USN

Internal Assessment Test I – JULY 2022

Sub:	MICROCONTROLLER AND EMBEDDED SYSTEM Sub Code:		18CS44	Branch :	CSE	
Date:	09/07/22	Duration:	90 mins Max Marks: 50 Sem / Sec:	IV Sem A/B/C		OBE
			Answer any FIVE FULL Questions		ARK	C O	RBT

1	**Explain the architecture of a typical embedded device based in ARM core, with a neat diagram.**	[4+6]	CO1	L1

Answer:

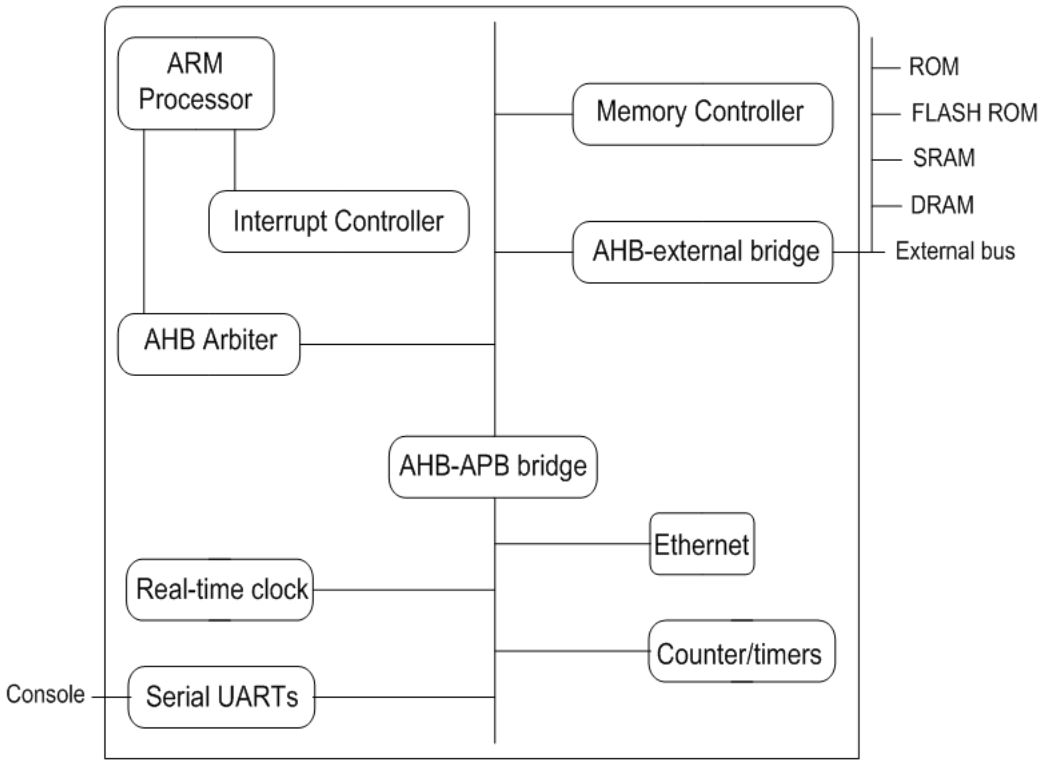


Figure shown below shows a typical embedded device based on ARM core. Each box represents a feature or function.

- ARM processor based embedded system hardware can be separated into the following four main hardware components:
  - o The ARM processor: The ARM processor controls the embedded device. Different versions of the ARM processor are available to suits the desired operating characteristics.
  - o Controllers: Controllers coordinate important blocks of the system. Two commonly found controllers are memory controller and interrupt controller.
  - o Peripherals: The peripherals provide all the input-output capability external to the chip and responsible for the uniqueness of the embedded device.
  - o Bus: A bus is used to communicate between different parts of the device.
- ARM Bus Technology
  - o 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.
  - o 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
  - o 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.
  - o The first AMBA buses introduced were the ARM System Bus (ASB) and the ARM Peripheral Bus (APB).
  - o Later ARM introduced another bus design, called the ARM High Performance Bus (AHB).
  - o 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
  - o An embedded system has to have some form of memory to store and execute code.
  - o 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.
  - o Generally the closer memory is to the processor core, the more it costs and the smaller its capacity.
- PERIPHERALS
  - o Embedded systems that interact with the outside world need some form of peripheral device.
  - o Controllers are specialized peripherals that implement higher levels of functionality within the embedded system.
  - o Memory controller: Memory controllers connect different types of memory to the processor bus.
  - o 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.

