CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

# InternalAssessmentTest -II

| Sub: | MICROCONTROLLERS | | | | | | | Code: | BCS402 |
|---|---|---|---|---|---|---|---|---|---|
| Date: | 26-05-2025 | Duration: | 90mins | Max Marks: | 50 | Sem: | 4th c | Branch: | CS(DS) |

AnswerAnyFIVEFULLQuestions

| | | Marks | OBE | |
|---|---|---|---|---|
| | | | CO | RBT |
| 1 | a. Explain with a neat diagram memory Hierarchy. | 4 | CO5 | L2 |
| | b. Write a short note on instruction scheduling. | 6 | CO3 | L1 |
| 2 | Briefly explain what happens when an IRQ and FIQ exception is raised with an ARMprocessor. | 10 | CO4 | L2 |
| 3 | Define Firmware, Bootloader. Explain firmware execution flow and explain Red HatRedBoot. | 10 | CO4 | L1 |
| 4 | Briefly explain cache line replacement policies. | 10 | CO5 | L2 |
| 5 | With a neat diagram explain ARM processor exceptions and modes. | 10 | CO4 | L2 |
| 6 | Write a C program that prints the square of the integers between 0 to 9 usingfunctions and explainhow to convert this C function to an assembly functionwith command. | 10 | CO3 | L3 |
| 7 | How the function calling efficiently use by ARM through APCS with an exampleprogram. | 10 | CO3 | L2 |

Hiremath 22/5/25

CCI

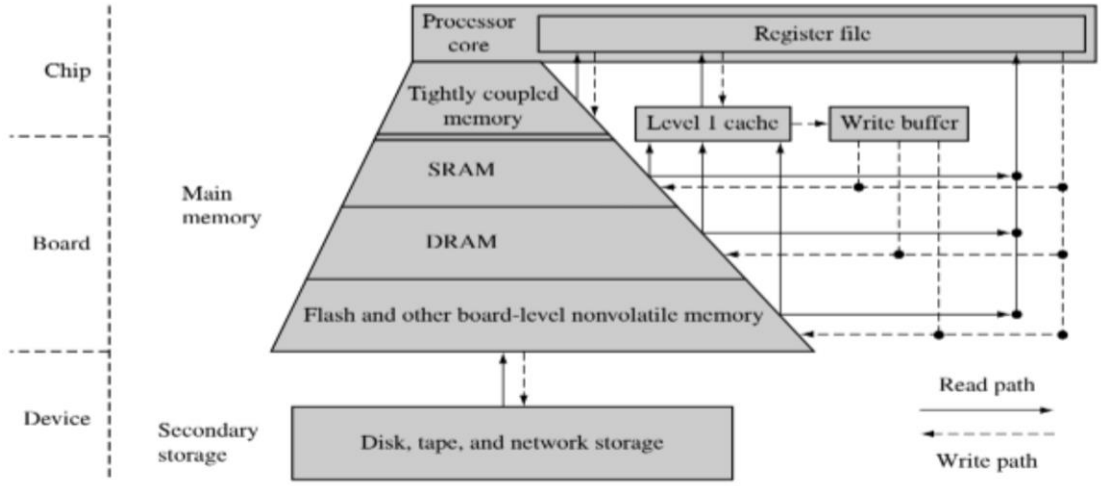
CI

HOD 23/5/25

MC_IAT II_Solution

1. a.

## 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.

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.

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.

- ➔ 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 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.

b.

Instruction scheduling is a compiler optimization technique used in computer architecture to improve the performance of a program by reordering the instructions in such a way that maximizes the utilization of the CPU's execution units and minimizes stalls. The primary goals of instruction scheduling are to reduce pipeline stalls, improve instruction-level parallelism, and optimize the usage of processor resources.

key aspects of instruction scheduling

1. **Dependencies:**
   - **Data Dependencies:** Instructions that depend on the result of previous instructions.
   - **Control Dependencies:** Instructions that depend on the outcome of branch instructions.
   - **Resource Dependencies:** Instructions that require the same computational resources.
2. **Pipeline Stalls:**
   - Occur when the CPU pipeline cannot continue to execute the next instruction because the current instruction has not yet completed. Instruction scheduling aims to minimize these stalls

### 3. Types of Instruction Scheduling:

- **Static Scheduling:** Performed at compile time by the compiler. It does not change during execution.
- **Dynamic Scheduling:** Performed at runtime by the processor. It can adapt to changing conditions and resource availability

