

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

Fall 2022-2023

Instructor: Mahmoud R. El-Sakka

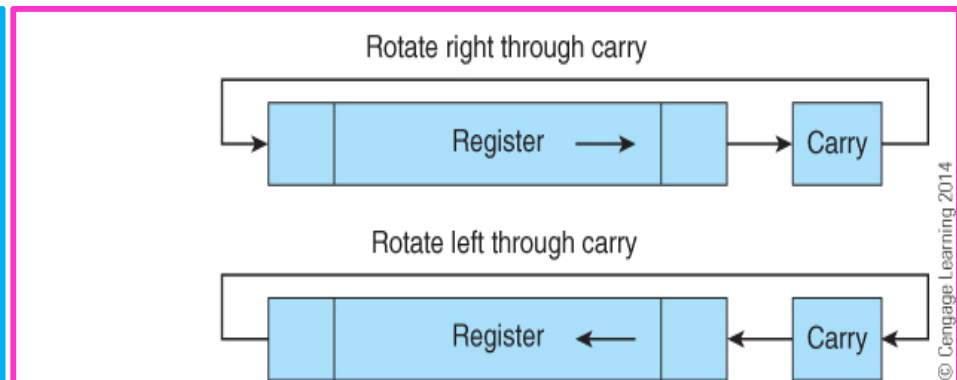
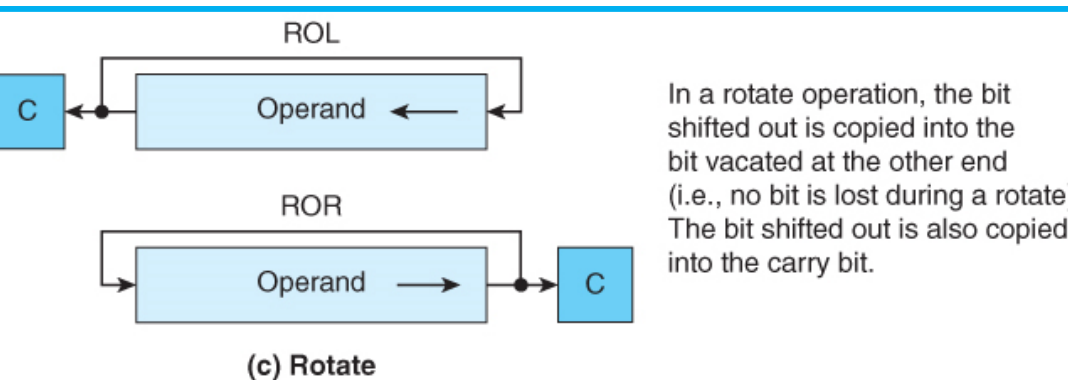
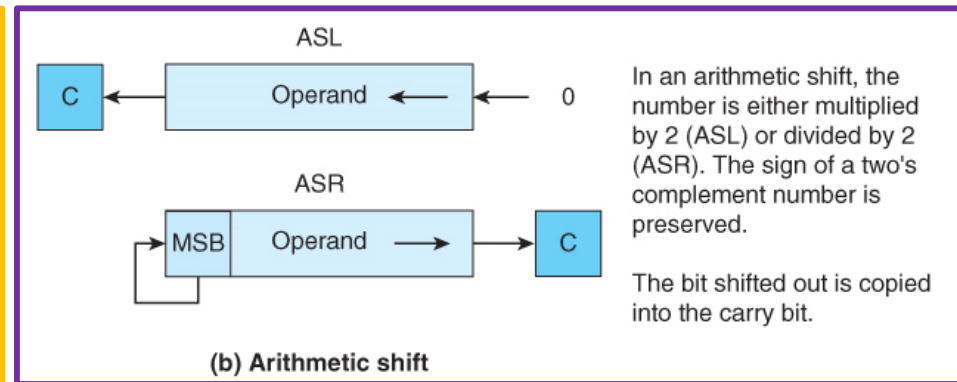
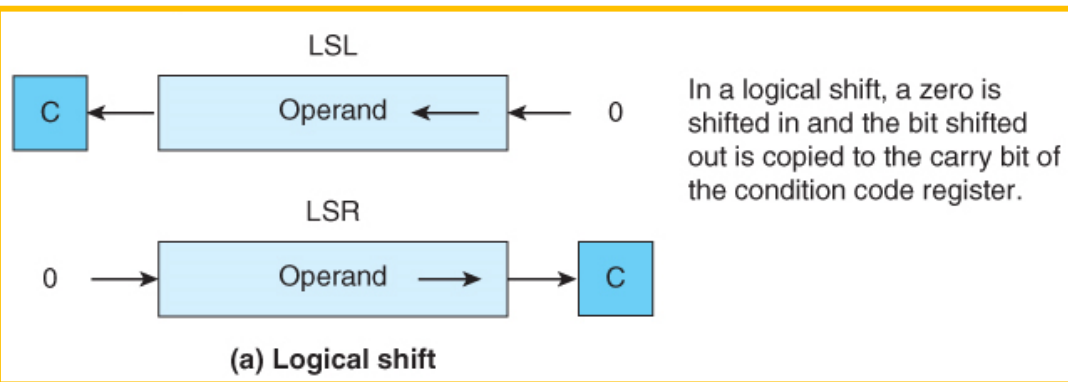
Office: MC-419

