

# ASM, LE manuel

## Volume I

### Notions de base

Version **1999.5.15**

Par Sébastien Munch (BBossByll), bbossbyll@i-france.com

Ce document est un volume du manuel intitulé "ASM, LE manuel". Il ne peut être distribué que conjointement au reste de ce manuel.

### *Sommaire*

A. Notions de binaire	1
B. Les fonctions logiques	2
C. Les codages	3
D. Addition de nombres binaires	5
E. Le code complément à deux	5
F. Présentation de l'assembleur	5

Ce volume explique les notions de base d'électronique et informatique nécessaires à la compréhension du manuel et de son contenu.

Source:

Cours de génie électrique 1<sup>ère</sup> S – 1998/1999 – Jean-Marc Richmann – Lycée Lavoisier – Mulhouse

### *A. Notion de binaire*

Tout système électronique fonctionne en tout ou rien. C'est-à-dire que, sur un fil électrique, il n'existe que deux états: allumé ou éteint / courant ou pas de courant. Ces deux états correspondent aux états logiques 1 et 0, ou haut (high) et bas (low). Les composants électroniques ne peuvent que manipuler ces données. Ce système est appelé *codage binaire*.

Vocabulaire:

Bit : variable binaire, qui peut prendre la valeur 0 ou 1.

Quartet: combinaison de quatre bits (en anglais: Nibble)

Octet (o): combinaison de huit bits (en anglais: Byte)

Mot : combinaison de 16 bits (en anglais: Word)

Mot long : combinaison de 32 bits (en anglais: Double Word)

Kilo- (K): préfixe signifiant, en électronique et en informatique, 1024 unités (1 Ko = 1024 octets)

Méga- (M): préfixe signifiant 1024 kilo, c'est-à-dire 1048576 unités

Le nombre de combinaisons possible dans un groupe de bits est calculé de la façon suivante:

$\text{nbre de combinaisons} = 2^{\text{nbre de variables d'entrée}}$

Par exemple, dans un octet, on peut mettre  $2^8=256$  combinaisons différentes.

## B. Les fonctions logiques

### Fonctions de base:

#### Identité:

$S(\text{sortie}) = a(\text{entrée})$ .

Table de vérité:

a	S
0	0
1	1

La sortie est à 1 quand l'entrée est à 1.

#### NON:

$S = \overline{a}$

a	S
0	1
1	0

La sortie est à 1 quand l'entrée est à 0.

#### AND (et):

$S = a \cdot b$

a	B	S
0	0	0
0	1	0
1	0	0
1	1	1

La sortie est à 1 quand TOUTES les entrées sont à 1.

#### OR (ou):

$S = a + b$

a	B	S
0	0	0
0	1	1
1	0	1
1	1	1

La sortie est à 1 quand AU MOINS une entrée est à 1.

### Fonctions composées:

Ce sont des assemblages des fonctions élémentaires.

#### NOR (ou non):

$S = \overline{a + b} = \overline{a} \cdot \overline{b}$

a	B	S
0	0	1
0	1	0
1	0	0
1	1	0

La sortie est à 1 quand AUCUNE des entrées n'est à 1.

#### NAND (et non):

$S = \overline{a \cdot b} = \overline{a} + \overline{b}$

a	b	S
0	0	1
0	1	1
1	0	1
1	1	0

La sortie est à 1 quand AU MOINS une entrée est à 0.

### XOR (ou exclusif):

$$S = \overline{a} \cdot \overline{b} + \overline{b} \cdot a$$

a	b	S
0	0	0
0	1	1
1	0	1
1	1	0

La sortie est à 1 quand UNE SEULE des entrées est à 1.

## C. Les codages

### Le code binaire naturel

C'est le plus simple et le plus utilisé. Chaque bit supplémentaire double le nombre de combinaisons possibles. Les nombres binaires sont représentés espacés tous les quartets pour permettre une meilleure lisibilité. Dans la machine, il ne faut naturellement pas entrer ces espaces.

Code bin. naturel				Code
d	c	b	a	déc.
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

389 = %1 1000 0101

(lire de droite à gauche)

$$1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 + 1 \times 2^8$$

### Le code binaire codé décimal

Il est utilisé dans l'électronique digitale.

C'est une (simple) transposition du code décimal en notation binaire.

