Le fonctionnement d'un ordinateur

Table des matières

Objectifs 3

I - Contexte 4

II - La machine de Turing 5

III - Exercice : Appliquer la notion 8

IV - Automates et diagramme états-transitions 9

V - Exercice : Appliquer la notion 12

VI - Architecture Von Neumann 13

VII - Exercice : Appliquer la notion 17

VIII - Mémoire vive et mémoire secondaire 18

IX - Exercice : Appliquer la notion 21

X - Langage machine et assembleur 22

XI - Exercice : Appliquer la notion 24

XII - Complexité algorithmique 25

XIII - Exercice : Appliquer la notion 29

XIV - Essentiel 30

XV - Quiz 31

Solutions des exercices 35

Crédits des ressources 43

Objectifs

- Découvrir les concepts théoriques à l'origine de l'informatique moderne ;
- Comprendre le fonctionnement d'un ordinateur.

I Contexte

Durée : 2h

Environnement de travail : Repl.it, terminal

Pré-requis : Aucun

L'informatique connaît depuis plusieurs décennies un essor fulgurant. Des objets connectés aux smartphones, les ordinateurs ont envahi chaque recoin de notre vie quotidienne.

L'enjeu de ce module sera de comprendre le fonctionnement de l'ordinateur moderne et de découvrir très simplement certaines bases théoriques sous-jacentes.

Nous verrons en particulier que tous les ordinateurs, de la simple montre connectée au serveur web, ont une architecture commune : l'architecture de von Neumann.

Un ordinateur dispose donc des éléments suivants :

- des **unités de calcul**, les micro-processeurs ou CPU ;
- des **mémoires vives** qui permettent de stocker temporairement l'information afin de la traiter ;
- des **périphériques d'entrées/sorties** qui permettent d'interagir avec la machine (comme le clavier, la souris ou l'écran) ou encore de sauvegarder les données une fois la machine éteinte (comme le disque dur).

Nous verrons également que tous les ordinateurs ont en commun le modèle de la **machine de Turing**. Il s'agit d'un modèle de calcul automatisé qui a été créé par Alan Turing en 1936, avant même l'apparition des premiers composants électroniques.

II La machine de Turing

Objectif

- Découvrir ce qu'est une machine de Turing.

Mise en situation

L'ordinateur possède une base théorique qui sous-tend la majorité des développements scientifiques et technologiques en informatique : la machine de Turing.

Ce modèle a été imaginé par **Alan Turing en 1936** afin de donner une définition rigoureuse de la manipulation automatique de l'information.

L'idée générale de ce modèle est qu'un acteur modifie ce qui est inscrit sur un ruban découpé en plusieurs cases, en appliquant des règles mécaniques, sans réfléchir. Par exemple une règle pourrait être : « si il y a un A dans la case que tu lis, alors écrit à la place Z et va à la case suivante ». Une telle machine pourrait servir à remplacer les A par des Z dans un texte.

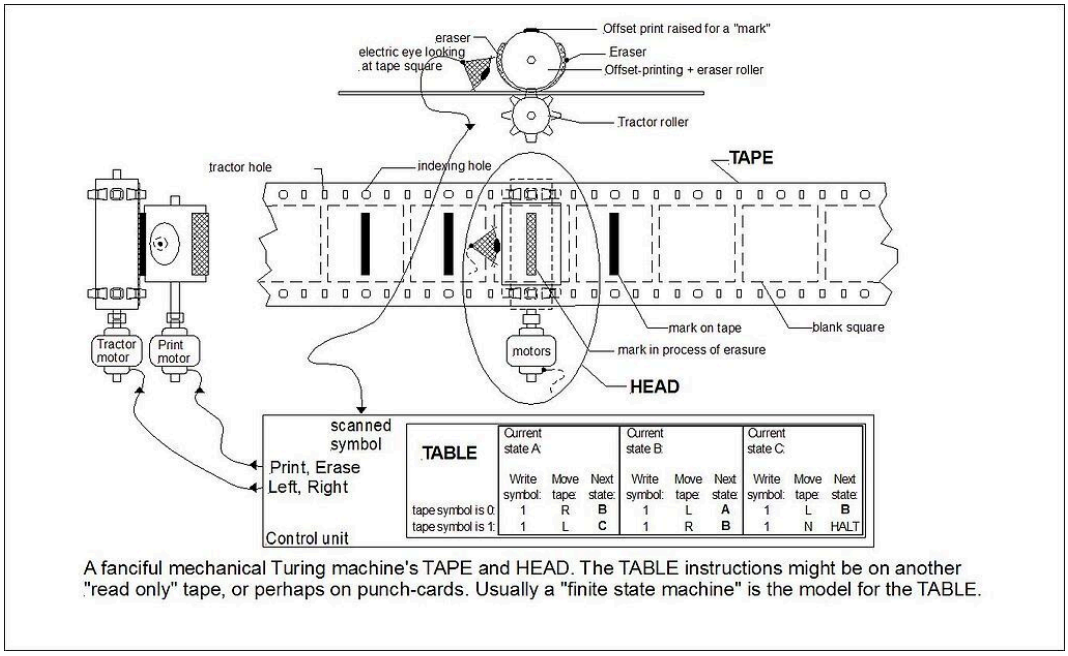
La machine de Turing Fondamental

La machine de Turing est un modèle abstrait du fonctionnement d'une machine (mécanique) à calculer (type ordinateur). Ce modèle pensé par Alan Turing permet d'apporter une définition précise au concept d'**algorithme** (appelé alors « procédure mécanique »).

Description d'une machine de Turing

Une machine de Turing est un objet abstrait composé de quatre éléments :

- Un ruban infini divisé en cases contenant chacune un symbole issu d'une liste finie (un alphabet).
- Une tête de lecture/écriture pouvant lire et écrire sur le ruban, et pouvant se déplacer vers la gauche ou la droite du ruban.
- Un registre d'état qui retient l'état actuel de la machine. Pour un être humain faisant du calcul mental, on pourrait comparer cela au fait de retenir l'état d'avancement du calcul (par exemple les retenues).
- Une table d'actions qui lie (en fonction de l'état courant) un symbole lu à une action à effectuer par la tête de lecture/écriture (déplacement et/ou écriture).



A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

Machine de turing

États

Une machine de Turing possède un registre permettant de retenir son état. Il existe trois types d'états :

- L'état initial : état lors du lancement de l'algorithme
- Les états intermédiaires : états transitoires permettant de faire avancer le calcul.
- Les états acceptants : états correspondant à une fin possible de l'algorithme

Démonstration

Pour mieux comprendre le concept de machine de Turing, voici une application Web qui simule très bien son fonctionnement.

Choisir le programme "Ajouter 1", entrer la valeur "1" comme valeur de départ et cliquer sur "Commencer".

Le ruban contiendra "10" en fin de programme (soit le nombre 2 en binaire). Le tableau en haut à droite fait office de table d'actions en ayant un curseur pointant sur l'état courant. Ce curseur est le registre d'état. L'alphabet est : 1, 0, VIDE.

⚠ Attention

Une telle machine peut sembler un peu simpliste mais les algorithmes les plus complexes ne sont qu'une succession d'un très grand nombre d'opérations de base.

Machine de Turing universelle

Fondamental

Alan Turing nomme **Machine de Turing universelle** une machine de Turing capable de « simuler » n'importe quel autre machine de Turing. Dans la mesure où il est possible de construire une machine de Turing pour n'importe quel algorithme, machine de Turing **universelle** peut calculer tout ce qui est considéré comme calculable.

Pour ce faire, une machine de Turing universelle doit recevoir le « codage » de la machine de Turing simulée ainsi que ses données.

Un ordinateur peut être vu comme une machine de Turing universelle.

