CMR INSTITUTE OF TECHNOLOGY

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

Internal Assessment Test - I

Sub:	MICROCONTROLLERS							Code:	BCS402
Date:	-03-2025	Duration:	90 mins	Max Marks:	50	Sem:	4th C	Branch:	CS(DS)

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	a. Explain the embedded software with the help of suitable block diagrams.	6	CO1	L2
	b. Explain with examples the following 32-bit instruction of ARM processor. i. CMN ii. MLA iii. MRS iv. BIC	4	CO2	L2
2	Explain the AMBA bus protocol used for ARM processors.	10	CO2	L2
3	Write ALP program to add array of 16-bit numbers and store the result in 32-bit memory.	10	CO2	L3
4	Explain in detail the processor modes available for ARM7.	10	CO1	L2
5	Explain memory management In ARM core. Compare cache and tightly coupled memory.	10	CO1	L2

Sub:	MICROCONTROLLERS							Code:	BCS402
Date:	-03-2025	Duration:	90 mins	Max Marks:	50	Sem:	4th C	Branch:	CS(DS)

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	a. Explain the embedded software with the help of suitable block diagrams.	6	CO1	L2
	c. Explain with examples the following 32-bit instruction of ARM processor. i. CMN ii. MLA iii. MRS iv. BIC	4	CO2	L2
2	Explain the AMBA bus protocol used for ARM processors.	10	CO2	L2
3	Write ALP program to add array of 16-bit numbers and store the result in 32-bit memory.	10	CO2	L3
4	Explain in detail the processor modes available for ARM7.	10	CO1	L2
5	Explain memory management In ARM core. Compare cache and tightly coupled memory.	10	CO1	L2

Sub:	MICROCONTROLLERS							Code:	BCS402
Date:	05-06-2024	Duration:	90 mins	Max Marks:	50	Sem:	4th C	Branch:	CS(DS)
Answer Any FIVE FULL Questions									

		Marks	OBE	
			CO	RBT
6	How registers are allocated to optimize the program?	10	CO3	L2
7	Explain the syntax and usage of B, BL, BX and BLX instructions with necessary examples..	10	CO2	L2

CCI **CI** **HOD**

Sub:	MICROCONTROLLERS							Code:	BCS402
Date:	05-06-2024	Duration:	90 mins	Max Marks:	50	Sem:	4th C	Branch:	CS(DS)
Answer Any FIVE FULL Questions									

		Marks	OBE	
			CO	RBT
6	How registers are allocated to optimize the program?	10	CO3	L2
7	Explain the syntax and usage of B, BL, BX and BLX instructions with necessary examples..	10	CO2	L2

CCI **CI** **HOD**

Solution

Q.1

EMBEDDED SYSTEM SOFTWARE:

An embedded system needs software to drive it. The following Figure shows four typical software components required to control an embedded device.

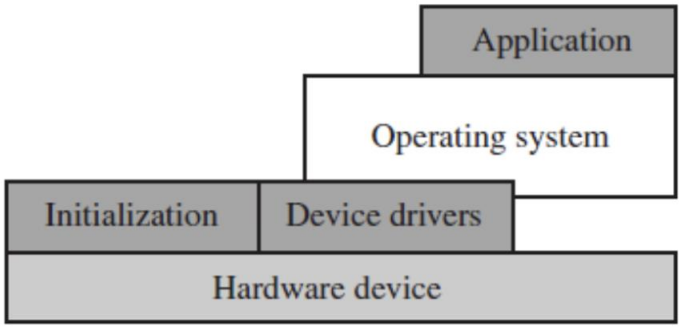

Figure: Software Abstraction Layers Executing on Hardware

- ✓ The *initialization code* is the first code executed on the board and is specific to a particular target or group of targets. It sets up the minimum parts of the board before handing control over to the operating system.
- ✓ The *operating system* provides an infrastructure to control applications and manage hardware system resources.
- ✓ The *device drivers* provide a consistent software interface to the peripherals on the hardware device.
- ✓ An *application* performs one of the tasks required for a device.

1.b

i. CMN:

- The comparison instructions are used to compare or test a register with a 32-bit value. They update the cpsr flag bits according to the result, but do not affect other registers.

CMN	compare negated	flags set as a result of $Rn + N$

ii. MLA

The multiply instructions multiply the contents of a pair of registers and depending upon the instruction, accumulate the results in another register.

