Math93.com

Architecture Matérielles et systèmes d'exploitation

Table des matières

Extraits du programme

— **B.O.** : "**Modèle d'architecture séquentielle (von Neumann)** :
Distinguer les rôles et les caractéristiques des différents constituants d'une machine. Dérouler l'exécution d'une séquence d'instructions simples du type langage machine. La présentation se limite aux concepts généraux. On distingue les architectures monoprocesseur et les architectures multiprocesseur. Des activités débranchées sont proposées. Les circuits combinatoires réalisent des fonctions booléennes."

— **B.O.** : "**Systèmes d'exploitation** :
Identifier les fonctions d'un système d'exploitation. Utiliser les commandes de base en ligne de commande. Gérer les droits et permissions d'accès aux fichiers. Les différences entre systèmes d'exploitation libres et propriétaires sont évoquées. Les élèves utilisent un système d'exploitation libre. "

I Histoire des ordinateurs

Des compléments sur : www.math93.com.

I.1 Machines à programmes externes

I.1.1 Machines électromagnétiques

— L'allemand Konrad Zuse achève le Z1 en 1938 puis le Z3 en 1941 qui lisait son programme sur une bande perforée. Le Z3 utilisait le calcul en virgule flottante et réalisait 3 ou 4 additions à la seconde.

— En 1944, L'américain Howard H . Aiken construit l'ordinateur électromécanique **Mark I** qui pesait 5 tonnes.

I.1.2 Machines électroniques

— L'apparition des **tubes à vides** marque le début de l'électronique.

— John Vincent Atanasoff en 1942 construit la première machine électronique sans l'achever complètement.

— Entre 1943 et 1945 les Britaniques Max Newmann et Tomy Flowers utilisent les premiers ordinateurs à tubes à vides nommés **Colossus** pour déchiffrer le code de Lorenz employé par les Allemands. Notons que cette information n'a été révélée qu'en 1970 !

— Le célèbre **ENIAC** de John W. Mauchly et J. presper Eckert est achevé en 1945 et permet avec ses 18 000 tubes à vide d'effectuer des calculs balistiques.

I.2 Machines à programmes enregistrés

— En 1948 apparaissent les premières machines à programmes enregistrés, ancêtres directs des ordinateurs actuels. Les données et programmes résident en mémoire. Ces machines sont basés sur les travaux de Mauchly, Eckert et von Neumann.

— Au début des années 1950 apparaisent les premiers ordinateurs commerciaux avec **IBM, DEC** et **Bull** (en France depuis 1931).

I.3 Du mini-ordinateur à la micro-informatique

I.3.1 Miniaturisation

— **Le transistor apparaît en 1947** à la place des tubes à vide et on commence à le fabriquer à faible coût au milieu des années 1950.

— Le **circuit intégré** apparaît en 1958.

— En 1971 le premier **microprocesseur** voit le jour, c'est l'Intel 4001. L'informatique s'ouvre alors aux particuliers.

— De multiples machines sont commercialisées : l'Altair 8008, l'Apple II (1977), l'IBM PC (1981), le ZX 81 (1981), le commodore 64 (1982), le Macintosh (1984) ...

— Le **ZX 81** est considéré à son époque comme le premier ordinateur familial en kit en France, sa résolution et sa capacité mémoire (1 ko) ne permettait pas énormément de prouesses au niveau des jeux.

I.4 Langage est système d'exploitation

Langages de programmation

— Après le premier compilateur conçu en 1951 par Grace Hopper, le langage **Fortran** est spécifié en 1954 et achevé en 1956 par John Backus.

— Suivent ensuite : le **Lisp**, le **Cobol** et le **Basic** en 1964. Puis de 1970 à 1980 : le C (1972), le ML (1973) dont est issu Caml, Ada (1983) et C++ (1986).

— La première version de **Python** date de 1991 (Guido van Rossum) et JavaScript a été publié en 1995.

Les systèmes d'exploitation

— Au milieu des années 1960, chaque constructeur développe son propre système d'exploitation : OS/360, puis MSV chez IBM, système Unix (1970) chez AT&T (Bell ...)

— **MS-DOS** écrit par Microsoft pour IBM s'impose. Il est suivi par **Windows** en 1985.

— Le 27 septembre 1983, Richard Stallman dévoile son projet de développer un système d'exploitation compatible UNIX appelé **GNU** - acronyme récursif (la forme développée du sigle contient sa forme réduite) qui signifie en anglais " GNU's Not UNIX " (littéralement, " GNU n'est pas UNIX ") , en invitant la communauté hacker à le rejoindre et participer à son développement.

Exercice 1

Question 1

Les tubes à vide, volumineux et peu fiables, ont été remplacé au milieu des années 1950 par :

a. les cartes perforées **b.** les tubes à plasma **c.** les transistors **d.** les cartes magnétiques

Question 2

Le premier microprocesseur (Intel 4004) a permis :

a. l'ouverture du marché aux particuliers **b.** de déchiffrer le code de la machine Enigma **c.** de permette les voyages spatiaux **d.** de déchiffrer le code de César

Question 3

Le premier langage informatique de haut niveau, autre que le langage machine a été (en 1954) :

a. Fortran **b.** Python **c.** C **d.** Java

II Architecture de von Neuman

Lire le cours plus complet présent sur la page du site math93.com : **https://urlz.fr/bs9A**.

Objectifs

- Décrire et présenter l'architecture von Neumann.
- Présenter la notion de cycle d'instructions et de pipeline.

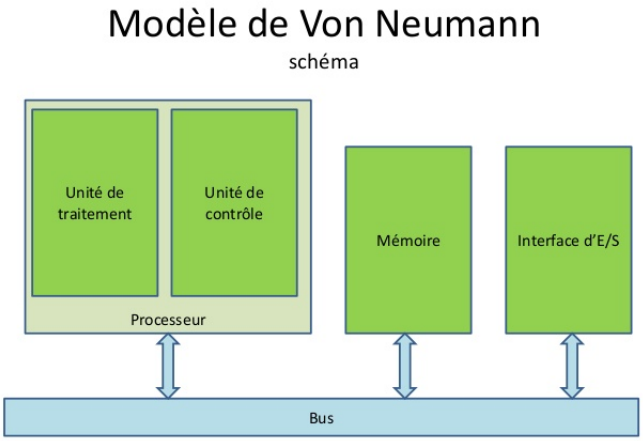
L'architecture de tous les ordinateurs actuels est conforme à un schéma qui a assez peu évolué depuis les premiers ordinateurs électronique à tubes à vide de 1945 (Colossus et ENIAC). Ce modèle est dit de **von Neumann**.

