Chapitre 2
Structure et fonctionnement d'un ordinateur

L'utilisation d'aides mécaniques pour faire des calculs date de plusieurs siècles. L'invention de l'abaque date d'avant l'histoire écrite. La première machine qui correspond a ce qu'on appelle maintenant un ordinateur a été définie par Charles Babbage en 1830. Il a voulu remplacer les hommes qui faisaient des calculs de tables fastidieux (logarithmes, etc.) à la main par des machines programmables. Ses idées étaient cent ans en avance sur la technologie et il a passé quarante ans à essayer de réaliser, sans succès, un ordinateur mécanique appelé « machine analytique ».

En 1937, Howard Aiken de l'université Harvard, propose une machine basée sur les idées de Babbage et sur les calculateurs électromécaniques d'IBM. La construction de cette machine, appelée MARK I, a été subventionnée par Harvard et IBM et sa réalisation, commencée en 1939, s'est achevée en 1944. Cette machine électromécanique, principalement composée de commutateurs et de relais, était extrêmement lente.

Le développement d'un autre ordinateur entièrement électronique, l'ENIAC, été entrepris en 1943 à l'université de Pennsylvanie (Moore School of Electrical Engineering), sous la direction de J.P. Eckert et J.W. Mauchly, avec l'aide du mathématicien John Von Neumann. Ce dernier est souvent considéré comme le personnage le plus important de l'histoire des ordinateurs parce que tous les ordinateurs qui ont été construits depuis son époque sont basés sur les concepts qu'il a énoncés. L'ENIAC a été mis en service le 15 février 1946.

2.1 Définition d'un ordinateur

Le MARK I de l'université Harvard était un ensemble de machines séquentielles électromécaniques travaillant parallèlement sur un même problème et l'ENIAC était une version électronique de la même machine. Von Neumann a alors tiré deux conclusions de l'expérience de l'ENIAC: premièrement, avec l'électronique, une seule unité centrale de traitement suffit; deuxièmement, il peut être intéressant de stocker des programmes en mémoire pour réutiliser des sections de programmes, ou sous-programmes, plusieurs fois.

Von Neumann a alors énoncé les cinq critères essentiels qui définissent un ordinateur, même de nos jours:

1. Un ordinateur doit posséder un médium d'entrée par lequel des quantités virtuellement illimitées d'instructions et de données peuvent être introduites.

2. Un ordinateur doit posséder une unité d'emmagasinage de laquelle on peut obtenir les instructions et les opérandes et dans laquelle on peut stocker des résultats, lorsque désiré.

3. Un ordinateur doit posséder une unité de calcul capable d'effectuer des opérations logiques ou arithmétiques sur n'importe quel opérande pris dans la mémoire.

4. Un ordinateur doit posséder un médium de sortie, par lequel des quantités virtuellement illimitées de données peuvent être transmises à l'extérieur de l'ordinateur.

5. Un ordinateur doit posséder une unité de contrôle, capable d'interpréter les instructions obtenues de la mémoire et capable de choisir parmi plusieurs actions en fonction de résultats calculés.

La structure résultant de ces cinq critères est appelée « structure de Von Neumann ». Depuis cette époque, on a grandement augmenté la puissance des ordinateurs, mais leur structure générale reste relativement conforme à la structure de Von Neumann. Ce n'est qu'en juin 1948 que l'ordinateur Mark 1 de l'université de Manchester a été mis en service, le premier ordinateur à programme enregistré en mémoire; il avait des mots mémoire de 32 bits et l'exécution d'une instruction prenait 1,2 milliseconde.

