Module 3: Ordinateur et processeur

1 Introduction

Dans ce module, nous allons présenter l'architecture de base d'un ordinateur et le principe de son fonctionnement.

L'architecture d'un ordinateur est fondée sur le modèle de Von Neumann. Ce modèle inclut:

- Un processeur: pour le traitement et exécution des programmes. Le processeur est composé de deux unités principales:
  - Unité arithmétique et logique
  - Unité de contrôle
- Une mémoire: pour le stockage des données et programmes;
- Un dispositif d'entrées/sorties: pour assurer l'échange avec l'extérieur.

Les composants sont reliés par un canal de communications (bus) constitué d'un ensemble de lignes électriques parallèles.

Le rôle du processeur est d'exécuter les instructions stockées dans la mémoire. Ces instructions sont généralement écrites par des programmeurs ou par le compilateur qui transforme ce que les programmeurs écrivent en un langage compréhensible par le processeur. Le processeur utilise ses composantes internes comme les séquenceurs, les registres, et l'unité arithmétique et logique pour pouvoir exécuter les instructions.

Nous allons présenter en détail toutes ces composantes en commençant d'abord par présenter l'architecture de base de Von Neumann, en suite nous aborderons le processeur et comment les instructions sont exécutées.

2 Architecture de Von Neumann

L'architecture de Von Neumann décrit le modèle de base de fonctionnement d'un ordinateur. Elle décompose l'ordinateur en 4 composantes importantes:

- l'unité arithmétique et logique (UAL ou ALU en anglais) ou unité de traitement: son rôle est d'effectuer les opérations de base;
- l'unité de contrôle (UC), chargée du séquençage des opérations;

- la mémoire qui stocke à la fois les données et les programmes à exécuter.
- les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.

Ces composantes peuvent être schématisées dans la figure suivante:

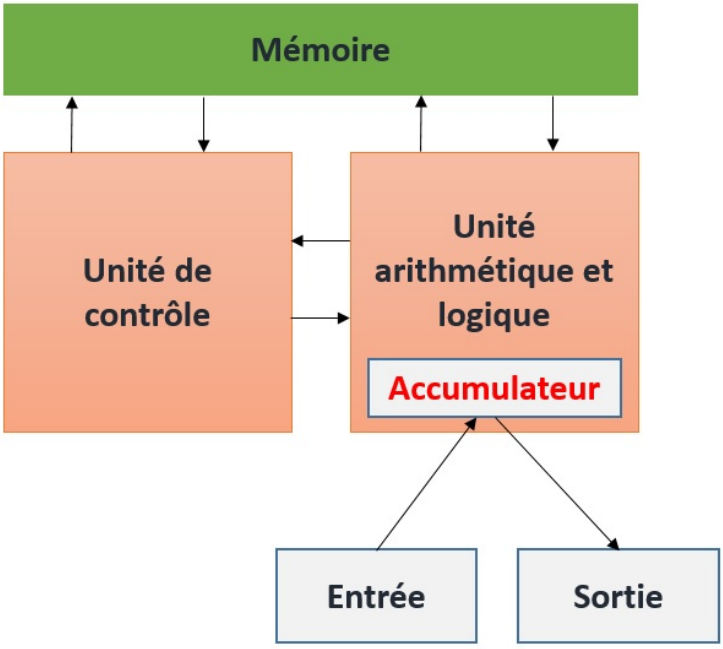

FIGURE 1 – Architecture de base de Von Neumann.

2.1 Unité arithmétique et logique (UAL)

L'UAL est un circuit logique combinatoire composé de portes logiques. Ces circuits logiques prennent deux nombres en entrée et génèrent un nombre en sortie.

L'UAL est responsable de réaliser les opérations de base: opérations logiques (et, ou et non), opérations arithmétiques, comparaisons. Un code d'entrée détermine la partie du circuit qui va fournir le résultat.

Les opérations logiques peuvent être résumées dans les points suivants:

- ET logique (AND),
- OU logique (OR),
- NON logique (NOT),
- OU exclusif logique (XOR),

- NON ET logique (NAND),
- NON OU logique (NOR).

La figure suivante présente les circuits logiques associés aux différentes opérations logiques.

