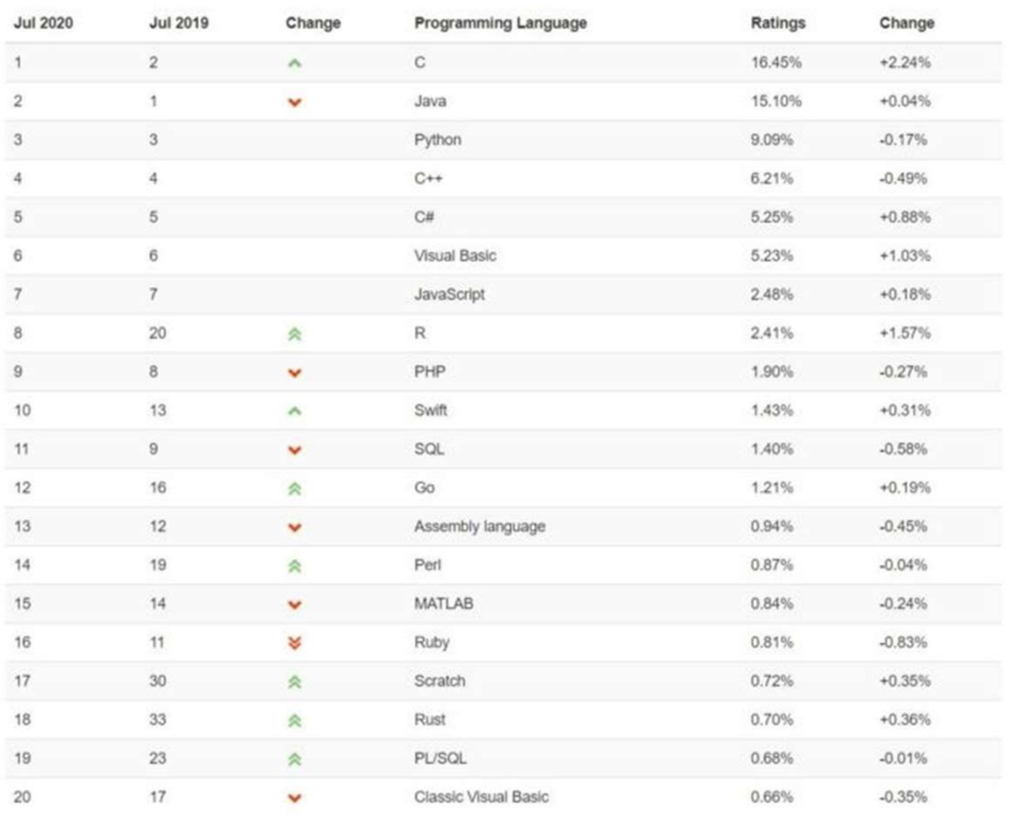
PART 4: Embedded Programming STM32

Assembly Language

ISEN
ALL IS DIGITAL!
OUEST
yncréa

Ranking Languages Most Used



Jul 2020	Jul 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	16.45%	+2.24%
2	1	▼	Java	15.10%	+0.04%
3	3		Python	9.09%	-0.17%
4	4		C++	6.21%	-0.49%
5	5		C#	5.25%	+0.88%
6	6		Visual Basic	5.23%	+1.03%
7	7		JavaScript	2.48%	+0.18%
8	20	▲▲	R	2.41%	+1.57%
9	8	▼	PHP	1.90%	-0.27%
10	13	▲	Swift	1.43%	+0.31%
11	9	▼	SQL	1.40%	-0.58%
12	16	▲▲	Go	1.21%	+0.19%
13	12	▼	Assembly language	0.94%	-0.45%
14	19	▲▲	Perl	0.87%	-0.04%
15	14	▼	MATLAB	0.84%	-0.24%
16	11	▼▼	Ruby	0.81%	-0.83%
17	30	▲▲	Scratch	0.72%	+0.35%
18	33	▲▲	Rust	0.70%	+0.36%
19	23	▲▲	PL/SQL	0.68%	-0.01%
20	17	▼	Classic Visual Basic	0.66%	-0.35%

Découvrez le nouveau classement de l'indice TIOBE pour le mois de juillet 2020. © TIOBE.

