éduSCOL
Informer et accompagner les professionnels de l'éducation

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'ÉDUCATION NATIONALE ET DE LA JEUNESSE

VOIE GÉNÉRALE

2DE | 1RE | TLE

Numérique et Sciences informatiques

ENSEIGNEMENT
SPÉCIALITÉ

MODÈLE D'ARCHITECTURE DE VON NEUMANN

- Histoire de l'informatique
- Représentation des données
- Traitement des données
- Interactions entre l'homme et la machine sur le Web
- Architectures matérielles et systèmes d'exploitation
- Langages et programmation
- Algorithmique

1. Des bribes d'électronique : du transistor au CPU

1.1. Tube à vide, transistor et circuit intégré

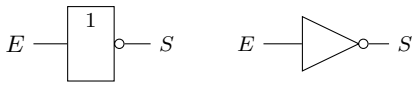
À la base de la plupart des composants d'un ordinateur, on retrouve le transistor. Ce composant électronique a été inventé fin 1947 par les Américains John Bardeen, William Shockley et Walter Brattain. L'invention du transistor a été un immense progrès, mais les premiers ordinateurs sont antérieurs à cette invention. En effet, ces premiers ordinateurs, par exemple le Colossus qui date de 1943, étaient conçus à base de tubes électroniques (on parle aussi de tubes à vide) qui, bien que beaucoup plus gros et beaucoup moins fiables que les transistors, fonctionnent sur le même principe que ce dernier.
Autre aspect historique qu'il est important de préciser : on ne trouve plus, depuis quelque temps déjà, de transistors en tant que composant électronique discret. Dans un ordinateur, les transistors sont regroupés au sein de ce que l'on appelle des circuits intégrés. Dans un circuit intégré, les transistors sont gravés sur des plaques de silicium, les connexions entre les millions de transistors qui composent un circuit intégré sont, elles aussi, gravées directement dans le silicium.

1.2. Circuits logiques

Le transistor est l'élément de base des circuits logiques. Un circuit logique permet de réaliser une opération booléenne. Ces opérations booléennes sont directement liées à l'algèbre de Boole. Un circuit logique prend en entrée un ou des signaux électriques (chaque entrée est dans un état "haut" (symbolisé par un "1") ou à un état "bas" (symbolisé par un "0")) et donne en sortie un ou des signaux électriques (chaque sortie est aussi dans un état "haut" ou à un état "bas"). Il existe deux catégories de circuit logique :

- les circuits combinatoires (les états en sortie dépendent uniquement des états en entrée)
- les circuits séquentiels (les états en sortie dépendent des états en entrée ainsi que du temps et des états antérieurs)

Dans la suite nous nous intéresserons principalement aux circuits combinatoires.
Le plus simple des circuits combinatoires est la porte "NON" qui inverse l'état en entrée : si l'entrée de la porte est dans un état "bas" alors la sortie sera dans un état "haut" et vice versa. Si on symbolise l'état "haut" par un "1" et l'état "bas" pour un "0".



E	S
1	0
0	1

FIGURE 1 – table de vérité de la porte NON

FIGURE 2 – symboles d'une porte NON : à gauche, symbole international ; à droite, symbole étasunien

E_1	E_2	S
0	0	0
0	1	1
1	0	1
1	1	1

FIGURE 3 – table de vérité de la porte OU

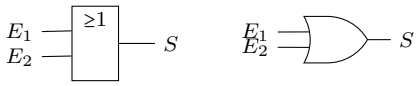


FIGURE 4 – symboles d'une porte OU : à gauche, symbole international ; à droite, symbole étasunien

E_1	E_2	S
0	0	0
0	1	0
1	0	0
1	1	1

FIGURE 5 – table de vérité de la porte ET

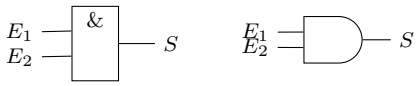


FIGURE 6 – symboles d'une porte ET : à gauche, symbole international ; à droite, symbole étasunien

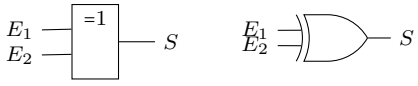


FIGURE 8 – symboles d'une porte XOR : à gauche, symbole international ; à droite, symbole étasunien

En combinant les portes logiques, on obtient des circuits plus complexes. Par exemple en combinant 2 portes "OU EXCLUSIF", 2 portes "ET" et une porte "OU" on obtient un additionneur 2 bits (figure 9).

