Unit – I

8 Hours

Microprocessors versus Microcontrollers, ARM Embedded Systems: The RISC design philosophy, The ARM Design Philosophy, Embedded System Hardware, Embedded System Software. ARM Processor Fundamentals: Registers, Current Program Status Register, Pipeline, Exceptions, Interrupts, and the Vector Table, Core Extensions.

Text Book: ARM System Developer's Guide, Designing and Optimizing System Software Andrew N. Sloss Dominic Symes Chris Wright

- **Microcontroller Vs Microprocessor:**

- What is a Microprocessor?

  A processing device implemented on a single chip.

- What is a Microcontroller?

  An electronic system which consists of a processing element, a small memory (RAM, ROM, EPROM), I/O ports, etc. on a single chip. (Onchip computer)

- What is an Embedded System?

  An embedded system is a combination of computer hardware and software designed for a specific function.

Sr. No	*Microprocessor*	*Microcontroller*
1.	We need to connect peripherals externally. So it makes circuit bulky.	The presence of peripherals such as RAM, ROM, Input-output, and Timers are In-built. So It is available on a single chip.
2.	It increases the overall cost of the system high.	The overall cost of the system is less.
3.	We can connect external memory in ranges of Mbytes and even Gbytes. But speed is less.	The inbuilt finite memory helps to improve the speed of operations.
4.	You can't use it in a compact system.	You can use it in compact systems.
5.	Due to external components, the total power consumption is high. Therefore, it is not ideal for the devices running on stored power like batteries.	As external components are low, total power consumption is less. So it can be used with devices running on stored power like batteries.
6.	Most of the microprocessors do not have power-saving features.	Most of the microcontrollers offer power-saving mode.
7.	The microprocessor has a smaller number of registers, so more operations are memory-based.	The microcontroller has more register. Hence the programs are easier to write.
8.	These are based on the von Neumann model where program and data are stored in the same memory module.	These are based on Harvard architecture where program memory and data memory are separate.
9.	It is a central processing unit on a single silicon-based integrated chip.	It is a byproduct of the development of microprocessors with a CPU along with other peripherals.
10	It uses an external bus to interface to RAM, ROM, and other peripherals.	It uses an internal controlling bus.
11	Microprocessor-based systems can run at a very high speed because of the technology involved.	Microcontroller based systems run up to 200MHz or more depending on the architecture.
12	It's useful for general purpose applications that allow you to handle loads of data.	It's useful for application-specific systems.
13	It's complex and expensive, with a large number of instructions to process.	It's simple and inexpensive with less number of instructions to process.

What is ARM core?

- **ARM(Advanced RISC Machine) microcontroller is a 32-bit architecture microcontroller** that was developed by Acorn Computers in 1983.
- Belongs to basically a family of Reduced Instruction Set Computing (RISC) architecture-based microprocessors. ARM microcontrollers consist of ARM processors, RAM, ROM, and I/O peripherals.
- ARM processor core is a key component of many successful 32-bit Embedded systems.
- Optimized for low power & high performance.
- Note: **90% of the mobile markets ruled by the ARM architecture technology.**

- **Applications:** Mobile phones, everyday portable consumer devices.

Embedded Solutions
• Smart Cards • Automotive • Dashboard Controls
Enterprise Solutions
• Flash Cards • Hard Disks • Printers
Home Solutions
• Still Cameras • Digital TV • Set Top Box
Mobile Solutions
• Bluetooth • PDA • Smart Phone
Emerging Applications
• Gaming • Robot • And much more….

The ARM7TDMI-S:

- Is a general-purpose 32-bit microprocessor, which offers high performance and very low power consumption.
- Based on Reduced Instruction Set (RISC) architecture, This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective processor Core.
- Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.
- Has two Instruction sets:

  The standard 32-bit ARM instruction set.

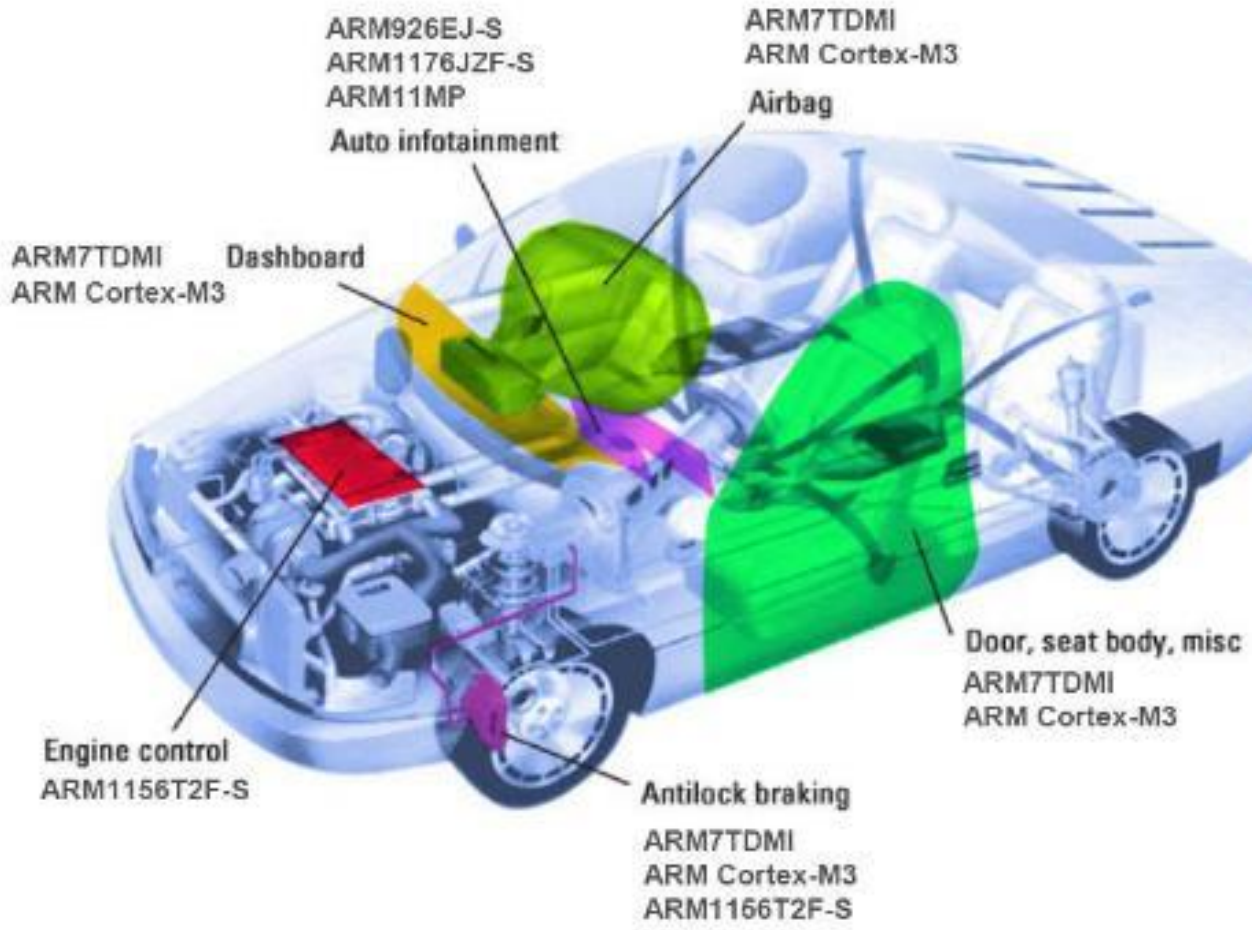
  A 16-bit THUMB instruction set.

