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

Department of AI-ML and AI-DS

Internal Assessment Test 1 – July 2023

Sub:	Microcontroller & Embedded Systems					Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	04-07-23	Duration:	90 min's	Max Marks:	50	Sem / Sec:	4th	OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Write a short note on RISC design philosophy. Compare Microprocessor and Microcontrollers.	[10]	CO2	L1
2.	Discuss the embedded system hardware with example of ARM based embedded device.	[10]	CO2	L2
3.	Explain the data flow model of ARM core in detail.	[10]	CO2	L2
4.	Briefly discuss the Processor modes and Instruction set of ARM controllers	[10]	CO2	L2
5.	Discuss in detail about the core extension of ARM controller.	[10]	CO2	L2
6.	With examples and syntax explain the working of compare group of Instructions of ARM.	[10]	CO2	L3
7.	With examples and syntax explain the working of following Instructions of ARM i. MVN, ii. RRX, iii. RSC, iv. BIC	[10]	CO2	L3

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

Department of AI-ML and AI-DS

Internal Assessment Test 1 – July 2023

Sub:	Microcontroller & Embedded Systems					Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	04-07-23	Duration:	90 min's	Max Marks:	50	Sem / Sec:	4th	OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Write a short note on RISC design philosophy. Compare Microprocessor and Microcontrollers.	[10]	CO2	L1
2.	Discuss the embedded system hardware with example of ARM based embedded device.	[10]	CO2	L2
3.	Explain the data flow model of ARM core in detail.	[10]	CO2	L2
4.	Briefly discuss the Processor modes and Instruction set of ARM controllers	[10]	CO2	L2
5.	Discuss in detail about the core extension of ARM controller.	[10]	CO2	L2
6.	With examples and syntax explain the working of compare group of Instructions of ARM.	[10]	CO2	L3
7.	With examples and syntax explain the working of following Instructions of ARM i. MVN, ii. RRX, iii. RSC, iv. BIC	[10]	CO2	L3

Internal Assessment Test 1– July. 2023

Sub:	Microcontroller & Embedded Systems					Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	04-07-23	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	4th		OBE

	Scheme and Solution	MARKS	CO	RBT
1			CO2	L1

RISC design philosophy is

- Aimed at simple but powerful instructions that execute within a single cycle at a high clock speed.
- Concentrates on reducing the complexity of instructions performed by the hardware.
- Provides greater flexibility and intelligence in software rather than hardware.

05

The RISC philosophy is implemented with four major design rules:

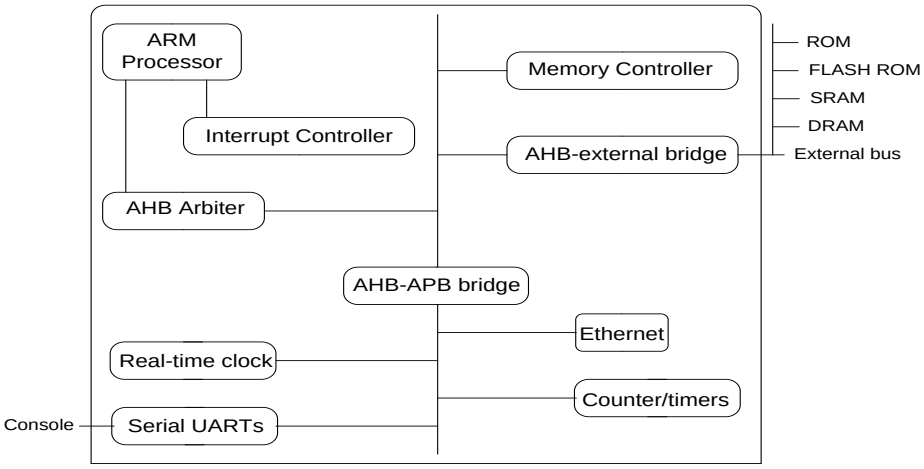
- Instructions: RISC has a reduced number of instruction classes. These classes provide simple operations so that each is executed in a single cycle. Each instruction is a fixed length to allow the pipeline to fetch future instructions before decoding the current instruction.
- Pipeline: The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines.
- Register: RISC machines have a large general-purpose register set. Any register can contain either data or an address.
- 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.

- These design rules allow a RISC processor to be simpler, and thus the core can operate at higher clock speed.
- Figure below shows the major difference between CISC and RISC processors, CISC emphasizes on hardware complexity, whereas RISC emphasizes on compiler complexity.

