Keil MDK-ARM Tutorial

In this tutorial we will see how to use Keil MDK-ARM tool.

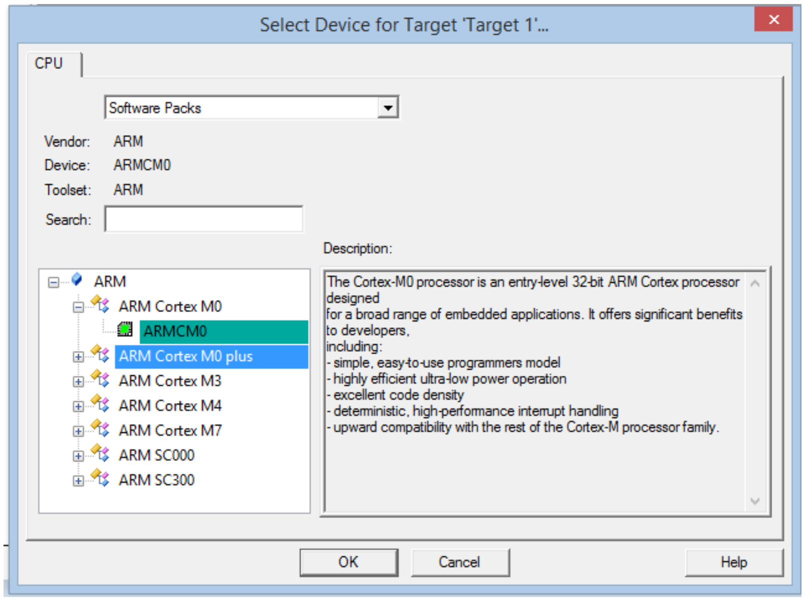
Step 1: Downloading and installing Keil MDK-ARM

- Go to https://www.keil.com/download/product/ and click on MDK-ARM
- Give your details and submit.
- Save **MDK516A.EXE** to your computer and install it.

Step 2: Creating a Project

Open the Keil µvision5 shortcut. Go to project>new µvision project.

After specifying the project name, you need to select the processor from the list (as shown in Fig 1) and click OK.



Software Packs

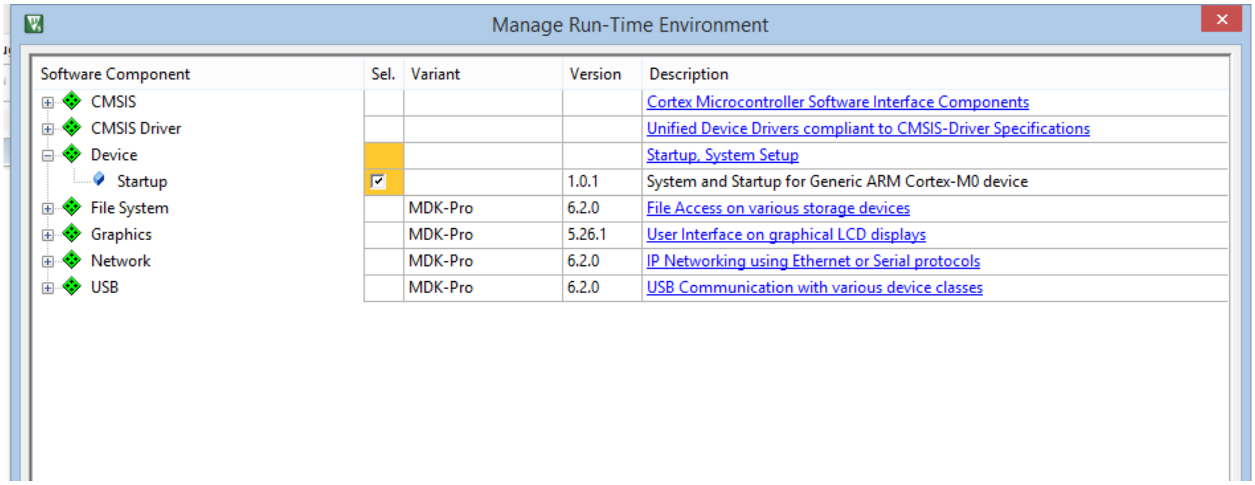
Vendor: ARM
Device: ARMCM0
Toolset: ARM
Search:

Description:

The Cortex-M0 processor is an entry-level 32-bit ARM Cortex processor designed
for a broad range of embedded applications. It offers significant benefits to developers,
including:
- simple, easy-to-use programmers model
- highly efficient ultra-low power operation
- excellent code density
- deterministic, high-performance interrupt handling
- upward compatibility with the rest of the Cortex-M processor family.

