# Code, langage, programme

*Séminaire CARISM*

Laurent Bloch lb@laurentbloch.org
http://www.laurentbloch.org

6 décembre 2016

# Définition du code

Selon Littré, les seules acceptions sont celles qui désignent soit un recueil de lois, comme le Code pénal, soit un « code de signaux, ensemble de signaux à la mer convenus entre les nations maritimes ».

Le Trésor de la langue française introduit des usages contemporains :

- langage secret servant à échanger des informations ;
- système de symboles permettant de représenter une information dans un domaine technique.

Les trois dernières acceptions désignent le changement de représentation d'une information connue par ailleurs, pour des raisons pratiques. Exemple typique : le code Morse.

Cette traduction est nommée *codage* parce qu'elle ne requiert aucune créativité. Émettre un message en Morse, une fois le code appris, est une opération purement mécanique. **Le code n'est pas un langage**.

# Langage et programme

Le mot *informatique* agrège *information* et *automatique* :

- l'information est ce qui donne à quelque chose, de l'intérieur de cette chose, une forme ;
- l'automatique permet l'automatisation de ce processus ;
- le résultat est un automate.

La chose à laquelle il s'agit ici de donner une forme est une idée, que nous nommerons un *algorithme*. La forme donnée sera appelée un *programme*, qui est un texte écrit selon un *langage*.

L'écriture d'un programme est une activité créatrice, elle suppose la maîtrise d'un langage, dont l'acquisition demande quelques années de travail.

