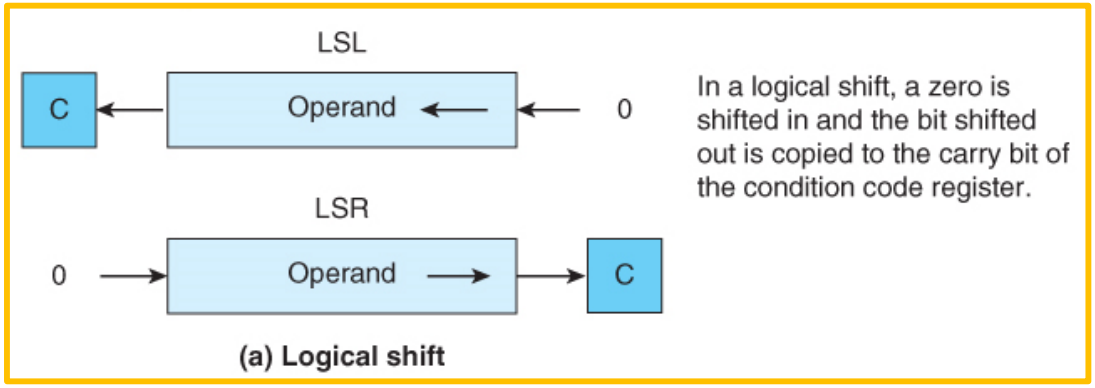
Tutorial 10: ARM Shift Instructions

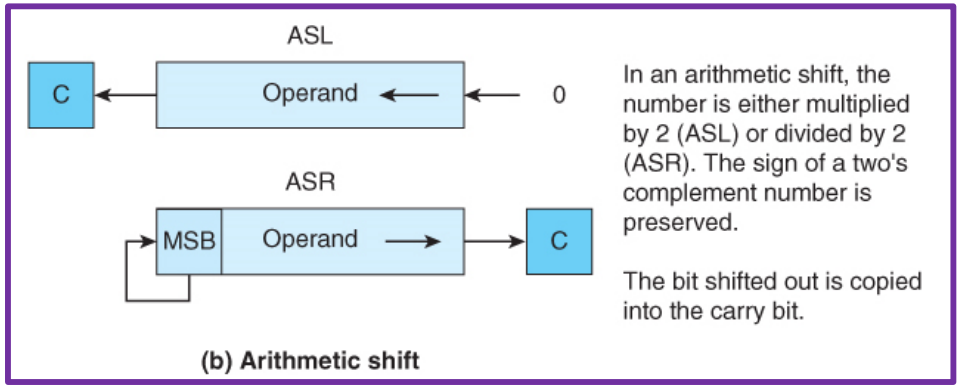
Computer Science Department
CS2208: Introduction to Computer Organization and Architecture
Fall 2022-2023
Instructor: Mahmoud R. El-Sakka
Office: MC-419
Email: elsakka@csd.uwo.ca
Phone: 519-661-2111 x86996

ARM's Data-Processing Instructions (Shift Operations)

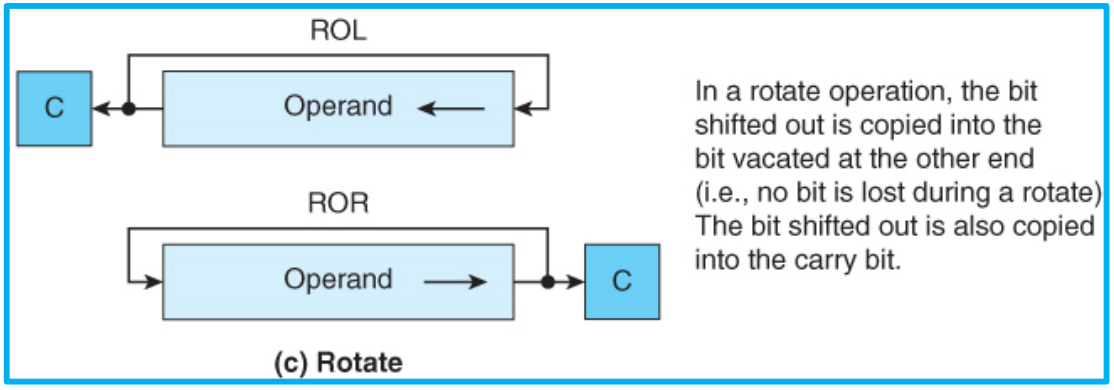
- *Shift* operations move bits one ***or more*** places *left* or *right*.
  - ***Logical shifts***
    - *insert a 0* in the vacated position.
  - ***Arithmetic shifts***
    - *replicate the sign-bit* during a right shift
  - ***Circular shifts***
    - *the bit shifted out of one end is shifted in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



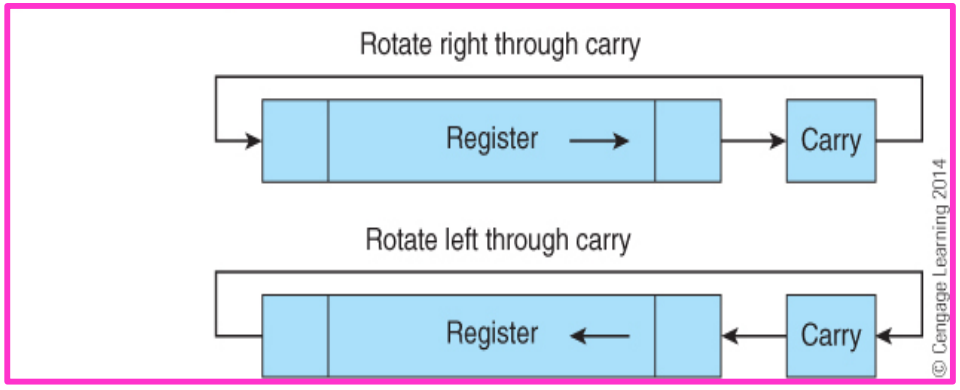
(a) Logical shift



(b) Arithmetic shift



(c) Rotate



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*
  - In *static shift*, the range of the number of shift places is as follow:
    - **LSL**: the range is from ***#0*** to ***#31 (32 different values)***
    - **LSR**: the range is from ***#1*** to ***#32 (32 different values)***
    - **ASR**: the range is from ***#1*** to ***#32 (32 different values)***
    - **ROR**: the range is from ***#1*** to ***#31 (31 different values)***

      The remaining value is used to encode RRX
      - **ROR** + ***a shift of #0*** ➔ **RRX**

LSL#0 = LSR#0 = ASR#0.
so NO #0 for LSR, ASR.

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

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

  - 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

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 × 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 ***LAST parameter in the assembly arithmetic instruction***) is allowed to be shifted ***before*** it is used.
  - For example,

    ```
    ADD r0,r1,r2,LSL #1          ;[r0] ← [r1] + [r2] × 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 MOV and MVN operations*
  - This way, a shift operation can be performed as a stand-alone operation.
  - For example,

    ```
    MOV r3,r3,LSL #1             ;[r3] ← [r3] × 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
    ```

    or simply

    ```
    LSL r3,#1
    ```

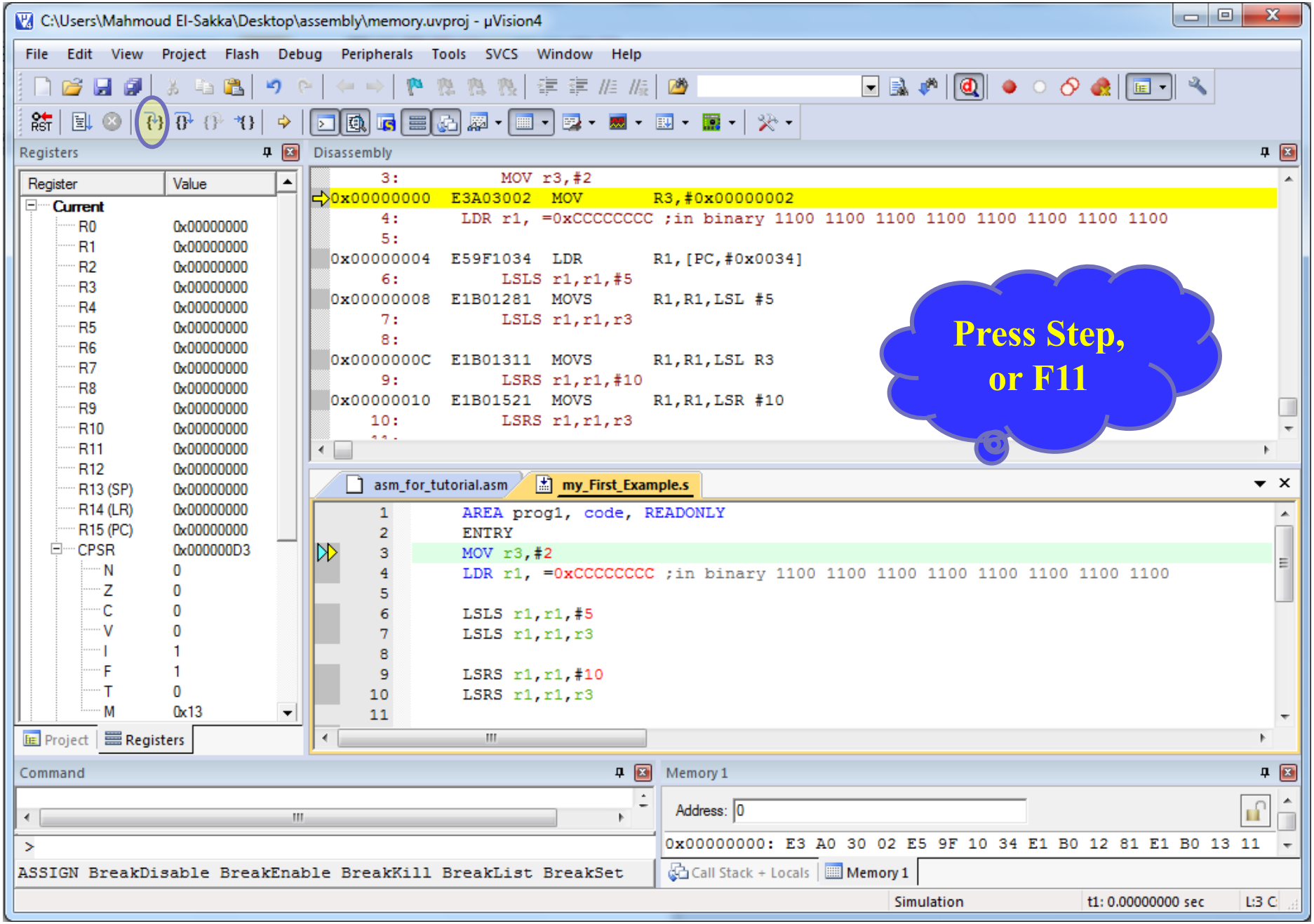
ARM's Data-Processing Instructions (Shift Operations)

```
AREA prog1, code, READONLY
ENTRY
MOV r3,#2
LDR r1, =0xCCCCCCCC ;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
```

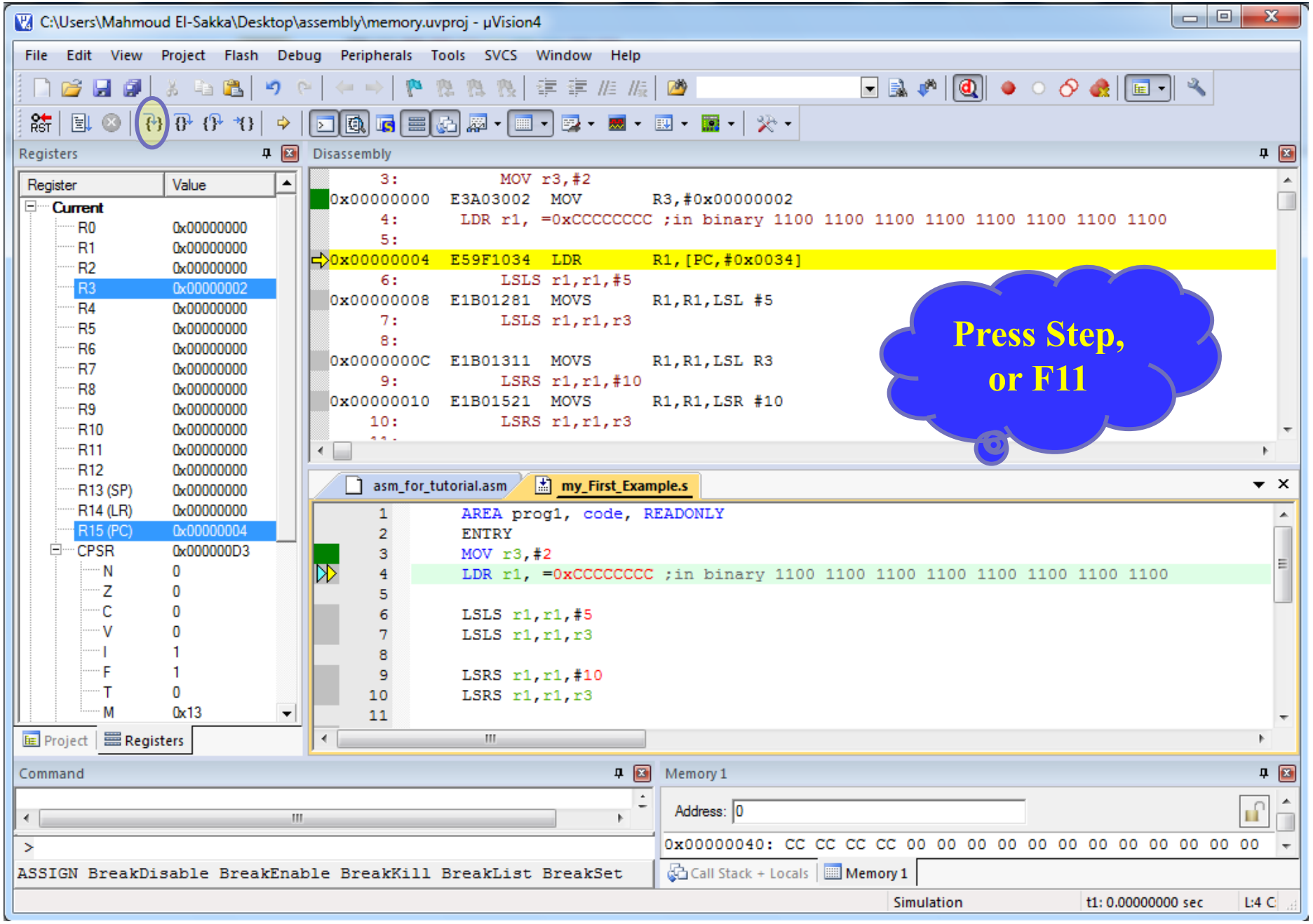
MOVS and MVNS

- ✓ *Update the N, Z and C flags according to the result*
- ✓ *Do NOT affect the V flag*

