

# Quelques aspects matériels de l'informatique...



Philippe Marquet

---

ressources à [eil.fil.univ-lille1.fr/hsw18/](http://eil.fil.univ-lille1.fr/hsw18/)

contact [philippe.marquet@univ-lille.fr](mailto:philippe.marquet@univ-lille.fr)

# du matériel au logiciel

- fonctionnement d'un ordinateur, du processeur
- programmation “bas niveau”
- lien avec les langages de plus haut niveau

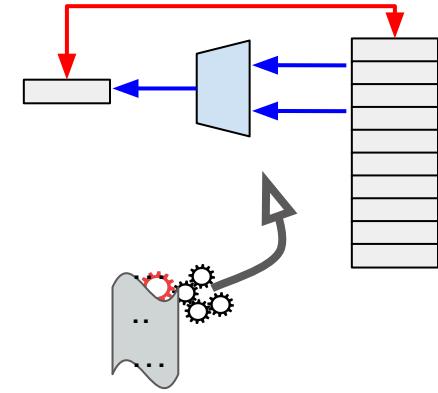
---

- des petites activités
- de petits éclairages 

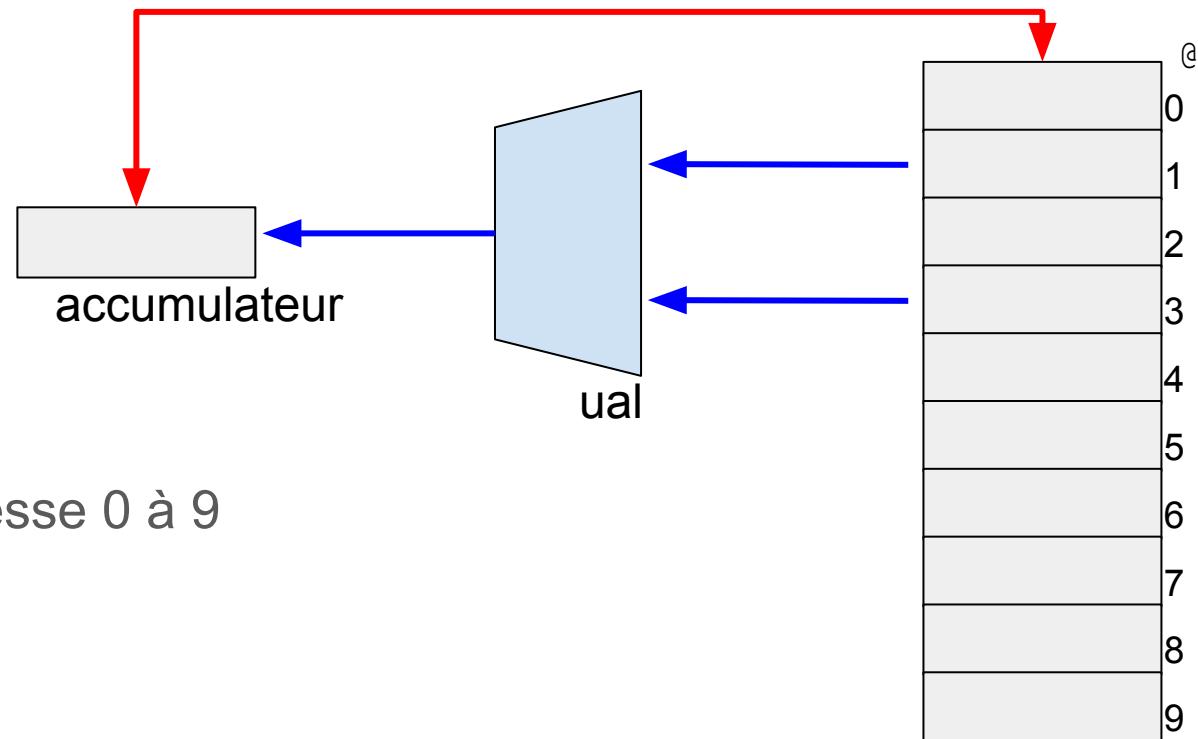
... sans ordinateur

# M-10

# la machine débranchée



# M-10 la petite machine débranchée



mémoire

10 "cases", adresse 0 à 9

instructions

- transfert mémoire ↔ accumulateur
- arithmétiques / logiques

mémoire

# M-10 jeu d'instructions

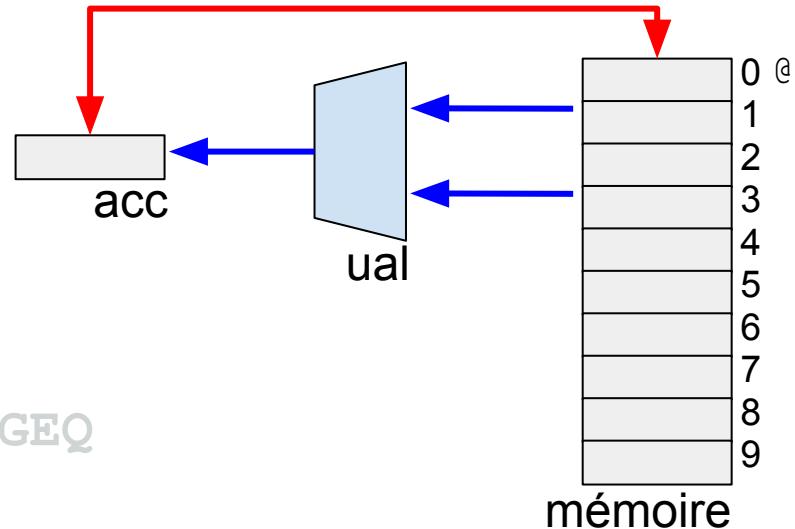
## arithmétiques et logiques

**ADD** @1 @2

- addition
- idem pour **SUB**, **MUL**, **DIV**

**LSS** @1 @2

- plus petit que
- idem pour **EQU**, **NEQ**, **LEQ**, **GTR**, **GEQ**



## transfert mémoire

**LOAD** @

- transfert de la mémoire vers l'accumulateur

**STORE** @

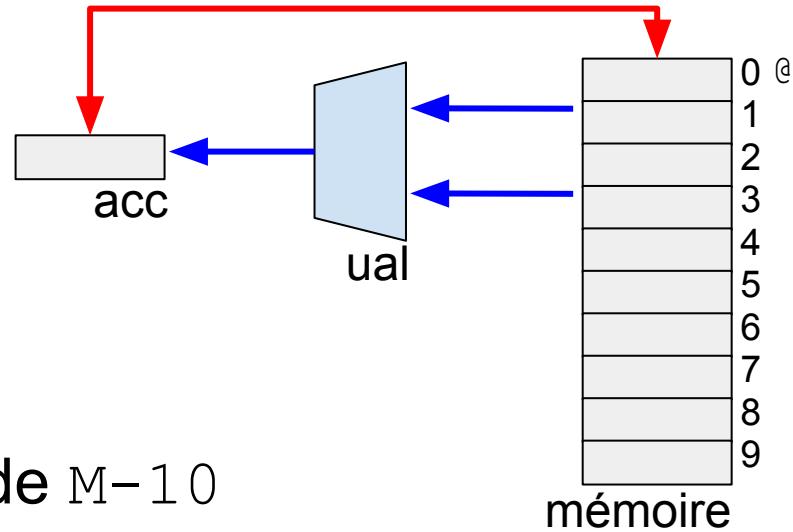
- transfert de l'accumulateur vers la mémoire