# Voir un programme

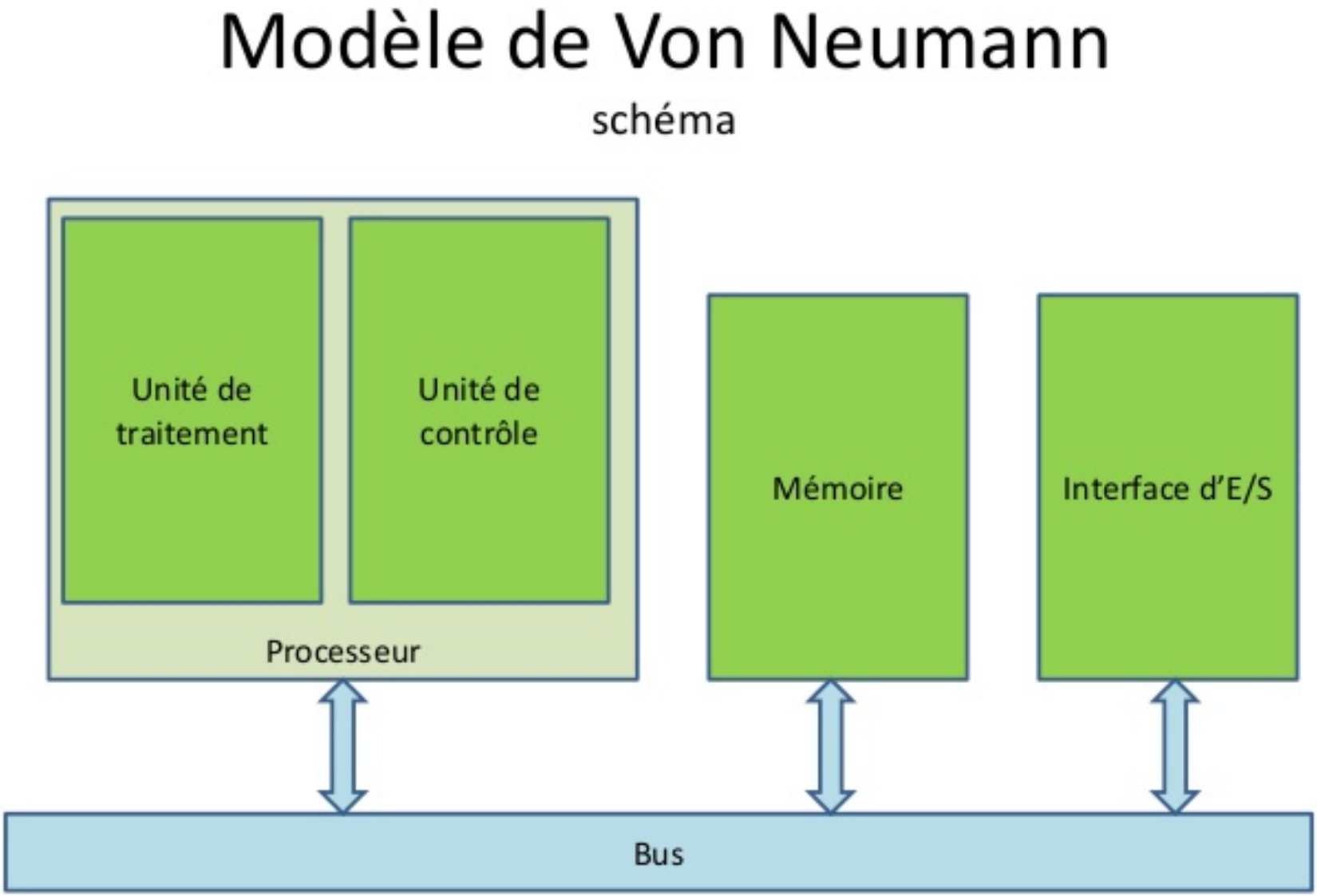
```
(define (hello Args)
   (print "Bonjour tout le monde !"))
```

```
(define (nw-2 s1 s2 match-bonus gap-penalty)
  (let ((ncol (+ (chain-length s1) 2))
        (nlin (+ (chain-length s2) 2)))
    (let ((T (make-matrix nlin ncol 0)))
      (matrix-margins T s1 s2) ;;
      (do ((j 2 (+ j 1)))
          ((= j ncol))
          (matrix-set! T 1 j (* (- j 1) gap-penalty)))
      (do ((i 2 (+ i 1)))
          ((= i nlin) T)
          (matrix-set! T i 1 (* (- i 1) gap-penalty))
          (do ((j 2 (+ j 1)))
              ((= j ncol))
              (let ((val
                     (max
                      (+ (matrix-ref T (- i 1) (- j 1))
                         (if (char=? (matrix-ref T i 0)
                                     (matrix-ref T 0 j))
                             match-bonus 0))
                      (+ (matrix-ref T (- i 1) j)
                         gap-penalty)
                      (+ (matrix-ref T i (- j 1))
                         gap-penalty))))
                (matrix-set! T i j val)))))))
```

```
(define (tri-rapide v)
   (define (tri-aux v imin imax)
      (if (< imin imax)
          (let ((q (partition v imin imax)))
             (tri-aux v imin q)
             (tri-aux v (+ q 1) imax)))
      v)
   (tri-aux v 0 (- (vector-length v) 1)))

(define (partition v imin imax)
;;  (print '(partition ,v ,imin ,imax))
  (let ((x (vector-ref v imin)))
    (let loop ()
      (let loop1 ()
        (if (> (vector-ref v imax) x)
            (begin
              (set! imax (- imax 1))
              (loop1))))
      (let loop2 ()
        (if (< (vector-ref v imin) x)
            (begin
              (set! imin (+ imin 1))
              (loop2))))
      (if (< imin imax)
          (begin
            (swap v imin imax)
            (set! imin (+ imin 1))
            (set! imax (- imax 1))
            (loop))
          imax))))
```