**Techniques:**

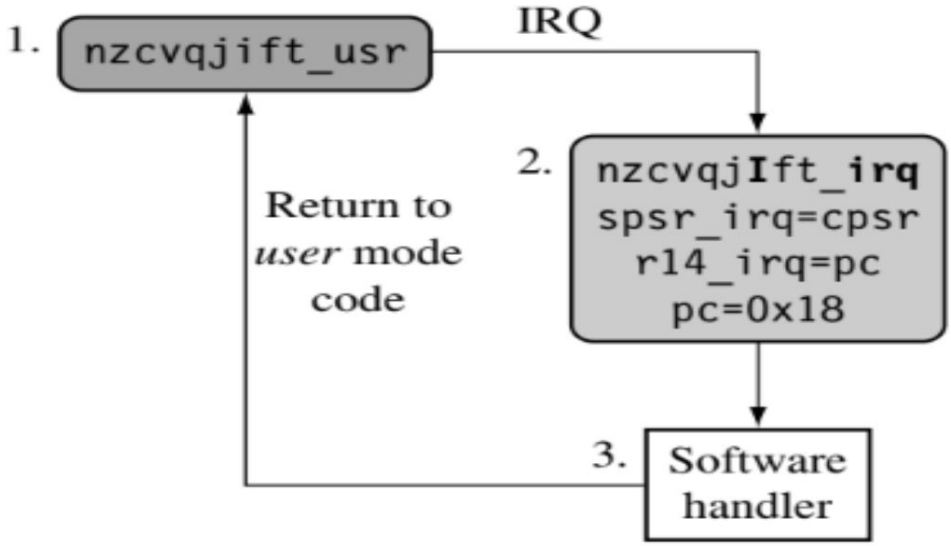
- **List Scheduling:** A priority list of instructions is maintained, and instructions are scheduled based on their dependencies and priorities.
- **Trace Scheduling:** Identifies frequently executed paths (traces) in the program and optimizes these traces.
- **Software Pipelining:** Overlaps the execution of instructions from different iterations of a loop to utilize the pipeline more effectively.

2.

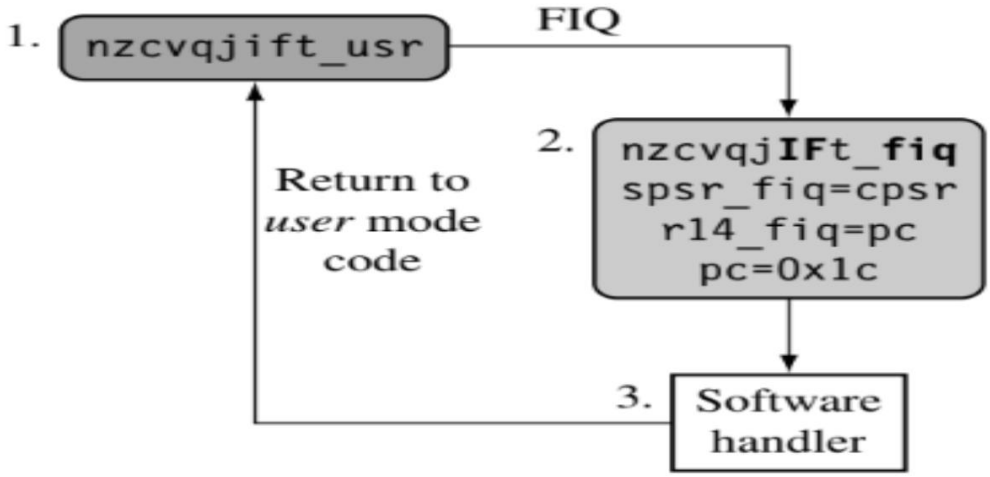
# IRQ and FIQ Exceptions

1. The processor changes to a specific interrupt request mode, which reflects the interrupt being raised.
2. The previous mode's cpsr is saved into the spsr of the new interrupt request mode.
3. The pc is saved in the lr of the new interrupt request mode.
4. Interrupt/s are disabled—either the IRQ or both IRQ and FIQ exceptions are disabled in the cpsr. This immediately stops another interrupt request of the same type being raised.
5. The processor branches to a specific entry in the vector table.

# Interrupt Request (IRQ).




# Fast Interrupt Request (FIQ).




3.

## Firmware

- 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.

## Bootloader

- The bootloader is a small application that installs the operating system or application onto a hardware target.
- The bootloader only exists up to the point that the operating system or application is executing, and it is commonly incorporated into the firmware.

# Firmware execution flow

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

4.

## 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.

- 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.

- ARM cached cores support two replacement policies, either pseudorandom or round-robin.
- Most ARM cores support both policies
- The round-robin replacement policy has greater predictability, which is desirable in an embedded system.
- A round-robin replacement policy is subject to large changes in performance given small changes in memory access.

