Conception et systèmes embarqués complexes

Master 2004

Antoine Fraboulet, Tanguy Risset
antoine.fraboulet@insa-lyon.fr, tanguy.risset@ens-lyon.fr
Lab CITI, INSA de Lyon, Lab LIP, ENS de Lyon

Conception de systèmes embarqués complexes

Antoine FRABOULET, Tanguy RISSET
antoine.fraboulet@insa-lyon.fr
tanguy.risset@ens-lyon.fr

Plan

1. Architecture matérielle
2. Organisation logicelle
3. Méthodes de conception
4. Plateforme SocLib

Présentation

- Architecture des systèmes embarqués
  - Processeur
  - Mémoires
  - Système à bus
  - Périphériques
  - Communications

Compilation et interprétation

Langage de haut niveau

```
int a,b,c;
a = b + c;
```

Assembleur

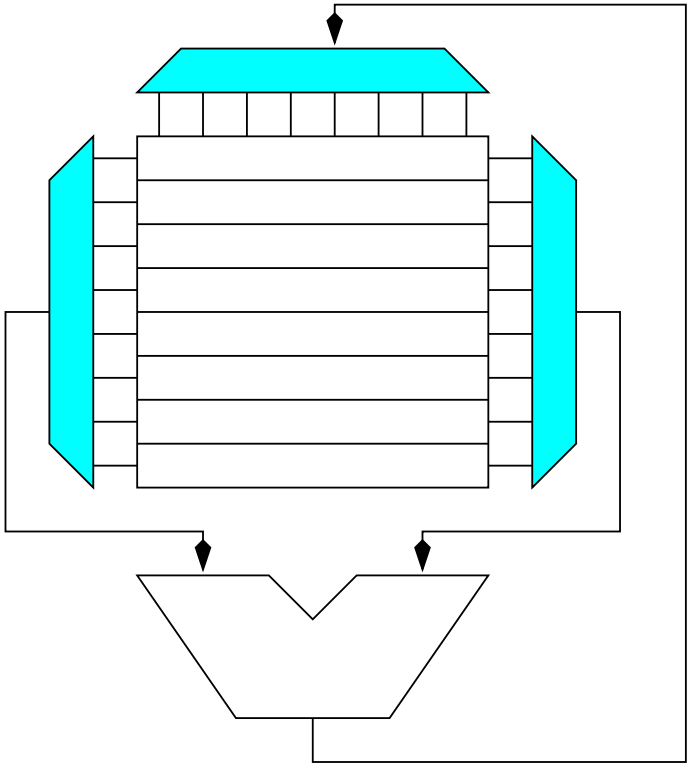
```
int a,b,c;

load R0, @b
load R1, @c
add R3,R0,R1
store R3, @a
```

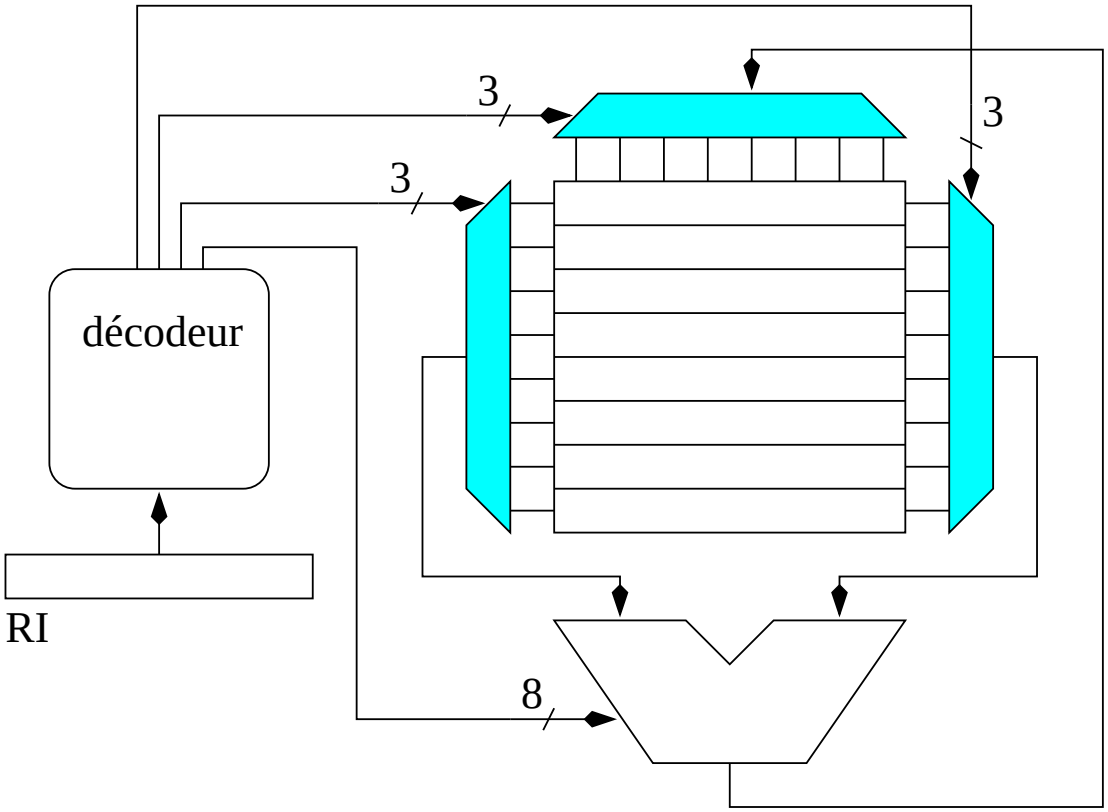
Binaire

```
01001011...10101
01001010...10001
      ...
10010011...00011
```

