Architecture des Ordinateurs
et
Systèmes d'Exploitation

Partie 1 : Architecture des Ordinateurs
Cours

Fabrice BOISSIER & Elena KUSHNAREVA
2017/2018

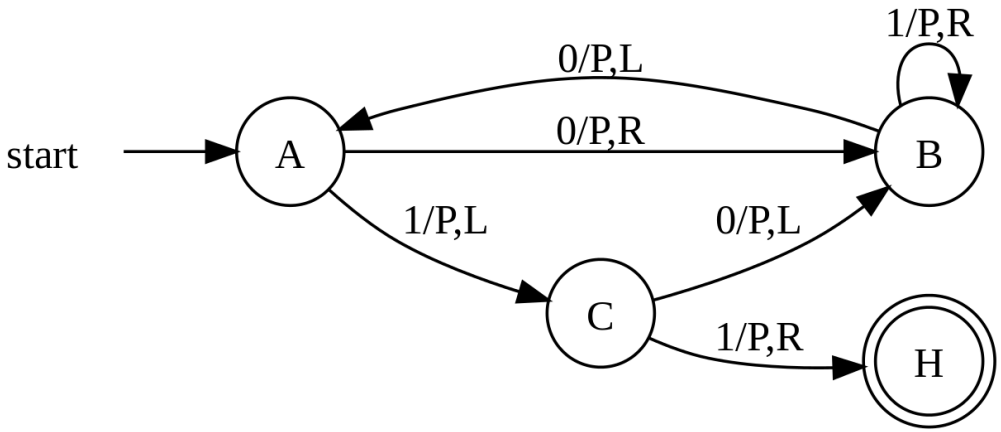
fabrice.boissier@gmail.com
elena.kushnareva@malix.univ-paris1.fr

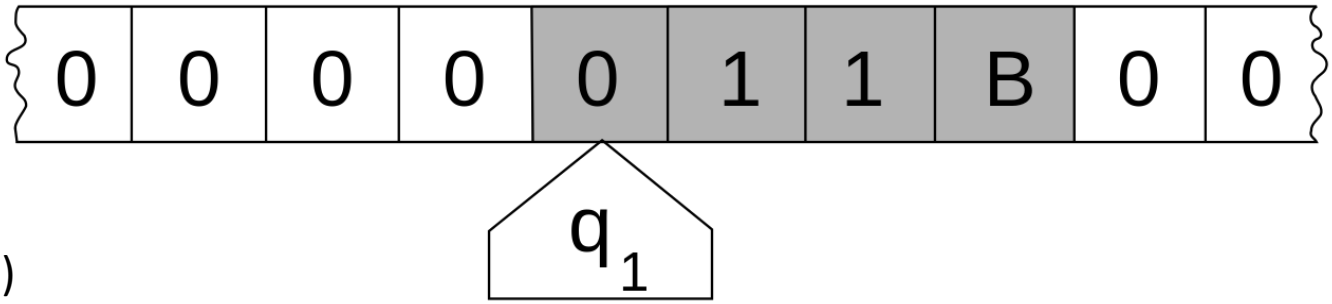
Machine de Turing

- Machine Abstraite :
  - à états
  - effectuant des opérations
  - sur un ruban infini
  - selon des actions définies à l'avance

Machine de Turing

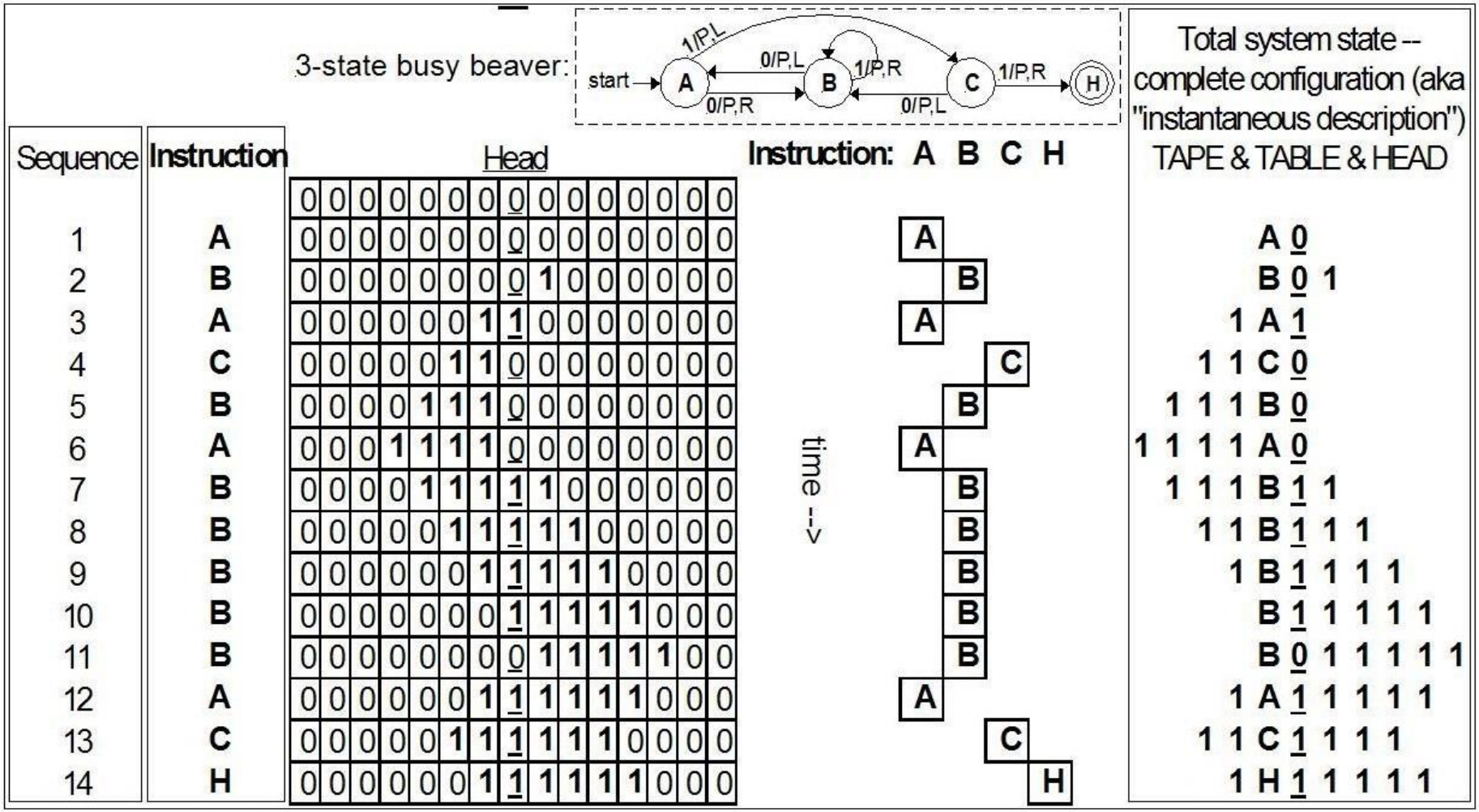
- Machine Abstraite à états, effectuant des opérations, sur un ruban infini





(illustrations de Wikipedia)

Machine de Turing

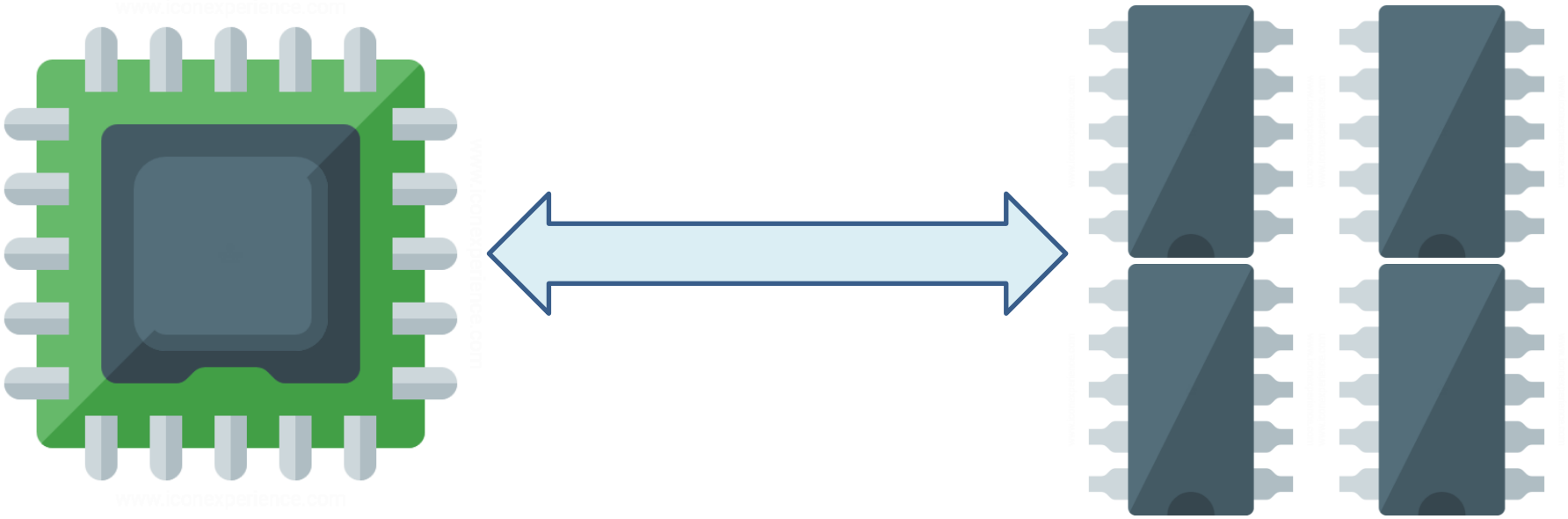


Progress of the computation (state-trajectory) of a 3-state busy beaver

(illustration de Wikipedia)

Les Composants Minimaux d'un Ordinateur

Un Processeur…



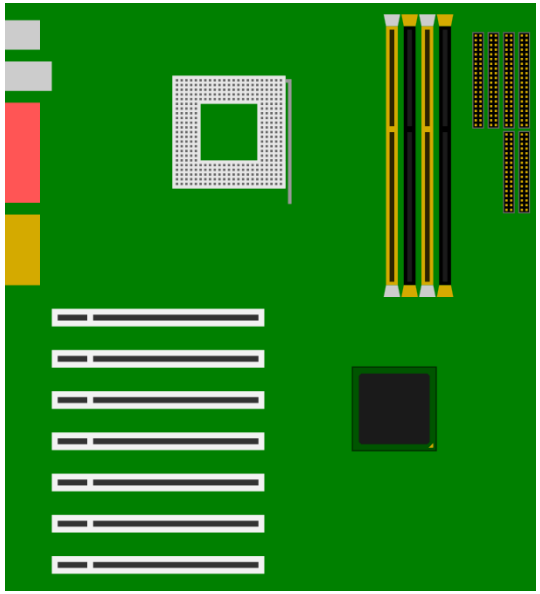
…et de la Mémoire

Les Composants Minimaux d'un Ordinateur

- Processeur :
  - Exécute des instructions
  - Manipule des données
  - Manipule des adresses en mémoire

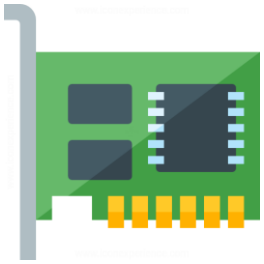
- Mémoire :
  - Contient des données à des adresses précises
  - Extrait ou Met à jour les données

Les Composants d'un Ordinateurs



- Carte Mère :
  - Nombre max de CPU
  - Nombre max de Mémoire
  - Nombre max de Périphériques
  - Horloge commune pour fonctionner

- Périphériques :
  - Entrée (clavier, souris, scanner, …)
  - Sortie (écran/carte graphique, imprimante, …)
  - Entrée & Sortie (disques durs, graveurs disques, carte son, contrôleurs réseau, contrôleurs USB, …)



Les Ordinateurs

- Anciennement, il y avait 3 classes d'ordinateurs :
  - Mainframes (IBM, …)
  - Minis (IBM, HP, Oracle, SGI, …)
  - Micros (Lenovo, HP, Sony, Asus, Acer, …)

- Quelques autres « classes » :
  - « Nanos » / Smartphones (Apple, Samsung, …)
  - « Nanos » / Embarqués (Texas Instrument, Atmel, …)
  - Super-Calculateurs (IBM, HP, SGI, Cray, Bull, …)

Les Micro-Ordinateurs

Les Mini-Ordinateurs

Les Mainframes

Les Super-Ordinateurs (Super Calculateurs)