Fig 1

Now you go to the design part and check the Startup (See Fig2). Then click on resolve in order to add the startup files to your project.



Manage Run-Time Environment

Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Components
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Device				Startup, System Setup
Startup	☑		1.0.1	System and Startup for Generic ARM Cortex-M0 device
File System		MDK-Pro	6.2.0	File Access on various storage devices
Graphics		MDK-Pro	5.26.1	User Interface on graphical LCD displays
Network		MDK-Pro	6.2.0	IP Networking using Ethernet or Serial protocols
USB		MDK-Pro	6.2.0	USB Communication with various device classes

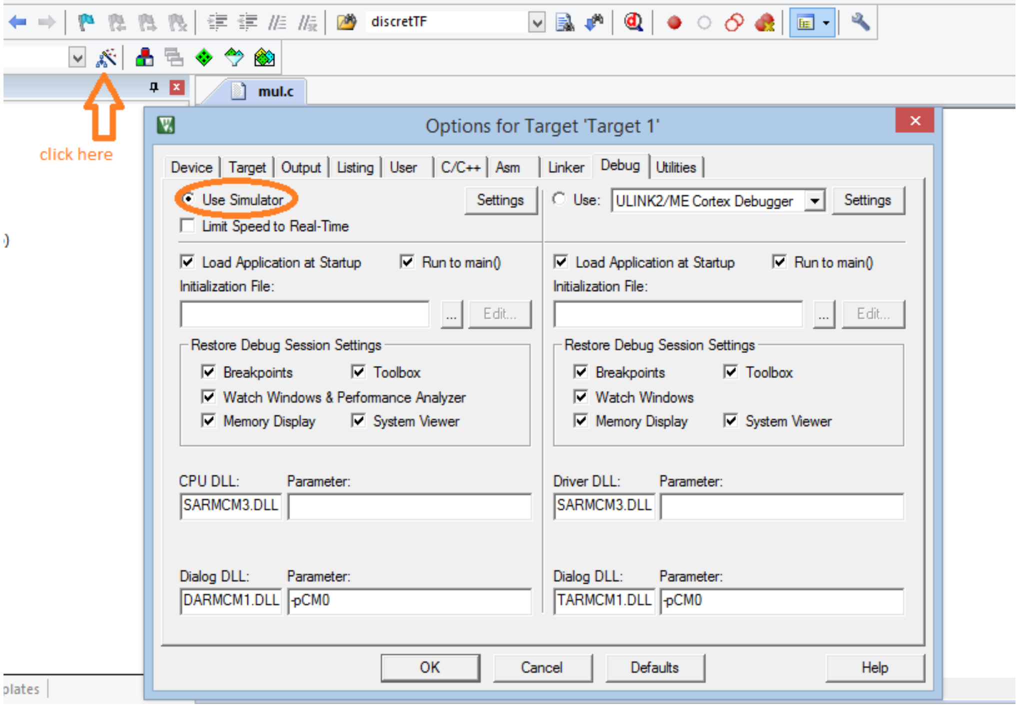
Fig 2

Step 3: Add the file

Right click on your Source Group 1 (in the Project window) and add a new file (C/C++). Write your program in the new created file.

