

Chapter 4
ARM Arithmetic and Logic Instructions
Exercises ANS

Z. Gu

Fall 2025

Barrel Shifter: Explanations

- ▶ LSL (logical shift left): **shifts left, fills zeros on the right**; C gets the last bit shifted out of bit 31. This is multiply by 2^n for non-overflowing values.
- ▶ LSR (logical shift right): **shifts right, fills zeros on the left**; C gets the last bit shifted out of bit 0. This is unsigned division by 2^n .
- ▶ ASR (arithmetic shift right): **shifts right, fills the sign bit on the left** to preserving the sign; C gets the last bit shifted out of bit 0. This is signed division by 2^n with sign extension
- ▶ ROR (rotate right): **rotates bits right with wraparound**; bits leaving bit 0 re-enter at bit 31, and C receives the bit that wrapped. This is a pure rotation without data loss.
- ▶ RRX (rotate right extended): **rotates right by one through the carry flag**, treating C as a 33rd bit; new bit 31 comes from old C, and C receives old bit 0.

Carry and Overflow Flags w/ Arithmetic Instructions

Carry flag C = 1 (Borrow flag = 0) upon an unsigned addition if the answer is wrong (true result $> 2^{n-1}$)

Carry flag C = 0 (Borrow flag = 1) upon an unsigned subtraction if the answer is wrong (true result < 0)

Overflow flag V = 1 upon a signed addition or subtraction if the answer is wrong (true result $> 2^{n-1}-1$ or true result $< -2^{n-1}$)

Overflow may occur when adding 2 operands with the same sign, or subtracting 2 operands with different signs; Overflow cannot occur when adding 2 operands with different signs or when subtracting 2 operands with the same sign.

Tip: Convert subtraction to addition with Two's complement. If two operands have same sign, and the result has opposite sign, then V = 1; else V = 0

	Unsigned Addition	Unsigned Subtraction	Signed Addition or Subtraction
Carry flag	true result $> 2^{n-1} \rightarrow$ Carry flag=1 Borrow flag=0 (Result incorrect)	true result $< 0 \rightarrow$ Carry flag=0 Borrow flag=1 (Result incorrect)	N/A
Overflow flag	N/A	N/A	true result $> 2^{n-1}-1$ or true result $< -2^{n-1}$ \rightarrow Overflow flag=1 (Result incorrect)

Bit Manipulations

- ▶ Compute 32-bit register values after each instruction
- ▶ MOV R0, #0x0ABC
- ▶ MOV RI, #0x0DEF
- ▶ AND R2, R0, RI
- ▶ ORR R3, R0, RI
- ▶ EOR R4, R0, RI
- ▶ ORN R5, R0, RI
- ▶ BIC r6, R0, RI

Bit Manipulations ANS

- ▶ MOV R0, #0x0ABC
- ▶ MOV RI, #0x0DEF
- ▶ AND R2, R0, RI
- ▶ ORR R3, R0, RI
- ▶ EOR R4, R0, RI
- ▶ ORN R5, R0, RI
- ▶ BIC r6, R0, RI
- ▶ R0 = 0x0ABC
- ▶ RI = 0x0DEF
- ▶ R2 = R0 AND RI = 0x08AC
- ▶ R3 = R0 ORR RI = 0xFFFF
- ▶ R4 = R0 EOR RI = 0x0753
- ▶ R5 = R0 ORN RI = R0 | (~RI) = 0x00000ABC | 0xFFFFF210 = 0xFFFFFABC
 - ▶ $\sim RI$ (32-bit) = 0xFFFFF210
- ▶ r6 = r0 BIC RI = R0 & (~RI) = 0x00000ABC & 0xFFFFF210 = 0x00000210

AND Rn Operand2 performs bitwise conjunction on Rn and Operand2.

ORR performs bitwise OR on Rn and Operand2.

EOR performs bitwise XOR on Rn and Operand2.

ORN computes Rn OR NOT Operand2, i.e., $Rn \mid (\sim Rm)$.

BIC computes Rn AND NOT Operand2, i.e., $Rn \& (\sim Rm)$.

(Hint: convert hex to binary, perform operation, and convert back to hex.)

Bit Manipulations

- ▶ Find the Register Value to Complement, CLEAR & SET 5th, 7th, 12th bit of the given value and also find the result: 0xDECB.

