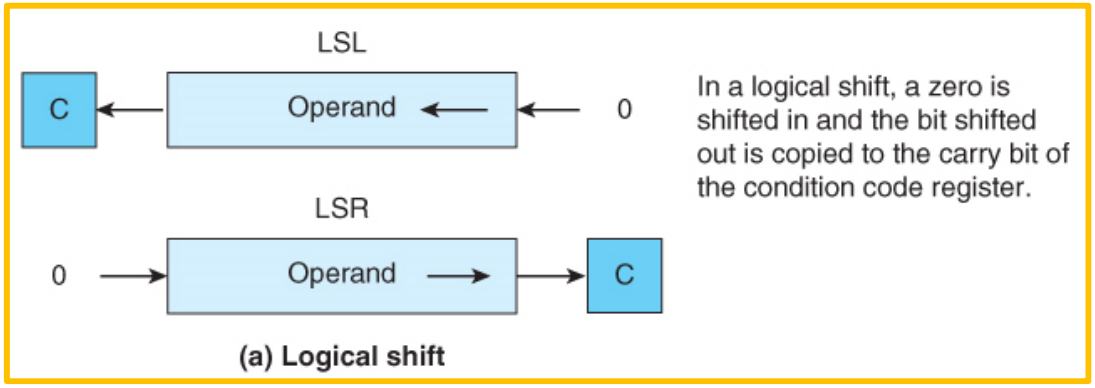
Tutorial 06: ARM Shift Instructions and ARM Addressing Modes

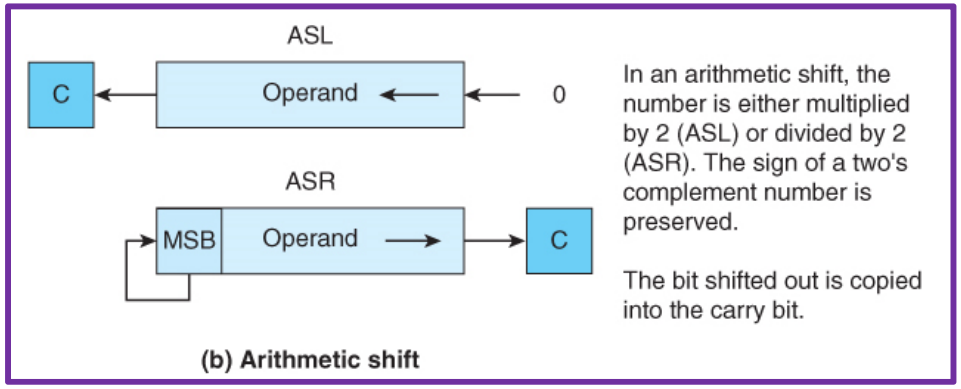
Computer Science Department
CS2208b: Introduction to Computer Organization and Architecture
Winter 2019
Instructor: Mahmoud R. El-Sakka
Office: MC-419
Email: elsakka@csd.uwo.ca
Phone: 519-661-2111 x86996

ARM's Data-Processing Instructions (Shift Operations)

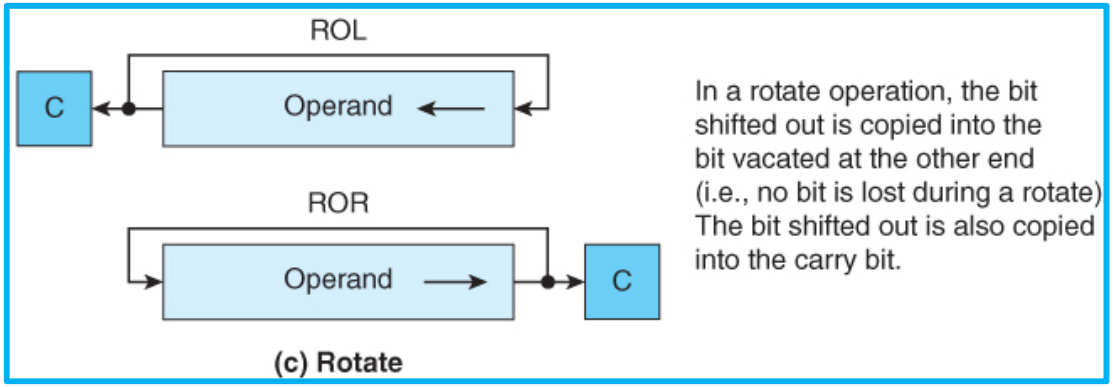
- *Shift* operations move bits one ***or more*** places *left* or *right*.
  - ***Logical shifts***
    - *insert a 0* in the vacated position.
  - ***Arithmetic shifts***
    - *replicate the sign-bit* during a right shift
  - ***Circular shifts***
    - *the bit shifted out of one end is shifted in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



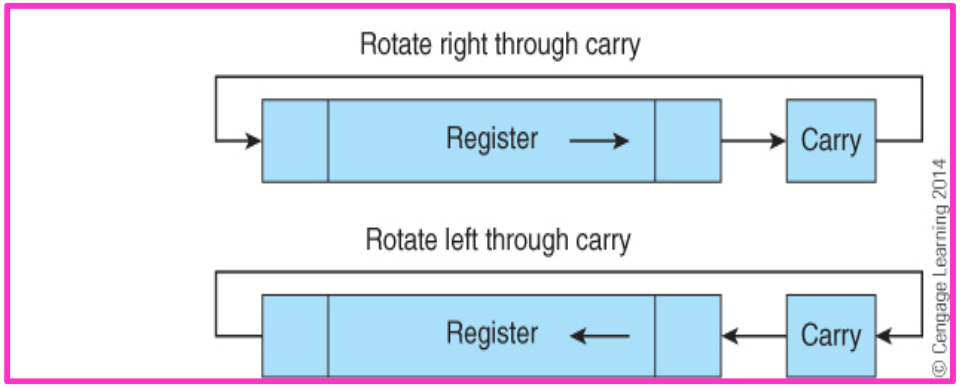
(a) Logical shift



(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*
    - 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 × 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] ← [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 moving operations*
  - This way, a shift operation can be performed as a stand alone operation.

  - For example,

```
MOV r3,r3,LSL #1             ;[r3] ← [r3] × 2
```

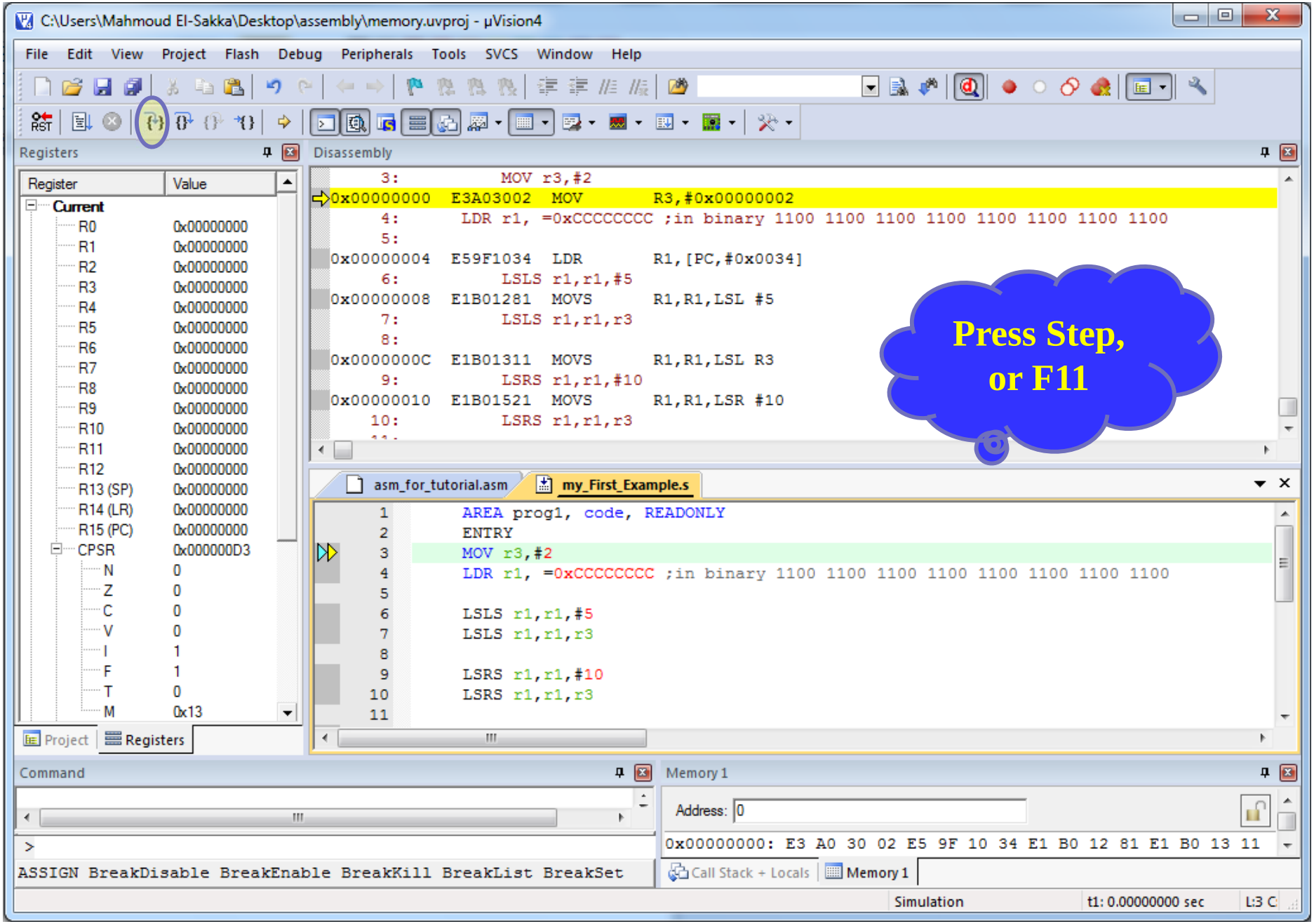
  - **ARM** provides pseudo shift instructions, which are translated to `MOV` instructions.

```
LSL r3,r3,#1     ; will be converted to MOV r3,r3,LSL #1
```

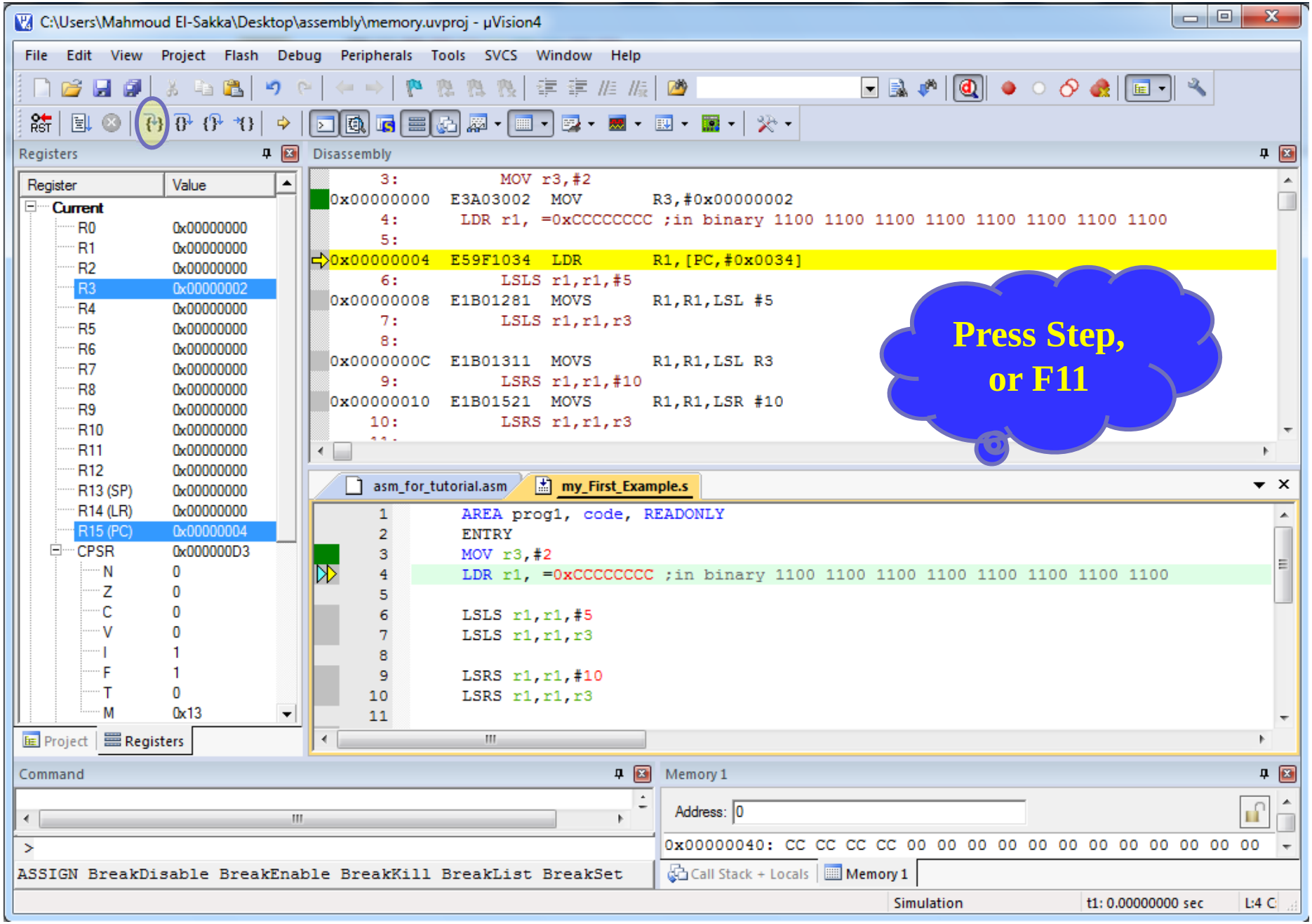
ARM's Data-Processing Instructions (Shift Operations)

```
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
```

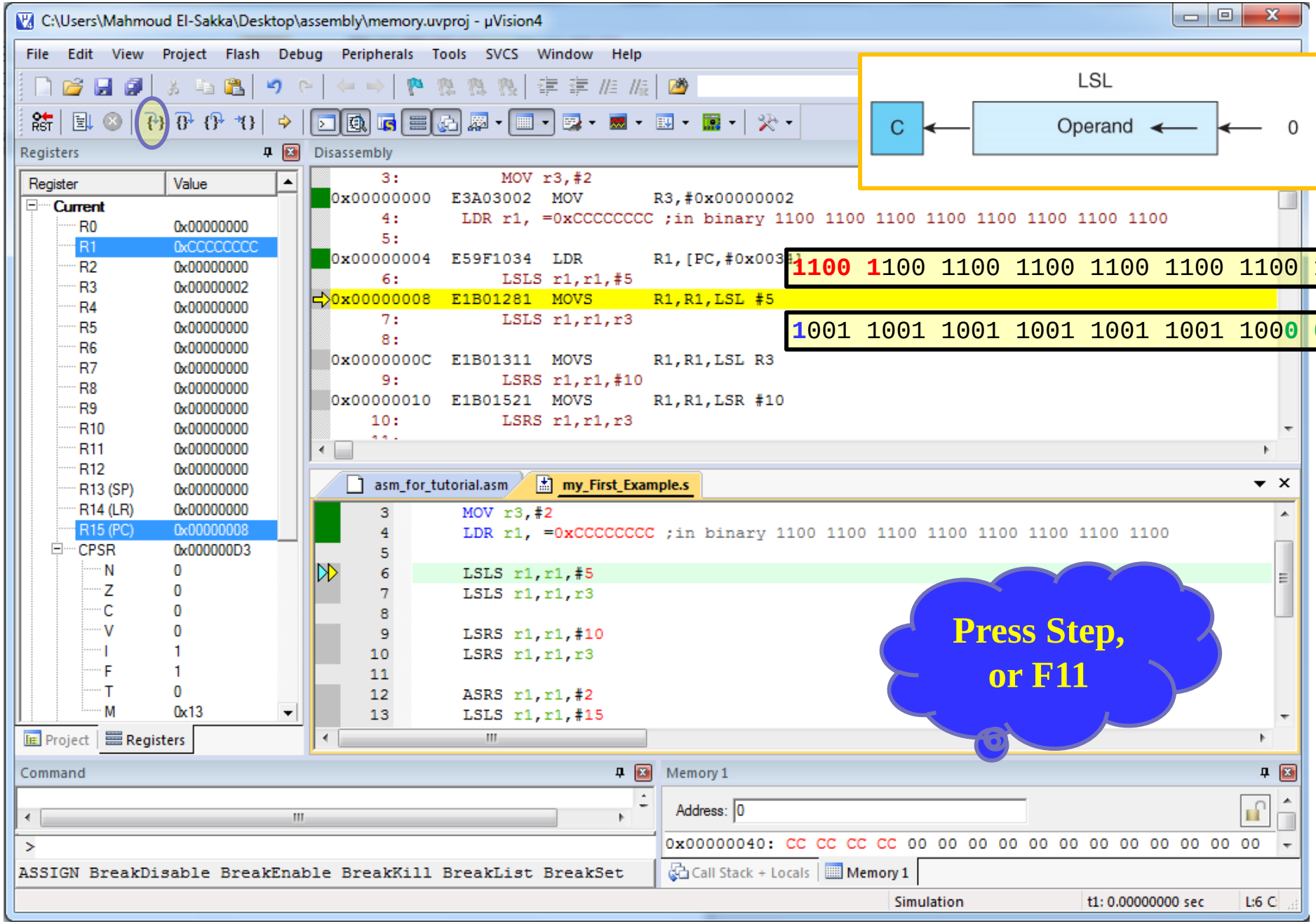
ARM's Data-Processing Instructions (Shift Operations)



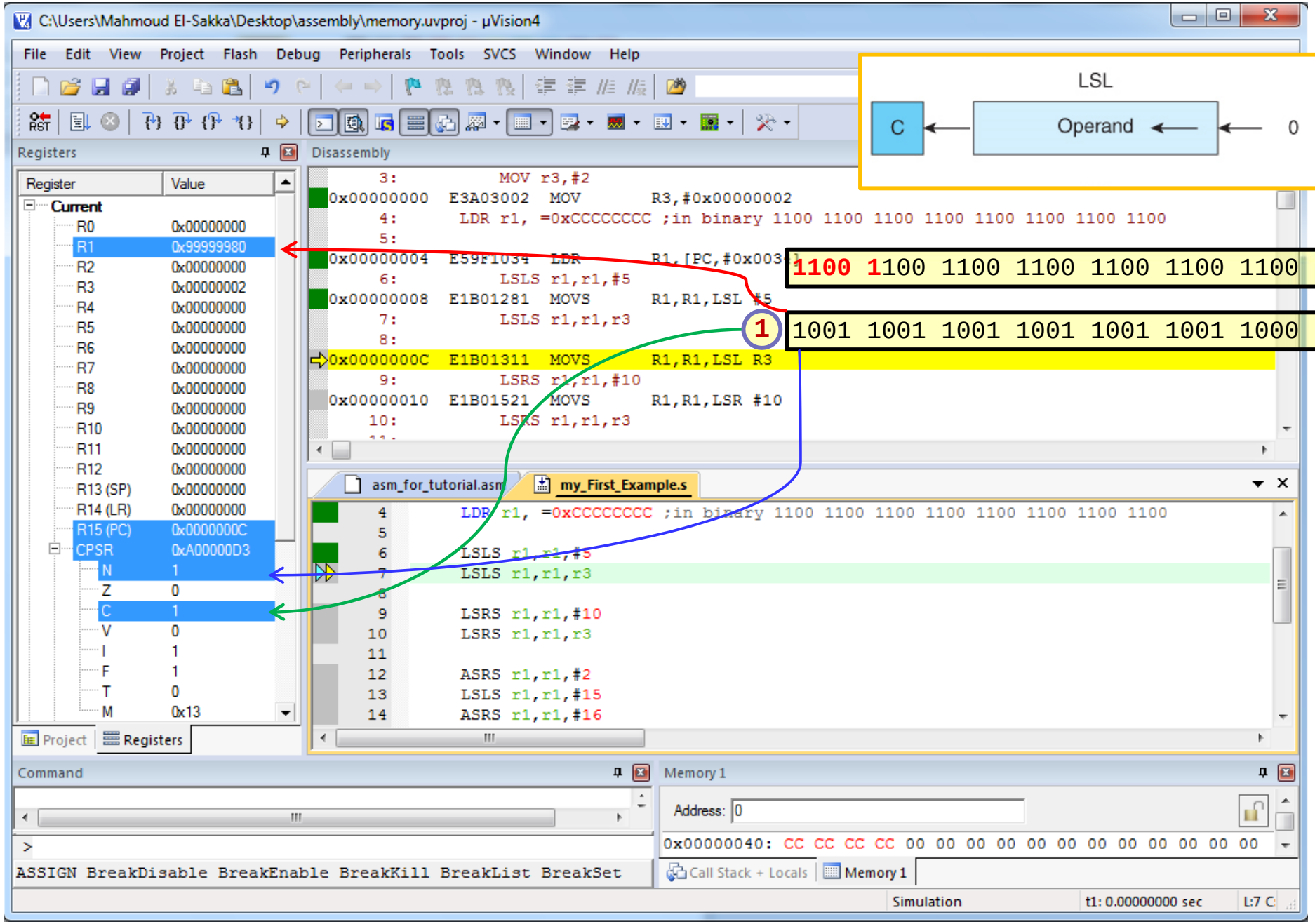
ARM's Data-Processing Instructions (Shift Operations)



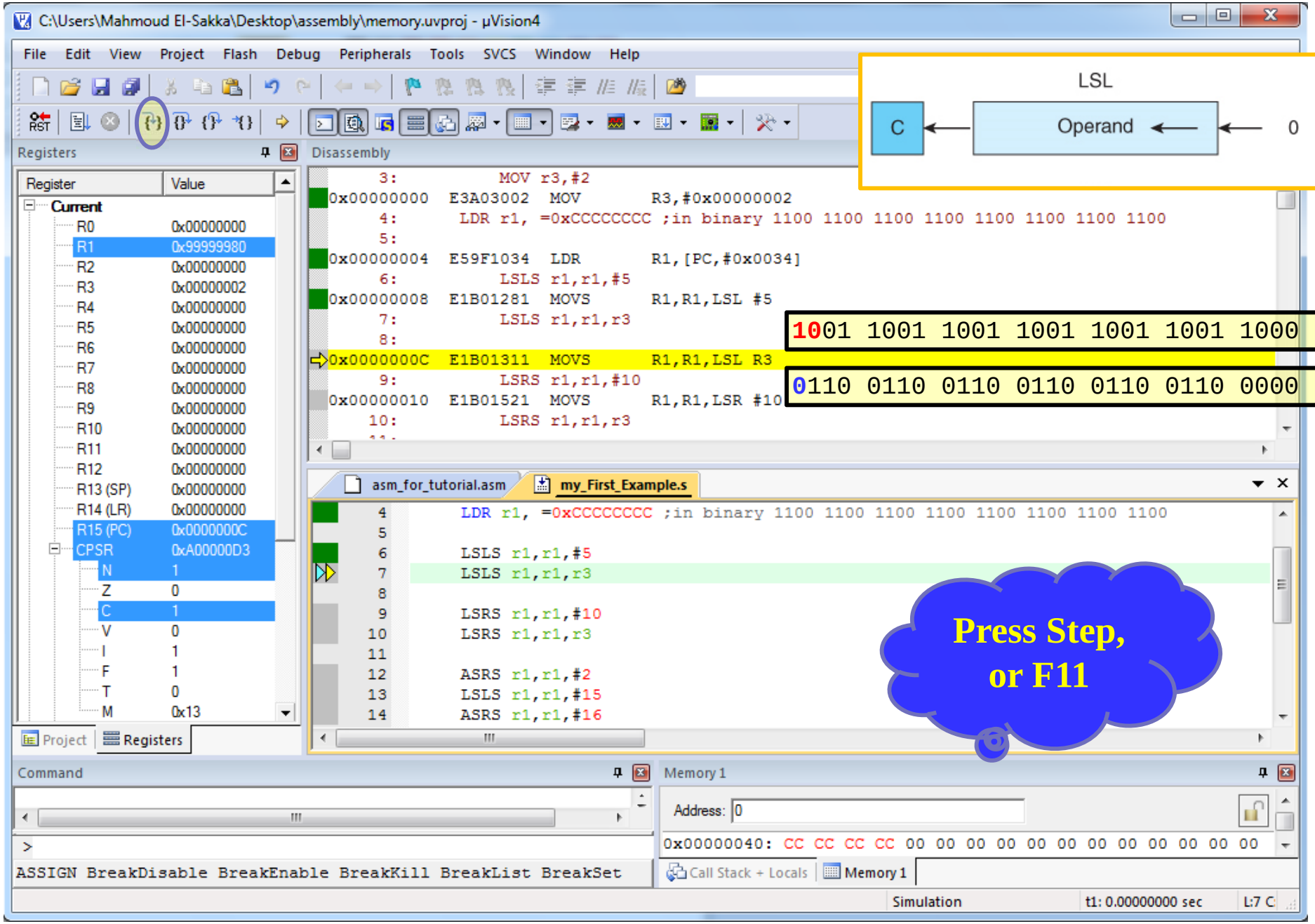
ARM's Data-Processing Instructions (Shift Operations)



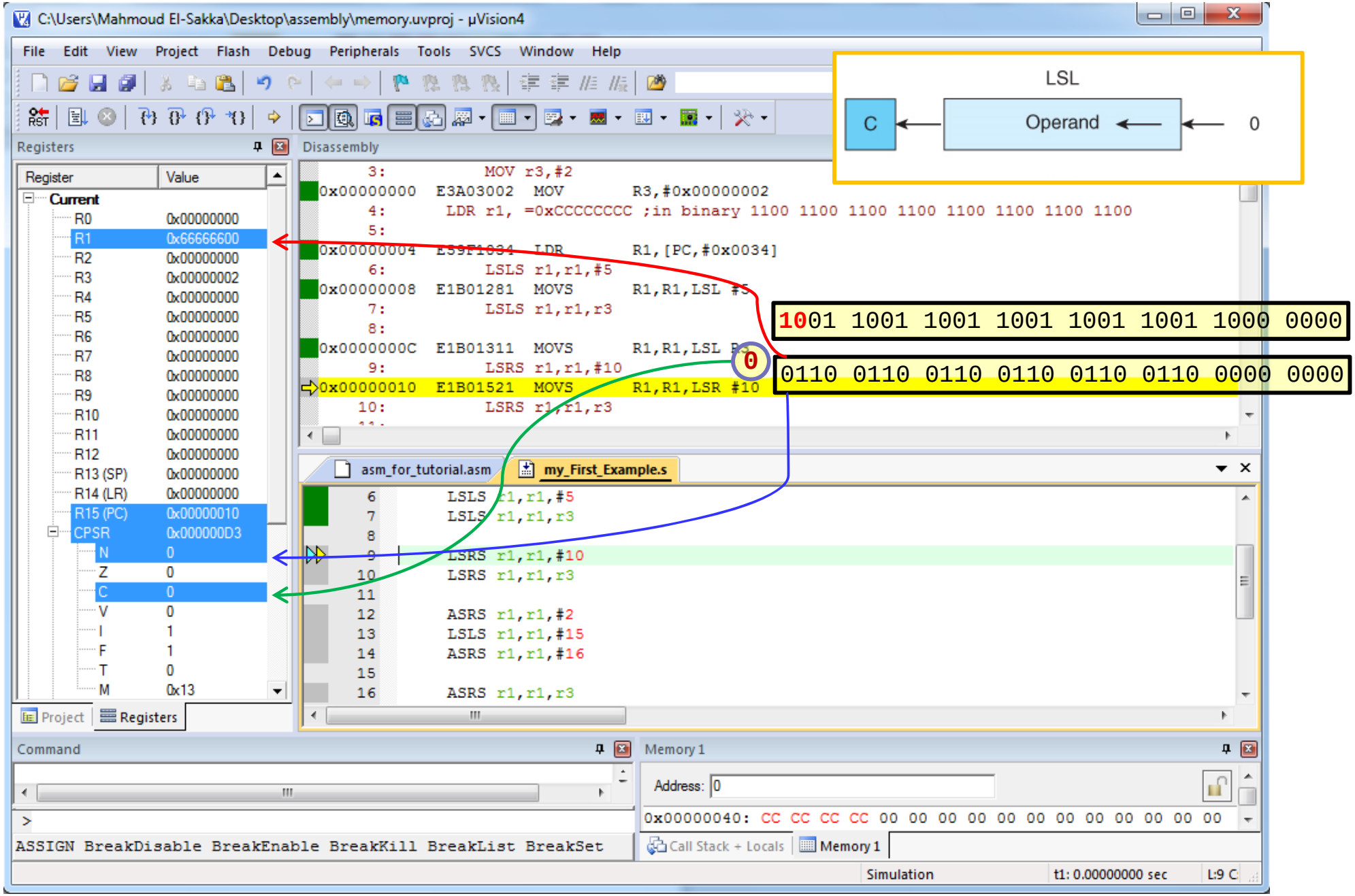
ARM's Data-Processing Instructions (Shift Operations)



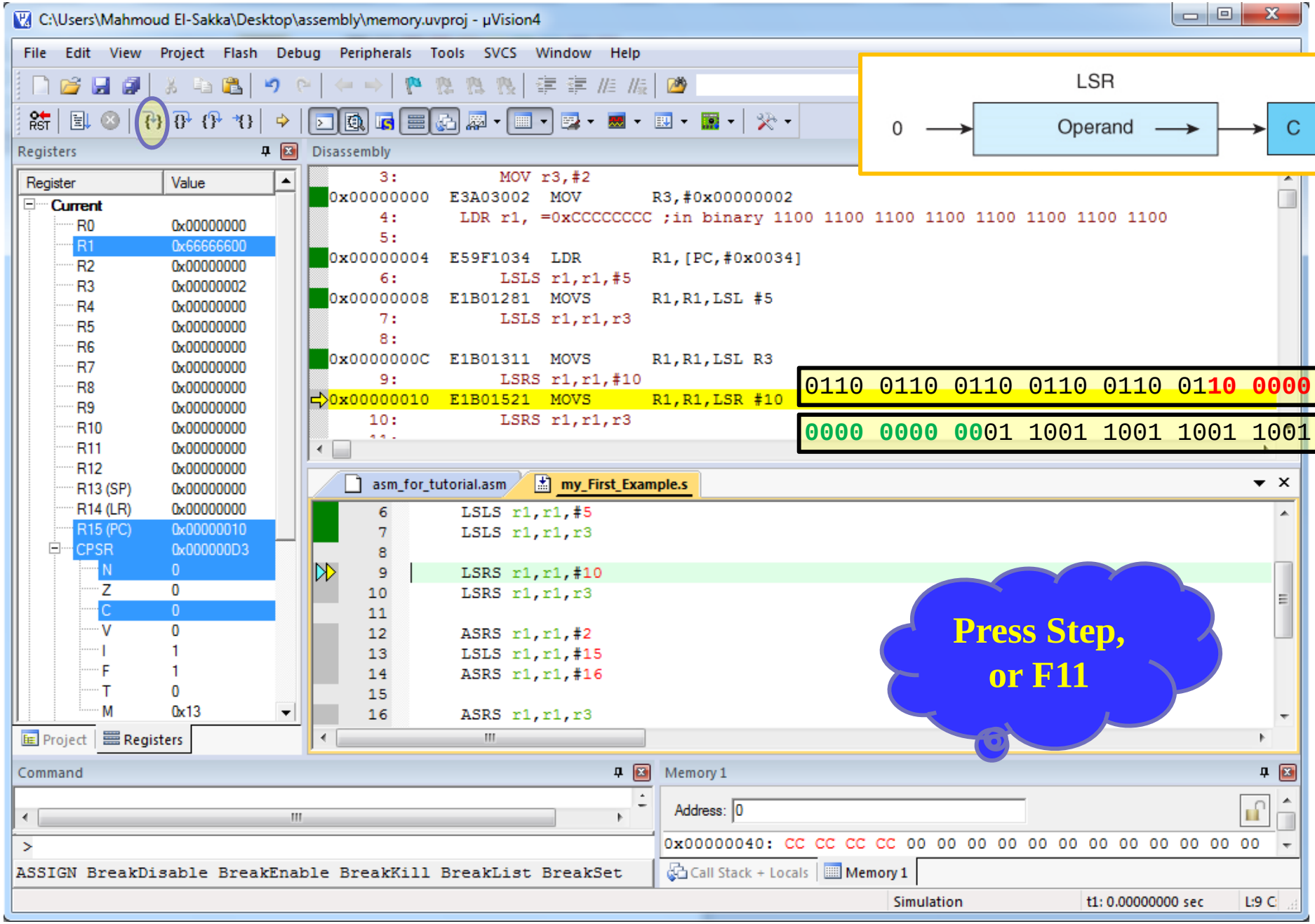
ARM's Data-Processing Instructions (Shift Operations)



