core executes a specialized branch instruction.

A third instruction set is called Jazelle.

Jazelle executes 8-bit instructions and is a hybrid mix of software and hardware designed to speed up the execution of Java bytecodes.

To execute Java bytecodes, you require the Jazelle technology plus a specially modified version of the Java virtual machine.

It is important to note that the hardware portion of Jazelle only supports a subset of the Java bytecodes; the rest are emulated in software.

Table 2.2 ARM and Thumb instruction set features.

	ARM (*cpsr* $T = 0$)	Thumb (*cpsr* $T = 1$)
Instruction size	32-bit	16-bit
Core instructions	58	30
Conditional execution[a]	most	only branch instructions
Data processing instructions	access to barrel shifter and ALU	separate barrel shifter and ALU instructions
Program status register	read-write in privileged mode	no direct access
Register usage	15 general-purpose registers +*pc*	8 general-purpose registers +7 high registers +*pc*

The Jazelle instruction set is a closed instruction set and is not openly available.

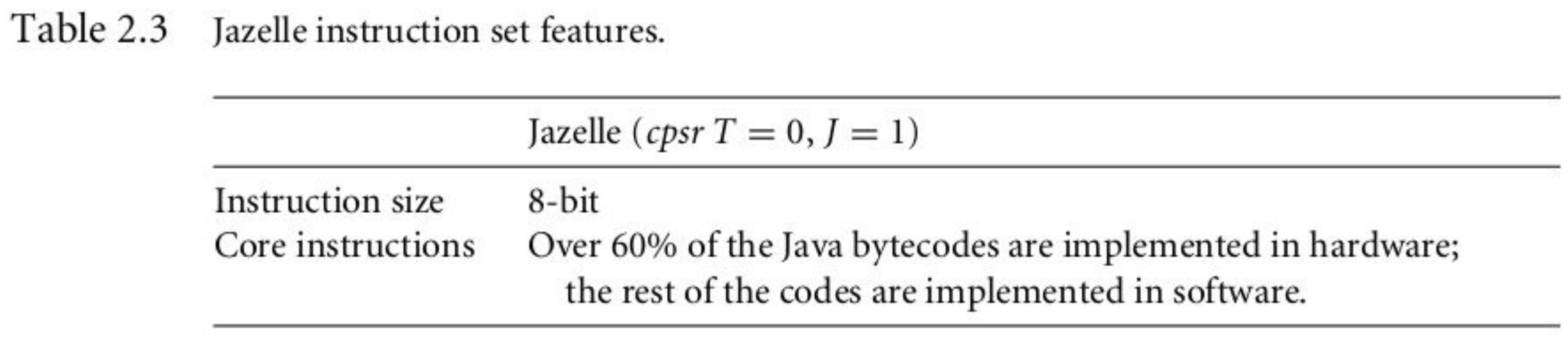


Table 2.3 Jazelle instruction set features.

	Jazelle ($cpsr$ $T = 0, J = 1$)
Instruction size	8-bit
Core instructions	Over 60% of the Java bytecodes are implemented in hardware; the rest of the codes are implemented in software.

Interrupt Masks

Interrupt masks are used to stop specific interrupt requests from interrupting the processor.

There are two interrupt request levels available on the ARM processor core—interrupt request (IRQ) and fast interrupt request (FIQ).

The cpsr has two interrupt mask bits, 7 and 6 (or I and F), which control the masking of IRQ and FIQ, respectively.

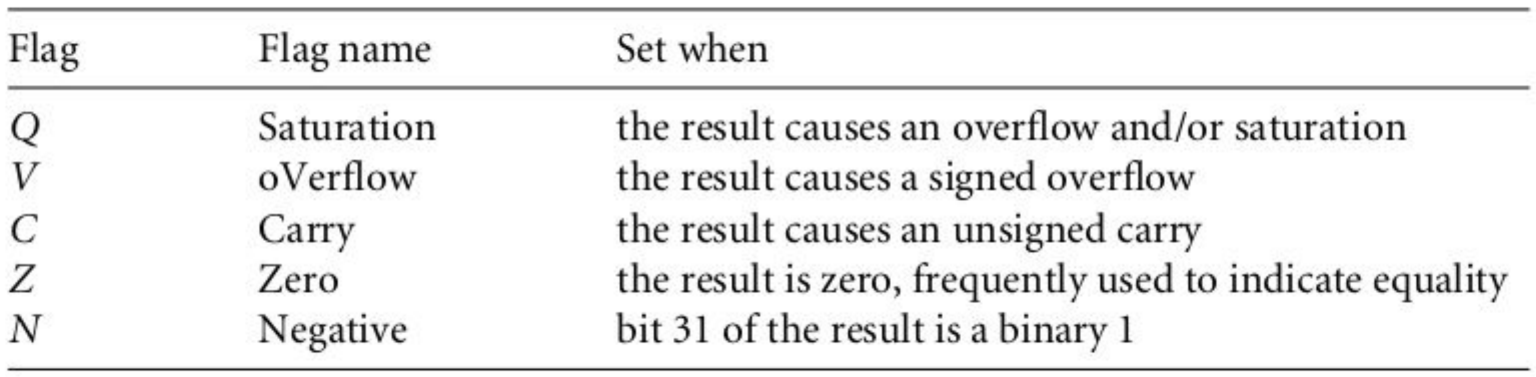
The I bit masks IRQ when set to binary 1, and similarly the F bit masks FIQ when set to binary 1.

