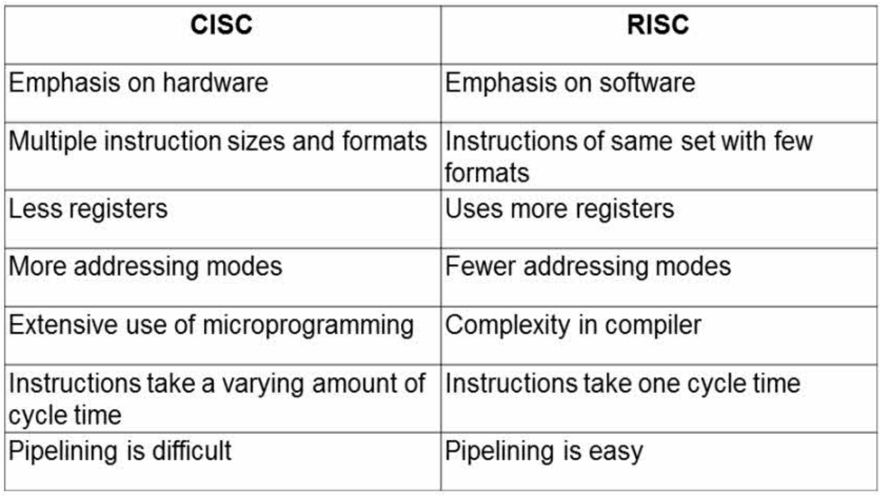
15CS44: MICROPROCESSORS AND MICROCONTROLLERS

QUESTION BANK with SOLUTIONS

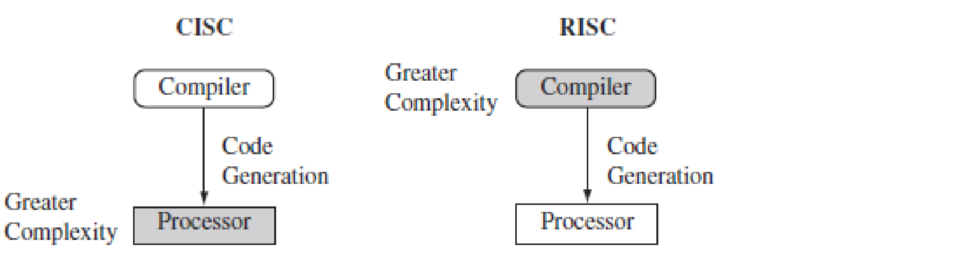
<u>MODULE-4</u>

1) Differentiate CISC and RISC architectures.



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

2) Explain the important design rules of RISC philosophy.



CISC vs. RISC. CISC emphasizes hardware complexity. RISC emphasizes compiler complexity.

The RISC philosophy is implemented with four major design rules:

1. ***Instructions***—RISC processors have a reduced number of instruction classes. These classes provide simple operations that can each execute in a single cycle. The compiler or programmer synthesizes complicated operations (for example, a divide operation) by combining several simple instructions. Each instruction is a fixed length to allow the

pipeline to fetch future instructions before decoding the current instruction. In contrast, in CISC processors the instructions are often of variable size and take many cycles to execute.

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. There is no need for an instruction to be executed by a miniprogram called microcode as on CISC processors.
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. In contrast, CISC processors have dedicated registers for specific purposes.
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. 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.

3) Which are the different features of ARM instruction set that make it suitable for embedded applications.

The ARM instruction set differs from the pure RISC definition in several ways that make the ARM instruction set suitable for embedded applications:

■ ***Variable cycle execution for certain instructions***—Not every ARM instruction executes in a single cycle. For 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 since sequential memory accesses are often faster than random accesses. Code density is also improved since multiple register transfers are common operations at the start and end of functions.

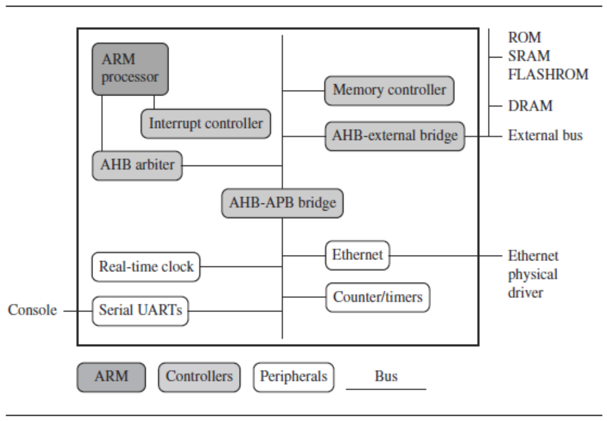
■ ***Inline barrel shifter leading to more complex instructions***—The inline barrel shifter is a hardware component that pre-processes one of the input registers before it is used by an instruction. This expands the capability of many instructions to improve core performance and code density. We explain this feature in more detail in Chapters 2, 3, and 4.

