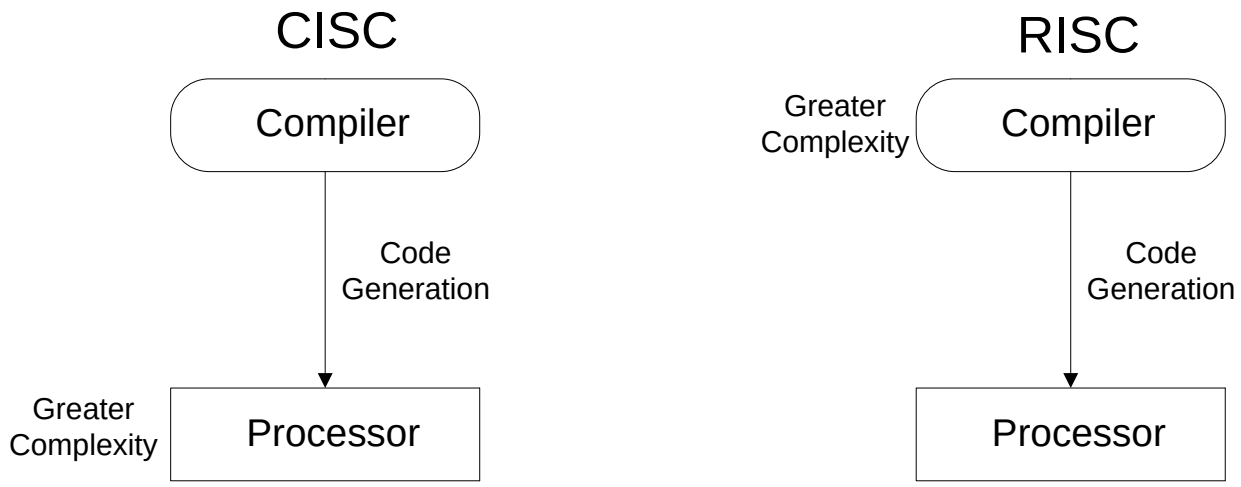
Q1. Explain briefly the RISC design philosophy.

Answer:
RISC is a design philosophy aimed at delivering simple but powerful instructions that execute within a single cycle at a high clock speed. The RISC philosophy concentrates on reducing the complexity of instructions performed by the hardware. The RISC philosophy provides greater flexibility and intelligence in software rather than hardware. RISC design places greater functionality to the compiler rather than the hardware.
The RISC philosophy is implemented with four major design rules:
Instructions: RISC has a reduced number of instruction classes. These classes provide simple operations so that each is executed in a single cycle. The compiler synthesizes complicated operations by combining several simple instructions. Each instruction is a fixed length to allow the pipeline to fetch future instructions before decoding the current instruction.
Pipeline: The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines. The pipeline advances by one step on each cycle for maximum throughput.
Register: 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.
Load-store architecture: The processor operates on the data held in registers. Separate load and store instructions transfer data between the register bank and external memory.