II.1 L'architecture de von Neumann

II.1.1 Les principales structures

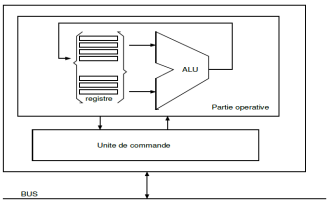
Dans l'architecture de von Neumann, un ordinateur est constitué de quatre parties distinctes :

1. le **CPU** : **Central Processing Unit** (unité centrale de traitement) appelé aussi processeur.
2. la **mémoire** où sont stockés les données et les programmes ;
3. des **bus** qui sont des fils conduisant des impulsions électriques et qui relient les différents composants.
4. des **entrées-sorties** (E/S ou I/O input/Output) pour échanger avec l'extérieur.



II.1.2 Les sous-structures

- Les échanges entre la mémoire et les registres du CPU se font **via des bus** selon une chronologie **organisée par l'horloge** et suivant la nature des échanges : données ou adresses.
- Un programme est enregistré dans la mémoire.
- L'**adresse** (un entier) **de l'instruction** en cours de traitement est stockée dans une mémoire interne au processeur, le **cp** ou **pc** : « *registre compteur de programme* »
- La **valeur de cette instruction** (un entier) est stockée dans une autre mémoire interne, le ri : « registre d'instruction » .
- Le **CPU** dispose aussi de mémoires internes dans le « banc de registres » où sont placées données du programme avant utilisation.
- L'**UAL** ou **ALU** (Unité Arithmétique et Logique) effectue les opérations arithmétiques et logiques, sur les données et les adresses, en interprétant les impulsions électriques.



II.2 Le rôle de l'horloge

Les traitements réalisés par l'ordinateur sont faits sur des représentations binaires des données et des traitements à réaliser, et ce en calculant des fonctions logiques.

II.2.1 Cadence d'un processeur

- Le CPU dispose d'une horloge qui cadence l'accomplissement des instructions et dont l'unité est appelée cycle.
- La fréquence s'exprime en GigaHertz (GHz), elle signifie le nombre d'opérations que fait le processeur en une seconde.
- **3 GHz (GigaHertz)** : **3 milliards** d'opération à la **seconde**. En clair, elle influe sur la vitesse de fonctionnement du processeur.
- En 2020, les processeurs tournent entre 1,5 et 3 GHz. Certains atteignent 3,6 GHz mais la course à la fréquence à pris fin depuis 2005 environ car au-delà d'un certain cap, la chaleur dégagée est trop importante et perturbe la lecture des tensions.

II.2.2 Cycle d'instructions

Le pipeline d'instruction
Dans un processeur, 5 étapes sont nécessaires pour traiter une instruction. Chacune de ces 5 instructions est exécuté lors d'un cycle.

- **LI** (ou **IF - Instruction Fetch**) : charge l'instruction à exécuter dans le pipeline (fetch = aller chercher).
- **ID** : décoder l'instruction ;
- **EX** : exécuter l'opération dans l'UAL ;
- **MEM** : accéder à la mémoire en lecture ou en écriture ;
- **ER** (ou **WB, Write Back**) : écrire le résultat dans un registre.

Pour gagner du temps, le processeur n'exécute pas les instruction de façon séquentielle mais il exécute simultanément plusieurs instructions qui sont à des étapes différentes de leur traitement.
C'est le **pipeline d'instruction**.

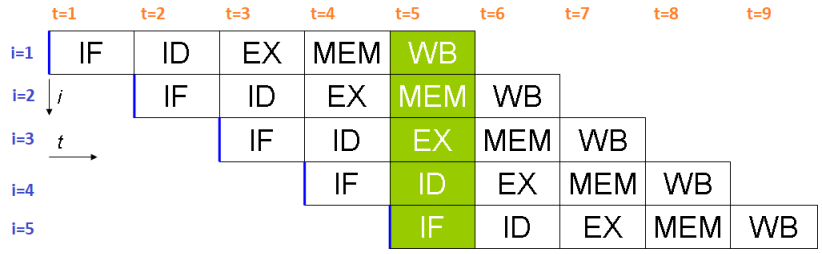
Grâce au pipeline, le traitement des instructions nécessite au maximum les cinq étapes précédentes. Dans la mesure où l'ordre de ces étapes est invariable (LI, DI, EX, MEM et WB), il est possible de créer dans le processeur un certain nombre de circuits spécialisés pour chacune de ces phases.

Analogie avec une chaîne de production
Le pipeline est un concept s'inspirant du fonctionnement d'une ligne de montage. Considérons que l'assemblage d'un véhicule se compose de trois étapes (avec éventuellement des étapes intermédiaires) :

- 1. Installation du moteur. 2. Installation du capot. 3. Pose des pneus.
- Un véhicule dans cette ligne de montage ne peut se trouver que dans une seule position à la fois.
- Une fois le moteur installé, le véhicule V1 continue pour une installation du capot, laissant le poste « installation moteur » disponible pour un prochain véhicule V2. Le véhicule V1 se fait installer ses pneumatiques (roues) tandis que le second V2 est à l'étape d'installation du capot. Dans le même temps un véhicule V3 commence l'étape d'installation du moteur.
- Si l'installation du moteur, du capot et des roues prennent respectivement 20, 5 et 10 minutes, la réalisation de trois véhicules prendra, s'ils occupent un à un toute la chaîne de production, $\underline{105 \text{ minutes}} = (20+5+10) \times 3$. Si on place un véhicule dans la chaîne de production dès que l'étage auquel le véhicule doit accéder est libre (principe du pipelining), le temps total pour réaliser les trois véhicules est de $\underline{75 \text{ minutes}}$.

