

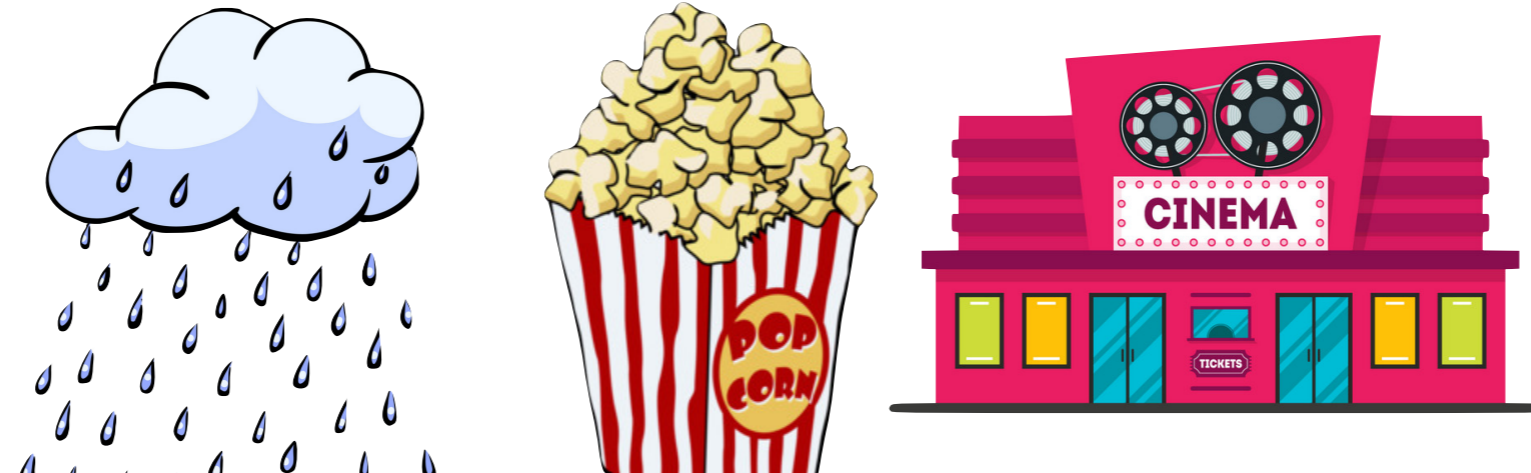
# Introduction à l'assembleur ARM: instructions conditionnelles



# Instruction logique: ET

## ET logique

« S'il pleut ET qu'il y a du popcorn, j'irai au cinéma »

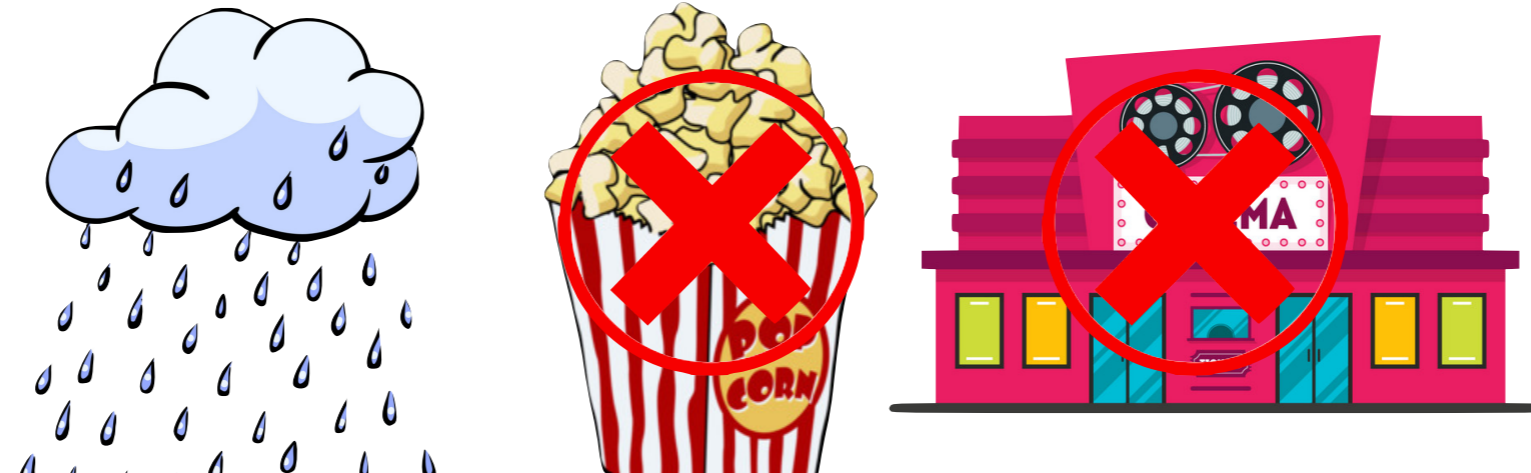


$p$	$q$	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

# Instruction logique: ET

## ET logique

« S'il pleut ET qu'il y a du popcorn, j'irai au cinéma »

