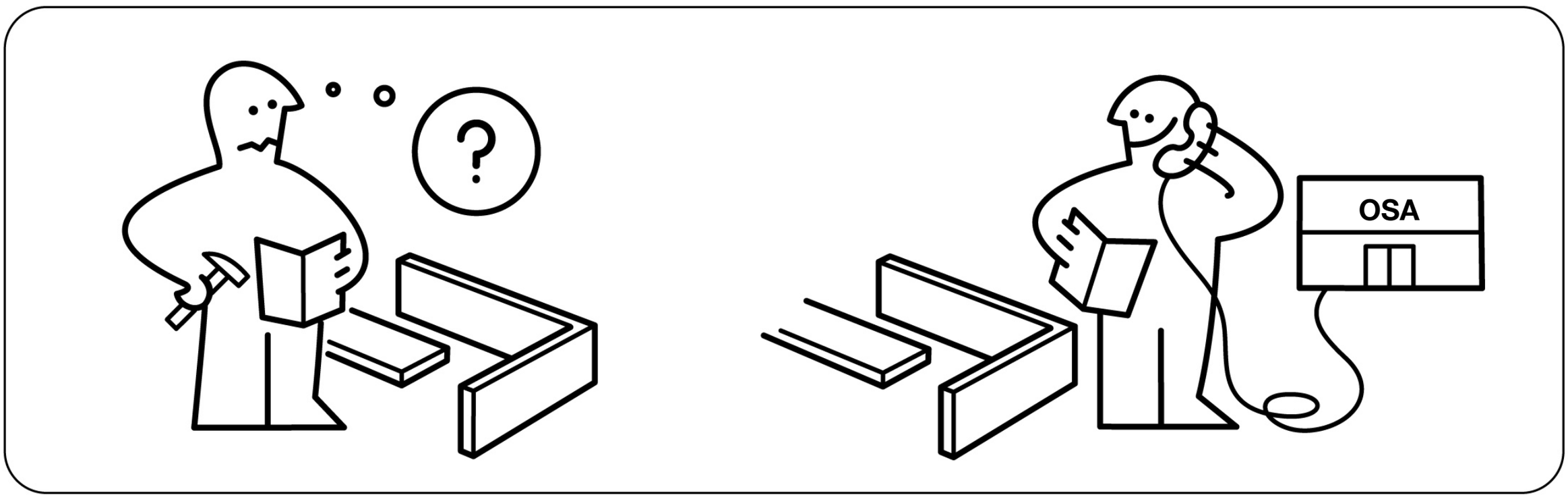
Instructions et jeu d’instructions

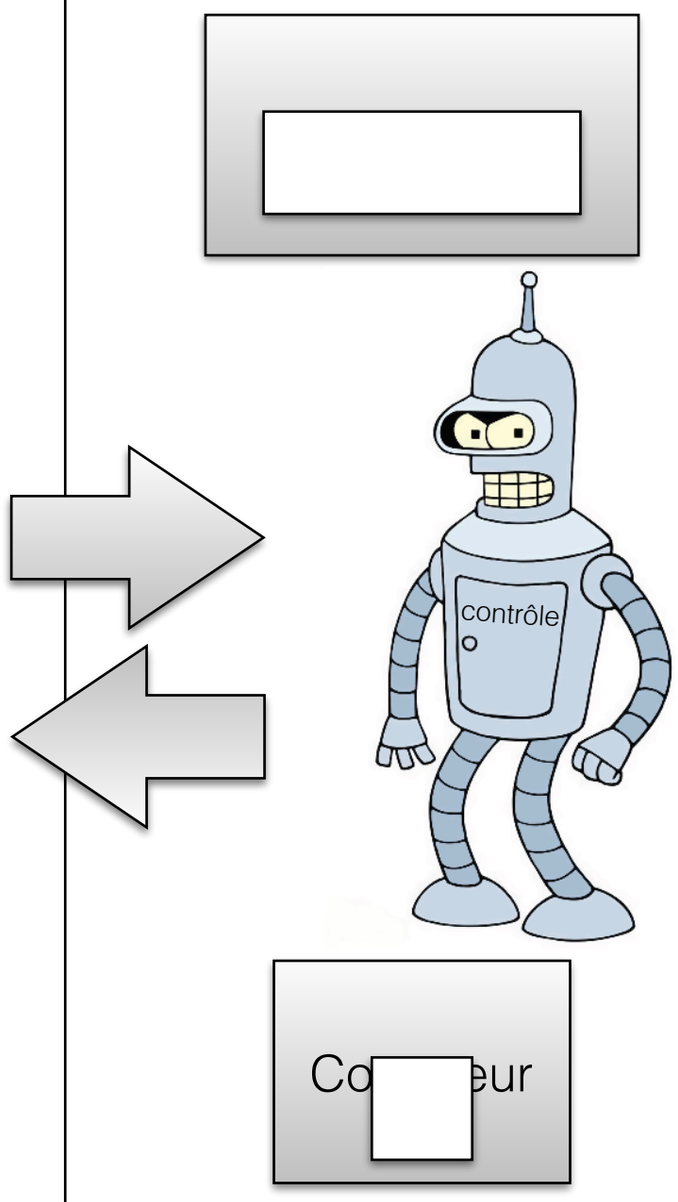


Ordinateurs: Structure et Applications, Hiver 2019

Jean-François Lalonde

Rappel: ordinateur simplifié

Ordinateur



Mémoire

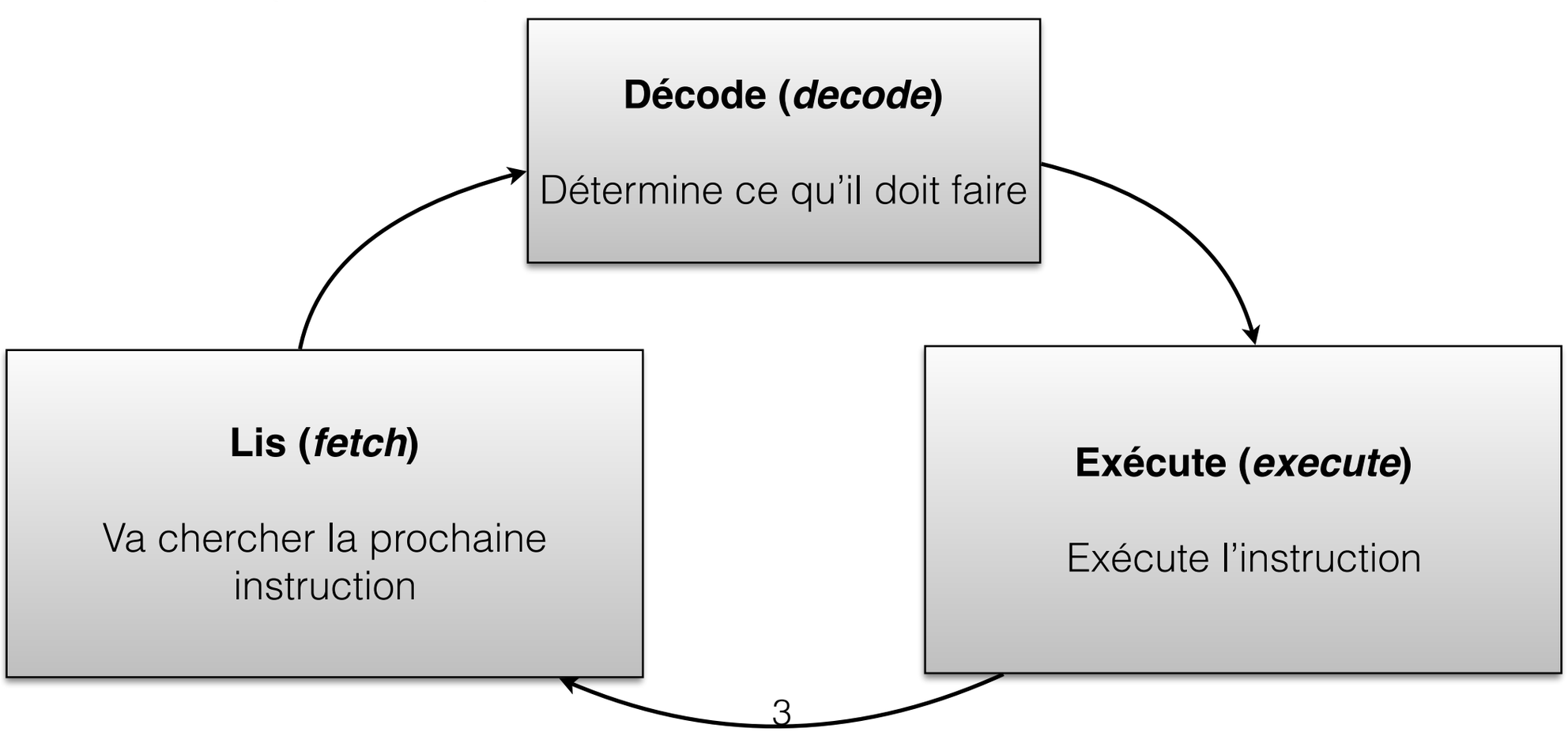
Adresse	Contenu
00	901
01	309
02	901
03	109
04	902
05	000
06	
07	
08	
09	

Liste des instructions disponibles

1xx	ADD calculatrice ← calculatrice + mémoire(xx)
