Fonctionnement d'un ordinateur

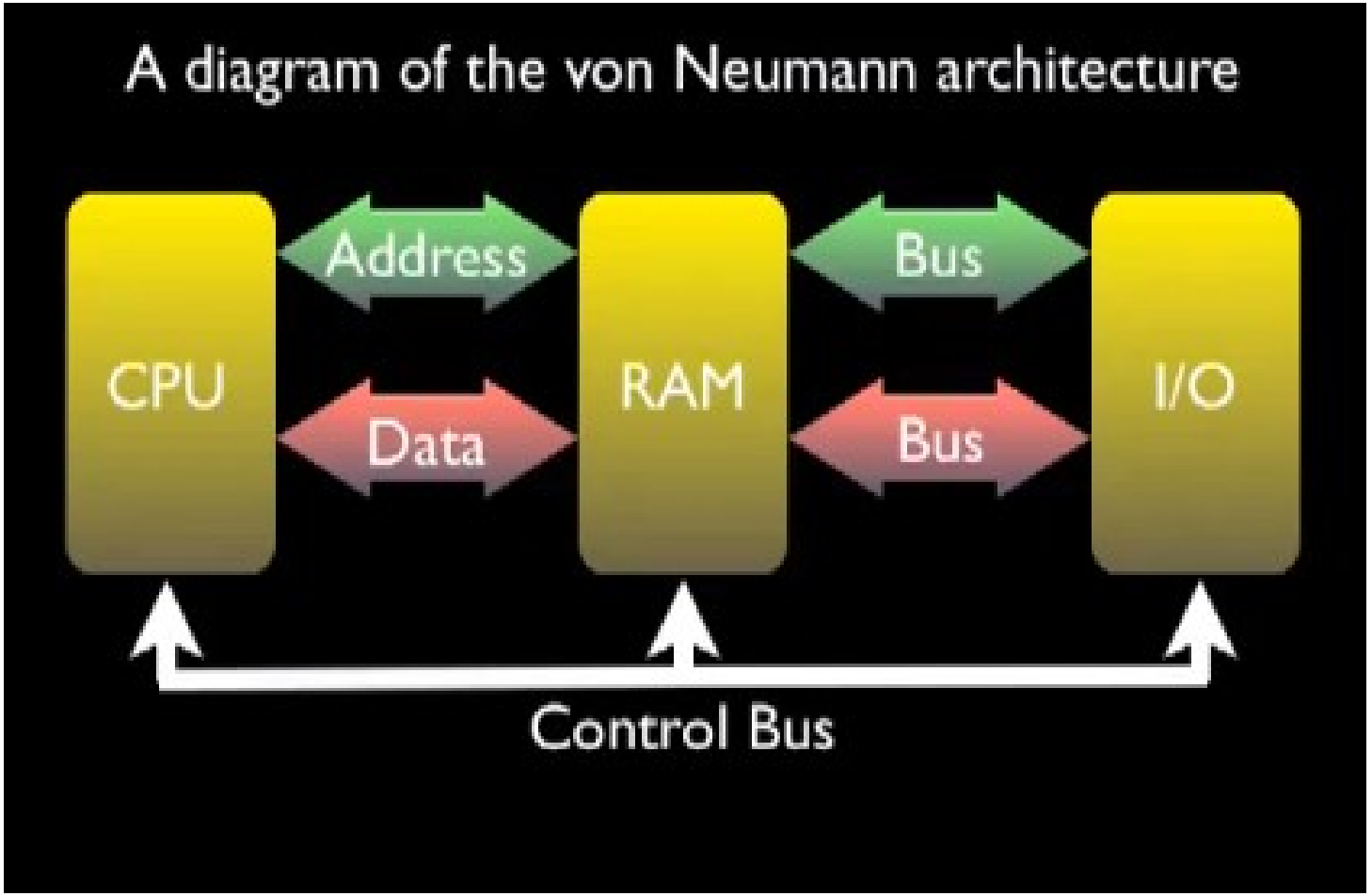
Un « ordinateur » est une machine de Von Neumann.

Une calculatrice programmable composée de :
- Une mémoire
- Un pointeur sur une liste d'instructions
- Un « processeur »
- Et des entrées / sorties

Depuis 1945 quasiment tous les ordinateurs sont construits selon ce principe,

Les composants sont reliés entre eux par des « bus »

Fonctionnement d'un ordinateur



Fonctionnement d'un ordinateur

Le « processeur » est capable de faire des calculs (addition, multiplication,...)

