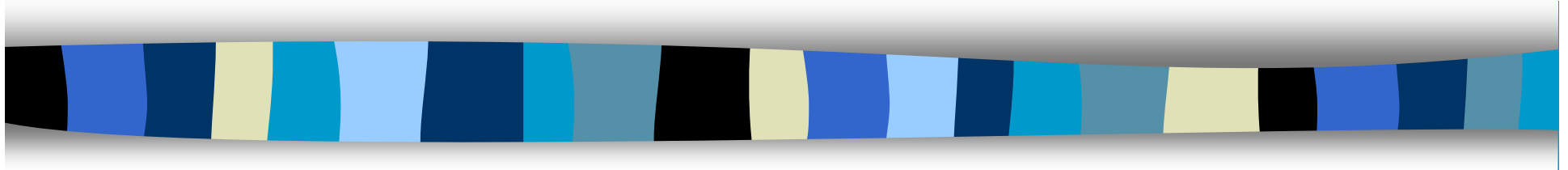


Support de cours

# Architecture des ordinateurs



Cours de 20 h, 1<sup>ère</sup> semestre

Module Architecture des ordinateurs I

1<sup>ère</sup> année, Filière Génie Informatique – EMI

Préparé en 2007/2008 par : F. Z. Belouadha et M. El Euldj

**Actualisé en 2010/2011 par : F. Z. Belouadha**

Dispensé par : F. Z. Belouadha et D. El Ghanami



# Références

- [1] John L. Hennessy et David A. Patterson, Architecture des ordinateurs : une approche quantitative, McGraw-Hill, 1992
- [2] Andrew Tanenbaum, Architecture de l'ordinateurs, Pearson education, 2005
- [3] Emmanuel Lazard, Architecture de l'ordinateur, Pearson education, 2006
- [4] Arvind and Krste Asanovic, Cours de MIT,  
<http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-823Fall-2005/>
- [5] Hakim Amrouche, Cours Structure machine,  
[http://amrouche.esi.dz/doc/ch7\\_memoires.pdf](http://amrouche.esi.dz/doc/ch7_memoires.pdf)
- [6] Support de cours de MIT adapté par M. Eleuldj 2008,  
<http://www.emi.ac.ma/~eleuldj>



# Intérêt de l'architecture des ord.

- Pas du tout
  - Faire du traitement de texte ou les bases de données
  - Créer ou gérer un site Internet
  - Développer des logiciels en Java ou en C++
- Un peu quand même
  - Satisfaire la curiosité intellectuelle : Comment marche cette machine sur laquelle je passe des journées ?
- Enormément
  - Développer des systèmes de traitement haute performance (Audio, Vidéo, Médical, Spatial...)
  - Développer des systèmes matériels (mémoire,  $\mu$ Processeur...)
  - Donner une expertise en choix de matériel
  - Écrire des systèmes d'exploitation
  - Développer des compilateurs



# Plan du cours

## Architecture des ordinateurs

- I. Structure de l'ordinateur
- II. Architecture de l'ordinateur de l'antiquité aux années quarante
- III. Architecture et évolution de l'ordinateur dans les années cinquante
- IV. Architecture et évolution de l'ordinateur dans les années soixante
- V. Microprogrammation
- VI. Hiérarchie de la mémoire



## Chapitre 1

# STRUCTURE DE L'ORDINATEUR

1. Terminologie
2. Définition
3. Unité centrale
4. Schéma d'UAL
5. Registres
6. Décodeur et séquenceur
7. Bus
8. Outils logiciels



# Terminologie

- Anglais : computer  $\Rightarrow$  calculateur
- Français : ordinateur  $\Rightarrow$  ordre (commande et organisation)
- Arabe : الكمبيوتر, الحاسوب
- 1955 : Création du mot français «ordinateur», déposé d'abord par IBM, pour désigner ce qui est en anglais un "computer"

# Définition

- Architecture des ordinateurs

## Organisation des ordinateurs

### Structure des ordinateurs

UCT (UAL+UC)

+ Mémoire

+ Unités d'E/S

+ Interconnexions

+ logiciels

+ compilateurs

+ systèmes d'exploitation

+ algorithmes

+ conception des circuits

+ SGBD + algorithmes

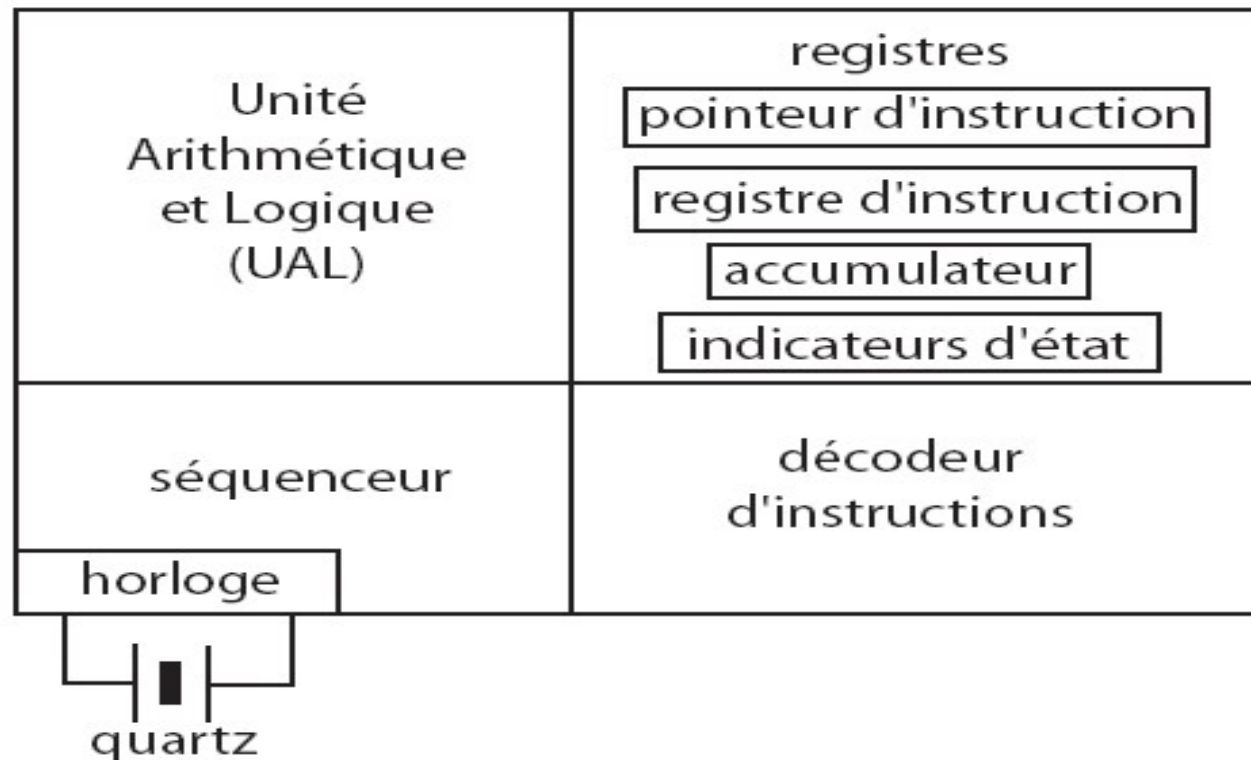
+ langages + ...

- Ordinateur :

- Composants matériels qui communiquent entre eux
- Outil utilisé pour le calcul et le traitement automatique de l'information

# Unité centrale de traitement

- ❑ Processeur (CPU): traite les données et envoie des ordres aux autres composants

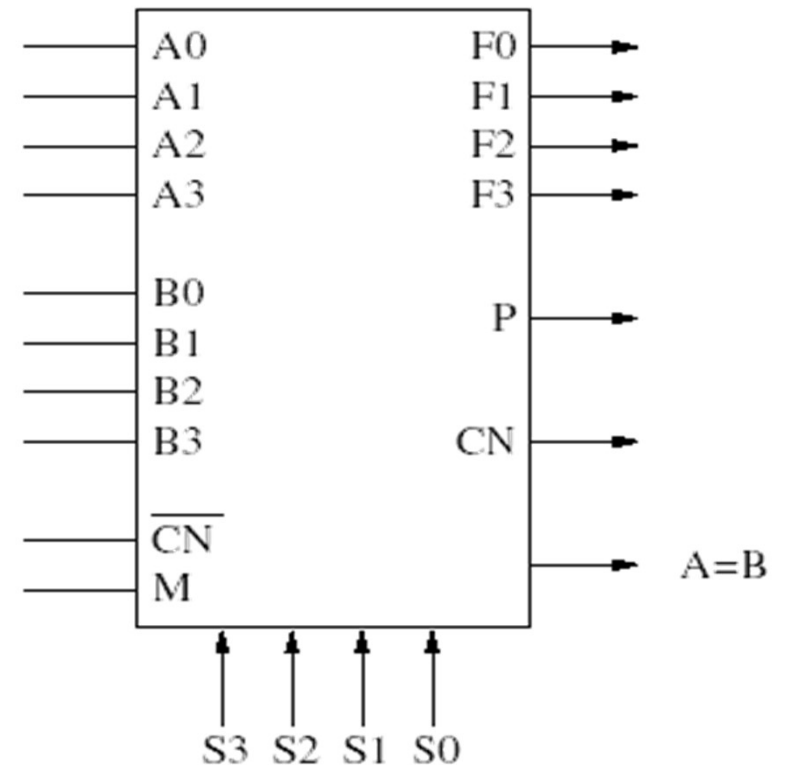




# Exemple d'UAL

## □ 74LS181 (UAL 4 bits)

- Opérandes sur A & B
- Type de fonctions sur M (1 logiques, 0 arithmétiques)
- Type d'opération sur S
- Résultat sur F





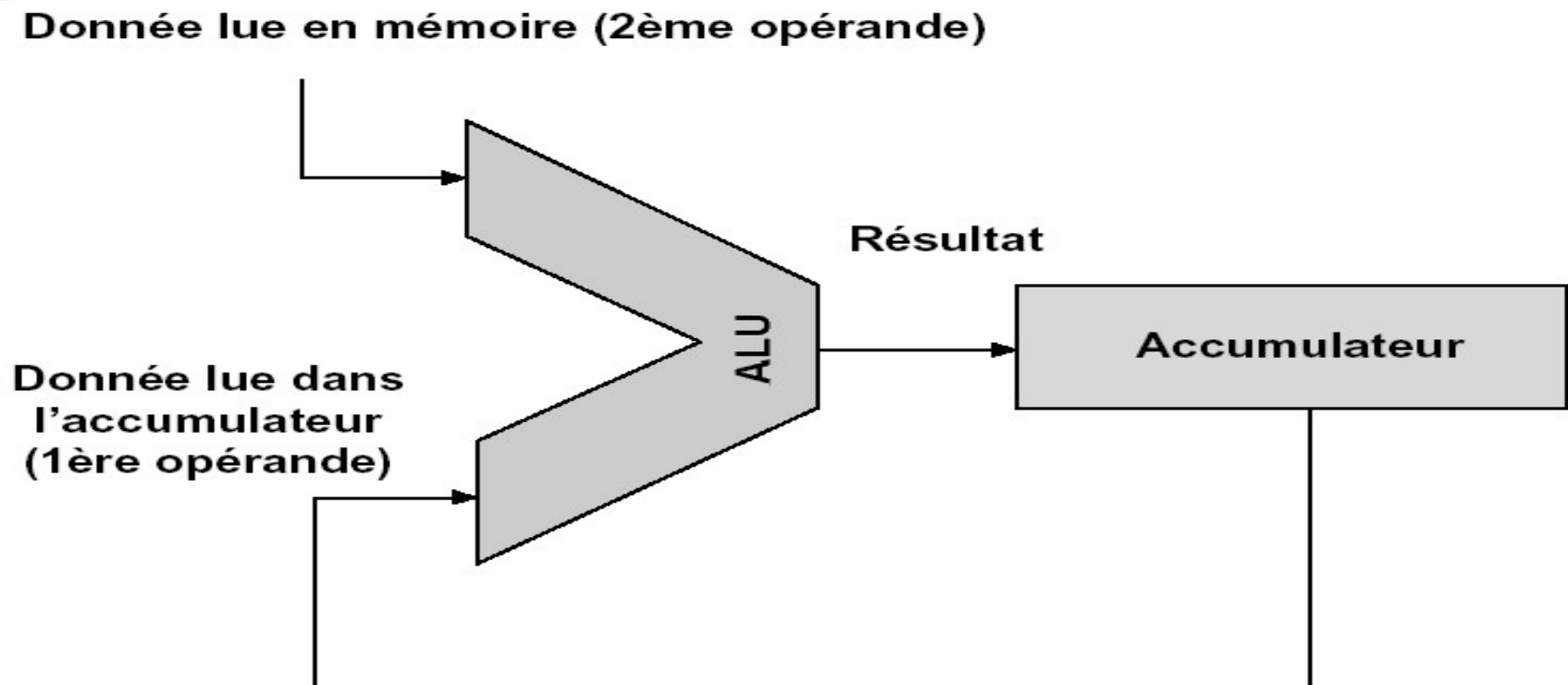
# Registres (1/4)

- ❑ Mémoires RAM internes au  $\mu$ p d'accès rapide
  
- ❑ Compteur ordinal (PC ou IP)
  - Adresse de la prochaine instruction à exécuter
  
- ❑ Registre d'instruction :
  - Code de l'instruction à exécuter

# Registres (2/4)

## □ Accumulateur :

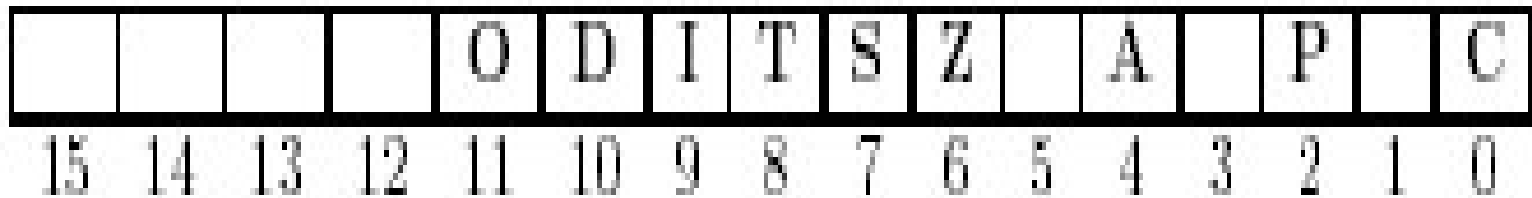
- Contient au début une opérande de l'opération et le résultat à la fin



# Registres (3/4)

## ❑ Registre d'état (PSW : Program Status Word)

- Contient des bits indicateurs d'état du  $\mu p$  (flags/drapeaux)
- Les flags sont actualisés en fonction du résultat de la dernière instruction (ex : 8086)



- Ce registre est utilisé par les instructions de saut conditionnel



# Registres (4/4)

## □ Pointeur de pile (SP)

- Contient l'adresse de la dernière case utilisée ou de la prochaine case libre de la pile
- La pile = zone mémoire réservée au stockage temporaire de données utiles au déroulement du programme

## □ Registres temporaires

- Utilisés par le  $\mu p$  pour le stockage temporaire des adresses ou données lors du déroulement d'instructions



# Décodeur et séquenceur

- ❑ Décodeur (ensemble de circuits)
  - Décode le code opératoire en une séquence de commandes et envoie les signaux correspondants à l'UAL
- ❑ Séquenceur
  - Dirigé par l'horloge, il synchronise les étapes d'exécution d'une instruction
  - Il gère chaque étape et la transforme en signaux de contrôle

# BUS

- ❑ Bus : systèmes de câblage pour lier et faire communiquer les composants d'un ordinateur
  - Fils de transmission d'informations (données, adresses ou commandes)
- ❑ 1 fil transmet un bit, 1 bus à n fils = bus n bits
- ❑ Types :
  - Séquentiels : 1 seul fil qui transmet bit par bit
  - Parallèles : transmission simultanée de +eurs bits
- ❑ Fonctions : Bus d'adresses, de données et de contrôle
  - L'espace mémoire adressable dépend de la largeur du bus d'adresses



# Outils logiciels

- ❑ Comment dialoguer avec l'ordinateur ?
  - Système d'exploitation (SE) : 1<sup>er</sup> logiciel à installer
  - Exemples : Unix, linux, MsDos...
  
- ❑ Comment traiter l'information ?
  - Différents logiciels : bureautique, comptabilité, jeux, applications...
  - Langages de programmation : Langage machine, de bas niveaux, évolués





## Chapitre 2

# ARCHITECTURE DE L'ORDINATEUR DE L'ANTIQUITÉ AUX ANNÉES 40

1. Motivation
2. Boulier
3. Règles de calcul, Pascaline, machines à différences et analytique
4. Mark I
5. ABC et ENIAC
6. Machine de Von Neumann
7. EDVAC, UNIVAC et EDSAC
8. Premières machines commercialisées (IBM 701...)
9. Facteurs ayant influencé l'architecture des ordinateurs

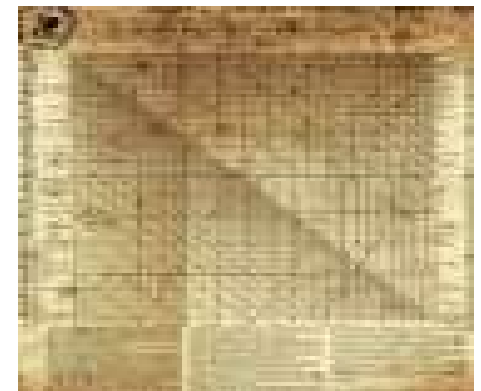


# Motivation

- ❑ Le besoin de calculer remonte au début de la société humaine
- ❑ L'homme utilisait des cailloux (calculus) et ses doigts pour compter
- ❑ L'homme était lent et se trompait souvent
- ❑ De nouveaux outils pour simplifier et accélérer le calcul étaient nécessaires

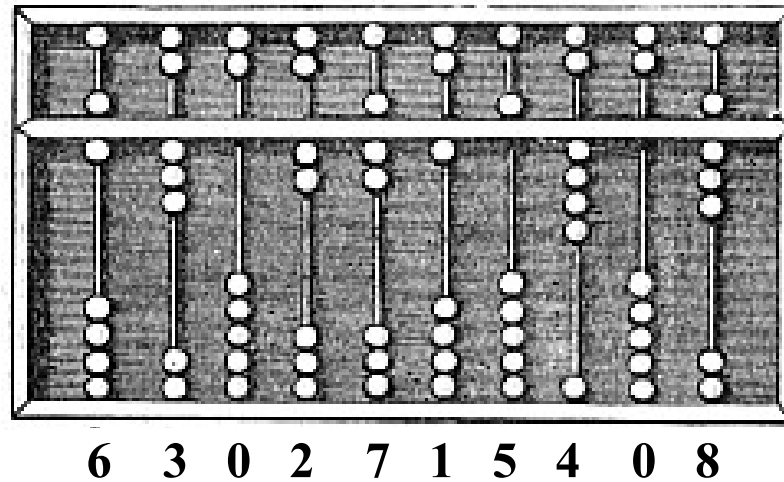
# Antiquité (1/3)

- Différentes civilisations ont inventé des bases de numérotation et des méthodes de calcul
  - Octogone à trigramme (-3000 en chine) : représentation binaire des huit 1<sup>ers</sup> chiffres par des traits interrompus ou non
  - Abaque (abacus) : table de calcul
    - Types : Chinois, grec et romain (sable, jetons...)
    - Boulrier en est un descendant



# Antiquité (2/3)

Boulier



- ❑ Ensemble de boules coulissantes sur des tiges
- ❑ Les boules d'une tige indiquent un nombre de 0 à 15 et représentent une unité, une dizaine...
- ❑ La partie inférieure (supérieure) d'une tige supérieure (inférieure) représente un multiple de 5 (une unité)
- ❑ outil servant à calculer : addition, soustraction, multiplication, division, racine carrée...



## Antiquité (3/3)

-1750 **Code d'Hammourabi** : le roi de Babylone a fait graver les sentences royales sous la forme :

SI *{personne}* ET *{action}* ALORS *{sentence}*

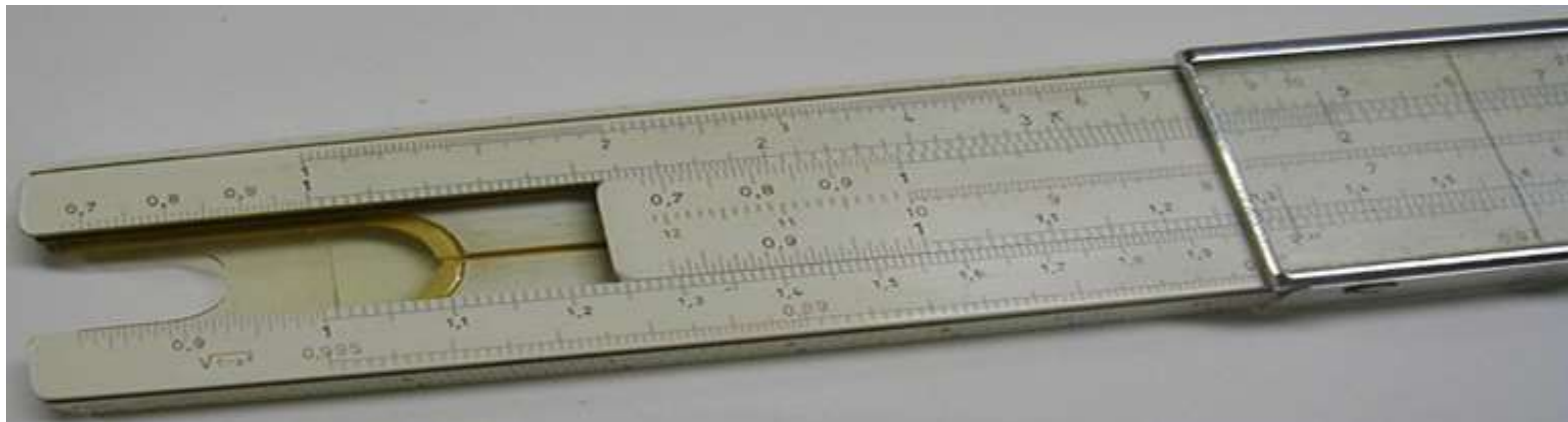
-330 **Logique** : définie par le philosophe grec Aristote

+ 820 **Travaux du mathématicien** Perce Abou Jaafar Mohammed Ibn Moussa Al Khawarizmi connu pour son livre "Al Jabr oua El Mokabala" écrit à l'an 825

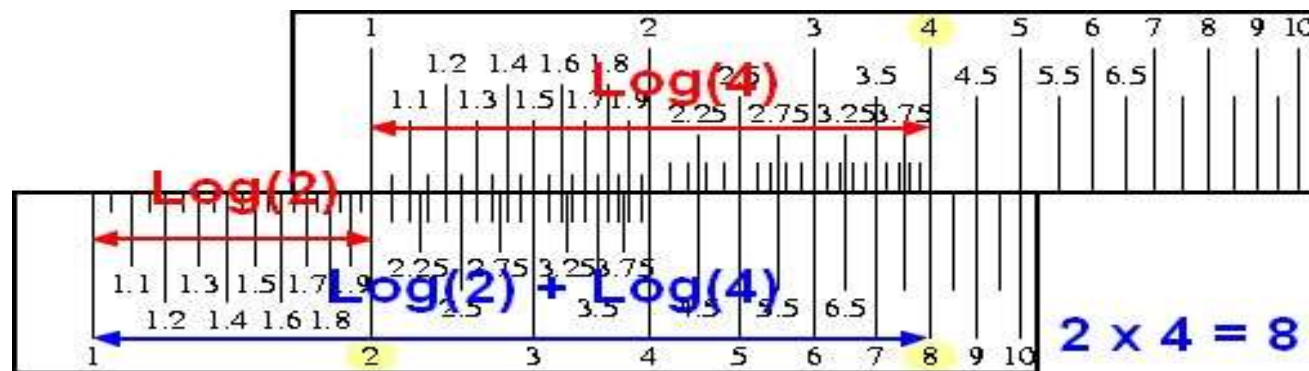
+1000 **Zéro** : Inventé en Inde, rapporté en Occident par les arabes et accepté en occident vers le XIVème siècle

# Calculateurs mécaniques

- ❑ Règle de calcul : W. Oughtred et E. Gunter en 1620
  - Après l'invention du logarithme par J. Napier en 1614



- ❑ Utilisée dans la multiplication :  $\log(a*b) = \log(a) + \log(b)$



# Pascaline

- ❑ Blaise Pascal (1642-France)
  - Machine à base de roues à ergot
  - Utilisée pour des additions et soustractions



B. Pascal

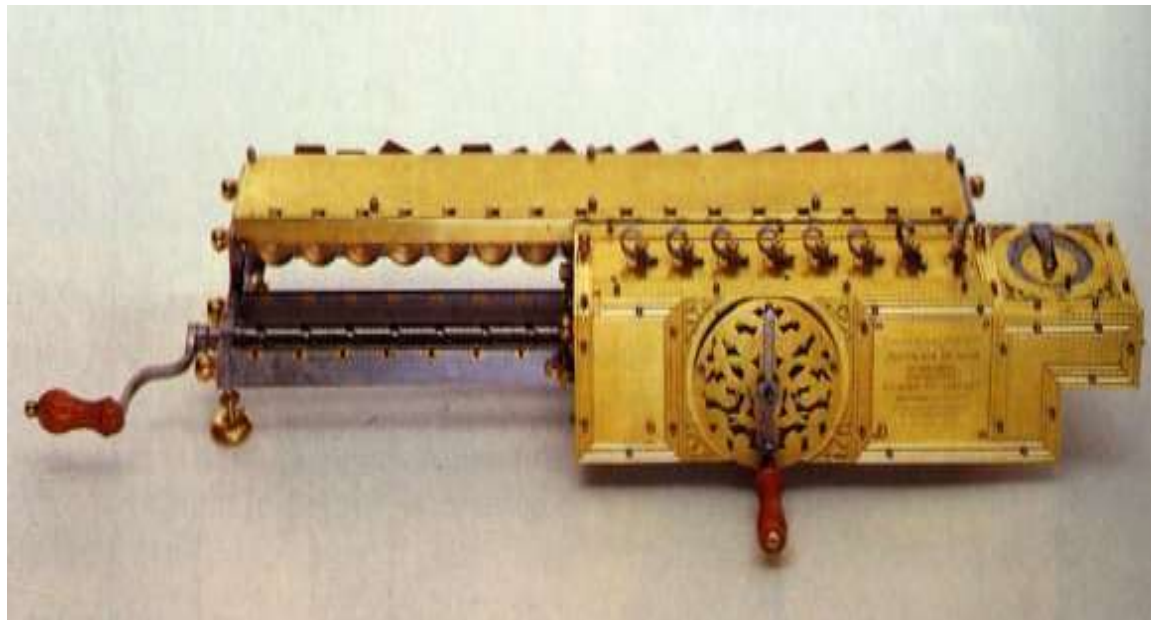


# Pascaline améliorée

- ❑ Construite par Leibniz (1673-Allemagne)
- ❑ Utilise des cylindres à dents de longueurs inégales
- ❑ Calcule les opérations  $+$ ,  $-$  et  $*$



G. W. Leibniz





# Machine à différences (1/2)

## ❑ Calculateur mécanique

- Roues dentées sur des tiges + manivelles
- Inventée par Charles Babbage (1823)
- Construite en 1855 à Paris
  - Évalue des polynômes de 6<sup>ème</sup> degré
  - 33 à 44 nombres de 32 chiffres par minute

## ❑ Utilité : tables mathématiques et nautiques (astronomie + marine)



C. Babbage



Machine à différences

# Machine à différences (2/2)

## ❑ Idée :

- Approximer une fonction continue par un polynôme
- Évaluer un polynôme à partir de tables de différence

## ❑ Exemple :

$$f(n) = n^2 + n + 41$$

$$d1(n) = f(n) - f(n-1) = 2n$$

$$d2(n) = d1(n) - d1(n-1) = 2$$

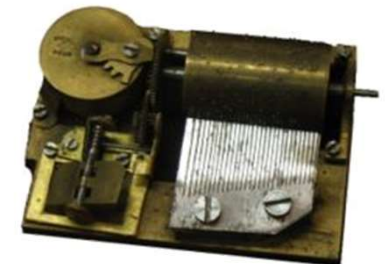
$$f(n) = f(n-1) + d1(n)$$

$$= f(n-1) + (d1(n-1)+2)$$

n	0	1	2	3	4	...
d2(n)			2	2	2	...
d1(n)		2	4	6	8	...
f(n)	41	43	47	53	61	...

