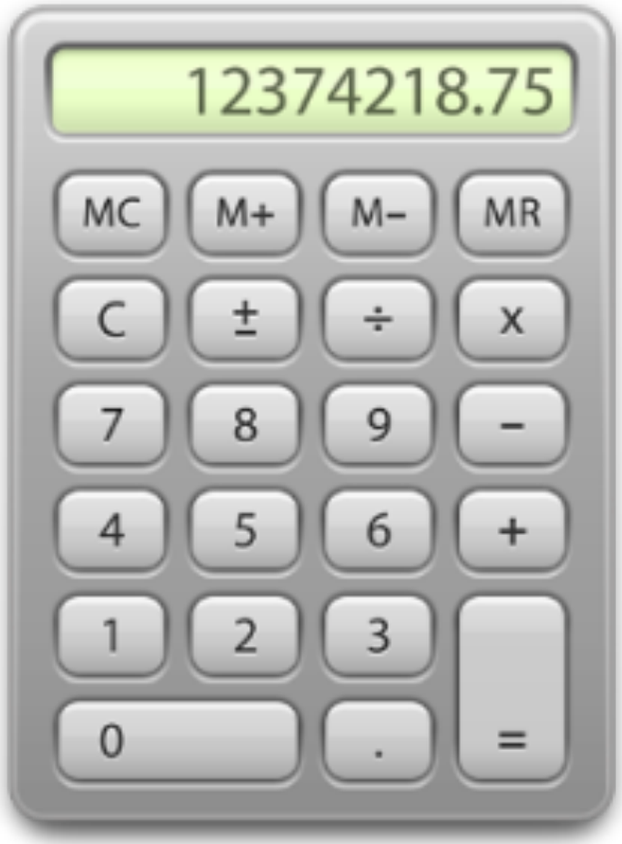
Introduction à l'assembleur ARM: arithmétique et conditions



GIF-1001 Ordinateurs: Structure et Applications, Hiver 2019

Jean-François Lalonde

Instructions arithmétiques et logiques

- Les opérations mathématiques et logiques ont la forme

```
INSTRUCTION Rd, Rs, Op1
```

- Où
  - Rd est le registre de destination
  - Rs est un registre source
  - Op1 est une opérande de type 1
- Le format de l'instruction ADD, par exemple, est:

```
ADD Rd, Rs, Op1          ; Rd ← Rs + Op1
```

```
ADD R0, R0, #1           ; R0 ← R0 + 1
ADD R0, R0, R1           ; R0 ← R0 + R1
ADD R0, R0, R1, LSL #1   ; R0 ← R0 + (R1 * 2)
```

Exemples

- Soustraction

```
SUB R0, R0, #1          ; R0 ← R0 - 1
SUB R0, R0, R1          ; R0 ← R0 - R1
```

- Décalage

```
LSL R0, R0, #1          ; R0 ← R0 * 2
ASR R0, R0, #2          ; R0 ← R0 / 4 (préserve le signe)
```

- “Et” logique

```
AND R0, R0, #1          ; R0 ← R0 ET 1
AND R0, R0, R1          ; R0 ← R0 ET R1
```

- Prendre le négatif

```
RSB R0, R0, #0          ; R0 ← 0 - R0, donc R0 = -R0
```