2.2 Organisation générale des ordinateurs

La majorité des ordinateurs peuvent être décrits par le schéma de la figure 2.1, bien que la façon de les réaliser puisse varier considérablement d'un modèle à un autre. Cette figure montre en particulier que le traitement informatique met en jeu la circulation de données entre l'unité centrale de traitement, la mémoire et les dispositifs d'entrée/sortie. Au cours de l'exécution d'un programme les instructions et les données résident en mémoire et font alors de celle-ci un point focal du cycle de traitement. Au niveau du code, un programme comporte instructions et données résidant en mémoire. Pour chaque instruction le processeur va chercher un code opération en mémoire; ce code spécifie à la fois l'opération et les données utilisées par l'opération. Ces données sont placées immédiatement après le code opération dans ce qu'on appelle des mots d'extension de l'instruction.

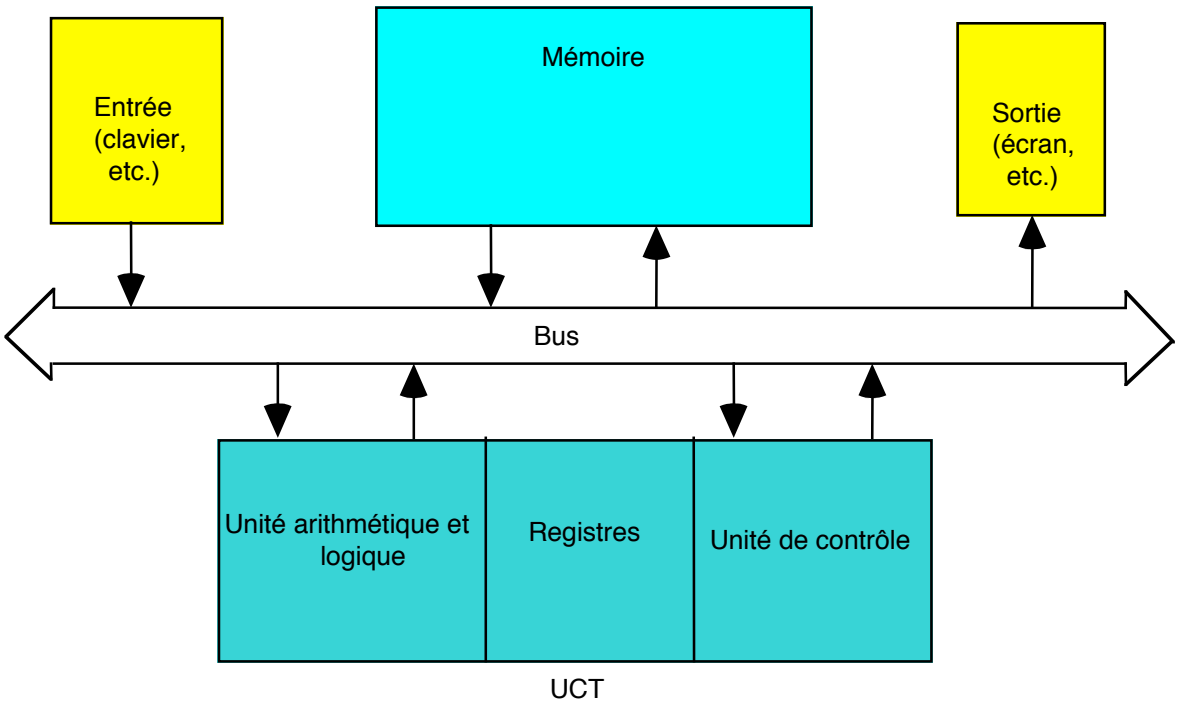


Figure 2.1 Schéma général d'un ordinateur.

Il existe actuellement deux grands types de microprocesseurs: les processeurs CISC et les processeurs RISC. Ces deux appellations correspondent en fait à deux philosophies différentes.

Les processeurs CISC (*complex instruction-set computer*) comprennent la plupart des microprocesseurs actuels. Ils possèdent une grande variété d'instructions conçues pour des traitements spécifiques et qui permettent à certains algorithmes d'être exécutés très efficacement. Les processeurs de la famille MC68000 et le processeur PEP 8 sont des processeurs CISC.