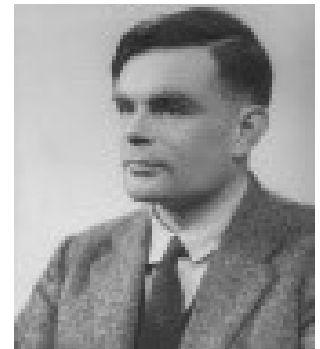
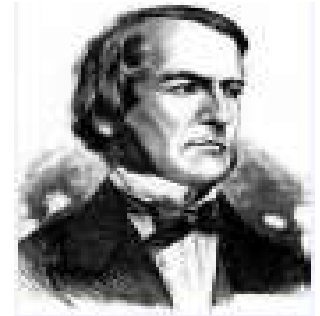
# Machine analytique (1842)

- ❑ Précurseur du calculateur numérique
- ❑ Utilise des cartes perforées
  - Inspirées du métier à tisser de Jacquard
- ❑ Composants :
  - Un magasin (mémoire) : Cartes des variables et résultats intermédiaires
  - Un moulin (unité de calcul) : Cartes d'opérations
- ❑ Cette machine n'a pu être réalisée
  - Augusta Ada (1843) : Description de la machine, 1ers algorithmes, boucles et branchements



# Grands noms (1/2)

- ❑ 1854 : George Boole démontre que tout processus logique est décomposable en opérations logiques appliquées sur 2 états
- ❑ 1904 : J. A. Fleming invente le tube à vide et diode
- ❑ 1937 : A. M. Turing invente la Machine de Turing



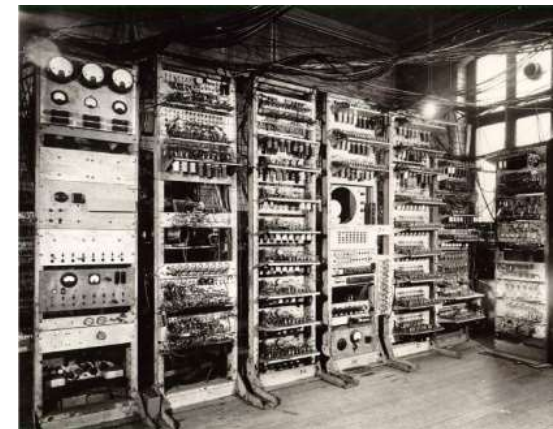
# Grands noms (2/2)

- ❑ 1938 : C. E. Shannon fait le parallèle entre les circuits électriques et l'algèbre booléenne et définit le bit
- ❑ 1945 : Von Neumann définit un modèle formel d'un calculateur (Machine de Von Neumann)



# Calculateurs électroniques

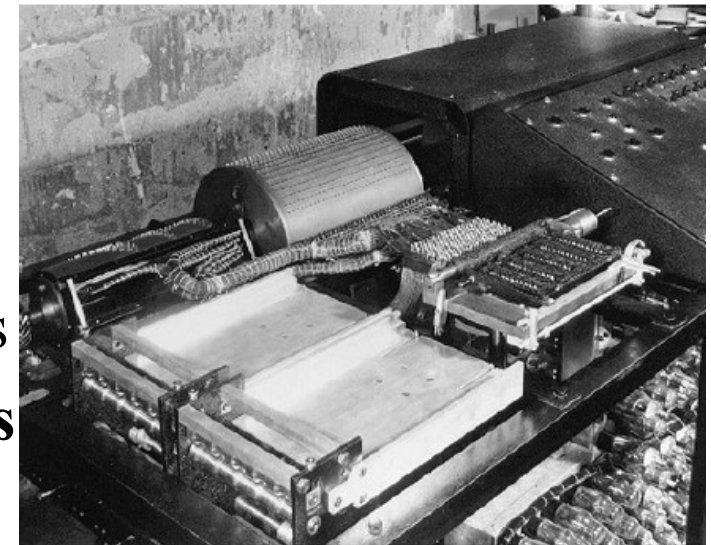
- ❑ **Mark I** Créé en 1944 par Howard Aiken (université de Harvard) chez IBM
- ❑ Caractéristiques :
  - Arbres mécaniques+**relais électromagnétiques**
  - **5 tonnes**, 750000 composants
  - 1 horloge de 100 Khz
  - Des calculateurs en parallèle, calcul **décimal**
- ❑ Performances :
  - 3 additions ou soustractions/s
  - Multiplication : 6 s, Division : 15,3 s
  - Logarithme/fonction trigonométrique : 1 min





# Machine ABC

- ❑ Créé en 1939 par John Atanasoff et son étudiant Clifford Berry (univ. de Iowa)
- ❑ Idée : Utiliser une machine numérique
- ❑ Caractéristiques :
  - 300 Tubes à vide+relais électromécaniques
  - 320 kg ; 1,5 km de fils
  - 1<sup>er</sup> à faire des calculs en **binaire**
  - Utilise l'algèbre de Boole
  - 30 additions/s ; 1 multiplication/s
- ❑ Résout des **équations différentielles**



# ENIAC (1/3)

## (Electronic Numerical Integrator and Computer)



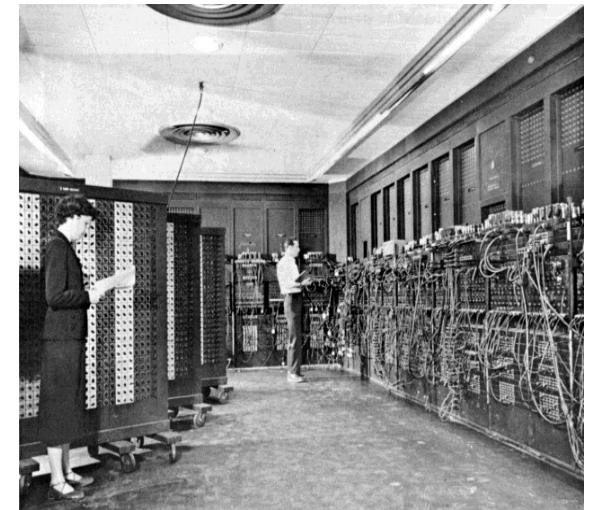
- ❑ Créé par le chercheur John. H. Mauchly et l'ingénieur Presper Eckert en 1943-1945 (univ. de Pennsylvania)
- ❑ Usage : **calculs balistiques**
  - Calcule la trajectoire d'un projectile en 20 s au lieu de 3 jours de calcul manuel



# ENIAC (2/3)

## ❑ Caractéristiques :

- 1<sup>er</sup> ordinateur moderne **non mécanique**
- 18000 Tubes, Lecteur de **cartes perforées**, imprimante électrique, 6000 commutateurs connectables
- **30 tonnes**, Forme en U de 6m et 12m
- **20 calculateurs en parallèle**
- Calcul décimal
- **5000** additions/s, 1 division en 6ms
- 120 cartes lues/min



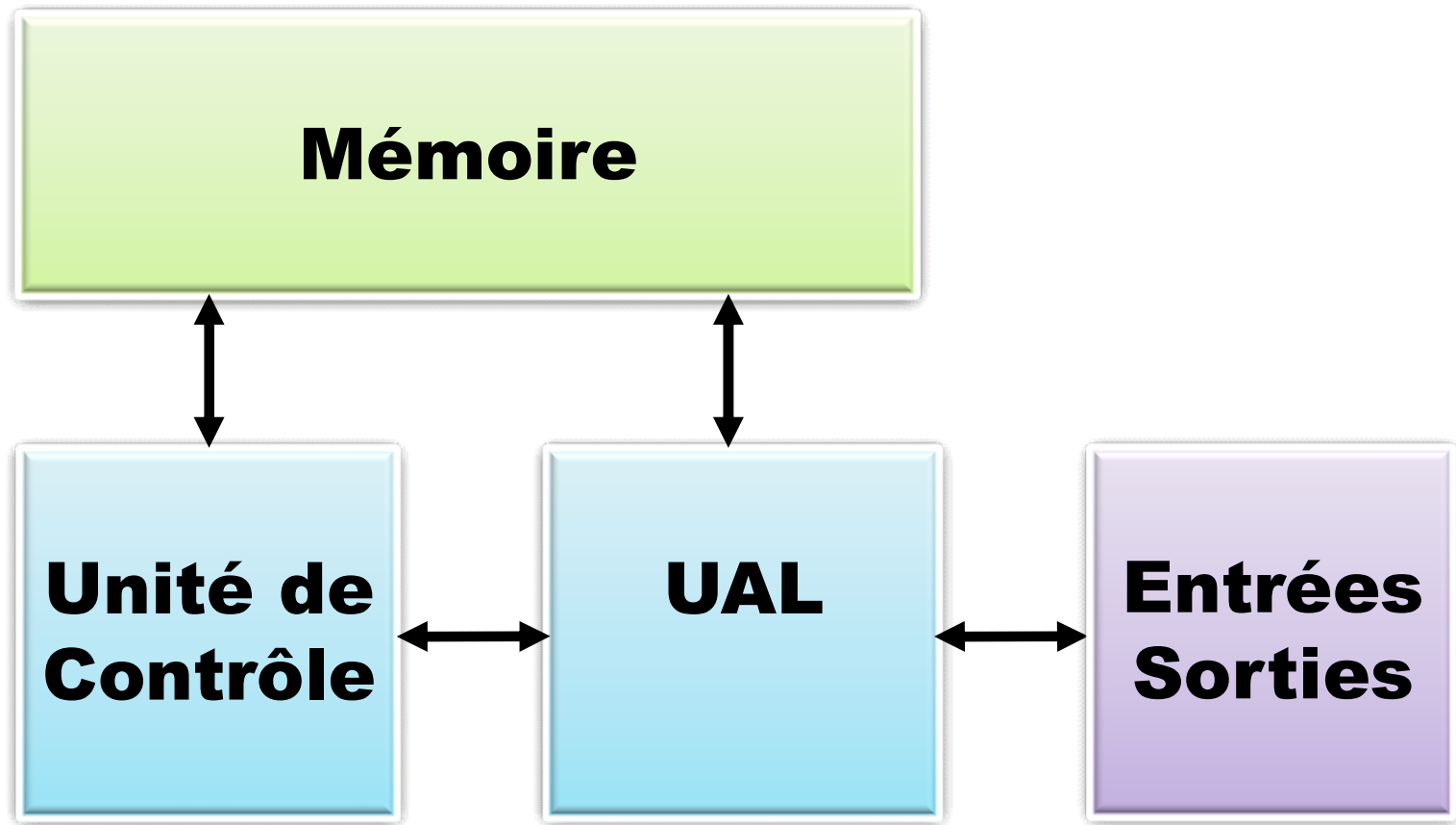
ENIAC

# ENIAC (3/3)

## ❑ Limitations

- Fiabilité: MTBF (Mean Time Between Failures) est de 20 mn
- Difficulté d'appeler un programme à partir d'un autre programme
- Exécution d'instructions selon un ordre prédéterminé
- Intervention manuelle pour rompre la séquence selon des résultats précédents (cas if else)

# Machine de Von Neumann (1945)





# Description de la machine de Von Neumann

- ❑ **UAL** : effectue les calculs
- ❑ **UC** : commande les autres unités
  - Envoie des signaux de contrôle aux autres unités
  - Supervise le fonctionnement de l'UAL
  - Envoie des signaux d'horloge aux autres unités...
- ❑ **Mémoire** : dispositif de stockage de données et programme
- ❑ **E/S** : permettent l'échange d'informations avec les dispositifs extérieurs



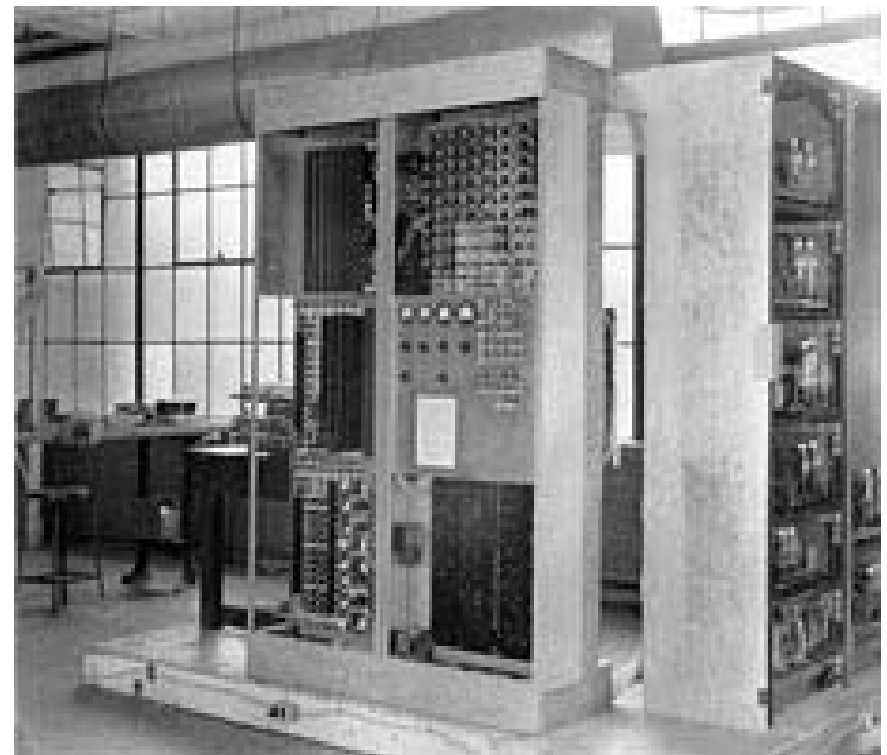
# Principes de la machine de Von Neumann

- ❑ Machine universelle contrôlée par un programme
- ❑ Données et programme en mémoire (binaire)
- ❑ Exécution séquentielle par défaut
- ❑ Possibilité de tests, boucles et sauts conditionnels
- ❑ Architecture SISD (Single Instruction Single Data)
  - Une UC traite une séquence d'instructions
  - Une UAL traite une séquence de données

# EDVAC

(Electronic Discrete Variable Automatic Computer)

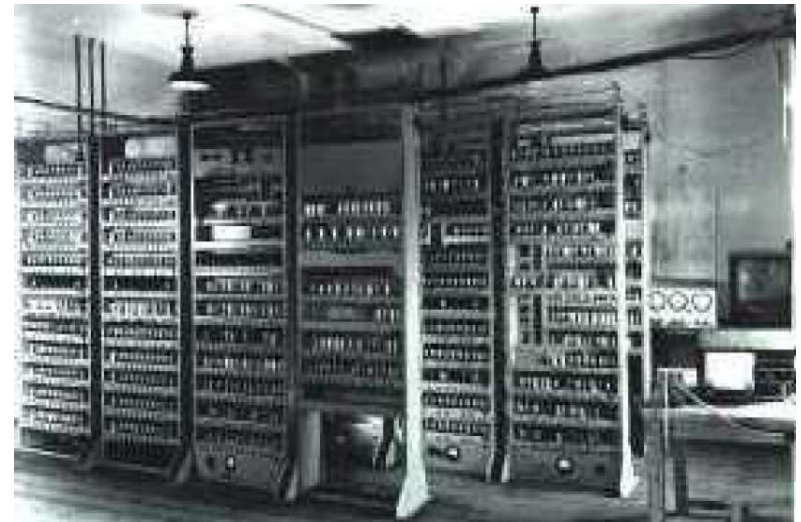
- ❑ EDVAC (1946) : Amélioration de l'ENIAC par l'aide de Von Neumann (**Autoséquencement** au lieu d'opératrices)
- ❑ Idée : **Enregistrer le programme en mémoire**
- ❑ Caractéristiques :
  - 2000 tubes,
  - 1 unité de calcul,
  - Mémoire de 200 mots.



EDVAC

# EDSAC

- ❑ Créé par Maurice Wilkes en 1950
- ❑ Inspiré du draft report de V. Neumann (EDVAC)
- ❑ Caractéristiques :
  - **6 fois + petit** que l'ENIAC
  - Mémoire de lignes à retard au mercure de 512 mots de 17 bits
  - 1 addition : 1,4 ms, 1 multiplication : 5,4ms

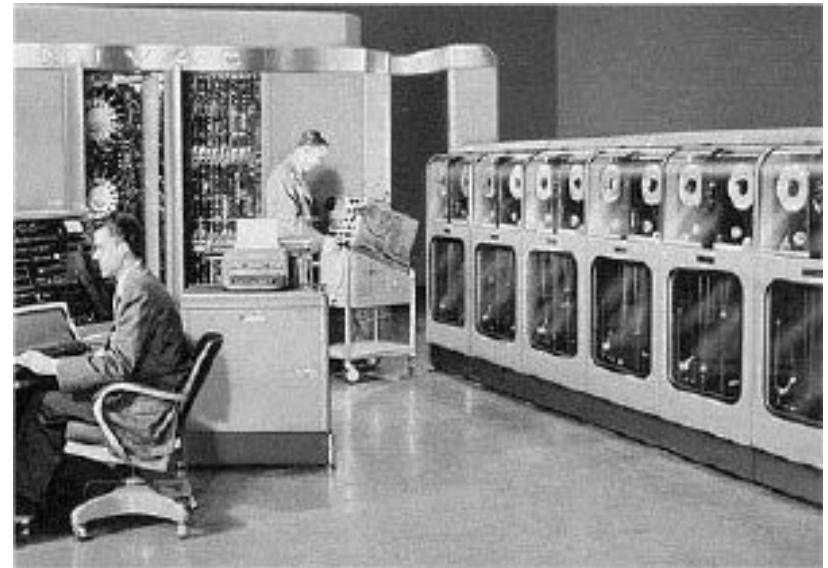


EDSAC

# 1<sup>ères</sup> machines commercialisées

## ❑ UNIVAC : 1er ordinateur commercialisé (1951)

- 56 exemplaires vendues  
dont 19 pour l'armée



## ❑ Caractéristiques :

- 5000 tubes
- **Bandes magnétiques** au lieu des cartes perforées
- 1 Addition en 0,5ms et 1 multiplication en 2,5 ms



# SSEC

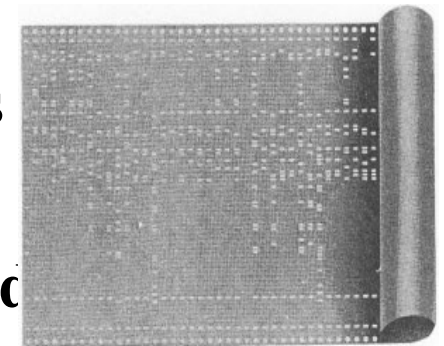
## (Selective Sequence Electronic Calculator)



- ❑ Créé par Wallace Eckert chez IBM en 1948
- ❑ Populaire et **utilisé en privé** (300\$/heure)
- ❑ Utilité : tables de positions de la lune (projet Apollo), Physique nucléaire et Bombe hydrogène

# Caractéristiques du SSEC

- ❑ 21400 relais, 13500 tubes
- ❑ 150 mots dans une mémoire à relais
- ❑ Bandes perforées de données ou d' instructions
- ❑ 66 lecteurs de bandes
- ❑ **Boucle par collage des extrémités d'une bande**
- ❑ 3 phases de calcul :
  - Produire une carte de résultats intermédiaires
  - En coller les extrémités et la placer dans un lecteur
  - Lire les résultats autant de fois qu'il le faut



Bande perforée

# IBM 701

- ❑ Inspiré de l'IAS
- ❑ Usage : **opérations scientifiques**
- ❑ 30 machines vendues en 1953-54
- ❑ Caractéristiques :
  - Machine **binaire**
  - **Mémoire principale** à tubes de 2048 mots de 36 bits
  - **Mémoire secondaire** à tambour de 8192 mots
  - Lecteurs de cartes perforées et bandes magnétiques (1 bande = 1500 cartes)
  - 16000 additions/s , 2000 multiplications/s



IBM 701

# IBM 704



- ❑ Lancé par IBM en avril 1955
- ❑ 1<sup>er</sup> ordinateur commercial effectuant des **calculs flottants**
- ❑ Mémoire à tores de ferrite de 32768 mots de 36 bits
- ❑ 40 000 instructions/s
- ❑ 123 machines vendues jusqu'en 1960

# IBM 650 (1954-1962)

- ❑ 1<sup>er</sup> ordinateur fabriqué en série
- ❑ Plus de 1000 machines vendues
- ❑ Usage : **opérations commerciales**
- ❑ Caractéristiques :
  - Machine à tubes, **900 Kg**
  - **Mémoire à Tambour** de 4000 mots
  - **Mémoire à ferrite** de 60 mots pour communiquer avec ses périphériques plus lents
  - Addition : 1,63 ms, Multiplication : 12,96 ms, Division : 16,90 ms





# IBM Leader du marché

- ❑ IBM possédait la plus grande part du marché
- ❑ Croissance exceptionnelle durant les années 50 :
  - Nombre d'employés passé de 30000 à 100000
  - Revenus multipliés par cinq : de 266 millions \$ à 1613 millions \$
- ❑ IBM a dominé le marché durant les années 60 grâce au système **IBM/360**



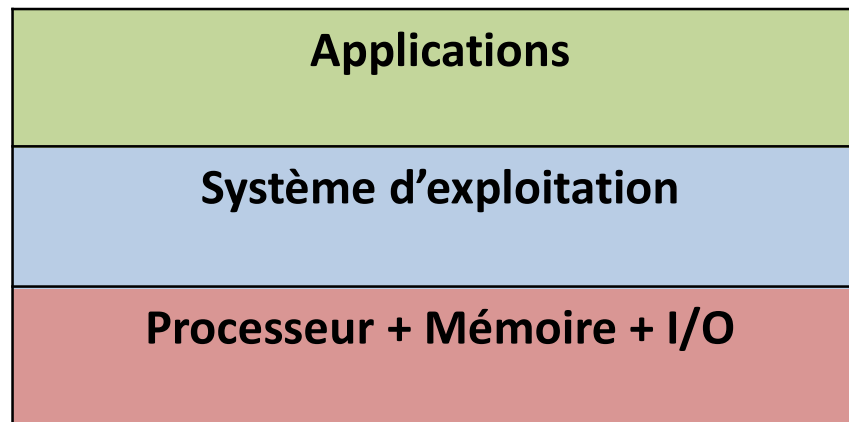
# Facteurs ayant influencé l'architecture des ordinateurs (1/3)

## ❑ Technologie :

- Transistors, Circuits intégrés, VLSI, Mémoire Core, ROM, RAM, Bandes magnétiques, Disques, CD, DVD...
- Révolution des microprocesseurs (depuis 1990)
  - Important investissement humain et financier (Pentium Pro : 500 ingénieurs, Itanium : 1000 ingénieurs)
  - Montée de la vitesse d'horloge et du rendement
  - Baisse des prix à 1 dixième

# Facteurs ayant influencé l'architecture des ordinateurs (2/3)

- ❑ Compatibilité d'architecture des jeux d'instructions et portabilité
  - Possibilité d'exécution de programmes sur différents modèles compatibles (ex : IBM 360/370, Intel x86...)







# Facteurs ayant influencé l'architecture des ordinateurs (3/3)

## ❑ Logiciel (Software) :

- Nécessité de satisfaire les besoins des concepteurs software et les exigences des concepteurs du matériel (hardware)
- Développement de micro-mécanismes pour réaliser des mécanismes abstraits demandés en logiciel
- Elaboration de langages et stratégies de compilation respectant des mécanismes pris en compte pour une performance matérielle



## Chapitre 3

# ARCHITECTURE ET ÉVOLUTION DE L'ORDINATEUR DANS LES ANNÉES 50

1. Machines de début des années 50
2. Evolution technologique
3. Langages de programmation
4. Evolution des langages de programmation
5. Format, exécution et Jeu d'instructions
6. Modes d'adressage
7. Type de langages machine
8. Instructions d'une machine à accumulateur
9. Machine à accumulateur
10. Machine à registres (IBM 360)

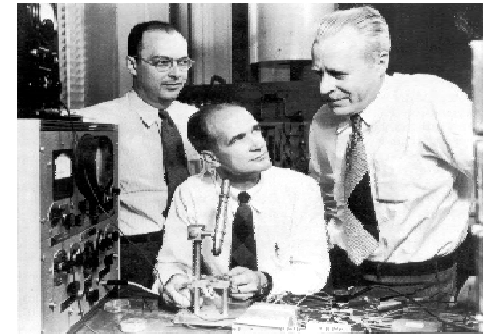


# Les machines du début des années 50

- ❑ Coût élevé du matériel (hardware)
- ❑ Mémoires de petite capacité (~1000 mots)
- ❑ Temps d'accès à la mémoire était de 10 à 50 fois inférieur au cycle du processeur
- ❑ Des circuits de commande d'exécution complexes
- ❑ Difficulté de programmation :
  - Dépendance de l'architecture de la machine
  - Programmation binaire (0 et 1)

# Evolution technologique

- ❑ Transistors (1947)
  - Créés par Bardeen, Brattain et Shockley
  - Plus petits, moins chers, plus fiables
- ❑ Circuits imprimés
- ❑ Mémoire magnétique, mémoire à tore
- ❑ Révolution du microprocesseur
- ❑ Premier ordinateur à transistors  
TRADIC de bell (1956)



Bardeen, Brattain et Shockley



TRADIC



# Langages de programmation

## ❑ Définition :

- Langage formel servant à l'écriture de programmes exécutables par l'ordinateur

## ❑ Catégories :

### – **Langages de bas niveau :**

- Langages machine
- Langages d'assemblage

### – **Langages de haut niveau ou évolués**

- Fortran, Basic, Pascal, C, C++, Visual Basic, Visual C++, Java...

