Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 & 2

a.05 - Traitement des nombres

Classes T-2/I-2 // 2017-2018

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Daniel Gachet | HEIA-FR/TIC

a.05 | 28.09.2017

Contenu

- Nombres entiers
  - Représentation et codage des nombres entiers
  - Conversion entre différentes bases
  - Cercle des nombres
  - Additions, soustractions et comparaisons
- Nombres réels
  - Représentation et codage des nombres réels
  - Précision des nombres réels
  - Standardisation des nombres réels

Représentation des nombres entiers positifs

- Les nombres entiers positifs N sont également appelés nombres cardinaux (*cardinal numbers*), nombres logiques ou nombres non-signés (*unsigned number*).
- Ils peuvent être représentés dans une base donnée b par une suite de chiffres a_i compris entre 0 et $b-1$, avec

$$N = \sum_{i=0}^{n-1} a_i * b^i \qquad \rightarrow 0 \leq a_i \leq b-1$$

- Par convention le nombre N est représenté par

$$N = a_{n-1}a_{n-2}...a_1a_0 \qquad \rightarrow 0 \leq N \leq b^n - 1$$

où

a_0	chiffre de poids le plus faible *(LSD : Least Significant Digit)*
a_{n-1}	chiffre de poids le plus fort *(MSD : Most Significant Digit)*
n	nombre de chiffres *(Digits)*

Codage des nombres entiers positifs

- Codage décimal

$$N = \sum_{i=0}^{n-1} a_i * 10^i \qquad \rightarrow 0 \leq a_i \leq 9$$

soit pour la valeur 1'386 : $1'386_{10} = 1 * 10^3 + 3 * 10^2 + 8 * 10^1 + 6 * 10^0$

- Codage binaire

$$N = \sum_{i=0}^{n-1} a_i * 2^i \qquad \rightarrow 0 \leq a_i \leq 1$$

soit pour la valeur 1'386 : $1'386_{10} = 101'0110'1010_2 =$
$1*2^{10}+0*2^9+1*2^8+0*2^7+1*2^6+1*2^5+0*2^4+1*2^3+0*2^2+1*2^1+0*2^0$

Codage des nombres entiers positifs (II)

- Codage octal

$$N = \sum_{i=0}^{n-1} a_i * 8^i \qquad \rightarrow 0 \leq a_i \leq 7$$

soit pour la valeur 1'386 : $1'386_{10} = 2'552_8 = 2 * 8^3 + 5 * 8^2 + 5 * 8^1 + 2 * 8^0$

- Codage hexadécimal

$$N = \sum_{i=0}^{n-1} a_i * 16^i \qquad \rightarrow 0 \leq a_i \leq 15$$

soit pour la valeur 1'386 : $1'386_{10} = 56A_{16} = 5x16^2 + 6x16^1 + 10x16^0$

Codage des nombres entiers positifs (III)

- Rapport entre les bases 10, 16, 8 et 2

décimal	hexadécimal	octal	binaire
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Conversions base b → décimal → base b

- Conversion base b → décimal

$$N = N_b = a_{n-1} * b^{n-1} + a_{n-2} * b^{n-2} + \ldots + a_1 * b^1 + a_0 * b^0 = N_{10}$$

par exemple $N = 652_8 = 6 * 8^2 + 5 * 8^1 + 2 * 8^0 = 426_{10}$

- Conversion décimal → base b
  - La conversion d'un nombre décimal N dans son équivalent en base b peut être effectuée par une succession de division.
  - Conversion du nombre décimal 426 en octal

$$\begin{array}{lrcrlr}
N = 426_{10} & & & & & \\
\quad\rightarrow & 426 : 8 & = & 53 & \textit{reste} & 2 \\
\quad\rightarrow & 53 : 8 & = & 6 & \textit{reste} & 5 \\
\quad\rightarrow & 6 : 8 & = & 0 & \textit{reste} & 6 \\
N = 652_8 & & & & &
\end{array}$$