Les processeurs RISC (*reduced instruction-set computer*) sont conçus à partir d'un petit ensemble d'instructions qui s'exécutent en un cycle d'horloge (voir 2.2.4). En théorie un processeur RISC utilise un grand nombre de registres internes pour créer du code simple et très efficace. Les processeurs modernes IBM RS6000 et IBM/Motorola PowerPC sont des processeurs RISC.

2.2.1 La mémoire

La mémoire est l'élément central de l'ordinateur parce que toutes les instructions traitées par les autres sections passent par elle. Voici l'ensemble des caractéristiques de la mémoire:

- La mémoire est un ensemble de cellules, chaque cellule pouvant contenir une certaine quantité d'information. À la naissance des micro-ordinateurs le nombre de cellules était aux alentours de 340K (lK = 1024), alors qu'à l'heure actuelle, ce nombre est de plusieurs centaines de Megs (1 Meg = 1024^2), selon l'ordinateur.

- À chaque cellule est associée une adresse par laquelle les autres composantes la désignent. Ces adresses sont des nombres variant normalement entre 0 et le nombre de cellules de mémoire -1. Par exemple, dans une mémoire de 4K, les cellules de mémoire auront des adresses allant de 0 à 4095.

- Chaque cellule est composée d'un nombre fixe de bits. Un bit est l'unité fondamentale d'information traitée par un ordinateur. Chaque bit peut avoir deux valeurs: 0 ou 1. Une suite de bits nous permet donc de représenter un nombre binaire, par exemple, une cellule de huit bits permet de représenter tous les nombres binaires entre 00000000 et 11111111, ce qui équivaut à des nombres entre 0 et 255 en base 10. Le nombre de bits par cellule peut varier entre 8 et 64 bits selon le modèle d'ordinateur. Une cellule de 8 bits est appelée un « *octet* » (ou *byte* en anglais).

- Une cellule de mémoire ne peut contenir qu'une seule valeur à la fois, le stockage d'une nouvelle valeur détruisant l'ancienne valeur contenue dans la cellule.

- L'unité fondamentale d'information que peut traiter un ordinateur, lors d'opérations arithmétiques de base, est appelée un « *mot de mémoire* ». Le mot peut contenir une ou plusieurs cellules de mémoire adressables. Quelques exemples pour des processeurs anciens:

INTEL 8080:	mot de 8 bits, 1 cellule de 8 bits
MC68000:	mot de 16 bits, 2 cellules de 8 bits
IBM 370:	mot de 32 bits, 4 cellules de 8 bits
VAX 11:	mot de 32 bits, 4 cellules de 8 bits
DEC 10:	mot de 36 bits, 1 cellule de 8 bits
CYBER 170:	mot de 60 bits, 10 cellules de 6 bits
CYBER 180:	mot de 64 bits, 8 cellules de 8 bits

Actuellement la taille des mots mémoire est de 16 ou de 32 bits, mais avec les nouveaux processeurs de 64 bits, cette taille va grandir.

- Les cellules de mémoire ne peuvent contenir que des nombres. Ceci implique que toute l'information traitée par l'ordinateur doit être codée sous forme numérique. Cela comprend les instructions machines qui se trouvent, elles aussi, en mémoire centrale. Par exemple, chaque cellule de mémoire d'un ordinateur *hypothétique* peut contenir 5 chiffres décimaux dont deux représenteront le code d'opération et trois l'adresse de l'opérande; nous pourrons alors avoir les trois instructions suivantes:

21306	obtenir (21) la valeur rangée à l'adresse 306
32308	ajouter (32) la valeur rangée à l'adresse 308
16310	ranger (16) la valeur à l'adresse 310.

Ces trois instructions seront elles-mêmes rangées dans trois cellules de mémoire consécutives comme suit:

200	21306
202	32308
204	16310

- L'accès aux cellules de mémoire est très rapide. Le temps nécessaire à la lecture ou à l'écriture d'instructions ou de données est mesuré en nanosecondes. Une nanoseconde représente un milliardième de seconde, soit un millième de microseconde ou 10^{-9} seconde. Un temps d'accès mémoire typique est habituellement compris entre 80 et 120 nanosecondes.