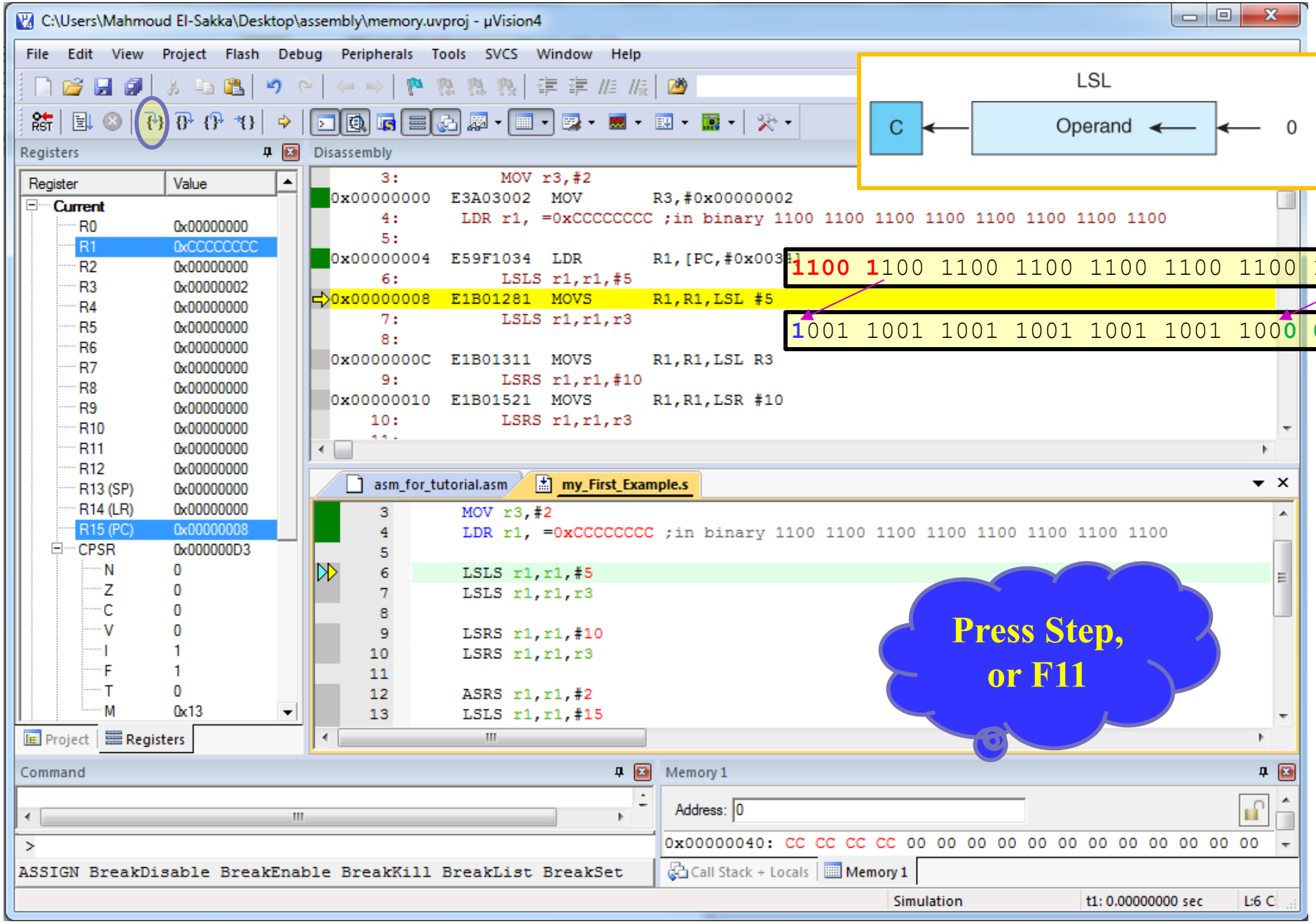
ARM's Data-Processing Instructions (Shift Operations)



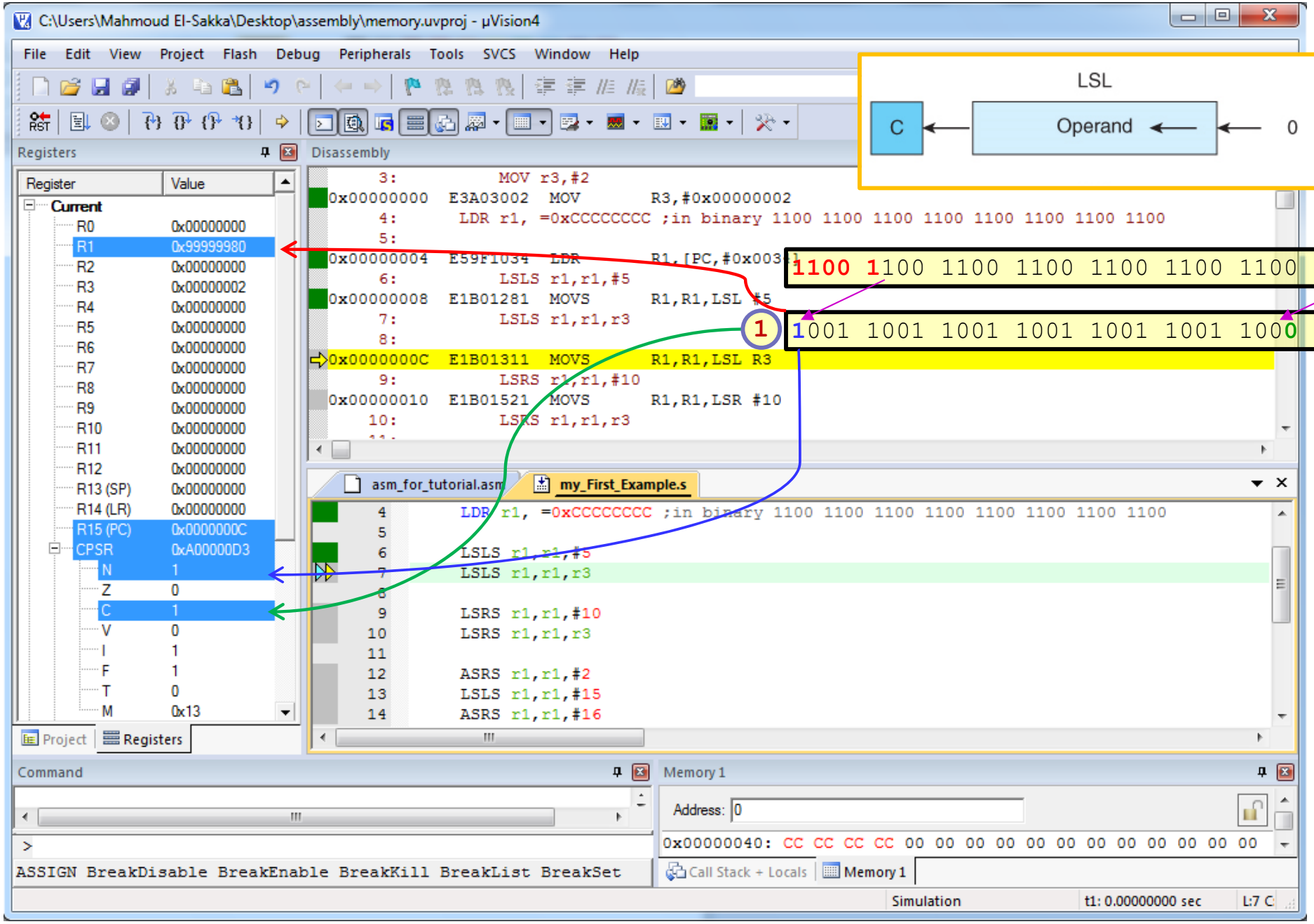
ARM's Data-Processing Instructions (Shift Operations)



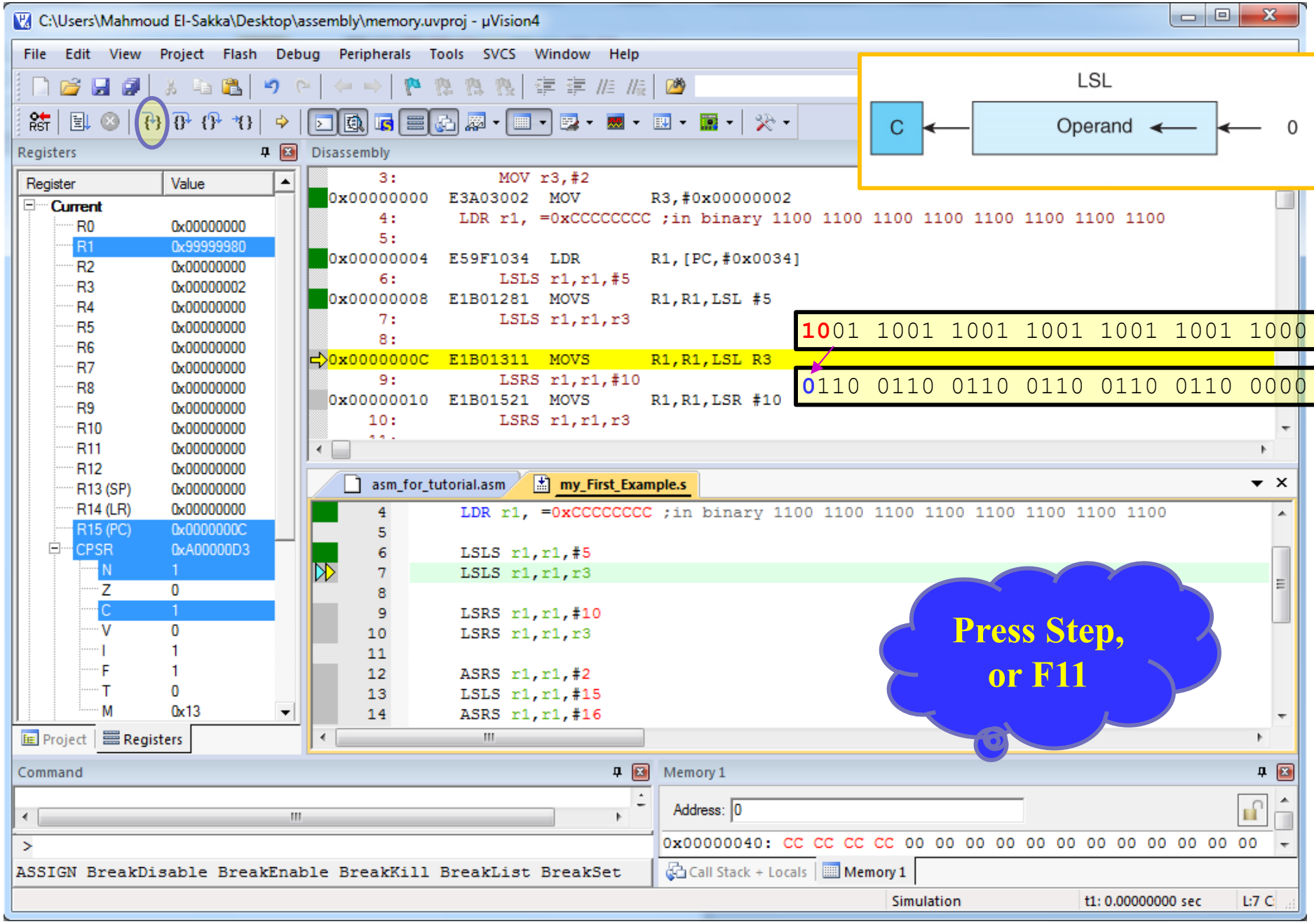
ARM's Data-Processing Instructions (Shift Operations)



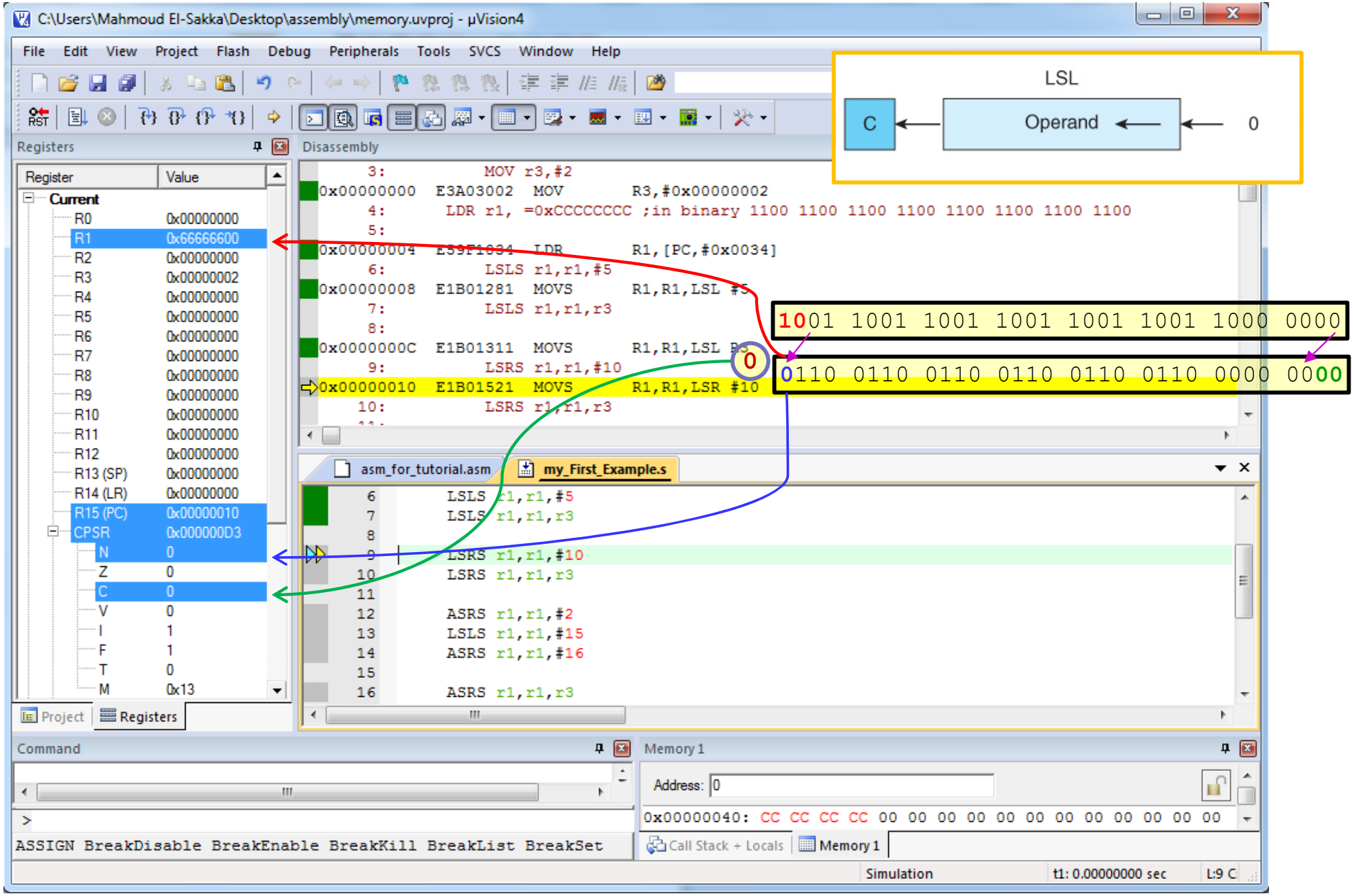
ARM's Data-Processing Instructions (Shift Operations)