Conversions hexadécimal → binaire → hexadécimal

- Conversion hexadécimal → binaire

$$\begin{array}{lccccccl} N = 8AC9_{16} & = & 8 & A & C & 9 & & \textit{hexadécimal} \\ & & \downarrow & \downarrow & \downarrow & \downarrow & & \\ & = & 1000 & 1010 & 1100 & 1001 & & \textit{binaire} \\ & & & & & & & \\ & = & \multicolumn{4}{l}{1000'1010'1100'1001_2} & & \end{array}$$

- Conversion binaire → hexadécimal *b*

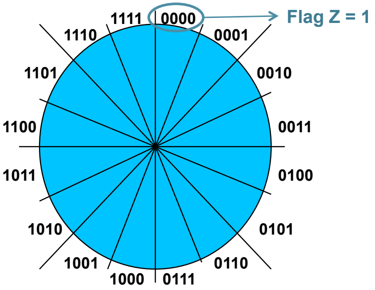
$$\begin{array}{lccccccl} N = 1000'1010'1100'1001_2 & = & 1000 & 1010 & 1100 & 1001 & & \textit{binaire} \\ & & \downarrow & \downarrow & \downarrow & \downarrow & & \\ & = & 8 & A & C & 9 & & \textit{hexadécimal} \\ & & & & & & & \\ & = & \multicolumn{4}{l}{8AC9_{16}} & & \end{array}$$

Cercle des nombres

- Pour un nombre de bits donné, la représentation binaire d'une valeur disposera d'un espace de représentation limité.

  Par exemple 4 bits représente une valeur comprise entre 0 et 15 ($[0...2^4 - 1]$) ou en binaire entre 0000 et 1111

- Représentation des valeurs d'un registre de 4 bits ($n = 4$) dans un cercle



$$Z = \overline{a_3} * \overline{a_2} * \overline{a_1} * \overline{a_0}$$

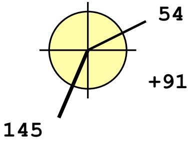
Addition binaire des nombres non-signés

```
ldr     r0, =54
ldr     r1, =91
adds    r2, r0, r1
```

54	:	0011 0110
+ 91	:	0101 1011
145	:	0 1001 0001

Carry (C) = 0

→ Opération dans les limites.

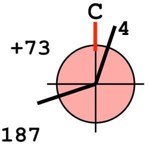



```
ldr     r0, =187
ldr     r1, =73
adds    r2, r0, r1
```

187	:	1011 1011
+ 73	:	0100 1001
4	:	1 0000 0100

Carry (C) = 1

→ Problème de capacité !



[0] **Attention** : Pour des raisons de simplicité, les exemples sont réalisés en mots de 8 bits, mais le µP ARM est une machine à 32 bits et n'est capable d'effectuer ces tests que sur 32 bits.

Soustraction binaire des nombres non-signés

```
ldr     r0, =54
ldr     r1, =45
subs    r2, r0, r1
```

```
ldr     r0, =127
ldr     r1, =128
subs    r2, r0, r1
```

54	:	0011 0110
- 45	:	0010 1101
9	:	0 0000 1001

127	:	0111 1111
- 128	:	1000 0000
255	:	1 1111 1111

Carry (C) = 1

Lors d'une soustraction avec des nombres non-signés, le carry est inversé (C=1)

→ Opération dans les limites.

Carry (C) = 0

Lors d'une soustraction avec des nombres non-signés, le carry est inversé (C=0)

→ Problème de capacité !

[0] **Attention** : Pour des raisons de simplicité, les exemples sont réalisés en mots de 8 bits, mais le µP ARM est une machine à 32 bits et n'est capable d'effectuer ces tests que sur 32 bits.

