ARCHI – Architecture des ordinateurs

Sylvain Brandel
2023 – 2024
sylvain.brandel@univ-lyon1.fr

CM 12

MÉMOIRE CACHE
PIPELINE

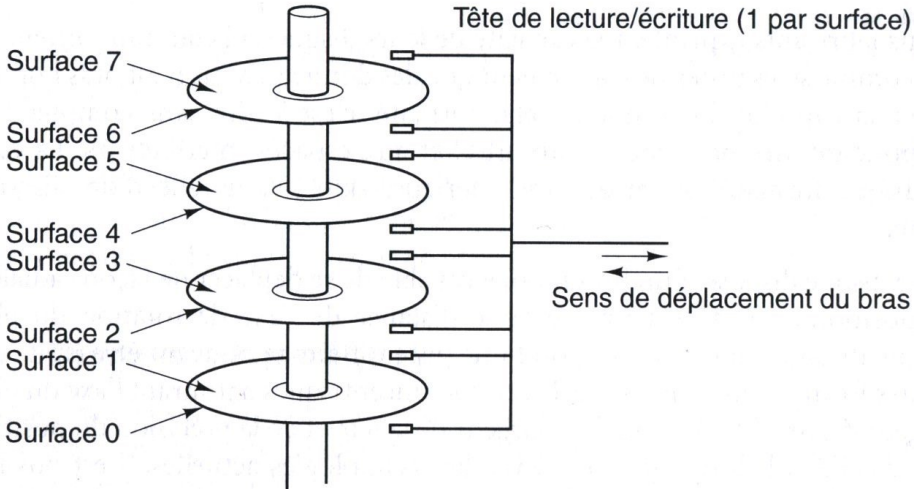
Hiérarchie mémoire

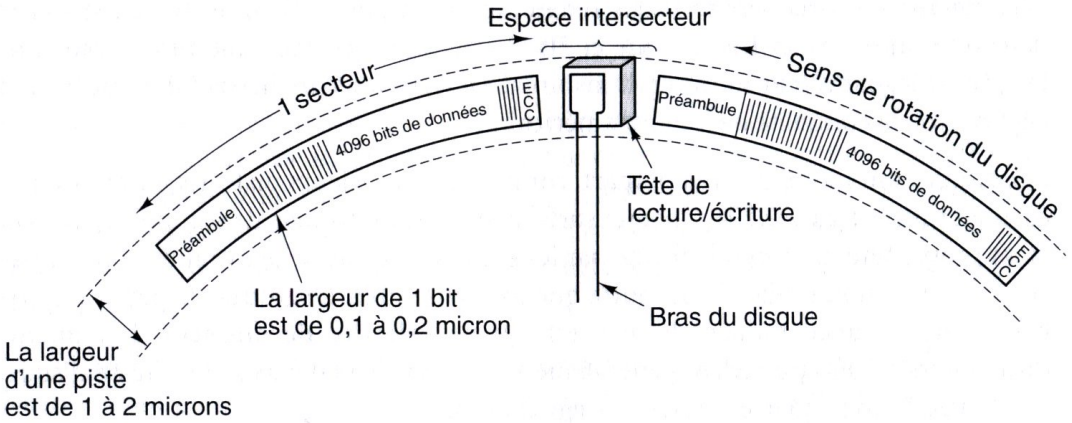
- Mémoires principales
  - Registres
  - Cache
  - RAM
- Mémoires secondaires
  - Supports magnétiques
    - HD (Hard Disc)
    - Disquettes, bandes magnétiques …
  - Supports optiques
    - CD-ROM (Compact Disc ROM), CD-RW (ReWritable)
    - DVD (Digital Versatil Disc)
    - HD-DVD (High Density) / BD (Blu-ray Disc)
  - Mémoires flash
    - Clés usb
    - SSD (Solid Stack Drive)

Hiérarchie mémoire

Support	Temps d'accès	Débit	Capacité
Registres	1 ns	-	kio
RAM	5 – 50 ns	1 – 20 Gio/s	Gio
SSD	0,1 ms	30 Mio/s – 3 Gio/s	Tio
HD	3 – 20 ms	10 – 300 Mio/s	Tio
CD	120 ms	1 – 8 Mio/s	650 Mio
DVD	140 ms	2 – 22 Mio/s	4,6 – 17 Gio
BD		36 – 72 Mio/s	7,5 – 128 Gio