2xx	SUB calculatrice ← mémoire(xx) - calculatrice
3xx	STORE mémoire(xx) ← calculatrice
5xx	LOAD calculatrice ← mémoire(xx)
901	INPUT calculatrice ← entrée
902	OUTPUT output ← calculatrice
000	BREAK arrête l'exécution

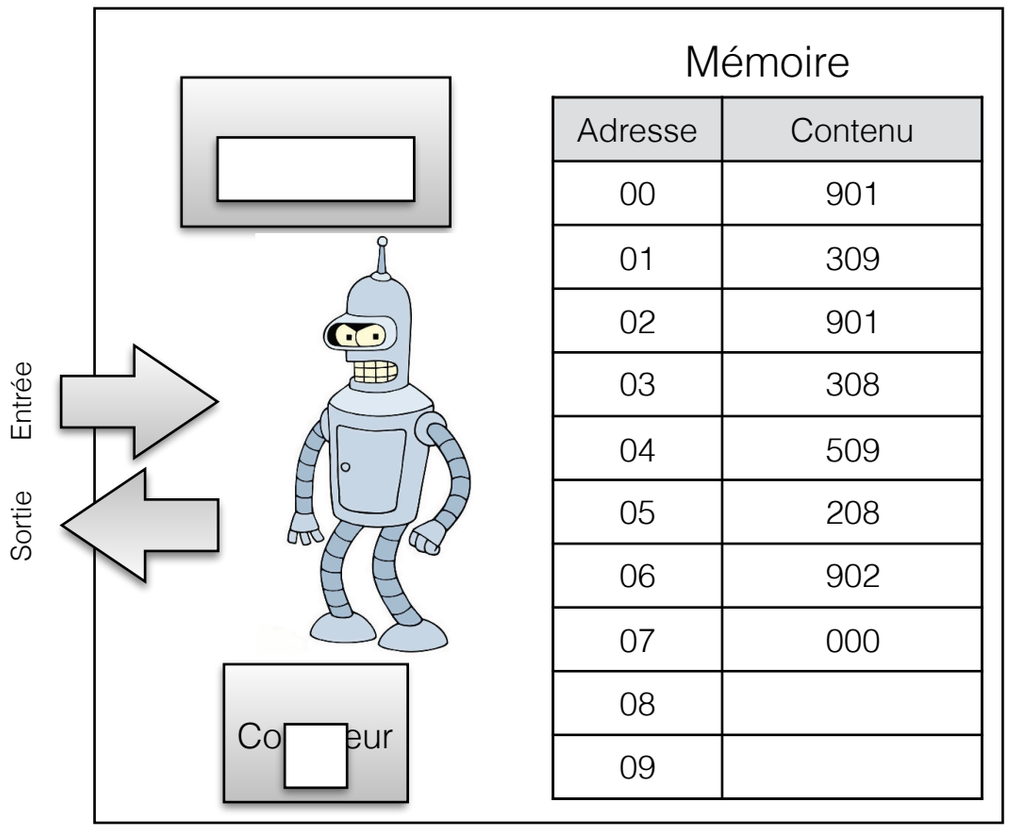
Cycle d'instructions

- Que fait le microprocesseur?
  - Lis (*fetch*): va chercher la prochaine instruction
  - Décode (*decode*): détermine ce qu'il doit faire, à partir de l'instruction
  - Exécute (*execute*): exécute l'instruction