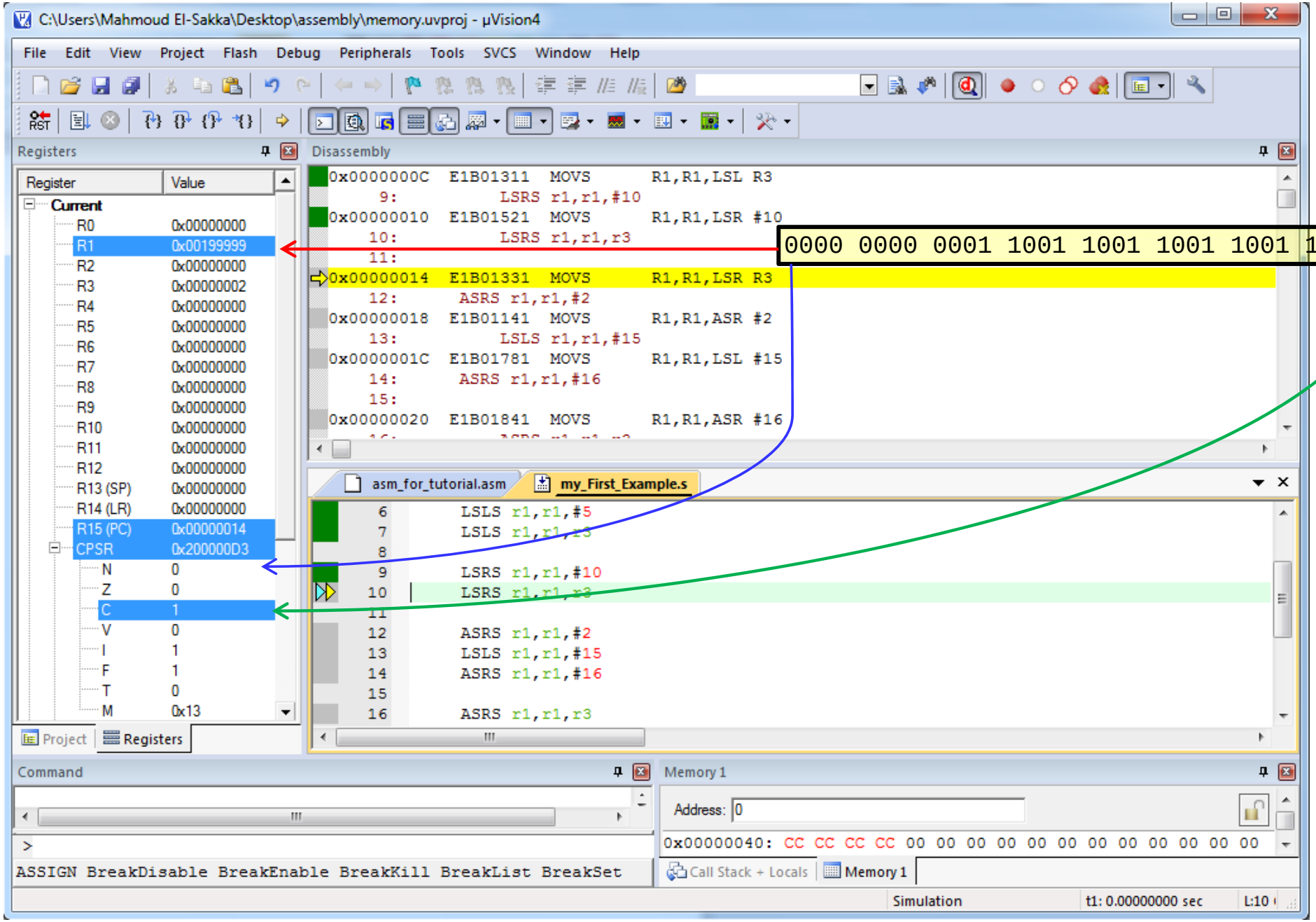
ARM's Data-Processing Instructions (Shift Operations)



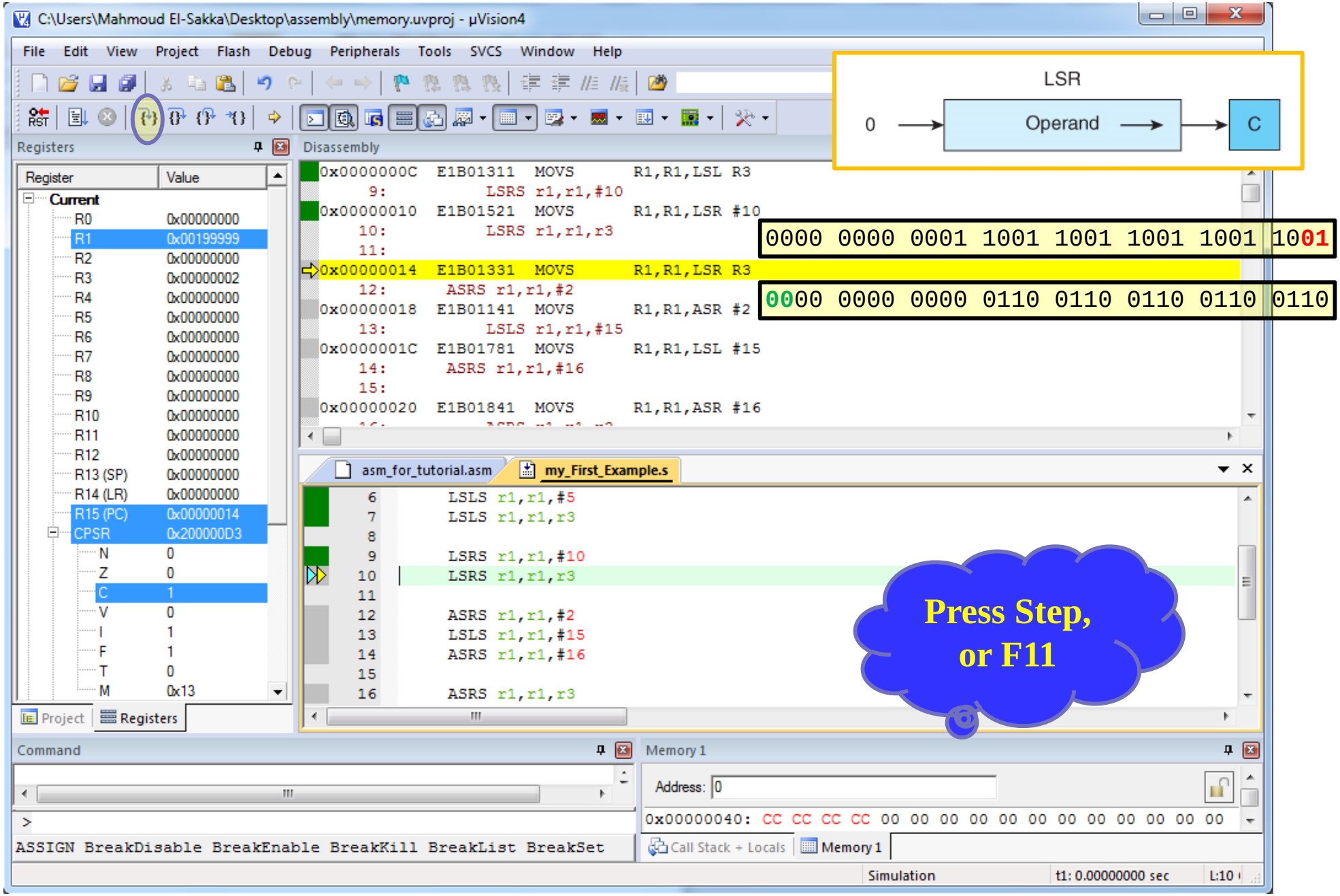
ARM's Data-Processing Instructions (Shift Operations)



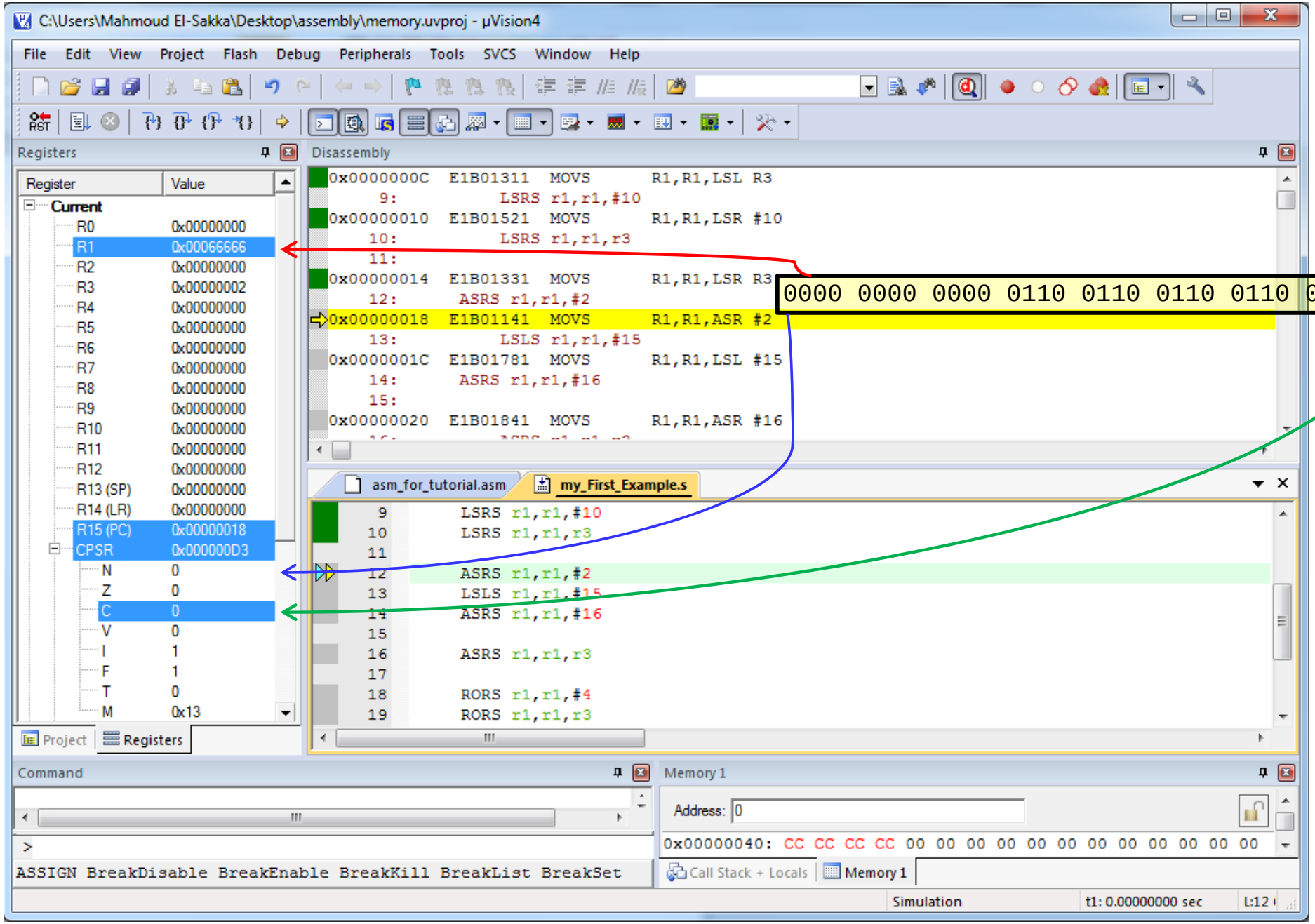
ARM's Data-Processing Instructions (Shift Operations)



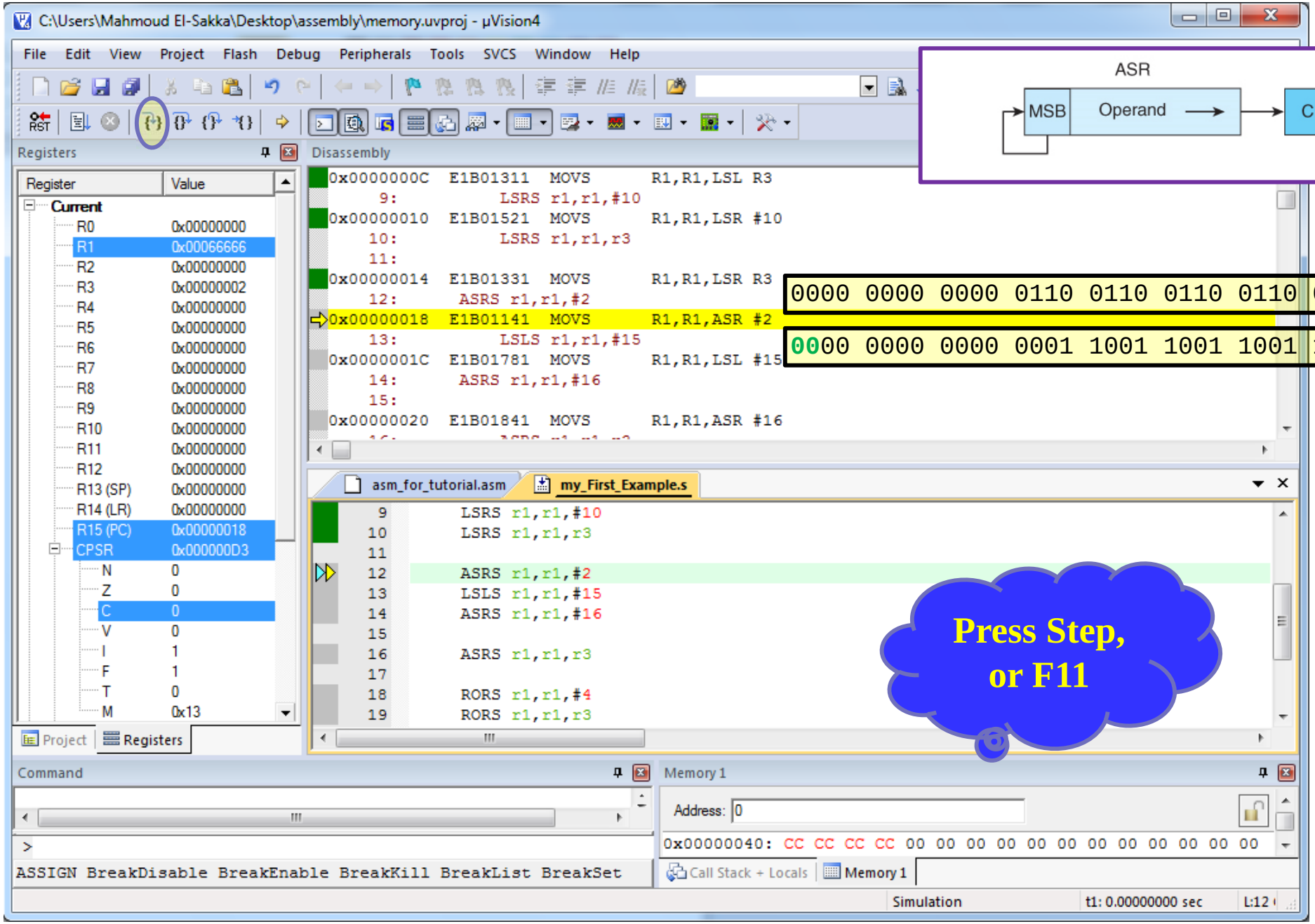
ARM's Data-Processing Instructions (Shift Operations)



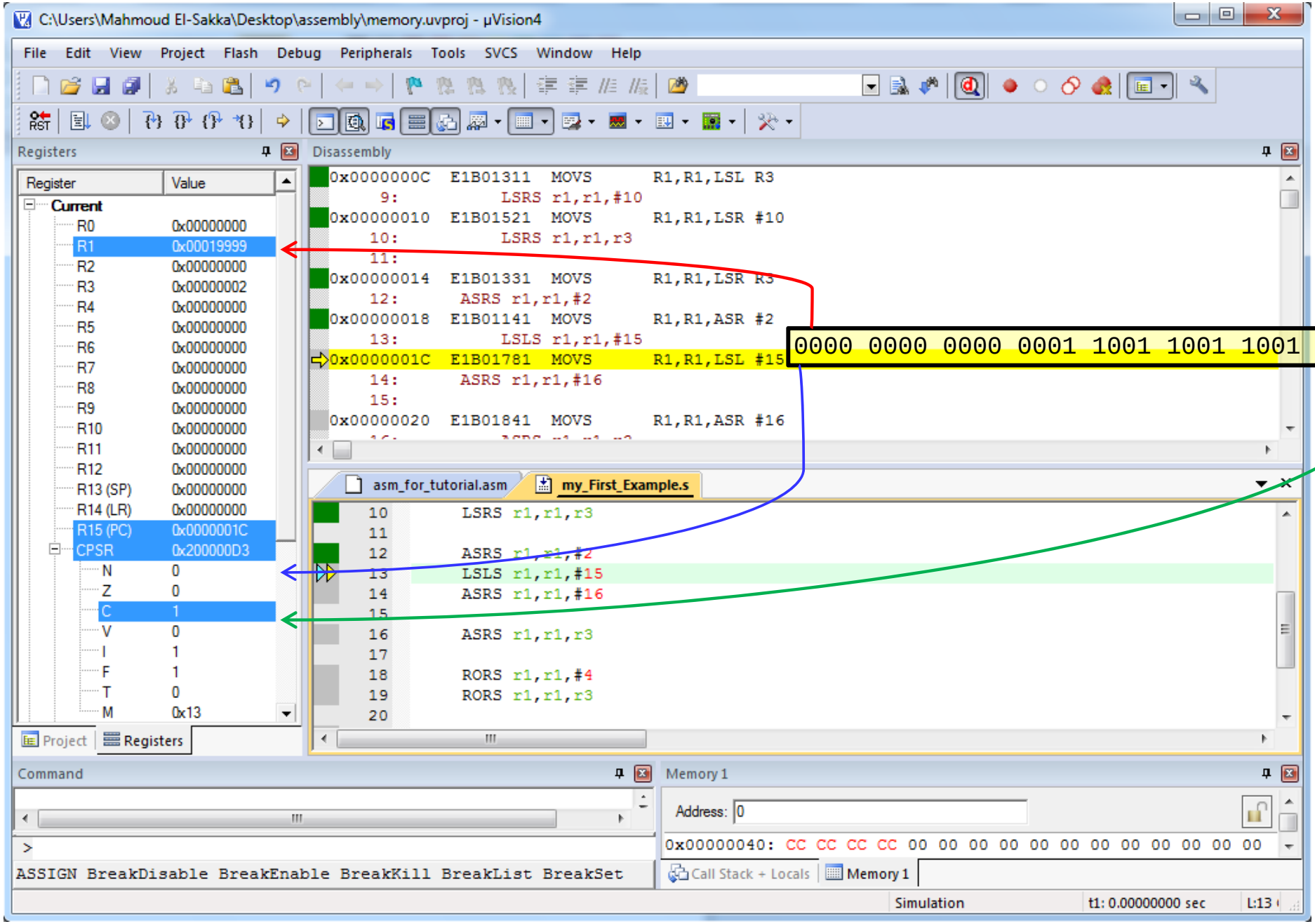
ARM's Data-Processing Instructions (Shift Operations)



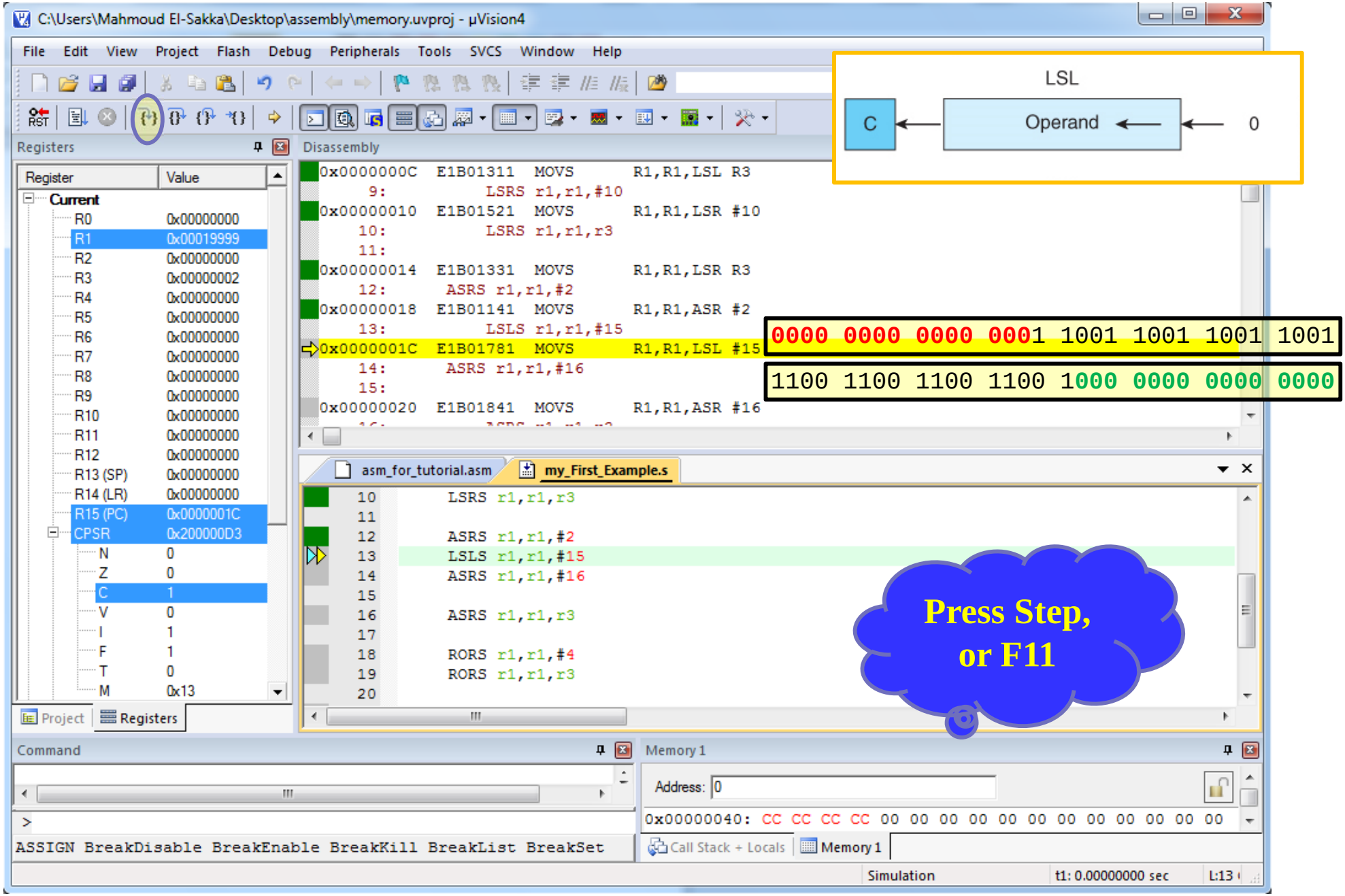
ARM's Data-Processing Instructions (Shift Operations)



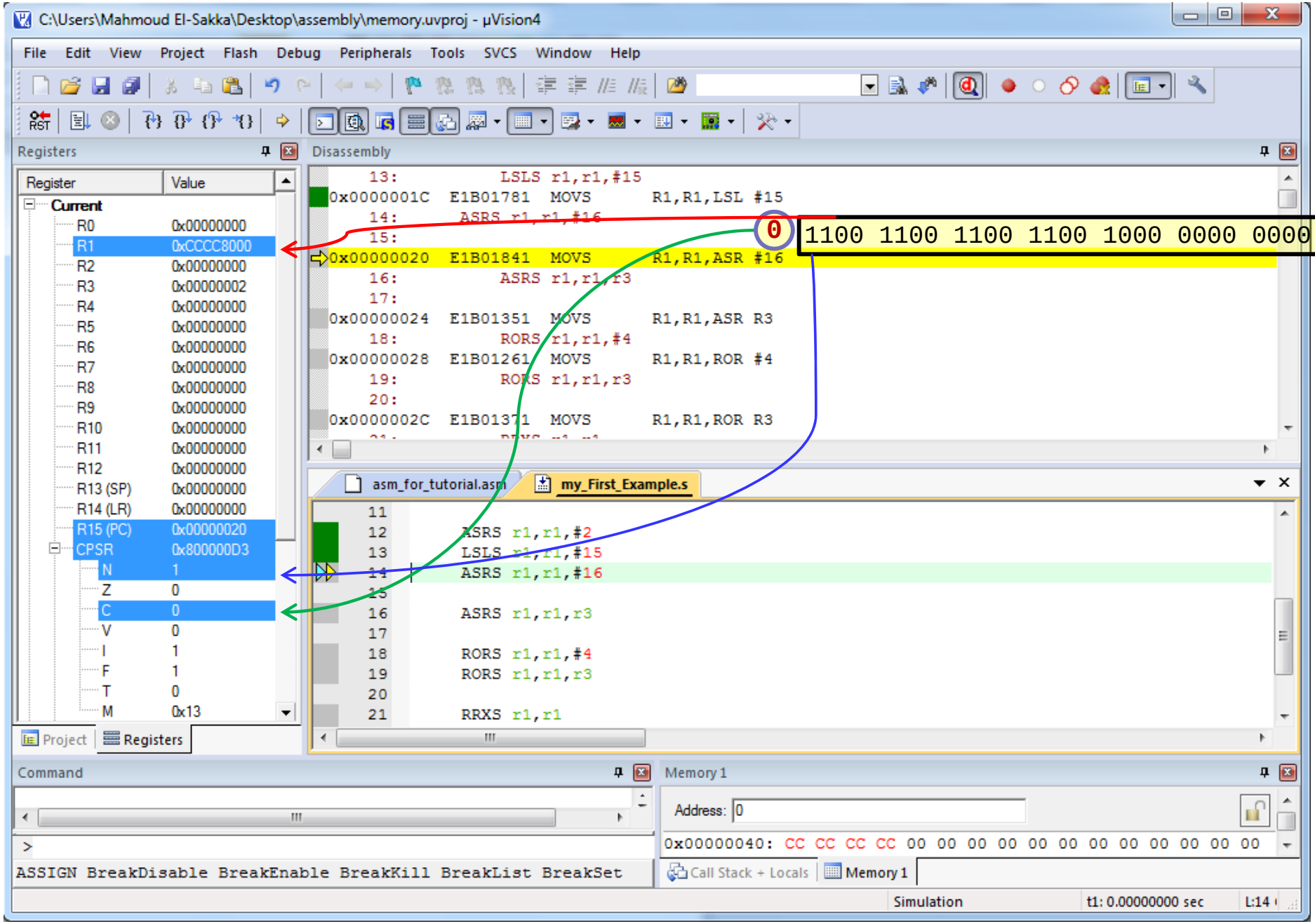
ARM's Data-Processing Instructions (Shift Operations)



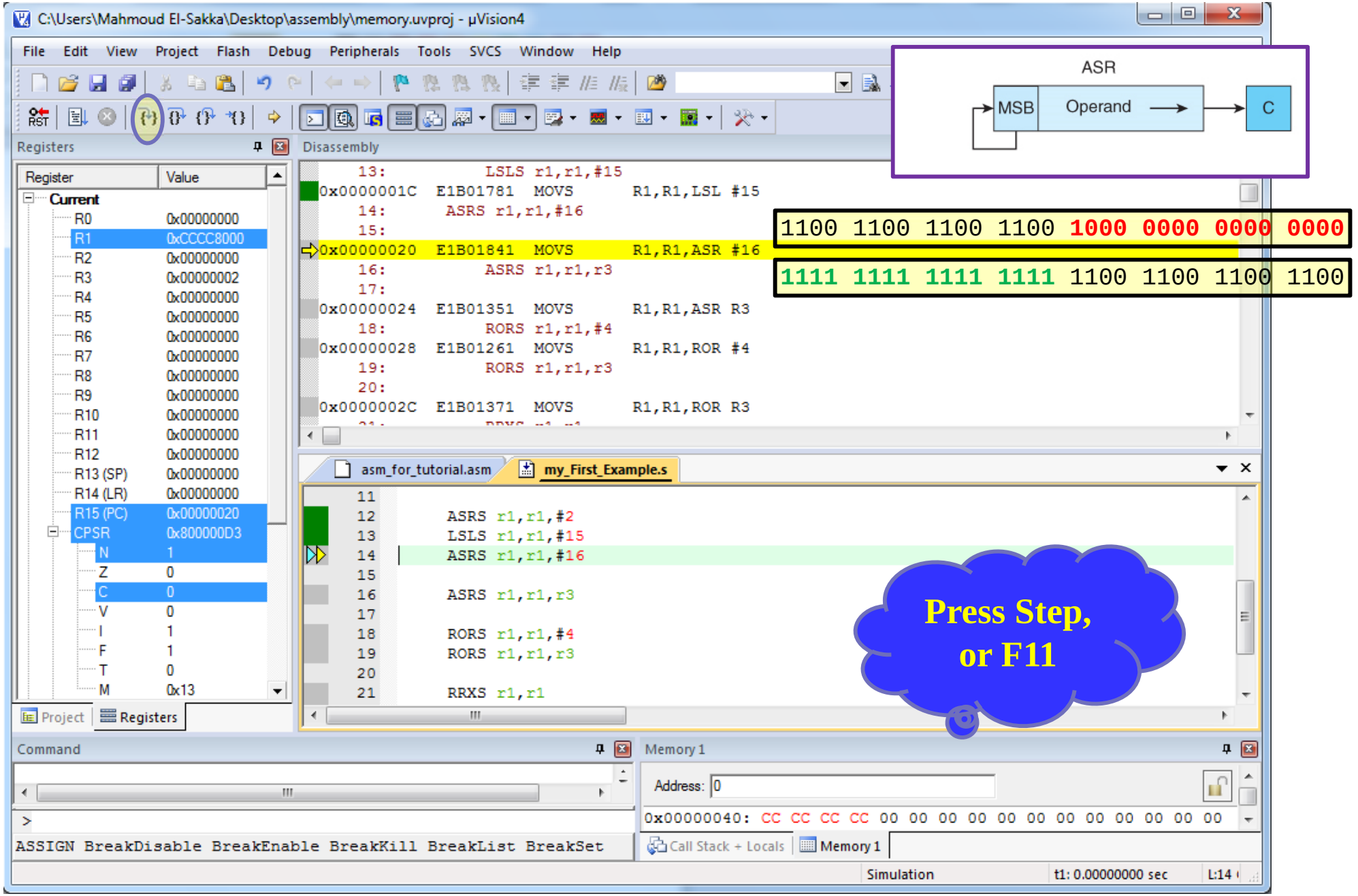
ARM's Data-Processing Instructions (Shift Operations)



