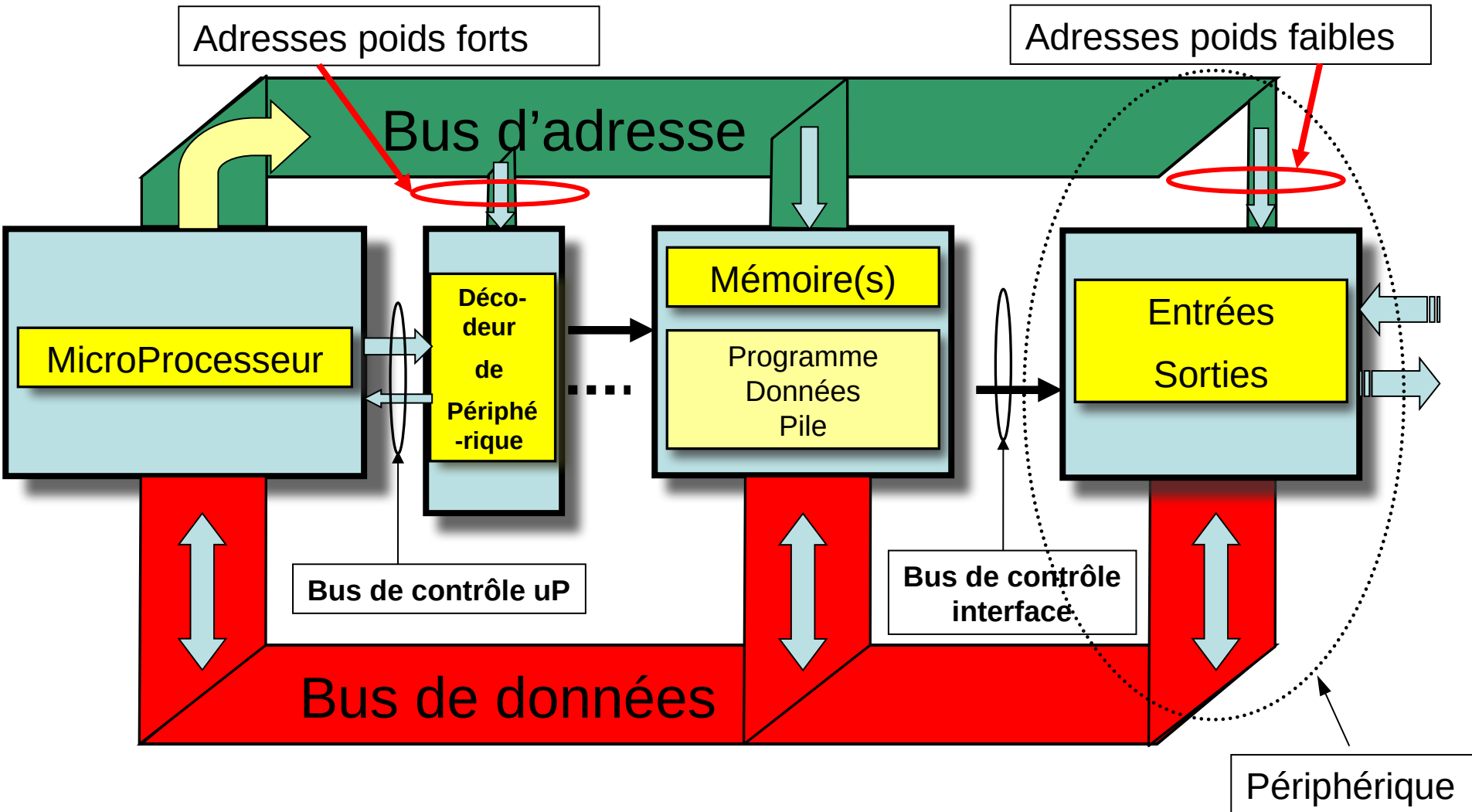


# Architecture d'un système informatique

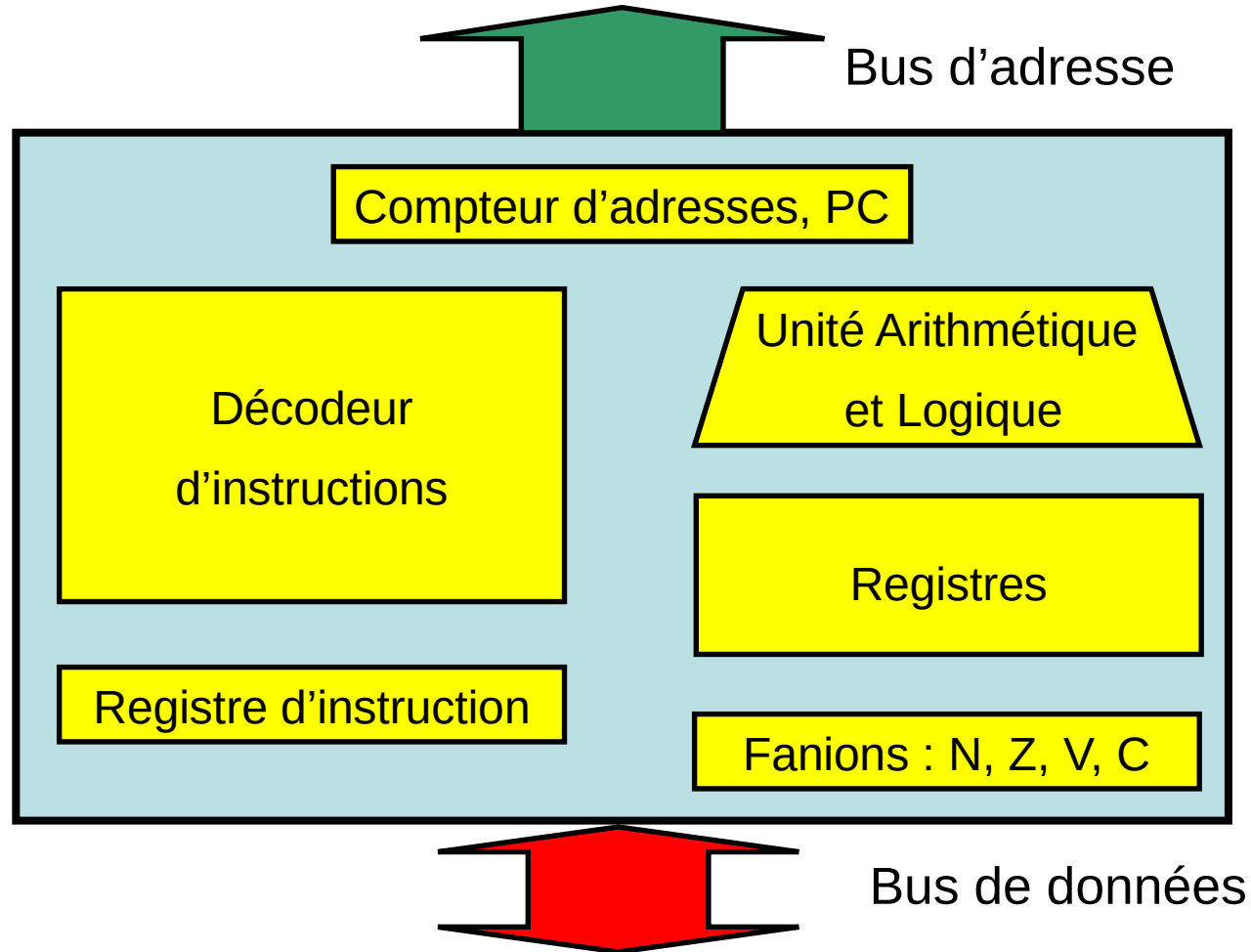
F. Vannel

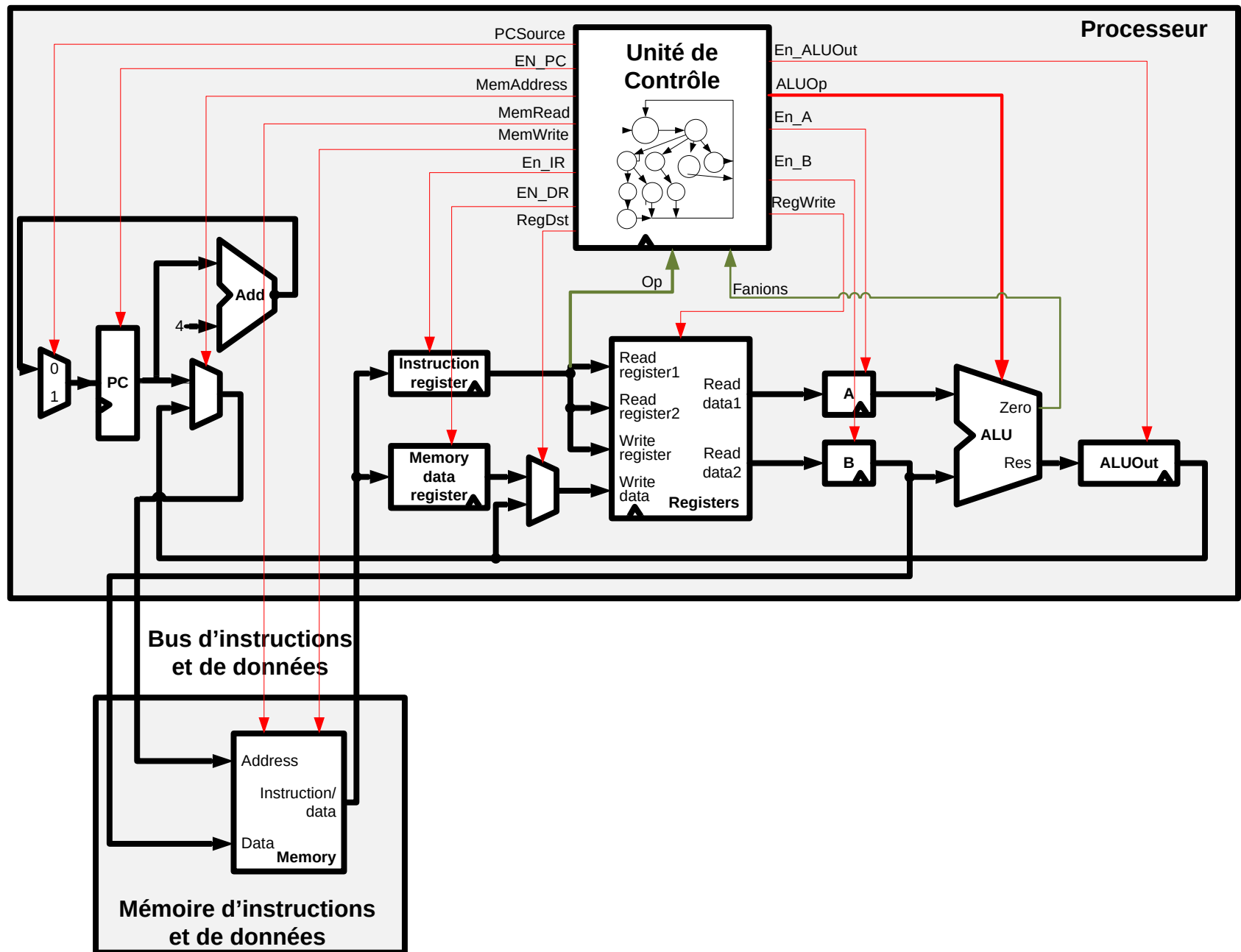
- Les niveaux d'abstraction d'un système informatique sont:
  - application
  - algorithme
  - langage de haut niveau
  - système d'exploitation
  - langage de bas niveau (assembleur/langage machine)
  - architecture de la machine
  - microarchitecture
  - circuits logiques
  - dispositifs électroniques