Les Classes d'Ordinateurs

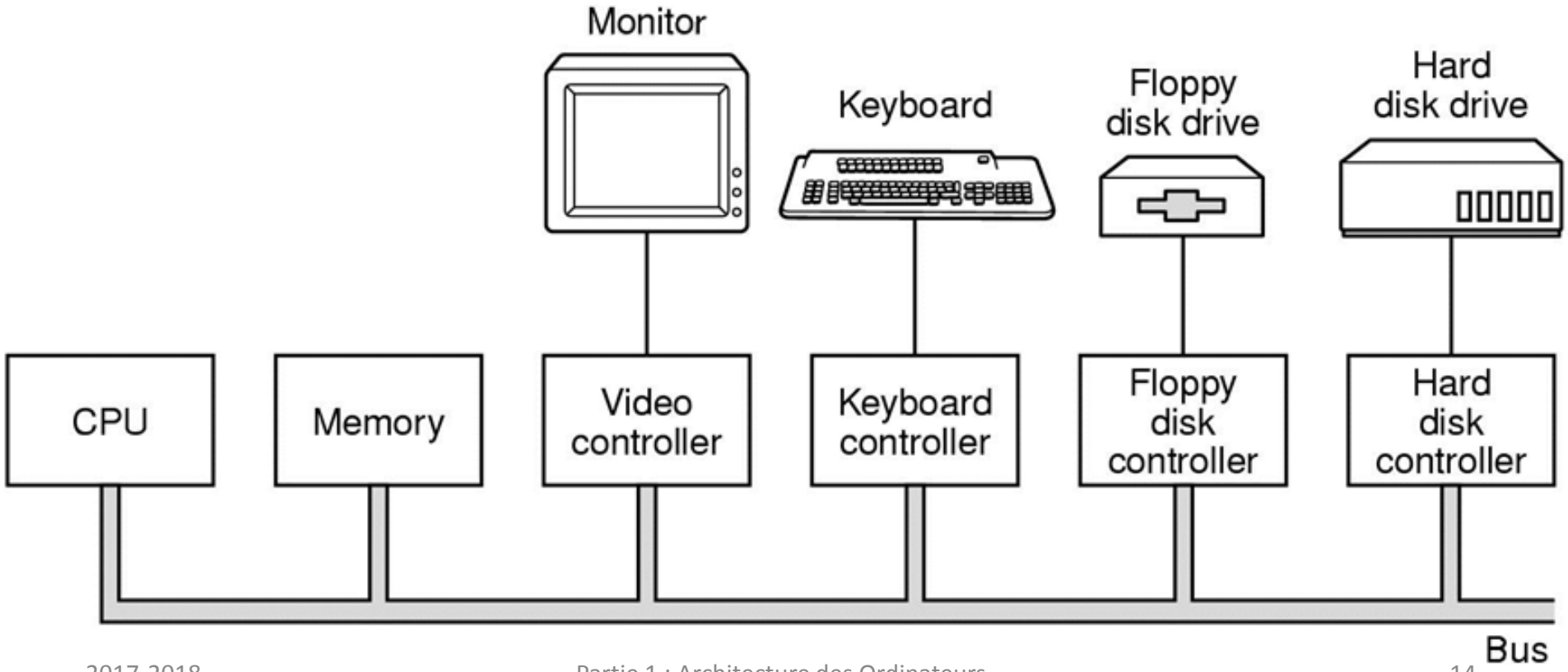
- Mainframes
  - IBM (z System), Bull (GCOS), Fujitsu, HP, …
  - Dédié I/O et très forte charge
  - Un seul référentiel
- Minis
  - IBM : System i (AS/400), System p (POWER)
  - DEC : VAX (VMS), PDP, … (†)
  - Plus récemment : HP, Sun/Oracle, SGI, …
  - Innombrables UNIX (AIX, HP-UX, Solaris, …)
  - Se relient les uns aux autres
- Micros
  - PC (5150 et « compatibles PC »)
  - Atari ST, Commodore, …
  - Bureautique, jeux : graphique (GUI, IHM, …)

Super-Ordinateurs
Super-Calculateurs

- Cray, Blue Gene, …
- POWER, Xeon, SPARC64, Opteron, GPU Nvidia, …
- Calcul ! Pas I/O !
  Effets d'une bombe atomique, simulation de neurones, …
- Multiples machines reliées,
  PS3 en série, …

Architecture Générale

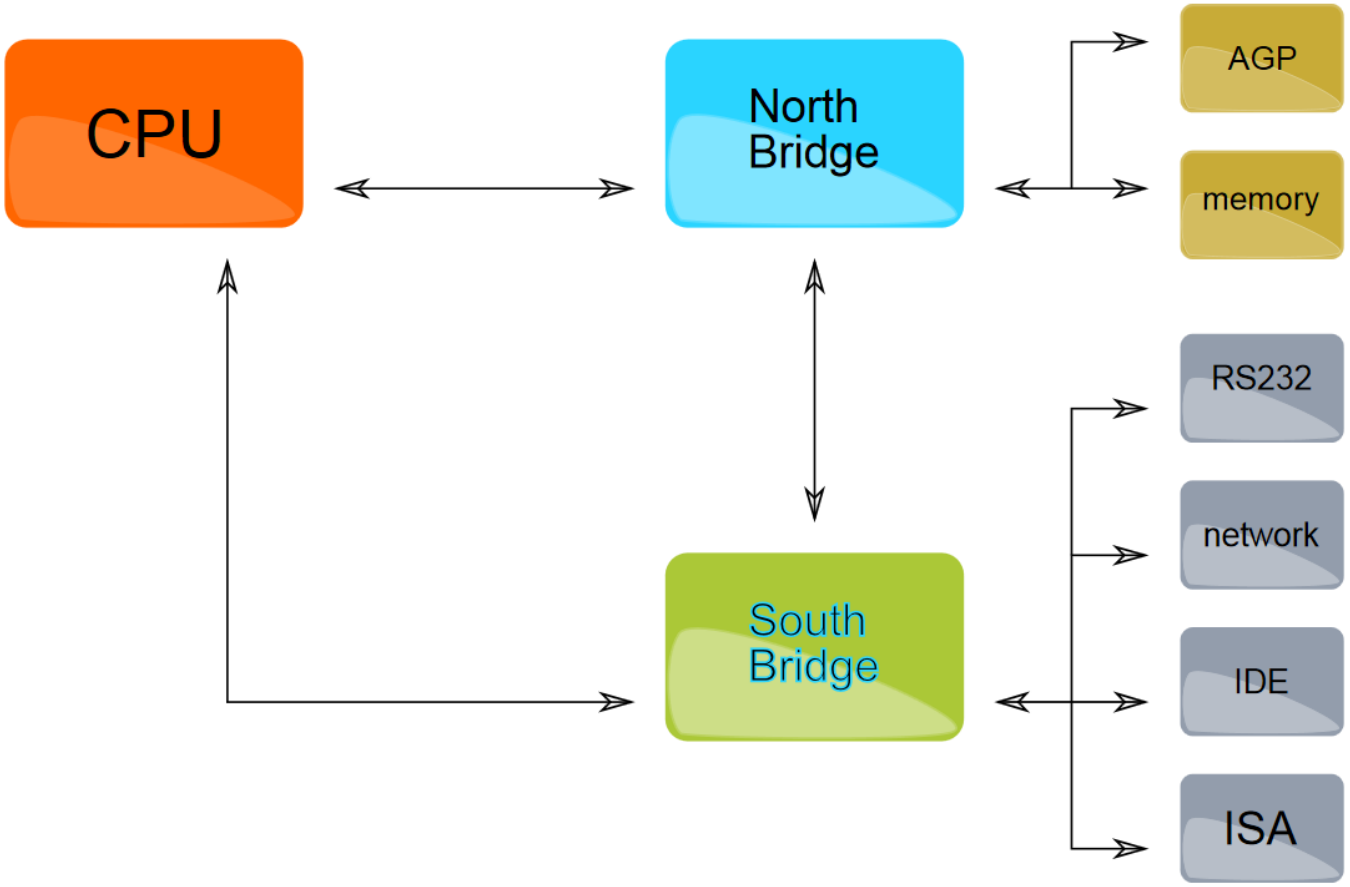
1 Bus relie tous les composants de l'ordinateur
Ce bus sert à faire communiquer les composants entre eux



Exemple incontournable de Micro-Ordinateur :

Le PC : *Personal Computers, IBM-PC / IBM-5150*

Utilise un processeur Intel 8086 ou x86



Références Bibliographiques

- Freescale/NXP® Motorola 68000 Microprocessors User's Manual – 9th edition
- Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1-2A-2B-2C-3A-3B-3C-3D
- SPARC International Inc. The SPARC Architecture Manual (v8, v9) SPARC Compliance Definition 2.2
- Zilog Z80 Microprocessors Z80 CPU – User Manual
- MIPS Technologies MIPS IV Instruction Set (Rev. 3.2) MIPS32™ Architecture For Programmers (Volume I)

- IBM (publib.boulder)
- Oracle
- HP / HPE
- Bull
- Silicon Graphics
- Texas Instrument
- Atmel
- Cray
- Apple
- Lenovo
- Asus
- Acer
- Sony
- Samsung
- …

Standards / Définition

- 1 bit peut prendre 2 valeurs : 0 ou 1

- 8 bits = 1 octet
- 1 octet peut prendre 256 valeurs (2^8) :
  - de 0 à 255 s'il est « non-signé »
  - de -128 à +128 s'il est « signé »

Taille des Mots / Words

- WORD = mot = taille d'une donnée « normale »
- DWORD = double word = 2 fois la taille d'un WORD
- QWORD = quad word = 4 fois la taille d'un WORD

WORD	DWORD	QWORD
8 bits	16 bits	32 bits
16 bits	32 bits	64 bits
32 bits	64 bits	128 bits
…	…	…

Taille des Mots / Words

- Diminutifs :
  - Bus d'@ = Bus d'adresses
  - Bus de données = Taille d'une donnée
    (8 bits, 16 bits, 32 bits, 64 bits)

- La taille d'un WORD est définie pour chaque famille de processeurs par le fabricant

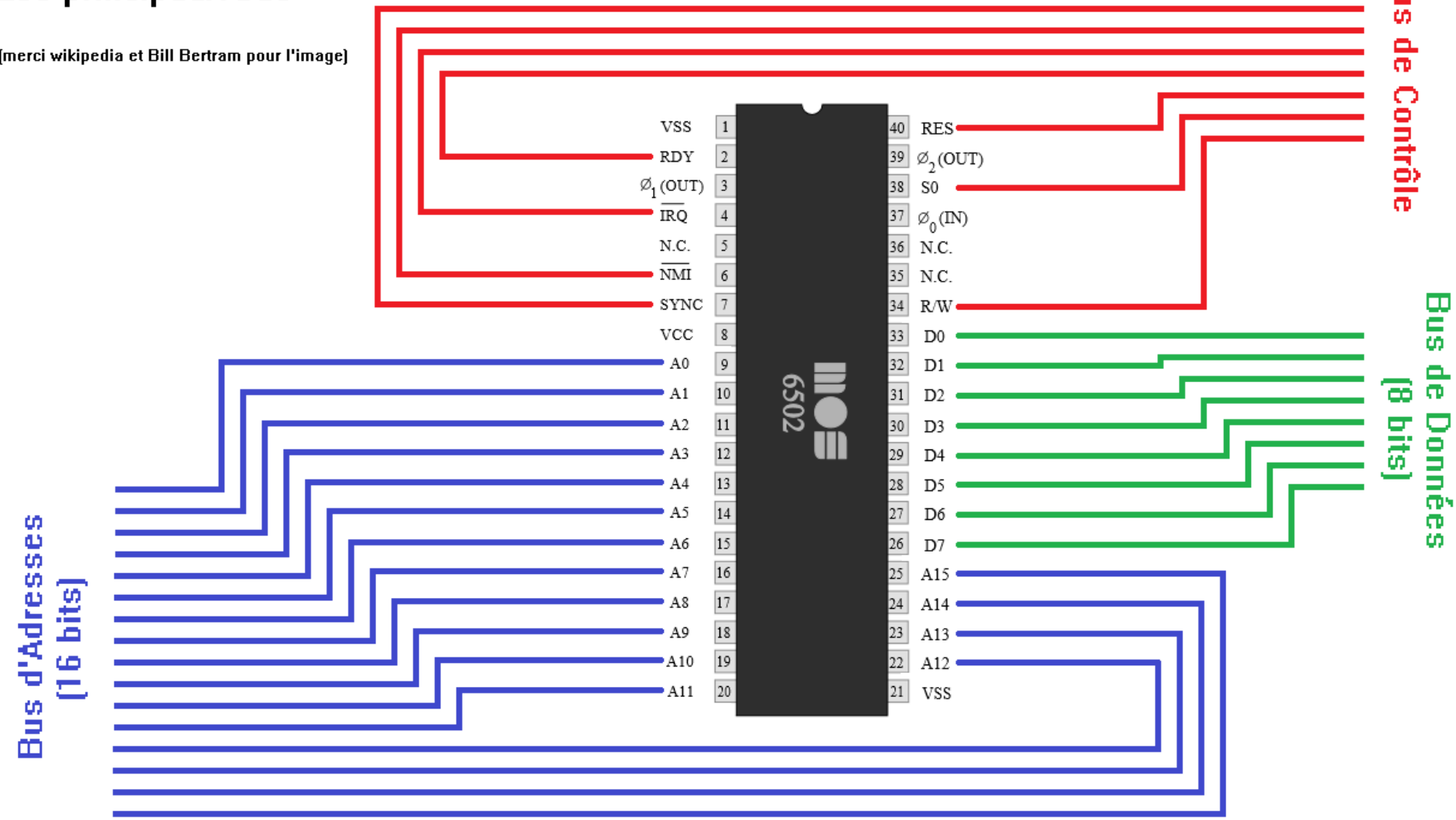
Les 3 Bus

- Bus d'adresse
  - Sélectionne une adresse
- Bus de données
  - Écrit un mot
  - Lit un mot
- Bus de contrôle
  - Gestion des interruptions
  - Gestion des composants externes

