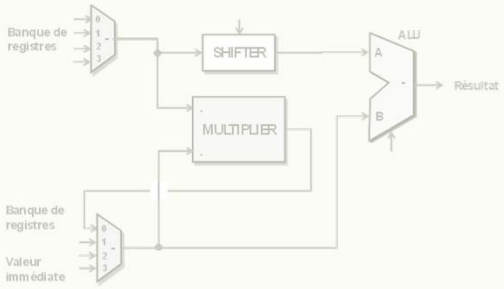


ARO2

Micro-architecture d'un processeur
La partie EXECUTE

Basé sur le cours du prof. E. Sanchez
et le cours ASP du prof. M.Starkier

Romuald Mosqueron

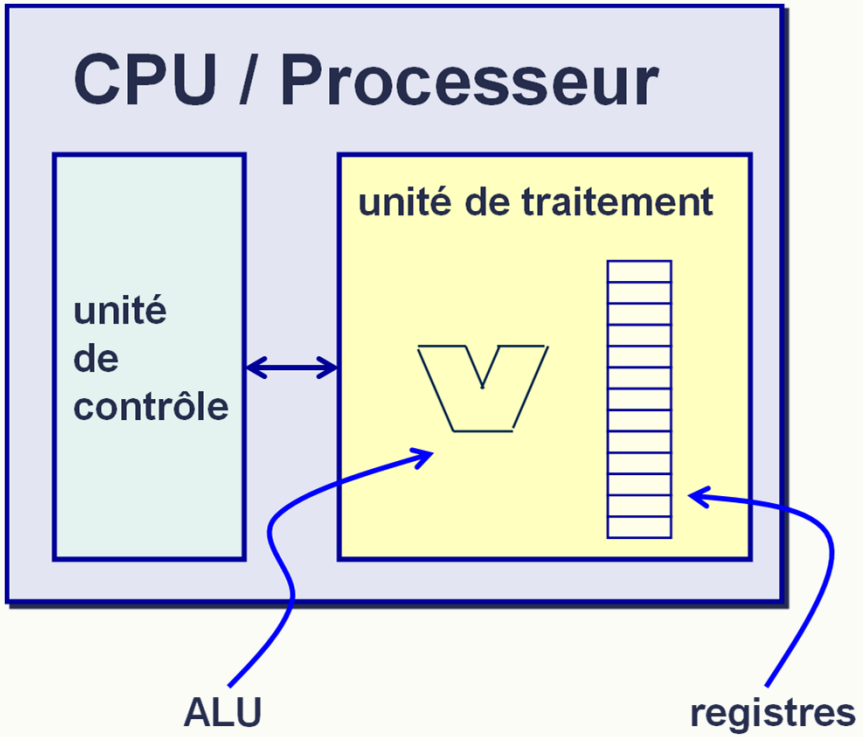
Le bloc EXECUTE

- Le bloc EXECUTE fait partie de l'unité de traitement
- Le bloc EXECUTE contient le bloc ALU, le registre CPSR, le bloc SHIFTER. Les opérations sont effectuées sur des entiers (signés/non-signés).
- Le bloc EXECUTE peut contenir aussi une FPU permettant le calcul en virgule flottante (IEEE 754).