# Architecture d'un système informatique



# Modèle simplifié d'un processeur





# Mais avant tout : le bit

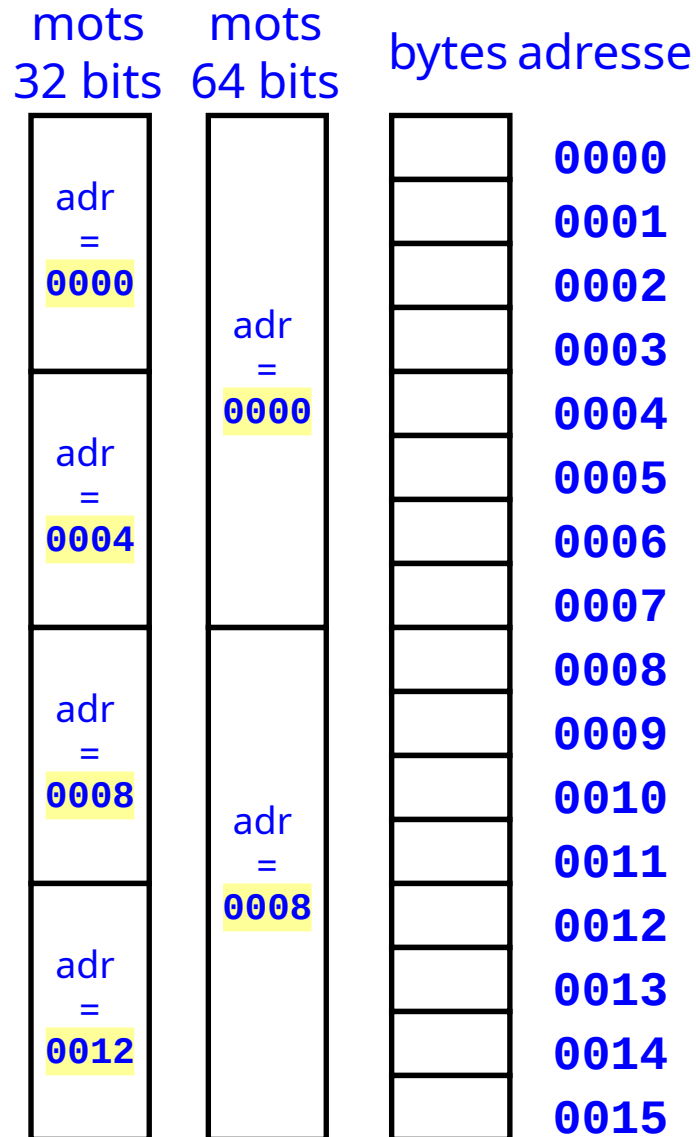
- Un ordinateur ne comprends que le **0** et le **1**
- Un humain préfère d'autres représentations (base 10, base 16, lettres...)

	Ordinateur	Humain
Nombre entiers signé	1010 0011 ( 8 bits )	-93
	0000 0001 0000 1111 ( 16 bits )	527
Nombres à virgules	01000000 01001001 00001110 01010110 ( 32 bits représentation IEEE754 )	3.14150
ASCII	0100 0001 ( 8 bits)	'A'
Valeur (adresse, donnée, etc.)	01000000 01001001 00001110 01010110	0x40 49 0E 56

# Modèle d'une mémoire

	Adresses	Contenu
0x00	0 0 0 0 0 0 0 0	0 1 1 0 1 1 0 1
0x01	0 0 0 0 0 0 0 1	0 1 0 0 0 1 0 1
	0 0 0 0 0 0 1 0	0 0 1 0 1 1 1 1
	0 0 0 0 0 0 1 1	1 1 0 1 0 1 0 1
	0 0 0 0 0 1 0 0	0 1 1 0 1 0 0 1
	. . . . .	// //
	0 1 1 1 1 0 1 1	1 0 1 0 1 1 0 1
	0 1 1 1 1 1 0 0	0 0 1 1 1 0 0 0
	0 1 1 1 1 1 0 1	1 1 0 0 0 1 0 1
	0 1 1 1 1 1 1 0	1 0 1 0 1 0 0 1
0x7F	0 1 1 1 1 1 1 1	0 1 1 1 1 0 1 0

