Conception d'un mini-processeur

Christophe Mouilleron

ensiie
école nationale supérieure d'informatique
pour l'industrie et l'entreprise

Introduction

Processeur =

- unité de calculs
  - types manipulés ?
  - opérations fournies ?

Introduction

Processeur =

- unité de calculs
  - types manipulés ?
  - opérations fournies ?
- programmable
  - assembleur ?
  - chargement du programme ?
  - entrées/sorties ?

Introduction

Processeur =

- unité de calculs
  - types manipulés ?
  - opérations fournies ?
- programmable
  - assembleur ?
  - chargement du programme ?
  - entrées/sorties ?

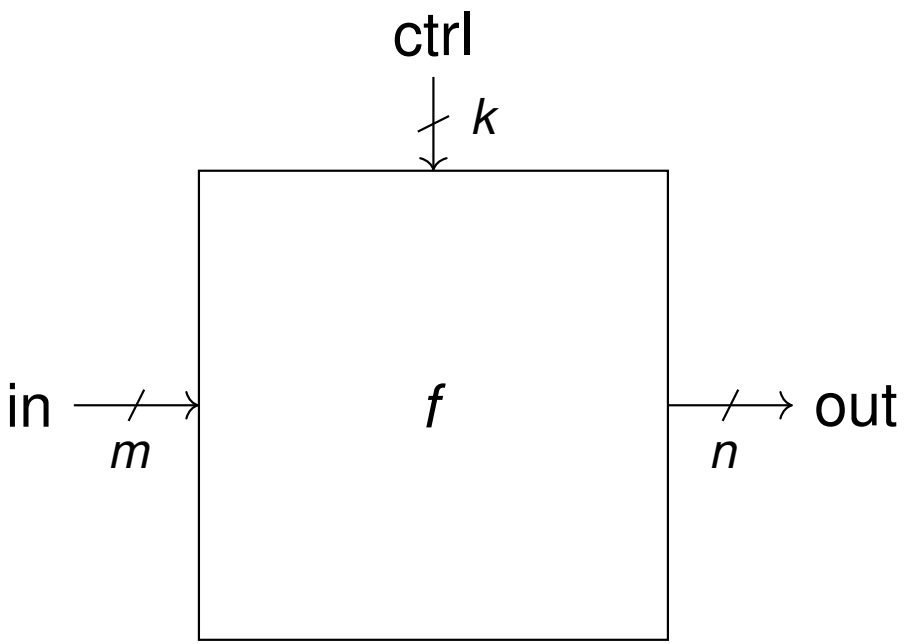
Objectif du jour :

- conception d'un mini-processeur
- réalisation dans Diglog
- tests avec de vrais programmes

Plan

Plan

Rappels : circuits combinatoires



$\text{out} = f(\text{ctrl}, \text{in})$

- multiplexeurs

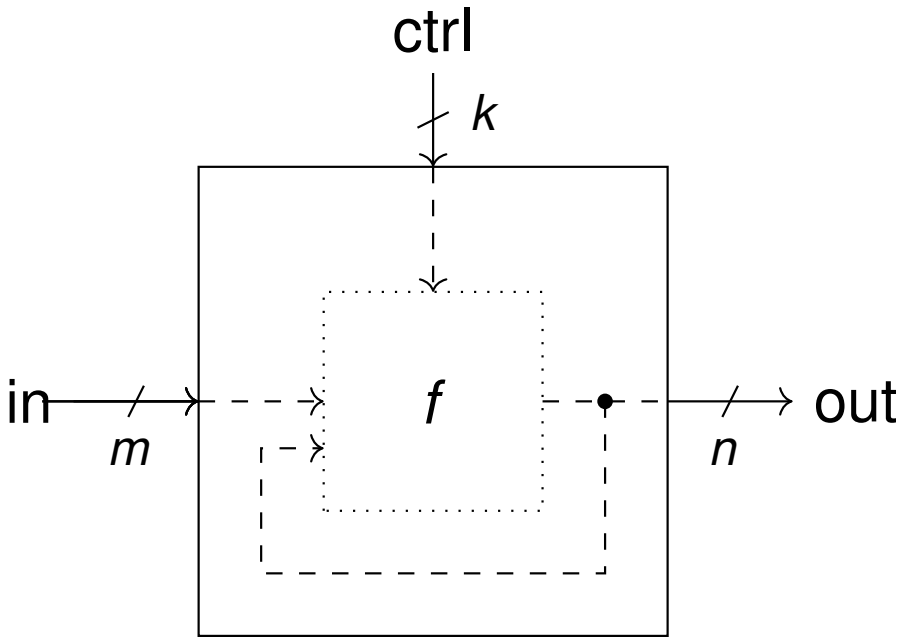
$$k = 1 \qquad m = 2n$$

$$\text{out}_i = \begin{cases} \text{in}_i & \text{si } \text{ctrl}_0 = 0 \\ \text{in}_{i+n} & \text{si } \text{ctrl}_0 = 1 \end{cases}$$

- décodeurs
- ALU
- fonctions booléennes

✓ réalisation par méthode directe, Karnaugh, etc.