En résumé, la mémoire est un ensemble de cellules adressables, toutes de même taille, regroupées en mots et ne pouvant contenir que des valeurs numériques. On rencontre, sur le marché, une grande variété de types et d'organisations de mémoire. Les deux extrêmes sont: petite, rapide et coûteuse, d'une part, et grande, lente (relativement) et peu coûteuse, d'autre part.

On utilise la *mémoire vive* pour ranger les instructions et les données d'un programme. La plupart des programmes utilisent cette mémoire dite à accès aléatoire ou *RAM* (*random-access memory*) qui permet à la fois les opérations de lecture et d'écriture. De tels programmes modifient le contenu de la mémoire utilisée.

Certains programmes font partie du système d'exploitation et résident dans une partie spéciale de mémoire, la *mémoire morte* ou *ROM* (*read-only memory*), une mémoire où le seul accès permis en est un de lecture. Ces programmes incluent des procédures d'initialisation, des procédures de communication avec clavier, écran et disques, et permettent à l'utilisateur de communiquer avec le système. Ces programmes définissent le logiciel de contrôle du système et ne doivent pas être modifiés: la mémoire ROM offre cette protection.

Habituellement la mémoire ROM est placée sur le circuit intégré principal (*motherboard*) sous forme de puce séparée. Une puce ROM est créée comme tout circuit intégré avec les données « cablées » dans le

circuit. Ceci présente deux problèmes: le coût d'insertion des données est relativement élevé, et il n'y a aucune marge d'erreur (si un bit est erroné il faut jeter tout le lot de puces ROM).

Une autre possibilité moins coûteuse est appelée *PROM* (*programmable read-only memory*). Pour ces puces, le processus d'écriture est fait électroniquement et peut être fait à un autre moment que la fabrication. Si le logiciel système doit être mis à jour on remplace les puces PROM du circuit principal.

Certaines puces PROM sont réutilisables car leur contenu peut être effacé et la puce peut être reprogrammée. De telles puces sont appelées *EPROM* (*erasable programmable read-only memory*). Elles sont effacées par exposition aux rayons ultra-violets pendant une vingtaine de minutes. Ces puces sont plus coûteuses que les puces PROM mais peuvent être réutilisées un certain nombre de fois.

Enfin il existe des puces *EEPROM* (*electrically erasable programmable read-only memory*) dans lesquelles on peut écrire à tout moment sans avoir à effacer au préalable les contenus antérieurs. L'opération d'écriture prend beaucoup plus de temps que l'opération de lecture, c'est à dire de l'ordre de plusieurs centaines de microsecondes par octet. Ces puces sont bien entendu plus coûteuses que les précédentes.

Le système d'exploitation des anciens ordinateurs Macintosh illustre bien l'utilisation de la mémoire ROM. En effet, ce système graphique faisait appel à un grand nombre de procédures regroupées dans ce que l'on appelait la boîte à outils (« toolbox »). Ces procédures étaient conservées dans le ROM de la boîte à outils, formaient le coeur du système MacOS et ne pouvaient être changées. Ces procédures étaient habituellement écrites en assembleur pour des raisons d'efficacité. Il existait, de plus, des procédures additionnelles qui étaient conservées en mémoire RAM.

2.2.2 Les unités d'entrée-sortie

Les unités d'entrée et de sortie servent à établir la communication entre l'ordinateur et le monde extérieur. Elles servent à convertir les données d'une forme de représentation connue de l'ordinateur à une forme de représentation connue du monde extérieur et vice-versa. Le monde extérieur est composé d'humains, de capteurs, de machines-outils, de centrales nucléaires, etc. Ces unités servent aussi à permettre l'enregistrement de données pour une réutilisation ultérieure (disques, bandes magnétiques).

