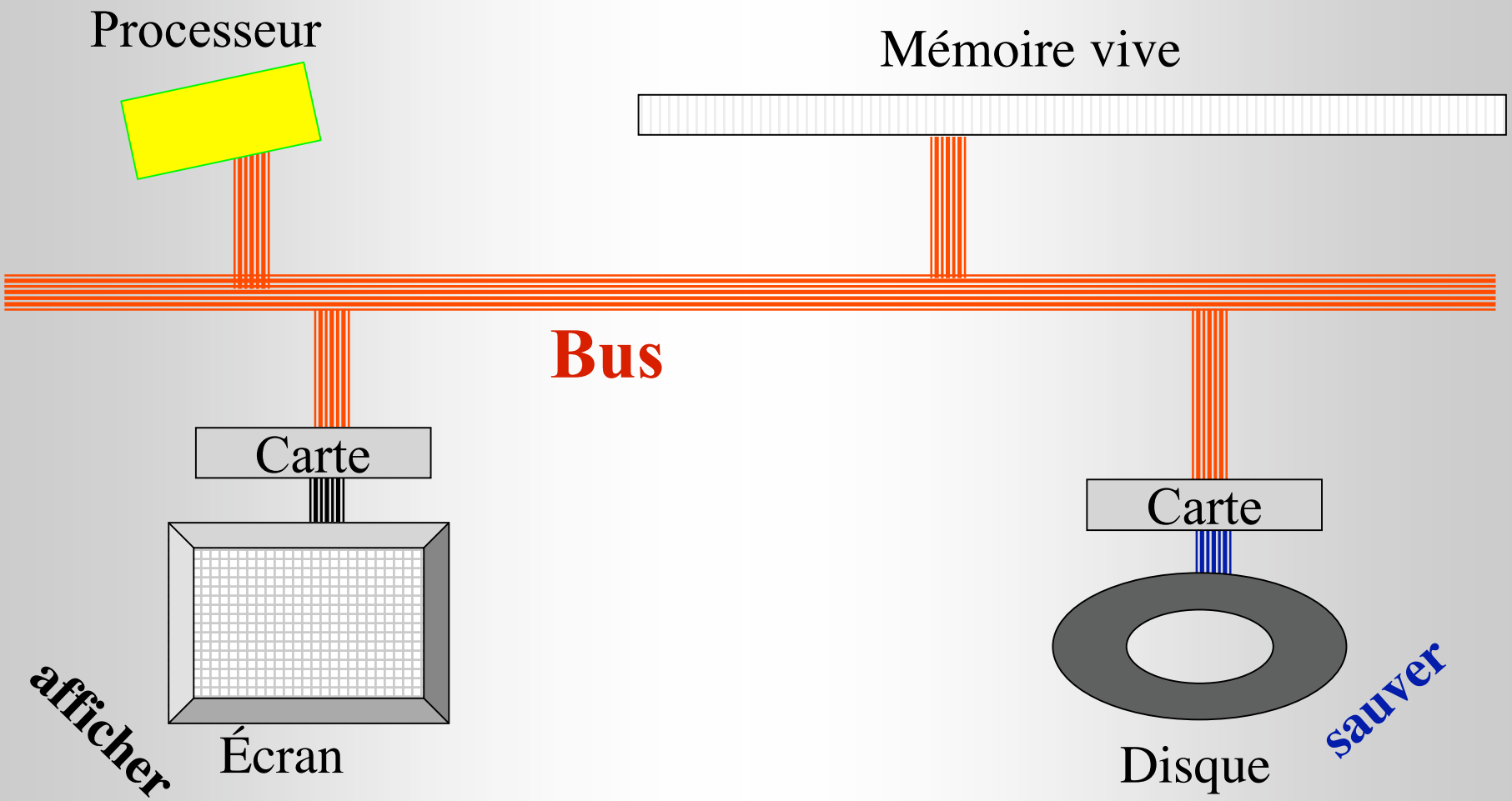
Codage : notions élémentaires

Les codes utilisés depuis toujours.