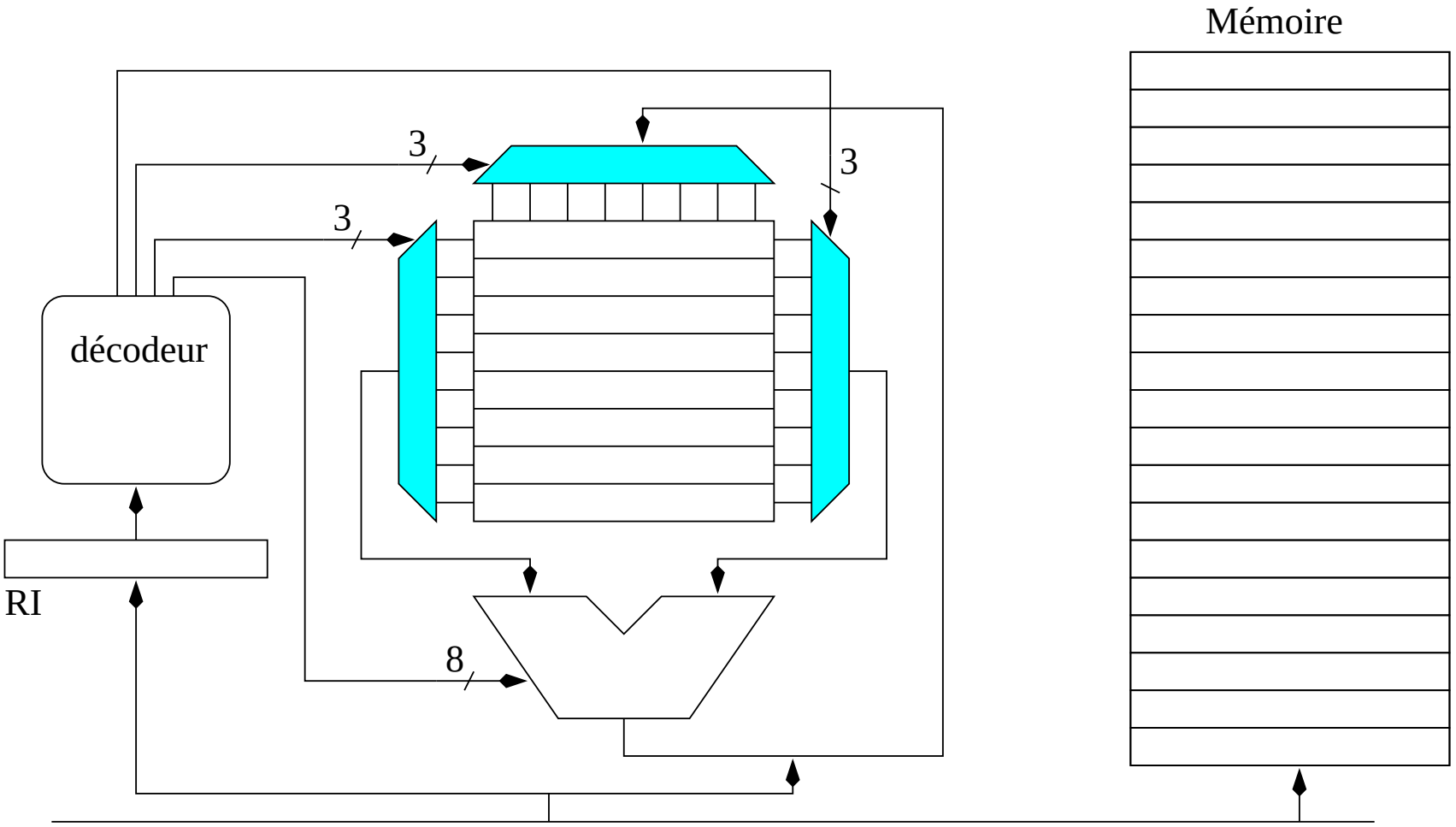
Processeur - Mémoire



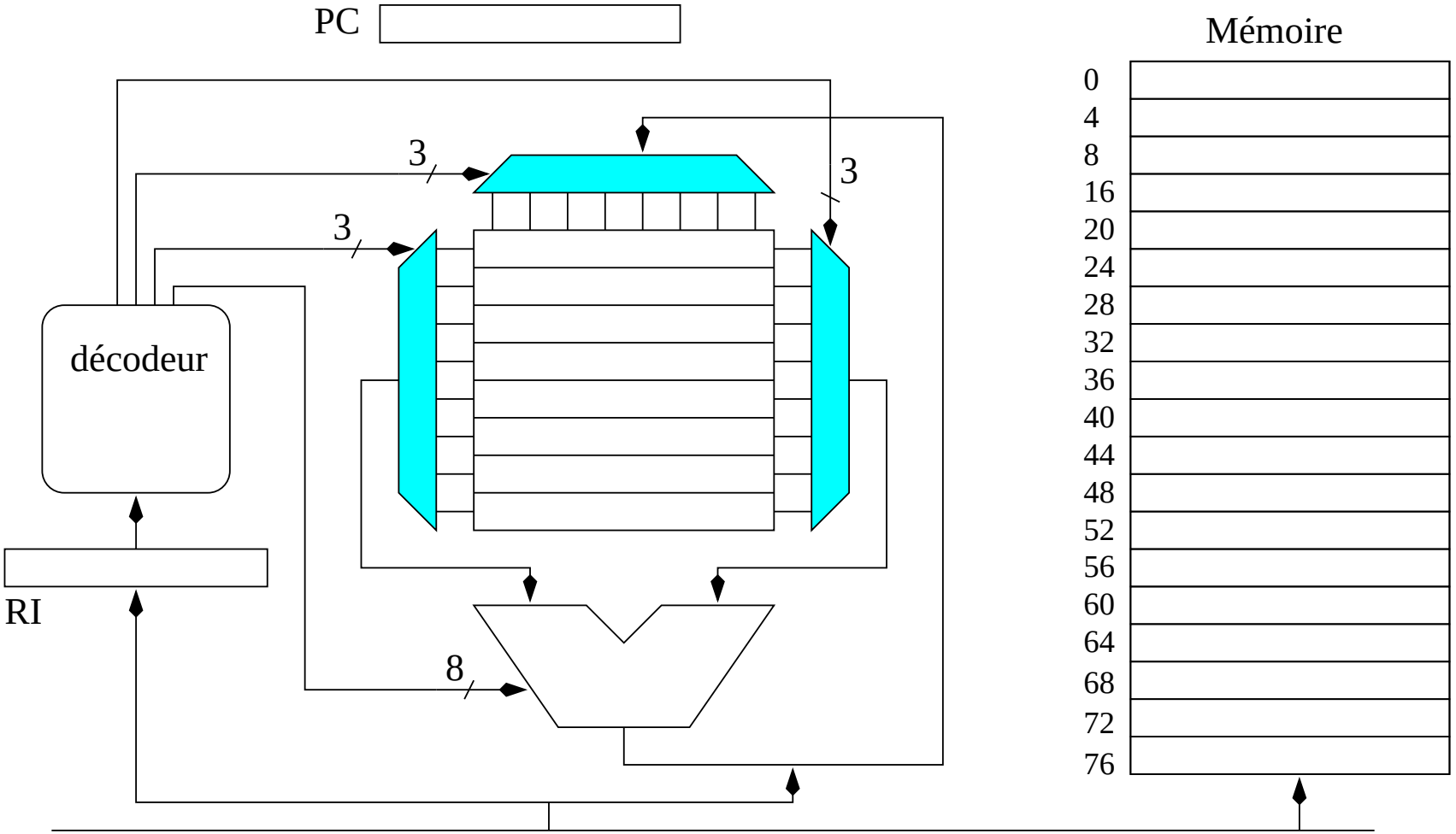
Processeur - Mémoire



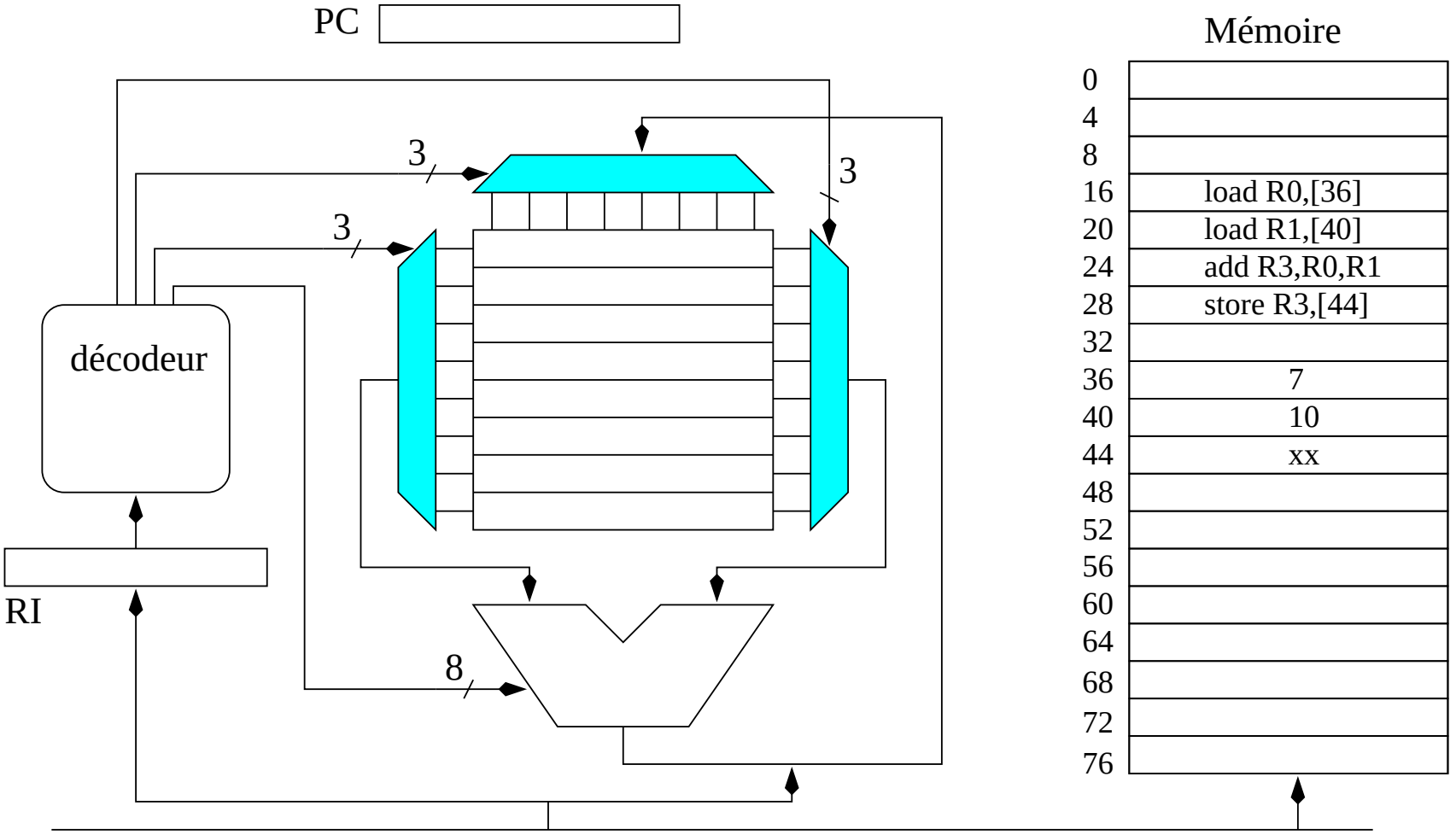
Processeur - Mémoire



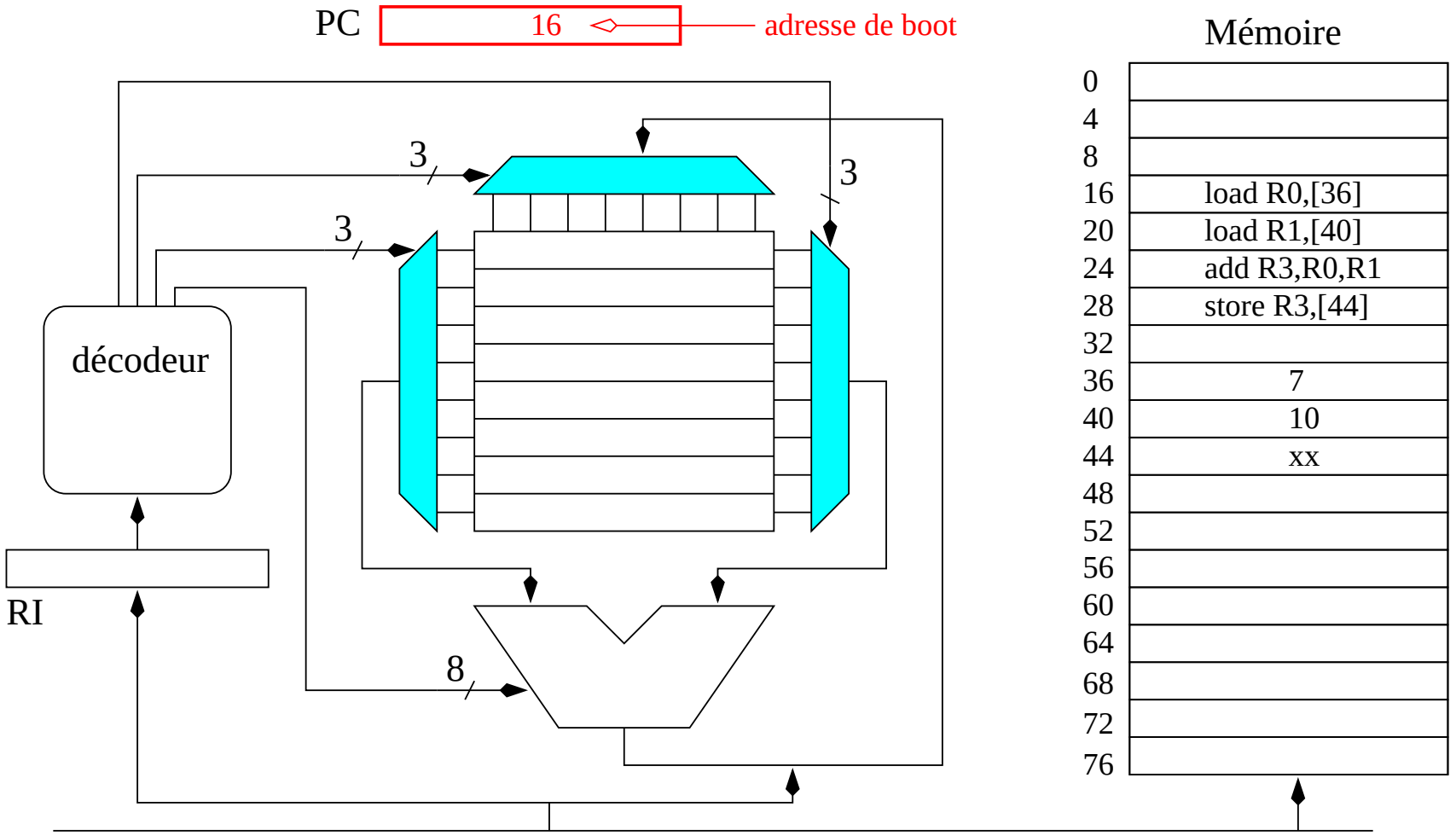
Processeur - Mémoire



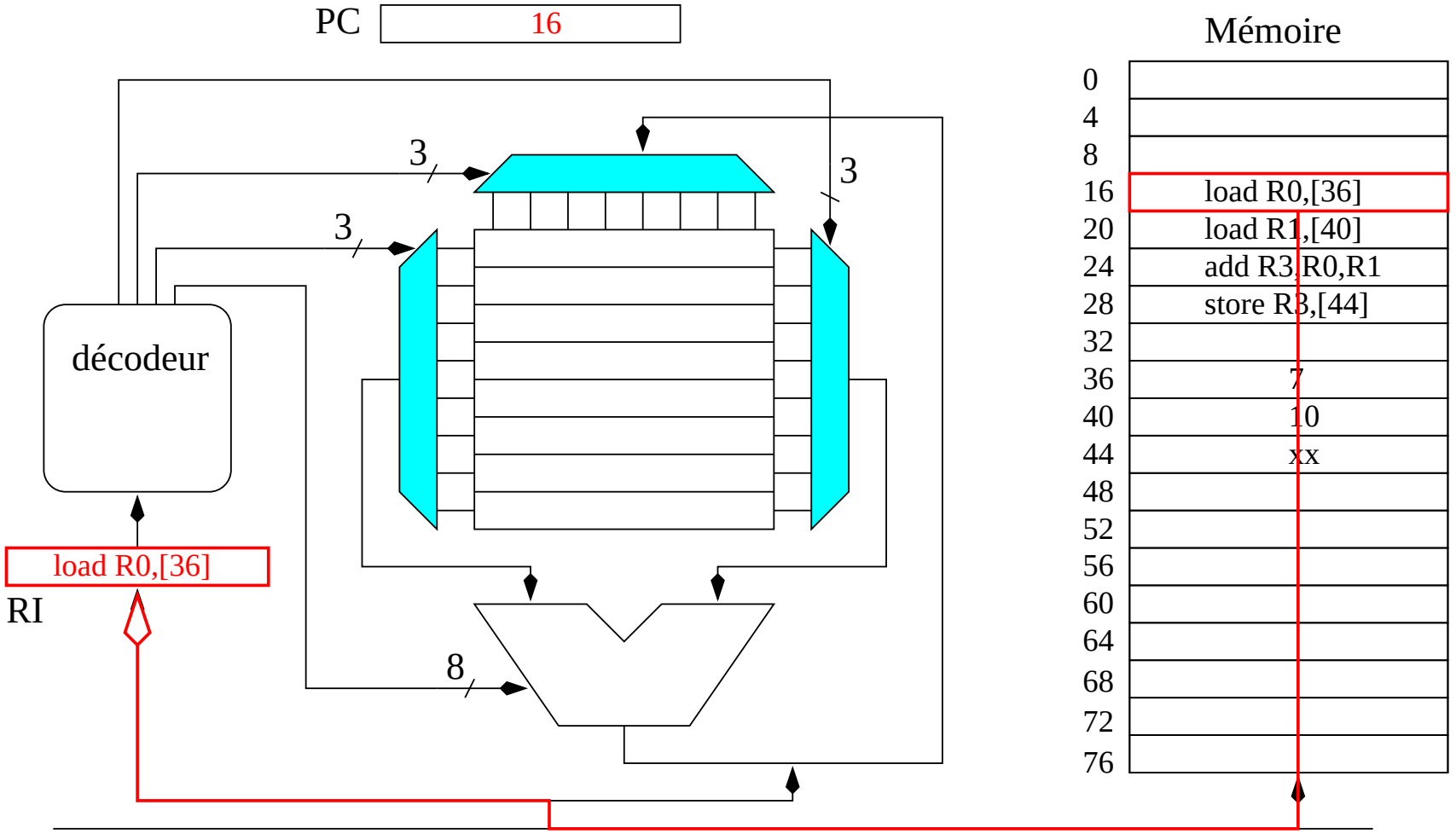
Processeur - Mémoire



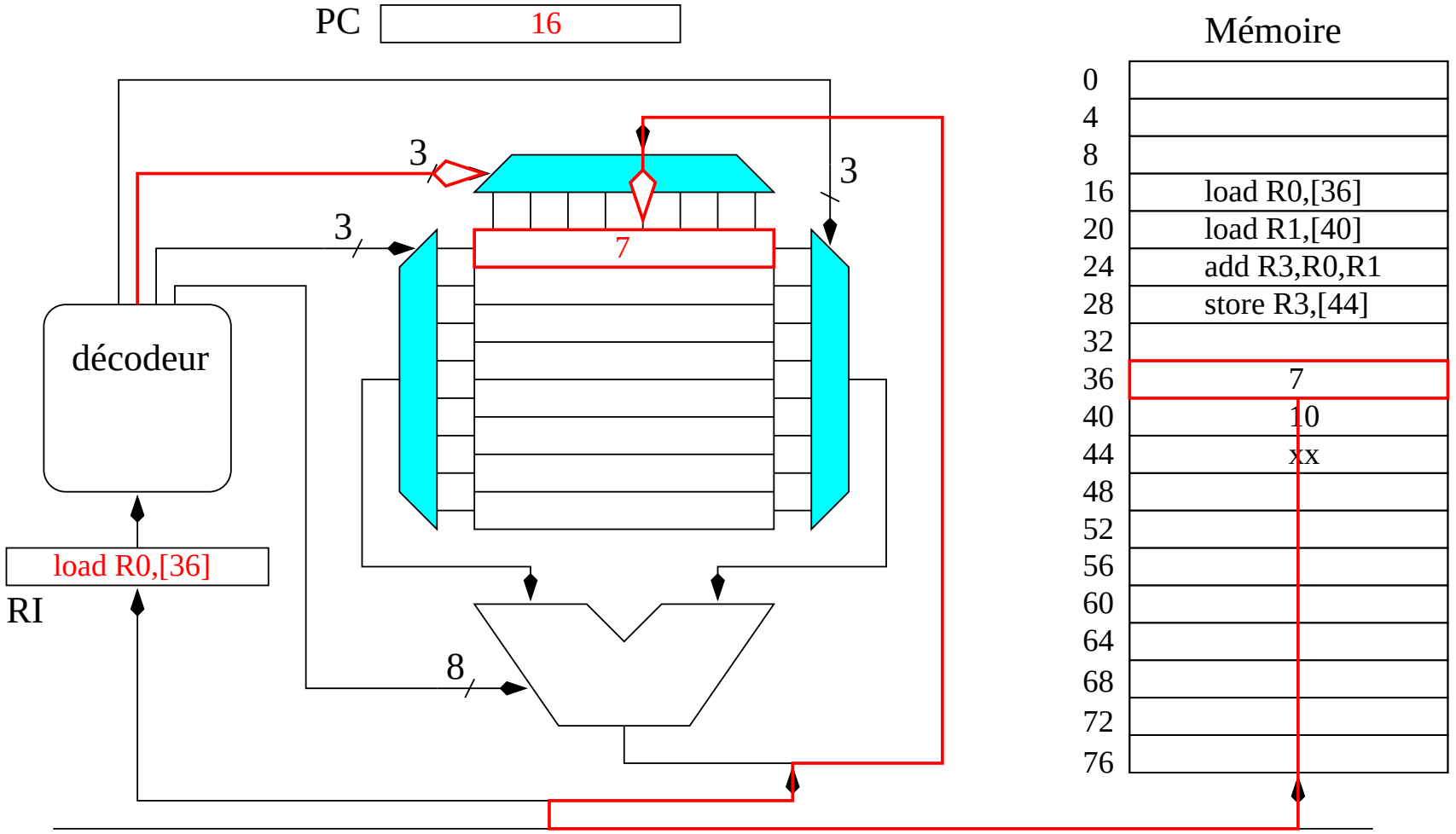
Processeur - Mémoire



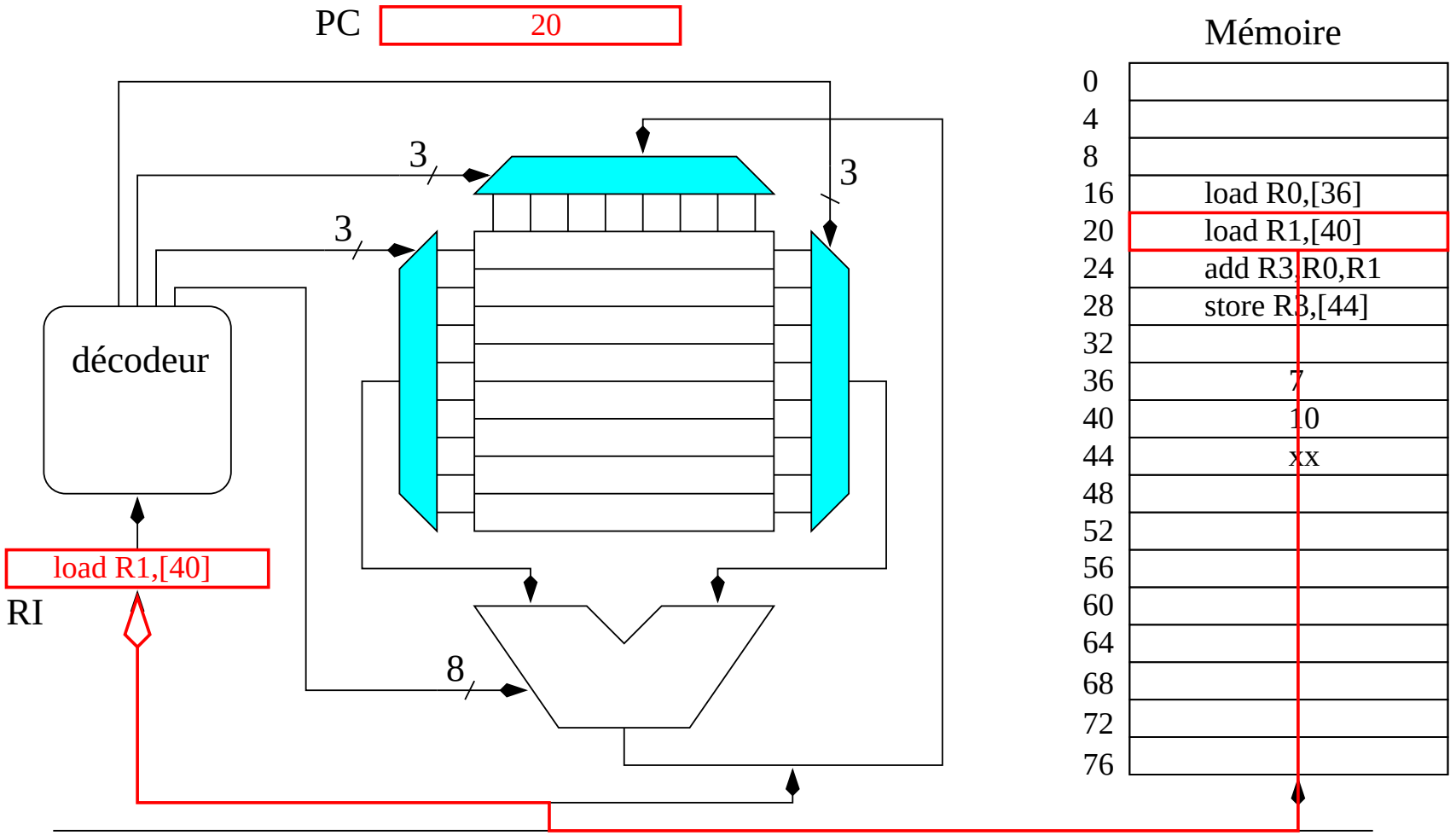
Processeur - Mémoire



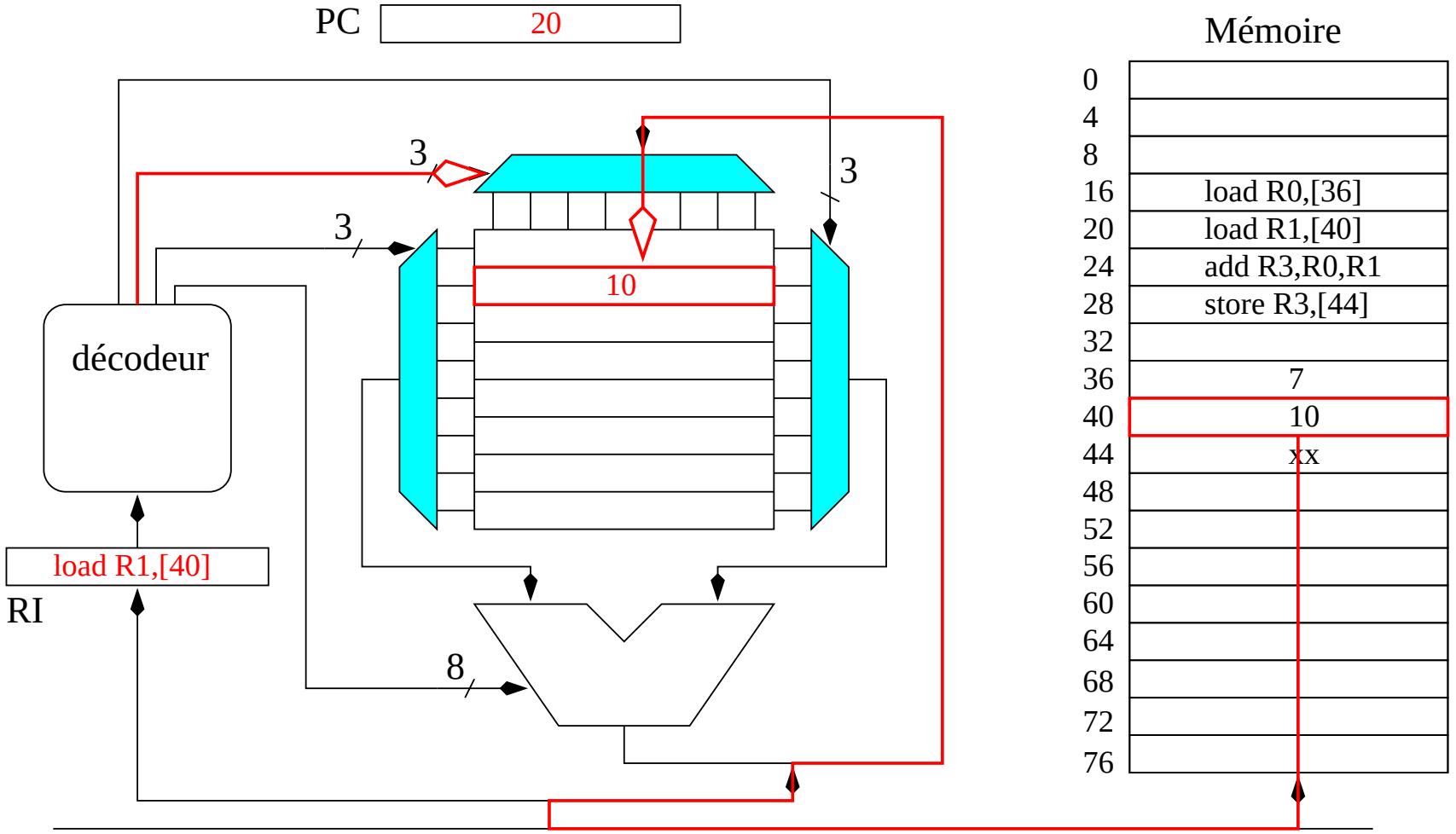
