Vue d'ensemble de l'ordinateur.

Un ordinateur est une machine conçue pour exécuter au moins un programme formé d'une séquence d'instructions.

Nous décrivons ici l'organisation d'un ordinateur répondant au modèle de *Von Neumann* : c'est un modèle mis au point dans les années 1950, et qui est toujours utile pour comprendre le fonctionnement des ordinateurs.

Plan

Dans le modèle de Von Neumann, l'ordinateur se compose :

- d'une *mémoire centrale*, qui contient le programme et les données ;
- d'une *unité centrale de traitement* (UCT), qui exécute un programme contenu en mémoire centrale ;
- d'une (ou plusieurs) *unité d'entrée-sortie* permettant l'échange d'informations avec l'environnement de l'UCT.

Un *système d'interconnexion* permet l'interaction entre ces unités.

Mémoire centrale

Au niveau physique, la mémoire ne supporte que des bits : les données et les instructions sont stockées sous forme de mots (suite de bits).

C'est une *mémoire à accès aléatoire* (*RAM*, pour *Random Access Memory*) :

- la mémoire est composée de cases, chacune identifiée par une adresse unique. La case mémoire est la plus petite unité d'information adressable.
- chaque case peut être lue/écrite en temps constant, indépendamment des accès précédents.

Chaque case mémoire est identifiée de façon unique par un indice appelé *adresse*. Avec des adresses sur m bits, cela donne 2^m adresses possibles, de 0 à $2^m - 1$.

La mémoire peut donc être vue comme un tableau de 2^m cases mémoires : si `add` est une adresse, on notera `mem[add]` la case mémoire d'adresse `add`.

L'UCT contient une petite quantité de registres : un *registre* est une cellule mémoire, physiquement présente dans l'UCT, permettant de stocker un mot de longueur fixée (typiquement, 8, 16, 32 ou 64 bits).

Les registres stockent temporairement les données et les résultats intermédiaires des calculs, ou des informations de contrôle. Ils ne sont pas référencés par une adresse en mémoire ; il existe un mode d'adressage dédié pour eux.

Les registres peuvent être lus et écrits très rapidement car :

- ils sont physiquement présents dans l'UCT,
- ils sont fabriqués à l'aide de composants « très rapides. »

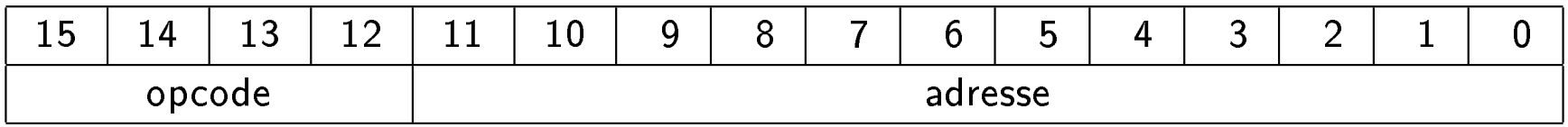
Au contraire, les accès à la mémoire sont beaucoup plus lents : elle est plus éloignée de l'UCT, et fabriquée à partir de composants « plus lents. »

Stockage des programmes en mémoire centrale

Chaque instruction est stockée en mémoire sous la forme d'un ou plusieurs mots binaire. Le code d'une instruction se décompose en :

- un champ appelé *code-opération* ou *opcode*, qui spécifie l'opération qui doit être réalisée lors de l'exécution de l'instruction ;
- éventuellement, un ou plusieurs champs *opérandes* qui définissent les emplacements où l'instruction doit lire ses sources et écrire son résultat.

Ex : sur une certaine machine, les instructions sont codées sur 16 bits :



Exemples d'opcodes hypothétiques :

- 0001 : charger le mot dont l'adresse est donnée dans ACC ;
- 0010 : stocker le mot contenu dans ACC à l'adresse donnée.