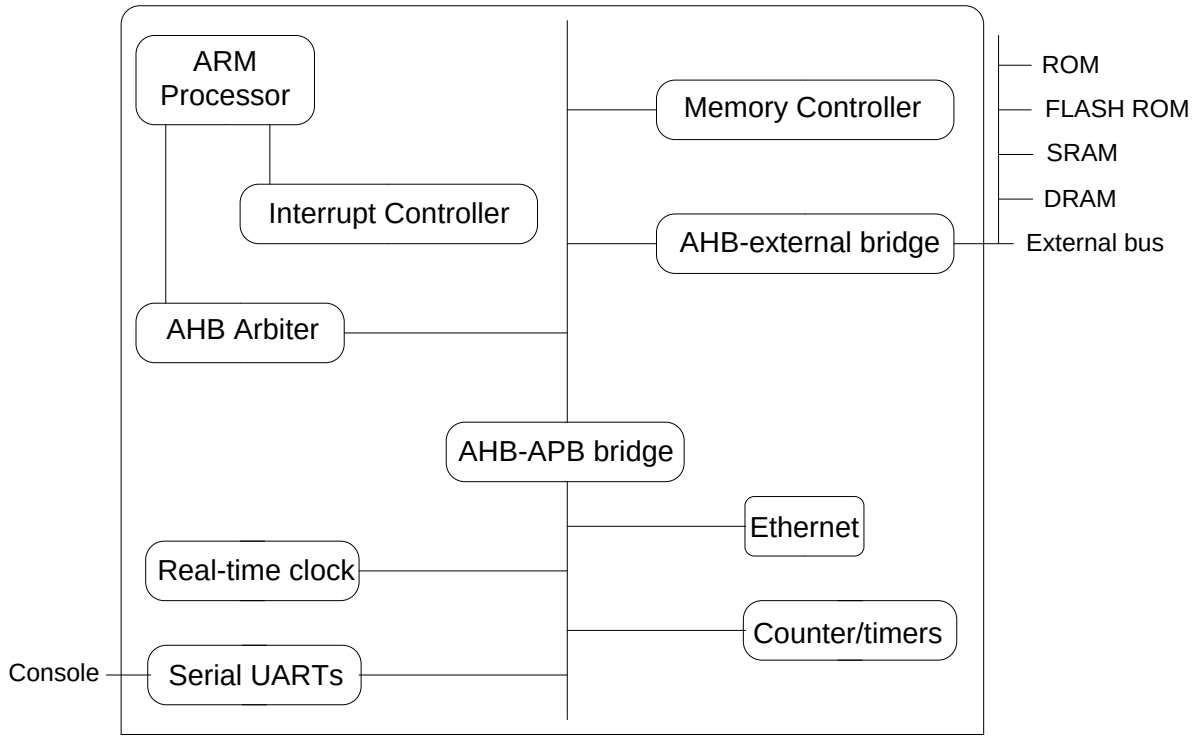
Q2, Explain briefly the ARM design philosophy.

Answer:
Following are the ARM design philosophy:
(i) The ARM processor has been specially designed to be small to reduce power consumption and extend battery operation- essentially for application such as mobile phones and personal digital assistants.

(ii) High code density is major requirement since embedded systems have limited memory due to cost and physical size restrictions. High code density is useful for applications that have limited on-board memory, such as mobile phones.
(iii) Embedded systems are price sensitive and use low cost memory devices. The ability to low-cost memory devices produces essential savings.
(iv) Another requirement is to reduce the area of the die taken up by the embedded processor. For a single-chip solution, the smaller the area used by the embedded processor, the more available space for specialized peripherals.
(v) Another requirement is the hardware debug technology within the processor so that software engineers can view what is happening while the processor executing code.

Q3. Explain briefly the ARM processor based embedded system hardware.

Answer: Figure shown below shows a typical embedded device based on ARM core. Each box represents a feature or function.



ARM processor based embedded system hardware can be separated into the following four main hardware components:

(i) The ARM processor: The ARM processor controls the embedded device. Different versions of the ARM processor are available to suits the desired operating characteristics.
(ii) Controllers: Controllers coordinate important blocks of the system. Two commonly found controllers are memory controller and interrupt controller.

