Systèmes Embarqués
IINFO 741
FI4 IAI

Guillaume Ginolhac

Avec Yajing Yan, Sébastien Monnet, Jean Jacques Curtelin

Polytech Annecy-Chambéry

Structure du module

- 5*1.5h CM :
  - langage C
  - systèmes embarqués
- 2*1.5h TD :
  - 1.5h TD : programmation langage C
  - 1.5h TD : exercices
- 6*4h TP :
  - 4 TP Arduino
  - 2 TP Rasbperry
- 1 ET

Plan du Cours

I. Introduction au langage C

II. Introduction au Systèmes Embarqués

III. Principes du Processeur

IV. Entrées/Sorties

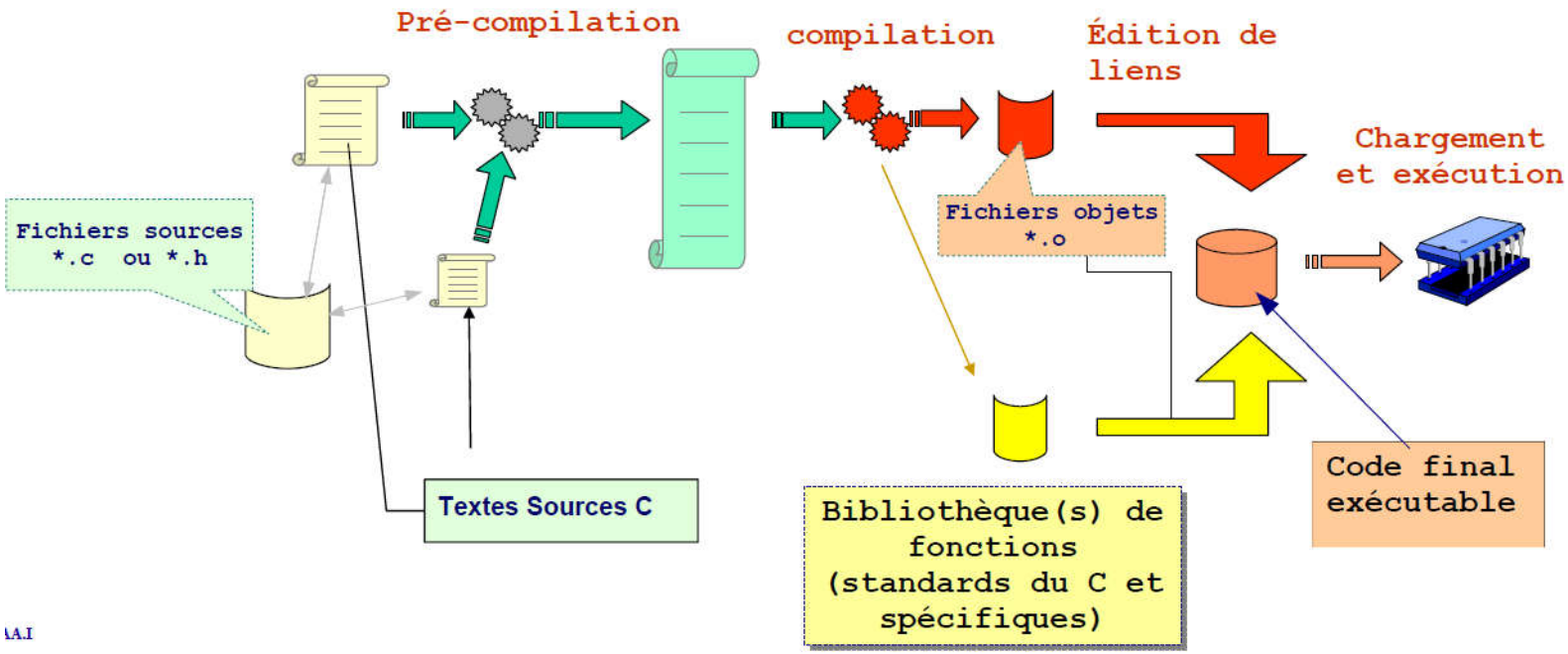
V. Arduino

Langage C : structure d'un programme

```c
// librairies
#include<stdio.h>
#include<stdlib.h>

// constantes
#define PI 3.1415

// declarations fonctions
int MaFct(int,float);

// declarations variables globales
int var1=1;
```

```c
int main()
{
   // declarations variables
   int vars;
   float var2;
   // saisie
   scanf("%f",&var2);

   // appel fonction
   vars=MaFct(var1,var2);

   // affichage
   printf("\n vars : %d",vars);

   return(0);
}
// definition fonctions
int MaFct(int a,float b)
{
   …
}
```

Traitement d'un programme source C



Types de variables

- `char (%c)` ou `unsigned char (%u)` : 1 octet ([-128,127] ou [0,255])
- `int (%d ou %u), long int (%ld), float (%f), double (%lf)`
- Exemple dans un processeur 32 bits :
  - `short int : 2 octets`
  - `long int : 4 octets`
  - `float : 4 octets`
  - `double : 8 octets`

Note : Il n'existe pas de type Booléen en C (un alias est disponible en Arduino)

Type	Déclaration	scanf et printf
Caractère	char	%c
Entier	int	%d
Réel	float	%f

Déclarations de variables : scalaires

- Déclaration des variables, obligatoire avant d'utiliser une variable :

```
type identificateur ;
```

- Exemples :

```
int x;
float x1, x2,
char rep;
```

- Pas de déclaration à l'intérieur du code ! (en C++, comme avec l'arduino possible mais à éviter SVP)

Déclarations de variables : tableaux

- Déclaration des variables, obligatoire avant d'utiliser une variable :

```
type identificateur[NbElements] ;
```

- Exemples :

```
#define MAX 50
int main(){
  int tab1[100];
  float xtab2[10];
  char rep[MAX];
```

- Remarque : indice des tableaux varie de 0 à N-1 (avec N la taille indiquée dans la déclaration)

Portée et qualificateur des variables

- **Portée :**
  - `locale`: déclaration variable à l'intérieur d'une fonction et valable uniquement à l'intérieur
  - `static` : permet à une variable locale de conserver sa valeur lors d'appels successifs de la fonction
  - `global` : déclaration au dessus du main. Elle est connue par tous les programmes du fichier
  - `extern` : permet de spécifier que la variable est déclarée dans un autre fichier

- **Qualificateur :**
  - `const` : valeur ne doit pas changer
  - `volatile` : peut être modifiée par une source externe du programme

Opérateurs

- Mathématiques : +, -, *,/,% (modulo)
- Affectation : = (`a = 5;`)
- Comparaison : ==, !=, >,>=,<,<=
- Logique : &&, ||

Exemple :

```
rep=(c>=5) && (c<=10) ; // rep égal à 1 si c
compris entre 5 et 10 sinon 0
```

- Conversion explicite :

```
new_var=(<type>) <nom_var>;
```

Exemple :

```
float x=3.2;
int a=(int)x; // a vaudra alors 3
```

Manipulations sur les bits

• Manipulations de bits beaucoup utilisées dans les SE car on doit souvent modifier / tester un seul bit dans une variable 1/2 octets

- Mettre à 1 le bit 4 de a :

```
unsigned a = 0x000F; /* 0b 0000 0000 0000 1111 */
unsigned b = 0x0010; /* 0b 0000 0000 0001 0000 */
unsigned a = a | b; /* 0b 0000 0000 0001 1111 soit 0x001F */
```

- Mettre à 0 le bit 3 de a :

```
unsigned a = 0x000F; /* 0b 0000 0000 0000 1111 */
unsigned b = 0xFFF7; /* 0b 1111 1111 1111 0111 */
unsigned a = a & b; /* 0b 0000 0000 0000 0111 soit 0x0007 */
```

• Masques très utiles aussi dans les tests (if ou while) pour tester certains bits d'un registre

• On utilise aussi souvent << et >> pour décaler des bits à gauche ou à droite.
Exemple : `5<<3`
`5 = 0b00000101 et donc 5<<3 = 0b00101000 =40 (5*2`^{3}`)`

Structures de contrôle

- If : sélection

```
if (condition)
{
        …
}
else
{
        …
}
```

- Switch : sélection multiple

```
switch (condition)
{
        case ?:
        …
        break;
        …
}
```

- While : répétition

```
while (condition)
{
        …
}
```

ou

```
do
{
        …
}while(condition);
```

- For : répétition

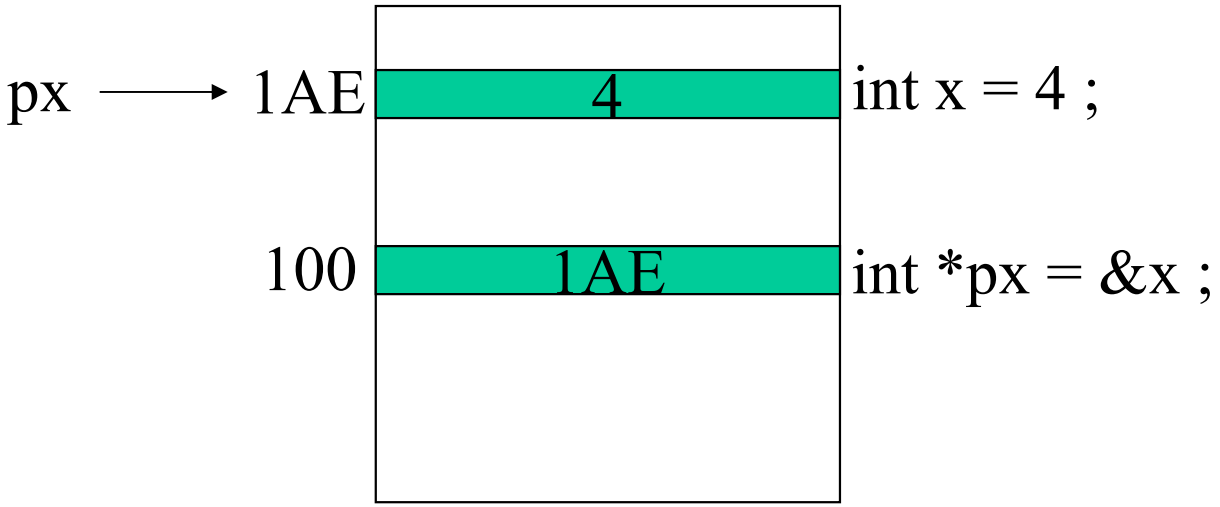
```
for(i=0;i<ValFinale;i=i+1)
{
        …
}
```

Remarque : pas de elseif, attention aux boucles infinies dans le while

Pointeurs

Type particulier de variables

Définition : variable contenant l'adresse d'une autre variable



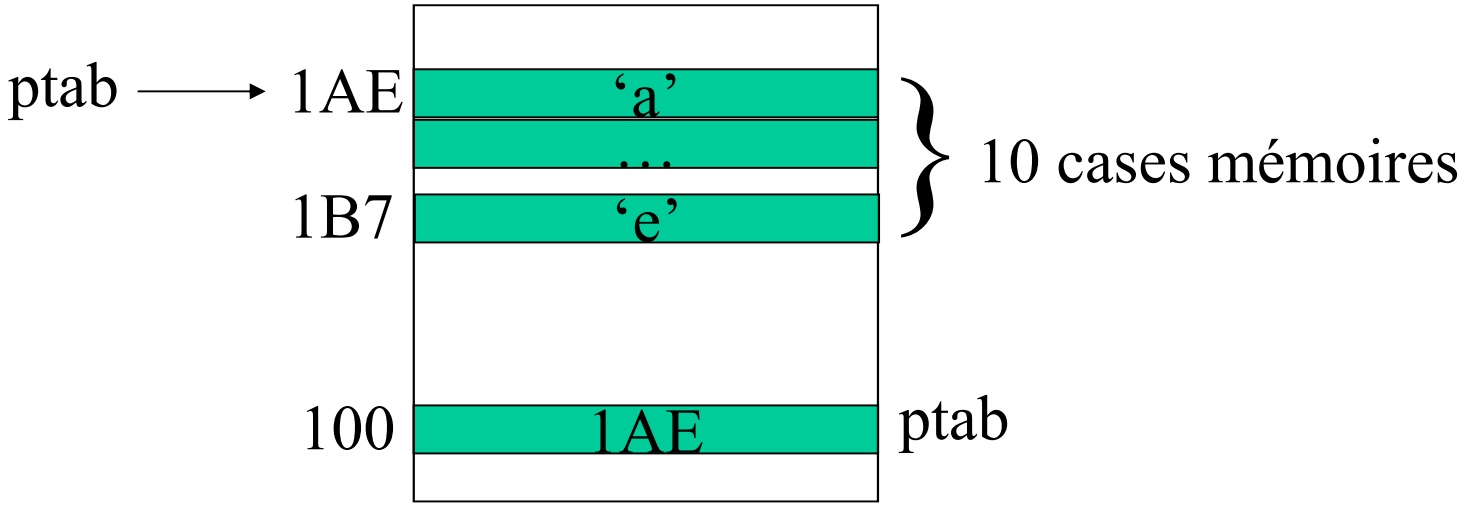
Pour avoir accès au contenu du pointeur : printf("\n %d" ,*px);

Pointeurs pour les types : char, int, float

Pointeurs et tableaux

En fait un pointeur est un tableau, dans la mémoire on a :

```
char ptab[10];
```



Autre solution pour déclarer :

```
char *ptab;
int n=10;
…
ptab = malloc(n*sizeof(n));
…
free(ptab);
```

Variables de type enregistrement

• Définir un nouveau type de variable (au-dessus de `int main()`):

```
typedef int boolean ;

boolean test;
```

• Définir une variable de type enregistrement (au-dessus de `int main()`):

```
typedef struct{
        int age;
        char nom[];
        char prenom[];
        }etudiant;

etudiant f1;
```

Fonctions et sous programmes

- Fonction : (d1,d2) : entrées (ou données) ; R : sortie (résultat)

```
type NomFonction(type d1,type d2, …){
        type R;
        …
        return(R);
}
```

- Sous programme : (d1,d2) : entrées (ou données) ; (R1,R2) : sorties (résultats)

```
void NomSousProg(type d1,type d2, type *R1,type R2[])
{
        *R1 = … ;
}
```

- Dans l'en-tête déclarer la fonction :

```
type NomFonction(type,type, …);
void NomSousProg(type,type , type *,type []);
```

- Dans le programme principal :

```
r=NomFonction(a1,a2);
NomSousProg(a1,a2,&r1,r2);
```

Passage de paramètres

• par valeur : paramètres d'entrées
Une copie de la valeur de la variable est fournie au programme, impossible à changer cette valeur

• par adresse : paramètres de sorties ou d'entrées / sorties
L'adresse dans la mémoire est utilisée par la fonction qui peut donc modifier la valeur de la variable

Exemple :

```
void echange (int *px,int *py);

int main(){
    int a=4,b=3;
    echange(&a,&b);
}
void echange (int *px,int *py){
    int temp;
    temp=*px;
    *px=*py;
    *py=temp;
}
```

Bibliothèques : affichage (stdio.h)

- `printf("\n N=%d",N);` :
  - Chaine de format pouvant contenir des indicateurs (%c,%d,%f,%s) (mais pas de tableau d'entiers et de réels)
  - Arguments : les variables dont on veut afficher les valeurs et qui viendront s'afficher à la place de l'indicateur

- Exemple :
  - `printf("\n la tempéature est de %f C°", temperature);`

- Caractère `putc()` ou chaine de caractères `puts()`

Bibliothèques : saisie (stdio.h)

- `scanf("%d",&N);` :
  - indicateurs (%c,%d,%f,%s) (mais pas de tableau d'entiers et de réels)
  - Arguments : les variables dont on veut afficher les valeurs et qui viendront s'afficher à la place de l'indicateur. On met l'adresse sauf pour %s car c'est un tableau (et donc une adresse)

- Exemple :
  - `scanf("%f", &temperature);`

- Caractère `getchar()` ou chaine de caractères `gets()` (permet d'éviter le `fflush(stdin);`

Bibliothèques : accès fichier (stdio.h)

- Ouverture fichier : fopen()
  - `FILE* fopen(char* chemin,char*mode)` : FILE * est un pointeur de fichier à déclarer, le chemin contient l'endroit où se trouve le fichier à ouvrir et le mode ce qu'on veut faire avec ce fichier (`"r","w","rb","wb", …`)
  - La fonction renvoie NULL en cas d'erreur (vérifier avec un if)

- Fermerture fichier : fclose(fd)
  - `int fclose(FILE *)`

Bibliothèques : accès fichier (stdio.h)

- Ecrire fichier : fprintf()
  - `int fprintf(FILE *fd,char*format,…)` :
    FILE * est un pointeur de fichier à déclarer, la chaine de format est précédée par un pointeur vers le fichier ouvert

```
int main() {
FILE * fp;
fp = fopen ("file.txt", "w+");
fprintf(fp, "%s %s %s %d", "We", "are", "in", 2012);
fclose(fp);}
```

- Autre fonctions : fputc(), fputs()

Bibliothèques : accès fichier (stdio.h)

- Lire fichier : fscanf()
  - **int fscanf(FILE *stream, const char *format, ...)**

```
int main() {
      char str1[10], str2[10], str3[10];
      int year;
      FILE * fp;
      fp = fopen ("file.txt", "r");
      fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
      printf("Read String1 |%s|\n", str1 );
      printf("Read String2 |%s|\n", str2 );
      printf("Read String3 |%s|\n", str3 );
      printf("Read Integer |%d|\n", year );
      fclose(fp);
return(0); }
```

- Autre fonction : fgetc(),fgets()

Exercice langage C

- Ecrire le sous programme moyenne qui calcule la moyenne de n notes rangées dans un tableau.

Coder l'algorithme du sous programme et du programme principal dans les deux cas suivants : la commande return et le passage par adresse

Plan du Cours

I. Introduction au langage C

II. Introduction au Systèmes Embarqués

Définition / Motivations / Schéma général
Développement / Exemples de SE

III. Principes du Processeur

IV. Entrées/Sorties

V. Arduino

Définition

- Système électronique et informatique autonome
- Interagit avec l'extérieur
- Pas ou peu d'interface utilisateur (systèmes enfouis)
- Spécialisé dans une tâche bien précise
- SOC : System On Chip
  - Système complet
  - En général SOC = 1 ou + processeur+ROM+ E/S

Motivations : Systèmes Embarqués

- Miniaturisation processeurs
- Dans tous les objets de la vie courante
  - Électroménager : cafetière, lave-vaiselle, …
  - Consoles de jeux vidéos, photos, télévisions
  - Transports : systèmes de navigation aérien et pour voiture (+ de 100 processeurs)
  - Outils de communications de l'information : téléphonie
  - Santé: implants, aide personnes handicapées, etc.
  - Bâtiment : ascenseurs, surveillance, éclairage, accès
  - Production : automate, commande d'énergie, stockage, robot
  - Spatial : satellites, …

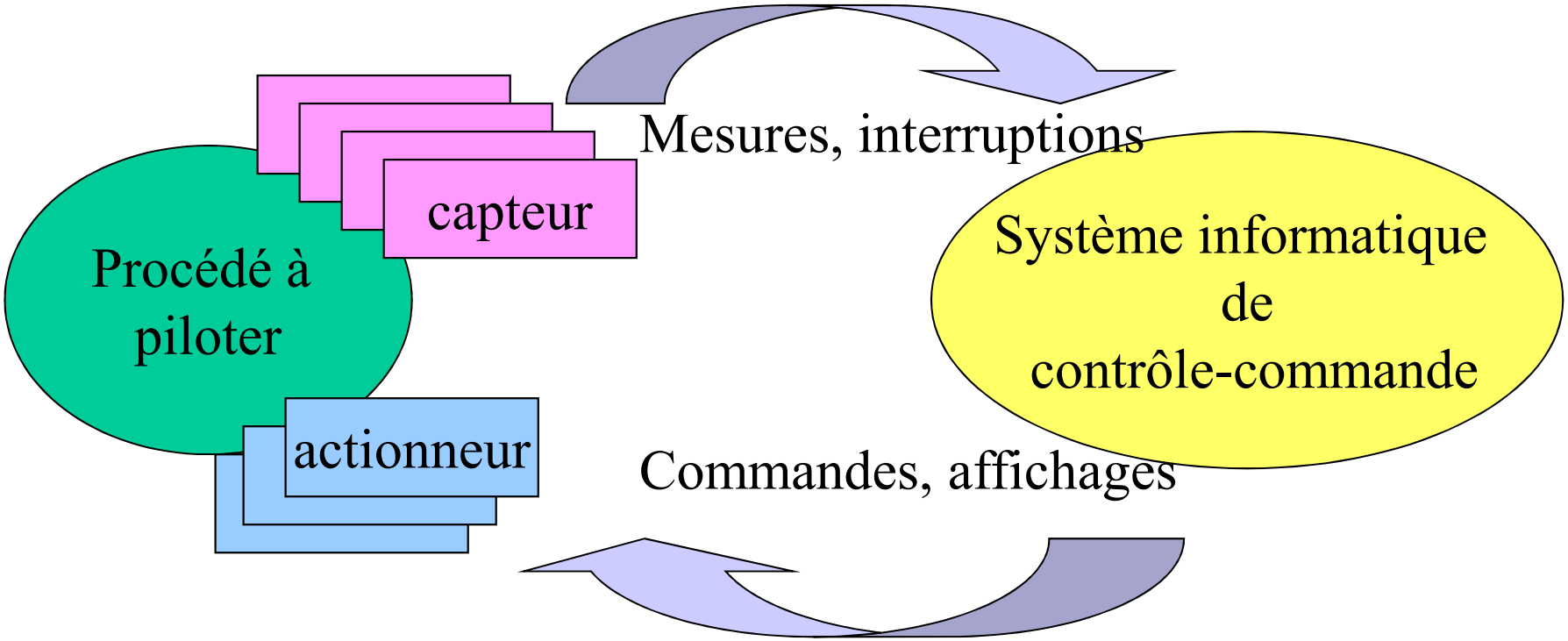
Différences SE / système classique

- Basse consommation.
  - exemple : téléphonie, santé, …
- Il faut trouver le bon compromis entre vitesse et consommation d ’énergie
  - Le SE le plus rapide n’est pas obligatoirement = recherché
- Système temps réel (Real Time RT):
  - La plupart des SE sont RT, Le système doit réagir dans un temps déterminé (deadline).
  - Les SE sont principalement des RT
  - Vitesse liée aux performances RT.
  - Certaines applications ont « plusieurs Deadline »
    - exemple : Synchronisation dans les applications Multimedia (arrivée image et son), airbag, ...

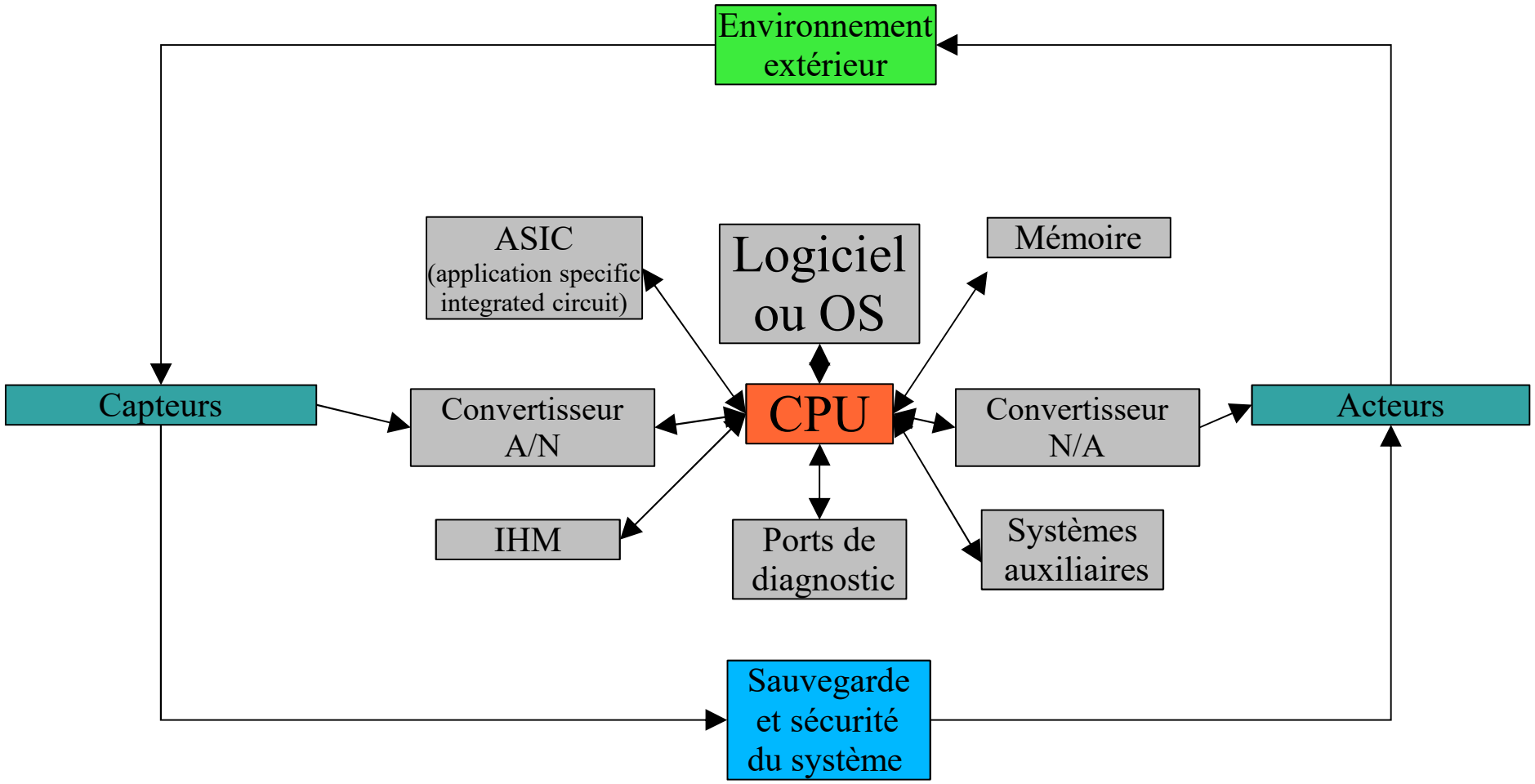
Résumé

Un système informatique embarqué

est un système de contrôle-commande



Organisation d'un système embarqué



Résumé

Un système informatique doit pouvoir :

- **Communiquer avec l'extérieur** : *capteur, actionneur, …*

- **Exécuter des instructions** : *processeur*

- **Conserver des informations** : *mémoire*

Schéma Général

Un système informatique est composé de circuits :

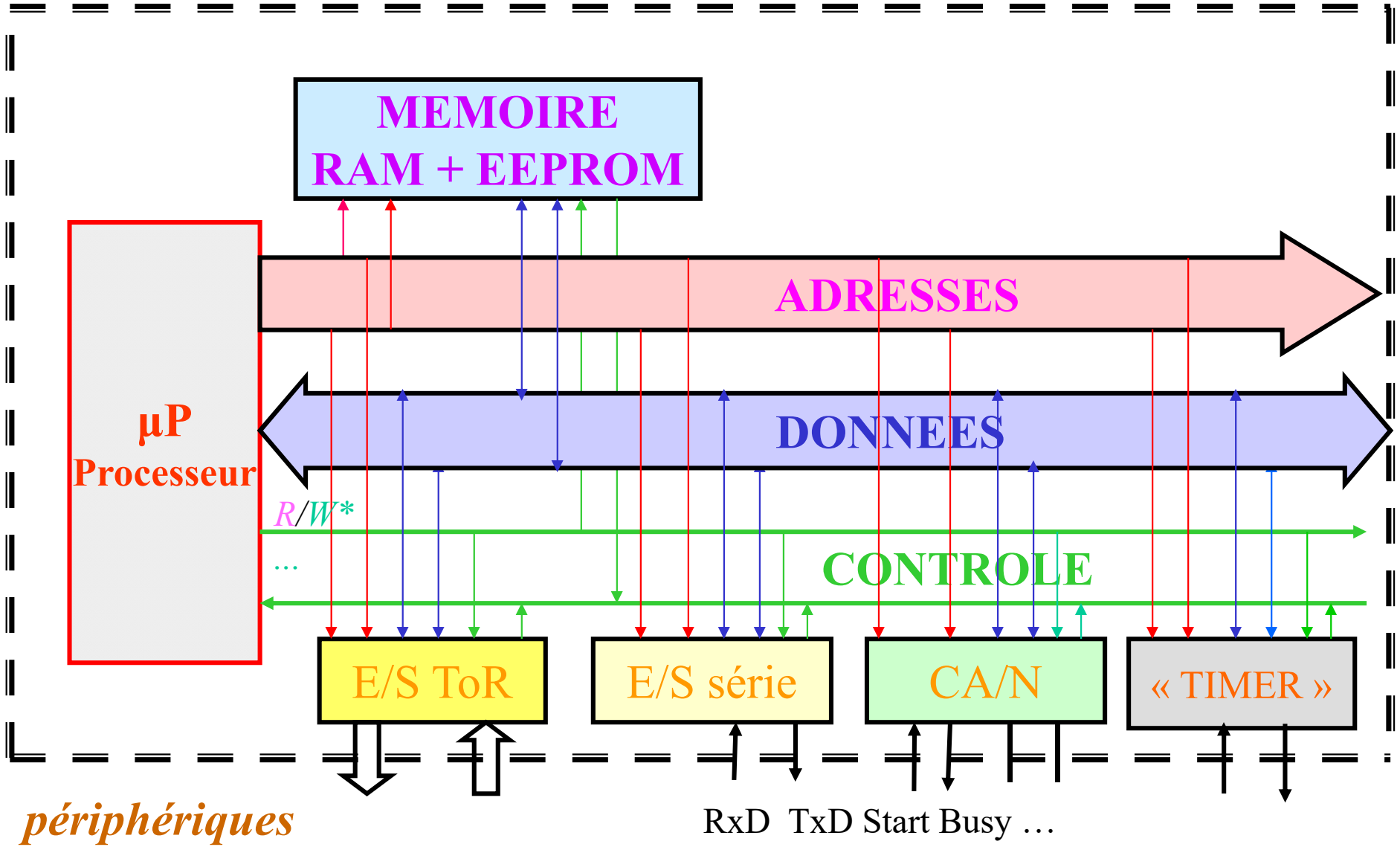
- **processeur(s)**
- **mémoire**
- coupleurs d'**Entrées/Sorties**

reliés entre eux par des **bus** :

