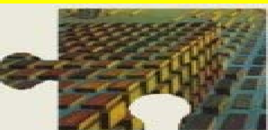
Architecture

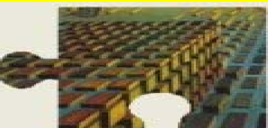
B. Goossens et D. Defour

Dali
Université de Perpignan Via Domitia

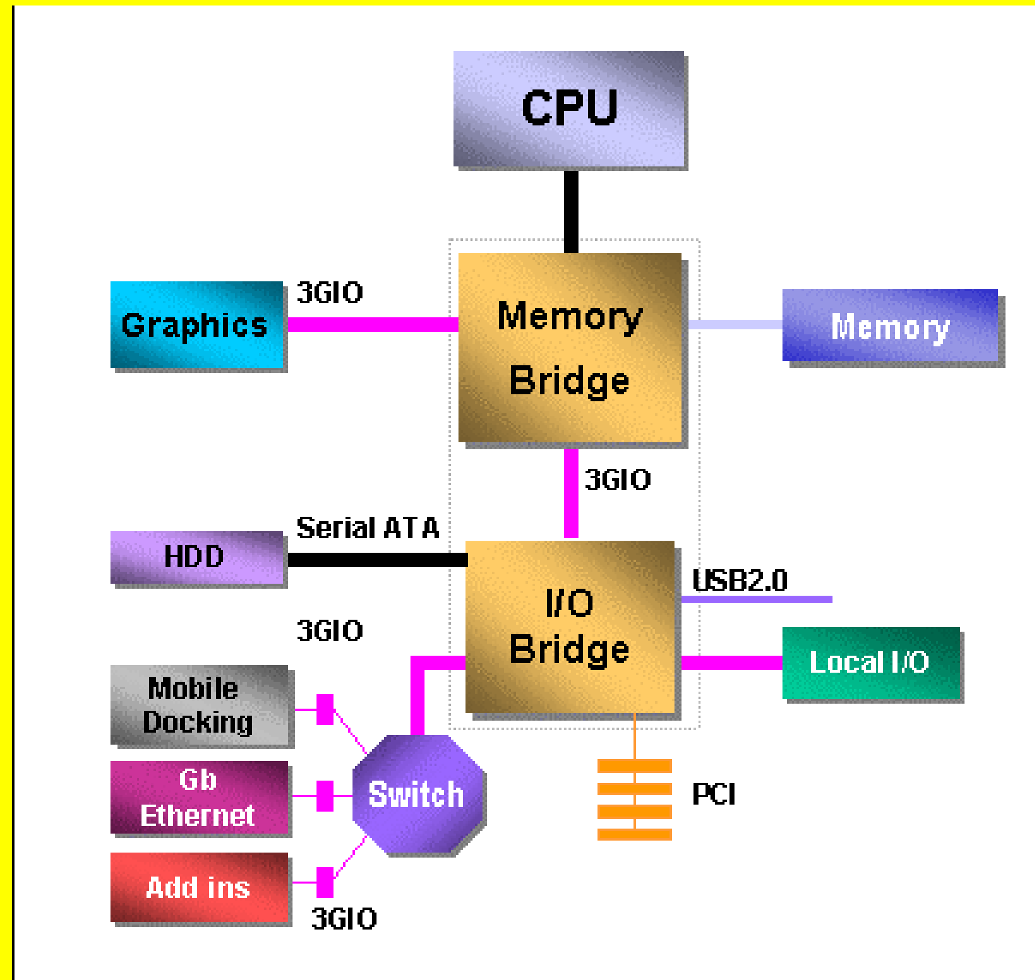


Plan du cours

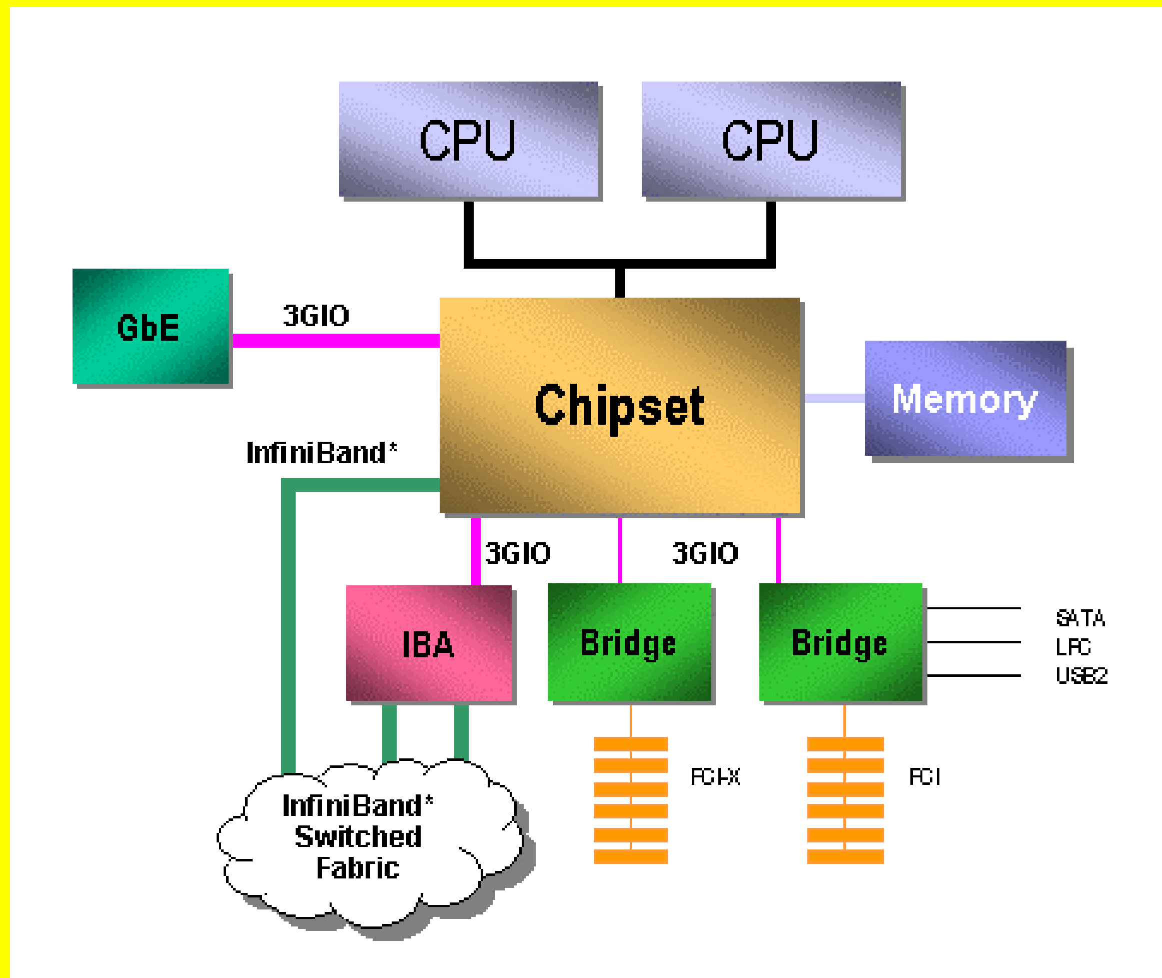
- Introduction
- Technologie
- Circuiterie
- Micro-architecture
- Quelques systèmes actuels



Quelle machine? un PC de base?



Quelle machine? Une station bi-processeur?



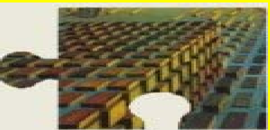
Quelle machine? Un serveur réseau?

CPU
CPU
GbE
3GIO
Chipset
Memory
Memory
PCI Bridge
3GIO x 8
3GIO x 8
Switch
Line Card
Line Card
Line Card
Line Card
Line Card

Quel processeur? Intel P4?

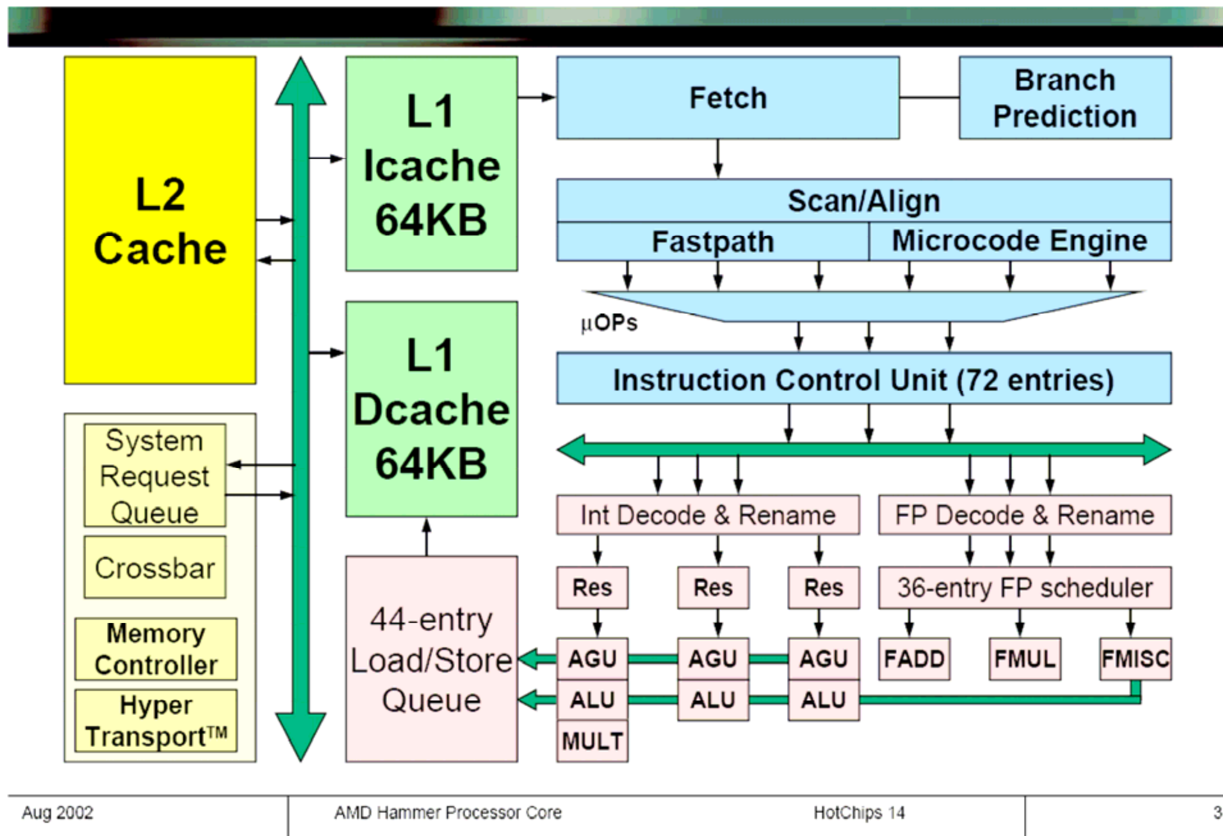
Pentium® 4 Processor
Block Diagram

3.2 GB/s System Interface
L2 Cache and Control
BTB
BTB & I-TLB
Decoder
Trace Cache
uCode ROM
3
Rename/Alloc
3
uop Queues
Schedulers
Integer RF
FP RF
Store AGU
Load AGU
ALU
ALU
ALU
ALU
FP move
FP store
FMul
FAdd
MMX
SSE
L1 D-Cache and D-TLB



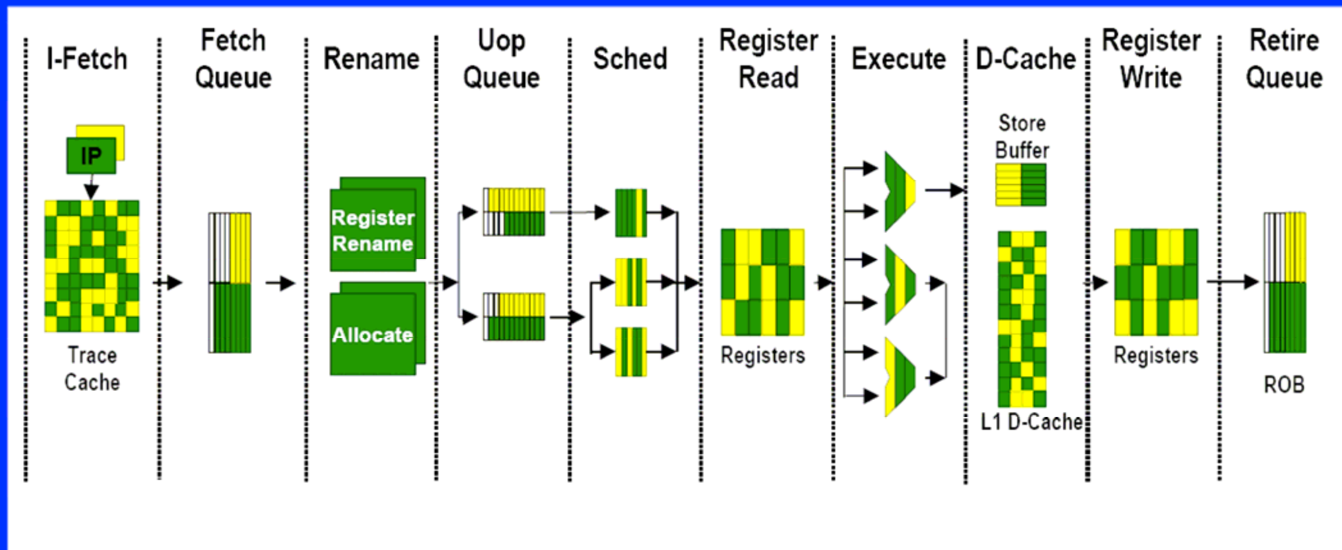
UP VD

Quel processeur? AMD Athlon 64?



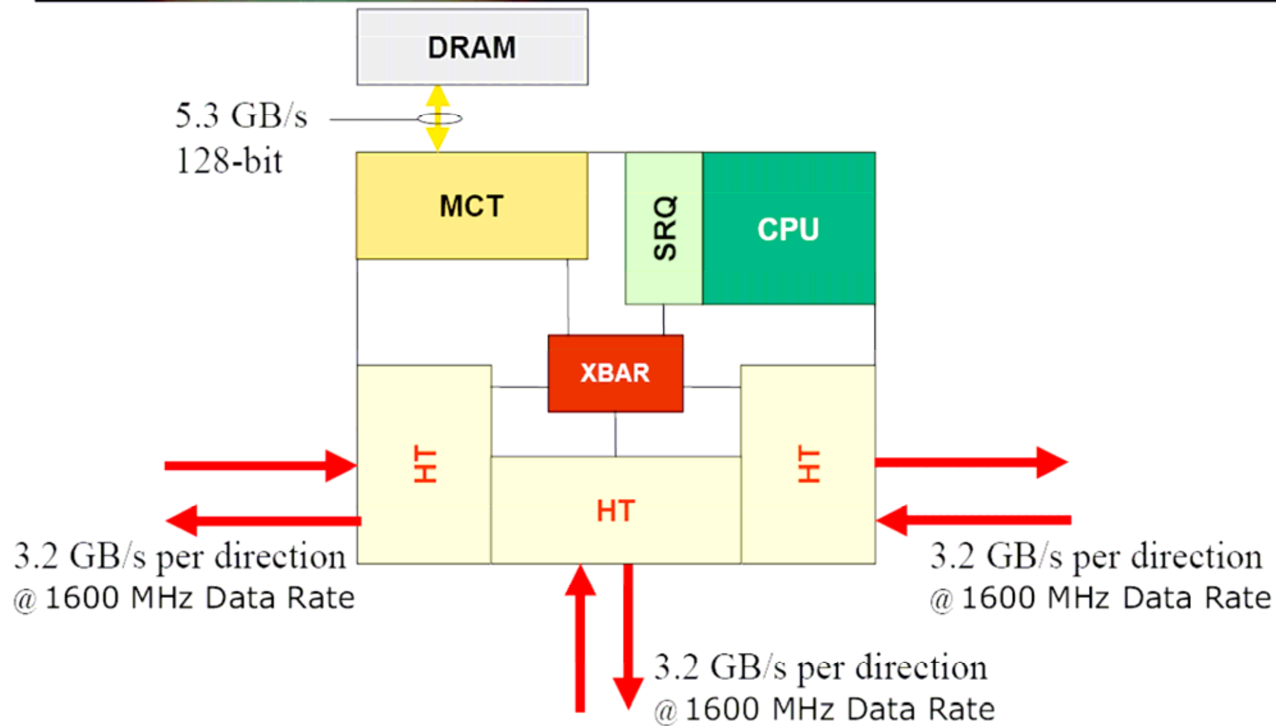
Quel processeur? P4 Hyperthreading?

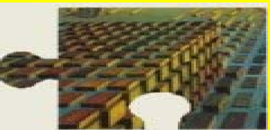
Execution Pipeline



Copyright © 2002 Intel Corporation.

Quel processeur? AMD Opteron?





