ARCHITETTURA DEGLI ELABORATORI

A.A. 2020-2021

Università di Napoli Federico II
Corso di Laurea in Informatica

Docenti
Proff. Luigi Sauro gruppo 1 (A-G)
Silvia Rossi gruppo 2 (H-Z)

ARCHITETTURA ARM

Shift Instructions

- `LSL`: logical shift left
- `LSR`: logical shift right
- `ASR`: arithmetic shift right
- `ROR`: rotate right
- Ampiezza è un immediato o un registro

Shift Instructions

- `LSL`: logical shift left

  Example: `LSL R0, R7, #5  ; R0=R7 << 5`

- `LSR`: logical shift right

- `ASR`: arithmetic shift right

- `ROR`: rotate right

Shift Instructions

- `LSL`: logical shift left

  Example: `LSL R0, R7, #5  ; R0=R7 << 5`

- `LSR`: logical shift right

  Example: `LSR R3, R2, #31 ; R3=R2 >> 31`

- `ASR`: arithmetic shift right

- `ROR`: rotate right

Shift Instructions

- `LSL`: logical shift left

  Example: `LSL R0, R7, #5  ; R0=R7 << 5`

- `LSR`: logical shift right

  Example: `LSR R3, R2, #31 ; R3=R2 >> 31`

- `ASR`: arithmetic shift right

  Example: `ASR R9, R11, R4 ; R9=R11 >>> R4`$_{7:0}$

- `ROR`: rotate right

Shift Instructions

- `LSL`: logical shift left

  Example: `LSL R0, R7, #5 ; R0=R7 << 5`

- `LSR`: logical shift right

  Example: `LSR R3, R2, #31 ; R3=R2 >> 31`

- `ASR`: arithmetic shift right

  Example: `ASR R9, R11, R4 ; R9=R11 >>> R4`$_{7:0}$

- `ROR`: rotate right

  Example: `ROR R8, R1, #3 ; R8=R1 ROR 3`

Shift Instructions: Example 1

- **Immediate** shift amount (5-bit immediate)
- Shift amount: 0-31

	Source register			
R5	1111 1111	0001 1100	0001 0000	1110 0111

Assembly Code		Result			
`LSL R0, R5, #7`	R0	1000 1110	0000 1000	0111 0011	1000 0000
`LSR R1, R5, #17`	R1	0000 0000	0000 0000	0111 1111	1000 1110
`ASR R2, R5, #3`	R2	1111 1111	1110 0011	1000 0010	0001 1100
`ROR R3, R5, #21`	R3	1110 0000	1000 0111	0011 1111	1111 1000

Shift Instructions: Example 2

- **Register** shift amount (uses low 8 bits of register)
- Shift amount: 0-255

	Source registers			
R8	0000 1000	0001 1100	0001 0110	1110 0111
R6	0000 0000	0000 0000	0000 0000	0001 0100

Assembly code		Result			
`LSL R4, R8, R6`	R4	0110 1110	0111 0000	0000 0000	0000 0000
`ROR R5, R8, R6`	R5	1100 0001	0110 1110	0111 0000	1000 0001

Multiplication

- **`MUL`:** 32 × 32 multiplication, 32-bit result
- **`UMULL`:** Unsigned multiply long: 32 × 32 multiplication, 64-bit result
- **`SMULL`:** Signed multiply long: 32 × 32 multiplication, 64-bit result

Multiplication

- **MUL:** 32 × 32 multiplication, 32-bit result

  ```
  MUL R1, R2, R3
  ```

  Result: $R1 = (R2 \times R3)_{31:0}$

- **UMULL:** Unsigned multiply long: 32 × 32 multiplication, 64-bit result

- **SMULL:** Signed multiply long: 32 × 32 multiplication, 64-bit result

Multiplication

- **MUL:** 32 × 32 multiplication, 32-bit result

  `MUL R1, R2, R3`

  Result: `R1 = (R2 x R3)`$_{31:0}$

- **UMULL:** Unsigned multiply long: 32 × 32 multiplication, 64-bit result

  `UMULL R1, R2, R3, R4`

  Result: `{R1,R2} = R3 x R4` (`R3`, `R4` unsigned)

- **SMULL:** Signed multiply long: 32 × 32 multiplication, 64-bit result

Multiplication

- **MUL:** 32 × 32 multiplication, 32-bit result

  `MUL R1, R2, R3`

  Result: `R1 = (R2 x R3)`$_{31:0}$

- **UMULL:** Unsigned multiply long: 32 × 32 multiplication, 64-bit result

  `UMULL R1, R2, R3, R4`

  Result: `{R1,R4} = R2 x R3` (`R2`, `R3` unsigned)

- **SMULL:** Signed multiply long: 32 × 32 multiplication, 64-bit result

  `SMULL R1, R2, R3, R4`

  Result: `{R1,R2} = R3 x R4` (`R3`, `R4` signed)

Programming Building Blocks

- **Data-processing Instructions**
- **Conditional Execution**
- **Branches**
- **High-level Constructs:**
  - if/else statements
  - for loops
  - while loops
  - arrays
  - function calls

Conditional Execution

Don’t always want to execute code sequentially

- For example:
  - if/else statements, while loops, etc.: only want to execute code *if* a condition is true
  - branching: jump to another portion of code *if* a condition is true