Assembly Language

- ✓ It is the language that is as close as possible to the transistors
- ✓ It uses the instructions of the microcontroller → the program can be different for each µController
- ✓ It is not recommended to write a full program in assembly language. C is preferred most of the time. But a C program is compiled in assembly language and it can be necessary to decode assembly language to understand bugs

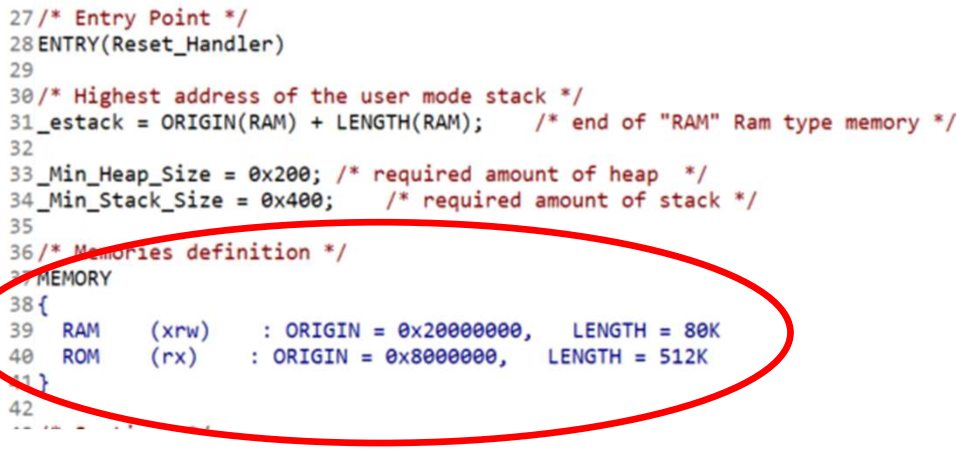
Assembly Language Files

- ✓ The assembly code is saved in a .s file
- ✓ Incorporated files are saved in .h files
- ✓ The analogy in C: .c files are for the C program and the .h ones are for the header files

Assembly Language Mapping

- ✓ Mapping files define memory boundaries
- ✓ This is the way the compiler knows the RAM memory starts at address 0x20000000, and the Flash at 0x8000000
- ✓ Give size of stack and heap and much more…
- ✓ For legacy reason, rom is the term used for program location even if today the main technology is flash

STM32l152RETX_FLASH.ld

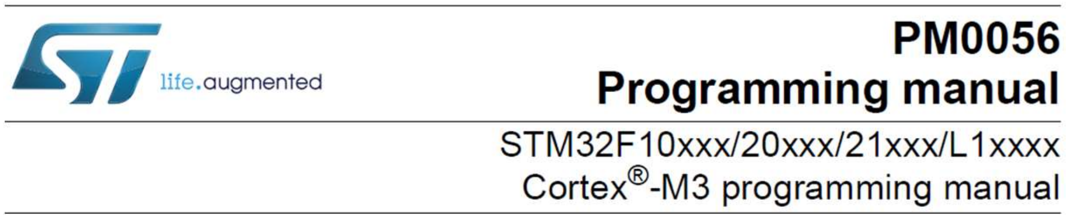


```
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM);    /* end of "RAM" Ram type memory */

_Min_Heap_Size = 0x200; /* required amount of heap  */
_Min_Stack_Size = 0x400;    /* required amount of stack */

/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 80K
  ROM    (rx)    : ORIGIN = 0x8000000,   LENGTH = 512K
}

```

DO NOT MODIFY THESE FILES

STM32 Instructions

- ✓ To program in assembly language is in fact to write a program with the instructions available for a given µController
- ✓ All available instructions are described in the programming Manual*