Condition Flags

Condition flags are updated by comparisons and the result of ALU operations that specify the S instruction suffix.

if a SUBS subtract instruction results in a register value of zero, then the Z flag in the cpsr is set.

Condition flags.



Flag	Flag name	Set when
Q	Saturation	the result causes an overflow and/or saturation
V	oVerflow	the result causes a signed overflow
C	Carry	the result causes an unsigned carry
Z	Zero	the result is zero, frequently used to indicate equality
N	Negative	bit 31 of the result is a binary 1

With processor cores that include the DSP extensions, the Q bit indicates if an overflow or saturation has occurred in an enhanced DSP instruction.

The flag is “sticky” in the sense that the hardware only sets this flag.

To clear the flag you need to write to the cpsr directly.

In Jazelle-enabled processors, the J bit reflects the state of the core; if it is set, the core is

in Jazelle state.

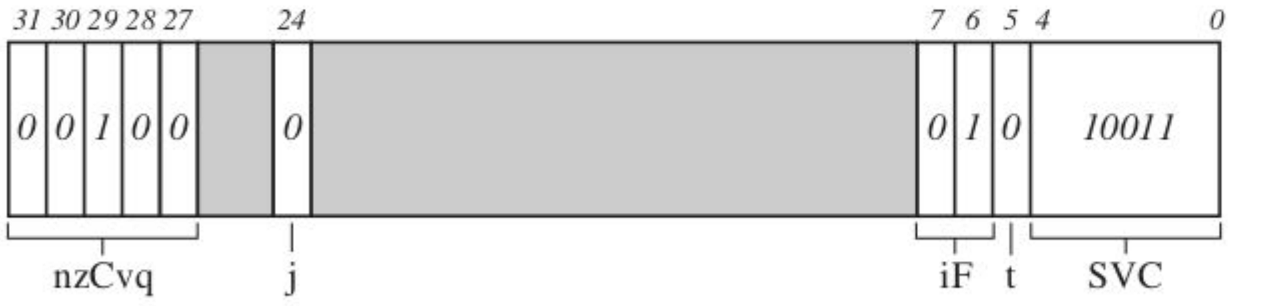
The J bit is not generally usable and is only available on some processor cores.

To take advantage of Jazelle, extra software has to be licensed from both ARM Limited and Sun Microsystems.

Most ARM instructions can be executed conditionally on the value of the condition flags.

These flags are located in the most significant bits in the cpsr.

These bits are used for conditional execution.



Example: *cpsr* = *nzCvqjiFt_SVC*.

When a bit is a binary 1 we use a capital letter; when a bit is a binary 0, we use a lowercase letter.

For the condition flags a capital letter shows that the flag has been set. For interrupts a capital letter shows that an interrupt is disabled.

the C flag is the only condition flag set. The rest nzvq flags are all clear. The processor is in ARM state because neither the Jazelle j or Thumb t bits are set. The IRQ interrupts are enabled, and FIQ interrupts are disabled.

the processor is in supervisor (SVC) mode since the mode[4:0] is equal to binary 10011

Conditional Execution

Conditional execution controls whether or not the core will execute an instruction.

Most instructions have a condition attribute that determines if the core will execute it

based on the setting of the condition flags.

Prior to execution, the processor compares the condition attribute with the condition flags in the cpsr.

If they match, then the instruction is executed; otherwise the instruction is ignored.

The condition attribute is postfixed to the instruction mnemonic, which is encoded into the instruction. Table 2.5 lists the conditional execution code mnemonics.