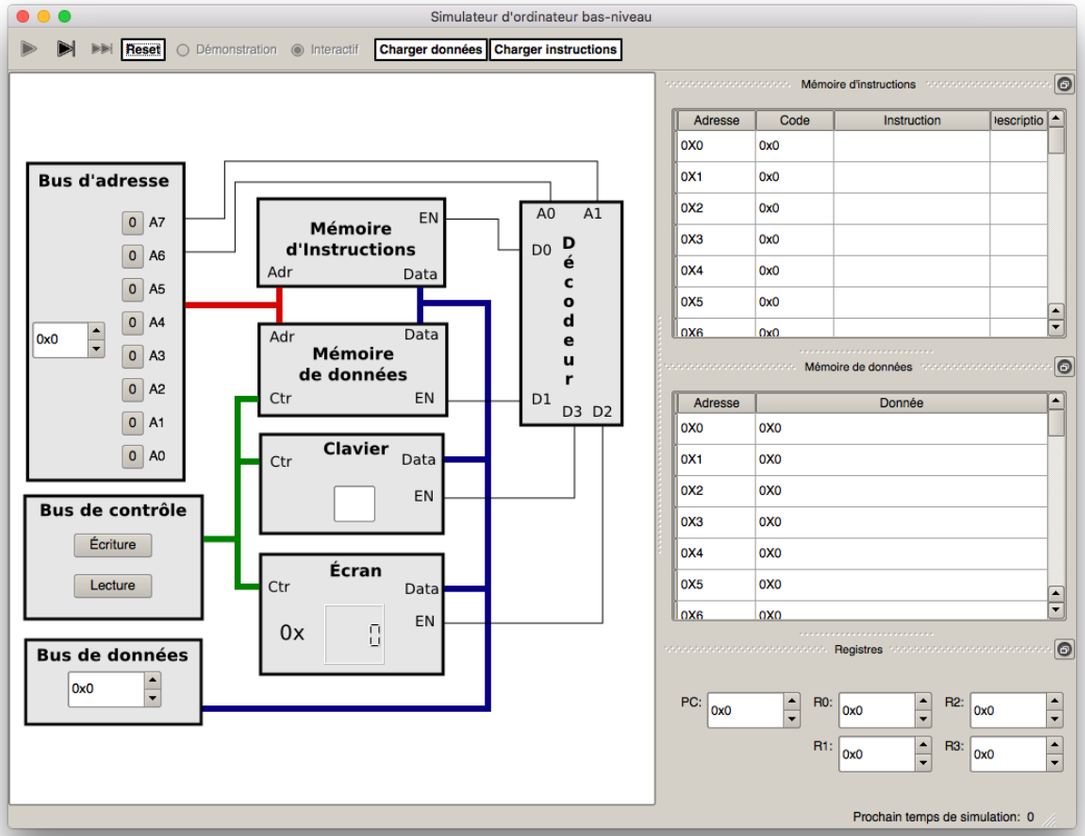


Aujourd’hui: d’un « ordinateur » à un autre

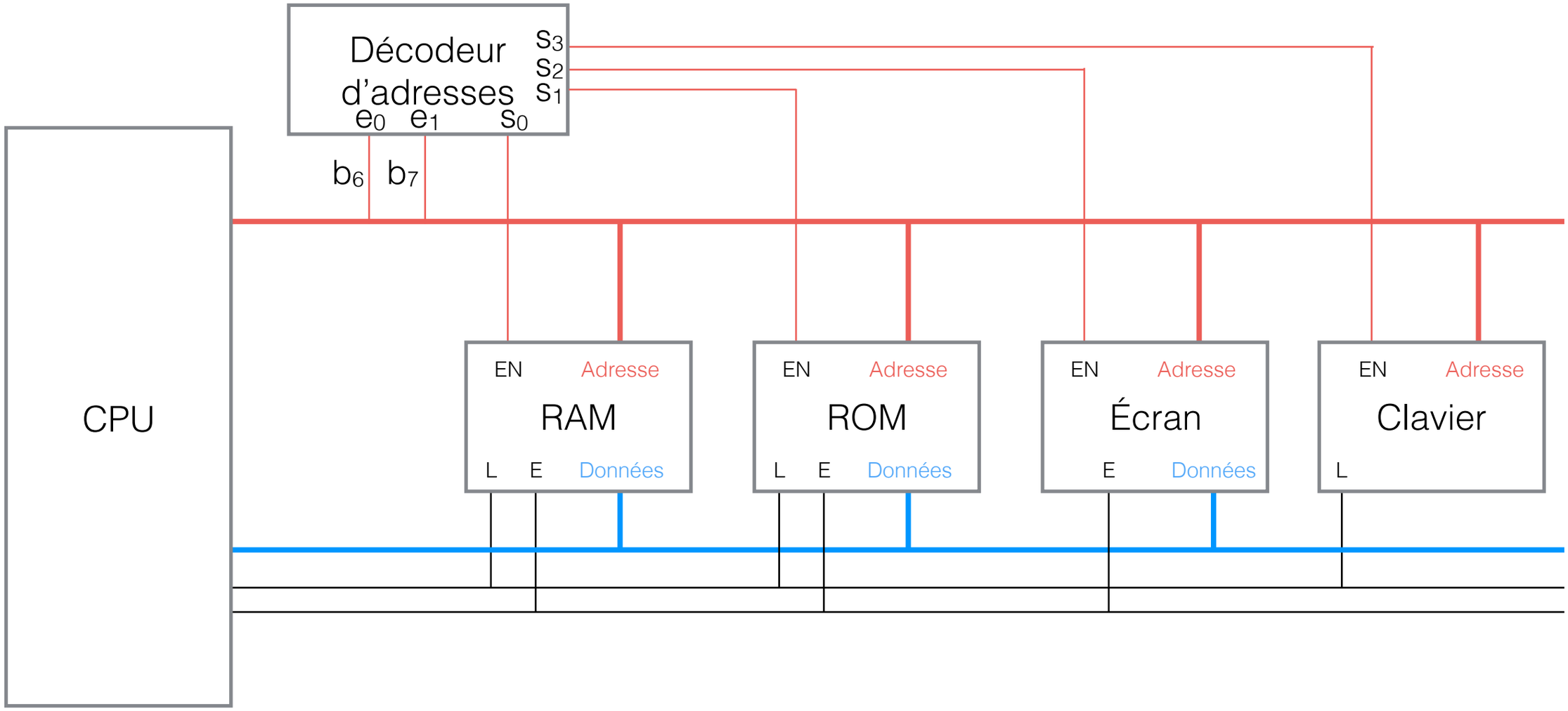
« Ordinateur » (très) simplifié