I. La mémoire

Le schéma matériel



Que fait le processeur

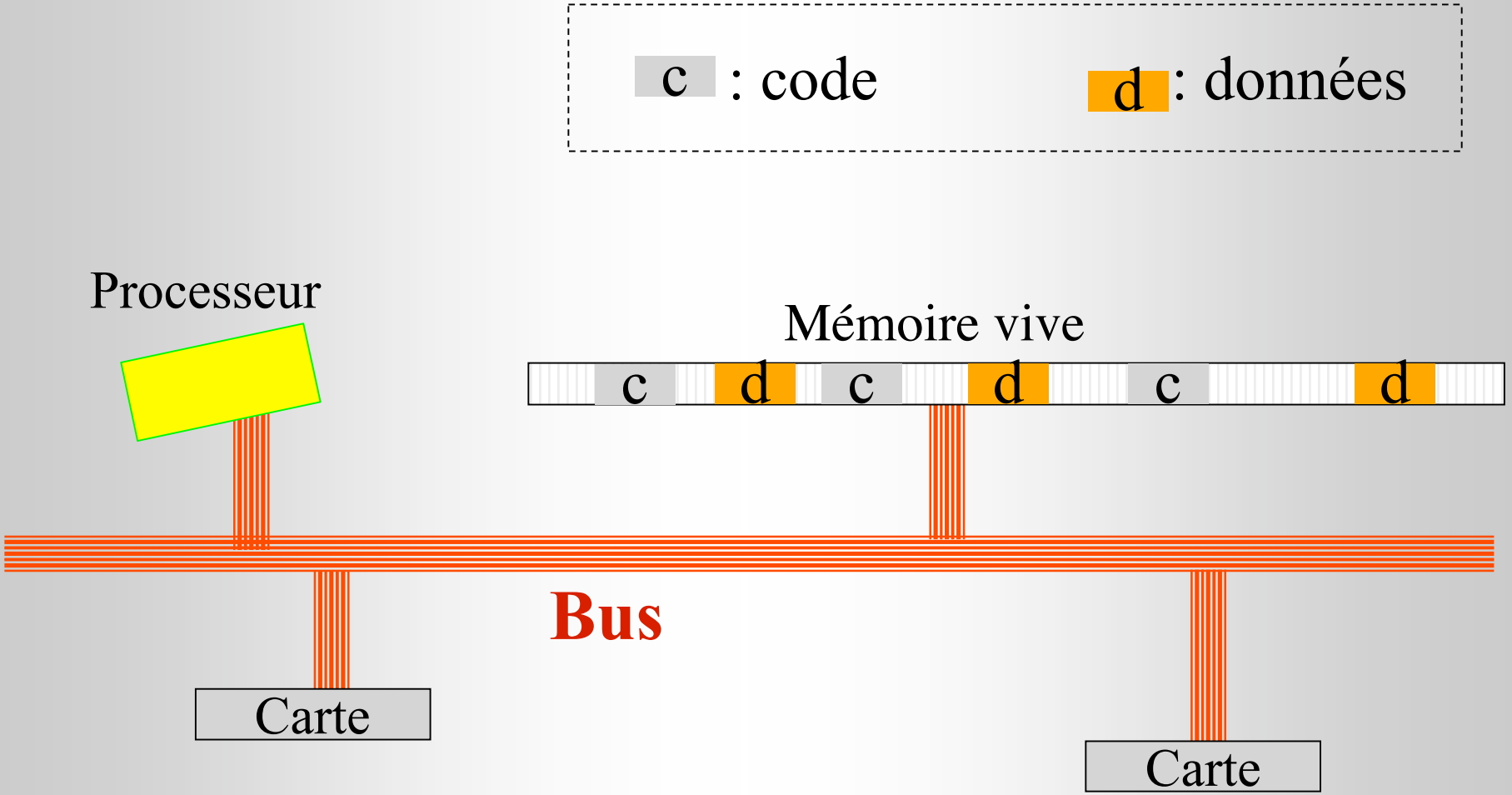
Il exécute des **opérations**, typiquement

