Architecture élémentaire

Jean-Luc Dekeyser
Jean-luc.dekeyser@lifl.fr

S3- 2010/11

12 séances

1. Un cours d'architecture pour des informaticiens
2. De l'électronique à l'informatique
3. Du binaire à l'information
4. Notion de circuits logiques
5. Un modèle d'exécution : Von Neumann
6. Comment mémoriser une donnée?
7. Comment réaliser un calcul?
8. Instruction et ordonnancement
9. Communication entre mémoire et processeur
10. Communication avec l'extérieur
11. Architecture et logiciel
12. Quoi de neuf dans mon PC?

Cours TD TP

cours	TD	TP
1 introduction	Nombres en informatique	
2 elctronique à l informatique	Portes logiques	Hyades Portes
3 données informatique	circuits combinatoires	additionneur hades
4 Architecture Von Neumann	logique sequentielle	feux hades
5 instrcution machine	langage assembleur	feux hades suite
6 contrôle continu	programmation asssembleur	Hades contrôle
7 micro progrtamation	micro programation	init nexys2
8 Fonction mémoire	mémoire	additionneur nexys2
9 Mémoire virtuelle	mémoire virtuelle	simulation Isim
10 Bus	bus	feux schematic nexys 2
11 I/O	I/o	controle nexys2
12 architecture evoluée	revision examen	memoire hades
		sceance de rattrapage

Un cours d'architecture pour des informaticiens

Sans architecture pas d'informatique!

Comprendre comment çà marche permet de mieux utiliser

Demain le hardware deviendra aussi programmable

Évaluation

- Contrôle continu
  - test 1 - un lundi en cours (début novembre) la 6ieme séances
  - test 2 - durant la session d'examen
  - test 3 - deux TP notés + pointage
    - Les TD et les TP commencent après le second cours
  - Les test 1 et 2 sont avec documents

Le bouquin à lire!!!

- **Computer Organization and Architecture 8th Edition**
- **http://williamstallings.com/COA/COA7e.html**
- http://www.filecrop.com/William-Stallings-Computer-Organization-and-Architecture.html

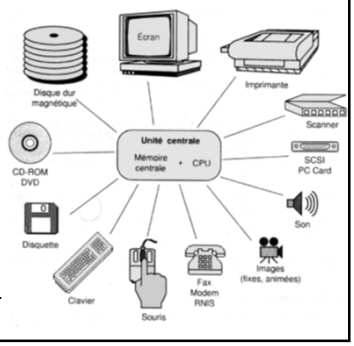
À la fin de cette unité

- vous aurez pris connaissance de l'évolution phénoménale de l'ordinateur depuis ses débuts.
- vous connaîtrez les différents systèmes de numération/codification utilisés de nos jours : binaire, décimal;
- vous comprendrez pourquoi on utilise encore le binaire en informatique;
- vous connaîtrez les composantes essentielles de la *Machine de von Neumann*.
- Vous verrez que le parallélisme est sous jacent à tout traitement informatique

7

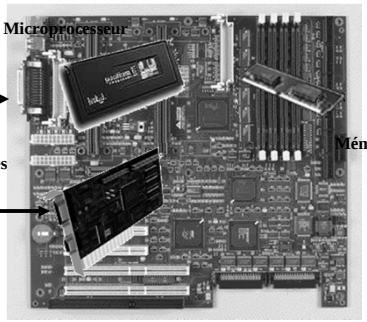
Principaux éléments d'un ordinateur

- Unité centrale
- Cédérom ou DVD
- Disque dur
- Disquette
- Clavier
- Souris
- Écran ou projecteur
- Modem
- Scanner
- Carte de son
- Images et vidéo
- Fax modem
- Ports SCSI, USB, etc.



9

Vue coté architecture…



Parlons chiffre!

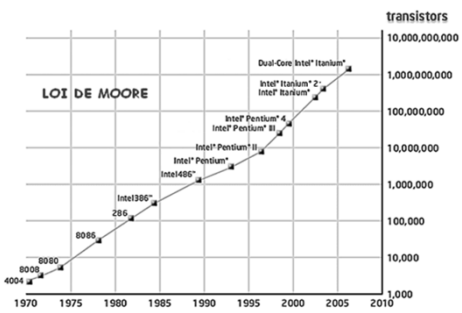
- Unités de mesure de capacité

Nom	Symbole	Puissances binaires et valeurs en décimal	Nombre	Hexa	Ordre de grandeur SI décimal
unité	o/B	$2^0 = 1$	un(e)	1	$10^0 = 1$
kilo	ko/Ko kB/KB	$2^{10} = 1\,024$	mille	400	$10^3 = 1\,000$
méga	Mo/MB	$2^{20} = 1\,048\,576$	million	100000	$10^6 = 1\,000\,000$
giga	Go/GB	$2^{30} = 1\,073\,741\,824$	milliard	40000000	$10^9 = 1\,000\,000\,000$
téra	To/TB	$2^{40} = 1\,099\,511\,627\,776$	billion	10000000000	$10^{12} = 1\,000\,000\,000\,000$
péta	Po/PB	$2^{50} = 1\,125\,899\,906\,842\,624$	billiard	400000000000	$10^{15} = 1\,000\,000\,000\,000\,000$
exa	Eo/EB	$2^{60} = 1\,152\,921\,504\,606\,846\,976$	trillion	10000000000000000	$10^{18} = 1\,000\,000\,000\,000\,000\,000$

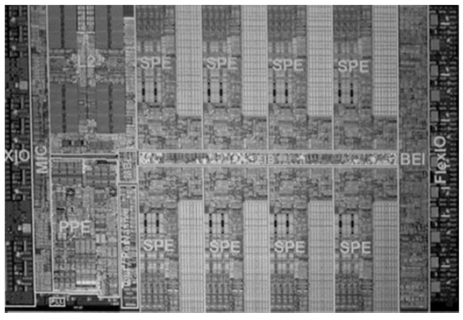
- Unités de mesure de temps
  - ms = milliseconde = 10^{-3} s = 0,001 s
  - µs = microseconde = 10^{-6} s = 0,000 0001 s
  - ns = nanoseconde = 10^{-9} s = 0,000 000 001 s
  - ps = picoseconde = 10^{-12} s = 0,000 000 000 001 s

10

Toujours plus…

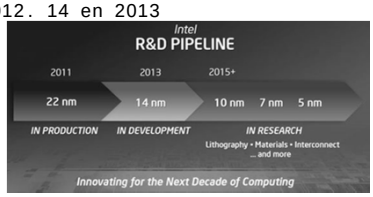


Le Cell

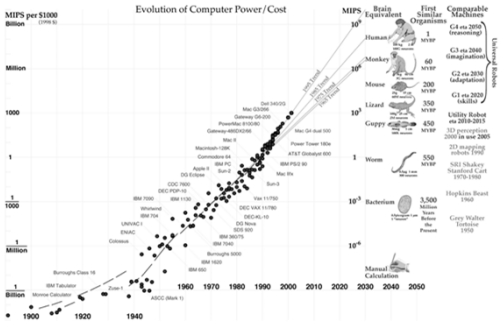


Dynamique de l'industrie du Semiconducteur

- **de la gravure horizontale et verticale dite en 3D**
  - relance la loi de Moore
- 2009: Intel 32nm, 2010: 28nm , çà continu 22 en 2012. 14 en 2013



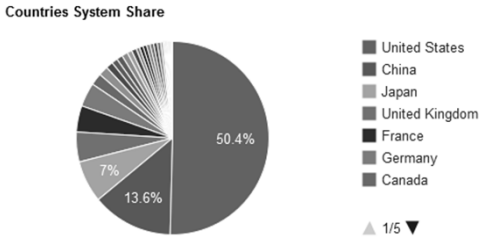
Prix/performance



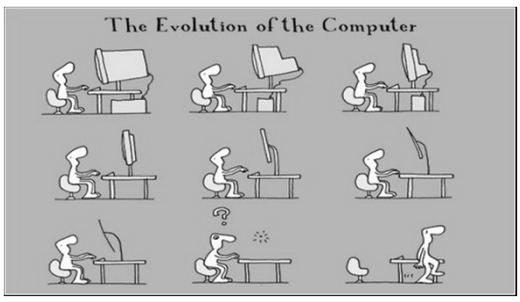
Ce qu'il y a de mieux aujourd'hui

	NAME	SPECS	SITE	COUNTRY	CORES	R_{max} Pflop/s
1	Sequoia	IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect	DOE / NNSA / LLNL	USA	1,572,864	16.33
2	K computer	Fujitsu SPARC64 VIIIfx 2.0GHz, Tofu interconnect	RIKEN AICS	Japan	705,024	10.51
3	Mira	IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect	DOE / SC / ANL	USA	786,432	8.153
4	SuperMUC	IBM iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband QDR	Leibniz Rechenzentrum	Germany	147,456	2.897
5	Tianhe-1A	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050	NUDT/NSCC/Tianjin	China	186,368	2.566

Un signe de puissance mondiale

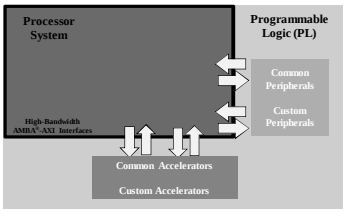


Et l'encombrement…



Les nouveau circuits programmable : FPGA

- *field-programmable gate array*
- Circuit intégré logique qui peut être reprogrammé après sa fabrication.

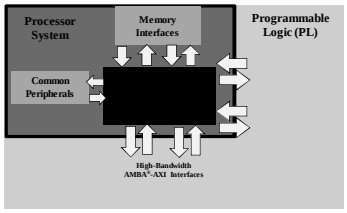


Change the program
Customize I/O interfaces
Add compute accelerators

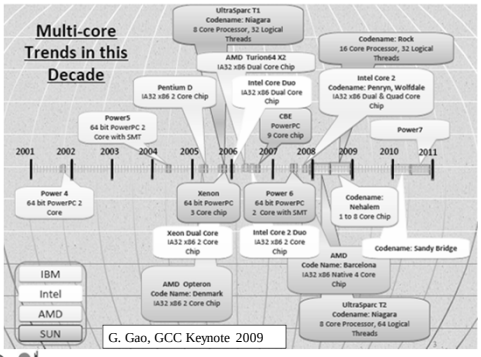
*PL = Programmable Logic

Vers les systèmes sur puce

Boots like a Processor
Acts like a Processor
Really … is a Processor
… and more



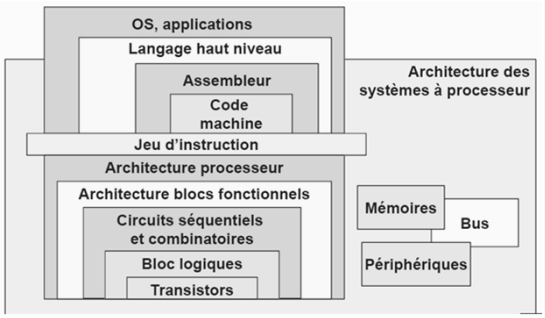
La tendance Multi- et Many-Core



Vue informaticienne de l'architecture

- Point de vue concepteur d'architecture
  - Définir des modèles d'exécution
  - Les valider : test preuve simulation prototypage
  - Les programmer : langages compilateurs
- Point de vue du programmeur
  - Comprendre le fonctionnement
  - Optimiser ces algorithmes
  - Critère de cout/performance
  - Réutiliser ces codes

Niveaux d'abstraction

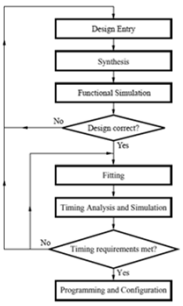


Vers la synthèse

- **Synthétiser** cette description matérielle pour obtenir un composant réalisant les fonctions désirées, à l'aide d'éléments logiques concrets (portes logiques, bascules ou registres).
- Après la synthèse viennent les phases de :
  - placement : on choisit l'emplacement physique des différents éléments ;
  - routage : on détermine les connexions entre éléments.
- prendre en compte les ressources disponibles telle la surface …

00fff587e7fe 00242cb565f9
0022fbae933e 0023ae3a0cab

Méthode de conception



Langage de modélisation : VHDL

- VHDL pour modéliser un système
- Modéliser quoi?
  - Spécification des besoins
  - Documentation
  - Test par simulation
  - Verification formelle
  - Synthèse
- Quel gain?
  - Processus de conceoption plus flexible, réduire les coût et donc les temps de conception
  - Augmenter la fiabilité de la conception

Basic VHDL Concepts

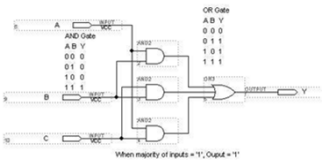
- Le langage VHDL a été commandé par le Département de la Défense des États-Unis.
- VHDL reprend la même syntaxe que celle utilisée par le langage Ada
  - Interfaces
  - Behavior
  - Structure
  - Test Benches
  - Analysis, elaboration, simulation
  - Synthesis

A Sample Model

Description

```
library ieee;
use ieee.std_logic_1164.all;

entity majconc is
   port ( A, B, C : in std_logic;
          Y: out std_logic
        );
end majconc;

ARCHITECTURE a of majconc is

begin

Y <= (A and B) or (A and C) or (B and C);
end a;
```

Implementation



La suite…

- Comment faire des portes logiques en éléctronique?
- Comment associer des portes logiques pour faire un circuits.. Un processeur????