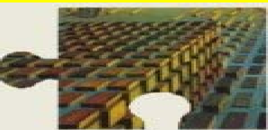
UP VD

Pour répondre à ces questions, il faut connaître:

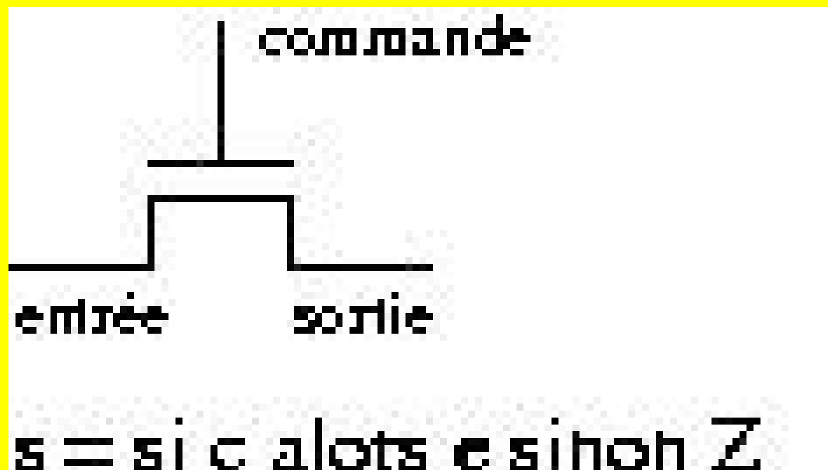
- **les standards de bus (FSB, AGP, PCI, PCIx, USB)**
- **la micro-architecture des contrôleurs (CM, CIO)**
- **la micro-architecture du processeur (pipeline, exécution ooo, spéculation, hiérarchie mémoire, E/S)**

Comprendre la micro-architecture suppose qu'on connaisse:

- **la technologie (CMOS) et les portes de base**
- **la circuiterie (mémoire, calcul, transmission, contrôle)**

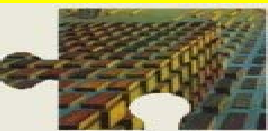


Technologie CMOS



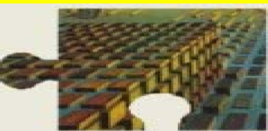
Transistor nMOS **Transistor pMOS**

Un transistor nMOS passe bien les ‘0’
Un transistor pMOS passe bien les ‘1’

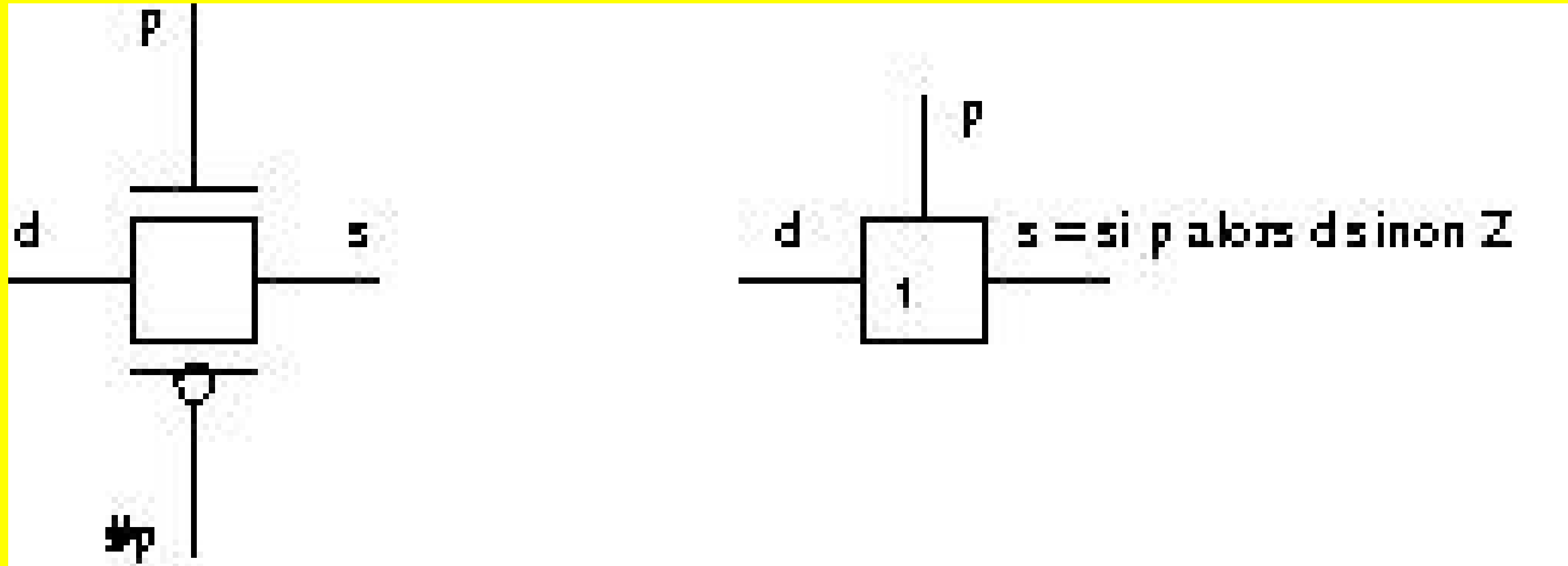


Combien de transistors sur une puce?

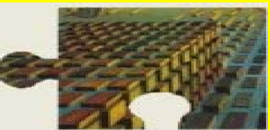
- Une puce est un carré de côté w, surface s
- La finesse de gravure λ permet d'y dessiner $w/4\lambda$ lignes et $w/4\lambda$ colonnes, soit $w^2/16\lambda^2$ cellules
- Dans chaque cellule, on peut déposer 1 transistor
- Plus la logique du circuit est régulière, plus le remplissage est dense (matrice mémoire)
- P4: $s = 237\ mm^2$, $\lambda = 90$ nm, 178M de transistors
- Athlon 64: $s = 193\ mm^2$, $\lambda = 130$ nm, 106M de tr.
- DDR2 512Mb: $s = 70\ mm^2$, $\lambda = 90$ nm, 512M de tr.
- Conso P4: 130W, Athlon: 90W, DDR2: 1W



Porte de transfert CMOS

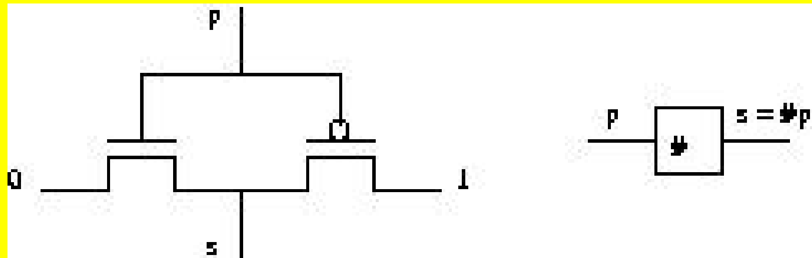


Quand d = 0 et p = 1, le transistor pMOS passe d en s
Quand d = 1 et p = 1, le transistor nMOS passe d en s
La porte de transfert passe bien les ‘0’ et les ‘1’

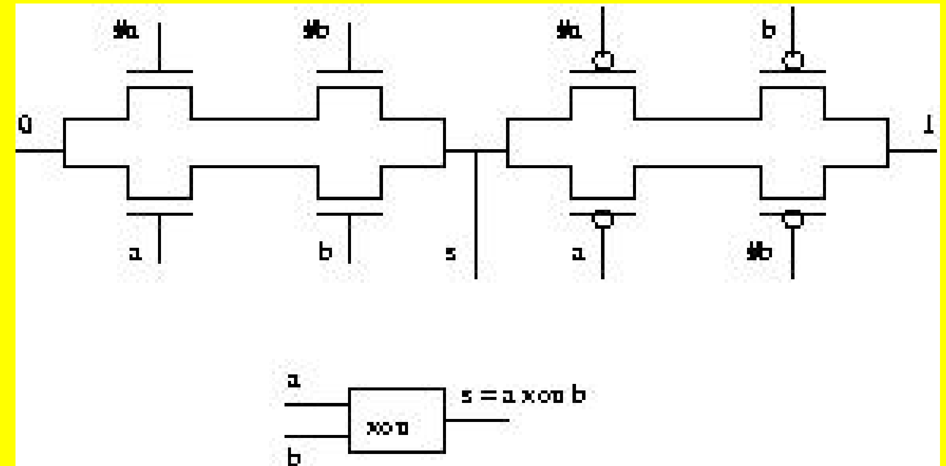


UP VD

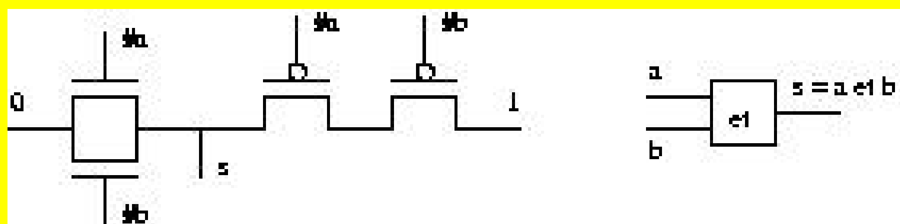
Portes logiques CMOS



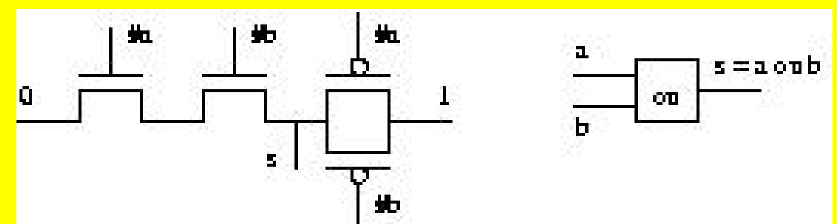
Inverseur



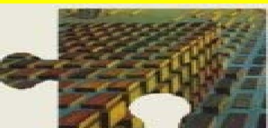
Xor



Et

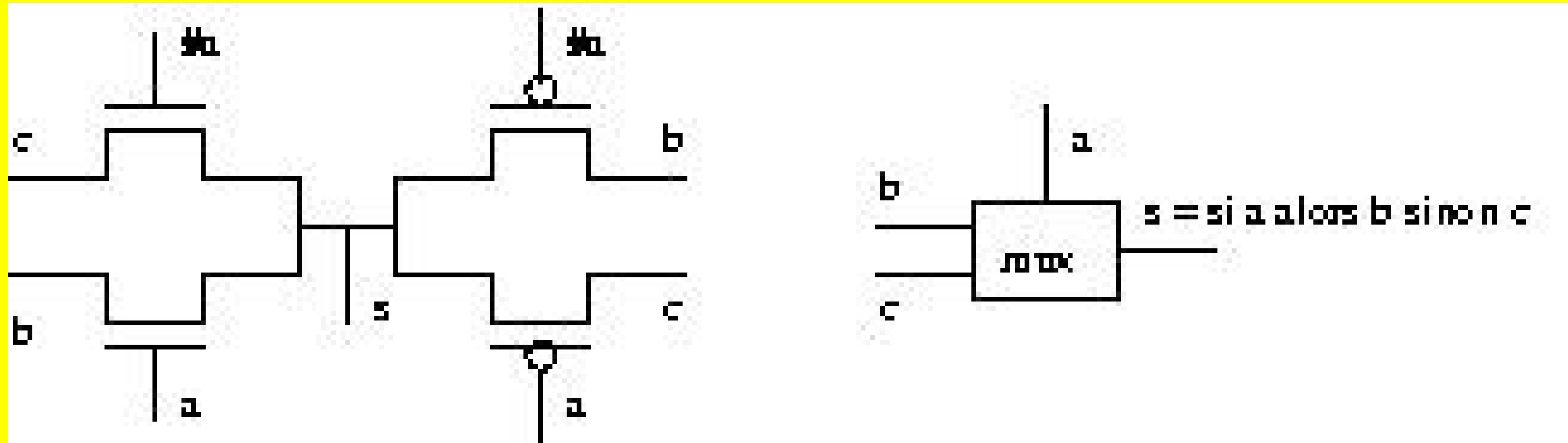


Ou



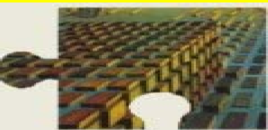
UPVD

Porte de transfert CMOS



Multiplexeur

Une porte de transfert passe une variable
Une porte logique passe une constante
Une porte de transfert affaiblit son entrée
Une porte logique amplifie son entrée



UPVD

Portes CMOS

a
b
et
s = a et b

a
b
#et
s = a #et b

a
b
ou
s = a ou b

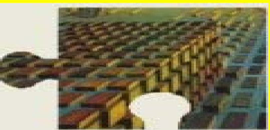
a
b
#ou
s = a #ou b

a
b
xou
s = a xou b

a
b
#xou
s = a #xou b

a
b
c
mux
s = si a
alors b
sinon c

a
#
s = #a



UP VD

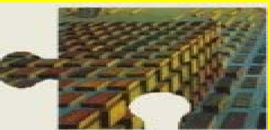
Séparateur CMOS: le verrou

Deux inverseurs en série

Quand h = 1 (verrou passant), s = e

Quand h = 0 (entrée stable), s mémorise e

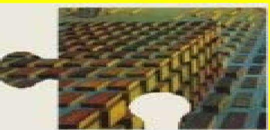
Quand e' remplace e (h stable), s conserve e



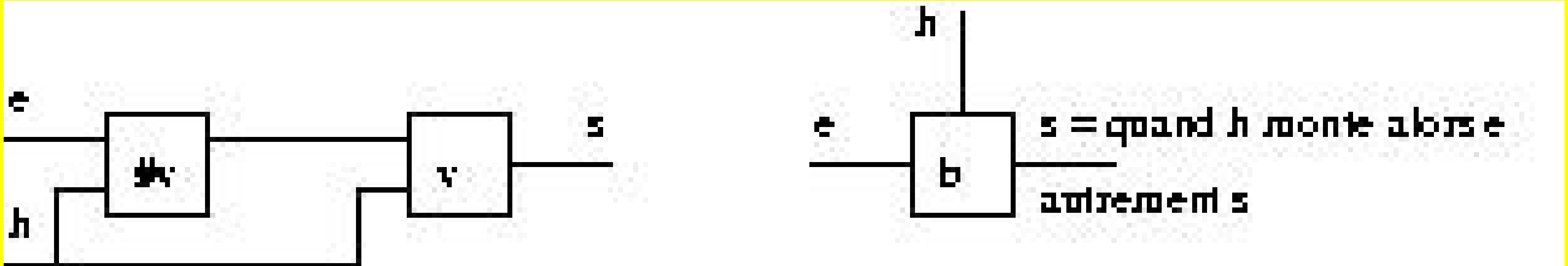
Séparateur CMOS: le verrou

s = si h alors s sinon e

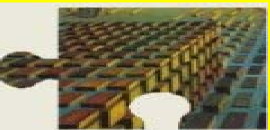
Verrou passant quand h = 0



Séparateur CMOS: la bascule



Quand h = 1 (#v non passant), s est isolé de e
Quand h = 0 (v non passant), s est isolé de e
Quand h monte, le dernier e devant v passe en s



Séparateurs CMOS

h
s = si h
alors a
a
v
sinon s

h
s = si h
alors s
a
[illegible]v
sinon a

p
s = si p
alors a
a
1
sinon Z

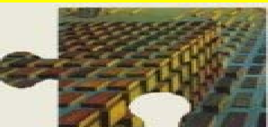
p
s = si p
alors Z
a
[illegible]1
sinon a

h
s = quand h
monte
alors a
a
b
sinon s

h
s = quand h
descend
alors a
a
[illegible]b
sinon s

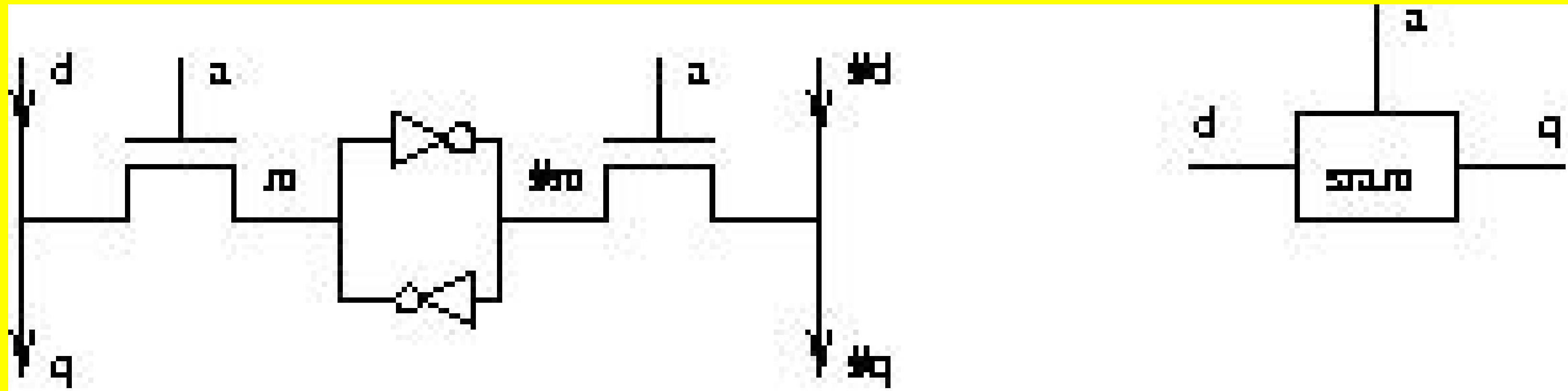
h
s = quand h
monte
alors a et [illegible]
a
b2
sinon s
[illegible]

h
s = quand h
descend
alors a et [illegible]
a
[illegible]b2
sinon s
[illegible]



