ELEC-H-305

Circuits logiques et numériques

2011-2012

Cours 1

Dragomir Milojevic

Dragomir.Milojevic@ulb.ac.be

Informations générales

ELEC-H-305 — Calendrier

Cours — 2 ECTS = 12 séances, 2h/séance
Lundi de 14.00 à 16.00 (UB4.136)

TPs — 2 ECTS = 12 séances (11 exercices+1Q&R), 2h/séance

Lundi de 16.00 à 18.00 (S.AW1.125)

Récupération: ???

Informations générales

ELEC-H-305 — Equipe

- **Dragomir Milojevic**
  BEAMS - dragomir.milojevic@ulb.ac.be (02.650.30.60)
- **Yannick Allard**
  BEAMS - yannick.allard@ulb.ac.be

Pour toute question (remarque), veuillez vous adresser à:

Dragomir Milojevic

et/ou

Yannick Allard

Informations générales

ELEC-H-305 — Sur le WEB

Adresse:

http://beams.ulb.ac.be/beams/teaching/elec-h-305.html

- Les notes de cours (celles de l'année passée + MAJ — en principe mineurs — de cette année), les énoncés des exercices ainsi que les corrigés (succincts) y seront déposés.
- Les infos concernant l'examen, séance Q&R
- Pour les TPs il faut arriver avec vos feuilles d'exercices (à organiser avec les délégués).

Informations générales

Conseils

- Travailler régulièrement
- Travailler dans l'ordre - la matière est fortement lié (faire le Cours/ TP 3 sans faire le 2 n'a pas trop de sens ...)
- Le contenu du cours peut être vu comme algorithmique, mais attention - en comprenant ce que vous faites:
  - ✤ Vous diminuez le risque d'erreur à l'examen
  - ✤ Vous pouvez résoudre les exercices qui ne ressemblent pas aux "exercices type" (l'examen contiendra de tels exercices)
  - ✤ Vous construisez un capital savoir important pour la suite (le cours de micro-électronique en 4ème p.e. sans parler d'après...)

Informations générales

Littérature conseillée

- Livres:
  - *Daniel D. Gajski, "Principales of Digital Design", Prentice Hall*
  - *Sajan G. Shiva, "Introduction to Logic Design"*
  - On peut les acheter aux PUB (mais ce n'est pas obligatoire)
- Bibliothèque
- Internet, mais vérifier les sources...

Informations générales

ELEC-H-305 — Examen

- Examen: écrit
- Durée: 4 heures
- Utilisation de la calculatrice (ou équivalent) est interdite
- Porte sur **toute** la matière vue au cours et aux TPs
- Ce n'est pas un examen difficile à condition de travailler régulièrement et de comprendre le contenu (la présence au cours et TPs peut aider...)
- Il faut travailler vite et sans commettre des erreurs
- Avant l'examen une séance de Questions & Réponses (Q&R) sera organisée (attention ceci n'est pas un 2ème cours - on n'explique pas la théorie mais on aide à comprendre les exercices)

But de cours

- Acquérir les “**maths**” nécessaires pour la conception des circuits logiques (systèmes de numérotation, algèbre de Boole, optimisation des expressions logiques ...)

- Apprendre à **formaliser** les circuits combinatoires et les circuits séquentiels asynchrones et synchrones
(Attention! Les circuits séquentiels posent généralement des problèmes)

- Apprendre la **synthèse manuelle** des circuits logiques:
Construire un circuit à partir d’un cahier de charge verbal - ce sera donc une “science appliquée”...