Memory controller: 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. These memory devices allow the initialization code to be executed. Some memory devices must be set up by software, for example, DRAM.
Interrupt controller: 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. There are two types of interrupt controller available for ARM processor. These are the standard interrupt controller and vector interrupt controller.
The standard interrupt controller sends an interrupt signal to the processor core when an external device requests servicing. It can be programmed to ignore or mask an individual device or set of devices. The interrupt handler determines which device requires servicing by reading a device bitmap register in the interrupt controller.
Vector interrupt controller (VIC) prioritized interrupt and simplifies the determination of which device causes the interrupt. After associating a priority and a handler address with each interrupt, the VIC asserts an interrupt signal to the core if the priority of a new interrupt is higher than the currently executing interrupt handler.
(iii) Peripherals: The peripherals provide all the input-output capability external to the chip and responsible for the uniqueness of the embedded device. Embedded systems that interact with the outside world need some form of peripheral device. A peripheral device performs input and output functions for the chip by connecting to other devices or sensors reside that are off-chip.All ARM peripherals are memory mapped.
Controllers are specialized peripherals that implement higher levels of functionality within the embedded system.
(iv) Bus: A bus is used to communicate between different parts of the device. 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. The ARM processor core is bus master-a logical device capable of initiating a data transfer with another device across the same bus. Peripheral tends to be bus slave- logical device capable only of responding to a bus transfer request from a bus master device.
A bus has two architectural levels. The first is a physical level that covers the electrical characteristics and bus width. The second level deals with the protocol- the logical rules that govern the communication between the processor and a peripheral.

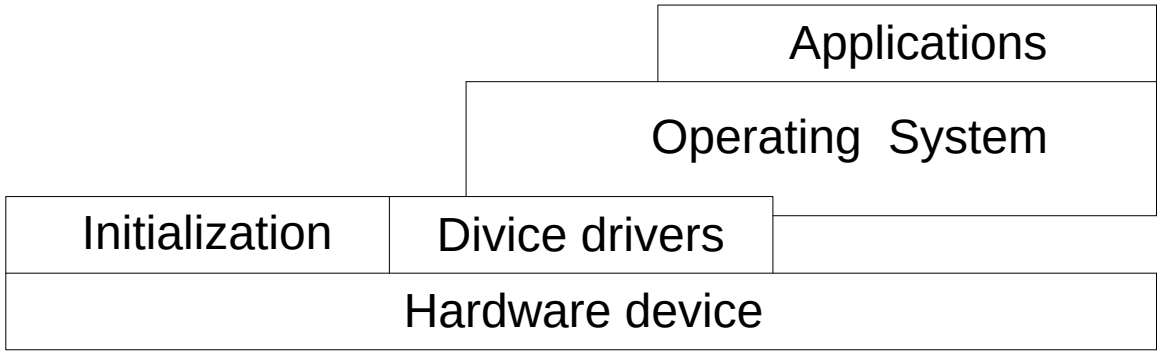
Advance Microcontroller Bus Architecture (AMBA) has been widely adopted as the on-chip architecture used for the ARM processors.

(v) Memory: An embedded system has to have some form of memory to store and execute code. There are three types of memory- cache, main memory and secondary memory. Cache is the fastest memory located near the ARM processor core. Cache is placed between main memory and the core. It is used to speed up data transfer between the processor and main memory. The main memory is large depending on the application. Secondary memory is the largest and slowest form of memory.

Q4. Explain briefly the ARM processor based embedded system software.

Answer:

An embedded system requires software to drive it. Figure below shows typical software components required to control an embedded device. Each software components in the stack uses a higher level of abstraction to separate the code from the hardware device.



(i) The initialization code: It 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 handling control over to the operating system. Initialization code (or boot code) takes the processor from the reset state to a state where the operating system can run. It usually configures the memory controller, processor caches and initializes some devices.

The initialization code handles a number of administrative tasks prior to handling control over to an operating system image. We can group these different tasks into three phases: initial hardware configuration, diagnostics and booting.

Initial hardware configuration involves setting up the target platform so it can boot an image.

Diagnostics are often embedded in the initialization code. Diagnostic code tests the system by exercising the hardware target to check if the target is in working order. The primary purpose of diagnostic code is fault identification and isolation.

Booting involves loading an image and handling control over the image. Loading an image involves anything from copying an entire program including code and data into RAM. Once booted, the system hands over control by modifying the program counter to point into the start of the image.

(ii) The operating system provides an infrastructure to control applications and manage hardware system resources. Many embedded systems do not require a full operating system but merely a simple task scheduler.

An operating system organizes the system resources: the peripherals, memory and processing time. With an operating system controlling these resources, they can be efficiently used by different applications running within the operating system environment.