E_1	E_2	S
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 7 – table de vérité de la porte OU EXCLUSIF (XOR)

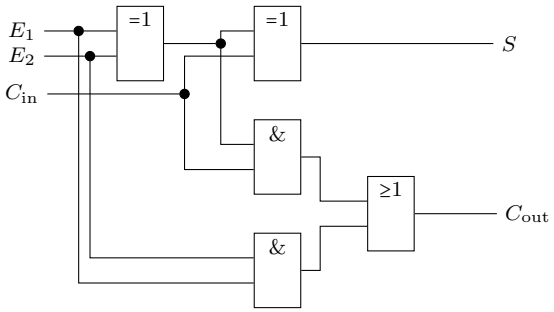


FIGURE 9 – Additionneur 2 bits

En s'aidant des différentes tables de vérité données ci-dessus, Il est possible de demander aux élèves de compléter la table de vérité de l'additionneur 2 bits (voir figure 10).

E_1	E_2	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

FIGURE 10 – table de vérité de l'additionneur 2 bits

1.3. La mémoire vive (RAM)

On peut se représenter la mémoire comme une série de cellules, chaque cellule étant capable de stocker 1 octet. Chacune de ces cellules possède une adresse. Les opérations sur la mémoire sont de 2 types : lecture / écriture. Une opération de lecture consiste à aller lire l'octet situé à l'adresse mémoire XXXXX et une opération d'écriture consiste à écrire un octet donné à l'adresse mémoire YYYYY.

On n'est ainsi pas obligé de passer par une cellule pour accéder à une autre (ce qui formerait une mémoire à accès séquentiel). Cette faculté d'accéder directement à chaque cellule explique le nom RAM : Random access memory, mémoire à accès arbitraire.

On peut réaliser un bit d'une cellule par l'association d'un transistor et d'un condensateur [1]. Un condensateur peut être soit chargé (on stocke alors un "1"), soit déchargé (on stocke alors un "0"). Un condensateur n'est pas capable de conserver sa charge pendant très longtemps, il doit donc être alimenté électriquement afin de conserver cette charge. La mémoire vive est donc une mémoire volatile : toutes les données présentes en mémoire vive sont perdues en cas de coupure de courant.

1. Il existe d'autres technologies, par exemple les circuits de type "bascule".

2. Le microprocesseur (CPU)

2.1. Organisation générale

Les instructions qui composent les programmes sont exécutées par le CPU (Central processing unit). Il est schématiquement constitué de 3 parties.

- L'unité arithmétique et logique (UAL ou ALU en anglais) est chargée de l'exécution de tous les calculs que peut réaliser le microprocesseur. Nous allons retrouver dans cette UAL des circuits comme l'additionneur.
- L'unité de commande permet d'exécuter les instructions (les programmes).
- Une toute petite quantité de mémoire appelée les registres, qui permettent de mémoriser de l'information transitoirement pour opérer dessus ou avec. Leur nombre et leur taille et leur rôle sont variables en fonction du type de microprocesseur. Certains registres jouent des rôles particuliers dans le fonctionnement du processeur et portent des noms en conséquence. D'autres registres ont un usage plus général ; dans la suite on nommera ces registres R1, R2, R3…

Les données doivent circuler entre les différentes parties d'un ordinateur, notamment entre la mémoire vive et le CPU. Le système permettant cette circulation est appelé bus. Il existe, sans entrer dans les détails, 3 grands types de bus :

- Le bus d'adresse permet de faire circuler des adresses (par exemple l'adresse d'une donnée à aller chercher en mémoire).
- Le bus de données permet de faire circuler des données.
- Le bus de contrôle permet de spécifier le type d'action (exemples : écriture d'une donnée en mémoire, lecture d'une donnée en mémoire).

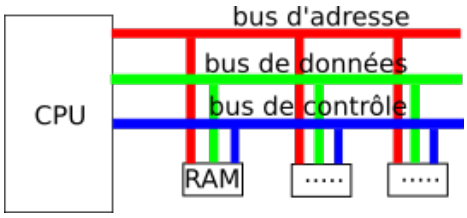


FIGURE 11 – bus d'adresse, bus de données et bus de contrôle

2.2. Fonctionnement

Un processeur donné est capable d'exécuter un certain nombre d'opérations de base, celles pour lesquelles il dispose d'un circuit électronique qui les réalise.
L'ensemble des instructions exécutables directement par le microprocesseur (instructions machines) constitue ce que l'on appelle le "langage machine" du processeur.
Chaque instruction machine correspond à une configuration électronique binaire composée principalement de 2 parties.

- Le champ "code opération" (opcode) qui indique au processeur le type de traitement à réaliser. Par exemple, sur un certain modèle de processeur, le code "00100110" donne l'ordre d'effectuer une multiplication.
- Le champ "opérandes" indique la nature des données sur lesquelles l'opération désignée par le "code opération" doit être effectuée.