# programmer un “programme de calcul”

---

- choisir un nombre
- soustraire 2 à ce nombre
- multiplier le résultat par 4

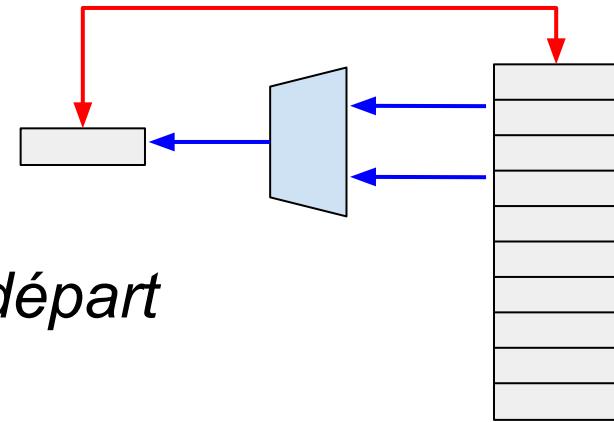
→ écrire une suite d'instructions de  $M-10$



# d'autres “programmes de calcul”

---

- choisir un nombre
  - soustraire 2 à ce nombre
  - multiplier le résultat par 4
  - *ajouter le quadruple du nombre de départ*
- 



- effectuer le produit de 3 nombres consécutifs
- 

- ...



# M-10 est un ordinateur

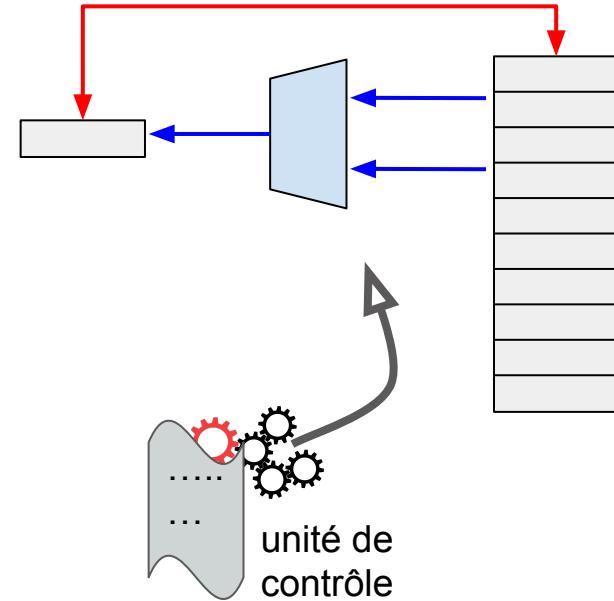
M-10 est une **machine informatique**

- mémoire (+accumulateur / registres)
- unité de calcul

M-10 est une **machine programmable**

- un **ordinateur**

- processeur
- exécute une à une des instructions
- déroule un programme



programmer M-10

# expression, affectation

---

– on “pense” plus haut niveau

$m = n - 2$

$res = m * 2 + 2 * n$

$res = res + 1$

→ traduire ce (type de) programme  
en un programme de la machine M-10

## programmer M-10

# expression, affectation

---

– on “pense” plus haut niveau

$m = n - 2$

$res = m * 2 + 2 * n$

$res = res + 1$

→ traduire ce (type de) programme  
en un programme de la machine M-10

utilisation d'une **table des symboles**

table des symboles

name	@mémoire
n	
m	
res	
_tmp_0	
...	
...	

# programmer M-10

## rupture de séquence

---

- on suppose la multiplication non disponible
- réalisation par additions successives

```
i = 0 ; res = 0
```

```
loop: res += b
```

```
    i += 1
```

```
    si i = a brancher à loop
```

a \* b

### instructions de branchement

**JMP@label**

- branche à l'instruction **@label**

**JNZ@label**

- branche à l'instruction **@label** si l'accumulateur est non nul

# structures de contrôle

---

## Les classiques

- if .. then ..
- if .. then .. else ..
- while .. do ..
- for  $i$  :  $lower \uparrow upper$  do ..

→ à l'aide des instructions de branchement **JMP** et **JNZ**

# programmer M-10 alternative

---

max = a

si max < b

    max = b

    si a < b

        max = b

    sinon

        max = a

→ à traduire à l'aide des instructions de branchement  
**JMP** et **JNZ** de M-10

---

# programmer M-10 répétitions

---

```
i = 0
res = 0
tantque i < a
    res += b
    i += 1
```

```
res = 0
pour i de 1 → a
    res += b
```

→ à traduire à l'aide des instructions de branchement  
**JMP** et **JNZ** de M-10

# compiler pour M-10

# structures de contrôle

---

Compilation des

- if .. then ..
- if .. then .. else ..
- while .. do ..
- for  $i$  :  $lower \nearrow upper$  do ..