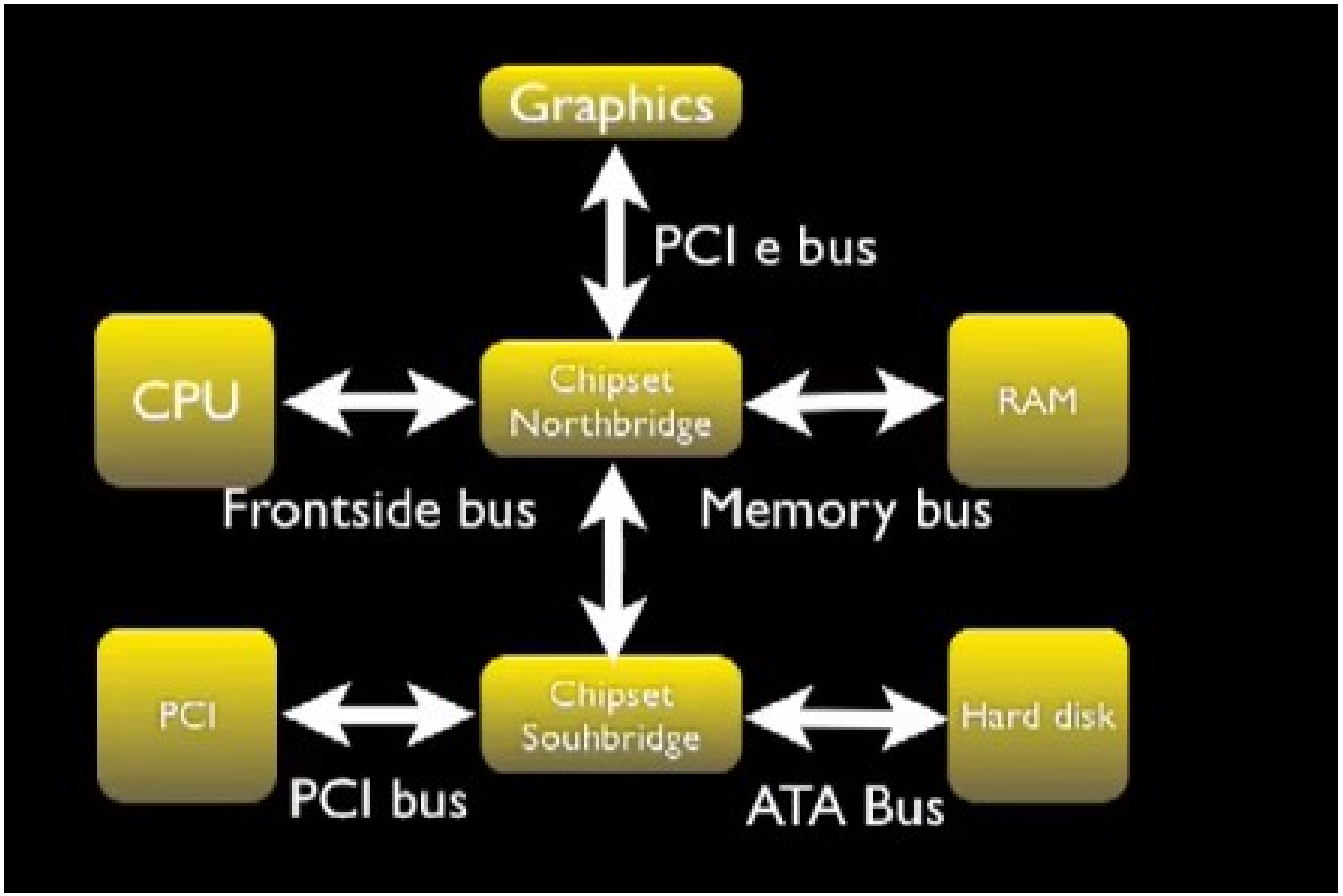
La mémoire permet de stocker des informations et de les conserver
Dans les ordinateurs modernes il y a différentes mémoires qui sont plus ou moins proches du processeur.
- Les registres (A, B, …) qui servent à effectuer les opérations
- La mémoire cache, très petite et très rapide
- La mémoire vive ou RAM, rapide et qui s'efface quand on éteint le PC
- La mémoire de stockage (disque dur, clé de stockage, bande magnétique, …)

La liste des opérations à effectuer est envoyée au processeur et un « registre » spécial sert à stocker l'instruction en cours. On peut imaginer un programme comme une liste d'opération à effectuer.