Exemple d'un pileline
En supposant que chaque étape met 1 cycle d'horloge pour s'exécuter, il faut normalement 5 cycles pour exécuter une instruction, 15 pour 3 instructions :

En utilisant la technique du pipeline, notre processeur peut alors contenir plusieurs instructions, chacune à une étape différente. Il faut 9 cycles pour exécuter 5 instructions. À t = 5, tous les étages du pipeline sont sollicités, et les 5 opérations ont lieu en même temps.



Le premier ordinateur à utiliser cette technique est l'**IBM Stretch**, conçu en 1961.

Exercice 2

Question 4

Un pipeline d'instruction permet :

a. de faire passer les instructions dans un tunnel

b. d'exécuter des instructions séquentiellement

c. d'exécuter des instructions simultanément

d. d'exécuter des instructions randomisées

Question 5

L'UAL :

a. permet de gérer la mémoire

b. permet de gérer les entrées/sorties

c. est une adresse internet

d. est l'endroit où les calculs sont effectués

III Mémoire et langage machine

Lire le cours et effectuer le TD présent sur la page du site math93.com : **https://urlz.fr/bs9w**.

Objectifs

- Décrire l'organisation de la mémoire.
- Introduire la notion de registres.
- Présenter et introduire le langage de bas niveau, l'assembleur.

En son coeur, la machine effectue des calculs à partir d'instructions simples en binaire qui peuvent être directement traduites dans un langage, l'assembleur. L'assembleur prend, traite et remplit des cases mémoires.

III.1 Organisation de la mémoire

III.1.1 Les différents types de mémoire

Il existe de nombreux mécaniques de mémoires qui se distinguent par leur durabilité (volatile ou permanente), leur mode d'accès (par adresse ou dans l'ordre de rangement).

1. **Mémoire vive RAM (Random Acces Memory)**.
   La mémoire vive (RAM) est une mémoire volatile (qui perd ses données lorsqu'on coupe son alimentation électrique). Il s'agit des registres, des mémoires cache, de la mémoire centrale.
   Il y a deux types principaux de mémoire vive :
   - la mémoire vive dynamique (DRAM) qui, même sous alimentation électrique, doit être réactualisé périodiquement pour éviter la perte d'information ;
   - la mémoire vive statique (SRAM) qui n'a pas besoin d'un tel processus lorsque sous alimentation électrique.

2. **Mémoire morte ROM (Read-Only Memory)**.
   La mémoire morte est une mémoire non volatile (mémoire rémanente qui conserve ses données même lorsqu'on coupe son alimentation électrique). Par exemple les disque SSD (Solid State Drive), les disques magnétiques.

   Les mémoires mortes sont utilisées, entre autres, pour stocker :
   - les informations nécessaires au démarrage d'un ordinateur (BIOS, instructions de démarrage, microcode) ;
   - des tables de constantes ou des tables de facteurs de conversion ;

   Elle fait aussi partie des microprogrammes présents dans les ordinateurs et la plupart des appareils électroniques (smartphone, baladeur et autres lecteurs de CD/DVD) mais aussi la plupart des appareils programmables (TV, réveil, machine à laver, lave vaisselle, etc.).

   Le temps d'accès à la mémoire morte est de l'ordre de grandeur de 150 nanosecondes comparativement à un temps d'accès d'environ 10 nanosecondes pour la mémoire vive.

III.1.2 Les registres

— Un registre est un emplacement de mémoire interne à un processeur. Les registres se situent au sommet de la hiérarchie mémoire : il s'agit de la mémoire la plus rapide d'un ordinateur, mais dont le coût de fabrication est le plus élevé car la place dans un microprocesseur est limitée.

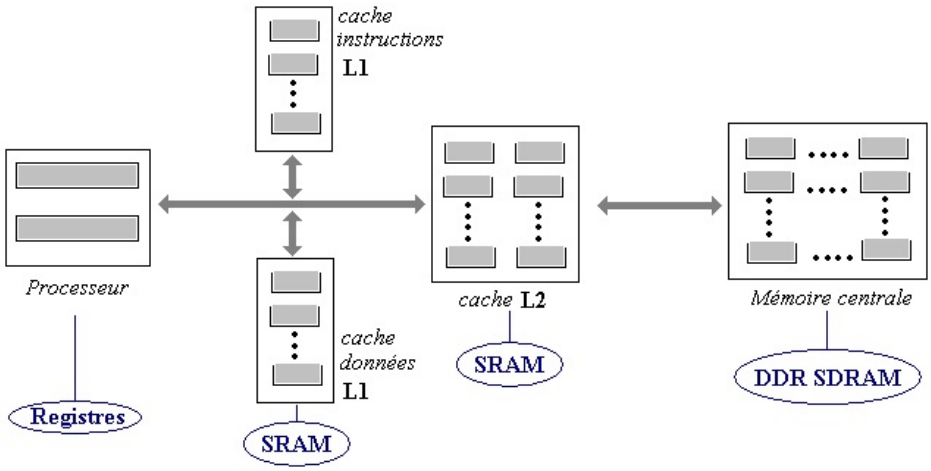
— L'accès par le processeur à une information située dans la DDR SDRAM de la mémoire centrale est 100 fois plus lente qu'un accès à une information contenue dans un registre.

— Les registres sont utilisés pour stocker des opérandes et des résultats intermédiaires lors des opérations effectuées dans l'UAL (Unité Arithmétique et Logique) du processeur.

— La plupart des PC actuels ont des registres de tailles 64 bits.

