

Cortex-M0 Technical Overview

**implementation Features.
Self-Study:**

System Features

Debug Features,

Advantages.

Cortex-M0 Technical Overview

- The Cortex-M0 processor is a 32-bit Reduced Instruction Set Computing (RISC) processor
- with a von Neumann architecture (single bus interface).
- It uses an instruction set called Thumb, which was first supported in the ARM7TDMI processor however, several newer instructions from the ARMv6 architecture and a few instructions from the Thumb-2 technology are also included.
- Thumb-2 technology extended the previous Thumb instruction set to allow all operations to be carried out in one CPU state.
- The instruction set in Thumb-2 included both 16-bit and 32-bit instructions;
- most instructions generated by the C compiler use the 16-bit instructions, and the 32-bit instructions are used when the 16-bit version cannot carry out the required operations.
- This results in high code density and avoids the overhead of switching between two instruction sets.

Cortex-M0 Technical Overview

- In total, the Cortex-M0 processor supports only 56 base instructions, although some instructions can have more than one form.
- Although the instruction set is small, the Cortex- M0 processor is highly capable because the Thumb instruction set is highly optimized.
- Academically, the Cortex-M0 processor is classified as load-store architecture, as it has separate instructions for reading and writing to memory, and instructions for arithmetic or logical operations that use registers.

Cortex-M0 Technical Overview

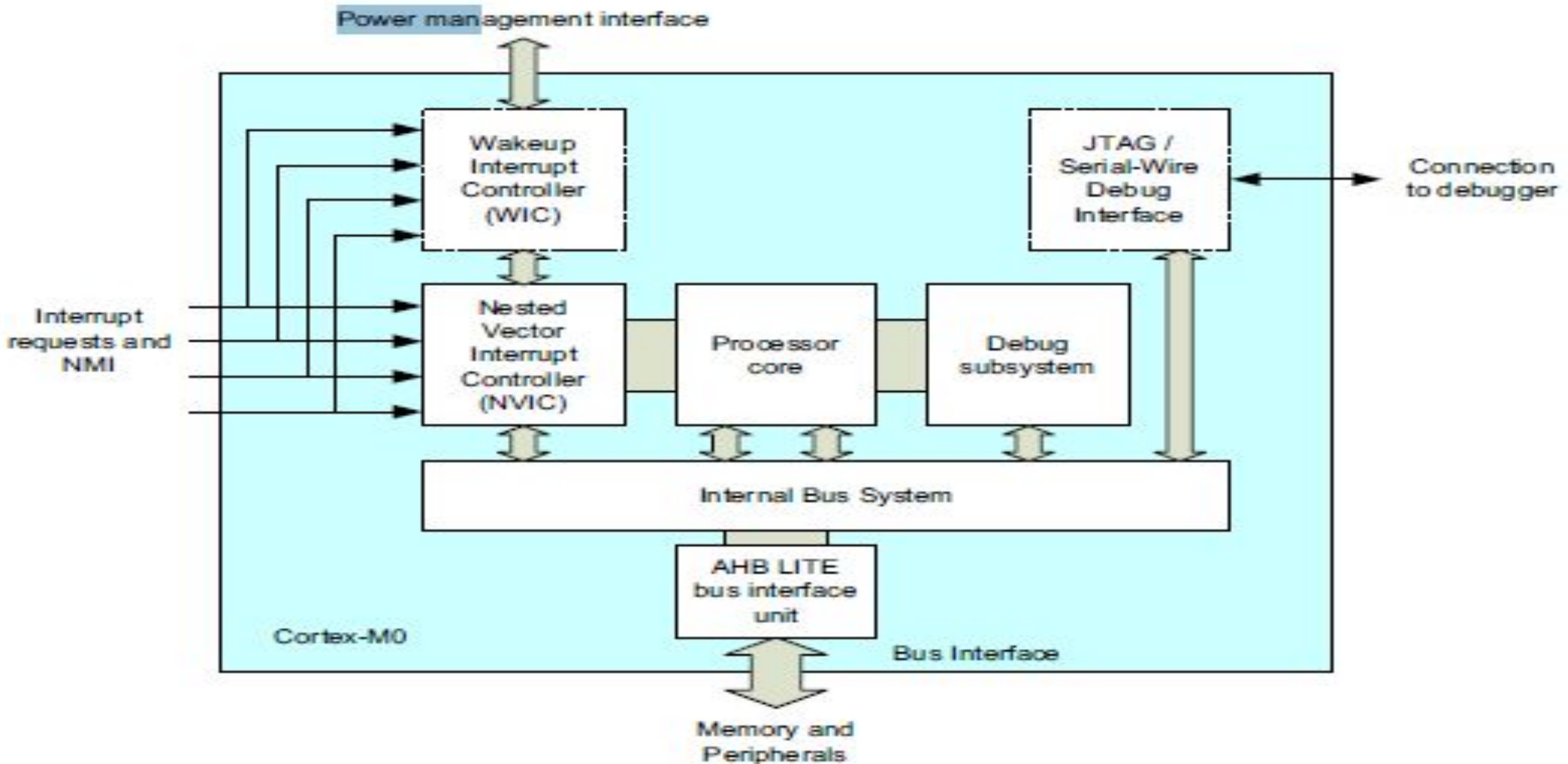


Figure 2.1:
Simplified block diagram of the Cortex-M0 processor.

Cortex-M0 Technical Overview

- **The processor core** contains the register banks, ALU, data path, and control logic.
- It is a three stage pipeline design with fetch stage, decode stage, and execution stage.
- The register bank has sixteen 32-bit registers.
- A few registers have special usages.
- **The Nested Vectored Interrupt Controller (NVIC)** accepts up to 32 interrupt request signals and a nonmaskable interrupt (NMI) input.
- It contains the functionality required for comparing priority between interrupt requests and the current priority level so that nested interrupts can be handled automatically.
- If an interrupt is accepted, it communicates with the processor so that the processor can execute the correct interrupt handler.

Cortex-M0 Technical Overview

- **The Wakeup Interrupt Controller (WIC)** is an optional unit.
- In low-power applications, the microcontroller can enter standby state with most of the processor powered down.
- In this situation, the WIC can perform the function of interrupt masking while the NVIC and the processor core are inactive.
- When an interrupt request is detected, the WIC informs the power management to power up the system so that the NVIC and the processor core can then handle the rest of the interrupt processing.
- **The debug subsystem** contains various functional blocks to handle debug control, program breakpoints, and data watchpoints.
- When a debug event occurs, it can put the processor core in a halted state so that embedded developers can examine the status of the processor at that point.