Bit Manipulations ANS

- ▶ Find the Register Value to set, clear, or toggle the 5th, 7th, 12th bit of the given value and also find the result: 0xDECB.
- ▶ ANS:
- ▶ Set/Clear/Complement
 - ▶ Set a bit: $x |= \text{mask}$, which forces those bit positions to 1 without changing others. ($|$ is OR)
 - ▶ Clear a bit: $x &= \sim\text{mask}$, which forces those bit positions to 0. ($\&$ is AND)
 - ▶ Toggle (Complement) a bit: $x ^= \text{mask}$, which flips those bit positions. ($^$ is XOR)
- ▶ Masks for 5th, 7th, 12th
 - ▶ Bit 5 → $1 << 5 = 0x20$.
 - ▶ Bit 7 → $1 << 7 = 0x80$.
 - ▶ Bit 12 → $1 << 12 = 0x1000$.
- ▶ Combined mask = $0x1000 | 0x80 | 0x20 = 0x10A0 = 0b0001\ 0000\ 1010\ 0000$
- ▶ Apply to 0xDECB
 - ▶ Set: $0xDECB | 0x10A0 = 0xDEEB$
 - ▶ Clear: $0xDECB & \sim 0x10A0 = 0xCE4B$
 - ▶ Toggle: $0xDECB ^ 0x10A0 = 0xCE6B$

Clearing a Register

- ▶ What are the different ways by which all bits in register r12 can be cleared? No other register is to be used.

Clearing a Register ANS

- ▶ What are the different ways by which all bits in register r12 can be cleared? No other register is to be used.
- ▶ Method 1: XOR with itself
 - ▶ EOR r12, r12, r12
- ▶ Method 2: AND with zero
 - ▶ AND r12, r12, #0
- ▶ Method 3: Move zero
 - ▶ MOV r12, #0
- ▶ Method 4: Subtract from itself
 - ▶ SUB r12, r12, r12
- ▶ Method 5: Clear 32 bits in r12 starting from bit 0
 - ▶ BFC r12, #0, #32
- ▶ Method 6: $r12 = r12 \text{ AND NOT } r12$ (bitwise clear)
 - ▶ BIC r12, r12, r12

Set bits

- ▶ Write an instruction that sets bits 0, 4, and 12 in register r6 and leave the remaining bits unchanged
- ▶ Write an instruction that clears bits 0, 4, and 12 in register r6 and leave the remaining bits unchanged

Set bits ANS

- ▶ Write an instruction that sets bits 0, 4, and 12 in register r6 and leave the remaining bits unchanged
- ▶ ANS:
 - ▶ ORR r6, r6, #(1<<0) | (1<<4) | (1<<12)
- ▶ Write an instruction that clears bits 0, 4, and 12 in register r6 and leave the remaining bits unchanged
- ▶ ANS:
 - ▶ BIC r6, r6, #(1<<0) | (1<<4) | (1<<12)
 - ▶ Or: AND r6, r6, #~((1<<0) | (1<<4) | (1<<12))

Add two 128-bit numbers

- ▶ Add two 128-bit numbers, assuming one number is stored in r4, r5, r6, r7 registers and the other stored in r8, r9, r10, r11. Store the result in r0, r1, r2, r3.

Add two 128-bit numbers ANS

- ▶ Add two 128-bit numbers, assuming one number is stored in r4, r5, r6, r7 registers and the other stored in r8, r9, r10, r11. Store the result in r0, r1, r2, r3.
- ▶ ANS:
 - ▶ @ Add 128-bit numbers (r4,r5,r6,r7) + (r8,r9,r10,r11) = (r0,r1,r2,r3)
 - ▶ ADDS r0, r4, r8 @ Add lowest 32 bits with carry flag update
 - ▶ ADCS r1, r5, r9 @ Add with carry
 - ▶ ADCS r2, r6, r10 @ Add with carry
 - ▶ ADC r3, r7, r11 @ Add with carry (final)
 - ▶ Or ADCS r3, r7, r11, but the flags are not used

Absolute value

- ▶ Write a program to calculate the absolute value of a number by using only two instructions (HINT: Check CMP and RSB)

Absolute value ANS

- ▶ Write a program to calculate the absolute value of a number by using only two instructions (HINT: Check CMP and RSB)
- ▶ ANS
 - ▶ `CMP r0, #0 @ Compare with zero`
 - ▶ `RSBLT r0, r0, #0 @ If less than zero, r0 = 0 - r0`

Arithmetic with Shifts

- ▶ Assuming 32-bit registers:

- ▶ Q1:

- ▶ LDR r0, =0x00000007
 - ▶ MOV r0, r0, LSL 7

- ▶ Q2:

- ▶ LDR r0, =0x00000400
 - ▶ MOV r0, r0, LSR 2

- ▶ Q3:

- ▶ LDR r0, =0xFFFFC000
 - ▶ MOV r0, r0, LSR 2

- ▶ Q4:

- ▶ LDR r0, =0xFFFFC000
 - ▶ MOV r0, r0, ASR 2

- ▶ Q5:

- ▶ LDR r0, =0x00000007
 - ▶ MOV r0, r0, ROR 2

Q1 ANS

- ▶ Q1:
 - ▶ LDR r0, =0x00000007
 - ▶ MOV r0, r0, LSL 7
- ▶ ANS:
 - ▶ Original r0 = 0000 0000 0000 0000 0000 0000 0000 0111
 - ▶ After LSL 7, r0 = 0000 0000 0000 0000 0000 0000 0011 1000 0000
= 0x00000380 (896 in decimal)
 - ▶ In decimal: $7 \times 2^7 = 7 \times 128 = 896$ (Not required for exam)