2.2.3 L'unité centrale de traitement

L'unité centrale de traitement (UCT) est composée de deux parties: l'unité de contrôle et l'unité de calcul. L'unité de calcul sert à exécuter les opérations logiques et arithmétiques sur des opérandes fournis par l'unité de contrôle. Le nombre d'opérations qui peuvent être réalisées par l'unité de calcul et leur vitesse d'exécution peuvent varier beaucoup d'un ordinateur à un autre (entre 50 et 400 opérations différentes, les vitesses variant généralement entre 10^{-9} et 10^{-6} seconde pour l'exécution d'une instruction). L'unité de contrôle de l'UCT lit les instructions en mémoire, les décode et envoie les signaux aux autres éléments de l'ordinateur pour qu'une instruction soit exécutée. Elle reçoit aussi des résultats de l'unité de calcul pour compléter l'exécution des instructions qui permettent des choix. Nous étudierons l'unité centrale de traitement un peu plus en détail dans la section suivante.

2.2.4 L'horloge

Les activités de l'unité centrale de traitement sont synchronisées au moyen de l'horloge du système, laquelle engendre un signal régulier, semblable au tic tac d'une horloge traditionnelle. Toutes les activités de l'UCT sont ainsi mesurées en cycles d'horloge. Les vitesses d'horloge sont mesurées en mégahertz (MHz) ou en gigahertz (GHz). Un mégahertz représente un million de hertz ou un million de cycles par seconde. Si une UCT évolue à 300 MHz, il y a 300 millions de cycles d'horloge par seconde: une instruction qui requiert 6 cycles d'horloge sera donc exécutée en un temps de:

$$6(1/300\ 000\ 000) = 20 \text{ ns } (20 \times 10^{-9})$$

Sur un autre processeur dont la fréquence est de 1 GHz, la durée sera de $6 \times 10^{-9} = 6$ ns.

2.2.5 Les bus

L'unité centrale de traitement communique avec la mémoire et les dispositifs périphériques au travers d'un bus. Un bus est un ensemble de lignes de communications qui permet le transfert d'adresses, de données et de signaux qui représentent de l'information entre les divers composants du système. Dans la plupart des ordinateurs le bus est divisé en composantes fonctionnelles: bus d'adresses, bus des données et bus de contrôle.

Bus d'adresses

L'UCT accède à une position mémoire en en plaçant l'adresse dans le bus d'adresses et en exécutant ensuite l'opération de lecture ou d'écriture. Le nombre de lignes du bus d'adresses détermine le domaine des adresses possibles. Comme chaque ligne du bus correspond à un bit de la valeur adresse, une UCT ayant un bus d'adresses de 16 bits ne peut accéder que 2^{16} (65536) octets de mémoire. Le processeur Motorola 68000 (datant du début des années 80) possèdait un bus d'adresses de 24 bits qui pouvait accéder aux adresses comprises entre 000000_{16} et $FFFFFF_{16}$. La taille maximum de mémoire que l'on pouvait adresser de cette manière était donc de 2^{24} ou approximativement de 16 millions d'octets, choix qui montrait clairement une bonne vision du futur.

Si l'on donne une adresse inexistante au bus d'adresses, il se produit une erreur de bus (« bus error »). Si votre ordinateur possédait 4 Mega-octets de mémoire RAM (un comble!) et que vous donniez l'adresse 500000_{16} au bus d'adresses, il y aura erreur de bus. Pourquoi? Un Mega-octet est égal à $1K^2$ soit 2^{20} octets (rappelez-vous que $1K = 1024 = 2^{10}$).