Aller plus loin que le calcul arithmétique

Complément

Ainsi, ce modèle abstrait proposé par Turing changea totalement la notion même de calcul. Avant lui, le mot calcul ne signifiait que calcul arithmétique (additionner, soustraire, etc.). Après Turing, la notion de calcul est redéfinie et englobe des opérations bien plus abstraites, englobant tous les algorithmes. Elle ouvre la voie à des domaines nombreux comme l'intelligence artificielle qui fut le sujet d'un de ses articles.

À retenir

- Une machine de Turing est un modèle abstrait à l'origine de la notion d'algorithme.
- Tout algorithme peut être représenté par une machine de Turing.
- L'ordinateur s'apparente à une machine de Turing universelle pouvant exécuter tous les algorithmes.

III Exercice : Appliquer la notion

Question

[solution n°1 p. 35]

Voici une application simulant des machines de Turing. Comment calculer 5 × 4 avec cette application ?

Indice :

Les programmes de l'application ne permettent que la multiplication par deux : il faut décomposer la multiplication.

Indice :

5 × 4 peut aussi s'écrire (5 × 2) × 2, et 5 s'écrit 101 en binaire. Il faudra donc utiliser deux fois le programme.

IV Automates et diagramme états-transitions

Objectifs

- Découvrir ce qu'est un automate ;
- Savoir lire un diagramme de transition.

Mise en situation

Le concept de machine de Turing s'inscrit au sein de la **théorie des automates**. Un automate est un dispositif qui est capable de produire une série d'actions sans intervention humaine. Il est programmé pour produire ces actions. Une horloge est un exemple d'automate simple.

Un ordinateur est un automate qui est capable de recevoir plusieurs programmes différents. Il peut effectuer des actions très diverses, pas seulement donner l'heure, grâce à des périphériques puissants, comme l'écran.

Théorie des automates

Fondamental

Un automate est un modèle mathématique permettant de réaliser des calculs, composé de plusieurs états. Le passage d'un état à un autre est conditionné par les données transmises à l'automate.

Diagramme états-transitions

Fondamental

Un diagramme états-transitions permet de représenter les états d'un automate et les passages possibles d'un état à un autre.

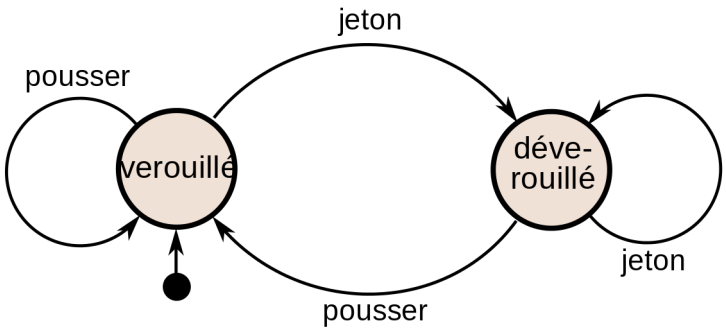
- Un état est représenté par un nœud (un cercle).
- Un changement d'état est modélisé par une flèche accompagnée de la condition de ce changement.

Il est possible qu'une flèche pointe sur l'état dont elle vient lorsqu'une action ne modifie pas l'état.

Automate très simple

Exemple

Un portillon possède deux états : verrouillé et déverrouillé. On comprend que pousser une porte verrouillée ne la débloque pas mais qu'insérer un jeton si.



Automate de portillon

Machine de Turing et automate

Remarque

Une machine de Turing est un type avancé d'automate : le passage d'un état à un autre est conditionné par le symbole lu sur le ruban.

Comment lire le diagramme états-transitions d'une machine de Turing ?

Méthode

Une machine de Turing étant un automate très particulier, son diagramme possède une particularité : la condition de changement d'état.

La condition de changement d'état est représentée par un triplet :

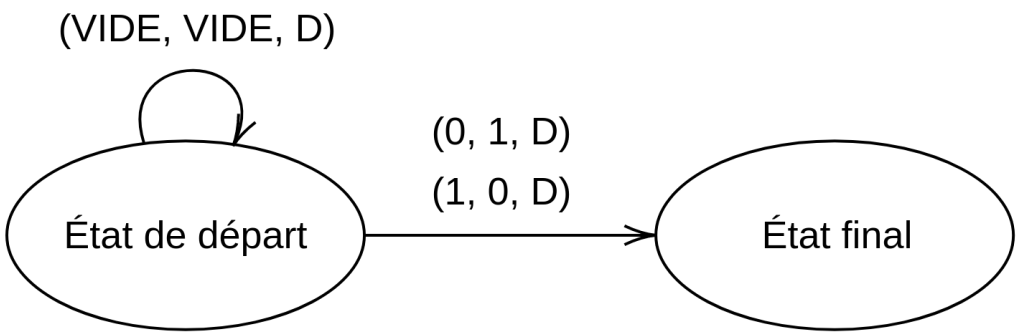
`([symbole lu] ; [symbole écrit] ; [déplacement])`

Exemple : la condition `(1; 0; D)` signifie que si la tête lit le symbole 1, il faut écrire le symbole 0 puis déplacer la tête de lecture/écriture vers la droite.

Machine très simple

Exemple

Cette machine très simple lit la bande de gauche à droite et inverse le premier bit. La machine passe enfin de l'état de départ à l'état final.



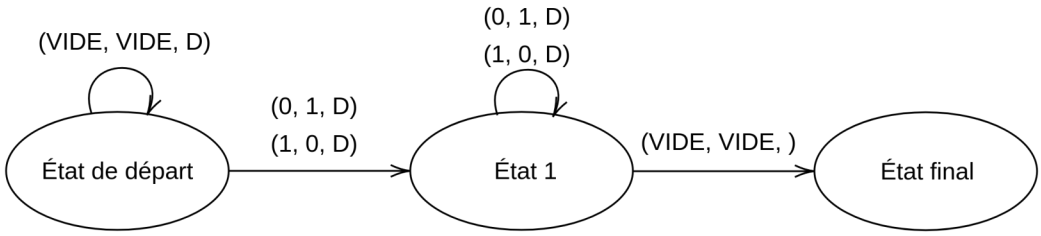
À retenir

- Une machine de Turing est un type d'automate.
- Il est possible de représenter une machine de Turing grâce à un diagramme états-transitions.

Exercice : Appliquer la notion

[solution n°2 p. 35]

Que renvoie la machine de Turing correspondant au diagramme états-transitions ci-dessous lorsqu'on lui donne entrée la valeur 0110 ?



VI Architecture Von Neumann

Objectif

- Connaître le modèle conceptuel à l'origine du fonctionnement des ordinateurs modernes.

Mise en situation

Tous les ordinateurs partagent un modèle de conception similaire, hérité de l'**architecture de Von Neumann**. Cette architecture repose en premier lieu sur une unité de calcul et de contrôle qui est capable de manipuler l'information. On peut voir cela comme le bureau où on réalise tous les opérations. L'unité de calcul ne traite qu'une petite quantité d'information à la fois, et seulement des nombres binaires, mais il fait cela très rapidement.

Ensuite, l'ordinateur est doté d'une mémoire vive qui permet de stocker une plus grosse quantité d'informations. C'est là que l'ordinateur dépose les résultats des calculs une fois effectués et où il prend les nouvelles données à traiter. On peut voir cela comme une armoire avec des casiers où l'on range les données.

Enfin, il y a les périphériques qui permettent d'interagir avec le monde.

Architecture de Von Neumann

Cette architecture est un modèle conceptuel décrivant le fonctionnement d'un ordinateur. Elle est utilisée par la quasi-totalité des ordinateurs.

Ce modèle se compose de quatre parties :

- L'unité arithmétique et logique (UAL ou ALU en anglais),
- L'unité de contrôle,
- La mémoire,
- Les entrées/sorties.

