

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

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

Tutorial 10:

ARM Shift Instructions

Computer Science Department

CS2208: Introduction to Computer Organization and Architecture

Winter 2021-2022

Instructor: Mahmoud R. El-Sakka

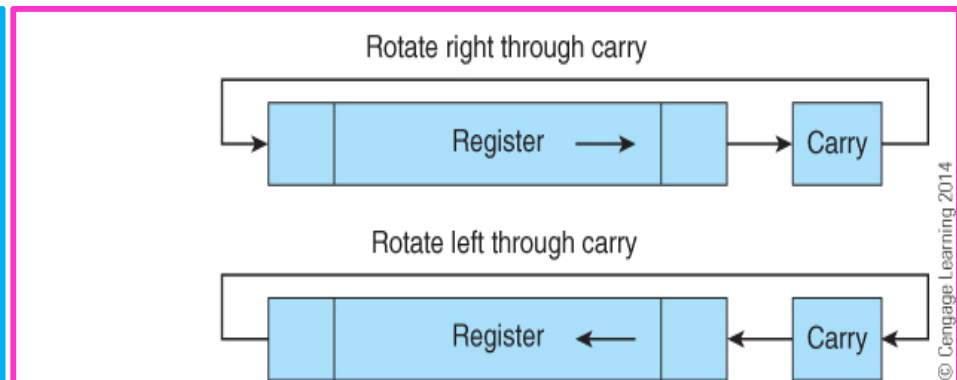
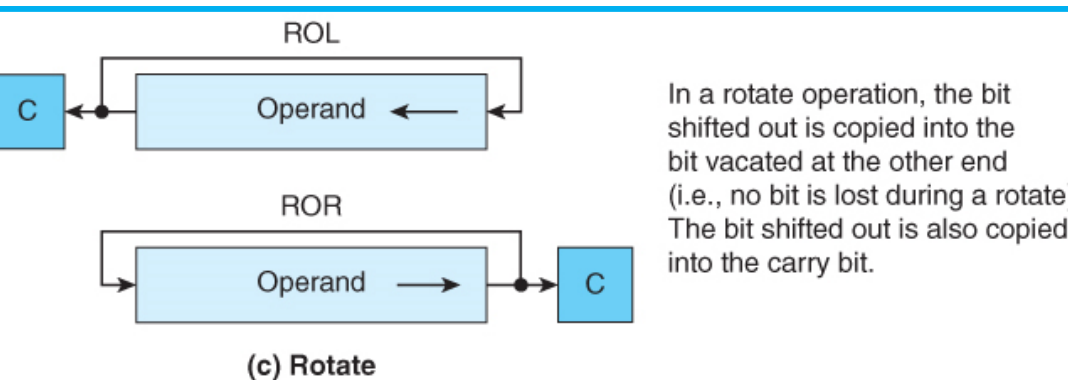
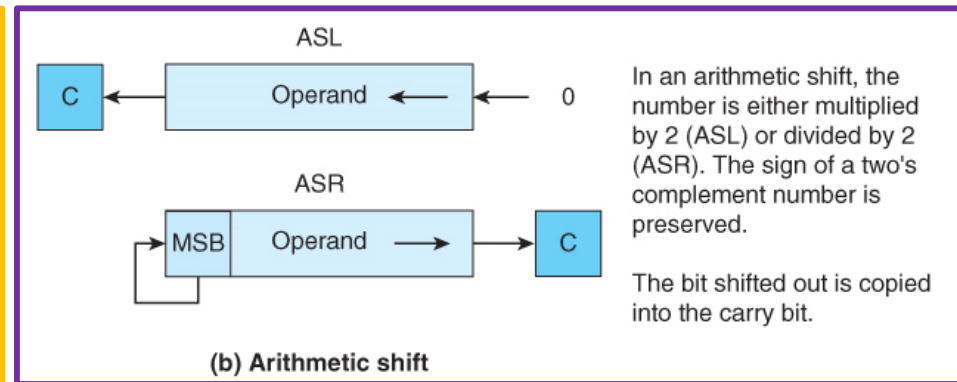
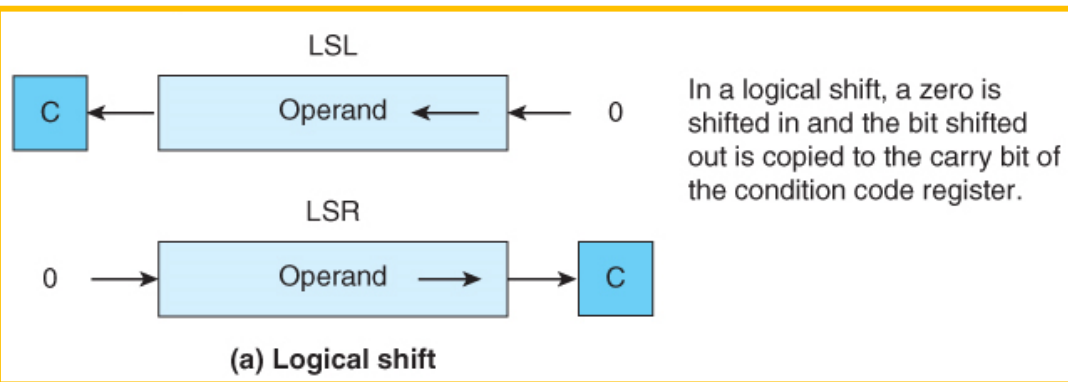
Office: MC-419