■ ***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.

4) With a neat diagram explain the different hardware components of an embedded device based on ARM core.



An example of an ARM-based embedded device, a microcontroller.

■ **The *ARM processor*** controls the embedded device. Different versions of the ARM processor are available to suit the desired operating characteristics. 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.

5) Explain the AMBA bus protocol.

The Advanced Microcontroller Bus Architecture (AMBA) was introduced in 1996 and has been widely adopted as the on-chip bus architecture used for ARM processors. The first AMBA buses introduced were the ARM System Bus (ASB) and the ARM Peripheral Bus (APB). Later ARM introduced another bus design, called the ARM High Performance Bus (AHB). Using AMBA, peripheral designers can reuse the same design on multiple projects. Because there are a large number of peripherals developed with an AMBA interface, hardware designers have a wide choice of tested and proven peripherals for use in a device. A peripheral can simply be bolted onto the on-chip bus without having to redesign an interface for different processor architecture. This plug-and-play interface for hardware developers improves availability and time to market.

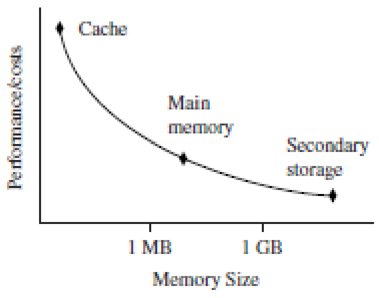
AHB provides higher data throughput than ASB because it is based on a centralized multiplexed bus scheme rather than the ASB bidirectional bus design. This change allows the AHB bus to run at higher clock speeds and to be the first ARM bus to support widths of 64 and 128 bits. ARM has introduced two variations on the AHB bus: Multi-layer AHB and AHB-Lite. In contrast to the original AHB, which allows a single bus master to be active on the bus at any time, the Multi-layer AHB bus allows multiple active bus masters. AHB-Lite is a subset of the AHB bus and it is limited to a single bus master. This bus was developed for designs that do not require the full features of the standard AHB bus.

6) Give a detailed account of different types of memories that can be used in embedded systems based on the hierarchy, width and types.

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

- **Hierarchy**

All computer systems have memory arranged in some form of hierarchy. Figure shows the memory trade-offs: the fastest memory cache is physically located nearer the ARM processor core and the slowest secondary memory is set further away. Generally the closer memory is to the processor core, the more it costs and the smaller its capacity. The cache is placed between main memory and the core. It is used to speed up data transfer between the processor and main memory. A cache provides an overall increase in performance but with a loss of predictable execution time.



The main memory is large—around 256 KB to 256 MB (or even greater), depending on the application—and is generally stored in separate chips. Load and store instructions access the main memory unless the values have been stored in the cache for fast access. Secondary storage is the largest and slowest form of memory.

- **Width**

Table summarizes theoretical cycle times on an ARM processor using different memory width devices.

Fetching instructions from memory.

Instruction size	8-bit memory	16-bit memory	32-bit memory
ARM 32-bit	4 cycles	2 cycles	1 cycle
Thumb 16-bit	2 cycles	1 cycle	1 cycle

- **Types**

➢ Read-only memory (ROM) is the least flexible of all memory types because it contains an image that is permanently set at production time and cannot be reprogrammed.

- 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.
- 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

7) What 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. These memory devices allow the initialization code to be executed. Some memory devices must be set up by software; for example, when using DRAM, you first have to set up the memory timings and refresh rate before it can be accessed.