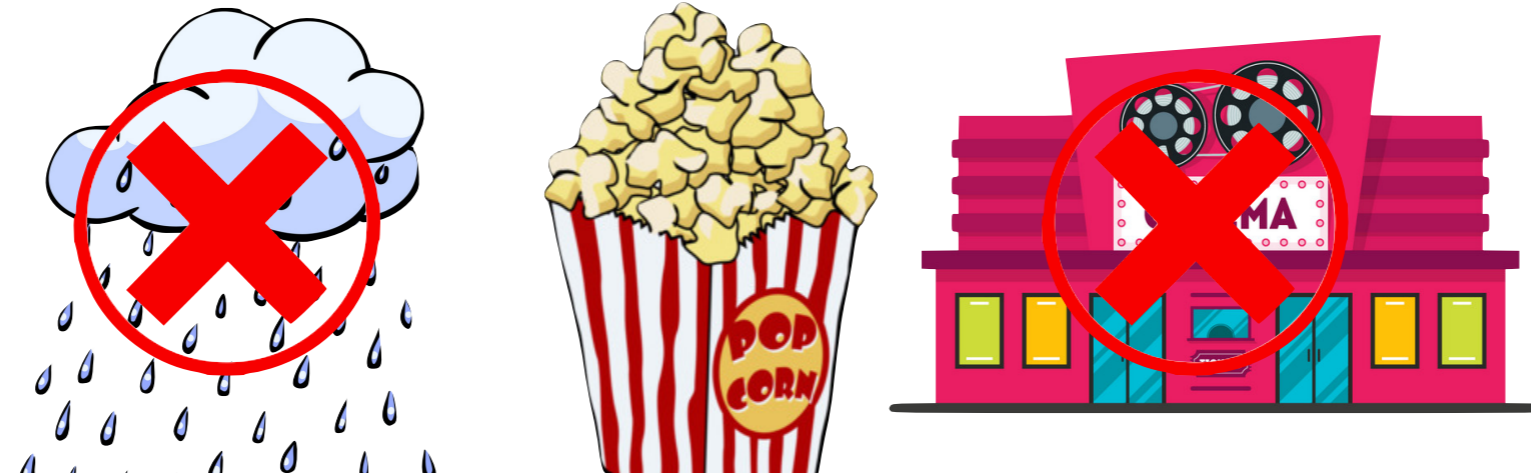


$p$	$q$	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

# Instruction logique: ET

## ET logique

« S'il pleut ET qu'il y a du popcorn, j'irai au cinéma »



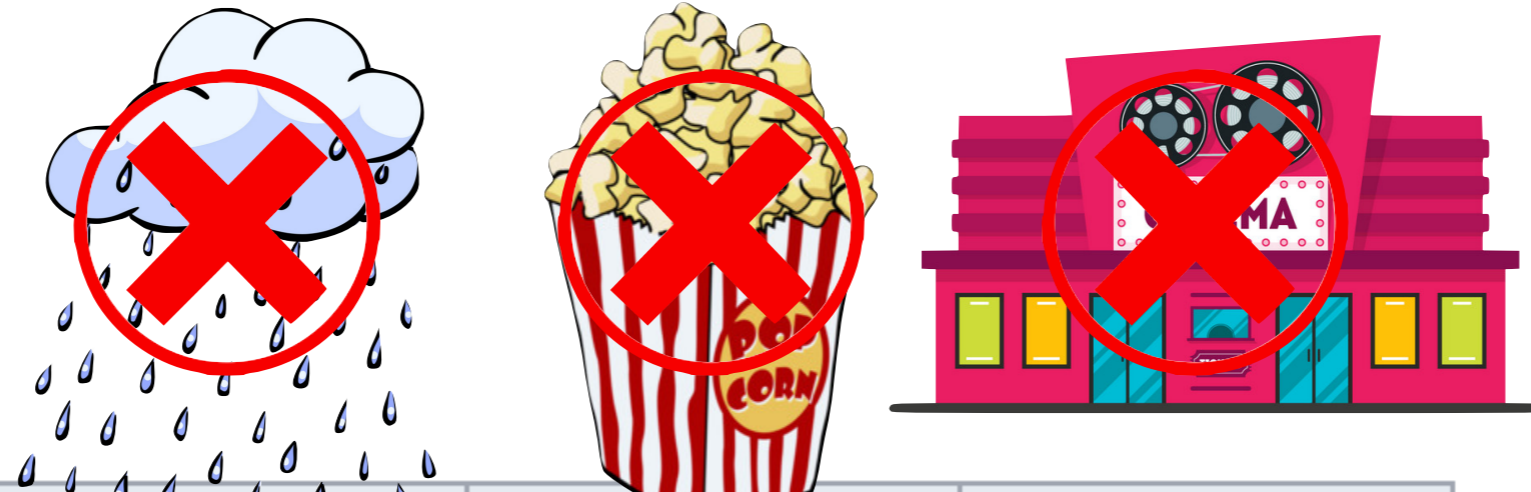
The illustration shows three elements: a blue cloud with raindrops and a large red 'X' over it, a red and white striped popcorn bucket with 'POPCORN' written on it, and a pink cinema building with a sign that says 'CINEMA' and a large red 'X' over it.

