Architecture des processeurs généralistes haute performance

Pierre Michaud
(pmichaud@irisa.fr)

19 Mars

Exemples en technologie 90 nm

- Intel Pentium M « Dothan » 1.5 GHz
  - ~ 85 mm^2
  - 77 millions de transistors
  - ~ 20 watts
  - ~ 200 $
- Intel Itanium « Montecito » 1.6 GHz
  - ~ 600 mm^2
  - 1.72 milliards de transistors
  - ~100 watts
  - ~ 3700 $

Faire travailler les électrons

Exemple: je veux calculer exp(x) $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$

$y \leftarrow 1$
$z \leftarrow 1$
Pour n de 1 à 20
  $z \leftarrow z \times \frac{x}{n}$
  $y \leftarrow y + z$

Résultat dans y

- Pour résoudre cet exemple, on doit
  - disposer de « cases »mémoires pour x,y,z et n
  - pouvoir écrire ou lire une valeur dans ces cases
  - pouvoir effectuer des opérations
  - pouvoir tester la valeur de n pour arrêter lorsque n=20

La technologie

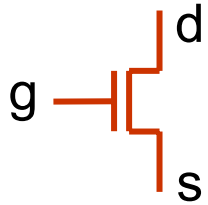
- Il faut un réservoir d'électrons ➔ cristal de silicium
  - silicium = semi-conducteur
  - On sait contrôler les électrons

- Élément de base: le transistor
  - On l'utilise comme un interrupteur
  - Technologie CMOS (Complementary Metal-Oxide Semi-conductor) ➔ 2 types de transistors
    - Transistor de type N
    - Transistor de type P

Transistors MOSFET

$1V < V_{dd} < 1.5\ V$

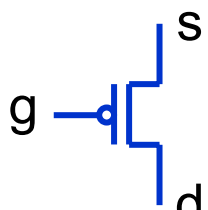
Type N



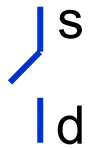
$V_{gs} = 0$ ⟷

$V_{gs} = V_{dd}$ ⟷

Type P

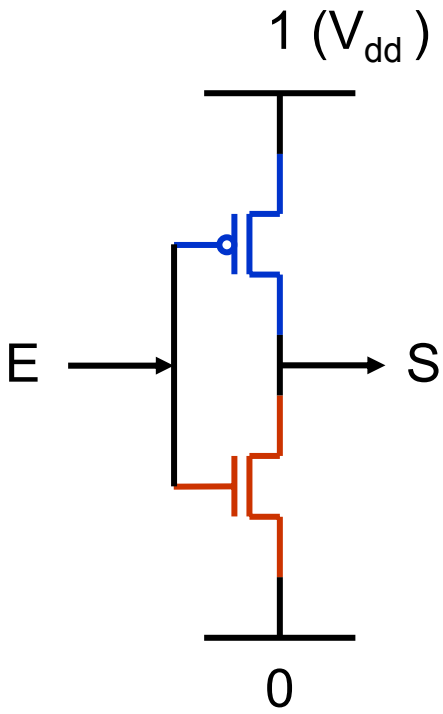


$V_{gs} = 0$ ⟷

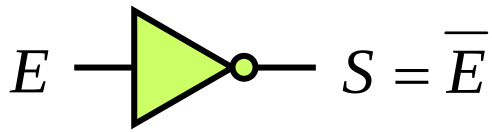


$V_{gs} = -V_{dd}$ ⟷

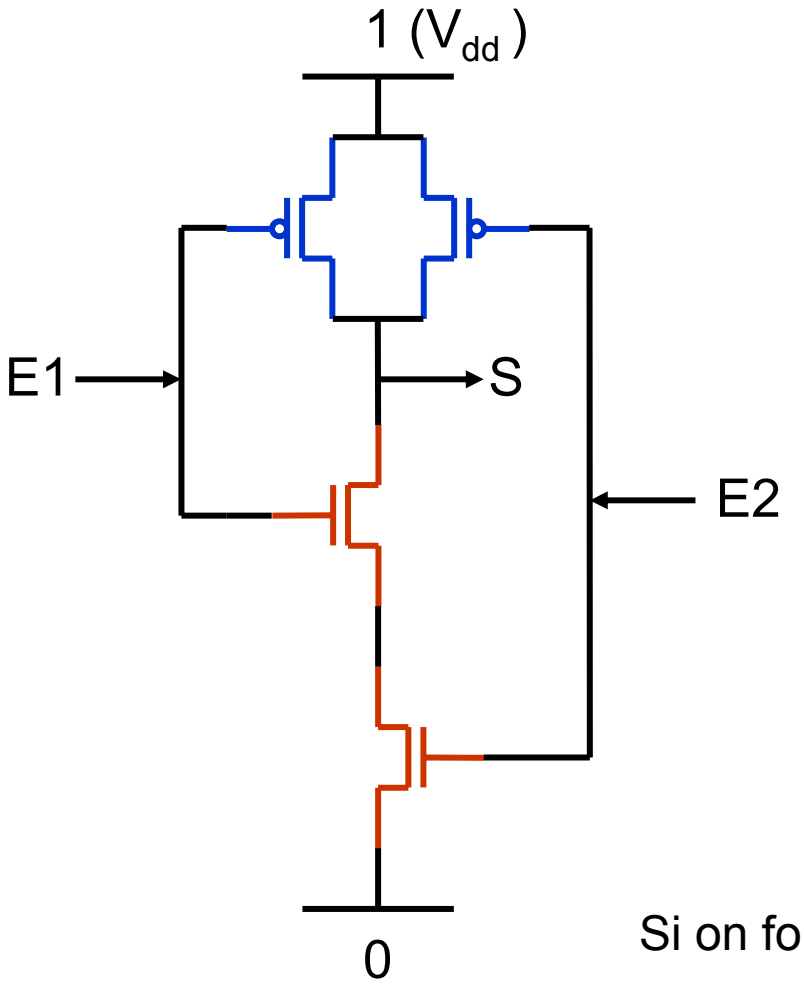
Porte NOT



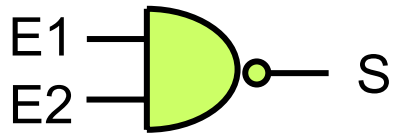
E	S
0	1
1	0



Porte NAND



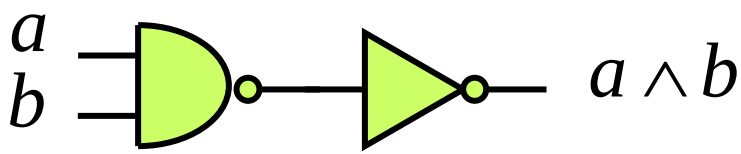
E1	E2	S
0	0	1
0	1	1
1	0	1
1	1	0



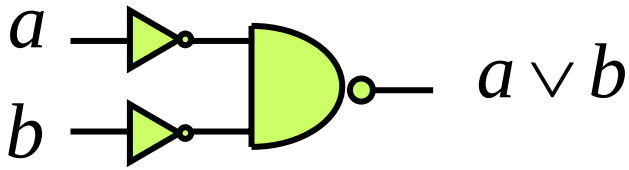
Si on force une des entrées à 1, on obtient un NOT

Portes AND,OR,XOR,…

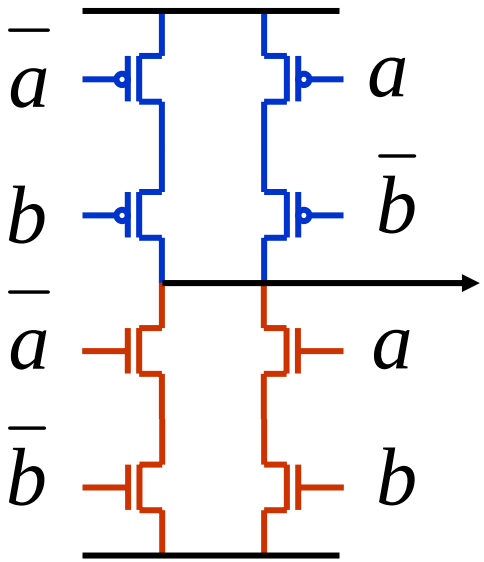
AND



OR



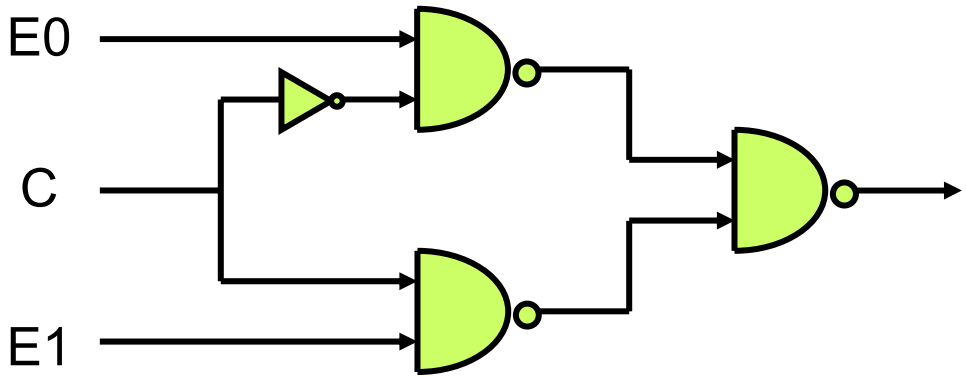
XOR



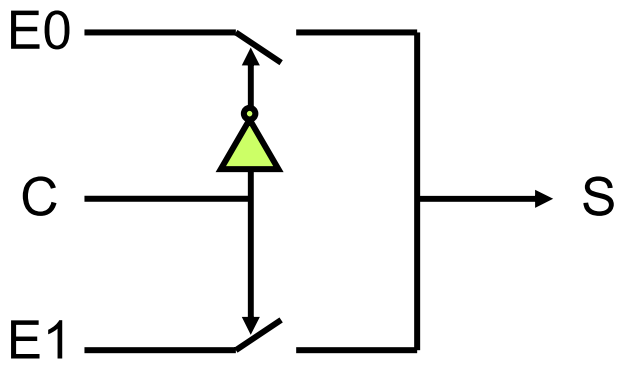
$$a \oplus b = (a \wedge \bar{b}) \vee (\bar{a} \wedge b)$$

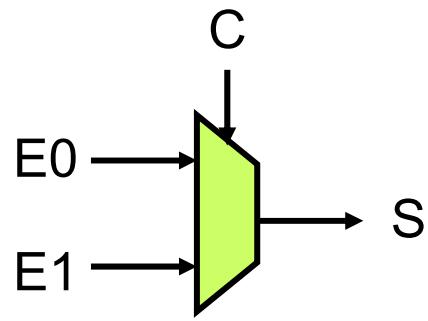
(vaut 0 si $a = b$)

Multiplexeur (MUX)



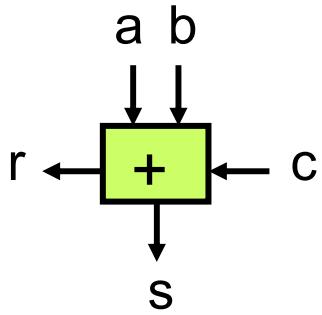
S = E0 si C=0
E1 si C=1





Additionner 2 nombres entiers

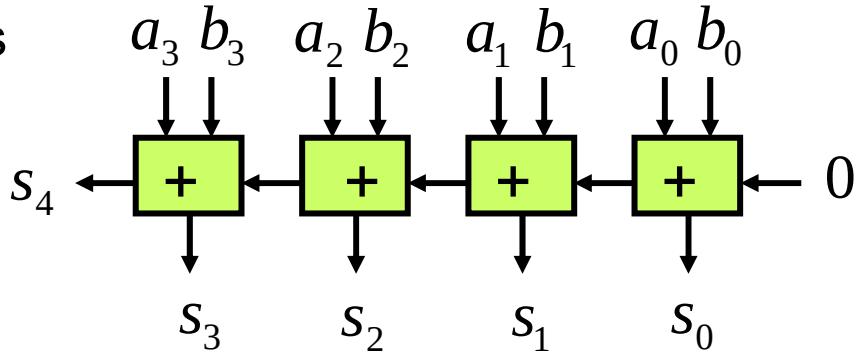
Additionneur
à 1 bit



$$s = a \oplus b \oplus c$$

$$r = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$$

Additionneur
à 4 bits



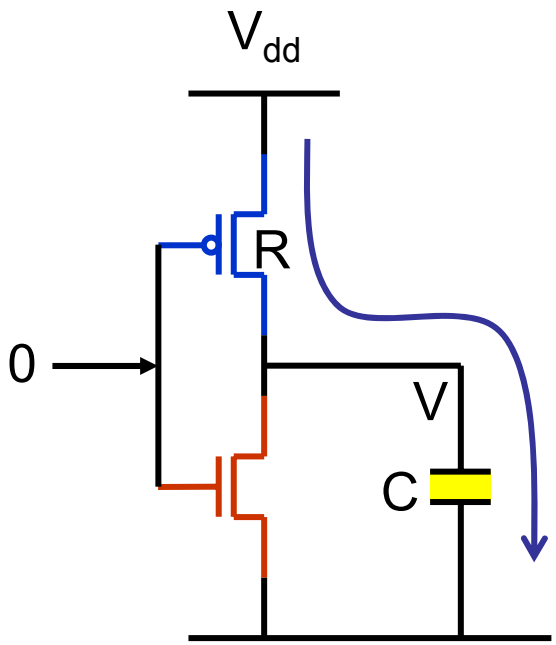
$$A = \sum_{n=0}^{3} a_n 2^n$$

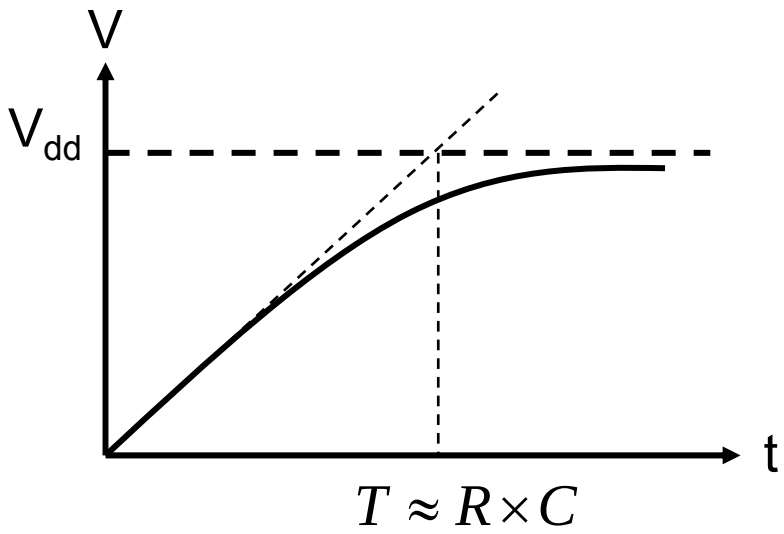
$$B = \sum_{n=0}^{3} b_n 2^n$$

$$S = \sum_{n=0}^{4} s_n 2^n = A + B$$

Temps de réponse

Ça dépend de ce qu'on met en sortie de la porte …

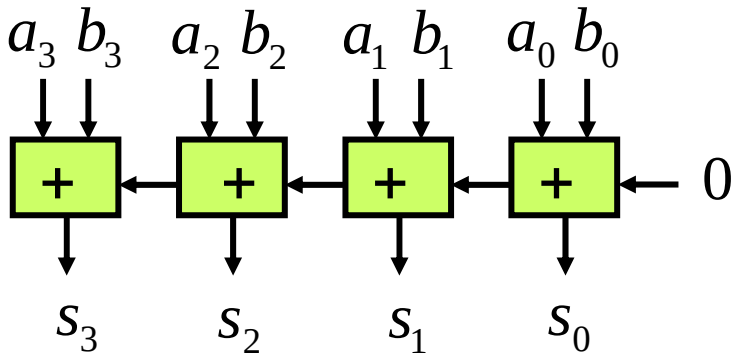




Si on connecte la porte à un grand nombre d'autres portes, la capacité C est grande, et le temps de réponse élevé

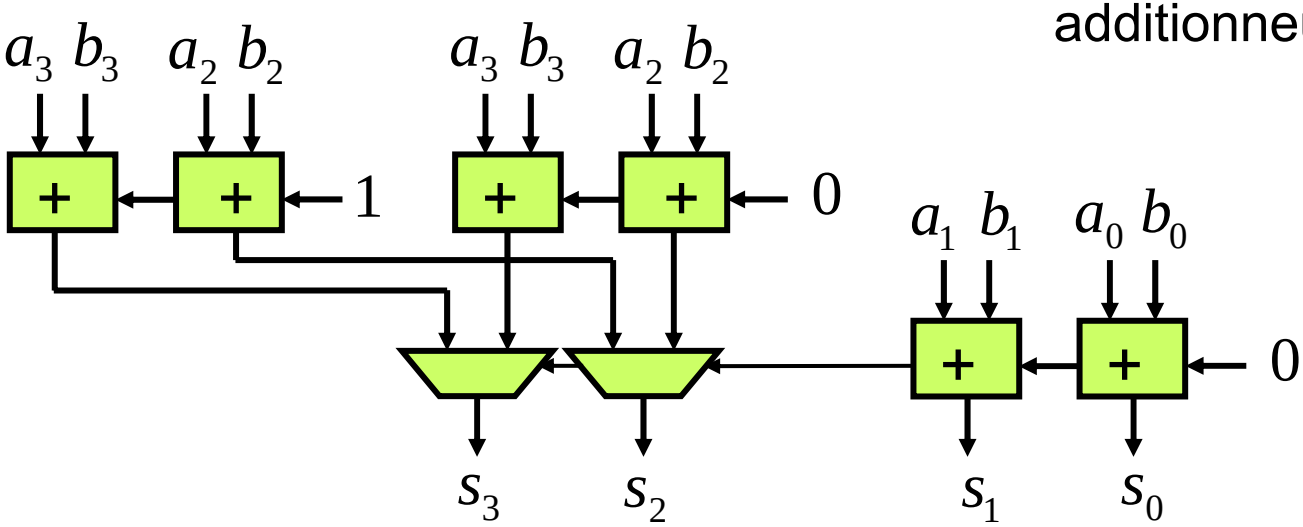
La miniaturisation des circuits diminue C ➔ circuits plus rapide