- Ajouter ou soustraire deux nombres
- Comparer deux nombres
- Copier des nombres de et vers la mémoire vive ("charger" et "sauver")
  - Explication : il y a beaucoup plus de place en mémoire vive

- L'opération à faire est indiquée par une **instruction** en mémoire.
  - (donc il faut aussi lire des instructions)
- Les nombres sont les **arguments** de l'opération
- Les **arguments et les** nombres sauvés en mémoire sont des **données**

Organisation de la mémoire

- Les instructions sont rangées en mémoire **dans l'ordre où elles doivent être exécutées**, dans des zones de code
- Les données sont aussi en mémoire, le plus souvent dans des zones séparées (zones de données)

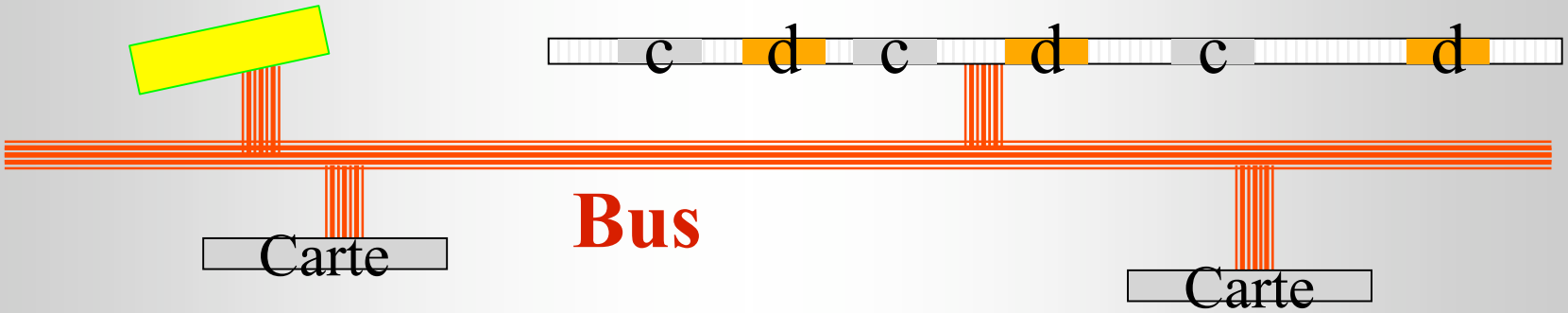

c : code
d : données
Processeur
Mémoire vive
c
d
c
d
c
d
Bus
Carte
Carte

A quoi ressemble la mémoire ?

- Ensemble de condensateurs qui peuvent être chargés ou pas.
- On note 0 pour déchargé, 1 pour chargé. L'état de chaque condensateur est un chiffre binaire ou **bit**.
- Une zone mémoire, c'est toujours une suite de 0 et de 1 (ex: 00110100011011111101…)

Echanges processeur - mémoire

- Un **mot** est le nombre maximum de bits que le processeur est capable de lire ou d'écrire en une fois.
- Un mot de 8 bits s'appelle un **octet**
- Les processeurs courants ont des mots de 4 ou 8 octets. Ils lisent aussi à l'occasion 1 ou 2 octets à la fois.



- Chaque octet a une **adresse** (comme un numéro dans une rue)
- Une partie du bus sert à transmettre l'adresse qui intéresse le processeur
- Une seconde partie sert à transmettre la **commande** (ex : "lire" ou "écrire")
- Le reste transmet la **donnée** (en général un mot)

Notion de codage

- Comment mettre en mémoire des objets différents (instructions, nombres, texte,..) ?
- Pour chaque type d'objet, on choisit un code
- Un code est une règle pour faire correspondre un nombre (sa représentation) à chaque objet du type

Exemples de codes

- Le code des mois : janvier = 1, février = 2, … décembre = 12.
- C'est arbitraire (pourquoi commencer en janvier ?)
- Si tout le monde utilise le même code, tout le monde comprend que "9" veut dire "septembre"

- Le code des départements : Ain = 1, Aisne = 2, Allier = 3, …
- Utilisé pour les immatriculations, les codes postaux, les numéros de sécurité sociale,..