en instructions M-10

→ traduction

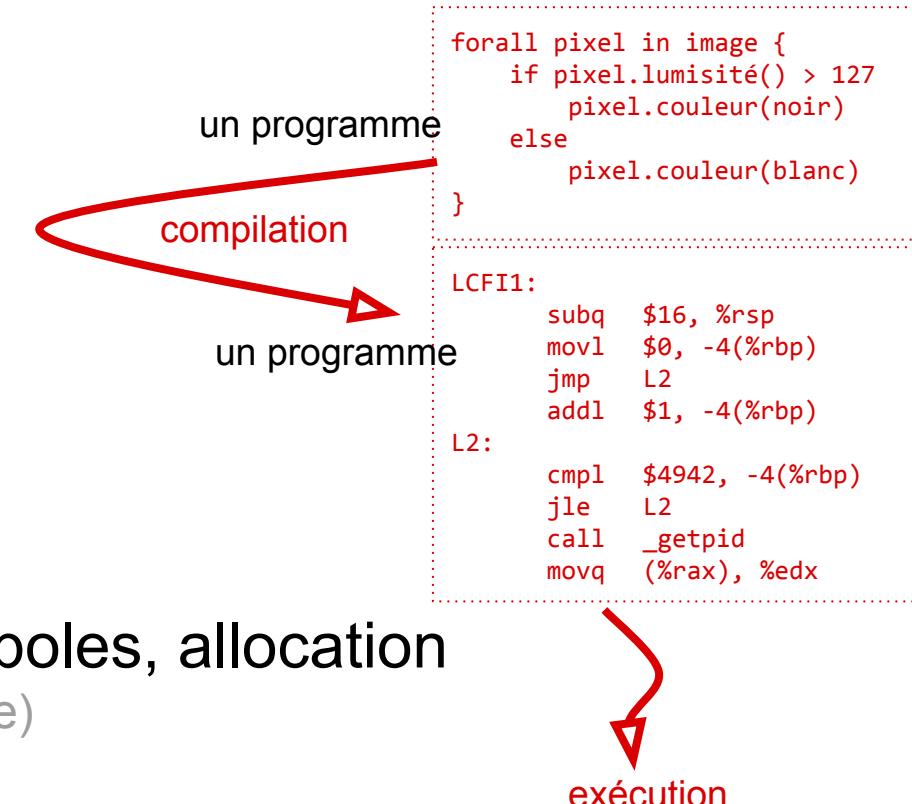


# compiler c'est traduire

- différents langages
  - de "haut" niveau
  - assembleur
- programmation
  - langage de "haut niveau"
- exécution sur la machine
  - assembleur M-10

→ traduire

- variables, table des symboles, allocation  
(et alloc. temporaire, durée de vie)
- structures de contrôle



M-10io

# entrées/sorties

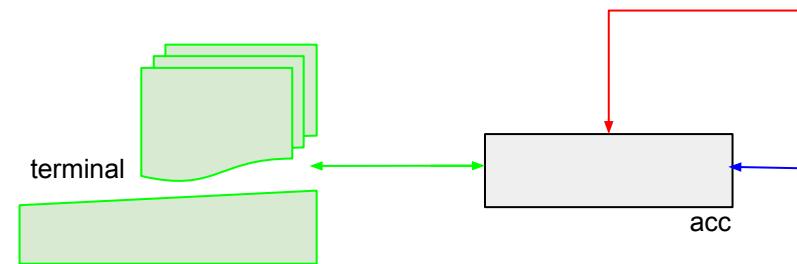
---

- nouveau dispositif matériel

- terminal
  - clavier / écran

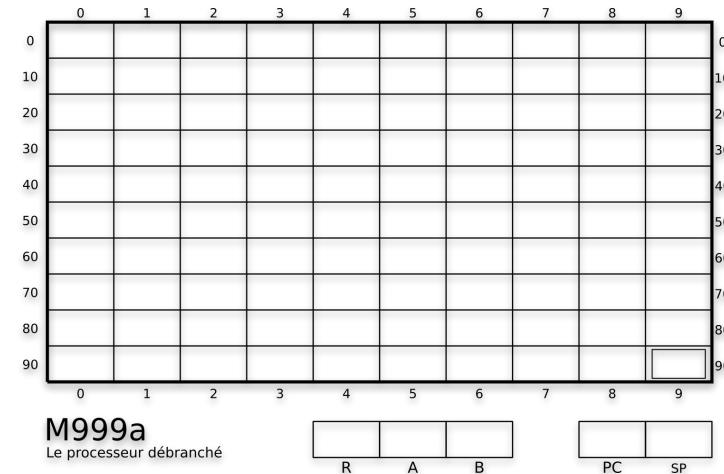
- nouvelles instructions

- **READ**
    - lit une valeur au clavier
  - **PRINT**
    - écrit la valeur de l'accumulateur sur l'écran