III.1.3 Mémoires centrales et mémoire cache

1. **La mémoire centrale : RAM (Random Acces Memory)**
   — La mémoire centrale est une mémoire vive qui contient les programmes en cours et les données qu'il utilise. Elle est de taille importante (plusieurs Go).
   — Elle est organisée en céllules qui contiennent chacune une donnée ou une instruction repérée par un entier : un adresse mémoire.
   — Le temps d'accès à chaque cellule est identique, on parle improprement de mémoire à accès aléatoire ou RAM (Random Acces Memory) mais on devrait plutôt parler de mémoire à accès direct.
2. **La mémoire cache**
   — Pour pouvoir adapter la très grande vitesse du processeur à celle bien plus faible de la RAM, on place entre les deux une mémoire très rapide, la mémoire cache.
   — Il existe souvent plusieurs niveaux de mémoire cache : L1, L2 ...
   — Généralement la mémoire cache de niveau L1 et celle de niveau L2 sont regroupées dans la même puce que le processeur (cache interne).



III.2 Assembleur et jeu d'instructions

Un programme écrit dans un langage de **haut niveau** (comme Python) éloigné du langage machine (dit de **bas niveau**) dépend le moins possible du processeur et du système d'exploitation.

Si on ouvre un fichier exécutable avec un éditeur (hexadécimal), on obtient :

01ebe81406372747374756666 2e6305f5f43544f525f4c 5f05f5f44544f525f4c4953545f5f05f5f4a43525f4c49
53545f5f05f5f646f5f676c6f62616c5f64746f72735f6 75780636f6d706c657465642e36353331064746f725f69

C'est une suite d'instructions comme **01ebe814**, que l'on peut traduire directement de façon plus lisible :

add $t7, $t3 , $sp

C'est ce qu'on appelle l'assembleur. Par exemple ici l'instruction **add**, va écrire dans le registre t7 la somme des valeurs des registres t3 et sp.
L'assembleur est donc une représentation du langage machine.
Il y a autant d'assembleurs que de type de processeurs différents.

III.3 TP assembleur et exercices

Exercice 3

Il faut ensuite faire le TP présent sur la page :

https://urlz.fr/bs9w.

IV Linux et Bash

Objectifs

- Une introduction au système Linux.
- Initiation à l'usage du Bash, l'interpréteur de commandes de ce système.
- Identifier les moyens de naviguer dans l'arborescence d'un système.
- Manipuler les entrées/sorties et les redirections.
- Gérer les droits et permissions d'accès aux fichiers.

IV.1 Le système Linux

IV.1.1 Histoire

- Le 27 septembre 1983, Richard Stallman dévoile son projet de développer un système d'exploitation compatible UNIX appelé **GNU** - acronyme récursif (la forme développée du sigle contient sa forme réduite) qui signifie en anglais " GNU's Not UNIX " (littéralement, " GNU n'est pas UNIX ") , en invitant la communauté hacker à le rejoindre et participer à son développement.
- Le noyau **Linux** a été créé en 1991 par **Linus Torvalds**, un informaticien américano-finlandais né le 28 décembre 1969 à Helsinki en Finlande.
- Le manchot Tux, dessiné par Larry Ewing en 1996, devient la mascotte du projet.
- À l'origine, le noyau **Linux** a été développé pour les ordinateurs personnels compatibles PC, et devait être accompagné des logiciels GNU pour constituer un système d'exploitation.
- Depuis les années 2000, le noyau Linux est utilisé sur du matériel informatique allant des téléphones portables aux super-ordinateurs, et n'est pas toujours accompagné de logiciels GNU. C'est notamment le cas d'Android, qui équipe plus de 80% des smartphones.

Linus Torvalds

IV.1.2 Développement

Linux s'est développé rapidement et à donné naissance à de nombreux systèmes d'exploitation partageant le même noyau dont Ubuntu, Débian et le système Android.
Le noyau Linux est écrit en langage C et contient des millions de lignes de code.

Tux

IV.1.3 Installation

L'installation de Linux a longtemps été réservée à des experts mais cela est bien plus aisé de nos jours.
De nombreuses personnes choisissent de configurer leurs machines et sous Windows, et sous Linux, l'utilisateur choisi lors du démarrage sur quel système il veut travailler.

IV.2 Le Bash

Nous allons ici présenter le Bash (Bourne Again Shell), l'interpréteur de commandes le plus courant sous Linux, sous mac OSX et même sous Windows 10 via le **PowerShell** (ou la commande cmd dans l'invité de commandes).

IV.2.1 Les commandes de bases Linux ou Windows PowerShell

Commandes Unix	PowerShell (si différent)	Description	Exemple
cd		Se déplacer dans l'arborescence (ou parfois chdir, abréviation de *change directory*)	cd home\NSI\eleve
cd ..		Se déplacer dans l'arborescence dans le répertoire parent	cd ..
pwd		Cette commande permet d'afficher l'emplacement ou on se situe actuellement dans la hiérarchie FHS (*Filesystem Hierarchy Standard*).	pwd
rmdir		Cette commande permet de supprimer un répertoire.	rmdir NSI
mkdir		Cette commande permet de créer un répertoire.	mkdir NSI
cd ~ (ou cd)	cd\	Se déplacer dans le répertoire d'accueil (HOME sous Linux) des utilisateurs. Sous Windows c'est souvent un prénom ou un dossier Public	cd ~ (ou cd)
cat	type	Visualiser le contenu d'un fichier (*catenate*, synonyme de *concatenate* : « concaténer »)	cat fichier1
cp	cpi	Copier des fichiers ou des répertoires (de *copy item*) *Remarque : on peut donner le chemin absolu du fichier copie si il n'est pas dans le même répertoire*	cp fichier1 copie2 cpi fichier1 copie2
echo		Afficher un message ou le contenu d'une variable (remplace le printf, **print f**ormated)	echo "Bonjour"
ls	dir	Lister le contenu du répertoire courant (de *list* en anglais)	ls
mv	mi	Déplacer des fichiers ou des répertoires (de *Move-Item*)	mv fichier1 deplace2
mv	rni	Renommer fichiers ou répertoires (de *ReName-Item*)	mv fichier1 rename2 rni fichier1 rename2
rm	ri	Effacer des fichiers ou des répertoires	
touch	echo $null > >	Actualise la date d'accès et/ou de modification d'un fichier (le *timestamp*). Si le fichier n'existe pas, est créé un fichier vide. Sous PowerShell on utilise echo $null > > qui ajoute une ligne vide au fichier.	touch fichier echo $null > > fichier
help		Permet d'avoir la description d'une commande	help cd