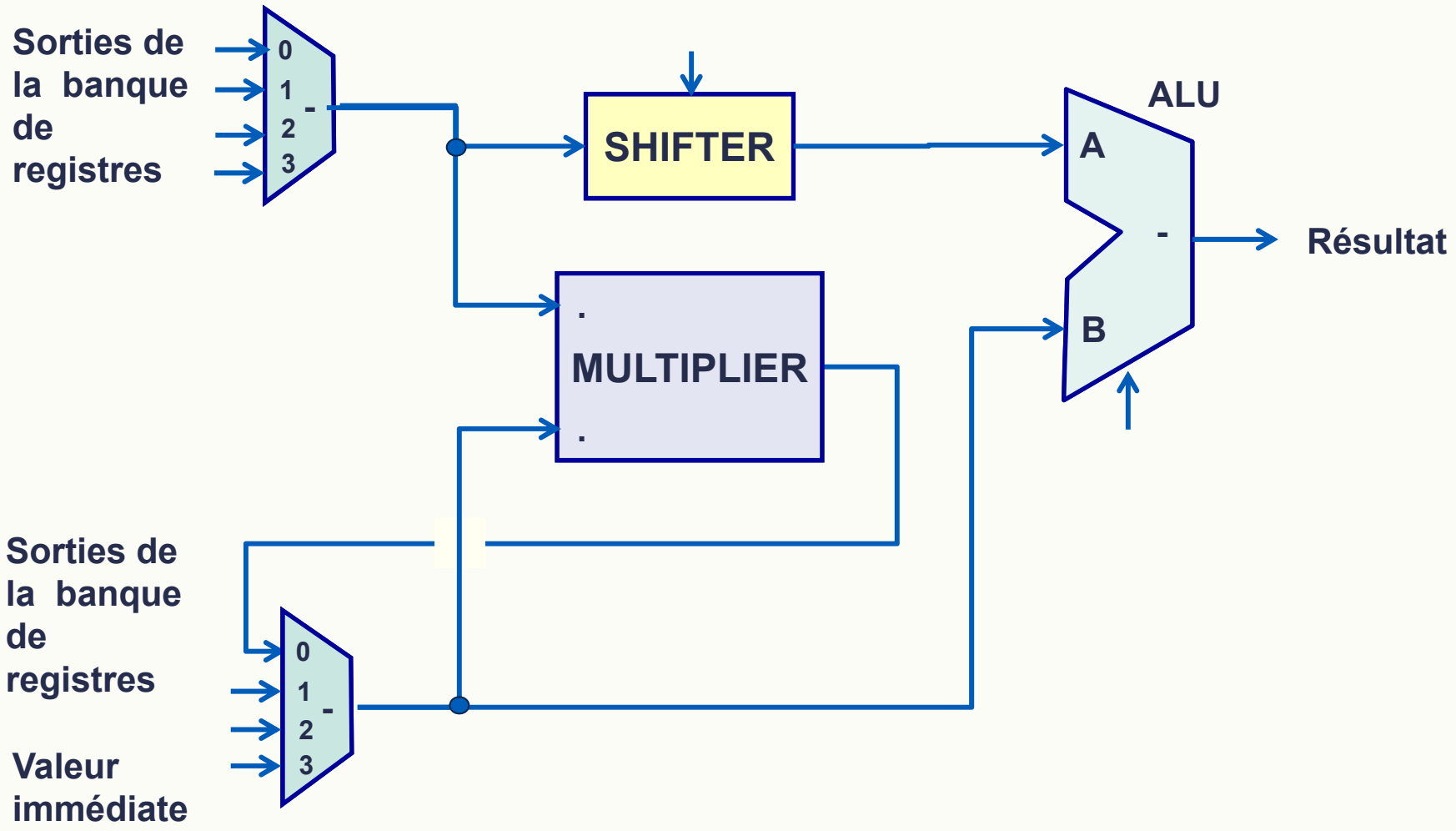
Le bloc EXECUTE

Rappel

Bloc diagramme d'un processeur avec l'unité de contrôle et l'unité de traitement



ARM :Bloc diagramme EXECUTE



Cours AR02

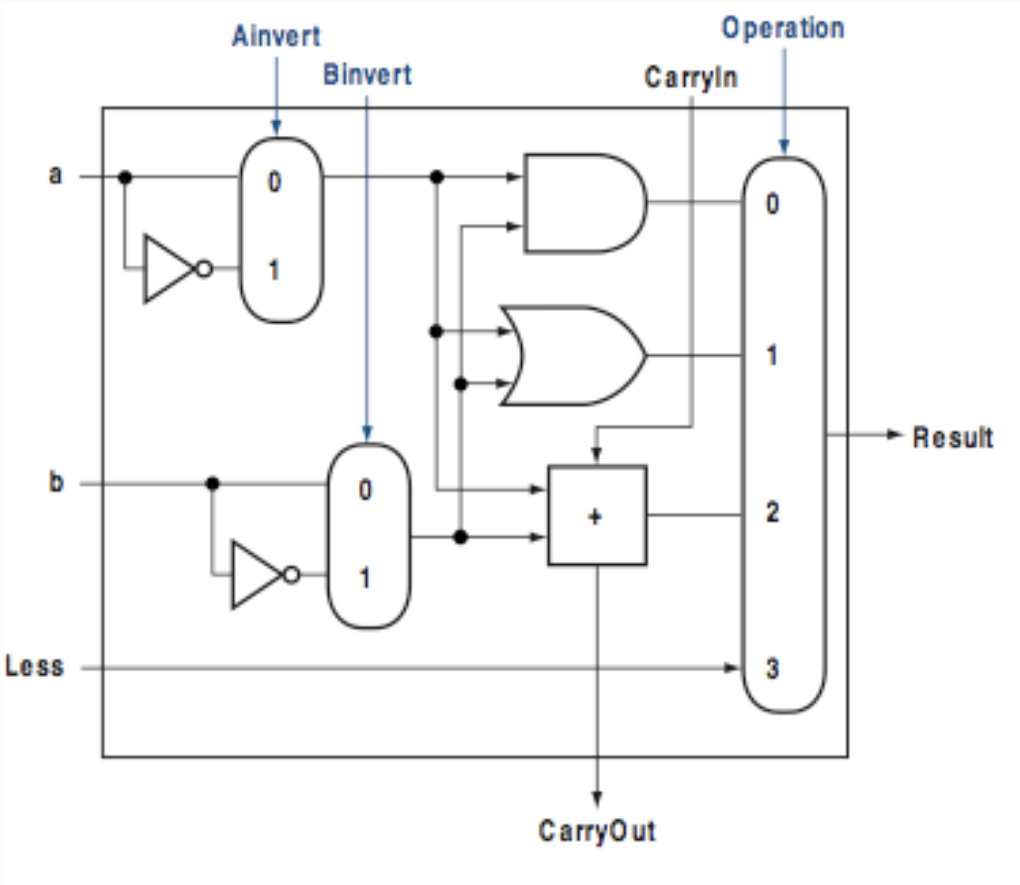
ALU : OPERATIONS SUR LES ENTIERS

Le bloc ALU

- Opérations arithmétiques
  - sur des entiers signés/non-signés
  - addition, soustraction, comparaison
  - multiplication, division, racine
  - détection zéro, signe, carry et overflow
- Opérations logiques
  - not, and, or, xor, nand, nor, nxor

Le bloc ALU

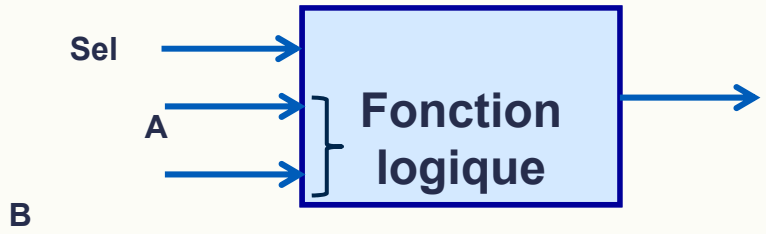
- ALU dans ARO1



Opérations logiques

- Les opérations logiques entre deux opérandes (contenus de registre ou valeur immédiate) sont effectuées bit à bit.

