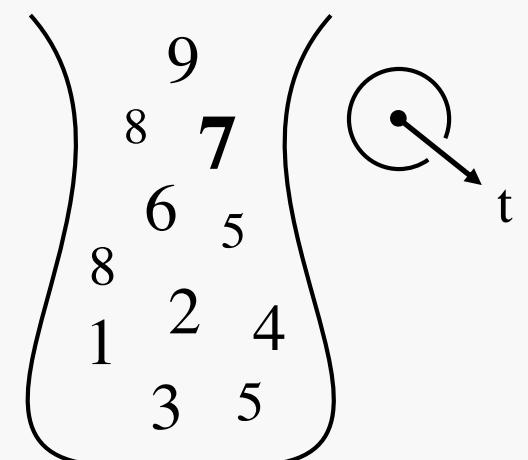


# CALCUL NUMÉRIQUE SÉQUENTIEL

ES102 / CM7



# RETOUR SUR L'ADDITIONNEUR BIT-SÉRIE

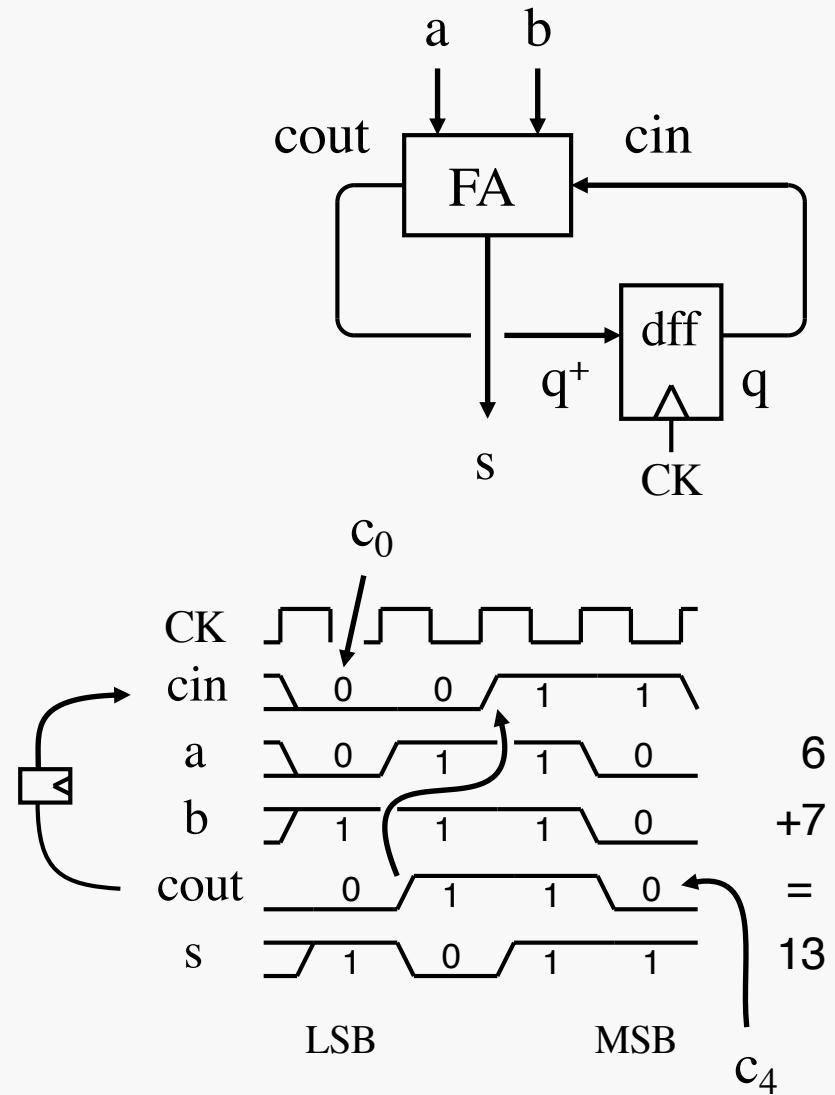
- Vu du CM6, c'est un *système dynamique discret* (concept d'importance pratique bientôt évidente)
  - signaux d'entrée  $a$  et  $b$
  - état  $q = \text{cin}$
  - état futur  $q^+ = \text{cout}$
  - signal de sortie  $s$ 
    - de type Mealy...
- Exemple d'utilisation (séquencé par CK)



- Motivation initiale (CM5) *algorithmique* : *utilisation itérative d'une même ressource de calcul* (ici un simple FA) *pour réaliser un calcul plus complexe* (ici une simple addition numérique)

idée intéressante à étendre :