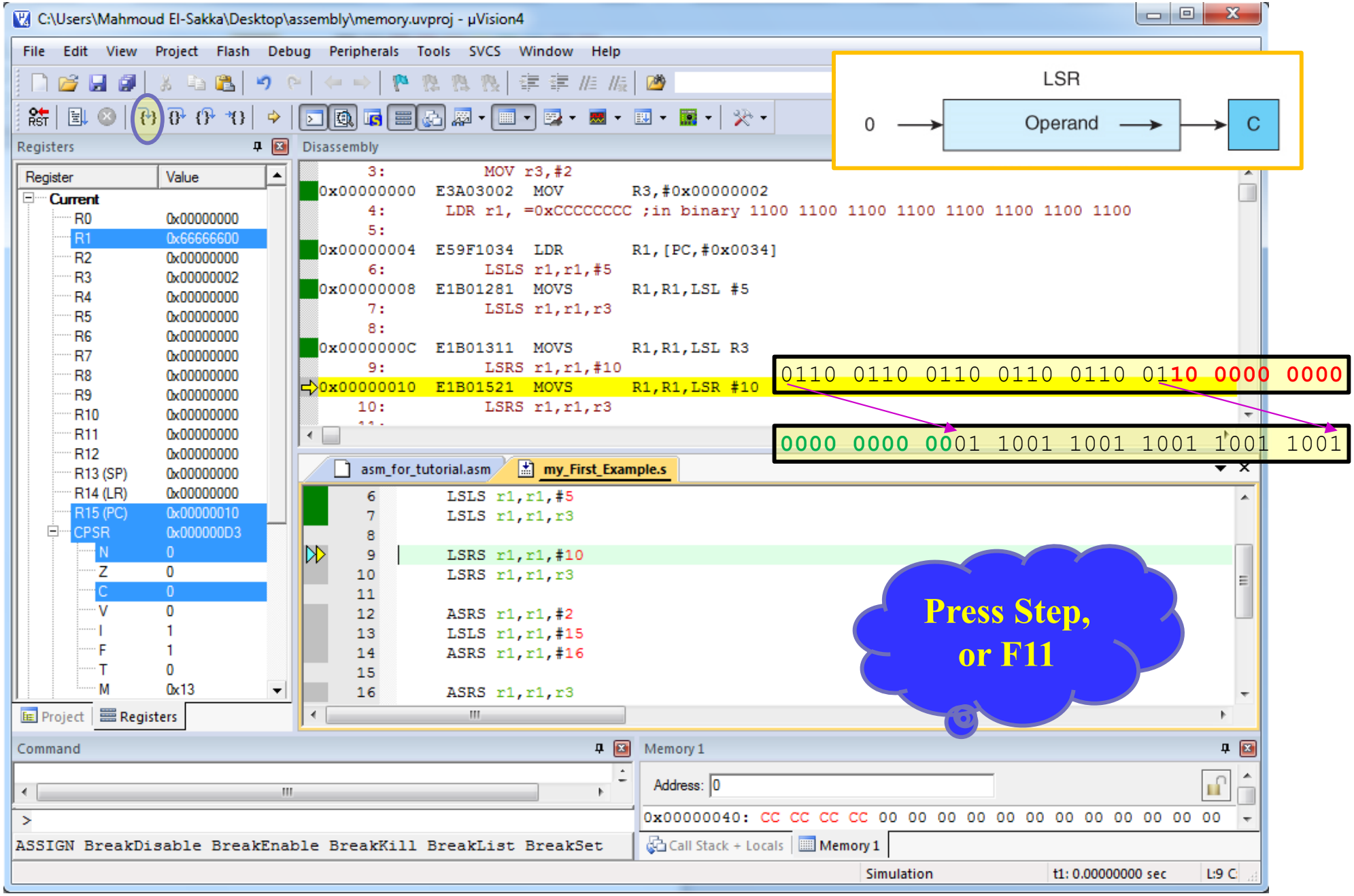
ARM's Data-Processing Instructions (Shift Operations)



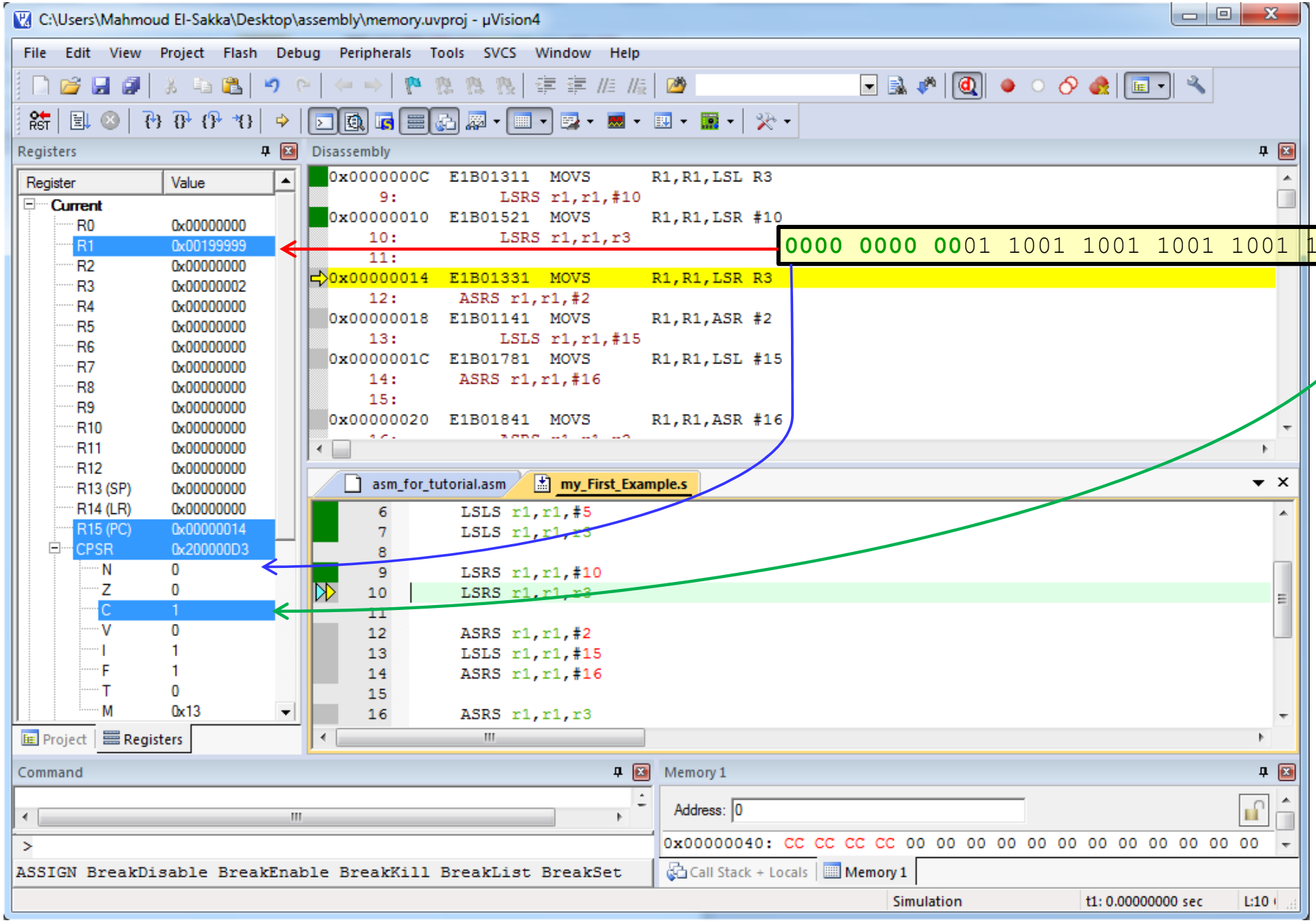
ARM's Data-Processing Instructions (Shift Operations)



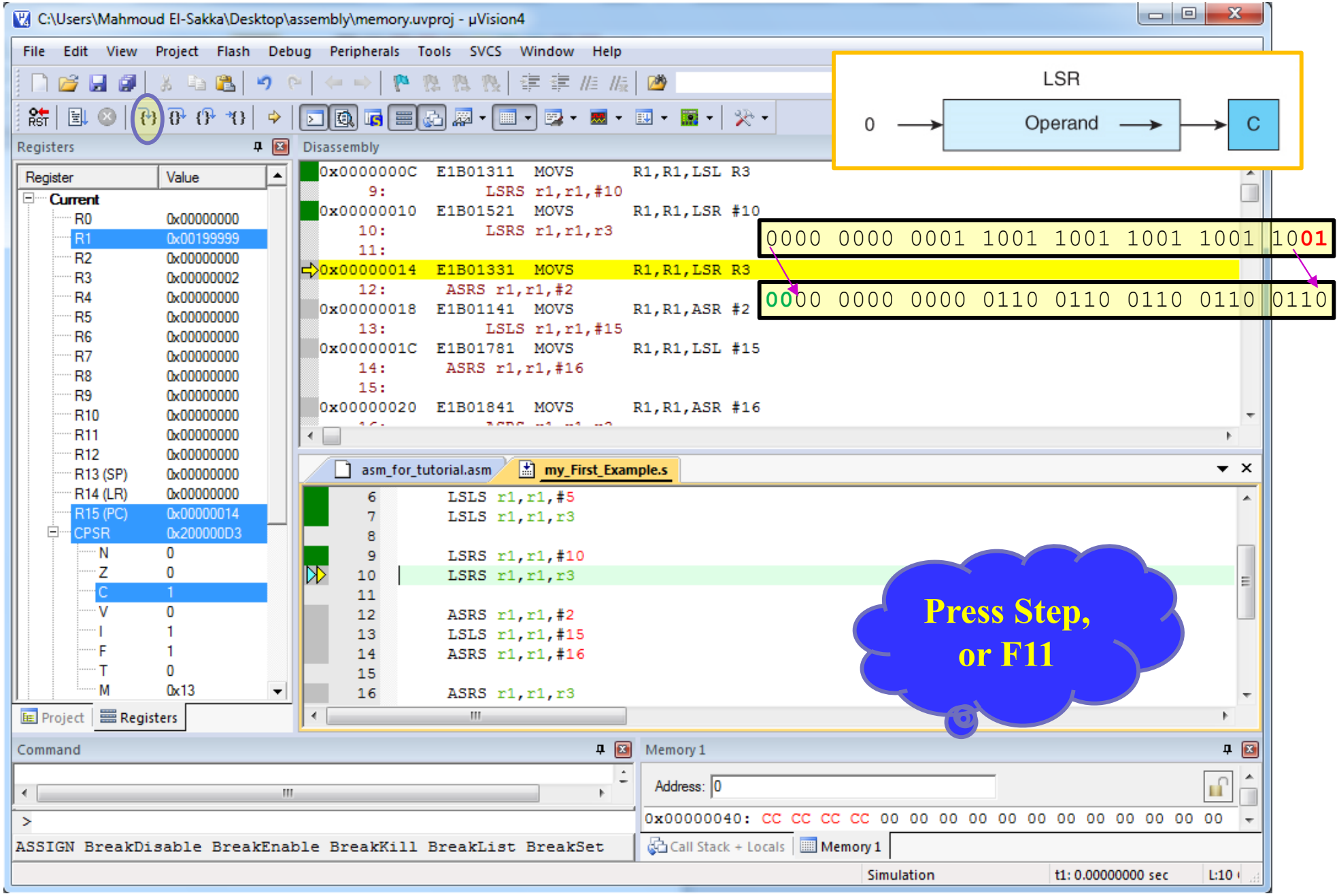
ARM's Data-Processing Instructions (Shift Operations)



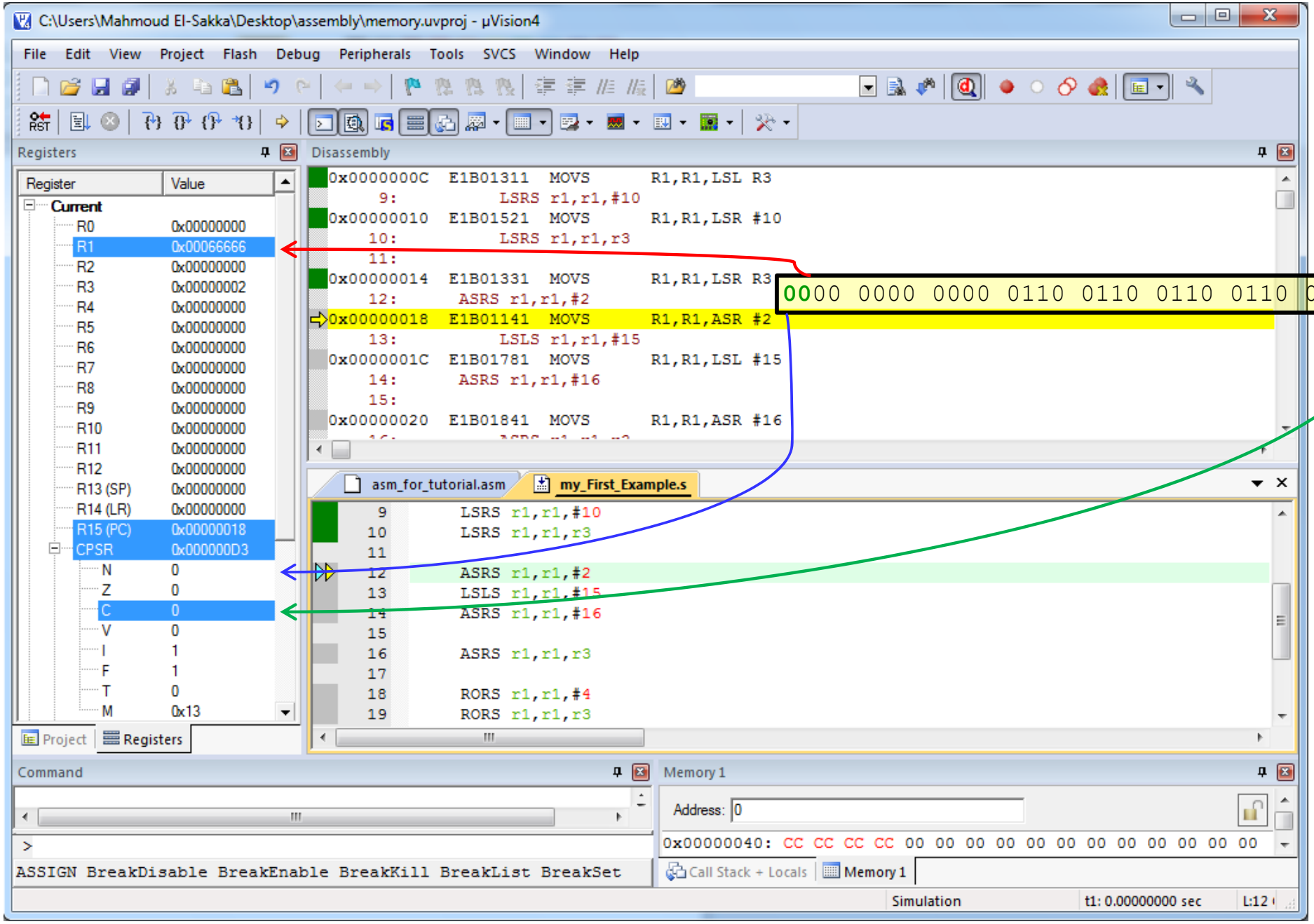
ARM's Data-Processing Instructions (Shift Operations)