# Exemples d'instructions

Langage machine	Langage d'assemblage	Commentaire
101001011100111	MOV A,103	transférer le contenu de l'adresse 103 à A
111100001011101	ADD A,B	ajouter le contenu de A à B



# Langages de bas niveau

- ❑ Etroitement liés à l'ordinateur utilisé
- ❑ Difficiles à lire et à écrire (des 0 et 1)
- ❑ Fortement exposés aux erreurs
- ❑ Programmes directement exécutables par la machine ou sont à assembler

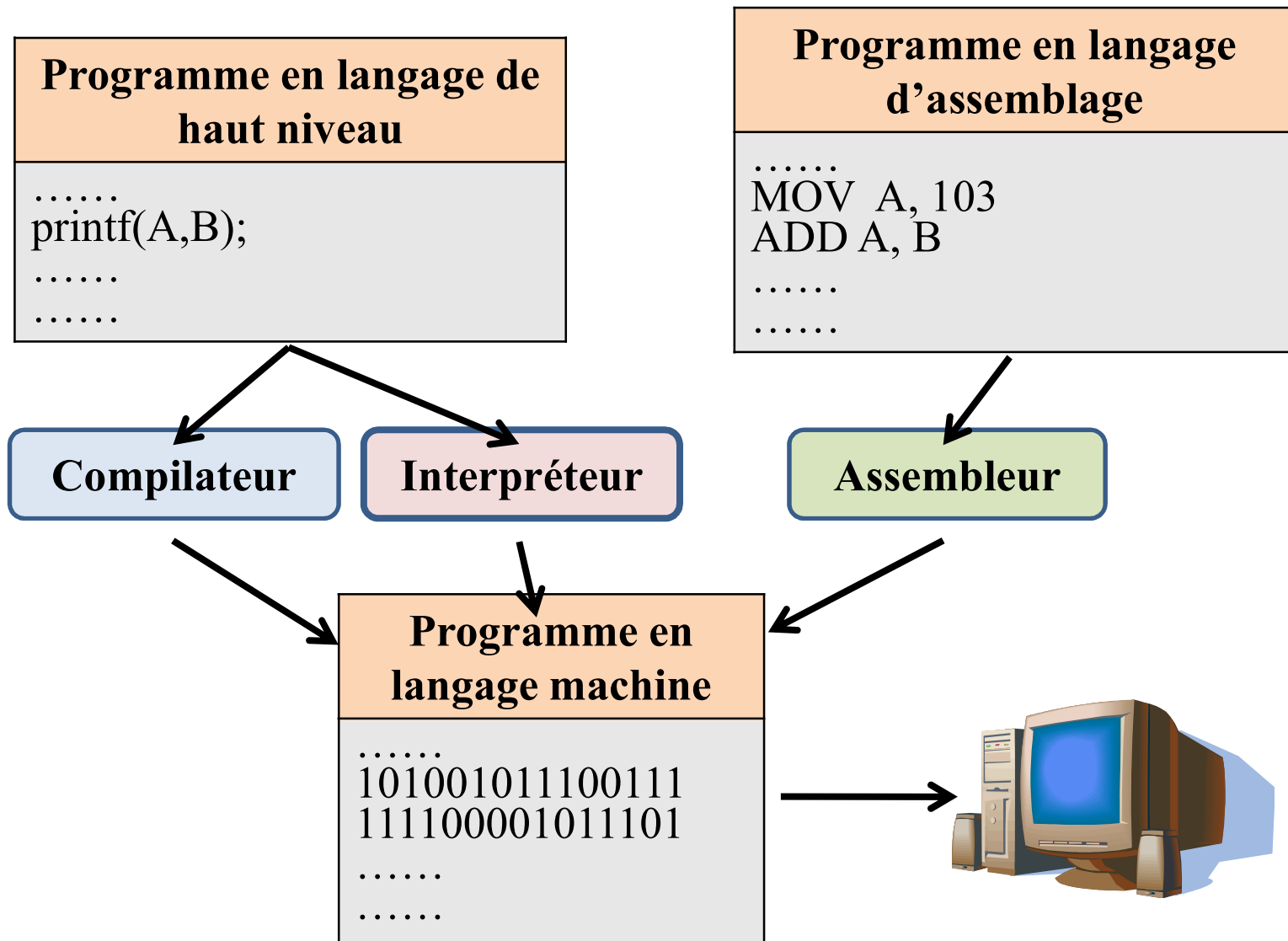


# Langages de haut niveau

- ❑ Indépendants de l'ordinateur (programmes portables)
- ❑ Faciles à utiliser (Instructions proches de la langue naturelle)
- ❑ Programmes facilement compréhensibles mais sont à compiler ou à interpréter



# Programmation



# Evolution des langages de programmation

- ❑ Langage binaire
- ❑ Codes mnémoniques (EDSAC)
- ❑ Langages de haut niveau et compilateurs :
  - FORTRAN, FORMula TRANslator (1957)
    - Créé par John Backus chez IBM
    - Fortran Monitor System (1958)
  - LISP (1956), l'assembleur (1958, M. Wilkes)
  - Algol (1959)
- ❑ Bibliothèques de routines (depuis 1955)
  - Virgule flottante, matrices, fonctions...



J. Backus



M. Wilkes



# Processeur et programmation

- ❑ Chaque processeur possède son langage machine
  - Ecrire un programme (instructions) exécutable
- ❑ Chaque processeur définit :
  - Un jeu d'instructions qu'il sait exécuter
  - Des modes d'adressage : manières d'accès à la mémoire

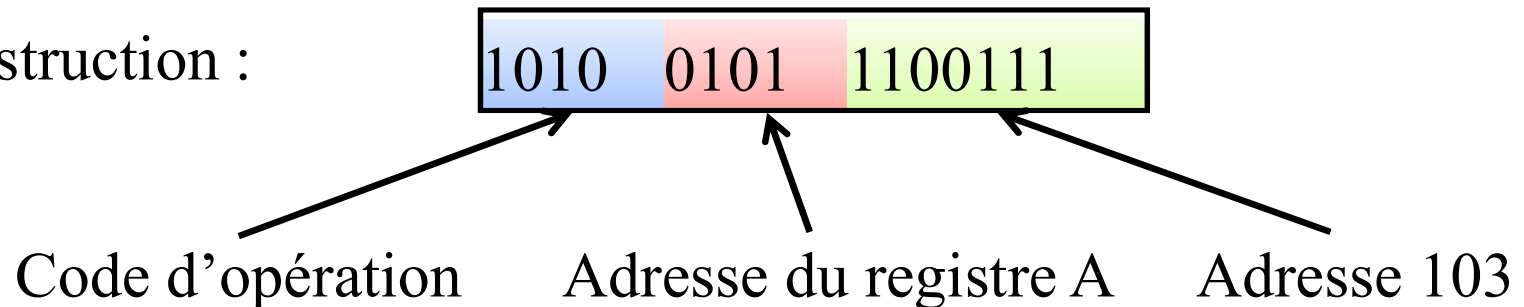
# Format d'instruction

- ❑ Format de stockage des instructions en mémoire
  - Code opératoire + opérandes

- ❑ Exemple :

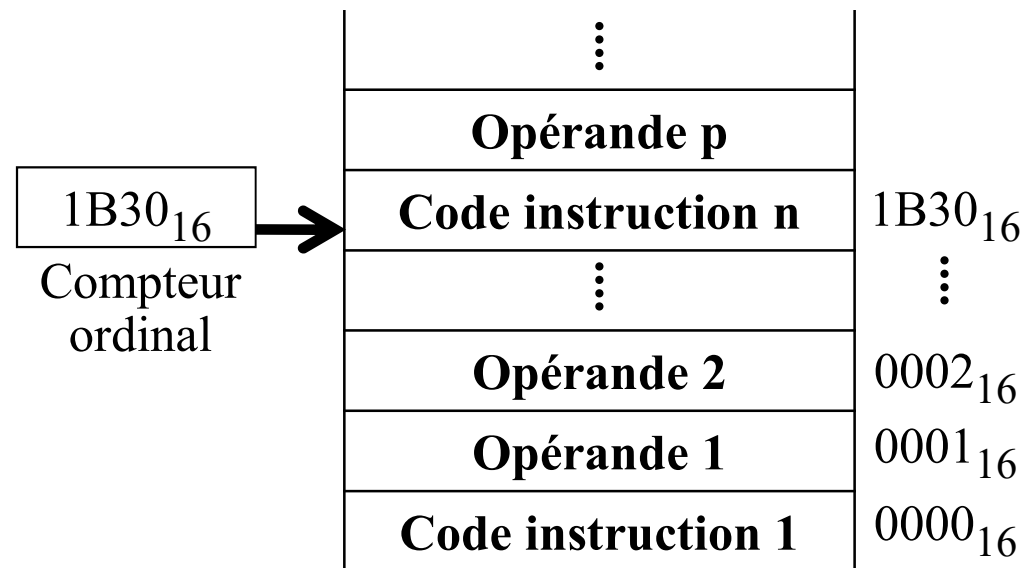
101001011100111 ; transférer le contenu de  
; l'adresse 103 à A

Instruction :



# Représentation d'une instruction en mémoire

- Une instruction est codée sur 1 ou +ieurs octets en mémoire





# Jeu d'instructions machine

- ❑ Transfert de données entre la mémoire et les registres
  - MOVE, LOAD, STORE...
- ❑ Opérations arithmétiques et logiques
  - ADD, SUB, MUL, DIV, AND, OR, NEG, NOT...
- ❑ Décalages et rotations
  - LLS (à gauche, remplacement par 0), LRS (à gauche par rotation)...
- ❑ Ruptures de séquence (contrôle d'aiguillage)
  - BR, JUMP, JZ, JGT ( $>0$ ), JLT ( $<0$ )...
- ❑ Entrées/Sorties (si l'espace des E/S est indépendant)
  - IN, OUT...

# Modes d'adressage (1/2)

Définition : manière ou chemin d'accès à l'adresse effective

- ❑ Mode immédiat : donnée directement dans l'instruction
  - `MOVI R1, 4 ; R1  $\leftarrow$  4`
  - `ADD R1, R2, #3 ; R1  $\leftarrow$  R2 + 3`
- ❑ Mode direct ou absolu : donnée dans un opérande
  - `MOVE R1, R2 ; R1  $\leftarrow$  (R2) l'opérande est un registre`
  - `MOVE R1, 4 ; R1  $\leftarrow$  (4) l'opérande est une adresse`

# Modes d'adressage (2/2)

- ❑ Mode indirect : donnée dans une adresse contenue dans l'opérande
  - MOVE R1, (R2) ;  $R1 \leftarrow ((R2))$  l'opérande est un registre
  - MOVE R1, @4 ;  $R1 \leftarrow ((4))$  l'opérande est une adresse
- ❑ Mode indexé : donnée dans une adresse calculée à partir d'un déplacement et du contenu d'un registre
  - MOVE R1, 10(R2) ;  $R1 \leftarrow (10 + (R2))$
  - MOVE R1, Table(R2) ;  $R1 \leftarrow (Table + (R2))$



# Types de langages machine (1/2)

- ❑ Considérons l'instruction :  $A = B + C$
- ❑ Machine à plusieurs opérandes (Vax 11) :
  - `ADDW3 B,C,A` ;  $A \leftarrow (B) + (C)$
- ❑ Machine à 2 opérandes (PDP 11) :
  - `MOVE B,A` ;  $A \leftarrow (B)$
  - `ADD C,A` ;  $A \leftarrow (A) + (C)$
- ❑ Machine à un opérande (Vax 11)
  - `LDM B` ; Charger B dans l'accumulateur
  - `ADD C` ; Ajouter C à l'accumulateur
  - `STM A` ; Stocker l'accumulateur dans A

# Types de langages machine (2/2)

- ❑ Considérons l'instruction :  $A = B + C$
- ❑ Machine à registres (Cyber 170)
  - EA1 B ;  $X1 \leftarrow (B)$
  - EA2 C ;  $X2 \leftarrow (C)$
  - IX6  $X1 + X2$  ;  $X6 \leftarrow (X1) + (X2)$
  - SA6 A ;  $A \leftarrow (X6)$
- ❑ Machine à pile (HP 3000)
  - Utilise la pile pour y stocker les opérandes d'une instruction
  - LOAD B ; mettre B dans la pile
  - LOAD C ; mettre C dans la pile
  - ADD ; additionner les 2 éléments sommets de la pile et empiler le résultat
  - STORE A



# Instructions

## d'une machine à accumulateur

- LOAD  $x$  ;  $AC \leftarrow M[x]$
- STORE  $x$  ;  $M[x] \leftarrow (AC)$
- ADD  $x$  ;  $AC \leftarrow (AC) + M[x]$
- SUB  $x$
- MUL  $x$
- DIV  $x$
- SHIFT LEFT ;  $AC \leftarrow 2 \times (AC)$
- SHIFT RIGHT
- JUMP  $x$  ;  $PC \leftarrow x$
- JGE  $x$  ; si  $(AC) \geq 0$  alors  $PC \leftarrow x$
- LOAD ADR  $x$  ;  $AC \leftarrow$  extraction du champs adresse de  $(M[x])$
- STORE ADR  $x$

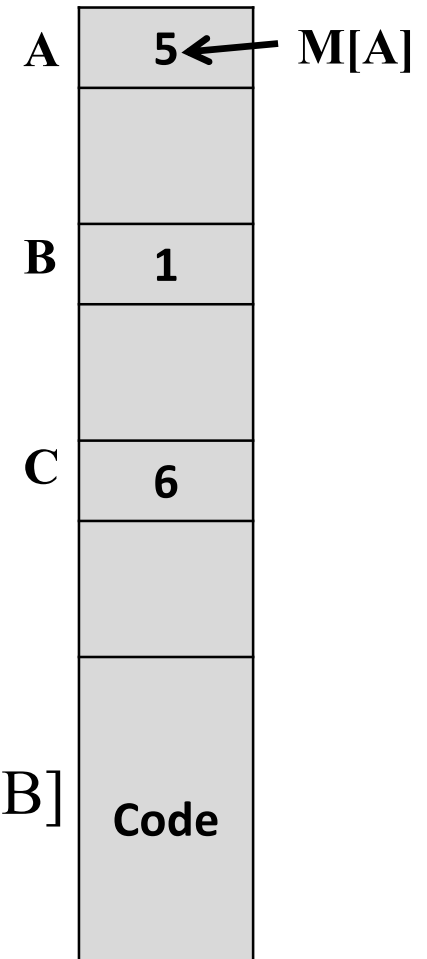
# Programmation d'une machine à accumulateur

## □ Registres

- Zones mémoires ayant un accès rapide
- L'accumulateur en est un exemple

## □ Exemple 1 : Calculer $C \leftarrow A+B$

F1	LOAD A	; $AC \leftarrow M[A]$
F2	ADD B	; $AC \leftarrow (AC) + M[B]$
F3	STORE C	; $M[C] \leftarrow (AC)$
	HLT	; Arrêt du programme



# Programmation d'une boucle

- ❑ Exemple 2 : Calculer  $C_i \leftarrow A_i + B_i$  pour  $i$  de 1 à  $n$

LOOP	LOAD	N	; Nombre d'itérations	
	JGE	DONE	; Terminer si $\geq 0$	
	ADD	ONE	; Incrémentation	
	STORE	N	;	
F1	LOAD	A	; Opérande 1	
F2	ADD	B	; Opérande 2	
F3	STORE	C	; résultat	
	JUMP	LOOP	; aller au début	ONE
DONE	HLT		; Arrêt du programme	

A	5
B	1
C	?
N	- 4
	1
	Code

Comment modifier les adresses A, B et C ?

# Code auto-modifiable

```

LOOP  LOAD  N
      JGE   DONE
      ADD   ONE
      STORE N
F1     LOAD  A
F2     ADD   B
F3     STORE C
  
```

```

LOAD  ADR F1
ADD   ONE
STORE ADR F1
LOAD  ADR F2
ADD   ONE
STORE ADR F2
LOAD  ADR F3
ADD   ONE
STORE ADR F3
  
```

```

JUMP  LOOP
  
```

```

DONE  HLT
  
```

Modification des  
adresses A, B, C

A	5
B	1
C	?
N	- 4
ONE	1
	Code

## Statistiques par itération

Recherche d'instructions 17  
 Recherche d'opérandes 10  
 Ecriture en mémoire 5

# Interblocage Processeur - Mémoire



- ❑ Mémorisation à accès rapide dans le processeur
  - 8 à 16 registres au lieu d'un accumulateur
- ❑ Indexation
  - Eviter la modification du code
- ❑ Instructions complexes
  - Réduction de la recherche d'instruction
- ❑ Instructions compactes
  - Utilisation des adresses des opérandes directement



# Etat du processeur

- ❑ L'information contenue dans le processeur à la fin de l'exécution d'une instruction détermine le contexte du traitement de la prochaine instruction
- ❑ La visibilité de l'état du processeur est très importante dans l'organisation des ordinateurs aussi bien pour le matériel que le logiciel :
  - Exploitation par les logiciels
  - Si une instruction est interrompue, le matériel doit sauvegarder et restaurer l'état du processeur de façon transparente



-

# Utilisation des registres d'index

Exemple : Calculer  $C_i \leftarrow A_i + B_i$  pour  $i$  de 1 à  $n$

	<b>LOADi N, IX</b>		<b>A</b>	<b>5</b>
				<b>2</b>
				<b>1</b>
<b>LOOP</b>	<b>JZi</b>	<b>DONE, IX</b>	<b>LASTA</b>	<b>3</b>
	<b>LOAD</b>	<b>LASTA, IX</b>	<b>B</b>	<b>0</b>
	<b>ADD</b>	<b>LASTB, IX</b>		<b>4</b>
	<b>STORE</b>	<b>LASTC, IX</b>	<b>LASTB</b>	<b>1</b>
	<b>JUMP</b>	<b>LOOP</b>	<b>C</b>	<b>2</b>
				<b>?</b>
<b>DONE</b>	<b>HALT</b>			<b>?</b>
				<b>?</b>
			<b>LASTC</b>	<b>?</b>
				<b>...</b>

## Avantages et Inconvénients :

- Programme non auto-modifiable
- Moins d'opérations par itération
- Instructions avec 1 à 2 bits de plus

### Statistiques par itération

Recherche d'instructions	6	<b>N</b>
Recherche d'opérandes	3	
Ecriture en mémoire	1	

5
2
1
3
0
4
1
2
?
?
?
?
...
- 4
...

# Manipulation des registres d'index

- ❑ L'indexation par registre au lieu de la mémoire réduit le nombre de recherche et de chargement en mémoire

- ❑ De nouvelles instructions sont à prévoir :

**INCi**      **k, IX**                      ;  $IX \leftarrow (IX) + k$

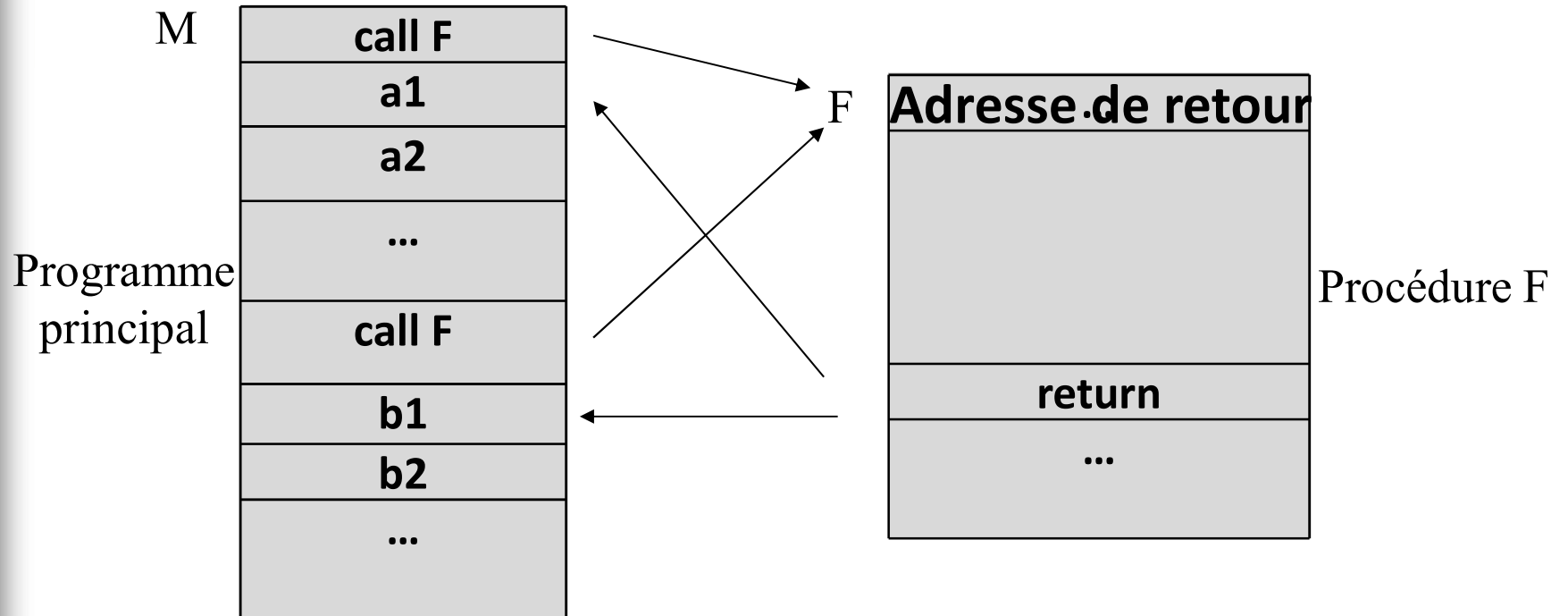
**STOREi** **x, IX**                      ;  $M[x] \leftarrow (IX)$

...

- ❑ Ces instructions permettent de manipuler directement les registres et les rendent similaires à l'accumulateur
- ❑ Plusieurs accumulateurs (registres) et plusieurs registres index

⇒ Registres d'ordre général (general purpose registers)

# Appel de procédure



- ❑ Instruction de branchement à une procédure (Jump to Sub Routine)

M: JSR F ;  $F \leftarrow M + 1$  and Jump to F+1

- ❑ Plusieurs appels imbriqués  $\Rightarrow$  utilisation d'une pile (Stack)



# IBM 360 : Machine à registres d'ordre général

## ❑ Etat du processeur

- 16 registres d'ordre général de 32 bits
  - Peuvent être utilisés comme index ou registre de base
  - Registre 0 a certaines propriétés
- 4 registres de virgule flottante de 64 bits
- 4 registres d'état (PSW : Program Status Word)
- PC, Code de condition, drapeaux d'états

## ❑ Mots mémoire sont de 32 bits

## ❑ Adresse est composée de 24 bits d'adresse

## ❑ Format de données

- Octet (8 bits), demi mot (16 bits), mot (32 bits) et mot double (64 bits)



# Microprocesseur d'IBM S/390 z900

- ❑ Adressage virtuel de 64-bits
  - S/390 est la première conception 64-bit design (version originale de S/360 était de 24-bit, et S/370 était de 31-bits)
- ❑ Fréquence d'horloge de 1.1 GHz
  - 0.18μm CMOS, 7 couches de semiconducteurs (layers)
  - En 2000 des systèmes à 770MHz
- ❑ Pipeline CISC à 7 étages
- ❑ Chemins de données Redondants
  - Toutes les instructions s'exécutent en 2 chemins de données parallèles et leur résultats est comparés
- ❑ 256KB L1 I-cache, 256KB L1 D-cache dans le circuit (on-chip)
- ❑ 20 CPUs + 32MB L2 cache par Module Multi-Chip
- ❑ Refroidissement par eau à 10° C



## Chapitre 4

# ARCHITECTURE ET ÉVOLUTION DE L'ORDINATEUR DANS LES ANNÉES 60

- 1- Evolution technologique et logicielle
2. Machine à pile (Burroughs 5000)
3. Machine à registres généraux (IBM 360)
4. Machine pipeline (CDC6600)
5. Evolution depuis 1965 : Microinformatique

# Evolution technologique



Kilby et Noyce

- ❑ Circuits intégrés :
  - Inventés par Kilby et Noyce en 1959
  - Dizaines de transistors sur une puce de silicium (SSI)
  - Puissance/miniaturisation (+petites,+rapides,—chères)
- ❑ Le coût du matériel a commencé à chuter
  - Mémoires de 32 K mots
  - processeurs spécialisés d'E/S...



# Evolution logicielle

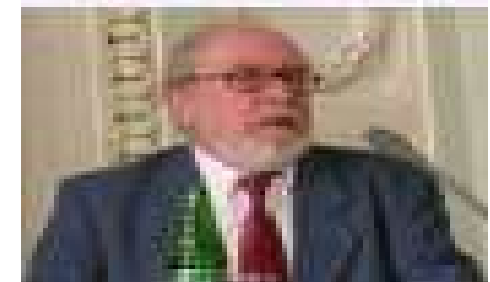
- ❑ Systèmes opératoires :
  - CTSS (1961), Multics (1965),
  - OS/360 (1966),
  - Unix (K. Thompson et D. Ritchie, 1969)
- ❑ Langages de programmation
  - COBOL en 1960
  - BASIC (T. Kurtz et J. Kemeny, 1964)
  - Pascal (Niklaus Wirth, 1969)
- ❑ Apparition du code ASCII (1964)
- ❑ Séparation du modèle de programmation et de l'implantation matérielle  $\Rightarrow$  Machines compatibles (jeux d'instructions identiques et architectures différentes)



Thompson & Ritchie



Kurtz et Kemeny



Niklaus Wirth



# Problématiques des architectes des années 60

- ❑ Définition d'une base stable pour le développement de logiciels
- ❑ Support des systèmes d'exploitation (processus, multi-utilisateurs, I/O...)
- ❑ Implantation des langages de haut niveau (récursivité...)
- ❑ Prise en considération de l'impact des grandes mémoires dans la taille des instructions
- ❑ Organisation de l'état du processeur afin d'être exploité par le programmeur
- ❑ Détermination des architectures pour lesquelles des implantations rapides peuvent être développées

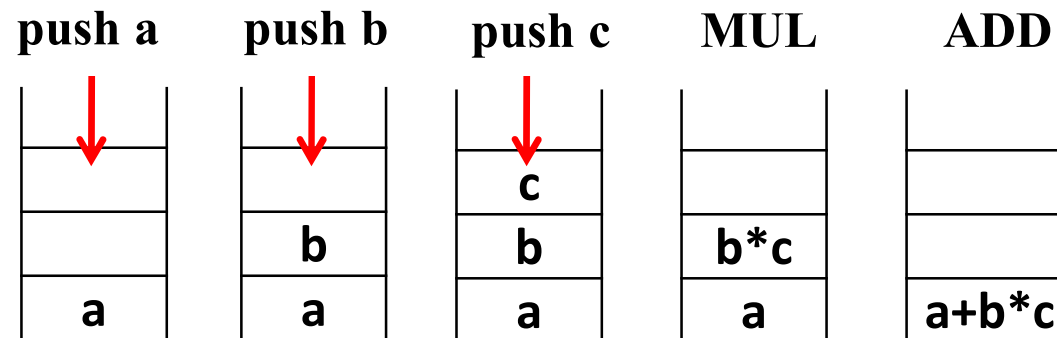


# Types de machines des années 60

- ❑ Machines avec seulement des langages de haut niveau
  - Stockent leurs opérandes dans une pile (stack machine)
  - Exemples : Burrough's 5000 et B6700 (mémoire virtuelle et plusieurs processeurs et mémoires)
- ❑ Machines à registres généraux (GPR machine)
  - Utilisent un jeu d'instructions commun
  - Exemple : IBM 360
- ❑ Machines pipeline (Load/Store machine)
  - Superordinateurs avec une horloge rapide
  - Exemple : CDC6600

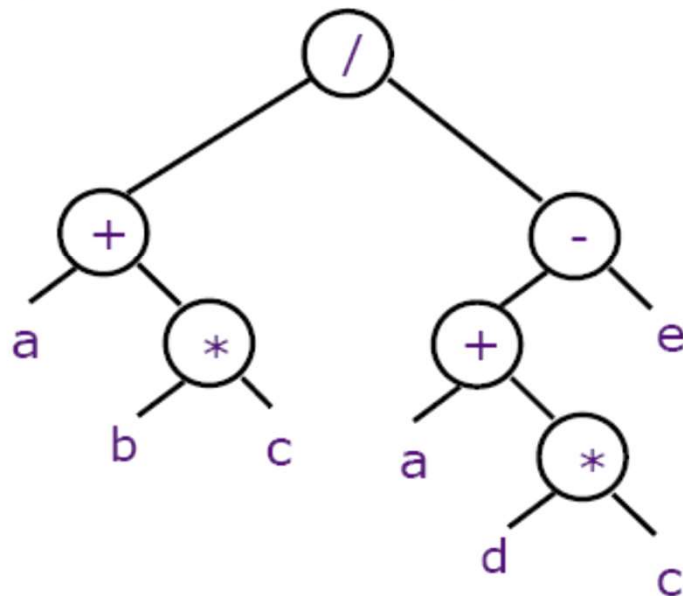
# La machine à pile

- ❑ Les machines à pile sont adaptées à :
  - Evaluation des expressions (arithmétiques, logiques,...)
  - Appel de procédures, récursivité, interruptions imbriquées...
  - Accès aux variables dans les langages structurés par bloc
- ❑ Utilise des instructions spécifiques : empiler (push), dépiler (pop)
- ❑ Exemple 1 : calcul de  $a+b*c$



