CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test - III

Sub:	MICROCONTROLLERS							Code:	BCS402
Date:		Duration:	90 mins	Max Marks:	50	Sem:	4th C	Branch:	CS(DS)

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	Explain with a neat diagram memory Hierarchy.	10	CO5	L2
2	With a neat diagram explain ARM processor exceptions and modes.	10	CO4	L2
3	Explain assigning interrupts and interrupt latency.	10	CO4	L2
4	Define Firmware. Explain firmware execution flow and explain Red Hat RedBoot.	10	CO4	L3
5	Briefly explain cache line replacement policies.	10	CO5	L2
6	Explain the basic architecture of cache memory.	10	CO5	L2
7	Briefly explain what happens when an IRQ and FIQ exception is raised with an ARM processor.	10	CO4	L2

CCI **CI** **HOD**

CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test - III

Sub:	MICROCONTROLLERS							Code:	BCS402
Date:		Duration:	90 mins	Max Marks:	50	Sem:	4th C	Branch:	CS(DS)

Answer Any FIVE FULL Questions

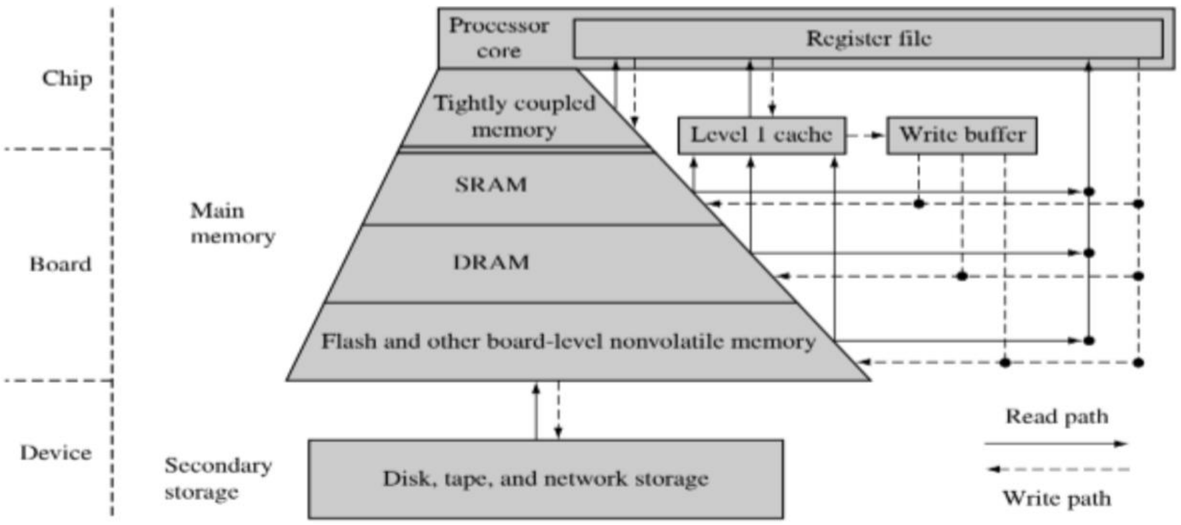
		Marks	OBE	
			CO	RBT
1	Explain with a neat diagram memory Hierarchy.	10	CO5	L2
2	With a neat diagram explain ARM processor exceptions and modes.	10	CO4	L2
3	Explain assigning interrupts and interrupt latency.	10	CO4	L2
4	Define Firmware. Explain firmware execution flow and explain Red Hat RedBoot.	10	CO4	L3
5	Briefly explain cache line replacement policies.	10	CO5	L2
6	Explain the basic architecture of cache memory.	10	CO5	L2
7	Briefly explain what happens when an IRQ and FIQ exception is raised with an ARM processor.	10	CO4	L2

CCI **CI** **HOD**

IAT III_MICROCONTROLLERS_solution

1.

