

These slides are being provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides and/or lecture recordings are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.

# Tutorial 09:

# ARM Pseudo Instructions

*Computer Science Department*

*CS2208: Introduction to Computer Organization and Architecture*

*Winter 2021-2022*

*Instructor: Mahmoud R. El-Sakka*

*Office: MC-419*

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

*Phone: 519-661-2111 x86996*

# ARM pseudo-instructions

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

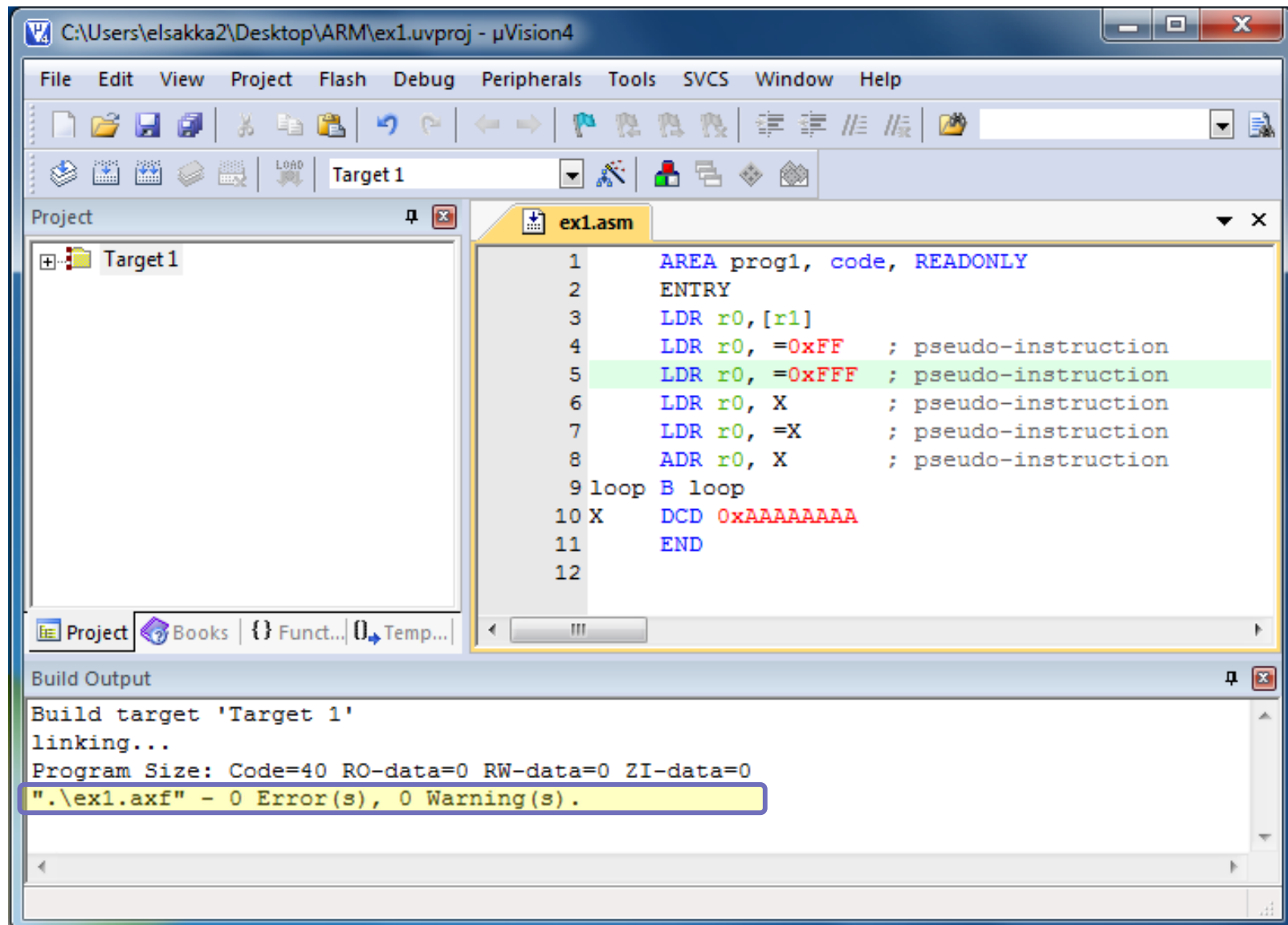
```

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

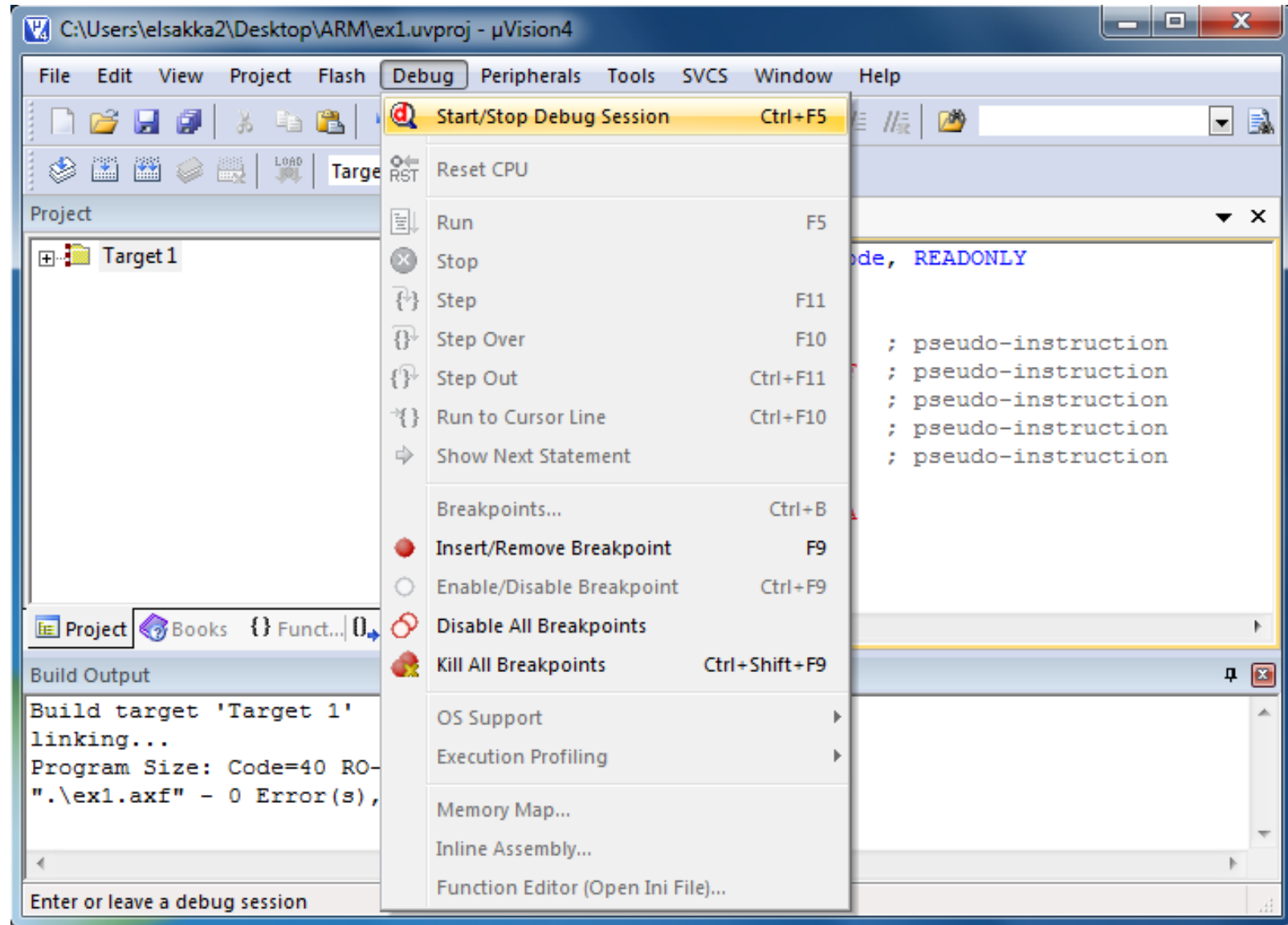
loop B loop
X    DCD 0xAAAAAAAA
END