Step 4: Check the Options for target

Click on options for target (See Fig3) and go to debug tab, select *use simulator*.



click here

Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

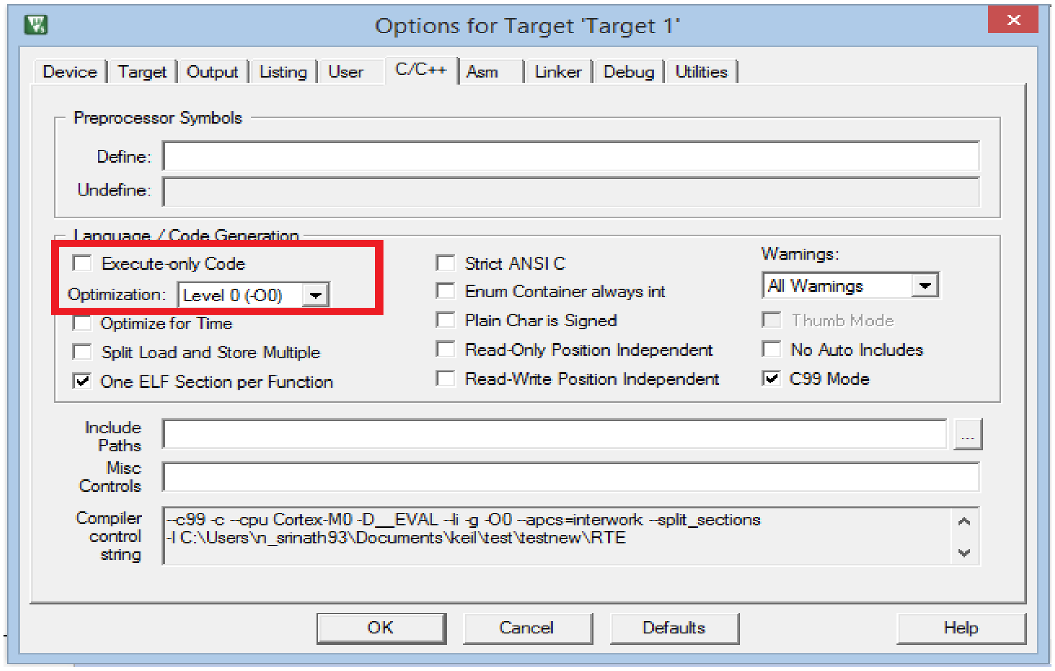
● Use Simulator Settings
□ Limit Speed to Real-Time
☑ Load Application at Startup ☑ Run to main()
Initialization File:
... Edit...
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display ☑ System Viewer
CPU DLL: SARMCM3.DLL Parameter:
Dialog DLL: DARMCM1.DLL Parameter: -pCM0

○ Use: ULINK2/ME Cortex Debugger Settings
☑ Load Application at Startup ☑ Run to main()
Initialization File:
... Edit...
Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows
☑ Memory Display ☑ System Viewer
Driver DLL: SARMCM3.DLL Parameter:
Dialog DLL: TARMCM1.DLL Parameter: -pCM0

OK Cancel Defaults Help

Fig 3

If there is a necessity for optimization, go to C/C++ tab and select the required optimization (O0, O1, O2, and O3). See Fig 4. In order to generate the hex file, go to output tab and select *create HEX file*



Preprocessor Symbols

Define:

Undefine:

Language / Code Generation

□ Execute-only Code

Optimization: Level 0 (-O0)

□ Optimize for Time

□ Split Load and Store Multiple

☑ One ELF Section per Function

□ Strict ANSI C

□ Enum Container always int

□ Plain Char is Signed

□ Read-Only Position Independent

□ Read-Write Position Independent

Warnings: All Warnings

□ Thumb Mode

□ No Auto Includes

☑ C99 Mode

Include Paths

Misc Controls

Compiler control string

```
--c99 -c --cpu Cortex-M0 -D__EVAL -li -g -O0 --apcs=interwork --split_sections
-I C:\Users\n_srinath93\Documents\keil\test\testnew\RTE
```

Fig 4

Step 5: Build target files and debug

After writing your code, click on build (See Fig 5). Once it is successfully built, you can go to debug mode.

