Architecture des ordinateurs

Jérémy Fix

CentraleSupélec

jeremy.fix@centralesupelec.fr

2017-2018

Cours

- les mémoires et la mémoire cache
- les périphériques : quoi ? canal d'échange, protocole d'échange, prise en compte par interruption

TP

- BE : Interruptions : écoute passive des périphériques
- TL : ordonnanceur : exécuter plusieurs programmes en parallèle avec un chemin de données

La mémoire

Caractéristique des mémoires

Caractéristiques

- Mode d'accès : aléatoire (RAM), séquentiel (disque dur), associatif (cache)
- Capacité d'écriture : ROM (en lecture seule), $\overline{\texttt{ROM}}$ (en lecture et écriture)
- volatilité : maintien des informations en l'absence de courant ? (ROM : forcément non volatile, RAM : ça dépend; SDRAM : volatile ; NVRAM : non volatile ou RAM + pile : CMOS)

Les mémoires mortes

Non volatiles, en lecture seule (≈)

- ROM : Read Only Memory (programmée à la fabrication)
- PROM : Programmable Read Only Memory (fusibles; programmable une fois)
- EPROM : Erasable Programmable Read Only Memory (Ultraviolet)
- EEPROM : Electrically Programmable Read Only Memory (e.g. pour contenir le BIOS : Basic Input Output System; les paramètres sont mémorisés avec une RAM volatile + batterie)

Les mémoires en lecture/écriture

Volatiles, accès aléatoire

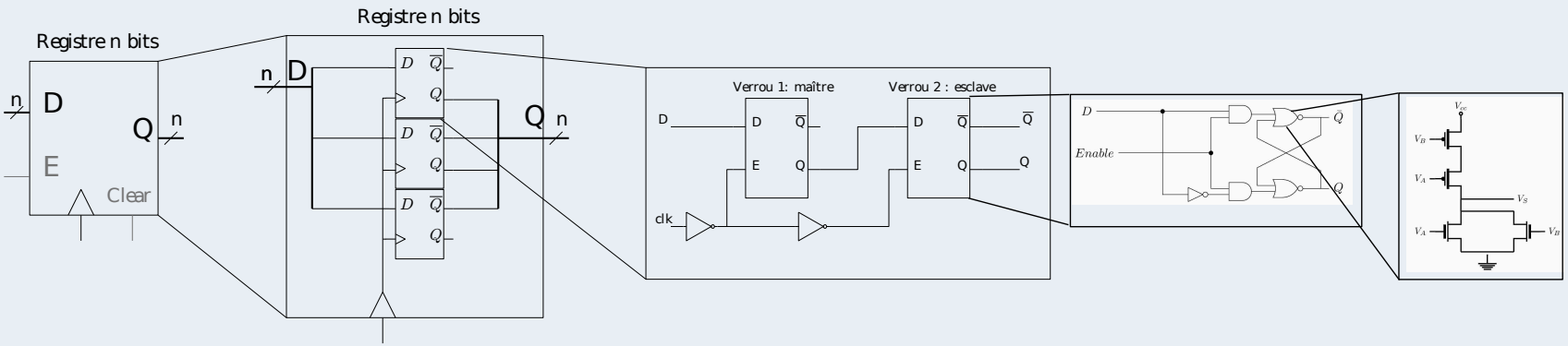
- Registres : bascules D maître/esclave
- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Non-Volatiles

- accès séquentiel : Disques durs magnétiques
- accès aléatoire : disques durs SSD, mémoire flash, nvSRAM, ..

Les mémoires vives

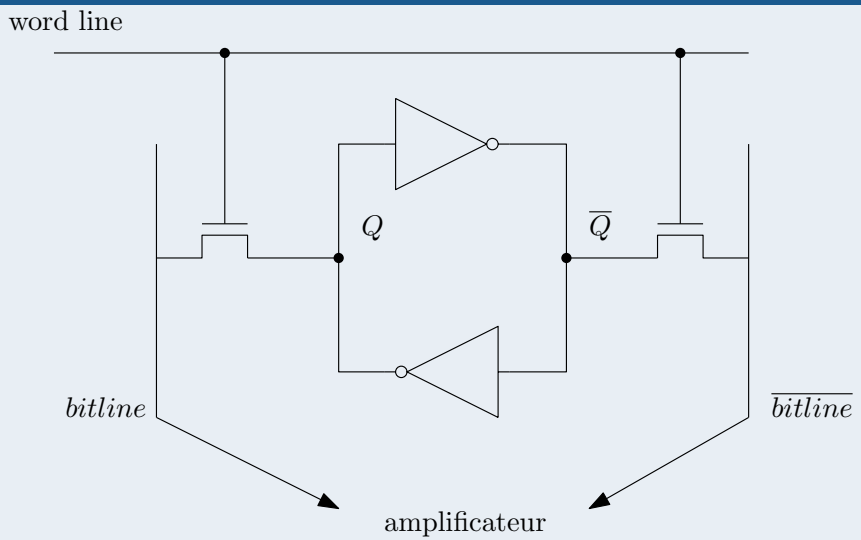
Registres



Capacité : $\approx$ 32-64 bits
Temps d'accès : $\approx$ 1 ps ($10^{-12}s$)
Volatile

Les mémoires vives

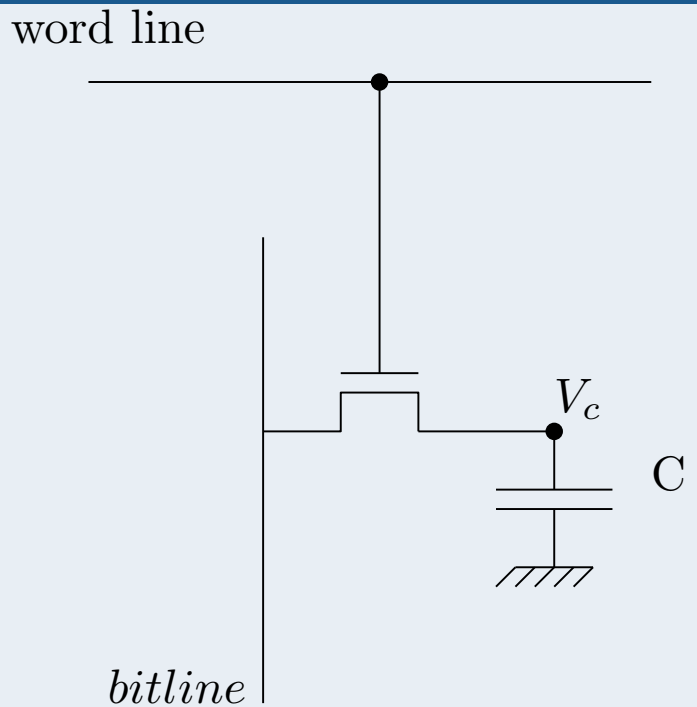
Static Random Access Memory (SRAM)



Capacité : 10 Ko - 10 Mo
Temps d'accès : $\approx$ 1ns
Volatile

Les mémoires vives

Dynamic random access memory (DRAM)



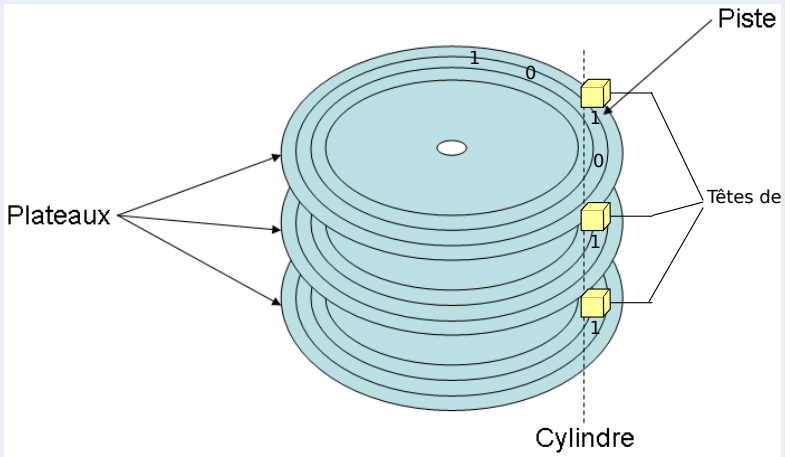
Capacité : 10 Go

Temps d'accès : 80 ns (réécriture toutes les ms)

Volatile

Les mémoires de masse

Disque dur magnétique



Capacité : 1 To
Temps d'accès : 10 ms
Non volatile

Synthèse des mémoires vives et de masse

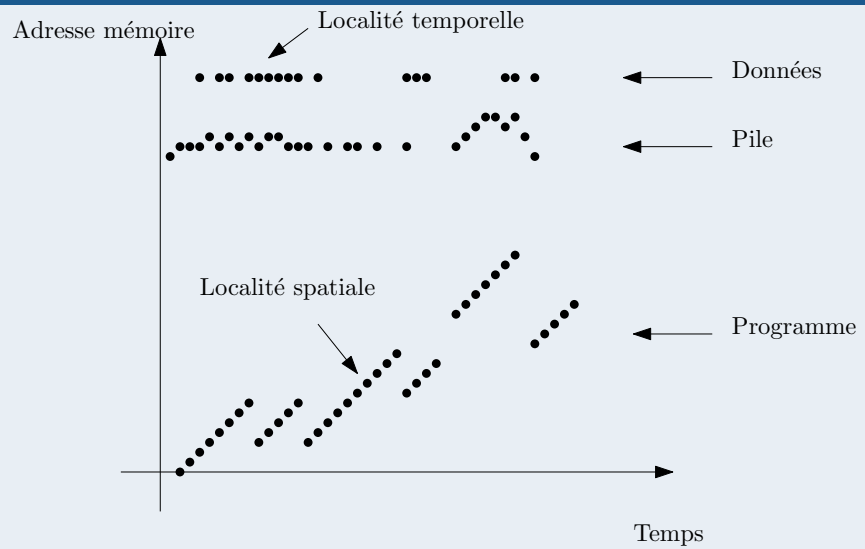
Compromis ?

Type	Capacité	Latence	Coût (au Gi-octet)
Registre	100 bits	20 ps	très cher
SRAM	10 Ko - 10 Mo	1-10 ns	$\approx$ 1000 €
DRAM	10 Go	80 ns	$\approx$ 10 €
Flash	100 Go	100 μs	$\approx$ 1 €
Disque dur Magn	1 To	10 ms	$\approx$ 0.1 €

Une mémoire rapide et de grosse capacité ??

La clé

Les principes de localité



- localité spatiale : les accès futures en RAM se feront à des addresses proches des accès courants
- localité temporelle : une donnée accédée récemment sera certainement réutilisée prochainement

Comment exploiter les principes de localité ?