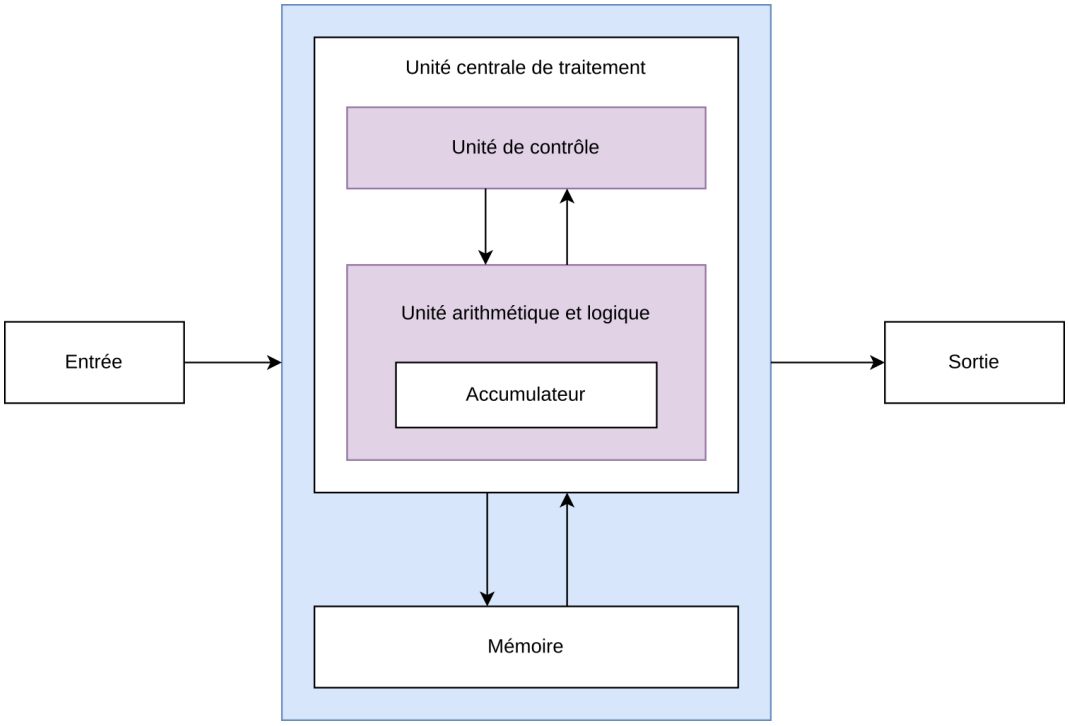


Schéma d'une architecture Von Neumann

L'unité arithmétique et logique

Ce composant est chargé de réaliser toutes les opérations de base : les opérations arithmétiques sur les nombres (addition, multiplication, etc.) et des opérations binaires (OR, AND, etc.).

Masques logiques

⊕ Complément

Les masques logiques diffèrent de l'arithmétique sur les nombres. Ces masques travaillent sur la représentation binaire des données, et effectuent des opérations bit à bit. Une explication plus complète est disponible ici : https://fr.wikibooks.org/wiki/Les_op%C3%A9rations_bit_%C3%A0_bit/Les_masques.

L'unité de contrôle

Ce composant est chargé d'ordonnancer les instructions et d'envoyer tous les calculs à effectuer à l'unité arithmétique et logique.

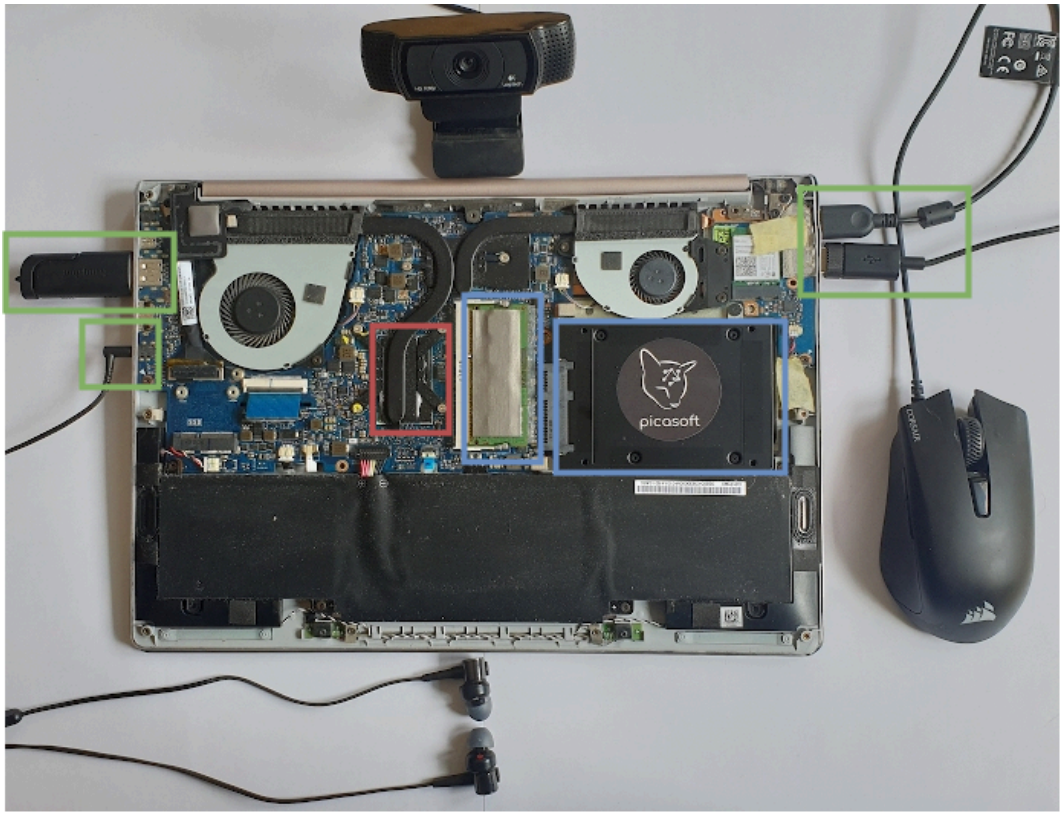
La mémoire

Cette mémoire stocke à la fois les instructions et les données. D'un côté, elle sera utilisée par l'unité de contrôle pour stocker les séquences d'instructions. De l'autre, elle sera utilisé par l'ALU pour stocker les données d'entrée d'un calcul et le résultat.

Entrées/Sorties

Il s'agit de toutes les interfaces permettant d'interagir avec l'ordinateur. Classiquement un clavier peut être vu comme une entrée et un écran comme une sortie.

Exemple



Ordinateur portable ouvert

Sur cette image d'un ordinateur portable moderne, on observe bien l'architecture Von Neumann :

- En rouge, l'unité de contrôle et l'ALU, dont l'ensemble forme un processeur,
- En bleu, la mémoire (ici, RAM et SSD),
- En vert, les périphériques (ici, USB et JACK).

L'écran et la clavier intégrés à l'ordinateur portable sont aussi des périphériques.

Processeurs et Architecture de Von Neumann

Les processeurs modernes (aussi appelé CPU, pour *Central Processing Unit*) regroupent l'unité de contrôle et l'ALU. Beaucoup de processeurs ont de multiples **cœurs**, c'est à dire plusieurs ALU.

En quoi est-il différent d'autres modèles ?

Complément

Une autre architecture très connue (mais moins répandue) existe : l'architecture Harvard. Elle se différencie de l'architecture de Von Neumann notamment par une séparation de la mémoire des instructions exécutée par la machine et de la mémoire de données.

À retenir

- Les ordinateurs modernes utilisent l'architecture dite de Von Neumann.
- L'architecture Von Neumann se compose en quatre parties : ALU, unité de contrôle, mémoire et entrées/sorties.
- Avec les avancées technologiques récentes (processeurs multi-cœurs, etc.), ce modèle perdure tout en évoluant.