Email: elsakka@csd.uwo.ca

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*



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**
 - 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 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, =0xCCCCCCC ;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

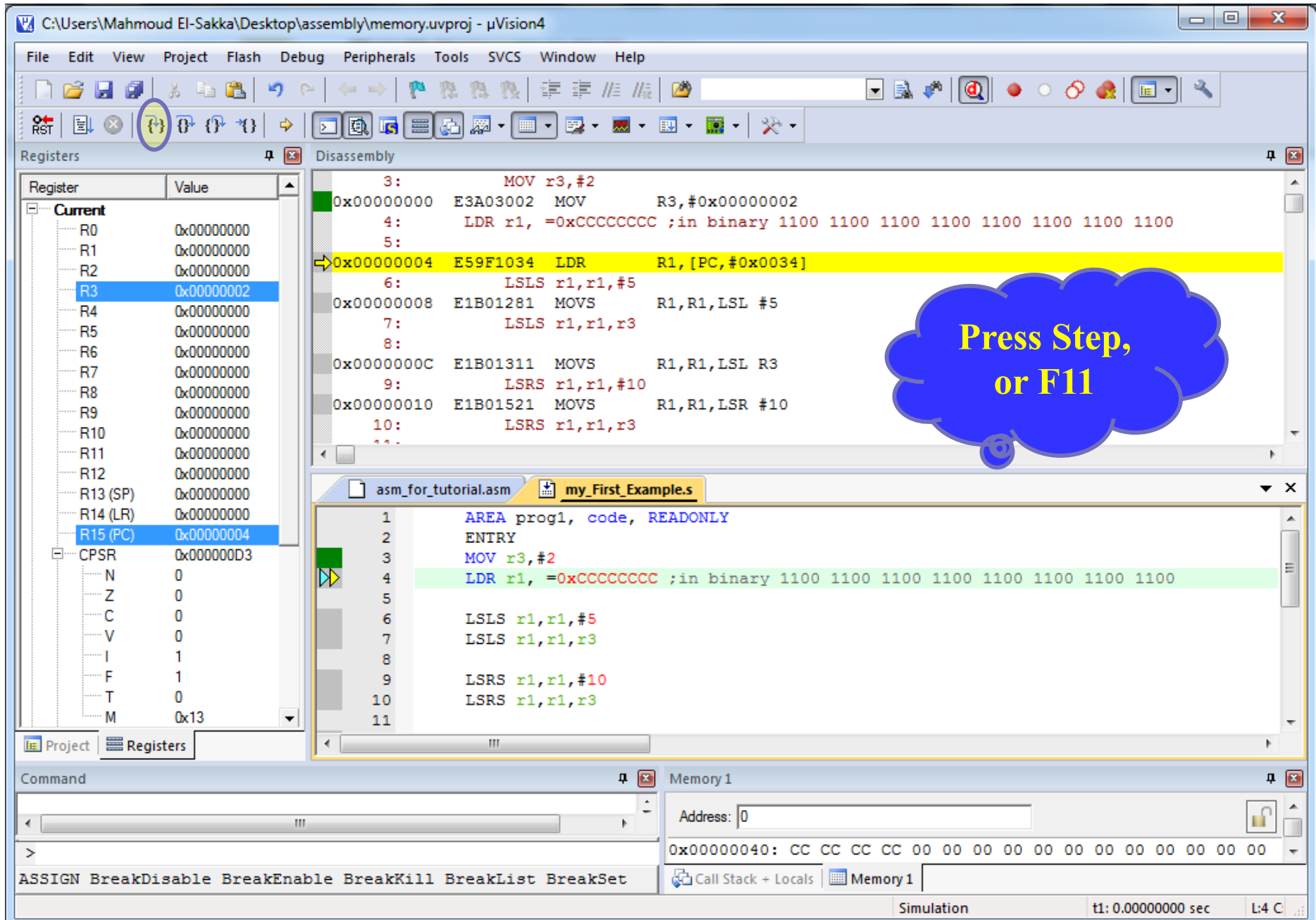
ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Lists registers R0 through R15 (PC) and CPSR. R15 (PC) is highlighted as the current register.
- Disassembly Window:** Shows assembly instructions with their addresses and hex codes. The instruction at address 0x00000000 is highlighted: `MOV r3, #2`.
- Source Code Window:** Shows the assembly source code for `my_First_Example.s`. The instruction `MOV r3, #2` is highlighted in green.
- Command Window:** Contains the command `ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet`.
- Memory Window:** Shows memory address 0x00000000 with the value `E3 A0 30 02 E5 9F 10 34 E1 B0 12 81 E1 B0 13 11`.