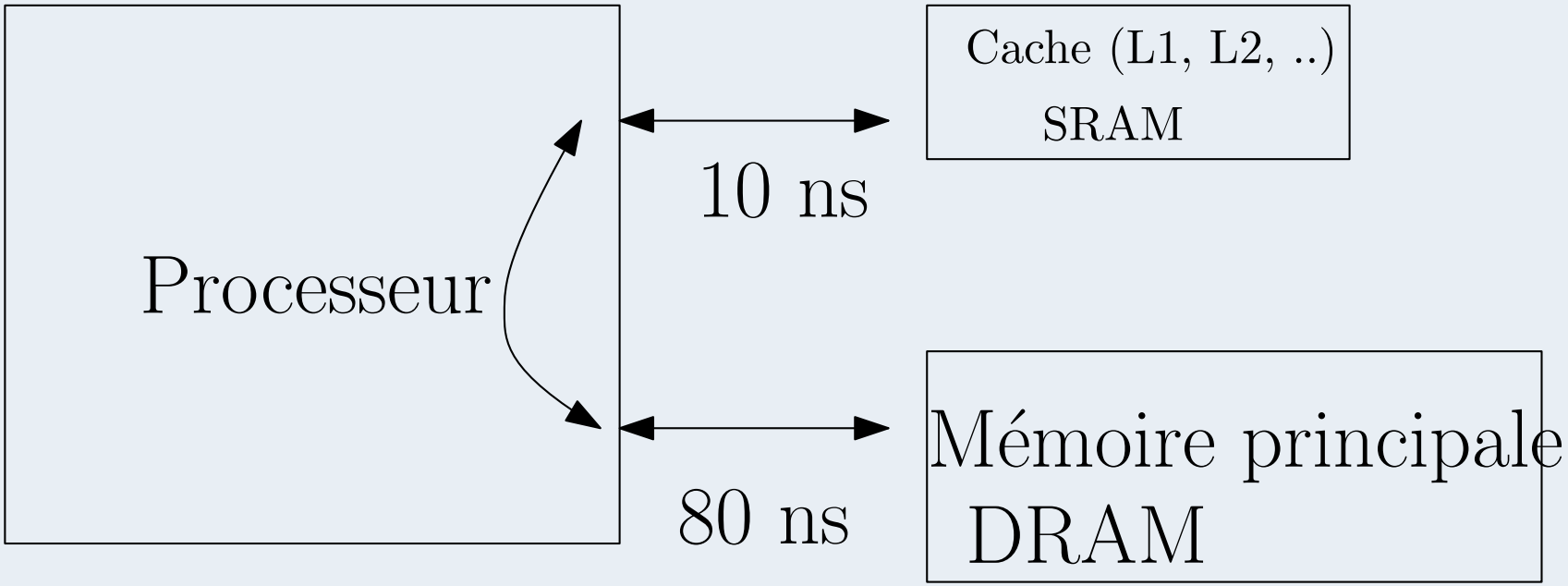
On combine :

1. une petite mémoire rapide avec
2. une grosse mémoire lente

Localité temporelle : on charge en mémoire rapide une donnée à laquelle on accède
Localité spatiale : on charge aussi les voisines
Politique de remplacement ? Dépends de la réalisation...

Comment exploiter les principes de localité ?

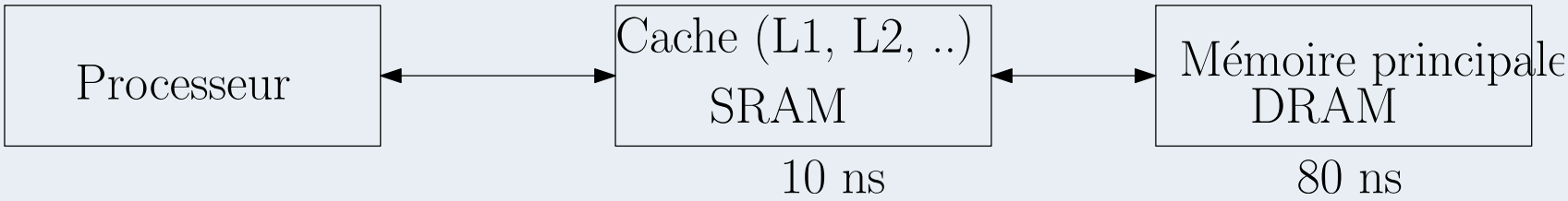
Solution 1) A la charge du programmeur



→ Pas très pratique...

Comment exploiter les principes de localité ?

Solution 2) Hierarchie de mémoires



Performances du cache

- Cache hit : la donnée est dans le cache $\Rightarrow$ accès rapide
- Cache miss : la donnée n'est pas dans le cache $\Rightarrow$ accès lent

Hit ratio : $\frac{hits}{hits+misses}$
Miss Ratio : $\frac{misses}{hits+misses}$
Temps moyen d'accès mémoire $T_m \approx HR.t_{cache} + MR.t_{mem}$
Pour $HR = 90\%$: $T_m = 17ns$.

Principe de fonctionnement d'un cache

Principe

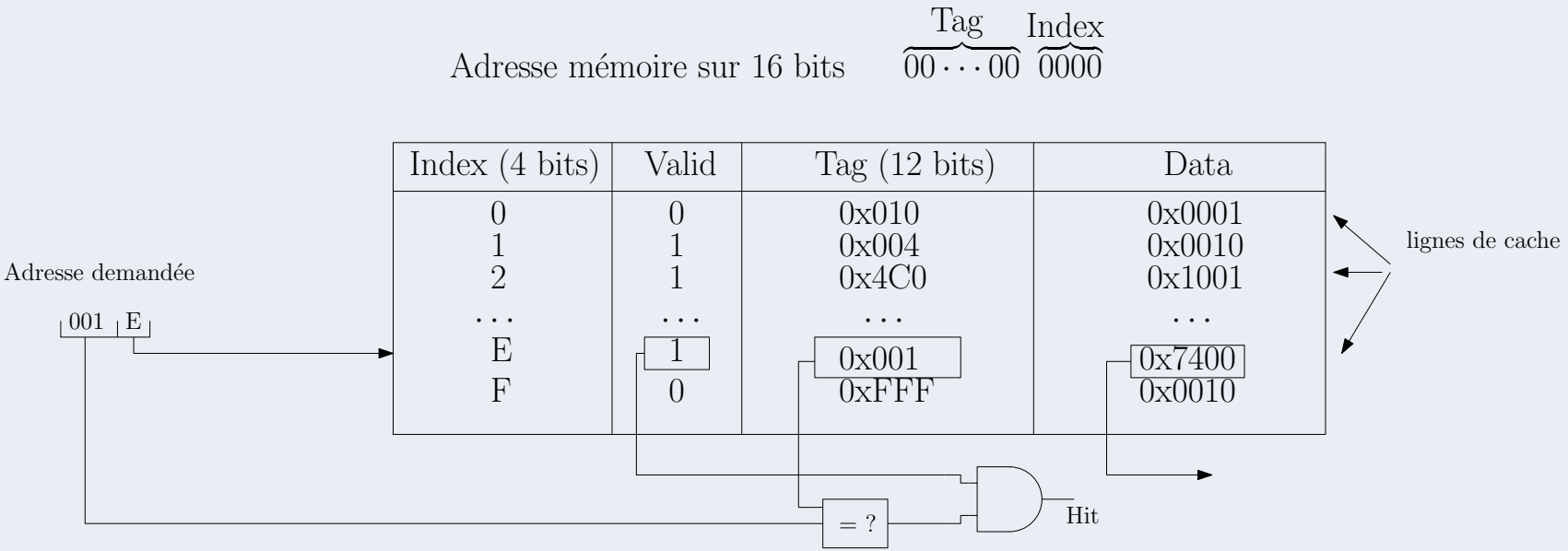
A partir d'une petite information (clé), on retrouve le reste (valeur).
Analogie : annuaire téléphone

En pratique

L'adresse demandée est divisée en plusieurs parties dont une sert de clé.
Tableau associatif : Cache[clé] = {adr, valeur, ...}
Si la donnée est présente en cache : valide et accès rapide
Sinon il faut récupérer la donnée et la placer dans le cache.

Réalisations matérielles d'un cache

Cache à correspondance directe

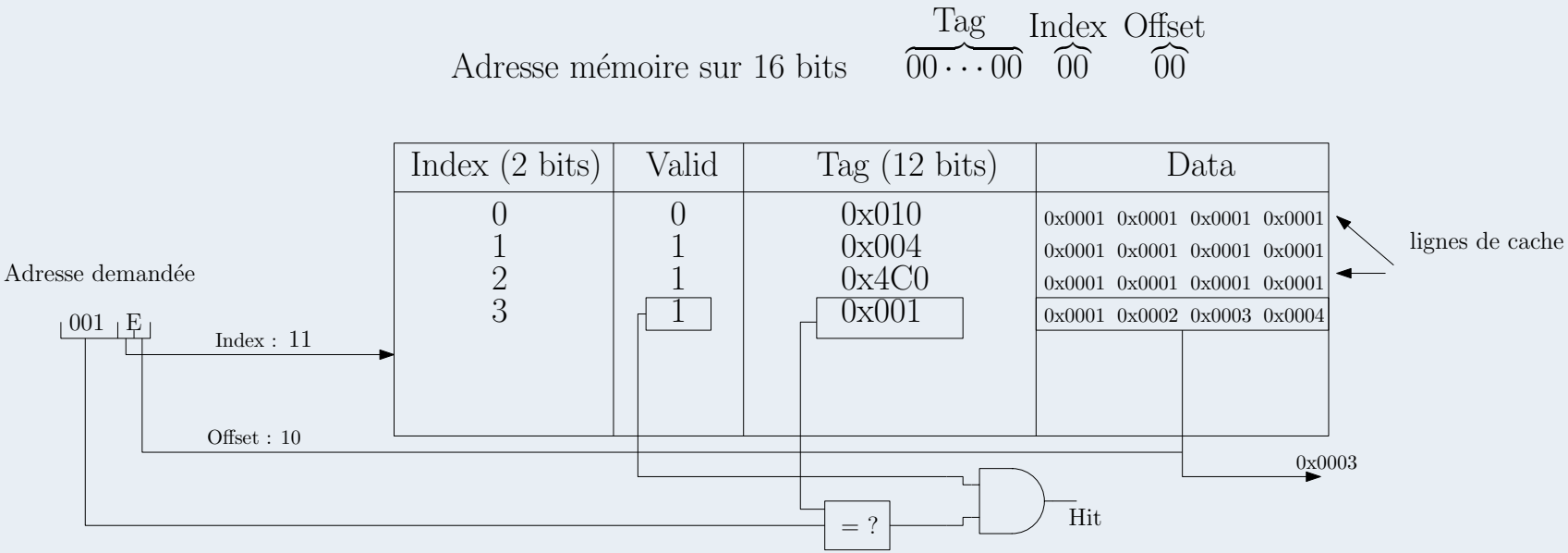


RAM[Adr] $\rightarrow$ Cache[$Adr_3 Adr_2 Adr_1 Adr_0$]

pourquoi les bits de poids faibles comme index ?