Un opérande peut être de 3 natures différentes :

- l'opérande est une valeur immédiate : l'opération est effectuée directement sur la valeur donnée dans l'opérande ;
- l'opérande est un registre du CPU : l'opération est effectuée sur la valeur située dans un des registres (R0,R1, R2,...) ; l'opérande indique de quel registre il s'agit ;
- l'opérande est une donnée située en mémoire vive : l'opération est effectuée sur la valeur située en mémoire vive à l'adresse XXXXX. Cette adresse est indiquée dans l'opérande.

Le programme exécuté se trouve en RAM, tout comme les données. Un registre particulier du processeur, nommé IP (instruction pointer), contient l'adresse de la cellule RAM de la prochaine instruction à exécuter.
Un deuxième registre, IR (instruction register), joue un rôle important en raison de sa connexion physique au reste du processeur : placer dans ce registre la configuration électronique qui dénote une instruction provoque l'activation du circuit dédié à la réalisation de l'opération sous-jacente.
Le CPU a un fonctionnement cyclique :

- il copie dans le registre IR le contenu de la RAM à l'adresse pointée par IP ;
- il décode l'instruction contenue dans IR : ceci provoque l'activation du circuit électronique qui réalise l'opération visée ;
- il exécute l'instruction décodée ; ceci met aussi à jour la valeur de IP pour continuer dans le programme.

La capacité du processeur à exécuter tous les programmes s'explique par ce fonctionnement très souple et par le fait que le jeu d'instructions de base est suffisamment riche pour être universel (on dit que le langage machine est Turing-complet).
Dans le modèle de von Neumann, il y a une seule mémoire vive pour le programme et les données : c'est par sa copie dans le registre IR qu'une configuration électronique initialement présente en RAM joue le rôle d'une instruction. La même configuration pourrait être interprétée comme une donnée (entier, etc.) dans un autre contexte.
La réciproque vaut aussi : une grande famille d'attaques informatiques consiste à exploiter des défauts dans le flux de contrôle d'un programme pour faire exécuter par le processeur des configurations électroniques qui étaient censées être des données, notamment des données choisies par l'utilisateur-attaquant. Pour ces raisons de sécurité, les processeurs modernes sont dotés de la capacité de marquer des portions de la mémoire comme non-exécutables, s'écartant ainsi du principe d'origine du modèle de von Neumann.

2.3. Exemples d'instructions CPU

Historiquement, les instructions machine sont relativement basiques ; on peut se contenter d'à peine plus que ce qui est nécessaire pour l'universalité. Pour des raisons de performance, les processeurs modernes savent exécuter nativement des opérations plus complexes, comme la composée addition-produit ou des opérations sur des vecteurs de petite dimension.
Les opérations fondamentales sont de trois sortes.
Les instructions arithmétiques (addition, soustraction, multiplication...) effectuent des calculs mathématiques, soit sur des entiers, soit sur des flottants. Par exemple, on peut avoir une instruction consistant à additionner la valeur contenue dans le registre R1 et le nombre 789 et ranger le résultat dans le registre R0.
Les instructions de transfert de données permettent de transférer une donnée d'un registre du CPU vers la mémoire vive et vice versa. Par exemple, on peut avoir une instruction consistant à prendre la valeur située à l'adresse mémoire 487 et la placer dans la registre R2, ou encore prendre la valeur située dans le registre R1 et la placer à l'adresse mémoire 512.
Sont également essentielles les instructions de rupture de séquence ou instructions de saut. Elles permettent d'agir sur le registre IP qui contient l'adresse de la prochaine instruction à exécuter. Les instructions arithmétiques ou de transfert de données, outre leur effet spécifique, incrémentent la valeur de IP : de cette façon, l'exécution du programme avance séquentiellement. Les instructions de saut permettent de réaliser les autres structures de contrôle d'exécution, notamment l'alternative et la boucle.
On distingue les instructions de saut inconditionnel, qui modifient toujours IP à la valeur donnée en opérande, et les instructions de saut conditionnel, qui ne font cette modification que selon certaines circonstances, et sinon ont le comportement habituel d'incrémentation de IP.
Une instruction de saut conditionnel permet par exemple d'agir comme suit : si la valeur contenue dans le registre R1 est strictement supérieure à 0 alors la prochaine instruction à exécutér est celle située à l'adresse mémoire 4521.

