

# 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**

# 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

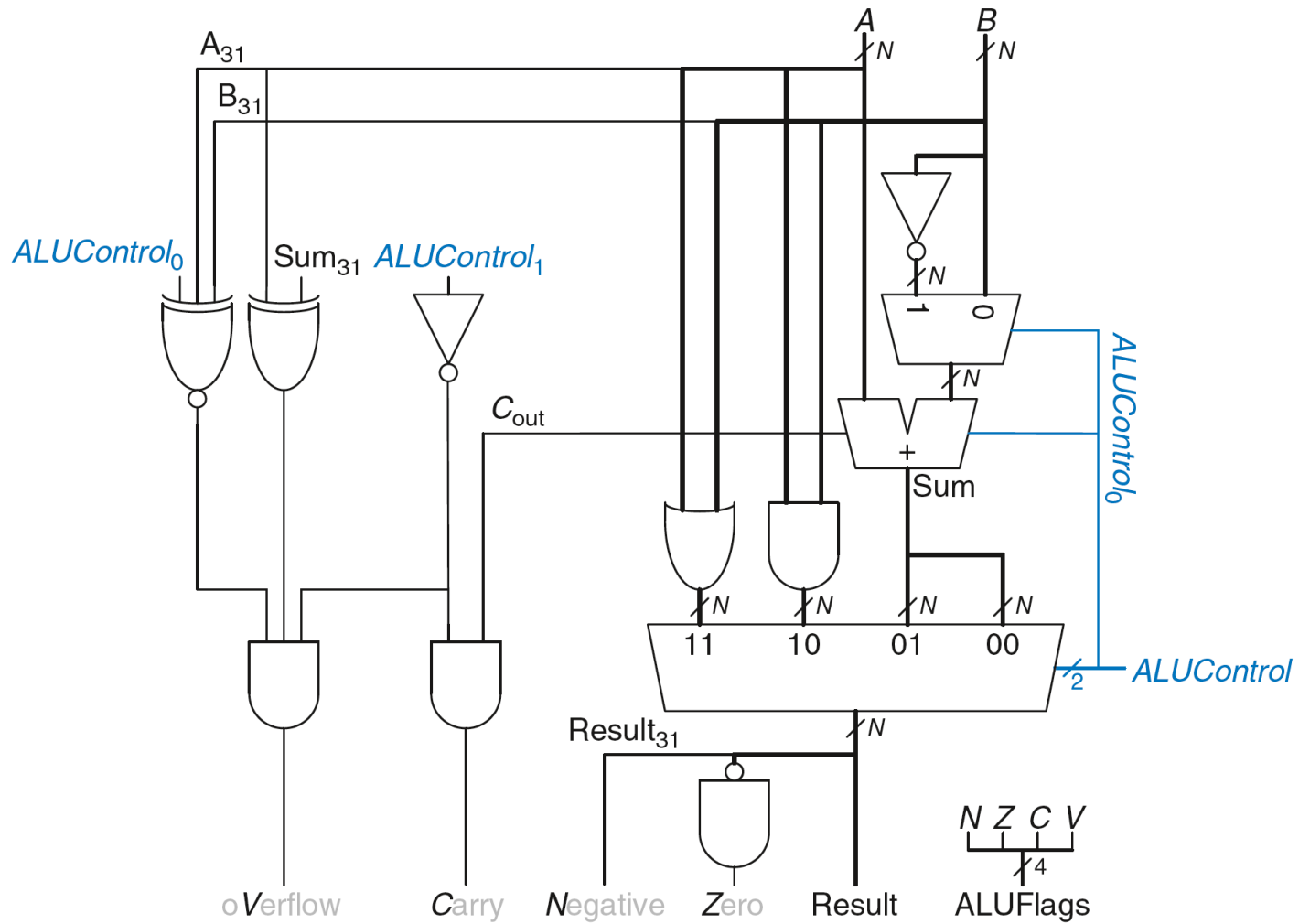
Flag	Name	Description
<i>N</i>	<b>N</b> egative	Instruction result is negative
<i>Z</i>	<b>Z</b> ero	Instruction results in zero
<i>C</i>	<b>C</b> arry	Instruction causes an unsigned carry out
<i>V</i>	<b>oV</b> erflow	Instruction causes an overflow

# ARM Condition Flags

Flag	Name	Description
<i>N</i>	<b>N</b> egative	Instruction result is negative
<i>Z</i>	<b>Z</b> ero	Instruction results in zero
<i>C</i>	<b>C</b> arry	Instruction causes an unsigned carry out
<i>V</i>	<b>oV</b> 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,  $V=1$

# Setting the Condition Flags: *NZCV*

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

**Example:** `CMP R5, R6`

- Performs: `R5-R6`
  - Sets flags: If result is 0 (`Z=1`), negative (`N=1`), etc.
  - Does not save result
- **Method 2:** Append instruction mnemonic with **S**

# Setting the Condition Flags: *NZCV*

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

**Example:** `CMP R5, R6`

- Performs: `R5-R6`
- Sets flags: If result is 0 (`Z=1`), negative (`N=1`), etc.
- Does not save result

