USN | | | | | | | | | |

Department of AI-ML and AI-DS

Internal Assessment Test 2 – August 2023

Sub:	Microcontroller & Embedded Systems				Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	08-08-23	Duration:	90 min's	Max Marks: 50	Sem / Sec:	4th		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1.	Explain in detail branch instructions of ARM processor.	[10]	CO2	L2
2.	Discuss the Load Multiple register instructions of ARM and Discuss the Stack implementation with example.	[10]	CO2	L2
3.	Write an ALP to sort an array in ascending order using bubble sort for ARM 7 controller with appropriate comments.	[10]	CO2	L3
4.	Discuss the C data types of embedded system and Local variable types with example program of "Checksum" with arguments passed and returned of "short" type.	[10]	CO1	L3
5.	Discuss the classification and purpose of embedded system in detail.	[10]	CO3	L2
6.	Differentiate between General Computing and Embedded system with examples.	[10]	CO3	L2
7.	Differentiate between i. RISC and CISC ii. Harvard and Von-Neumann architectures	[10]	CO3	L2

USN | | | | | | | | | |

Department of AI-ML and AI-DS

Internal Assessment Test 2 – August 2023

Sub:	Microcontroller & Embedded Systems				Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	08-08-23	Duration:	90 min's	Max Marks: 50	Sem / Sec:	4th		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1.	Explain in detail branch instructions of ARM processor.	[10]	CO2	L2
2.	Discuss the Load Multiple register instructions of ARM and Discuss the Stack implementation with example.	[10]	CO2	L2
3.	Write an ALP to sort an array in ascending order using bubble sort for ARM 7 controller with appropriate comments.	[10]	CO2	L3
4.	Discuss the C data types of embedded system and Local variable types with example program of "Checksum" with arguments passed and returned of "short" type.	[10]	CO1	L3
5.	Discuss the classification and purpose of embedded system in detail.	[10]	CO3	L2
6.	Differentiate between General Computing and Embedded system with examples.	[10]	CO3	L2
7.	Differentiate between i. RISC and CISC ii. Harvard and Von-Neumann architectures	[10]	CO3	L2

CI: **CCI:** **HOD:**

Internal Assessment Test 2– August. 2023

Sub:	Microcontroller & Embedded Systems					Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	08-08-23	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	4th		OBE

	Scheme and Solution	MARKS	CO	RBT
1	(see below)		CO2	L2
2	(see below)	10	CO2	L2

1

- A branch instruction changes the flow of execution or is used to call a routine.
- This type of instruction allows programs to have subroutines, if-then-else structures, and loops.
- The change of execution flow forces the program counter (pc) to point to a new address.

```
Syntax: B{<cond>} label
        BL{<cond>} label
        BX{<cond>} Rm
        BLX{<cond>} label | Rm
```

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 BL

- T refers to the Thumb bit in the cpsr.
- When instruction set T, the ARM switches to Thumb state.
- The example shown below is a forward branch. The forward branch skips three instructions.

```
        B       forward
        ADD     r1, r2, #4
        ADD     r0, r6, #2
        ADD     r3, r7, #4
forward
        SUB     r1, r2, #4
```

- The branch with link (BL) instruction changes the execution flow in addition overwrites the link register lr with a return address. The example shows below a fragment of code that branches to a subroutine using the BL instruction.

```
        BL      subroutine      ; branch to subroutine
        CMP     r1, #5          ; compare r1 with 5
        MOVEQ   r1, #0          ; if (r1==5) then r1 = 0
        :
subroutine
        <subroutine code>
        MOV     pc, lr          ; return by moving pc = lr
```

- **The branch exchange (BX)** instruction uses an absolute address stored in register Rm. It is primarily used to branch to and from Thumb code. The T bit in the cpsr is updated by the least significant bit of the branch register.
- Similarly, **branch exchange with link (BLX)** instruction updates the T bit of the cpsr with the least significant bit and additionally sets the link register with the return address.

2

- Load-store multiple instructions can transfer multiple registers between memory and the processor in a single instruction. The transfer occurs from a base address register Rn pointing into memory.

- Multiple-register transfer instructions are more efficient from single-register transfers for moving blocks of data around memory and saving and restoring context and stacks.

