



# CPE 221: Computer Organization

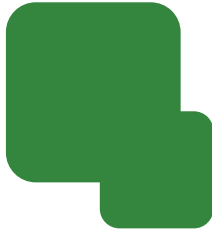
O 4 ARM Shift and Rotate Operation  
[rahul.bhadani@uah.edu](mailto:rahul.bhadani@uah.edu)

# Announcement

Homework 03 Due: Feb 14, 2025, 11:59 PM

Exam 1: February 19, 2025, Wednesday: 4:20 PM - 5:40 PM

- 1 Page, One-sided , Handwritten Sheet Allowed
- Calculator Allowed
- Close Notes, Close Books
- May ask to write ARM Programming
- Convert C Code to ARM Code
- Will cover topics: Number Systems, 2s complements Floating Points, Digital Logic, Register-Transfer Language, Transistor/Gate Basics, K-map Minimization, Truth Tables, ARM Instructions, Questions on Disassembly and Memory Map of ARM Programs



# Shift and Rotate Instructions



# Logical Shift Left (LSL)

- Syntax: `LSL <destination>, <source>, #n` (shift left by n bits).
- Shifts the bits in a register to the left by n positions.
- Vacated bits on the right are filled with zeros.
- Equivalent to multiplying by  $2^n$ .
- Example:

`LSL R0, R1, #3` @ shifts the value in R1 left by 3 bits.

- ASL and LSL are the same.

# Logical Shift Right

- Syntax: LSR <destination>, <source>, #n (shift left by n bits).
- Shifts the bits in a register to the right by n positions.
- Vacated bits on the left are filled with zeros.
- Equivalent to dividing by  $2^n$  (for Signed Numbers).
- Example:  
LSR R0, R1, #3 @ shifts the value in R1 right by 3 bits.

# Arithmetic Shift Right (ASR)

- Syntax: ASR <destination>, <source>, #n (shift left by n bits).
- Shifts the bits in a register to the right by n positions.
- Vacated bits on the left are filled with the sign bit (preserves sign for signed numbers).
- Equivalent to dividing by  $2^n$  (for Signed Numbers).
- Example:

ASR R0, R1, #3 @ shifts the value in R1 right by 3 bits.

# Rotate Right ROR

Syntax: ROR R3, R5, #21 (rotate right by `n` bits).

- Shifts the bits in a register to the right by n positions.
- Bits that are shifted out are reinserted on the left.
- Useful for circular bit manipulation.
- No equivalent ROL, but left operation can be performed with right rotation by a complementary amount.



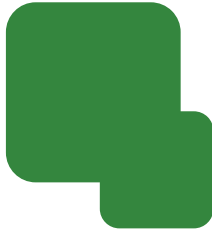
# Examples

LSR:

Let  $R8 = 0011\ 0011$ . Shift right by 3 bits. What is the result?

ASR:

Let  $R9 = 1111\ 0000$ . Shift right by 2 bits. What is the result?



Using LSL with ADD/SUB

# Combined Operations

- By combining arithmetic operations (ADD, SUB) with bit manipulation (LSL, ASR, ROR), ARM allows multiple operations to be performed in a single instruction.
- This reduces the number of instructions needed, leading to faster execution and smaller code size
- The ability to combine operations provides flexibility in handling data transformations efficiently
-

# Example 1

## ARM Assembly Code

```
@ performs both a shift operation (multiplication by 4)
@ and an addition in one step
@ R0=R1+R2<<2 or R0=R1+4*R2
ADD R0, R1, R2, LSL #2
```

Here, R2 is shifted left by 2 bits (multiplying it by 16), and the result is added to R1 and is stored in R0.

## Example 2

### ARM Assembly Code

```
@ R0=R1+R2<<R3 or R0 = R1 + (2^R3)*R2  
ADD R0, R1, R2, LSL R3
```

Here, R2 is shifted left by R3 bits (multiplying it by  $(2^{R3})$ ), and the result is added to R1 and is stored in R0.

## Example 3

### ARM Assembly Code

```
@ R0 = R1 + (R2 rotated right by 9 bits)
ADD R0, R1, R2, ROR #9
```

## Example 4: Combined LSL and ASR for Scaling

A signed integer in R1 is multiplied by 16 using LSL and then divided by 4 using ASR. What is the final value of R1 if its initial value is -16?

Explain how the sign bit affects the result.

## Example 4: Combined LSL and ASR for Scaling

### ARM Assembly Code

```
.global _start
_start:
    LDR R0, =minus_num
    LDR R1, [R0]
    @A signed integer in R1 is multiplied by 16 using LSL
    LSL R1, R1, #4
    @divided by 4 using ASR
    ASR R1, R1, #2
done: B done

.data
    minus_num: .word -16
```



## Example 4: Combined LSL and ASR for Scaling

- 16 is stored as 0xFF FF FF F0 (2's complement)
- Multiplied by 16 : 0xFF FF FF 00
- Divided by 4: 0x FF FF FF C0

## Example 5: Bit Extraction with LSR

Extract bits 8–15 from  $R4 = 0xA5A5A5A5$  into  $R5$

## Example 5: Overflow Detection with LSL

### ARM Assembly Code

```
.global _start
_start:
    LDR R0, =number
    LDR R4, [R0]
    LSR R5, R4, #8
    AND R5, R5, #0xFF
done: B done

.data
    number: .word 0xA5A5A5A5
```

# Example 6: Simulator Rotate Left with Shifts

Rotate R10 left by 5 bits without using ROR

## Example 6: Simulate Rotate Left with Shifts

### ARM Assembly Code

@ R9 = upper 5 bits shifted out

```
MOV R10, #0xFFFF77FF
```

```
MOV R9, R10, LSL #5
```

@ Combine with lower 27 bits shifted right

```
ORR R8, R9, R10, LSR #27
```

# Example 6: Simulate Rotate Left with Shifts

```
MOV R10, #0xFFFF77FF
```

R10 is

```
1111 1111 1111 1111 0111 0111 1111 1111
```

```
MOV R9, R10, LSL #5
```

```
R9 = 1111 1111 1110 1110 1111 1111 1110 0000 = FF EE FF E0
```

```
ORR R8, R9, R10, LSR #27
```

```
R8 = 0000 0000 0000 0000 0000 0000 0001 1111
```

```
1111 1111 1110 1110 1111 1111 1110 0000
```

```
R8 OR 1111 1111 1110 1110 1111 1111 1111 1111 = FF EE FF FF
```

```
If you ROR 27 to R10 (Rotate left 5) then 1111 1111 1110 1110 1111 1111 1111 1111 = FF EE FF FF
```