Soustraction en complément à 1

- L'addition de nombres positifs ne pose aucun problème et peut être réalisée très simplement, d'où l'idée d'utiliser un additionneur pour effectuer la soustraction de deux nombres ($V = A - B$)
- La méthode consiste à passer par un biais R, tel que $R - B$ soit positif

$$\begin{aligned} V &= A - B \\ &= A + \underbrace{(R - B)} - R \\ V &= A + \overline{B} - R \end{aligned}$$

- Pour un nombre binaire de n bits, on choisira

$$R = 2^n - 1 = \sum_{i=0}^{n-1} 2^i$$

Soustraction en complément à 1 (II)

- Le complément à 1 $\overline{B}$ d'un nombre B est défini par $\overline{B} = R - B$

  pour $B = 0101'0110$, alors $\overline{B} = 1111'1111 - 0101'0110 = 1010'1001$
  $\rightarrow$ inversion des bits

- La soustraction de deux nombres revient à

$$\begin{aligned} V &= A + \overline{B} - R \\ &= A + \overline{B} - (2^n - 1) \\ &= A + \overline{B} + 1 - (2^n) \\ V &= A + \overline{B} + 1 \end{aligned}$$

Soustraction en complément à 2

- La méthode du complément à 2 est basée sur le même principe que celui du complément à 1, mais avec un biais R égal à

$$R = 2^n = 1 + \sum_{i=0}^{n-1} 2^i$$

- Le complément à 2 $\dot{B}$ d'un nombre B est défini par $\dot{B} = R - B = \overline{B} + 1$

  pour $B = 0101'0110$, alors $\dot{B} = 1010'1001 + 1 = 1010'1010$

- La soustraction de deux nombres revient à

$$\begin{aligned} V &= A + \dot{B} - R \\ &= A + \dot{B} - (2^n) \\ V &= A + \dot{B} \end{aligned}$$

Soustraction en complément à 2 de deux nombres non-signés

```
ldr     r0, =54
ldr     r1, =45
subs    r2, r0, r1
```

```
ldr     r0, =64
ldr     r1, =100
subs    r2, r0, r1
```

54	:	0011 0110
+ (-45)	:	1101 0011
9	:	1 0000 1001

Carry (C) = 1

→ Opération dans les limites.

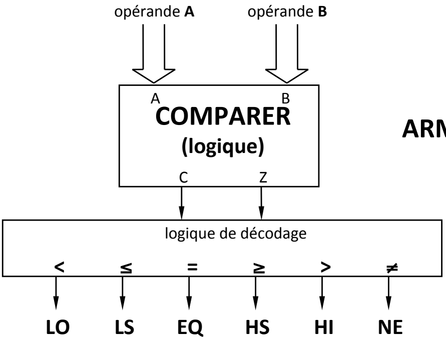
64	:	0100 0000
+ (-100)	:	1001 1100
220	:	**0** 1101 1100

Carry (C) = 0

→ Problème de capacité !

[0] **Attention** : Pour des raisons de simplicité, les exemples sont réalisés en mots de 8 bits, mais le µP ARM est une machine à 32 bits et n'est capable d'effectuer ces tests que sur 32 bits.

Comparaison de nombres non-signés



ARM: CMP <Ra>, <Rb>

comparaison = A - B

LO	(lower)	C == 0	$A < B$	(strictement inférieur)
LS	(lower or same)	C == 0 || Z == 1	$A \leq B$	(inférieur ou égal)
EQ	(equal)	Z == 1	$A = B$	(égal)
NE	(not equal)	Z == 0	$A \neq B$	(différent)
HS	(higher or same)	C == 1	$A \geq B$	(supérieur ou égal)
HI	(higher)	C == 1 && Z == 0	$A > B$	(strictement supérieur)

Représentation des nombres entiers négatifs

- Les nombres entiers N susceptibles d'être positifs ou négatifs sont également appelés nombres arithmétiques ou signés *(signed numbers)*
- Il existe différents types de représentations des nombres signés
  - Représentation par signe-amplitude
  - Représentation biaisée
  - Représentation en complément à 1
  - Représentation en complément à 2

Représentation par signe-amplitude

- Représentation naturelle des nombres négatifs, qui consiste à faire précéder la valeur par un bit de signe, p. ex 0 si le nombre est positif et 1 s'il est négatif, soit

$$\begin{aligned} V &= +125_{10} &= {\color{red}0}111'1101_2 \\ V &= -125_{10} &= {\color{red}1}111'1101_2 \end{aligned}$$

- Pour un format à n bits on peut représenter

$$-(2^{n-1}-1) \leq N \leq (2^{n-1}-1)$$

avec deux représentations du zéro : $+0$ et -0

Représentation biaisée

- Un nombre positif ou négatif N est codé sous forme d'un nombre V tel que

  $$V = N + R$$

- R est un biais positif choisi de telle sorte que V soit toujours positif et que la plage des positifs soit approximativement la même que celle des négatifs, p. ex.