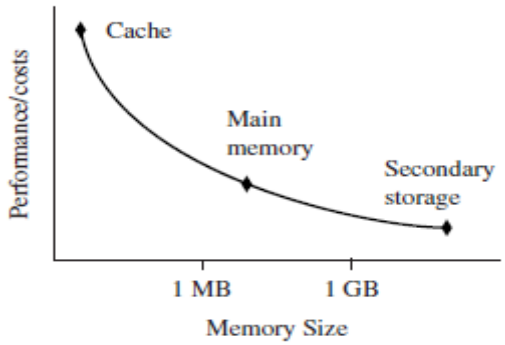
05

Microprocessor	Micro Controller
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.

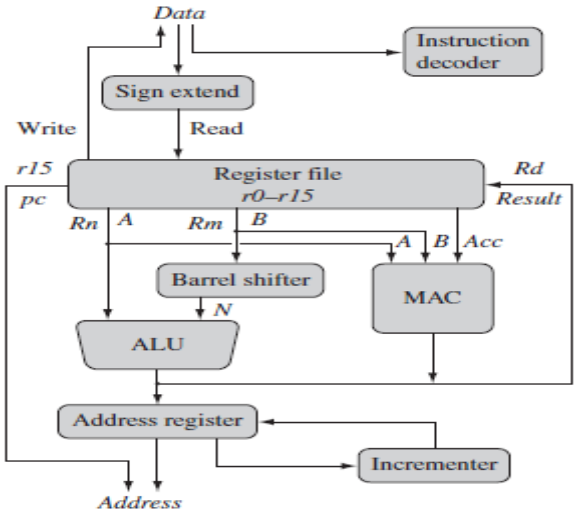
2	Figure shown below shows a typical embedded device based on ARM core. Each box represents a feature or function.	10	CO2	L2

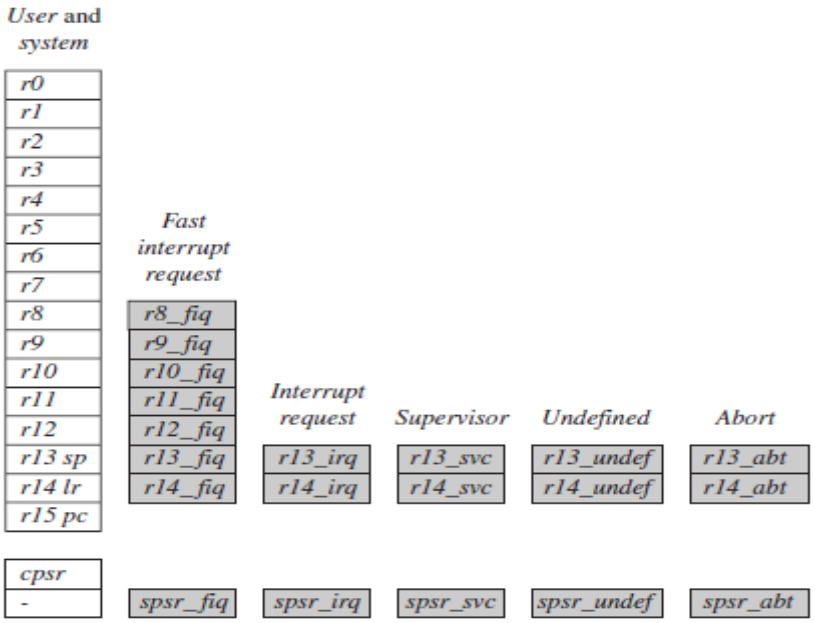


- ARM processor based embedded system hardware can be separated into the following four main hardware components:
  - **The ARM processor:** The ARM processor controls the embedded device. Different versions of the ARM processor are available to suits the desired operating characteristics.
  - **Controllers:** Controllers coordinate important blocks of the system. Two commonly found controllers are memory controller and interrupt controller.
  - **Peripherals:** The peripherals provide all the input-output capability external to the chip and responsible for the uniqueness of the embedded device.
  - **Bus:** A bus is used to communicate between different parts of the device.
- **ARM Bus Technology**
  - 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.
  - There are two different classes of devices attached to the bus.
    - The ARM processor core is a **bus master**—a logical device capable of initiating a data transfer with another device across the same bus.
    - Peripherals tend to be **bus slaves**—logical devices capable only of responding to a transfer request from a bus master device.
- **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).
  - AHB provides higher data throughput than ASB because it is based on a centralized multiplexed bus scheme rather than the ASB bidirectional bus design.
- **MEMORY**
  - An embedded system has to have some form of memory to store and execute code.
  - Figure below 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.