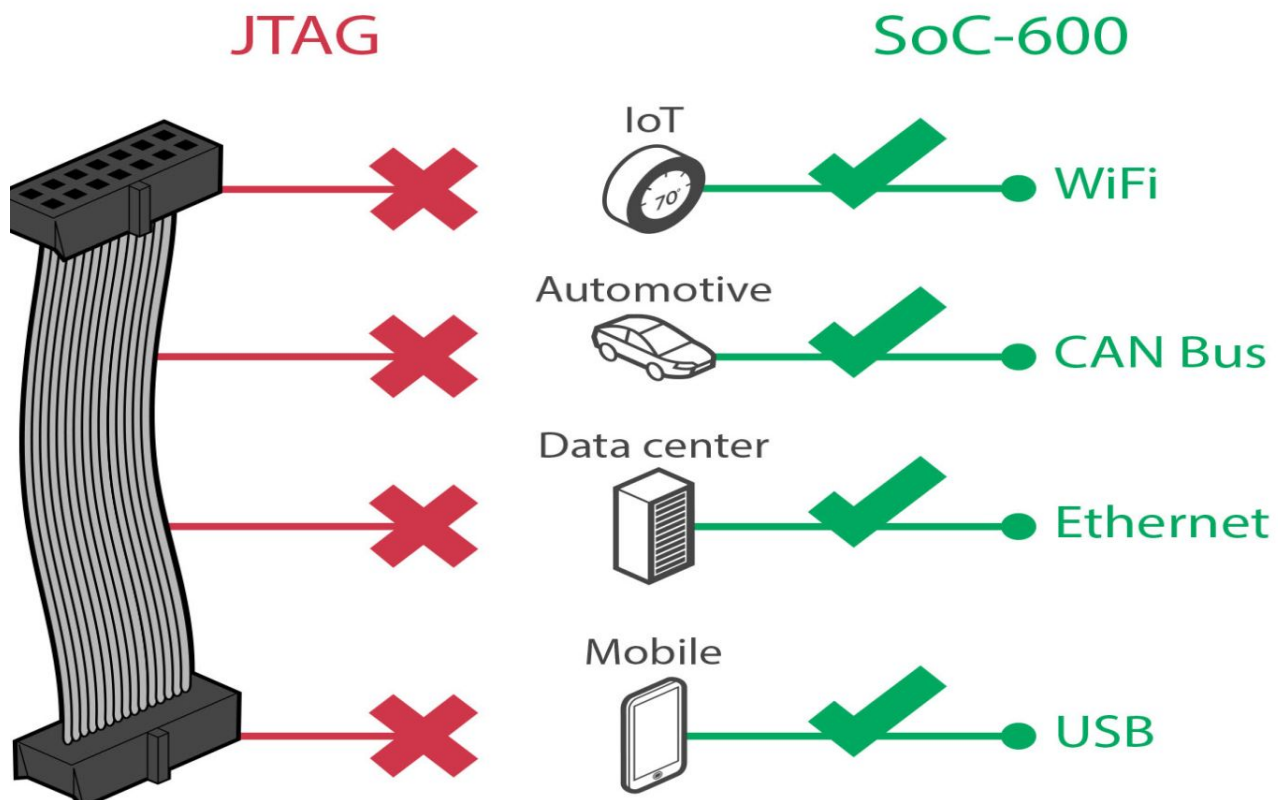
Cortex-M0 Technical Overview

- **The JTAG or serial wire interface** units provide access to the bus system and debugging functionalities.
- The JTAG protocol is a popular five-pin communication protocol commonly used for testing.
- The serial wire protocol is a newer communication protocol that only requires two wires, but it can handle the same debug functionalities as JTAG.
- **The internal bus system**, the data path in the processor core, and the AHB LITE (**Advanced High-performance Bus (AHB)**- **AHB** is a bus protocol introduced in Advanced Microcontroller Bus Architecture version 2)
- Bus interface are all 32 bits wide. AHB-Lite is an on-chip bus protocol used in many ARM processors.
- This bus protocol is part of the Advanced Microcontroller Bus Architecture (AMBA) specification, a bus architecture developed by ARM that is widely used in the IC design industry.

JTAG

The future of debug: CoreSight SoC-600

Empowering developers with increased debug visibility throughout the product lifespan



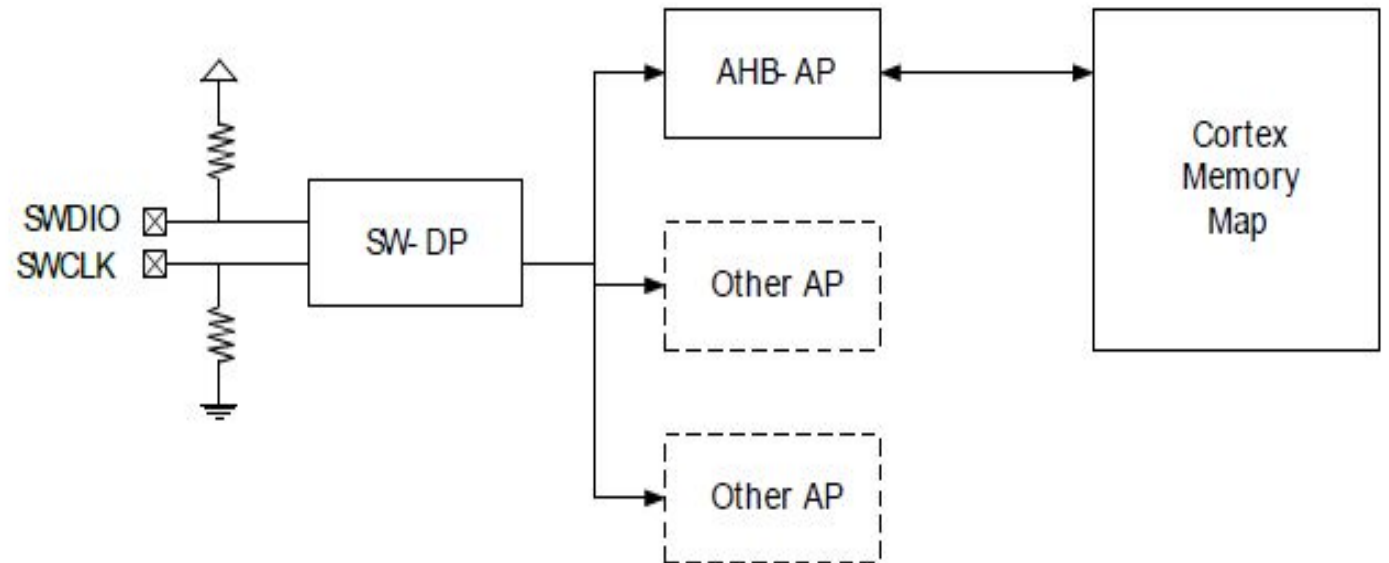
Serial Wire Debug Interface

Serial Wire Debug (SWD) is a two-wire protocol for accessing the ARM debug interface. It is part of the **ARM Debug Interface Specification v5** and is an alternative to JTAG.

The physical layer of SWD consists of two lines: • **SWDIO: a bidirectional data line** • **SWCLK: a clock driven by the host** Connecting to these pins allow an external device (such as a debug probe) to communicate directly with the **Serial Wire Debug Port (SW-DP)**.

The SW-DP in turn can access one or several Access Ports (APs) that give access to the rest of the system.. In other words the **AHB-AP can access the internal memory map of the core**. Since the internal flash, SRAM, debug components and peripherals all are memory mapped, this AP can control the entire device including programming it.

Serial Wire Debug interface



Serial Wire Debug Port (**SW-DP**) interface

AHB-AP is a Memory Access Port (MEM-AP)

System Configuration

- **POWER ON Setting**
- CON5 : Power Jack + 5V DC IN
- VCC: VCC power in/out
- VCC5: 5VCC power in/out
- VCC33:3VCC power in/out
- **JP3: System voltage**
- The Lower Base board is support 3V for system.
- **Debug Connect**
- **InCircuitEmulator:ICECON: USB connect to PC for debug NUC1XX.**
- **USB Connect**
- J3 mini USB Connector for NCU1XX USB function.
- **Reset**
- SW_RESET:Reset NCU140(low reset)

The ARM Cortex-M0 Processor Features

- The ARM Cortex-M0 processor contains many features. Some are visible system features, and others are not visible to embedded developers.
- **System Features**
 - Thumb instruction set. Highly efficient, high code density and able to execute all Thumb instructions from the ARM7TDMI processor.
 - High performance. Up to 0.9 DMIPS/MHz (Dhrystone 2.1) with fast multiplier or 0.85 DMIPS/MHz with smaller multiplier.

