Jeux d’instructions et modèles d’exécution

Vincent Risch, mai 2008, révision mai 2014

I.U.T., Aix-Marseille Université

Introduction

- Ordinateur : dispositif servant à l'exécution d'un programme, c'est-à-dire d'un ensemble d'*instructions* rangées dans la mémoire principale.

Introduction

- Ordinateur : dispositif servant à l’exécution d’un programme, c’est-à-dire d’un ensemble d’*instructions* rangées dans la mémoire principale.
- Chaque instruction est successivement lue à partir de la mémoire, c’est-à-dire transférée à l’UC pour y être *décodée*, puis *exécutée*.

- Ordinateur : dispositif servant à l'exécution d'un programme, c'est-à-dire d'un ensemble d'*instructions* rangées dans la mémoire principale.
- Chaque instruction est successivement lue à partir de la mémoire, c'est-à-dire transférée à l'UC pour y être *décodée*, puis *exécutée*.
- Ce processus se fait par l'intermédiaire de *cycles* rythmés par une horloge.

Cycles

On distingue :

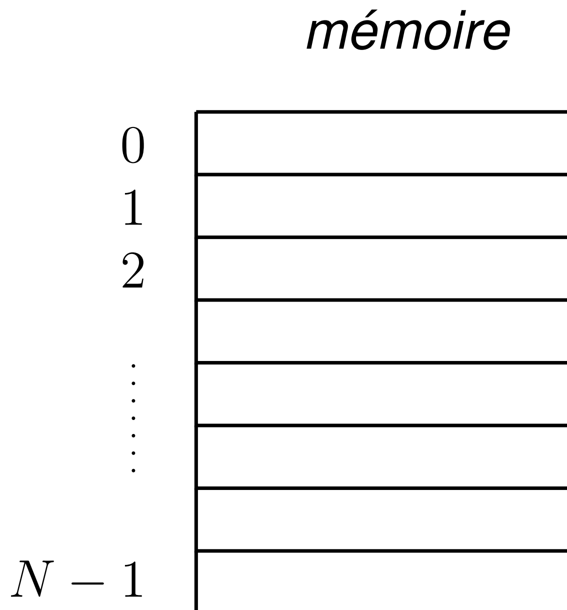
- Un *cycle de recherche* : l'instruction à exécuter, pointée par le Compteur de Programme (PC) est extraite de la mémoire et recopiée dans une registre du processeur, appelé Registre Instruction (IR).

On distingue :

- Un *cycle de recherche* : l'instruction à exécuter, pointée par le Compteur de Programme (PC) est extraite de la mémoire et recopiée dans une registre du processeur, appelé Registre Instruction (IR).
- Un *cycle d'exécution* : l'instruction stockée dans IR est interprétée par le dispositif de commande, des signaux sont générés et envoyés au sein de l'UC pour permettre le traitement de l'instruction.

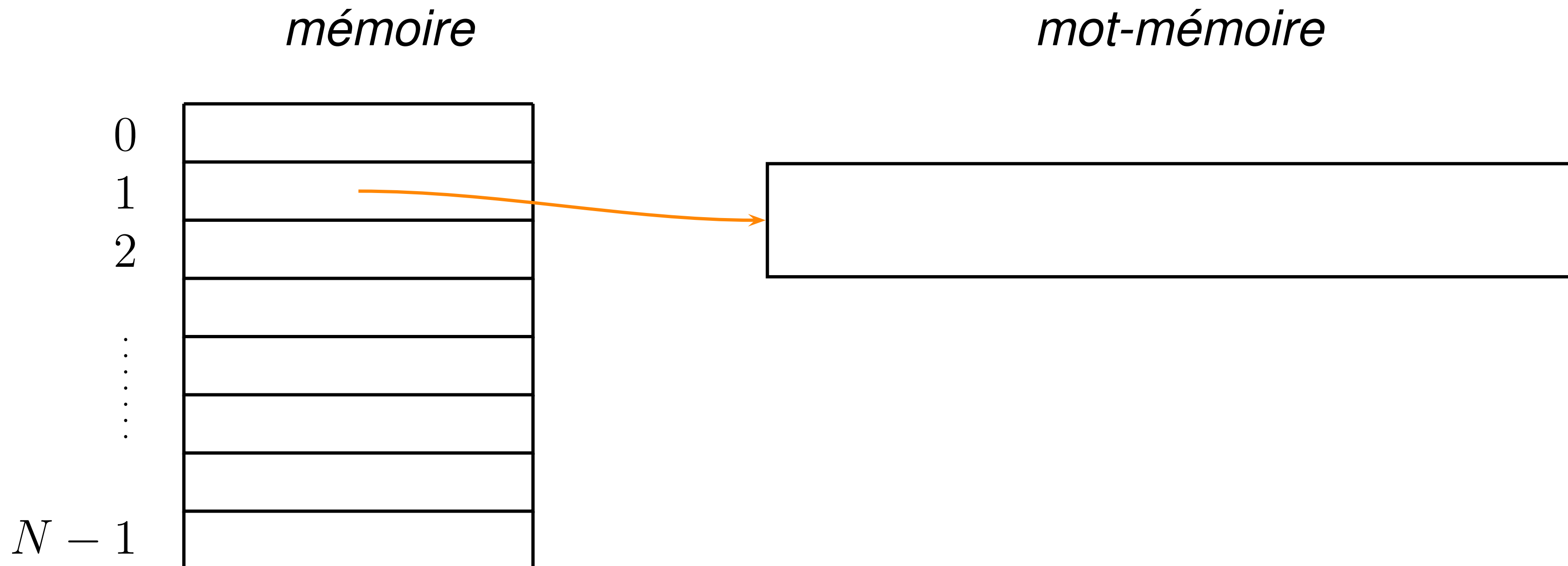
Organisation de la mémoire

Ensemble *aligné* de N *mots*, eux même composés d'un nombre fixe d'octets.



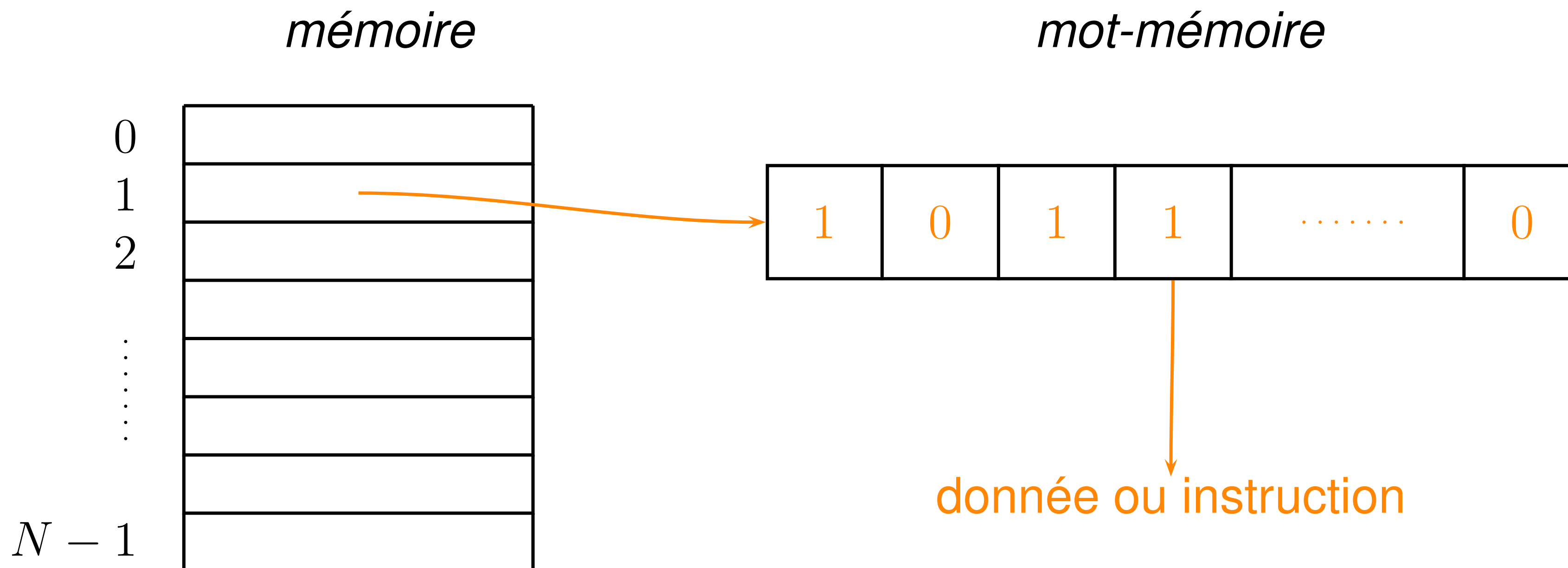
Organisation de la mémoire

Ensemble *aligné* de N *mots*, eux même composés d'un nombre fixe d'octets.



Organisation de la mémoire

Ensemble *aligné* de N *mots*, eux même composés d'un nombre fixe d'octets.



Format d'instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Format d'instruction

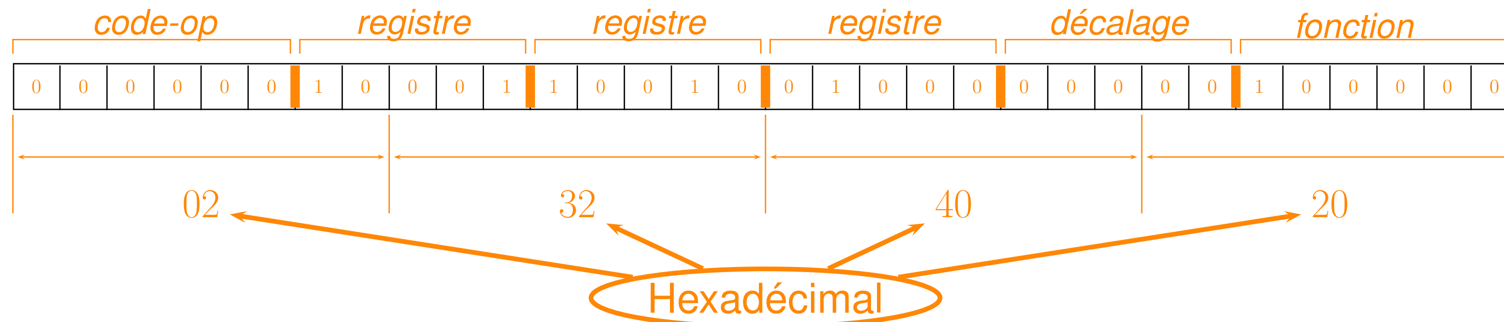
Par exemple, sur 32 bits, un mot mémoire (MIPS) :

Format d'instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :

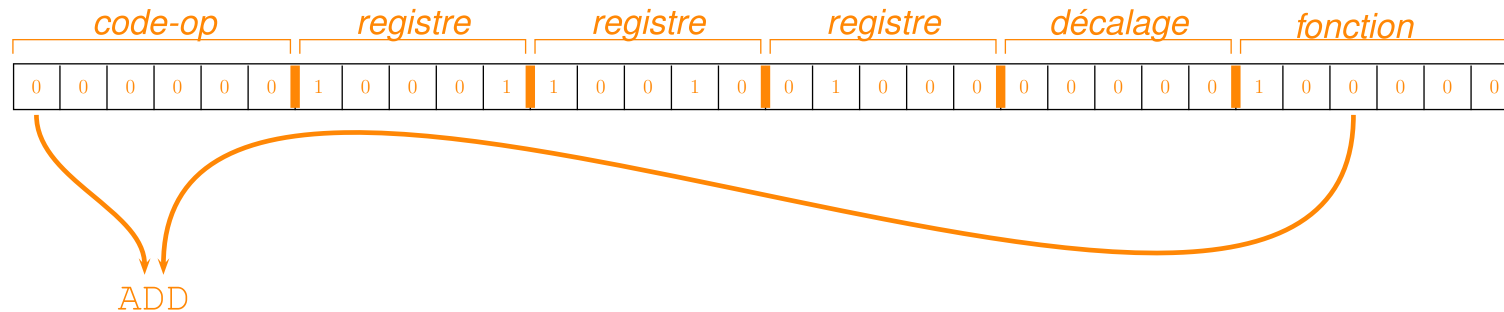
Format d'instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :



Format d'instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :

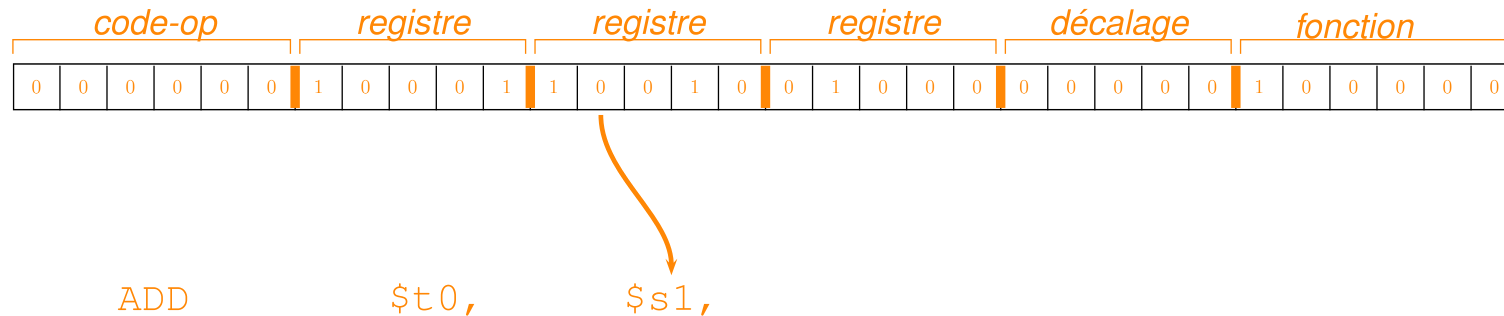


Format d'instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :

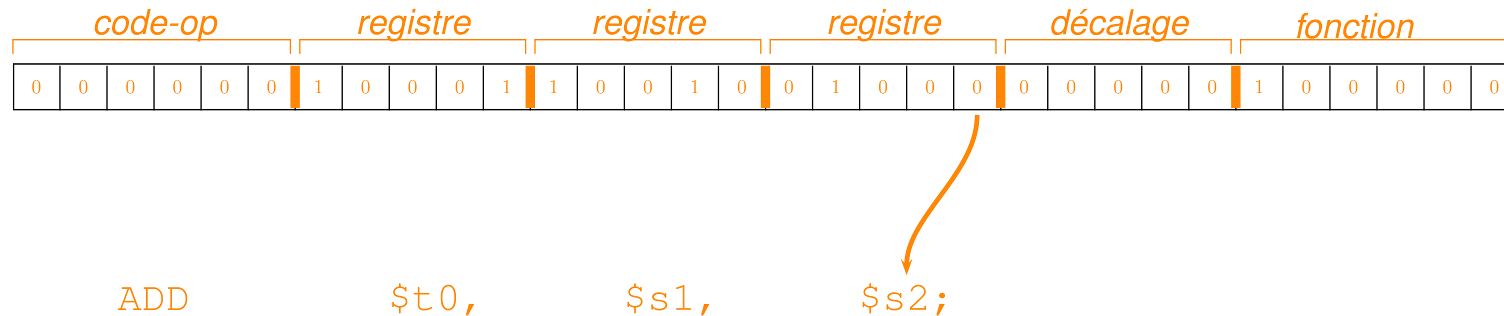
Format d’instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :



Format d’instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :



Format d'instruction

Par exemple, sur 32 bits, un mot mémoire (MIPS) :

`ADD        $t0,      $s1,      $s2;`

Issu de la compilation de l'instruction C/C++ : `C=A+B;`

Jeu d'instructions

- Ensemble des instructions qu'un processeur peut exécuter.