Les 3 Bus

Exemple MOS 6502 :
Les principaux bus

(merci wikipedia et Bill Bertram pour l'image)

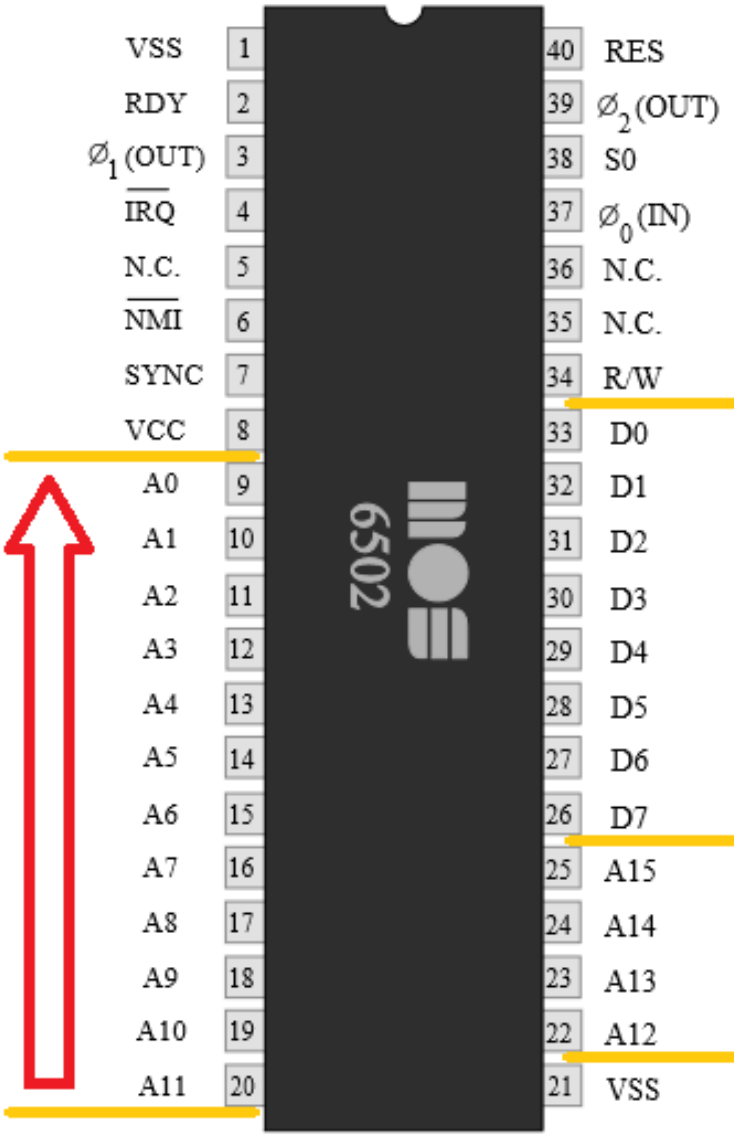


La taille des bus

Exemple MOS 6502 :

Taille des bus

(merci wikipedia et Bill Bertram pour l'image)



Bus d'@ de taille 16 bits

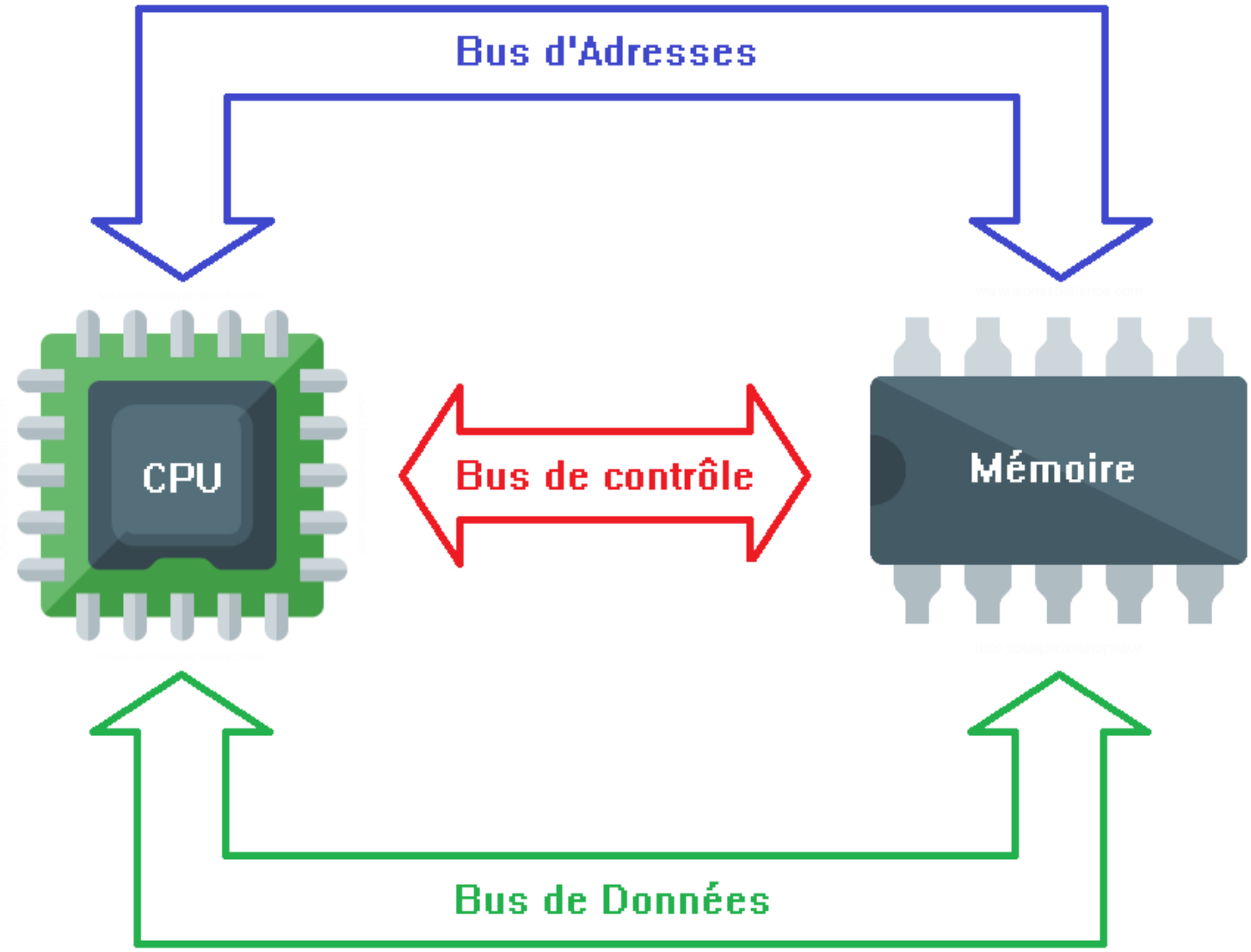
Soit 65.536 adresses accessibles

8 pins = 8 bits (de 0 à 7)

Bus de données WORD = 8 bits

16 pins = 16 bits (0 à 15)

La Mémoire



La Mémoire

- Lecture :

  1 - Le processeur met sur le bus d'@, l'adresse qu'il souhaite atteindre

  1 - Le processeur indique, via le bus de contrôle, qu'il souhaite lire (READ)

  2 - La mémoire charge l'adresse et récupère la donnée

  3 - Le processeur indique sur le bus de contrôle qu'il est prêt à recevoir la donnée

  4 - La mémoire prend la donnée, et l'envoie sur le bus de données

La Mémoire

- Ecriture :

  1 - Le processeur met sur le bus d'@, l'adresse qu'il souhaite atteindre

  1 - Le processeur indique sur le bus contrôle qu'il souhaite écrire (WRITE)

  2 - La mémoire charge l'adresse

  3 - Le processeur indique, via le bus de contrôle, qu'il est prêt à émettre la donnée

  3 - Le processeur charge la donnée sur le bus de données

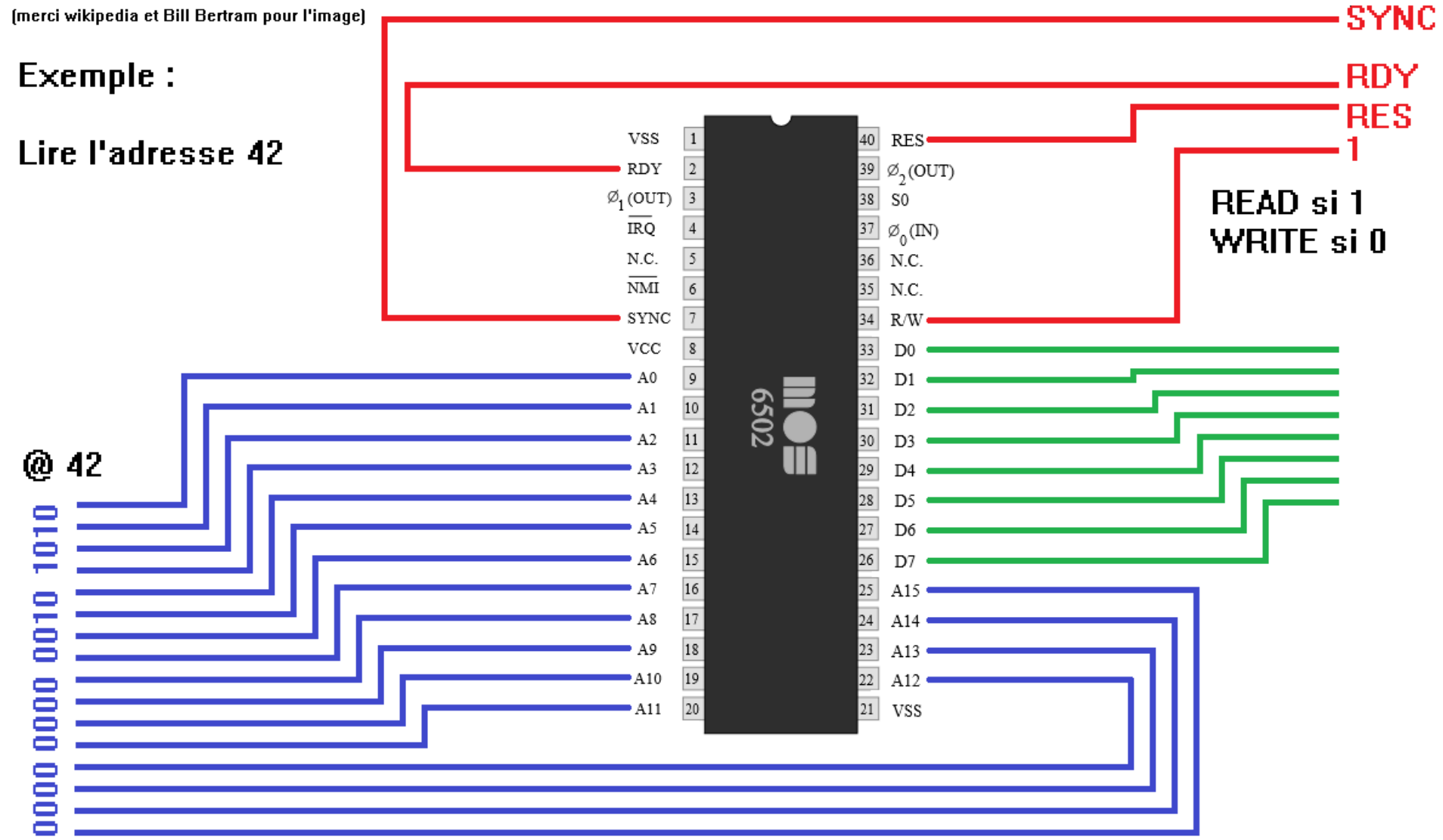
  4 - La mémoire prend la donnée, et l'écrit à l'adresse indiquée

Exemple de Lecture en Mémoire

(merci wikipedia et Bill Bertram pour l'image)