The ARM Cortex-M0 Processor Features

- **Built-in Nested Vectored Interrupt Controller (NVIC).** This makes interrupt configuration and coding of exception handlers easy.
- When an interrupt request is taken, the corresponding interrupt handler is executed automatically without the need to determine the exception vector in software.
- Interrupts can have **four** different programmable priority levels.
- **The NVIC automatically handles nested interrupts.**
- Deterministic **exception response** timing.
- The design can be set up to respond to exceptions (e.g., interrupts) with a fixed number of cycles (constant interrupt latency arrangement) or to respond to the exception as soon as possible (**minimum 16 clock cycles**).
- **Nonmaskable interrupt (NMI)** input for safety critical systems.

The ARM Cortex-M0 Processor Features

- Architectural predefined memory map. The memory space of the Cortex-M0 processor is architecturally predefined to make software porting easier and to allow easier optimization of chip design. However, the arrangement is very flexible.
- The memory space is linear and there is no memory paging required like in a number of other processor architectures.

The ARM Cortex-M0 Processor Features

- ```
/*

*/
```
- ```
/*          Peripheral memory map          */
```
- ```
/*

*/
```
- ```
/* Peripheral and SRAM base address */
```
- ```
#define FLASH_BASE ((uint32_t)0x00000000)
```
- ```
#define SRAM_BASE       (( uint32_t)0x20000000)
```
- ```
#define AHB_BASE ((uint32_t)0x50000000)
```
- ```
#define APB1_BASE       (( uint32_t)0x40000000)
```
- ```
#define APB2_BASE ((uint32_t)0x40100000)
```

# The ARM Cortex-M0 Processor Features

- Easy to use and C friendly. There are only **two modes** (Thread mode and Handler mode).
- The whole application, including exception handlers, can be written in C without any assembler.
- Built-in optional System Tick timer for **OS support**.
- A **24-bit timer** with a dedicated exception type is included in the architecture, which the OS can use as a tick timer or as a general timer in other applications without an OS.

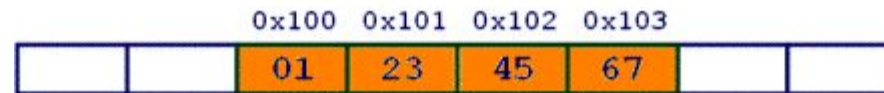
# The ARM Cortex-M0 Processor Features

- SuperVisor Call (SVC) instruction with a dedicated SVC exception
- PendSV (PendableSupervisor service) to support various operations in an embedded OS.
- Architecturally defined sleep modes and instructions to enter sleep.
- The sleep features allow power consumption to be reduced dramatically. Defining sleep modes as an architectural feature makes porting of software easier because sleep is entered by a specific instruction rather than implementation defined control registers.
- Fault handling exception to catch various sources of errors in the system.

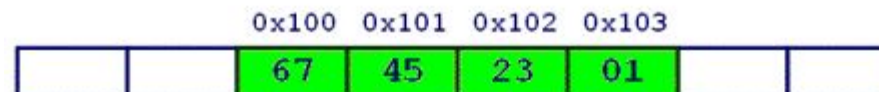


# Implementation Features

- Configurable number of interrupts (1 to 32)
- Fast multiplier (single cycle) or small multiplier (for a smaller chip area and lower power, 32 cycles)
- **Little endian or big endian memory support**
- Little and big endian are two ways of storing multibyte data-types ( int, float, etc).
- In little endian machines, last byte of binary representation of the multibyte data-type is stored first.
- On the other hand, in big endian machines, first byte of binary representation of the multibyte data-type is stored first. a variable x with value **0x01234567** will be stored as following.



Big Endian



Little Endian

# ***Implementation Features***

- Optional Wakeup Interrupt Controller (WIC) to allow the processor to be powered down during sleep, while still allowing interrupt sources to wake up the system
- Very low gate count, which allows the design to be implemented in mixed signal semiconductor processes

# CORTEX M0 Supports Little endian **memory support**

| Register        | Value      |
|-----------------|------------|
| <b>Core</b>     |            |
| R0              | 0x20000010 |
| R1              | 0x20000120 |
| R2              | 0x00000080 |
| R3              | 0x00000000 |
| R4              | 0x04030201 |
| R5              | 0x08070605 |
| R6              | 0x00000000 |
| R7              | 0x00000000 |
| R8              | 0x00000000 |
| R9              | 0x00000000 |
| R10             | 0x00000000 |
| R11             | 0x00000000 |
| R12             | 0x00000000 |
| R13 (SP)        | 0x20000420 |
| R14 (LR)        | 0x000000DD |
| R15 (PC)        | 0x000001CC |
| + xPSR          | 0x41000000 |
| <b>Banked</b>   |            |
| <b>System</b>   |            |
| <b>Internal</b> |            |
| Mode            | Thread     |
| Stack           | MSP        |
| States          | 47         |
| Sec             | 0.00000392 |

```

1 ;additon 32 bit
2
3
4 PRESERVE8 ; Indicate the code here preserve
5 ; 8 byte stack alignment
6 THUMB ; Indicate THUMB code is used
7 AREA |.text|, CODE, READONLY
8
9 EXPORT __main
10 ; Start of CODE area
11 __main
12 LDR r0,=0x20000000 ; Source address
13 LDR r1,=0x20000120 ; Destination address
14 LDR r2,=128 ; number of bytes to copy, also
15 copy_loop ; acts as loop counter
16 LDmia r0!,{r4-r7} ; Read 4 words and increment r0
17 STmia r1!,{r4-r7} ; Store 4 words and increment r1
18 LDmia r0!,{r4-r7} ; Read 4 words and increment r0
19 STmia r1!,{r4-r7} ; Store 4 words and increment r1
20 LDmia r0!,{r4-r7}
21 STmia r1!,{r4-r7}
22 LDmia r0!,{r4-r7}
23 STmia r1!,{r4-r7}
24 SUBS r2, r2, #64
25 BNE copy_loop ;
26 stop_B stop
27 END

```

**Memory 2**

| Address:    | 0x20000000                   |
|-------------|------------------------------|
| 0x20000000: | 01 02 03 04 05 06 07 08 00 C |
| 0x20000010: | 00 00 00 00 00 00 00 00 00 C |
| 0x20000020: | 00 00 00 00 00 00 00 00 00 C |
| 0x20000030: | 00 00 00 00 00 00 00 00 00 C |
| 0x20000040: | 00 00 00 00 00 00 00 00 00 C |

[illegible]

# ***Debug Features***

- Halt mode debug. Allows the processor activity to stop completely so that register values can be accessed and modified.
- No overhead in code size and stack memory size.
- CoreSight technology. Allows memories and peripherals to be accessed from the debugger without halting the processor.
- It also allows a system-on-chip design with multiple processors to share a single debug connection.
- Supports JTAG connection and serial wire debug connections.
- The serial wire debug protocol can handle the same debug features as the JTAG, but it only requires two wires and is already supported by a number of debug solutions from various tools vendors.