Jeu d'instructions

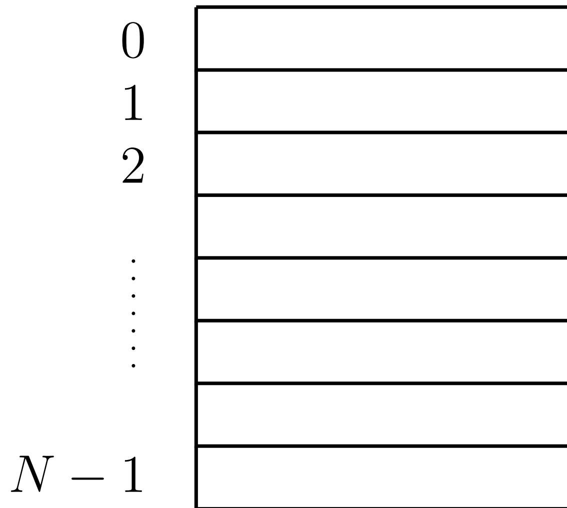
- Ensemble des instructions qu'un processeur peut exécuter.
- Défini par un *modèle d'exécution* : l'organisation générale des échanges d'information entre processeur, registres et mémoire.

Jeu d’instructions

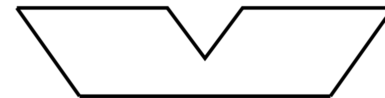
- Ensemble des instructions qu’un processeur peut exécuter.
- Défini par un *modèle d’exécution* : l’organisation générale des échanges d’information entre processeur, registres et mémoire.
- On distingue donc *différentes classes* de jeux d’instructions correspondant à des modes de spécification différents des opérandes dans une instruction de traitement par l’UAL.

Modèle d'exécution

Mémoire



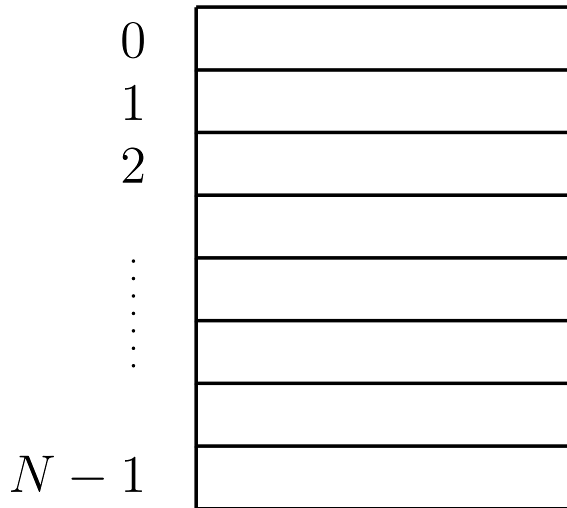
Processeur

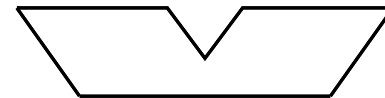


Modèle d'exécution

Mémoire

Processeur

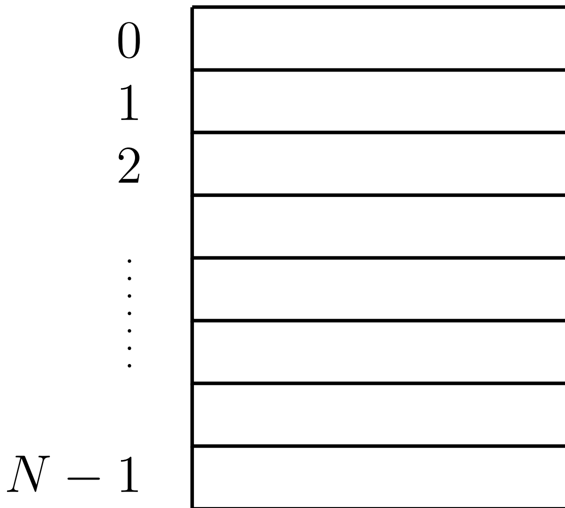




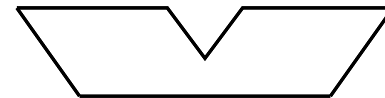
Modèle d’exécution

Mémoire

Processeur



mémoire-mémoire
mémoire-accumulateur
⟸ ? ⟹
mémoire-pile
mémoire-registres



Notion de registre

Les *registres* sont des éléments de mémoires associés au processeur.

- *Avantages :*

Notion de registre

Les *registres* sont des éléments de mémoires associés au processeur.

- *Avantages :*
  - plus rapides que la RAM

Notion de registre

Les *registres* sont des éléments de mémoires associés au processeur.

- *Avantages :*
  - plus rapides que la RAM
  - plus efficaces pour un compilateur, car le traffic mémoire est réduit

Notion de registre

Les *registres* sont des éléments de mémoires associés au processeur.

- *Avantages :*
  - plus rapides que la RAM
  - plus efficaces pour un compilateur, car le traffic mémoire est réduit
- *Inconvénients :*

Notion de registre

Les *registres* sont des éléments de mémoires associés au processeur.

- *Avantages :*
  - plus rapides que la RAM
  - plus efficaces pour un compilateur, car le traffic mémoire est réduit
- *Inconvénients :*
  - capacité moindre que la RAM

Notion de registre

Les *registres* sont des éléments de mémoires associés au processeur.

- *Avantages :*
  - plus rapides que la RAM
  - plus efficaces pour un compilateur, car le traffic mémoire est réduit
- *Inconvénients :*
  - capacité moindre que la RAM
  - plus chère

Principaux modèles d'exécution

- mémoire–mémoire : chaque opérande d'une instruction UAL peut être situé en mémoire.

Principaux modèles d'exécution

- mémoire–mémoire : chaque opérande d'une instruction UAL peut être situé en mémoire.
- mémoire–accumulateur : l'instruction UAL ne spécifie qu'un opérande ; les autres sont implicites, contenus dans l'accumulateur.

Principaux modèles d'exécution

- mémoire–mémoire : chaque opérande d'une instruction UAL peut être situé en mémoire.
- mémoire–accumulateur : l'instruction UAL ne spécifie qu'un opérande ; les autres sont implicites, contenus dans l'accumulateur.
- mémoire–pile : l'instruction UAL ne spécifie aucun opérande, et fait implicitement référence au sommet d'une pile.

Principaux modèles d'exécution

- mémoire–mémoire : chaque opérande d'une instruction UAL peut être situé en mémoire.
- mémoire–accumulateur : l'instruction UAL ne spécifie qu'un opérande ; les autres sont implicites, contenus dans l'accumulateur.
- mémoire–pile : l'instruction UAL ne spécifie aucun opérande, et fait implicitement référence au sommet d'une pile.
- registre–registre : l'instruction UAL ne spécifie aucun opérande en mémoire, et ne fait référence qu'à des registres.

Notation

Tout modèle d'exécution est caractérisé par un couple

$$(m, n)$$

avec :

- $m \leq n$;

Notation

Tout modèle d'exécution est caractérisé par un couple

$$(m, n)$$

avec :

- $m \leq n$;
- $n =$ nombre total d'opérandes spécifiés par une instruction UAL ;

Notation

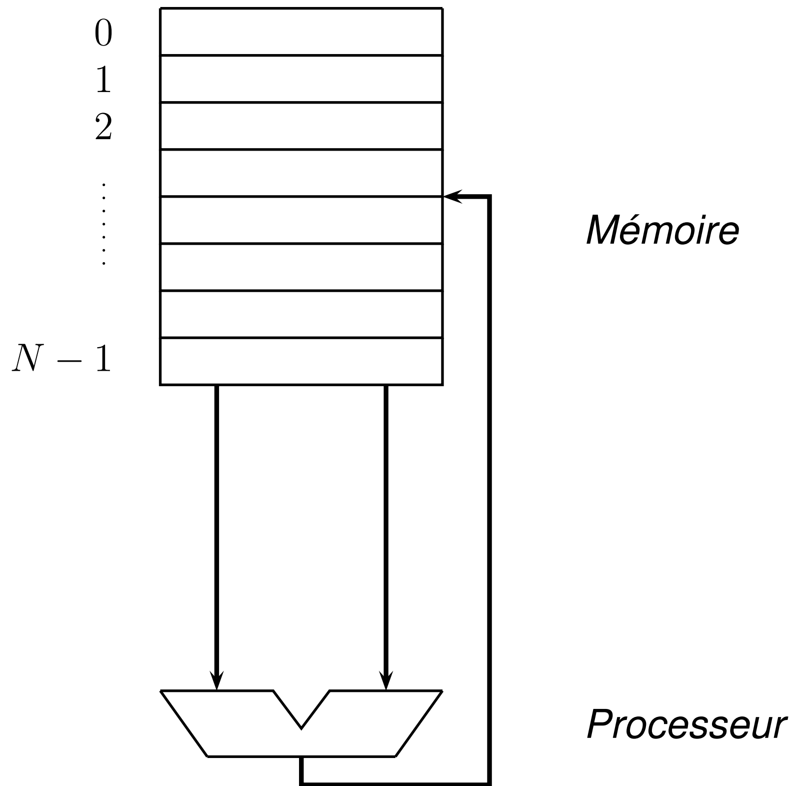
Tout modèle d'exécution est caractérisé par un couple

$$(m, n)$$

avec :

- $m \leq n$;
- $n =$ nombre total d'opérandes spécifiés par une instruction UAL ;
- $m =$ nombre d'opérandes mémoire autorisés dans une instruction UAL.

Modèle mémoire/mémoire (3,3)



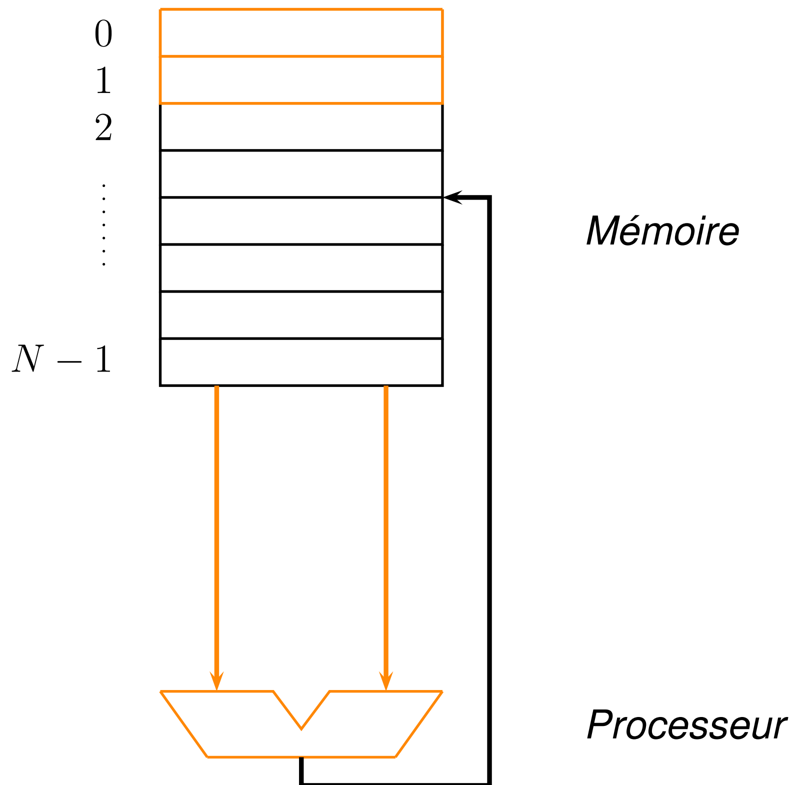
Mémoire

Processeur

$C = A + B$:

```
ADD @C, @A, @B;
```

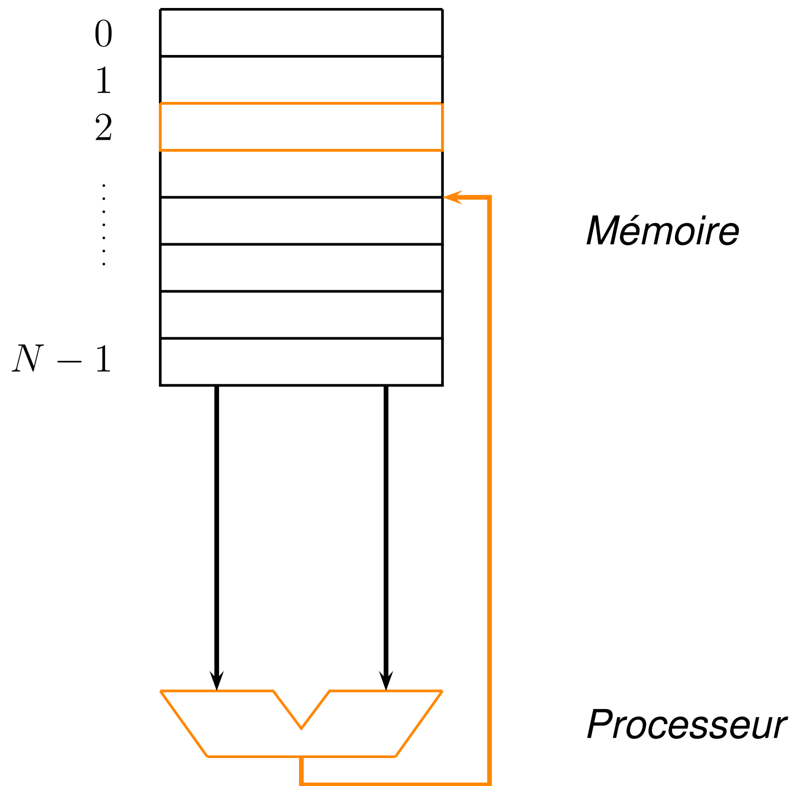
Modèle mémoire/mémoire (3,3)