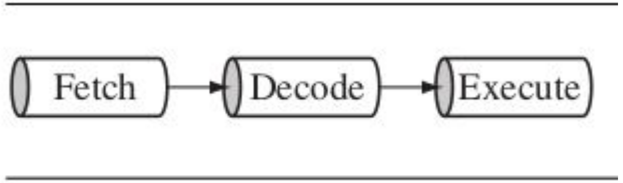
When a condition mnemonic is not present, the default behavior is to set it to always (AL) execute.

Table 2.5 Condition mnemonics.

Mnemonic	Name	Condition flags
`EQ`	equal	*Z*
`NE`	not equal	*z*
`CS HS`	carry set/unsigned higher or same	*C*
`CC LO`	carry clear/unsigned lower	*c*
`MI`	minus/negative	*N*
`PL`	plus/positive or zero	*n*
`VS`	overflow	*V*
`VC`	no overflow	*v*
`HI`	unsigned higher	*zC*
`LS`	unsigned lower or same	*Z* or *c*
`GE`	signed greater than or equal	*NV* or *nv*
`LT`	signed less than	*Nv* or *nV*
`GT`	signed greater than	*NzV* or *nzv*
`LE`	signed less than or equal	*Z* or *Nv* or *nV*
`AL`	always (unconditional)	ignored

Pipeline

A pipeline is the mechanism a RISC processor uses to execute instructions. Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed. One way to view the pipeline is to think of it as an automobile assembly line, with each stage carrying out a particular task to manufacture the vehicle.

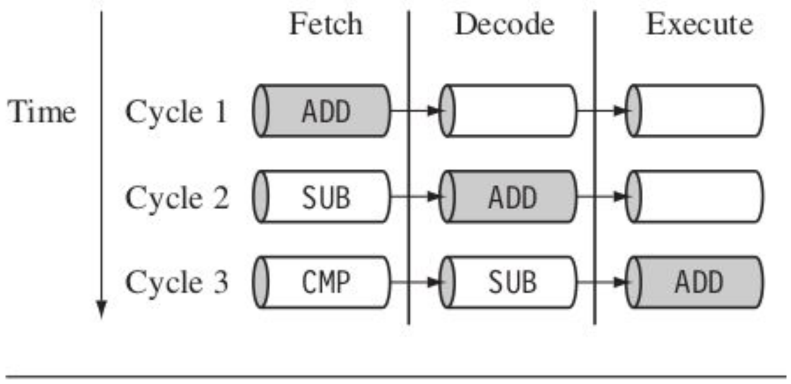


ARM7 Three-stage pipeline.

Fetch loads an instruction from memory.

Decode identifies the instruction to be executed.

Execute processes the instruction and writes the result back to a register.



Pipelined instruction sequence.

A sequence of three instructions being fetched, decoded, and executed by the processor.

Each instruction takes a single cycle to complete after the pipeline is filled.

The three instructions are placed into the pipeline sequentially.

In the first cycle the core fetches the ADD instruction from memory.

In the second cycle the core fetches the SUB instruction and decodes the ADD instruction.

In the third cycle, both the SUB and ADD instructions are moved along the pipeline.

The ADD instruction is executed, the SUB instruction is decoded, and the CMP instruction is fetched.

This procedure is called filling the pipeline.

The pipeline allows the core to execute an instruction every cycle.

As the pipeline length increases, the amount of work done at each stage is reduced, which allows the processor to attain a higher operating frequency. This in turn increases the performance. The system latency also increases because it takes more cycles to fill the pipeline before the core can execute an instruction. The increased pipeline length also means there can be data dependency between certain stages. This dependency can be reduced by instruction scheduling.

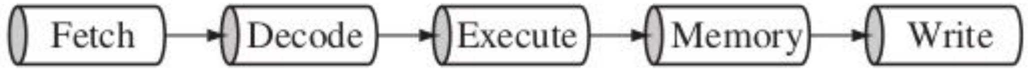


Figure 2.9 ARM9 five-stage pipeline.

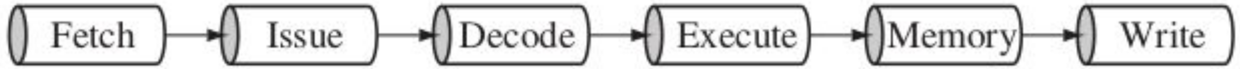


Figure 2.10 ARM10 six-stage pipeline.

Pipeline Executing Characteristics

The ARM pipeline has not processed an instruction until it passes completely through the execute stage.

For example, an ARM7 pipeline (with three stages) has executed an instruction only when the fourth instruction is fetched.