The Memory Hierarchy and Cache Memory



The Memory Hierarchy and Cache Memory

- ➔ A memory hierarchy depends as much on architectural design as on the technology surrounding it.
- ➔ For example, TCM and SRAM are of the same technology yet differ in architectural placement: TCM is located on the chip, while SRAM is located on a board.
- ➔ A cache may be incorporated between any level in the hierarchy where there is a significant access time difference between memory components.
- ➔ A cache can improve system performance whenever such a difference exists.
- ➔ A cache memory system takes information stored in a lower level of the hierarchy and temporarily moves it to a higher level.

The Memory Hierarchy and Cache Memory

Processor core:

- ➔ The innermost level of the hierarchy is at the processor core.
- ➔ This memory is so tightly coupled to the processor that in many ways it is difficult to think of it as separate from the processor.
- ➔ This memory is known as a register file.
- ➔ These registers are integral to the processor core and provide the fastest possible memory access in the system.

The Memory Hierarchy and Cache Memory

Tightly coupled memory (TCM):

- ➔ Memory components are connected to the processor core through dedicated on-chip interfaces.
- ➔ The primary level is main memory.
- ➔ It includes volatile components like SRAM and DRAM, and nonvolatile components like flash memory.
- ➔ The purpose of main memory is to hold programs while they are running on a system.

L1 and L2 cache (Primary and Secondary caches)

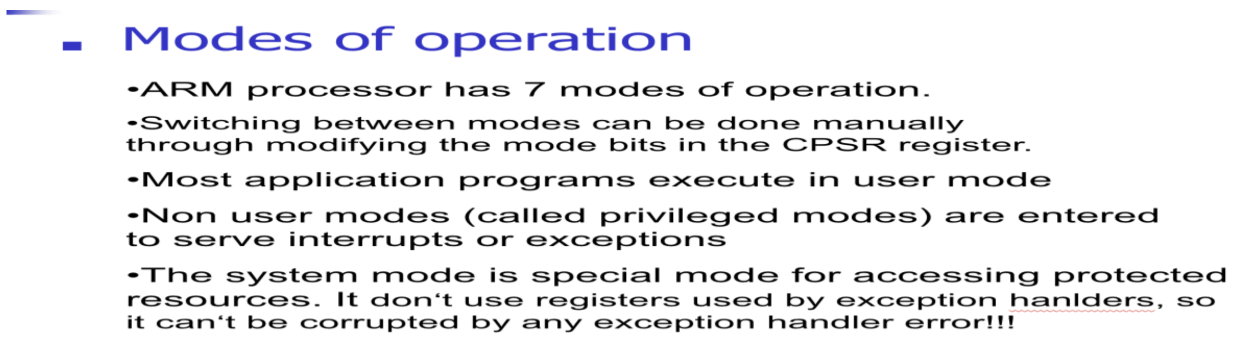
- ➔ The L1 and L2 caches are also known as the primary and secondary caches.
- ➔ The L1 cache is an array of high-speed, on-chip memory that temporarily holds code and data from a slower level.
- ➔ A cache holds this information to decrease the time required to access both instructions and data.
- ➔ The write buffer is a very small FIFO buffer that supports writes to main memory from the cache.
- ➔ An L2 cache is located between the L1 cache and slower memory.

The Memory Hierarchy and Cache Memory

The secondary storage:

- ➔ storage—large, slow, relatively inexpensive mass storage devices such as disk drives or removable memory.
- ➔ In this level is data derived from peripheral devices, which are characterized by their extremely long access times.
- ➔ Secondary memory is used to store unused portions of very large programs that do not fit in main memory and programs that are not currently executing.

2.