Exercice : Appliquer la notion

[solution n°3 p. 35]

Après avoir découvert l'architecture de Von Neumann, il est intéressant d'appliquer ce modèle en essayant de disséquer un *smartphone*. Le but de cet exercice sera de lier chaque partie de l'architecture à des composants électroniques d'un *smartphone*.

En cas de manque d'inspiration, il est possible de consulter le site mobilecellphonerepairing.com[1] qui liste les composants classiques d'un *smartphone*.

A Processeur | **B** Vibreur | **C** Connecteur USB | **D** GPS | **E** Écran | **F** RAM | **G** Carte SD | **H** Dalle tactile | **I** Batterie | **J** Haut-parleur | **K** Carte SIM

Aucune	ALU et unité de contrôle	Mémoire	Entrée	Sortie

1. http://www.mobilecellphonerepairing.com/mobile-phone-parts.html

VIII Mémoire vive et mémoire secondaire

Objectifs

- Connaître la différence entre mémoire vive et mémoire secondaire ;
- Découvrir la notion de *swap*.

Mise en situation

La mémoire secondaire est la mémoire de stockage de l'ordinateur. Elle conserve les données même si l'ordinateur est éteint. C'est par exemple le cas d'un disque dur.

La mémoire vive est la mémoire de travail de l'ordinateur. Pour exécuter un programme l'ordinateur doit copier les données depuis une mémoire de stockage vers la mémoire vive.

Le développeur doit comprendre ce principe d'architecture. En effet, les copies entre les mémoires de stockage et la mémoire vive sont des actions très lentes, chaque fois qu'on en fait, on ralentit le programme. Pour une application web cela peut vite devenir un goulot d'étranglement. À l'inverse la mémoire vive est souvent assez limitée en volume, il n'est donc pas possible de remonter trop d'information à la fois dans la mémoire de travail.

Mémoire vive Az Définition

La mémoire vive (ou RAM, pour *Random Access Memory*) est caractérisée par sa volatilité : les données s'effacent lors de l'extinction du système.

Elle contient les données des processus qui s'exécutent sur l'ordinateur et est constamment sollicitée par le processeur pour traiter ces données.

Mémoire secondaire Az Définition

La mémoire secondaire est caractérisée par sa non-volatilité : les données sont conservées même lorsque le système est éteint.

Elle est moins sollicitée pas le CPU et contient les fichiers et données des programmes qui ne sont pas utilisés.

Types de mémoire secondaire

Il existe plusieurs types de mémoire secondaire classées selon la possibilité de les modifier, comme par exemple :

- ROM (pour *Read-Only Memory*) : ce type de mémoire n'est pas effaçable et est utilisé, par exemple, pour stocker les instructions de démarrage d'un ordinateur. Les données stockées sont enregistrées par le constructeur et ne pourront pas être altérées.
- EEPROM (pour *Electrically Erasable Programmable Read Only Memory*) : ce type est le plus répandu - c'est celui des clés USB, des cartes SD, etc. La suppression et la modification des données est possible.

Stockage de la mémoire secondaire Exemple

Les **disques durs** sont des exemples de mémoire secondaire.

Deux grandes familles se distinguent : les disques durs utilisant des variantes d'EEPROM - les **SDD** - et les disques durs à disques - appelés **HDD**.

On utilise improprement le nom de disque dur, même si seul ce deuxième type de disque historique correspond à cette appellation.

Dépassement de pile Attention

La mémoire vive étant limitée en taille, il peut arriver que des programmes essaient de stocker plus de données que ne le permet la mémoire vive. Lorsque cela arrive, on parle de **dépassement de pile** (ou *stack overflow*).

En tant que développeur, il faut faire attention à l'utilisation de la mémoire vive pour éviter que les programme ne dysfonctionnent. Avant le dépassement de pile, il peut également arriver que la vitesse d'exécution soit fortement ralentie.

La mémoire swap

Pour éviter le problème de dépassement de pile, certains systèmes d'exploitation placent des parties de la mémoire vive inutilisée sur la mémoire secondaire pour économiser la première.

Ce processus est appelé **échange** et on appelle **swap** la mémoire associée à ce procédé.

Si le processeur a besoin de données qui se trouvent dans le swap, un échange a de nouveau lieu. Cet échange est plus lent qu'un accès direct en RAM, mais en contrepartie de la mémoire vive aura pu être mieux utilisée.

Mémoire virtuelle Définition

La mémoire swap est étroitement lié au concept de mémoire virtuelle[2]. La mémoire virtuelle est le regroupement de la mémoire vive et du swap.

Comparaison des temps d'accès mémoire Remarque

La mémoire vive a un temps d'accès en mémoire bien plus rapide que la mémoire secondaire.

À titre informatif, voici quelques exemples dans le tableau ci dessous.

Type d'accès mémoire	Temps en nano-secondes
Mémoire vive	100
Mémoire secondaire (HDD)	10 000 000

2. https://fr.wikipedia.org/wiki/M%C3%A9moire_virtuelle

ramfs pour accélérer ses programmes

⊕ Complément

`ramfs` est un système de fichiers temporaire, sur Linux, placé sur la RAM.

Cette pratique est intéressante lorsque le développeur souhaite accéder à un petit nombre de fichiers avec rapidité d'accès très haute. Il faut absolument limiter la taille des fichiers pour éviter le dépassement de pile. Une telle pratique peut accélérer significativement certaines opérations.

À retenir

- La mémoire vive est dédiée à des données peu nombreuses qui nécessitent un accès rapide.
- La mémoire secondaire est dédiée à des données potentiellement volumineuses qui ne nécessitent pas un accès immédiat.
- La mémoire vive est limitée en taille et certains mécanismes optimisent son utilisation et évitent des dépassements de pile.

Exercice : Appliquer la notion

[solution n°4 p. 36]

Dans cet exercice on s'intéresse à différents logiciels et au type de mémoire qu'ils utilisent.

Exercice

Le BIOS correspond au micrologiciel contenu dans la carte mère qui s'exécute au démarrage de l'ordinateur.

En utilisant cette article Wikipédia sur le BIOS[3], quel type de mémoire stocke le BIOS ?

- **A** Mémoire secondaire
- **B** Mémoire virtuelle
- **C** Mémoire vive

Exercice

Redis est un système de gestion de bases de données non relationnelles très populaire.

Grâce à cette cet article Wikipédia[4], quels type de mémoire utilise Redis pour ses données?

- **A** Mémoire secondaire
- **B** Mémoire flash
- **C** Mémoire virtuelle
- **D** Mémoire vive

3. https://fr.wikipedia.org/wiki/BIOS_(informatique)
4. https://fr.wikipedia.org/wiki/Redis

X Langage machine et assembleur

Objectifs

- Découvrir le langage machine ;
- Découvrir les bases de l'assembleur.

Mise en situation

Les développeurs écrivent des programmes dans des langages compréhensibles par les humains. Mais l'ordinateur travaille à un niveau différent, il manipule des séquences binaires.

Le langage le plus proche de la machine se nomme assembleur. C'est un langage qui permet des instructions très basiques. On l'utilise assez rarement car il est beaucoup plus efficace pour un être humain d'écrire dans un langage de haut niveau comme le C ou le JavaScript. D'autant qu'il existe des compilateurs ou des interpréteurs qui transforment ce code de haut niveau que l'on écrit en langage machine.

Mais il est néanmoins utile de comprendre les mécaniques de l'assembleur. Cela aide à comprendre certains concepts comme l'adressage mémoire. Et c'est utile pour programmer plus efficacement.

Codage binaire — Fondamental

Un processeur ne comprend pas les langages de programmation. Il ne peut traiter que des instructions sous forme binaire, c'est-à-dire constituées de 0 et de 1. Un processeur ne traitera pas le nombre 42 mais la suite `00101010`, qui est l'équivalent de 42 en codage binaire.