On peut parfois aller plus vite en utilisant plus de transistors …



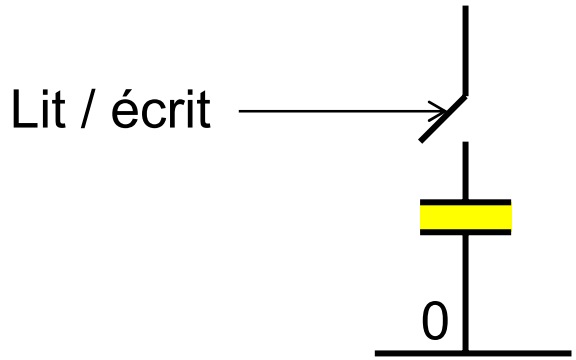
Temps de réponse pour S_3 = 4 additionneurs 1 bit

Temps pour S_3 = 2 additionneurs 1 bit et 1 MUX



Mémoire dynamique (DRAM)

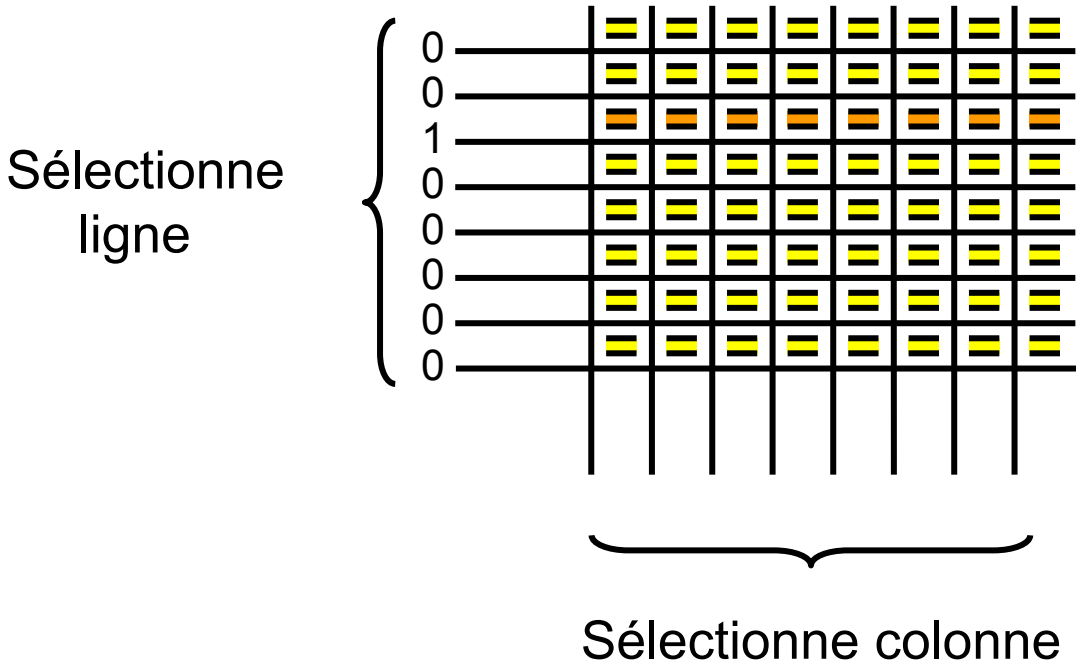
Une capacité chargée, lorsqu'elle est isolée, conserve sa charge ➔ mémorise 1 bit



- Les courants de fuite déchargent la capacité progressivement ➔ il faut « rafraîchir » le bit périodiquement

- La lecture détruit le bit ➔ Il faut réécrire le bit après chaque lecture

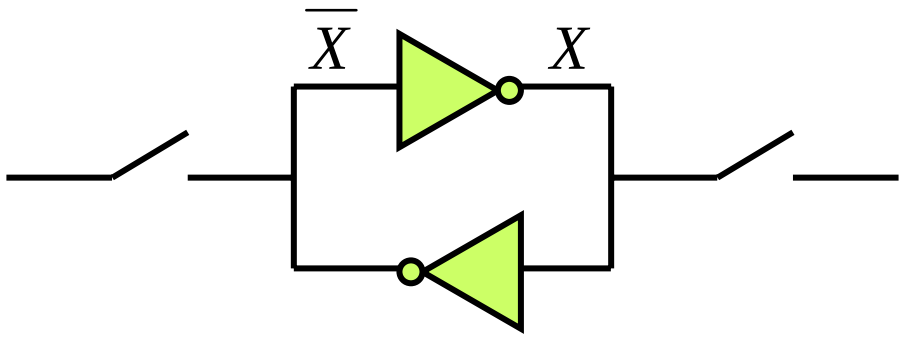
Mémoire à grand nombre de bits



- Chaque case a un numéro distinct = adresse
- L'adresse détermine la ligne et la colonne sélectionnées

Mémoire statique (SRAM)

Portes NOT « tête-bêche » mémorisent 1 bit

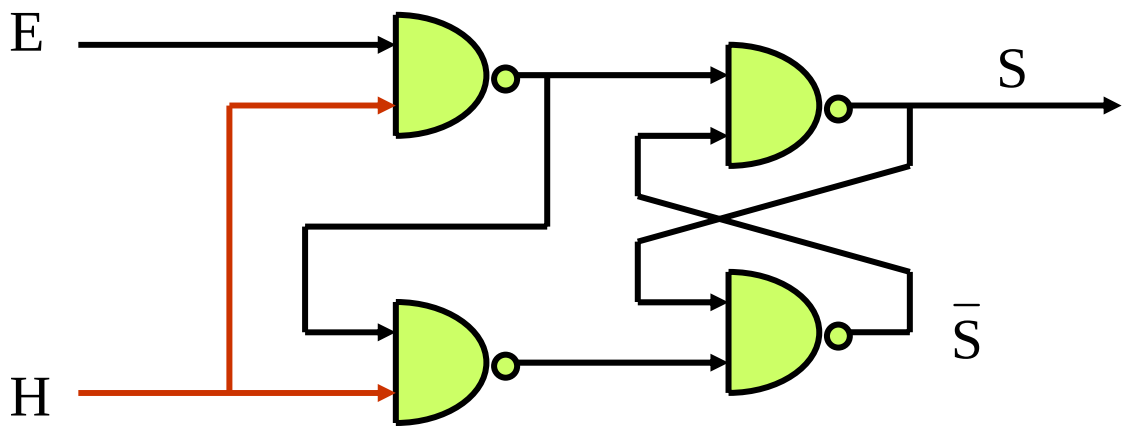


On sait faire des mémoires à plusieurs ports de lecture/écriture

➔ permet d'accéder indépendamment plusieurs cases mémoire en même temps

L'aire augmente avec le nombre de ports

Verrou (« latch »)



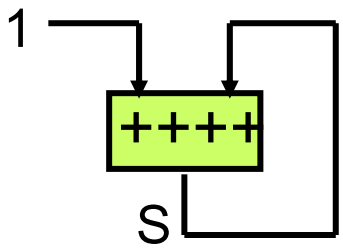
$H = 1$ ➡ $S = E$ Le verrou est « passant »

$H = 0$ ➡ S figé à la valeur qu'il avait juste avant que H passe de 1 à 0

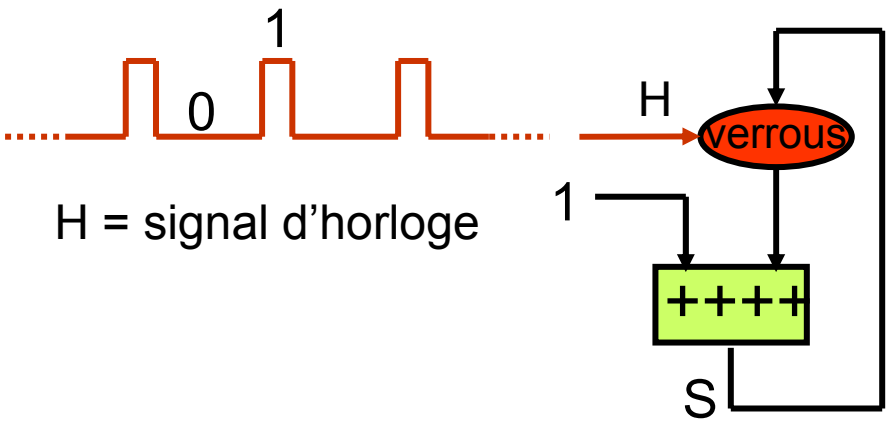
La sortie est maintenant isolée de l'entrée

Comment faire un compteur ?

S= 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,1,2,3,etc…



Problème: les bits de poids faible arrivent avant ceux de poids fort

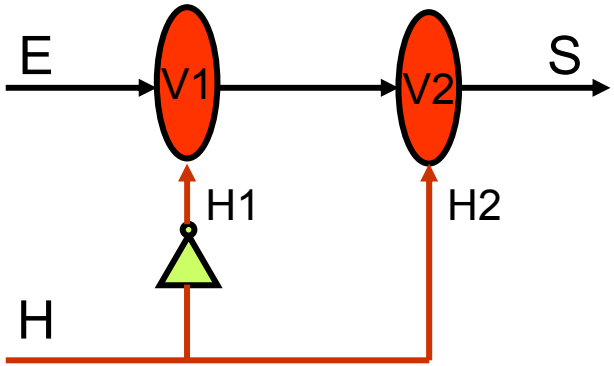


H = signal d'horloge

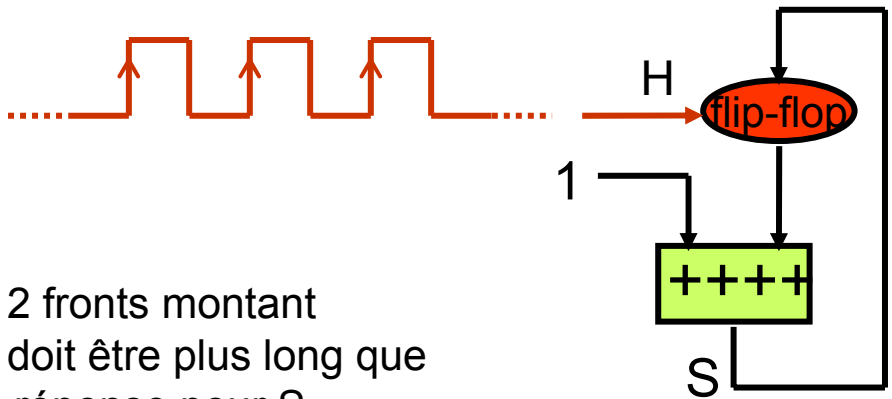
H=0: plus long que le temps de réponse pour S_3

H=1: plus court que le temps de réponse pour S_0

« Flip-flop » = 2 verrous



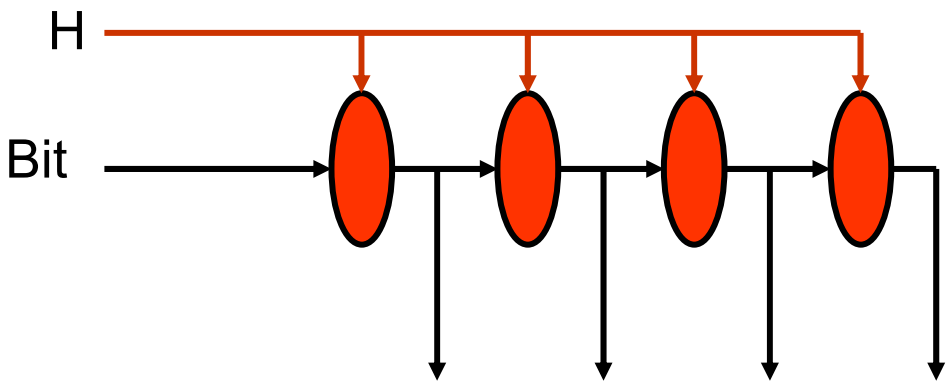
S prend la valeur de E lorsque H passe de 0 à 1 (= front montant)



Le temps entre 2 fronts montant consécutifs doit être plus long que le temps de réponse pour S_3

Registre à décalage

Exemple: registre à décalage 4 bits

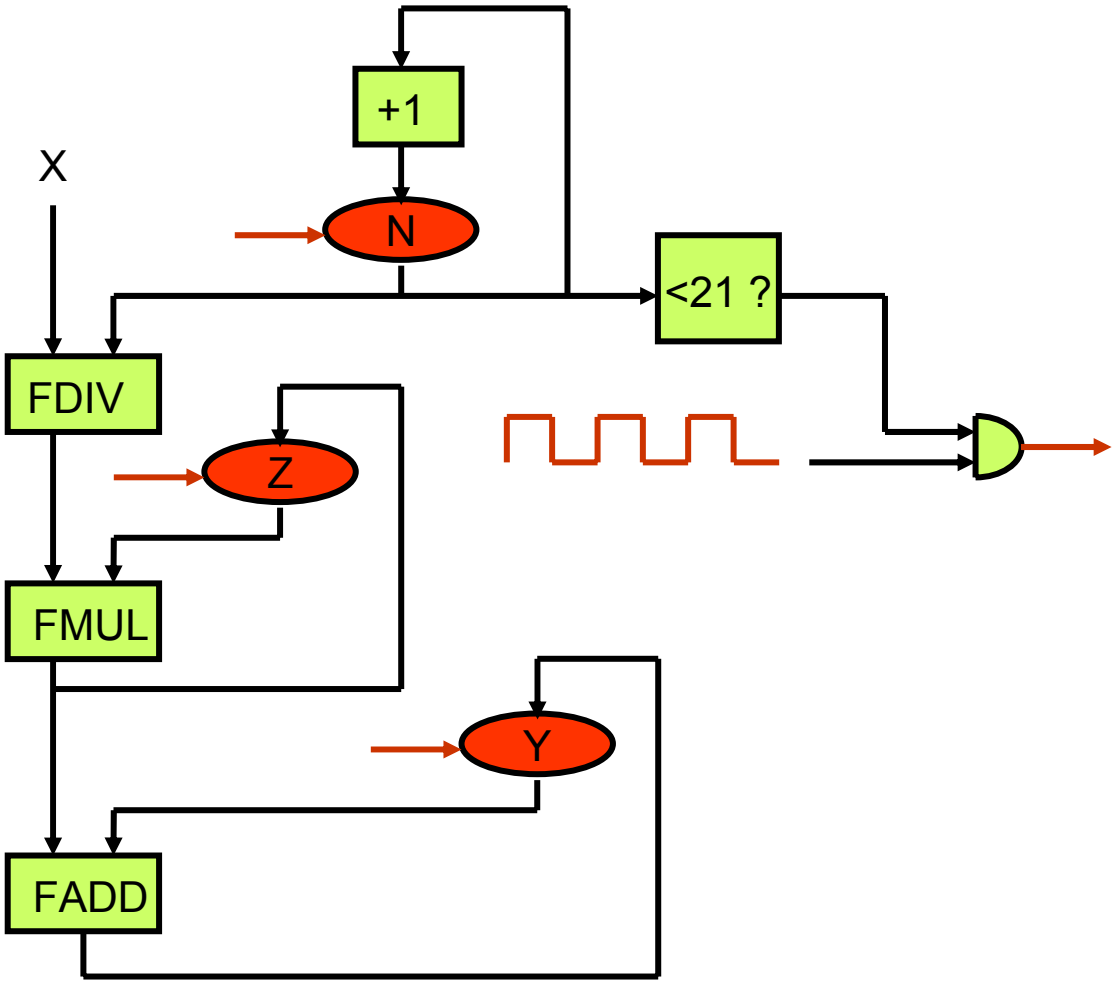


Pour insérer un nouveau bit, envoyer un front montant sur H

Le registre mémorise les 4 derniers bits insérés

Exp(x)

On pourrait faire un circuit spécialisé …



Processeur

- On crée une architecture et un jeu d'instructions

- On construit un circuit qui permet d'exécuter des programmes écrits dans ce jeu d'instructions
  - Programme = suite d'instructions stockée en mémoire
  - Après avoir exécuté l'instruction (de longueur X) à l'adresse A, on exécute l'instruction à l'adresse A+X, et ainsi de suite
    - Sauf quand on exécute une instruction de contrôle
  - A = compteur de programme (PC: program counter)

- Lorsqu'on veut résoudre un autre problème, on garde le même matériel mais on change de programme

Architecture

- L'architecture est la machine abstraite perçue par le programmeur
  - La microarchitecture est la machine physique, c'est une mise en œuvre particulière d'une architecture donnée
- On définit des registres, leur nombre et leur type
  - Registres = petite mémoire dont l'accès est direct
  - Exemple: 32 registres « entiers », 32 registres « flottants »
- On définit un espace mémoire virtuel
  - Exemple: architecture 64 bits ➔ 2^{64} octets de mémoire virtuelle
  - C'est la machine abstraite qui a 2^{64} octets, pas la machine physique
  - Mémoire physique principale (DRAM) beaucoup plus petite que 2^{64}
  - Mécanisme de pagination ➔ pages mémoire en excès stockées sur disque
- On définit un jeu d'instructions
- (Et un certains nombre d'autres choses …)