Processeur - Mémoire



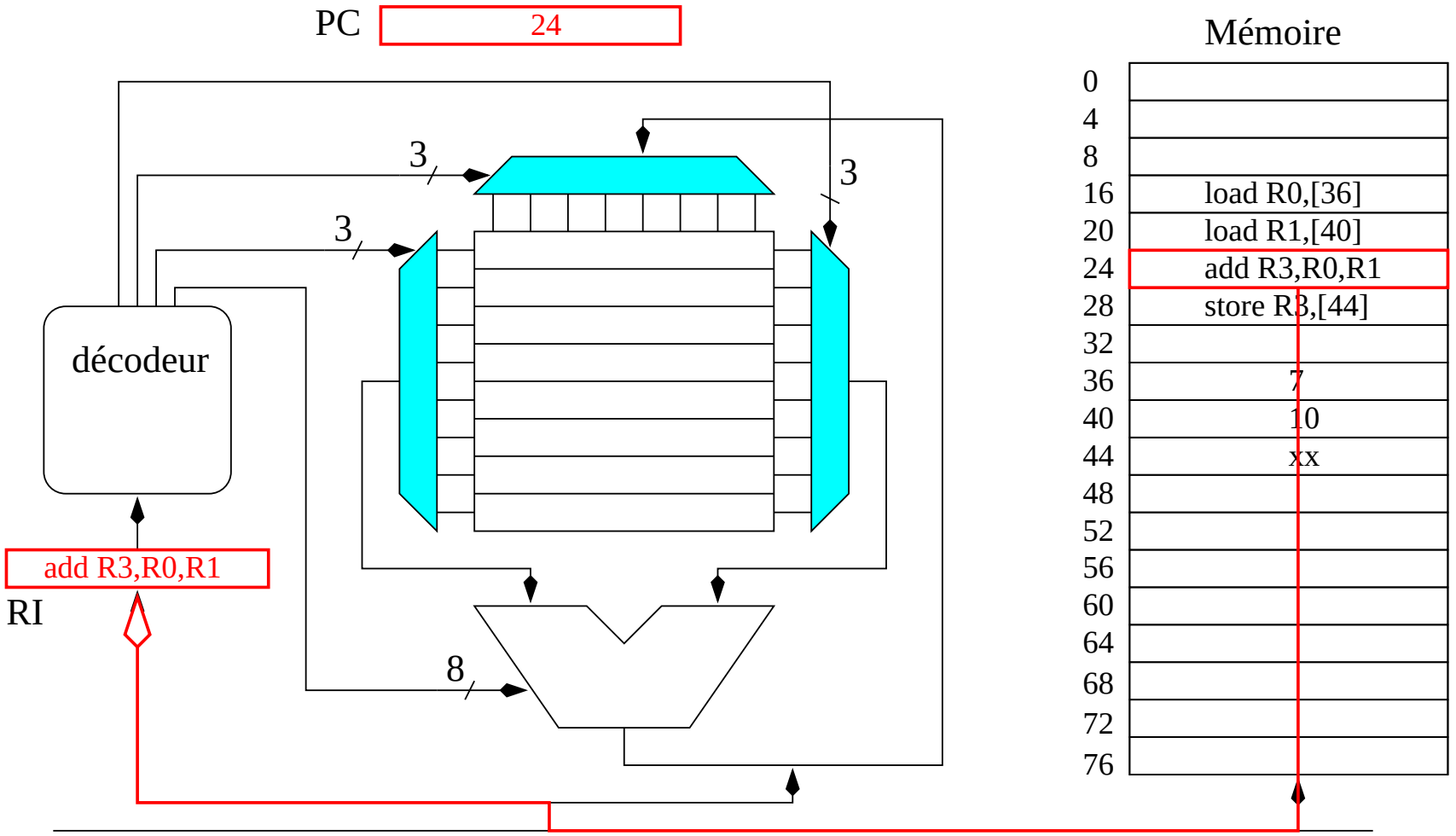
Processeur - Mémoire



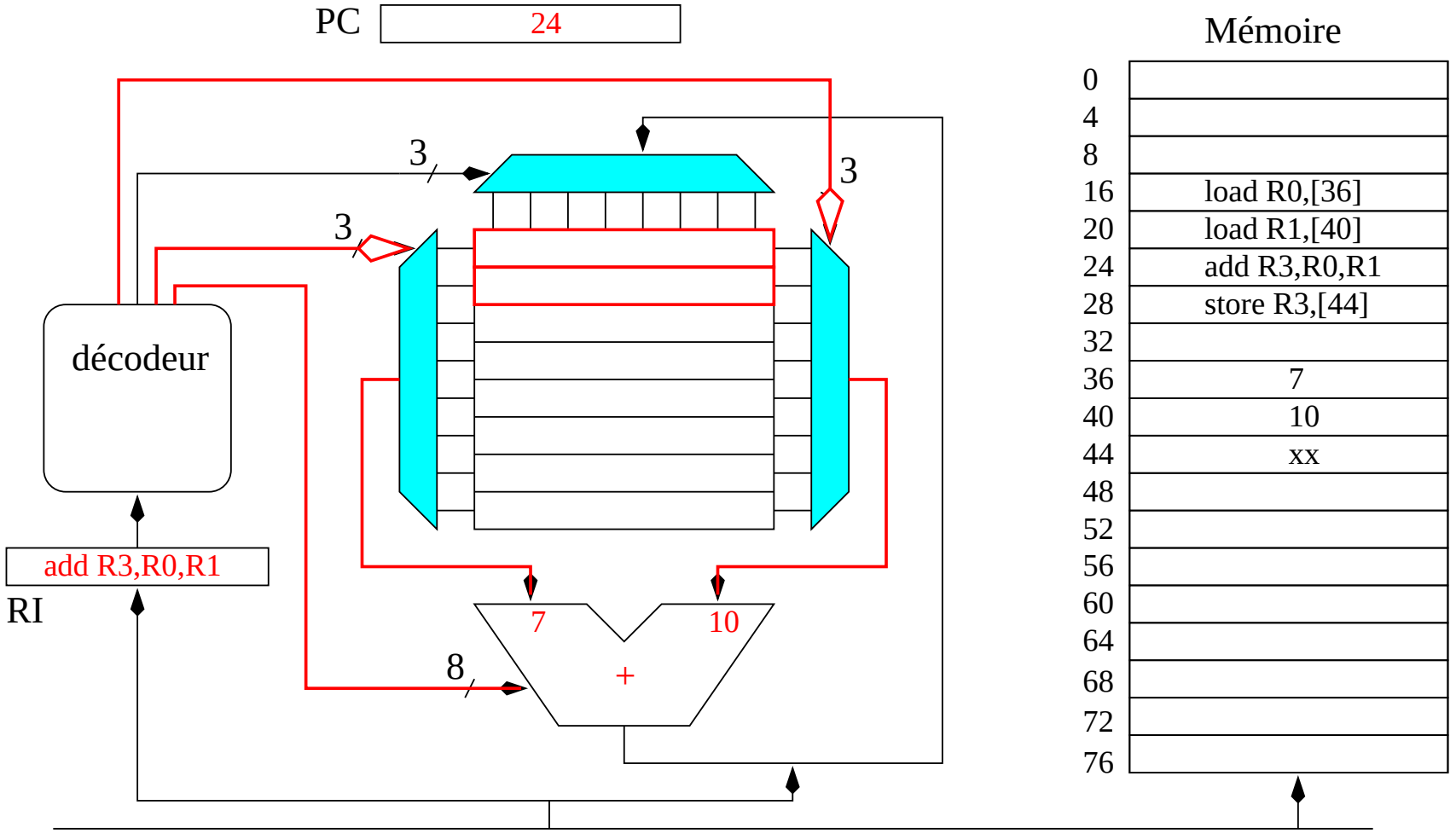
Processeur - Mémoire



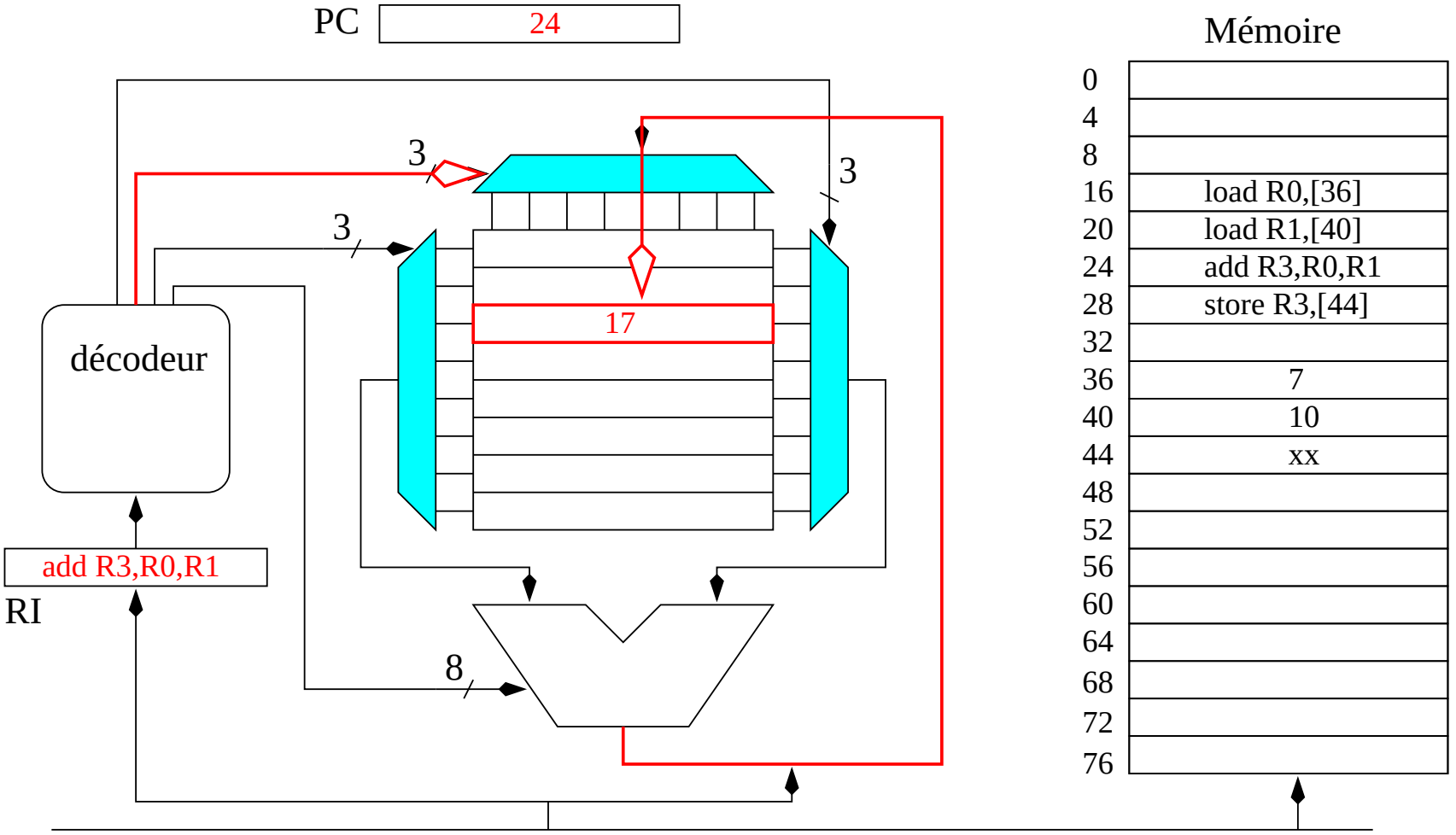
Processeur - Mémoire



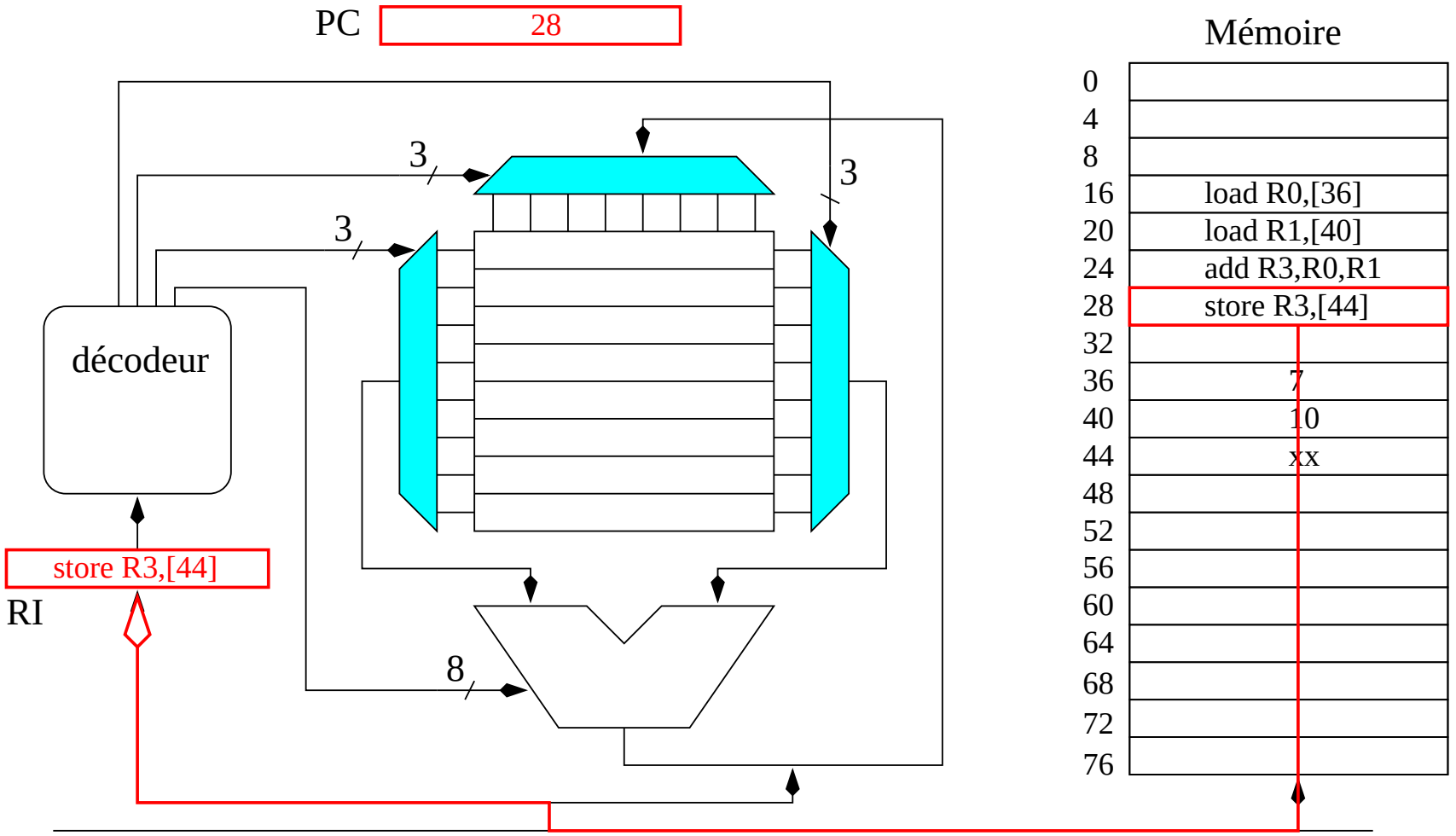
Processeur - Mémoire



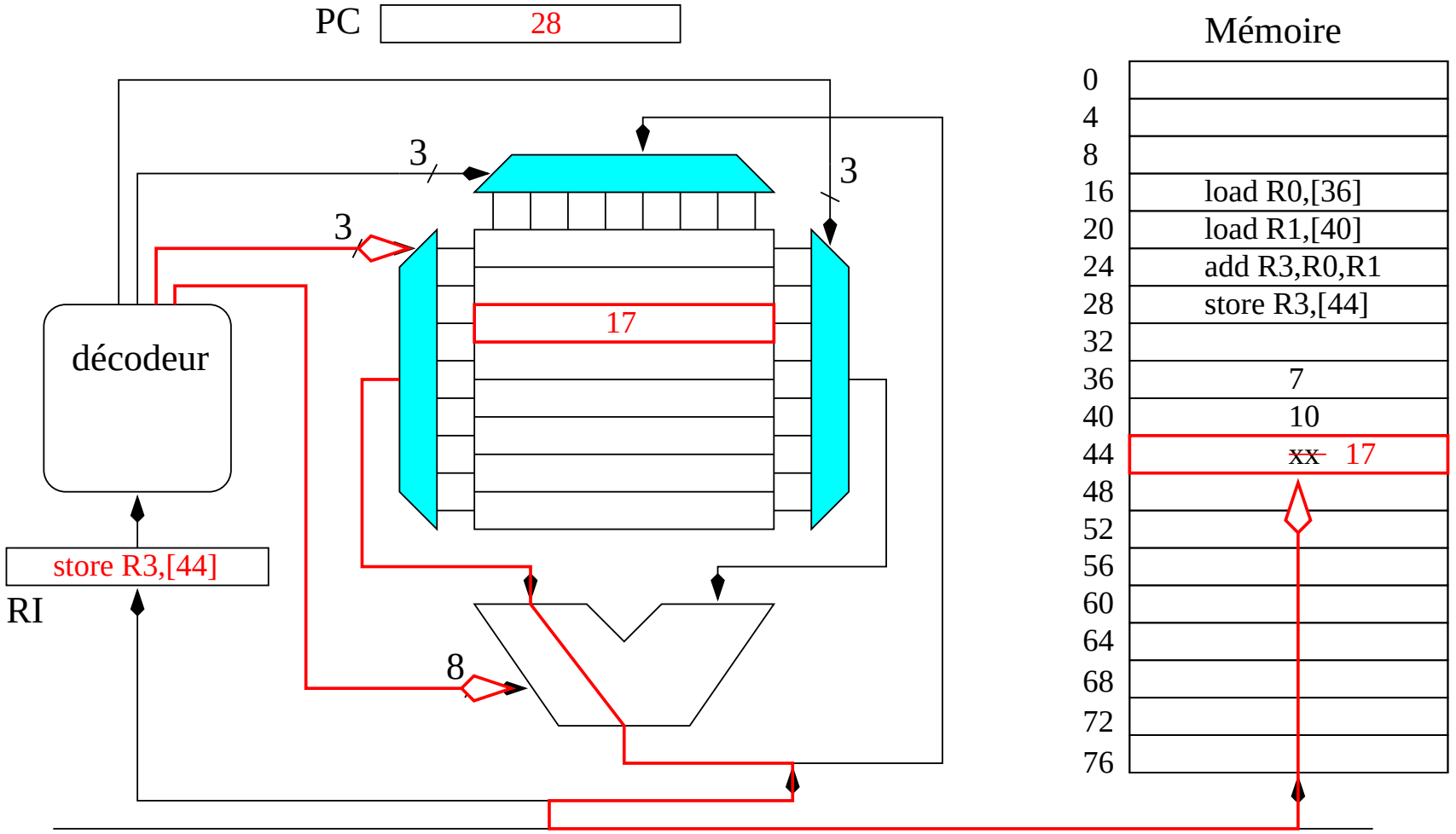
Processeur - Mémoire



Processeur - Mémoire



Processeur - Mémoire

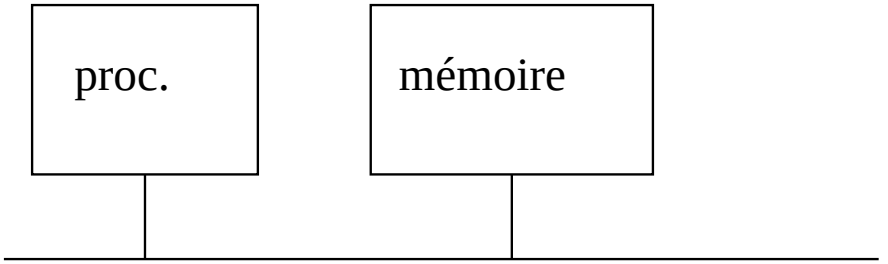


Processeur

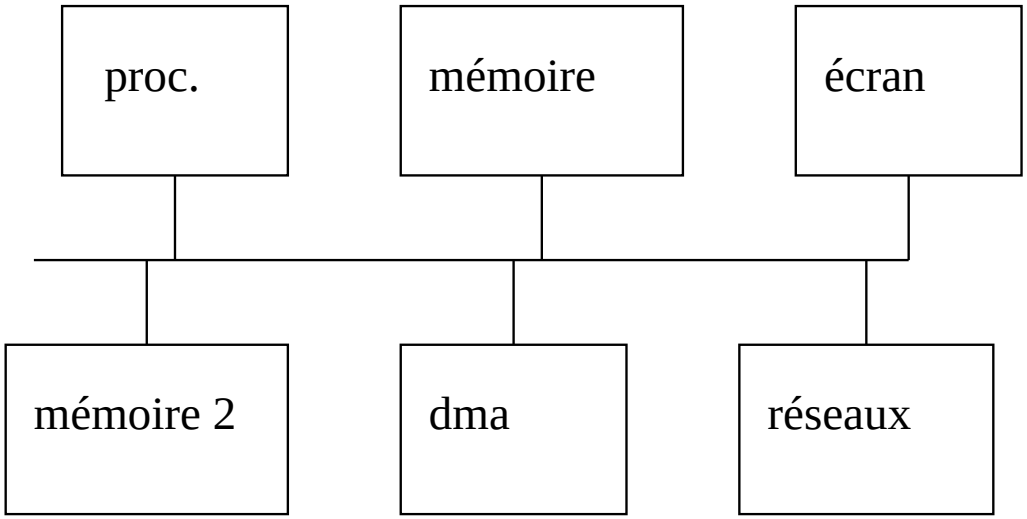
- L'adresse de démarrage (`boot`) d'un processeur est fixée matériellement
- Le processeur accède directement à la mémoire
- La mémoire contient le code et les données
- La suite des instructions est interprétée depuis la mémoire

- Les systèmes embarqués fonctionnent souvent avec des mémoires non volatiles (re-)programmables.
  - mémoire flash
  - eeprom
  - ...
- Cette mémoire peut être reprogrammé en téléchargeant une nouvelle version du logiciel depuis une autre machine.