- Il existe une instruction pour chaque fonction logique.
  - Exemples ARM :AND <Rd>, <Rm>, ORR <Rd>, <Rm>,...
- Un circuit combinatoire permet de sélectionner la fonction logique désirée



Opérations logiques

- **Exemple de fonction logique universelle (4 bits de sélection)**

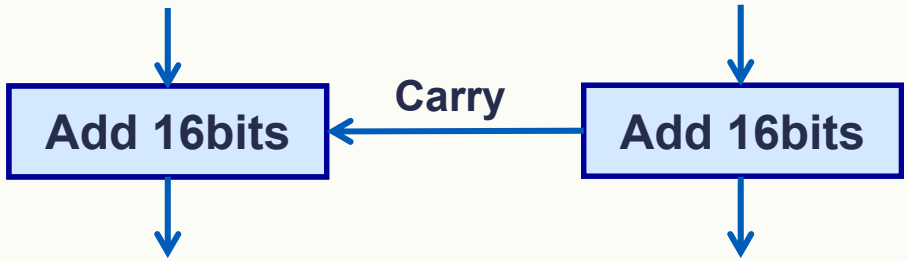
- $F_{logique}(S_3, S_2, S_1, S_0) = S_3 \cdot \bar{A} \cdot \bar{B} + S_2 \cdot A \cdot \bar{B} + S_1 \cdot \bar{A} \cdot B + S_0 \cdot A \cdot B$

Fonction	S_3	S_2	S_1	S_0	Equation
AND	0	0	0	1	$A \cdot B$
OR	0	1	1	1	$A \cdot \bar{B} + \dot{A} \cdot B + A \cdot B$
NAND	1	1	1	0	$\dot{A} \cdot \bar{B} + A \cdot \bar{B} + \dot{A} \cdot B$
NOR	1	0	0	0	$\dot{A} \cdot \bar{B}$
XOR	0	1	1	0	$A \cdot \bar{B} + \dot{A} \cdot B$
NXOR	1	0	0	1	$\dot{A} \cdot \bar{B} + A \cdot B$
NOT(A)	1	0	1	0	$\dot{A} \cdot \bar{B} + \dot{A} \cdot B$

Addition des nombres non-signés

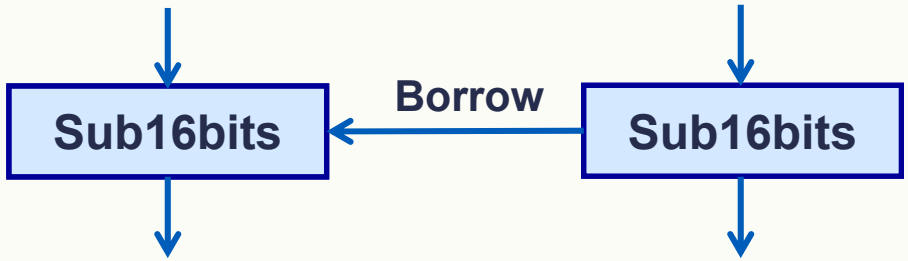
- Pour n bits : $0 \leq val \leq 2^n-1$

- Dépassement en cas d'addition si le résultat est supérieure à 2^n-1
- Génération d'un flag de Carry (report) dans le registre de status
- Le Carry permet
  1. générer une erreur
  2. étendre le nombre de bits (cascader les additions)