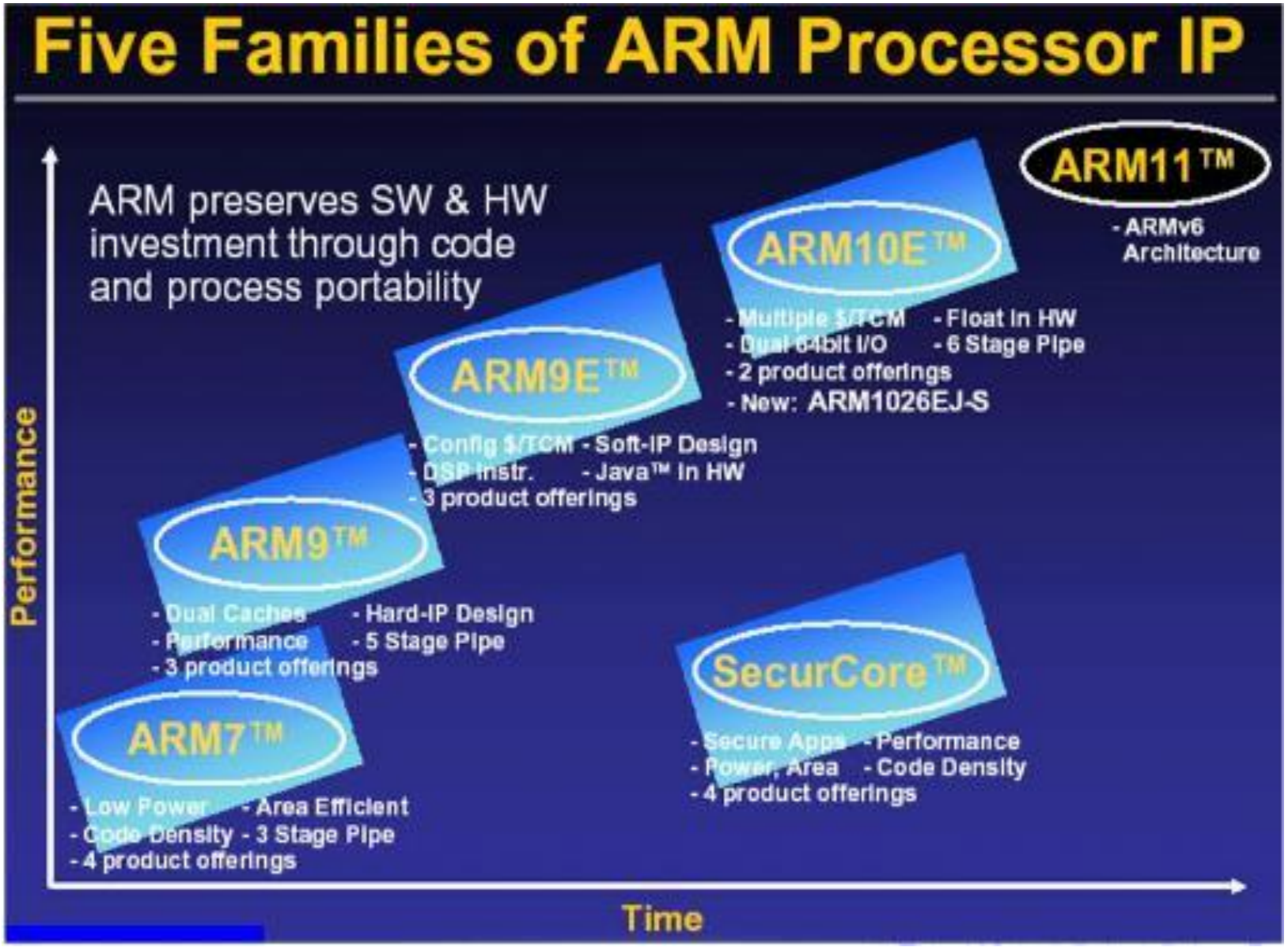
ARM7TDMI-S

T : supports both ARM (32-bit) and Thumb (16-bit) instruction sets.
D : Contains Debug extensions(mechanism by which normal operation of the processor can be suspended for debug).
M : Enhanced (relative to earlier ARM cores) 32x8 Multiplier block. (reduces the number of cycles required for a multiplication of two registers)
I : EmbeddedICE macrocell (consists of on-chip logic to support debug operations.)
-S : synthesizable (ie. distributed as RTL rather than a hardened layout)

ARM Powered Products

ARM Products




Five Families of ARM Processor IP
ARM preserves SW & HW investment through code and process portability
ARM11™
- ARMv6 Architecture
ARM10E™
- Multiple $/TCM - Float in HW
- Dual 64bit I/O - 6 Stage Pipe
- 2 product offerings
- New: ARM1026EJ-S
ARM9E™
- Config $/TCM - Soft-IP Design
- DSP instr. - Java™ in HW
- 3 product offerings
ARM9™
- Dual Caches - Hard-IP Design
- Performance - 5 Stage Pipe
- 3 product offerings
SecurCore™
- Secure Apps - Performance
- Power, Area - Code Density
- 4 product offerings
ARM7™
- Low Power - Area Efficient
- Code Density - 3 Stage Pipe
- 4 product offerings
Performance
Time

Brief History of ARM Core

- 1985, First ARM (ARM1)
- 1995, ARM7TDMI (**Technology Driven Market Intelligence**)

  Most successful ARM core

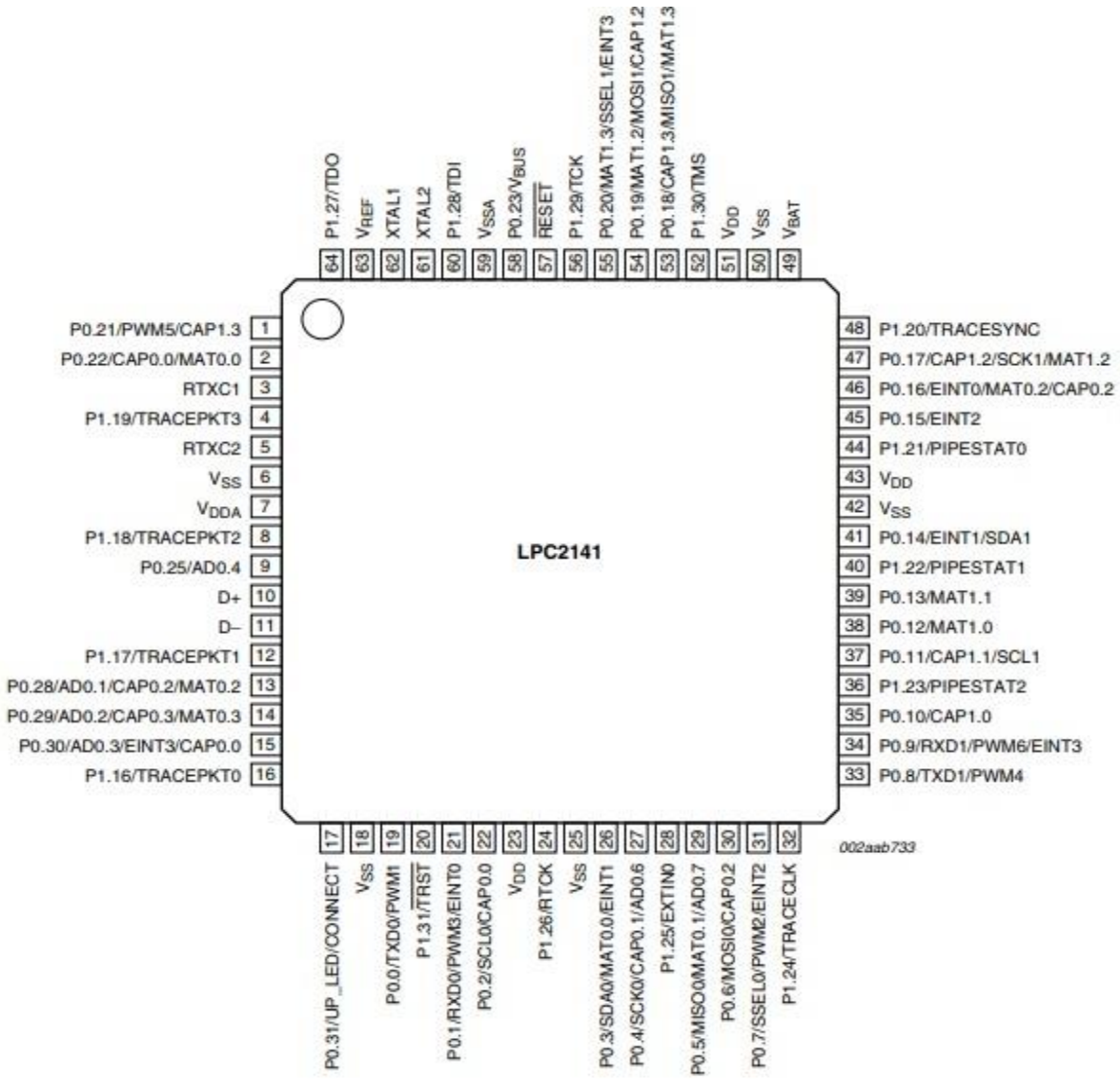
  3-stage pipeline, 120 Dhrystone MIPS
- 1997, ARM9

  5-stage pipeline (Instruction Fetch, Decode, Execute, Memory access, Register write back

  Harvard (I+D cache), MMU (OS's VM)
- 1999, ARM10
- 6-stage pipeline (Fetching the instruction from memory, Decode, Calculating EA, Fetch the operands from the given memory. Execute, Store the result in a proper place.

  VFP(Vector Float Point) (7-stage pipeline)
- 2003, ARM11 8-stage pipeline

LPC2148 Pin Configuration



Reduced Instruction Set Architecture (RISC)

The main idea behind this is to simplify hardware by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, a store command will store the data.

Complex Instruction Set Architecture (CISC)

The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it’s complex.