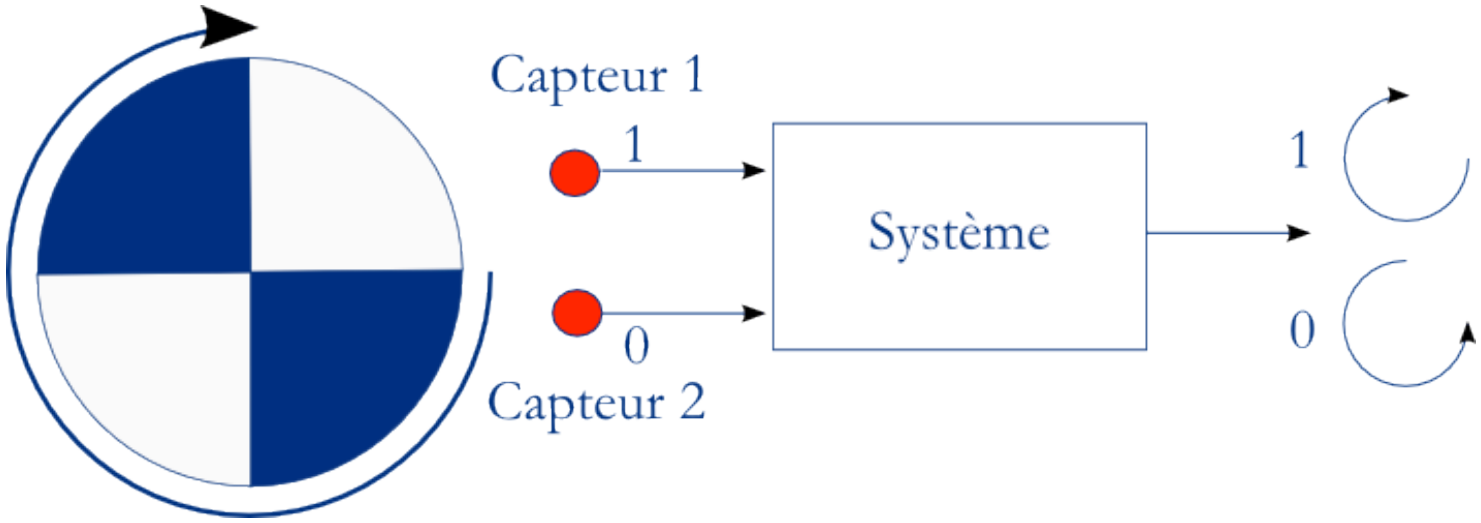
Contenu du cours

Table de matières

Objectif: exemple

Exemple de cahier de charges

On souhaite construire un détecteur de sens de rotation à partir de deux capteurs lumineux et une roue colorié:



Vous allez apprendre comment concevoir la boite noire permettant de dire dans quel sens tourne la roue.

Démarche

- On parle de:

  Conception, réalisation, synthèse, …

  d'un **circuit logique** à partir de la **spécification**

- En fait il s'agit de :
  - la **Modélisation** (processus de conception) et
  - la **Réalisation physique** (implémentation)
- Les modèles des circuits logiques sont d'abord construits et vérifiés à l'aide des fonctions logiques et puis traduits en portes logiques (afin d'éviter les erreurs).

Les circuits et les modèles

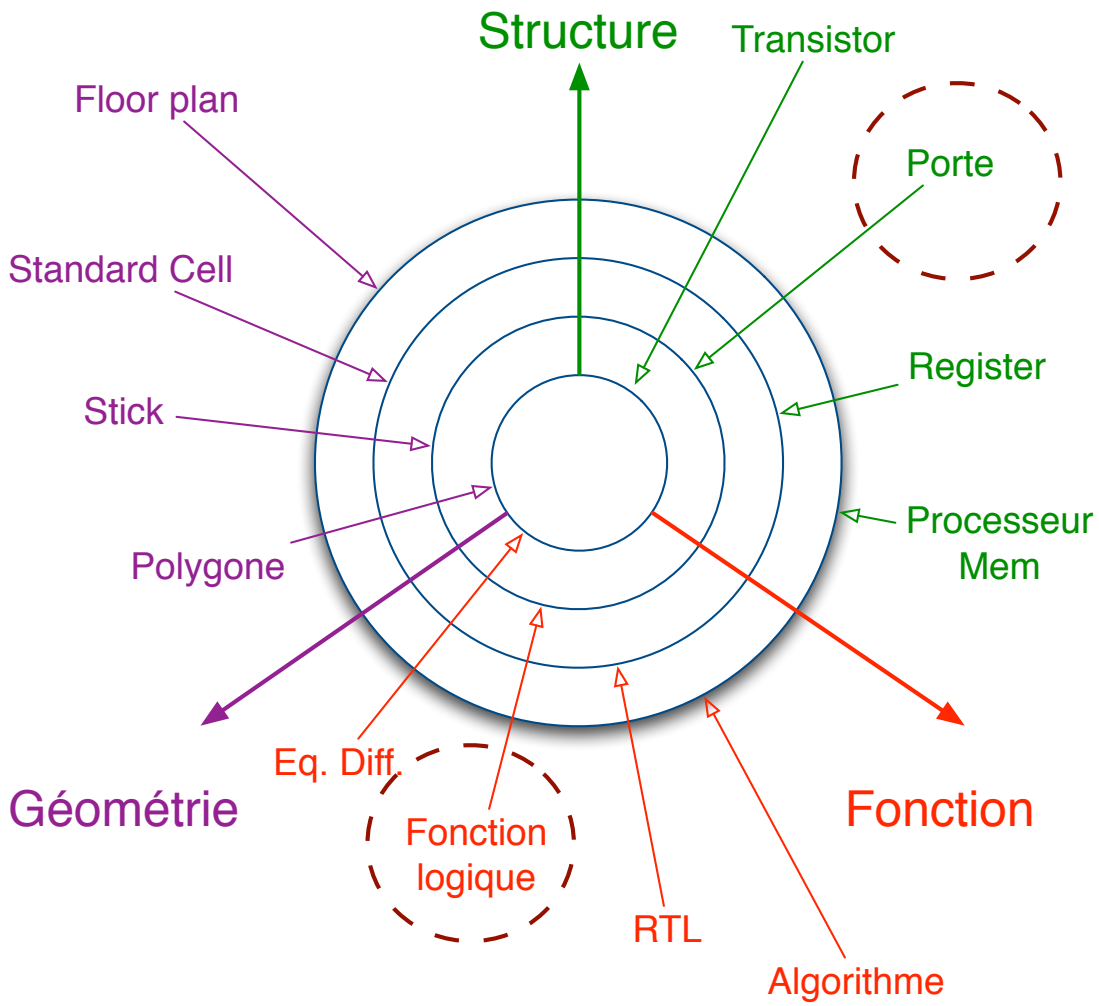
Y-chart de Gajski

Modèles de circuit :

I. A différents niveaux d'abstraction — cercles concentriques

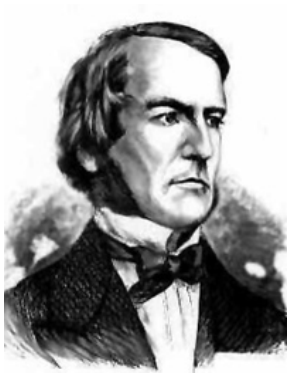
et

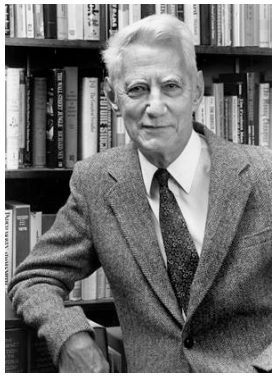
II. Selon trois axes:
- Géométrie
- Fonction
- Structure



Nous allons apprendre à modéliser les circuits à l'aide des fonctions logiques (portes logiques).