Les bus d'adresses des processeurs MC68020, MC68030 et MC68040 avaient 32 bits ce qui permettait d'adresser 4 Giga-octets (2^{32} octets). Le bus d'adresse du modèle G5 de Apple, basé sur un processeur PowerPC de 64 bits développé par la compagnie IBM, pourrait avoir jusqu'à 64 bits, pour une taille maximale gigantesque (16 Exa-octets, voir table ci-dessous). Le bus d'adresse du G5 ne possède que 42 bits, ce qui lui donne une possibilité de 2^{42} = 4 tera-octets; Apple a pourtant limité la taille mémoire à un maximum confortable de 8 Giga-octets (le double du maximum des PC). Lorsqu'on parle de tailles de mémoire, il faut se rappeler que Bill Gates a dit au début des années 80 du siècle dernier : « je ne vois pas pourquoi les gens auraient besoin d'un Meg de mémoire ».

http://en.wikipedia.org/wiki/Exabyte

Multiples of bytes as defined by IEC 60027-2					
SI prefix			Binary prefixes		
Name	Symbol	Multiple	Name	Symbol	Multiple
kilobyte	kB	10^3 (or 2^{10})	kibibyte	KiB	2^{10}
megabyte	MB	10^6 (or 2^{20})	mebibyte	MiB	2^{20}
gigabyte	GB	10^9 (or 2^{30})	gibibyte	GiB	2^{30}
terabyte	TB	10^{12} (or 2^{40})	tebibyte	TiB	2^{40}
petabyte	PB	10^{15} (or 2^{50})	pebibyte	PiB	2^{50}
exabyte	EB	10^{18} (or 2^{60})	exbibyte	EiB	2^{60}
zettabyte	ZB	10^{21} (or 2^{70})			
yottabyte	YB	10^{24} (or 2^{80})			

Cette table, trouvée sur Internet, donne les noms correspondant à différentes tailles de mémoire en octets. Notez que les préfixes du système international sont ambigus, indiquant soit une puissance de dix, Exa-octets = 10^{18} soit un trillion sur l'échelle longue (voir à ce sujet la table des préfixes à l'uri http://en.wikipedia.org/wiki/SI_prefix), soit une puissance de deux, Exa-octets ou Exbi-octets = 2^{60} soit 1 152 921 504 606 846 976 octets (ce qui se lit : un trillion 152 mille 921 billions 504 milliards 606 millions 846 mille 976). Comme le montre la table, pour être sûr d'utiliser les puissances de deux il faudrait utiliser les préfixes possédant la syllabe *bi*; cette pratique est encore loin d'être adoptée.

Pour fixer les idées, en 2005 on estimait que la quantité totale d'information imprimée correspondait à 5 Exa-octets. Et à cette date on n'avait pas construit de dispositif électronique capable d'enregistrer un seul Exa-octet. On a aussi estimé que la somme des connaissances humaines au début de l'an 2000, et comprenant les textes aussi bien que les documents audio et vidéo, équivalait à 12 Exa-octets.

Bus de données

La quantité de données transférée en un cycle mémoire est déterminée par le nombre de lignes du bus de données. Avec un bus de données de 8 lignes (dont la largeur est 8) un cycle de lecture ou d'écriture ne peut transférer qu'un seul octet. Dans ce cas, l'accès à un mot mémoire (2 octets) requiert deux cycles de mémoire. Le processeur MC68000 utilisait un bus de données de 16 bits.

Les bus de données des processeurs MC68020, MC68030 et MC68040 avaient 32 bits ce qui permettait d'accéder simultanément à 4 octets. Quelle est la taille du bus de données de l'ordinateur que vous utilisez régulièrement?

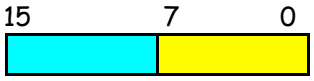
Bus de contrôle
Un ordinateur a besoin de lignes de transmission des signaux pour contrôler les différents évènements se produisant dans les différentes parties du système. Par exemple, une erreur de bus est transmise sur une ligne de signal permettant d'identifier la condition erronée. Lorsque vous démarrez l'ordinateur et que le processus d'initialisation est lancé, un signal « reset » est envoyé à l'UCT. D'autres lignes de signal identifient des demandes de service venant des périphériques (interruptions) et permettent à l'unité centrale d'y répondre.