Rappels : circuits séquentiels



$$\text{out(t)} = f\Big(\text{ctrl}, \text{in}(t-1), \text{out}(t-1)\Big)$$

- bascule D

  $k = 0 \qquad m = n = 1$

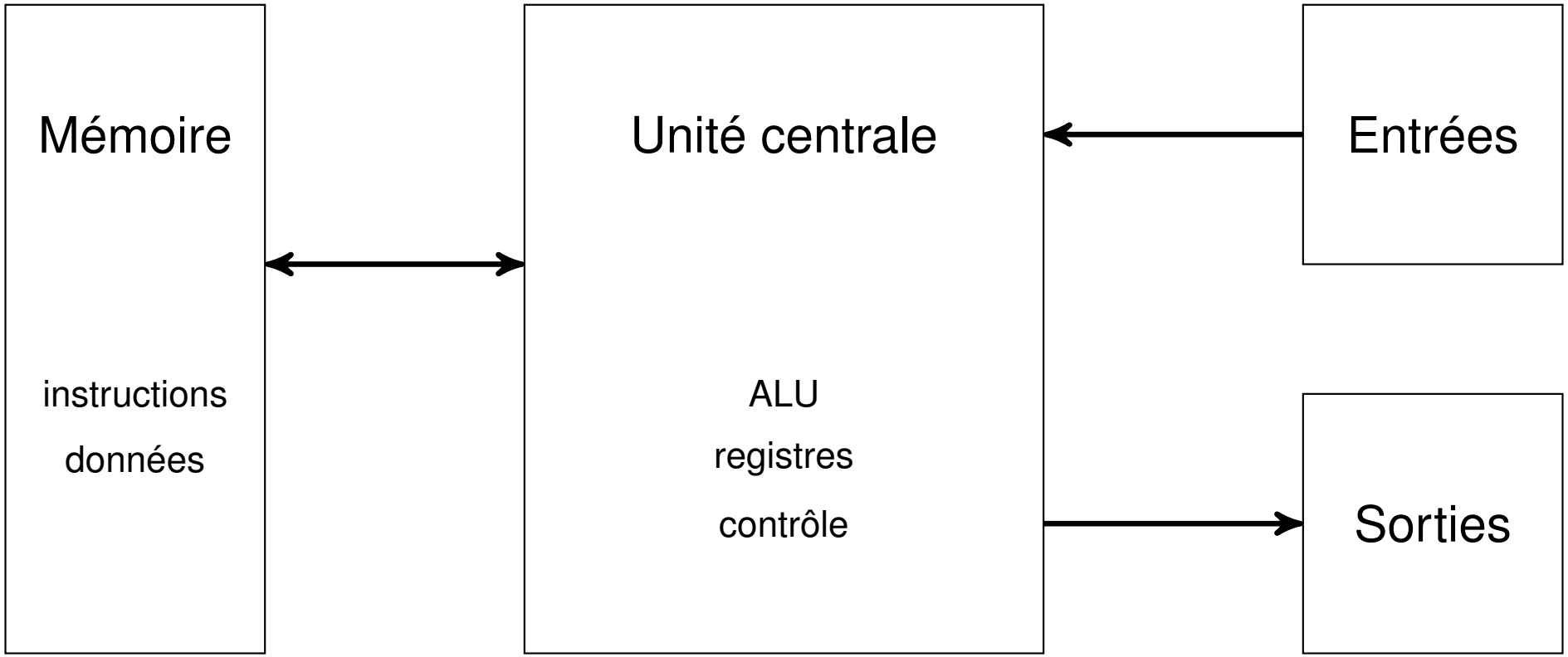
  $\text{out}(t) = \text{in}(t-1)$

- registre / mémoire SRAM
- compteurs
- automates de Moore et de Mealy

✓ réalisation des circuits synchrones avec bascules + combinatoire

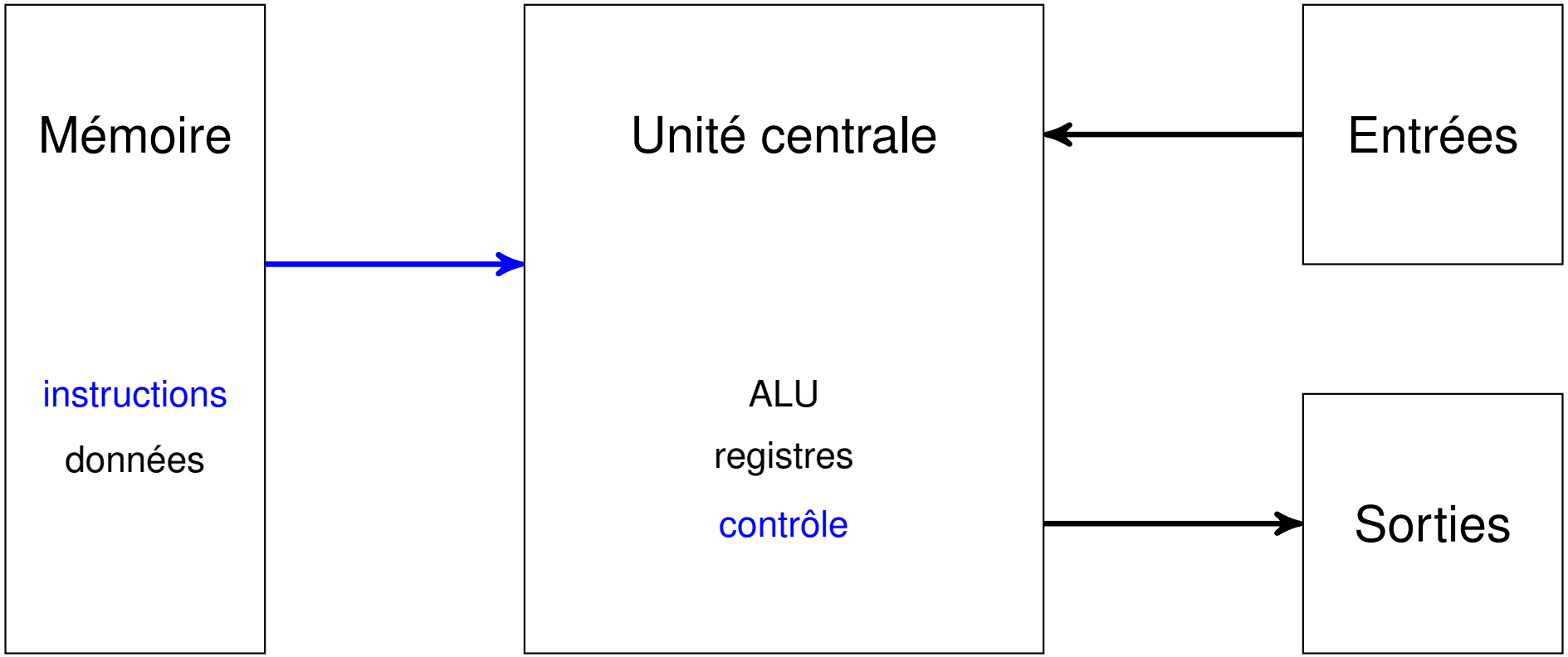
Architecture de von Neumann (1)

Modèle d'ordinateur :



Architecture de von Neumann (1)

Modèle d'ordinateur :