  $R = 2^{n-1} - 1$ ou $R = 2^n - 1$

- La notation biaisée est utilisée pour les exposants des nombres à virgule flottante

Les compléments à 1 et à 2

Le complément à 1

Codage binaire	Valeur
00000000	(+0)
00000001	1
00000010	2
...	...
01111101	125
01111110	126
01111111	127
10000000	-127
10000001	-126
10000010	-125
...	...
11111101	-2
11111110	-1
11111111	(-0)

```
ldr     r0, =25
mvn     r0, r0
```

$$25 = 0001'1001_2$$
$$\rightarrow \quad \overline{25} = 1110'0110_2$$

Le complément à 2

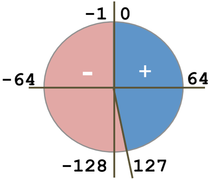
Codage binaire	Valeur
00000000	0
00000001	1
00000010	2
...	...
01111101	125
01111110	126
01111111	127
10000000	-128
10000001	-127
10000010	-126
...	...
11111101	-3
11111110	-2
11111111	-1

```
ldr     r0, =25
rsb     r0, r0, #0
```

$$25 = 0001'1001_2$$
$$\rightarrow \quad \dot{2}5 = 1110'0111_2$$

Fanion N (Negative)

- Le fanion N *(Negative)* est la copie du bit de poids fort
- Il est mis à 1 (set) pour des valeurs allant de $-2^{n-1} \ldots -1$
- Par exemple, pour un registre de 8 bits (-128 à 127), $N = 1$ pour les valeurs de -128 à -1



Par exemple : $-74 = 1011'0110_2$

```
ldr    r0, =-74
cmp    r0, #0
```

Flag N = 1

Fanion V (oVerflow)

- Le fanion V *(oVerflow)* permet de détecter l'incohérence du signe lors d'opérations arithmétiques sur des valeurs signées

```
ldr     r0, =54
ldr     r1, =10
adds    r2, r0, r1
```

```
ldr     r0, =54
ldr     r1, =91
adds    r2, r0, r1
```

54	:	0011 0110
+ 10	:	0000 1010
64	:	0100 0000

54	:	0011 0110
+ 91	:	0101 1011
-111	:	1001 0001

oVerlow (V) = 0
→ résultat cohérent

→ Opération dans les limites.

oVerflow (V) = 1
→ résultat incohérent

→ Problème de capacité !

[0] **Attention** : Pour des raisons de simplicité, les exemples sont réalisés en mots de 8 bits, mais le µP ARM est une machine à 32 bits et n'est capable d'effectuer ces tests que sur 32 bits.

Fanion V (oVerflow) et la soustraction

- Lors de la soustraction d'une valeur positive à une valeur négative, le résultat devrait être toujours négatif

```
ldr     r0, =-10
ldr     r1, =5
subs    r2, r0, r1
```

```
ldr     r0, =-64
ldr     r1, =100
subs    r2, r0, r1
```

-10	:	1111 0110
+ (-5)	:	1111 1011
-15	:	1111 0001

oVerlow (V) = 0
→ résultat cohérent

→ Opération dans les limites.

-64	:	1100 0000
+ (-100)	:	1001 1100
92	:	0101 1100

oVerflow (V) = 1
→ résultat incohérent

→ Problème de capacité !

[0] **Attention** : Pour des raisons de simplicité, les exemples sont réalisés en mots de 8 bits, mais le µP ARM est une machine à 32 bits et n'est capable d'effectuer ces tests que sur 32 bits.

