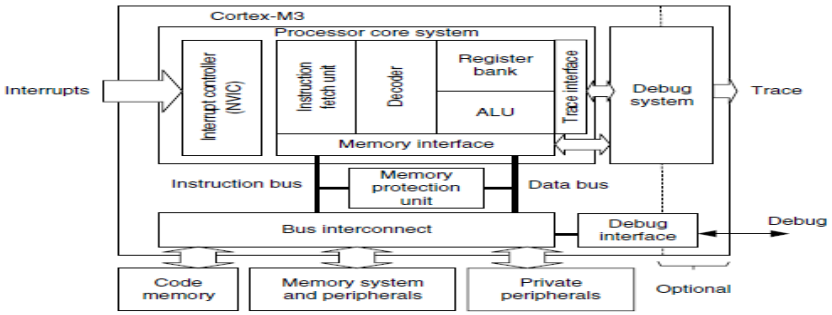
1. Describe with neat diagram, the architecture of Cortex M3 Processor.

Ans. The ARM Cortex™-M3 processor, the first of the Cortex generation of processors released by ARM **in 2006**, **Cortex M3 is a ,**§Small,§Low power,§Rich interfaces,§Minimum 3 stage pipeline §Harvard Architecture.Multiple core buses capable of simultaneous accesses.Tightly integrated interrupt controller with Wake-up Interrupt Controller Interruptible/continuable multiple load and store instructions Supports MPU. 18 x 32-bit registers, Excellent compiler target, Reduced pin count requirements, Efficient interrupt handling, Power management, Efficient debug and development support features, Breakpoints, Watchpoints, Flash Patch support, Instruction Trace Strong OS support User/Supervisor model OS support features Designed to be fully programmed in C (even reset, interrupts and exceptions), ARMv7M Architecture No Cache - No MMU, Debug is optimized for microcontroller applications, Vector table contains addresses, not instructions, DIV instruction, Interrupts automatically save/restore state, Exceptions programmed in C (No Coprocessor 15 - All registers are memory-mapped), Interrupt controller is part of Cortex-M3 macrocell, Fixed memory map, Bit-banding, Non-Maskable Interrupt (NMI), Only one processor status reg, Thumb-2 processing core, Mix of 16 and 32 bit instructions for very high code density, Gives complete Thumb compatibility.



2. Write short notes on Exceptions, Interrupts and Vector Table.

Ans. The Cortex-M3 supports a number of exceptions, including a fixed number of system exceptions and a number of interrupts, commonly called IRQ. The number of interrupt inputs on a Cortex-M3 microcontroller depends on the individual design. Interrupts generated by peripherals, except System Tick Timer, are also connected to the interrupt input signals. The typical number of interrupt inputs is 16 or 32. However, you might find some microcontroller designs with more (or fewer) interrupt inputs. The vector table is an array of word data inside the system memory, each representing the starting address of one exception type.

Table 3.4 Exception Types in Cortex-M3

Exception Number	Exception Type	Priority	Function
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Nonmaskable interrupt
3	Hard fault	–1	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking
4	MemManage	Settable	Memory management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a nonexecutable region)
5	Bus fault	Settable	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	Usage fault	Settable	Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3)
7–10	—	—	Reserved
11	SVC	Settable	Supervisor call via SVC instruction
12	Debug monitor	Settable	Debug monitor
13	—	—	Reserved
14	PendSV	Settable	Pendable request for system service
15	SYSTICK	Settable	System tick timer
16–255	IRQ	Settable	IRQ input #0–239

Table 3.5 Vector Table Definition after Reset

Exception Type	Address Offset	Exception Vector
18–255	0x48–0x3FF	IRQ #2–239
17	0x44	IRQ #1
16	0x40	IRQ #0
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	Reserved
12	0x30	Debug monitor
11	0x2C	SVC
7–10	0x1C–0x28	Reserved
6	0x18	Usage fault
5	0x14	Bus fault
4	0x10	MemManage fault
3	0x0C	Hard fault
2	0x08	NMI
1	0x04	Reset
0	0x00	Starting value of the MSP

3a. Explain the operating modes of Cortex M3 with the help of neat diagrams indicating the switching of states on the occurrence of exceptions.

→ The Cortex-M3 processor has 2 operation modes & 2 privilege levels

↳ The operation modes determine whether the processor is running a normal program or running an exception handler like an interrupt handler or system exception handler.