- Ex : disque dur

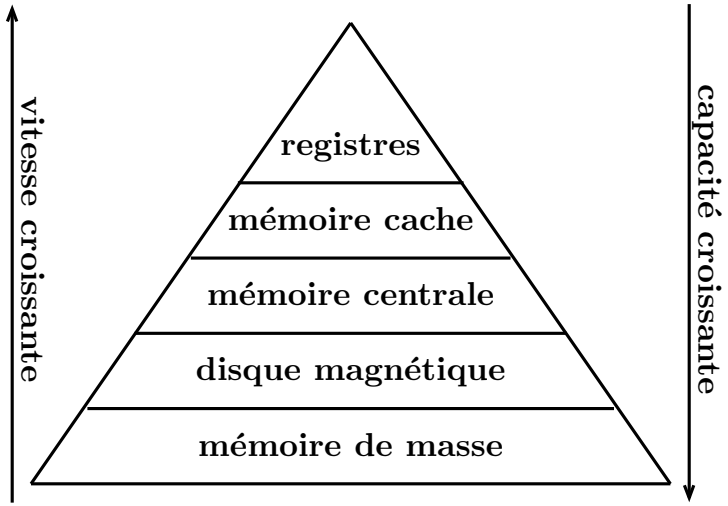




Hiérarchie mémoire

Support	Temps d'accès	Débit	Capacité
Registres	1 ns	-	kio
Cache	2 – 3 ns		Mio
RAM	5 – 50 ns	1 – 20 Gio/s	Gio
SSD	0,1 ms	30 Mio/s – 3 Gio/s	Tio
SSD	3 – 20 ms	10 – 300 Mio/s	Tio
CD	120 ms	1 – 8 Mio/s	650 Mio
DVD	140 ms	2 – 22 Mio/s	4,6 – 17 Gio
BD		36 – 72 Mio/s	7,5 – 128 Gio

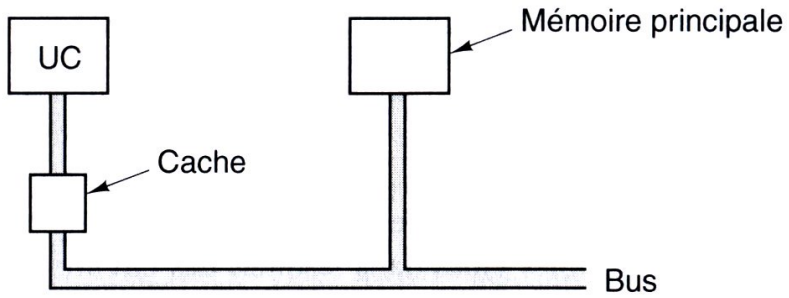
- Cache : stockage des mots
  les plus souvent accédés
- Compromis à trouver :
  - Vitesse ↑ → coût ↑
  - Capacité ↑ → temps d'accès ↑



Mémoire cache

- Deux exemples d'organisation :
  - Cache à correspondance directe
  - Cache associatif

- On suppose un seul niveau de cache



- Lors d'un accès mémoire :
  - Donnée présente dans le cache → UCT y accède directement
  - Donnée absente du cache → chargement dans le cache puis accès

Mémoire cache

Cache à correspondance directe

Ex : ordinateur avec

- RAM :
  - 4 Gio : 2^{32} octets adressables par octet
  - Divisée en blocs de taille fixe appelés lignes de cache
    - 1 ligne → 2^5 = 32 octets
    - Mémoire → 2^{27} lignes de cache
    - Chaque ligne de cache identifiée par un entier sur 27 bits
- Mémoire cache :
  - 64 kio : 2^{11} entrées de 32 octets

- « Mapper » 2^{11} entrées sur 2^{27} lignes de cache :

- À partir d'une adresse n d'une donnée d en RAM

  → n appartient à quelle ligne de cache ?

  → d doit être rangée dans quelle entrée du cache ?

Mémoire cache

Cache à correspondance directe

- Décomposition d'une adresse n en RAM :

16 bits (31 – 16)	11 bits (15 – 5)	5 bits (4 – 0)
Indicateur	Entrée	Octet

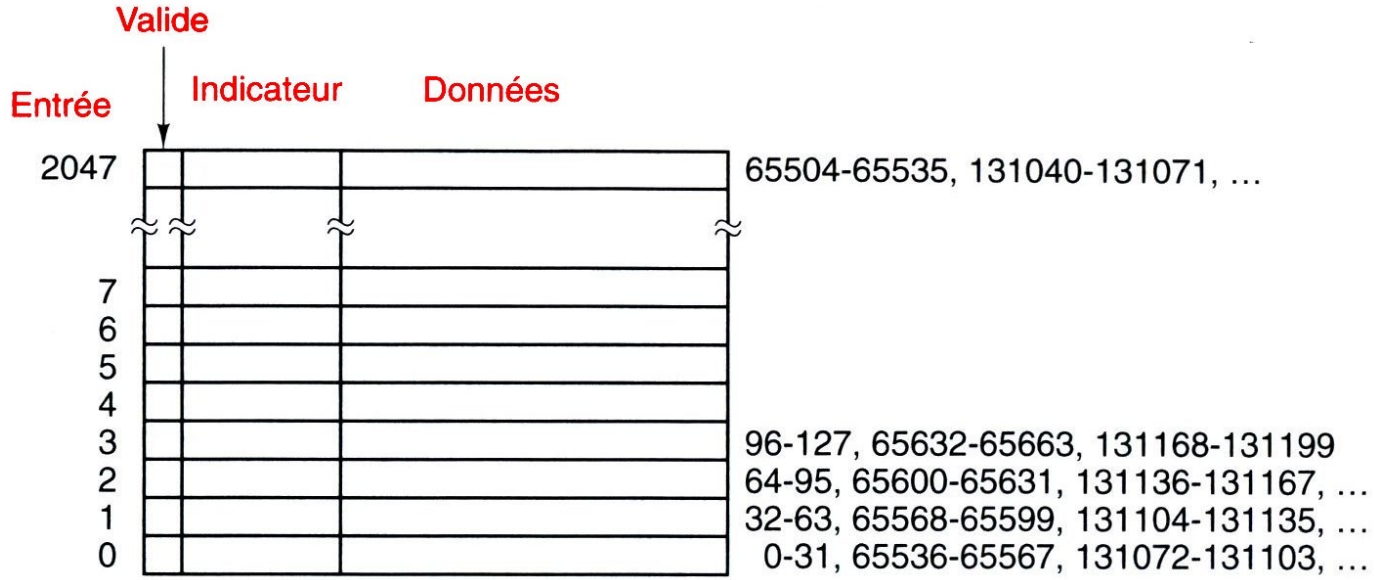
- Indicateur + Entrée (27 bits) : identification de chaque ligne de cache
- Entrée (11 bits) : identification d'une entrée du cache
- Octet (5 bits) : identification d'un des 32 octets d'une ligne ou d'une entrée

