Scheme Of Evaluation and Solution
Internal Assessment Test 1 – March 2019

Sub:	ARM MICROCONTROLLER & EMBEDDED SYSTEM							Sec	A
Date:	5 / 03 / 2019	Duration:	90 mins	Max Marks:	50	**Sem:**	VI	**Branch:**	TCE

Note: Answer Any Five Questions

	Description	**Marks Distribution**	
1	**Explain interrupts and exceptions in Cortex M3 processor.** Definition/ theory of interrupts and exceptions Tabulation of Any 7 interrupt types	3M 7M	10 M
2	**List and explain the applications of Cortex M3 processor.** **5 application each carry 2M**	10M	10 M
3	Draw and explain the Cortex M3 core and list out its significant features Drawing of cortex M3 core Explain Any 5 significant features	3 M 2 M 5 M	10 M
4	Describe register structure of Cortex M3 in detail(both GPRs and SPRs). GPRS SPRs	4 M 6M	10 M
5	Explain stack memory operations on Cortex M3 processor with necessary diagrams. Push pop	5M 5 M	10 M
6	List and give details of different profiles of ARM cortex (processor family and Architecture Versions) **4 versions each 2.5Marks**	10M	10 M

Exceptions are numbered 1 to 15 for system exceptions and the rest 240 for external interrupt inputs (Total 256 entries in vector table)

Most of the exceptions have programmable priority, and a few have fixed priority.

The value of the current running exception is indicated by the special register IPSR or from the NVIC's Interrupt Control state Register.

Exception Number	Exception Type	Priority	Description
1.	Reset	-3(Highest	Reset
2	NMI	-2	Nonmaskable interrupt (external NMI i/p)
3	Hardfault	-1	All fault condition, if the corresponding fault handler is not enabled.
4.	MemManage fault	Programmable	Memory management fault, MPU violation access to illegal locations
5.	Bus fault	Programmable	Bus error like Prefetch abort
6.	Usage fault	Programmable	Exceptions due to program error or typing to access coprocessor.

7-10	Reserved	N/A	—
11	SVCall	Programmable	System Service call
12.	Debug Monitor	Programmable	Debug Monitor.
13	Reserved	N/A	—
14	PendSV	Programmable	Pipe-able Pendable request for System device
15.	SYSTICK	Programmable	System Tick Timer
16.	External Interrupt #0	Programmable	External Interrupt
17.	External Interrupt #1	Programmable	External Interrupt
...	...	...	...
255	External Interrupt #239	Programmable	External Interrupt.

Vector Table

The processor will need to locate the starting address of the exception handler when an exception is being handled. This information is stored in the vector table.

Exception Vector Table After Power Up.

Address	Exception Number	Value (Word Size)
0x00000000	—	MSP Initial Value
0x00000004	1	Reset vector (Program counter initial value)
0x00000008	2	NMI handler starting address
0x0000000C	3	Hard fault handler starting address
	...	other handler starting address

(a) Low-cost micro-controller: 5-Applications each 2m

↳ It is ideally suited for low-cost micro-controllers which are commonly used in consumer products from toys to electrical appliances.

2a) ↳ It is highly competitive market due to the many well known 8-bit & 16-bit microcontroller products on the market.

(b) Automotive:

↳ It has very high performance efficiency & low interrupt latency allowing it to be used in real-time system in the automotive industry.

↳ It supports 240 external vectored interrupts with a built-in interrupt controller with nested interrupt supports & an optical MPU making it ideal for highly integrated & cost-sensitive automotive application.

(c) Data communication:

↳ It's low power & high efficiency features coupled with thumb-2 instructions for bit-field manipulation make the CORTEX M3 ideal for many communication application.

(d) Industrial control:

↳ The processor's interrupts for features

low latency & enhanced fault-finding features make it a strong candidate in this area.