A lire

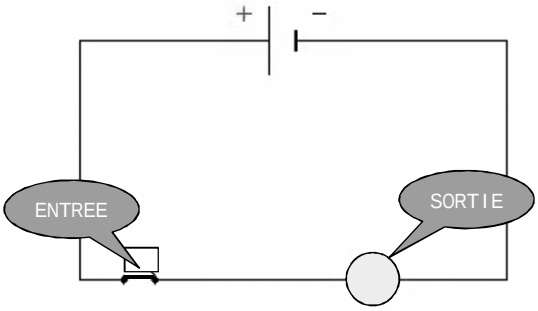
- http://www.commentcamarche.net/contents/histoire/ordinateur.php3
- http://fr.wikipedia.org/wiki/Histoire_de_l%27informatique
- http://fr.wikipedia.org/wiki/VHDL
- http://fr.wikipedia.org/wiki/Loi_de_moore

- Un livre en ligne http://www.scribd.com/doc/47783341/Architecture-de-l'ordinateur

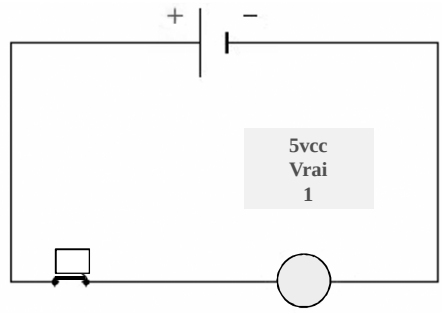
AE Cours 2

De l'électronique à l'informatique

Au commencement était l'electricité



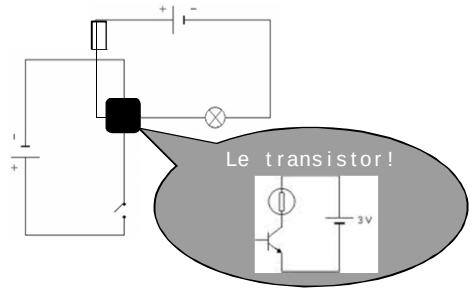
Au commencement était l'electricité



Fausses idées

- NON il n'y a pas des 0 et des 1 dans un ordinateur!
- Juste des électrons qui se baladent

Un interrupteur programmable!!



Mon premier « ordinateur »

- Un inverseur!
  - Si j envoie un courant rien ne sort
  - Si je n'envoie rien alors il sort un courant!

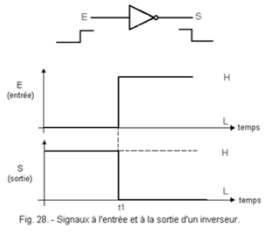
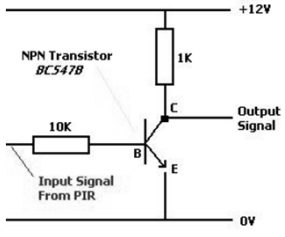
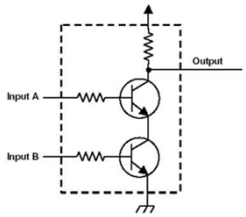


Fig. 28. - Signaux à l'entrée et à la sortie d'un inverseur.

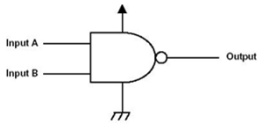


Avec deux entrées en séries

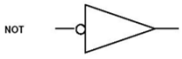


NAND:

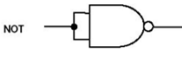
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



Inverseur à base de Nand

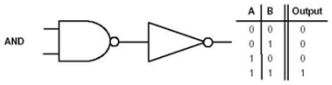


Input	Output
0	1
1	0

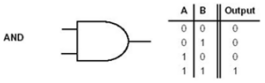


Input	Output
0	1
1	0

Un And

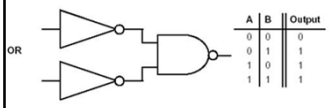


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

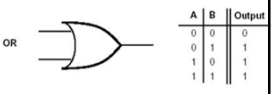


A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

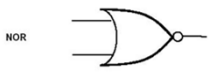
Le Or à partir du Nand



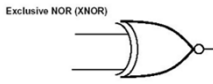
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

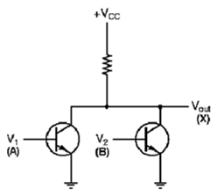


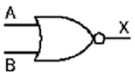
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Le Nor en transistor

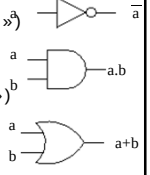




A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

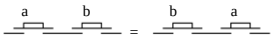
Algèbre de Boole

- Pour pouvoir manipuler des 0 et des 1, on a donc trois opérations :
  - Fonction **négation** (complémentation) **« NON »** (« NOT »)
    - noté avec une barre
    - $\overline{0} = 1$ et $\overline{1} = 0$
  - Fonction **conjonction « ET »** (« AND »)
    - noté « . »
    - 0.0 = 0.1 = 1.0 = 0 1.1 = 1
  - Fonction **disjonction « OU »** (« OR »)
    - noté « + »
    - 0+0 = 0 0+1 = 1+0 = 1+1 = 1

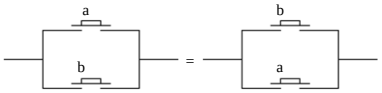


Axiomes de base (1/4)

- Commutativité :
  - **a.b = b.a**

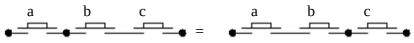


  - **a + b = b + a**

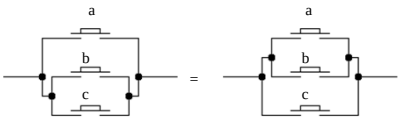


Axiomes de base (2/4)

- Associativité
  - **a.(b.c) = (a.b).c**

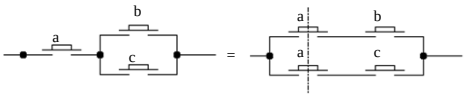


  - **a + (b + c) = (a + b) + c**

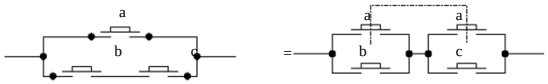


Axiomes de base (3/4)

- Distributivité :
  - **a.(b + c) = (a.b) + (a.c)**

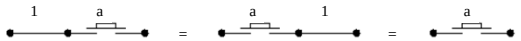


  - **a + (b.c) = (a + b).(a + c)**

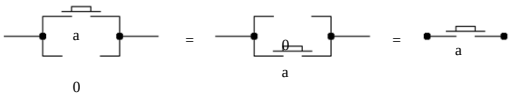


Axiomes de base (4/4)

- Éléments neutres
  - **1.a = a.1 = a**



  - **0 + a = a + 0 = a**

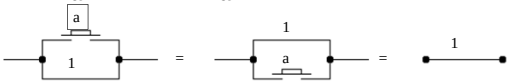


- Complément **$\overline{a}$.a = 0** **$\overline{a}$ + a = 1**

Propriétés (1/2)

- Élément absorbant :
  - **a.0 = 0.a = 0**

  - **a + 1 = 1 + a = 1**

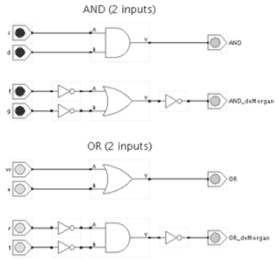


- Absorption :
  - **a.(a + b) = a** **a + (a.b) = a**

Propriétés (2/2)

- Idempotence :
  - **a.a = a** **a + a = a**
- Involution :
  - **$\overline{\overline{a}}$ = a**
- Théorème de De Morgan :
  - **$\overline{a.b} = \overline{a} + \overline{b}$** **$\overline{a + b} = \overline{a}.\overline{b}$**

De Morgan « graphiquement »



Un AND3 en VHDL

```
entity AND_3 is
  port(
    e1 : in  bit;
    e2 : in  bit;
    e3 : in  bit;
    s  : out bit
    );
end entity;

architecture arc of AND_3 is

begin  -- arc

  s <= e1 and e2 and e3;

end arc;
```

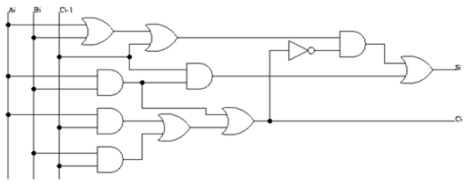
Additionneur élémentaire

- La table de vérité d 'un additionneur élémentaire est

Ai	Bi	Ci-1	Ci	Si
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

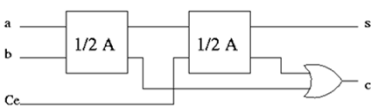
- le nombre de 1 (en entrée) est impair ⇔ Si=1
  - Si = Ai ⊕ Bi ⊕ Ci-1
- le nombre de 1 est supérieur (strictement) à 1 ⇔ Ci=1
  - Ci = AiBi + AiCi-1 + BiCi-1

Additionneur élémentaire (2/2)



C'est aussi….

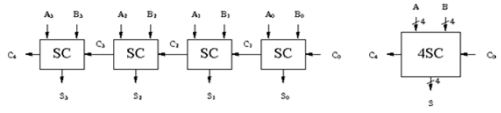
Deux demi-additionneurs



Et en VHDL alors?

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity full_add1 is
  port (
    a, b, cin : in  std_logic;
    s, cout   : out std_logic;
    );
end entity;

architecture arc of full_add1 is

  signal resultat : unsigned(1 downto 0);

begin

  resultat <= ('0' & a) + ('0' & b) + ('0' & cin);
  s        <= resultat(0);
  cout     <= resultat(1);

end arc;
```

Add 4 Bits en cascade



Pour un ADD 4 bits

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity adder4 is
  port (
    a, b : in  unsigned(3 downto 0);
    s    : out unsigned(3 downto 0);
    cout : out std_logic
    );
end entity;

architecture arc of adder4 is

  component full_add1 is
    port
      (a, b, cin : in  std_logic;
       s, cout   : out std_logic);
  end component full_add1;

  signal c : unsigned(2 downto 0);

begin

  inst_full_add0 : full_add1 port map (a(0), b(0), '0' , s(0), c(0));
  inst_full_add1 : full_add1 port map (a(1), b(1), c(0), s(1), c(1));
  inst_full_add2 : full_add1 port map (a(2), b(2), c(1), s(2), c(2));
  inst_full_add7 : full_add1 port map (a(3), b(3), c(2), s(3), cout);

end arc;
```

A lire

- http://mpicartier.free.fr/ancien_site/electricite/transistor/transit.htm
- Pour un point de vue électronique : http://www-lemm.univ-lille1.fr/physique/physicie/lec12.htm
- Un cours VHDL : http://comelec.enst.fr/hdl/vhdl_intro.html

AE cours 3

Du binaire à l'information
-
Quelques cicuits élémentaires

Objectifs

- Connaître les circuits élémentaires :
  - Fonctions arithmétiques : additionneurs, comparateurs
  - Fonctions combinatoires : codeurs, décodeurs, multiplexeurs, démultiplexeurs
- Eléments de circuits séquentiels

Du binaire à l'information

TOUT EST DIT DANS LE COURS « CODAGE »

A partir d'un bit….

- Fermé = 1 = le courant passe
- Ouvert = 0 = le courant ne passe pas

- Un bit (**BI**nary dig**IT**) peut prendre la valeur 0 ou 1 et permet de représenter 2 informations
  - Présent/absent
  - Ecoute/dort
  - Homme/femme
  - Droite/gauche
  - Pour/contre
  - ……

Et si on associait plusieurs bits?

Plusieurs Bits

- 2 bits = 00 01 10 11 soit 4 informations différentes
  - Nord, Sud , Est, Ouest
- 3 bits = 000 001 010 011 100 101 110 111 ,soit 8 informations

- N bits = 2^n informations différentes
- On parle de mots:
  - 8 bits (octets)
  - 16, 32, 64, 128 …

- L'utilisateur (ou le constructeur) donne une signification à la valeur d'un mot

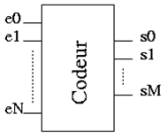
Exemples de nombres signés

	M&S	Cà1	Cà2
011	3	3	3
010	2	2	2
001	1	1	1
000	0	0	0
111	-3	-0	-1
110	-2	-1	-2
101	-1	-2	-3
100	-0	-3	-4

A SUIVRE DONC…

Codeurs

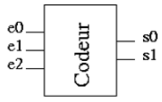
- Code une entrée à N entrée où il n 'y a qu 'un seul 1 en binaire.
  - $M = \log_2(N+1) - 1$
  - « codeur N+1 vers M+1 »



Codeurs

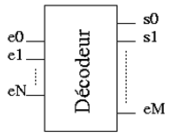
- Codeur 3 vers 2 : N=2, M=1

e2	e1	e0	s0	s1
0	0	0	*	*
0	0	1	0	0
0	1	0	0	1
0	1	1	*	*
1	0	0	1	0
1	0	1	*	*
1	1	0	*	*
1	1	1	*	*



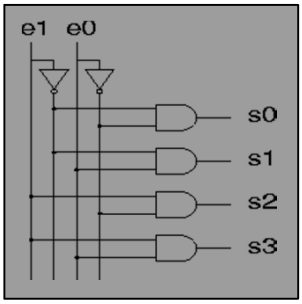
Décodeurs

- Décode un mot binaire
  - $2^{N-1} \leq M < 2^N$
  - sJ = 1 ssi $(eN…e1e0)_2 = J$
  - « décodeur N+1 vers M+1 »
- Exemple :
  - N=1 M=2
  - « décodeur 2 vers 3 »



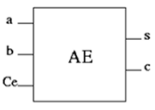
e1	e0	s0	s1	s2
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	0	0	0

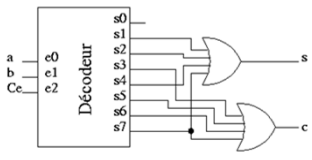
Décodeur 2 vers 4



Décodeurs

- Toute fonction logique f(x0, x1, … , xN) peut être réalisée avec un décodeur.
- Exemple l 'additionneur élémentaire :
  - s = f1(a, b, Ce) = Σ(1, 2, 4, 7)
  - c = f2(a, b, Ce) = Σ(3, 5, 6, 7)

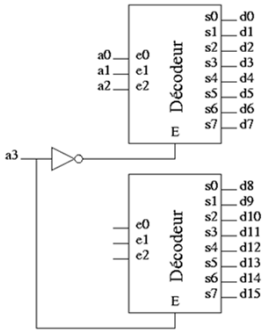




Décodeurs

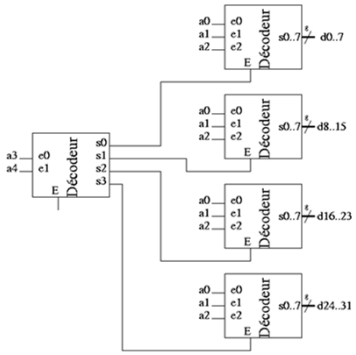
- On ajoute une entrée « Enabled »
  - si E = 0 alors toutes les sorties sont à 0
  - permet de construire de plus « gros » décodeurs...