- Si l'on n'utilise que ces deux codes :
  - 110 ne représente rien
  - 68 est le Haut Rhin
  - 9 est ambigu ("septembre" ou "Ariège")

Bien comprendre que...

- En informatique, la règle est implémentée dans des logiciels ou du matériel qui utilisent les données codées
- Les mots mémoire sont tous pareils. Il n'y a pas trace en mémoire du code utilisé.
- Les données sont inutilisables (ambiguës) si l'on ne connaît pas le code qui leur est attaché.

Bien faire la différence !

- Comment décrire la mémoire : écritures binaires, décimales, hexadécimales
- Qu'est-ce que veut dire ce qu'on a décrit ? connaître le code en vigueur dans cette zone mémoire.
- Il peut y avoir des données, avec plusieurs règles de codage possibles, ou un exécutable (code machine).

Les codes de base

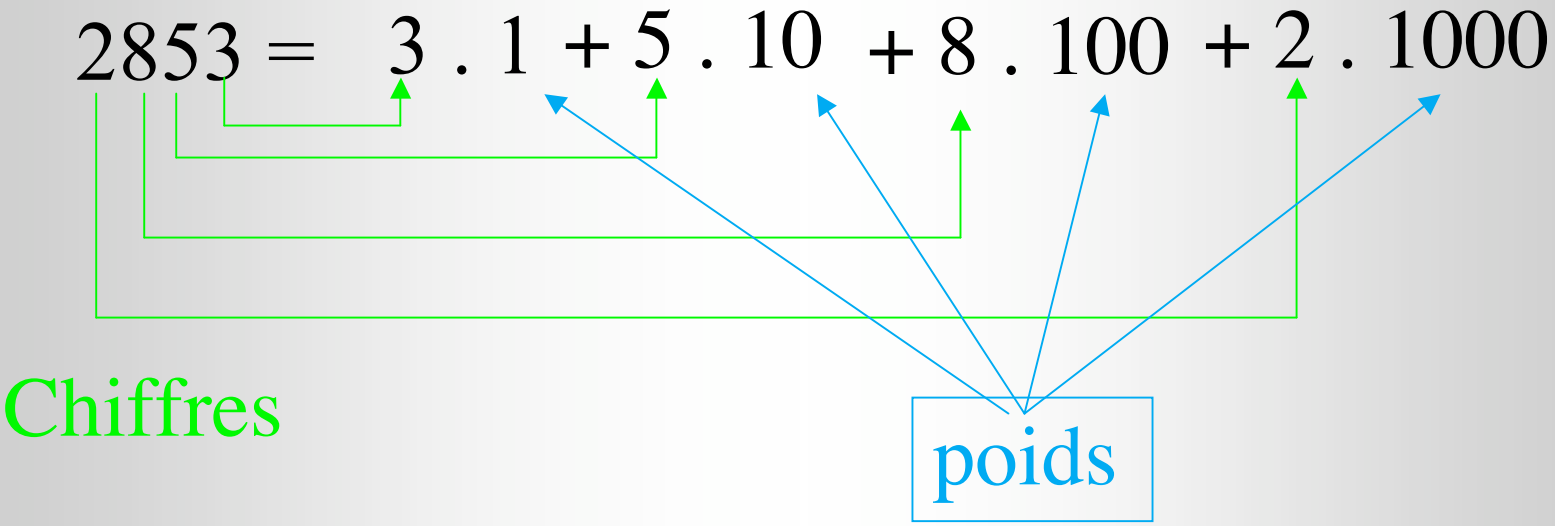
- Le code des instructions (plus tard).
- Ecrire un texte
- Calculer avec des nombres entiers
- Calculer avec des nombres à virgule (plus tard).

II. Décrire le contenu de la mémoire

Des 0 et des 1 !!!