- **Interrupt Controllers**

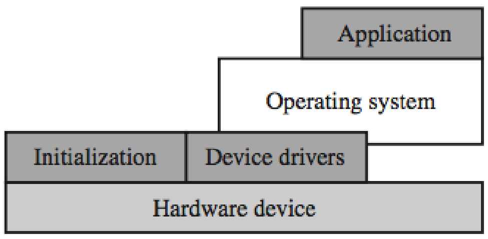
When a peripheral or device requires attention, it raises an interrupt to the processor. An interrupt controller provides a programmable governing policy that allows software to determine which peripheral or device can interrupt the processor at any specific time by setting the appropriate bits in the interrupt controller registers.

There are two types of interrupt controller available for the ARM processor: **the standard interrupt controller** and **the vector interrupt controller (VIC).**

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.

The VIC is more powerful than the standard interrupt controller because it prioritizes interrupts and simplifies the determination of which device caused the interrupt. After associating a priority and a handler address with each interrupt, the VIC only asserts an interrupt signal to the core if the priority of a new interrupt is higher than the currently executing interrupt handler. Depending on its type, the VIC will either call the standard interrupt exception handler, which can load the address of the handler for the device from the VIC, or cause the core to jump to the handler for the device directly.

8) With a neat diagram explain the different software components of an embedded system.



- **Initialization (Boot) Code**

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 and processor caches and initializes some devices. In a simple system the operating system might be replaced by a simple scheduler or debug monitor.

- **Operating System**

The initialization process prepares the hardware for an operating system to take control. 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. A hard real-time application requires a guaranteed response to work at all. In contrast, a soft real-time application requires a good response time, but the performance degrades more gracefully if the response time overruns.

Platform operating systems require a memory management unit to manage large, non- real-time applications and tend to have secondary storage. The Linux operating system is a typical example of a platform operating system.

- **Applications**

The operating system schedules applications—code dedicated to handling a particular task. An application implements a processing task; the operating system controls the environment.

9) Give different applications of ARM processors.

ARM processors are found in numerous market segments, including networking, automotive, mobile and consumer devices, mass storage, and imaging.

Within each segment ARM processors can be found in multiple applications.

For example, the ARM processor is found in networking applications like home gateways, DSL modems for high-speed Internet communication, and 802.11 wireless communications. The mobile device segment is the largest application area for ARM processors because of mobile phones. 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.

10) Give the significance of initialization code in embedded system software.

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 and processor caches and initializes some devices. In a simple system the operating system might be replaced by a simple scheduler or debug monitor.

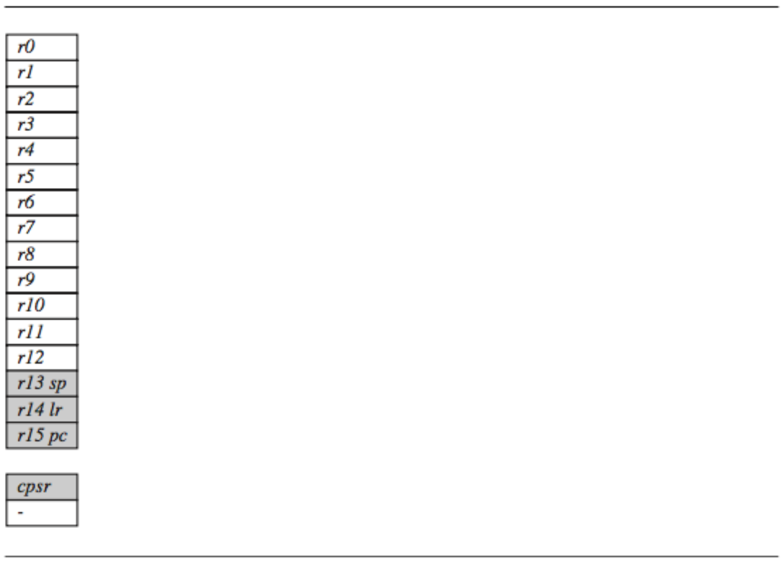
The initialization code handles a number of administrative tasks prior to handing control over to an operating system image. We can group these different tasks into three phases: initial hardware configuration, diagnostics, and booting.

Initial hardware configuration involves setting up the target platform so it can boot an image. Although the target platform itself comes up in a standard configuration, this configuration normally requires modification to satisfy the requirements of the booted 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. It also tracks down standard system-related issues.

Booting involves loading an image and handing control over to that image. The boot process itself can be complicated if the system must boot different operating systems or different versions of the same operating system.

11) With a neat diagram explain the different general purpose registers of ARM processors.



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. 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.