$C = A + B$:

```
ADD @C, @A, @B;
```

Modèle mémoire/mémoire (3,3)



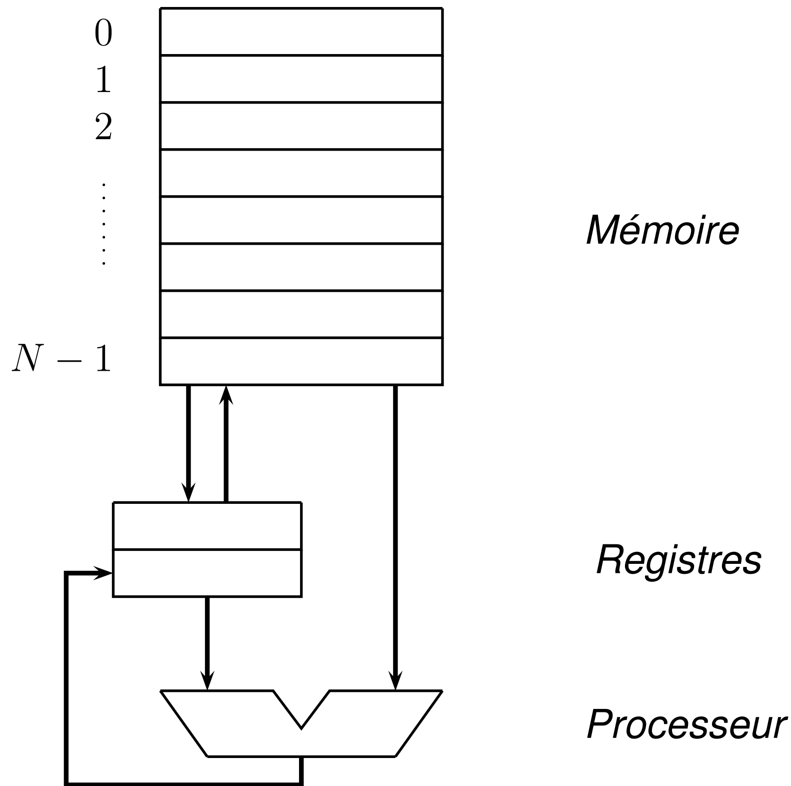
Mémoire

Processeur

$C = A + B$:

```
ADD @C, @A, @B;
```

Modèle mémoire/registres (1,2)



Mémoire

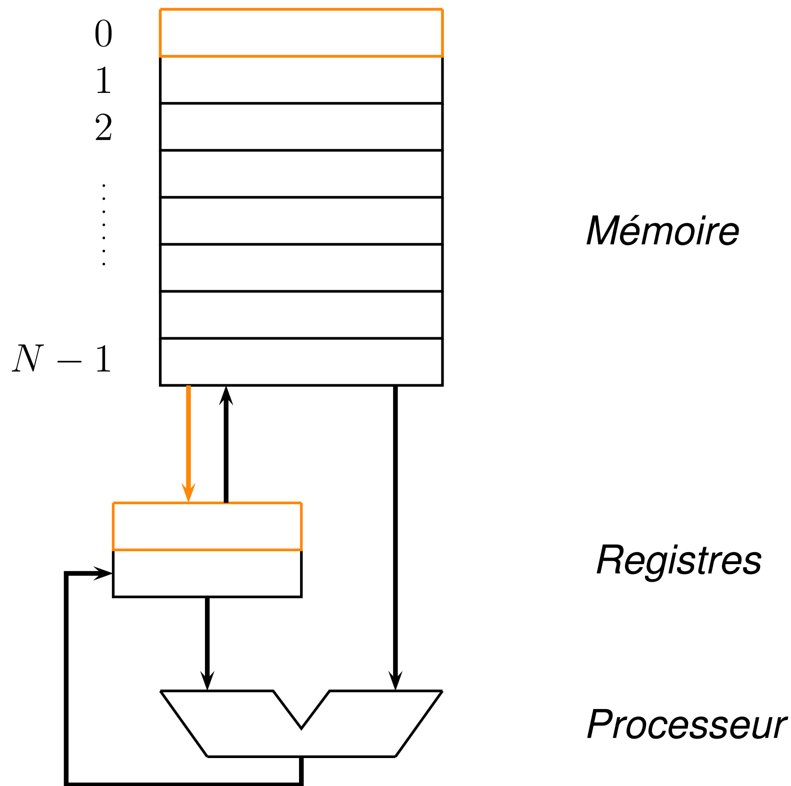
Registres

Processeur

$C = A + B$:

```
LOAD R1, @A;
ADD R1, @B;
STORE R1, @C;
```

Modèle mémoire/registres (1,2)



Mémoire

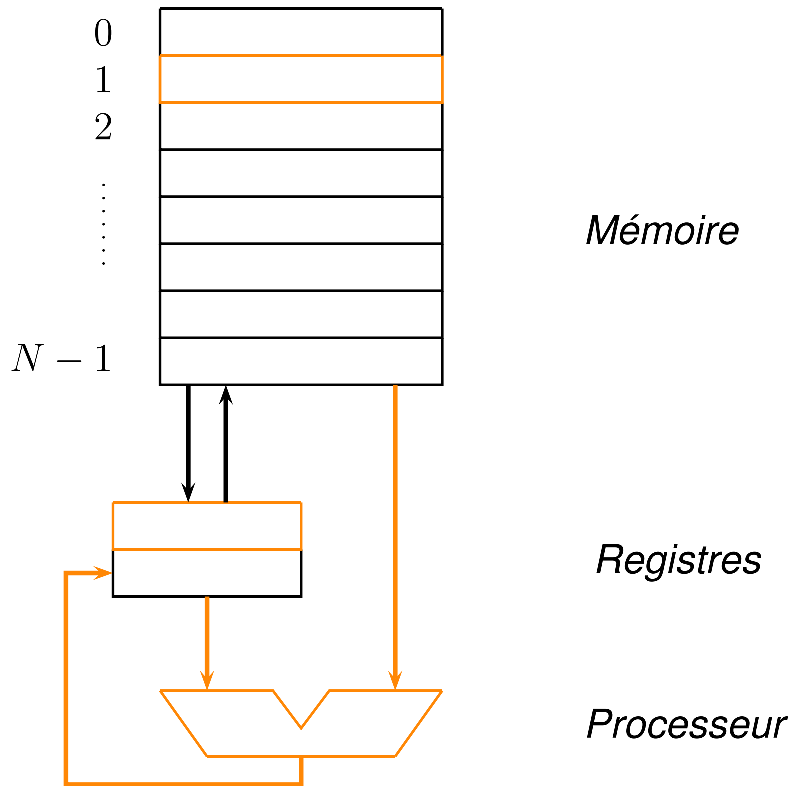
Registres

Processeur

$C = A + B$:

```
LOAD R1, @A;
ADD R1, @B;
STORE R1, @C;
```

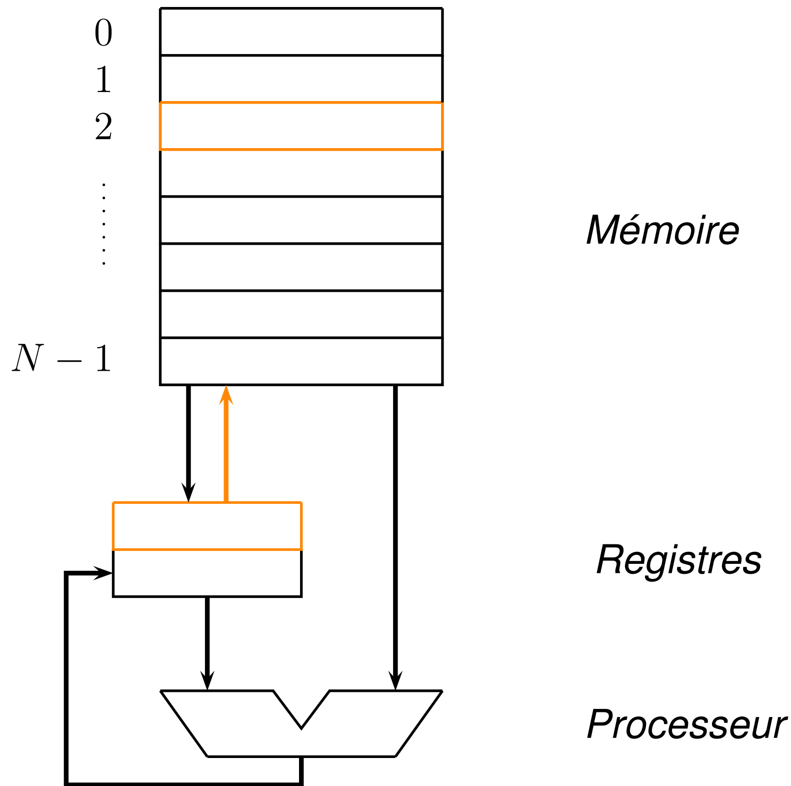
Modèle mémoire/registres (1,2)



$C = A + B$:

```
LOAD R1, @A;
ADD R1, @B;
STORE R1, @C;
```

Modèle mémoire/registres (1,2)



Mémoire

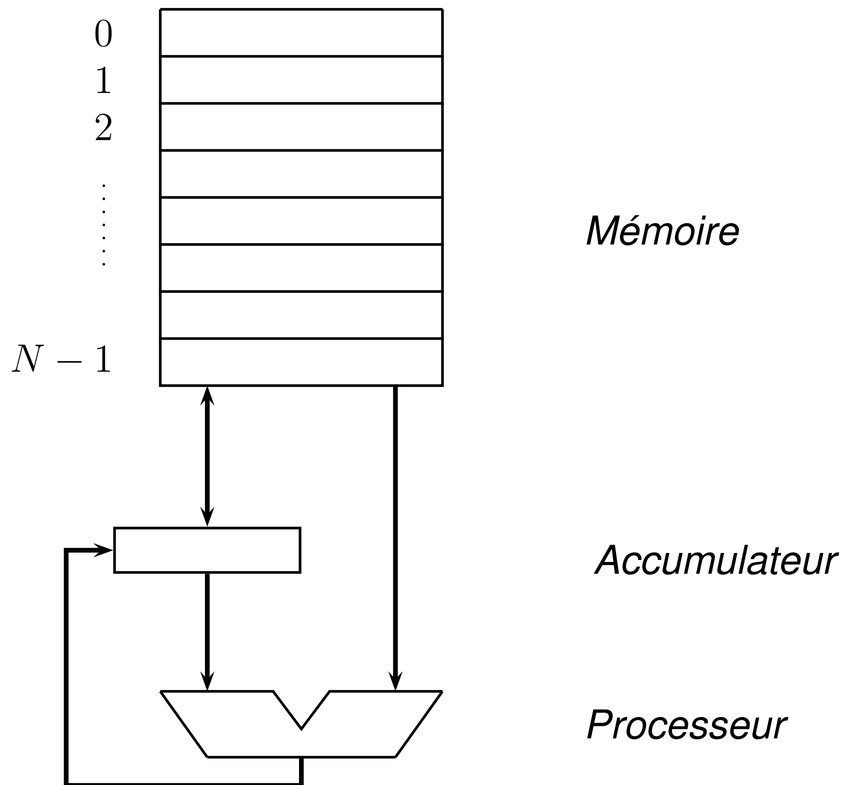
Registres

Processeur

$$C = A + B:$$

```
LOAD R1, @A;
ADD R1, @B;
STORE R1, @C;
```

Modèle mémoire/accumulateur (1,1)



Mémoire

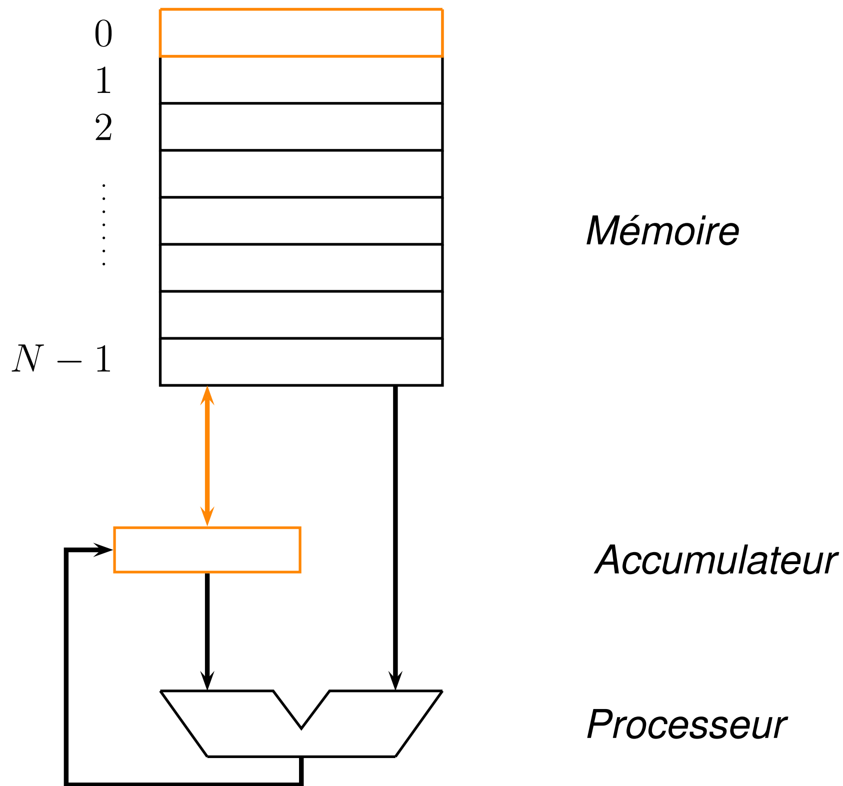
Accumulateur

Processeur

$C = A + B$:

```
LOAD @A;
ADD @B;
STORE @C;
```

Modèle mémoire/accumulateur (1,1)