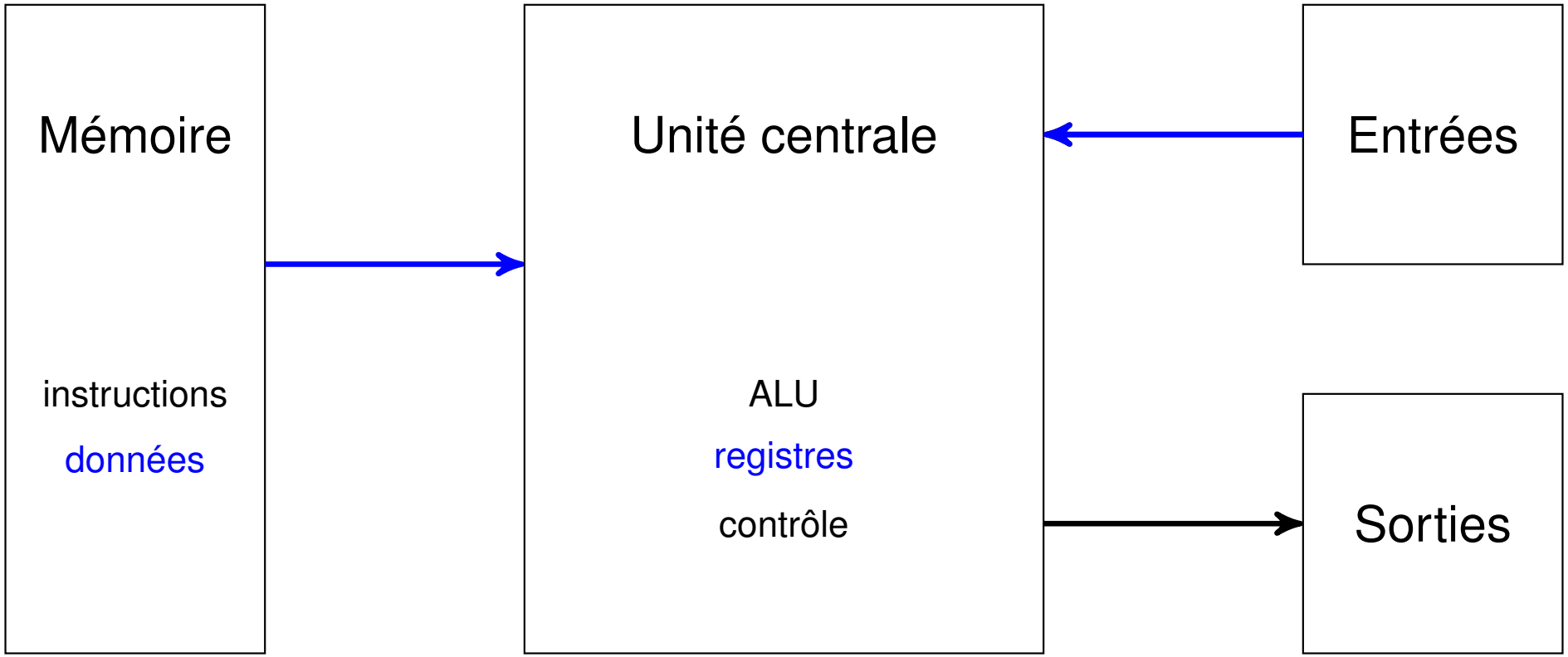


Cycle de calcul :

1. décodage de l'instruction

Architecture de von Neumann (1)

Modèle d'ordinateur :

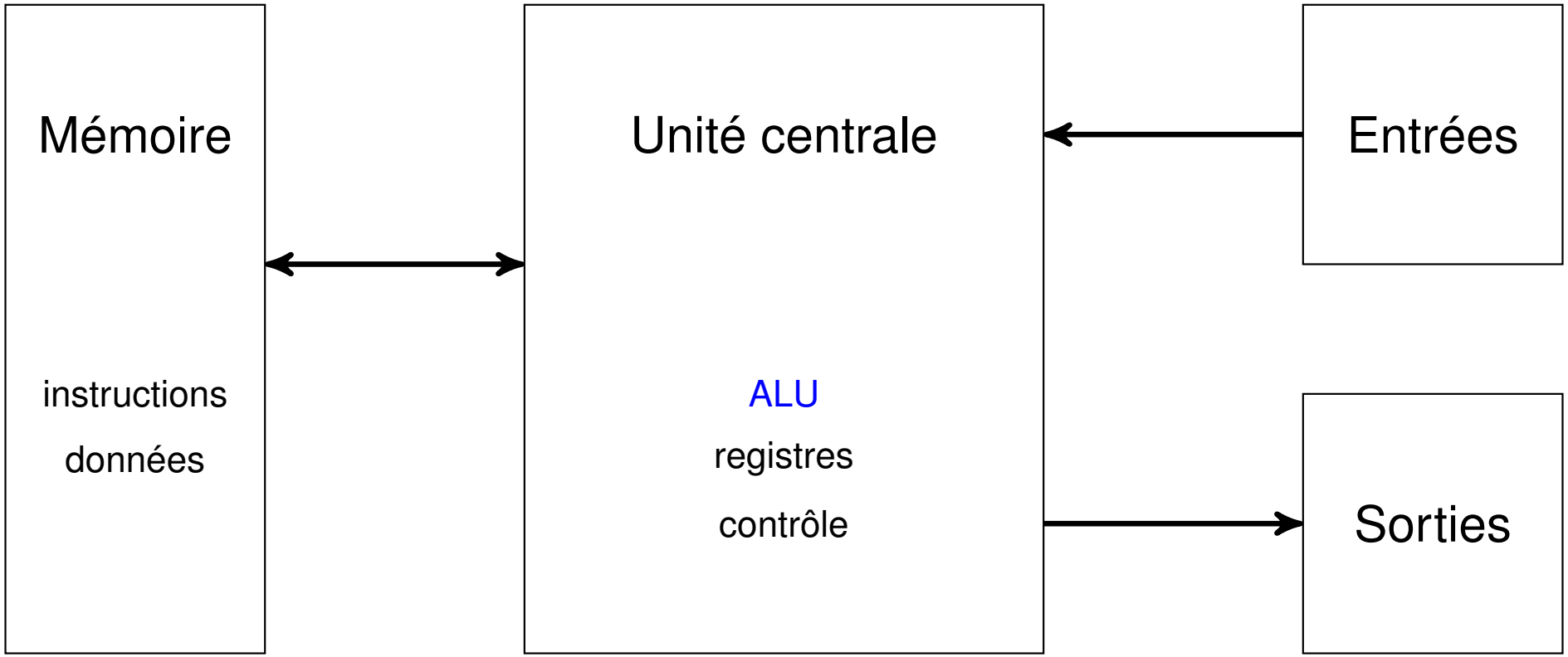


Cycle de calcul :

