

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

Fall 2022-2023

Instructor: Mahmoud R. El-Sakka

Office: MC-419

Email: 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]
```

copy 4 bytes in address pointed by r1

```
LDR r0, =0xFF ; pseudo-instruction
```

```
LDR r0, =0xFFF ; pseudo-instruction => literal pool
```

```
LDR r0, X ; pseudo-instruction } Add/Sub
```

```
LDR r0, =X ; pseudo-instruction } literal pool.
```

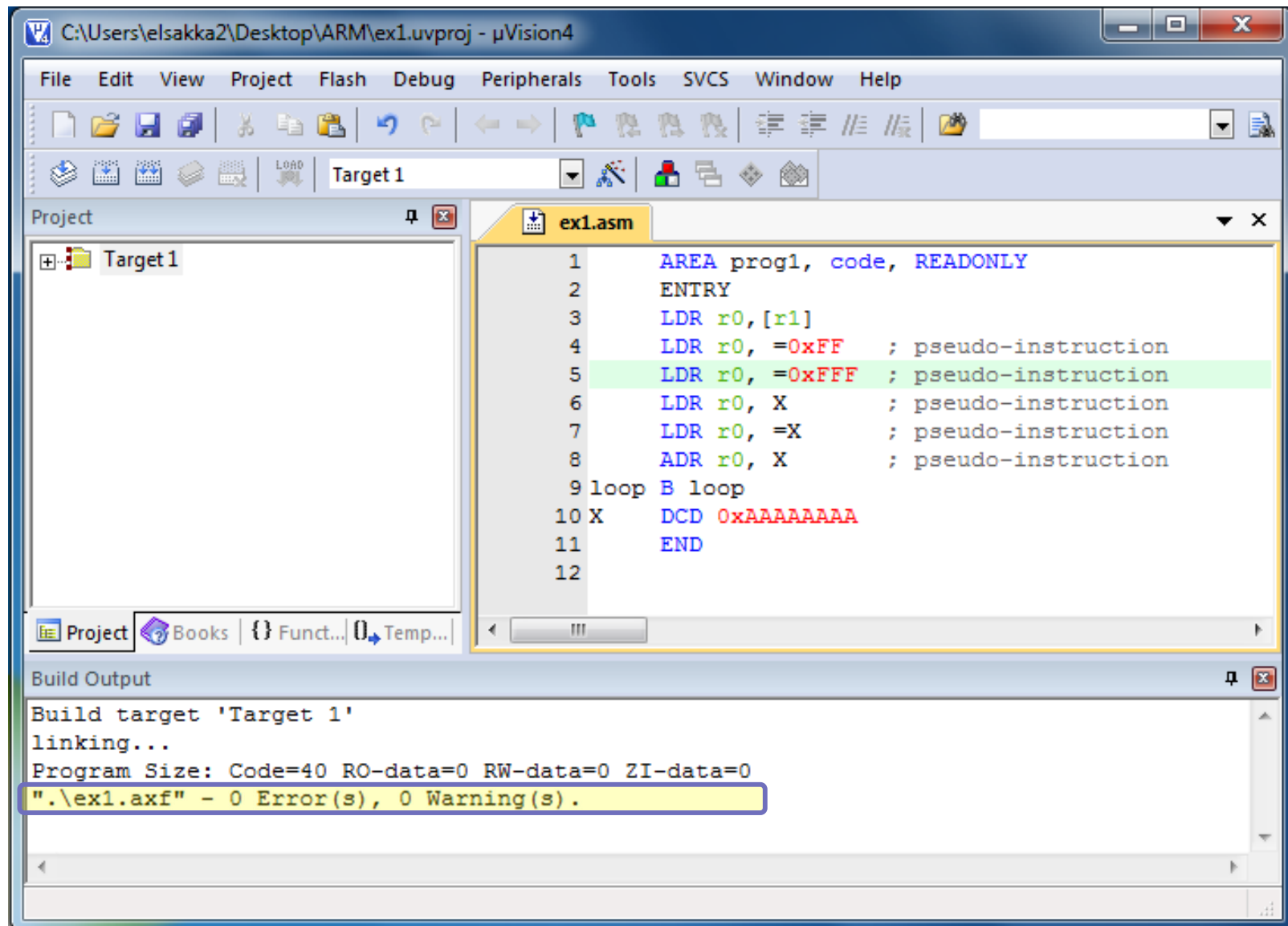
```
ADR r0, X ; pseudo-instruction
```

