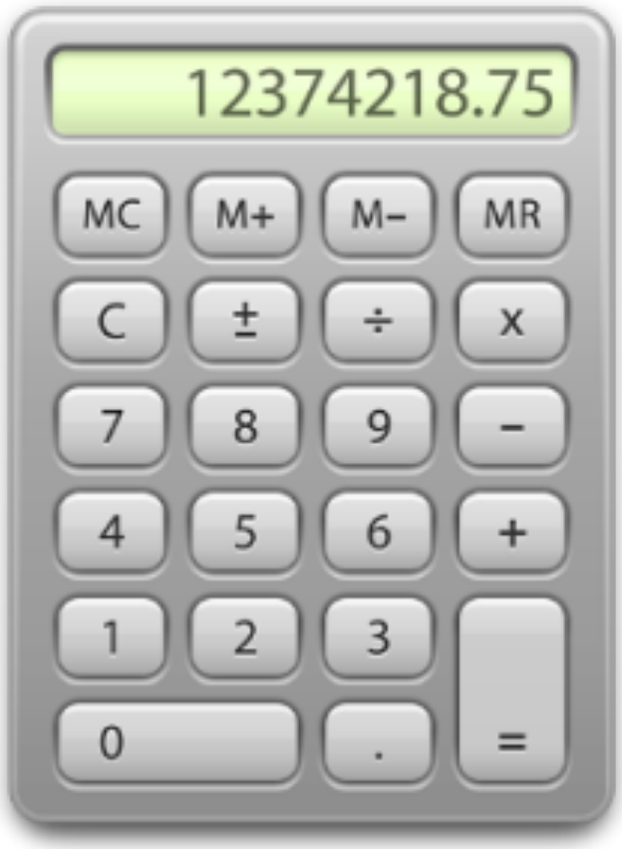
Introduction à l'assembleur ARM: arithmétique et conditions



GIF-1001 Ordinateurs: Structure et Applications, Hiver 2015

Jean-François Lalonde

Logistique

- TP1
  - notes disponibles cette semaine
- TP2: “Modes d'adressage en assembleur ARM”
  - dû ce soir (23h59), remettez le fichier tp2.docx sur Piazza
  - des questions?
- TP3: “Branchements et appel de fonctions”
  - disponible dès aujourd’hui
  - dû dans deux semaines: 24 février, 23h59
  - on en parle à la fin du cours — rappelez-le moi!

Plan

- La semaine dernière:
    - Déclarer des variables et leur affecter des valeurs
- Cette semaine:
    - Effectuer des opérations mathématiques et logiques
    - Gérer la séquence des opérations avec des énoncés conditionnels et des boucles
    - Appeler des fonctions
- La semaine prochaine:
    - Gérer les évènements et les exceptions

Rappel: instructions

- De quoi une instruction est-elle constituée?
  - “Opcode” (ou code d’opération): code identifiant quelle instruction est effectuée (MOV, LDR, etc.)
  - Paramètres: un ou plusieurs, dépendent de l’opcode.

Rappel: ARM

- Les instructions sont encodées sur combien de bits?
  - 32!
- Quelle est la valeur de PC?
  - L'adresse de l'instruction courante **+ 8**
  - Toujours 2 instructions "en avance"
- Quelle est la différence entre MOV et LDR/STR?
  - MOV: entre les registres, à l'intérieur du microprocesseur
  - LDR/STR: entre le microprocesseur et la mémoire