Modes of operation

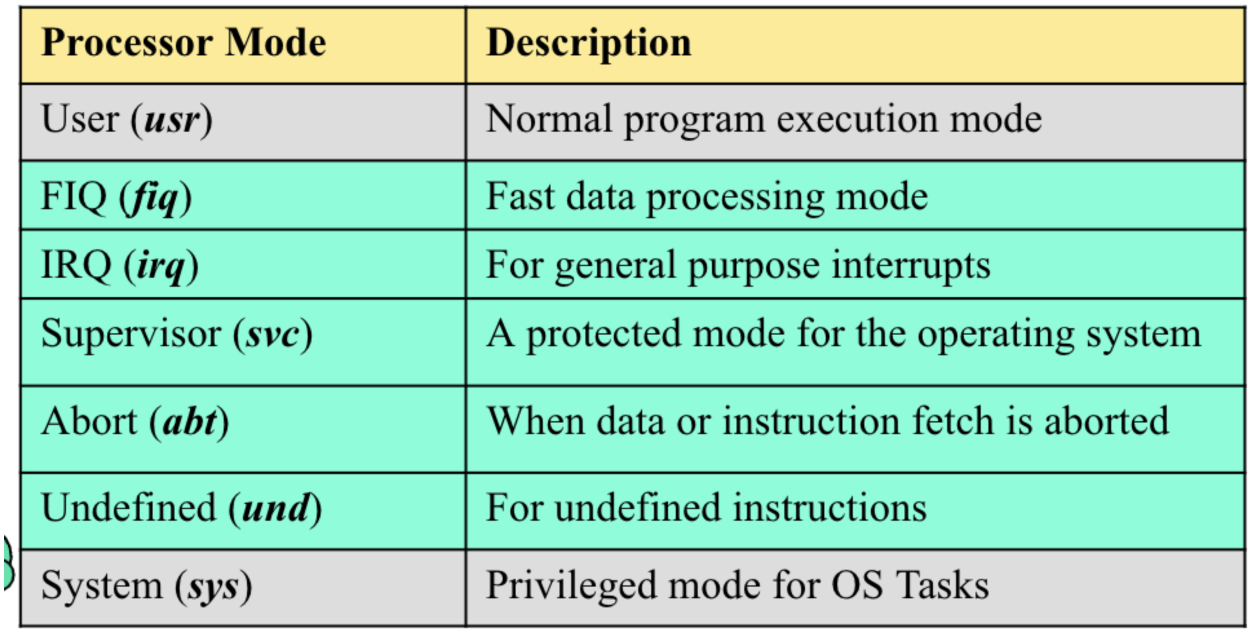
•ARM processor has 7 modes of operation.

•Switching between modes can be done manually through modifying the mode bits in the CPSR register.

•Most application programs execute in user mode

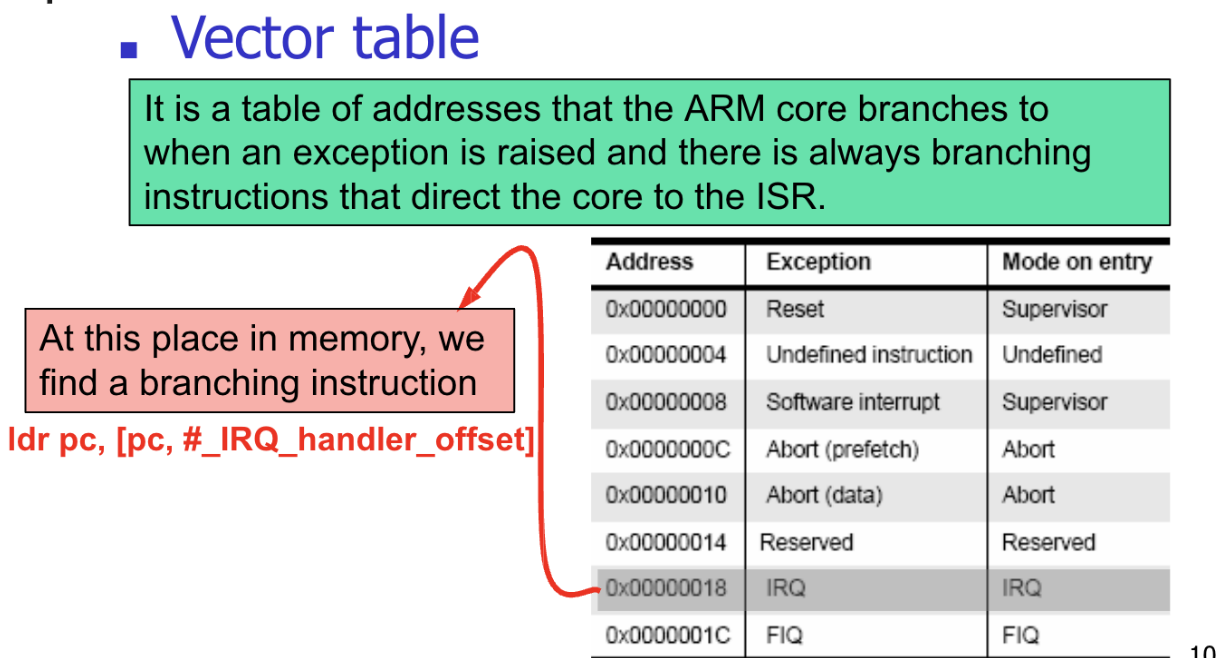
•Non user modes (called privileged modes) are entered to serve interrupts or exceptions