# Adressage en fonction de la taille des données

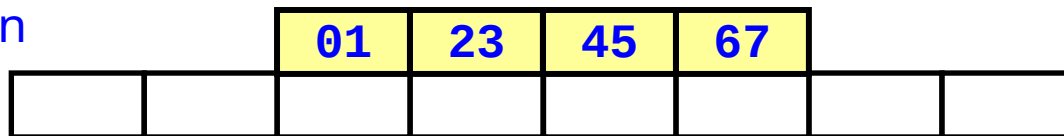




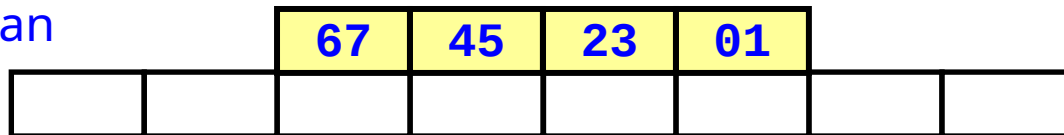
# little/big endian

- supposez que la valeur de la variable **toto** est **0x01234567** et
- qu'elle est stockée à l'adresse **0x0100** (c'est-à-dire **&toto=0x0100**)

big endian



little endian



# Instruction machine

- Une instruction machine est une opération simple et précise exécutée par le processeur.
- Une suite de bits (ou mot de n bits) construit selon les spécifications du fabricant du processeur définit une instruction.
- Exemple (cas d'un processeur ARM) :

Instruction en:	
binaire	1110 0000 0100 0001 0011 0000 0000 0000
hexadécimal	E0413000
assembleur	SUB R3, R1, R0
pseudo-code	$R3 = R1 - R0$

# Exemple de programme en assembleur ARM

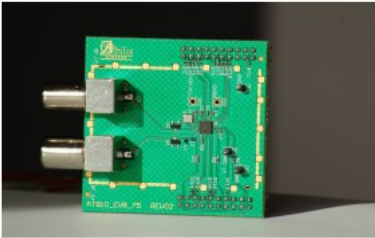
```
checksum
    LDR r3,[r0]          ; r3 = data
    MOV r0,#0            ; parity = 0
    MOV r1,#0            ; i = 0
checksum_loop
    ADD r1,r1,#1         ; i = i+1
    CMP r1,#0x20         ; compare i, 32
    AND r4,r3,#0x1       ; r4 = r3 & 1
    XOR r0,r4,r0         ; parity = parity ^ r4
    MOV r3, r3 LSR #1    ; r3 = r3 >> 1
    BCC checksum_loop   ; if (i<32) loop
    MOV pc,r14           ; return parity
```

Note : r0 contient en début de fonction l'adresse de val  
En fin de fonction r0 doit contenir la valeur à retourner

## PSEUDO-CODE équivalent

```
fonction checksum ( int * val)
{
    parity = 0
    faire 32 fois boucle
    {
        parity = parity XOR val[0];
        val = val >> 1;
    }
    retourner parity
}
```