Mémoire

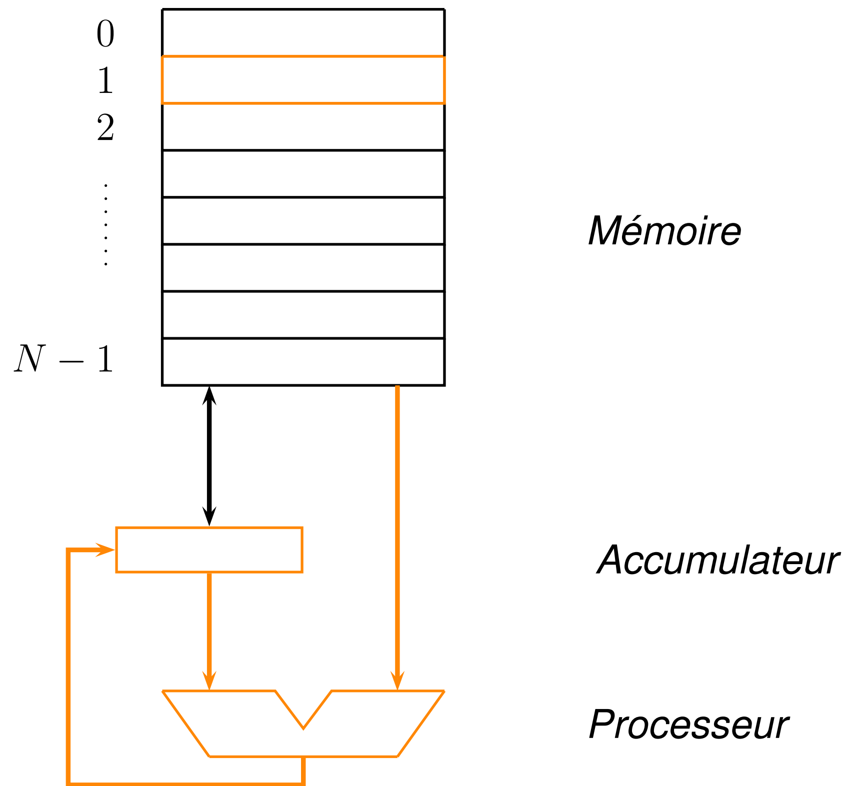
Accumulateur

Processeur

$C = A + B$:

```
LOAD @A;
ADD @B;
STORE @C;
```

Modèle mémoire/accumulateur (1,1)



Mémoire

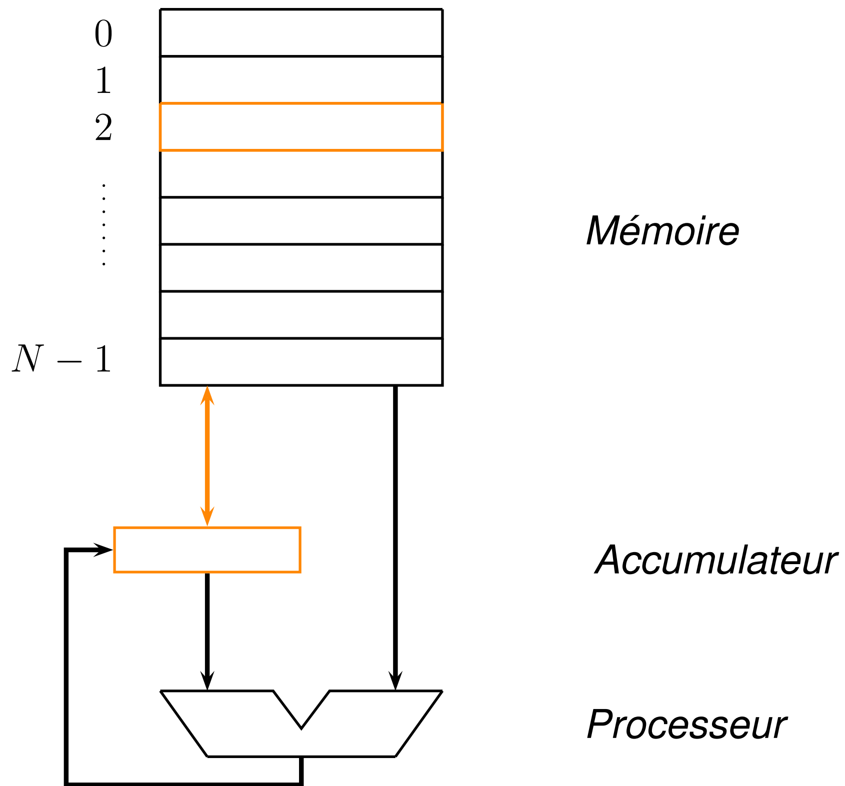
Accumulateur

Processeur

$C = A + B$:

```
LOAD @A;
ADD @B;
STORE @C;
```

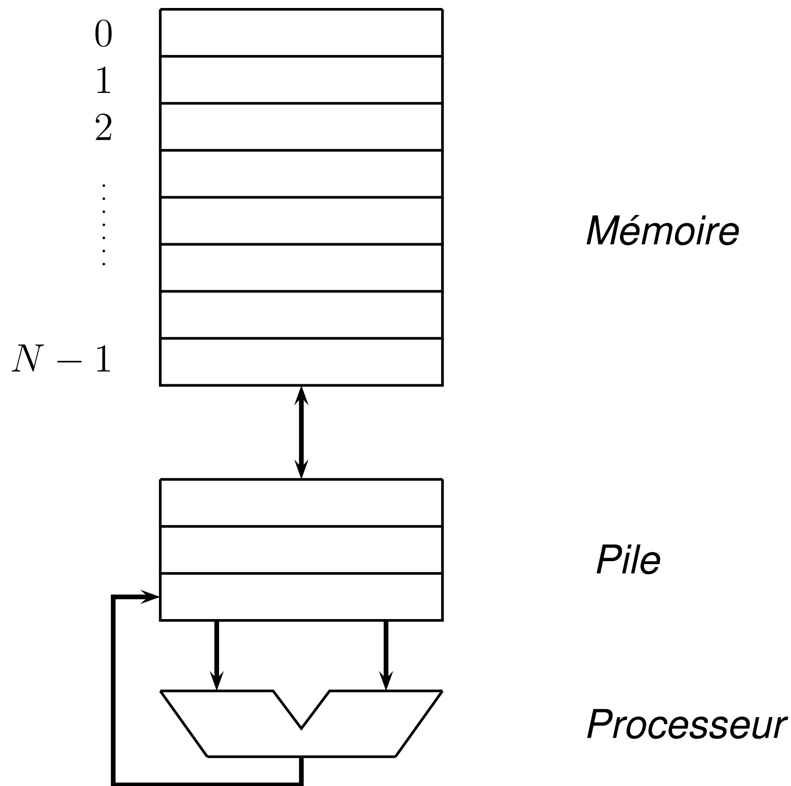
Modèle mémoire/accumulateur (1,1)



$$C = A + B :$$

```
LOAD @A;
ADD @B;
STORE @C;
```

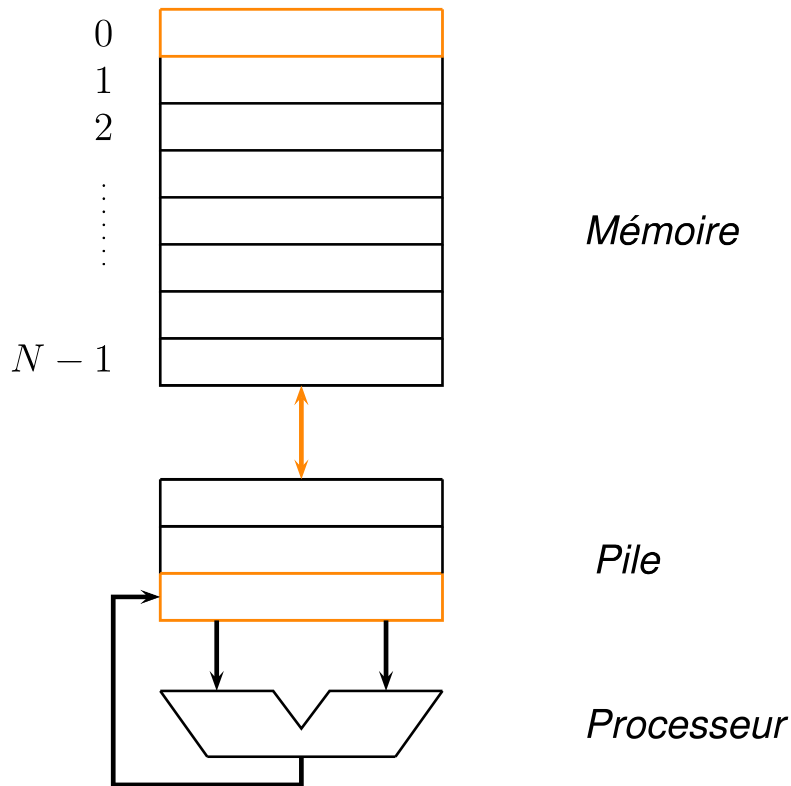
Modèle pile (0,0)



$C = A + B$:

```
PUSH @A;
PUSH @B;
ADD ;
POP @C;
```

Modèle pile (0,0)



Mémoire

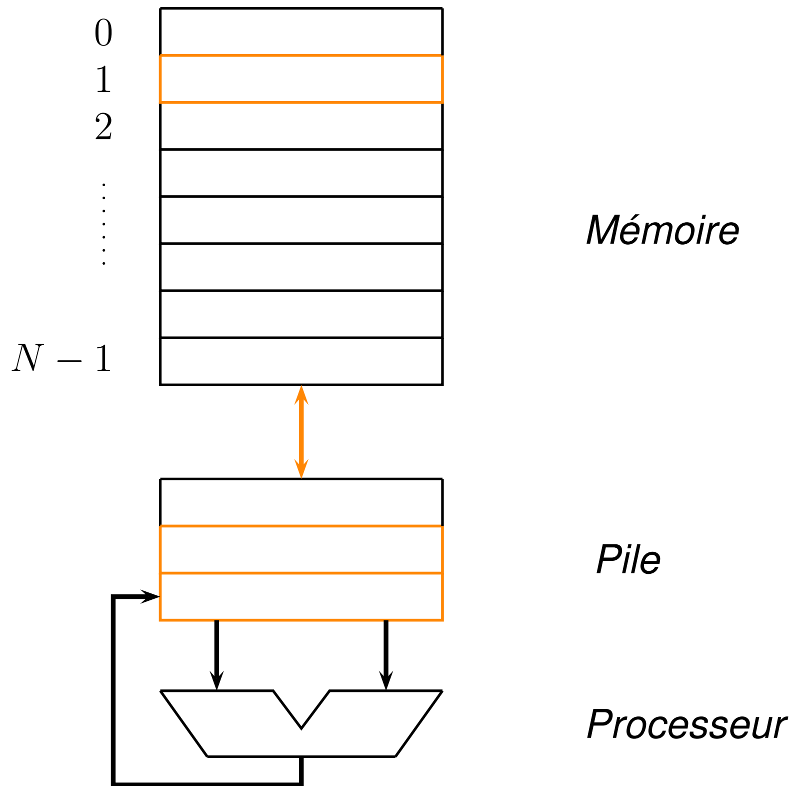
Pile

Processeur

$C = A + B$:

```
PUSH @A;
PUSH @B;
ADD ;
POP @C;
```

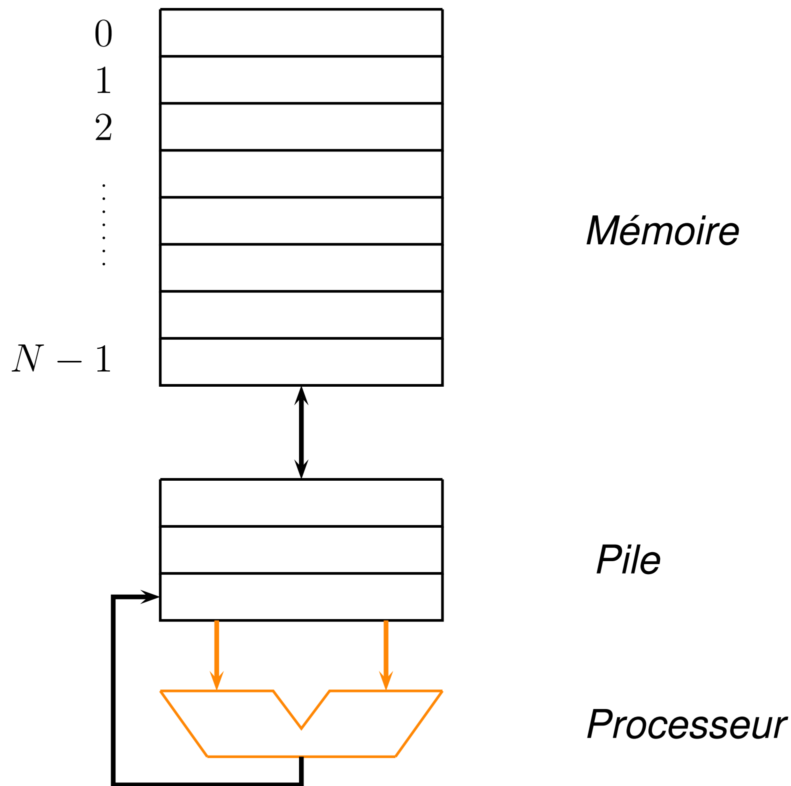
Modèle pile (0,0)



$C = A + B$:

```
PUSH @A;
PUSH @B;
ADD ;
POP @C;
```

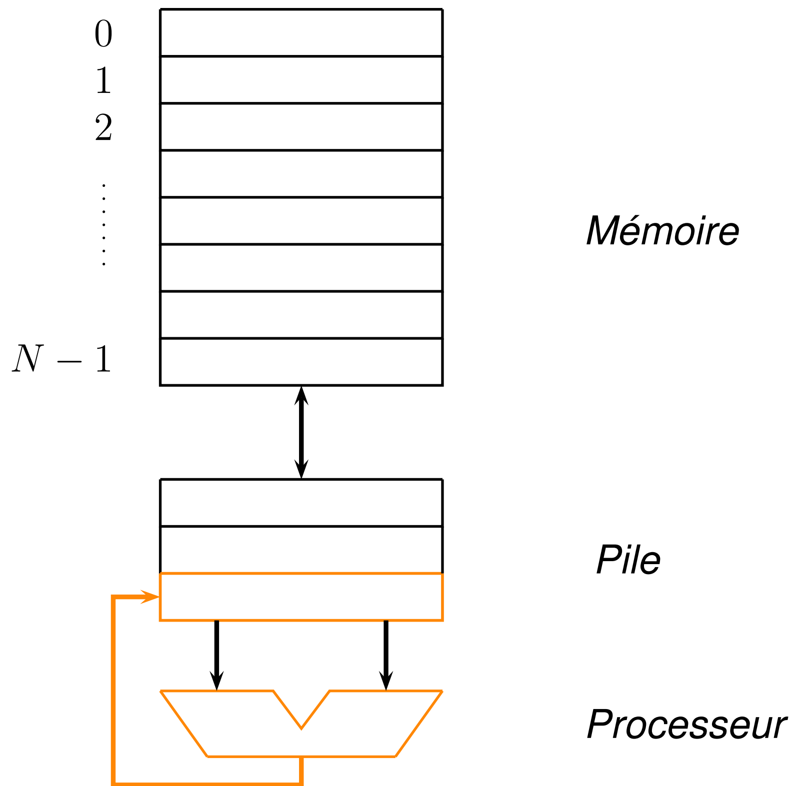
Modèle pile (0,0)