- à des nombres, au lieu de simples bits
- à de multiples ressources de calcul



# CALCUL (NUMÉRIQUE) SÉQUENTIEL

- Exploitation d'un jeu limité mais polyvalent de ressources de calcul combinatoires (numériques typiquement)
- pour réaliser des calculs plus complexes,
- chaque ressource étant exploitable une fois par période d'horloge, entre tops d'horloge successifs
- grâce à des bascules D multi-bit (alias vectorielles), souvent numériques (c'est-à-dire portant un nombre entier) qui savent mémoriser le résultat d'une opération pour en faire un opérande pour une ou plusieurs opérations à venir.  
par transfert, au(x) prochain(s) top(s) d'horloge



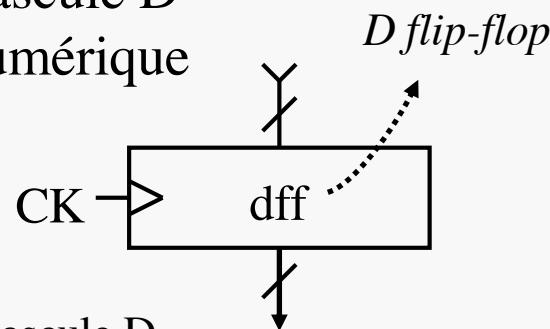
→ Fondement de l'*architecture* d'un microprocesseur, entre autres



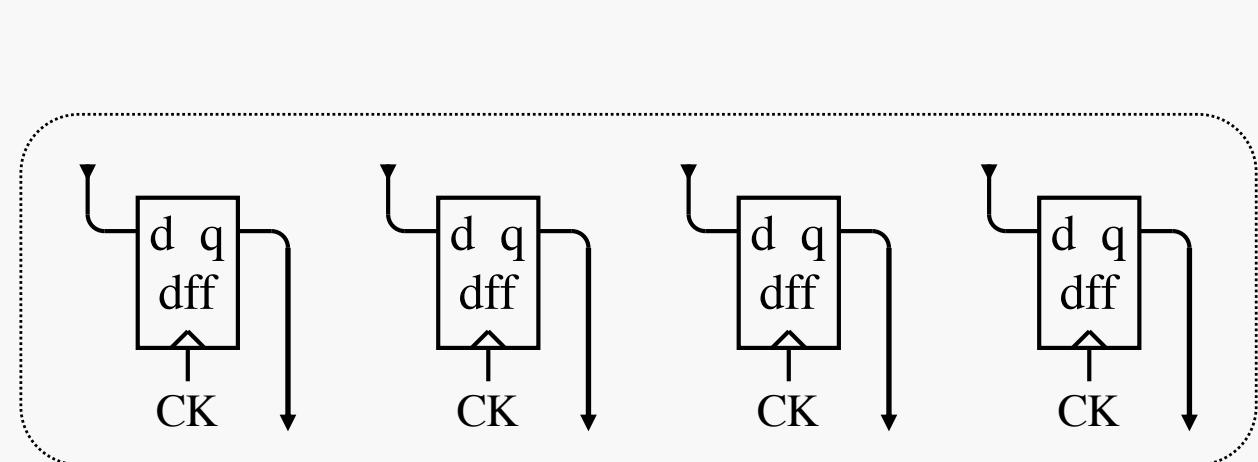
Quelques primitives usuelles sur les 4 planches suivantes