```

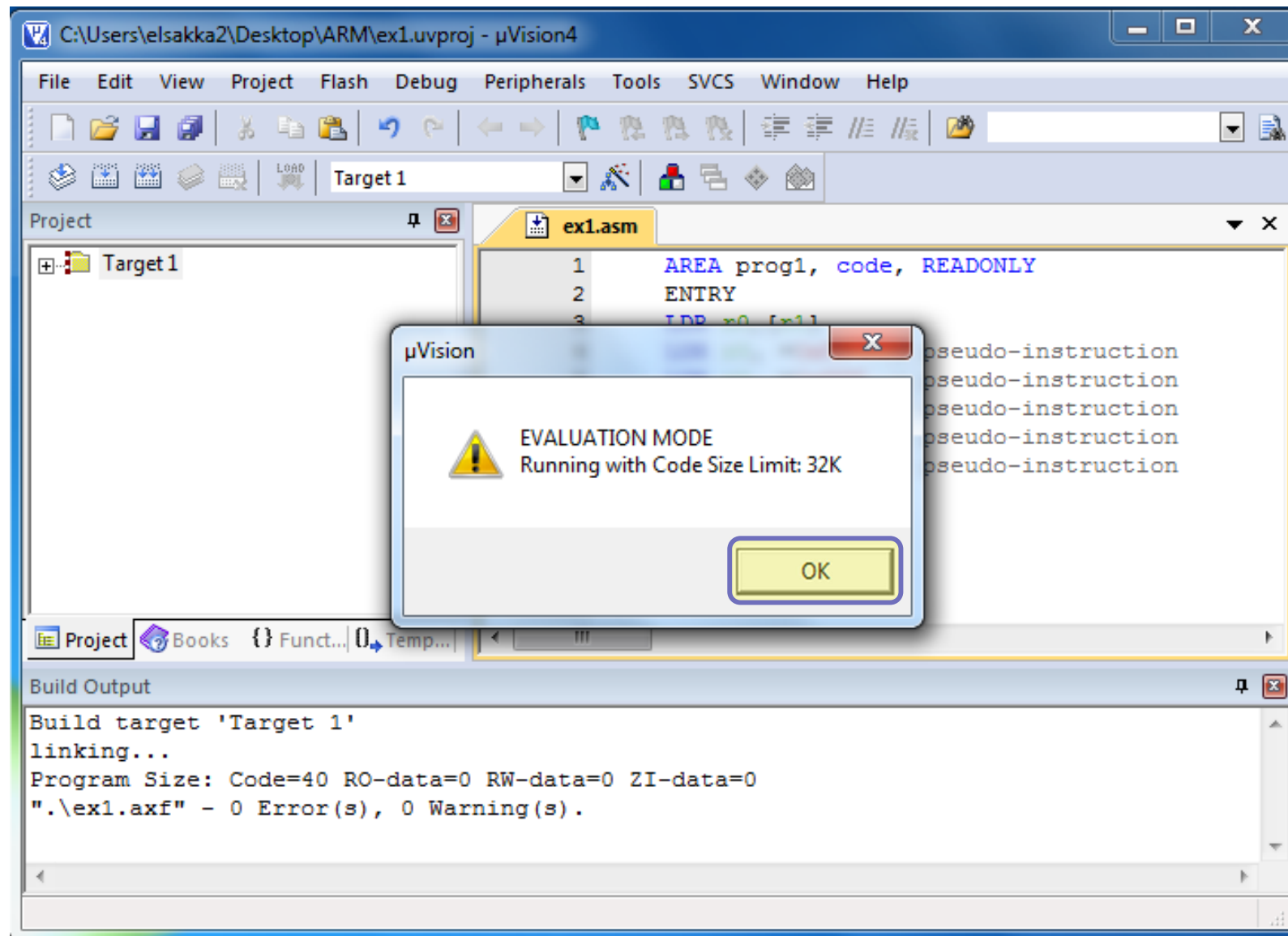
# ARM pseudo-instructions



# ARM pseudo-instructions



# ARM pseudo-instructions



# ARM pseudo-instructions

The screenshot displays the µVision4 IDE interface. The 'Registers' window on the left shows the current state of the ARM registers, with the CPSR register highlighted at 0x000000D3. The 'Disassembly' window shows a list of instructions, with the first instruction 'LDR r0, [r1]' highlighted in yellow. The 'Source' window at the bottom shows the assembly code for 'ex1.asm', with the first instruction 'LDR r0, [r1]' highlighted in green. A blue cloud callout points to the 'Step' button in the toolbar, indicating that pressing it or F11 will execute the highlighted instruction.