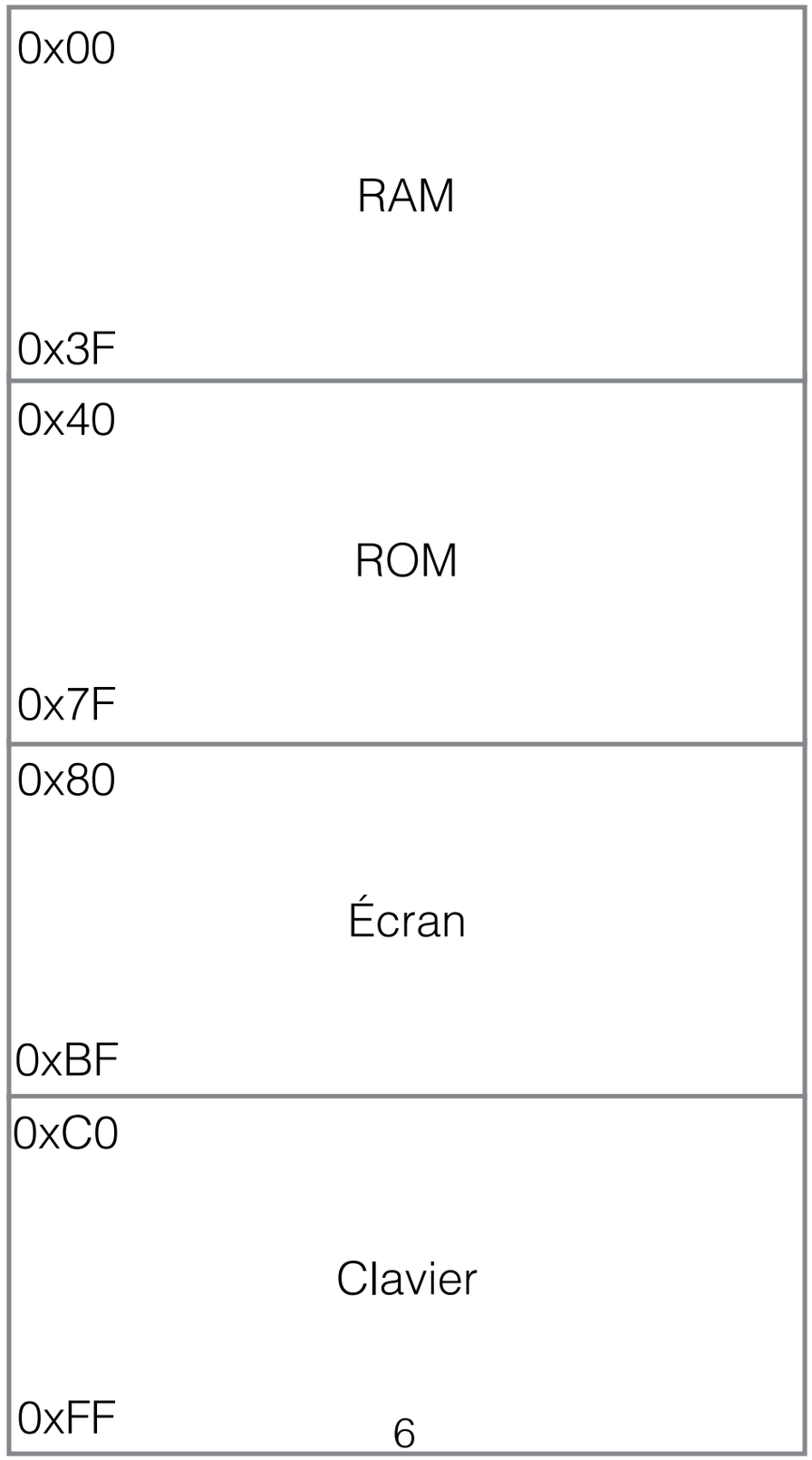
« Ordinateur » (un peu moins) simplifié



Rappel: CPU, mémoires, bus



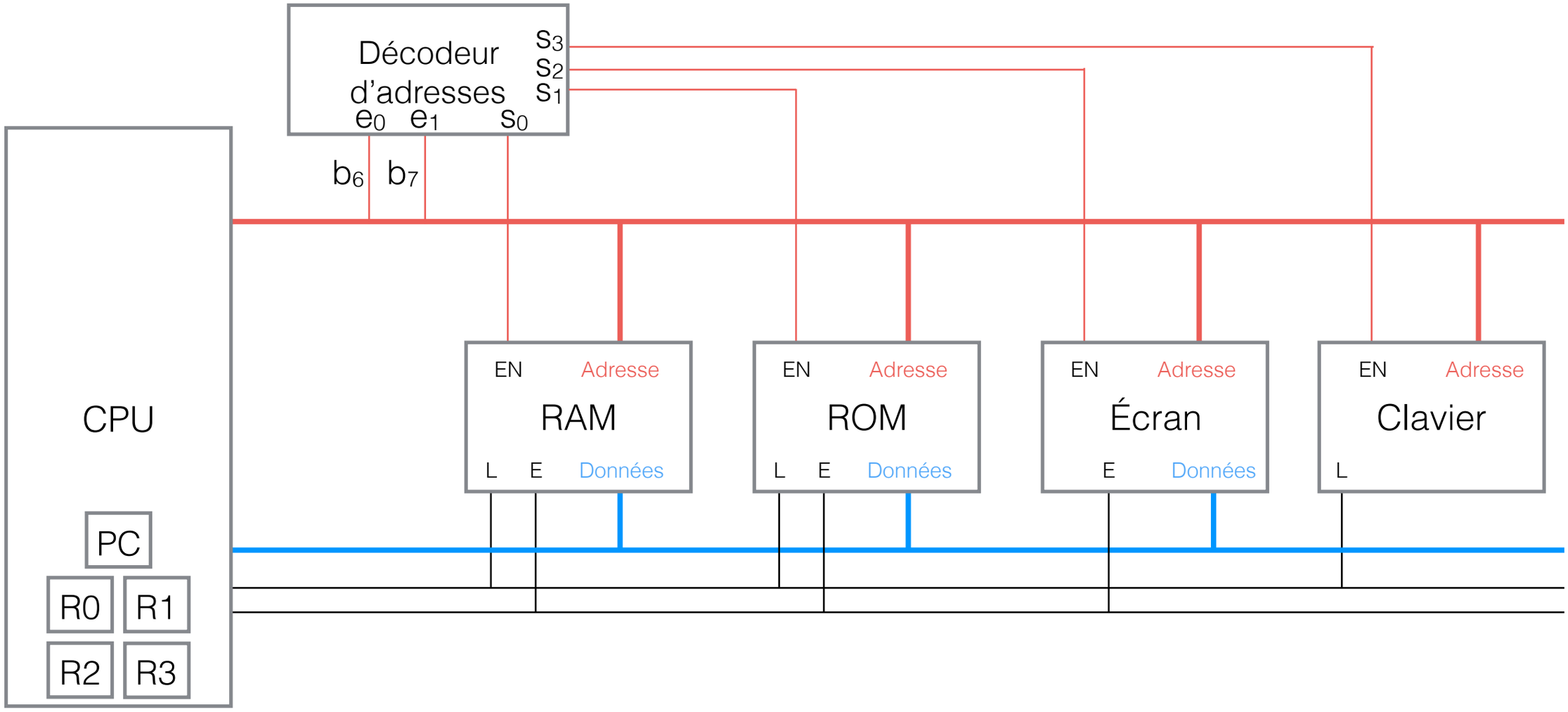
Rappel: carte de la mémoire (*memory map*)