2. lecture des opérandes

Architecture de von Neumann (1)

Modèle d'ordinateur :

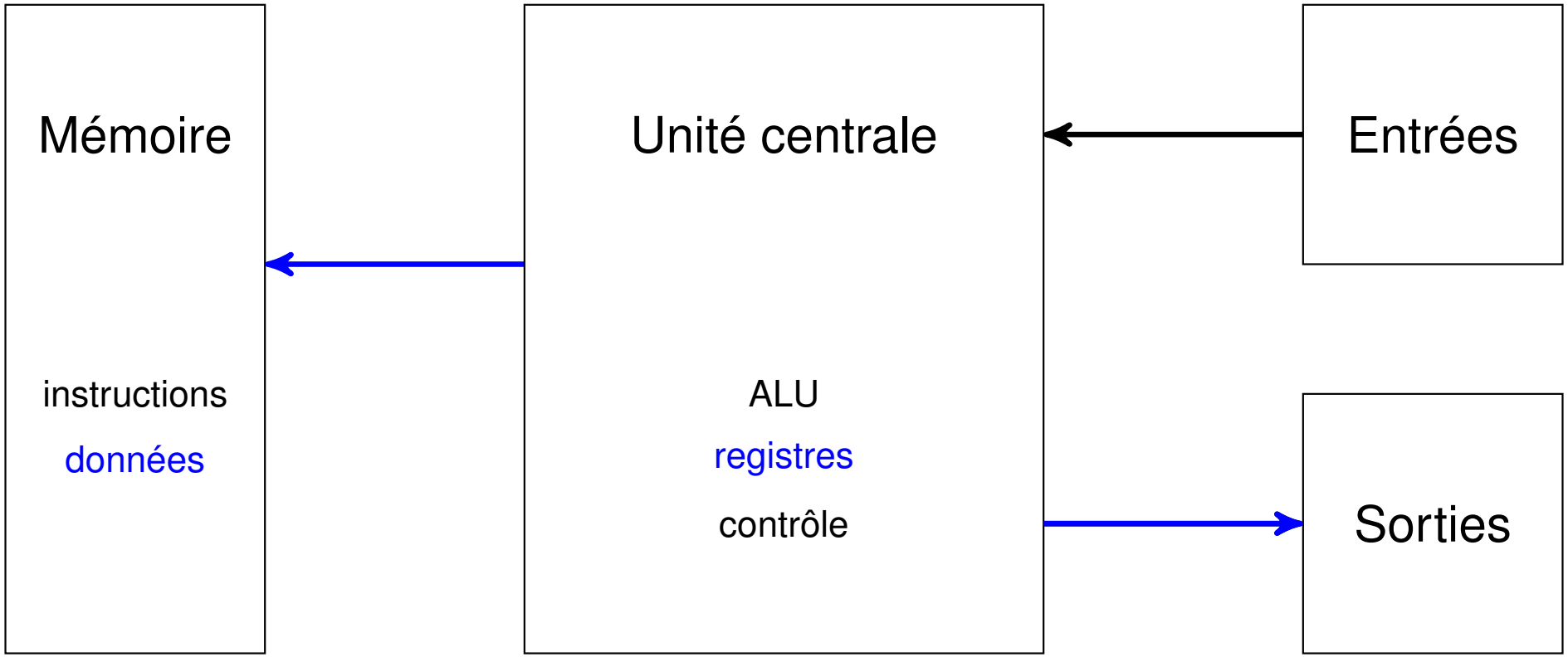


Cycle de calcul :

3. exécution de l'instruction

Architecture de von Neumann (1)

Modèle d'ordinateur :