↳ 2 types of operating modes:

(a) Thread mode

(b) Handler mode.

↳ The privilege levels provide a mechanism for safeguarding memory access to critical regions as well as providing a basic security ~~level~~ model.

↳ 2 types of privilege level:-

(a) Privileged level

(b) User level.

↳ When the processor is running a main program (thread mode), it can be either in a privileged state or a user state, but exception handlers can only be in a privileged state.

↳ When the processor exits reset, it is in thread mode with privileged access rights.

↳ In privileged state, a program has access to all memory ranges and can use all supported instructions.

↳ Software in the privileged access levels can switch the program into the user access level using the control register.

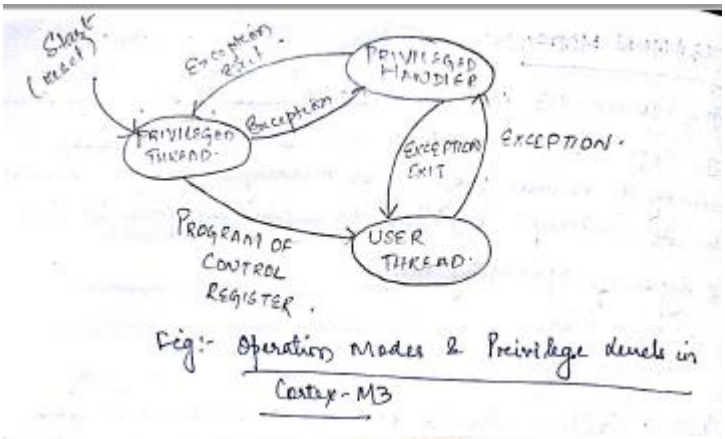


Fig:- Operation Modes & Privilege levels in Cortex-M3

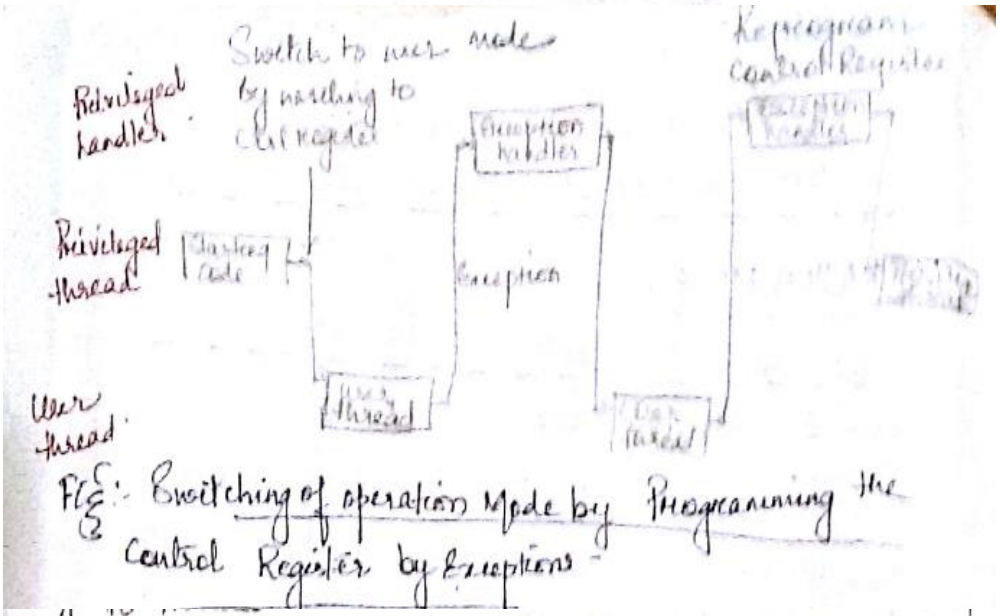


Fig:- Switching of operation Mode by Programming the Control Register by Exceptions