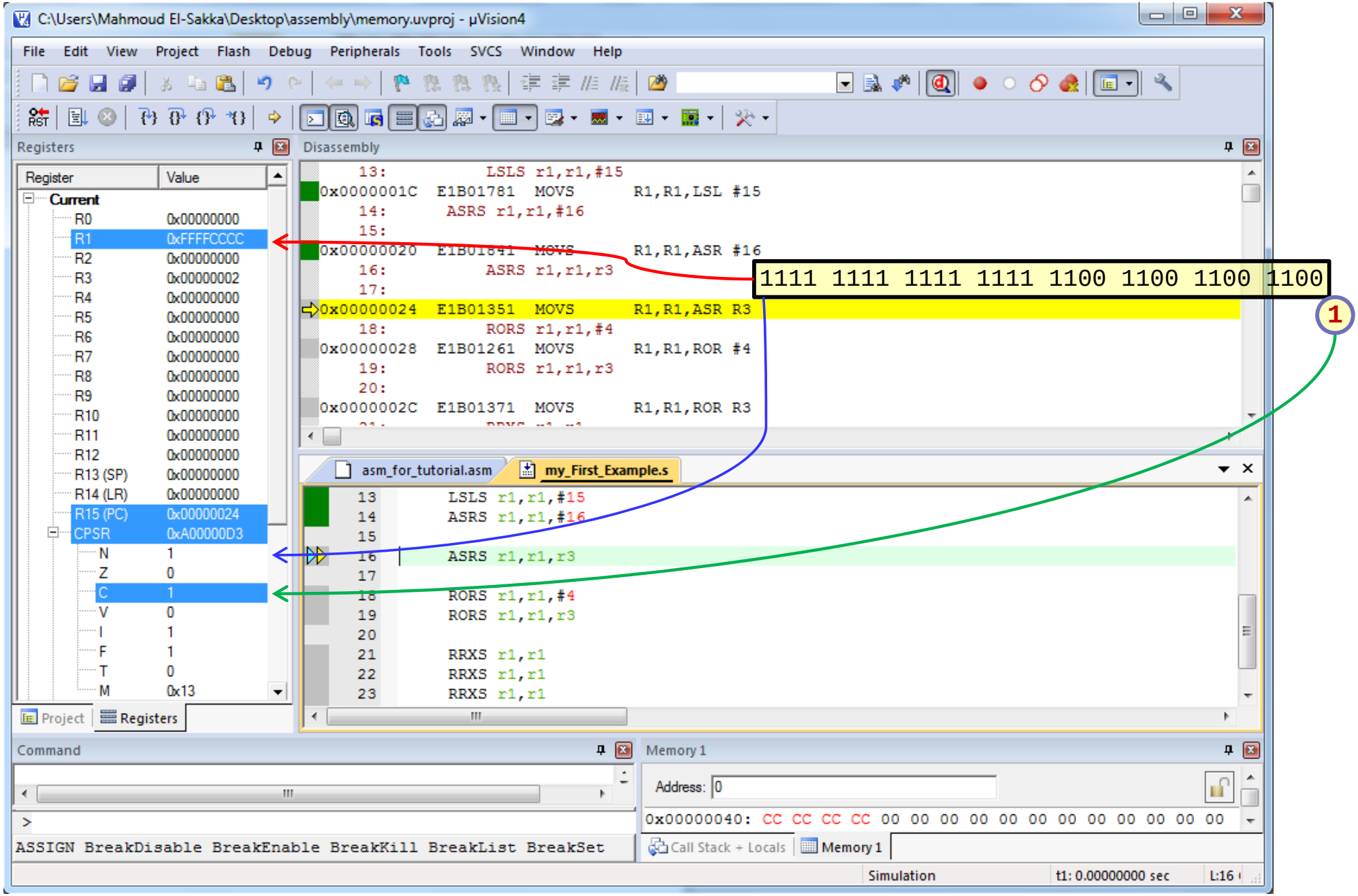
ARM's Data-Processing Instructions (Shift Operations)



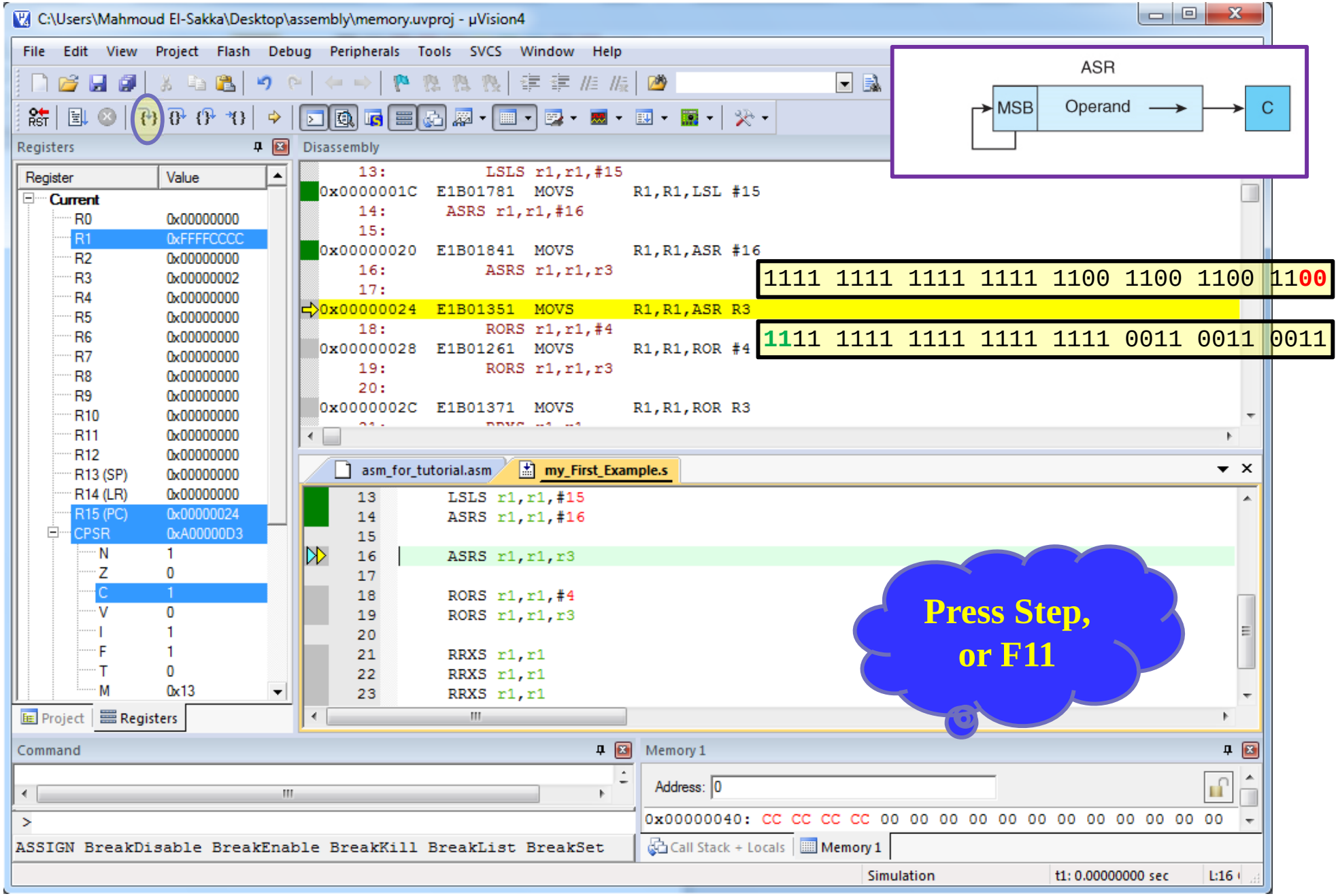
ARM's Data-Processing Instructions (Shift Operations)



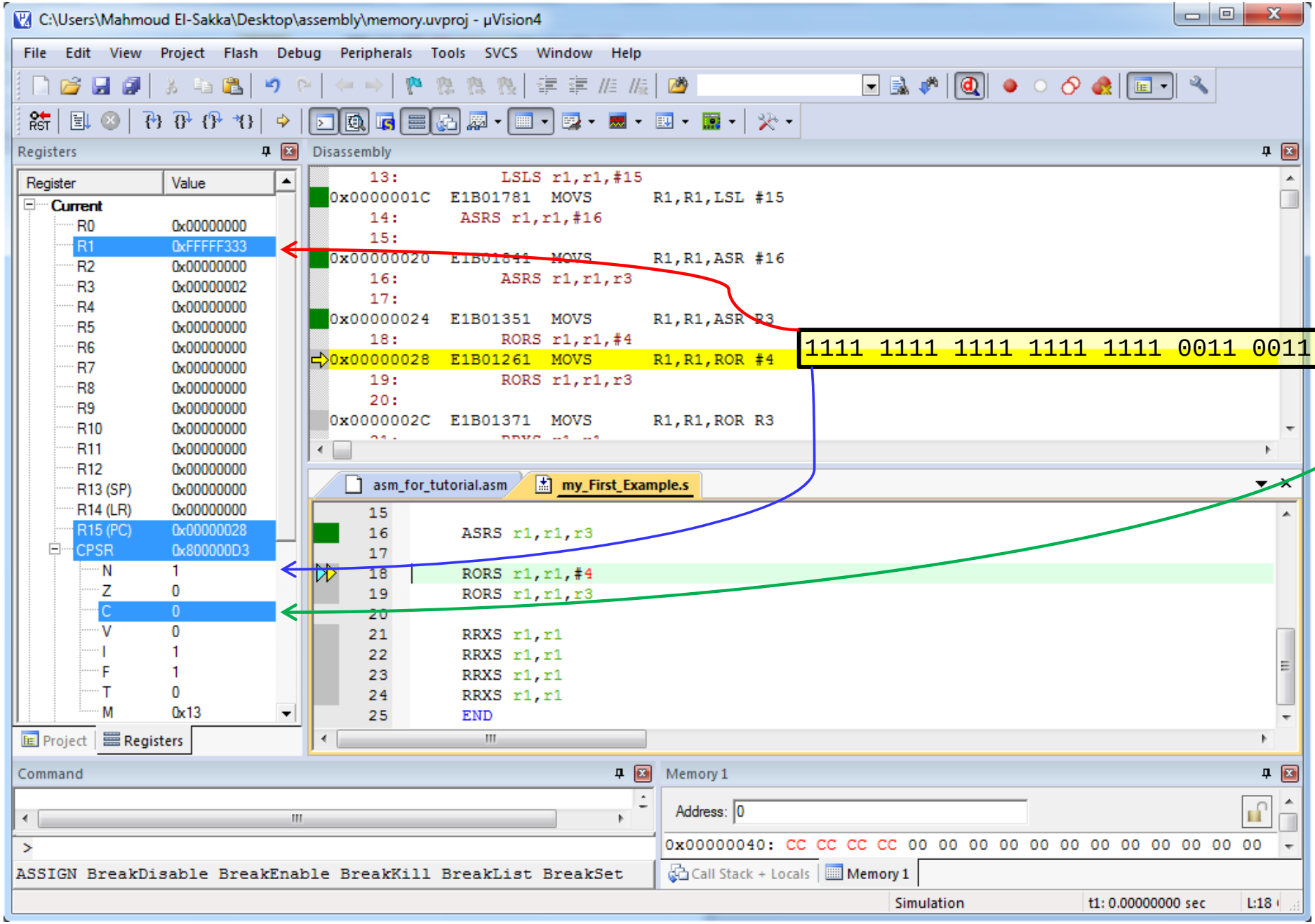
ARM's Data-Processing Instructions (Shift Operations)



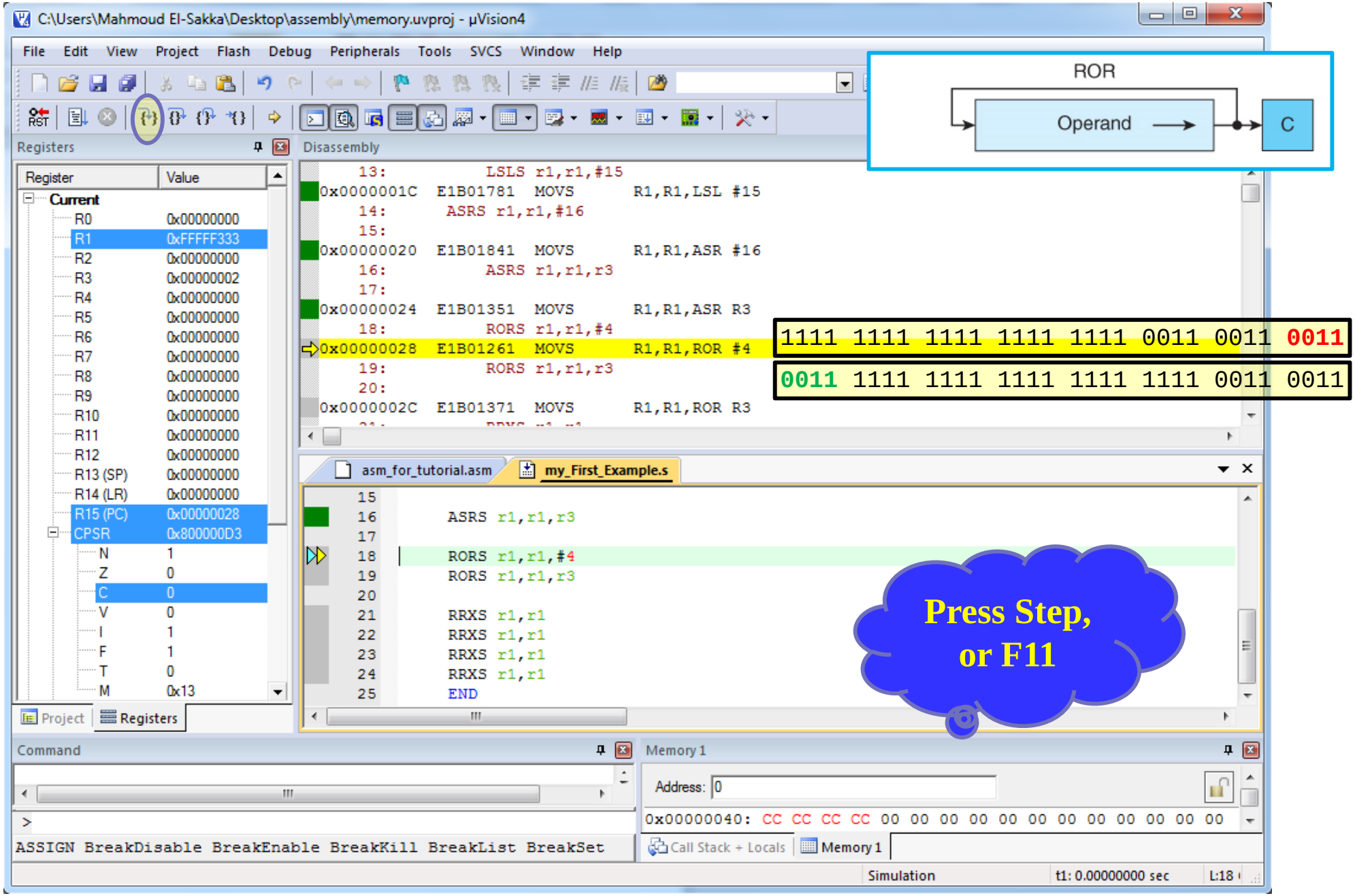
ARM's Data-Processing Instructions (Shift Operations)



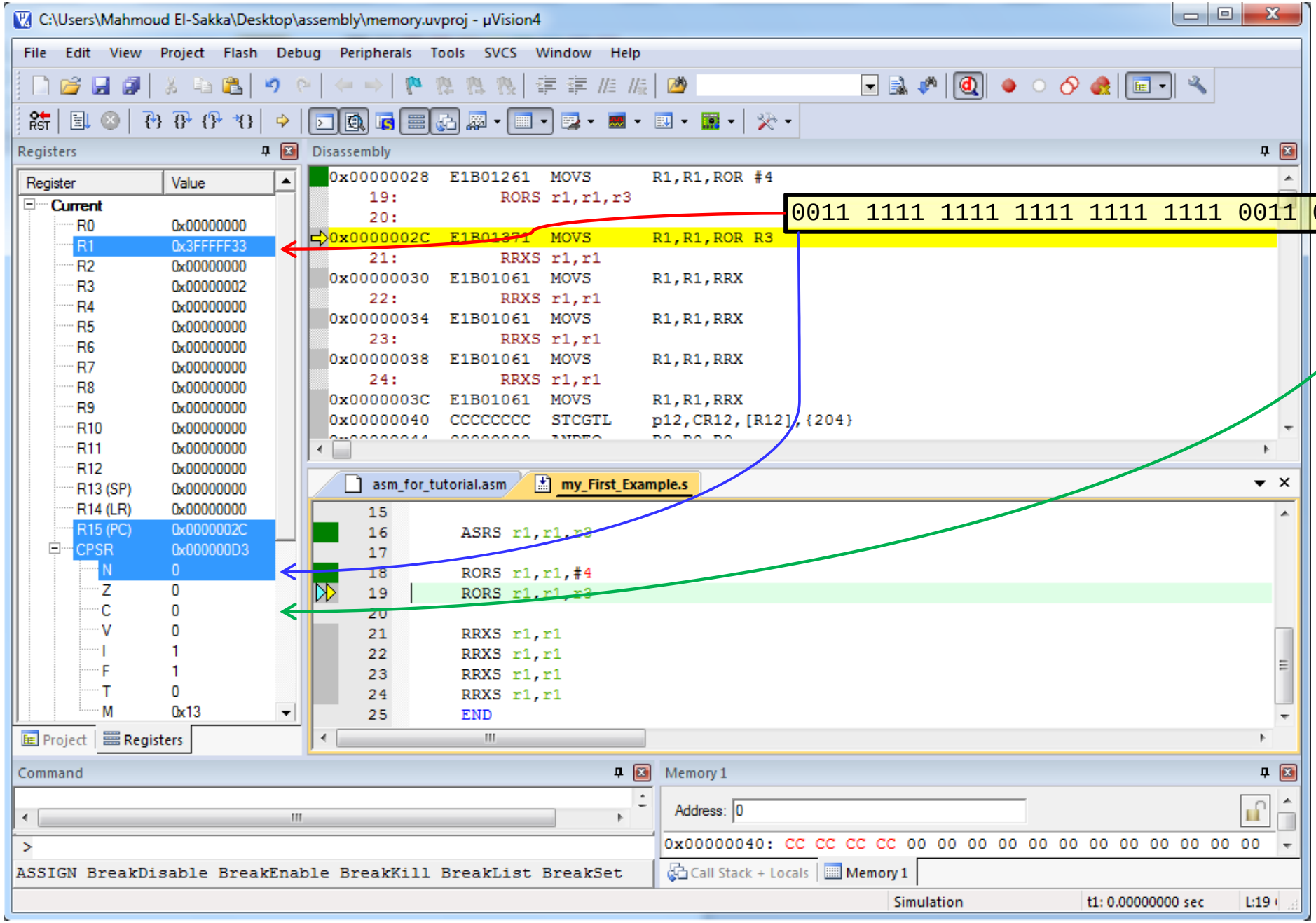
ARM's Data-Processing Instructions (Shift Operations)



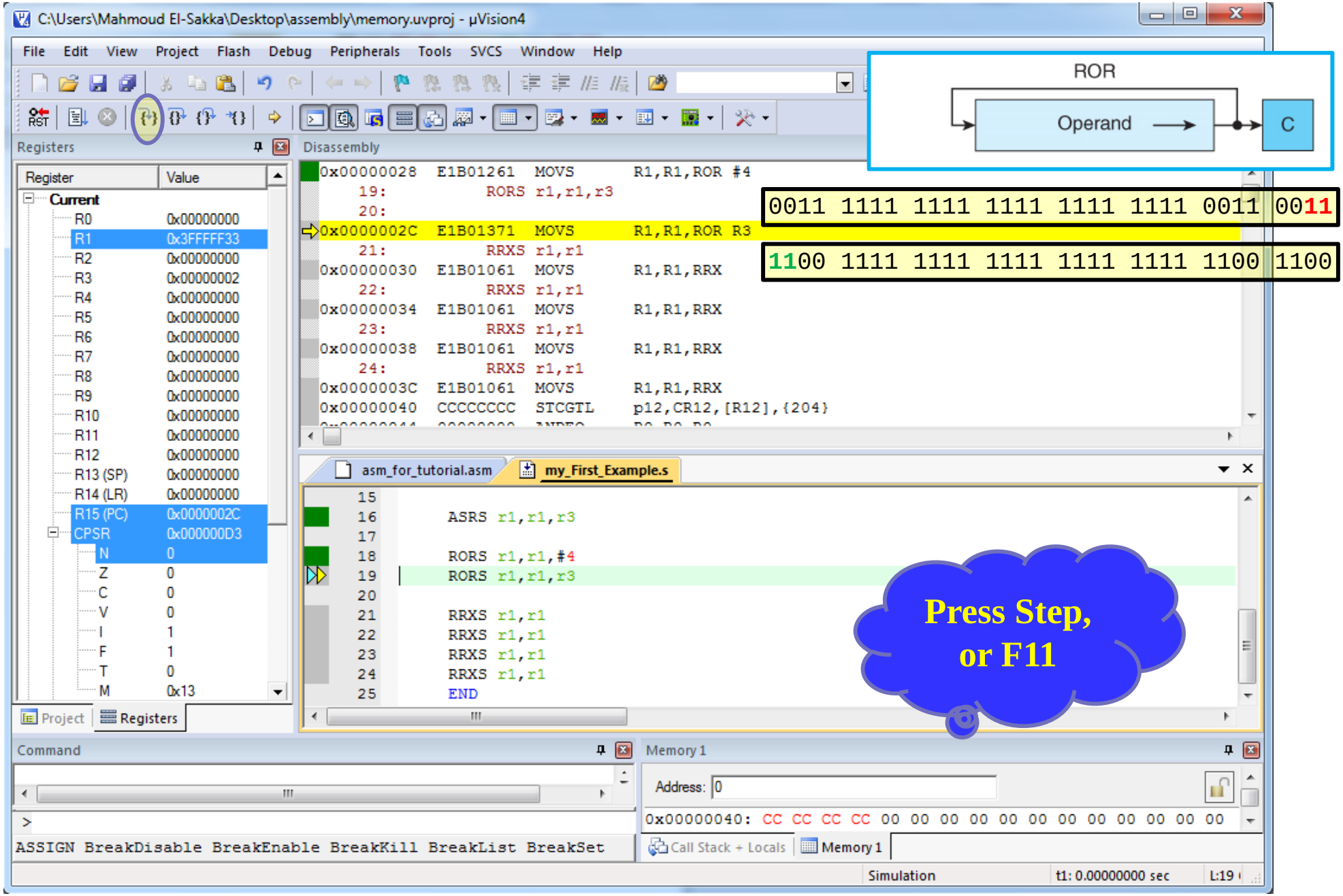
ARM's Data-Processing Instructions (Shift Operations)



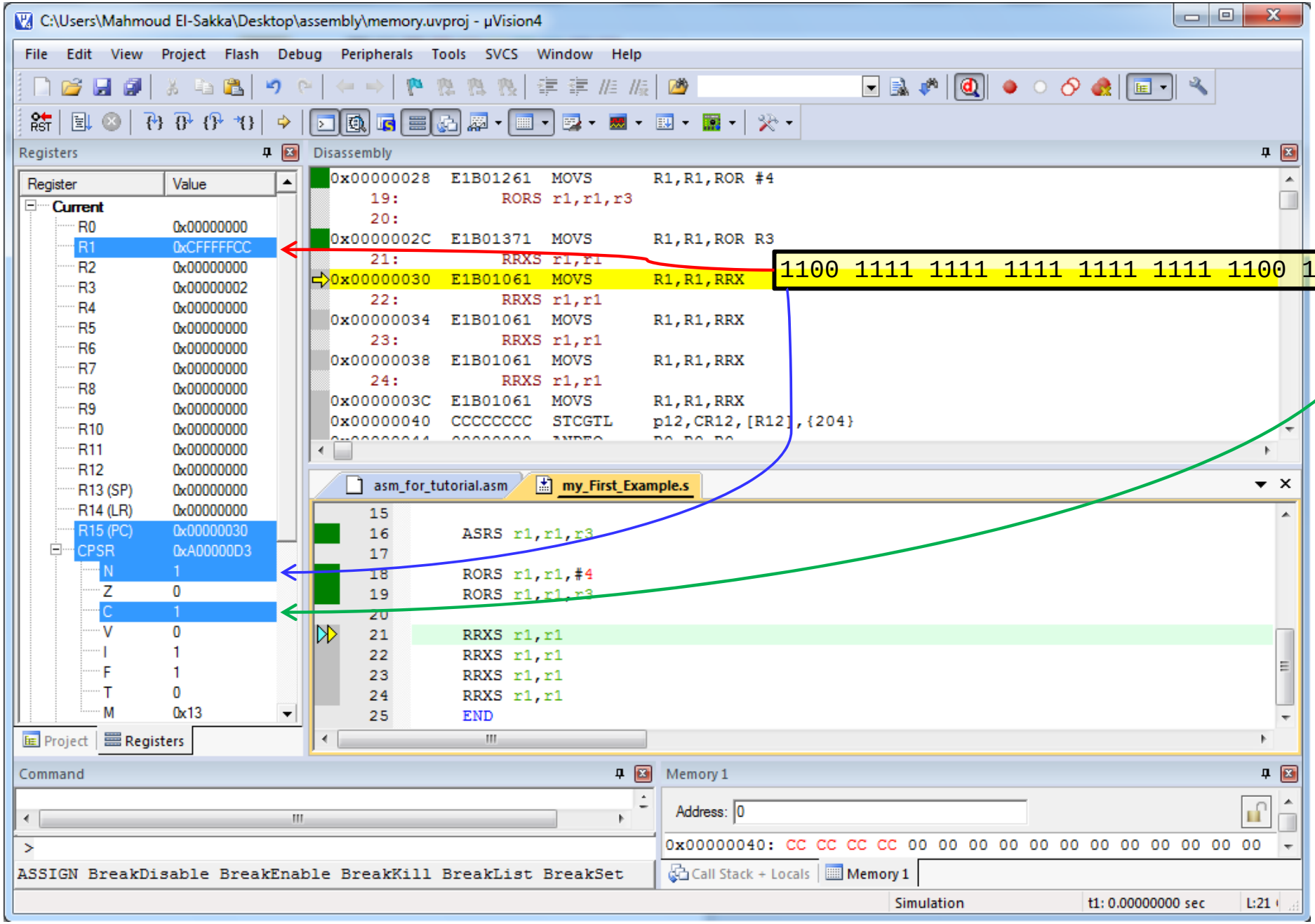
ARM's Data-Processing Instructions (Shift Operations)



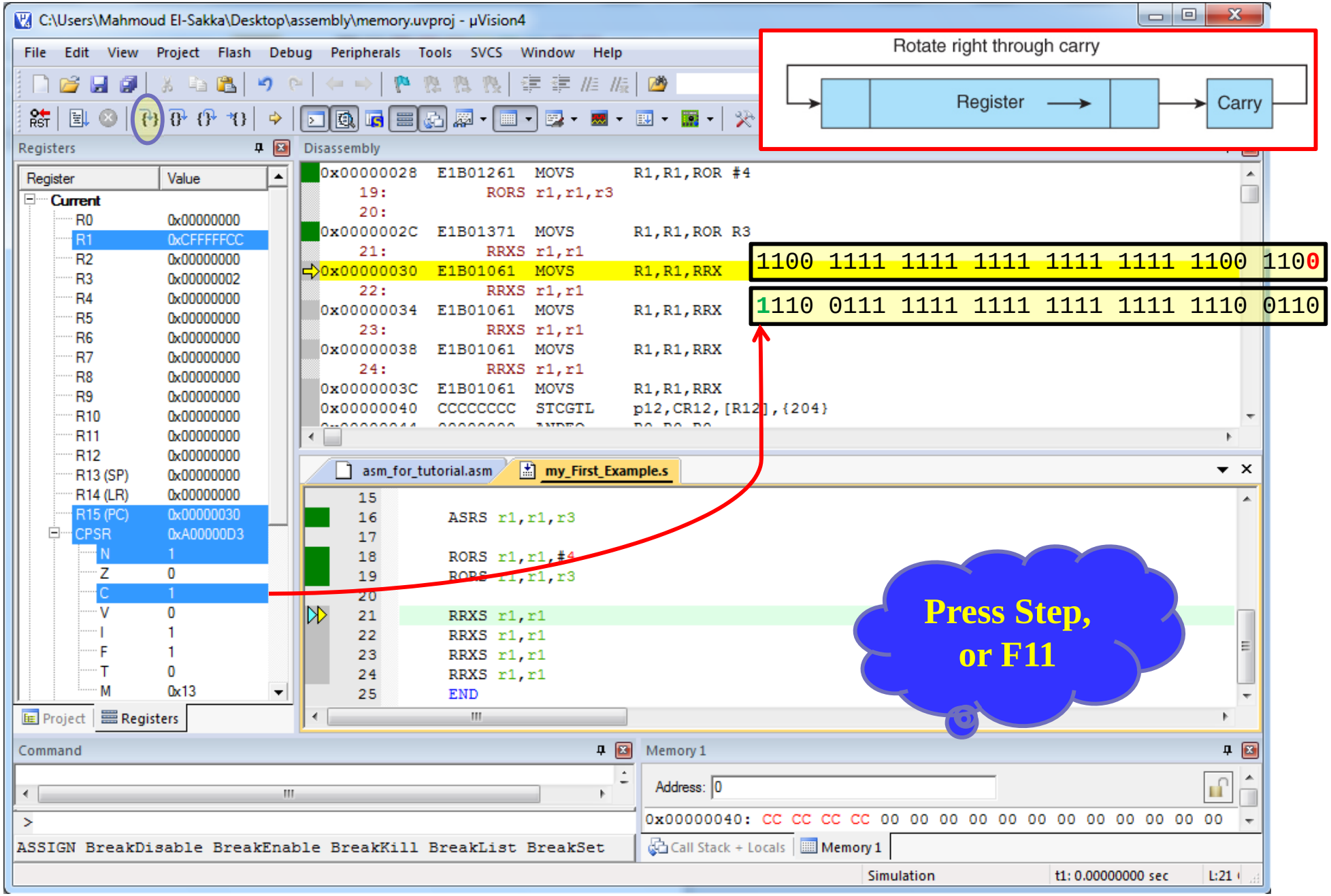
ARM's Data-Processing Instructions (Shift Operations)



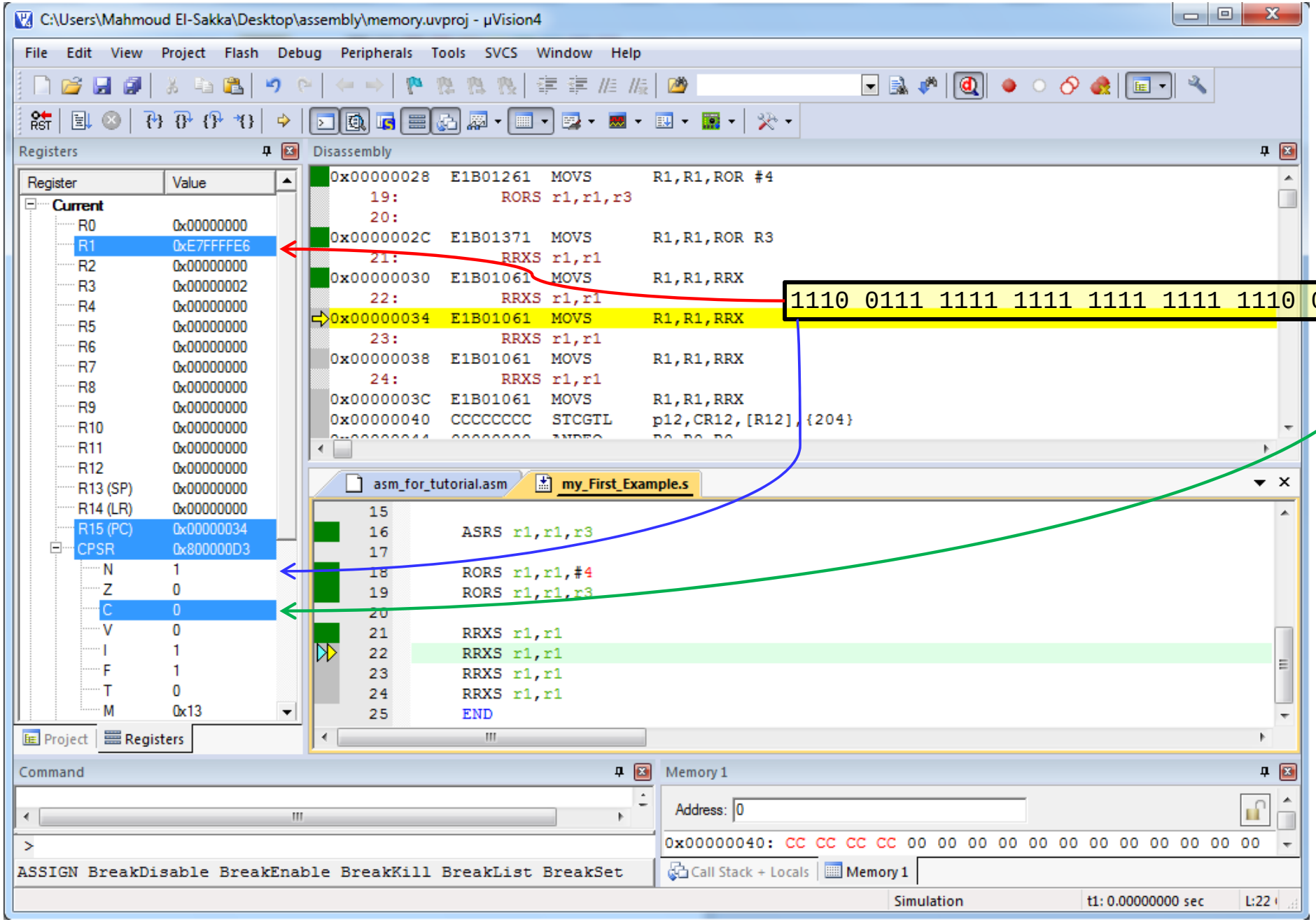
ARM's Data-Processing Instructions (Shift Operations)