Syntax: <LDM|STM>{<cond>}<addressing mode> Rn{!},<registers>{^}

LDM	load multiple registers	$\{Rd\}^{*N}$ <- mem32[start address + 4*N] optional Rn
STM	save multiple registers	$\{Rd\}^{*N}$ -> mem32[start address + 4*N] optional Rn

Addressing mode for load-store multiple instructions

- Table below shows the different addressing modes for the load-store multiple instructions.

Addressing mode	Description
IA	increment after
IB	increment before
DA	decrement after
DB	decrement before

Example:

```
         mem32[0x8001c] =0x04
PRE      mem32[0x80018] = 0x03
         mem32[0x80014] = 0x02
         mem32[0x80010] = 0x01
         r0 = 0x00080010
         r1 = 0x00000000
         r2 = 0x00000000
         r3 = 0x00000000

         LDMIA   r0!, {r1-r3}

POST     r0 = 0x0008001c
         r1 = 0x00000001
         r2 = 0x00000002
         r3 = 0x00000003
```

- If LDMIA is replaced with LDMIB post execution the content of registers is shown below

```
r3 = 0x00000004
r2 = 0x00000003
r1 = 0x00000002
r0 = 0x8001c
```

STACK OPERATIONS

- The ARM architecture uses the load-store multiple instructions to carry out stack operations.
- The pop operation (removing data from a stack) uses a load multiple instruction; similarly, the push operation (placing data onto the stack) uses a store multiple

instruction.

- When you use a **full stack (F)**, the stack pointer sp points to an address that is the last used or full location.
- In contrast, if you use an **empty stack (E)** the sp points to an address that is the first unused or empty location.
- A stack is either ascending (A) or descending (D). Ascending stacks grow towards higher memory addresses; in contrast, descending stacks grow towards lower memory addresses.
- Addressing modes for stack operation

Addressing mode	Description
FA	full ascending
FD	full descending
EA	empty ascending
ED	empty descending

- The LDMFD and STMFD instructions provide the pop and push functions, respectively.
- Example1: With full descending

```
PRE     r1 = 0x00000002
        r4 = 0x00000003
        sp = 0x00080014

        STMFD   sp!, {r1,r4}
POST    r1 = 0x00000002
        r4 = 0x00000003
        sp = 0x0008000c
```

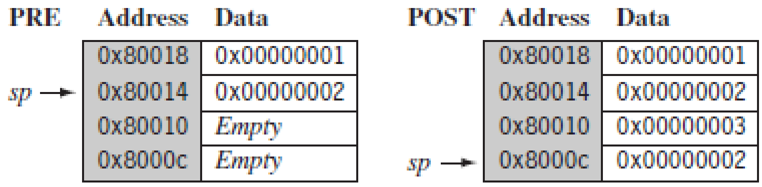

Figure: STMFD instruction full stack push operation.

Example 2: With empty descending

```
PRE     r1 = 0x00000002
        r4 = 0x00000003
        sp = 0x00080010

        STMED   sp!, {r1,r4}

POST    r1 = 0x00000002
        r4 = 0x00000003
        sp = 0x00080008
```

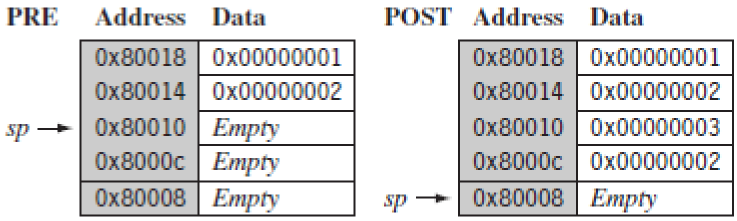

Figure: STMED instruction empty stack push operation.

3 [6+4] CO2 L3

```
        AREA CODE1,CODE,READONLY
        ENTRY
        LDR R0,=INPUT
        LDMIA R0,{R1-R10} ; load the registers r1 to r10 from memory r0
        holding starting address
        LDR R0,=SORT
        STMIA R0,{R1-R10} ;store the registers r1 to r10 to memory r0
        holding starting address
        MOV R2,#9 ; R2=10 NUMBER COUNT i=9
OUTER LDR R0,=SORT ; R0 load the address of variable input
        MOV R3,R2 ; inner loop count j=i
REPEAT LDR R4,[R0],#4
        ; load element from memory and increment memory pointer by 4
        LDR R5,[R0] ; load next element from memory
        CMP R4,R5 ; compare two consecutive elements
        BLE SKIP
        ; if first element is less than second element than jump to label skip
        SWP R5,R4,[R0] ; otherwise swap the elements
        SUB R0,R0,#4
        SWP R4,R5,[R0]
        ADD R0,R0,#4
SKIP SUB R3,#1 ; decrease inner loop count j=j-1
        CMP R3,#0 ; compare j==0
        BNE REPEAT ; if j!= 0 repeat the inner loop
        SUB R2,#1 ; decrease outer loop count i=i-1
        CMP R2,#0 ; compare i==0
        BNE OUTER ; if i!=0 repeat the outer loop
        STOP B STOP
        INPUT DCD 0X11
        DCD 0X88
        DCD 0X33
        DCD 0X77
        DCD 0XAA
        DCD 0X44
        DCD 0X99
        DCD 0X66
        DCD 0X22
        DCD 0X55
        AREA DATA1,DATA,READWRITE
        SORT SPACE 40
        END
```