Comparison of CISC and RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

1.1 The RISC design philosophy

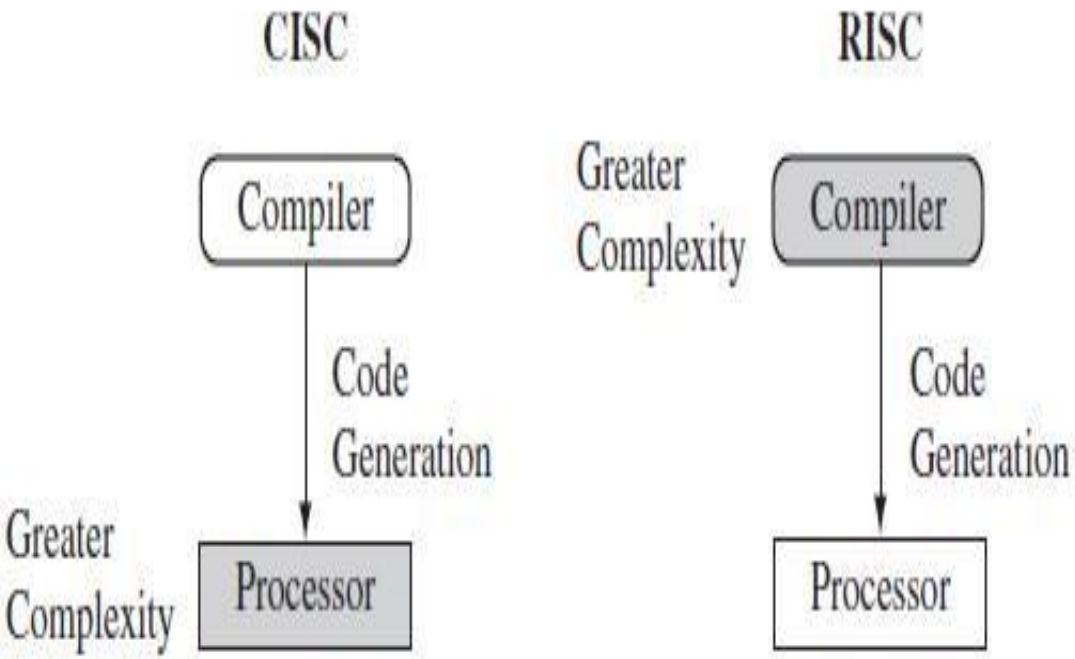
- The ARM core uses a RISC architecture.
- RISC aimed at

- simple and powerful instructions that execute within a single cycle at a high clock speed.

– Provides greater flexibility and intelligence in software rather than hardware.

- CISC relies more on the hardware for instruction Functionality.
- CISC instructions are more complicated.

CISC vs. RISC


CISC
Compiler
Code
Generation
Greater
Complexity
Processor
RISC
Greater
Complexity
Compiler
Code
Generation
Processor

RISC Design Rules

- Four major design rules: 1. Instructions
  2. Pipelines
  3. Registers
  4. Load-store architecture

1. **Instructions**
   - Reduced number of instruction classes.
   - Each class provides simple operations that can each execute in a single cycle.
   - Complicated instructions are synthesized by combining simpler instructions. (for example, a divide operation)
   - Each instruction has fixed length (to fetch future instructions before decoding the current instruction.)

Note: In CISC –variation in length and take no. of cycles for execution.

RISC Design Rules continued…

2. **Pipelines:**

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

– Ideally the pipeline advances by one step on each cycle for maximum throughput.

– Instructions can be decoded in one pipeline stage.

Note: In CISC – An instruction is executed by a miniprogram called microcode.

RISC Design Rules continued…

3. **Registers**

– RISC machines have a large general-purpose register set.

– Any register can contain either data or an address.

– Registers act as the fast local memory store for all data processing operations.

Note: In CISC – dedicated registers for specific purposes.

RISC Design Rules continued…

4. **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.
– As memory accesses are costly, they perform operations only on register data.

Note: In CISC – data processing operations can act on memory directly.

The ARM Design Philosophy

• Physical features of ARM processor design:

1. **Power:** The ARM processor has been specifically designed to be small to reduce power consumption and extend battery operation that is essential for applications such as mobile phones and personal digital assistants (PDAs).
2. **High Code Density:** Major requirement since embedded systems have limited memory due to cost and/or physical size restrictions. It is useful for applications that have limited on-board memory, such as mobile phones and mass storage devices.
3. **Price sensitive:** Use slow and low-cost memory devices. For high-volume applications like digital cameras, every cent has to be accounted for in the design. The ability to use low-cost memory devices produces substantial savings.
4. **Area:** Another important requirement is to reduce the area of the die taken up by the embedded processor. The smaller the area used by the embedded processor, the more available space for specialized peripherals. It also reduces the cost of the design and manufacturing.
5. **Use of hardware debug technology** : Incorporated within the processor.

Instruction Set for Embedded Systems:

The ARM instruction set differs from the pure RISC definition in following ways.

- **Variable cycle execution** for certain instructions—Not every ARM instruction executes in a single cycle.
- Example: **load-store-multiple** instructions vary in the number of execution cycles depending upon the number of registers being transferred. The transfer can occur on sequential memory addresses, which increases performance. **Code density** is also improved since multiple register transfers are common.
- **Inline barrel shifter leading to more complex instructions**—The inline barrel shifter is a hardware component that preprocesses one of the input registers before it is used by an instruction. This expands the capability of many instructions to improve core performance and code density.

(used to shift and rotate n-bits , typically within a single clock cycle)

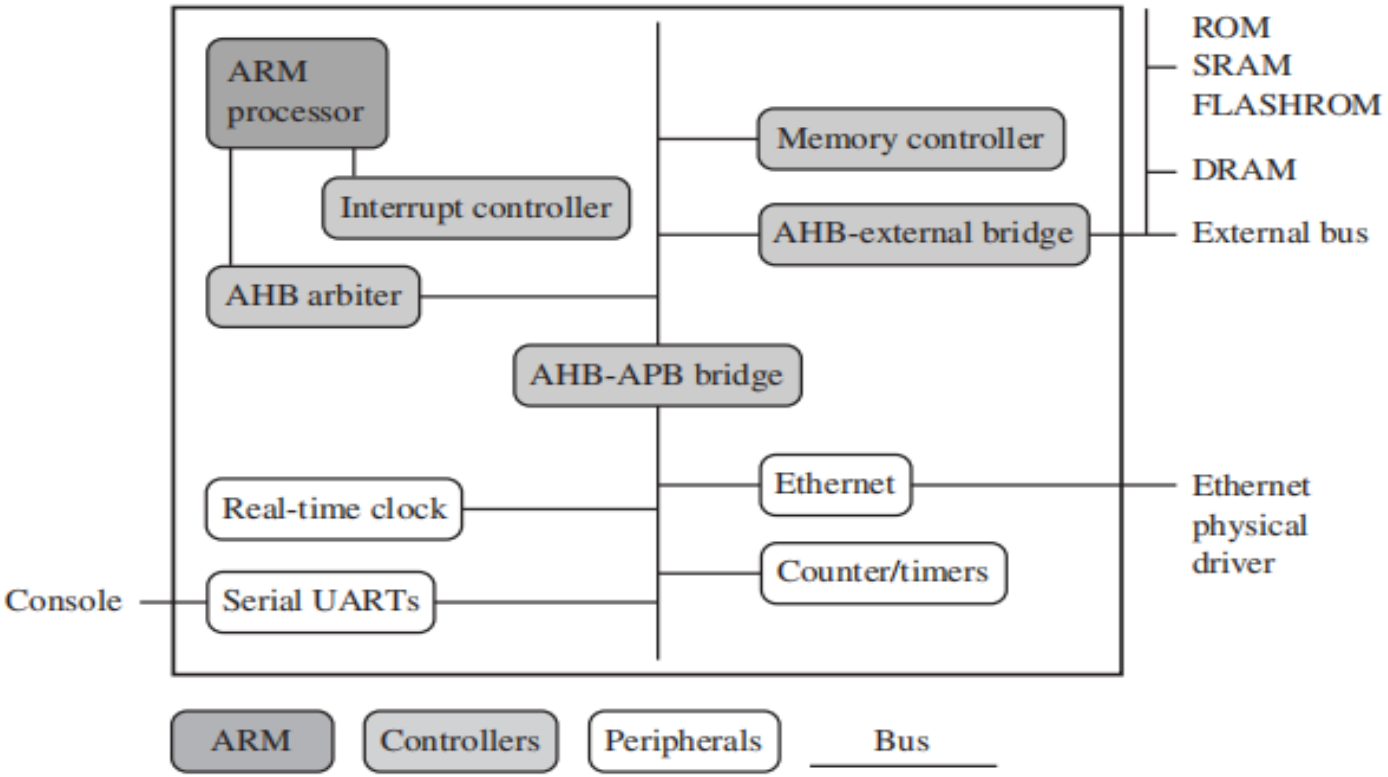
- **Thumb 16-bit instruction set**—ARM enhanced the processor core by adding a second 16-bit instruction set called Thumb that permits the ARM core to execute either 16- or 32-bit instructions. The 16-bit instructions improve code density by about 30% over 32-bit fixed-length instructions.

- **Conditional execution**—An instruction is only executed when a specific condition has been satisfied. This feature improves performance and code density by reducing branch instructions.

- **Enhanced instructions**—The enhanced digital signal processor (DSP) instructions were added to the standard ARM instruction set to support fast 16 × 16-bit multiplier operations and saturation. These instructions allow a faster-performing ARM processor in some cases to replace the traditional combinations of a processor plus a DSP.

- **Embedded System Hardware:** Embedded systems can control many different devices, from small sensors found on a production line, to the real-time control systems used on a NASA space probe. All these devices use a combination of software and hardware components. Each component is chosen for efficiency and, if applicable, is designed for future extension and expansion.

An ARM-based embedded device : A microcontroller.



Each box represents a feature or function. The lines connecting the boxes are the buses carrying data.

Embedded System Hardware : There are four main hardware components:

- **Core:** An ARM processor comprises a core (the execution engine that processes instructions and manipulates data) plus the surrounding components that interface it with a bus. These components can include memory management and caches.
- **Controllers** coordinate important functional blocks of the system. Two commonly found controllers are interrupt and memory controllers.
- **The peripherals** provide all the input-output capability external to the chip and are responsible for the uniqueness of the embedded device.
- **A bus** is used to communicate between different parts of the device.