Quelques repères historiques




1850 1936 1937 1945

Aristote:
(384 – 322 BC)
Logique de premier ordre,
Syllogismes, ET, OU, NON,...

Boole:
(1815 – 1864)
Algèbre de Boole

Turing:
(1912 – 1954)
Machine universelle : peut
calculer tout ce qui est calculable.

Shannon:
(1916 – 2001)
Fondement des circuits numériques
Matérialisation de l'algèbre de Boole

Von Neumann:
(1903 – 1857)
Architecture pratique d'une
machine universelle

Aujourd'hui

1. Représentation des nombres
2. Techniques de conversion
3. Conversion entre les différents bases
4. Bases utiles
5. Opérations arithmétiques
6. Nombres négatifs
7. Virgule flottante

1. Représentation des nombres

Décimal en virgule fixe, base 10 (nombre réel):

$$(1372.6450)_{10}$$

1 3 7 2	.	6 4 5 0
Partie entière	Point décimal	**Partie fractionnaire**
10^3 10^2 10^1 10^0		10^{-1} 10^{-2} 10^{-3} 10^{-4}

$1x10^3+3x10^2+7x10^1+2x10^0$	+	$6x10^{-1}+4x10^{-2}+5x10^{-3}$

1. Représentation des nombres

Forme généralisée (n'importe quelle base)

- Nombre *N* en base *r*, noté N_r
- *n+1* chiffres, de $a_0, \ldots, a_n$ pour la **Partie entière** (index *i*)
- *m* chiffres, de $b_1, \ldots, b_m$ pour la **Partie fractionnaire** (index *j*)

Ceci est une suite des chiffres!

$$N=(a_n a_{n-1} \ldots a_0 . b_1 b_2 \ldots b_m)_r$$

$$N=a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \ldots + a_0 \cdot r^0 + b_1 \cdot r^{-1} + b_2 \cdot r^{-2} + \ldots + b_m \cdot r^{-m}$$

$$N=\sum_{i=0}^{i=n} a_i \cdot r^i + \sum_{j=1}^{j=m} b_j \cdot r^{-j}$$

Partie entière **Partie fractionnaire**

1. Représentation des nombres

Dans:

$$N = \sum_{i=0}^{i=n} a_i \cdot r^i + \sum_{j=1}^{j=m} b_j \cdot r^{-j}$$

Indexes *i, j* ➟ **les poids**

- Pour la partie entière on distingue:
  - *i=n* — **bit de poids plus fort** (*Most Significant Bit* - MSB)
  - *i=0* — **bit de poids plus faible** (*Least Significant Bit* - LSB)

1. Représentation des nombres

- **Base utiles**:
  - `r=10` - décimal {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - `r= 2` - binaire {0, 1}
  - `r= 8` - octal {0, 1, 2, 3, 4, 5, 6, 7,}
  - `r=16` - hexadécimal {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

- **Important** : Pouvoir **passer** d'une base à l'autre — les conversions et les calculs arithmétiques.

- Conversions entre les bases utiles 2, 8 et 16 : c'est très simple! Pour les autres, il faut un peu d'exercice.

1. Représentation des nombres

Décimal	Binaire (2)	Octal (8)	Hexadécimal (16)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

1. Représentation des nombres

A. Décimal vers binaire/Partie entière — Principe

Nombre A, en base *r=2* , codé sur 4 chiffres, écrit sous forme:

$$A = \sum_{i=0}^{i=3} a_i \cdot r^i$$

$$A = a_0 \cdot 1 + a_1 \cdot 2 + a_2 \cdot 2 \cdot 2 + a_3 \cdot 2 \cdot 2 \cdot 2$$

$$A = 2 \cdot (2 \cdot (2 \cdot (a_3) + a_2) + a_1) + a_0$$

Reste de la division

(A - a_0) : 2

(((A - a_0) : 2) - a_1) : 2)

...

Méthode des divisions successives

2. Techniques de conversion

A. Décimal vers binaire/Partie entière — Exemple

$(245)_{10}=(?)_2$

Méthode de divisions successives:

Nombre à convertir	Base	Reste de la division
245	:2	1
122	:2	0
61	:2	1
30	:2	0
15	:2	1
7	:2	1
3	:2	1
1	:2	1
0		

Sens de calcul

Sens de lecture (MSB -> LSB)

245:2=122 reste 1

122:2= 61 reste 0

...