•The system mode is special mode for accessing protected resources. It don't use registers used by exception hanlders, so it can't be corrupted by any exception handler error!!!



Processor Mode	Description
User (***usr***)	Normal program execution mode
FIQ (***fiq***)	Fast data processing mode
IRQ (***irq***)	For general purpose interrupts
Supervisor (***svc***)	A protected mode for the operating system
Abort (***abt***)	When data or instruction fetch is aborted
Undefined (***und***)	For undefined instructions
System (***sys***)	Privileged mode for OS Tasks

An exception is any condition that needs to halt normal execution of the instructions.



Vector table

It is a table of addresses that the ARM core branches to when an exception is raised and there is always branching instructions that direct the core to the ISR.

At this place in memory, we find a branching instruction

ldr pc, [pc, #_IRQ_handler_offset]

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

10

3.

Assigning interrupts

It is up to the system designer who can decide which HW peripheral can produce which interrupt.

But system designers have adopted a standard design for assigning interrupts:

•SWI are used to call privileged OS routines.
•IRQ are assigned to general purpose interrupts like periodic timers.
•FIQ is reserved for one single interrupt source that requires fast response time.

Interrupt latency

It is the interval of time from an external interrupt signal being raised to the first fetch of an instruction of the ISR of the raised interrupt signal.

System architects try to achieve two main goals:

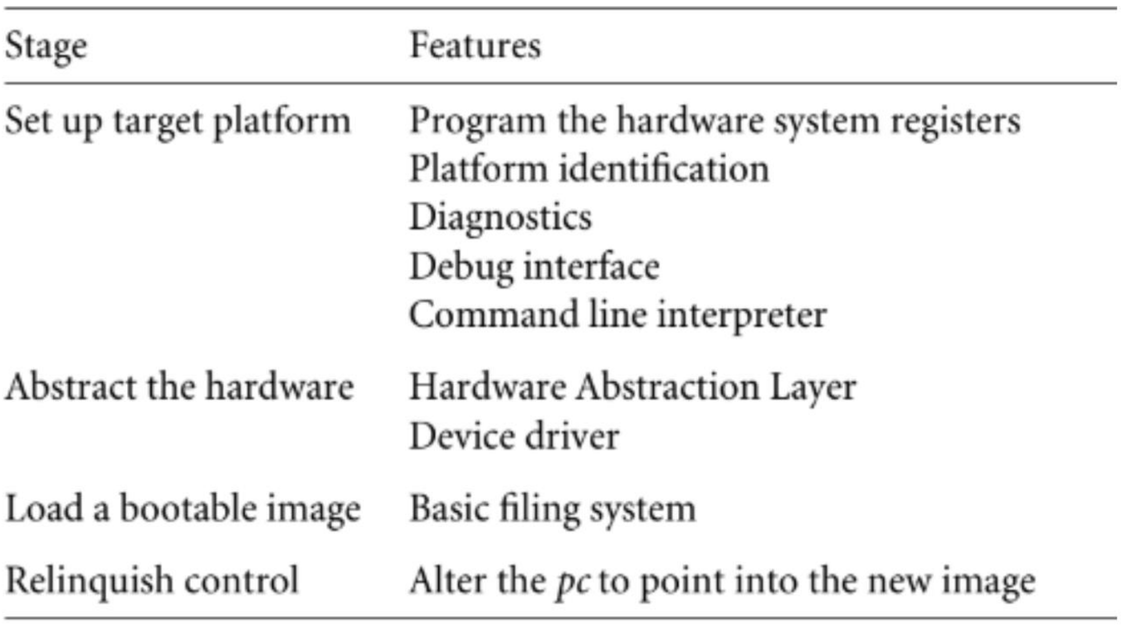
•To handle multiple interrupts simultaneously.
•To minimize the interrupt latency.