Interconnexion des Entrées / Sorties



Interconnexion des Entrées / Sorties

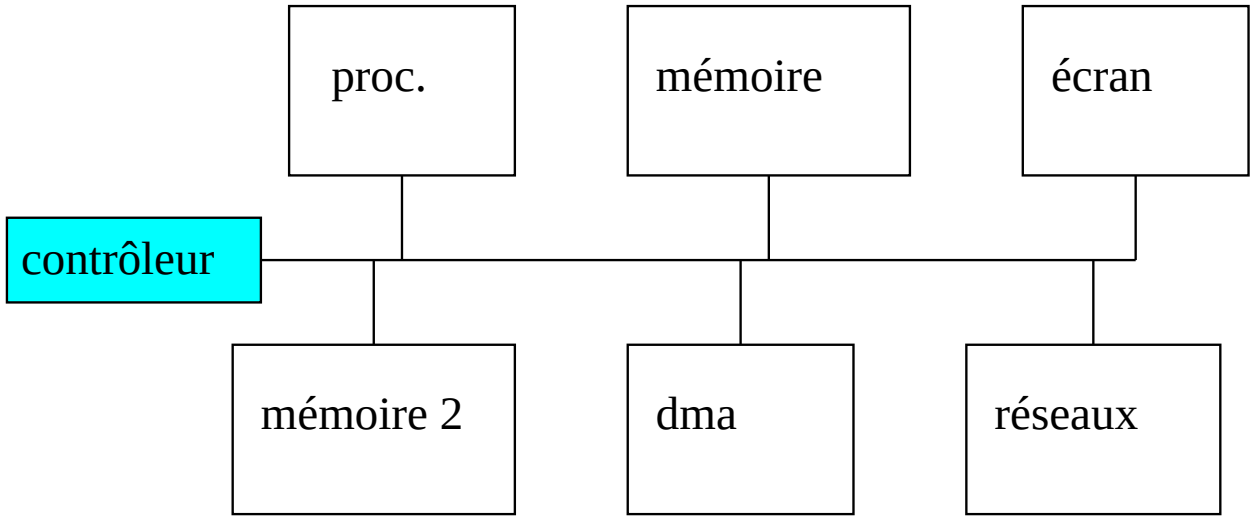


Interconnexion des Entrées / Sorties

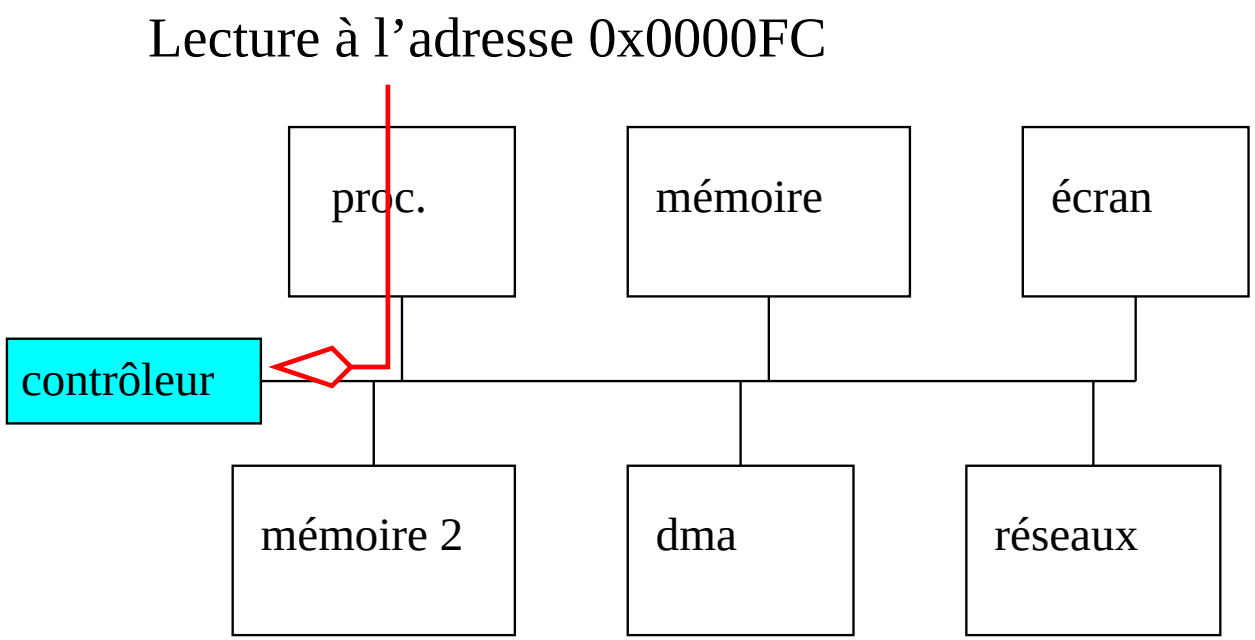
Choix architecturaux:

- Instructions d'E/S
  - Instructions spéciales (`in`, `out`)
  - Espace d'adressage séparé de la mémoire de données
  - Exemple : intel x86
- E/S mappées en mémoire
  - Une tranche d'adresse est allouée à chaque périphérique
  - Utilisation des instructions de lecture, écriture
  - Cas le plus souvent rencontré

