Architecture de von Neumann

Table des matières

Prérequis : l'activité sur les booléens.
Objectifs :

- Distinguer les rôles et les caractéristiques des différents constituants d'un processeur.
- Dérouler l'exécution d'une séquence d'instructions simples d'assembleur.

Outil nécessaire : aucun (activité débranchée).

I) Introduction

1. Présentation de l'activité

Dans l'activité antérieure sur les booléens nous avons exploré un peu d'algèbre booléenne et de logique combinatoire. Nous allons maintenant étudier quelques aspects matériels de base de l'ordinateur qui mettent en application cette théorie.

2. Un peu d'histoire

1936
Alan Mathison Turing (1912-1954) publie *On Computable Numbers with an Application to the Entscheidungsproblem*, ouvrage qui définit les limites théoriques de l'ordinateur. Il présente le modèle des machines de Turing et construit mathématiquement la première machine universelle.

Cette activité est inspirée du livre « Informatique et sciences du numérique » de Gilles Dowek (Editions Eyrolles), libre d'accès en ligne et des sites suivants :

- https://pixees.fr/informatiquelycee/ ;
- http://isnalti.free.fr/ressources/cours/cours_langage.pdf ;
- https://www.apprendre-en-ligne.net.

1944

John von Neumann (né János Neumann, 1903-1957) a donné son nom à « l'architecture de von Neumann » utilisée dans la quasi-totalité des ordinateurs modernes. Cela est dû au fait qu'il était, en 1944, le rapporteur des travaux pionniers en la matière (*First Draft of a Report on the EDVAC 1*). Le modèle de calculateur à programme auquel son nom reste attaché, et qu'il attribuait lui-même à Alan Turing, possède une unique mémoire qui sert à conserver les logiciels et les données. Ce modèle, extrêmement innovant pour l'époque, est à la base de la conception d'une très grande majorité d'ordinateurs.

1958

Alors qu'il travaillait pour Texas Instrument, l'Américain Jack Kilby (1923-2005) invente le premier circuit intégré, jetant ainsi les bases du matériel informatique moderne. Cette découverte a valu à Kilby le prix Nobel de physique en 2000.

Exercice 1

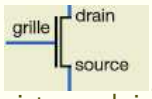
En révisant l'introduction de l'activité sur les booléens si nécessaire, situer chronologiquement l'invention du transistor par rapport aux évènements décrits ci-dessus.

II) Architecture séquentielle de von Neumann

1. Les portes logiques

L'activité sur les booléens a permis de découvrir la logique combinatoire et notamment des opérateurs logiques comme le ET (ou AND), le OU (ou OR), le OU EXCLUSIF (ou XOR) etc, mais aussi la fonction de multiplexeur et d'additionneur. Ces fonctions (et quelques autres) sont à la base du fonctionnement d'un ordinateur et comme il y est fait allusion dans cette activité sur les booléens, le **transistor** est le composant électronique qui a permis le développement en grand nombre dans un petit volume de ces fonctionnalités sur des circuits intégrés.
Nous n'allons pas étudier en détail le transistor mais on peut retenir les points suivants.

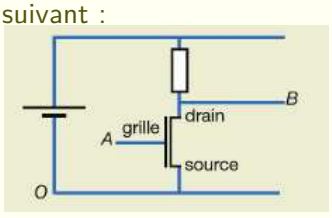
- Le transistor est un composant électronique à trois broches représenté ci-contre :



- Si on applique entre la grille et la source une tension inférieure au seuil de basculement du transistor, celui-ci est bloqué. Si, en revanche, cette tension est supérieure au seuil de basculement du transistor, celui-ci est passant.

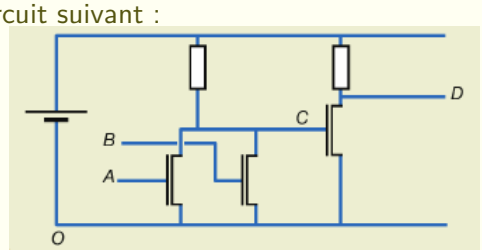
Sans entrer dans les détails, il faut savoir que ce fonctionnement permet de commander des niveaux électriques de sorties en fonction de niveaux électriques d'entrées et notamment d'obtenir les fonctions logiques étudiées dans l'activité sur les booléens.
Par exemple,

- la fonction NON s'obtient avec le circuit suivant :



;

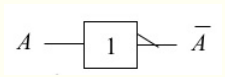
- la fonction OU s'obtient avec le circuit suivant :



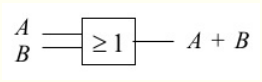
.