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*



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) LSL #0 = LSR #0 = ASR #0.
 - **LSR**: the range is from **#1** to **#32** (32 different values) so no #0 for LSR, ASR.
 - **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, =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)

The screenshot shows the µVision4 IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons, with the Step button (a square with a right-pointing arrow) 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 ARM registers. The PC register (R15) is highlighted, showing a value of 0x00000000. The CPSR register is also visible, showing a value of 0x000000D3.

The Disassembly window shows the following instructions:

```

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 Source window shows the corresponding assembly code for the selected instruction:

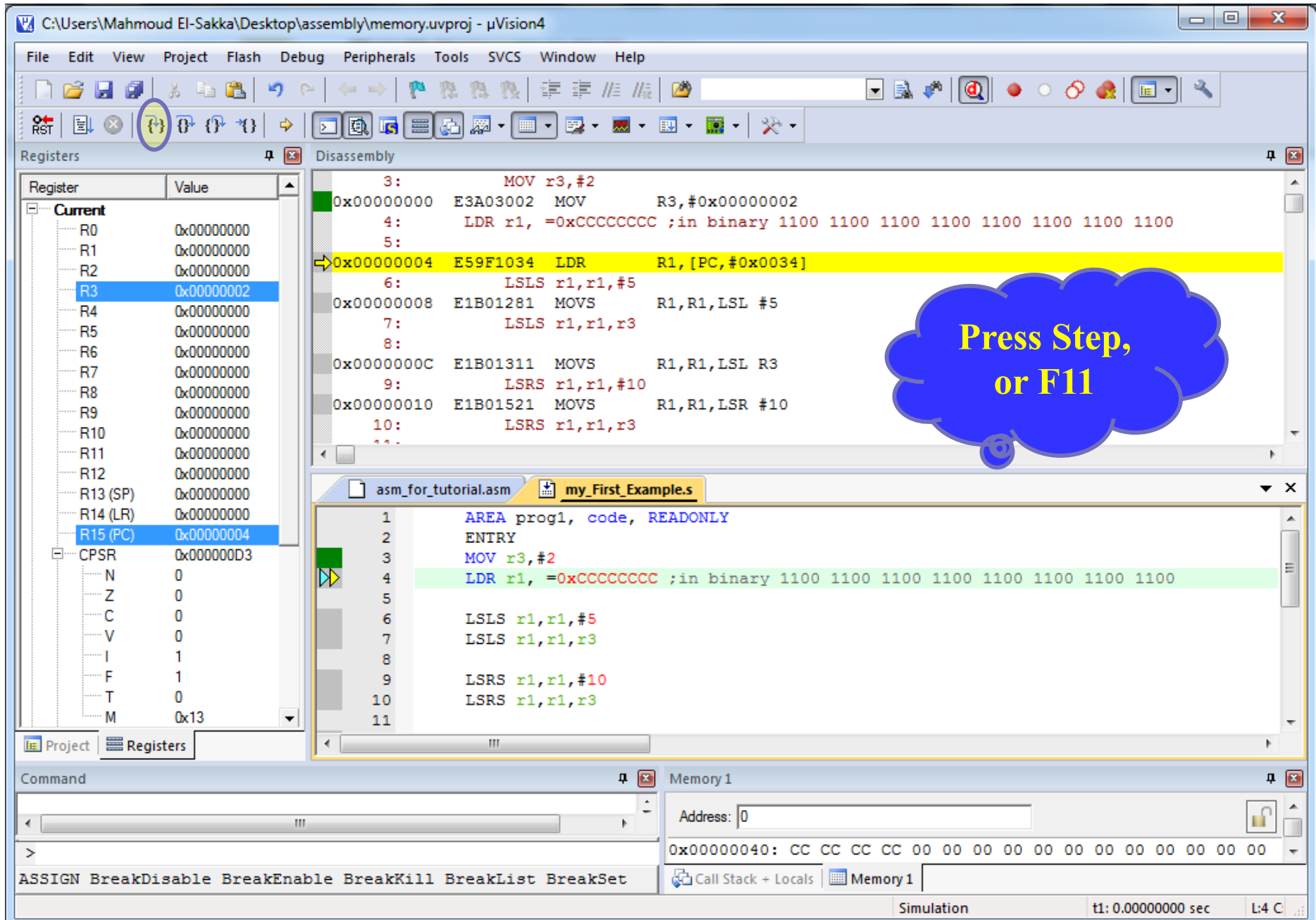
```