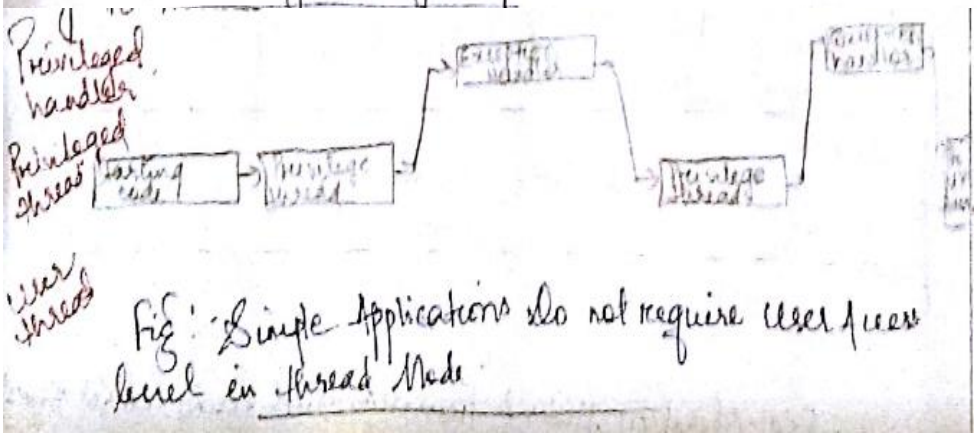


Fig:- Simple Applications do not require User/user level in thread Mode

b. List the applications of Cortex M3 .

Low-cost microcontrollers: The Cortex-M3 processor is ideally suited for low-cost microcontrollers, which are commonly used in consumer products, from toys to electrical appliances. It is a highly competitive market due to the many well-known 8-bit and 16-bit microcontroller products on the market. Its lower power, high performance, and ease-of-use advantages enable embedded developers to migrate to 32-bit systems and develop products with the ARM architecture.

Automotive: Another ideal application for the Cortex-M3 processor is in the automotive industry. The Cortex-M3 processor has very high-performance efficiency and low interrupt latency, allowing it to be used in real-time systems. The Cortex-M3 processor supports up to 240 external vectored interrupts, with a built-in interrupt controller with nested interrupt supports and an optional MPU, making it ideal for highly integrated and cost-sensitive automotive applications.

Data communications: The processor's low power and high efficiency, coupled with instructions in Thumb-2 for bit-field manipulation, make the Cortex-M3 ideal for many communications applications, such as Bluetooth and ZigBee.

Industrial control: In industrial control applications, simplicity, fast response, and reliability are key factors. Again, the Cortex-M3 processor's interrupt feature, low interrupt latency, and enhanced fault-handling features make it a strong candidate in this area.

Consumer products: In many consumer products, a high-performance microprocessor (or several of them) is used. The Cortex-M3 processor, being a small processor, is highly efficient and low in power and supports an MPU enabling complex software to execute while providing robust memory protection.

4a Explain the Program Status registers with bit pattern.

* The PSRs are divided into 3 status registers.
  - Application PSR
  - Interrupt PSR
  - Execution PSR

The 3 PSRs can be accessed together / separately using the special register access instructions MSR & MRS. When they are accessed as a collective term, the name xPSR is used.

We can read the PSRs using the MRS instruction and can also change the APSR using the MSR instruction, but EPSR & IPSR are read-only. for eg:-

```
MRS   R0, APSR ; Read flag state into R0
MRS   R0, IPSR ; Read exception/Interrupt state
MRS   R0, EPSR ; Read Execution State
MSR   APSR, R0 ; Write flag state
```

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												EXCEPTION NUMBER				
EPSR						ICI/IT	T			ICI/IT						

Fig:- PSRs in Cortex-M3.

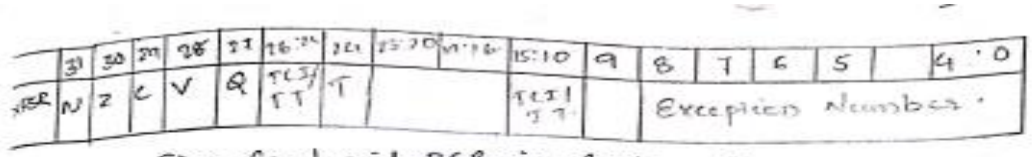


Fig:- Combined xPSR in Cortex-M3.

↳ In ARM Assembler, while accessing xPSR (All three PSRs as one), the Symbol PSR is used:

MRS r0, PSR; Read the combined program Status Word.

MSR PSR, r0; Write the combined program Status Word.

Description of the bit fields:

N → Negative (Set after the arithmetic/logical operation) → 1 → Negative, 0 → Positive.

Z → Zero → (shows the comparison operation is zero or equal). ↓ 1: zero/equal, 0: Not zero/Not equal)

C → Carry / borrow (Set to 1).

V → Overflow.

Q → Sticky Saturation flag.