ARM processors support over 50 operating systems. We can divide operating systems into two main categories: real time operating systems (RTOSs) and platform operating systems.

RTOSs provide guaranteed response times to events. Different operating systems have different amounts of control over the system response time. A hard real-time application requires a guaranteed response within specified response time. A soft real-time application requires a good response time, but the performance degrades more gracefully if the response time overruns. Systems running an RTOS generally do not have secondary storage.
Platform operating systems require a memory management unit to manage large, non-real time applications and tends to have secondary storage.

(iii) The device drivers are the third component that provides a consistent software interface to the peripherals on the hardware device.
(iv) Finally, an application performs one of the tasks required for a device. For example, a mobile phone might have diary application. There may be multiple applications running on the same device, controlled by the operating systems.
An embedded system can have one active application or several applications running simultaneously.
ARM processors are found in numerous market segments, including networking, mobile, consumer devices, mass storage and imaging.
The software components can run from ROM or RAM. ROM code that is fixed on the device is called firmware, for example the initialization code.

Q5. Explain briefly the active registers available in user mode.

Answer: Figure shown below shows the active registers available in user mode. All the registers shown are 32 bits in size.

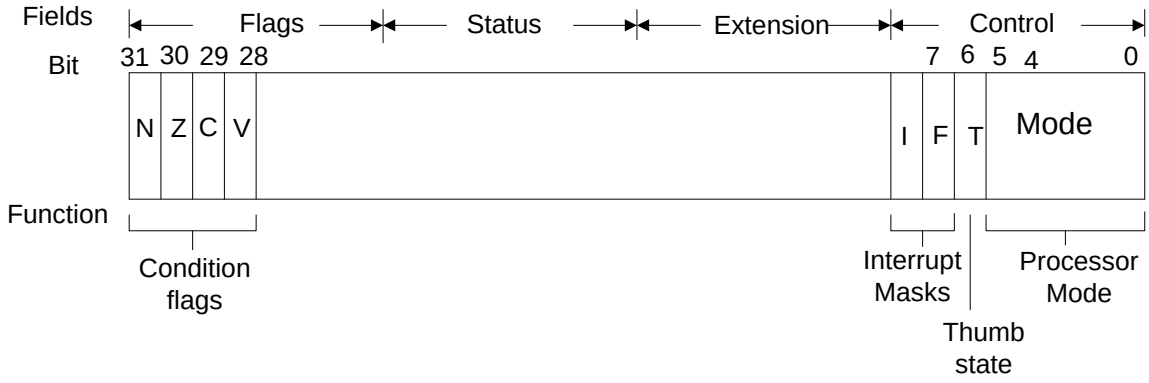
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15

cpsr
-

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: r13, r14 and r15.
Register r13: Register r13 is traditionally used as the stack pointer (sp) and stores the head of the stack in the current processor mode.
Register r14: Register r14 is called the link register(lr) and is where the core puts the return address whenever it calls a subroutine.
Register r15: Register r15 is the program counter (pc) and contains the address of the next instruction to be fetched by the processor.
In ARM state the registers r0 r13 are orthogonal-any instruction that can be applied to r0 equally can be applied to any 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: current program status register (cpsr) and saved program status register (spsr).
The register file contains all the registers available to a programmer. Registers visible to the programmer depend upon the current mode of the processor.

Q6. Explain briefly the current program status register (cprs).

Answer: Figure below 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 interrupts mask bits. The flag field contains the condition flags.

The following table gives the bit patterns that represent each of the processor modes in the cpsr.

Mode	Mode[4:0]
Abort	10111
Fast interrupt request	10001
Interrupt request	10010
Supervisor	10011
System	11111
Undefined	11011
User	10000

When cpsr bit 5 , T=1 , then the processor is in Thumb state. When T=0, the processor is in ARM state.