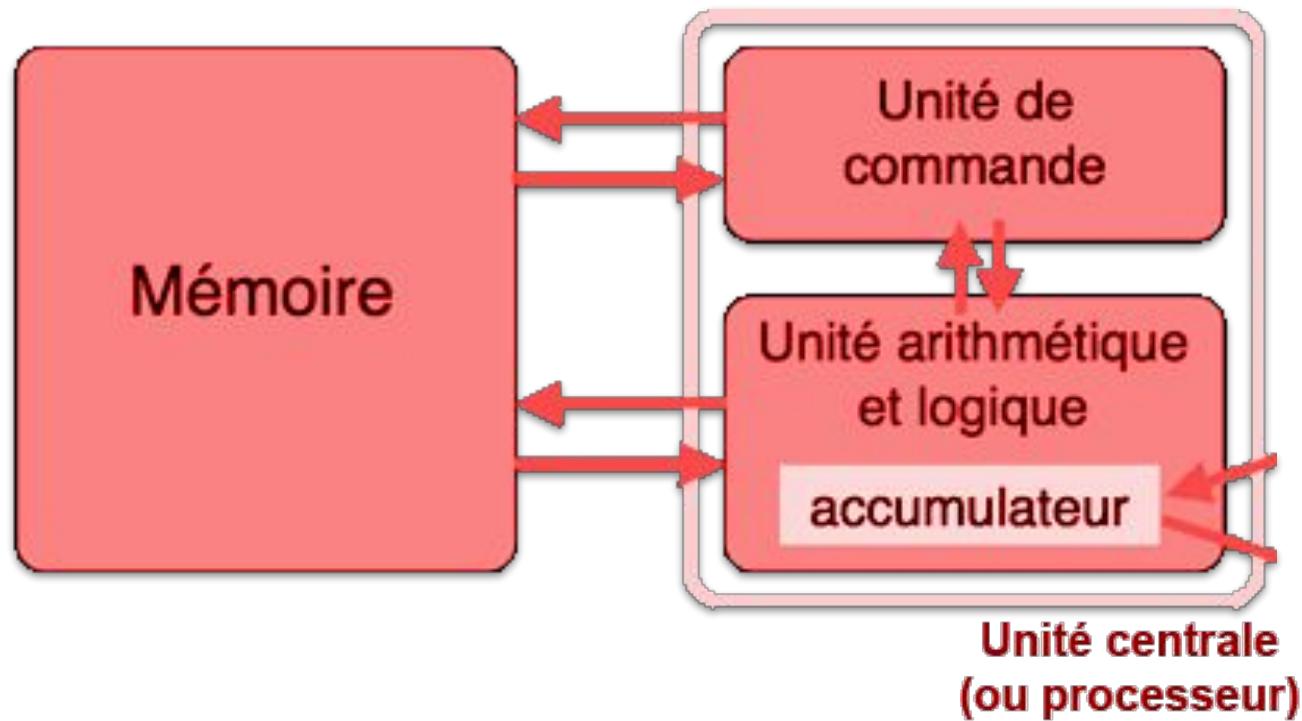
# M-999a

## le processeur débranché



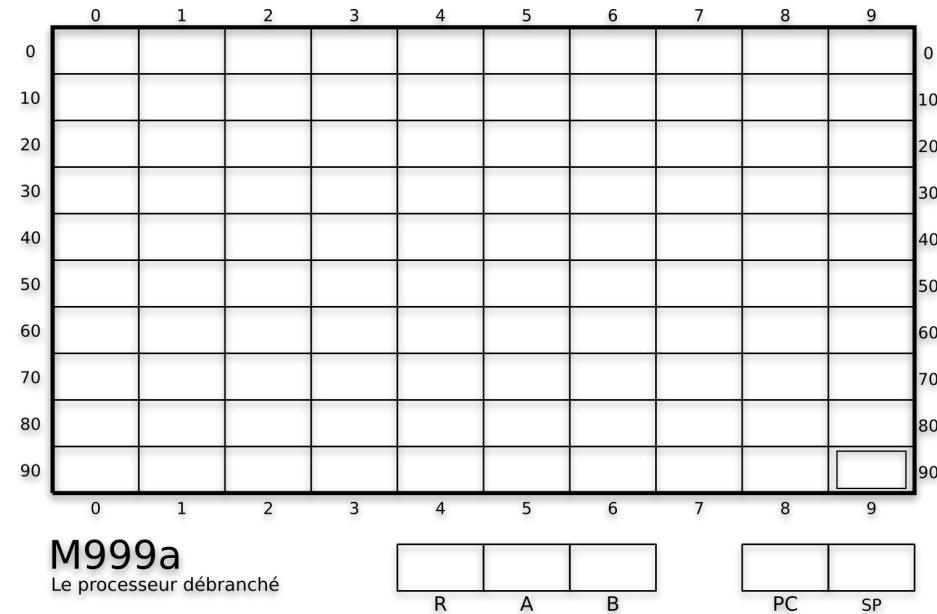
# le processeur M999a

---



# la mémoire de M999a

- adresses – 2 chiffres
- données – 3 chiffres
- 100 “mots” mémoire données et instructions
- valeurs de 000 à 999 entier ou codage instruction



# les registres de M999a

---

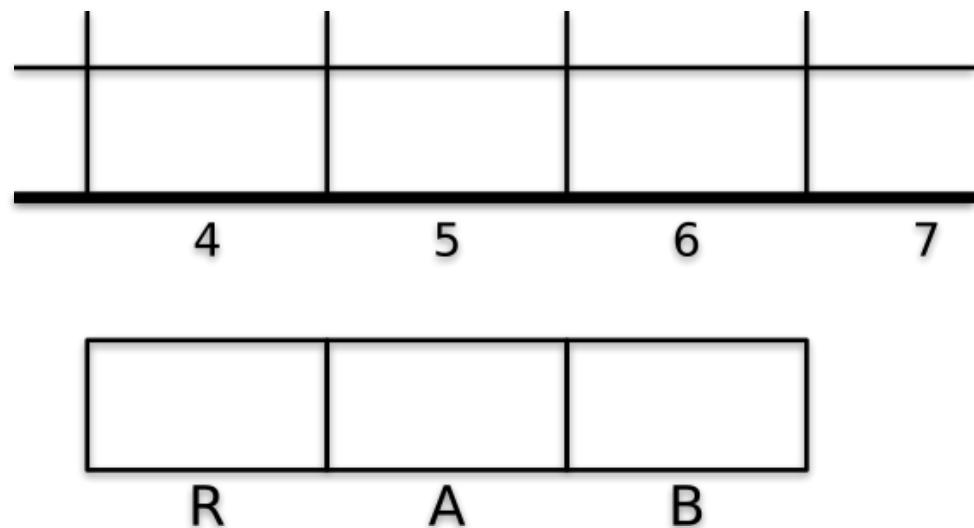
deux registres généraux – A et B

un registre accumulateur/résultat – R

– valeurs de 000 à 999

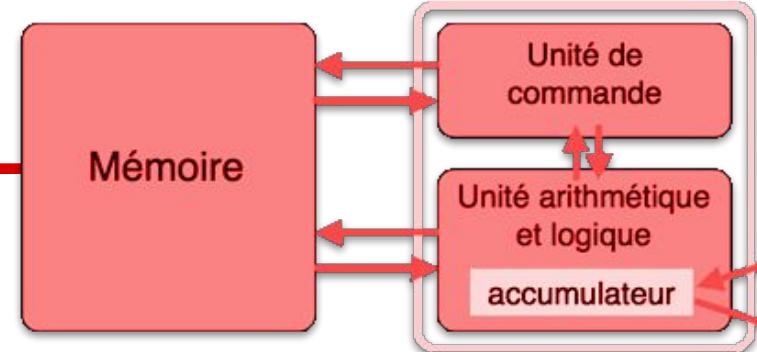
– transfert mémoire

- **LDA** @
- **LDB** @
- **STR** @



# unité arithmétique et logique

– chargée d'effectuer les calculs



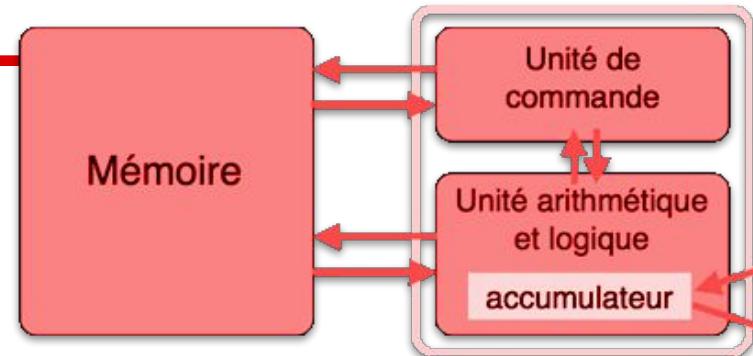
opérandes et résultats dans les registres

- opérandes – A et B
- résultat – R

arithmétique	logique	comparaison
- <b>ADD</b>	- <b>AND</b>	- <b>EQ</b>
- <b>SUB</b>	- <b>NOT</b>	- <b>LSS</b>
- ...	- ...	- ...