Décodeur 4 vers 16 à partir de décodeur 3 vers 8



Décodeurs

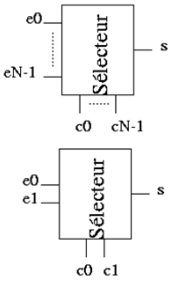
- Décodeur 5 vers 32 avec des décodeurs 3 vers 8 (et un 2 vers 4)...



14

Sélecteurs

- Un sélecteur prend N entrées (e0..N-1) et N commandes (c0..N-1) et copie l 'entrée J si CJ est la seule à être à 1.
  - S = 1 ssi EJ = 1 et CJ est la seule commande à 1
- Exemple à 2 entrées :

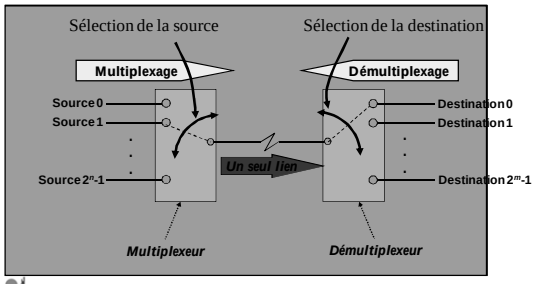


c0	c1	s
0	0	*
0	1	e1
1	0	e0
1	1	*

15

Multiplexeurs et Démultiplexeurs

- Utilisation :

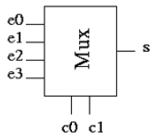


16

Multiplexeurs

- Le multiplexeur (**MUX**) sélectionne, à l'aide de N entrées de commande (c0..n-1), **une** des 2^N entrées d'information (e0..2^N-1)et la dirige à la sortie.
  - La sortie est égale à 1 ssi l 'entrée $(c0c1\ldots cN-1)_2$ est vraie.
- Exemple: Multiplexeur 4 à 1.

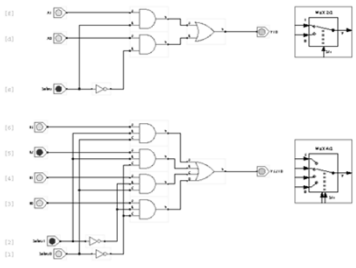
c0	c1	s
0	0	e0
0	1	e1
1	0	e2
1	1	e3



- On utilise un *décodeur* et un *sélecteur*...

17

Inside…

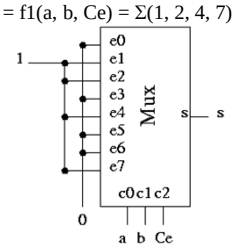


Multiplexeurs

- Synthèse de fonctions logiques avec un multiplexeur
- Exemple de l 'AE… *(encore !)*

a	b	Ce	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

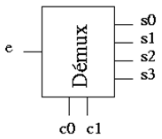
$s = f1(a, b, Ce) = \Sigma(1, 2, 4, 7)$



Démultiplexeurs

- Opérateur inverse du multiplexeur. Il distribue le bit e reçu en entrée unique vers l'une des 2^N sorties possibles *D* (N: nombre d'entrées de sélection S).
- Exemple multiplexeur 4 sorties :

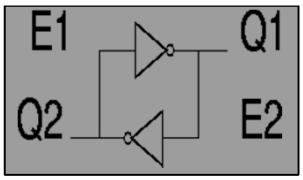
c0	c1	s0	s1	s2	s3
0	0	e	0	0	0
0	1	0	e	0	0
1	0	0	0	e	0
1	1	0	0	0	e



- C 'est un décodeur avec e comme « enabled ».

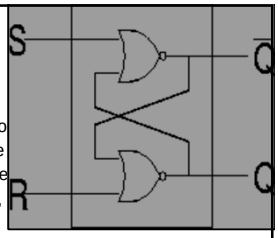
Notion d'état stable

- Bistable = 2 portes NON montées en opposition.

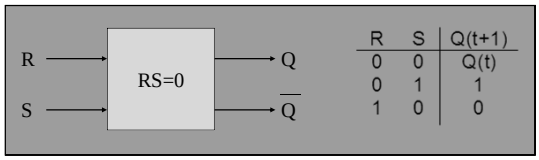


- Ce circuit a 2 états *stables* différents :
- $Q_1 = 0$ et $Q_2 = 1$
- $Q_1 = 1$ et $Q_2 = 0$

Bascules, registres et mémoires



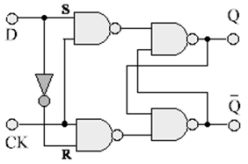
- Bascule (bistable/flip-flo
  - = moyen de mémoriser de
- Différents type de bascule
  - RS=0, RS=0, RST, D, JK,
- Ex. Bascule $\overline{R}S=0$

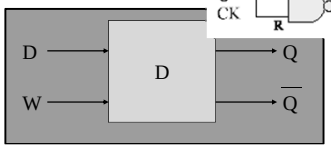


R	S	Q(t+1)
0	0	Q(t)
0	1	1
1	0	0

Bascules, registres et mémoires

- Ex. Bascule D: Aussi appelé





- La donnée stockée copie D si W=1 sinon, l 'état de la bascule est inchangée
- Une bascule RS avec un inverseur entre les entrées S et R
  - élimine le problème du cas d'indétermination. Si S = 1 donc on a R = 0 et inversement. Ni S=1 et R= 1, ni S=0 et R=0.

- http://tk5yp.fr/elec/elec.htm
- http://www.info.univ-tours.fr/~marcel/archi/node63.html

AE cours

Un modèle d'exécution
Von Neumann

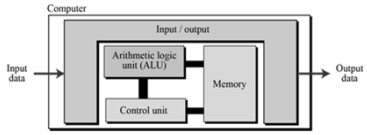
Introduction

De quoi traite l'Informatique

- **Computer Science**
  compute = calculer
- **Informatique = Information + Automatique**
  traitement automatique de l'information

Modèle de Von Neumann

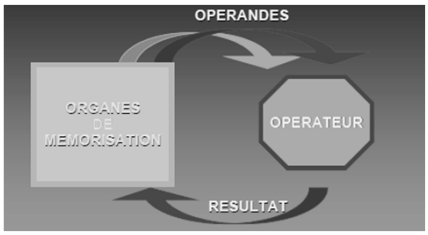
- **John Von Neumann 1943-45**
  - **Lignes essentielles pour construire une machine électronique. (Projet ENIAC)**
  - **Appliqués jusqu'à nos jours**
- **Quatre blocs fonctionnels:**
  - **Le processeur (ALU + Control Unit)**
  - **La mémoire**
  - **Le bus**
  - **Lesd I/O**



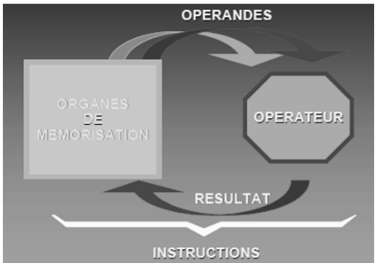
Du câblé au programmé..

- Premiers ordinateurs les pas d'exécution du programme étaient directement câblés dans le circuit
  - Aujourd'hui on parlerait d'accélérateur matériel
- On invite le programme vu comme des données stockées dans la mémoire principale
  - l'architecture de Von Neumann
- La fonction de l'unité de contrôle est de
  - lire le programme de la mémoire
  - décoder les instructions
  - commander leur exécution
- Un changement de programme se fait par une simple réécriture de la mémoire

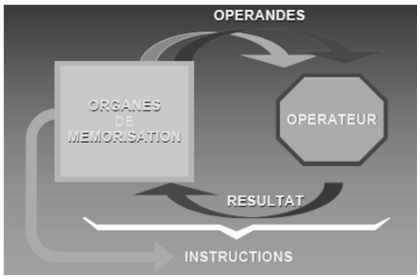
Du calcul sur des données…



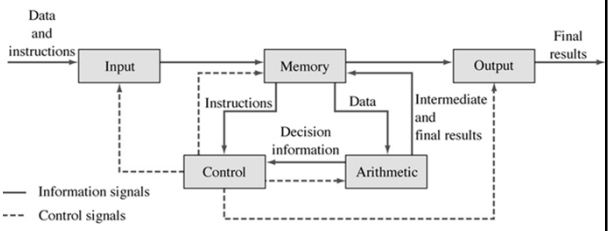
Notion d'instruction



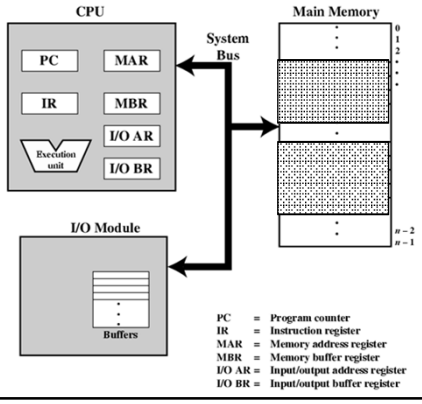
Mémoire banalisée



Quels échanges dans la machine?

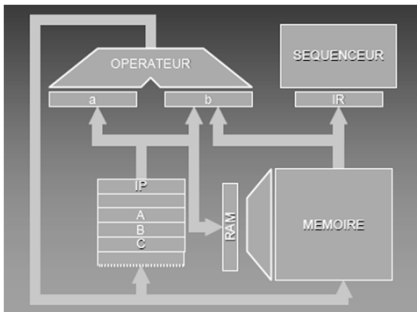


Zoom inside

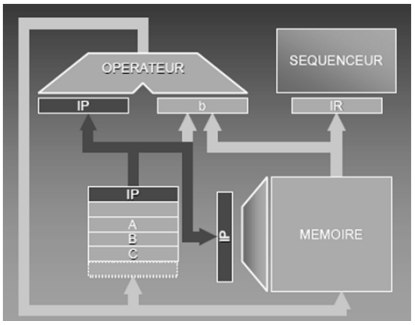


Un ordinateur simplifié…

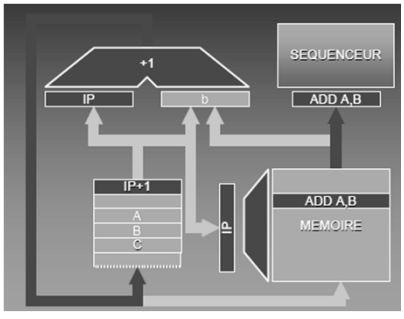
IP ~
CO~
CP~
PC



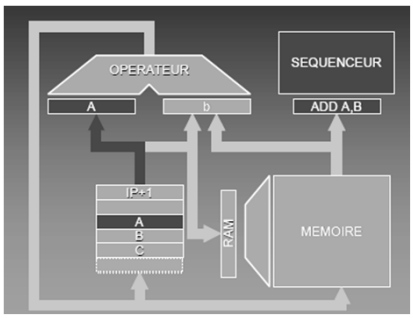
Lecture de l'intsrtcution



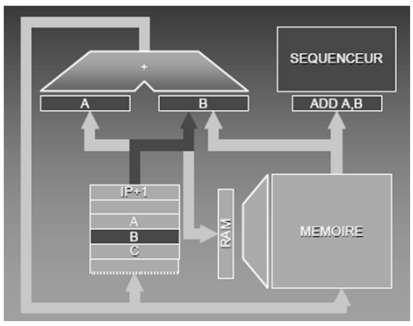
L'instruction arrive
on prépare la suite…



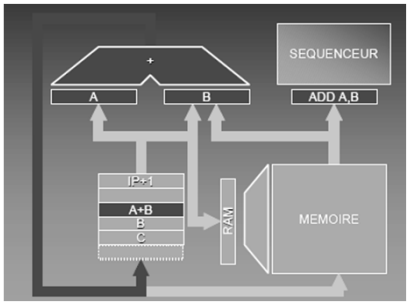
Les opérandes arrivent!