Le langage assembleur — Az Définition

L'assembleur est un langage compréhensible par le processeur. Il ne comporte que quelques instructions très simples (déplacer une donnée, multiplier, additionner, etc.), là où les langages de programmation classiques comportent des instructions plus complexes, correspondant à une combinaisons d'instructions en assembleur.

Les registres

Les registres sont des petites zones mémoire dans le processeur où sont stockées les variables d'entrée ou de sortie d'un calcul. Chaque processeur n'a que quelques registres nommés AX, BX, CX, DX, etc.

Leur taille étant extrêmement limitée, leur tâche est de stocker une donnée **uniquement** le temps du calcul (quelques fractions de secondes) et d'envoyer le résultat en mémoire vive par la suite, quitte à récupérer le résultat dans la mémoire vive pour un calcul ultérieur.

Quelques instructions de base

Exemple

Chaque instruction assembleur a une équivalence binaire direct. Le processeur peut ainsi comprendre ces instructions.

Voici quelques exemples d'instructions de base pour le processeur Intel 8086 :

- `MOV destination source` : déplace une valeur fixe ou celle d'un registre dans un autre,
- `INC registre` : additionne 1 à la valeur d'un registre,
- `NEG registre` : change le signe de la valeur stockée dans un registre,
- `IMUL destination source` : multiplie les valeurs et stocke le résultat dans le premier registre.

Traduire un langage de programmation en langage machine

Attention

L'écriture d'un programme informatique ne se fait la plupart du temps **pas** en assembleur.

On utilise des langages dits de plus haut niveau qui sont eux-mêmes traduits automatiquement en assembleur par un autre programme :

- le compilateur pour les langages compilés (comme C ou C++),
- ou l'interpréteur pour les langages interprétés (comme JavaScript ou Python).

Toutes les machines ne parlent pas exactement le même langage

Complément

Tous les processeurs comprennent un langage binaire. Cependant, chaque type de processeur a son propre jeu d'instructions.

Par exemple, la plupart des ordinateurs ont des processeurs utilisant un jeu d'instructions **étendu** (dit CISC) alors que les smartphones ont des processeurs utilisant un jeu d'instructions **réduit** (RISC). Ainsi, un code dans un langage de programmation donné sera traduit différemment en fonction du processeur cible.

À retenir

- Un processeur ne traite que des instructions binaires.
- L'assembleur est un langage composé d'instructions basiques et facile à traduire en binaire.
- Une instruction dans un langage de programmation donné correspond à de multiples instructions en assembleur.

Exercice : Appliquer la notion

[solution n°5 p. 37]

Ordonnez les instructions en assembleur basé sur l'architecture de l'Intel 8086 afin que le programme :

- incrémente la valeur contenue dans le registre AX,
- double cette valeur incrémentée,
- stocke cette valeur doublée dans BX,
- finisse en remettant à 0 la valeur contenue dans AX.

fr.wikipedia.org/wiki/Jeu d'instructions x86[5]

A MOV ax,0 | **B** MOV bx,ax | **C** IMUL ax,2 | **D** INC ax

Réponse : ___ ___ ___ ___

5. https://fr.wikipedia.org/wiki/Jeu_d%27instructions_x86#Instructions_originales_des_8086/8088

XII Complexité algorithmique

Objectif

- Découvrir la notion de complexité algorithmique.

Mise en situation

Lorsqu'un développeur écrit un programme, il lui est nécessaire d'estimer si ce programme est coûteux. Coûteux cela signifie par exemple qu'il nécessite beaucoup de calculs de la part du processeur. Cela peut aussi signifier qu'il nécessite beaucoup d'espace mémoire.

Savoir calculer ce coût permet de comparer plusieurs solutions alternatives entre elles. Cela permet aussi de savoir si un programme peut « passer à l'échelle » c'est à dire continuer de fonctionner correctement si on le confronte à la réalité.

Par exemple si un développeur a écrit un programme permettant de vérifier une empreinte de carte de paiement, mais que cette vérification prend plusieurs minutes ou nécessite un ordinateur doté d'une mémoire très importante, il ne sera pas possible de l'utiliser pour du paiement en ligne.

Compter le nombre d'opérations pour résoudre un problème Exemple

Pour créer la table de 2 des **n** premiers entiers, il faut réaliser **n** calculs de type multiplication.

De manière générale, on cherche à estimer de manière théorique le nombre d'opérations à effectuer pour résoudre un problème : c'est ici que l'on parle de **complexité** d'un problème.

Complexité algorithmique Az Définition

La **complexité algorithmique** est une estimation du nombre d'opérations élémentaires pour résoudre un problème en fonction de la taille des données d'entrée, notée **n**.

O() Syntaxe

On note O() **l'ordre de grandeur** de la complexité d'un algorithme.

- O(n) signifie que le nombre d'opérations est de l'ordre de n (si on a 1000 données en entrée, on s'attend à environ 1000 opérations).
- O(n^2) signifie que le nombre d'opérations est de l'ordre de n au carré (si on a 1000 données en entrée, on s'attend à environ 1 000 000 d'opérations).

Exemple simple

Exemple

Ce code fait la somme d'une liste d'entiers.

```
function somme(number_list) {
  let res = 0;
  for (var i = 0; i < number_list.length; i++) {
      res += number_list[i];
  }
  return res;
}
```

Étant donné **n** la taille de la liste, il serait nécessaire d'effectuer **n** opérations d'addition pour obtenir la somme finale. Ici, on dira que la complexité de l'algorithme est de l'ordre de **n** et on la notera **O(n)**.

Pourquoi utiliser la complexité algorithmique ?

Remarque

La complexité algorithmique peut se voir comme l'estimation du "temps d'exécution" d'un algorithme en fonction de la taille des données d'entrée.

On suppose pour ce faire que les opérations de base (affectation, addition, etc.) ont le même temps d'exécution : estimer le temps d'exécution revient alors à estimer le nombre d'opérations de base.

L'exécution reste proportionnelle à la taille des données

Attention

La complexité ne change pas en fonction de la taille des données. En revanche, l'exécution sera évidemment plus longue si la taille des données augmente. Ainsi, pour 100 données, un algorithme en **O(n)** effectuera de l'ordre de 100 opérations. Pour 1000 données il en effectuera de l'ordre de 1000.

Estimer rapidement la complexité

Méthode

La complexité permet d'avoir une idée grossière de la durée d'un algorithme. Par exemple, si on détermine que pour une liste de **n** éléments, il faut effectuer **2×n** opérations, on simplifiera en disant que la complexité est **de l'ordre** de **O(n)**.

La raison principale est que les constantes ne changent pas fondamentalement l'ordre de grandeur de la complexité. Par exemple, **$O(n^2)$** et **$O(2 \times n^2)$** sont très proches, et c'est surtout la valeur de **n** qui influencera la durée finale.

Complexités usuelles des algorithmes

Fondamental

On retrouve très régulièrement les complexités suivantes :

- **O(1)** : si le nombre d'opérations ne dépend pas de la taille des données. Par exemple, afficher le premier élément d'une liste.
- **O(n)** : cette complexité se retrouve souvent quand on a besoin de parcourir les éléments d'une liste, par exemple pour calculer la somme des éléments.

- **O(n²)** : par exemple, l'algorithme naïf pour trier une liste de nombre nécessite, pour chaque élément de la liste, de parcourir l'ensemble de la liste.
- **O(n×m)** : par exemple, pour trouver les nombres communs de deux listes de taille **n** et **m**, il faut pour chaque nombre de la première liste parcourir l'ensemble de la deuxième liste pour vérifier s'il existe.

⊕ Complément

La complexité algorithmique permet de dire, qu'à tailles de données égales, un algorithme en **O(n)** terminera bien avant un second algorithme dont la complexité est **O(n²)**.

