

Tutorial 06: ARM Shift Instructions and ARM Addressing Modes

Computer Science Department

CS2208b: Introduction to Computer Organization and Architecture

Winter 2019

Instructor: Mahmoud R. El-Sakka

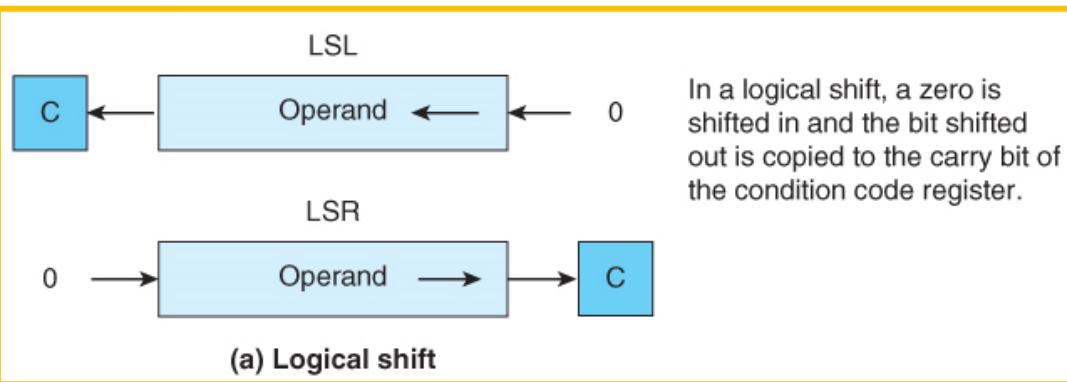
Office: MC-419

Email: elsakka@csd.uwo.ca

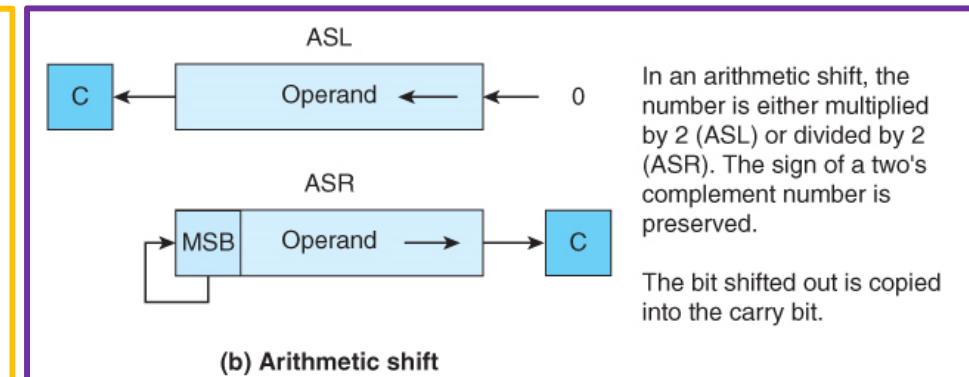
Phone: 519-661-2111 x86996

ARM's Data-Processing Instructions (Shift Operations)

- **Shift** operations move bits one or more places *left* or *right*.
 - **Logical shifts**
 - *insert a 0* in the vacated position.
 - **Arithmetic shifts**
 - *replicate the sign-bit* during a right shift
 - **Circular shifts**
 - *the bit shifted out of one end is shifted in the other end*
i.e., the register is treated as a ring
 - **Circular shifts through carry**
 - *included the carry bit in the shift path*



In a logical shift, a zero is shifted in and the bit shifted out is copied to the carry bit of the condition code register.



In an arithmetic shift, the number is either multiplied by 2 (ASL) or divided by 2 (ASR). The sign of a two's complement number is preserved.

The bit shifted out is copied into the carry bit.

(a) Logical shift

(b) Arithmetic shift

ROL

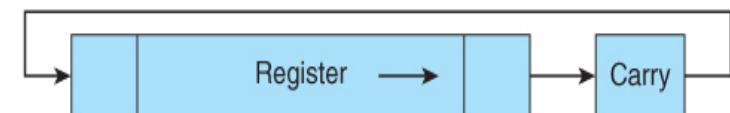
ROR

RRN

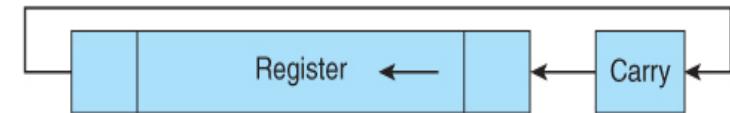
(c) Rotate

In a rotate operation, the bit shifted out is copied into the bit vacated at the other end (i.e., no bit is lost during a rotate). The bit shifted out is also copied into the carry bit.

Rotate right through carry



Rotate left through carry



ARM's Data-Processing Instructions (Shift Operations)

- ❑ ARM support both *static* and *dynamic* shifts (except *rotate through carry* instruction which allows *only one single shift* per instruction)
 - In *static shift*, the number of shift places
 - is determined *when the code is written*
 - can only have the following values, inclusive:
 - **LSL**: allowable values are from #0 to #31 (*32 different values*)
 - **LSR**: allowable values are from #1 to #32 (*32 different values*)
 - **ASR**: allowable values are from #1 to #32 (*32 different values*)
 - **ROR**: allowable values are from #1 to #31 (*31 different values*)
 - *The remaining value is used to encode RRX*
 - **ROR + a shift of #0 → RRX**
 - In *dynamic shift*, the number of shift places
 - is determined *when the code is executed, i.e., at run time*
 - If the number of dynamic shifts is ≥ 32 , zero will be stored in the destination

Only 5 bits are needed to encode the amount of shifts.

In case of **LSR** and **ASR**, the value #32 is encoded as 00000

ARM's Data-Processing Instructions (Shift Operations)

- ARM implements only the following five shifts
 - **LSL** logical shift left
 - **LSR** logical shift right
 - **ASR** arithmetic shift right
 - **ROR** rotate right
 - **RRX** rotate right through carry (one shift)
- *Other shift operations have to be synthesized by the programmer.*
 - An *arithmetic shift left* is effectively the same as a *logical shift left*
 - For a 32-bit value, an *n-bit rotate shift left* is identical to a *32 – n rotate shift right*
 - *Rotate left through carry* can be implemented by means of ADCS **r0,r0,r0 ; add r0 to r0 with carry and set the flags**
 - The instruction means $r0 + r0 + C$, i.e., $2 \times r0 + C$, i.e.,
 - shifting left the content of r0
 - store the value of C in the vacant bit to the left, and
 - storing the shifted out bit in the carry flag

ARM's Data-Processing Instructions (Shift Operations)

- ARM has no explicit shift operations!!.
- ARM combines shifting with other data processing operations, where
 - the second operand in the arithmetic operation (i.e., the third parameter in the assembly arithmetic instruction) is allowed to be shifted before it is used.
 - For example,
ADD r0, r1, r2, LSL #1 ; [r0] \leftarrow [r1] + [r2] \times 2
 - logically shift left the contents of r2,
 - add the result to the contents of r1, and
 - put the results in r0
- ARM also combines shifting with moving operations
 - This way, a shift operation can be performed as a stand alone operation.
 - For example,
MOV r3, r3, LSL #1 ; [r3] \leftarrow [r3] \times 2
 - ARM provides pseudo shift instructions, which are translated to MOV instructions.
LSL r3, r3, #1 ; will be converted to MOV r3, r3, LSL #1

ARM's Data-Processing Instructions (Shift Operations)

```
AREA prog1, code, READONLY
ENTRY
MOV r3,#2
LDR r1, =0xFFFFFFFF ;in binary 1100 1100 1100 1100 1100 1100 1100 1100
LSLS r1,r1,#5
LSLS r1,r1,r3

LSRS r1,r1,#10
LSRS r1,r1,r3

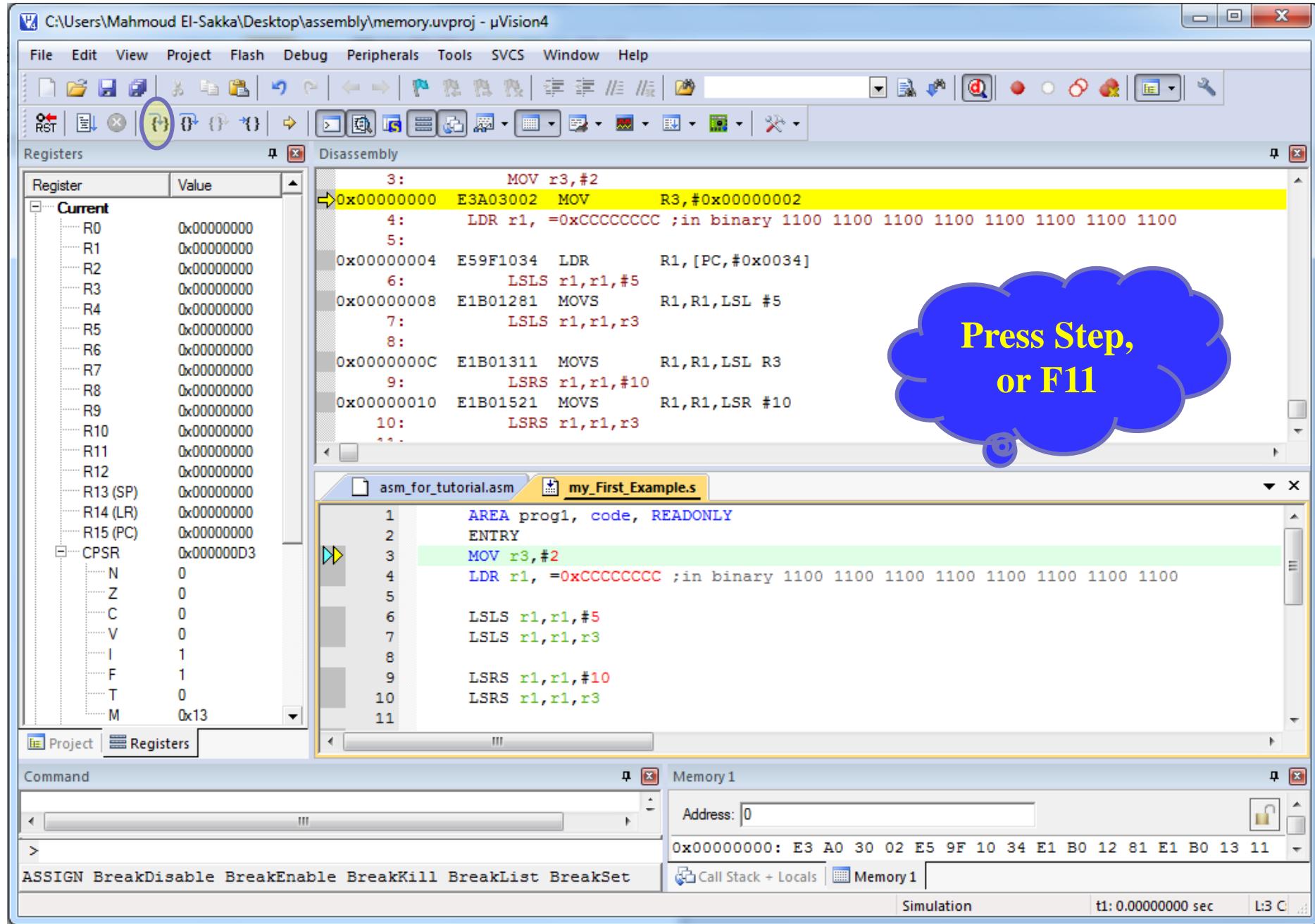
ASRS r1,r1,#2
LSLS r1,r1,#15
ASRS r1,r1,#16

ASRS r1,r1,r3

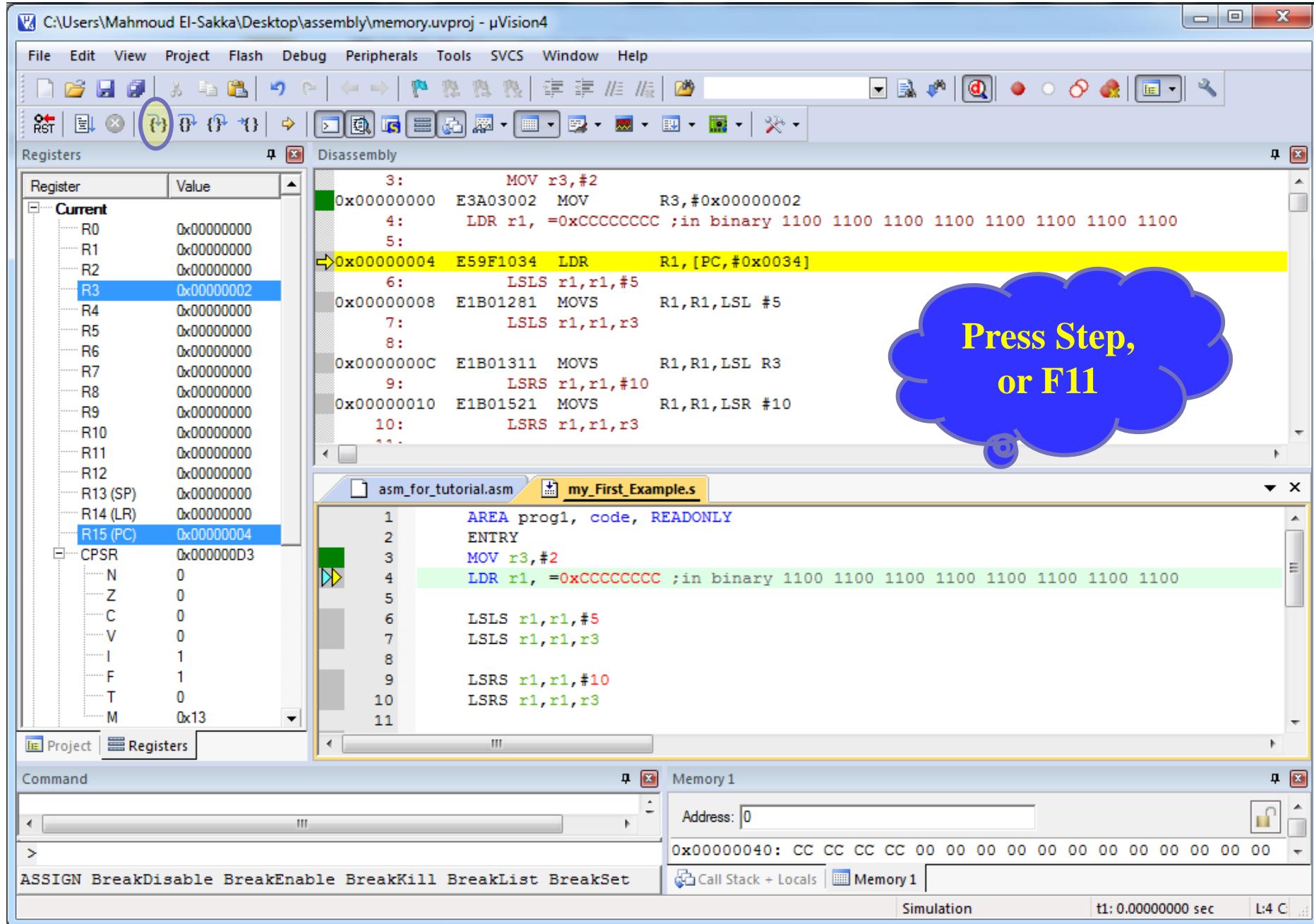
RORS r1,r1,#4
RORS r1,r1,r3

RRXS r1,r1
RRXS r1,r1
RRXS r1,r1
RRXS r1,r1
END
```

ARM's Data-Processing Instructions (Shift Operations)

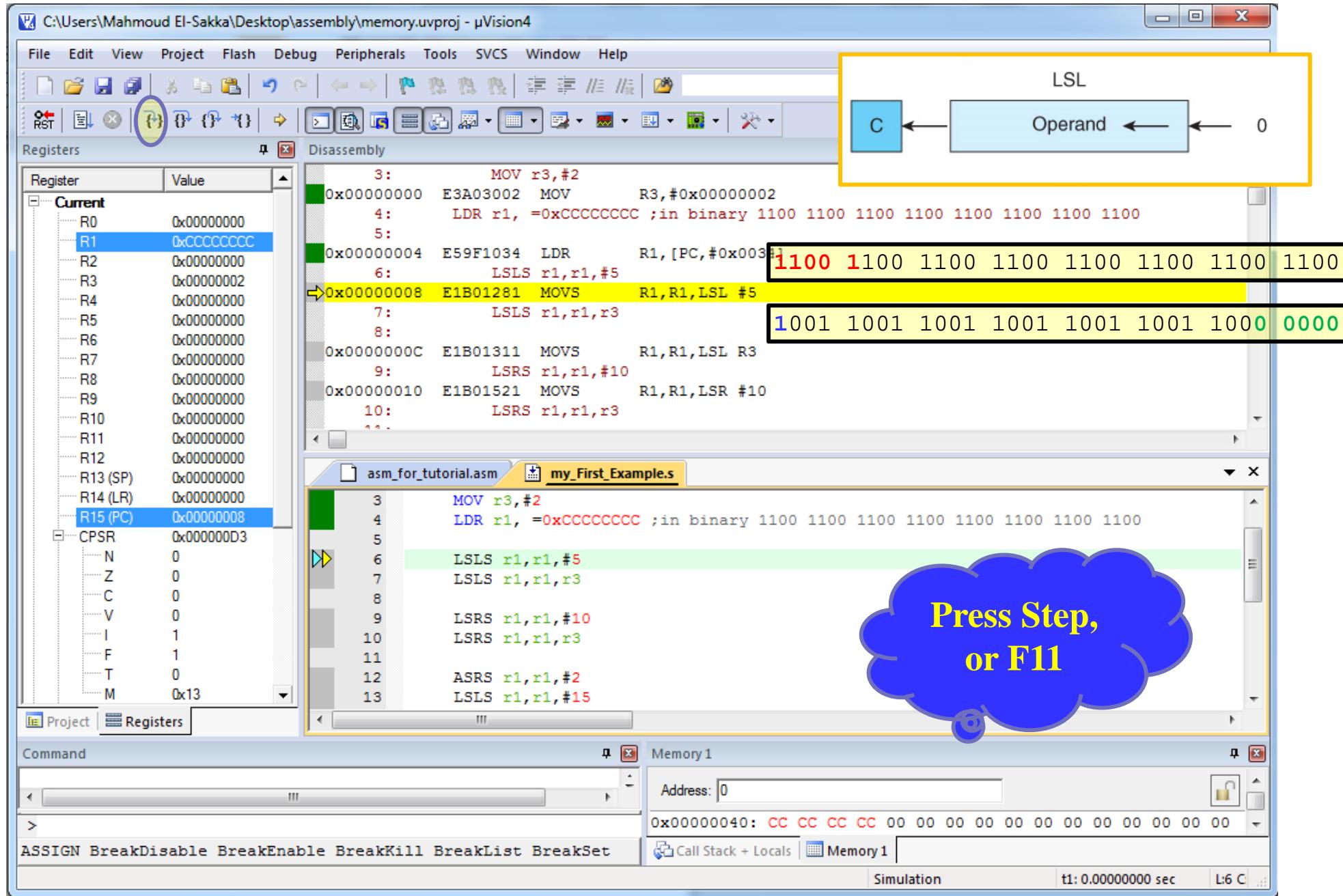


ARM's Data-Processing Instructions (Shift Operations)



Press Step, or F11

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

- Registers:** Shows the current values of various registers. R1 is highlighted with a red arrow pointing to the Disassembly window.
- Disassembly:** Displays assembly code with binary instruction details. A yellow box highlights the LSL (Logical Shift Left) operation at address 0x00000004. The diagram to the right illustrates the LSL operation: $C \leftarrow \text{Operand} \ll 0$.
- Memory Dump:** Shows the memory starting at address 0x00000040. The value at address 0 is highlighted with a red circle labeled '1'.
- Assembly Editor:** Shows the source code for `asm_for_tutorial.asm` with several instructions highlighted in green.
- Command Window:** Contains the command `ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet`.
- Memory View:** Shows the memory dump with the address 0 and the value `0x00000040: CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00`.