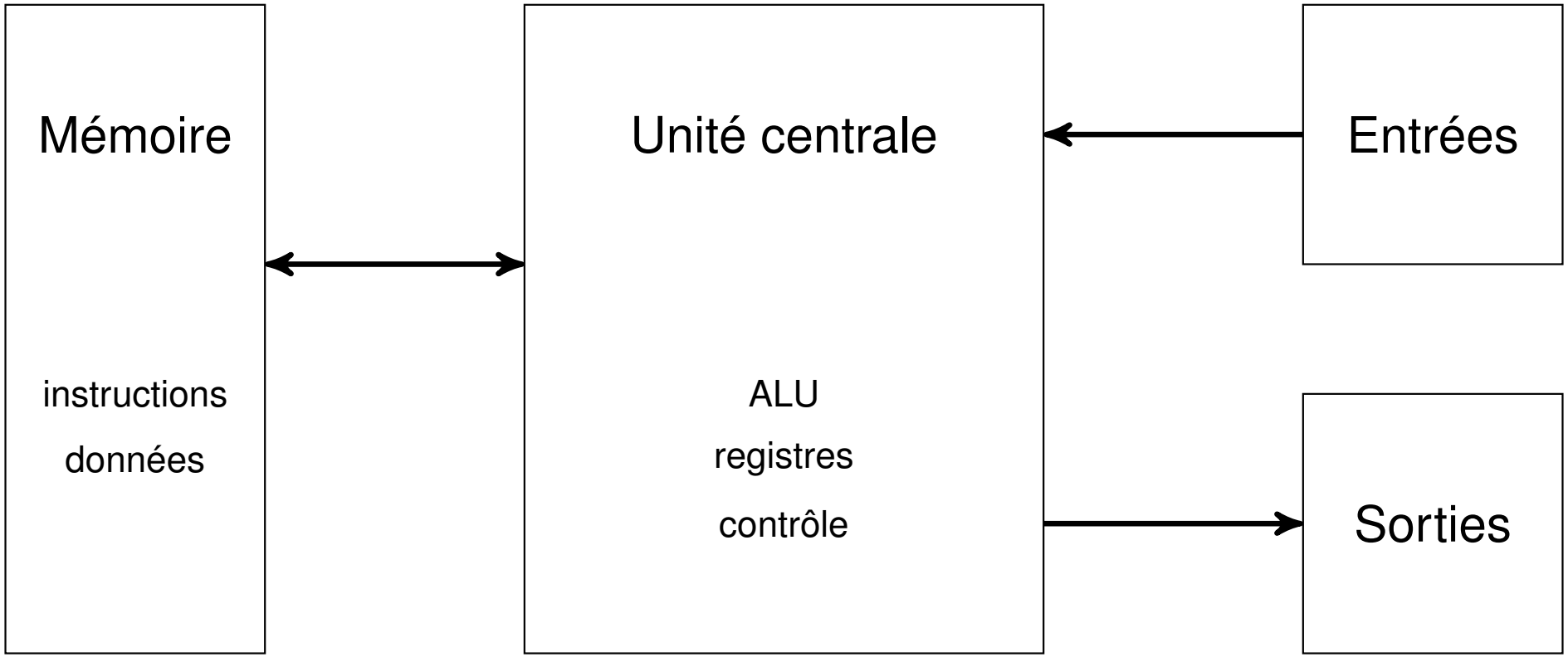


Cycle de calcul :

4. écriture du résultat

Architecture de von Neumann (1)

Modèle d'ordinateur :



Architecture de von Neumann (2)

Cycle de calcul :

1. décodage de l'instruction — combinatoire
2. lecture des opérandes — séquentiel
3. exécution de l'instruction — combinatoire
4. écriture du résultat — séquentiel

Architecture de von Neumann (2)

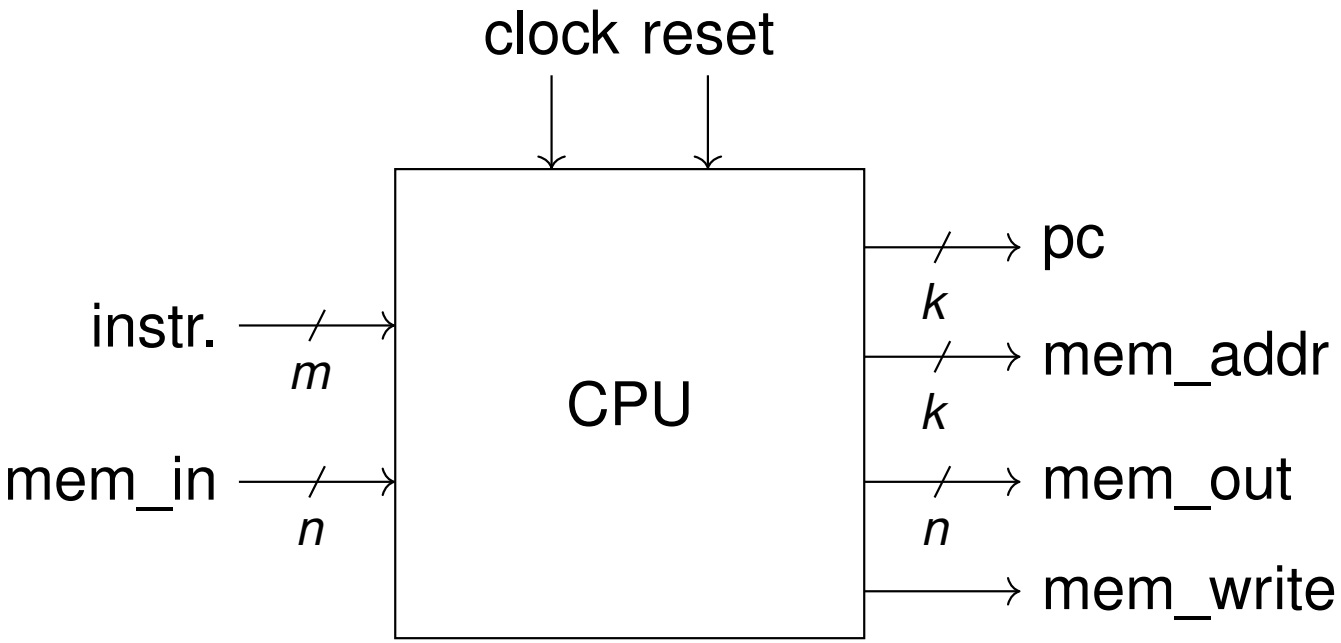
Cycle de calcul :

1. décodage de l'instruction — combinatoire
2. lecture des opérandes — séquentiel
3. exécution de l'instruction — combinatoire
4. écriture du résultat — séquentiel

Choix pour notre mini-processeur :

- lecture quand horloge à 1 — état haut
- écriture quand horloge à 0 — état bas
- séparation mémoire
  - ROM pour le programme
  - SRAM pour les données

Processeur

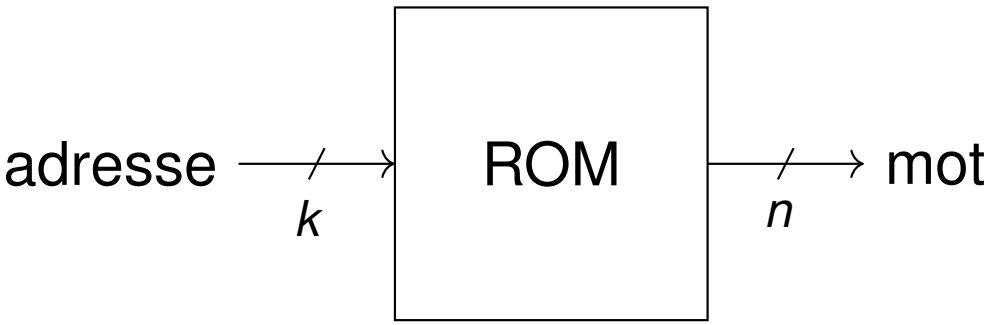


CPU = Circuit synchrone

- mémoire interne = banc de registre
- unité de calculs = ALU
- gestion contrôle + entrées/sorties = combinatoire