```
loop B loop
```

```
X DCD 0xAAAAAAAA
```

```
END
```

ARM pseudo-instructions



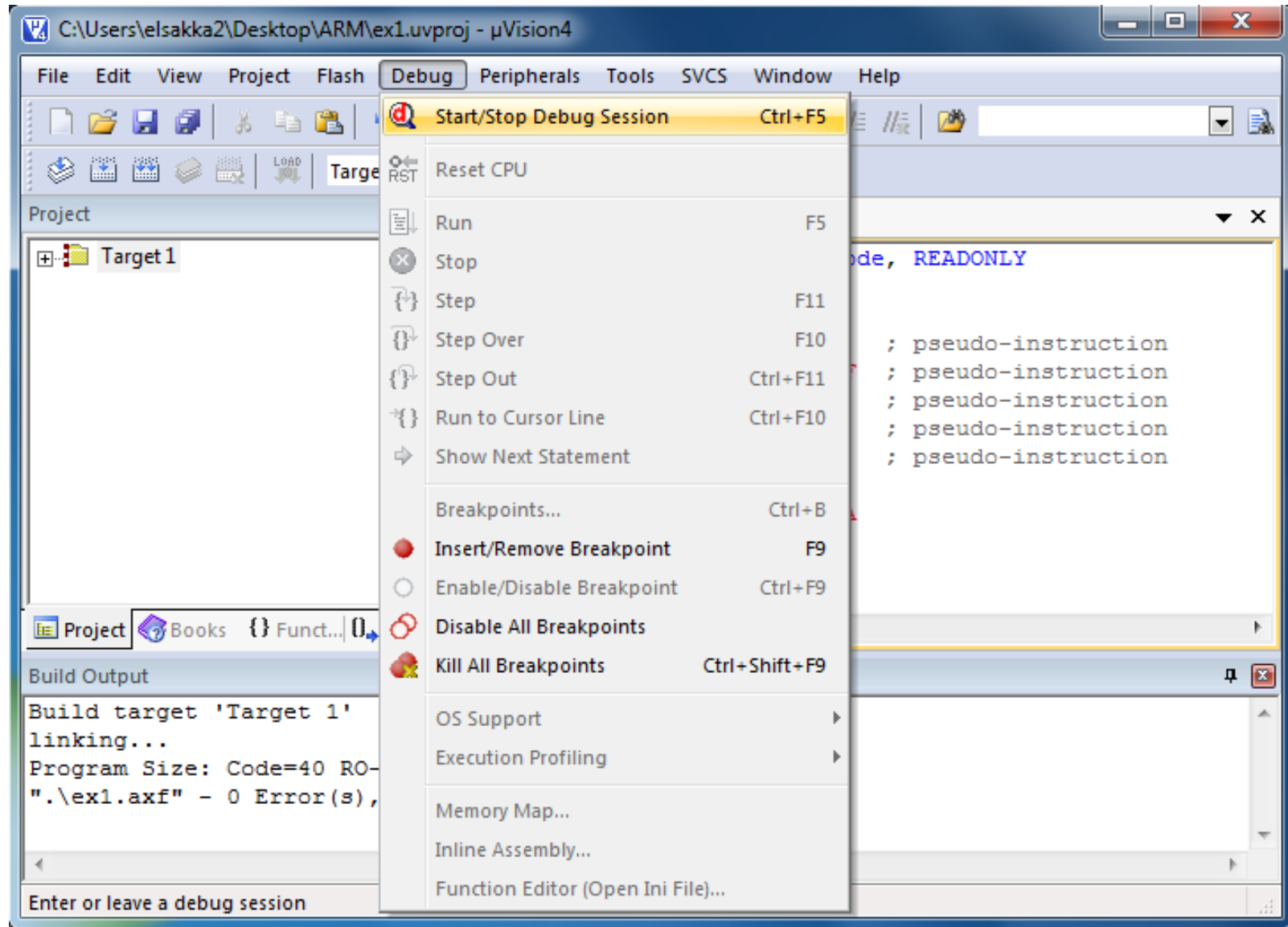
The screenshot shows the µVision4 IDE interface. The main window displays the