$(245)_{10}=(11110101)_2$

MSB LSB

2. Techniques de conversion

B. Décimal vers binaire/Partie fractionnaire — Exemple

$(0.345)_{10}=(?)_2$

Méthode de multiplications successives:

.345	x2	0.690	0
.690	x2	1.380	1
.380	x2	0.760	0
.760	x2	1.520	1
.520	x2	1.040	1
.040	x2	0.080	0
.080	x2	0.160	0
.160	x2	0.320	0
.320	x2	0.640	0
.640	x2	1.280	1
.280	x2	0.560	0

Sens de calcul

Sens de lecture

$(0.345)_{10}=$
$(.0101100001...)_2$

Condition d'arrêt:
En fonction de la précision voulue.

2. Techniques de conversion

C. Binaire vers décimal/Partie entière — Exemple

$(11110101)_2 = (?)_{10}$

$= 1x2^0+0x2^1+1x2^2+0x2^3+1x2^4+1x2^5+1x2^6+1x2^7$

$= 1+0+4+0+16+32+64+128$

$= 245$

Binaire vers décimal/Partie fractionnaire

$(.0101100001..)_2 = (?)_{10}$

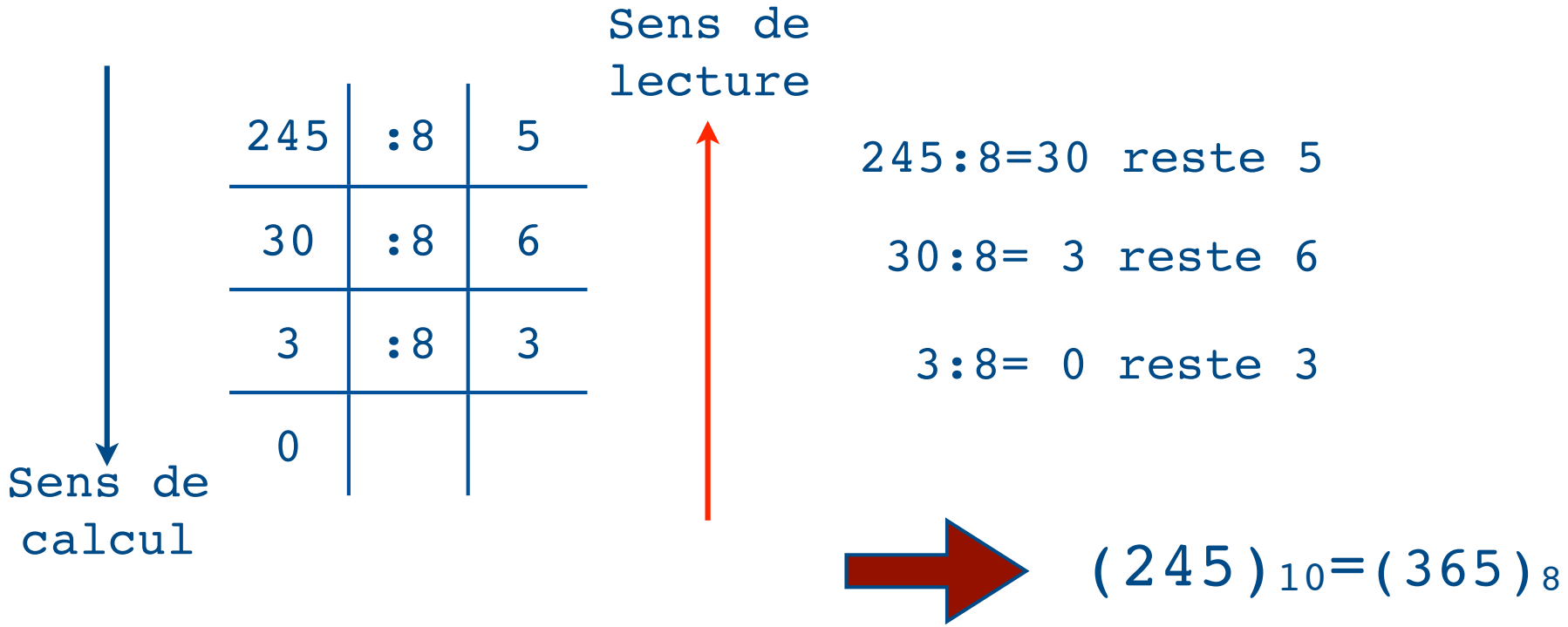
$= 0x2^{-1}+1x2^{-2}+0x2^{-3}+1x2^{-4}+1x2^{-5}+0x2^{-6}+0x2^{-7}+0x2^{-8}+...$

$= 0x1/2+1x1/4+0x1/8+1x1/16+1x1/32+0x1/64+...$

$= (8+2+1)/32 = 11/32 = 0,34375$

2. Techniques de conversion

D. Décimal vers base 8 (octal)



Octal vers décimal:

$$(365)_8 = 5x8^0+6x8^1+3x8^2$$
$$= 5+48+192$$
$$= (245)_{10}$$

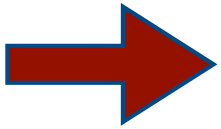
2. Techniques de conversion

E. Décimal vers base 16

245	:16	5
15	:16	F
0		

245:16 = 15 reste 5

15:16 = 0 reste F



$(245)_{10}=(F5)_{16}$

Base 16 vers décimal

$$(A1F.1C)_{16} = (?)_{10}$$
$$= A*16^2+1*16^1+F*16^0+1*16^{-1}+C*16^{-2}$$
$$= 10*16^2+1*16^1+15*16^0+1*16^{-1}+12*16^{-2}$$
$$= 2560+16+15+28/256$$
$$= 2591.1093...$$

3. Conversions généralisées

Problème de conversion

Nombre N à la base p en nombre ? à la base q

$$(N)_p \quad \dashrightarrow \quad (?)_q$$

Comment faire?