- Toutes les lignes avec même Entrée → même entrée du cache
- Toutes les lignes avec même Indicateur → consécutives en RAM

Mémoire cache

Cache à correspondance directe

- Organisation de la mémoire cache :



- Une ligne est stockée dans l'entrée correspondant à son Entrée
- Indicateur correspond à Indicateur : ligne de cache d'origine en RAM
- Valide indique si l'entrée contient des données cohérentes cache / RAM
- Données contient la copie d'une ligne de cache

Mémoire cache

Cache à correspondance directe

- Lecture d'un octet en RAM adressé par Indicateur Entrée Octet
  - Entrée → entrée du cache
  - Si Indicateur = Indicateur
    → cache hit : lecture de l'octet dans le cache
      - à l'adresse Entrée
      - et à la position Octet dans Données
  - Si Indicateur != Indicateur
    → cache miss : la ligne accédée est copiée de la RAM vers le cache

- Ecriture : cohérence entre cache et RAM ? 2 stratégies
  - Immédiate : MàJ simultanée cache et RAM → plus d'intérêt du cache
  - Différée : écriture dans la RAM quand éviction du cache
    - Si la ligne écrite est dans le cache → MàJ du cache et Valide ← 0
    - Sinon chargement dans le cache → MàJ du cache et Valide ← 0

Mémoire cache

Cache à correspondance directe

- Principe de localité
  - Localité temporelle : une case mémoire utilisée le sera à nouveau
  - Localité spatiale : si on utilise la case n, on utilisera $n+1, \mathrm{n}+2$ ...

- Le programmeur doit respecter ce principe
  - Ex. précédent, accès intensif à l'adresse 0 et à l'adresse 65536

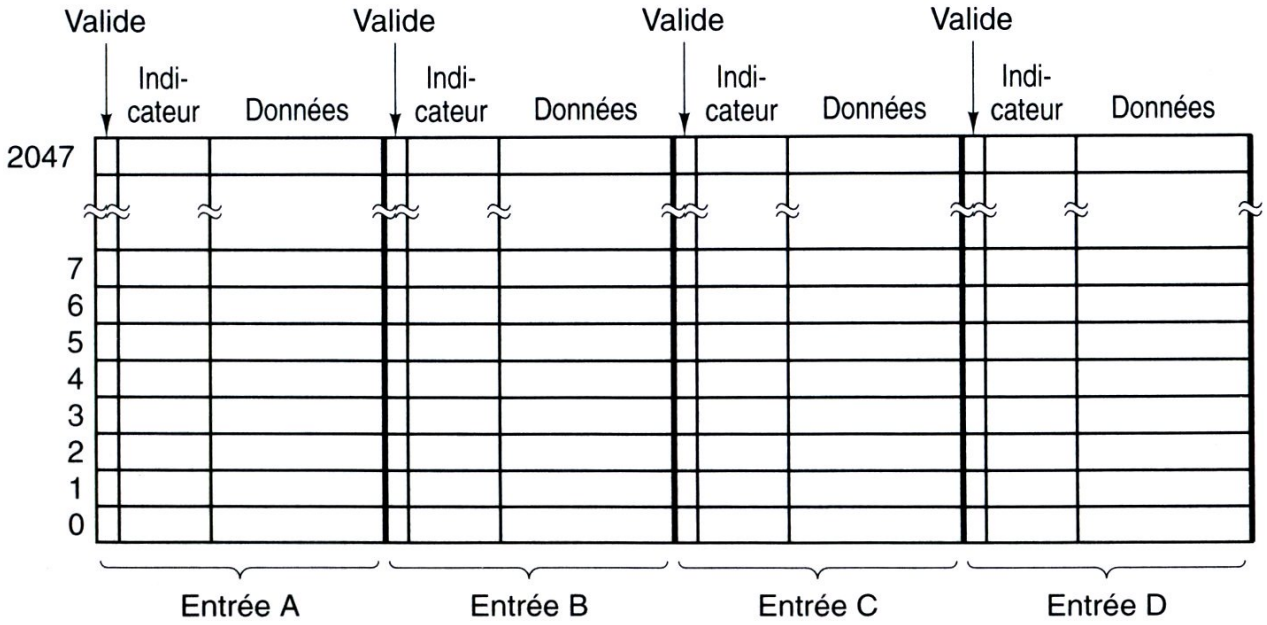
    → Même champ Entrée

    → Conflits permanents sur l'entrée 0

  ⇒ Cache associatif : $n \geq 2$ lignes pour chaque Entrée

Mémoire cache
Cache associatif

- Ex : $n = 4$



- Nouveau problème : quelle ligne remplacer dans le cache ?
  - Aléatoire
  - FIFO (First In First Out)
  - LRU (Least Recently Used)
  - …