$C = A + B$:

```
PUSH @A;
PUSH @B;
ADD ;
POP @C;
```

Modèle pile (0,0)



Mémoire

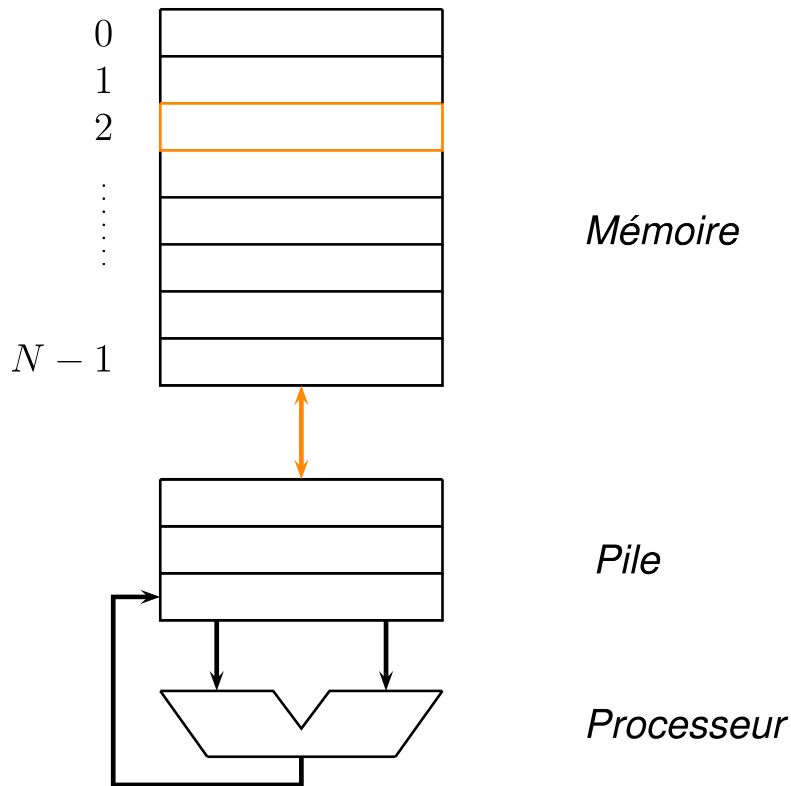
Pile

Processeur

$C = A + B$:

```
PUSH @A;
PUSH @B;
ADD ;
POP @C;
```

Modèle pile (0,0)



Mémoire

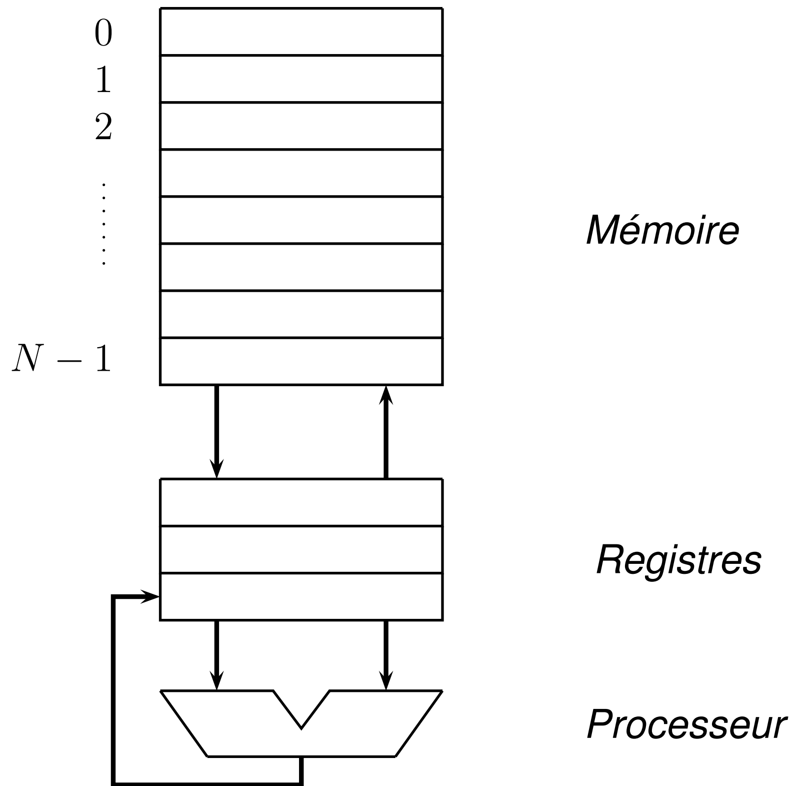
Pile

Processeur

$C = A + B$:

```
PUSH @A;
PUSH @B;
ADD ;
POP @C;
```

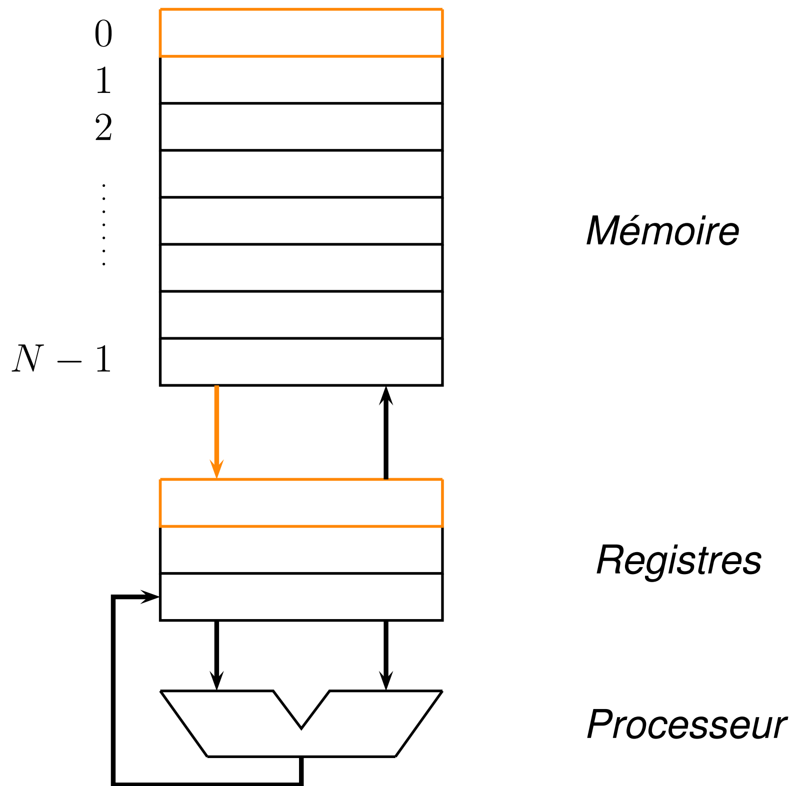
Modèle registre/registre (0,3)



$C = A + B$:

```
LOAD R1, @A;
LOAD R2, @B;
ADD R3, R1, R2;
STORE R3, @C
```

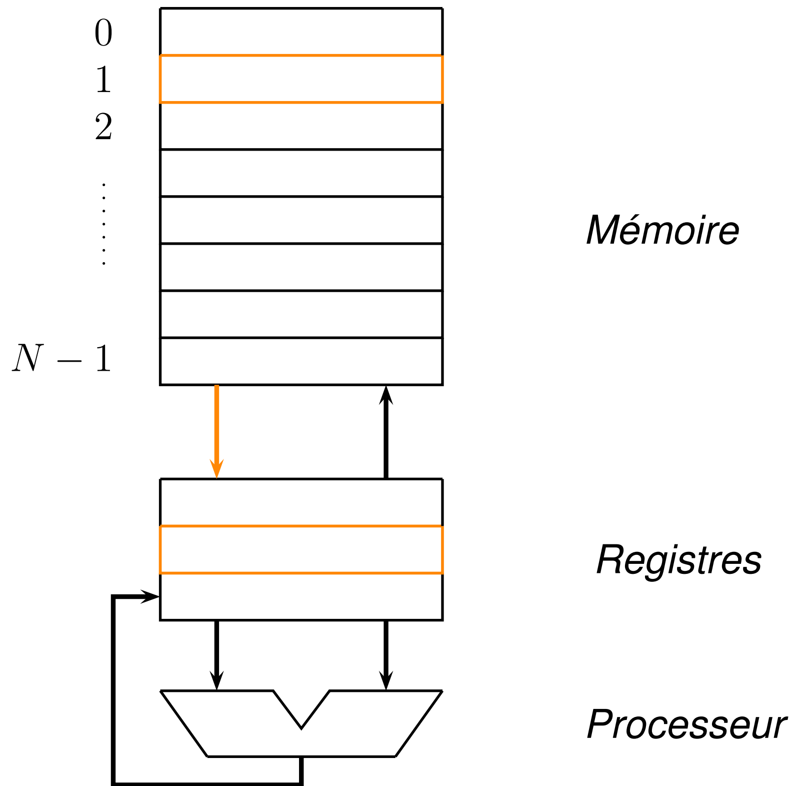
Modèle registre/registre (0,3)



$C = A + B$:

```
LOAD R1, @A;
LOAD R2, @B;
ADD R3, R1, R2;
STORE R3, @C
```

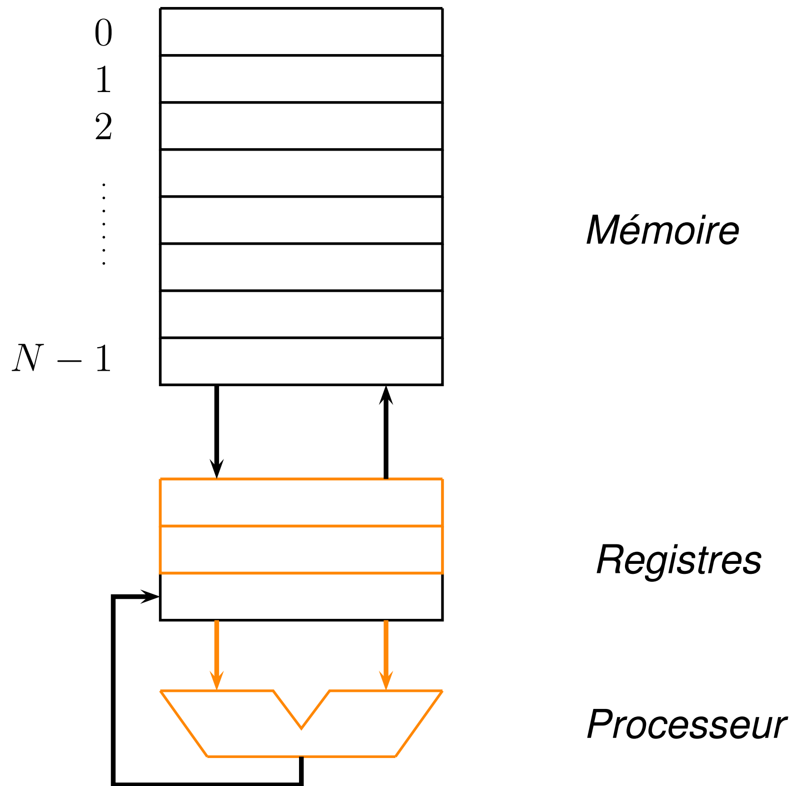
Modèle registre/registre (0,3)



$C = A + B$:

```
LOAD R1, @A;
LOAD R2, @B;
ADD R3, R1, R2;
STORE R3, @C
```

Modèle registre/registre (0,3)



Mémoire

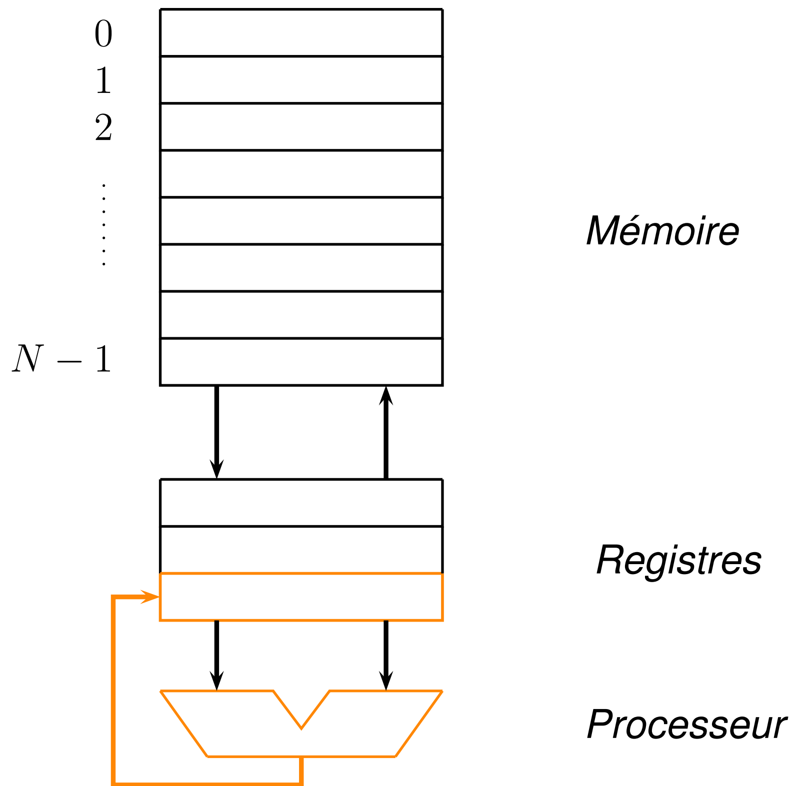
Registres

Processeur

$C = A + B$:

```
LOAD R1, @A;
LOAD R2, @B;
ADD R3, R1, R2;
STORE R3, @C
```

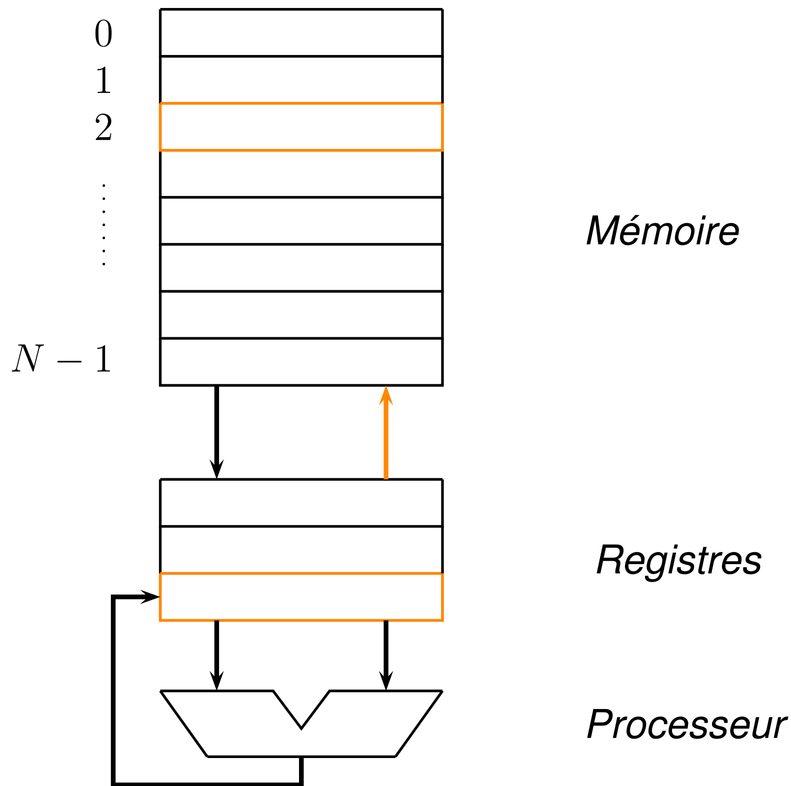
Modèle registre/registre (0,3)