And this can be done by 2 methods:

•allow nested interrupt handling
•give priorities to different interrupt sources

4.

- The firmware is the deeply embedded, low-level software that provides an interface between the hardware and the application/operating system level software.
- It resides in the ROM and executes when power is applied to the embedded hardware system.
- Firmware can remain active after system initialization and supports basic system operations.
- The choice of which firmware to use for a particular ARM-based system depends upon the specific application, which can range from loading and executing a sophisticated operating system to simply relinquishing control to a small microkernel.



Stage	Features
Set up target platform	Program the hardware system registers Platform identification Diagnostics Debug interface Command line interpreter
Abstract the hardware	Hardware Abstraction Layer Device driver
Load a bootable image	Basic filing system
Relinquish control	Alter the *pc* to point into the new image

- RedBoot is a firmware tool developed by Red Hat. It is provided under an open source license with no royalties or up front fees. RedBoot is designed to execute on different CPUs (for instance, ARM, MIPS, SH, and so on).
  It provides both debug capability through GNU Debugger (GDB), as well as a bootloader.
- The RedBoot software core is based on a HAL.

▪ Communication—configuration is over serial or Ethernet.
▪ RedBoot supports a range of network standards, such as bootp, telnet, and tftp.
▪ Flash ROM memory management—provides a set of filing system routines that can
download, update, and erase images in flash ROM.
▪ In addition, the images can either be compressed or uncompressed.
▪ Full operating system support—supports the loading and booting of Embedded Linux,
Red Hat eCos, and many other popular operating systems. For Embedded Linux,
RedBoot supports the ability to define parameters that are passed directly to the kernel upon booting.

Cache Policy

- ➢ There are three policies that determine the operation of a cache: the write policy, the replacement policy, and the allocation policy.
- ➢ The cache write policy determines where data is stored during processor write operations.
- ➢ The replacement policy selects the cache line in a set that is used for the next line fill during a cache miss.
- ➢ The allocation policy determines when the cache controller allocates a cache line.

Write Policy

- ➢ When the processor core writes to memory, the cache controller has two alternatives for its write policy.
- ➢ The controller can write to both the cache and main memory, updating the values in both locations; this approach is known as writethrough.
- ➢ The cache controller can write to cache memory and not update main memory, this is known as writeback or copyback.

Writethrough

- ➢ When the cache controller uses a writethrough policy, it writes to both cache and main memory when there is a cache hit on write, ensuring that the cache and main memory stay coherent at all times.
- ➢ Under this policy, the cache controller performs a write to main memory for each write to cache memory.
- ➢ Because of the write to main memory,a writethrough policy is slower than a writeback policy.

Writeback

- When a cache controller uses a writeback policy, it writes to valid cache data memory and not to main memory.
- Valid cache lines and main memory may contain different data.
- The cache line holds the most recent data, and main memory contains older data, which has not been updated.