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
```

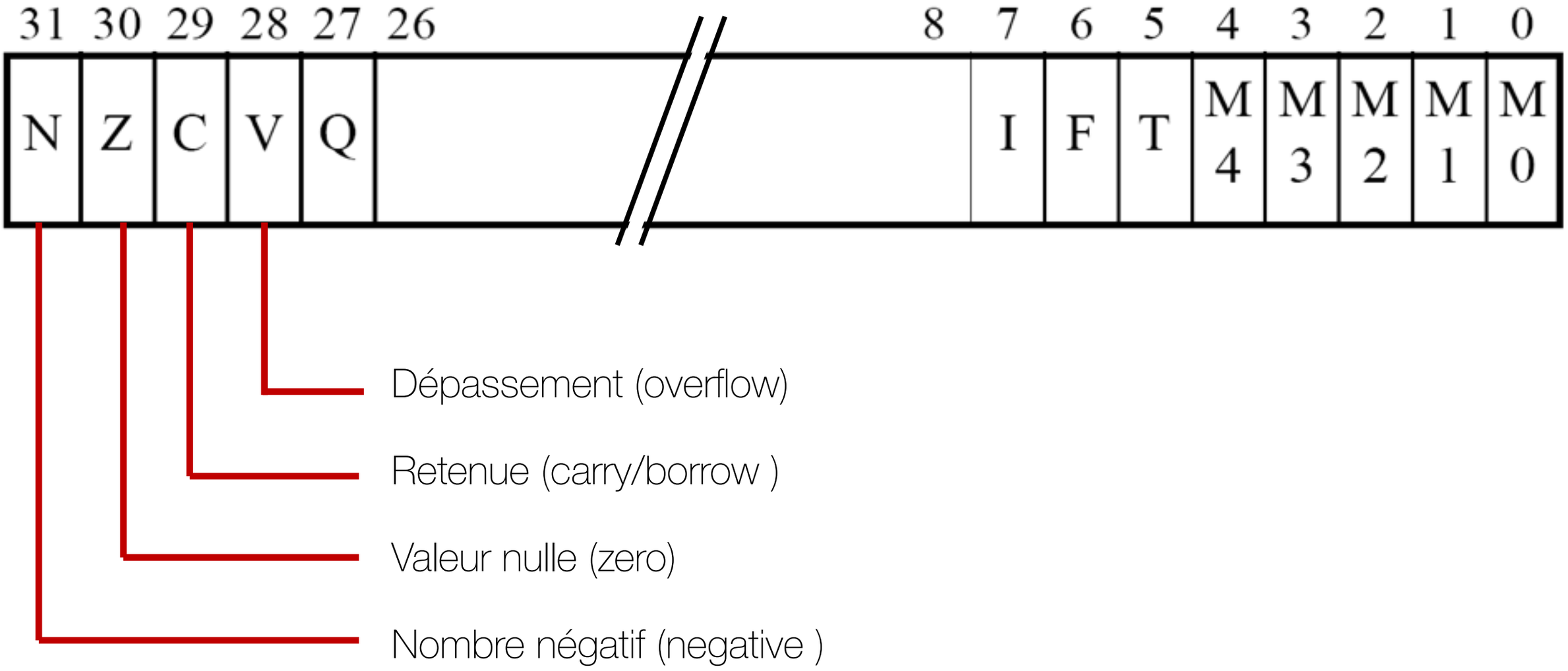
Problème à résoudre

- But: comparer deux nombres placés dans R1 et R2
  - Si R1 > R2, mettre R3 dans R0
  - Si R2 >= R1, mettre R4 dans R0
- Comment faire?

- Nous allons avoir besoin de trois mécanismes:
  - Une instruction pour comparer R1 et R2
  - Un endroit pour stocker le résultat de la comparaison
  - Des instructions pouvant être activées si la comparaison répond à certains critères

Rappel: registre de statut (CPSR)

- Un registre de statut décrit l'état du processeur



Instructions avec drapeaux

- Les instructions arithmétiques et logiques changent les drapeaux de l'ALU, lorsque l'option "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?

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 dépassements (overflow)
  - 1 si l’opération a impliqué un dépassement, 0 autrement
  - Ex. dépassement signé lors d’une addition

Instructions conditionnelles

- L'instruction