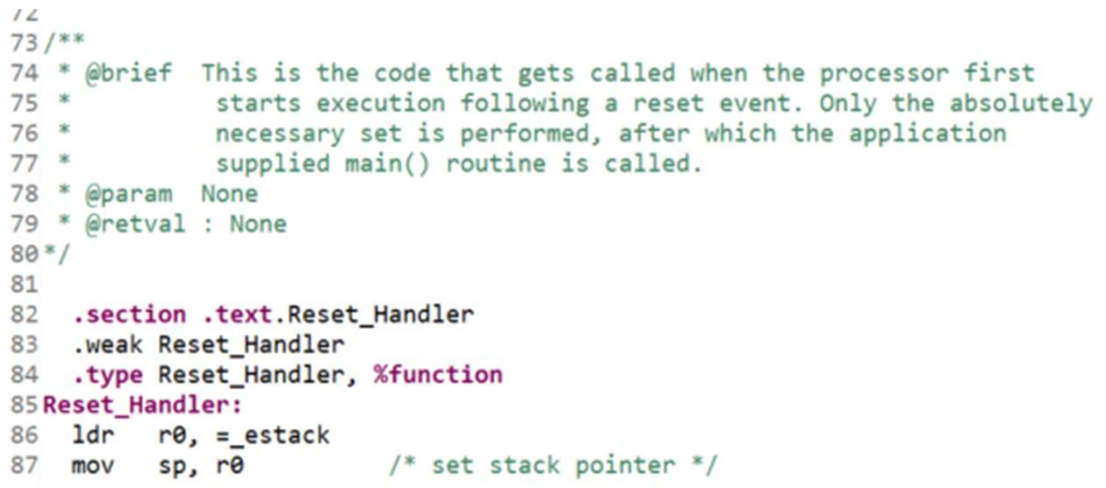

PM0056
Programming manual
STM32F10xxx/20xxx/21xxx/L1xxxx
Cortex®-M3 programming manual

- ✓ Can be downloaded on the STMicroelectronics web site https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l1-series/stm32l151-152/stm32l152re.html#resource

Program Start-up

- ✓ At start up, the first program that is called is the startup_stm32l152retx.s file that is written in assembly language.

- ✓ The first function that is called is the reset_Handler. Address of this interrupt is equal to 0x00000004



```
/**
 * @brief  This is the code that gets called when the processor first
 *          starts execution following a reset event. Only the absolutely
 *          necessary set is performed, after which the application
 *          supplied main() routine is called.
 * @param  None
 * @retval : None
*/

  .section .text.Reset_Handler
  .weak Reset_Handler
  .type Reset_Handler, %function
Reset_Handler:
  ldr   r0, =_estack
  mov   sp, r0          /* set stack pointer */
```