Registres

- L'unité de contrôle à l'intérieur du microprocesseur contient des emplacements mémoire très rapides dédiés à des fonctions particulières: les registres
- Registres généraux:
  - permettent d'entreposer temporairement des données pour exécuter les programmes
- Registres particuliers:
  - PC: "Program Counter", stocke l'adresse (en mémoire) de la prochaine instruction à exécuter
  - IR: "Instruction Register", stocke l'instruction à décoder, puis à exécuter
  - ACC: "accumulator", stocke les résultats temporaires pour les calculs effectués par l'ALU
  - MAR: "Memory Address Register", détermine la prochaine adresse lue en mémoire
  - MDR: "Memory Data Register", emmagasine temporairement le contenu de la mémoire
  - Statut: "Status Register(s)", stocke des drapeaux (carry, overflow, zero, erreurs, etc.)

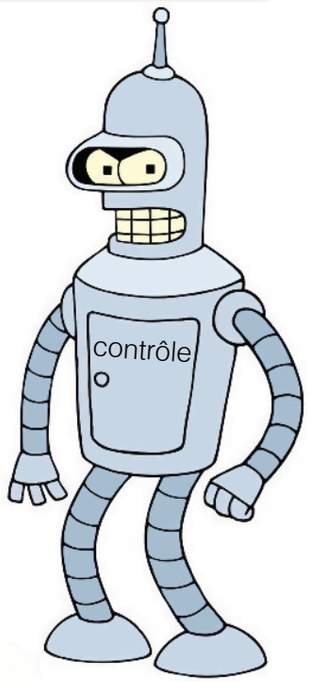
CPU, mémoire, entrées-sorties, bus, registres



Qu’est-ce qui nous manque?

Ordinateur

Calculatrice



Entrée

Sortie

Compteur

Mémoire

Adresse	Contenu
00	901
01	309
02	901
03	109
04	902
05	000
06	
07	
08	
09	

Liste des instructions disponibles

1xx	ADD calculatrice ← calculatrice + mémoire(xx)
2xx	SUB calculatrice ← mémoire(xx) - calculatrice
3xx	STORE mémoire(xx) ← calculatrice
5xx	LOAD calculatrice ← mémoire(xx)
901	INPUT calculatrice ← entrée
902	OUTPUT output ← calculatrice
000	BREAK arrête l’exécution

Instructions et jeu d’instructions

- Une instruction est une « action » pouvant être exécutée par le microprocesseur.

- Un « jeu d’instructions » représente **toutes** les instructions pouvant être exécutées par un microprocesseur.

- Il existe plusieurs types d’instructions, notamment:
  - mouvements de données
  - arithmétique et logique
  - contrôle/flot de programme

Instructions (ordinateur simplifié)

Jeu d'instructions

1xx	ADD calculatrice ← calculatrice + mémoire(xx)	Opérations arithmétiques
2xx	SUB calculatrice ← mémoire(xx) - calculatrice	
3xx	STORE mémoire(xx) ← calculatrice	Mouvement de données
5xx	LOAD calculatrice ← mémoire(xx)	
901	INPUT calculatrice ← entrée	
902	OUTPUT output ← calculatrice	
000	BREAK arrête l'exécution	Flot du programme

Instructions (TP1)

Jeu d'instructions

Mnémonique	
MOV Rd, Rs	Déplacement de données
MOV Rd, Const	
ADD Rd, Rs	Opérations arithmétiques
ADD Rd, Const	
SUB Rd, Rs	
SUB Rd, Const	
LDR Rd, [Rs]	Déplacement de données
STR Rd, [Rs]	
JZE Rc, Const	Flot du programme
JZE Rc, Rs	

Déplacement de données

- D’un registre à l’autre

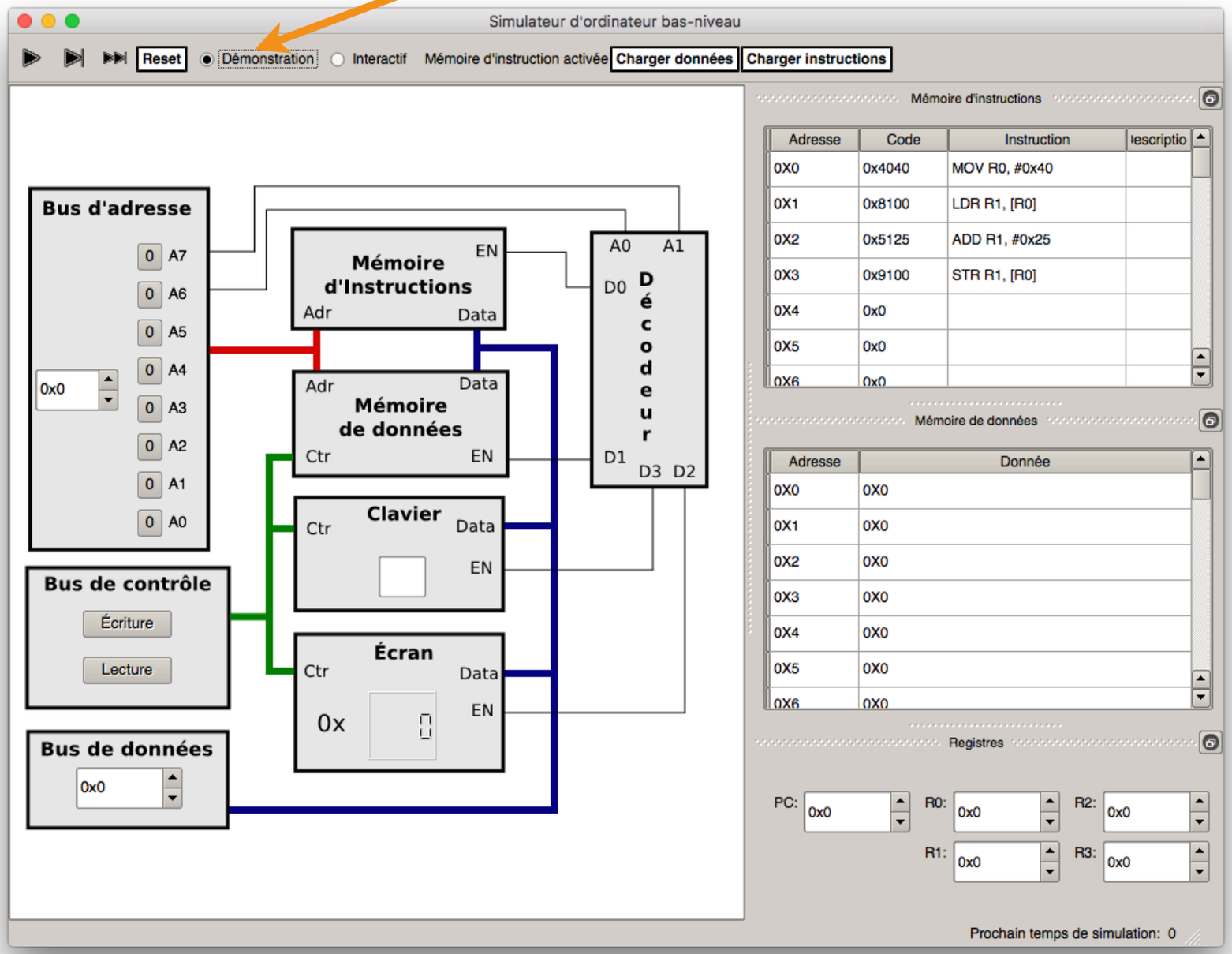
Instructions	Signification
`MOV R1, R2`	`R1 ← R2`
`MOV R0, #0x71`	`R0 ← 0x71`