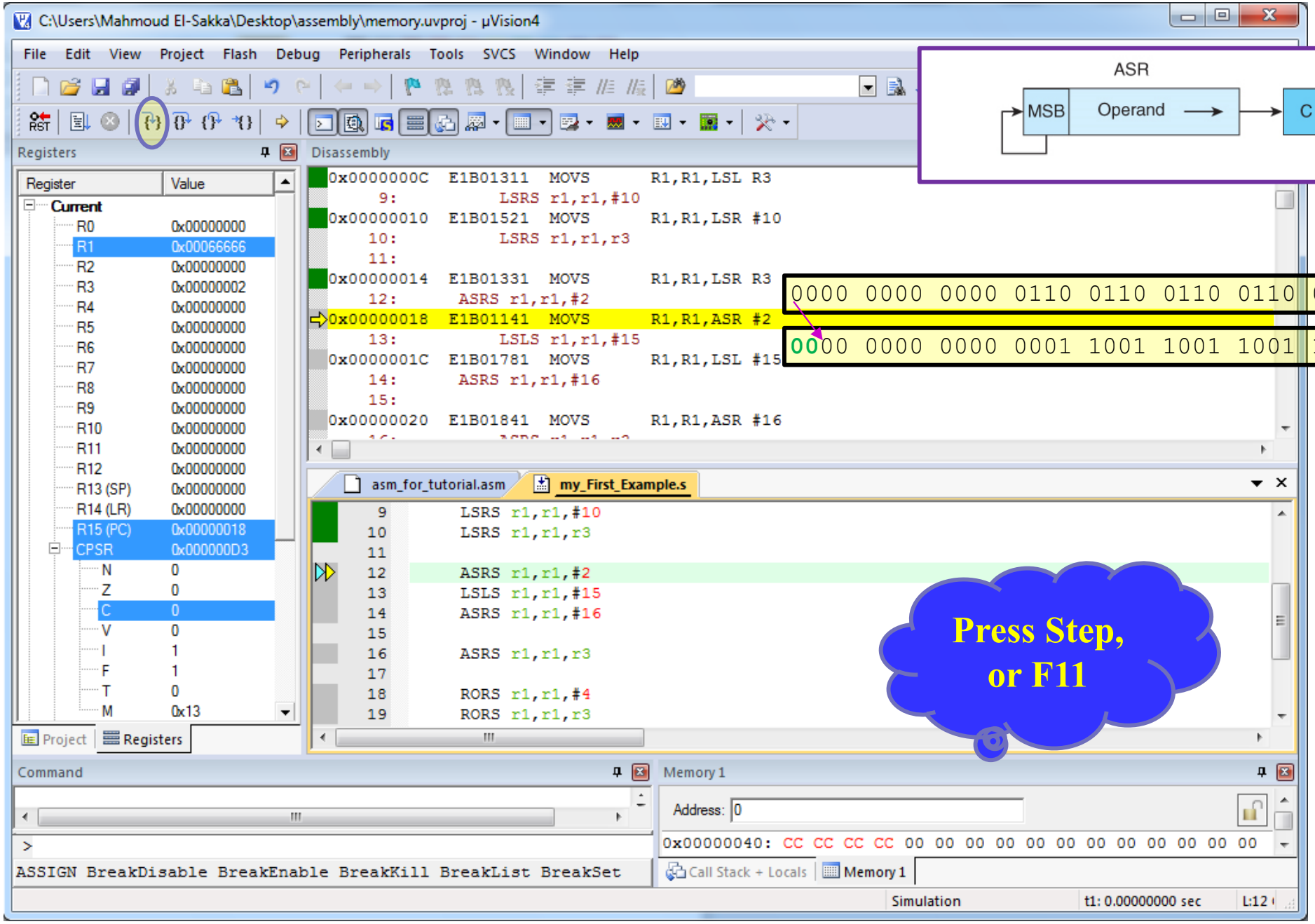
ARM's Data-Processing Instructions (Shift Operations)



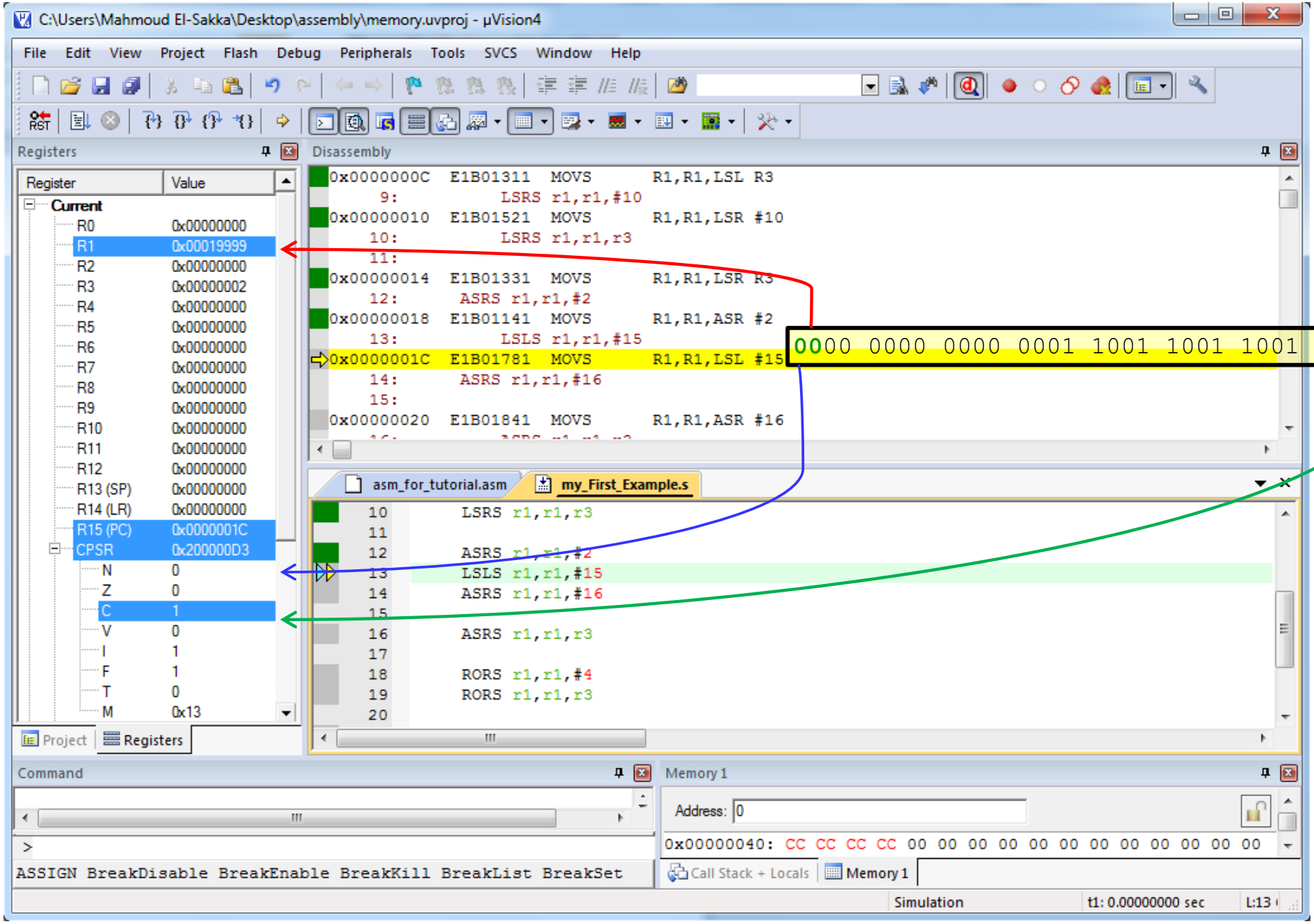
ARM's Data-Processing Instructions (Shift Operations)



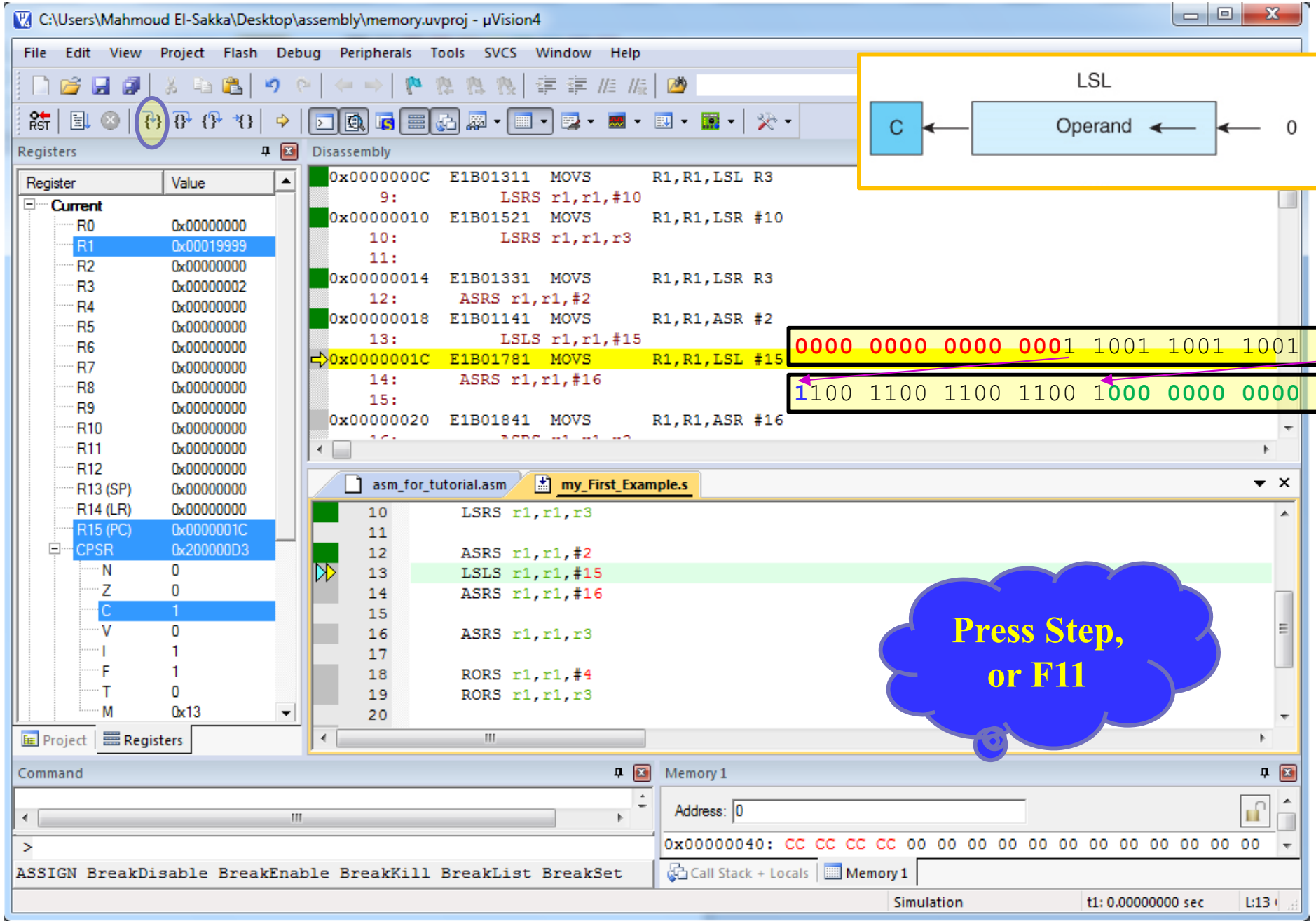
ARM's Data-Processing Instructions (Shift Operations)



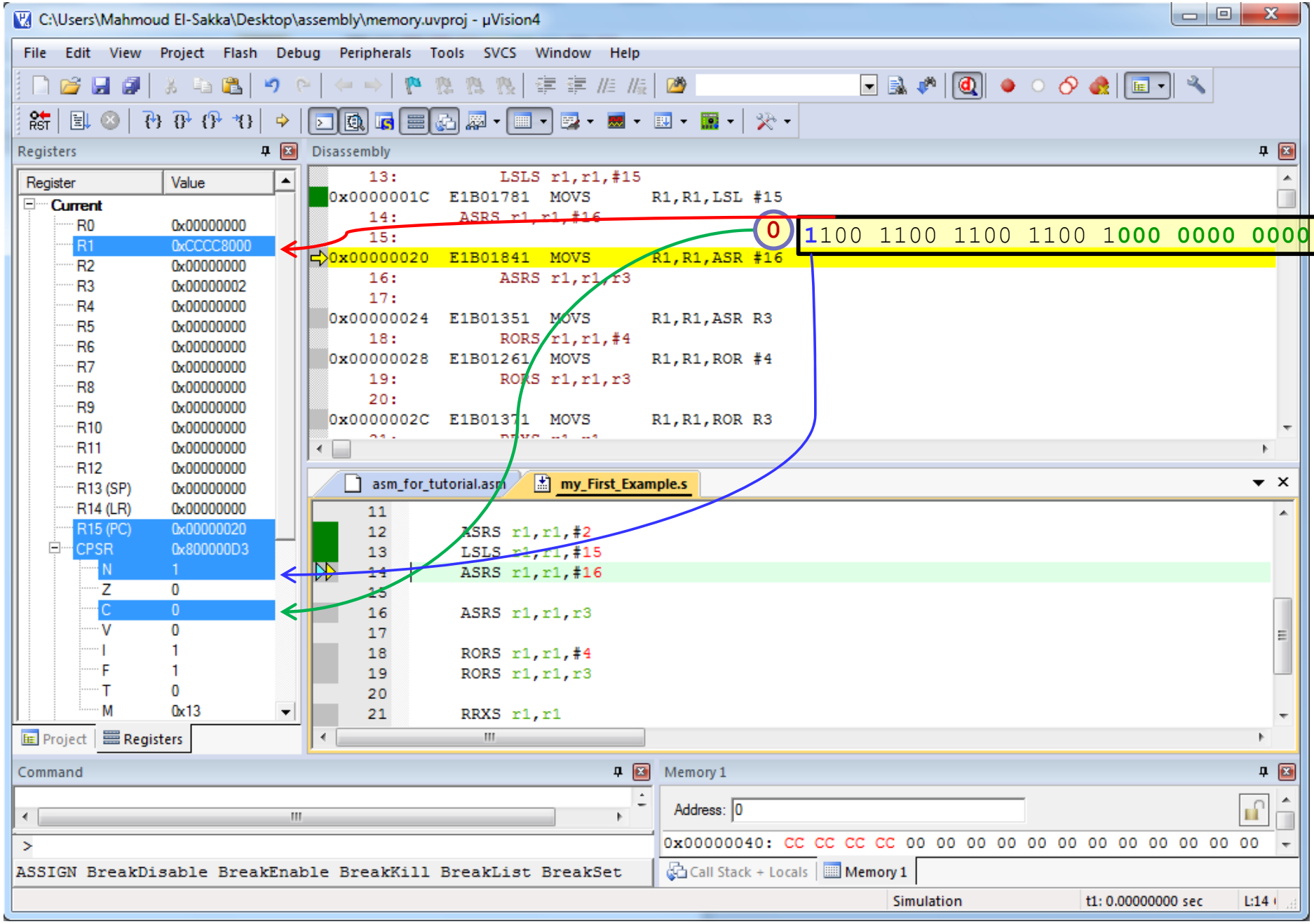
ARM's Data-Processing Instructions (Shift Operations)