- **Method 2:** Append instruction mnemonic with **S**

**Example:** `ADDS R1, R2, R3`

- Performs: `R2 + R3`
- Sets flags: If result is 0 (`Z=1`), negative (`N=1`), etc.
- Saves result in `R1`

# Condition Mnemonics

- Instruction may be *conditionally executed* based on the condition flags
- Condition of execution is encoded as a *condition mnemonic* appended to the instruction mnemonic

**Example:**      `CMP      R1, R2`  
                 `SUBNE R3, R5, R8`

- **NE:** condition mnemonic
- SUB will only execute if  $R1 \neq R2$   
(i.e.,  $Z = 0$ )

# Condition Mnemonics

<i>cond</i>	Mnemonic	Name	CondEx
0000	EQ	Equal	$Z$
0001	NE	Not equal	$\bar{Z}$
0010	CS / HS	Carry set / Unsigned higher or same	$C$
0011	CC / LO	Carry clear / Unsigned lower	$\bar{C}$
0100	MI	Minus / Negative	$N$
0101	PL	Plus / Positive of zero	$\bar{N}$
0110	VS	Overflow / Overflow set	$V$
0111	VC	No overflow / Overflow clear	$\bar{V}$
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z \text{ OR } \bar{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\bar{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	$Z \text{ OR } (N \oplus V)$
1110	AL (or none)	Always / unconditional	ignored

# Conditional Execution

## Example:

```
CMP    R5, R9           ; performs R5-R9
                        ; sets condition flags

SUBEQ  R1, R2, R3        ; executes if R5==R9 (Z=1)
ORRMI  R4, R0, R9        ; executes if R5-R9 is
                        ; negative (N=1)
```

# Conditional Execution

## Example:

```
CMP    R5, R9           ; performs R5-R9
                        ; sets condition flags

SUBEQ  R1, R2, R3        ; executes if R5==R9 (Z=1)
ORRMI  R4, R0, R9        ; executes if R5-R9 is
                        ; negative (N=1)
```

**Suppose R5 = 17, R9 = 23:**

CMP performs:  $17 - 23 = -6$  (Sets flags:  $N=1$ ,  $Z=0$ ,  $C=0$ ,  $V=0$ )

SUBEQ **doesn't execute** (they aren't equal:  $Z=0$ )

ORRMI **executes** because the result was negative ( $N=1$ )



# Programming Building Blocks

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

# Branching

- Branches enable out of sequence instruction execution
- Types of branches:
  - **Branch (B)**
    - branches to another instruction
  - **Branch and link (BL)**
    - discussed later
- Both can be conditional or unconditional

# The Stored Program

## Assembly code

MOV R1, #100

MOV R2, #69

CMP R1, R2

STRHS R3, [R1, #0x24]

## Machine code

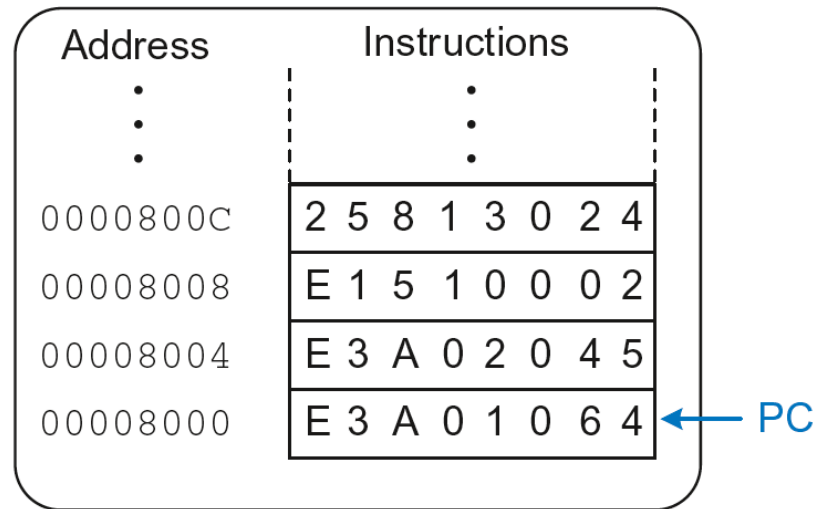
0xE3A01064

0xE3A02045

0xE1510002

0x25813024

## Stored program



## Main memory

# Unconditional Branching (B)

## ARM assembly

```
MOV R2, #17           ; R2 = 17
B    TARGET          ; branch to target
ORR R1, R1, #0x4       ; not executed
```

TARGET

```
    SUB R1, R1, #78      ; R1 = R1 - 78
```

# Unconditional Branching (B)

## ARM assembly

```
MOV R2, #17          ; R2 = 17
B    TARGET        ; branch to target
ORR R1, R1, #0x4      ; not executed
```