Réalisations matérielles d'un cache

Cache à correspondance directe par bloc

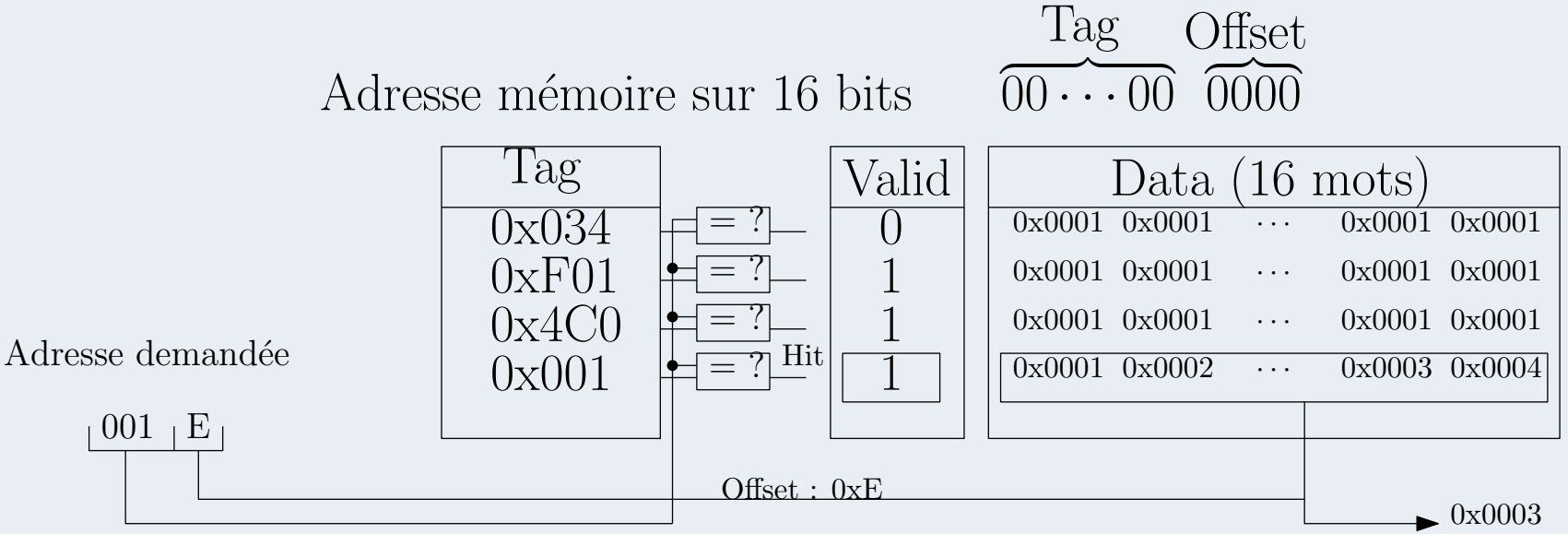


Un bloc : $2^{|offset|}$ mots;

Toutes les adresses de même "Index" ciblent la même ligne.

Réalisations matérielles d'un cache

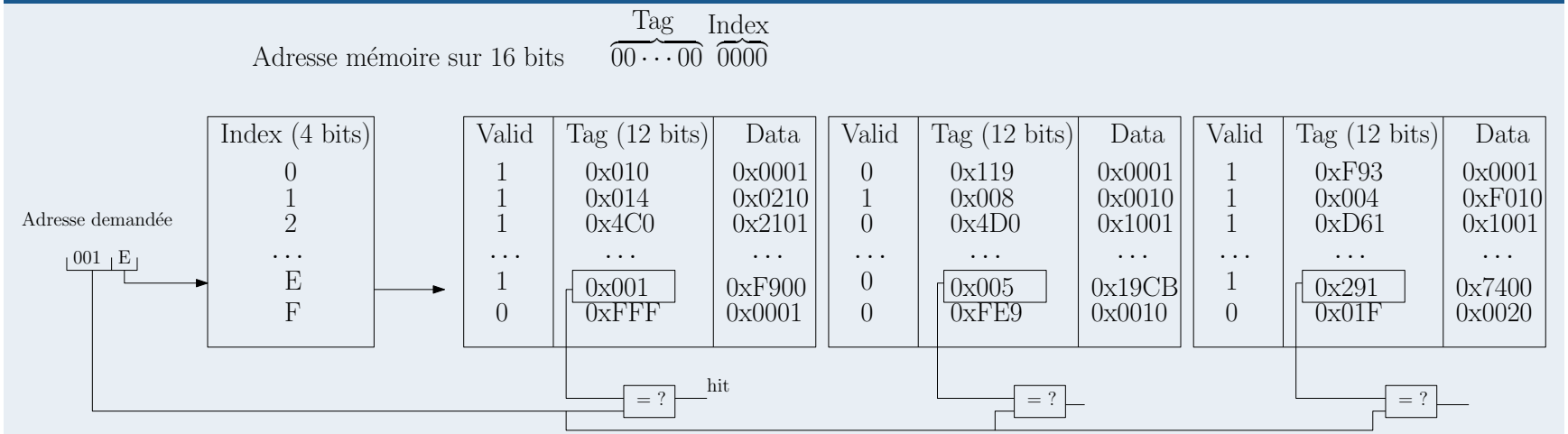
Cache associatif



Une donnée peut occuper n'importe quelle ligne de cache. Lourd matériellement.

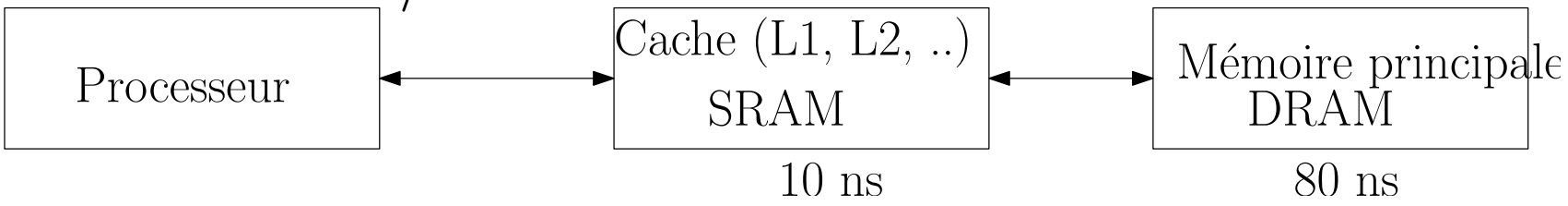
Réalisations matérielles d'un cache

Cache associatif à n entrées



Indexable comme les caches directs, cache[index] étant associatif

Cohérence cache/mémoire centrale



Problème

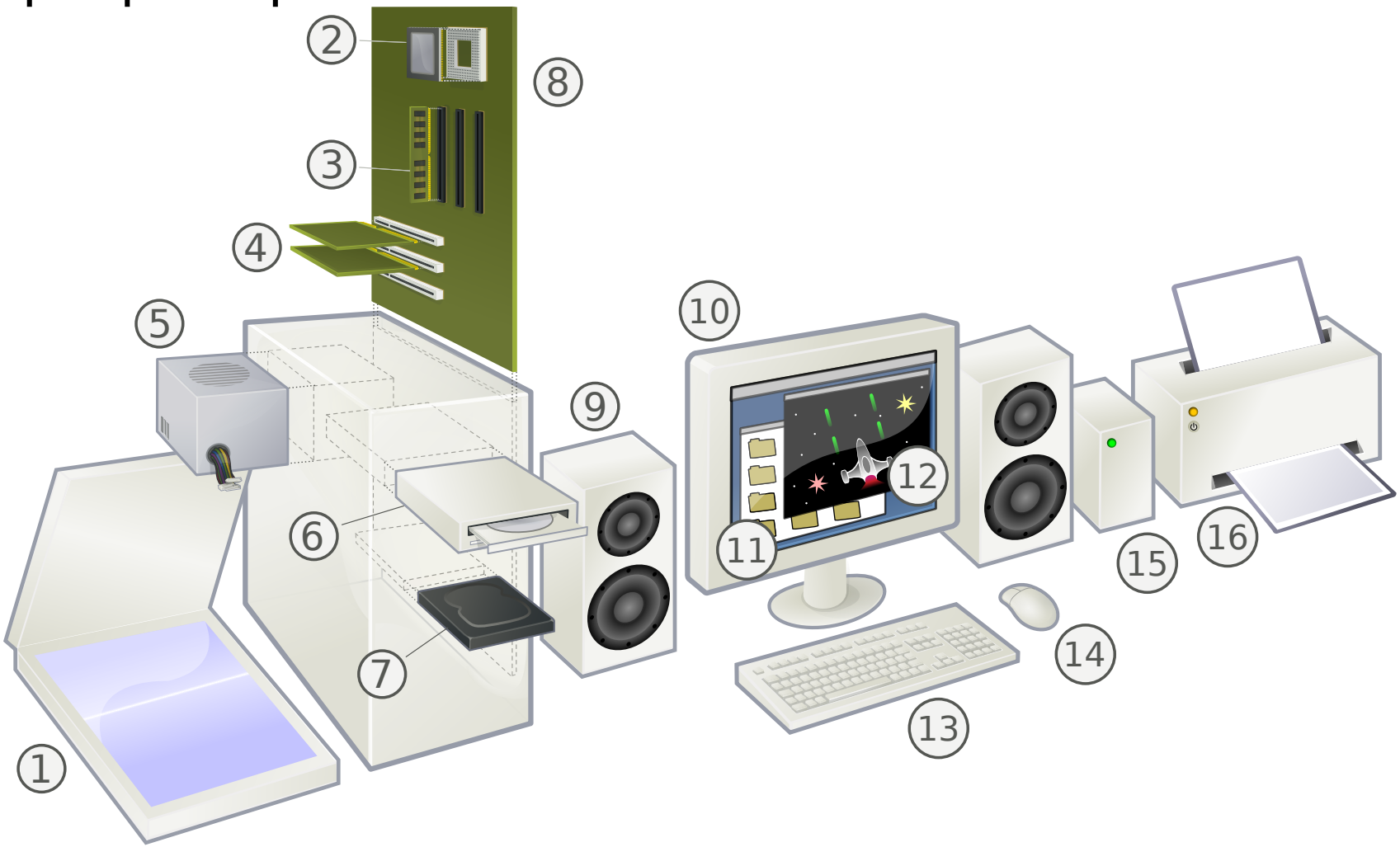
Une donnée d'une **même adresse** peut être à la fois en **cache et** en **mémoire centrale.**