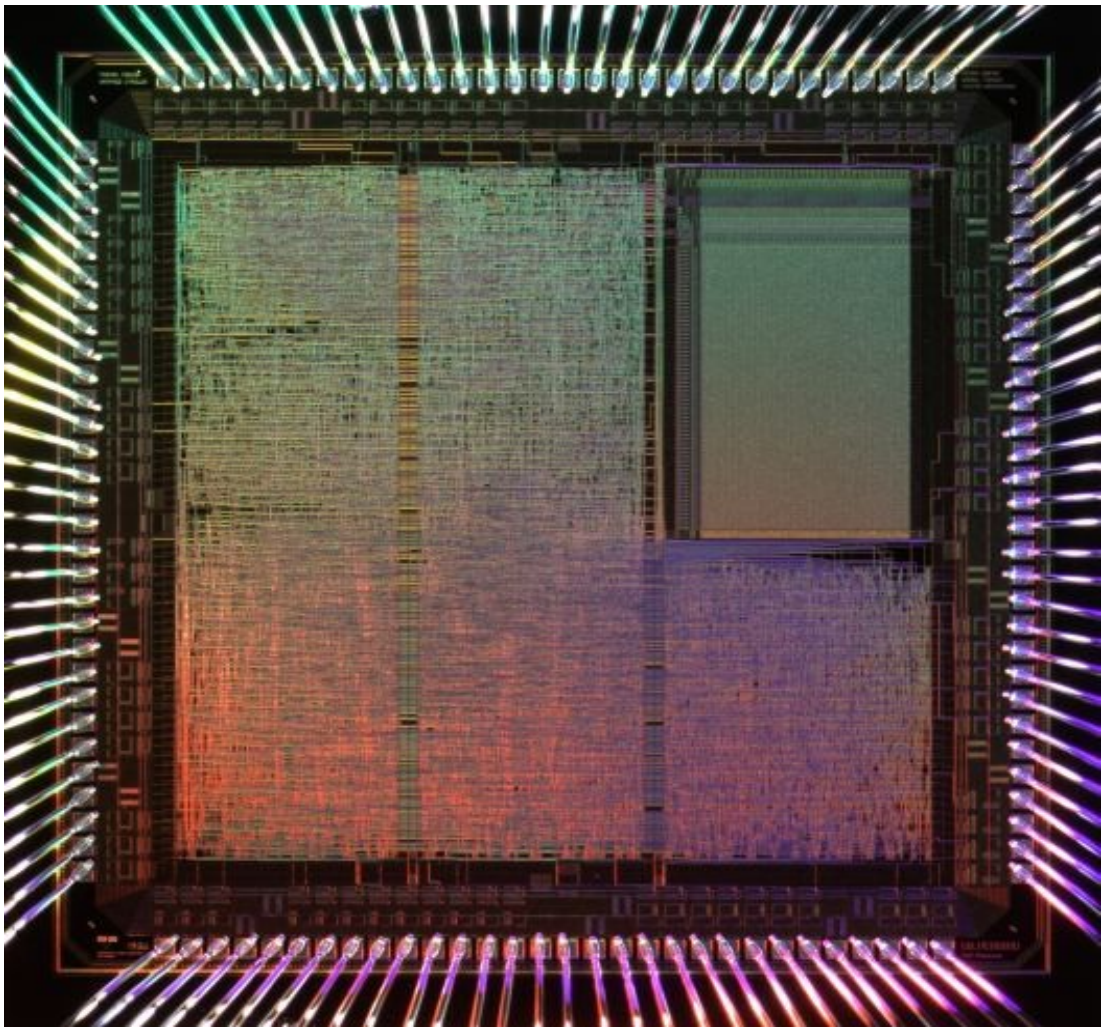
Les premiers programmes étaient représentés sous forme de cartes perforées.
Une « perforeuse » permet d'écrire des instructions ou des résultats en sortie