# NOMBRES EN BASCULE

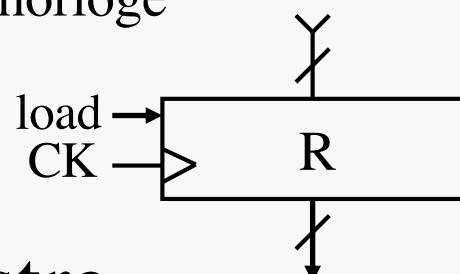
Bascule D  
numérique



Bascule D  
multi-bit portant un nombre  
 $\Rightarrow$  symbole pivoté de 90°

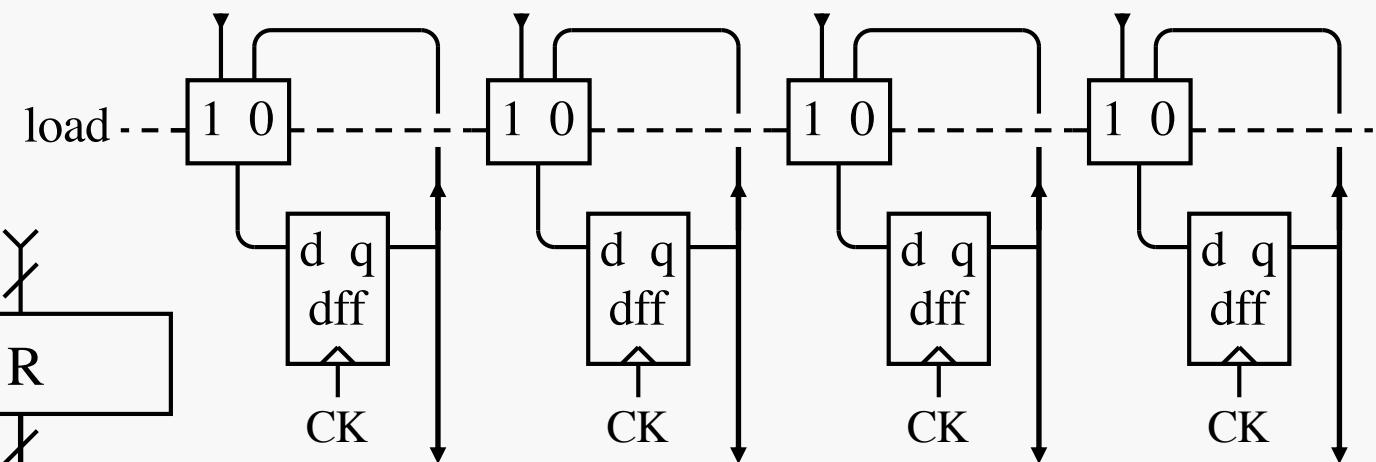


Souvent besoin de  
conserver un nombre  
pendant plusieurs  
cycles d'horloge



Registre

si  $load=1$ , chargement d'une nouvelle donnée  
si  $load=0$ , conservation de la donnée stockée



} au top  
d'horloge

36t / tranche

(12t pour MUX + 24t pour dff)

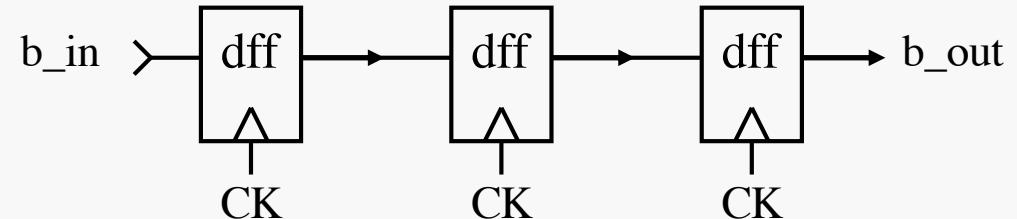
$\$R$  = contenu d'un registre R

# REGISTRE À DÉCALAGE

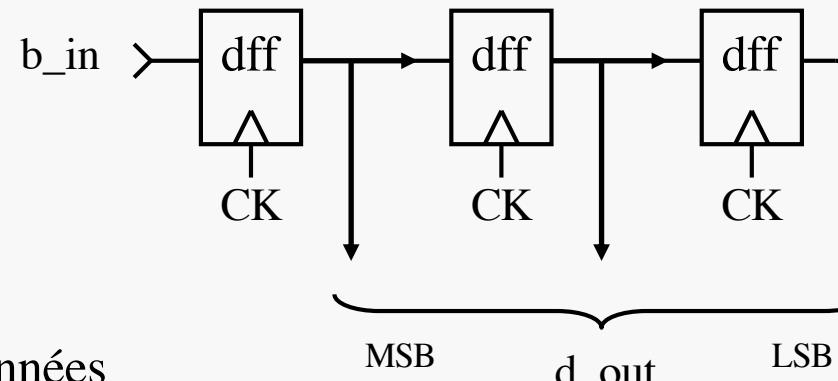


→ PC6/Exo1

- Ce n'est pas un registre (!)
- mais une bascule D multi-bit à chargement et déchargement série
  - pile FIFO : First In, First Out