ARM Bus Technology

- **Peripheral Component Interconnect (PCI) bus :** Most common PC bus technology, which connects devices such as video cards and hard disk controllers to the x86 processor bus. This type of technology is external or off-chip and is built into the motherboard of a PC.
- In contrast, **embedded devices** use an on-chip bus that is internal to the chip and that allows different peripheral devices to be interconnected with an ARM core.

Two different classes of devices attached to the bus.
The ARM processor core is a bus master—a logical device capable of initiating a data transfer with another device across the same bus.
Peripherals tend to be bus slaves—logical devices capable only of responding to a transfer request from a bus master device.

Advanced Microcontroller Bus Architecture (AMBA) Bus Protocol: A bus has two architecture levels.

- Physical level that covers the electrical characteristics and bus width (16, 32, or 64 bits).
- Second level deals with protocol—the logical rules that govern the communication between the processor and a peripheral.

(AMBA) is widely adopted as the on-chip bus architecture.

- **ARM System Bus (ASB) (**for external peripherals and requires a bridge).
- **ARM Peripheral Bus (APB)(**for the slower peripherals).
- **ARM High Performance Bus (AHB)(**for high performance peripherals)

Note: Using AMBA, peripheral designers can reuse the same design on multiple projects.

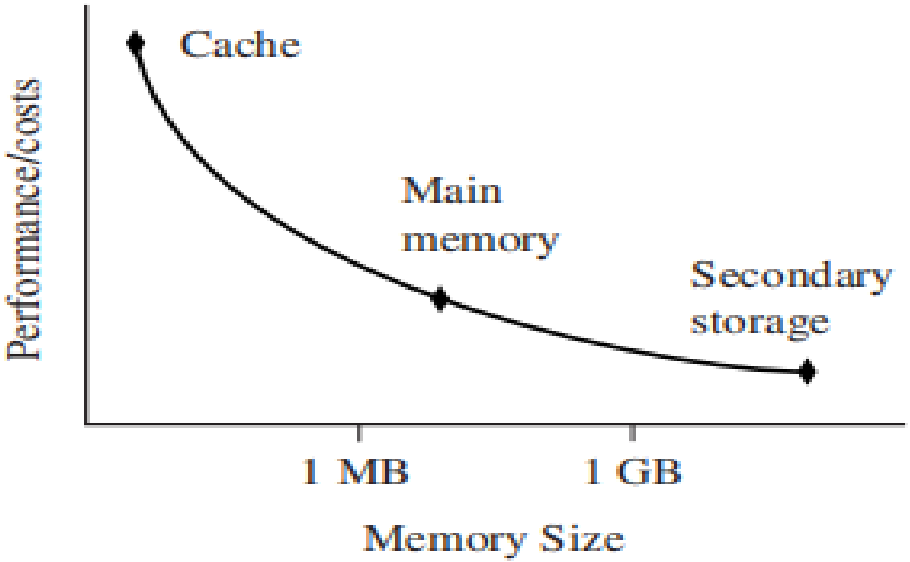
Memory

An embedded system has to have some form of memory to store and execute code.

Memory characteristics are hierarchy, width, and type. To decide, compare price, performance, and power consumption.

Hierarchy

Figure 1.2 shows a device that supports external off-chip memory. Internal to the processor there is an option of a cache (not shown in Figure 1.2) to improve memory performance.



Width

Memory width: The number of bits the memory returns on each access—typically 8, 16, 32, or 64 bits. It has a direct effect on the overall performance and cost ratio.

If you have an uncached system using 32-bit ARM instructions and 16-bit-wide memory chips, then the processor will have to make two memory fetches per instruction. Each fetch requires two 16-bit loads. This obviously reduces the system performance, but the benefit is that **16-bit memory is less expensive.**

In contrast, if the core executes 16-bit Thumb instructions, it will achieve better performance with a 16-bit memory. Using Thumb instructions with 16-bit-wide memory devices provides both **improved performance and reduced cost.**

Types of memory

- Read-only memory (ROM) is the least flexible of all memory types because it cannot be reprogrammed. Many devices also use a ROM to hold **boot code**.
- Dynamic random access memory (DRAM) is the most commonly used RAM for devices. It has the **lowest cost per megabyte** compared with other types of RAM. DRAM is dynamic— it needs to have its storage cells **refreshed . S**o you need to set up a DRAM controller before using the memory.
- Static random access memory (SRAM) is **faster** than the more traditional DRAM, but requires more **silicon area**. SRAM is static— the RAM **does not require refreshing**.
- Synchronous dynamic random access memory (SDRAM) is one of many subcategories of DRAM. It can run at **much higher clock speeds** than conventional memory. SDRAM synchronizes itself with the processor bus because it is clocked.

Peripherals : A peripheral device performs input and output functions for the chip by connecting to other devices or sensors that are off-chip. Each peripheral device usually performs a single function and may reside on-chip.

Examples: Peripherals range from a simple serial communication device to a more complex 802.11 wireless device.

- **Controllers are specialized peripherals** that implement higher levels of functionality within an embedded system.
- Two important types of controllers are **memory controllers and interrupt controllers.**

Memory Controllers

- Memory controllers **connect different types of memory to the processor bus.**
- On power-up a memory controller is configured in hardware to **allow certain memory devices to be active.**
- Allow the initialization code to be executed.

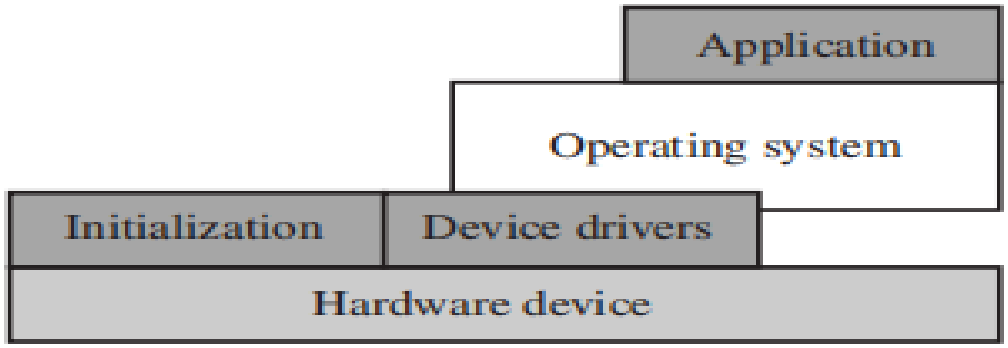
Interrupt Controllers

- An interrupt controller provides a programmable governing policy that allows software to determine which peripheral or device can interrupt the processor at any specific time by setting the appropriate bits in the interrupt controller registers.

Types of interrupt controllers: **standard interrupt controller and the vector interrupt controller (VIC).**

Embedded System Software:

- An embedded system needs software to drive it.
- 4 typical software components are required to control an embedded device. Each software component in the stack uses a higher level of abstraction to separate the code from the hardware device.
- The initialization code is the first code executed on the board and is specific to a particular target or group of targets. It sets up the minimum parts of the board before handing control over to the operating system.

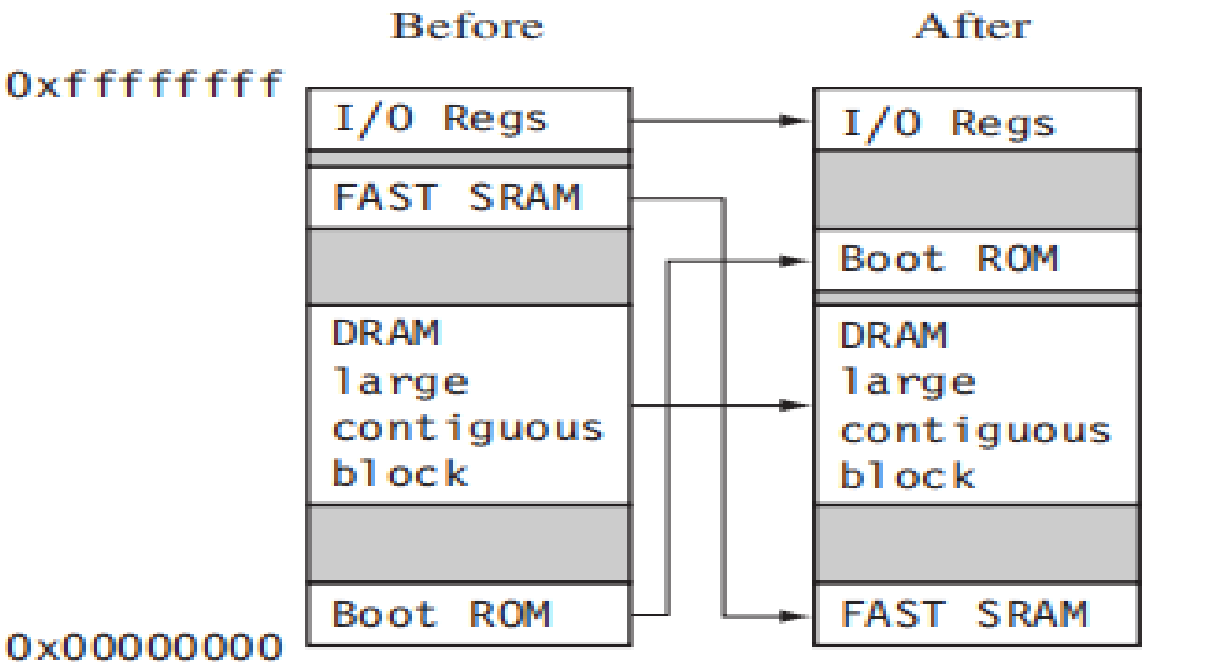


Software abstraction layers executing on hardware.