Prêt pour le calcul!!!



On range le résultat ☺



Et si on en exécute plusieurs?

- **Le processeur exécute (interprète) les instructions élémentaires à la suite**
- **Une séquence d'opérations peut décrire tous les problèmes**

notion de Programme

16

Déroulement du programme

- **Le déroulement du programme est contrôlé par le Compteur Ordinal qui pointe vers la prochaine instruction à exécuter.**
  - **La séquentialité est intrinsèque au modèle VN ☹**
- **Les instructions sont exécutées en séquence sauf en cas de saut. (Jump)**

17

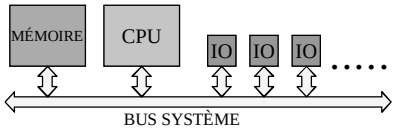
Les données

- **La mémoire conserve les données et les instructions, on parle de mémoire banalisée.**
- **Les instructions sont amenées une à une vers le processeur**
- **Les échanges entre mémoire / processeur se font via le système de communication : souvent le bus**

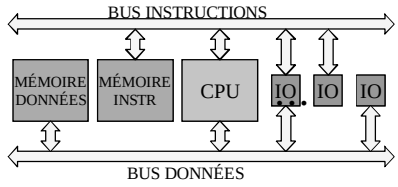
18

Von Neumann vs. Harvard

- Von Neumann



- Harvard



Séquencement des instructions

- **L'exécution d'une instruction passe par plusieurs étapes successives, chacune étant considérée comme une micro-opération.**
- **Certaines de ces actions correspondent à une activité mémoire, d'autres à une activité processeur.**
- **Pour effectuer une instruction, il faut toujours effectuer les actions suivantes:**

20

Séquencement des instructions

1. **Aller chercher l'instruction en mémoire;**
2. **Calculer l'adresse de la prochaine instruction, incrémenter le compteur ordinal**
3. **Décoder le code de l'opération**
4. **Calculer les adresses des opérandes si nécessaire**
5. **Extraire les opérandes éventuelles de la mémoire**
6. **Exécuter l'instruction**
7. **Calculer l'adresse du résultat**
8. **Ranger le résultat en mémoire**

21

Phases 1 2 3

- 1: Lit instruction suivante
  - Bus **Adresse** ← PC
  - Bus **Commande** ← « Lire instruction »
  - RI ← Bus **Donnée**
- 2: Incrémente compteur ordinal
  - PC ← PC + taille(RI)
- 3 :Décode Instruction *ex: Add A,(123)*
  - *A ← A + contenu @123.*

Département Informatique 22

Phases 4 5

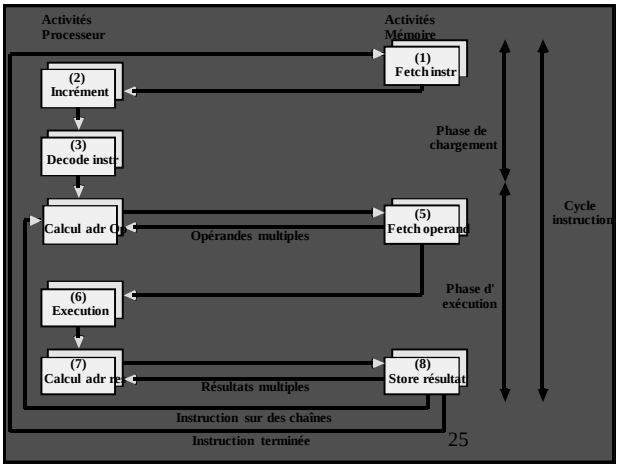
- 4 et 5 : Lit données (facultatif)
  - Bus **A** ← 123
  - Bus **C** ← « lire donnée »
  - tmp ← Bus **D**
- Transfert données → UAL
  - UAL.1 ← A
  - UAL.2 ← tmp
  - UAL.RI ← « addition »

Département Informatique 23

Phases 6 7 8

- 6: UAL calcule opération
  - Activation de l'additionneur intégral
  - → tmp
- 7 et 8 : UC range résultat
  - A ← tmp
- Recommence
  - Lit & Exécute instruction suivante
- Pas de repos pour un processeur…

Département Informatique 24

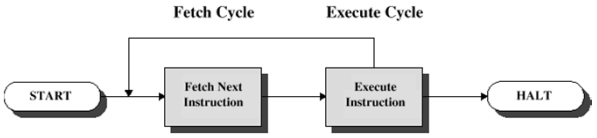


Différentes classes d'instruction

Instructions UAL	*Instructions Mémoire*	*Instructions Branchement*
Lecture instruction	Lecture instruction	Lecture instruction
Incrémentation CP	Incrémentation CP	Incrémentation CP
Décodage de l'instruction	Décodage de l'instruction	Décodage de l'instruction
Lecture des opérandes	Calcul de l'adresse mémoire	Calcul de l'adresse de branchement
Exécution	Accès mémoire	Exécution
Ecriture du résultat	Rangement du résultat	

Cycle du processeur

- **L'exécution d'une instruction peut être découpée en plusieurs phases successives.**
- **Deux phases sont définies: la phase de chargement 'Fetch' et la phase d'exécution 'Execute''**



27

AE cours

Instructions machines

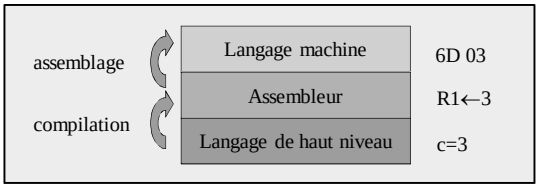
Objectifs

- Voir les instructions élémentaires du microprocesseur
- Comment on les réalise à l'intérieur du µP…
- Comment on contrôle le flot des instructions
  - Branchement
  - Sous routines
  - Passage de paramètres

2

Du langage à la machine

- Programme = suite d'instructions



- Trois types d'instructions
  - De rangement
  - De calcul
  - De branchement

3

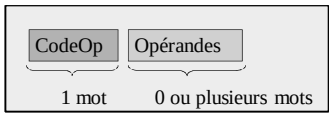
Exemples d'instructions machine

R1←4	MOV R1,4 LD R1,4	Mettre la valeur 4 dans le registre R1
R1←R2	MOV R1,R2 LD R1,R2	Mettre la valeur de R2 dans le registre R1
R1←MEM(1515)	MOV R1,(1515)	Copie la valeur stockée en mémoire à l'@ 1515 dans R1
R1←R1+2	ADD R1,2	Additionne 2 à la valeur de R1 et range dans R1
R1←R1+1	INC R1	Incrémente de 1 la valeur de R1
R2←R2-R4	SUB R2,R4	Soustraie à R2 la valeur de R4 et range dans R2

4

Codage des instructions

- Structure :



- On représente les instructions par des chaines de caractères => ce sont des chaines de bits!!!
  - MOV R1, 4 2 mots = 89 04
  - ADD R2, 3 2 mots = B2 03

5

Codage des instructions

- MOV R1, 4
  - 89 04
  - 1000 1001 0000 0100
  - 10001 001 0000 0100
  - MOV R1 4
- ADD R2, 3
  - B2 03
  - 1011 0010 0000 0011
  - 10110 010 0000 0011
  - ADD R2 3

6

Instructions en mémoire

- Mémoire :
  - 1515 = 89
  - 1516 = 04
  - 1517 = B2
  - 1518 = 03

Combien d'opérandes?

- 3 opérandes

- 2 opérandes

- 1 opérandes

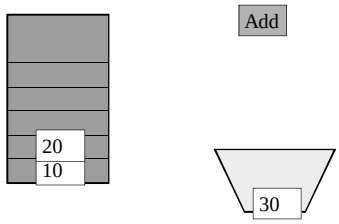
- 0 opérande

Exemples d'instruction

- ADD R2, R3, R4 — R3 + R4 => R2
- ADD R2, R3 — R2 + R3 => R2
- ADD R2 — Acc + R2 => Acc
- ADD — Sur la pile

Les machines à pile

- On utilise une pile pour ranger les opérandes
- On passe en notation post-fixée (Polonaise)



Modes d'adressage

- Valeur immédiate
  - MOV R1, 3E
- Adressage direct registre
  - MOV R1, R2
- Adressage direct mémoire
  - MOV R1, (1515)
- Adressage indirect registre
  - MOV R1, (R2)
- Adressage indirect indexé
  - MOV R1, (R3, R2)

D'autres adressages

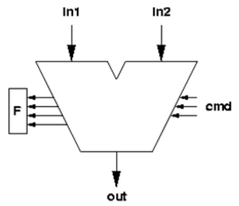
- Post incrémenté
  - MOV R1, (R4)+
- Pré décrémenté
  - MOV R1, -(R4)
- Avec déplacement
  - MOV R1, 10(R1)
- Indirect indexé avec déplacement
  - MOV R1, 10(R2, R3)

Instructions de branchement

- Trois types de branchement
  - Branchements simples
  - Branchements conditionnels
  - Branchements sous-routines
- Deux types de références
  - Absolu ou relatif
  - En relatif, on ne donne pas une adresse mais un déplacement…
- Branchements simples :
  - JMP 1789 ou JP 1789
  - C'est équivalent à un MOV dans PC

Instructions de branchement

- Branchement conditionnels
  - Sur quoi porte la condition ?
  - Lorsque l'on fait une opération arithmétique et logique, l'ALU positionne un registre de flags !



Instructions de branchement

- Chaque bit du registre F est un « flag »
  - Ex :
    - Z le résultat est zéro
    - N le résultat est négatif
    - V l'opération a engendré un dépassement de capacité
    - Etc.
- La condition de saut porte sur les flags
  - JZ 1789
    - Saute à l'adresse 1789 si Z est positionné

Exemple 1

- En C :

```
if ( cpt == 10 )
  cpt = 0;
else
  cpt++;
```

- La variable cpt est soit en mémoire soit dans un registre (ici on suppose que c'est dans R3).

```
1515: CMP R3, 0A
1517: JNZ 1525   ; partie else
1520: MOV R3, 00
1522: JMP 1526   ; suite du programme
1525: INC R3
1526: …
```


Exemple 2

- En C :

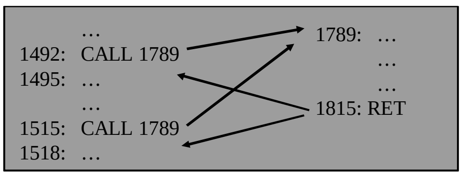
```
for ( i=0 ; i<15 ; i++ )
  a[i] = a[i] + i;
```

- i est dans le registre R1, a est un tableau de bytes dont l'adresse de début est dans R2

```
1515: MOV R1, 00       ; initialisation boucle
1517: MOV R3, R2
1518: CMP R1, 0F       ; début boucle
1520: JGE 1531         ; codé sur 3 octets
1523:
1524:  ADD (R3), R1    ; a[i] = a[i] + i
1525:
1526: INC R1
1527: INC R3
1528: JMP 1518
1531: …
```


Sous-routines

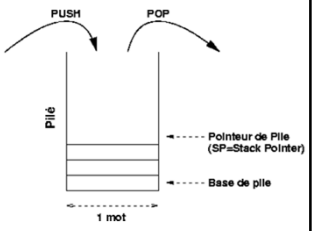
- Instruction CALL ou JSR
  - Ce sont des instructions de branchement



  - Pb. Comment savoir si l'on revient en 1495 ou 1518 ?
  - Quels sont les registres que l'on peut utiliser dans la sous-routines ?

Adresse de retour

- On utilise une pile
  - = LIFO
  - Deux instructions :
    - push
    - pop
  - On utilise un registre pointeur de pile SP
    - push → on décrémente SP
    - pop → on incrémente SP



Adresse de retour

- Après la lecture de l'instruction CALL, PC contient l'adresse de retour
- Lors du CALL :
  - On « pushe » PC
  - On met l'adresse de la sous-routine dans PC
- Lors du RET
  - On « pope » l'adresse de retour et on la met dans PC

La sous-routine doit rendre la pile « propre », *i.e.* autant de push que de pop…

Registres utilisables

- Pour les registres…
  - La sous-routine sauvegarde dans la pile les valeurs des registres qu'elle va utiliser

```
1789: push R1
      push R2
      push R5
      mov R1, 5
      …
      pop R5
      pop R2
      pop R1
1815: ret
```

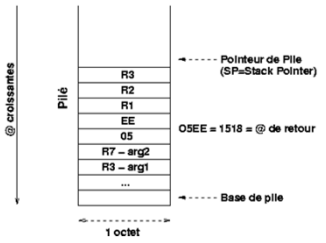

Passage de paramètres

- Deux possibilités
  - Par registres
    - Fenêtrage de registres
  - Par la pile

```
        push R3     ; argument 1
        push R7     ; argument 2
1515:   call 1789
        pop R7                      1789: push R1
        pop R3                            push R2
                                          push R3
                                          …
                                          ret
```

Obligatoire pour rendre la pile « propre »

  - Argument 1 et 2 accessibles depuis SP



AE cours

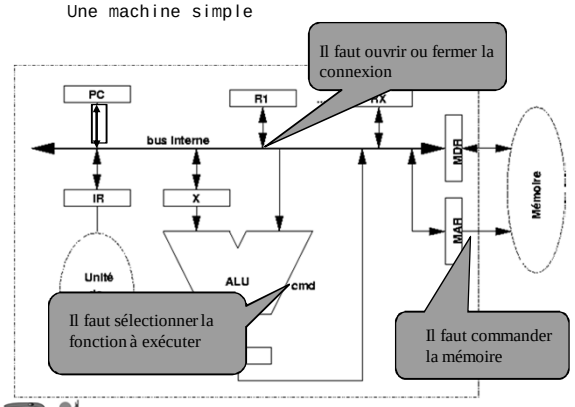
De l'instruction à la micro programmation

Objectifs

- Une vue globale du déroulement d'une instruction
- Du RI vers l'exécution
  - Décodages d'instruction
  - séquencement des micro-opérations
  - l'exécution des micro-opérations
    - Notion de signaux/commandes
    - Micro architecture

2

Une machine simple



Phase de chargement

- Phase 1 – Chargement de l'instruction
  - PC pointe vers la case mémoire contenant la prochaine instruction à exécuter: ADD R1, 4
    - Ici codé sur 2 mots : ADD R1 et 4
  - Mettre PC dans MAR et donner l'ordre de lecture
    - PCout – MARin (En même temps!!!)
    - Read – (WaitMemory)
  - Placer la valeur de MDR dans IR pour que l'instruction soit décodée
    - MDRout – Irin (En même temps!!!)