In Figure, 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.

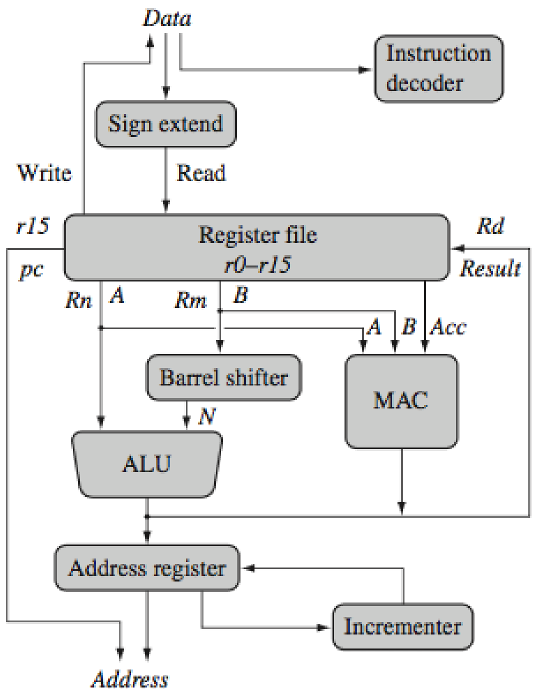
Depending upon the context, registers *r13* and *r14* can also be used as general-purpose registers, which can be particularly useful since these registers are banked during a processor mode change.

12) Explain ARM core dataflow model with a neat diagram.

Data enters the processor core through the *Data* bus. The data may be an instruction to execute or a data item. Figure shows a **Von Neumann** implementation of the ARM— data items and instructions share the same bus. In contrast, Harvard implementations of the ARM use two different buses.

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 **load-*store architecture***. This means 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.



Data items are placed in the ***register file***—a storage bank made up of 32-bit registers. Since the 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. Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the *Address* bus.

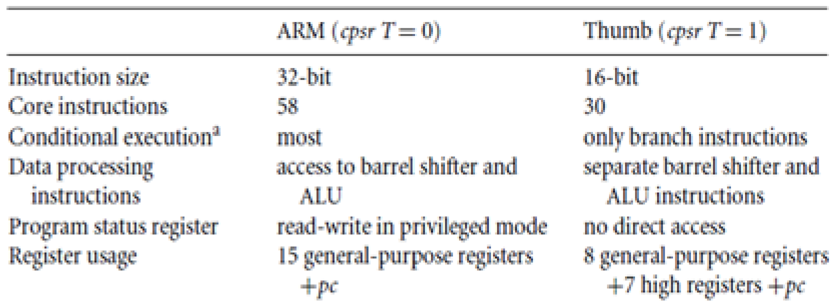
One important feature of the ARM is that register *Rm* alternatively can be preprocessed in the barrel shifter before it enters the ALU. Together the **barrel shifter** and **ALU** can **calculate a wide range of expressions and addresses.**

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.

13) Differentiate ARM and Thumb instruction set features.

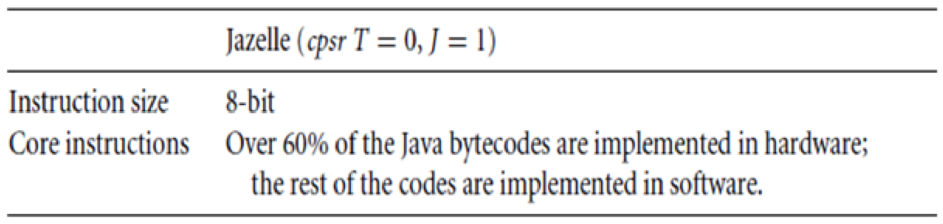
There are three instruction sets: ARM, Thumb, and Jazelle.

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*

Jazelle instruction set features.