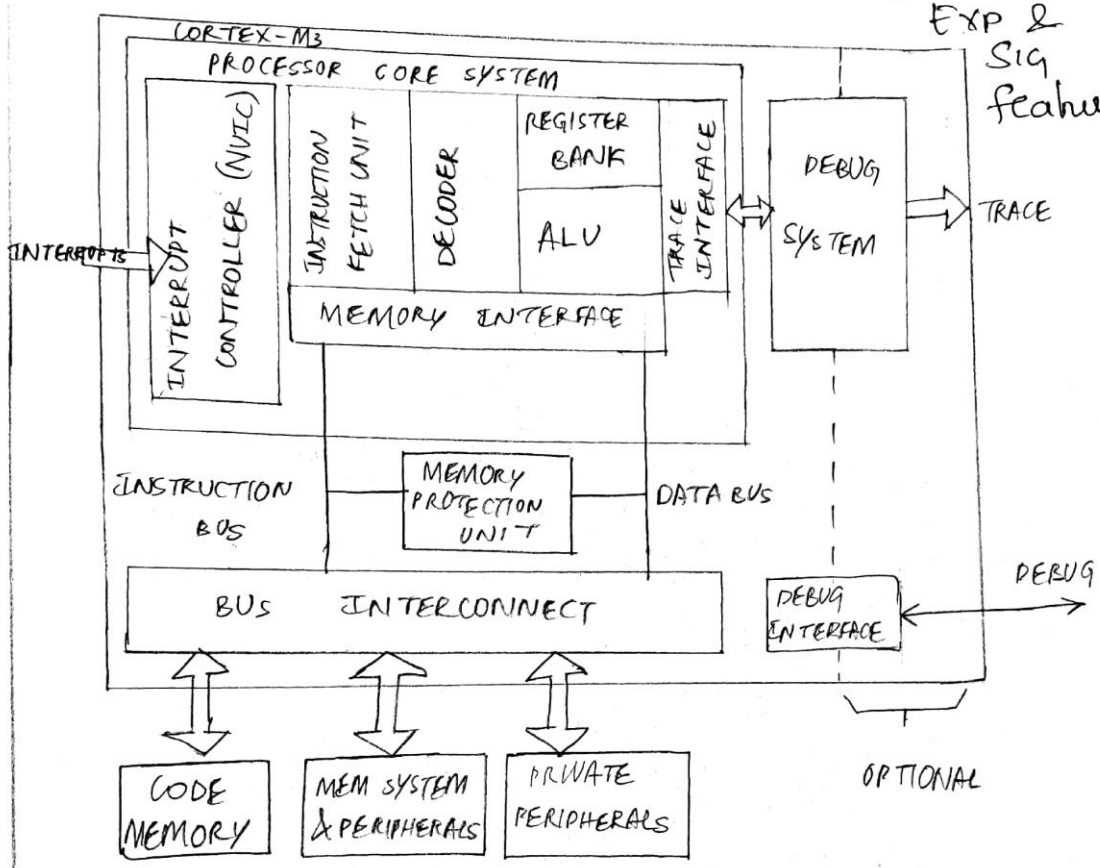
(e) Consumer products:

↳ CORTEX-M3, being a small processor, is highly efficient & low in power & support a MPU enabling ~~supply~~ complex software to execute while providing robust memory protection.

features.

Q5)

DIAGRAM - 4m
EXP & SIG features - 6m



- The Cortex-M3 processor, based on the ARMv7-M architecture has a hierarchical structure. It integrates the central processor core, called the CM3 core, with advanced system peripherals to enable integrated capabilities like interrupt control, memory protection and system debug and trace.
- These peripherals are highly configurable to allow the Cortex M3 processor to address a wide range of applications an be more closely aligned with the system requirements.
- The Cortex M3 core and the integrated components have been specifically designed to meet the requirements of minimal memory implementation, reduced pin count and low power consumption.