# De quoi sont faits les ordinateurs ?



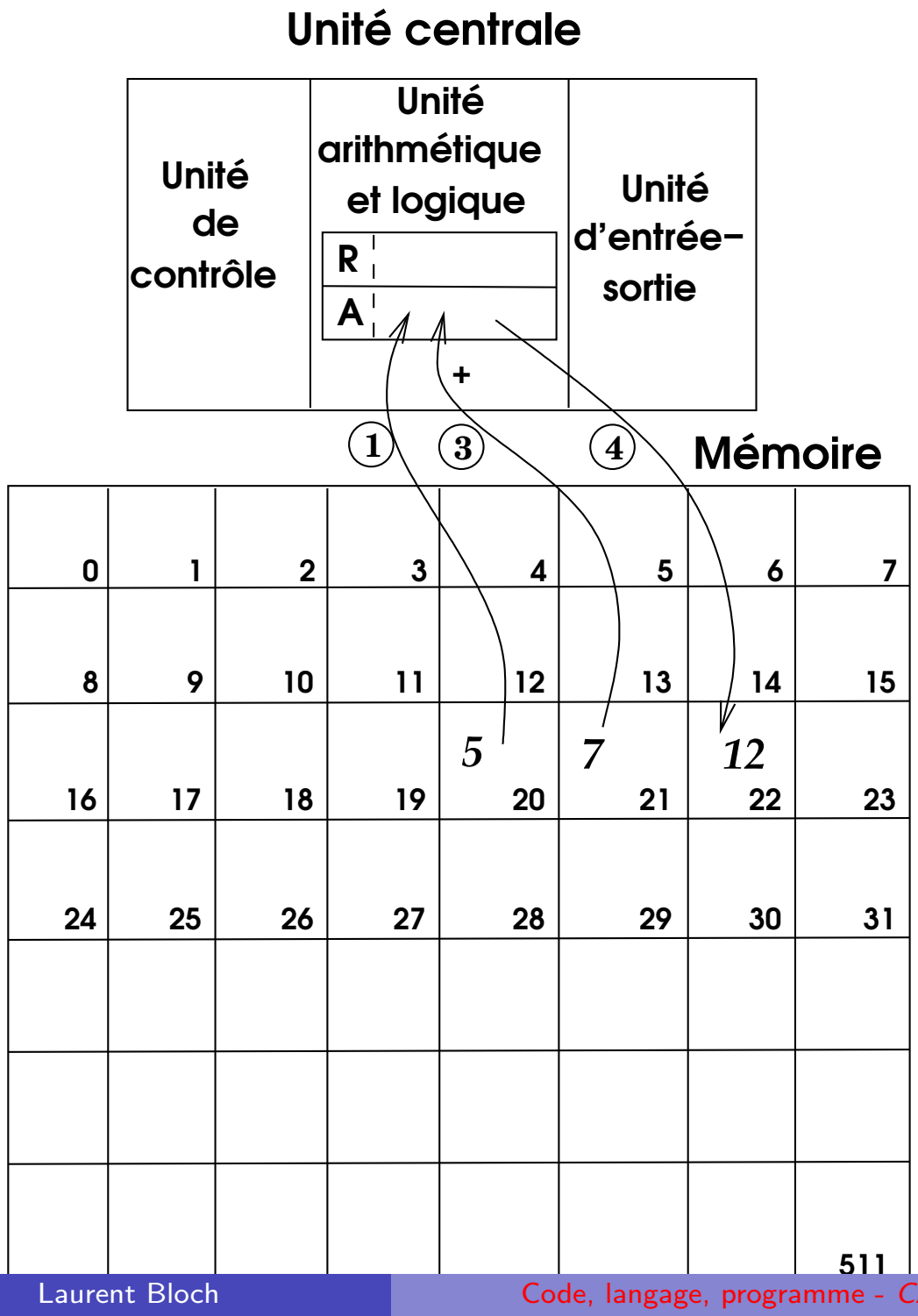
# De quoi sont faits les ordinateurs ?

- Les opérations sont des dispositifs physiques, constitués de circuits électroniques, qui ont des effets sur les données stockées en mémoire.
- La séquence des opérations est décrite par le texte d'un programme (c'est la partie compliquée de la chose) ; les phrases du texte sont appelées instructions ;
- une instruction correspond à une opération de l'unité de traitement (ALU) ou de l'unité de contrôle ;
- la contribution majeure de John von Neumann à cette conception consiste à envisager le texte du programme comme un ensemble de données ordinaires, stockées en mémoire comme les autres ;
- les cicuits de l'unité de contrôle « lisent » le texte du programme et déclenchent l'exécution des opérations, une par une.

# Un programme vu de la machine : du code

1. charge dans le registre A le nombre qui est dans la case mémoire numéro 20 ;
2. teste le contenu de A : s'il vaut zéro passe directement à l'instruction 6 ; sinon ne fais rien, c'est à dire continue en séquence ;
3. additionne au contenu du registre A le nombre qui est dans la case mémoire numéro 21 (le résultat effacera l'ancien contenu du registre A et prendra sa place) ;
4. copie le contenu du registre A dans la case mémoire numéro 22 (le résultat effacera l'ancien contenu de la case mémoire numéro 22 et prendra sa place) ;
5. imprime le contenu de la case mémoire numéro 22 ;
6. fin.

Chaque élément de la liste ci-dessus décrit en langage humain une instruction d'ordinateur.