2.3 Structure et fonctionnement de l'unité centrale de traitement

La figure 2.1 nous a donné le schéma général simplifié d'une unité centrale de traitement. Elle est composée de trois parties: une unité de contrôle et une unité de calcul, dont nous avons déjà parlé, ainsi que d'un ensemble de registres. De façon générale, un registre est une cellule de taille fixe contenue dans l'UCT. L'accès à un registre peut se faire beaucoup plus rapidement qu'un accès à la mémoire centrale. Les registres sont utilisés pour le contrôle de l'ordinateur et pour ranger les résultats intermédiaires des calculs, évitant ainsi des accès à une mémoire relativement lente. Le schéma de l'ordinateur PEP 8, qui se trouve aux pages 148-149 du manuel, est extrêmement simple, PEP 8 étant un processeur assez rudimentaire.

2.3.1 Les registres

Les instructions utilisent généralement l'information qui se trouve dans les registres du processeur. Nous verrons les utilisations qu'on peut faire de ces registres en détail dans les chapitres ultérieurs mais nous pouvons dès maintenant les énumérer. Les processeurs modernes possèdent un certain nombre de registres pouvant avoir des rôles bien définis. Ainsi, on peut avoir 16 ou 32 registres, généralement de 32 bits, pouvant appartenir à deux catégories : registres d'index et registres de calcul. Pour PEP 8, il n'existe que deux registres : le registre X peut être utilisé comme accumulateur ou comme registre d'index, tandis que le registre A ne peut être utilisé que comme accumulateur.



Les 16 bits des registres sont numérotés de 0 à 15 de droite à gauche. Le bit le moins significatif (bit de droite) porte le numéro zéro, tandis que le bit le plus significatif (bit de gauche) porte le numéro 15. Certaines instructions ne manipulent qu'une partie du registre comme un octet qui est alors nécessairement l'octet le moins significatif du registre.

2.3.2 Le compteur ordinal

Le compteur ordinal, souvent repéré par PC (pour « program counter »), est un registre spécial de l'unité centrale de traitement qui contient l'adresse de la cellule de mémoire contenant la prochaine instruction à exécuter. Le compteur ordinal est mis à jour pendant l'exécution de chaque instruction; nous verrons de quelle façon lorsque nous décrirons le processus d'exécution d'une instruction dans une prochaine section. La taille du compteur ordinal va dépendre de la taille de l'espace mémoire utilisé. Dans un processeur actuel (32 bits), elle sera de 32 bits; mais dans un processeur de 64 bits, le compteur ordinal occupera plus de 32 bits. Le compteur ordinal de PEP 8 occupe deux octets, ce qui limite la taille de la mémoire à 2^{16} ou 65536.

2.3.3 Les codes de condition

Les codes de condition sont généralement représentés par un certain nombre de bits qui varie selon le processeur. Dans PEP 8, les codes de condition sont représentés par quatre bits donnés habituellement dans l'ordre suivant et notés N (bit 3), Z (bit 2), V (bit 1) et C (bit 0). L'exécution de certaines instructions donne une valeur à ces codes, souvent en plus des résultats des opérations effectuées. Ainsi par exemple, après une addition, les codes de condition peuvent prendre les valeurs suivantes:

N=1	si le résultat est inférieur à zéro
Z=1	si le résultat est zéro
V=1	s'il y a eu débordement (un débordement se produit lorsque le résultat est plus grand que la valeur maximum que peut contenir l'opérande)
C=1	si une retenue se produit

On peut vérifier la valeur des codes de condition à l'aide des instructions de transfert de contrôle (instructions `BRcc`). Ces instructions permettent de modifier l'ordre d'exécution séquentiel des instructions. L'instruction `BRcc` est équivalente à l'instruction C++:

if(condition) goto adresse;

où condition représente un code de condition. Nous reparlerons des codes de condition dans les chapitres suivants.

2.3.4 Exécution d'un programme

L'UCT ne s'arrête jamais. Elle exécute continuellement des instructions contenues dans la mémoire centrale. Pour exécuter un programme, l'UCT extraira les instructions une à une de la mémoire et réalisera les fonctions qu'elles désignent. Les instructions sont extraites de cellules consécutives de la mémoire jusqu'à ce qu'une instruction de saut soit exécutée. Comme nous l'avons déjà mentionné, l'UCT conserve l'adresse de la *prochaine instruction* à exécuter dans le compteur ordinal. Après l'extraction de chaque instruction, le contenu de ce compteur est ajusté pour désigner la prochaine instruction.

