Cryptage d'une image par un programme assembleur.

Thématique :

Architecture

Pré-requis :

- en représentation des données : on supposera connue l'écriture d'un entier positif dans une base $b \geq 2$ notamment les bases 2 et 16.
- en langage et programmation : on supposera connues les constructions éléments (affectation, conditions et boucles).

Objectif :

Cryptage d'une image en assembleur avec l'opérateur xor.

Les contenus du programme :

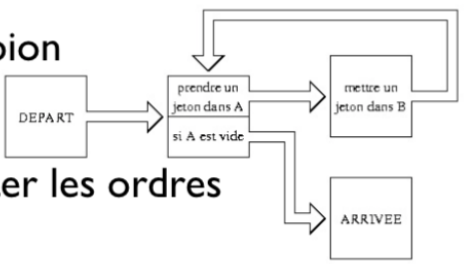
Événements clés de l'histoire de l'informatique	Situer dans le temps les principaux événements de l'histoire de l'informatique et leurs protagonistes.	Ces repères historiques seront construits au fur et à mesure de la présentation des concepts et techniques.
Modèle d'architecture séquentielle (von Neumann)	Distinguer les rôles et les caractéristiques des différents constituants d'une machine. Dérouler l'exécution d'une séquence d'instructions simples du type langage machine.	La présentation se limite aux concepts généraux. On distingue les architectures monoprocesseur et les architectures multiprocesseur. Des activités débranchées sont proposées. Les circuits combinatoires réalisent des fonctions booléennes.

Valeurs booléennes : 0, 1. Opérateurs booléens : and, or, not. Expressions booléennes	Dresser la table d'une expression booléenne.	Le ou exclusif (xor) est évoqué. Quelques applications directes comme l'addition binaire sont présentées. L'attention des élèves est attirée sur le caractère séquentiel de certains opérateurs booléens.

Activité d'introduction

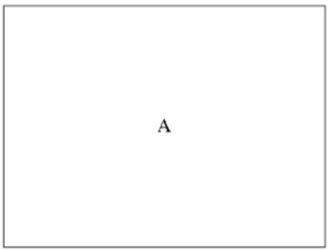
(+ 1 *réserve*)

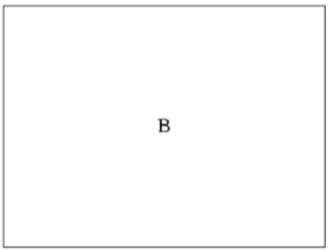
poser le pion ici ☞

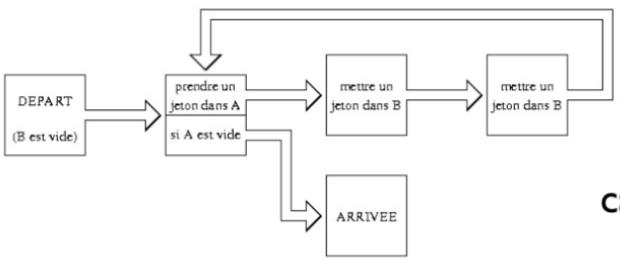


et exécuter les ordres

Quelle opération calcule cette machine ?

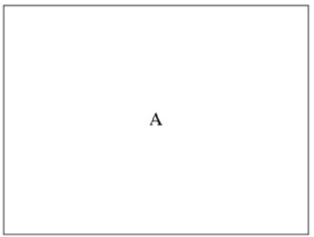


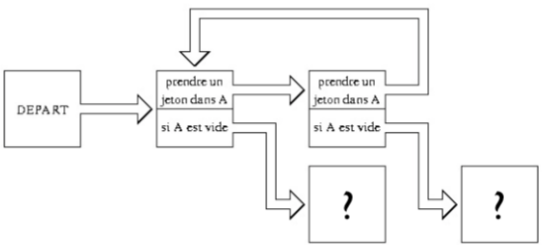




Quelle opération calcule cette machine ?

1 seul registre :





Quelle propriété teste cette machine ?