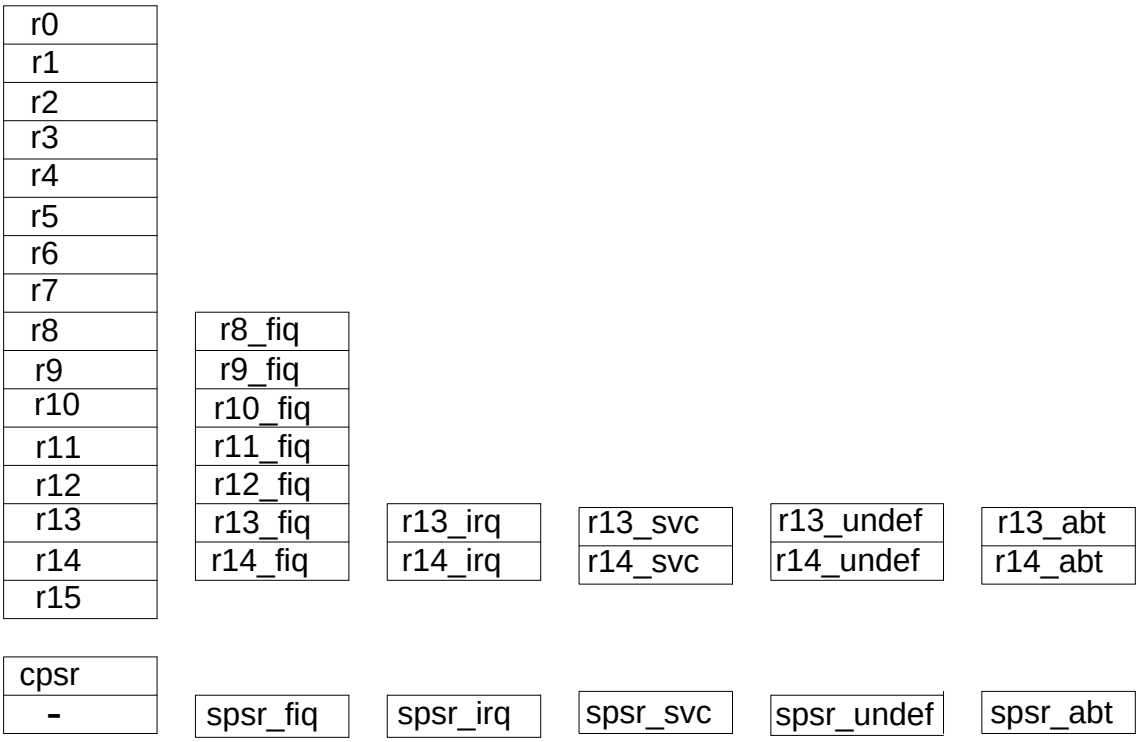
The cpsr has two internal mask bits,7 and 6 (I and F) which control the masking Interrupt request(IRQ) and Fast Interrupt Request(FIR).

The following table shows the conditional flags:

Flag	Flag Name	Set when
N	Negative	Bit 31 of the result is a binary 1
Z	Zero	The result is zero, frequently used to indicate equality
C	Carry	The result causes an unsigned carry
V	oVerflow	The result causes a signed overflow

Q7. Explain briefly the processor modes for ARM processor.
Answer: The processor mode determines which registers are active and the access rights to the cprs register itself. Each processor mode is either privileged or nonprivileged. A privileged mode allows read-write access to the cprs. A nonprivileged mode only allows read access to the control field in the cpsr but allows read-write access to the conditional flags.
There are seven processor modes : six privileged modes and one nonprivileged mode. The privilege modes are abort, fast interrupt request , interrupt request, supervisor, system and undefined. The nonprivileged mode is user.
The processor enter 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 program and applications.

Q8. Explain briefly the complete ARM register set.
Answer: Figure below shows all 37 registers in the register file. Of these, 20 registers are hidden from a program at different times. These registers are called banked registers. They are available only when the processor is in a particular mode, for example, 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.