De tels circuits, que l'on appelle **portes logiques**, sont lourds à représenter aussi on leur substitue les schémas suivants :

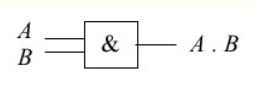
- selon la norme internationale **CEI**[2] aujourd'hui en vigueur :



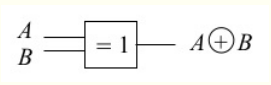
Porte NON



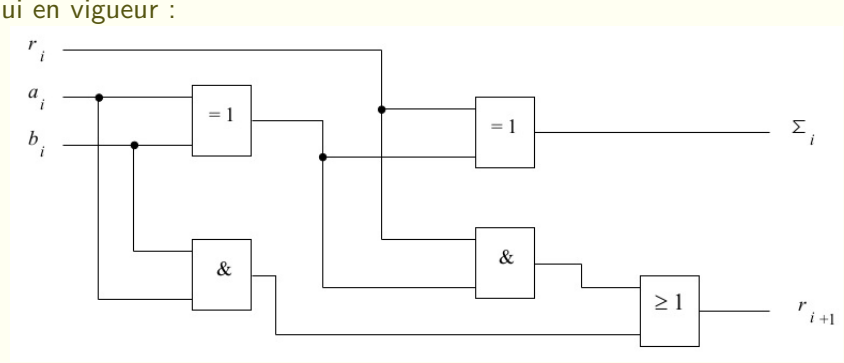
Porte OU



Porte ET



Porte OU EXCLUSIF

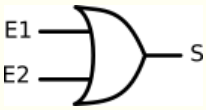


Additionneur[a]

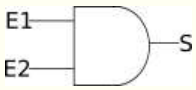
a. L'additionneur : voir l'activité sur les booléens.

- selon la norme étatsunienne *ANSI*[3] encore souvent utilisée en France :

Porte NON

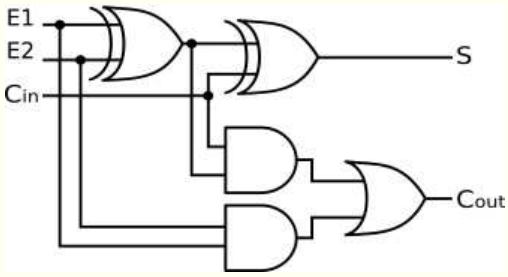


Porte OU



Porte ET

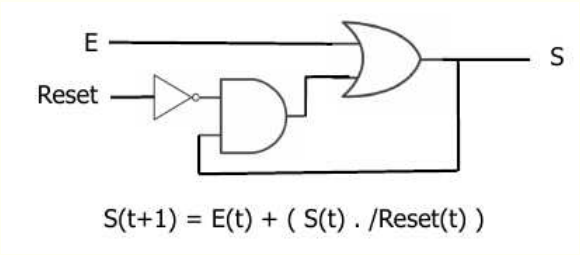
Porte OU EXCLUSIF



Additionneur

2. Circuit séquentiel

Dans tous les circuits que nous avons vus jusqu'à présent (des circuits combinatoires), les informations circulent de la gauche vers la droite mais on peut aussi créer une boucle de retour d'une sortie vers une entrée et ainsi prendre en compte la notion de temps : on parle alors de circuits séquentiels.
Sans entrer dans les détails, retenons que parmi les circuits séquentiels on a les circuits mémoires qui permettent de conserver des données dans le temps et les horloges qui cadencent le fonctionnement de l'ordinateur.
Par exemple voici un circuit mémoire :