UP VD

Cellule mémoire SRAM

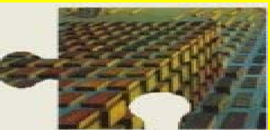


Lecture: d = #d = Z et a = 1 alors q = m

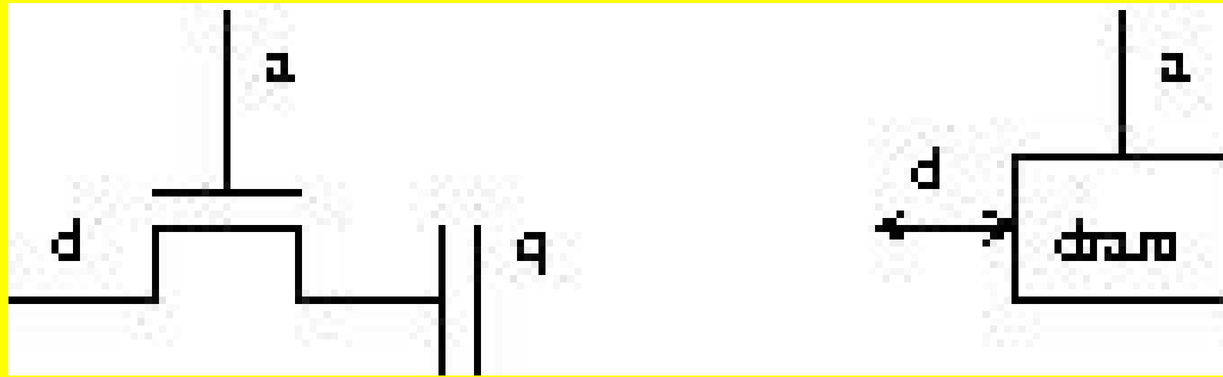
Ecriture: d = m’ et #d = #m’ et a = 1

Quand a = 0, la cellule n’est pas accédée

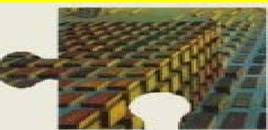
En écriture, ce qui est écrit est aussi lu



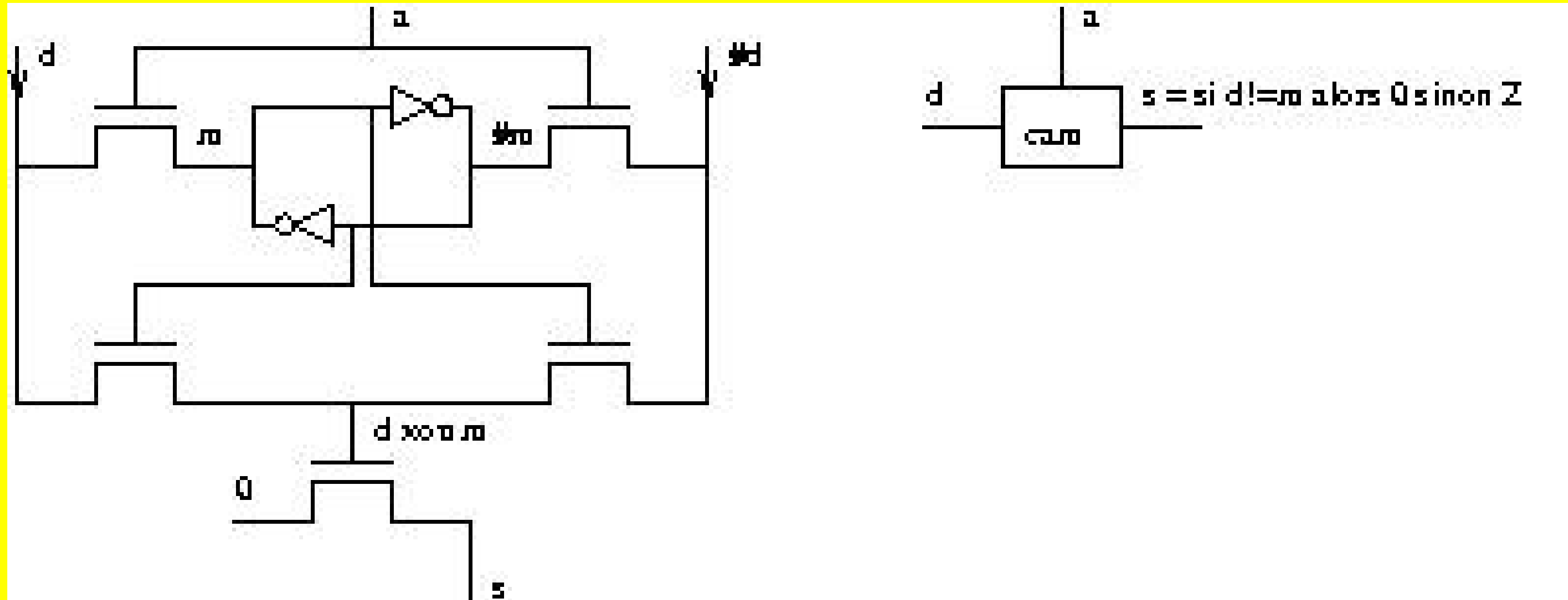
Cellule mémoire DRAM



Lecture: d = Z et a = 1 alors d = q
Ecriture: d = q' et a = 1 alors q = q'
La lecture est destructrice (suivie d'une réécriture)
Quand a = 0, la cellule n'est pas accédée
La charge mémorisée en q disparait progressivement
Le contenu de la cellule doit être rafraichi (lecture puis réécriture)



Cellule mémoire: CAM



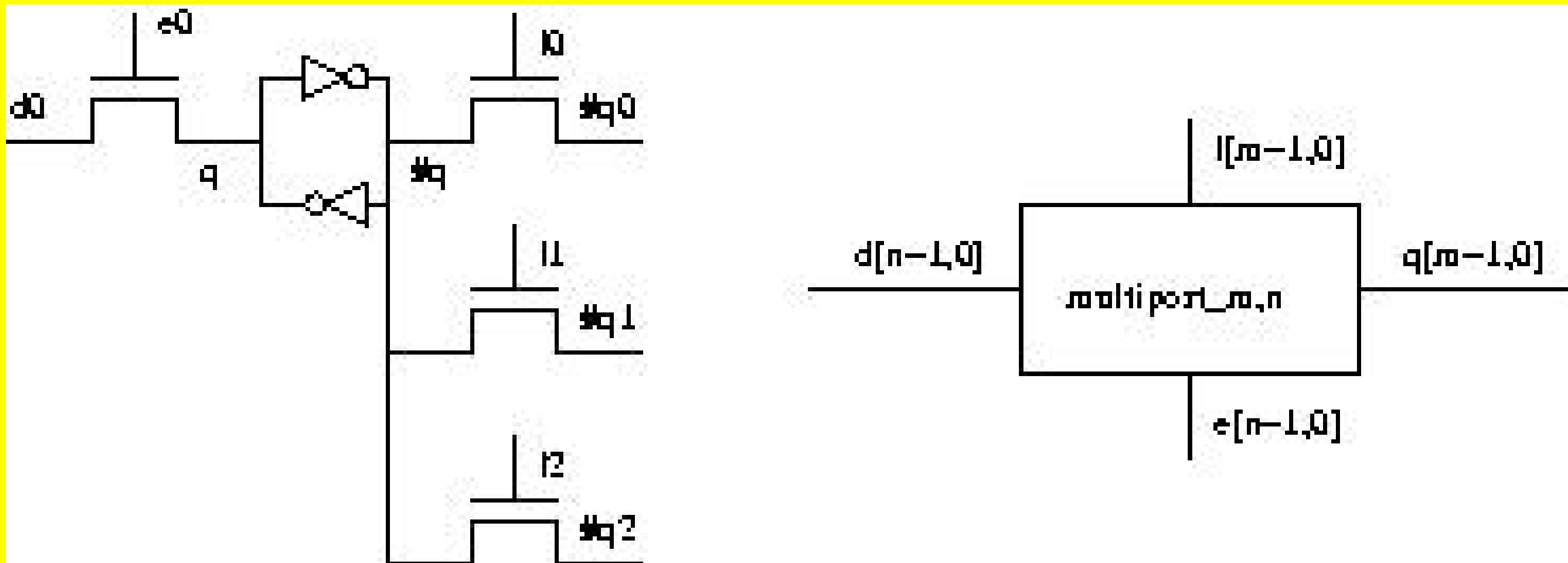
Recherche: a = 0 et d = m’ alors

si m == m’ alors s = Z (succès)

sinon s = 0 (échec)

Remplacement: a = 1 et écriture en SRAM

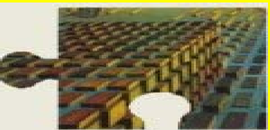
Cellule mémoire: plusieurs ports



Fifo: un port de lecture et un port d'écriture

Registre: n ports de lecture et m ports d'écriture

Attention: la surface de la cellule est proportionnelle au carré du nombre de ports (temps d'accès)



Cellules mémoires

p
d
vd
s = si p alors d
sinon s

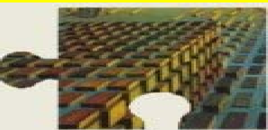
p
d
invd
s = si p alors s
sinon d

p
d
sram
q

p
d
dram

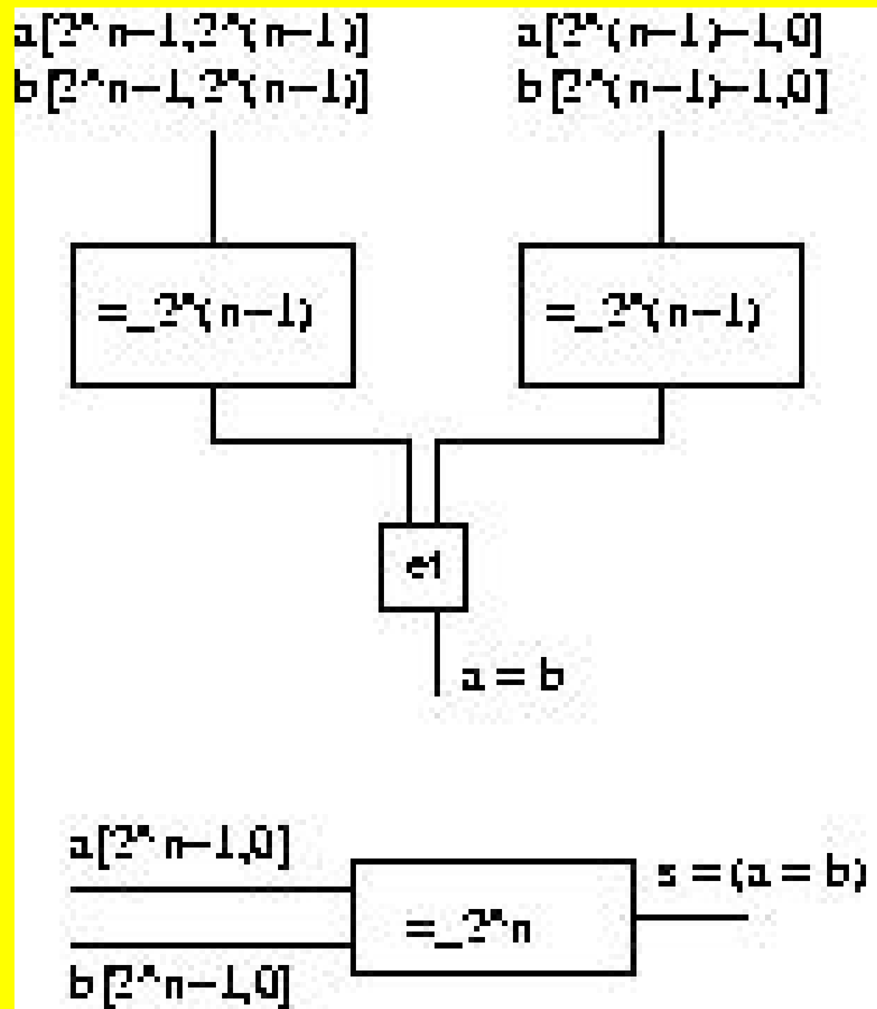
l[m−1,0]
d[n−1,0]
multiport_m,n
q[m−1,0]
e[n−1,0]

e/w
d
cam
s = si d!=m alors 0 sinon Z

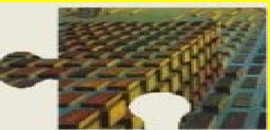


UP VD

Circuit: diviser pour régner

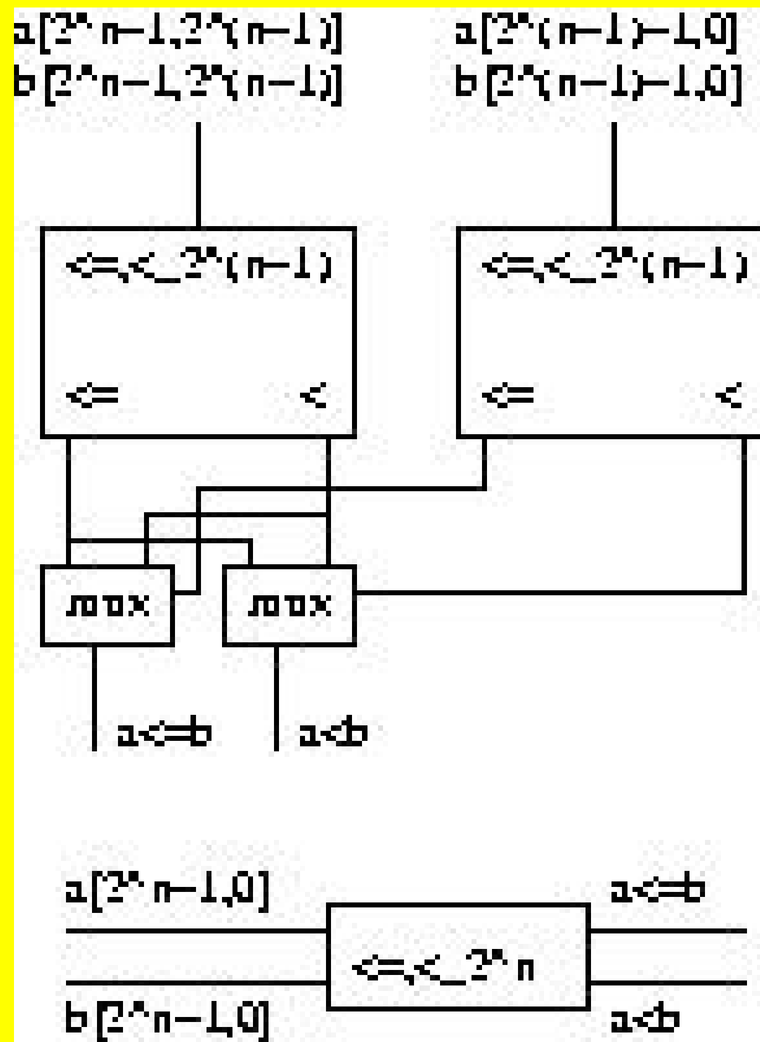


Comparateur d'égalité



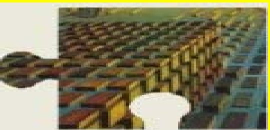
UPVD

Circuit: comparateur d’ordre

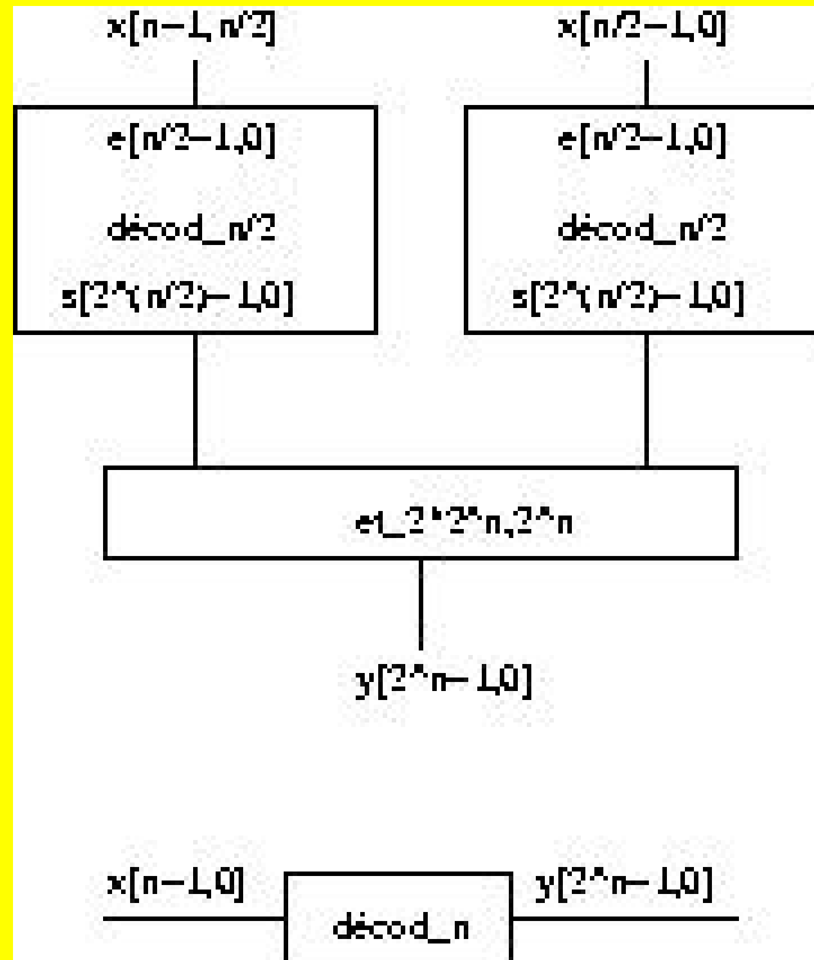


$a < b \Leftrightarrow$ si $ab < bb$ alors $ah \leq bh$ sinon $ah < bh$

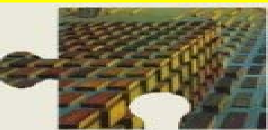
$a \leq b \Leftrightarrow$ si $ab \leq bb$ alors $ah \leq bh$ sinon $ah < bh$