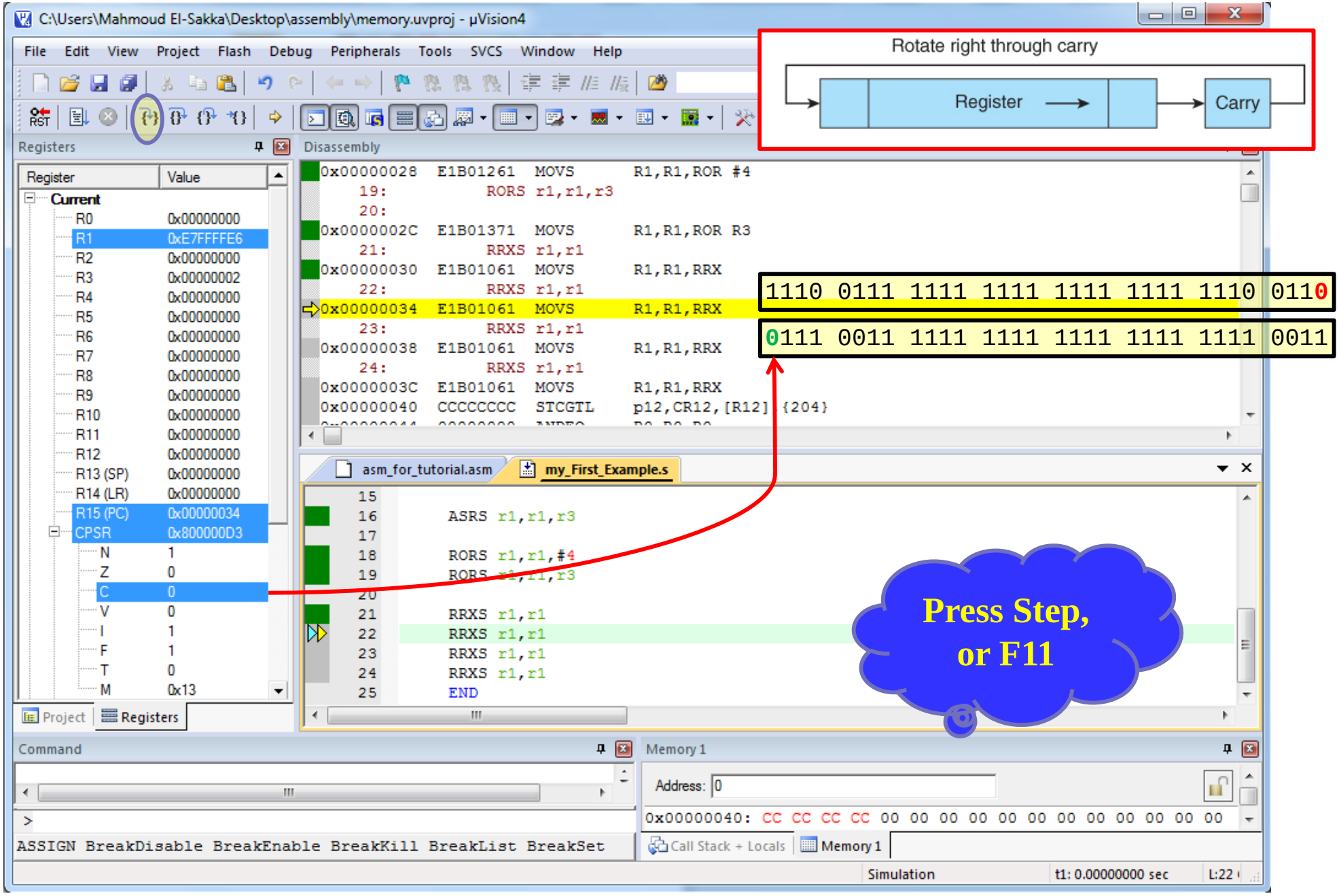
ARM's Data-Processing Instructions (Shift Operations)



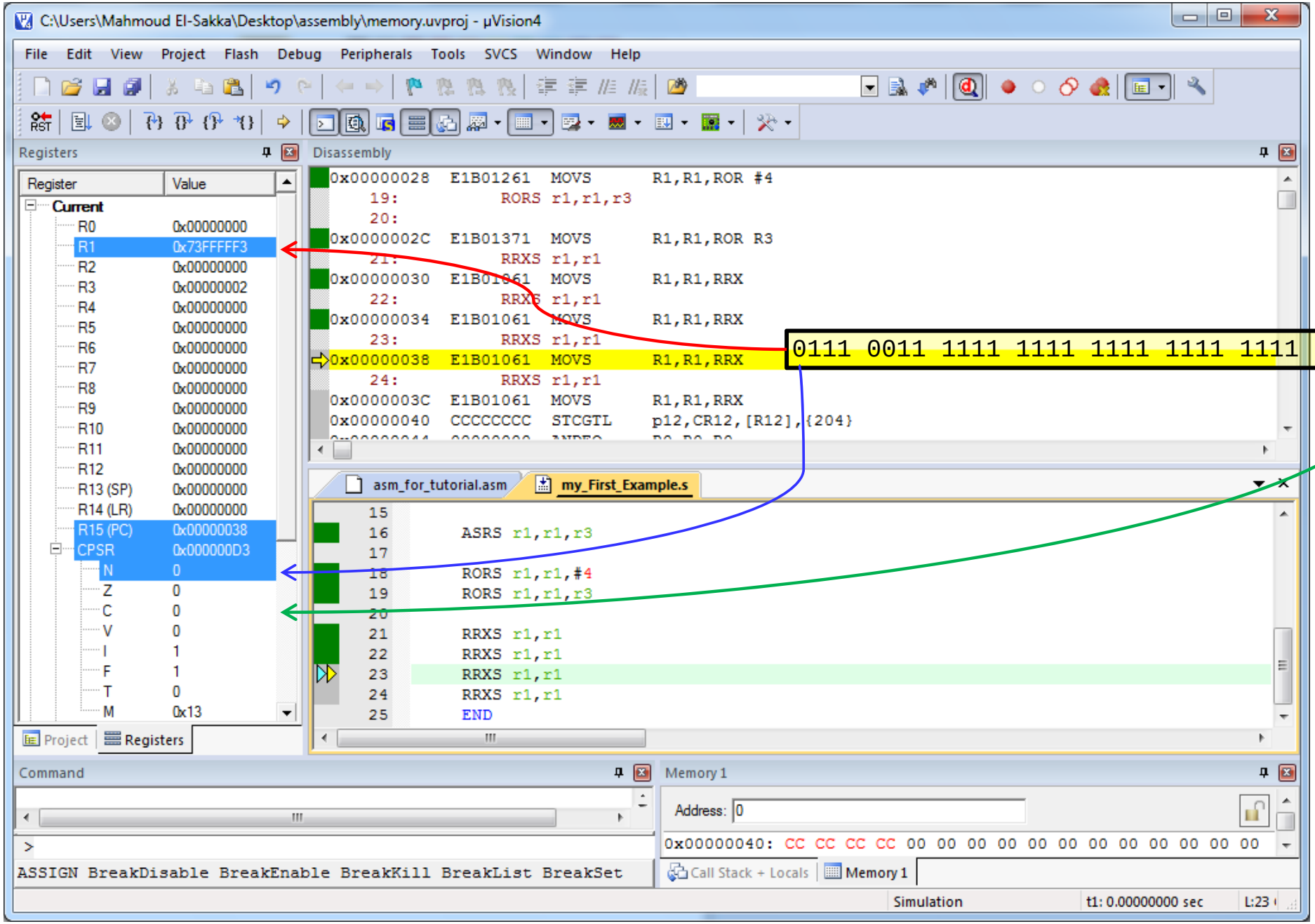
ARM's Data-Processing Instructions (Shift Operations)



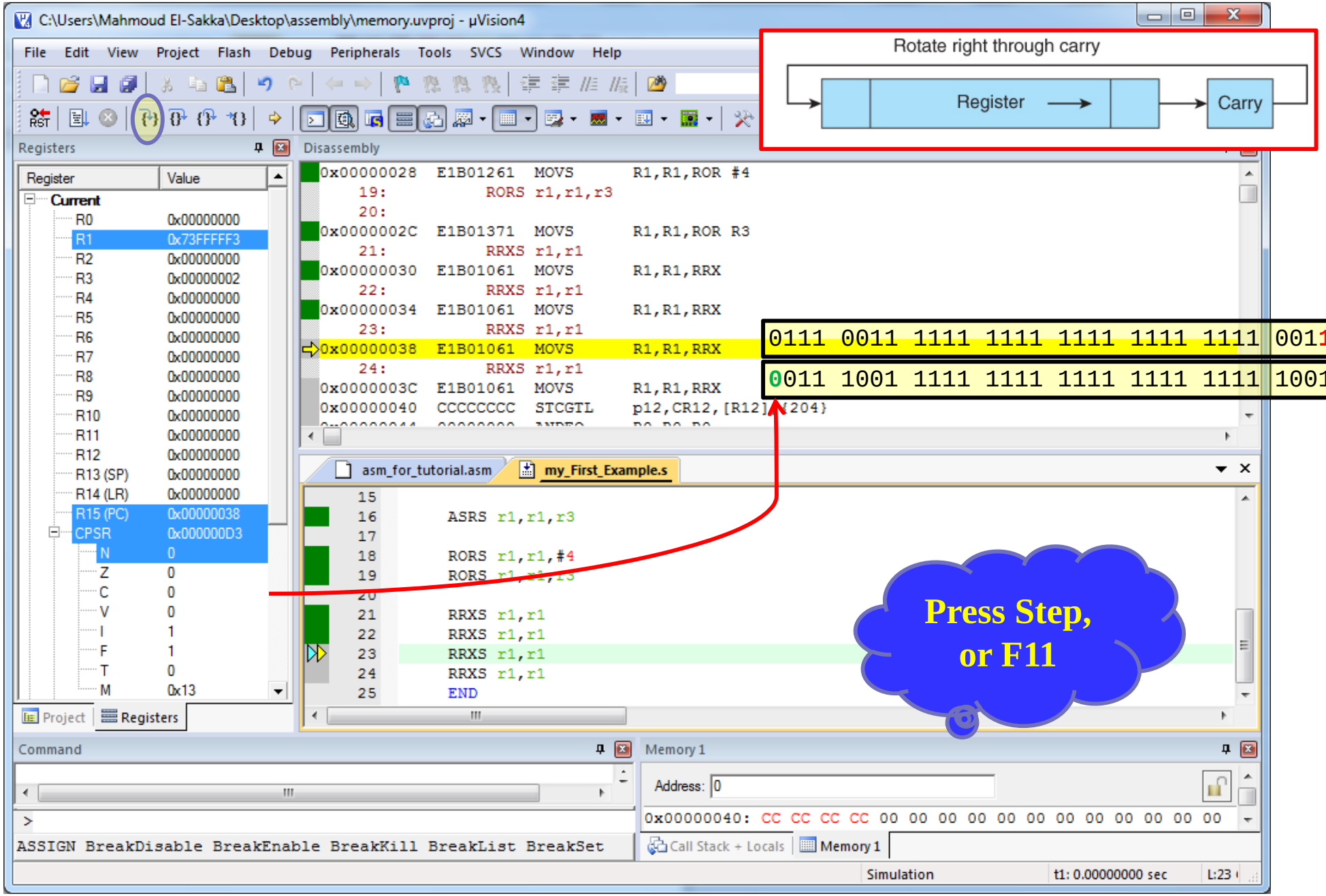
ARM's Data-Processing Instructions (Shift Operations)



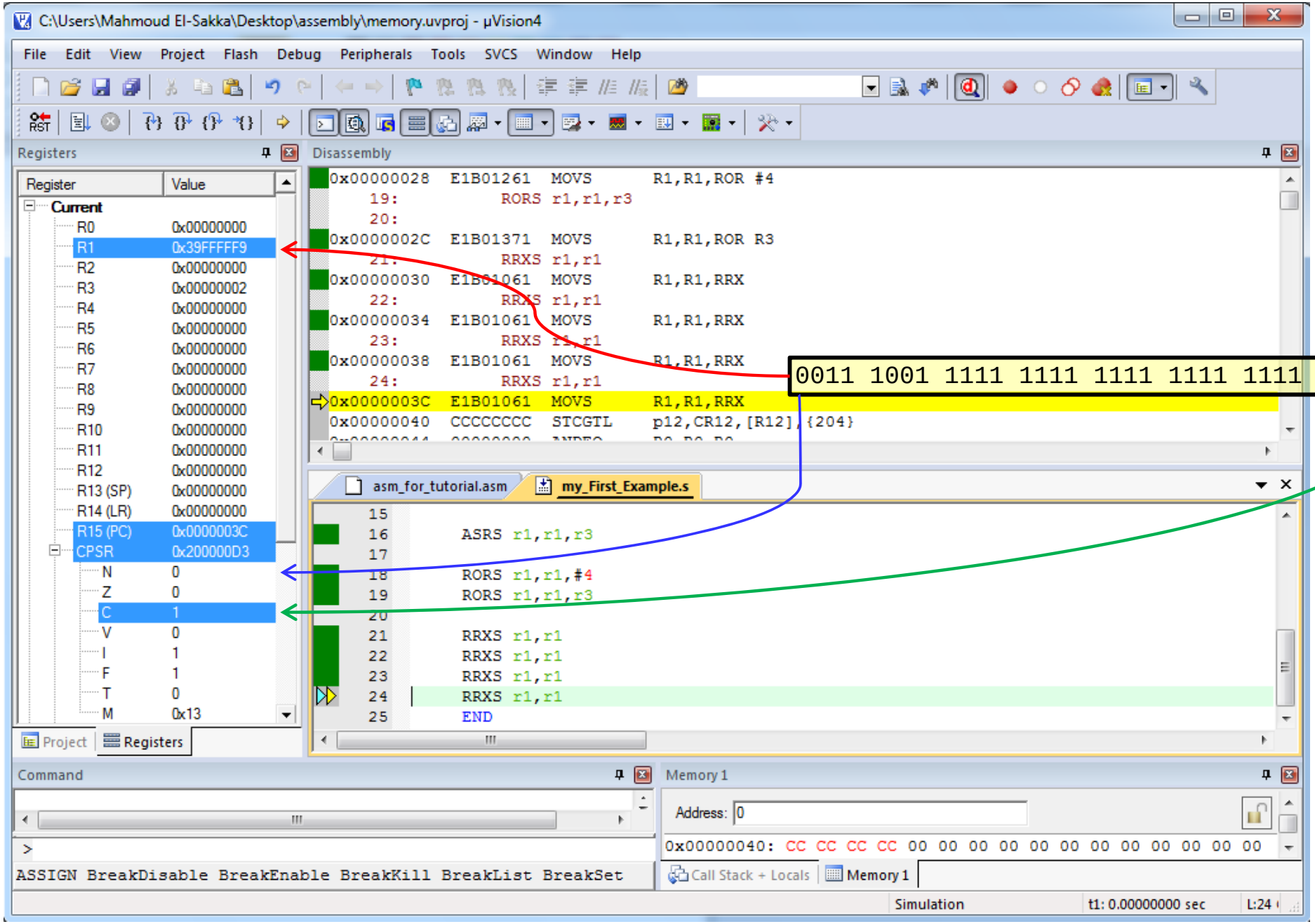
ARM's Data-Processing Instructions (Shift Operations)



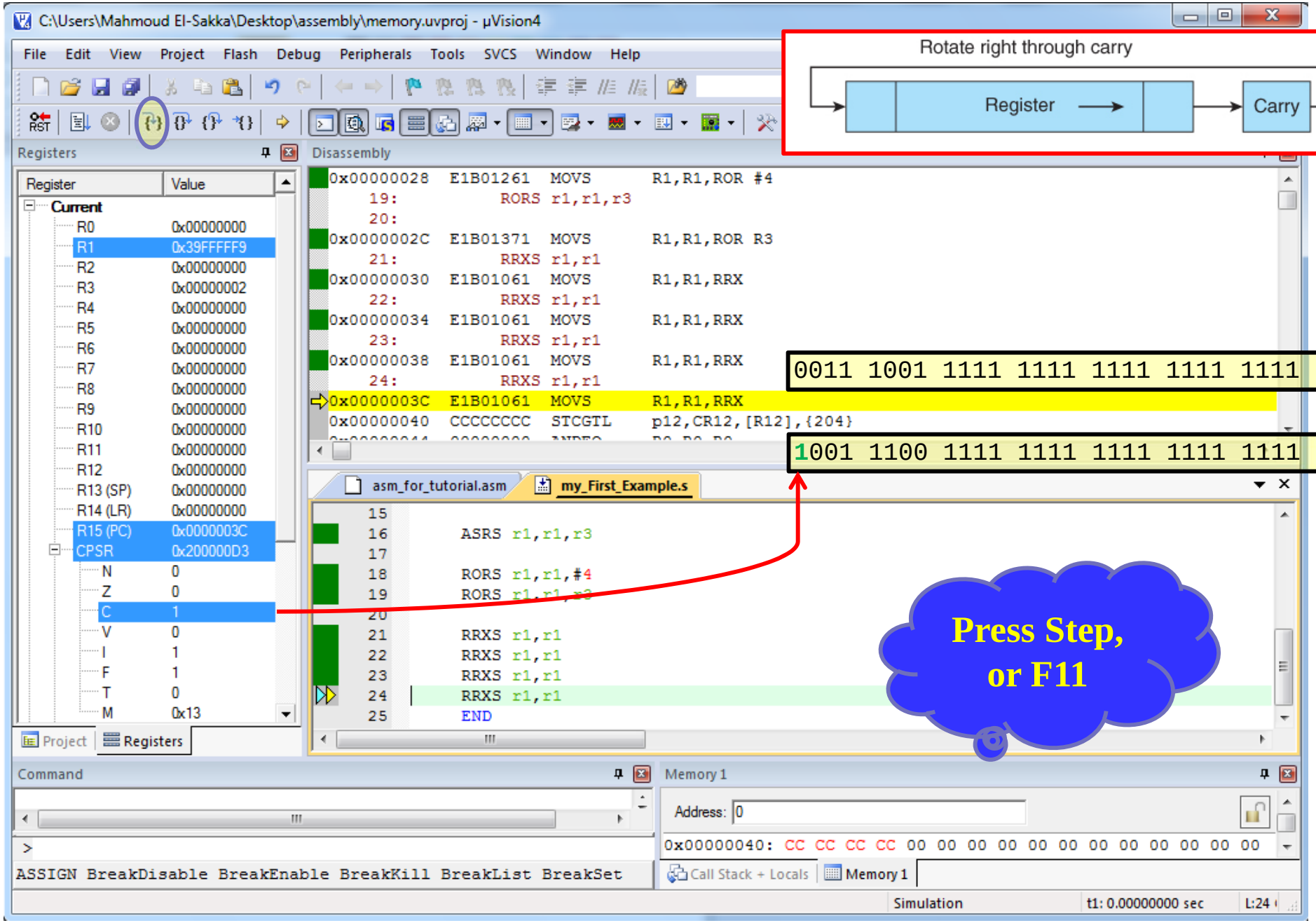
ARM's Data-Processing Instructions (Shift Operations)



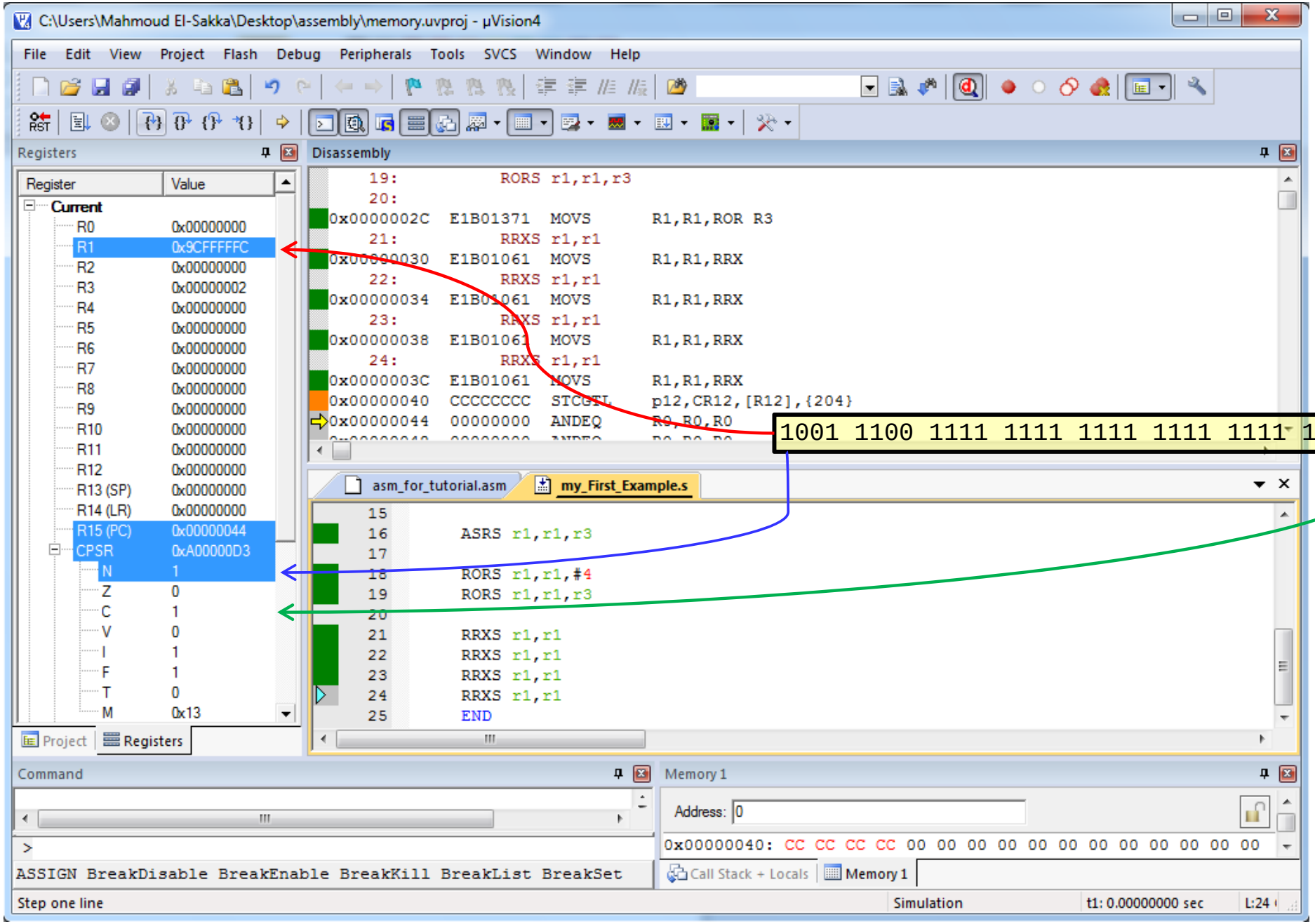
ARM's Data-Processing Instructions (Shift Operations)



ARM's Data-Processing Instructions (Shift Operations)

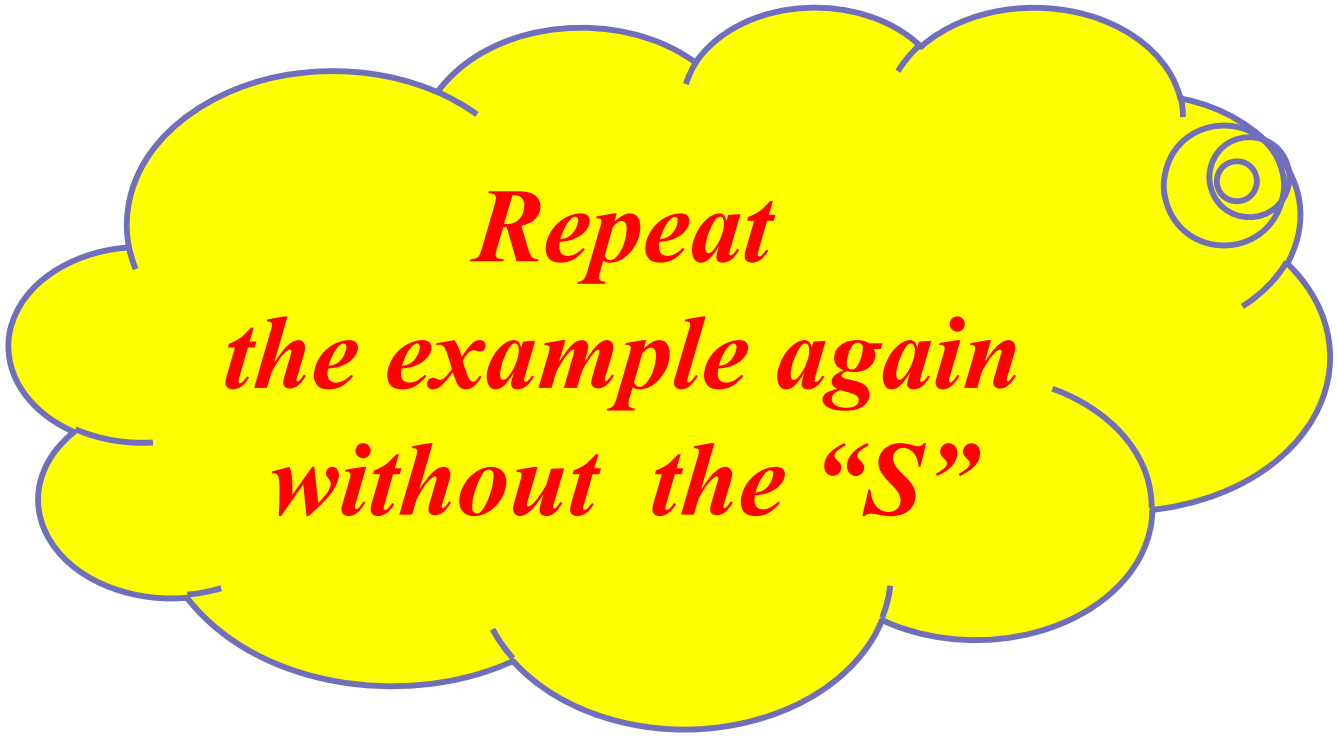


ARM's Data-Processing Instructions (Shift Operations)

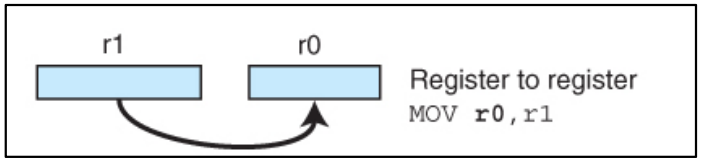


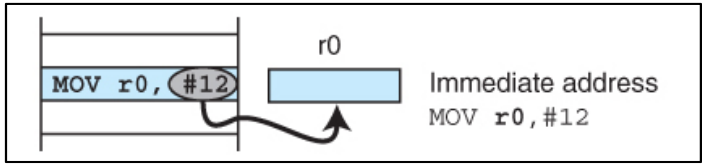
ARM's Data-Processing Instructions (Shift Operations)

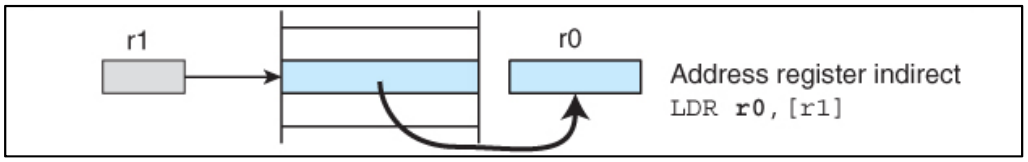
```
AREA prog1, code, READONLY
ENTRY
MOV r3,#2
LDR r1, =0xCCCCCCCC ;in binary 1100 1100 1100 1100 1100 1100 1100 1100

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
```

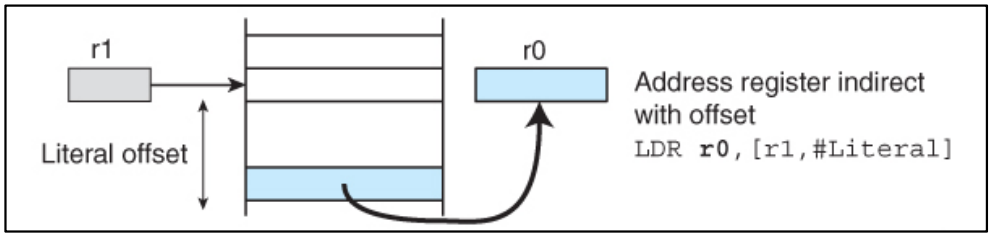


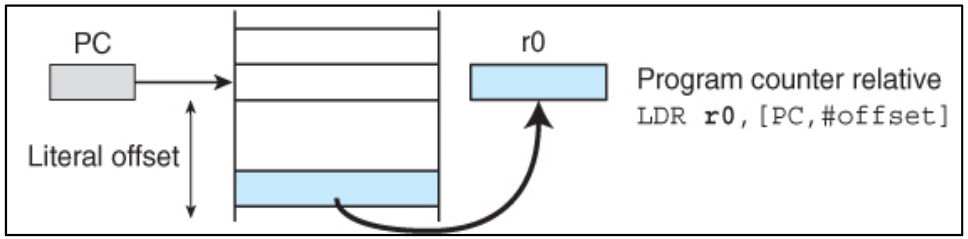
ARM Addressing Modes

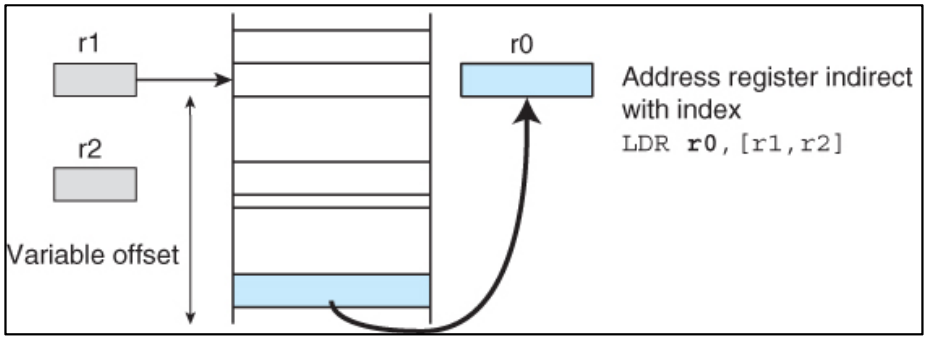


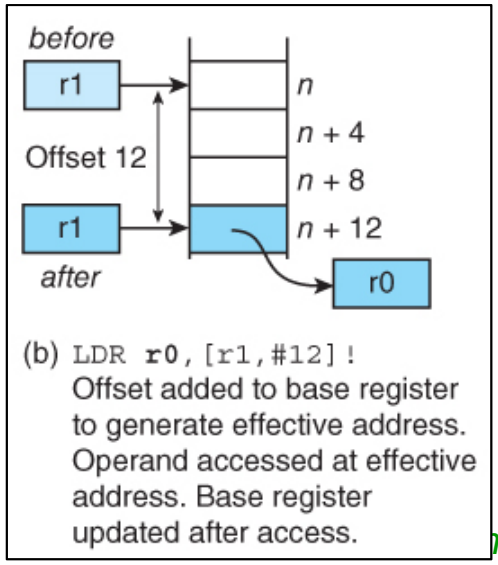




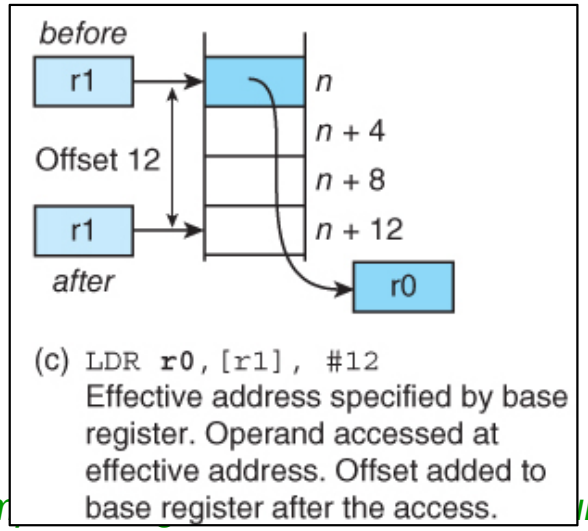








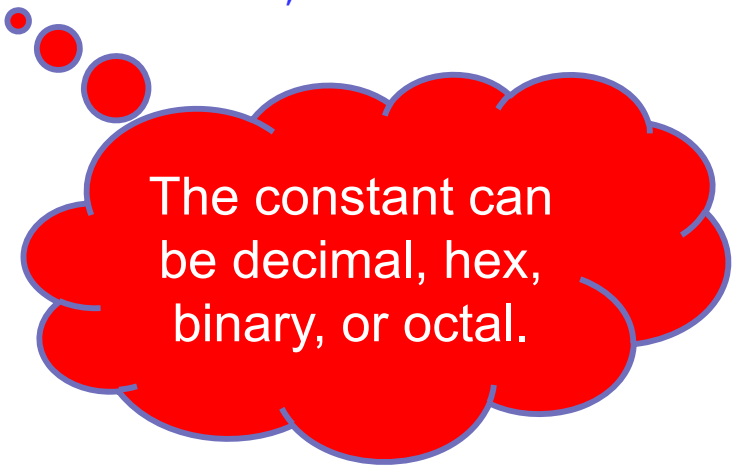
(b) `LDR r0,[r1,#12]!`
Offset added to base register to generate effective address. Operand accessed at effective address. Base register updated after access.