Q2 ANS

- ▶ Q2:
 - ▶ LDR r0, =0x00000400
 - ▶ MOV r0, r0, LSR 2
- ▶ ANS:
 - ▶ Original r0 = 0000 0000 0000 0000 0000 0100 0000 0000
 - ▶ After LSR 2, r0 = 0000 0000 0000 0000 0000 0001 0000 0000
= 0x00000100 (256 in decimal)
 - ▶ In decimal: $1024 \div 2^2 = 256$ (Not required for exam)

Q3 ANS

- ▶ Q3:
 - ▶ LDR r0, =0xFFFFC000
 - ▶ MOV r0, r0, LSR 2
- ▶ ANS:
 - ▶ Original r0 = 1111 1111 1111 1111 1111 1111 1100 0000 0000
 - ▶ After LSR 2, r0 = 0011 1111 1111 1111 1111 1111 0000 0000
= 0x3FFF000
 - ▶ In decimal: $4,294,951,424 \div 4 = 1,073,737,728$ (Not required for exam)

Q4 ANS

- ▶ Q3:
 - ▶ LDR r0, =0xFFFFC000
 - ▶ MOV r0, r0, ASR 2
- ▶ ANS:
 - ▶ Original r0 = 1111 1111 1111 1111 1111 1111 1100 0000 0000
 - ▶ After ASR 2, r0 = 1111 1111 1111 1111 1111 1111 1111 0000 0000
= 0xFFFFF00 (-256 in decimal)
 - ▶ In decimal: $-16384 \div 2^2 = -4096$ (Not required for exam)

Q5 ANS

- ▶ Q4:
 - ▶ LDR r0, =0x00000007
 - ▶ MOV r0, r0, ROR 2
- ▶ ANS:
 - ▶ Original r0 = 0000 0000 0000 0000 0000 0000 0000 0111
 - ▶ After ROR r0 = 1100 0000 0000 0000 0000 0000 0000 0001 = 0xC0000001

Assembly Programming

- ▶ Write ARMv7 assembly for pseudocode
 - ▶ $r1 = (r0 >> 4) \& 15$

Assembly Programming ANS

- ▶ Write ARMv7 assembly for pseudocode
 - ▶ $r1 = (r0 >> 4) \& 15$
- ▶ ANS:
 - ▶ `MOV r1, r0, LSR #4`
 - ▶ `AND r1, r1, #15`

Shift LSL ANS

- ▶ Compute register values:
 - ▶ LDR R1, =0x11223344
 - ▶ MOV R2, R1, LSL #4
 - ▶ MOV R3, R1, LSL #8
 - ▶ MOV R4, R1, LSL #16
 - ▶ MOV R5, R1, LSL #6
- ▶ ANS:
 - ▶ Pseudo-instruction LDR R1, =0x11223344 loads 32-bit constant 0x11223344 into R1.
 - ▶ LSL performs a logical left shift of Rm by the immediate count and writes the result to Rd, zero-filling low bits and discarding overflow in 32-bit registers
 - ▶ MOV R2, R1, LSL #4 performs a logical left shift of R1 by 4 bits, inserting zeros; $0x11223344 \ll 4 = 0x12233440$.
 - ▶ MOV R3, R1, LSL #8 shifts by 8 bits; $0x11223344 \ll 8 = 0x22334400$.
 - ▶ MOV R4, R1, LSL #16 shifts by 16 bits; $0x11223344 \ll 16 = 0x33440000$.
 - ▶ MOV R5, R1, LSL #6 shifts by 6 bits; $0x11223344 \ll 6 = 0x488CD100$. (convert to binary and back)

Shift LSLS ANS

- ▶ Compute register values, and NZCV flags after execution of each instruction, assuming all flags are initially 0.
 - ▶ LDR R1, =0x11223344
 - ▶ MOV R2, R1, LSLS #4
 - ▶ MOV R3, R1, LSLS #8
 - ▶ MOV R4, R1, LSLS #16
- ▶ ANS:
 - ▶ Pseudo-instruction LDR R1, =0x11223344 loads 32-bit constant 0x11223344 into R1.
 - ▶ LSL performs a logical left shift of Rm by the immediate count and writes the result to Rd, zero-filling low bits and discarding overflow in 32-bit registers
 - ▶ MOV R2, R1, LSLS #4 performs a logical left shift of R1 by 4 bits, inserting zeros;
 $0x11223344 \ll 4 = 0x12233440$.
 - ▶ NZCV = 0010 (last bit shifted out was 1, LSB of 0x1)
 - ▶ MOV R3, R1, LSLS #8 shifts by 8 bits; $0x11223344 \ll 8 = 0x22334400$.
 - ▶ NZCV = 0010 (last bit shifted out was 1, LSB of 0x11)
 - ▶ MOV R4, R1, LSLS #16 shifts by 16 bits; $0x11223344 \ll 16 = 0x33440000$.
 - ▶ NZCV = 0000 (last bit shifted out was 0, LSB of 0x122)
 - ▶ MOV R5, R1, LSLS #6 shifts by 6 bits; $0x11223344 \ll 6 = 0x488CD100$. (convert to binary and back)
 - ▶ NZCV = 0000 (last bit shifted out 0, LSB of 000100)