2	**a) Describe the different operating modes of ARM processors.**	[6+4]	CO1	L2,L3

Answer:

- 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.

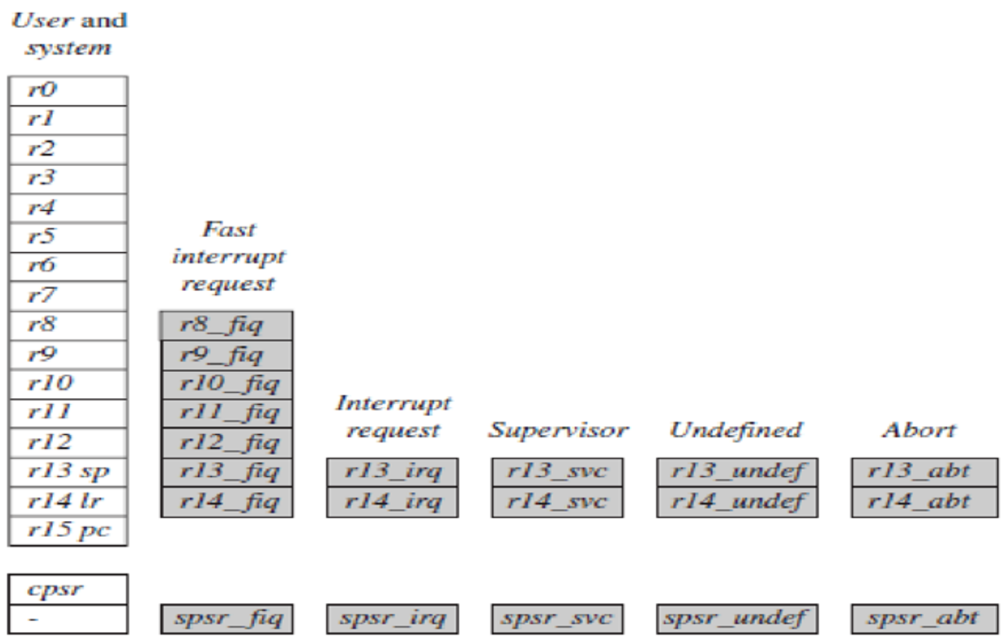
b) Illustrate four major rules of RISC design.

Answer:

- The RISC philosophy is implemented with four major design rules:
  - o **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.
  - o **Pipeline**: The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines.
  - o **Register**: RISC machines have a large general-purpose register set. Any register can contain either data or an address.
  - o **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.

3 a. **Explain Banked Registers of ARM7 MicroController with neat diagram.** [6+4] CO1, CO2 L2,L3

Answer:



- Figure below shows all 37 registers in the register file.
- Of these, 20 registers are hidden from a program at different times. These registers are called banked registers.
- 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.
- Every processor mode except user mode can change mode by writing directly to the mode bits of the cpsr.
- All privileged modes except system mode have a set of associated banked registers that are subset of the main 16 registers.
- If the processor mode is changed, a banked register from the new mode will replace an existing register.
- The processor mode can be changed by a program that writes directly to the cpsr when the processor core is in privilege mode.
- The following exception and interrupts causes a mode change: reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort and undefined instructions.
- Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location.
- Following figure 2 illustrates the happening when an interrupt forces a mode change.
- The figure 2 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.
- r14_irq contains the return address and r13_irq contains the stack pointer for interrupt request mode.
- The saved program status register (spsr), which stores the previous mode's cpsr.

b. If r5 = 5, r3 = 0, r7 = 8 and using the following instruction, write values of r5, r7 after execution ADD r3, r7, r5, LSL #2.