4

Phase de chargement

- Phase 1 (suite)
  - On profite du fait que la lecture en mémoire est lente pour incrémenter PC
  - PC pointe vers l'emplacement mémoire suivant
  - « code » de la phase 1 :
    - PCout – MARin – Xin
    - Read – INCX
    - Yout – PCin – (WaitMemory)
    - MDRout – IRin
- Prêt pour décoder !!!! ADD R1 est dans le RI

5

Phase d'exécution

- Phase 2 – décodage de l'instruction
  - Pris en charge par l'UC (unité de contrôle)
- Phase 3 – exécution
  - Deux sous-phases :
    - 3.1 Récupérer les arguments éventuellement
    - 3.2 Exécution

6

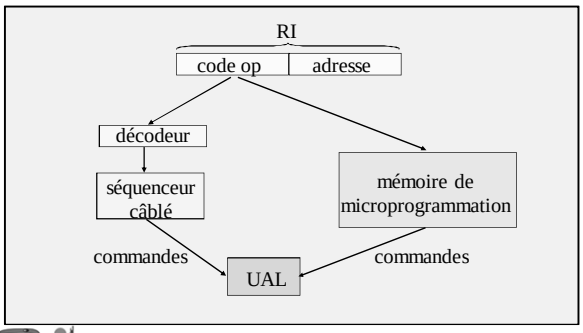
Phase d'exécution

- 3.1 lecture argument + incrémentation PC
  - PCout – MARin – Xin
  - Read – INCX
  - Yout – PCin – (WaitMemory)
- 3.2 exécuter l'instruction (ici ADD R1,4)
  - R1out – Xin
  - MDRout – ADD
  - Yout – R1in

Décodage et contrôle

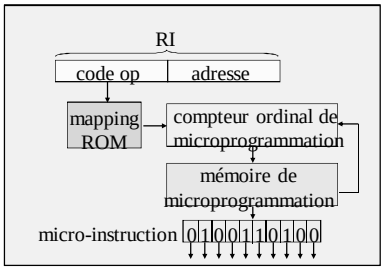
- Comment sont réalisées les Unités de Contrôle dans les microprocesseurs ?
- Automates d'états finis
  - Circuits logique combinatoire
    - Prise en compte des signaux de contrôle et registre condition
    - codeur multiplexeur etc..
  - Micro architecture
    - Micro programmation
    - Mémoire morte chaque case contient les signaux à activer à chaque top d'horloge

Du RI aux commandes ou signaux



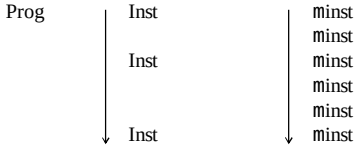
Le modèle avec mémoire morte

- Le microprogramme peut être stocké dans une ROM ou une EPROM.



Instruction vs Microinstruction

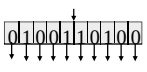
- Les instructions machines(macro-instructions) sont interprétées par le micro-séquenceur qui exécute le microcode.
- Plusieurs micro-instructions sont exécutées séquentiellement à chaque top d'horloge pour réaliser une (macro-)instruction



Microprogrammation « horizontale »

Chaque ligne de la mémoire de microprogramme est une micro-instruction

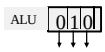
- Chaque bit du mot est relié à un signal de contrôle



- Toutes les combinaisons sont possibles mais çà prend beaucoup de place

Microprogrammation « verticale »

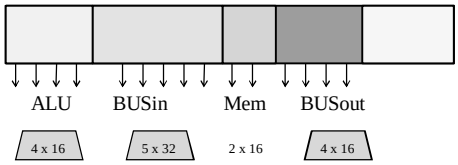
- Une minstruction est associée à un groupe de commande
  - Par exemple un groupe
    - lect/ecr mémoire
    - Fonction ALU ADD, INCX ….
- Il faut alors identifier le groupe correspondant aux commandes déclenchées



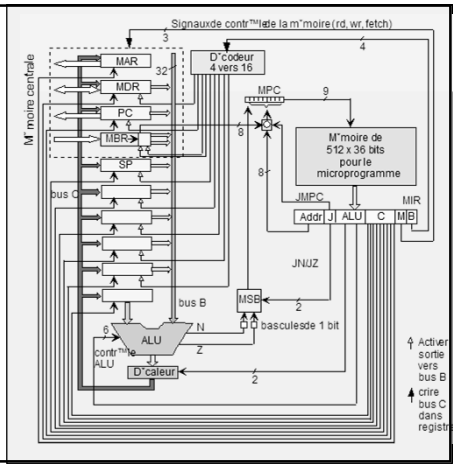
- Permet de déclencher 8 commandes avec 3 bits (et un décodeur)

Microporgammation par champs

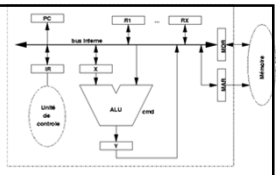
- On mélange horizontale et verticale
  - Un ensemble de champs et un codage par groupe dans chaque champs



- La minst doit correspondre à des signaux sur le CPU



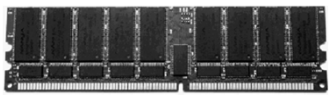
Une machine simple



- ALU Cmd
  - add sub shift rot inc…
- BUSin
  - Pcin, ROin..Rxin, MDRin, MARin, Xin, IRin
- BUSout
  - Pcout, ROout..Rxout, MDRout, Yout, IRout
- Mem
  - Lect, Ecr, (waitmemory)

AE cours

La fonction mémoire



Objectifs

- Comment stocker une information?
- Comment adresser un emplacement mémoire
- Quelques types de mémoire?
- Notion hiérarchie mémoire

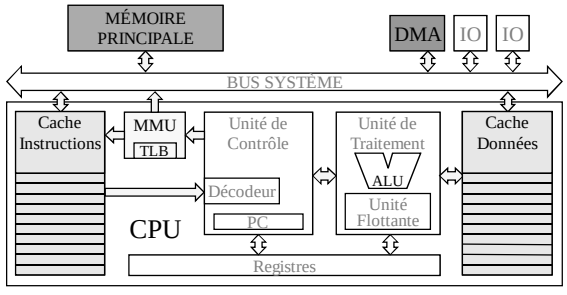
Introduction

- Mémoire = dispositif de stockage
  - Enregistre de l'information
  - Restitue à la demande cette information
- Différentes formes de stockage
  - Numérique (Circuits intégrés, CDROM,…)
  - Analogique (Cassettes, Disques durs, …)
- Ce cours s'intéresse aux mémoires à **semi-conducteurs**. (Circuits intégrés)

Les mémoires numériques

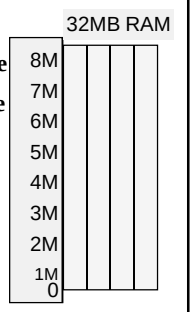
- Plus petite information stockable : 1 bit
- ➔ notion de *case mémoire*
- 2 grandes catégories de mémoires :
  - Accès *aléatoire* :
    - Chaque case mémoire est utilisable séparément
    - Notion d'*adresse mémoire*
  - Accès *sériel* : (mémoires *séquentielles*)
    - Accès aux cases mémoires séquentiellement
    - Pas d'adresse utilisée

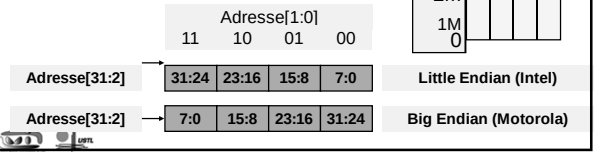
Place de la mémoire



Terminologie

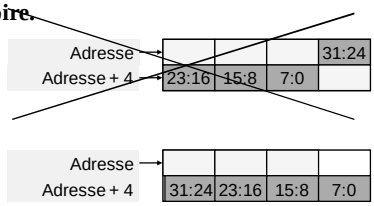
- **Mémoires sont souvent byte-addressable**
  - **A chaque adresse correspond un byte**
- **L'ordonnancement des bytes dans un mot est aussi un choix architectural:**



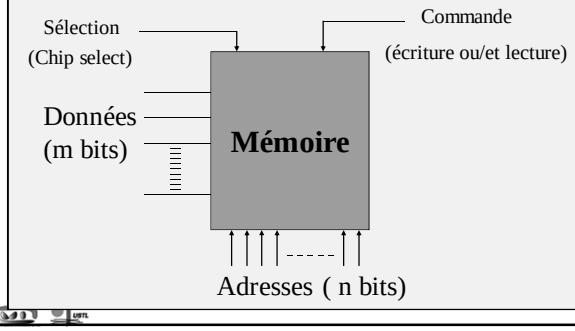


Alignement

L'architectures imposent souvent que les données soient alignées par rapport à la taille des mots mémoire.



Les broches d'une mémoire



Paramètres d 'une mémoire

- **TECHNOLOGIE**
- **FORMAT**

Le format représente le nombre de bits de la donnée (m).

- **NOMBRE DE CASES MEMOIRES**

Nombre de données pouvant être stockées. (N)

$N = 2^n$ (où n est le nombre de bits d 'adresse)

- **CAPACITE**

$C = m \times N$

Calcul de la capacité d 'une mémoire

Mémoire : 8 bits de données
16 bits d 'adresse.

Format : m = 8

Nombre de cases : $N = 2^{16} = 65536$

Capacité : $C = m \times N = 8 \times 65536 = 524288$ bits

Expression de la capacité mémoire

Pour des raisons de simplification, on exprime la capacité :

- en kilo-octets (Ko) 1 octet : 8 bits
  1 Ko = 2^{10} = 1024 octets
- en méga-octets (Mo) 1 Mo = 2^{20} octets
- en giga-octets (Go) 1 Go = 2^{30} octets

Pour l'exemple précédent C = 524288 bits

C = 524288/8 = 65536 octets

C = 65536 octets = 65536/1024 = 64 Ko

Principales Caractéristiques

- *Temps* (*timings*)
  - Écriture : délai entre la présentation des données et leur mémorisation réelle.
  - Lecture : délai entre la présentation de l'adresse et la disponibilité des données
  - Accès : en général, la moyenne des temps de lecture et d'écriture
- *Consommation* : puissance nécessaire à une opération (≠ selon repos ou lecture/écriture)

DIFFERENTS TYPES DE MEMOIRES

- **RAM** Random Access Memory mémoires à accès aléatoire (à la demande)
  Mémoires vives

  On peut lire et écrire des données.
  Ces mémoires sont volatiles: Elles perdent leur contenu lorsqu'elles ne sont plus alimentées.

- **ROM** Read Only Memory
  Mémoires mortes

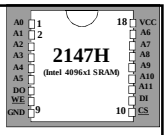
  On ne que lire les données.
  Elles conservent leur contenu lorsqu'elles ne sont plus alimentées.

Autres mémoires lecture/écriture

- *Magnetic Bubble Memory* : mémoire de masse non volatile. Utilise des « bulles magnétiques »
  - Plus utilisées
- Mémoires séquentielles
  - *First In First Out* : mémoires gérées en file; premier bit écrit, premier bit lu.
  - *First In Last Out* : mémoires gérées en pile; premier bit écrit, dernier bit lu.

14

DIFFERENTS TYPES DE RAM



- RAM statique SRAM

  Les informations sont mémorisées par des bascules tant que l'alimentation est présente.

  Application : Mémoire cache d 'un ordinateur
  Zone de mémoire rapide où sont stockées les données les plus fréquemment utilisées par l'ordinateur.

- RAM dynamique DRAM

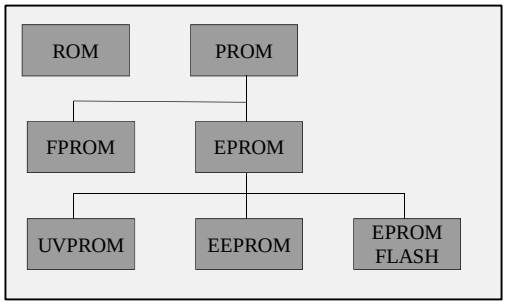
  Les informations sont mémorisées par des condensateurs et des transistors.
  Cette technologie oblige à rafraîchir les données.