# Evaluation d'une expression (1/2)

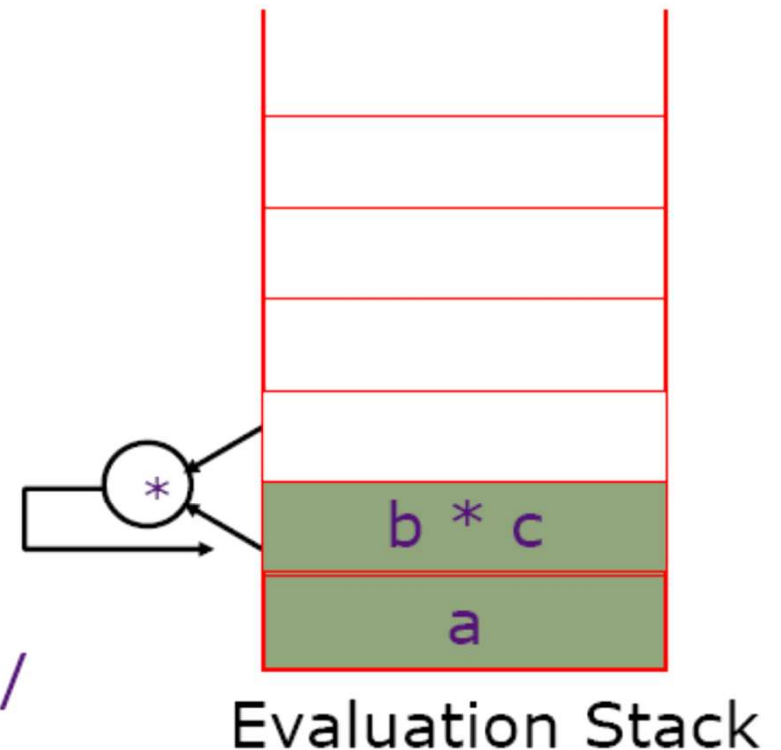
$$(a + b * c) / (a + d * c - e)$$



Reverse Polish

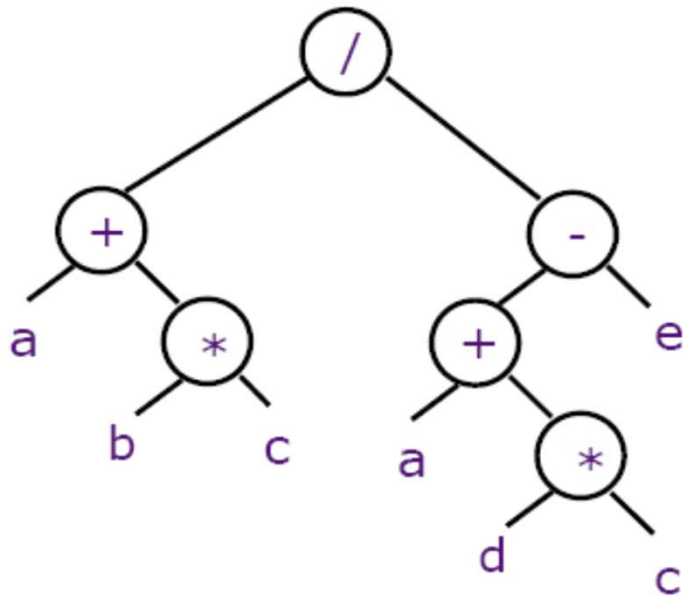
a b c \* + a d c \* + e - /

↑    ↑    ↑    ↑  
push a Push c  
Push b    multiply



# Evaluation d'une expression (2/2)

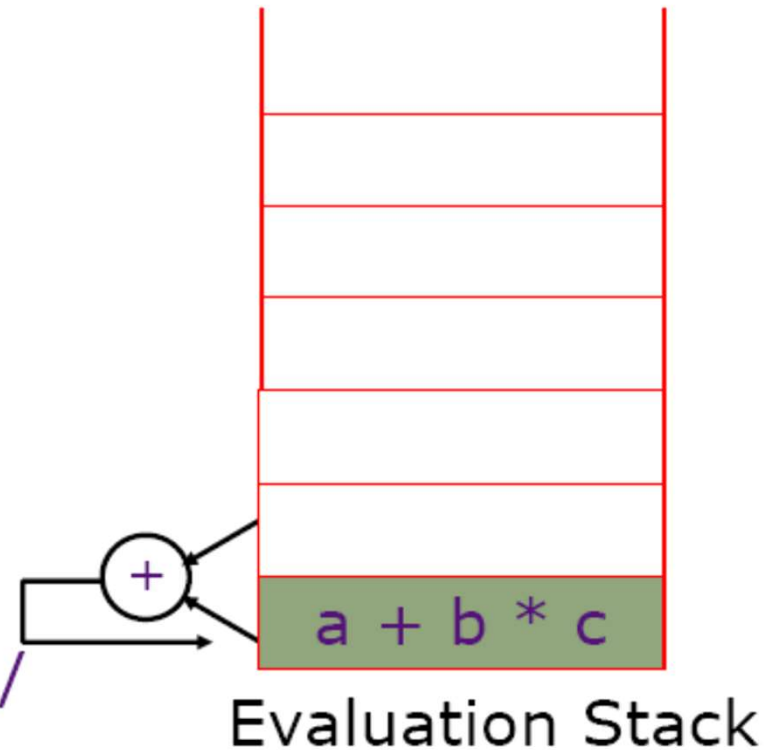
$$(a + b * c) / (a + d * c - e)$$



Reverse Polish

a b c \* + a d c \* + e - /

↑  
add



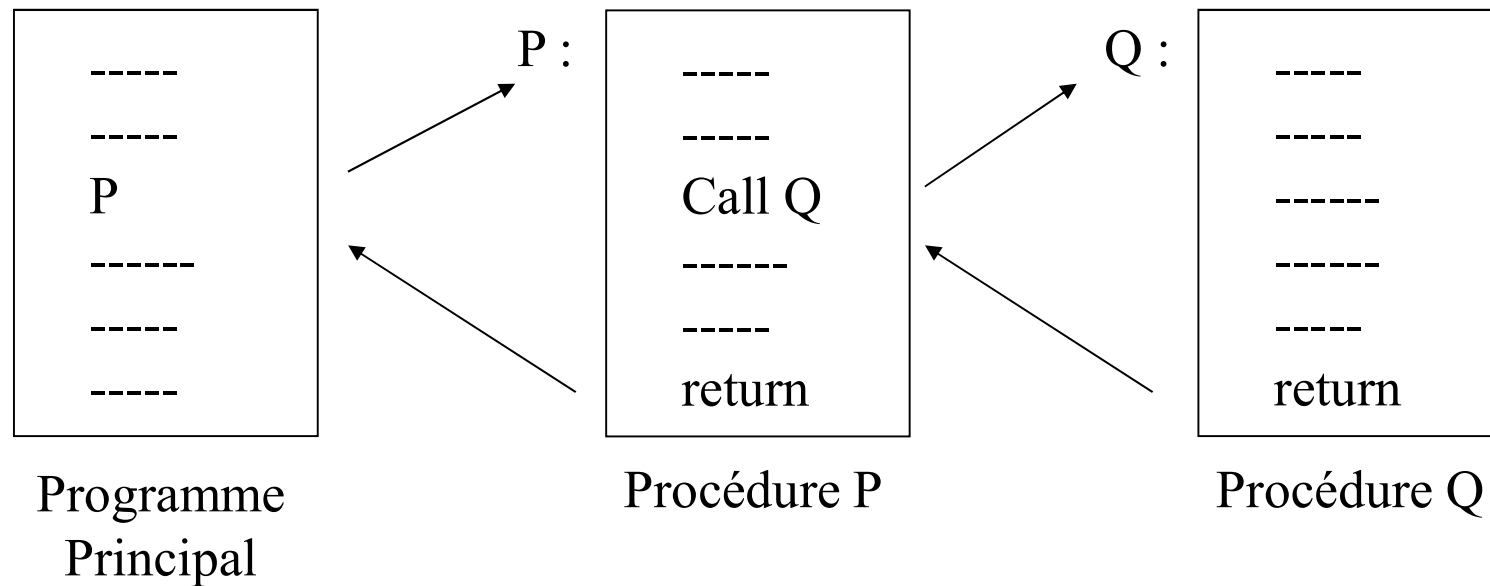


# Organisation matérielle de la pile

- ❑ La pile fait partie de l'état du processeur
  - La pile doit être limitée et de petite taille (quelques registres)
- ❑ Conceptuellement la pile est illimitée
  - Une partie de la pile est incluse dans l'état du processeur et le reste est gardé en mémoire.

# Appels de procédures

- La sauvegarde des appels de procédures utilise la pile



- Problème : si  $Q = P$  (récursivité) → taille de pile



# Burroughs 5000

- ❑ Créée en 1961
- ❑ Machine à pile utilisant des transistors
- ❑ Conçue pour être programmée en algol 60 (ancêtre de C et Java)
- ❑ 1 bit de flag pour distinguer code et données
- ❑ Mémoire magnétique (core)
- ❑ Mémoire virtuelle



Burroughs 5000/5500

# La machine à registres généraux

## GPR machine

- ❑ Utilise un petit nombre de registres qui portent des noms
- ❑ Utilise différentes opérations de chargement de registres
  - LOAD Ri m, LOAD Ri (Rj)...
- ❑ Evite des références inutiles à la mémoire (Réutilisation de registres)
- ❑ Machine favorite depuis 1980 :
  - Registres de courtes adresses
  - Compilateurs qui gèrent bien l'espace des registres

Exemple :

abc\*+ac-/

**LOAD R0 a**

**LOAD R1 b**

**LOAD R2 c**

**MUL R1 R2**

**ADD R1 R0**

**SUB R0 R2**

**DIV R1 R0**

# IBM 360

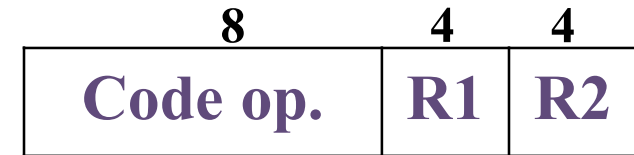
- ❑ Lancée par IBM en 1964
- ❑ A base de circuits intégrés
- ❑ Machine 32 bits à registres
  - Adresses de 24 bits (index+base)
- ❑ Différents registres
  - 16 registres généraux de 32 bits : Index, base, registre 0
  - 4 registres flottants de 64 bits
  - 1 registre d'état (PSW) : flag, PC
- ❑ Données sur 1 octet, Demi mot, mot, Double mot
- ❑ Séries d'ordinateurs de même jeu d'instructions mais de tailles et puissances variées



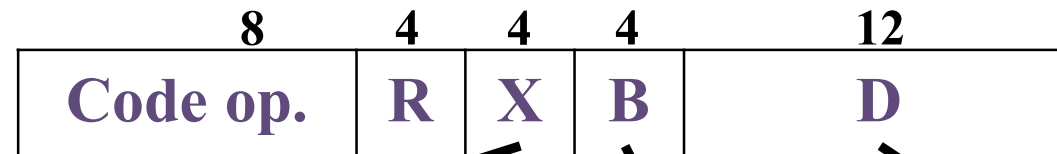
IBM 360

# Formats d'instructions de IBM 360

□ RR :  $R1 \leftarrow (R1) \text{ op } (R2)$



□ RD :  $R \leftarrow (R) \text{ op } M[(X) + (B) + D]$



← **Registre d'index**
← **Registre de base**
← **Déplacement**

□ SS :  $M[(B1)+D1] \leftarrow M[(B1)+D1] \text{ op } M[(B2)+D2]$



– Utilisé dans le cas des chaînes de caractères et des décimaux

# La machine pipeline

- ❑ Problème d'inactivité des circuits :
  - Les étapes d'exécution d'une instruction sont exécutées par des circuits différents
  - Lorsqu'une étape est en cours, les autres circuits sont inactifs
- ❑ Solution : Lorsqu'une instruction passe à l'étape  $i$ , l'instruction suivante passe à l'étape  $i-1$

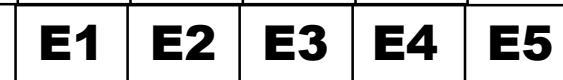
Instruction1



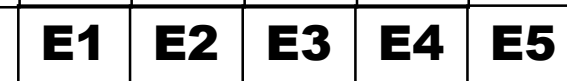
Instruction2



Instruction3



Instruction4

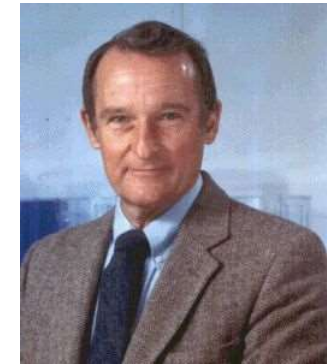


Instruction5



# CDC 6600

- ❑ Créée par Seymour Cray (1965)
- ❑ pipeline (10 instructions à la fois)
- ❑ Mémoire de 256000 mots de 60 bits
- ❑ 10 unités : addition, multiplication et division
- ❑ 10 processeurs d'E/S
- ❑ Horloge à 10MHz (3 millions d'opérations/s)
- ❑ 3 types de registres :
  - 8 registres de données de 60 bits (X)
  - 8 registres d'adresses de 18 bits (A)
  - 8 registres d'index de 18 bits (B)



S. Cray



CDC 6600

# Architecture Load/store de CDC6600

## □ Instructions de calcul register to register

–  $R_i \leftarrow (R_j) \text{ op } (R_k)$

6	3	3	3
<b>Opcode</b>	<b>i</b>	<b>j</b>	<b>k</b>

## □ Instructions Load/store utilisant la mémoire

–  $R_i \leftarrow M[(R_j) + \text{offset}]$

– Registres d'adresses 1 à 5 (6 et 7) pour load (store)

6	3	3	18
<b>Opcode</b>	<b>i</b>	<b>j</b>	<b>offset</b>

# Evolution depuis 1965 (1/5)

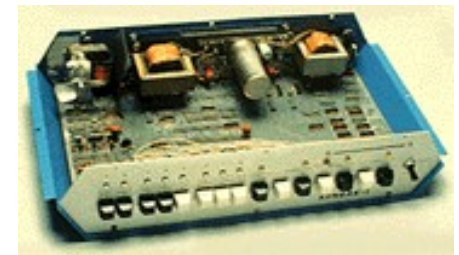
## ❑ Mini-ordinateur DEC-PDP 8

- Créé en 1960 et introduit en 1965
- Mémoire de 4096 mots de 12 bits
- 2 registres : Accumulateur+registre de lien
- 8 instructions



## ❑ Machines à circuits intégrés SSI et MSI 1965–1971      PDP 8

- 1<sup>er</sup> micro-ordinateur Kenback 1 (1971)
  - mémoire de 256 o, 3 registres
- 1<sup>er</sup> microprocesseur Intel 4004 (1971)
  - Processeur 4 bits 108 KHz,
  - 60000 instructions/s, 2300 transistors



Kenback 1



# Evolution depuis 1965 (2/5)

## ❑ Machines à circuits intégrés LSI 1972-1977 : Microordinateurs

- Micral (1973) : 8080 d'Intel
- Altair 8800 (MITS, 1975) : personnel
  - Intel 8080 à 2 MHz, 256 o de mémoire
- Apple I (S. Wozniak & S. Jobs, 1976) :
  - muni de clavier
  - 6502 à 1 MHz, 4 Ko de RAM,
  - 1 Ko de RAM vidéo.
- PET (Personal Electronic Transactor, 1977)
  - Z80 de Zilog



Micral



Altair



Apple I



PET

# Evolution depuis 1965 (3/5)

- VLSI (Very Large Scale Integration) depuis 1978
  - Intel 8086-8088 (1978) : 330000 instructions/s
  - Pentium III (1999) : 10 millions de transistors...
- Micro-informatique (performance et coût améliorés)
  - 1<sup>er</sup> PC d'IBM (1981)
    - 8088 à 5 MHz, 64 Ko de RAM
  - Osborne I (1981) : 1<sup>er</sup> ordinateur portable
  - Macintosh 128 (1984)
    - MC68000 de Motorola à 8 MHz, 128 Ko de
    - Interface graphique et souris
- ❑ Ordinateurs de poche
  - PIC (Sony) et PDA (Apple), SE «micro-noyau»



1<sup>er</sup> PC d'IBM



MAC 128



# Evolution depuis 1965 (4/5)

- ❑ Systèmes opératoires
  - RD-DOS, MS-DOS 1981, Windows 1.0 1985, Linux 1992
- ❑ Langages de programmation et logiciels
  - C (D. Ritchie, 1972), réécriture d'unix en C
  - PROLOG (1972)
  - C++, CLIPS (1990), Java (1995)...
  - Microsoft MS Word 1985...
- ❑ Réseaux
  - ARPAnet (1969) présenté au public en 1972
  - Internet (1981) : 213 machines connectées

# Evolution depuis 1965 (5/5)

- ❑ Architecture RISC (IBM 801) en 1975

<b>RISC</b> <b>Reduced Instruction Set</b> <b>Computer</b>	<b>CISC</b> <b>Complex Instruction Set</b> <b>Computer</b>
Rapides	Lentes
Nombre réduit d'instructions (10 à 30)	Nombre important d'instructions (75 à 150)
Traitements simples	Traitements complexes
Instruction s'exécutant en une période d'horloge	Instruction s'exécutant en plusieurs périodes d'horloges



## Chapitre 5

# MICROPROGRAMMATION

1. Chemin de données et unité de contrôle
2. Cycle d'exécution d'instruction et microprogramme
3. Contrôle câblé
4. Microcontrôle (microprogrammation)
5. Architecture de MIPS
6. Instructions MIPS
7. Formats d'instructions MIPS
8. Compilations des programmes MIPS

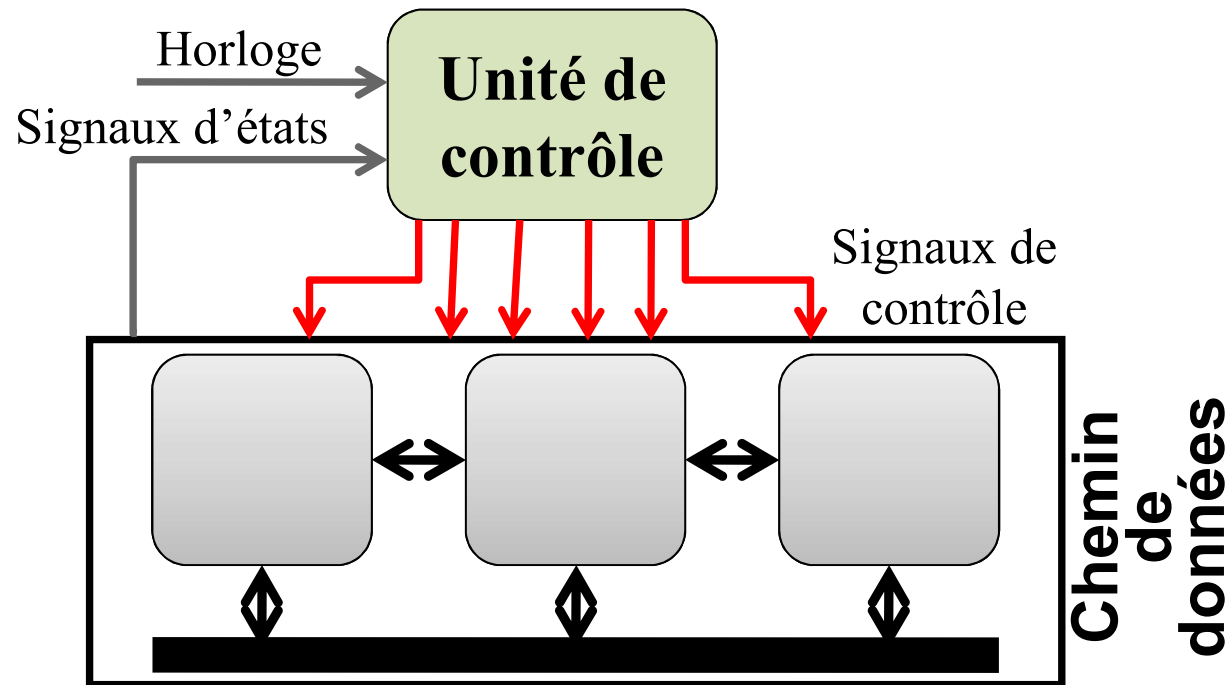


# Chemin de données

- ❑ Défini de manière statique par :
  - L'ensemble d'unités utilisées pour réaliser une instruction :
    - UAL, Registres, Mémoires... et Bus (transformation, stockage et transfert)
  - L'interconnexion des ces unités
- ❑ Définit le chemin emprunté par les données lors de l'exécution des diverses instructions
- ❑ Pb : Le parcours du chemin est dynamique (relatif à l'instruction exécutée)
  - ➔ Nécessité d'une unité de contrôle

# Unité de contrôle

- ❑ Coordonne le fonctionnement des différentes unités :
  - Selon le calcul courant et le prochain calcul à faire, elle envoie des signaux de contrôle à chaque top d'horloge
    - Exemple : Ouverture de ports entre 2 registres





# Cycle d'exécution d'une instruction

## □ 5 étapes :

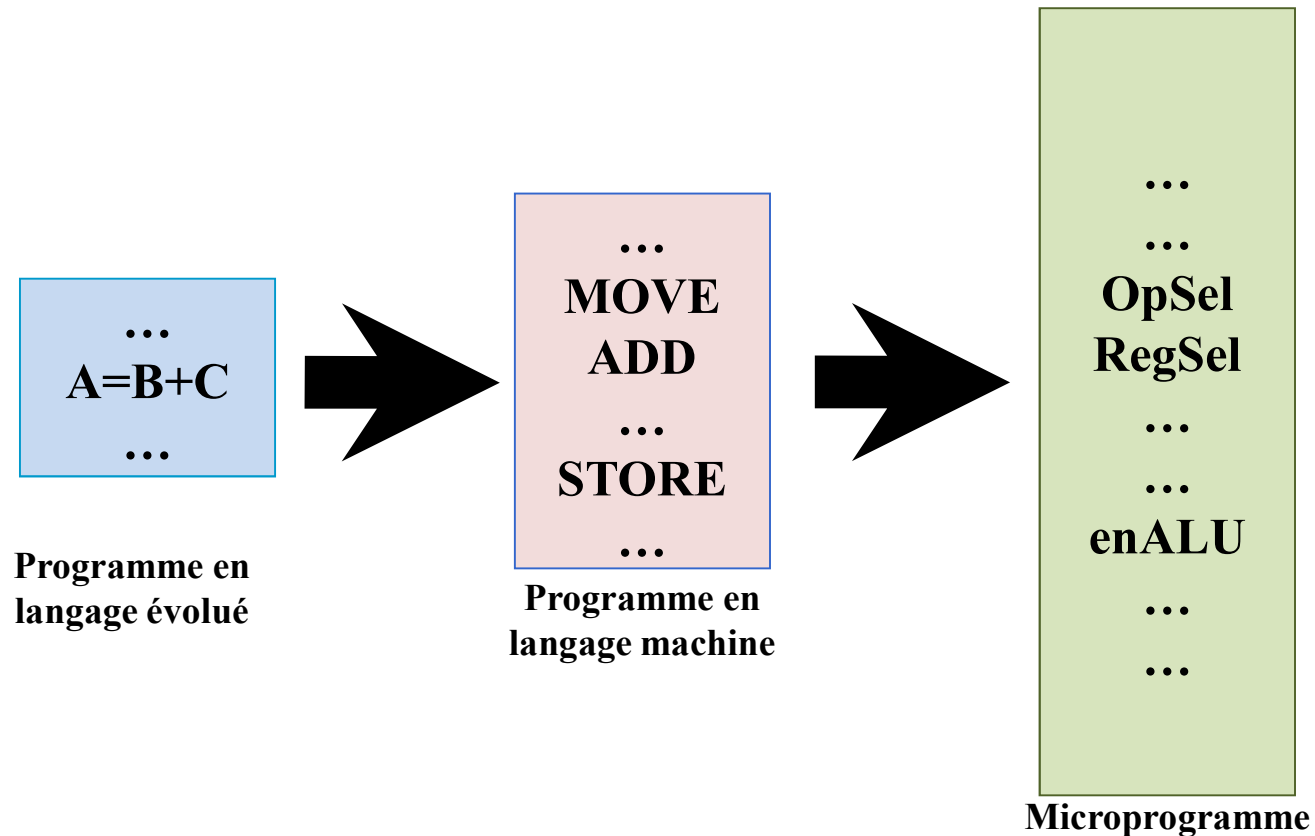
- Recherche de l'instruction
  - Placer le code opératoire dans le registre d'instruction (RI)
- Décodage
  - Traduire le code en une séquence de signaux de contrôle envoyées aux unités concernées
- Exécution
  - Effectuer le calcul ou la lecture/écriture...
- Recherche des opérandes en mémoire (optionnel)
- Ecriture des résultats et mise à jour des registres (optionnel)



# Microopérations

- ❑ Chaque étape est réalisée par des microopérations
- ❑ Exemple 1 : Recherche d'instruction
  - $RA \leftarrow (PC)$                        $RA$  : registre d'adresse
  - $RD \leftarrow \text{Mémoire}[RA]$        $RD$  : registre de données
  - $RI \leftarrow (RD)$
- ❑ Exemple 2 : exécution de  $\text{ADD } R1, [R2]$ 
  - $RA \leftarrow R2$
  - $RD \leftarrow \text{Mémoire}[RA]$
  - $R1 \leftarrow R1 + RD$

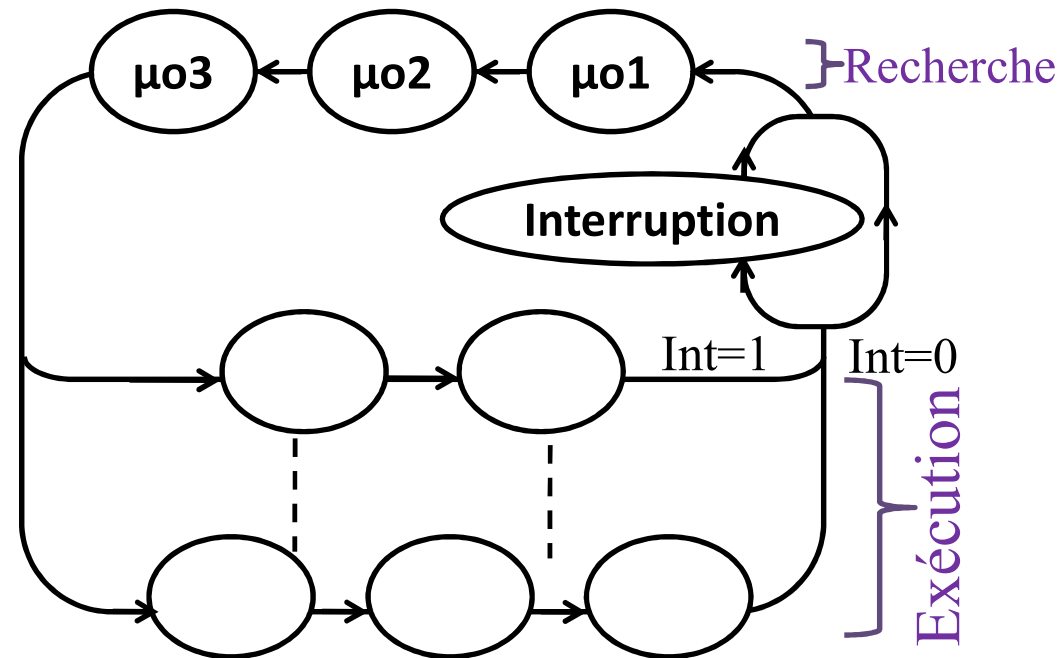
# Taille du programme



- ❑ L'unité de contrôle doit séquencer les différentes microopérations pour exécuter une instruction