Answer:

4	**Consider the pre-scenario**	[2.5X 4]	CO 2	L4

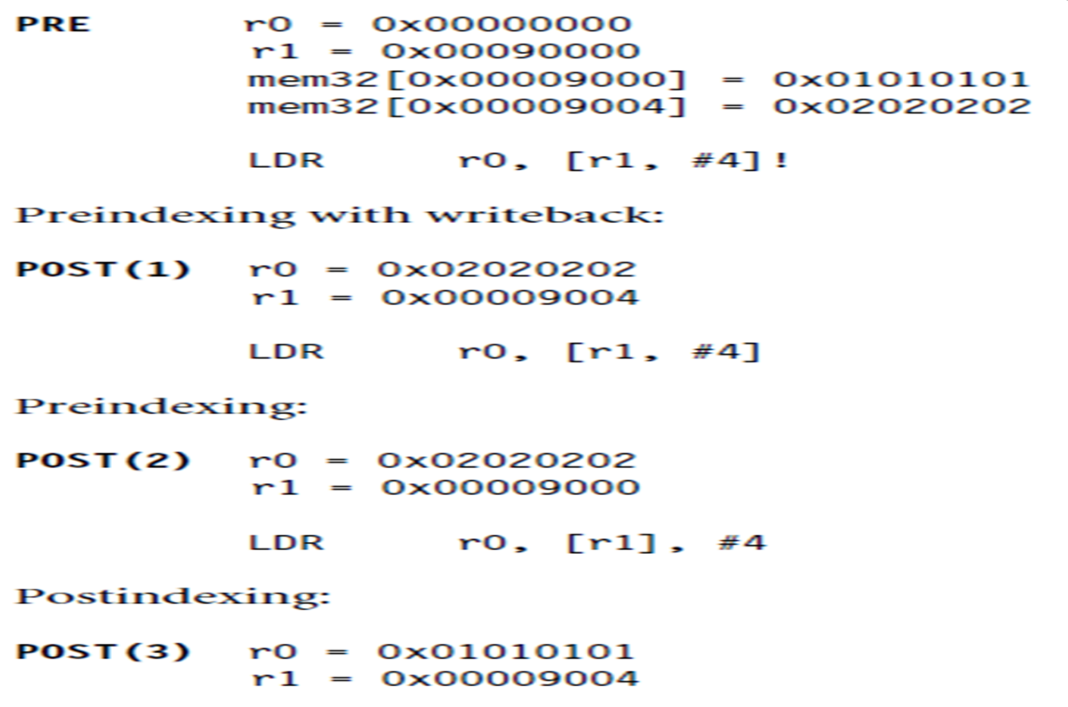
r0 = 0x00000000,
r1 = 0x00009000,
mem32[0x00009000] = 0x01010101
mem32[0x00009004] = 0x02020202

Write post-scenario with respect to execution of the following instructions (i.e., content of r0and r1)

a) LDR r0, [r1]
b) LDR r0, [r1, #4] !
c) LDR r0, [r1, #4]
d) LDR r0, [r1], #4

Answer:
LDR r0,[r1]