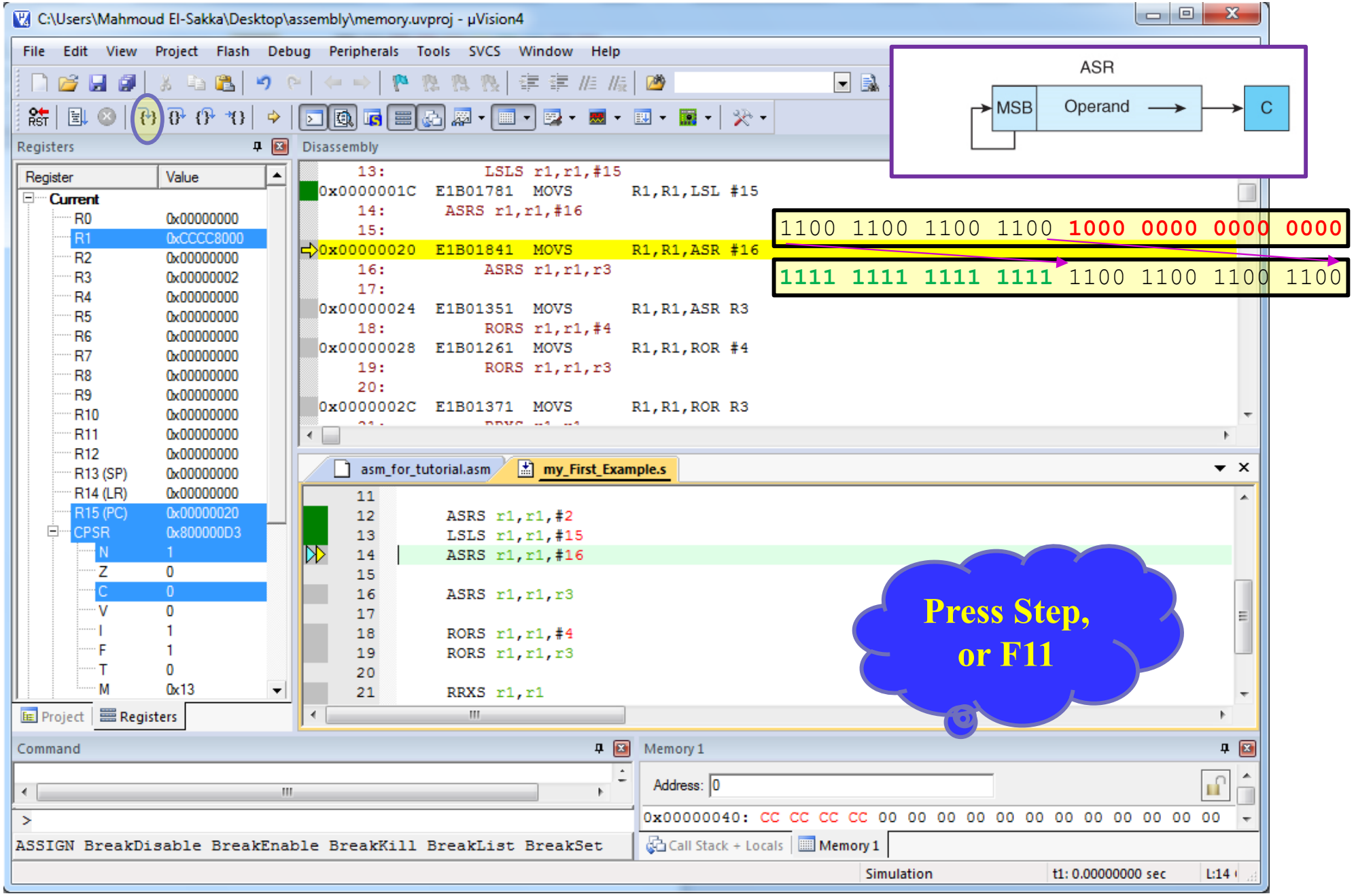
ARM's Data-Processing Instructions (Shift Operations)



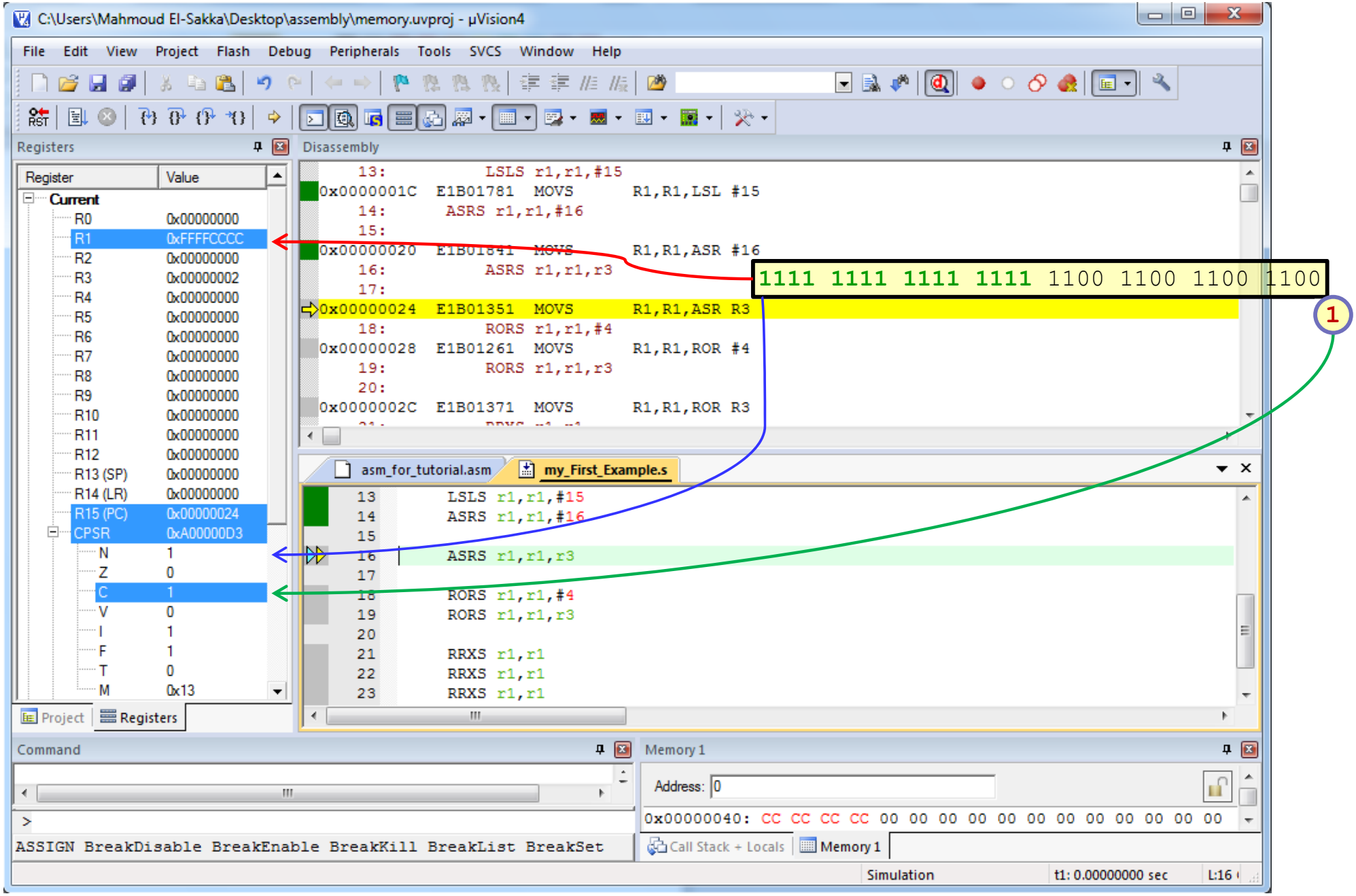
ARM's Data-Processing Instructions (Shift Operations)



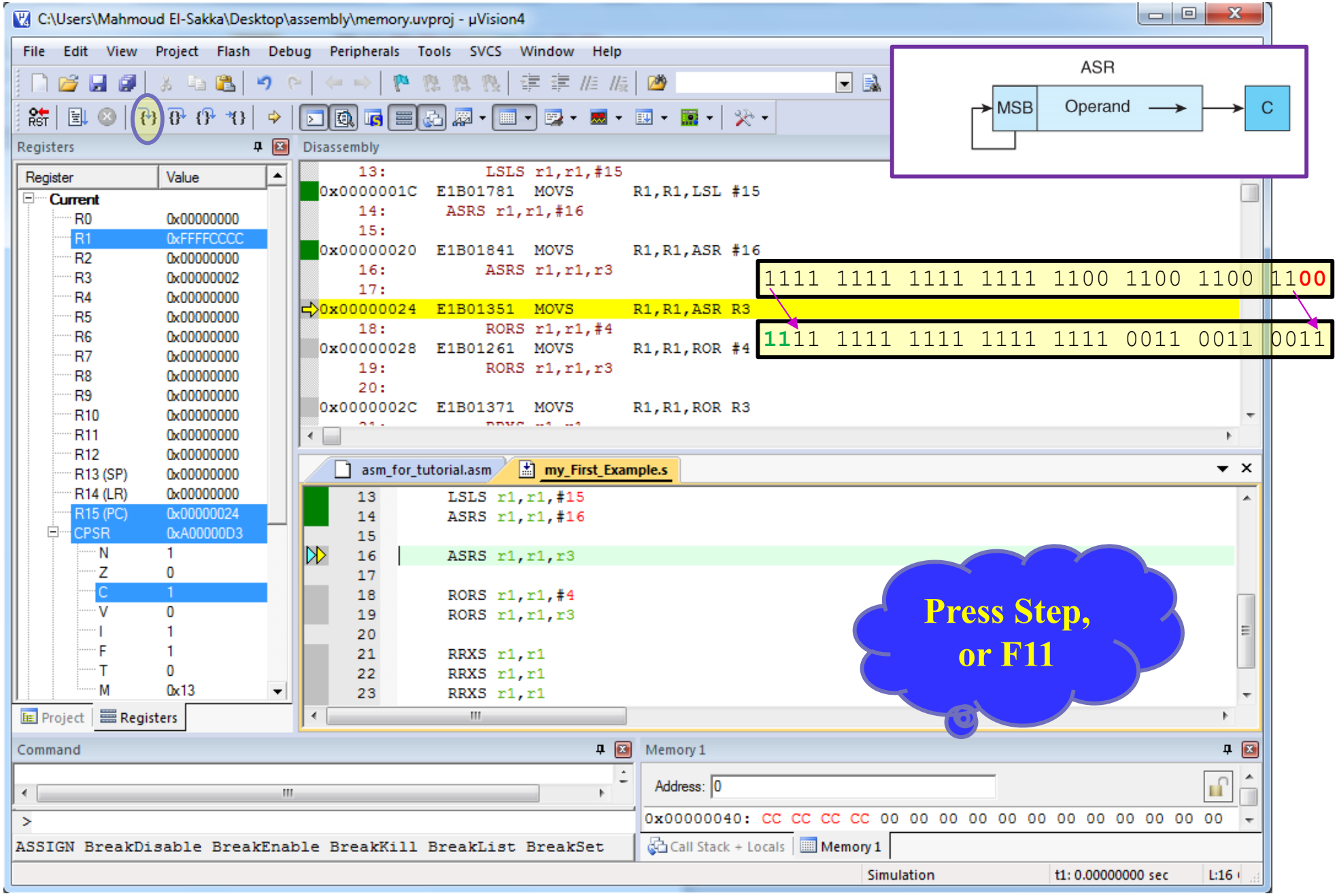
ARM's Data-Processing Instructions (Shift Operations)



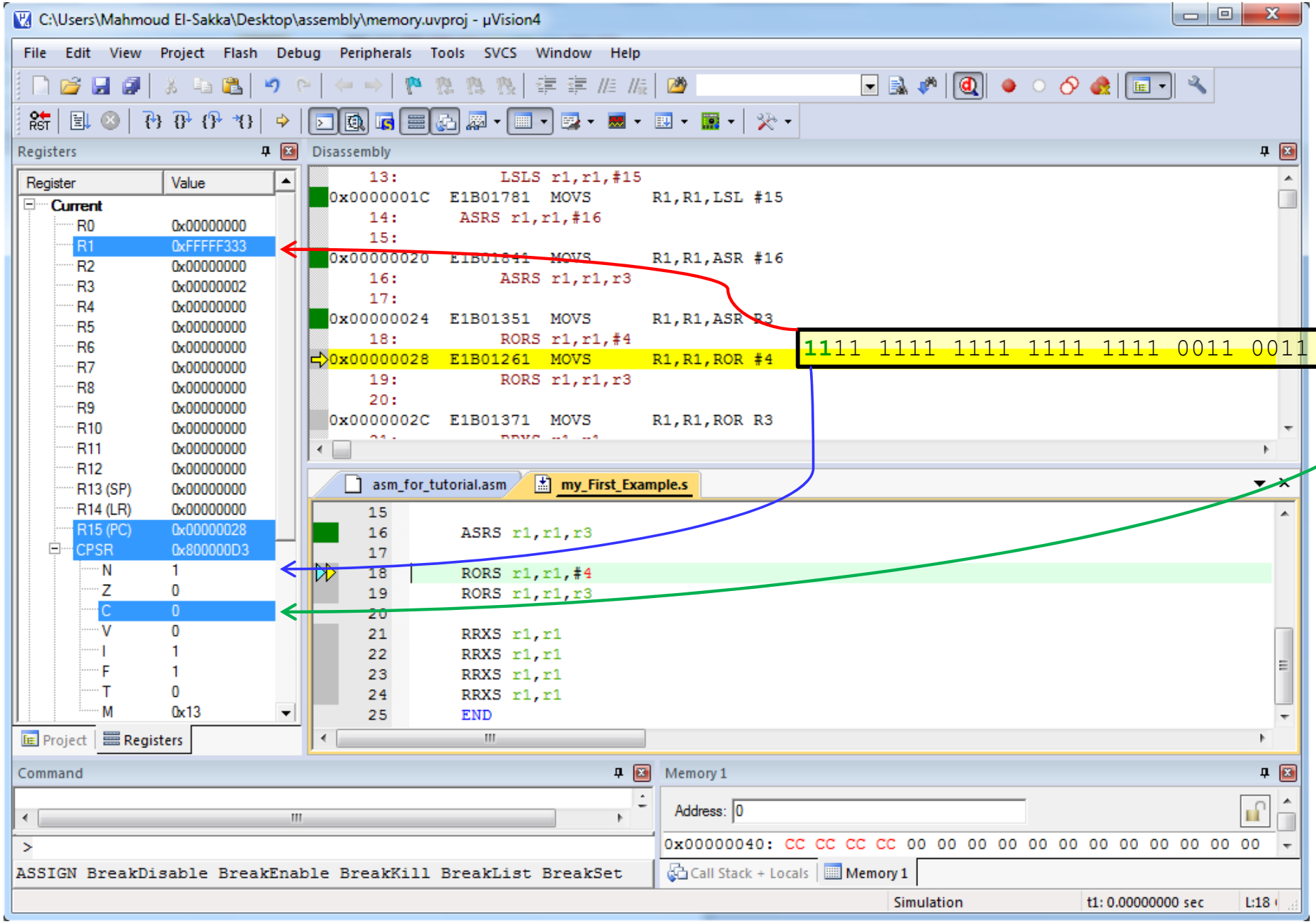
ARM's Data-Processing Instructions (Shift Operations)