Circuit: décodeur (nom –> position, x –> 2^x)



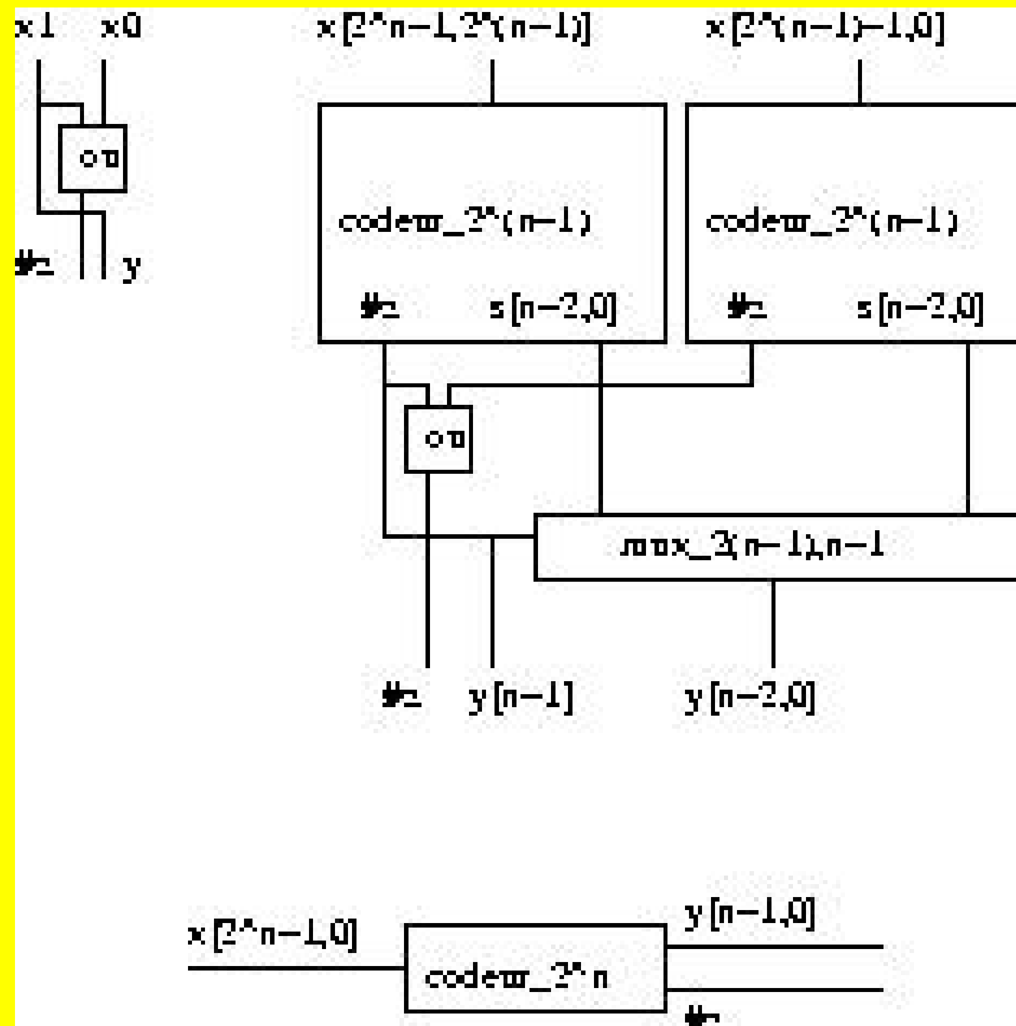
$$y[3,0] = (x_1x_0 , x_1\#x_0 , \#x_1x_0 , \#x_1\#x_0)$$



Circuit: codeur (position –> nom, 2^x –> x)

x=1100
y=11

x=0011
y=01



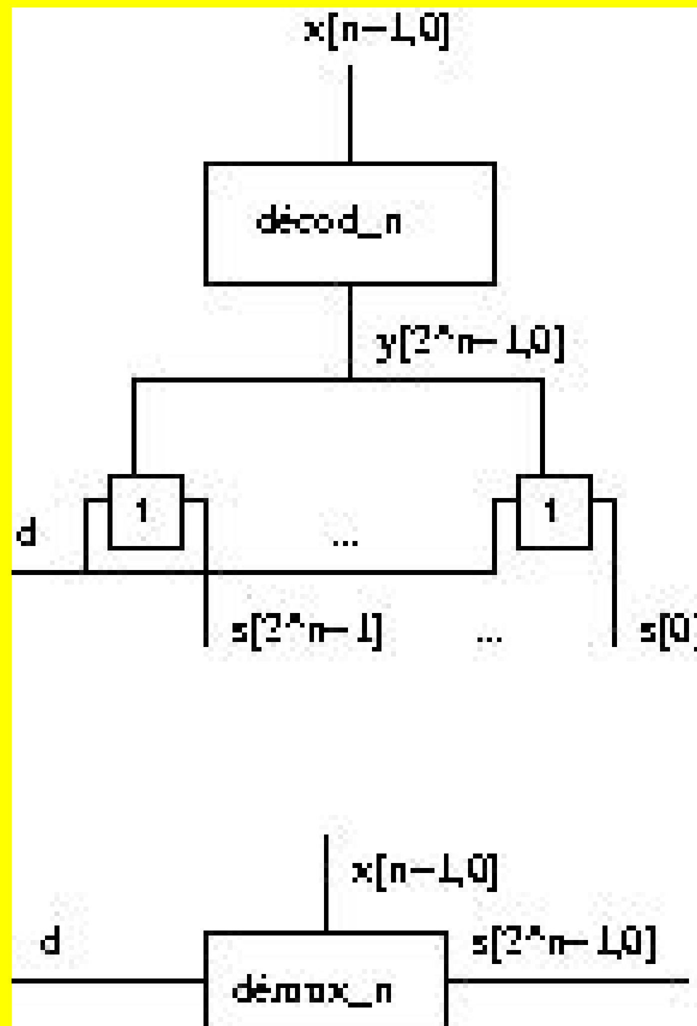
x=0110
y=10

x=0000
#z=0

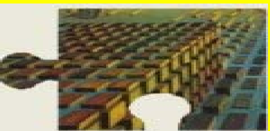
$y_1 = (x_3x_2 \; != 0)$ $\qquad$ $\#z = (x \; != 0)$

y_0 = si $\#z_1$ alors code(x_3x_2) sinon code(x_1x_0)

Circuit: démultiplexeur



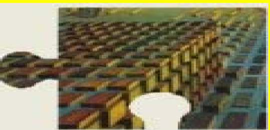
La donnée d est aiguillée vers la voie s_x



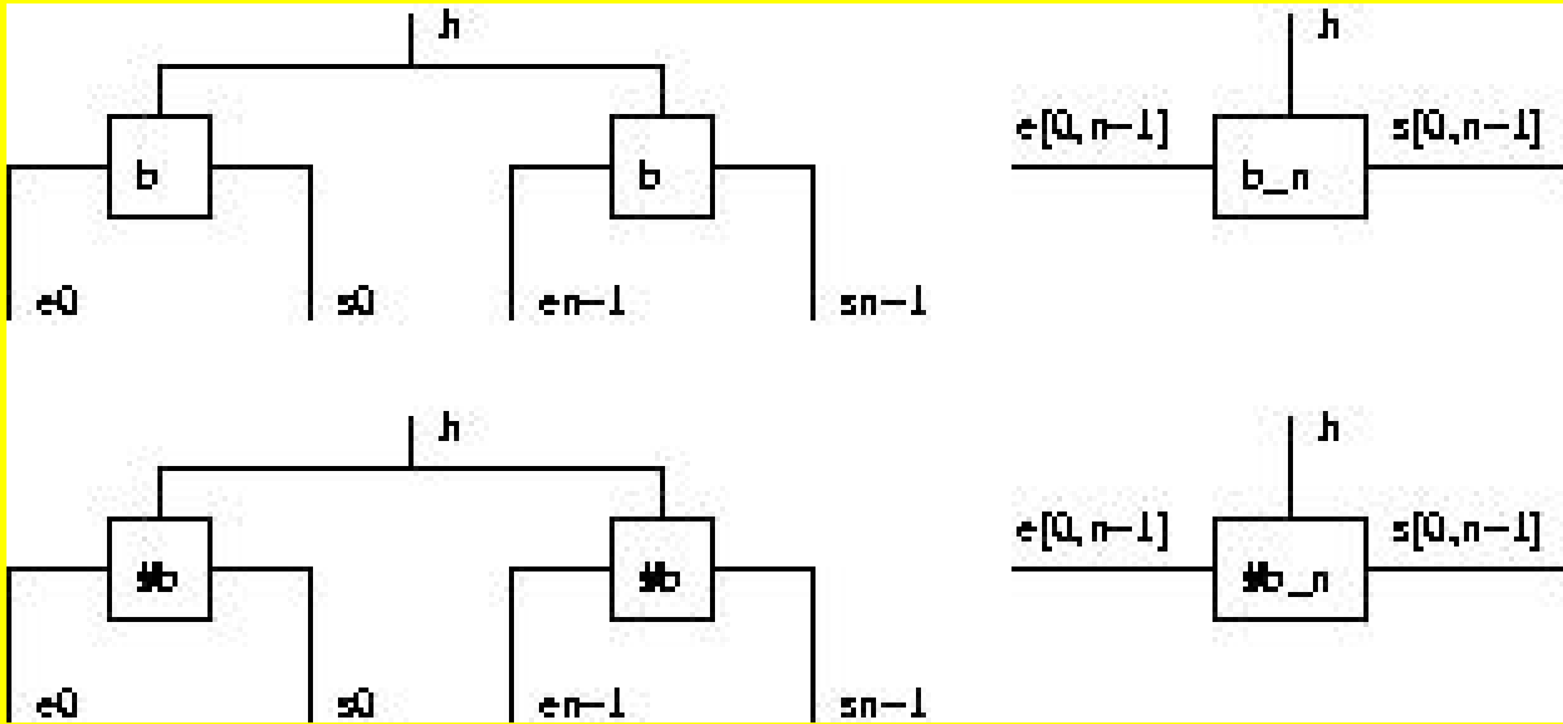
Circuit de séparation asynchrone: le verrou

Pour v: si h = 1 alors s = e

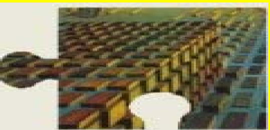
Pour #v: si h = 0 alors s = e



Circuit de séparation synchrone: le registre



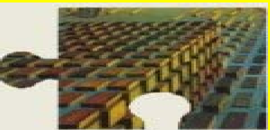
Pour b: quand h monte, e passe en s
Pour #b: quand h descend, e passe en s



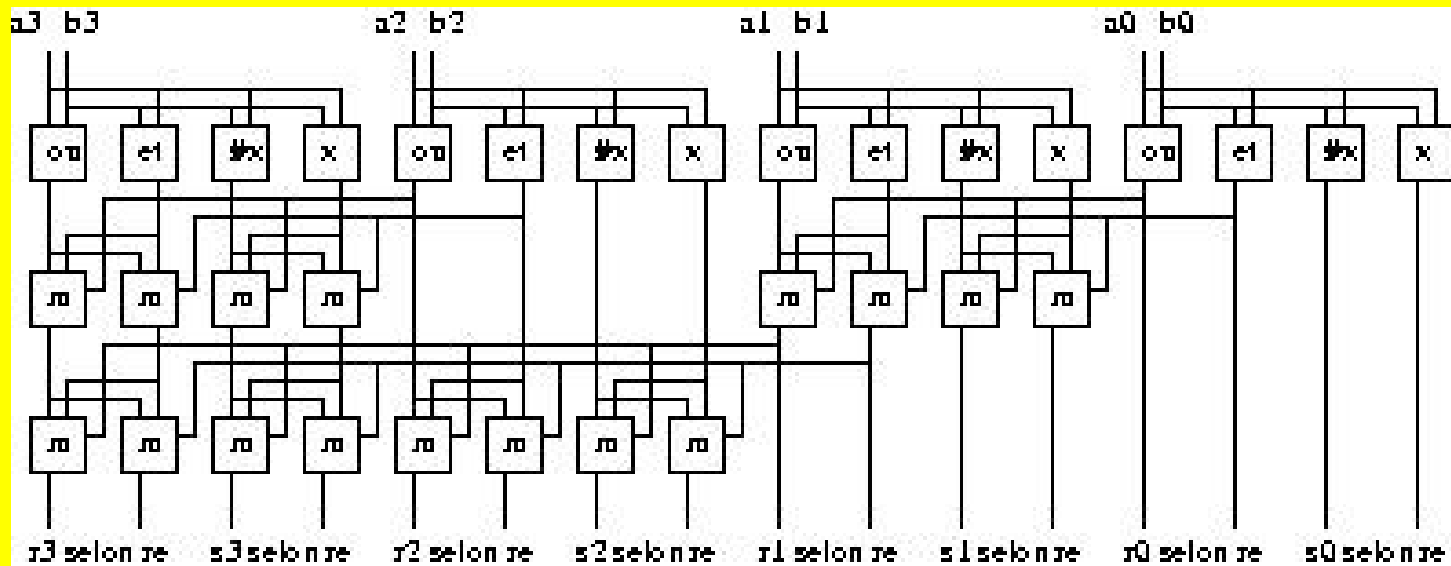
UPVD

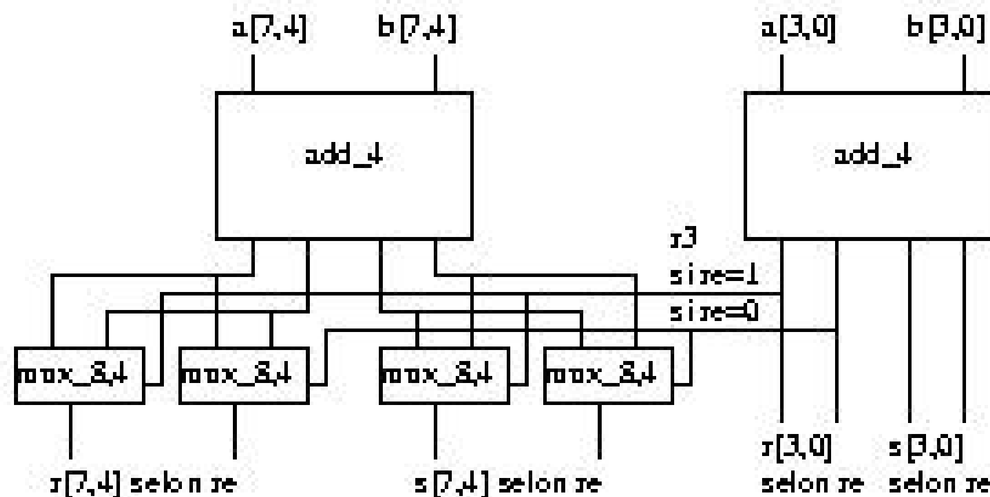
Pipeline

Un traitement peut être subdivisé (pipeliné).
Un morceau (le producteur) est séparé de son successeur (le consommateur) par un registre.
Cela permet d'augmenter la fréquence des traitements.