Le Cours

Il est organisé en trois parties :

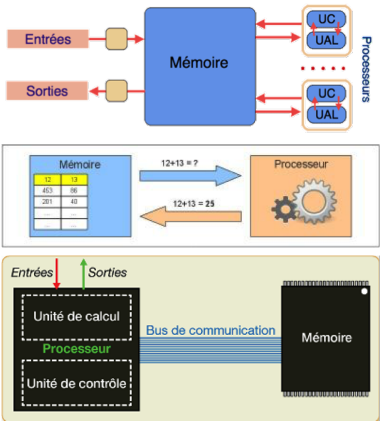
1. L'architecture des ordinateurs.
2. Le langage machine
3. Les portes logiques

1. L'architecture des ordinateurs.

L'ordinateur minimal : Modèle de Von Neumann.

John Von Neumann est à l'origine d'un modèle de machine universelle de traitement programmé de l'information (1946). Cette architecture sert de base à la plupart des systèmes à microprocesseur actuel. Elle est composée des éléments suivants :

- une unité centrale
- une mémoire principale
- des interfaces d'entrées/sorties



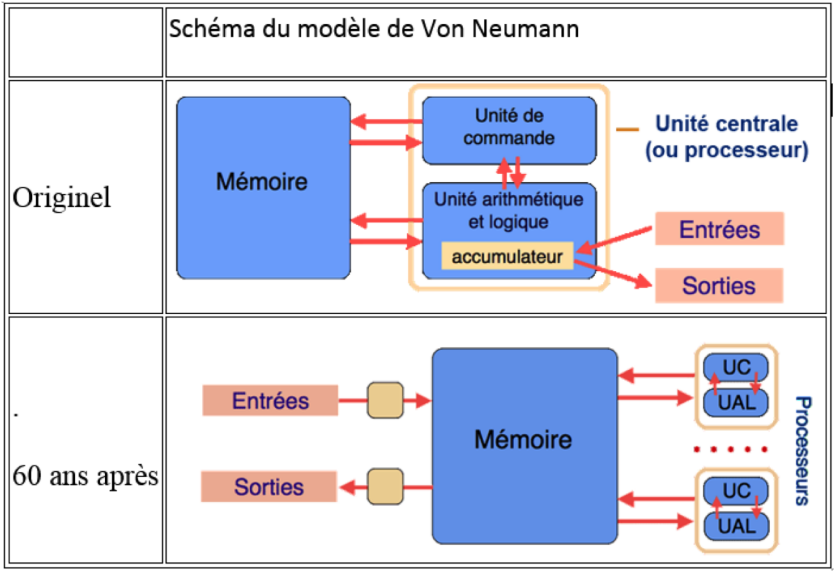


Schéma du modèle de Von Neumann
Originel
Mémoire
Unité de commande
Unité arithmétique et logique
accumulateur
Unité centrale (ou processeur)
Entrées
Sorties
60 ans après
Entrées
Sorties
Mémoire
UC
UAL
UC
UAL
Processeurs

2. Le langage machine

- Un ordinateur doit être capable d'éxécuter un programme. Il faut donc un moyen d'indiquer au processeur la séquence des instructions qu'il doit exécuter.

- Pour échanger des données avec la mémoire, le processeur utilise deux procédés qui permettent l'un de transférer l'état d'un registre dans une case mémoire et l'autre de transférer l'état d'une case mémoire dans un registre.

- Les trois opérations utilisées sont :
  - load/store (Chargement/ Rangement du registre accumulateur avec le contenu de l'adresse correspondante)
  - add/sub/..., and/or, ... (Addition du contenu de l'accumulateur avec le contenu de l'adresse correspondante)
  - Jump/boucle/... (Instruction de branchement, Notion de bloc, de code conditionnel, Boucles ...)

```
1
2   .pos 0
3       mrmovl N, %eax
4       xorl %ecx, %ecx
5   boucle:
6       iaddl 5, %ecx
7       isubl 1, %eax
8       jne boucle
9       halt
10
11  .align 4
12  N:  .long 10
```

3. Les portes booléennes

Au commencement était le transistor, puis nous créâmes les portes booléennes et, à la fin de la journée, les ordinateurs.

Un ordinateur :

- est construit autour d'un ensemble de circuits électroniques (ça passe ou ça ne passe pas) ;
- ces circuits s'appellent des portes booléennes ou parfois portes logiques ;
- il traite donc que des signaux assimilables à 0 ou 1.