    ⇒ On trouvera toujours des algorithmes qui mettront le cache en défaut

Mémoire cache

- Programmer en fonction du cache

- Ex : produit matriciel $C = A \times B$

$$C_{i,j} = \sum_{k=1}^{N} A_{i,k} \times B_{k,j}$$

```
double A[M][N];
double B[N][M];
double C[M][M];

for (int i=0; i<M; i=i+1)
  for (int j=0; j<M; j=j+1) {
    C[i][j] = A[i][k] * B[k][j];
    for (int k=1; k<N; k=k+1)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
  }
```

Mémoire cache

```
A : 2 lignes et 3 colonnes
B : 3 lignes et 2 colonnes
C : 2 lignes et 2 colonnes

                          B[0][0] B[0][1]
                          B[1][0] B[1][1]
                          B[2][0] B[2][1]

A[0][0] A[0][1] A[0][2] C[0][0] C[0][1]
A[1][0] A[1][1] A[1][2] C[1][0] C[1][1]


En mémoire :
A[0][0] A[0][1] A[0][2] A[1][0] A[1][1] A[1][2]
B[0][0] B[0][1] B[1][0] B[1][1] B[2][0] B[2][1]
C[0][0] C[0][1] C[1][0] C[1][1]
```

Mémoire cache

- ijk (naif)

```
double A[N][N];
double B[N][N];
double C[N][N];

for (int i=0; i<N; i=i+1)
  for (int j=0; j<N; j=j+1)
    for (int k=0; k<N; k=k+1)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

```
(i,j,k)
(0,0,0) C[0][0] = C[0][0] + A[0][0] * B[0][0];
(0,0,1) C[0][0] = C[0][0] + A[0][1] * B[1][0];
(0,0,2) C[0][0] = C[0][0] + A[0][2] * B[2][0];
(0,1,0) C[0][1] = C[0][1] + A[0][0] * B[0][1];
(0,1,1) C[0][1] = C[0][1] + A[0][1] * B[1][1];
(0,1,2) C[0][1] = C[0][1] + A[0][2] * B[2][1];
(1,0,0) C[1][0] = C[1][0] + A[1][0] * B[0][0];
(1,0,1) C[1][0] = C[1][0] + A[1][1] * B[1][0];
(1,0,2) C[1][0] = C[1][0] + A[1][2] * B[2][0];
(1,1,0) C[1][1] = C[1][1] + A[1][0] * B[0][1];
(1,1,1) C[1][1] = C[1][1] + A[1][1] * B[1][1];
(1,1,2) C[1][1] = C[1][1] + A[1][2] * B[2][1];
...
```

Mémoire cache

- kji

```
double A[N][N];
double B[N][N];
double C[N][N];

for (int k=0; k<N; k=k+1)
  for (int j=0; j<N; j=j+1)
    for (int i=0; i<N; i=i+1)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

```
(i,j,k)
(0,0,0) C[0][0] = C[0][0] + A[0][0] * B[0][0];
(1,0,0) C[1][0] = C[1][0] + A[1][0] * B[0][0];
(0,1,0) C[0][1] = C[0][1] + A[0][0] * B[0][1];
(1,1,0) C[1][1] = C[1][1] + A[1][0] * B[0][1];
(0,0,1) C[0][0] = C[0][0] + A[0][1] * B[1][0];
(1,0,1) C[1][0] = C[1][0] + A[1][1] * B[1][0];
(0,1,1) C[0][1] = C[0][1] + A[0][1] * B[1][1];
(1,1,1) C[1][1] = C[1][1] + A[1][1] * B[1][1];
(0,0,2) C[0][0] = C[0][0] + A[0][2] * B[2][0];
(1,0,2) C[1][0] = C[1][0] + A[1][2] * B[2][0];
(0,1,2) C[0][1] = C[0][1] + A[0][2] * B[2][1];
(1,1,2) C[1][1] = C[1][1] + A[1][2] * B[2][1];
...
```