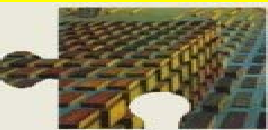
Circuit de calcul: additionneur



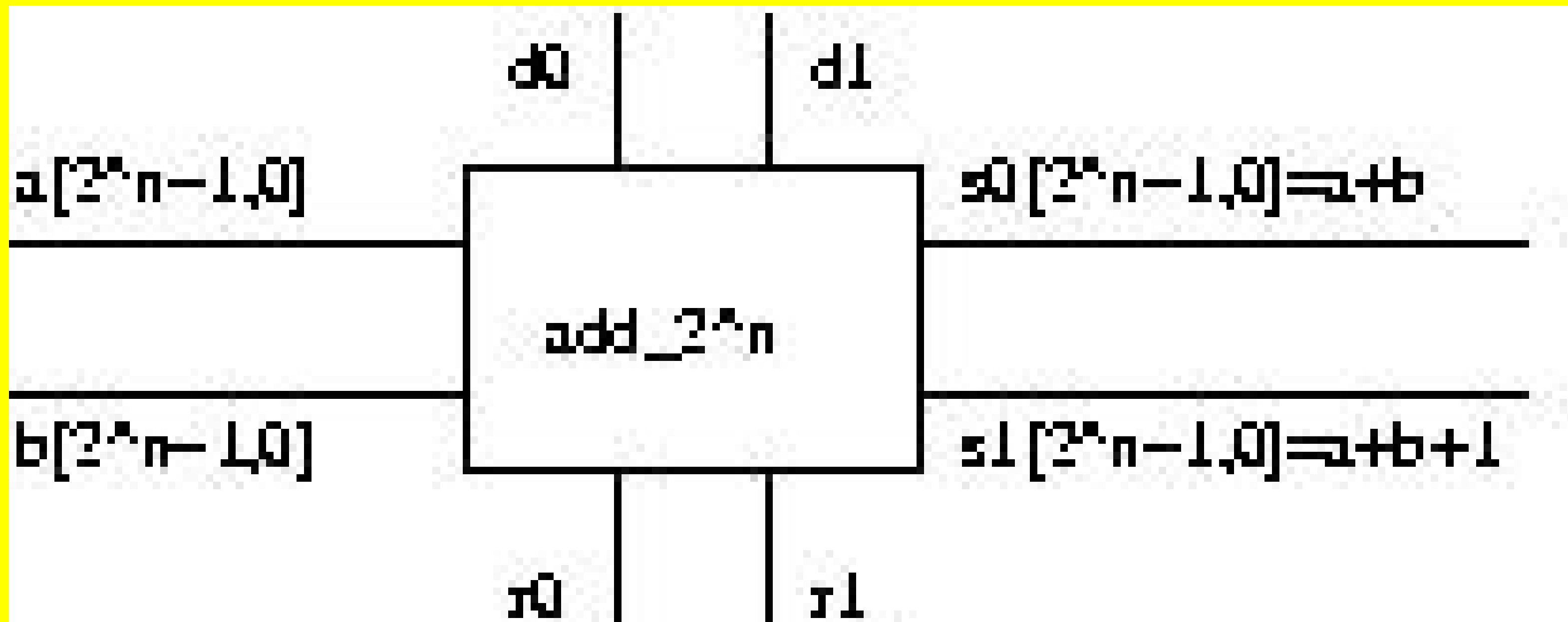


calcule
a+b+0
a+b+1

Additionneur 4 bits, composable pour extension



Circuit de calcul: additionneur



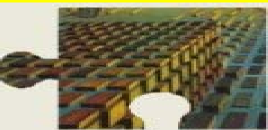
s0 = a + b

s1 = a - #b = a + b + 1

Dans N: r0, retenue de a + b, r1 = (a >= #b)

Dans Z: d0, dépassement de capacité de a + b

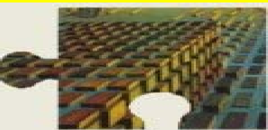
Dans Z: d1, dépassement de capacité de a - #b



Circuit de calcul: décaleur à gauche

$g = r * 2^x$

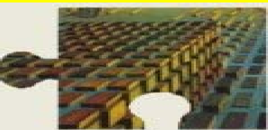
décalage signé ou non selon s/#ns (e.g.: (-1)*2 = -2)



Circuit de calcul: décaleur à droite

$d = r / 2^x$

décalage signé ou non selon s/#ns (e.g.: (-1)/2 = -0,5)



Circuit de calcul: multiplieur

				a3	a2	a1	a0
			*	b3	b2	b1	b0
				a3.b0	a2.b0	a1.b0	a0.b0
+			a3.b1	a2.b1	a1.b1	a0.b1	
+		a3.b2	a2.b2	a1.b2	a0.b2		
+	a3.b3	a2.b3	a1.b3	a0.b3			

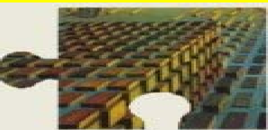
Le produit est une somme de produits partiels
Chaque produit partiel est soit 0 soit a décalé
Produit 32 bits * 32 bits: 31 additions en série!

Circuit de calcul: multiplieur

Une seule addition à retenue

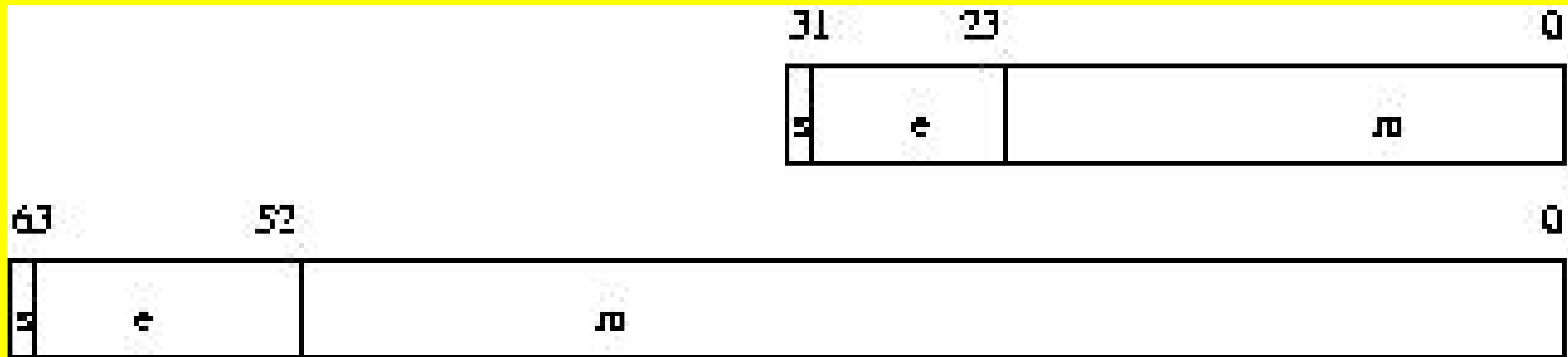
r3,2: réducteur 3 vers 2 ((a, b, c) –> (r, s))

Réduire la hauteur (base élevée redondante)



UPVD

Circuit de calcul: calcul flottant



Norme IEEE 754

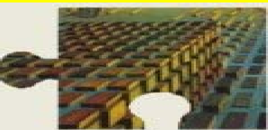
$f = (-1)^{s} * 2^{e-b} * (1 + m/(2^{p}))$

En simple précision, p = 23 et b = 127

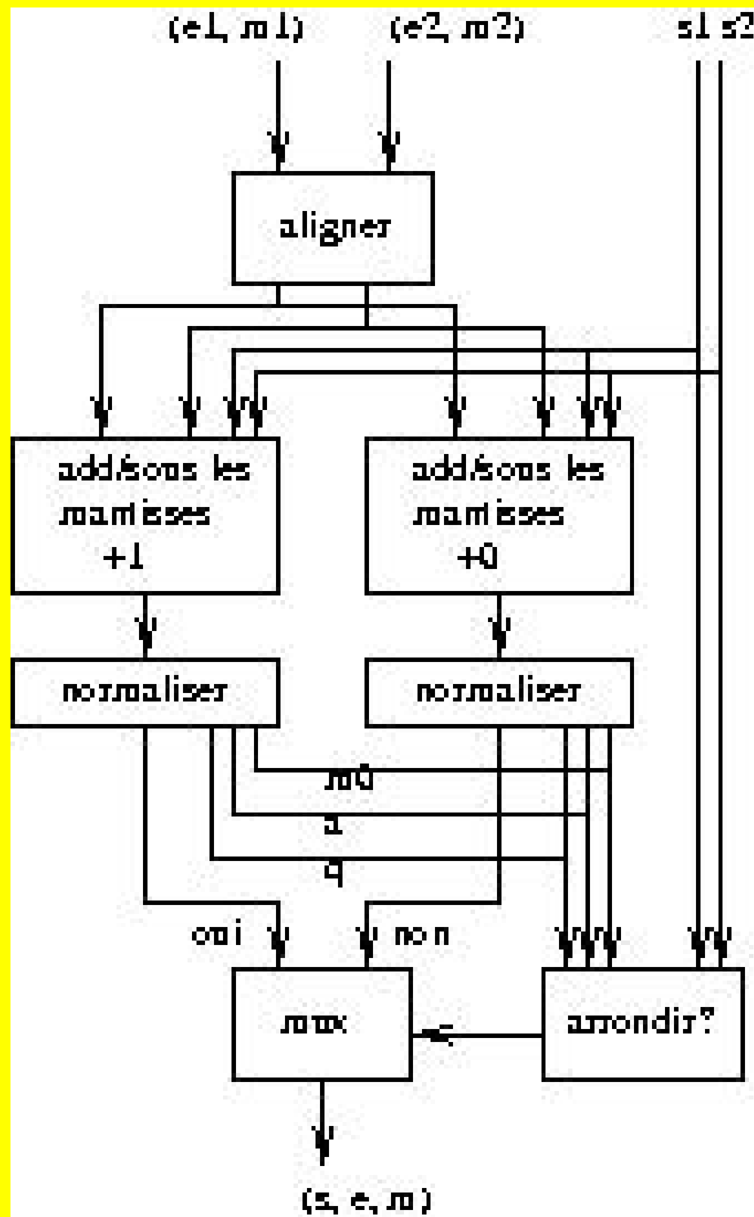
En double précision, p = 52 et b = 1023

(b est le biais; ainsi (e1,m1) < (e2,m2) ⇔ f1<f2)

Circuits de calcul spéciaux



Circuit de calcul: addition flottante



(1) alignement mantisses

(2) m1 + m2, m1 + m2 + 1

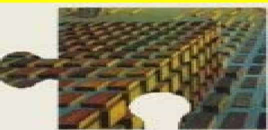
(3) normaliser e, m, e’, m’

(4) s’il faut arrondir:

choisir (s’, e’, m’)

sinon

choisir (s, e, m)



Circuit de calcul: division flottante

E1
E2
début
préparation des mantisses
mux
Q
mux
dividende D
diviseur d
compteur
table de quotients
+1
4D−qd
q+
q−
+
=(p+3)/2
−
reste
quotient
reste
résultat établi
quotient

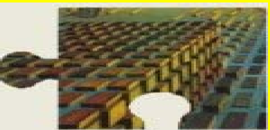
(1) table de quotients
en base redondante

(2) adressée par des
préfixes de D et d

(3) calcul du reste

(4) boucler 13 fois (sp)
ou 28 fois (dp)

(5) q = q+ - q-
corriger r



Unité fonctionnelle: unité entière

TC1

+/-

résultat opérateur 1

décalage à gauche

décalage à droite

TC2

opérations logiques bit à bit

BIT

OCTET

mux

résultat opérateur 2

opération

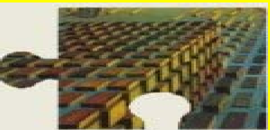
Deux opérations en //

TC: tables d'opérandes

Opérateur: 2 sources

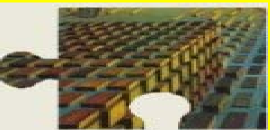
et un code d'opération

Sélection en sortie



Unité fonctionnelle: unité flottante

Deux opérations en //
(arith, conversion)
Opérateur: 2 sources
et une opération
Sélection en sortie



Circuit de mémoire SRAM

64 bits en 32*2

a[4,2]: ligne

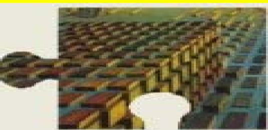
a[1,0]: mot

écrire = 0: t_2 fermés

écrire = 1: d aiguillé

lecture permanente

maintenir a tant que écrire = 1



Circuit de mémoire deux ports

adresse d'écriture
ae[m−1]
ae[0]
d[2^b−1,0]
écrire
dup_n
décod log(n)
et_2^n,n
w[n−1]
w[0]
t_2^b
t_2^b
décod log(l)
pe
Matrice de l*c cellules
pl
d
multiport_1,1
q
pe
décod log(l)
pl
décod log(n)
t_2^b
t_2^b
q[2^b−1,0]
al[m−1]
al[0]
adresse de lecture

écrire
al[m−1,0]
d[2^b−1,0]
ae[m−1,0]
sram2_2^m,2^b
q[2^b−1,0]

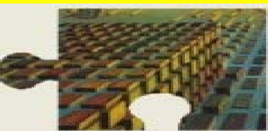
Ports l et e séparés
VRAM
FIFO

Circuit de mémoire cache

e/#r = 1: écriture; e/#r = 0: recherche

#pre: préchargement

e_i = 0: échec en ligne i; e != 0: succès



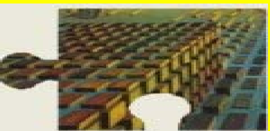
Cache totalement associatif

mémoire associative | mémoire adressable

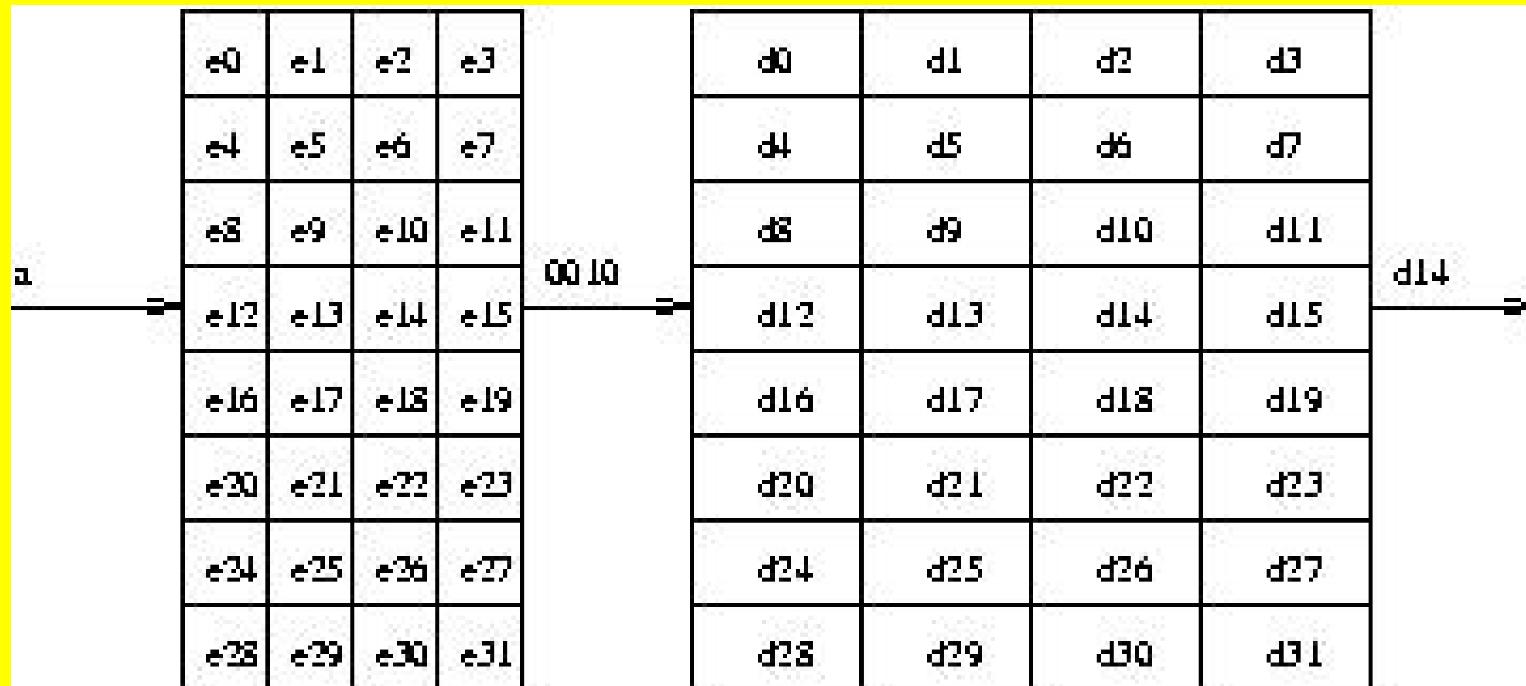
a2 → a0 a1 a2 a3 a4 a5 a6 a7 → 0 0 1 0 0 0 0 0 → d0 d1 d2 d3 d4 d5 d6 d7 → d2

On cherche la donnée d'adresse a2 ; la donnée d2 est lue

Une CAM contient les étiquettes
Une RAM contient les données cachées
La recherche se fait dans toutes les entrées en //



Cache associatif de degré 4

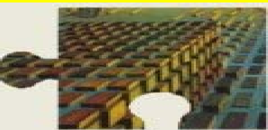


La recherche se fait dans l'ensemble adressé

Associativité de degré 1, 2, 4 ou 8

Degré != 1: choix de l'entrée remplacée

Degré 1: conflits fréquents dans les petits caches



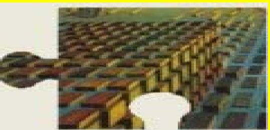
UP VD

Banc de registres

écrire
1_2^b
rd[2^b-1,0]
rd[0]
rd[2^b-1]
décod m
i[m-1,0]
multiport 3,1
registre R0
multiport 3,1
décod m
j[m-1,0]
décod m
d[m-1,0]
multiport 3,1
registre R2^m-1
multiport 3,1
décod m
k[m-1,0]
lire
1_2^b
1_2^b
1_2^b
ri[2^b-1,0]
rj[2^b-1,0]
rk[2^b-1,0]

i[m-1,0]
j[m-1,0]
k[m-1,0]
d[m-1,0]
ri[2^b-1,0]
rj[2^b-1,0]
rd[2^b-1,0]
reg_2^m,2^b
rk[2^b-1,0]
lire
écrire

- 2^m registres de 2^b bits
- 3 ports de lecture (i, j, k)
- 1 port d'écriture (d)

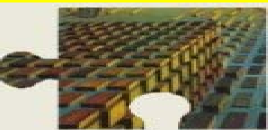


Exemple de processeur élémentaire

Instruction: mot de contrôle (op, d, j, k)

h = 1: lecture instruction

h = 0: écriture résultat



Instruction Rx = Rx op Rx

dd, dj, dk : décodeurs 4 vers 16

accès au bit b du registre r (j = k = d = r)

h = 1: port d fermé, ports j et k passants

h = 0: ports j et k fermés, port d passant

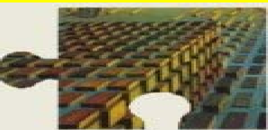
Architecture: ISA

Instructions de calculs
- booléens (conditions), entiers, flottants
- scalaires, vectoriels

Instructions de transferts
- conditionnels, inconditionnels
- mémoire-registre, registre-registre

Instructions de sauts
- conditionnels, inconditionnels
- immédiats, indirects
- avec lien, sans lien, retours

Instructions systèmes
- appels systèmes, mode privilégié



Instruction

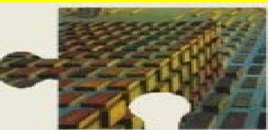
Une instruction contrôle les unités du processeur
(u.f. pour les calculs, hmd pour les transferts mémoire, calcul cp pour les sauts)

Elle adresse ses sources et sa destination
(numéros de registre, adresse mémoire)

Elle paramètre son unité de calcul et en choisit le résultat
(calcul signé ou non, simple ou double précision, + ou -, source constante ou registre, ...)