Shift ASR ANS

- ▶ Compute register values:
- ▶ LDR R1, =0x81223344
- ▶ MOV R2, R1, ASR #4
- ▶ MOV R3, R1, ASR #8
- ▶ MOV R4, R1, ASR #16
- ▶ ANS:
 - ▶ $R2 = R1 \text{ ASR } \#4 = 0xF8122334$ (right shift 4; top nibble becomes F due to sign extension).
 - ▶ $R3 = R1 \text{ ASR } \#8 = 0xFF812233$ (right shift 8; top byte becomes FF).
 - ▶ $R4 = R1 \text{ ASR } \#16 = 0xFFFF8122$ (right shift 16; top halfword becomes FFFF)

Shift LSR ASR ANS

- ▶ Compute register values:
 - ▶ LDR R1, =0X11223344
 - ▶ MOV R2, R1, LSR #4
 - ▶ MOV R3, R1, LSR #8
 - ▶ MOV R4, R1, LSR #16
 - ▶ MOV R5, R1, ASR #4
 - ▶ MOV R6, R1, ASR #8
 - ▶ MOV R7, R1, ASR #16
- ▶ ANS:
 - ▶ Pseudo-instruction LDR R1, =0x11223344 loads 32-bit constant 0x11223344 into R1.
 - ▶ LSR #4: R2 = 0x11223344 >> 4 = 0x01122334.
 - ▶ LSR #8: R3 = 0x11223344 >> 8 = 0x00112233.
 - ▶ LSR #16: R4 = 0x11223344 >> 16 = 0x00001122.
 - ▶ ASR #4: R5 = 0x11223344 >> 4 = 0x01122334.
 - ▶ ASR #8: R6 = 0x11223344 >> 8 = 0x00112233.
 - ▶ ASR #16: R7 = 0x11223344 >> 16 = 0x00001122.
 - ▶ ASR (arithmetic shift right) replicates the sign bit; for a positive value like 0x11223344 (bit 31 is 0), ASR behaves exactly like LSR and shifts in zeros.

Shift LSRS ASR ANS

- ▶ Compute register values, and NZCV flags after execution of each instruction, assuming all flags are initially 0.
 - ▶ LDR R1, =0x11223344
 - ▶ MOV R2, R1, LSRS #4
 - ▶ MOV R3, R1, LSRS #8
 - ▶ MOV R4, R1, LSRS #16
 - ▶ MOV R5, R1, ASRS #4
 - ▶ MOV R6, R1, ASRS #8
 - ▶ MOV R7, R1, ASRS #16
- ▶ ANS:
 - ▶ Pseudo-instruction LDR R1, =0x11223344 loads 32-bit constant 0x11223344 into R1.
 - ▶ LSR #4: R2 = 0x11223344 >> 4 = 0x01122334.
 - ▶ NZCV = 0000 (last bit shifted out was 0, MSB of 0x4)
 - ▶ LSR #8: R3 = 0x11223344 >> 8 = 0x00112233.
 - ▶ NZCV = 0000 (last bit shifted out was 0, MSB of 0x44)
 - ▶ LSR #16: R4 = 0x11223344 >> 16 = 0x00001122.
 - ▶ NZCV = 0000 (last bit shifted out was 0, MSB of 0x3344)
 - ▶ ASR #4: R5 = 0x11223344 >> 4 = 0x01122334.
 - ▶ ASR #8: R6 = 0x11223344 >> 8 = 0x00112233.
 - ▶ ASR #16: R7 = 0x11223344 >> 16 = 0x00001122.
 - ▶ NZCV = 0000 (ASR has the same flag settings as LSR, since they both shift out the same bits)
 - ▶ ASR (arithmetic shift right) replicates the sign bit; for a positive value like 0x11223344 (bit 31 is 0), ASR behaves exactly like LSR and shifts in zeros.

Multiply without MUL

- ▶ Without using MUL instruction, give instructions that multiply a register, r3 by
 - ▶ 135
 - ▶ 153
 - ▶ 255
 - ▶ 18
 - ▶ 16384