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

ARM's Data-Processing Instructions (Shift Operations)



The screenshot shows the uVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons, with the 'Step' button (a single step icon) circled in blue. A blue callout bubble with the text 'Press Step, or F11' points to this button.

The 'Registers' window on the left shows the current state of the registers. The 'Current' register is R3, with a value of 0x00000002. The 'CPSR' register is also visible, with a value of 0x000000D3.

The 'Disassembly' window shows the following assembly code:

```

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:

```

The 'asm_for_tutorial.asm' window shows the source code for 'my_First_Example.s':

```

1  AREA prog1, code, READONLY
2  ENTRY
3  MOV r3,#2
4  LDR r1, =0xCCCCCCCC ;in binary 1100 1100 1100 1100 1100 1100 1100 1100
5
6  LSLS r1,r1,#5
7  LSLS r1,r1,r3
8
9  LSRS r1,r1,#10
10 LSRS r1,r1,r3
11

```

The 'Memory' window at the bottom shows the address 0x00000040 with the value CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. R1 is highlighted with value 0xCCCCCCCC. R15 (PC) is 0x00000008.
- Disassembly Window:** Shows the assembly code being executed. The instruction at address 0x00000008 is `MOV r1, r1, LSL #5`.
- Assembly Editor:** Shows the source code for `my_First_Example.s`. The instruction `LSLS r1, r1, #5` is highlighted.
- Diagram:** A diagram illustrating the LSL (Logical Shift Left) operation. It shows a carry flag (C) and an operand being shifted left by 5 bits. The operand is 1100 1100 1100 1100 1100 1100 1100 1100. The result is 1001 1001 1001 1001 1001 1001 1001 0000.
- Call to Action:** A blue cloud bubble with the text "Press Step, or F11" is overlaid on the assembly editor.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

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 Window:**

```

3:      MOV r3,#2
0x00000000 E3A03002 MOV      R3,#0x00000002
4:      LDR r1, =0xCCCCCCCC ;in binary 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

```
- Source Code Window (asm_for_tutorial.asm):**

```

4      LDR r1, =0xCCCCCCCC ;in binary 1100 1100 1100 1100 1100 1100 1100
5
6      LSLS r1,r1,#5
7      LSLS r1,r1,r3
8
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

```
- Memory Window:**

Address: 0

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

Annotations:

- A yellow box highlights the **LSL** instruction: `LSL Operand ← 0`, with a blue box labeled **C** (Carry flag) pointing to the operand.
- A red arrow points from the **C** flag in the CPSR register to the **C** flag in the LSL instruction box.
- A green arrow points from the **C** flag in the CPSR register to the **C** flag in the LSL instruction box.
- A blue circle with the number **1** is placed next to the **LSL** instruction in the disassembly window.
- Two yellow boxes show the binary value of R1 before and after the LSL instruction:
 - Before: `1100 1100 1100 1100 1100 1100 1100 1100`
 - After: `1001 1001 1001 1001 1001 1001 1001 0000`

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

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 Window:**

```

3:      MOV r3,#2
0x00000000 E3A03002 MOV      R3,#0x00000002
4:      LDR r1, =0xCCCCCCCC ;in binary 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

```
- Source Code Window (asm_for_tutorial.asm):**

```

4      LDR r1, =0xCCCCCCCC ;in binary 1100 1100 1100 1100 1100 1100 1100
5
6      LSLS r1,r1,#5
7      LSLS r1,r1,r3
8
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

```
- Diagram:** A diagram showing the LSL operation: $C \leftarrow \text{Operand} \leftarrow 0$.
- Binary Representation:**

Initial value of R1: 1001 1001 1001 1001 1001 1001 1000 0000

Value after LSL R3 (5 shifts): 0110 0110 0110 0110 0110 0110 0000 0000
- Instruction:** A blue cloud contains the text "Press Step, or F11".