# Implémentation du séquenceur

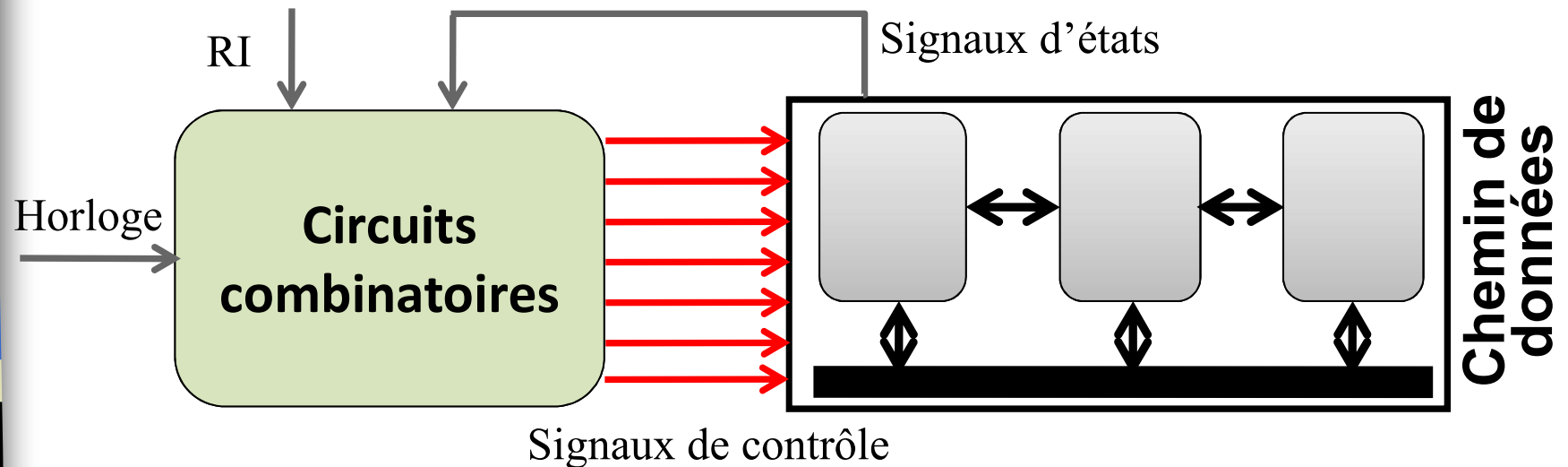
- ❑ Le séquenceur est une machine à états finis où les états représentent des microopérations



- ❑ 2 types d'implémentation du séquenceur :
  - Câblée
  - Microprogrammée

# Séquenceur câblé

## □ Circuits logiques combinatoires séquentiels



- Les signaux résultent des fonctions logiques appliquées aux bits du code opératoire
- Avantage : circuit simple
- Inconvénient : incapable d'implémenter une instruction complexe (nombre fini de fonctions logiques)



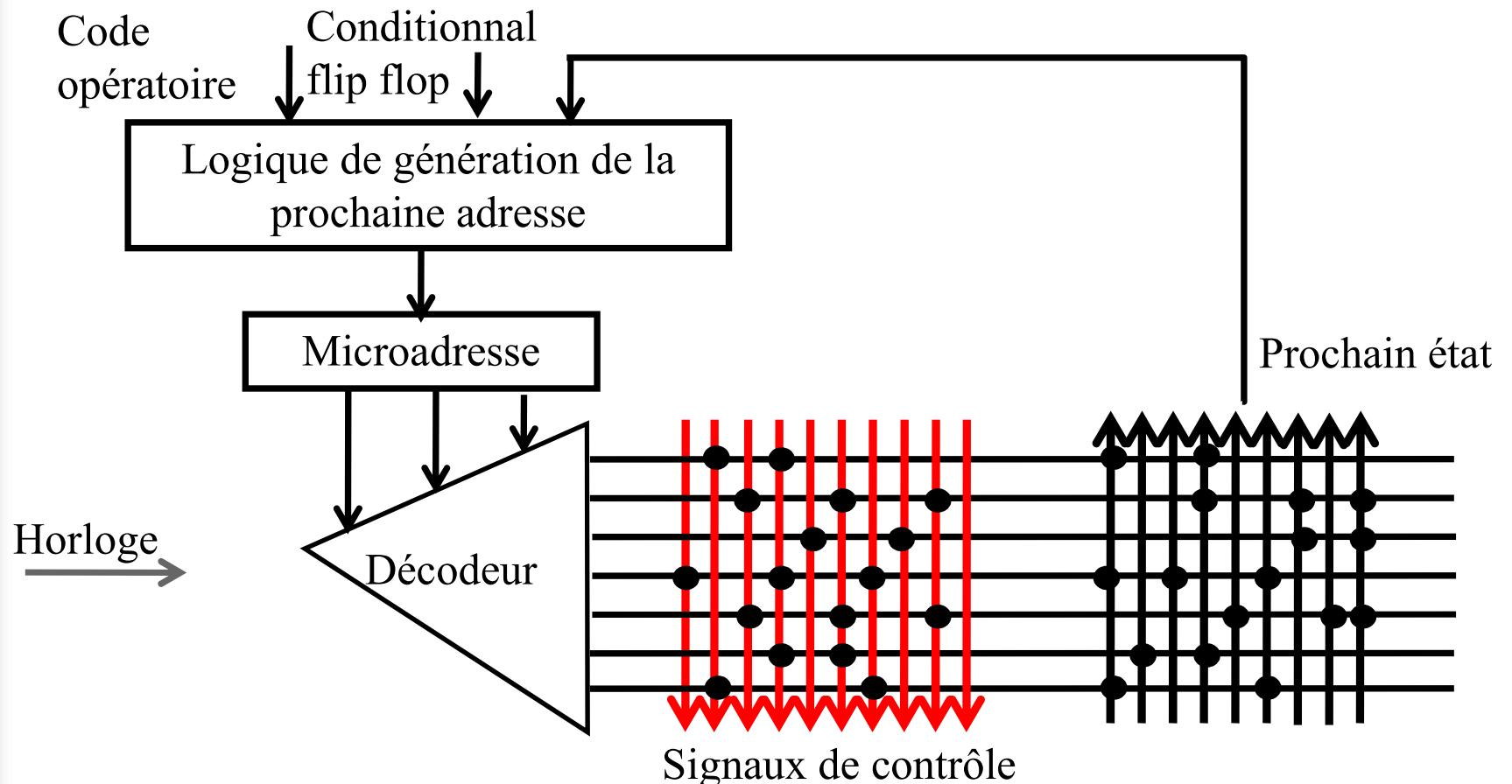
# Microprogrammation

- ❑ Inventée par Maurice Wilkes en Angleterre au début des années 1950
- ❑ Principe :
  - Remplacer un séquenceur câblé par un séquenceur programmé pour réaliser des instructions complexes
  - Utiliser une mémoire de contrôle qui contient un microprogramme (constitué de micro-instructions)
- ❑ Avantage : Jeu d'instructions modifiable (remplacer la mémoire ou modifier son contenu)
- ❑ Inconvénients : séquencement compliqué (+ de place) et ralenti

# Microprogrammation horizontale

## Modèle de Wilkes

- ❑ Un décodeur décode la microadresse et sensibilise le fil horizontal de la microinstruction concernée
- ❑ Chaque bit correspond à un signal



# Taille de la mémoire de contrôle

- ❑ Considérons :
  - CRL : bits de contrôle,      ETAT : bits d'état,
  - OP : bits du code opératoire, COND : bits de conditions
- ❑ Taille d'un mot =  $(CRL + ETAT)$  bits
- ❑ Nombre de mots =  $2^{OP + COND + ETAT}$
- ❑ Taille de la mémoire =  $2^{OP + COND + ETAT} * (CRL + ETAT)$  bits
- ❑ Problème : Une mémoire d'accès rapide est coûteuse  
→ Nécessité de réduire sa taille



# Amélioration de la taille de la mémoire de contrôle

- ❑ Réduction des nombres de mots (hauteur)
  - Réduire le nombre de bits (logique externe) : 1 bit double la taille
  - Utiliser des groupes opératoires pour des actions ayant des microinstructions communes → Etats réduits
  - Condenser les bits conditionnels en 1 bit (vrai ou non)
- ❑ Réduction de la taille d'un mot (largeur)
  - Réduire aux états suivants possibles ( $\mu$ adresse suivante, saut à une  $\mu$ adresse...)
  - Grouper les signaux de contrôle (Microprogrammation verticale)





# Microprogrammation verticale

- ❑ Regrouper les signaux de contrôle mutuellement exclusifs
- ❑ Utiliser un décodeur pour convertir les codes en signaux
- ❑ Avantage :
  - Mémoire de contrôle plus compacte en largeur (moins de bits)
- ❑ Inconvénients :
  - Un peu plus de logique
  - Lenteur de traitement

# Intel 8086

- ❑ Microprocesseur 16 bits
- ❑ Types de données
  - octet, mot et double mot
- ❑ Mémoire adressable par octet
- ❑ Registres
  - AX, BX, CX, DX
  - SI, DI
  - DS, CS, SS, ES
  - ...



# Jeu d'instructions 8086

- ☐ Instructions de transfert
- ☐ Instructions de calcul
- ☐ Instructions de saut/appel
- ☐ Instructions de décalage

# Instruction de transfert

## □ **mov dest , source**

– dest reçoit source

– Exemples :

–           mov al,0   #   al  $\leftarrow$  0

–           mov n,al   #   n  $\leftarrow$  (al)

# Instructions de calcul

## ❑ add dest , source

- Dest reçoit  $\text{dest} + \text{source}$
- Exemple : add al, n      #     $\text{al} \leftarrow (n)$

## ❑ sub dest , source

- Dest reçoit  $\text{dest} - \text{source}$

## ❑ mul dest , source      (\*)

## ❑ div dest , source      (/)

## ❑ inc source

- source reçoit  $\text{source} + 1$

## ❑ dec source

- source reçoit  $\text{source} - 1$

## ❑ and dest , source      (masquage)

- Comme add mais  $1 + 1 = 1$

# Instructions de saut/appel

## ❑ **jmp address**

- Effectue un saut à adress (étiquette en général)
- Exemple : `jmp L   #   Aller à L`

## ❑ **Jz address** (Branch if Zero)

- Effectue un saut à adress si dernier résultat = 0
- Autres instructions similaires : **jnz**

## ❑ **Ja address** (Branch if Above)

- Effectue un saut à adress si dernier résultat > 0
- Autres instructions similaires : **jb, je, jbe, jae**

## ❑ **call address** (étiquette)

- Effectue un appel de procédure débutant à adress et se terminant par **Ret** (branchement à l'adresse de retour)



# Instructions de décalage

## ❑ **shl ax, cx**

- Décalage à gauche de ax de cx bits
- Exemple : shl ax, cx  $\#ax \leftarrow ax * 2^{cx}$

## ❑ **shr ax, cx**

- Décalage à droite de ax de cx bits

# Exercice

□ Copier 87654321h dans AX

- add AX, 8765H
- mov CX, 16d
- shl AX, CX
- Add AX, 4321H



# Compilation de programmes en assembleur 8086 (1/2)

□  $a = b + c;$

- `mov al, b`             $\# al \leftarrow (b)$
- `add al, c`             $\# al \leftarrow (al) + (c)$
- `mov a, al`             $\# a \leftarrow (al)$

□  $d = a - e;$

- `mov al, a`             $\# al \leftarrow (a)$
- `sub al, e`             $\# al \leftarrow (al) - (e)$
- `mov d, al`             $\# d \leftarrow (al)$

# Compilation de programmes en assembleur 8086 (2/2)

□  $f = (g + h) - (i + j);$

- `mov al, i`             $\# al \leftarrow (i)$
- `add al, j`             $\# al \leftarrow (al) + (j)$
- `mov bl, al`            $\# bl \leftarrow (al)$
- `mov al, g`             $\# al \leftarrow (g)$
- `add al, h`             $\# al \leftarrow (al) + (h)$
- `sub al, bl`            $\# al \leftarrow (al) - (bl)$
- `mov f, al`             $\# f \leftarrow (al)$

# Exercices (1/3)

□ Soient les données  $g$ ,  $h$ ,  $i$  et le tableau  $A$

a) Opérande en mémoire

- $g = h + A[3];$
- $A$  : un tableau d'octets

b) Chargement et sauvegarde dans la mémoire

- b.1)  $A[5] = h + A[3];$
- b.2)  $g = h + A[i];$
- $A$  : un tableau de mots

# Exercices (2/3)

□ Soient les données  $f$ ,  $g$ ,  $h$ ,  $i$  et  $j$

c) Instruction de sélection

– if ( $i == j$ ) go to L1;

$f = g + h$ ;

L1:  $f = f - i$ ;

d) Sélection binaire

– if ( $i == j$ )

$f = g + h$  ;

else  $f = g - h$ ;

# Exercices (3/3)

□ Soient les données  $g, h, i, j$  et les tableaux  $A$  et  $B$

e) Boucle

- Loop :  $g = g + A[i];$   
 $i = i + j;$   
if ( $i \neq h$ ) goto Loop;

f) Boucle « tant que »

- While ( $B[i] == h$ )  
 $i = i + j;$

# Solutions d'exercices (1/6)

a)  $g = h + A[3];$

– mov al, h	# al	← (h)
– mov SI, 3	# SI	← 3
– mov bl, A[SI]	# bl	← (A[3])
– add al, bl	# al	← (al) + (bl)
– mov g, al	# g	← (al)

b.1)  $A[5] = h + A[3];$

– mov ax, h	# ax	← (h)
– mov SI, 3	# SI	← 3
– mov bx, A[SI]	# bx	← (A[3])
– add ax, bx	# ax	← (ax) + (bx)
– mov DI, 5	# DI	← 5
– mov A[DI], ax	# A[5]	← (ax)

# Solutions d'exercices (2/6)

b.2)  $g = h + A[i];$

– mov al, h	# al	← (h)
– mov SI, i	# SI	← (i)
– mov bl, A[SI]	# bl	← (A[i])
– add al, bl	# al	← (h) + (A[i])
– mov g, al	# g	← (al)

# Solutions d'exercices (3/6)

c) if (i == j) go to L1;

    f = g + h;

L1: f = f - i;

- mov al,i       # al ← (i)
- cmp al,j       #comparer al et j
- jz L1          # si (al) = (bl) aller à L1
- mov al,g       # sinon al ← (g)
- add al,h       # al ← (al) + (h)
- mov f,al       # f ← (al)
- ...            # Quitter
- L1: mov al,f       # al ← (f)
- sub al,i       # al ← (al) - (i)
- mov f,al       # f ← (al)



# Solutions d'exercices (4/6)

d) if (i == j)

    f = g + h ;

else f = g - h;

- mov al,i     # al ← (i)
- cmp al,j     #comparer al et j
- jnz L1       # si (al) ≠ (bl) aller à L1
- mov al,g     # sinon al ← (g)
- add al,h     # al ← (al) + (h)
- mov f,al     # f ← (al)
- ...          # Quitter
- L1:  mov al,g     # al ← (g)
- sub al,h     # al ← (al) - (h)
- mov f,al     # f ← (al)

# Solutions d'exercices (5/6)

e) Loop : g=g+A[i]; i=i+j; if (i!=h) goto Loop;

- Loop : mov al, g           # al   ← (g)
- mov SI,i           # SI   ← (i)
- mov bl, A[SI]   # bl   ← (A[i])
- add al,bl       # al   ← (al) + (bl)
- mov g ,al       # g   ← (al)
- mov al, i       # al   ← (i)
- add al,j       # al   ← (al) + (j)
- mov i,al       # i   ← (al)
- cmp al,h       #comparer al et h
- jnz Loop       # si (al) ≠ (h) aller à Loop

# Solutions d'exercices (6/6)

f) While ( $B[i] == h$ )  $i = i + j$ ;

- Loop : `mov al, h`           # al   ← (h)
- `mov SI, i`           # SI   ← (i)
- `mov bl, B[SI]`       # bl   ← (B[i])
- `cmp al, bl`           # comparer al et bl
- `jz L1`               # si (al) = (bl) aller à L1
- ...                   # Quitter
- L1 : `mov al, i`           # al   ← (i)
- `add al, j`           # al   ← (al) + (j)
- `mov i, al`           # i     ← (al)
- `jmp Loop`           # aller à Loop



# Structure d'un programme

## □ Dosseg

- Met les segments du programme dans l'ordre selon la convention d'organisation Microsoft.

## □ .Model

- Permet de choisir un modèle mémoire : tiny, small, medium, compact, large, hogue
- small : CS et DS utilisent un segment de 64ko chacun.

## □ .stack

- Détermine la taille de la pile.

## □ .data

- Indique le début du segment de données (déclarations).

## □ .code

- Indique le segment de code



# Syntaxe d'une ligne de déclaration

□ **<étiquette> <type de donnée> <initialisation>  
<;commentaire>**

- Etiquette : identificateur.
- Type de données :db, dw, dd
- Initialisation : valeur repérée par rapport à sa base (d, b, h; par défaut, d) ou ?.
- Commentaire : commence par ; et se termine à la fin de la ligne.

# Exemples de déclarations

- ❑ x db 12h
- ❑ y db 12d
- ❑ z db 12
- ❑ message db 'bonjour\$'
- ❑ M db 13,10,'bonjour',13,10,[\$]
- ❑ Tableau db 7 dup (0) ; Le tableau peut ne pas être ; initialisé. Il suffit de mettre ; ? Au lieu de 0.
- ❑ Liste db 1,2,3



# Syntaxe d'une ligne de code

□ **<étiquette:> <instruction> <opérandes>  
<;commentaire>**

- Etiquette : référence à un emplacement mémoire (une partie du code comme une procédure).
- Instruction : appartient au jeu d'instructions prédéfini.
- Opérandes : doivent être séparés par une virgule.
- Commentaire : commence par ; et se termine à la fin de la ligne.

□ **Remarques :**

- Le fichier source doit avoir l'extension .asm.
- Le programme doit se terminer par end. Le code qui suit end sera ignoré.



# Exemple d'un premier programme (1/2)

- Dosseg
- `.model small` ; choix du modèle small
- `.data`
- `n db 2` ; nombre initialisé à 2
- `m db ?` ; donnée 8 bits non initialisée.
- `. code`
- `mov ax,@data` ; ces deux instructions servent à initialiser
- `mov ds,ax` ; l'adresse du segment de données.
- `mov al,n` ; met le contenu de n dans al
- `call addition` ; appelle la procédure addition
- `jmp fin` ; se déplace à l'emplacement libellé fin





# Exemple d'un premier programme (2/2)

- addition:; procédure addition
- mov bl,n       ; met le contenu de n dans bl.
- add al,bl       ; calcule la somme de al et bl.
- mov m,al       ; met le résultat dans m.
- ret            ; retour à la prochaine instruction  
                        ; après l'appel de procédure.
- fin:                ; procédure qui termine l'exécution.
- mov ah,4ch     ; met la valeur 4ch dans ah.
- int 21h        ; appelle l'interrupteur Dos avec 21h
- end                 ; fin du code.



# Installation de TASM, assemblage et exécution

## ❑ Installation

- Simple copie de TASM.
- Basculer en mode DOS :

*Tous les programmes/Accessoires/Invites de commandes*

- Changer la variable PATH : *set path=C:\TASM*

## ❑ Assemblage et exécution :

- *tasm nom\_fichier.asm*
- *tlink nom\_fichier.obj*
- Taper le nom de l'exécutable.

# HIÉRARCHIE DE LA MÉMOIRE

1. Définition, terminologie et types de mémoires
2. Mémoire principale
3. Conception de mémoires
4. Mémoire multi-modules et entrelacement
5. Mémoire associative
6. Mémoire cache
7. Mémoire virtuelle



# Définition et terminologie

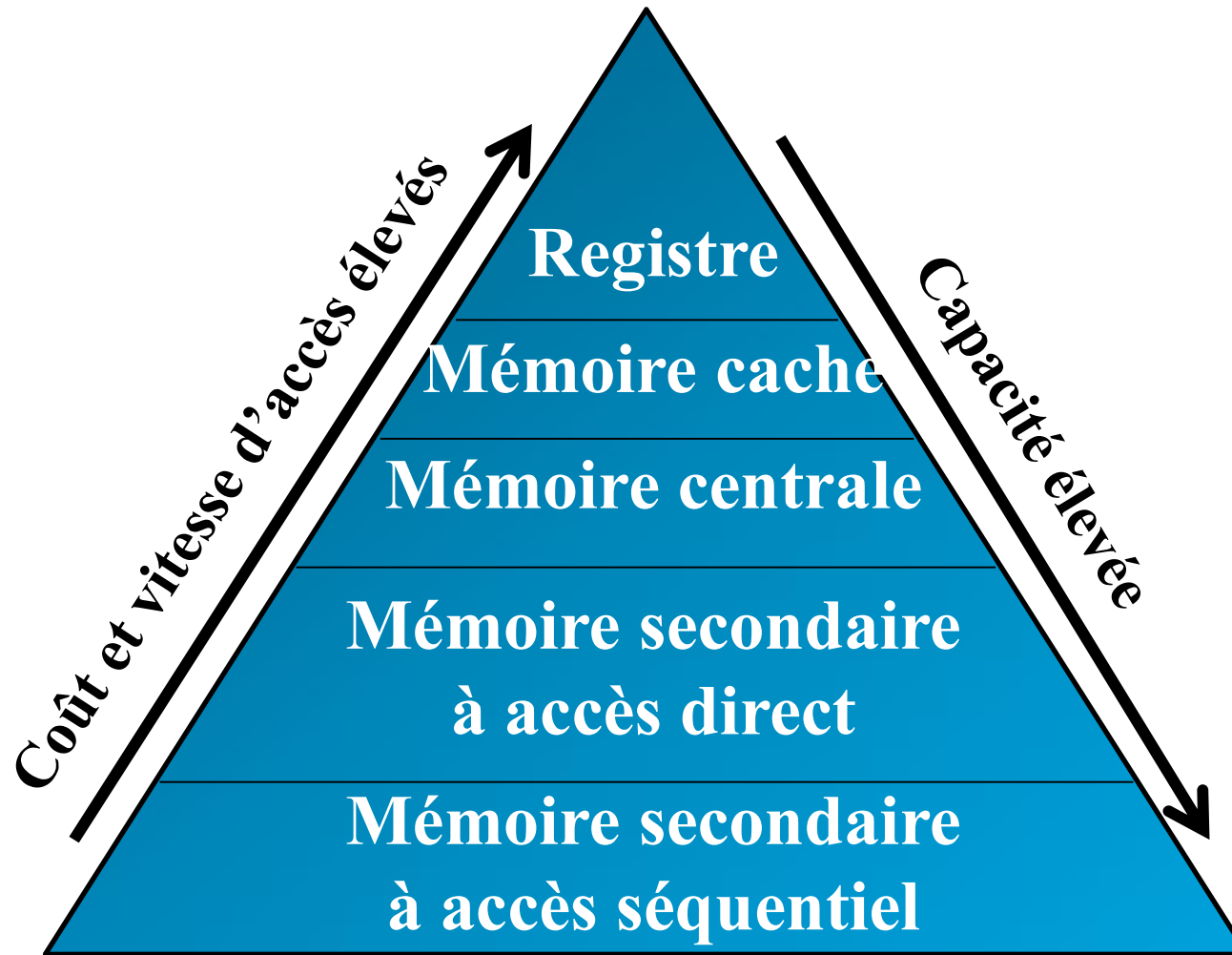
- ❑ Dispositif pour stocker et restituer une information sous forme binaire
- ❑ Capacité : nombre d'octets ou de mots ou de bits (registres)
  - Bit, Octet (8 bits), KO ( $1024 = 2^{10}$  octets), MO, GO, To
- ❑ Temps d'accès (temps de latence) : lecture ou écriture
  - Instant où les données sont disponibles - instant où l'adresse est fournie
- ❑ Temps de cycle (si accès aléatoire) : temps d'accès + temps nécessaire avant un 2<sup>ème</sup> accès
- ❑ Volatilité : conservation ou non des données en cas de coupure d'alimentation



# Types de mémoires

- ❑ Selon la possibilité de lecture/écriture
  - Mémoires vives : RAM à lecture/écriture, Volatiles
    - SRAM (+rapide) et DRAM (-coûteuse)
  - Mémoires mortes : ROM à lecture seule, non volatiles
    - ROM, PROM à fusibles, EPROM, EEPROM, Flash
- ❑ Selon la technologie utilisée
  - Mémoire à semi-conducteur (RAM, ROM, PROM...)
  - Mémoires magnétiques (disque dur, disquettes...)
  - Mémoires optiques (CD, DVD...)
- ❑ Selon l'emplacement
  - Mémoires intégrées au processeur (Registres)
  - Mémoires internes (Mémoire principale)
  - Mémoires externes (Mémoire secondaire ).