ARM's Data-Processing Instructions (Shift Operations)

```

    graph LR
        C[ ] --> Operand[ ]
        Operand --> Z[0]
    
```

The diagram shows the logical shift left (LSL) instruction format. It consists of three components: the Condition register (C), the Operand, and the shift amount (0). The C register is highlighted with a yellow box.

Registers:

Register	Value
R0	0x00000000
R1	0x99999980
R2	0x00000000
R3	0x00000002
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	0xA00000D3
N	1
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13

Disassembly:

```

3:      MOV r3,#2
0x00000000 E3A03002 MOV     R3,#0x00000002
4:      LDR r1, =0xCCCCCCCC ;in binary 1100 1100 1100 1100 1100 1100 1100 1100
5:
0x00000004 E59F1034 LDR     R1,[PC,#0x0034]
6:      LSLS r1,r1,#5
0x00000008 E1B01281 MOVS   R1,R1,LSL #5
7:      LSLS r1,r1,r3
8:
0x0000000C E1B01311 MOVS   R1,R1,LSL R3
9:      LSRS r1,r1,#10
0x00000010 E1B01521 MOVS   R1,R1,LSR #10
10:     LSRS r1,r1,r3
11:
12:
13:
14:

```

Binary Representation:

```

1001 1001 1001 1001 1001 1001 1001 1000 0000
0110 0110 0110 0110 0110 0110 0110 0000 0000

```

Assembly Editor:

```

asm_forTutorial.asm my_First_Example.s
4:      LDR r1, =0xCCCCCCCC ;in binary 1100 1100 1100 1100 1100 1100 1100 1100
5:
6:
7:      LSLS r1,r1,#5
8:      LSLS r1,r1,r3
9:
10:     LSRS r1,r1,#10
11:     LSRS r1,r1,r3
12:     ASRS r1,r1,#2
13:     LSLS r1,r1,#15
14:     ASRS r1,r1,#16

```

Memory View:

Address: 0

```

0x00000040: CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Call Stack + Locals | Memory 1

Simulation t1: 0.0000000 sec L:7 C:

Cloud Callout:

Press Step,
or F11

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

- Registers:** Shows the current values of various registers. R1 is highlighted with a red arrow pointing to the Disassembly window.
- Disassembly:** Displays ARM assembly code. Instruction 9: LSRS r1,r1,#10 is highlighted with a yellow box. A callout diagram to the right shows the LSLS (Logical Shift Left) operation: $C \leftarrow \text{Operand} \leftarrow 0$. The value 0 is circled in yellow and points to the green line in the assembly code.
- Memory Dump:** Shows the memory dump at address 0x00000040. The first four bytes are CC CC CC CC (hex 00000040: CC CC CC CC). A blue line connects the value 0 in the assembly code to the first byte of the memory dump.
- Assembly Editor:** Shows the assembly code for "my_First_Example.s". It includes instructions like LSLS, LSRS, ASRS, and ASRS.
- Command Line:** Shows the command ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet.
- Memory View:** Shows the memory view with address 0 and data 0x00000040: CC CC CC CC.

ARM's Data-Processing Instructions (Shift Operations)

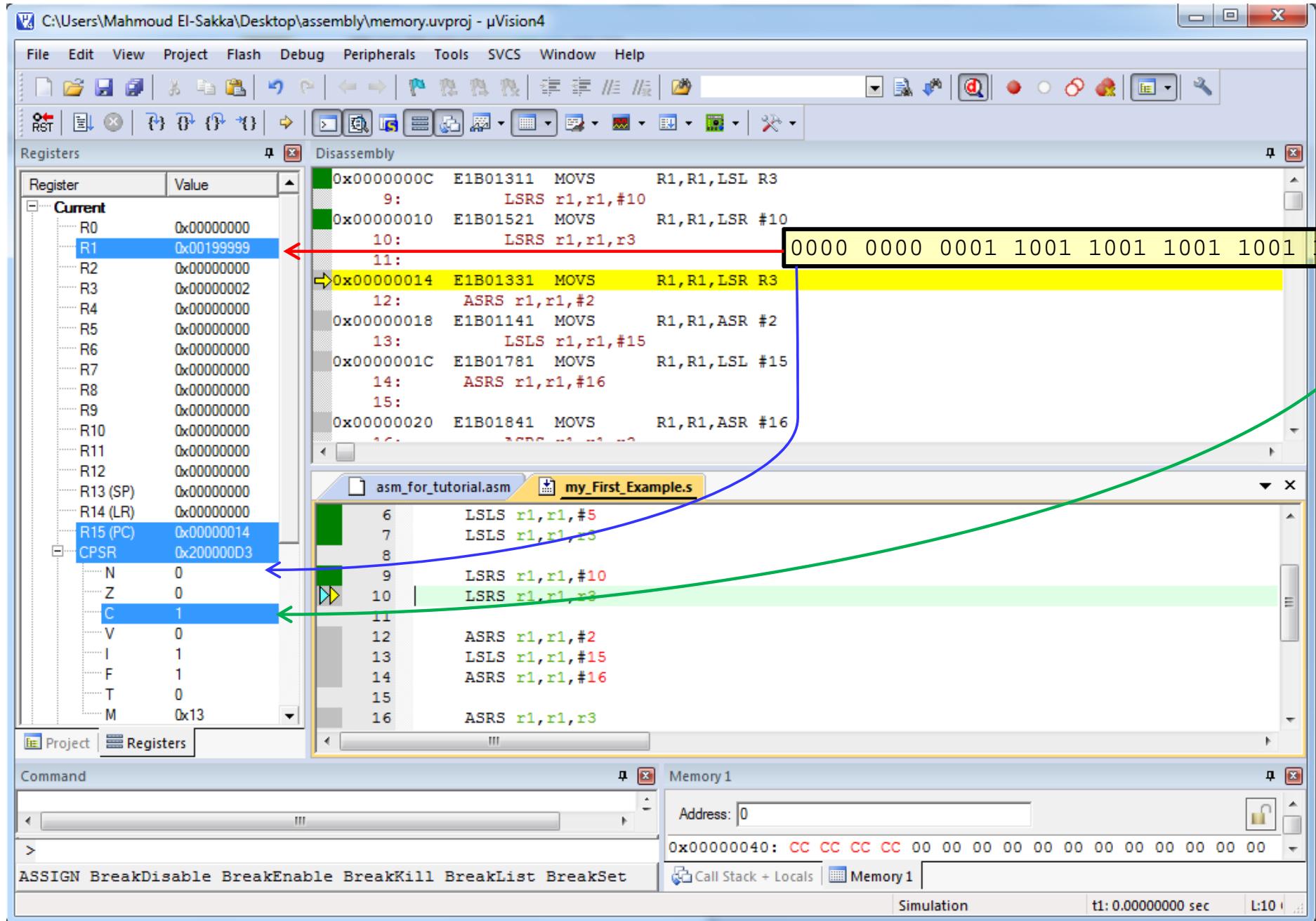
The diagram illustrates the logical shift right (LSR) operation. An input value of 0 is shifted right by the operand (the number of bits to shift). The result is C.

Screenshot of µVision4 IDE:

- Registers:** Shows the current register values. R1 is highlighted with a yellow circle.
- Disassembly:** Shows assembly code with memory addresses and opcodes. Instruction 9: LSRS r1,r1,#10 is highlighted with a yellow box. The result of this instruction is shown in the Registers window.
- Registers Window:** Shows the state of registers after the instruction. R1 contains 0000 0000 0001 1001 1001 1001 1001.
- Code Editor:** Shows the assembly source code for the tutorial.
- Memory View:** Shows memory starting at address 0x00000040 containing the value CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00.

Cloud Callout: A blue cloud-shaped callout with the text "Press Step, or F11" inside it, pointing towards the Disassembly window.

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

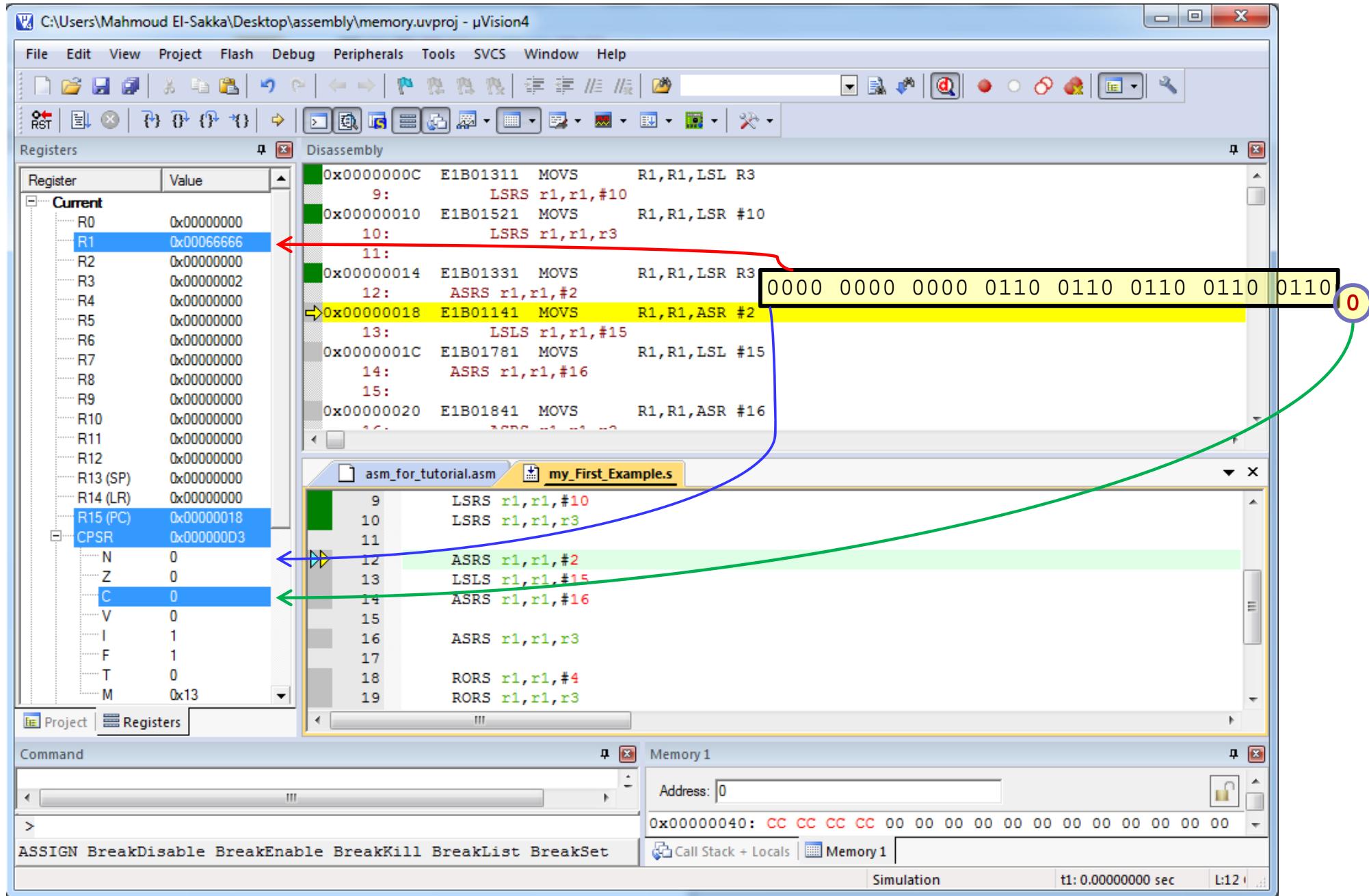
- Title Bar:** C:\Users\Mahmoud El-Sakka\Desktop\assembly\memory.uvproj - µVision4
- Menu Bar:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help
- Toolbars:** Standard, Project, Debug, Memory, Registers, Disassembly, Registers, Stack, Call Stack + Locals, Memory, Simulation.
- Registers Window:** Shows the current register values. R1 is highlighted with a yellow circle.
- Disassembly Window:** Displays assembly code with addresses, opcodes, and comments. A yellow box highlights the LSR instruction at address 0x00000014. The assembly code includes:
 - 0x0000000C E1B01311 MOVS R1,R1,LSL R3
 - 9: LSRS r1,r1,#10
 - 0x00000010 E1B01521 MOVS R1,R1,LSR #10
 - 10: LSRS r1,r1,r3
 - 11:
 - 0x00000014 E1B01331 MOVS R1,R1,LSR R3
 - 12: ASRS r1,r1,#2
 - 0x00000018 E1B01141 MOVS R1,R1,ASR #2
 - 13: LSLS r1,r1,#15
 - 0x0000001C E1B01781 MOVS R1,R1,LSL #15
 - 14: ASRS r1,r1,#16
 - 15:
 - 0x00000020 E1B01841 MOVS R1,R1,ASR #16
 - 16: LSRS r1,r1,r3
- Memory Window:** Shows memory starting at address 0x00000040 with data CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00.
- Cloud Callout:** A blue cloud contains the text "Press Step, or F11".
- Code Editor:** Shows the assembly code for the current file "asm_for_tutorial.asm". Line 10 is highlighted.

A detailed diagram of the LSR (Logical Shift Right) operation is shown in the top right corner:

```

    graph LR
        0[0] --> Operand[Operand]
        Operand --> C[C]
        style Operand fill:#d9e1f2,stroke:#333,stroke-width:1px
        style C fill:#d9e1f2,stroke:#333,stroke-width:1px
    
```

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

```

    graph LR
        MSB[MSB] --> FB(( ))
        FB --> O[Operand]
        O --> C[C]
    
```

The screenshot shows the µVision4 IDE interface during assembly code execution. The Registers window highlights R1 (Value: 0x00066666). The Disassembly window shows the instruction at address 0x00000018: E1B01141 MOVS R1,R1,ASR #2. The assembly editor shows the source code for this instruction. The Memory window displays the value 0000 0000 0000 0110 0110 0110 0110 0110 at memory location 0x00000040. A callout bubble says "Press Step, or F11".

Register	Value
Current	
R0	0x00000000
R1	0x00066666
R2	0x00000000
R3	0x00000002
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
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13

Disassembly:

```

0x0000000C E1B01311 MOVS R1,R1,LSL R3
  9:           LSRS r1,r1,#10
0x00000010 E1B01521 MOVS R1,R1,LSR #10
  10:          LSRS r1,r1,r3
  11:
0x00000014 E1B01331 MOVS R1,R1,LSR R3
  12:          ASRS r1,r1,#2
→ 0x00000018 E1B01141 MOVS R1,R1,ASR #2
  13:          LSLS r1,r1,#15
0x0000001C E1B01781 MOVS R1,R1,LSL #15
  14:          ASRS r1,r1,#16
  15:
0x00000020 E1B01841 MOVS R1,R1,ASR #16
  16:          ASRS r1,r1,#16
  17:
  18:
  19:

```

asm_for_tutorial.asm my_First_Example.s

```

  9:           LSRS r1,r1,#10
 10:          LSRS r1,r1,r3
 11:
 12:          ASRS r1,r1,#2
 13:          LSLS r1,r1,#15
 14:          ASRS r1,r1,#16
 15:
 16:          ASRS r1,r1,r3
 17:
 18:          RORS r1,r1,#4
 19:          RORS r1,r1,r3

```

Memory 1:

Address: 0	0x00000040: CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00
------------	---

Call Stack + Locals | Memory 1

Simulation: t1: 0.0000000 sec L:12

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

- File Menu:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help.
- Toolbar:** Includes icons for RST, Open, Save, Run, Stop, Break, and various simulation and memory dump tools.
- Registers Window:** Shows the current values of all ARM registers. A red arrow points from the highlighted value of R1 (0x00019999) in the Registers window to the LSRS instruction in the Disassembly window.
- Disassembly Window:** Displays the assembly code. The instruction at address 0x0000001C, E1B01781 MOVS R1,R1,LSL #15, is highlighted in yellow and has its binary representation (0000 0000 0000 0001 1001 1001 1001 1001) shown in a hex dump view below it. A blue arrow points from the assembly line to the corresponding line in the Source Editor.
- Source Editor:** Shows the assembly source code for `asm_for_tutorial.asm`. The instruction at address 0x0000001C is also highlighted in yellow. A green arrow points from the assembly line to the corresponding line in the Source Editor.
- Memory Window:** Shows a memory dump starting at address 0x00000040. The first few bytes are CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00.
- Command Window:** Shows the command `ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet`.

ARM's Data-Processing Instructions (Shift Operations)

```

    graph LR
        Operand[Operand] --> LSL[LSL]
        LSL --> C((C))
        LSL --> Result[Result]
        C --> Result
    
```

The screenshot shows the µVision4 IDE interface during a debugger session. The Registers window highlights the CPSR register, specifically the C bit which is set to 1. The Disassembly window shows several MOVS and LSLS instructions. The assembly code editor has line 13 selected, which contains the LSLS r1,r1,#15 instruction. The Memory window shows memory starting at address 0x00000040 filled with the byte CC. A callout bubble says "Press Step, or F11".

Registers

Register	Value
R0	0x00000000
R1	0x00019999
R2	0x00000000
R3	0x00000002
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)	0x0000001C
CPSR	0x200000D3
N	0
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13

Disassembly

```

0x0000000C E1B01311 MOVS R1,R1,LSL R3
  9:          LSRS r1,r1,#10
0x00000010 E1B01521 MOVS R1,R1,LSR #10
  10:         LSRS r1,r1,r3
  11:
0x00000014 E1B01331 MOVS R1,R1,LSR R3
  12:         ASRS r1,r1,#2
0x00000018 E1B01141 MOVS R1,R1,ASR #2
  13:         LSLS r1,r1,#15
→ 0x0000001C E1B01781 MOVS R1,R1,LSL #15
  14:         ASRS r1,r1,#16
  15:
0x00000020 E1B01841 MOVS R1,R1,ASR #16
  16:         ASRS r1,r1,#16
  17:
  18:
  19:
  20:

```

asm_for_tutorial.asm

```

10      LSRS r1,r1,r3
11
12      ASRS r1,r1,#2
13      LSLS r1,r1,#15
14      ASRS r1,r1,#16
15
16      ASRS r1,r1,r3
17
18      RORS r1,r1,#4
19      RORS r1,r1,r3
20