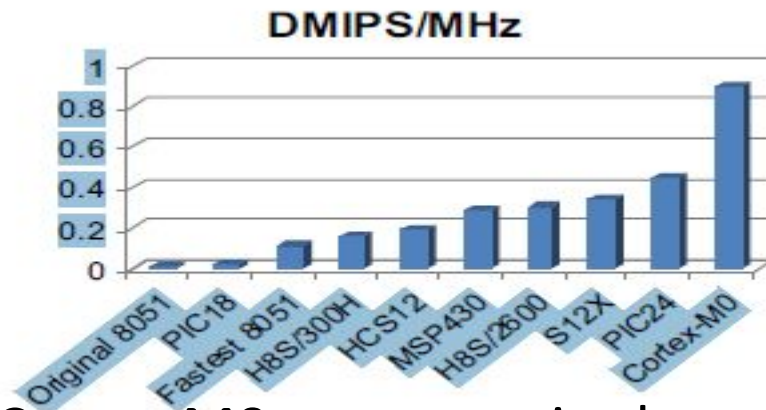
# Debug Features

- Configurable number of hardware breakpoints (from 0 to maximum of 4) and watchpoints (from 0 to maximum of 2). The chip manufacturer defines this during implementation.
- Software **breakpoints** can easily be set if the program is located in RAM (such as on a PC).
- Most debug probes support only **hardware breakpoints** if the program is located in flash memory.
- Breakpoint instruction support for an unlimited number of software breakpoints.
- All debug features can be omitted by chip vendors to allow minimum size implementations.

# Debug Features

- SW **breakpoints** can only be placed in RAM because they rely on modifying target memory.
- A HW (**Hardware**) **breakpoint** is set by programming a watchpoint unit to monitor the core busses for an instruction fetch from a specific memory location.

# *Advantages of the Cortex-M0 Processor*



**Energy Efficiency:** The Cortex-M0 processor is about the same size as a typical 16-bit processor and possibly several times bigger than some of the 8-bit processors.

However, it has much better performance than 16-bit and 8-bit architectures

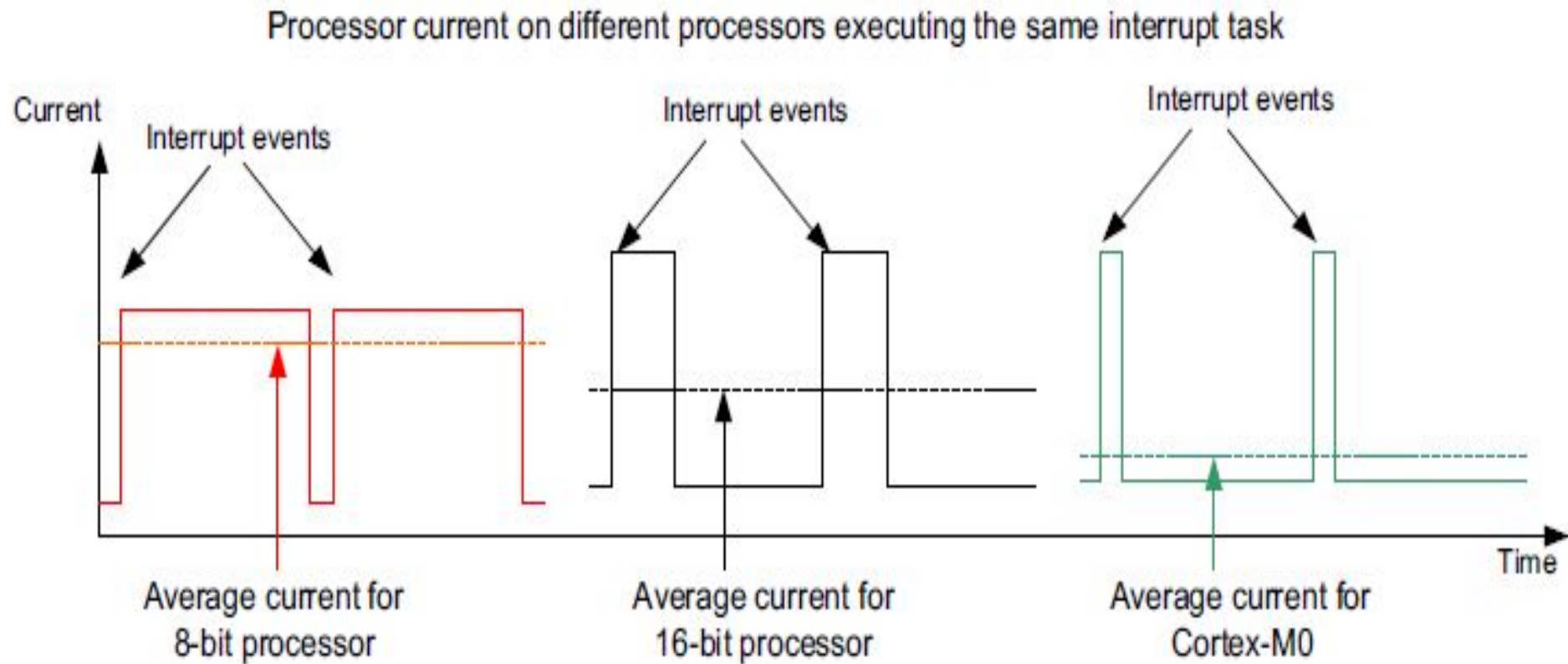
# Performance of Cortex M0 in Terms of DMIPS/MHz

Table 2.1: Dhrystone Performance Data Based on Information Available on the Internet

| Architecture   | Estimated DMIPS/MHz with Dhrystone 2.1                         |
|----------------|----------------------------------------------------------------|
| Original 80C51 | 0.0094                                                         |
| PIC18          | 0.01966                                                        |
| Fastest 8051   | 0.113                                                          |
| H8S/300H       | 0.16                                                           |
| HCS12          | 0.19                                                           |
| MSP430         | 0.288                                                          |
| H8S/2600       | 0.303                                                          |
| S12X           | 0.34                                                           |
| PIC24          | 0.445                                                          |
| Cortex-M0      | 0.896 (if a small multiplier is used, the performance is 0.85) |