MLA	multiply and accumulate	$Rd = (Rm^{*} Rs) + Rn$

iii. MRS

The MRS instruction transfers the contents of either the cpsr or spsr to general purpose register.

MRS	copy program status register to a general-purpose register	$Rd = psr$

iv. BIC

BIC	logical bit clear (AND NOT)	$Rd = Rn \,\&\, \sim N$

Q.2. Explain the AMBA bus protocol used for ARM processors.

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. A peripheral can simply be bolted onto the on-chip bus without having to redesign an interface for each 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.
- ✓ ARM has introduced ***two variations*** on the AHB bus: *Multi-layer AHB* and *AHB-Lite*.
  - 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.
- ✓ The example device shown in the above Figure has three buses:
  - an *AHB bus* for the high- performance peripherals
  - an *APB bus* for the slower peripherals
  - a third *bus for external peripherals*, proprietary to this device.

Q. 3 Write ALP program to add array of 16-bit numbers and store the result in 32-bit memory.

```
 AREA ArrayAddition, CODE, READONLY
        ENTRY
        LDR R0, =0X40000000
        LDRB R1, [R0]
        MOV R5, #0
UP      ADD R0, R0, #2
        LDRH R2, [R0]
        ADD R1, #-1
        ADD R0, R0, #2
        LDRH R3, [R0]
        ADD R1, #-1
        ADD R4, R2, R3
        ADD R5, R5, R4
        CMP R1, #0
        BNE UP
        STR R5, [R0, #4]
        MOV R0, #0X18
        LDR R1, =0X20026
        SVC #0123456
        END
```

Q.4. Explain in detail the processor modes available for ARM7.

- ✓ There are *seven processor modes* in total:
  - *six privileged modes* (abort, fast interrupt request, interrupt request, supervisor, system, and undefined)
    - The processor enters ***abort mode*** when there is a failed attempt to access memory.
    - ***Fast interrupt request*** and ***interrupt request modes*** correspond to the two interrupt levels available on the ARM processor.
    - ***Supervisor mode*** is the mode that the processor is in after reset and is generally the mode that an 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.
  - *one non-privileged mode* (user).
    - ***User mode*** is used for programs and applications.

Q.5 Explain memory management In ARM core. Compare cache and tightly coupled memory.

Memory Management:

- ✓ Embedded systems often use multiple memory devices. It is usually necessary to have a method to organize these devices and protect the system from applications trying to make inappropriate 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 providing no protection
  - a memory protection unit (MPU) providing limited protection
  - a memory management unit (MMU) providing full protection
- ✓ ***Non protected memory*** is fixed and provides very little flexibility. It is normally used for small, simple embedded systems that require no protection from rogue applications.

- ✓ ***MPU***s 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.

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 is found attached to the Von Neumann–style cores. It combines both data and instruction into a single unified cache, as shown in the following Figure.

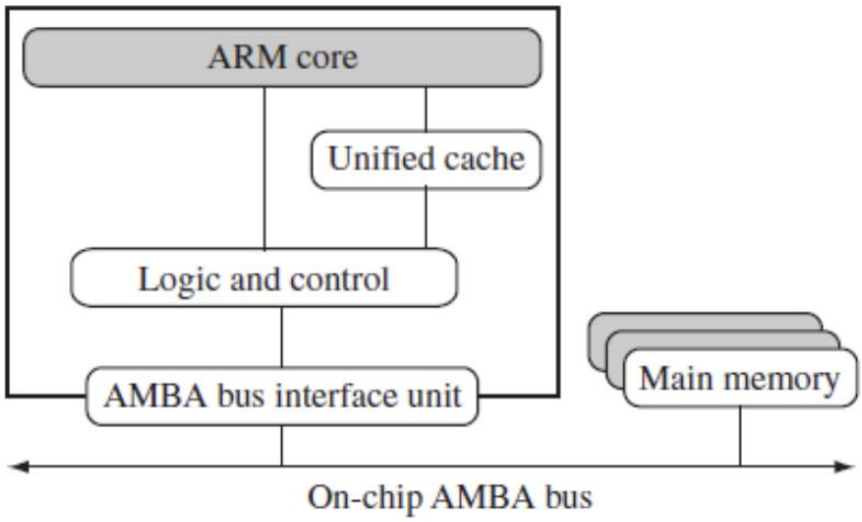

Figure: Von Neumann Architecture with Cache

- ✓ The second form, attached to the Harvard-style cores, has separate caches for data and instruction, as shown in the following Figure.