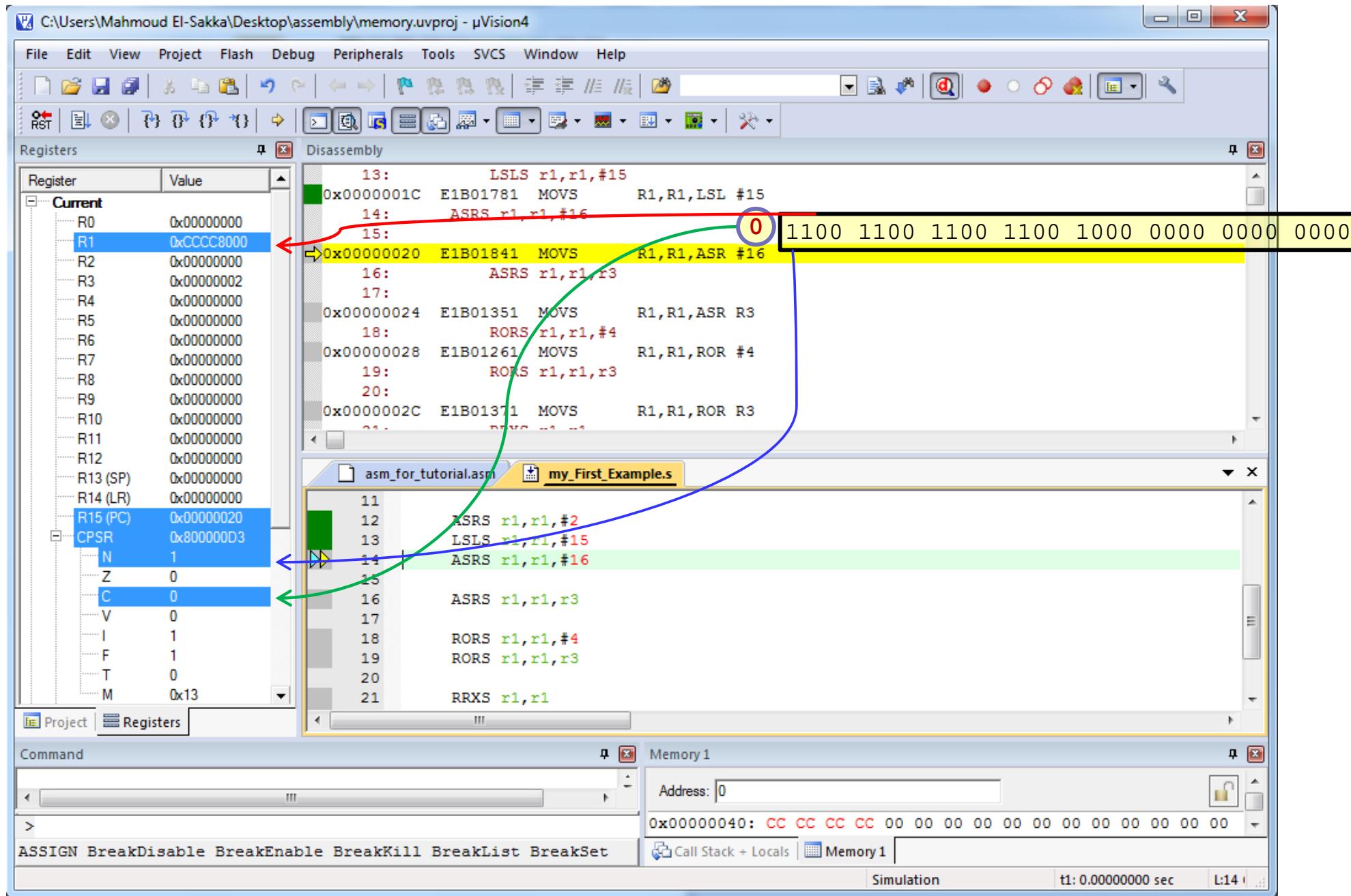
```

Memory 1

Address:	Value
0x00000040	CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00

© Mahmoud R. El-Sakka 19 CS 2208: Introduction to Computer Organization and Architecture

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

Diagram illustrating the ARM ASR (Arithmetic Shift Right) operation:

```

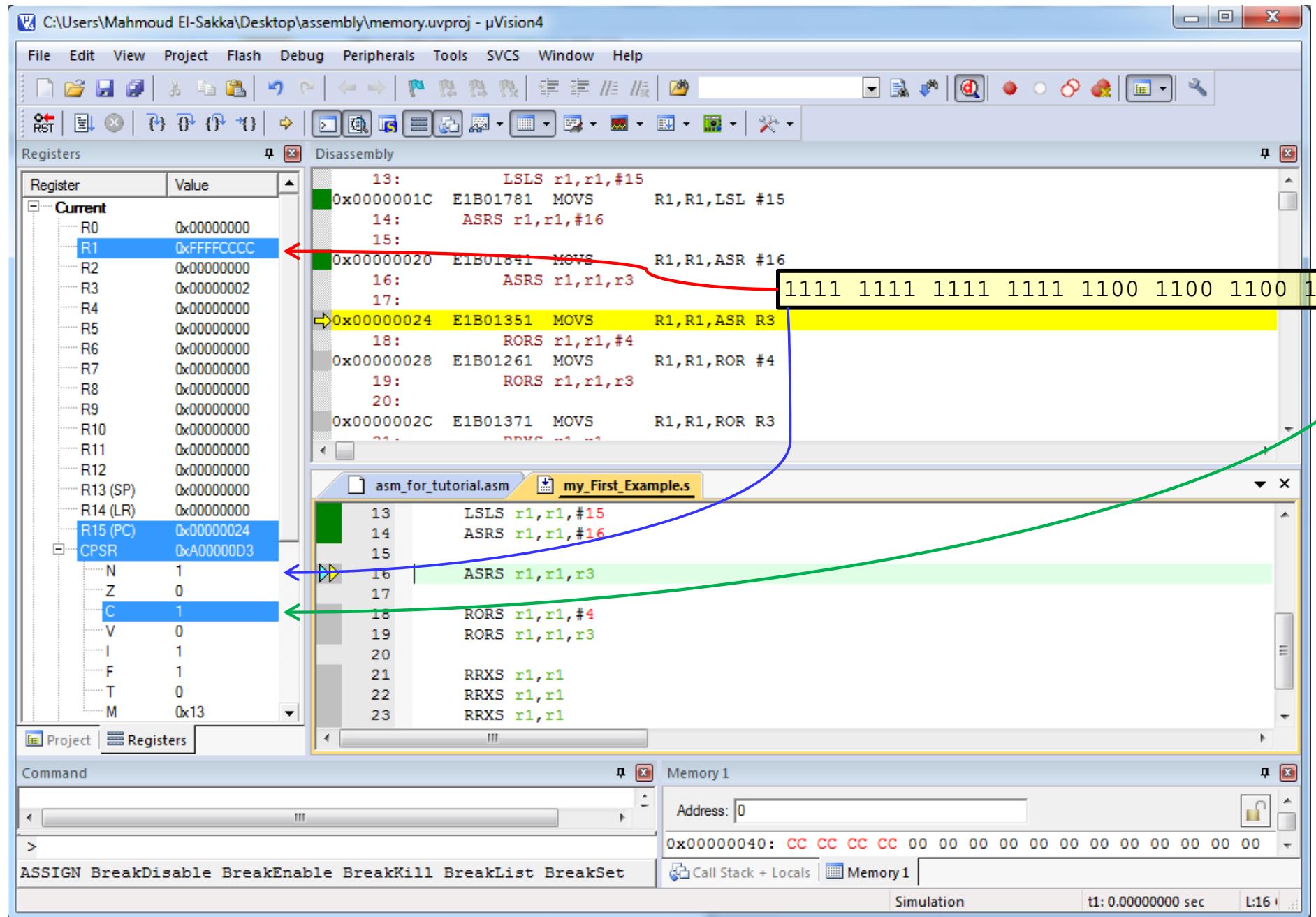
    graph LR
        MSB[MSB] --> Operand[Operand]
        Operand --> C[C]
    
```

The diagram shows the MSB (Most Significant Bit) of the operand being shifted right along with the rest of the bits.

Screenshot of the µVision4 IDE showing ARM assembly code and registers:

- Registers:** Shows the current state of various ARM registers (R0-R14, CPSR, R15 PC). The R1 register is highlighted in yellow.
- Disassembly:** Shows assembly code with addresses, opcodes, and comments. The highlighted instruction at address 0x00000020 is `ASRS r1,r1,#16`.
- Memory View:** Displays the memory contents at address 0x00000040. The bytes are shown in hex (CC CC CC CC) and binary (1100 1100 1100 1100).
- Code Editor:** Shows the source code for `asm_for_tutorial.asm`. The instruction `ASRS r1,r1,#16` is highlighted in green.
- Cloud Callout:** A blue cloud-shaped callout contains the text "Press Step, or F11".

ARM's Data-Processing Instructions (Shift Operations)



1

ARM's Data-Processing Instructions (Shift Operations)

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 title bar displays the project path: C:\Users\Mahmoud El-Sakka\Desktop\assembly\memory.uvproj - µVision4.

Registers:

Register	Value
Current	
R0	0x00000000
R1	0xFFFFCCCC
R2	0x00000000
R3	0x00000002
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)	0x00000024
CPSR	0xA00000D3
N	1
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13

Disassembly:

```

13:      LSLS r1,r1,#15
0x0000001C E1B01781 MOVS    R1,R1,LSL #15
14:      ASRS r1,r1,#16
15:
0x00000020 E1B01841 MOVS    R1,R1,ASR #16
16:      ASRS r1,r1,r3
17:
→0x00000024 E1B01351 MOVS    R1,R1,ASR R3
18:      RORS r1,r1,#4
0x00000028 E1B01261 MOVS    R1,R1,ROR #4
19:      RORS r1,r1,r3
20:
0x0000002C E1B01371 MOVS    R1,R1,ROR R3
21:      DDVS r1,r1

```

Memory1:

Address	Value
0x00000040	CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Cloud Callout:

Press Step,
or F11

Diagram:

```

graph LR
    MSB[MSB] --> ASR[ASR]
    ASR[ASR] --> C[C]
    Operand[Operand] --> ASR

```

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

- Registers Window:** Shows the current values of various ARM registers. R1 is highlighted with a blue selection bar and has a red arrow pointing from it to the Disassembly window.
- Disassembly Window:** Displays the assembly code. The instruction at address 0x00000028 is highlighted in yellow and has a blue arrow pointing from the Registers window to it. The instruction is `E1B01261 MOVS R1,R1,ROR #4`. The result of this instruction is shown in the Memory dump window.
- Memory Dump Window:** Shows the memory starting at address 0x00000040. The value at this address is 0xCCCCCCCC, which corresponds to the binary representation shown in the Disassembly window: 1111 1111 1111 1111 0011 0011 0011 0011. A green circle with the number 0 is located near the bottom right of the dump area.
- Assembly Editor:** Contains the assembly source code:

```

asm_for_tutorial.asm my_First_Example.s
15
16     ASRS r1,r1,r3
17
18     RORS r1,r1,#4
19     RORS r1,r1,r3
20
21     RRXS r1,r1
22     RRXS r1,r1
23     RRXS r1,r1
24     RRXS r1,r1
25     END
    
```
- Command Window:** Shows the command `ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet`.
- Memory Window:** Shows the memory starting at address 0x00000040 with the value 0xCCCCCCCC.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

- Title Bar:** C:\Users\Mahmoud El-Sakka\Desktop\assembly\memory.uvproj - µVision4
- Menu Bar:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help
- Toolbars:** Standard toolbar with icons for file operations, project management, and simulation.
- Registers Window:** Shows the current register values. R1 is highlighted with a yellow circle.
- Disassembly Window:** Displays assembly code with addresses, opcodes, and comments. Instruction 18 is highlighted with a yellow box.
- Binary View:** Shows the binary representation of the highlighted instruction (RORS r1,r1,#4). The result is 1111 1111 1111 1111 1111 0011 0011 0011.
- Assembly Editor:** Shows the source assembly code for the current file.
- Memory Window:** Shows memory starting at address 0x00000040 with data CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00.
- Diagram:** A block diagram of the ROR (Rotate Right) operation. It shows an "Operand" block with an arrow pointing to a "C" block, representing the result of the rotation.
- Cloud Callout:** A blue cloud-shaped callout with the text "Press Step, or F11" inside it.

Registers Window Content:

Register	Value
R0	0x00000000
R1	0xFFFFF333
R2	0x00000000
R3	0x00000002
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)	0x00000028
CPSR	0x800000D3
N	1
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13

Disassembly Window Content:

```

13:      LSLS r1,r1,#15
0x00000001C E1B01781 MOVS    R1,R1,LSL #15
14:      ASRS r1,r1,#16
15:
0x000000020 E1B01841 MOVS    R1,R1,ASR #16
16:      ASRS r1,r1,r3
17:
0x000000024 E1B01351 MOVS    R1,R1,ASR R3
18:      RORS r1,r1,#4
0x000000028 E1B01261 MOVS    R1,R1,ROR #4
19:      RORS r1,r1,r3
20:
0x00000002C E1B01371 MOVS    R1,R1,ROR R3
21:      DDVS r1,r1

```

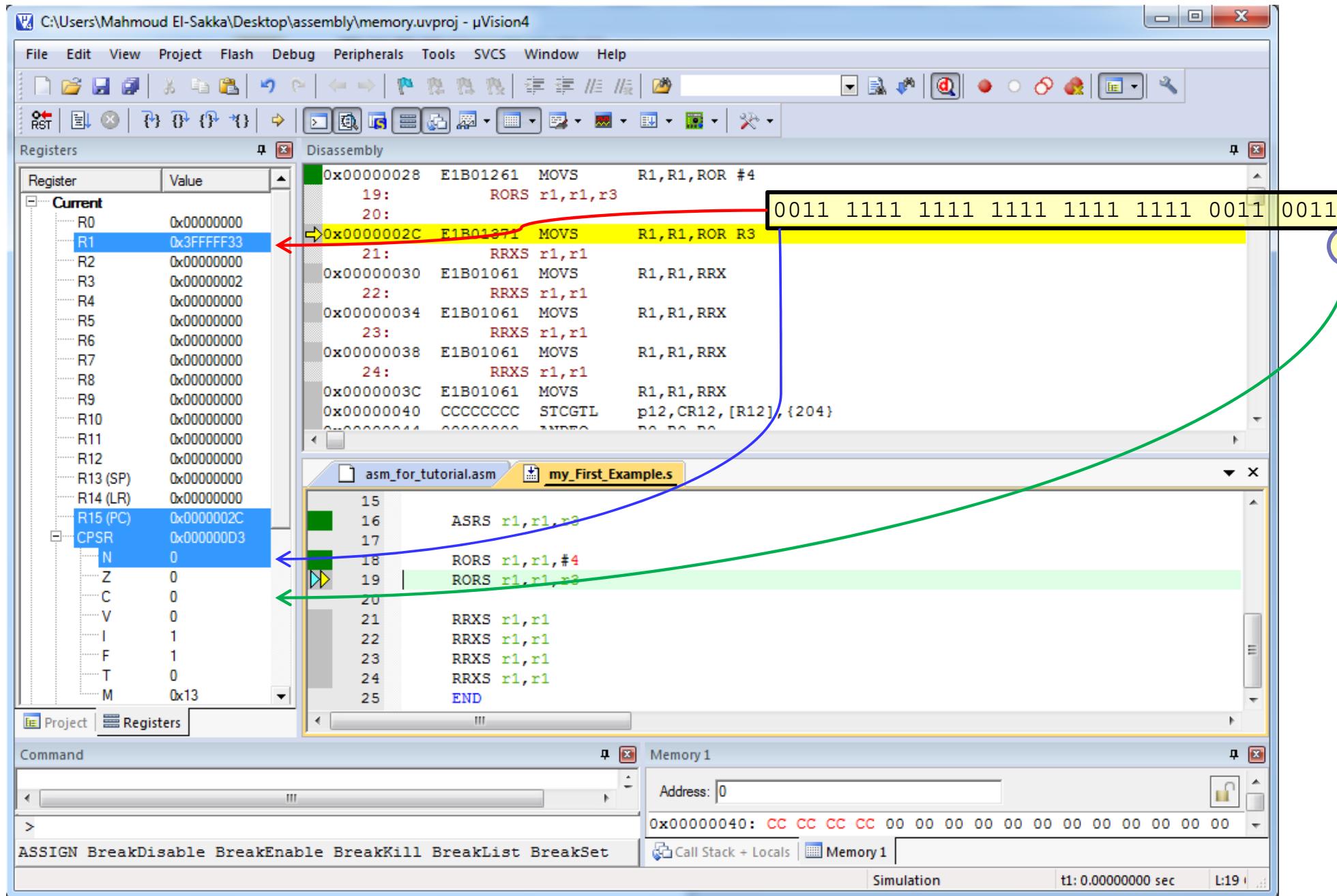
Assembly Editor Content:

```

15
16      ASRS r1,r1,r3
17
18      RORS r1,r1,#4
19      RORS r1,r1,r3
20
21      RRXS r1,r1
22      RRXS r1,r1
23      RRXS r1,r1
24      RRXS r1,r1
25      END

```

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

```

    graph LR
        Operand[Operand] --> ROR[ROR]
        ROR --> C[C]
    
```

The screenshot shows the µVision4 IDE interface. The Registers window highlights the R1 register (Value: 0x3FFFFF33). The Disassembly window shows assembly code with the instruction at address 0x0000002C: RORS r1,r1,r3. The Registers window also shows the CPSR register with bit N set to 0. The Memory window shows memory starting at address 0x00000040 with the value CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

Press Step, or F11

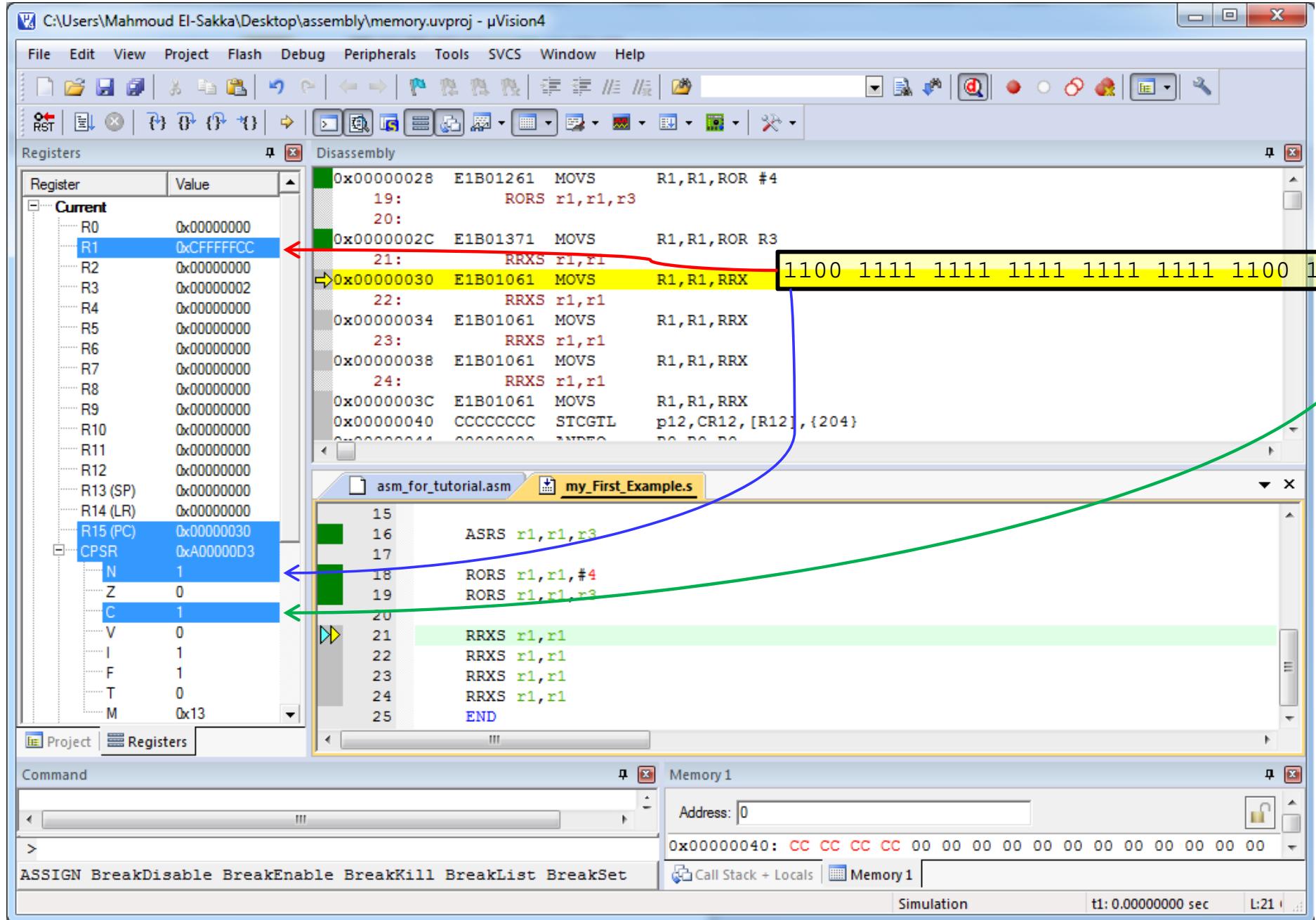
Assembly code in the editor:

```

asm_forTutorial.asm my_First_Example.s
15
16     ASRS r1,r1,r3
17
18     RORS r1,r1,#4
19     RORS r1,r1,r3
20
21     RRXS r1,r1
22     RRXS r1,r1
23     RRXS r1,r1
24     RRXS r1,r1
25     END

```

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

Rotate right through carry

Diagram illustrating the Rotate right through carry operation:

```

    graph LR
        In(( )) --> Register[Register]
        Register --> Out(( ))
        Out --> Carry[Carry]
        Carry --> In
    
```

Registers

Register	Value
R0	0x00000000
R1	0xCFFFFFFC
R2	0x00000000
R3	0x00000002
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)	0x00000030
CPSR	0xA00000D3
N	1
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13

Disassembly

```

0x00000028 E1B01261 MOVS   R1,R1,ROR #4
19:          RORS r1,r1,r3
20:
0x0000002C E1B01371 MOVS   R1,R1,ROR R3
21:          RRXS r1,r1
→ 0x00000030 E1B01061 MOVS   R1,R1,RRX
22:          RRXS r1,r1
0x00000034 E1B01061 MOVS   R1,R1,RRX
23:          RRXS r1,r1
0x00000038 E1B01061 MOVS   R1,R1,RRX
24:          RRXS r1,r1
0x0000003C E1B01061 MOVS   R1,R1,RRX
0x00000040 CCCCCCCC STCGTL p12,CR12,[R12],{204}
0x00000044 00000000 ANDEQ  R0,R0,R0

```

asm_for_tutorial.asm

```

15
16      ASRS r1,r1,r3
17
18      RORS r1,r1,#4
19      RORS r1,r1,r3
20
21      RRXS r1,r1
22      RRXS r1,r1
23      RRXS r1,r1
24      RRXS r1,r1
25      END

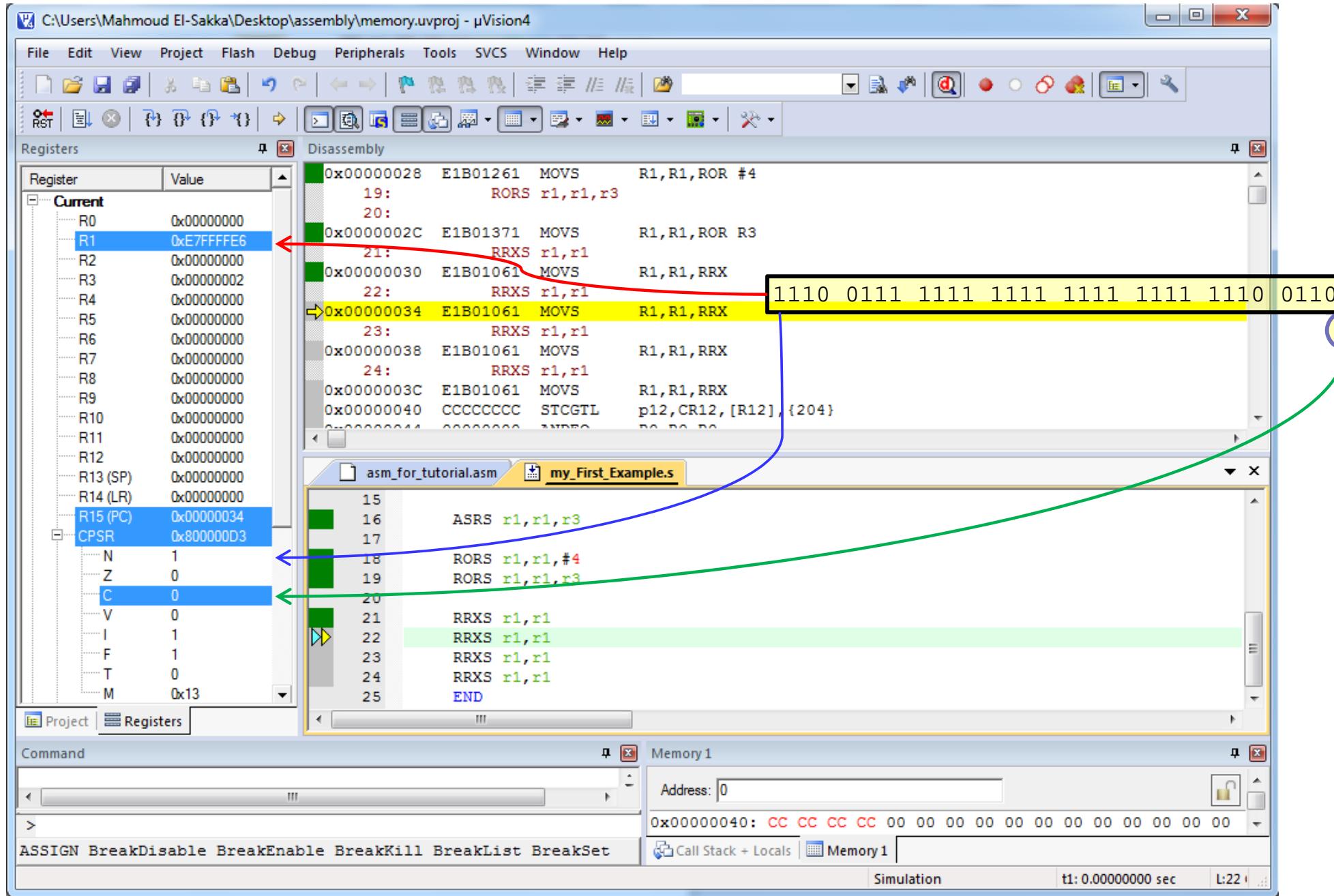
```

Memory 1

Address	Value
0x00000040	CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00

© Mahmoud R. El-Sakka

ARM's Data-Processing Instructions (Shift Operations)

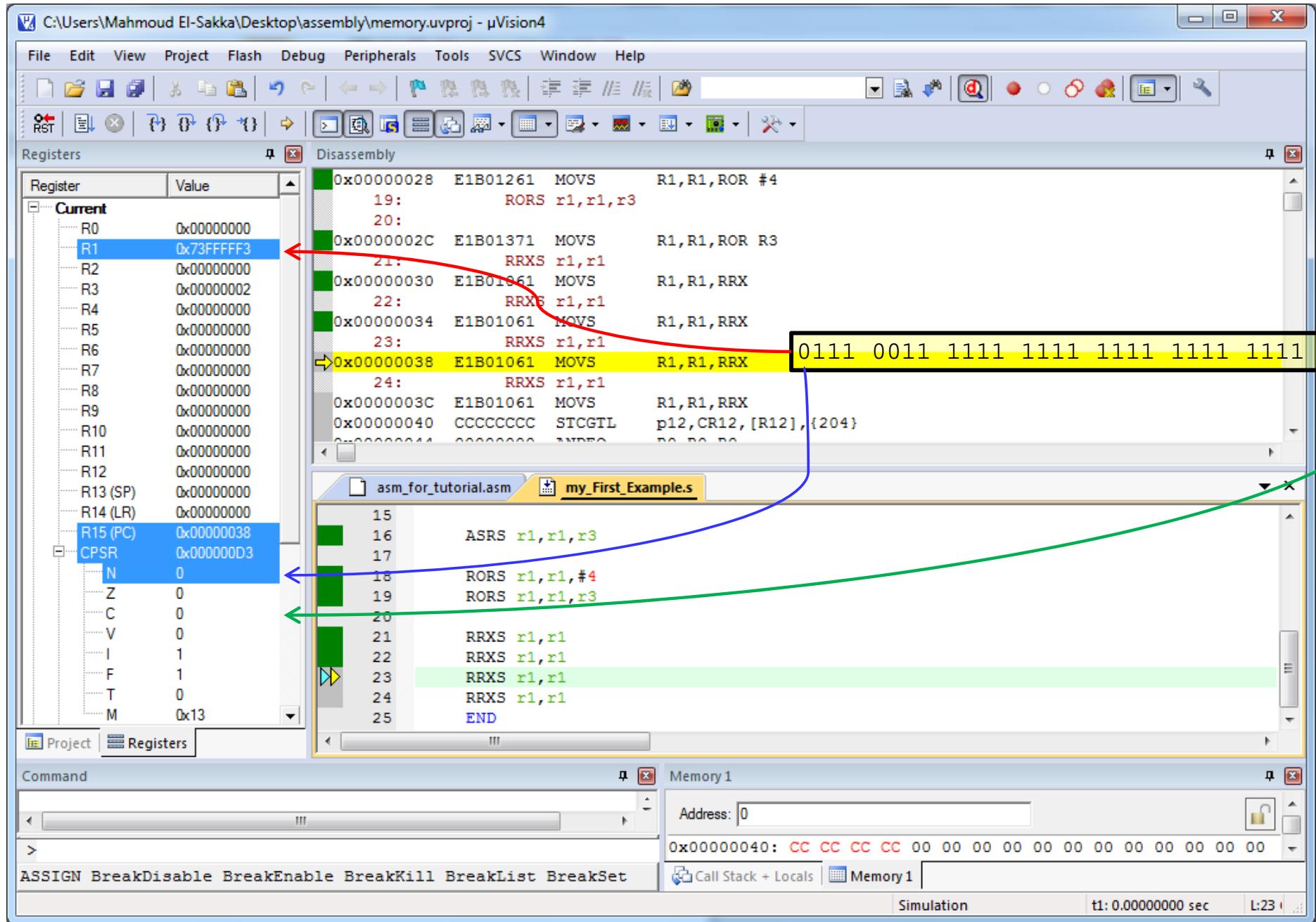


ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

- Registers:** Shows the current values of various ARM registers. The PC (R15) is at 0x00000034, and the CPSR flags N, Z, C, V, I, F, T, M are set to 1, 0, 0, 0, 1, 1, 0, 0 respectively.
- Disassembly:** Displays assembly code for the program. The highlighted instruction is MOVS R1, R1, RRX at address 0x00000034. The assembly code window shows the following sequence of instructions:
 - 15: ASRS r1, r1, r3
 - 16: RORS r1, r1, #4
 - 17: RORS r1, r1, r3
 - 18: RRXS r1, r1
 - 19: RRXS r1, r1
 - 20: RRXS r1, r1
 - 21: RRXS r1, r1
 - 22: RRXS r1, r1
 - 23: RRXS r1, r1
 - 24: RRXS r1, r1
 - 25: END
- Memory Dump:** Shows the memory starting at address 0x00000040. The bytes are displayed as CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00.
- Diagram:** A red box highlights a diagram titled "Rotate right through carry". It shows a register shifting its bits to the right, with the carry bit from the most significant position being moved back into the least significant position.
- Cloud Callout:** A blue cloud contains the text "Press Step, or F11", with a red arrow pointing from the text in the assembly code window towards it.

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

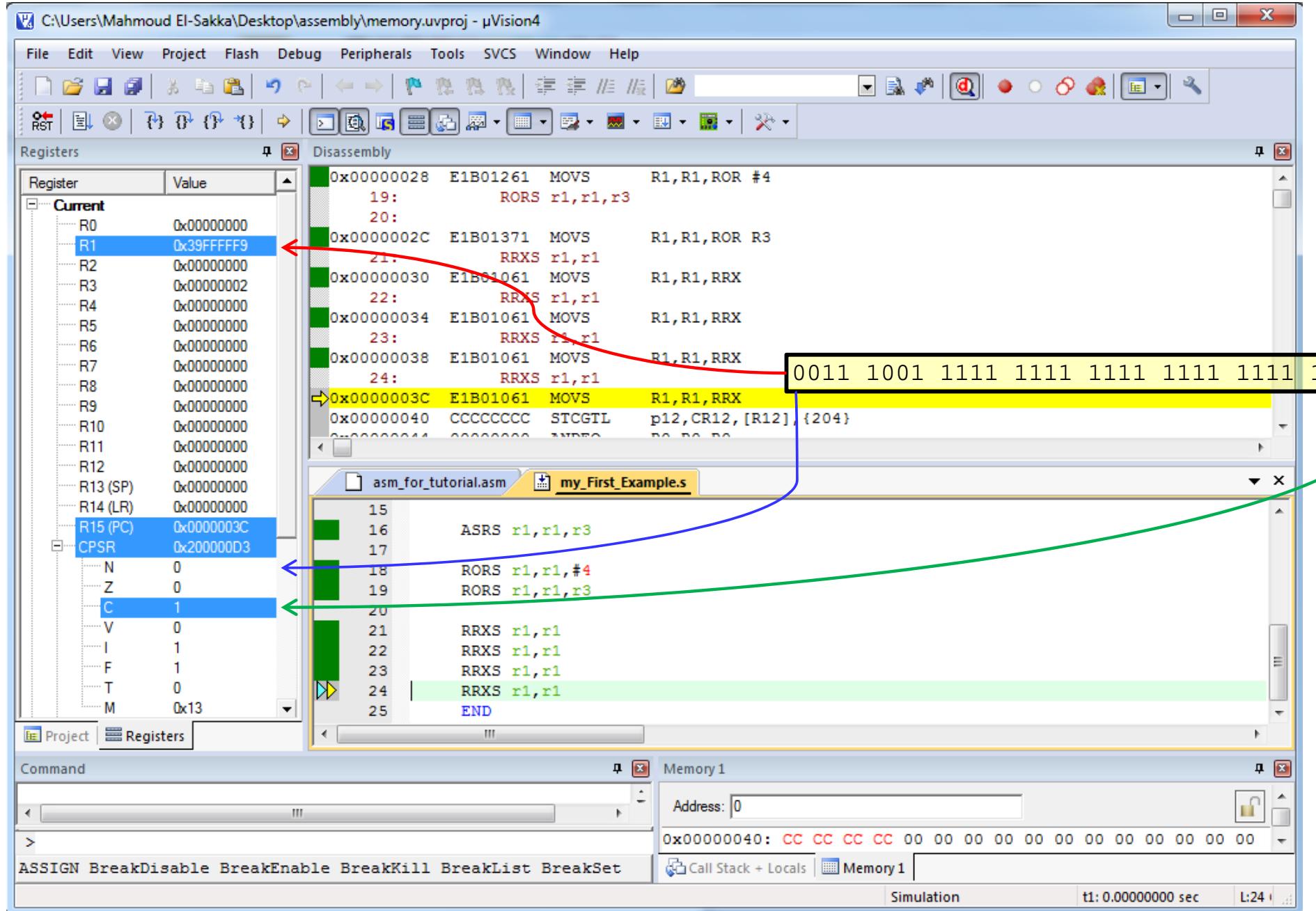
The screenshot shows the µVision4 IDE interface with the following components:

- Registers:** Shows the current values of various ARM registers. The R1 register is highlighted with a blue circle.
- Disassembly:** Displays assembly code for the program. Instruction 23 (R00000038) is highlighted in yellow. The assembly code for this instruction is `E1B01061 MOVS R1,R1,RRX`. To its right is a memory dump showing the binary value `0111 0011 1111 1111 1111 1111 1111 0011` followed by `0011 1001 1111 1111 1111 1111 1111 1001`.
- Code Editor:** Shows the assembly source code:


```

asm_forTutorial.asm my_First_Example.s
15
16     ASRS r1,r1,r3
17
18     RORS r1,r1,#4
19     RORS r1,r1,r3
20
21     RRXS r1,r1
22     RRXS r1,r1
23     RRXS r1,r1
24     RRXS r1,r1
25     END
            