Jeu d'instructions

- Différents types de jeux d'instructions

- RISC (Reduced Instruction-Set Computer)
  - Exemple: MIPS, Alpha, PowerPC, Sparc
  - Taille d'instruction fixe
    - exemple: 4 octets par instruction ➔ PC=PC+4
  - Petit nombre d'opérations simples
    - Lit 2 registres maxi, écrit 1 registre maxi
  - Modes d'adressage mémoire simples

- CISC (Complex Instruction-Set Computer)
  - Exemple: Intel X86
  - Taille d'instruction variable (codée dans l'en-tête de l'instruction)
  - Grand nombre d'opérations, dont certaines complexes
  - Plus de modes d'adressage mémoire, dont certains complexes

Exemples d'instructions

- r1 ← CONST
  - Écrit CONST dans registre r1
  - CONST = constante codée dans l'instruction

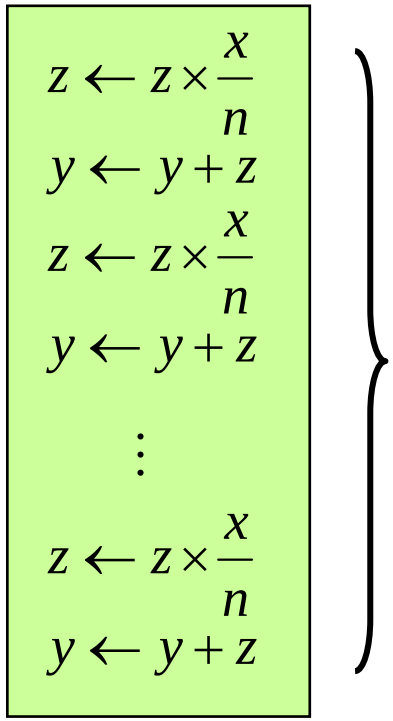
- r3 ← r1 OP r2
  - Lit registres r1 et r2, exécute opération OP, écrit résultat dans r3
  - Opérations sur les entiers: add,sub,mul,div,and,or,xor,shift, etc…
  - Opérations sur les flottants: fadd,fsub,fmul,fdiv,sqrt, …

- r2 ← LOAD r1
  - Utilise la valeur contenue dans registre r1 comme adresse mémoire
  - Lit la valeur stockée en mémoire à cette adresse
  - Copie la valeur dans registre r2

- STORE r1,r2
  - Valeur dans r1 copiée dans la case mémoire dont l'adresse est dans r2

Instructions de contrôle

- Savoir faire des opérations n'est pas suffisant



20 fois

Et si on veut le faire 10^9 fois ?
➔ le programme prend une place énorme en mémoire

- On veut pouvoir manipuler le compteur de programme (PC) pour faire des tests, des boucles, des procédures…

Exemples d'instructions de contrôle

- BNZ r1,DEP
  - Branchement conditionnel
  - Saute à l'adresse PC+DEP si la valeur dans r1 est non nulle
  - DEP est une constante codée dans l'instruction
    - peut être négative (saut vers l'arrière)

- JUMP r1
  - Saute à l'adresse contenue dans r1

- r2 ← CALL r1
  - Saute à l'adresse contenue dans r1 et écrit PC+4 dans r2
  - Utilisé pour connaître le point d'appel lorsqu'on fait un retour de procédure
  - on peut aussi utiliser un registre spécialisé implicite au lieu de r2

Modes d'adressage

- r2 ← LOAD r1,DEP
  - Adresse = r1+DEP
  - DEP = constante codée dans l'instruction
  - Pratique pour accéder aux champs d'une structure

- r3 ← LOAD r1,r2
  - Adresse = r1+r2
  - Pratique pour accéder à un tableau

- r2 ← LOAD (r1)
  - 2 lectures mémoire (CISC !)
  - Lit la valeur 64 bits stockée en mémoire à l'adresse contenue dans r1
  - Utilise cette valeur comme adresse finale
  - Pratique quand on programme avec des pointeurs
  - RISC: on doit utiliser 2 LOAD

Exp(x)

x dans r33

y dans r34

z dans r35

n dans r1

```
        r34 ← 1
        r35 ← 1
         r1 ← 1
Boucle: r36 ← r33  FDIV  r1
        r34 ← r34  FMUL  r36
        r35 ← r35  FADD  r34
         r1 ← r1   ADD   1
         r2 ← r1   SUB   21
        BNZ  r2,–5
```

Exécuter une instruction: différentes étapes

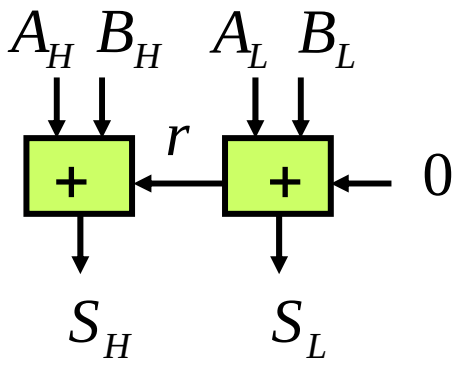
- PC stocké dans un flip-flop
- Lire l'instruction stockée en mémoire à l'adresse PC
- Décoder l'instruction
  - Le format d'instruction est compact, pour minimiser la taille des programme
  - Extraire du code de l'instruction les différentes informations (type d'opération , registres accédés)
- Lire les registres
- Exécuter l'opération
- Écrire le résultat dans les registres
- Envoyer un « coup » d'horloge pour passer au PC suivant

Processeur haute performance

- On sait « faire travailler les électrons »

- On veut que ce travail soit effectué le plus rapidement possible

La technique du pipeline

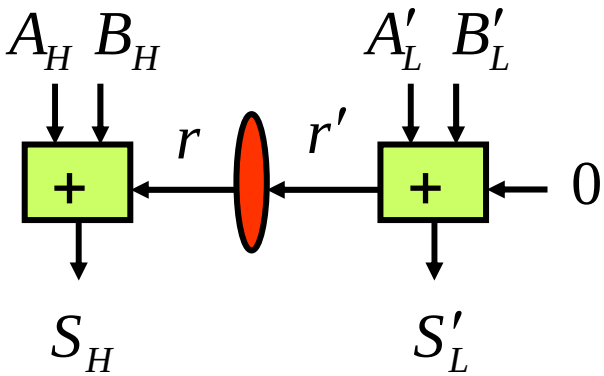


T = temps de réponse d'un additionneur 32 bits

Durée d'une addition 64 bits = 2T

Pour effectuer 10 additions 64 bits avec le même additionneur ➔ 20 T

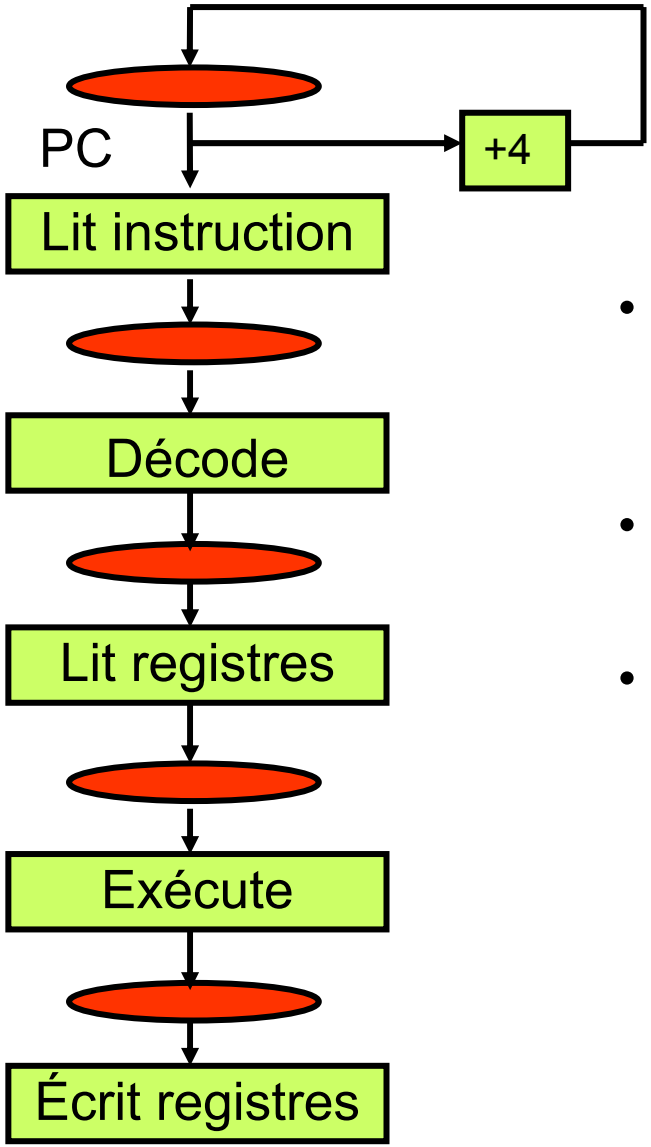
Additionneur 32 bits inutilisé 50% du temps



Une addition prend légèrement plus de temps

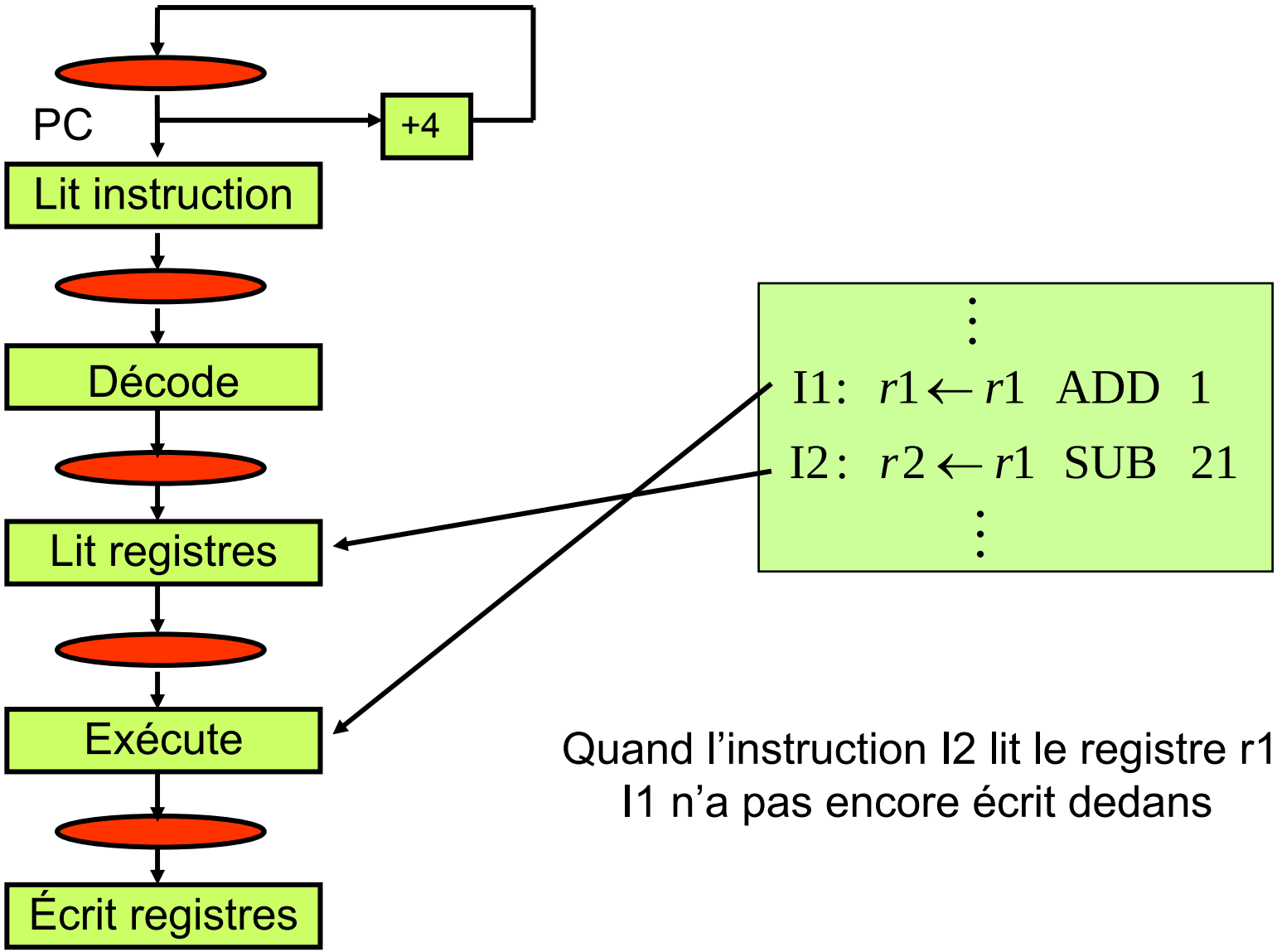
Mais le débit d'exécution est double ➔ 11 T

Le pipeline d'instructions



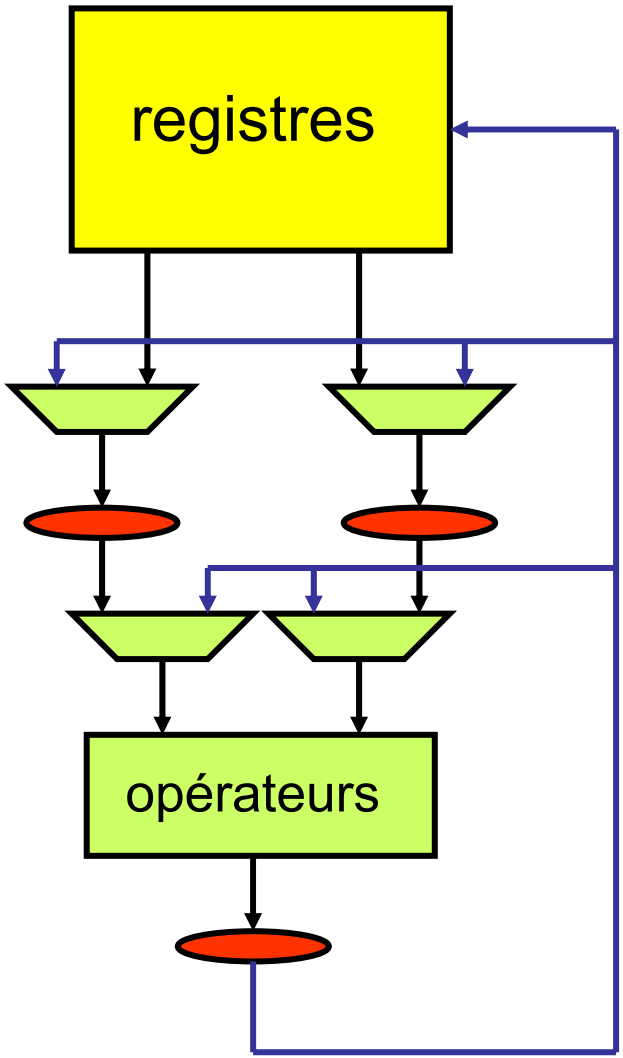
- À chaque cycle d'horloge, une instruction rentre dans le pipeline
- Registres ➔ SRAM à plusieurs ports
- Attention, en réalité c'est plus compliqué
  - Dépendances entre instructions ?
  - Accès mémoire ?
  - instructions de contrôle ?