(c) `LDR r0,[r1], #12`
Effective address specified by base register. Operand accessed at effective address. Offset added to base register after the access.

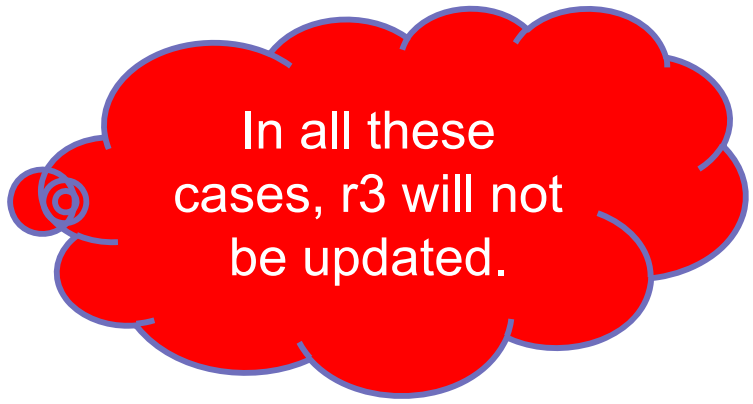
ARM Addressing Modes

```
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
```



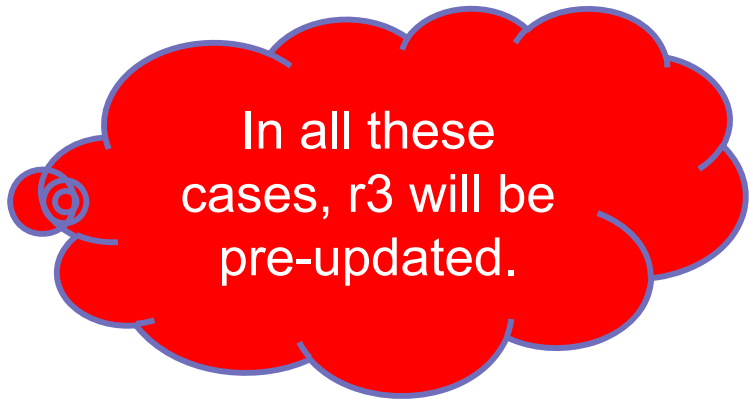
ARM Addressing Modes

```
;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
```



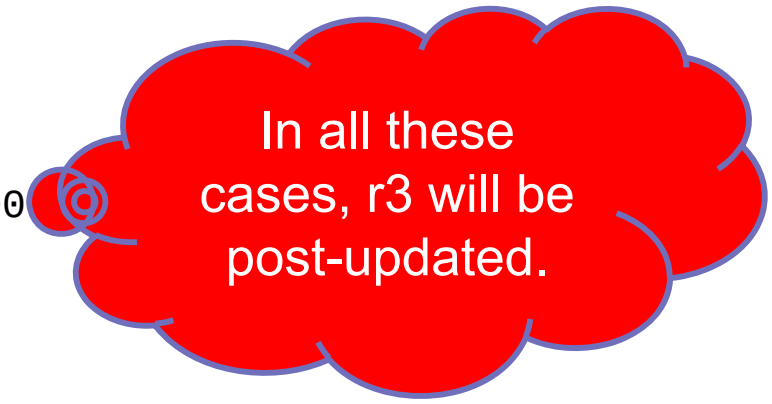
ARM Addressing Modes

```
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
```

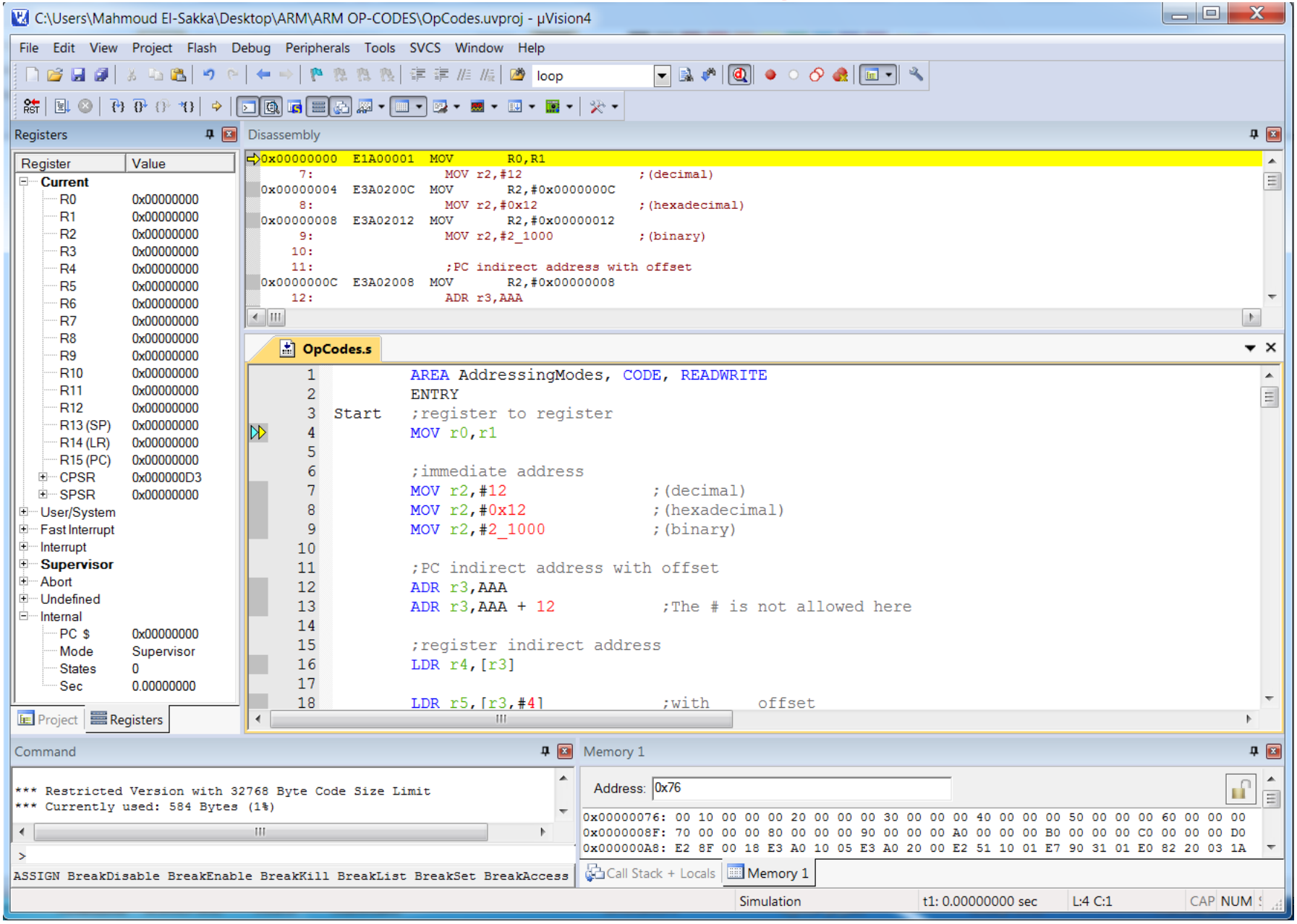


ARM Addressing Modes

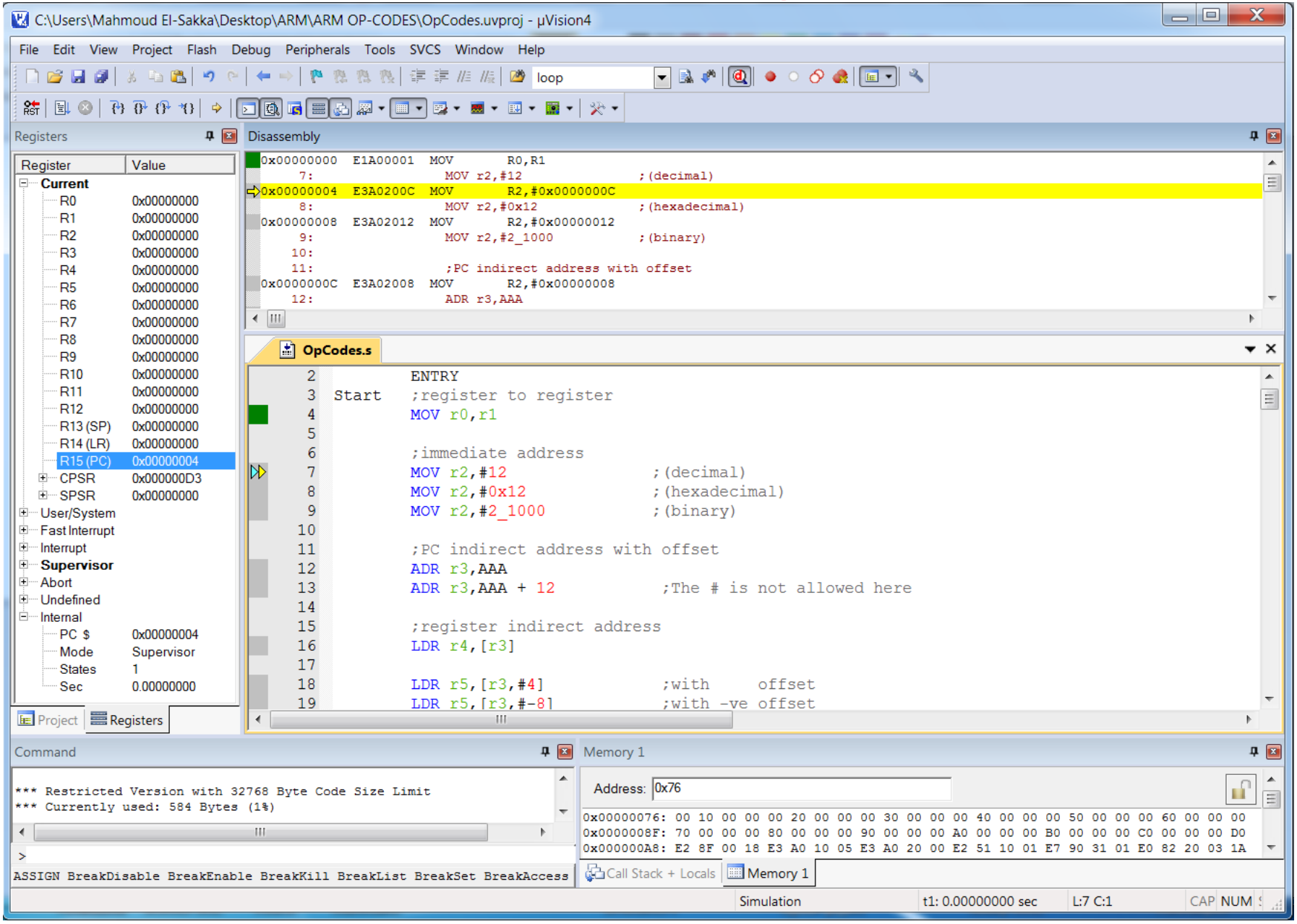
```
      LDR r5,[r3],#4              ;with     offset and autoindexing post-update
      LDR r5,[r3],#-8             ;with -ve offset and autoindexing post-update
      LDR r5,[r3],#(-4*2+40)/4    ;the offset can be constant expression
                                  ;to be evaluated by the assembler

      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
```



ARM Addressing Modes



ARM Addressing Modes



ARM Addressing Modes

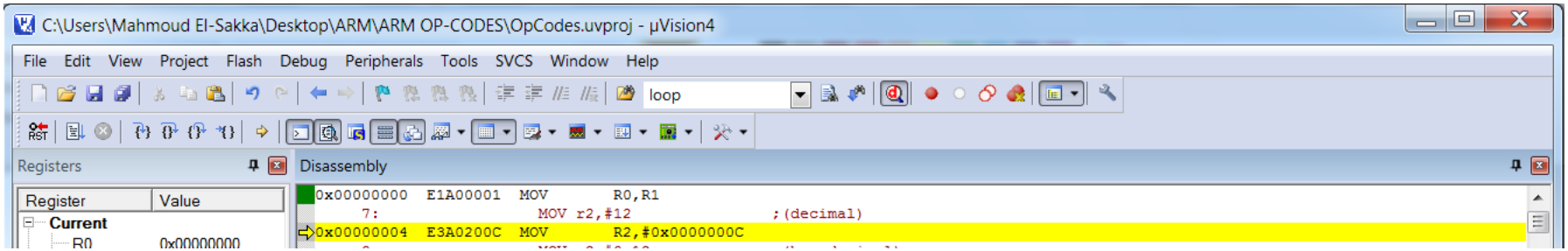
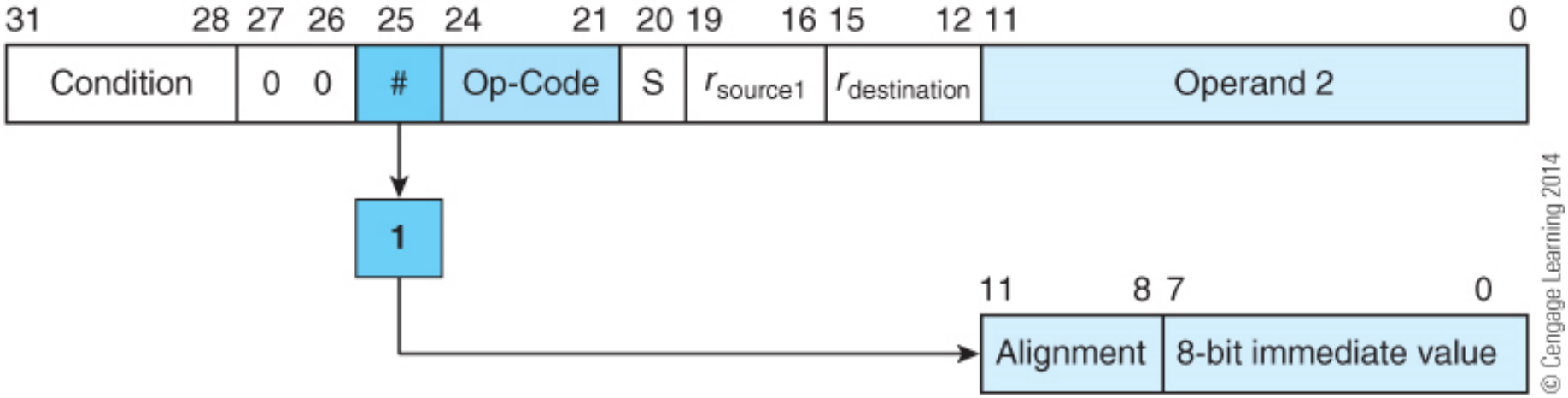


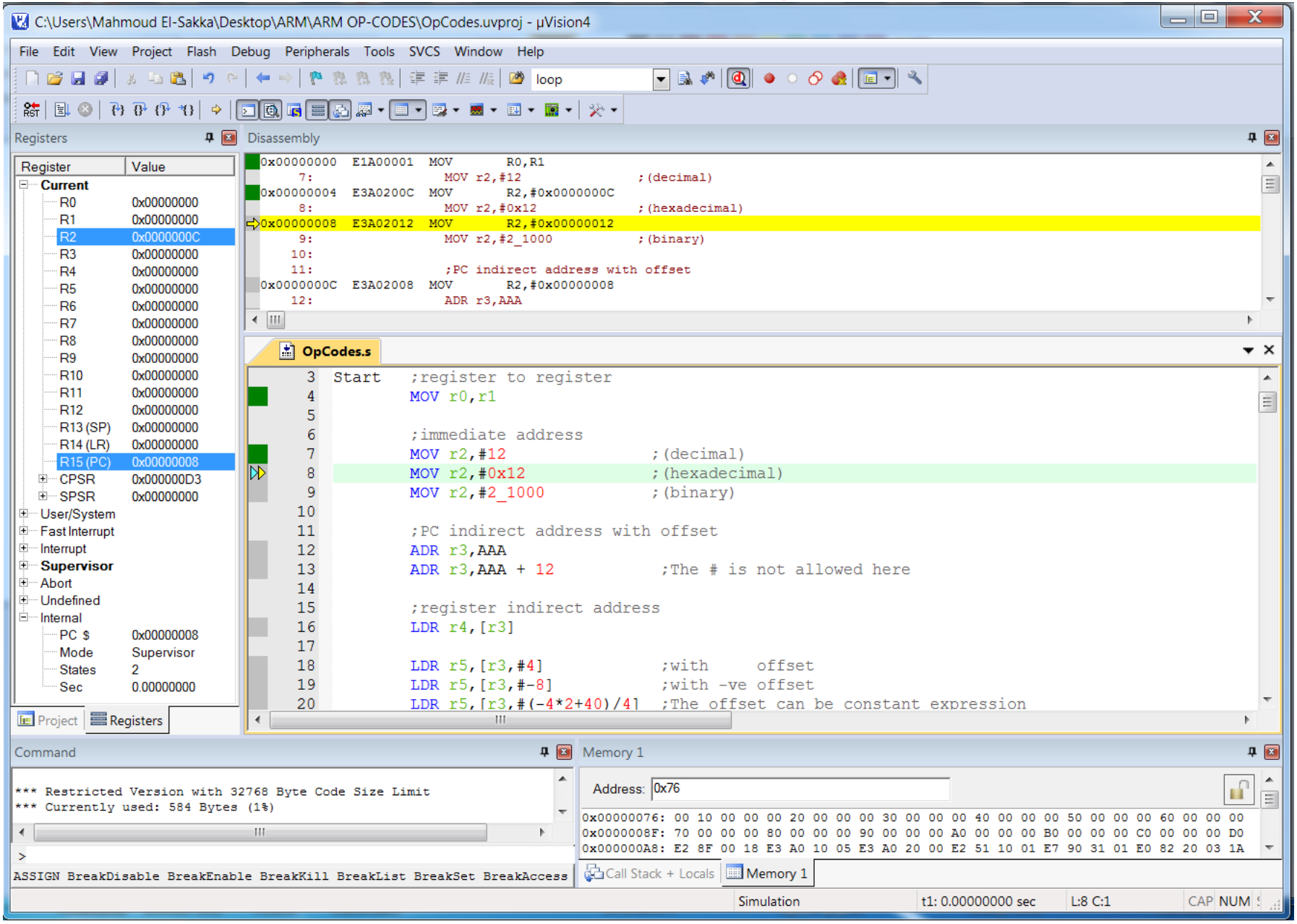
FIGURE 3.28 Diagram of ARM's literal operand encoding



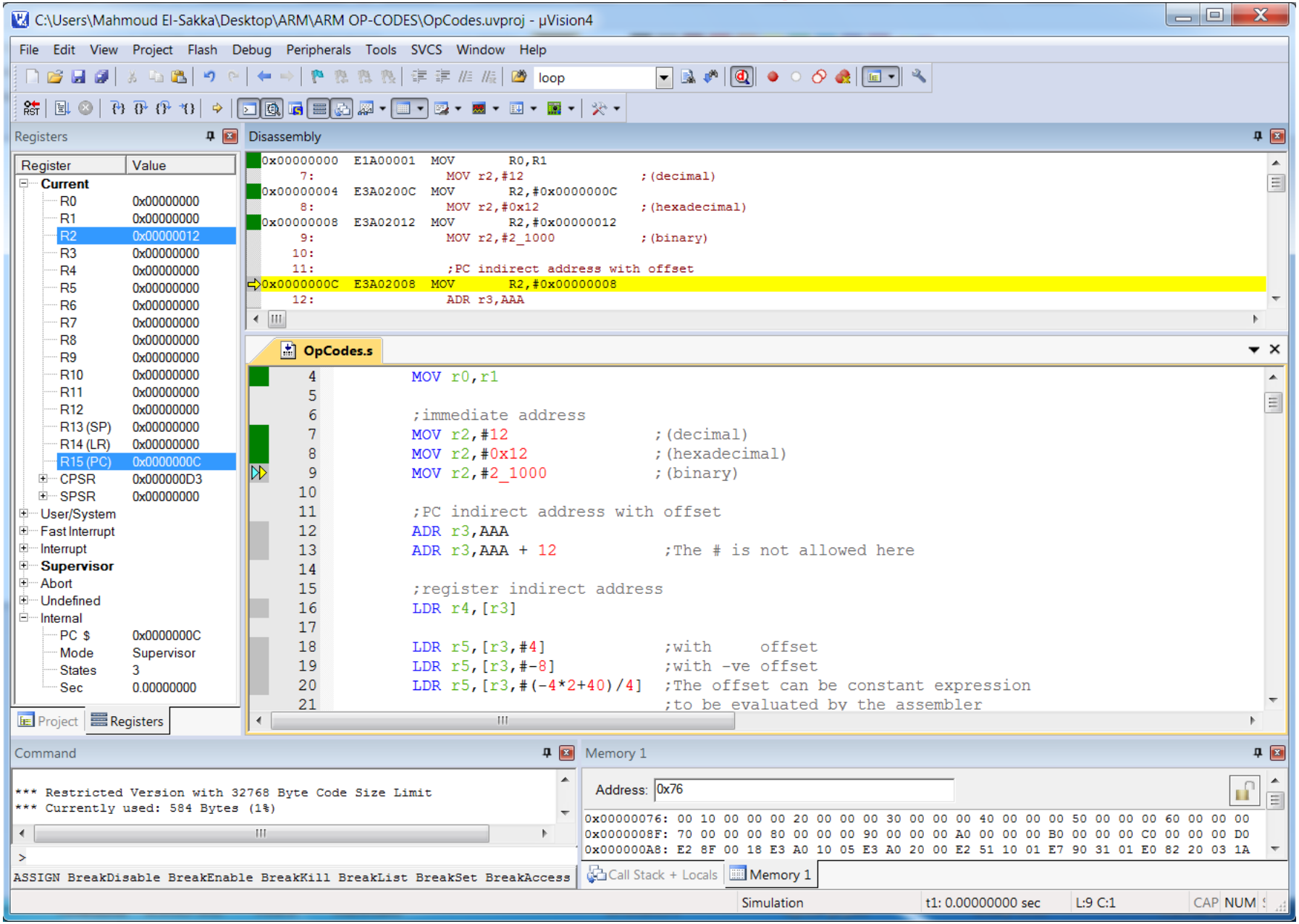
ARM Addressing Modes

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 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 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