$p$	$q$	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

# Instruction logique: ET

## ET logique

« S'il pleut ET qu'il y a du popcorn, j'irai au cinéma »






$p$	$q$	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

# Instruction logique: OU

## OU logique

« S'il pleut OU s'il y a du popcorn, j'irai au cinéma »






$p$	$q$	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

# Instruction logique: OU

## OU logique

« S'il pleut OU s'il y a du popcorn, j'irai au cinéma »

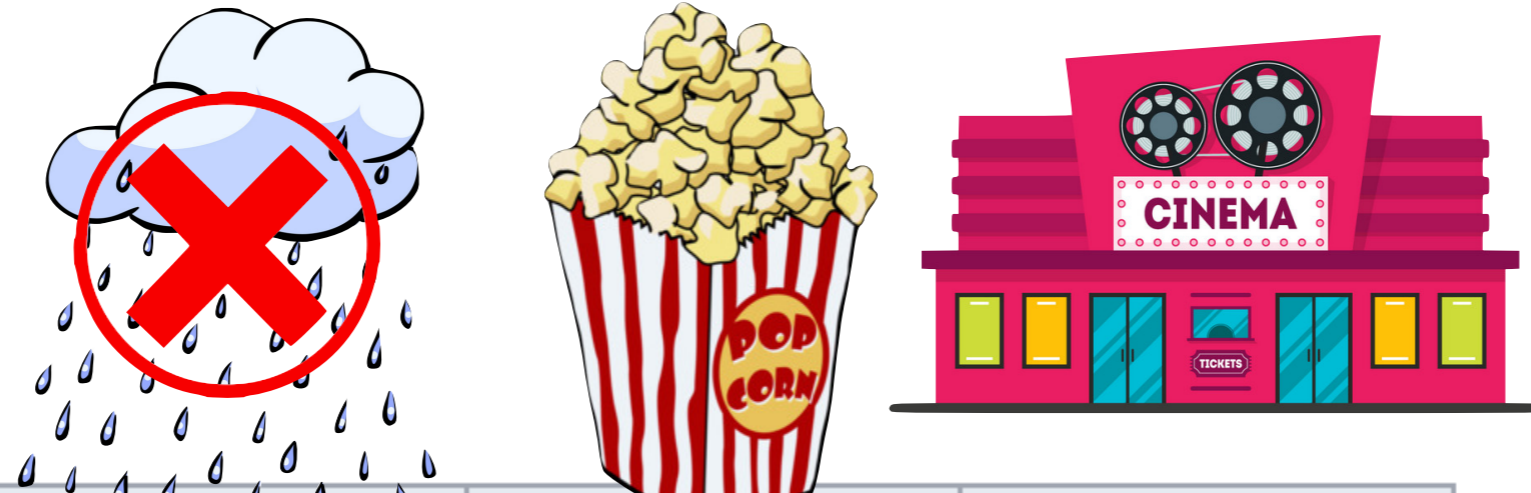


$p$	$q$	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

# Instruction logique: OU

## OU logique

« S'il pleut OU s'il y a du popcorn, j'irai au cinéma »



The illustration shows three elements: a blue cloud with raindrops and a large red 'X' over it, a red and white striped bucket of popcorn with a 'POP CORN' label, and a pink cinema building with a sign that says 'CINEMA' and a 'TICKETS' sign.