Dépendances entre instructions



Quand l'instruction I2 lit le registre r1,
I1 n'a pas encore écrit dedans

Réseau de *bypass*



Les MUX sont commandés par des comparaisons sur les numéros de registre

Latence des accès mémoire

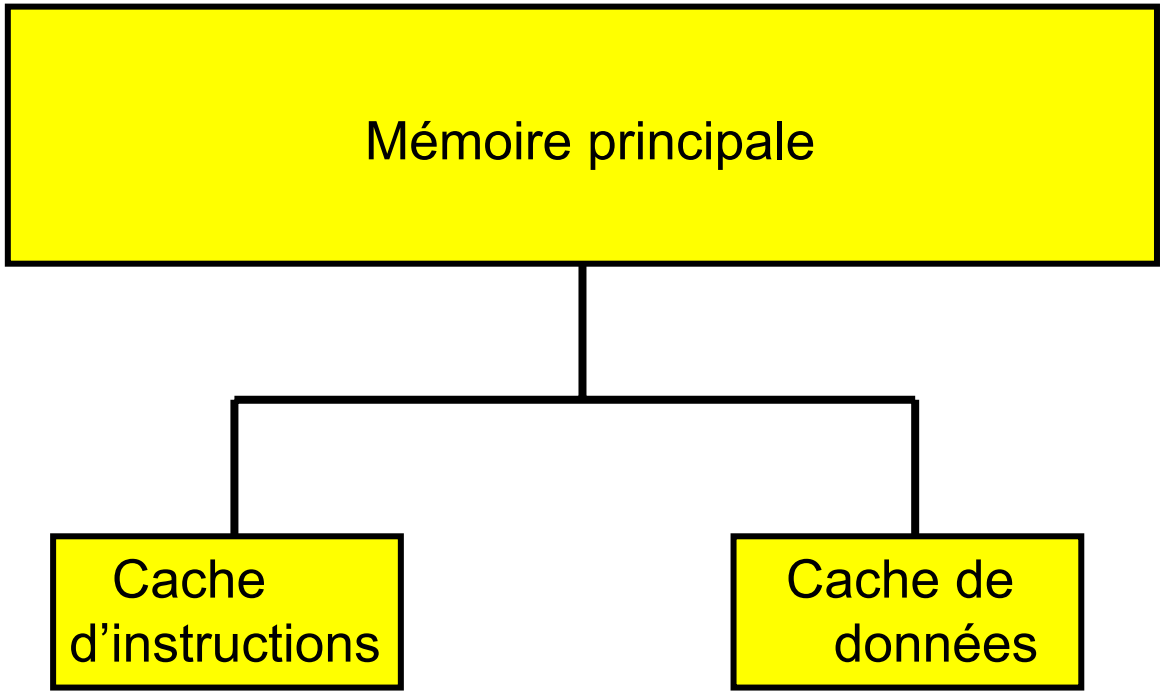
- Une petite mémoire est accédée plus rapidement qu'une grande mémoire
  - temps d'accès aux registres ➔ < 1 ns

- Station Dell Precision 360
  - Processeur Intel Pentium 4
  - Fréquence horloge 3 Ghz ➔ 0.33 ns de temps de cycle
  - 1 Go de mémoire physique
  - Latence mémoire (accès aléatoire) ➔ ~ 100 ns

$$\frac{100}{0.33} = 300 \text{ cycles processeur !}$$

Les caches mémoire

- Principe de localité temporelle
  - La plupart des programmes ont tendance à réaccéder des instructions et des données qu'ils ont accédées récemment
    - Exemple: une boucle
- Cache = petite mémoire située sur la puce
  - 1 cache pour les instructions
  - 1 cache pour les données
  - ➔ permet d'accéder une instruction et une donnée dans le même cycle
- On met dans les caches les données et instructions accédées récemment
- Tout se fait automatiquement
  - Transparent pour le programmeur et le compilateur



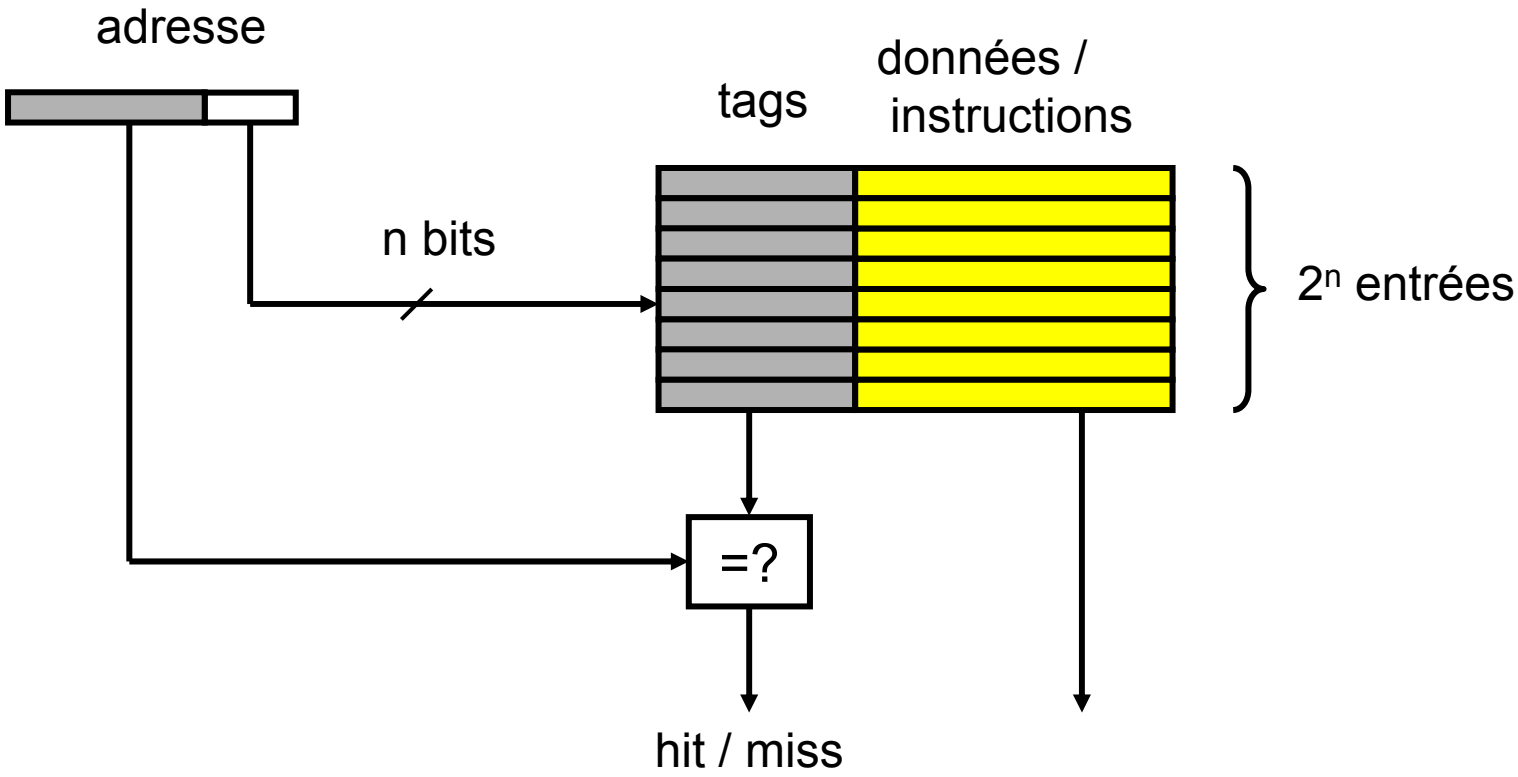
1. On regarde dans le cache

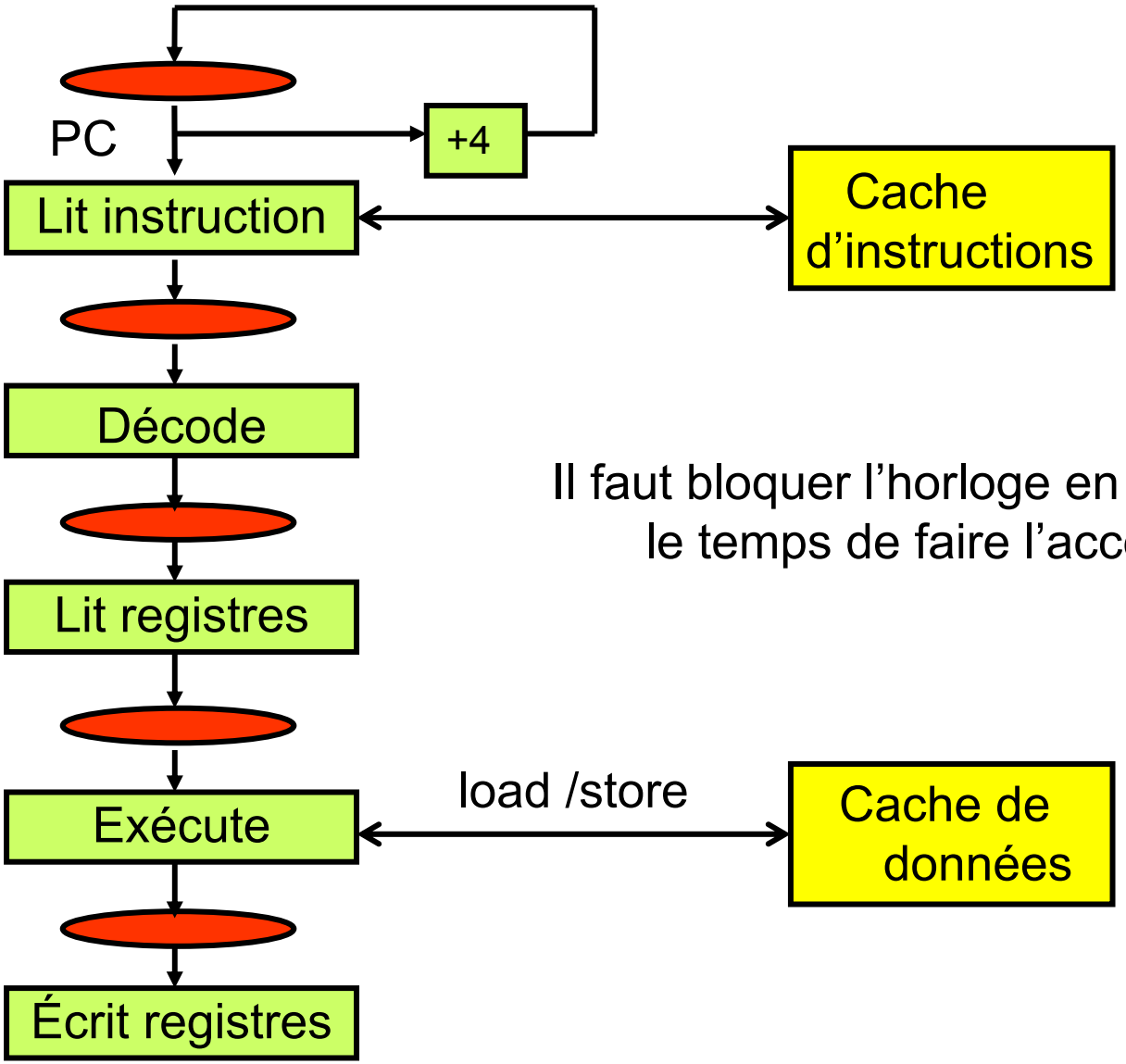
2. Si la donnée (ou l'instruction) s'y trouve, c'est un hit

3. Sinon, c'est un miss (= *défaut* de cache)
   - La donnée (ou l'instruction) est lue en mémoire
   - Elle est copiée dans le cache

Cache direct (*direct-mapped)*

- Sélectionner l'entrée avec les bits de poids faible de l'adresse
- Comparer les bits de poids fort avec le tag stocké dans l'entrée





Il faut bloquer l'horloge en cas de miss,
le temps de faire l'accès mémoire

Lignes de cache

- Principe de localité spatiale
  - La plupart des programmes ont tendance à accéder dans des temps rapprochés des données ou instructions situées à des adresses proches

- Au lieu de stocker dans une entrée du cache une seule donnée ou une seule instruction, on stocke une ligne
  - Ligne = nombre fixe d'octets consécutifs en mémoire
  - Taille typique: 64 octets

Exemple:

```
int x[N];

for (i=0; i<N; i++)
    x[i] = 0;
```

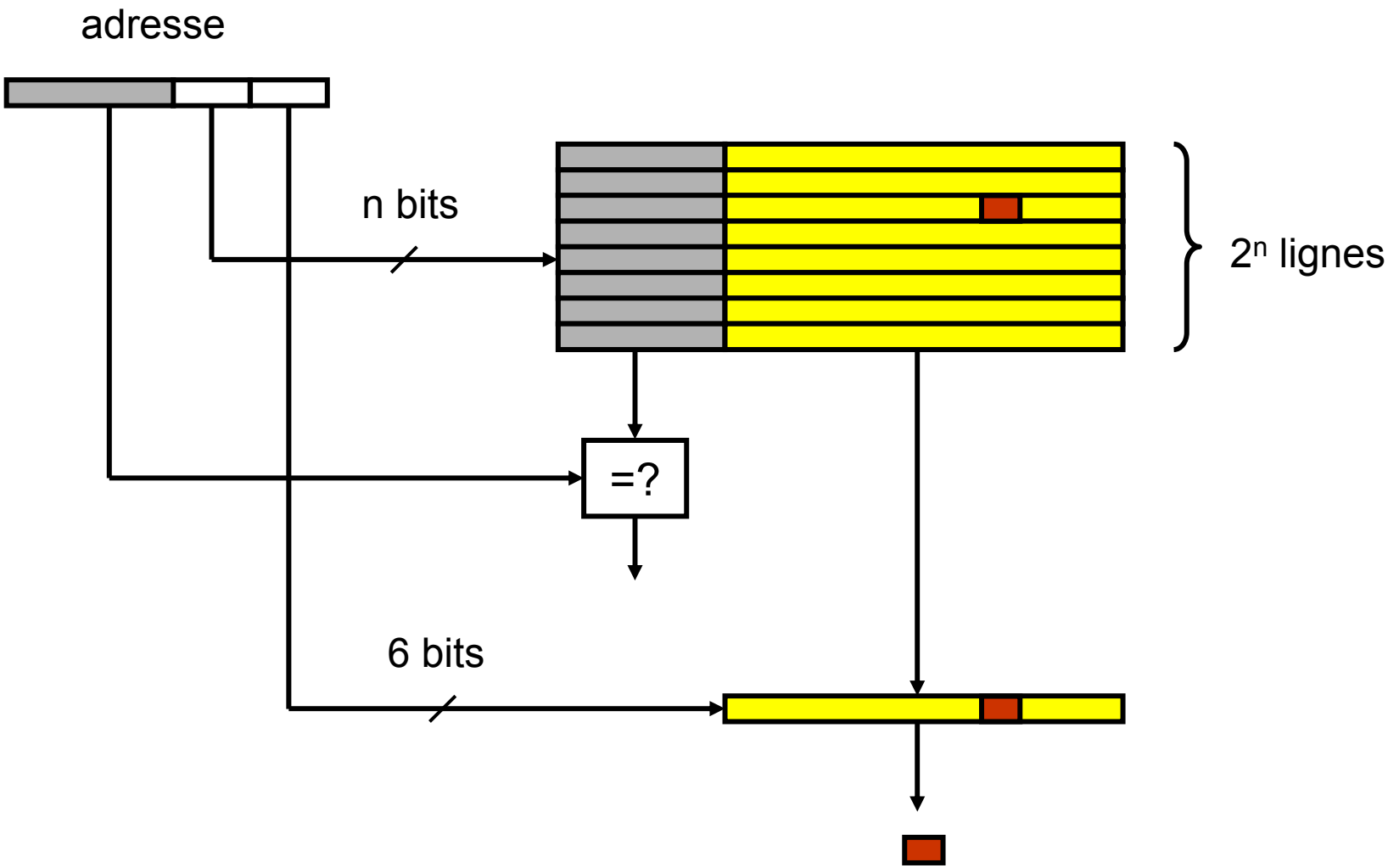
1 *int* ➔ 4 octets

1 ligne de 64 octets ➔ 16 *int*

1 miss toutes les 16 itérations

Exemple: ligne de 64 octets

Les 6 bits de poids faible de l'adresse sélectionnent la donnée à l'intérieur de la ligne



Cache associatif par ensemble

- Plusieurs type de miss
  - Miss de démarrage: la 1ère fois qu'on accède à une ligne
  - Miss de capacité: le cache n'est pas assez grand pour contenir toutes les données ou instructions du programme
  - Miss de conflit: des lignes se disputent la même entrée du cache alors que d'autres entrées sont inutilisées

- On peut réduire les miss de conflit avec un cache associatif par ensemble
  - Chaque entrée du cache contient plusieurs lignes
  - À taille égale, moins d'entrées qu'un cache direct

- Exemple: cache 4-way
  - chaque entrée contient 4 lignes et 4 tags
  - On fait 4 comparaisons de tags et on sélectionne la ligne qui a le bon tag
    - Si aucun tag parmi les 4 ne correspond, c'est un miss

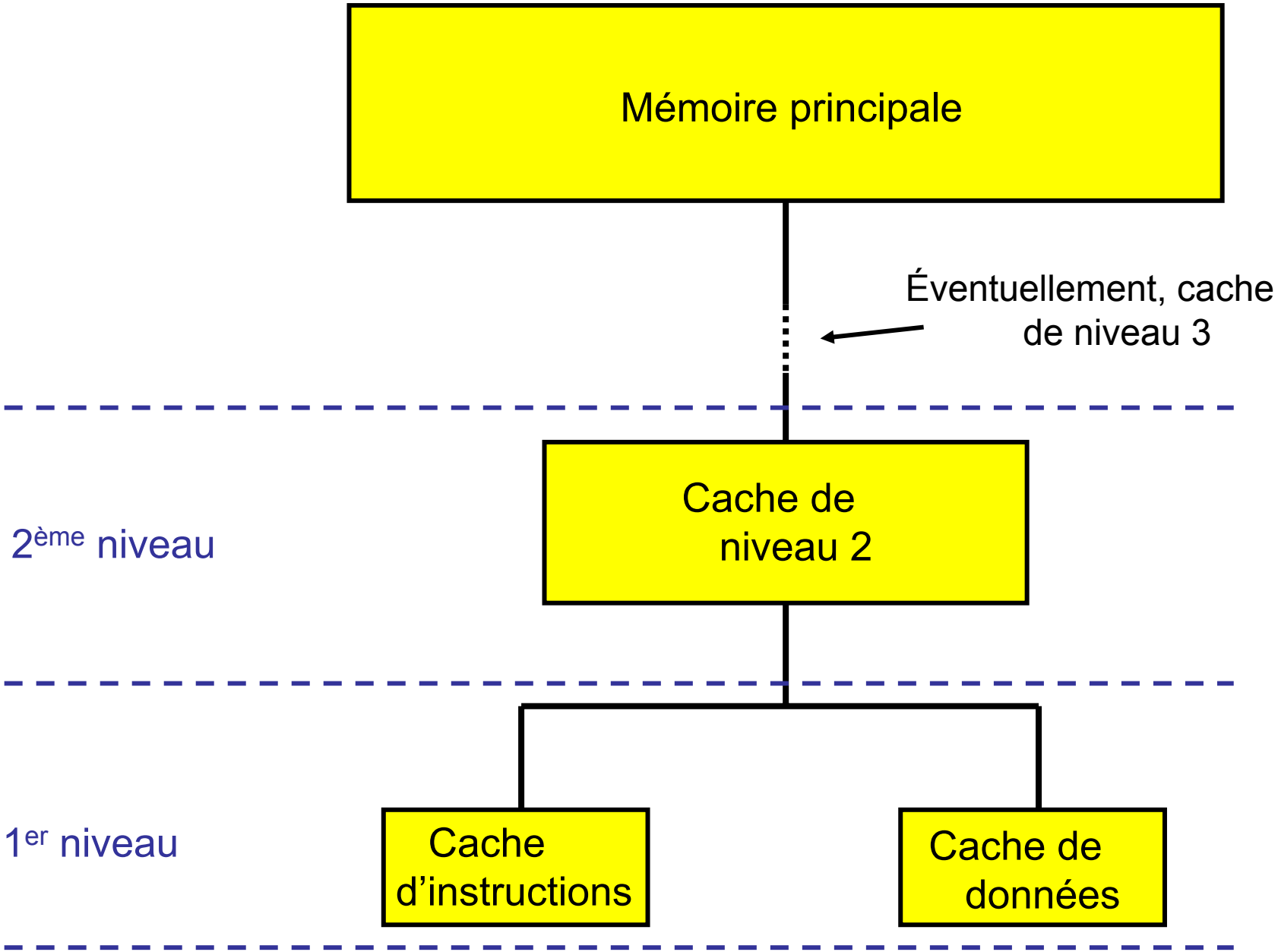
Hiérarchie mémoire

- La latence d'un cache dépend de sa taille et de sa structure
  - Un petit cache est plus rapide qu'un grand cache
  - cache direct plus rapide que cache associatif par ensemble