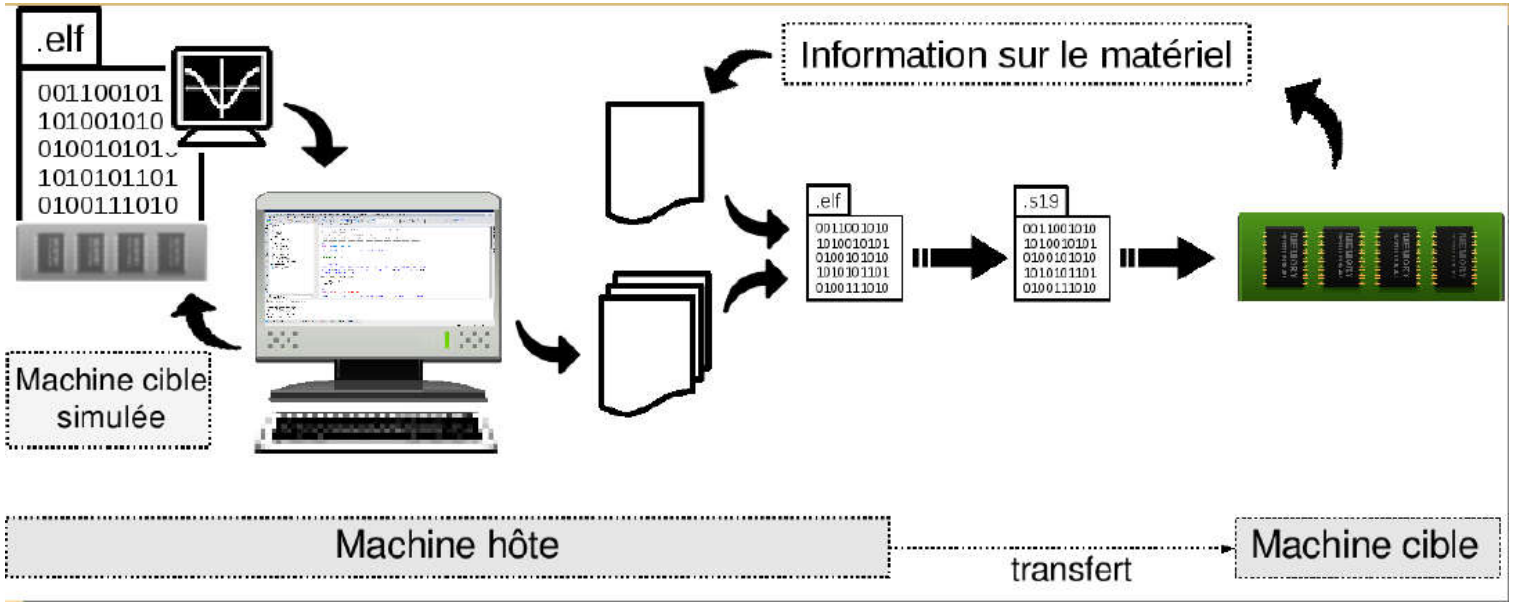
- **d'adresses**
- **de données**
- **de contrôle** (commande et état)

Quand ces circuits sont intégrés sur une même puce on parle de **microcontrôleur (µC)**.

Schéma Général (suite)



Développement sur un système embarqué



- Différence entre machine hôte / machine cible
- Développement sur machine hôte
- Test/validation avec machine cible simulée
- Cross compilation : jeu d'instruction / config matérielle
- Transfert vers machine cible (JTAG)

Programmation sans OS

- Pas d'OS sur la plupart des SE (coût, lourdeur, …)

- Programmation proche du matériel : optimisation temps, mémoire

- Mettre en place votre environnement de développement :
    - Choisir un langage de développement (Assembleur, C/C++)
    - Trouver compilateur / linker / simulateur
    - datasheets du matériel

- Gérer les entrées / sorties

- Choisir son mapping : où placer les différentes informations dans la mémoire RAM

- Bien comprendre / utiliser les registres :
    - configuration
    - utilisateurs

Programmation avec OS

- intéressant dans certains cas : programme lourd contenant beaucoup de tâches
- ordonnanceur intégré
- driver intégré
- exemple : linux (embarqué et/ou temps réel)

Exemples de SE

Microcontrôleur (comme arduino) :
- Le SE le plus courant
- Comprend un µP, des bus, de la ROM et des E/S
- pas d'OS

- Avantages : pas cher, faible encombrement et faible consommation

- Inconvénients : énormément de choix (avec plus ou moins de périphériques), traitements plutôt léger

Exemples de SE

Mini-Ordinateurs (ARM) :
-Exemples :Raspberry Pi (processeur ARM), plateforme Armadeus (processeur ARM + FPGA)
- OS Linux Embarqué

-Avantages : puissance de linux (driver, ordonnanceur intégré, …)

- Inconvénients : consommation, moins de liberté dans la programmation, la gestion de la mémoire et très bonne connaissance de Linux

Exemples de SE

FPGA (basé sur des portes logiques) :
- Circuits logiques complexes reconfigurables
- Programmation en VHDL (possible d'autres langages mais sans garantie de compilation)

- Avantages :
flexibles permettant des évolutions, plus performants qu'un logiciel, moins chers que des ASICs

- Inconvénients :
Développement long pour des applications compliquées, gourmands, moins fiables (???)

Exemples de SE

DSP : Digital Signal Processing
- SE conçu pour les applications en Traitement Signal / Images / Vidéo
- Exemples : Télévision, amplificateur HiFi, Routeurs, Radar/Sonar, …

- Avantages : précision, prédiction (simulations sur ordinateur), bibliothèques de calcul (fft, …), Cycle Multiplication/Accumulation (MAC) :
A = A+(X.Y) : 1 cycle au lieu de 4 cycles

- Inconvénients : coût élevé (inutile pour des réalisations simples), complexité (optimisation du calcul)

Plan du Cours

I. Introduction au langage C

II. Introduction au Systèmes Embarqués

III. Principes du Processeur

IV. Entrées/Sorties

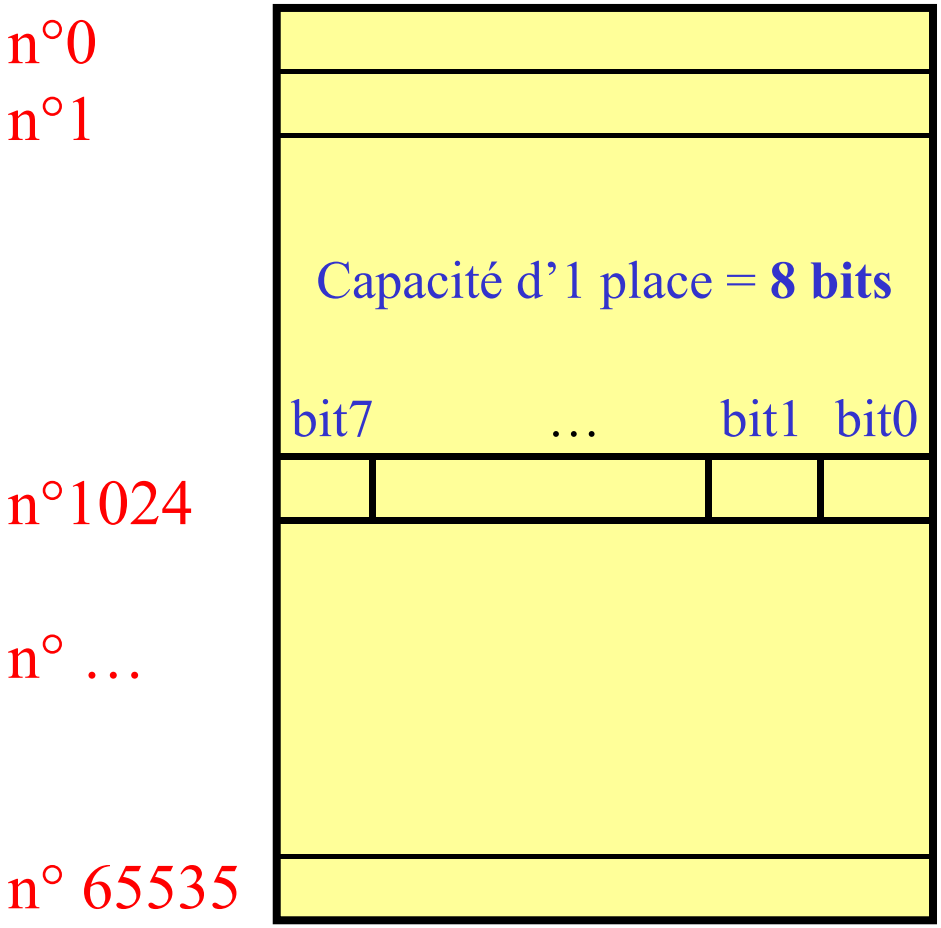
V. Arduino

Principes d'un processeur

- **Un processeur** exécute des ***programmes***.
- ***Un programme*** exprime la solution d'un problème (décrit par un algo.) par une ***suite d'instructions*.**
- Les ***programmes exécutables*** et les ***données traitées*** sont stockés *(en binaire*) en **mémoire principale**.

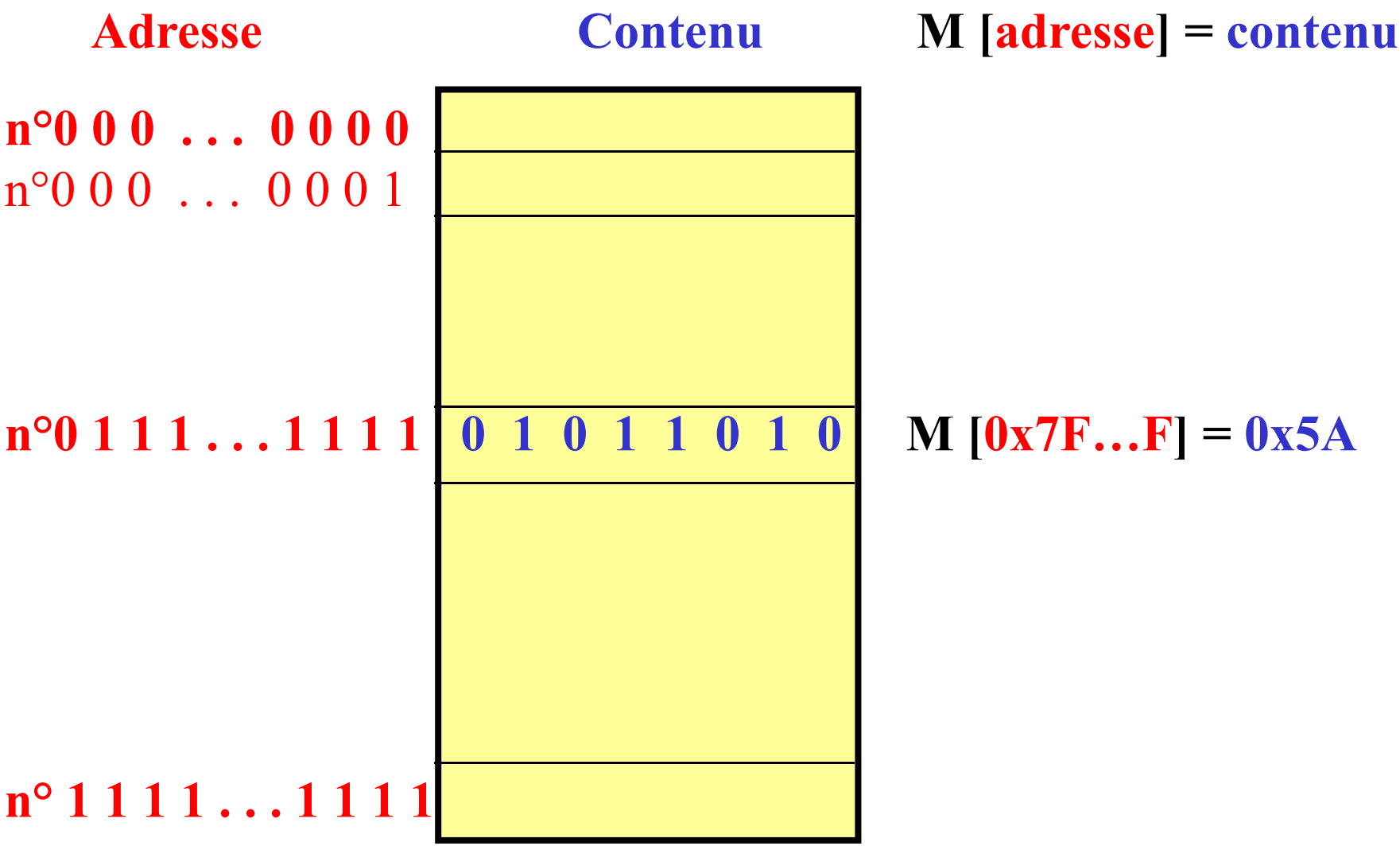
La Mémoire Principale

Représentation schématique Mémoire

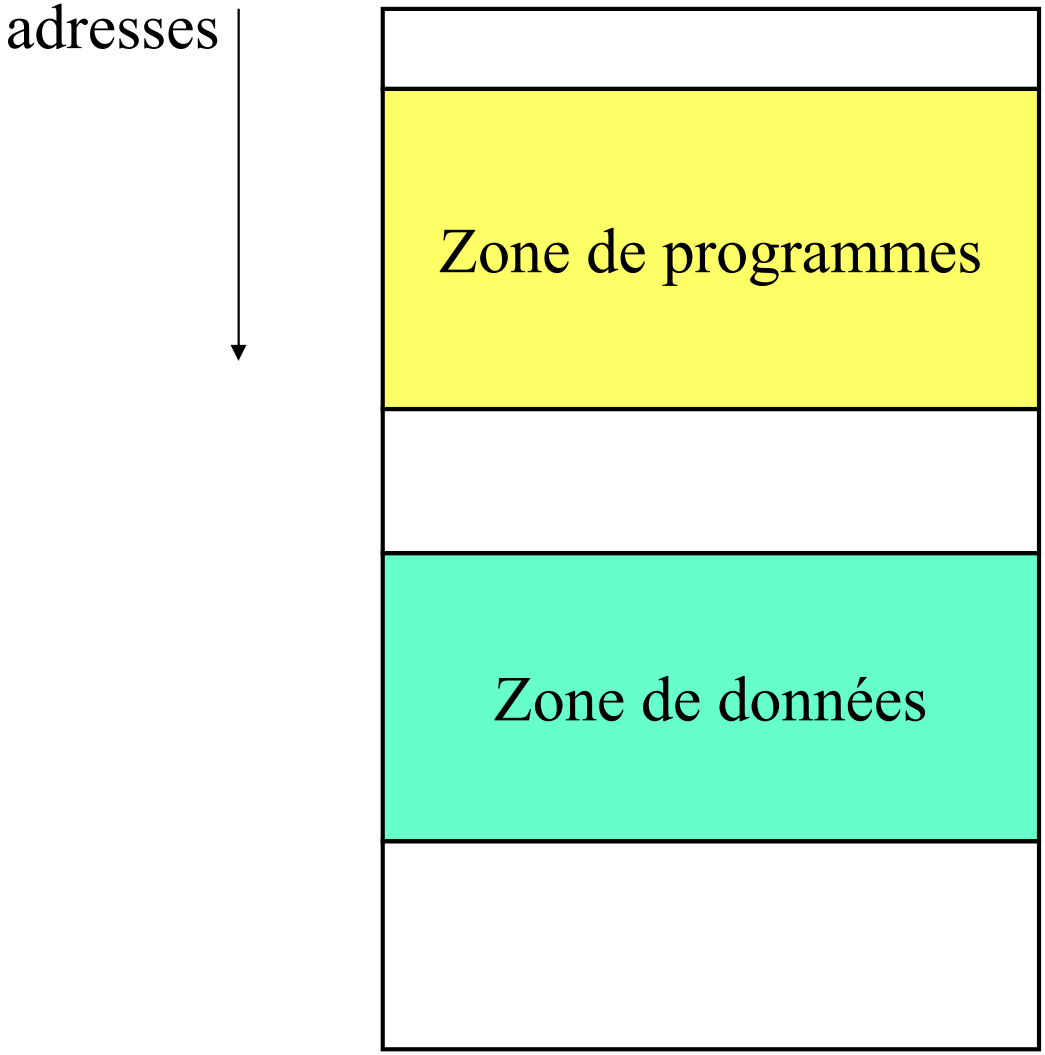


Exemple :
64 kilo octets

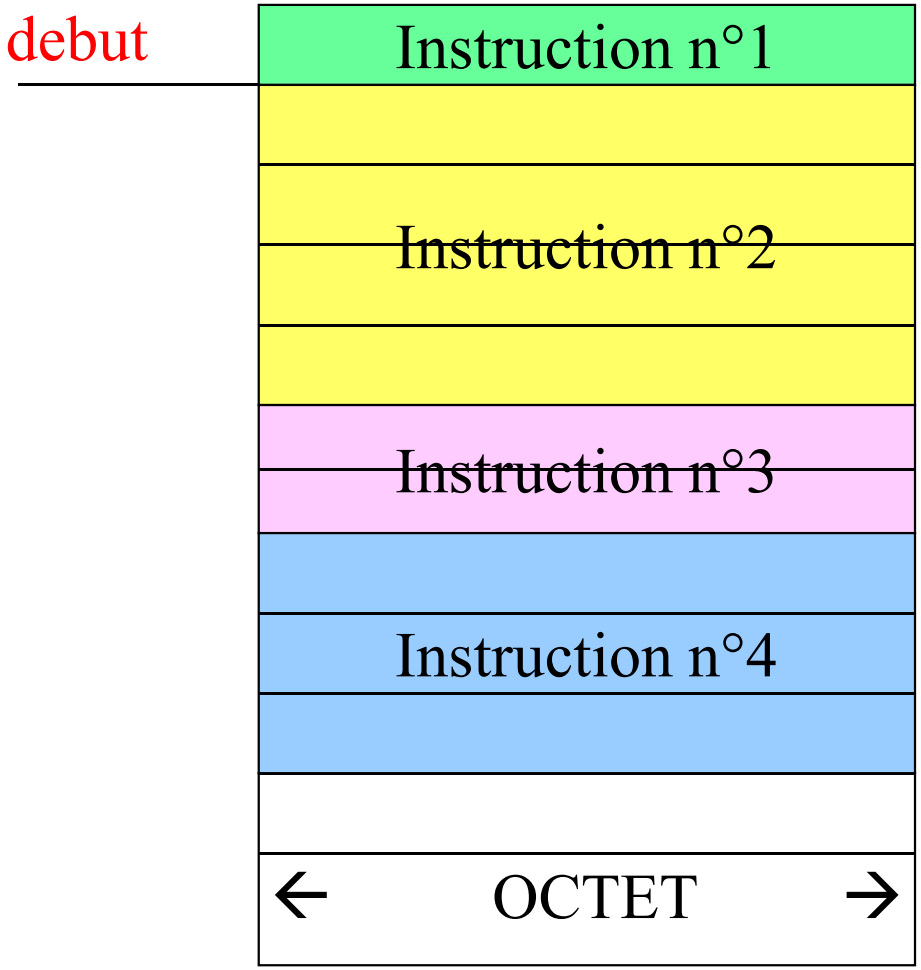
Représentation schématique Mémoire (suite)



Organisation de la mémoire



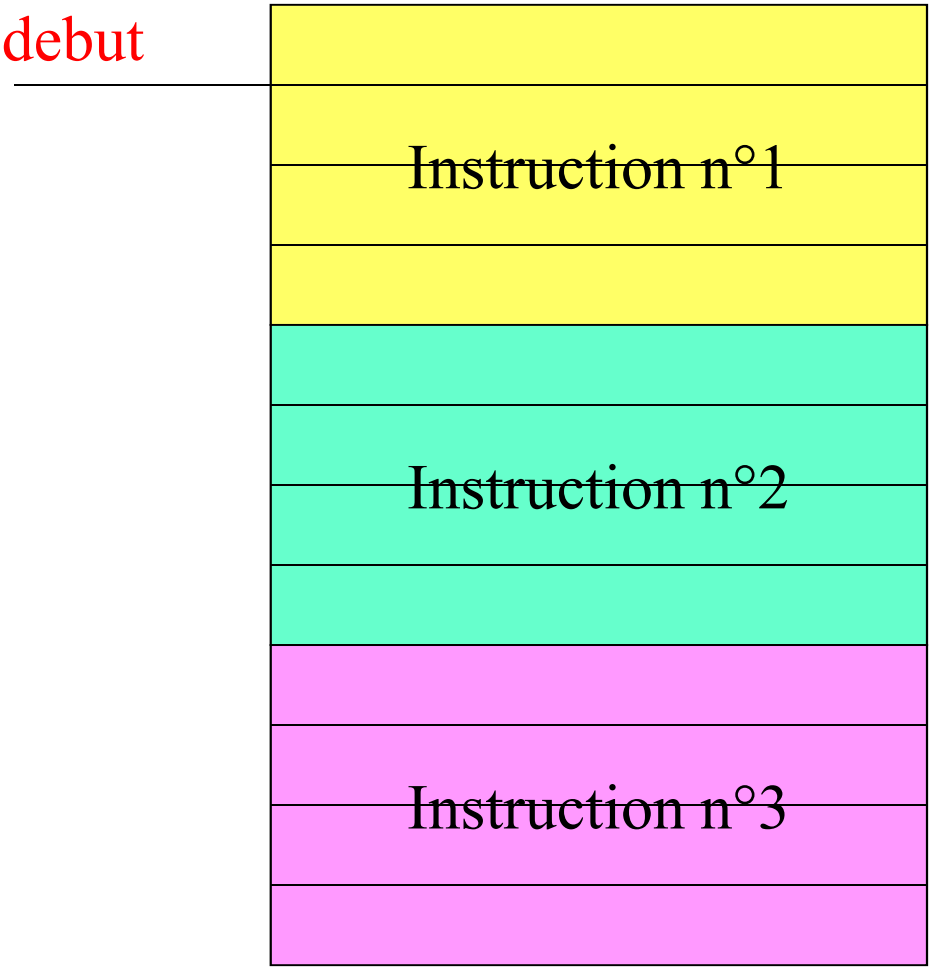
Organisation de la mémoire – Zone Programmes
Cas des processeurs CISC



Instructions de tailles différentes et peut être > 4 octets

Instructions complexes

Organisation de la mémoire – Zone Programmes
Cas des processeurs RISC

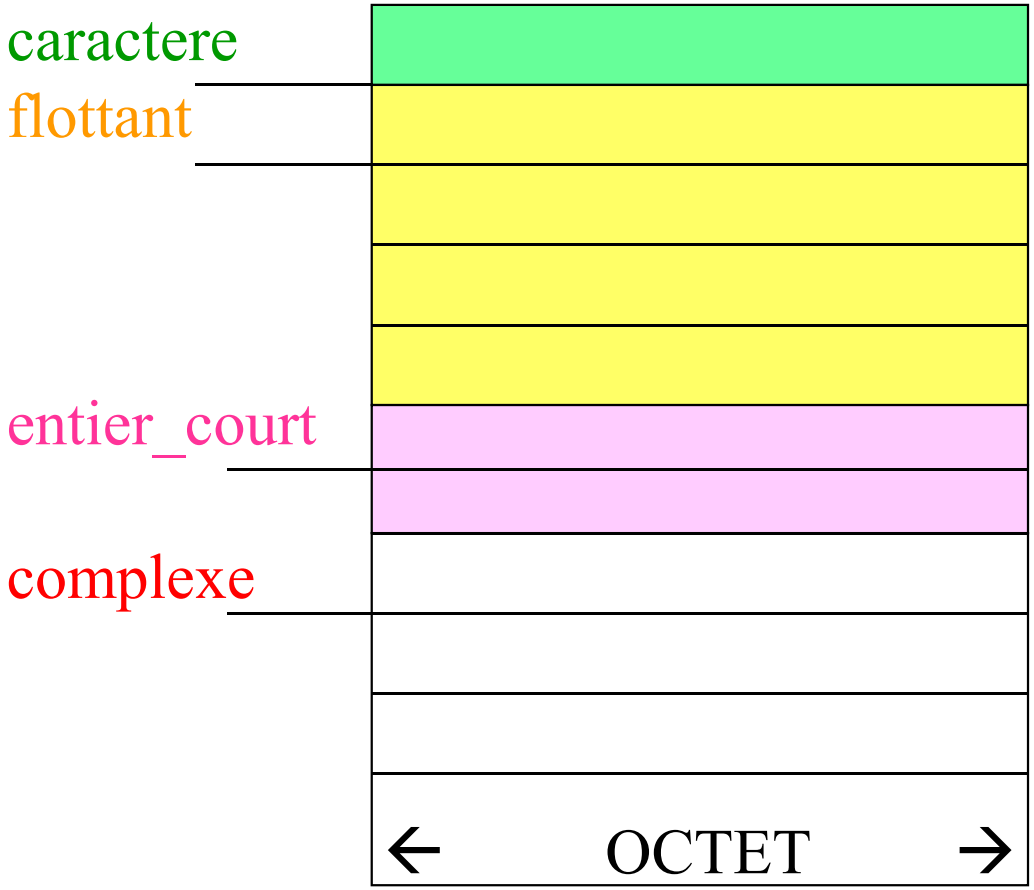


Toutes les instructions ont la même taille : 4 octets

Mais instruction de bas niveau

Organisation de la mémoire – Zone des données

Les données n'ont pas toutes la même taille.



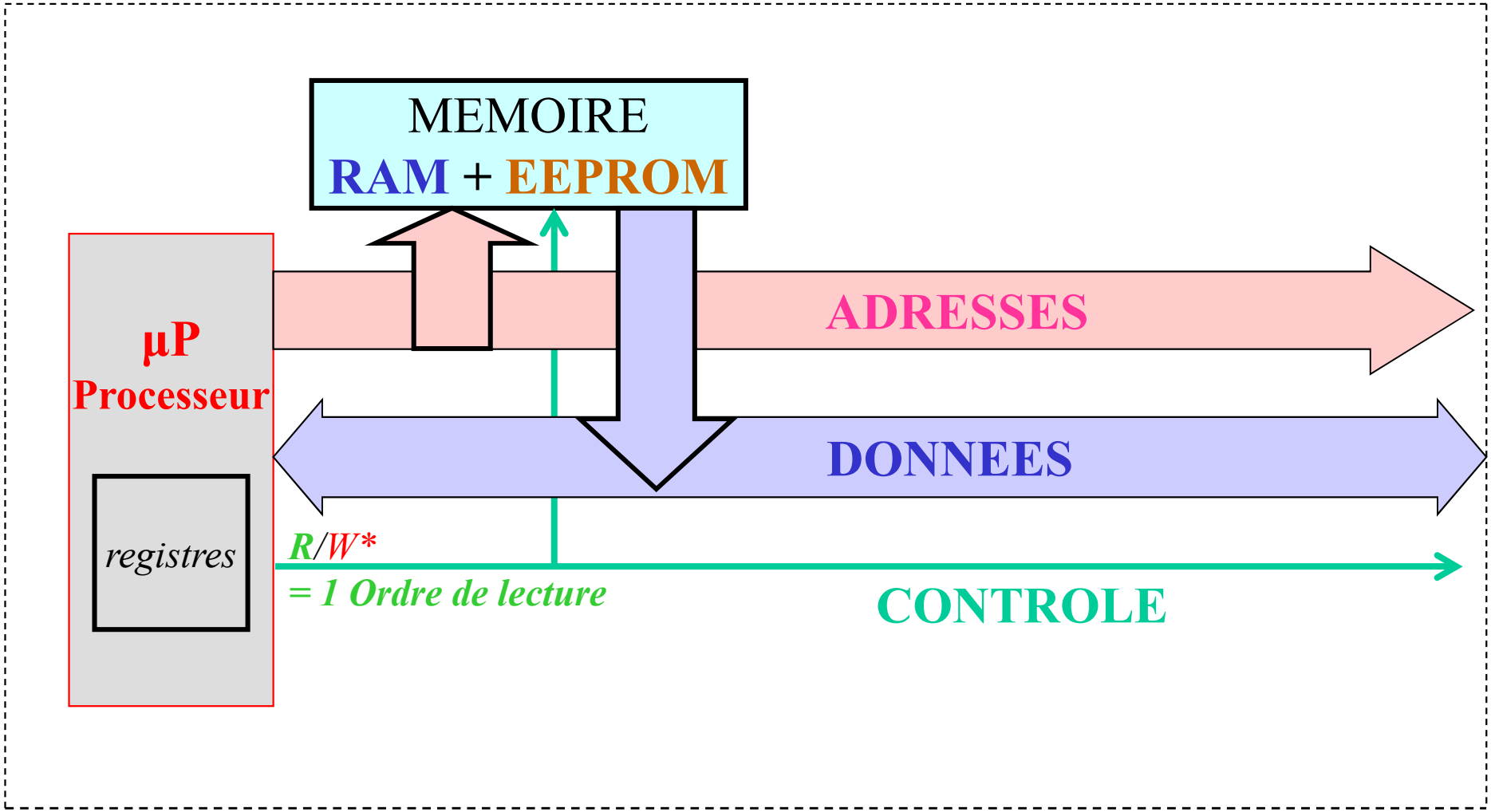
L'adresse d'une donnée est celle du premier octet.