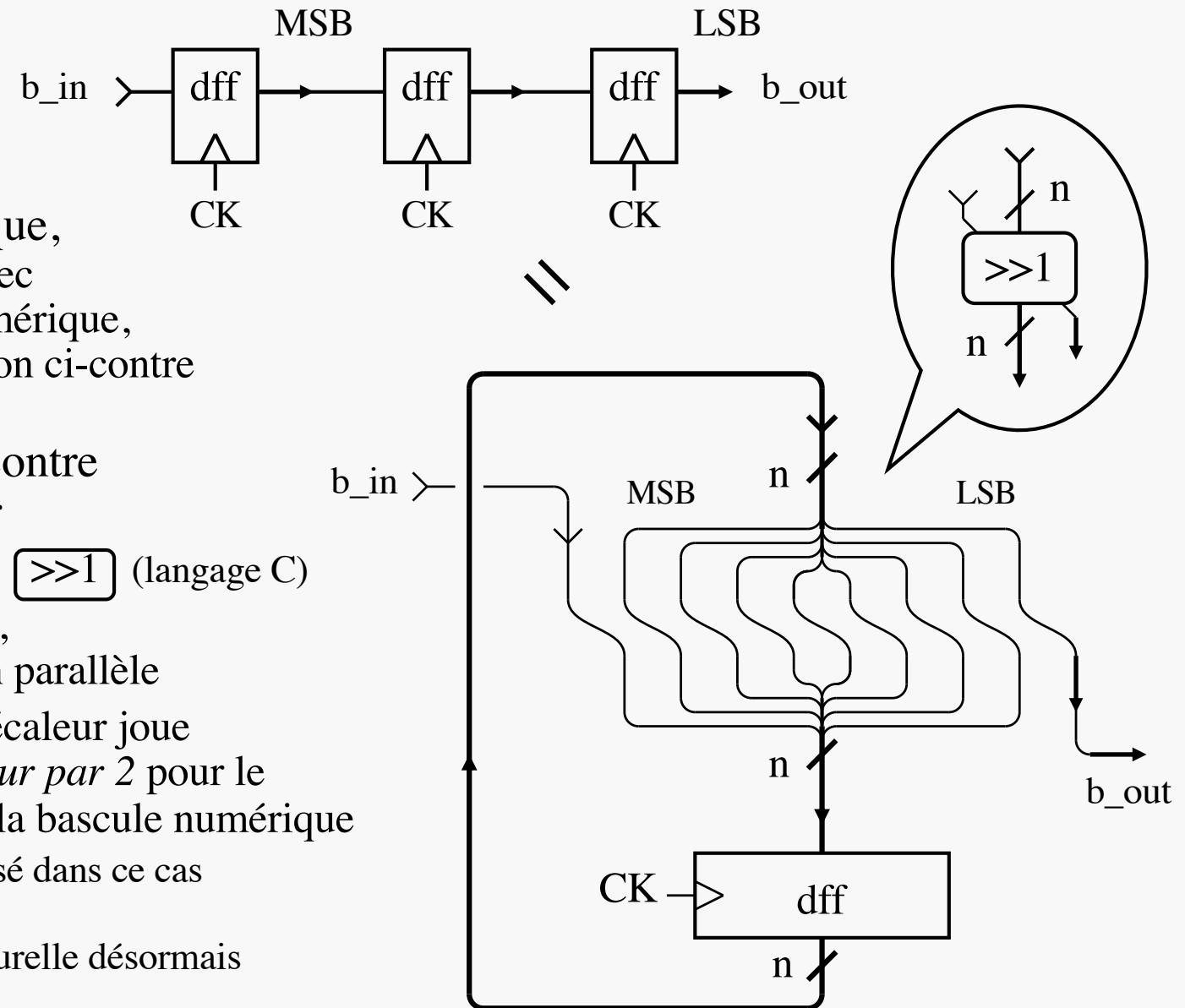
- Parfois utile de mixer accès série et parallèle
  - ci-contre : chargement série ( $b_{in}$ ) et sortie parallèle ( $d_{out}$ )
  - utilisation typique : réception de données transmises sur liaison série (ex. : USB)
    - $n$  périodes d'horloge pour charger un entier  $n$  bits sur un registre à décalage de largeur  $n$



- chargeons un *entier*, LSB en tête (choix dit *little-endian*)
  - $d_{out}$  doit être lu aussitôt le chargement terminé
  - un top d'horloge de plus et l'entier se retrouve *divisé par 2* (si  $b_{in}$  bien retombé à 0) ...
    - intéressant opérateur arithmétique obtenu presque gratuitement