Politiques d'écriture

- Write through : propagation immédiate d'une modification vers la mémoire principale
- Write back : écriture différée; écriture au remplacement (*dirty bit*)

Les périphériques d'entrée/sortie

Les périphériques



http://www.wikipedia.org

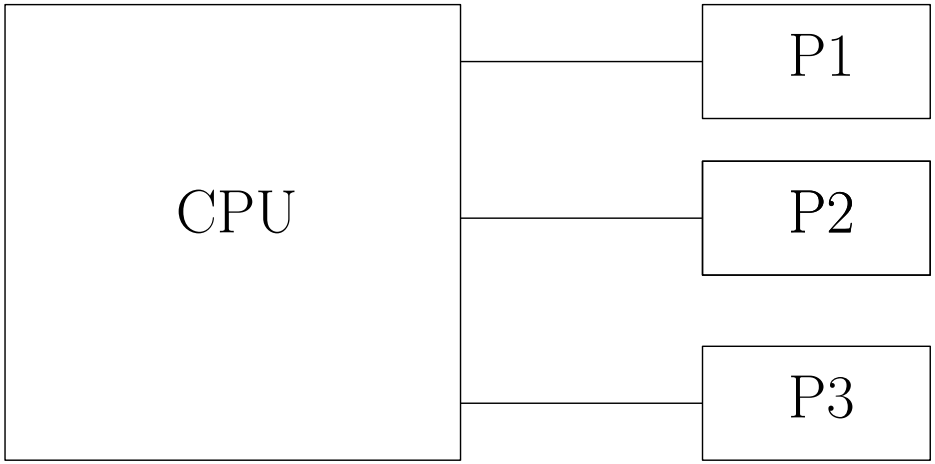
Structure d'un périphérique

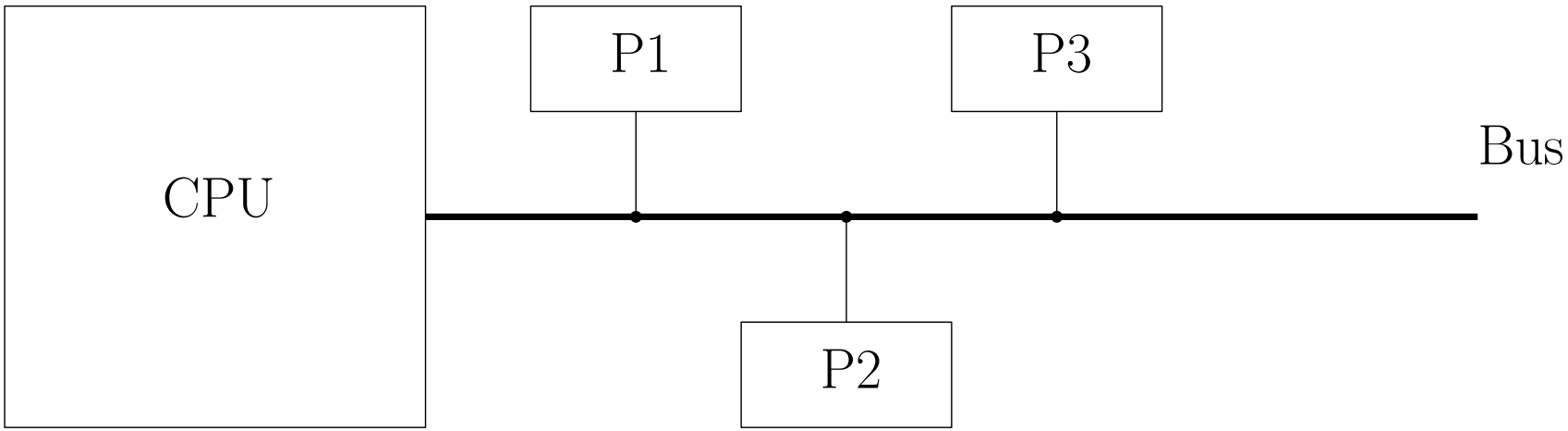
Au périphérique est associé un contrôleur (e.g. contrôleur disque) qui possède :

- des registres
- des mémoires
- des machines à état
- ...

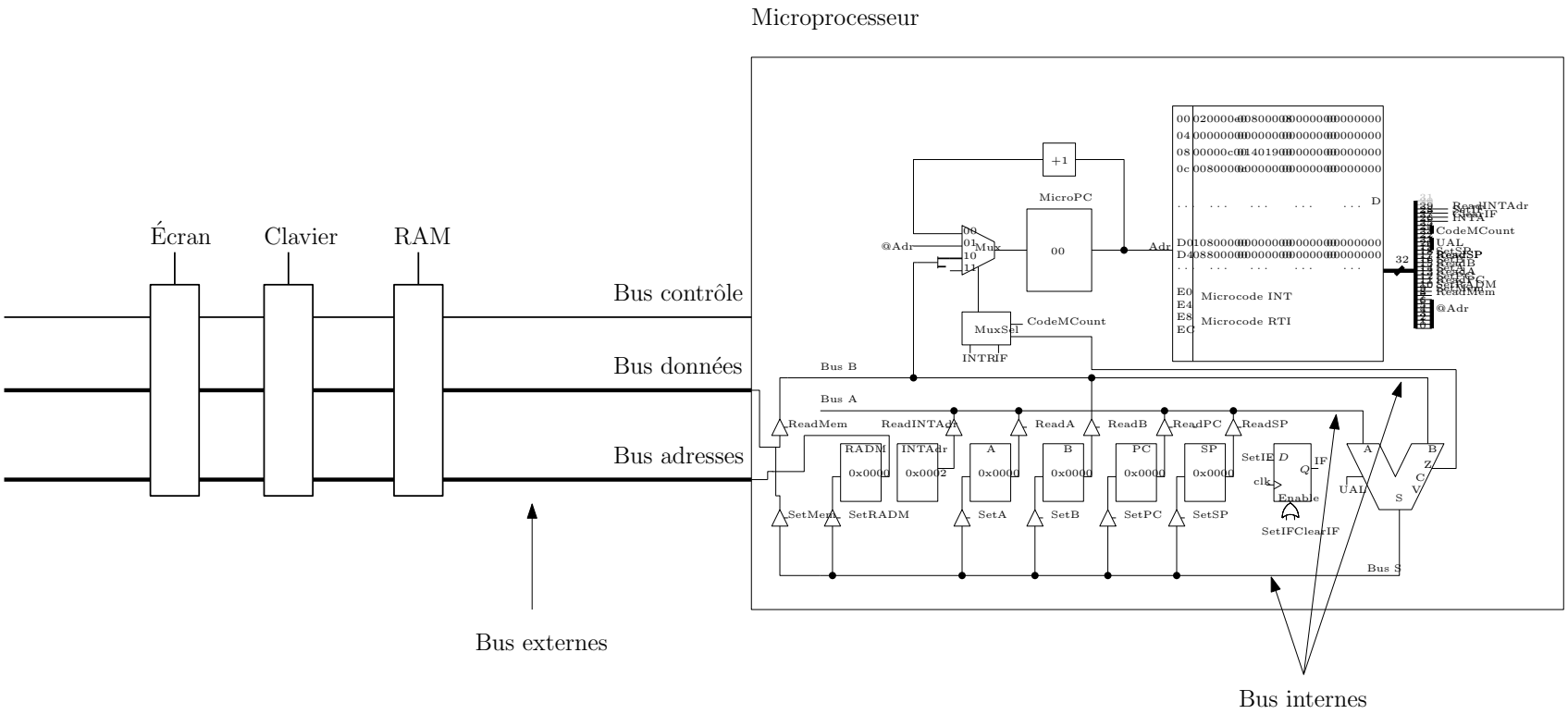
Le périphérique est couplé aux bus d'adresses, données, contrôle.

Interconnexion des périphériques et du chemin de données

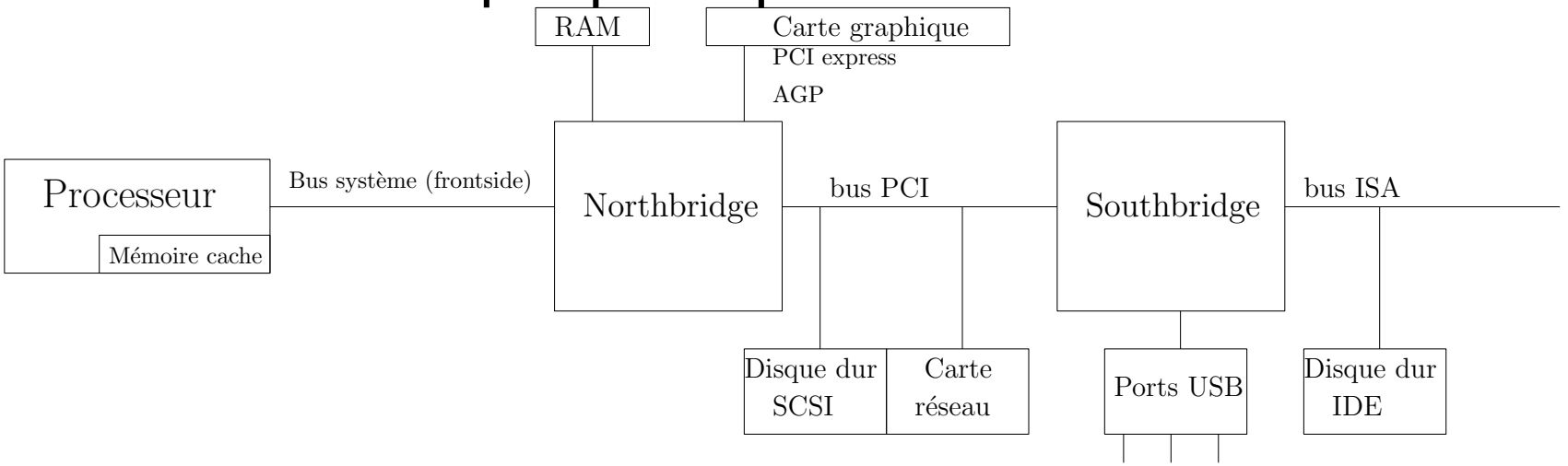




Interconnexion des périphériques et du chemin de données



Interconnexion des périphériques et du chemin de données



Les bus

Bus	Largeur (bits)	Clk (MHz)	Débit (Mo/s)	Année
ISA 16	16	8.33	15.9	1984
PCI 32	32	33	125	1993
AGP	32	66	250	1997
PCI-E 3 (x16)	16	8000	16000	2011

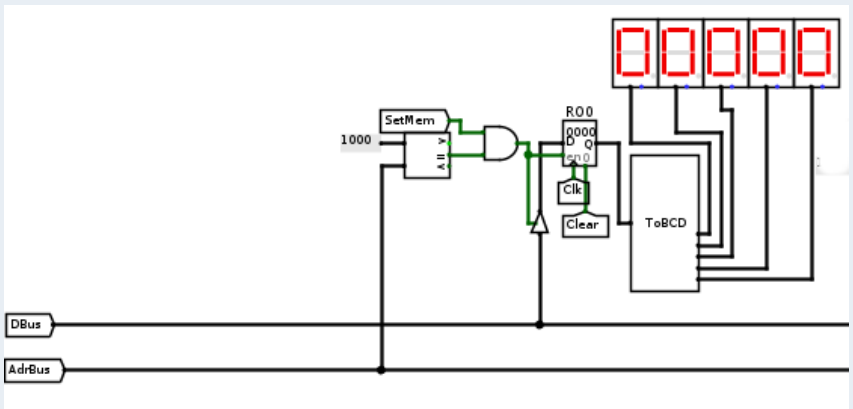
Bus parallèles (échange de mots, large mais court) / Bus série (bit à bit, e.g. USB)

Comment contacter les entrées/sorties ?