FUNDAMENTALS

- The Cortex M3 is a 32 bit microprocessor. It has a 32 bit data path, 32 bit registers bank, and 32 bit memory interfaces.
- The processor has a harvard architecture which means that it has a separate instruction bus and data bus. This allows instructions and data accesses to take place at the same time and as a result of this, the performance of the processor increases because data ac do not affect the instruction pipeline.
- For complex applications that require more memory and system features, the Cortex M3 processor has optional protection unit (MPU) and it is possible to u

an external cache if its required.

The Cortex M3 processor includes a number of fixed internal debugging components. These components provide debugging operation supports & features, such as breakpoints & watchpoints.

Q4

→ The cortex-M3 processor has registers, R0 through R15, R13 is banked with only one copy of the R13 visible at a time.

i] **R0-R12 General Purpose Registers:-**

R0-R12 are 32 bit GPR's for data operations where R0-R7 (low registers), R8-R12 (High registers).

Some of the 16-bit thumb instructions can only access a subset of these registers (ie, low registers)

R13: Stack Pointer:-

The cortex-M3 contains 2 stack pointers

They are banked so that only one is visible at a time. The banked out registers are not currently active and are not in use or accessible.

These banked out registers become active only when the processor changes modes.

The 2 stack pointers are: i] Main Stack Pointer
ii] Process Stack Pointer.

* MSP : Also called as the SP_main in arm documen
This is the default SP. It is used by the OS as kernel, exception handler and all application codes that require privileged access.

* PSP : Also called as SP_process in ARM controller documentation. This is used by base level application code (when not running an exception handler).

The duality of SP allows 2 seperate stack memories to be set up. When using the register name R13, one can only access the current SP, the other one is also inaccessible unless one uses special instruction to move to special register from general purpose register (MSR) and move special register to general purpose.

R14 : The Link Register :
It is used to store the return program counter when a sub-routine or a function is called. When a sub-routine is called, the return address is stored in the link register. It holds the address to return to when a function call completes.

R15 : The Program Counter :
It is the current program address. This register

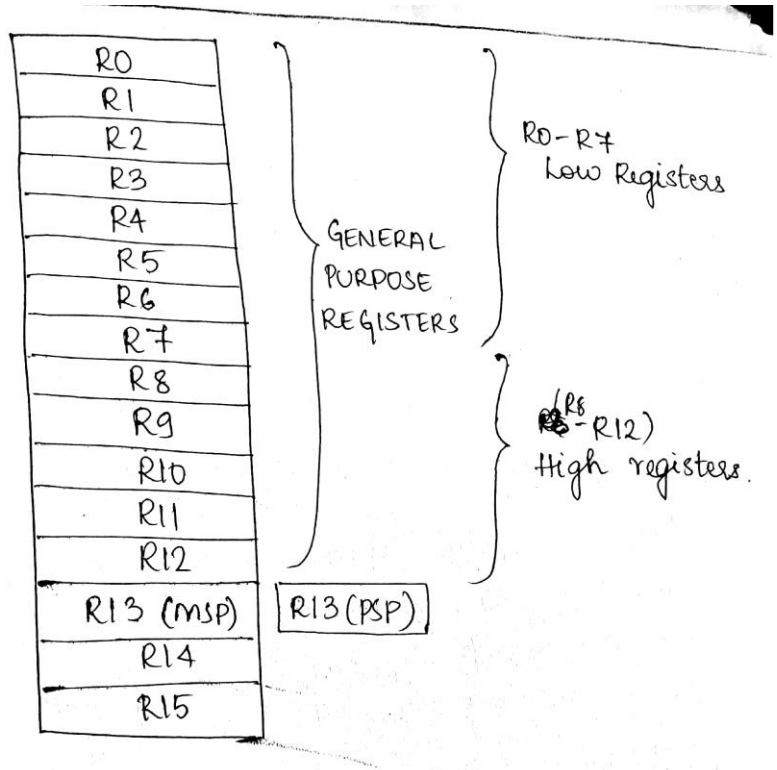
can be written to control the program flow.