4 [10] CO1 L3

Compilers armcc and gcc use the datatype mappings in Table for an ARM target. The exceptional case for type char is worth noting as it can cause problems when you are porting code from another processor architecture. A common example is using a char type variable i as a loop counter, with loop continuation condition i 0. As i is unsigned for the ARM compilers, the loop will never terminate. Fortunately armcc produces a warning in this situation: unsigned comparison with 0. Compilers also provide an override switch to make char signed. For example, the command line option -fsigned-char will make char signed on gcc. The command line option -zc will have the same effect with armcc.

	C compiler datatype mappings.
	
	C Data Type / Implementation
	char — unsigned 8-bit byte
	short — signed 16-bit halfword
	int — signed 32-bit word
	long — signed 32-bit word
	long long — signed 64-bit double word
	
	the data packet contains 16-bit values and we need a 16-bit checksum. It is tempting to write the following C code:
	short checksum_v3(short *data)
	{
	unsigned int i; short sum = 0;
	
	for (i = 0; i < 64; i++)
	{
	sum = (short)(sum + data[i]);
	}
	return sum;
	}
	
	You may wonder why the for loop body doesn't contain the code
	
	sum += data[i];
	
	With armcc this code will produce a warning if you enable implicit narrowing cast warnings using the compiler switch -W+ n. The expression sum + data[i] is an integer and so can only be assigned to a short using an (implicit or explicit) narrowing cast. As you can see in the following assembly output, the compiler must insert extra instructions to implement the narrowing cast:
	
	checksum_v3
	MOV r2,r0 ; r2 = data
	MOV r0,#0 ; sum = 0
	MOV r1,#0 ; i = 0
	checksum_v3_loop
	ADD r3,r2,r1,LSL #1 ; r3 = &data[i]
	LDRH r3,[r3,#0] ; r3 = data[i]
	ADD r1,r1,#1 ; i++
	CMP r1,#0x40 ; compare i, 64
	ADD r0,r3,r0 ; r0 = sum + r3
	MOV r0,r0,LSL #16
	MOV r0,r0,ASR #16 ; sum = (short)r0
	BCC checksum_v3_loop ; if (i<64) goto loop
	MOV pc,r14 ; return sum			

5	The classification of embedded system is based on following criteria's: • On generation • On complexity & performance • On deterministic behavior • On triggering On generation: 1. First generation (1G): • Built around 8bit microprocessor & microcontroller. • Simple in hardware circuit & firmware developed. • Examples: Digital telephone keypads. 2. Second generation (2G): • Built around 16-bit µp & 8-bit µc. • They are more complex & powerful than 1G µp & µc. • Examples: SCADA systems	[4+6]	CO3	L2

3. Third generation (3G):
• Built around 32-bit µp& 16-bit µc.
• Concepts like Digital Signal Processors (DSPs), Application Specific Integrated Circuits(ASICs) evolved. Examples: Robotics, Media, etc.
4. Fourth generation:
• Built around 64-bit µp & 32-bit µc.
• The concept of System on Chips (SoC), Multicore Processors evolved.
• Highly complex & very powerful. Examples: Smart Phones.
 On complexity & performance:
1. Small-scale:
 Simple in application need
 Performance not time-critical.
 Built around low performance& low cost 8 or 16 bit µp/µc. Example: an electronic toy
2. Medium-scale:
 Slightly complex in hardware & firmware requirement.
 Built around medium performance & low cost 16 or 32 bit µp/µc.
 Usually contain operating system.
 Examples: Industrial machines.
3. Large-scale:
 Highly complex hardware & firmware.
 Built around 32 or 64 bit RISC µp/µc or PLDs or Multicore-Processors.
 Response is time-critical.
 Examples: Mission critical applications.