# Average Current for different processors



# Microcontroller Current on different Architectures executing the same interrupt task

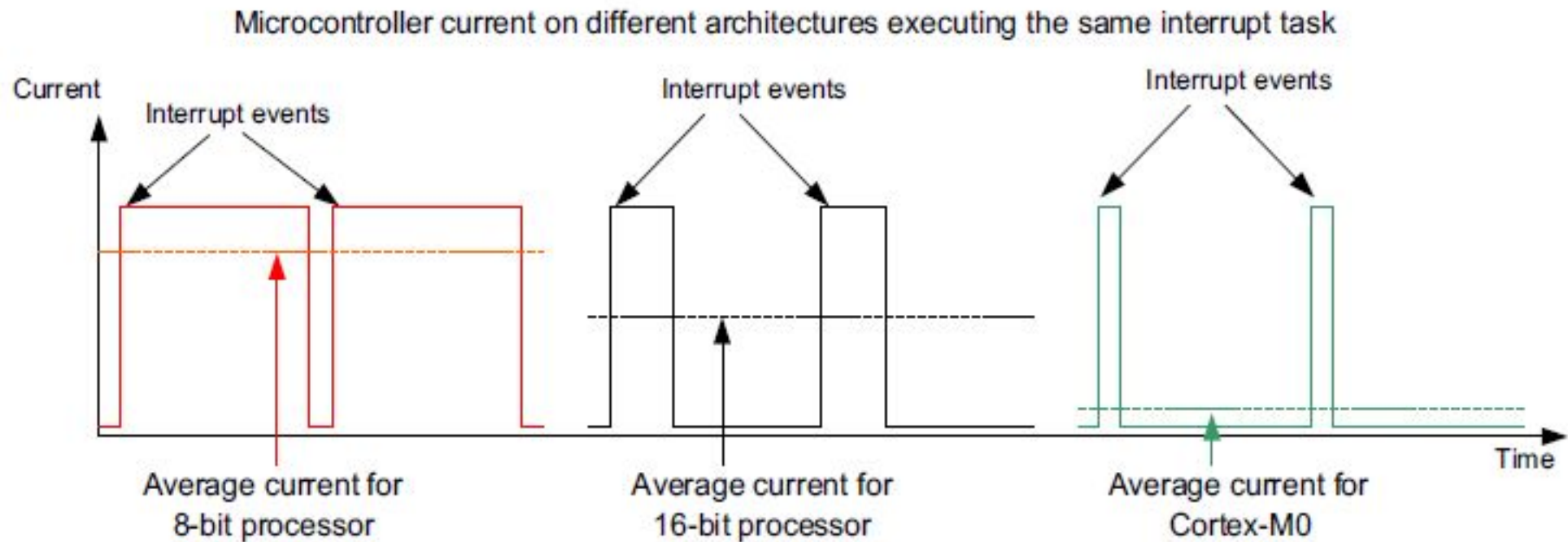
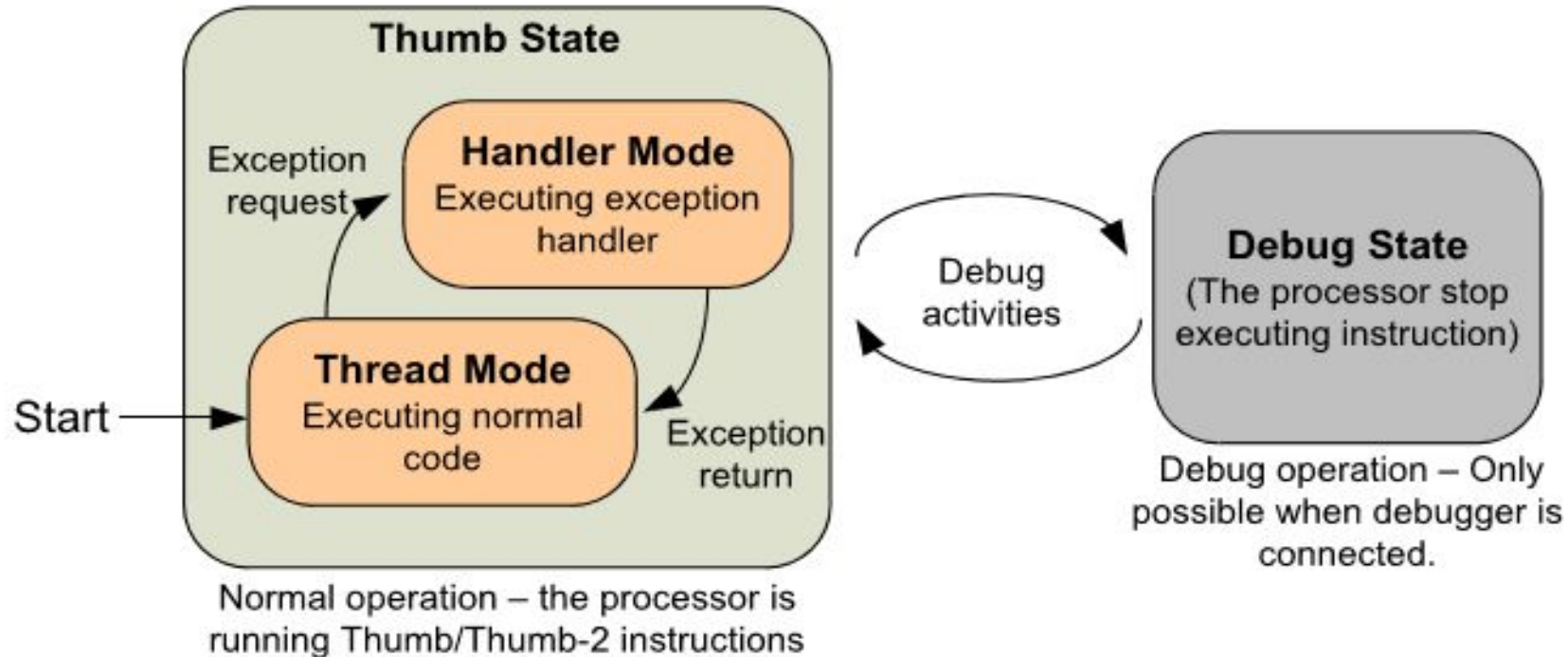


Figure 2.4:

At the chip level, the duty cycle of processor activity becomes more significant.

# PROGRAMMERS MODEL

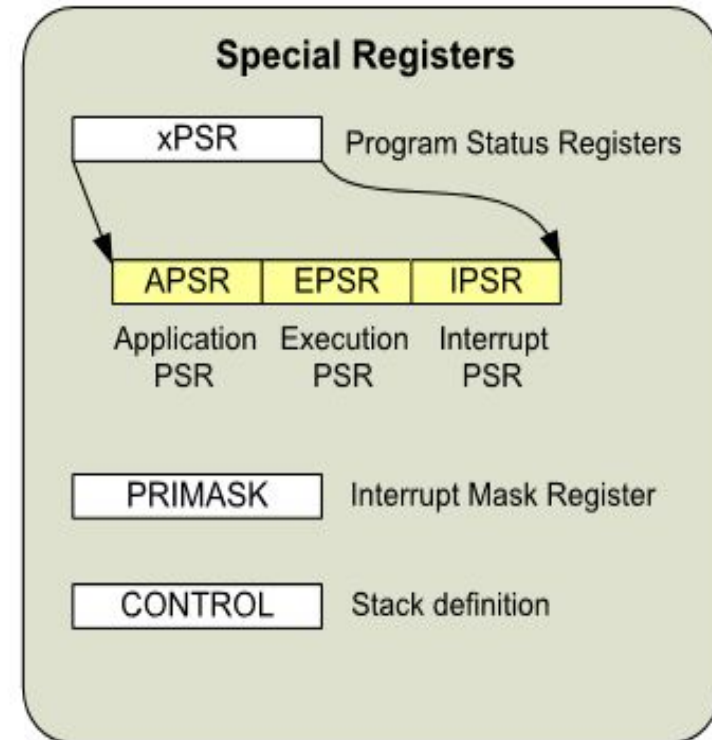
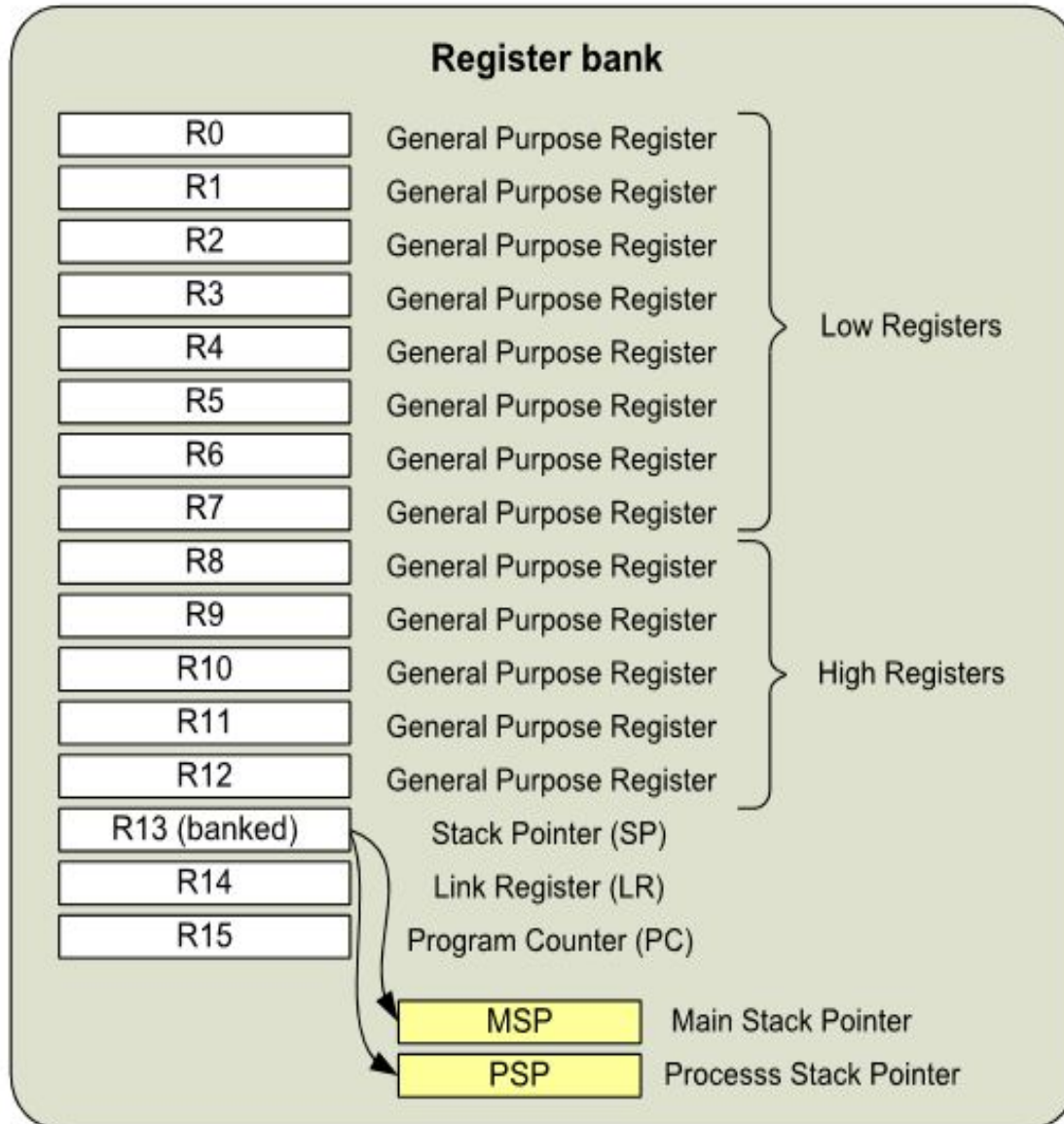