Des types de DRAM

- De nouvelles versions régulièrement
  - *Fast Page Mode* : les barrettes des 486
  - *Extensed Data Out* : premiers Pentiums
  - *Burst EDO* : cousine de l'EDO
  - *Synchronous DRAM* : années 2000 synchronisation avec horloge du *m*P
  - *Double-data-rate DRAM* : SDRAM à cadence double (2 fronts)
  - *Error Correction Code* : SDRAM ou DDRAM à correction d'erreurs (cf. codes correcteurs)
  - *RamBus* : Mémoires à haute vitesse 600-800 MHz

16

DIFFERENTS TYPES DE ROM



DIFFERENTS TYPES DE PROM

PROM : Programmable ROM

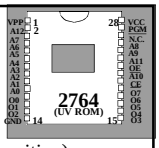
FPROM (Fuse PROM)

Ces mémoires sont livrées non enregistrées par le fabricant. Une fois programmées, on ne peut plus modifier leur contenu.

EPROM (Erasable PROM)

Ces mémoires possèdent les avantages de la PROM avec un plus qui est l'effacement des données par l'utilisateur et la possibilité de reprogrammer.

DIFFERENTS TYPES D'EPROM



UVPROM (ou EPROM)

Effaçable aux UV (10 à 20 minutes d 'exposition).

Effacement total de la mémoire.

Programmation par tension de 25V.

EEPROM (Electricaly EPROM)

Effaçable et programmable électriquement.

Effacement adresse par adresse.

Coût de fabrication élevé.

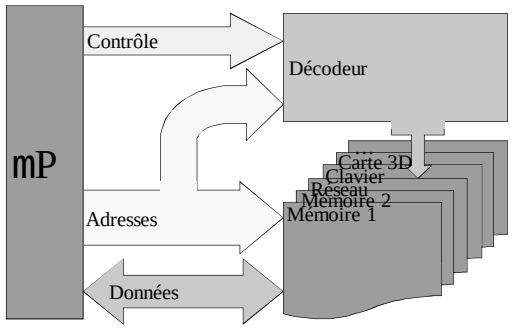
EPROM FLASH

Effaçable électriquement.

Effacement total de la mémoire.

Plus rapide et moins cher que l 'EEPROM.

Synoptique



20

METHODES DE DECODAGE D'ADRESSES

•Le CPU fournit l'adresse de la donnée désirée,
un circuit de décodage
localiser la donnée dans la mémoire.

•La puce mémoire possède un pin « CS ou CE » qui est activé pour pouvoir accéder au contenu de la mémoire.

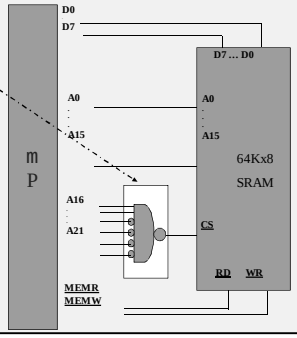
•Les bus de données sont connectés aux pins de données de la mémoire

- •MEMR/MEMW aux pins RD/WR de la mémoire.
- •Les lignes d'adresses les moins significatives sont connectées aux pins d'adresse de la mémoire
- •Les plus significatives sont utilisées pour activer CS.

Un exemple…

Pour sélectionner la puce mémoire,
A21-A16 sont choisies par le Concepteur comme « 000011 ».
On dit que la mémoire est décodée par A21-A16 = 000011.

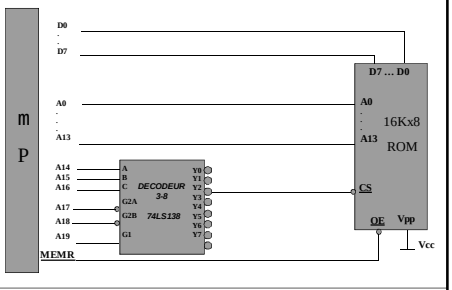
La puce mémoire est sélectionnée A21-A0 se trouvent dans la bande d'adresse: 030000h – 03FFFFh



CIRCUIT DE DECODAGE AVEC LE *DECODEUR 3-8 '74xx138'*

La bande d'adresses qui sélectionne la mémoire de la figure est donc:

A19-A0 =
E8000h – EBFFFh



Pour sélectionner le DECODEUR A17-A18-A19 = 100. Les 3 lignes A14-A15-A16 sélectionnent une puce mémoire parmi 8 mémoires. Celle de la figure est sélectionnée à la sortie **Y2** du décodeur, soit quand **A14-A15-A16 = 010**.

Hiérarchie mémoires

- Théorie de la localité
  - Les instructions se suivent (en général)
  - Les données sont rangées ensemble (idem)
- Idée :
  - Charger en mémoire les parties utiles
  - → Peu de mémoire
  - → Gros programmes

Département Informatique 24

Principes financiers

- Mémoire rapide = coût élevé (€)
  - Beaucoup de mémoire lente
  - Peu de mémoire rapide
- En pratique :
  - DRAM :
    - Plus de 5 ns (5.10^{-9} secondes)
    - Environ 200 MHz
  - SRAM :
    - Moins de 1 ns
    - Plusieurs GHz

25

Hiérarchie des mémoires

- Cache L1
  Dans le *m*P, ultra rapide, très chère
- Cache L2
  SRAM, souvent dans le *m*P
- Cache L3
  SRAM, hors du processeur
- RAM classique
  DRAM
- Disque dur
- Bande magnétique
  Très bon marché, vraiment lente
- **Objectif=** coût bande magnétique + vitesse L1
  - Vers la mémoire virtuelle!!!

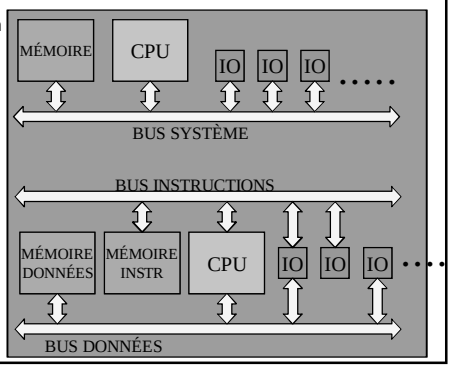
26

Mémoire virtuelle

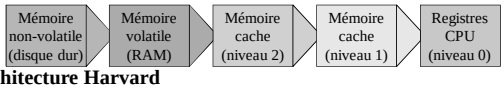
Jean-Luc Dekeyser
Jean-luc.dekeyser@lifl.fr

Von Neumann vs. Harvard

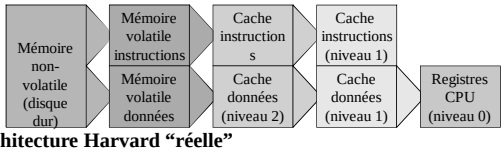
- Von Neumann
- Harvard



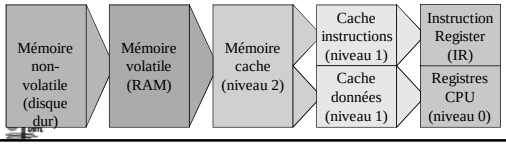
- **Architecture Von Neumann**



- **Architecture Harvard**



- **Architecture Harvard "réelle"**



Espace d'adressage

Processeurs 32-bit: espace d'adressage = 2^{32} = 4GB

Le programme/données est chargé complètement en mémoire avant de s'exécuter.

- **La mémoire ne peut pas dépasser 4GB.**
- **Les références à la mémoire se font avec l'adresse physique des données.**

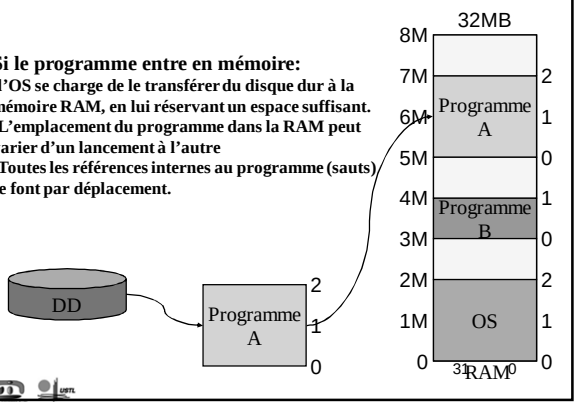
Il y a <u>toujours</u> des programmes trop grands pour être stockés complètement en mémoire.

- **Passage à 64 bits (compatibilité avec les 32 bits)**
- **Un référencement des données par une adresse virtuelle**

Adressage relatif

Si le programme entre en mémoire:

- **l'OS se charge de le transférer du disque dur à la mémoire RAM, en lui réservant un espace suffisant.**
- **L'emplacement du programme dans la RAM peut varier d'un lancement à l'autre**
- **Toutes les références internes au programme (sauts) se font par déplacement.**



Mémoire virtuelle

Lors du lancement d'un programme

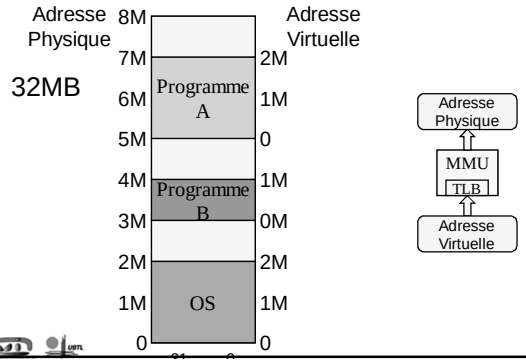
- **L'OS réserve un espace d'adressage virtuel pour le programme.**
- **Cet espace a une taille variable, déterminée par le programme lui-même lors de sa compilation (mais limitée par l'OS).**

Toute référence à la mémoire est faite par rapport à cet espace d'adressage virtuel, et non pas par rapport à la mémoire physique.

- **La mémoire physique hiérarchique héberge tous les espaces virtuels de tous les programmes en train de s'exécuter sur le processeur.**

Une unité dans le processeur (le MMU ou *memory management unit*) se charge de traduire les adresses virtuelles en adresses physiques.

Mémoire virtuelle



Pagination et segmentation

La mémoire virtuelle ne résout pas à elle seule tous les problèmes liés à l'adressage:
Un moyen pour stocker tous les programmes en train de s'exécuter dans une mémoire physique, sachant que celle-ci est trop petite pour tous les accueillir dans leur entièreté.

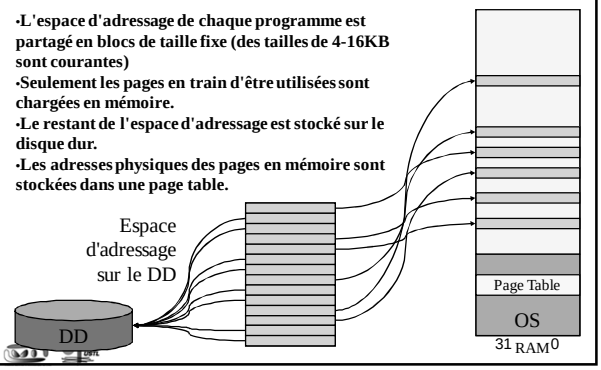
- **Stocker en mémoire seulement les parties des programmes qui sont effectivement utilisées.**

Il y a trois approches "standards":

- **Pagination**
- **Segmentation**
- **Segmentation paginée**

Pagination

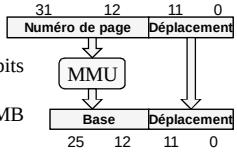
- **L'espace d'adressage de chaque programme est partagé en blocs de taille fixe (des tailles de 4-16KB sont courantes)**
- **Seulement les pages en train d'être utilisées sont chargées en mémoire.**
- **Le restant de l'espace d'adressage est stocké sur le disque dur.**
- **Les adresses physiques des pages en mémoire sont stockées dans une page table.**



Pagination - Adressage

Une adresse virtuelle est donc composée de deux parties: un numéro de page et un déplacement dans la page.

Exemple:
Largeur des adresses = 32bits
Taille des pages = 4KB
Taille de la mémoire = 64MB



Cette transformation, réalisée par le MMU, permet de travailler avec des espaces d'adressage plus grands que la mémoire physique

Pagination- Défaut de page

Une adresse virtuelle peut référencer

- **soit une page en mémoire (*page hit*)**
- **soit une page sur le disque dur (*page miss*).**

Pour un *page miss*, le MMU engendre une « interruption » (*page fault*) et une procédure transfert la page du disque vers la mémoire (plusieurs millions de cycles d'horloge).

La taille des pages influe sur l'efficacité de la gestion des *page faults*:

- **une taille trop petite augmente le nombre de *page miss***
- **une taille trop grande augmente le temps de transfert des données.**
- **Dans les systèmes actuels, la taille des pages varie entre 4 et 16KB.**

Pagination - Remplacement

Comment amener une page quand la mémoire centrale est pleine?

- **On transfère une autre page de la mémoire vers le disque dur (*swap*).**

Il faut choisir la bonne page à écraser en mémoire...

- **L'algorithme de choix pour cette opération est souvent le LRU (*least recently used*).**

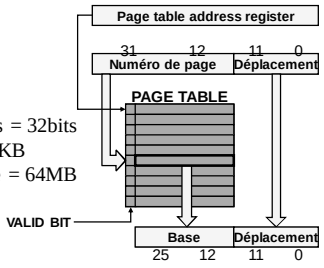
Une mémoire est trop petite, un algorithme de swap peu efficace ou des données "mal" rangées en mémoire => le phénomène de *thrashing :**

- **la machine passe tout son temps à transférer des pages et « oublie » de calculer.**

*a situation where large amounts of computer resources are used to do a minimal amount of work

Pagination - Implémentation

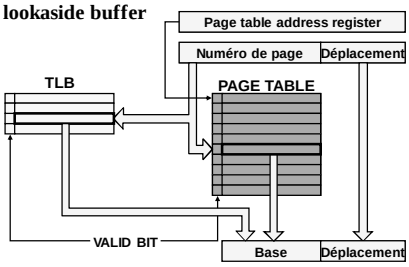
Le MMU doit trouver l'adresse de la page physique correspondant à une page virtuelle
Tableau de pages (*page table*), stocké en mémoire et géré par l'OS.