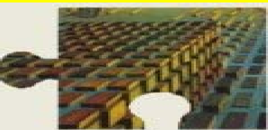
C’est un quadruplet éventuellement conditionnel
([si p] dest = sg opération sd)

Toutes les instructions modifient cp (i suivante)



Codage des instructions

Coder les adresses
(variables en mémoire, cibles de sauts)
Coder les constantes
(initialisation, comparaison, 0, 1, -1, 0.0, 1.0, -1.0)
Coder les registres
(numéro, type)
Coder les opérations
(type, opérateur, modifieur)
Coder le type d'instruction
(calcul, accès mémoire, saut, système)

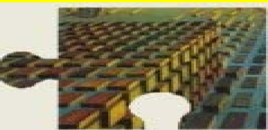


UPVD

Codage des instructions

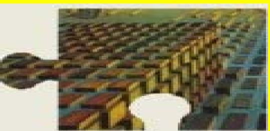
Quatre vertus pour un 'bon code':

- **compact**
- **même taille pour toutes les instructions**
- **même nombre de sources et destinations pour toutes les instructions (en général, 1d et 2s)**
- **sauts évitables, repérables et prédictibles (instructions conditionnelles, type d'instruction, suggestion de prédiction statique)**

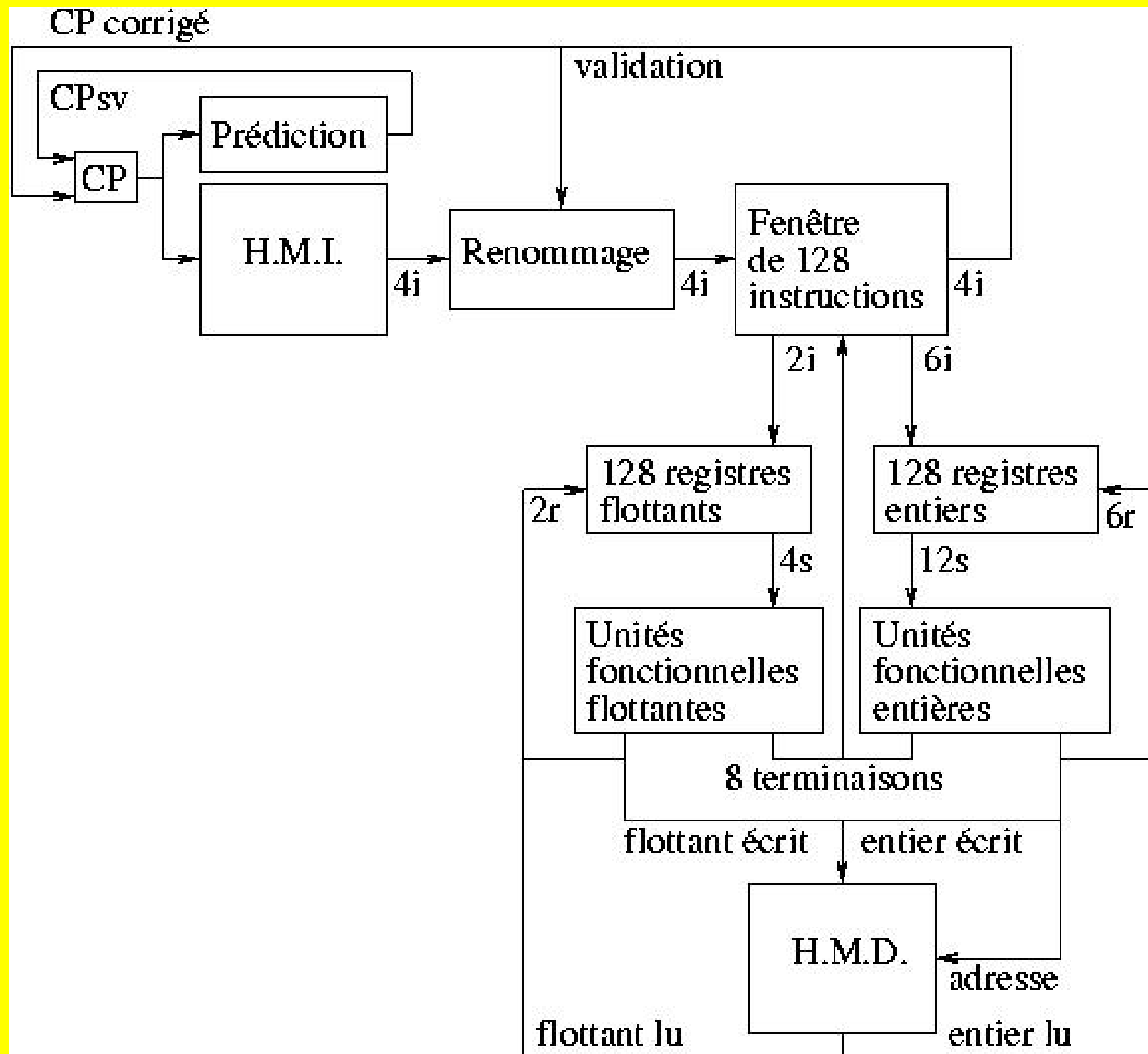


Exemple

```
R1 = R2 + R3          /* calc, int, d1, sg2, sd3, op+ */
R1 = R1 + 1           /* calc, int, d1, sg1, sdk1, op+ */
F1 = F2 + F3          /* calc, fl, d1, sg2, sd3, op+sp */
R1 = M2[R2 + R3]      /* mém, int, d1, sg2, sd3, opl2 */
SI R1<'Z' VERS e      /* saut, int, e, sg1, sdk'Z', op< */
APPEL f               /* saut, int, d31, -, sdkf, opcall */
RETOUR                /* saut, int, -, sg31, -, opret */
VERS R3               /* saut, int, -, sg3, -, opgoto */
APPELSYS  fs          /* saut, sys, rsys, -, sdkfs, opcall */
RETOURSYS             /* saut, sys, -, rsys, -, opret */
SI P (R1 = R2)        /* calc, int, d1, sg2, -, op=cond */
```



Processeur spéculatif ooo de degré 4



UP VD

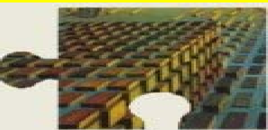
Extraire une ligne de cache par cycle dans une machine spéculative

En examinant les instructions extraites, on calcule cpsv:

pas de saut dans la ligne à partir de cp: cpsv = ligne suivante
le premier saut s à partir de cp est inconditionnel immédiat: cpsv = k(s)
le premier saut s à partir de cp est conditionnel et prédit pris: cpsv = k(s)
le premier saut s1 à partir de cp est conditionnel et prédit non pris: cpsv = s2
(s2 est le second saut dans la même ligne ou à défaut le début de ligne suivante)
le premier saut s1 à partir de cp est un retour: cpsv = pile[sp]
le premier saut s1 à partir de cp est indirect: prédire cpsv

Sans examen des instructions à extraire, on prédit cpsv:

on conserve des couples (saut, cible) dans un cache: le BTB
le BTB doit être accédé en un cycle soit actuellement un cache de 8KO



Extraire une ligne de cache par cycle dans une machine spéculative

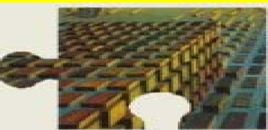
Pour un cache d'instruction (lignes de 2^n instructions):

- un bloc de base (BB) est un groupe d'instructions en séquence tel que:
  - le BB figure tout entier dans une ligne de cache
  - le BB ne comporte qu'au plus un saut
  - si le BB comprend un saut, celui–ci en est la dernière instruction
- un BB compte entre 1 et 2^n instructions

Le BTB est un cache de lignes de 2^n couples (saut, cible):

- un saut d'adresse a entre en colonne (a mod 2^n)
- le BTB peut fournir en parallèle les cibles de tous les sauts d'une ligne de cache

Un BTB de 256 lignes de 4 couples occupe 7Koctets



Extraire une ligne de cache par cycle dans une machine spéculative

correction cp

cp

i–cache L1 (16KO)

aligne

examen
BB extrait

BTB
(7KO)

calcul
cpsv

e/d/n

pdsc

préd

pile

ajustement pile

cpsv

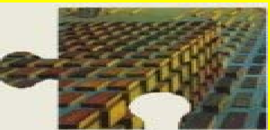
nouvelle ligne de couples (BB, cible)

extraction cycle 1

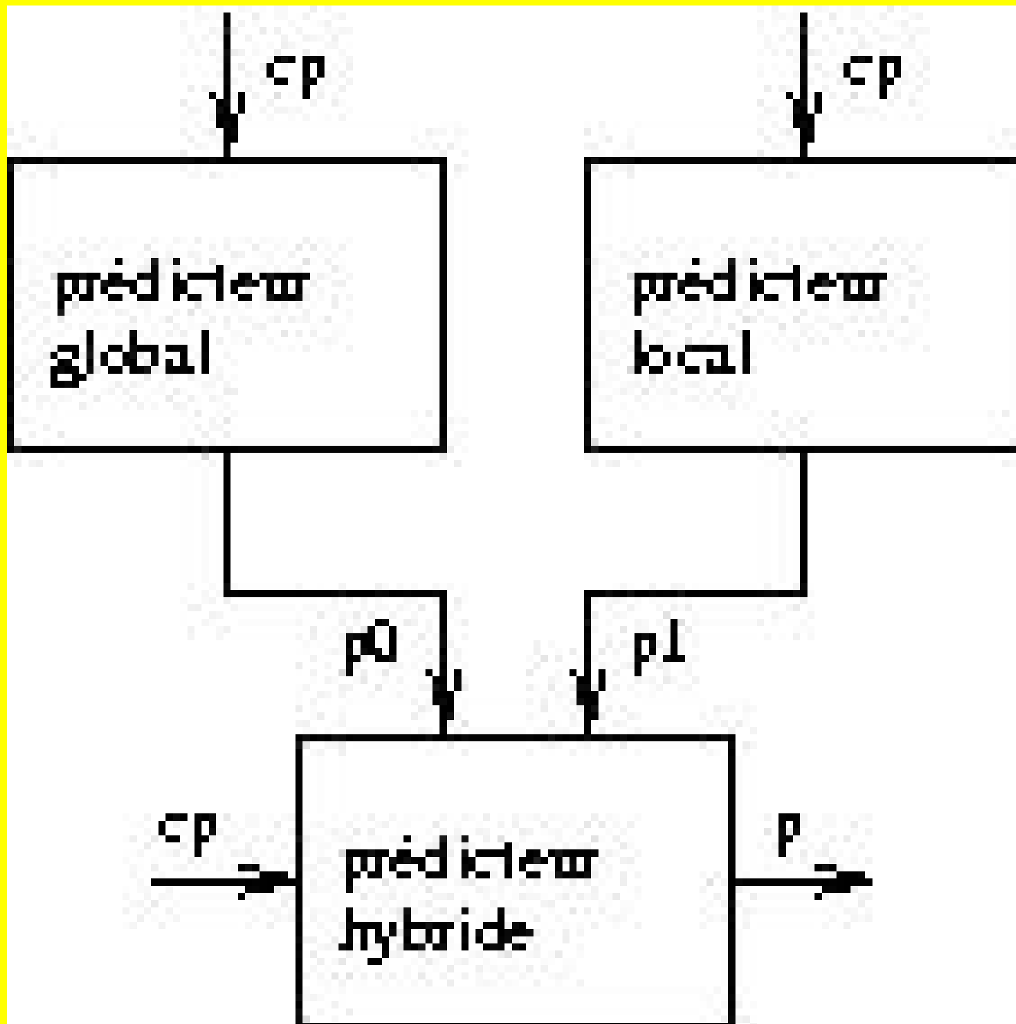
prédiction cpsv

extraction cycle 2

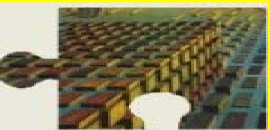
correction cpsv



Prédiction de la direction des sauts conditionnels

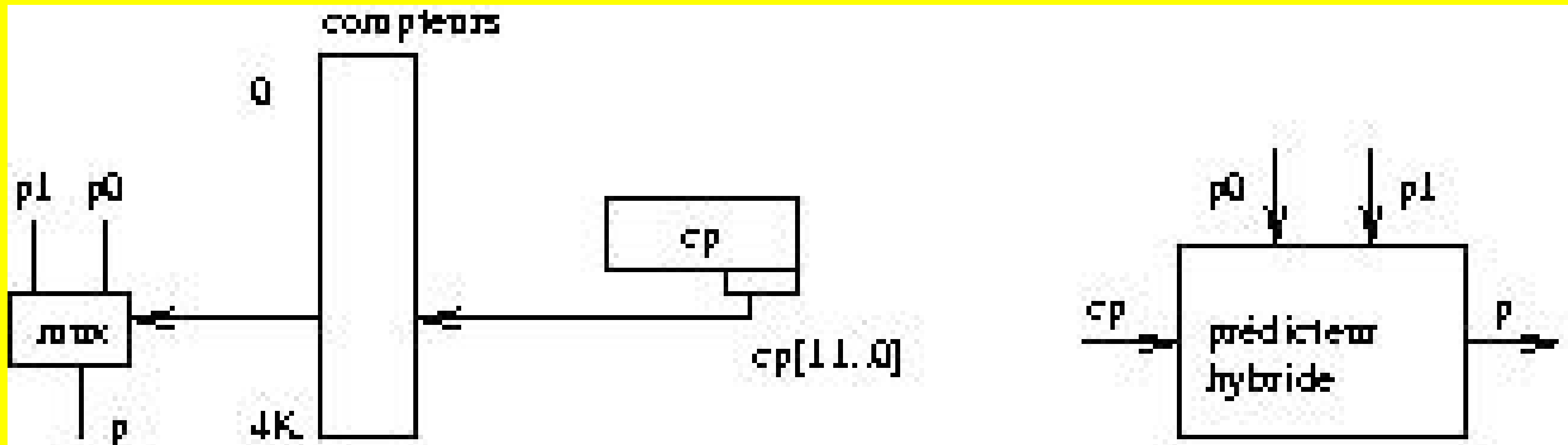


La direction prédite p est choisie parmi deux prédictions issues de prédicteurs spécialisés et basées sur le cp et sur le comportement antérieur des sauts conditionnels



UPVD

Prédicteur hybride



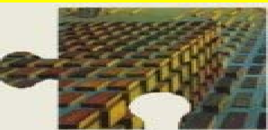
Une table de compteurs 2 bits à saturation adressée par la partie basse de cp

(la table est un cache sans étiquette)