Il faut pouvoir contacter les registres du contrôleur du périphérique.

E/S mappées en mémoire

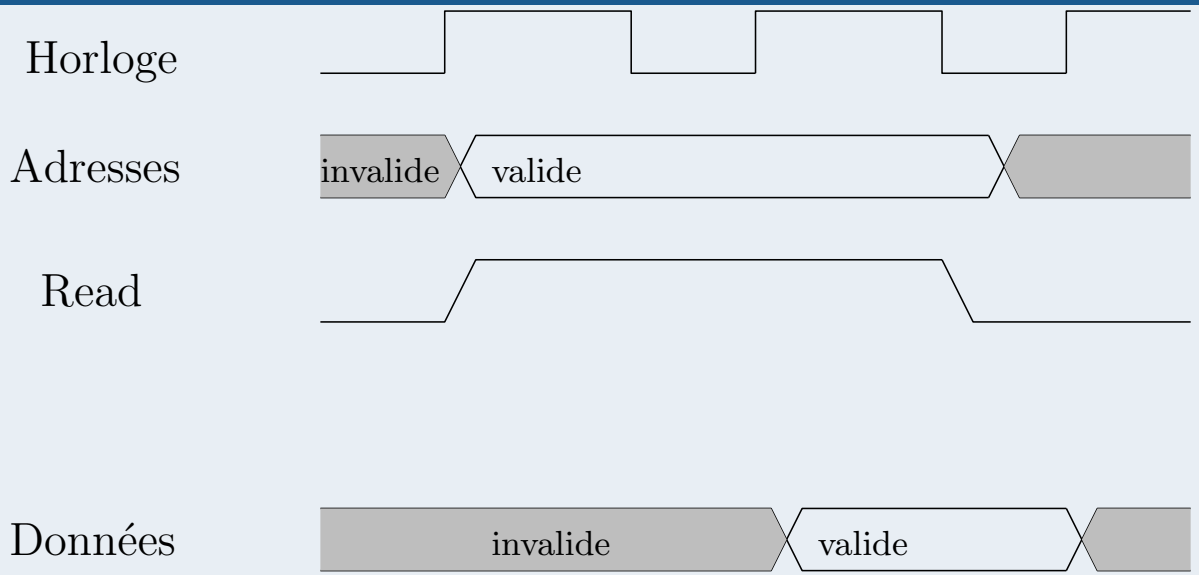
- comme pour un accès mémoire LDA, STA
- avec des adresses réservées, ciblant les périphériques



Autre solution : pas de bus d'adresse; placer dans le message un identifiant du destinataire.

Protocoles d'échange : e.g. lecture mémoire

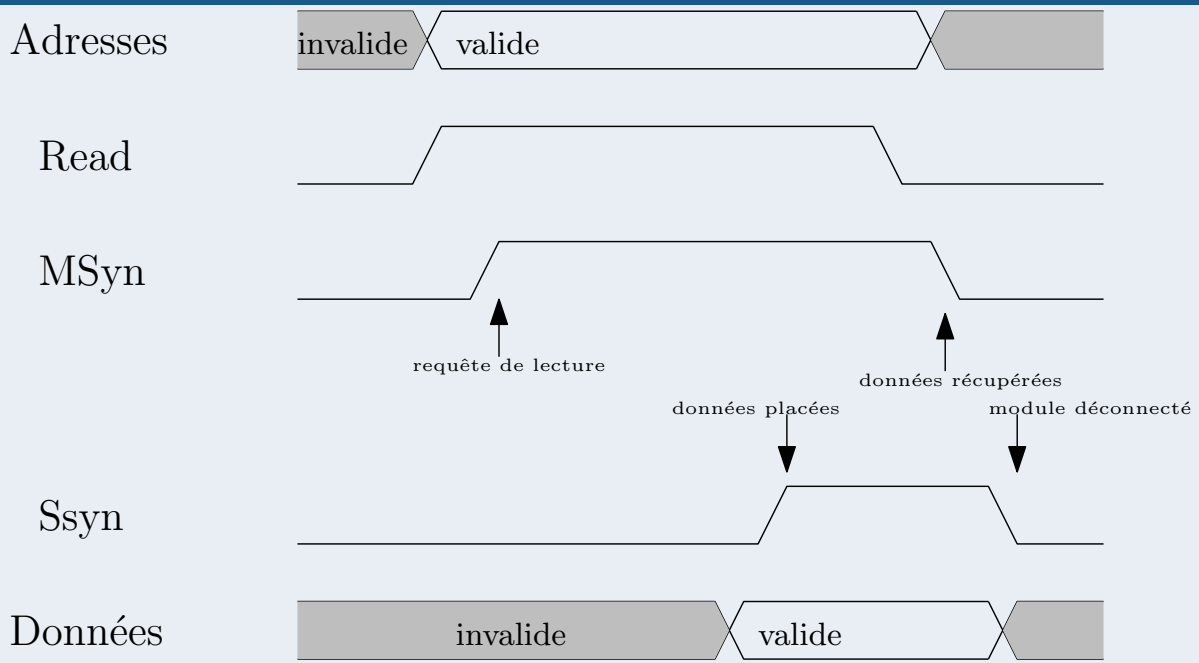
Echange synchrone



Fréquence d'horloge : ajustée au périphérique le plus lent

Protocoles d'échange : e.g. lecture mémoire

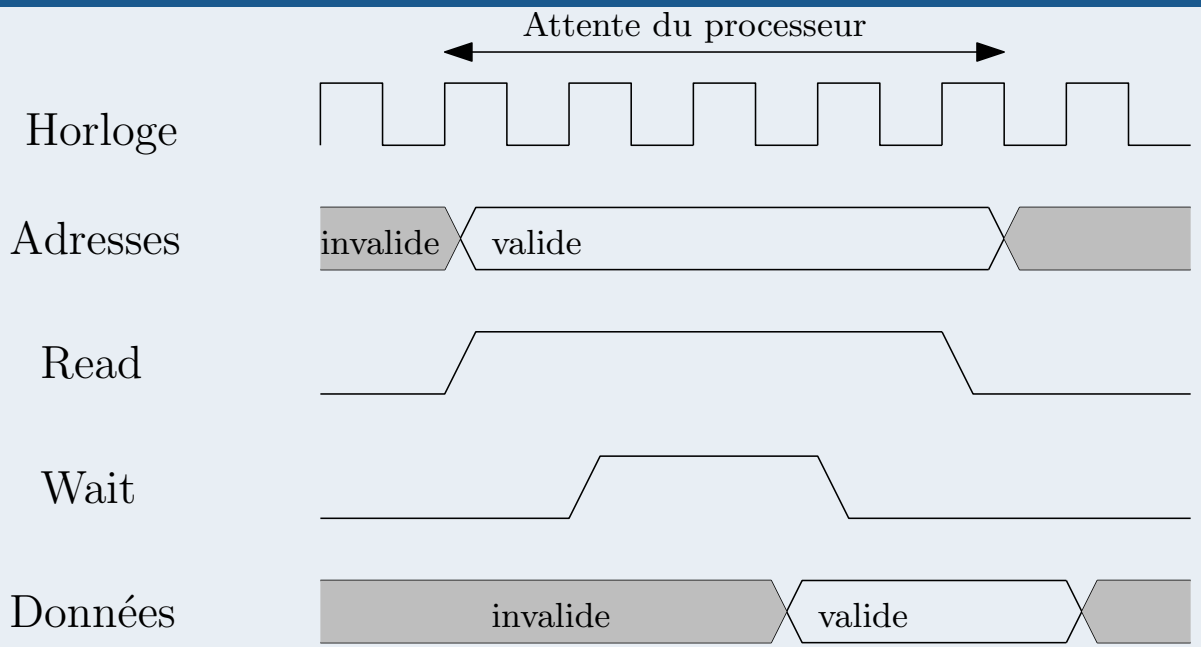
Echange asynchrone : Master/Slave



Pas d'horloge mais des signaux d'état : *handshaking*
Plus réactif mais plus compliqué à mettre en oeuvre.

Protocoles d'échange : e.g. lecture mémoire

Echange demi-synchrone



Attente ? e.g. Wait State : attente active

Et si on est plusieurs à vouloir parler ?

On sait :

- comment interconnecter un périphérique avec le chemin de données : bus
- comment réaliser un transfert synchrone/asynchrone

mais on a :

- “un” bus
- plusieurs acteurs : mémoires, périphériques, CPU, ...

Nécessité d’arbitrer, e.g. le premier dans un certain ordre.
Et comment la demande d’échange du périphérique est prise en charge ?

Gestion des périphériques par interruption

Evènements synchrones

Déroutements (*trap*, *exception*): événements synchrones avec l'exécution d'un programme, liés à une erreur : division par zéro, overflow, accès invalide à la mémoire : e.g. stack overflow

Ex : Overflow

Comment prendre en charge des débordements lors d'opérations arithmétiques ?

1. le programme teste, après "chaque" opération, le bit *overflow* (*Status register*)
   - programme plus long en mémoire et à l'exécution et plus compliqué à écrire
2. dérouter le fil d'exécution si un débordement a lieu (à la JZA); pas de surcoût !

Au regard d'un programme, on peut prévoir quand un déroutement

Evènements asynchrones

Evènements asynchrones; e.g. appui sur un bouton, une touche de clavier, bourrage papier, lecture mémoire terminée, ...