ii] SPECIAL REGISTERS:-
The cortex-M3 processor has a no. of special registers:

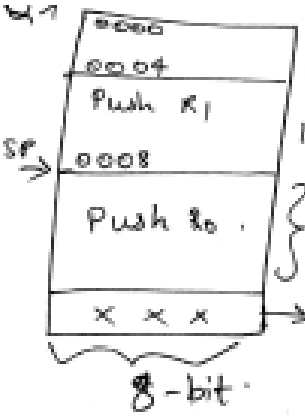
a) Program status registers:
- Provides arithmetic & logic processing flags (zero flag & carry flag), execution status & current executing interrupt number.

b) Interrupt / Exception Mask Registers (PRIMASK, FAULTMASK & BASEPRI) :-
When hard fault happens, the address is no longer in main memory, but has been swapped.
- PRIMASK: Disable all interrupts, except the NMI hard fault. When hard fault occurs, block of memory had to be returned from the page file instead of physical.
- FAULTMASK: Disable all interrupts except the NMI
- BASEPRI :- Disable all interrupts of specific prior or low priority level.

c) CONTROL REGISTER:-
- Define privileged status and stack pointer selection


R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (MSP)
R14
R15
R13 (PSP)
GENERAL PURPOSE REGISTERS
R0-R7
Low Registers
(R8-R12)
High registers.

5



PUSH - STM
POP - STM

Push : Decrements SP by 4.
POP : Increments SP by 4.

Push {R0}; R13 = R13 - 4, then memory [R13] = R0
POP {R0}; R0 = memory [R13], then R13 = R13 + 4

Multiple registers can be pushed & popped in one instruction.

Eg: Push {R0, R1}
Push {R0 - R7, R12, R8}

Cortex M3 contains 2 stack pointers (R13). They are banked so that only one is visible at a time.

* Main stack pointer (MSP) : The default stack pointer
* Process stack pointer (PSP) – Used by user application code.

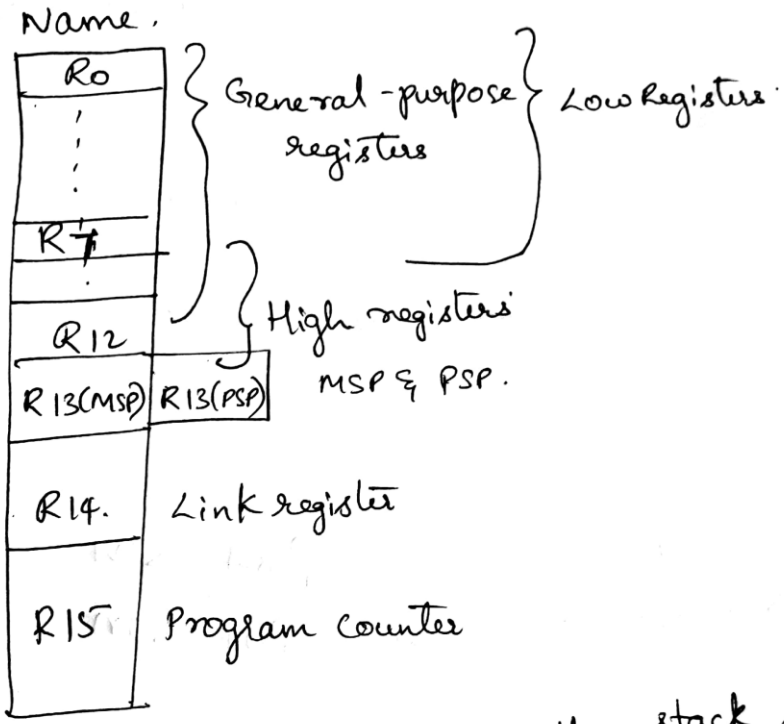


Fig: Diagram showing the stack register.