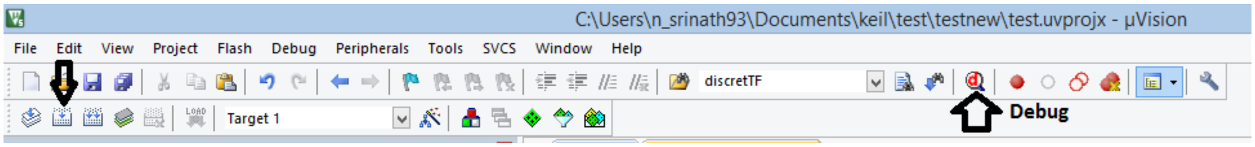
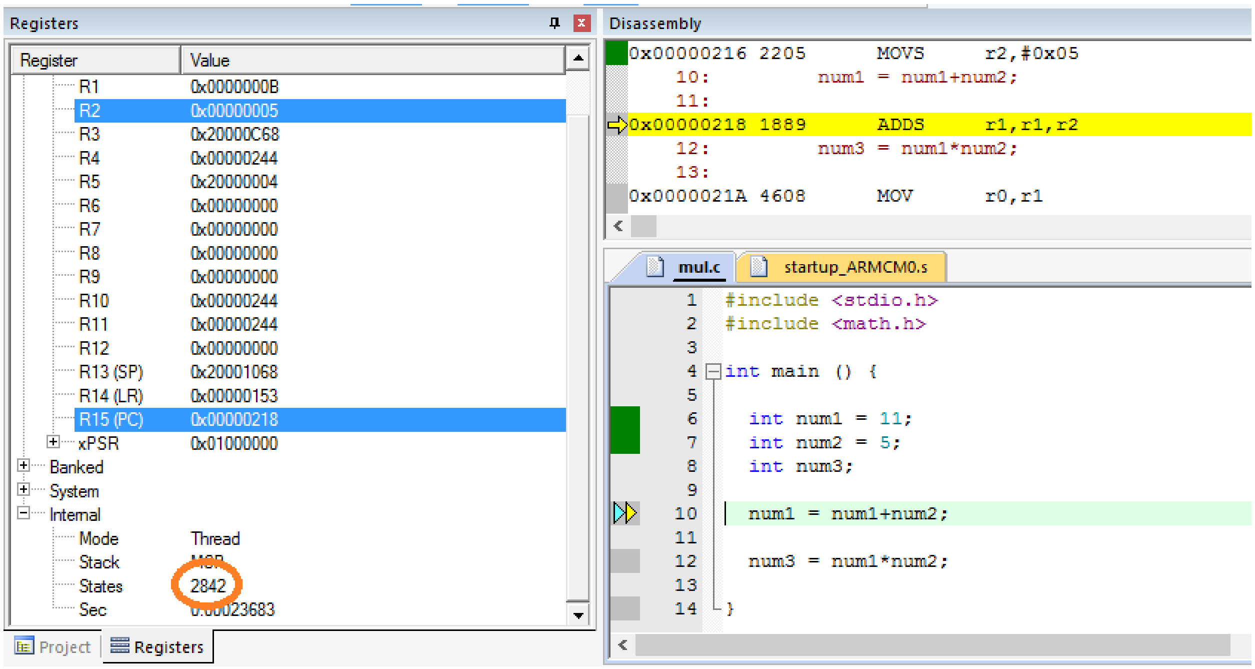


Fig 5

In the debug mode, you can observe the disassembly in the Disassembly window. See Fig 6.



Registers

Register	Value
R1	0x0000000B
R2	0x00000005
R3	0x20000C68
R4	0x00000244
R5	0x20000004
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000244
R11	0x00000244
R12	0x00000000
R13 (SP)	0x20001068
R14 (LR)	0x00000153
R15 (PC)	0x00000218
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Stack	MSP
States	2842
Sec	0.00023683

Disassembly

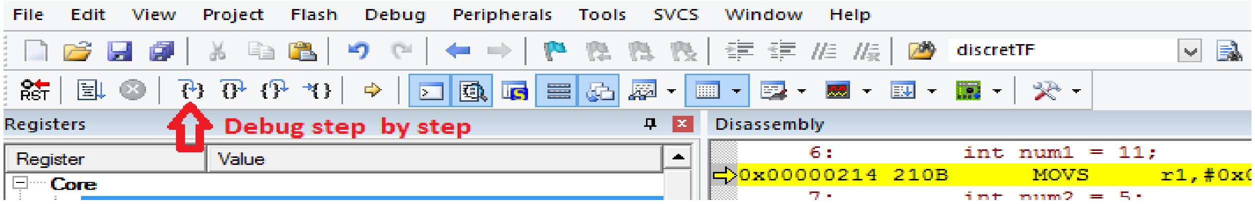
```
0x00000216 2205      MOVS     r2,#0x05
    10:          num1 = num1+num2;
    11:
0x00000218 1889      ADDS     r1,r1,r2
    12:          num3 = num1*num2;
    13:
0x0000021A 4608      MOV      r0,r1
```