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Panel:** Shows the current state of registers. R1 is highlighted with a value of 0x66666600. The CPSR (Current Program Status Register) is also shown, with the C (Carry) flag set to 0.
- Disassembly Panel:** Shows the assembly code being executed. The instruction at address 0x00000004 is `LSLS r1, r1, #5`, which is highlighted in yellow. A red arrow points from this instruction to the R1 register in the Registers panel.
- Source Code Panel:** Shows the assembly code in `asm_for_tutorial.asm`. The instruction `LSLS r1, r1, #5` is highlighted in green. A blue arrow points from this instruction to the Disassembly panel.
- Memory Panel:** Shows the memory address 0x00000040 with the value `CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00`.
- Annotations:**
 - A yellow box in the top right corner shows the LSL instruction with a diagram: `C ← Operand ← 0`.
 - Two yellow boxes show the binary representation of the register R1 before and after the shift operation. The first box shows the value `1001 1001 1001 1001 1001 1001 1001 1000 0000`. The second box shows the value `0110 0110 0110 0110 0110 0110 0110 0000 0000`. A red circle with the number 0 is placed between the two boxes, indicating the shift amount.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x00000000
R1	0x66666600
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)	0x00000010
CPSR	0x000000D3
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
- Disassembly Window:**

```

3:      MOV r3,#2
0x00000000 E3A03002 MOV      R3,#0x00000002
4:      LDR r1, =0xCCCCCCC ;in binary 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

```
- Assembly Source Window (asm_for_tutorial.asm):**

```

6      LSLS r1,r1,#5
7      LSLS r1,r1,r3
8
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

```
- Diagram:** A box labeled "LSR" shows the operation: 0 → Operand → C. This indicates that the carry flag (C) is shifted into the least significant bit of the operand.
- Binary Representation:**
 - Initial value of R1: 0110 0110 0110 0110 0110 0110 0000 0000
 - Result after LSR: 0000 0000 0001 1001 1001 1001 1001 1001
- Instruction:** A blue cloud contains the text "Press Step, or F11".
- Memory Window:** Shows address 0x00000040 with data CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00.
- Command Window:** Contains the text "ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet".
- Status Bar:** Shows "Simulation" and "t1: 0.00000000 sec".

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x00000000
R1	0x00199999
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)	0x00000014
CPSR	0x20000003
N	0
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13
- Disassembly Window:**

```

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
  
```
- asm_for_tutorial.asm Window:**

```

6      LSLS r1,r1,#5
7      LSLS r1,r1,r3
8
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
  
```
- Memory Window:**

Address: 0

0x00000040: CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00
- Status Bar:**

Simulation t1: 0.00000000 sec L:10

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x00000000
R1	0x00199999
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)	0x00000014
CPSR	0x200000D3
N	0
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13
- Disassembly Window:**

```

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,r3

```
- Source Code Window (asm_for_tutorial.asm):**

```

6      LSLS r1,r1,#5
7      LSLS r1,r1,r3
8
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

```
- Diagram:** A diagram showing the LSR (Logical Shift Right) operation. It takes a value '0' and shifts it right through an 'Operand' box to a 'C' (Carry) register.
- Hexadecimal Values:**
 - Initial value: 0000 0000 0001 1001 1001 1001 1001 1001
 - Result after shift: 0000 0000 0000 0110 0110 0110 0110 0110
- Callout:** A blue cloud-shaped box with the text "Press Step, or F11".
- Memory Window:** Shows address 0x00000040 with data CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00.
- Command Window:** Contains the text "ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet".
- Status Bar:** Shows "Simulation" and "t1: 0.00000000 sec".

ARM's Data-Processing Instructions (Shift Operations)

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

- Registers Window:**

Register	Value
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 Window:**

```

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

```
- Source Window (asm_for_tutorial.asm):**

```

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 Window:**

Address: 0

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

Annotations in the image include a red arrow pointing from the R1 register value to the first instruction, a blue arrow pointing from the source code line 12 to the disassembly line 12, and a green arrow pointing from the source code line 14 to the disassembly line 14. A binary representation of the R1 value is shown as 0000 0000 0000 0110 0110 0110 0110 0110, with a circled '0' at the end.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. R15 (PC) is at 0x00000018, and the CPSR is at 0x000000D3. The C flag is 0.
- Disassembly Window:** Shows the assembly code for the program. The instruction at address 0x00000018 is `MOVSR R1, R1, ASR #2`.
- Diagram:** A diagram of the ASR (Arithmetic Shift Right) operation. It shows the MSB (Most Significant Bit) of the operand being shifted into the C (Carry) flag. The operand is shifted right by 2 positions.
- Callout Box:** A blue cloud-shaped box with the text "Press Step, or F11" pointing to the Step button in the IDE's toolbar.
- Memory Window:** Shows the memory address 0x00000040 with the value 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

The assembly code in the Disassembly window is as follows:

```

0x0000000C E1B01311 MOVSR R1, R1, LSL R3
9:          LSRS r1, r1, #10
0x00000010 E1B01521 MOVSR R1, R1, LSR #10
10:         LSRS r1, r1, r3
11:
0x00000014 E1B01331 MOVSR R1, R1, LSR R3
12:         ASRS r1, r1, #2
0x00000018 E1B01141 MOVSR R1, R1, ASR #2
13:         LSLS r1, r1, #15
0x0000001C E1B01781 MOVSR R1, R1, LSL #15
14:         ASRS r1, r1, #16
15:
0x00000020 E1B01841 MOVSR R1, R1, ASR #16
16:         LSRS r1, r1, r3
17:
0x00000024 E1B01331 MOVSR R1, R1, LSR R3
18:         RORS r1, r1, #4
19:         RORS r1, r1, r3

```

The assembly code in the Source window is as follows:

```

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

```

ARM's Data-Processing Instructions (Shift Operations)

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

- Registers Window:**

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 Window:**

```

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,r3

```
- Source Code Window (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 Window:**

Address: 0

0x00000040: CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00
- Annotations:**
 - A red arrow points from the value of R1 (0x00019999) to the LSL instruction at address 0x0000001C.
 - A blue arrow points from the CPSR register to the LSL instruction at address 0x0000001C.
 - A green arrow points from the CPSR register to the LSL instruction at address 0x0000001C.
 - A yellow box highlights the binary value 0000 0000 0000 0001 1001 1001 1001 1001.
 - A red circle with the number 1 is next to the binary value.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. R15 (PC) is at 0x0000001C, and CPSR is at 0x200000D3. The 'C' (Carry) flag is set to 1.
- Disassembly Window:** Shows the assembly code. The instruction at address 0x0000001C is `MOVSL R1, R1, LSL R3`, which is highlighted in yellow.
- Diagram:** A diagram labeled 'LSL' shows a box 'C' (Carry) with an arrow pointing to a box 'Operand', which has an arrow pointing to a box '0'. This illustrates the shift operation where the carry flag is shifted into the operand.
- Assembly Source Window:** Shows the source code for 'my_First_Example.s'. The instruction `LSLS r1, r1, #15` is highlighted in green.
- Callout Box:** A blue cloud-shaped box with the text "Press Step, or F11" is overlaid on the assembly source window.
- Memory Window:** Shows the memory address 0x00000040 with the value 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

ARM's Data-Processing Instructions (Shift Operations)

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

- Registers Window:** Shows the current state of ARM registers. R1 is highlighted with a value of 0xCCCC8000. Other registers like R0, R2, R3, etc., are at 0x00000000. R15 (PC) is at 0x00000020. CPSR is at 0x800000D3.
- Disassembly Window:** Shows the assembly code being executed. The instructions are:
 - 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
- Source Code Window (asm_for_tutorial.asm):** Shows the original assembly code:
 - 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
 - 21 RRXS r1, r1
- Binary Representation:** A yellow box highlights the binary value of R1: 1100 1100 1100 1100 1000 0000 0000 0000. A red arrow points from the ASRS instruction in the disassembly window to this binary value.
- Command Window:** Shows the command: ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
- Memory Window:** Shows the memory address 0x00000040 with the value CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x00000000
R1	0xCCCC8000
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)	0x00000020
CPSR	0x800000D3
N	1
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
- Disassembly Window:**

```

13:      LSLS r1,r1,#15
0x0000001C E1B01781 MOVS    R1,R1,LSL #15
14:      ASRS r1,r1,#16
15:      MOVS r1,r1,ASR #16
0x00000020 E1B01841 MOVS    R1,R1,ASR #16
16:      ASRS r1,r1,r3
17:      MOVS r1,r1,ASR R3
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:      MOVS r1,r1,ROR R3
0x0000002C E1B01371 MOVS    R1,R1,ROR R3

```
- Source Code Window (asm_for_tutorial.asm):**

```

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
21      RRXS r1,r1

```
- ASR Diagram:**