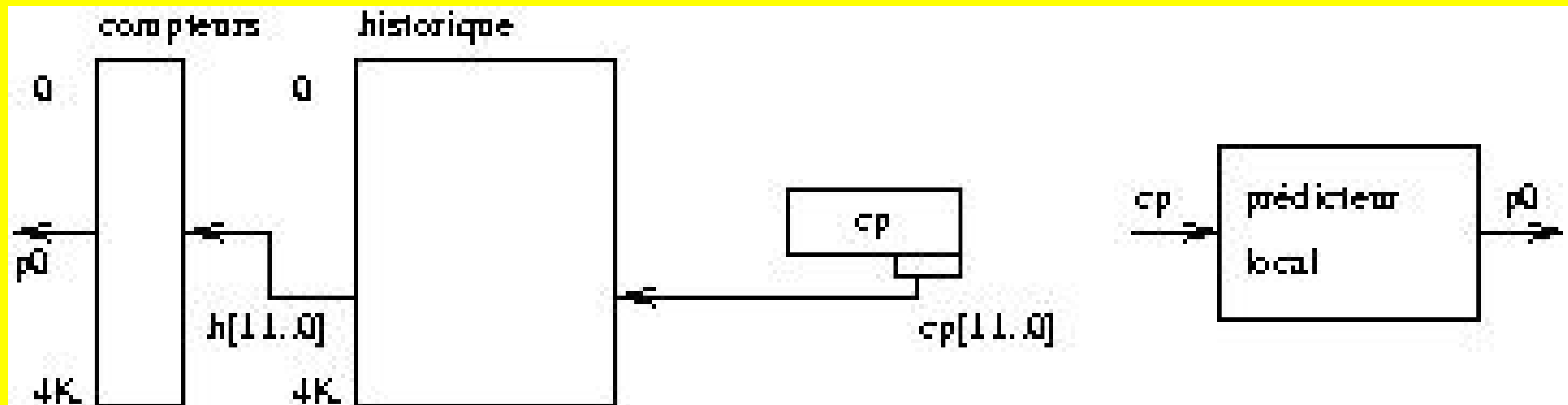
Le bit fort du compteur adressé fixe le choix

compteur++ ⇔ p0 != p1 et p1 correcte

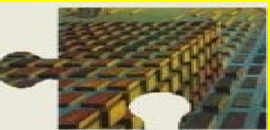
compteur-- ⇔ p0 != p1 et p0 correcte



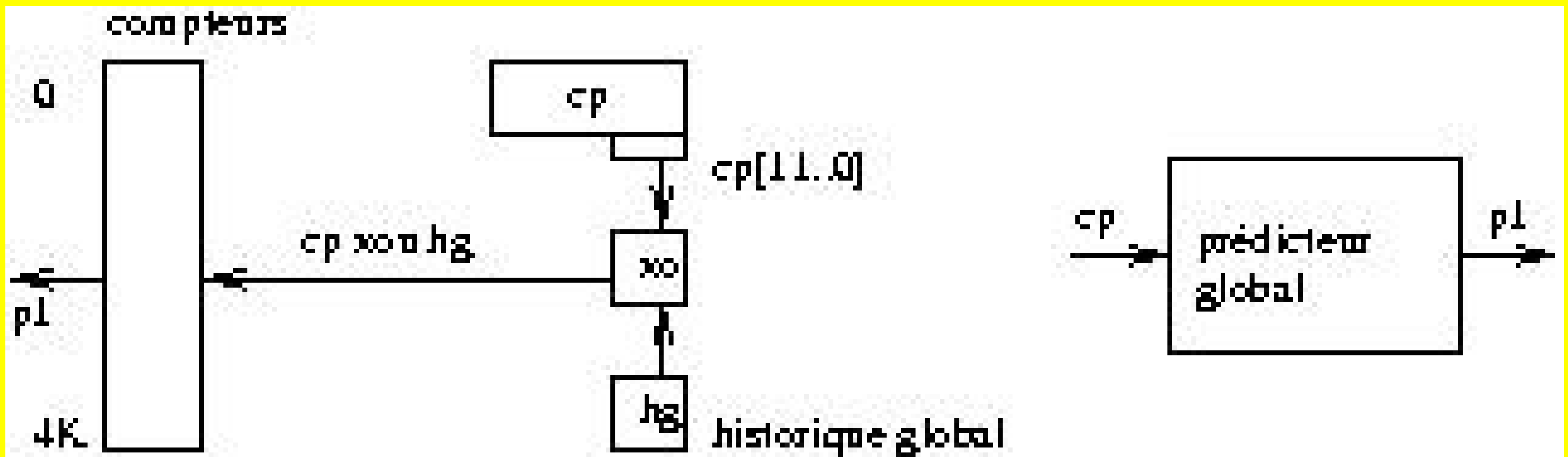
Prédicteur local



- L'historique est un mot de 12 bits correspondant aux 12 dernières directions d'un saut
- Le motif obtenu adresse un cache de compteurs
- Le bit fort issu du cache est la prédiction
  - compteur++/-- ⇔ saut pris/saut non pris
- Le prédicteur local prédit bien les sorties de boucles



Prédicteur global

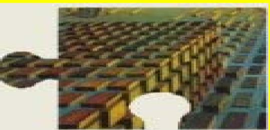


L'historique global est un mot de 12 bits formé des directions des 12 derniers sauts conditionnels

Le mélange de cp et de l'historique adresse un cache de compteurs deux bits à saturation

Le bit fort du compteur adressé est la direction prédite compteur++/-- ⇔ saut pris/saut non pris

Le prédicteur global prédit bien les sauts corrélés

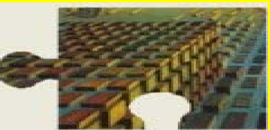


Prédiction des retours

sp retour
—
mux
sp
retour
mux
ni appel
ni retour
si la condition commandant
le multiplexeur est vraie,
la voie du haut est choisie
++
sp appel
adresse
d'écriture
adresse
de lecture
type = appel
type = retour
adresse de
retour
cp+1
prédicteur
des retours
15
0
cp+1
retour

Le prédicteur adresse une pile (mémoire à 2 ports)

On empile à chaque appel, on dépile à chaque retour (attention: pas de détection de débordement)



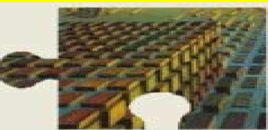
Extraire une ligne de cache par cycle dans une machine spéculative

La pile matérielle est mise à jour au premier cycle:

si le BTB indique que le BB extrait est terminé par un retour: dépilement
si le BTB indique que le BB extrait est terminé par un appel: empilement
(on empile cp + longueur(BB))
autrement, la pile est inchangée

La pile matérielle est corrigée au second cycle:

si le BB extrait contient un retour non signalé par le BTB: dépilement
(l'adresse dépilée est envoyée pour corriger le cp)
si le BB extrait contient un appel non signalé par le BTB: empilement
(on empile l'adresse de l'instruction qui suit l'appel en séquence)
(on envoie l'adresse appelée pour corriger le cp)
(pour un appel indirect, la pile est corrigée mais pas le cp)



Extraire une ligne de cache de trace par cycle dans une machine spéculative

Dans une ligne de cache de 4i, il y a en général un seul BB

Pour extraire plus d'un BB à la fois, il faut:

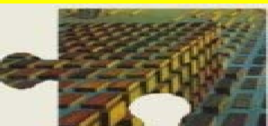
soit lire plusieurs ligne à la fois, ce qui implique un cache multi–banc
soit disposer de plusieurs BB dans une ligne, c'est–à–dire un cache de trace

un cache multi–banc ne permet pas de lire deux lignes dans le même banc
(ce qui est le cas quand on doit lire deux fois la même ligne)

Un cache de trace contient des traces

Une trace est une juxtaposition de BB

trois sauts conditionnels au plus (prédicteur multi–saut)
2^n instructions au plus (taille de la ligne de cache)



Extraire une ligne de cache de trace par cycle dans une machine spéculative

Contraintes supplémentaires sur les traces:

un **BB** (terminé par un saut ou une fin de ligne) ne peut être coupé
(il entre tout entier dans la trace de 2^n instructions ou dans la suivante)

deux traces du cache ne peuvent avoir le même préfixe
(si la trace AB est cachée, la trace AC ne l'est pas)

Constitution des traces:

les traces sont construites à la fin de l'extraction à partir des **BB** prédits

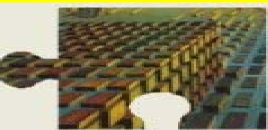
un **BB** extrait est ajouté en queue de la trace en cours de construction ssi:

- la nouvelle trace a au plus 2^n instructions
- la nouvelle trace a au plus 3 sauts immédiats

dans le cas contraire, la trace ancienne est écrite dans le cache

et le **BB** extrait entre dans une nouvelle trace vierge

(sauf s'il se termine par un saut indirect: dans ce cas, il est écarté)

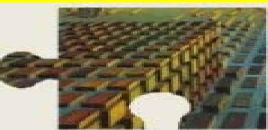


Extraire une ligne de cache de trace par cycle dans une machine spéculative

Constituants d'une trace:

- l'adresse de son BB de départ (index et étiquette dans le cache)
- la direction des trois sauts immédiats de la trace ("pris" pour un saut inconditionnel)
- l'adresse de l'instruction suivante (dernier saut pris)
- l'adresse de l'instruction suivante (dernier saut non pris)
- le nombre de sauts immédiats de la trace
- l'indication que la trace est ou n'est pas terminée par un saut immédiat

Un cache de 64 traces occupe 4,7Koctets



Fonctionnement de l'extracteur basé sur un cache de trace

une trace est extraite du cache à l'adresse (cp mod taille)

un prédicteur multi–saut délivre une prédiction de la direction
des trois prochains sauts immédiats

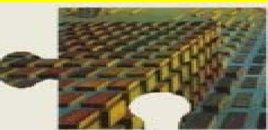
ces prédictions sont confrontées aux directions des sauts immédiats
de la trace extraite (à concurrence du nombre de tels sauts dans la trace)

l'accès au cache est un succès ssi:

- l'étiquette de la ligne lue est cp/taille
- les directions des saut immédiats coincident avec les prédictions

en cas d'échec, un extracteur standard fournit le BB suivant et le cpsv

en cas de succès, le cpsv est fourni par le cache de trace



Extraire une ligne de cache de trace par cycle dans une machine spéculative

Cas d’échec:

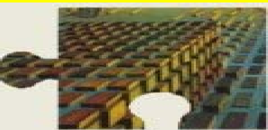
saut indirect ou retour
(les sauts indirects sont prédits par l’extracteur standard)
succès partiel (le cache fournit un préfixe du chemin prédit)
échec d’étiquette (nouveau chemin)

Alternatives:

Associativité des adresses
Associativité des traces (AB et AC dans le cache)
Succès partiel (le cache doit conserver les cibles des sauts intermédiaires)

Le cache de trace est incompatible avec le BTB et la pile:

on ne dispose pas des adresses des BB internes à une trace
on ne connaît pas le nombre d’appels internes à la trace



Extraire une ligne de cache de trace par cycle dans une machine spéculative

trace lue

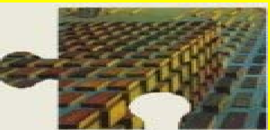
cp

cpsv si succès

cache de trace

prédicteur
multi–saut

prédiction de la direction
des 3 prochains
sauts conditionnels



Extraire une ligne de cache de trace par cycle dans une machine spéculative

cp

nouvelle trace

préd

BTB

pile

cache L1

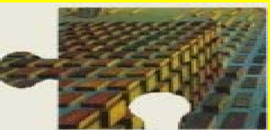
cache de trace

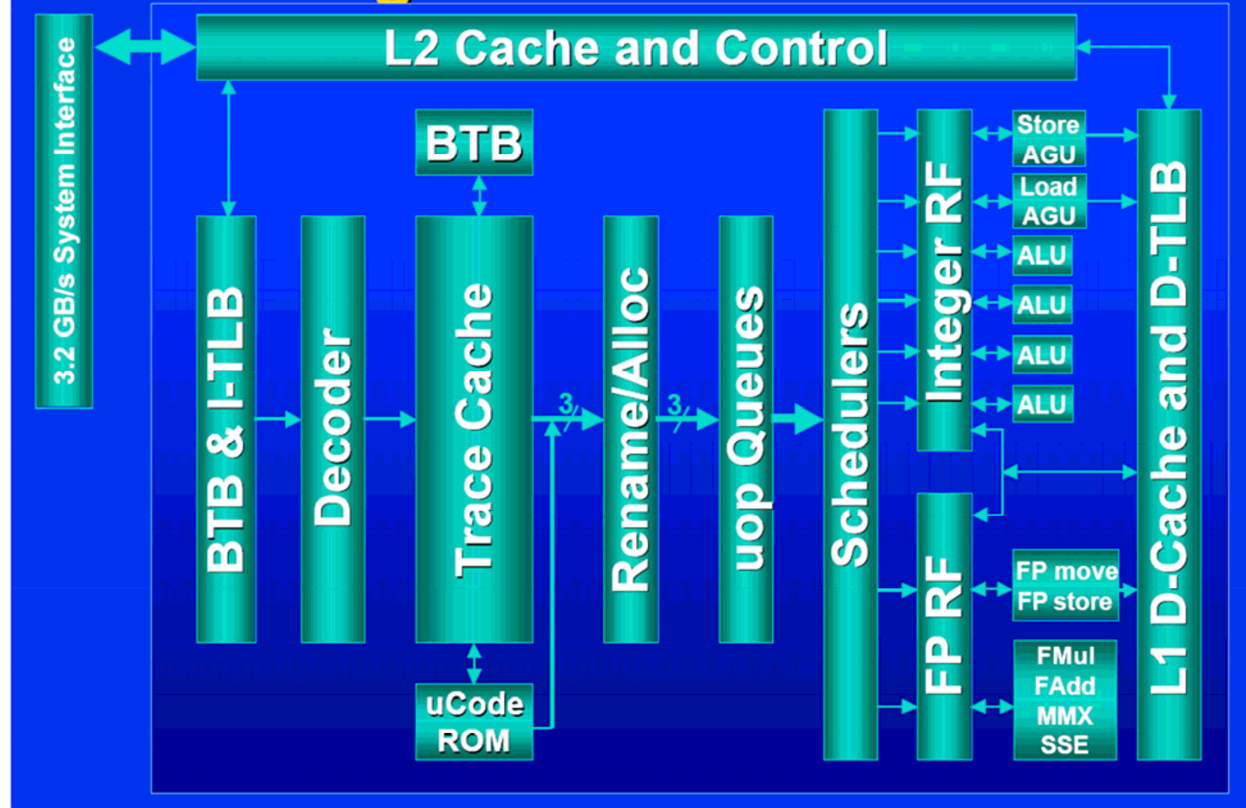
p0,p1,p2

cible 1er saut prédit pris

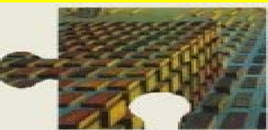
cpsv

instructions à décoder





Le cache de trace du P4 contient des blocs de micro-instructions, mais pas de traces contenant des sauts prédits.



UPVD

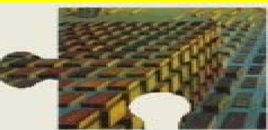
Réordonnancement dynamique

```
racine( unsigned x) {           /* x dans R1 */
int i = –1;                     R0 = –1
unsigned s = 0;                 R9 = 0
while (s<x) {                   si R0>= R1 vers f
   i+=2;                    e:  R0 = 2 + R0
   s+=i;                        R9 = R9 + R0
  }                             si R0 < R1 vers e
return (i/2+1);             f:  R0 = R0 / 2
                                R0 = R0 + 1
}                               RET
                                /* résultat dans R0 */
```

On sait extraire les instructions en //

Peut-on les exécuter en //?

Il faut construire l'ordre partiel des instructions



Dépendances LAE, EAE et EAL

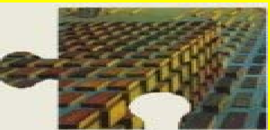
code source

```
      /* x dans R1 */
      R0 = –1
      R9 = 0
      si R0>= R1 vers f
e:    R0 = 2 + R0
      R9 = R9 + R0
      si R0 < R1 vers e
f:    R0 = R0 / 2
      R0 = R0 + 1
      RET
      /* résultat dans R0 */
```

fragment d’exécution

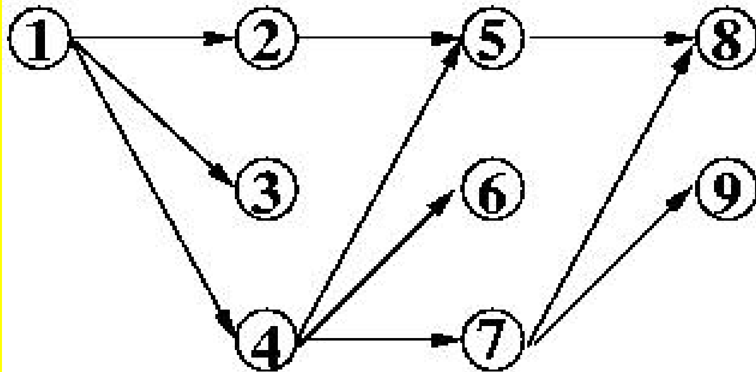
```
R0 = 2 + R0
R9 = R9 + R0
si R0 < R1 vers e
R0 = 2 + R0
R9 = R9 + R0
si R0 < R1 vers e
R0 = 2 + R0
R9 = R9 + R0
si R0 < R1 vers e
```

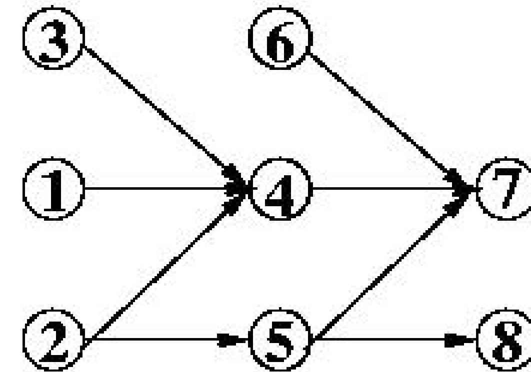
Dépendance LAE: 2 dépend de 1
Dépendance EAE: 4 dépend de 1
Dépendance EAL: 4 dépend de 3

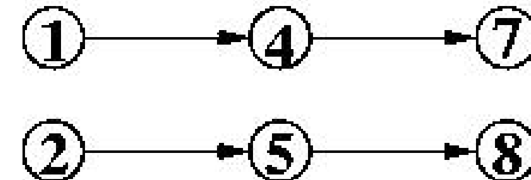


UPVD

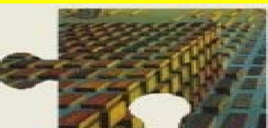
Graphe des dépendances LAE, EAE et EAL







Les dépendances EAE et EAL sont de fausses dépendances. On les élimine en dédoublant le registre de destination: c'est le renommage.