Initialization (Boot) Code : Handles a number of administrative tasks which are grouped in three phases:

Initial hardware configuration, diagnostics, and booting.

Example, the memory system normally requires reorganization of the memory map, as shown in below. It gets the Boot ROM at the top & places the RAM at address 0x00000000 because the Exception vector table can be in RAM.



Memory remapping

Diagnostics: Fault identification and isolation. These are embedded in the initialization code. Diagnostic code tests the system by exercising the hardware target to check if the target is in working order.

Booting : Involves loading an image and handing control over to that image. Loading an image involves anything from **copying an entire program including code and data into RAM, to just copying a data area containing volatile variables into RAM.** Once booted, the system hands over control by modifying the program counter to point into the start of the image.

Operating System :

ARM processors support over 50 operating systems. O S can be divided into two main categories:

Real-time operating systems (RTOSs) and

Platform operating systems.

- **RTOSs p**rovide **guaranteed response times** to events.

Hard real-time application requires a guaranteed response.

Soft real-time application requires a good response time,

Systems running an RTOS generally do not have secondary storage.

Example: **FreeRTOS** (Amazon), **Keil RTX** (ARM)

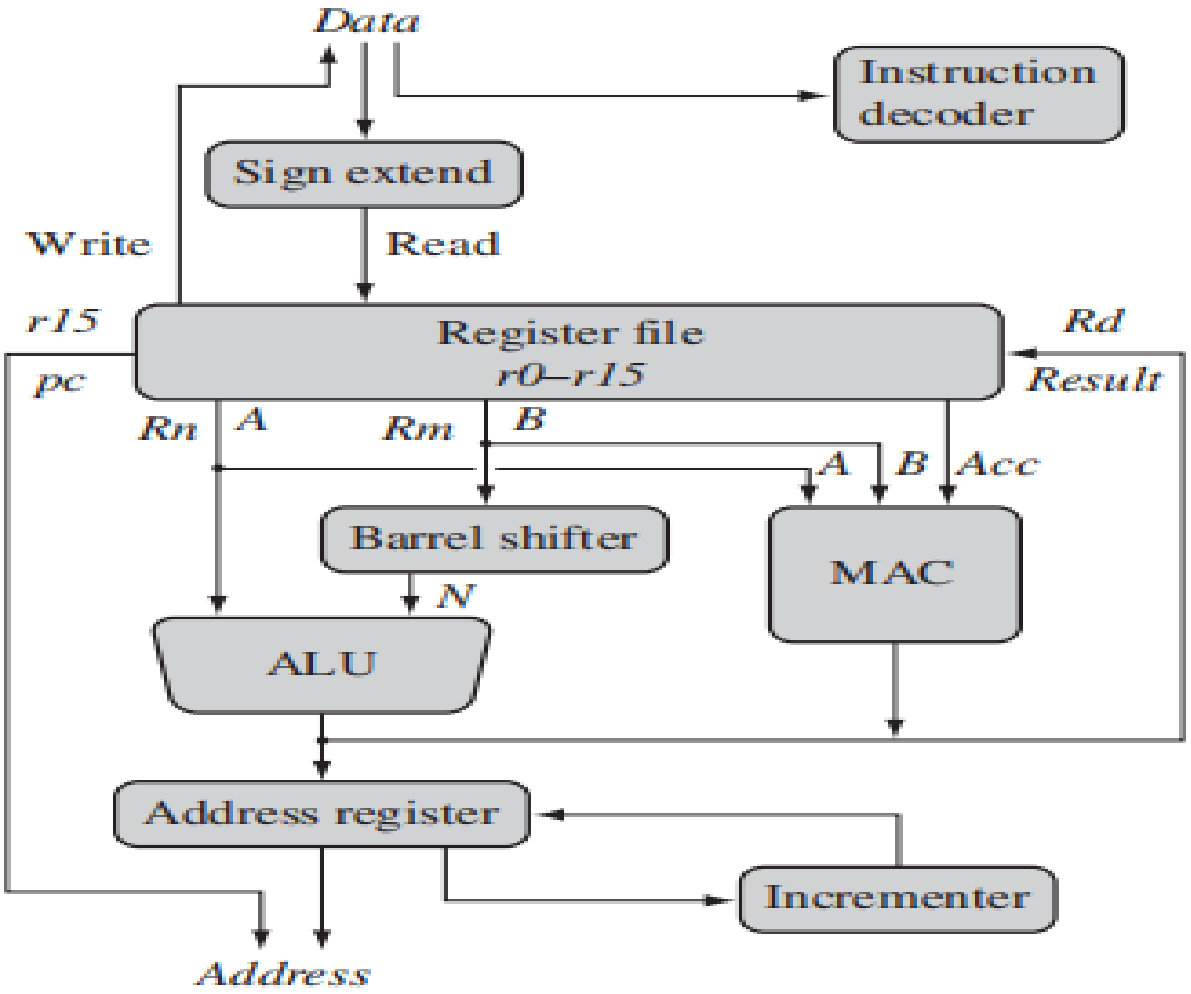
- **Platform o.ss require a memory management unit** to manage large, non-real-time applications and tend to have secondary storage.
- Example: Linux operating system**,** macOS, ios

Applications:

- Networking, automotive, mobile and consumer devices, mass storage, and imaging.
- Home gateways, DSL modems for high-speed Internet communication, and 802.11 wireless communication.
- ARM processors are also found in mass storage devices such as hard drives and imaging products such as inkjet printers—applications that are cost sensitive and high volume.

ARM Processor Fundamentals: ARM core dataflow model

An ARM core is a functional unit connected by data buses, as shown below, where, **arrows represent the flow of data, the lines represent the buses, and the boxes represent either an operation unit or a storage area.**



Abstract components of an ARM core:

- Data enters the processor core through the Data bus. The data may be an instruction to execute or a data item.
- Data items and instructions share the same bus (Von Neumann impl). In contrast, Harvard implementations of the ARM use two different buses.
- 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. Load instructions copy data from **memory to registers** in the core, and conversely the store instructions copy data from **registers to memory**.
- **Note:** 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.
- ARM core is a 32-bit processor, most instructions treat the registers as holding signed or unsigned 32-bit values. The sign extend hardware converts signed 8-bit and 16-bit numbers to 32-bit values as they are read from memory and placed in a register.
- 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.

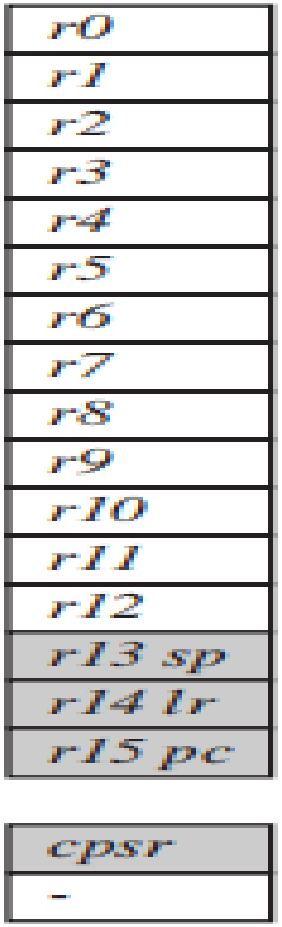
After passing through the functional units, the result in Rd is written back to the register file using the Result bus. For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location. 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 shows the active registers available in user mode—a protected mode normally used when executing applications. The processor can operate in seven different modes.

Registers available in user mode



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 tor15.

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.

In above Figure the shaded registers identify the assigned special-purpose registers:

■ Registerr13 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.

■ Registerr15 is the program counter (pc) and contains the address of the next instruction to be fetched by the processor.