The syntax depends on compiler

- ✓ Gcc (used by STM32CuibeIDE)
- ✓ Keil
- ✓ IAR

Assembly Program Skeleton

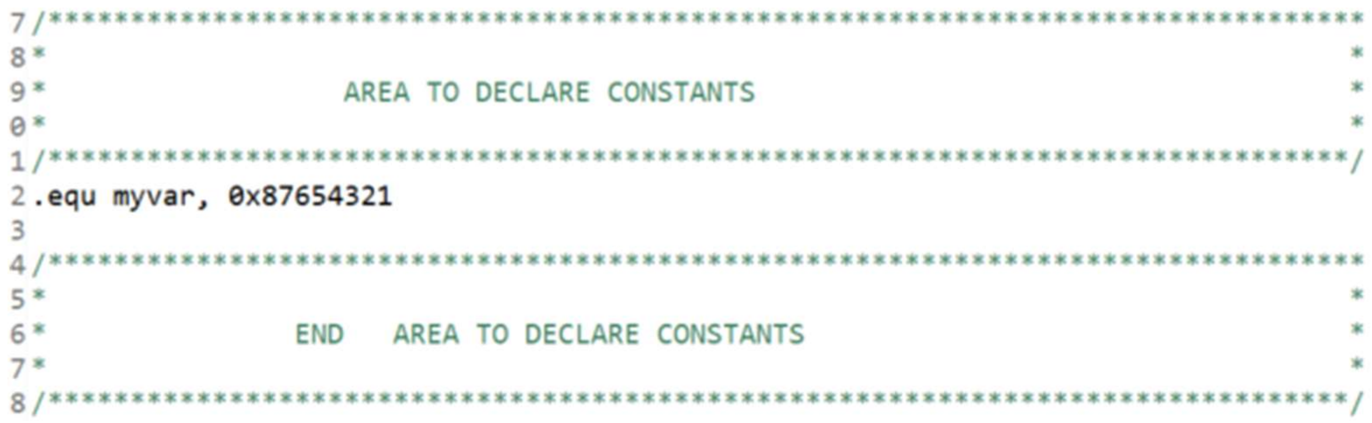
This file has different sections for:

- Symbols declaration,
- Variables declaration,
- Constants declaration,
- Sub-programs declaration,
- Main program,
- Interrupt* sub-programs declaration,
- Interrupt* vectors declaration.

*Interrupts will be seen at the end of the course

Area To Declare Symbols

This area is used to declare symbols

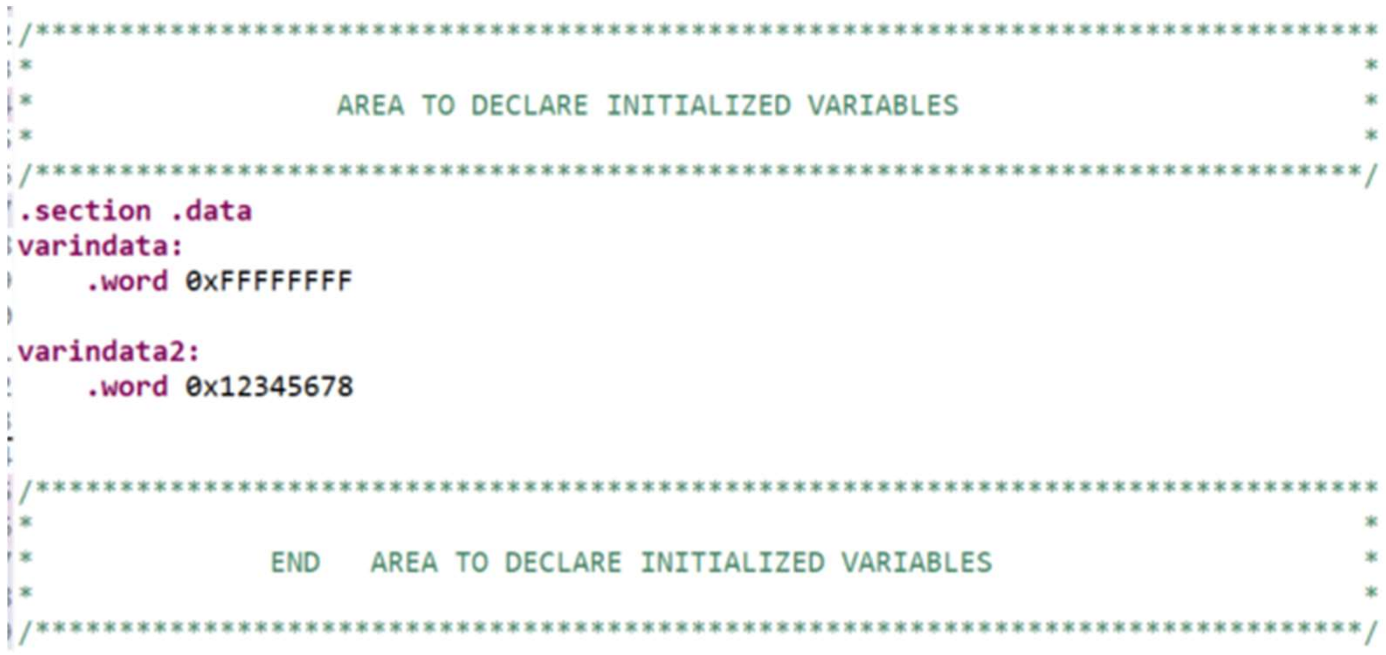


```
/****************************************************************************
*                                                                           *
*                   AREA TO DECLARE CONSTANTS                               *
*                                                                           *
****************************************************************************/
.equ myvar, 0x87654321

/****************************************************************************
*                                                                           *
*                END   AREA TO DECLARE CONSTANTS                            *
*                                                                           *
****************************************************************************/
```