```
MOVcc Rn Op1
```

met l'opérande de type 1 `Op1` dans le registre `Rn`, *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 (CC)

- Plusieurs instructions s'exécutent si une condition est rencontrée.
- Toutes les conditions sont évaluées à partir des drapeaux de L'ALU et assument que ceux-ci ont été déterminés auparavant.
  - Par exemple, la condition EQ (equal) assume qu'une soustraction ou comparaison a été faite avant: si le résultat de l'opération est 0, le drapeau Z sera à 1 et la condition EQ sera rencontrée.
- Les drapeaux N (Negative), Z (Zero), C (Carry) et V (Overflow) servent à évaluer toutes les conditions.
- Les drapeaux et les conditions à évaluer changent si les nombres comparés sont signés ou s'ils ne le sont pas.

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.

Problème à résoudre

- But: comparer deux nombres placés dans R1 et R2
  - Si R1 > R2, mettre R3 dans R0
  - Si R2 >= R1, 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
```

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):

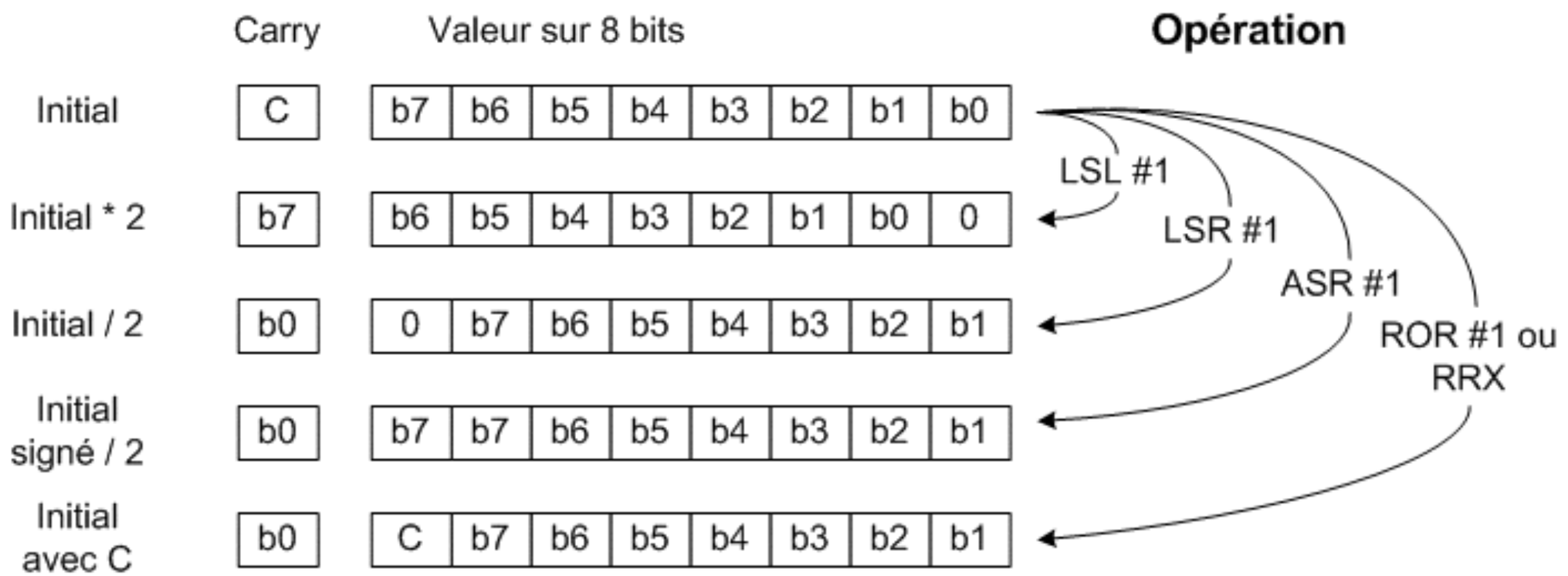
```
SUBS  R0, R1, R2    ; calcule R1 - R2, change les drapeaux
RSBLE R0, R0, #0    ; si R1 <= R2, R0 = -R0 (donc R0 = R2 - R1)
```