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

```

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 EAffffff B 0x00000018
0x0000001C AAAAAAAA BGE 0xFEAAAAACC
0x00000020 0000FFFF ???EQ
0x00000024 0000001C ANDEQ R0, R0, R12, LSL R0
0x00000028 00000000 ANDEQ R0, R0, R0
  
```

```

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 displays the uVision4 IDE interface. The 'Registers' window on the left shows the current state of ARM registers, with R0 containing the value 0xE5910000. The 'Disassembly' window in the center shows the instruction stream, with instruction 4 (LDR r0, =0xFF) highlighted in yellow. The 'Source' window at the bottom shows the assembly code for 'ex1.asm', with line 4 (LDR r0, =0xFF) also highlighted. A blue cloud bubble on the right contains the text 'Press Step, or F11'.

Register	Value
R0	0xE5910000
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

```

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 EAFFFFFE B        0x00000018
0x0000001C AAAAAAAA BGE     0xFEAAAAACC
0x00000020 0000FFFF ???EQ
0x00000024 0000001C ANDEQ   R0,R0,R12,LSL R0
0x00000028 00000000 ANDEQ   R0,R0,R0
  
```

```

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
  
```

Press Step,  
or F11



# ARM pseudo-instructions

When executing the instruction at location 0x00000008, the PC value will be 0x00000010

How is this offset calculated?

Press Step, or F11

Registers

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)	0x00000008
CPSR	0x000000D3
SPSR	0x00000000

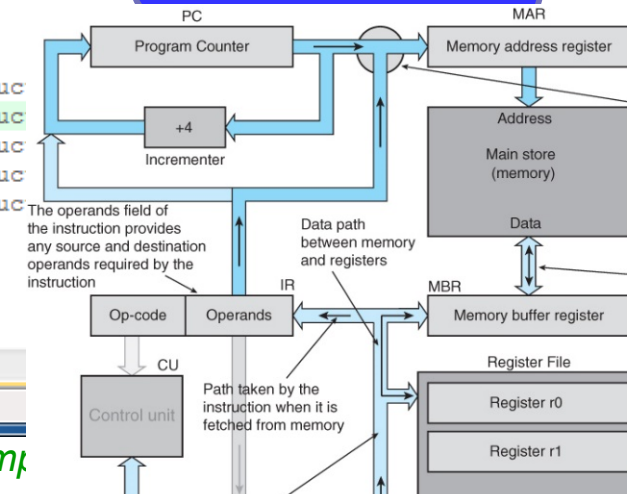
Disassembly

Address	Instruction	Comment
0x00000000	LDR r0, [r1]	
0x00000004	LDR r0, =0xFF ; pseudo-instruction	
0x00000008	LDR r0, =0xFFFF ; pseudo-instruction	
0x0000000C	LDR r0, X ; pseudo-instruction	
0x00000010	LDR r0, =X ; pseudo-instruction	
0x00000014	ADR r0, X ; pseudo-instruction	
0x00000018	loop B loop	
0x0000001C	EAFFFFFE B 0x00000018	
0x00000020	AAAAAA C 0xFEAAAA	
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 0xAAAAAA
11 END
12
  
```





# ARM pseudo-instructions

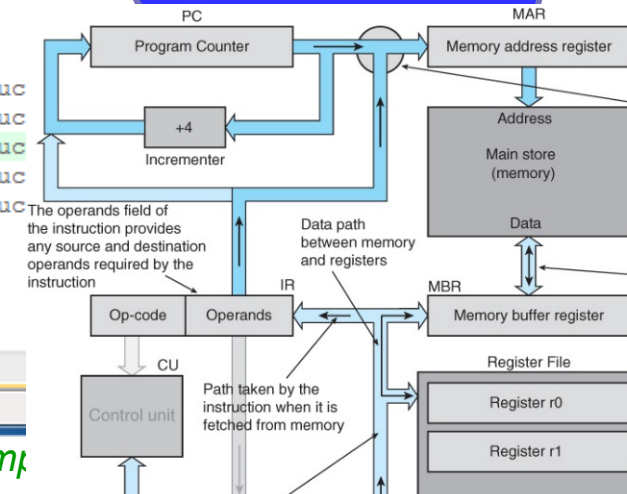
When executing the instruction at location 0x0000000C, the PC value will be 0x00000014

How is this offset calculated?