Symbols are used to ease source code reading. For example, the instruction `ldr r3, =myvar` loads 0x87654321 into register r3

Area to Declare Initialized Variables

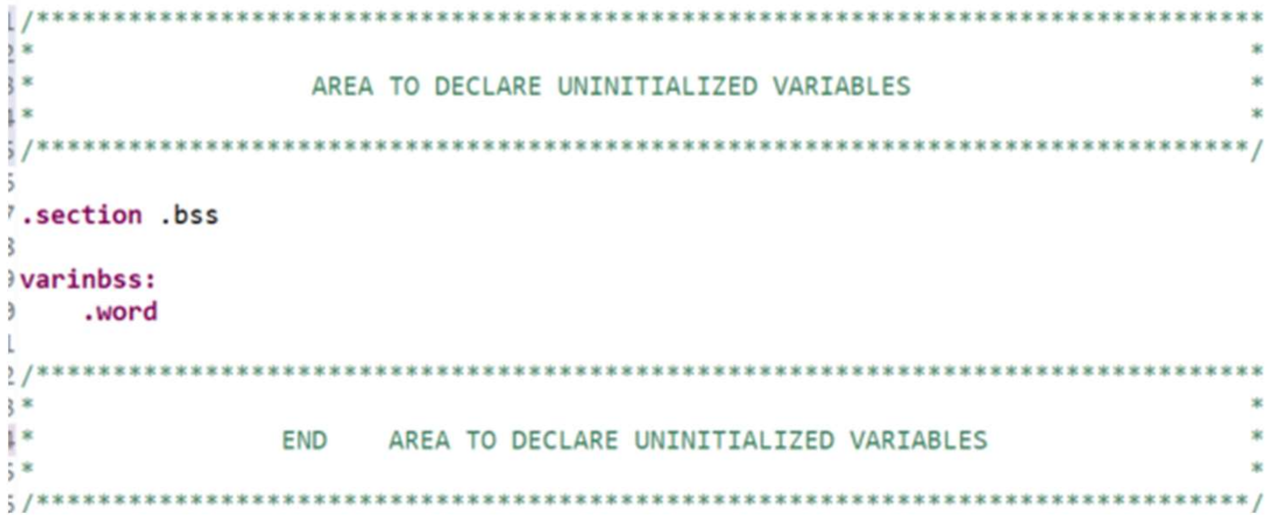
- ✓ This area is inside the SRAM in the data section
- ✓ A name is associated to a memory space



```
/*************************************************************************
*                                                                        *
*                   AREA TO DECLARE INITIALIZED VARIABLES                *
*                                                                        *
*************************************************************************/
.section .data
varindata:
    .word 0xFFFFFFFF

varindata2:
    .word 0x12345678


/*************************************************************************
*                                                                        *
*               END   AREA TO DECLARE INITIALIZED VARIABLES              *
*                                                                        *
*************************************************************************/
```