Mémoire ROM

ROM = Read Only Memory



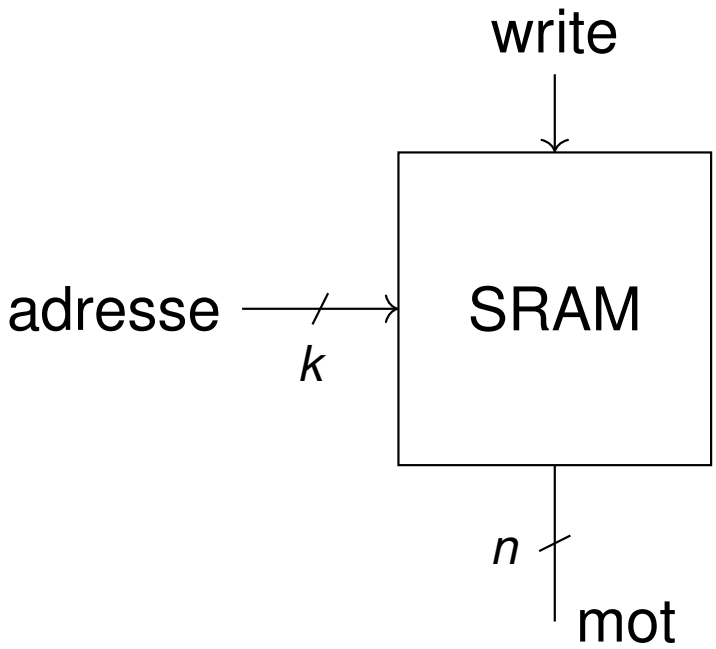
- mots sur n bits
- 2^k adresses

$\rightsquigarrow$ mémoire SRAM sans écriture

sauf à l'init.

Mémoire SRAM

SRAM = Static Random Access Memory



- Si write $= 0$

  $\text{mot} \leftarrow \text{MEM[adresse]}$

- Si write $= 1$

  $\text{MEM[adresse]} \leftarrow \text{mot}$

$\rightsquigarrow$ composant `SRAM8K` dans Diglog

Gestion des entrées/sorties

Problème : E/S asynchrones $\Rightarrow$ données asynchrones

Solutions :

1. utilisation d'une mémoire tampon par périphérique

2. utilisation d'une zone de la RAM prédéfinie

Gestion des entrées/sorties

Problème : E/S asynchrones $\Rightarrow$ données asynchrones

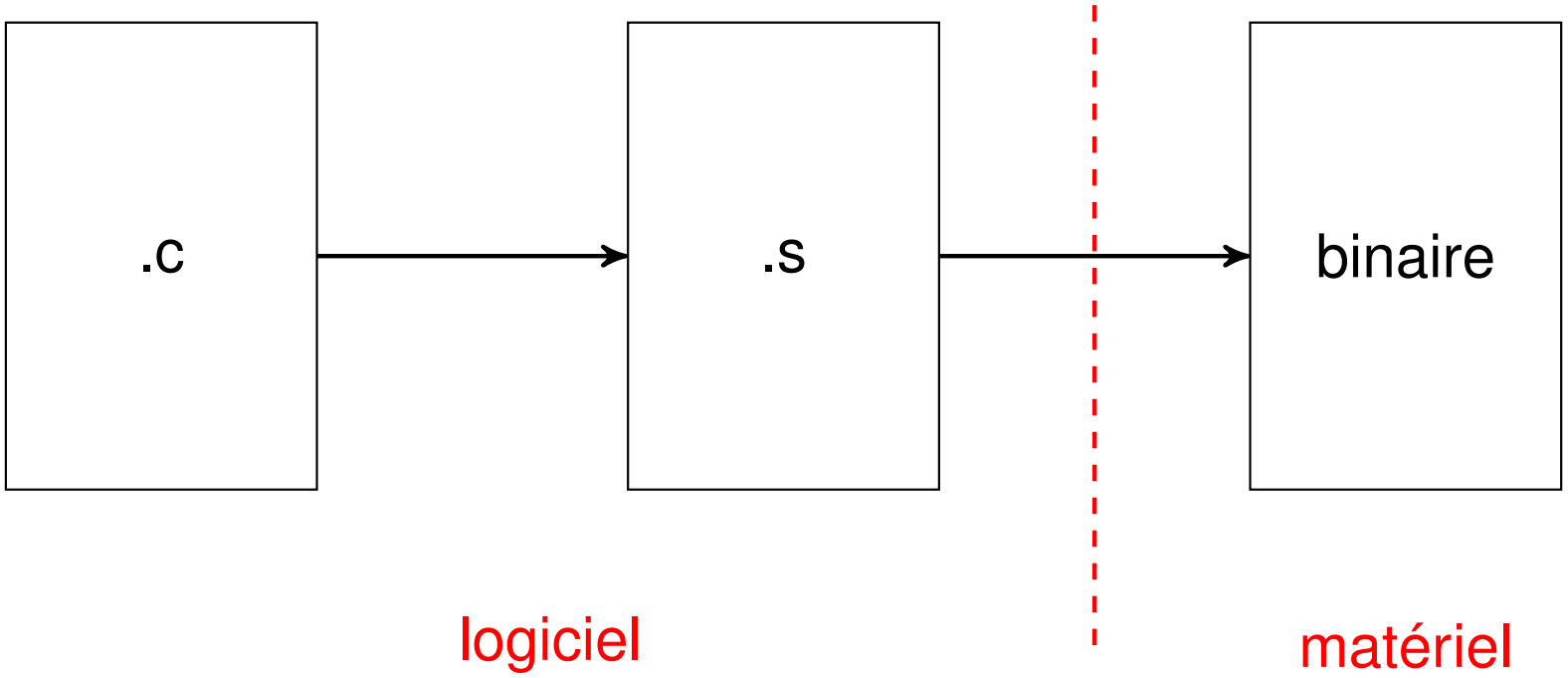
Solutions :

1. utilisation d'une mémoire tampon par périphérique
   - ✓ facile à mettre en œuvre
   - ✓ accès dédié au périphérique
   - ✗ capacité limité

2. utilisation d'une zone de la RAM prédéfinie
   - ✓ capacité adaptable au besoin
   - ✓ interaction avec le CPU simplifiée
   - ✗ sacrifice d'une partie de la RAM

Plan

Compilation et exécution d'un programme



jeu d'instructions = abstraction du matériel

processeur = implantation particulière du jeu d'instructions

Le jeu d'instructions

Description des capacités du processeur :

- type de processeur — généraliste, FPU, GPU
- quantité et types de registres
- modes d'adressage
- modes de branchement — instructions à prédicat
- parallélisme — SIMD, VLIW

Modèle d'exécution (n, m)

n = nb. d'opérandes par instruction

m = nb. d'accès mémoire par instruction hors load/store

Exemples :

- MIPS ?
- Intel x86 ?