Cache Line Replacement Policies

- On a cache miss, the cache controller must select a cache line from the available set in cache memory to store the new information from main memory.
- The cache line selected for replacement is known as a **victim.**
- If the victim contains valid, dirty data, the controller must write the dirty data from the cache memory to main memory before it copies new data into the victim cache line.
- The process of selecting and replacing a victim cache line is known a eviction.

Cache Line Replacement Policies

- The strategy implemented in a cache controller to select the next victim is called its **replacement policy**.
- The replacement policy selects a cache line from the available associative member set; that is, it selects the way to use in the next cache line replacement.
- To summarize the overall process, the set index selects the set of cache lines available in the ways, and the replacement policy selects the specific cache line from the set to replace.

Replacement Policies

Round-robin or cyclic replacement:

- ➢ Simply selects the next cache line in a set to replace.
- ➢ The selection algorithm uses a sequential, incrementing victim counter that increments each time the cache controller allocates a cache line.
- ➢ When the victim counter reaches a maximum value, it is reset to a defined base value.

Replacement Policies

Pseudorandom replacement policy:

- ➢ Randomly selects the next cache line in a set to replace.
- ➢ The selection algorithm uses a nonsequential incrementing victim counter.
- ➢ In a pseudorandom replacement algorithm the controller increments the victim counter by randomly selecting an increment value and adding this value to the victim counter.
- ➢ When the victim counter reaches a maximum value, it is reset to a defined base value.

Allocation Policy on a Cache Miss

- ❖ There are two strategies ARM caches may use to allocate a cache line after a the occurrence of a cache miss.
- ❖ The first strategy is known as read-allocate
- ❖ The second strategy is known as read-write-allocate.
- ❖ The ARM7, ARM9, and ARM10 cores use a read-allocate on miss policy.
- ❖ The Intel XScale supports both read-allocate and write-allocate on miss.

Read Allocate

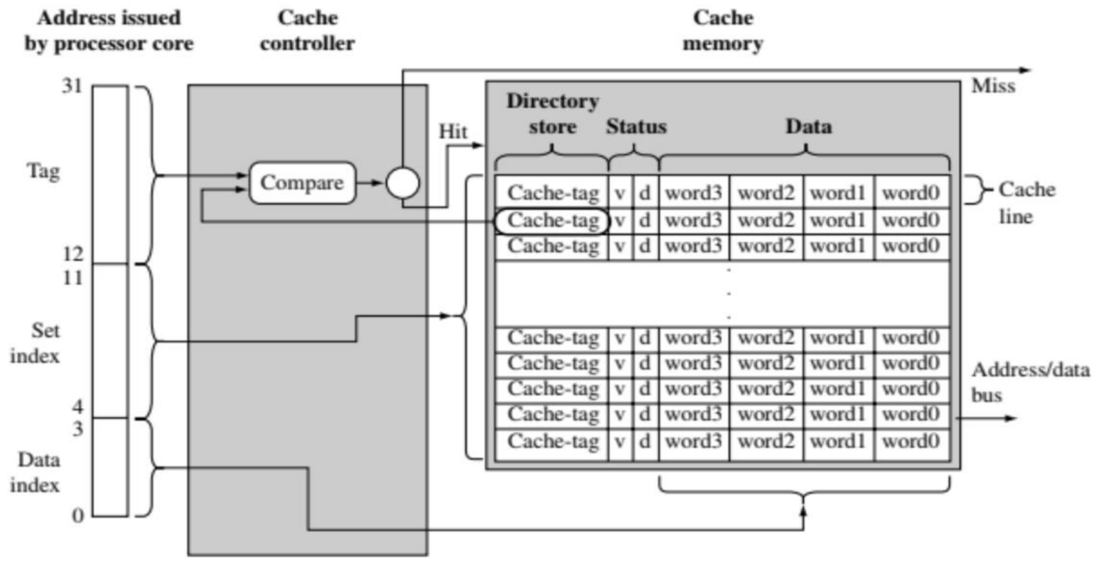
- ❖ A read allocate on cache miss policy allocates a cache line only during a read from main memory.
- ❖ If the victim cache line contains valid data, then it is written to main memory before the cache line is filled with new data.
- ❖ Under this strategy, a write of new data to memory does not update the contents of the cache memory unless a cache line was allocated on a previous read from main memory.
- ❖ If the cache line contains valid data, then a write updates the cache and may update main memory if the cache write policy is writethrough.
- ❖ If the data is not in cache, the controller writes to main memory only.