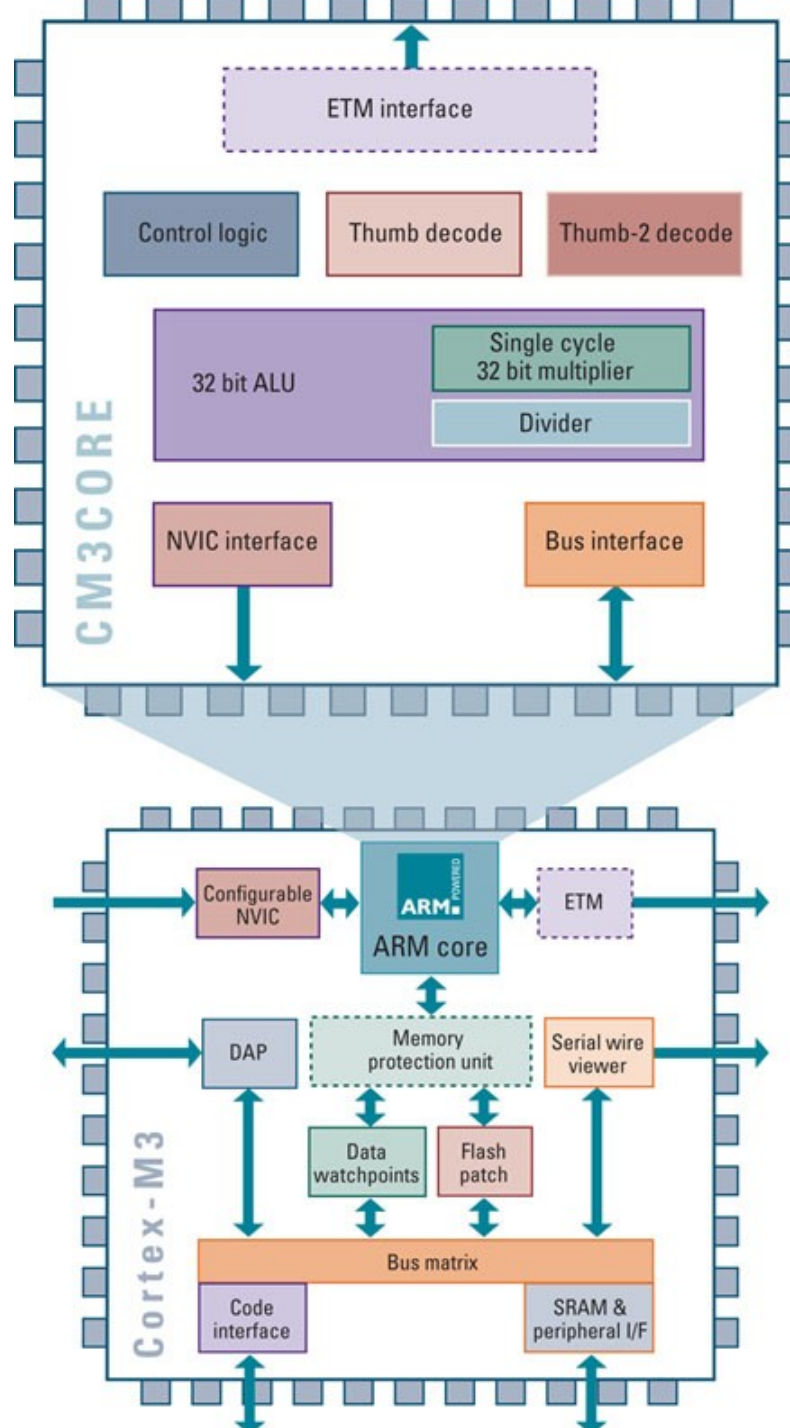
# Qu'est-ce qu'un système embarqué?