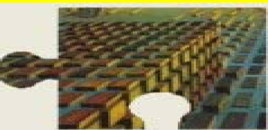


UP VD

Renommage des registres

fragment d'exécution	fragment renommé
R0 = 2 + R0	RR10 = 2 + RR00
R9 = R9 + R0	RR19 = RR09 + RR10
si R0 < R1 vers e	si RR10 < RR01 vers e
R0 = 2 + R0	RR20 = 2 + RR10
R9 = R9 + R0	RR29 = RR19 + RR20
si R0 < R1 vers e	si RR20 < RR01 vers e
R0 = 2 + R0	RR30 = 2 + RR20
R9 = R9 + R0	RR39 = RR29 + RR30
si R0 < R1 vers e	si RR30 < RR01 vers e

Pour matérialiser les r registres de l'architecture on dispose de q (q>r) registres de renommage
Chaque destination est associée à un registre de renommage disponible (allocation)

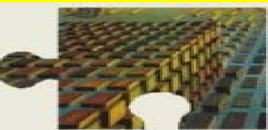


UP VD

Le renommage permet d'éliminer les fausses dépendances (dépendances EAE et EAL)
pour ne conserver que l'ordre partiel d'exécution imposé par les vraies dépendances (dépendances LAE)

Les destinations EAE et EAL sont dédoublées, ce qui permet une exécution dans un ordre quelconque

L'association des sources avec la destination dont elles dépendent (LAE) doit se faire après son renommage

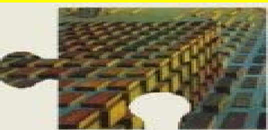


Renommage

Une table RAT associe chaque registre architectural à son renommage le plus récent.

Pour n instructions extraites et renommées solidairement (ligne extraite):

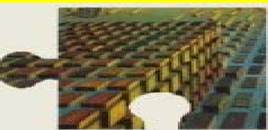
- Renommage des destinations: on alloue n registres pris dans une liste de registres libres. La table RAT est mise à jour.
- Renommage des sources (dépendance intra–ligne): comparer les deux sources de l'instruction i aux (i–1) destinations des instructions précédentes.
- Renommage des sources (dépendance inter–ligne): la table RAT donne le nom du renommage le plus récent.



Renommage et validation:

Renommage du Pentium 4 (NetBurst): pointeurs

Renommage de l'instruction: F-RAT[d] = RR alloué
Validation de l'instruction: R-RAT[d] = RR validé



4 résultats

4

TM

accès mémoire

TS

saut

TC1

calcul

TC2

calcul long

instruction renommée

RR spéc.

lec

t

r

v

correction

RR non spéc.

v

file des validations

4 terminaisons

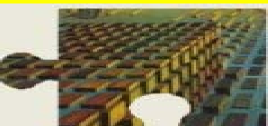
lec: lecture des sources (lecture du banc spéculatif)
r: restauration des registres (copie du banc non spéculatif dans le banc spéculatif)
t: terminaison (écriture de résultat dans le banc spéculatif)
v: validation (report de résultat du banc spéculatif dans le banc non spéculatif)

Après renommage:

- **lecture en RR des sources prêtes**
- **allocation d'une station d'attente**
- **réception des résultats LAE**
- **lancement**
- **exécution pipelinée**
- **envoi du résultat**

Après exécution:

- **validation en ordre**



aucune entrée libre

numéro de l'entrée à écrire

instruction renommée à stocker

priorité

table à 8 entrées

état	(libre/occupé)						
contenu							
condition	(prête/pas prête)						

priorité

instruction prête à exécuter

aucune instruction prête

numéro de l'entrée à lire

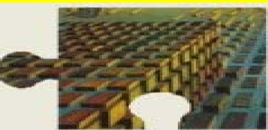
Dans chaque table de stations d'attente, une instruction parmi celles qui sont prêtes est choisie pour démarrer son exécution.

fenêtre de stations d'attente

g d g d g d g d g d g d g d g d

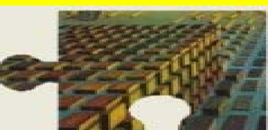
uf0 uf1 uf2 uf3

Chaque résultat est distribué à toutes les stations d'attente et copié partout où une dépendance LAE est détectée

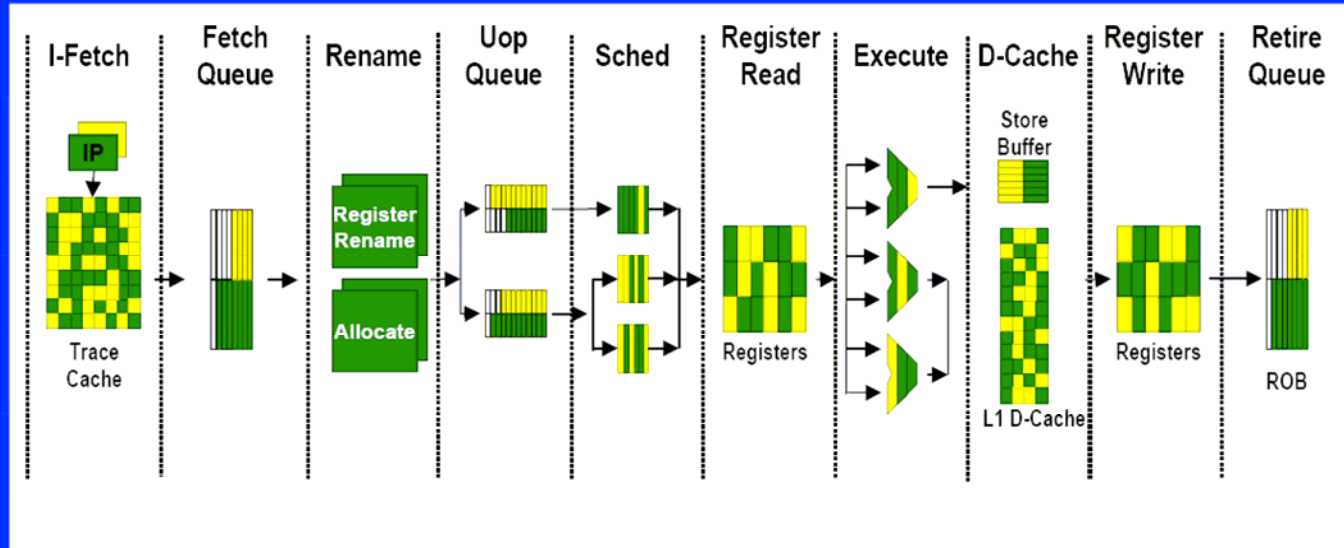


RR nsp
validation
mémoire d'instruction
tables de prédiction
tables de renommage
RR sp
file locale de validation
extraction
prédiction
renommage
transport
RR sp
exécution
TC1
RR sp
exécution
exécution
TC2
RR sp
exécution
TS
RR sp
exécution
exécution
TM

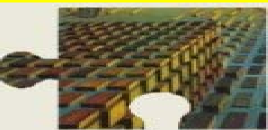
Le traitement des instructions est pipeliné



Multithreading SMT

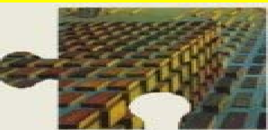


Deux IP, deux BdR-arch, files doubles.

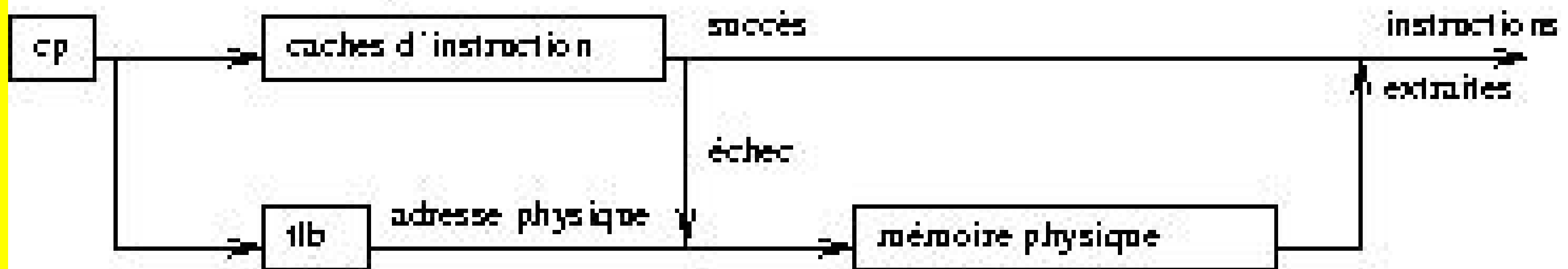


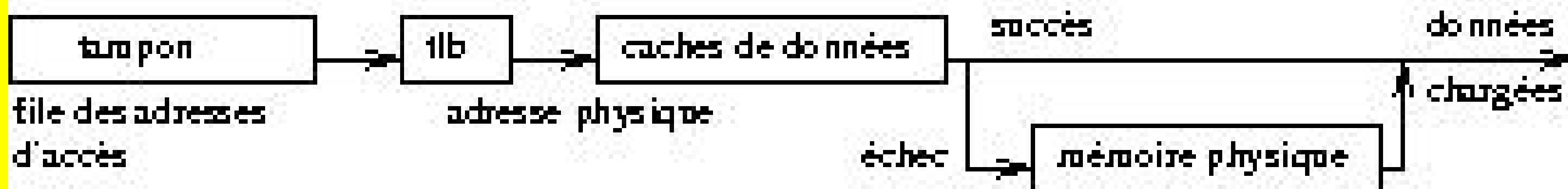
Fonctionnement SMT

Deux modes d'exécution: simple et double thread.
En mode simple, le thread actif utilise toutes les ressources.
En mode double, les files sont partitionnées et un thread ne peut empiéter sur sur l'autre.
L'extraction se fait à tour de rôle tant que les deux threads sont demandeurs.
Quand un thread est en attente, l'autre peut utiliser sa moitié des ressources.
Les caches sont partagés.

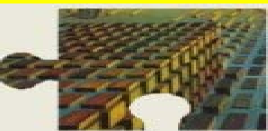


Les accès à la mémoire





Les adresses manipulées par le processeur sont celles produites par le compilateur: ce sont des adresses virtuelles à traduire en adresses physiques. Les caches peuvent contenir des adresses virtuelles ou des adresses physiques.



UPVD

table des régions

table des segments

table des pages

r s p d

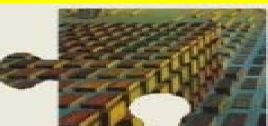
2	3	1	6

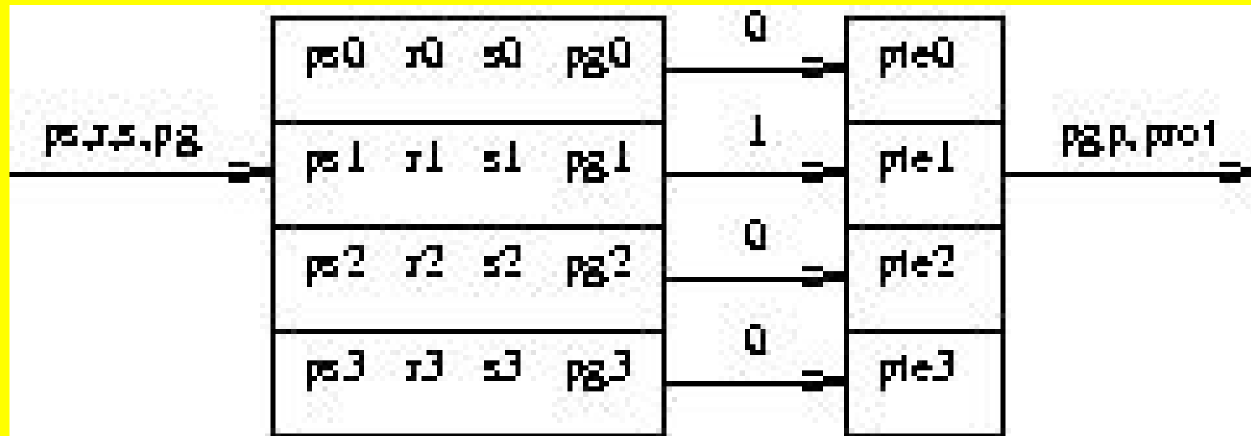
adresse virtuelle

8	6

adresse physique

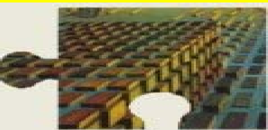
La traduction d’adresse est un parcours en mémoire





on recherche l'adresse virtuelle a=(ps1,r1,s1,pg1,d1); le TLB retourne pte1=(pgp,prot)
pgp est le numéro de page physique
prot sont les bits rwxrwx de protection de la page en mode utilisateur et système

Le TLB est un cache totalement associatif
Il cache des couples (av, ar)
Il est accédé en moins d'un cycle



cache L1 (instructions)

coeur

cache L1 (données)

cache L2 (unifié)

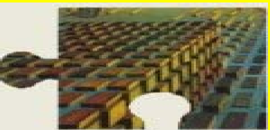
SRAM

mémoire centrale

DRAM

disque

La hiérarchie mémoire se compose de caches (SRAM), de mémoire (DDR2) et du disque.
Tailles: 16K, 1M, 1G, 200G
Temps d’accès: 0,5ns, 2ns, 45ns/2ns, 5ms
Rapport Fmém/Fcycle: 2, 8, 180/8, 20.10^6

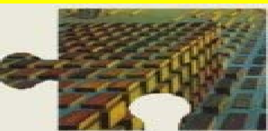


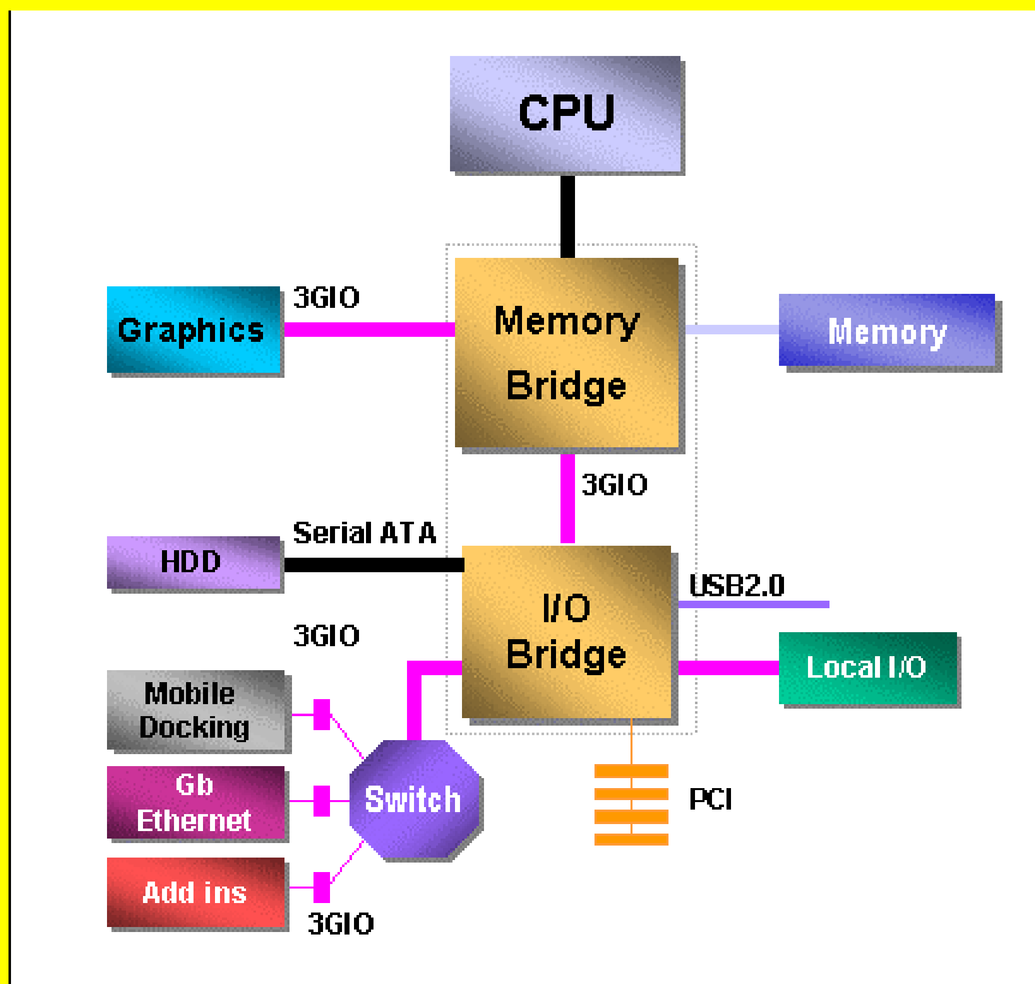
cache externe
bus privé
processeur
bus système
contrôleur
mémoire
bus rapide
périphériques rapides
contrôleur
bus lent
périphériques lents

Les E/S sont mappées en mémoire

Les contrôleurs assurent l'interface avec les organes périphériques et le processeur

Les échanges se font par interruption





FSB: 800Mhz (6,4GO/s)

AGP 8x: 133Mhz (2,1GO/s) ATA: async (133MO/s)

PCI express: 2,5Ghz (2,5Gb/s * 1 à 32 bits)

USB2: 480Mhz (48MO/s) PCI: 266Mhz (2,1GO/s)