- est-ce qu’on accède à la mémoire?

Tous les exercices pour ce module sont disponibles sur le site web du cours (voir semaine 3).

Chargez-les dans le simulateur du TP1 et activez le mode « démonstration ».

Notez: le mode « démonstration » est habituellement grisé, et c'est normal!
Il n'est activé que pour les exercices que nous vous donnons.



Exercice #1

Échangez le contenu des registres R0 et R1 (R0 ← R1 et R1 ← R0).

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #1

Échangez le contenu des registres R0 et R1 (R0 ← R1 et R1 ← R0).

- Utilisons un registre temporaire, par exemple R2
- Sauvegardons la valeur de R0 dans R2

```
MOV R2, R0
```

- Plaçons la valeur de R1 dans R0

```
MOV R0, R1
```

- Plaçons la valeur initiale de R0 (maintenant dans R2) dans R1

```
MOV R1, R2
```

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #1 (solution)

Échangez le contenu des registres R0 et R1 (R0 ← R1 et R1 ← R0).

```
MOV R2, R0
MOV R0, R1
MOV R1, R2
```

Déplacement de données

- De la mémoire vers un registre

Instructions	Signification
`LDR R1, [R2]`	`R1 ← Memoire[R2]`

  - lecture ou écriture?

- D’un registre vers la mémoire

Instructions	Signification
`STR R1, [R2]`	`Memoire[R2] ← R1`

  - lecture ou écriture?

```
Memoire[XX] = contenu de la mémoire à l’adresse ‘XX’
```

Exercice #2

Copiez le mot situé en mémoire à l'adresse 0x40
vers l'adresse 0x41

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #2

Copiez le mot situé en mémoire à l'adresse 0x40
vers l'adresse 0x41

- Tout d'abord, plaçons le contenu de la mémoire à l'adresse 0x40 dans un registre, par exemple R1
  - Je ne peux pas faire LDR R1, [#0x40]. Je devrai donc utiliser un registre (ex: R0) pour stocker l'adresse source

```
MOV R0, #0x40
LDR R1, [R0]
```

- Ensuite, stocker le contenu de R1 à l'adresse 0x41.
  - Je ne peux pas faire STR R1, [#0x41]. Je devrai donc utiliser un registre (ex: R0) pour stocker l'adresse de destination

```
ADD R0, #0x1
STR R1, [R0]
```

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #2 (solution)

Copiez le mot situé en mémoire à l'adresse 0x40
vers l'adresse 0x41

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #0x1
STR R1, [R0]
```

Opérations arithmétiques

- Additions de nombres entiers

Instructions	Signification
`ADD R1, R2`	`R1 ← R1 + R2`
`ADD R1, #0x2`	`R1 ← R1 + 0x2`

- Soustractions de nombres entiers

Instructions	Signification
`SUB R1, R2`	`R1 ← R1 - R2`
`SUB R1, #0x2`	`R1 ← R1 - 0x2`

Exercice #3

Additionnez 0x25 à ce qu'il y a en mémoire à l'adresse 0x40.
Stockez le résultat à cette même adresse.

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #3

Additionnez 0x25 à ce qu'il y a en mémoire à l'adresse 0x40.
Stockez le résultat à cette même adresse.

- Tout d'abord, allons chercher le contenu de la mémoire à l'adresse 0x40.
  - Je ne peux pas faire LDR R0, [0x40]!

```
MOV R0, #0x40
LDR R1, [R0]
```

- Additionnons-y 0x25

```
ADD R1, #0x25
```

- Stockons le résultat à l'adresse 0x40
  - Je ne peux pas faire STR R1, [0x40]!
  - Cependant, R0 contient déjà la bonne adresse…

```
STR R1, [R0]
```

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #3 (solution)

Additionnez 0x25 à ce qu'il y a en mémoire à l'adresse 0x40.
Stockez le résultat à cette même adresse.