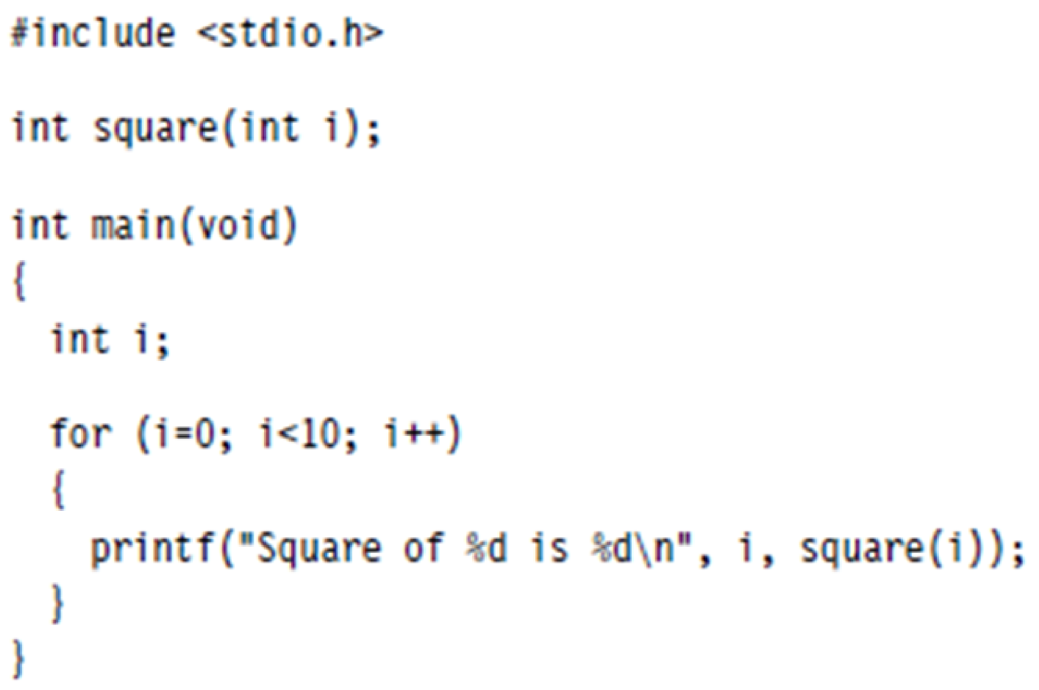
r0= **0x01010101**
r1= 0x**00009000**



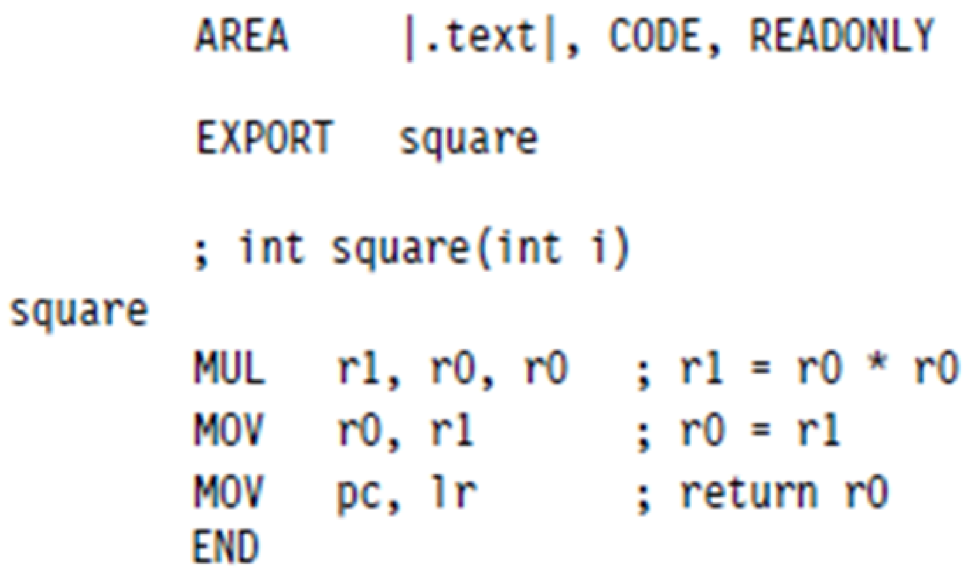
```
PRE        r0  =  0x00000000
            r1  =  0x00090000
           mem32[0x00009000]  =  0x01010101
           mem32[0x00009004]  =  0x02020202

           LDR      r0, [r1, #4]!

Preindexing with writeback:

POST(1)    r0  =  0x02020202
            r1  =  0x00009004

           LDR      r0, [r1, #4]

Preindexing:

POST(2)    r0  =  0x02020202
            r1  =  0x00009000

           LDR      r0, [r1], #4

Postindexing:

POST(3)    r0  =  0x01010101
            r1  =  0x00009004
```

5	**(a) Write a short note on** **i) Register allocation ii) Allocation variables to register numbers.** Answer: **Pre Operation:** **R0= 0X00000000, R1= 0X 80000004 and CPSR= nzcvqiFt.** **MOV R0, R1, LSL #1** **Post Operation:** **R0 = R1 * 2** **R0 = 0x00000008, R1 = 0x80000004** **(b) Show the post condition when MOVs instruction shifts register R1 left by one bit and result is stored in R0. Where R0= 0X00000000, R1= 0X 80000004 and CPSR= nzcvqiFt.** Answer: **Register Allocation** · **can use 14 of the 16 visible ARM registers to hold general-purpose data. The other two registers are the stack pointer r13 and the program counter r15.** · **For a function to be ATPCS compliant it must preserve the callee values of registers r4 to r11. ATPCS also specifies that the stack should be eight-byte aligned; therefore you must preserve this alignment if calling subroutines.** · **Use the following template for optimized assembly routines requiring many registers:** · **Our only purpose in stacking r12 is to keep the stack eight-byte aligned.** · **In this section we look at how best to allocate variables to register numbers for register intensive tasks, how to use more than 14 local variables, and how to make the best use of** **the 14 available registers.** **Allocating Variables to Register Numbers** • **When you write an assembly routine, it is best to start by using names for the variables, rather than explicit register numbers. This allows you to change the allocation of variables to register numbers easily.** • **You can even use different register names for the same physical register number when their use doesn't overlap. Register names increase the clarity and readability of optimized code.** • **However, there are several cases where the physical number of the register is important:** **i. Argument registers. The ATPCS convention defines that the first four arguments to a function are placed in registers r0 to r3. Further arguments are placed on the stack. The return value must be placed in r0.**	[6+4]	CO2	L1,L3