Basic Operations of the Stack

• 3.6.2 Cortex-M3 Stack Implementation

Main program

```
    ...
; R0 = X, R1 = Y, R2 = Z
BL      function1
```

Subroutine

```
function1
    PUSH    {R0} ; store R0 to stack & adjust SP
    PUSH    {R1} ; store R1 to stack & adjust SP
    PUSH    {R2} ; store R2 to stack & adjust SP
    ... ; Executing task (R0, R1 and R2
      ; could be changed)
    POP     {R2} ; restore R2 and SP re-adjusted
    POP     {R1} ; restore R1 and SP re-adjusted
    POP     {R0} ; restore R0 and SP re-adjusted
    BX      LR   ; Return
```

```
; Back to main program
; R0 = X, R1 = Y, R2 = Z
... ; next instructions
```

Main program

```
...
; R0 = X, R1 = Y, R2 = Z
BL    function 1
```

Subroutine

```
function 1
    PUSH    {R0-R2} ; Store R0, R1, R2 to stack
    ... ; Executing task (R0, R1 and R2
      ; could be changed)
    POP     {R0-R2} ; restore R0, R1, R2
    BX      LR   ; Return
```

```
; Back to main program
; R0 = X, R1 = Y, R2 = Z
... ; next instructions
```

FIGURE 3.12

Stack Operation Basics: Multiple Register Stack Operation.

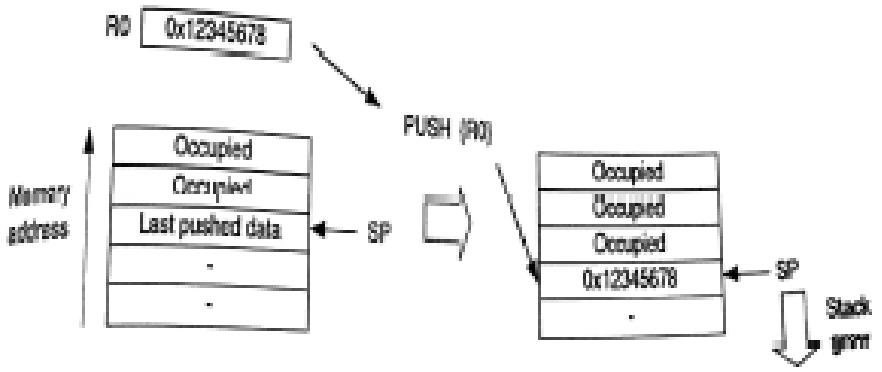
Main program

```
...
; R0 = X, R1 = Y, R2 = Z
BL    function 1
```

Subroutine

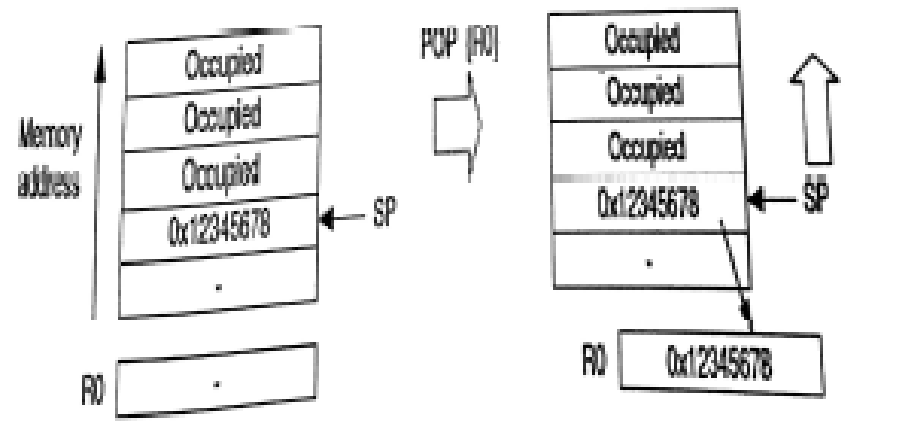
```
function 1
    PUSH    {R0-R2, LR} ; Save registers
              ; including link register
    ... ; Executing task (R0, R1 and R2
      ; could be changed)
    POP     {R0-R2, PC} ; Restore registers and
              ; return
```

```
; Back to main program
; R0 = X, R1 = Y, R2 = Z
... ; next instructions
```



URE 3.14

rtex-M3 Stack PUSH Implementation.