Mémoire cache

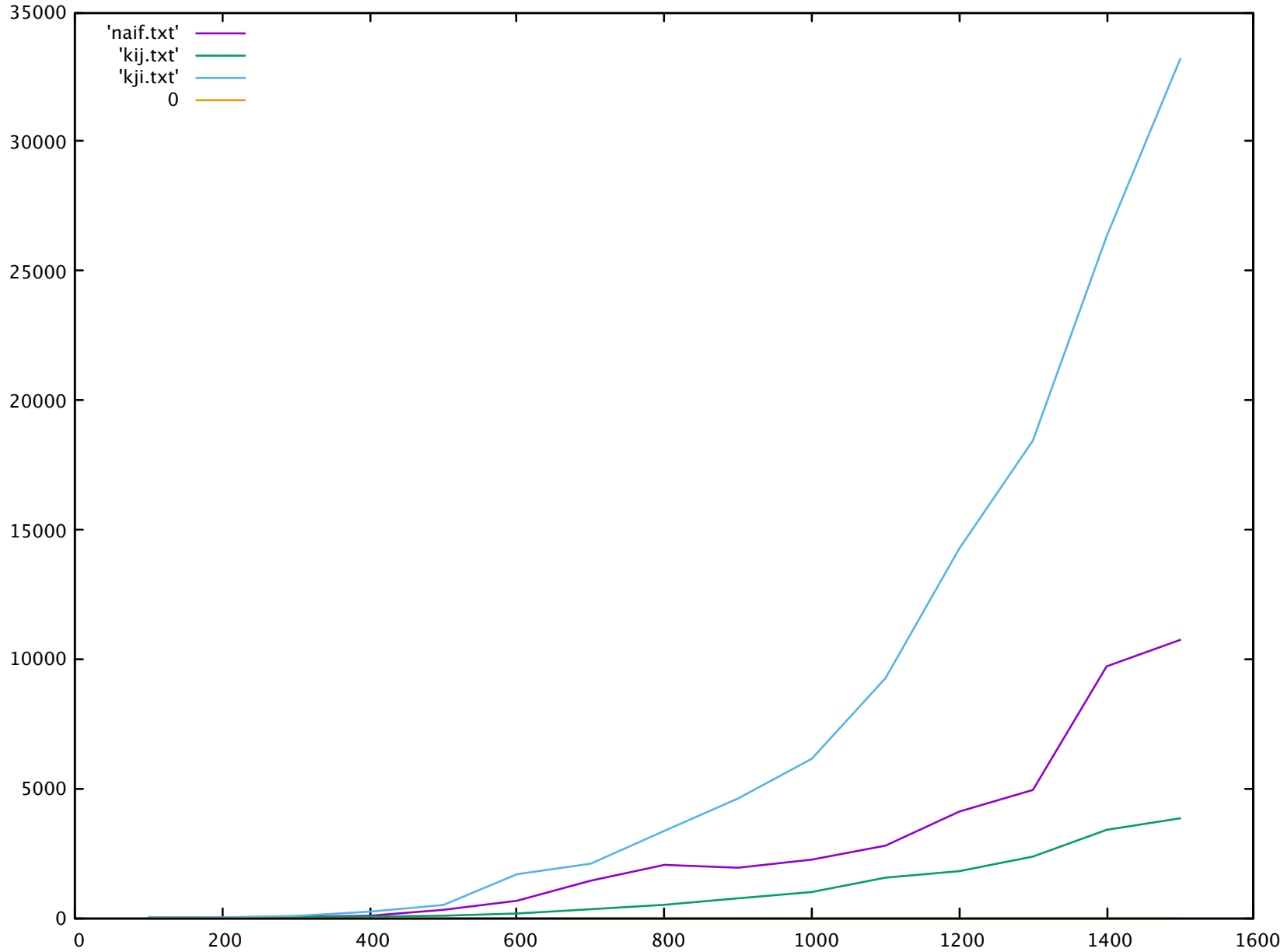
- kij

```
double A[N][N];
double B[N][N];
double C[N][N];

for (int k=0; k<N; k=k+1)
  for (int i=0; i<N; i=i+1)
    for (int j=0; j<N; j=j+1)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