Conditional Execution

Don't always want to execute code sequentially

- For example:
  - if/else statements, while loops, etc.: only want to execute code *if* a condition is true
  - branching: jump to another portion of code *if* a condition is true
- ARM includes **condition flags** that can be:
  - set by an instruction
  - used to conditionally execute an instruction

ARM Condition Flags

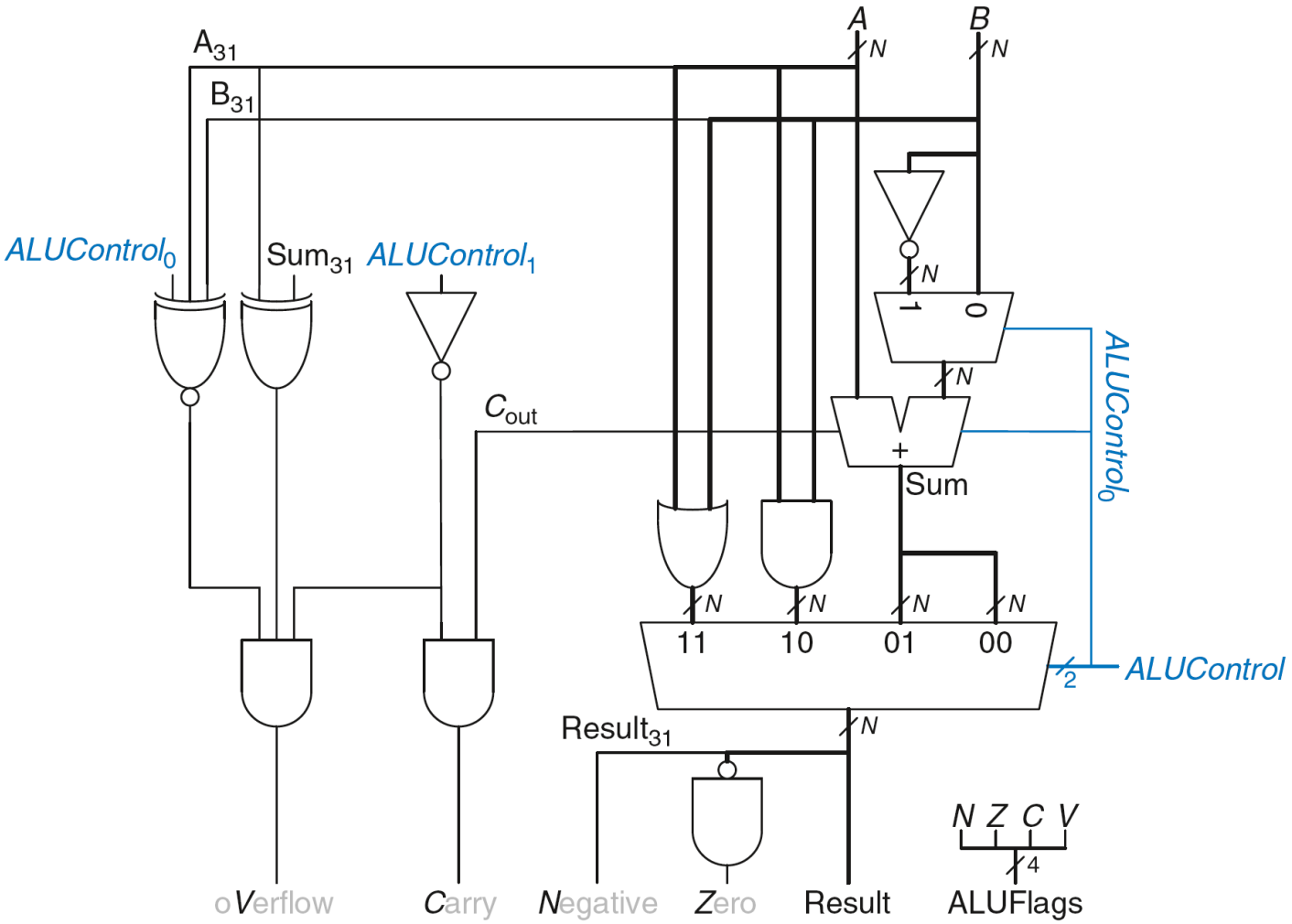
Flag	Name	Description
N	**N**egative	Instruction result is negative
Z	**Z**ero	Instruction results in zero
C	**C**arry	Instruction causes an unsigned carry out
V	o**V**erflow	Instruction causes an overflow

ARM Condition Flags

Flag	Name	Description
N	**N**egative	Instruction result is negative
Z	**Z**ero	Instruction results in zero
C	**C**arry	Instruction causes an unsigned carry out
V	o**V**erflow	Instruction causes an overflow

- Set by ALU (recall from Chapter 5)
- Held in *Current Program Status Register* (*CPSR*)

Review: ARM ALU



Setting the Condition Flags: *NZCV*

- **Method 1:** Compare instruction: `CMP`

  Example: `CMP R5, R6`

  - Performs: R5-R6
  - Does not save result
  - Sets flags

Setting the Condition Flags: *NZCV*

- **Method 1:** Compare instruction: `CMP`

  Example: `CMP R5, R6`

  - Performs: R5-R6
  - Does not save result
  - Sets flags. If result:
    - Is 0, *Z*=1
    - Is negative, *N*=1
    - Causes a carry out, *C*=1
    - Causes a signed overflow, *V*=1