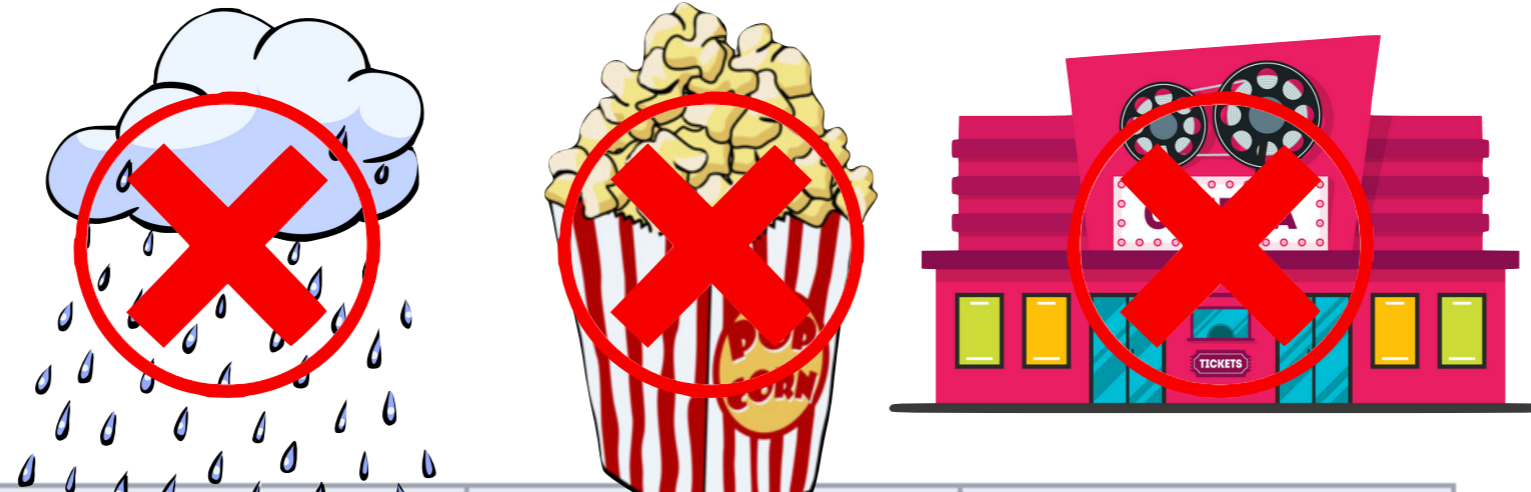
$p$	$q$	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0



# Instruction logique: OU

## OU logique

« S'il pleut OU s'il y a du popcorn, j'irai au cinéma »