- 1er niveau de cache
  - Typique: 32 Ko, 4-way
    - (les tags ne sont pas comptés dans les 32 Ko)
  - Latence: < 1 ns

- 2ème niveau de cache
  - Typique: 512 Ko, 8-way
  - Latence: < 5 ns

- (3ème niveau de cache)
  - Itanium Montecito: 12 Mo, 12-way, latence < 10 ns


Mémoire principale
Éventuellement, cache de niveau 3
2ème niveau
Cache de niveau 2
1er niveau
Cache d'instructions
Cache de données

Bande passante

- Bande passante = nombre moyen d'octets par unité de temps que peut délivrer une mémoire, un cache, ou un bus

- Important pour la performance. Exemple:
  - Bus mémoire de 64 bits de largeur, cycle bus = 8 cycles processeur
  - ➔ bande passante = 1 octet par cycle processeur
  - Ligne de cache 64 octets ➔ 64 cycles processeur
    - Latence de 300+3 cycles pour obtenir la donnée de 4 octets demandée
    - + 60 cycles pour le reste de la ligne
  - Le pipeline d'instructions peut redémarrer sans attendre que le reste de la ligne soit stocké dans le cache
  - Mais si un 2ème défaut de cache se produit moins de 60 cycles après le redémarrage du pipeline, il faudra attendre

- La bande passante doit être suffisante pour supporter des rafales de défauts de cache

Écritures mémoire (1)

- Cache à écriture transmise (*write-through*)
  - Si la donnée est dans le cache, on la met à jour
  - On transmet l'écriture au niveau suivant de la hiérarchie mémoire

- Cache à écriture différée (*write-back*)
  - Chaque ligne de cache comporte un bit *modifiée*
    - Bit mis à 0 lorsque la ligne est insérée dans le cache
    - Bit mis à 1 lorsqu'on écrit dans la ligne
  - Si la donnée est absente du cache, on va chercher la ligne manquante
  - On met à jour la donnée seulement dans le cache
  - Lorsqu'une ligne est évincée, et si la ligne a été modifiée, on écrit son contenu dans le niveau suivant de la hiérarchie mémoire

- Caches de niveau 2 et 3 sont généralement à écriture différée

Écriture mémoire (2)

- Pour plus de performance, on permet au pipeline d'instructions de continuer sans attendre que l'écriture soit terminée

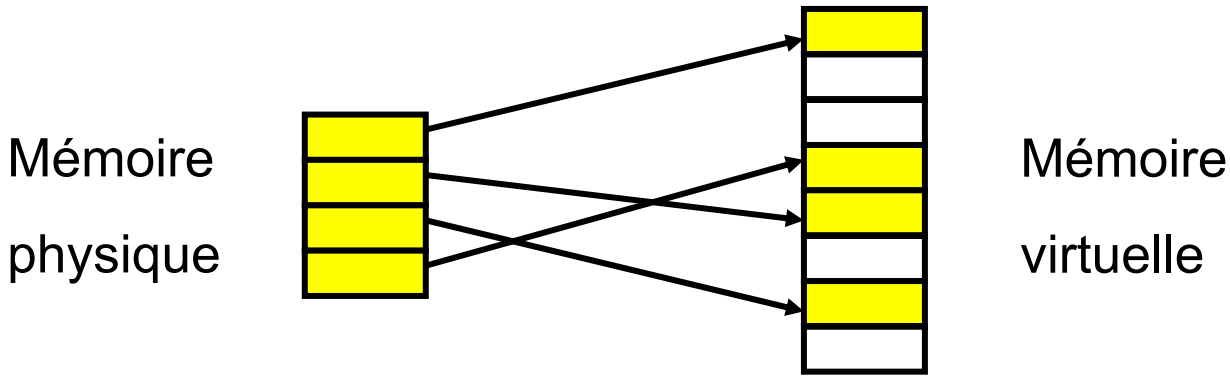
- Les écritures sont mises en attente dans un tampon d'écriture
  - Petite mémoire sur la puce
  - Hiérarchie mémoire ➔ plusieurs tampons
  - Cache à écriture différée: on met dans le tampon les lignes évincées

- Les écritures sont effectuées lorsque la bande passante est disponible ou lorsque le tampon est plein
  - Si le tampon n'est pas plein, priorité aux lectures

- Sur un défaut de cache, il faut regarder dans le tampon
  - Si la donnée demandée s'y trouve, c'est le tampon qui répond

Mémoire virtuelle / mémoire physique

- Le programme utilise des adresses virtuelles
- Il faut traduire les adresses virtuelles en adresses physiques
- L'espace mémoire est divisé en pages
  - Exemple: page de 4 Ko



- Il faut traduire le numéro de page virtuelle en numéro de page physique ➔ table des pages stockée en mémoire
- Si la page demandée n'est pas en mémoire physique, il faut récupérer la page sur disque et mettre à jour la table des pages
  - ➔ « défaut de page »

Traduction d'adresse avec un TLB

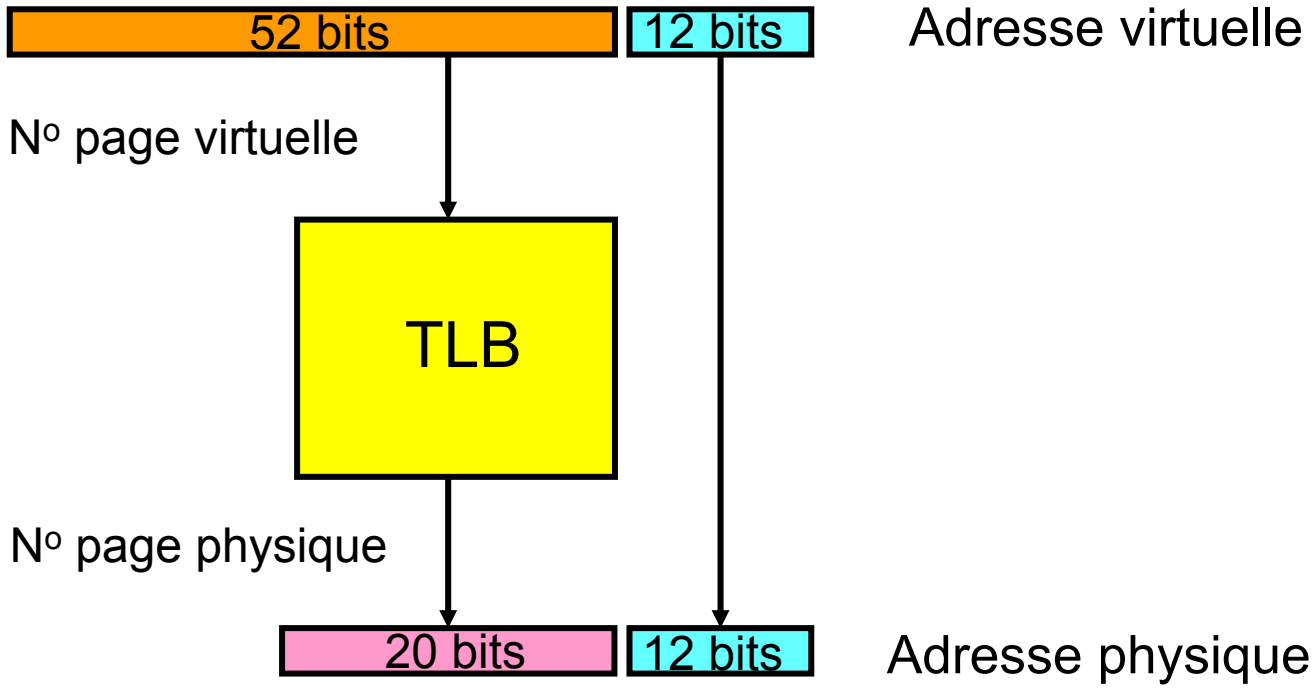
- Comme la table des pages se trouve en mémoire, chaque accès mémoire du programme nécessite au moins 2 accès en mémoire physique ☹
  - Voire plus …

- TLB (Translation Lookaside Buffer) ➔ petite mémoire sur la puce contenant les correspondances page virtuelle/page physique pour les pages accédées récemment
  - = cache de traduction d'adresse
  - En cas de miss TLB ➔ accès table des pages en mémoire

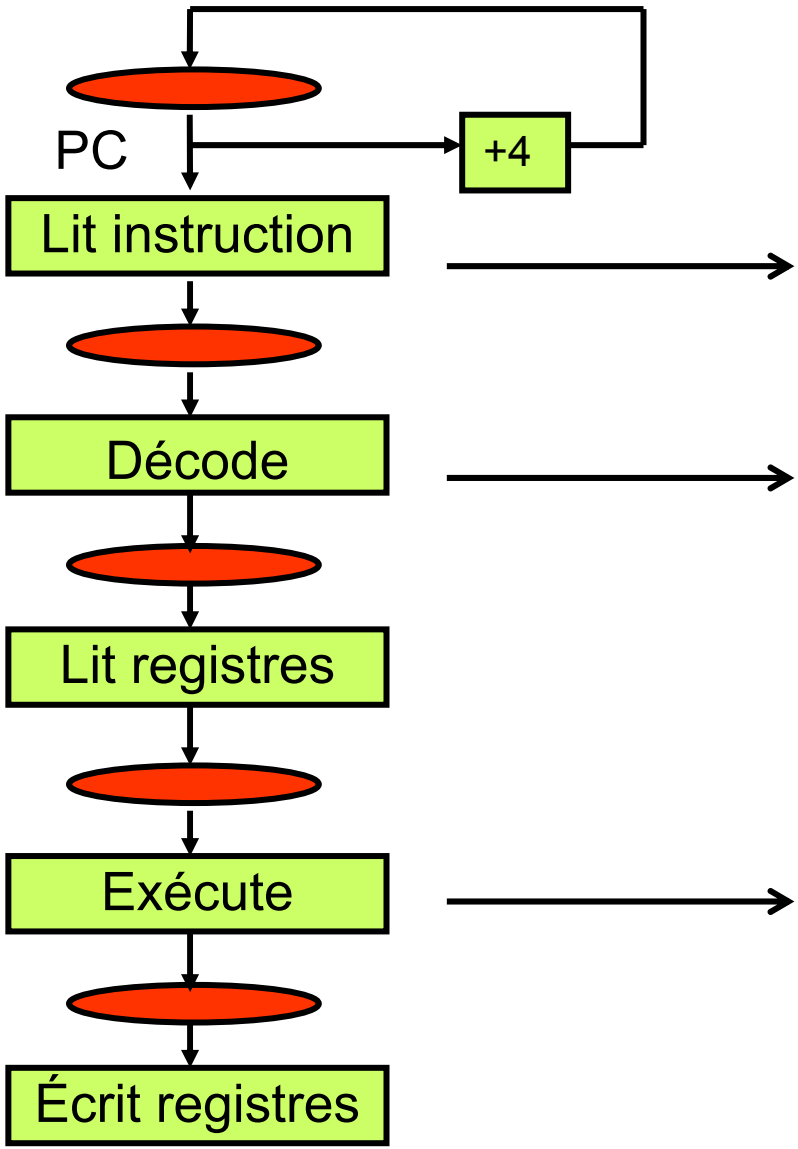
- Exemple: Pentium 4, pages 4 Ko
  - TLB d'instructions 128 pages, 4-way
  - TLB de données 64 pages, associatif (=64-way)

Exemple

- Pages de 4 Ko
- Adresses virtuelles sur 64 bits
- Adresses physique sur 32 bits



Instructions de contrôle



C'est une instruction de contrôle, mais on ne le sait pas encore

On détecte que c'est une instruction de contrôle, mais on ne connaît toujours pas son résultat

On connaît enfin la bonne valeur de PC

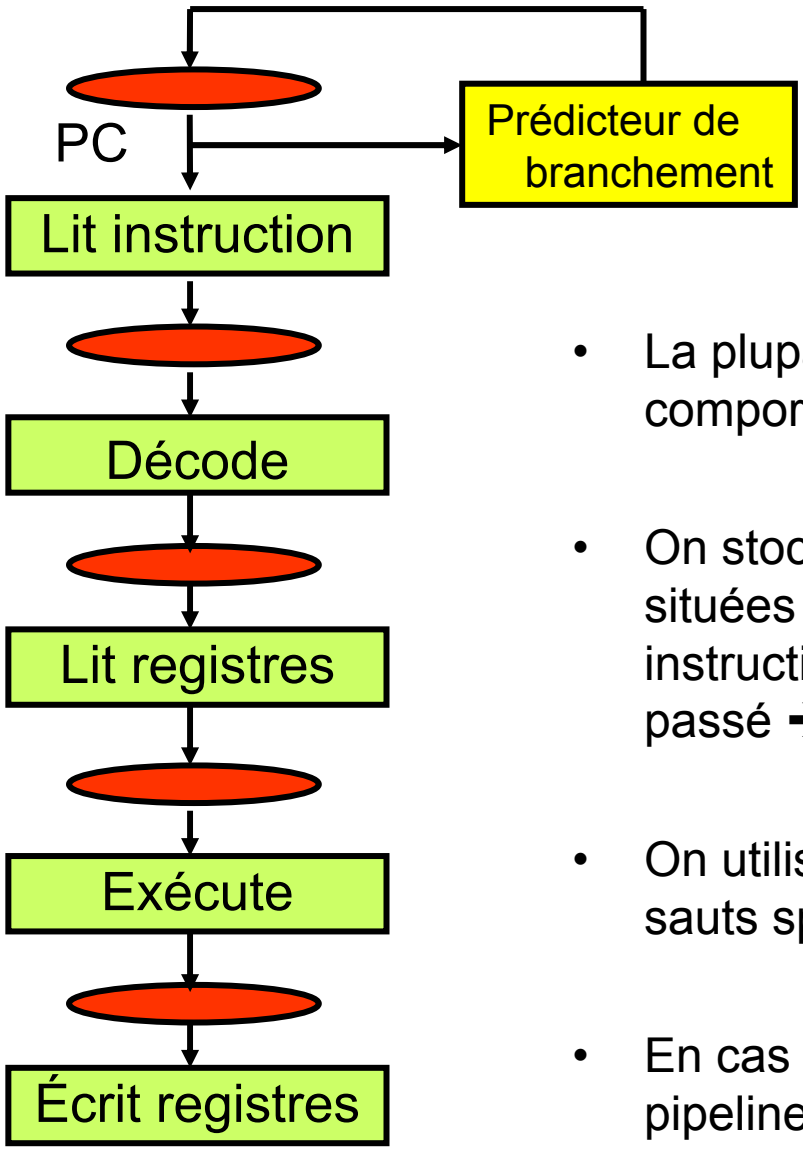
En cas de saut

- À l'étage d'exécution, si on s'aperçoit qu'on aurait du faire un saut, on invalide les instructions dans les étages en amont et on recommence la lecture d'instructions à partir du bon PC

- Problème: on perd de la performance
  - Les instructions de contrôle sont fréquentes
  - Le nombre de cycles « perdus » dépend de la longueur du pipeline
    - Pentium 4: 20 cycles

- Exemple:
  - 1 saut toutes les 10 instructions
  - Si 10 cycles sont perdus toutes les 10 instructions, le débit d'instructions vaut 10/(10+10) = 50% du débit maximum
    - ➔ le temps d'exécution du programme est double

Prédicteur de branchements



- La plupart des programmes ont un comportement répétitif
- On stocke dans des mémoires spéciales situées sur la puce des informations sur les instructions de contrôle et leur comportement passé ➔ prédicteur de branchements
- On utilise ces informations pour effectuer les sauts spéculativement
- En cas de mauvaise prédiction, on vide le pipeline ➔ pénalité de mauvaise prédiction

Exemples de branchements

Calcul de valeur min sur des valeurs aléatoires x[i]

```
int MIN = x[0];
for (i=1; i<1000; i++)
    If (x[i] < MIN)
        MIN = x[i];
```