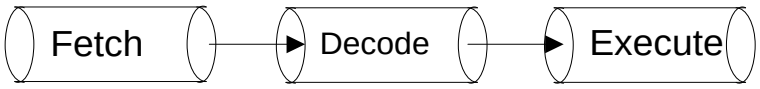
Every processor mode except user mode can change mode by writing directly to the mode bits of the cpsr. All processor modes except system mode have a set of associated banked registers that are subset of the main 16 registers. A banked register maps one-to-one onto a user mode register. If the processor mode is changed, a banked register from the new mode will replace an existing register.
For example, when the processor mode is in the interrupt request mode, the instructions you execute still excess 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 when the processor core is in privilege mode. The following exception and interrupts causes a mode change: reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort and undefined instructions. Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location.
Following figure illustrates the happening when an interrupt forces a mode change. The figure shows 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. r14_irq contains the return address and r13_irq contains the stack pointer for interrupt request mode.

Following table lists the various modes and associated binary patterns. The last column of the table gives the bit patterns that represent each of the processor modes in the cpsr.

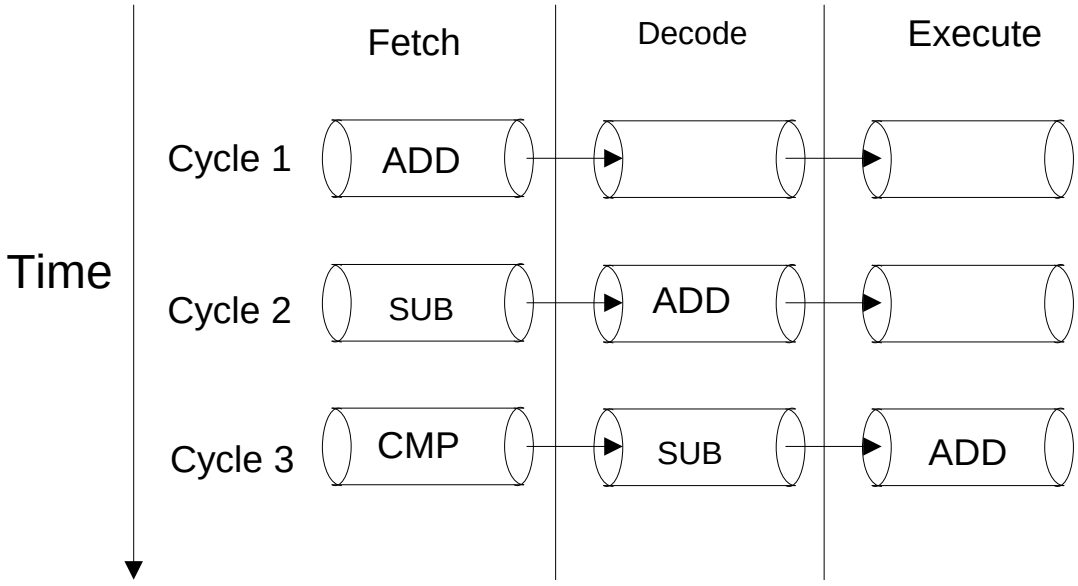
Mode	Abbreviation	Privilege	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	n0	10000

Q9. Explain briefly the pipeline for ARM processor.

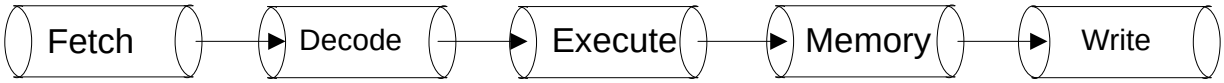
Answer:
Pipeline is the mechanism to speed up execution by fetching the next instruction while other instruction are being decoded and executed.
Figure below shows the ARM7 three-stage pipeline.