Les BUS

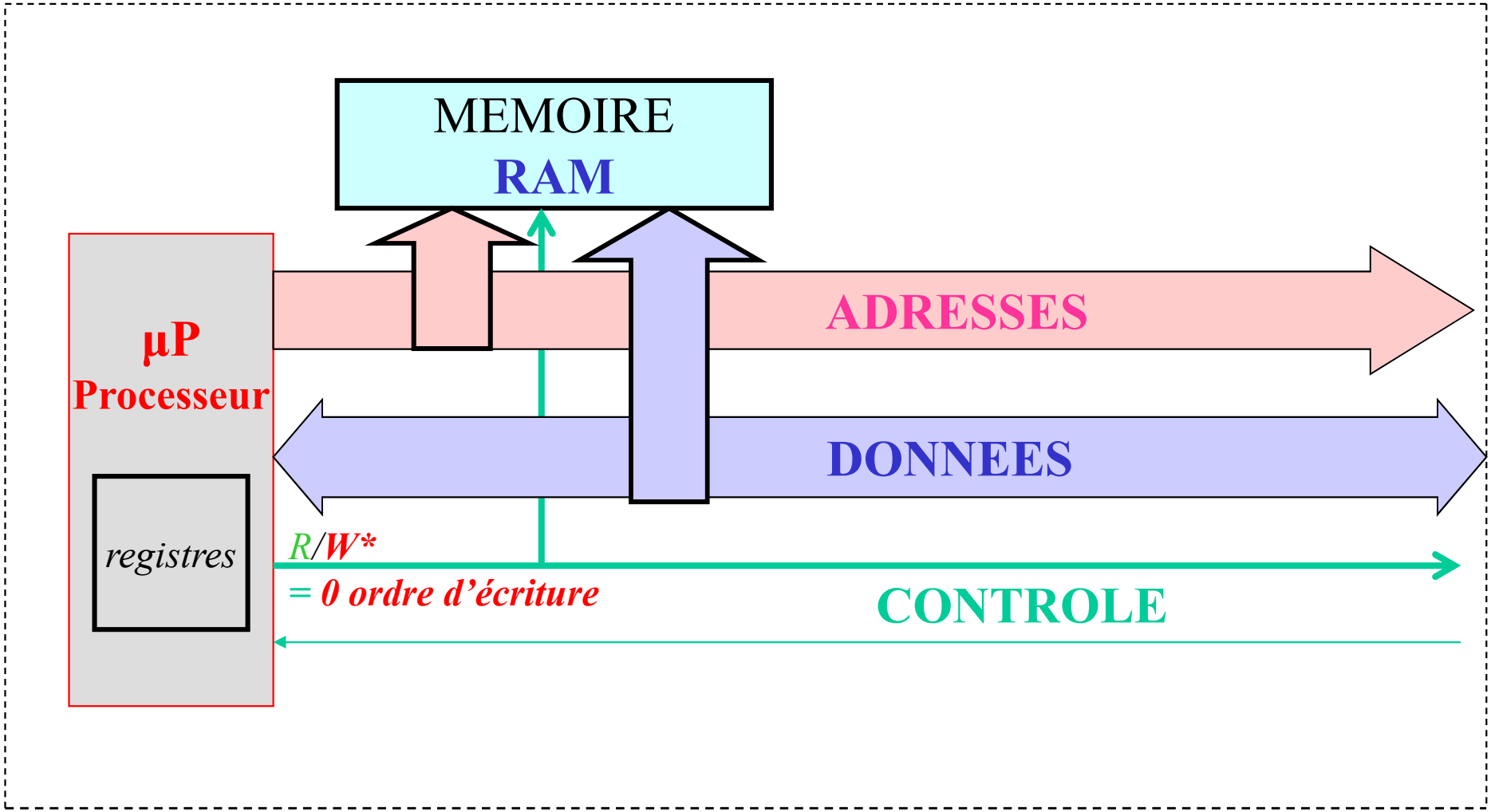
- Les **bus** (d'**adresses**, de **données**, de **contrôle**) véhiculent les infos entre µP et mémoire ou entre µP et coupleur d'E/S.
- Le µP peut **lire (écrire)** en mémoire ou dans un coupleur d'E/S en donnant :

  - **l'adresse** de la place choisie
  - une commande de **lecture (écriture)**
  - (**la valeur** de la donnée à écrire).

Accès en Lecture



Accès en Ecriture



Exemple à Traiter

Dans un programme, on a besoin de mettre à jour la valeur d'une variable ***indice*** de type entier court (16 bits).

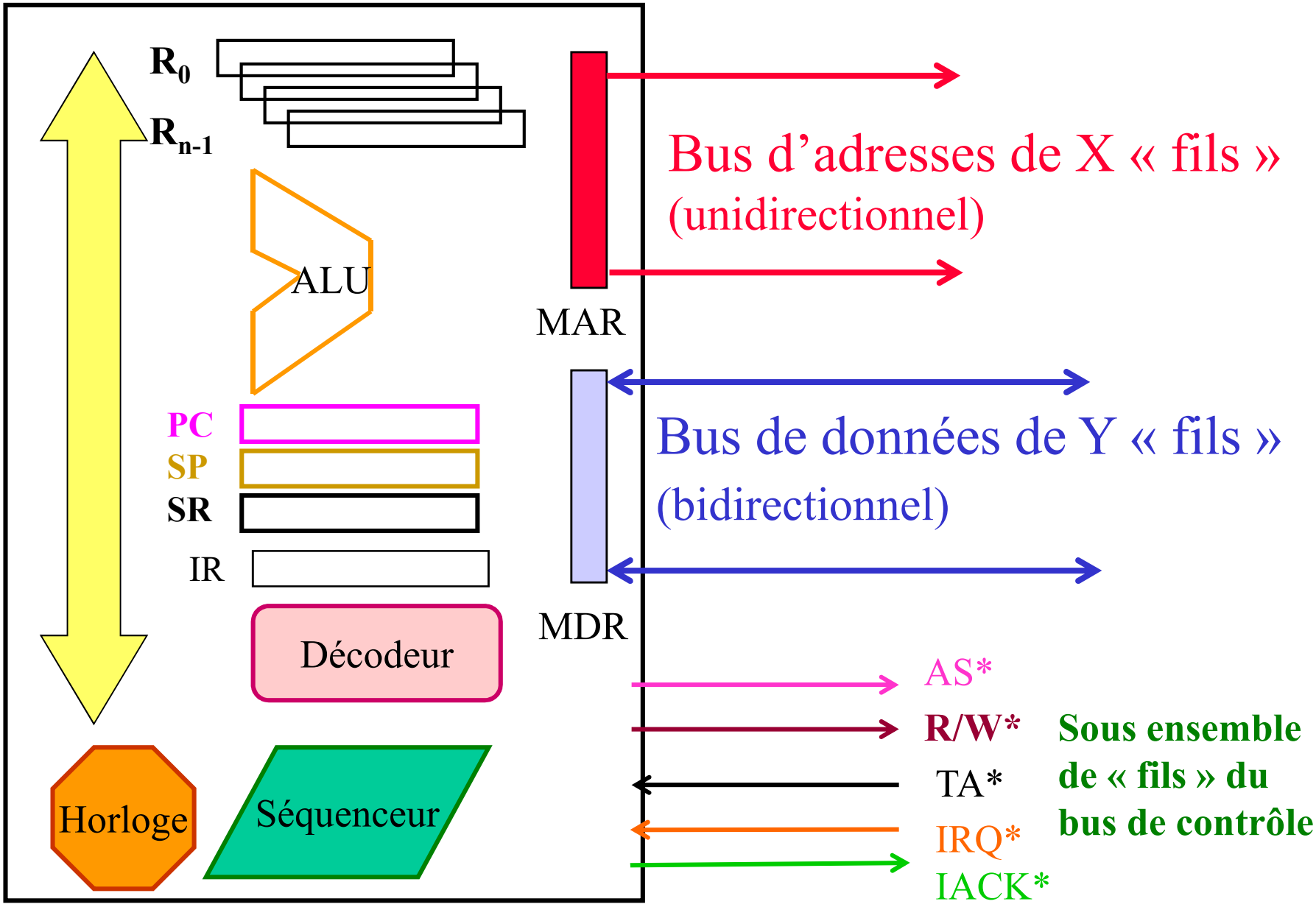
indice ← *indice* + 1

On suppose que la taille du bus de données est

a) 8 bits ;
b) 16 bits ;
c) 32 bits.

Combien d'accès à la mémoire (et de quel type), le µP doit il faire dans chaque cas ?

Le Processeur (« CPU »)



Processeur : horloge

- L'activité du processeur est cadencée par une **horloge** Clk (« clock »)

- La rapidité d'exécution du processeur est fonction de :

  - la **fréquence** de Clk (en MHz ou GHz)

  - la **taille** de l'information élémentaire traitée (µP 8 bits, 16 bits, 32 bits, 64 bits)

Processeur : registres

Le processeur possède des **registres internes** (8, 16, 32 bits) (mémoire de petite capacité) pour stocker temporairement les informations nécessaires à l'exécution d'un programme.

Pour traiter les informations, **il fait des transferts (*duplications*)**

- entre une case mémoire ou E/S et un registre **(*lecture : Read)***
- entre un registre et une case mémoire ou E/S **(*écriture : Write)***
- entre registres internes.

Certains registres sont spécialisés :

- le registre compteur du programme **PC (Program Counter)**
- le registre pointeur de pile **SP (Stack Pointer)**
- le **registre d'état** (drapeaux de retenue, signe, débordement, IT...)

Processeur : fonctionnement

- Le μP répète sans arrêt ces actions :

  - **Lire** en mémoire principale l'instruction pointée par le registre PC.

  - **Décoder** l'instruction lue.

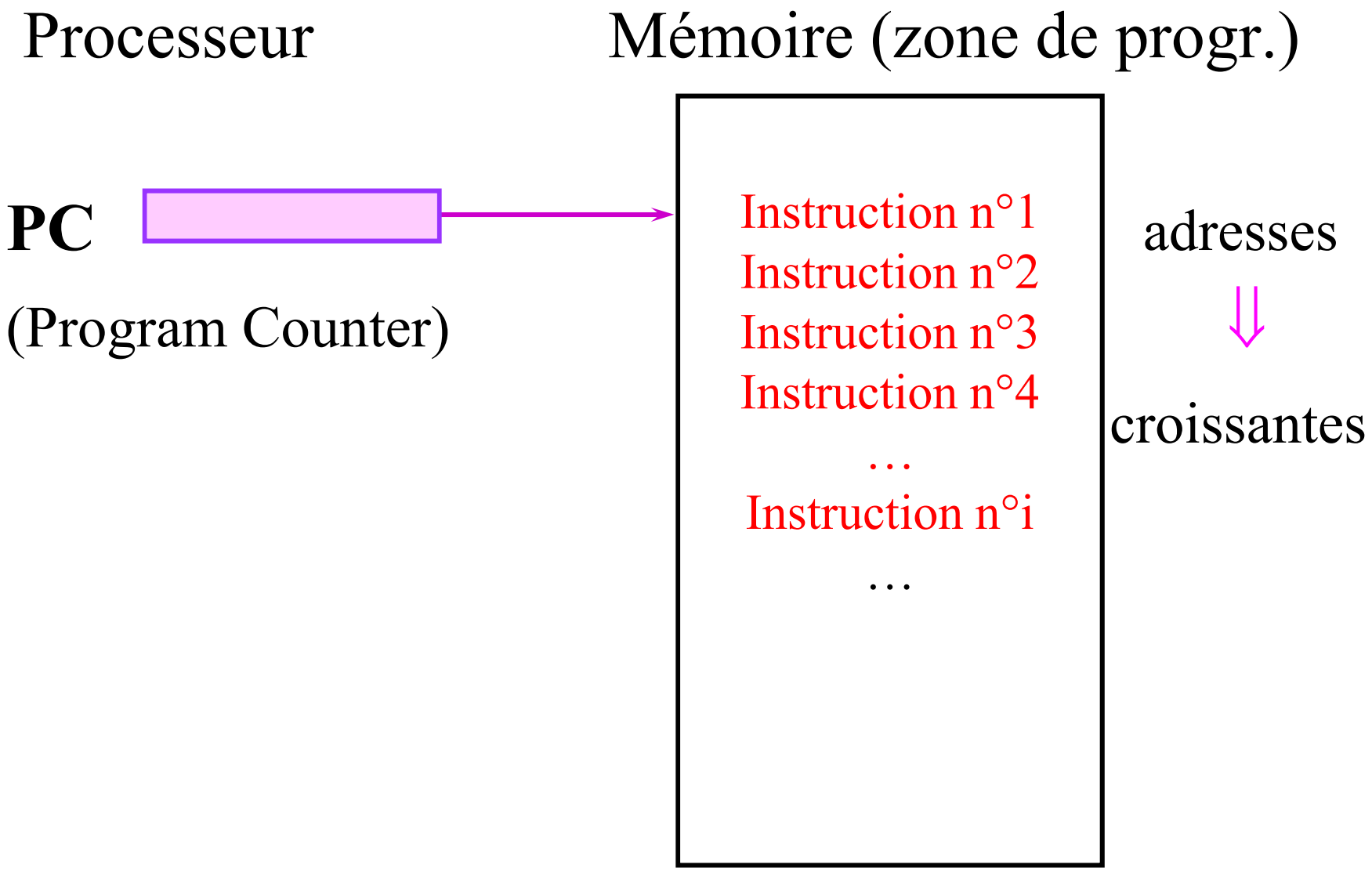
  - **Mettre à jour** le PC (PC ← PC + n).

  - **Obtenir** donnée(s) nécessaire(s).

  - **Exécuter** l'instruction.

- Rq : n = taille en octets de l'instruction

Illustration: l'accès au programme



Processeur : fonctionnement

- La plupart du temps, les instructions sont exécutées dans l'ordre de rangement (adresses consécutives).

- On peut *modifier le déroulement du programme* par des instructions de saut (*rupture de séquence*)

- Une instruction de saut est un ordre de branchement

  à une autre adresse

  - du même programme : *if, while, ...*
  - d'un autre programme : sous programme, fonction, …

Processeur : fonctionnement

Toute instruction de saut est une modification du PC
(*changement de l'adresse de l'instruction à traiter*)

ancienne valeur du PC = adresse de la suite prévue
nouvelle valeur du PC = adresse de la suite effective.

⇒ Saut à l'intérieur du même programme:
l'ancienne valeur du PC est perdue, remplacée par la nouvelle.

⇒ Saut à un sous progr. (fonction/procédure):
l'ancienne valeur du PC (ou adresse de retour) est mise de coté avant d'être remplacée par la nouvelle.
Remarque : A la fin du sous progr. il faut retourner au progr. appelant.
On utilise une instruction de retour (saut particulier), qui récupère l'adresse de retour mise de coté et la met dans le PC.

Exemple sous-programme : diviser

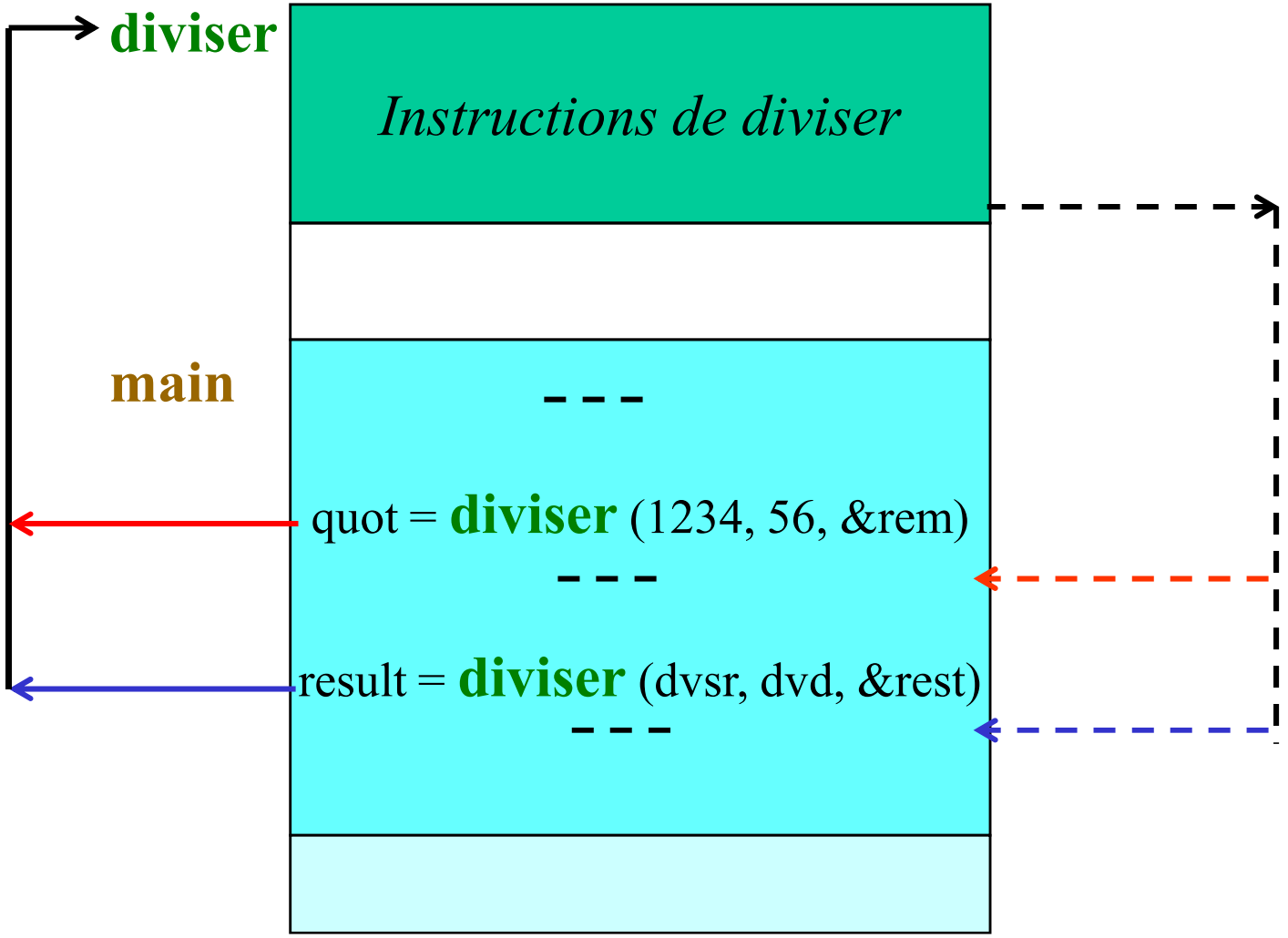
Fonction ***diviser*** **(A, B, *R)** qui retourne **Q**

A, B, R, Q de type entier non signé

Q ← 0	*écrire 0 dans* Q
Tant Que A ≥ B Faire	**tq:** *comparer* A à B
	***sauter si* A < B** *à* **ftq**
A ← A – B	*retrancher* B *à* A
Q ← Q + 1	*ajouter 1 à* Q
	sauter toujours *à* **tq**
Fin Tant Que	
R ← A	**ftq:** *copier* A *dans* R
retourner Q	**?** ***retourner*** Q

…

Exemple sous-programme : diviser



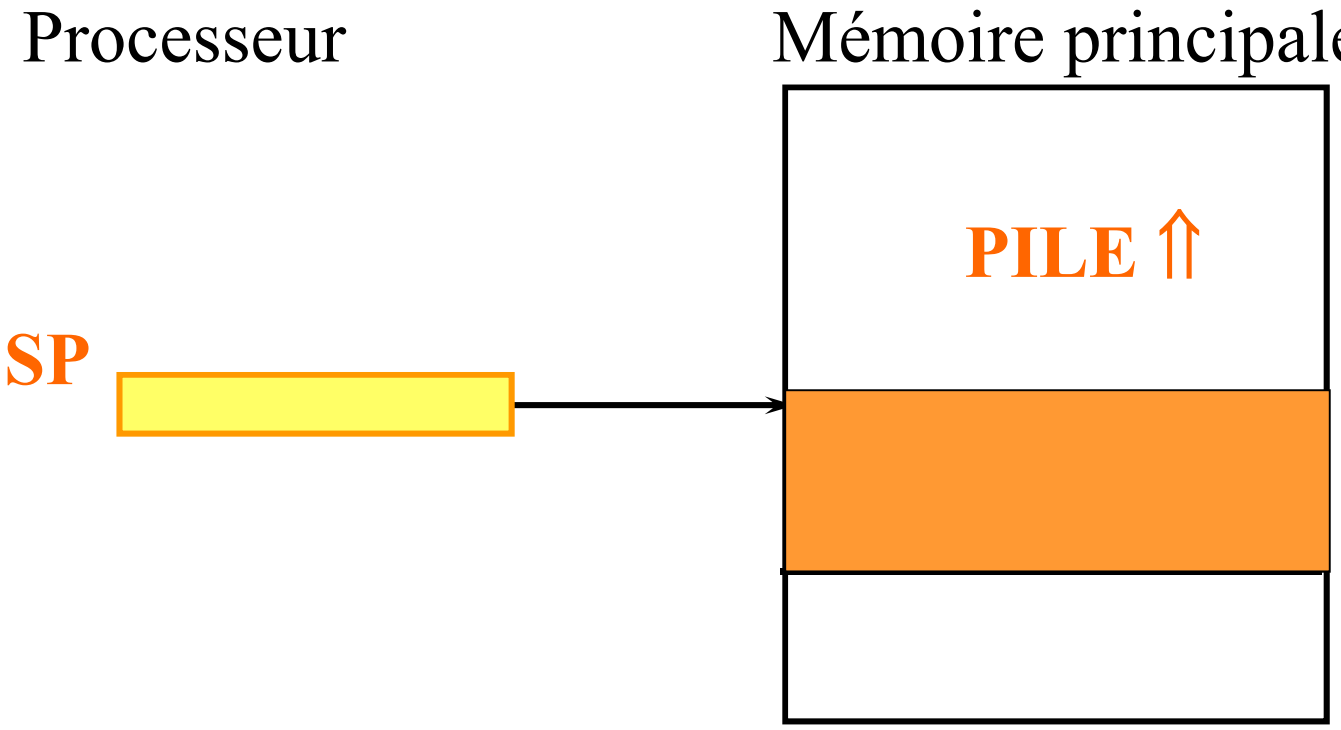
Retenir adresse retour programme principal

La pile et le pointeur de pile

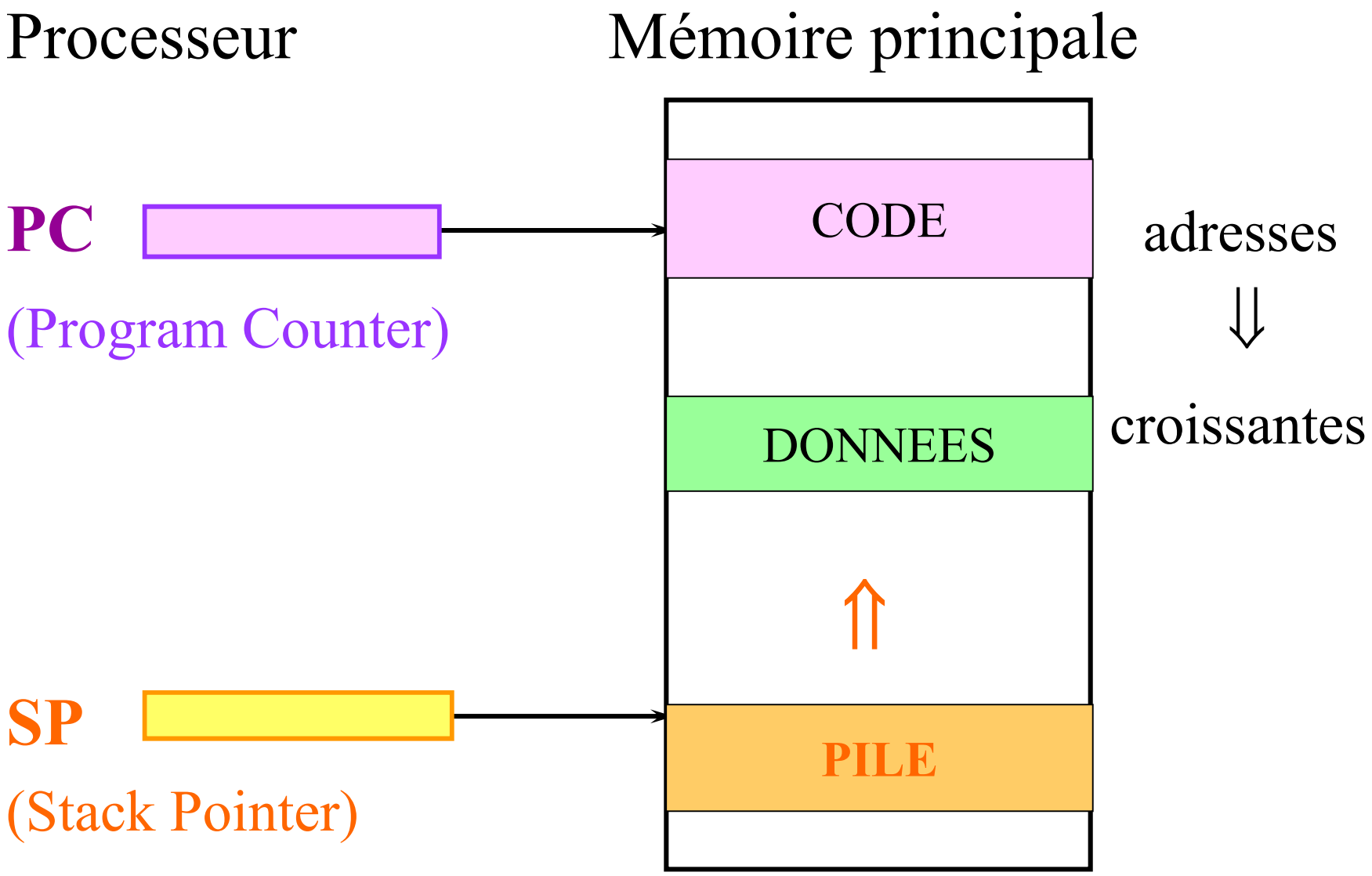
• Le µP a accès à une zone de mémoire : **la pile (« Stack »),**

➡ dédiée au stockage d'infos essentielles, ***adresses de retour***, ***paramètres***, ***variables locales***, …

• Le registre SP (« Stack Pointer ») contient l'adresse de la dernière info empilée (sommet de la pile).



