UNDERSTANDING NZCV FLAGS AND CONDITIONAL CODES

NAMIG PLANOV

OBJECTIVE OF THIS PRESENTATION

EXPLAIN WHAT NZCV FLAGS ARE AND HOW THEY WORK IN A SIGNED BIT.

SHOW HOW TO INTERPRET THE NZCV PATTERNS FOR EACH CONDITION CODE.

USE 8-BIT EXAMPLES FOR EASIER UNDERSTANDING.

NZCV Pattern

Conditional loops and branching in ARM64 are controlled by certain condition flags, also called ALU flags, stored in the global condition flag register called NZCV. These flags work as follows:

- N (Negative) – Set to 1 if the result of a signed operation is negative; otherwise, it’s 0.
- Z (Zero) – Set to 1 if the result is zero; otherwise, it’s 0.
- C (Carry) – Set to 1 when an addition overflows or when a subtraction does not need a borrow. During shift operations, it stores the last bit shifted out. Otherwise, it’s 0.
- V (Overflow) – Set to 1 when a signed addition or subtraction causes an overflow; otherwise, it’s 0.

Only instructions specifically designed to update them do, which is usually indicated by an S at the end of the instruction (for example, ADDS updates the flags, while ADD does not).

BRANCHING

BRANCHING REFERS TO CHANGING THE NORMAL SEQUENCE OF INSTRUCTION EXECUTION.
NORMALLY, THE CPU EXECUTES INSTRUCTIONS ONE AFTER ANOTHER, BUT A BRANCH CHANGES THIS FLOW — FOR EXAMPLE, BY JUMPING TO ANOTHER PART OF THE PROGRAM.

THERE ARE **TWO MAIN TYPES OF BRANCHING**:

UNCONDITIONAL BRANCHING
THIS TYPE OF BRANCH *ALWAYS* TRANSFERS CONTROL TO ANOTHER INSTRUCTION, NO MATTER WHAT.
IT DOESN'T DEPEND ON ANY CONDITION.

CONDITIONAL BRANCHING
THIS TYPE OF BRANCH HAPPENS **ONLY IF A SPECIFIC CONDITION IS TRUE.**
THE CPU CHECKS CERTAIN **STATUS FLAGS** (LIKE ZERO, NEGATIVE, CARRY, OR OVERFLOW) THAT ARE SET BY PREVIOUS OPERATIONS.

CONDITION CODES

GE – GREATER THAN OR EQUAL -- $\overline{N \oplus V}$

```
 MOV      R1, #50                    @ 00110010
 MOV      R2, #20                    @ 00010100
 SUBS    R0, R1, R2                 @ R0 = R1 - R2, UPDATES NZCV FLAGS
 BGE     GREATER_EQUAL              @ BRANCH IF R1 >= R2 (SIGNED)
@ IF BRANCH NOT TAKEN
MOV      R3, #0                     @ R1 < R2
B        DONE
GREATER_EQUAL:
MOV      R3, #1                     @ R1 >= R2
DONE:
B        DONE                       @ INFINITE LOOP SO FLAGS CAN BE OBSERVED
```

LT -- LESS THAN -- $N \oplus V$

```
    MOV     R1, #-10       @ 8-BIT VALUE: 0XF6 IN 2'S COMPLEMENT
    MOV     R2, #20        @ 8-BIT VALUE: 0X14
    SUBS    R0, R1, R2     @ R0 = R1 - R2, UPDATES NZCV FLAGS
    BLT     LESS_THAN      @ BRANCH IF R1 < R2 (N ⊕ V = 1)
    @ IF BRANCH NOT TAKEN
    MOV     R3, #0         @ R1 >= R2
    B       DONE
LESS_THAN:
    MOV     R3, #1         @ R1 < R2
DONE:
    B       DONE           @ STOP PROGRAM (INFINITE LOOP)
    END
```

GREATER THAN – GT – $\overline{Z}\,(\overline{N \oplus V})$

```
MOV     R1, #50         @ 00110010
MOV     R2, #20         @ 00010100
SUBS    R0, R1, R2      @ R0 = R1 - R2, UPDATES NZCV FLAGS
BGT     GREATER_THAN    @ BRANCH IF R1 > R2 (SIGNED)
@ BRANCH NOT TAKEN
MOV     R3, #0          @ R1 <= R2
B   DONE
GREATER_THAN:
MOV     R3, #1          @ R1 > R2
DONE:
B   DONE        @ INFINITE LOOP SO YOU CAN OBSERVE
```

LESS THAN OR EQUAL – LE – Z OR (N ⊕ V)

```
MOV     R1, #-40      @ 11011000  (−40)

MOV     R2, #-10      @ 11110110  (−10)

SUBS    R0, R1, R2    @ R0 = R1 - R2, UPDATES NZCV

BLE     LESS_EQUAL    @ BRANCH IF R1 <= R2 (SIGNED)

@ BRANCH NOT TAKEN

MOV     R3, #0        @ R1 > R2

B       DONE

LESS_EQUAL:

MOV     R3, #1        @ R1 <= R2

DONE:

B       DONE          @ LOOP FOREVER TO INSPECT FLAGS
```