Le binaire

- Dans notre écriture habituelle :



poids 1, 10, 100, …. ⇒ base dix

plus grand chiffre : 9

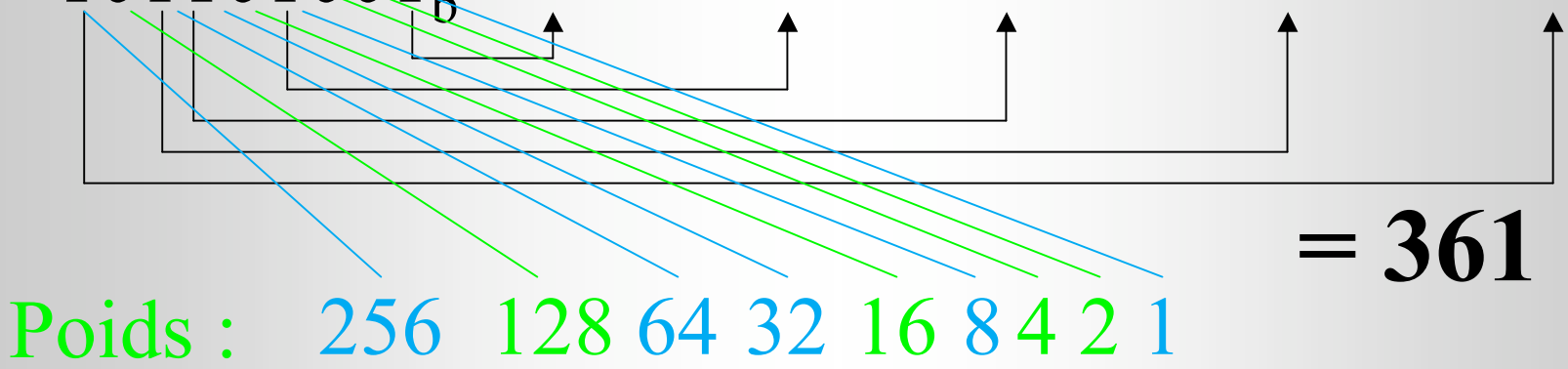
Base deux

Même principe, mais :

- poids 1, 2, 4, 8, 16, 32, 64, 128, 256, …
- plus grand chiffre : 1

Exemple :

$$1011101001_b = 1 \cdot 1 + 1 \cdot 8 + 1 \cdot 32 + 1 \cdot 64 + 1 \cdot 256$$



Comparaison binaire / décimal

- On peut décrire la même zone de mémoire par 101101001_b ou par (256+64+32+8+1) ou par 361
- La forme décimale est plus facile à comprendre et à mémoriser
- Il faut faire un calcul pour retrouver les chiffres binaires (et pour un grand nombre, c'est long).

Du décimal au binaire

- en binaire,
  ajouter un 0 à droite = multiplier par 2,
  supprimer un zéro à droite = diviser par 2

Décimal :

1473 = (147 x 10) + 3

Quotient | reste

de division par 10

Binaire :