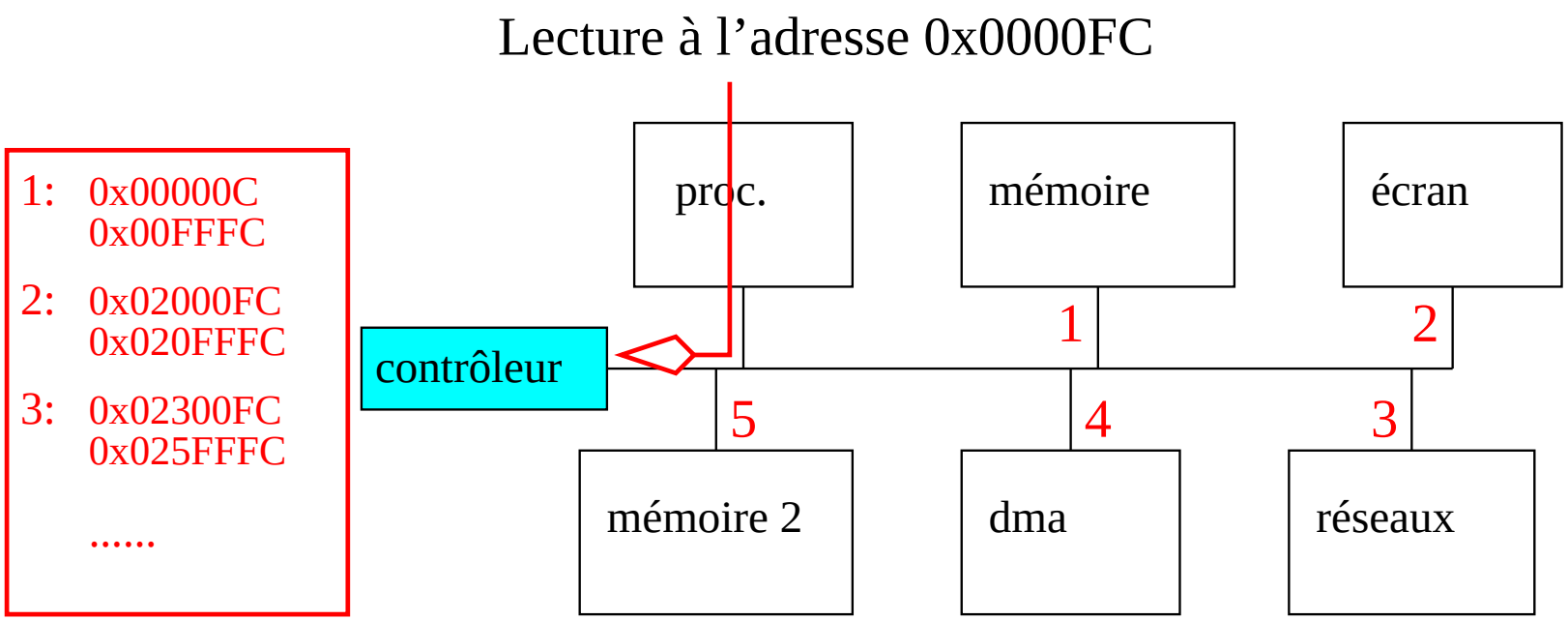
Interconnexion des Entrées / Sorties



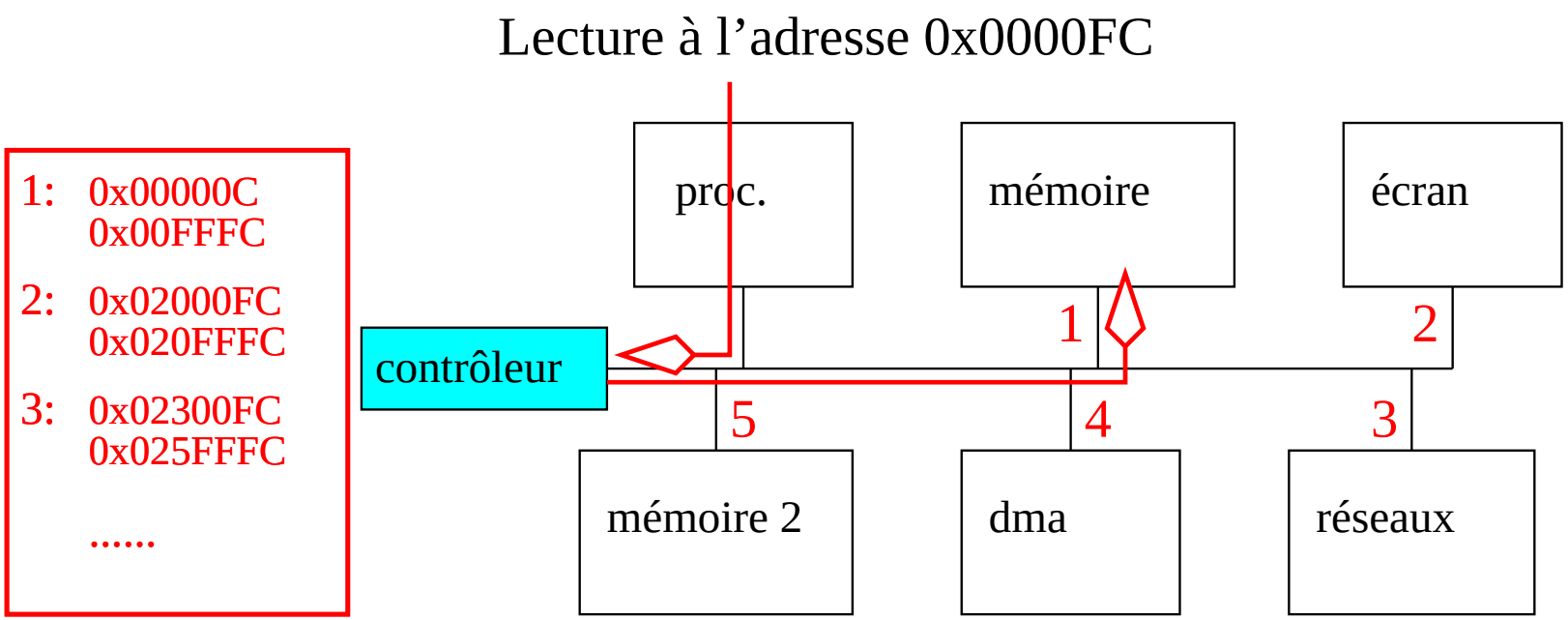
Interconnexion des Entrées / Sorties



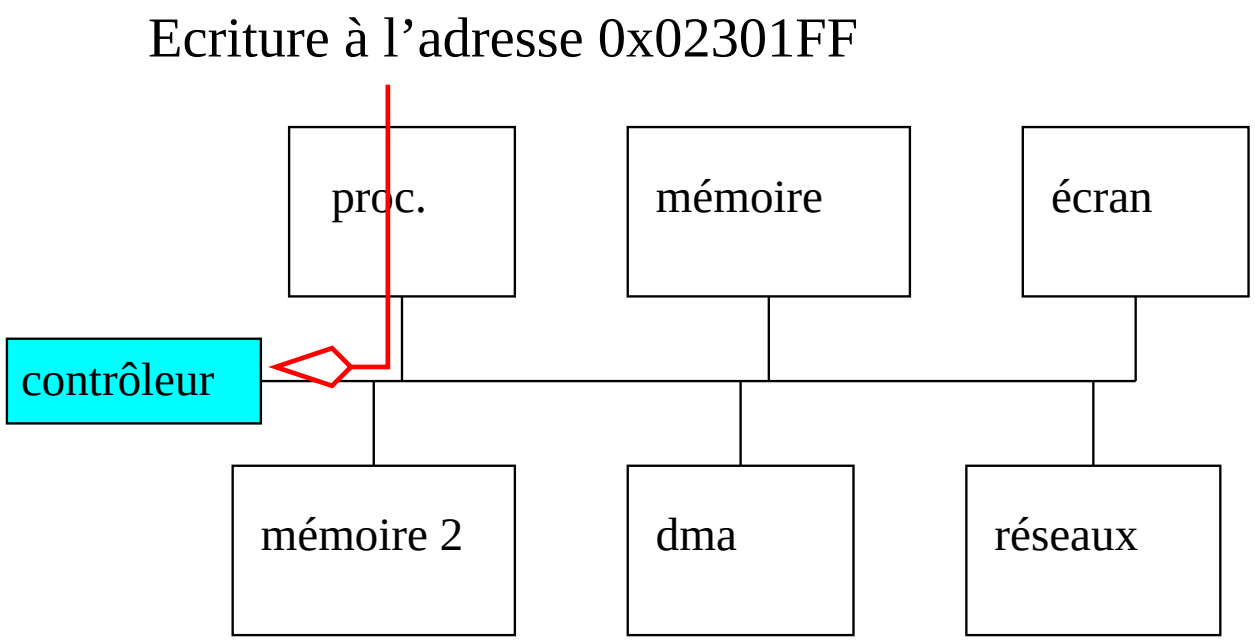
Interconnexion des Entrées / Sorties



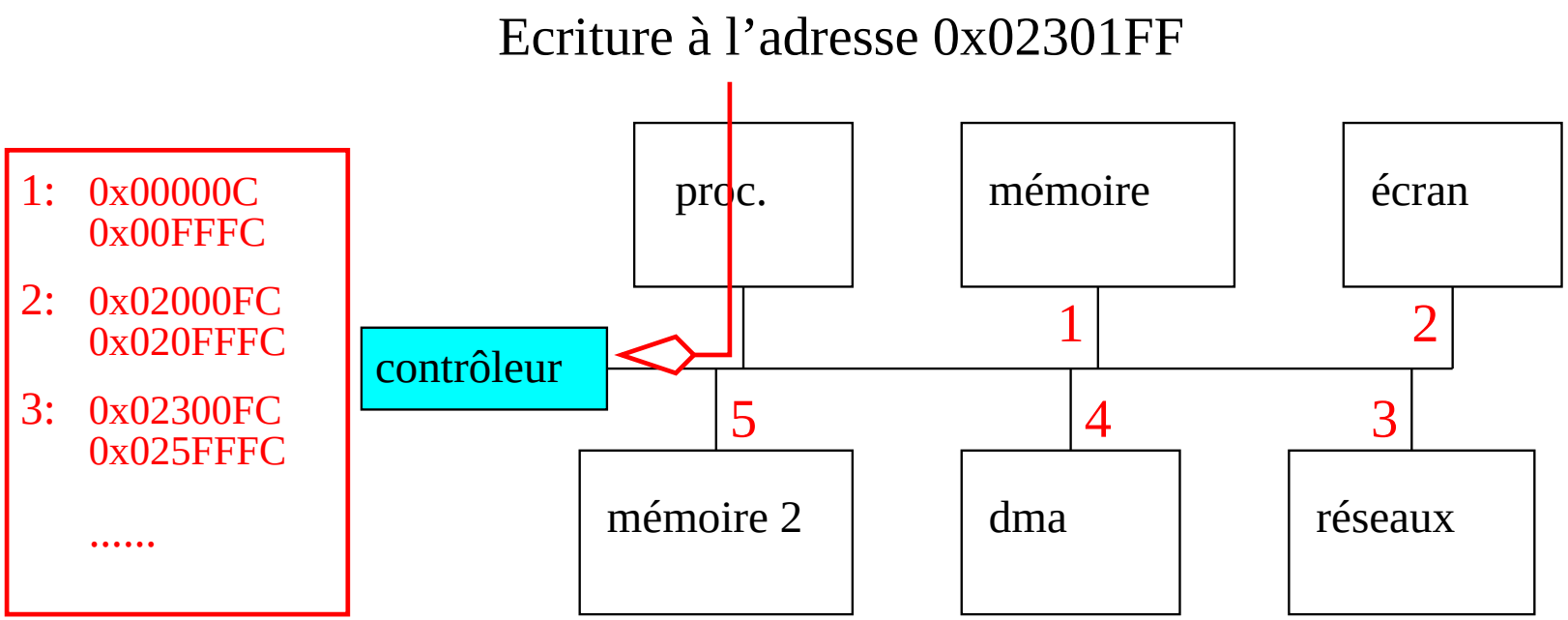
Interconnexion des Entrées / Sorties



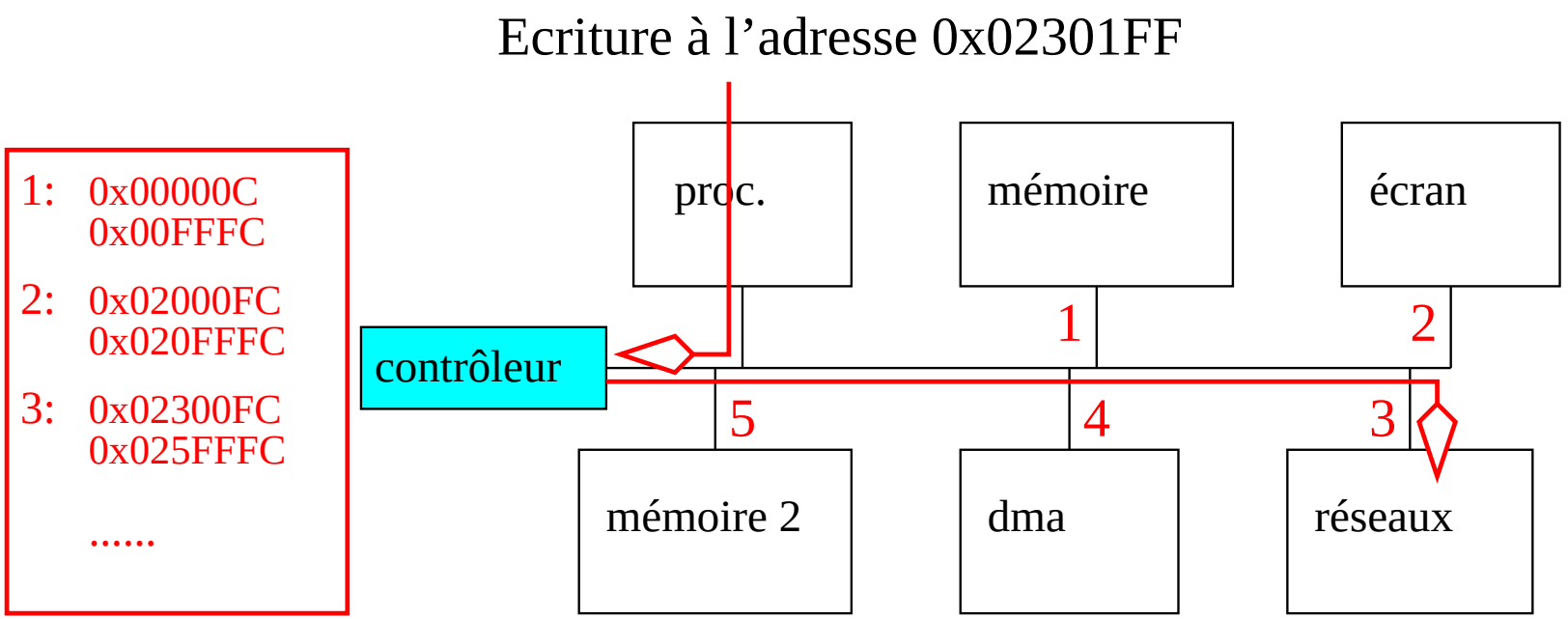
Interconnexion des Entrées / Sorties



Interconnexion des Entrées / Sorties



Interconnexion des Entrées / Sorties

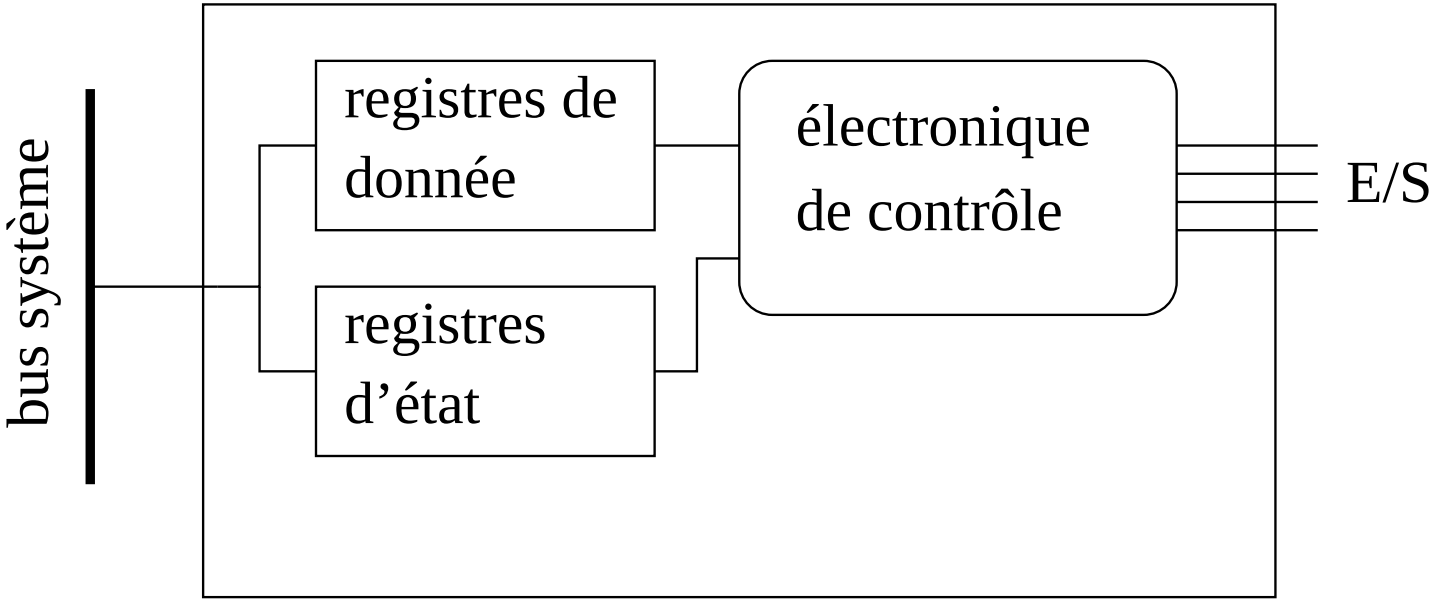


Interconnexion des Entrées / Sorties

Rôle de l'arbitre de bus (contrôleur) :

- Gérer les accès si plusieurs composants veulent écrire simultanément sur le bus
- Gérer la destination des écriture en fonction des adresses
  - Le contrôleur connaît donc la cartographie mémoire (mapping) distribuée entre les composants
  - Cette carte mémoire est définie lors de la conception du système
- Le logiciel doit connaître ces adresses pour accéder aux périphériques et les contrôler

Périphériques



Périphériques

- Un périphérique est un bloc pouvant être manipulé par l'intermédiaire de ses registres de contrôle et de données.
- Il existe différents type de périphériques
  - maîtres : peuvent initier une communication sur le bus
  - esclave : ne peuvent que répondre à une requête de lecture et/ou d'écriture
  - certains composants peuvent avoir les deux interfaces (DMA par exemple)

Écriture de pilotes de périphériques

- Un pilote (driver) est la partie logicielle permettant de faire fonctionner un périphérique
- Les pilotes sont spécifiques aux périphériques
  - Adresses mémoires (décalage par rapport à une base) : ports de communications et registres
  - Connaissances des mécanismes de communications : automates de fonctionnement
- Les pilotes sont également liés au système d'exploitation

Écriture de pilotes de périphériques

Les communications avec les périphériques nécessitent un accès aux adresses mémoires réelles.

```
#define ALPHA_IN        0x40000000
#define ALPHA_OUT       0x41000000

static inline void write(int v)
{
  *ALPHA_IN = v;
}

static inline int read()
{
  return *ALPHA_OUT;
}
```

Scrutation, attente active

- le processeur est en boucle de lecture sur un registre d'état
- la valeur de ce registre indique si le périphérique peut accepter une nouvelle lecture ou écriture

```
#define OUT_DATA   0x1000
#define STATUS_REG 0x1001

char s[]="hello.";
char *ptr = s;

while (*ptr != 0) {
  write(OUT_DATA,*ptr);
  while (read(STATUS_REG) != 0);
  ptr++;
}
```

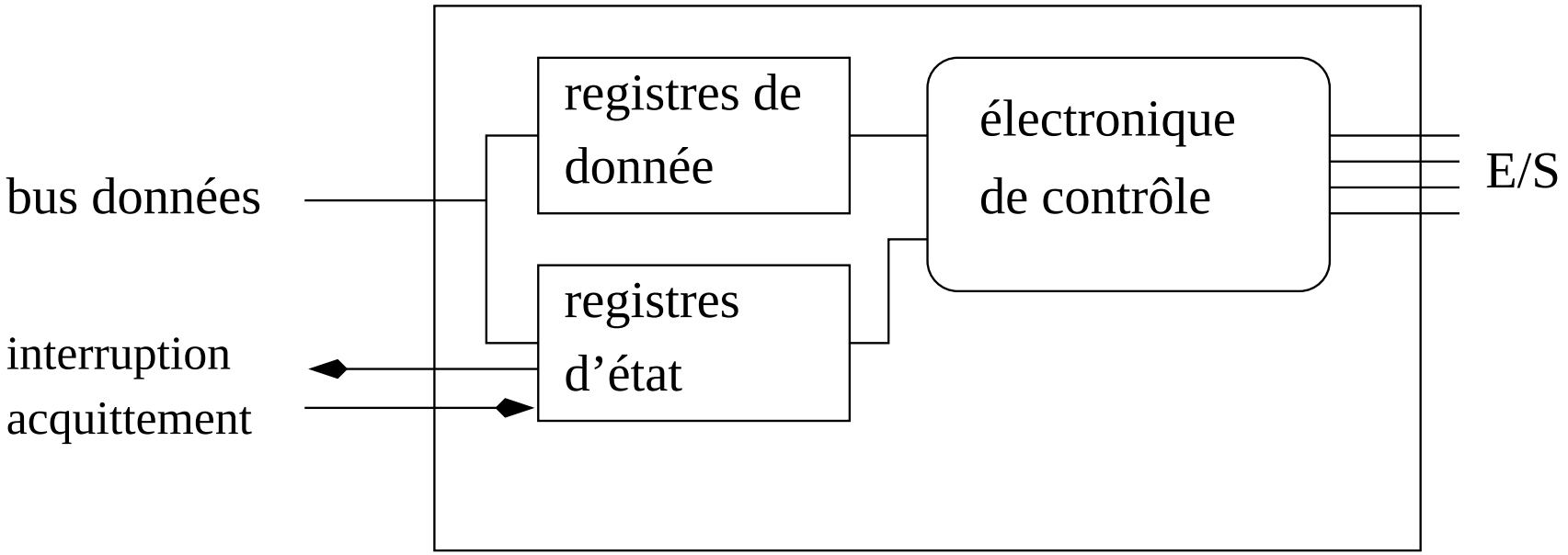
Communication avec les périphériques

- Les communications par scrutation sont simples à programmer
- Le débit de transfert des E/S est limité par la vitesse du processeur
- La latence de traitement dépend de la période de scrutation du périphérique
- Le processeur prend en charge tout le transfert