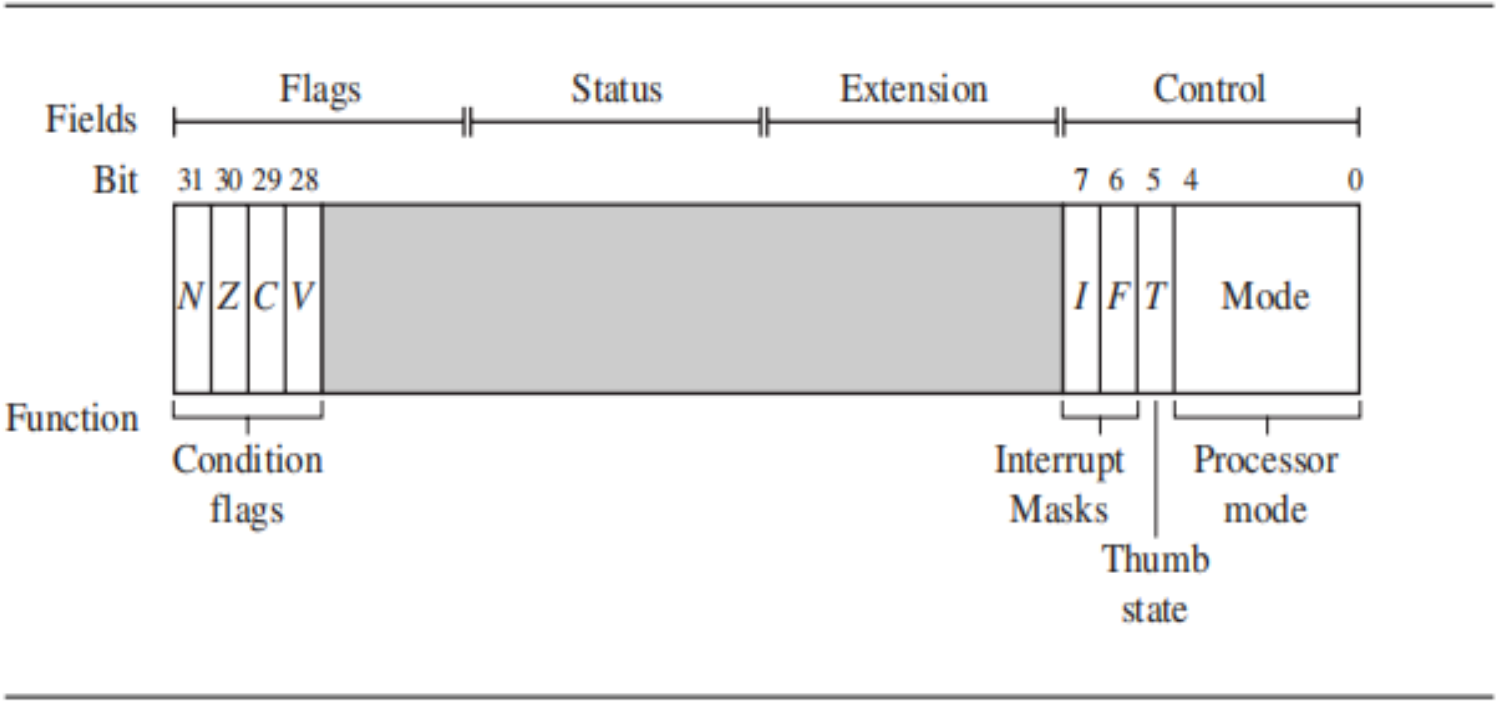
Depending upon the context, registers r13 and r14 can also be used as general-purpose registers, which can be particularly useful since these registers are banked during a processor mode change. However, it is dangerous to use r13 as a general register when the processor is running any form of operating system because operating systems often assume that r13 always points to a valid stack frame.

In ARM state the registers r0 to r13 are orthogonal—any instruction that you can apply to r0 you can equally well apply to any of the other registers. However, there are instructions that treat r14 and r15 in a special way.

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.
Figure above shows the basic layout of a generic program status register.

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.

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.
Privileged mode : It allows full read-write access to the cpsr.
Nonprivileged mode : It only allows read access to the control field in the cpsr but still allows read-write access to the condition flags.

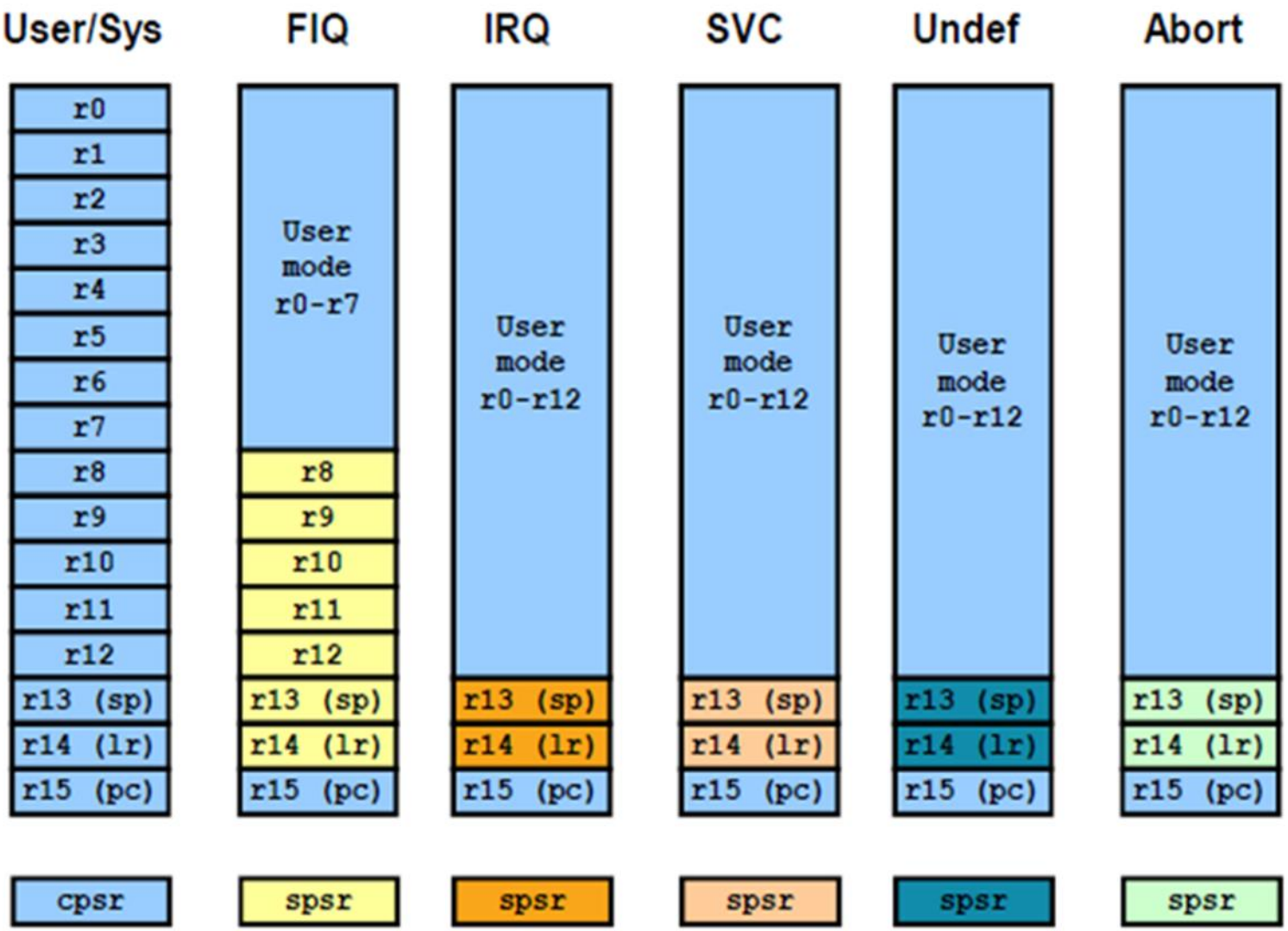
Seven processor modes: 6 privileged modes (abort, fast interrupt request, interrupt request, supervisor, system, and undefined) and one nonprivileged mode (user).

- **Abort mode:** 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 o.s 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 :

- There are 37 registers in the register file. Of those, 20 registers are hidden from a program at different times. These registers are called **banked registers** and are identified by the shading in the diagram. They are available only when the processor is in a particular mode;
- Example: **Abort mode has banked registers r13_abt, r14_abt and spsr_abt.**
- 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.

- Example: when the processor is in the **Interrupt request mode(IRQ),** the instructions you execute still access registers named r13 and r14. However, these registers are the banked registers r13_irq and r14_irq.
- The user mode registers r13_usr and r14_usr are not affected by the instruction referencing these registers. A program still has normal access to the other registers r0 to r12.
- 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 n 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.
- **Note** : Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location.



Note: System mode uses the User mode register set

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 :

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

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*

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.

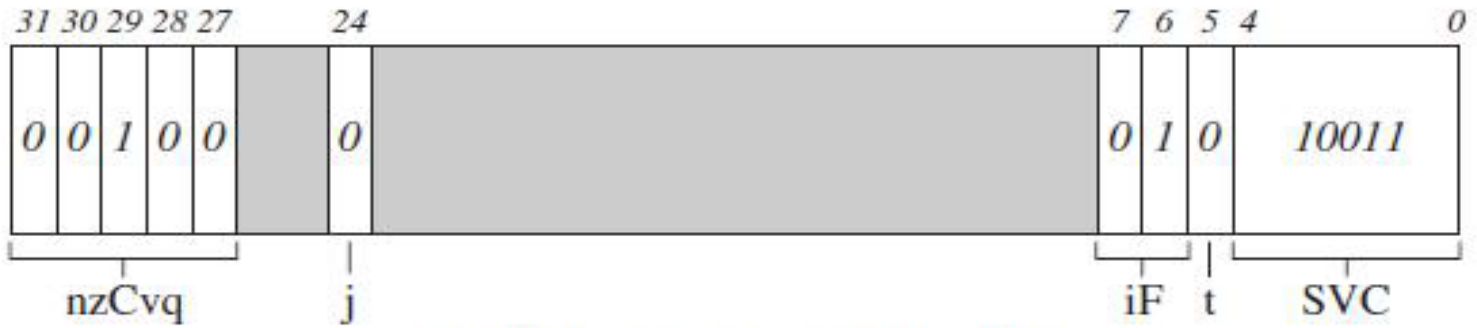
• Two interrupt request levels available on the

– *interrupt request* (IRQ) =>(I bit=1)

– *fast interrupt request* (FIQ) =>(F bit=1)