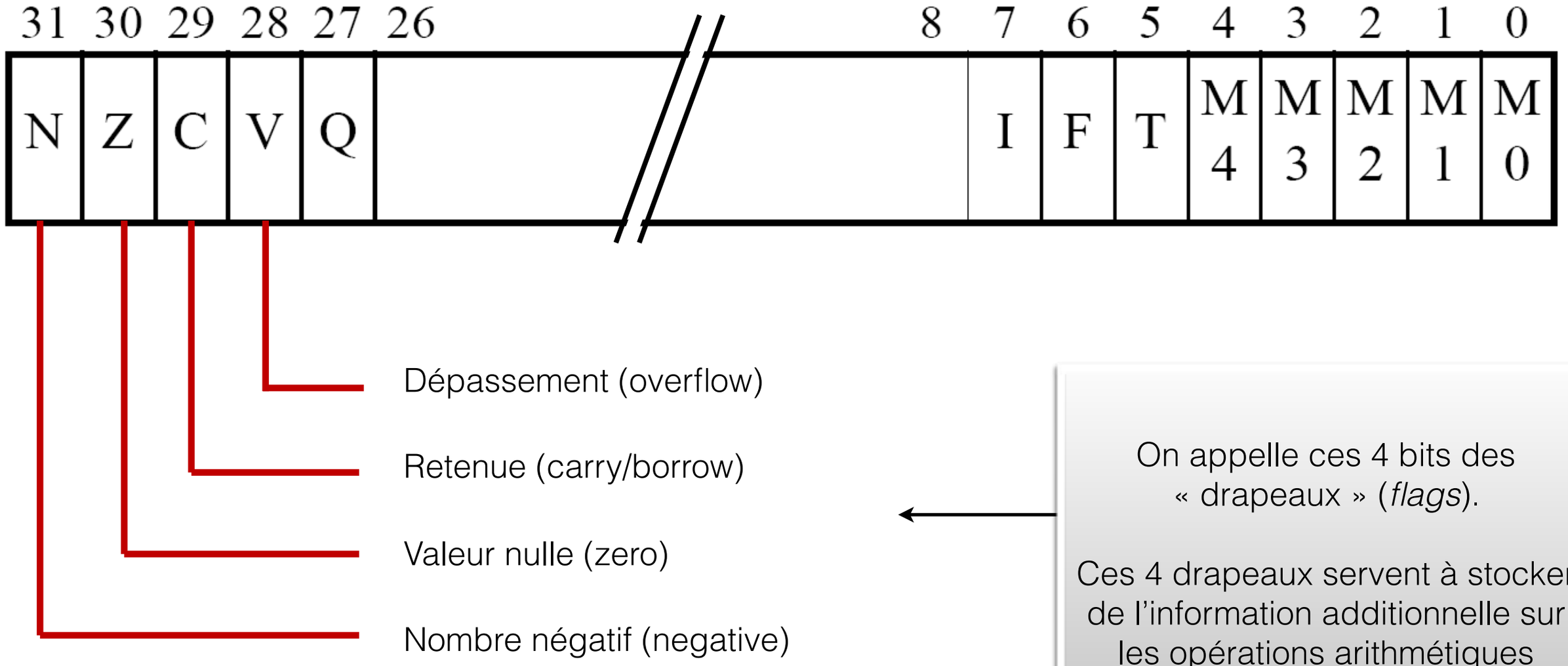
Problème à résoudre

- 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

- Un registre d'état 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

Retenue (*carry*) avec nombres **non-signés**

- Nombres **non-signés** sur 4 bits
- 10 + 8 = ?
  - 0b1010 + 0b1000 = 0b1 0010
    - Nous avons besoin d’un 5e bit pour que le résultat soit valide, c’est le bit *carry*!

Retenue (*carry*) avec nombres **signés**

- Nombres **signés** (complément-2) sur 4 bits
- -2 + 2 = ?
  - 0b1110 + 0b0010 = 0b1 0000
    - Le résultat est valide sur 4 bits, mais un 5e bit est activé: il y a *carry*!
- Est-ce qu'il y a débordement?
  - Non car les deux bits de signes étaient différents initialement.

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

```
INSTRUCTIONS Rd, Rs, Op1    ; exécute l'instruction,
                            ; et met à jour les drapeaux
```

- Exemple:

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

Quels seront les drapeaux N et Z du CPSR?

Démonstration (comparaisons)

3. Instructions conditionnelles

- L'instruction

```
MOVcc Rd, Op1
```

met l'opérande de type 1 `Op1` 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 seulement si le drapeau Z est 1
```

Équivalent, en C, à:

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

Code assembleur:

```
ADDNE R2, R2, R1   ; R2 ← R2 + R1 seulement 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) [a]	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS [b]	Carry set	Greater than, equal, or unordered	C == 1
0011	CC [c]	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) [d]	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.