Read-Write Allocate

- ❖ A read-write allocate on cache miss policy allocates a cache line for either a read or write to memory.
- ❖ Any load or store operation made to main memory, which is not in cache memory, allocates a cache line.
- ❖ On memory reads the controller uses a read-allocate policy.
- ❖ On a write, the controller also allocates a cache line.
- ❖ If the victim cache line contains valid data, then it is first written back to main memory before the cache controller fills the victim cache line with new data from main memory.
- ❖ If the cache line is not valid, it simply does a cache line fill.
- ❖ After the cache line is filled from main memory, the controller writes the data to the corresponding data location within the cache line.
- ❖ The cached core also updates main memory if it is a writethrough cache.

6.

Cache Architecture



Cache Architecture

ARM uses two bus architectures in its cached cores, the Von Neumann and the Harvard.

The Von Neumann and Harvard bus architectures differ in the separation of the instruction and data paths between the core and memory.

A different cache design is used to support the two architectures.

Von Neumann architecture

In processor cores using the Von Neumann architecture, there is a single cache used for instruction and data.

This type of cache is known as a unified cache.

A unified cache memory contains both instruction and data values.

Harvard architecture

The Harvard architecture has separate instruction and data buses to improve overall system performance, but supporting the two buses requires two caches.

In processor cores using the Harvard architecture, there are two caches: an instruction cache (I-cache) and a data cache (D-cache).

This type of cache is known as a split cache.

In a split cache, instructions are stored in the instruction cache and data values are stored in the data cache.

Cache Architecture

The two main elements of a cache are the cache controller and the cache memory.

The cache memory is a dedicated memory array accessed in units called cache lines.

The cache controller uses different portions of the address issued by the processor during a memory request to select parts of cache memory.

Cache Architecture

- ➔ It has three main parts:a directory store, a data section, and statu information.
- ➔ All three parts of the cache memory are present for each cache line.
- ➔ The cache must know where the information stored in a cache line originates from in main memory. It uses a directory store to hold the address identifying where the cache line was copied from main memory. The directory entry is known as a **cache-tag.**
- ➔ A cache memory must also store the data read from main memory. This information is held in the **data section.**
- ➔ The size of a cache is defined as the actual code or data the cache can store from main memory. Not included in the cache size is the cache memory required to support cache-tags or status bits.
- ➔ Status bit

Status bits

- ➔ Two common status bits are the valid bit and dirty bit.
- ➔ A valid bit marks a cache line as active, meaning it contains live data originally taken from main memory and is currently available to the processor core on demand.
- ➔ A dirty bit defines whether or not a cache line contains data that is different from the value it represents in main memory.

Cache Controller

- ➔ The cache controller is hardware that copies code or data from main memory to cache memory automatically.
- ➔ It performs this task automatically to conceal cache operation from the software it supports.
- ➔ The same application software can run unaltered on systems with and without a cache.
- ➔ The cache controller intercepts read and write memory requests before passing them on to the memory controller. It processes a request by dividing the address of the request into three fields, the tag field, the set index field, and the data index field.
- ➔ The controller uses the set index portion of the address to locate the cache line within the cache memory that might hold the requested code or data. This cache line contains the cache-tag and status bits, which the controller uses to determine the actual data stored there.