# Hiérarchie de la mémoire



# Mémoires en chiffre

	Temps d'accès	Capacité
<b>Registre</b>	0,3 ns	64 bits
<b>Cache</b>	2 à 5 ns	8 Ko à 1 Mo
<b>RAM</b>	50 ns	1 Go
<b>Cache disque</b>	1 ms	8 Mo
<b>Disque dur</b>	10 ms	160 Go



# Mémoire principale (MP)

- ❑ Contient les informations utilisées par le processeur lors de l'exécution
- ❑ Sa capacité et son temps d'accès ont un impact sur la performance de la machine
- ❑ Est une mémoire à semi-conducteurs pour un accès rapide
- ❑ Constituée de mots possédant chacun une adresse unique
  - Taille des adresses dépend de la capacité de la mémoire



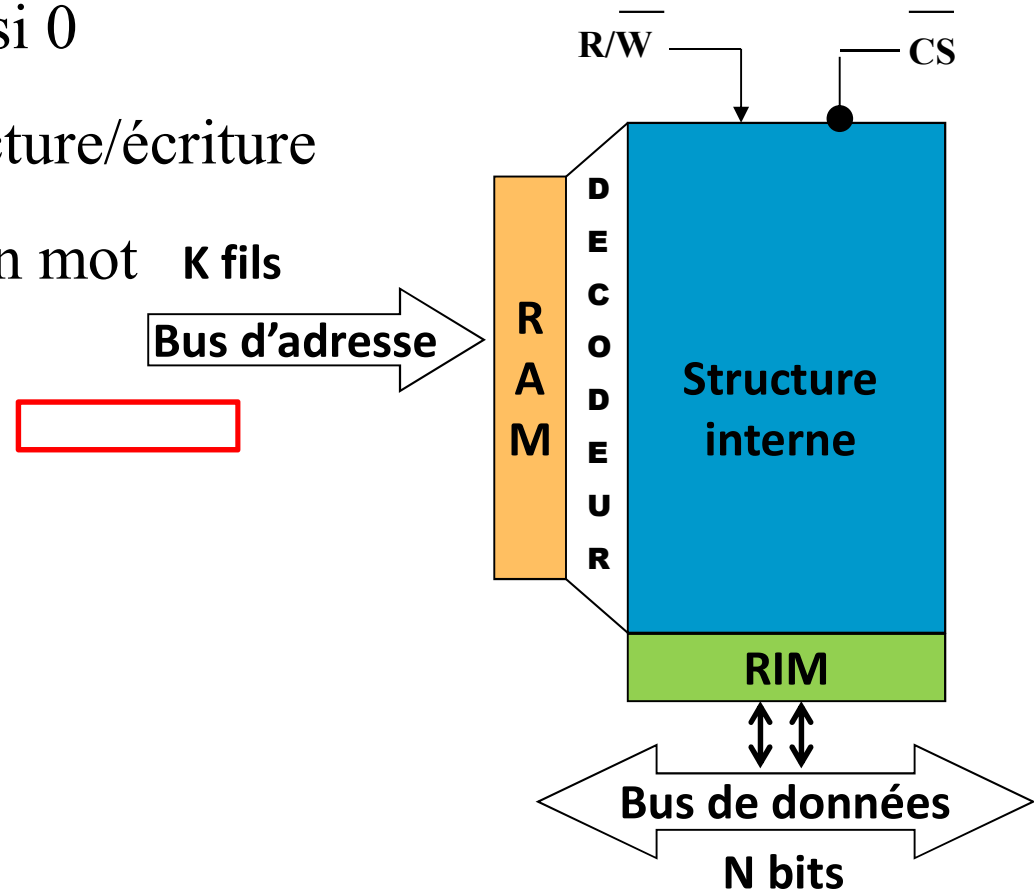


# Caractéristiques de la MP

- ❑ Mémoire vive
- ❑ Accès aléatoire (RAM)
- ❑ A lecture-écriture
- ❑ Volatile
- ❑ Capacité limitée (possibilité d'extension)
- ❑ Communique au moyen des bus d'adresses et de données
- ❑ Types
  - Mémoires statiques (SRAM) : à base de bascules D
  - Mémoires dynamiques (DRAM) : à base de condensateurs

# Structure physique d'une MP

- RAM : Registre d'adresse Mémoire
- $\overline{CS}$  : boîtier sélectionné si 0
- $R/\overline{W}$  : Commande de lecture/écriture
- Décodeur : sélectionne un mot K fils



- Capacité =  $2^k$  Mots =  $2^k * n$  Bits



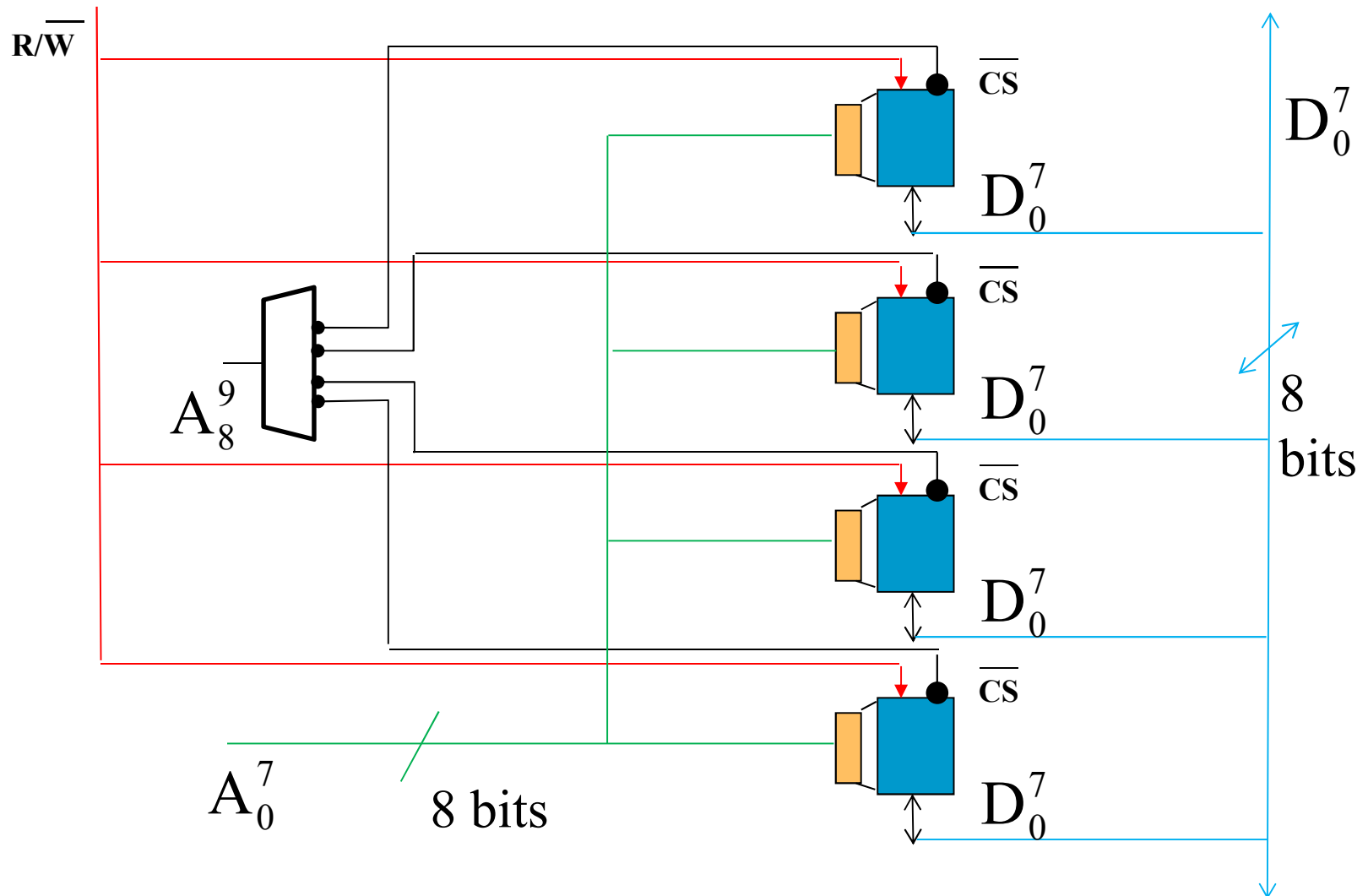
# Conception des MP

- ❑ Pb : Comment réaliser une mémoire à partir de boîtiers de petite taille?
  - Mémoire M de capacité C et de m mots de n bits
  - Boîtier M' de capacité C' et de m' mots de n' bits
  - $C > C'$  ( $m \geq m'$ ,  $n \geq n'$ )
- ❑ Nombre de boîtiers nécessaires : P.Q
  - $P = m/m'$  (facteur d'extension lignes)
  - $Q = n/n'$  (facteur d'extension colonnes)
- ❑ K bits de poids forts d'adresses pour sélectionner Q boîtiers ( $2^k = P$ ), le reste pour sélectionner un mot

# Exemple 1

- ❑ Réaliser une mémoire de 1Ko (un mot est de 8 bits) en utilisant des boîtiers de taille 256 mots de 8 bits
- ❑ Solution :
  - $m=1024 \rightarrow$  bus d'adresses de 10 bits ( $A_0^9$ )
  - $n=8 \rightarrow$  bus de données de 8 bits ( $D_0^7$ )
  - $m'=256 \rightarrow$  bus d'adresses de 8 bits ( $A_0^7$ )
  - $n'=8 \rightarrow$  bus de données de 8 bits ( $D_0^7$ )
  - $P = m/m' = 1024/256 = 4$
  - $Q = n/n' = 8/8 = 1$
  - Nombre total de boîtiers :  $P.Q=4$

# Solution 1



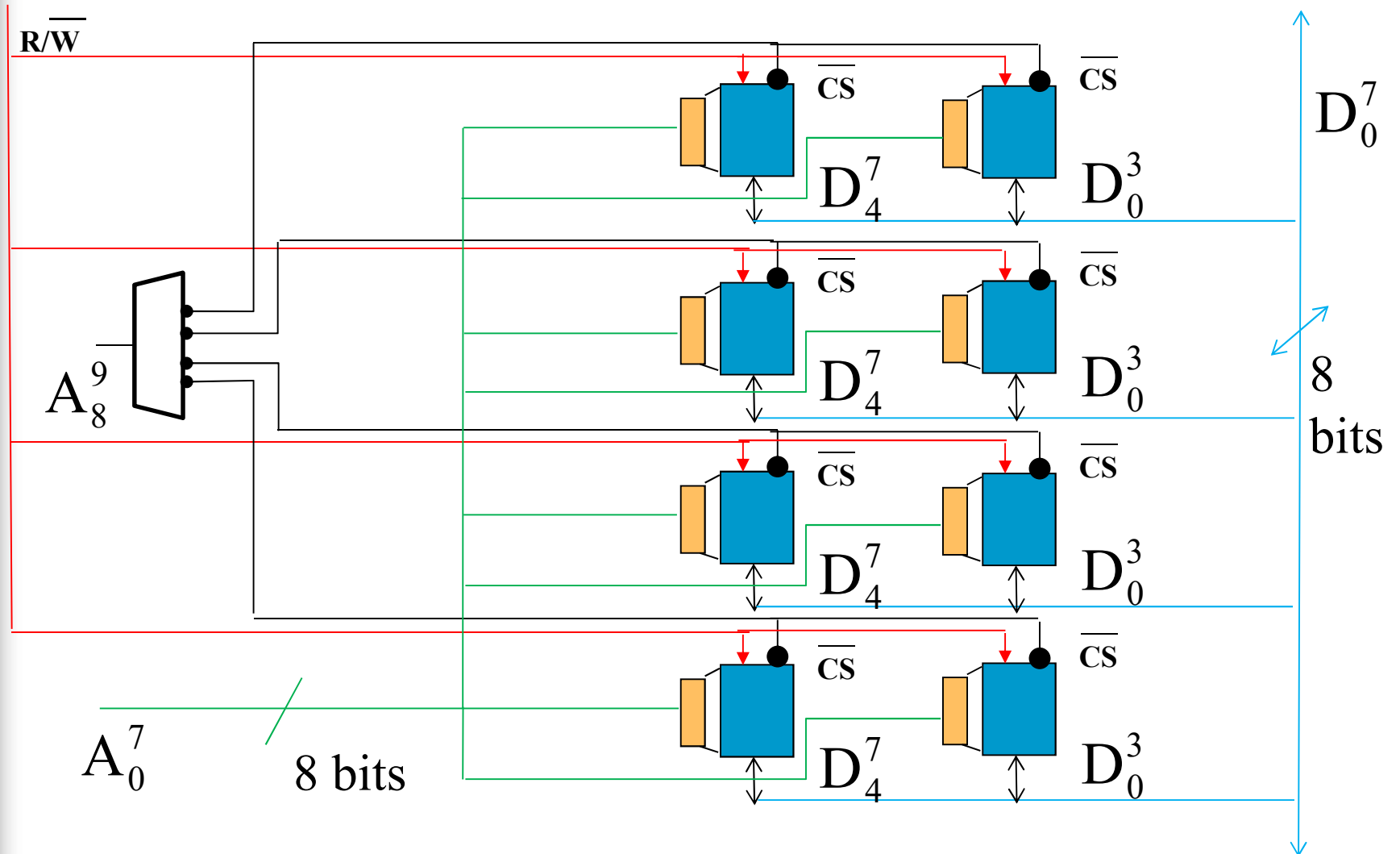
## Exemple 2

- ❑ Réaliser une mémoire de 1KO ( et de mots de 8 bits) en utilisant des boîtiers de taille 256 mots de 4 bits

Solution :

- $m=1024 \rightarrow$  bus d'adresses de 10 bits (  $A_0^9$  )
- $n=8 \rightarrow$  bus de données de 8 bits (  $D_0^7$  )
- $m'=256 \rightarrow$  bus d'adresses de 8 bits (  $A_0'^7$  )
- $n'=4 \rightarrow$  bus de données de 4 bits (  $D_0'^3$  )
- $P = m/m' = 1024/256 = 4$
- $Q = n/n' = 8/4 = 2$
- Nombre total de boîtiers :  $P.Q=8$

# Solution 2





# Mémoire multi-modules

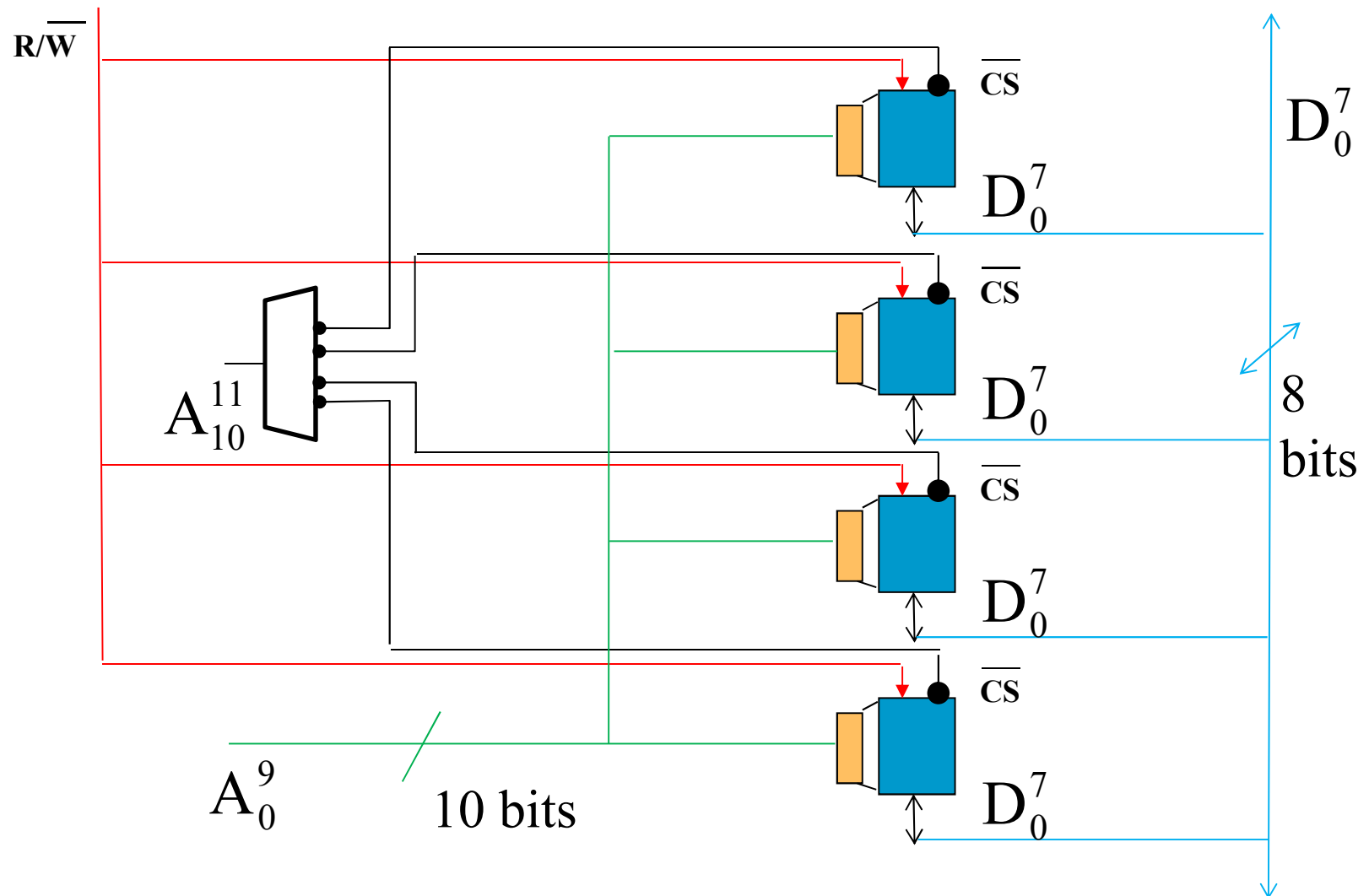
- ❑ Pb : mémoire accessible par 1 seul processeur à la fois
- ❑ Solution : découper la mémoire en plusieurs modules
- ❑ Possibilité d'accès simultané aux différents modules par plusieurs bus
- ❑ Modules comprenant des mots d'adresses séquentielles



# Adressage d'une mémoire modulaire

- Adresse divisée en 2 parties :
  - K Bits de poids forts pour sélectionner un module tel que :  $2^k \geq \text{nombre de modules}$
  - Bits de poids faibles pour sélectionner un mot dans un module
  
- Exemple : mémoire de 4 Ko et 4 modules et des boîtiers de 1 Ko
  - Capacité = 4 Ko =  $4 * 2^{10} = 2^{12}$  o  $\rightarrow$  bus d'adresses de 12 bits
  - 2 bits du poids forts pour la sélection des modules ( $A_{10}^{11}$ )
  - ( $A_0^9$ ) pour la sélection d'un mot

# Exemple de mémoire à 4 modules





# Mémoire entrelacée

- ❑ Pb : Module mémoire accessible par 1 seul processeur à la fois (ex : accès à la fois aux données consécutives)
- ❑ Solution :
  - Diviser la mémoire en plusieurs blocs dotés de leurs propres registres d'adresses → plusieurs accès simultanés à la mémoire
  - Placer les données consécutives dans des blocs différents
  - Le nombre de blocs représente le degré d'entrelacement
- ❑ Adresse divisée en deux parties :
  - K bits de poids faibles pour sélectionner le bloc ( $2^k \geq \text{nombre de blocs}$ )
  - Bits de poids forts pour sélectionner le mot dans le bloc

# Exemple 1

- ❑ Mémoire entrelacée avec un degré d'entrelacement égale à 4, un bloc est de taille de 4 mots de 4 bits
- ❑ Solution :
  - 4 blocs et taille d'un bloc égale à 4 mots de 4 bits → taille de la mémoire = 16 mots de 4 bits
  - 4 blocs → 2 bits de poids faibles pour la sélection  $A_0^1$
  - Les bits de poids forts ( $A_2^3$ ) pour sélectionner un mot dans un bloc



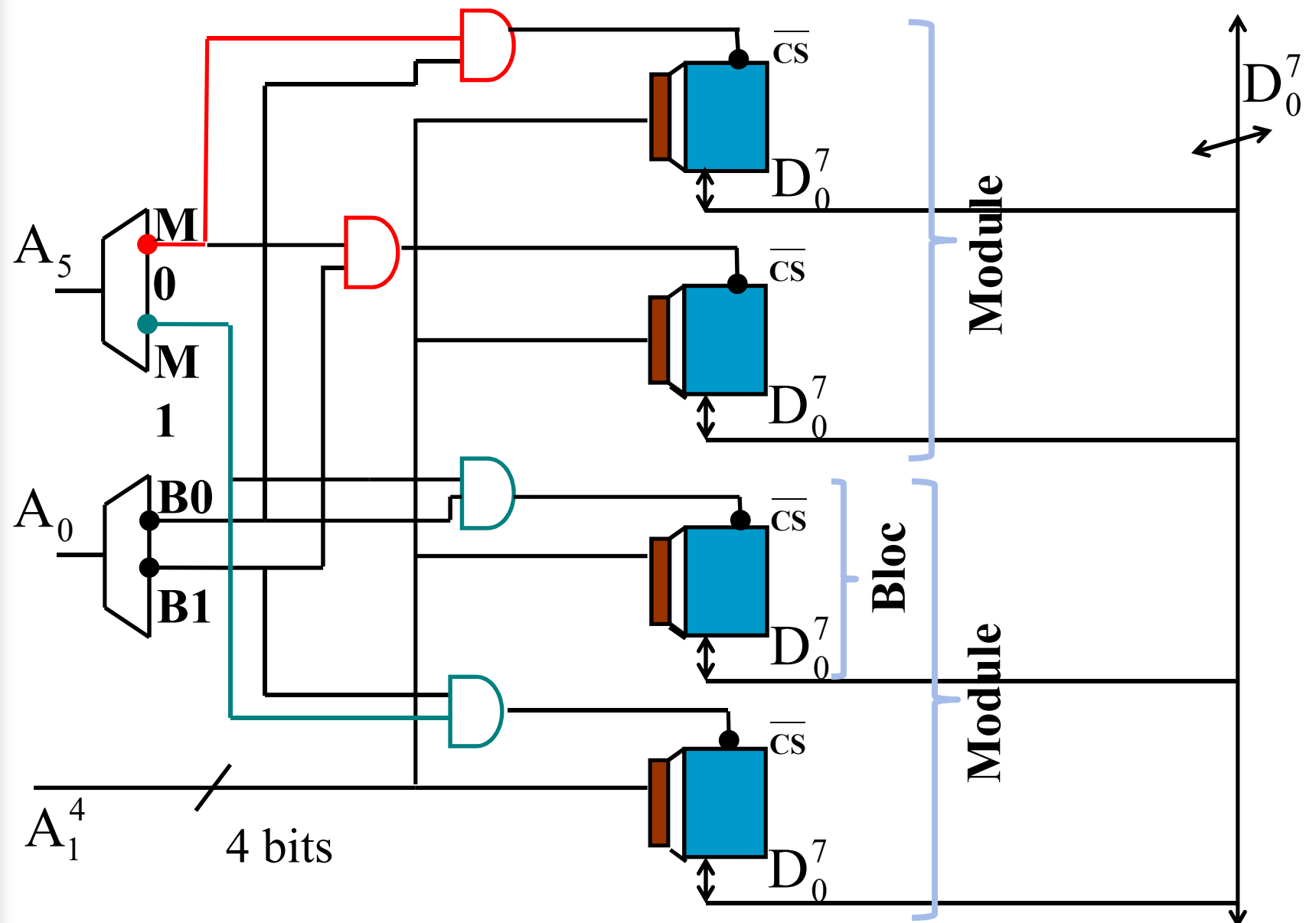
# Mémoire modulaire entrelacée

- ❑ MP divisée en plusieurs modules
- ❑ Chaque module est divisé en  $n$  blocs
- ❑ Sélection de mots
  - Bits de poids forts pour sélectionner le module
  - Bits de poids faibles pour sélectionner le bloc dans le module
  - Bits restants pour sélectionner le mot dans le bloc

# Exemple

- ❑ Mémoire de 64 mots de 8 bits organisée en 2 modules entrelacés (degré d'entrelacement  $D=2$ ). On utilise des boîtiers de 16 mots de 8 bits
  - Taille du bus d'adresses  $k=6$  ( $64=2^6$ )  $\rightarrow A^5_0$
  - Nombre de modules  $m=2$ , Taille d'un module=32 mots
  - Nombre de bits pour sélectionner un module = 1 ( $A^5$ )
  - Nombre de blocs dans un module  $D=2 \rightarrow$  Nombre de bits nécessaire pour sélectionner un bloc = 1 ( $A_0$ )
  - Taille d'un bloc = 16 mots  $\rightarrow$  un boîtier suffit pour réaliser un bloc
  - Nombre de bits nécessaire pour sélectionner un mot dans le bloc = 4 ( $A^4_1$ )

# Solution



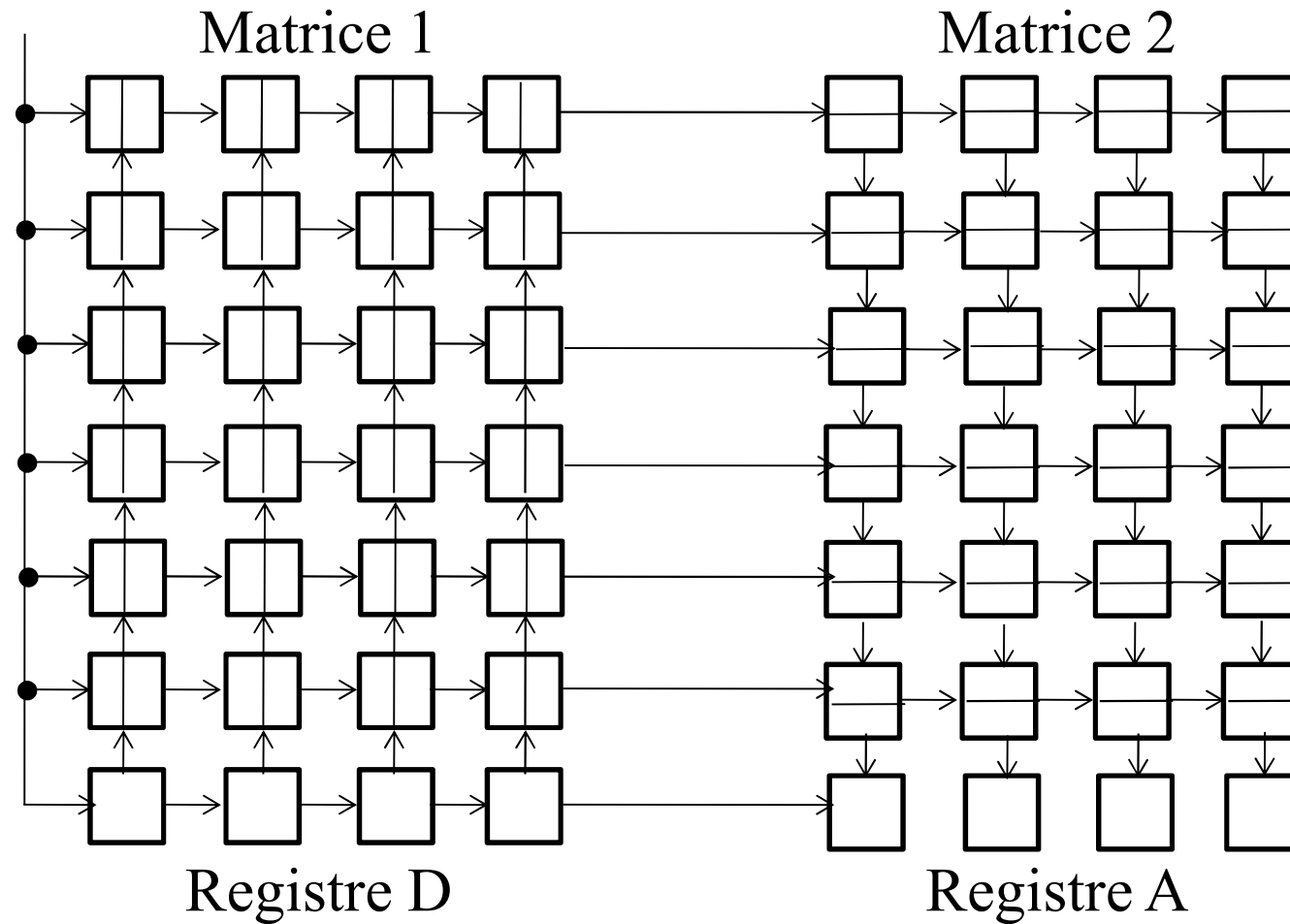


# Mémoire associative

- ❑ Mémoire adressable par le contenu pour une recherche plus rapide
  - Mémoire à accès aléatoire : information à partir d'une adresse
- ❑ Mémoire associative : fournir un descripteur (clé) et obtenir l'information associée s'il existe
- ❑ Divisée en 2 parties M1 et M2
  - M1 : mots comparés en parallèle au descripteur
  - M2 : fournit l'information associée dans un registre A