Manuel de référence ARM, p. A8-8

Codes de condition

Table A8-1 Condition codes

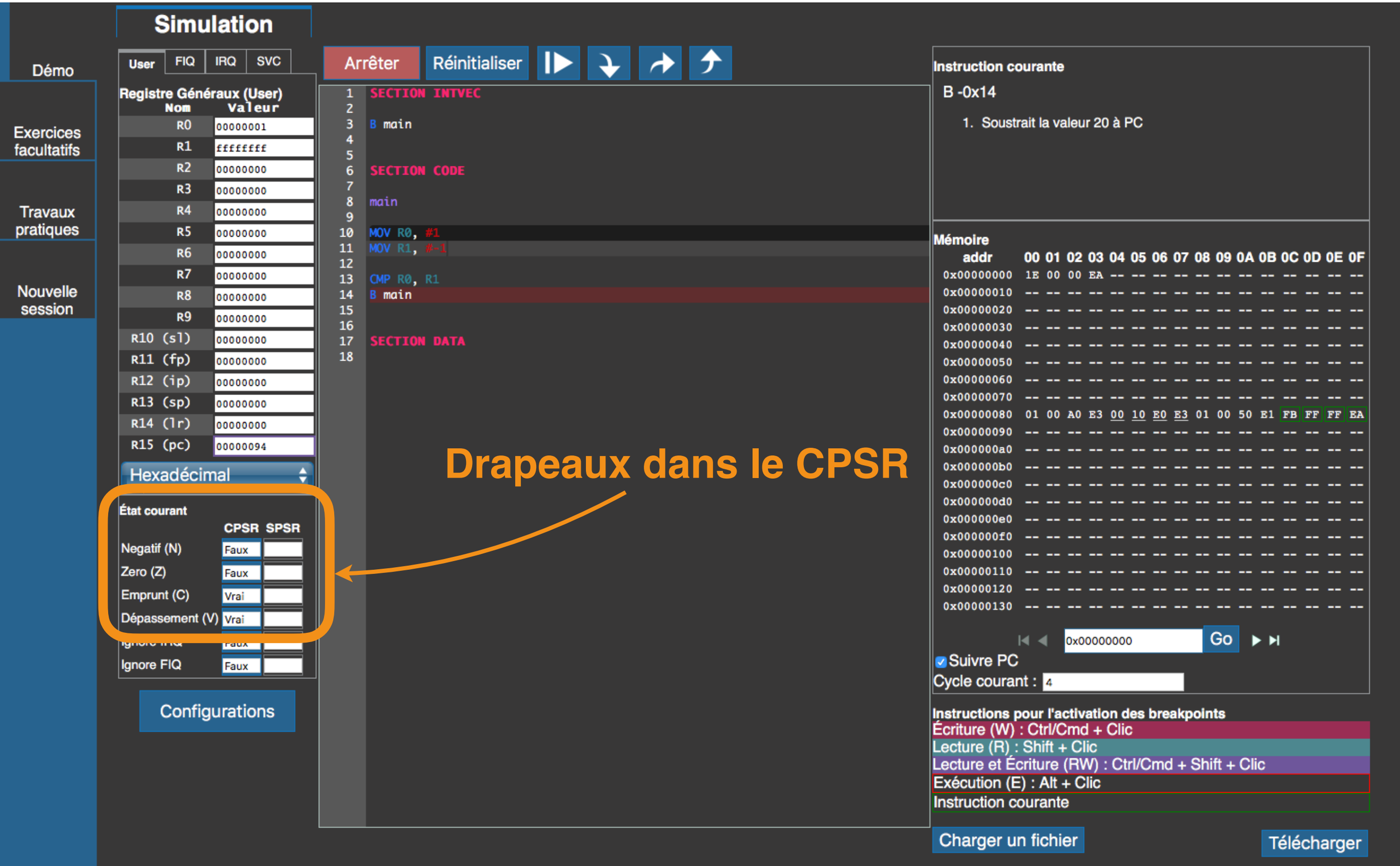
cond	Mnemonic extension	Meaning (integer)	Meaning (floating-point) [a]	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS [b]	Carry set	Greater than, equal, or unordered	C == 1
0011	CC [c]			C == 0
0100	MI			N == 1
0101	PL			N == 0
0110	VS			V == 1
0111	VC			V == 0
1000	HI			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) [d]	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.