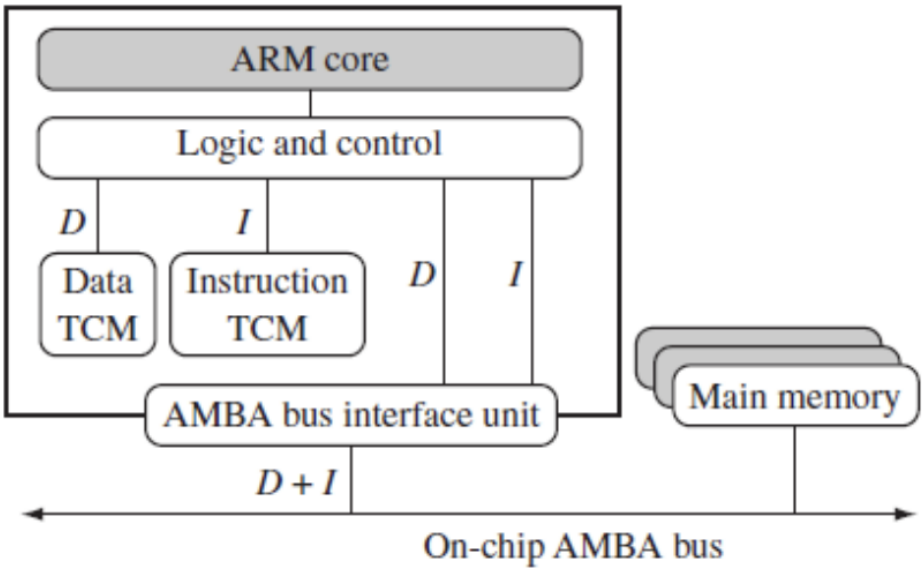

Figure: Harvard Architecture with TCMs

Q.6 How registers are allocated to optimize the program?

- The compiler attempts to allocate a processor register to each local variable you use in a C function.
- It will try to use the same register for different local variables if the use of the variables do not overlap. When there are more local variables than available registers, the compiler stores the excess variables on the processor stack. These variables are called spilled
- or swapped out variables since they are written out to memory (in a similar way virtual memory is swapped out to disk).
- Spilled variables are slow to access compared to variables allocated to registers.

To implement a function efficiently, you need to

- minimize the number of spilled variables
- ensure that the most important and frequently accessed variables are stored in registers

C compiler register usage

Register number	Alternate register names	ATPCS register usage
r0	*a1*	Argument registers. These hold the first four function arguments on a function call and the return value on a function return. A function may corrupt these registers and use them as general scratch registers within the function.
r1	*a2*	
r2	*a3*	
r3	*a4*	
r4	*v1*	General variable registers. The function must preserve the callee values of these registers.
r5	*v2*	
r6	*v3*	
r7	*v4*	
r8	*v5*	

C compiler register usage

r9	*v6 sb*	General variable register. The function must preserve the callee value of this register except when compiling for *read-write position independence* (RWPI). Then *r9* holds the *static base* address. This is the address of the read-write data.
r10	*v7 sl*	General variable register. The function must preserve the callee value of this register except when compiling with stack limit checking. Then *r10* holds the stack limit address.
r11	*v8 fp*	General variable register. The function must preserve the callee value of this register except when compiling using a frame pointer. Only old versions of *armcc* use a frame pointer.
r12	*ip*	A general scratch register that the function can corrupt. It is useful as a scratch register for function veneers or other intraprocedure call requirements.
r13	*sp*	The stack pointer, pointing to the full descending stack.
r14	*lr*	The link register. On a function call this holds the return address.
r15	*pc*	The program counter.

Q.7 Explain the syntax and usage of B, BL, BX and BLX instructions with necessary examples.

B	branch	*pc* = *label*
BL	branch with link	*pc* = *label* *lr* = address of the next instruction after the BL
BX	branch exchange	*pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1
BLX	branch exchange with link	*pc* = *label*, *T* = 1 *pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1 *lr* = address of the next instruction after the BLX