- The Cortex-M0 processor has two operation modes and two states.
- When the processor is running a program, it is in the Thumb state. In this state, it can be either in the Thread mode or the Handler mode.

# Registers and Special Registers

- To perform data processing and controls, a number of registers are required inside the processor core.
- If data from memory are to be processed, they have to be loaded from the memory to a register in the register bank, processed inside the processor, and then written back to the memory if needed.
- This is commonly called a “load-store architecture.” By having a sufficient number of registers in the register bank, this mechanism is easy to use and is C friendly.
- It is easy for C compilers to compile a C program into machine code with good performance. By using internal registers for short-term data storage, the amount of memory accesses can be reduced.
- The Cortex-M0 processor provides a register bank of 13 general-purpose 32-bit registers and a number of special registers

# Register Banks and Special



# Register Banks and Special Registers

| Register    | Value      |
|-------------|------------|
| <b>Core</b> |            |
| R0          | 0x20000010 |
| R1          | 0x20000120 |
| R2          | 0x00000080 |
| R3          | 0x00000000 |
| R4          | 0x04030201 |
| R5          | 0x08070605 |
| R6          | 0x00000000 |
| R7          | 0x00000000 |
| R8          | 0x00000000 |
| R9          | 0x00000000 |
| R10         | 0x00000000 |
| R11         | 0x00000000 |
| R12         | 0x00000000 |
| R13 (SP)    | 0x20000420 |
| R14 (LR)    | 0x000000DD |
| R15 (PC)    | 0x000001CC |
| xPSR        | 0x41000000 |
| N           | 0          |
| Z           | 1          |
| C           | 0          |
| V           | 0          |
| T           | 1          |
| ISR         | 0          |

# Register Banks and Special Registers

- R0-R12 Registers R0 to R12 are for general uses. Because of the limited space in the 16-bit Thumb instructions, many of the Thumb instructions can only access R0 to R7, which are also called the low registers, whereas some instructions, like MOV (move), can be used on all registers.
- When using these registers with ARM development tools such as the ARM assembler, you can use either uppercase (e.g., R0) or lowercase (e.g., r0) to specify the register to be used.
- The initial values of R0 to R12 at reset are undefined.

# Stack Pointer

- R13, Stack Pointer (SP) R13 is the stack pointer. It is used for accessing the stack memory via PUSH and POP operations. There are physically two different stack pointers in Cortex-M0.

PRESERVE8 ; Indicate the code here preserve

; 8 byte stack alignment

THUMB ; Indicate THUMB code is used

AREA |.text|, CODE, READONLY

EXPORT \_\_main

; Start of CODE area

\_\_main

LDR r3,=0x20000100

LDR r0,=0x20000050

LDMIA r3!,{r1,r2}

MOV SP,r0

PUSH {r1,r2}

POP {r4,r5}

stop B stop

END ; End of file



# Stack Pointer

- There are physically two different stack pointers in Cortex-M0.
- The main stack pointer (MSP, or SP\_main in ARM documentation) is the default stack pointer after reset, and it is used when running exception handlers.
- The process stack pointer (PSP, or SP\_process in ARM documentation) can only be used in Thread mode (when not handling exceptions).
- The stack pointer selection is determined by the CONTROL register, one of the special registers that will be introduced later. When using ARM development tools, you can access the stack pointer using either “R13” or “SP.” Both uppercase and lowercase (e.g., “r13” or “sp”) can be used.

# Stack Pointer

- Only one of the stack pointers is visible at a given time. However, you can access to the MSP or PSP directly when using the special register access instructions MRS and MSR.
- In such cases, the register names “MSP” or “PSP” should be used
- ;additon 32 bit
- PRESERVE8 ; Indicate the code here preserve
- ; 8 byte stack alignment
- THUMB     ; Indicate THUMB code is used
- AREA   |.text|, CODE, READONLY
- 
- EXPORT \_\_main
- ; Start of CODE area
-

# Stack Pointer

- Only one of the stack pointers is visible at a given time. However, you can access to the MSP or PSP directly when using the special register access instructions MRS and MSR.
- In such cases, the register names “MSP” or “PSP” should be used
- PRESERVE8
- THUMB ; Indicate THUMB code is used
- AREA [.text], CODE, READONLY
- EXPORT \_\_main
- ;\_\_main
- LDR r0,=0x20000000 ; Source address
- LDR r1,=0x20000040 ; Destination address
- LDR r2,=10; number of bytes to copy
- MRS r0,MSP
- stop B stop
- END.
-

|            |            |
|------------|------------|
| Core       |            |
| R0         | 0x20000420 |
| R1         | 0x20000040 |
| R2         | 0x0000000A |
| R3         | 0x00000000 |
| R4         | 0x00000000 |
| R5         | 0x00000000 |
| R6         | 0x00000000 |
| R7         | 0x00000000 |
| R8         | 0x00000000 |
| R9         | 0x00000000 |
| R10        | 0x00000000 |
| R11        | 0x00000000 |
| R12        | 0x00000000 |
| R13 (SP)   | 0x20000420 |
| R14 (LR)   | 0x000000DD |
| R15 (PC)   | 0x000001CE |
| + xPSR     | 0x41000000 |
| + Banked   |            |
| + System   |            |
| - Internal |            |
| Mode       | Thread     |
| Stack      | MSP        |
| States     | 43         |
| Sec        | 0.00000358 |

```

0x000001D0 0000 MOVS r0,r0
0x000001D2 2000 MOVS r0,#0x00