Multiply without MUL ANS

- ▶ Without using MUL instruction, give instructions that multiply a register, r3 by:
 - ▶ 135
 - ▶ 153
 - ▶ 255
 - ▶ 18
 - ▶ 1025
- ▶ ANS: Decompose $135 = 128 + 4 + 2 + 1$
 - ▶ MOV r0, r3, LSL #7 @ $128 \cdot r3$
 - ▶ ADD r0, r0, r3, LSL #2 @ $+4 \cdot r3 \rightarrow 132 \cdot r3$
 - ▶ ADD r0, r0, r3, LSL #1 @ $+2 \cdot r3 \rightarrow 134 \cdot r3$
 - ▶ ADD r0, r0, r3 @ $+1 \cdot r3 \rightarrow 135 \cdot r3$
- ▶ Decompose $153 = (8 + 1) * (16 + 1)$
 - ▶ ADD r0, r3, r3, LSL#3 @ $r0 = r3 + 8 \cdot r3 = 9 \cdot r3$
 - ▶ ADD r0, r0, r0, LSL#4 @ $r0 = 9 \cdot r3 + 16 \cdot 9 \cdot r3 = 135 \cdot r3$
- ▶ Decompose $255 = 256 - 1$
 - ▶ RSB r0, r3, r3, LSL#8 @ $r0 = 256 \cdot r3 - r3 = 255 \cdot r3$
- ▶ Decompose $1025 = 2^{10} + 1$:
 - ▶ ADD r0, r3, r3, LSL#10 @ $r0 = r3 + 1024 \cdot r3 = 1025 \cdot r3$
- ▶ Decompose $18 = (16 + 1) + 1$
 - ▶ ADD r0, r3, r3, LSL#4 @ $r0 = r3 + 16 \cdot r3 = 17 \cdot r3$
 - ▶ ADD r0, r0, r3 @ $r0 = 17 \cdot r3 + r3 = 18 \cdot r3$

Count number of ones

- ▶ Write a program to count the number of ones in a 32-bit register r0.

Count the number of ones ANS

- ▶ Write a program to count the number of ones in a 32-bit register r0.
- ▶ ANS:
 - ▶ @ Count ones in r0, result in r3
 - ▶ MOV r3,#0 @ Initialize counter
 - ▶ count_loop:
 - ▶ CMP r0,#0
 - ▶ BEQ count_end
 - ▶ AND r1,r0,#1 @ Check LSB
 - ▶ ADD r3,r3,r1 @ Add to counter
 - ▶ LSR r0,r0,#1 @ Shift right
 - ▶ B count_loop
 - ▶ count_end:

Count the number of zeros

- ▶ Based on the program that counts 1's, modify it to count the number of zeros a 32-bit register r0.

Count the number of zeros ANS

- ▶ Based on the program that counts 1's, modify it to count the number of zeros a 32-bit register r0.
- ▶ ANS: count ones, then subtract from 32
 - ▶ @ Count zeros in r0, result in r3
 - ▶ MOV r3, #0 @ Initialize counter
 - ▶ count_loop:
 - ▶ CMP r0, #0
 - ▶ BEQ count_end
 - ▶ AND r1, r0, #1 @ Check LSB
 - ▶ ADD r3, r3, r1 @ Add to counter
 - ▶ LSR r0, r0, #1 @ Shift right
 - ▶ B count_loop
 - ▶ count_end:
 - ▶ RSB r3, r3, #32 @ zeros = 32 – ones
- ▶ Or Invert all bits of r0, then count 1's
 - ▶ @ Count zeros in r0, result in r3
 - ▶ MVN r0, r0
 - ▶ MOV r3, #0 @ Initialize counter
 - ▶ count_loop:
 - ▶ CMP r0, #0
 - ▶ BEQ count_end
 - ▶ AND r1, r0, #1 @ Check LSB
 - ▶ ADD r3, r3, r1 @ Add to counter
 - ▶ LSR r0, r0, #1 @ Shift right
 - ▶ B count_loop
 - ▶ count_end:

Note that SUB r3, #32, r3 is not valid ARM syntax,
Since the middle operand cannot be an immediate.
You can load 32 into a register and subtract:

MOV r2, #32
SUB r3, r2, r3

Compute Polynomial

- ▶ Write a program that computes $6x^2 - 9x + 2$ and stores the result in register r2. Assume x is stored in register r3.

Compute Polynomial

Option A: Direct Translation

```
// r3 = x, result -> r2
MUL r0, r3, r3      @ r0 = x2
MOV r1, #6
MUL r0, r0, r1      @ r0 = 6x2
MOV r1, #9
MUL r1, r1, r3      @ r1 = 9x
SUB r2, r0, r1      @ r2 = 6x2 - 9x
ADD r2, r2, #2       @ r2 = 6x2 - 9x + 2
```

Note that MUL r0, r0, #6 is not a valid instruction.

Compute Polynomial

Option B: Minimal multiplies with a single MUL and shift-adds

```
// r3 = x, result -> r2
MUL    r0, r3, r3          // r0 = x*x
ADD    r0, r0, r0, LSL #1 // r0 = 3*x*x
ADD    r0, r0, r0          // r0 = 6*x*x
ADD    r1, r3, r3, LSL #3 // r1 = x + 8x = 9x
SUB    r2, r0, r1          // r2 = 6x^2 - 9x
ADD    r2, r2, #2          // r2 = 6x^2 - 9x + 2
```

Option C: Horner form $6x^2 - 9x + 2 = x(6x - 9) + 2$

```
// r3 = x, result -> r2
ADD    r0, r3, r3, LSL #1 // r0 = 3x
ADD    r0, r0, r0          // r0 = 6x
SUB    r0, r0, #9          // r0 = 6x - 9
MUL    r2, r3, r0          // r2 = x*(6x - 9)
ADD    r2, r2, #2          // r2 = 6x^2 - 9x + 2
```