Ex : Compilons le petit programme C suivant avec `gcc -O0 -Wa,-alh` :

```
int a, b;
int main(void) {
  int c;
  a = 6; b = -5; c = a + b;
  return(c);
}
```

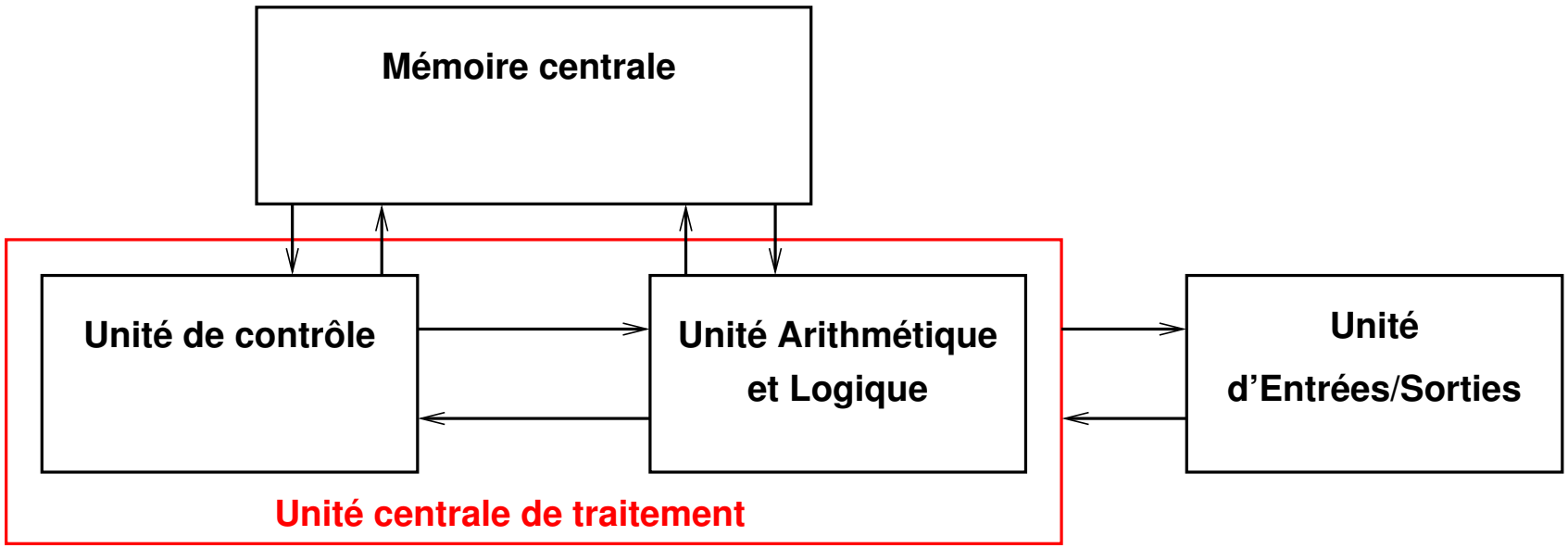
Sur une machine implantant l'architecture x86, on obtient le listing suivant, qui comporte en particulier le codage en langage machine du programme :

```
 1                  .file "test.c"
 2                  .comm a,4,4
 3                  .comm b,4,4
 4                  .text
 5                  .globl main
 6                  .type main, @function
 7                  main:
 8  0000 55         pushl %ebp
 9  0001 89E5       movl  %esp, %ebp
10  0003 83EC10     subl  $16, %esp
11  0006 C7050000   movl  $6, a
11       00000600
11       0000
12  0010 C7050000   movl $-5, b
12       0000FBFF
12       FFFF
13  001a 8B150000   movl a, %edx
13       0000
14  0020 A1000000   movl b, %eax
14       00
...
```

Unité centrale de traitement

L'*UCT* contient (au moins) deux unités fonctionnelles :

- L'*unité de contrôle* (UC), pour gérer l'exécution des instructions ;
- L'*unité arithmétique et logique* (UAL ou ALU), pour effectuer les opérations sur les données.