```
MOV R0, #0x40
LDR R1, [R0]
ADD R1, #0x25
STR R1, [R0]
```

Questions

- Où les instructions sont-elles stockées?
  - En mémoire!
- Comment représente-t-on une instruction dans un ordinateur?
  - En binaire, pardi!
- Comment faire pour stocker une instruction en binaire?
  - Nous aurons besoin d'une recette! Vous vous rappelez du PHIR™ #4?

PHIR™ #3

- A priori, nous ne pouvons pas savoir ce qu'une chaîne binaire signifie.
  - Ex: que veut dire 0x4040 (sur 16 bits)?
  - La bonne réponse est: ça dépend!

entier non-signé	16448
entier signé	16448
caractères ASCII	@@
instruction TP1	MOV R0, #0x40

- Il nous *faut* donc savoir quel format (quelle «recette») utiliser pour bien interpréter les données!

La recette d’une instruction

- Séparée en deux:
  - code d’opération (“opcode”)
  - des paramètres: format et taille dépendent de l’opcode
- Par exemple (TP1):
  - instruction sur 16 bits
  - opcode: 4 premiers bits
    - combien d’opcodes peut-on définir au total?
  - paramètres: 12 derniers bits

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Jeu d'instructions

- La table ci-dessous est un exemple de jeu d'instructions.

- Chaque instruction possède un mnémonique en assembleur.

Toutes les instructions du microprocesseur sont sur 16 bits et se décomposent comme suit :
Bits 15 à 12 : Opcode de l'instruction
Bits 11 à 8 : Registre utilisé comme premier paramètre.
Bits 7 à 0 : Registre ou constante utilisés comme deuxième paramètre

Le microprocesseur possède quatre registres généraux nommés R0, R1, R2 et R3. En plus de ces quatre registres, un registre de pointeur d'instruction PC est disponible. Cependant, ce registre ne peut être utilisé qu'avec l'instruction MOV. Le nombre identifiant le registre PC est 0xF (15).

Le jeu d'instruction supporte les instructions suivantes où Rd est le registre destination, Rs le registre source et Rc le registre de condition :

Mnémonique	**Opcode**	**Description**
MOV Rd, Rs	0000	Écriture de la valeur du registre Rs dans le registre Rd
MOV Rd, Const	0100	Écriture d'une constante dans le registre Rd
ADD Rd, Rs	0001	Addition des valeurs des registres Rd et Rs et insertion du résultat dans le registre Rd
ADD Rd, Const	0101	Addition de la valeur du registre Rd avec une constante et insertion du résultat dans Rd
SUB Rd, Rs	0010	Soustraction de la valeur Rs à l'intérieur de registre Rd.
SUB Rd, Const	0110	Soustraction d'une constante à l'intérieur du registre Rd
LDR Rd, [Rs]	1000	Chargement d'une valeur se trouvant à l'adresse Rs de l'ordinateur dans un registre.
STR Rd, [Rs]	1001	Écriture de la valeur d'un registre à l'adresse Rs de l'ordinateur.
JZE Rc, Const	1111	Saut à l'instruction située à l'adresse identifiée par la constante, mais seulement si Rc = 0 (sinon, cette instruction n'a aucun effet).
JZE Rc, Rs	1011	Saut à l'instruction située à l'adresse Rs seulement si Rc = 0 (sinon, cette instruction n'a aucun effet).

TABLE 1 – Jeu d'instructions du microprocesseur

Exercice #4

Prenons l'exercice #2 de tout à l'heure.
Quelle est sa représentation binaire?

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #0x1
STR R1, [R0]
```

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #4

Prenons l'exercice #2 de tout à l'heure.
Quelle est sa représentation binaire?

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #0x1
STR R1, [R0]
```

- Traduisons chaque instruction une à une
  - MOV R0, #0x40
    - opcode (4 bits): MOV Rd, Const = 0b0100
    - paramètre 1 (4 bits): 0b0000
    - paramètre 2 (8 bits): 0x40 = 0b01000000
    - donc, 0b0100 0000 0100 0000 = 0x4040

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #4 (suite)

Prenons l'exercice #2 de tout à l'heure.
Quelle est sa représentation binaire?

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #0x1
STR R1, [R0]
```

- Traduisons chaque instruction une à une
  - LDR R1, [R0]
    - opcode (4 bits): LDR Rd, [Rs] = 0b1000
    - paramètre 1 (4 bits): 0b0001
    - paramètre 2 (8 bits): 0b00000000
    - donc, 0b1000 0001 0000 0000 = 0x8100

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #4 (suite)

Prenons l'exercice #2 de tout à l'heure.
Quelle est sa représentation binaire?

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #0x1
STR R1, [R0]
```

- Traduisons chaque instruction une à une
  - ADD R0, #0x1
    - opcode (4 bits): ADD Rd, Const = 0b0101
    - paramètre 1 (4 bits): 0b0000
    - paramètre 2 (8 bits): 0b00000001
    - donc, 0b0101 0000 0000 0001 = 0x5001

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #4 (suite)

Prenons l'exercice #2 de tout à l'heure.
Quelle est sa représentation binaire?

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #0x1
STR R1, [R0]
```

- Traduisons chaque instruction une à une
  - STR R1, [R0]
    - opcode (4 bits): STR Rd, [Rs] = 0b1001
    - paramètre 1 (4 bits): 0b0001
    - paramètre 2 (8 bits): 0b00000000
    - donc, 0b1001 0001 0000 0000 = 0x9100

Mnémonique	Opcode
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #4 (solution)

Prenons l'exercice #2 de tout à l'heure.
Quelle est sa représentation binaire?

```
MOV R0, #0x40
LDR R1, [R0]
ADD R0, #0x1
STR R1, [R0]
```

```
0x4040
0x8100
0x5001
0x9100
```

Exercice #5

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #5

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

- Traduisons chaque instruction une à une
  - 0x4040
    - opcode (4 bits): 0b0100 = MOV Rd, Const
    - paramètre 1 (4 bits): 0b0000 = R0
    - paramètre 2 (8 bits): #0x40
    - donc: MOV R0, #0x40

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #5

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

- Traduisons chaque instruction une à une
  - 0x8100
    - opcode (4 bits): 0b1000 = LDR Rd, [Rs]
    - paramètre 1 (4 bits): 0b0001 = R1
    - paramètre 2 (8 bits): 0b00000000 = R0
    - donc: LDR R1, [R0]

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #5

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

- Traduisons chaque instruction une à une
  - 0x4041
    - opcode (4 bits): 0b0100 = MOV Rd, Const
    - paramètre 1 (4 bits): 0b0000 = R0
    - paramètre 2 (8 bits): #0x41
    - donc: MOV R0, #0x41

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #5

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

- Traduisons chaque instruction une à une
  - 0x8200
    - opcode (4 bits): 0b1000 = LDR Rd, [Rs]
    - paramètre 1 (4 bits): 0b0002 = R2
    - paramètre 2 (8 bits): 0b00000000 = R0
    - donc: LDR R2, [R0]

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #5

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

- Traduisons chaque instruction une à une
  - 0x1201
    - opcode (4 bits): 0b0001 = ADD Rd, Rs
    - paramètre 1 (4 bits): 0b0010 = R2
    - paramètre 2 (8 bits): 0b00000001 = R1
    - donc: ADD R2, R1

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #5

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

- Traduisons chaque instruction une à une
  - 0x9200
    - opcode (4 bits): 0b1001 = STR Rd, [Rs]
    - paramètre 1 (4 bits): 0b0010 = R2
    - paramètre 2 (8 bits): 0b00000000 = R0
    - donc: STR R2, [R0]

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Opcode	Paramètre 1	Paramètre 2
4 bits	4 bits	8 bits

Exercice #5 (solution)

Décrivez, *en une phrase*,
ce que fait le programme suivant.

```
0x4040
0x8100
0x4041
0x8200
0x1201
0x9200
```

```
MOV R0, #0x40
LDR R1, [R0]
MOV R0, #0x41
LDR R2, [R0]
ADD R2, R1
STR R2, [R0]
```

Le programme calcule la somme des valeurs en mémoire placées aux adresses 0x40 et 0x41, et stocke le résultat à l'adresse 0x41.