Remarque

De nombreux compléments et exemples sur les sites :

- https://www.sitedetout.com/tutoriels/commandes-linux-de-base/
- https://windows.developpez.com/cours/ligne-commande/?page=page_4
- https://en.wikipedia.org/wiki/List_of_Unix_commands
- **Compatibilité Unix / PowerShell** : https://urlz.fr/bugu

IV.2.2 Naviguer dans une arborescence

Naviguer dans une arborescence

Soit l'arborescence suivante (ce sont des répertoires) : **C :\Users\lycee\nsi\eleve**

- Pour aller dans le répertoire **eleve** situé dans le répertoire **nsi** à partir du répertoire **lycee** on utilise soit :
  - un "chemin relatif" qui part du répertoire où l'on se trouve : **cd NSI\eleve**
  - ou un chemin absolu c'est à dire un chemin qui part de la racine on peut faire : **cd C :\Users\lycee\nsi\eleve**
- Pour remonter dans le répertoire parent on utilise deux points ".." : **cd ..**
- Si on veut copier le fichier a.txt du répertoire nsi, dans le répertoire eleve alors que l'on se trouve dans le répertoire eleve on peut faire : **cpi . ./ a.txt b.txt**

	cpi	. ./ a.txt	b.txt
instruction	chemin fichier 1	chemin fichier 2	

IV.2.3 Quelques exemples

— **Exemple 1**.

1. On affiche "Hello world!" : **echo "Hello world!"**
2. On se déplace dans le dossier *lycee* : **cd lycee**
3. On créé le dossier *NSI* : **mkdir NSI**

```
PS C:\Users> echo "Hello world !"
Hello world !
PS C:\Users> cd lycee
PS C:\Users\lycee> mkdir NSI


    Directory: C:\Users\lycee


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        04/01/2020     18:24                NSI
```

— **Exemple 2**.

1. On se déplace dans le dossier *NSI* : **cd NSI**
2. On écrit "Bonjour" dans le fichier *test.txt* : **echo "Bonjour" > test.txt**

```
PS C:\Users\lycee> cd NSI
PS C:\Users\lycee\NSI> echo "Bonjour" > test.txt
```

— **Exemple 3**.

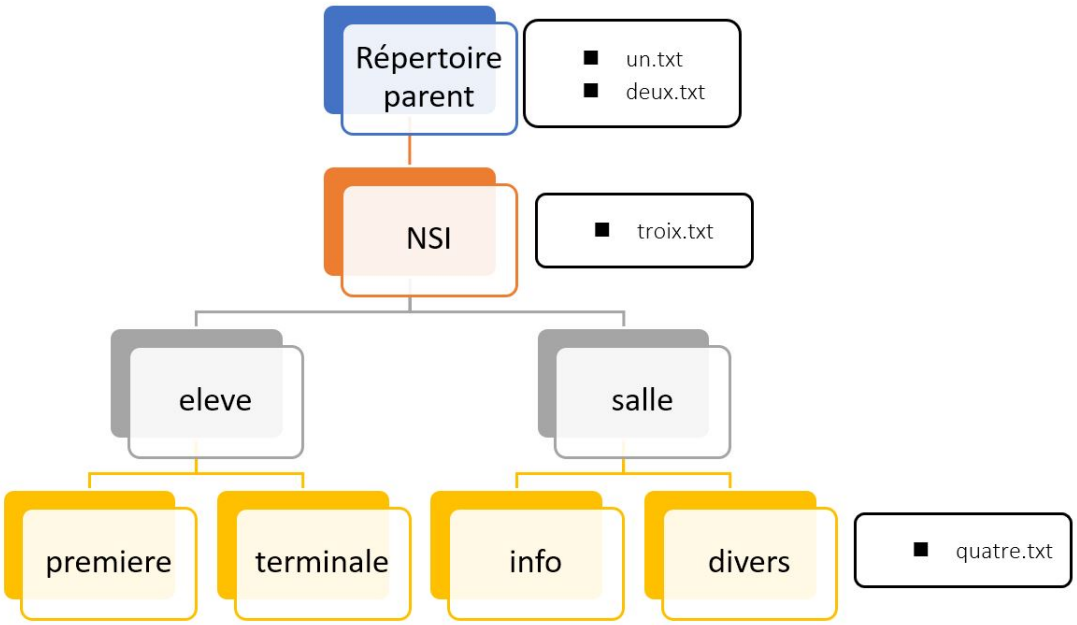
1. On affiche le contenu du dossier courant *NSI* : **ls**
2. On affiche le contenu du fichier *test.txt* : **cat test.txt**

```
PS C:\Users\lycee\NSI> ls

    Directory: C:\Users\franc\NSI


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        04/01/2020     18:26             20 test.txt

PS C:\Users\lycee\NSI> cat test.txt
Bonjour
```

Exercice 4

Remarque : vous pouvez réaliser cet exercice sur repl.it en créant un fichier bash (ou lieu de python).

1. Créer dans le répertoire d'accueil (répertoire parent dans le schéma), l'arborescence ci-dessous constituée des 7 répertoires suivants : NSI, eleve, salle, premiere, terminale, info, divers.



2. Créer deux fichiers vides appelés « *un.txt* » et « *deux.txt* » dans votre répertoire d'accueil (le parent de NSI).
   Utilisez la commande : **echo "" > fichier.txt**
3. Écrire le texte "La NSI, c'est ma passion!" dans le fichier « *un.txt* ».
4. Vous déplacer dans votre répertoire **eleve**.
   Afficher le contenue du fichier « *un.txt* », à partir de ce répertoire en utilisant un chemin absolu du fichier.
5. Déplacer vous dans le répertoire **NSI**.
   En utilisant un chemin relatif, copier le fichier « *un.txt* » dans le répertoire **NSI** en lui donnant le nom « *trois.txt* » .
6. Déplacer vous dans le répertoire **salle**.
   En utilisant un chemin absolu, copier le fichier « *trois.txt* » dans le répertoire **divers** en lui donnant le nom « *quatre.txt* » .