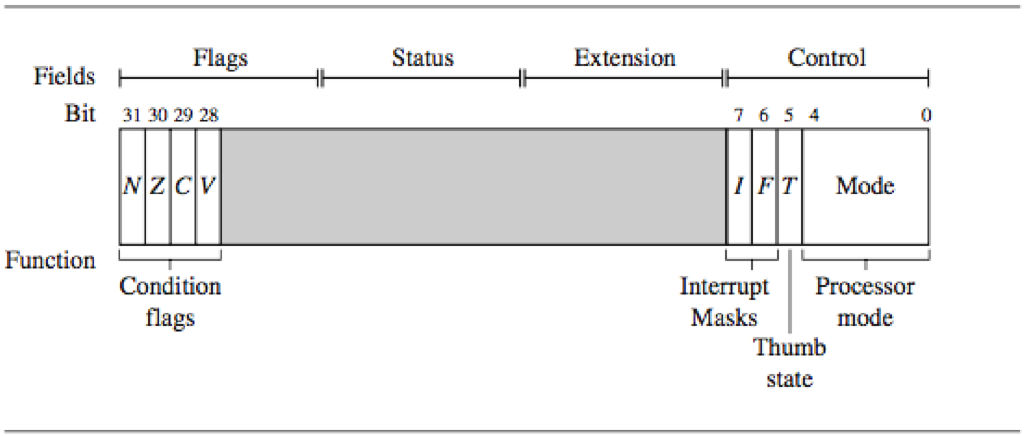


	Jazelle (*cpsr T* = 0, *J* = 1)
Instruction size	8-bit
Core instructions	Over 60% of the Java bytecodes are implemented in hardware; the rest of the codes are implemented in software.

14) Explain current program status register (CPSR) with neat diagram.

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 shows the basic layout of a generic program status register. Note that the 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.



A generic program status register (*psr*).

- **Processor Modes**

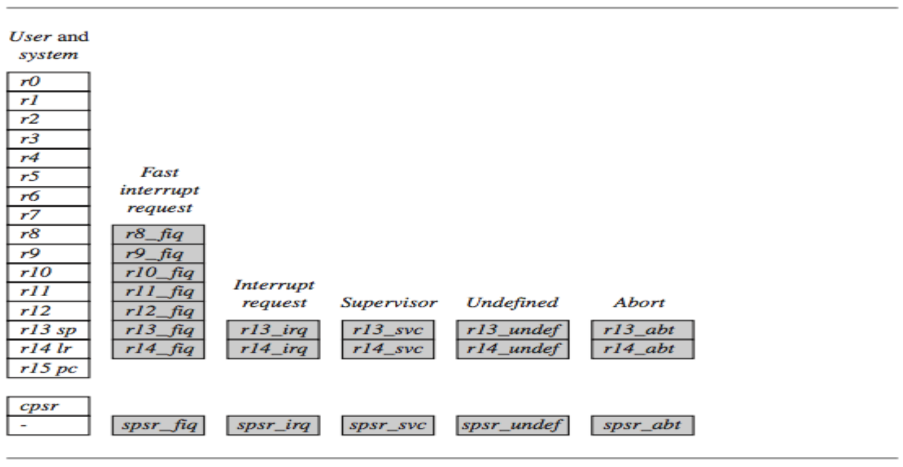
The processor mode determines which registers are active and the access rights to the cpsr register itself. Each processor mode is either privileged or non-privileged: A privileged mode allows full read-write access to the cpsr. Conversely, a non-privileged mode only allows read access to the control 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 non-privileged mode (user).

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

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

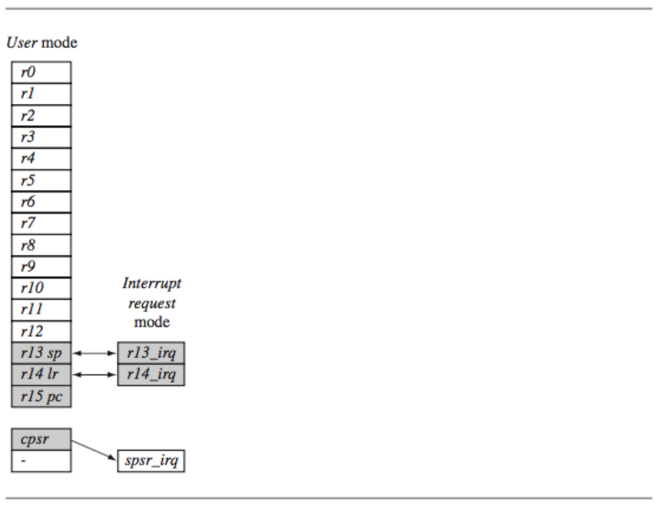
15) What are banked registers? Show how the banked registers are utilized when the user mode changes to IRQ mode.



Complete ARM register set.

The above figure shows all **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; 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 or ***_mode***.