Exemple: Addition de variables sur 64 bits

```
  NAME    Additionne64Bits    ; Nom du fichier/module; Addition sur 64 bits en little endian

  PUBLIC  Additionne64Bits    ; Rend l'étiquette/fonction Additionne64Bits disponible pour d'autres
fichiers

  SECTION .intvec : CODE (2)  ; Le texte qui suit doit être assemblé dans la mémoire FLASH
  CODE32                      ; Les instructions qui suivent sont sur 32 bits

Additionne64Bits
  LDR     R0, =Op1        ; Met l'adresse de Op1 dans R0
  LDR     R1, [R0]        ; Met les 4 octets les moins significatifs d'Op1 dans R1
  LDR     R2, [R0,#4]     ; Met les 4 octets les plus significatifs d'Op1 dans R2
  LDR     R0, =Op2        ; Met l'adresse de Op2 dans R0
  LDR     R3, [R0]        ; Met les 4 octets les moins significatifs d'Op2 dans R3
  LDR     R4, [R0,#4]     ; Met les 4 octets les plus significatifs d'Op2 dans R4
  ADDS    R5, R1,R3       ; R5 = R1+R3, additionne les octets les moins significatifs, met à jour Carry
  ADC     R6, R2,R4       ; Additionne les octets les plus significatifs avec la retenue
  LDR     R0, =Res        ; Met l'adresse du résultat dans R0
  STR     R5, [R0]        ; Sauvegarde le résultat: bit les moins significatifs
  STR     R6, [R0,#4]     ; Sauvegarde le résultat: bit les plus significatifs
Fin
  B       Fin             ; Terminé

  DATA                    ; Le texte qui suit doit être assemblé dans la mémoire FLASH, constantes

Op1  DC32     0x11111111, 0x22222222      ; Déclaration de l'opérande 1 sur 2 fois 32bits
Op2  DC32     0x33333333, 0x44444444      ; Déclaration de l'opérande 2 sur 2 fois 32bits

  SECTION `no_init`: DATA (2)  ; Le texte qui suit doit être assemblé dans la mémoire RAM
Res  DS32  2                ; Déclaration du résultat sur 2 fois 32 bits

  END                       ; Fin du fichier/module
```

Annexe 1: Décalage de bits

- LSL, Logical Shift Left, décale les bits vers la gauche et met des zéros à droite. Décaler un bit vers la gauche équivaut à multiplier par 2. Carry devient égal au bit le plus significatif.
- LSR, Logical Shift Right, décale les bits vers la droite et met des 0 à gauche. Décaler un bit vers la droite équivaut à diviser un nombre non-signé par 2. Carry devient égal au bit le moins significatif.
- ASR, Arithmetical Shift Right, décale les bits vers la droite et copie le bit le plus significatif à gauche. Décaler un bit vers la droite en conservant le bit de signe équivaut à diviser un nombre signé par 2. Carry devient égal au bit le moins significatif.
- ROR, Rotate Right, décale les bits vers la droite et met le Carry à gauche. Carry devient égal au bit le moins significatif.
- RRX, Rotate Right eXtended équivaut à ROR #1.



Annexe 2: Qu'est-ce qu'une variable?

- Pour le microprocesseur, les variables n'existent pas: le microprocesseur lit et exécute des instructions. Certaines instructions (LOAD et STORE) lui demandent d'accéder à certaines adresses de la mémoire. La plupart des instructions lui demandent de modifier ses registres internes.

  Pour le programmeur et le mathématicien, une variable est un objet ayant une certaine valeur qui peut changer dans le temps. Pour le programmeur, une variable a un type, c'est-à-dire un format et une taille (exemple: un entier sur 32 bits) et une portée (la variable peut être utilisée dans la fonction seulement, dans le fichier seulement ou dans tout le programme).

  Le compilateur (ou l'assembleur) et l'éditeur de liens font la relation entre les variables du programmeur et le monde du microprocesseur. Ces programmes associent une adresse de mémoire (ou un registre) aux variables que le programmeur déclare. Lorsque le programme du programmeur lit ou écrit une variable, le compilateur transforme cette lecture ou écriture en instructions qui accèderont aux adresses de mémoires (ou aux registres) allouées aux variables...