10110100 1

Quotient | reste

de division par 2

$$101101001_b = 361$$

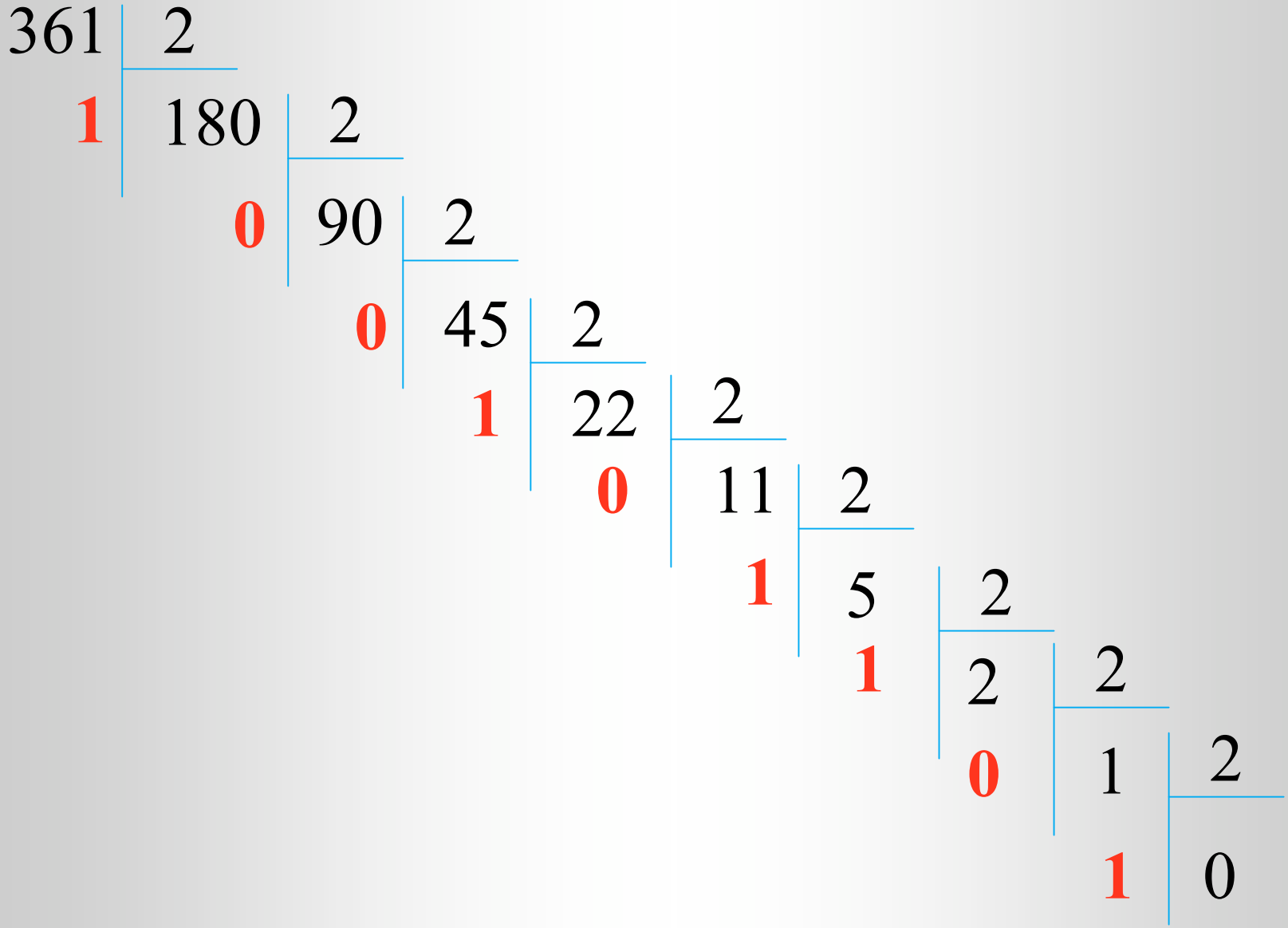
$$101101000_b = 360$$

$$10110100_b = 180$$

Une division par deux fournit

le chiffre de droite (= le reste)

On recommence sur le quotient


361 | 2
1 | 180 | 2
0 | 90 | 2
0 | 45 | 2
1 | 22 | 2
0 | 11 | 2
1 | 5 | 2
1 | 2 | 2
0 | 1 | 2
1 | 0

Décrire la mémoire : l'hexadécimal

- Depuis longtemps, on lit et on écrit en mémoire par octets

  1 octet = 8 chiffres binaires (ex : 10010110_b)
- Un octet se coupe en deux morceaux de 4 chiffres binaires (ex $1001\ 0110_b$)
- Il n'y a pas tellement de combinaisons de 4 chiffres binaires :

binaire	valeur	binaire	valeur
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

Il y en a 16...

Principe de base

- idée : remplacer un groupe par sa valeur
  ex $1001\ 0110_b \rightarrow 96_h$
- problème : les valeurs > 9 sont ambiguës (1111 et 0001 **01**01s'écriraient tous deux "15")
- Solution : on note les **chiffres hexadécimaux** au delà de 9 par une lettre A, B, ...