Prise en charge par scrutation

On interroge régulièrement le périphérique

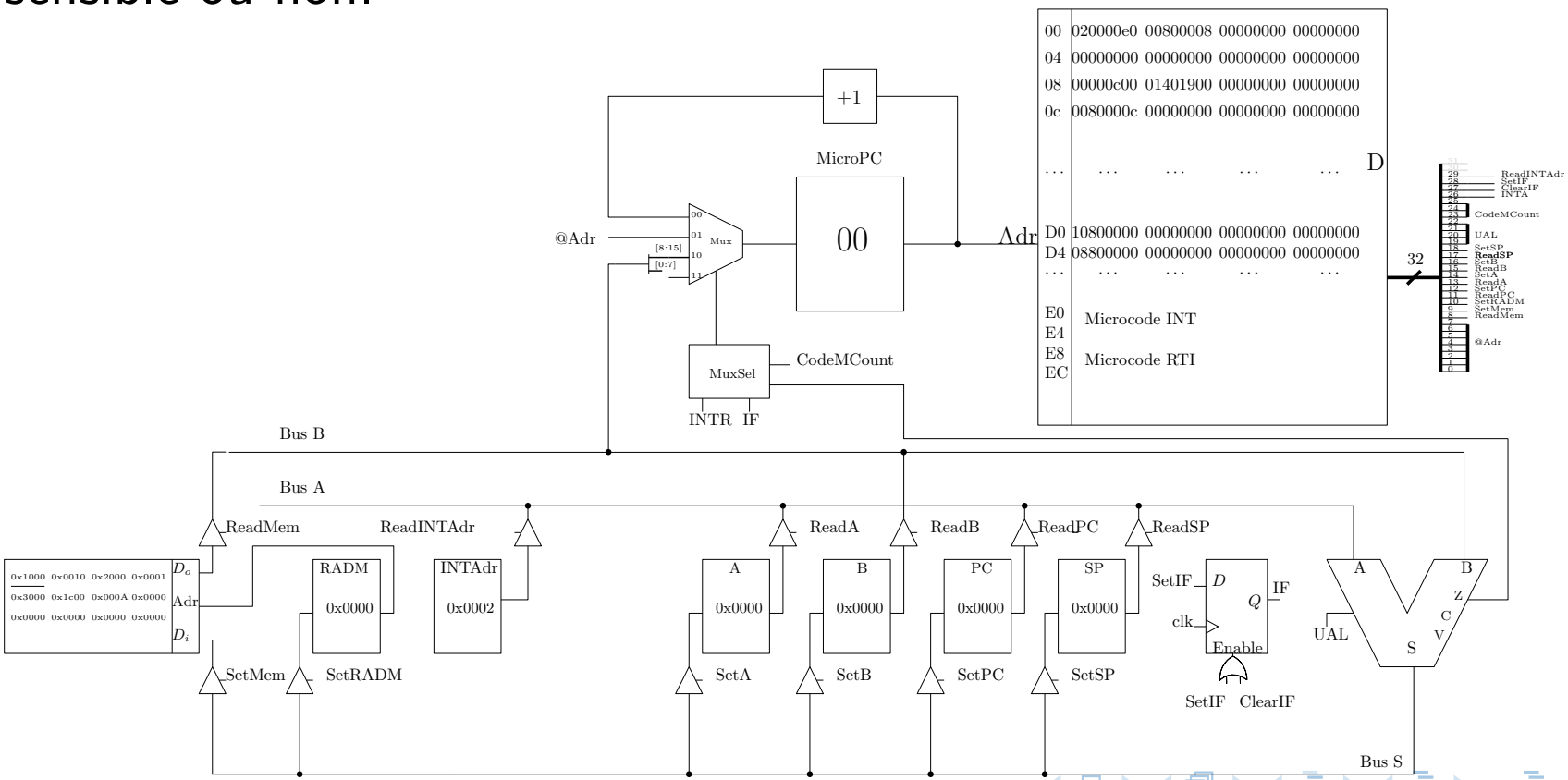
- programmation explicite
- surcoût à l'exécution

Prise en charge par interruption

Ecoute passive du périphérique : exécute un programme particulier que lorsque le périphérique lève une demande d'interruption
$\rightarrow$ sollicite moins le processeur (e.g. lecture sur disque : données prêtes)

Modification du chemin de données

On ajoute une ligne de requête d'interruption (INTR) qu'un périphérique peut mettre à l'état haut. Registre IF pour y être sensible ou non.



Ecoute passive des demandes d'interruption

Avant le fetch/decode, on test si une demande d'interruption est levée :

- sinon, on entre dans la phase de fetch/decode et exécution de l'instruction
- si oui, on part en interruption

CodeMCount	Z	INTR & IF	$S_1 S_0$	Sémantique
000	-	-	00	MicroPC := MicroPC+1
001	-	-	01	MicroPC := @Adr
010	-	-	10	MicroPC := Instruction
011	0	-	00	MicroPC := MicroPC+1 si $S \neq 0$
011	1	-	01	MicroPC := @Adr si $S = 0$
100	-	0	01	MicroPC := @Adr si $\overline{INTR\&IF}$
100	-	1	00	MicroPC := MicroPC+1 si $INTR\&IF$

Prise en charge de la demande d'interruption

Principe

L'interruption doit être **gérée de manière transparente** pour le programme interrompu
$\Rightarrow$ sauvegarde du contexte d'exécution (sur la pile)

Nouvelles instructions

- INT (0xE000) : partir en interruption :
  1. sauvegarde du contexte d'exécution sur la pile
  2. chargement de l'adresse du programme d'interruption
- RTI (0xE800): revenir d'une interruption :
  1. restauration du contexte d'exécution du programme interrompu
- STI (0xD400) : démasque les interruptions : IF := 1
- CLI (0xD000) : Masque les interruptions : IF := 0

Comment savoir quel programme exécuter en cas d'interruptions

Vecteur d'interruption : programme à exécuter en cas d'interruption

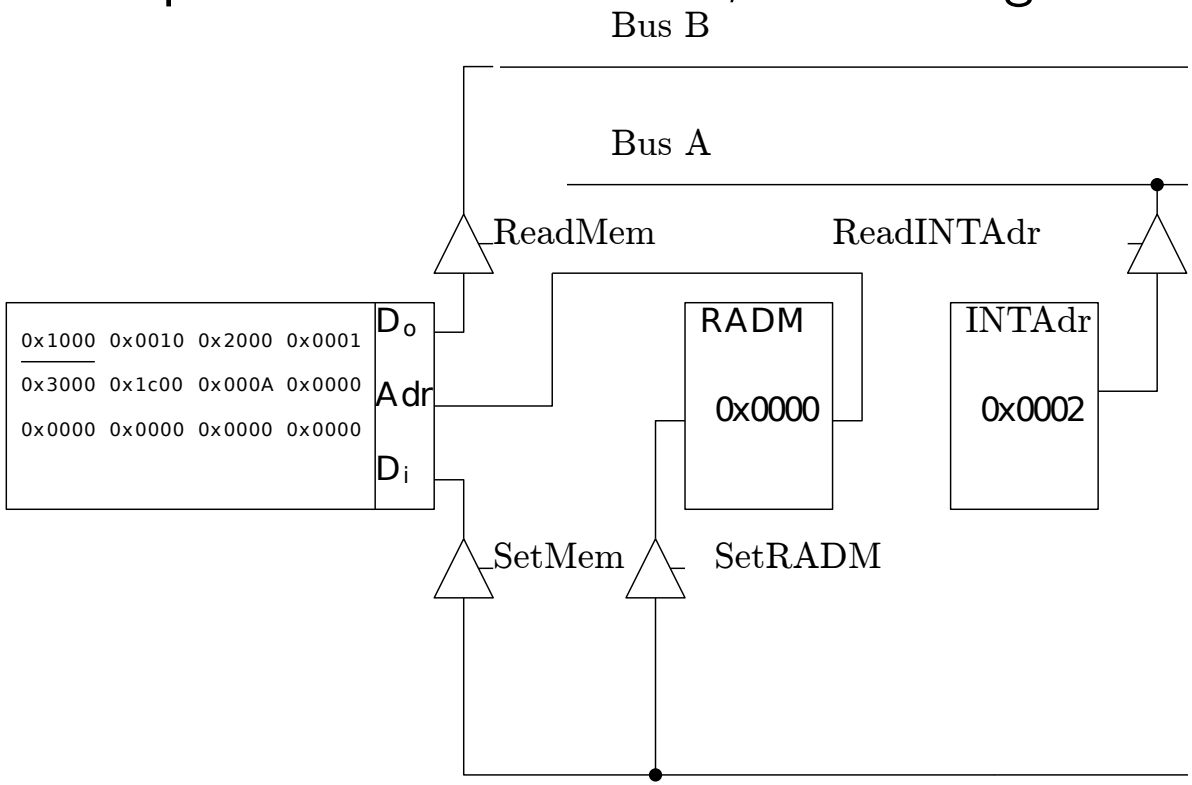
On stocke une table des vecteurs d'interruption en début de mémoire :

```
        JMP init
        JMP introutine
init:   ... ; le programme principal
        ...

introutine: ... ; la routine d'interruption
            ...
            RTI ; le retour d'interruption
```

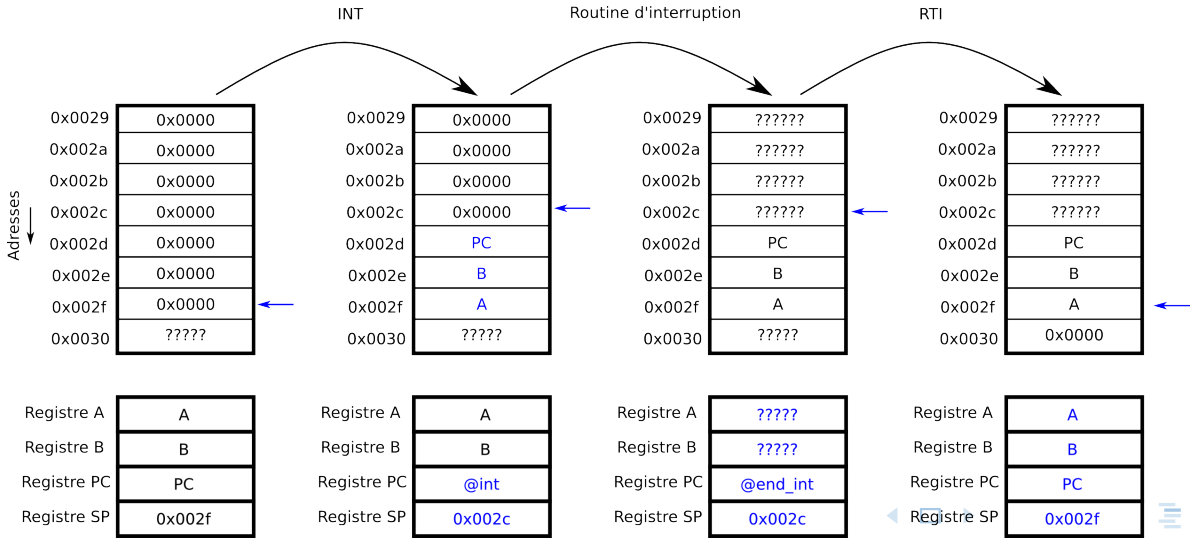
Chez nous : une seule interruption

Nous ne prendrons ici en charge qu'une interruption, l'adresse de la routine d'interruption est codée en dur, dans le registre INTAdr