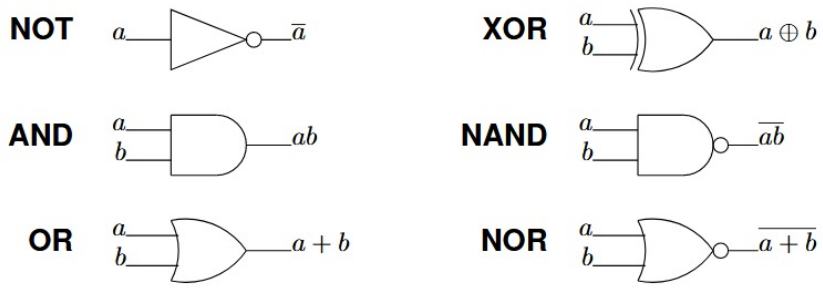

FIGURE 2 – Circuits logiques associés aux opérations logiques.

Pour comprendre les entrées et sorties pour ces circuits logiques, nous utilisons ce qu'on appelle la **table de vérité**. Dans ce qui suit, nous allons présenter les tables de vérité pour chaque opérateur logique.

Table 1: Tables de vérité des opérateurs logiques: a et b représentent les entrées, et s représente la sortie.

ET (AND)

a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

OU (OR)

a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

NON ET (NAND)

a	b	s
0	0	1
0	1	1
1	0	1
1	1	0

NON OU (NOR)

a	b	s
0	0	1
0	1	0
1	0	0
1	1	0

NON (NOT)

a	s
0	1
1	0

OU exclusif (XOR)

a	b	s
0	0	0
0	1	1
1	0	1
1	1	0

Comme indiqué dans la table 1, les résultats sont différents pour chaque opérateur. Les résultats sont interprétés comme suit:

- AND: le résultat sera VRAI (1) si la valeur a ET la valeur b sont vraies.
- OR: le résultat sera VRAI si la valeur a OU la valeur b sont vraies.
- NOT: le résultat sera toujours l'inverse de la valeur d'entrée.
- XOR: le résultat sera VRAI si la valeur a est vraie OU si la valeur b est vraie, mais pas les deux en même temps.
- NAND: le résultat sera l'inverse de AND.
- NOR: le résultat sera l'inverse de OR.

L'UAL est le composant central qui permet de réaliser tous les calculs. Grâce à des bus, l'UAL peut accéder:

- aux registres,
- à la mémoire centrale,
- aux différents périphériques d'entrées/sorties.

2.2 Unité de contrôle

L'unité de contrôle (UC), appelée aussi unité de commande ou encore séquenceur, est chargée de commander et de gérer tous les différents constituants de l'ordinateur (contrôler les échanges, gérer l'enchaînement des différentes instructions, etc...). Elle contrôle, interprète les instructions, lit le compteur d'instructions et fait la séquence d'actions correspondantes au cycle d'exécution des instructions.

L'UC est composée:

- d'un registre instruction RI,
- d'un compteur ordinal CO,
- d'un registre adresse RA,
- d'un décodeur de fonctions,
- d'une horloge.

L'UC contrôle toutes les parties du système. Elle gère les quatre opérations de base du cycle d'exécution appelé **Cycle Fetch/Execute** ou parfois nommé aussi **Cycle Fetch/Decode/Execute** comme suit:

- **Fetch**: récupère la prochaine commande de programme de la mémoire de l'ordinateur,
- **Decode**: déchiffre ce que le programme demande à l'ordinateur de faire,
- **Execute**: effectue l'action demandée,
- **Store**: enregistre les résultats de l'exécution dans les registres ou la mémoire.

2.3 Mémoire

La mémoire centrale est constituée de cellules pour enregistrer à la fois les programmes et les données. La figure suivante montre un exemple de mémoire DELL d'un ordinateur. La figure 3 est tirée du site web de DELL `http://www.dell.com`.

FIGURE 3 – Exemple d'une barrette de mémoire DELL.

La mémoire centrale permet le transfert des données en effectuant les opération de chargement des données (du registre vers la mémoire centrale) et de stockage des données (de la mémoire centrale vers le registre). Chaque cellule dans la mémoire possède une adresse.

L'accès à la mémoire centrale est beaucoup plus lent que l'accès aux registres.

La mémoire principale est reliée au processeur via le bus principal (ensemble de fils véhiculant les informations et les commandes).