```

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

```
- Bit Patterns:**

Initial value of R1: 1100 1100 1100 1100 1000 0000 0000 0000

Result after ASRS r1, r1, #16: 1111 1111 1111 1111 1100 1100 1100 1100
- Callout:** Press Step, or F11

ARM's Data-Processing Instructions (Shift Operations)

[illegible]

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Panel:** Shows the current state of registers. R15 (PC) is at 0x00000024. CPSR is 0xA00000D3. The 'C' (Carry) flag is set to 1.
- Disassembly Panel:** Shows the assembly code being executed. The current instruction is `ASRS r1,r1,r3` at address 0x00000024.
- ASR Diagram:** A diagram in the top right shows the ASR instruction taking an 'Operand' and shifting it to the right by the value in the 'C' (Carry) flag. The 'MSB' (Most Significant Bit) is shifted into the 'C' flag.
- Bit Patterns:** Two bit patterns are highlighted in yellow boxes:
 - Top: 1111 1111 1111 1111 1100 1100 1100 1100
 - Bottom: 1111 1111 1111 1111 1111 0011 0011 0011
- Code Editor:** Shows the assembly code for 'my_First_Example.s':


```

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:
21: RRXS r1,r1
22: RRXS r1,r1
23: RRXS r1,r1
      
```
- Command Panel:** Shows the command `ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet`.
- Memory Panel:** Shows the memory address 0x00000040 with the value `CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00`.

A blue cloud bubble with the text "Press Step, or F11" is overlaid on the code editor.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays the uVision4 IDE interface for an ARM assembly project. The **Registers** window on the left shows the current state of the processor registers. Register R1 contains the value 0xFFFF333. The **Disassembly** window shows the following instructions:

```

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

```

A yellow box highlights the instruction `RORS r1,r1,#4` at address 0x00000028. A green box highlights the instruction `RORS r1,r1,r3` at address 0x00000029. A binary value `1111 1111 1111 1111 1111 0011 0011 0011` is shown in a yellow box, with a red arrow pointing to register R1 and a green arrow pointing to the `RORS r1,r1,r3` instruction. The **Registers** window shows the following values:

Register	Value
R0	0x00000000
R1	0xFFFF333
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

The **Command** window shows the following text:

```

>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet

```

The **Memory** window shows the address 0x00000040 with the value `CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00`.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. R15 (PC) is at 0x00000028. CPSR is 0x800000D3. The C flag is 0.
- Disassembly Window:** Shows assembly instructions. Instruction 18 is highlighted: `RORS r1, r1, #4`. Instruction 19 is `RORS r1, r1, r3`.
- Source Code Window:** Shows the assembly file `asm_for_tutorial.asm` with instructions 15 to 25. Instruction 18 is highlighted: `RORS r1, r1, #4`.
- Diagram:** A diagram of the ROR instruction. It shows an 'Operand' box with an arrow pointing to a 'C' (Carry) box. The diagram is labeled 'ROR'.
- Bit Pattern:** A 32-bit binary representation of the register value after the ROR instruction. The bits are: 1111 1111 1111 1111 1111 0011 0011 0011. The last four bits (0011) are highlighted in red.
- Callout:** A blue cloud-shaped callout with the text "Press Step, or F11" pointing to the instruction 18 in the source code window.

ARM's Data-Processing Instructions (Shift Operations)

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

- Registers Window:**

Register	Value
R0	0x00000000
R1	0x3FFFFFF3
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)	0x0000002C
CPSR	0x000000D3
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
- Disassembly Window:**

```

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 CCCCCCCC INDEG   R0,R0,R0
  
```
- Source File Window (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 Window:**

Address: 0

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

A diagram illustrates a 32-bit shift operation. The initial value is `0011 1111 1111 1111 1111 1111 0011 0011`. A green arrow indicates a right shift by 4 bits. The result is `0011 1111 1111 1111 1111 1111 0011 0011` with a `0` in the carry-out position.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x00000000
R1	0x3FFFFFF3
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)	0x0000002C
CPSR	0x000000D3
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13
- Disassembly Window:**

```

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 CCCCCCCC INDEG R0,R0,R0

```
- Source Code Window (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

```
- Diagram:** A box labeled "ROR" contains a diagram showing an "Operand" being rotated right into a register "C".
- Hexadecimal Values:**
 - Initial value (R1): 0011 1111 1111 1111 1111 1111 0011 0011
 - Value after RORS r1,r1,r3: 1100 1111 1111 1111 1111 1111 1100 1100
- Callout:** A blue cloud-shaped box with the text "Press Step, or F11" and a circular arrow icon.
- Memory Window:** Shows address 0x00000040 with data CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00.
- Status Bar:** Simulation, t1: 0.00000000 sec, L:19.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays the uVision4 IDE interface for an ARM assembly project. The **Registers** window on the left shows the current state of registers, with R1 at 0xCFFFFFFC and CPSR at 0xA00000D3. The **Disassembly** window shows the assembly code, with instructions like `MOVS R1, R1, ROR #4`, `RORS r1, r1, r3`, and `RRXS r1, r1`. A yellow box highlights the instruction `RRXS r1, r1` at address 0x00000030, with a red arrow pointing to the `RRXS` instruction in the Disassembly window. A green arrow points from the `RRXS` instruction to the `RRXS r1, r1` instruction in the assembly window. A yellow box highlights the value `1100 1111 1111 1111 1111 1111 1100 1100` in the Disassembly window, with a red arrow pointing to the `RRXS` instruction in the Disassembly window. A green arrow points from the `RRXS` instruction to the `RRXS r1, r1` instruction in the assembly window. A red circle with the number '1' is next to the yellow box.