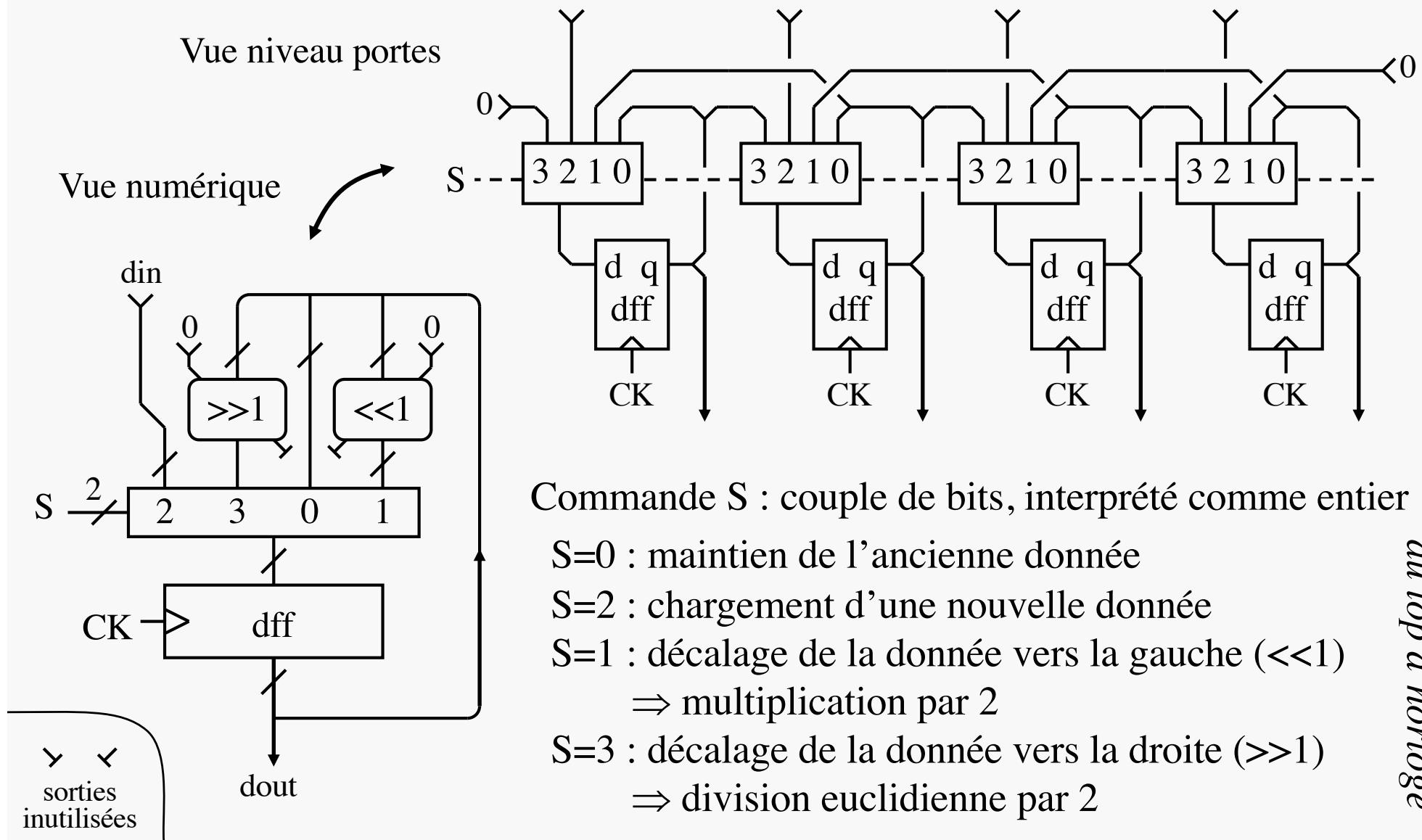
# ISOLEMENT NUMÉRIQUE DU DÉCALAGE

- Fonctionnalité isolable/séparable à un niveau numérique, pour être combinée avec celle de bascule D numérique, grâce à la réorganisation ci-contre
- La nappe de fils ci-contre est appelée *décaleur*
  - et symbolisée par  $\gg 1$  (langage C)
  - pour  $n$  bascules D, ce sont  $n+1$  fils en parallèle
  - avec  $b_{in}=0$ , le décaleur joue le rôle d'un *diviseur par 2* pour le nombre porté par la bascule numérique
    - $b_{out}$  non utilisé dans ce cas
- sortie parallèle  $d_{out}$  naturelle désormais



# REGISTRE, CAPABLE DE MULTIPLIER/DIVISER PAR 2

*grâce à un décalage bidirectionnel à volonté*



Commande  $S$  : couple de bits, interprété comme entier

$S=0$  : maintien de l'ancienne donnée

$S=2$  : chargement d'une nouvelle donnée

$S=1$  : décalage de la donnée vers la gauche ( $<<1$ )  
⇒ multiplication par 2

$S=3$  : décalage de la donnée vers la droite ( $>>1$ )  
⇒ division euclidienne par 2