Unité centrale
Unité de contrôle
Unité arithmétique et logique
R
A
+
Unité d'entrée–sortie
1
3
4
Mémoire
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
511
5
7
12

# Format des instructions machine

Supposons que nous disposons de 16 bits pour représenter une instruction (notre ordinateur a des mots de 16 bits), nos instructions pourront avoir le format suivant :

| bits 0 à 4 | bit 5 | bit 6 | bits 7 à 15 |
|---|---|---|---|
| code opération | numéro du premier registre concerné (0 pour R, 1 pour A) | numéro du second registre concerné (0 pour R, 1 pour A) | adresse mémoire |

# Les instructions machine

| instruction | code opération | nom mnémonique |
|---|---|---|
| chargement mémoire $\rightarrow$ registre | 00001 | LOAD |
| copie registre $\rightarrow$ mémoire | 00010 | STORE |
| addition mémoire à registre | 00011 | ADD |
| imprimer mémoire | 00100 | PRINT |
| test registre et branchement si zéro | 00101 | BZ |
| fin | 00110 | END |

# Le programme (là c'est du code)

Notre petit programme s'écrit en binaire :

| code opération | $1^{er}$ registre | $2^{nd}$ registre | adresse (en binaire) | adresse (en décimal) |
|---|---|---|---|---|
| 00001 | 1 | 0 | 0 0001 0100 | 20 |
| 00101 | 1 | 0 | 0 0000 0101 | 5 |
| 00011 | 1 | 0 | 0 0001 0101 | 21 |
| 00010 | 1 | 0 | 0 0001 0110 | 22 |
| 00100 | 1 | 0 | 0 0001 0110 | 22 |
| 00110 | 1 | 0 | 0 0001 0110 | 22 |

soit, une fois éliminées les fioritures à l'usage du lecteur humain :

```
0000 1100 0001 0100
0010 1100 0000 0101
0001 1100 0001 0101
0001 0100 0001 0110
0010 0100 0001 0110
0011 0100 0001 0110
```

# Un vrai programme

```
 1                                          section .data
 2 00000000 7669646520766964652D              vide: db 'vide vide vide', 0xA
 3 00000009 20766964650A
 4 0000000F 426F6E6A6F75722074-               hello: db 'Bonjour tout le monde !', 0xA
 5 00000018 6F7574206C65206D6F-
 6 00000021 6E646520210A
 7                                            strLen: equ $-hello
 8                                          section .text
 9                                            global    _start
10                                            _start:
11 00000000 B804000000                              mov eax,4
12 00000005 BB01000000                              mov ebx,1
13 0000000A B9[0F000000]                            mov ecx, hello
14 0000000F BA18000000                              mov edx, strLen
15 00000014 CD80                                    int 0x80
16
17 00000016 B801000000                              mov eax,1
18 0000001B BB00000000                              mov ebx,0
19 00000020 CD80                                    int 0x80

-- Notation hexadécimale : 0 1 2 3 4 5 6 7 8 9 A B C D E F
-- Notation hexadécimale : B8    1011 1000    184
```

# Commentaire du programme

```
— Notation hexadécimale : 0 1 2 3 4 5 6 7 8 9 A B C D E F
— Notation hexadécimale : B8    1011 1000    184

— mov     eax,4                ;system call number (sys_write)
— mov     ebx,1                ;first argument: file handle (stdout)
— mov     ecx, hello           ;second argument: pointer to message
— mov     edx, strLen          ;third argument: message length
— int     0x80                 ;call kernel
```

# Une hiérarchie de langages et métalangages

- bas niveau, près du matériel : langage machine ;
- assembleur (jadis autocodeur) ;
- à peine au-dessus de l'assembleur : C ;
- langages utilisables : Java, Scheme, OCaml...
- langages interprétés : PHP, Python, Ruby...
- pseudo-code ;
- langage humain.

# Les 4 concepts de l'informatique

- Machine ;
- langage ;
- programme ;
- information.

# *Code is Law. Really ?*

Lawrence Lessig : « Le droit du cyberespace tel quel, c'est le code (...) Et c'est ce code, non pas le code américain ou le code français, qui détermine quels droits et quels pouvoirs y existent. »

Olivier Iteanu : « La loi est l'expression de la volonté du peuple, pas le code. »

Emmanuel Cauvin : « *Law is Code*, ou plus exactement, *Law must be Code*. Pour s'appliquer la loi doit être traduite sous forme de code informatique. »

# Pseudo-code

Pseudo-code : code arbitraire, indépendant des caractéristiques matérielles de l'ordinateur, qui devra être traduit dans l'ordinateur en code qu'il peut employer. Un programme écrit en pseudo-code est traduit en langage machine par un programme de traduction en vue de son exécution.