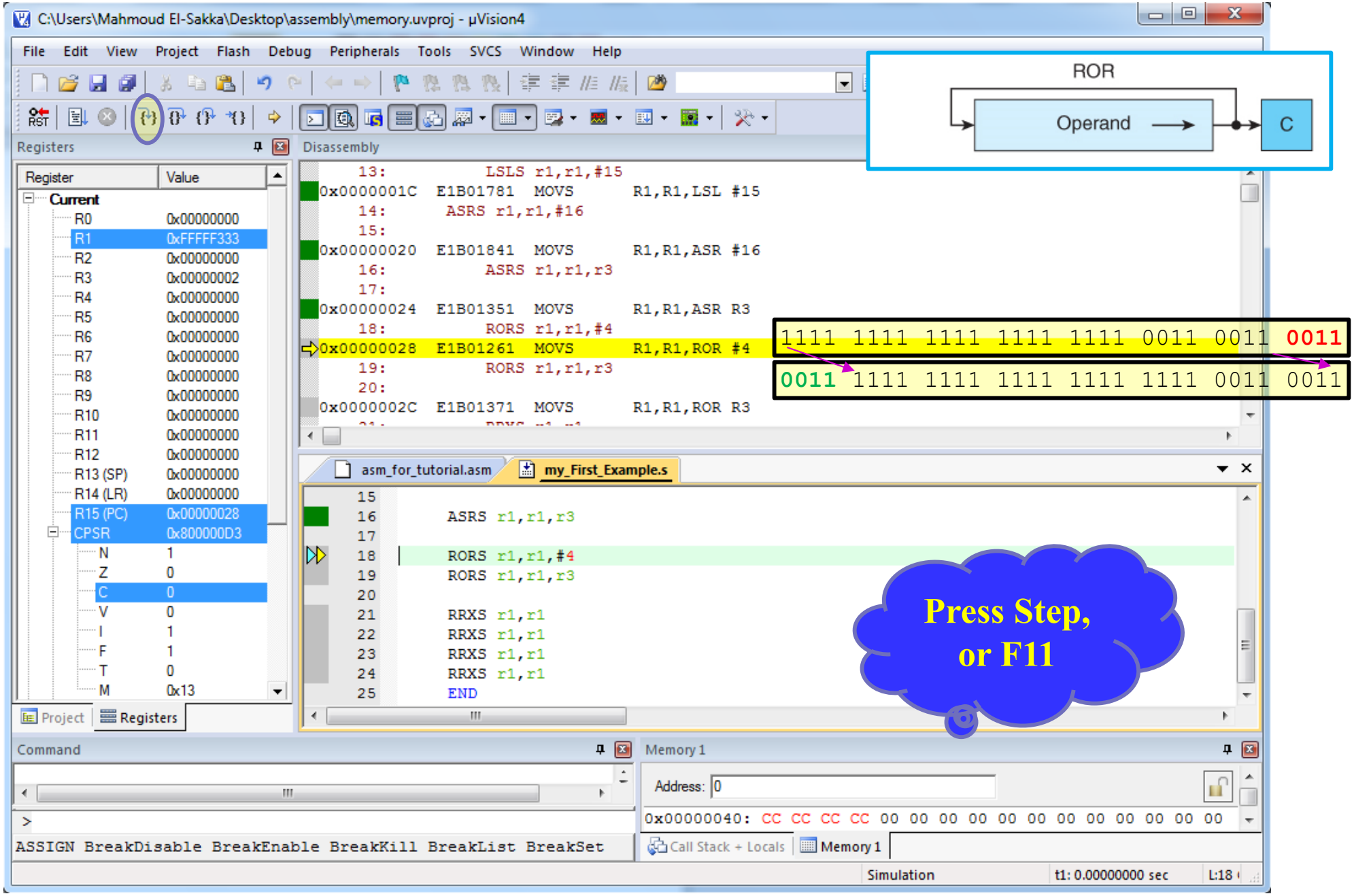
ARM's Data-Processing Instructions (Shift Operations)



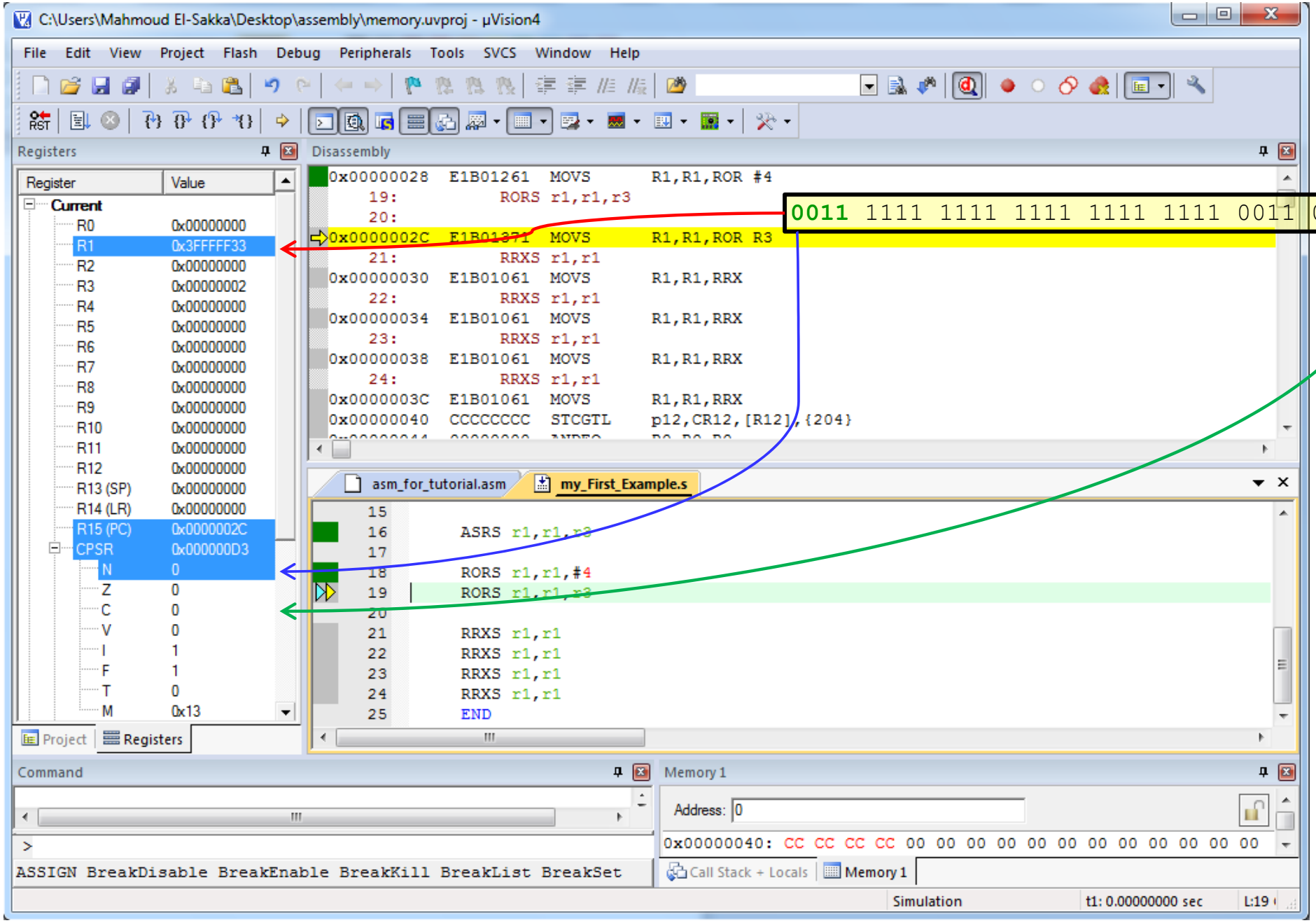
ARM's Data-Processing Instructions (Shift Operations)



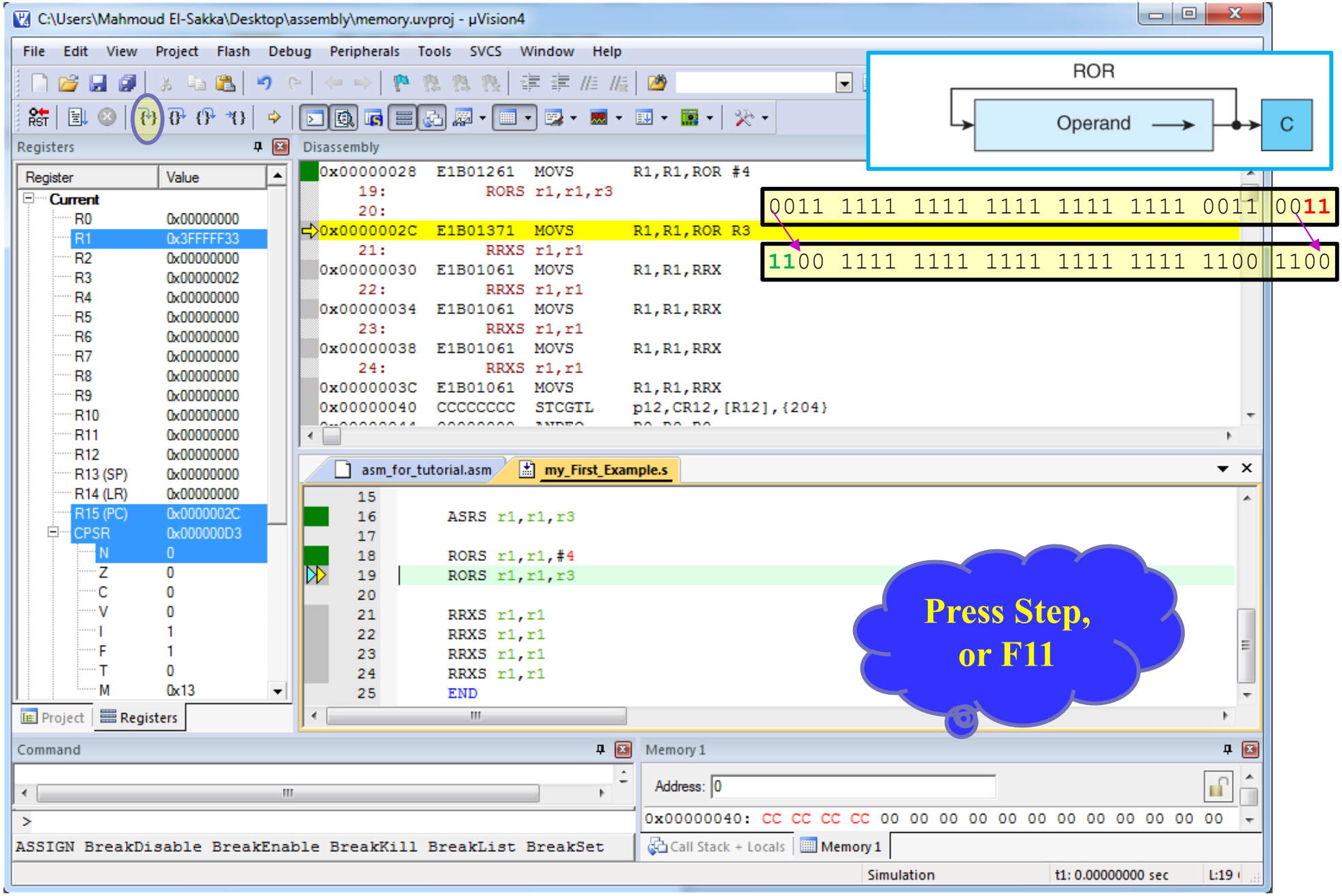
ARM's Data-Processing Instructions (Shift Operations)



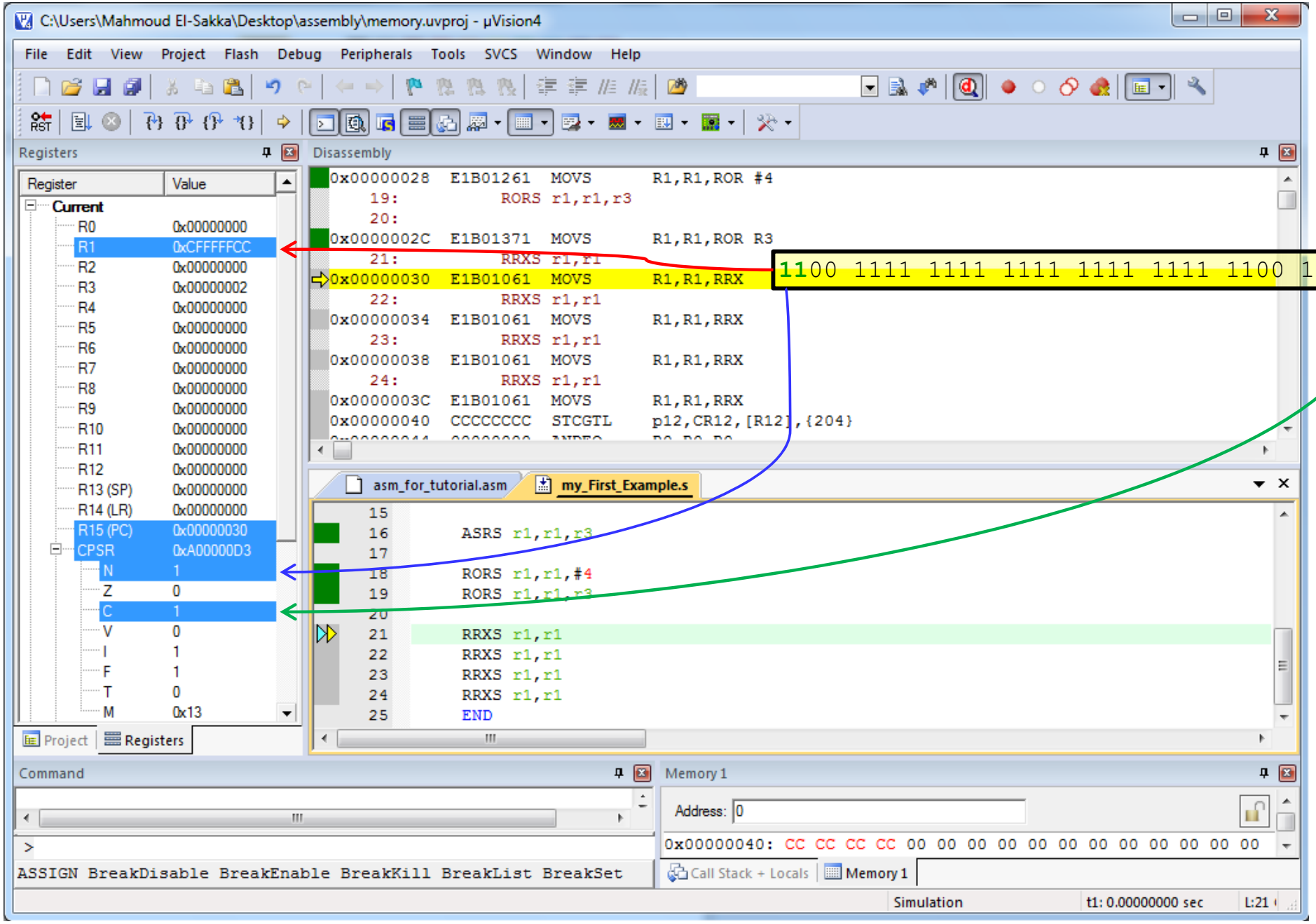
ARM's Data-Processing Instructions (Shift Operations)