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

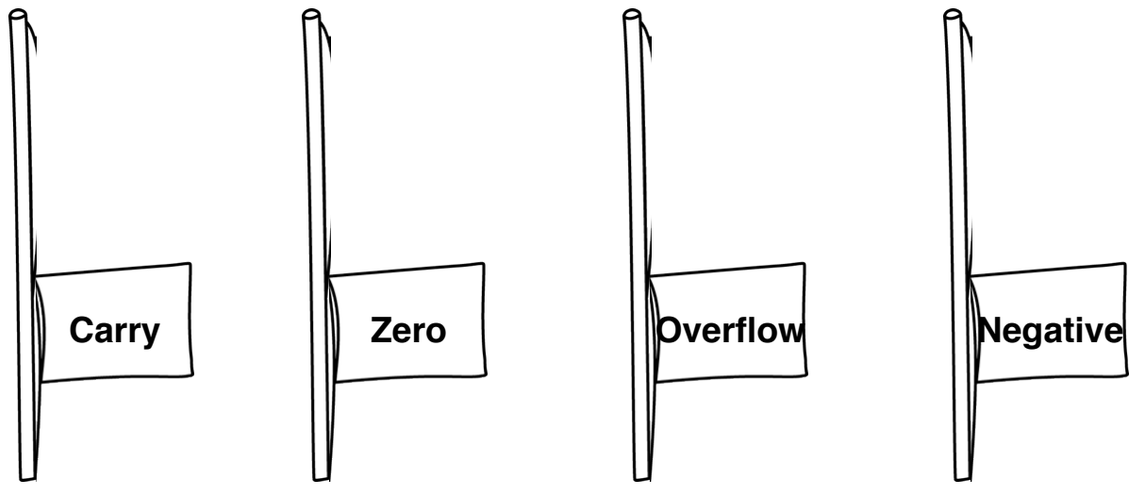
Manuel de référence ARM, p. A8-8

Dans le simulateur…



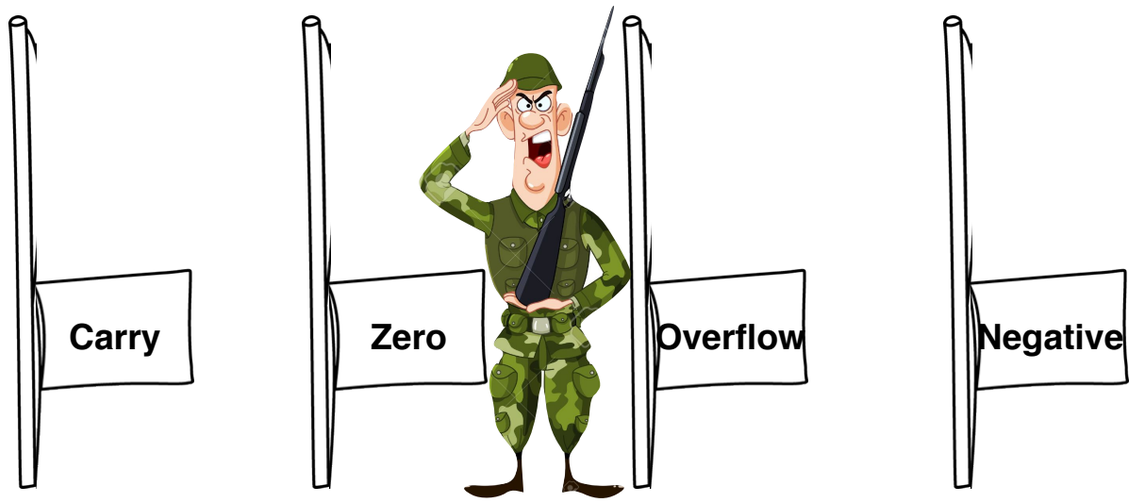
Dans la série «analogies visuelles déconcertantes™»