Press Step, or F11

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. R15 (PC) is highlighted with a value of 0x0000000C.
- Disassembly Window:** Shows the instruction at address 0x0000000C: `LDR r0, X` (pseudo-instruction). The next instruction at 0x00000014 is `ADD R0, PC, #0x00000000`.
- Source Code Window (ex1.asm):** Shows the assembly code with the instruction `LDR r0, X` highlighted.



# ARM pseudo-instructions

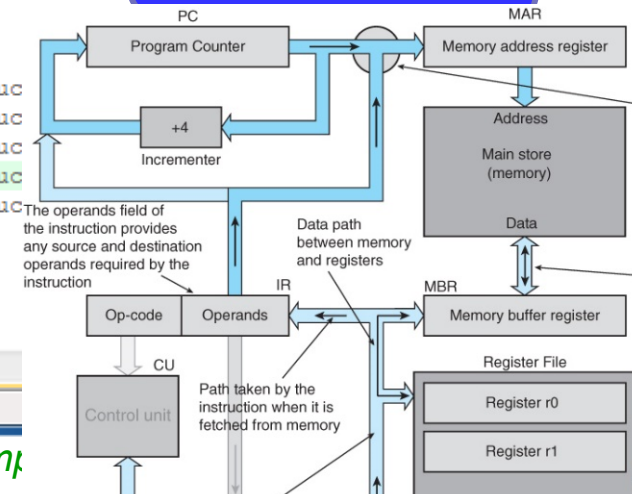
When executing the instruction at location 0x00000010, the PC value will be 0x00000018

How is this offset calculated?

Press Step, or F11

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. R15 (PC) is highlighted with a value of 0x00000010.
- Disassembly Window:** Shows the instruction stream. Instruction 7 at address 0x00000010 is highlighted: `LDR r0, =X ; pseudo-instruction`. Instruction 8 at 0x00000014 is `ADR r0, X ; pseudo-instruction`. Instruction 9 is a loop label.
- Source Window (ex1.asm):** Shows the assembly code. Instruction 7 is highlighted: `LDR r0, =X ; pseudo-instruction`.





# ARM pseudo-instructions

Same  
address  
(no change)

The screenshot displays the uVision4 IDE interface. The **Registers** window on the left shows the current state of ARM registers, with R15 (PC) highlighted at address 0x00000018. The **Disassembly** window shows the execution of instructions, with instruction 9 (loop B loop) highlighted. The **ex1.asm** window at the bottom shows the source code for the program, including pseudo-instructions like `LDR r0, =0xFF` and `LDR r0, =0xFFFF`.

Address	Instruction	Comment
0x00000000	E5910000 LDR R0, [R1]	
0x00000004	E3A000FF MOV R0, #0x000000FF	; pseudo-instruction
0x00000008	E59F0010 LDR R0, [PC, #0x0010]	
0x0000000C	E59F0008 LDR R0, [PC, #0x0008]	
0x00000010	E59F000C LDR R0, [PC, #0x000C]	
0x00000014	E28F0000 ADD R0, PC, #0x00000000	
0x00000018	EAF00000 B 0x00000018	; pseudo-instruction
0x0000001C	AAAAAA BGE 0xFEAAAA	
0x00000020	0000FFF ???EQ	
0x00000024	0000001C ANDEQ R0, R0, R12, LSL R0	
0x00000028	00000000 ANDEQ R0, R0, R0	

```

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
  