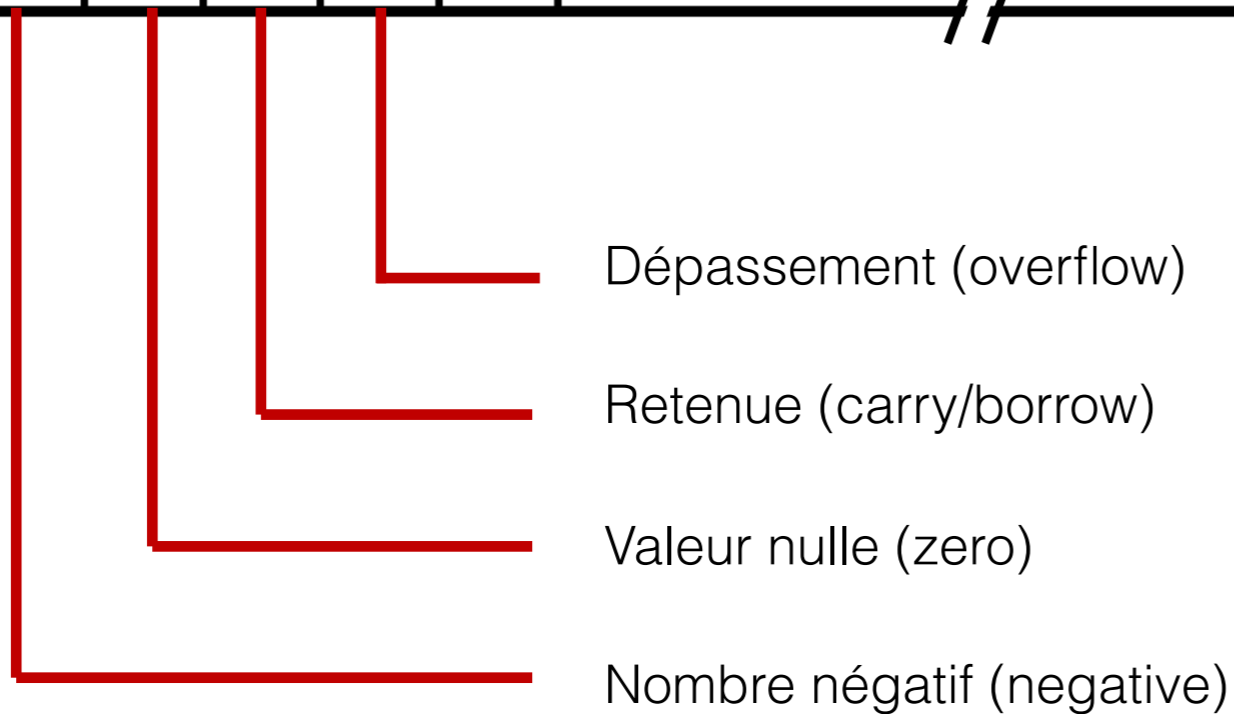
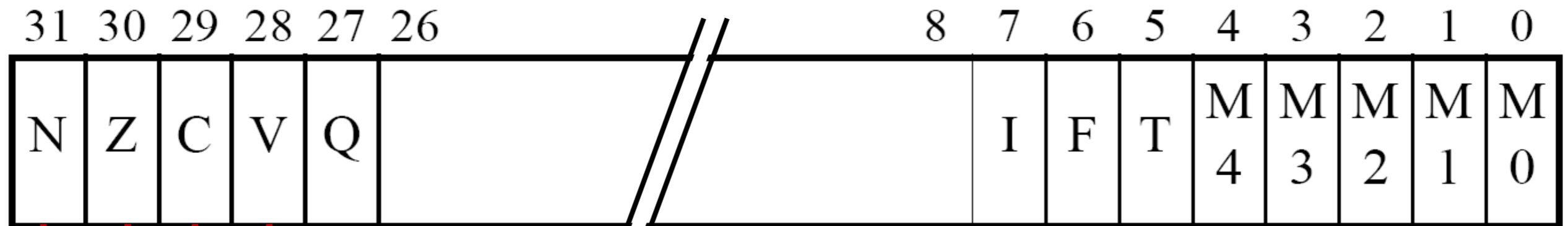
$p$	$q$	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

# Problème à résoudre

- But: comparer deux nombres placés dans R1 et R2
  - Si  $R1 > R2$ , mettre R3 dans R0
  - Sinon, mettre R4 dans R0
- Comment faire?
- Nous allons avoir besoin de trois mécanismes:
  1. **Une instruction pour comparer** R1 et R2
  2. **Un endroit pour stocker** le résultat de la comparaison
  3. **Des instructions conditionnelles**, activées seulement si la comparaison répond à certains critères

# 2. Endroit pour stocker

- Le registre d'état (CPSR) décrit l'état du processeur



On appelle ces 4 bits des **drapeaux** (*flags*).

Ces 4 drapeaux servent à stocker de l'information additionnelle sur les opérations arithmétiques effectuées par l'ALU.

# CPSR: détection de conditions

- N: Détection de signe négatif
  - 1 si résultat  $< 0$ , 0 autrement
- Z: Détection de zéro
  - 1 si résultat = 0, 0 autrement
  - Souvent utilisé pour détecter les égalités
- C: Détection de retenue (*carry*) ou d'emprunt (*borrow*)
  - 1 si l'opération a impliqué une retenue, 0 autrement
  - Ex. retenue d'addition de nombres positifs
- V: Détection de débordements (*overflow*)
  - 1 si l'opération a impliqué un débordement, 0 autrement
  - Ex. débordement signé lors d'une addition

# Dans le simulateur...

The screenshot displays a Cortex-M simulator interface with several key components:

- Simulation Panel:** Includes a 'User' tab, control buttons (Arrêter, Réinitialiser, Play, Step Back, Step Forward, Stop), and a 'Registre Généraux (User)' table.
- Registers Table:**

Nom	Valeur
R0	00000001
R1	fffffff
R2	00000000
R3	00000000
R4	00000000
R5	00000000
R6	00000000
R7	00000000
R8	00000000
R9	00000000
R10 (s1)	00000000
R11 (fp)	00000000
R12 (ip)	00000000
R13 (sp)	00000000
R14 (lr)	00000000
R15 (pc)	00000094
- Code Editor:** Shows assembly code with sections: SECTION INTVEC, SECTION CODE, and SECTION DATA. The current instruction is 'B main' at address 0x00000094.
- CPSR Flags:** A table showing the current state of flags:

État courant	CPSR	SPSR
Negatif (N)	Faux	
Zero (Z)	Faux	
Emprunt (C)	Vrai	
Dépassement (V)	Vrai	
Ignore IRQ	Faux	
Ignore FIQ	Faux	
- Memory View:** A table showing memory addresses and their corresponding hex values. The current instruction 'B -0x14' is highlighted in the memory view at address 0x00000094.
- Instruction Panel:** Shows the current instruction 'B -0x14' and its effect: '1. Soustrait la valeur 20 à PC'.
- PC and Cycle Info:** Shows the current PC value (0x00000094) and the current cycle (4).
- Breakpoint Instructions:** A list of instructions for activating breakpoints: Écriture (W) : Ctrl/Cmd + Clic, Lecture (R) : Shift + Clic, Lecture et Écriture (RW) : Ctrl/Cmd + Shift + Clic, Exécution (E) : Alt + Clic.

Drapeaux dans le CPSR

# 1. Instruction pour comparer

- L'instruction CMP compare deux nombres, et modifie les drapeaux de l'ALU

```
CMP R1, R2 ; calcule R1 - R2, change les drapeaux
```

# 1. Instruction(s) pour comparer

- Les instructions arithmétiques et logiques

```
INSTRUCTION Rd, Rs, Op  
; exécute l'instruction
```

modifient les drapeaux de l'ALU lorsque la lettre **S** est rajoutée après le nom de l'instruction

```
INSTRUCTIONS Rd, Rs, Op  
; exécute l'instruction et met à jour les drapeaux
```

- Exemple:

```
SUBS R0, R1, R2 ; R0 ← R1 - R2  
; et met à jour les drapeaux
```

# 3. Instructions conditionnelles

- L'instruction

```
MOVcc Rd, Op
```

met l'opérande  $Op$  dans le registre  $Rd$ , **si la condition  $cc$  est vraie**

- Exemple:

```
MOVEQ R3, R1
```

```
; R3 ← R1 seulement si le drapeau Z est 1
```

```
ADDNE R2, R2, R1
```

```
; R2 ← R2 + R1 seulement si le drapeau Z est 0
```



# Instructions conditionnelles

Code assembleur:

```
MOVEQ R3, R1  
; R3 ← R1 si le drapeau Z est 1
```

Équivalent, en C, à:

```
if (Z == 1) {  
    R3 = R1;  
}
```

Code assembleur:

```
ADDNE R2, R2, R1  
; R2 ← R2 + R1 si le drapeau Z est 0
```

Équivalent, en C, à:

```
if (Z == 0) {  
    R2 = R2 + R1;  
}
```

# Codes de condition

Table A8-1 Condition codes

cond	Mnemonic extension	Meaning (integer)	Meaning (floating-point) <sup>a</sup>	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS <sup>b</sup>	Carry set	Greater than, equal, or unordered	C == 1
0011	CC <sup>c</sup>	Carry clear	Less than	C == 0
0100	MI	Minus, negative	Less than	N == 1
0101	PL	Plus, positive or zero	Greater than, equal, or unordered	N == 0
0110	VS	Overflow	Unordered	V == 1
0111	VC	No overflow	Not unordered	V == 0
1000	HI	Unsigned higher	Greater than, or unordered	C == 1 and Z == 0
1001	LS	Unsigned lower or same	Less than or equal	C == 0 or Z == 1
1010	GE	Signed greater than or equal	Greater than or equal	N == V
1011	LT	Signed less than	Less than, or unordered	N != V
1100	GT	Signed greater than	Greater than	Z == 0 and N == V
1101	LE	Signed less than or equal	Less than, equal, or unordered	Z == 1 or N != V
1110	None (AL) <sup>d</sup>	Always (unconditional)	Always (unconditional)	Any

a. Unordered means at least one NaN operand.

b. HS (unsigned higher or same) is a synonym for CS.

c. LO (unsigned lower) is a synonym for CC.

d. AL is an optional mnemonic extension for always, except in IT instructions. For details see *IT* on page A8-104.