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



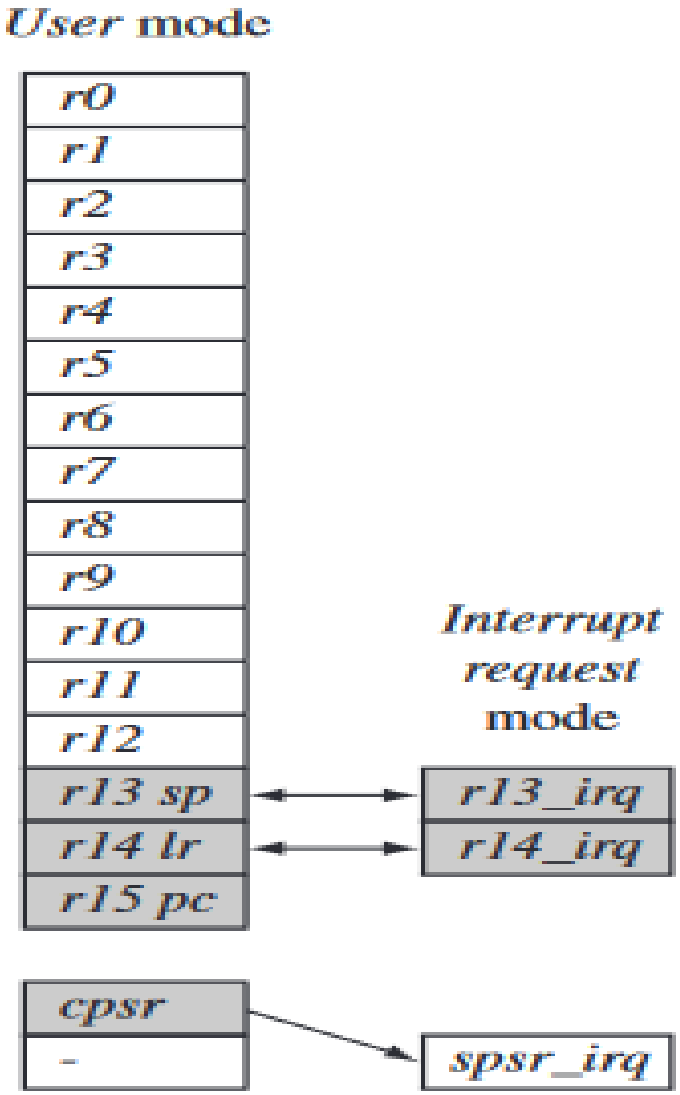
Example: *cpsr = nzCvqjiFt_SVC.*

Conditional Execution :

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

Changing mode on an exception



- The core changing from **user mode to interrupt request mode**, which happens when an interrupt request occurs due to an external device raising an interrupt to the processor core.
- This change causes user registers r13 and r14 to be banked. The user registers are replaced with registers r13_irq and r14_irq, respectively. Note r14_irq contains the return address and r13_irq contains the stack pointer for interrupt request mode.
- A new register appearing in interrupt request mode: the **Saved program status register (spsr),** which stores the previous mode's cpsr. You can see in the diagram the cpsr being copied into spsr_irq. To return back to user mode, a special return instruction is used that instructs the core to restore the original cpsr from the spsr_irq and bank in the user registers r13 and r14.
- **Note :** The spsr can only be modified and read in a privileged mode. There is no spsr available in user mode

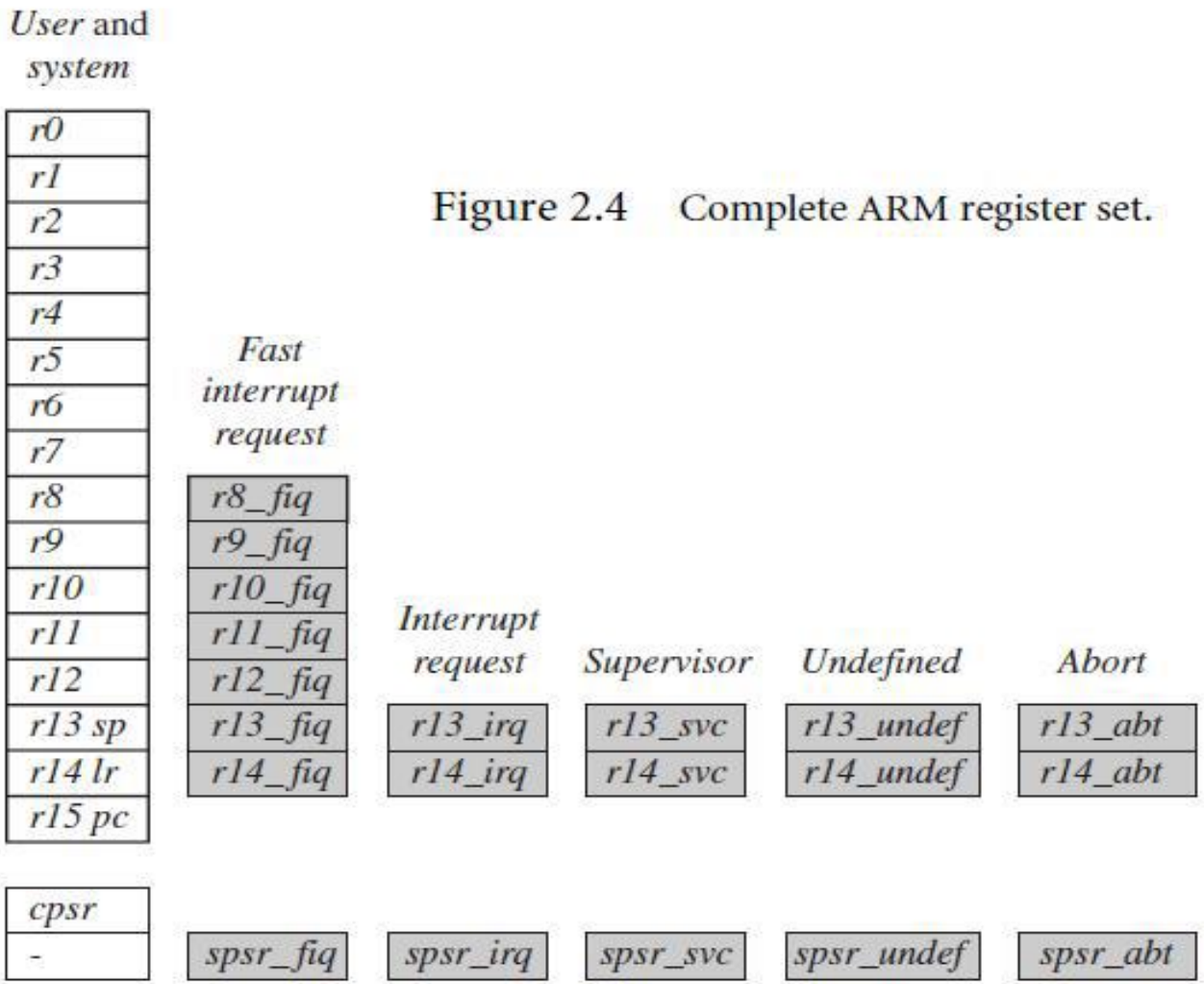


Figure 2.4 Complete ARM register set.

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

Figure above shows a 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.

Example :

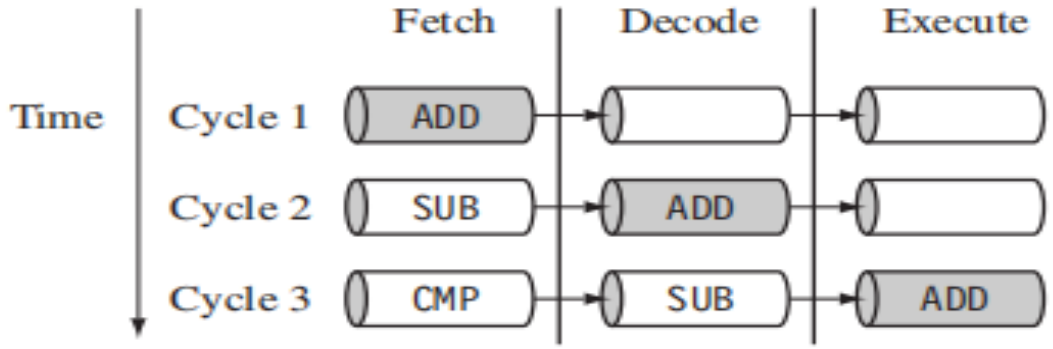


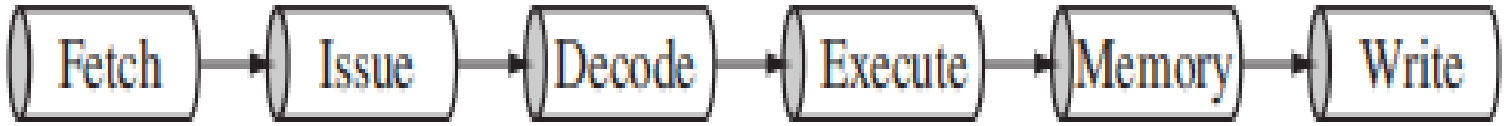
Figure shows a sequence of three instructions being fetched, decoded, and executed by the processor. Each instruction takes a single cycle to complete after the pipeline is filled.

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.

ARM9 five-stage pipeline

ARM10 six-stage pipeline



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.

. 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.

Core Extensions : These are standard components placed next to the ARM core. They improve performance, manage resources, and provide extra functionality and are designed to provide flexibility in handling particular applications. Each ARM family has different extensions available.