# unité de commande

- chargée de la commande, du contrôle

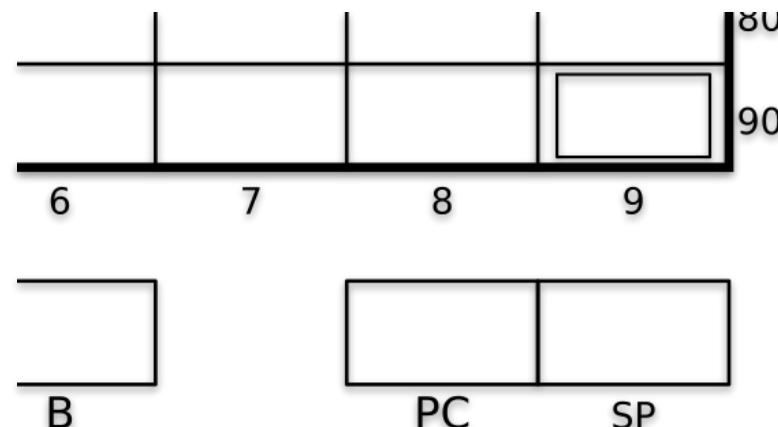


compteur ordinal – PC

adresse d'une case mémoire

instruction courante

- cycle, pour chaque instruction
- **HLT**



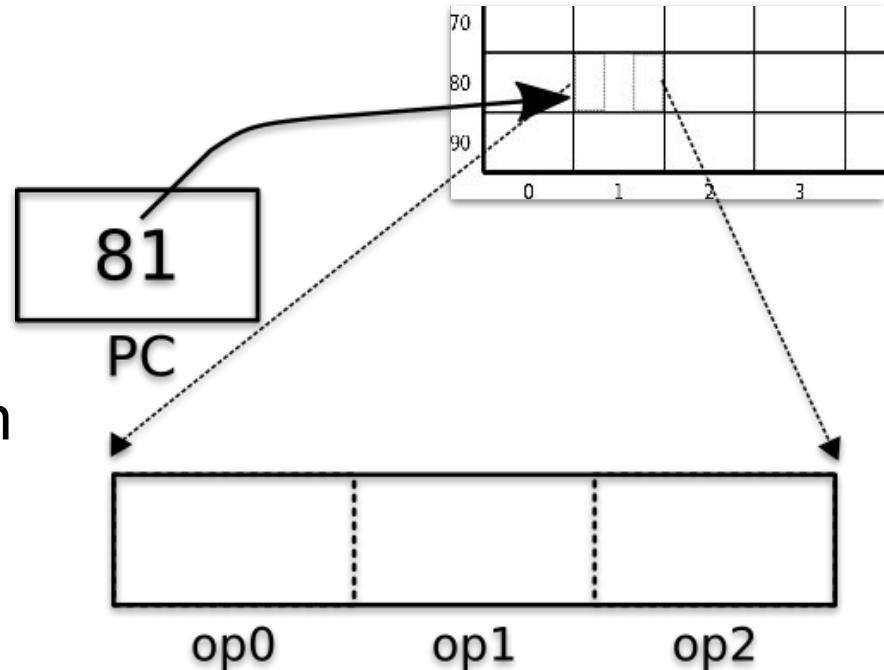
# charge, décode, exécute

unité de commande

1- “fetch”

- chargement de l'instruction

- incrément PC



# charge, décode, exécute

---

## 2- décode

- déterminer quelle est l'opération quelles sont ses opérandes

## 3- exécute

- réalise l'instruction

décode			exécute
op0	op1 op2	mnémonique	instruction à réaliser
0	addr	<b>LDA</b>	copie le mot mémoire d'adresse <i>addr</i> dans le registre A
...	...	...	...
3	0 0	<b>ADD</b>	ajoute les valeurs des registres A et B, produit le résultat dans R
...	...	...	...

# jeu d'instructions

op0	op1 op2	mnémonique	instruction à réaliser
0	<i>addr</i>	<b>LDA</b>	copie le mot mémoire d'adresse <i>addr</i> dans le registre A
1	<i>addr</i>	<b>LDB</b>	copie le mot mémoire d'adresse <i>addr</i> dans le registre B
2	<i>addr</i>	<b>STR</b>	copie la valeur du registre R dans le mot mémoire d'adresse <i>addr</i>
3	0 0	<b>ADD</b>	ajoute les valeurs des registres A et B, produit le résultat dans R
3	0 1	<b>SUB</b>	soustrait la valeur du registre B à celle du registre A, résultat dans R
3	0 2	<b>MUL</b>	multiplie les valeurs des registres A et B, produit le résultat dans R
...	...	...	...
5	<i>addr</i>	<b>JMP</b>	branche en <i>addr</i> (PC reçoit la valeur <i>addr</i> )
6	<i>addr</i>	<b>JNZ</b>	branche en <i>addr</i> si la valeur du registre R est non-nulle

# boot et halt

---

- M999a démarre en PC = 0
- M999a s'arrête si le pointeur d'instruction vaut 99
  - le mnémonique **HLT** est synonyme de **JMP 99**

# programmons M999a

---

- la somme de deux entiers
  - programmons
    - mnémonique
    - traduction en “binaire” (codage des instructions en “décimal” ?)
- un état initial de la mémoire
- exécutons
  - cycles charge, décode, exécute

# entrées/sorties M999a

---

- entrées/sorties “mappées” en mémoire
  - modifier le mot mémoire 99 écrit sur le terminal
  - les valeurs saisies sur le terminal sont lues à l'adresse 99

# copie registre à registre

op0	op1 op2	mnémonique	instruction à réaliser
4	<i>rs rd</i>	<b>MOV</b>	copie la valeur du registre source <i>rs</i> dans le registre destination <i>rd</i>

- les registres sont désignés par

permet d'éviter

- des copies vers/depuis la mémoire
- des allocations mémoire de “variables” temporaires

valeur	registre
0	A
1	B
2	R

# programmons M999a

---

- lire 3 entiers, afficher la somme
- produit non-nul de 2 entiers
  - $(0,0 \rightarrow 0) \quad i, 0 \rightarrow i \quad 0, j \rightarrow j \quad i, j \rightarrow i^*j$
- etc.



# machine von Neumann



- John von Neumann, 1945
  - mathématicien, physicien
- modèle d'architecture pour un ordinateur