L'*unité de contrôle* (UC) active dans un ordre précis les circuits nécessaires à l'exécution des instructions d'un programme. Elle utilise deux registres :

- le *registre d'instruction* **IR**, qui contient l'instruction en cours d'exécution ;
- le *compteur de programme* **PC**, qui contient l'adresse en mémoire centrale de la prochaine instruction a exécuter.

L'UC contient un circuit pour le *décodage des instructions*, qui détermine quelle opération doit être exécutée en fonction de l'opcode de l'instruction courante.

L'UC est synchronisée sur une *horloge*, dont la fréquence détermine celle à laquelle sont executées les instructions.

L'*unité arithmétique et logique* (ALU) contient les circuits nécessaires pour exécuter des instructions du langage machine : addition, soustraction, multiplication, division, opérations logiques (or, and, not. . .).

L'UCT doit échanger des données ou des instructions avec la mémoire centrale. Elle utilise pour cela deux registres internes :

- Un *registre d'adresse* **AR**, qui spécifie l'adresse de la mémoire centrale où devra être réalisée la prochaine lecture ou d'écriture. La taille du registre **AR** détermine le nombre maximal de cases mémoires adressables.

- Un *registre de données mémoire* **MDR** est utilisé pour les accès en écriture et les accès en lecture. Le **MDR** peut
  - soit contenir la donnée qui doit être écrite en mémoire centrale,
  - soit recevoir la donnée qui doit être lue en mémoire centrale.

On peut imaginer un mécanisme similaire pour échanger avec les unités d'E/S.

Classes d'instructions et modes d'adressage

On appelle *jeu d'instructions* l'ensemble des instructions disponibles dans le langage machine d'un ordinateur (qu'il peut exécuter).

On distingue essentiellement trois classes d'instructions :

- *Opération arithmétique et logique :* opération sur des données.
  - ↪ addition, soustraction, ou, et, ou-exclusif...
- *Accés mémoire :* transfert entre un registre de l'UCT et la mémoire.
  - ↪ LOAD : chargement du contenu d'une case mémoire dans un registre.
  - ↪ STORE : rangement du contenu d'un registre dans une case mémoire.
- *Contrôle du flot d'instructions :* Le **PC** est incrémenté à chaque cycle d'instruction, mais certaines instructions peuvent modifier sa valeur.
  - ↪ utilisé pour les boucles et les blocs si-alors-sinon ;
  - ↪ instructions de branchement, de saut, ou d'appel à des routines.

Ex : Le LC3 dispose de 8 registres, notés `R0,...,R7`. Dans le langage d'assemblage, l'instruction d'addition se décline de deux façons :

- `ADD DR, SR1, SR2`, qui effectue `DR <- SR1 + SR2`.
  - ↪ Tous les opérandes de l'instruction sont des registres : adressage *par registre*.
  - ↪ Exemple : `ADD R1, R2, R3` effectue `R1 <- R2 + R3`.

- `ADD DR, SR1, imm5`, qui effectue `DR <- SR + imm5`.
  - ↪ Le dernier opérande est un *immédiat*, *i.e.*, il est codé dans l'instruction.
  - ↪ Exemple : `ADD R1, R2, 5` effectue `R1 <- R2 + 5`.

Dans le codage de l'instruction en langage machine, un bit permet de spécifier quel mode d'adressage est utilisé pour le troisième opérande :

assembleur	**action**	**codage**															
		opcode				arguments											
		F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADD DR,SR1,SR2	DR <- SR1 + SR2	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD DR,SR1,Imm5	DR <- SR1 + Imm5	0	0	0	1	DR			SR1			1	Imm5				

Ex (suite) : Le LC3 présente deux instructions d'accès à la mémoire :

- `LD DR, add`, qui effectue `DR <- mem[add]`.
- `ST SR, add`, qui effectue `mem[add] <- SR`.

Dans les deux cas, `add` désigne une adresse mémoire, et `mem[add]` la case mémoire d'adresse `add`. On parle d'*adressage direct* pour le second opérande de ces instructions, car c'est une adresse en mémoire centrale qui est désignée.