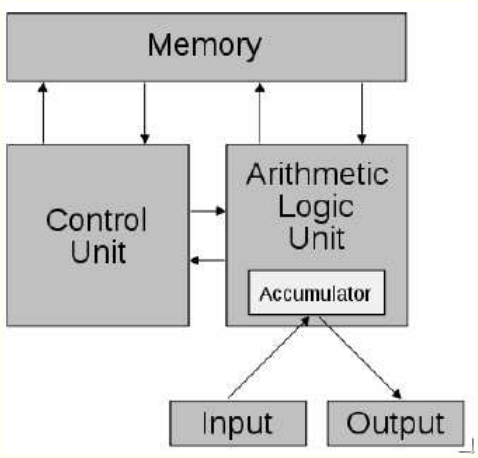
2. La Commission Électrotechnique Internationale (CEI) ou *International Electrotechnical Commission (IEC)* en anglais, est l'organisation internationale de normalisation chargée des domaines de l'électricité, de l'électronique, de la compatibilité électromagnétique, de la nanotechnologie et des techniques connexes. Elle est complémentaire de l'organisation internationale de normalisation (*ISO*), qui est chargée des autres domaines.
La CEI est composée de représentants de différents organismes de normalisation nationaux, elle a été créée en 1906 et compte actuellement 69 pays participants. Les normes CEI sont reconnues dans plus de 100 pays.

3. *ANSI : American National Standards Institute.*

3. Architecture de von Neumann

L'architecture de von Neumann décompose l'ordinateur en 4 parties distinctes :

- l'unité arithmétique et logique (UAL) ou unité de traitement, qui effectue les opérations de base ;
- l'unité de contrôle, qui est chargée du séquençage des opérations ;
- la mémoire, qui contient à la fois les données et le programme qui indique à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise en mémoire vive (programmes et données en cours de fonctionnement) et mémoire de masse (programmes et données de base de la machine) ;
- les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.



III) Communiquer avec l'ordinateur

La programmation consiste à donner à un ordinateur les consignes qui lui permettront de réaliser une succession de tâches bien définies.
En pratique on définit ces tâches dans un langage naturel (le français), pour réaliser un algorithme.
Une fois l'algorithme écrit, on doit informer l'ordinateur, par l'intermédiaire d'un langage compris de lui, de la liste des opérations qu'il va devoir réaliser.

1. Le langage machine

Le microprocesseur (μP) d'un ordinateur ne connait qu'un seul langage : le langage machine. C'est comme sa langue maternelle.
Le langage machine est constitué d'instructions écrites en binaire (une succession 0 et de 1) ou en hexadécimal (symboles de 0 à F).
Pour un être humain normalement constitué, ce langage est plutôt incompréhensible :

```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa
00000200
```

Le fait d'être incompréhensible comporte un avantage : un programme écrit en langage machine est très difficile à modifier sinon par son auteur !
Chaque microprocesseur a son propre langage machine. En effet le codage des instructions n'est pas le même pour tous les microprocesseurs. En général le langage machine compris par une série de µP (x86 par exemple) est compris par la série suivante (x87). L'inverse n'est pas vrai (http://fr.wikipedia.org/wiki/Jeu_d'instructions_x86).

2. Les langages plus compréhensibles

Plutôt que des suites de nombres écrits en base 2 ou 16, on peut essayer de "parler" avec un ordinateur par l'intermédiaire de mots clefs proches du langage courant (souvent l'anglais).

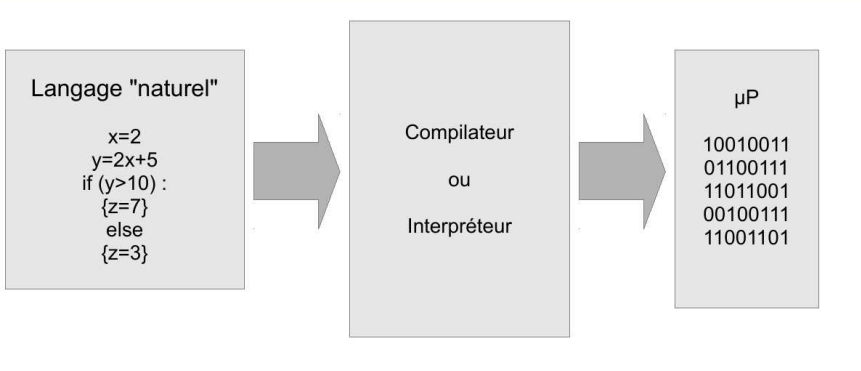
Les langages informatiques qui utilisent des mots clefs sans se soucier du fonctionnement interne de l'ordinateur s'appellent des langages de haut niveau.
Les langages qui prennent en compte le fonctionnement interne du processeur (registres, piles) sont appelés langages de bas niveau.
Pour nous il sera facile de nous exprimer en langage de haut niveau.