Modèle d'exécution (n, m)

n = nb. d'opérandes par instruction

m = nb. d'accès mémoire par instruction hors load/store

Exemples :

- MIPS ? $(3, 0)$
- Intel x86 ? $(2, 1)$

Modèle d'exécution (n, m)

n = nb. d'opérandes par instruction

m = nb. d'accès mémoire par instruction hors load/store

Exemples :

- MIPS ? $(3, 0)$
- Intel x86 ? $(2, 1)$
- $(1, 0)$ = machine à accumulateur

Modèle d'exécution (n, m)

n = nb. d'opérandes par instruction

m = nb. d'accès mémoire par instruction — hors load/store

Exemples :

- MIPS ? $(3, 0)$
- Intel x86 ? $(2, 1)$
- $(1, 0)$ = machine à accumulateur
- $(0, 0)$ = machine à pile — notation polonaise inversé

Modèle d'exécution (n, m)

n = nb. d'opérandes par instruction

m = nb. d'accès mémoire par instruction — hors load/store

Exemples :

- MIPS ? $(3, 0)$
- Intel x86 ? $(2, 1)$
- $(1, 0)$ = machine à accumulateur
- $(0, 0)$ = machine à pile — notation polonaise inversé

Exercice :

Soient `A0`, `A1` et `A2` des adresses de 3 entiers en mémoire.
Coder `MEM[A2]` $\leftarrow$ `MEM[A0]` + `MEM[A1]` dans les 4 modèles.

Réponse à l'exercice (1)

En $(3, 0)$:

En $(2, 1)$:

Réponse à l'exercice (1)

En $(3, 0)$:

```
ld  r0, A0       // r0 <- MEM[A0]
ld  r1, A1       // r1 <- MEM[A1]
add r0, r1, r0   // r0 <- r1 + r0
st  r0, A2       // MEM[A2] <- r0
```

En $(2, 1)$:

Réponse à l'exercice (1)

En $(3, 0)$:

```
ld  r0, A0       // r0 <- MEM[A0]
ld  r1, A1       // r1 <- MEM[A1]
add r0, r1, r0   // r0 <- r1 + r0
st  r0, A2       // MEM[A2] <- r0
```

En $(2, 1)$:

```
ld  r0, A0       // r0 <- MEM[A0]
add r0, MEM[A1]  // r0 <- r0 + MEM[A1]
st  r0, A2       // MEM[A2] <- r0
```

Réponse à l'exercice (2)

En $(1, 0)$:

En $(0, 0)$:

Réponse à l'exercice (2)

En $(1,0)$:

```
ld  A0     // acc <- MEM[A0]
mov r1     // r1  <- acc
ld  A1     // acc <- MEM[A1]
add r1     // acc <- acc + r1
st  A2     // MEM[A2] <- acc
```

En $(0,0)$:

```
ld A0      // [ MEM[A0] ]
ld A1      // [ MEM[A1] , MEM[A0] ]
add        // [ x ] avec x = MEM[A1] + MEM[A0]
st A2      // [ ]
           // et MEM[A2] <- x
```

Choix pour le jeu d'instruction

Modèle d'exécution :

- compromis entre la simplicité de $(3, 0)$
  et la facilité d'accès à la mémoire ($m \geq 1$)

Encodage des instructions :

- taille fixe
- taille variable

Choix pour le jeu d'instruction

Modèle d'exécution :

- compromis entre la simplicité de $(3, 0)$
  et la facilité d'accès à la mémoire ($m \geq 1$)

Encodage des instructions :

- taille fixe
  - ✓ régularité
- taille variable
  - ✗ décodage complexe

Choix pour le jeu d'instruction

Modèle d'exécution :

- compromis entre la simplicité de $(3, 0)$
  et la facilité d'accès à la mémoire ($m \geq 1$)

Encodage des instructions :

- taille fixe
  - ✓ régularité
  - ✗ nb. d'instructions limité ou taille importante
- taille variable
  - ✗ décodage complexe
  - ✓ taille réduite pour instructions courantes
  - ✓ nb. d'instructions moins limité

Plan

Choix effectués

Choix effectués

Modèle d'exécution $(3, 0)$:

- plus simple à comprendre / mettre en œuvre
- accès mémoire uniquement via `ld` et `st`

8 registres (8 bits) généraux :

- utilisation libre
- compromis expressivité / taille du banc de registre

Instruction de taille fixe = 16 bits :

- accès à la ROM constant
- compromis expressivité / taille

Encodage des instructions (1)

Contraintes :

- architecture 8 bits $\Rightarrow$ valeurs immédiates sur 8 bits
- 8 registres $\Rightarrow$ 3 bits pour désigner un registre
- adresses sur 13 bits

Encodage des instructions (1)

Contraintes :

- architecture 8 bits $\Rightarrow$ valeurs immédiates sur 8 bits
- 8 registres $\Rightarrow$ 3 bits pour désigner un registre
- adresses sur 13 bits

Types d'instruction selon nature des arguments :

- type 1a =
- type 1b =
- type 2 =
- type 3 =

Encodage des instructions (1)

Contraintes :

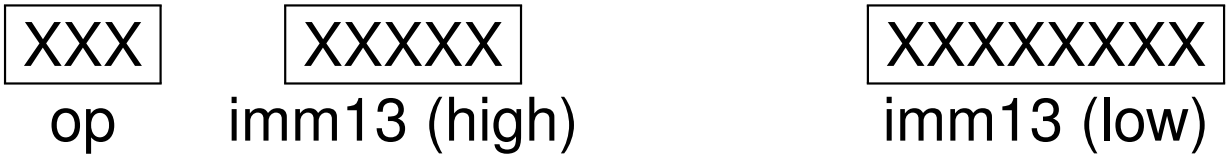
- architecture 8 bits $\Rightarrow$ valeurs immédiates sur 8 bits
- 8 registres $\Rightarrow$ 3 bits pour désigner un registre
- adresses sur 13 bits

