Chapter 3 · Section 3.3 — Exercises (Mazidi)

Chapter 3 · Section 3.3 — Exercises (Mazidi)

> Problems are paraphrased to respect copyright. Results are shown in **32-bit hex** unless stated. When an instruction has the **S** suffix (e.g., `MOVS`), the **N/Z/C** flags are updated from the **shifter**: for `ROR #n` the carry-out is the original bit **(n−1)**; for `LSR #n` it is the original bit **(n−1)**; for `LSL #n` it is the original bit **(32−n)**; for `RRX` the carry-out is the original bit **0** and bit31 is filled with the old **C**.

---

6) Assuming `C=0`, what is the value of R1 after:

```
MOV   R1,#0x25
MOVS  R1,R1,ROR #4
```

Answer: `R1 = 0x50000002`.
Why: `0x00000025 ROR 4 = 0x50000002`. (Carry-out would be old bit3 = **0**.)

---

7) Assuming `C=0`, what are the values of R0 and C after:

```
LDR   R0,=0x3FA2
MOV   R2,#8
MOVS  R0,R0,ROR R2
```

Answer: `R0 = 0xA200003F`, `C = 1`.
Why: Rotate right by 8: bytes `00 00 3F A2 → A2 00 00 3F`; carry-out is original bit7 = **1**.

---

8) Assuming `C=0`, what are the values of R2 and C after:

```
MOV   R2,#0x55
MOVS  R2,R2,RRX
```

Answer: `R2 = 0x0000002A`, `C = 1`.
Why: `RRX` shifts right by one, **bit31←old C(0)**, and carry-out gets the original **bit0 (1)**.

---

9) Assuming `C=0`, what is the value of R1 after:

```
MOV   R1,#0xFF
MOV   R3,#5
MOVS  R1,R1,ROR R3
```

Answer: `R1 = 0xF8000007`.
Why: `0x000000FF ROR 5 = 0xF8000007` (carry-out would be original bit4 = **1**).

---

10) Give the destination register value after the instruction executes

(Assembler shorthand: `MOV Rd,#imm, ROR #n` rotates the 8-bit immediate by **n** and writes the result to `Rd`.)

- **(a)** `MOV R1,#0x88, ROR #4` → `R1 = 0x80000008`
- **(b)** `MOV R0,#0x22, ROR #22` → `R0 = 0x00008800`
- **(c)** `MOV R2,#0x77, ROR #8` → `R2 = 0x77000000`
- **(d)** `MOV R4,#0x5F, ROR #28` → `R4 = 0x000005F0`
- **(e)** `MOV R6,#0x88, ROR #22` → `R6 = 0x00022000`
- **(f)** `MOV R5,#0x8F, ROR #16` → `R5 = 0x008F0000`
- **(g)** `MOV R7,#0xF0, ROR #20` → `R7 = 0x000F0000`
- **(h)** `MOV R1,#0x33, ROR #28` → `R1 = 0x00000330`

---

11) Give the destination register value for each MVN (bitwise NOT of the rotated immediate)

- **(a)** `MVN R2,#0x01` → `R2 = 0xFFFFFFFE`
- **(b)** `MVN R2,#0xAA, ROR #20` → `R2 = 0xFFF55FFF`
- **(c)** `MVN R1,#0x55, ROR #4` → `R1 = 0xAFFFFFFA`
- **(d)** `MVN R0,#0x66, ROR #28` → `R0 = 0xFFFFF99F`
- **(e)** `MVN R2,#0x80, ROR #24` → `R2 = 0xFFFF7FFF`
- **(f)** `MVN R6,#0x10, ROR #20` → `R6 = 0xFFFEFFFF`
- **(g)** `MVN R7,#0xF0, ROR #24` → `R7 = 0xFFFF0FFF`
- **(h)** `MVN R4,#0x99, ROR #4` → `R4 = 0x6FFFFFF6`

---

12) Compute the results (and the C flag) for mixed immediate/register shifts

(a)

```
MOV   R0,#0x04
MOVS  R1,R0,LSR #2
MOVS  R3,R0,LSR R1      ; R1 holds the shift amount (=1)
```

Answer: `R1 = 0x00000001`, `R3 = 0x00000002`, `C = 0`.
Why: `0x04 >>2 = 0x01` (carry-out old bit1=0); then `0x04 >>1 = 0x02` (carry-out old bit0=0).

(b)

```
LDR   R1,=0x0000A0F2
MOV   R2,#0x3
MOVS  R3,R1,LSL R2
```

Answer: `R3 = 0x00050790`, `C = 0`.
Why: `A0F2 <<3 = 0x50790`; bits shifted out of the top were zero → `C=0`.

(c)

```
LDR   R1,=0x0000B085
MOV   R2,#3
MOVS  R4,R1,LSR R2
```

Answer: `R4 = 0x00001610`, `C = 1`.
Why: `B085 >>3 = 0x1610`; carry-out is original bit2 = **1**.

---

13) Give the register values and final C after each sequence

(a)

```
SUBS  R2,R2,R2      ; R2 = 0, sets Z=1, C=1 (no borrow)
MOV   R0,#0xAA
MOVS  R1,R0,ROR #4
```

Answer: `R2 = 0x00000000`, `R0 = 0x000000AA`, `R1 = 0xA000000A`, **`C = 1`**.
Why: Rotate right 4 → carry-out = original bit3 of `R0` = **1**.

(b)

```
MOV   R2,#0xAA, ROR #4
MOV   R0,#1
MOVS  R1,R2,ROR R0
```

Answer: `R2 = 0xA000000A`, `R1 = 0x50000005`, **`C = 0`**.
Why: ROR by 1 → carry-out = original bit0 of `R2` = **0**.

(c)

```
LDR   R1,=0x00001234
MOV   R2,#0x10, ROR #2   ; R2 low byte = 0x04 → shift amount = 4
MOVS  R1,R1,ROR R2
```

Answer: `R2 = 0x40000004`, `R1 = 0x40000123`, **`C = 0`**.
Why: `0x1234 ROR 4 = 0x40000123`; carry-out = original bit3 = **0**.

(d)

```
MOV   R0,#0xAA
; assume entry C=0 unless set earlier
MOVS  R1,R0,RRX
```

Answer: `R1 = 0x00000055`, **`C = 0`**.
Why: `RRX`: `bit31←old C(0)`, result `0xAA >>1 = 0x55`, carry-out = original bit0 (**0**).

---

14) Find the C flag after each of the following

(a)

```
MOV   R0,#0x20
MOVS  R1,R0,LSR #2
```

Answer: **`C = 0`** (original bit1 of `R0` is 0).

(b)

```
LDR   R8,=0x00000006
MOVS  R1,R8,LSR #2
```

Answer: **`C = 1`** (original bit1 of `R8` is 1).

(c)

```
LDR   R6,=0x0000001F
MOVS  R1,R6,LSL #3
```

Answer: **`C = 0`** (for `LSL #3`, carry-out is original bit29 which is 0 here).

Notes for learners

- `RRX` is a **rotate with carry**: `C_out ← old bit0`, `bit31 ← old C`.
- For register-specified shifts (`... ROR Rm`, `... LSR Rm`, etc.), only the **low byte** of `Rm` is used; the effective shift is modulo 32.
- When using the **S** suffix, remember: the flags come from the **shifter**, not the destination ALU stage.