assembly file `ex1.asm` with the following code:

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

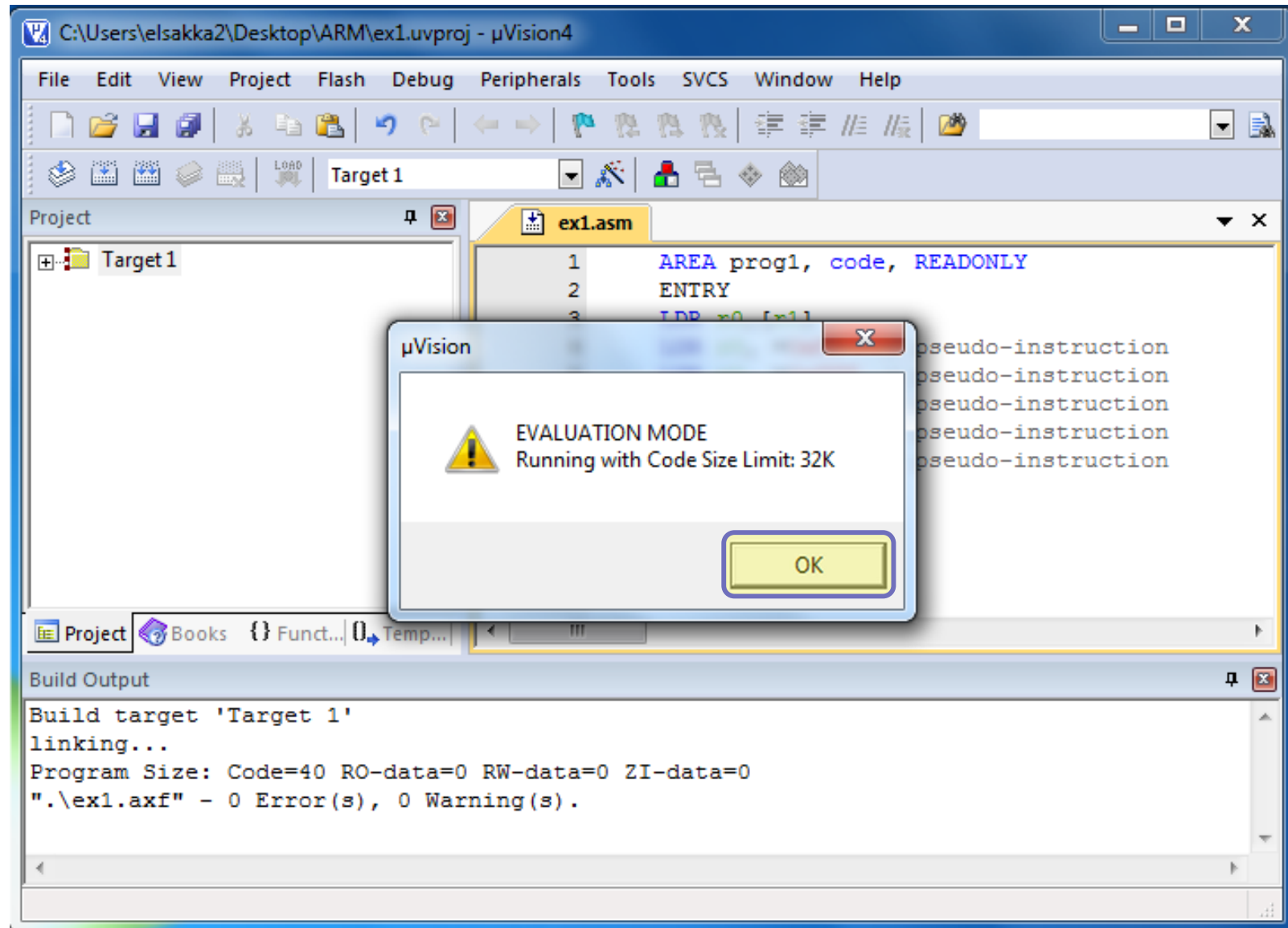
The `Build Output` window at the bottom shows the following text:

```
Build target 'Target 1'
linking...
Program Size: Code=40 RO-data=0 RW-data=0 ZI-data=0
".\ex1.axf" - 0 Error(s), 0 Warning(s).
```

ARM pseudo-instructions



ARM pseudo-instructions



ARM pseudo-instructions

The screenshot displays the uVision4 IDE interface. The **Disassembly** window shows the following instructions:

| Address | Hex | Mnemonic | Comment |
|------------|----------------|----------|----------------------|
| 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 | AAAAAAA | BGE | 0xFEAAAACC |
| 0x00000020 | 0000FFF | ??? | EQ |
| 0x00000024 | 0000001C | ANDEQ | R0, R0, R12, LSL R0 |
| 0x00000028 | 00000000 | ANDEQ | R0, R0, R0 |

The **Registers** window shows the current state of registers R0-R15, CPSR, and SPSR. The **Source** window shows the assembly code for '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 0xAAAAAAA
11 END
12

```

Press Step,
or F11

ARM pseudo-instructions

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

| 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 |
- 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     0xFEAAAAACC
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
12     END

```

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?

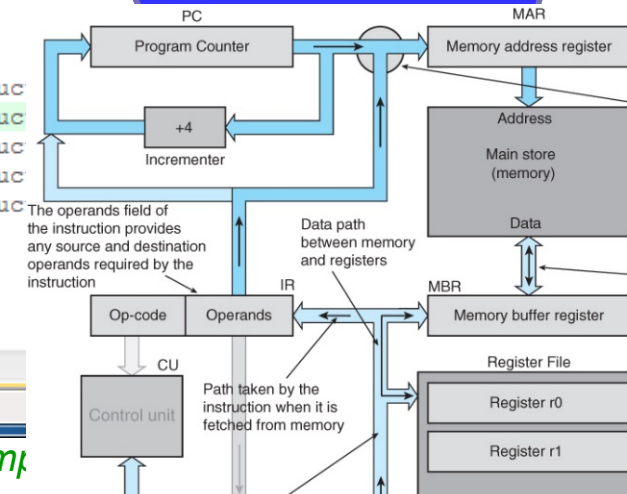
Press Step, or F11

The screenshot shows the uVision4 IDE with the Disassembly window open. The PC register (R15) is highlighted with a value of 0x00000008. The Disassembly window shows instructions at memory locations 0x00000000 to 0x00000028. Instruction 5 at 0x00000008 is 'LDR r0, =0xFFFF', which is a pseudo-instruction. Instruction 6 at 0x0000000C is 'LDR r0, X', also a pseudo-instruction. The source file 'ex1.asm' shows the assembly code corresponding to the disassembly.

| Address | Disassembly | Comment |
|------------|------------------------------------|----------------------|
| 0x00000000 | E5910000 LDR R0, [R1] | |
| 0x00000004 | E3A000FF MOV R0, #0x000000FF | ; pseudo-instruction |
| 0x00000008 | E59F0010 LDR R0, [PC, #0x0010] | ; pseudo-instruction |
| 0x0000000C | E59F0008 LDR R0, [PC, #0x0008] | ; pseudo-instruction |
| 0x00000010 | E59F000C LDR R0, [PC, #0x000C] | ; pseudo-instruction |
| 0x00000014 | E28F0000 ADD R0, PC, #0x00000000 | ; pseudo-instruction |
| 0x00000018 | EAFFFFFE B 0x00000018 | |
| 0x0000001C | AAAAAAAB BGE 0xFEAAACC | |
| 0x00000020 | 000000FF ???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
  
```



ARM pseudo-instructions

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

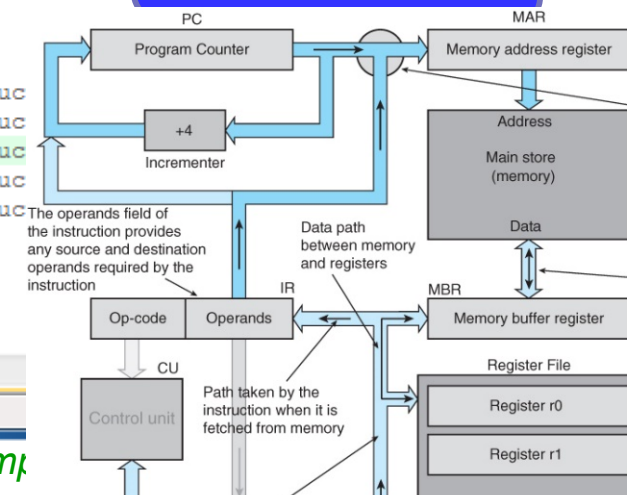
1C - C - 8

How is this offset calculated?

Press Step, or F11

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the PC register (R15) at address 0x0000000C.
- Disassembly Window:** Shows the instruction at address 0x0000000C: `LDR r0, [PC, #0x0008]`. This is a pseudo-instruction that will be replaced by `LDR r0, X` at address 0x00000014.
- Source Code Window (ex1.asm):** Shows the assembly code with the instruction `LDR r0, X` highlighted at line 6.