Types d'instruction selon nature des arguments :

- type 1a = 3 registres
- type 1b = 2 registres + 1 immédiat
- type 2 = 1 registre + 1 immédiat
- type 3 = 1 adresse

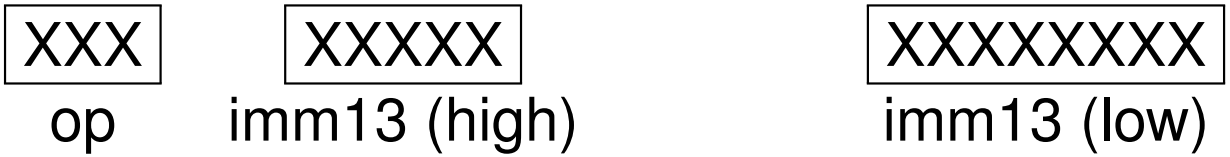
Encodage des instructions (2)

Type 3 :



Encodage des instructions (2)

Type 3 :

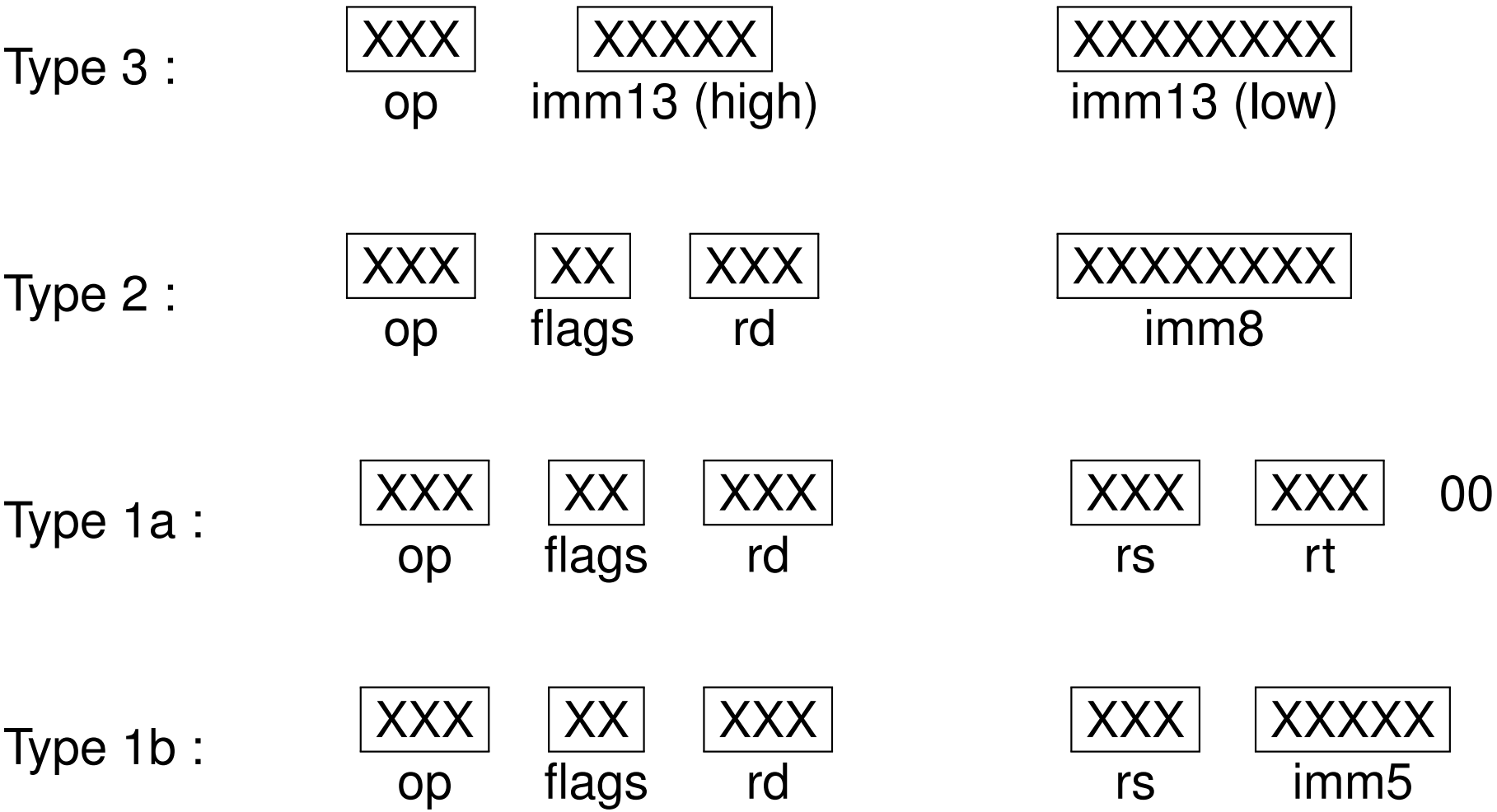


Type 2 :

Type 1a :

Encodage des instructions (2)

Encodage des instructions (2)



Résumé du jeu d'instructions

op	catégorie	instructions
000	autre	`nop` `ldi`
001	logique	`not` `or` `and` `lsr`
010	arithmétique 1	`add(i)` `sub(i)`
011	arithmétique 2	`mul(i)`
100	mémoire	`ld` `st` `in` `out`
101	saut autre	`jr`
110	saut conditionnel	`jeq` `jle` `jlt` `jne`
111	saut absolu	`jmp`

- `jr` = saut à l'adresse donnée sur 2 registres
- `in` = lecture dans le buffer du clavier
- `out` = écriture dans le buffer de l'écran

Objectif en TP

Implantation du processeur dans Diglog :

- décodage des instructions / calcul des bits de contrôle
- élaboration du chemin de données
- amélioration de l'ALU pour gérer les sauts conditionnels
- écriture de programmes de tests
- validation du bon fonctionnement du mini-processeur