2.4 Dispositifs d'entrées/sorties

Les dispositifs d'entrées/sorties permettent à l'ordinateur d'échanger de l'information avec le monde extérieur comme le clavier, la souris, l'écran, l'imprimante, etc.

La variété des périphériques exige que l'ordinateur soit capable de se connecter de façon flexible à des périphériques répondant à des commandes différentes

et opérant sur des vitesses différentes. De façon générale, les utilisateurs changent ou ajoutent des périphériques de manière fréquente contrairement au processeur ou à la mémoire centrale qui restent inchangés.

Pour des raisons de flexibilité et de variété, les périphériques d'entrées/sorties ne peuvent pas être reliés directement au bus principal, car cela empêcherait complètement les possibilités d'évolution [1]. Par conséquent, et dans le but de garantir la flexibilité, les dispositifs d'entrées/sorties sont donc branchés sur un bus d'entrées/sorties secondaire, normalisé suivant les standards internationaux. Cela permettra de garantir une compatibilité de matériels et surtout une évolution aisée comme le montre la figure 4.

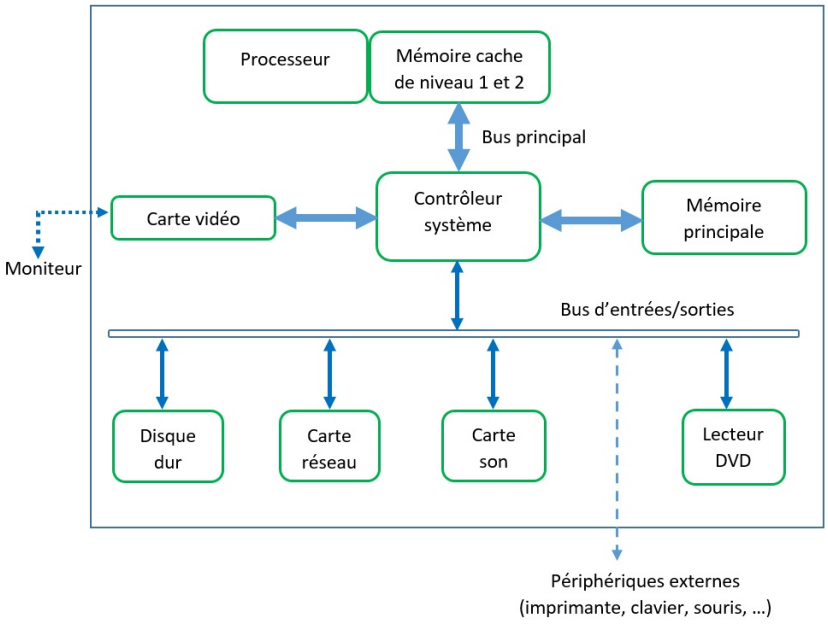

FIGURE 4 – Structure d'un ordinateur.

La seule exception est la carte vidéo comme nous pouvons le constater dans la figure 4 qui est connectée directement au bus principal. Cela est dû principalement à l'augmentation phénoménale dans les besoins en visualisation. Les dispositifs de visualisation maintenant ont besoin d'un énorme débit d'informations entre le processeur et la carte vidéo et qui nécessite d'être traité de façon beaucoup plus rapide, une chose qui ne pourra jamais être satisfaite avec un bus d'entrées/sorties beaucoup top lent. La figure 4 montre comment la carte vidéo est connectée au processeur via le contrôleur système.

[1] Texte modifié tiré du livre d'architecture de l'ordinateur d'Emmanuel Lazard.

3 Instructions

Le processeur est chargé d'exécuter les instructions se trouvant en mémoire centrale. Si les instructions se trouvent dans un autre emplacement comme le disque dur par exemple, le processeur charge d'abord ces instruction en mémoire puis les exécute.

Chaque instruction est caractérisée par un code numérique correspondant à l'opération requise. Les codes sont stockés en mémoire à la suite les uns des autres dans des cases mémoires successives. Cela permettra de repérer plus facilement chaque instruction par son adresse.

Pour exécuter une instruction, le processeur procède comme suit:

- Le processeur envoi un ordre de lecture à la mémoire en spécifiant l'adresse de l'instruction souhaitée.
- La mémoire renvoie alors le code numérique de l'instruction au processeur.
- Le processeur exécute l'instruction.
- Le processeur passe à l'instruction suivante.