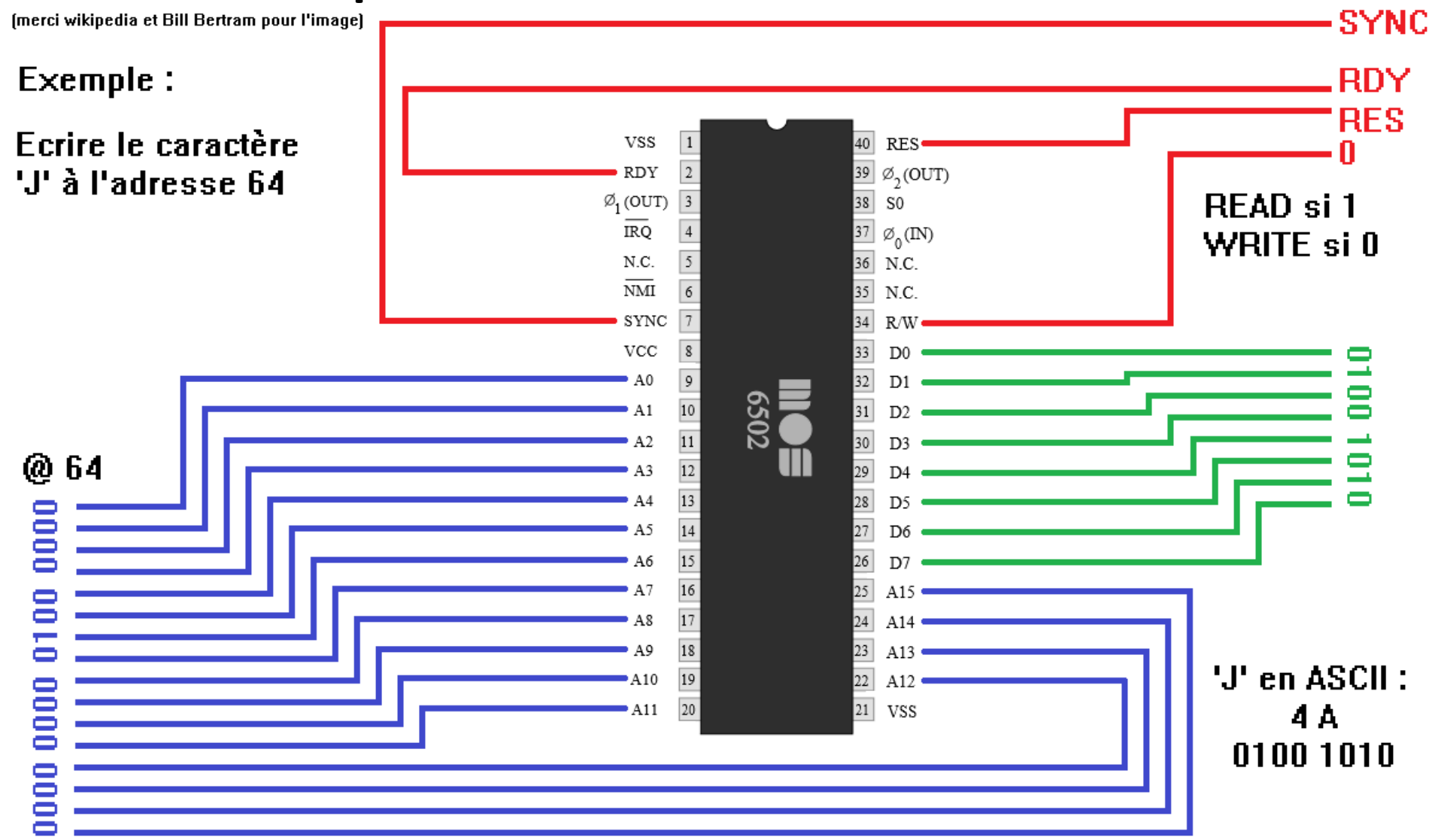
Exemple :

Lire l'adresse 42

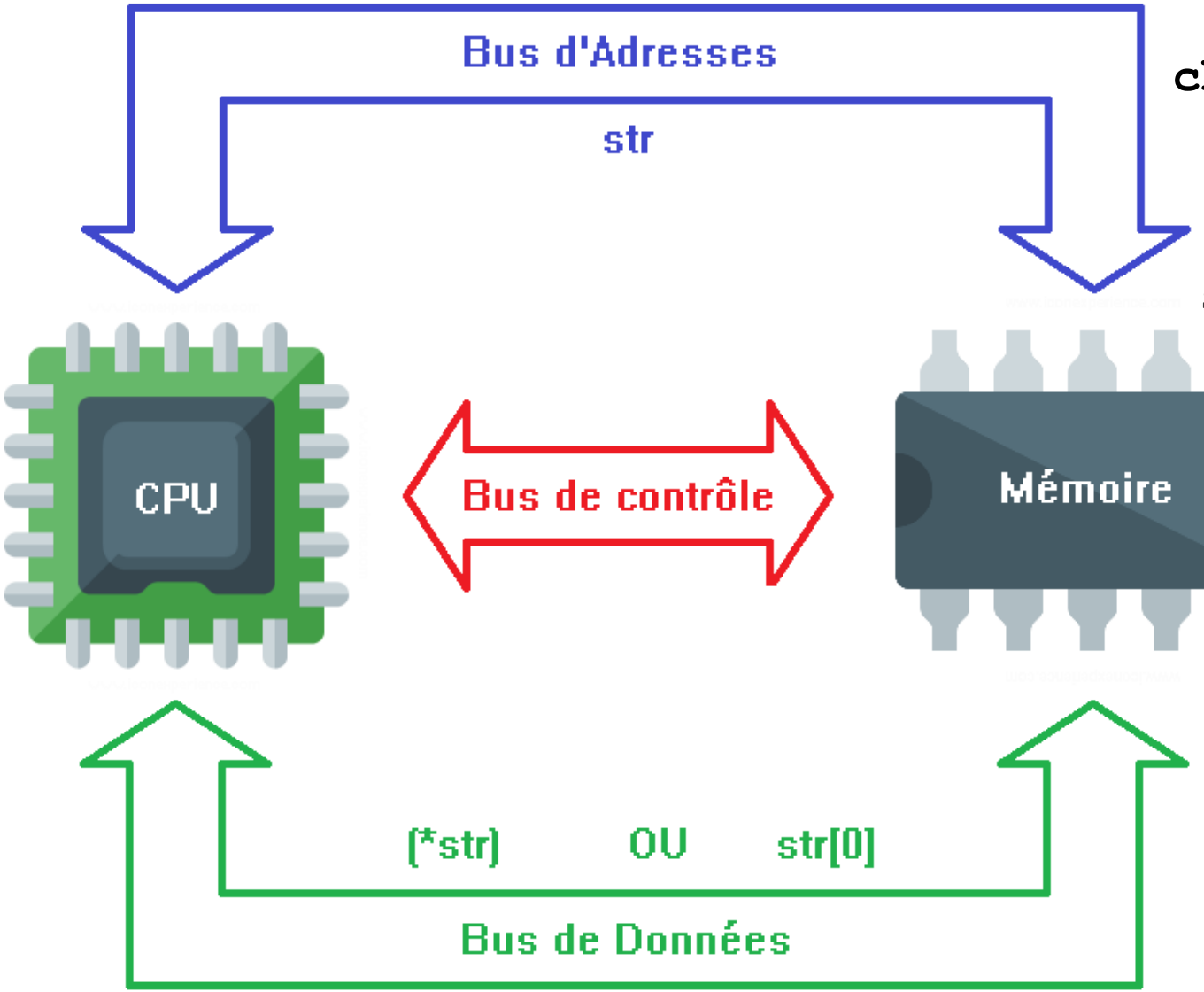


Exemple d'Écriture en Mémoire

(merci wikipedia et Bill Bertram pour l'image)