```

findsum.asm   Loadstoremultiplebytes.asm   add.asm   startup\_NUC1xx.s

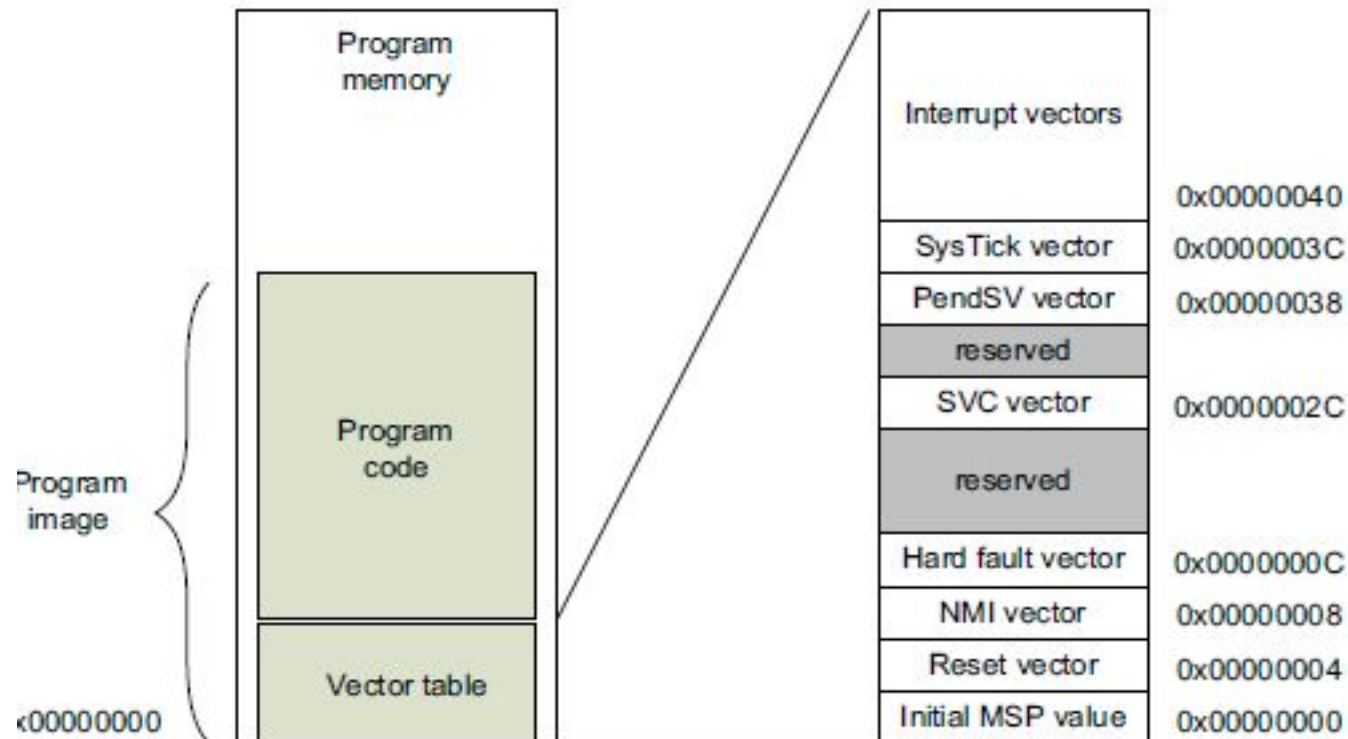
```

1 ;additon 32 bit
2
3
4 PRESERVE8 ; Indicate the code here preserve
5 ; 8 byte stack alignment
6 THUMB ; Indicate THUMB code is used
7 AREA |.text|, CODE, READONLY
8
9 EXPORT __main
10 ; Start of CODE area
11
12
13 __main
14 LDR r0,=0x20000000 ; Source address
15 LDR r1,=0x20000040 ; Destination address
16 LDR r2,=10; number of bytes to copy
17 MRS r0,MSP
18
19 stop B stop
20
21 END

```

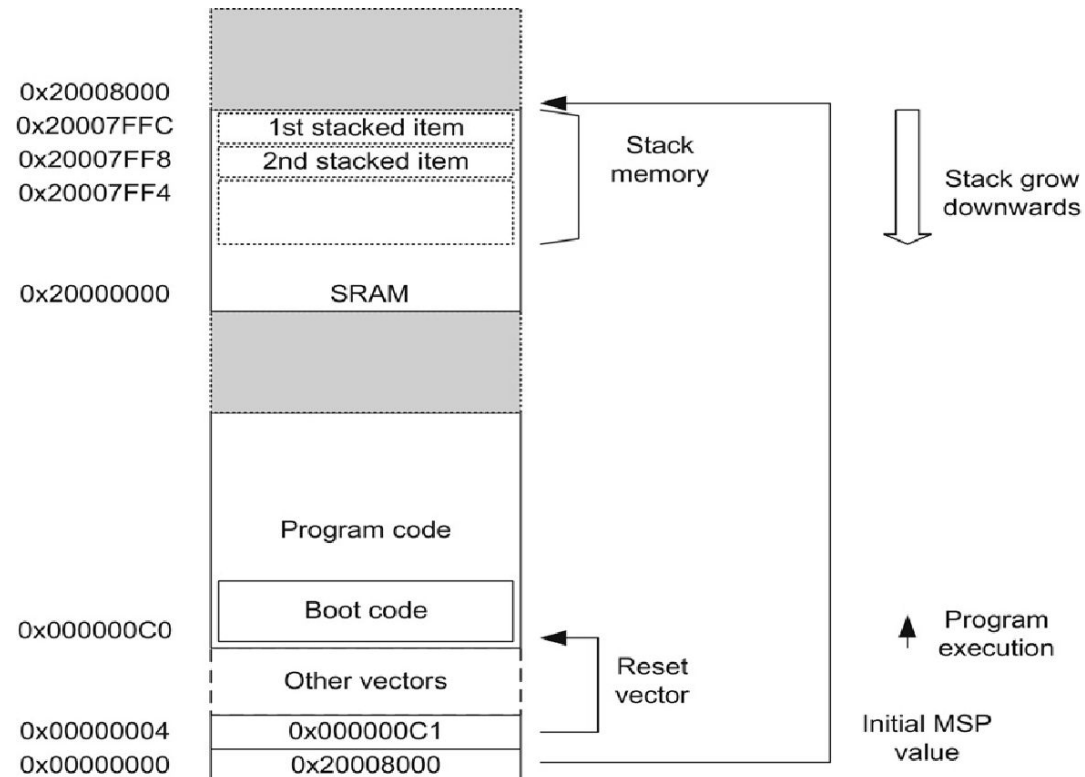
# Stack Pointer

- In ARM processors, PUSH and POP are always 32-bit accesses because the registers are 32-bit, and the transfers in stack operations must be aligned to a 32-bit word boundary.
- The initial value of MSP is loaded from the first 32-bit word of the vector table from the program memory during the startup sequence.



# Stack Pointer

- The lowest two bits of the stack pointers are always zero, and writes to these two bits are ignored. Check the address of the stack pointers.
- If we start at 0x20008000, the next address would be 0x20007FFC (0x20007FF1100), Next address would obviously be 0x20007FF8, 0x20007FF4 (0100), it goes on.