→ premiers ordinateurs

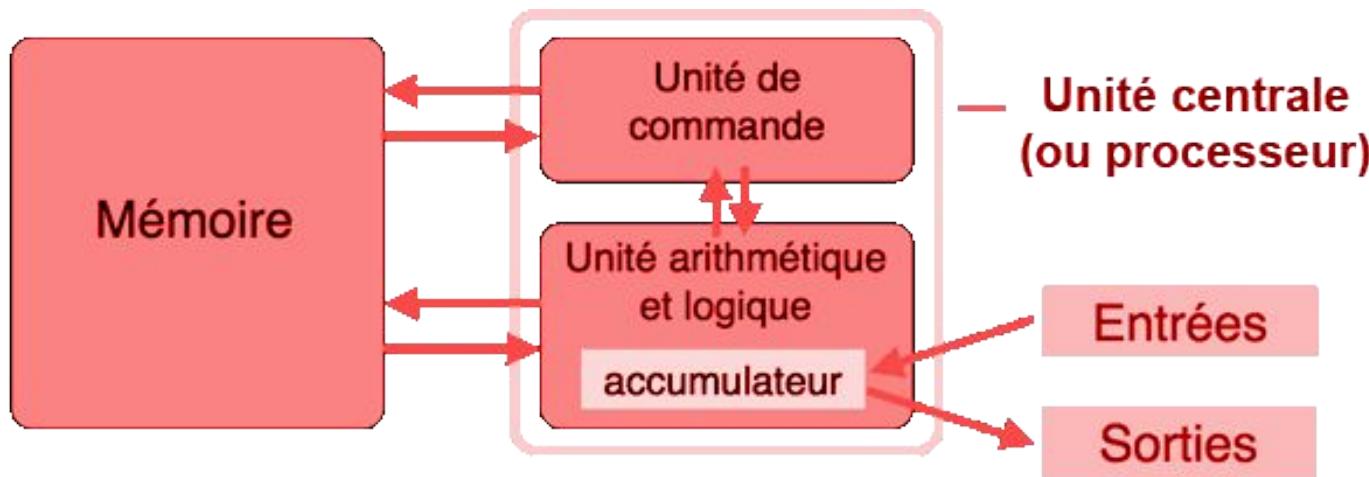
- modèle simple, toujours d'actualité
- machine “universelle”
  - Alan Turing - machine 1936
  - tous les langages de programmation

Algol · APL · ASP · Assembleur · BASIC · BCPL ·  
Shell Unix · C · COBOL · Natural · Forth · Fortran · Go  
· Limbo · Lua · Modula-2 · NQC · NXC · OPL · Pascal  
· Perl · PHP · Rust · PL/I · Tcl / C++ · C# ·  
CoffeeScript · D · Delphi · Eiffel · Groovy · Java ·  
JavaScript · Lisaac · Logo · Objective-C · PHP ·  
Python · Ruby · Scala · Simula · Smalltalk · Visual  
Basic / Haskell · Lisp · Common Lisp · ML · OCaml ·  
Gallina · F# · Standard ML · Opa · Scheme · XSLT /  
Clips · Prolog / Ada · Erlang



# machine von Neumann

- séparation UC / UAL
- programme “enregistré”
  - et non pas externe ruban, cartes...
  - compteur ordinal
- codage des instructions
  - en mémoire “banalisée”
  - programme traité comme une donnée (par d’autres programmes, compilateurs...)





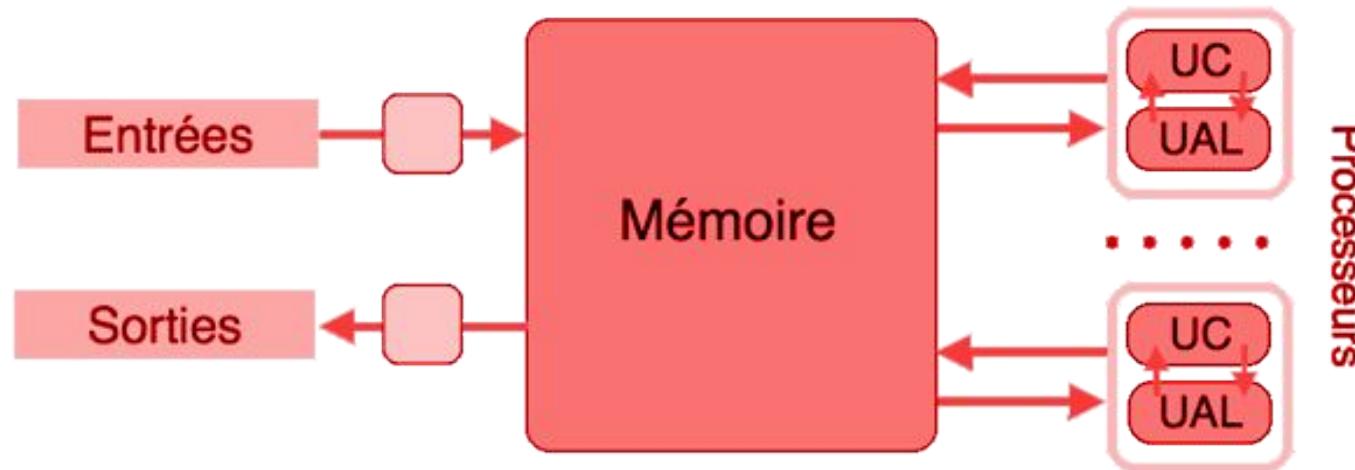
# modèle von Neumann aujourd'hui

- multiples processeurs

- multipro. - multicœurs
- parallélisme
- ↑ performances
- ↓ coût
- ↓ consommation

- entrées/sorties

- pilotées par des UC indépendantes





# assembleur, exécutable

---

- **langage assembleur**
  - langage de bas niveau
  - spécifique à chaque processeur
  - représentation lisible par un humain du langage machine

mnémonique  
label  
valeur entière
- **traduction en code objet, code exécutable**
  - “compréhensible” par le processeur
  - mot dans la mémoire du processeur

binaire  
code-op  
adresse
- **assembleur – traducteur**
  - phase finale de compilation
  - langage haut niveau → assembleur → code objet

asm



# Plus loin avec ces ressources

---

## 1- simulateur de M999a (M-10 ?)

- activité de programmation
- compréhension du fonctionnement d'un ordinateur
- assembleur M999 → code "binaire/décimal" M999a

## 2- notion de variable

- nom / emplacement mémoire
- notion d'état → expliquer la notion de variable

## 3- notion de fonction

- nom associé à un bloc de code
- mécanisme de passage paramètre

→ dévoiler un peu de la mécanique sous-jacente  
pour comprendre des notions de l'informatique



# Les sous-routines

```
def mul x y :  
    return x*y
```

enrichir M-10

# sous-routines, procédure, fonction

bloc d'instruction

≈ une nouvelle instruction

exemple de la multiplication

```
; bloc d'instructions qui
; calcule en @4
; le produit de @2 et @3
mul: load 2
    sto @i
    ...
    ...
    sto 4
    jmp retour
```

```
; utilisation pour
; calculer d = a*b
    load @a
    sto 2
    ...
appel: jmp mul
retour: load 4
        sto @d
```



enrichir M-10

## retour de sous-routines ?

utiliser plusieurs fois cette multiplication ?