- **PERIPHERALS**
  - Embedded systems that interact with the outside world need some form of peripheral device.
  - Controllers are specialized peripherals that implement higher levels of functionality within the embedded system.
  - **Memory controller:** Memory controllers connect different types of memory to the processor bus.
  - **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.

3	 • An ARM core as functional units connected by data buses, as shown in Figure1, where, the arrows represent the flow of data, the lines represent the buses, and the boxes represent either an operation unit or a storage area. • The instruction decoder translates instructions before they are executed. • The ARM processor, like all RISC processors, uses a load - store architecture. • Load instructions copy data from memory to registers, and conversely the store instructions copy data from registers to memory. • There are no data processing instructions that directly manipulate data in memory. • ARM instructions typically have two source registers, Rn and Rm, and a single 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. • After passing through the functional units, the result in Rd is written back	[03] [07]	CO2	L2

	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			
4	• 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. 1. The processor enter **abort mode** when there is a failure to attempt to access memory. 2. **Fast interrupt request and interrupt request modes** correspond to the two interrupt levels available on the ARM processor. 3. **Supervisor mode** is the mode that the processor is in after reset and is generally the mode that an operating system kernel operates in. 4. **System mode** is a special version of user mode that allows full read-write access to the cpsr. 5. **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.  State of the core determines which instruction set is being executed. Three instruction sets: ARM, Thumb and Jazelle. ARM instruction set is only active when the processor is in ARM state. Thumb instruction set is only active in Thumb state. Intermingle of sequential ARM, Thumb, and Jazelle instructions not allowed Jazelle J and Thumb T bits in the CPSR reflects the state of the processor. If both J and T bits are 0, the processor is in ARM state and executes ARM instructions. (During power on) If the T bit is 1, then the processor is in Thumb state. To change states the core executes a specialized branch instruction. Jazelle executes 8-bit instructions. It is hybrid mix of software and hardware designed to speed up the execution of Java byte codes. It requires the Jazelle technology plus a specially modified version of the Java virtual machine. Note that the hardware portion of Jazelle only supports a subset of the Java bytecodes; the rest are emulated in software.	[05] [05]	CO2	L2

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 $+pc$

[a] See Section 2.2.6.

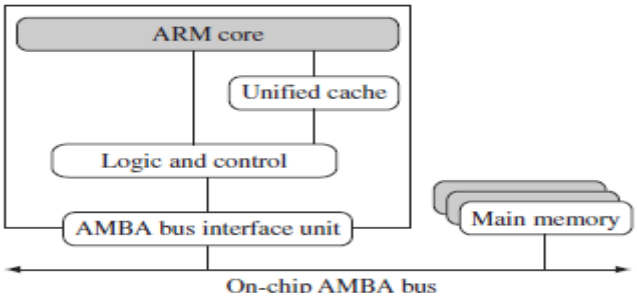
Jazelle instruction set features.

	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.

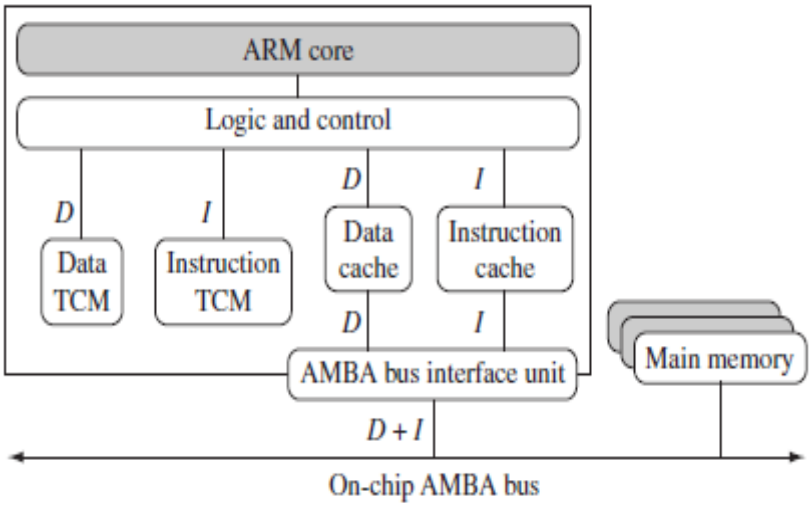
5 There are three core extensions wrap around ARM processor: cache and tightly coupled memory, memory management and the coprocessor interface. [10] CO2 L2

1. Cache and tightly coupled memory: The cache is a block of fast memory placed between main memory and the core. 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 found attached to the Von Neumann-style cores. It combines both data and instruction into a single unified cache as shown in the figure 1 below.