For example, when the processor is in the ***interrupt request*** mode, 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*.



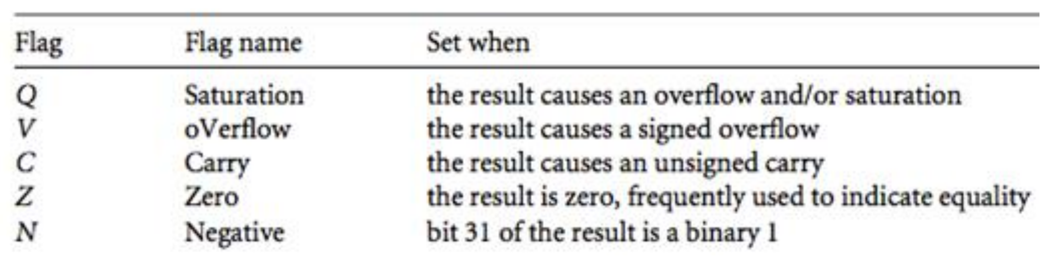
Changing mode on an exception.

Figure illustrates what happens 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. Note *r14_irq* contains the return address and *r13_irq* contains the stack pointer for *interrupt request* mode.

Figure also shows 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 that the *spsr* can only be modified and read in a privileged mode. There is no *spsr* available in *user* mode.

16) Which are the different conditional flags of ARM processor.

Condition flags are updated by comparisons and the result of ALU operations that specify the S instruction suffix. For example, if SUBS subtract instruction results in a register value of zero, then the *Z* flag in the *cpsr* is set. This particular subtract instruction specifically updates the *cpsr*.


Condition flags.

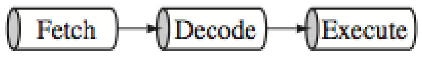
Flag	Flag name	Set when
Q	Saturation	the result causes an overflow and/or saturation
V	oVerflow	the result causes a signed overflow
C	Carry	the result causes an unsigned carry
Z	Zero	the result is zero, frequently used to indicate equality
N	Negative	bit 31 of the result is a binary 1

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.

17) Explain ARM pipeline with 3,5,6 stages.

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.

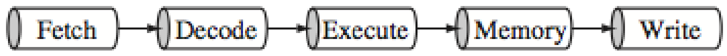


ARM7 Three-stage pipeline.

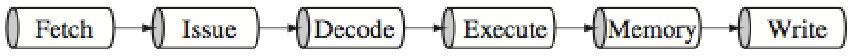
■ *Fetch* loads an instruction from memory.

■ *Decode* identifies the instruction to be executed.

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



ARM9 five-stage pipeline.



ARM10 six-stage pipeline.

The pipeline design for each ARM family differs. For example, The ARM9 core increases the pipeline length to five stages. The ARM9 adds a memory and writeback stage, which allows the ARM9 to process on average 1.1 Dhrystone MIPS per MHz—an increase in instruction throughput by around 13% compared with an ARM7. The maximum core frequency attainable using an ARM9 is also higher.

The ARM10 increases the pipeline length still further by adding a sixth stage. The ARM10 can process on average 1.3 Dhrystone MIPS per MHz, about 34% more throughput than an ARM7 processor core, but again at a higher latency cost.

Even though the ARM9 and ARM10 pipelines are different, they still use the same *pipeline executing characteristics* as an ARM7. Code written for the ARM7 will execute on an ARM9 or ARM10.

18) What are interrupts or exceptions? How are they handled in ARM processors?

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 (see Table 2.6). 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.