- par exemple évaluer  $(a * b) * c$
  - un premier saut à mul pour  $tmp = a * b$
  - un second saut à mul pour  $tmp * c$
- quel saut faire en fin de mul pour revenir à l'appelant ?

todo

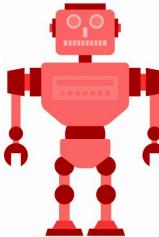
# La variable informatique

$\exists? x / x = x + 1$

**x = x+1**

# un langage pour programmer un (autre) robot

- robot

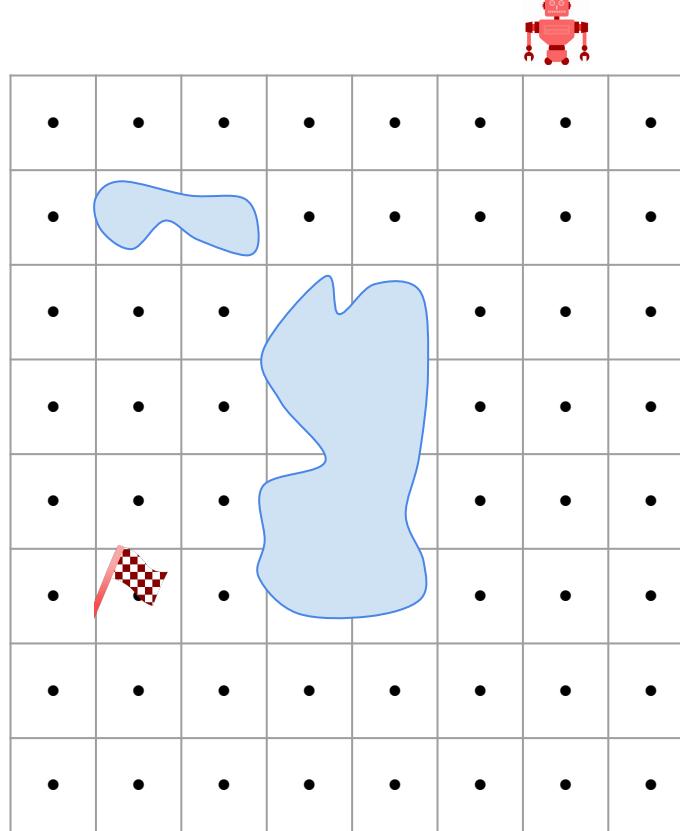
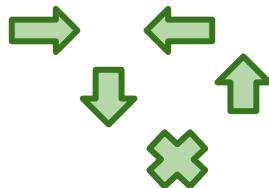


- plan de jeu

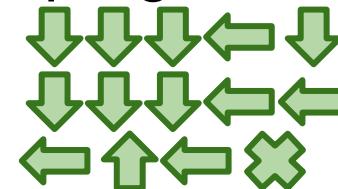
- obstacles
  - objectif

- langage

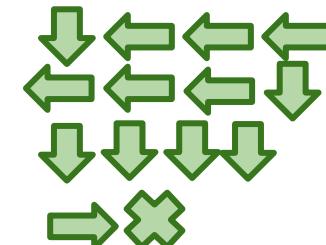
- ## ○ 5 instructions



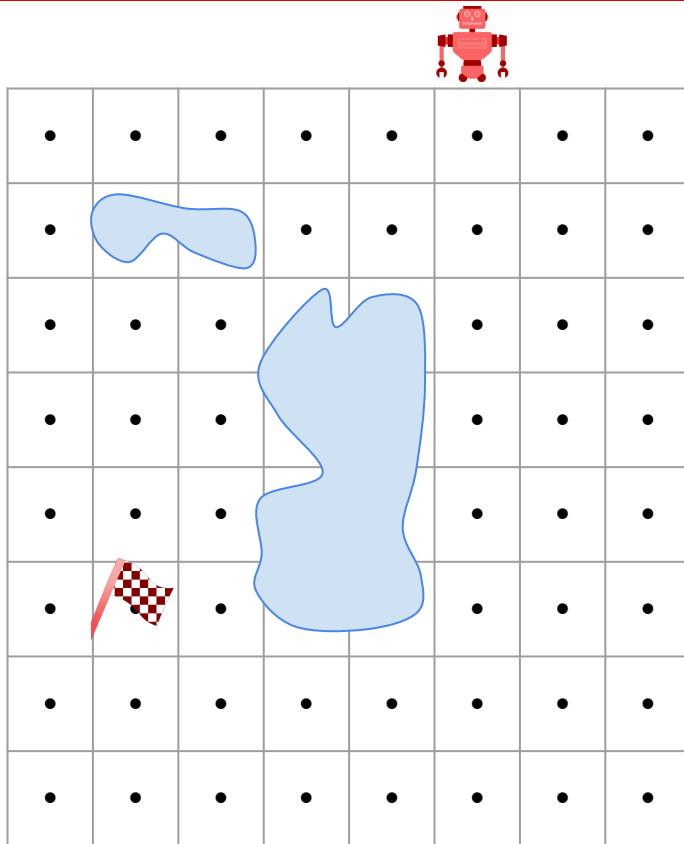
## un programme



un autre  
programme



# un (autre) langage pour programmer un (autre) robot



autre langage

- Avance
- tourne-Droite
- tourne-Gauche
- Fin

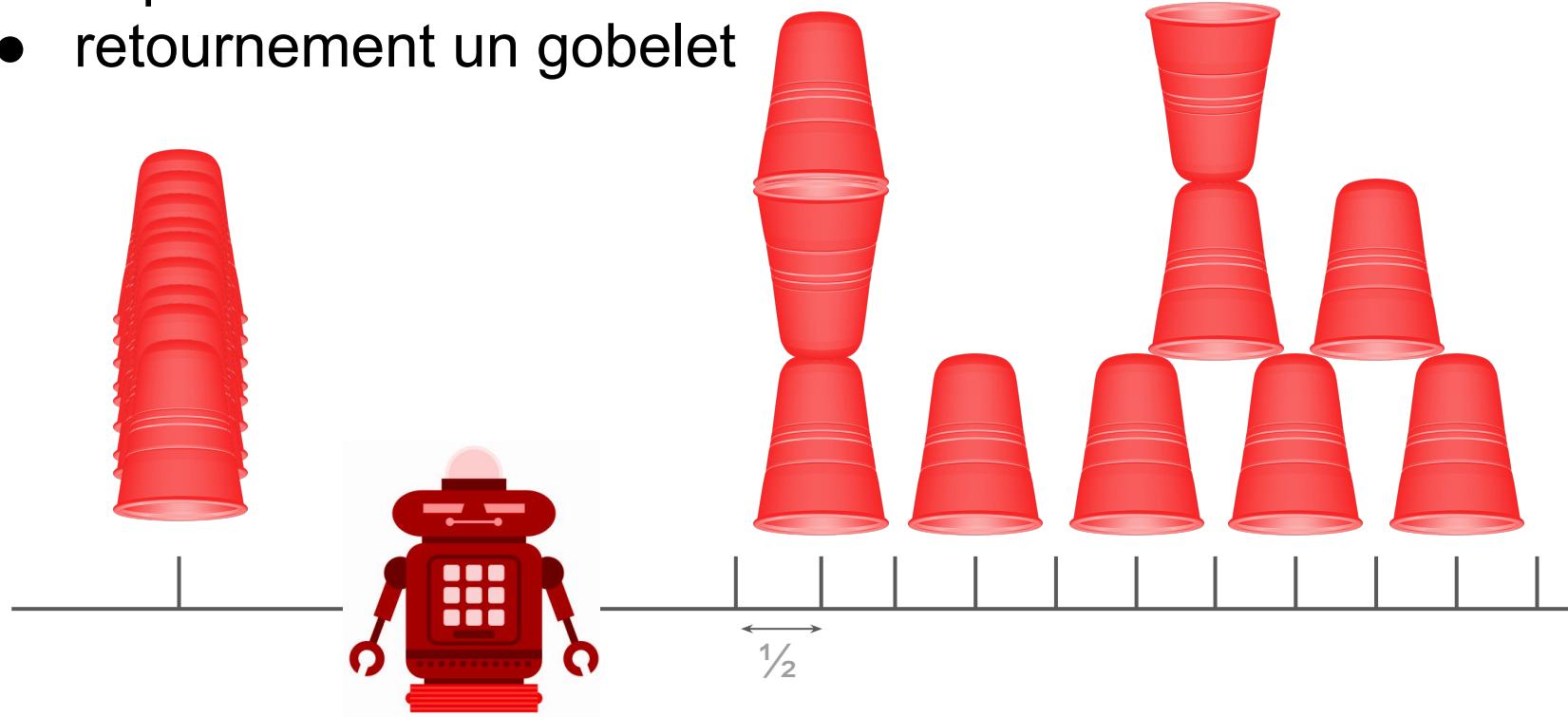
→ programmons ce robot !