3. Initiation à l'assembleur

Programmer en langage machine est extrêmement difficile (très longue suite de 0 et de 1), pour pallier cette difficulté, les informaticiens ont remplacé les codes binaires abscons par des symboles mnémoniques (plus facile à retenir qu'une suite de "1" et de "0"), cela donne l'assembleur. Par exemple un "ADD R1,R2,#125" sera équivalent à "11100001010000010000010000011111101".
Le processeur est uniquement capable d'interpréter le langage machine, un programme appelé "assembleur" assure donc le passage de "ADD R1,R2,#125" à "11100001010000010000010000011111101". Par extension, on dit que l'on programme en assembleur quand on écrit des programmes avec ces symboles mnémoniques à la place de suite de "0" et de "1".
Il n'est pas question d'apprendre aux élèves à programmer en assembleur, mais voici tout de même quelques exemples d'instructions qui peuvent être donnés aux élèves :

```
LDR R1,78
```

Place la valeur stockée à l'adresse mémoire 78 dans le registre R1 (par souci de simplification, nous continuons à utiliser des adresses mémoire codées en base 10)

```
STR R3,125
```

Place la valeur stockée dans le registre R3 en mémoire vive à l'adresse 125

```
ADD R1,R0,#128
```

Additionne le nombre 128 (une valeur immédiate est identifiée grâce au symbole #) et la valeur stockée dans le registre R0, place le résultat dans le registre R1

```
ADD R0,R1,R2
```

Additionne la valeur stockée dans le registre R1 et la valeur stockée dans le registre R2, place le résultat dans le registre R0

```
SUB R1,R0,#128
```

Soustrait le nombre 128 de la valeur stockée dans le registre R0, place le résultat dans le registre R1

```
SUB R0,R1,R2
```

Soustrait la valeur stockée dans le registre R2 de la valeur stockée dans le registre R1, place le résultat dans le registre R0

```
MOV R1, #23
```

Place le nombre 23 dans le registre R1

```
MOV R0, R3
```

Place la valeur stockée dans le registre R3 dans le registre R0

```
B 45
```

Nous avons une structure de rupture de séquence, la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 45

```
CMP R0, #23
```

Compare la valeur stockée dans le registre R0 et le nombre 23. Cette instruction CMP doit précéder une instruction de branchement conditionnel BEQ, BNE, BGT, BLT (voir ci-dessous)

```
CMP R0, R1
```

Compare la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1.

```
CMP R0, #23
BEQ 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est égale à 23

```
CMP R0, #23
BNE 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 n'est pas égale à 23

```
CMP R0, #23
BGT 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est plus grand que 23

```
CMP R0, #23
BLT 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est plus petit que 23

```
HALT
```

Arrête l'exécution du programme

À ce stade, il est possible de demander aux élèves d'expliquer quelques instructions, par exemple

```
ADD R0, R1, #42
```

ou encore

```
CMP R4, #18
BGT 77
```

Il est aussi possible de demander aux élèves d'écrire des instructions en assembleur :

- ▷ Additionne la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1, le résultat est stocké dans le registre R5.
- ▷ la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 478. Si la valeur stockée dans le registre R0 est égale 42 alors la prochaine instruction à exécuter se situe à l'adresse mémoire 85.

Les instructions assembleur B, BEQ, BNE, BGT et BLT n'utilisent pas directement l'adresse mémoire de la prochaine instruction à exécuter, mais des "labels". Un label correspond à une adresse en mémoire vive. L'utilisation d'un label évite donc d'avoir à manipuler des adresses mémoires en binaire ou en hexadécimale :

```
CMP R4, #18
   BGT monLabel
   MOV R0,#14
   HALT
monLabel:
   MOV R0,#18
   HALT
```

Il est possible de demander aux élèves de comparer un programme écrit en Python

```
x = 4
y = 8
if x == 10:
  y = 9
else :
  x=x+1
z=6
```

et le même programme écrit en assembleur

```
   MOV R0, #4
   STR R0,30
   MOV R0, #8
   STR R0,75
   LDR R0,30
   CMP R0, #10
   BNE else
   MOV R0, #9
   STR R0,75
   B endif
else:
   LDR R0,30
   ADD R0, R0, #1
   STR R0,30
endif:
```

```
16  MOV R0, #6
17  STR R0,23
18  HALT
```

On peut alors demander aux élèves à quoi correspond les adresses mémoires 23, 75 et 30. Pour en savoir plus sur l'architecture de Von Neuman, il est possible de consulter le livre de Claude Timsit [1].

4. Simulateur

Afin de mettre en pratique ce qui vient d'être étudié, on peut faire travailler les élèves sur le simulateur développé par Peter L Higginson [2].

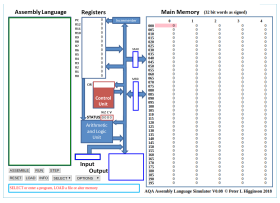


FIGURE 12 – Simulateur de Peter L Higginson

On utilise ce simulateur en le programmant en assembleur (le simulateur accepte l'assembleur étudié précédemment).

Références

[1] Claude Timsit. *Du transistor à l'ordinateur*. 2010.

[2] Peter L Higginson. Simulateur. http://www.peterhigginson.co.uk/AQA/.