Fonctionnement d'un ordinateur

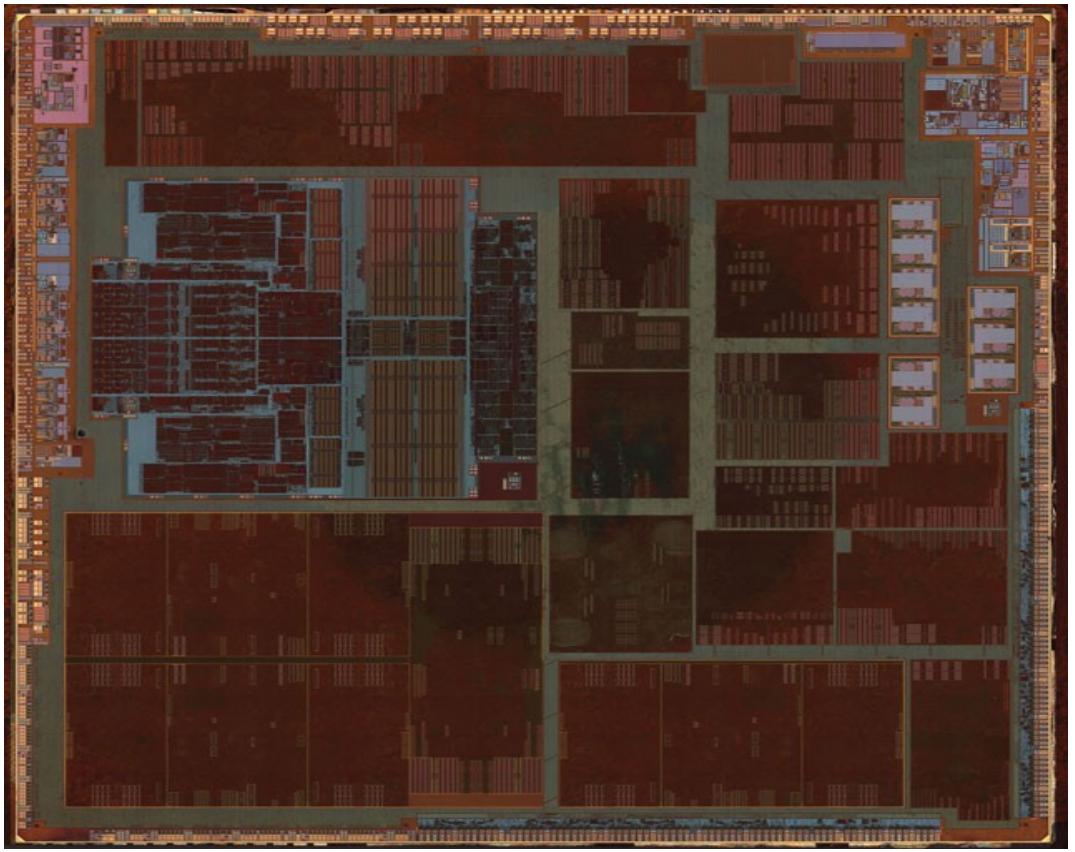
PC Moderne :



Processeur



IPhone 5



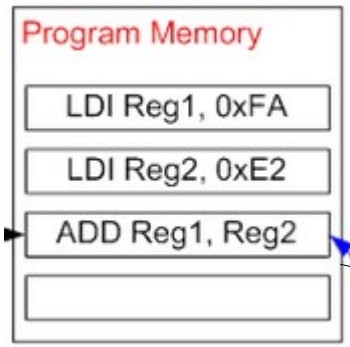
Processeur

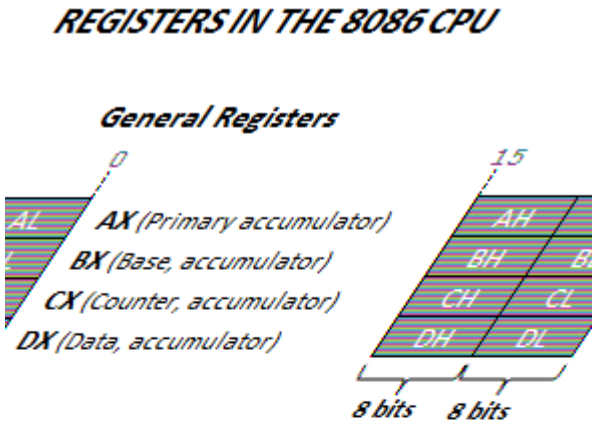
Registres de données
A, B, C, D...