Option D: Use MLA (multiply-accumulate) and Horner form $x(6x - 9) + 2$

```
// r3 = x, result -> r2
ADD    r0, r3, r3, LSL #1 // r0 = 3x
ADD    r0, r0, r0          // r0 = 6x
SUB    r0, r0, #9          // r0 = 6x - 9
MLA    r2, r3, r0, #2      // r2 = r3*r0 + 2
```

Flags ANDS

- ▶ What are value of r2, and NZCV flags after execution, assuming all flags are initially 0.

```
LDR r0, =0xFFFFF00  
LDR r1, =0x00000001  
ANDS r2, r1, r0, LSL #1
```

Flags ANDS ANS

- ▶ What are value of r2, and NZCV flags after execution, assuming all flags are initially 0.

```
LDR r0, =0xFFFFF00  
LDR r1, =0x00000001  
ANDS r2, r1, r0, LSL #1
```

- ▶ ANS: $r2 = 0x00000000$
- ▶ NZCV = 0110
- ▶ ANS:
- ▶ Left shift (LSL #1) moves the bits in r0 left by 1 bit. The bitwise AND is then computed: $r2 = r1 \& (r0 << 1)$.
- ▶ N (Negative flag): Set to 0 because result $r2 = 0x00000000$, which is not negative.
- ▶ Z (Zero flag): Set to 1 because the result is zero ($r2 = 0$).
- ▶ C (Carry flag): Set to 1. The last bit shifted out was 1. (ANDS does not affect C flag.)
- ▶ V (Overflow flag): Unchanged by AND operation.

Flags ADDS

- ▶ What are value of r2, and NZCV flags after execution, assuming all flags are initially 0.

```
LDR r0, =0xFFFFFFFF00  
LDR r1, =0x00000001  
ADDS r2, r1, r0, LSL #1
```

Flags ADDS ANS

- ▶ What are value of r2, and NZCV flags after execution, assuming all flags are initially 0.

```
LDR r0, =0xFFFFFFFF00  
LDR r1, =0x00000001  
ADDS r2, r1, r0, LSL #1
```

- ▶ ANS: $r2 = 0xFFFFFE01$
- ▶ NZCV = 1000
- ▶ ANS: Left shift (LSL #1) moves the bits in r0 left by 1 bit, so $r0 = 0xFFFFFE00$. ADDS is then computed: $r2 = r1 + (r0 \ll 1) = 0xFFFFFE01$
 - ▶ LSL does not update flags; ADDS updates flags
- ▶ N (Negative): Since result $r2 = 0xFFFFFE01$ when interpreted as signed 32-bit (two's complement) is negative (most significant bit is 1), $N = 1$.
- ▶ Z (Zero): Result is not zero, $Z = 0$.
- ▶ C (Carry): for ADDS, C is the carry-out from the addition (unsigned overflow). Since $0x0000_0001 + 0xFFFF_FE00 < 2^{32}$, there is no carry-out $\rightarrow C = 0$.
 - ▶ Note: the shift's carry-out was 1, but ADDS overwrites the flags; the final C is the addition carry-out (0).
- ▶ V (Overflow): Overflow is set if there is a signed overflow. Here:
 - ▶ $r1$ is positive; $r0 \ll 1$ is negative since high bit is set; 2 operands have different signs, no overflow, so $V = 0$.
 - ▶ Recall “Overflow cannot occur when adding 2 operands with different signs or when subtracting 2 operands with the same sign.”

r0	0xFFFFFFFF
r1	0x00000001
r2	0x00000003
r3	0xFFFFFFF0

Flags

- ▶ Suppose registers have the following values:
- ▶ What are value of r4, and NZCV flags after execution, assuming all flags are initially 0. (Assume each instruction runs individually, not sequentially.)
 - ▶ (a) ADD r4, r0, r2, ASRS #3
 - ▶ (b) ADDS r4, r0, r1
 - ▶ (c) LSRS r4, r0, #1
 - ▶ (d) ANDS r4, r0, r3
 - ▶ (e) CMP r2, #3