```
- Memory:** A window showing memory starting at address 0x00000040. The first few bytes are `CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00`.
- Diagram:** A red box highlights a diagram titled "Rotate right through carry". It shows a sequence of four boxes: "Register", "→", "Carry", and another "→". An arrow points from the "Carry" box back to the "Register" box, indicating that the carry bit from the most significant bit position is rotated back into the least significant bit position.
- Callout:** A blue cloud-shaped callout contains the text "Press Step, or F11", with a red arrow pointing from the text towards the assembly code editor.

ARM's Data-Processing Instructions (Shift Operations)

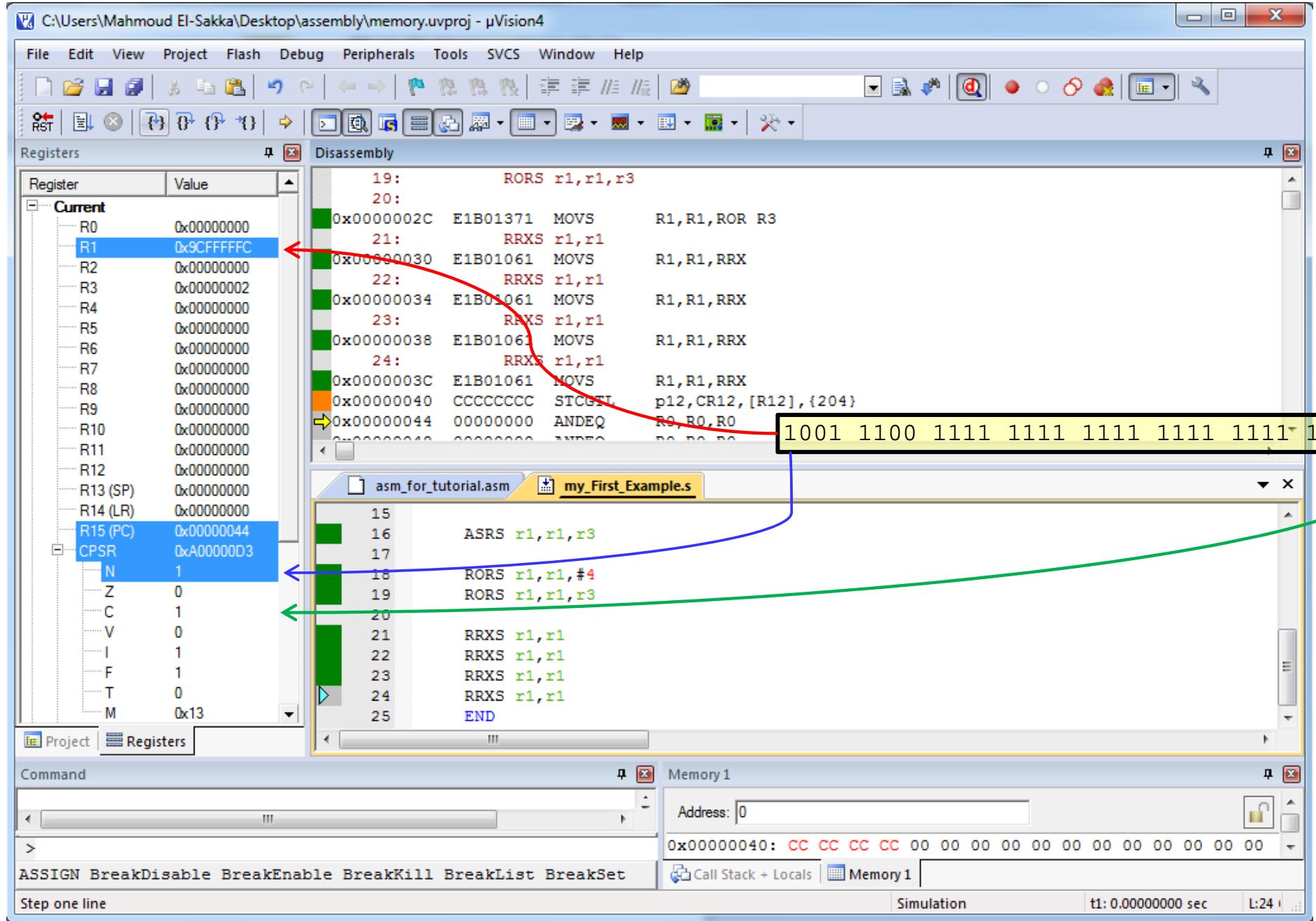


ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the µVision4 IDE interface with the following components:

- File Menu:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help.
- Toolbars:** Standard toolbar with icons for file operations, project management, and debugging.
- Registers Window:** Shows the current values of various ARM registers. The R1 register (Value: 0x39FFFFFF) is highlighted with a yellow circle.
- Disassembly Window:** Displays assembly code. The current instruction is highlighted in yellow: E1B01061 MOVS R1,R1,RRX. To its right is a memory dump showing the byte sequence: 0011 1001 1111 1111 1111 1111 1111 1001. Below it is another memory dump: 1001 1100 1111 1111 1111 1111 1111 1100.
- Code Editor:** Shows the assembly source code in the file "asm_forTutorial.asm". The current line is highlighted in green: RRXS r1,r1.
- Memory Window:** Shows memory starting at address 0x00000040 with the value CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00.
- Diagram:** A red box highlights a diagram titled "Rotate right through carry" which illustrates the ROR operation: a register shifts right, and the carry bit from the most significant position is moved to the least significant position.
- Cloud Callout:** A blue cloud-shaped callout contains the text "Press Step, or F11" with a red arrow pointing from the text towards the assembly code editor.

ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

```
AREA prog1, code, READONLY
ENTRY
MOV r3,#2
LDR r1, =0xFFFFFFFF ;in binary 1100 1100 1100 1100 1100 1100 1100 1100
LSL r1,r1,#5
LSL r1,r1,r3

LSR r1,r1,#10
LSR r1,r1,r3

ASR r1,r1,#2
LSL r1,r1,#15
ASR r1,r1,#16

ASR r1,r1,r3

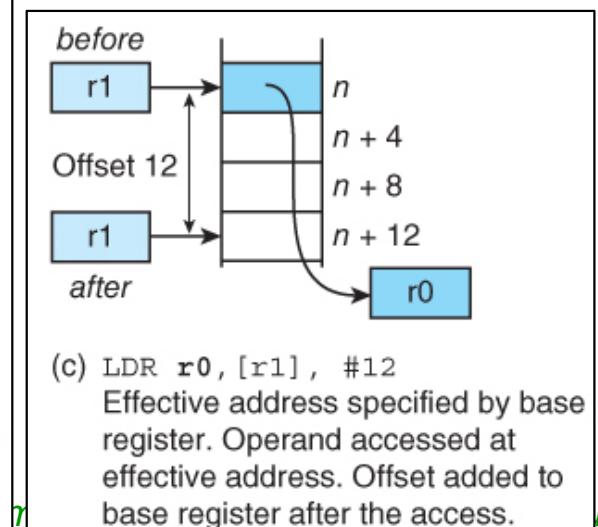
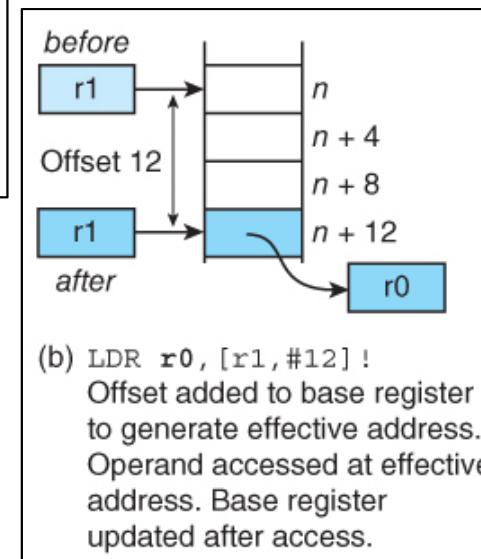
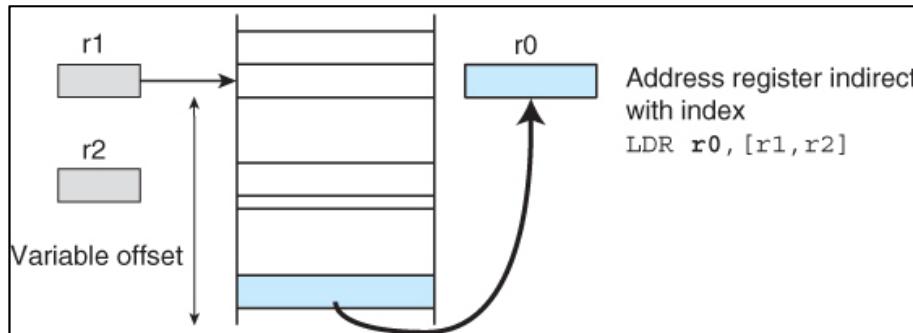
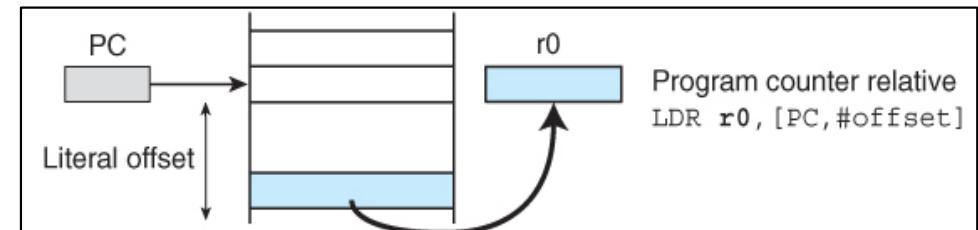
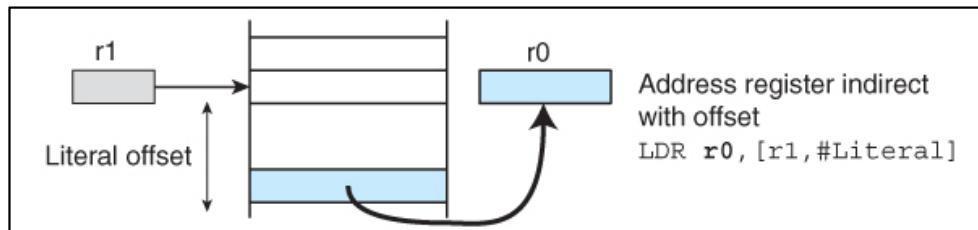
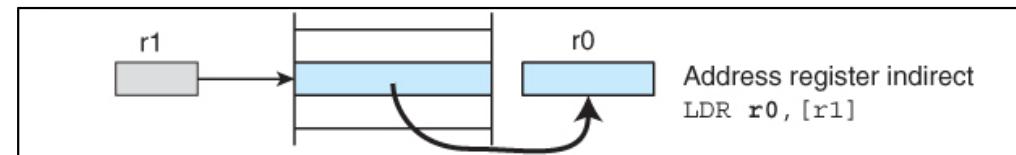
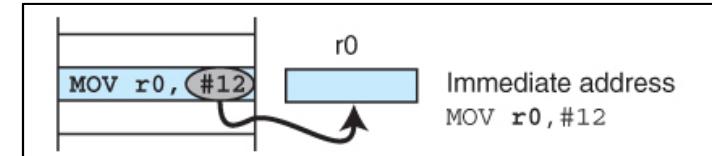
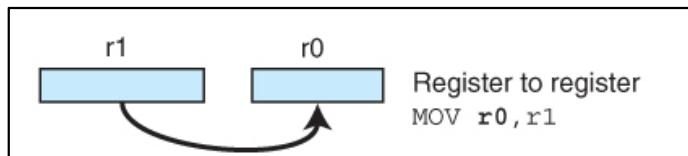
ROR r1,r1,#4
ROR r1,r1,r3

RRX r1,r1
RRX r1,r1
RRX r1,r1
RRX r1,r1
END
```



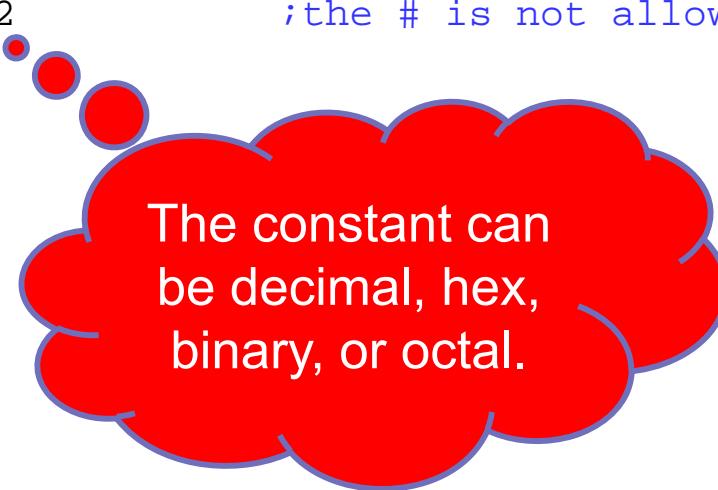
*Repeat
the example again
without the “S”*

ARM Addressing Modes



ARM Addressing Modes

```
Start ;register to register  
MOV r0,r1  
  
;immediate address  
MOV r2,#12           ;(decimal)  
MOV r2,#0x12         ;(hexadecimal)  
MOV r2,#2_1000       ;(binary)  
  
;PC indirect address with offset  
ADR r3,AAA  
ADR r3,AAA + 12      ;the # is not allowed here
```



The constant can
be decimal, hex,
binary, or octal.

ARM Addressing Modes

;register indirect address

```
LDR r4,[r3]
```

```
LDR r5,[r3,#4]           ;with      offset
```

```
LDR r5,[r3,#-8]          ;with -ve offset
```

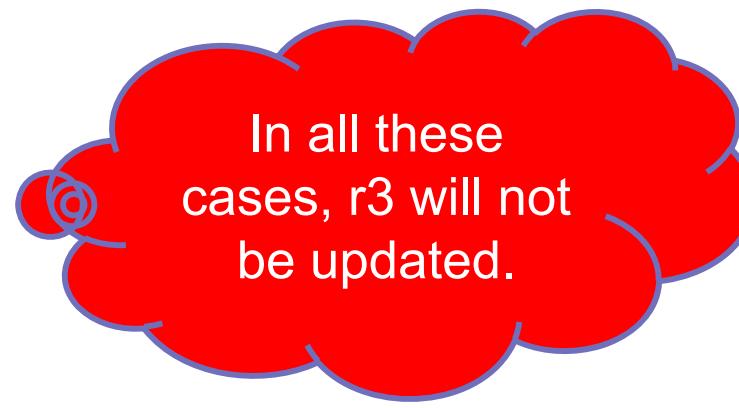
```
LDR r5,[r3,#(-4*2+40)/4] ;The offset can be constant expression  
                           ;to be evaluated by the assembler
```

```
LDR r5,[r3,r2]           ;with      index
```

```
LDR r5,[r3,-r2]          ;with -ve index
```

```
LDR r5,[r3,r2,ASR #1]    ;with      index and scaling
```

```
LDR r5,[r3,-r2,ASR #1]   ;with -ve index and scaling
```



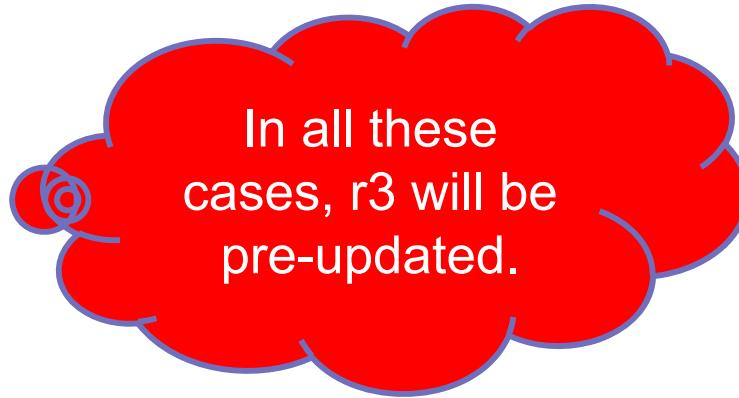
In all these cases, r3 will not be updated.

ARM Addressing Modes

```
LDR r5,[r3,#4]!           ;with      offset and autoindexing pre-update
LDR r5,[r3,#-8]!          ;with -ve offset and autoindexing pre-update
LDR r5,[r3,#(-4*2+40)/4]! ;the offset can be constant expression
                            ;to be evaluated by the assembler

LDR r5,[r3,r2]!           ;with      index and autoindexing pre-update
LDR r5,[r3,-r2]!          ;with -ve index and autoindexing pre-update

LDR r5,[r3,r2,ASR #1]!    ;with      index, scaling, and autoindexing pre-update
LDR r5,[r3,-r2,ASR #1]!    ;with -ve index, scaling, and autoindexing pre-update
```



In all these cases, r3 will be pre-updated.

ARM Addressing Modes

```

LDR r5,[r3],#4           ;with      offset and autoindexing post-update
LDR r5,[r3],#-8          ;with -ve offset and autoindexing post-update
LDR r5,[r3],#(-4*2+40)/4 ;the offset can be constant expression
                           ;to be evaluated by the assembler

LDR r5,[r3],r2            ;with      index and autoindexing post-update
LDR r5,[r3],-r2           ;with -ve index and autoindexing post-update

LDR r5,[r3],r2,LSL #1     ;with      index, scaling, and autoindexing post-update
LDR r5,[r3],-r2,LSL #1     ;with -ve index, scaling, and autoindexing post-update

loop B loop

AAA DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
      DCD 2_1110000,2_10000000,0x90,0xA0,0xB0,0xC0, 0xD0

```

In all these cases, r3 will be post-updated.

ARM Addressing Modes

The screenshot shows the µVision4 IDE interface with the following components:

- File Menu:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help.
- Toolbar:** Includes icons for RST, Run, Stop, Break, and various simulation and memory manipulation tools.
- Registers Window:** Shows the current values of all ARM registers (R0-R15, CPSR, SPSR) in hex format.
- Disassembly Window:** Displays the assembly code with comments explaining the addressing modes used in each instruction.
- Source Editor:** Shows the assembly source code for "OpCodes.s" with line numbers and comments.
- Command Window:** Displays memory usage information: "Restricted Version with 32768 Byte Code Size Limit" and "Currently used: 584 Bytes (1%)".
- Memory Window:** Shows the memory dump at address 0x76, displaying the byte sequence: 0x00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00.
- Bottom Status Bar:** Shows the current simulation time (t1: 0.0000000 sec), clock state (L4 C1), and CAP NUM.

Disassembly Window Content:

```

0x00000000 E1A00001 MOV R0, R1
    7:          MOV r2,#12      ;(decimal)
0x00000004 E3A0200C MOV R2,#0x0000000C ;(hexadecimal)
    8:          MOV r2,#0x12      ;(hexadecimal)
0x00000008 E3A02012 MOV R2,#0x00000012 ;(hexadecimal)
    9:          MOV r2,#2_1000    ;(binary)
   10:
   11:          ;PC indirect address with offset
0x0000000C E3A02008 MOV R2,#0x00000008
   12:          ADR r3,AAA

```

Source Editor Content:

```

1 AREA AddressingModes, CODE, READWRITE
2
3 Start ;register to register
4     MOV r0,r1
5
6 ;immediate address
7     MOV r2,#12      ;(decimal)
8     MOV r2,#0x12      ;(hexadecimal)
9     MOV r2,#2_1000    ;(binary)
10
11 ;PC indirect address with offset
12     ADR r3,AAA
13     ADR r3,AAA + 12 ;The # is not allowed here
14
15 ;register indirect address
16     LDR r4,[r3]
17
18     LDR r5,[r3,#4] ;with offset

```

ARM Addressing Modes

The screenshot shows the µVision4 IDE interface with the following windows:

- Registers**: Shows the current values of various ARM registers, with R15 (PC) highlighted at 0x00000004.
- Disassembly**: Displays the assembly code for the program. The highlighted instruction is at address 0x00000004: E3A0200C MOV R2, #0x0000000C, annotated with ;(decimal).
- OpCodes.s**: Shows the source assembly file with comments explaining the instructions. It includes entries for immediate addresses, PC indirect addresses with offsets, and register indirect addresses.
- Command**: Shows compiler messages about restricted version and memory usage.
- Memory 1**: Shows the memory dump starting at address 0x00000076.