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 Command window at the bottom shows the simulation status: "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	0x00000000
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)	0x00000004
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, =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

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

```

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

```
- Toolbar:** The 'Step' button (represented by a right-pointing arrow) is circled in blue.
- Callout:** A blue cloud-shaped bubble contains the text "Press Step, or F11".
- Memory Window:** Shows address 0x00000040 with hex 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:** Shows the current state of registers. R1 is highlighted with a value of 0xCCCCCCCC. R15 (PC) is 0x00000008.
- Disassembly Window:** Shows the assembly code being executed. The instruction at address 0x00000008 is `MOV r1, r1, LSL #5`, which is highlighted in yellow.
- Assembly Source Window:** Shows the source code for `my_First_Example.s`. The instruction `LSLS r1, r1, #5` is highlighted in green.
- Diagram:** A diagram illustrating the LSL (Logical Shift Left) operation. It shows a box labeled 'Operand' with a value of 0. An arrow points from the 'Operand' box to a box labeled 'C' (Carry flag). The diagram is labeled 'LSL'.
- Binary Representation:** Two binary representations of the value 0xCCCCCCCC are shown. The first is `1100 1100 1100 1100 1100 1100 1100 1100` and the second 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 source window.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Panel:**

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 Panel:**

```

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

```
- Source File Panel (asm_for_tutorial.asm):**

```

4      LDR r1, =0xCCCCCCC ;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 Panel:**

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 R3** instruction in the disassembly panel.
- Two binary strings are shown in yellow boxes:
 - `1100 1100 1100 1100 1100 1100 1100 1100` (highlighted in red)
 - `1001 1001 1001 1001 1001 1001 1001 0000` (highlighted in green)

ARM's Data-Processing Instructions (Shift Operations)

Registers

Register	Value
R0	0x00000000
R1	0x99999980
R2	0x00000000
R3	0x00000002
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000000C
CPSR	0xA00000D3
N	1
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13

Disassembly

```

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

```

asm_for_tutorial.asm

```

4      LDR r1, =0xCCCCCCC ;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

```

my_First_Example.s

```

4      LDR r1, =0xCCCCCCC ;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 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:7 C

Diagram: LSL

```

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

```

Binary Representation:

1001 1001 1001 1001 1001 1001 1001 1000 0000

0110 0110 0110 0110 0110 0110 0110 0000 0000

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 labeled "LSL" shows a diagram of the shift operation: `C ← Operand ← 0`.
 - Two yellow boxes show the binary representation of the register R1 before and after the shift. 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 "5" indicates the shift amount.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Panel:**

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 Panel:**

```

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 Panel (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 a logical shift right where the carry flag (C) is shifted into the operand.
- Binary Representation:**

Initial value of R1: 0110 0110 0110 0110 0110 0110 0000 0000

Result after LSR #10: 0000 0000 0001 1001 1001 1001 1001 1001
- Callout:** A blue cloud-shaped box with a play button icon 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 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

Flags in CPSR: N=0, Z=0, C=1, V=0, I=1, F=1, T=0, M=0x13.
- Disassembly Window:**

Address	Instruction	Comment
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:**

Line	Instruction
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

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".

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays the Keil uVision4 IDE interface during a simulation. The main window is divided into several panes:

- Registers:** Located on the left, it shows the current state of registers R0 through R15 and the CPSR. R1 is highlighted with a blue background and contains the value 0x00066666. CPSR is also highlighted and contains 0x000000D3.
- Disassembly:** The central pane shows the disassembled instructions. A red arrow points from the instruction `LSRS r1, r1, r3` (address 0x00000010) to the R1 register. A green arrow points from the instruction `ASRS r1, r1, #2` (address 0x00000012) to the CPSR register. A yellow box highlights the instruction `ASRS r1, r1, #2` and its corresponding assembly code `0000 0000 0000 0110 0110 0110 0110 0110`.
- asm_for_tutorial.asm:** The bottom pane shows the source code. A green arrow points from the instruction `ASRS r1, r1, #2` (line 12) to the CPSR register.
- Memory 1:** The bottom right pane shows the memory address 0x00000040 with the value `CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00`.

The status bar at the bottom indicates the simulation is running, with a time of 0.00000000 seconds and a label 'L:12'.

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. The CPSR register shows the Carry flag (C) is 0.
- Disassembly Window:** Shows the assembly code being executed. The instruction at address 0x00000018 is `ASRS r1,r1,#2`, which is highlighted in yellow.
- Source File Window:** Shows the assembly source code `asm_for_tutorial.asm` with the same instruction `ASRS r1,r1,#2` highlighted in green.
- Diagram:** A diagram of the ASR (Arithmetic Shift Right) instruction. It shows the MSB (Most Significant Bit) of the operand being shifted into the Carry flag (C). The operand is shifted right by a specified number of positions.
- Hexadecimal Representation:** Two rows of hexadecimal data are shown, representing the operand and the result of the shift operation. The first row is `0000 0000 0000 0110 0110 0110 0110 0110` and the second row is `0000 0000 0000 0001 1001 1001 1001 1001`. The last two digits of each row are highlighted in red.
- Call to Action:** A blue cloud-shaped callout with the text "Press Step, or F11" is overlaid on 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	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 `LSRS r1,r1,#10` instruction to the R1 register.
- A blue arrow points from the `LSLS r1,r1,#15` instruction to the CPSR register.
- A green arrow points from the `RORS r1,r1,r3` instruction to the CPSR register.
- 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)

Registers

Register	Value
R0	0x00000000
R1	0x00019999
R2	0x00000000
R3	0x00000002
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000001C
CPSR	0x200000D3
N	0
Z	0
C	1
V	0
I	1
F	1
T	0
M	0x13

Disassembly

```

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

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
  
```

Binary Representation of LSL #15:

Operand: 0000 0000 0000 0001 1001 1001 1001 1001

Result: 1100 1100 1100 1100 1000 0000 0000 0000

Press Step, or F11

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays the uVision4 IDE with the following components:

- Registers Window:** Shows the current state of ARM registers. R1 contains the value 0xCCCC8000. Other registers like R0, R2, R3, etc., contain zeros. The CPSR register shows flags: N=1, Z=0, C=0, V=0, I=1, F=1, T=0, M=0x13.
- 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 circle with the number 0 is placed above the first '0' of the last byte.

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
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):**