# Codes de condition

Table A8-1 Condition codes

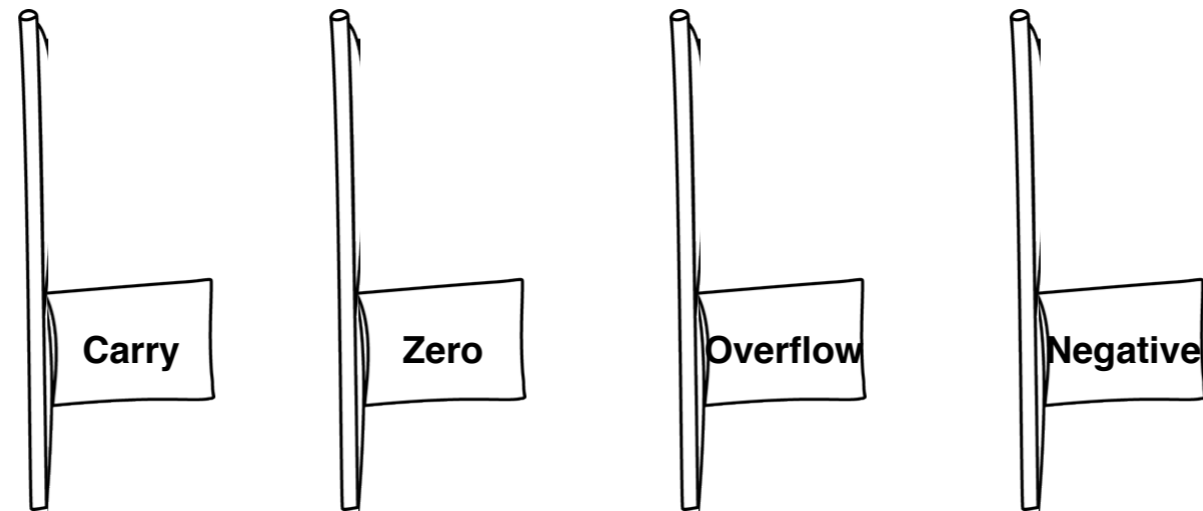
cond	Mnemonic extension	Meaning (integer)	Condition flags
0000	EQ	Equal	Z == 1
0001	NE	Not equal	Z == 0

Dans le cadre du cours, ces codes de condition nous seront les plus utiles.

1010	GE	Signed greater than or equal	N == V
1011	LT	Signed less than	N != V
1100	GT	Signed greater than	Z == 0 and N == V
1101	LE	Signed less than or equal	Z == 1 or N != V

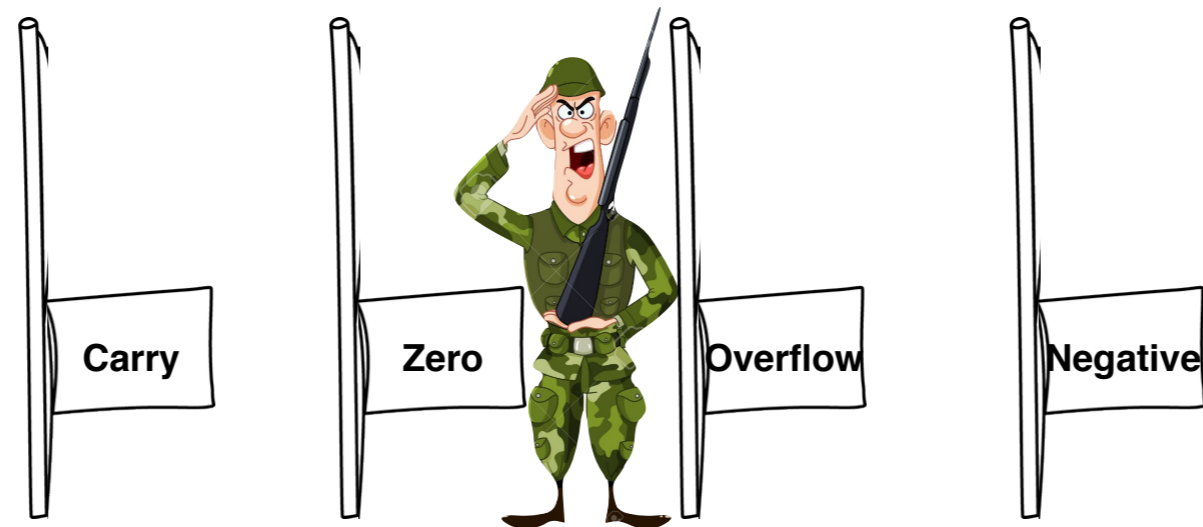
# Dans la série «analogies visuelles déconcertantes™»

```
SUBS  R0, R1, R2    ; R0 ← R1 - R2  
                        ; et met à jour les drapeaux
```



# Dans la série «analogies visuelles déconcertantes™»

```
SUBS  R0, R1, R2    ; R0 ← R1 - R2  
                        ; et met à jour les drapeaux
```

