

# *Éléments d'informatique – Cours 1. Éléments d'architecture des ordinateurs, mini-assembleur*

Pierre Boudes

19 septembre 2011



This work is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License*.

*Survol du contenu du cours (ce semestre)*

*Architecture de von Neumann*

*Représentation des informations*

*Cycle d'exécution*

*Instructions*

*Trace d'exécution*

*Démos et fin*

# *Survol du contenu du cours (ce semestre)*

## *Survol du contenu du cours (ce semestre)* ✖

- Éléments d'architecture des ordinateurs (+mini-assembleur)

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while



## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, types composés (enregistrements)

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, types composés (enregistrements)
  - Fonctions d'entrées/sorties (scanf/printf)

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, types composés (enregistrements)
  - Fonctions d'entrées/sorties (scanf/printf)
  - Écriture et appel de fonctions

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, types composés (enregistrements)
  - Fonctions d'entrées/sorties (scanf/printf)
  - Écriture et appel de fonctions
  - Débogage

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, types composés (enregistrements)
  - Fonctions d'entrées/sorties (scanf/printf)
  - Écriture et appel de fonctions
  - Débogage
- Notions de compilation
  - Analyse lexicale, analyse syntaxique, analyse sémantique
  - préprocesseur du compilateur C (include, define)
  - Édition de lien

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, types composés (enregistrements)
  - Fonctions d'entrées/sorties (scanf/printf)
  - Écriture et appel de fonctions
  - Débogage
- Notions de compilation
  - Analyse lexicale, analyse syntaxique, analyse sémantique
  - préprocesseur du compilateur C (include, define)
  - Édition de lien
- Algorithmes élémentaires

## *Survol du contenu du cours (ce semestre)*

- Éléments d'architecture des ordinateurs (+mini-assembleur)
- Éléments de systèmes d'exploitation
- Programmation structurée impérative (éléments de langage C)
  - Structure d'un programme C
  - Variables : déclaration (et initialisation), affectation
  - Évaluation d'expressions
  - Instructions de contrôle : if, for, while
  - Types de données : entiers, caractères, réels, tableaux, types composés (enregistrements)
  - Fonctions d'entrées/sorties (scanf/printf)
  - Écriture et appel de fonctions
  - Débogage
- Notions de compilation
  - Analyse lexicale, analyse syntaxique, analyse sémantique
  - préprocesseur du compilateur C (include, define)
  - Édition de lien
- Algorithmes élémentaires
- Méthodologie de résolution, manipulation sous linux

*« L'informatique n'est pas plus la science des ordinateurs  
que l'astronomie n'est celle des télescopes. »*

*E. W. Dijkstra*



# *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de nos processeurs

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de nos processeurs
- De quoi cette machine est-elle faite ?

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de nos processeurs
- De quoi cette machine est-elle faite ?
  - De mémoire (une suite de cases numérotées)

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de nos processeurs
- De quoi cette machine est-elle faite ?
  - De mémoire (une suite de cases numérotées)
  - d'une **unité de calcul**, travaillant sur des **registres**

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de nos processeurs
- De quoi cette machine est-elle faite ?
  - De mémoire (une suite de cases numérotées)
  - d'une **unité de calcul**, travaillant sur des **registres**
  - d'un **bus** système (adresses et données) reliant mémoire et UC



## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de nos processeurs
- De quoi cette machine est-elle faite ?
  - De mémoire (une suite de cases numérotées)
  - d'une **unité de calcul**, travaillant sur des **registres**
  - d'un **bus** système (adresses et données) reliant mémoire et UC
  - ~~De périphériques~~ (on oublie !)

## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de nos processeurs
- De quoi cette machine est-elle faite ?
  - De mémoire (une suite de cases numérotées)
  - d'une **unité de calcul**, travaillant sur des **registres**
  - d'un **bus** système (adresses et données) reliant mémoire et UC
  - ~~De périphériques~~ (on oublie !)
  - La mémoire contient le programme et les données.

# *Représentation en binaire des informations*

## *Représentation en binaire des informations*

### *Definition (bit)*

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).

## *Représentation en binaire des informations*

### *Definition (bit)*

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est constituée de bits.

## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est constituée de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.

## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est constituée de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.
- les instructions du langage machine sont écrites en binaire.

## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est constituée de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.
- les instructions du langage machine sont écrites en binaire.
- le **langage assembleur** est une notation du langage machine plus pratique pour les humains.



## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est constituée de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.
- les instructions du langage machine sont écrites en binaire.
- le **langage assembleur** est une notation du langage machine plus pratique pour les humains.

Nous en verrons un peu plus sur les codages en binaire des données dans un autre cours.

# *Cycle d'exécution*

## *Cycle d'exécution*

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction

## *Cycle d'exécution*

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)

## *Cycle d'exécution*

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)

## Cycle d'exécution

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)
- le contenu de RI est décodé afin de déterminer l'opération à exécuter

## Cycle d'exécution

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)
- le contenu de RI est décodé afin de déterminer l'opération à exécuter
- l'opération est exécutée (le contenu d'un ou plusieurs registres est modifié, ou bien celui d'une case mémoire)

## Cycle d'exécution

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)
- le contenu de RI est décodé afin de déterminer l'opération à exécuter
- l'opération est exécutée (le contenu d'un ou plusieurs registres est modifié, ou bien celui d'une case mémoire)
- Fin du cycle d'exécution et démarrage d'un nouveau cycle



## *Instructions*

Une instruction type comporte un **code d'opération** et, si nécessaire, une ou deux *opérandes* (ou *arguments* de l'opération).

## *Instructions*

Une instruction type comporte un **code d'opération** et, si nécessaire, une ou deux *opérandes* (ou *arguments* de l'opération).

### *Vocabulaire*

Dans l'expression arithmétique usuelle  $3 + 5$ , le signe  $+$  est l'opérateur et les nombres 3 et 5 sont les opérandes.

# *Quelques instructions typiques (Amil) ~~✎~~*

## *Quelques instructions typiques (Amil)*

stop

Arrête l'exécution du programme.

noop

N'effectue aucune opération.

## *Quelques instructions typiques (Amil)*

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture $i$ $r_j$	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture $r_i$ $j$	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .

## Quelques instructions typiques (Amil) ~~✎~~

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture $i$ $rj$	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture $ri$ $j$	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .
saut $i$	Met CP à la valeur $i$ .
sautpos $ri$ $j$	Si la valeur contenue dans le registre $i$ est positive ou nulle, met CP à la valeur $j$ .

## Quelques instructions typiques (Amil)

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture $i$ $rj$	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture $ri$ $j$	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .
saut $i$	Met CP à la valeur $i$ .
sautpos $ri$ $j$	Si la valeur contenue dans le registre $i$ est positive ou nulle, met CP à la valeur $j$ .
inverse $ri$	Inverse le signe du contenu du registre $i$ .
add $ri$ $rj$	Ajoute la valeur du registre $i$ à celle du registre $j$ .
soustr $ri$ $rj$	Soustrait la valeur du registre $i$ à celle du registre $j$ .
mult $ri$ $rj$	Multiplie ...
div $ri$ $rj$	Divise ...

## Quelques instructions typiques (Amil)

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture $i$ $rj$	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture $ri$ $j$	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .
saut $i$	Met CP à la valeur $i$ .
sautpos $ri$ $j$	Si la valeur contenue dans le registre $i$ est positive ou nulle, met CP à la valeur $j$ .
inverse $ri$	Inverse le signe du contenu du registre $i$ .
add $ri$ $rj$	Ajoute la valeur du registre $i$ à celle du registre $j$ .
soustr $ri$ $rj$	Soustrait la valeur du registre $i$ à celle du registre $j$ .
mult $ri$ $rj$	Multiplie ...
div $ri$ $rj$	Divise ...
lecture $*ri$ $rj$	Charge, dans $rj$ , le contenu de la mémoire dont l'adresse est la valeur du registre $i$



## *Trace d'exécution*

On simule pas à pas l'exécution du programme.

## *Trace d'exécution*

On simule pas à pas l'exécution du programme.

## Trace d'exécution

On simule pas à pas l'exécution du programme.

### *Programme*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12

## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?

## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				

## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			

## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0

2. lecture 11 r2

3. soustr r2 r0

4. sautpos r0 8

5. lecture 10 r2

6. add r2 r0

7. saut 4

8. ecriture r0 12

9. stop

10. 14

11. 5

12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				
sautpos r0 8	4	<b>8</b>					

## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				
sautpos r0 8	4	<b>8</b>					
ecriture r0 12	5	9					9

## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				
sautpos r0 8	4	<b>8</b>					
ecriture r0 12	5	9					9
stop	6	10					

# *Démos et fin*