```

0x00000000 E1A00001 MOV R0, R1
    7:           MOV r2, #12          ;(decimal)
0x00000004 E3A0200C MOV R2, #0x0000000C
    8:           MOV r2, #0x12          ;(hexadecimal)
0x00000008 E3A02012 MOV R2, #0x00000012
    9:           MOV r2, #2_1000        ;(binary)
   10:
   11:           ;PC indirect address with offset
0x0000000C E3A02008 MOV R2, #0x00000008
   12:           ADR r3,AAA

```

```

2      ENTRY
3  Start ;register to register
        MOV r0,r1
5
6      ;immediate address
7  MOV r2,#12          ;(decimal)
8  MOV r2,#0x12          ;(hexadecimal)
9  MOV r2,#2_1000        ;(binary)
10
11      ;PC indirect address with offset
12  ADR r3,AAA
13  ADR r3,AAA + 12      ;The # is not allowed here
14
15      ;register indirect address
16  LDR r4,[r3]
17
18  LDR r5,[r3,#4]        ;with offset
19  LDR r5,[r3,#-8]       ;with -ve offset

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 584 Bytes (1%)

Address: 0x76

0x00000076: 00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00
0x0000008F: 70 00 00 00 80 00 00 00 90 00 00 00 A0 00 00 00 B0 00 00 00 C0 00 00 D0
0x000000A8: E2 8F 00 18 E3 A0 10 05 E3 A0 20 00 E2 51 10 01 E7 90 31 01 E0 82 20 03 1A

ARM Addressing Modes

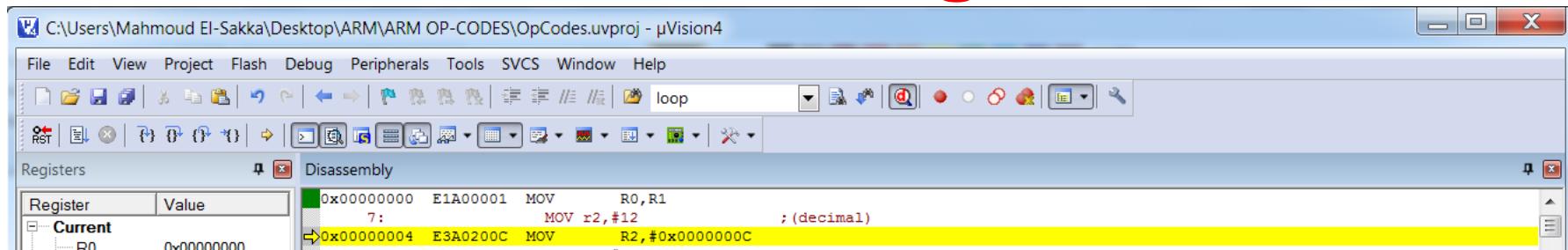
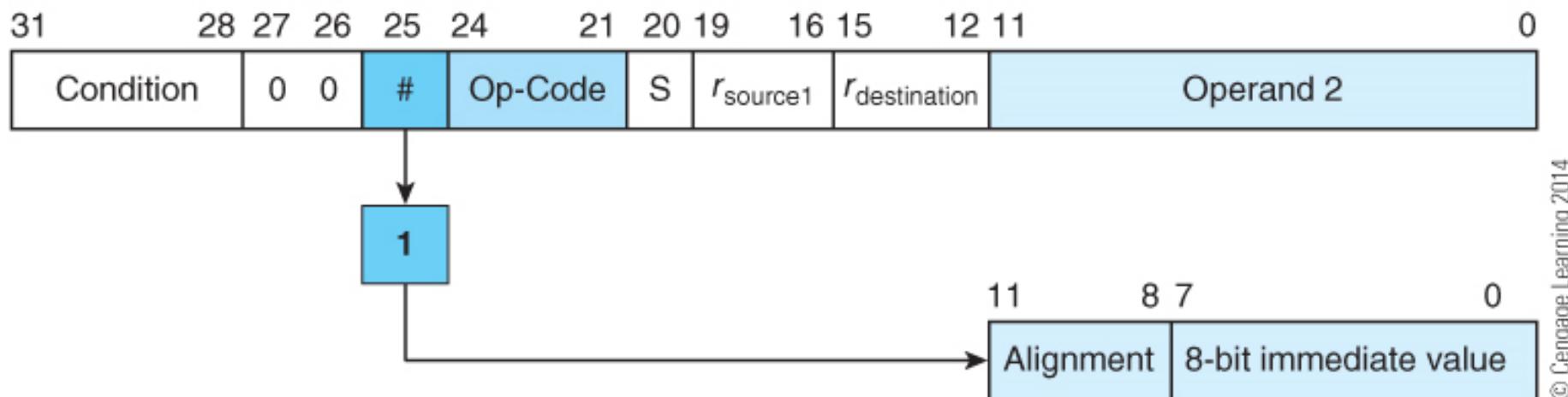


FIGURE 3.28 Diagram of ARM's literal operand encoding



© Cengage Learning 2014

ARM Addressing Modes

Encoded Literal	Scale value	# of shift right 2 × Scale value	# of shift left 32 - 2 × Scale value	Decoded literal
0000 mnop wxyz	0	0	(32) ₁₀	0000 0000 0000 0000 0000 0000 mnop wxyz
1111 mnop wxyz	(15) ₁₀	(30) ₁₀	2	0000 0000 0000 0000 0000 0000 00mn opwx yz00
1110 mnop wxyz	(14) ₁₀	(28) ₁₀	4	0000 0000 0000 0000 0000 mnop wxyz 0000
1101 mnop wxyz	(13) ₁₀	(26) ₁₀	6	0000 0000 0000 0000 0000 00mn opwx yz00 0000
1100 mnop wxyz	(12) ₁₀	(24) ₁₀	8	0000 0000 0000 0000 mnop wxyz 0000 0000
1011 mnop wxyz	(11) ₁₀	(22) ₁₀	(10) ₁₀	0000 0000 0000 00mn opwx yz00 0000 0000
1010 mnop wxyz	(10) ₁₀	(20) ₁₀	(12) ₁₀	0000 0000 0000 mnop wxyz 0000 0000 0000
1001 mnop wxyz	9	(18) ₁₀	(14) ₁₀	0000 0000 00mn opwx yz00 0000 0000 0000
1000 mnop wxyz	8	(16) ₁₀	(16) ₁₀	0000 0000 mnop wxyz 0000 0000 0000 0000
0111 mnop wxyz	7	(14) ₁₀	(18) ₁₀	0000 00mn opwx yz00 0000 0000 0000 0000
0110 mnop wxyz	6	(12) ₁₀	(20) ₁₀	0000 mnop wxyz 0000 0000 0000 0000 0000
0101 mnop wxyz	5	(10) ₁₀	(22) ₁₀	00mn opwx yz00 0000 0000 0000 0000 0000
0100 mnop wxyz	4	8	(24) ₁₀	mnop wxyz 0000 0000 0000 0000 0000 0000
0011 mnop wxyz	3	6	(26) ₁₀	opwx yz00 0000 0000 0000 0000 0000 00mn
0010 mnop wxyz	2	4	(28) ₁₀	wxyz 0000 0000 0000 0000 0000 0000 mnop
0001 mnop wxyz	1	2	(30) ₁₀	yz00 0000 0000 0000 0000 0000 00mn opwx

ARM Addressing Modes

The screenshot shows the µVision4 IDE interface with the following components:

- File Menu:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help.
- Toolbar:** Includes icons for RST, Open, Save, Run, Stop, Break, and various simulation and memory manipulation tools.
- Registers Window:** Shows the current values of various ARM registers. The R2 and R15 (PC) registers are highlighted in blue, indicating they are selected.
- Disassembly Window:** Displays the assembly code with comments explaining the addressing modes. The highlighted instruction is `MOV r2,#0x12 ;(hexadecimal)`.
- OpCodes.s Editor:** Shows the source assembly file with the same code, where the highlighted instruction is `MOV r2,#0x12 ;(hexadecimal)`.
- Command Window:** Displays memory usage information: "Restricted Version with 32768 Byte Code Size Limit" and "Currently used: 584 Bytes (1%)".
- Memory Window:** Shows the memory dump starting at address 0x76, displaying bytes from 0x00000076 to 0x000000A8.
- Status Bar:** Shows simulation time (t1: 0.0000000 sec), clock (L8 C:1), and CAP NUM.

```

0x00000000 E1A00001 MOV R0,R1
7:          MOV r2,#12      ;(decimal)
0x00000004 E3A0200C MOV R2,#0x0000000C
8:          MOV r2,#0x12      ;(hexadecimal)
0x00000008 E3A02012 MOV R2,#0x00000012
9:          MOV r2,#2_1000    ;(binary)
10:         ;PC indirect address with offset
11:         ;PC indirect address with offset
0x0000000C E3A02008 MOV R2,#0x00000008
12:         ADR r3,AAA

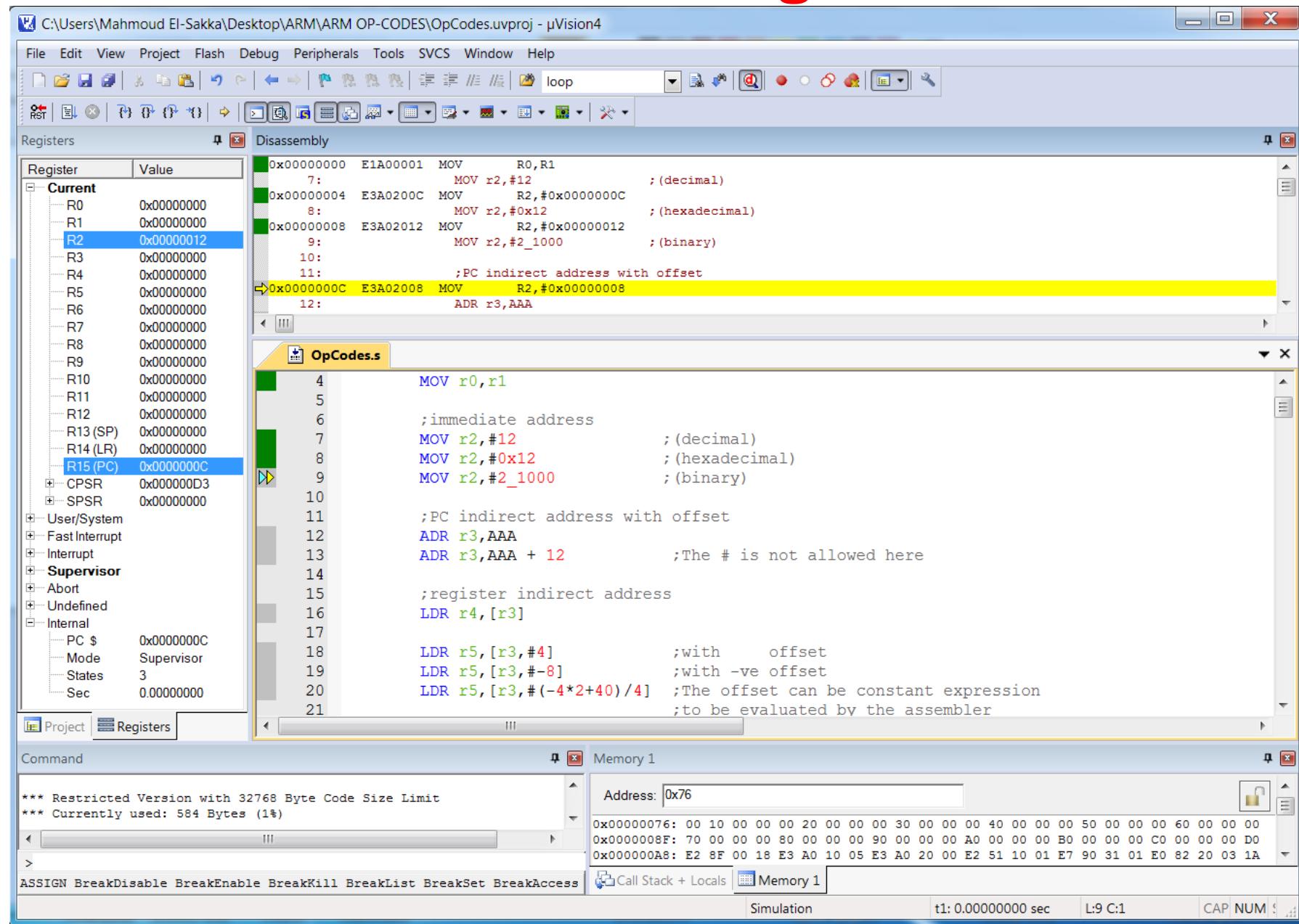
```

```

3 Start ;register to register
4     MOV r0,r1
5
6     ;immediate address
7     MOV r2,#12      ;(decimal)
8     MOV r2,#0x12      ;(hexadecimal)
9     MOV r2,#2_1000    ;(binary)
10
11    ;PC indirect address with offset
12    ADR r3,AAA
13    ADR r3,AAA + 12   ;The # is not allowed here
14
15    ;register indirect address
16    LDR r4,[r3]
17
18    LDR r5,[r3,#4]    ;with offset
19    LDR r5,[r3,-#8]    ;with -ve offset
20    LDR r5,[r3,#(-4*2+40)/4] ;The offset can be constant expression

```

ARM Addressing Modes



ARM Addressing Modes

What is the value to be stored in R3?

The screenshot shows the µVision4 IDE interface with the following windows:

- Registers**: Shows the current values of various ARM registers. R2 is highlighted with a blue selection bar and has a value of 0x00000008. R15 (PC) is also highlighted and has a value of 0x00000010.
- Disassembly**: Displays the assembly code. Line 12: MOV R2, #0x00000008. Line 13: ADD R3, PC, #0x0000005C. A red annotation on line 13 states: "The # is not allowed here". Line 15: LDR r4, [r3]. Line 17: LDR R4, [R3].
- OpCodes.s**: Shows a sample assembly file with comments explaining addressing modes. Lines 11-12: PC indirect address with offset. Line 13: The same instruction as in the disassembly, annotated with "The # is not allowed here". Line 15: Register indirect address. Lines 18-24: Examples of LDR instructions with various addressing modes and offsets.
- Memory 1**: Shows memory dump starting at address 0x76. The dump includes bytes from 0x00000076 to 0x000000A8.
- Command**: Displays build logs and usage statistics. It says "Currently used: 584 Bytes (1%)".
- Call Stack + Locals**: Shows the call stack and local variables.

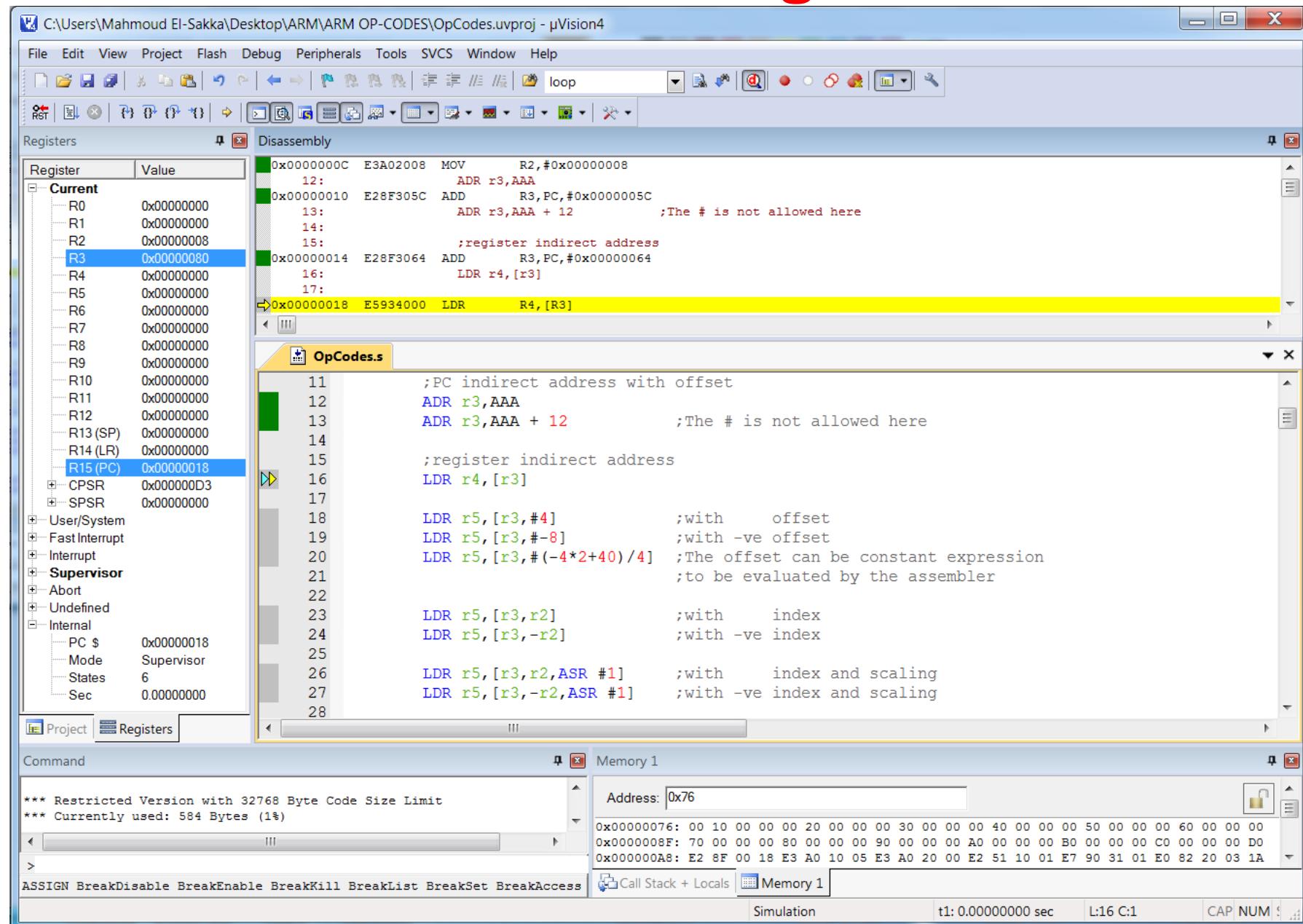
ARM Addressing Modes

What is the value to be stored in R3?

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current values of various ARM registers. The R3 register is highlighted with a blue selection bar and contains the value 0x00000074.
- Disassembly:** Displays the assembly code for the program. The highlighted instruction is at address 0x00000014: ADD R3, PC, #0x00000064. A note next to it says: "The # is not allowed here".
- OpCodes.s:** An assembler source file containing examples of various ARM instructions and their addressing modes. The highlighted line is: ADD R3, PC, #0x00000064. A note next to it says: "The # is not allowed here".
- Memory 1:** A memory dump window showing the byte content at address 0x76. The bytes are: 0x00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00 00 00 D0.
- Command:** A text input field showing compiler messages: "*** Restricted Version with 32768 Byte Code Size Limit" and "*** Currently used: 584 Bytes (1%)".
- Bottom Status Bar:** Shows simulation information: t1: 0.0000000 sec, L:13 C:1, CAP NUM: 4.

ARM Addressing Modes



ARM Addressing Modes

What is the value to be stored in R4?

The screenshot shows the uVision4 IDE interface with the following components:

- Registers** window: Shows the current register values, with R3 and R15 (PC) highlighted.
- Disassembly** window: Displays the assembly code. Line 16 contains the instruction `LDR r4, [r3]`. A note next to it says: `;The # is not allowed here`.
- OpCodes.s** window: Shows examples of various ARM addressing modes. Line 16 shows `LDR r4, [r3]`.
- Memory 1** window: Shows the memory dump starting at address 0x76. The value at 0x76 is 0x00000080, which is highlighted with a blue speech bubble containing the text "0x00000080".
- Command** window: Shows the message: `*** Restricted Version with 32768 Byte Code Size Limit` and `*** Currently used: 584 Bytes (1%)`.

ARM Addressing Modes

What is the value to be stored in R5?