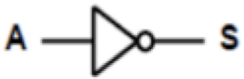
Les opérateurs booléens (George Boole, milieu du XIX$^{\text{ème}}$ siècle) :

- partie des mathématiques qui s'intéresse aux opérations et fonctions sur les variables logiques ;
- utilisés dans la conception des circuits électroniques.

NEGATION

x	NOT x
F	V
V	F

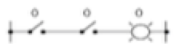
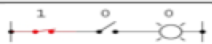
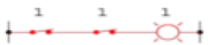
x	¬ x
0	1
1	0



Négation des inégalités	Exemple
NOT (x < y) = x ≥ y	NOT (x < 0) = x ≥ 0
NOT (x ≤ y) = x > y	NOT (x ≤ 1) = x > 1
NOT (x > y) = x ≤ y	NOT (x > -2) = x ≤ -2
NOT (x ≥ y) = x < y	NOT (x ≥ 2012) = x < 1012

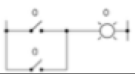
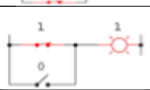
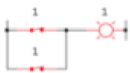
AND (ET) : cette fonction prend le ET logique

x	y	x AND y
F	F	F
F	V	F
V	F	F
V	V	V

x	y	x.y	Dans un ordinateur
0	0	0	
0	1	1	
1	0	0	
1	1	1	

OR (OU) : Cette fonction prend le OU logique

x	y	x OU y
F	F	F
F	V	V
V	F	V
V	V	F

x	y	x+y	Dans un ordinateur
0	0	0	
0	1	1	
1	0	1	
1	1	0	

XOR (OU EXLUSIVE) :

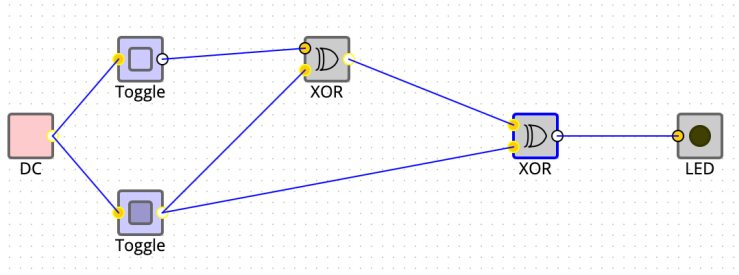
cette fonction prend le ou exclusif, elle n'est vrai que si une seule de ses entrées est vrai

x	Y	x XOR y
F	F	F
F	V	V
V	F	V
V	V	F

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

Exercice

Construire la table de vérité schématisée par le schéma ci-dessous :



Activité Y86

A l'aide du simulateur Y86 (`http://dept-info.labri.fr/ENSEIGNEMENT/archi/js-y86/index.html`), on charge le fichier « cryptage.ys » ci-après :

```
.pos 0

irmovl 8, %ecx
irmovl 0, %edi
boucle:
mrmovl image1(%edi), %eax
mrmovl image2(%edi), %ebx
xorl %eax, %ebx
rmmovl %ebx,image3(%edi)
iaddl 4, %edi
isubl 1, %ecx
jne boucle
halt

.pos 0x0040
image1:
.long 0x00ff0f00
.long 0x00ff0f00
.long 0x0ff00000
.long 0xf0ffff00
.long 0x00f0000f
.long 0x000f0f00
.long 0xf0000f00
.long 0x0f000f00

.pos 0x0070
image2:
.long 0xf00ff0f0
.long 0xf0f0f00f
.long 0x0f0f0f0f
.long 0xff000ff0
.long 0x0f00ff0f
.long 0x00f0f0f0
.long 0xff0ff00f
.long 0x0f0f0ff0

.pos 0x00a0
image3:
```

En 0x0040 (label Image1) l'image de départ à crypter est stockée sous forme d'entiers hexadécimaux d'une longueur de 4 octets (soit 32 bits). Chacun de ces nombres est composé de 8 chiffres hexadécimaux. On a choisi de représenter un pixel noir par « f » (1111 en binaire) et un pixel blanc par « 0 » (0000 en binaire). L'intérêt de ce choix est qu'après une opération XOR, on obtiendra forcément soit « 0 », soit « f ».
(Propriété à faire vérifier aux élèves ?)