La Mémoire & Pointeurs C



`char *str = "Bla";`

`str = une adresse`

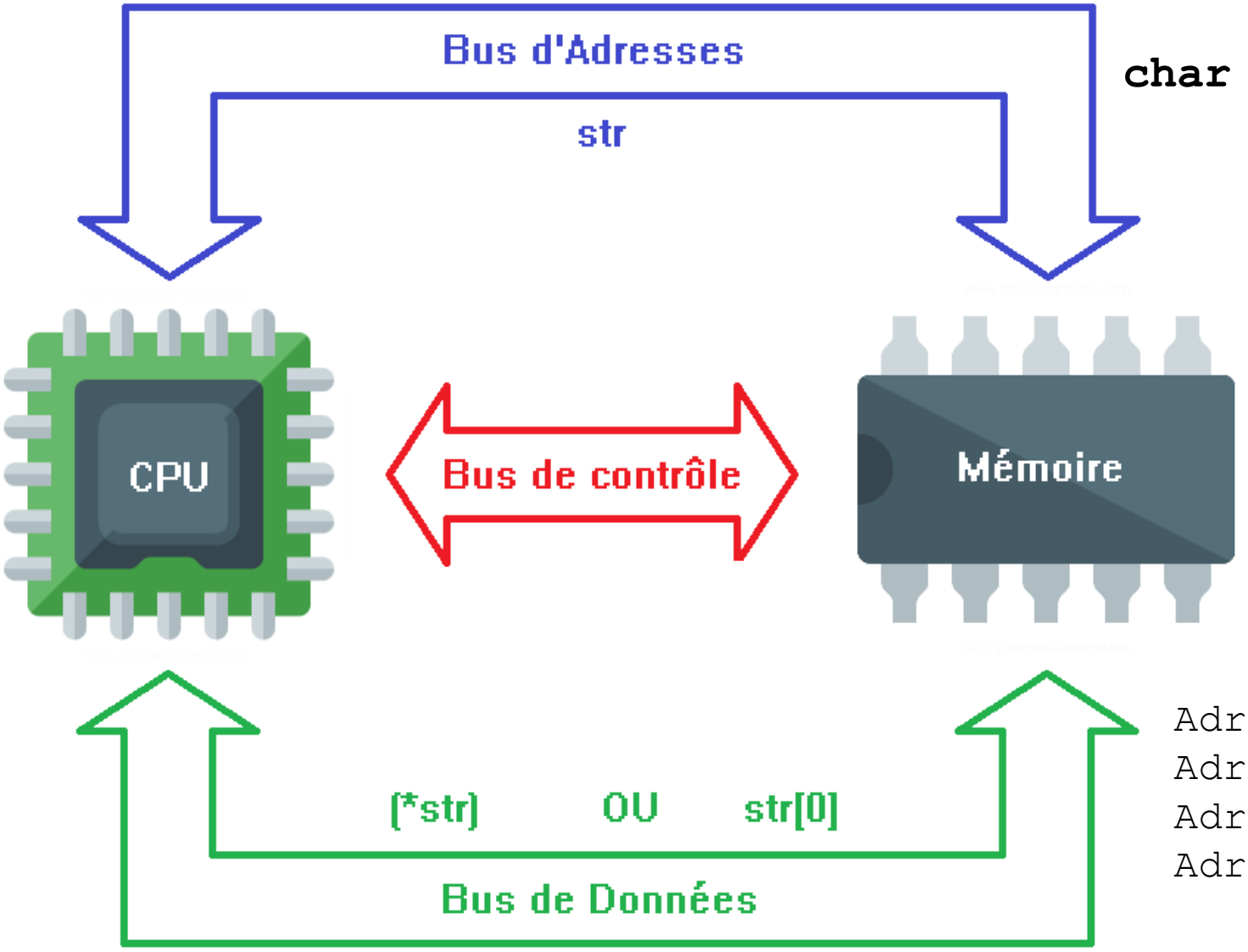
`(*str) = la donnée`

`str[0] = 1er octet pointé à l'@`

`str = le pointeur`

`(*str) = déréférencement de l'adresse contenue dans` ***`str`***

La Mémoire & Pointeurs C




```
char *str = "Bla";
```

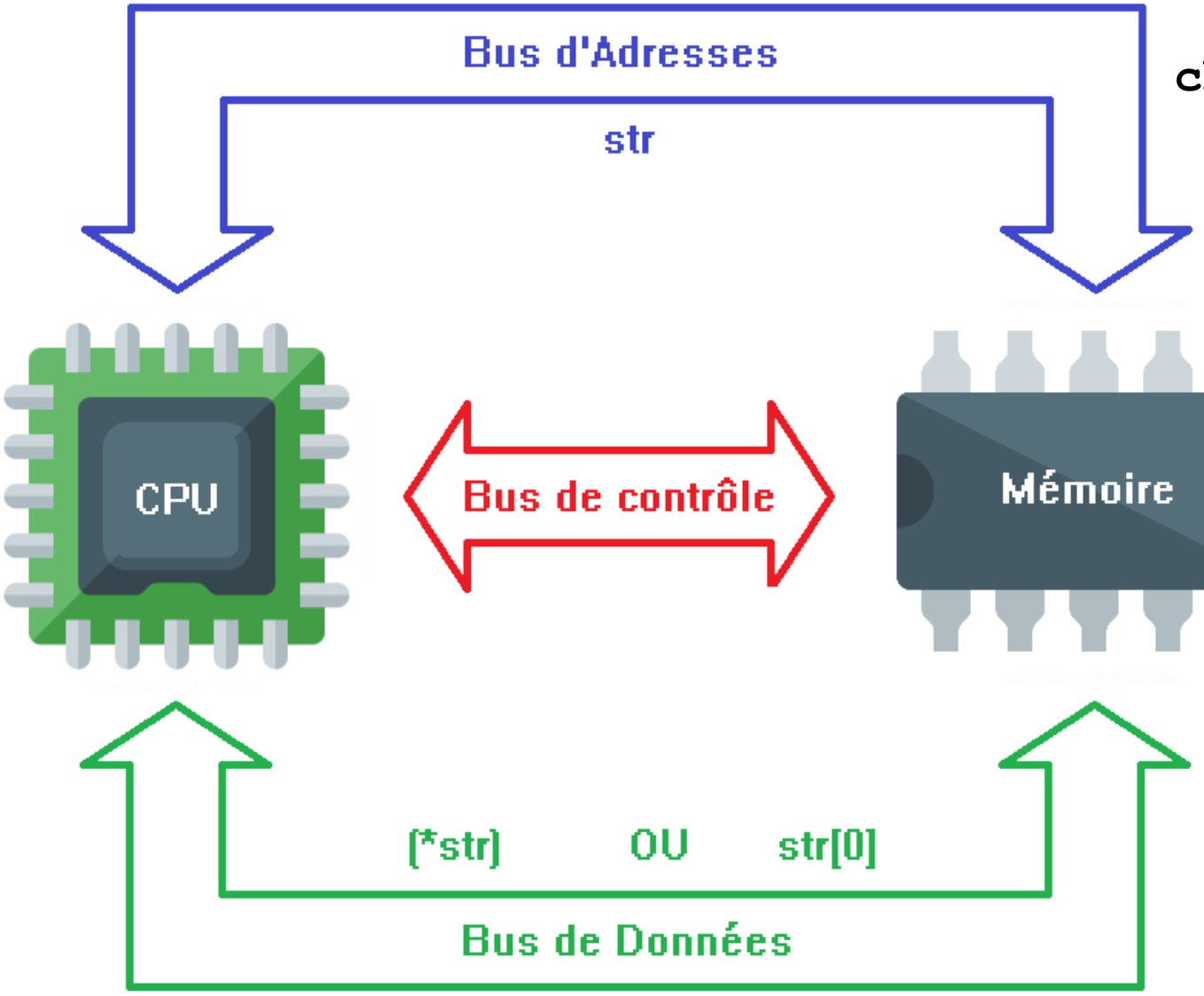
Exemple :

Admettons que str contient l'adresse 42 (str == 42)

En mémoire : (cas ASCII)

Adresse 42 == 'B'
Adresse 43 == 'l'
Adresse 44 == 'a'
Adresse 45 == \0

La Mémoire & Pointeurs C




```
char *str = "Bla";
```

Exemple avec :
str == 42

En mémoire :
(cas ASCII)

```
str[0] = 42+0 = 'B'
str[1] = 42+1 = 'l'
str[2] = 42+2 = 'a'
str[3] = 42+3 = \0
```

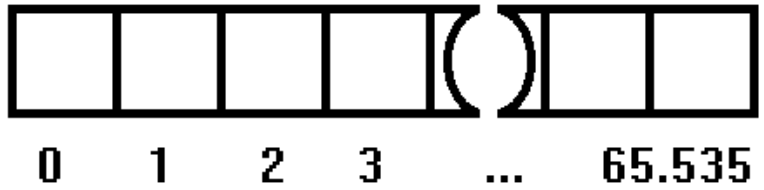
Espace d'Adressage

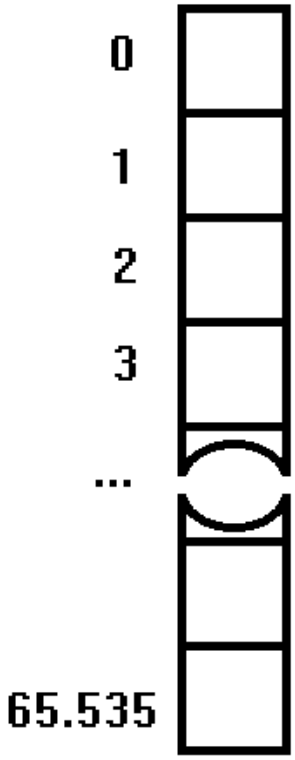
- Aucune convention sur la représentation n'existe. Il s'agit juste d'un ensemble de "cases" qui se suivent. La taille des cases varie selon plusieurs paramètres que nous ne verrons pas.

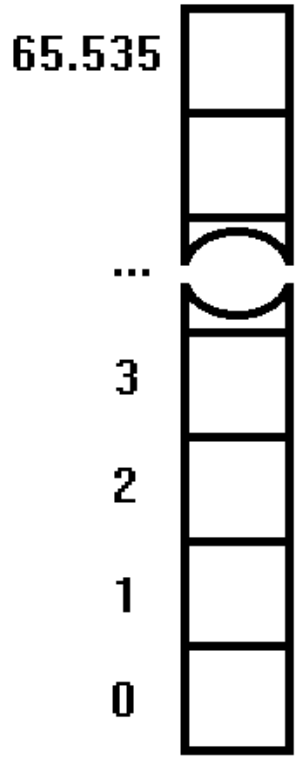
- Chaque modèle de processeur dispose d'un bus d'adresse qui va définir les adresses minimales et maximales accessibles :
de 0 à 2^(largeur du bus d'adresse) – 1

- Le processeur y lit et écrit des mots (taille du mot définie par le modèle).

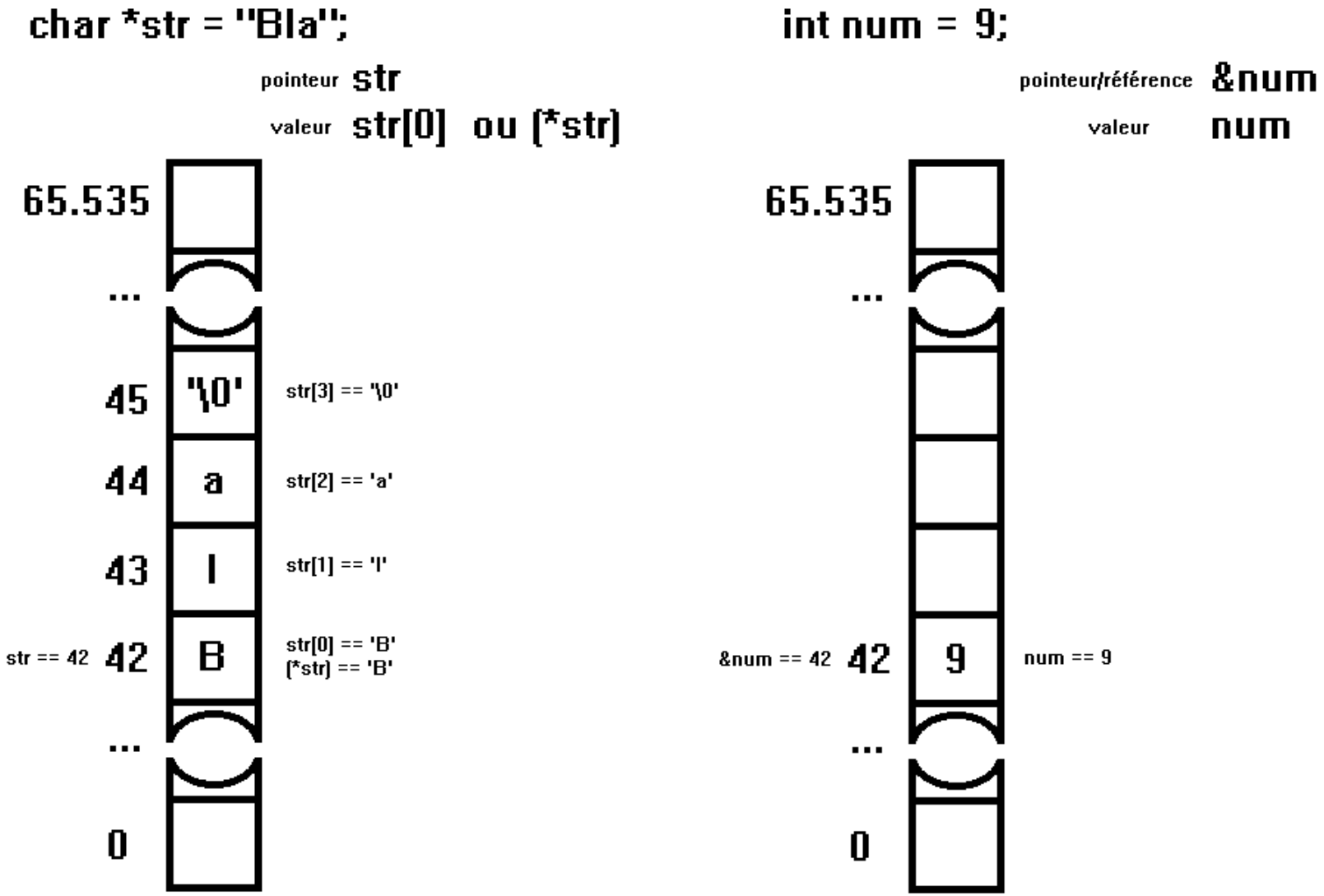
- Selon les modèles, certaines adresses ne sont pas accessibles pour une question "d'alignement" : les adresses impaires ne sont pas accessibles sur certains processeurs, car elles contiennent la moitié d'un mot.







Espace d'Adressage et C

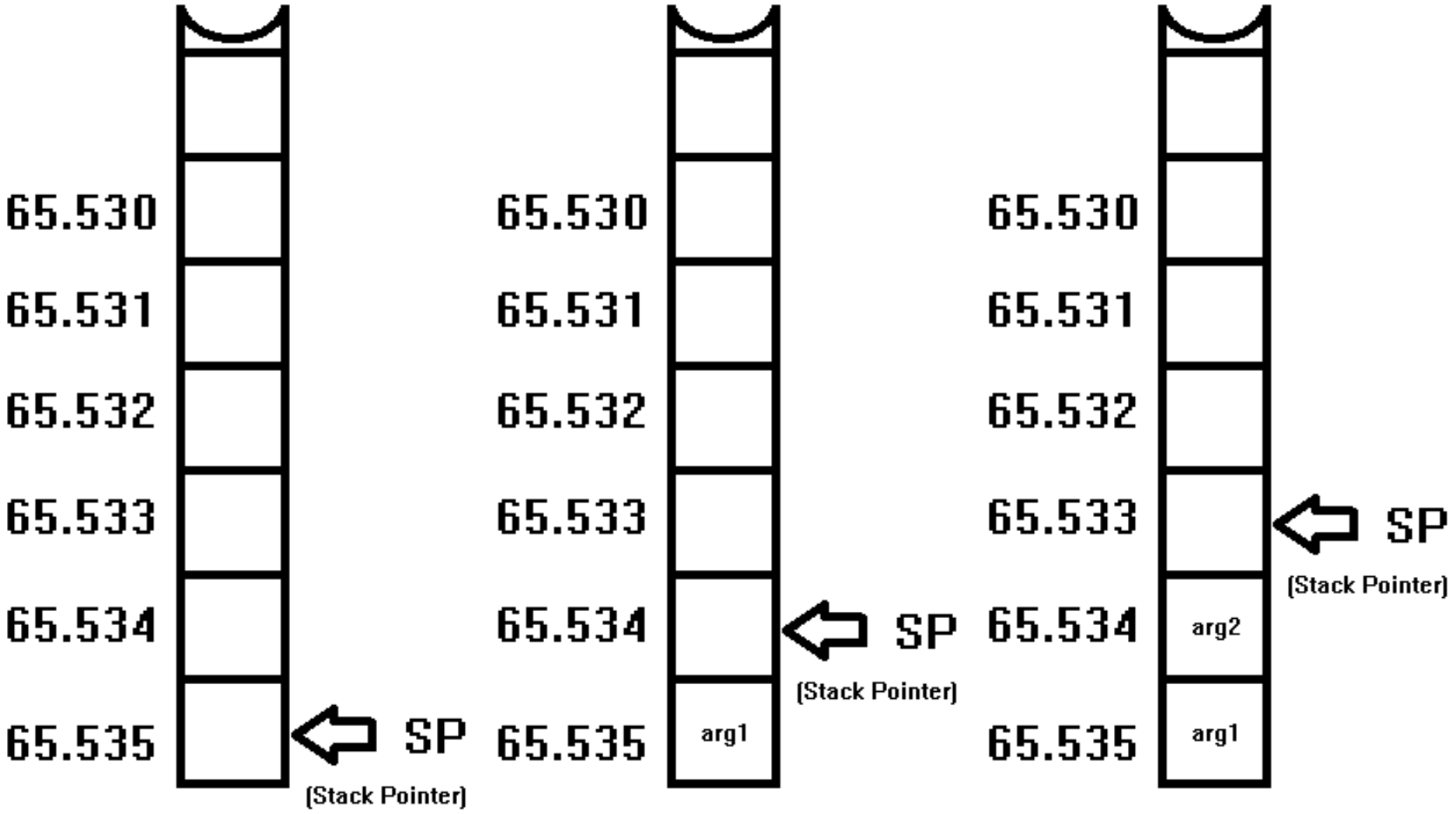


La Pile / Stack

- A l'usage, un processeur dispose d'une "pile" (stack) où sont stockées des données et divers paramètres du programme en cours d'exécution.
  Cette pile démarre à la première ou à la dernière adresse disponible de la mémoire, et se remplit au fur et à mesure des besoins (bien évidemment, si la pile est pleine, la mémoire l'est aussi, et plus rien ne fonctionne).