IV.3 Ligne de commande et motif glob

1. **Motifs génériques**.
   Une chaîne est un motif générique si elle contient un ou plusieurs caractères parmi «? », « * » et « [». Le développement (**globbing**) est l'opération qui transforme un motif générique en une liste de noms de fichiers correspondant à ce motif. La correspondance est définie ainsi :
   — Un «? » (sans les guillemets) correspond à n'importe quel caractère.
   — Un « * » (sans les guillemets) correspond à n'importe quelle chaîne, y compris la chaîne vide.
2. **Exemple 1**
   Si on souhaite par exemple afficher tous les noms de fichiers dont l'extension est .jpg on peut utiliser :

```
ls *.jpg
```

3. **Exemple 2**
   Si on souhaite par exemple afficher tous les noms de fichiers qui commencent par img et se terminent par .txt on peut utiliser :

```
ls img*.txt
```

4. **Exemple 3**
   Si on souhaite par exemple afficher tous les noms de fichiers qui sont de la forme nom?.txt où? est un seul caractère, on peut utiliser :

```
ls nom?.txt
```

Exercice 5

```
% ls
entier.py
flottant.py
readme.md
% mkdir foo
% mv *.py foo
```

Question 6

Que peut-on dire du système de fichiers suite à l'exécution ci-dessus?

- **a.** Les fichiers **entier.py, flottant.py** et **foo** ont été déplacés dans le répertoire de l'utilisateur
- **b.** L'utilisateur **foo** est propriétaire des fichiers **entier.py** et **flottant.py**
- **c.** Le répertoire **foo** contient le résultat de l'exécution des deux fichiers **entier.py** et **flottant.py**
- **d.** Le répertoire **foo** contient deux fichiers d'extension **.py**

IV.4 Entrées/Sorties en shell Bash

Objectifs

— Manipuler les entrées/sorties et les redirections

1. **Les entrées : read et Read-Host**
   Pour effectuer une saisie utilisateur et afficher le contenu de la variable.

```
# Sous Unix
read Age
echo $Age

# Sous PowerShell
$Age = Read-Host "Quel est votre âge s'il vous plaît"
echo $Age
```

2. **Les entrées sorties dans un fichier : > et > >**
   — **echo "Bonjour" > salut.txt** :
   Pour envoyer le mot "*Bonjour*" dans le fichier *salut.txt* qui est créé (ou réinitialiser, avec contenu effacé) .
   — **echo "Comment ça va ?" > > salut.txt** :
   Pour ajouter une nouvelle ligne "*Comment ça va ?*" dans le fichier existant *salut.txt.*

```
echo "Bonjour" > salut.txt
echo "Comment ça va ?" > > salut.txt
```

IV.5 Les filtres, tubes (pipes) et redirections

Unix permet l'utilisation de filtres (comme wc, sort, cut, tr) qui sont souvent combinés avec des tubes (pipes) notés "|" pour envoyer le résultat dans un fichier.

1. **Compter des lignes ou des caractères : wc ou gc (Get-Content)**

```
# Sous Unix
wc test.txt | wc -l # pour compter les lignes
wc test.txt | wc -c # pour compter les caractères

# Sous PowerShell
# pour compter les lignes (gc ou Get-Content)
Get-Content test4.txt | Measure-Object -Line

# pour compter les caractères (gc ou Get-Content)
gc test4.txt | Measure-Object -Character
```

2. **Pour trier : sort**
   Pour trier le fichier test4.txt et envoyer le fichier trié dans test5.txt.

```
cat test4.txt | sort > test5.txt
cat test5.txt
```

IV.6 Droits et permissions sous Unix

IV.6.1 Droits et groupes

1. **Les groupes**
   Unix sépare le monde en 3 groupes et chaque fichier (ou répertoire) va préciser les droits attribués à chacun de ces groupes.
   - l'utilisateur ou propriétaire (user) ;
   - le groupe (group) ;
   - le reste (others ou public) .
2. **Les droits**
   Les droits sur un fichier UNIX s'attribuent sur trois « actions » différentes possibles :
   - la lecture (r) : on peut par exemple lire le fichier avec un logiciel.
     Lorsque ce droit est alloué à un répertoire, il autorise l'affichage du contenu du répertoire (la liste des fichiers présents à la racine de ce répertoire).
   - l'écriture (w) : on peut modifier le fichier et le vider de son contenu.
     Lorsque ce droit est alloué à un répertoire, il autorise la création, la suppression et le changement de nom des fichiers qu'il contient, quels que soient les droits d'accès des fichiers de ce répertoire (même s'ils ne possèdent pas eux-mêmes le droit en écriture).
   - l'exécution (x) : on peut exécuter le fichier s'il est prévu pour, c'est-à-dire si c'est un fichier exécutable.
     Lorsque ce droit est attribué à un répertoire, il autorise l'accès (ou ouverture) au répertoire.

   On appelle parfois **r**, **w** et **x** des « **flags** » ou « **drapeaux** ». Sur un fichier donné, ces 3 flags doivent être définis pour son propriétaire, son groupe, mais aussi les autres utilisateurs (différents du propriétaire et n'appartenant pas au groupe).
3. **Représentation des droits**.
   - Cet ensemble de 3 droits sur 3 entités se représente généralement de la façon suivante : on écrit côte à côte les droits r, w puis x respectivement pour le propriétaire (u), le groupe (g) et les autres utilisateurs (o).
   - Les codes u, g et o (u comme user, g comme group et o comme others) sont utilisés par les commandes UNIX qui permettent d'attribuer les droits et l'appartenance des fichiers. Lorsqu'un flag est attribué à une entité, on écrit ce flag (r, w ou x), et lorsqu'il n'est pas attribué, on écrit un « - ».
4. **En ligne de commande**
   - Les droits des fichiers d'un répertoire peuvent être affichés par la commande : **ls -l**