Branchement conditionnel de boucle

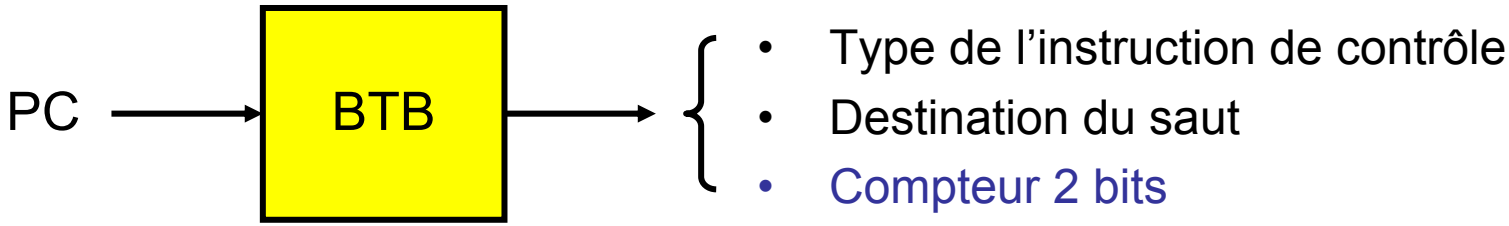
Branchement conditionnel de test

- Branchement de boucle: non pris à la dernière itération seulement
- Branchement de test: une chance sur N pour que la N^{ème} valeur soit le min des N premières valeurs
  - 1ère itération: une chance sur 2 d'être non pris
  - 2ème itération: une chance sur 3
  - 3eme itération: une chance sur 4
  - …
  - 999ème itération: une chance sur 1000

$$\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{1000}\approx 6.5$$

BTB (*Branch Target Buffer*)

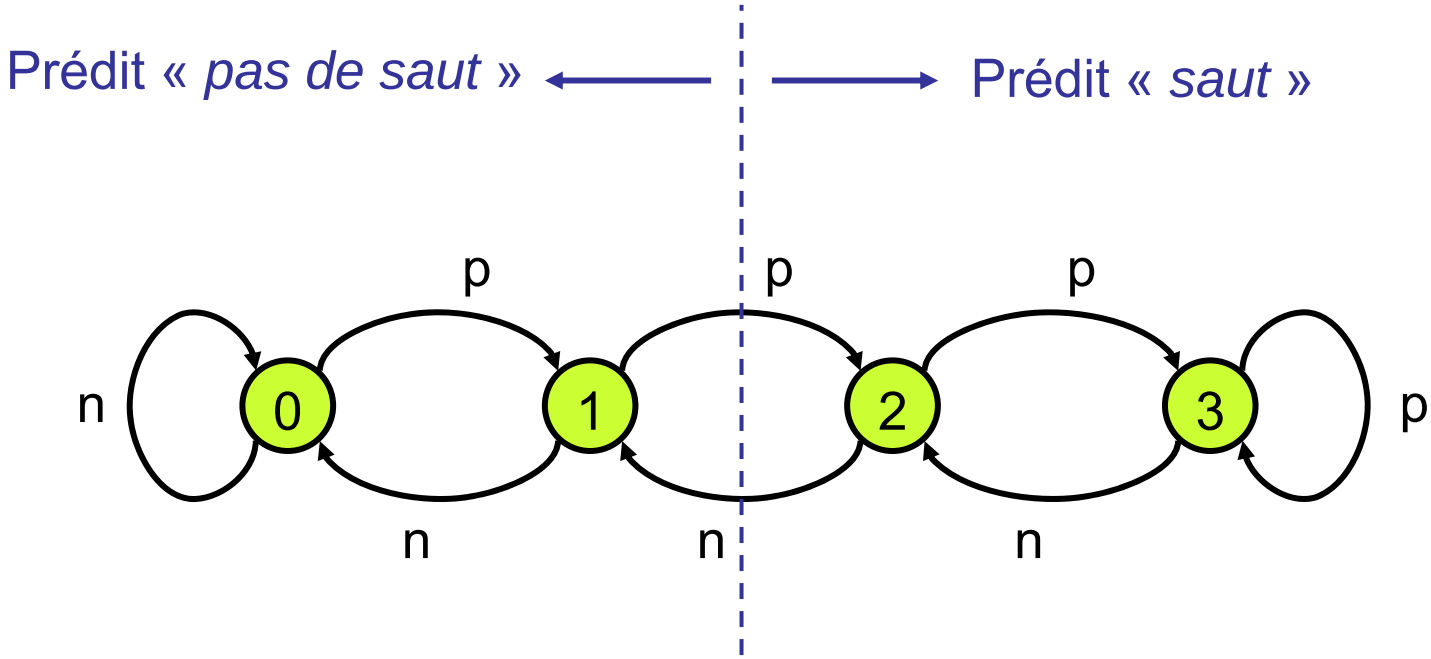
- Comme un cache, sauf qu'on y stocke des informations sur les instructions de contrôle
  - Tag = bits de poids fort de l'adresse de l'instruction de contrôle
  - Peut être direct ou associatif par ensemble
  - Pentium 4: 4k entrées



- En cas de miss, on incrémente le PC normalement
- On écrit une entrée lorsqu'on exécute un saut et qu'il y a eu un miss BTB pour cette instruction de contrôle
- Si par la suite on *hit* sur cette entrée, on utilise l'information stockée pour faire une prédiction

Compteur 2 bits

Un même branchement conditionnel peut être parfois pris, parfois non pris



- Incrémente lorsque le branchement est pris
- Décrémente lorsque le branchement est non pris
- Fait un saut (spéculativement) si valeur ≥ 2

Pourquoi un compteur 2 bits ?

Exemple:

```
for (i=0; i<10; i++)
    for (j=0; J<10; j++)
        x = x + a[i][j];
```

Branchement exécuté 100 fois

➔ $(p^9\, n)^{10}$

- Avec le compteur 2 bits, on ne fait une mauvaise prédiction qu'une fois sur 10, à chaque fois qu'on sort de la boucle interne
  - pppppppppnpppppppppnppp…
- Si, au lieu d'utiliser un compteur 2 bits, on effaçait l'entrée BTB lorsque le branchement est non pris, on ferait une mauvaise prédiction en entrée et en sortie de la boucle interne
  - pppppppppnpppppppppnppp…

Pile d'adresses de retour

- Retour de procédure
  - Destination du saut = adresse enregistrée dans registre lors du CALL
  - Dépend du point d'appel

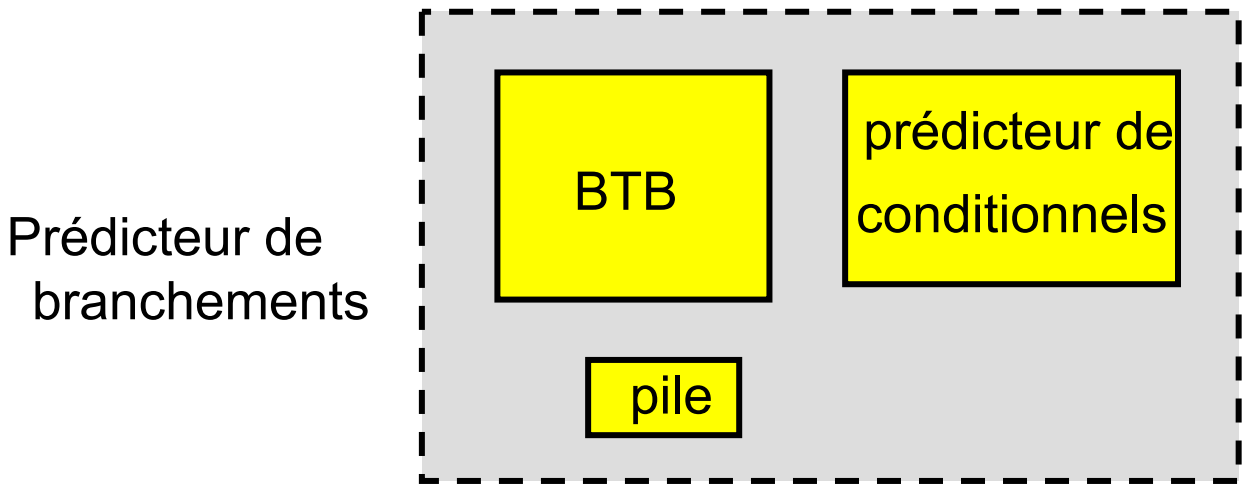
- Le BTB prédit correctement dans certains cas
  - Exemple: procédure appelée plusieurs fois de suite depuis le même point d'appel

- Mais BTB insuffisant lorsqu'il y a plusieurs points d'appel

- ➔ Le prédicteur contient une petite pile (=mémoire+pointeur)
  - Lorsque le BTB détecte un CALL, empiler l'adresse de retour
  - Lorsqu'on exécute un retour, dépiler l'adresse et sauter spéculativement
    - plus facile si le jeu d'instructions permet d'identifier les retours de procédure

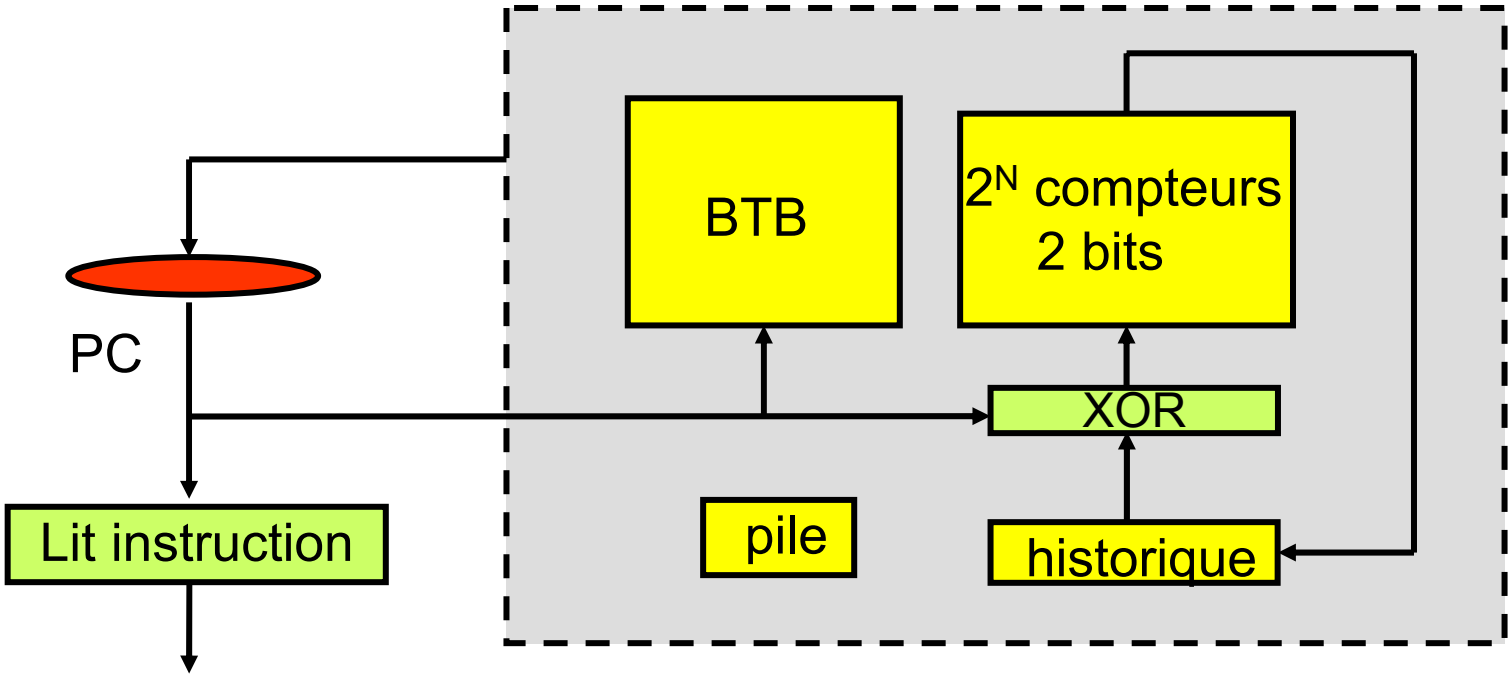
Prédicteur de conditionnels

- Le BTB seul permet d'obtenir ~ 90 % (en moyenne) de bonnes prédictions sur les branchements conditionnels ➔ on sait faire mieux
- Stocker les compteurs 2 bits dans une mémoire séparée = prédicteur de conditionnels
- ➔ On peut l'accéder avec des informations autres que le seul PC



Exemple: prédicteur *gshare*

- 2^N compteurs 2 bits
- Historique = direction des N derniers branchements conditionnels dans un registre à décalage (non pris → 0, pris → 1)
- Accède compteur 2 bits en XORant N bits du PC avec les N bits d'historique
- En cas de mauvaise prédiction, on « répare » l'historique



Pourquoi ça marche: exemple 1

```
for (i=0; i<10; i++)
    for (j=0; J<10; j++)
        x = x + a[i][j];
```

→ p^9n

→ p^9n

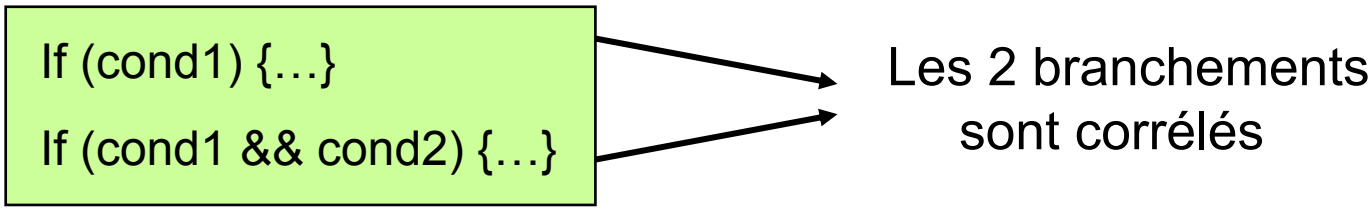
pppppppppnppppppppppnppp…

Prédit *non-pris* lorsque historique = npppppppppp

➔ Besoin de 11 bits d'historique

➔ 2048 compteurs 2 bits

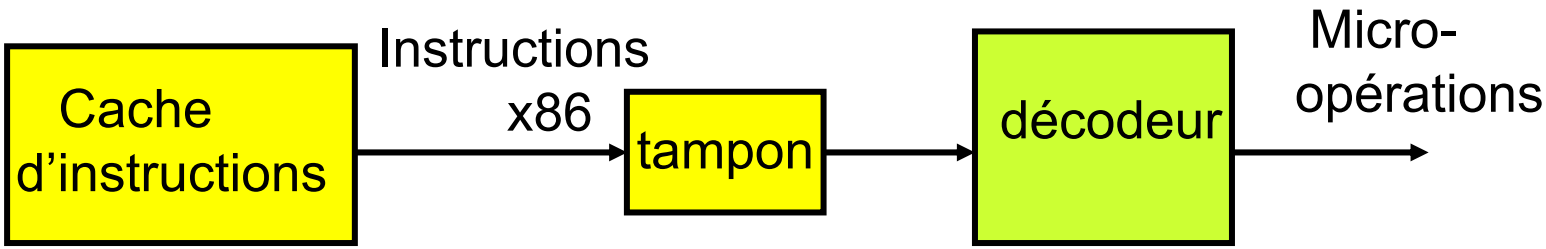
Pourquoi ça marche: exemple 2



Si le 1er branchement est pris (cond1 faux), on est sur que le 2ème branchement sera pris

Cas des jeux d'instructions CISC

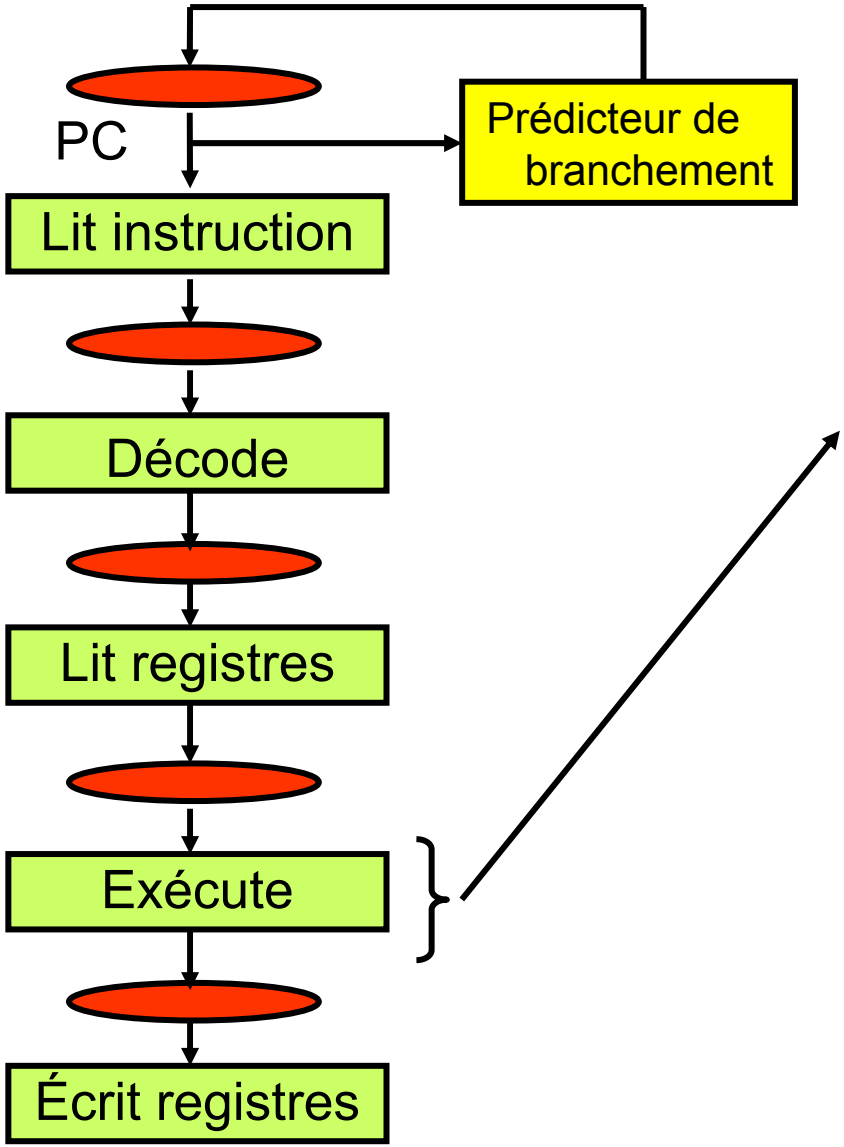
- La complexité matérielle augmente avec le nombre de ports sur les registres
  - Réseau de bypass plus complexe
  - Latence d'accès aux registres augmente
- Jeux d'instructions RISC: 2 ports de lecture et 1 port d'écriture
- Jeux d'instructions CISC: certaines instructions nécessitent plus de ports
- Solution utilisée dans les processeurs Intel x86 ➔ décomposer instruction CISC en micro-opérations RISC