$C = A + B$:

```
LOAD R1, @A;
LOAD R2, @B;
ADD R3, R1, R2;
STORE R3, @C
```

Modèle registre/registre (0,3)



$C = A + B$:

```
LOAD R1, @A;
LOAD R2, @B;
ADD R3, R1, R2;
STORE R3, @C
```

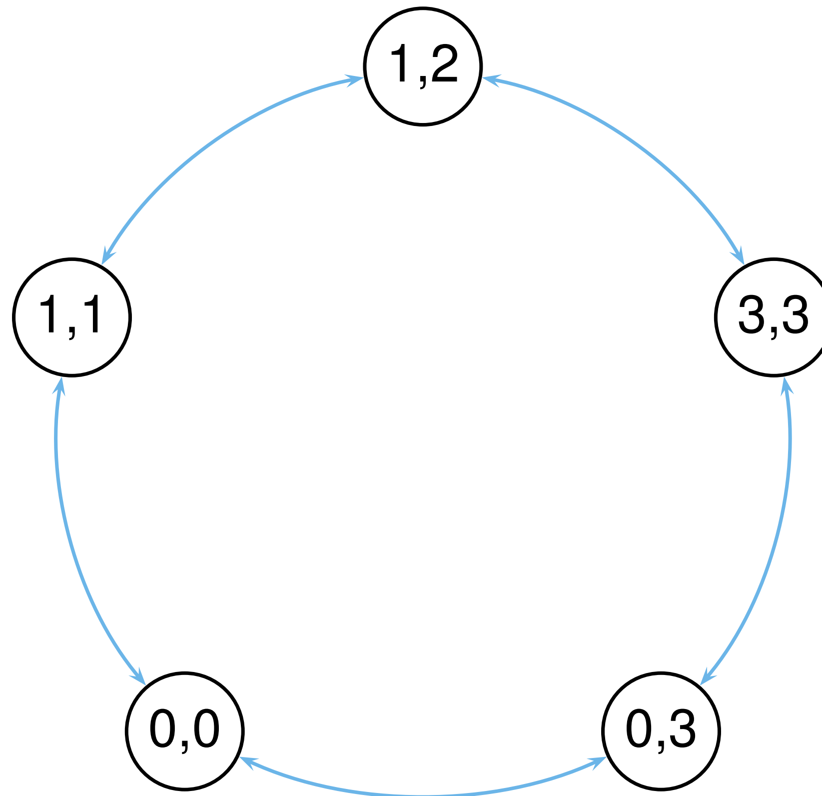
Jeux “équivalents”. . .

(3, 3)	(1, 2)	(1, 1)	(0, 0)	(0, 3)

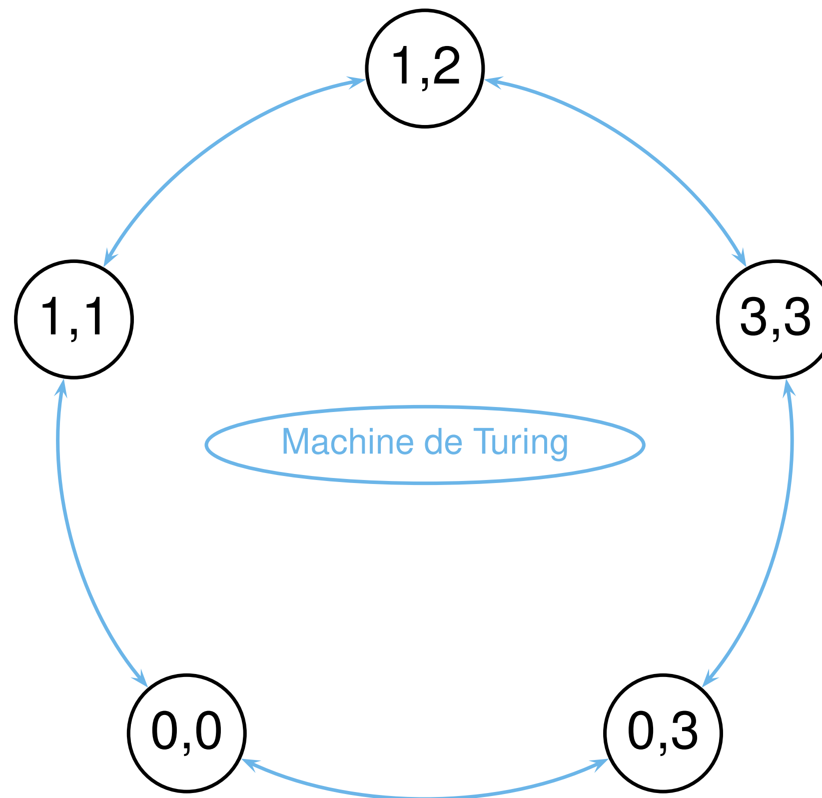
`ADD @C, @A, @B;`	`LOAD R1, @A;`
	`ADD R1, @B;`
	`STORE R1, @C;`
		`LOAD @A;`
		`ADD @B;`
		`STORE @C;`
			`PUSH @A;`
			`PUSH @B;`
			`ADD ;`
			`POP @C;`
				`LOAD R1, @A;`
				`LOAD R2, @B;`
				`ADD R3, R1, R2;`
				`STORE R3, @C`

Chacun de ces modèles d’exécution effectue l’instruction $C = A + B$, et par conséquent, *simule* les autres modèles.

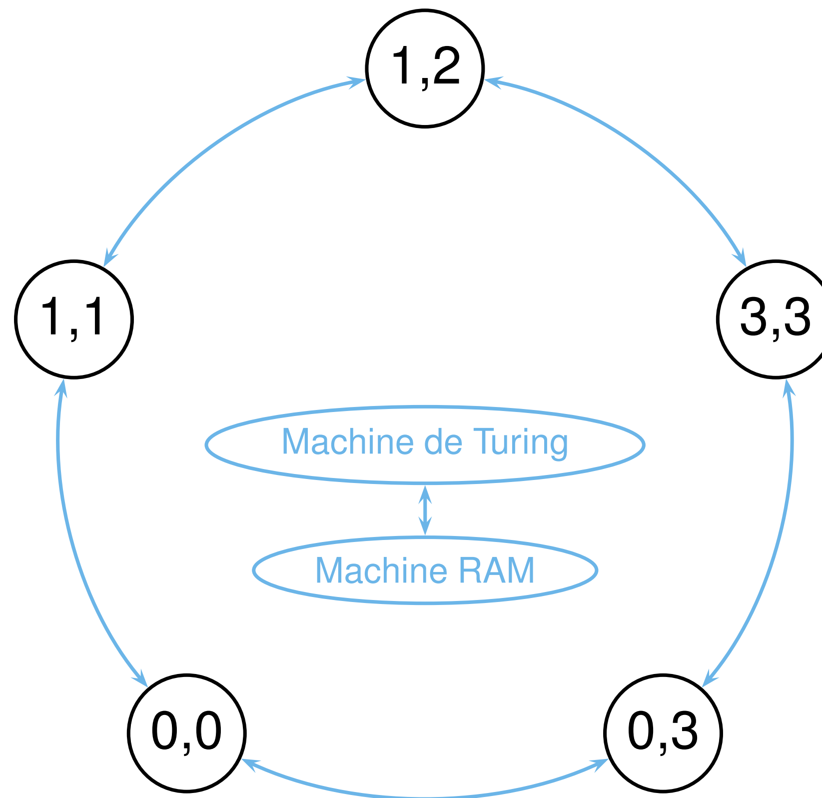
Simulations inter–modèles



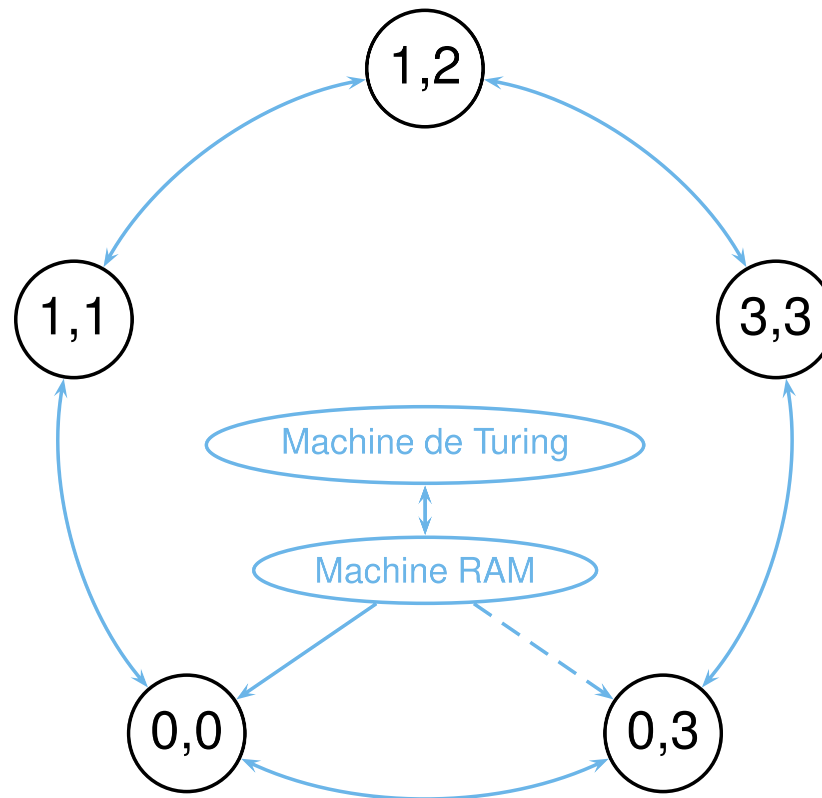
Simulations inter–modèles



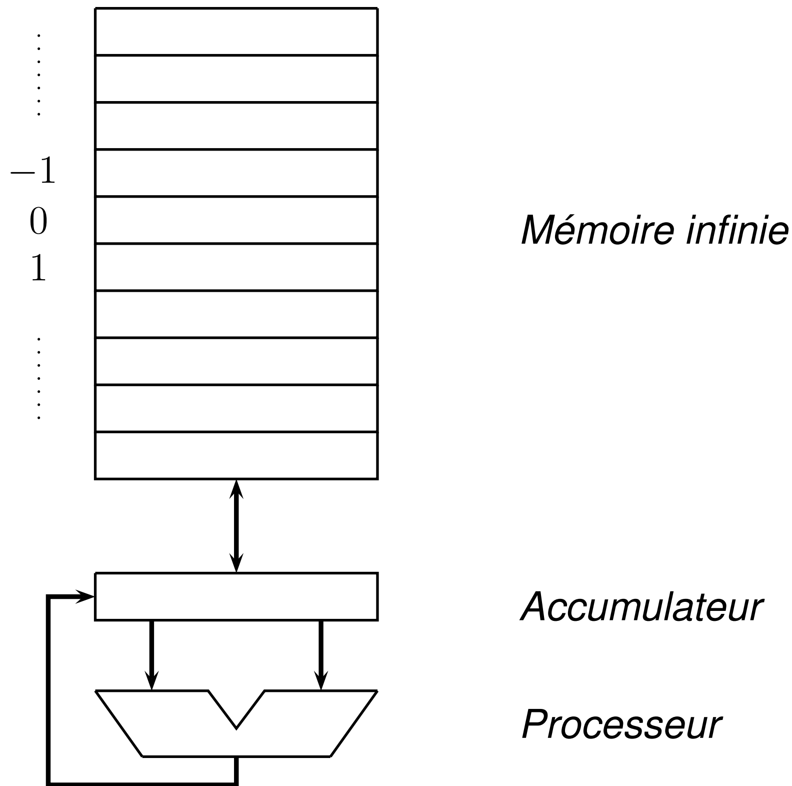
Simulations inter–modèles



Simulations inter–modèles

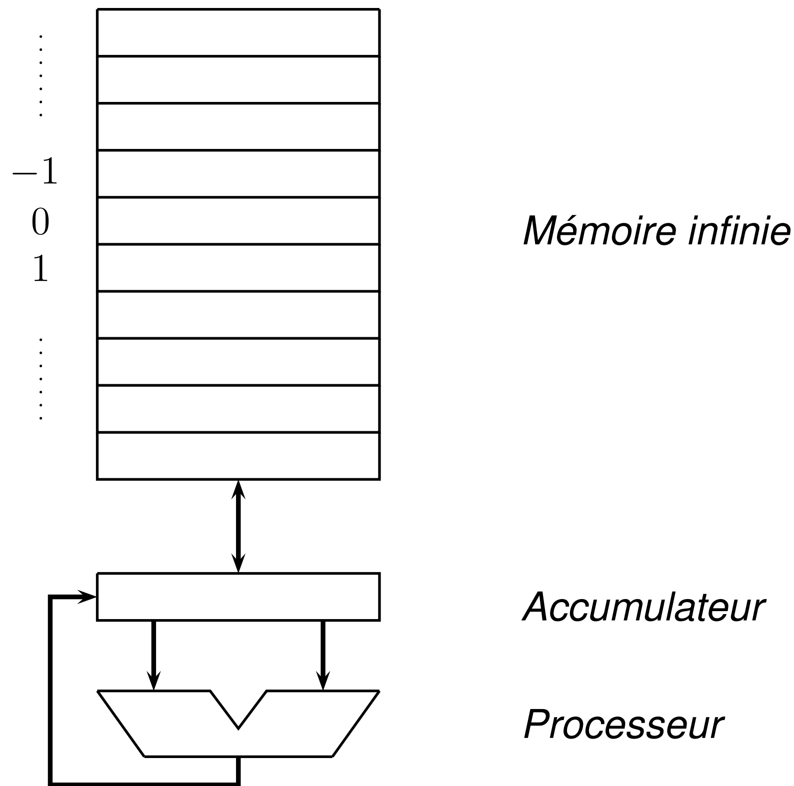


Machine RAM



Random Access Machine : machine dotée d'un processeur rudimentaire (incrémentation, décrémentation, comparaison simple), d'un registre unique (accumulateur), et d'une mémoire infinie ne codant que des entiers naturels...