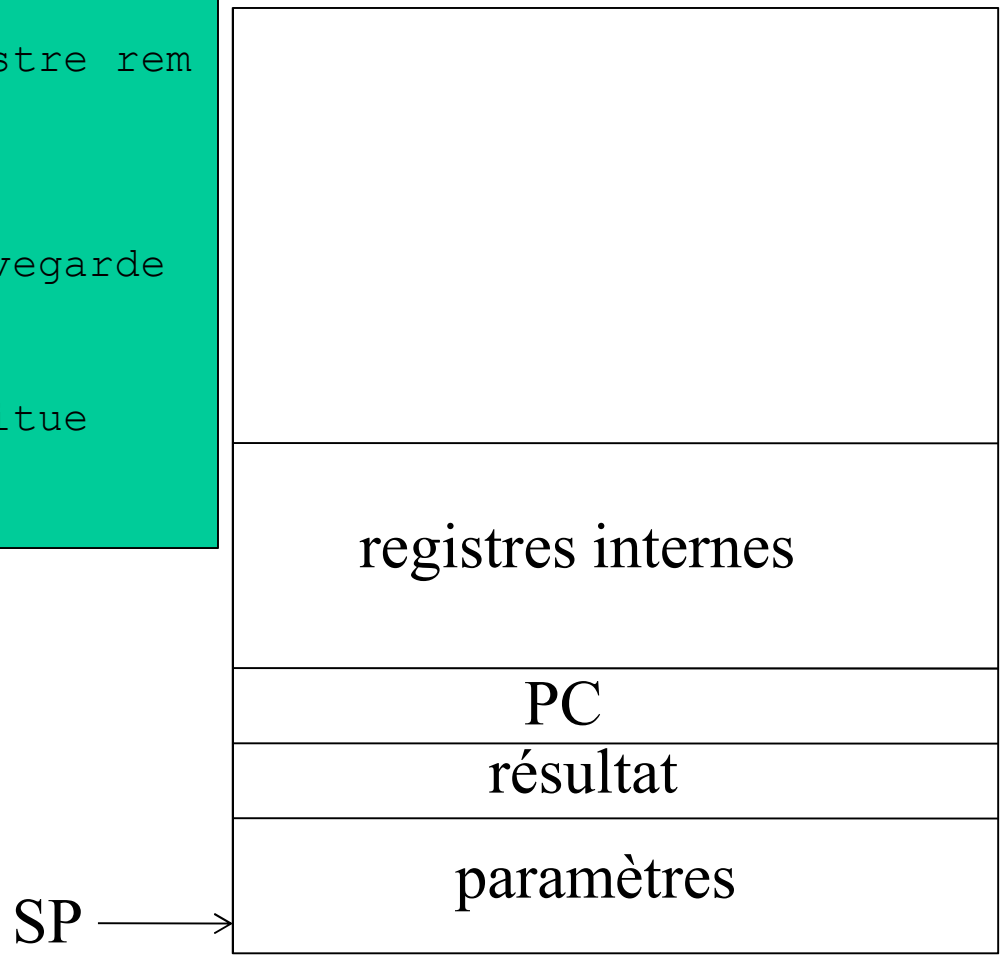
Organisation de la Mémoire



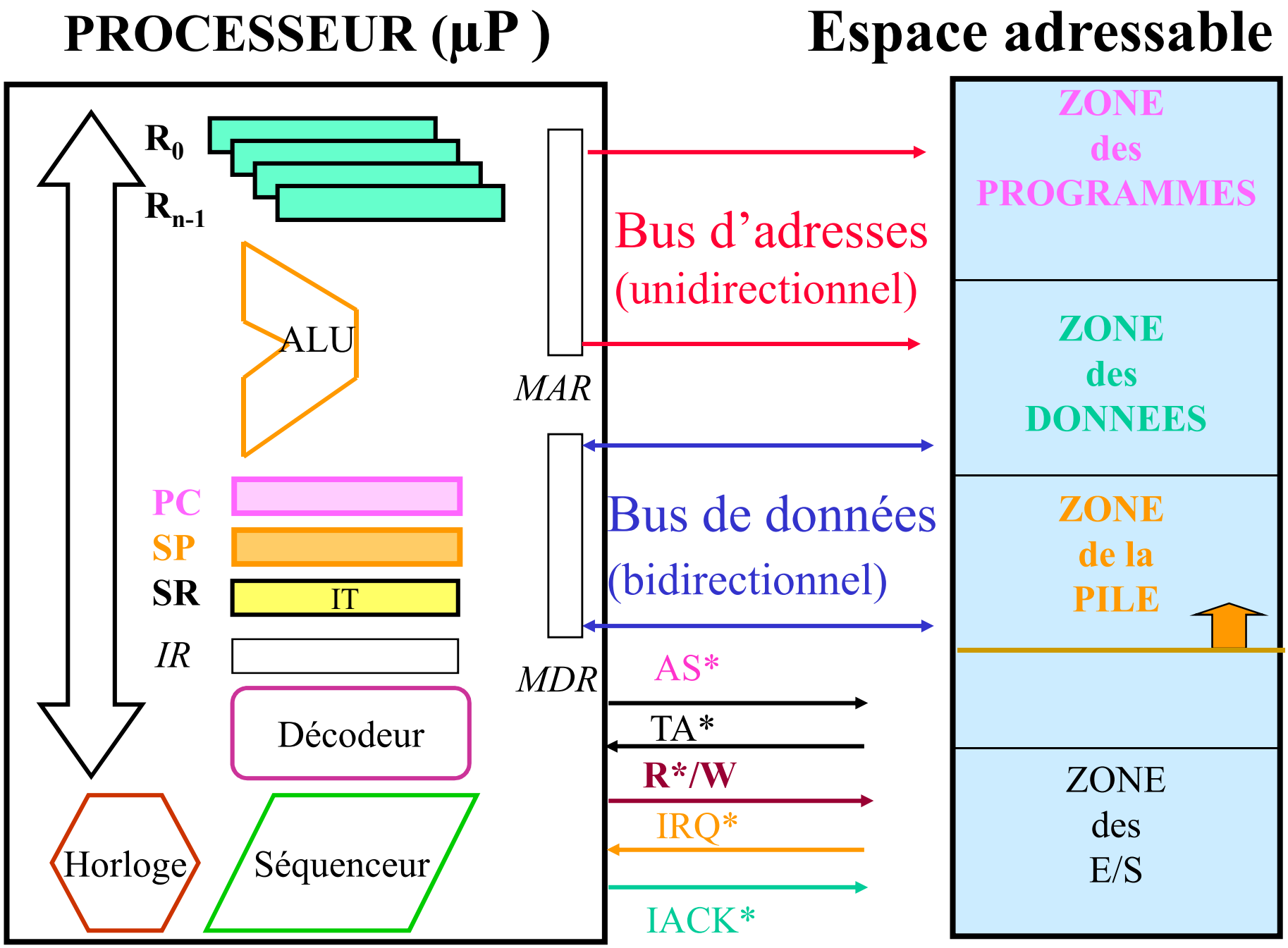
Exemple sous-programme : diviser

```
main :

push(1234,54)// parametres fct
SP ← SP - 4 // taille résultat rem
BSR diviser
pop rem // recupere ds un registre rem
SP ← SP + taille parametres

diviser :
push registres internes // sauvegarde
contexte
…
pop registres internes // restitue
contexte
```

quot = **diviser** (1234, 56, &rem)



- push x : sauvegarde x dans la pile et décrémente automatiquement SP
- pop x : récupère x dans la pile et incrémente automatiquement SP


PROCESSEUR (µP)
Espace adressable
R0
Rn-1
ALU
PC
SP
SR
IR
IT
Décodeur
Horloge
Séquenceur
MAR
MDR
Bus d'adresses
(unidirectionnel)
Bus de données
(bidirectionnel)
AS*
TA*
R*/W
IRQ*
IACK*
ZONE
des
PROGRAMMES
ZONE
des
DONNEES
ZONE
de la
PILE
ZONE
des
E/S

Plan du Cours

I. Introduction au langage C

II. Introduction au Systèmes Embarqués

III. Principes du Processeur

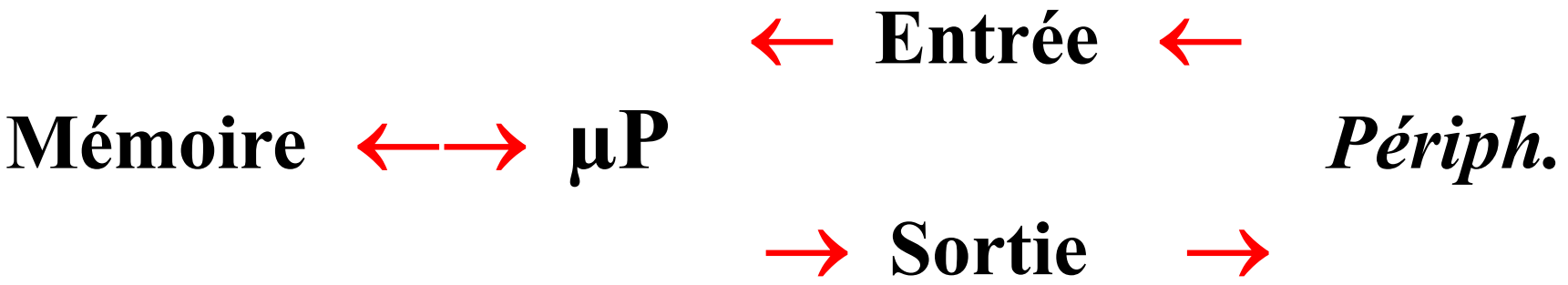
IV. Entrées/Sorties

Traitement Informatique
CAN et CNA / Liaison Série / Timer

V. Arduino

Généralités sur les coupleurs d'Entrées/Sorties

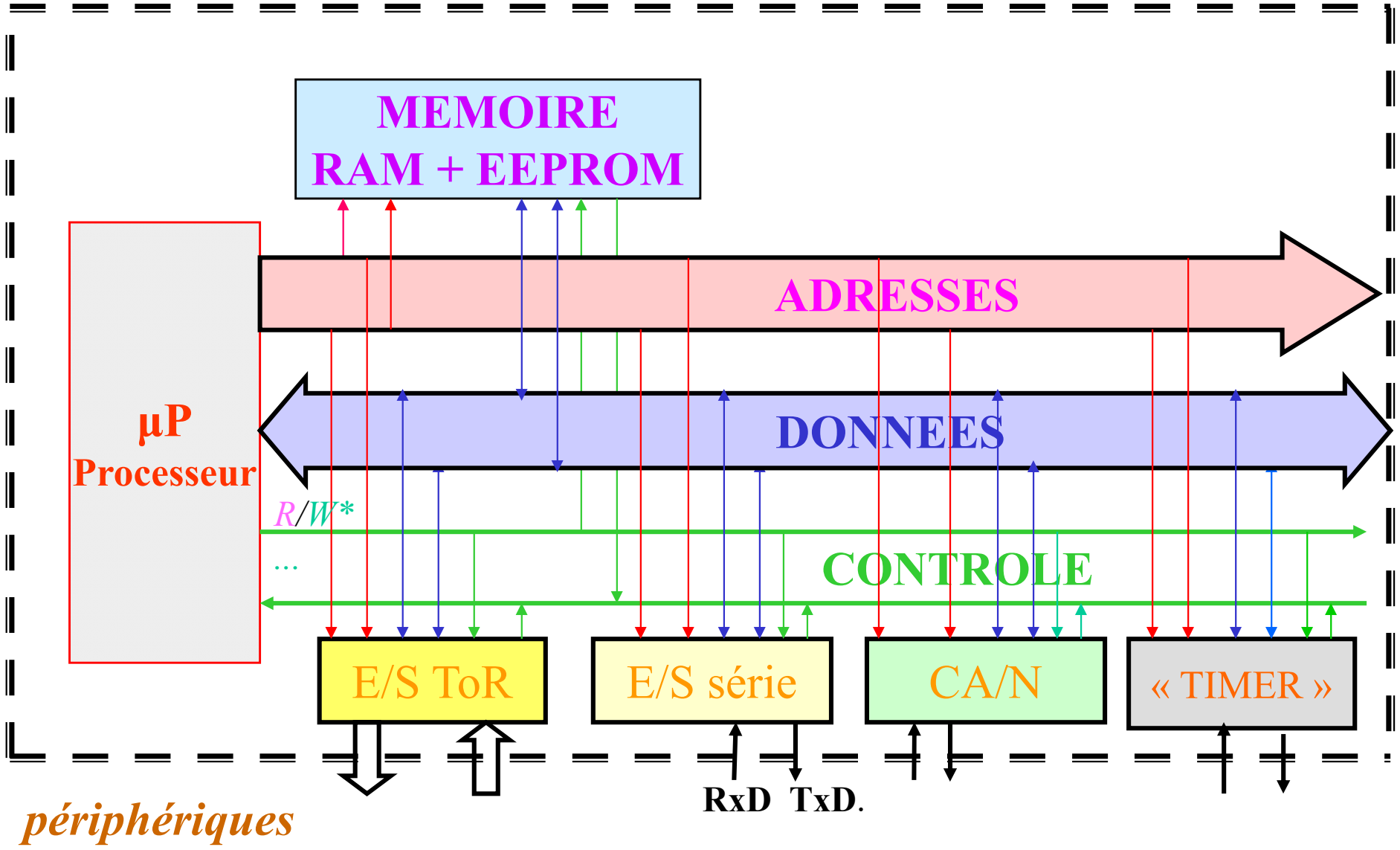
Les coupleurs d'E/S sont des circuits **tampons** entre les périphériques et le µP



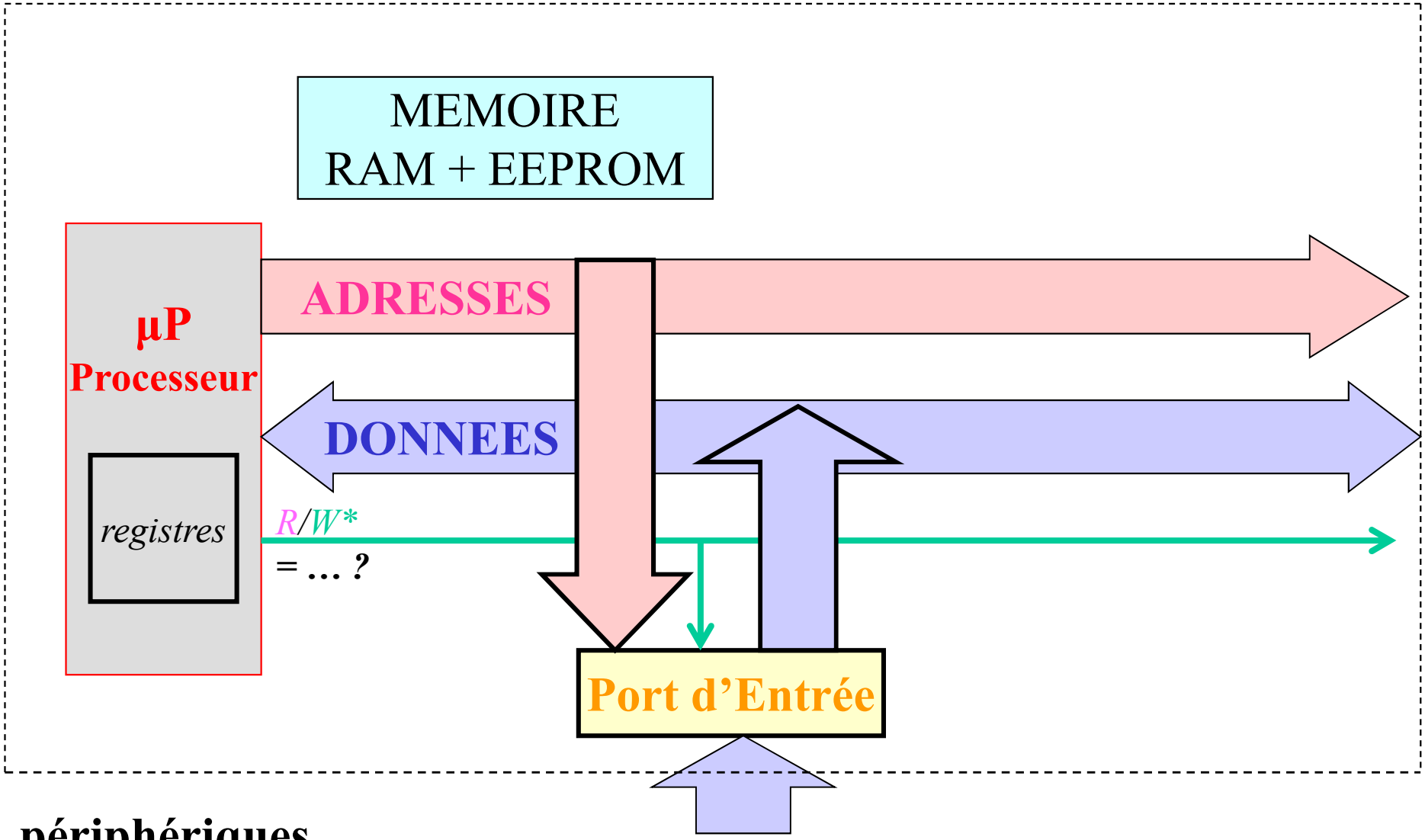
Entrée : le **µP lit** une donnée (que le ***périph*** **a écrite**).
Sortie : le **µP écrit** une donnée (que le ***périph*** **lira**).

Mémoire : le **µP lit ou écrit** une donnée.

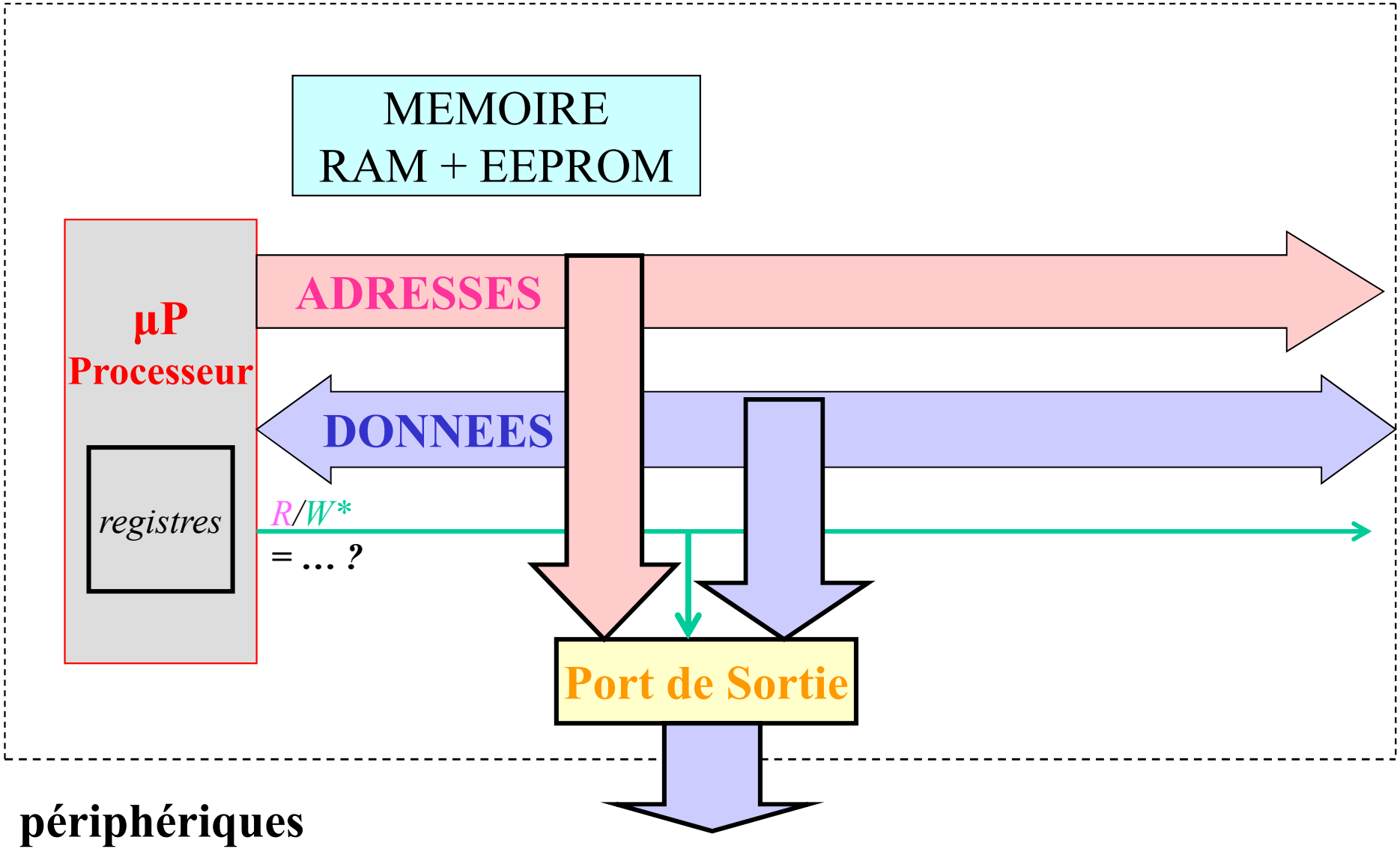
Schéma général des E/S



Lecture port d'entrée



Ecriture sur le port de sortie



Synchronisation des accès E/S

- **µC** et **périph.** traitent les données à des cadences très différentes.

- Il faut associer aux données des signaux de synchro. (analogie avec *feu rouge ou vert*) pour adapter le débit à l'élément le plus lent.

⇒ On doit **éviter au µC** (l'élément le plus rapide) de **relire** (plusieurs fois) une même donnée, **écraser** une donnée (avant qu'elle ait été lue).

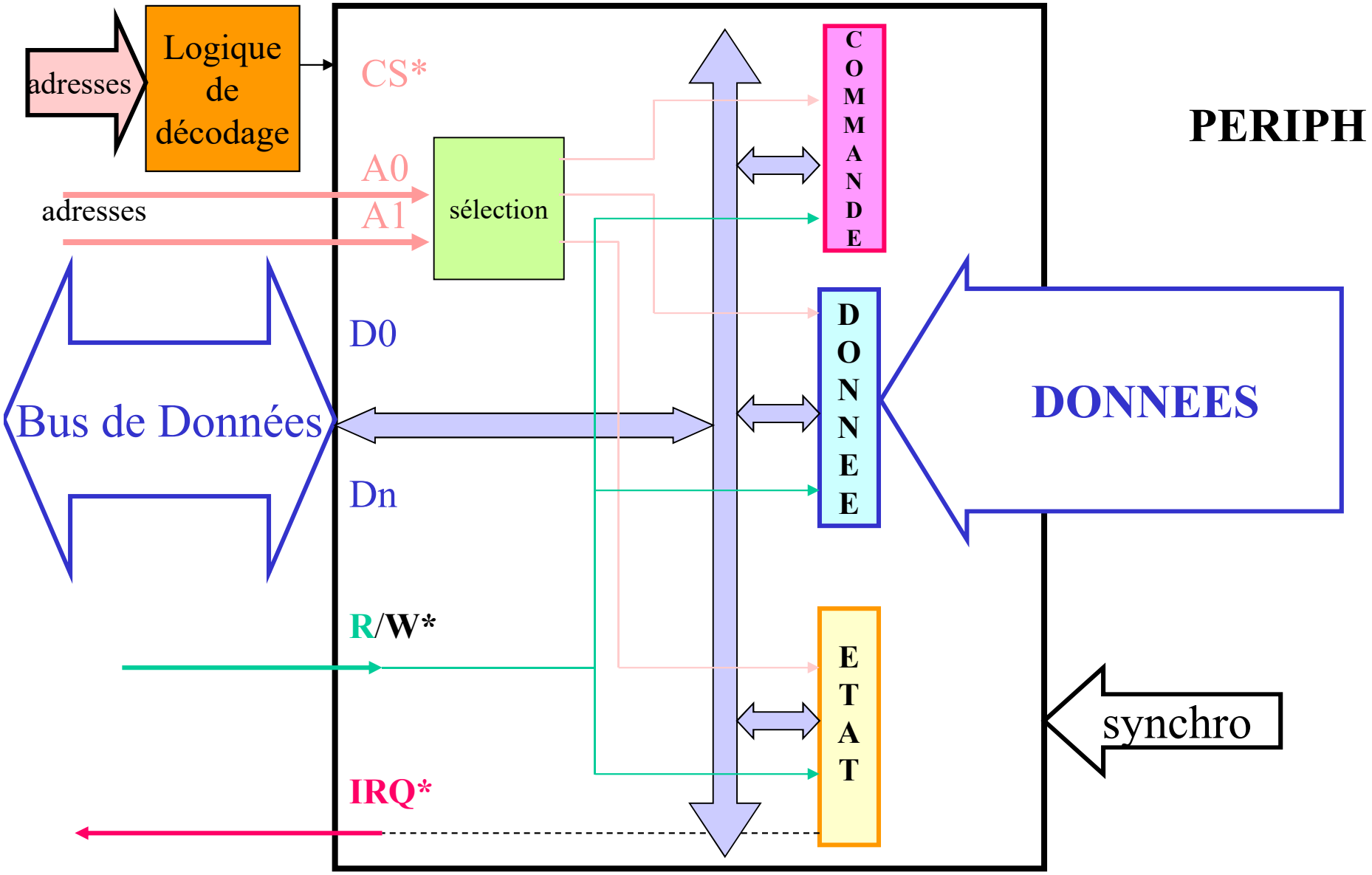
E/S Programmables

• Pour s'adapter à **la grande diversité** des périphériques et aux problèmes de synchronisation, les circuits d'E/S ne se limitent pas à des tampons de données.

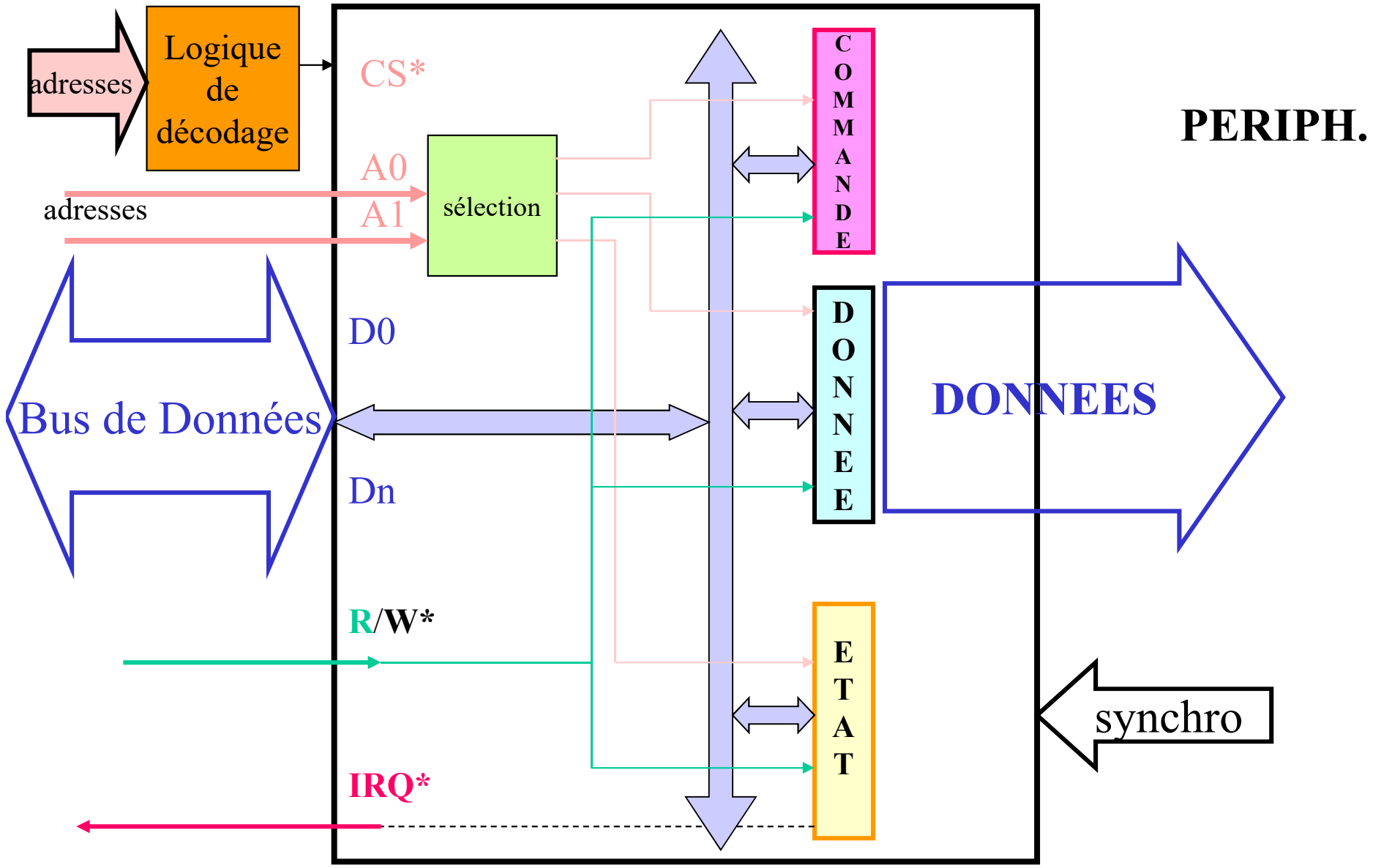
• Ils sont **configurables par l'utilisateur** qui devra choisir un ***mode de fonctionnement*** parmi plusieurs.

• Ils sont constitués de 3 types de registres
  - registres de **commande (« control register »)**
  - registres d'**état (« status register »)**
  - registres tampons de **données** (d'entrée ou de sortie).

E/S Programmables : Modèle



E/S Programmables : Modèle



E/S Programmables : Utilisation

⇒ Le **MODE de FONCTIONNEMENT**

écrit par le µP dans le **registre de Commande** à l'initialisation

(analogie avec une rangée d'interrupteur).

⇒ Les **INFOS de SYNCHRONISATION et/ou d'ERREUR**

lues par le µP dans le **registre d'Etat**

écrites, au niveau bit, par le matériel

(analogie avec des voyants).