r0	0xFFFFFFFF
r1	0x00000001
r2	0x00000003
r3	0xFFFFFFF0

Flags ANS

- ▶ (a) ADD r4, r0, r2, ASRS #3
 - ▶ First, r2 ASRS #3: $0x00000003 \gg 3 = 0x00000000$ (arithmetic shift preserves sign)
 - ▶ Carry bit is 0 (last bit shifted out was 0, the first bit of 011)
 - ▶ Then: $r4 = r0 + 0 = 0xFFFFFFF + 0 = 0xFFFFFFF$
 - ▶ Result: $r4 = 0xFFFFFFF$, NZCV = 0000
 - ▶ ADD does not affect flags
- ▶ (b) ADDS r4, r0, r1
 - ▶ $r4 = 0xFFFFFFF + 0x00000001 = 0x00000000$ (truncated to 32-bit)
 - ▶ N = 0 (result bit 31 = 0, not negative)
 - ▶ Z = 1 (result is zero)
 - ▶ C = 1 (carry out from bit 31: $0xFFFFFFF + 1$ produces carry)
 - ▶ V = 0 (no signed overflow: $-1 + 1 = 0$, valid in 32-bit signed range)
 - ▶ Result: $r4 = 0x00000000$, NZCV = 0110
- ▶ (c) LSRS r4, r0, #1
 - ▶ Logical shift right with flag update: $0xFFFFFFF \gg 1 = 0x7FFFFFF$
 - ▶ N = 0 (result bit 31 = 0)
 - ▶ Z = 0 (result is not zero)
 - ▶ C = 1 (last bit shifted out was 1)
 - ▶ V = 0 (logical shifts don't affect overflow flag)
 - ▶ Result: $r4 = 0x7FFFFFF$, NZCV = 0010

r0	0xFFFFFFFF
r1	0x00000001
r2	0x00000003
r3	0xFFFFFFF0

Flags ANS

- ▶ (d) ANDS r4, r0, r3
 - ▶ Bitwise AND with flag update: $0xFFFFFFFF \& 0xFFFFFFF0 = 0xFFFFFFF0$
 - ▶ N = 1 (result MSB = 1, negative)
 - ▶ Z = 0 (result is not zero)
 - ▶ C = previous value, assumed to be 0 (C is unaffected by ANDS)
 - ▶ V = previous value, assumed to be 0 (V is unaffected by ANDS)
 - ▶ Result: r4 = 0xFFFFFFF0, NZCV = 1000
- ▶ (e) CMP r2, #3
 - ▶ Compare operation: $r2 - 3 = 3 - 3 = 0$ (result not stored, only flags updated)
 - ▶ N = 0 (subtraction result is 0, bit 31 = 0)
 - ▶ Z = 1 (subtraction result is 0)
 - ▶ C = 1 (no borrow: $3 \geq 3$, so C = 1 = not borrow)
 - ▶ V = 0 (no signed overflow in 3 - 3)
 - ▶ Result: NZCV = 0110

r0	0xFFFFFFFF
r1	0x00000001
r2	0x00000000

Flags

- ▶ Suppose registers have the following values:
- ▶ What are value of r3, and NZCV flags after execution, assuming all flags are initially 0. (Assume each instruction runs individually, not sequentially.)
 - ▶ (a) ADD r3, r0, r2
 - ▶ (b) ADDS r3, r0, r2
 - ▶ (c) SUBS r3, r0, r0
 - ▶ (d) LSL r3, r0, #1
 - ▶ (e) LSRS r3, r1, #1
 - ▶ (f) ANDS r3, r0, r2

r0	0xFFFFFFFF
r1	0x00000001
r2	0x00000000

Flags ANS

- ▶ (a) ADD r3, r0, r2
 - ▶ $r3 = 0xFFFFFFFF$
 - ▶ NZCV = 0000 (ADD without S does not update flags)
- ▶ (b) ADDS r3, r0, r2
 - ▶ $r3 = 0xFFFFFFFF$, NZCV = 1000
 - ▶ N = 1 (MSB = 1)
 - ▶ Z = 0 (nonzero)
 - ▶ C = 0 (no carry out)
 - ▶ V = 0 (no signed overflow)
- ▶ (c) SUBS r3, r0, r0
 - ▶ $r3 = 0x00000000$ ($r0 - r0 = 0$), NZCV = 0110
 - ▶ N = 0 (MSB = 0)
 - ▶ Z = 1 (result == 0)
 - ▶ C = 1 (no borrow: unsigned $r0 \geq r0$)
 - ▶ V = 0 (no signed overflow)

r0	0xFFFFFFFF
r1	0x00000001
r2	0x00000000

Flags ANS

- ▶ (d) LSL r3, r0, #1
 - ▶ $r3 = 0xFFFFFFF$ (logical left shift: $0xFFFF_FFFF \ll 1$), NZCV = 0000
 - ▶ LSL without S does not update flags
- ▶ (e) LSRS r3, r1, #1
 - ▶ $r3 = 0x00000000$ ($0x0000_0001 \gg 1 = 0$), NZCV = 0110
 - ▶ N = 0
 - ▶ Z = 1 (result zero)
 - ▶ C = 1 (bit shifted out = original bit0 = 1)
 - ▶ V = 0 (shifts do not set V)
- ▶ (f) ANDS r3, r0, r2
 - ▶ $r3 = 0x00000000$ ($0xFFFF_FFFF \& 0x0000_0000 = 0$), NZCV = 0100
 - ▶ N = 0
 - ▶ Z = 1 (result zero)
 - ▶ C = 0 (unchanged)
 - ▶ V = 0 (unchanged)