Résumons

1. le périphérique lève une requête d'interruption INTR=1
2. le processeur détecte la requête
3. le processeur accuse réception
4. le processeur sauvegarde l'état courant et se branche sur la routine d'interruption (vecteur d'interruption ou *interrupt handler*)
5. le processeur restaure l'état



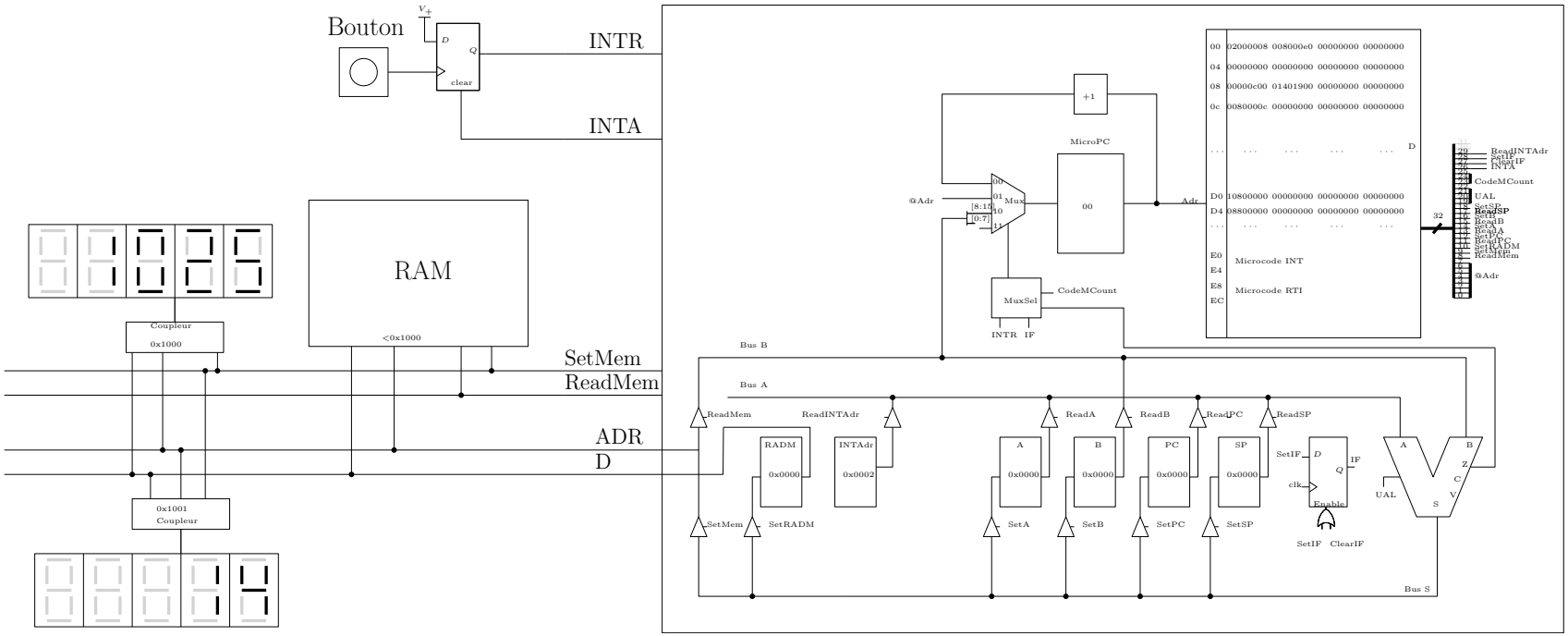
L'initialisation

Attention

La gestion de l'interruption implique la pile !!
$\Rightarrow$ Phase d'initialisation (JMP init) minimale :

1. par défaut IF = 0
2. LDSPI ...
3. STI

Exemple jouet : En BE

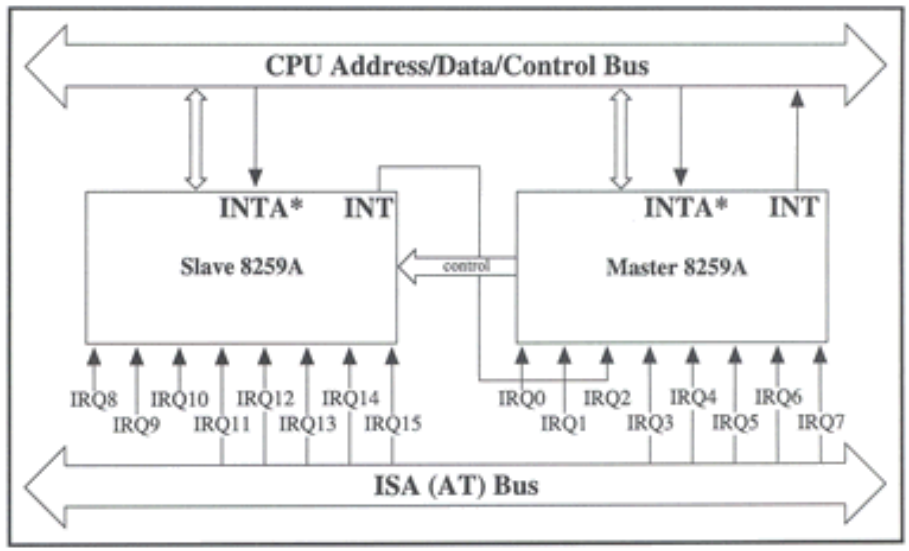


Plusieurs interruptions ?

On s'est limité pour le moment à un seul périphérique.

Prise en charge de plusieurs interruptions

1. Plusieurs lignes d'interruptions
2. Centralisation et arbitrage des demandes d'interruptions; Le vecteur d'interruption est récupéré sur le bus de données



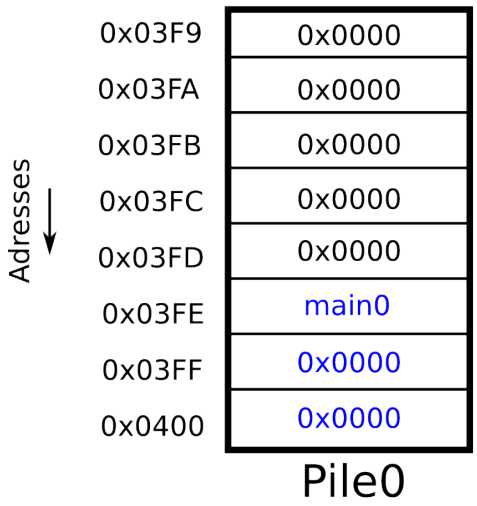
Petits pas vers l'OS : Ordonnanceur

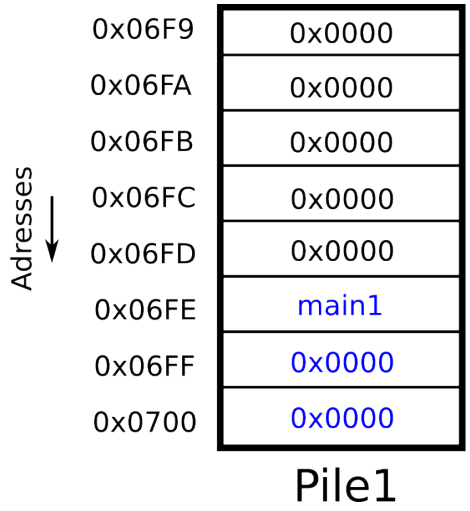
Problème

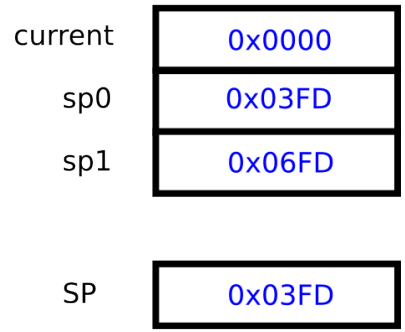
Comment exécuter plusieurs programmes "en même temps" :

1. avec plusieurs chemins de données (architectures multi-coeurs)
2. avec un seul chemin de données partagé par deux programmes : en TL