Chaque instruction du processeur PEP 8 occupant une ou trois cellules de mémoire, l'exécution d'une instruction requerra les quatre étapes suivantes:

1. Extraire le contenu de la cellule de mémoire dont l'adresse se trouve dans le compteur ordinal.

2. Décoder l'instruction obtenue en 1, c'est-à-dire en extraire le code d'opération et les caractéristiques des opérandes.

3. Augmenter le contenu du compteur ordinal de 1 ou de 3.

4. Exécuter les opérations indiquées par l'instruction.

Ces quatre étapes sont répétées indéfiniment tant que l'ordinateur est en marche. À la troisième étape, le compteur ordinal doit être ajusté de la longueur de l'instruction courante, car, comme on l'a mentionné, les instructions du processeur PEP 8 ont une longueur variable et peuvent occuper 1 ou 3 cellules de mémoire.

Pour illustrer le fonctionnement de l'unité centrale de traitement nous allons reprendre l'exemple de la section 2.2.1 et en suivre l'exécution pas à pas.

200	21306	306	12
202	32308	308	5
204	16310	310	0

1. obtenir l'instruction à l'adresse 200
2. décoder 21306 « obtenir le contenu de l'adresse 306 »
3. ajuster le compteur ordinal à 202
4. exécuter l'instruction en copiant la valeur 12 dans l'organe arithmétique
5. obtenir l'instruction à l'adresse 202
6. décoder 32308 « ajouter la valeur rangée à l'adresse 308 »
7. ajuster le compteur ordinal à 204
8. exécuter l'instruction qui ajoute 12 à 5
9. obtenir l'instruction à l'adresse 204
10. décoder 16310 « ranger le résultat à l'adresse 310 »
11. ajuster le compteur ordinal à 206
12. exécuter l'instruction en rangeant la valeur 17 à l'adresse 310
13. obtenir l'instruction à l'adresse 206
14. etc.

2.4 Instructions et données

Dans l'exemple que nous venons de voir les instructions se trouvaient aux adresses 200, 202 et 204, et les données aux adresses 306, 308 et 310. La séparation des données et des instructions est cruciale; l'unité centrale passe d'une adresse à l'autre de façon *aveugle*, obtenant des *valeurs numériques* contenues dans les cellules correspondantes et les décodant comme des instructions. L'unité centrale n'a pas moyen de distinguer les instructions des données, c'est au programmeur de séparer les instructions des données et de positionner le compteur ordinal à la première instruction à exécuter. L'unité centrale de traitement exécutera alors les instructions à la suite les unes des autres; *les données ne doivent pas être placées à un endroit où on peut essayer de les exécuter comme des instructions*.

Dans les langages évolués, le programmeur n'a pas ce problème, car le compilateur se charge de ce travail à sa place. En langage d'assemblage le programmeur doit s'en occuper et c'est un point important de la programmation en langage d'assemblage. De la même façon qu'un programme en langage évolué doit être traduit en langage machine par le compilateur avant de pouvoir être exécuté, un programme en langage d'assemblage doit être traduit par un assembleur avant de pouvoir être exécuté. Un assembleur effectue la traduction du langage d'assemblage symbolique au langage machine numérique; cette traduction est bien plus simple que la traduction effectuée par un compilateur.

Le microprocesseur que nous allons étudier est un processeur virtuel, PEP 8. Sur le plan conceptuel le langage d'assemblage ne diffère pas beaucoup d'un processeur à l'autre, à part si l'on passe d'un processeur CISC à un processeur RISC. Notre étude sera centrée sur le microprocesseur PEP 8, mais sera applicable dans sa quasi totalité à tous les autres microprocesseurs CISC ou même RISC.