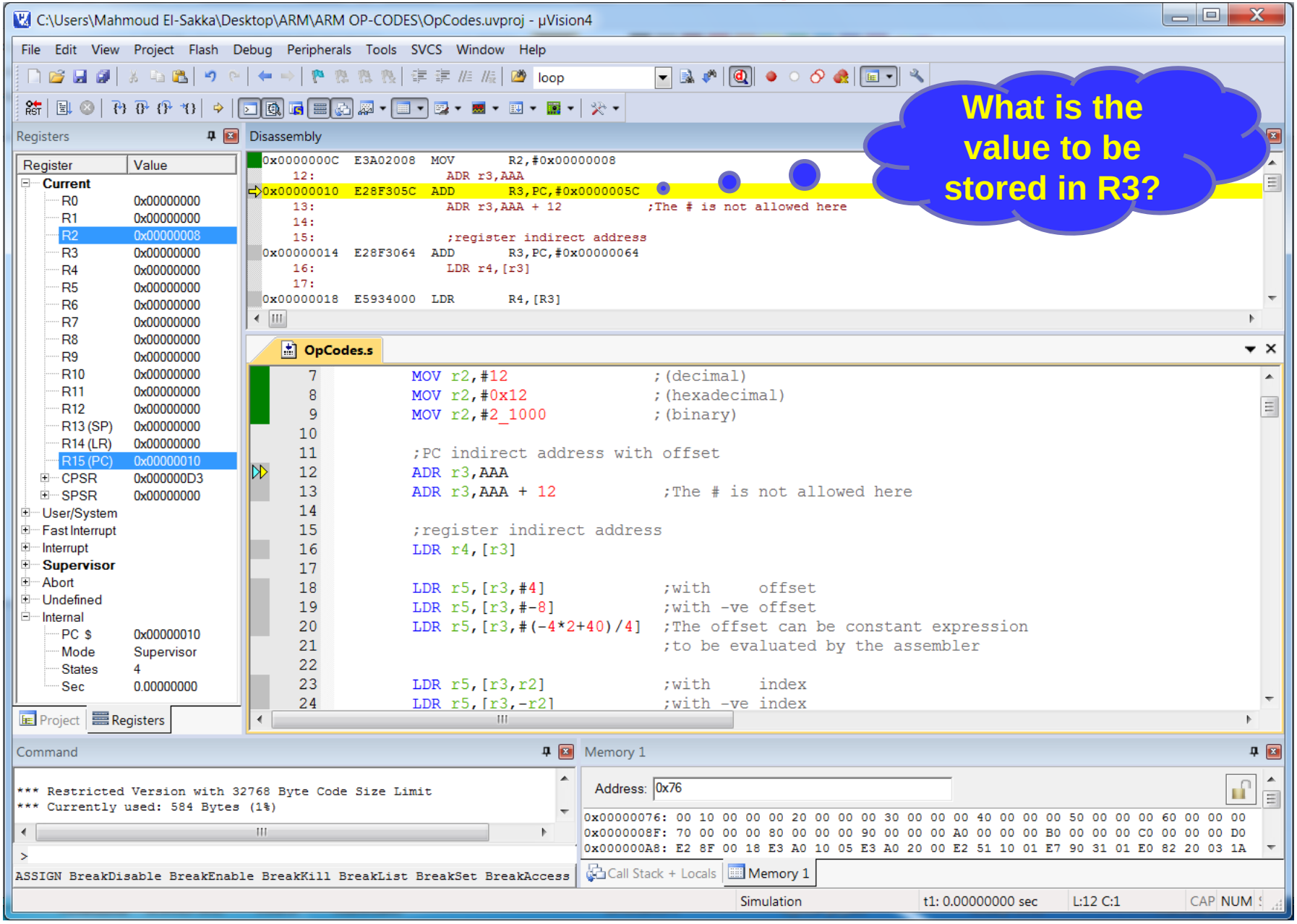
ARM Addressing Modes



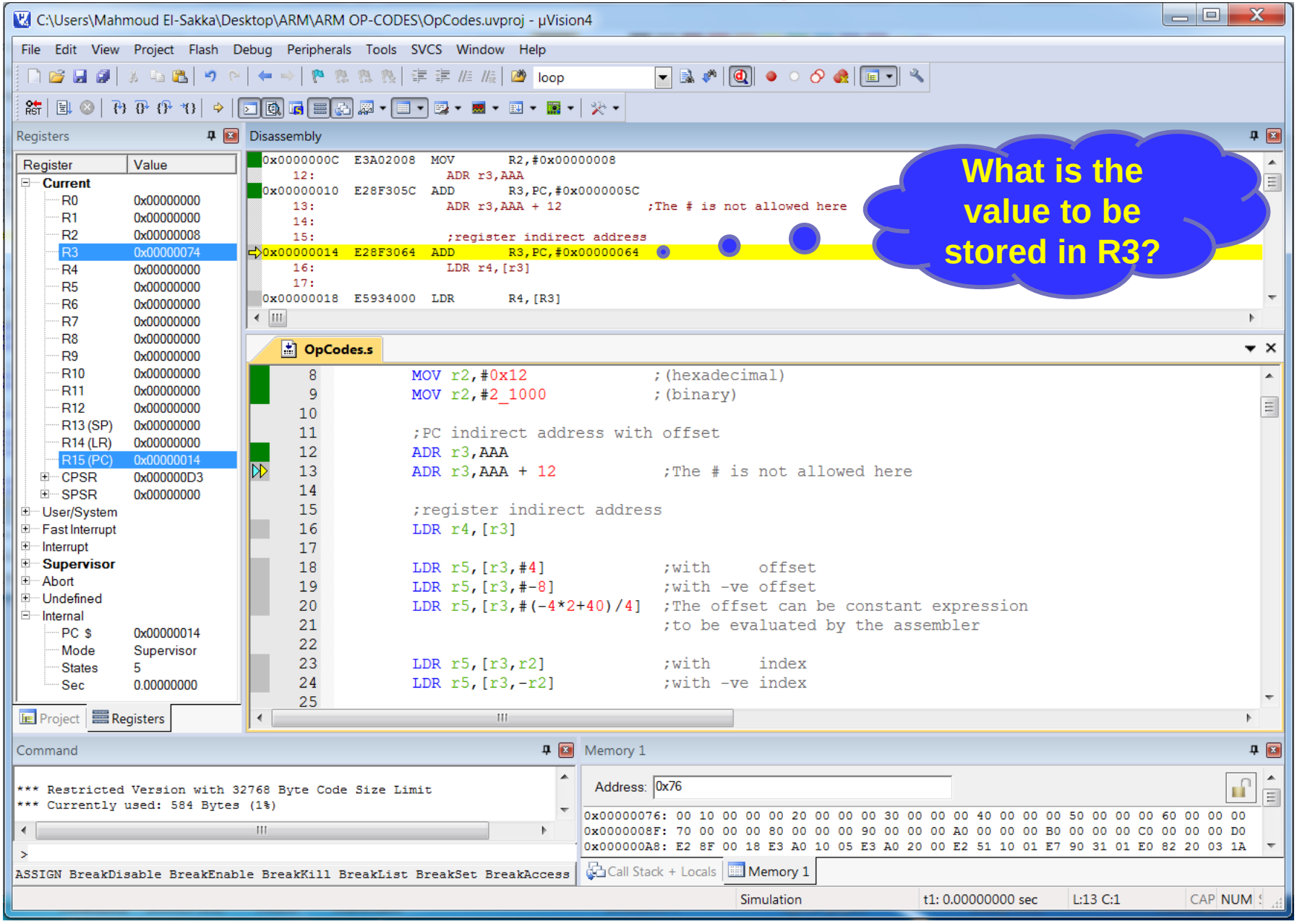
ARM Addressing Modes



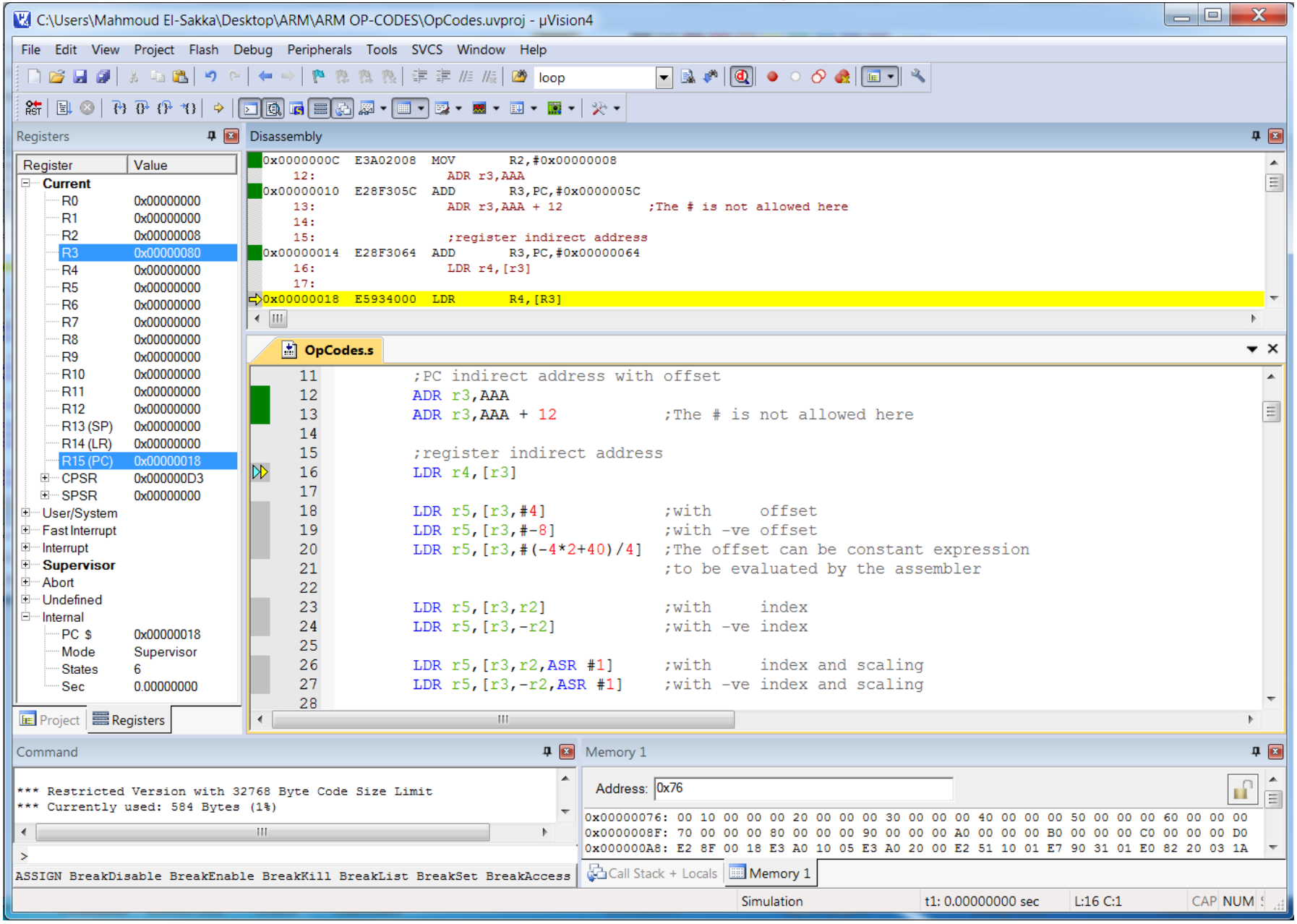
ARM Addressing Modes



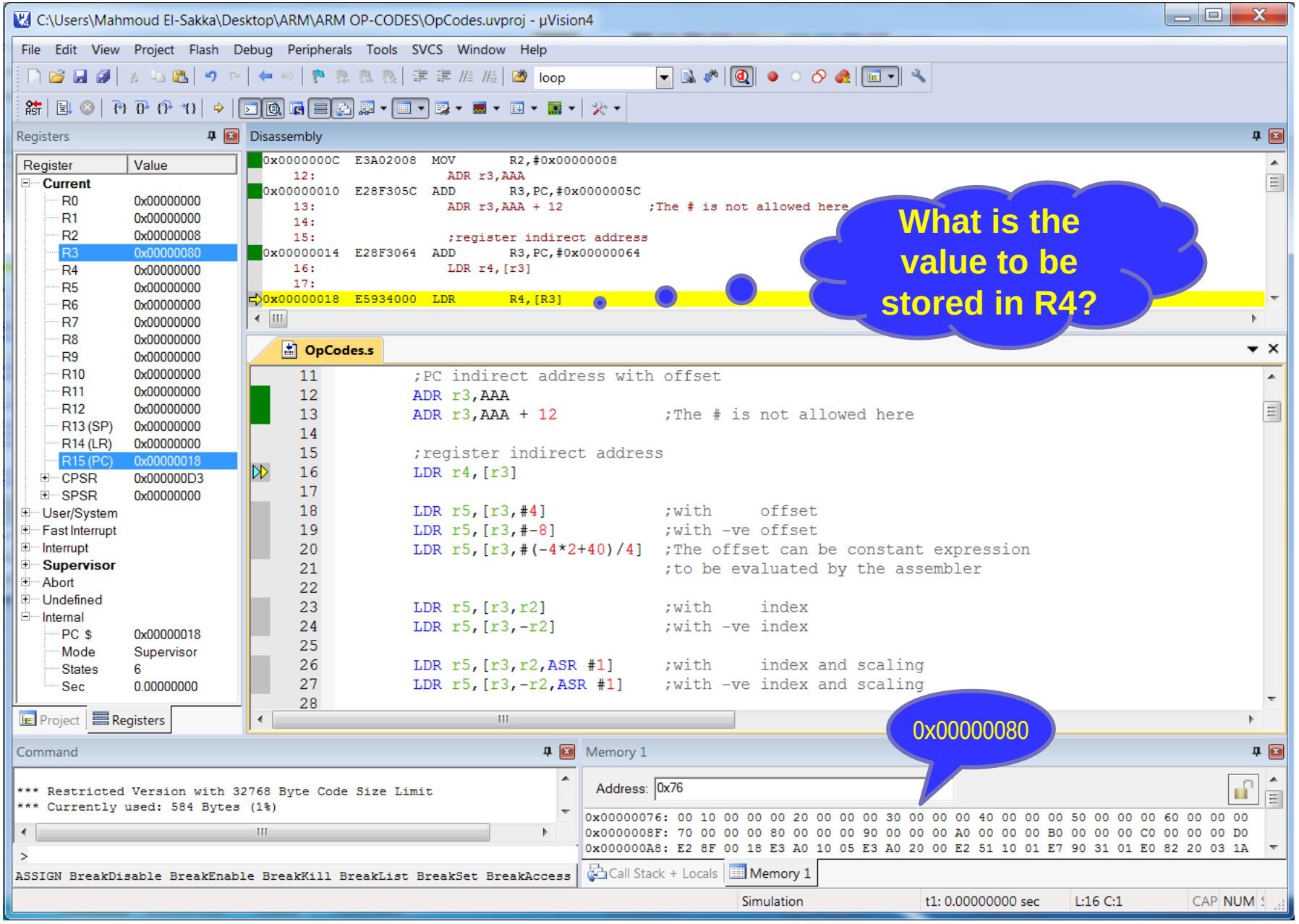
ARM Addressing Modes



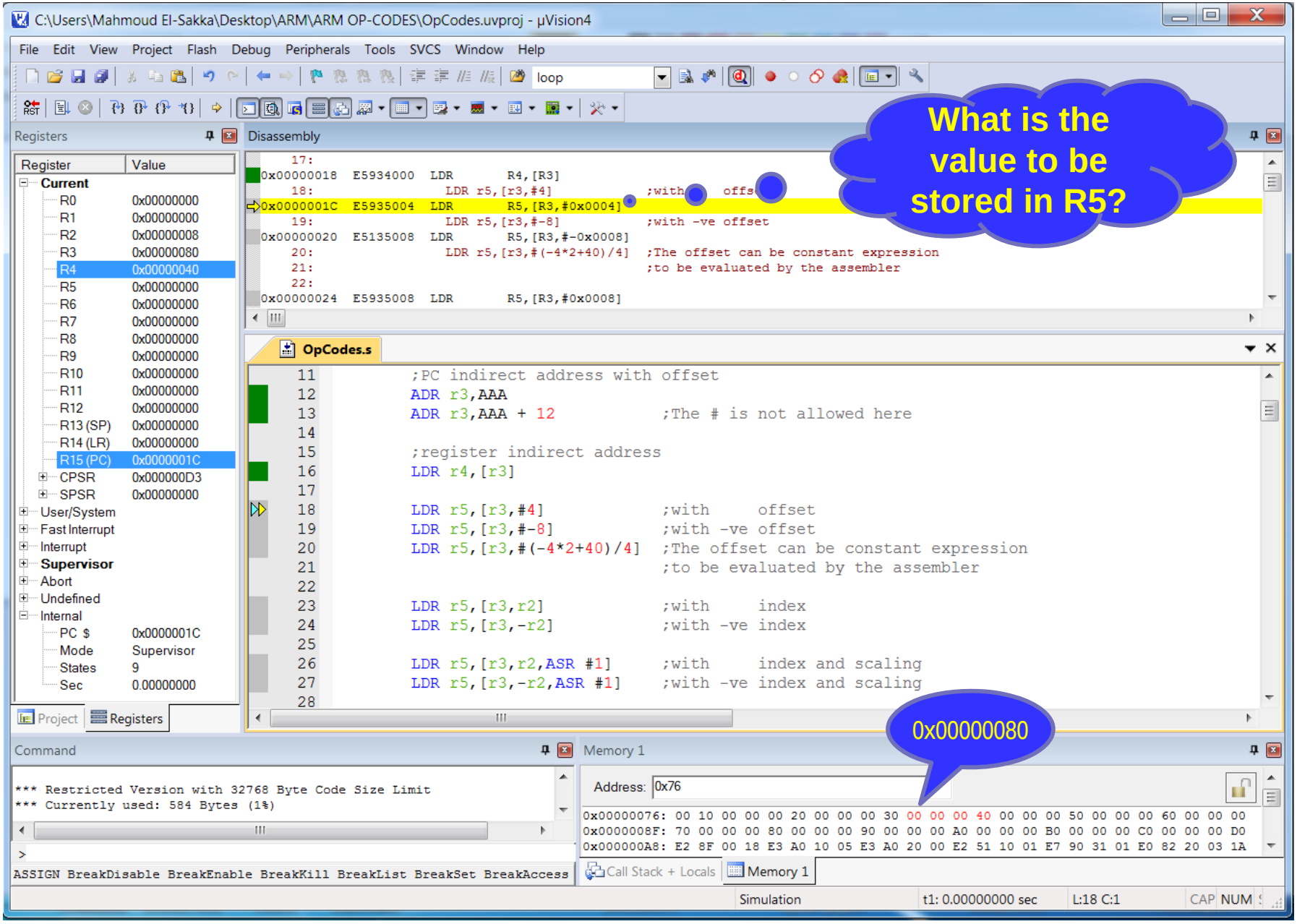
ARM Addressing Modes



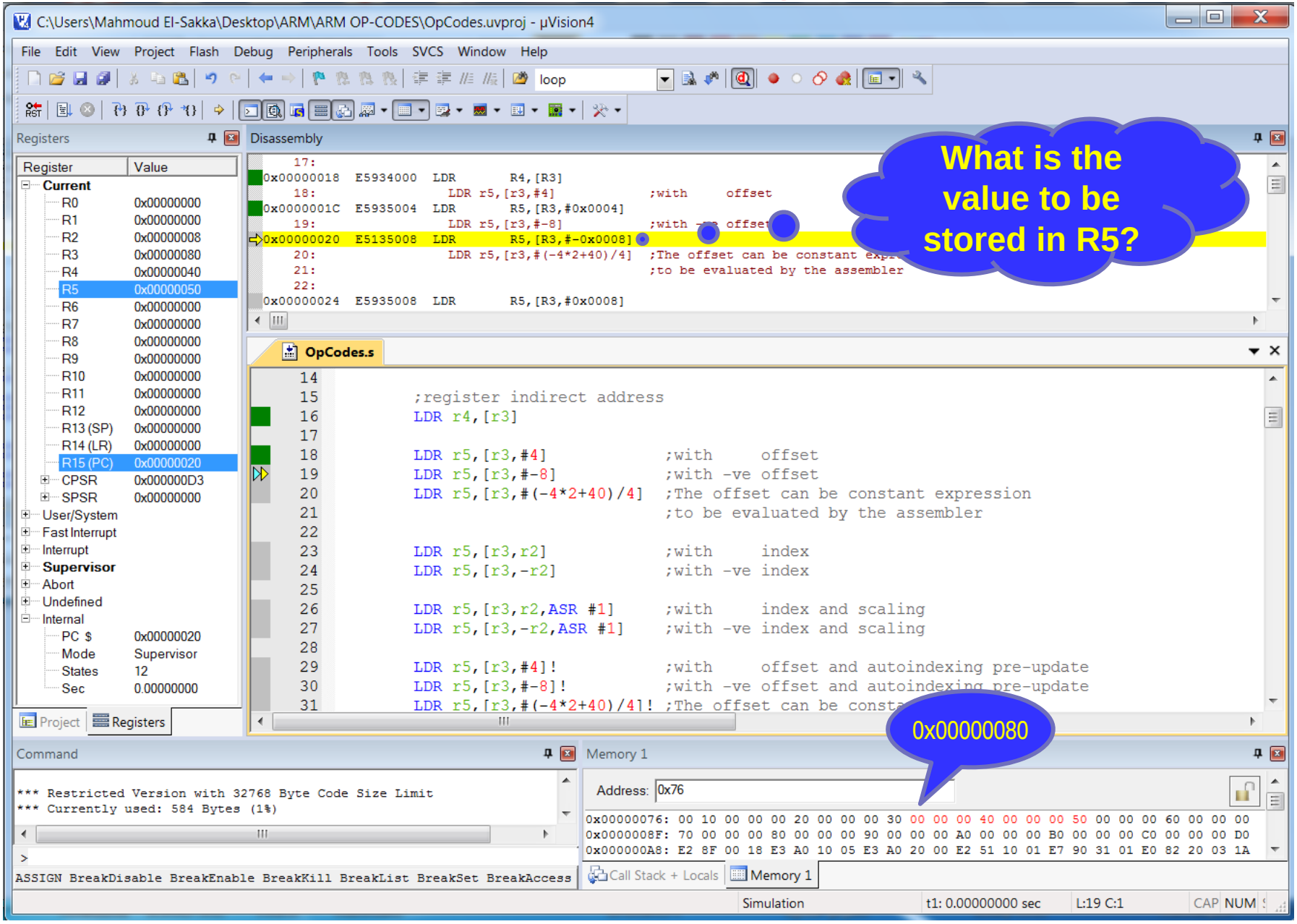
ARM Addressing Modes



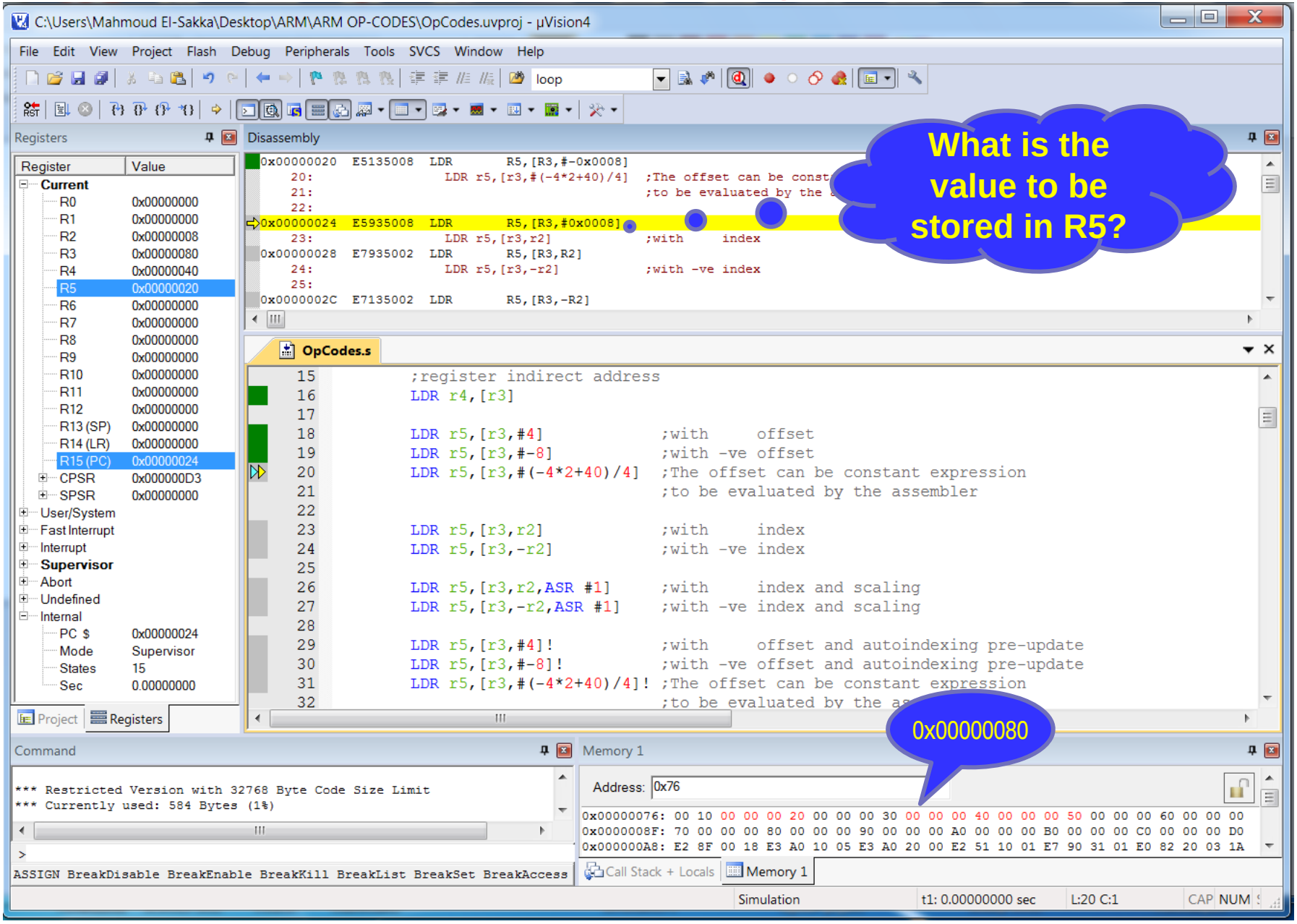
ARM Addressing Modes



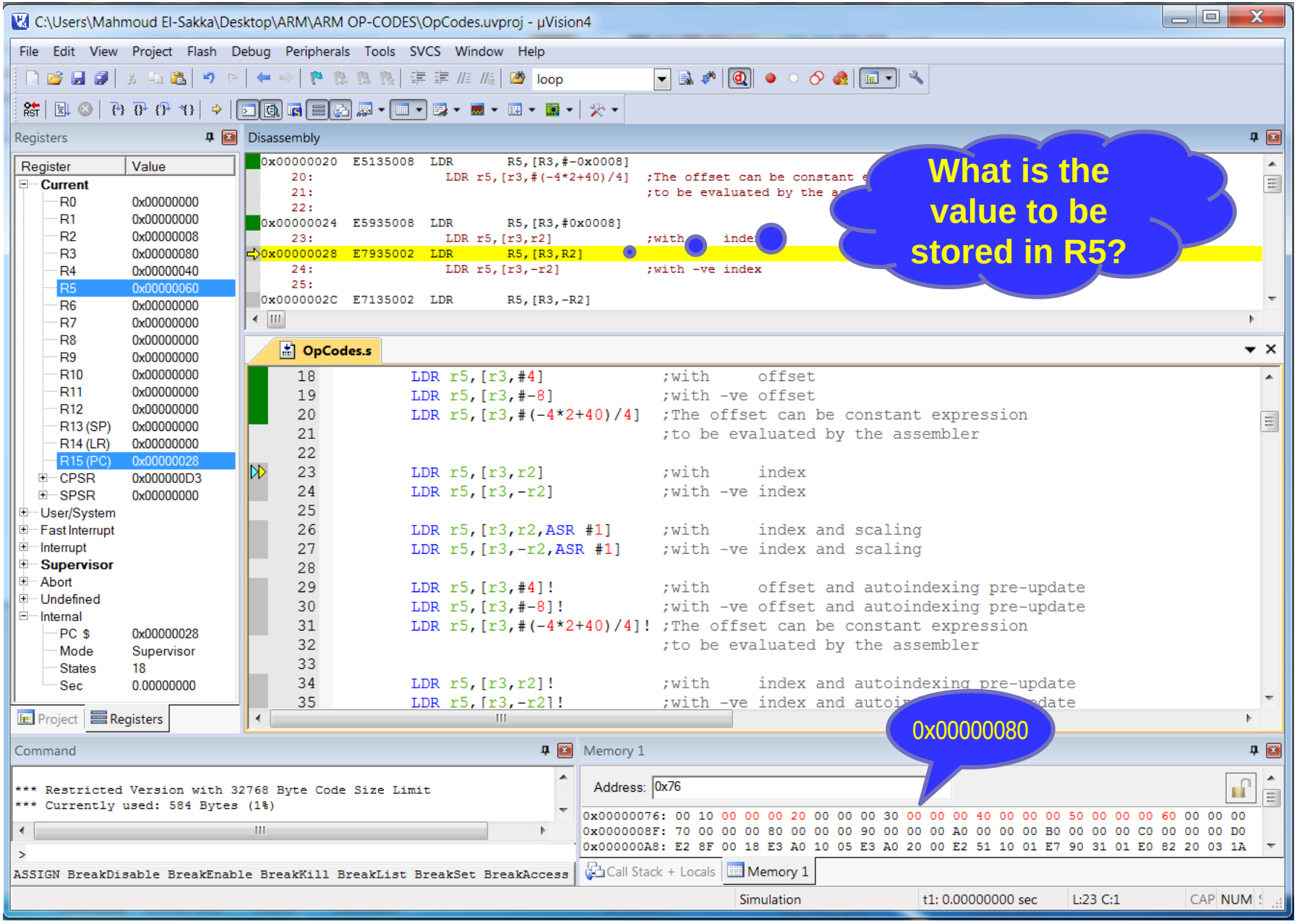
ARM Addressing Modes



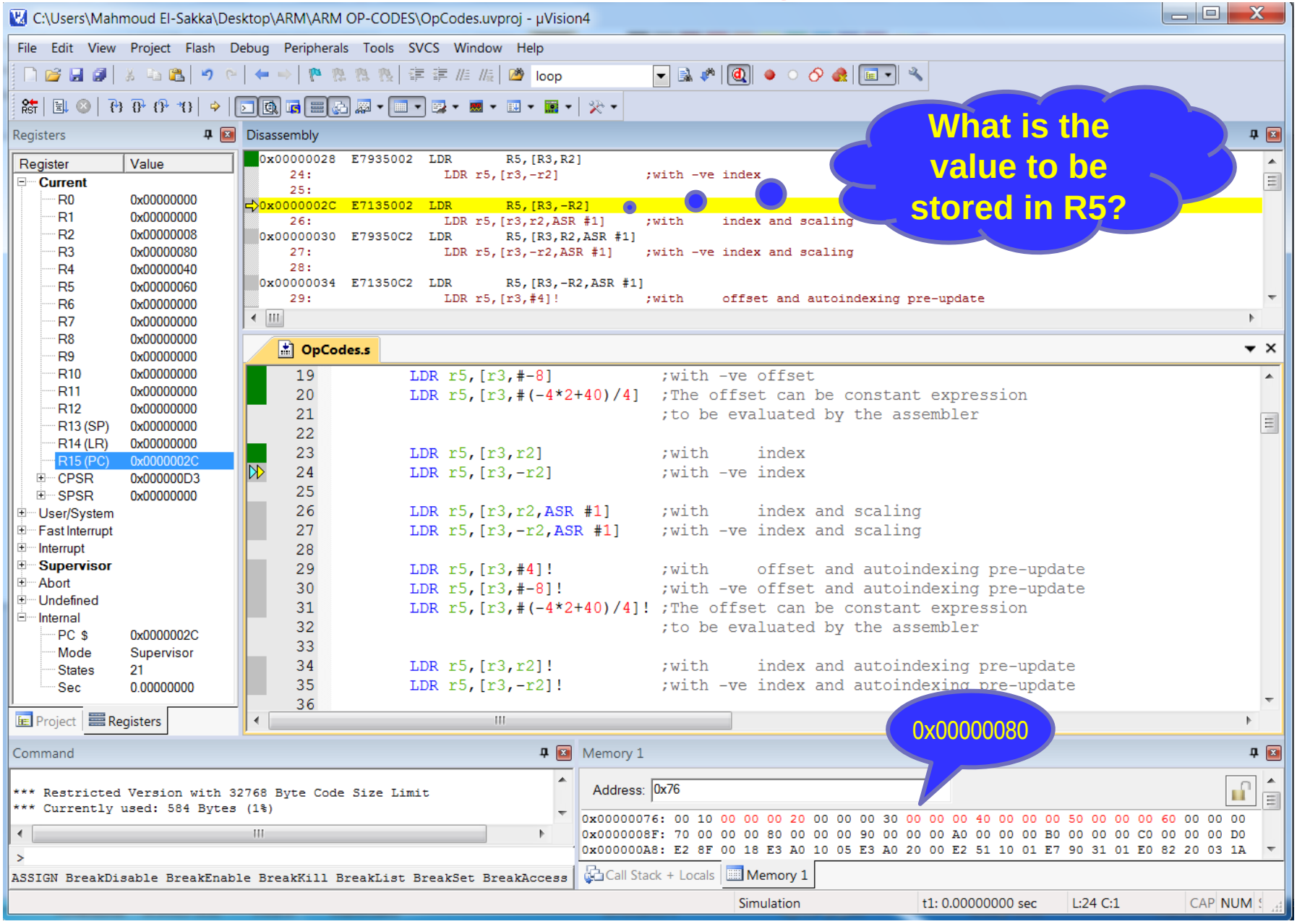
ARM Addressing Modes



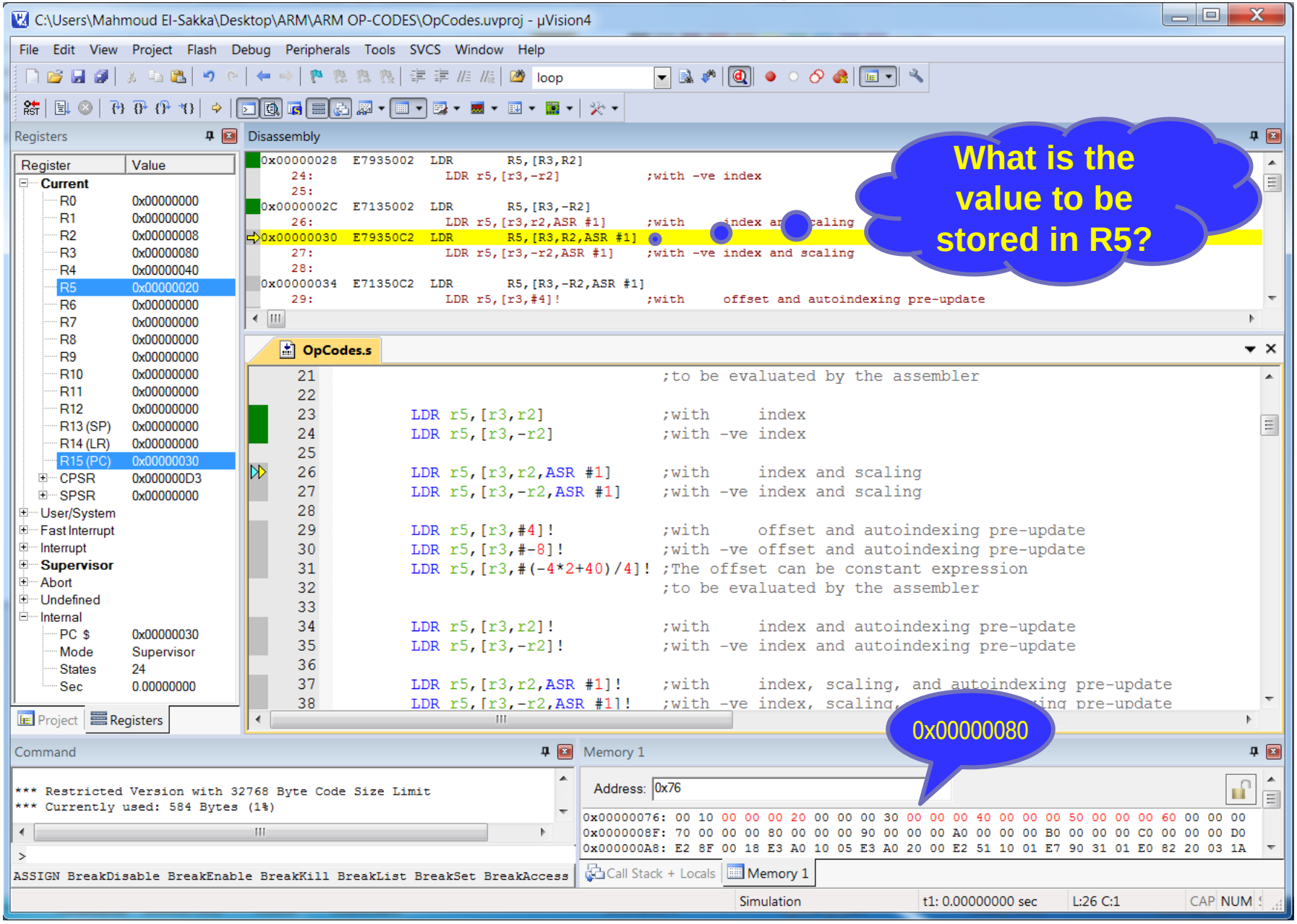
ARM Addressing Modes