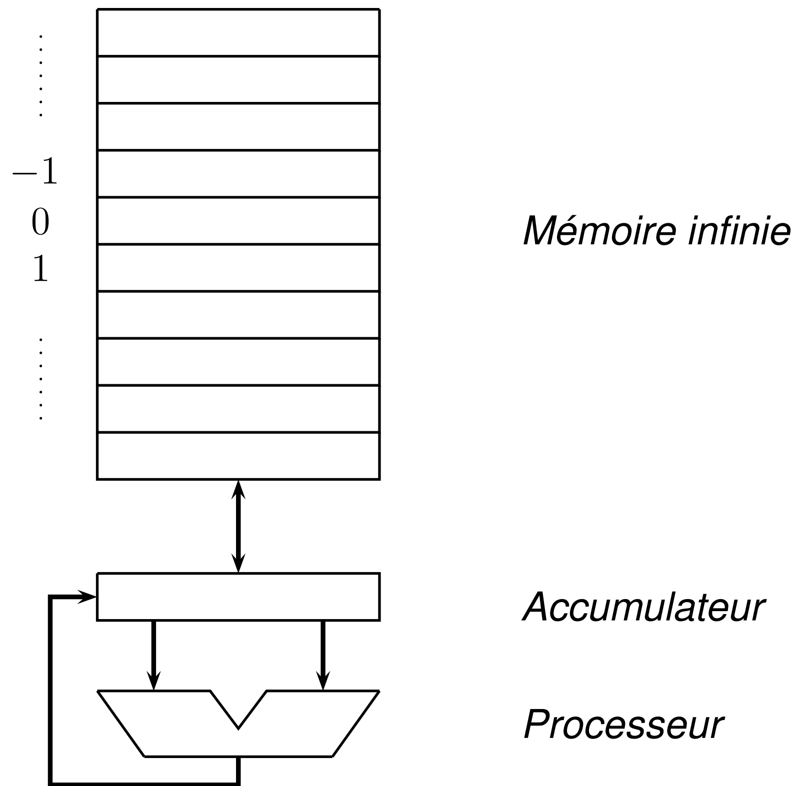
Machine RAM



Random Access Machine : machine dotée d'un processeur rudimentaire (incrémentation, décrémentation, comparaison simple), d'un registre unique (accumulateur), et d'une mémoire infinie ne codant que des entiers naturels...

→ machine (0,0) virtuelle à registre unique et à mémoire infinie

Machine RAM



Random Access Machine : machine dotée d'un processeur rudimentaire (incrémentation, décrémentation, comparaison simple), d'un registre unique (accumulateur), et d'une mémoire infinie ne codant que des entiers naturels...

→ machine (0,0) virtuelle à registre unique et à mémoire infinie
→ aussi vue comme un cas particulier de machine (0,3) virtuelle.

Jeu d'instructions RAM

Chargements :

LOAD# <valeur numérique> ;

LOAD@ <adresse mémoire> ;

LOAD@ [<adresse mémoire>] ;

Rangements :

STORE@ <adresse mémoire> ;

STORE@[<adresse mémoire>] ;

Jeu d'instructions RAM

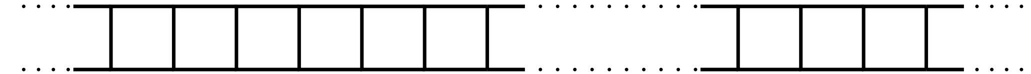
Opérations :

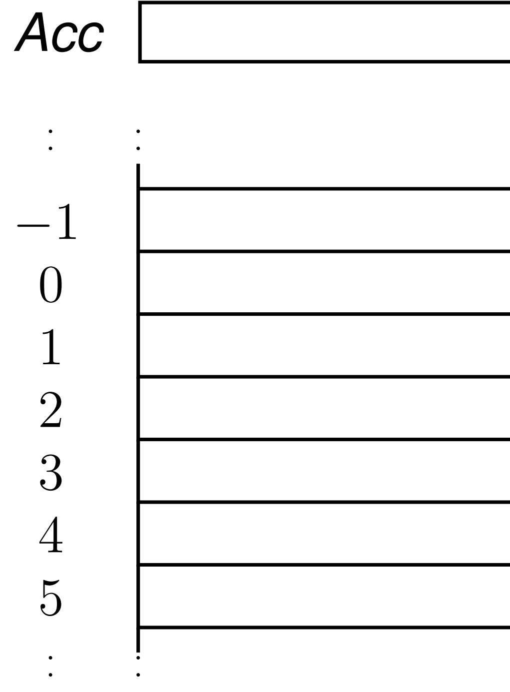
```
INCR ;
DECR ;
```

Instructions de contrôle :

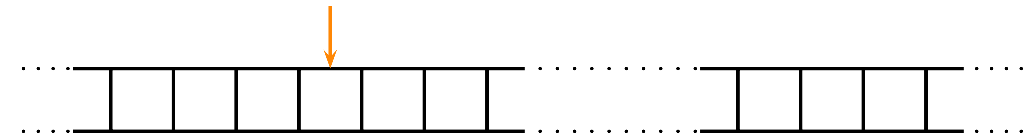
JUMP <étiquette> ;
JLTZ <étiquette> ;
HALT ;

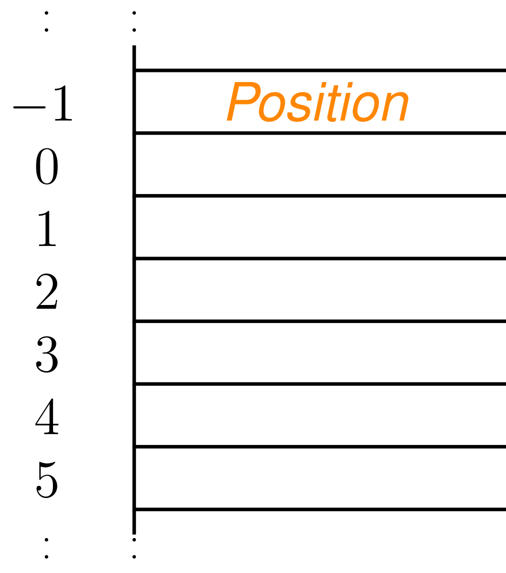
Equivalence Machine de Turing - Machine RAM



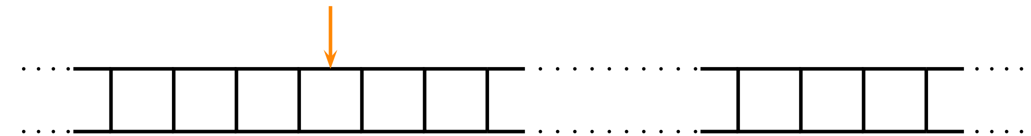


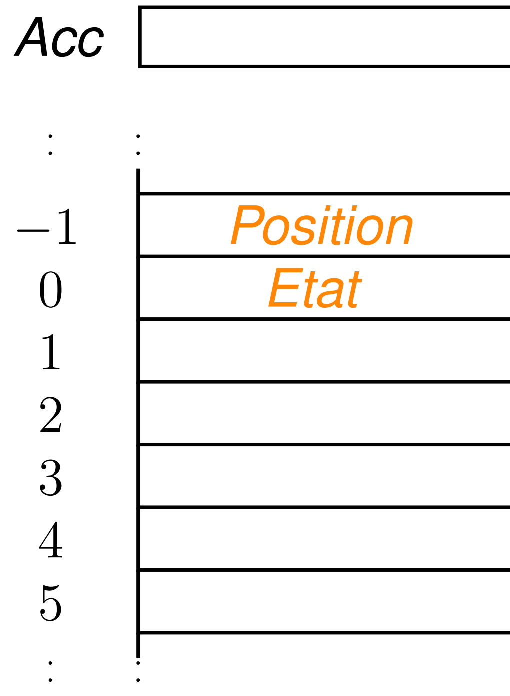
Equivalence Machine de Turing - Machine RAM