```



# 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: 0x00000000 E5910000 LDR R0,[R1]
4: 0x00000004 E3A000FF MOV R0,#0x000000FF ; pseudo-instruction
5: 0x00000008 E59F0010 LDR R0,[PC,#0x0010] ; pseudo-instruction
6: 0x0000000C E59F0008 LDR R0,[PC,#0x0008] ; pseudo-instruction
7: 0x00000010 E59F000C LDR R0,[PC,#0x000C] ; pseudo-instruction
8: 0x00000014 E28F0000 ADD R0,PC,#0x00000000 ; pseudo-instruction
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
  
```

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
12
  
```

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: 0x00000000 E5910000 LDR R0,[R1]
4: 0x00000004 E3A000FF MOV R0,#0x000000FF ; pseudo-instruction
5: 0x00000008 E59F0010 LDR R0,[PC,#0x0010] ; pseudo-instruction
6: 0x0000000C E59F000C LDR R0,[PC,#0x000C] ; pseudo-instruction
7: 0x00000010 E59F0010 LDR R0,[PC,#0x0010] ; pseudo-instruction
8: 0x00000014 E59F0008 LDR R0,[PC,#0x0008] ; pseudo-instruction
9: 0x00000018 E28F0008 ADD R0,PC,#0x00000008 ; pseudo-instruction
10: 0x0000001C EAffFFFF B 0x00000018
11: 0x00000020 0000FFFF ???EQ
12: 0x00000024 00000024 ANDEQ R0,R0,R4,LSR #32
13: 0x00000028 AAAAAAAA BGE 0xFEAAAAD4
14: 0x00000000 00000000 ANDEQ R0,R0,R0
  
```

ex1.asm

```

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
11 AREA prog1, data, READONLY
12 DCD 0xAAAAAAAA
13 END
14
  
```

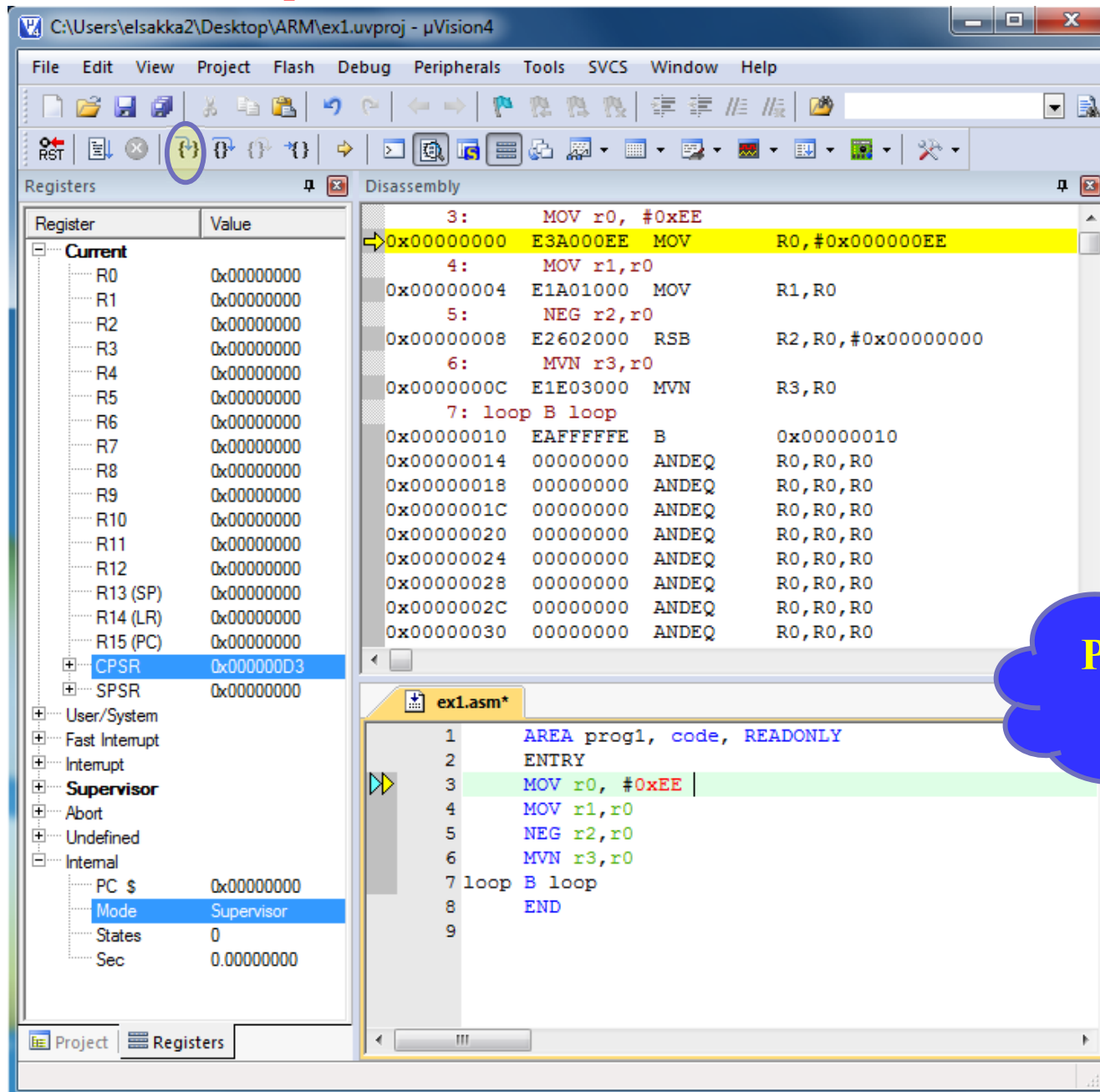


# ARM pseudo-instructions

- 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



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 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