```

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
    style MSB fill:none,stroke:none
    style C fill:none,stroke:none

```
- 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)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**
 - R0: 0x00000000
 - R1: 0xFFFFCCCC (highlighted with a red arrow from the ASRS instruction)
 - R2: 0x00000000
 - R3: 0x00000002
 - R4: 0x00000000
 - R5: 0x00000000
 - R6: 0x00000000
 - R7: 0x00000000
 - R8: 0x00000000
 - R9: 0x00000000
 - R10: 0x00000000
 - R11: 0x00000000
 - R12: 0x00000000
 - R13 (SP): 0x00000000
 - R14 (LR): 0x00000000
 - R15 (PC): 0x00000024
 - CPSR: 0xA00000D3
 - N: 1
 - Z: 0
 - C: 1 (highlighted with a green arrow from the R1 register)
 - V: 0
 - I: 1
 - F: 1
 - T: 0
 - M: 0x13
- Disassembly View:**
 - 13: LSLS r1,r1,#15
 - 0x0000001C E1B01781 MOVs R1,R1,LSL #15
 - 14: ASRS r1,r1,#16
 - 15: 0x00000020 E1B01841 MOVs R1,R1,ASR #16
 - 16: ASRS r1,r1,r3
 - 17: 0x00000024 E1B01351 MOVs R1,R1,ASR R3
 - 18: RORS r1,r1,#4
 - 0x00000028 E1B01261 MOVs R1,R1,ROR #4
 - 19: RORS r1,r1,r3
 - 20: 0x0000002C E1B01371 MOVs R1,R1,ROR R3
 - 21: RORS r1,r1,r3
- asm_for_tutorial.asm 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
- Memory Window:**
 - Address: 0
 - 0x00000040: CC CC CC CC 00 00 00 00 00 00 00 00 00 00 00 00
- Command Window:**
 - ASSIGN BreakDisable BreakEnable BreakKill1 BreakList BreakSet
- Simulation Status:**
 - Simulation
 - t1: 0.00000000 sec
 - L16

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 0x00000024. CPSR is 0xA00000D3. The C flag is 1.
- Disassembly Window:** Shows assembly code for 'my_First_Example.s'. The code includes:


```

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
      
```
- ASR Diagram:** A diagram in the top right shows the ASR instruction. It takes an 'Operand' and shifts it 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 the assembly window:
 - 1111 1111 1111 1111 1100 1100 1100 1100
 - 1111 1111 1111 1111 1111 0011 0011 0011
- Instruction List:** A list of instructions is shown in the bottom right:


```

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
      
```
- Simulation Status:** The bottom status bar shows 'Simulation' with a time of 0.00000000 sec and a line number of L:16.

A blue cloud bubble with the text "Press Step, or F11" is overlaid on the bottom right of the assembly window.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays the uVision4 IDE interface. The **Registers** window on the left shows the current state of ARM registers, with R1 containing 0xFFFF333 and the CPSR (CPSR) showing the Carry flag (C) as 0. 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

```

The instruction **RORS r1, r1, #4** is highlighted in yellow, and its binary representation is shown as **1111 1111 1111 1111 1111 0011 0011 0011**. A green arrow points from the rightmost '1' of this binary value to the **C** (Carry) flag in the CPSR register, which is currently 0. The **asm_for_tutorial.asm** window shows the following instructions:

```

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

```

The **Project** window shows the files **asm_for_tutorial.asm** and **my_First_Example.s**. The **Command** window at the bottom shows the simulation status: **Simulation** with a time of **t1: 0.00000000 sec** and a line number of **L:18**.