```
(i,j,k)
(0,0,0) C[0][0] = C[0][0] + A[0][0] * B[0][0];
(0,1,0) C[0][1] = C[0][1] + A[0][0] * B[0][1];
(1,0,0) C[1][0] = C[1][0] + A[1][0] * B[0][0];
(1,1,0) C[1][1] = C[1][1] + A[1][0] * B[0][1];
(0,0,1) C[0][0] = C[0][0] + A[0][1] * B[1][0];
(0,1,1) C[0][1] = C[0][1] + A[0][1] * B[1][1];
(1,0,1) C[1][0] = C[1][0] + A[1][1] * B[1][0];
(1,1,1) C[1][1] = C[1][1] + A[1][1] * B[1][1];
(0,0,2) C[0][0] = C[0][0] + A[0][2] * B[2][0];
(0,1,2) C[0][1] = C[0][1] + A[0][2] * B[2][1];
(1,0,2) C[1][0] = C[1][0] + A[1][2] * B[2][0];
(1,1,2) C[1][1] = C[1][1] + A[1][2] * B[2][1];
...
```

Mémoire cache



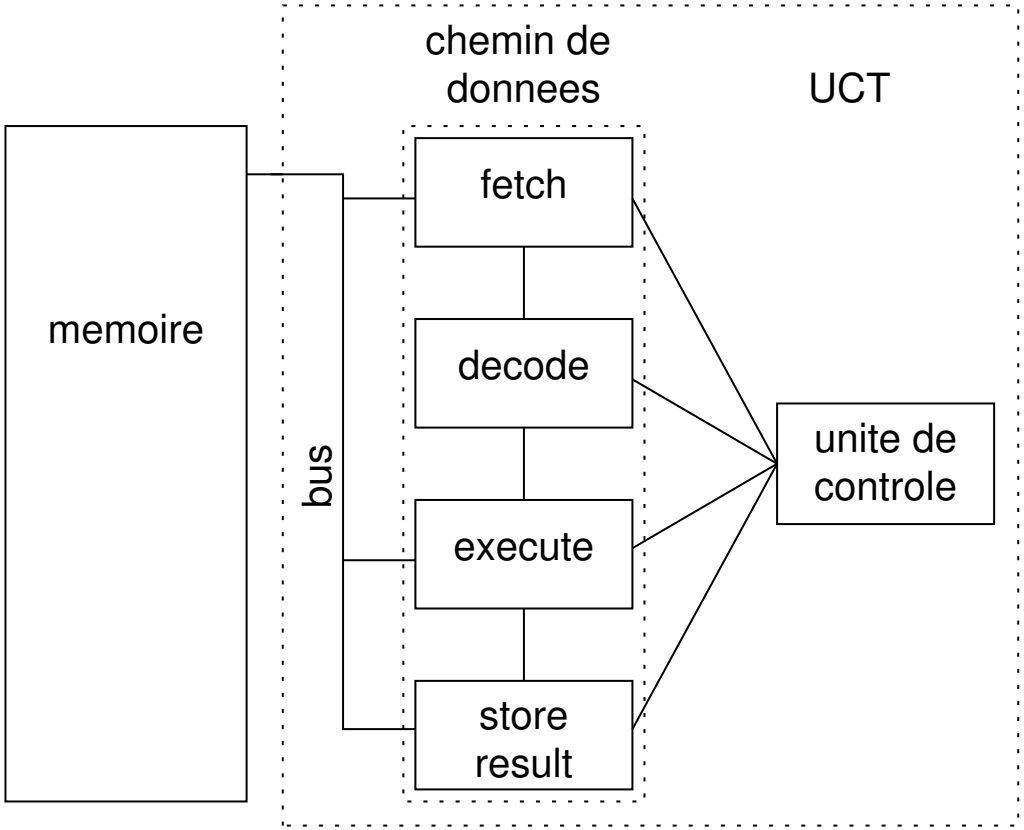
Parallélisme d'instruction
Pipeline

- Cycle d'instruction typique :
  - FETCH : chargement de l'instruction depuis la RAM
  - DECODE : décodage de l'instruction
  - EXECUTE : exécution de l'instruction
  - STORE RESULT : stockage des résultats en RAM
- Chaque phase : un ou plusieurs cycles d'horloge

- Supposons chaque instruction découpée en 4 phases,
  et chaque phase = 1 cycle d'horloge :
  1. FE
  2. DE
  3. EX
  4. SR

Parallélisme d'instruction
Pipeline

- Exemple d'organisation

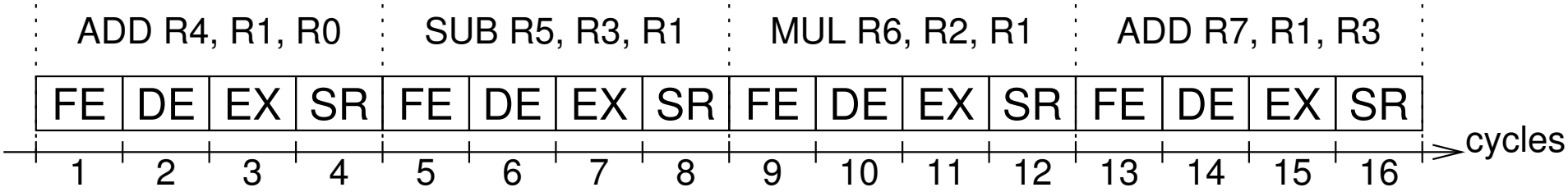


Parallélisme d'instruction
Pipeline

- Exemple