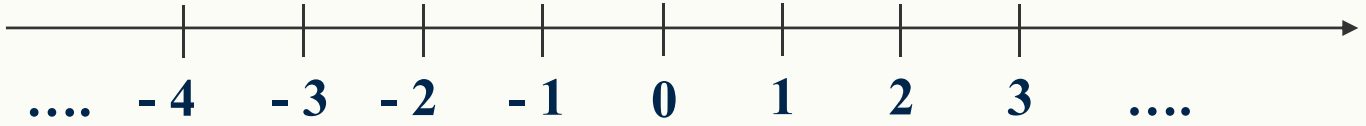
Soustraction des nombres non-signés

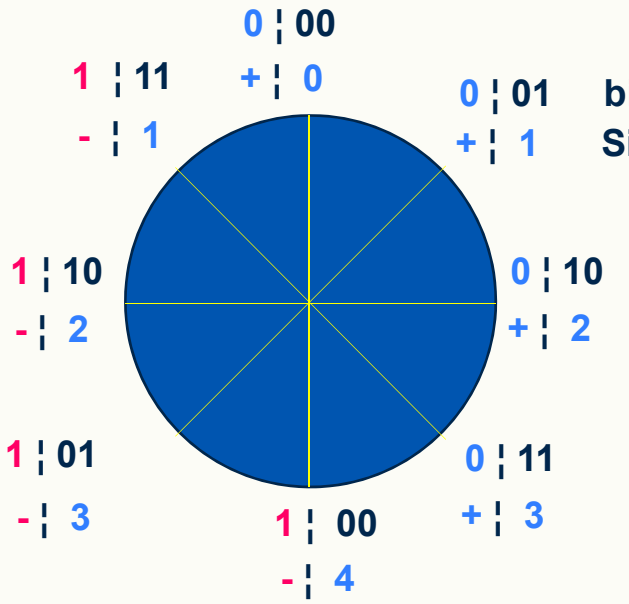
- Dépassement en cas de soustraction si le résultat est inférieur à 0
- Génération d'un Borrow (emprunt)
- Le Borrow est traduit par un flag de Carry inversé dans le registre de status : Carry = NOT(Borrow)
- Le NOT(Borrow) permet
  1. générer une erreur
  2. étendre le nombre de bits (cascader les soustractions)



Rappel: nombres signés Complément à 2^n

$$C_2(A) = 2^n - A = NOT(A) + 1$$





bit de signe ¦ code en complément à 2
Signe ¦ valeur en décimal

Exemple de représentation sur 3 bits avec 1 bit de signe et 2 bits pour la valeur du nombre

Addition et soustraction de nombres signés

- Pour n bits : $-2^{n-1} \leq val \leq 2^{n-1}-1$

- Dépassement en cas d'addition de nombre signés:
  - Si la somme de 2 nombres négatifs est inférieure à -2^{n-1}
  - Si la somme de 2 nombres positifs est supérieure à $2^{n-1}-1$

- Dépassement en cas de soustraction de nombre signés:
  - Si la soustraction d'un nombre positif à un nombre négatif donne un résultat inférieur à -2^{n-1}
  - Si la soustraction d'un nombre négatif à un nombre positif donne un résultat supérieur à $2^{n-1}-1$

Carry

- Le flag Carry doit être utilisé comme bit d'erreur uniquement pour des opérations avec des nombres non-signés

- Bit de report / emprunt pour addition ou soustraction non-signée
- Permet d'augmenter le nombre de bits des opérations signées ou non signées
- Instructions ARM spéciales avec Carry:
  - ADC <Rd>, <Rm> Rd ← Rd + Rm + C
  - SBC <Rd>, <Rm> Rd ← Rd - Rm - NOT(C)

Overflow

- Le flag Overflow doit être utilisé uniquement pour détecter une erreur pour des opérations avec des nombres signés

- Overflow = 1 si :

1. L'addition de deux nombres positifs (ou la soustraction d'un nombre négatif à un nombre positif) donne un résultat négatif
2. L'addition de deux nombres négatifs (ou la soustraction d'un nombre positif à un nombre négatif) donne un résultat positif