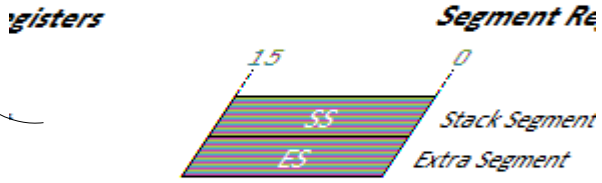
Registre d'instructions
I

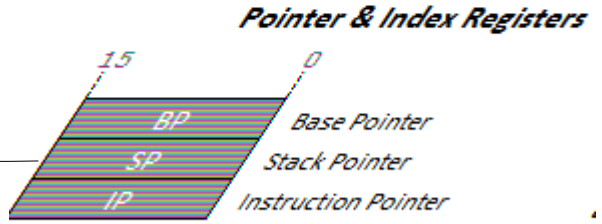
Registre pointant sur la mémoire
SP

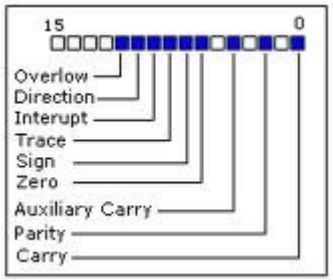
Taille de SP
---
16 bits = 640 Ko
32 bits = 6 Go
64 bits = 16 exabytes












640K
ought to be enough
for anybody
- Bill Gates?
MICROSOFT

Liste de codes d'instruction

RISC vs CISC

Assembleur

```
writeln:                # l'argument est dans a0
        li  $v0, 1      # le numéro de print_int
        syscall         # appel système
        li  $v0, 4      # la primitive print_string
        la  $a0, nl     # la chaîne "\n"
        syscall
        j   $ra         # saut à l'adresse ra
read:
        li  $v0, 5
        syscall
        j   $ra


fib:
        li $v1, 1
        li $v0, 1
        li $t0, 2
loop:
        bgt  $t0, $a0, fibout
        move $t1, $v0
        add  $v0, $v0, $v1
        move $v1, $t1
        add  $t0, $t0, 1
        j loop
fibout:
        j $ra


.globl  __start
__start:
        jal  read
        move    $a0, $v0
        jal     fib
        move    $a0, $v0
        jal     writeln
```

Binaire (assembleur compilé)

001000 00000 01000 00000 00000 101100 584 ADDI R8, R0, #44
001000 00000 00110 00000 00000 000001 588 ADDI R6, R0, #1
101011 00000 00110 00000 01010 111100 592 SW R6, 700(R0)
001000 00000 00111 00000 00000 000100 596 ADDI R7, R0, #4
101011 00111 00110 00000 01010 111100 600 SW R6, 700(R7)
001000 00000 01010 00000 00000 001000 604 ADDI R10, R0, #8
001000 01010 00011 11111 11111 111000 608 ADDI R3, R10, #-8
100011 00011 00001 00000 01010 111100 612 LW R1, 700(R3)
001000 01010 00100 11111 11111 111100 616 ADDI R4, R10, #-4
100011 00100 00010 00000 01010 111100 620 LW R2, 700(R4)
000000 00001 00010 00101 00000 100000 624 ADD R5, R1, R2
101011 01010 00101 00000 01010 111100 628 SW R5, 700(R10)
001000 01010 01010 00000 00000 000100 632 ADDI R10, R10, #4
000100 01010 01000 00000 00000 000001 636 BEQ R10, R8, #4
000010 00000 00000 00000 00010 011010 640 J #616
000000 00000 00000 00000 00000 000000 644 NOP
000000 00000 00000 00000 00000 001101 648 BREAK

C (itératif)

```
{
	int n, i;
	double a, b, c;
	c = 0;
	b = 1;

	printf ("Calcul de la suite de Fibonacci.\n");
	printf ("Entrez N\n");
	scanf ("%d", &n);

	if (n==0 || n==1)
	{
		printf ("n = %d", n);
	}
	else {
		for (i=2;i<=n;i++) {
			a = b + c;
			c = b;
			b = a;
		}
	}
	printf ("F = %lf\n", a);
}
```

JAVA (récusif)

```
public long fibonacci(int n) {
        if (n <= 1) {
            return n;
        } else {
            return fibonacci(n-1) + fibonacci(n-2);
       }
}
```