The screenshot shows the µVision4 IDE interface with the following windows:

- Registers**: Shows the current register values, with R4 and R15 (PC) highlighted.
- Disassembly**: Shows ARM assembly code with annotations explaining addressing modes:
 - Line 18: LDR r5, [r3, #4] ;with offset
 - Line 19: LDR r5, [r3, #-8] ;with -ve offset
 - Line 20: LDR r5, [r3, #(-4*2+40)/4] ;The offset can be constant expression to be evaluated by the assembler
- OpCodes.s**: Shows the assembly source code corresponding to the disassembly.
- Memory 1**: Shows the memory dump at address 0x76, where the value 0x00000080 is highlighted.

ARM Addressing Modes

What is the value to be stored in R5?

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 below has various icons for file operations, debugging, and project management. The main window is divided into several panes:

- Registers** pane: Shows the current register values. R5 is highlighted with a blue selection bar.
- Disassembly** pane: Displays assembly code with comments. The highlighted instruction is LDR R5, [R3, #-0x0008]. A yellow arrow points from the question bubble to this instruction.
- OpCodes.s** pane: Shows a list of LDR instructions with their descriptions.
- Memory 1** pane: Displays the memory dump at address 0x76. The value 0x00000080 is highlighted with a blue selection bar and a speech bubble containing the value.
- Command** pane: Shows the message "*** Restricted Version with 32768 Byte Code Size Limit *** Currently used: 584 Bytes (1%)".
- Bottom status bar**: Shows Simulation, t1: 0.00000000 sec, L:19 C:1, and CAP NUM.

ARM Addressing Modes

What is the value to be stored in R5?

The screenshot shows the µVision4 IDE interface with the following windows:

- Registers**: Shows the current values of various registers, with R5 and R15 (PC) highlighted.
- Disassembly**: Displays the assembly code for the current program. The highlighted instruction is LDR r5, [r3, #0x0008]. A yellow arrow points from the question "What is the value to be stored in R5?" to this instruction.
- OpCodes.s**: A separate window showing the assembly code for the LDR instruction, which is used to explain the addressing mode.
- Memory 1**: A dump of memory starting at address 0x00000076, showing the byte sequence: 00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00.

OpCodes.s (Assembly code for LDR instruction):

```

15      ;register indirect address
16      LDR r4, [r3]
17
18      LDR r5, [r3, #4]           ;with    offset
19      LDR r5, [r3, #-8]          ;with -ve offset
20      LDR r5, [r3, #(-4*2+40)/4];The offset can be constant expression
                                ;to be evaluated by the assembler
21
22      LDR r5, [r3, r2]          ;with    index
23      LDR r5, [r3, -r2]          ;with -ve index
24
25      LDR r5, [r3, r2, ASR #1]  ;with    index and scaling
26      LDR r5, [r3, -r2, ASR #1] ;with -ve index and scaling
27
28      LDR r5, [r3, #4]!         ;with    offset and autoindexing pre-update
29      LDR r5, [r3, #-8]!        ;with -ve offset and autoindexing pre-update
30
31      LDR r5, [r3, #(-4*2+40)/4]!;The offset can be constant expression
                                ;to be evaluated by the assembler
32

```

0x00000080

ARM Addressing Modes

What is the value to be stored in R5?

0x00000080

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current values of various registers. R5 is highlighted with a blue selection bar and has a value of 0x00000060.
- Disassembly:** Displays assembly code with comments explaining addressing modes. A yellow selection bar highlights the instruction at address 0x00000028: LDR R5, [R3, R2].
- OpCodes.s:** Shows a list of LDR instructions with their descriptions. The highlighted instruction corresponds to the one in the disassembly.
- Memory 1:** Shows the memory dump starting at address 0x00000076. The value at address 0x00000076 is 0x00000080.
- Command:** Displays compiler messages about code size and memory usage.

ARM Addressing Modes

What is the value to be stored in R5?

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 title bar indicates the project is C:\Users\Mahmoud El-Sakka\Desktop\ARM\ARM OP-CODES\OpCodes.uvproj - µVision4. The main window contains several panes:

- Registers** pane: Shows the current register values, with R15 (PC) highlighted at 0x0000002C.
- Disassembly** pane: Displays assembly code with annotations. The first instruction is LDR R5, [R3, R2]. The second instruction is LDR R5, [R3, -R2], annotated with ";with -ve index". The third instruction is LDR R5, [R3, R2, ASR #1], annotated with ";with index and scaling". The fourth instruction is LDR R5, [R3, -R2, ASR #1], annotated with ";with -ve index and scaling". The fifth instruction is LDR R5, [R3, -R2, ASR #1], annotated with ";with offset and autoindexing pre-update". A blue callout bubble points to the second instruction.
- OpCodes.s** pane: A list of ARM instructions numbered 19 to 36, each with its assembly code and a descriptive annotation.
- Memory 1** pane: A hex dump of memory starting at address 0x76. The first few bytes are 00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00.
- Command** pane: Shows compiler messages: "*** Restricted Version with 32768 Byte Code Size Limit" and "*** Currently used: 584 Bytes (1%)".
- Bottom status bar**: Includes Simulation, t1: 0.0000000 sec, L:24 C:1, CAP NUM, and a small icon.

0x00000080

ARM Addressing Modes

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current values of various registers. R5 is highlighted with a blue selection bar and has a value of 0x00000020.
- Disassembly:** Displays the assembly code for the current program. The highlighted instruction is at address 0x00000030, which is LDR r5, [r3, -r2, ASR #1]. This is annotated with a blue thought bubble asking "What is the value to be stored in R5?"
- OpCodes.s:** Shows the assembly code for the entire file. The highlighted instruction is at address 26, which is LDR r5, [r3, r2, ASR #1]. This is annotated with a yellow speech bubble showing the value "0x00000080".
- Memory 1:** Shows the memory dump starting at address 0x00000076. The value at address 0x00000076 is 0x00, and the value at address 0x00000080 is 0x80.
- Command:** Displays compiler messages about code size and memory usage.

ARM Addressing Modes

What is the value to be stored in R5?

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current values of various registers. R5 is highlighted with a blue selection bar and has a value of 0x00000050.
- Disassembly:** Displays several LDR instructions. The instruction at address 0x00000034 is highlighted with a yellow selection bar and has the following assembly:


```
0x00000034 E71350C2 LDR R5, [R3,-R2,ASR #1] ;with -ve index and scaling
29:          LDR r5, [r3,#4]! ;with offset and autoindexing pre-update
```
- OpCodes.s:** Shows a list of LDR instructions with comments explaining the addressing modes. The instruction at address 0x00000034 is highlighted with a yellow selection bar:


```
22
23     LDR r5, [r3,r2]           ;with index
24     LDR r5, [r3,-r2]          ;with -ve index
25
26     LDR r5, [r3,r2,ASR #1]   ;with index and scaling
27     LDR r5, [r3,-r2,ASR #1]  ;with -ve index and scaling
28
29     LDR r5, [r3,#4]!         ;with offset and autoindexing pre-update
30     LDR r5, [r3,-8]!         ;with -ve offset and autoindexing pre-update
31     LDR r5, [r3, #(-4*2+40)/4]! ;The offset can be constant expression
32                               ;to be evaluated by the assembler
33
34     LDR r5, [r3,r2]!         ;with index and autoindexing pre-update
35     LDR r5, [r3,-r2]!        ;with -ve index and autoindexing pre-update
36
37     LDR r5, [r3,r2,ASR #1]!  ;with index, scaling, and autoindexing pre-update
38     LDR r5, [r3,-r2,ASR #1]! ;with -ve index, scaling, and autoindexing pre-update
39
```
- Memory 1:** Shows the memory dump starting at address 0x00000076. The value at address 0x00000076 is 0x00000080, indicated by a blue speech bubble.
- Command:** Displays compiler messages about code size and memory usage.

ARM Addressing Modes

What are the values to be stored in R3 and R5?

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current values of various ARM registers. R5 is highlighted with a blue selection bar and has a value of 0x00000030. R15 (PC) is also highlighted and has a value of 0x00000038.
- Disassembly:** Displays ARM assembly code. Instruction 29: LDR r5, [r3, #4]! is highlighted with a blue selection bar. A blue speech bubble next to it asks, "What are the values to be stored in R3 and R5?".
- OpCodes.s:** Shows the source assembly file with the same code.
- Memory 1:** Shows memory starting at address 0x00000076. The address field is set to 0x76. A yellow speech bubble points to the memory dump area showing the byte sequence: 0x00 10 00 00 00 20 00 00 30 00 00 00 40 00 00 50 00 00 00 60 00 00 00 00 00 D0. The address 0x00000080 is also highlighted in the memory dump.
- Command:** Displays compiler messages: "*** Restricted Version with 32768 Byte Code Size Limit" and "*** Currently used: 584 Bytes (1%)".

ARM Addressing Modes

What are the values to be stored in R3 and R5?

The screenshot shows the µVision4 IDE interface with the following windows:

- Registers**: Shows the current register values, with R3 and R5 highlighted.
- Disassembly**: Shows the assembly code for the current program. The instruction at address 0x0000003C is highlighted: LDR r5, [r3, #-0x0008]!. A blue callout bubble points to this instruction with the question "What are the values to be stored in R3 and R5?".
- OpCodes.s**: Shows a list of LDR instructions from addresses 24 to 41, illustrating various addressing modes.
- Memory 1**: Shows a memory dump starting at address 0x00000076. The value at 0x00000076 is highlighted with a blue callout bubble containing the value "0x00000080".
- Command**: Displays compiler messages about code size and memory usage.
- Call Stack + Locals**: Shows the call stack and local variables.

```

29:           LDR   r5, [r3,#4]!      ;with    offset and autoindex
0x00000038 E5B35004 LDR   R5,[R3,#0x0004]!
30:           LDR   r5, [r3,#-8]!     ;with -ve offset and autoindex
→0x0000003C E5335008 LDR   R5,[R3,#-0x0008]!  ;with -ve index
31:           LDR   r5, [r3,#(-4*2+40)/4]! ;The offset can be constant expression
32:                           ;to be evaluated by the assembler
33:
34:           LDR   r5, [r3,r2]!      ;with    index and autoindexing pre-update
35:           LDR   r5, [r3,-r2]!    ;with -ve index and autoindexing pre-update
0x00000040 E5B35008 LDR   R5,[R3,#0x0008]!
36:           LDR   r5, [r3,r2]!      ;with    index and autoindexing pre-update
37:           LDR   r5, [r3,-r2]!    ;with -ve index and autoindexing pre-update
38:           LDR   r5, [r3,r2,ASR #1]! ;with    index, scaling, and autoindexing pre-update
39:           LDR   r5, [r3,-r2,ASR #1]! ;with -ve index, scaling, and autoindexing pre-update
40:           LDR   r5, [r3],#4        ;with    offset and autoindexing post-update
41:           LDR   r5, [r3],#-8       ;with -ve offset and autoindexing post-update

```

ARM Addressing Modes

What are the values to be stored in R3 and R5?

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current values of various registers, with R3 and R5 highlighted.
- Disassembly:** Displays ARM assembly code. The highlighted instruction is LDR R5, [R3, #0x0008]!. A callout bubble points to the offset #0x0008, asking about its value.
- OpCodes.s:** A list of LDR instructions with their descriptions. The highlighted instruction is LDR r5, [r3, #(−4*2+40)/4]!. A callout bubble points to the offset #(−4*2+40)/4, asking about its value.
- Memory 1:** A hex dump of memory starting at address 0x00000076. The value at 0x0000008F is 0x00000080, which is highlighted in yellow and has a callout bubble pointing to it.

```

29:           LDR r5, [r3,#4]!      ;with      offset and autoindexing p
0x00000038 E5B35004 LDR      R5,[R3,#0x0004]!
30:           LDR r5, [r3,#-8]!     ;with -ve offset and autoindexin
0x0000003C E5335008 LDR      R5,[R3,#-0x0008]!
31:           LDR r5, [r3,#(-4*2+40)/4]! ;The offset can be constant ex
32:                           ;to be evaluated by the assembler
33:
34:           LDR r5, [r3,r2]!      ;with      index and autoindexing p
0x00000040 E5B35008 LDR      R5,[R3,R2]!
0x00000044 E7B35002 LDR      R5,[R3,R2]!

26:           LDR r5, [r3,r2,ASR #1]    ;with      index and scaling
27:           LDR r5, [r3,-r2,ASR #1]   ;with -ve index and scaling
28:
29:           LDR r5, [r3,#4]!        ;with      offset and autoindexing pre-update
30:           LDR r5, [r3,#-8]!       ;with -ve offset and autoindexing pre-update
31:           LDR r5, [r3,#(-4*2+40)/4]! ;The offset can be constant expression
32:                           ;to be evaluated by the assembler
33:
34:           LDR r5, [r3,r2]!        ;with      index and autoindexing pre-update
35:           LDR r5, [r3,-r2]!       ;with -ve index and autoindexing pre-update
36:
37:           LDR r5, [r3,r2,ASR #1]!  ;with      index, scaling, and autoindexing pre-update
38:           LDR r5, [r3,-r2,ASR #1]! ;with -ve index, scaling, and autoindexing pre-update
39:
40:           LDR r5, [r3],#4         ;with      offset and autoindexing post-update
41:           LDR r5, [r3],#-8        ;with -ve offset and autoindexing post-update
42:           LDR r5, [r3],#(-4*2+40)/4 ;The offset can be constant expression
43:                           ;to be evaluated by the assembler

```

ARM Addressing Modes

What are the values to be stored in R2, R3 and R5?

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current register values. R3 is highlighted in blue.
- Disassembly:** Shows the assembly code with comments explaining the addressing modes. The highlighted instruction is LDR r5,[r3,r2]!.
- OpCodes.s:** Shows the assembly code with comments explaining the addressing modes.
- Memory 1:** Shows the memory dump at address 0x76. The value at 0x76 is 0x00000080.

```

29:    LDR r5,[r3,#4]!           ;with      offset and autoindexing pre-update
30:    LDR r5,[r3,-8]!          ;with -ve offset and autoindexing pre-update
31:    LDR r5,[r3, #-0x0008]!   ;The offset can be constant expression
32:                                ;to be evaluated by the assembler
33:
34:    LDR r5,[r3,r2]!          ;with      index and autoindexing pre-update
35:    LDR r5,[r3,-r2]!         ;with -ve index and autoindexing pre-update
36:
37:    LDR r5,[r3,r2,ASR #1]!   ;with      index, scaling, and autoindexing pre-update
38:    LDR r5,[r3,-r2,ASR #1]!  ;with -ve index, scaling, and autoindexing pre-update
39:
40:    LDR r5,[r3],#4            ;with      offset and autoindexing post-update
41:    LDR r5,[r3],#-8           ;with -ve offset and autoindexing post-update
42:    LDR r5,[r3],#(-4*2+40)/4 ;The offset can be constant expression
43:                                ;to be evaluated by the assembler
44:
45:    LDR r5,[r3],r2             ;with      index and autoindexing post-update
46:    LDR r5,[r3],-r2            ;with -ve index and autoindexing post-update

```

ARM Addressing Modes

What are the values to be stored in R2, R3 and R5?

The screenshot shows the µVision4 IDE interface with the following windows:

- Registers**: Shows the current register values. R3 is highlighted with a blue box and has a value of 0x0000008C.
- Disassembly**: Shows assembly code with annotations. A yellow arrow points to the instruction at address 0x00000048. The assembly code includes comments like ";with index and autoindexing pre-update" and ";with -ve index and autoindexing pre-update".
- OpCodes.s**: Shows a list of assembly instructions with their addresses and descriptions.
- Memory 1**: Shows a memory dump starting at address 0x76. The value at 0x76 is 0x00000080, which is highlighted with a blue box and labeled "0x00000080".
- Command**: Displays compiler messages about code size and memory usage.

ARM Addressing Modes

The screenshot shows the µVision4 IDE interface with the following components:

- Registers** window: Shows the current values of various registers. The R2, R3, and R5 registers are highlighted in blue.
- Disassembly** window: Displays ARM assembly code. Lines 34, 35, 37, and 38 are highlighted in yellow. A large blue speech bubble points to the R2, R3, and R5 registers in the Registers window, asking: "What are the values to be stored in R2, R3 and R5?"
- OpCodes.s** window: Shows the assembly code being assembled. Line 37 is highlighted in green, indicating it is the current instruction being evaluated.
- Memory 1** window: Shows a memory dump starting at address 0x00000076. The value at address 0x00000076 is 0x00000080, indicated by a blue speech bubble.
- Command** window: Displays compiler messages about restricted version and code size.

```

34:           LDR r5,[r3,r2]!          ;with index and autoindexing pre
0x00000044 E7B35002 LDR      R5,[R3,R2]!
35:           LDR r5,[r3,-r2]!        ;with -ve index and autoindexin
36:
37:           LDR r5,[R3,-R2]!        ;with index, scaling, and au
0x0000004C E7B3502 LDR      R5,[R3,R2,ASR #1]!
38:           LDR r5,[r3,r2,ASR #1]!  ;with -ve index, scaling, and auto
39:
0x00000050 E733502 LDR      R5,[R3,-R2,ASR #1]!

```

What are the
values to be
stored in
R2, R3 and R5?