- A cache provides an overall increase in performance but will not give predictable execution.
- But for real-time systems it is paramount that code execution is deterministic.
- 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.
- 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 as shown in figure



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 supports multitasking.

3. **Coprocessors:**

- 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.

6

- 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.
- After the bits have been set, the information can be used to change program flow by using conditional execution.

[10] CO2 L3

Syntax: <instruction> {<cond>} Rn, N

CMN	compare negated	flags set as a result of $Rn + N$
CMP	compare	flags set as a result of $Rn - N$
TEQ	test for equality of two 32-bit values	flags set as a result of $Rn \wedge N$
TST	test bits of a 32-bit value	flags set as a result of $Rn \& N$

- Example shown below for CMP instruction, both r0 and r1 are equal before the execution of the instruction. The value of the z flag prior to the execution is 0 and after the execution z flag changes to 1 (upper case of Z).

```
PRE     cpsr = nzcvqiFt_USER
        r0 = 4
        r9 = 4

        CMP   r0, r9

POST    cpsr = nZcvqiFt_USER
```

- The CMP is effectively a subtract instruction with the result discarded;
- Similarly the TST instruction is a logical AND operation and TEQ is a logical XOR operation. For each, the results are discarded but the condition bits are updated in the cpsr.

7 **MOVE INSTRUCTIONS:** 4*2.5 CO2 L3

- It copies N into a destination register Rd, where N is a register or immediate value. This instruction is useful for setting initial values and transferring data between registers.

Syntax: <instruction> {<cond>} {S} Rd, N

MOV	Move a 32-bit value into a register	$Rd = N$
MVN	move the NOT of the 32-bit value into a register	$Rd = \sim N$

- In the example shown below, the MOV instruction takes the contents of register r5 and copies them into register r7.

```
PRE     r5 = 5
        r7 = 8
        MOV     r7, r5     ; let r7 = r5
POST    r5 = 5
        r7 = 5
```

- Data processing instructions are processed within the arithmetic and logic unit (ALU).
- A unique and powerful feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before it enters the ALU.

Mnemonic	Description	Shift	Result	Shift amount y
LSL	logical shift left	*x*LSL *y*	$x \ll y$	#0–31 or *Rs*
LSR	logical shift right	*x*LSR *y*	(unsigned)$x \gg y$	#1–32 or *Rs*
ASR	arithmetic right shift	*x*ASR *y*	(signed)$x \gg y$	#1–32 or *Rs*
ROR	rotate right	*x*ROR *y*	((unsigned)$x \gg y$) | ($x \ll (32 - y)$)	#1–31 or *Rs*
RRX	rotate right extended	*x*RRX	(c flag $\ll 31$) | ((unsigned)$x \gg 1$)	none

Note: *x* represents the register being shifted and *y* represents the shift amount.

- The arithmetic instructions implement addition and subtraction of 32-bit signed and unsigned values.

Syntax: <instruction>{<cond>} {S} Rd, Rn, N

ADC	add two 32-bit values and carry	$Rd = Rn + N +$ carry
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn -$!(carry flag)
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N -$!(carry flag)
SUB	subtract two 32-bit values	$Rd = Rn - N$

- In the following example, subtract instruction subtracts a value stored in register r2 from a value stored in the register r1. The result is stored in register r0.

```
PRE     r0 = 0x00000000
        r1 = 0x00000002
        r2 = 0x00000001

        SUB r0, r1, r2

POST    r0 = 0x00000001
```

- Logical instructions perform bitwise operations on the two source registers.

Syntax: <instruction> {<cond>} {S} Rd, Rn, N

AND	logical bitwise AND of two 32-bit values	$Rd = Rn \& N$
ORR	logical bitwise OR of two 32-bit values	$Rd = Rn \mid N$
EOR	logical exclusive OR of two 32-bit values	$Rd = Rn \wedge N$
BIC	logical bit clear (AND NOT)	$Rd = Rn \& \sim N$