RE 3.15

3.6.3 The Two-Stack Model in the Cortex-M3

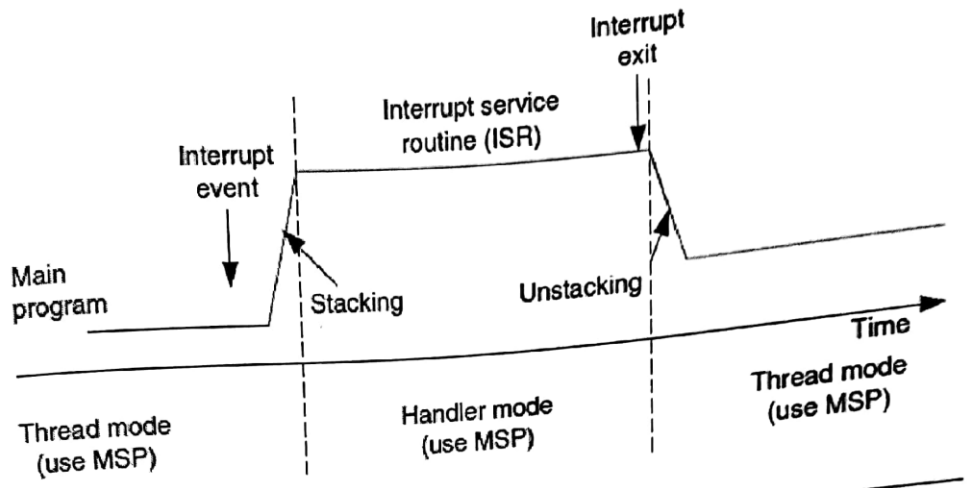


FIGURE 3.16
CONTROL[1]□0: Both Thread Level and Handler Use Main Stack.

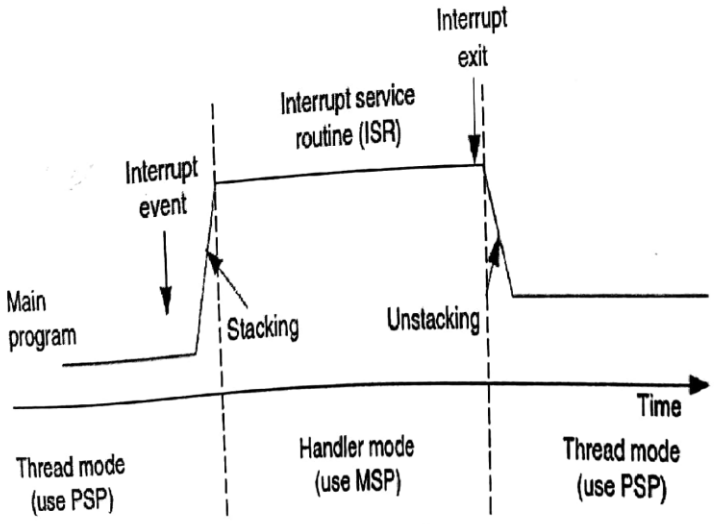


FIGURE 3.17
CONTROL[1]=1: Thread Level Uses Process Stack and Handler Uses Main Stack.