Fanion V (oVerflow) et les cas spéciaux

- Dans les cas ci-dessous, le fanion V est toujours mis à zéro (clear)
  - Addition d'une valeur positive et négative

```
ldr     r0, =10
ldr     r1, =-5
adds    r2, r0, r1
```

```
ldr     r0, =-10
ldr     r1, =12
subs    r2, r0, r1
```

  - Soustraction de deux valeurs positive

```
ldr     r0, =10
ldr     r1, =12
subs    r2, r0, r1
```

Fanion V (oVerflow) et les cas spéciaux (II)

- Il y a oVerflow si
  - La somme de deux nombres positifs est négative
  - La somme de deux nombres négatifs est positive
  - La soustraction d'un nombre positif à un négatif donne un résultat positif
  - La soustraction d'un nombre négatif à un positif donne un résultat négatif
- $V = 1$, si

  $(+) + (+) \rightarrow (-)$

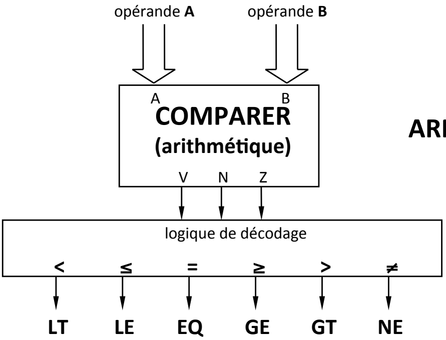
  $(-) + (-) \rightarrow (+)$

  $(+) - (-) \rightarrow (-)$

  $(-) - (+) \rightarrow (+)$

- Pour $c = a + b$, $V = (\overline{a_s} \& \overline{b_s} \& c_s) | (a_s \& b_s \& \overline{c_s})$

Comparaison de nombres signés



ARM: CMP <Ra>, <Rb>

comparaison = A - B

LT	(lower than)	(V $\oplus$ N) == 1	$A < B$	(strictement inférieur)
LE	(lower or equal)	(V $\oplus$ N) == 1 || Z == 1	$A \leq B$	(inférieur ou égal)
EQ	(equal)	Z == 1	$A = B$	(égal)
NE	(not equal)	Z == 0	$A \neq B$	(différent)
GE	(greater or equal)	(V $\oplus$ N) == 0	$A \geq B$	(supérieur ou égal)
GT	(greater than)	(V $\oplus$ N) == 0 && Z == 0	$A > B$	(strictement supérieur)

Représentation des nombres réels

- Les nombres réels N peuvent être représentés dans une base b par une suite infinie de chiffres a_i compris entre 0 et $b-1$, avec

$$N = \sum_{i=-\infty}^{n-1} a_i * b^i$$

- La partie entière N_e et la partie fractionnaire N_f sont définies respectivement par

$$N_e = \sum_{i=0}^{n-1} a_i * b^i \qquad N_f = \sum_{i=-\infty}^{-1} a_i * b^i$$

Représentation des nombres réels (II)

- De la même manière que les nombres entiers, les nombres réels sont également représentés par une suite de chiffres ordonnée de gauche à droite par ordre de poids décroissant et une virgule pour séparer la partie entière de la partie fractionnaire.

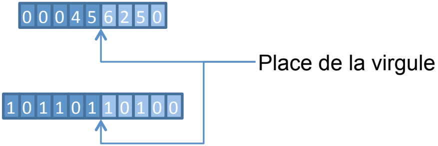
- En informatique on peut faire appel à deux représentations différentes
  - Représentation en virgule fixe
  - Représentation en virgule flottante