Contrôle de programmes

- «Sauter» d'une adresse mémoire à un autre

Instructions (TP1)	Signification
`JZE R0, R1`	`Si R0 == 0, alors PC ← R1`
`JZE R0, #0x2`	`Si R0 == 0, alors PC ← 0x2`

Exercice #6 (plus difficile)

Si le contenu en mémoire à l'adresse 0x40 *n'est pas* égal à 0, placer la valeur 0x10 dans R1. Sinon, ne pas modifier R1.

Indices

Chaque instruction possède une adresse.

Écrivez l'adresse de chaque instruction (en commençant à 0x0).

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #6 (plus difficile)

Si le contenu en mémoire à l'adresse 0x40 *n'est pas* égal à 0, placer la valeur 0x10 dans R1. Sinon, ne pas modifier R1.

Indices

Chaque instruction possède une adresse. Écrivez l'adresse de chaque instruction (en commençant à 0x0).

- Chargeons le contenu en mémoire à l'adresse 0x40

```
Adresse             Instruction
0x0                 MOV R0, #0x40
0x1                 LDR R0, [R0]
```

- Si le contenu est égal à 0, sauter par-dessus l'instruction qui place 0x10 dans R1

```
Adresse             Instruction
0x2                 JZE R0, #0x4
0x3                 MOV R1, #0x10
```

Mnémonique
MOV Rd, Rs
MOV Rd, Const
ADD Rd, Rs
ADD Rd, Const
SUB Rd, Rs
SUB Rd, Const
LDR Rd, [Rs]
STR Rd, [Rs]
JZE Rc, Const
JZE Rc, Rs

Exercice #6 (solution)

Si le contenu en mémoire à l'adresse 0x40 *n'est pas* égal à 0, placer la valeur 0x10 dans R1. Sinon, ne pas modifier R1.

Indices

Chaque instruction possède une adresse. Écrivez l'adresse de chaque instruction (en commençant à 0x0).

```
Adresse             Instruction

0x0                 MOV R0, #0x40
0x1                 LDR R0, [R0]
0x2                 JZE R0, #0x4
0x3                 MOV R1, #0x10
```

Exercice #7 (défi)

Décrivez, *en une phrase*, ce que fait le programme suivant.

```
Adresse     Contenu

0x0         0x4040
0x1         0x8100
0x2         0x0301
0x3         0x4200
0x4         0x6101
0x5         0x1203
0x6         0xF108
0x7         0x4F04
0x8         0x9200
```

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Exercice #7, étape 1: traduction en instructions

Décrivez, *en une phrase*, ce que fait le programme suivant.

```
Adresse     Contenu

0x0         0x4040
0x1         0x8100
0x2         0x0301
0x3         0x4200
0x4         0x6101
0x5         0x1203
0x6         0xF108
0x7         0x4F04
0x8         0x9200
```

```
Adresse     Contenu

0x0         MOV R0, #0x40
0x1         LDR R1, [R0]
0x2         MOV R3, R1
0x3         MOV R2, #0x0
0x4         SUB R1, #0x1
0x5         ADD R2, R3
0x6         JZE R1, #0x8
0x7         MOV PC, #0x4
0x8         STR R2, [R0]
```

Mnémonique	**Opcode**
MOV Rd, Rs	0000
MOV Rd, Const	0100
ADD Rd, Rs	0001
ADD Rd, Const	0101
SUB Rd, Rs	0010
SUB Rd, Const	0110
LDR Rd, [Rs]	1000
STR Rd, [Rs]	1001
JZE Rc, Const	1111
JZE Rc, Rs	1011

Exercice #7, étape 2: interprétation des instructions

Décrivez, *en une phrase*, ce que fait le programme suivant.

```
Adresse     Contenu

0x0         MOV R0, #0x40
0x1         LDR R1, [R0]
0x2         MOV R3, R1
0x3         MOV R2, #0x0
0x4         SUB R1, #0x1
0x5         ADD R2, R3
0x6         JZE R1, #0x8
0x7         MOV PC, #0x4
0x8         STR R2, [R0]
```

- Trucs:
  - tester avec une petite valeur
  - exécuter les instructions une à une
  - écrire le contenu des registres au fur et à mesure

Exercice #7 (solution)

Décrivez, *en une phrase*, ce que fait le programme suivant.

Le programme élève au carré la valeur stockée à l'adresse 0x40 et écrit le résultat à cette même adresse.

```
Adresse     Contenu

0x0         MOV R0, #0x40
0x1         LDR R1, [R0]
0x2         MOV R3, R1
0x3         MOV R2, #0x0
0x4         SUB R1, #0x1
0x5         ADD R2, R3
0x6         JZE R1, #0x8
0x7         MOV PC, #0x4
0x8         STR R2, [R0]
```

RISC & CISC

- Il existe plusieurs approches pour la conception d'un microprocesseur et de son jeu d'instructions. Ces approches influencent chaque aspect du design de l'architecture d'un microprocesseur. Les principales approches utilisées à ce jour sont CISC et RISC.

- CISC (Complex Instruction Set Computer)

  - jeu d'instructions complexe dont la longueur (des instructions) varie. Comme les instructions peuvent être longues et complexes, peu de registres sont requis.

  - Exemple: x86: (8086, Pentium)

- RISC (Reduced Instruction Set Computer)

  - jeu d'instructions simple dont la longueur est fixe (ex: 4 octets). Plusieurs registres sont requis pour exécuter des tâches complexes.

  - Exemple: PowerPC, ARM

RISC vs. CISC

Avantages RISC (**R**educed **I**nstruction **S**et **C**omputer)	**Avantages CISC** (**C**omplex **I**nstruction **S**et **C**omputer)
10 instructions sont utilisées 70% du temps	Plus de flexibilité au programmeur (par exemple, transferts mémoire-mémoire)
Avoir plusieurs registres permet d'éviter les accès mémoires (qui sont plus lents)	Programmes plus courts, plus petits
Opérations "fetch-decode-execute" sont simplifiées, car toutes les instructions ont la même taille	
Opérations simplifiées = architecture simplifiée = consommation réduite	

- Les microprocesseurs modernes sont des microprocesseurs RISCs ou hybrides (un microprocesseur supportant des instructions ayant deux longueurs seulement par exemple).

- Les microprocesseurs CISCs disponibles découpent habituellement les instructions complexes en instructions simples (comme du RISC) avant de les exécuter.