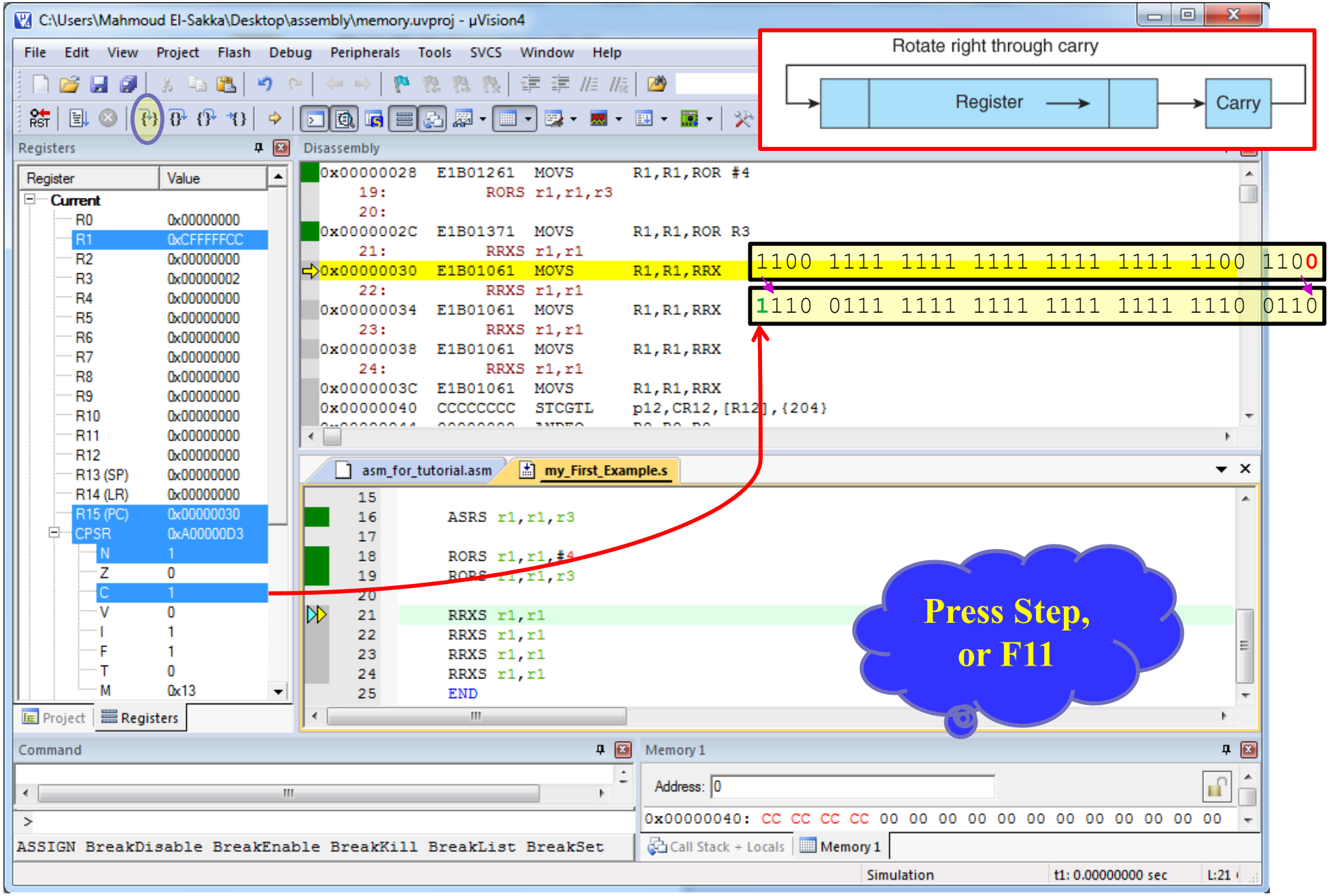
ARM's Data-Processing Instructions (Shift Operations)



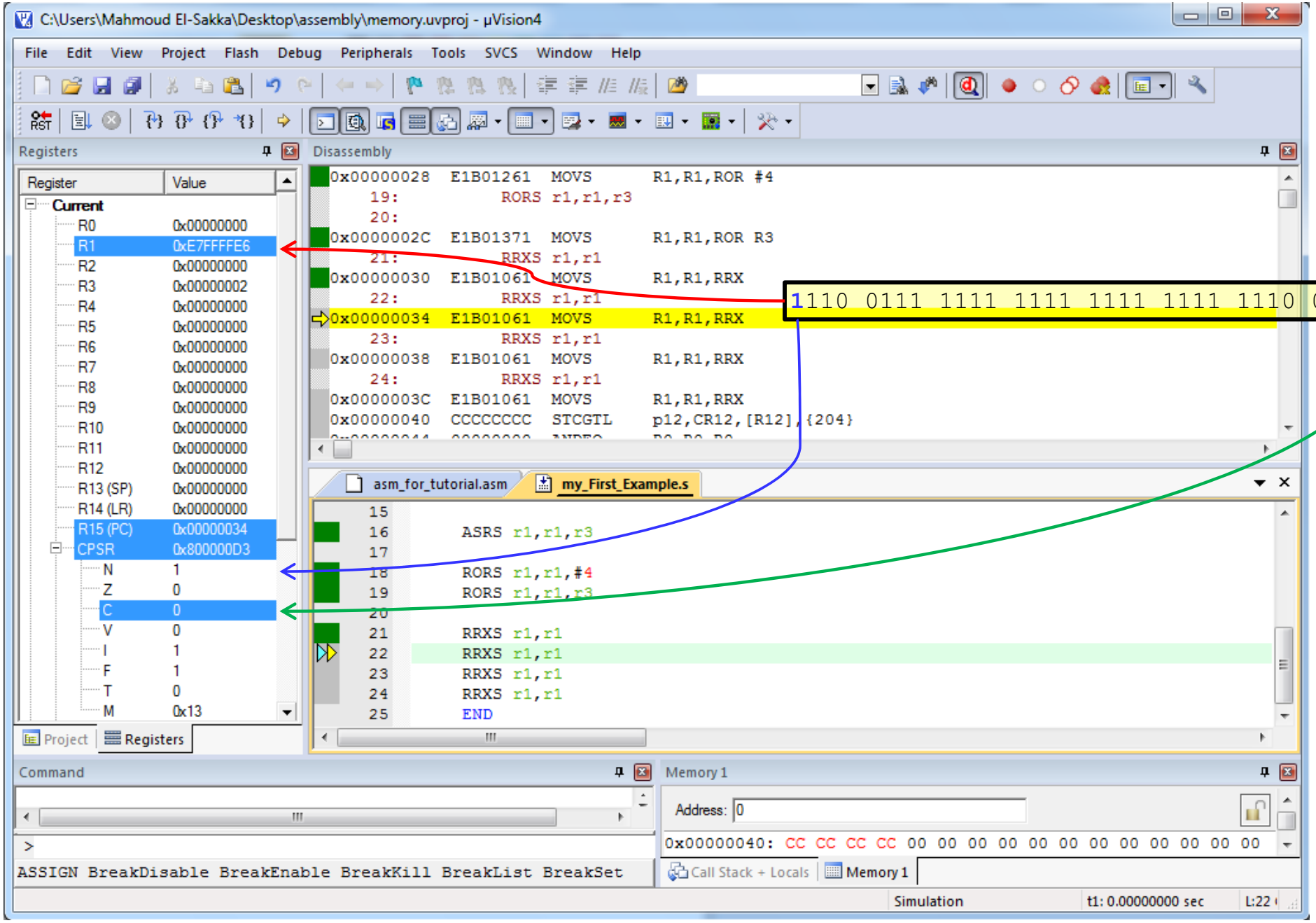
ARM's Data-Processing Instructions (Shift Operations)



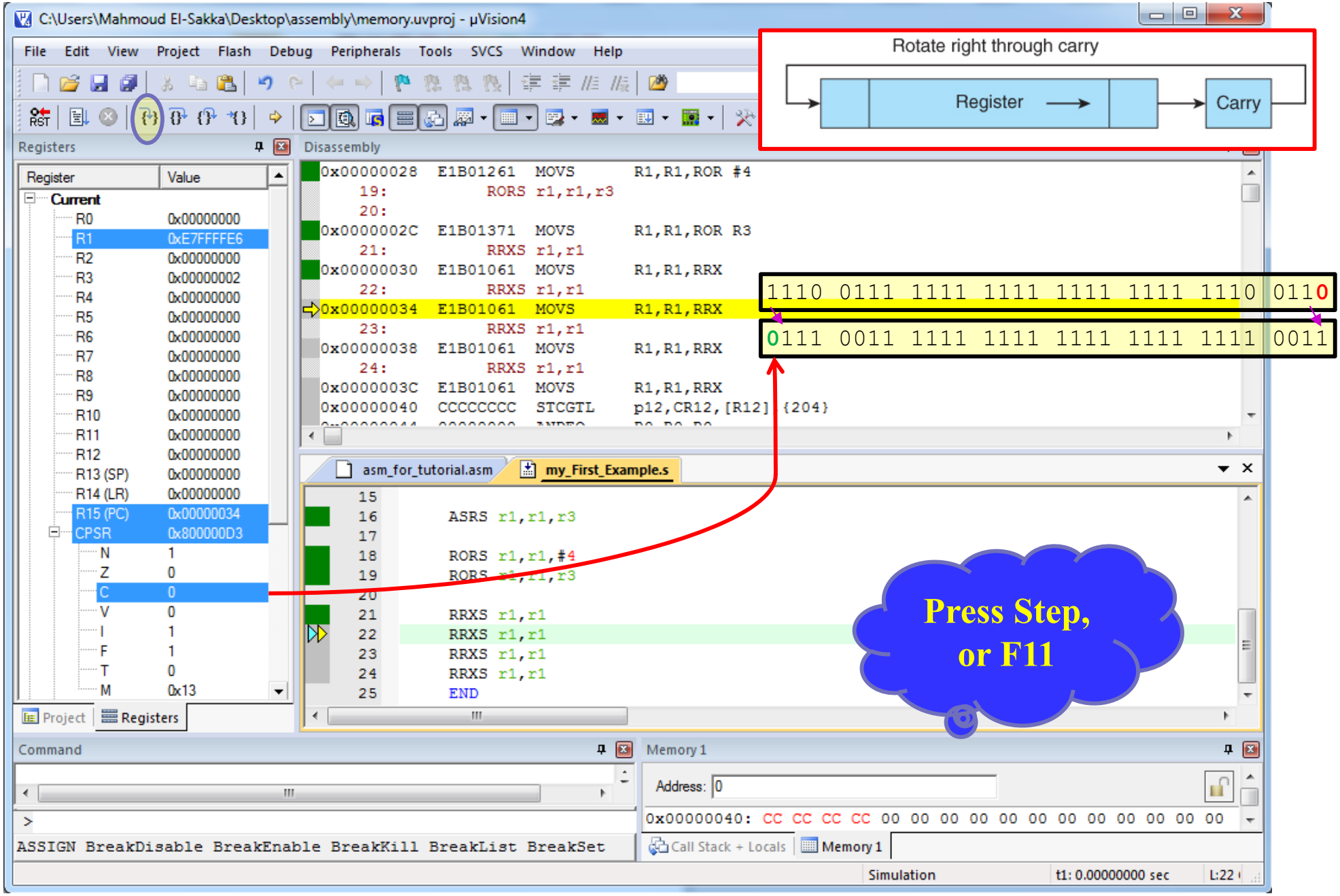
ARM's Data-Processing Instructions (Shift Operations)



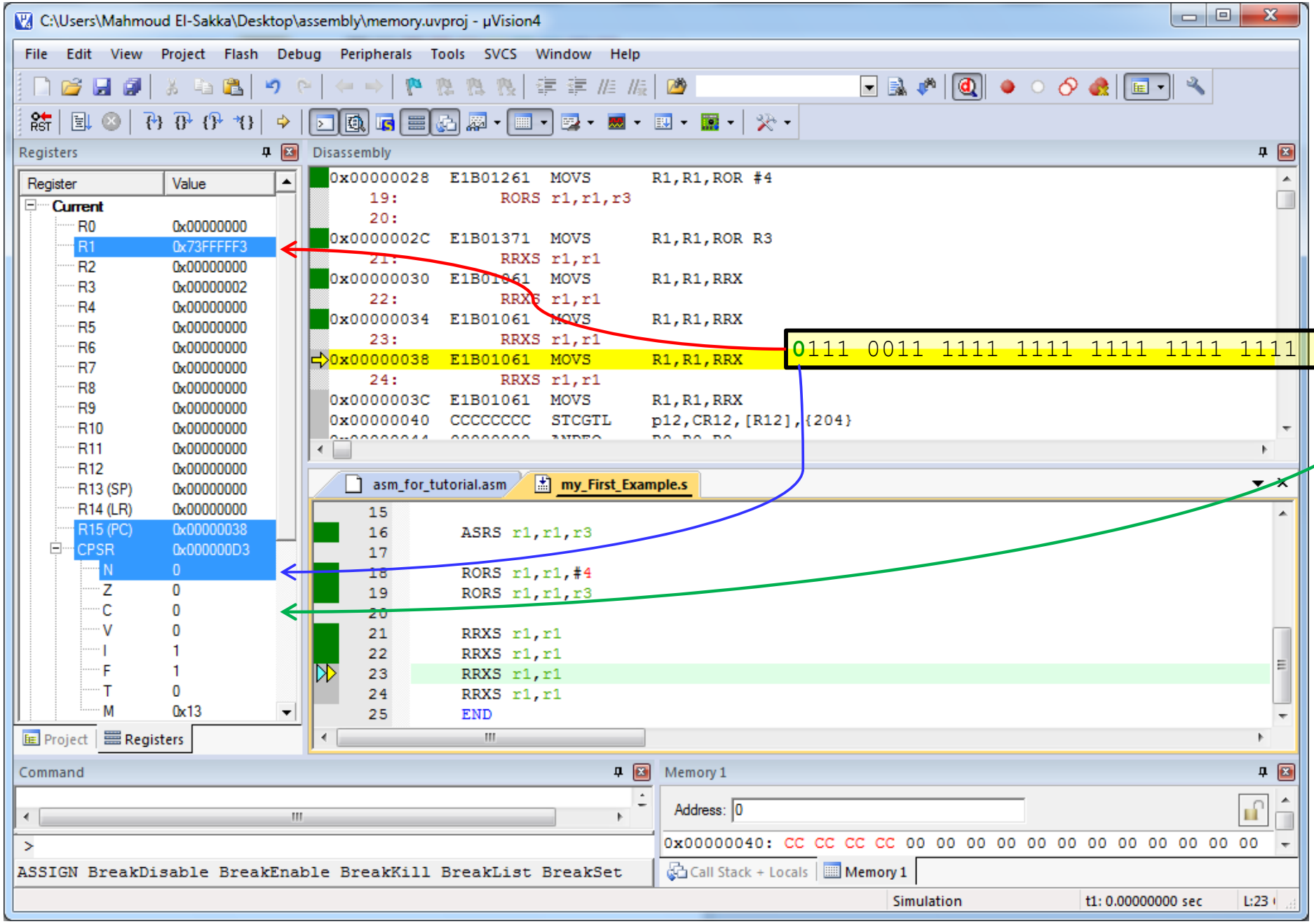
ARM's Data-Processing Instructions (Shift Operations)