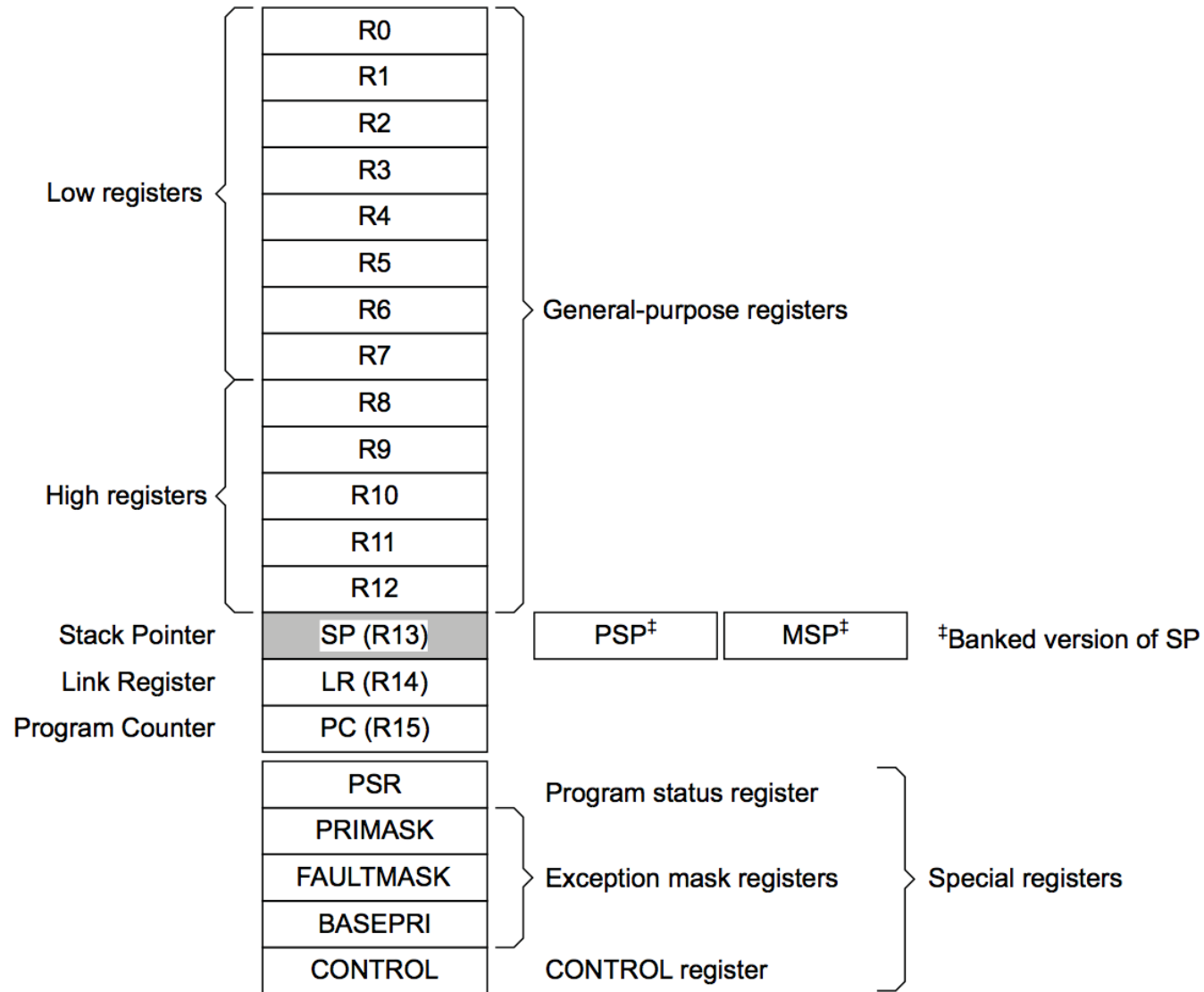
# Le processeur ARM Cortex M3

- Fabricant : ARM
- Famille : Cortex M3
- Architecture : ARMv7-M (ceci n'a rien à voir avec le ARM7!)
- Introduit en 2006
- Architecture de Harvard
- Processeur 32 bits RISC
- Jeu d'instruction 16 ou 32 bits
  - Instruction soit du type ARM = 32 bits
  - Soit du type réduit (Thumb) = 16 bits
- Pipeline à 3 étages
- MPU (Memory Protection Unit)