ARM's Data-Processing Instructions (Shift Operations)

Rotate right through carry

```

    graph LR
    Register[Register] --> Carry[Carry]
    Carry --> Register
  
```

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 CCCCCCCC STCGTL p12,CR12,[R12],{204}
  
```

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
  
```

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 1

Address: 0

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

Simulation t1: 0.00000000 sec L:21

Press Step, or F11

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays the uVision4 IDE interface for an ARM assembly project. The main window shows the disassembly of assembly code, with the following instructions visible:

```

0x00000028 E1B01261 MOVs    R1,R1,ROR #4
19:          RORS    r1,r1,r3
20:
0x0000002C E1B01371 MOVs    R1,R1,ROR R3
21:          RRXS    r1,r1
22:
0x00000030 E1B01061 MOVs    R1,R1,RRX
23:          RRXS    r1,r1
24:
0x00000034 E1B01061 MOVs    R1,R1,RRX
25:          RRXS    r1,r1
26:          RRXS    r1,r1
27:          RRXS    r1,r1
28:          RRXS    r1,r1
29:          END

```

The Registers window on the left shows the current state of the registers:

Register	Value
R0	0x00000000
R1	0xE7FFFE6
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)	0x00000034
CPSR	0x800000D3
N	1
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13

The CPSR register shows the following flags: N=1, Z=0, C=0, V=0, I=1, F=1, T=0, M=0x13.

The Memory window at the bottom shows the address 0x00000040 with the value 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

A binary representation of the R1 register value is shown as 1110 0111 1111 1111 1111 1111 1111 0110. A green circle with the number 0 is also visible on the right side of the image.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of registers. R15 (PC) is at 0x00000034. CPSR is 0x800000D3. The 'C' (Carry) flag is 0.
- Disassembly Window:** Shows assembly instructions. Line 22 is highlighted: `RRXS r1, r1`. The instruction list includes:
 - 0x00000028: `MOVN R1, R1, ROR #4`
 - 19: `RORS r1, r1, r3`
 - 20: (blank)
 - 0x0000002C: `MOVN R1, R1, ROR R3`
 - 21: `RRXS r1, r1`
 - 0x00000030: `MOVN R1, R1, RRX`
 - 22: `RRXS r1, r1` (highlighted)
 - 0x00000034: `MOVN R1, R1, RRX`
 - 23: `RRXS r1, r1`
 - 0x00000038: `MOVN R1, R1, RRX`
 - 24: `RRXS r1, r1`
 - 0x0000003C: `MOVN R1, R1, RRX`
 - 0x00000040: `CCCCCCC STCGTL p12, CR12, [R12] {204}`
 - 0x00000044: `CCCCCCC INDEC R0, R0, R0`
- Source Code Window:** Shows the assembly file `asm_for_tutorial.asm` with instructions:
 - 15: (blank)
 - 16: `ASRS r1, r1, r3`
 - 17: (blank)
 - 18: `RORS r1, r1, #4`
 - 19: `RORS r1, r1, r3`
 - 20: (blank)
 - 21: `RRXS r1, r1`
 - 22: `RRXS r1, r1` (highlighted)
 - 23: `RRXS r1, r1`
 - 24: `RRXS r1, r1`
 - 25: `END`
- Diagram:** A red box highlights the 'Rotate right through carry' operation. It shows a 'Register' box with an arrow pointing to a 'Carry' box, and another arrow from the 'Carry' box back to the 'Register' box.
- Bit Patterns:** Two rows of bit patterns are shown:
 - 1110 0111 1111 1111 1111 1111 1111 1110 0110
 - 0111 0011 1111 1111 1111 1111 1111 1111 0011
- Annotation:** A blue cloud contains the text "Press Step, or F11".