En passant par la base 10 :

$$(N)_p \quad \dashrightarrow \quad (N)_{10} \quad \dashrightarrow \quad (?)_q$$

Pour les bases utiles, passer d'abord en base 2.

3. Conversions

Exemple complet de conversion: n'importe quelle base

$(25.34)_8$ ➟ $(?)_5$

$(25.34)_8$ ➟ $(?)_{10} = 2x8+5x1+3x8^{-1}+4x8^{-2}=...= (21.4375)_{10}$

$(21.4375)_{10}$ ➟ $(?)_5$

21	:5	1
4	:5	4
0		

.4375	x5	2.1875	2
.1875	x5	0.9375	0
.9375	x5	4.6875	4
.6875	x5	3.4375	3
.4375	x5	2.1875	2
.1875	x5	0.9375	0

$(21.4375)_{10}$ ➟ $(41.2043220432.....)_5$

4. Bases utiles — Conversions

Dans des bases utiles 2, 8 et 16:
regroupement (ou expansion) des 1 et des 0 (en partant de LSB)

Groupes de 3 - base 8

Base	Nombre		
10	245		
2	11110101		
Par 3	11	110	101
8	3	6	5
Par 4		1111	0101
16		F	5

Groupes de 4 - base 16

Base	Nombre		
16	1F2		
2	0001 | 1111 | 0010		
Par 3	111	110	010
8	7	6	2

4. Bases utiles

Exemple complet de conversions

$(378)_{10}$ ➟ $(0101111010)_2$ ➟ $(?)_8$, $(?)_{16}$

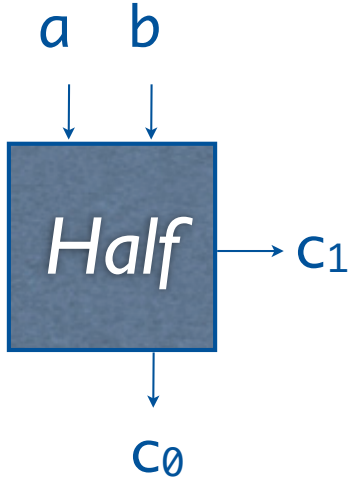
	101111010		
Groupement Par 3	101	111	010
Base 8	**5**	**7**	**2**
Groupement Par 4	0001	0111	1010
Base 16	**1**	**7**	**A**

5. Opérations arithmétiques

Addition (binaire) de deux mots a et b d'un bit:

$$c = a + b$$

Résultat, mot c :
mot de 2 bits $c_1 c_0$



a	b	c_1	c_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Report (Carry)

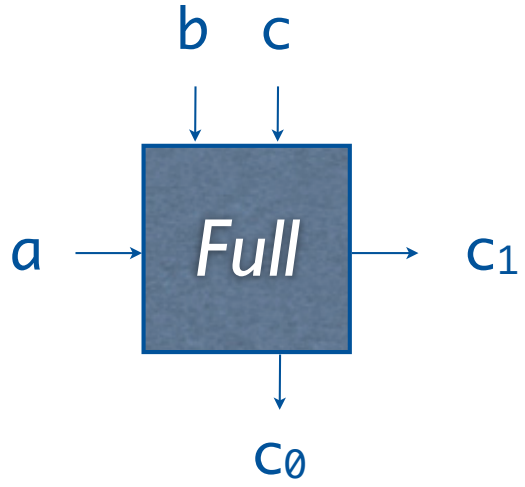
➟ **1/2 additionneur (*Half-adder*)**

5. Opérations arithmétiques

Addition de trois mots a, b, c d'un bit:

$$d = a + b + c$$

Résultat d : mot de 2 bits $c_1 c_0$



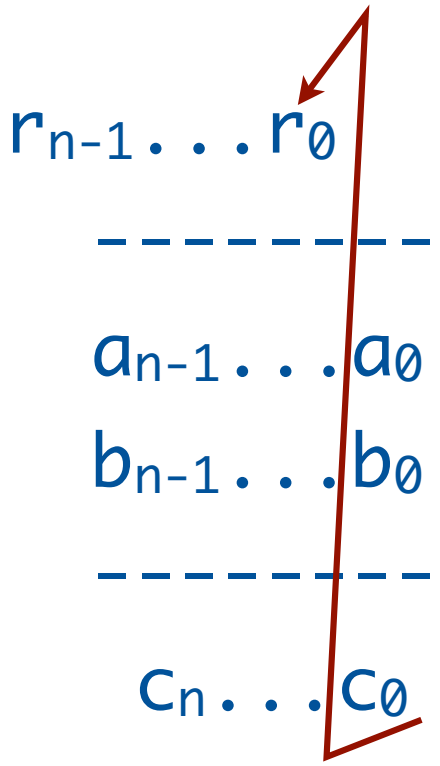
a	b	c	d_1	d_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

➟ **Additionneur complet (*Full adder*)**

5. Opérations arithmétiques

Addition de deux mots a et b de n bits:

$c=a+b$



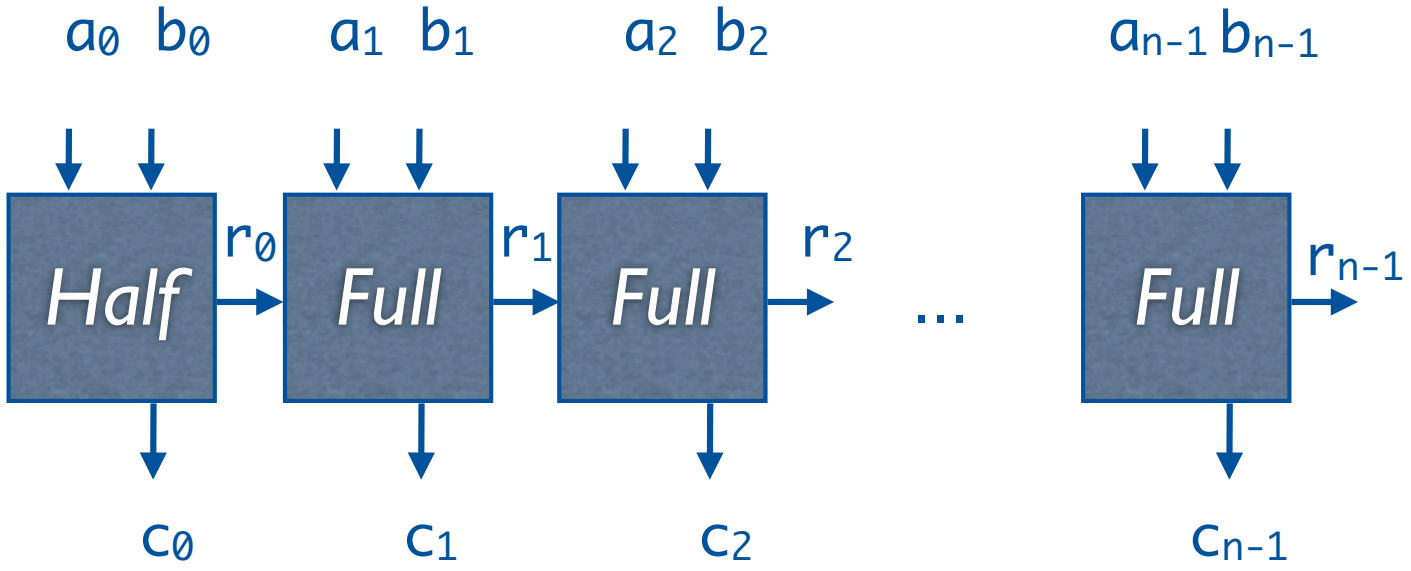
Résultat c : un mot de $n+1$ bits, obtenu en effectuant n sommes bit à bit

5. Opérations arithmétiques

Addition de deux mots *a* et b de n bits :

A l'aide de n — circuits

- un 1/2 additionneur pour le premier bit (bit LSB)
- n-1 additionneurs complets pour les autres bits



5. Opérations arithmétiques

Exemple

c = a + b = 236 + 170 = ?

		128	64	32	16	8	4	2	1
Bit		7	6	5	4	3	2	1	0
Report	1	1	1	0	1	0	0	0	
a		1	1	1	0	1	1	0	0
b		1	0	1	0	1	0	1	0
c	1	1	0	0	1	0	1	1	0

Vérification:
$236+170=(436)_{10}$
$(436)_{10}$ = 1 1001 0110

Problème de débordement (*overflow*) — en rouge : le résultat sera **tronqué** si le mot de résultat est codé sur 8 bits

5. Opérations arithmétiques

Soustraction de deux mots a et b d'un bit:

$$c = a - b$$

Lorsque 0 — 1, alors il faut emprunter
(marqué en rouge)

a_1	a_0	b	c_0
0	0	0	0
1	0	1	1
0	1	0	1
0	1	1	0

Emprunt
Borrow

5. Opérations arithmétiques

Exemple: 236 - 170 = ?

		128	64	32	16	8	4	2	1
Bit		7	6	5	4	3	2	1	0
Emprunt							1		
a		1	1	1	0	1	~~1~~ 0	0	0
b		1	0	1	0	1	0	1	0
c		0	1	0	0	0	0	1	0

Vérification:
$236-170=(66)_{10}$
$(66)_{10}$ = 0100 0010

5. Opérations arithmétiques

Multiplication de deux mots a et b de n bits:

$$c=a*b$$

Additions successives:

```
101*111 =  101
           101
          101
         ------
         100011
```

```
Vérification:
5*7 = 35
```

Attention au report!

Même algorithme pour n'importe quel autre base.

5. Opérations arithmétiques

Division de deux mots *a* et *b* de *n* bits:

$$c = a : b$$

Soustractions successives:

```
110101:111 = 111   ➟  quotient
 111
 ---
 1100
  111
  ---
  1011
   111
   ---
   100             ➟  reste
```

```
Vérification:
53:7 =  7  reste  4
     = 111 reste 100
```

Même algorithme pour n'importe quel autre base.