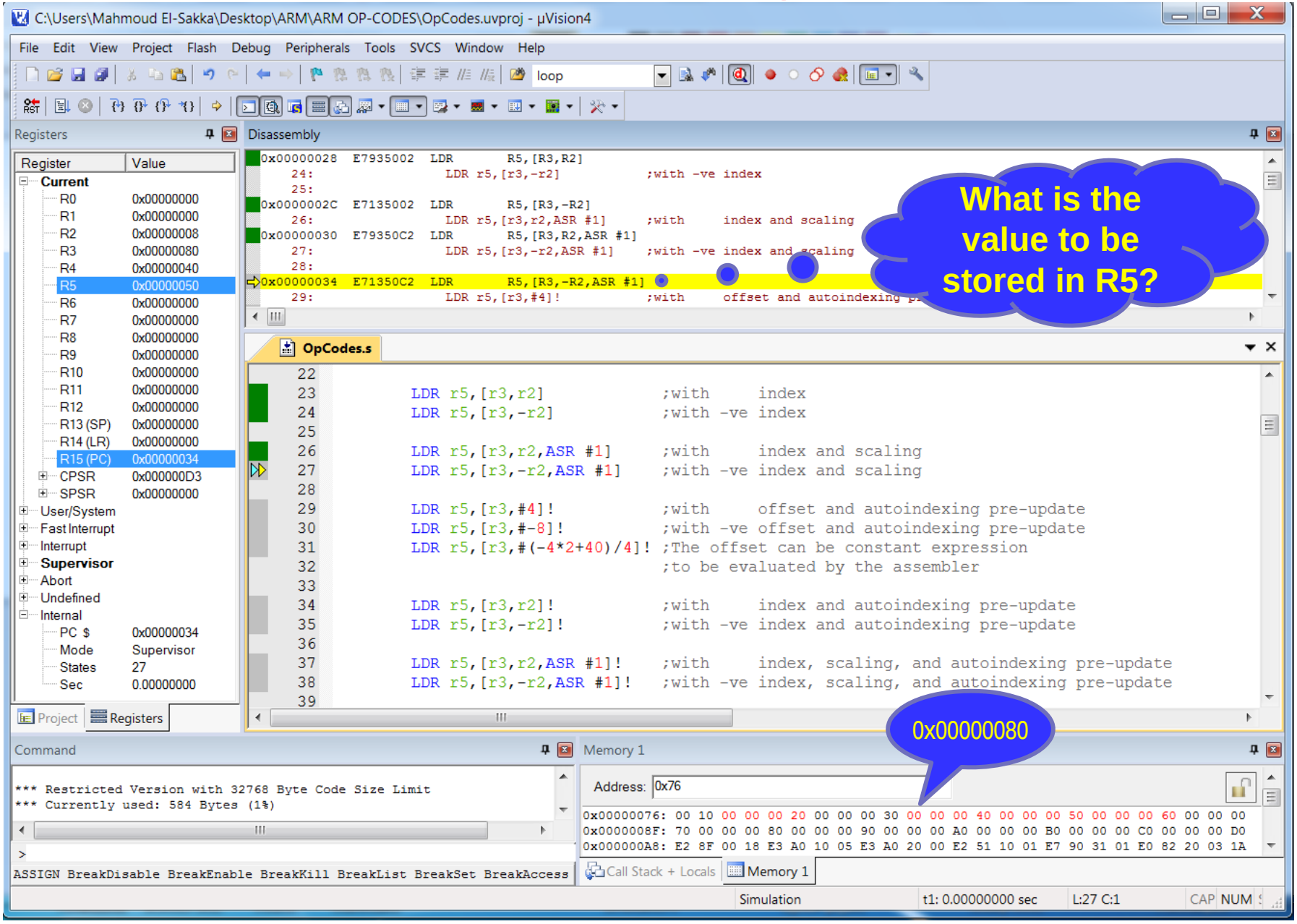
ARM Addressing Modes



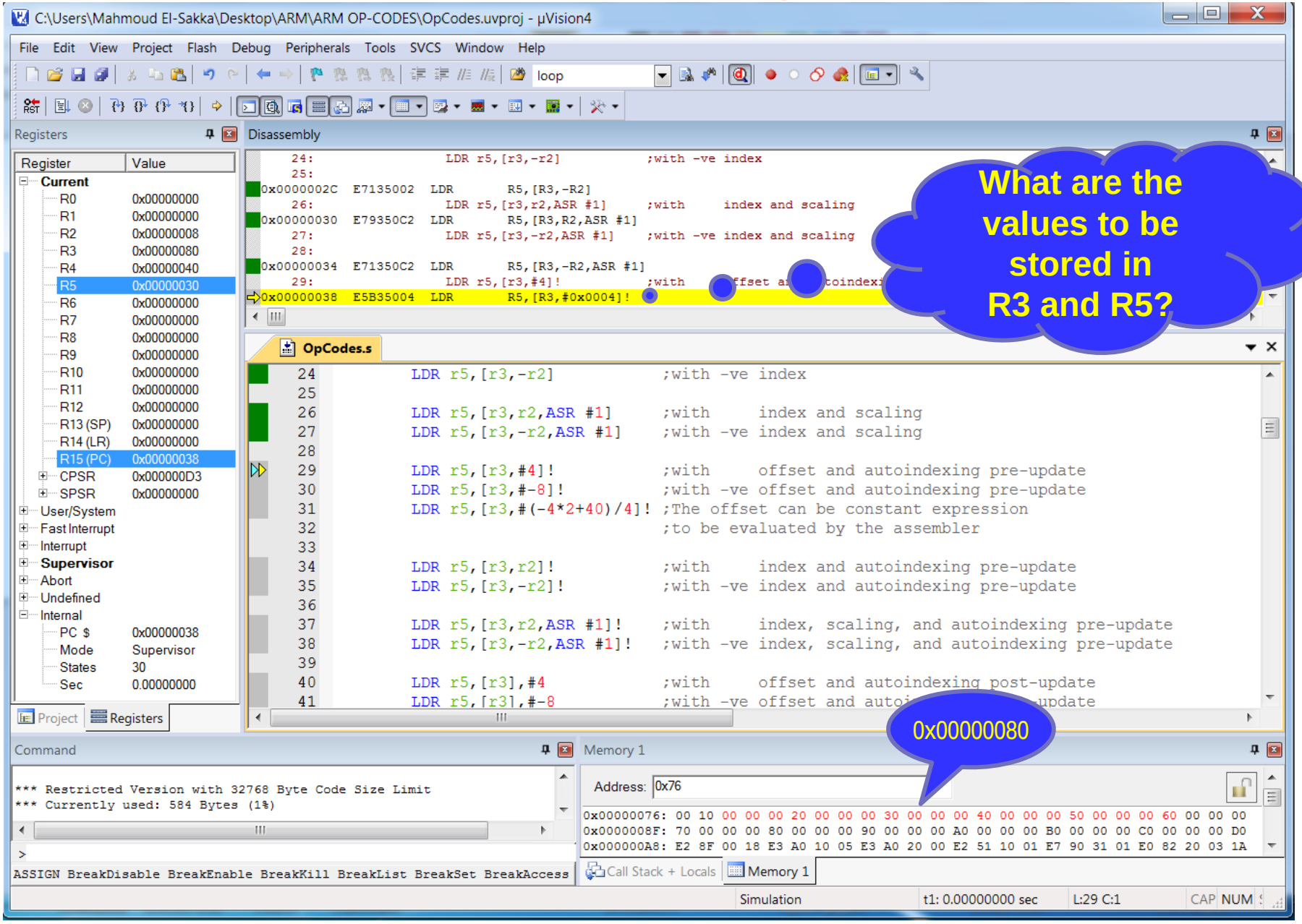
ARM Addressing Modes



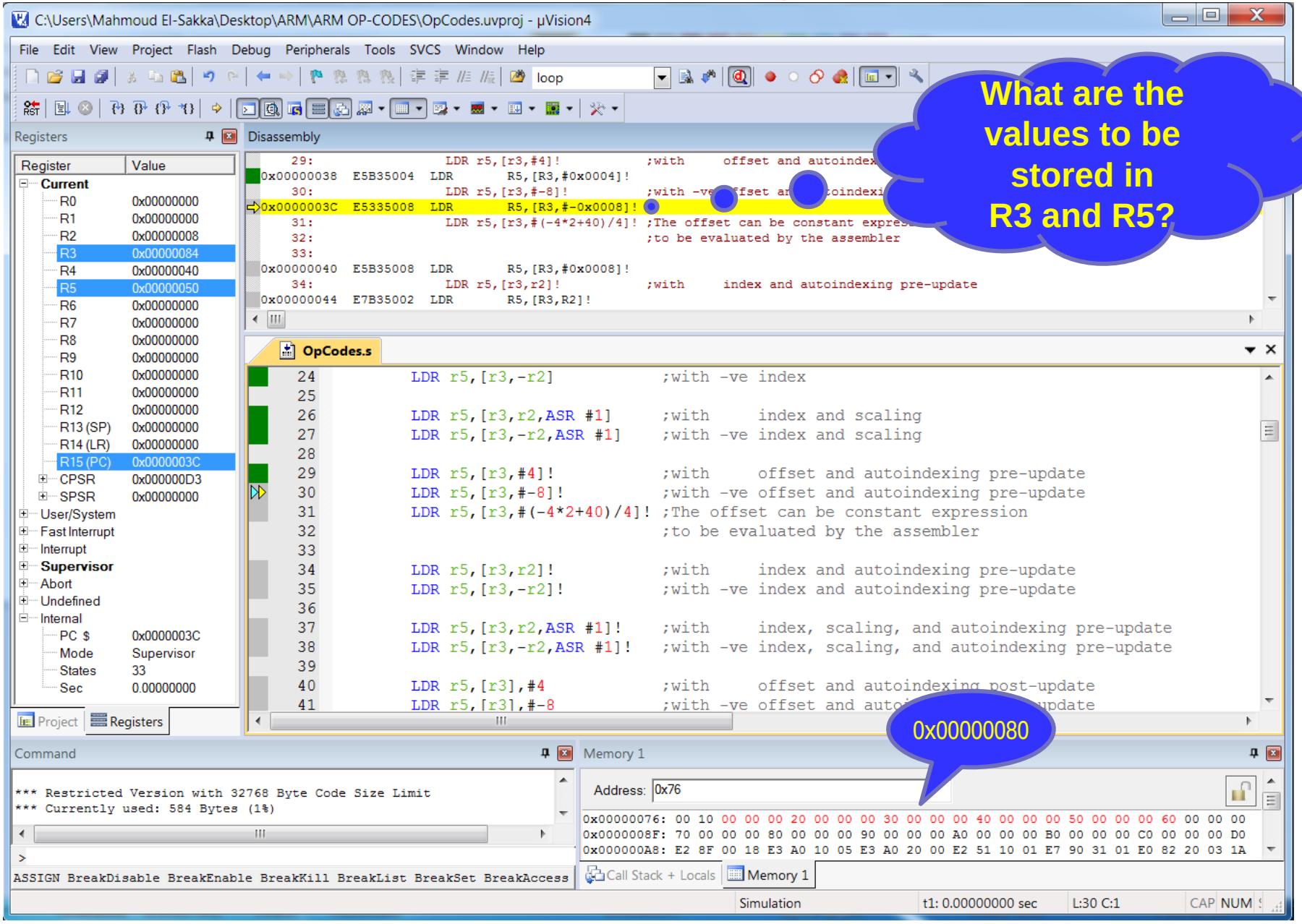
ARM Addressing Modes



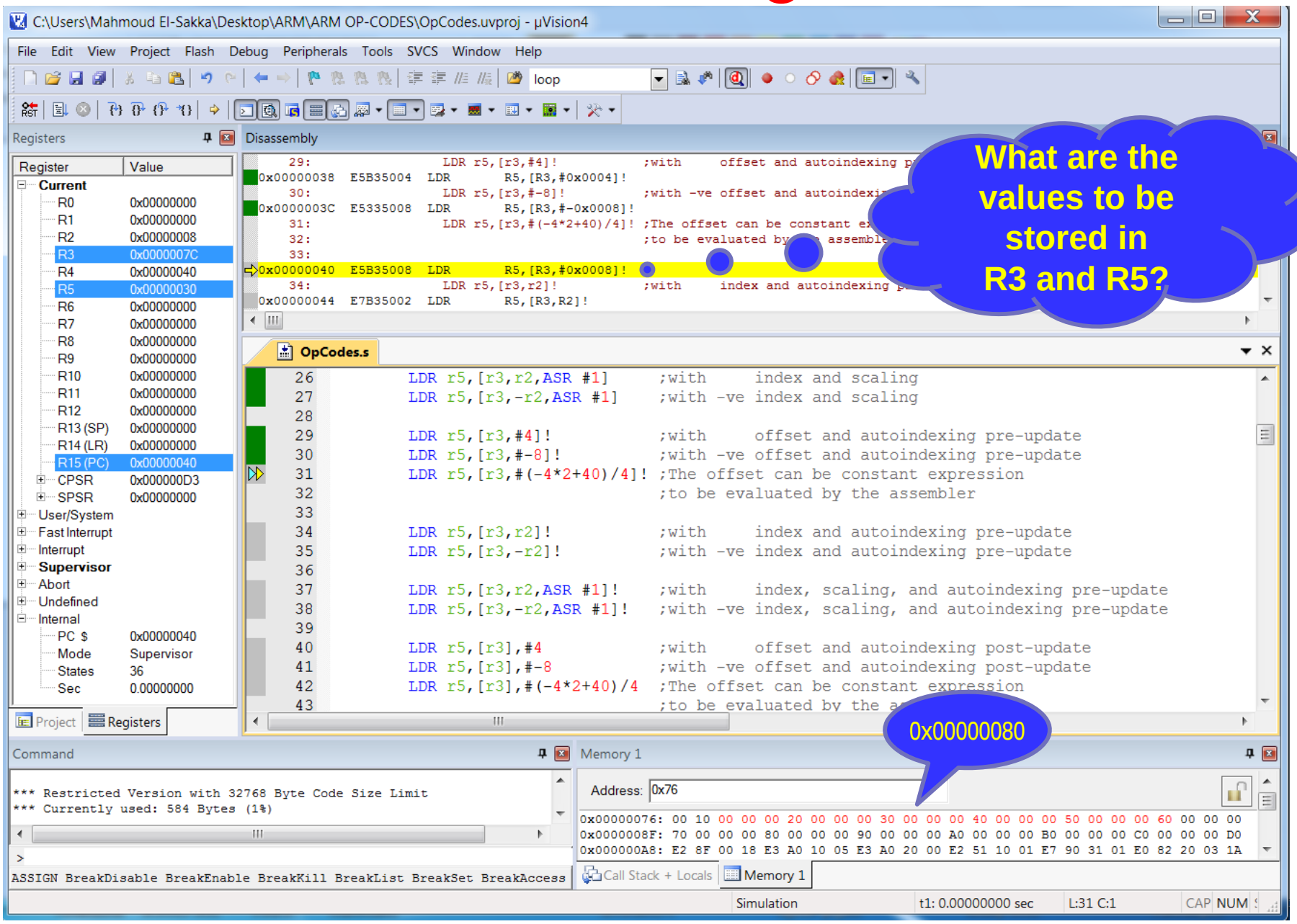
ARM Addressing Modes



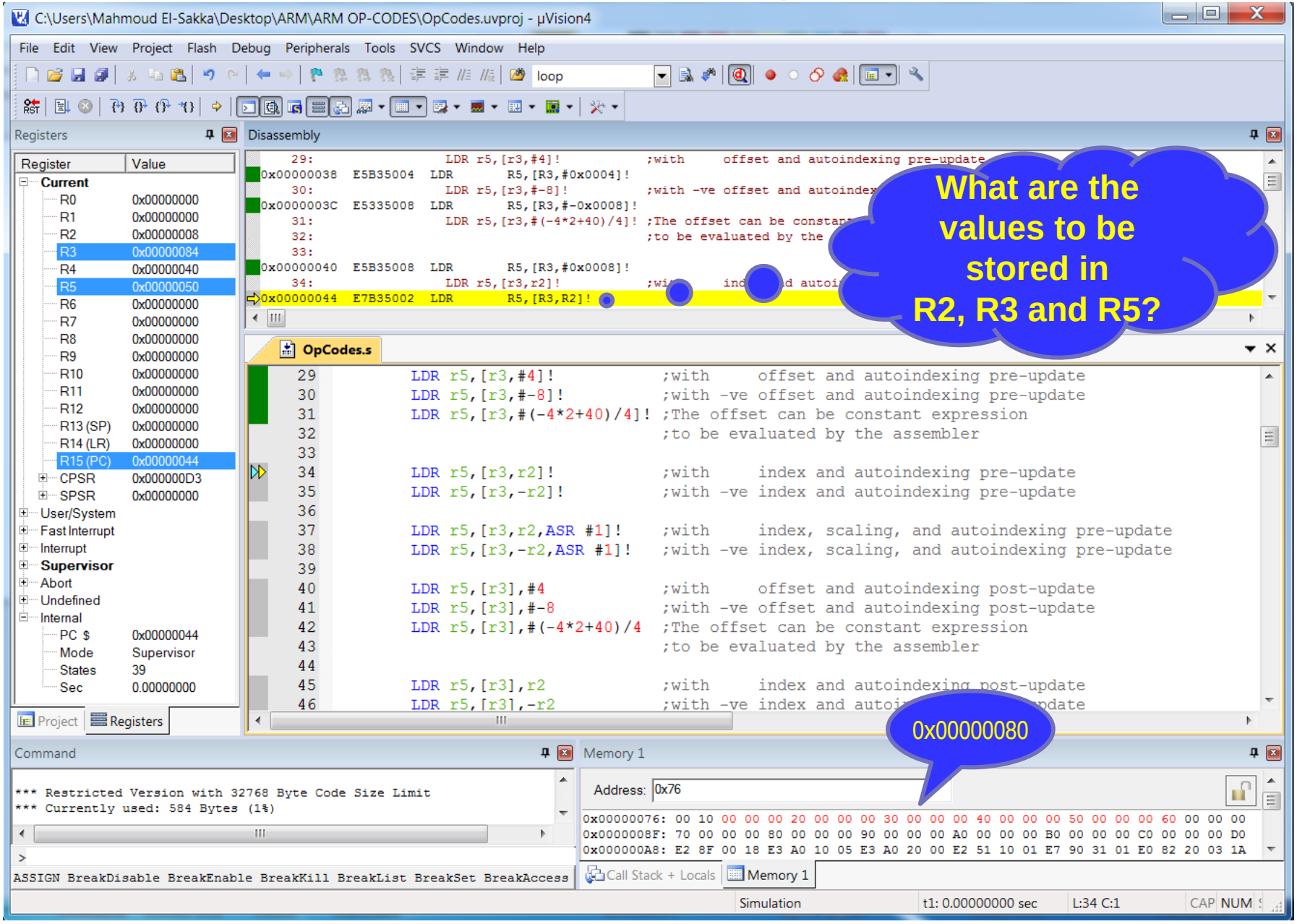
ARM Addressing Modes



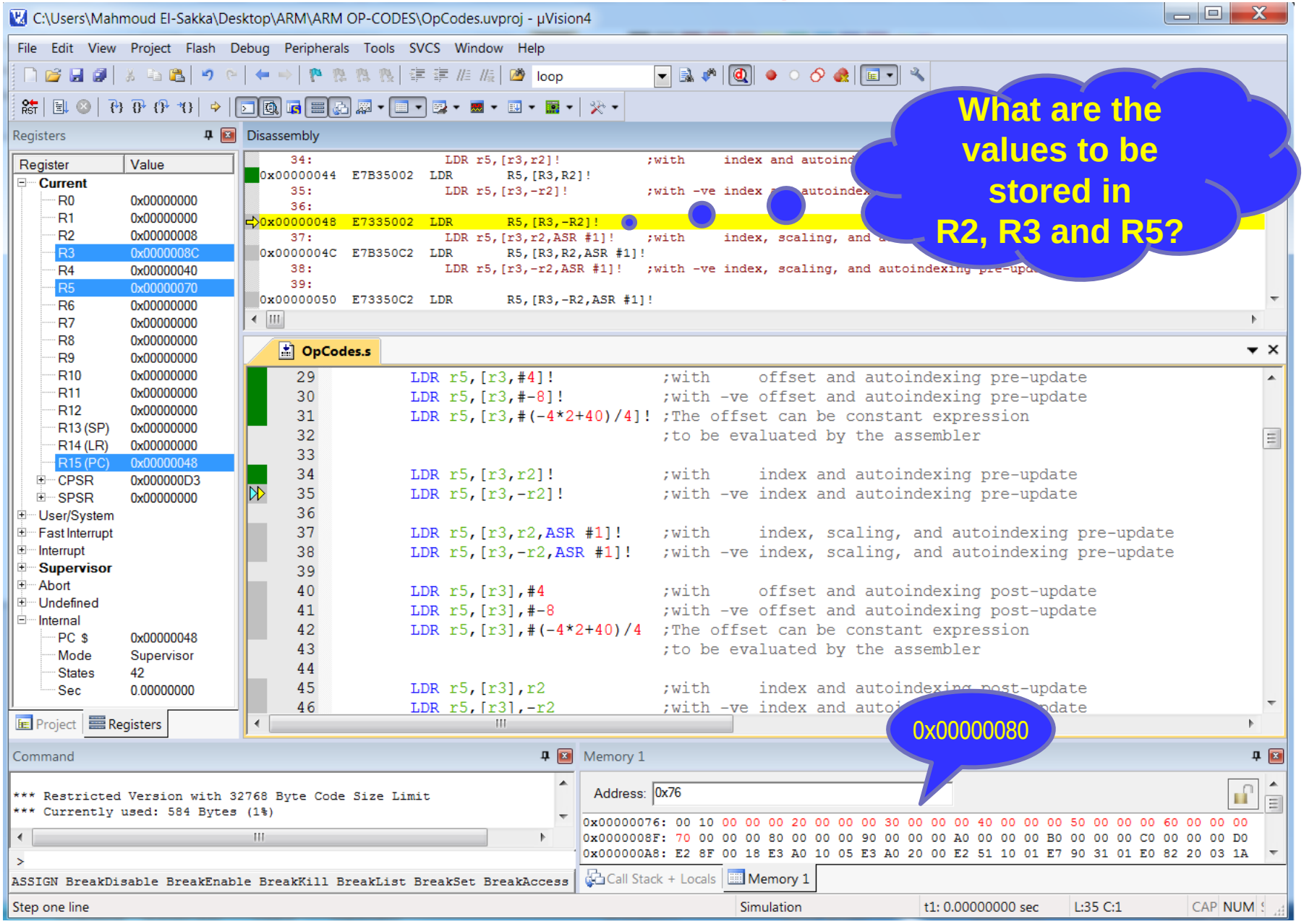
ARM Addressing Modes



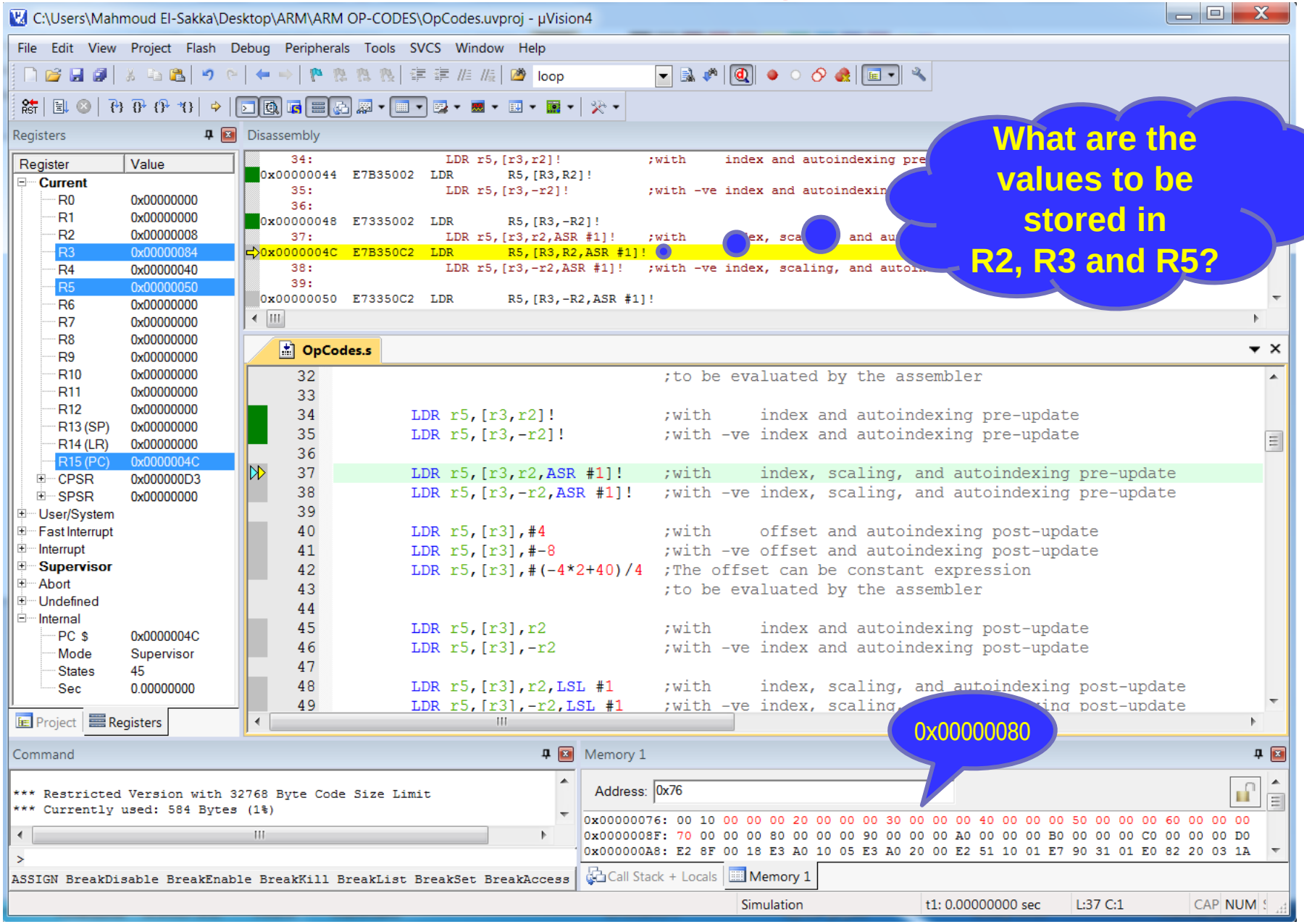
ARM Addressing Modes



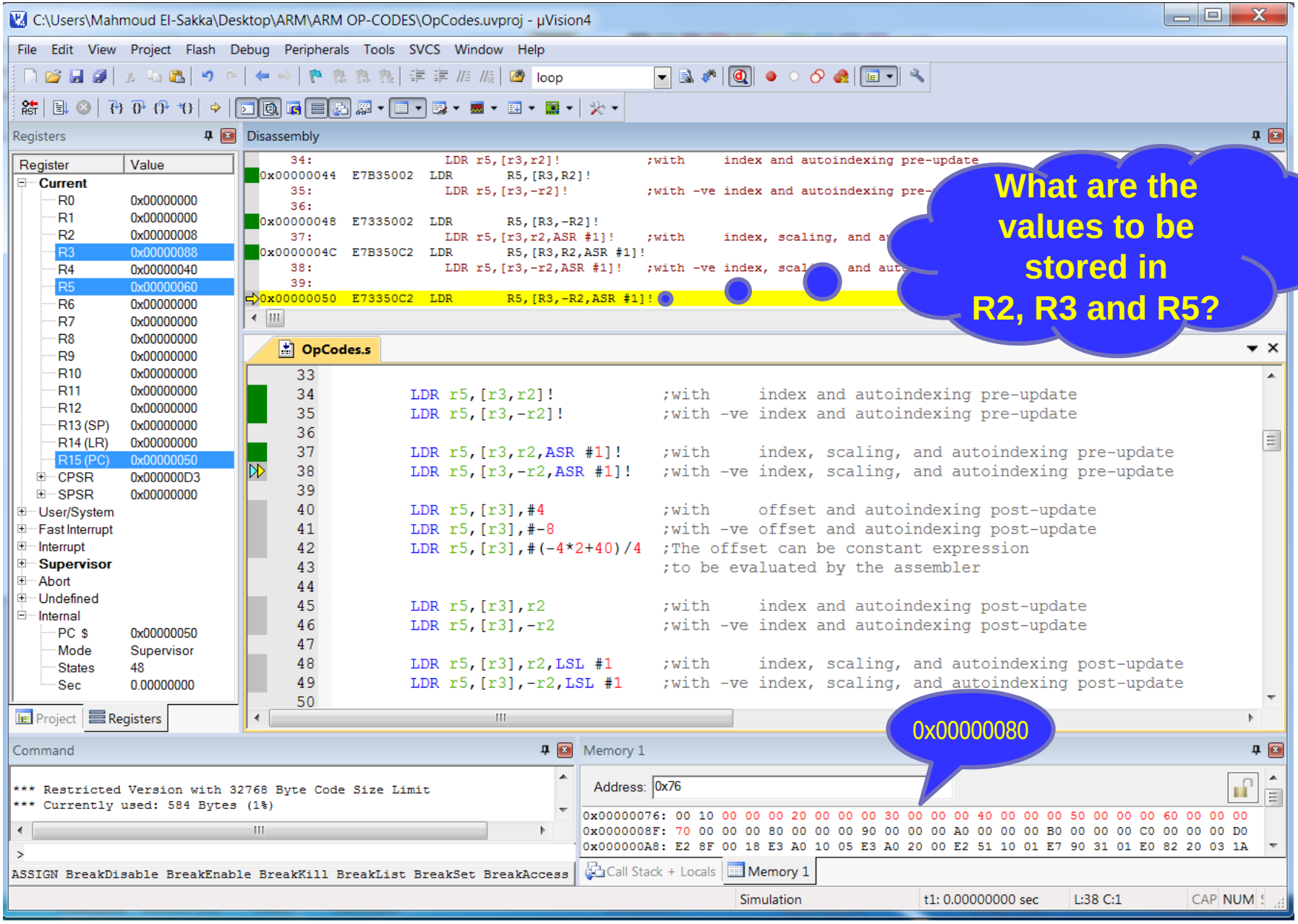
ARM Addressing Modes



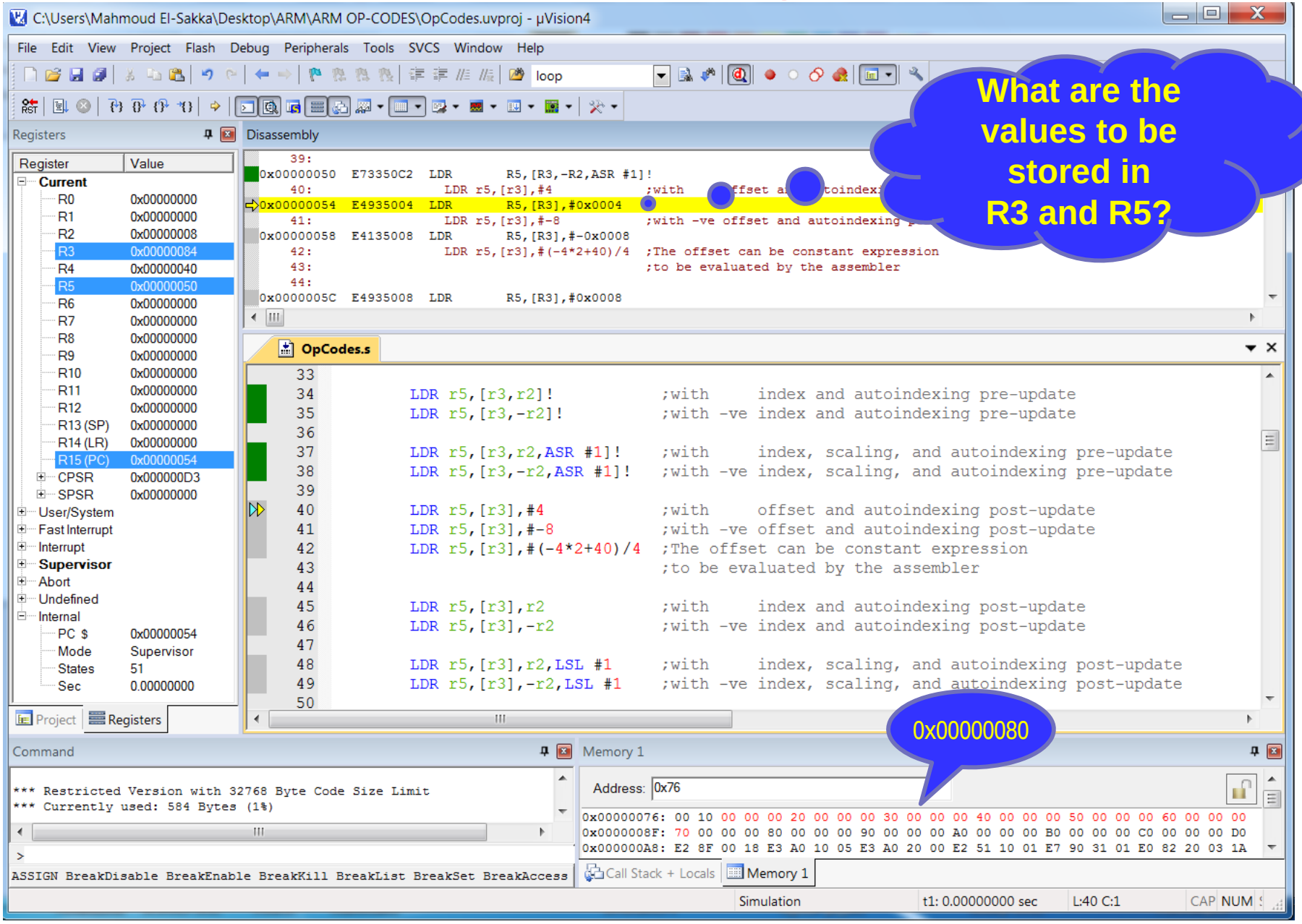
ARM Addressing Modes



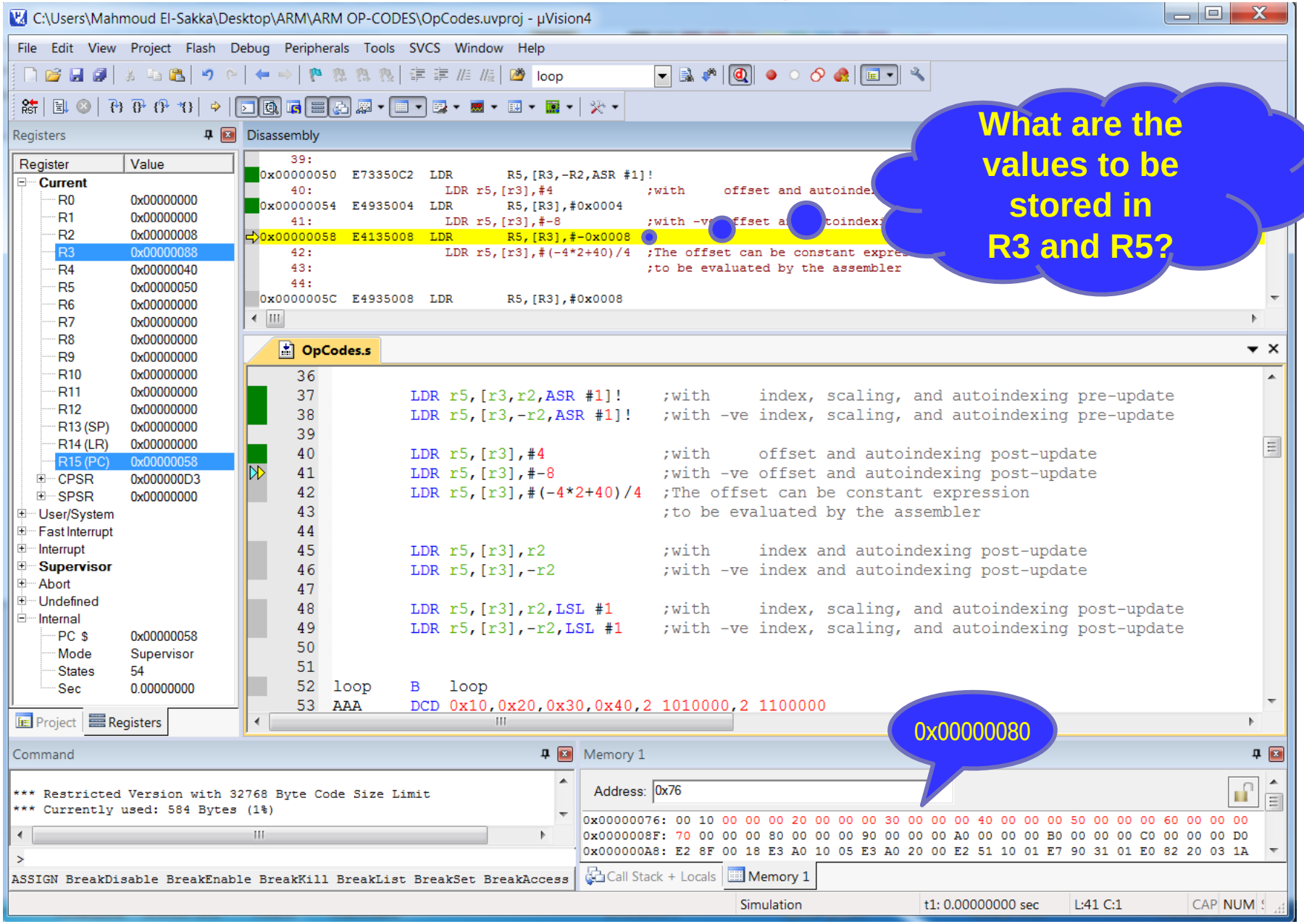
ARM Addressing Modes