	ii. Registers used in a load or store multiple. Load and store multiple instructions LDM and STM operate on a list of registers in order of ascending register number. If r0 and r1 appear in the register list, then the processor will always load or store r0 using a lower address than r1 and so on. **iii. Load and store double word. The LDRD and STRD instructions introduced in ARMv5E operate on a pair of registers with sequential register numbers, Rd and Rd + 1. Furthermore, Rd must be an even register number.** **iv. There are several possible ways we can proceed when we run out of registers:** · **Reduce the number of registers we require by performing fewer operations in each loop.** · **Use the stack to store the least-used values to free up more registers.** **Alter the code implementation to free up more registers.**			

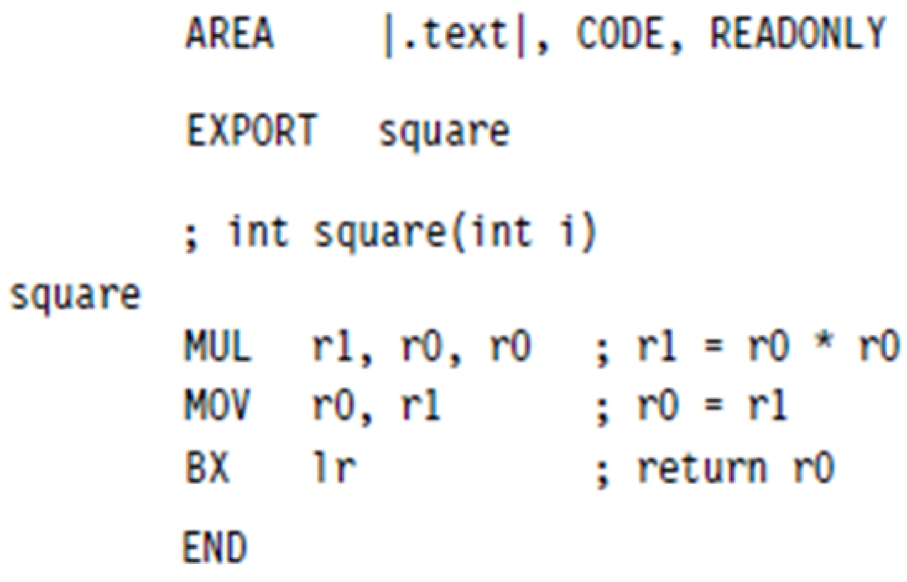
6	**Let's consider an array of ten (10) numbers between 0 to 9. The user wants to print square of the array elements. Write down an optimized high level code (preferably in C) and the corresponding ALP of the square function for the this scenario.**
	
	Answer:
	
	
	`#include <stdio.h>`
	
	`int square(int i);`
	
	`int main(void)`
	`{`
	`  int i;`
	
	`  for (i=0; i<10; i++)`
	`  {`
	`    printf("Square of %d is %d\n", i, square(i));`
	`  }`
	`}`
	
	int square(int i)
	
	{
	
	return i*i;
	
	}
		[3+2+ 2+3]	CO2	L4

- The AREA directive names the area or code section that the code lives in. If you use nonalphanumeric characters in a symbol or area name, then enclose the name in vertical bars.
- The EXPORT directive makes the symbol square available for external linking.
- The input argument is passed in register r0, and the return value is returned in register r0.
- The multiply instruction has a restriction that the destination register must not be the same as the first argument register. Therefore we place the multiply result into r1 and move this to r0.
- The END directive marks the end of the assembly file. Comments follow a semicolon.



```
        AREA    |.text|, CODE, READONLY

        EXPORT  square

        ; int square(int i)
square
        MUL     r1, r0, r0   ; r1 = r0 * r0
        MOV     r0, r1       ; r0 = r1
        MOV     pc, lr       ; return r0
        END
```

Thumb Code:



```
        AREA    |.text|, CODE, READONLY

        EXPORT  square

        ; int square(int i)
square
        MUL     r1, r0, r0   ; r1 = r0 * r0
        MOV     r0, r1       ; r0 = r1
        BX      lr           ; return r0

        END
```