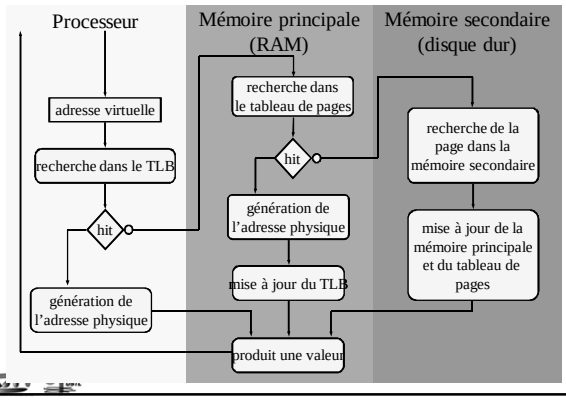
Exemple:
Largeur des adresses = 32bits
Taille des pages = 4KB
Taille de la mémoire = 64MB

Pagination - TLB

Pendant l'exécution la même page est souvent référencée plusieurs fois de suite. Pour accélérer la recherche de l'adresse physique, on utilise un "raccourci" matériel: le TLB ou translation lookaside buffer

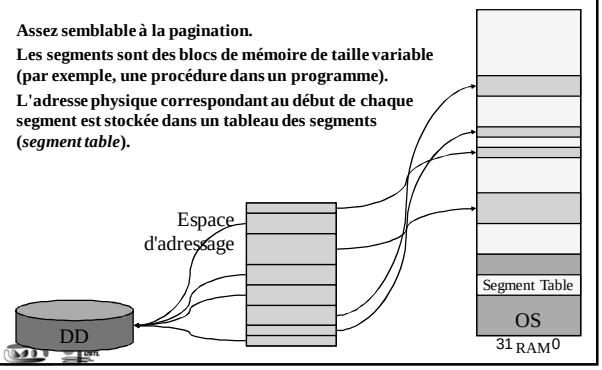


Pagination - Algorithme



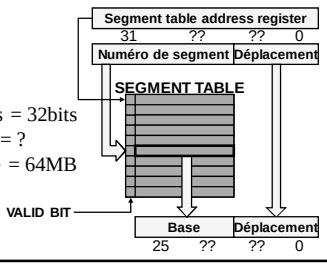
Segmentation

Assez semblable à la pagination.
Les segments sont des blocs de mémoire de taille variable (par exemple, une procédure dans un programme).
L'adresse physique correspondant au début de chaque segment est stockée dans un tableau des segments (*segment table*).



Segmentation - Implémentation

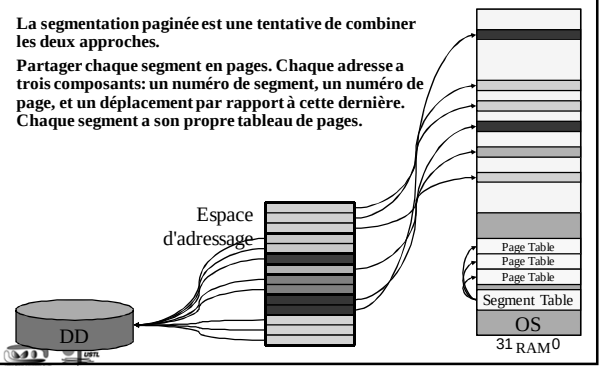
- **les segments correspondent à des éléments précis (procédure, bloc de données).**
- **une manipulation des données plus aisée (la taille des segments peut varier dynamiquement)**
- **une meilleure protection des données des programmes.**
- **taille variable des segments complique le travail du MMU.**



Exemple:
Largeur des adresses = 32bits
Taille des segments = ?
Taille de la mémoire = 64MB

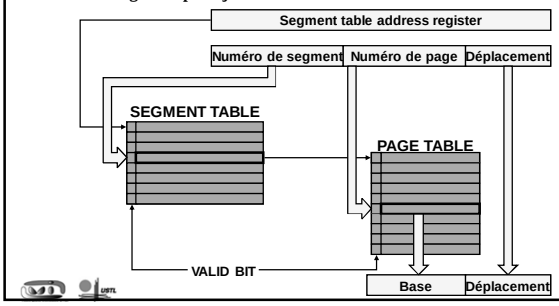
Segmentation paginée

La segmentation paginée est une tentative de combiner les deux approches.
Partager chaque segment en pages. Chaque adresse a trois composants: un numéro de segment, un numéro de page, et un déplacement par rapport à cette dernière. Chaque segment a son propre tableau de pages.



Segmentation paginée - Implémentation

Le contrôle de ce genre de système reste plus compliqué que dans le cas de la pagination, mais les avantages sont généralement considérés comme suffisamment grands pour justifier l'effort additionnel.

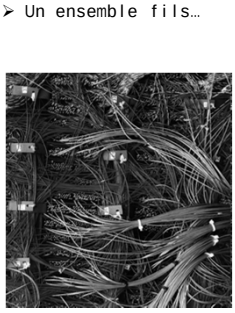


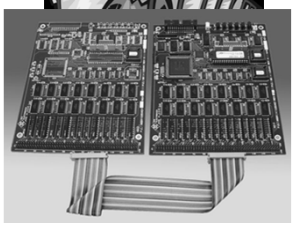
BUS

Jean-Luc Dekeyser
Jean-luc.dekeyser@lifl.fr

C'est quoi un bus?

- Véhicule lent, mais pour beaucoup de personnes
- Un ensemble fils…

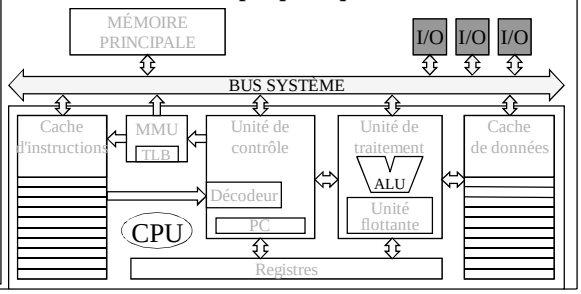




2

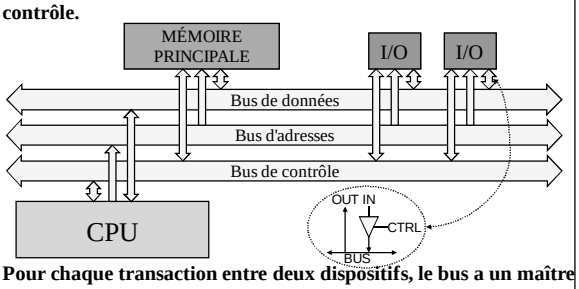
Périphériques et bus système

- **Structure du bus système**
- **Gestion du bus système**
- **Fonctionnement des périphériques**



Structure du bus système

Largeur de 50 à 100 lignes séparées,
3 groupes fonctionnels: bus de données, bus d'adresses et bus de contrôle.



Pour chaque transaction entre deux dispositifs, le bus a un maître (normalement le CPU) et un esclave. Le maître est le dispositif qui démarre la transaction.

Classification des bus

Les bus peuvent être classifiés selon plusieurs paramètres:

- **Largeur du bus**
  - **Adresses**
  - **Données**
- **Type**
  - **Dédié**
  - **Multiplexé**
- **Timing**
  - **Synchrone**
  - **Asynchrone**
- **Type de transfert de données**
  - **Read**
  - **Write**
  - **Read-modify-write**
  - **Block**
- **Méthode d'arbitrage**
  - **Centralisée**
  - **Distribuée**

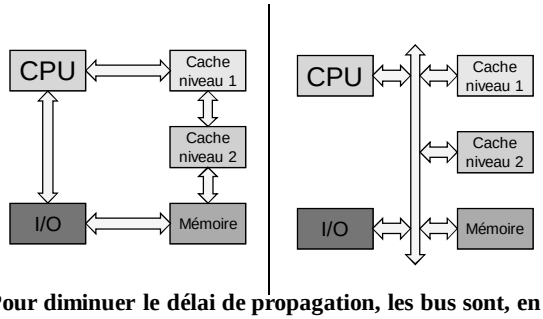
Largeur du bus système

- **La taille du bus de données est un élément clé pour la performance globale du système.**
- **Exemple:**
  - **le bus de données a une largeur de 8 bits et les registres du processeur ont une largeur de 16 bits**
  - **deux cycles de l'horloge du bus sont nécessaires pour chaque accès.**

Largeur du bus système

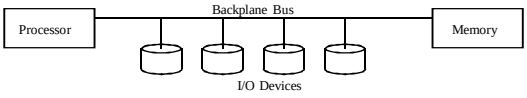
- **Les bus d'adresses et de données peuvent être partagés (multiplexés).**
  - **La perte en performance est toutefois considérable.**
  - **Un bus d'adresses dédié augmente considérablement la performance du bus système.**
- **La largeur du bus d'adresses détermine la capacité maximale du système de mémoire.**

Type de bus système



Pour diminuer le délai de propagation, les bus sont, en général, organisés de façon hiérarchique.

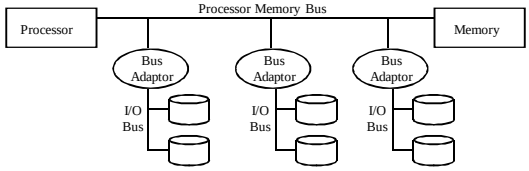
Avec unBus: Backplane Bus



- **Le bus (backplane bus) est utilisé pour:**
  - **Communication processeur à mémoire**
  - **Communication entre I/O devices et mémoire**
- **Avantage: Simple et peu couteux**
- **Désavantage: lent et goulot d'étranglement**

9

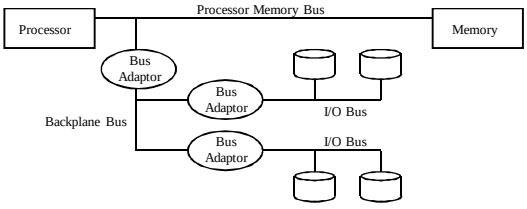
Sysème à deux Bus



- **Bus I/O se connecte au Bus mémoire bus via un adaptateur:**
  - **Processor-memory bus: principalement le trafic mémoire processeur**
  - **I/O bus: slots d'extension pour I/O devices**

10

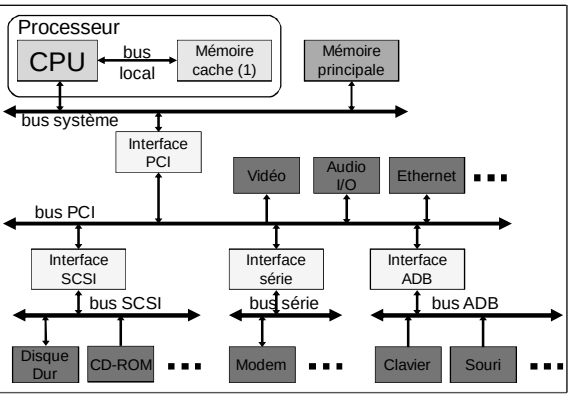
Système trois bus



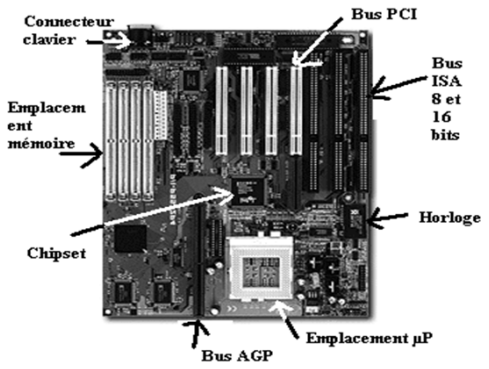
- **Backplane bus connecté au bus proc/mem**
  - **Proc/mem bus utilisé pour le trafic processor mémoire**
  - **I/O bus sont connectés au backplane bus**
- **Avantage: réduit la charge du bus Proc/mem**

11

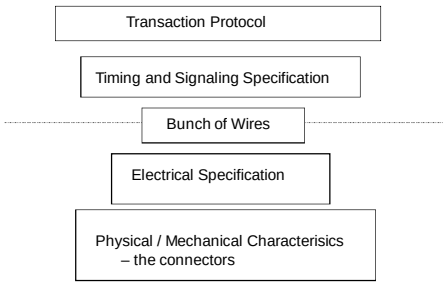
Exemple - Apple Macintosh 7200



Suffit d'ouvrir votre PC….



Niveau d'abstraction d'un bus?



