Prof. Mariacarla Staffa
a.a. 2022/2023

Laboratorio di Architettura Degli Elaboratori

Architettura ARM

Programming

- **High-level languages:**
  - e.g., C, Java, Python
  - Written at higher level of abstraction

Ada Lovelace, 1815-1852

- British mathematician
- Wrote the first computer program
- Her program calculated the Bernoulli numbers on Charles Babbage's Analytical Engine
- She was a child of the poet Lord Byron

Programming Building Blocks

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

Programming Building Blocks

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

Data-processing Instructions

- Logical operations
- Shifts / rotate
- Multiplication

Logical Instructions

- AND
- ORR **(OR)**
- EOR **(XOR)**
- BIC **(Bit Clear)** → BIC R1 R2 R3 => R1 = R2 AND not(R3)
  - Operano tutte bit a bit
  - Prima sorgente un registro, seconda registro/immediato

- MVN **(MoVe and NOT)**

Logical Instructions: Examples

Source registers

R1	0100 0110	1010 0001	1111 0001	1011 0111
R2	1111 1111	1111 1111	0000 0000	0000 0000

Assembly code

```
AND  R3, R1, R2
ORR  R4, R1, R2
EOR  R5, R1, R2
BIC  R6, R1, R2
MVN  R7, R2
```

Result

R3	0100 0110	1010 0001	0000 0000	0000 0000
R4	1111 1111	1111 1111	1111 0001	1011 0111
R5	1011 1001	0101 1110	1111 0001	1011 0111
R6	0000 0000	0000 0000	1111 0001	1011 0111
R7	0000 0000	0000 0000	1111 1111	1111 1111

Logical Instructions: Uses

- AND or BIC: useful for **masking** bits

Logical Instructions: Uses

- AND or BIC: useful for **masking** bits
- **Example:** Masking all but the least significant byte of a value
  - 0xF234012F AND 0x000000FF = 0x0000002F
  - 0xF234012F BIC 0xFFFFFF00 = 0x0000002F

Logical Instructions: Uses

- AND or BIC: useful for **masking** bits
- **Example:** Masking all but the least significant byte of a value
  - 0xF234012F AND 0x000000FF = 0x0000002F
  - 0xF234012F BIC 0xFFFFFF00 = 0x0000002F

- ORR: useful for **combining** bit fields

Logical Instructions: Uses

- AND or BIC: useful for **masking** bits
  - **Example:** Masking all but the least significant byte of a value
    - 0xF234012F AND 0x000000FF = 0x0000002F
    - 0xF234012F BIC 0xFFFFFF00 = 0x0000002F

- ORR: useful for **combining** bit fields
  - **Example:** Combine 0xF2340000 with 0x000012BC:
    - 0xF2340000 ORR 0x000012BC = 0xF23412BC

Generating Constants

- Generare costanti usando MOV e ORR:

C Code

```
int a = 0x7EDC8765;
```

ARM Assembly Code

```
; R0 = a
MOV R0, #0x7E000000
ORR R0, R0, #0xDC0000
ORR R0, R0, #0x8700
ORR R0, R0, #0x65
```

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, #4 ; R9 = R11 >>> 4

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

```
LSL R4, R8, R6
ROR R5, R8, R6
```

Result

R4	0110 1110	0111 0000	0000 0000	0000 0000
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: $R1 = (R2 \times R3)_{31:0}$

- **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,R2} = R3 x R4` (`R3`, `R4` 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)