

# **Tutorial 05: ARM Data Definition Directives and ARM Pseudo Instructions**

*Computer Science Department*

*CS2208b: Introduction to Computer Organization and Architecture*

*Winter 2019*

*Instructor: Mahmoud R. El-Sakka*

*Office: MC-419*

*Email: [elsakka@csd.uwo.ca](mailto:elsakka@csd.uwo.ca)*

*Phone: 519-661-2111 x86996*

# Data Definition Directives

❑ Assembly language directives include:

AREA To name a region of **code** or **data**

ENTRY The execution starting point

END The physical end of the program

*name* EQU *value* Equate a *name* to a *value*  
***Will not make any memory allocation, i.e.,  
 Similar to #define in C***

*{label}* DCD v. expr {, v. expr} ... Set up one or more 32-bit constant in memory  
*Must start at a multiple of 4 address location*

*{label}* DCW v. expr {, v. expr} ... Set up one or more 16-bit constant in memory  
*Must start at an even address location*

*{label}* DCB v. expr {, v. expr} ... Set up one or more 8-bit constant in memory  
*Can start anywhere*

*{label}* SPACE size expr Reserves a zeroed block of memory  
*Can start anywhere*

ALIGN Ensures that next instruction is  
 correctly aligned on 32-bit boundaries,  
 i.e., to start at a multiple of 4 address location

# Data Definition Directives

- ❑ Some symbols in Keil assembler have different meanings, based on their location within the instruction:

○ Equal sign “=”

- at the opcode column *means* DCB
- as a prefix of the 2<sup>nd</sup> operand of an LDR instruction *means* pseudo instruction

Example 1:

XYZ = 0x41 ; the = sign in this context means DCB, i.e.,

XYZ DCB 0x41

What will happen if the “=” sign is omitted?

Example 2:

LDR r0,=0x12345678 ; to LDR the 32-bit value 0x12345678 into r0

LDR r0,=PPP ; to LDR the 32-bit address of PPP into r0

the = sign in this context means the LDR here is a pseudo instruction

○ Ampersand sign “&”

- at the opcode column *means* DCD
- as a prefix of an operand *means* a HEX value of a *single byte* (i.e., similar to 0x)

Example 3:

AAA & 0x123456 ; the & sign in this context means DCD, i.e.,

AAA DCD 0x123456

Example 4:

MOV r0,#&8F ; the & sign in this context means a HEX value of a *single byte*

○ Percent sign “%”

- at the opcode column *means* SPACE

Example 5:

BBB % 0x40 ; the % sign in this context means SPACE, i.e.,

BBB SPACE 0x40

# Writing Numbers with Various Radix

□ The Keil assembler uses

- a prefix **0x** to indicate hexadecimal, e.g.,  
`MOV r1, #0x9C`  
or  
`DCD 0x9C`
- a prefix **2\_** to indicate binary, e.g.,  
`MOV r1, #2_10011100`  
or  
`DCD 2_10011100`
- a prefix **8\_** to indicate octal, e.g.,  
`MOV r1, #8_234`  
or  
`DCD 8_234`
- **no** prefix to indicate decimal, e.g.,  
`MOV r1, #156`  
or  
`DCD 156`

In ARM assembly,  
the "#" means  
**Literal or immediate**  
addressing mode

In ARM assembly,  
It is **illegal** to use "#" with  
`DCD`, `DCW`, or `DCB`

# Data Definition Directives

```
AREA More_data_definitions, CODE, READONLY
```

```
ENTRY
```

```
MOV r0, # 0xFC; Store a Positive HEX number in r0
```

```
MOV r1, #-0xFC; Store a negative HEX number in r1
```

```
MOV r2, # 240; Store a Positive decimal number in r2
```

```
MOV r3, # -240; Store a negative decimal number in r3
```

```
loop B loop
```

```
one = 1,1,1,1 ; the "=" here means DCB
```

```
Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
```

```
; The "0x" prefix is NOT allowed after the "&"
```

```
; DCB can start at any memory location.
```

```
two DCW 2 ; Must start at an even address location.
```

```
; One byte to be skipped to adjust the location counter.
```

```
; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
```

```
; TO MAKE THE ADDRESS MULTIPLE OF 4
```

```
four & 4,4 ; the "&" here means DCD
```

```
; DCD must start at a multiple of 4 address location
```

```
DCD 2_1010 ; Binary positive number
```

```
DCD -2_1010 ; Binary negative number
```

```
DCD 8_12345670 ; Octal positive number
```

```
DCD -8_12345670 ; Octal negative number
```

```
DCB 1 ; Any data directive can be without label
```

```
data_1 SPACE 5 ; reserves a ZEROED 5 bytes block of memory
```

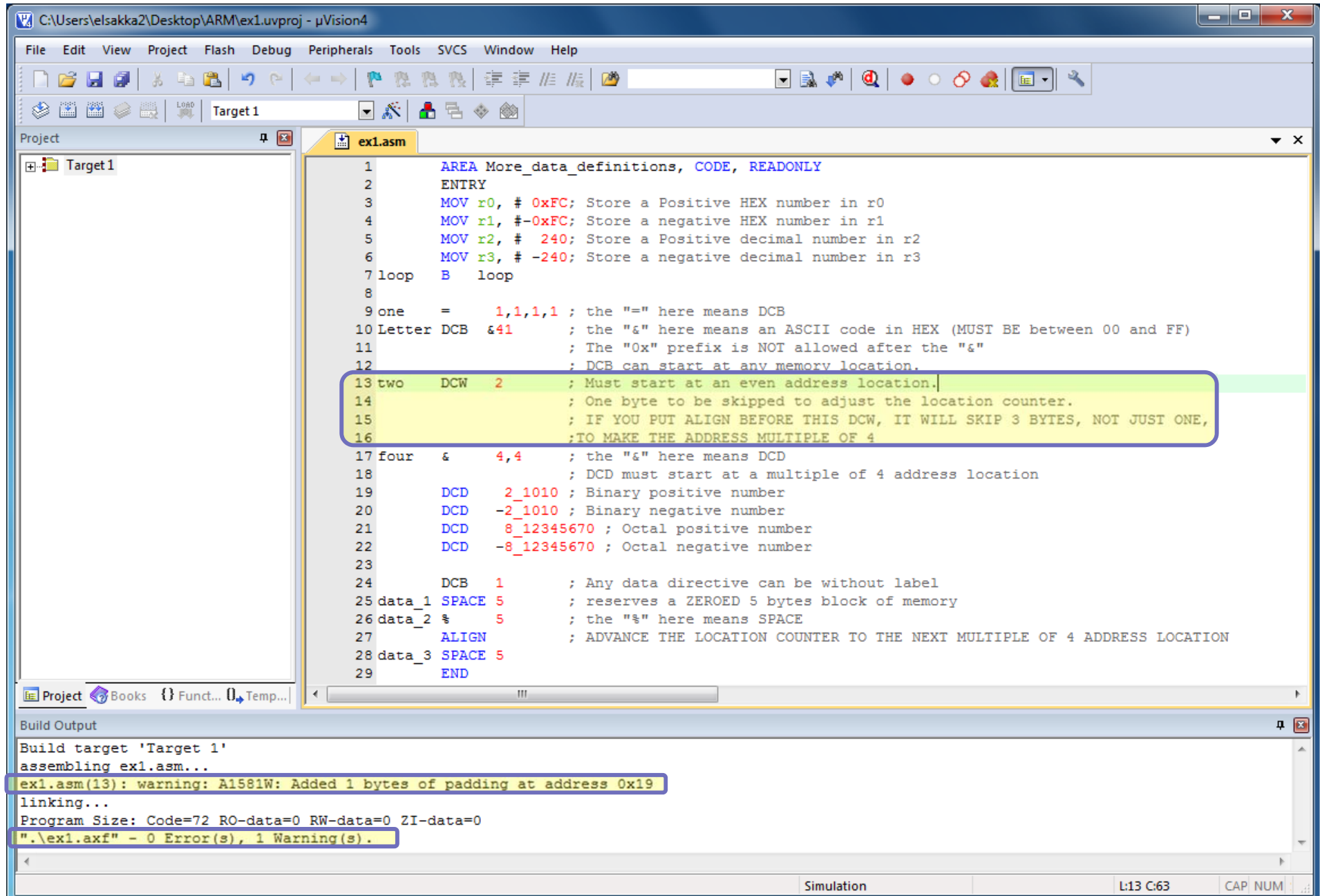
```
data_2 % 5 ; the "%" here means SPACE
```

```
ALIGN ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE OF 4 ADDRESS LOCATION
```

```
data_3 SPACE 5
```

```
END
```

# Data Definition Directives



The screenshot shows the uVision4 IDE with the following assembly code in `ex1.asm`:

```

1  AREA More_data_definitions, CODE, READONLY
2  ENTRY
3  MOV r0, # 0xFC; Store a Positive HEX number in r0
4  MOV r1, #-0xFC; Store a negative HEX number in r1
5  MOV r2, # 240; Store a Positive decimal number in r2
6  MOV r3, #-240; Store a negative decimal number in r3
7  loop B loop
8
9  one = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11 ; The "0x" prefix is NOT allowed after the "&"
12 ; DCB can start at any memory location.
13 two DCW 2 ; Must start at an even address location.
14 ; One byte to be skipped to adjust the location counter.
15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16 ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four & 4,4 ; the "&" here means DCD
18 ; DCD must start at a multiple of 4 address location
19 DCD 2_1010 ; Binary positive number
20 DCD -2_1010 ; Binary negative number
21 DCD 8_12345670 ; Octal positive number
22 DCD -8_12345670 ; Octal negative number
23
24 DCB 1 ; Any data directive can be without label
25 data_1 SPACE 5 ; reserves a ZEROED 5 bytes block of memory
26 data_2 % 5 ; the "%" here means SPACE
27 ALIGN ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE OF 4 ADDRESS LOCATION
28 data_3 SPACE 5
29 END

```

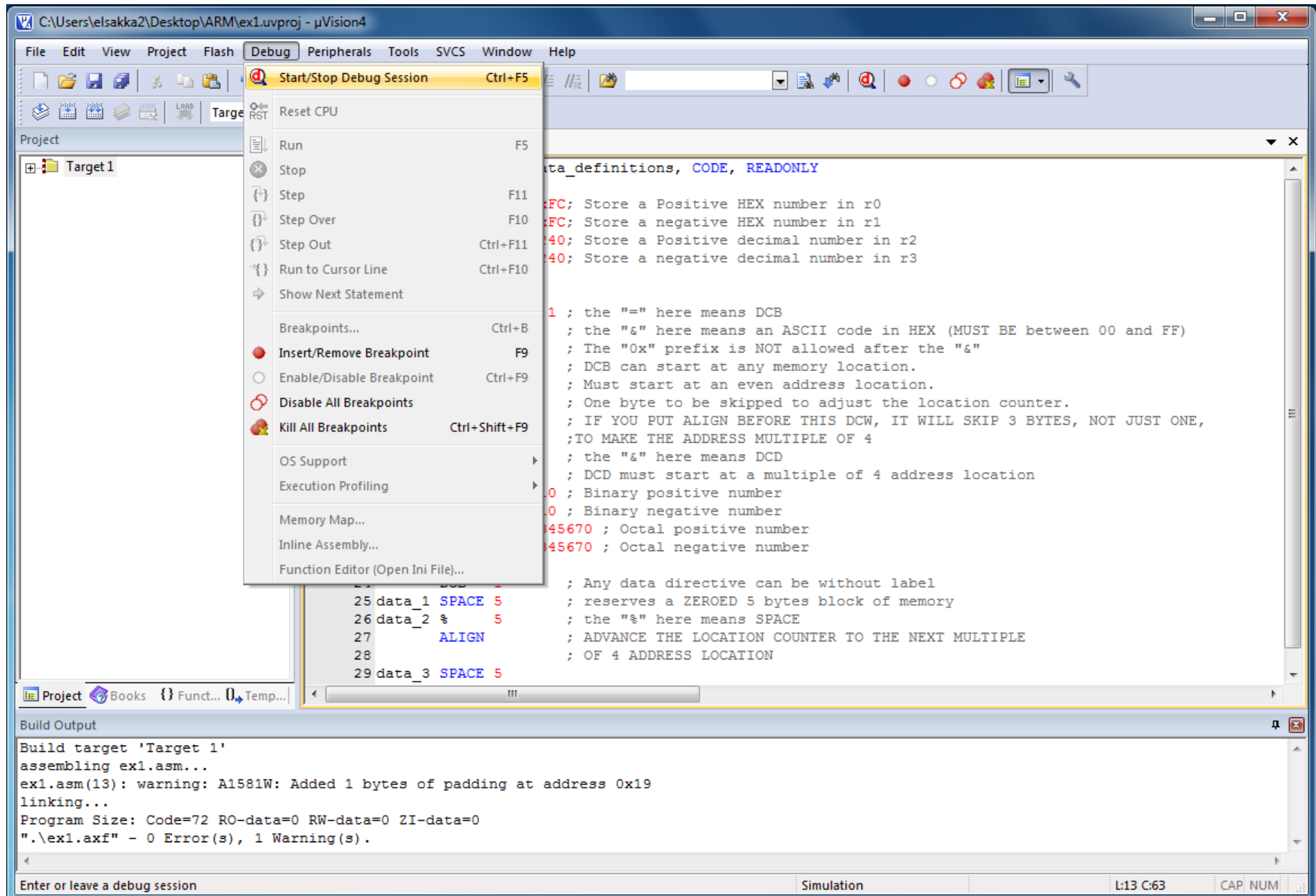
The build output at the bottom shows the following messages:

```

Build target 'Target 1'
assembling ex1.asm...
ex1.asm(13): warning: A1581W: Added 1 bytes of padding at address 0x19
linking...
Program Size: Code=72 RO-data=0 RW-data=0 ZI-data=0
".\ex1.axf" - 0 Error(s), 1 Warning(s).