➔ Pipeline d'instructions plus long et plus complexe

➔ Pénalité de mauvaise prédiction de branchement plus élevée

Opérations de durées différentes



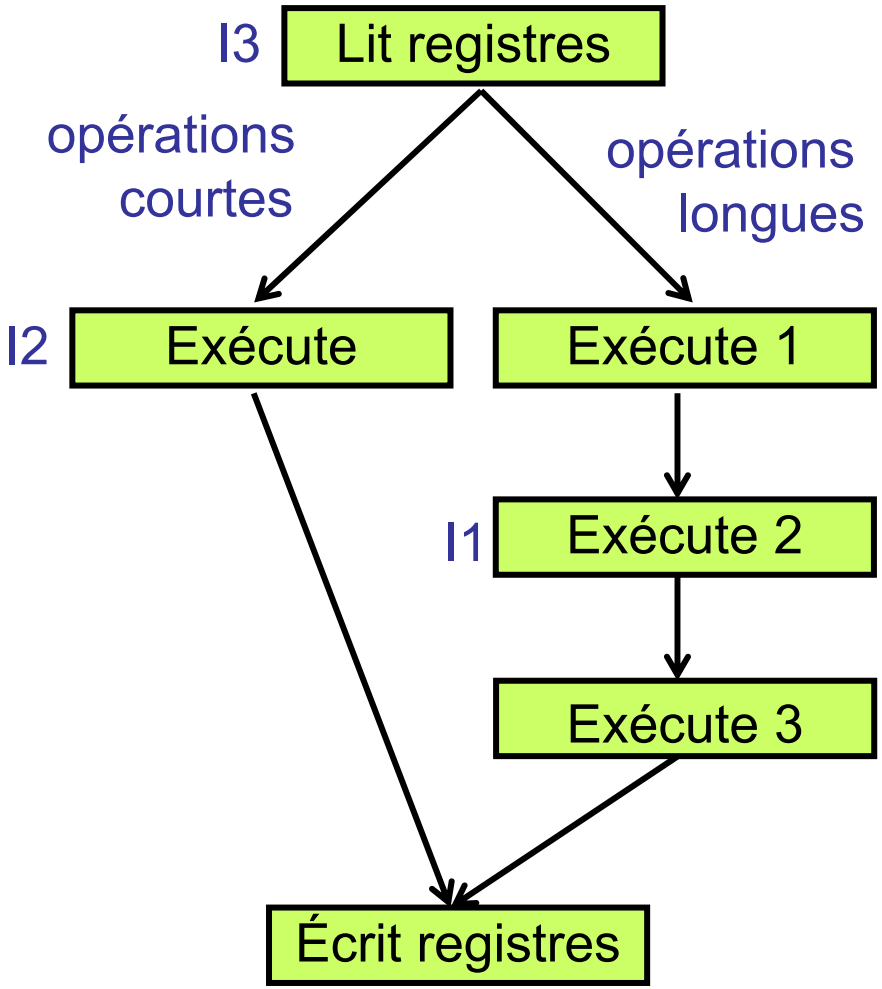
C'est l'opérateur le plus lent qui dicte la fréquence d'horloge ?

☹

Solution: pipeliner les opérateurs

Mais attention …

Problème des fausses dépendances



- Instruction longue I1 suivie de 2 instructions courtes I2 et I3
  - Écriture I1: cycle N
  - Écriture I2: cycle N-1
  - Écriture I3: cycle N
- I1 et I3 écrivent en même temps
  - Problème si 1 seul port d'écriture
- I2 écrit avant I1
  - (Et si une interruption se produit ?)
- Si I1 et I2 écrivent le même registre
  ➔ état incohérent !

Solutions possibles

- Attendre 2 cycles avant de lancer I2
  - Si on veut éviter ça, le compilateur doit être « intelligent »

- Rajouter des étages fictifs aux opérations courtes pour retarder l'écriture
  - Étage fictif = flip-flop
  - Les écritures se font dans l'ordre du programme
  - Inconvénient: réseau de bypass complexe

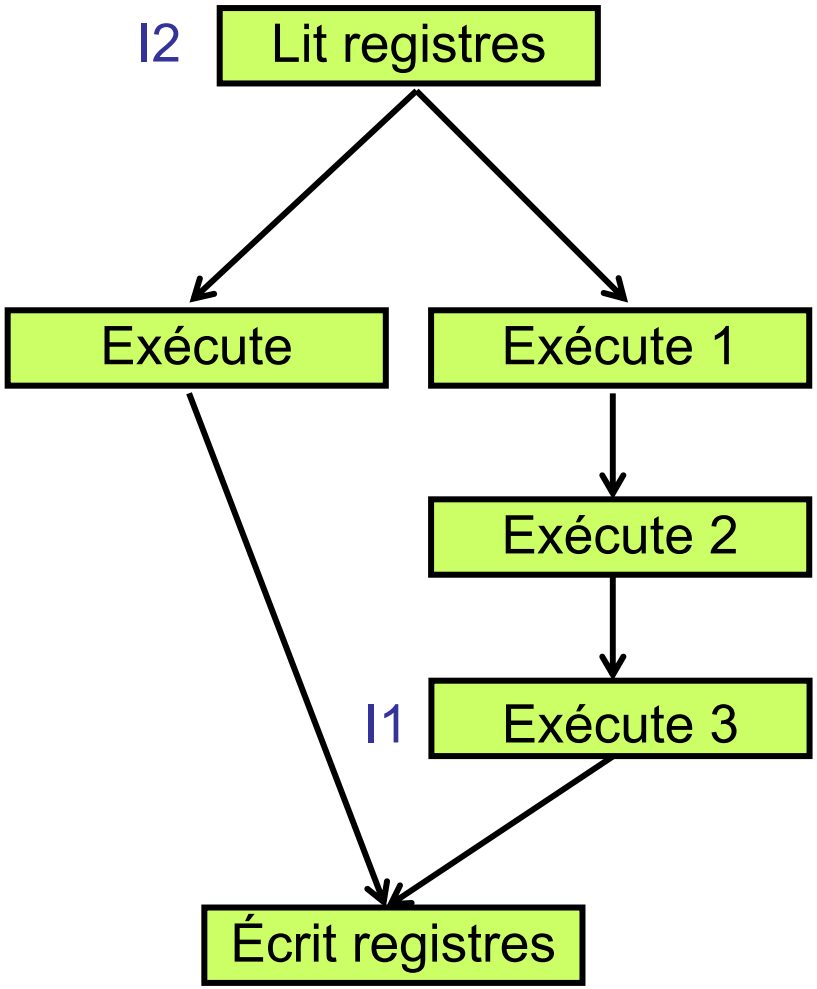
- Renommage de registre
  - Registres architecturaux (RA) du jeu d'instruction: ceux vus par le programmeur
  - Registres physiques (RP) plus nombreux que registres architecturaux
    - ~ RA + nombre d'instructions dans pipeline
  - Table de renommage donne correspondance RA→RP
  - Avant l'exécution de chaque instruction, on lui affecte un RP libre et on met à jour la table de renommage
  - Les écriture peuvent se faire dans le désordre car les RP sont distincts
  - « Réparer » la table de renommage sur branchement mal prédit

Différentes méthodes de renommage

- Renommage « permanent » (Alpha 21264, MIPS R10k, IBM POWER4…)
  - Maintenir une liste de RP libres
  - À l'étage de décodage, on affecte un RP libre à l'instruction
  - Écriture dans RP immédiatement après l'exécution
  - RP remis dans liste libre lorsque RA réécrit par une autre instruction
  - Nécessite plusieurs ports d'écriture sur RP

- Renommage temporaire (Intel x86, …)
  - RP = RA + registres physiques temporaires (RPT)
  - À l'étage de décodage, on affecte un RPT libre à l'instruction
  - Écriture dans RPT immédiatement après l'exécution
  - Recopie RPT dans RA, dans l'ordre du programme
    - Libère le RPT
    - Met à jour la table de renommage (2ème fois)
  - 1 port d'écriture sur RA suffisant
    - Mais plus de ports sur table de renommage

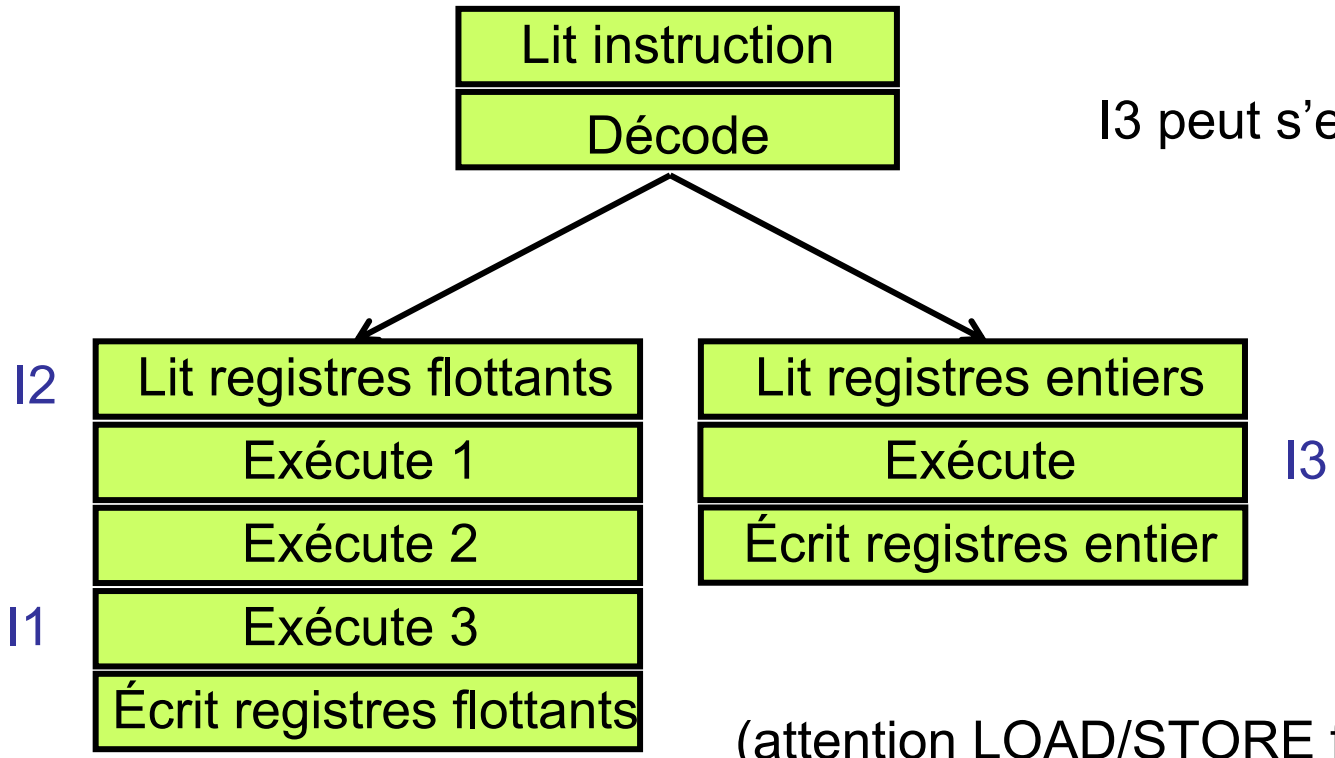
Vraies dépendances



- Si le résultat produit par I1 est un opérande de I2, l'exécution de I2 doit être différée de 2 cycles
- Que faire pour I3 ?
- Méthode simple: lorsqu'une instruction est bloquée, bloquer toutes les instructions en amont
- En théorie, si I3 ne dépend ni de I1 ni de I2, on pourrait l'exécuter sans attendre

Entiers / flottants

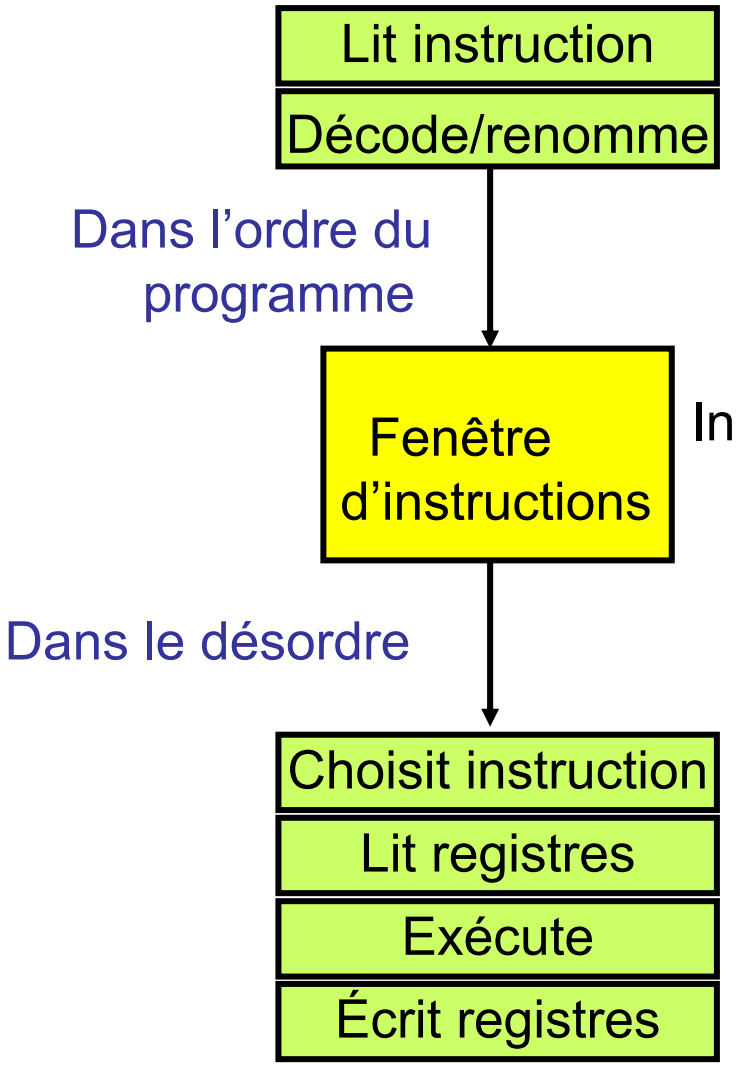
Registres « entiers » et « flottants » dans des mémoires SRAM séparées



I3 peut s'exécuter avant I2

(attention LOAD/STORE flottants: accès registres entiers pour calcul d'adresse)

Processeur *Out-of-Order*



Intel x86, AMD, IBM, …

Instructions en attente d'exécution (qq dizaines)

Les vraies dépendances sont respectées

Processeur superscalaire

- Plusieurs instructions décodées par cycle
- ➔ Lit et exécute plusieurs instructions par cycle

- Exemples:
  - Intel x86 ➔ décode 3 inst/cycle
  - AMD Opteron ➔ décode 3 inst/cycle
  - IBM POWER5 ➔ décode 5 inst/cycle
  - Intel Itanium 2 ➔ décode 6 instructions/cycle

- Nécessite plus de ports de lecture/écriture sur les registres

- Plusieurs ports sur le cache de données
  - Autre méthode: dupliquer le cache ➔ 2 LOAD en parallèle

- Les circuits de contrôle de l'exécution sont complexes

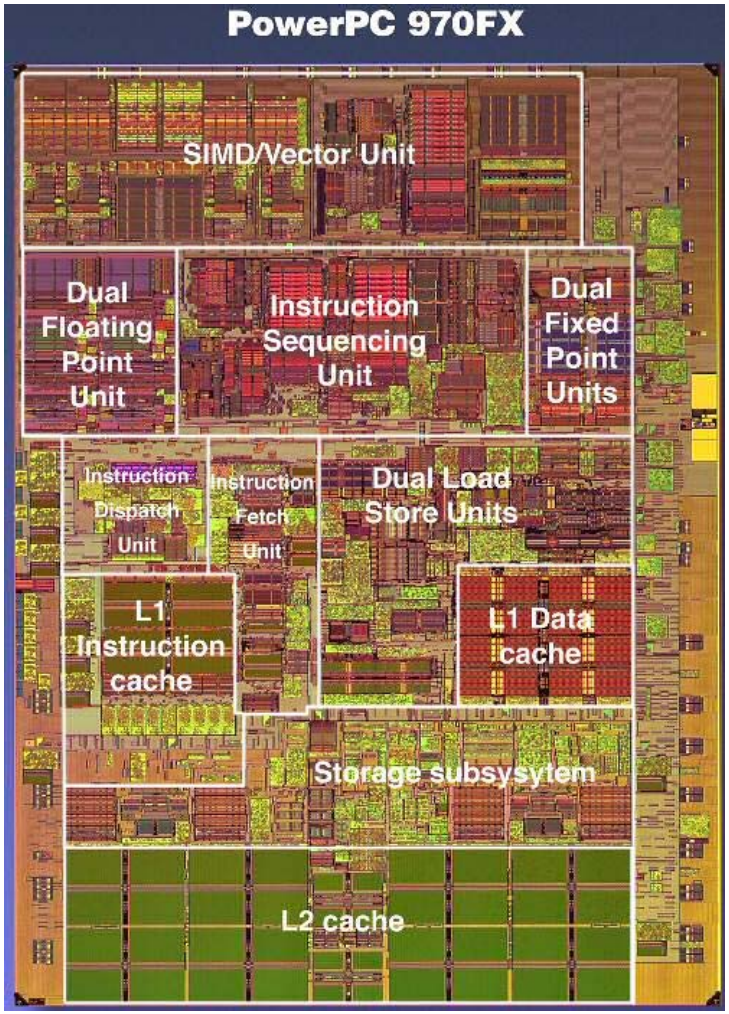
Défauts de cache pipelinés

- Défaut de cache ➔ jusqu'à plusieurs centaines de cycles d'attente
- Exécution out-of-order permet de continuer à travailler en attendant
- ➔ D'autres défauts de cache peuvent se produire
- Plusieurs défauts de cache peuvent être en cours de traitement simultanément
- Les accès mémoire sont pipelinés
- ➔ permet de « masquer » la latence mémoire
  - Sauf si accès par pointeurs (liste chainée, arbre)