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



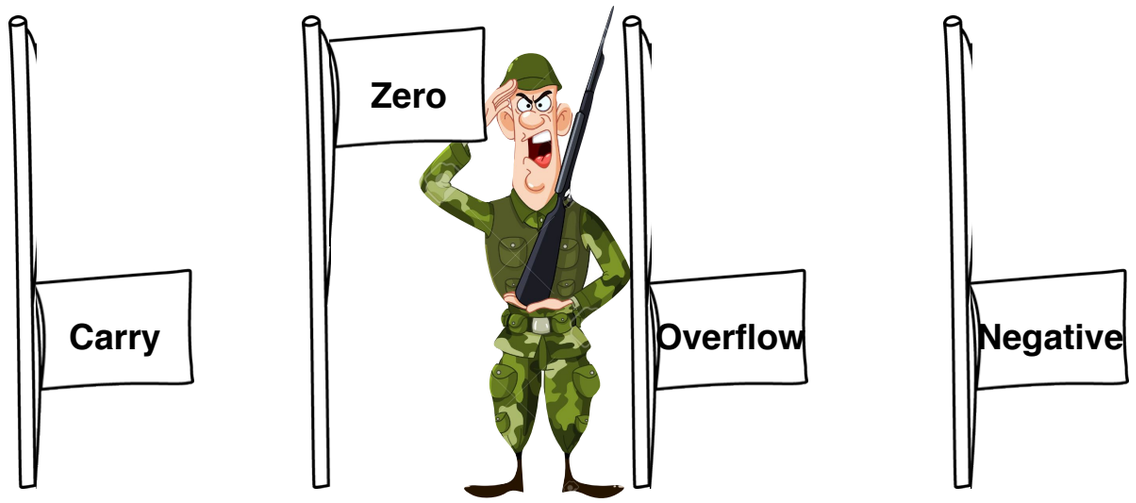
Dans la série «analogies visuelles déconcertantes™»

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

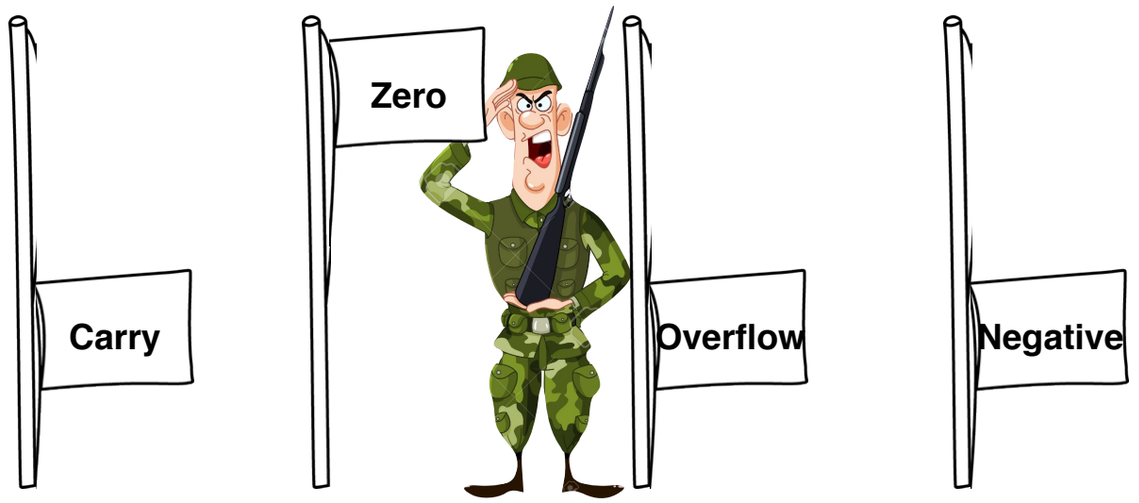


Dans la série «analogies visuelles déconcertantes™»

```
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

- But: comparer deux nombres placés dans R1 et R2
  - Si R1 > R2, mettre R3 dans R0
  - Sinon (R1 <= R2), mettre R4 dans R0

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

- Comment faire?

```
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

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

- But:
  - R0 = abs(R1 - R2) ; valeur absolue
- Comment faire? Indices:
  - R0 = R1 - R2 si R1 > R2
  - R0 = R2 - R1 sinon
  - l'instruction RSB peut être utilisée pour calculer le négatif d'un registre

```
RSB R0, R0, #0      ; R0 ← -R0
```

- Solution (à 3 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
```

- Solution (à 2 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)