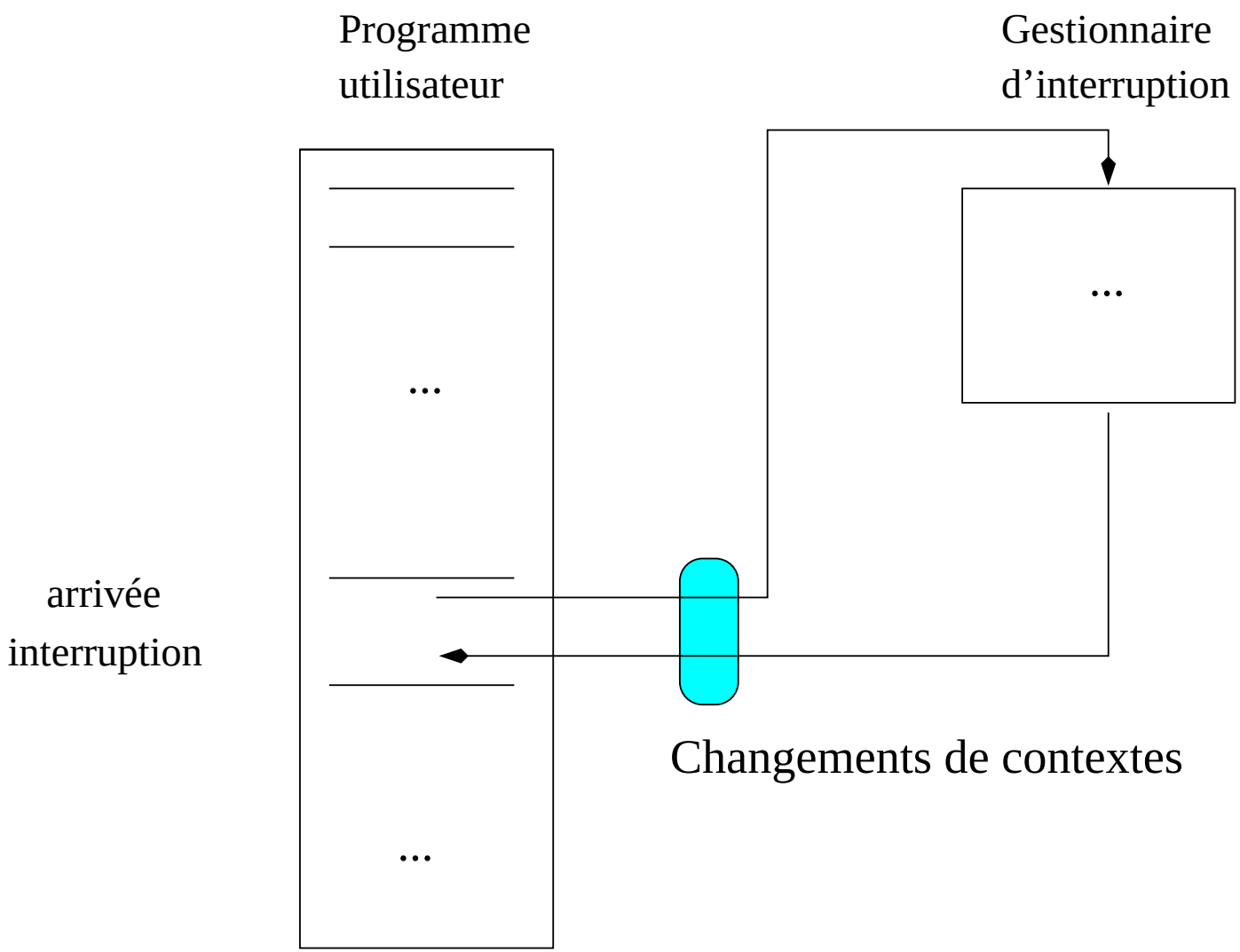
Mécanisme d'interruptions

- Le périphérique peut signaler au processeur qu'il a fini une action
- Nécessite un cablâge supplémentaire pour la signalisation
- Une interruption peut arriver à n'importe quel moment dans le déroulement du programme s'exécutant sur le processeur
- Changements de contextes
  1. Terminaison de l'instruction en cours
  2. **Sauvegarde** de l'état du processeur
  3. Exécution du gestionnaire d'interruption
  4. **Restauration** de l'état précédent
  5. Reprise du fonctionnement normal

Communication avec les périphériques



Communication avec les périphériques



Changement de contexte

La gestion des interruptions nécessite un **support** du processeur.

- Le processeur empile son registre PC et son registre d'état dans une pile d'exécution à l'adresse `[base]` et `[base+4]`
- Le registre CP est modifié pour pointer sur une adresse prédéfinie : le **gestionnaire d'interruption**
- Le code ainsi appelé commence par sauvegarder les registres du processeur dans la mémoire

```
store r0, [base+8]
store r1, [base+12]
...
store r31, [base+132]
```

Changement de contexte

- Le gestionnaire d'interruption peut alors appeler la fonction prévue pour traiter l'interruption.
- Lorsque le traitement est terminé, on restaure le contenu des registres sauvegardés dans la pile
- La dernière action à effectuer est la restauration du registre d'état suivie d'un saut à l'adresse `[base]`.

Communication avec les périphériques

- Les changements de contextes ont un coût non négligeable en nombre de cycles. Il est parfois plus avantageux, pour les petits transferts de données, de faire de la scrutation.

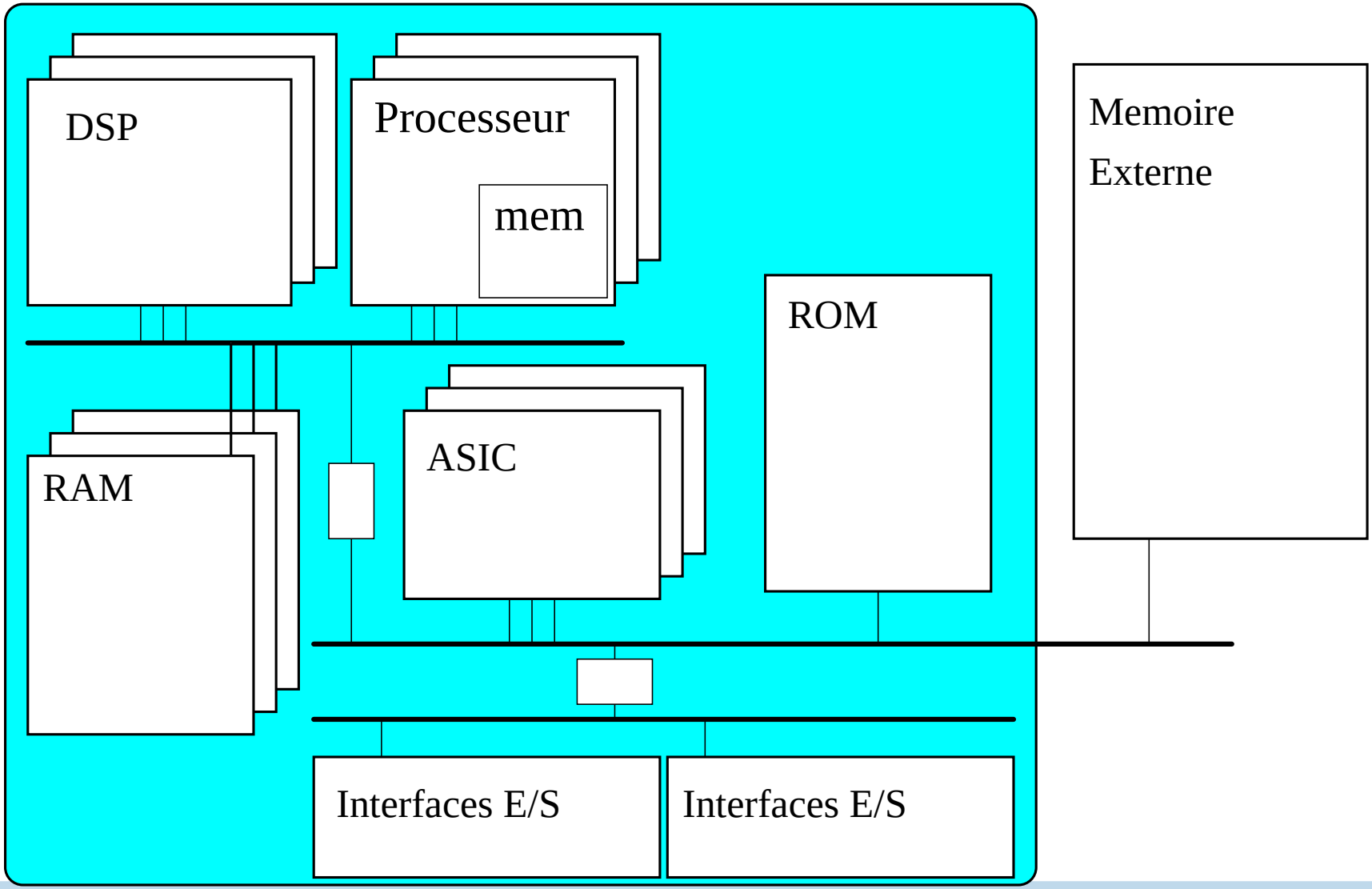
- Les interruptions permettent de libérer le processeurs en attendant qu'un périphérique ne se signale
- Il faut une ligne d'interruption par périphérique
- Ces lignes sont connectées sur un **contrôleur d'interruption**
  - mise en place de priorités
  - vectorisation des interruptions : le processeur demande au contrôleur quelle est le périphérique qui a généré l'interruption

Communications efficaces : DMA

Pour les transfers de gros volumes, il faut de préférence utiliser un DMA (*Direct Memory Access*).

1. Le processeur configure le DMA (adresses et taille de transfert)
2. Le DMA prend la main sur le bus et déplace les données
3. Une interruption est générée à la fin du transfert

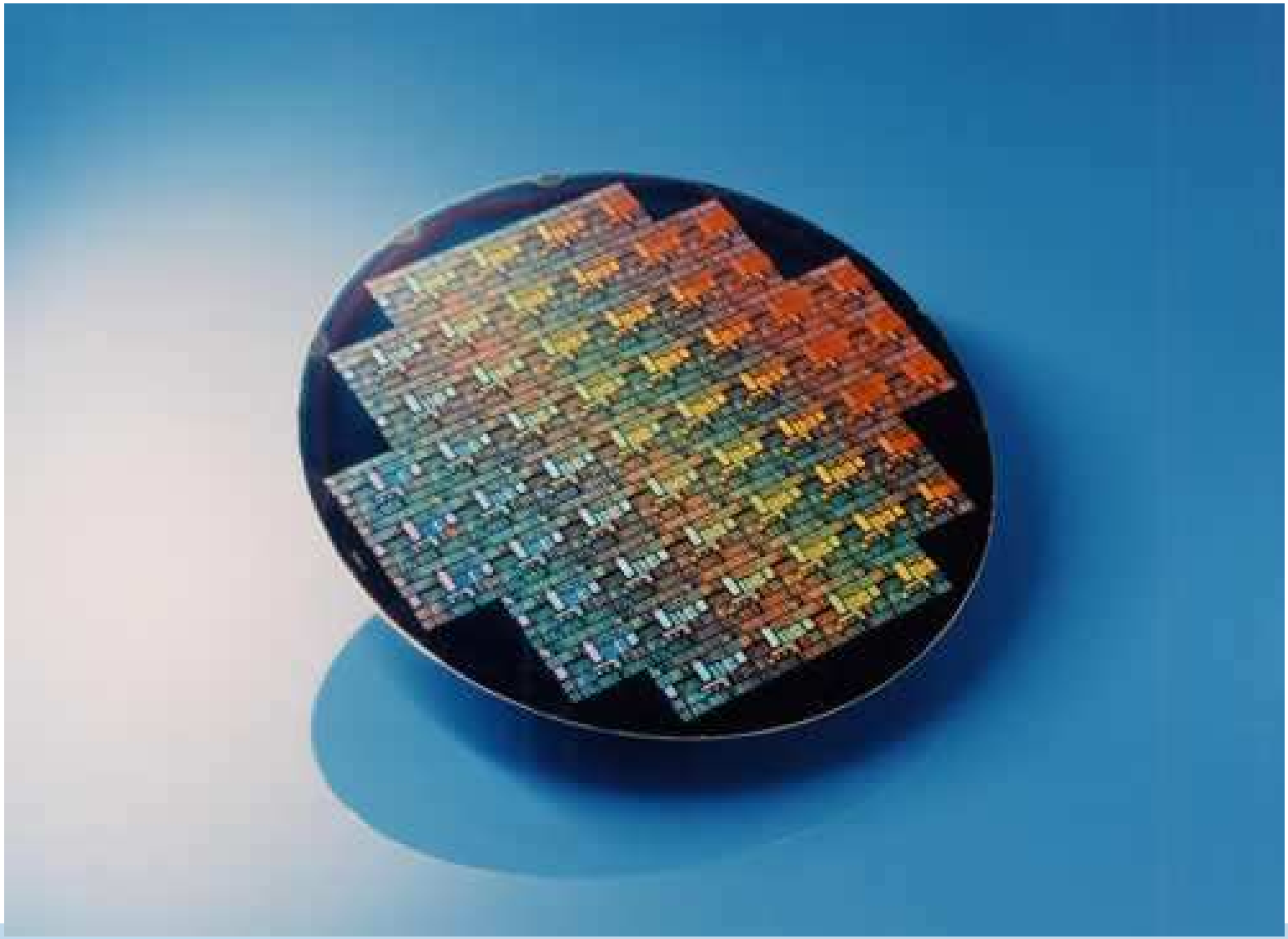
Architectures complexes



Architectures complexes

- Architecture hiérarchique : connexion de bus
- Nombre de composants importants
- Mémoire distribuée sur plusieurs modules
- Pas de limite en complexité

Technologie cible : System on Chip



Organisaiton logicielle

- Gestion des ressources
- Utilisation de la mémoire
- Utilisation des périphériques

Contraintes des logiciels

- Les sytèmes n'intégrent en général pas de protection mémoire
  - Tout l'espace d'adressage est accessible depuis l'applicatif
  - Nécessiter d'avoir un programme fiable et "sans bug"
- La programmation en assembleur reste présente au moins pour les pilotes et les couches basses
- Le reste peut être écrit en langage de haut niveau et compilé

- Une part importante du temps de conception est passée dans la mise au point du logiciel et du matériel

Systèmes d’exploitation

Système d’exploitations pour l’embarqué:

- Windows CE
- Wind River VxWorks
- Symbian
- Qnx
- Linux, μcLinux, RTLinux . . .
- OS propriétaires

Systèmes d’exploitation

- Système applicatifs : API de manipulation de matériel, une seule application
- Systèmes à commutation de tâches : temps partagé entre plusieurs programmes
  - coopératif, préemptif
  - tables des tâches statique / dynamique
  - gestion de priorité / temps réél
- Ajout d’une interface homme-machine
  - Prise en compte des interactions, interruptions
- Ajout d’un système de fichier / support de stockage
  - Possibilité de rajouter des programmes (et de les charger)
- Ajout d’une topographie mémoire (segmentation / pagination)
  - mémoire virtuelle
  - protection mémoire entre applications