Les chiffres hexadécimaux

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

Passer de binaire en hexadécimal

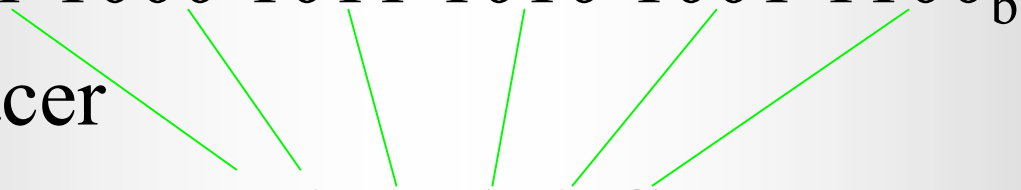
- On peut décrire une zone mémoire dans les deux langages. Ex :

$$10110001011101010011100_b$$

- Séparer en paquets de 4 en partant de la droite :

$$101\ 1000\ 1011\ 1010\ 1001\ 1100_b$$

- Remplacer



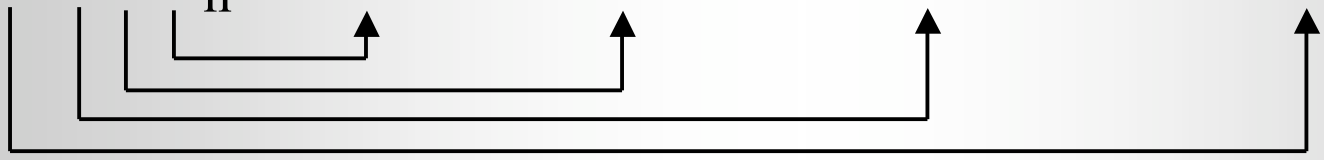
$$5\ 8\ B\ A\ 9\ C_h$$

Passer d'hexadécimal à décimal

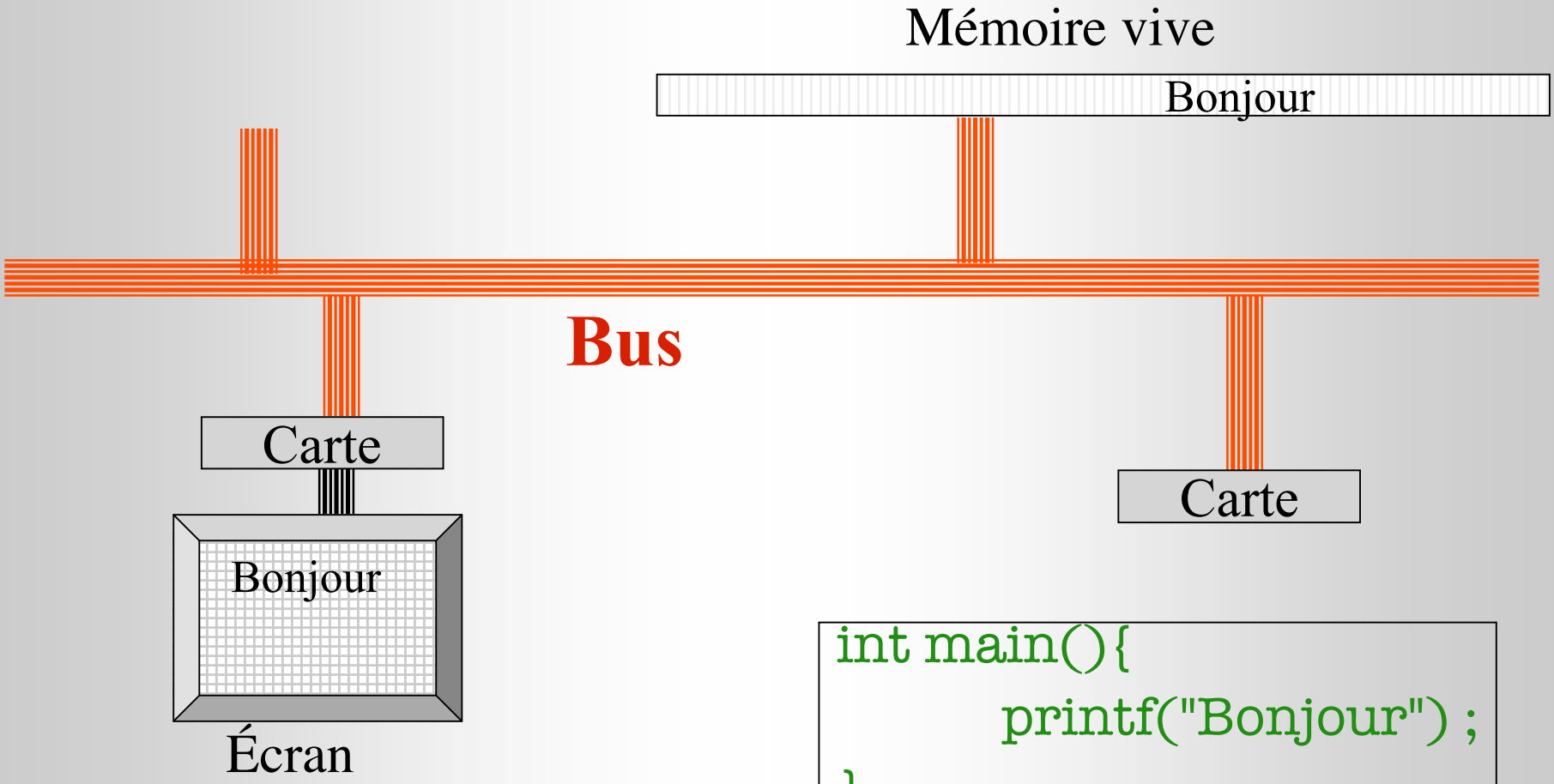
- Décaler de 1 chiffre hexa, c'est décaler de 4 chiffres binaires, donc multiplier par 16