Problème : les processeurs comprennent uniquement le langage machine !
Il va donc falloir utiliser une sorte d'interprète entre notre langage et le langage machine. C'est la façon dont l'interprète agit qui va définir le type de langage :

- avec les langages compilés, le programme est traduit une fois, puis envoyé au µP qui s'en sert tel quel (C, C++...) ;
- avec les langages interprétés, le programme est traduit au fur et à mesure (et autant de fois) qu'il est exécuté par le µP (PHP, Python...).

Avantages et inconvénients
Un programme écrit dans un langage compilé a comme avantage de ne plus avoir besoin, une fois compilé, de programme annexe pour s'exécuter. De plus, la traduction étant faite une fois pour toute, il est plus rapide à l'exécution.
Toutefois il est moins souple qu'un programme écrit avec un langage interprété car à chaque modification du fichier source (fichier intelligible par l'homme : celui qui va être compilé) il faudra recompiler le programme pour que les modifications prennent effet.
À noter que la compilation (traduction) valable pour une famille de µP ne l'est pas forcément pour une autre. Lors de la compilation on doit savoir à quelle famille on va s'adresser !

Bilan



IV) Codage

On l'a dit, le langage machine est un code traité par le microprocesseur qui sait interpréter un jeu d'instructions.

1. Mise en situation

Pour y voir un peu plus clair, on va imaginer un microprocesseur MP0 constitué de registres 8 bits. Il ne manipule que des octets et accède à des adresses comprises entre 0 et 11111111 (ou 255 ou FF).
On veut que notre µP réalise l'algorithme suivant :

```
x = 3
z = 0
donner à y la valeur 2.x + 1
si y est supérieur à 10 donner à z la valeur 5
```

Vocabulaire

- Une adresse est un nombre entier représentant un endroit précis d'une partie de la mémoire.
- Un registre est une zone mémoire où l'on peut stocker une valeur.
- Un pointeur est une zone mémoire contenant comme valeur une adresse.
- Une pile (*stack* en anglais) est une zone mémoire dans laquelle on peut stocker temporairement des données. En fait c'est un ensemble constitué d'une zone de mémoire plus un pointeur permettant de repérer le sommet

de la pile pour aller chercher la dernière valeur empilée : les piles sont de type *Last In First Out* (LIFO) comme des piles d'assiettes.

Dans le cas de notre MP0, nous avons les éléments suivants.

- Un compteur ordinal (CO). Il contient l'adresse de la prochaine instruction à exécuter.
- Un registre d'état (SR). Il contient les bits Z et N. On a Z = 1 si la dernière instruction a produit un résultat nul (sinon Z = 0) et N = 1 si la dernière instruction a produit un résultat < 0 (sinon N = 0).
- Deux registres données (D0 et D1). Ils stockent chacun un octet à traiter.
- Deux registres d'adresses (A0 et A1). Ils stockent chacun une adresse de la RAM.
- Un pointeur de pile (PP). C'est un registre d'adresse réservé pour la pile.

2. L'assembleur

En pratique chaque instruction est codée sur deux octets :

- le premier correspond au type d'opération (faire la somme, stocker,etc) ;
- le second correspond à la valeur utilisée dans l'opération (adresse, donnée, valeur).

Exemples

- Mettre 28 dans D1 sera codé 121 28 (le µP sait que 121 veut dire "mettre dans D1").
- Mettre D1 dans D0 sera codé 248 0 (248 veut dire "mettre D1 dans D0" et il n'y a pas de valeur).
  Ces deux expressions donneront :
  - en binaire : 01111001 00011100 11111000 00000000 ;
  - en hexadécimal : 79 1C F8 00.