⇒ Les **DONNEES ECHANGEES**

- lors d'une opération de **SORTIE** sont :

  écrites par le **µP** dans un **registre de Sortie** (d'Emission)

  lues ensuite par le **périph.**

- lors d'une opération d'**ENTREE** sont :

  écrites par le **périph**. dans un **reg. d'Entrée** (de Réception)

  lues ensuite par le **µP.**

Traitement informatique des E/S

• Un circuit d'E/S peut être considéré comme **une « variable » de type enregistrement** avec plusieurs champs :

- **commande (« control »)**
- **état (« status »)**
- **données (« data »)**

• PARTICULARITES :
L'adresse de cette « variable » (du circuit) est **figée.**

• **Certains champs** (registres) peuvent
- avoir un **accès limité** (***en lecture ou en écriture seulement***)
- être constitués de **groupes de bits** indépendants les uns des autres.

Programme des gestions des E/S

• Une opération d'Entrée/Sortie est effectuée par un programme spécifique **lié au type de périph et à la forme de l'échange** : **pilote (« driver »).**

• Ensembles de fonctions de mise en œuvre d'opérations d'E/S
  • Fonction d'**initialisation**
  • Fonction d'**écriture** (entrée)
  • Fonction de **lecture** (sortie)
  •…

Techniques de Gestion des E/S

⇒ **E/S par INTERROGATION ou SCRUTATION** (« status driven »)

Le pilotage des échanges se fait par le programme :
avant l'émission (ou la réception) de données, le µP doit s'assurer
et ***donc attendre*** **que l'échange soit possible** pour enfin émettre
(ou recevoir) la donnée.

⇒ **E/S par INTERRUPTION** (« interrupt driven »)

Le déclenchement de l'échange est sous la responsabilité du périph.
Quand l'échange est possible, un signal d'**IT** est envoyé au **µP** :
le programme exécuté par le µP est ***interrompu,***
le temps de traiter l'émission (ou la réception) de données.

Gestion des E/S par Interrogation

• Avant de faire un nouveau transfert d'information, il faut vérifier si c'est possible :

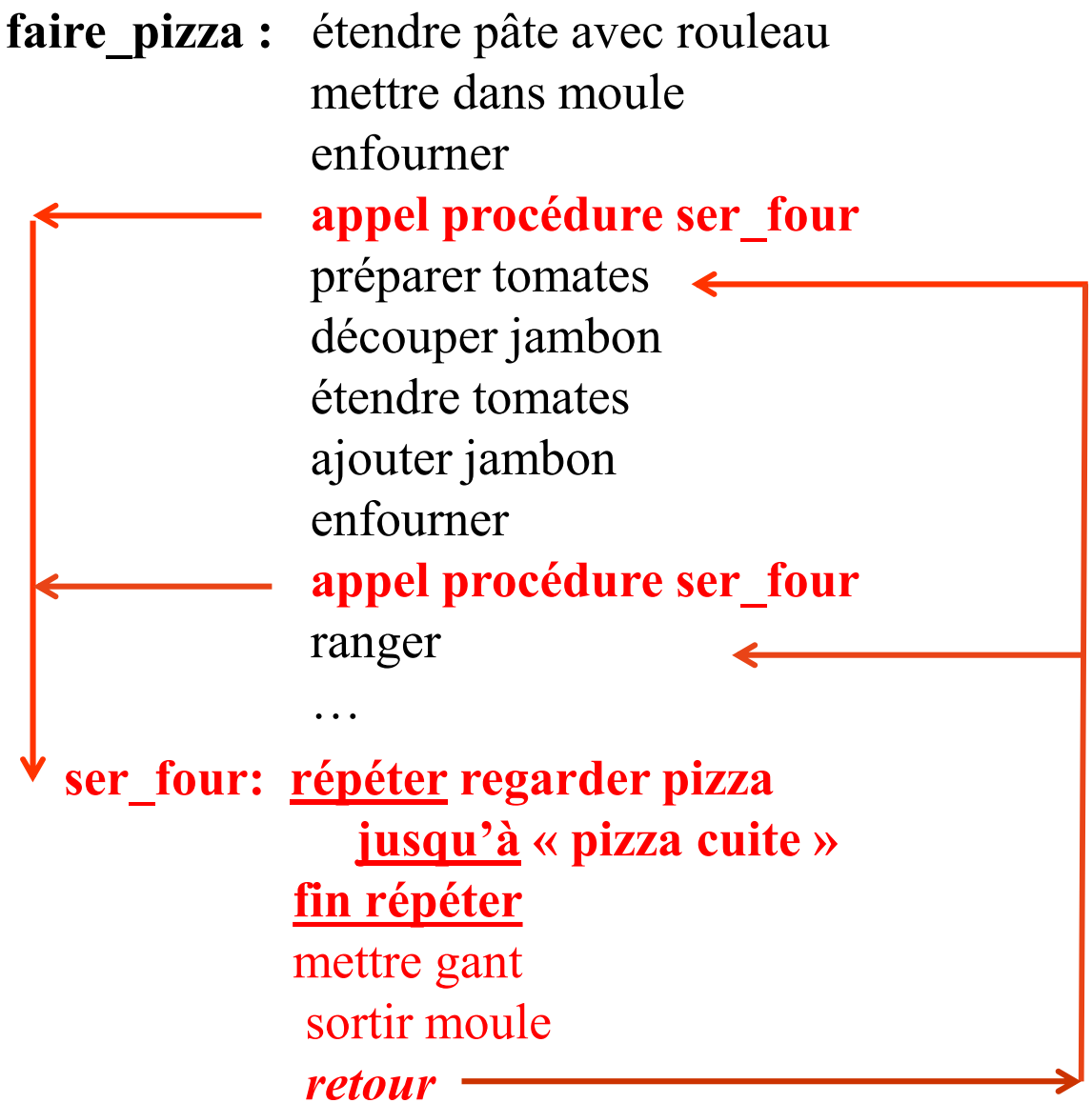
- pour une **entrée**, y a t'il une nouvelle donnée à lire?
- pour une **sortie**, peut-on écrire une nouvelle donnée?

➡ **lire** la valeur d'un bit particulier dit **drapeau** (*flag*) dans un **registre d'état** du circuit d'E/S.

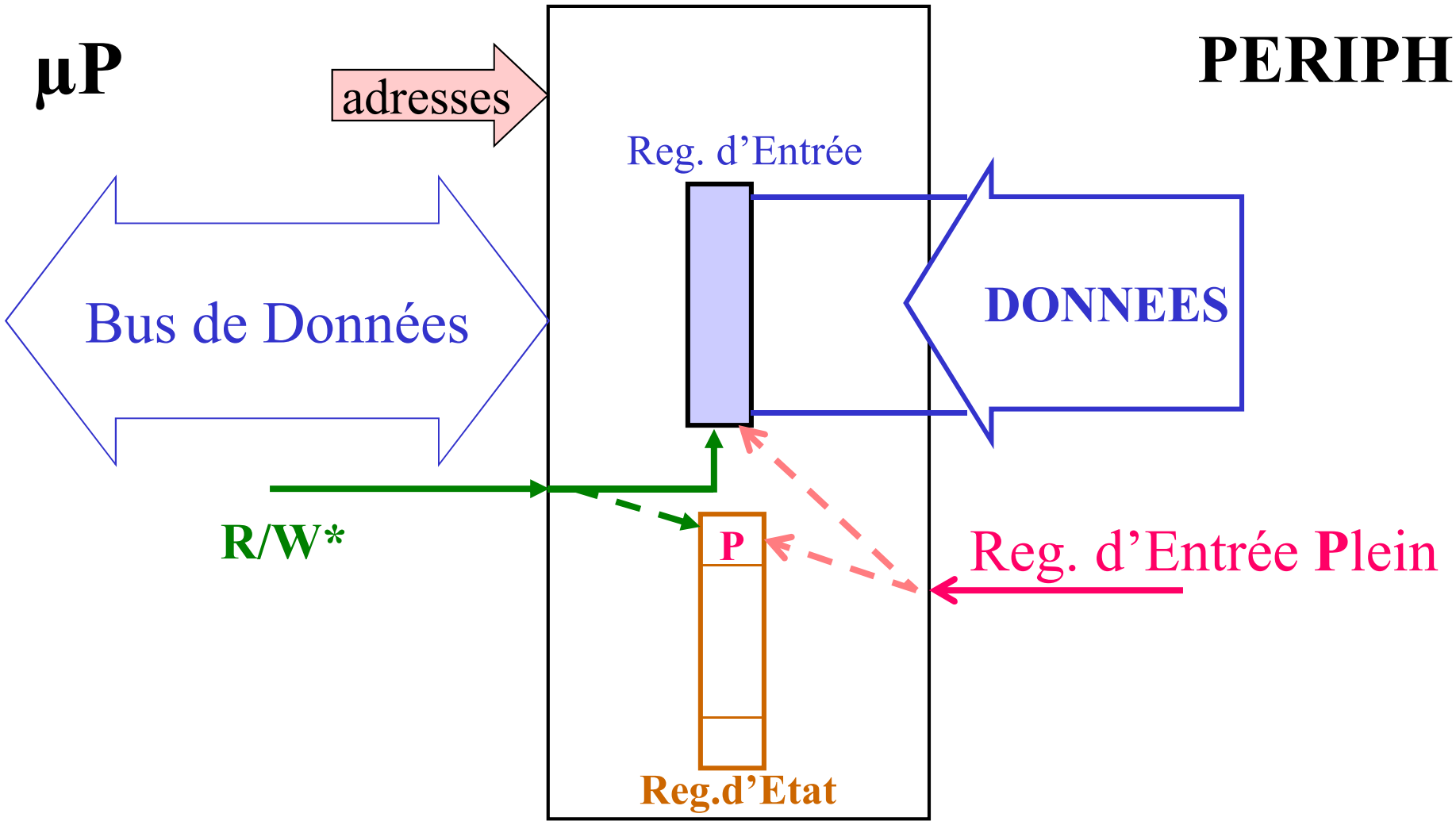
• Il existe 2 techniques d'**interrogation** :

- mode **bloquant** par boucle d'attente (**tant que …**)
- mode **non bloquant** par simple vérification (**si ... alors**).

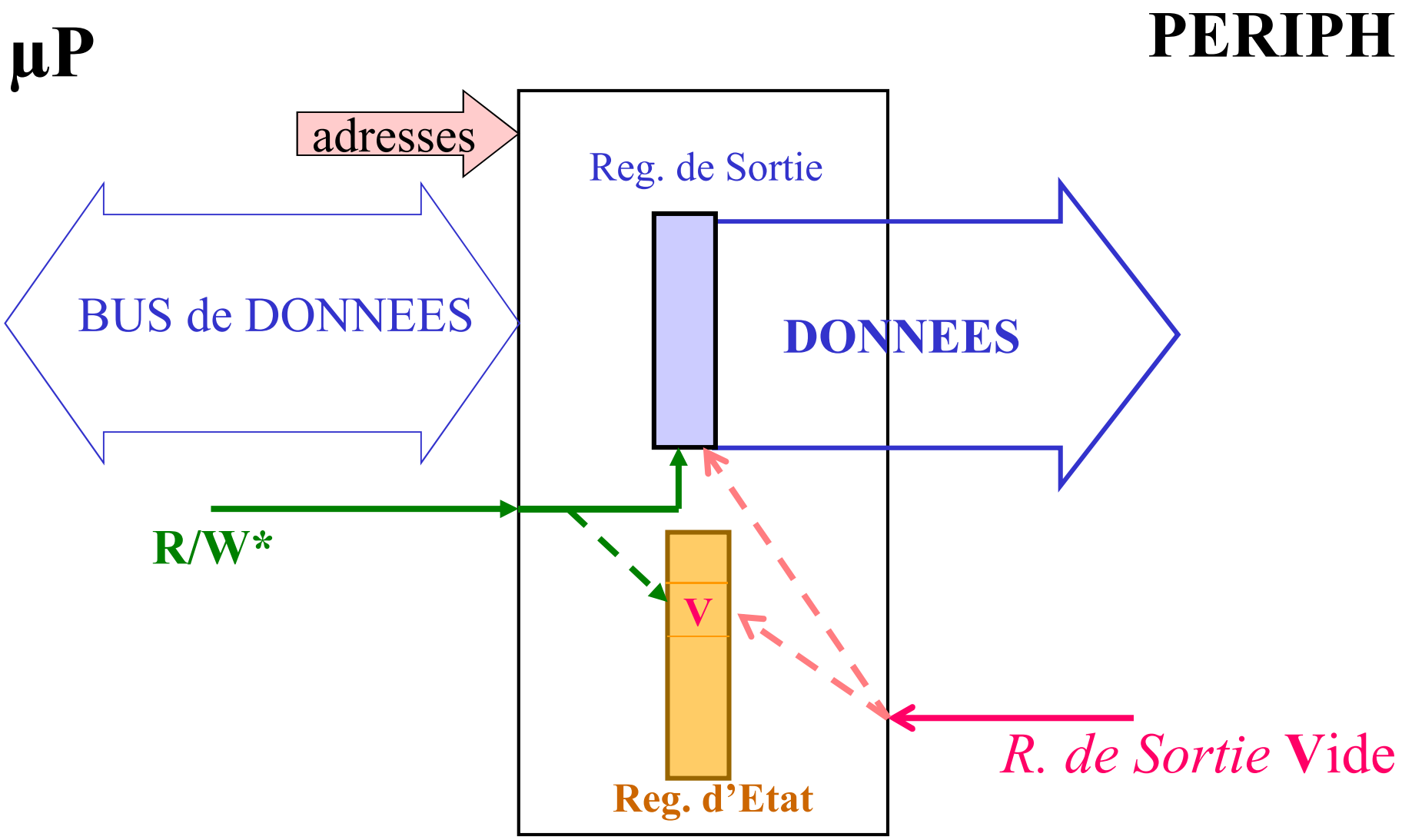
E/S par interrogation : illustration



Exemple : Synchronisation d'entrée



Exemple : Synchronisation de sortie



Annexe : Fonctionnement du µP avec IT

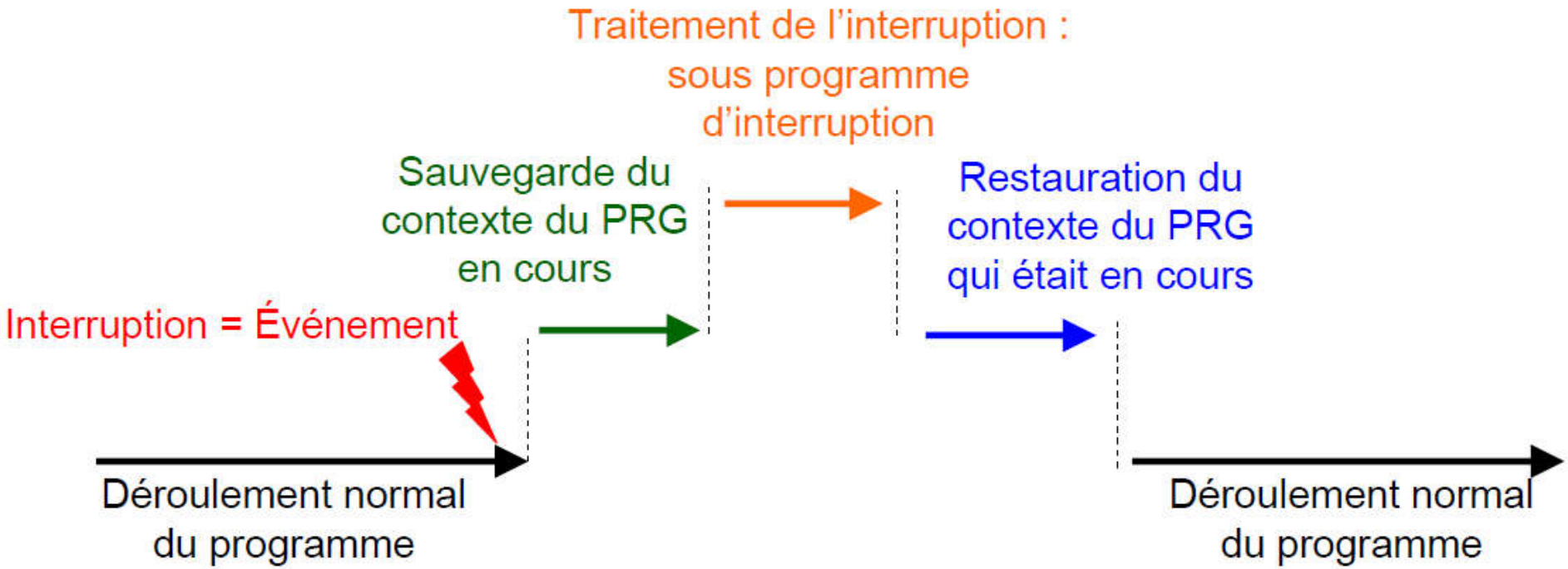
LE PROCESSEUR REPETE SANS ARRET :

- Lire l'instruction pointée par le PC
- Mettre à jour le PC
- Décoder l'instruction
- Exécuter l'instruction
- **Traiter demande d'IT en attente**

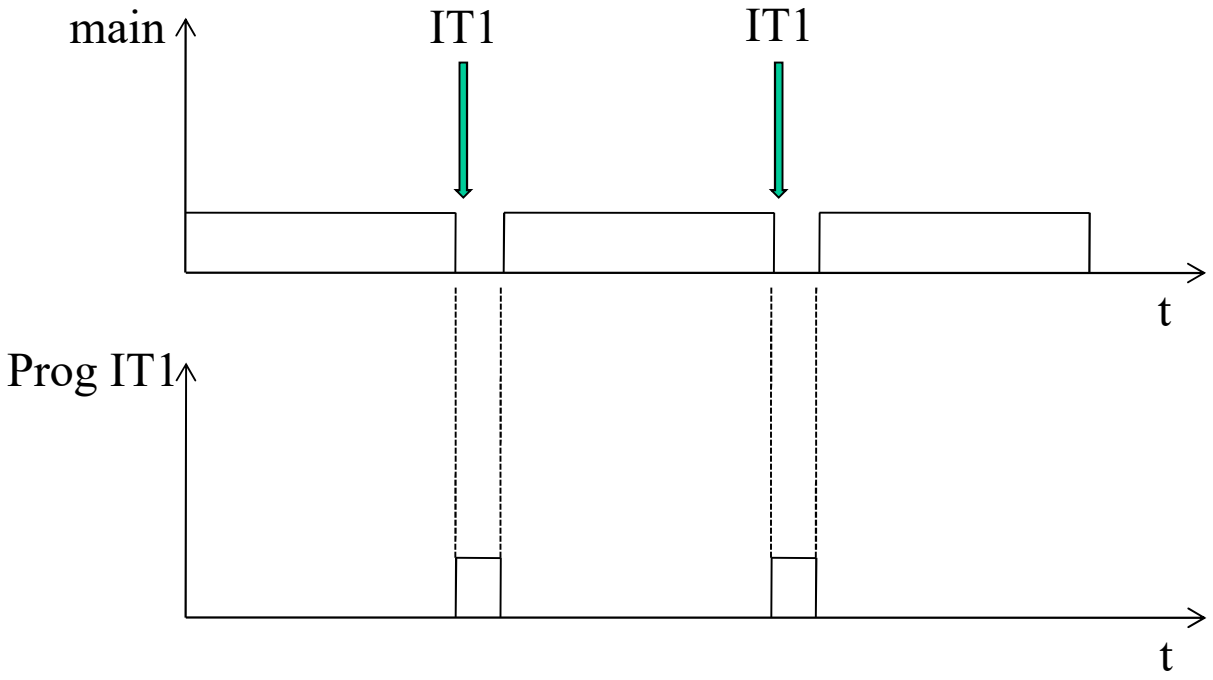
(saut à la procédure de service de l'IT)

Définition et déroulement d'une interruption

• Définition : une **interruption est un arrêt temporaire de l'exécution normale d'un** programme informatique par le microprocesseur afin d'exécuter un autre programme (appelé routine d'interruption).



Chronogramme d'un programme avec une interruption



- Dans la pile au début IT1 : sauvegarde PC + contexte
- PC ← routine associée à IT1
- Fin IT1 : restitution contexte, puis PC reprend son ancienne valeur

Types d'interruption

- Interruption masquable : un masque d'interruption est un mot binaire de configuration du microprocesseur qui permet de choisir **(démasquer) quels** modules pourront interrompre le processeur parmi les interruptions disponibles :
  - **interruption matérielle** : composant d'entrées/sorties. Il faut alors initialiser les registres de contrôle, les vecteurs d'interruptions et autoriser le niveau d'interruption pour le processeur.
  - **Interruption logicielle :** déclenchée par l'exécution d'une instruction particulière dans le programme

- Interruption non masquable : elles s'exécutent quoi qu'il arrive, souvent avec une priorité élevée (ex : Reset)

Configuration des interruptions

- Un système peut accepter plusieurs sources d'interruption. Chacune est configurable par un registre d'interruption

- Méthode de configuration des interruptions :
  - Sélectionner les interruptions qui nous intéressent
  - Valider les interruptions de façon globale
  - Ecrire le/les sous programme d'interruption
  - Définir les priorités entre interruptions

- Remarque : une procédure de service d'IT étant exécutée à des **moments inconnus à l'avance,** le **passage de paramètres est donc impossible**: *la communication avec les autres progr. se fait par variables globales exclusivement.*

Accès à la procédure d'IT

• Les adresses des procédures de service d'IT sont disponibles dans une table résidant en mémoire principale, *à un endroit fixé par le constructeur.*

⟹ Cette table est généralement appelée table des vecteurs d'IT.

• L'élément n°i ($1 \leq i \leq N$) de la table contient l'adresse de la procédure associée à la borne IRQ_i du µP : dans la zone mémoire ISR_IRQi

• Pour sauter à la procédure d'IT associée à la requête émise sur la borne IRQ_i, le µP fait :

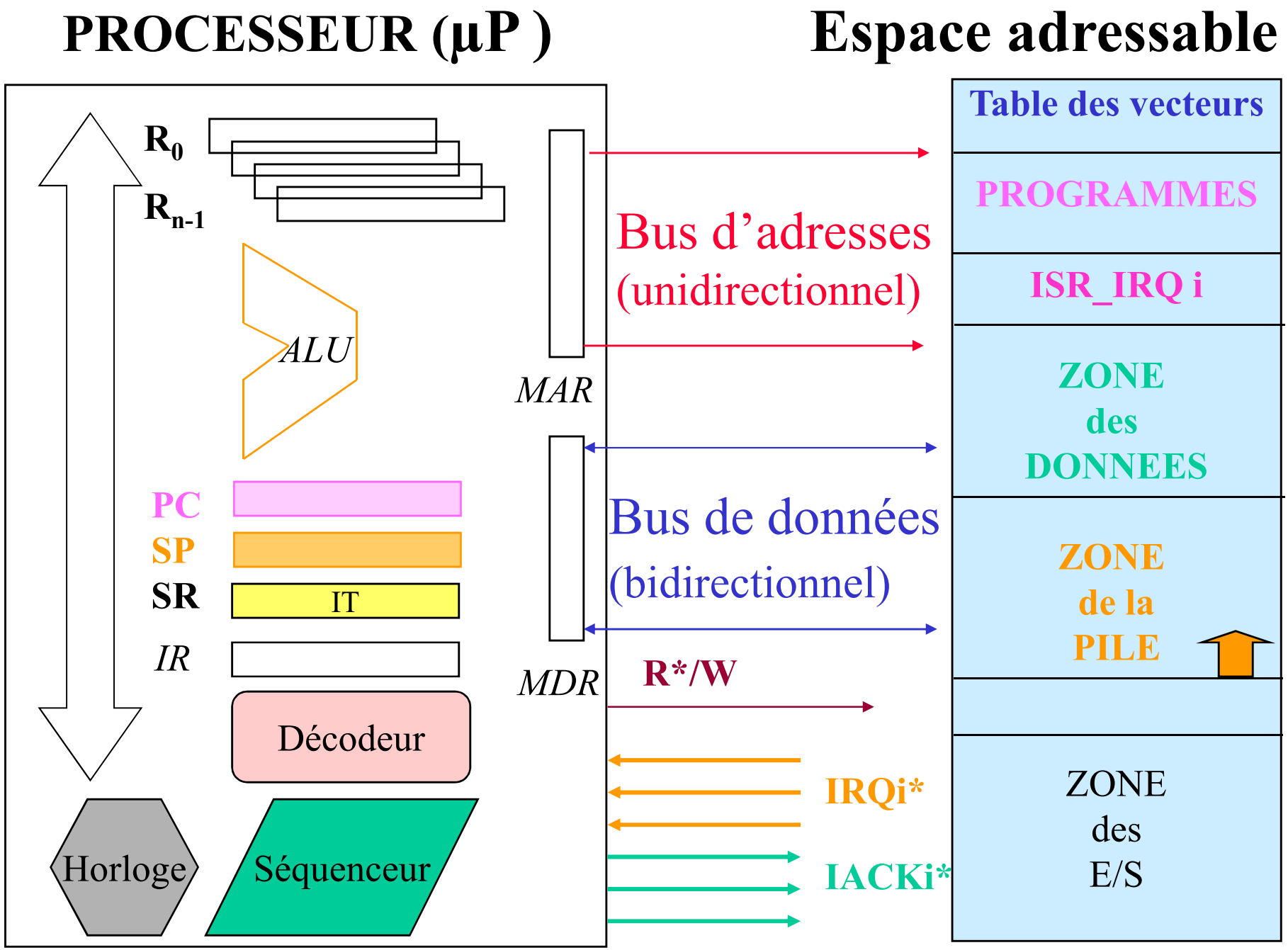
PC ← tab_vec[i]

Rôle de la pile dans le traitement d'une interruption

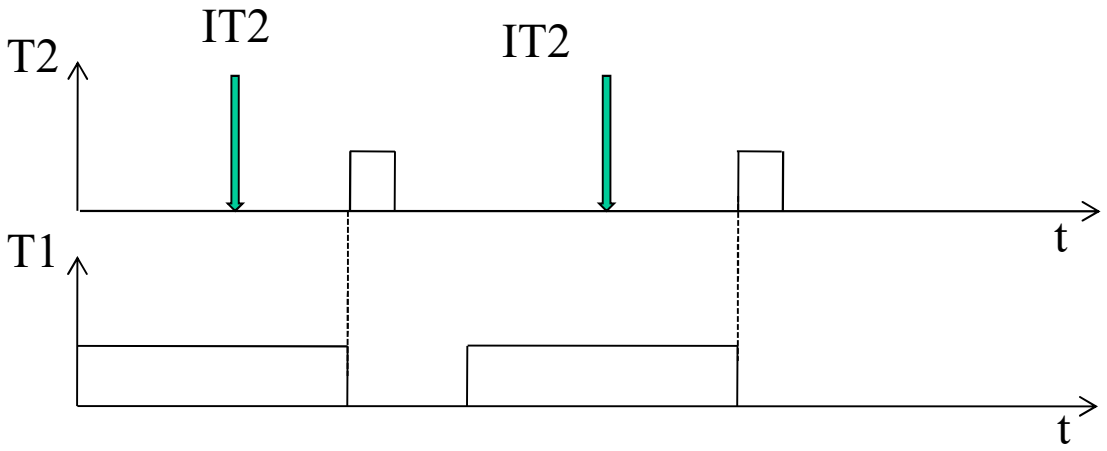
- Elle va servir à sauvegarder le contexte avant le départ en routine :
  - adresse du programme PC pour le retour
  - valeur des registres au moment de l'interruption

- Elle va servir à restituer le contexte à la fin de la routine :
  - valeur des registres au moment de l'interruption
  - adresse du programme PC pour le retour

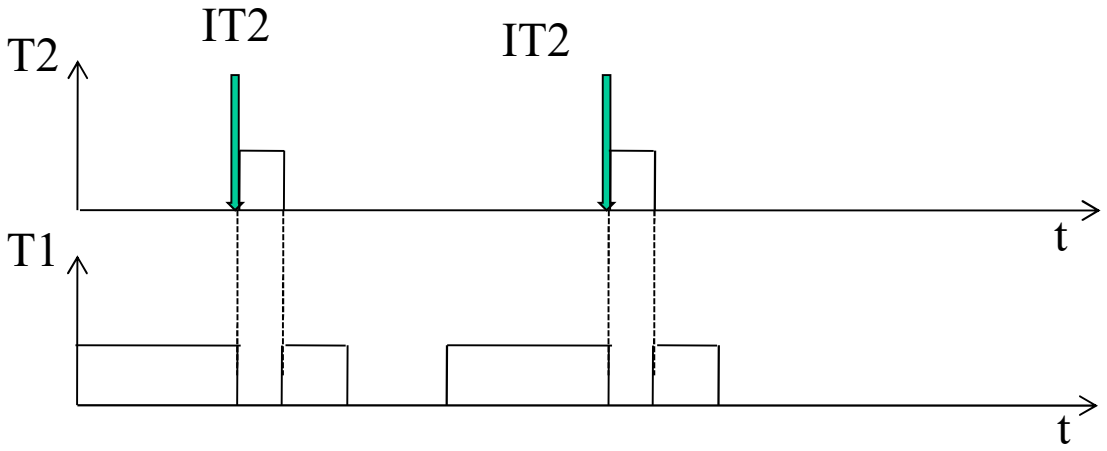
- Remarque :
  - sauvegarde et restitution faite implicitement en C mais à faire en assembleur
  - en C, mettre une déclaration spéciale pour permettre au compilateur de placer une instruction de type RTI à la place de RTS (dans le cas d'une fonction normale)


PROCESSEUR (µP)
Espace adressable
R0
Rn-1
ALU
PC
SP
SR
IR
IT
Décodeur
Horloge
Séquenceur
MAR
MDR
Bus d'adresses
(unidirectionnel)
Bus de données
(bidirectionnel)
R*/W
IRQi*
IACKi*
Table des vecteurs
PROGRAMMES
ISR_IRQ i
ZONE
des
DONNEES
ZONE
de la
PILE
ZONE
des
E/S

Rôle des priorités dans un programme avec IT



Priorité T1 > priorité T2
donc T1 n'est pas interrompu



Priorité T1 < priorité T2
donc T1 est interrompu

Gestion des E/S par Interruption

• **But** : éviter au **µP** les boucles d'attente.