A D A A A A A G A  
A G A D A A F

# *gobol — langage gobot*

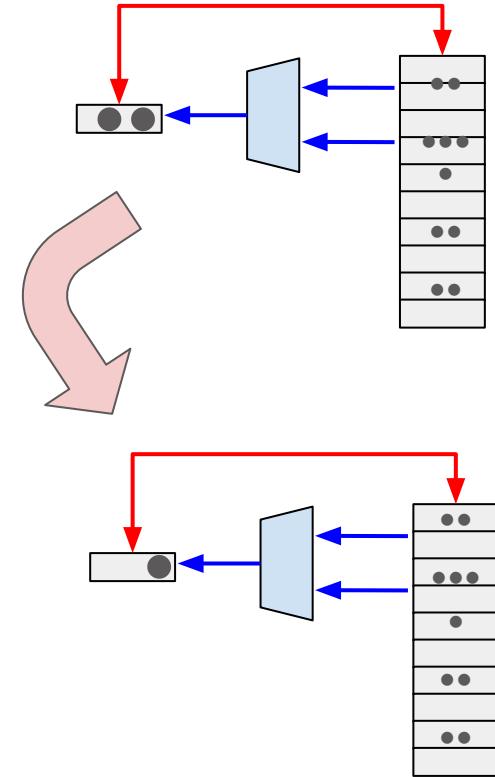
---

- prise / dépôt d'un gobelet
- déplacement
- retournement un gobelet



# la machine débranchée

- programme
  - suite d'instructions
  - une instruction
    - modifie mémoire, ou
    - modifie valeur accumulateur
- modifie l'état de la machine
  - contenu mémoire et de l'accumulateur



# programme – instruction

---

- suite d'instructions
  - **syntaxe** d'une instruction
    - “sucre syntaxique”
    - → LOAD ...
  - **sémantique** d'une instruction
    - effet de l'**exécution** de l'instruction
- effet sur un **environnement**,  
sur un **état**
  - position / orientation du robot
  - tas de gobelet et position de gobol
  - contenu mémoire et de l'accumulateur
  - ...

# et les variables ?

---

- nom auquel on associe une valeur
- état
  - ensemble des valeurs associées aux variables
- valeur d'une variable
  - valeur "associée" à la variable
- instruction
  - modifie l'état
    - = modifier la valeur d'une variable
    - = associer une nouvelle valeur à une variable

a	42
b	12
x	5
y	33

# un détour par les expressions

---

- opérateurs
  - + \* - / ...

- valeurs
  - immédiates : 6 174 ...
  - valeurs des variables : a ...

$$6 * x + a$$

## évaluation d'une expression

- réduction de l'expression à une valeur
  - “calcul”

$$6 * x + a \rightarrow 72$$

a  
42

b  
12

x  
5

y  
33

# instruction d'affectation

---

- sémantique
  - associer une nouvelle **valeur** à une **variable**
- **valeur** ?
  - une **expression** !

1. **évaluation** de l'expression  
→ une valeur
2. **modification** de la valeur de la variable

a	42
b	12
x	5
y	33

# instruction d'affectation

---

- **syntaxe**

- deux éléments
- nom d'une **variable**
- **expression**

b = 6

x = 3 \* a + y

a = a - 2

***syntaxe***

- v := expr
  - v = expr
  - v ← expr

a  
42

b  
12

x  
5

y  
33

# état — notion de variable

---

- une variable : *nom, valeur*
- ensemble des valeurs associées à chaque variable
- la valeur d'une variable  $x$ 
  - dépend de l'état
  - donc dépend du temps
  - ne change pas sans qu'une instruction ne soit exécutée — **affectation**
  - instruction d'affectation

(différent des variables  $x$  en mathématique)

---

— ?

— ...

— !

---

# crédits

## )i(nterstices interstices.info

- *Le modèle d'architecture de von Neumann*, Sacha Krakowiak

groupe InfoSansOrdi

- *M999, le processeur débranché*, Martin Quinson, Philippe Marquet  
[github.com/InfoSansOrdi/M999](https://github.com/InfoSansOrdi/M999)

## images

- <https://pixabay.com/fr/roue-dent%C3%A9e-engins-cog-cog-310906/>
- <https://pixabay.com/fr/ic%C3%B4ne-main-%C3%A9crire-stylo-note-1691335/>
- <https://commons.wikimedia.org/wiki/File:EmacsIcon.svg>
- <https://interstices.info/le-modele-darchitecture-de-von-neumann/>
- <https://commons.wikimedia.org/wiki/File:JohnvonNeumann-LosAlamos.jpg>
- [https://fr.wikipedia.org/wiki/Alan\\_Turing#/media/File:Alan\\_Turing\\_Aged\\_16.jpg](https://fr.wikipedia.org/wiki/Alan_Turing#/media/File:Alan_Turing_Aged_16.jpg)
- <http://www.opengraphicdesign.com/art/retro-robots-in-vector-format/>
- <https://pixabay.com/fr/verre-tasse-en-plastique-656716/>