6. Nombres négatifs

A. Signe et Valeur Absolue (SVA)

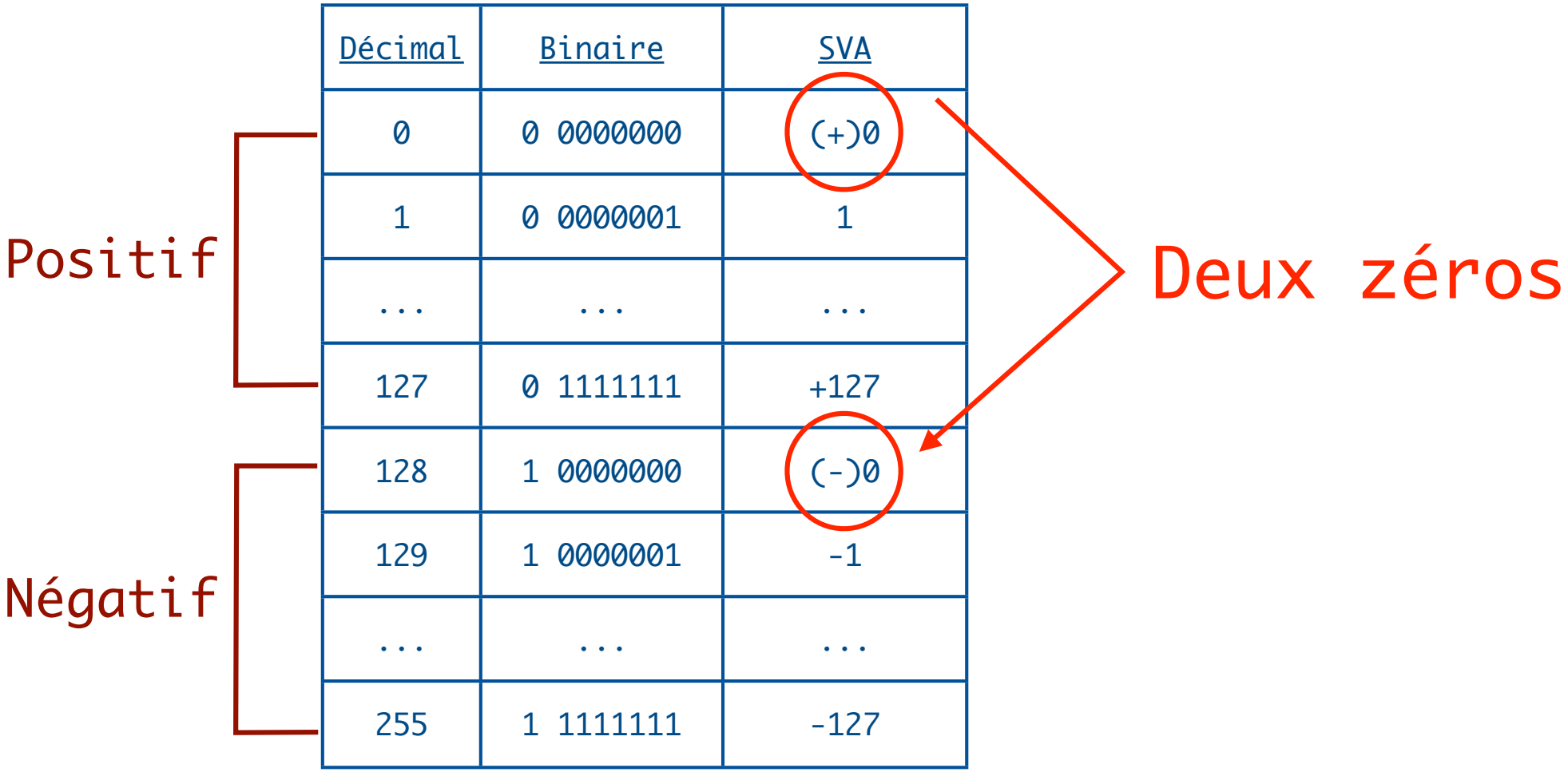
- Par convention:
  - `0` – positif
  - `1` – négatif
- Pour un mot de 8 bits:
  - 1 bit réservé pour le signe et
  - 7 bits réservés pour la valeur absolue.
- On peut donc représenter des valeurs de: `-127` à `+127` car:

```
1 1111111 (-127)        0 1111111 (+127)
```

6. Nombres négatifs

A. Signe et Valeur Absolue (SVA)

Problème de deux zéros



6. Nombres négatifs

A. Signe et Valeur Absolue (SVA)

Opérations arithmétiques

- Comment faire des opérations arithmétiques sur deux mots A et B en SVA ?
  - Comparaison de signe pour déterminer le signe de résultat
  - Comparaison de magnitude pour déterminer le sens de l'opération A—B:
    - si A > B, alors A — B
    - si A < B, alors B — A
  - Peu pratique pour une réalisation matérielle...

6. Nombres négatifs

B. Complément à la base (n'importe quel base!)

Motivation — réaliser la soustraction comme une addition.

Considérons deux mots A et B, en base r, codés sur m chiffres.

$$A-B = A + (-B)$$
$$= A + (r^m - B) = A + \overline{B}$$

Complément à la base 10

(radix complement) — $B+\overline{B}=r^m$

Exemple: $r=10$, $m=3$, $r^m=1000$

$+87 \Rightarrow 87 \Rightarrow 87$

$-53 \Rightarrow 1000-53=947 \Rightarrow + 947$

1034

6. Nombres négatifs

B. Complément à la base (pour n'importe quel base!)

Pour effectuer la soustraction A−B il faut réaliser l'opération:

$\overline{B} = (r^m - B)$ — Il faut une soustraction ...