# Mémoire associative en logique cellulaire





# Mémoire cache

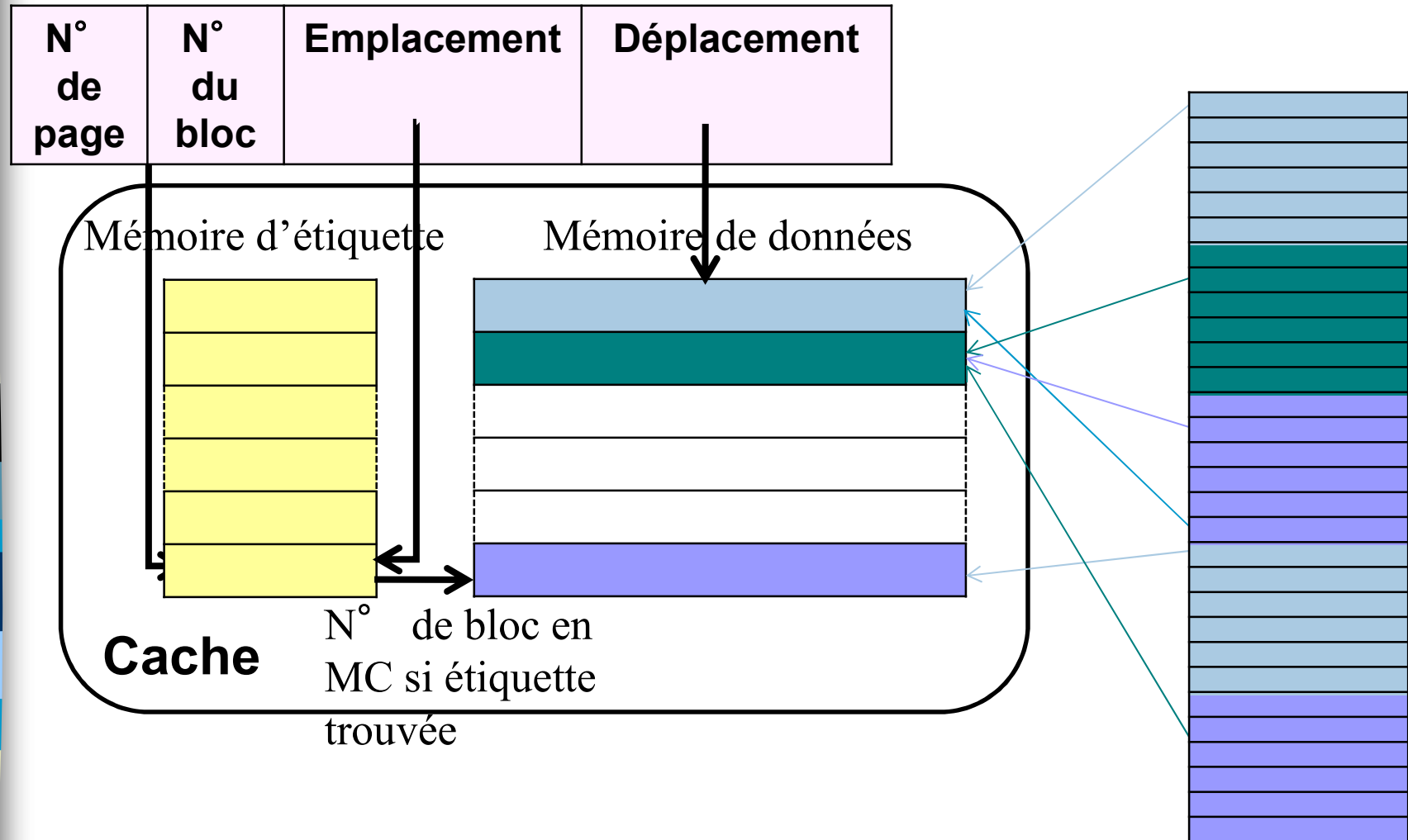
- ❑ Antémémoire : + rapide que MP, taille + petite
- ❑ Mots les + fréquemment utilisés
- ❑ Chercher d'abord les données dans le cache, si défaut de cache, les copier de la MP
- ❑ Accès rapide, Efficacité dépend de la taille et la politique de remplissage (remplacement)



# Mémoire cache à correspondance directe

- ❑ Chaque ligne du cache est constituée de :
  - Etiquette : N° de page et du bloc copié dans le cache
  - Données : données des blocs copiés
- ❑ Considérons :
  - P : nombre de blocs de la MP   - J : N° du bloc en MP
  - Q : nombre de blocs de la MC   - I : N° du bloc en MC
- ❑ A un bloc de la MC sont mappés n blocs de la MP
  - $n = P/Q$
  - $I = J \text{ modulo } Q$
- ❑ L'emplacement en MC du bloc demandé est connu
- ❑ Inconvénient : risque de défaut de cache (remplacer souvent les mêmes blocs de la MC)

# Exemple de MC à correspondance directe

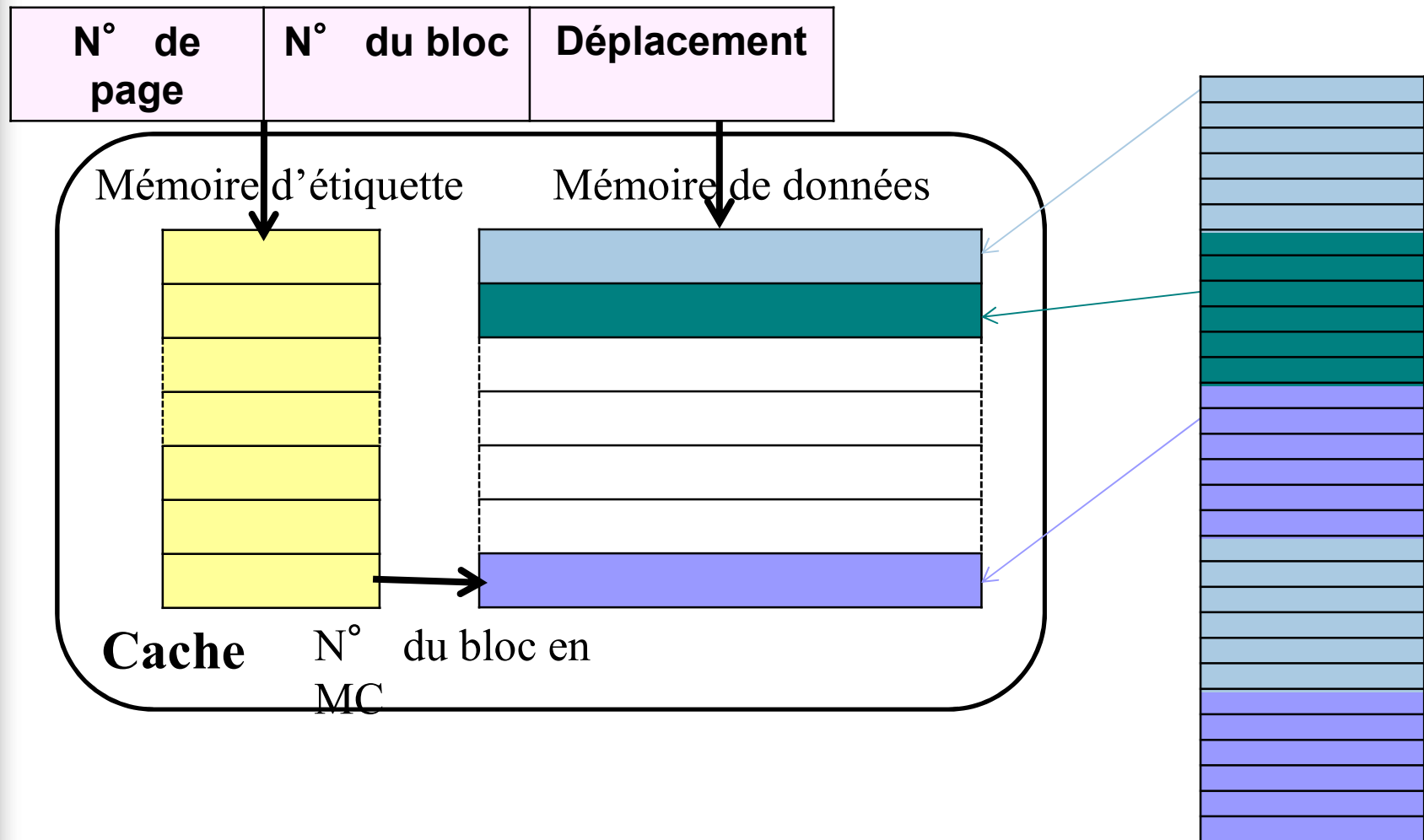




# Mémoire cache complètement associative

- ❑ Mémoire (SRAM) découpée en blocs de même taille que celle des blocs de la MP
- ❑ Constituée de :
  - Mémoire d'étiquettes : N° de page et de blocs copiés dans le cache et leurs adresses dans la MC
  - Mémoire de données : données des blocs copiés
- ❑ Tout bloc de la MP est mappé indifféremment dans l'un des blocs de la MC
- ❑ N° de bloc comparé à tous les N° de blocs en MC
- ❑ Cache efficace mais complexe et volumineux

# Exemple de MC complètement associative

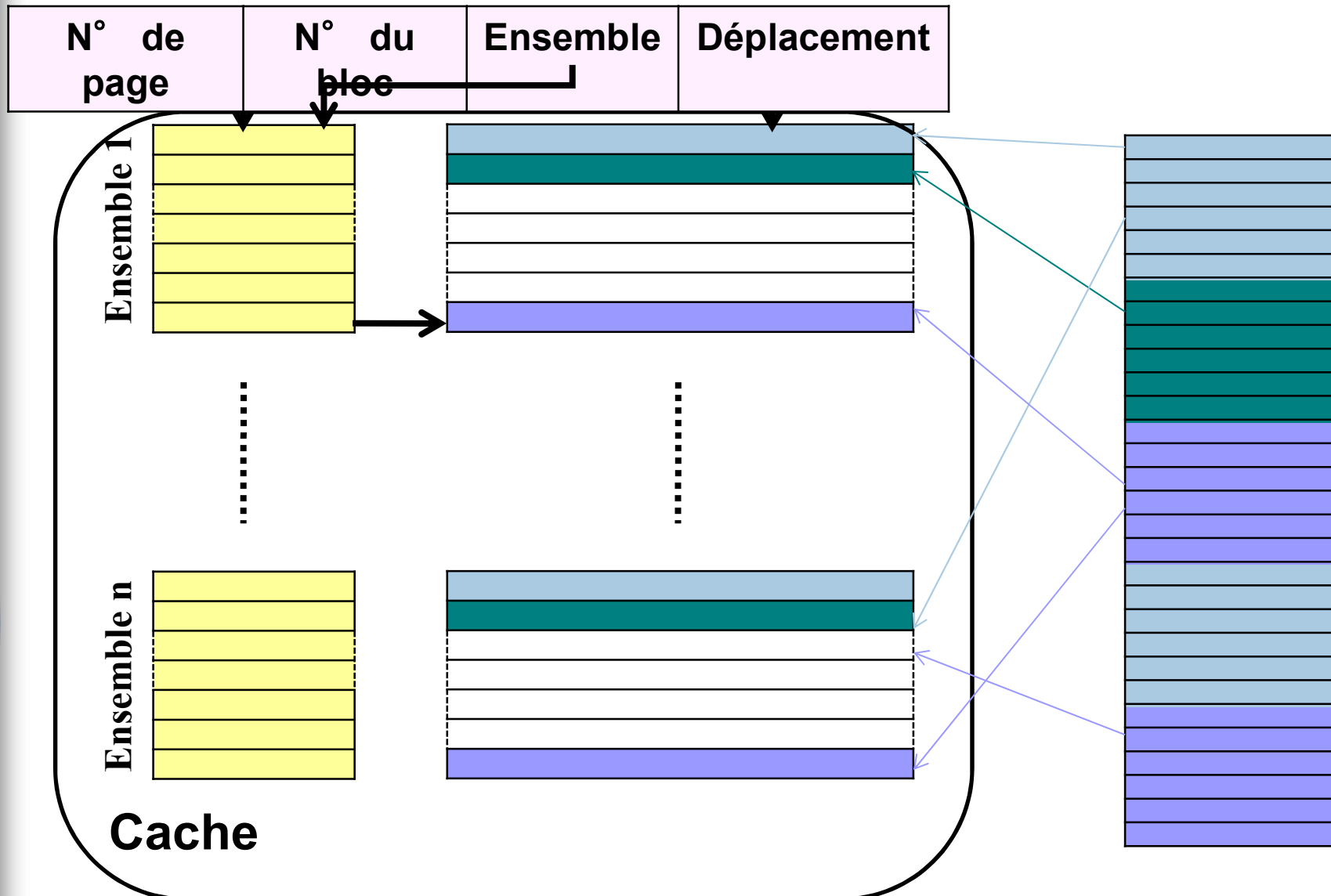




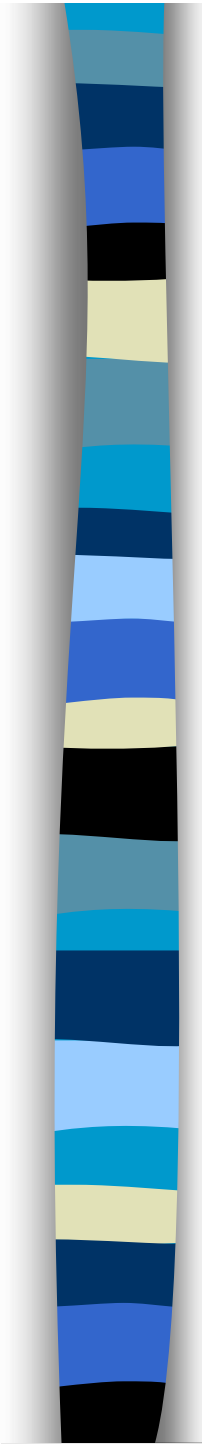
# Mémoire cache associative par ensemble

- ❑ Approche hybride (mixte)
- ❑ Mémoire d'étiquettes contient le N° du bloc, le N° de page, l'emplacement en MC et le déplacement
- ❑ MP et MC découpées en pages pas forcément de même taille
- ❑ Un bloc de la MP est mappé indifféremment dans l'un des blocs d'un ensemble donné de la MC
- ❑ Réduire les comparaisons (comparer à un ensemble de N° de blocs)
- ❑ Peu efficace en cas d'accès à des blocs concurrents

# Exemple de MC associative par ensemble



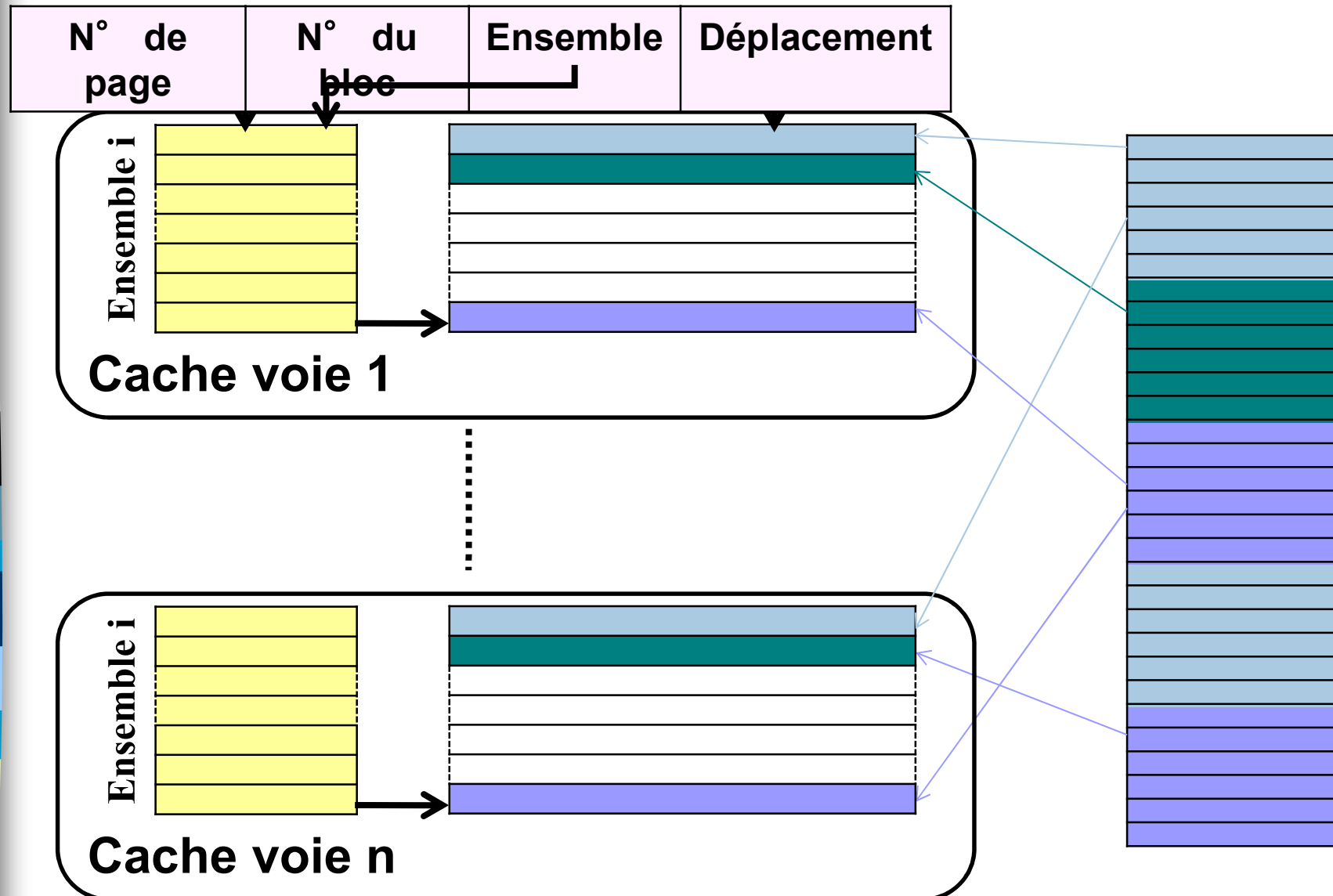




# Mémoire cache associative par ensemble à N voies

- ❑ Constituée de N MC associatives par ensemble
- ❑ Un bloc de la MP est mappé indifféremment dans l'un des blocs d'un ensemble donné de l'une des N MC
- ❑ Comparaison parallèle avec des ensembles des N MC
- ❑ Plusieurs blocs concurrents peuvent coexister en MC
- ❑ Bon compromis rapidité/efficacité

# Exemple de MC associative à N voies



# Algorithmes de remplacement

- ❑ Aléatoire : le bloc le + sollicité
  - Très rapide, peu efficace
- ❑ FIFO : le + ancien (file d'attente circulaire)
  - Rapide, assez réaliste
  - Problème : le + ancien = l'un des + sollicités
- ❑ LFU (Least Frequently Used) : le – utilisé (compteur)
  - Problème : bloc – utilisé = bloc prochainement demandé
- ❑ LRU (Least Recently Used) : le + anciennement utilisé
  - Moins rapide, Performant (éviter de futurs défauts de cache), Gestion complexe



# Performances des caches (1/2)

## ❑ Stratégie d'écriture

- Ecriture simultanée (Write Through), Pb : trafic mémoire important
- Ecriture différée ou réécriture (Write Back)
  - Modifier uniquement le cache ; Positionner un bit d'état de modification
  - En cas de remplacement, écrire dans MP si bit modifié=1
  - Problème : modules d'E/S ou multiprocesseurs accédant à la MP invalide → Solution : circuits complexes.

## ❑ Taille des blocs



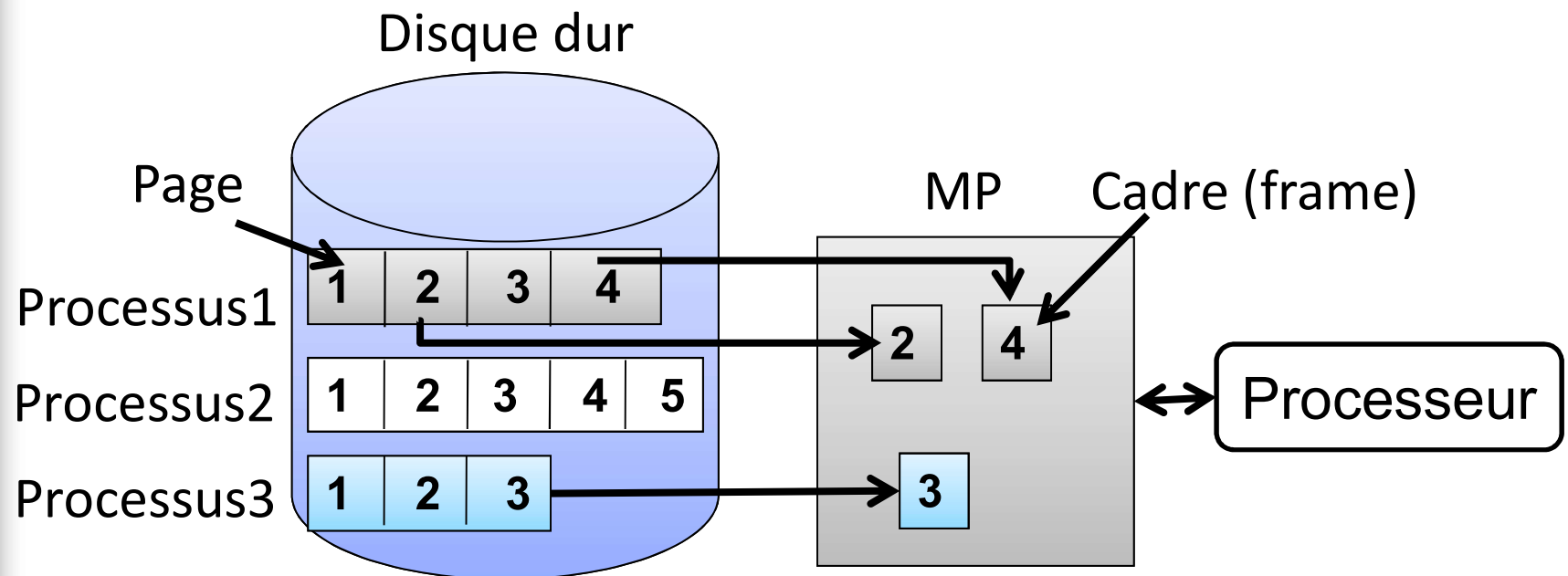
# Performances des caches (2/2)

## □ Nature et niveaux de caches

- Caches multiniveaux : cache interne de niveau 1 et caches externes de niveaux 2 et 3
  - Cache interne (même puce que le processeur) : réduit le temps d'exécution et améliore les performances du système
- Caches unifiés ou séparés : 1 cache interne unique ou 2 caches de données et instructions
  - Ex : caches séparés des processeurs superscalaires (Pentium ou PowerPC favorisant l'exécution parallèle et le préchargement des instructions).
  - Cache séparé : élimine les conflits entre l'unité de lecture/décodage d'instructions et l'unité de traitement (ou d'exécution).

# Mémoire virtuelle

- ❑ Pb : MP ne peut héberger +eurs processus à la fois
- ❑ Idée : utiliser le disque + stockage partiel de pages





# Fonctionnement

- ❑ Double adressage : virtuel+réel
- ❑ Transformation d'adresses par la MMU (Memory Management Unit)
  - MMU implanté actuellement dans la CPU → rapidité
- ❑ Si page en MP, MMU traduit l'adresse virtuelle en réelle (50 ns)
- ❑ Si défaut de page, le SE recherche la page sur le disque et la copie en MP (10 ms)



# Traitement de défaut

- ❑ Suspendre l'exécution du processus en cours
- ❑ Générer une interruption de défaut de page
- ❑ Rapatrier la page demandée
  - S'il faut remplacer une page, elle sera copiée sur le disque en cas de modification (dirty bit=1)
- ❑ Exécuter un autre processus pendant la lecture
- ❑ Générer une interruption de fin de lecture
- ❑ Mettre à jour la table et débloquer le processus initial





# Algorithmes de remplacement

## ❑ FIFO :

- PB: page sollicitée éliminée
- Sol : 2<sup>ème</sup> chance : si non accédée (bit de référence=0 et non modifiée (bit dirty=0), la remplacer sinon passer à la suivante.

## ❑ LRU :

- Non répandu (besoin d'un dispositif rapide de mise à jour à chaque accès du compteur ou de la date).

## ❑ LFU :

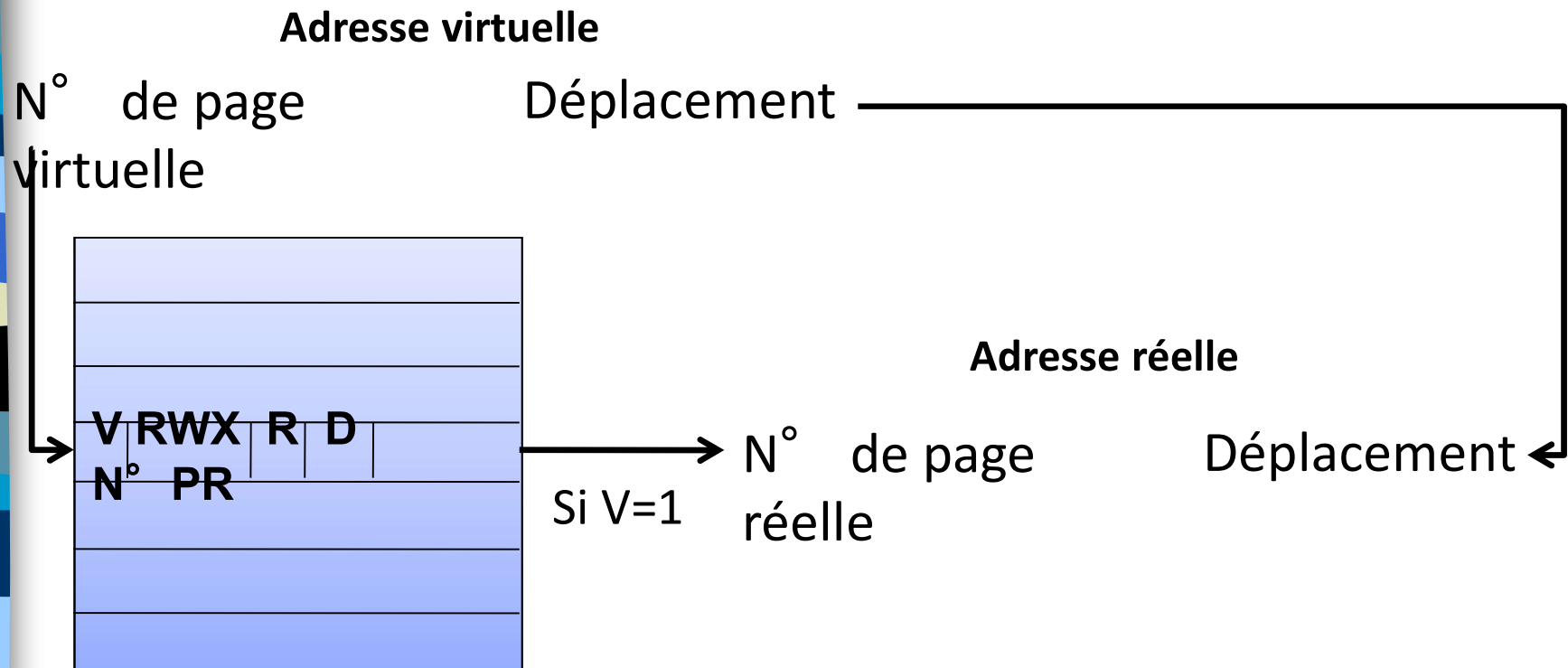
- Pb : idem que LRU.