hexa	B				A				9				C			
binaire	1	0	1	1	1	0	1	0	1	0	0	1	1	1	0	0
Poids	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

$BA9C_h = 12 . 1 + 9 . 16 + 10 . 256 + 11 . 4096$



III Coder du texte




```
int main(){
        printf("Bonjour") ;
}
```

Code des caractères

- caractère = lettre (a,b,..,A,B,..), chiffre (1,…,9), ponctuation (, ; : . ! ? {}[]), etc. Comment les noter en mémoire ???
- Code inventé par l'association des ingénieurs américains (ascii).
- Caractères standard sur 7 bits (sans accents), étendu sur 8 bits. ⇒ chaque caractère étendu est codé par un nombre entre 1 et 255 (standard : de 1 à 127)

Quelques codes faciles

Les chiffres

car.	hexa	dec
0	**30**	48
1	31	49
2	32	50
.....		
9	39	57

Les majuscules

car	hexa	dec
A	**41**	65
B	42	66
.....		
O	4F	79
P	50	80
.....		
Z	5A	90

Les minuscules

car	hexa	dec
a	**61**	97
b	62	98
.....		
o	6F	111
p	70	112
.....		
z	7A	122

Dans un programme C

- Quand on écrit `char c ;`
  le compilateur réserve un octet en mémoire pour mettre un caractère (pas encore connu)
- Quand on écrit `c = 'A' ;`
  le programme compilé écrira $0100\ 0001_b$, $(= 41_h = 65_d)$, à l'emplacement réservé pour c.

Lire une touche clavier

- La fonction getchar() renvoie la prochaine touche appuyée (un code étendu)
- Ex : demander à l'utilisateur "Voulez-vous quitter le programme (o/n) ?" et attendre sa réponse

```
#include <stdio.h>

 int main(void) {
     char c ;

     do {printf("\nVoulez-vous quitter… (o/n) : ") ;
     c=getchar() ;
     } while(c !='o' && c != 'n') ;
     if (c == 'n') …..
}
```