• **Solution** : un circuit d'E/S peut demander au µP d'interrompre le progr. en cours pour traiter une E/S.

• **Mise en œuvre** : le µP possède

- plusieurs bornes d'entrée pour les requêtes d'IT,
- un moyen d'accepter ou de refuser d'interrompre le progr. en cours.

E/S par interruption : illustration

faire_pizza : étendre pâte avec rouleau
mettre dans moule
enfourner
cuit ← faux
armer minuterie four
préparer tomates
découper jambon
tant que cuit = faux
nettoyer cuisine
fin tant que
étendre sauce tomate
ajouter jambon
enfourner
cuit ← faux
armer minuterie four
tant que cuit = faux
ranger ustensiles
fin tant que …

IT_four (sonnerie)

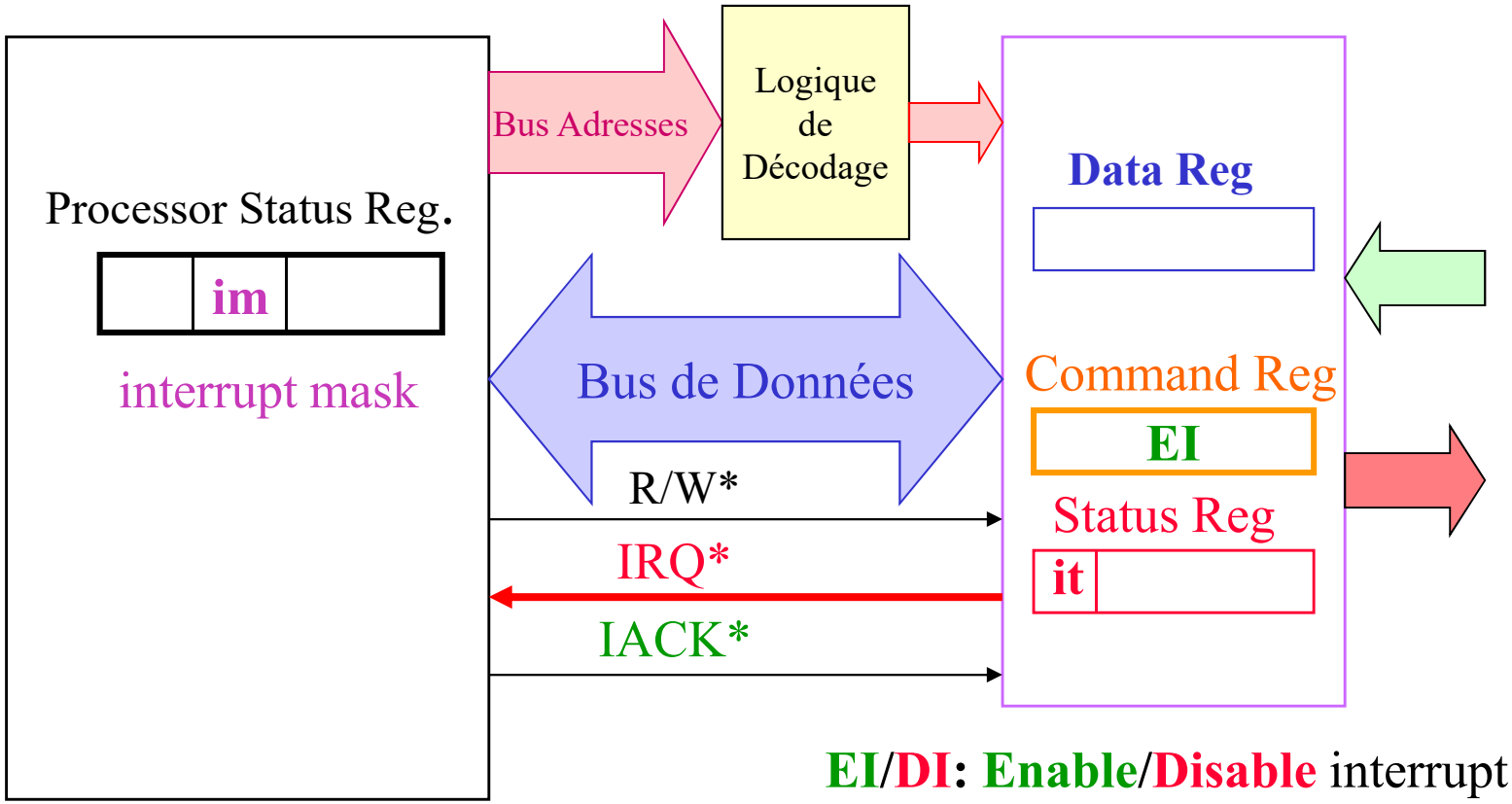
ser_four: cuit ← vrai
mettre gant
ouvrir four
sortir moule
ret_it

IT_four (sonnerie)

E/S par interruption : Mise en Œuvre des IT

Processeur
masquer/démasquer IT

Circuit d'E/S
armer/désarmer IT

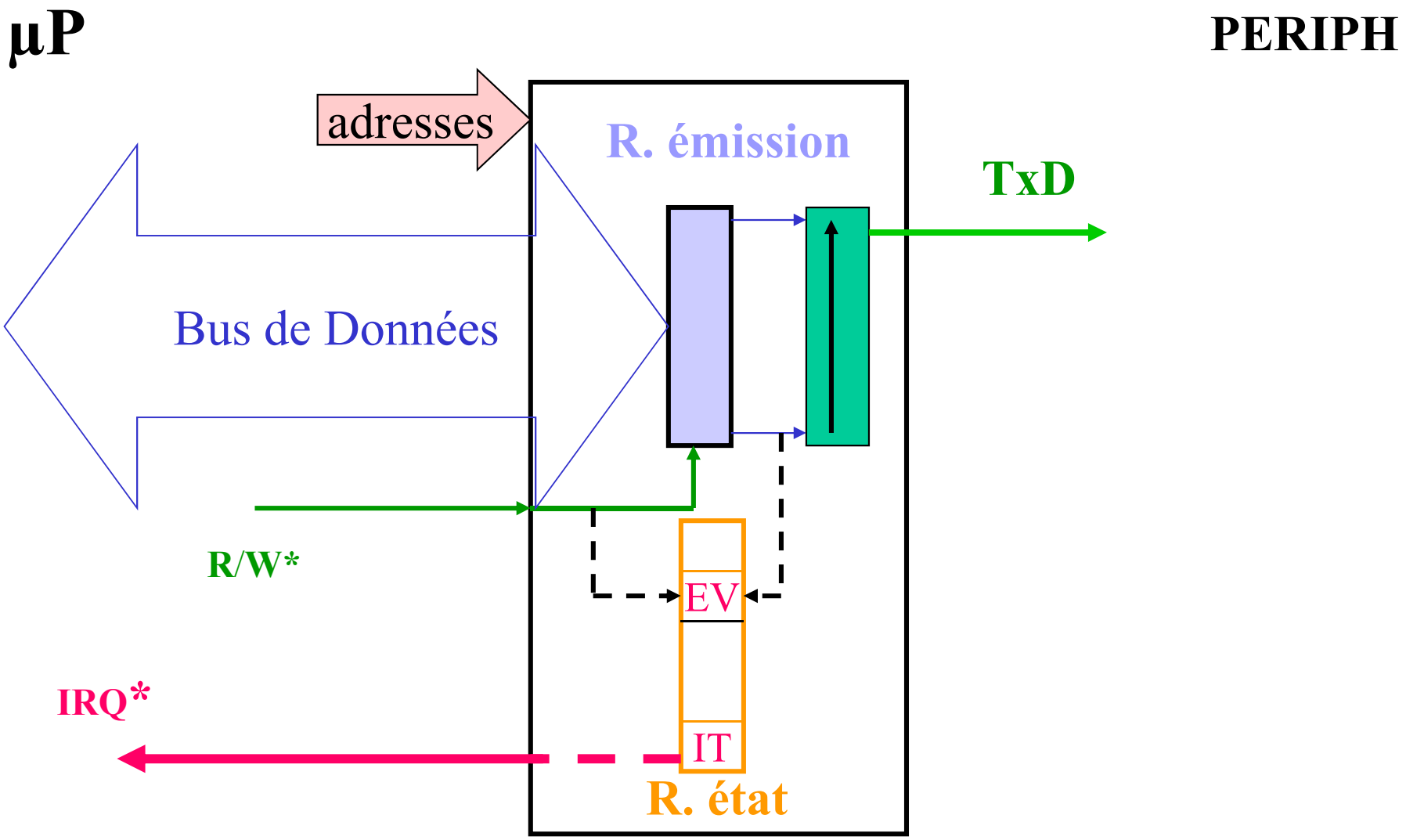


EI/DI: Enable/Disable interrupt

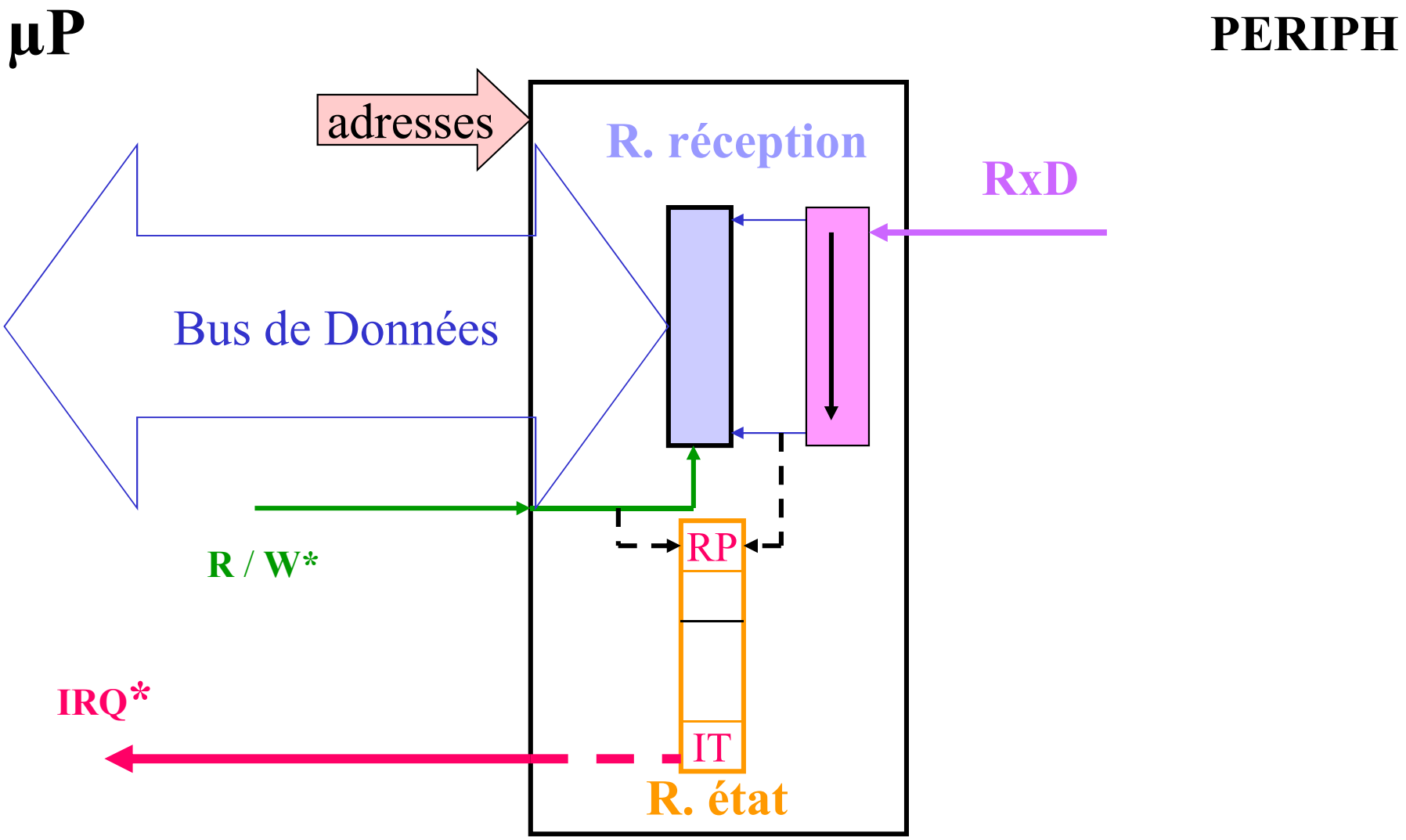
Interdire/ Autoriser au *processeur*
l'interruption du programme **en cours**

Armer/ Interdire au *circuit*
la création d'un signal d'IT

Exemple synchronisation entrée



Exemple : synchronisation de sortie



Liaison série

Principe liaison série asynchrone

- Transmission en **série (bit après bit) sur une seule ligne**

  mode simplex :

  mode semi-duplex (à l'alternat) :

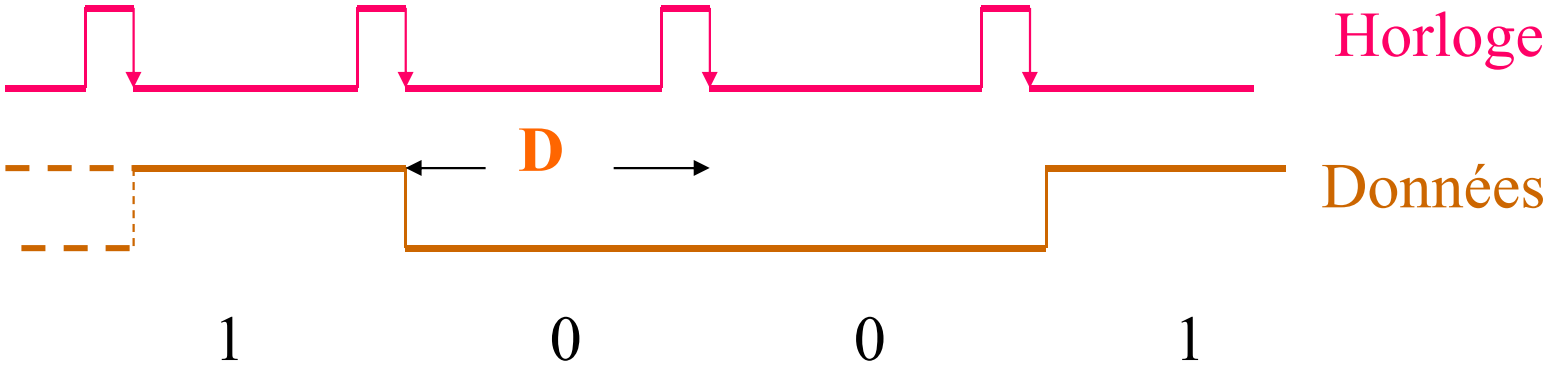
  mode duplex intégral :

- **Avantages** : simplicité, coût, pas de diaphonie, …
- **Inconvénient** : vitesse

Principe de l'émission

• La cadence (nb de bits /sec.) est fournie par une horloge.

• HORLOGE d'EMISSION : l'horloge fixe l'instant d'émission du bit suivant



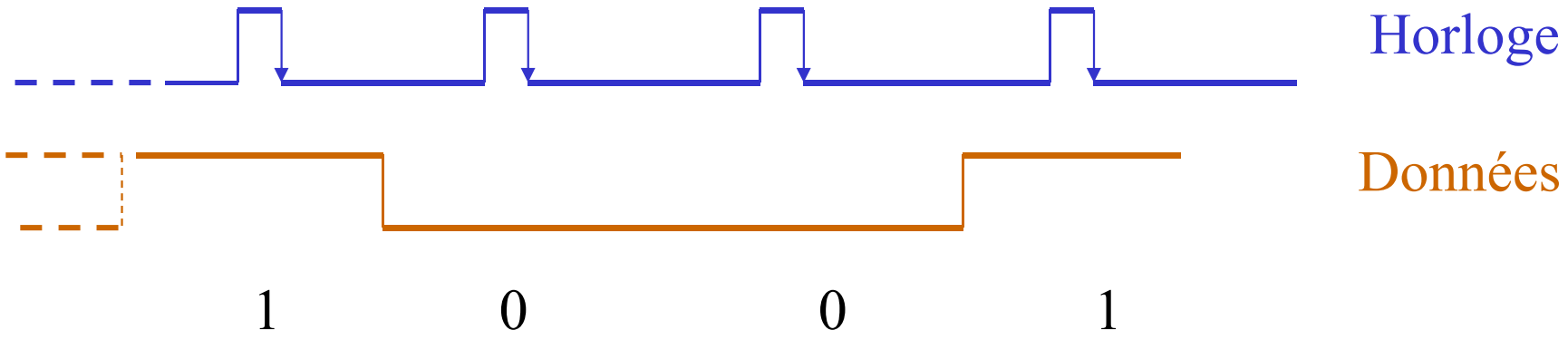
Durée d'un bit $\mathbf{D = T_{ck}}$ = période d'horloge

N.B. : en général, $\mathbf{D = N\ T_{ck}}$ avec $\mathbf{N > 1}$ (par ex.**16**)

Principe de la réception

• La cadence (nb de bits /sec.) est fournie par une horloge.

• HORLOGE de RECEPTION : l'horloge de réception fixe l'instant où il faut lire le bit (au milieu pour éviter erreurs)



REMARQUES :

- les horloges de réception et d'émission sont **déphasées**
- l'horloge de réception est créée par le récepteur lui même
- on choisit **D = N T_{ck}** avec **N > 1** (horloge peu stable)

Mise en Œuvre d'une Liaison Série

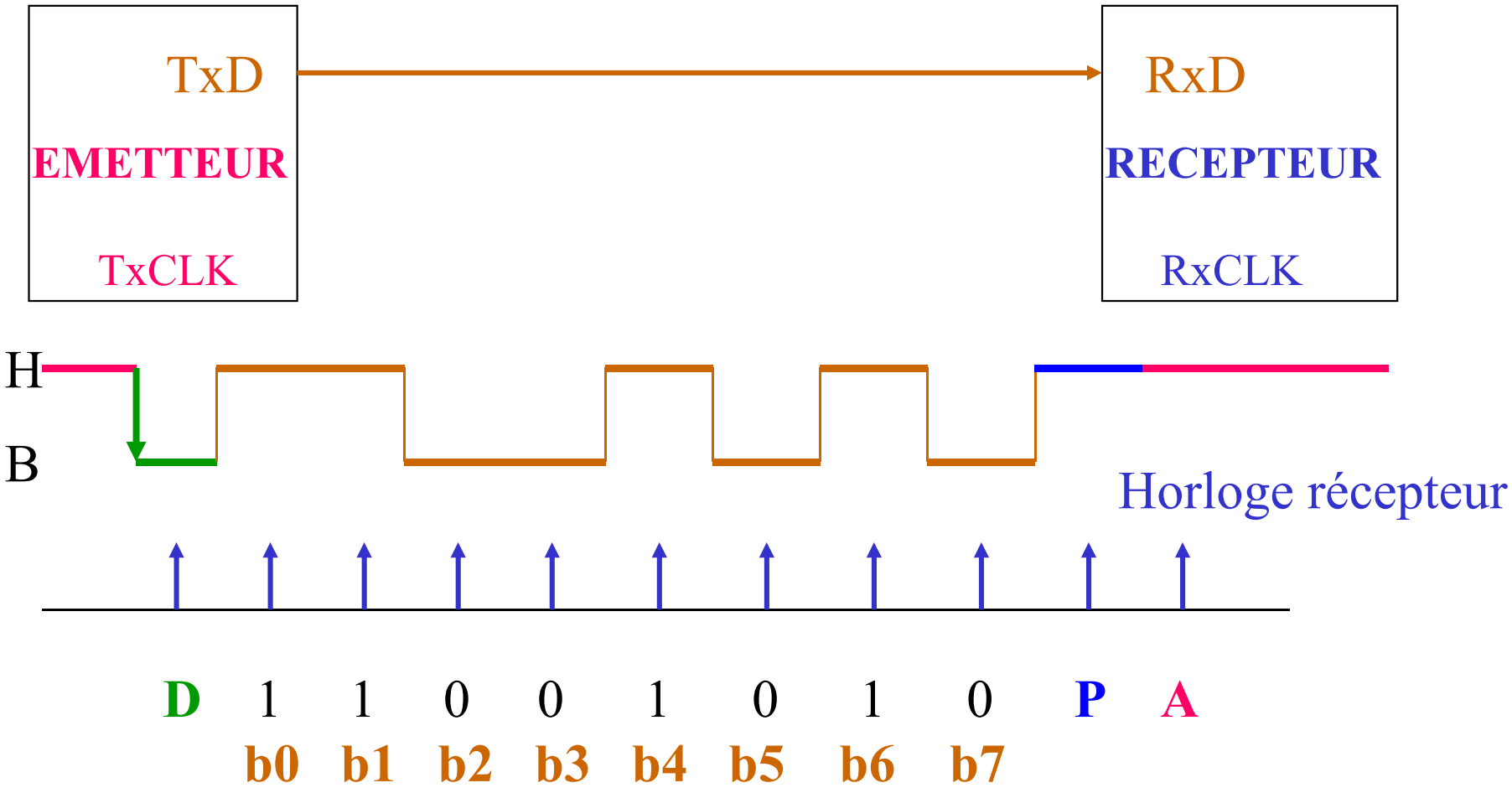
- Les données émises sont englobées dans une TRAME

- En absence de signal, la ligne est à l'état HAUT
  - Un bit DEPART à l'état BAS signale qu'une trame débute.
  - La donnée est la série de bits dans l'ordre b0 puis b1 puis b2, …
  - Un bit de PARITE peut être transmis.
  - Un bit d'ARRET à l'état HAUT indique la fin de la trame.

- L'intervalle de temps entre 2 trames peut être quelconque.

- L'horloge de réception est déclenchée par le bit DEPART (START)

Illustration d'une trame



N.B. après le bit de départ, il y a le bit 0 (pas le bit7)

Question : quelle est la parité choisie dans cet exemple ?

Rôle du bit de parité

- PRINCIPE du CONTRÔLE de PARITE

- L'émetteur ajoute aux bits de données un bit de parité :
  - tel que le nombre total de 1 ait une parité donnée
  - soit IMPAIR (« odd ») soit PAIR (« even »).

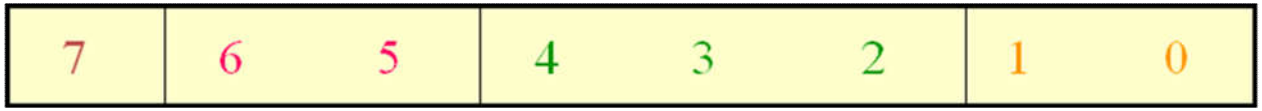
- Le récepteur compte le nombre de 1 reçu avant le bit STOP.

- Si le nombre a la parité attendue, la trame est réputée bonne sinon elle est erronée.

hypothèse de l'erreur unique

Exercice : UART par scrutation / interruption

UART : **Registre de commande / contrôle**



- bit 7 = 1 : Armer IT Réception
- bit 6 = 0 : activer RTS
- bit 5 = 1 : Armer IT Emission
- bit 4 = 0/1 : caractère 7/8 bits
- bit 3 = 0/1 : 2 STOP / 1 STOP
- bit 2 = 0/1 : PAIR/IMPAIR
- bit 1 et 0 = 00 : Horloge : 1
  = 01 : Horloge : 16
  = 10 : Horloge : 64
  = 11 : RESET

UART : **Registre d'état de l'UART**

IRQ	ERREURS : P. C. T.	CTS	DCD	E.Vide	R.Plein

- bit 7 = 1 : IT émise
- bit 6 = 1 : erreur de parité
- bit 5 = 1 : erreur de cadence (Overrun)
- bit 4 = 1 : erreur de Trame (Frame)
- bit 3 = 1 : copie de la borne CTS
- bit 2 = 1 : copie de la borne DCD
- bit 1 = 1 : Reg. Emission vide
- bit 0 = 1 : Reg. Réception plein

Questions

Le circuit UART est représenté en informatique par la variable globale ***uart*** de type enregistrement

cntrl de type octet non signé
status de type octet non signé
data_in de type octet non signé
data_out de type octet non signé
fin enregistrement

1. Fonctions d'entrée par interrogation. Pour chaque fonction/sous programme donnez le prototype.
   a) Ecrire en langage C une fonction **bloquante** (boucle d'attente) baptisée **lire_b()** qui retourne la valeur du registre d'entrée.

   b) Ecrire en langage C une fonction **non bloquante** baptisée **lire_nb()** qui copie dans une variable *result* (*paramètre de sortie*) une nouvelle donnée si elle est disponible et retourne 0 (-1 dans l'autre cas).

Questions

2. Fonctions de sortie par interrogation. Pour chaque fonction/sous programme donnez le prototype.

   a) Ecrire en langage C une procédure **bloquante** (boucle d'attente) baptisée **ecrire_b()** qui écrit la valeur donnée en paramètre dans le registre de sortie, dès que possible .

   b) Ecrire en langage C une fonction **non bloquante** baptisée **ecrire_nb()** qui écrit la valeur donnée en paramètre dans le reg. de sortie si c'est possible et retourne 0 (sinon -1).

Questions

3. Procédure de service d'IT de l'UART. On supposera que le périphérique désarme les ITs en rentrant dans la routine mais ne les réarme automatiquement en sortant de la routine.

a) On veut échanger des données de type char *car_in* et *car_out*. Comment déclarer ces variables ?

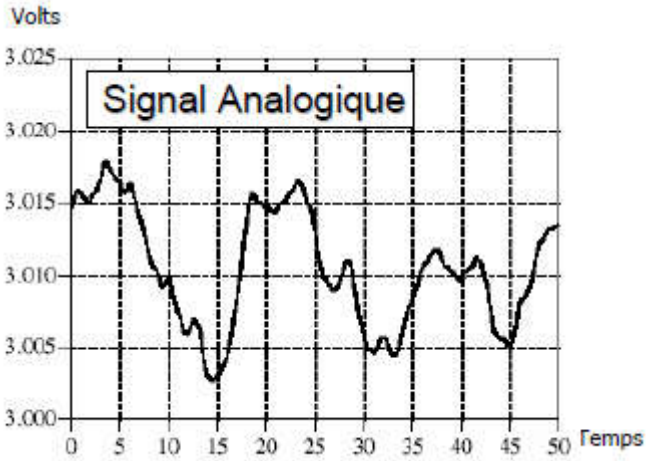
b) Ecrire en langage C une procédure de service d'IT baptisée **isr_in()** qui récupère le caractère reçu par l'UART et le range à *car_in.*

c) Soit *mesure[128]* de type tableau d'entier court et *nb_mes* de type entier court déclarés en variables globales. Ecrire en langage C la procédure de service d'IT **isr_input()** qui récupère le caractère reçu par l'UART et le range à sa place dans le tableau *mesure[]*.