Ainsi, pour 100 données, le premier algorithme effectuera environ 100 opérations de bases et le second 10 000 opérations. On comprend que, si la taille des données se compte en millions (ou même en milliards), le second algorithme sera beaucoup plus coûteux que le premier.

n	O(n)	O(n²)
10	10	100
100	100	10 000
1 000	1 000	1 000 000
10 000	10 000	100 000 000

Comparaison des complexités

Complexité algorithmique en espace

⊕ Complément

Le temps de calcul est important mais l'espace nécessaire au calcul l'est aussi. La mémoire d'un ordinateur n'est pas infinie : il est utile d'estimer l'espace mémoire nécessaire pour exécuter un algorithme.

La complexité décrite jusqu'ici se nomme complexité en **temps**. Ici, on parle complexité algorithmique en **espace**. La notation sera toujours la même : **O(1)**, **O(n)**, **O(n²)**, etc.

Complexité et machine de Turing

⊕ Complément

Le concept de complexité algorithmique est étroitement lié aux machines de Turing, une des bases théorique des ordinateurs. Elles ont introduit une nouvelle définition de la notion de calcul qui permet également de théoriser sa complexité. Ainsi, aujourd'hui, on peut classer les problèmes de calcul selon la complexité des algorithmes pouvant les résoudre.

On peut distinguer, très schématiquement, trois grands types de problèmes :

- Problème dont l'algorithme finira en un temps polynomial : **O(n)**, **O(n²)**, etc.
- Problème dont l'algorithme finira en un temps exponentiel : **O(2^n)**, **O(n!)**, etc. On évite que ce type d'algorithme car il suffit que **n** augmente un tout petit peu pour que le temps d'exécution se compte en mois, en années voire en siècles.
- Problème dont il n'existe pas de solution (dits indécidables).

À retenir

- Il est possible d'attribuer une complexité à un algorithme.
- Estimer la complexité d'un programme permet d'avoir un ordre de grandeur de son temps d’exécution.
- Comparer la complexité de deux algorithmes permet de déterminer lequel est le plus optimisé pour répondre à un problème donné.

Exercice : Appliquer la notion

[solution n°6 p. 37]

Soit le programme ci-après écrit en JavaScript qui utilise la fonction *mystery* qui prend en entrée une liste de nombres.

```
1 function mystery (numberList) {
2   let max = numberList[0]
3   for (var i = 1; i < numberList.length; i++) {
4     if (max < numberList[i]) max = numberList[i]
5   }
6   return max
7 }
8
9 console.log(mystery([5, 7, 1, 0, 42, 3]))
10
```

La ligne commençant par `for` crée ce qu'on appelle une **boucle** : elle permet de parcourir la liste.

Exercice

Cette fonction permet de trouver :

- **A** Le plus petit nombre présent dans la liste.
- **B** Le plus grand nombre présent dans la liste.

Exercice

Quelle est la complexité de cette fonction ?

- **A** O(1)
- **B** O(n)
- **C** $O(n^2)$
- **D** $O(n^3)$
- **E** $O(n^4)$

XIV Essentiel

Tous les ordinateurs, que ce soit un portable, un poste de travail fixe, un serveur puissant, un téléphone ou même un simple objet connecté qui ne fait *que* compter des pas, sont construits selon un schéma similaire.

Ce sont des machines de Turing qui exécutent les **instructions** d'un **programme** pour modifier l'état d'une **mémoire**.

La conception technique des ordinateurs repose sur des micro-processeurs, une mémoire vive et des périphériques. Un programme est chargé de récupérer des données depuis les mémoires secondaires (les fichiers), afin de les traiter au sein de la mémoire vive, avant d'enregistrer à nouveau les données modifiées dans les mémoires secondaires.

Tous les programmes, quelque soit le langage dans lequel ils sont écrits (C, Java, Python, JavaScript, PHP) sont en fin de compte transformés en langage machine (ou assembleur). Mais il y a toujours plusieurs façons de résoudre un problème, et donc plusieurs suites d'instructions qui permettent de parvenir au même résultat. Et parmi celles-ci, certaines sont plus simples que d'autres. Le **calcul de complexité** permet de discriminer les solutions simples des plus coûteuses.

XV Quiz

Exercice 1 : Quiz - Culture

[solution n°7 p. 38]

Exercice

Alan Turing a construit une machine physique (nommée Machine de Turing) qui servit d'exemple par la suite.

- **A** Vrai
- **B** Faux

Exercice

Lorsqu'on change la taille des données d'un algorithme, celui-ci conserve-t-il la même complexité ?

- **A** Oui
- **B** Non

Exercice

L'architecture de Von Neumann partage-t-elle la mémoire pour les données et les instructions ?

- **A** Oui
- **B** Non

Exercice

Les processeurs des ordinateurs modernes regroupent les parties suivantes de l'architecture de Von Neumann :

- **A** Unité arithmétique et logique
- **B** Unité de contrôle
- **C** Mémoire
- **D** Entrées/Sorties

Exercice

Une machine de Turing est :

- **A** Un diagramme de transition
- **B** Un objet mathématique
- **C** Un automate
- **D** Un ordinateur respectant l'architecture de Von Neumann

Exercice

Qu'est-ce qui sert à traduire le langage du programmeur en langage machine ?

- **A** Interpréteur
- **B** Compilateur
- **C** Assembleur
- **D** Necronomicon

Exercice

Lequel de ces termes correspond à une optimisation (effectuée par le système d'exploitation) de la mémoire vive ?

- **A** swap
- **B** ramfs
- **C** ssh
- **D** Cloud computing

Exercice 9 : Quiz - Méthode

[solution n°8 p. 40]

Exercice

Que faut-il examiner en premier pour déterminer la complexité arithmétique d'un algorithme ?

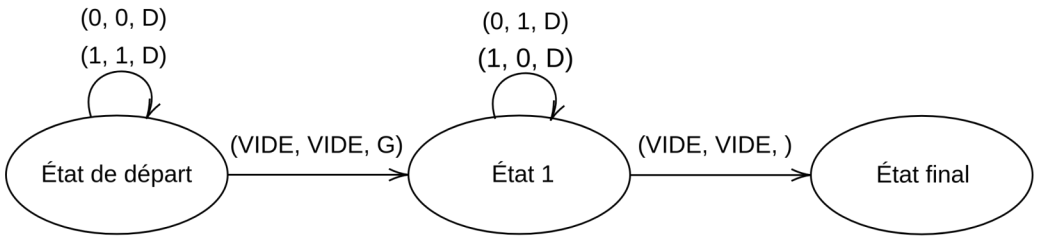
- **A** La taille des listes de données
- **B** Les valeurs des listes de données

C Le type d'opérations arithmétiques

D L'espace mémoire occupé par les données

Exercice

Voici le diagramme états-transitions d'une machine de Turing. Que fait cette machine ?