Multiplication

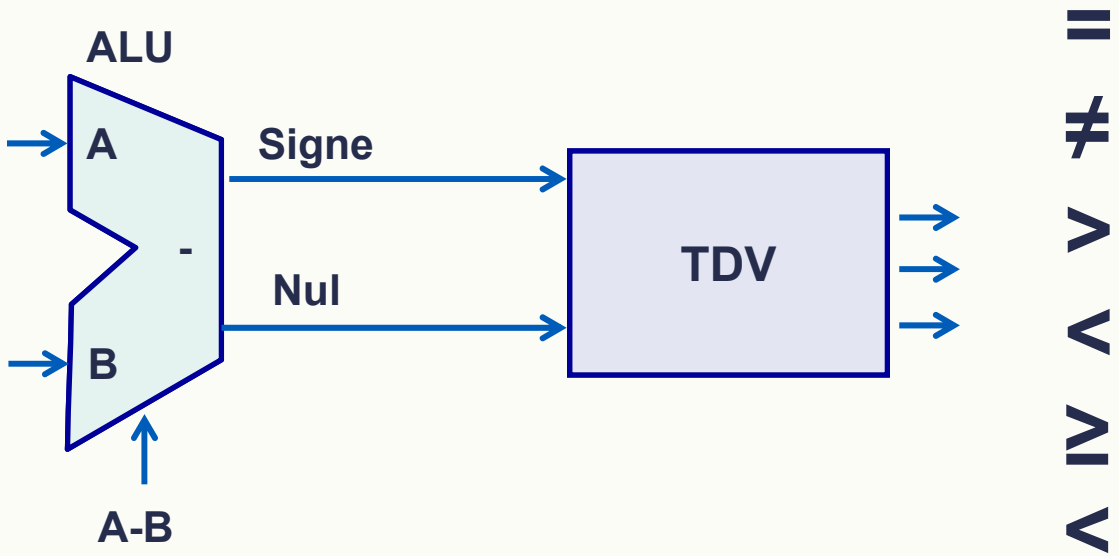
- Le nombre de bits du résultat d'une multiplication est la somme du nombre de bits des opérandes
- Suivant les processeurs, le résultat peut être dans 2 registres (lsb et msb) ou uniquement dans un seule registre (lsb uniquement)
- Les multiplications doivent être précisées : signées ou non-signées

- Instructions multiplication ARM 32 bits

MUL <Rd>, <Rm> , Rd ← (Rm * Rd)[31:0]
SMULL <RdLo>, <RdHi>, <Rm>, <Rs>
RdHi,RdLo ← Rm*Rs

Comparaison

- effectuée par soustraction



Cours AR02

ALU: OPERATIONS EN VIRGULE FLOTTANTE

Virgule flottante : Standard IEEE 754

- **Précision simple:**
  - **nombre de bits: 32**
  - **mantisse sur 23 bits**
  - **exposant sur 8 bits (biais = $2^{8-1} - 1 = 127$)**
- **La valeur d'un nombre est donnée par l'expression:**

$$(-1)^s \times (1 + \sum_{i=1}^{23} m_i 2^{-i}) \times 2^{e - E_{\max}}$$

Virgule flottante : Standard IEEE 754

- **Précision simple normalisée:**
  - 1) déterminer le signe : 0 positif, 1 négatif
  - 2) Exposant: ici le biais est de 127 donc

    Exp= champ_Exposant – 127
  - 3) Mantisse: pour déterminer le nombre il faut

    1.Mantisse le 1 étant implicite dans la représentation
  - 4) donc la valeur réelle sera
    - 1.Mantisse * $2^{exposant}$
  - 5) décalage à gauche ou à droite de la virgule suivant l'exposant
  - 6) Conversion binaire décimal avec le signe

Virgule flottante : Standard IEEE 754

- Précision simple non-normalisée:
  - 1) déterminer le signe : 0 positif, 1 négatif
  - 2) Exposant: ici le biais est de 126 donc
    champs exposant = 0000 0000
  - 3) Mantisse: pour déterminer le nombre il faut
    0.Mantisse le 0 étant implicite dans la représentation
  - 4) donc la valeur réelle sera
    - 0.Mantisse * 2^{-126}
  - 5) décalage à gauche ou à droite de la virgule pour obtenir 1.xxxx * 2^{-z} avec z=-126 - nombre de décalage à gauche
  - 6) Conversion binaire décimal avec le signe ou en puissance de 2