```
ADD R4, R1, R0      ; R4 <- R1 + R0
SUB R5, R3, R1      ; R5 <- R3 - R1
MUL R6, R2, R1      ; R6 <- R2 * R1
ADD R7, R1, R3      ; R7 <- R1 + R3
```

- Exécution séquentielle : UC active FE → DE → EX → SR dans l'ordre



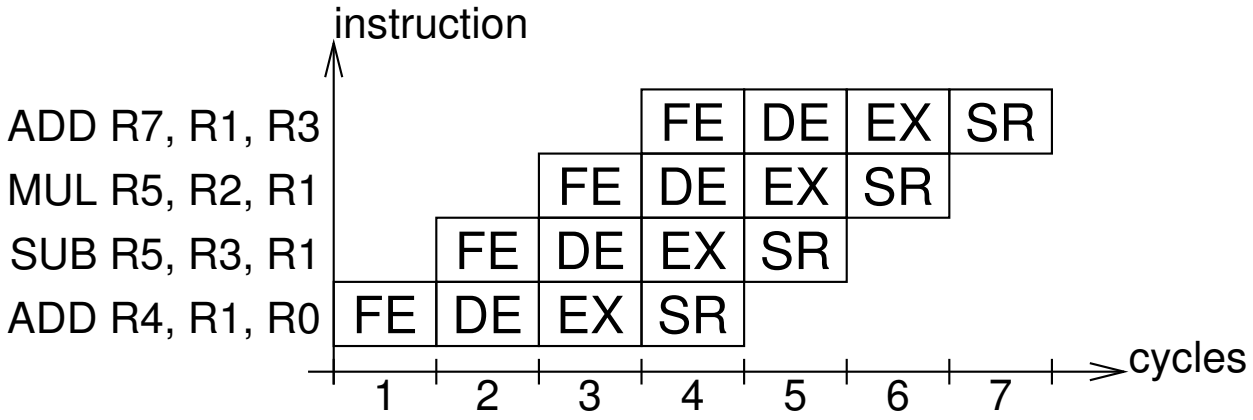
→ 16 cycles

Parallélisme d'instruction
Pipeline

- Si l'UC active les 4 unités du chemin de données à chaque cycle

  → Pipeline de 4 étages

  – Cycle 1 : Instr. 1 – phase FE
  – Cycle 2 : Instr. 1 – phase DE et Instr. 2 – phase FE
  – …



→ 7 cycles (dans un monde idéal)

Parallélisme d'instruction
Dépendances entre instructions

- Mais pas toujours possible de terminer l'exécution d'une instruction à chaque cycle :
  - Latence d'accès à la mémoire (défaut de cache par ex.)
  - Dépendance entre instructions :
    - Dépendance de données
    - Dépendance de noms
    - Dépendance de contrôle

- Dépendance de données :
  - Une instruction j dépend d'une instruction i si i produit un résultat qui est opérande source de j
  - Exemple

```
ADD R2, R0, R1
ADD R3, R2, R1
```

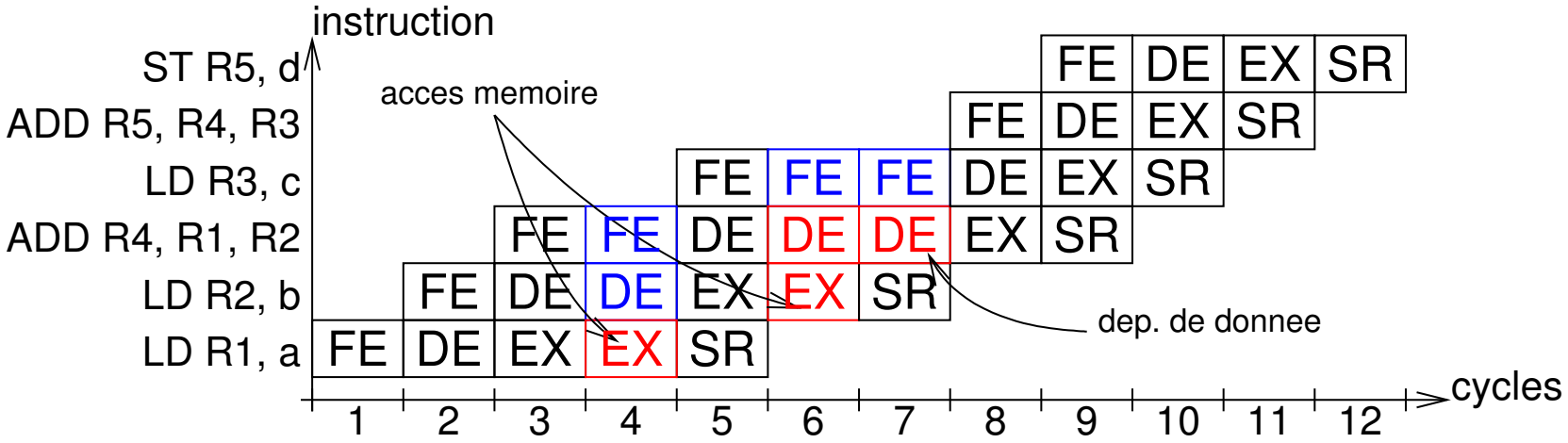
La 2de instr. ne peut commencer sa phase EX que quand la 1ère a fini SR

Parallélisme d'instruction

Dépendances entre instructions

- Dépendance de données
- Exemple

```
LD R1, a            ; R1 <- mem[a]
LD R2, b            ; R2 <- mem[b]
ADD R4, R1, R2      ; R4 <- R1+R2
LD R3, c            ; R3 <- mem[c]
ADD R5, R4, R3      ; R5 <- R4+R3
ST R5, d            ; mem[d] <- R5
```



→ 12 cycles sur cet exemple

Parallélisme d'instruction
Dépendances entre instructions

- Dépendance de nom
  - Deux instructions utilisent le même registre ou emplacement mémoire sans circulation de données

    → renommage
- Exemple