- Selon l'architecture du processeur, la pile va permettre d'appeler des fonctions et de passer des arguments : on pose l'argument 1 (push), puis l'argument 2 (push), on appelle la fonction (call), celle-ci va récupérer l'argument 2 (pop), puis l'argument 1 (pop), et les utiliser.
  C'est ce que font les x86 (PC 16-32 bits).

La Pile / Stack



Les Interruptions

- Rappel sur les Périphériques :
  - D'entrée : écrit des données en mémoire
  - De sortie : lit des données en mémoire
  - D'entrée/sortie : lit et écrit des données

Les Interruptions

- Un périphérique d'entrée, pour émettre ses données, ne peut pas écrire sur le bus directement, car celui-ci est "peut être" utilisé par un autre périphérique ou par le processeur lui-même.

1. Pour demander l'autorisation d'utiliser le bus, le périphérique va émettre une "interruption" matérielle (envoyer des bits sur le bus de contrôle) qui sera entendue par le processeur.
2. Le processeur va décider si oui ou non il laisse le bus libre au périphérique en lui renvoyant des données sur le bus de contrôle.

Les Interruptions

3. Une fois que le périphérique obtient le droit d'écrire sur le bus, celui-ci va émettre ses données sur le bus de données.
4. Le processeur va prendre chacun des mots (mot par mot), et les renvoyer vers la mémoire.

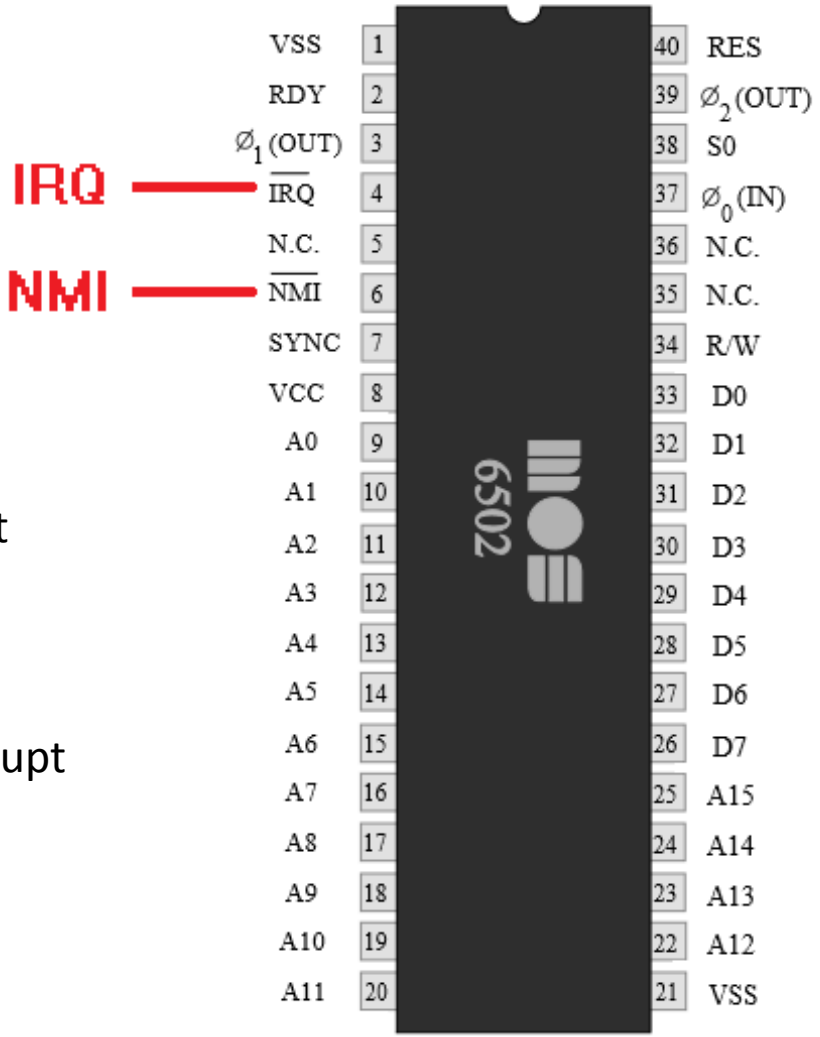
- Depuis plusieurs années, des "DMA" (*Direct Memory Access*) permettent au processeur de se décharger de la fonction de copie : le périphérique va directement écrire en mémoire, laissant le processeur reprendre ses activités immédiatement. Mais une interruption est levée à la fin de la copie.

Les Interruptions

- Une "interruption" interrompt l'exécution du programme en cours dans le processeur, et force celui-ci à exécuter un programme précis enregistré au préalable.
  Une fois que l'interruption est gérée (le programme lancé par l'interruption se termine), le processeur peut reprendre l'exécution du programme précédent, exactement où il s'était arrêté.

Les Interruptions

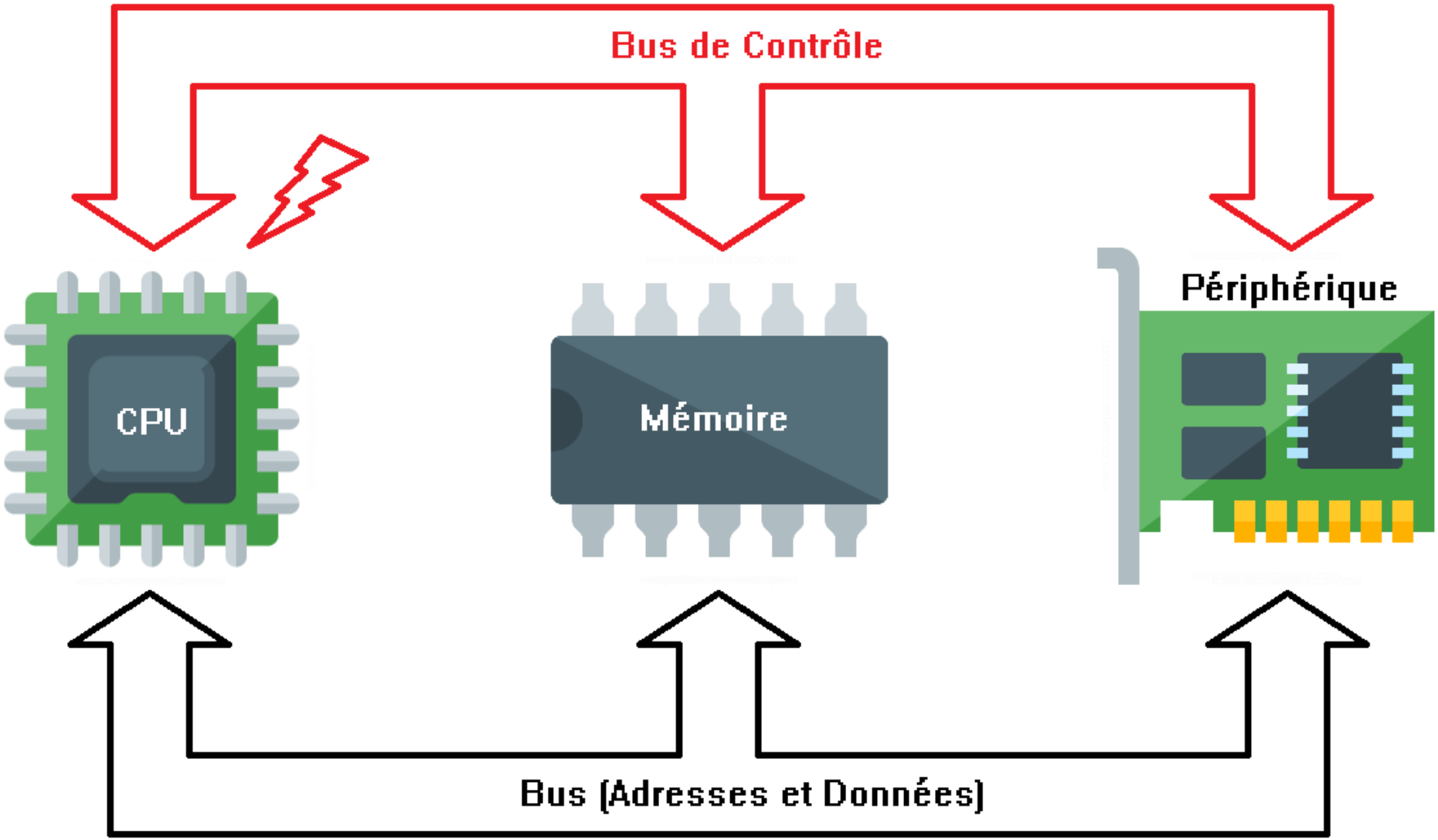
(merci wikipedia et Bill Bertram pour l'image)



IRQ :
Interrupt ReQuest

NMI :
Non Maskable Interrupt

Les Interruptions



Les Interruptions

- Deux types d'interruptions :
  - **Matérielles** : déclenchée par un périphérique ou des horloges sur la carte mère
  - **Logicielles** (trap) : les « appels système » (*syscalls*), les erreurs arithmétiques (div 0), données non disponibles en mémoire (page fault)

Les Interruptions

- À chaque interruption, un programme précis est lancé pour traiter celle-ci

  Leur gestion passe donc par un programme

- Les interruptions sont masquables :
  - Les interruptions les moins importantes peuvent être interrompues par des interruptions plus importantes
  - Les plus importantes ne peuvent pas être interrompues

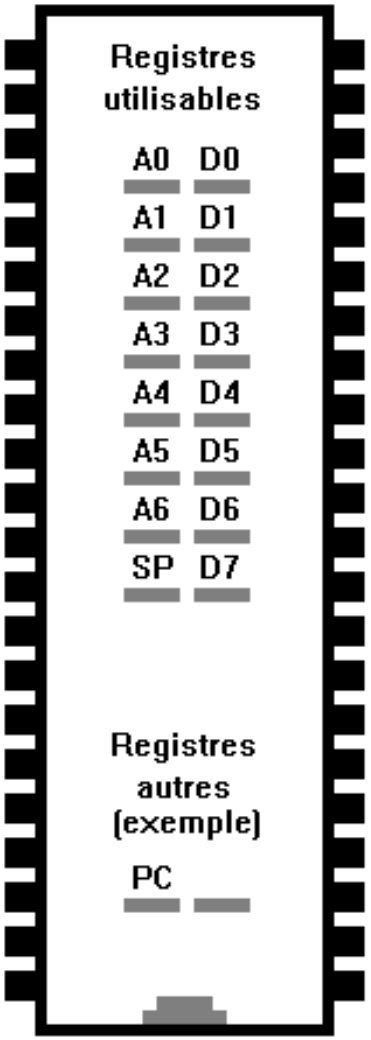
Les Registres

- Plusieurs registres sont disponibles dans les processeurs pour que les programmes puissent enregistrer des valeurs temporairement (par exemple les variables) selon le modèle de processeur.

- Les langages de développement permettent de s'abstraire de ces registres. Cependant, la connaissance de leur existence lors du développement permet d'optimiser l'exécution des programmes.

Les Registres

- En général, les processeurs disposent d'au moins :
  - quelques registres pour stocker des données (adresses et/ou données)
  - un registre pour le pointeur de pile (Stack Pointer/SP)
  - un registre contenant l'adresse de l'instruction en cours d'exécution (Program Counter/PC)



Les Flags

- Des flags sont également stockés dans le processeur (parfois dans un registre dédié). Les flags sont souvent des valeurs à 0 ou 1. Parmi les flags communs :
  - Zero : la donnée est égale à 0
  - Negative : la donnée, si signée, est négative
  - Overflow : la donnée, si signée, a dépassé le max
  - Carry : la donnée, si non signée, a dépassé le max
  - Supervisor/Ring : mode utilisateur ou superviseur

Assembleur

- L'assembleur est la traduction en langage "humain" des instructions que la machine va exécuter (les *instructions machine*)

- Chaque ligne est constituée (parfois d'un label), d'une instruction et d'opérandes

- Exemple : MOVEA 8(A6), A0
  - **MOVEA** est une instruction
  - **8(A6)** est une opérande
  - **A0** est une opérande

- On retrouve dans l'assembleur les registres dont on avait parlé, des adresses mémoire, et des "labels" pour s'abstraire des adresses mémoire.