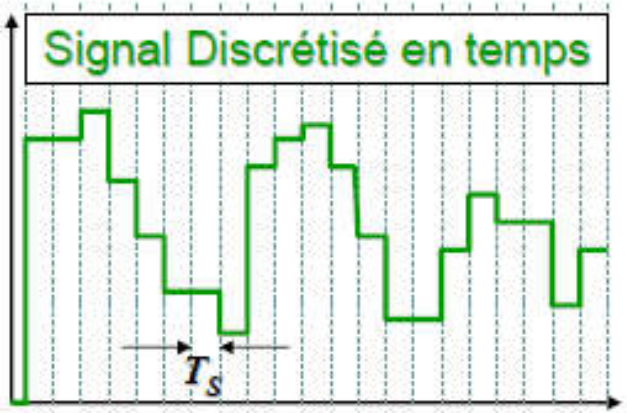
```
main: ...
        nb_mes ← 0
        armer les IT de réception
Tantque nb_mes < 128 Faire
        // instructions d'une tache (interruptible) à faire
FinTantque
désarmer les IT de réception
```

CAN / CNA

De l'analogique au numérique



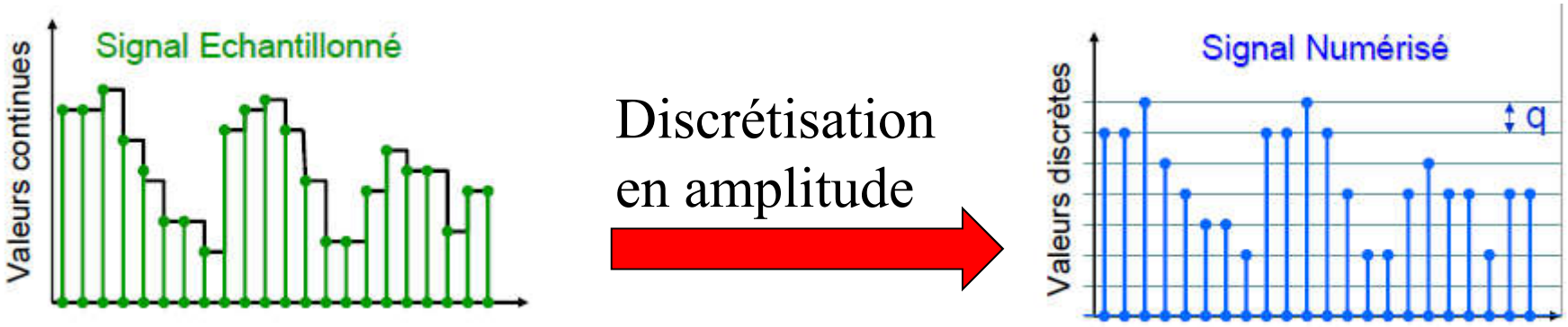
Discrétisation en temps



- Valeurs continues de tension en fonction du temps
- Discrétisation en temps : échantillonneur bloqueur

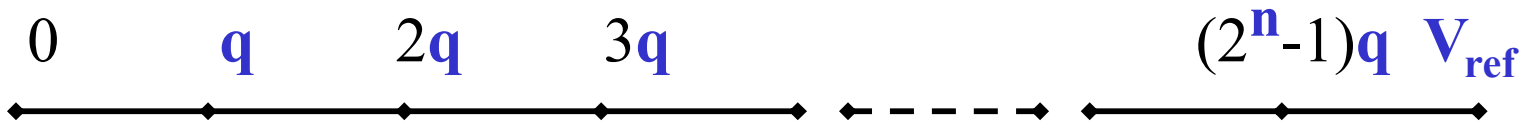
→ Fréquence : fs = 1/Ts

De l'analogique au numérique

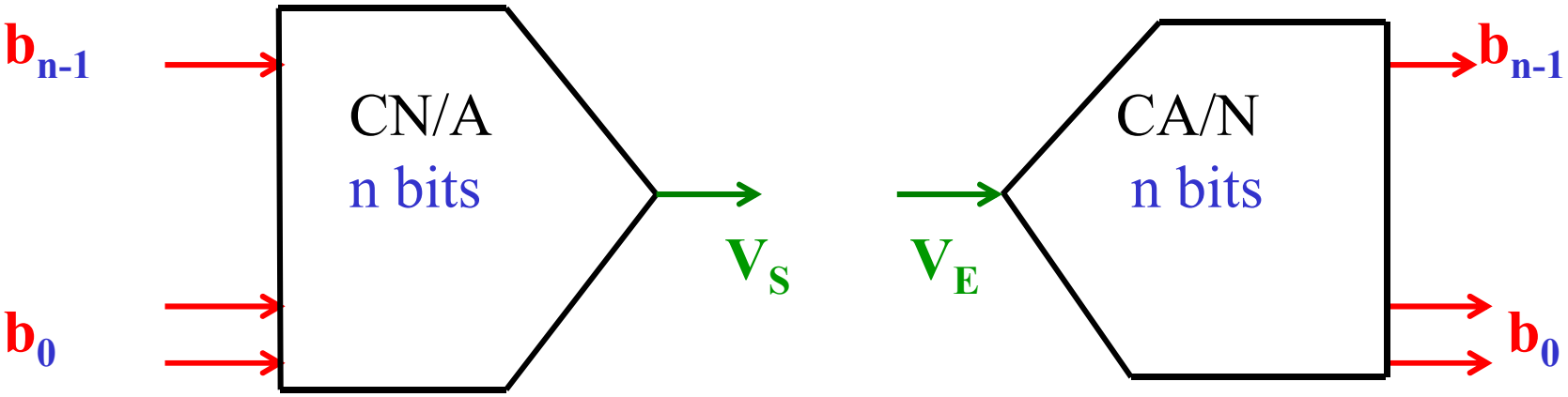


- Restriction des valeurs de magnitude possible
- Discrétisation en amplitude :
  → Quantification : q
- Représentable par un mot binaire de N bits

Principe des CN/A et CA/N



V_{ref} est aussi appelée V_{FS} (Full Scale) : $q = V_{ref}/2^n$



N_E (n bits)
Nombre entier

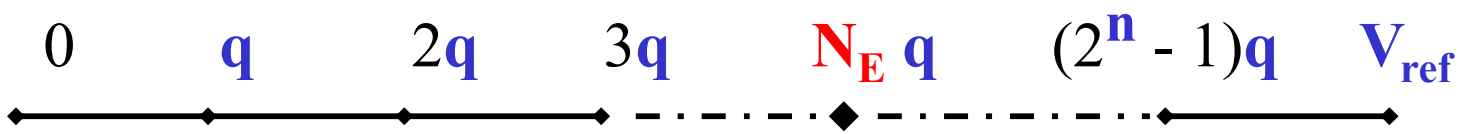
V_S V_E
tensions

N_S (n bits)
Nombre entier

$V_S = N_E\, q$

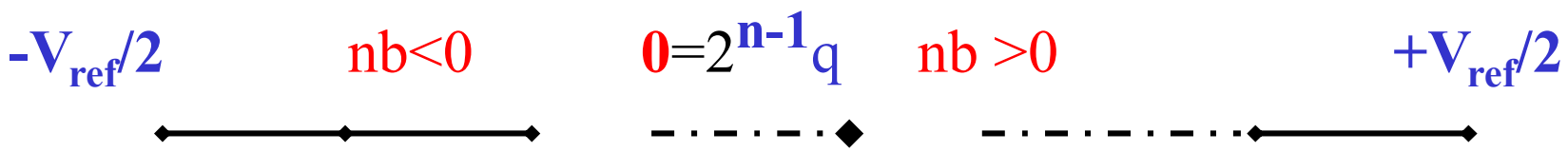
$N_S = V_E/q$

CN/A et CA/N unipolaire et bipolaire



Une tension (interne ou externe) V_{ref} est divisée en 2^n parties **q**.

Analogique $0 \leq V_s < V_{ref}$



La tension V_{ref} divisée en 2^n parties est décalée de $V_{ref}/2$.

Analogique $-V_{ref}/2 \leq V_s < +V_{ref}/2$

Quantification uniforme

• Définition : on a une quantification uniforme si les intervalles sont de même longueur q

• Erreur de quantification : écart entre valeur réelle et la valeur quantifiée :

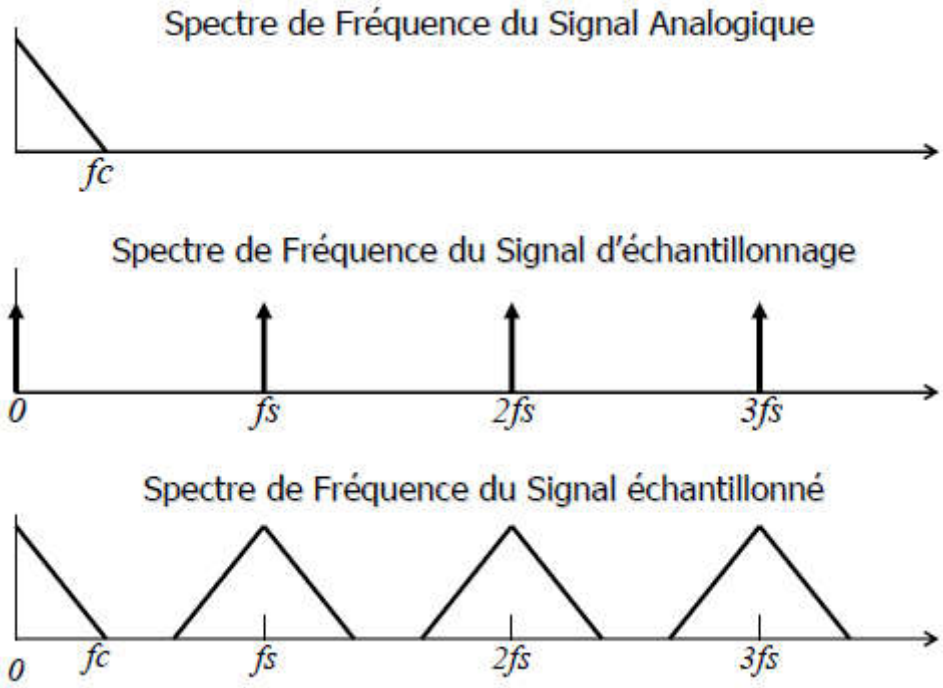
$$|e| \leq q/2$$

• Erreur de Saturation : elle se produit quand l'amplitude de l'échantillon dépasse |Vref| (ou |Vref/2|)

• Existe d'autres méthodes de quantification : logarithmique

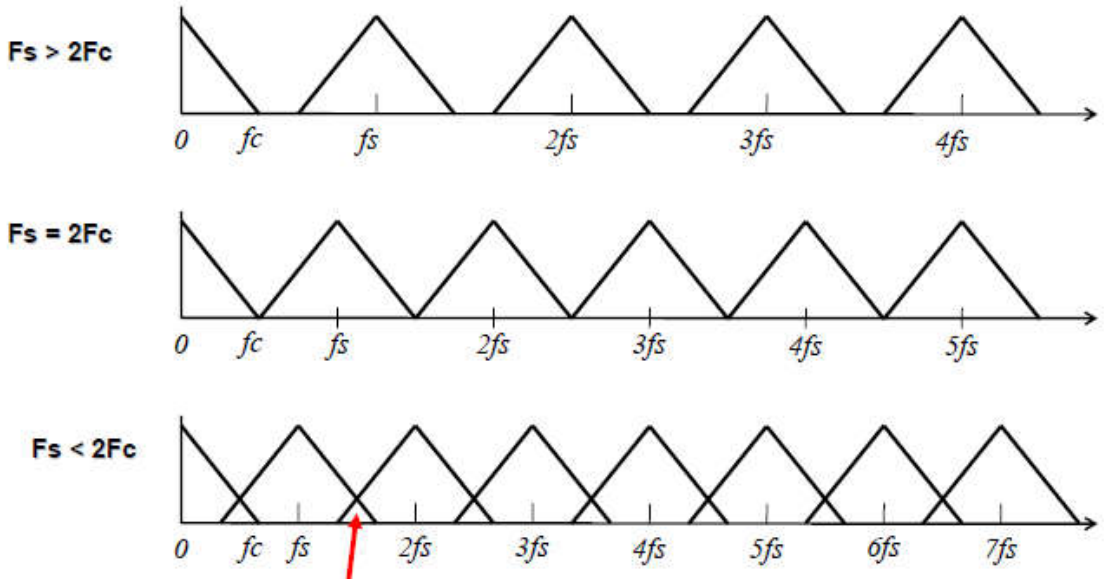
Caractéristiques du signal échantillonné

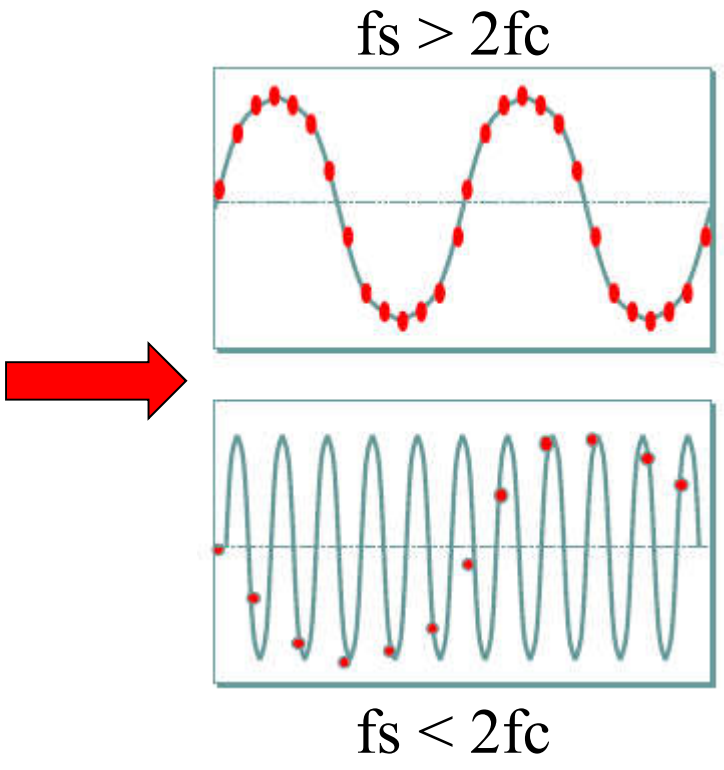
THEOREME de SHANNON : $\mathbf{F_s > 2\, f_{cmax}}$



➡ Échantillonnage en temps ⇔ périodisation en fréquence

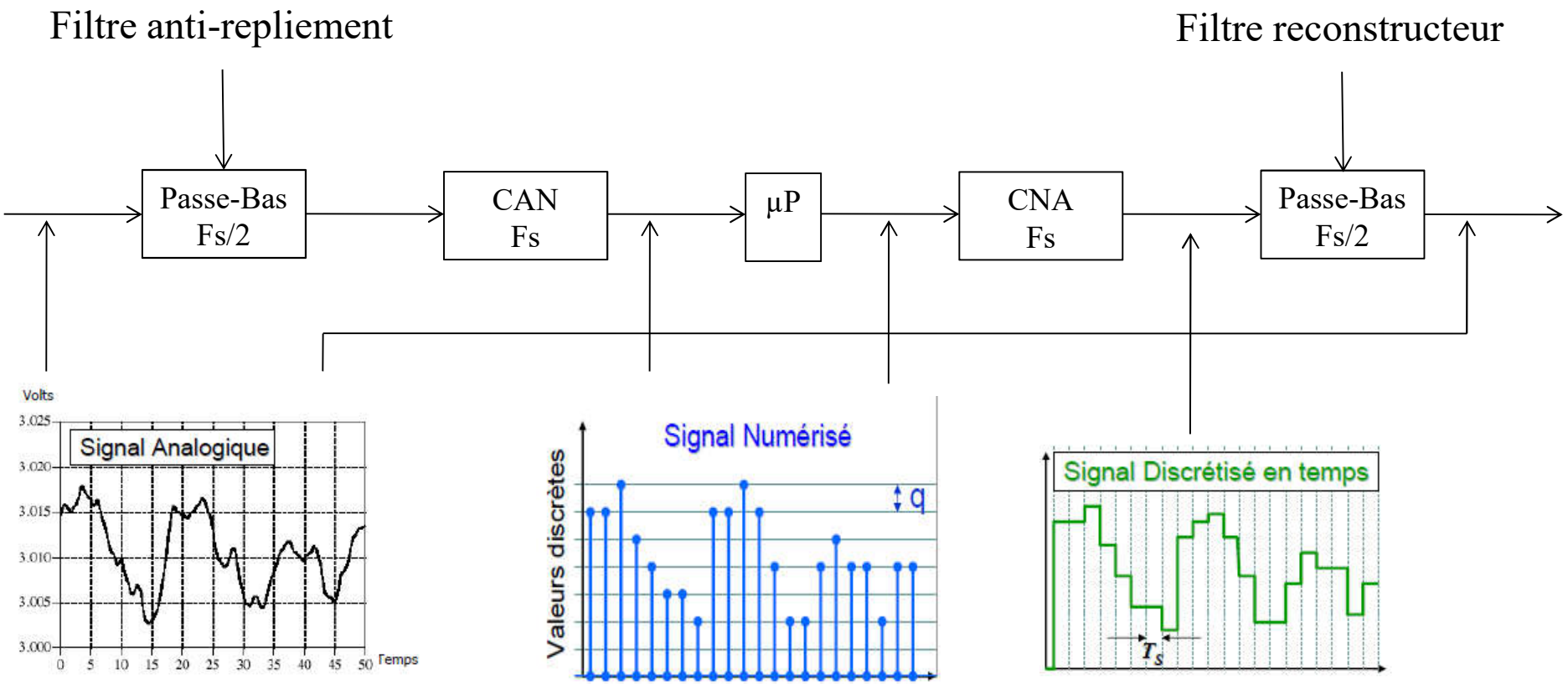
Effet du repliement de spectre





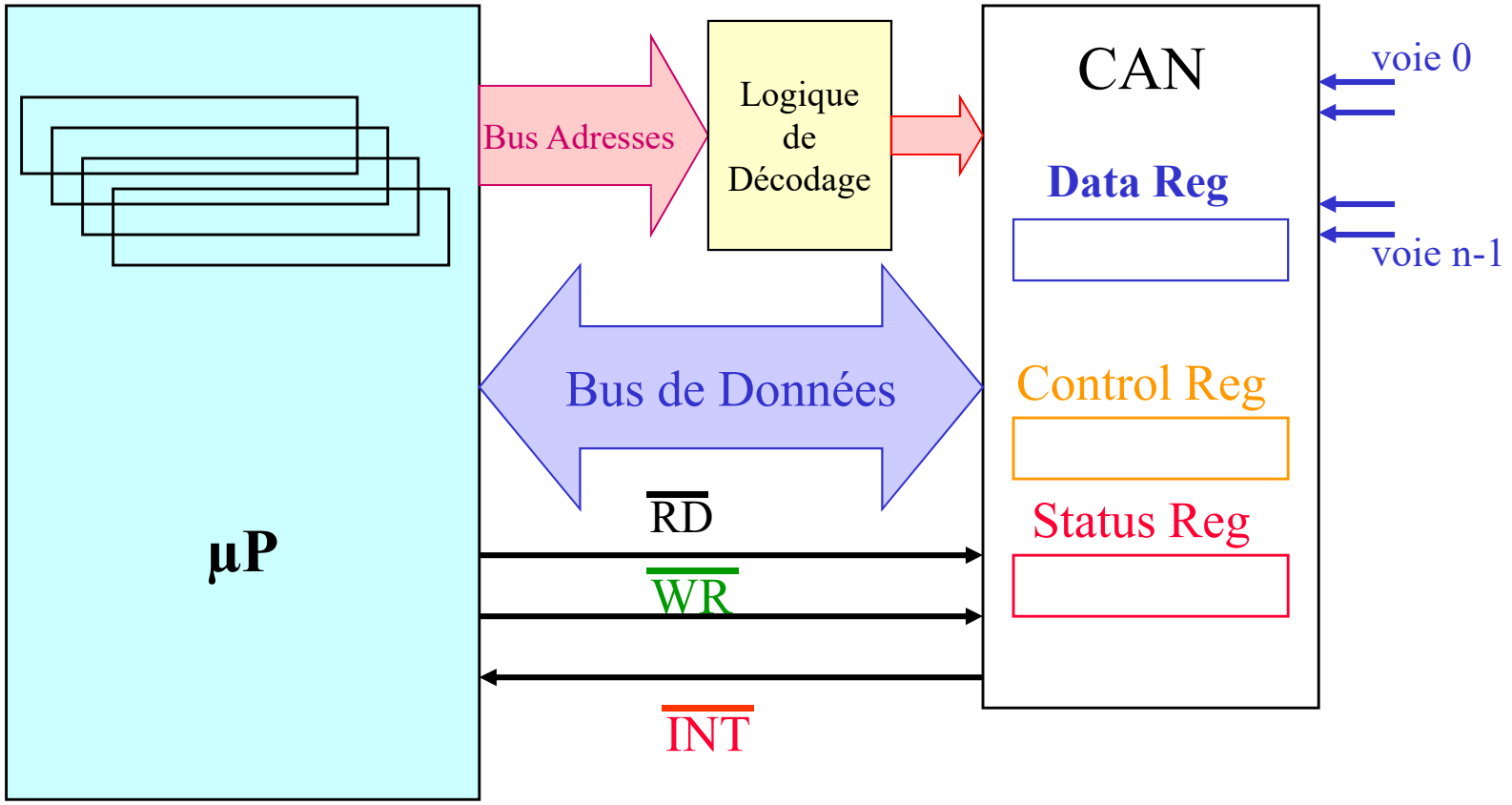
Repliement de spectre : impossible de retrouver le signal d'origine

Filtres Anti-Repliement et Reconstructeur



Mise en Œuvre d'un CAN

ATTENTION : la mémoire principale n'est pas représentée



Problème : taille du bus de données / nombre de bits du CAN

Registre de Commande d'un CA/N

• Les différents champs du reg. de commande permettent de choisir (un mode de fonctionnement):

- le mode unipolaire/bipolaire ;
- la tension de référence Vref (interne/externe, valeur) ;
- la fréquence d'horloge (interne/externe, valeur) ;
- le numéro de la voie à convertir ;
- l'instant de démarrage (unique, périodique) ;
- IT armée/désarmée ;
- …

Exercice : CAN par scrutation

Description des registres d'un CA/N 10 bits : **ADMUX (Multiplexer Selection Register)**

Bit	7	6	5	4	3	2	1	0
Name	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0

REFS1	REFS0	Vref
0	0	Aref (tension externe fournie sur la broche Aref du microcontrôleur)
0	1	Vcc (la tension d'alimentation fournie au microcontrôleur) soit 5V
1	0	Réservé
1	1	1.1V

ADLAR : 1 écriture à gauche sinon à droite

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8
1001	Réservé
1010	Réservé
1011	Réservé
1100	Réservé
1101	Réservé
1110	1.1V
1111	GND (0V)

Description des registres d'un CA/N 10 bits :
Registre d'état et de contrôle : ADCSRA

Bit	7	6	5	4	3	2	1	0
Name	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

- ADEN : allume le CAN
- ADSC : lance la conversion en mettant un 1 (fin de conversion qd ADSC retourne à 0)
- ADATE : conversion sur trigger si 1
- ADIF : Flag de fin de conversion qd on est en mode IT
- ADIE : interruption validée si 1
- ADPS : permet de diviser l'horloge (souvent obligatoire)

ADPS2	ADPS1	ADPS0	Facteur de division
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Description des registres d'un CA/N 10 bits : **Registres de données : ADCH et ADCL**

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	**ADC9**	**ADC8**	**ADCH**
	ADC7	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADC1**	**ADC0**	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 0

Lire d'abord ADCL puis ADCH

Questions

Du point de vue informatique, le CA/N est une variable globale :

can de type enregistrement
champ *admux* de type entier
champ *adcsra* de type entier
champs *adcl et adch* de type entier signé.
fin enregistrement

a) On utilise le CAN en mode : acquisition interne avec Vref = 5v. Donnez au mieux les valeurs des registres *can.admux* et *can.adcsra*. Vous pouvez aussi proposer un squelette de programme principal.

b) Ecrire la fonction bloquante **convertir** () qui retourne le nombre entier **NS** résultat de la conversion de la tension **VE** appliquée sur la voie dont le numéro est passé en paramètre. Avant tout, donnez le prototype.

c) Modifier la fonction pour qu'elle retourne la valeur de la tension d'entrée exprimée en millivolts.

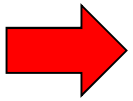
TIMER

Circuit « TIMER »

• Un **compteur** (de **n bits**) est mis à jour à la cadence d'une **horloge** interne ou externe, pouvant être **pré-divisée**.

• Quand une de ces conditions est respectée :

- le compteur repasse à 0 (après 2^n-1),
- ou atteint une valeur de référence,
- un front ↑ ou ↓ est détecté sur un signal d'entrée,

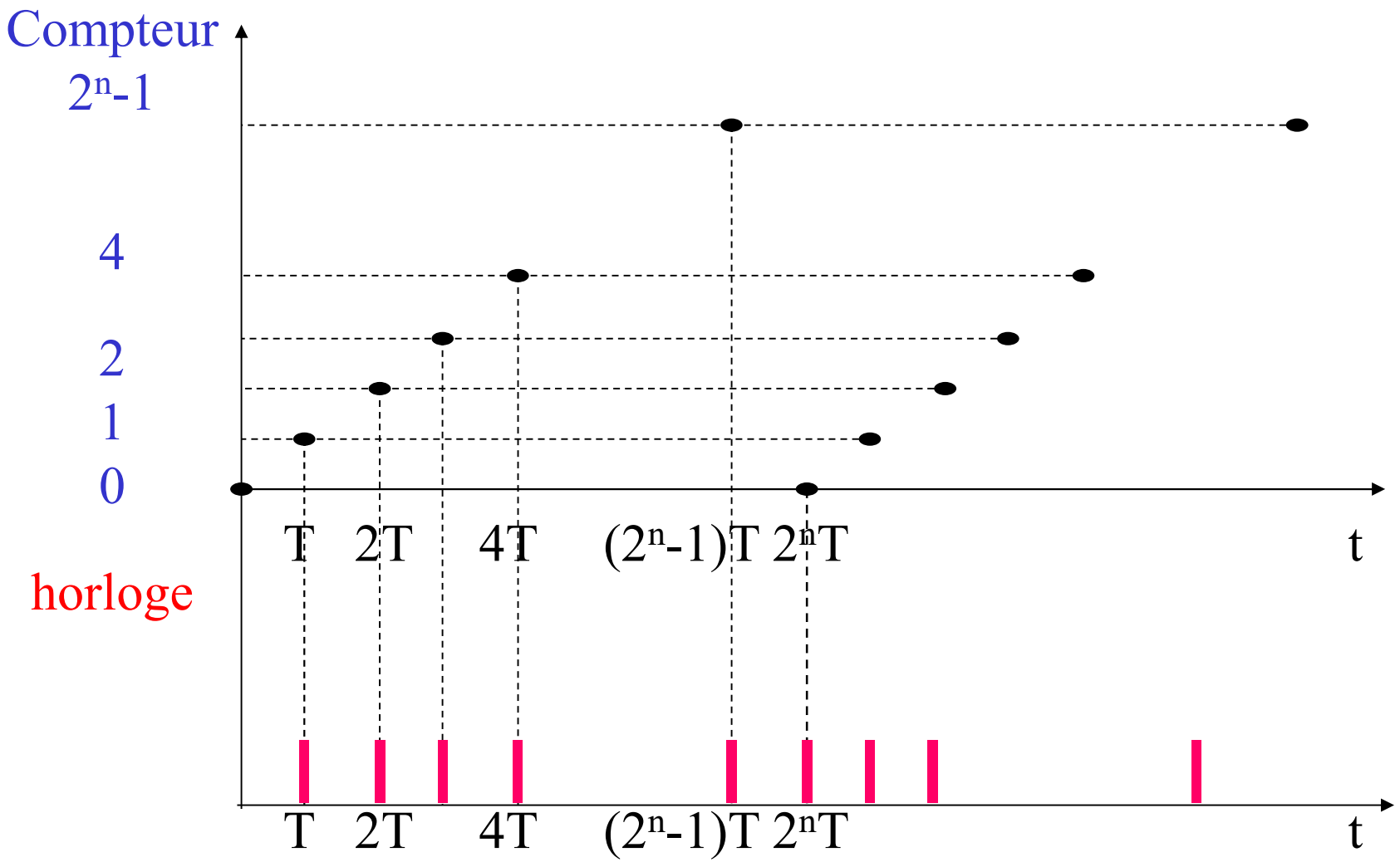


l'événement est consigné dans un reg. d'état et un signal de requête d'IT est émis si les IT ont été armées.

• Le « timer » peut aussi créer sur une borne de sortie

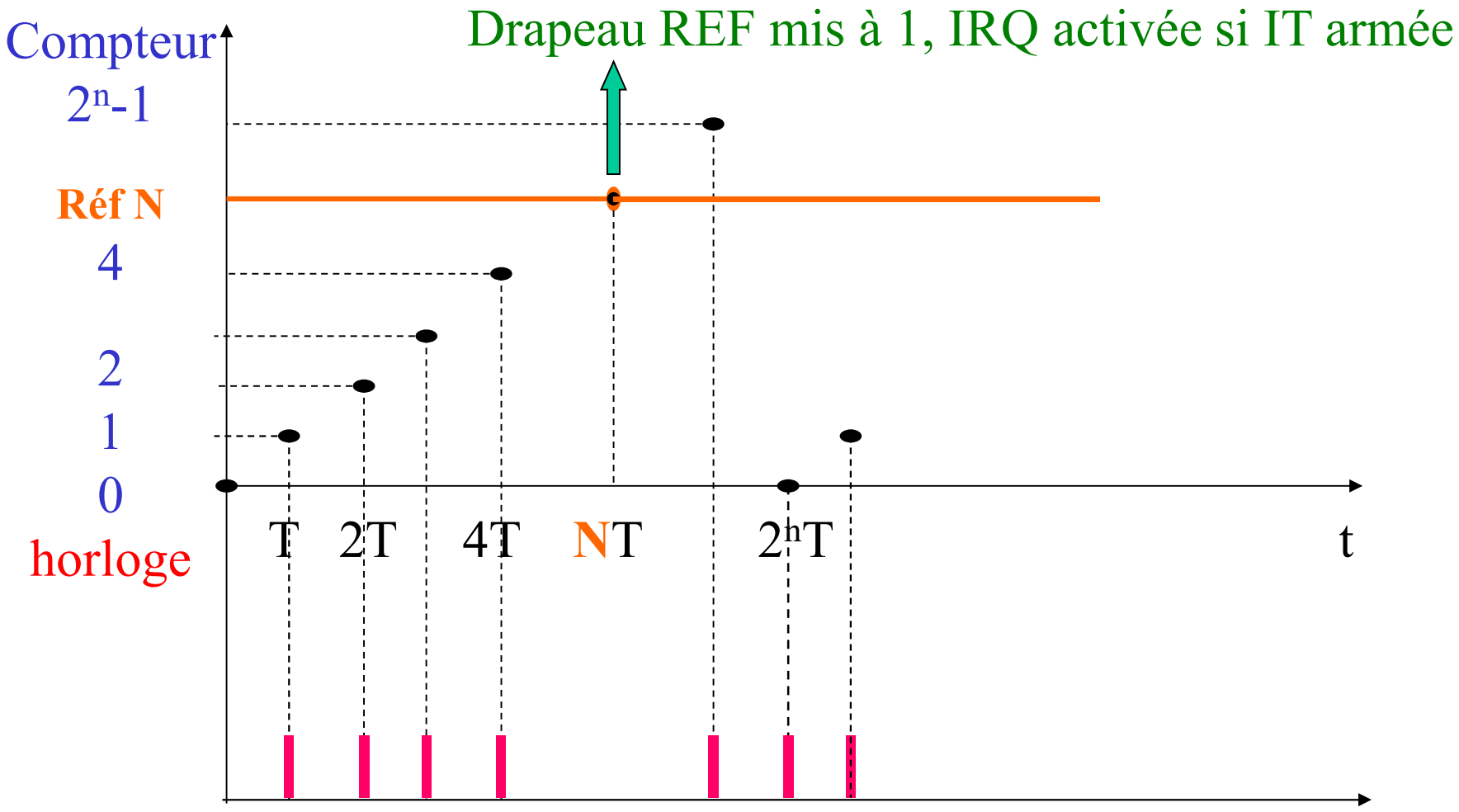
- des signaux rectangulaires ;
- des signaux modulés en largeur (« PWM ») ;
- des impulsions ...