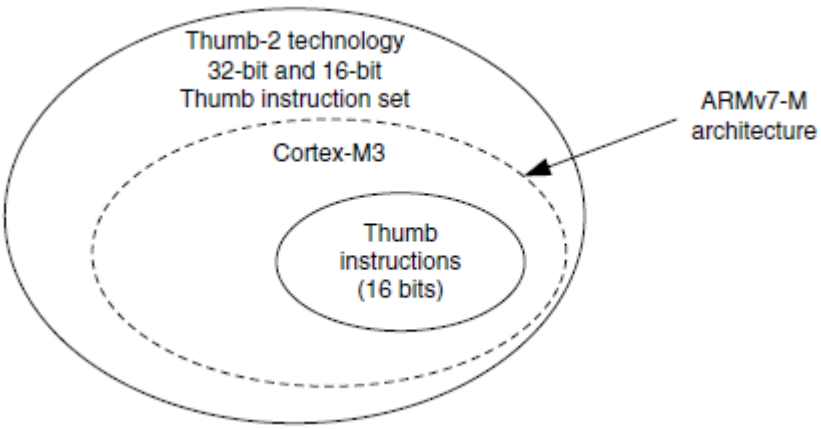
ICI / IT → Interrupt-Continuable Instruction bits, If-Then instruction status bits.

T → Thumb State, always 1; trying to clear this bit will cause a fault exception.

Exception number → Indicates which exception the processor is handling.

* The Carry flag is used for unsigned addition / subtraction.
* The Overflow flag is used for signed addition / subtraction.

4b Explain Thumb2 technology.



The Thumb-2[3] technology extended the Thumb Instruction Set Architecture (ISA) into a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance (see Figure 1.4). The extended instruction set in Thumb-2 is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions. It allows more complex operations to be carried out in the Thumb state, thus allowing higher efficiency by reducing the number of states switching between ARM state and Thumb state.

Focused on small memory system devices such as microcontrollers and reducing the size of the processor, the Cortex-M3 supports only the Thumb-2 (and traditional Thumb) instruction set. Instead of using ARM instructions for some operations, as in traditional ARM processors, it uses the Thumb-2 instruction set for all operations. As a result, the Cortex-M3 processor is not backward compatible with traditional ARM processors. That is, you cannot run a binary image for ARM7 processors on the Cortex-M3 processor. Nevertheless, the Cortex-M3 processor can execute almost all the 16-bit Thumb instructions, including all 16-bit Thumb instructions supported on ARM7 family processors, making application porting easy.

With support for both 16-bit and 32-bit instructions in the Thumb-2 instruction set, there is no need to switch the processor between Thumb state (16-bit instructions) and ARM state (32-bit instructions). For example, in ARM7 or ARM9 family processors, you might need to switch to ARM state if you want to carry out complex calculations or a large number of conditional operations and good performance is needed, whereas in the Cortex-M3 processor, you can mix 32-bit instructions with 16-bit instructions without switching state, getting high code density and high performance with no extra complexity.

The Thumb-2 instruction set is a very important feature of the ARMv7 architecture. Compared with the instructions supported on ARM7 family processors (ARMv4T architecture), the Cortex-M3 processor instruction set has a large number of new features. For the first time, hardware divide instruction is available on an ARM processor, and a number of multiply instructions are also available on the Cortex-M3 processor to improve data-crunching performance. The Cortex-M3 processor also supports unaligned data accesses, a feature previously available only in high-end processors.

5a Explain the 2-level Stack model in Cortex M3.

THE 2-STACK MODEL IN CORTEX-M3 :-

↳ As mentioned the Cortex M3 has 2 SPs :-

(a) MSP (Main Stack Pointer)

(b) PSP (Process Stack Pointer)

↳ The SP register to be used is controlled by the control register bit 1

↳ When CONTROL[1] = 1, PSP is used in Thread Mode.

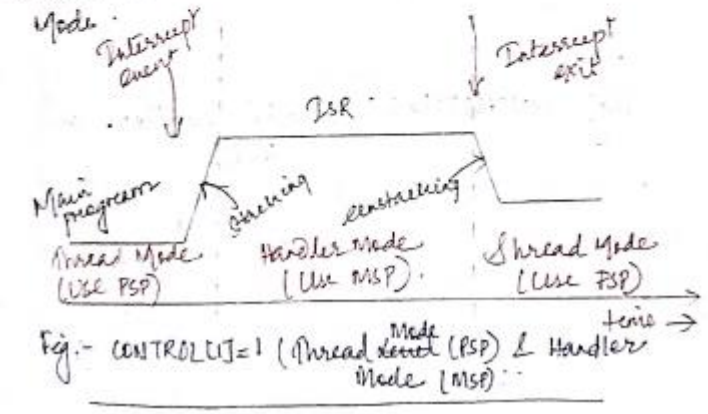


Fig.- CONTROL[1]=1 (Thread Mode (PSP) & Handler Mode (MSP))

↳ In the above arrangement, the main program & the exception handlers can have separate stack memory regions which can prevent a stack error in a user application from damaging the stack used by the OS.