Langage Machine

```
00100000
00100000  CE56 0000
00100004  206E 0008
00100008  226E 000C
0010000C  1018
0010000E  0C00 0041
00100012  650A
00100014  0C00 005A
00100018  6204
0010001A  0600 0020
0010001E  12E0
00100020  66E8
00100022  4E5E
00100024  4E75
00100026
```

Assembleur

```
; strtolower:
; Copy a null-terminated ASCII string, converting
; all alphabetic characters to lower case.
;
; Entry parameters:
;   (SP+0): Source string address
;   (SP+4): Target string address

                org     $00100000       ;Start at 00100000
strtolower      public
                link    a6,#0           ;Set up stack frame
                movea   8(a6),a0        ;A0 = src, from stack
                movea   12(a6),a1       ;A1 = dst, from stack
loop            move.b  (a0)+,d0        ;Load D0 from (src)
                cmpi    #'A',d0         ;If D0 < 'A',
                blo     copy            ;skip
                cmpi    #'Z',d0         ;If D0 > 'Z',
                bhi     copy            ;skip
                addi    #'a'-'A',d0     ;D0 = lowercase(D0)
copy            move.b  d0,(a1)+        ;Store D0 to (dst)
                bne     loop            ;Repeat while D0 <> NUL
                unlk    a6              ;Restore stack frame
                rts                     ;Return
                end
```

(merci Wikipedia, article Motorola 68000)

Assembleur & C

- Chaque famille de processeur dispose de son propre assembleur
- Le C offre une logique plus abstraite que l'assembleur, mais il garde quelques notions liées à la mémoire.
- Le C est surnommé « l'Assembleur Portable », car il ne dépend d'aucun processeur

Assembleur & C

Assembleur

```
; strtolower:
; Copy a null-terminated ASCII string, converting
; all alphabetic characters to lower case.
;
; Entry parameters:
;   (SP+0): Source string address
;   (SP+4): Target string address

                org     $00100000       ;Start at 00100000
strtolower      public
                link    a6,#0           ;Set up stack frame
                movea   8(a6),a0        ;A0 = src, from stack
                movea   12(a6),a1       ;A1 = dst, from stack
loop            move.b  (a0)+,d0        ;Load D0 from (src)
                cmpi    #'A',d0         ;If D0 < 'A',
                blo     copy            ;skip
                cmpi    #'Z',d0         ;If D0 > 'Z',
                bhi     copy            ;skip
                addi    #'a'-'A',d0     ;D0 = lowercase(D0)
copy            move.b  d0,(a1)+        ;Store D0 to (dst)
                bne     loop            ;Repeat while D0 <> NUL
                unlk    a6              ;Restore stack frame
                rts                     ;Return
                end
```

C

```
void    strtolower(char *str_in, /* string to convert */
                   char *str_out /* string converted */)
{
 /* this exemple assumes str_out is already allocated */
 int    i = 0;
 char   c; /* one_character */

 c = str_in[i];             /* first character get */
 while (c != NULL)          /* while string is not finished */
 {
  if (c < 'A')              /* if the char is not a letter */
    str_out[i] = c;         /* let's copy it */
  else
    if (c > 'Z')            /* if the char is already low */
      str_out[i] = c;       /* let's copy it */
    else
      str_out[i] = 'a' - 'A' + c;  /* let's make the char lower */
  i++;                      /* next character */
 }
}
```

(merci Wikipedia, article Motorola 68000)

Assembleur

- Les assembleurs contiennent en général :
  - MOVE/PUSH/POP : des instructions pour déplacer des données
  - ADD/SUB/MUL/DIV : des opérations mathématiques
  - OR/XOR/AND/NOT : des opérations logiques
  - ROT/SWITCH/SWAP : des rotations et décalages
  - TEST/CMP : des instructions mettant à jour les flags internes
  - JUMP/BRANCH/CALL : des instructions pour se déplacer dans le code avec ou sans condition
  - INT : des instructions pour déclencher des interruptions sur le processeur lui-même

Programmes

- Les programmes sont des suites d'instructions
- Le code, pour être exécuté, doit être placé en mémoire
- Sans système d'exploitation, on écrit les programmes dans des ROM (Read-Only Memory)

Code

@ mémoire	instructions
00100000	
00100000	CE56 0000
00100004	206E 0008
00100008	226E 000C
0010000C	1018
0010000E	0C00 0041
00100012	650A
00100014	0C00 005A
00100018	6204
0010001A	0600 0020
0010001E	12E0
00100020	66E8
00100022	4E5E
00100024	4E75
00100026	

(merci Wikipedia, article Motorola 68000)

Boot

- Lorsque le processeur est allumé, celui-ci démarre l'exécution du code en lisant une adresse fixée par le constructeur
  - un z80 démarre l'exécution à l'adresse 0
  - un 6502 va récupérer deux mots de 8 bits à des adresses fixées pour constituer la première adresse du code
  - etc...

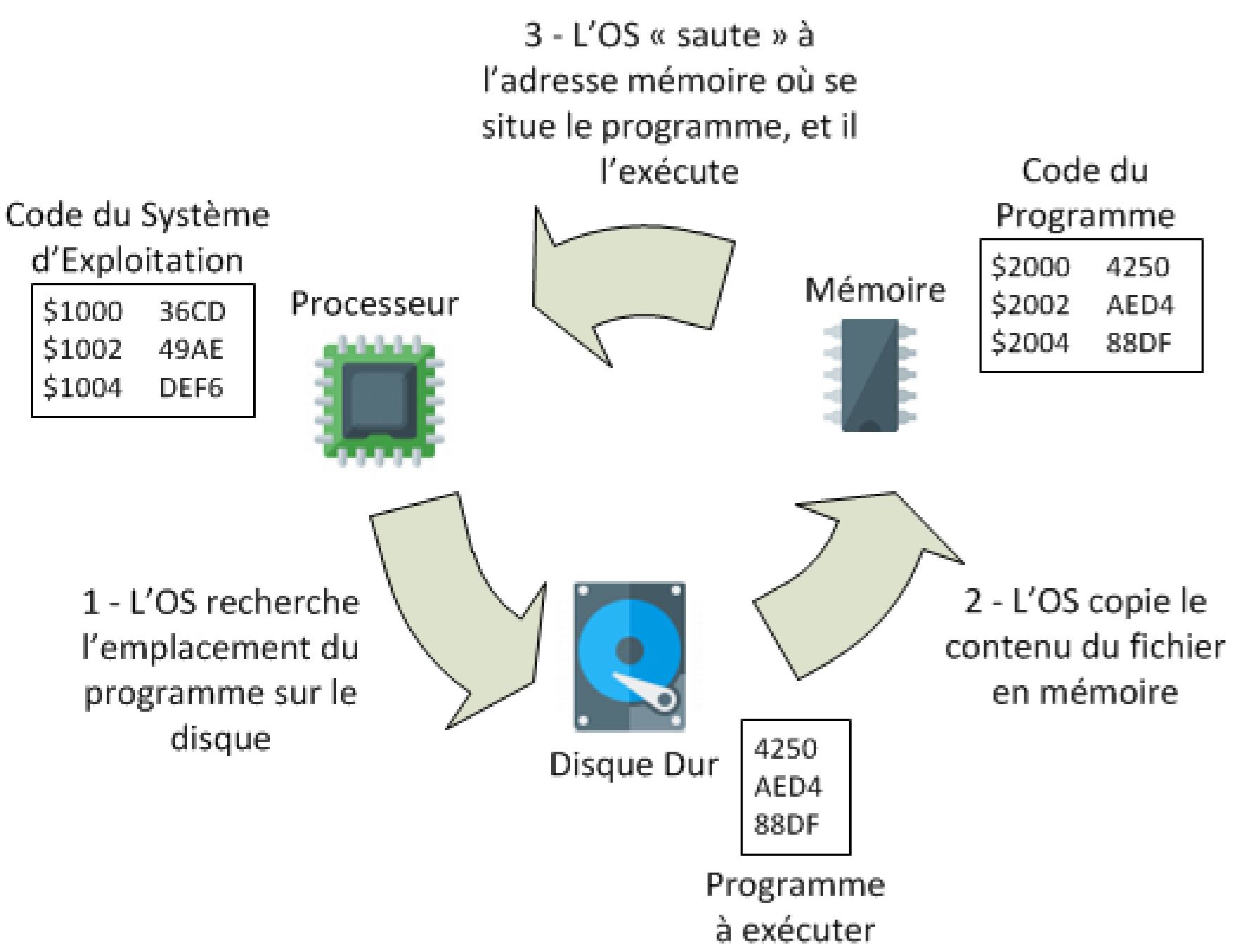
- Ce que l'on appelle le "boot", c'est la procédure où le système d'exploitation est chargé en mémoire par le processeur grâce à un programme écrit en ROM
  - dans le cas du PC, cette ROM s'appelle le BIOS ou UEFI

Chargement des Programmes

- Lorsque des développeurs écrivent des programmes, les compilent, et les exécutent, c'est au système d'exploitation de charger le code en mémoire.

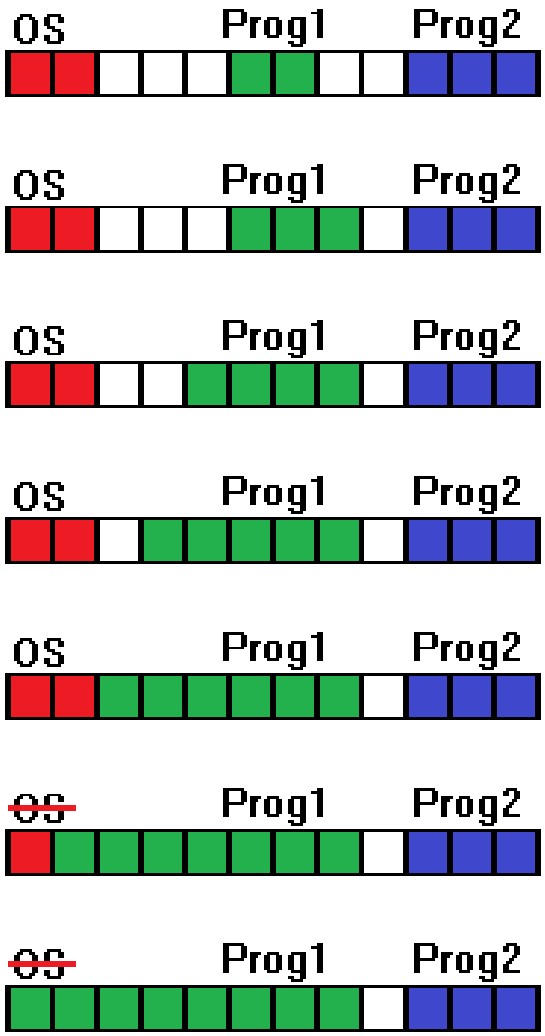
- Le rôle des compilateurs est donc de traduire le code (C, C++, ...) en langage machine dans un fichier, pour que le système d'exploitation puisse le charger en mémoire et démarrer l'exécution.

- Pour lancer un programme, le système d'exploitation va rechercher le fichier sur le périphérique de stockage, copier son contenu en mémoire, et "sauter" à l'adresse de la 1ère ligne du code compilé.


3 - L'OS « saute » à l'adresse mémoire où se situe le programme, et il l'exécute
Code du Système d'Exploitation
$1000 36CD
$1002 49AE
$1004 DEF6
Processeur
Mémoire
Code du Programme
$2000 4250
$2002 AED4
$2004 88DF
1 - L'OS recherche l'emplacement du programme sur le disque
Disque Dur
2 - L'OS copie le contenu du fichier en mémoire
4250
AED4
88DF
Programme à exécuter

Modes Utilisateur & Superviseur

- Afin d'empêcher certains programmes d'écraser d'autres programmes (comme l'OS, par exemple), des protections ont été ajoutées avec le temps :
  - interdiction d'écrire dans la mémoire utilisée par un autre programme
  - instructions réservées à l'OS
  - interdiction d'exécuter certaines parties de la mémoire
  - …



Modes Utilisateur & Superviseur

- Les processeurs sont souvent dotés de "modes" dans lesquels des protections sont actives.

- Typiquement :
  - le processeur fonctionne en mode "utilisateur" pour l'exécution classique des programmes
  - puis il passe en mode "superviseur" lorsque des fonctions de l'OS sont nécessaires.

- Le mode "utilisateur" empêche d'accéder à certaines zones mémoire, alors que le mode "superviseur" autorise tout.