Dans le codage de ces instructions en langage machine, l'addresse de chargement ou de rangement en mémoire se retrouve codée[1] dans l'instruction :

assembleur	**action**	**codage**															
		opcode				arguments											
		F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
LD DR,add	DR <- mem[add]	0	0	1	0	DR			add								
ST SR,add	mem[add] <- SR	0	0	1	1	SR			add								

1. Sous une forme particulière, que nous ne détaillerons pas tout de suite...

Ex (suite) : Le jeu d'instruction du LC3 permet d'effectuer un *branchement inconditionnel* de la façon suivante :

- `BR add`, qui effectue `PC <- add`.

Dans ce cas, `add` désigne l'adresse mémoire de l'instruction qui sera exécutée après l'instruction `BR add`. A nouveau, il s'agit d'un *adressage direct* :

assembleur	**action**	**codage**															
		opcode				arguments											
		F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
BR add	PC <- add	0	0	0	0	0	0	0	add								

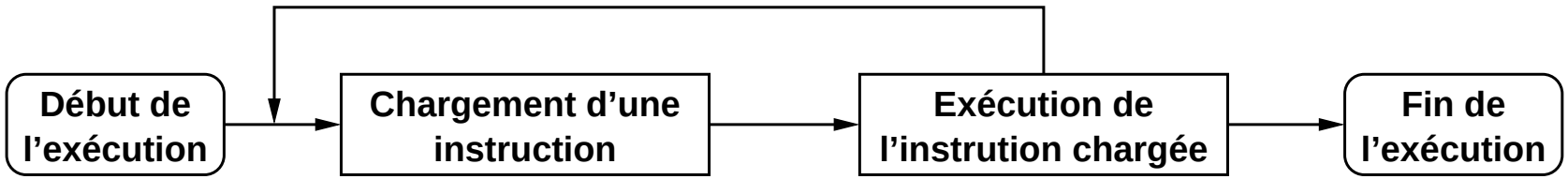
La méthode de localisation des opérandes, dans la mémoire ou parmi les registres, est appelé *mode d'adressage* ; on parle d'adressage :

- *par registre* lorsque l'emplacement désigné est simplement un registre.
- *immédiat* lorsque l'opérande est une valeur codée dans l'instruction.
- *direct* lorsque l'opérande est une adresse en mémoire centrale.

Cycle d'instruction

Lors de l'exécution d'un programme, l'UCT doit effectuer une séquence de tâches pour exécuter chacune des instructions de ce programme. Cette séquence, gérée par l'unité de contrôle, est appelée *cycle d'instruction*.

Chaque tâche du cycle d'instruction occupe un cycle d'horloge du processeur. L'exécution d'un programme est une succession de cycles d'instructions :



Notons que :

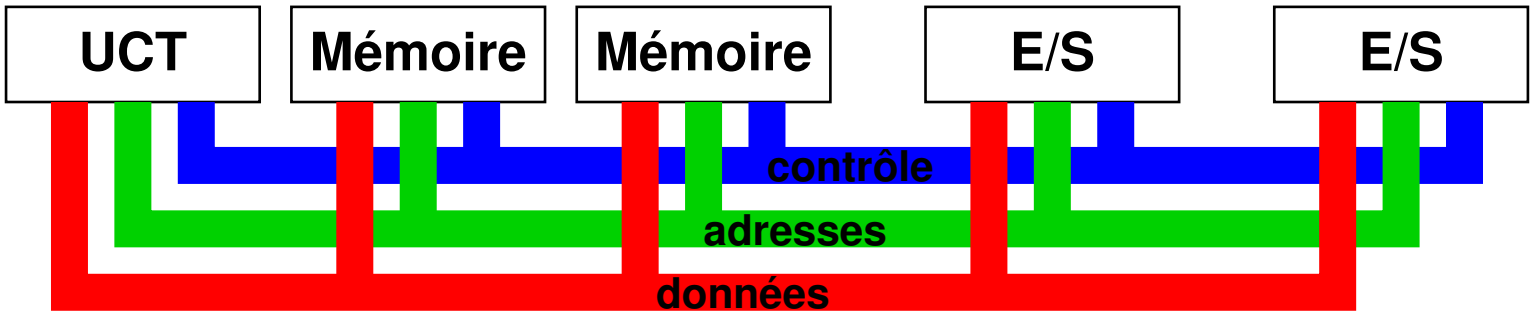
- certaines parties du cycle peuvent varier d'une instruction à l'autre,
- les cycles d'instructions peuvent varier d'un processeur à l'autre.