Code BCD				Code
d	c	b	a	déc.
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

389 = (BCD) 0011 1000 1001

( 0011 = 3 ; 1000 = 8 ; 1001 = 9 )

## Le code binaire réfléchi

Il est appelé réfléchi car pour l'élaborer il faut recopier les données comme si elles étaient réfléchies dans un miroir. Dans le code binaire naturel, plusieurs variables peuvent changer d'état simultanément: entre 7 et 8, 4 variables changent d'état. Dans un système automatisé, il est impossible de changer deux variables exactement en même temps. **Dans le code binaire réfléchi, entre deux valeurs voisines, il n'y a qu'une variable qui change d'état à la fois.**

Code b. réfléchi				Code déc.
d	c	b	a	
0	0	0	0	0
0	0	0	1	1
0	0	1	1	2
0	0	1	0	3
0	1	1	0	4
0	1	1	1	5
0	1	0	1	6
0	1	0	0	7
1	1	0	0	8
1	1	0	1	9
1	1	1	1	10
1	1	1	0	11
1	0	1	0	12
1	0	1	1	13
1	0	0	1	14
1	0	0	0	15

Ce code n'est pas utilisé pour des calculs car son élaboration est trop complexe. Il sert uniquement en automatismes pour résoudre des problèmes de logique combinatoire.

## Le code hexadécimal

Le code hexadécimal sert à simplifier l'écriture du binaire. Il permet de représenter un quartet entier avec un seul caractère.

Code bin. naturel				Code hexa
d	c	b	a	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

%0100 1100 1000 1111 = \$4C8F

( %0100 = \$4 ; %1100 = \$C ; %1000 = 8 ; %1111 = \$F )

## D. Addition de nombres binaires

L'addition s'effectue de la droite vers la gauche. 0 + 0 donne 0. 0 + 1 donne 1. 1 + 1 donne 0 avec une retenue.

Exemple:

5 =       % 0000 0101  
14 =       % 0000 1110  
somme: % 0001 0011 = 19

## E. Le code complément à deux

Le binaire naturel ne permet de coder que des nombre positifs. Pour les nombres négatifs, les microprocesseurs utilisent le *code complément à 2*.

### Le Code complément à un

Avec ce codage, le bit de poids fort est utilisé pour indiquer le signe du nombre. Les nombres négatifs sont représentés en inversant chaque bit du nombre.

Exemple: (sur 8 bits) +13 : 0000 1101 ; -13 : 1111 0010

Malheureusement, ce codage a un gros défaut: il ne fonctionne pas correctement:

-4 s'écrit 1111 1011  
+6 s'écrit 0000 0110  
somme: 0000 0001  
Le résultat est faux: il devrait être 0000 0010

### Le code complément à deux

Pour pallier à ce problème a été inventé le code complément à deux, déduit du code complément à un.

On ajoute tout simplement 1 au nombre négatif après qu'il ait été transformé.

Exemple:

% 0010 0100 = +36  
Inversement de tous les bits:  
% 1101 1011 = -36 en complément à un  
Ajout de 1 à la valeur du nombre:  
% 1101 1100 = -36 en complément à deux

## Notions de débordement et de report

### Le bit de report

Il est aussi appelé retenue.

Nous verrons son utilité dans le Saturn plus loin.

### Le bit de débordement

Il est activé lorsqu'il y a un dépassement de calcul (le résultat est faux).

## F. Présentation de l'assembleur

Je pense qu'en commençant à lire ce manuel, vous deviez déjà avoir quelques notions de programmation dans un ou plusieurs langages. Vous noterez une différence fondamentale entre l'assembleur et tous ces langages: L'assembleur "parle" directement au matériel, alors que les autres langages sont interprétés (passent par diverses étapes de décomposition des instructions avant d'être traduits en langage machine pour être exécutés).

Il faut faire la différence entre deux chose: le langage machine et l'assembleur. Le langage machine (ML, pour *machine language*), c'est ce que comprend le processeur (suite de bits). L'assembleur (ASM), c'est le langage machine traduit en un peu plus compréhensible pour l'être humain, avec des *mnémoniques*.

Par exemple, sur le processeur Saturn (HP48...), le mnémotique **CSRB** correspond, en langage machine, à **\$81E**, c'est-à-dire à **% 1000 0001 1110**. Vous remarquez donc qu'il est plus facile d'écrire en assembleur qu'en langage machine. Le seul inconvénient, c'est qu'il faut *assembler* (traduire en langage machine) un programme ASM avant de pouvoir l'exécuter.