     ```
     ls -l test.txt
     ```

   - Les droits d'accès apparaissent alors comme une liste de 10 symboles. :

     ```
     drwxr-xr-x
     ```

   - Le premier symbole peut être **« - »**, **« d »**, soit **« l »**, entres autres . Il indique la nature du fichier :
     - fichier classique : **-**
     - répertoire : **d**
     - lien symbolique : **l**
   - Dans cet exemple :

     ```
     drwxr-xr-x
     ```

     drwxr-xr-x => d | rwx | r-x | r-x

     - d : c'est un répertoire.
     - rwx pour le 1er groupe de 3 symboles : son propriétaire peut lire, écrire et exécuter.
     - r-x pour le 2nd groupe de 3 symboles : le groupe peut uniquement lire et exécuter le fichier, sans pouvoir le modifier.
     - r-x pour le 3ème groupe de 3 symboles : le reste du monde peut uniquement lire et exécuter le fichier, sans pouvoir le modifier.

— En pratique, en exécutant la commande suivante : **ls -l**
on obtient la liste du contenu du répertoire courant, par exemple :

```
drwxr-xr-x   6 roza lycee     4096 2019-10-29 23:09 Bureau
drwxr-x---   2 roza lycee     4096 2019-10-22 22:46 Documents
lrwxrwxrwx   1 roza lycee       26 2019-09-22 22:30 Examples -> /usr/share/ex
-rw-r--r--   1 roza lycee  1544881 2019-10-18 15:37 forum.xcf
drwxr-xr-x   7 roza lycee     4096 2019-09-23 18:16 Images
```

On retrouve dans la première colonne le groupe de 10 caractères permettant de connaître les droits pour chaque fichier.
Ainsi, pour le fichier forum.xcf, on a : **-rw-r–r–** soit [-] [rw-] [r - -] [r - -]

— Le 1er caractère est - : c'est un fichier.

— Le premier groupe de 3 caractères est [rw-] :
Le propriétaire roza a le droit de lecture et écriture (mais pas d'exécution) sur le fichier.

— Les 2 groupes suivants sont [r - -] :
Les utilisateurs du groupe lycee et les autres n'ont que le droit de lecture (pas d'écriture, ni d'exécution) .

— Bilan : Ce fichier appartient à l'utilisateur roza qui a les droits d'écritures et lecture, et est accessible en lecture au groupe lycee et aux autres

IV.6.2 Changer les droits

Seul le propriétaire d'un fichier peut changer les permissions d'accès.

— L'outil **chmod** (change mode, changer les permissions) permet de modifier les permissions sur un fichier. Il peut s'employer de deux façons :
- soit en précisant les permissions de manière octale, à l'aide de chiffres) ;
- soit en ajoutant ou en retirant des permissions à une ou plusieurs catégories d'utilisateurs à l'aide des symboles r w et x, que nous avons présentés plus haut.

— Avec la deuxième méthode, on va choisir :

. À qui s'applique le changement :

- u (user, utilisateur) représente la catégorie "propriétaire" ;
- g (group, groupe) représente la catégorie "groupe propriétaire" ;
- o (others, autres) représente la catégorie "reste du monde" ;
- a (all, tous) représente l'ensemble des trois catégories.

. La modification que l'on veut faire

- + : ajouter
- - : supprimer
- = : affectation

. Le droit que l'on veut modifier

- r : read : lecture
- w : write : écriture
- x : execute : exécution, autorisation d'exécution. La permission d'exécution régit également l'accès à un répertoire : si l'exécution n'est pas autorisée sur un répertoire, on ne peut faire un chdir (commande cd) sur ce répertoire.

— Par exemples, on enlèvera le droit d'écriture pour les autres avec :

```
chmod o-w fichier3
```

— Par exemples, ajoutera le droit d'exécution à tout le monde avec :

```
chmod a+x fichier3
```

— On peut aussi combiner plusieurs actions en même temps :
  . On ajoute la permission de lecture, d'écriture et d'exécution sur le fichier fichier3 pour le propriétaire ;
  . On ajoute la permission de lecture et d'exécution au groupe propriétaire, on retire la permission d'écriture ;
  . On ajoute la permission de lecture aux autres, on retire la permission d'écriture et d'exécution.

```
chmod u+rwx,g+rx-w,o+r-wx fichier3
```

— **En Octal**.
En octal, chaque « groupement » de droits (pour user, group et other) sera représenté par un chiffre et à chaque droit correspond une valeur :
  . r (read) = 4
  . w (write) = 2
  . x (execute) = 1
  . - = 0

Par exemple,
  . Pour rwx, on aura : 4+2+1 = 7
  . Pour rw-, on aura : 4+2+0 = 6
  . Pour r–, on aura : 4+0+0 = 4

Ce qui permet de faire toutes les combinaisons :
  . 0 : - - - (aucun droit)
  . 1 : - - x (exécution)
  . 2 : - w - (écriture)
  . 3 : - w x (écriture et exécution)
  . 4 : r - - (lecture seule)
  . 5 : r - x (lecture et exécution)
  . 6 : r w - (lecture et écriture)
  . 7 : r w x (lecture, écriture et exécution)

— Exemple : Reprenons le répertoire Documents. Ses permissions sont :

```
drwxr-x---
```

En octal, on aura 750 :

rwx	r-x	- - -
7(4+2+1)	5(4+0+1)	0(0+0+0)

Pour mettre ces permissions sur le répertoire on taperait donc la commande :

```
chmod 750 Documents
```

Exercice 6

En exécutant la commande suivante : **ls -l**
on obtient la liste du contenu du répertoire courant ci-dessous :

1. Donner le nom de l'utilisateur auquel appartient le fichier1, les droits qu'il a sur ce fichier, ceux du groupe et des autres.
2. Faire de même pour les fichiers 2 et 3.
3. Donner l'équivalent en octal du droit correspondant.

```
-rwx------   1 roza lycee     4096 2019-10-29 23:09 fichier1
-rwx--x--x   1 pierre admin     4096 2019-10-22 22:46 fichier2
-r-xr-----   1 alice etu       26 2019-09-22 22:30 fichier3
```

↩ **Fin du cours** ↪

IV.7 script Bash

IV.7.1 Les quotes

- Simple quote ou apostrophe (touche du 4)
  Les simples quotes délimitent une chaîne de caractères. Même si cette chaîne contient des commandes ou des variables shell, celles-ci ne seront pas interprétées. Par exemple :

```
$ variable="secret"
$ echo 'Mon mot de passe est $variable.'
Mon mot de passe est $variable.
```

- Doubles quotes ou guillemets (touche du 3)
  Les doubles quotes délimitent une chaîne de caractères, mais les noms de variable sont interprétés par le shell. Par exemple :

```
$ variable="secret"
$ echo "Mon mot de passe est $variable."
Mon mot de passe est secret.
```

  Ceci est utile pour générer des messages dynamiques au sein d'un script.