En réorganisant:

$\overline{B} = (r^m - B) = ((r^m - 1) - B) + 1$

$(r^m - 1) - B$ est un **complément** de chaque chiffre de B

m chiffres de r-1 p.e. m=3, r=10 -> 1000-1=999

$$(r^m - 1) - B = ((r-1)(r-1)\ldots(r-1)) - (b_{m-1}b_{m-2}\ldots b_0)$$

pas une multiplication, mais concaténation

$$= ((r-1)-b_{m-1})((r-1)-b_{m-2})\ldots((r-1)-b_0)$$

$$= b'_{m-1}b'_{m-2}\ldots b'_0$$

Le **complément** de chaque chiffre — en binaire très pratique: chaque chiffre est tout simplement inversé.

6. Nombres négatifs

B. Complément à la base: Cas binaire

Deux variantes de complément:

$(r^m-1)-B$ **est Complément à 1**

(mais attention on a toujours deux zéros: 00000000 et 11111111)

Déc.	Binaire	C1
0	0	(+)0
1	1	1
...	...	...
127	1111111	+127
128	10000000	-127
129	10000001	-126
...	...	...
255	11111111	(-)0

$(r^m-1)-B + 1$ **est Complément à 2**

introduit un **biais**, un seul zéro...

Déc.	Binaire	C2
0	0	0
1	1	1
...	...	...
127	1111111	+127
128	10000000	-128
129	10000001	-127
...	...	...
255	11111111	-1

6. Nombres négatifs

C. Opérations arithmétiques (addition, soustraction)

- En **Complément à 1 (C1)**
  Possible, mais il existe beaucoup de cas particuliers qui doivent être analysés (ici on en parle à titre informatif).

- En **Complément à 2 (C2)**
  - ★ C'est direct : lorsque le nombre est négatif — conversion (soustraction = addition)
  - ★ Une seule exception : gestion de débordement ... et encore c'est très simple. Règle: Le bit d' *overflow* est ignoré tant que le résultat de l'opération $c=a-b$ est dans l'intervalle:

$$-r^m < c < r^m-1$$

pour 8 bits: $-2^7 < c < 2^7-1$ $\quad -128 < c < 127$

(exemple d'*overflow*: 01000000+01000000=10000000)

6. Nombres négatifs

B. Complément à la base (C1, C2)

- Conversion
  - La valeur absolue d'un nombre négatif est représentée en binaire et complétée avec des '0' pour avoir la longueur de *m* bits
  - Chaque bit du mot ainsi obtenu est inversé, **Complément à 1 - C1**
  - On additionne un '1' au C1 pour avoir le **Complément à 2 - C2**
- Propriétés de **C2**
  - Pour des mots codés sur 8 bits on peut représenter de -128 à 127
  - Un seul zéro : 00000000
  - Opération arithmétiques beaucoup plus simples ...

6. Nombres négatifs

Exemples de représentation des nombres négatifs

-23 sur 8 bits: valeur absolue $(23)_{10}=(10111)_2$

Nombre négatif: conversion en binaire, inversion (C1), puis +1 (C2).

SVA	C1	C2
1 001 0111	0001 0111	
	1110 1000	1110 1001

25 sur 8 bits: $(25)_{10}=(11001)_2$

Nombre positif: On ne fait RIEN !!!

SVA	C1	C2
0 001 1001	0001 1001	0001 1001

6. Nombres négatifs

Exemple de soustraction en C2

57 - 23 = ?

-23 en C2 sur 8 bits : $(23)_{10}=(10111)_2$ C2: (1110 1001)

57 en C2 sur 8 bits : $(57)_{10}=(111001)_2$ C2: (0011 1001)

```
 1 1111   1
    1110 1001
    0011 1001
-------------
  1 0010 0010   ➟   (34)10
```

Règle: On examine les deux derniers bits de report:

* soit les mêmes (00 ou 11) — résultat OK

* soit différents (01 ou 10) — alors débordement

Ici on ignore *overflow* car: les deux bits de report sont les mêmes et le résultat -128 < 34 < 127

7. Virgule flottante

Avec le calcul en virgule fixe à 32 ou 64 bits nous ne pouvons pas faire des calculs scientifiques
(limitation due à la taille de l'ALU — généralement de 32, 64 bits).

Format général d'un nombre en virgule flottante:

$$N = mantisse \cdot (base)^{exposant}$$

Ici on va toujours supposer la base = 10,

Pour un nombre réel on doit décrire (sujet de standard IEEE):

- **Le signe**
- **La mantisse**
- **L'exposant**

7. Virgule flottante

Représentation de la virgule flottante en standard IEEE 754:
Simple (32b) et **Double précision** (64b)

Simple précision (bit 0 peut prendre la position 31)

0	1 ... 8	9 ... 31
Signe de la mantisse	Exposant de 0 à 255	Fraction normalisée
	Biais de 127	

Double précision

0	1 ... 11	12 ... 63
Signe de la mantisse	Exposant de 0 à 1024	Fraction normalisée
	Biais de 1023	

7. Virgule flottante

Exemple de conversion: décimal — virgule flottante

$(46.8125)_{10} = (101110.1101)_2$

Décalage: 1.011101101 (5 x) - fraction normalisée

Signe = 0 - nombre est positif

Exposant = 5 + 127 = 132 (10000100) - on introduit le biais

Mantisse = 0111 0110 1000 0000 0000 000 - 23 bits

0	1 ... 8	9 ... 31
0	10000100	0111 0110 1000 0000 0000 000

Attention on suppose toujours ce '1' et on ne l'écrit donc pas...