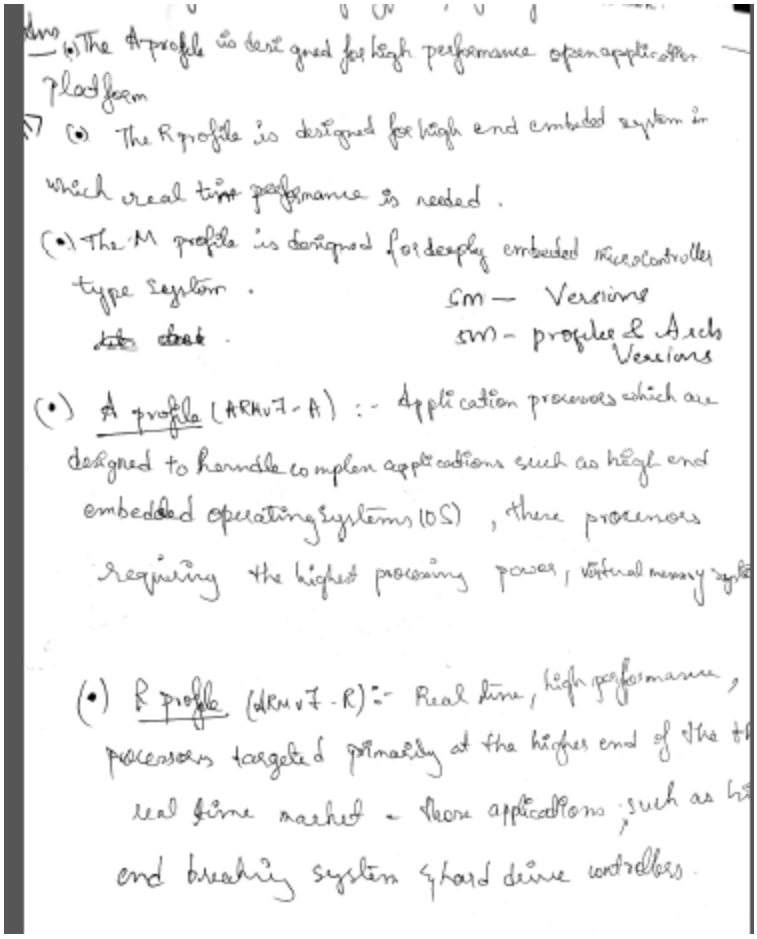
It is possible to perform read/write operations directly to the MSP and PSP, without any confusion of which R13 you are referring to. Provided that you are in privileged level, you can access MSP and PSP values:

```
x = __get_MSP(); // Read the value of MSP
__set_MSP(x); // Set the value of MSP
x = __get_PSP(); // Read the value of PSP
__set_PSP(x); // Set the value of PSP
```

In general, it is not recommended to change current selected SP values in a C function, as the stack memory could be used for storing local variables. To access the SPs in assembly, you can use the MRS and MSR instructions:

```
MRS R0, MSP  ; Read Main Stack Pointer to R0
MSR MSP, R0  ; Write R0 to Main Stack Pointer
MRS R0, PSP  ; Read Process Stack Pointer to R0
MSR PSP, R0  ; Write R0 to Process Stack Pointer
```

ure 3.18):



(•) The A profile is best used for high performance open application platform

(•) The R profile is designed for high end embedded system in which real time performance is needed.

(•) The M profile is designed for deeply embedded microcontroller type system.

SM — Versions

SW - profiles & Archs Versions

(•) A profile (ARMv7-A) :- Application processors which are designed to handle complex applications such as high end embedded operating systems (OS), these processors requiring the highest processing power, virtual memory syst

(•) R profile (ARMv7-R) :- Real time, high performance, processors targeted primarily at the higher end of the th real time market - these applications such as hi end braking system, hard drive controllers.

(*) M profile (ARMv7-M): Processor targeting low cost application in which processing efficiency as important as cost, power consumption, low interrupt latency & H.w easy use are critical, as well as industrial control application including real time control system.

Background of ARM & ARM architecture