```

# Data Definition Directives





# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. Below the menu is a toolbar with various icons for file operations, execution, and debugging.

The **Registers** window on the left shows the current state of ARM registers. The **Disassembly** window in the center displays the following assembly code:

```

3:      MOV r0, # 0xFC; Store a Positive HEX number in r0
0x00000000 E3A000FC MOV     R0,#0x000000FC
4:      MOV r1, #-0xFC; Store a negative HEX number in r1
0x00000004 E3E010FB MVN     R1,#0x000000FB
5:      MOV r2, # 240; Store a Positive decimal number in r2
0x00000008 E3A020F0 MOV     R2,#0x000000F0

```

The **ex1.asm\*** window shows the source assembly code with comments:

```

1      AREA More_data_definitions, CODE, READONLY
2      ENTRY
3      MOV r0, # 0xFC; Store a Positive HEX number in r0
4      MOV r1, #-0xFC; Store a negative HEX number in r1
5      MOV r2, # 240; Store a Positive decimal number in r2
6      MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one   = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11      ; The "0x" prefix is NOT allowed after the "&"
12      ; DCB can start at any memory location.
13 two   DCW 2 ; Must start at an even address location.
14      ; One byte to be skipped to adjust the location counter.
15      ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16      ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four   & 4,4 ; the "&" here means DCD
18      ; DCD must start at a multiple of 4 address location
19      DCD 2_1010 ; Binary positive number

```

The **Command** window at the bottom left shows the following text:

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 72 Bytes (0%)

```

The **Call Stack + Locals** window at the bottom right shows a single entry:

Name	Location/Value	Type
__asm_0x0	0x00000000	void f()

The status bar at the bottom indicates the simulation is running, with a time of 0.00000000 sec and a location of L3 C:1.



# Data Definition Directives

The screenshot shows the uVision4 IDE interface. The main window displays assembly code for a file named `ex1.asm`. The code includes several `MOV` instructions and data definition directives. The left sidebar shows the 'Memory Windows' menu, which is expanded to show 'Memory 1', 'Memory 2', 'Memory 3', and 'Memory 4'. The bottom status bar indicates the simulation is running.

**Assembly Code:**

```

3:      MOV r0, # 0xFC; Store a Positive HEX number in r0
0x00000000 E3A000FC MOV      R0,#0x000000FC
4:      MOV r1, #-0xFC; Store a negative HEX number in r1
0x00000004 E3E010FB MVN      R1,#0x000000FB
5:      MOV r2, # 240; Store a Positive decimal number in r2
0x00000008 E3A020F0 MOV      R2,#0x000000F0

```

**Data Definitions:**

```

1      AREA More_data_definitions, CODE, READONLY
2      ENTRY
3      MOV r0, # 0xFC; Store a Positive HEX number in r0
4      MOV r1, #-0xFC; Store a negative HEX number in r1
5      MOV r2, # 240; Store a Positive decimal number in r2
6      MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one = 1,1,1,1 ; the "=" here means DCB
        &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
        ; The "0x" prefix is NOT allowed after the "&"
        ; DCB can start at any memory location.
        ; Must start at an even address location.
        ; One byte to be skipped to adjust the location counter.
        ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
        ; TO MAKE THE ADDRESS MULTIPLE OF 4
16
17 four & 4,4 ; the "&" here means DCD
18      ; DCD must start at a multiple of 4 address location
19      DCD 2,1010 ; Binary positive number

```

**Command Window:**

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 72 Bytes (0%)

```

**Call Stack + Locals:**

Name	Location/Value	Type
__asm_0x0	0x00000000	void f()

**Status Bar:** Simulation t1: 0.00000000 sec L3 C:1 CAP NUM

# Data Definition Directives

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Lists registers R0 through R15, CPSR, and SPSR. R0 is highlighted with a value of 0x00000000.
- Disassembly Window:** Shows assembly instructions:
 

```

      3:      MOV r0, # 0xFC; Store a Positive HEX number in r0
      ->0x00000000 E3A000FC MOV      R0,#0x000000FC
      4:      MOV r1, #-0xFC; Store a negative HEX number in r1
      0x00000004 E3E010FB MVN      R1,#0x000000FB
      5:      MOV r2, # 240; Store a Positive decimal number in r2
      0x00000008 E3A020F0 MOV      R2,#0x000000F0
      
```
- Source Window (ex1.asm):**

```

      1      AREA More_data_definitions, CODE, READONLY
      2      ENTRY
      3      MOV r0, # 0xFC; Store a Positive HEX number in r0
      4      MOV r1, #-0xFC; Store a negative HEX number in r1
      5      MOV r2, # 240; Store a Positive decimal number in r2
      6      MOV r3, #-240; Store a negative decimal number in r3
      7 loop   B   loop
      8
      9 one    =    1,1,1,1 ; the "=" here means DCB
      10 Letter DCB  &41    ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
      11                                ; The "0x" prefix is NOT allowed after the "&"
      12                                ; DCB can start at any memory location.
      13 two    DCW   2      ; Must start at an even address location.
      14                                ; One byte to be skipped to adjust the location counter.
      15                                ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
      16                                ; TO MAKE THE ADDRESS MULTIPLE OF 4
      17 four    &    4,4    ; the "&" here means DCD
      18                                ; DCD must start at a multiple of 4 address location
      19      DCD    2_1010 ; Binary positive number
      
```
- Command Window:**

```

      *** Restricted Version with 32768 Byte Code Size Limit
      *** Currently used: 72 Bytes (0%)
      >
      ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
      
```
- Memory Window:**

```

      Address: 0
      0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
      0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 00 04 00 00 04
      0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
      0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      
```
- Simulation Status:** Simulation, t1: 0.00000000 sec, L3 C:57, CAP NUM

A blue callout bubble with the text "Press Step, or F11" points to the 'Step' button in the toolbar.

# Data Definition Directives

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Lists ARM registers R0 through R15 (PC), CPSR, and SPSR. R0 is highlighted with value 0x000000FC, and R15 (PC) is highlighted with value 0x00000004.
- Disassembly Window:** Shows assembly instructions:
 

```

3:      MOV r0, # 0xFC; Store a Positive HEX number in r0
0x00000000 E3A000FC MOV      R0,#0x000000FC
4:      MOV r1, #-0xFC; Store a negative HEX number in r1
0x00000004 E3E010FB MVN      R1,#0x000000FB
5:      MOV r2, # 240; Store a Positive decimal number in r2
0x00000008 E3A020F0 MOV      R2,#0x000000F0
      
```
- Source Window (ex1.asm):** Shows assembly code with data definitions:
 

```

1      AREA More_data_definitions, CODE, READONLY
2      ENTRY
3      MOV r0, # 0xFC; Store a Positive HEX number in r0
4      MOV r1, #-0xFC; Store a negative HEX number in r1
5      MOV r2, # 240; Store a Positive decimal number in r2
6      MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one   = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11                                     ; The "0x" prefix is NOT allowed after the "&"
12                                     ; DCB can start at any memory location.
13 two   DCW 2 ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four  & 4,4 ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19      DCD 2_1010 ; Binary positive number
      
```
- Command Window:** Displays:
 

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 72 Bytes (0%)
      
```
- Memory Window:** Shows memory dump starting at address 0:
 

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 02 00 00 04 00 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      
```
- Toolbar:** The 'Step' button (represented by a right-pointing arrow) is circled in blue.
- Callout:** A blue cloud-shaped bubble contains the text "Press Step, or F11".

# Data Definition Directives

The screenshot shows the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations, debugging, and simulation. The 'Registers' window on the left shows the current state of ARM registers, with R15 (PC) highlighted at 0x00000008. The 'Disassembly' window shows the following assembly code:

```

3:      MOV r0, # 0xFC; Store a Positive HEX number in r0
0x00000000 E3A000FC MOV      R0,#0x000000FC
4:      MOV r1, #-0xFC; Store a negative HEX number in r1
0x00000004 E3E010FB MVN      R1,#0x000000FB
5:      MOV r2, # 240; Store a Positive decimal number in r2
0x00000008 E3A020F0 MOV      R2,#0x000000F0

```

The 'ex1.asm' source file is open, showing the following assembly code:

```

1      AREA More_data_definitions, CODE, READONLY
2      ENTRY
3      MOV r0, # 0xFC; Store a Positive HEX number in r0
4      MOV r1, #-0xFC; Store a negative HEX number in r1
5      MOV r2, # 240; Store a Positive decimal number in r2
6      MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one   = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11      ; The "0x" prefix is NOT allowed after the "&"
12      ; DCB can start at any memory location.
13 two   DCW 2 ; Must start at an even address location.
14      ; One byte to be skipped to adjust the location counter.
15      ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16      ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four   & 4,4 ; the "&" here means DCD
18      ; DCD must start at a multiple of 4 address location
19      DCD 2_1010 ; Binary positive number

```

The 'Memory' window shows the memory dump starting at address 0x00000000:

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

A blue callout bubble with the text "Press Step, or F11" points to the 'Step' button in the toolbar.



# Data Definition Directives

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Lists registers R0 through R15 (PC), CPSR, and SPSR with their current values. R15 (PC) is highlighted with a value of 0x0000000C.
- Disassembly Window:** Shows the disassembled code. The instruction `MOVN R3, #0x000000EF` is highlighted in yellow, corresponding to the assembly instruction `E3E030EF` in the source file.
- Source File (ex1.asm):** Contains ARM assembly code with data definition directives. The instruction `MOV r3, #-240; Store a negative decimal number in r3` is highlighted in green, corresponding to the assembly instruction `E3E030EF` in the disassembly window.
- Memory Window:** Displays a memory dump starting at address 0. The first four bytes (0x00000000 to 0x00000003) are highlighted in yellow, corresponding to the assembly instruction `E3E030EF` in the disassembly window.
- Command Window:** Shows the status of the simulation, including the code size limit (32768 bytes) and the current usage (72 bytes).
- Simulation Status:** Displays the simulation time (0.00000000 sec) and the current instruction (L6 C:61).

A blue callout bubble with the text "Press Step, or F11" points to the "Step" button in the toolbar, indicating the next action to be taken.

# Data Definition Directives

The screenshot shows the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons, including a 'Step' button (a square with a right-pointing arrow) which is circled in blue. A blue callout bubble with the text 'Press Step, or F11' points to this button.

The 'Registers' window on the left shows the current state of ARM registers. R15 (PC) is highlighted with a value of 0x00000010.

The 'Disassembly' window shows the following assembly code:

```

0x00000008 E3A020F0 MOV R2,#0x000000F0
6:          MOV r3, #-240; Store a negative decimal number in r3
0x0000000C E3E030EF MVN R3,#0x000000EF
7: loop    B    loop
0x00000010 EAF0FFFE B    0x00000010
0x00000014 01010101 (???)EQ
  
```

The 'ex1.asm\*' window shows the source code with data definition directives:

```

1 AREA More_data_definitions, CODE, READONLY
2 ENTRY
3 MOV r0, # 0xFC; Store a Positive HEX number in r0
4 MOV r1, #-0xFC; Store a negative HEX number in r1
5 MOV r2, # 240; Store a Positive decimal number in r2
6 MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11 ; The "0x" prefix is NOT allowed after the "&"
12 ; DCB can start at any memory location.
13 two DCW 2 ; Must start at an even address location.
14 ; One byte to be skipped to adjust the location counter.
15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16 ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four & 4,4 ; the "&" here means DCD
18 ; DCD must start at a multiple of 4 address location
19 DCD 2_1010 ; Binary positive number
  
```

The 'Memory' window shows a memory dump starting at address 0:

```

Address: 0
0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  
```

The 'Command' window shows the status: '\*\*\* Restricted Version with 32768 Byte Code Size Limit' and '\*\*\* Currently used: 72 Bytes (0%)'. The bottom status bar indicates 'Simulation' mode, 't1: 0.00000000 sec', 'L7 C:16', and 'CAP NUM'.

# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The left pane shows the 'Registers' window with a list of registers (R0-R15, CPSR, SPSR) and their current values. The main window shows the 'Disassembly' view of the assembly code, with the following instructions visible:

```

0x00000008 E3A020F0 MOV R2,#0x000000F0
6:          MOV r3, #-240; Store a negative decimal number in r3
0x0000000C E3E030EF MVN R3,#0x000000EF
7: loop B loop
->0x00000010 EAEFFFE B 0x00000010
0x00000014 01010101 (???)EQ
  
```

The 'ex1.asm' file is open, showing the following assembly code:

```

1 AREA More_data_definitions, CODE, READONLY
2 ENTRY
3 MOV r0, # 0xFC; Store a Positive HEX number in r0
4 MOV r1, #-0xFC; Store a negative HEX number in r1
5 MOV r2, # 240; Store a Positive decimal number in r2
6 MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11 ; The "0x" prefix is NOT allowed after the "&"
12 ; DCB can start at any memory location.
13 two DCW 2 ; Must start at an even address location.
14 ; One byte to be skipped to adjust the location counter.
15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16 ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four & 4,4 ; the "&" here means DCD
18 ; DCD must start at a multiple of 4 address location
19 DCD 2_1010 ; Binary positive number
  
```

The bottom pane shows the 'Memory' window with a memory dump starting at address 0x00000000. The dump shows the following data:

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  
```

The bottom status bar shows 'Simulation' mode, 't1: 0.00000000 sec', 'L7 C:16', and 'CAP NUM'.



# Data Definition Directives

The "=" here means DCB

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of ARM registers. R0 is 0x000000FC, R1 is 0xFFFFF04, R2 is 0x000000F0, R3 is 0xFFFFF10, R4 is 0x00000000, and R5 is 0x00000000. The PC register is 0x00000010.
- Disassembly Window:** Shows the disassembled code. The instruction at address 0x00000014 is highlighted: `01010101 (???) EQ`.
- Source Code Window (ex1.asm):**

```

1      AREA More_data_definitions, CODE, READONLY
2      ENTRY
3      MOV r0, # 0xFC; Store a Positive HEX number in r0
4      MOV r1, #-0xFC; Store a negative HEX number in r1
5      MOV r2, # 240; Store a Positive decimal number in r2
6      MOV r3, #-240; Store a negative decimal number in r3
7      loop B loop
8
9      one = 1,1,1,1; the "=" here means DCB
10     Letter DCB &41; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11
12
13     two DCW 2; DCB can start at any memory location.
14
15
16
17     four & 4,4; Must start at an even address location.
18
19     DCD 2,1010; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
; TO MAKE THE ADDRESS MULTIPLE OF 4
; the "&" here means DCD
; DCD must start at a multiple of 4 address location
; Binary positive number

```
- Memory Window:** Shows the memory dump starting at address 0. The first few bytes are: `E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF`.
- Command Window:** Shows the status: `*** Restricted Version with 32768 Byte Code Size Limit` and `*** Currently used: 72 Bytes (0%)`.
- Simulation Window:** Shows the simulation status: `Simulation`, `tl: 0.00000000 sec`, `L:9 C:46`, and `CAP: NUM`.

# Data Definition Directives

The “&” an HEX ASCII code

**Registers**

Register	Value
R0	0x000000FC
R1	0xFFFFFFFF
R2	0x000000F0
R3	0xFFFFFFFF
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000010
CPSR	0x000000D3
SPSR	0x00000000

**Disassembly**

```

7: loop B loop
0x00000010 EAfffffe B 0x00000010
0x00000014 01010101 (???)EQ
0x00000018 41000002 (???)MI
0x0000001C 00000004 ANDEQ R0,R0,R4
0x00000020 00000004 ANDEQ R0,R0,R4
  
```

**ex1.asm\***

```

1 AREA More_data_definitions, CODE, READONLY
2 ENTRY
3 MOV r0, # 0xFC; Store a Positive HEX number in r0
4 MOV r1, #-0xFC; Store a negative HEX number in r1
5 MOV r2, # 240; Store a Positive decimal number in r2
6 MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11 ; The "0x" prefix is NOT allowed after the "&"
12 ; DCB can start at any memory location.
13 two DCW 2 ; Must start at an even address location.
14 ; One byte to be skipped to adjust the location counter.
15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16 ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four & 4,4 ; the "&" here means DCD
18 ; DCD must start at a multiple of 4 address location
19 DCD 2_1010 ; Binary positive number
  
```

**Command**

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 72 Bytes (0%)
  
```

**Memory 1**

Address: 0

0x00000000:	E3 A0 00 FC	E3 E0 10 FB	E3 A0 20 F0	E3 E0 30 EF	EA FF
0x00000012:	FF FE 01 01	01 01 41	00 00 02 00	00 00 04 00	00 00 04
0x00000024:	00 00 00 0A	FF FF FF F6	00 29 CB B8	FF D6 34 48	01 00
0x00000036:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0x00000048:	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

Simulation t1: 0.00000000 sec L:9 C:46 CAP NUM

# Data Definition Directives

The screenshot shows the uVision4 IDE with the following components:

- Registers Panel:** Lists registers R0 through R15, CPSR, and SPSR with their current values. R0 is 0x000000FC, R1 is 0xFFFFF04, R2 is 0x000000F0, R3 is 0xFFFFF10, R4 is 0x00000000, R5 is 0x00000000, R6 is 0x00000000, R7 is 0x00000000, R8 is 0x00000000, R9 is 0x00000000, R10 is 0x00000000, R11 is 0x00000000, R12 is 0x00000000, R13 (SP) is 0x00000000, R14 (LR) is 0x00000000, R15 (PC) is 0x00000010, CPSR is 0x000000D3, and SPSR is 0x00000000.
- Disassembly Panel:** Shows assembly instructions. The instruction at address 0x00000014 is highlighted: `01010101 (???)EQ`.
- Assembly Editor (ex1.asm):**

```

1      AREA More_data_definitions, CODE, READONLY
2      ENTRY
3      MOV r0, # 0xFC; Store a Positive HEX number in r0
4      MOV r1, #-0xFC; Store a negative HEX number in r1
5      MOV r2, # 240; Store a Positive decimal number in r2
6      MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one  = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11                                     ; The "0x" prefix is NOT allowed after the "&"
12                                     ; DCB can start at any memory location.
13 two  DCW 2 ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four & 4,4 ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19 DCD 2_1010 ; Binary positive number

```
- Memory Panel:** Shows a memory dump starting at address 0. The first few lines are:
 

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```
- Command Panel:** Shows the status: "Restricted Version with 32768 Byte Code Size Limit" and "Currently used: 72 Bytes (0%)".
- Simulation Panel:** Shows the simulation status: "Simulation", "t1: 0.00000000 sec", "L:9 C:46", and "CAP NUM".

A red callout bubble with the text "Must start at an even address location" points to the `one` directive in the assembly code.

# Data Definition Directives

The screenshot shows the uVision4 IDE with the following components:

- Registers Panel:** Lists ARM registers R0-R15, CPSR, and SPSR with their current values. R0 is 0x000000FC, R1 is 0xFFFFF04, R2 is 0x000000F0, R3 is 0xFFFFF10, R4 is 0x00000000, R5 is 0x00000000, R6 is 0x00000000, R7 is 0x00000000, R8 is 0x00000000, R9 is 0x00000000, R10 is 0x00000000, R11 is 0x00000000, R12 is 0x00000000, R13 (SP) is 0x00000000, R14 (LR) is 0x00000000, R15 (PC) is 0x00000010, CPSR is 0x000000D3, and SPSR is 0x00000000.
- Disassembly Panel:** Shows assembly instructions:
 

```

7: loop B loop
0x00000010 EAfffffe B 0x00000010
0x00000014 01010101 (???)EQ
0x00000018 41000002 (???)MI
0x0000001C 00000004 ANDEQ R0,R0,R4
0x00000020 00000004 ANDEQ R0,R0,R4
      
```
- Source Code Panel (ex1.asm):**

```

1 AREA More_data_definitions, CODE, READONLY
2 ENTRY
3 MOV r0, # 0xFC; Store a Positive HEX number in r0
4 MOV r1, #-0xFC; Store a negative HEX number in r1
5 MOV r2, # 240; Store a Positive decimal number in r2
6 MOV r3, #-240; Store a negative decimal number in r3
7 loop B loop
8
9 one = 1,1,1,1 ; the "=" here means DCB
10 Letter DCB &41 ; the "&" here means an ASCII code in HEX (MUST BE between 00 and FF)
11 ; The "0x" prefix is NOT allowed after the "&"
12 ; DCB can start at any memory location.
13 two DCW 2 ; Must start at an even address location.
14 ; One byte to be skipped to adjust the location counter.
15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16 ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four & 4,4 ; the "&" here means DCD
18 ; DCD must start at a multiple of 4 address location
19 DCD 2_1010 ; Binary positive number
      
```
- Memory Panel:** Shows a memory dump starting at address 0:
 

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      
```
- Call Stack + Locals Panel:** Shows the current stack frame.
- Simulation Panel:** Shows simulation time (0.00000000 sec) and location (L9 C:46).

A callout bubble points to the "&" symbol in the DCD directive, stating: "The "&" here means DCD".



# Data Definition Directives

The “&” here means DCD

**Registers**

Register	Value
R0	0x000000FC
R1	0xFFFFFFFF04
R2	0x000000F0
R3	0xFFFFFFFF10
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
PC	0x00000010

**Disassembly**

```

0x0000001C  00000004  ANDEQ    R0,R0,R4
0x00000020  00000004  ANDEQ    R0,R0,R4
0x00000024  0000000A  ANDEQ    R0,R0,R10
0x00000028  FFFFFFFF6  (???)
0x0000002C  0029CBB8  (???) EQ
0x00000030  FFD63448  (???)
  
```

**ex1.asm\***

```

12                                     ; DCB can start at any memory location.
13 two      DCW      2                ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four      &          4,4           ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19          DCD      2_1010           ; Binary positive number
20          DCD     -2_1010           ; Binary negative number
21          DCD      8_12345670       ; Octal positive number
22          DCD     -8_12345670       ; Octal negative number
23
24          DCB      1                ; Any data directive can be without label
25 data_1    SPACE  5                ; reserves a ZEROED 5 bytes block of memory
26 data_2    %          5            ; the "%" here means SPACE
27          ALIGN                                ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28                                     ; OF 4 ADDRESS LOCATION
29 data_3    SPACE  5
30          END
  
```

**Memory 1**

Address	Value
0x00000000	E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012	FF FE 01 01 01 01 41 00 00 02 00 00 00 04 00 00 04
0x00000024	00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

**Command**

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 72 Bytes (0%)
  
```

**Simulation** t1: 0.00000000 sec L:9 C:46 CAP: NUM

# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The left pane shows the 'Registers' window with a list of registers (R0-R15, CPSR, SPSR) and their current values. The main window shows the 'Disassembly' view of the assembly code, with the following instructions highlighted:

```

0x0000001C 00000004 ANDEQ R0,R0,R4
0x00000020 00000004 ANDEQ R0,R0,R4
0x00000024 0000000A ANDEQ R0,R0,R10
0x00000028 FFFFFFFF6 (???)
0x0000002C 0029CBB8 (???) EQ
0x00000030 FFD63448 (???)
  
```

The 'ex1.asm' file is open, showing the following assembly code:

```

12                                     ; DCB can start at any memory location.
13 two      DCW      2                ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four      &      4,4                ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19      DCD      2_1010                ; Binary positive number
20      DCD      -2_1010               ; Binary negative number
21      DCD      8_12345670           ; Octal positive number
22      DCD      -8_12345670          ; Octal negative number
23
24      DCB      1                    ; Any data directive can be without label
25 data_1     SPACE 5                 ; reserves a ZEROED 5 bytes block of memory
26 data_2     %      5                ; the "%" here means SPACE
27      ALIGN                                ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28                                     ; OF 4 ADDRESS LOCATION
29 data_3     SPACE 5
30      END
  
```

The 'Memory' window shows the memory dump starting at address 0:

```

Address: 0
0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  
```

The bottom status bar shows 'Simulation' mode, 't1: 0.00000000 sec', 'L9 C:46', and 'CAP NUM'.

# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The left pane shows the 'Registers' window with a list of registers (R0-R15, CPSR, SPSR) and their current values. The main window shows the 'Disassembly' view of the assembly code, with the following instructions:

```

0x0000001C 00000004 ANDEQ    R0,R0,R4
0x00000020 00000004 ANDEQ    R0,R0,R4
0x00000024 0000000A ANDEQ    R0,R0,R10
0x00000028 FFFFFFFF6 (???)
0x0000002C 0029CBB8 (???) EQ
0x00000030 FFD63448 (???)
  
```

The 'ex1.asm' file is open, showing the following assembly code:

```

12                                     ; DCB can start at any memory location.
13 two      DCW      2                ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four      &      4,4                ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19          DCD      2 1010            ; Binary positive number
20          DCD     -2 1010            ; Binary negative number
21          DCD      8 12345670        ; Octal positive number
22          DCD     -8 12345670        ; Octal negative number
23
24          DCB      1                ; Any data directive can be without label
25 data_1    SPACE  5                ; reserves a ZEROED 5 bytes block of memory
26 data_2    %      5                ; the "%" here means SPACE
27          ALIGN                                ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28                                     ; OF 4 ADDRESS LOCATION
29 data_3    SPACE  5
30          END
  
```

The 'Memory' window shows a memory dump starting at address 0, with the following data:

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  
```

The bottom status bar shows 'Simulation' mode, 't1: 0.00000000 sec', 'L9 C:46', and 'CAP NUM'.



# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. Below the menu is a toolbar with various icons for file operations, debugging, and viewing.

The **Registers** window on the left shows the current state of ARM registers. The **Disassembly** window in the center displays the assembly code being executed, with the instruction `0x00000030 FFD63448 (???)` highlighted in yellow. The **ex1.asm\*** window shows the source code with several data definition directives:

```

12                                     ; DCB can start at any memory location.
13 two      DCW      2                ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four      &      4,4                ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19          DCD      2_1010            ; Binary positive number
20          DCD     -2_1010            ; Binary negative number
21          DCD      8_12345670        ; Octal positive number
22          DCD     -8_12345670        ; Octal negative number
23
24          DCB      1                ; Any data directive can be without label
25 data_1  SPACE 5                    ; reserves a ZEROED 5 bytes block of memory
26 data_2  %      5                    ; the "%" here means SPACE
27          ALIGN                                ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28                                     ; OF 4 ADDRESS LOCATION
29 data_3  SPACE 5
30          END

```

The **Command** window at the bottom left shows the status: `*** Restricted Version with 32768 Byte Code Size Limit` and `*** Currently used: 72 Bytes (0%)`. The **Memory** window at the bottom right displays a memory dump starting at address 0, with the value `00 29 CB B8` highlighted in yellow, corresponding to the instruction in the disassembly window.

# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The left pane shows the 'Registers' window with a list of registers (R0-R15, CPSR, SPSR) and their current values. The main window shows the 'Disassembly' view of the assembly code, with the following instructions:

```

0x0000001C 00000004 ANDEQ    R0,R0,R4
0x00000020 00000004 ANDEQ    R0,R0,R4
0x00000024 0000000A ANDEQ    R0,R0,R10
0x00000028 FFFFFFFF6 (???)
0x0000002C 0029CBB8 (???) EQ
0x00000030 FFD63448 (???)
  
```

The 'ex1.asm' file is open, showing the following assembly code:

```

12                                     ; DCB can start at any memory location.
13 two      DCW      2                ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four      &      4,4                ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19          DCD      2_1010            ; Binary positive number
20          DCD      -2_1010           ; Binary negative number
21          DCD      8 12345670        ; Octal positive number
22          DCD      -8 12345670       ; Octal negative number
23
24          DCB      1                ; Any data directive can be without label
25 data_1    SPACE  5                ; reserves a ZEROED 5 bytes block of memory
26 data_2    %      5                ; the "%" here means SPACE
27          ALIGN                                ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28                                     ; OF 4 ADDRESS LOCATION
29 data_3    SPACE  5
30          END
  
```

The 'Memory' window shows a memory dump starting at address 0, with the following data:

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  
```

The bottom status bar shows 'Simulation' mode, 't1: 0.00000000 sec', 'L9 C:46', and 'CAP NUM'.

# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The main workspace is divided into several panes:

- Registers:** A list of ARM registers (R0-R15, CPSR, SPSR) with their current values. R0 is 0x000000FC, R1 is 0xFFFFF04, R2 is 0x000000F0, R3 is 0xFFFFF10, R4 is 0x00000000, R5 is 0x00000000, R6 is 0x00000000, R7 is 0x00000000, R8 is 0x00000000, R9 is 0x00000000, R10 is 0x00000000, R11 is 0x00000000, R12 is 0x00000000, R13 (SP) is 0x00000000, R14 (LR) is 0x00000000, R15 (PC) is 0x00000010, CPSR is 0x000000D3, and SPSR is 0x00000000.
- Disassembly:** Shows the assembly code being executed. The current instruction is `0x00000040 00000000 ANDEQ R0,R0,R0`, which is highlighted in yellow.
- ex1.asm\*:** The source assembly file. It contains several data definition directives:
  - `12 ; DCB can start at any memory location.`
  - `13 two DCW 2 ; Must start at an even address location.`
  - `14 ; One byte to be skipped to adjust the location counter.`
  - `15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,`
  - `16 ; TO MAKE THE ADDRESS MULTIPLE OF 4`
  - `17 four & 4,4 ; the "&" here means DCD`
  - `18 ; DCD must start at a multiple of 4 address location`
  - `19 DCD 2_1010 ; Binary positive number`
  - `20 DCD -2_1010 ; Binary negative number`
  - `21 DCD 8_12345670 ; Octal positive number`
  - `22 DCD -8_12345670 ; Octal negative number`
  - `23`
  - `24 DCB 1 ; Any data directive can be without label`
  - `25 data_1 SPACE 5 ; reserves a ZEROED 5 bytes block of memory`
  - `26 data_2 % 5 ; the "%" here means SPACE`
  - `27 ALIGN ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE`
  - `28 ; OF 4 ADDRESS LOCATION`
  - `29 data_3 SPACE 5`
  - `30 END`
- Command:** Shows the status of the simulation: `*** Restricted Version with 32768 Byte Code Size Limit` and `*** Currently used: 72 Bytes (0%)`.
- Memory:** A memory dump starting at address 0. The first few lines show:
  - `0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF`
  - `0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 04 00 00 04`
  - `0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00`
  - `0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
  - `0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`

The bottom status bar indicates the simulation is running, with a time of 0.00000000 sec, L9 C:46, and CAP NUM.

# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The main workspace is divided into several panes:

- Registers:** A list of ARM registers (R0-R15, CPSR, SPSR) with their current values. R0 is 0x000000FC, R1 is 0xFFFFF04, R2 is 0x000000F0, R3 is 0xFFFFF10, R4 is 0x00000000, R5 is 0x00000000, R6 is 0x00000000, R7 is 0x00000000, R8 is 0x00000000, R9 is 0x00000000, R10 is 0x00000000, R11 is 0x00000000, R12 is 0x00000000, R13 (SP) is 0x00000000, R14 (LR) is 0x00000000, R15 (PC) is 0x00000010, CPSR is 0x000000D3, and SPSR is 0x00000000.
- Disassembly:** A list of assembly instructions with their addresses and values. The instruction at address 0x00000040 is highlighted: `0x00000040 00000000 ANDEQ R0,R0,R0`.
- ex1.asm:** The source assembly file. It contains several data definition directives:
  - `12 ; DCB can start at any memory location.`
  - `13 two DCW 2 ; Must start at an even address location.`
  - `14 ; One byte to be skipped to adjust the location counter.`
  - `15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,`
  - `16 ; TO MAKE THE ADDRESS MULTIPLE OF 4`
  - `17 four & 4,4 ; the "&" here means DCD`
  - `18 ; DCD must start at a multiple of 4 address location`
  - `19 DCD 2_1010 ; Binary positive number`
  - `20 DCD -2_1010 ; Binary negative number`
  - `21 DCD 8_12345670 ; Octal positive number`
  - `22 DCD -8_12345670 ; Octal negative number`
  - `23`
  - `24 DCB 1 ; Any data directive can be without label`
  - `25 data_1 SPACE 5 ; reserves a ZEROED 5 bytes block of memory`
  - `26 data_2 % 5 ; the "%" here means SPACE`
  - `27 ALIGN ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE`
  - `28 ; OF 4 ADDRESS LOCATION`
  - `29 data_3 SPACE 5`
  - `30 END`
- Command:** A text area showing the status of the simulation: `*** Restricted Version with 32768 Byte Code Size Limit` and `*** Currently used: 72 Bytes (0%)`. Below this are commands: `ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet`.
- Memory:** A memory dump showing the contents of memory starting at address 0. The dump is organized into rows of 16 bytes each, with the first row starting at 0x00000000 and ending at 0x0000000F. The second row starts at 0x00000010 and ends at 0x0000001F. The third row starts at 0x00000020 and ends at 0x0000002F. The fourth row starts at 0x00000030 and ends at 0x0000003F. The fifth row starts at 0x00000040 and ends at 0x0000004F. The sixth row starts at 0x00000050 and ends at 0x0000005F. The seventh row starts at 0x00000060 and ends at 0x0000006F. The eighth row starts at 0x00000070 and ends at 0x0000007F. The ninth row starts at 0x00000080 and ends at 0x0000008F. The tenth row starts at 0x00000090 and ends at 0x0000009F. The eleventh row starts at 0x000000A0 and ends at 0x000000AF. The twelfth row starts at 0x000000B0 and ends at 0x000000BF. The thirteenth row starts at 0x000000C0 and ends at 0x000000CF. The fourteenth row starts at 0x000000D0 and ends at 0x000000DF. The fifteenth row starts at 0x000000E0 and ends at 0x000000EF. The sixteenth row starts at 0x000000F0 and ends at 0x000000FF.



# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations, editing, and debugging.

The **Registers** window on the left shows the current state of ARM registers. The **Disassembly** window in the center displays the following assembly code:

```

0x0000002C 0029CBB8 (???) EQ
0x00000030 FFD63448 (???)
0x00000034 01000000 (???) EQ
0x00000038 00000000 ANDEQ R0,R0,R0
0x0000003C 00000000 ANDEQ R0,R0,R0
0x00000040 00000000 ANDEQ R0,R0,R0

```

The **ex1.asm\*** window shows the source assembly code with comments:

```

12 ; DCB can start at any memory location.
13 two DCW 2 ; Must start at an even address location.
14 ; One byte to be skipped to adjust the location counter.
15 ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16 ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four & 4,4 ; the "&" here means DCD
18 ; DCD must start at a multiple of 4 address location
19 DCD 2_1010 ; Binary positive number
20 DCD -2_1010 ; Binary negative number
21 DCD 8_12345670 ; Octal positive number
22 DCD -8_12345670 ; Octal negative number
23
24 DCB 1 ; Any data directive can be without label
25 data_1 SPACE 5 ; reserves a ZEROED 5 bytes block of memory
26 data_2 % 5 ; the "%" here means SPACE
27 ALIGN ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28 ; OF 4 ADDRESS LOCATION
29 data_3 SPACE 5
30 END

```

The **Command** window at the bottom left shows the status: "Restricted Version with 32768 Byte Code Size Limit" and "Currently used: 72 Bytes (0%)". It also includes a list of break commands: ASSIGN, BreakDisable, BreakEnable, BreakKill, BreakList, BreakSet.

The **Memory** window at the bottom right shows a memory dump starting at address 0:

```

Address: 0
0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

The status bar at the bottom indicates the simulation is running, with a time of 0.00000000 sec, L9 C:46, and CAP NUM.

# Data Definition Directives

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Lists ARM registers R0 through R15, CPSR, and SPSR with their current values. R0 is 0x000000FC, R1 is 0xFFFFF04, R2 is 0x000000F0, R3 is 0xFFFFF10, R4 is 0x00000000, R5 is 0x00000000, R6 is 0x00000000, R7 is 0x00000000, R8 is 0x00000000, R9 is 0x00000000, R10 is 0x00000000, R11 is 0x00000000, R12 is 0x00000000, R13 (SP) is 0x00000000, R14 (LR) is 0x00000000, R15 (PC) is 0x00000010, CPSR is 0x000000D3, and SPSR is 0x00000000.
- Disassembly Window:** Shows assembly instructions at memory addresses 0x0000002C to 0x00000040. The instruction at 0x00000040 is highlighted: `0x00000040 00000000 ANDEQ R0,R0,R0`.
- Source Code Window (ex1.asm):** Contains assembly code with comments. A callout bubble points to the `SPACE 5` directive on line 26, stating: "The \"%\" here means SPACE".
 

```

12                                     ; DCB can start at any memory location.
13 two      DCW      2      ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four      &      4,4      ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19      DCD      2_1010      ; Binary positive number
20      DCD      -2_1010     ; Binary negative number
21      DCD      8_12345670  ; Octal positive number
22      DCD      -8_12345670 ; Octal negative number
23
24      DCB      1      ; Any data directive can be without label
25 data_1  SPACE 5      ; reserves a ZEROED 5 bytes block of memory
26 data_2  %      5      ; the "%" here means SPACE
27      ALIGN      ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28                                     ; OF 4 ADDRESS LOCATION
29 data_3  SPACE 5
30      END
      
```
- Memory Window:** Shows a memory dump starting at address 0. The first few lines of memory are:
 

```

0x00000000: E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF
0x00000012: FF FE 01 01 01 01 41 00 00 02 00 00 00 04 00 00 04
0x00000024: 00 00 00 0A FF FF FF F6 00 29 CB B8 FF D6 34 48 01 00
0x00000036: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      
```
- Status Bar:** Shows "Simulation", "t1: 0.00000000 sec", "L9 C:46", and "CAP NUM".

# Data Definition Directives

The screenshot shows the uVision4 IDE with the following components:

- Registers Panel:** Lists ARM registers R0 through R15, CPSR, and SPSR with their current values. R0 is 0x000000FC, R1 is 0xFFFFF04, R2 is 0x000000F0, R3 is 0xFFFFF10, R4 is 0x00000000, R5 is 0x00000000, R6 is 0x00000000, R7 is 0x00000000, R8 is 0x00000000, R9 is 0x00000000, R10 is 0x00000000, R11 is 0x00000000, R12 is 0x00000000, R13 (SP) is 0x00000000, R14 (LR) is 0x00000000, R15 (PC) is 0x00000010, CPSR is 0x000000D3, and SPSR is 0x00000000.
- Disassembly Panel:** Shows assembly instructions at addresses 0x0000002C to 0x00000040. The instruction at 0x00000040 is `ANDEQ R0,R0,R0`, which is highlighted in yellow.
- Assembly Editor (ex1.asm):** Contains the following code:
 

```

12                                     ; DCB can start at any memory location.
13 two      DCW      2      ; Must start at an even address location.
14                                     ; One byte to be skipped to adjust the location counter.
15                                     ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16                                     ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four      &      4,4      ; the "&" here means DCD
18                                     ; DCD must start at a multiple of 4 address location
19      DCD      2_1010      ; Binary positive number
20      DCD      -2_1010     ; Binary negative number
21      DCD      8_12345670  ; Octal positive number
22      DCD      -8_12345670 ; Octal negative number
23
24      DCB      1      ; Any data directive can be without label
25 data_1 SPACE 5      ; reserves a ZEROED 5 bytes block of memory
26 data_2 %      5      ; the "%" here means SPACE
27      ALIGN      ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28                                     ; OF 4 ADDRESS LOCATION
29 data_3 SPACE 5
30      END
      
```
- Memory Panel:** Shows a memory dump starting at address 0. The first row of data is `E3 A0 00 FC E3 E0 10 FB E3 A0 20 F0 E3 E0 30 EF EA FF`.
- Call Stack + Locals:** Shows the current stack frame.
- Simulation Status:** Shows the simulation is running at 0.00000000 sec, with L9 C:46 and CAP NUM.

A yellow callout bubble with the text "Skip one byte to address 0x40" points to the `ALIGN` directive in the assembly code.



# Data Definition Directives

The screenshot displays the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations, editing, and debugging.

The **Registers** window on the left shows the current state of ARM registers. The **Disassembly** window in the center displays the assembly code, with the instruction `ANDEQ R0,R0,R0` highlighted at address `0x00000040`. The **ex1.asm\*** window shows the source code with comments explaining data directives:

```

12          ; DCB can start at any memory location.
13 two      DCW  2          ; Must start at an even address location.
14          ; One byte to be skipped to adjust the location counter.
15          ; IF YOU PUT ALIGN BEFORE THIS DCW, IT WILL SKIP 3 BYTES, NOT JUST ONE,
16          ; TO MAKE THE ADDRESS MULTIPLE OF 4
17 four     & 4,4          ; the "&" here means DCD
18          ; DCD must start at a multiple of 4 address location
19          DCD  2_1010      ; Binary positive number
20          DCD -2_1010      ; Binary negative number
21          DCD  8_12345670  ; Octal positive number
22          DCD -8_12345670  ; Octal negative number
23
24          DCB  1          ; Any data directive can be without label
25 data_1    SPACE 5        ; reserves a ZEROED 5 bytes block of memory
26 data_2    % 5            ; the "%" here means SPACE
27          ALIGN           ; ADVANCE THE LOCATION COUNTER TO THE NEXT MULTIPLE
28          ; OF 4 ADDRESS LOCATION
29 data_3    SPACE 5
30          END

```

The **Command** window at the bottom left shows the status: "Restricted Version with 32768 Byte Code Size Limit" and "Currently used: 72 Bytes (0%)". The **Memory** window at the bottom right displays a memory dump starting at address 0, showing hexadecimal values and their corresponding ASCII representations.

# ARM pseudo-instructions

- The ARM assembler supports a number of pseudo-instructions that are translated into the appropriate combination of ARM instructions at assembly time.
- Consider the following assembly program:

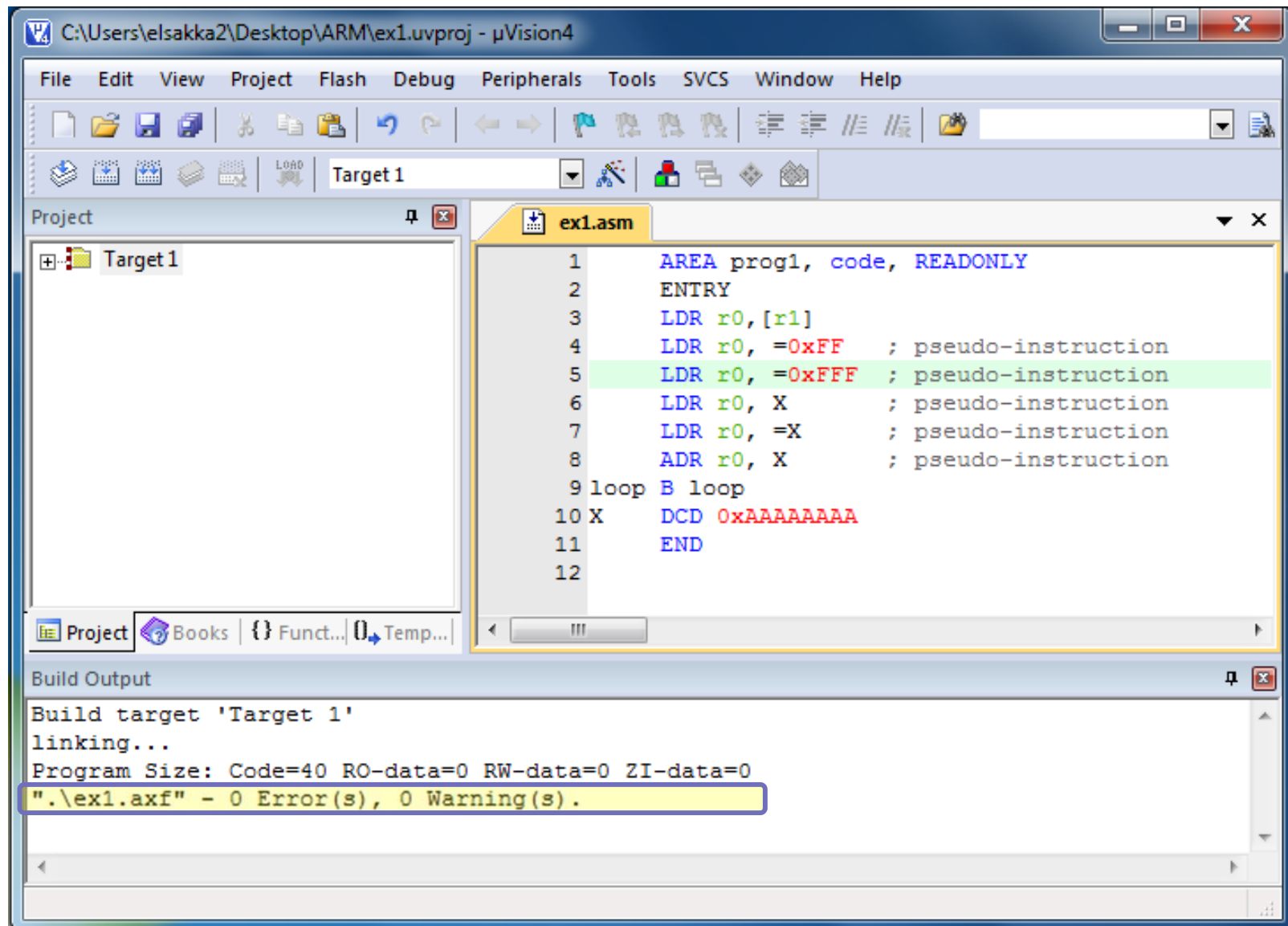
```

AREA prog1, code, READONLY
ENTRY
LDR r0, [r1]
LDR r0, =0xFF      ; pseudo-instruction
LDR r0, =0xFFF     ; pseudo-instruction
LDR r0, X          ; pseudo-instruction
LDR r0, =X         ; pseudo-instruction
ADR r0, X          ; pseudo-instruction

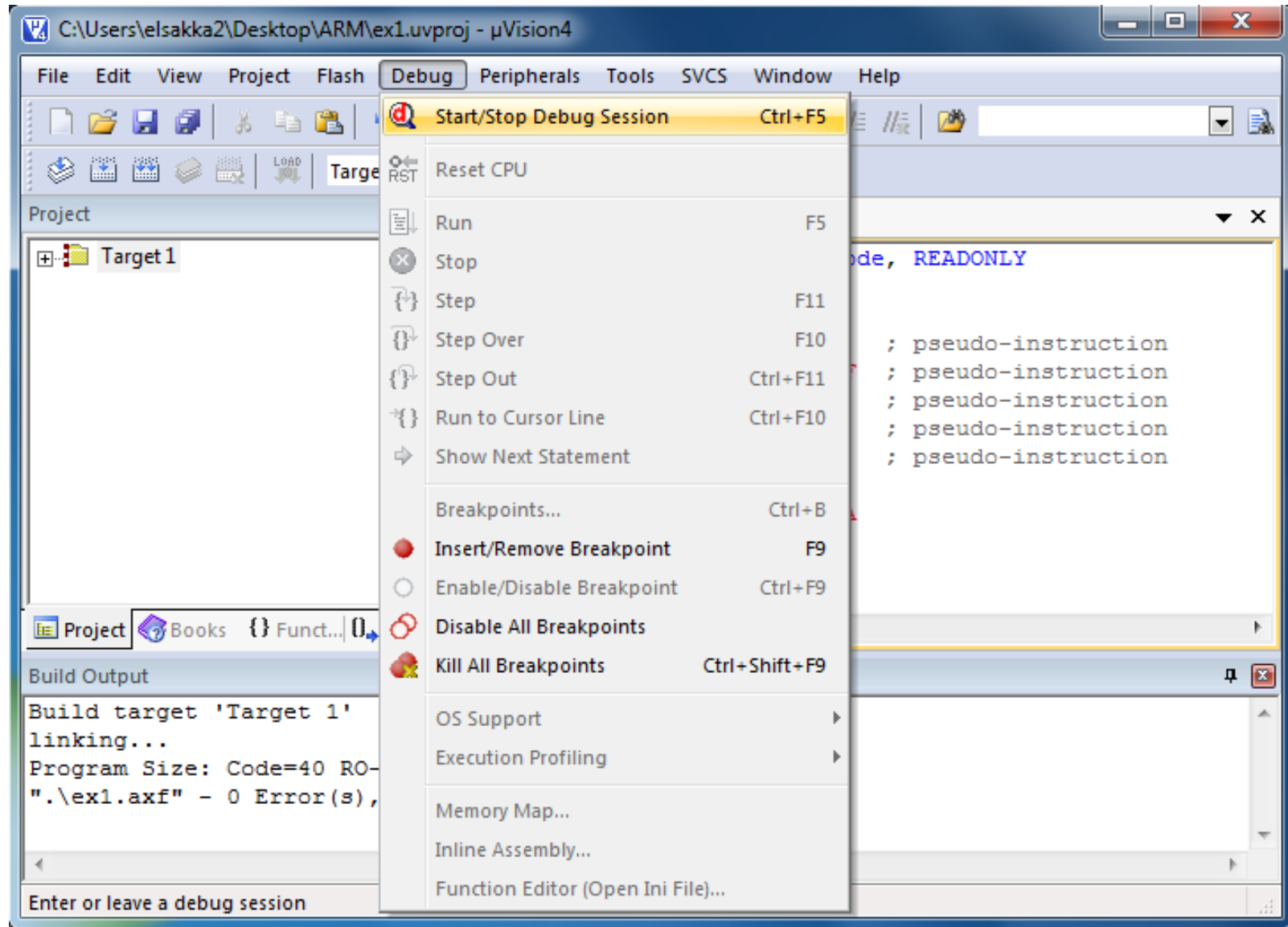
loop B loop
X    DCD 0xAAAAAAAA
END

```

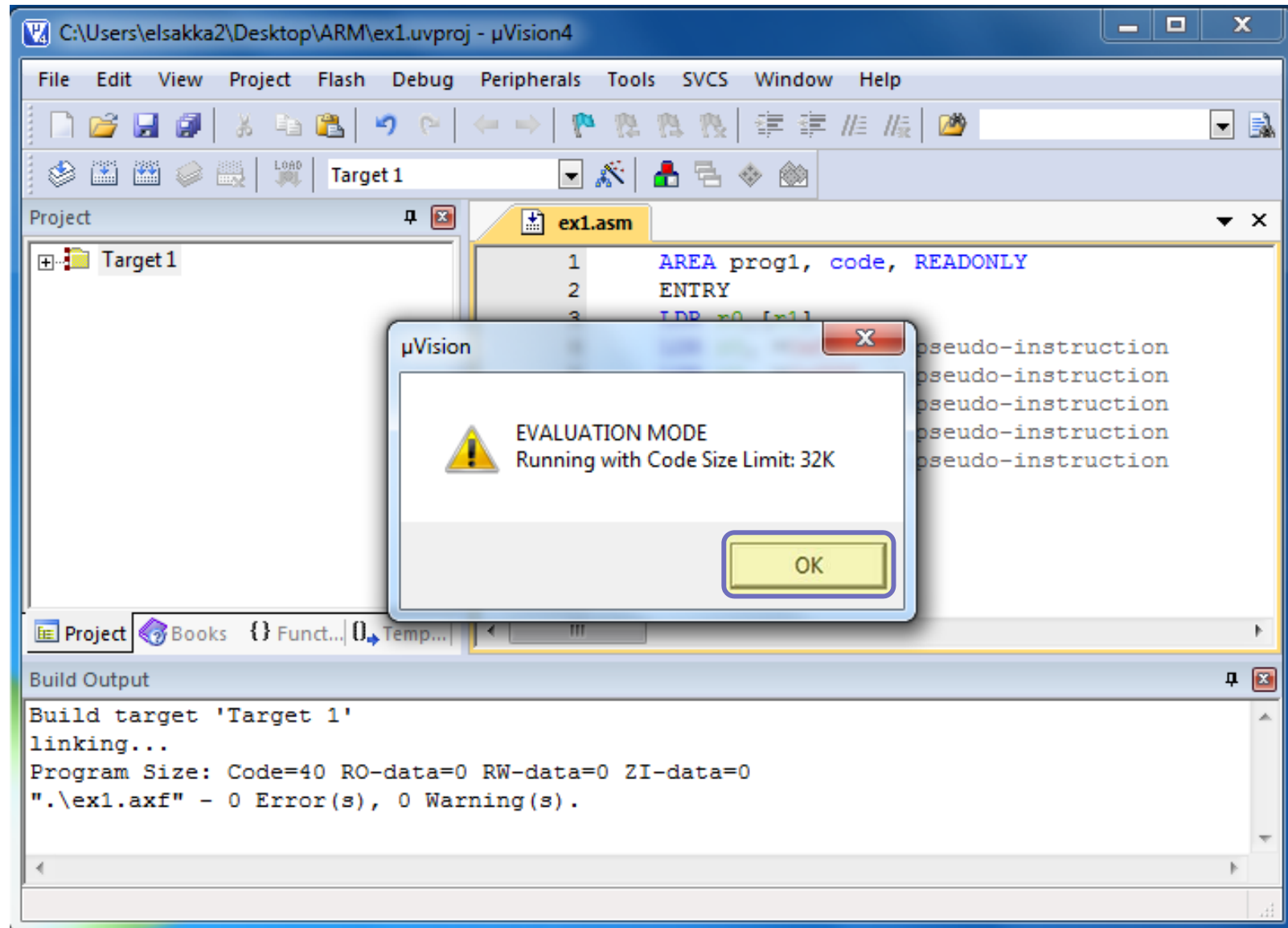
# ARM pseudo-instructions



# ARM pseudo-instructions



# ARM pseudo-instructions



# ARM pseudo-instructions

The screenshot shows the µVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations, editing, and debugging. The 'Registers' window on the left lists registers R0 through R15, CPSR, and SPSR, all with a value of 0x00000000. The 'Disassembly' window shows a list of instructions, with the first instruction highlighted: 3: LDR r0,[r1] at address 0x00000000. The 'Source' window at the bottom shows the corresponding assembly code for 'ex1.asm', including directives like AREA, ENTRY, and pseudo-instructions like LDR r0, =0xFF and LDR r0, =0xFFFF. A blue callout bubble points to the 'Step' button in the toolbar, indicating the next action.

**Registers**

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000

**Disassembly**