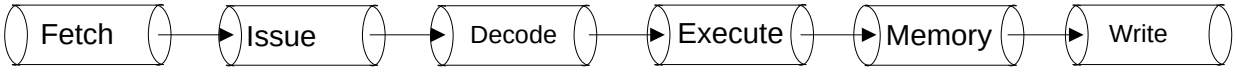
Fetch loads an instruction from memory. Decode identifies the instruction to be executed. Execute processes the instruction and writes the result back to a register.
Figure shown below illustrates the pipeline using a simple example. It shows a sequence of three instructions being fetched, decoded and executed by the processor. Each instruction takes a single cycle to complete after the pipeline is filled. In the first cycle, the core fetches the ADD instruction from the memory. In the second cycle, the core fetches the SUB instruction and decode the ADD instruction. In the third cycle, the core fetches CMP instruction from the memory, decode the SUB instruction and execute the ADD instruction.



The pipeline design for each ARM family differs. For example, the ARM9 core increases the pipeline length to five stages as shown in the figure below.



The ARM10 increases the pipeline length still further by adding a sixth stage as shown in the figure below.



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.

Q10. Explain briefly the interrupt and the vector table.
Answer:
When an exception or interrupt occurs, the processor sets the program counter (pc) to a specific memory address. The address is within a specified address range called the vector table. The entries in the vector table are the instructions that branch to specific routines designed to handle 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 optionally located at higher address in memory starting at the 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 start of a specific routine.
Following is the vector table:

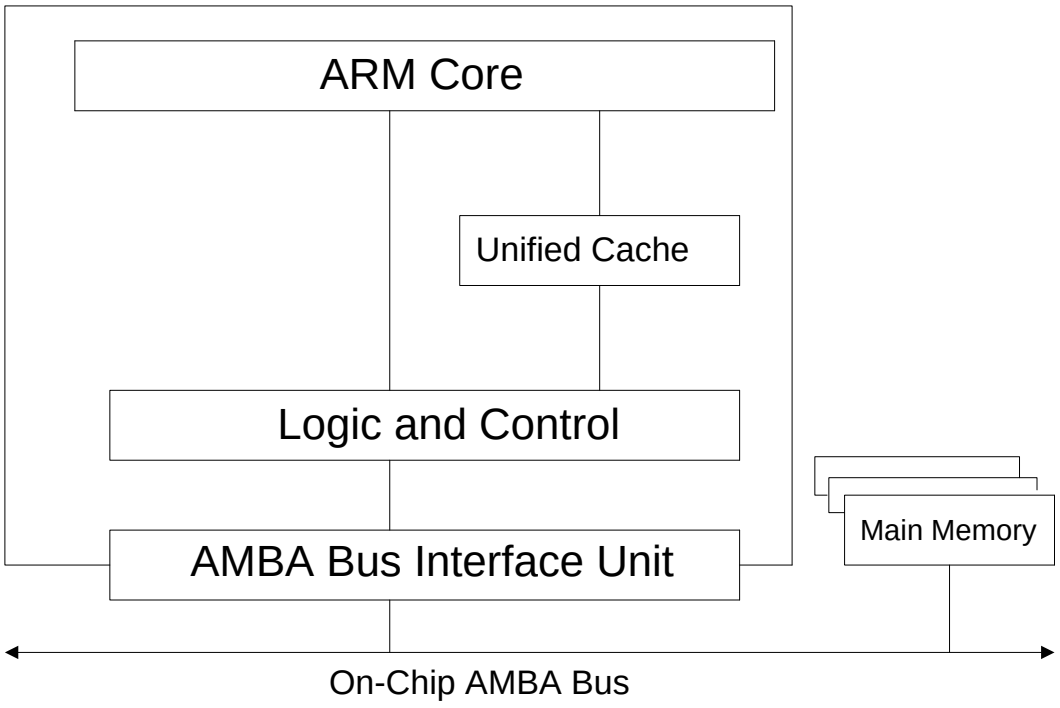
Exception/Interrupt	Shorthand	Address	High address
Reset	RESET	0x00000000	0xffff0000
Undefined instruction	UNDEF	0x00000004	0xffff0004
Software interrupt	SWI	0x00000008	0xffff0008
Prefetch abort	PABT	0x0000000c	0xffff000c
Data abort	DABT	0x00000010	0xffff0010
Reserved	---	0x00000014	0xffff0014
Interrupt request	IRQ	0x00000018	0xffff0018
Fast interrupt request	FIQ	0x0000001c	0xffff001c

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 the instruction.
Software interrupt vector is called when SWI instruction is executed. The SWI 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.
Data abort vectors 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.