 On deterministic behavior:
 This classification is applicable for "Real Time" systems.
 The task execution behavior for an embedded system may be deterministic or non- deterministic.
 Based on execution behavior Real Time embedded systems are divided into Hard and Soft.
 On triggering
 Embedded systems which are "Reactive" in nature can be based on triggering.
 Reactive systems can be:
 Event triggered
 Time triggered

PURPOSE OF EMBEDDED SYSTEM

1. Data Collection/Storage/Representation
 Embedded system designed for the purpose of data collection performs acquisition of data from the external world.
 Data collection is usually done for storage, analysis, manipulation and transmission.
 Data can be analog or digital.
 Embedded systems with analog data capturing techniques collect data directly in the form of analog signal whereas embedded systems with digital data collection mechanism converts the analog signal to the digital signal using analog to digital converters.
 If the data is digital it can be directly captured by digital embedded system.
 A digital camera is a typical example of an embedded System with data collection/storage/representation of data.
 Images are captured and the captured image may be stored within the memory of the camera. The captured image can also be presented to the user through a graphic LCD unit.
2. Data communication
 Embedded data communication systems are deployed inapplications from complex satellite communication to simple home networking systems.
 The transmission of data is achieved either by a wire-lin medium or by a wire-less medium. Data can either be transmitted by analog means or by digital means.
 Wireless modules-Bluetooth, Wi-Fi.
 Wire-line modules-USB, TCP/IP.
 Network hubs, routers, switches are examples of dedicated data transmission embedded systems.

3. Data signal processing

- Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, audio video codec, transmission applications etc.
- A digital hearing aid is a typical example of an embedded system employing data processing. Digital hearing aid improves the hearing capacity of hearing impaired person.

4. Monitoring

- All embedded products coming under the medical domain are with monitoring functions. Electro cardiogram machine is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.
- Other examples with monitoring function are digital CRO, digital multi-meters, and logic analyzers.

5. Control

- A system with control functionality contains both sensors and actuators Sensors are connected to the input port for capturing the changes in environmental variable and the actuators connected to the output port are controlled according to the changes in the input variable.
- Air conditioner system used to control the room temperature to a specified limit is a typical example for CONTROL purpose.

6. Application specific user interface

- Buttons, switches, keypad, lights, bells, display units etc are application specific user interfaces.
- Mobile phone is an example of application specific user interface.
- In mobile phone the user interface is provided through the keypad, system speaker, vibration alert etc.

6 | [10] | CO3 | L2

The Embedded System and the General purpose computer are at two extremes. The embedded system is designed to perform a specific task whereas as per definition the general purpose computer is meant for general use. It can be used for playing games, watching movies, creating software, work on documents or spreadsheets etc. Following are certain specific points that differentiates between embedded systems and general purpose computers:

Criteria	General Purpose Computer	Embedded system
Contents	It is combination of generic hardware and a general purpose OS for executing a variety of applications.	It is combination of special purpose hardware and embedded OS for executing specific set of applications
Operating System	It contains general purpose operating system	It may or may not contain operating system.
Alterations	Applications are alterable by the user.	Applications are non-alterable by the user.
Key factor	Performance" is key factor.	Application specific requirements are key factors.
Power Consumption	More	Less
Response Time	Not Critical	Critical for some applications

7 | [5+5] | CO3 | L2

- RISC and CISC are the two common Instruction Set Architectures (ISA) available for processor design.

RISC	CISC
☐ Reduced Instruction Set Computing	☐ Complex Instruction Set Computing
☐ It contains lesser number of instructions.	☐ It contains greater number of instructions.
☐ Instruction pipelining and increased execution speed.	☐ Instruction pipelining feature does not exist.
☐ Orthogonal instruction set(allows each instruction to operate on any register and use any addressing mode.	☐ Non-orthogonal set(all instructions are not allowed to operate on any register and use any addressing mode.
☐ Operations are performed on registers only, only memory operations are load and store.	☐ Operations are performed either on registers or memory depending on instruction.
☐ A larger number of registers are available.	☐ The number of general purpose registers are very limited.
☐ Programmer needs to write more code to execute a task since instructions are simpler ones.	☐ Instructions are like macros in C language. A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using more simpler single instruction in RISC.
☐ It is single, fixed length instruction.	☐ It is variable length instruction.