14

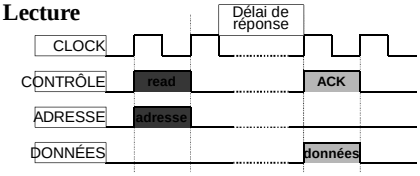
Les bus synchrones

- **Inclut une horloge dans les lignes de contrôle**
- **Avantage: ne demande que peu de logique et donc fonctionnement rapide**
- **Désavantages:**
  - **Toutes les devices doivent utiliser la même horloge**
  - **Limite la longueur des fils pour garantir la propagation de la clock**
- **Protocole est fixé et relatif à la clock**

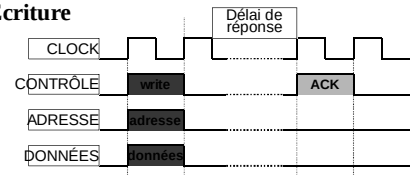
15

Les bus synchrones

- **Lecture**



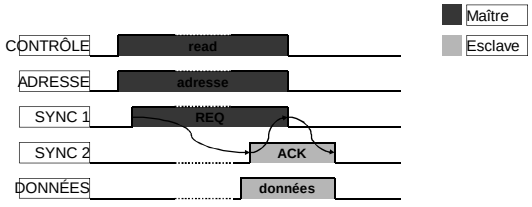
- **Écriture**



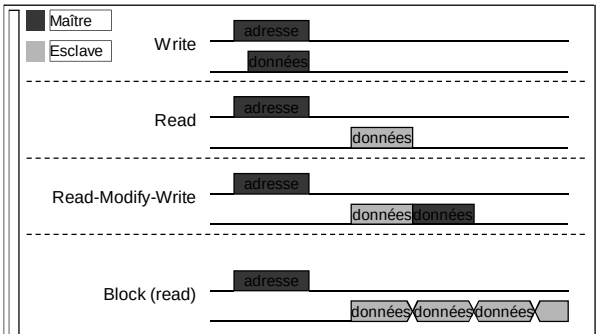
Les bus asynchrones

- **Les bus asynchrones ne nécessitent pas d'horloge.**
  - **signaux spéciaux (p.ex. REQ, ACK) pour gérer les transferts des données**
  - **Protocole de synchronisation (handshaking).**
- **Le timing asynchrone est plus rapide que le timing synchrone.**
- **Longueur des fils augmente**
- **Son contrôle est plus difficile.**

Lecture asynchrones



Types de transfert de données

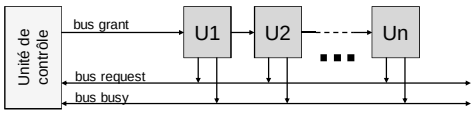


Arbitrage du bus

- **Pour déclencher un transfert sur le bus, il faut:**
  1. **Obtenir le contrôle du bus**
  2. **Envoyer une demande à l'autre dispositif**
  3. **Attendre/envoyer les données**
  4. **Relâcher le bus**
- **L'arbitrage du bus est le mécanisme qui alloue le contrôle du bus aux dispositifs qui le demandent**
  - **Eviter tout conflit.**

Arbitrage du bus

- **Soit gérer localement dans chaque dispositif (arbitrage distribué) ou,**
- **Soit par une unité de contrôle dédiée (arbitrage centralisé)**
  - **Réalise un parmi plusieurs algorithmes possibles (p.ex. daisy-chaining).**



Exemple: le bus PCI

- **Le bus PCI (Peripheral Component Interconnect) est un bus synchrone, avec un système d'arbitrage centralisé, développé par Intel qui le rendit public**
- **On peut utiliser jusqu'à 64 lignes de données, à 33MHz (taux de transfert = 264 MB/sec)**
- **Il y a 50 lignes obligatoires, divisées en 5 groupes:**
  - **Système (clock, reset, etc)**
  - **Contrôle d'interface**
  - **Erreur**
  - **Adresses et données (multiplexées)**
  - **Arbitrage**
- **Les 50 lignes optionnelles sont divisées en 4 groupes:**
  - **Interruption**
  - **Extension à 64 bits**
  - **Support du cache**
  - **JTAG/Boundary scan**
- **Les commandes possibles sont:**
  - **Interrupt acknowledge**
  - **I/O read**
  - **Memory read**
  - **Memory read multiple**
  - **Memory write and invalidate**
  - **Configuration write**
  - **Special cycle**
  - **I/O write**
  - **Memory read line**
  - **Memory write**
  - **Configuration read**
  - **Dual address cycle**

A lire

- **http://www.exercicegratuit.com/cours-maintenance-pc/les-bus-isa-agp-pci-mca**

Les entrées sorties

Jean-Luc Dekeyser
Jean-luc.dekeyser@lifl.fr

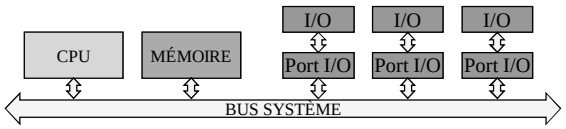
S3- 2010/11

Périphériques

- **Les dispositifs d'entrée/sortie (périphériques ou E/S ou I/O)**
  - **des caractéristiques physiques très différentes de celles du processeur ou des mémoires (en vitesse et en timing, par exemple):**
  - **il faut dans ces cas un système d'interface.**

Périphériques

- **Afin d'uniformiser les accès aux périphériques, l'interface (port I/O) fournit également un ensemble de registres de données et de contrôle.**
- **Seulement deux opérations sont alors nécessaires, indépendamment du périphérique:**
  - **la lecture et l'écriture d'un registre I/O.**

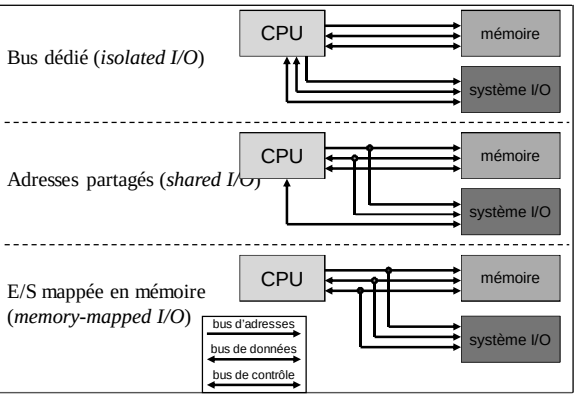


Classification des périphériques

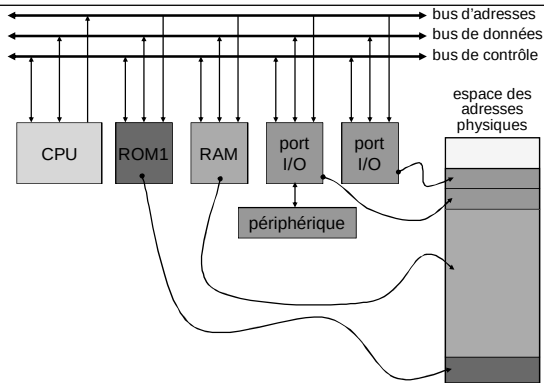
Les méthodes de gestion des périphériques peuvent être classifiées selon deux paramètres principaux:

- **Accès**
  - **Bus dédié**
  - **Adresses partagées**
  - **E/S mappées en mémoire**
- **Gestion**
  - **Par interrogation**
  - **Par interruption**
  - **DMA**
  - **Processeurs d'E/S**

Accès au périphériques



E/S mappées en mémoire



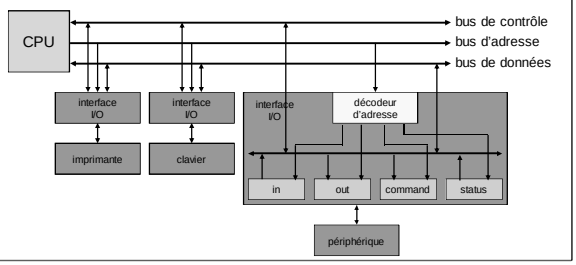
Gestion des périphériques

Transactions entre les périphériques, le processeur et la mémoire

- **par interrogation, le processeur les consulte périodiquement pour déterminer leurs besoins.**
- **par interruption; signaler au processeur qu'ils ont besoin d'attention. Le processeur peut alors déclencher la procédure appropriée.**
- **Si les périphériques ont accès direct à la mémoire (DMA), ils sont capables de communiquer avec la mémoire sans la supervision du processeur, qui se limite à déclencher les transactions.**
- **Les processeurs d'entrée/sortie sont capables de gérer les transactions de façon "intelligente" (p.ex. à partir de programmes chargés par le système d'exploitation).**

Gestion par interrogation

- **La gestion par interrogation demande très peu d'effort (et donc très peu de matériel) de la part des périphériques.**
- **Temps perdu par le processeur pendant le polling.**



Registre Status

- **L'accès au périphériques peut être réalisé par le processeur à travers des instructions de transferts**
- **Des registres de statuts permettent au processeur de déterminer si des données peuvent être transférées et l'état des transferts (erreurs)**

Scrutation (Interrogation)

- **Par scrutation de registres de status, le processeur synchronise les transferts**
  - **Wait (status_transfert == OK)**
    - **Effectue le transfert**

Ou

  - **If (status_transfert == OK)**
    - **Effectue le transfert**
  - **EndIf**

Scrutation

- **Avec la scrutation le système est synchrone**
  - **le programme contrôle exactement lorsqu'il peut accéder à une ressource**
- **Inconvénient, le programme doit souvent tester les statuts pour rien, et suffisamment souvent pour ne pas rater de données**

Interruption

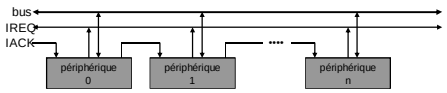
- **Les interruptions mettent en œuvre du matériel spécialisé et dépendent du processeur utilisé (vecteur, type d'accès à la routine de traitement, etc…)**
- **De nombreuses instructions sont souvent nécessaires pour le traitement de l'interruption (sauvetage de contexte, test de la requête, transfert I/O-mémoire, *interrupt handler*, …)**
  → bande passante limitée

Gestion par interruption

- La gestion par interruption demande qu'un périphérique signale au processeur qu'il a besoin d'attention.
  - Le processeur arrête l'exécution courante pour exécuter un code associé au périphérique.
- L'interruption est synchronisée par des signaux de *handshaking*:
  - le périphérique active IREQ
  - lorsque le processeur est prêt, il répond avec IACK
  - le périphérique s'identifie, de façon à ce que le processeur exécute la bonne procédure (*interrupt handler* ou *interrupt service routine*)

Gestion par interruption

- En général, il y a une seule ligne IREQ, partagée par tous les périphériques.
- Un seul périphérique doit recevoir le signal IACK.
- La solution courante est la chaîne de priorité (*daisy-chain*).



Direct memory access

- Avec les interruptions, on a transformé les attentes actives (*e.g.* par scrutation) en attente passive :
  - Le processeur peut faire une autre tache en attendant le périphérique
- Mais pour le transfert d'informations, c'est le µP qui sert de passerelle entre le périphérique et la mémoire…
- Solution : DMA

DMA

- Pour des systèmes où le taux de transfert entre les I/O et la mémoire est élevé, la scrutation ou les interruptions sont inutilisables
- Un système plus performant est nécessaire

Accès direct à la mémoire (DMA)

- Pour accélérer les transferts, on peut permettre au périphérique de devenir maître du bus et d'accéder directement à la mémoire
  - Particulièrement avantageux lors des transferts de grands blocs.
- Pour implémenter cette fonction (DMA pour *direct memory access*), l'interface du périphérique doit contenir un contrôleur DMA
  - un circuit capable de gérer les transferts de données.

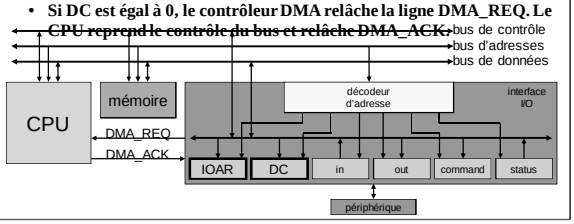
Accès direct à la mémoire (DMA)

- Le contrôleur DMA contient deux registres,
  - le registre d'adresse IOAR stocke l'adresse du prochain mot à transférer. Il est incrémenté après chaque transfert.
  - le décompteur DC stocke le nombre de mots restant à transférer. Il est décrémenté après chaque transfert.
- Deux lignes de contrôle dédiées (*DMA_REQ* et *DMA_ACK*) reliant le processeur et le contrôleur DMA sont également nécessaires.

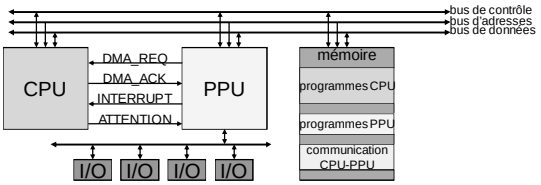
DMA - Implémentation

L'accès direct à la mémoire se déroule ainsi:

- **Le CPU charge IOAR et DC.**
- **Quand le contrôleur DMA est prêt, il active la ligne DMA_REQ. Le CPU donne le contrôle du bus au contrôleur DMA et active DMA_ACK.**
- **Le périphérique et la mémoire échangent les données. DC est décrémenté et IOAR incrémenté après chaque transfert.**
- **Si DC est égal à 0, le contrôleur DMA relâche la ligne DMA_REQ. Le CPU reprend le contrôle du bus et relâche DMA_ACK.**



Processeurs d'entrée/sortie

- **Les processeurs d'entrée/sortie (PPU ou *peripheral processing units*) sont l'extension du concept de DMA.**
- **Un processeur prend la place du contrôleur DMA et gère les transferts de données (généralement un seul processeur s'occupe de plusieurs périphériques).**
- **Ce type de processeur a un jeu d'instructions limité et fonctionne comme "esclave" du CPU.**
- **Il est capable d'exécuter des programmes fournis par le système d'exploitation.**
- **Une zone mémoire est normalement réservée pour la communication entre le CPU et le PPU.**

Processeurs d'entrée/sortie