L'ensemble des instruction que le processeur peut exécuter, appelé aussi jeu d'instructions (instruction set), et la correspondance entre chacune d'entre elles et son code numérique, sont définis par le constructeur du processeur et lui sont propres. Cette Information est tirée du livre d'architecture de l'ordinateur d'Emmanuel Lazard.

3.1 Types d'instructions

Dans cette section, nous allons présenter les différents types d'instructions que l'on trouve généralement dans le jeu d'instructions des processeurs. Ces types peuvent être regroupés en cinq (5) catégories décrites comme suit:

- **Transfert de données**: nous pouvons demander explicitement le transfert de données entre la mémoire et un registre. L'instruction doit spécifier la case mémoire cible, le registre, ainsi que la taille de la donnée à transférer. Par exemple, chaque case mémoire correspond à 1 octet, alors que les registres ont généralement entre 4 et 8 octets. Donc, si le nombre d'octets dépasse 1 dans le cas de la mémoire, nous devons donc chercher les données dans des cases mémoires successives.
- **Opérations arithmétiques et logiques, décalages et rotations**: ces instructions permettent d'exécuter des opérations arithmétiques et logiques sur des données se trouvant dans les registres. Notez que chaque instruction permet une seule opération. Par exemple, un calcul qui nécessite plusieurs opérations requiert plusieurs instructions élaborées instruction

par instruction. L'UAL permet également d'exécuter des instructions de décalage et de rotation des bits dans des registres. La rotation consiste à remplacer chaque bit par celui situé directement à gauche ou à droite. Le décalage ressemble à la rotation à l'exception qu'un bit disparaît et un 0 (éventuellement un 1) est mis dans le premier bit. Ces deux opérations sont assez importantes car le décalage à droite représente une division par 2 comme nous l'avons vu dans module 2, et une décalage à gauche représente une multiplication par 2 qui sont beaucoup plus rapides que l'exécution des opérations de division ou de multiplication proprement dite. Le décalage et la rotation sont illustrés dans les figures 5 et 6 respectivement.

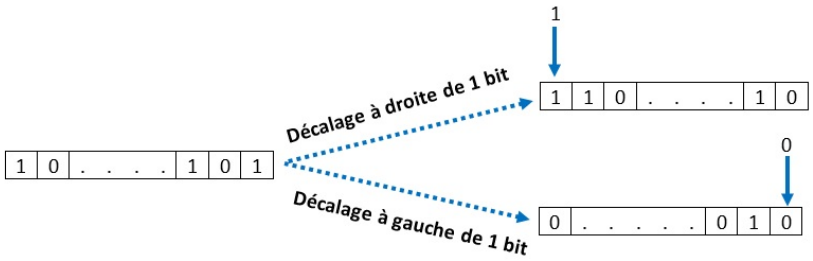

FIGURE 5 – Exemple de décalage d'un bit.

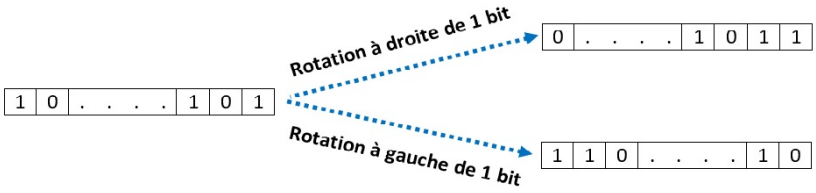

FIGURE 6 – Exemple de ration d'un bit.

- **Tests et comparaisons**: ces instructions permettent d'effectuer des comparaisons entre deux données (égalité, inégalité). À la lumière des résultats de comparaisons, nous pouvons définir des comportements différents. Le changement de comportement, appelé aussi état, est indiqué dans un registre spécial appelé registre d'état. Notez aussi que les données dans les registres ne changent pas dans le cas des comparaisons.
- **Ruptures de séquences**: les instructions sont exécutées successivement en mémoire. Cependant, cette façon de faire ne permet pas d'avoir des exécutions différentes selon les données. Nous avons deux types de ruptures de séquences: 1) les sauts, appelés aussi "branchements" inconditionnels qui transfèrent de façon systématique l'exécution à une autre adresse

au lieu d'exécuter l'instruction suivante dans la mémoire, et 2) les sauts conditionnels qui transfèrent l'exécution uniquement si une condition est vérifiée. La figure 7 montre un exemple de branchement[2].