mul.c startup_ARMCM0.s

```
#include <stdio.h>
#include <math.h>

int main () {

  int num1 = 11;
  int num2 = 5;
  int num3;

  num1 = num1+num2;

  num3 = num1*num2;

}
```

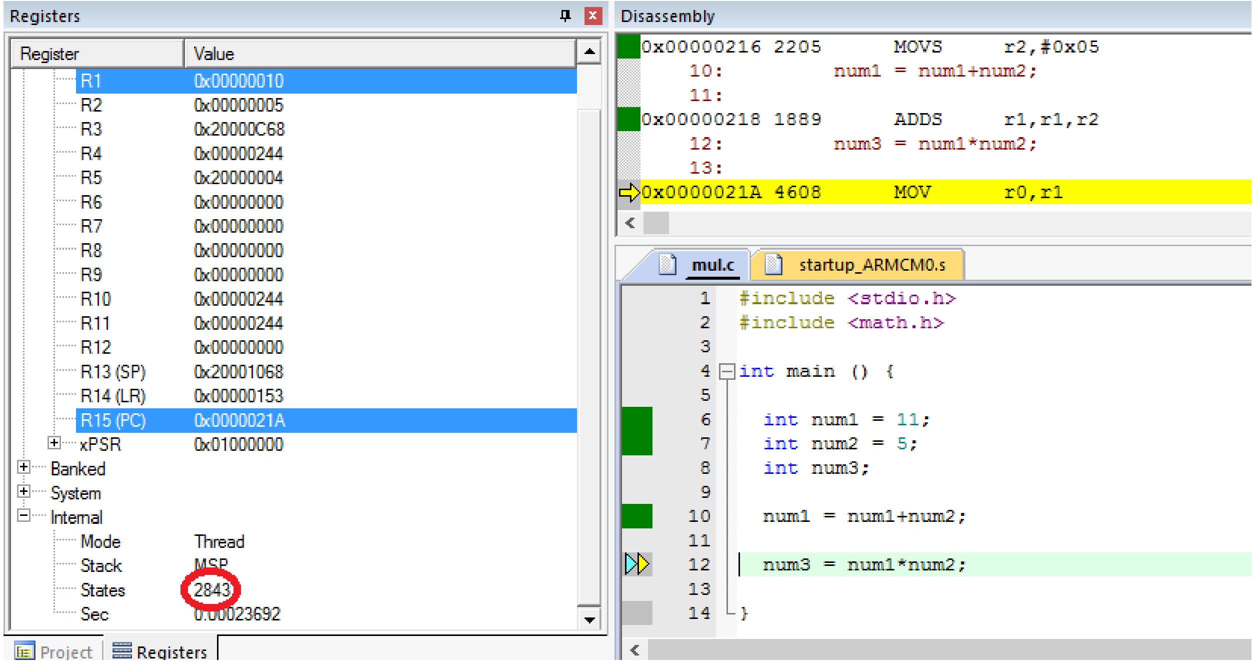
Fig 6

Let us consider a simple example of addition and multiplication. The increment to the number of states refers to the number of clock cycles in that step. Once the addition is done, you can observe an increment of one, if it’s a int type. For multiplication, the increment to the number of number of steps would be 3. You can debug step by step and see the change in the states (See Fig 8 and 9). Fig 7 shows how to debug step by step.



```
       6:          int num1 = 11;
0x00000214 210B        MOVS      r1,#0x(
```