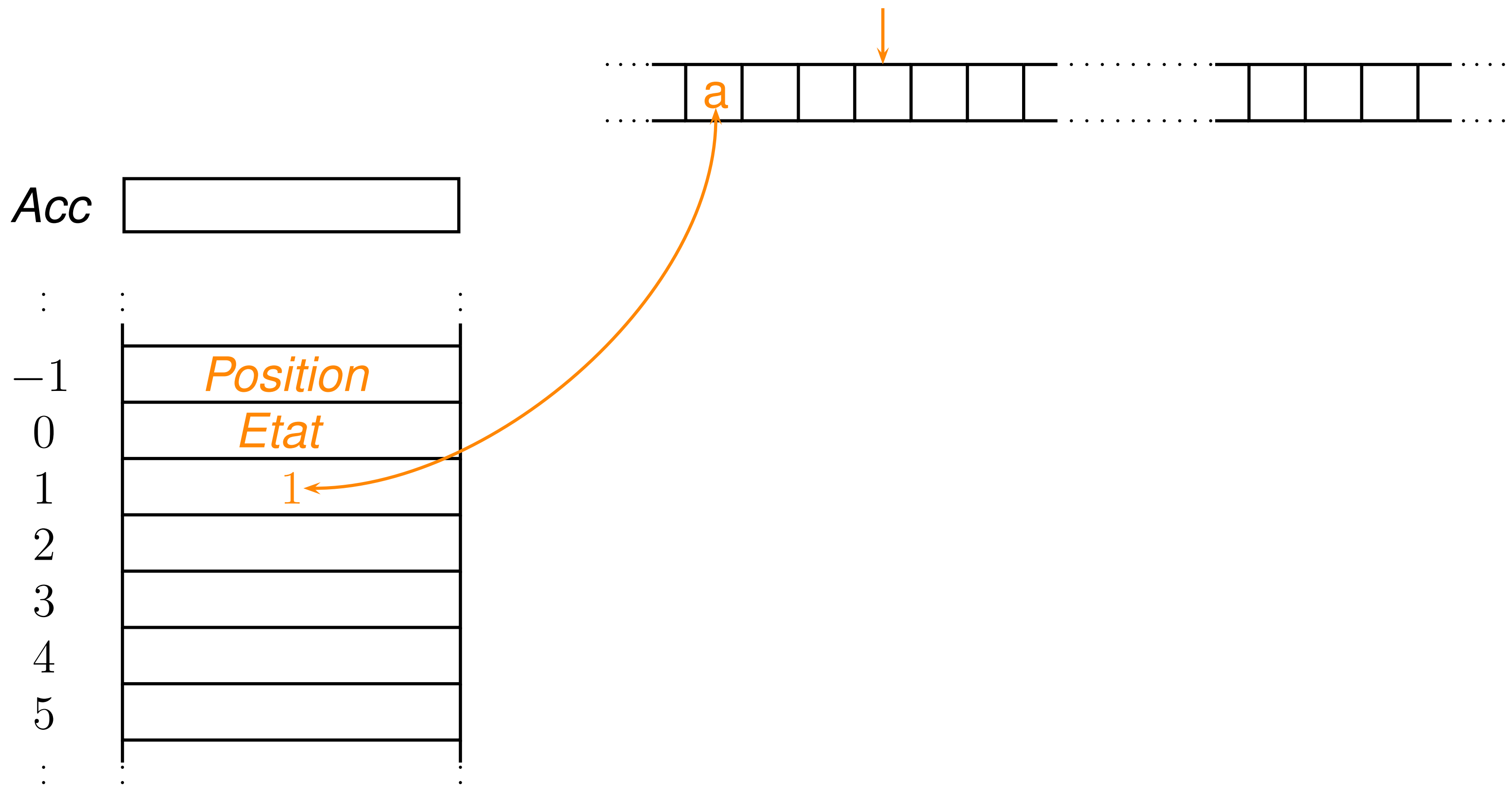


Equivalence Machine de Turing - Machine RAM





Equivalence Machine de Turing - Machine RAM



Equivalence Machine de Turing - Machine RAM

a c

Acc

-1	*Position*
0	*Etat*
1	1
2	3
3	
4	
5	

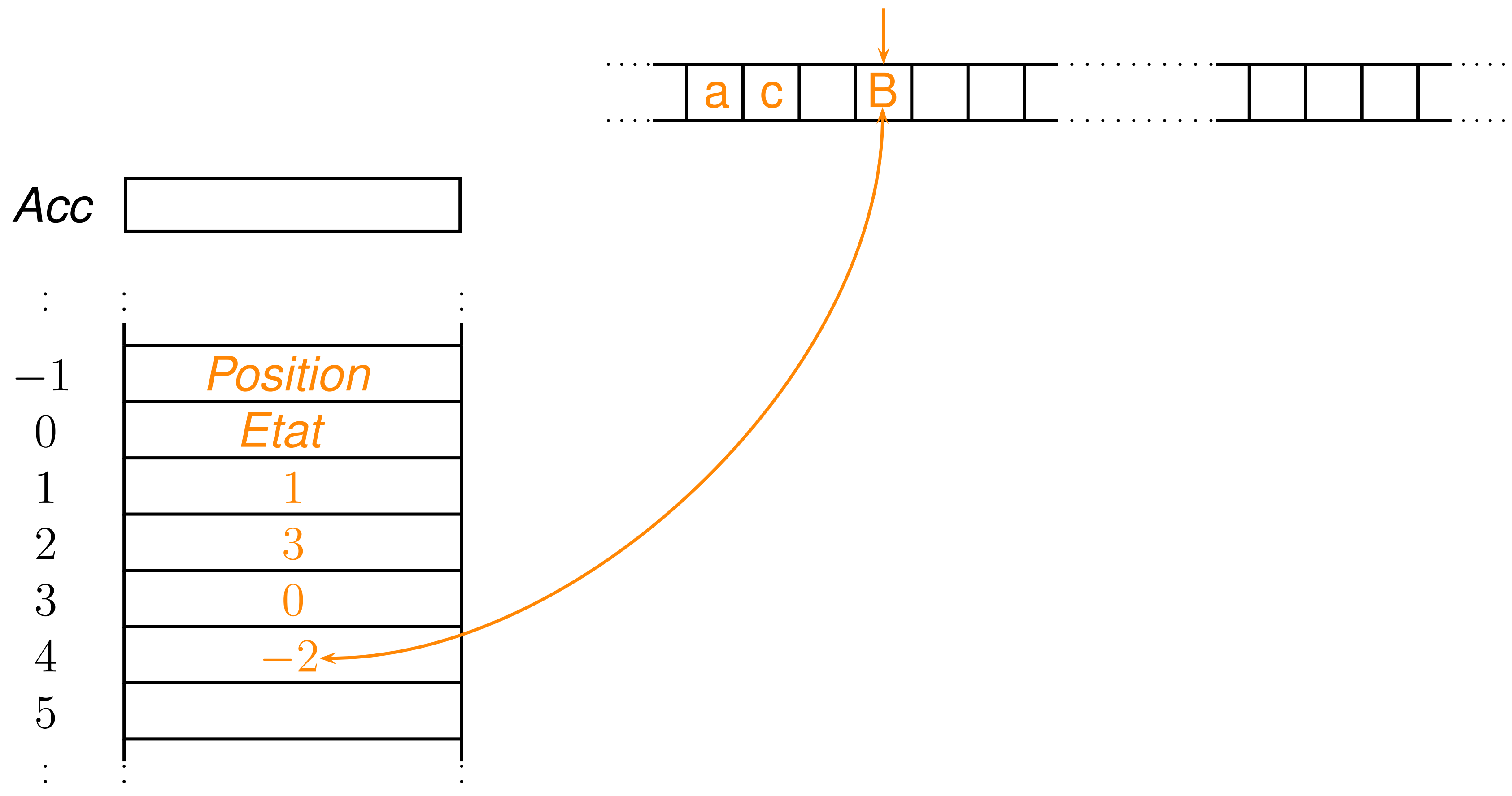
Equivalence Machine de Turing - Machine RAM

a c

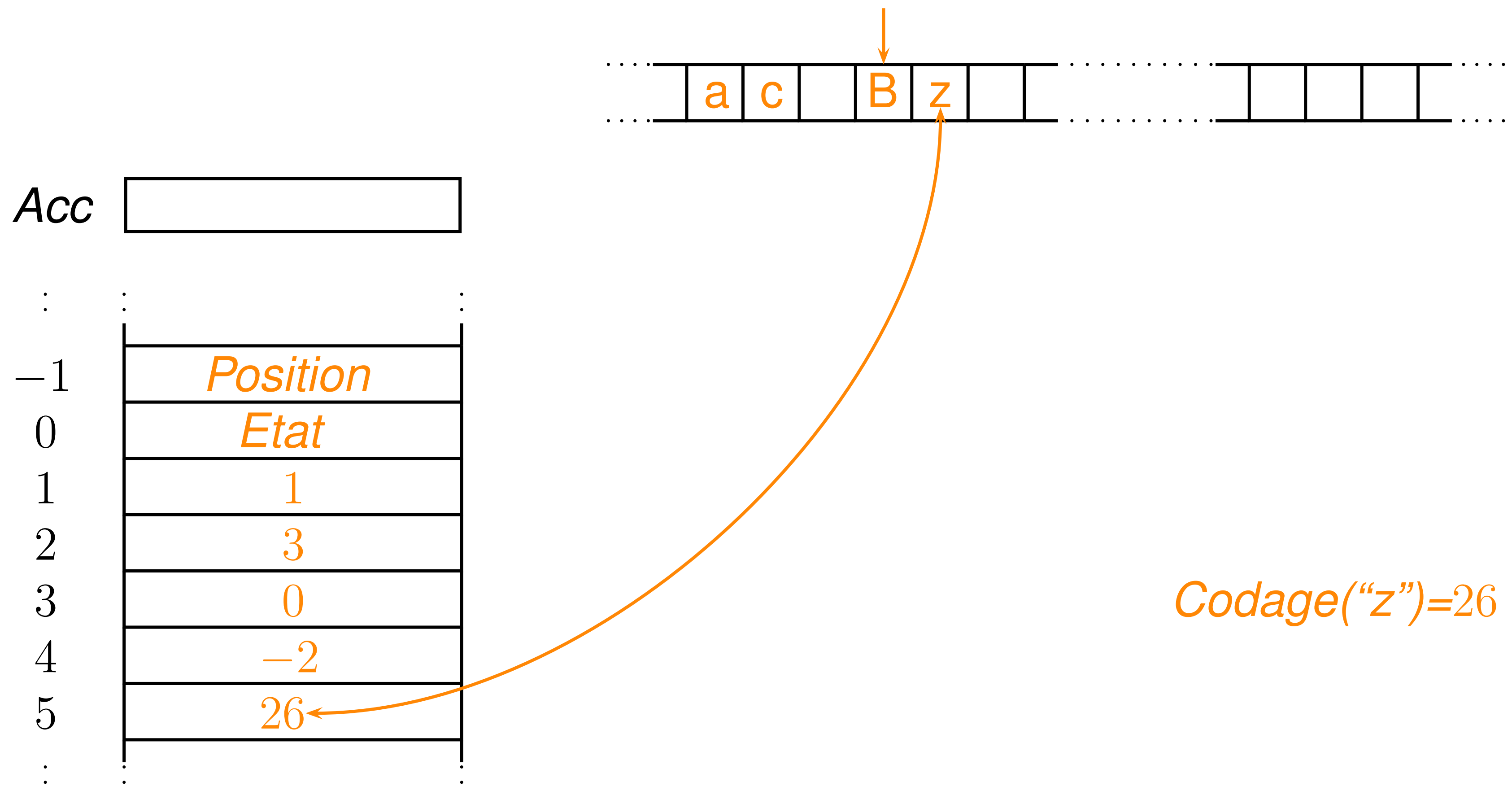
Acc

-1	*Position*
0	*Etat*
1	1
2	3
3	0
4	
5	

Equivalence Machine de Turing - Machine RAM



Equivalence Machine de Turing - Machine RAM



Simulation Turing -> RAM

A chaque état de la machine de Turing est associé un morceau de programme de la machine RAM qui simule le comportement de la machine de Turing. Par exemple :

q_0 a $\rightarrow$ q_1 b D;
q_0 b $\rightarrow$ q_1 c G;
q_0 # $\rightarrow$ q_0 # D;

```
q0    :  LOAD@ [-1];
         JLTZ q0#;
         DECR;
         JLTZ q0a;
         DECR;
         JLTZ q0b;
stop  :  HALT;
         LOAD@ -1;
         INCR;
         STORE@ -1;
         JUMP q0;
q0a   :  LOAD# 2;
         STORE@ [-1];
         LOAD@ -1;
         INCR;
         STORE@ -1;
         JUMP q1;
```

Simulation RAM -> Turing

La machine RAM est simulée par une machine de Turing comportant deux marqueurs : le premier marqueur délimite une zone du ruban contenant les données en mémoire, sous forme binaire. A chaque instruction du programme est associée une partie de la machine de Turing qui simule le fonctionnement de cette instruction. Le numéro des registres est écrit sur la bande à partir du second marqueur.

Jeux “équivalents” (suite)

(3, 3)	(1, 3)	(1, 1)	(0, 0)	(0, 3)

`ADD @C, @A, @B;`	`LOAD R1, @A;`
	`ADD R1, @B;`
	`STORE R1, @C;`
		`LOAD @A;`
		`ADD @B;`
		`STORE @C;`
			`PUSH @A;`
			`PUSH @B;`
			`ADD ;`
			`POP @C;`
				`LOAD R1, @A;`
				`LOAD R2, @B;`
				`ADD R3, R1, R2;`
				`STORE R3, @C`

Parmi ces modèles, y en a t’il un *meilleur* ?

Machines mémoire/mémoire (3,3)

- Avantages :

Machines mémoire/mémoire (3,3)

- Avantages :
  - excellente densité d'instructions

Machines mémoire/mémoire (3,3)

- Avantages :
  - excellente densité d'instructions
  - ne gaspille pas de registres pour des données temporaires

Machines mémoire/mémoire (3,3)

- Avantages :
  - excellente densité d'instructions
  - ne gaspille pas de registres pour des données temporaires
- Inconvénients :

Machines mémoire/mémoire (3,3)

- Avantages :
  - excellente densité d'instructions
  - ne gaspille pas de registres pour des données temporaires
- Inconvénients :
  - grande variation de la taille des instructions

Machines mémoire/mémoire (3,3)

- Avantages :
  - excellente densité d’instructions
  - ne gaspille pas de registres pour des données temporaires
- Inconvénients :
  - grande variation de la taille des instructions
  - grande variation du travail effectué par instruction

Machines mémoire/mémoire (3,3)

- Avantages :
  - excellente densité d’instructions
  - ne gaspille pas de registres pour des données temporaires
- Inconvénients :
  - grande variation de la taille des instructions
  - grande variation du travail effectué par instruction
  - les accés mémoire créent un goulot d’étranglement

Machines registre/mémoire (1,2)

- Avantages :

Machines registre/mémoire (1,2)

- Avantages :
  - des données sont accessibles sans chargement préalable

Machines registre/mémoire (1,2)

- Avantages :
  - des données sont accessibles sans chargement préalable
  - facilite le codage et une assez bonne densité

Machines registre/mémoire (1,2)

- Avantages :
  - des données sont accessibles sans chargement préalable
  - facilite le codage et une assez bonne densité
- Inconvénients :

Machines registre/mémoire (1,2)

- Avantages :
  - des données sont accessibles sans chargement préalable
  - facilite le codage et une assez bonne densité
- Inconvénients :
  - les opérandes ne sont pas équivalents

Machines registre/mémoire (1,2)

- Avantages :
  - des données sont accessibles sans chargement préalable
  - facilite le codage et une assez bonne densité
- Inconvénients :
  - les opérandes ne sont pas équivalents
  - le codage d’un numéro de registre et d’une adresse mémoire dans chaque instruction peut restreindre le nombre de registres

Machines registre/mémoire (1,2)

- Avantages :
  - des données sont accessibles sans chargement préalable
  - facilite le codage et une assez bonne densité
- Inconvénients :
  - les opérandes ne sont pas équivalents
  - le codage d’un numéro de registre et d’une adresse mémoire dans chaque instruction peut restreindre le nombre de registres
  - le nombre de cycles par instruction varie selon la position de l’opérande

Machines registre/registre (0,3)

- Avantages :

Machines registre/registre (0,3)

- Avantages :
  - codage simple d’instructions de longueur fixe

Machines registre/registre (0,3)

- Avantages :
  - codage simple d'instructions de longueur fixe
  - modèle simple de génération de code

Machines registre/registre (0,3)

- Avantages :
  - codage simple d'instructions de longueur fixe
  - modèle simple de génération de code
  - facilite l'implémentation d'un pipeline

Machines registre/registre (0,3)

- Avantages :
  - codage simple d'instructions de longueur fixe
  - modèle simple de génération de code
  - facilite l'implémentation d'un pipeline
- Inconvénients :

Machines registre/registre (0,3)

- Avantages :
  - codage simple d’instructions de longueur fixe
  - modèle simple de génération de code
  - facilite l’implémentation d’un pipeline
- Inconvénients :
  - moins bonne densité d’instructions

Machines registre/registre (0,3)

- Avantages :
  - codage simple d'instructions de longueur fixe
  - modèle simple de génération de code
  - facilite l'implémentation d'un pipeline
- Inconvénients :
  - moins bonne densité d'instructions
  - certaines instructions sont courtes et le codage peut gaspiller des bits

Deux jeux remarquables

(3, 3)	(1, 2)	(1, 1)	(0, 0)	(0, 3)
ADD @C, @A, @B;	LOAD R1, @A; ADD R1, @B; STORE R1, @C;	LOAD @A; ADD @B; STORE @C;	PUSH @A; PUSH @B; ADD ; POP @C;	LOAD R1, @A; LOAD R2, @B; ADD R3, R1, R2; STORE R3, @C

Ils correspondent à des approches othogonales :

- architectures (3,3) : machines CISC (Complex Instructions Set Computer)

Deux jeux remarquables

(3, 3)	(1, 2)	(1, 1)	(0, 0)	(0, 3)

`ADD @C, @A, @B;`	`LOAD R1, @A;`
	`ADD R1, @B;`
	`STORE R1, @C;`
		`LOAD @A;`
		`ADD @B;`
		`STORE @C;`
			`PUSH @A;`
			`PUSH @B;`
			`ADD ;`
			`POP @C;`
				`LOAD R1, @A;`
				`LOAD R2, @B;`
				`ADD R3, R1, R2;`
				`STORE R3, @C`

Ils correspondent à des approches othogonales :

- architectures (3,3) : machines CISC (Complex Instructions Set Computer)
- architectures (0,3) : machines RISC (Reduced Instructions Set Computer)

Machines RISC

Elles facilitent la gestion d'un *pipeline* de données, et donc une certaine *parallélisation* des traitements.

MIPS : un exemple de machine (0,3)

Microprocessor without Interlocked Pipeline Stages :

- Processeur RISC

MIPS : un exemple de machine (0,3)

Microprocessor without Interlocked Pipeline Stages :

- Processeur RISC
- Initialement développé fin des années 90, à partir des travaux de John L. Hennessy

MIPS : un exemple de machine (0,3)

Microprocessor without Interlocked Pipeline Stages :

- Processeur RISC
- Initialement développé fin des années 90, à partir des travaux de John L. Hennessy
- Implémentations actuelles : MIPS32 et MIPS64

MIPS : un exemple de machine (0,3)

Microprocessor without Interlocked Pipeline Stages :

- Processeur RISC
- Initialement développé fin des années 90, à partir des travaux de John L. Hennessy
- Implémentations actuelles : MIPS32 et MIPS64
- Equipe routeurs CISCO, consoles de jeu (Nintendo 64, Sony Playstations), cartes Silicon Graphics (jusqu'en 2006)