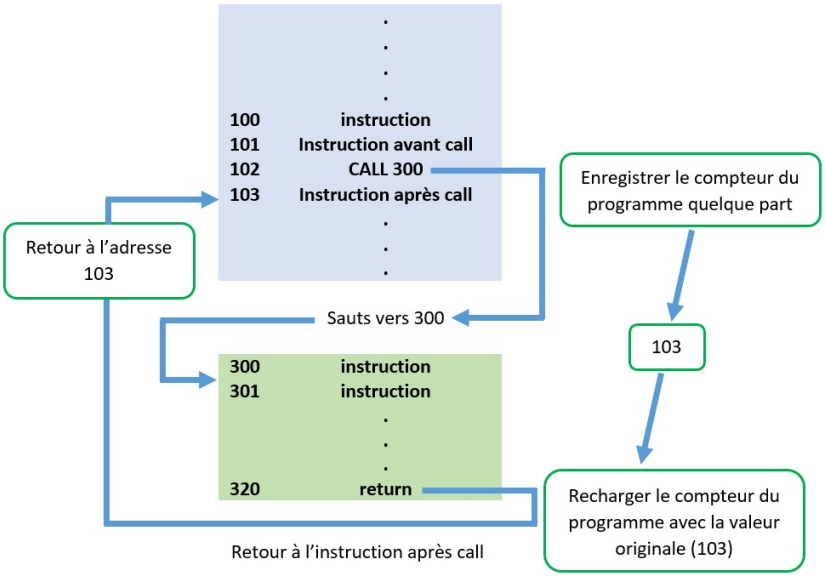

FIGURE 7 – Exemple de rupture de séquence.

- **Instructions divers**: ces instructions incluent par exemples les instructions d'entrées/sorties (envoi d'une commande à un périphérique), le contrôle du processeur (arrêter ou redémarrer un processeur), la gestion des interruptions, etc.

3.2 Format des instructions

Une instruction peut être divisée en deux parties:

- l'opération à exécuter, appelée "code opération". Elle correspond aux opérations (mnémoniques) utilisées par l'assembleur comme (ADD, MOV, SUB, etc.).

- les données sur lesquelles se porte l'opération. Ces données doivent être spécifiées via des modes d'adressage qui indiquent l'emplacement des données (registres, mémoire, etc.). Notez qu'en général, nous avons besoin besoin de spécifier également l'emplacement où ranger les résultats d'une opération. Par exemple, nous pouvons écrire en langage assembleur ADD r1, r2, r3 pour indiquer que le résultat d'addition des données se trouvant

[2]Exemple inspiré du livre Little Man Computer, Chapitre 6.

dans les registres r2 et r3 doit être placé dans le registre r1. Mais, cette écriture n'est pas la seule représentation, c'est le concepteur qui spécifie la manière d'indiquer où stocker les résultats.

Le code opération indique au processeur l'opération à effectuer parmi celles qui existent. Le mnémonique correspondant à l'opération se traduit donc en une valeur numérique dont les bits spécifient quelles actions doivent intervenir à l'intérieur du processeur. Ces bits du code opération sont utilisés par le séquenceur pour l'envoi des commandes aux différents composants internes du processeur [3]. La figure 8 présente un exemple de format d'instruction.

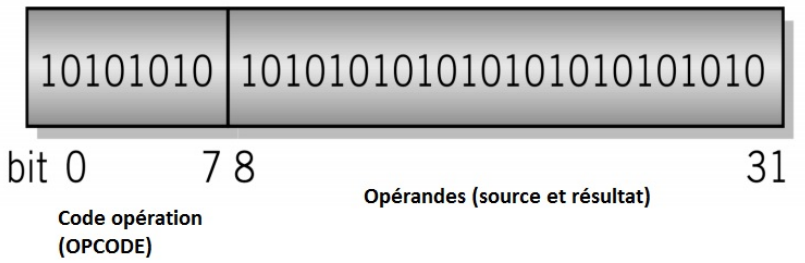

FIGURE 8 – Exemple de format d'instruction.

[3] Texte tiré du livre architecture de l'ordinateur d'Emmanuel Lazard.