Fig 7

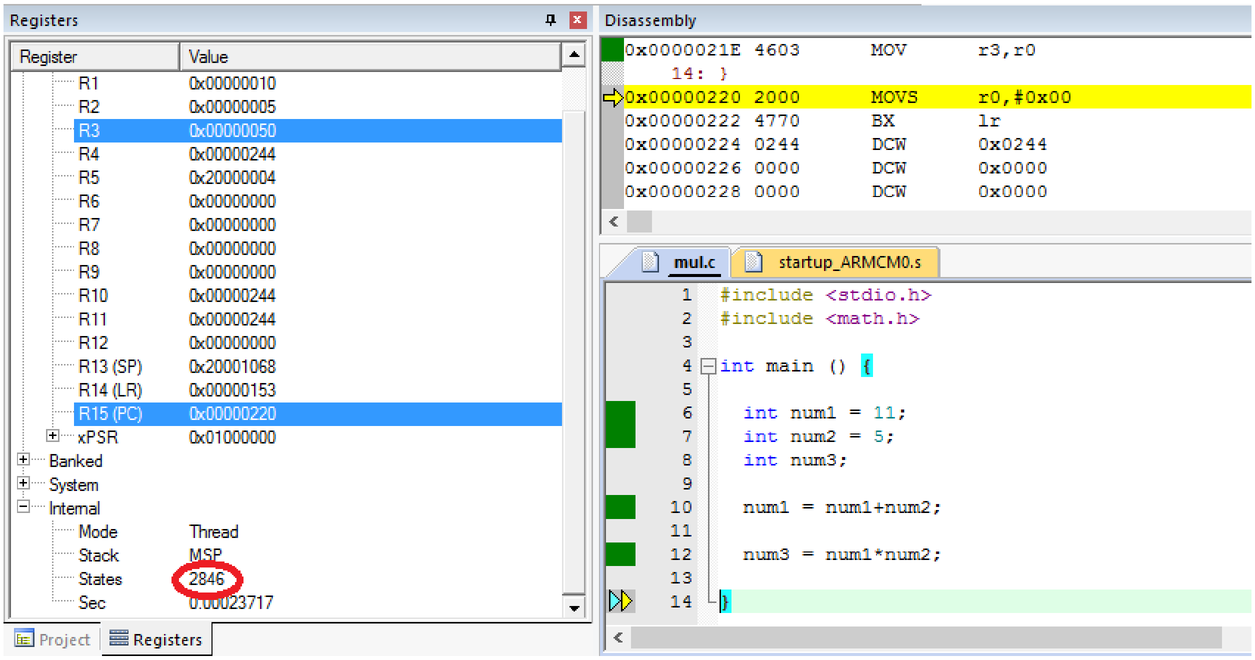


Register	Value
R1	0x00000010
R2	0x00000005
R3	0x20000C68
R4	0x00000244
R5	0x20000004
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000244
R11	0x00000244
R12	0x00000000
R13 (SP)	0x20001068
R14 (LR)	0x00000153
R15 (PC)	0x0000021A
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Stack	MSP
States	2843
Sec	0.00023692

```
0x00000216 2205      MOVS      r2,#0x05
    10:          num1 = num1+num2;
    11:
0x00000218 1889      ADDS      r1,r1,r2
    12:          num3 = num1*num2;
    13:
0x0000021A 4608      MOV       r0,r1
```

```
#include <stdio.h>
#include <math.h>

int main () {

  int num1 = 11;
  int num2 = 5;
  int num3;

  num1 = num1+num2;

  num3 = num1*num2;

}
```

Fig 8: Increment of 1 after an addition



Register	Value
R1	0x00000010
R2	0x00000005
R3	0x00000050
R4	0x00000244
R5	0x20000004
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000244
R11	0x00000244
R12	0x00000000
R13 (SP)	0x20001068
R14 (LR)	0x00000153
R15 (PC)	0x00000220
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Stack	MSP
States	2846
Sec	0.00023717

```
0x0000021E 4603      MOV       r3,r0
    14: }
0x00000220 2000      MOVS      r0,#0x00
0x00000222 4770      BX        lr
0x00000224 0244      DCW       0x0244
0x00000226 0000      DCW       0x0000
0x00000228 0000      DCW       0x0000
```

```
#include <stdio.h>
#include <math.h>

int main () {

  int num1 = 11;
  int num2 = 5;
  int num3;

  num1 = num1+num2;

  num3 = num1*num2;

}
```

Fig 9: Increment of 3 after a multiplication