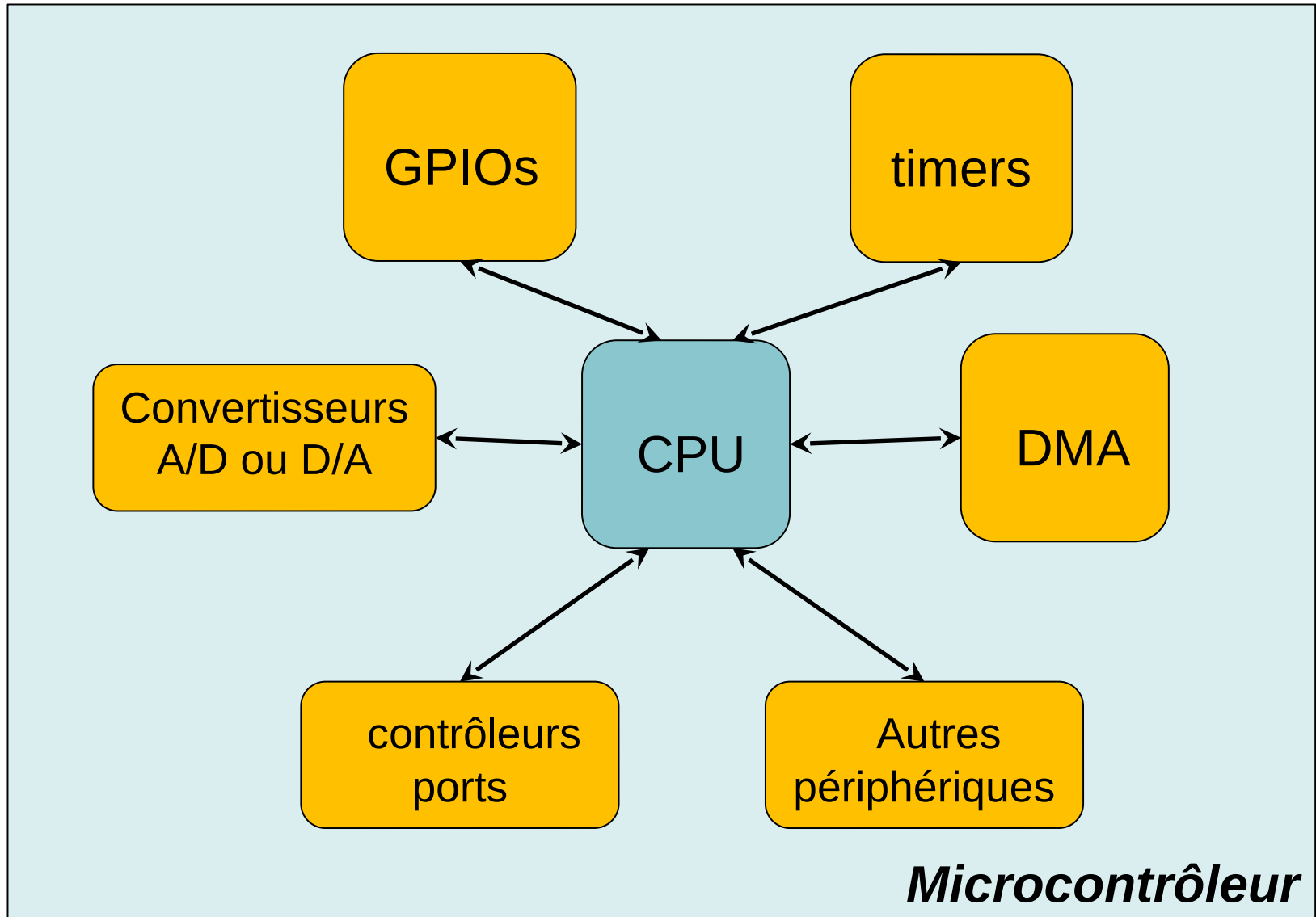
# ARM Cortex M3



# Registres



# Différence entre processeur et microcontrôleur?





# Le microcontrôleur LPC1769