Organisation des unités autour d'un bus

Les échanges entre les unités de l'ordinateur sont réalisés grâce à des groupes de fils permettant le transfert de mots binaires : si la structure d'interconnexion était réalisée par des liaisons point-à-point, elle deviendrait vite inextricable...

On utilise donc un *bus* : un bus est une nappe de fils paralléles permettant des échanges d'informations cohérents entres plus de deux unités. *Plusieurs unités peuvent s'échanger des informations au travers d'un même bus*, mais une seule unité à la fois peut envoyer des données avec succès sur le bus.

Sur un bus circulent : *des adresses*, *des données*, *des signaux de contrôle*.



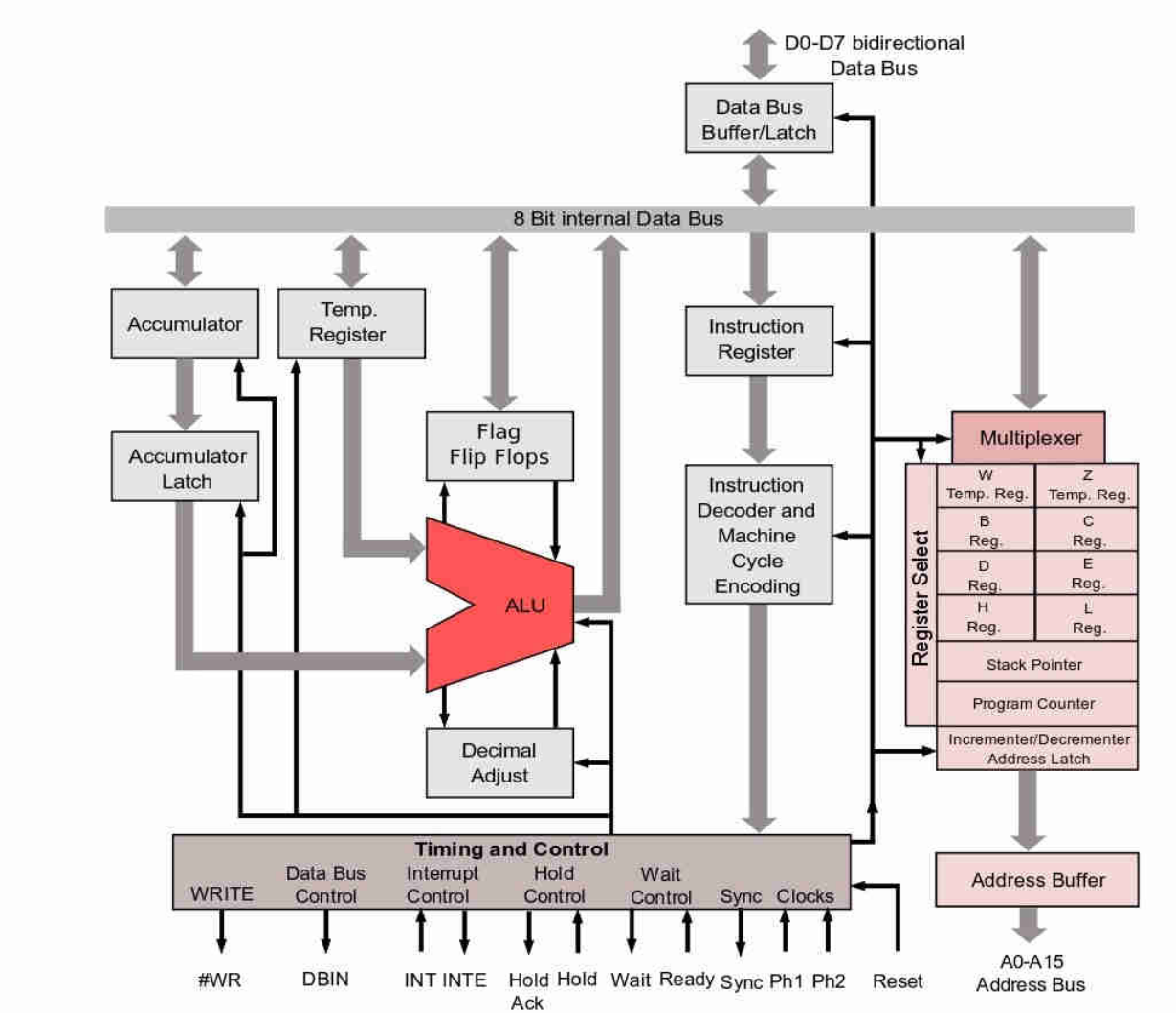
- *Les lignes de données* permettent la transmission de mots binaires.
  Ex : si le bus de données a une largeur de 8 bits, et que les instructions sont codées sur 16 bits, alors il faut deux accès pour charger chaque instruction.