- Back-quote, apostrophe inversée ou accent grave (Alt Gr + 7)
  Bash considère que les Back-quotes délimitent une commande à exécuter. Les noms de variable et les commandes sont donc interprétés. Par exemple :

```
$ echo  `variable="connu"; echo "Mon mot de passe est $variable."`
Mon mot de passe est connu.
```

  Autre exemple :

```
echo `ls`
```

Exercice 7

Exécuter les commandes suivantes, changer les quotes pour voir les différents résultats :

```
message='Bonjour tout le monde'
echo 'Le message est : $message'
message='Bonjour tout le monde'
echo "Le message est : $message"
message=`pwd`
echo "Vous êtes dans le dossier $message".
```

IV.7.2 Enregistrer une variable

La commande read joue le role de 'input'. Le flag -p sert à associer un mesasge.

```
read  prenom nom
echo $prenom $nom
read -p 'entrez votre nom :' nom
echo $nom
```

IV.7.3 Ecrire un script

Pour enregistrer un script bash, il faut ouvrir un editeur et mettre le format .sh comme par exemple :

```
toto.sh
```

Pour lancer le script en ligne de commande vous devez taper :

```
./toto.sh
```

Un script de base s'écrit comme suit remarquer l'en-tête nécessaire #!/bin/bash

```
#!/bin/bash
let "a = 5"
let "b = 2"
let "c = a + b"
echo $c
```

Exercice 8

Rédiger le script précédent dans toto.sh, lancer le, changer les quotes pour voir les différents résultats

Remarque

Attention les espaces sont fondamentales dans un script bash, si votre script ne marche pas, vérifier d'abord les espaces.
Parfois le fichier ne se lance pas parce que vous n'avez pas les droits, n'hésitez pas à vous attribuer les droits d'exécution avec 'chmod'.

IV.7.4 Les paramétres

On peut lancer un script en mode commande avec des paramétres comme par exemple :

```
./toto.sh 25 bonjour 47
```

```
./variables.sh param1 param2 param3
$# : contient le nombre de paramètres ;
$0 : contient le nom du script exécuté (ici ./variables.sh) ;
$1 : contient le premier paramètre ;
$2 : contient le second paramètre ;
...
```

Exercice 9

Rédiger le script suivant appelé addition.sh et lancer le

```
#!/bin/bash
let "c = $1 + $2"
echo "$1 + $2"$c
echo "le script s'appelle $0 et contient $# paramétres"
```

Exercice 10

Ecrire un script nommé bonjour.sh qui prend vos prénom et nom en paramétre et écrit "Bonjour [Prénom] [Nom]".

IV.7.5 Les tableaux

```
#!/bin/bash
tableau=('valeur0' 'valeur1' 'valeur2')
tableau[5]='valeur5'
echo ${tableau[*]}
echo " le troisieme element est" ${tableau[2]}
```

Exercice 11

Reprendre le script bonjour.sh précédent, copier les paramètres dans un tableau, puis afficher le tableau (Bonjour « prénom » « nom »!)

IV.7.6 Les tests if

La structure ressemble beaucoup au python sauf le 'fi' à la fin.

```
#!/bin/bash
if [ test ]
then
    echo "Le premier test a été vérifié"
elif [ autre_test ]
then
    echo "Le second test a été vérifié"
elif [ encore_autre_test ]
then
    echo "Le troisième test a été vérifié"
else
    echo "Aucun des tests précédents n'a été vérifié"
fi
```

Exercice 12

Ecrire le script suivant sans copier-coller. Si vous ne faites aucune erreur en l'écrivant vous êtes trés fort. Remarquer que l'on peut éviter les passages à la ligne avec des points-virgules

```
#!/bin/bash

read -p 'qui es-tu ?' nom
if [ $nom = "Bruno" ]; then
    echo "Salut Bruno !"
else
    echo "Je ne vous connais pas!"
fi
```

IV.7.7 Opérateurs logiques

- && pour le ET logique
- || pour le OU logique
- ! (point d'exclamation) pour la négation.

Exercice 13

Ecrire le script suivant sans copier-coller.

```
#!/bin/bash

read -p "Si vous etes d'accord entrez o ou oui : " reponse
if [ ! $reponse = "o" ] && [ ! $reponse = "oui" ]; then
    echo "Non, je ne suis pas d'accord !"
else
    echo "Oui, je suis d'accord"
fi
```

IV.7.8 Les boucles

Voici quelques exemples

```
#!/bin/bash
while [ -z $reponse ] || [ $reponse != 'oui' ]
do
    read -p 'Dites oui : ' reponse

done
```

```
#!/bin/bash
for variable in 'valeur1' 'valeur2' 'valeur3'
do
    echo "La variable vaut $variable"
done
```

```
#!/bin/bash
for animal in 'chien' 'souris' 'moineau'
do
    echo "Animal en cours d'analyse : $animal"
done
```

Exercice 14

Que va faire le script suivant ?

```
#!/bin/bash

for fichier in `ls`
do
    mv $fichier $fichier-old
done
```

IV.7.9 Pour continuer à s'entrainer

Faire les exercices proposés au lien suivant : https://ineumann.developpez.com/tutoriels/linux/exercices-shell/
Ne regardez pas tout de suite les solutions.