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:

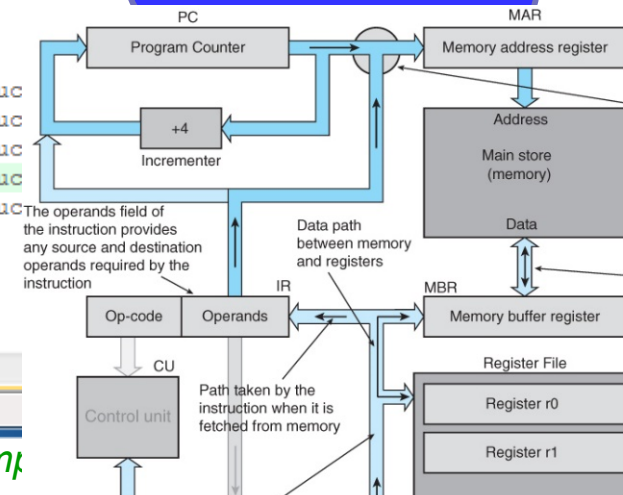
- Registers Window:** Shows R15 (PC) at 0x00000010.
- Disassembly Window:**

| 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] | ; pseudo-instruction |
| 0x00000014 | E28F0000 ADD R0, PC, #0x00000000 | ; pseudo-instruction |
| 0x00000018 | EAFFFFFE B 0x00000018 | |
| 0x0000001C | AAAAAAABGE 0xFEAAACC | |
| 0x00000020 | 0000FFFF ???EQ | |
| 0x00000024 | 0000001CANDEQ R0, R0, R12, LSL R0 | |
| 0x00000028 | 00000000ANDEQ 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

```



ARM pseudo-instructions

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

When
 executing the
 instruction
 at location
 0x00000014,
 the PC value
 will be
 0x0000001C

How is this
 offset
 calculated?

Press Step,
 or F11

Registers window:

| 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) | 0x00000014 |
| CPSR | 0x000000D3 |
| SPSR | 0x00000000 |

Disassembly window:

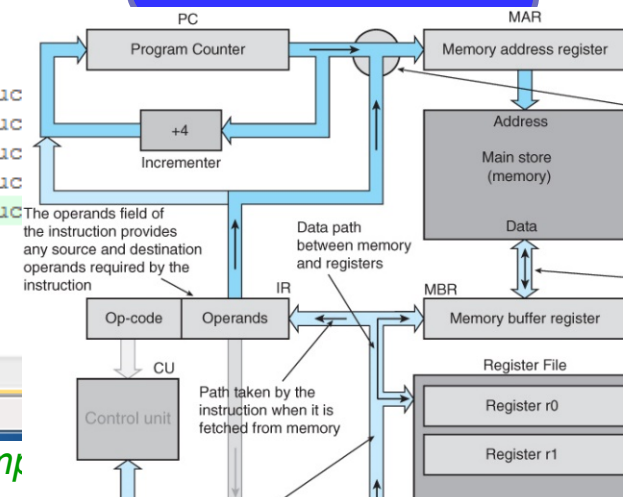
```

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

Source code 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
  
```



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. The **Disassembly** window in the center shows the instruction stream, with instruction 9 highlighted. The **ex1.asm** window at the bottom shows the source code for the program.

| 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 |

| 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 | EAFFFFFE B 0x00000018 | |
| 0x0000001C | AAAAAA BGE 0xFEAAAACC | |
| 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, =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 window (Left):

| 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 window (Right):

```

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: EAFEEEE B 0x00000018
11: AAAAAA BGE 0xFEAAACC
12: 0000FFF ???EQ
13: 000001C ANDEQ R0,R0,R12,LSL R0
14: 0000000 ANDEQ R0,R0,R0

```

Source window (Bottom):

```

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 0xAAAAAAA
11 END
12

```

Registers window (Left):

| 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 window (Right):

```

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: EAFEEEE B 0x00000018
11: 0000FFF ???EQ
12: 0000024 ANDEQ R0,R0,R4,LSR #32
13: AAAAAA BGE 0xFEAAAD4
14: 0000000 ANDEQ R0,R0,R0

```

Source window (Bottom):

```

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 0xAAAAAAA
13 END
14

```