Représentation graphique d'un compteur



L'horloge émet des impulsions avec la période T

Fonctionnement du compteur TCN



Fonctionnement du compteur TCN (mode périodique)

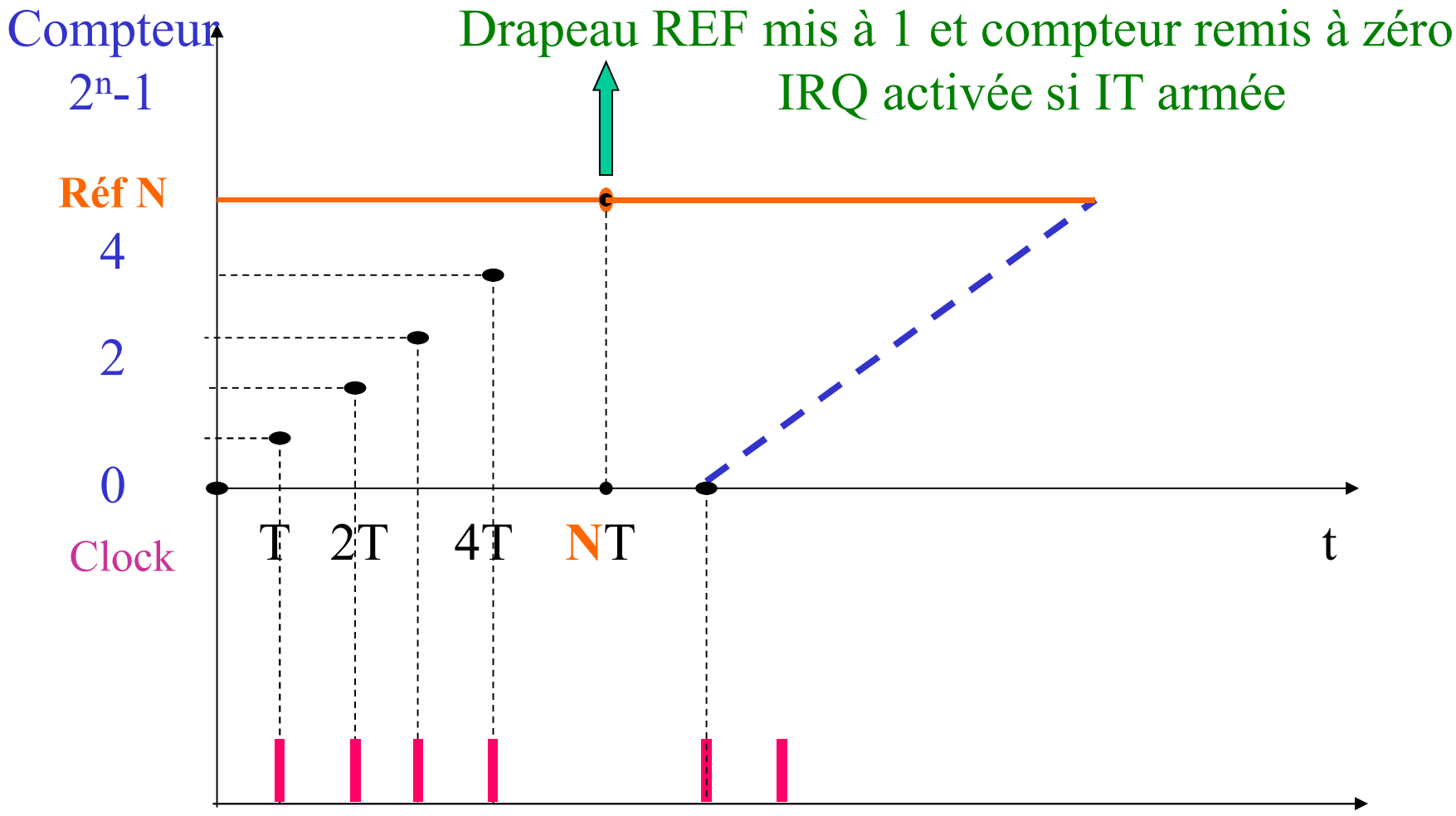
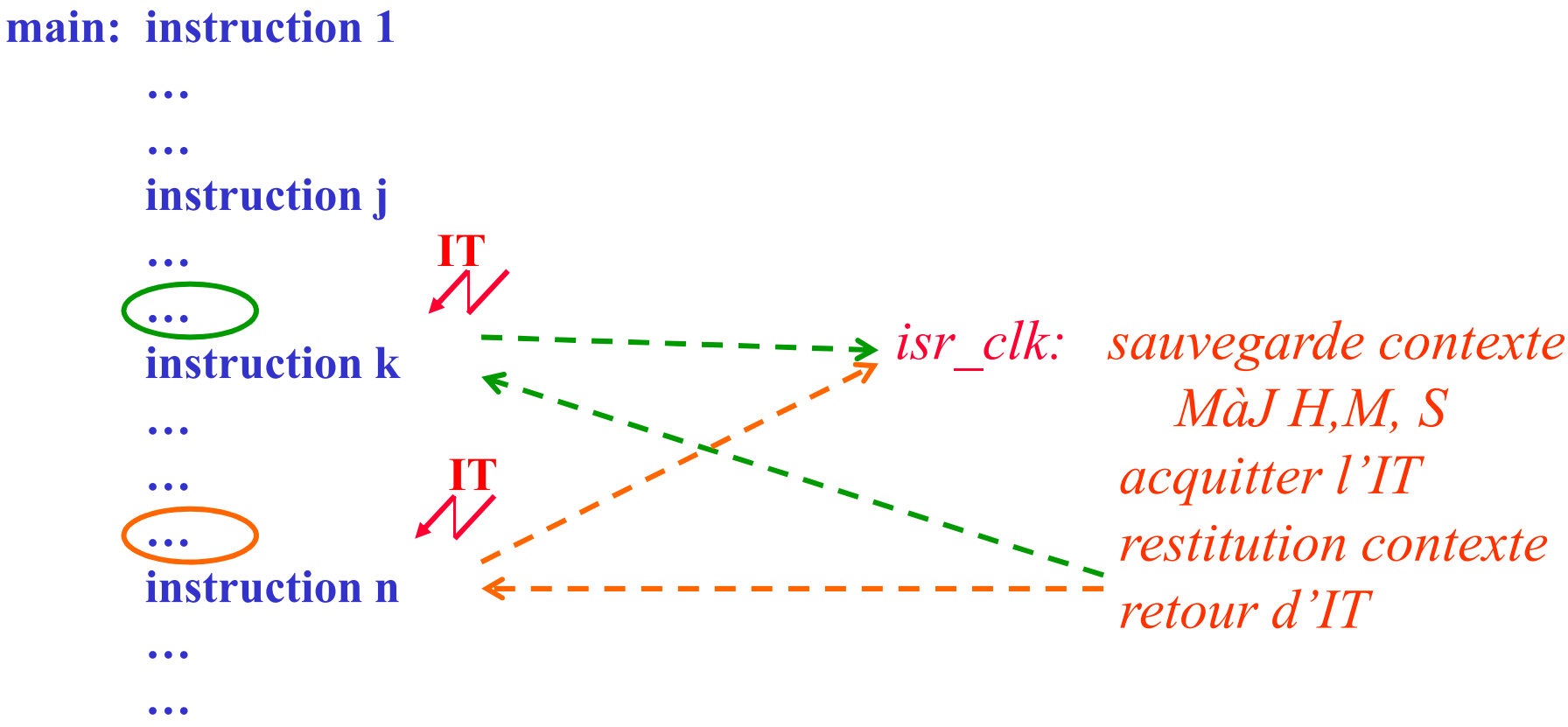


Illustration du traitement des IT Timer



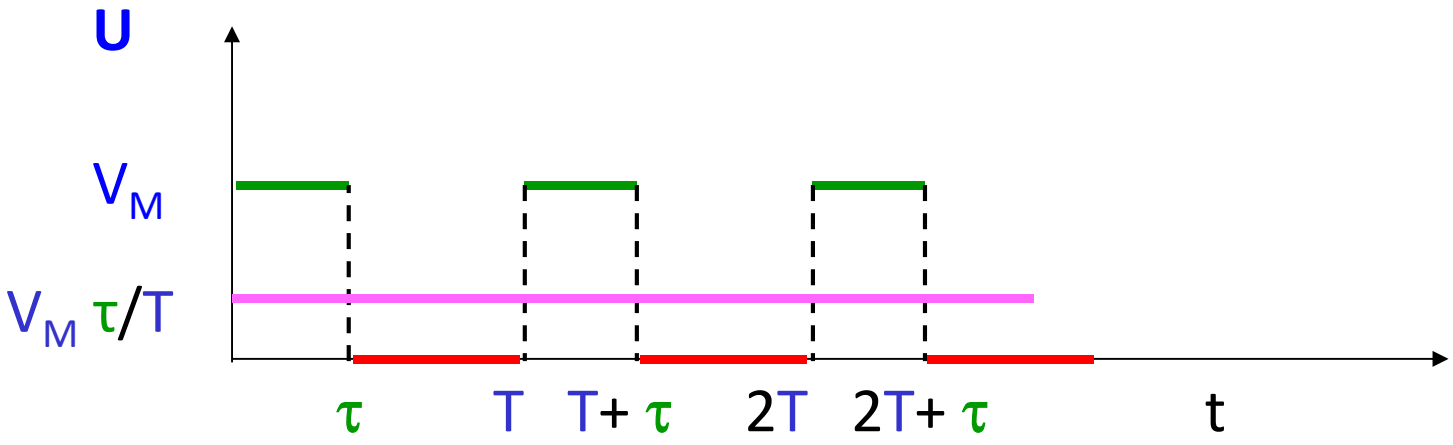
TIMER et PWM
(« Pulse Width Modulation »)

• Certains TIMERS : mode permettant de générer un **signal modulé en largeur** (« PWM ») sur une borne de sortie :

➡ **rapport cyclique τ/T (« Duty cycle »)** du signal modifiable.

Remarque: la valeur moyenne de ce signal est V_M **τ/T**

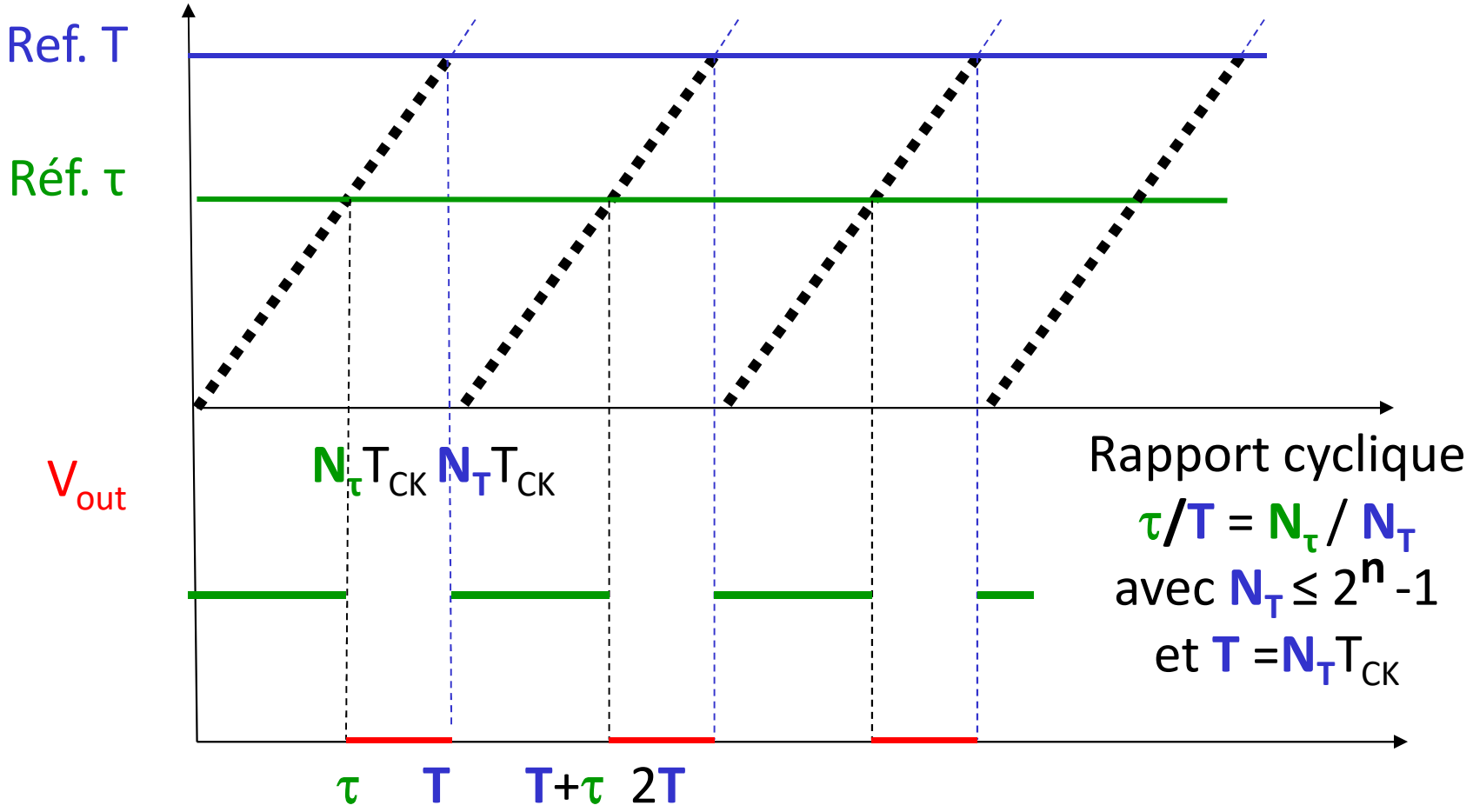
• Un registre contient la valeur $\mathbf{N_\tau}$ permettant de fixer τ.
• Un registre contient une valeur $\mathbf{N_T}$ permettant de fixer **T**.



Création d'un signal PWM (méthode 1)

On a 3 registres contenant respectivement

la valeur pour τ (Duty), celle pour **T** et celle du **compte courant**.

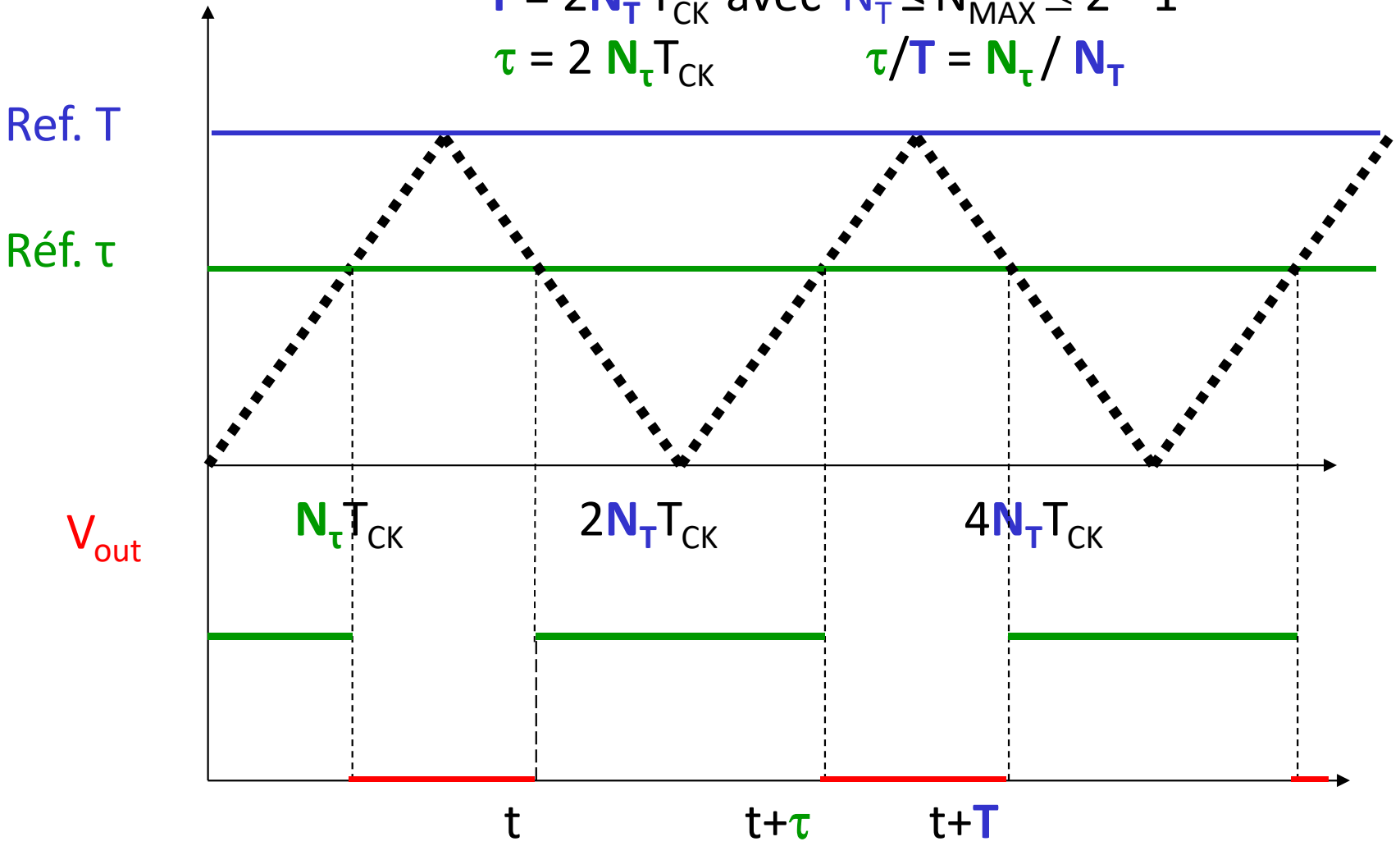


Création d'un signal PWM (méthode 2)

le registre du compte courant est un **compteur-décompteur**.

$T = 2N_T T_{CK}$ avec $N_T \leq N_{MAX} \leq 2^n - 1$

$\tau = 2 N_\tau T_{CK}$ $\tau/T = N_\tau / N_T$



Exercice : TIMER

TCCR1A et TCCR1B : Registres de mode du TIMER 1

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- COM1.. : mettre à 0 pour un fonctionnement en mode normal
- WGM… : idem
- ICNC1, ICES1 : idem
- CS1. : clock select

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Registres de compteur et de comparaison du TIMER 1

• TCNT1 : Compteur 16 bits, à penser à mettre à 0

• OCR1A et OCR1B : deux registres 16 bits servant de valeurs de références. TCNT1 est comparé à ces deux registres à chaque coup d'horloge

Registre de masque d'interruption TIMSK1 du TIMER 1

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- ICIE1 : à mettre à 0 en mode comparaison
- COIE1B : si 1 IT sur quand TCNT1 = OCR1B
- COIE1A : si 1 IT sur quand TCNT1 = OCR1A
- TOIE1 : si 1 IT sur overflow

Registre de masque de flag TIFR1 du TIMER 1

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- ICF1 : flag inutile en mode comparaison
- OCF1B : flag quand TCNT1 = OCR1B (en mode scrutation, mettre un 1 pour libérer ce bit)
- OCF1A : flag quand TCNT1 = OCR1A (idem)
- TOV1 : flag sur overflow

Questions

Le timer étudié fonctionne avec l'horloge interne de fréquence **BCLKO = 22,5 MHz**. Il comporte 8 registres :

- TCCR1A et TCCR1B: choix de mode
- TCNT1 : compteur
- OCR1A et OCR1B : référence
- TIMSK1 : masque IT
- TIFR1 : flag

Lorsque la fonction est **démarrée**, **TCNT1 est mis à jour** (+1) à la **cadence d'une horloge pré-divisée** par la valeur mise dans TCCR1B. Quand le compteur atteint la valeur de référence de **OCR1A et OCR1B**,

- un bit **dans TIFR1** est mis à 1;
- une IT est émise *si les IT sont armées dans* **TIMSK1 ;**

Questions

1. Programmer le timer pour qu'il envoie une IT périodique **toutes les secondes**. On utilisera une variable *timer1* de type enregistrement avec 8 champs *tccr1a, tccr1b, tcnt1,ocr1a, ocr1b, tifr1 et timsk1*. On mettra ces lignes de code dans une fonction ***init_timer()***.
2. Ecrire la procédure de service ***isr_clk()*** qui met à jour une variable globale *temps* de type enregistrement avec 3 champs *heu*, *min*, *sec* de type octet.
3. Le progr. *main,* interrompu par le Timer1, gère une application qui utilise une procédure *horloge()* pour afficher l'instant d'arrivée d'un événement particulier. Montrer par un exemple que le temps affiché peut être erroné. Proposer une solution au problème mis en évidence.

Plan du Cours

I. Introduction au langage C

II. Introduction au Systèmes Embarqués

III. Principes du Processeur

IV. Entrées/Sorties

V. Arduino

Qu'est ce que Arduino ?

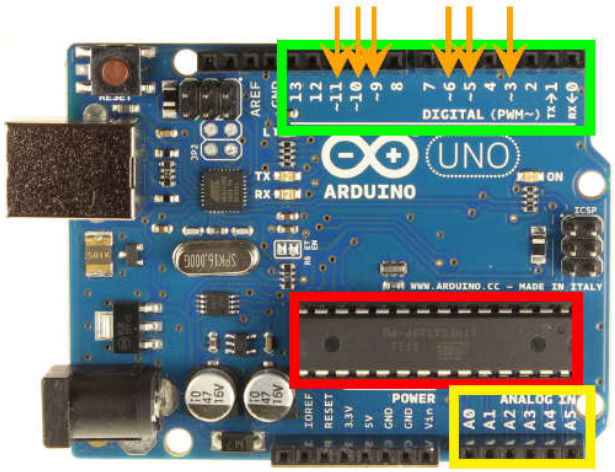
- Start-up italienne fondée en 2005

- Microcontrôleur à bas coût avec un environnement de programmation simplifié

- Hardware open-source (ide) : reproduire, modifier, améliorer
  - ✓ Enorme communauté
  - ✓ Bcp de produits liés à l'Arduino

- Public visé : étudiants, bidouilleur, …

Caractéristiques de l'Arduino Uno



Microcontrôleur :
- 32 ko de ROM
- 2 ko de RAM, 1ko d'EEPROM
- 16 MHz

Entrées / sorties numériques :
- 14
- ON/OFF
- lire interrupteurs, boutons
- contrôler LED, moteurs

Entrées analogiques :
- 6
- lire potentiomètres, capteurs, …
- pas de sortie analogique sauf PWM (modulation) : exemple sortie numérique 3

Caractéristiques électriques :
- tension fonctionnement : 5V
- tension entrée : 6-20 V
- intensité du courant par port E/S : 40mA

Objets à connecter à l'Arduino

Capteurs / actionneurs nus :

- Capteurs de température
- Capteurs de pression
- Capteurs de lumière
- Moteurs pas à pas
- Moteurs à courant continu
- écran LCD
- Relais électrique
- potentiomètre

Modules (shield) : plus complet mais plus cher (permet d'éviter le montage électronique)

Interface de programmation

Programme : sketch
Compiler, reset, transférer
Langage proche du C
3 blocs principaux :

- déclaration variables
- setup : initialiser
- loop : cœur du programme

Fonctions simples :

- pinMode() : choisit un port E/S
- digitalRead/Write() : lit/écrit valeur numérique sur un port
- analogRead/Write() : idem pour une valeur analogique
- delay() : attend une durée

Structure

Fonctions de base

Ces deux fonctions sont obligatoires dans tout programme en langage Arduino :

- void setup()
- void loop()

Structures de contrôle

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Syntaxe de base

- ; (point virgule)
- {} (accolades)
- // (commentaire sur une ligne)
- /* */ (commentaire sur plusieurs lignes)
- #define
- #include

Opérateurs arithmétiques

- = (égalité)
- + (addition)
- - (soustraction)
- * (multiplication)
- / (division)
- % (modulo)

Opérateurs de comparaison

- == (égal à)
- != (différent de)
- < (inférieur à)
- > (supérieur à)
- <= (inférieur ou égal à)
- >= (supérieur ou égal à)

Opérateurs booléens

- && (ET booléen)
- || (OU booléen)
- ! (NON booléen)

Pointeurs

- * pointeur
- & pointeur

Voir également :

- Manipulation des Ports

Opérateurs bit à bit

- & (ET bit à bit)
- |(OU bit à bit)
- ^(OU EXCLUSIF bit à bit)
- ~(NON bit à bit)
- << (décalage à gauche)
- >> (décalage à droite)

Opérateurs composés

- ++ (incrémentation)
- -- (décrémentation) (à revoir)
- += (addition composée)
- -= (soustraction composée)
- *= (multiplication composée)
- /= (division composée)
- &= (ET bit à bit composé)
- |= (OU bit à bit composé)

Variables et constantes

Les variables sont des expressions que vous pouvez utilisez dans les programmes pour stocker des valeurs, telles que la tension de sortie d'un capteur présente sur une broche analogique.

Constantes prédéfinies

Les constantes prédéfinies du langage Arduino sont des valeurs particulières ayant une signification spécifique.

- HIGH | LOW
- INPUT | OUTPUT
- true | false

A ajouter : constantes décimales prédéfinies

Expressions numériques

- Expressions numériques entières
- Expressions numériques à virgule

Types des données

Les variables peuvent être de type variés qui sont décrits ci-dessous.

Synthèse des types de données Arduino

- boolean
- char
- byte
- int
- unsigned int
- long
- unsigned long
- float (nombres à virgules)
- double (nombres à virgules)
- Les chaînes de caractères
- objet String NEW
- Les tableaux de variables
- le mot-clé void (fonctions)
- word
- PROGMEM

Voir également :

- Déclaration des variables

Pour info : les types de données avr-c

Conversion des types de données

- char()
- byte()
- int()
- long()
- float()
- word()

Portée des variables et qualificateurs

- Portée des variables
- static
- volatile
- const

Utilitaires

- sizeof() (opérateur sizeof)

Référence

- Code ASCII

Fonctions

Entrées/Sorties Numériques

- pinMode(broche, mode)
- digitalWrite(broche, valeur)
- int digitalRead(broche)

Entrées analogiques

- int analogRead(broche)
- analogReference(type)

Sorties "analogiques" (génération d'impulsion)

- analogWrite(broche, valeur) - PWM

Entrées/Sorties Avancées

- tone()
- noTone()
- shiftOut(broche, BrocheHorloge, OrdreBit, valeur)
- unsigned long pulseIn(broche, valeur)

Communication

- Serial

Temps

- unsigned long millis()
- unsigned long micros()
- delay(ms)
- delayMicroseconds(us)

Math

- min(x, y)
- max(x, y)
- abs(x)
- constrain(x, a, b)
- map(valeur, toLow, fromHigh, toLow, toHigh)
- pow(base, exposant)
- sq(x)
- sqrt(x)

Pour davantage de fonctions mathématiques, voir aussi la librairie math.h : log, log10, asin, atan, acos, etc...

Nombres randomisés (hasard)

- randomSeed(seed)
- long random(max)
- long random(min, max)

Trigonométrie

- sin(rad)
- cos(rad)
- tan(rad)

Bits et Octets

- lowByte()
- highByte()
- bitRead()
- bitWrite()
- bitSet()
- bitClear()
- bit()

Interruptions Externes

- attachInterrupt(interruption, fonction, mode)
- detachInterrupt(interruption)

Interruptions

- interrupts()
- noInterrupts()

Voir également la librairie interrupt.h.