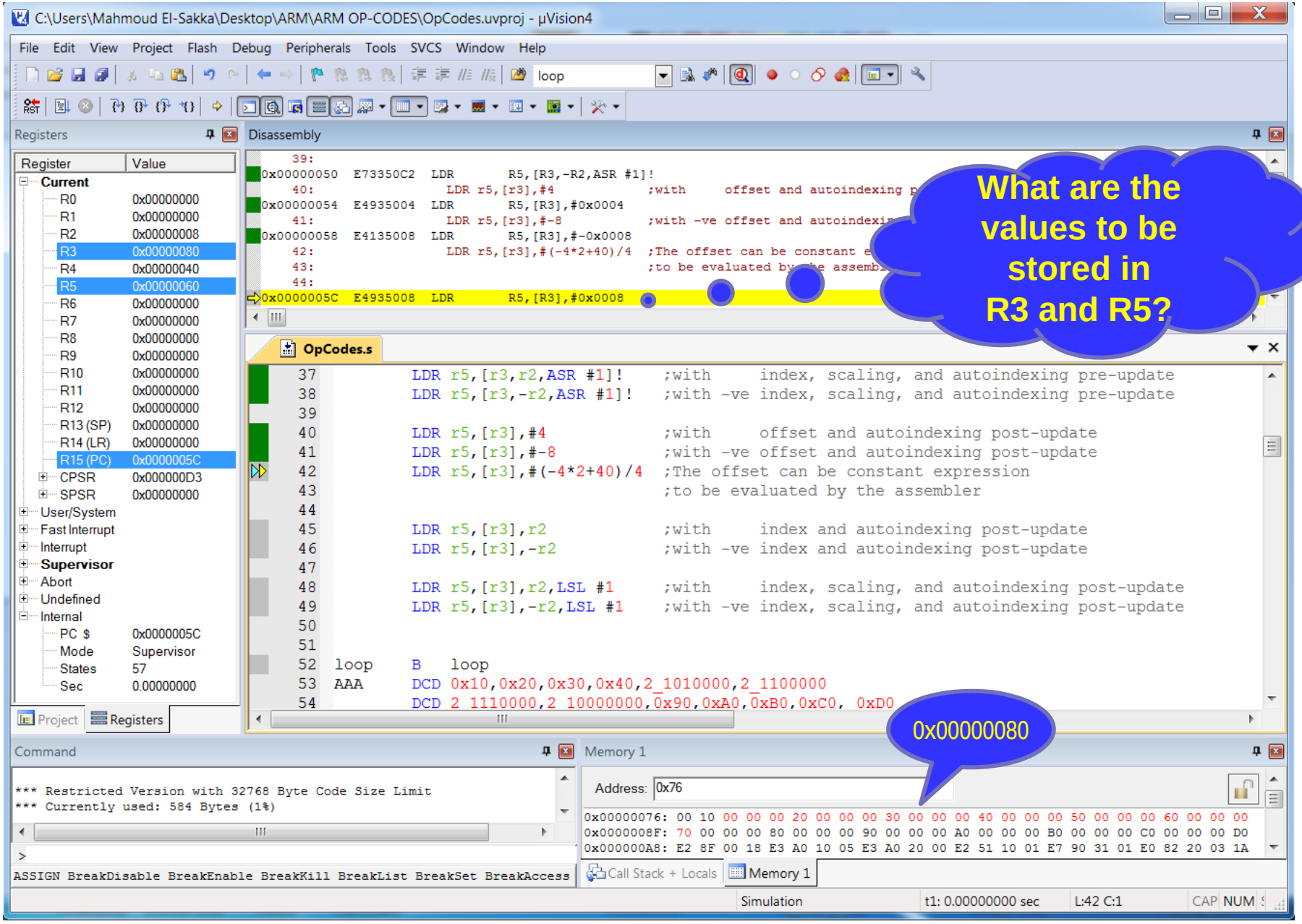
ARM Addressing Modes



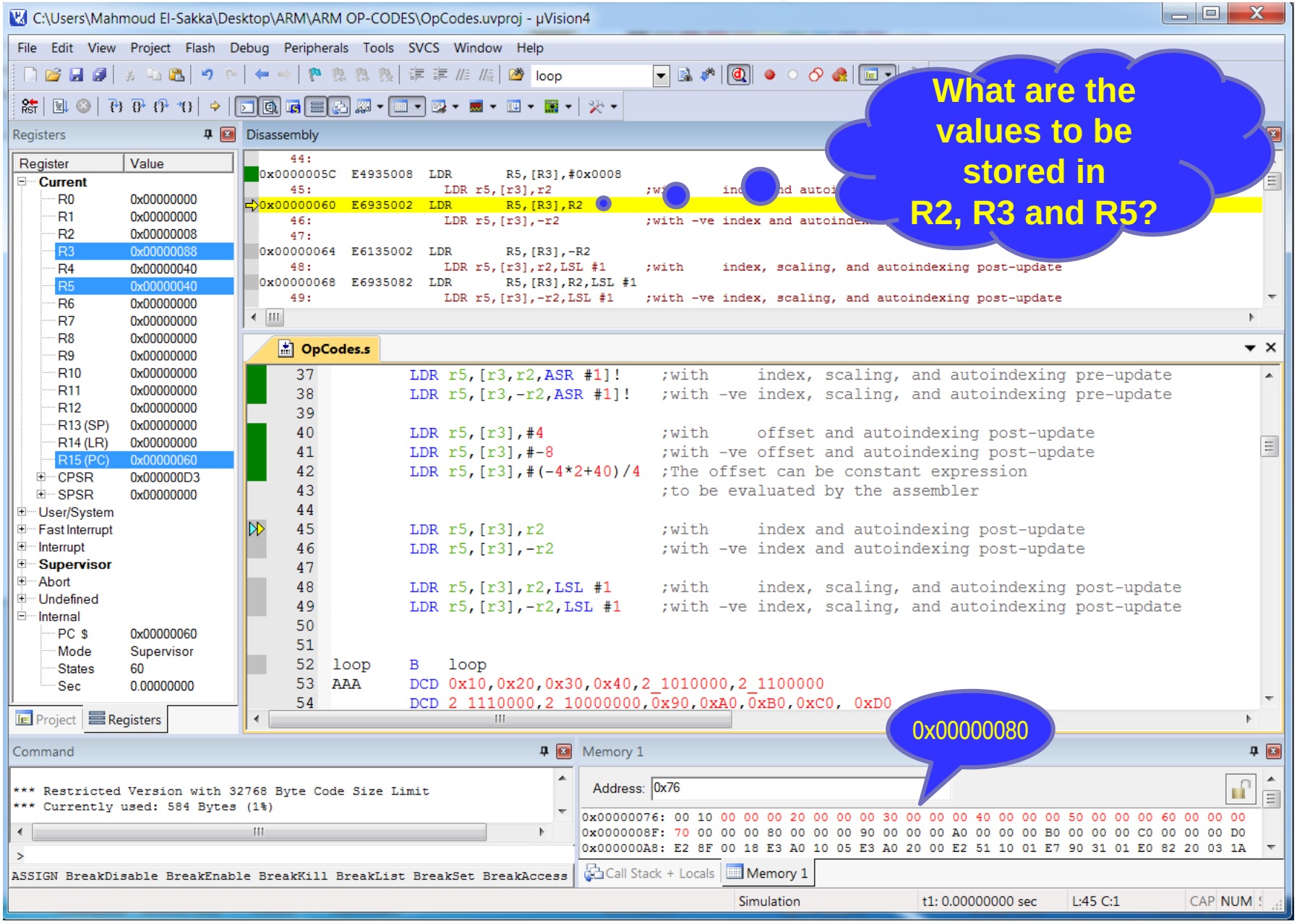
ARM Addressing Modes



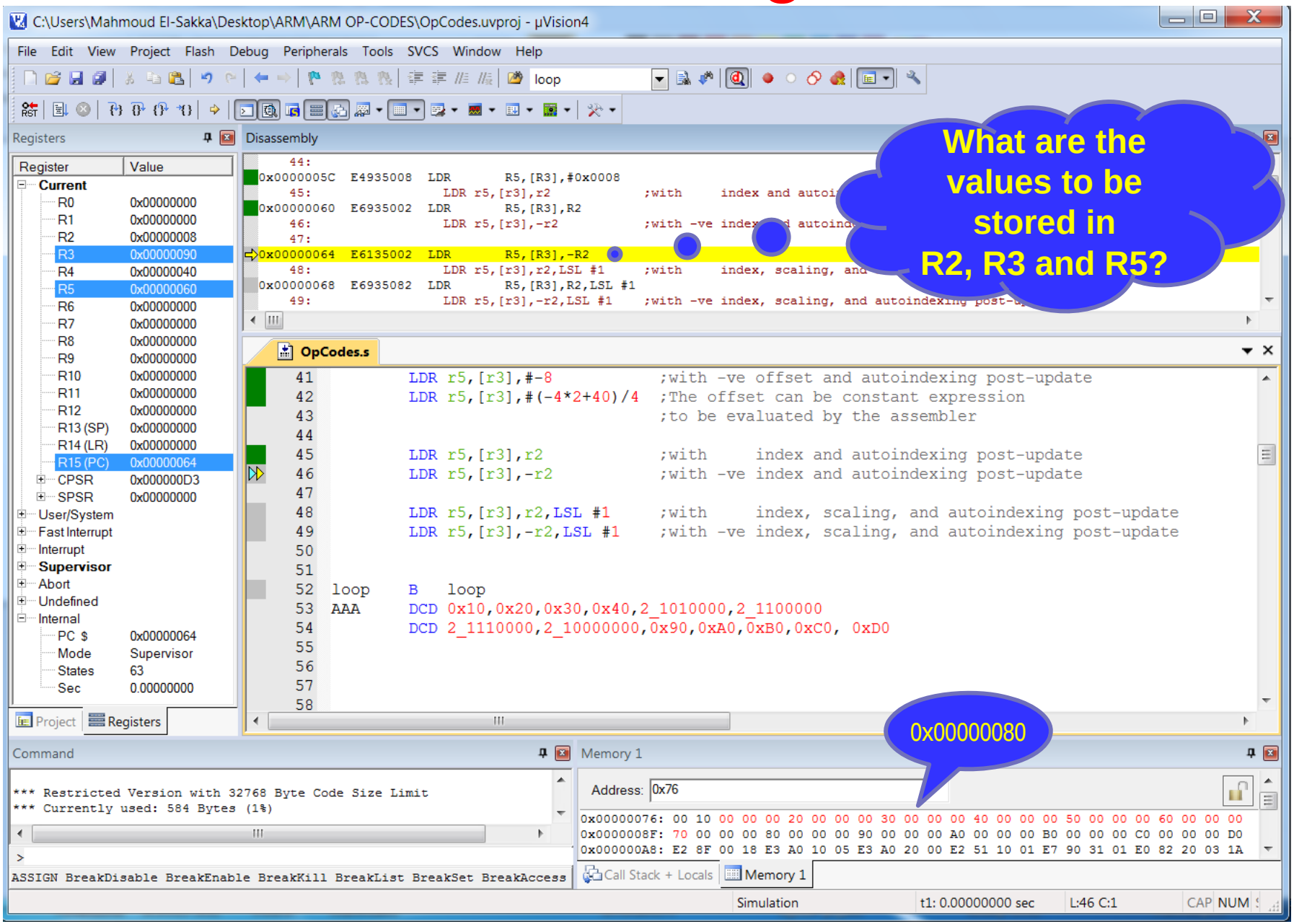
ARM Addressing Modes



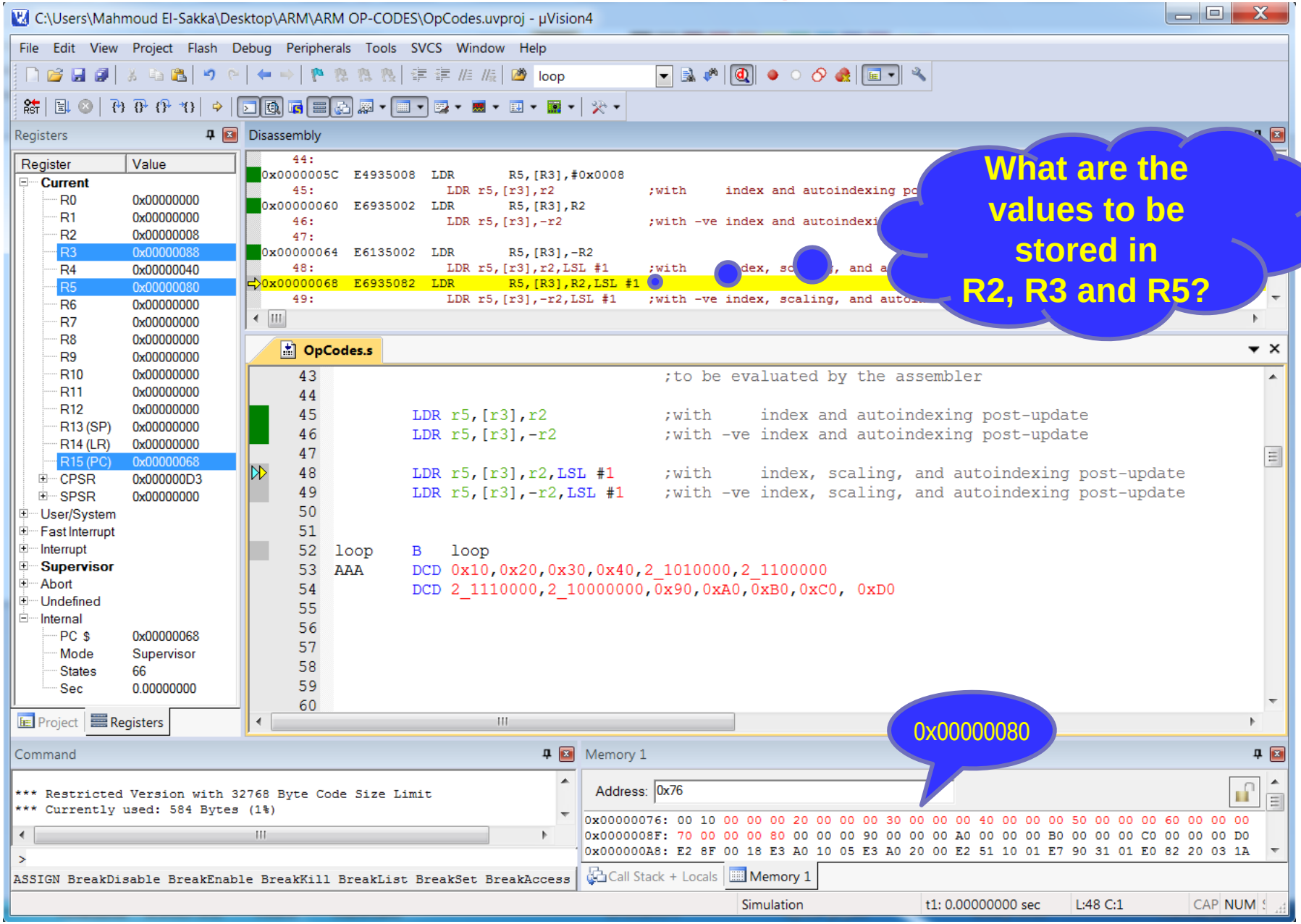
ARM Addressing Modes



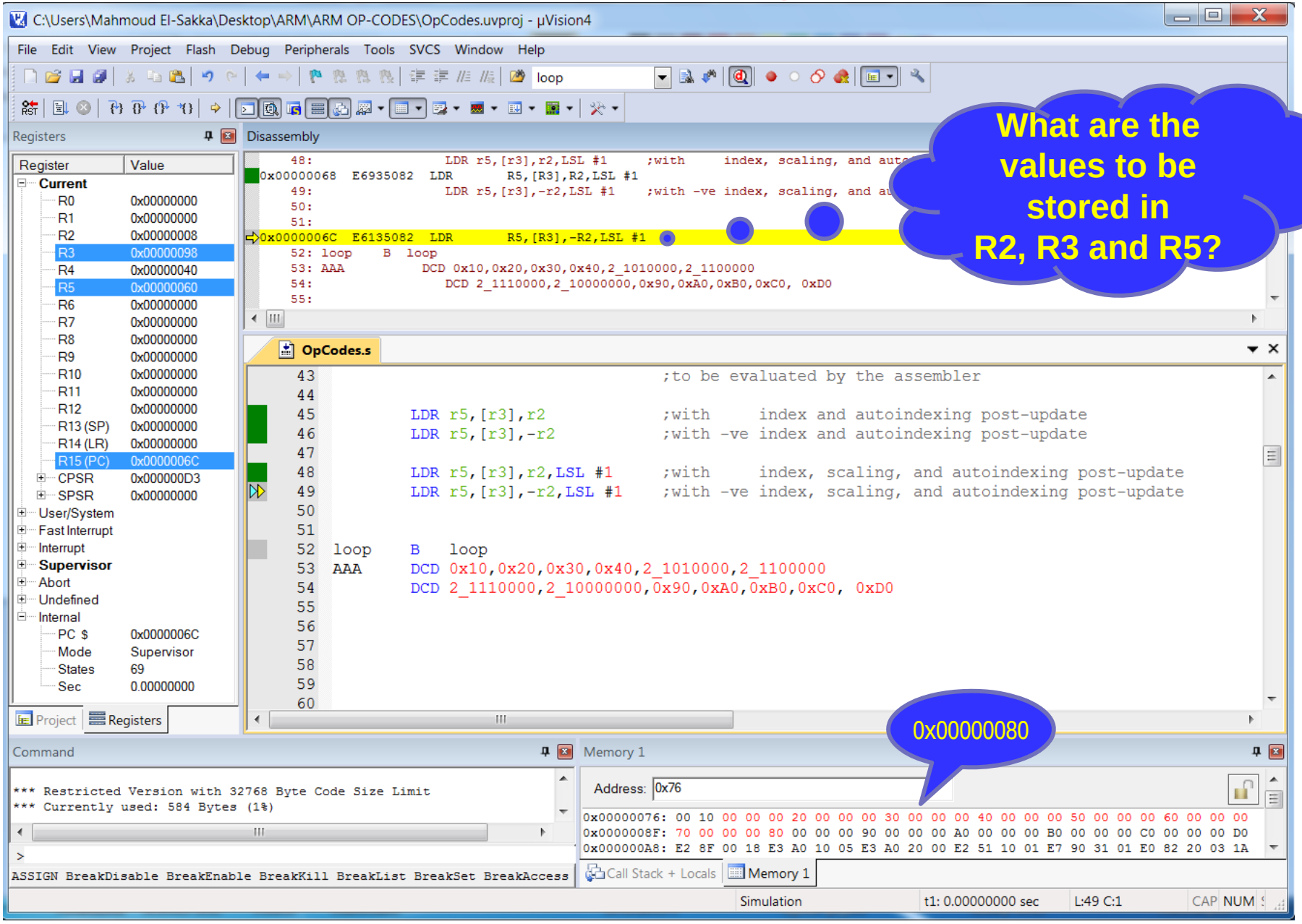
ARM Addressing Modes



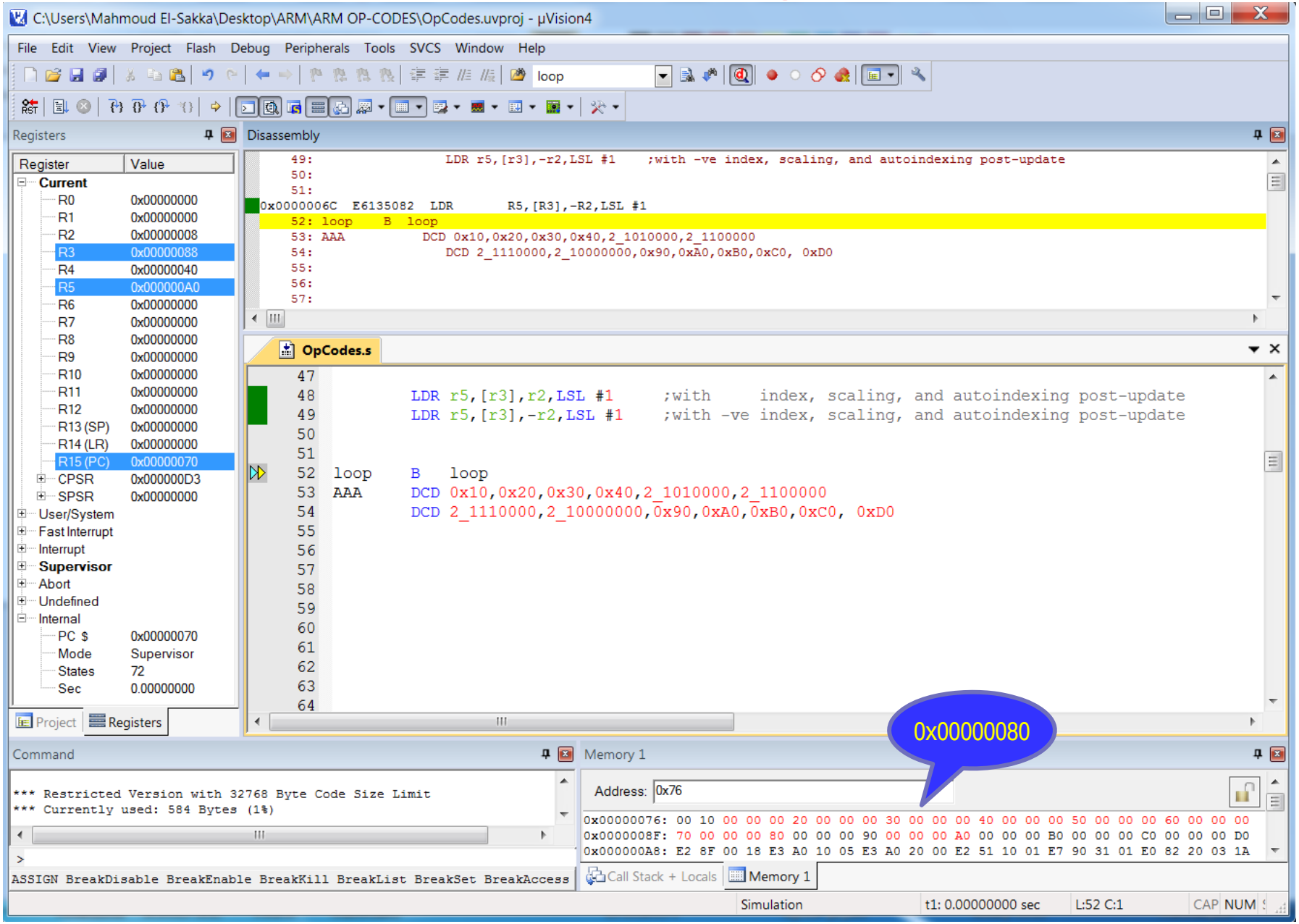
ARM Addressing Modes



ARM Addressing Modes

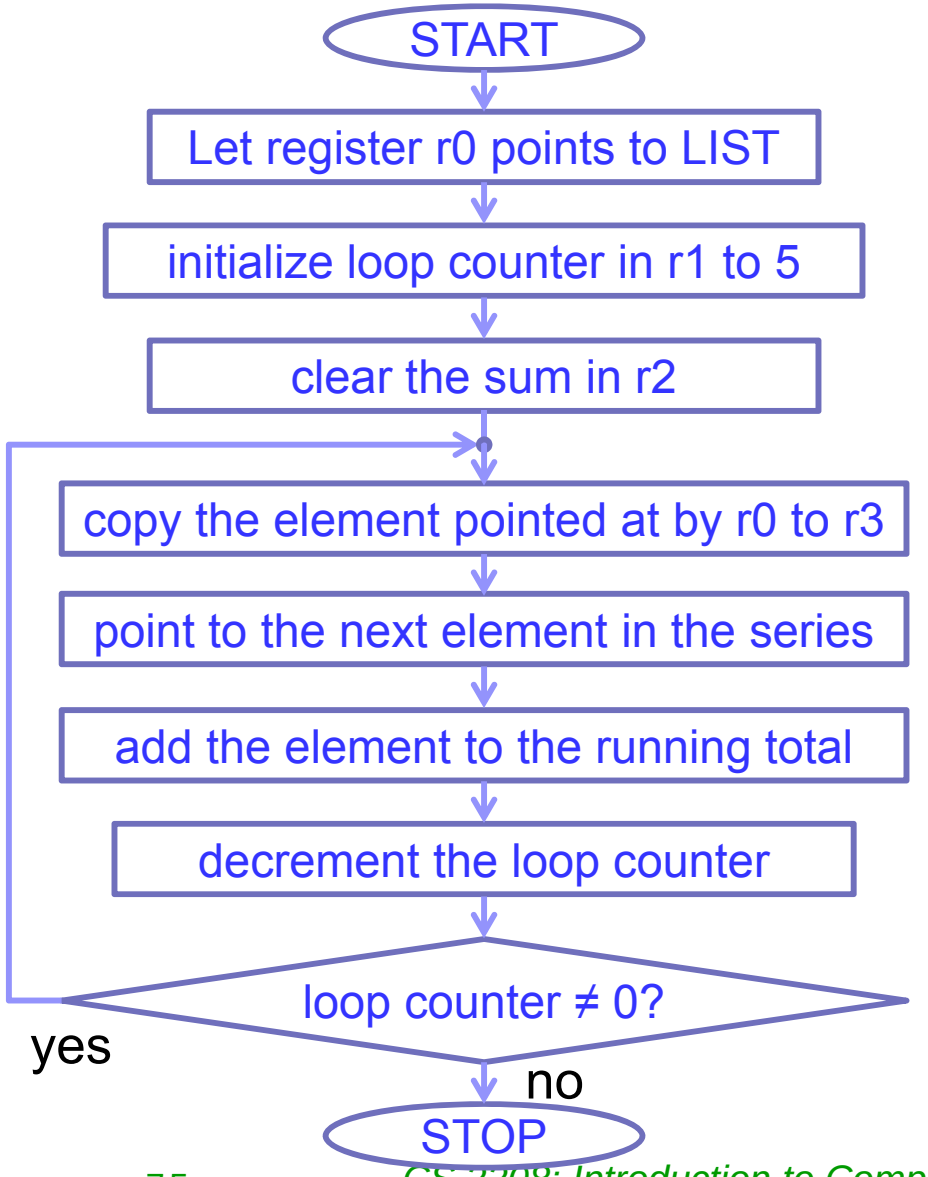


ARM Addressing Modes



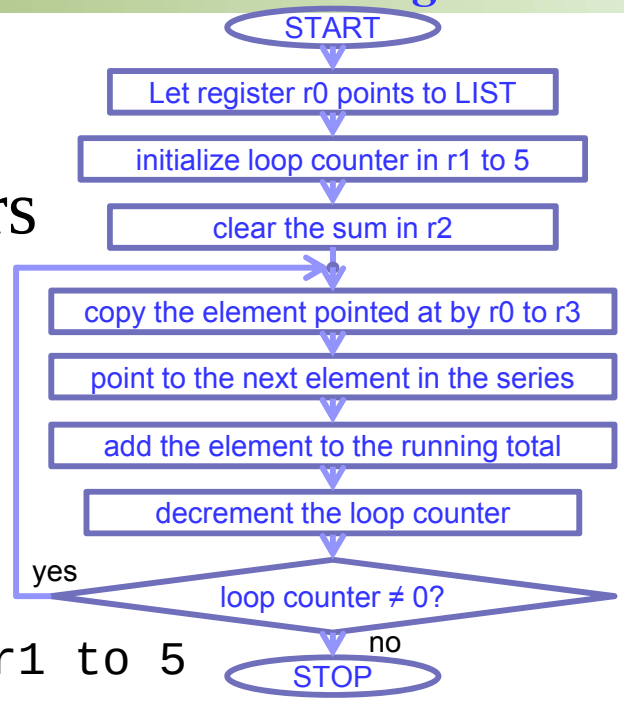
A Pointer Example

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
        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.