↳ When CONTROL[1]=0, MSP is used for both thread & handler mode

↳ In this Arrangement, the main program & exception handlers share the same stack memory which is called as default setting after power-up and it is disadvantageous

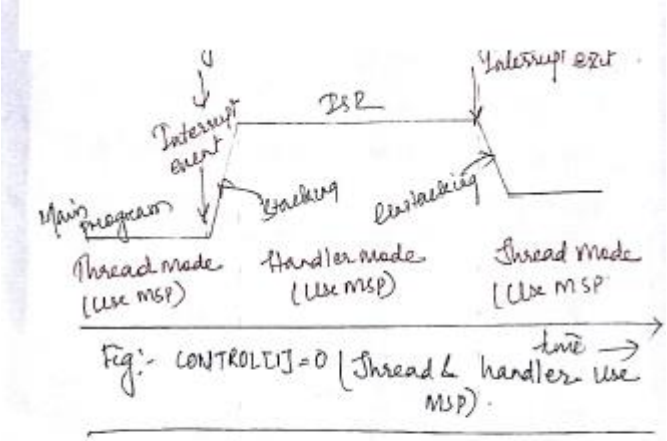


Fig:- CONTROL[1]=0 (Thread & handler use MSP).

↳ It is possible to perform read/write operations directly to the MSP & PSP, without any confusion of which R13, we are referring to.

5b. What is Stack? Explain the multiple register Stack operation.

Stack:-

↳ A memory usage model. It is simply a part of the system memory & a pointer register is used to make it work as a FILO/LIFO buffer.

↳ The common use of a Stack is to save register contents before some data processing & then restore those contents from the Stack after the processing task is done.

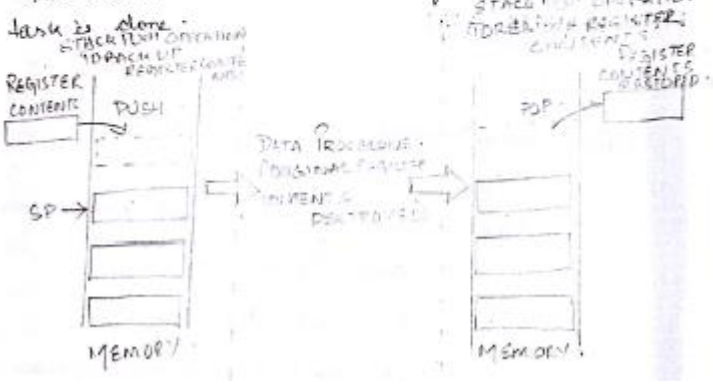


Fig:- Basic Concept of Stack Memory.

↳ Instructions used are PUSH & POP :-

↳ ALP Syntax :-

PUSH {R0}; R13 = R13 - 4, then Memory[R13] = [R0].

POP {R0}; R0 = Memory[R13], then R13 = R13 + 4.

↳ In general Stack operations are memory write / read operations with the addresses specified by an SP. Data in registers is saved into stack memory by a PUSH operation & can be restored to registers later by a POP operation.

↳ The SP is adjusted automatically in PUSH & POP so that multiple data PUSH will not cause old stacked data to be erased.

↳ Below mention are the pseudo ALPs describing stack operations in 3 ways :-

(a) For normal use, for each store (PUSH), there must be a corresponding read (POP), & the address of the POP operation should match that of the PUSH operation. For this, the ALP is :

(b) The PUSH & POP operations can allow multiple load & store.

↳ In this case, the ordering of a register POP is automatically reversed by the processor.

↳ The ALP is :-

Main program
...
; R0 = x, R1 = y, R2 = z
BL function1 ; Subroutine 1
...

function1
PUSH {R0-R2}; Store R0, R1, R2 to stack
... ; Executing task {R0, R1, R2 can
; be changed}
POP {R0-R2}; Restore R0, R1, R2
BX LR; Return

; Back to main program
; R0 = x, R1 = y, R2 = z
... ; next instructions

ALP Pseudocode :- Multiple Register Stack operations.