Addition & Subtraction

- ▶ For a 4-bit system, calculate the result of summation, setting C and V flags

Expression	Result	C	V	Correct? Unsigned / Signed	Decimal Unsigned (GT: Ground Truth)	Decimal Signed (GT: Ground Truth)
0100 + 0010	0110					
0100 + 0110	1010					
1100 + 1110	1010					
1100 + 1010	0110					
0100 - 0010	0010					
0100 - 0110	1110					
1100 - 0110	0110					
0100 - 1110	0110					

Addition & Subtraction ANS

- ▶ For a 4-bit system, calculate the result of summation, setting C and V flags
 - ▶ For a 4-bit system, unsigned range is [0, 15], signed range is [-8, 7]

Expression	Result	C	V	Correct? Unsigned / Signed	Decimal Unsigned (GT: Ground Truth)	Decimal Signed (GT: Ground Truth)
$0100 + 0010$	0110	0	0	Y / Y	$4 + 2 = 6$	$4 + 2 = 6$
$0100 + 0110$	1010	0	1	Y / N	$4 + 6 = 10$	$4 + 6 = -6 \text{ (GT 10)}$
$1100 + 1110$	1010	1	0	N / Y	$12 + 14 = 10 \text{ (GT 26)}$	$-4 + (-2) = -6$
$1100 + 1010$	0110	1	1	N / N	$12 + 10 = 6 \text{ (GT 22)}$	$-4 + (-6) = 6 \text{ (GT -10)}$
$0100 - 0010$	0010	1	0	Y / Y	$4 - 2 = 2$	$4 - 2 = 2$
$0100 - 0110$	1110	0	0	N / Y	$4 - 6 = 14 \text{ (GT -2)}$	$4 - 6 = -2$
$1100 - 0110$	0110	1	1	Y / N	$12 - 6 = 6$	$-4 - 6 = 6 \text{ (GT -10)}$
$0100 - 1110$	0110	0	0	N / Y	$4 - 14 = 6 \text{ (GT -10)}$	$4 - (-2) = 6$

Addition & Subtraction ANS

$C = 0$	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Carry</td></tr><tr><td>0</td><td>I</td><td>0</td><td>0</td><td></td></tr><tr><td>0</td><td>0</td><td>I</td><td>0</td><td></td></tr><tr><td>0</td><td>I</td><td>I</td><td>0</td><td></td></tr></table>	0	0	0	0	Carry	0	I	0	0		0	0	I	0		0	I	I	0	
0	0	0	0	Carry																	
0	I	0	0																		
0	0	I	0																		
0	I	I	0																		

Carry

$C = 0$	<table border="1"><tr><td>I</td><td>0</td><td>0</td><td>0</td><td>Carry</td></tr><tr><td>0</td><td>I</td><td>0</td><td>0</td><td></td></tr><tr><td>0</td><td>I</td><td>I</td><td>0</td><td></td></tr><tr><td>I</td><td>0</td><td>I</td><td>0</td><td></td></tr></table>	I	0	0	0	Carry	0	I	0	0		0	I	I	0		I	0	I	0	
I	0	0	0	Carry																	
0	I	0	0																		
0	I	I	0																		
I	0	I	0																		

$C = 0$
 $V = I$

$C = I$	<table border="1"><tr><td>I</td><td>0</td><td>0</td><td>0</td><td>Carry</td></tr><tr><td>I</td><td>I</td><td>0</td><td>0</td><td></td></tr><tr><td>I</td><td>I</td><td>I</td><td>0</td><td></td></tr><tr><td>I</td><td>0</td><td>I</td><td>0</td><td></td></tr></table>	I	0	0	0	Carry	I	I	0	0		I	I	I	0		I	0	I	0	
I	0	0	0	Carry																	
I	I	0	0																		
I	I	I	0																		
I	0	I	0																		

Carry

$C = I$	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Carry</td></tr><tr><td>I</td><td>I</td><td>0</td><td>0</td><td></td></tr><tr><td>I</td><td>0</td><td>I</td><td>0</td><td></td></tr><tr><td>0</td><td>I</td><td>I</td><td>0</td><td></td></tr></table>	0	0	0	0	Carry	I	I	0	0		I	0	I	0		0	I	I	0	
0	0	0	0	Carry																	
I	I	0	0																		
I	0	I	0																		
0	I	I	0																		

$C = I$
 $V = I$

Addition & Subtraction ANS

$$0100 - 0010 = 0100 + 1110$$

I	0	0	0
0	I	0	0
I	I	I	0
C = I	I I I 0		
V = 0	0	0	I 0

Carry

$$0100 - 0110 = 0100 + 1010$$

0	0	0	0
0	I	0	0
I	0	I	0
C = 0	I 0 I 0		
V = 0	I	I	I 0

Carry

$$1100 - 0110 = 1100 + 1010$$

0	0	0	0
I	I	0	0
I	0	I	0
C = I	I 0 I 0		
V = I	0	I	I 0

Carry

$$0100 - 1110 = 0100 + 0010$$

0	0	0	0
0	I	0	0
0	0	I	0
C = 0	0 0 I 0		
V = 0	0	I	I 0

Carry