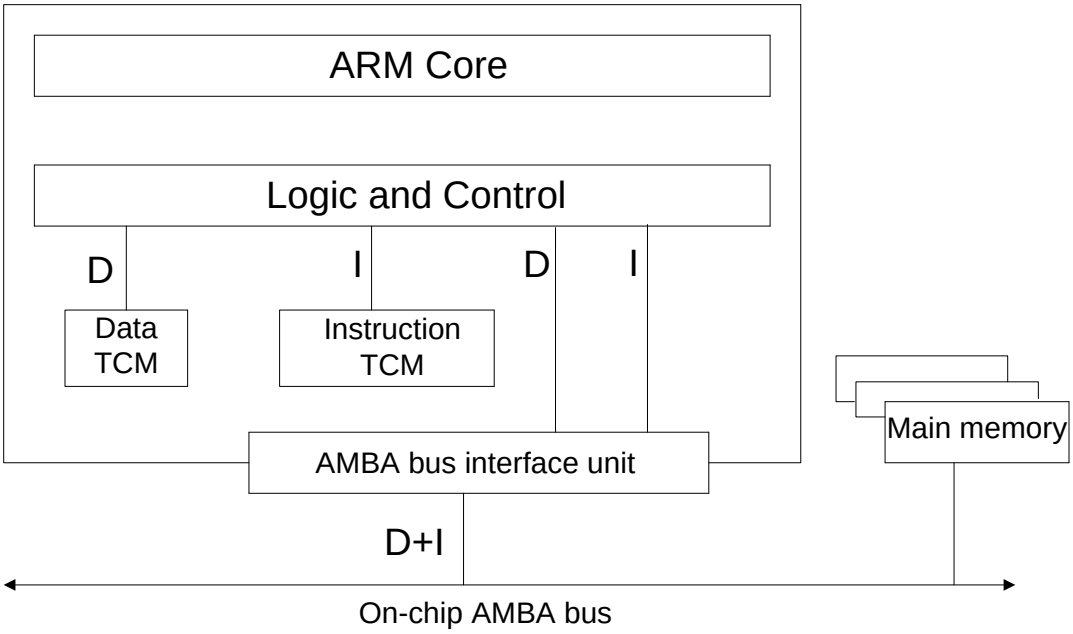
Q11. Discuss the core extensions for ARM processor.

Answer: There are three core extensions wrap around ARM processor: 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.
ARM has two forms of cache. The first found attached to the Von Neumann-style cores. It combines both data and instruction into a single unified cache as shown in the figure below.



The second form, attached to the Harvard-style cores, has separate cache for data and instruction.

A cache provides an overall increase 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.



By combining both technologies, ARM processors can behave both improved performance and predictable real-time response. The following diagram shows an example of core with a combination of caches and TCMs.

Memory management: Embedded systems often use multiple memory devices. It is usually necessary to have a method to help organize these devices and protect the system from applications trying to make appropriate accesses to hardware. This is achieved with the assistance of memory management hardware.

ARM cores have three different types of memory management hardware- no extensions provide 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 normally used for small, simple embedded systems that require no protection from rogue applications.

Memory protection unit (MPU) employs 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 permission but don't have a complex memory map.

Memory management unit (MMU) 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 virtual to physical address map as well as access permission. MMU designed for more sophisticated system 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. The coprocessor can also extend the instruction set by providing a specialized instructions that can be added to standard ARM instruction set to process vector floating-point (VFP) operations.
These new instructions are processed in the decode stage of the ARM pipeline. If the decode stage sees a coprocessor instruction, then it offers it to the relevant coprocessor. But, if the coprocessor is not present or doesn't recognize the instruction, then the ARM takes an undefined instruction exception.