There are three hardware extensions ARM wraps around the core: cache and tightly coupled memory, memory management, and the coprocessor interface.

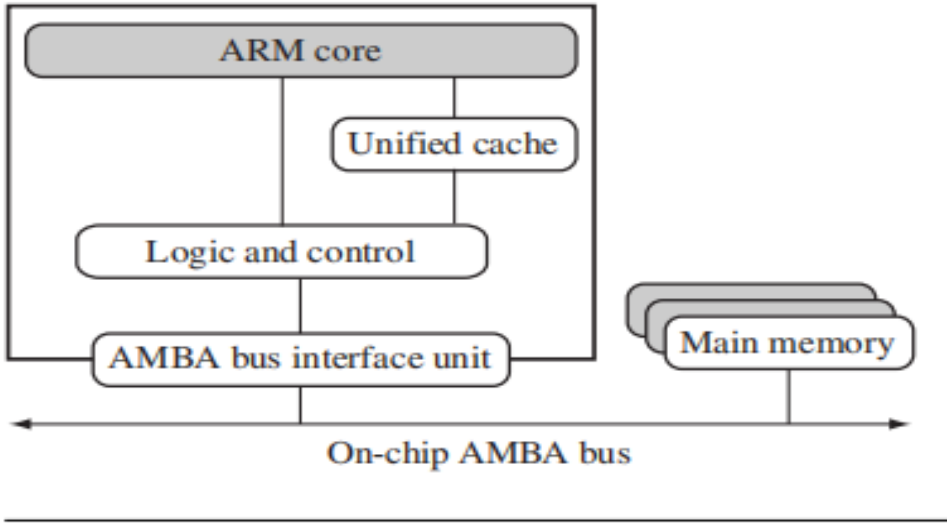
Cache and Tightly Coupled Memory:

The cache is a block of fast memory placed between main memory and the core. It allows for more efficient fetches from some memory types. With a cache the processor core can run for the majority of the time without having to wait for data from slow external memory.

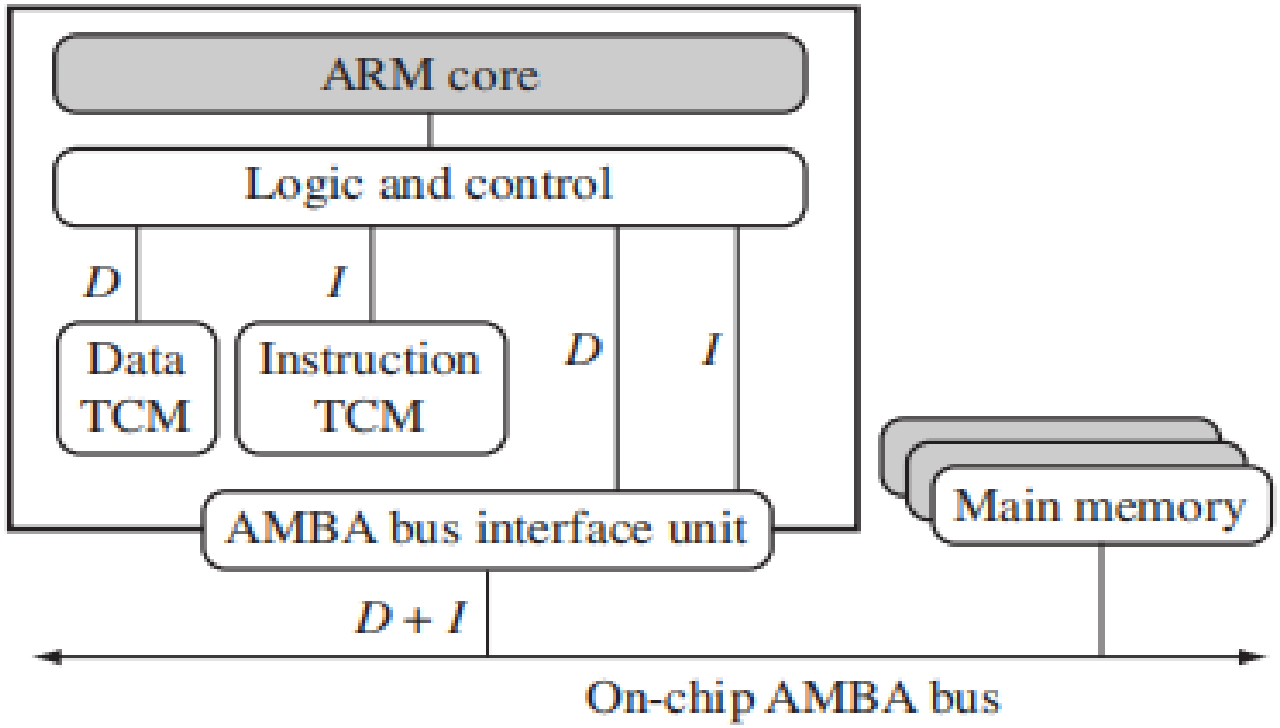
Most ARM-based embedded systems use a single-level cache internal to the processor. Of course, many small embedded systems do not require the performance gains that a cache brings.

ARM has two forms of cache. The first is found attached to the Von Neumann–style cores. It combines both data and instruction into a single unified cache, as shown in Figure 2.13. For simplicity, we have called the glue logic that connects the memory system to the AMBA bus logic and control.

A simplified Von Neumann architecture with cache.



A simplified Harvard architecture with TCMs.

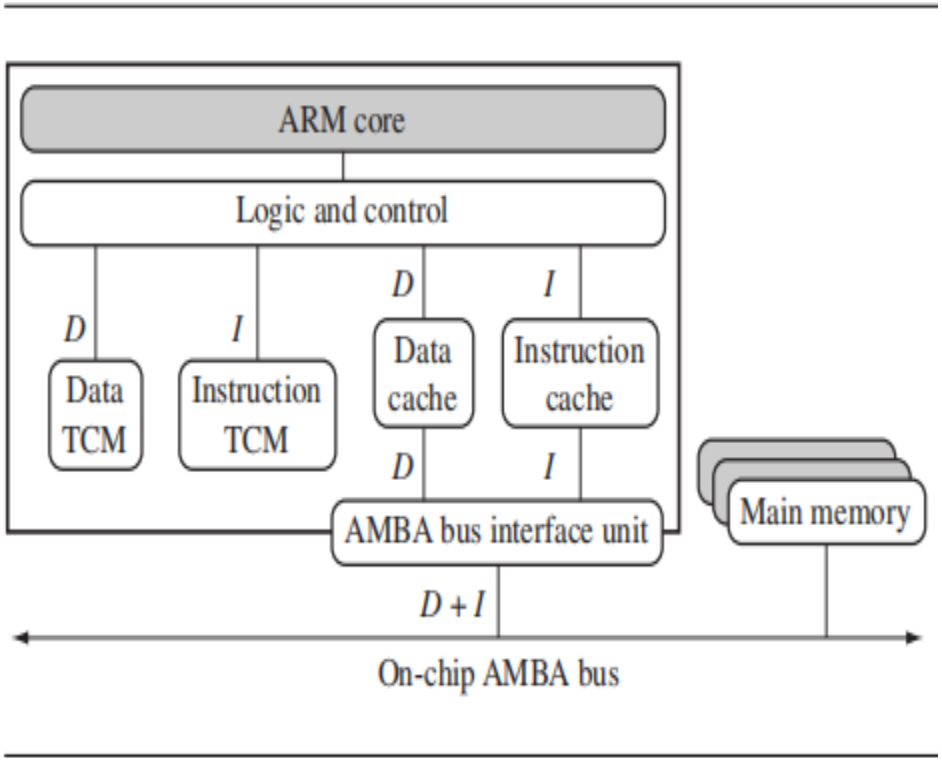


By contrast, the second form, attached to the Harvard-style cores, has separate caches for data and instruction.

A cache provides an overall increase in performance but at the expense of predictable execution. But for real-time systems it is paramount that code execution is deterministic—

the time taken for loading and storing instructions or data must be predictable. This is achieved using a form of memory called tightly coupled memory (TCM). TCM is fast SRAM located close to the core and guarantees the clock cycles required to fetch instructions or data—critical for real-time algorithms requiring deterministic behavior. TCMs appear as memory in the address map and can be accessed as fast memory.

Memory Management : ARM cores have three different types of memory management hardware—no extensions providing no protection, a memory protection unit (MPU) providing limited protection, and a memory management unit (MMU) providing full protection: Nonprotected memory is fixed and provides very little flexibility. It is normally used for small, simple embedded systems that require no protection from rogue applications



- MPUs employ a simple system that uses a limited number of memory regions. These regions are controlled with a set of special coprocessor registers, and each region is defined with specific access permissions. This type of memory management is used for systems that require memory protection but don’t have a complex memory map.
- MMUs are the most comprehensive memory management hardware available on the ARM. The MMU uses a set of translation tables to provide fine-grained control over memory. These tables are stored in main memory and provide a virtual-to-physical address map as well as access permissions. MMUs are designed for more sophisticated platform operating systems that support multitasking.

Coprocessors: Coprocessors can be attached to the ARM processor. A coprocessor extends the processing features of a core by extending the instruction set or by providing configuration registers. More than one coprocessor can be added to the ARM core via the coprocessor interface.

The coprocessor can be accessed through a group of dedicated ARM instructions that provide a load-store type interface. C