```

3:      LDR r0,[r1]
0x00000000 E5910000 LDR      R0,[R1]
4:      LDR r0,=0xFF ; pseudo-instruction
0x00000004 E3A000FF MOV      R0,#0x000000FF
5:      LDR r0,=0xFFFF ; pseudo-instruction
0x00000008 E59F0010 LDR      R0,[PC,#0x0010]
6:      LDR r0,X ; pseudo-instruction
0x0000000C E59F0008 LDR      R0,[PC,#0x0008]
7:      LDR r0,=X ; pseudo-instruction
0x00000010 E59F000C LDR      R0,[PC,#0x000C]
8:      ADR r0,X ; pseudo-instruction
0x00000014 E28F0000 ADD      R0,PC,#0x00000000
9:      loop B loop
0x00000018 EAFFFFFE B        0x00000018
0x0000001C AAAAAAAA BGE      0xFEAAAAAC
0x00000020 0000FFFF ???EQ
0x00000024 0000001C ANDEQ    R0,R0,R12,LSL R0
0x00000028 00000000 ANDEQ    R0,R0,R0
  
```

**Source (ex1.asm)**

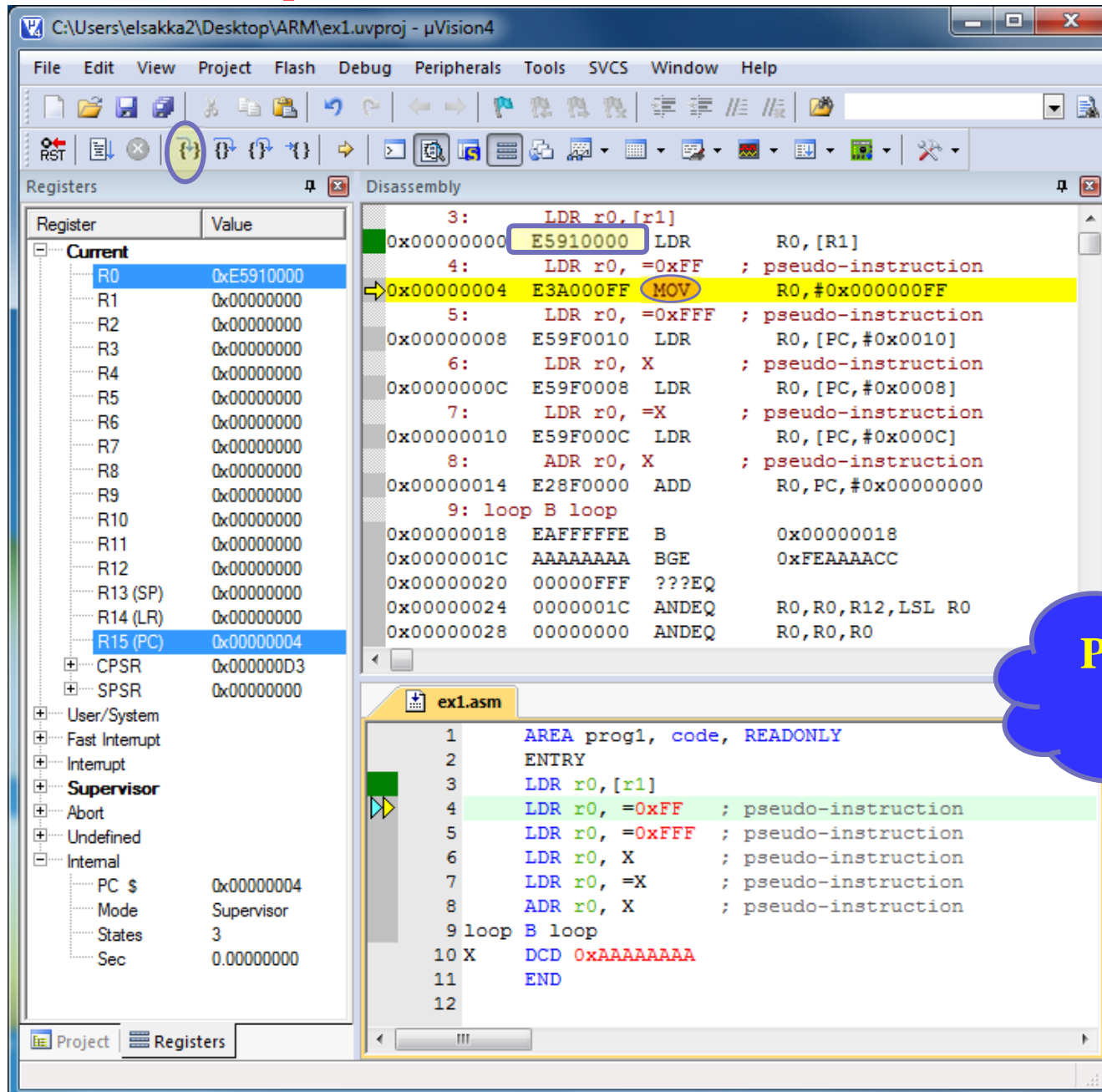
```

1      AREA prog1, code, READONLY
2      ENTRY
3      LDR r0,[r1]
4      LDR r0,=0xFF ; pseudo-instruction
5      LDR r0,=0xFFFF ; pseudo-instruction
6      LDR r0,X ; pseudo-instruction
7      LDR r0,=X ; pseudo-instruction
8      ADR r0,X ; pseudo-instruction
9      loop B loop
10     X      DCD 0xAAAAAAAA
11     END
12
  
```

Press Step,  
or F11



# ARM pseudo-instructions



The screenshot shows the µVision4 IDE interface. The 'Registers' window on the left lists registers R0 through R15, with R15 (PC) highlighted. The 'Disassembly' window shows the following assembly code:

```

3:      LDR r0,[r1]
0x00000000 E5910000 LDR      R0,[R1]
4:      LDR r0,=0xFF ; pseudo-instruction
0x00000004 E3A000FF MOV      R0,#0x000000FF
5:      LDR r0,=0xFFF ; pseudo-instruction
0x00000008 E59F0010 LDR      R0,[PC,#0x0010]
6:      LDR r0,X ; pseudo-instruction
0x0000000C E59F0008 LDR      R0,[PC,#0x0008]
7:      LDR r0,=X ; pseudo-instruction
0x00000010 E59F000C LDR      R0,[PC,#0x000C]
8:      ADR r0,X ; pseudo-instruction
0x00000014 E28F0000 ADD      R0,PC,#0x00000000
9: loop B loop
0x00000018 EAffffff B 0x00000018
0x0000001C AAAAAAAA BGE 0xFEAAAAACC
0x00000020 0000FFFF ???EQ
0x00000024 0000001C ANDEQ R0,R0,R12,LSL R0
0x00000028 00000000 ANDEQ R0,R0,R0

```

The 'ex1.asm' source file window shows the following assembly code:

```

1 AREA prog1, code, READONLY
2 ENTRY
3 LDR r0,[r1]
4 LDR r0,=0xFF ; pseudo-instruction
5 LDR r0,=0xFFF ; pseudo-instruction
6 LDR r0,X ; pseudo-instruction
7 LDR r0,=X ; pseudo-instruction
8 ADR r0,X ; pseudo-instruction
9 loop B loop
10 X DCD 0xAAAAAAAA
11 END
12

```

A blue callout bubble with the text "Press Step, or F11" points to the 'Step' button in the toolbar.



# ARM pseudo-instructions

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of ARM registers. R0 is 0x000000FF, R15 (PC) is 0x00000008.
- Disassembly Window:** Shows the disassembled instructions. A red bracket highlights the instructions from 3 to 9. The instruction at address 0x00000008 is `LDR r0, [PC, #0x0010]`.
- Source Code Window (ex1.asm):** Shows the assembly source code. A blue bracket highlights the instructions from 3 to 9. The instruction at line 5 is `LDR r0, =0xFFFF`.

Annotations:

- A red cloud bubble on the right asks: "How is this offset calculated?"
- A blue cloud bubble on the right says: "Press Step, or F11"

# ARM pseudo-instructions

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x000000FF
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000000C
CPSR	0x000000D3
SPSR	0x00000000
- Disassembly Window:**

```

3:    LDR r0,[r1]
0x00000000 E5910000 LDR    R0,[R1]
4:    LDR r0,=0xFF ; pseudo-instruction
0x00000004 E3A000FF MOV    R0,#0x000000FF
5:    LDR r0,=0xFFFF ; pseudo-instruction
0x00000008 E59F0010 LDR    R0,[PC,#0x0010]
6:    LDR r0,X ; pseudo-instruction
0x0000000C E59F0008 LDR    R0,[PC,#0x0008]
7:    LDR r0,=X ; pseudo-instruction
0x00000010 E59F000C LDR    R0,[PC,#0x000C]
8:    ADR r0,X ; pseudo-instruction
0x00000014 E28F0000 ADD    R0,PC,#0x00000000
9:    loop B loop
0x00000018 EAFFFFF0 B       0x00000018
0x0000001C AAAAAAAA BGE    0xFEAAAAAC
0x00000020 0000FFFF ???EQ
0x00000024 0000001C ANDEQ   R0,R0,R12,LSL R0
0x00000028 00000000 ANDEQ   R0,R0,R0

```
- Source Window (ex1.asm):**

```

1  AREA prog1, code, READONLY
2  ENTRY
3  LDR r0,[r1]
4  LDR r0,=0xFF ; pseudo-instruction
5  LDR r0,=0xFFFF ; pseudo-instruction
6  LDR r0,X ; pseudo-instruction
7  LDR r0,=X ; pseudo-instruction
8  ADR r0,X ; pseudo-instruction
9  loop B loop
10 X DCD 0xAAAAAAAA
11 END
12

```

How is this  
offset  
calculated?

Press Step,  
or F11

# ARM pseudo-instructions

The screenshot shows the µVision4 IDE with the following components:

- Registers Window:** Shows the current state of ARM registers. R0 is highlighted with a value of 0xAAAAAAAA. R15 (PC) is highlighted with a value of 0x00000010.
- Disassembly Window:** Shows the disassembled code. A red bracket highlights several pseudo-instructions:
  - 3: LDR r0, [r1]
  - 4: LDR r0, =0xFF ; pseudo-instruction
  - 5: LDR r0, =0xFFFF ; pseudo-instruction
  - 6: LDR r0, X ; pseudo-instruction
  - 7: LDR r0, =X ; pseudo-instruction
  - 8: ADR r0, X ; pseudo-instruction
- Assembly Window (ex1.asm):** Shows the source code corresponding to the disassembly. Lines 4 through 8 are highlighted in green, matching the pseudo-instructions in the Disassembly window.
 