```
- 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

```

Press Step,  
or F11

# ARM pseudo-instructions

The screenshot shows the µVision4 IDE interface. The Disassembly window displays the following instructions:

Address	Hex	Mnemonic	Operands
3:	MOV r0, #0xEE		
0x00000000	E3A000EE	MOV	R0, #0x000000EE
4:	MOV r1, r0		
0x00000004	E1A01000	MOV	R1, R0
5:	NEG r2, r0		
0x00000008	E2602000	<b>RSB</b>	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

The Registers window on the left shows the current state of the registers. The R15 (PC) register is highlighted with the value 0x00000008.

The Source window shows the assembly code for ex1.asm:

```

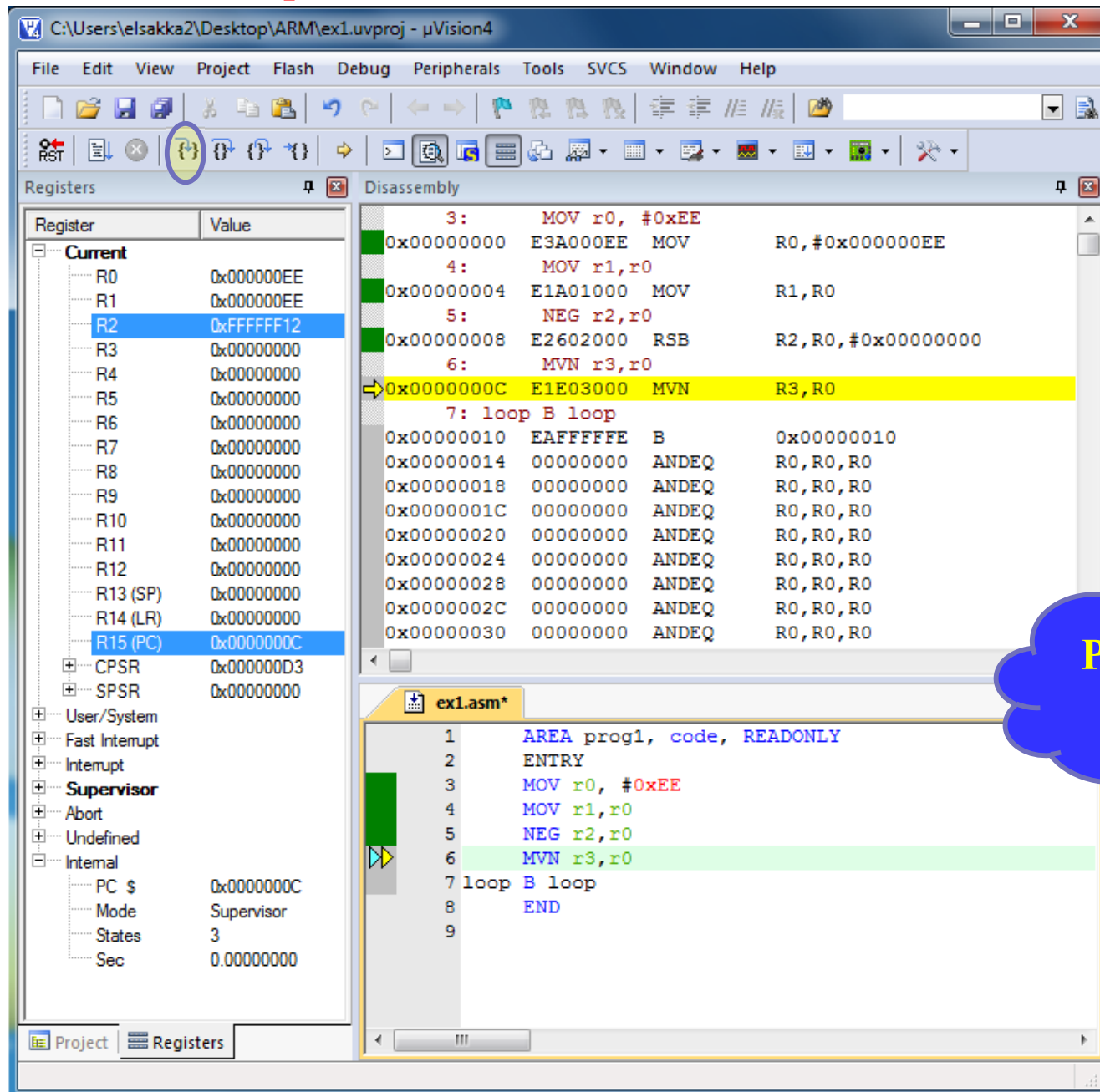
1  AREA prog1, code, READONLY
2  ENTRY
3  MOV r0, #0xEE
4  MOV r1, r0
5  NEG r2, r0
6  MVN r3, r0
7  loop B loop
8  END

```

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

Press Step,  
or F11

# ARM pseudo-instructions



Press Step,  
or F11

# ARM pseudo-instructions

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, editing, and debugging.

The **Registers** window on the left shows the current state of the ARM registers. The **Current** register set includes R0 through R15, CPSR, and SPSR. R3 is highlighted with a value of 0xFFFFFFFF. R15 (PC) is highlighted with a value of 0x00000010. The **Internal** register set shows PC \$ at 0x00000010, Mode as Supervisor, States as 4, and Sec as 0.00000000.