6a. Explain the reset sequence with the help of memory map.

RESET SEQUENCE:-

↳ After the processor gets reset, it will need 2 words from the memory.

* Address 0x00000000 : Starting value of R13 (SP).
* Address 0x00000004 : Reset vector [The starting address of program execution].

↳ This method is illustrated in the following diagram.

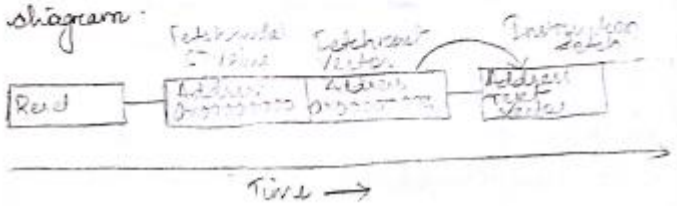


Fig:- Reset Sequence

↳ In cortex-M3, the initial value for the MSP is put at the beginning of the memory map followed by the vector table which contains vector address values.

↳ The contents of the vector table are address values & not branch instructions. The 1st vector in the vector table is the reset vector which is the second piece of data fetched by the processor after reset.

↳ As the stack operation in cortex-M3 is a full descending stack (SP decrements before store), the initial SP value should be set to the first memory after the top of the stack region. For eg:- if a stack memory ranges from 0x20007C00 to 0x20007FFF (1KB), the initial stack value should be set to 0x20008000.

↳ The vector table starts after the initial SP value.

↳ The first vector is the reset vector. After the reset vector is fetched, the cortex-M3 can then start to execute the program from the reset vector address & begin normal operation.

↳ In Cortex-M3, the vector addresses in the vector table should have their LSB set to 1 to indicate that they are Thumb code. This is illustrated in the diagram next:-

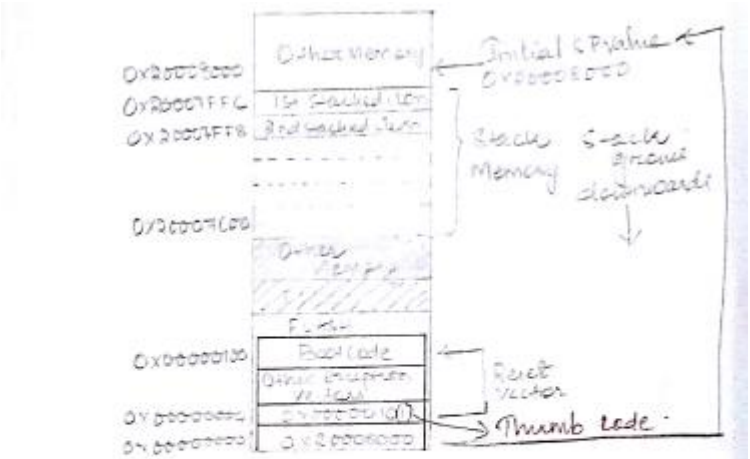


Fig:- Initial SP value & Initial PC value Example.

1) In the figure, to Indicate the thumb code, the reset-vector individual address become 0x101 whereas the boot code (offset) vector address is 0x100.

6b. Explain the Control register .

THE CONTROL REGISTER:-

↳ used to define the privilege level and the SP selection

↳ This register has 2 bits

(2) CONTROL[1]: This bit is always 0 in handler mode. but in thread / base level it is 0/1.

b) This bit is writable only in thread mode & privileged level. In user state / handler mode, writing to this bit is not allowed.

(b) CONTROL[0]: This bit is writable only in privileged state. Once it enters user state, the only way to switch back to privileged is to trigger an interrupt & change this in exception handler.

Summarizing both the bits:-

BITS	STATUS (FUNCTION)
CONTROL[1]:	Stack Status: 1 = Alternate Stack is used. 0 = Default (MSP) stack is used If it is in the thread or base level, the alternate stack is PSP. There is no alternate stack for handler mode, so this bit must be 0 when the processor is in handler mode.
CONTROL[0]:	0 = Privileged in thread mode. 1 = user state in thread mode. If in handler mode, the processor operates in privileged level.

To access this register in C:-

```
x = __get_CONTROL(); // Read the current value of control
__set_CONTROL(); // Set the control value to x.
```