Préchargement de cache

- Si on détecte des miss successifs à des lignes consécutives, on précharge les lignes suivantes dans le cache, avant que le programme ne les demande
  - Principe de localité spatiale
- Peut être plus sophistiqué: accès mémoire par pas constant
  - Utilisé dans Pentium M et dérivés
- Préchargement efficace si suffisamment de bande passante

IBM PowerPC 970FX



- Technologie 90 nm
- 58 millions de transistors
- 9.4 mm × 7.1 mm
- 2 GHz
- Superscalaire, out-of-order
- Décode 5 inst/cycle
- Pipeline: 16 étages minimum
- Cache d'instructions: 64 Ko
- Cache de donnée: 32 Ko
- Cache de niveau 2: 512 Ko
- ~ 200 instructions simultanément dans le pipeline

Multi-threading

- À un instant donné, une tâche n'utilise pas toutes les ressources d'exécution du processeur
  - Défauts de cache, accès mémoire
  - Dépendances de données

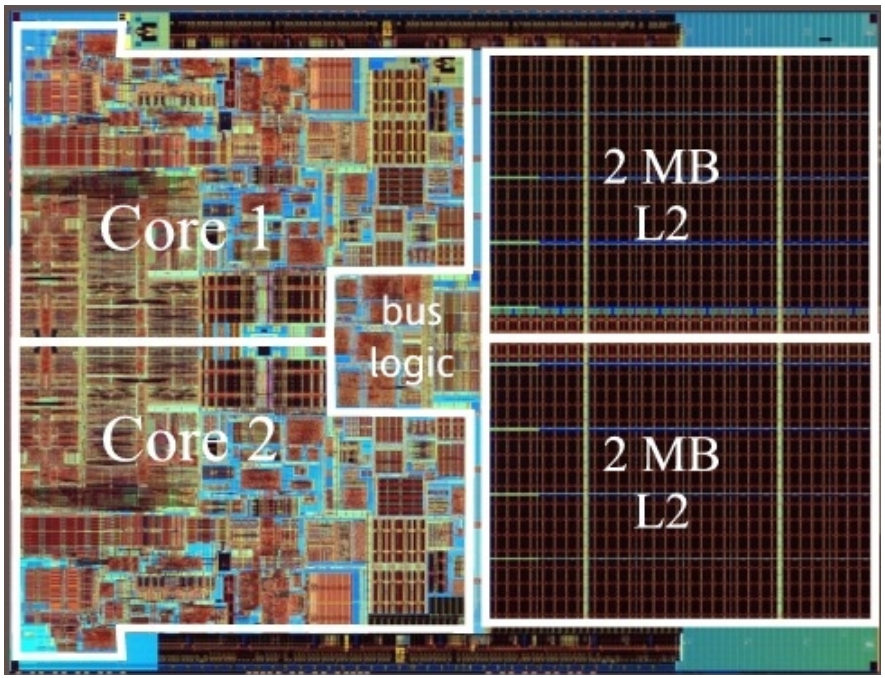
- Autoriser plusieurs tâche à s'exécuter simultanément permet de mieux utiliser les ressources

- *Switch-on-event*
  - Pendant qu'une tâche attend un accès mémoire, exécute une autre tâche
  - Utilisé dans Intel Itanium Montecito

- SMT (*Simultaneous Multi-Threading*)
  - Les tâches sont vraiment en exécution simultanée
  - Utilisé dans Intel Pentium 4, IBM POWER5, Sun UltraSPARC T1

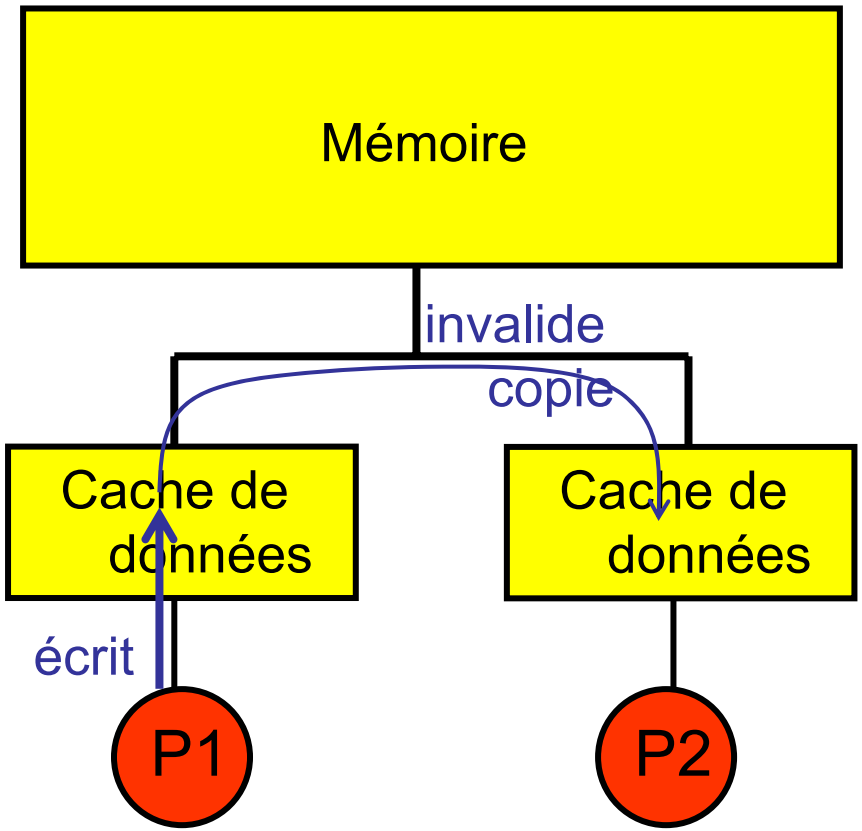
Multi-coeur

Plusieurs processeurs sur une même puce



- Exemple: Intel Core 2 Duo (65nm, 2.66 Ghz)
  - 2 processeurs
  - 291 millions de transistors (total)
  - 144 mm^2
  - 65 watts

Multi-cœur: maintenir la cohérence des caches



- Exemple
  - Tâche T1 sur processeur P1
  - Tâche T2 sur processeur P2
  - tâches communiquent via des données en mémoire
  - ➔ certaines lignes de cache sont répliquées sur les 2 processeurs
  - Si T1 écrit à l'adresse X, et qu'une copie de X se trouve sur le cache de P2, il faut invalider la copie
  - Si ensuite T2 lit X, et que les caches sont à écriture différée, c'est le cache de P1 qui fournit la donnée (*cache-to-cache*)

Consommation d'énergie / dissipation

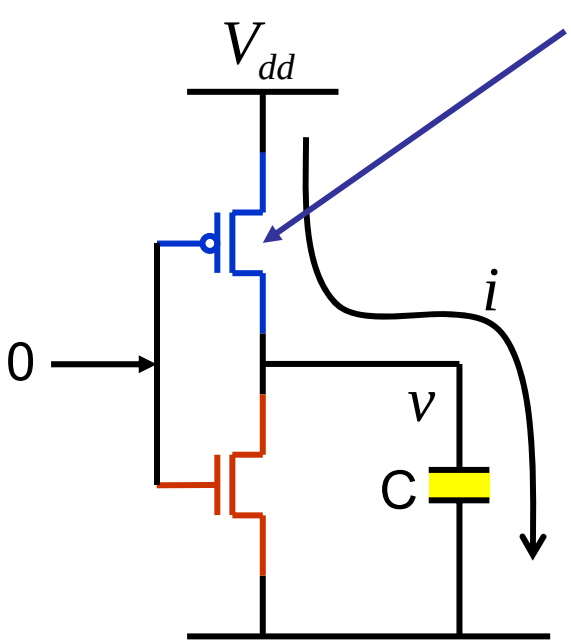
- Contrainte majeure (technique et économique)

- Énergie (joules)
  - Coût (€,$) ➔ important pour les data centers
  - Systèmes portables (temps de déchargement de la batterie)

- Température
  - Température sur la puce: 85 à100 °C maxi pour un bon fonctionnement
  - Machine de bureau: coût de la machine, bruit du ventilateur
  - Système portable: fortes contraintes géométriques

- Puissance (watts)
  - Ampérage limité
  - Data centers: échauffement de la pièce par effet de masse
    - ➔ limite le nombre de processeurs par pièce

Consommation dynamique



$$\int_0^{\infty} (V_{dd} - v) i dt = \int_0^{V_{dd}} C(V_{dd} - v) dv = \boxed{\frac{1}{2} C V_{dd}^2}$$

L'énergie dissipée par transition de bit ne dépend que de la capacité et de la tension

On peut diminuer l'énergie consommée en réduisant l'échelle de gravure (C plus petit) et en réduisant la tension d'alimentation

Consommation statique

- Les transistors ne sont pas des interrupteurs parfaits, il y a des courants de fuite
- ➔ Une porte consomme de l'énergie même lorsqu'elle ne travaille pas (pas de transition)
- 20 à 30 % de la consommation dans les processeurs haute performance actuels
  - C'est la consommation principale dans les caches de niveau 2 et 3
- Pour supprimer la consommation statique, il faut déconnecter la porte de l'alimentation électrique
  - *Power gating*
  - Attention: Les mémoires (SRAM) perdent leur contenu

Réduire la consommation

- Quand un circuit n'est pas utilisé, il consomme de l'énergie inutilement
- Exemples:
  - certains programmes n'utilisent pas les opérateurs en virgule flottante
  - sur un accès mémoire long et bloquant, les opérateurs sont inutilisés durant l'accès

- *Clock gating*
  - Quand un circuit n'a pas été utilisé depuis quelques cycles, le signal d'horloge est déconnecté automatiquement de cet opérateur
  - ➔ plus de transitions, consommation dynamique minimale
  - Le circuit se « réveille » à la prochaine utilisation
    - Quasi immédiat, 1 ou 2 cycles

- *Power gating*
  - Le « réveil » est beaucoup plus long

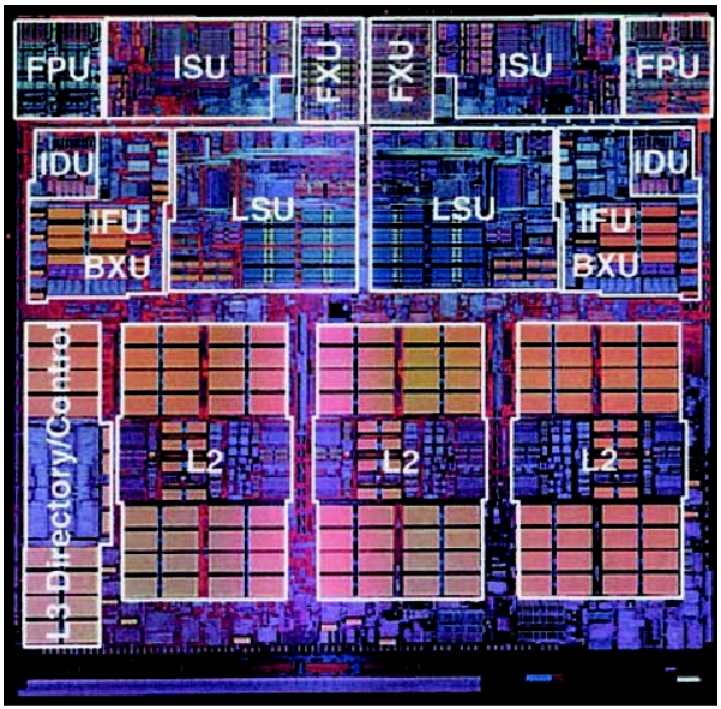
Température

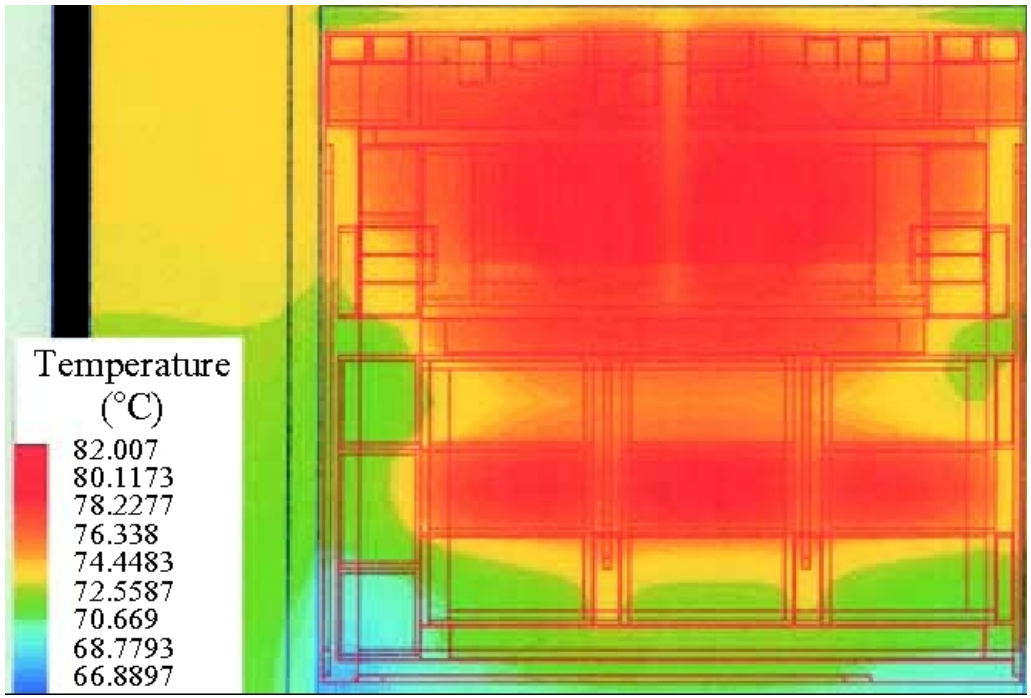
- Température nominale sur la puce = 85 °C
- Durée de vie du circuit dépend exponentiellement de la température
  - Un circuit en fonctionnement se dégrade avec le temps
    - Exemple: electromigration
  - Température élevée augmente la vitesse de dégradation
  - +10 °C ➔ durée de vie moyenne divisée par 2
- Le temps de réponse des circuits augmente avec la température
  - Augmentation de la résistivité des matériaux (silicium,cuivre)
- Les courants de fuite augmentent exponentiellement avec la température

La température sur la puce n'est pas uniforme

Différences spatiales ➔ jusqu'à plusieurs dizaines de °C

Exemple: IBM POWER4 (2 processeurs)





Régulation puissance / température

- La puissance dissipée dépend de l'application qui s'exécute
  - Peut aller du simple au double
  - Voire plus (« virus » thermique)
- La température dépend …
  - De l'application qui s'exécute
  - De la vitesse du ventilateur
  - De la température ambiante

- Capteurs (puissance,température) intégrés sur la puce
- Lorsque risque de dépassement de la limite, ralentir le circuit pour diminuer la consommation

- Différentes méthodes (au niveau circuit):
  - Réduire la fréquence d'horloge
  - Réduire la fréquence d'horloge **et** la tension d'alimentation
  - *Clock gating* periodique (on/off/on/off…)

Évolution actuelle

- Augmentation du nombre de processeurs par puce
  - Sun UltraSPARC T1: 8 processeurs
  - Intel, AMD: 4 processeurs (2007)

- Augmentation du nombre de threads
  - IBM POWER5: 2 processeurs, 2 tâches par processeur ➔ 4 tâches //
  - Sun UltraSPARC T1: 4 tâches par processeurs ➔ 32 tâches //

- La fréquence est bloquée entre 1 et 4 GHz depuis plusieurs années
  - IBM annonce un POWER6 à 5 GHz pour 2007 (technologie 65 nm)

- L'évolution future dépendra de la capacité du logiciel (programmeurs, compilateurs) à exploiter les multi-coeurs

Quelques références

- John Hennessy & David Patterson
  - *Computer Architecture: A Quantitative Approach*, 4ème édition
  - Éditeur: Morgan Kaufmann
- William Stallings
  - *Computer Organization and Architecture: Designing for Performance*, 7ème édition
  - Éditeur: Prentice Hall
- WWW Computer Architecture Page
  - http://www.cs.wisc.edu/~arch/www
- Processeurs des années 90
  - http://www.irisa.fr/caps/projects/TechnologicalSurvey
  - (en français)