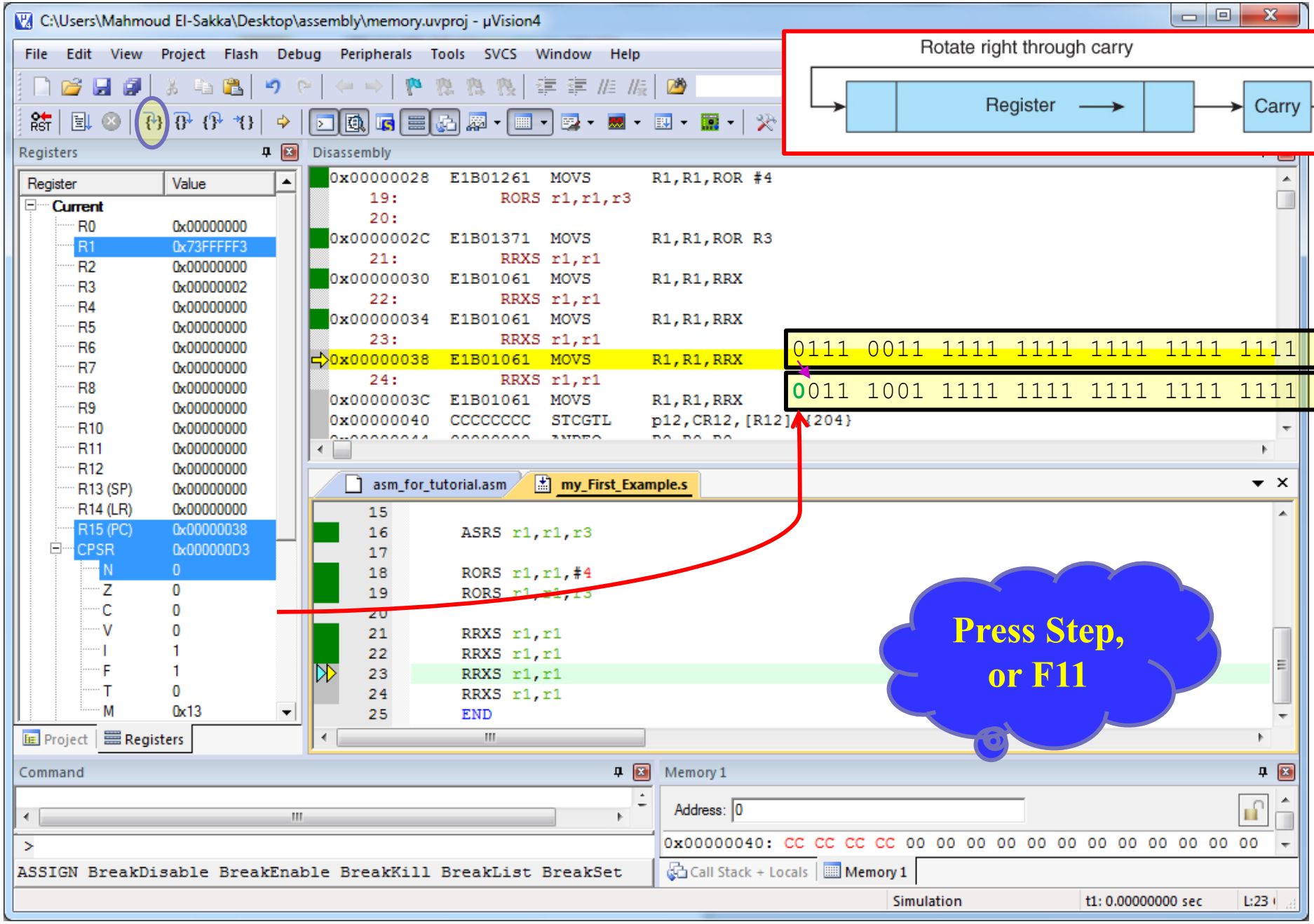
ARM's Data-Processing Instructions (Shift Operations)



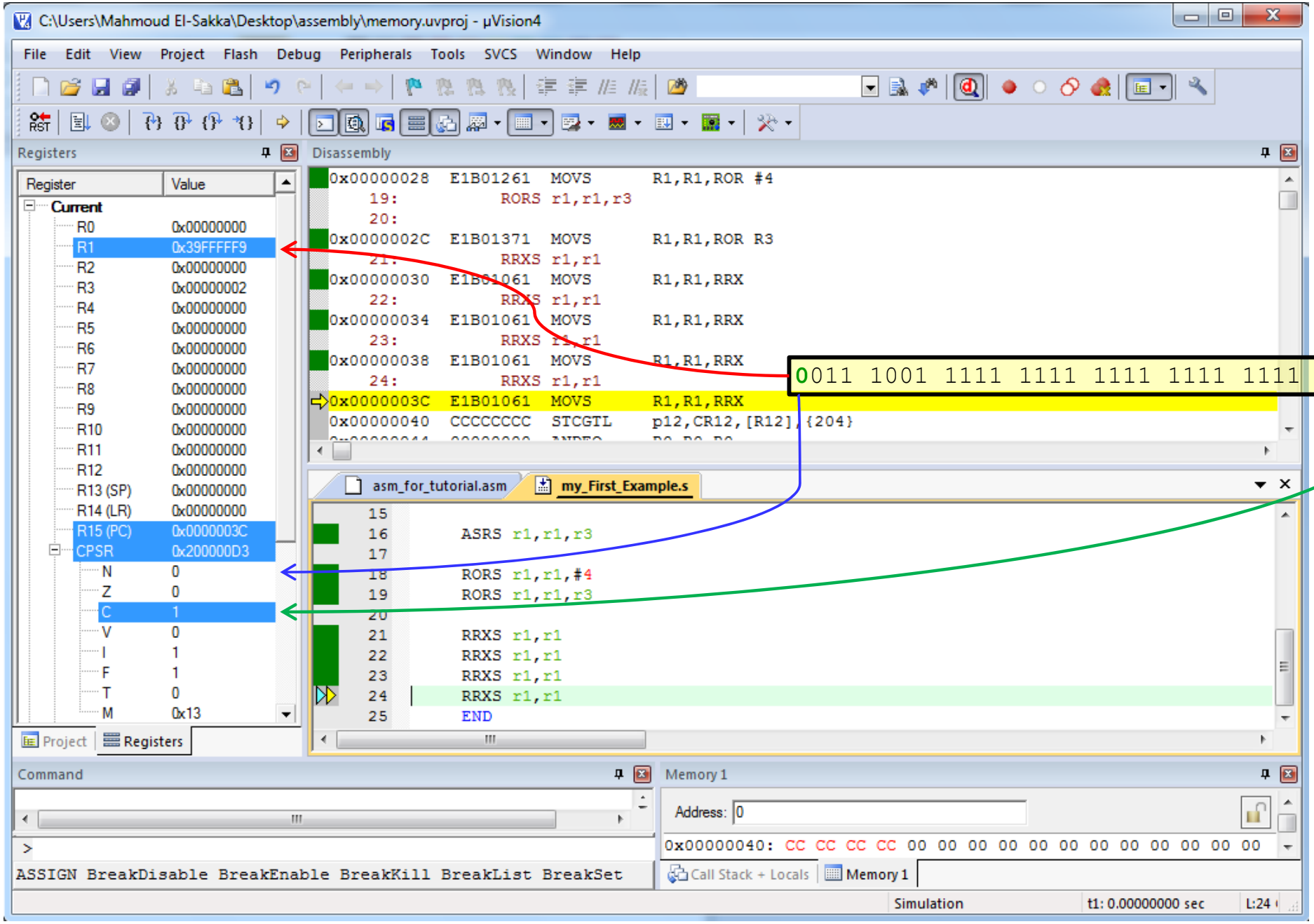
ARM's Data-Processing Instructions (Shift Operations)



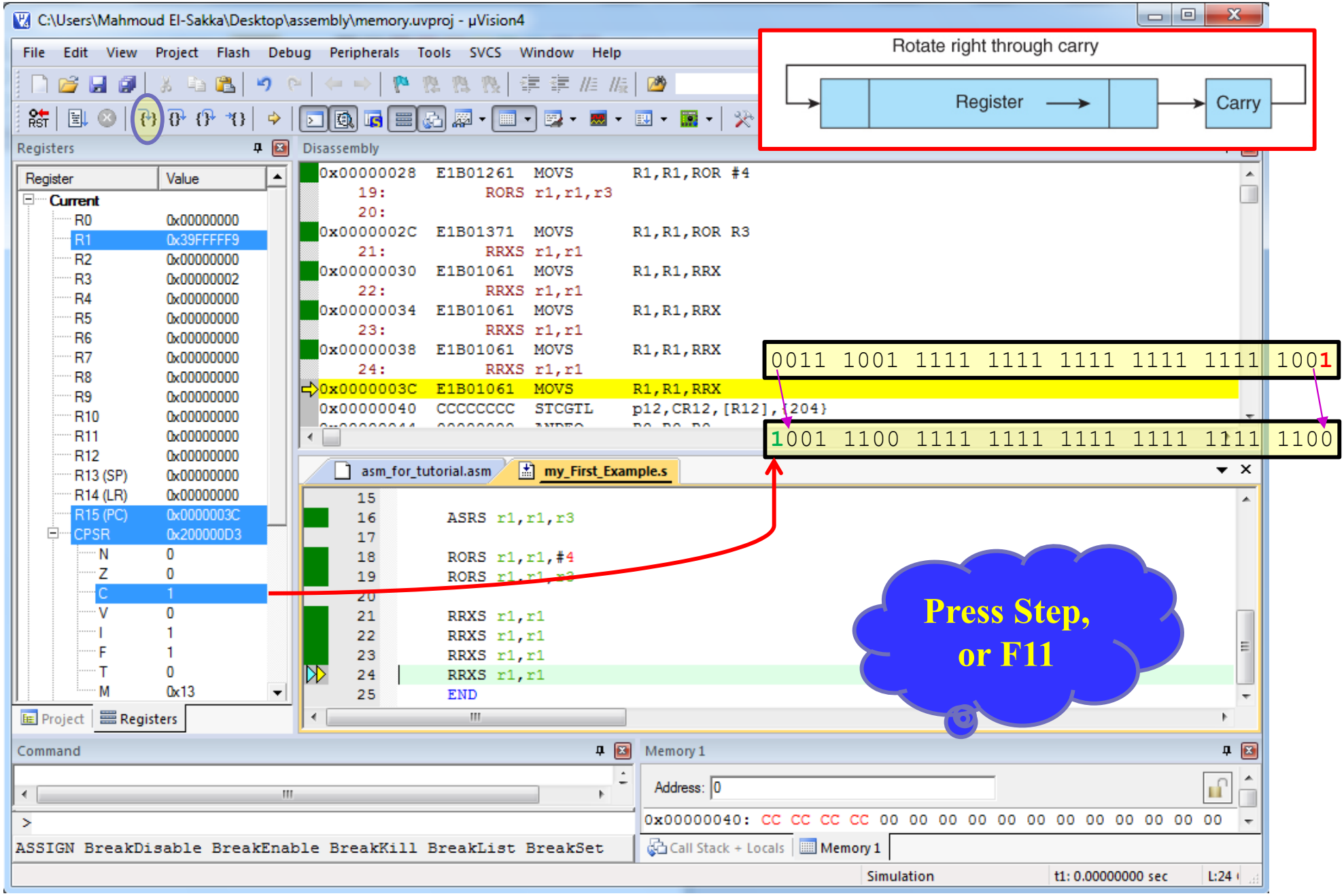
ARM's Data-Processing Instructions (Shift Operations)



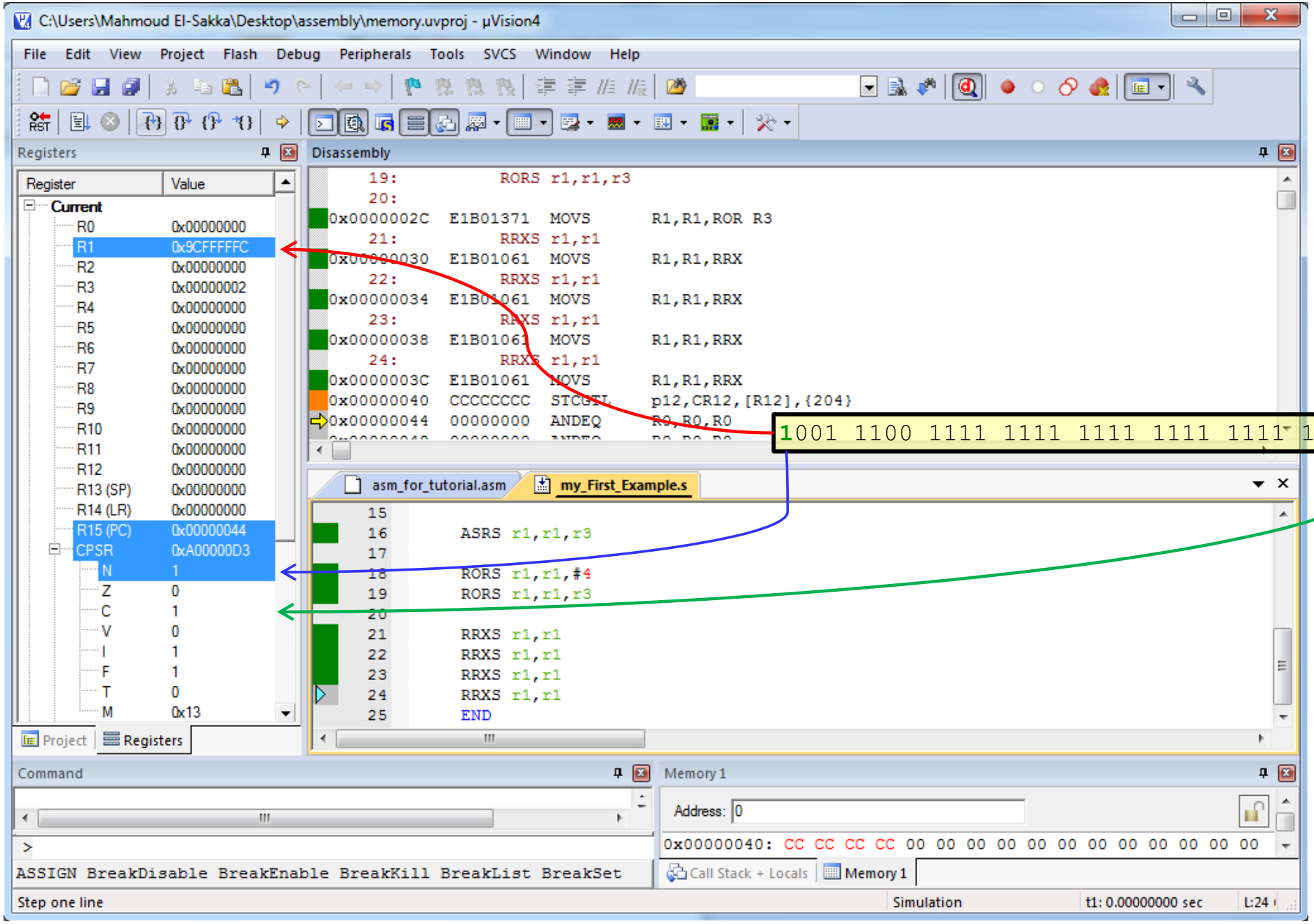
ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

```
AREA prog1, code, READONLY
ENTRY
MOV r3,#2
LDR r1, =0xCCCCCCCC ;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"