Connaitre les codes binaires de chaque instruction serait un vrai calvaire. Pour un peu simplifier la programmation en langage machine on remplace les codes des instructions par des noms (MOVE, ADD, SUB...).
On dit alors qu'on programme en **Assembleur**.
Voici quelques actions que peut réaliser notre MP0 avec son langage LM0 :

- Adressage
  MOVE #3, D0 Met la valeur "3" dans le registre D0
  MOVE 100, D0 Met la valeur située à l'adresse 100 dans le registre D0
  MOVE (A1), D0 Met la valeur dont l'adresse est A1 dans le registre D0

  Remarque : MOVE #10, 100 n'est pas valide. Pour mettre la valeur 10 à l'adresse 100 l'instruction est :
  MOVE #10,D0
  MOVE D0, 100
- Opérations
  ADD D0, D1 met le résultat de D0 + D1 dans D1
  SUB D0, D1 met le résultat de D1 - D0 dans D1
  MUL D0, D1 met le résultat de D0 x D1 dans D1
  DIV D0, D1 met le quotient de D1 / D0 dans D1
- Sauts
  JMP #100 met la valeur 100 dans CO (adresse de la prochaine instruction)
  JMP 10 met la valeur située à l'adresse 10 dans CO
  JMP A0 met la valeur dont l'adresse est A0 dans CO
- Sauts conditionnels
  Une instruction de saut conditionnel se place toujours après une instruction de comparaison CMP :
  CMP D0, D1
  Cette instruction agit comme SUB mais ne modifie pas D1 et modifie les registres de condition Z et N comme suit.

Instructions	Conditions du saut	Effets
JEQ	(si D0 = D1)	Z = 1
JNE	(si !=)	Z = 0
JLT	(si D0 < D1)	N = 0
JLE	(si <=)	N = 0 ou Z = 1
JGT	(si >)	N = 1
JGE	(si >=)	N = 1 ou Z = 1

  Exemples

① MOVE #5, D0 met la valeur 5 dans D0
MOVE #2, D1 met la valeur 2 dans D1
CMP D0, D1 calcule 2 - 5. Comme le résultat est < 0 alors Z $= 0$ et N $= 1$

② MOVE #1, D0 met la valeur 1 dans D0
MOVE #2, D1 met la valeur 2 dans D1
CMP D0, D1 calcule 2 - 1. Comme le résultat est > 0 alors Z $= 0$ et N $= 0$

③ MOVE #2, D0 met la valeur 2 dans D0
MOVE #2, D1 met la valeur 2 dans D1
CMP D0, D1 calcule 2 - 2. Comme le résultat est $= 0$ alors Z $= 1$ et N $= 0$

④ On met dans D0 la valeur 10 et dans D1 la valeur 15. Comme D0 $<$ D1 on veut mettre l'adresse 150 dans CO :

```
MOVE #10, D0   met la valeur 10 dans D0
MOVE #15, D1   met la valeur 15 dans D1
CMP D0, D1     15-10 > 0 donc Z = 0 et N = 0
JLT #150       comme N = 0 alors je mets l'adresse 150 dans CO !
```

- Saut vers un sous programme

```
JSR #150   saute à l'adresse 150
RTS        fin du sous programme et retour
```

Exercices

2

① Décrire ce que fait chacun des programmes suivants :

```
MOVE 10, D0
MUL #5, D0
MOVE D0, 11
```

```
MOVE #70, D0
MOVE #30, D1
ADD D0, D1
```

```
MOVE 10, D0
MOVE 11, D1
ADD D0,D1
MOVE D1, 15
```

② À l'adresse 10 se trouve x.

À l'adresse 11 se trouve y.

À l'issue de ce programme, quelle valeur aurons-nous à l'adresse 15 si $x = 5$ et $y = 7$?

Même question si $x = 7$ et $y = 5$.

```
104 MOVE 10, D0
106 MOVE 11, D1
108 CMP D0, D1
110 JLT #116
112 MOVE D0, 15
114 JMP #118
116 MOVE D1,15
118 END
```

Que fait ce programme ?

③ Quelle opération effectue ce programme (x est à l'adresse 10, y à l'adresse 11) ?

```
MOVE 10, D0
MUL #3, D0
ADD #5, D0
MOVE D0, 11
```

④ Décrire ce que fait ce programme (x est une variable mémorisée dans le registre D0).

```
100 : CMP #3, D0
102 : JGE #108
104 : SUB #5, D0
106 : JMP #110
108 : ADD #10, D0
110 : END
```

⑤ Décrire ce que fait ce programme (x est une variable mémorisée dans le registre D0).

```
104 : CMP #10, D0
106 : JGE #112
108 : SUB #1, D0
110 : JMP #104
112 : END
```

⑥ Écrire le programme réalisant l'objectif initial affiché au paragraphe IV)1. page 5 (on mettra x à l'adresse mémoire 10, y à la 11 et z à la 12).