ARM's Data-Processing Instructions (Shift Operations)

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

- Registers Window:** Shows the current state of ARM registers. R1 is highlighted with a value of 0x73FFFFFF. Other registers like R0, R2, R3, etc., are shown with zero values. The CPSR register is also visible with various flags set.
- Disassembly Window:** Shows the disassembled instructions. The instruction at address 0x00000038 is highlighted: `MOV R1, R1, RRX`. This instruction is annotated with the binary value 0111 0011 1111 1111 1111 1111 1111 0011, which represents the value in R1 shifted right by one position (RRX).
- Source Code Window:** Shows the assembly code for the example. The instructions are:


```

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 Window:** Shows the memory address 0x00000040 with a value of 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

Annotations in the image include a red arrow pointing from the R1 register to the disassembly window, a blue arrow pointing from the disassembly window to the source code window, and a green arrow pointing from the source code window to the disassembly window. A yellow box highlights the binary value 0111 0011 1111 1111 1111 1111 1111 0011, and a red circle with the number 0 is placed next to it.

ARM's Data-Processing Instructions (Shift Operations)

Rotate right through carry

```

    graph LR
      Register[Register] --> Carry[Carry]
      Carry --> Register
  
```

Registers

Register	Value
R0	0x00000000
R1	0x73FFFFFF
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)	0x00000038
CPSR	0x000000D3
N	0
Z	0
C	0
V	0
I	1
F	1
T	0
M	0x13

Disassembly

```

0x00000028 E1B01261 MOVN    R1,R1,ROR #4
19:         RORS    r1,r1,r3
20:
0x0000002C E1B01371 MOVN    R1,R1,ROR R3
21:         RRXS    r1,r1
22:         RRXS    r1,r1
23:         RRXS    r1,r1
24:         RRXS    r1,r1
0x00000038 E1B01061 MOVN    R1,R1,RRX
24:         RRXS    r1,r1
0x0000003C E1B01061 MOVN    R1,R1,RRX
0x00000040 CCCCCCCC STCGTL   p12,CR12,[R12],#204
0x00000044 CCCCCCCC STCGTL   p12,CR12,[R12],#204
  
```

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
  
```

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 1

Address: 0

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

Simulation t1: 0.00000000 sec L:23

Press Step, or F11

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays the uVision4 IDE interface for an ARM assembly project. The **Registers** window on the left shows the current state of the processor registers. Register R1 contains the value 0x39FFFFFF, and the CPSR (Current Program Status Register) shows the Carry flag (C) set to 1. The **Disassembly** window in the center shows the assembly code being executed. The instructions include MOVs, RORS, and RRXS. A red arrow points from the R1 register to the instruction at address 0x0000002C, and a green arrow points from the CPSR C flag to the instruction at address 0x0000003C. A binary value 0011 1001 1111 1111 1111 1111 1111 1001 is highlighted in a yellow box, with a red arrow pointing to R1 and a green arrow pointing to the CPSR C flag. The **asm_for_tutorial.asm** file is open, showing the assembly code for **my_First_Example.s**.

Register	Value
R0	0x00000000
R1	0x39FFFFFF
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)	0x0000003C
CPSR	0x200000D3
N	0
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13

```

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 STCCTL   p12,CR12,[R12] {204}
0x00000044 CCCCCCCC STCCTL   p12,CR12,[R12] {204}
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
  
```

0011 1001 1111 1111 1111 1111 1111 1001

Simulation t1: 0.00000000 sec L:24

ARM's Data-Processing Instructions (Shift Operations)

Rotate right through carry

```

    graph LR
    Register[Register] --> Carry[Carry]
    Carry --> Register
  
```

Registers

Register	Value
R0	0x00000000
R1	0x39FFFFFF
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)	0x0000003C
CPSR	0x200000D3
N	0
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
22: RRXS r1,r1
0x00000030 E1B01061 MOVS R1,R1,RRX
23: RRXS r1,r1
0x00000034 E1B01061 MOVS R1,R1,RRX
24: RRXS r1,r1
0x00000038 E1B01061 MOVS R1,R1,RRX
0x0000003C E1B01061 MOVS R1,R1,RRX
0x00000040 CCCCCCCC STCGTL p12,CR12,[R12],{204}
  
```

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
  
```

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 1

Address: 0

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

Simulation t1: 0.00000000 sec L:24

Press Step, or F11

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

Register	Value
R0	0x00000000
R1	0x9CFFFFFF
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)	0x00000044
CPSR	0xA00000D3
N	1
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13
- Disassembly Window:**

```

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 STCGT     p12,CR12,[R12],{204}
0x00000044 00000000 ANDEQ     R0,R0,R0

```
- Source Code Window (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 Window:**

Address: 0

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

A red arrow points from the R1 register value (0x9CFFFFFF) to the assembly code line 'RRXS r1, r1'. A green arrow points from the '1' in the binary sequence '1001 1100 1111 1111 1111 1111 1111 1100' to the '1' in the assembly code line 'RRXS r1, r1'.

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
```

```
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”