Modes Utilisateur & Superviseur

- Le mode est codé dans un flag du processeur
  - toute opération nécessitant des droits va vérifier si le bit est actif ou non

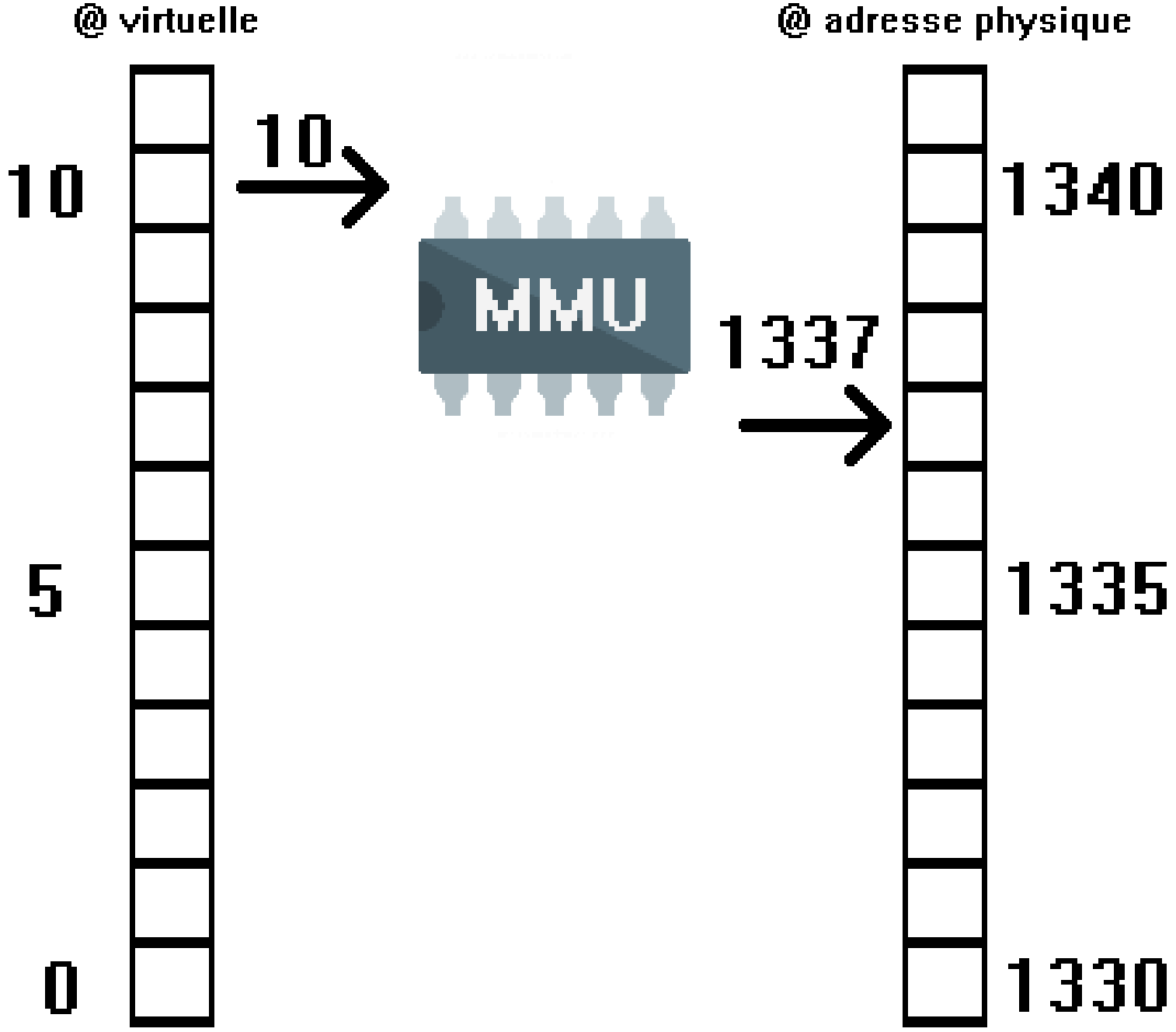
- Certaines instructions activent ou désactivent ce flag
  - Indirectement : l'instruction qui émet une interruption

Mémoire Virtuelle

- Dans un but d'extension de la mémoire, mais aussi de protection, la "mémoire virtuelle" est un mécanisme supplémentaire dans les ordinateurs.

- Au sein du code exécuté dans le processeur, les adresses utilisées (adresses virtuelles) ne correspondent pas directement à celles sur le bus d'adresse (adresses physiques).
- Une table contient la correspondance entre les adresses virtuelles et les adresses physiques : la "table de pages".

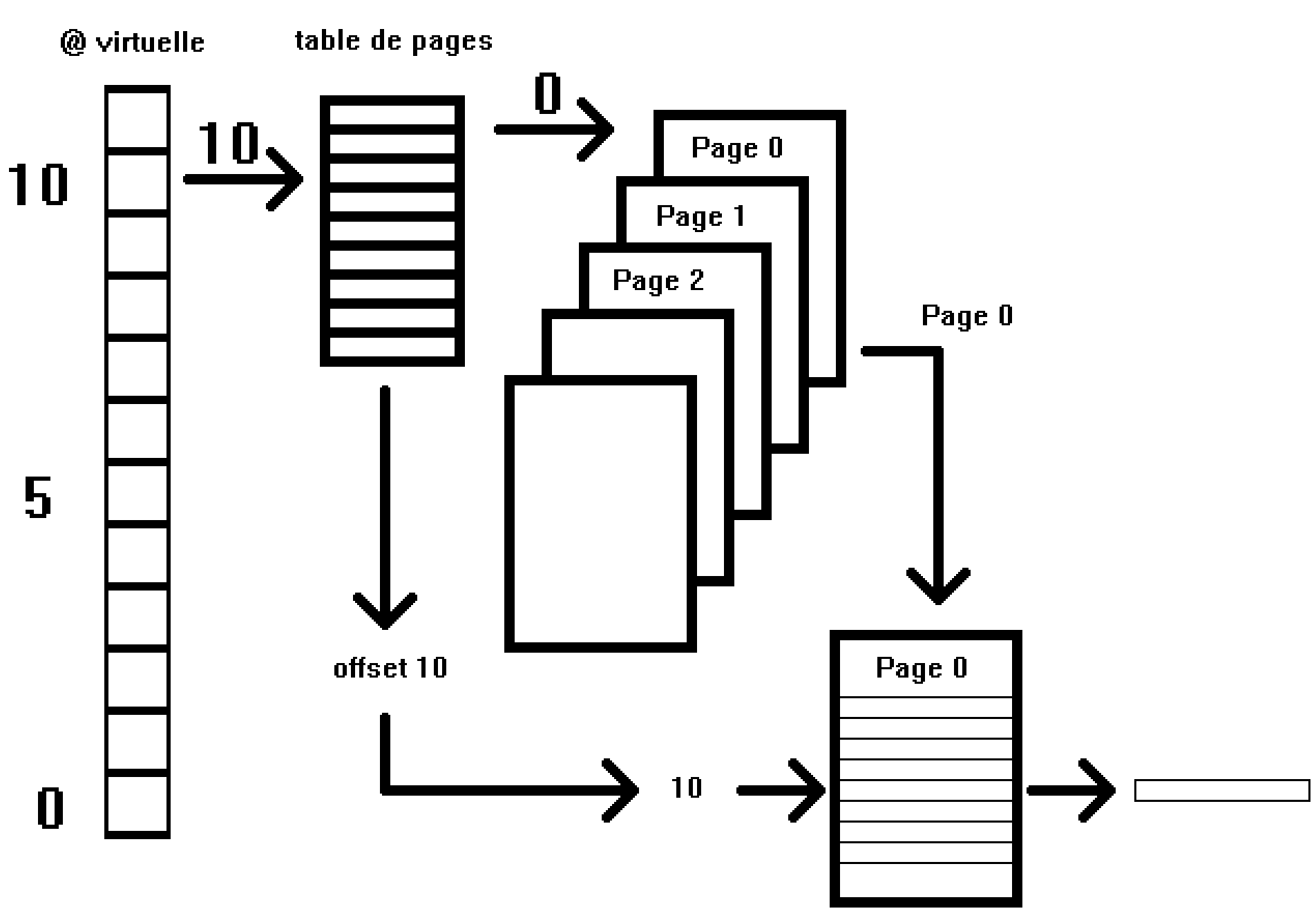
- Les adresses virtuelles sont traduites en adresses physiques grâce à la "MMU" (Memory Management Unit).


@ virtuelle
@ adresse physique
10
10
MMU
1337
1340
5
1335
0
1330

Mémoire Virtuelle

- Les adresses mémoires sont regroupées sous forme de "pages".
- En général une page contient 4Ko, par exemple de l'adresse 0 à 4095.

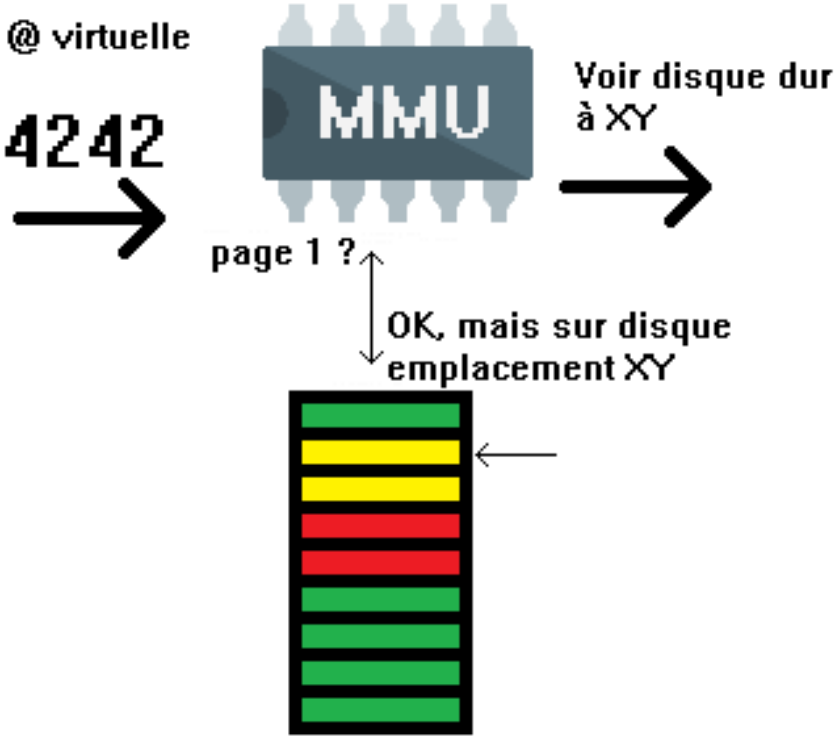
- Les adresses virtuelles encodent un numéro de page, et un décalage sur cette page.
- La MMU effectue le calcul et utilise la table de pages pour retrouver l'emplacement de la page.

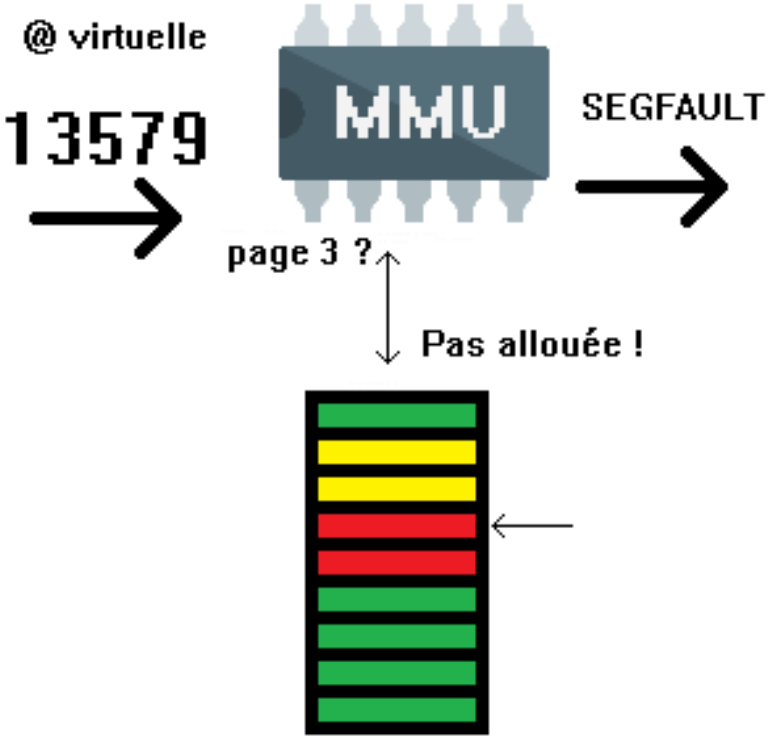

@ virtuelle
table de pages
10
10
0
Page 0
Page 1
Page 2
Page 0
5
offset 10
Page 0
10
0

Mémoire Virtuelle

- Il arrive que les pages ne soient pas retrouvées dans la mémoire physique :
  - soit la table des pages indique que la page voulue est stockée sur un disque dur
    (le "swap", et c'est à l'OS d'aller la récupérer)
  - soit la page est invalide, et une erreur sera renvoyée
    (sur Linux, le célèbre "Segmentation Fault")

Mémoire Virtuelle





Mémoire Virtuelle

- Ce mécanisme peut être utilisé pour permettre à plusieurs programmes d'utiliser l'intégralité de l'espace d'adressage, sans s'écraser mutuellement : leurs pages seront stockées sur le disque dur.

- Seul l'OS dispose des droits pour écrire dans la table des pages.
- Pour protéger les pages, des "flags" sont ajoutés dans la table.

Machine de Turing & Ordinateur

Machine abstraite composée de :	: Machine concrète composée de
Un ruban infini composé de cases (contenant un symbole chacune)	**La mémoire / L'espace d'adressage** (tableau fini de cases)
Une tête de lecture/écriture (lit/écrit dans une case du ruban à la fois)	**Un processeur** (exécute des instructions, ou lit & écrit en mémoire)
Un registre d'état (mémorise l'état courant de la machine)	**Le registre PC du processeur** (mémorise l'@ de l'instruction courante)
Une table d'actions (explique quel symbole écrire, et comment déplacer le ruban)	**L'Assembleur / Le Jeu d'Instructions** (contient l'ensemble des instructions possible)