Area to Declare UnInitialized Variables

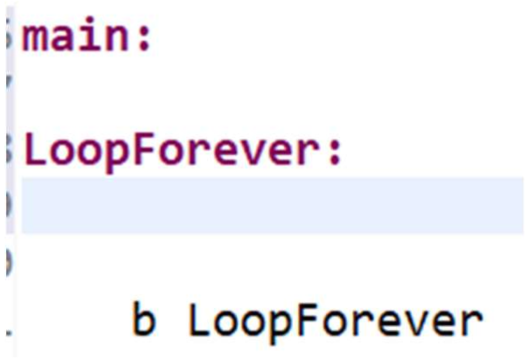
- ✓ This area is inside the SRAM in the bss section
- ✓ A name is associated to a memory space



```
/***************************************************************************
*                                                                         *
*                   AREA TO DECLARE UNINITIALIZED VARIABLES               *
*                                                                         *
***************************************************************************/

.section .bss

varinbss:
    .word

/***************************************************************************
*                                                                         *
*                 END    AREA TO DECLARE UNINITIALIZED VARIABLES          *
*                                                                         *
***************************************************************************/
```

Main Program

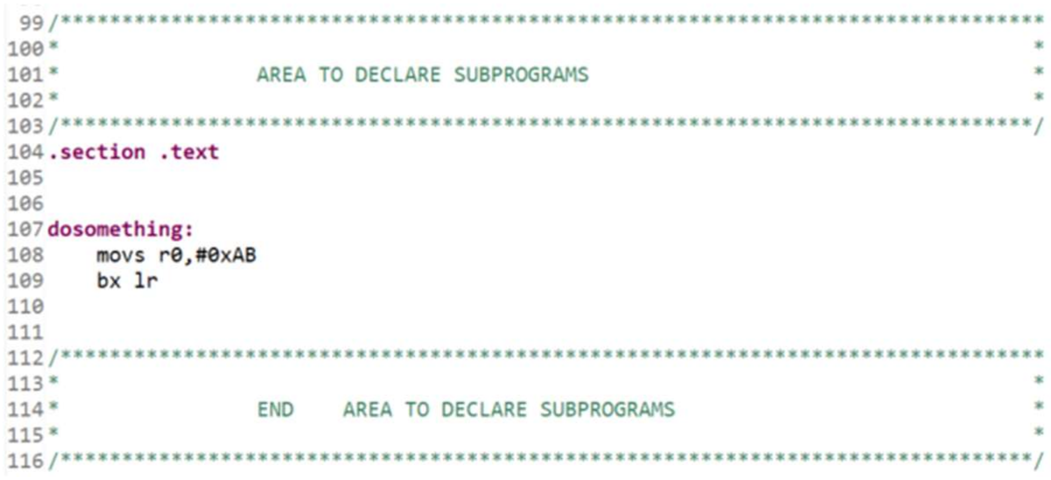
- ✓ The main program starts at address 0x0800 0000.



```
main:

LoopForever:


        b  LoopForever
```

- ✓ main is the starting point of the program
- ✓ In an embedded system, an infinite loop is always needed. As long as the system is powered up, the main program must run

Area To Declare Subprograms



```
/*****************************************************************************
*                                                                           *
*                  AREA TO DECLARE SUBPROGRAMS                              *
*                                                                           *
*****************************************************************************/
.section .text


dosomething:
    movs r0,#0xAB
    bx lr


/*****************************************************************************
*                                                                           *
*                  END    AREA TO DECLARE SUBPROGRAMS                       *
*                                                                           *
*****************************************************************************/
```

- ✓ Subprograms are part of the program
- ✓ A subprogram ends with the instruction bx lr
- ✓ Un subprogram is called with the instruction bl *something*