## 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.

Pseudorandom replacement policy:

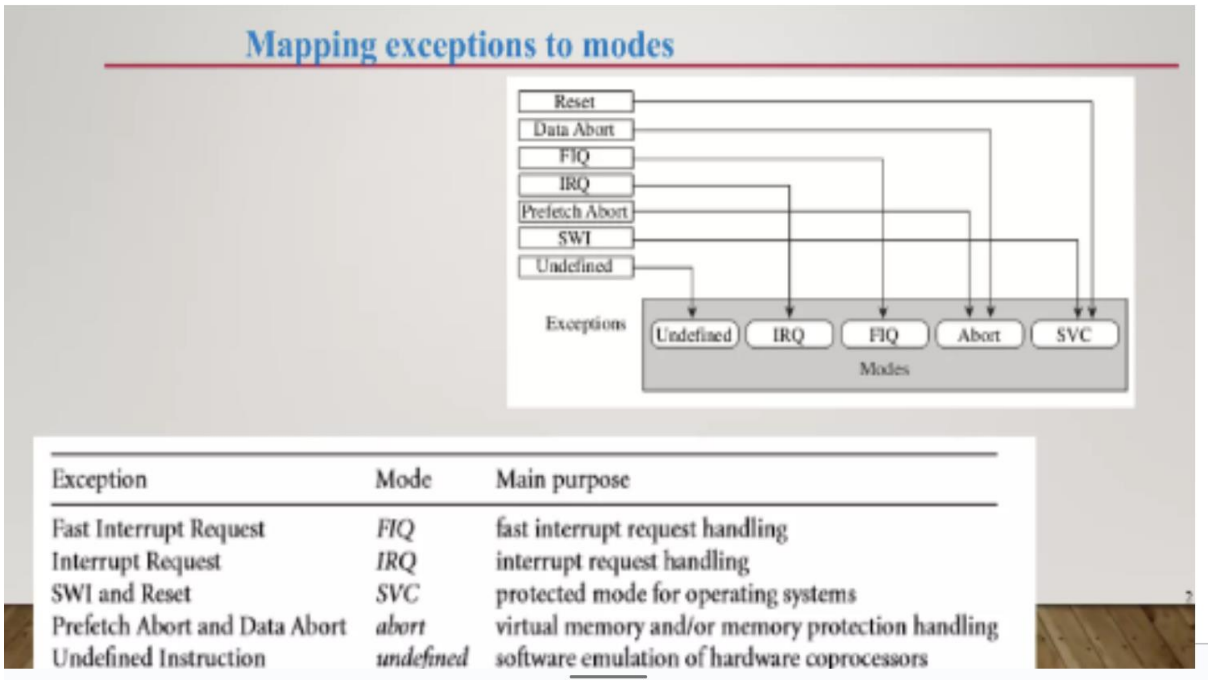
- Randomly selects the next cache line in a set to replace.
- The selection algorithm uses a nonsequential incrementing victim counter.
- In a pseudorandom replacement algorithm the controller increments the victim counter by randomly selecting an increment value and adding this value to the victim counter.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

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

# Exceptions and modes

Each exception causes the ARM core to enter a specific mode.

| Exception | Mode | Purpose |
|---|---|---|
| Fast Interrupt Request | FIQ | Fast interrupt handling |
| Interrupt Request | IRQ | Normal interrupt handling |
| SWI and RESET | SVC | Protected mode for OS |
| Pre-fetch or data abort | ABT | Memory protection handling |
| Undefined Instruction | UND | SW emulation of HW coprocessors |




6.

```
#include &lt;stdio.h&gt;
// Function to calculate square
int square(int n) {
return n * n;
}
int main() {
for (int i = 0; i &lt; 10; i++) {
printf(&quot;Square of %d is %d\n&quot;, i, square(i));
}
return 0;
}

square: // int square(int n)
```

```
PUSH {LR} // Save return address
MUL R0, R0, R0 // R0 = R0 * R0
POP {LR} // Restore return address
BX LR // Return with result in R0
main:
PUSH {R4-R6, LR} // Save used registers and LR
MOV R4, #0 // i = 0
loop:
CMP R4, #10 // if (i &gt;= 10)
BGE end // break
MOV R0, R4 // R0 = i
BL square // Call square(i), result in R0
MOV R5, R0 // Save square result in R5 (for debug)
ADD R4, R4, #1 // i++
B loop // repeat loop
end:
POP {R4-R6, LR}
BX LR
```

7.

The ARM Procedure Call Standard (APCS) defines how functions (procedures) interact in ARM systems—how arguments are passed, results returned, and registers used. It ensures interoperability between code, allows modular development, and helps in optimizing function calls.

If more than 4 arguments: the rest go on the stack.
Return address stored in LR during a function call (BL instruction).
BX LR used to return from a function.

```
int add(int a, int b) {
return a + b;
}
int main() {
int result = add(5, 10);
while(1); // loop forever
}
// Function: int add(int a, int b)
add:
PUSH {LR} // Save return address
ADD R0, R0, R1 // R0 = R0 + R1 (R0 holds a, R1 holds b)
POP {LR} // Restore return address
BX LR // Return (R0 has result)
main:
MOV R0, #5 // First argument (a)
MOV R1, #10 // Second argument (b)
BL add // Call add(a, b), result in R0
MOV R4, R0 // Store result in R4 for observation
B . // Infinite loop (watch R4 in debugger)
```