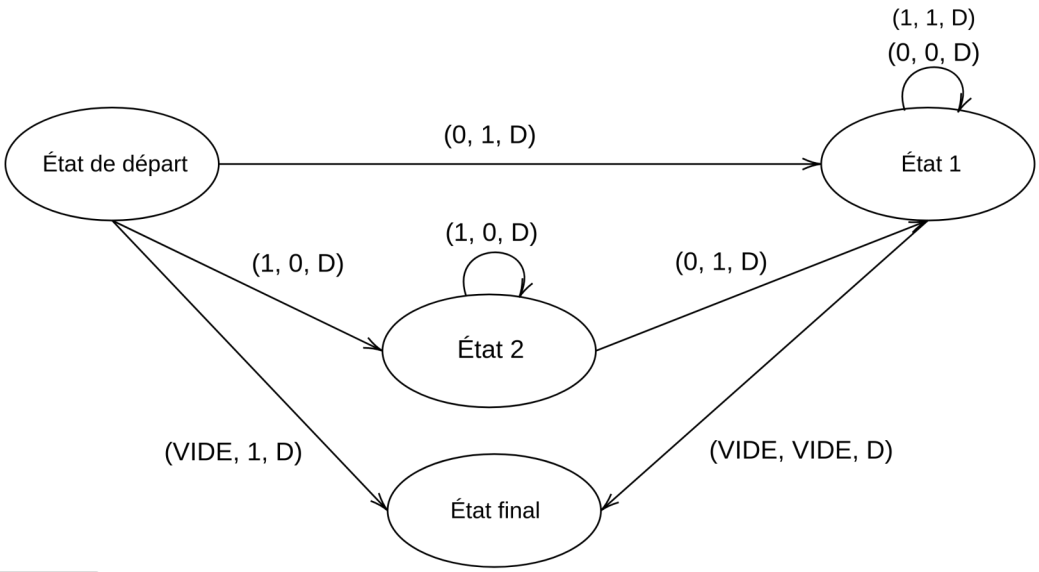
A Elle inverse le premier bit de la donnée d'entrée

B Elle inverse tous les bits de la donnée d'entrée

C Elle inverse le dernier bit de la donnée d'entrée

Exercice

Que renvoie la machine de Turing correspondant au diagramme états-transitions ci-dessous lorsqu'on lui donne entrée la valeur 1100 ?



Exercice 13 : Quiz - Code

[solution n°9 p. 42]

Exercice

Quelles instructions assembleur sont incorrectes ?

A `mov 1,2`

B `inc ax, 1`

C `add ax, bx`

D `inc ax`

Solutions des exercices

Solution n°1

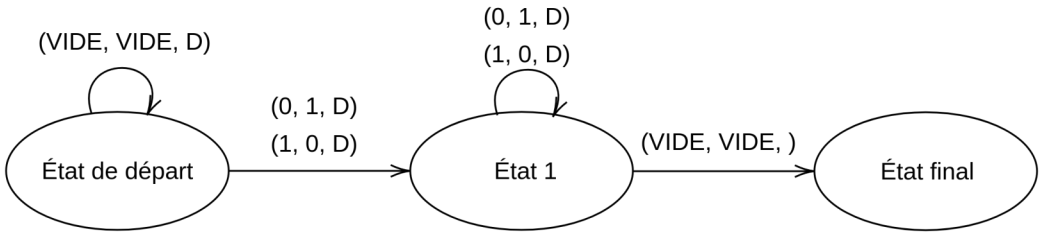
[exercice p. 8]

- La première utilisation du programme avec 101 en entrée produit le résultat 1010 (soit 10 = $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$)
- La première utilisation du programme avec 1010 en entrée produit le résultat 10100 (soit 20 = $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$)

Solution n°2

[exercice p. 12]

Que renvoie la machine de Turing correspondant au diagramme états-transitions ci-dessous lorsqu'on lui donne entrée la valeur 0110 ?



1001

On reste sur l'état de départ tant que la tête de lecture lit VIDE. On passe ensuite à l'état 1, sur lequel on reste tant qu'on lit une donnée. Chaque fois, cette donnée est inversée (lecture de 0 = écriture de 1, et inversement). Enfin, dès que la donnée est terminée, on passe à l'état final.

Solution n°3

[exercice p. 17]

Après avoir découvert l'architecture de Von Neumann, il est intéressant d'appliquer ce modèle en essayant de disséquer un *smartphone*. Le but de cet exercice sera de lier chaque partie de l'architecture à des composants électroniques d'un *smartphone*.

En cas de manque d'inspiration, il est possible de consulter le site mobilecellphonerepairing.com[6] qui liste les composants classiques d'un *smartphone*.

Aucune	ALU et unité de contrôle	Mémoire	Entrée	Sortie
Batterie	Processeur	RAM	Connecteur USB	Écran
		Carte SD	Dalle tactile	Vibreur
		Carte SIM	GPS	Haut-parleur

6. http://www.mobilecellphonerepairing.com/mobile-phone-parts.html

- L'architecture de Von Neumann est un modèle conceptuel qui est indépendant de toute solution technique. Ainsi, la source d'énergie permettant le fonctionnement de l'ordinateur n'est pas incluse dans le modèle.
- Le processeur d'un smartphone (composé ou non de plusieurs cœurs) inclut la ou les ALU ainsi que l'unité de contrôle.
- Les types de mémoire sont multiples dans un smartphone : RAM (mémoire vive), mémoire cache des processeurs (mémoire vive), disque dur (mémoire secondaire), carte SD (mémoire secondaire), carte SIM (mémoire secondaire).
- Les entrées/sorties d'un smartphone sont également multiples : dalle tactile, touches, connecteur USB, microphone, GPS, antenne, gyroscope, accéléromètre, écran, haut-parleur, vibreur etc.

Solution n°4

[exercice p. 21]

Exercice

Le BIOS correspond au micrologiciel contenu dans la carte mère qui s'exécute au démarrage de l'ordinateur.

En utilisant cette article Wikipédia sur le BIOS[7], quel type de mémoire stocke le BIOS ?

A Mémoire secondaire

B Mémoire virtuelle

C Mémoire vive

Le BIOS utilise une mémoire de type ROM, persistante sur le disque : c'est donc une mémoire secondaire.

Exercice

Redis est un système de gestion de bases de données non relationnelles très populaire.

Grâce à cette cet article Wikipédia[8], quels type de mémoire utilise Redis pour ses données?

A Mémoire secondaire

B Mémoire flash

C Mémoire virtuelle

D Mémoire vive

7. https://fr.wikipedia.org/wiki/BIOS_(informatique)
8. https://fr.wikipedia.org/wiki/Redis

Redis utilise prioritairement de la mémoire vive pour assurer un accès extrêmement rapide aux données. Cependant, il utilise également de la mémoire virtuelle (et donc de la mémoire secondaire via le *swap*) lorsque les données deviennent volumineuses.

Solution n°5

[exercice p. 24]

Ordonnez les instructions en assembleur basé sur l'architecture de l'Intel 8086 afin que le programme :

- incrémente la valeur contenue dans le registre AX,
- double cette valeur incrémentée,
- stocke cette valeur doublée dans BX,
- finisse en remettant à 0 la valeur contenue dans AX.

fr.wikipedia.org/wiki/Jeu d'instructions x86[9]

A INC ax | **B** IMUL ax,2 | **C** MOV bx,ax | **D** MOV ax,0

```
1 INC ax
2 IMUL ax,2
3 MOV bx,ax
4 MOV ax,0
```

Solution n°6

[exercice p. 29]

Exercice

Cette fonction permet de trouver :

A Le plus petit nombre présent dans la liste.

B Le plus grand nombre présent dans la liste.

Exercice

Quelle est la complexité de cette fonction ?

A O(1)

B O(n)

C $O(n^2)$

D $O(n^3)$

9. https://fr.wikipedia.org/wiki/Jeu_d%27instructions_x86#Instructions_originales_des_8086/8088

E $O(n^4)$

Sa complexité est O(n).

On pourrait dire O(2n) (voire O(4n) en comptant les affectations) pour être plus précis mais cela n'est pas nécessaire ; l'ordre de grandeur O(n) est plus instructif.

Solution n°7

[exercice p. 31]

Exercice

Alan Turing a construit une machine physique (nommée Machine de Turing) qui servit d'exemple par la suite.

A Vrai

B Faux

La Machine de Turing est un modèle mathématique permettant de formaliser la notion de calcul et d'algorithme. Sa structure même ne le rend pas possible à construire (ruban infini) même s'il est possible d'assimiler les ordinateurs modernes à des machines de Turing universelles.

Exercice

Lorsqu'on change la taille des données d'un algorithme, celui-ci conserve-t-il la même complexité ?