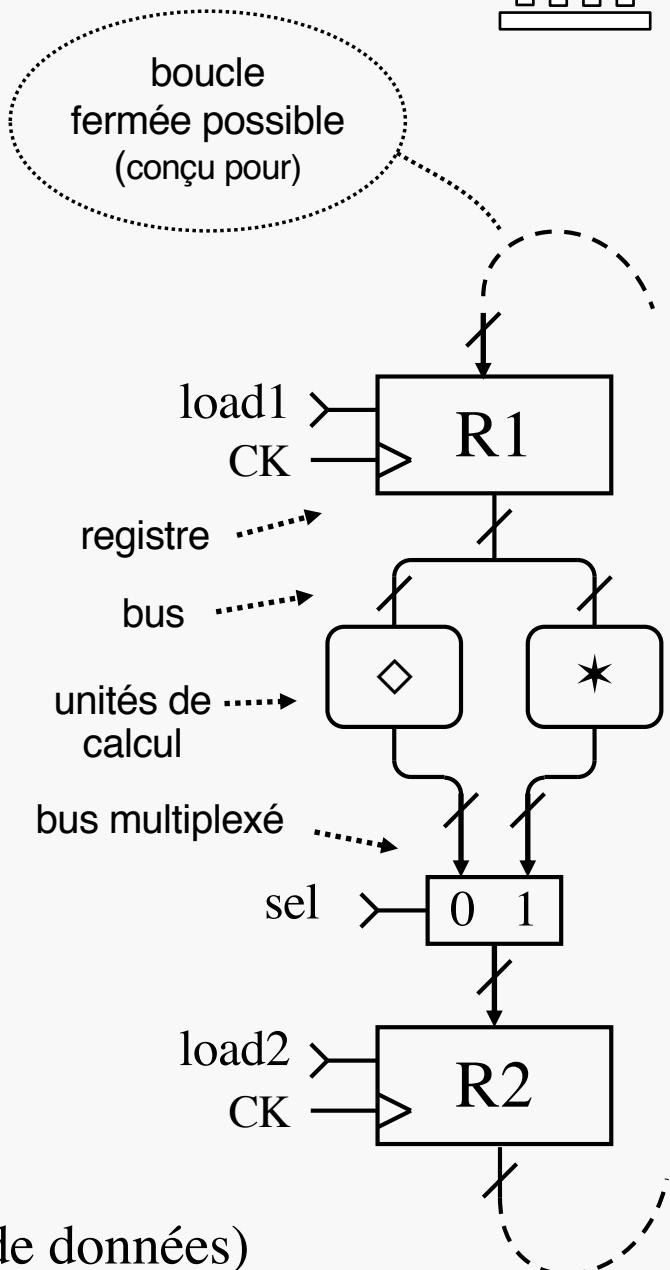
*au top d'horloge*

# CHEMIN DE DONNÉES (*DATAPATH*)



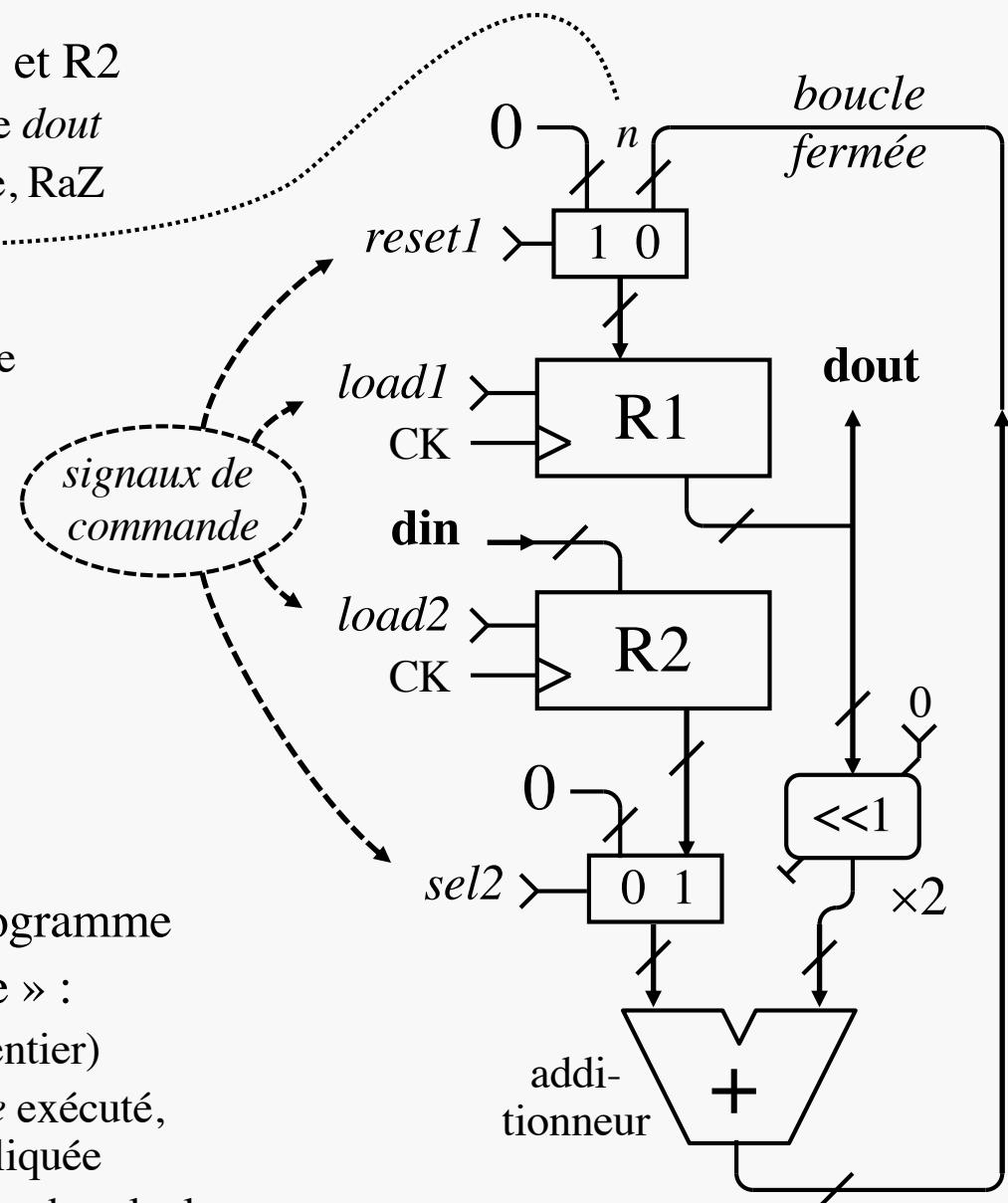
- Ensemble de *registres* (ou simples bascules numériques) contenant les données/variables numériques (c-à-d l'*état*) de l'algorithme en cours d'exécution reliés entre eux, ou avec l'extérieur, par des *bus* qui sont les *chemins* empruntés par les *données traitées*
  - certains multiplexés (configurations en Y)
- bus entrecoupés d'*unités de calcul combinatoires* ( $\diamond, \ast$ )
  - arithmétiques typiquement
  - effectuant une opération par cycle (période) d'horloge
- Chaque registre :
  - bloque sa donnée entre 2 tops d'horloge, pour utilisation par les unités de calcul
  - la conserve au top d'horloge si son *load*=0
  - en change si son *load*=1, en chargeant son entrée

*rappels*
- Les multiplexeurs :
  - déterminent quelles unités de calcul interviennent et comment, pendant la période d'horloge courante
- Les signaux *load* des registres et *sel(ect)* des MUX constituent les *signaux de commande du CD* (chemin de données)