En 0x0070 (label Image2), on trouve l'image correspondant à la clef de cryptage : il s'agit d'une image où l'état des pixels a été choisi aléatoirement.

Les deux images sont les suivantes :

Image de départ :

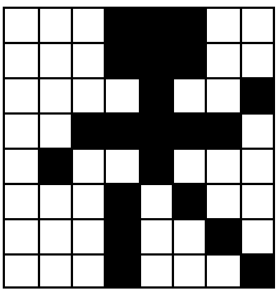
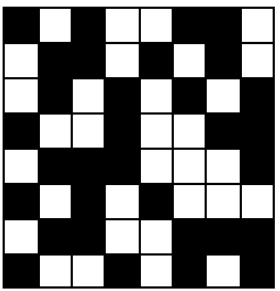
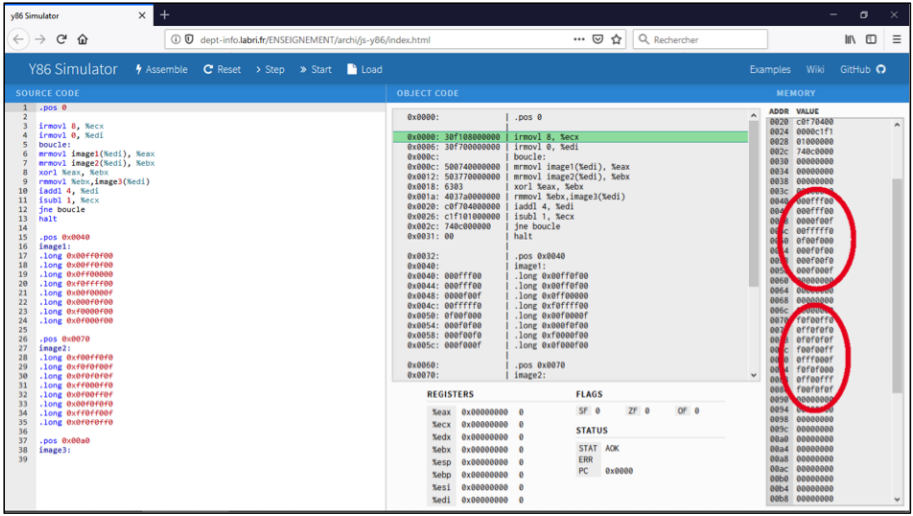


Image clef :



On peut les voir dans la colonne « Memory » du simulateur Y86 :

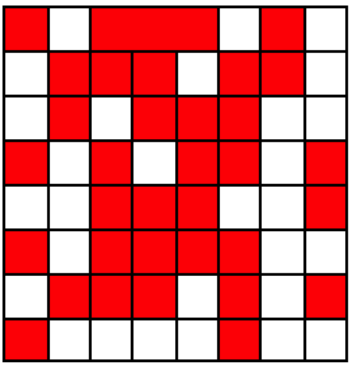


La différence entre l'aspect des images dans la dernière colonne et l'écriture des nombres dans le code source vient du fait que y86 fonctionne en « big endian », donc le nombre hexadécimal « 12345678 » sera stockée en mémoire sous la forme « 78563412 ». Il faudra y penser si on veut coder une autre image.

On fait fonctionner le programme pas à pas : bouton « Assemble », puis « Step ».

- Que se passe-t-il à chaque pas ?
- Lors de chaque instruction d'affectation, regardez la valeur de la case mémoire ou du registre concerné.
- Le programme charge dans les registres %eax et %ebx les nombres correspondant aux lignes successives des images 1 et 2. Quelle opération mathématique réalise-t-il entre ces deux registres ?

- A la fin de l'exécution du programme, que retrouve-t-on à l'adresse 0x00a0 (label Image3) ?
- Représenter ci-dessous l'image obtenue à la fin du programme :



- Modifier le code afin d'appliquer l'algorithme à l'image précédente. Attention à ne pas modifier l'image clef et à bien penser au fait que les nombres sont stockés en mémoire en « big endian ».
- Retrouve-t-on l'image de départ ?

Prolongement

Faire un programme Python permettant de réaliser l'opération XOR entre les pixels d'une image à crypter et d'une image bruitée. Réaliser deux fois l'opération pour montrer qu'on retrouve l'image de départ :