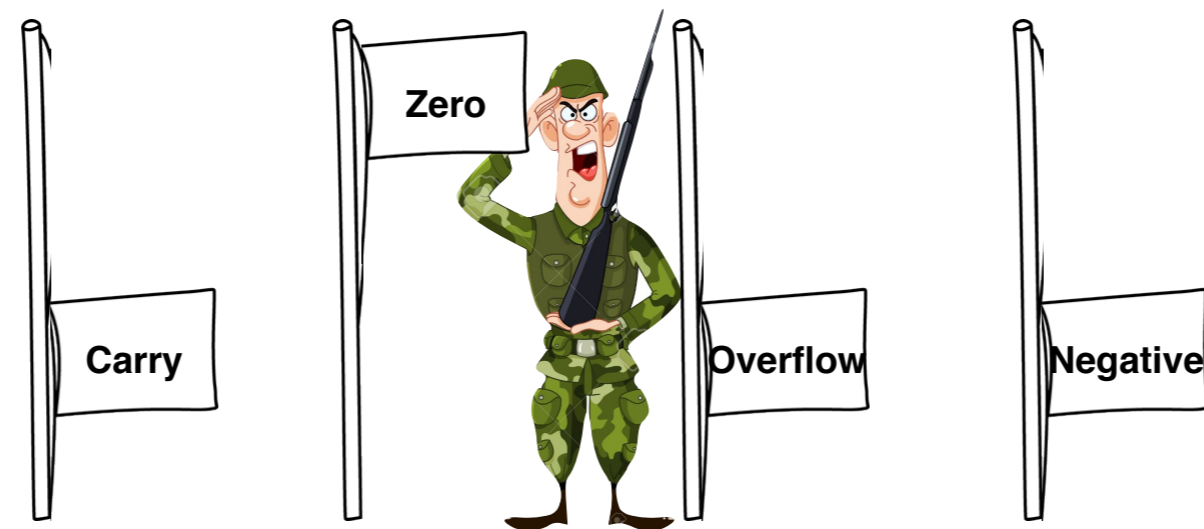


# Dans la série «analogies visuelles déconcertantes™»

```
SUBS R0, R1, R2 ; R0 ← R1 - R2  
                ; et met à jour les drapeaux
```



# Dans la série «analogies visuelles déconcertantes™»



```
ADDEQ R2, R2, R1
```

```
; R2 ← R2 + R1 seulement si le drapeau Z est 1
```

```
; sinon, on passe à l'instruction suivante
```

# Problème à résoudre #1

- But: comparer deux nombres placés dans R1 et R2
  - Si  $R1 > R2$ , mettre R3 dans R0
  - Sinon ( $R1 \leq R2$ ), mettre R4 dans R0
- Comment faire?

Code	Symbole
GT	>
GE	>=
LT	<
LE	<=



# Problème à résoudre #1

- But: comparer deux nombres placés dans R1 et R2
  - Si  $R1 > R2$ , mettre R3 dans R0
  - Sinon ( $R1 \leq R2$ ), mettre R4 dans R0
- Comment faire?

Code	Symbole
GT	>
GE	>=
LT	<
LE	<=

```
CMP  R1, R2      ; calcule R1 - R2, change les drapeaux
MOVGT R0, R3     ; si R1 > R2, R0 ← R3
MOVLE R0, R4     ; si R2 >= R1, R0 ← R4
```

# Démonstration (comparaisons #2)

# Problème à résoudre #2

Calculer la valeur absolue d'une différence:

$$R0 = \text{abs}(R1 - R2)$$

- Indices:

- si  $R1 > R2$ ,  $R0 = R1 - R2$
- sinon ( $R1 \leq R2$ ),  $R0 = R2 - R1$

Code	Symbole
GT	>
GE	>=
LT	<
LE	<=

# Problème à résoudre #2

Calculer la valeur absolue d'une différence:

$$R0 = \text{abs}(R1 - R2)$$

Code	Symbole
GT	>
GE	>=
LT	<
LE	<=

Solution (à trois instructions)

```
CMP R1, R2
; calcule R1 - R2, change les drapeaux
SUBGT R0, R1, R2 ; si R1 > R2, R0 ← R1 - R2
SUBLE R0, R2, R1 ; si R1 <= R2, R0 ← R2 - R1
```

# Problème à résoudre #2

Calculer la valeur absolue d'une différence:

$$R0 = \text{abs}(R1 - R2)$$

Code	Symbole
GT	>
GE	>=
LT	<
LE	<=

Solution (à deux instructions)

```
SUBS R0, R1, R2
; calcule R1 - R2, change les drapeaux

RSBLE R0, R0, #0
; si R1 <= R2, R0 = -R0 (donc R0 = R2 - R1)
```

# Démonstration (valeur absolue)