```
LD R1, a
ADD R1, R1, 1
ST R1, aa
LD R1, b
ADD R1, R1, 1
ST R1, bb
```

→

```
LD R1, a
LD R2, b
ADD R1, R1, 1
ADD R2, R2, 1
ST R1, aa
ST R2, bb
```

- Renommage
  - Par le programmeur
  - À la compilation ou à l'assemblage
  - Par le processeur (renommage de registre)

Parallélisme d'instruction
Dépendances entre instructions

- Dépendance de contrôle
  - Conditionné par une instruction de branchement
- Exemple

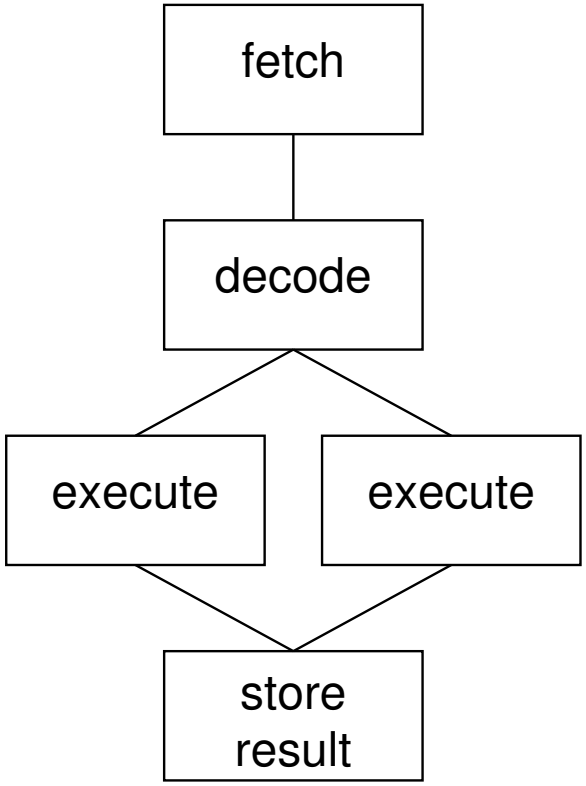
```
add:   ADD R2,R0,0
loop:  NOT R3,R1
       ADD R3,R3,1
       ADD R3,R2,R3
       BRp endl ; branchement de sortie de boucle
       LDR R3,R2,0
       ADD R3,R3,1
       STR R3,R2,0
       ADD R2,R2,1
       BR loop ; branchement vers le début de boucle
endl:  RET
```

- Solutions
  - Retarder instructions précédent le branchement
  - Prédiction de branchement → automates

Parallélisme d'instruction

Parallélisme d'instruction

- Pipelines : parallélisme partiel
- Exécution superscalaire :
  - FE capable de charger deux instructions à chaque cycle
  - DE sélectionne deux instructions dont les opérandes sont prêts
  - Deux unités EX → exécutions de deux instructions en parallèle

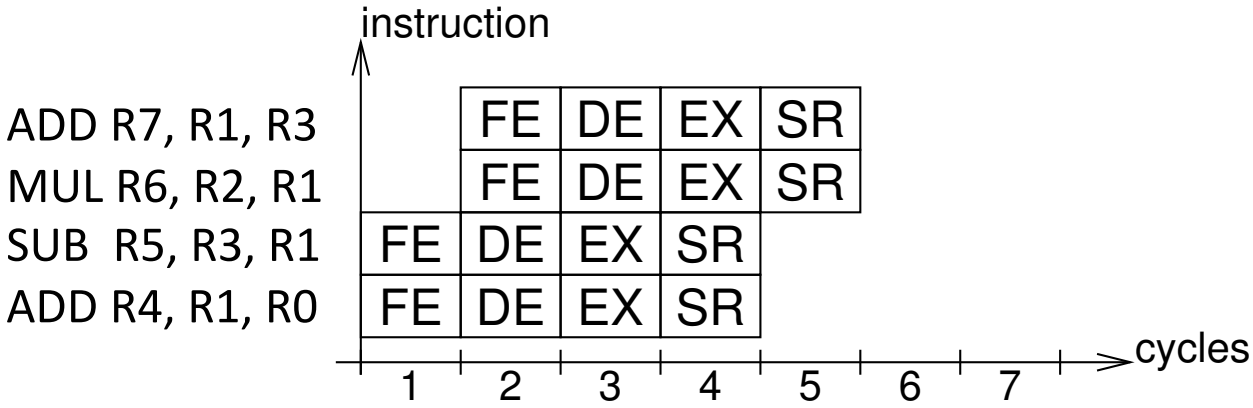


Parallélisme d'instruction

Parallélisme d'instruction

- Exécution superscalaire
- Exemple (très favorable)

```
ADD R4, R1, R0 ; R4 <- R1 + R0
SUB R5, R3, R1 ; R5 <- R3 - R1
MUL R6, R2, R1 ; R6 <- R2 * R1
ADD R7, R1, R3 ; R7 <- R1 + R3
```



→ 5 cycles (au lieu de 7)

Pour conclure

- CISC : Complex Instruction-Set Computer
  - Taille d'instruction variable
  - Instructions plus complexes à disposition
  - Ex. x86, 11/780 (VAX), 68k (Motorolla)

- RISC : Reduced Instruction-Set Computer
  - Taille d'instruction fixe, 3 opérandes, 0 mémoire (sauf lecture et écriture)
  - Cycle d'exécution simple et prévisible
  - Ex. MIPS, Alpha, PowerPC, Sparc, 88k (Motorolla), ARM, LC-3

- Et donc, CISC ou RISC ?