The **Disassembly** window on the right shows the assembly code being executed. The code is as follows:

```

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

```

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

```



# 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

Disassembly

```

3:  MOV r0, #0xFF
4:  MOV r1, r0
5:  NEG r2, r0
6:  MVN r3, r0
7:  loop B loop
8:  EAFFFFFF B 0x00000010
9:  0x00000014 00000000 ANDEQ R0, R0, R0
10: 0x00000018 00000000 ANDEQ R0, R0, R0
11:0x0000001C 00000000 ANDEQ R0, R0, R0
12:0x00000020 00000000 ANDEQ R0, R0, R0
13:0x00000024 00000000 ANDEQ R0, R0, R0
14:0x00000028 00000000 ANDEQ R0, R0, R0
15:0x0000002C 00000000 ANDEQ R0, R0, R0
16:0x00000030 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	0xFFFFFFFF12
R1	0xFFFFFFFF12
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

Disassembly

```

3:  MOVN r0, #-0xEE
4:  MOV r1, r0
5:  NEG r2, r0
6:  MVN r3, r0
7:  loop B loop
8:  EAFFFFFF B 0x00000010
9:  0x00000014 00000000 ANDEQ R0, R0, R0
10:0x00000018 00000000 ANDEQ R0, R0, R0
11:0x0000001C 00000000 ANDEQ R0, R0, R0
12:0x00000020 00000000 ANDEQ R0, R0, R0
13:0x00000024 00000000 ANDEQ R0, R0, R0
14:0x00000028 00000000 ANDEQ R0, R0, R0
15:0x0000002C 00000000 ANDEQ R0, R0, R0
16:0x00000030 00000000 ANDEQ R0, R0, R0

```

ex1.asm

```

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

```