```

1  AREA prog1, code, READONLY
2  ENTRY
3  LDR r0, [r1]
4  LDR r0, =0xFF ; pseudo-instruction
5  LDR r0, =0xFFFF ; pseudo-instruction
6  LDR r0, X ; pseudo-instruction
7  LDR r0, =X ; pseudo-instruction
8  ADR r0, X ; pseudo-instruction
9  loop B loop
10 X DCD 0xAAAAAAAA
11 END
12
      
```

How is this offset calculated?

Press Step, or F11

# ARM pseudo-instructions

Compare the translations of  
**LDR r0, =X**  
 and  
**ADR r0, X**

Registers

Register	Value
<b>Current</b>	
R0	0x0000001C
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
<b>R15 (PC)</b>	<b>0x00000014</b>
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
<b>Supervisor</b>	
Abort	
Undefined	
Internal	
PC \$	0x00000014
Mode	Supervisor
States	13
Sec	0.00000000

Disassembly

```

3:      LDR r0,[r1]
0x00000000 E5910000 LDR      R0,[R1]
4:      LDR r0,=0xFF ; pseudo-instruction
0x00000004 E3A000FF MOV      R0,#0x000000FF
5:      LDR r0,=0xFFFF ; pseudo-instruction
0x00000008 E59F0010 LDR      R0,[PC,#0x0010]
6:      LDR r0,X ; pseudo-instruction
0x0000000C E59F0008 LDR      R0,[PC,#0x0008]
7:      LDR r0,=X ; pseudo-instruction
0x00000010 E59F000C LDR      R0,[PC,#0x000C]
8:      ADR r0,X ; pseudo-instruction
0x00000014 E28F0000 ADD      R0,PC,#0x00000000
9: loop B loop
0x00000018 EAFFFFFE B        0x00000018
0x0000001C AAAAAAAA BGE     0xFEAAAAAC
0x00000020 0000FFFF ???EQ
0x00000024 0000001C ANDEQ   R0,R0,R12,LSL R0
0x00000028 00000000 ANDEQ   R0,R0,R0
  
```

ex1.asm

```

1  AREA prog1, code, READONLY
2  ENTRY
3  LDR r0,[r1]
4  LDR r0,=0xFF ; pseudo-instruction
5  LDR r0,=0xFFFF ; pseudo-instruction
6  LDR r0,X ; pseudo-instruction
7  LDR r0,=X ; pseudo-instruction
8  ADR r0,X ; pseudo-instruction
9 loop B loop
10 X DCD 0xAAAAAAAA
11
12
  
```

How is this  
 offset  
 calculated?

Press Step,  
 or F11



# ARM pseudo-instructions

Same  
address (no  
change)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Reg	Value
R0	0x0000001C
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000018
CPSR	0x000000D3
SPSR	0x00000000
- Disassembly Window:**

```

3:      LDR r0,[r1]
0x00000000 E5910000 LDR      R0,[R1]
4:      LDR r0,=0xFF ; pseudo-instruction
0x00000004 E3A000FF MOV      R0,#0x000000FF
5:      LDR r0,=0xFFF ; pseudo-instruction
0x00000008 E59F0010 LDR      R0,[PC,#0x0010]
6:      LDR r0,X ; pseudo-instruction
0x0000000C E59F0008 LDR      R0,[PC,#0x0008]
7:      LDR r0,=X ; pseudo-instruction
0x00000010 E59F000C LDR      R0,[PC,#0x000C]
8:      ADR r0,X ; pseudo-instruction
0x00000014 E28F0000 ADD      R0,PC,#0x00000000
9: loop B loop
0x00000018 EAffffff B        0x00000018
0x0000001C AAAAAAAA BGE      0xFEAAAAAC
0x00000020 0000FFFF ???EQ
0x00000024 0000001C ANDEQ   R0,R0,R12,LSL R0
0x00000028 00000000 ANDEQ   R0,R0,R0

```
- Source Window (ex1.asm):**

```

1  AREA prog1, code, READONLY
2  ENTRY
3  LDR r0,[r1]
4  LDR r0,=0xFF ; pseudo-instruction
5  LDR r0,=0xFFF ; pseudo-instruction
6  LDR r0,X ; pseudo-instruction
7  LDR r0,=X ; pseudo-instruction
8  ADR r0,X ; pseudo-instruction
9 loop B loop
10 X DCD 0xAAAAAAAA
11 END
12

```



# ARM pseudo-instructions

- Consider we changed the previous program as follow:

```
AREA prog1, code, READONLY
ENTRY
LDR r0,[r1]
LDR r0, =0xFF      ; pseudo-instruction
LDR r0, =0xFFF     ; pseudo-instruction
LDR r0, X           ; pseudo-instruction
LDR r0, =X          ; pseudo-instruction
ADR r0, X           ; pseudo-instruction

loop B loop
X DCD 0xAAAAAAAA
END
```

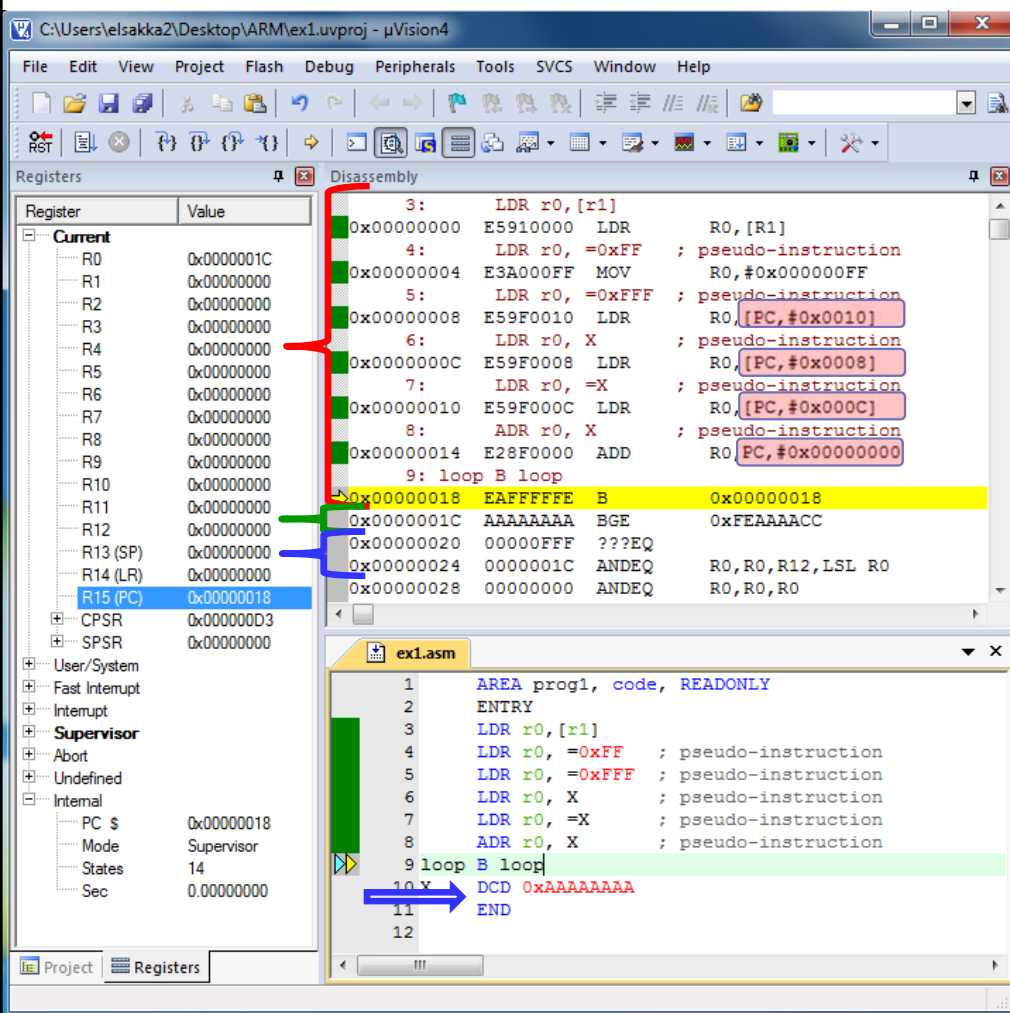
```
AREA prog1, code, READONLY
ENTRY
LDR r0,[r1]
LDR r0, =0xFF      ; pseudo-instruction
LDR r0, =0xFFF     ; pseudo-instruction
LDR r0, X           ; pseudo-instruction
LDR r0, =X          ; pseudo-instruction
ADR r0, X           ; pseudo-instruction

loop B loop

AREA prog1, data, READONLY
X DCD 0xAAAAAAAA
END
```

- What is the effect of this change on the generated code?

# ARM pseudo-instructions



Registers

Register	Value
R0	0x0000001C
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000018
CPSR	0x000000D3
SPSR	0x00000000

Disassembly

```

3: LDR r0, [r1] ; E5910000 LDR R0, [R1]
4: LDR r0, =0xFF ; pseudo-instruction E3A000FF MOV R0, #0x000000FF
5: LDR r0, =0xFFFF ; pseudo-instruction E59F0010 LDR R0, [PC, #0x0010]
6: LDR r0, X ; pseudo-instruction E59F0008 LDR R0, [PC, #0x0008]
7: LDR r0, =X ; pseudo-instruction E59F000C LDR R0, [PC, #0x000C]
8: ADR r0, X ; pseudo-instruction E28F0000 ADD R0, PC, #0x00000000
9: loop B loop
10: DCD 0xAAAAAAAA
11: END

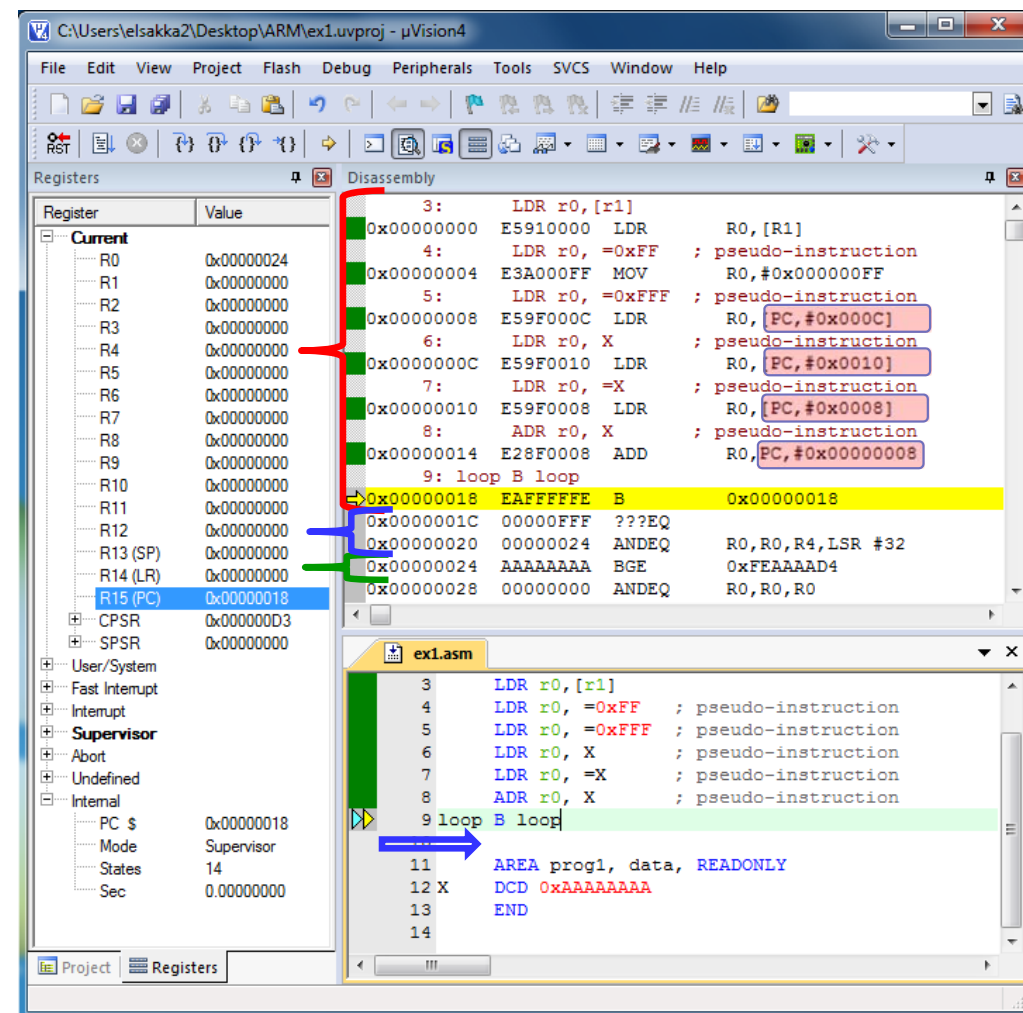
```

ex1.asm

```

1 AREA prog1, code, READONLY
2 ENTRY
3 LDR r0, [r1]
4 LDR r0, =0xFF ; pseudo-instruction
5 LDR r0, =0xFFFF ; pseudo-instruction
6 LDR r0, X ; pseudo-instruction
7 LDR r0, =X ; pseudo-instruction
8 ADR r0, X ; pseudo-instruction
9 loop B loop
10 DCD 0xAAAAAAAA
11 END

```



Registers

Register	Value
R0	0x00000024
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000018
CPSR	0x000000D3
SPSR	0x00000000

Disassembly

```

3: LDR r0, [r1] ; E5910000 LDR R0, [R1]
4: LDR r0, =0xFF ; pseudo-instruction E3A000FF MOV R0, #0x000000FF
5: LDR r0, =0xFFFF ; pseudo-instruction E59F0010 LDR R0, [PC, #0x0010]
6: LDR r0, X ; pseudo-instruction E59F0008 LDR R0, [PC, #0x0008]
7: LDR r0, =X ; pseudo-instruction E59F000C LDR R0, [PC, #0x000C]
8: ADR r0, X ; pseudo-instruction E28F0000 ADD R0, PC, #0x00000000
9: loop B loop
10: DCD 0xAAAAAAAA
11: END

```

ex1.asm

```

1 AREA prog1, code, READONLY
2 ENTRY
3 LDR r0, [r1]
4 LDR r0, =0xFF ; pseudo-instruction
5 LDR r0, =0xFFFF ; pseudo-instruction
6 LDR r0, X ; pseudo-instruction
7 LDR r0, =X ; pseudo-instruction
8 ADR r0, X ; pseudo-instruction
9 loop B loop
10 DCD 0xAAAAAAAA
11 END

```

# ARM pseudo-instructions

- Consider the following assembly program:

```
AREA prog1, code, READONLY
ENTRY
MOV r0, #0xEE
MOV r1, r0
NEG r2, r0
MVN r3, r0
loop B loop
END
```

# ARM pseudo-instructions

The screenshot shows the uVision4 IDE interface. The 'Registers' window on the left lists registers R0 through R15, CPSR, and SPSR. The 'Disassembly' window on the right shows the following assembly code:

```

3:      MOV r0, #0xEE
0x00000000 E3A000EE MOV      R0,#0x000000EE
4:      MOV r1,r0
0x00000004 E1A01000 MOV      R1,R0
5:      NEG r2,r0
0x00000008 E2602000 RSB       R2,R0,#0x00000000
6:      MVN r3,r0
0x0000000C E1E03000 MVN      R3,R0
7:      loop B loop
0x00000010 EAF00000 B         0x00000010
0x00000014 00000000 ANDEQ    R0,R0,R0
0x00000018 00000000 ANDEQ    R0,R0,R0
0x0000001C 00000000 ANDEQ    R0,R0,R0
0x00000020 00000000 ANDEQ    R0,R0,R0
0x00000024 00000000 ANDEQ    R0,R0,R0
0x00000028 00000000 ANDEQ    R0,R0,R0
0x0000002C 00000000 ANDEQ    R0,R0,R0
0x00000030 00000000 ANDEQ    R0,R0,R0

```

The 'ex1.asm\*' window at the bottom shows the source code for the assembly file:

```

1      AREA prog1, code, READONLY
2      ENTRY
3      MOV r0, #0xEE
4      MOV r1,r0
5      NEG r2,r0
6      MVN r3,r0
7      loop B loop
8      END
9

```

Press Step,  
or F11

# ARM pseudo-instructions

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x000000EE
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000004
CPSR	0x000000D3
SPSR	0x00000000
- Disassembly Window:**

```

3:      MOV r0, #0xEE
0x00000000 E3A000EE MOV      R0,#0x000000EE
4:      MOV r1,r0
0x00000004 E1A01000 MOV      R1,R0
5:      NEG r2,r0
0x00000008 E2602000 RSB      R2,R0,#0x00000000
6:      MVN r3,r0
0x0000000C E1E03000 MVN      R3,R0
7: loop B loop
0x00000010 EAF00000 B        0x00000010
0x00000014 00000000 ANDEQ    R0,R0,R0
0x00000018 00000000 ANDEQ    R0,R0,R0
0x0000001C 00000000 ANDEQ    R0,R0,R0
0x00000020 00000000 ANDEQ    R0,R0,R0
0x00000024 00000000 ANDEQ    R0,R0,R0
0x00000028 00000000 ANDEQ    R0,R0,R0
0x0000002C 00000000 ANDEQ    R0,R0,R0
0x00000030 00000000 ANDEQ    R0,R0,R0

```
- Source Window (ex1.asm):**

```

1  AREA prog1, code, READONLY
2  ENTRY
3  MOV r0, #0xEE
4  MOV r1,r0
5  NEG r2,r0
6  MVN r3,r0
7 loop B loop
8  END
9

```

A blue callout bubble with the text "Press Step, or F11" points to the Step button in the toolbar.



# ARM pseudo-instructions

The screenshot shows the µVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons, with the Step button (a blue square with a white right-pointing arrow) circled in blue. Below the toolbar, the 'Registers' window on the left lists registers R0 through R15, with R15 (PC) selected. The 'Disassembly' window on the right shows assembly code with instructions like MOV, NEG, and RSB. The bottom window shows the source file 'ex1.asm' with assembly code including AREA, ENTRY, MOV, NEG, MVN, and a loop. A blue callout bubble with the text 'Press Step, or F11' points to the Step button in the toolbar.

**Registers**

Register	Value
R0	0x000000EE
R1	0x000000EE
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000008
CPSR	0x000000D3
SPSR	0x00000000

**Disassembly**

```

3:      MOV r0, #0xEE
0x00000000 E3A000EE MOV      R0,#0x000000EE
4:      MOV r1,r0
0x00000004 E1A01000 MOV      R1,R0
5:      NEG r2,r0
0x00000008 E2602000 RSB      R2,R0,#0x00000000
6:      MVN r3,r0
0x0000000C E1E03000 MVN      R3,R0
7: loop B loop
0x00000010 EAF000FE B        0x00000010
0x00000014 00000000 ANDEQ    R0,R0,R0
0x00000018 00000000 ANDEQ    R0,R0,R0
0x0000001C 00000000 ANDEQ    R0,R0,R0
0x00000020 00000000 ANDEQ    R0,R0,R0
0x00000024 00000000 ANDEQ    R0,R0,R0
0x00000028 00000000 ANDEQ    R0,R0,R0
0x0000002C 00000000 ANDEQ    R0,R0,R0
0x00000030 00000000 ANDEQ    R0,R0,R0
  
```

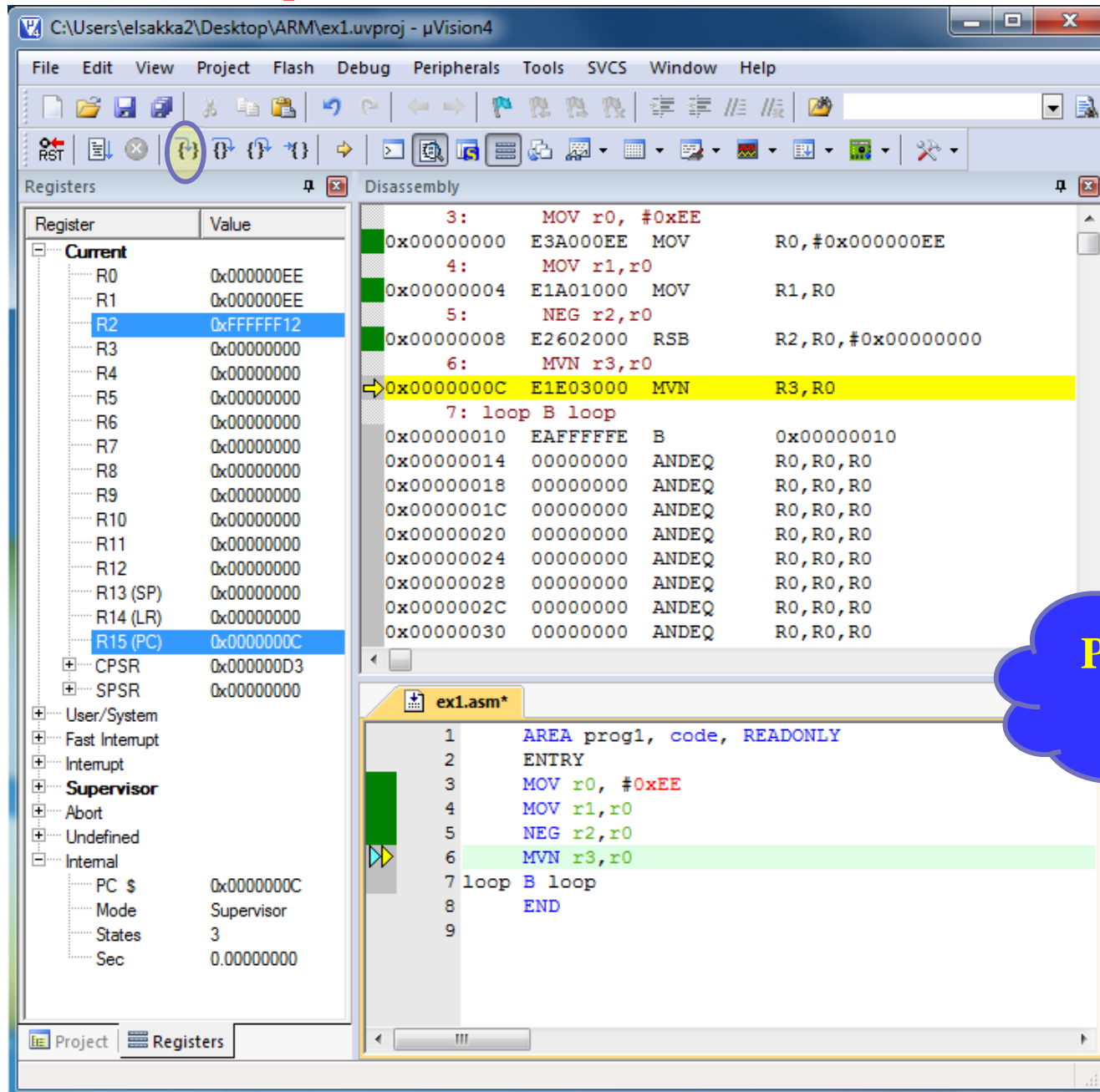
**ex1.asm\***

```

1  AREA prog1, code, READONLY
2  ENTRY
3  MOV r0, #0xEE
4  MOV r1,r0
5  NEG r2,r0
6  MVN r3,r0
7 loop B loop
8  END
9
  
```

Press Step,  
or F11

# ARM pseudo-instructions



Press Step,  
or F11

# ARM pseudo-instructions

The screenshot displays the µVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. Below the menu is a toolbar with various icons for file operations, editing, and debugging.

The main window is divided into three panes:

- Registers:** A table showing the current state of ARM registers.
 

Register	Value
R0	0x000000EE
R1	0x000000EE
R2	0xFFFFFFFF
R3	0xFFFFFFFF
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000010
CPSR	0x000000D3
SPSR	0x00000000
- Disassembly:** A list of assembly instructions with their addresses and hex values.
 

```

      3:      MOV r0, #0xEE
      0x00000000 E3A000EE MOV      R0, #0x000000EE
      4:      MOV r1, r0
      0x00000004 E1A01000 MOV      R1, R0
      5:      NEG r2, r0
      0x00000008 E2602000 RSB      R2, R0, #0x00000000
      6:      MVN r3, r0
      0x0000000C E1E03000 MVN      R3, R0
      7: loop B loop
      0x00000010 EAF000FE B        0x00000010
      0x00000014 00000000 ANDEQ    R0, R0, R0
      0x00000018 00000000 ANDEQ    R0, R0, R0
      0x0000001C 00000000 ANDEQ    R0, R0, R0
      0x00000020 00000000 ANDEQ    R0, R0, R0
      0x00000024 00000000 ANDEQ    R0, R0, R0
      0x00000028 00000000 ANDEQ    R0, R0, R0
      0x0000002C 00000000 ANDEQ    R0, R0, R0
      0x00000030 00000000 ANDEQ    R0, R0, R0
      
```
- ex1.asm:** The source assembly file being edited.
 

```

      1  AREA prog1, code, READONLY
      2  ENTRY
      3  MOV r0, #0xEE
      4  MOV r1, r0
      5  NEG r2, r0
      6  MVN r3, r0
      7  loop B loop
      8  END
      9
      
```

The bottom status bar shows the Project and Registers tabs.

# ARM pseudo-instructions

- Consider we changed the previous program as follow:

```
AREA prog1, code, READONLY
ENTRY
MOV r0, #0xEE
MOV r1, r0
NEG r2, r0
MVN r3, r0
loop B loop
END
```

```
AREA prog1, code, READONLY
ENTRY
MOV r0, #-0xEE
MOV r1, r0
NEG r2, r0
MVN r3, r0
loop B loop
END      END
```

- What is the effect of this change on the generated code?

# ARM pseudo-instructions

Registers

Register	Value
R0	0x000000EE
R1	0x000000EE
R2	0xFFFFFFFF
R3	0xFFFFFFFF
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000010
CPSR	0x000000D3
SPSR	0x00000000

Disassembly

```

3:  MOV r0, #0xFF
0x00000000  E3A000EE  MOV     R0, #0x000000EE
4:  MOV r1, r0
0x00000004  E1A01000  MOV     R1, R0
5:  NEG r2, r0
0x00000008  E2602000  RSB     R2, R0, #0x00000000
6:  MVN r3, r0
0x0000000C  E1E03000  MVN     R3, R0
7:  loop B loop
0x00000010  EAF00000  B       0x00000010
0x00000014  00000000  ANDEQ   R0, R0, R0
0x00000018  00000000  ANDEQ   R0, R0, R0
0x0000001C  00000000  ANDEQ   R0, R0, R0
0x00000020  00000000  ANDEQ   R0, R0, R0
0x00000024  00000000  ANDEQ   R0, R0, R0
0x00000028  00000000  ANDEQ   R0, R0, R0
0x0000002C  00000000  ANDEQ   R0, R0, R0
0x00000030  00000000  ANDEQ   R0, R0, R0

```

ex1.asm

```

1  AREA prog1, code, READONLY
2  ENTRY
3  MOV r0, #0xFF
4  MOV r1, r0
5  NEG r2, r0
6  MVN r3, r0
7  loop B loop
8  END
9

```

Registers

Register	Value
R0	0xFFFFFFFF
R1	0xFFFFFFFF
R2	0x000000EE
R3	0x000000ED
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000010
CPSR	0x000000D3
SPSR	0x00000000

Disassembly

```

3:  MOV r0, #-0xEE
0x00000000  E3E000ED  MVN     R0, #0x000000ED
4:  MOV r1, r0
0x00000004  E1A01000  MOV     R1, R0
5:  NEG r2, r0
0x00000008  E2602000  RSB     R2, R0, #0x00000000
6:  MVN r3, r0
0x0000000C  E1E03000  MVN     R3, R0
7:  loop B loop
0x00000010  EAF00000  B       0x00000010
0x00000014  00000000  ANDEQ   R0, R0, R0
0x00000018  00000000  ANDEQ   R0, R0, R0
0x0000001C  00000000  ANDEQ   R0, R0, R0
0x00000020  00000000  ANDEQ   R0, R0, R0
0x00000024  00000000  ANDEQ   R0, R0, R0
0x00000028  00000000  ANDEQ   R0, R0, R0
0x0000002C  00000000  ANDEQ   R0, R0, R0
0x00000030  00000000  ANDEQ   R0, R0, R0

```

ex1.asm

```

1  AREA prog1, code, READONLY
2  ENTRY
3  MOV r0, #-0xEE
4  MOV r1, r0
5  NEG r2, r0
6  MVN r3, r0
7  loop B loop
8  END
9

```