0x00000080

ARM Addressing Modes

C:\Users\Mahmoud El-Sakka\Desktop\ARM\ARM OP-CODES\OpCodes.uvproj - µVision4

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Registers Disassembly

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000008
R3	0x00000088
R4	0x00000040
R5	0x00000060
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13(SP)	0x00000000
R14(LR)	0x00000000
R15(PC)	0x00000050
CPSR	0x000000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000050
Mode	Supervisor
States	48
Sec	0.00000000

OpCodes.s

```

33
34      LDR r5,[r3,r2]!           ;with    index and autoindexing pre-update
35      LDR r5,[r3,-r2]!          ;with -ve index and autoindexing pre-
36
37      LDR r5,[r3,r2,ASR #1]!   ;with    index, scaling, and a
38      LDR r5,[r3,R2,ASR #1]!   ;with -ve index, scale and auto
39
40      LDR r5,[r3],#4           ;with    offset and autoindexing post-update
41      LDR r5,[r3],#-8          ;with -ve offset and autoindexing post-update
42      LDR r5,[r3],#(-4*2+40)/4 ;The offset can be constant expression
                                ;to be evaluated by the assembler
43
44      LDR r5,[r3],r2            ;with    index and autoindexing post-update
45      LDR r5,[r3],-r2           ;with -ve index and autoindexing post-update
46
47      LDR r5,[r3],r2,LSL #1    ;with    index, scaling, and autoindexing post-update
48      LDR r5,[r3],-r2,LSL #1   ;with -ve index, scaling, and autoindexing post-update
49
50

```

Command

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 584 Bytes (1%)

```

Memory 1

Address:	0x76
0x00000076:	00 10 00 00 00 20 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00
0x0000008F:	70 00 00 00 80 00 00 00 90 00 00 00 A0 00 00 00 B0 00 00 00 C0 00 00 D0
0x000000A8:	E2 8F 00 18 E3 A0 10 05 E3 A0 20 00 E2 51 10 01 E7 90 31 01 E0 82 20 03 1A

Call Stack + Locals Memory 1 Simulation t1: 0.00000000 sec L:38 C:1 CAP NUM \$

What are the values to be stored in R2, R3 and R5?

0x00000080

ARM Addressing Modes

What are the values to be stored in R3 and R5?

The screenshot shows the µVision4 IDE interface with the following components:

- Registers** window: Shows the current register values. R3 is highlighted with a blue selection bar.
- Disassembly** window: Displays assembly code with annotations explaining addressing modes. Lines 40 and 41 are highlighted with yellow boxes and blue circles pointing to specific fields. Line 41 is annotated with "with -ve offset and autoindexing". Line 42 is annotated with "The offset can be constant expression to be evaluated by the assembler".
- OpCodes.s** window: Shows a list of assembly opcodes with their descriptions.
- Memory 1** window: Shows a hex dump of memory starting at address 0x00000076. The value at 0x0000008F is highlighted with a blue box and labeled "0x00000080".
- Command** window: Displays build messages, including "Restricted Version with 32768 Byte Code Size Limit" and "Currently used: 584 Bytes (1%)".

ARM Addressing Modes

The screenshot shows the µVision4 IDE interface with the following components:

- Registers** window: Shows the current values of various registers. R3 is highlighted with a blue selection bar and has a value of 0x00000080.
- Disassembly** window: Displays ARM assembly code. The instruction at address 0x00000058 is highlighted in yellow: LDR R5, [R3], #-0x0008. A blue callout bubble asks: "What are the values to be stored in R3 and R5?"
- OpCodes.s** window: Shows a list of assembly opcodes and their descriptions.
- Memory 1** window: Shows a memory dump starting at address 0x00000076. The value at address 0x00000076 is 0x00000080, indicated by a blue callout bubble.
- Command** window: Displays compiler messages about code size and memory usage.

What are the
values to be
stored in
R3 and R5?

0x00000080

ARM Addressing Modes

What are the values to be stored in R3 and R5?

The screenshot shows the µVision4 IDE interface with the following details:

- Registers:** Shows the current values of various registers. R3 is highlighted with a blue selection bar and has a value of 0x00000080. R5 is also highlighted with a blue selection bar and has a value of 0x00000060.
- Disassembly:** The assembly code window displays several LDR instructions. Instruction 39: LDR R5, [R3, -R2, ASR #1]! is highlighted in yellow. Instruction 40: LDR r5, [r3], #4 is annotated with ";with offset and autoindexing pre-update". Instruction 41: LDR r5, [r3], #-8 is annotated with ";with -ve offset and autoindexing post-update". Instruction 42: LDR r5, [r3], #(-4*2+40)/4 is annotated with ";The offset can be constant expression to be evaluated by the assembler". Instruction 44: LDR R5, [R3], #0x0008 is highlighted in yellow.
- OpCodes.s:** The source code file contains the assembly code shown in the disassembly window.
- Memory 1:** A dump of memory starting at address 0x76. The address field is set to 0x76. The memory dump shows three 32-bit words starting at 0x00000076: 00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00 00 00 00 00. Below these are addresses 0x0000008F and 0x000000A8, each followed by a series of zeros.

ARM Addressing Modes

What are the values to be stored in R2, R3 and R5?

Registers

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000008
R3	0x00000088
R4	0x00000040
R5	0x00000040
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13(SP)	0x00000000
R14(LR)	0x00000000
R15(PC)	0x00000060
CPSR	0x000000D3
SPSR	0x00000000

Disassembly

```

44: 0x0000005C E4935008 LDR    R5,[R3],#0x0008
45: 0x00000060 E6935002 LDR    R5,[R3],R2      ;with index and autoindexing
46:          LDR    R5,[r3],-r2    ;with -ve index and autoindexing
47:
48: 0x00000064 E6135002 LDR    R5,[R3],-R2
49:          LDR    R5,[r3],r2,LSL #1   ;with index, scaling, and autoindexing post-update
50: 0x00000068 E6935082 LDR    R5,[R3],R2,LSL #1
51:          LDR    R5,[r3],-r2,LSL #1   ;with -ve index, scaling, and autoindexing post-update
52: loop     B     loop
53: AAA
54: DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
      DCD 2_1110000,2_10000000,0x90,0xA0,0xB0,0xC0,0xD0

```

OpCodes.s

```

37: LDR r5,[r3,r2,ASR #1]!    ;with index, scaling, and autoindexing pre-update
38: LDR r5,[r3,-r2,ASR #1]!    ;with -ve index, scaling, and autoindexing pre-update
39:
40: LDR r5,[r3],#4            ;with offset and autoindexing post-update
41: LDR r5,[r3],#-8           ;with -ve offset and autoindexing post-update
42: LDR r5,[r3],#(-4*2+40)/4 ;The offset can be constant expression
43:                           ;to be evaluated by the assembler
44:
45: LDR r5,[r3],r2            ;with index and autoindexing post-update
46: LDR r5,[r3],-r2           ;with -ve index and autoindexing post-update
47:
48: LDR r5,[r3],r2,LSL #1    ;with index, scaling, and autoindexing post-update
49: LDR r5,[r3],-r2,LSL #1    ;with -ve index, scaling, and autoindexing post-update
50:
51:
52: loop     B     loop
53: AAA
54: DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
      DCD 2_1110000,2_10000000,0x90,0xA0,0xB0,0xC0,0xD0

```

Memory 1

Address	Value
0x00000076	00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00
0x0000008F	70 00 00 00 80 00 00 00 90 00 00 00 A0 00 00 00 B0 00 00 00 C0 00 00 00 D0
0x000000A8	E2 8F 00 18 E3 A0 10 05 E3 A0 20 00 E2 51 10 01 E7 90 31 01 E0 82 20 03 1A

ARM Addressing Modes

What are the values to be stored in R2, R3 and R5?

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 title bar displays the project path: C:\Users\Mahmoud El-Sakka\Desktop\ARM\ARM OP-CODES\OpCodes.uvproj - µVision4. The main window contains several panes:

- Registers** pane: Shows the current register values. R3 is highlighted with a blue selection bar.
- Disassembly** pane: Displays ARM assembly code. Lines 44-49 show LDR instructions. Line 44: LDR R5, [R3], #0x0008. Line 45: LDR r5, [r3], r2 ;with index and autoindexing post-update. Line 46: LDR R5, [R3], R2. Line 47: LDR r5, [r3], -r2 ;with -ve index and autoindexing post-update. Line 48: LDR r5, [r3], r2, LSL #1 ;with index, scaling, and autoindexing post-update. Line 49: LDR r5, [r3], -r2, LSL #1 ;with -ve index, scaling, and autoindexing post-update.
- OpCodes.s** pane: Shows the source code for the assembly file. It includes comments explaining the addressing modes used in the assembly code.
- Memory 1** pane: Displays the memory dump at address 0x76. The value at 0x00000076 is 00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00.
- Command** pane: Shows compiler messages: *** Restricted Version with 32768 Byte Code Size Limit and *** Currently used: 584 Bytes (1%).

ARM Addressing Modes

What are the values to be stored in R2, R3 and R5?

Registers

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000008
R3	0x00000088
R4	0x00000040
R5	0x00000080
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000068
CPSR	0x000000D3
SPSR	0x00000000

Disassembly

```

44: E4935008 LDR    R5, [R3],#0x0008 ;with index and autoindexing post-update
45: E6935002 LDR    R5,[R3],R2          ;with -ve index and autoindexing post-update
46: E6135002 LDR    R5,[R3],-R2         ;with index, scaling, and autoindexing post-update
47: E6935082 LDR    R5,[R3],R2,LSL #1   ;with -ve index, scaling, and autoindexing post-update
48: E6935082 LDR    R5,[R3],-R2,LSL #1  ;with index, scaling, and autoindexing post-update
49: E6935082 LDR    R5,[R3],-R2,LSL #1  ;with -ve index, scaling, and autoindexing post-update

```

OpCodes.s

```

43
44
45: LDR r5,[r3],r2           ;with index and autoindexing post-update
46: LDR r5,[r3],-r2          ;with -ve index and autoindexing post-update
47
48: LDR r5,[r3],r2,LSL #1   ;with index, scaling, and autoindexing post-update
49: LDR r5,[r3],-r2,LSL #1  ;with -ve index, scaling, and autoindexing post-update
50
51
52: loop B    loop
53: AAA   DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
54: DCD 2_1110000,2_10000000,0x90,0xA0,0xB0,0xC0, 0xD0
55
56
57
58
59
60

```

Command

```

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 584 Bytes (1%)

```

Memory 1

Address	Value
0x00000076	00 10 00 00 00 20 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00
0x0000008F	70 00 00 00 80 00 00 00 90 00 00 00 A0 00 00 00 B0 00 00 00 C0 00 00 00 D0
0x000000A8	E2 8F 00 18 E3 A0 10 05 E3 A0 20 00 E2 51 10 01 E7 90 31 01 E0 82 20 03 1A

ARM Addressing Modes

The screenshot shows the µVision4 IDE interface with the following components:

- Registers** window: Shows the current values of various registers. The R2, R3, and R5 registers are highlighted in blue.
- Disassembly** window: Displays the assembly code. A yellow arrow points from the question bubble to the instruction at address 0x0000006C, which is LDR R5, [R3], -R2, LSL #1.
- OpCodes.s** window: Shows the source code corresponding to the assembly instructions.
- Memory 1** window: Displays the memory dump starting at address 0x00000076. A blue speech bubble highlights the value 0x00000080 at address 0x0000008F.

What are the values to be stored in R2, R3 and R5?

```

48:           LDR r5,[r3],r2,LSL #1    ;with      index, scaling, and auto
0x00000068 E6935082 LDR      R5,[R3],R2,LSL #1
49:           LDR r5,[r3],-r2,LSL #1  ;with -ve index, scaling, and auto
50:
51:
52: loop     B   loop
53: AAA      DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
54:           DCD 2_1110000,2_10000000,0x90,0xA0,0xB0,0xC0, 0xD0
55:

;to be evaluated by the assembler
43
44
45:           LDR r5,[r3],r2          ;with      index and autoindexing post-update
46:           LDR r5,[r3],-r2        ;with -ve index and autoindexing post-update
47
48:           LDR r5,[r3],r2,LSL #1  ;with      index, scaling, and autoindexing post-update
49:           LDR r5,[r3],-r2,LSL #1  ;with -ve index, scaling, and autoindexing post-update
50
51
52: loop     B   loop
53: AAA      DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
54:           DCD 2_1110000,2_10000000,0x90,0xA0,0xB0,0xC0, 0xD0
55
56
57
58
59
60

```

Address: 0x76

0x00000076: 00 10 00 00 00 20 00 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00 00 00 00 00
0x0000008F: 70 00 00 00 80 00 00 00 90 00 00 00 A0 00 00 00 B0 00 00 00 C0 00 00 00 D0
0x000000A8: E2 8F 00 18 E3 A0 10 05 E3 A0 20 00 E2 51 10 01 E7 90 31 01 E0 82 20 03 1A

ARM Addressing Modes

Screenshot of the µVision4 IDE showing ARM assembly code and memory dump.

Registers window:

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000008
R3	0x00000088
R4	0x00000040
R5	0x000000A0
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13(SP)	0x00000000
R14(LR)	0x00000000
R15(PC)	0x00000070
CPSR	0x000000D3
SPSR	0x00000000

Disassembly window:

```

49:          LDR r5,[r3],-r2,LSL #1    ;with -ve index, scaling, and autoindexing post-update
50:
51:
52: 0x0000006C E6135082 LDR      R5,[R3],-R2,LSL #1
53: loop     B   loop
54: AAA       DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
55:
56:
57:

```

OpCodes.s window:

```

47
48          LDR r5,[r3],r2,LSL #1    ;with      index, scaling, and autoindexing post-update
49          LDR r5,[r3],-r2,LSL #1    ;with -ve index, scaling, and autoindexing post-update
50
51
52  loop     B   loop
53  AAA       DCD 0x10,0x20,0x30,0x40,2_1010000,2_1100000
54          DCD 2_1110000,2_10000000,0x90,0xA0,0xB0,0xC0, 0xD0
55
56
57
58
59
60
61
62
63
64

```

Memory 1 window:

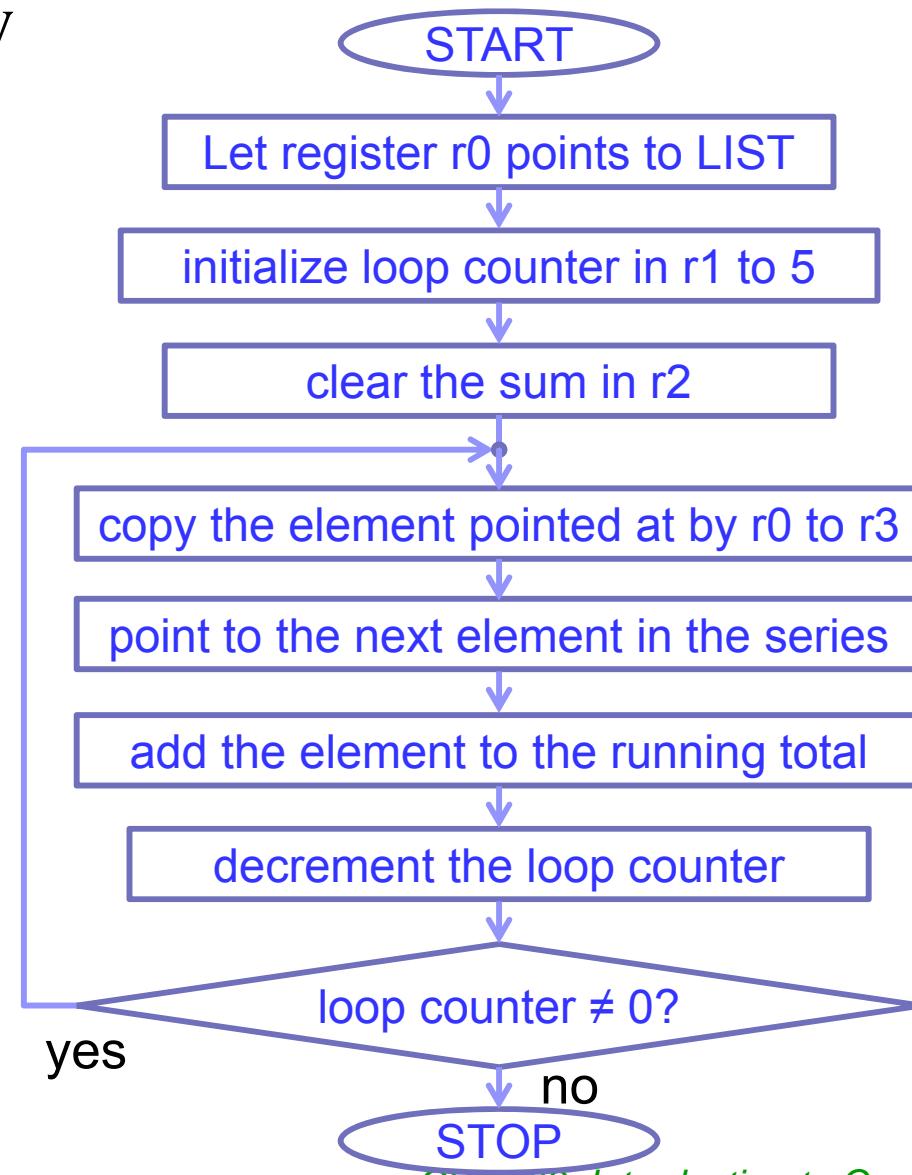
Address: 0x76

0x00000076: 00 10 00 00 00 20 00 00 30 00 00 00 40 00 00 00 50 00 00 00 60 00 00 00
0x0000008F: 70 00 00 80 00 00 00 90 00 00 00 A0 00 00 00 B0 00 00 00 C0 00 00 00 D0
0x000000A8: E2 8F 00 18 E3 A0 10 05 E3 A0 20 00 E2 51 10 01 E7 90 31 01 E0 82 20 03 1A

A blue speech bubble points to the value **0x00000088** in the Registers window for R3.

A Pointer Example

We want to add together a LIST of five numbers stored in memory



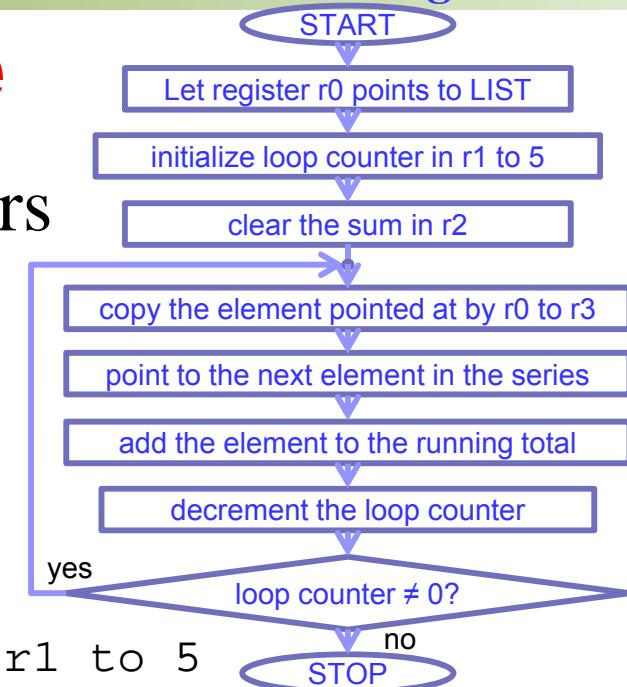
A Pointer Example

We want to add together a LIST of five numbers stored in memory

```

AREA Pointers, CODE, READONLY
ENTRY
Start    ADR  r0,List      ;register r0 points to List
          MOV  r1,#5       ;initialize loop counter in r1 to 5
          MOV  r2,#0       ;clear the sum in r2
Loop     LDR  r3,[r0]      ;copy the element pointed at by r0 to r3
          ADD  r0,r0,#4    ;point to the next element in the series
          ADD  r2,r2,r3    ;add the element to the running total
          SUBS r1,r1,#1    ;decrement to the loop counter
          BNE Loop         ;repeat until all elements added
Endless   B    Endless     ;infinite loop
List     DCD  3,4,3,6,7  ;the numbers to be added together
                  ;each one is 4 bytes (20 bytes in total)
END

```



How about if
we want to
store the
result in
memory?

A Pointer Example

We want to add together a LIST of five numbers stored in memory

```

AREA Pointers, CODE, READONLY
ENTRY
Start    ADR  r0, List      ;register r0 points to List
         MOV  r1, #5       ;initialize loop counter in r1 to 5
         MOV  r2, #0       ;clear the sum in r2
Loop     LDR  r3, [r0]      ;copy the element pointed at by r0 to r3
         ADD  r0, r0, #4   ;point to the next element in the series
         ADD  r2, r2, r3   ;add the element to the running total
         SUBS r1, r1, #1   ;decrement to the loop counter
         BNE Loop          ;repeat until all elements added
         ADR  r4, SUM
         STR  r2, [r4]
Endless  B    Endless      ;infinite loop
List     DCD  3, 4, 3, 6, 7 ;the numbers to be added together
                           ;each one is 4 bytes (20 bytes in total)
SUM      DCD  0
END

```

How can we utilize autoindexing pre-indexing mode?

How can we utilize autoindexing post-indexing mode?

You will do that in lab 3

How about if you want to store the result in memory?

You need to map the memory to allow SUM location to be read/write (Debug/Memory Map/Map Range). Need to be done every time you run the code.
The other option is to use a .ini file.