## ❑ MFU :

- Peu utilisé

# Table des pages

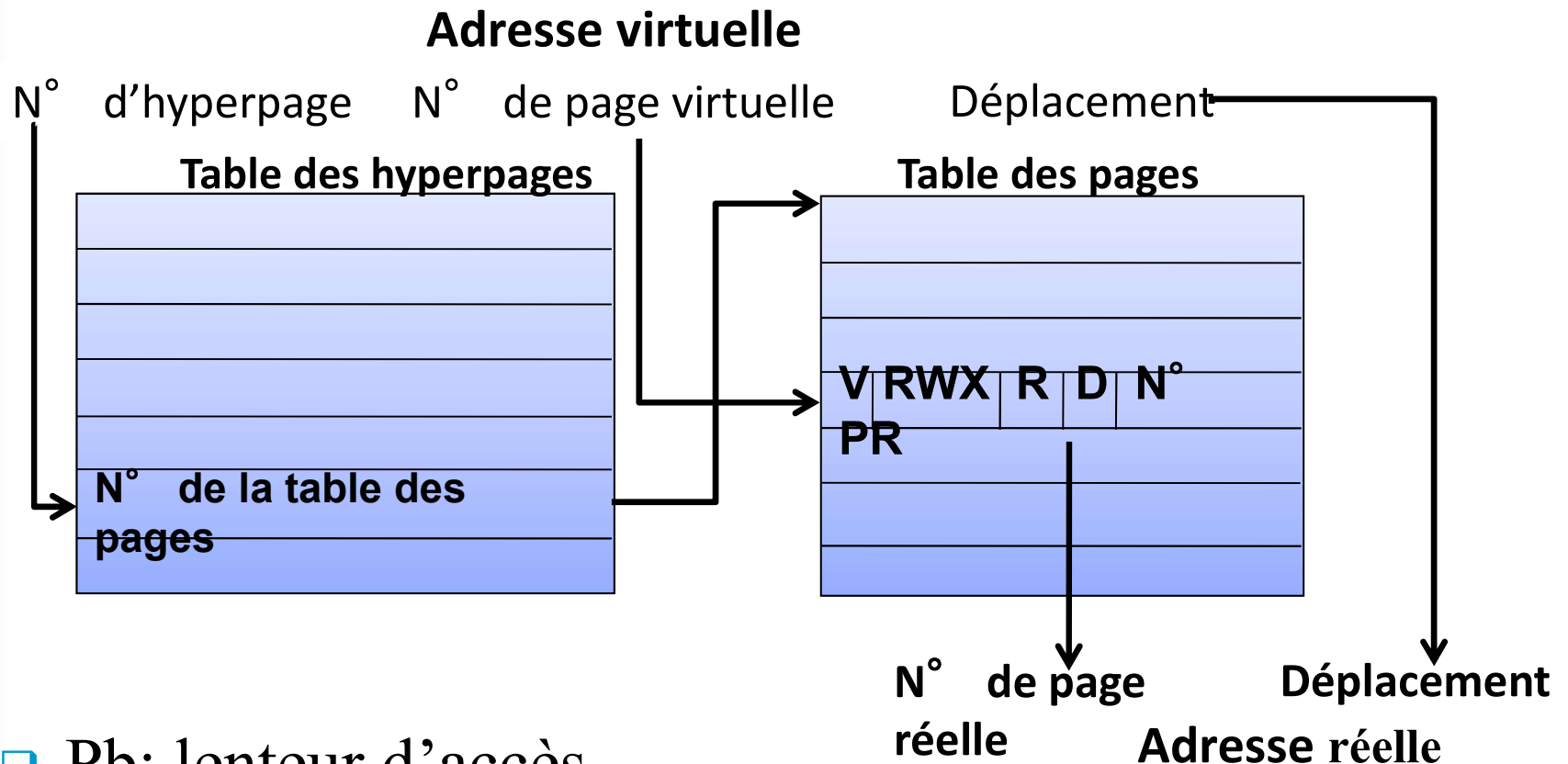
- ❑ Indexée par le numéro de page virtuelle, assure la correspondance avec les numéros de pages réelles



- ❑ Stockée en MP : son adresse dans un registre MMU
- ❑ Pb : table ne tenant pas en MP

# Table de pages à plusieurs niveaux

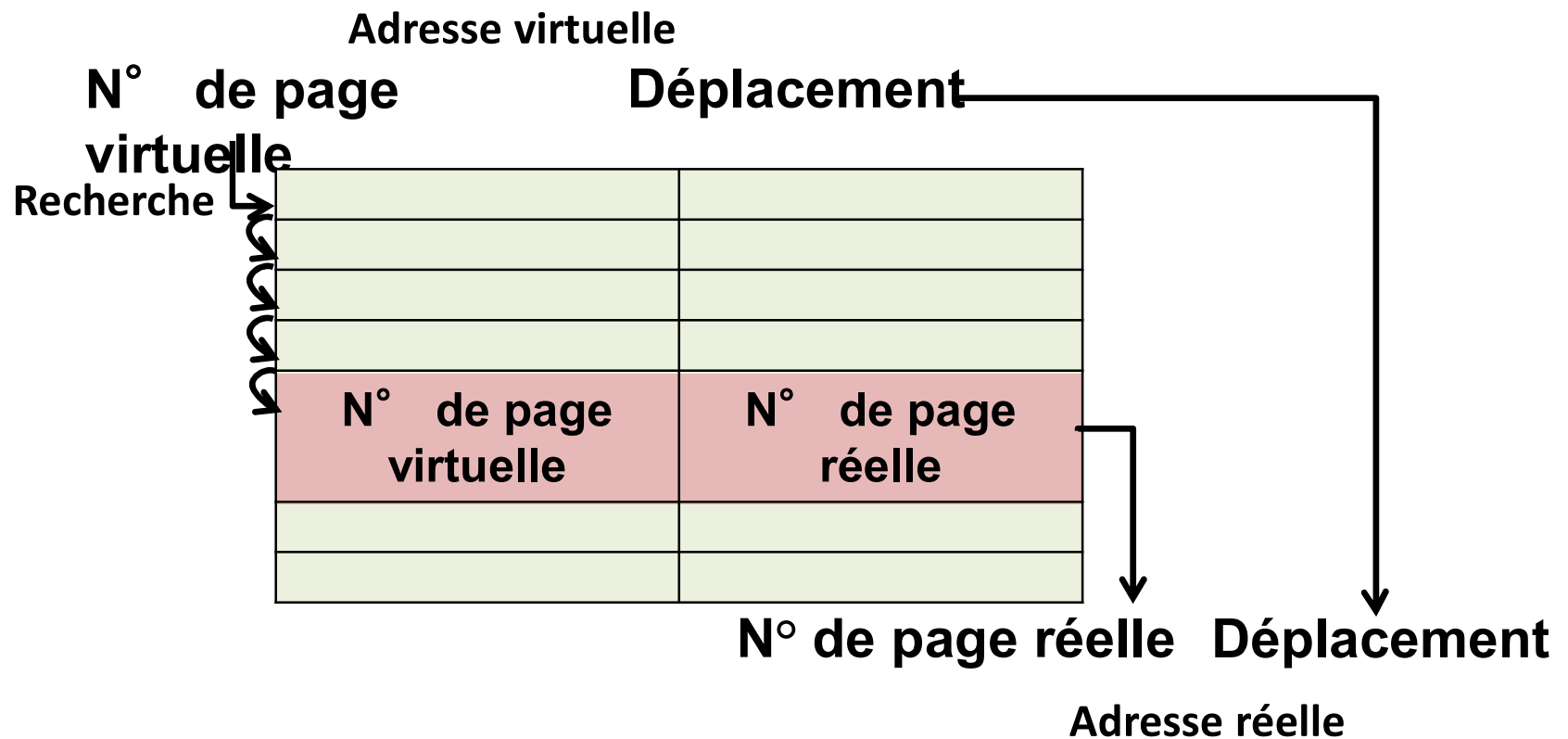
- ❑ Découper les n° de pages en 2 (ou +eurs) niveaux.
- ❑ Garder la table de niveau 1 en MP et charger les autres au besoin.



- ❑ Pb: lenteur d'accès

# Table des pages inverse

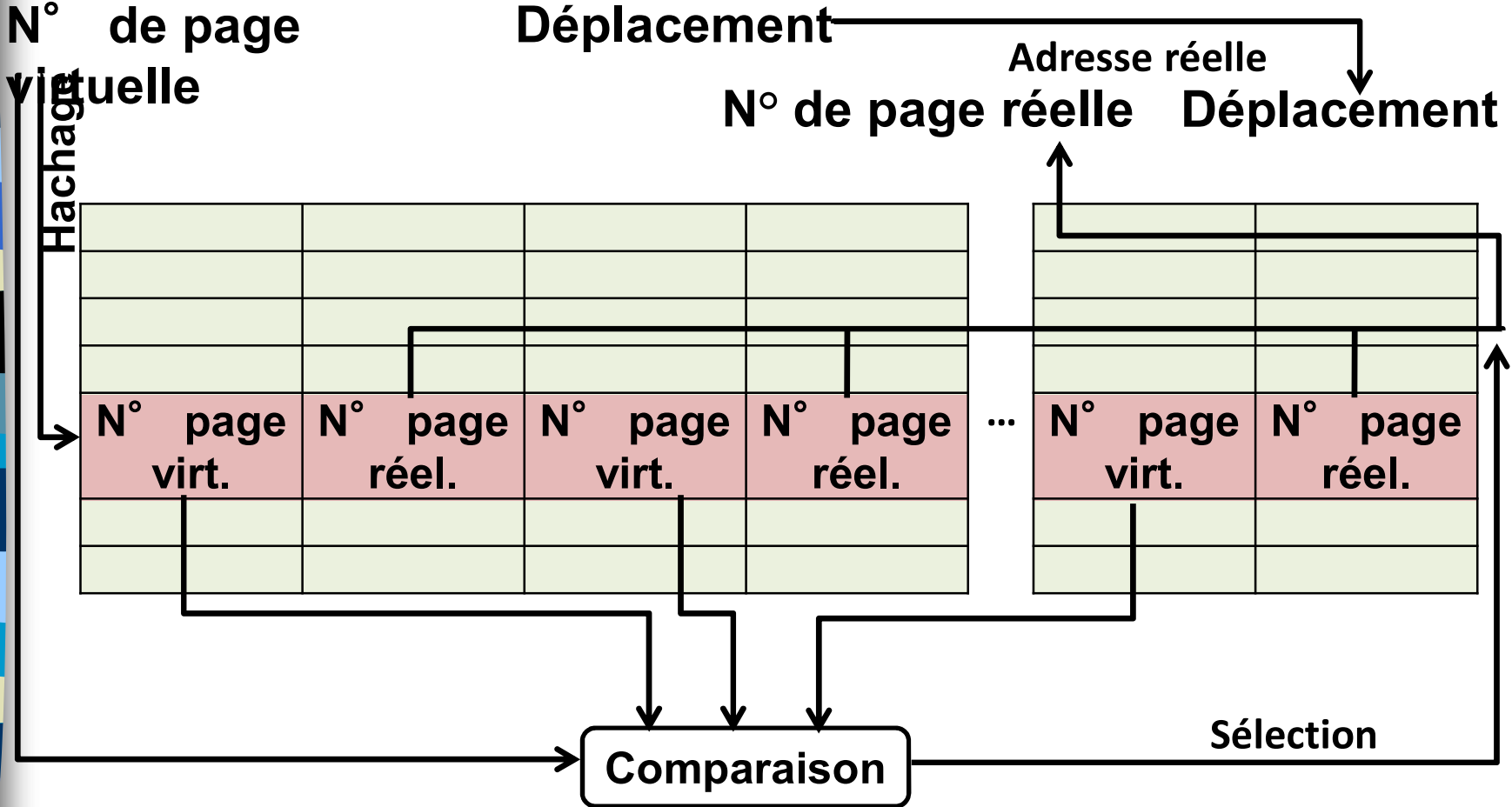
- Contient uniquement les correspondances des pages en MP → taille de table réduite.



- PB: recherche lente (parcours total de la table).

- 

# Adresse virtuelle

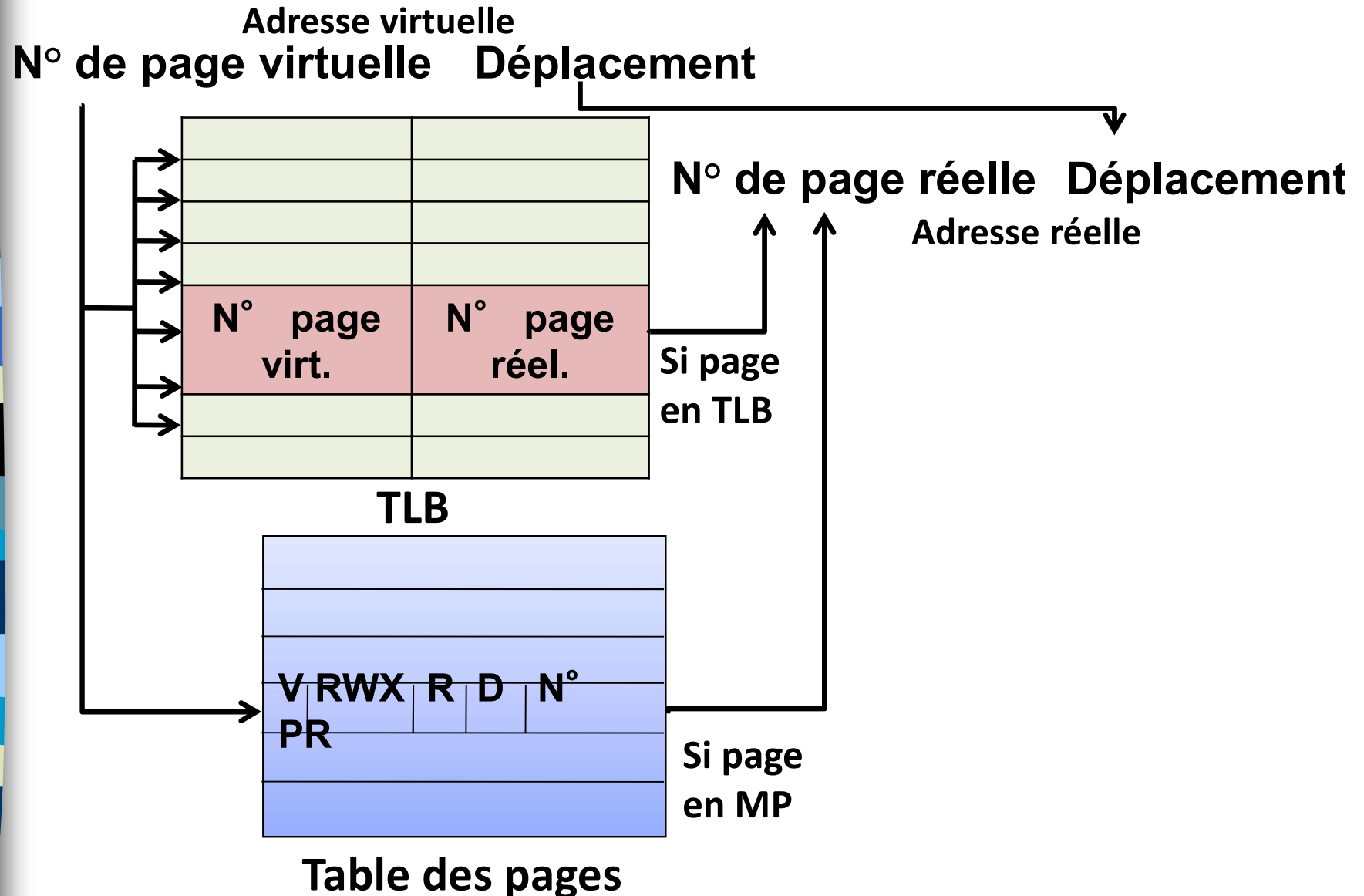




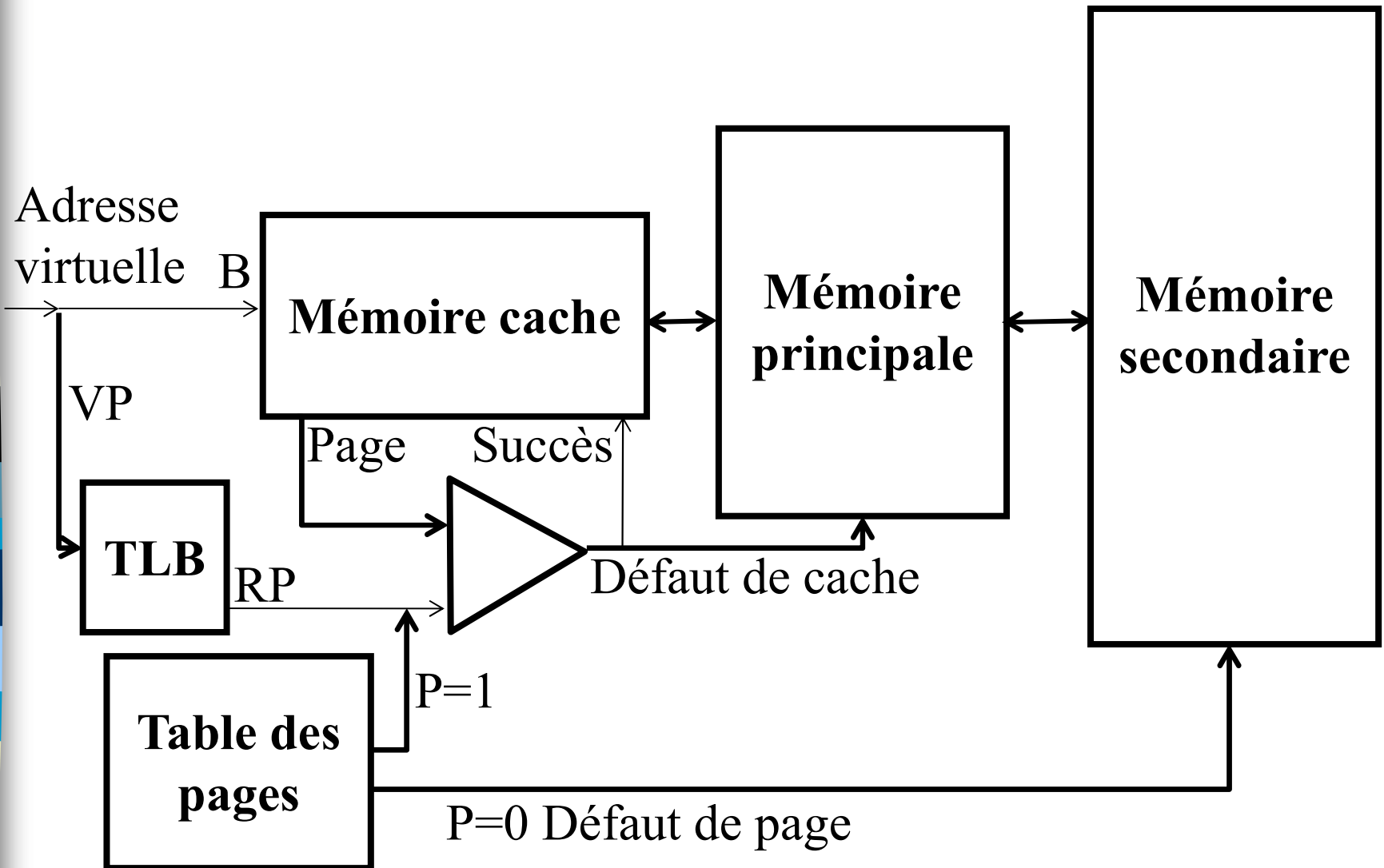
# Tampon de traduction anticipée

- ❑ Pb : Accès à la table + accès à l'information  
→ lenteur.
- ❑ Utiliser un TLB (Translation Lookaside Buffer).
- ❑ TLB : petite mémoire associative rapide en MMU (256 o à qq Ko).
  - Contient des couples de dernières pages accédées.
- ❑ La page virtuelle est recherchée en parallèle dans la table des pages et le TLB.

# Traduction utilisant un TLB

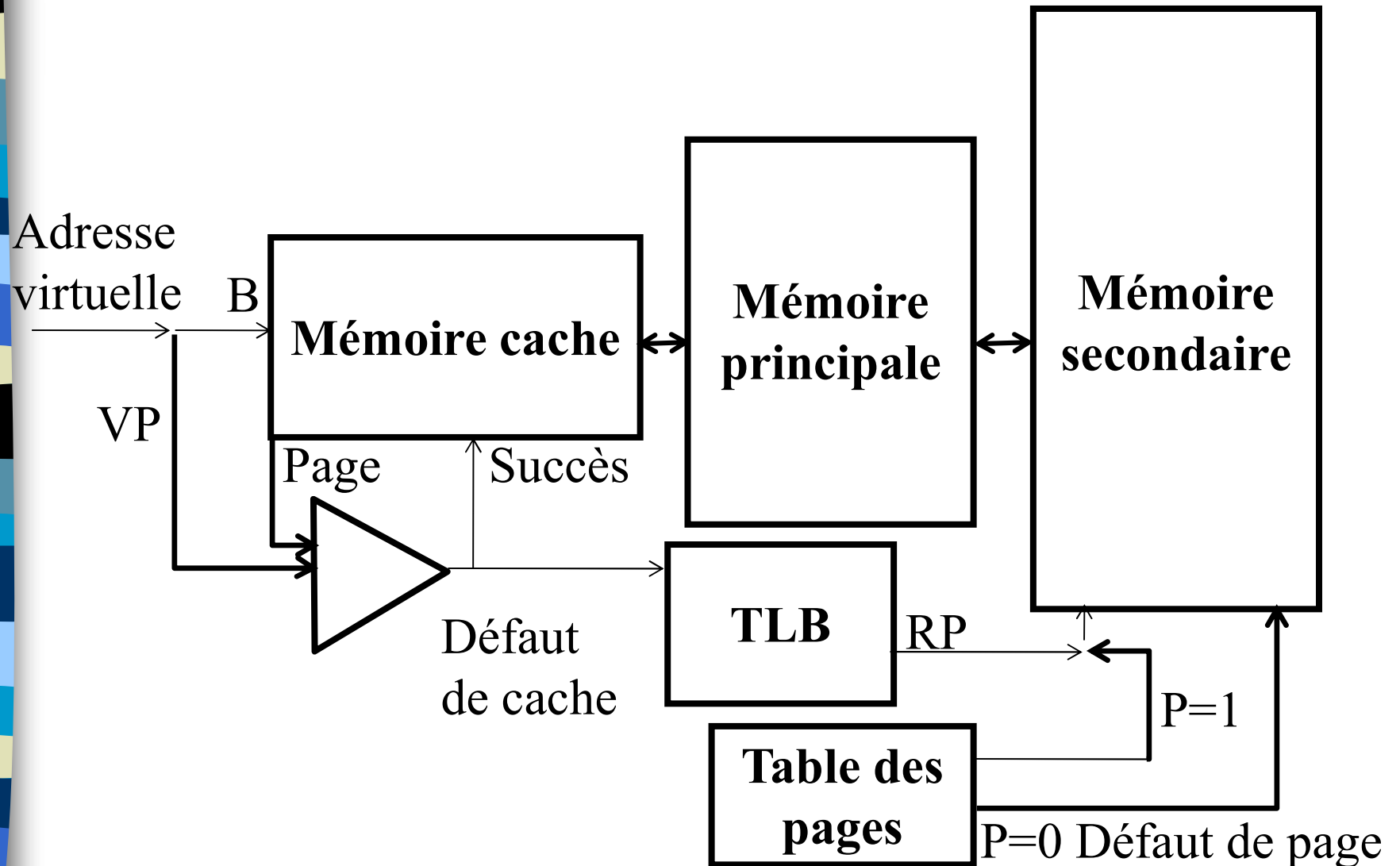


# Recherche de pages dans un système à mémoire cache (1/2)



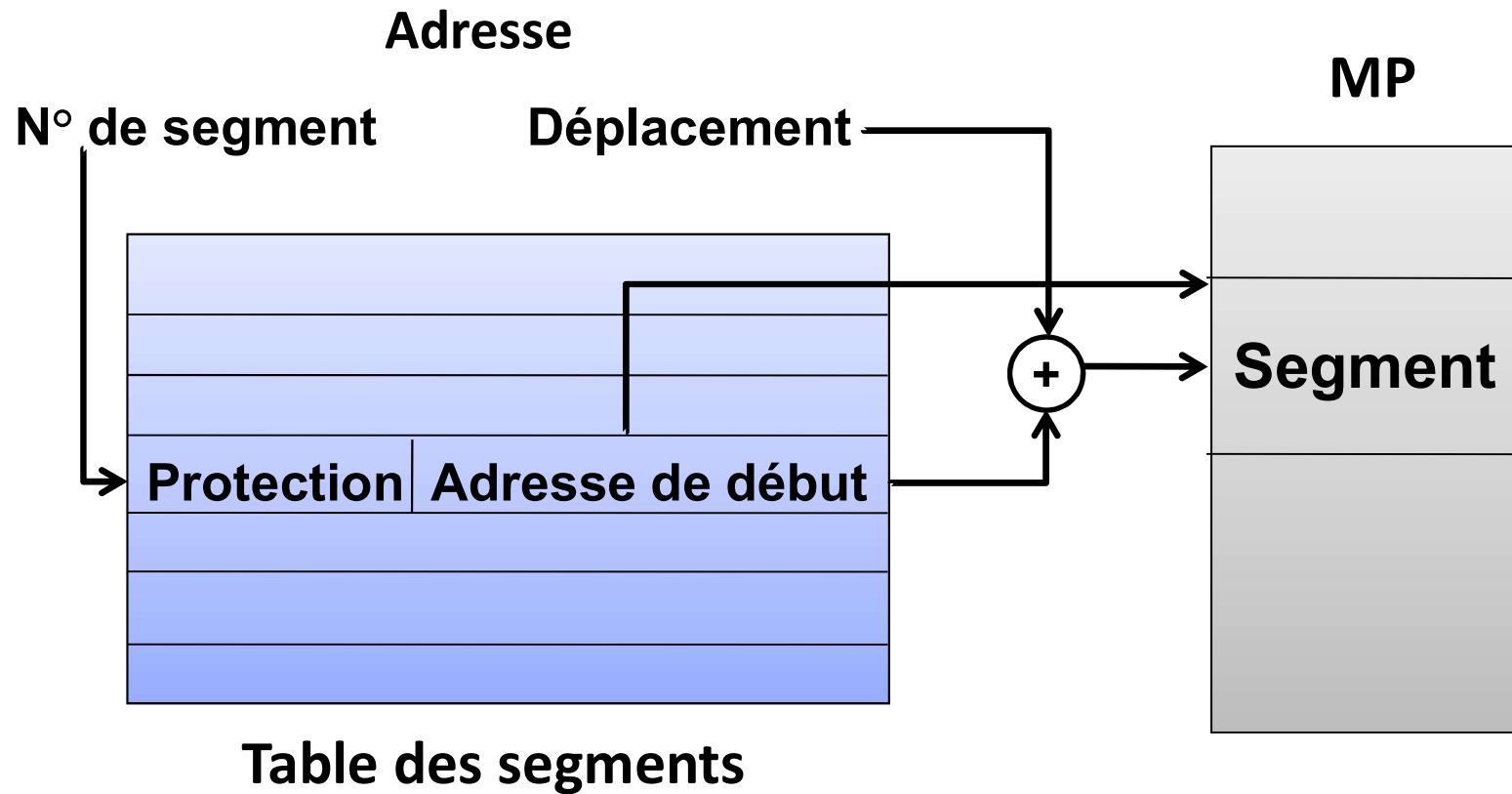


## Recherche de pages dans un système à mémoire cache (2/2)



# Segmentation

- ❑ Découper un processus en segments



# Segmentation et pagination

- ❑ Découper un processus en segments