Cache Controller

- ➔ The controller then checks the valid bit to determine if the cache line is active, and compares the cache-tag to the tag field of the requested address.
- ➔ If both the status check and comparison succeed, it is a cache hit.
- ➔ If either the status check or comparison fails, it is a cache miss.
- ➔ On a cache miss, the controller copies an entire cache line from main memory to cache memory and provides the requested code or data to the processor.
- ➔ The copying of a cache line from main memory to cache memory is known as a cache line fill.
- ➔ On a cache hit, the controller supplies the code or data directly from cache memory to the processor.
- ➔ To do this it moves to the next step, which is to use the data index field of the address request to select the actual code or data in the cache line and provide it to the processor.

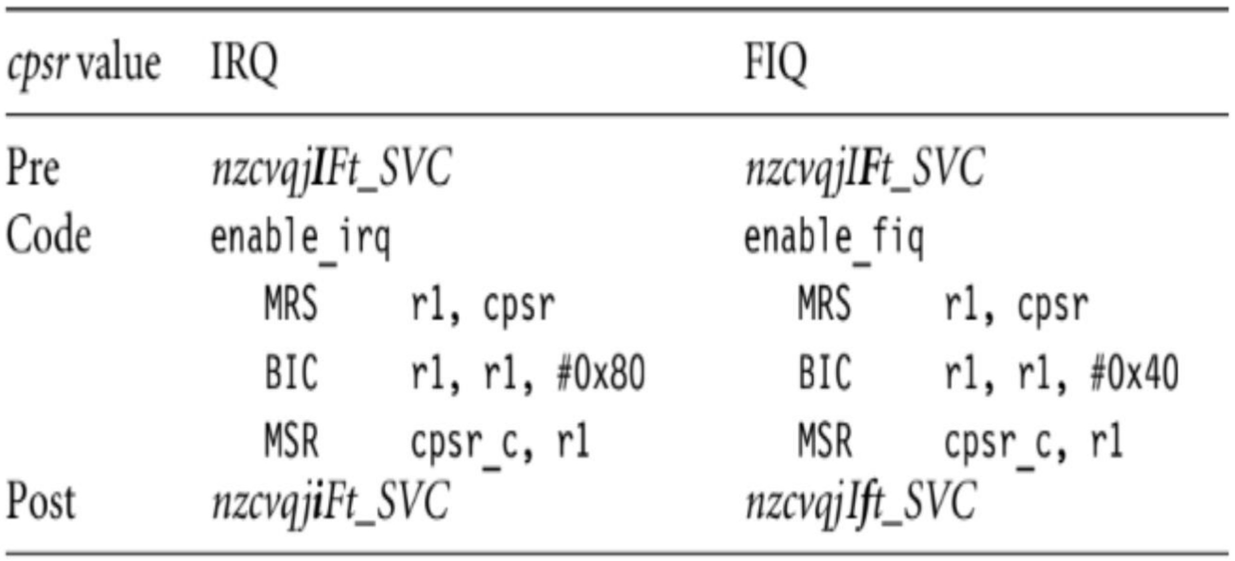
7.

Enabling an IRQ/FIQ Interrupt:

```
MRS     r1, cpsr
BIC r1, r1, #0x80/0x40
MSR     cpsr_c, r1
```

Disabling an IRQ/FIQ Interrupt:

```
MRS     r1, cpsr
ORR     r1, r1, #0x80/0x40
MSR     cpsr_c, r1
```



cpsr value	IRQ	FIQ
Pre	*nzcvqj**I**Ft_SVC*	*nzcvqjI**F**t_SVC*

Code	`enable_irq`
	`MRS r1, cpsr`
	`BIC r1, r1, #0x80`
	`MSR cpsr_c, r1`
		`enable_fiq`
		`MRS r1, cpsr`
		`BIC r1, r1, #0x40`
		`MSR cpsr_c, r1`

Post	*nzcvqj**i**Ft_SVC*	*nzcvqjI**f**t_SVC*