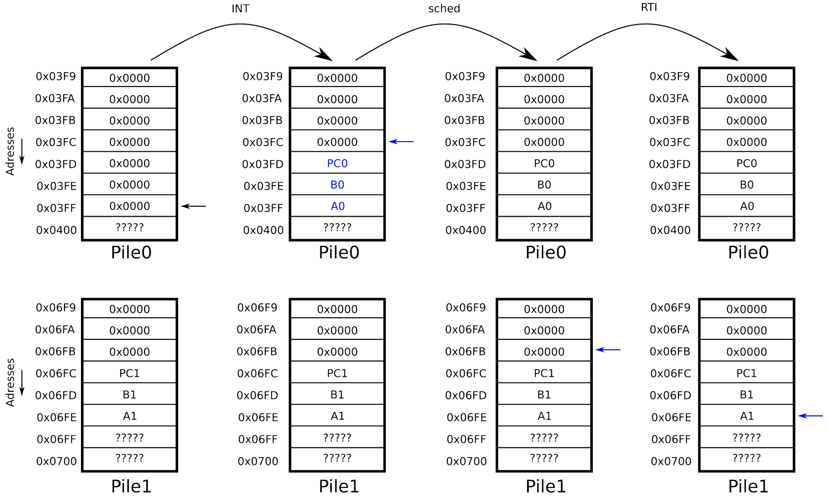
Initialisation







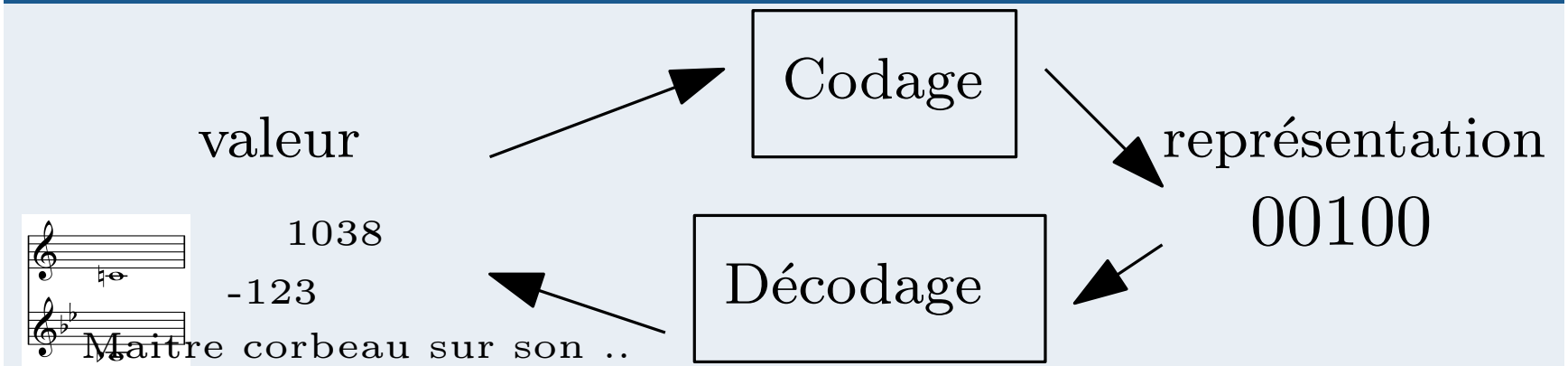
Basculement de contexte



Rétrospective et conclusion

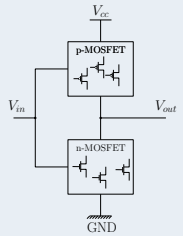
Cours 1 : Codons tout en binaire

Codage/décodage



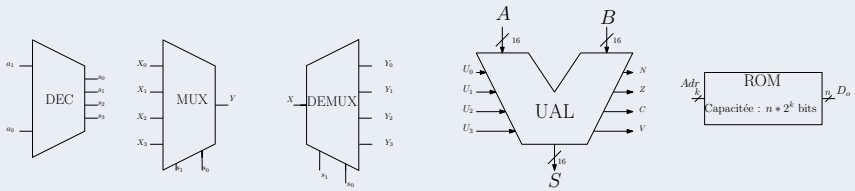
Cours 2-3 : Transistors, Circuits logiques, séquenceur

Couche physique



$\Rightarrow$ Electronique analogique (ELAN), physique des semi-conducteurs

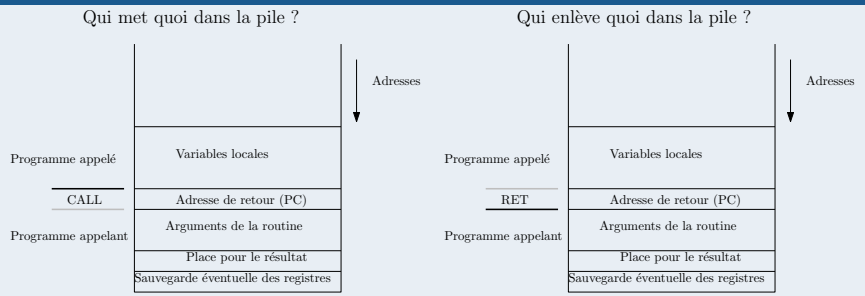
Couche logique



$\Rightarrow$ Systèmes logiques et électronique associée (SLEA)

Cours 4-5 : pile et programmation

Pile et procédures

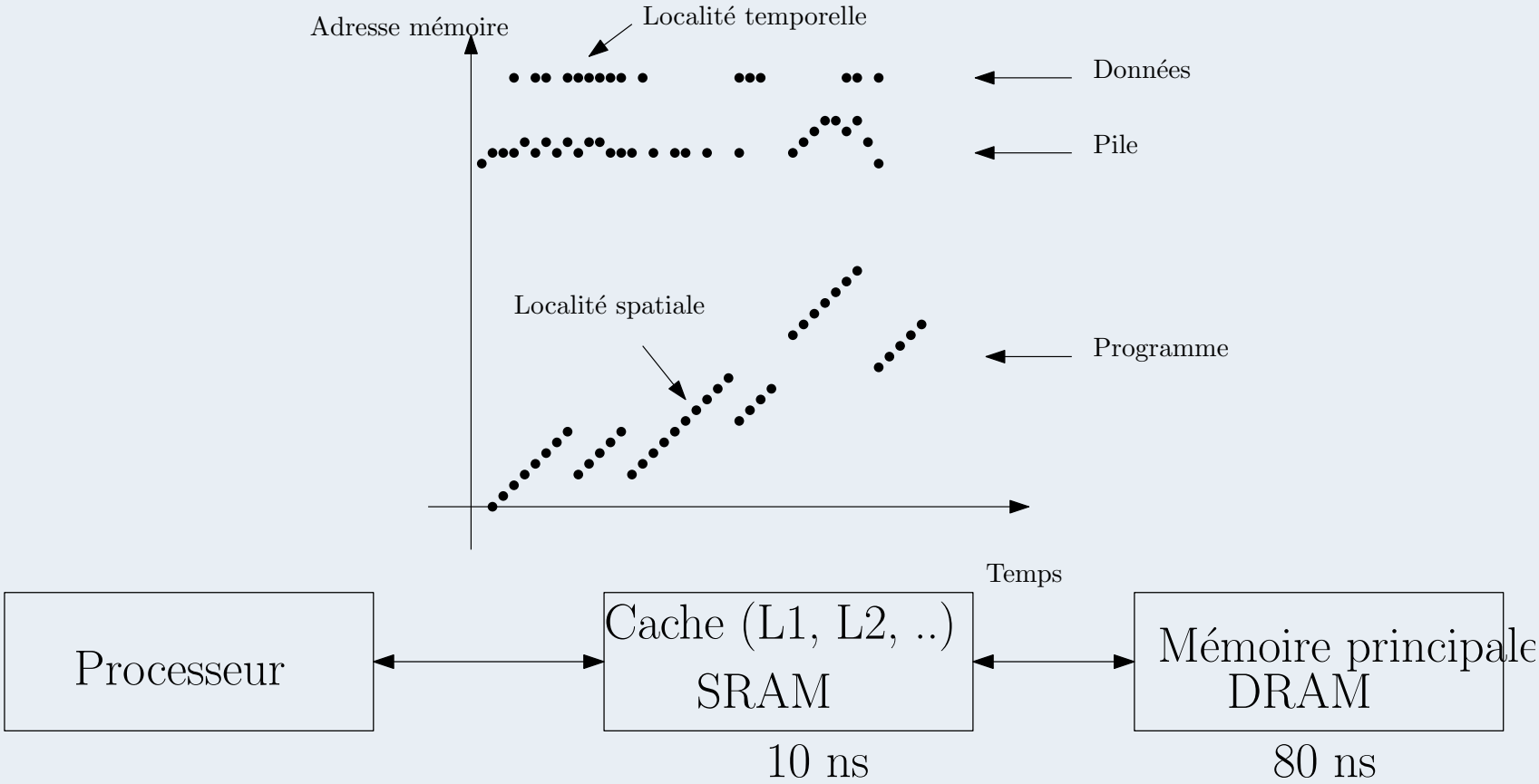


Programmation

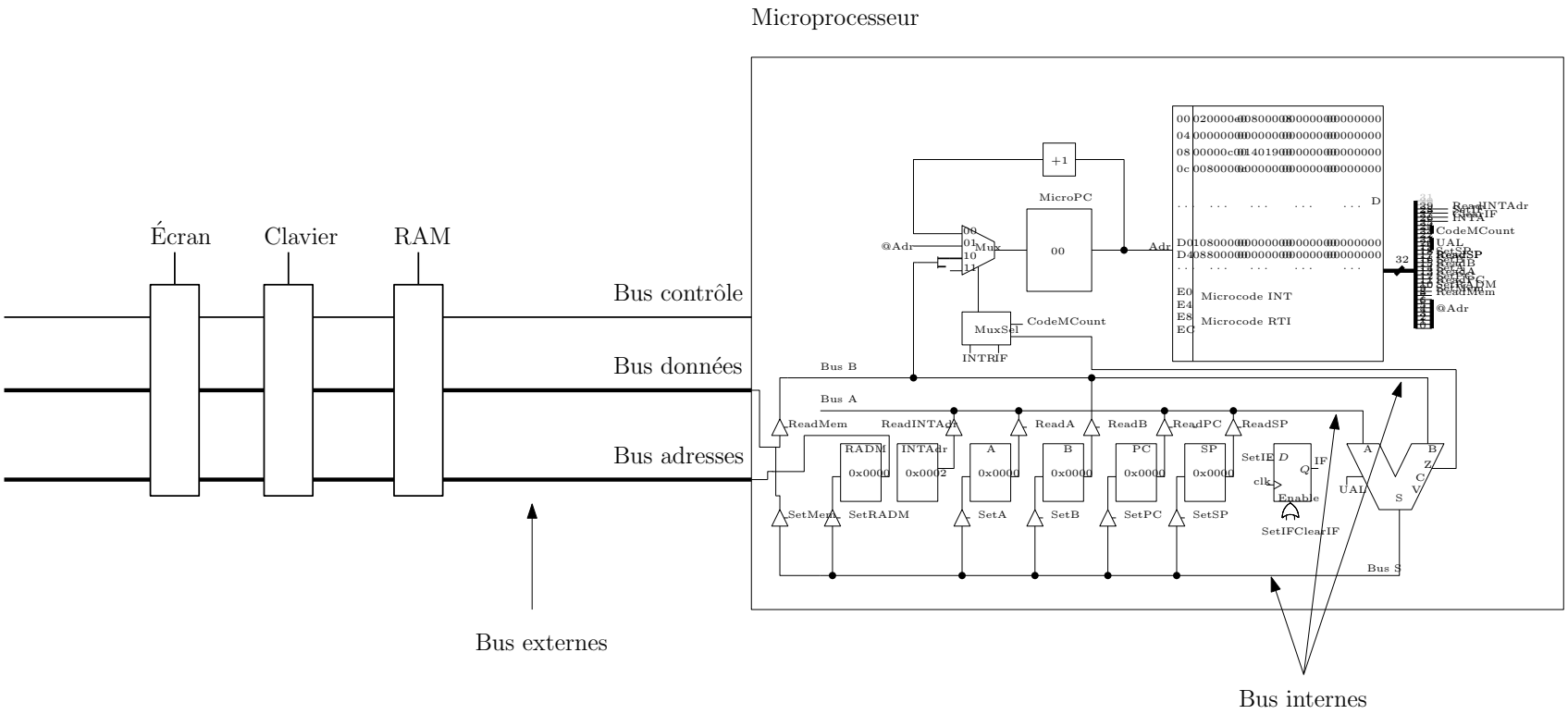
$\Rightarrow$ Fondement de l'informatique et structures de données (FISDA)

Cours 6 : mémoires caches

Hierarchie de mémoire



Cours 7 : périphériques et interruptions



Cours xx

$\Rightarrow$ Systèmes d'informations (SI)

1. Systèmes d'exploitation :
   - couche d'abstraction supplémentaire
   - systèmes de fichiers
   - mémoire virtuelle
   - ordonnanceur
   - ...
2. .....