  $\longrightarrow$ Unix

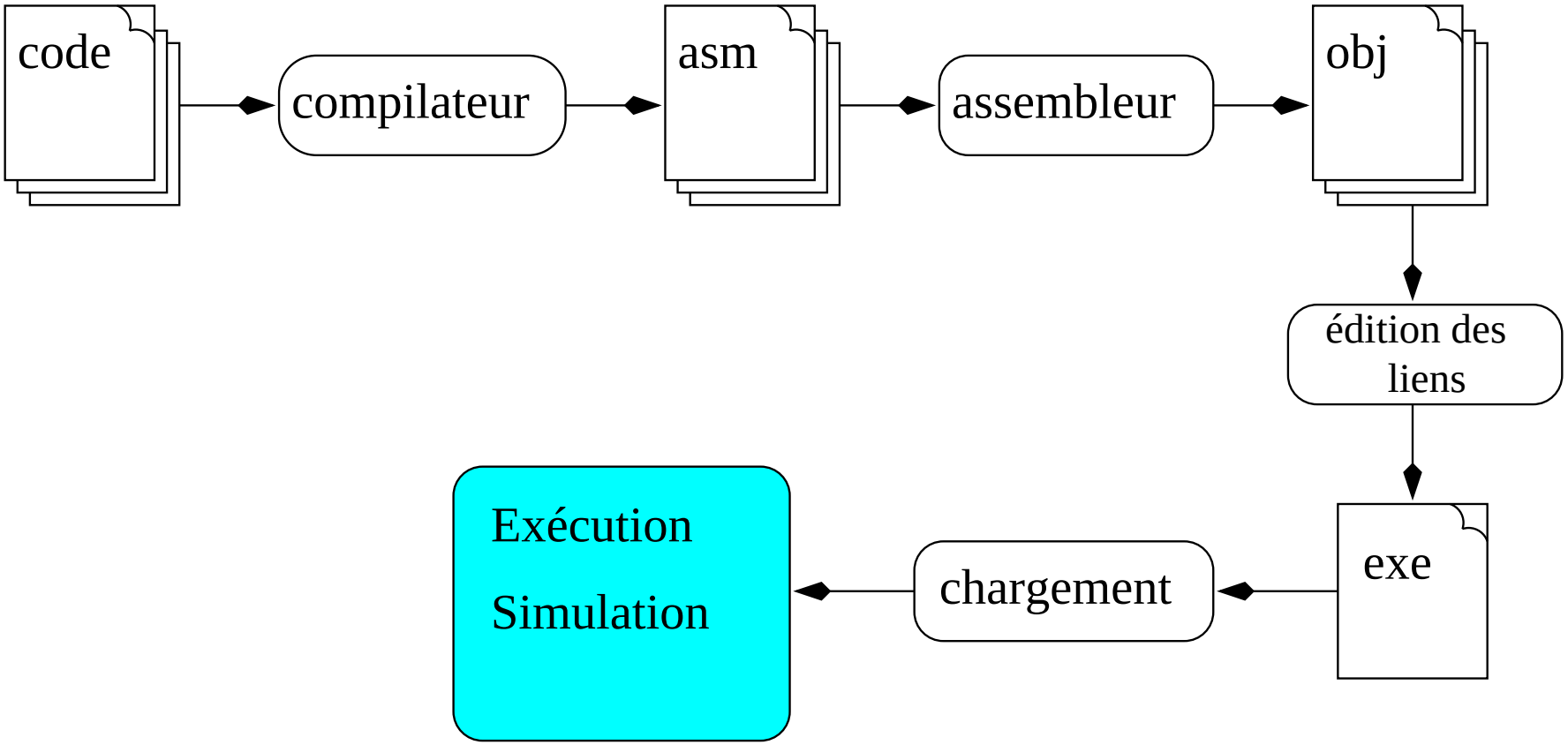
Systèmes multitâches

- Coopératif
  - L'application "rend la main"
    - Appels système bloquants (mutex par exemple)
    - Demande explicite
- Préemptif
  - Un timer matériel génère une interruption périodique
  - Le rôle du gestionnaire d'interruption est de choisir le prochain processus à être exécuté
    - Gestions de priorité
    - Notions de temps réel

Contraintes des systèmes

- Empreinte mémoire minimale
- Commutation de contexte rapide (latence)
- Gestion des tâches (ordonnancement)
- Gestion multiprocesseur (synchronisation)
- API de communication entre les applications souple
- Possibilité de communiquer facilement entre matériel et logiciel

Chaîne logicielle



Chaîne logicielle

Les étapes de compilation de l'application dépendent de la nature du système cible

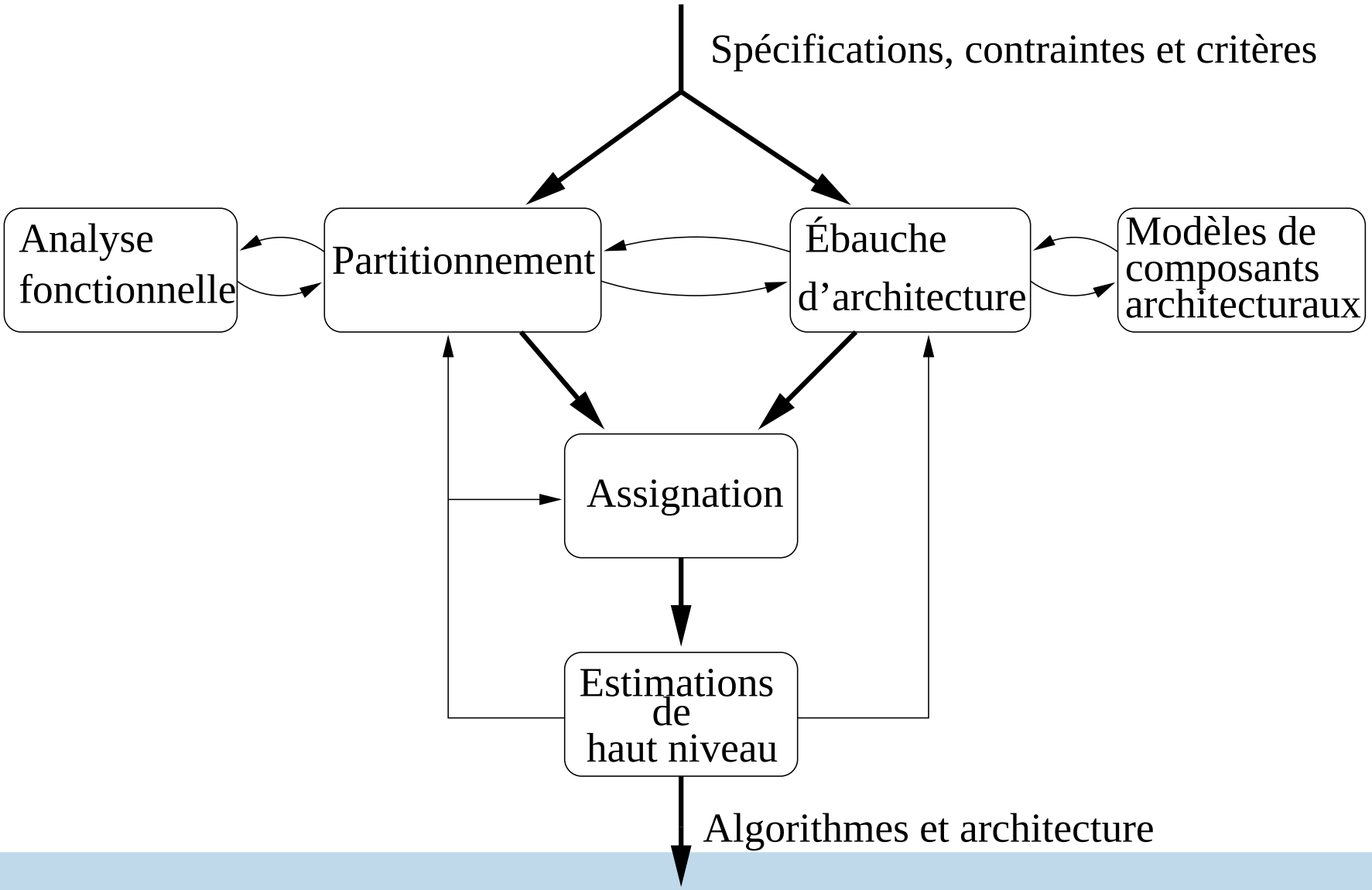
- Cross compilation classique pour les gros systèmes : l'application peut-être ajoutée dynamiquement au système
- Intégration du système à l'édition des liens : le chargement remplace tout à chaque fois

Conception de système

Le matériel et le logiciel peuvent être séparés, il faut les prendre en compte dans un processus de **codesign**.

- Modélisations du système complet
  - Étude des fonctionnalités et des tâches concurrentes
- Partitionnement Matériel / Logiciel
  - Hypothèses architecturales
  - Choix du processeur
  - Choix de l'interface
  - Estimation des contraintes physiques et économiques
- Modélisation Matérielle/Logicielle, Validation
  - Analyser la fonctionnalité en fonction des choix
  - Validation de fonctionnement et du respect des contraintes

Méthodologie de conception



Méthodologie de conception

Intérêts du «codesign»

- Conception rapide de SoC : time to market
  - Cycle de conception de haut niveau pour réduire les temps d'estimations des solutions
- Réduire la difficulté de validation et déboguage
  - Réutilisation d'IP
  - Modules reconfigurables
- Converger vers une solution optimale en fonction des contraintes de départ

Méthodologie de conception

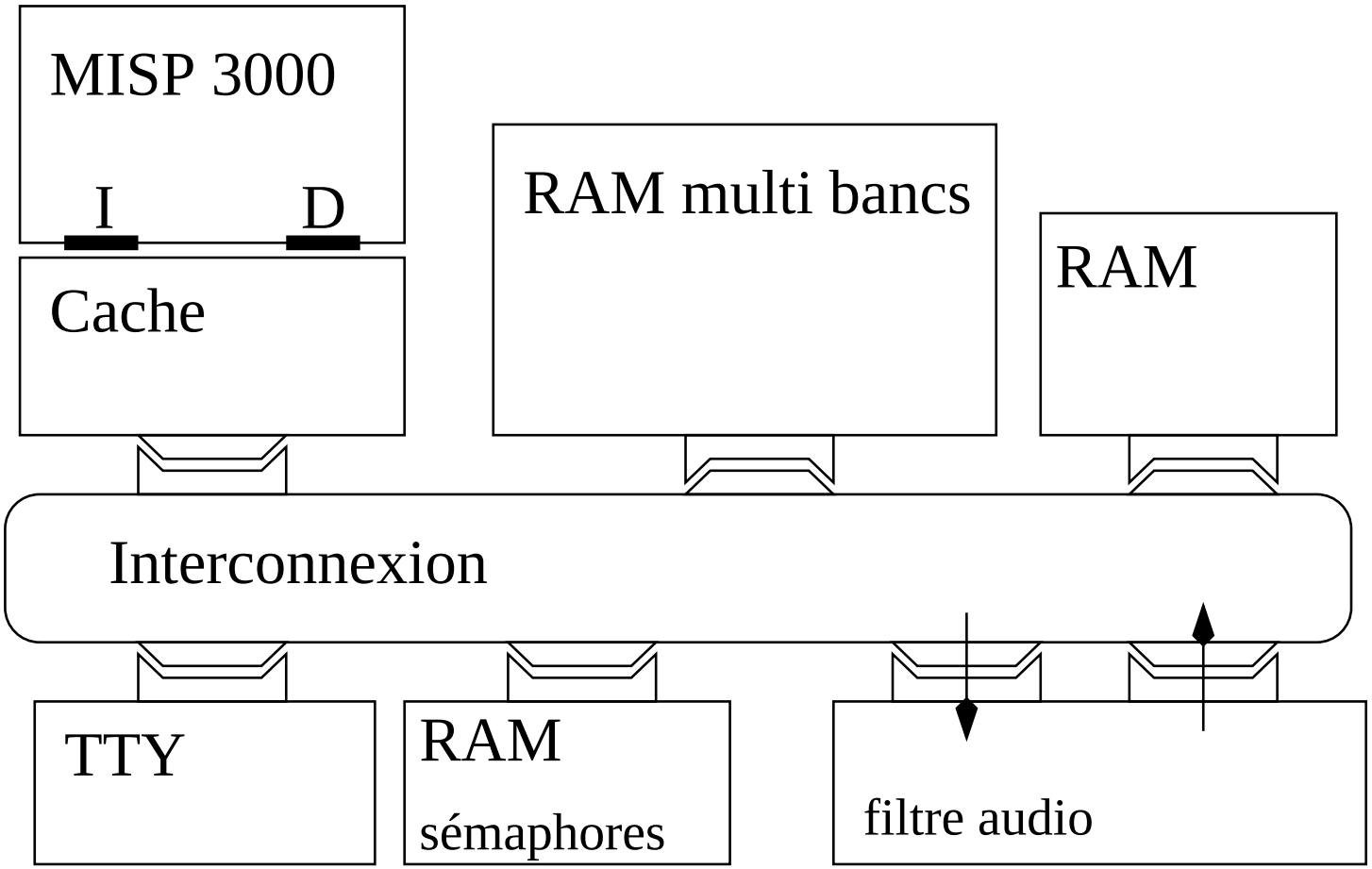
- La conception nécessite d'avoir une modélisation complète
- Approche logicielle
  - les objets migrent vers le HW jusqu'à ce que les contraintes de performances soit atteintes (pour un coût minimum)
- Approche matérielle
  - les objets migrent en SW tant que les contraintes de performances restent atteintes (pour un coût minimum)
- Le meilleur partitionnement nécessite en général l'expertise d'un concepteur.

SocLib

`http://soclib.lip6.fr/`

- Environnement de conception et de simulation gratuit et open source
- Utilise des modèles de composants écrits en SystemC
  - processeur
  - interconnexion
  - périphériques
  - ...
- Utilisation d'une chaîne de compilation standard (GNU)

Architecture de la démonstration



Organisation logicielle

- Système d'exploitation multithread : `Mutek`
  - Gestion de l'API de thread Posix
  - Gestion des interruptions
  - Commutation de contexte rapide
- Logiciel cross-compilé avec `gcc`
- Intégration du système avec l'applicatif à l'édition des liens

Utilisation de l’ensemble

démonstration