A Oui

B Non

Il conserve en effet la même complexité algorithmique car celle-ci est exprimée en fonction d'un taille abstraite notée **n**.

Exercice

L'architecture de Von Neumann partage-t-elle la mémoire pour les données et les instructions ?

A Oui

B Non

Les données et les instructions sont stockées au même endroit. C'est un des éléments qui sépare l'architecture de Von Neumann de l'architecture Harvard.

Exercice

Les processeurs des ordinateurs modernes regroupent les parties suivantes de l'architecture de Von Neumann :

A Unité arithmétique et logique

B Unité de contrôle

C Mémoire Cette réponse peut également être valable si on considère les « caches » des processeurs modernes qui confèrent une mémoire très petites aux processeurs.

D Entrées/Sorties

Exercice

Une machine de Turing est :

A Un diagramme de transition

B Un objet mathématique

C Un automate

D Un ordinateur respectant l'architecture de Von Neumann

Une machine de Turing est un objet mathématique : elle peut être représentée par un diagramme de transition mais ça n'est pas un diagramme de transition, ni un ordinateur.

Exercice

Qu'est-ce qui sert à traduire le langage du programmeur en langage machine ?

A Interpréteur Pour les langages interprétés comme le Python ou le JavaScript.

B Compilateur Pour les langages compilés comme le C/C++

C Assembleur Il s'agit du langage machine lui-même et non pas l'outil qui opère la traduction.

D Necronomicon

Exercice

Lequel de ces termes correspond à une optimisation (effectuée par le système d'exploitation) de la mémoire vive ?

A swap

B

ramfs Ce mécanisme permet de stocker des fichiers directement dans la mémoire vive. Ceci peut optimiser un calcul mais pas l'usage de la mémoire vive. Au contraire, cela augmente la charge sur la mémoire vive.

C ssh Il s'agit d'un protocole de communication avec un serveur distant.

D

Cloud computing D'une part le *cloud computing* est effectué par un tiers et non par le système d'exploitation lui-même. D'autre part, il ne s'agit pas d'optimiser la mémoire vive mais de sous-traiter le calcul.

Solution n°8

[exercice p. 32]

Exercice

Que faut-il examiner en premier pour déterminer la complexité arithmétique d'un algorithme ?

A La taille des listes de données

B Les valeurs des listes de données

C Le type d'opérations arithmétiques

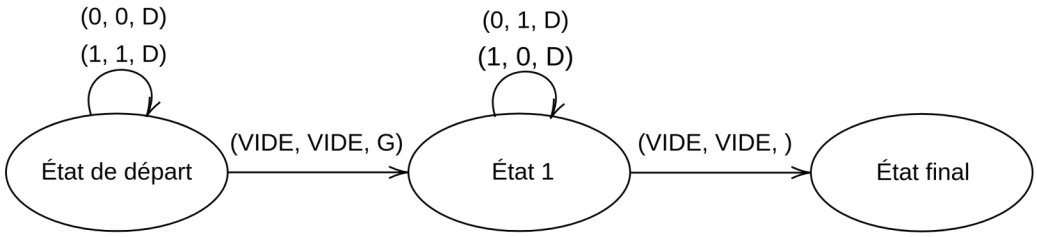
D L'espace mémoire occupé par les données

Les opérations arithmétiques sont considérées comme étant réalisées en un temps constant. Ni les valeurs effectives ni les opérations n'ont donc d'impact dans le calcul de la complexité arithmétique.

C'est la quantité de données qui détermine la quantité d'opérations et permet de connaître la complexité.

Exercice

Voici le diagramme états-transitions d'une machine de Turing. Que fait cette machine ?



A Elle inverse le premier bit de la donnée d'entrée

B Elle inverse tous les bits de la donnée d'entrée

C Elle inverse le dernier bit de la donnée d'entrée

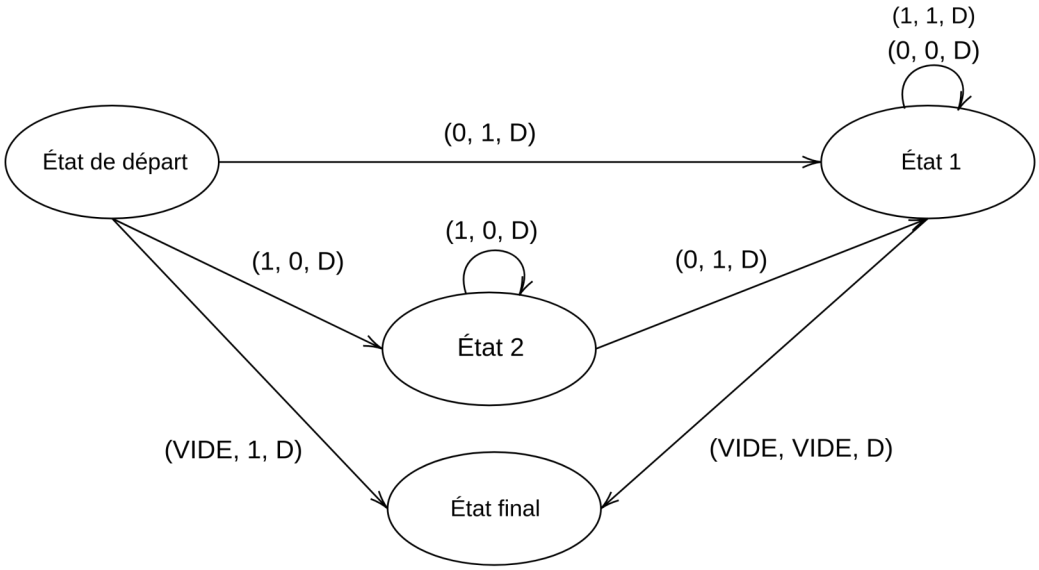
Le mieux pour s'en rendre compte est de choisir un mot et de suivre le diagramme.

L'analyse permet de constater que la tête de lecture se déplace à droite sans rien modifier jusqu'à arriver à la fin du mot, tant qu'elle rencontre des 0 et des 1.

À la fin du mot, quand la tête de lecture rencontre une case vide, elle se déplace à **gauche**, pour opérer.

Exercice

Que renvoie la machine de Turing correspondant au diagramme états-transitions ci-dessous lorsqu'on lui donne entrée la valeur 1100 ?



0010

- Le premier caractère (1) fait passer dans l'état 2, produit un 0 et se déplace à droite.
- Le deuxième caractère (1) fait rester dans l'état 2, produit un 0 et se déplace à droite.
- Le troisième caractère (0) fait passer dans l'état 3, produit un 0 et se déplace à droite.

- Le dernier caractère (0) fait rester dans l'état 3, produit un 0 et se déplace à droite.
- La donnée est finie, la tête de lecture rencontre une case vide, on arrive dans l'état final.

La machine de Turing représentée par ce diagramme états-transitions ajoute un caractère à la donnée d'entrée.

Solution n°9

[exercice p. 33]

Exercice

Quelles instructions assembleur sont incorrectes ?

A ☐ `mov 1,2` — Il est nécessaire d'avoir un registre comme destination du déplacement de données.

B ☐ `inc ax, 1` — Le principe de l'incrémentation est d'ajouter 1 à un entier. Il n'est donc pas nécessaire d'ajouter une valeur après le registre à incrémenter.

C `add ax, bx`

D `inc ax`

Crédits des ressources

Machine de turing p. 6
Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions - Wvbailey

Automate de portillon p. 10
Universel - Transfert dans le Domaine Public - ManiacParisien

Schéma d'une architecture Von Neumann p. 14
Attribution - Partage dans les Mêmes Conditions - Schéma original Kapooht, traduction Quentin Duchemin

Ordinateur portable ouvert p. 15
Attribution - Quentin Duchemin