Virgule flottante : Standard IEEE 754

Type	Exposant	Mantisse	Valeur approchée	Écart / préc
Zéro	0000 0000	000 0000 0000 0000 0000 0000	0,0	
Plus petit **nombre dénormalisé**	0000 0000	000 0000 0000 0000 0000 0001	$1{,}4\times10^{-45}$	$1{,}4\times10^{-45}$
Nombre dénormalisé suivant	0000 0000	000 0000 0000 0000 0000 0010	$2{,}8\times10^{-45}$	$1{,}4\times10^{-45}$
Nombre dénormalisé suivant	0000 0000	000 0000 0000 0000 0000 0011	$4{,}2\times10^{-45}$	$1{,}4\times10^{-45}$
Autre nombre dénormalisé	0000 0000	100 0000 0000 0000 0000 0000	$5{,}9\times10^{-39}$	
Plus grand nombre dénormalisé	0000 0000	111 1111 1111 1111 1111 1111	$1{,}17549421\times10^{-38}$	
Plus petit **nombre normalisé**	0000 0001	000 0000 0000 0000 0000 0000	$1{,}17549435\times10^{-38}$	$1{,}4\times10^{-45}$
Nombre normalisé suivant	0000 0001	000 0000 0000 0000 0000 0001	$1{,}17549449\times10^{-38}$	$1{,}4\times10^{-45}$
Presque le double	0000 0001	111 1111 1111 1111 1111 1111	$2{,}35098856\times10^{-38}$	$1{,}4\times10^{-45}$
Nombre normalisé suivant	0000 0010	000 0000 0000 0000 0000 0000	$2{,}35098870\times10^{-38}$	$1{,}4\times10^{-45}$
Nombre normalisé suivant	0000 0010	000 0000 0000 0000 0000 0001	$2{,}35098898\times10^{-38}$	$2{,}8\times10^{-45}$
Presque 1	0111 1110	111 1111 1111 1111 1111 1111	0,99999994	$0{,}6\times10^{-7}$
1	0111 1111	000 0000 0000 0000 0000 0000	1,00000000	
Nombre suivant 1	0111 1111	000 0000 0000 0000 0000 0001	1,00000012	$1{,}2\times10^{-7}$
Presque le plus grand nombre	1111 1110	111 1111 1111 1111 1111 1110	$3{,}40282326\times10^{38}$	
Plus grand nombre normalisé	1111 1110	111 1111 1111 1111 1111 1111	$3{,}40282346\times10^{38}$	2×10^{31}
Infini	1111 1111	000 0000 0000 0000 0000 0000	Infini	
Première valeur (dénormalisée) de ***NaN*** avertisseur	1111 1111	000 0000 0000 0000 0000 0001	*NaN*	
NaN normalisé (avertisseur)	1111 1111	010 0000 0000 0000 0000 0000	*NaN*	
Dernière valeur (dénormalisée) de ***NaN*** avertisseur	1111 1111	011 1111 1111 1111 1111 1111	*NaN*	
Première valeur (dénormalisée) de ***NaN*** silencieux	1111 1111	100 0000 0000 0000 0000 0000	*NaN*	
Dernière valeur (dénormalisée) de ***NaN*** silencieux	1111 1111	111 1111 1111 1111 1111 1111	*NaN*	

Virgule flottante

- Instructions arithmétiques en virgule flottante : addition, soustraction, multiplication, division, racine, ...

- Instructions ARM (exemples)

  Floating-point Addition, Single-precision
  FADDS <Sd>, <Sn>, <Sm>

  Floating-point Divide, Single-precision
- FDIVS <Sd>, <Sn>, <Sm>

Cours AR02

SHIFTER

Shifter

- Logical shift : multiplication ou division par une puissance de 2
  - ajout de 0 à droite ou à gauche
- Arithmetic shift : multiplication ou division par une puissance de 2 sur des nombres signés
  - ajout de 0 à droite, extension de signe à gauche
- Rotation : report des bits retirés d'un coté vers l'autre coté

Shifter : implémentation

- Le shift register n'est pas utilisé (trop lent)
- Utilisation du barrel shifter : un circuit numérique qui permet instantanément les shifts logiques et arithmétiques ou les rotations
- Un multiplexeur par bit avec en entrée tous les bits à shiftés, 0 et 1
- Exemple pour 16 bits : 16 multiplexeurs