- *Les lignes d'adresses* spécifient les adresses de source ou de destination des données circulant sur le bus de données (ou bien l'adresse d'un port d'E/S).
  Ex : sur un hypothétique bus d'adresse de 8 bits les adresses 00000000_2 à 01111111_2 désignent une adresse de la mémoire centrale, et les adresses 10000000_2 à 11111111_2 désignent l'un des ports d'une unité d'E/S.

- *Les lignes de contrôle* permettent l'implantation d'un protocole d'assurant la fiabilité des communications entre les unités connectés au bus.
  Ex : les signaux de contrôle suivants peuvent être utilisés :
  - *clock* : un signal d'hologe pour synchroniser les échanges ;
  - *bus request* : lorsqu'une unité veut prendre le contrôle du bus ;
  - *bus grant* : indique qu'une unité a le contrôle du bus ;
  - *write memory* : provoque l'écriture des données à l'adresse spécifiée ;

Un exemple : le processeur Intel 8080

Sorti en 1974, le 8080 d'Intel est un processeur 8 bits comportant environ 6000 transistors, et cadencé à 2 MHz. Il est souvent considéré comme le premier microprocesseur véritablement utilisable dans une unité centrale.



Le 8080 est un processeur 8 bits, au sens ou il travaille de manière privilégiée avec des mots de 8 bits : l'UAL manipule des octets, et le bus de données a une largeur de 8 bits. Par contre, le bus d'adresse a une largeur de 16 bits : quelle est la capacité maximale d'adressage du processeur ?

On a huit registres de 8 bits, dont certains peuvent être appariés pour former un registre de 16 bits (BC, DE, HL). Deux registres 16 bits sont directement liés aux adresses mémoire : PC et SP.

Toutes les opérations arithmétiques et logiques sont faites par le biais du registre `Accumulator`, et d'un registre temporaire (non accessible au programmeur).

`Flag` est un registre d'état (*Program Status Register*, PSR) : il indique par exemple si le dernier résultat calculé par l'ALU était positif, négatif ou nul.

D'autre part, le processeur va chercher dans la même mémoire les instructions et les données : il réalise bien une architecture dite de von Neumann.

Conclusion

- Nous avons présenté le *modèle dit de Von Neumann*, qui permet d'aborder le fonctionnement général des ordinateurs.

- Dans les cours suivants, nous verrons les outils nécessaires à l'implantation d'une architecture de Von Neumann simple, celle du LC3 : codage des nombres, circuit logiques combinatoires et séquentiels...

- Nous éludons pour l'instant de nombreuses caractéristiques des processeurs actuels : utilisation des caches, parallélisme d'instructions, de données ou de tâches... Nous en dirons quelques mots vers la fin du cours.