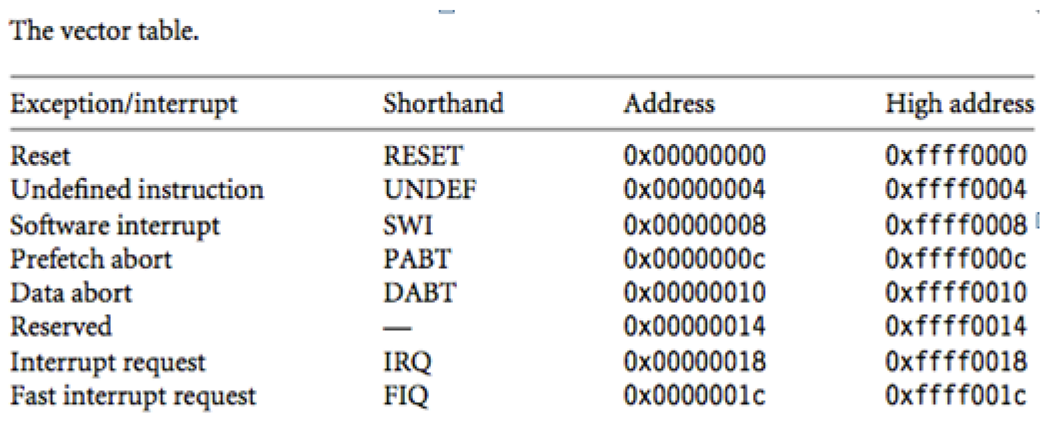
■ *Software interrupt vector* is called when you execute a SWI instruction. The SWI instruction is frequently used as the mechanism to invoke an operating system routine.

■ *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

19) Explain Non-protected memory, MPU & MMU.

■ ***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.

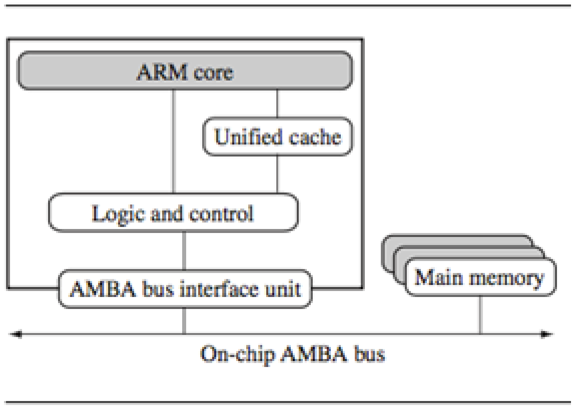
■ ***MMUs*** are the most comprehensive **memory management** hardware available on the ARM. The MMU uses a set of translation tables to provide fine-grained control over memory. These tables are stored in main memory and provide a virtual-to-physical address map as well as access permissions. MMUs are designed for more sophisticated platform operating systems that support multitasking.

20) What are the different techniques of core extensions?

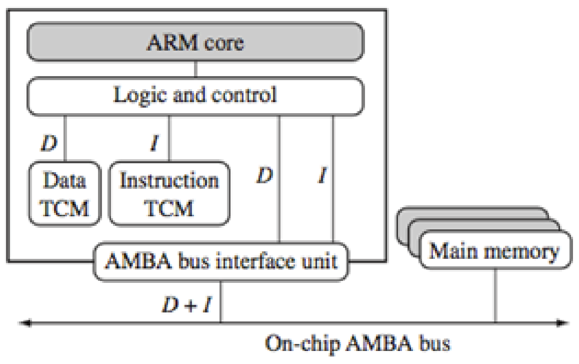
A. Cache and Tightly Coupled Memory

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.

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. 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.

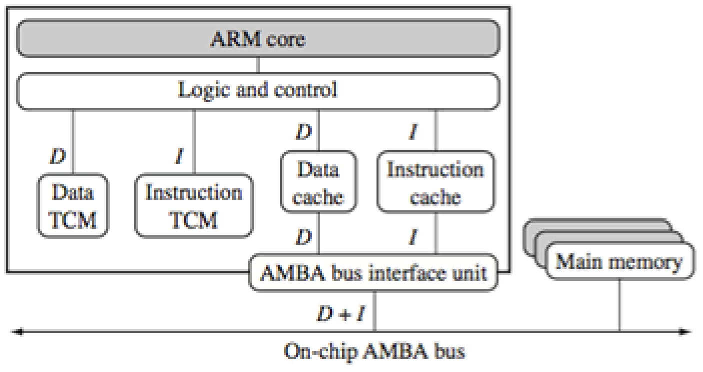


A simplified Harvard architecture with TCMs.

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. An example of a processor with TCMs is shown in Figure.

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



A simplified Harvard architecture with caches and TCMs.

B. 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.

■ ***MMU*s** 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.

C. 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. Consider, for example, coprocessor 15: The ARM processor uses coprocessor 15 registers to control the cache, TCMs, and memory management.

The coprocessor can also extend the instruction set by providing a specialized group of new instructions. For example, there are a set of specialized instructions that can be added to the 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, which allows you to emulate the behavior of the coprocessor in software.