Représentation en virgule fixe

- En représentation en virgule fixe *(fixed point)*, les nombres sont traités sous forme de mots comportant un nombre fixe de chiffres pour la partie entière et la partie fractionnaire. Par exemple

$$N_{10} = 45.625$$

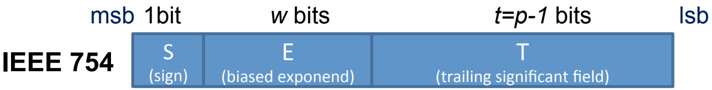
$$N_2 = 10'1101.10100$$



- Exercice
  - Représenter la valeur 456.3467 en binaire

Représentation des nombres à virgules flottantes

- Les nombres réelles N en virgules flottantes *(floating point)* sont représentées sous la forme suivante



$$N = (-1)^{S} * 2^{E-bias} * 1,T$$

- S : signe de la valeur
  - 0 → « + »
  - 1 → « - »
- E : exposant
  - valeur + biais
- T : mantisse
  - 1,fraction

Représentation des nombres à virgules flottantes (II)

- Quelques indications importantes sur la représentation des nombres réelles à virgules flottantes
  - Nombres normalisés

Biais de l'exposant	$2^{w-1} - 1$	*(si w=8 → biais=127)*
Portée de l'exposant	$0 < E < 2^w - 1$	*(si w=8 → 0 < E < 255)*
Portée de la fraction	zéro et non zéro	

  - Nombres infinis

Signe	$+, -$	
Exposant	$2^w - 1$	*(si w=8 → E=255)*
Mantisse	zéro	

  - NAN (Not a Number)

Signe	pas considéré	
Exposant	$2^w - 1$	*(si w=8 → E=255)*
Mantisse	non zéro	

Un exemple...

- Donnée

$$N_{10} = 45.625$$
$$N_2 = 101101,101 * 2^0$$
$$N_2 = 1,01101101 * 2^5$$

- Codage sur 32 bits avec un biais de 127(8 bits)

S = 0
E = 5 + 127 = 132 → 1000'0100
T = 01101101 0...0

- Résultat

$N_2 =$ 0 100'0010'0 011'0110'1000'0000'0000'0000
$N_{16} =$ 0x4236'8000

Précision des nombres réels

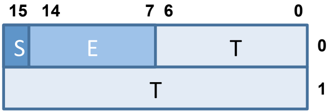
- La représentation des nombres réels pose un problème de précision, car il n'est pas possible de représenter la partie fractionnaire par une série infinie de chiffres. Celle-ci est donc limitée à u chiffres.

$$N_f \to \widetilde{N_f} = \sum_{i=-u}^{n-1} a_i * b^i$$

- Cette troncature produit une erreur inférieur à b^{-u}
- En pratique, on remplace la troncature par un arrondi qui consiste à choisir le nombre fractionnaire le plus proche de N_f par excès ou par défaut. Ceci nous donne une erreur d'arrondi comprise entre $-b^{-u}/2$ et $+b^{-u}/2$

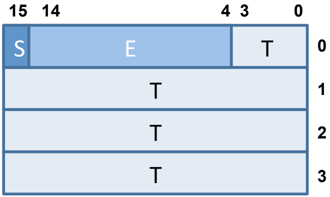
Nombres standards en virgules flottantes selon IEEE 754

- Format en simple précision sur 32 bits *(float)*



- Simple précision sur 32 bits
  - 1 bit de signe
  - 8 bits d'exposant
  - 23 bits de mantisse

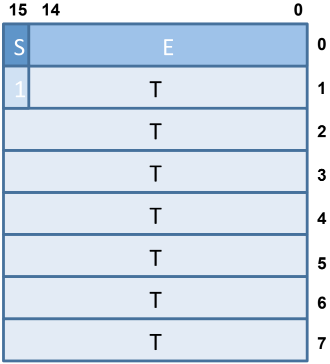
- Format en double précision sur 64 bits *(double)*



- Double précision sur 64 bits
  - 1 bit de signe
  - 11 bits d'exposant
  - 52 bits de mantisse

Nombres standards en virgules flottantes selon IEEE 754 (II)

- Format en quadruple précision sur 128 bits *(long double)*



- Quadruple précision sur 128 bits
  - 1 bit de signe
  - 15 bits d'exposant
  - 1 bit de partie entière
  - 111 bits de mantisse