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 code. Line 18: `RORS r1, r1, #4` is highlighted. The instruction is `E1B01261 MOVS R1, R1, ROR #4`.
- Source File Window:** Shows the assembly code for `my_First_Example.s`. Line 18: `RORS r1, r1, #4` is highlighted.
- 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 operation. 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 Step button in the toolbar.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot displays 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 UNDEF
  
```
- 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

Annotations in the image include a red arrow pointing from the R1 register value to the MOV instruction at address 0x0000002C, and a green arrow pointing from the R1 register value to the RORS instruction at address 0x00000019. A yellow box highlights the MOV instruction, and a green box highlights the RORS instruction. A binary representation of the R1 register value is shown as 0011 1111 1111 1111 1111 1111 0011 0011, 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:**

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 UNDEF R12,R12,R12
  
```
- 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".
- Bit Patterns:**
 - Initial value of R1: 0011 1111 1111 1111 1111 1111 0011 0011
 - Value after RORS r1,r1,r3: 1100 1111 1111 1111 1111 1111 1100 1100
- Annotation:** A blue cloud with the text "Press Step, or F11" is overlaid on the source code window.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**

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 Window:**

```

0x00000028 E1B01261 MOVN    R1,R1,ROR #4
19:          RORS    r1,r1,r3
20:
0x0000002C E1B01371 MOVN    R1,R1,ROR R3
21:          RRXS    r1,r1
0x00000030 E1B01061 MOVN    R1,R1,RRX
22:          RRXS    r1,r1
0x00000034 E1B01061 MOVN    R1,R1,RRX
23:          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 Window:**

```

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

Annotations in the image:

- A red arrow points from the highlighted instruction 21 (RRXS r1, r1) to the R1 register in the Registers window.
- A green arrow points from the highlighted value 1100 1111 1111 1111 1111 1111 1100 1100 to the CPSR register in the Registers window, specifically to the C flag which is set to 1.
- A yellow box highlights the instruction 21 and the value 1100 1111 1111 1111 1111 1111 1100 1100.
- A red circle with the number 1 is next to the highlighted 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. R1 contains 0xCFFFFFFC. R15 (PC) contains 0x00000030. CPSR contains 0xA00000D3. The 'C' (Carry) flag is set to 1.
- Disassembly Window:** Shows assembly instructions. Instruction 21 is highlighted: `RRXS r1,r1`. Instruction 22 is `RRXS r1,r1`. Instruction 23 is `RRXS r1,r1`. Instruction 24 is `RRXS r1,r1`. Instruction 25 is `END`.
- Diagram:** A diagram titled "Rotate right through carry" shows a "Register" box connected to a "Carry" box. An arrow points from the Register to the Carry, and another arrow points from the Carry back to the Register, indicating a circular shift.
- Bit Patterns:** Two bit patterns are shown in yellow boxes:
 - 1100 1111 1111 1111 1111 1111 1100 1100
 - 1110 0111 1111 1111 1111 1111 1110 0110
- Code Editor:** Shows the assembly code for `asm_for_tutorial.asm`. The code includes instructions like `ASRS r1,r1,r3`, `RORS r1,r1,#4`, and `RRXS r1,r1`.
- 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`.

A blue cloud bubble with the text "Press Step, or F11" is overlaid on the code editor, indicating the next step in the tutorial.

ARM's Data-Processing Instructions (Shift Operations)

The screenshot shows the uVision4 IDE with the following components:

- Registers Window:**
 - R0: 0x00000000
 - R1: 0xE7FFFE6 (highlighted)
 - 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 (highlighted)
 - N: 1
 - Z: 0
 - C: 0 (highlighted)
 - 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: RORS r1, r1, r3
 - 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 (highlighted)
 - 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}
- Assembly Window:**
 - 15: ASRS r1, r1, r3
 - 16: ASRS r1, r1, r3
 - 17: ASRS r1, r1, r3
 - 18: RORS r1, r1, #4
 - 19: RORS r1, r1, r3
 - 20: RORS r1, r1, r3
 - 21: RRXS r1, r1
 - 22: RRXS r1, r1 (highlighted)
 - 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

Annotations in the image include:

- A yellow box highlighting the instruction `RRXS r1, r1` at address `0x00000034` in the Disassembly window.
- A red arrow pointing from the CPSR register in the Registers window to the `RRXS r1, r1` instruction.
- A green arrow pointing from the CPSR register in the Registers window to the `END` instruction in the Assembly window.
- A blue box highlighting the value `1110 0111 1111 1111 1111 1111 1110 0110` in the Disassembly window, which is the binary representation of the CPSR 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 0x00000034. CPSR is at 0x800000D3. The 'C' (Carry) flag is 0.
- Disassembly Window:** Shows assembly instructions. Line 22 is highlighted: `RRXS r1, r1`. The instruction is `0x00000034 E1B01061 MOVVS R1, R1, RRX`.
- Diagram:** A red box highlights the 'Rotate right through carry' operation. It shows a 'Register' box connected to a 'Carry' box. An arrow points from the Register to the Carry, and another arrow points from the Carry back to the Register, indicating a circular shift.
- Bit Patterns:** Two rows of bit patterns are shown, representing the state of the register and carry before and after the shift. The first row is `1110 0111 1111 1111 1111 1111 1111 1110 0110`. The second row is `0111 0011 1111 1111 1111 1111 1111 1111 0011`. A red arrow points from the 'C' flag in the registers window to the first bit of the second row.
- Source Code Window:** Shows the assembly code for 'my_First_Example.s'. Line 22 is highlighted: `RRXS r1, r1`.
- 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`.
- Simulation Status:** Shows 'Simulation' mode with a timer of 0.00000000 sec and line 22.

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 for the current address range. Instructions include MOV, ROR, and RRX. A red arrow points from the R1 register value to the MOV instruction at address 0x0000002C.
- Source Code Window:** Shows the assembly source code for 'my_First_Example.s'. Instructions include ASRS, RORS, and RRXS. A green arrow points from the R1 register value to the RRXS instruction at address 0x00000038.
- Memory Window:** Shows the memory contents at address 0x00000040. The memory is filled with zeros.
- Command Window:** Shows the status of the simulation, including the time taken (t1: 0.00000000 sec) and the current instruction (L:23).

The binary representation of the R1 register value (0x73FFFFFF) is shown as 0111 0011 1111 1111 1111 1111 1111 0011. A yellow box highlights this binary value, and a green arrow points from it to the RRXS instruction in the source code window.

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 contains 0x73FFFFFF. R15 (PC) contains 0x00000038.
- Disassembly Panel:** Shows the assembly code for the current instruction. The instruction at address 0x00000038 is `RRXS r1, r1`, which is highlighted in yellow.
- Source Code Panel:** Shows the assembly code for the current file, `my_First_Example.s`. The instruction `RRXS r1, r1` is highlighted in green.
- Diagram:** A diagram titled "Rotate right through carry" shows a "Register" box with an arrow pointing to a "Carry" box. The diagram illustrates the rotation of the register's value to the right, with the carry bit being shifted into the register's least significant bit.
- Bit Pattern:** A bit pattern is shown in a yellow box, representing the register's value after the rotation. The bit pattern is `0111 0011 1111 1111 1111 1111 1111 0011`. The last bit, `1`, is highlighted in red.
- Callout:** A blue cloud-shaped callout with the text "Press Step, or F11" points to the assembly code.

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 (C) flag set to 1. The **Disassembly** window on the right shows the assembly code being executed. The instructions include MOV, RORS, and RRXS. A red arrow points from the R1 register to the first MOV instruction, and a green arrow points from the C flag in CPSR to the RORS instruction. 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 C flag in CPSR.

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

1

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:**

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:**

```

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
0x00000048 00000000 ANDEQ     R0,R0,R0

```
- asm_for_tutorial.asm:**

```

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

```
- Memory 1:**

Address: 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. A green arrow points from the CPSR register value (0xA00000D3) to the assembly code. A yellow box highlights the binary value 1001 1100 1111 1111 1111 1111 1111 1100, which is the value of R1 after the ASRS instruction. A blue arrow points from this binary value to the assembly code. A green arrow points from the CPSR register value to the assembly code.

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”