TARGET

```
SUB R1, R1, #78      ; R1 = R1 + 78
```

**Labels** (like TARGET) indicate instruction location.  
Labels can't be reserved words (like ADD, ORR, etc.)

# The Branch Not Taken

## ARM Assembly

```
MOV    R0, #4           ; R0 = 4
ADD    R1, R0, R0       ; R1 = R0+R0 = 8
CMP    R0, R1           ; sets flags with R0-R1
BEQ    THERE          ; branch not taken (Z=0)
ORR    R1, R1, #1       ; R1 = R1 OR R1 = 9
THERE
ADD    R1, R1, 78       ; R1 = R1 + 78 = 87
```

# Programming Building Blocks

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

# if Statement

## C Code

```
if (i == j)
    f = g + h;
```

```
f = f - i;
```



# if Statement

## C Code

## ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j
```

```
if (i == j)      CMP R3, R4      ; set flags with R3-R4
    f = g + h;    BNE L1         ; if i!=j, skip if block
                  ADD R0, R1, R2  ; f = g + h

                  L1
f = f - i;        SUB R0, R0, R3  ; f = f - i
```

**Nota:** il codice assembly effettua il test opposto ( $i \neq j$ ) rispetto a quello di alto livello ( $i == j$ )

# Istruzioni condizionali

## C Code

```
if (i == j)
    f = g + h;
f = f - i;
```

## ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j
```

```
CMP    R3, R4        ; set flags with R3-R4
ADDEQ  R0, R1, R2     ; if (i==j) f = g + h
SUB    R0, R0, R3     ; f = f - i
```

# if Statement: Alternate Code

Codice alternativo per piccoli blocchi di codice:

## Original

```
CMP R3, R4
BNE L1
ADD R0, R1, R2
L1
SUB R0, R0, R2
```

## Alternate Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j
```

```
CMP    R3, R4          ; set flags with R3-R4
ADDEQ  R0, R1, R2      ; if (i==j) f = g + h
SUB     R0, R0, R2      ; f = f - i
```

# if Statement: Alternate Code

## Original

```
CMP R3, R4
BNE L1
ADD R0, R1, R2
L1
SUB R0, R0, R2
```

## Alternate Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j
```

```
CMP    R3, R4          ; set flags with R3-R4
ADDEQ  R0, R1, R2      ; if (i==j) f = g + h
SUB     R0, R0, R2     ; f = f - i
```

Useful for **short** conditional blocks of code

# if/else Statement

## C Code

```
if (i == j)
    f = g + h;
```

```
else
    f = f - i;
```

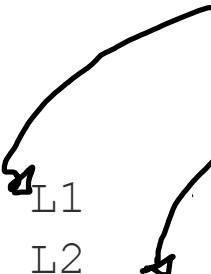
## ARM Assembly Code

# if/else Statement

## C Code      ARM Assembly Code

;R0=f, R1=g, R2=h, R3=i, R4=j

if (i == j)		CMP R3, R4	; set flags with R3-R4
f = g + h;		<u>BNE</u> L1	; if i!=j, skip if block
		ADD R0, R1, R2	; f = g + h
		B L2	; branch past else block
else	L1	SUB R0, R0, R3	; f = f - i
f = f - i;	L2	...	



# if/else Statement: Alternate Code

## C Code

```
if (i == j)
    f = g + h;
else
    f = f - i;
```

## ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j

CMP    R3, R4        ; set flags with R3-R4
ADDEQ  R0, R1, R2     ; if (i==j) f = g + h
SUBNE  R0, R0, R3     ; else f = f - i
```

# if/else Statement: Alternate Code

Codice alternativo per piccoli blocchi di codice:

## Original

```
CMP R3, R4
BNE L1
ADD R0, R1, R2
B    L2
L1
  SUB R0, R0, R2
L2
```

## Alternate Assembly Code

;R0=f, R1=g, R2=h, R3=i, R4=j

```
CMP    R3, R4          ; set flags with R3-R4
ADDEQ  R0, R1, R2      ; if (i==j) f = g + h
SUBNE  R0, R0, R2      ; else f = f - i
```



# while Loops

## C Code

```
// determines the power
// of x such that 2x = 128
int pow = 1;
int x    = 0;

while (pow != 128) {

    pow = pow * 2;
    x = x + 1;
}
```

## ARM Assembly Code

```
MOV R0, #1
MOV R1, #0
L2    CMP R0, #128
      BEQ L1
      MUL R0, R0, #2
      ADD R1, R1, #1
      B L2
L1
```

# while Loops

## C Code

```
// determines the power  
// of x such that 2x = 128  
int pow = 1;  
int x    = 0;
```

```
while (pow != 128) {  
  
    pow = pow * 2;  
    x = x + 1;  
}
```

## ARM Assembly Code

```
; R0 = pow, R1 = x  
MOV    R0, #1          ; pow = 1  
MOV    R1, #0          ; x = 0  
  
WHILE  
    CMP R0, #128        ; R0-128  
    BEQ DONE            ; if (pow==128)  
                        ; exit loop  
  
    LSL R0, R0, #1       ; pow=pow*2  
    ADD R1, R1, #1       ; x=x+1  
    B   WHILE            ; repeat loop  
  
DONE
```

**Il codice assembly verifica la condizione opposta (`pow == 128`) a quella del C (`pow != 128`).**