Instructions information

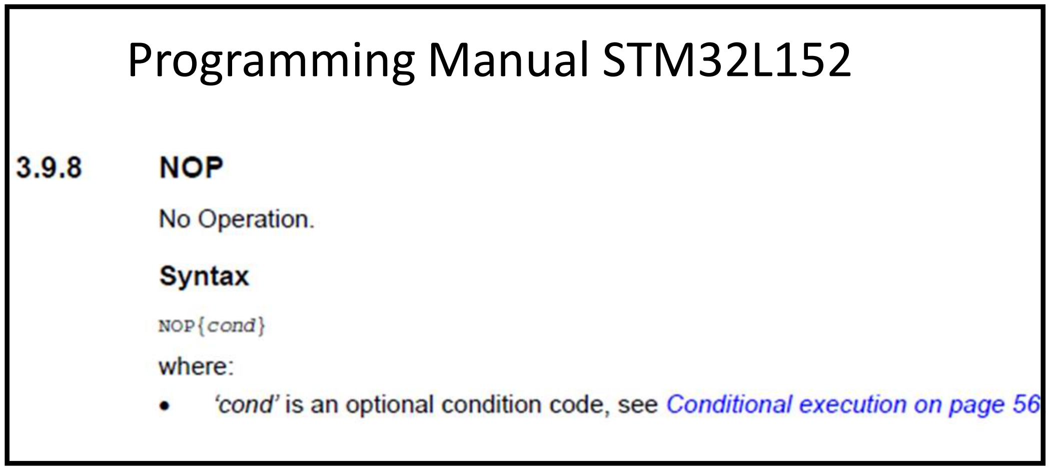
The Programming Manual provides the following information:

- ✓ Syntax
- ✓ Options
- ✓ Operands
- ✓ Result
- ✓ Flags of the PSR affected by the operation
- ✓ Examples of Assembly code

The ARMv7-M Architecture Reference Manual provides:

- ✓ Instruction encoding

NOP Instruction



Programming Manual STM32L152

3.9.8 NOP

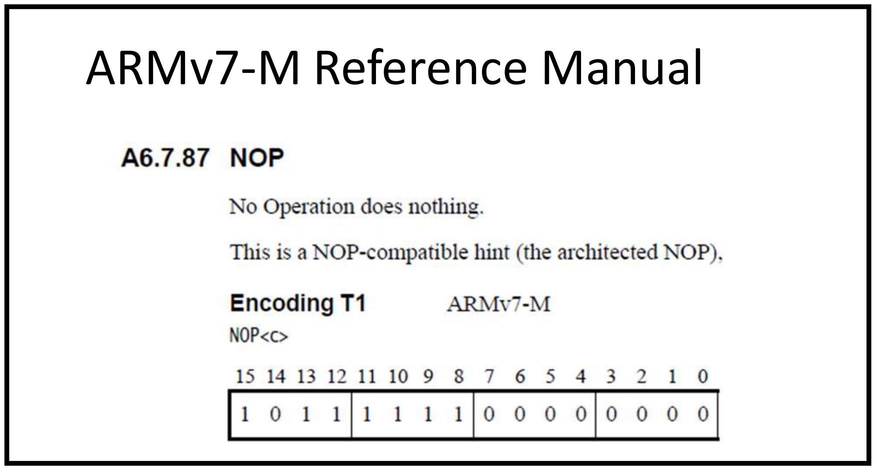
No Operation.

Syntax

`NOP{cond}`

where:

- *'cond'* is an optional condition code, see *Conditional execution on page 56*



ARMv7-M Reference Manual

A6.7.87 NOP

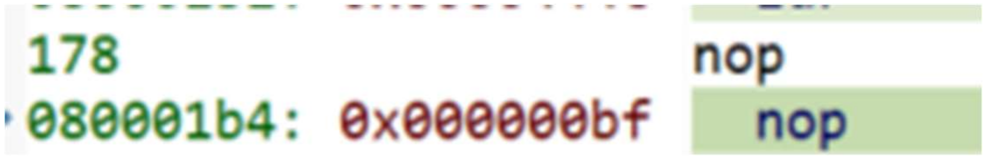
No Operation does nothing.

This is a NOP-compatible hint (the architected NOP).

Encoding T1 ARMv7-M

`NOP<c>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0



```
178                              nop
080001b4:  0x000000bf              nop
```

A nop in the program is encoded as 0xBF00