# Stack Pointer

- The initial value of PSP is undefined. It is not necessary to use the PSP.
- In many applications, the system can completely rely on the MSP.
- The PSP is normally used in designs with an OS, where the stack memory for OS Kernel and the thread level application code must be separated.

# R14-Link Register

- R14, Link Register (LR) R14 is the Link Register.
- The Link Register is used for storing the return address of a subroutine or function call.
- At the end of the subroutine or function, the return address stored in LR is loaded into the program counter so that the execution of the calling program can be resumed.
- In the case where an exception occurs, the LR also provides a special code value, which is used by the exception return mechanism.
- When using ARM development tools, you can access to the Link Register using either “R14” or “LR.”



# R14-Link Register

- Both upper and lowercase (e.g., “r14” or “lr”) can be used.
- Although the return address in the Cortex-M0 processor is always an even address (bit[0] is zero because the smallest instructions are 16-bit and must be half-word aligned), bit zero of LR is readable and writeable.
- In the ARMv6-M architecture, some instructions require bit zero of a function address set to 1 to indicate Thumb state.

# R14-Link Register

## FUNCTION

SUB SP, SP, #0x8 ; Reserve 2 words of stack;(8 bytes) for local variables

;Data processing in function

MOVS r0, #0x12 ; set a dummy value

STR r0, [sp, #0] ; Store 0x12 in 1st local variable

STR r0, [sp, #4] ; Store 0x12 in 2nd local variable

LDR r1, [sp, #0] ; Read from 1st local variable

LDR r2, [sp, #4] ; Read from 2nd local variable

ADD SP, SP, #0x8; Restore SP to original position

BX LR;(Branch to address stored in the Link;Register. This instruction is often used for;function return.)

\_\_main

BL FUNCTION;branch and link to the address of FUNCTION

ADDS R2, R2, #10; Add R0 a

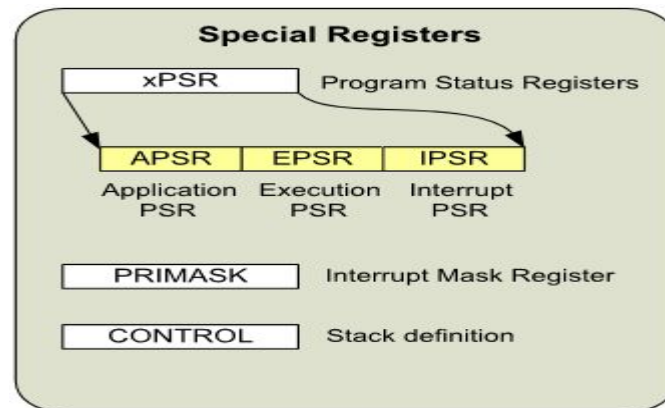
END

# R15-Program Counter

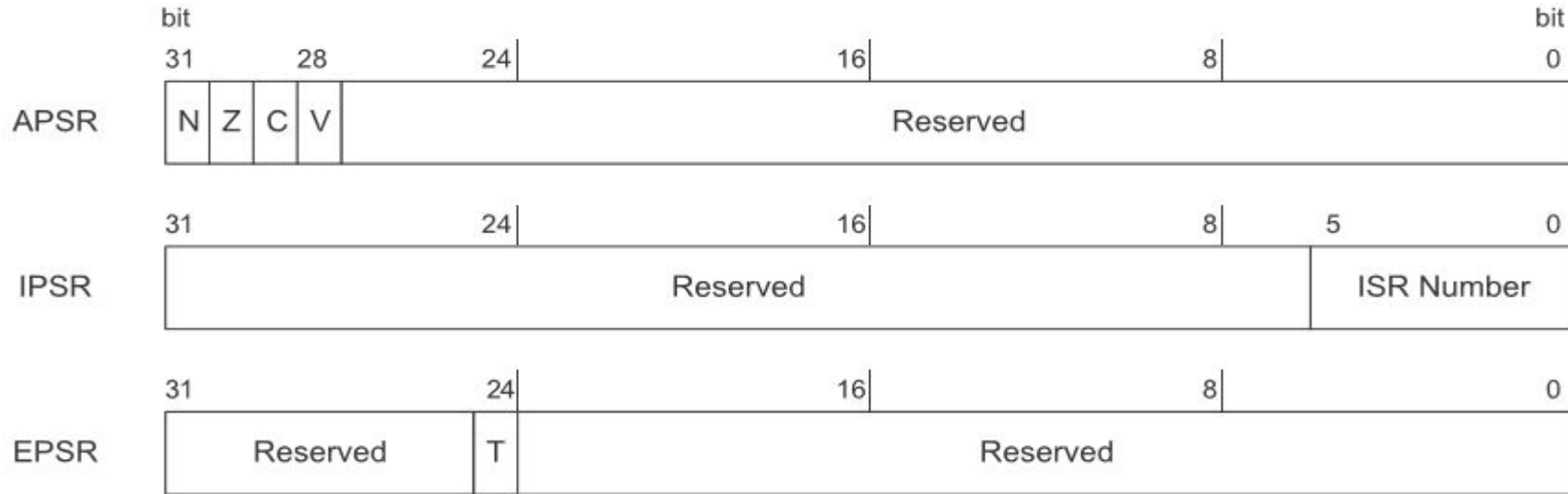
- R15, Program Counter (PC) R15 is the Program Counter. It is readable and writeable. A read returns the current instruction address plus four (this is caused by the pipeline nature of the design).
- Writing to R15 will cause a branch to take place (but unlike a function call, the Link Register does not get updated).
- In the ARM assembler, you can access the Program Counter, using either “R15” or “PC,” in either upper or lower case (e.g., “r15” or “pc”). Instruction addresses in the Cortex-M0 processor must be aligned to half-word address, which means the actual bit zero of the PC should be zero all the time.
- However, when attempting to carry out a branch using the branch instructions (BX or BLX), the LSB of the PC should be set to 1. This is to indicate that the branch target is a Thumb program region. Otherwise, it can imply trying to switch the processor to ARM state (depending on the instruction used), which is not supported and will cause a fault exception.

# xPSR

- xPSR, combined Program Status Register The combined Program Status Register provides information about program execution and the ALU flags.
- It consists of the following three Program Status Registers (PSRs)
  - Application PSR (APSR)
  - Interrupt PSR (IPSR)
  - Execution PSR (EPSR)



# APSR, IPSR and EPSR



**Figure 3.3:**  
APSR, IPSR, and EPSR.

# xPSR

- The APSR contains the ALU flags: N (negative flag), Z (zero flag), C (carry or borrow flag), and V (overflow flag). These bits are at the top 4 bits of the APSR. The common use of these flags is to control conditional branches.
- The IPSR contains the current executing interrupt service routine (ISR) number. Each exception on the Cortex-M0 processor has a unique associated ISR number (exception type).
- This is useful for identifying the current interrupt type during debugging and allows an exception handler that is shared by several exceptions to know what exception it is serving.
- The EPSR on the Cortex-M0 processor contains the T-bit, which indicates that the processor is in the Thumb state.
- On the Cortex-M0 processor, this bit is normally set to 1 because the Cortex-M0 only supports the Thumb state.

# CONTROL REGISTERS

- During running of an exception handler (when the processor is in Handler mode), only the MSP is used, and the CONTROL register reads as zero.
- The CONTROL register can only be changed in Thread mode or via the exception entrance and return mechanism. Bit 0 of the CONTROL register is reserved to maintain compatibility with the Cortex-M3 processor.
- In the Cortex-M3 processor, bit 0 can be used to switch the processor to User mode (non-privileged mode). This feature is not available in the Cortex-M0 processor.