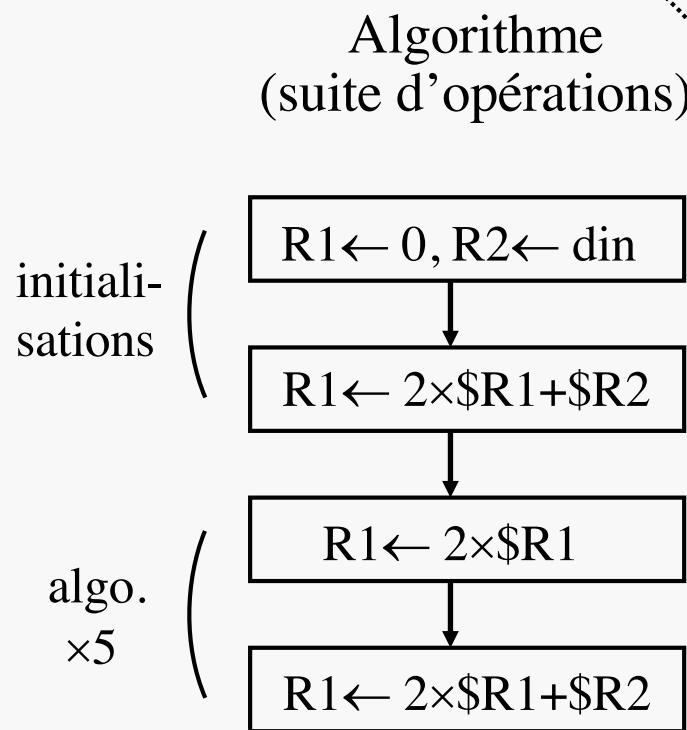
# EXEMPLE : UNE CALCULETTE PRIMITIVE

- Chemin de données à deux registres : R1 et R2
  - \$R1 (contenu de R1) accessible sur sortie *dout*
  - circuit séquentiel avec addition, décalage, RaZ
  - investissement > 3kt pour n=32 bits
- Plusieurs *opérations* possibles  
chacune en un cycle (une période) d'horloge  
selon les commandes appliquées :
  - Initialisations :
    - $R1 \leftarrow 0$  (*reset1*·*load1*)
    - $R2 \leftarrow \text{din}$  (*load2*)
  - Calculs :
    - $R1 \leftarrow 2 \times \$R1 + 0$  (*reset1'*·*load1*·*sel2'*)
    - $R1 \leftarrow 2 \times \$R1 + \$R2$  (*reset1'*·*load1*·*sel2'*)
      - R1 est dit monté en « accumulateur »
  - NOP : « no operation » (*load1'*·*load2'*)
- Séquence d'opérations (instructions) = programme
- Exploitation à suivre de cette « calculette » :
  - $dout \leftarrow c \times \text{din}$ , où  $c$  est un coefficient (entier)
  - valeur de  $c$  déterminée par le *programme* exécuté,  
donc par la séquence de *commandes* appliquée
  - programme basé sur un certain algorithme de calcul



# EXEMPLE : MULTIPLICATION PAR 5 SUR LA CALCULETTE

$d$  : valeur numérique présentée initialement sur l'entrée din



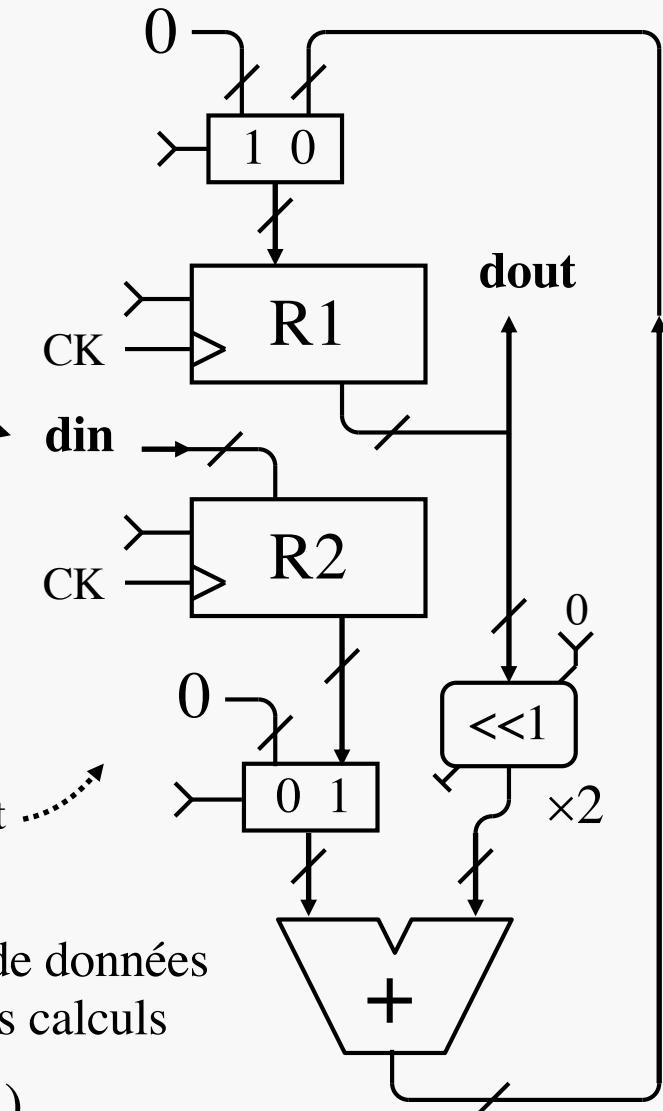
Contenus successifs des registres

$\$R2=d \quad \$R1=0$

$\$R2=d \quad \$R1=d$

$\$R2=d \quad \$R1=2d$

$\$R2=d \quad \$R1=4d+d$   
5d sur dout



Description « algorithmique » :

on ne voit du chemin de données (CD) que les mouvements de données entre ses registres (*Register Transfer*), combinés (ou pas) à des calculs

Niveau d'abstraction dit *RTL* (*Register Transfer Level*)

# QUI PILOTE LA CALCULETTE ?



Qui se charge de générer les signaux de commande du CD ?

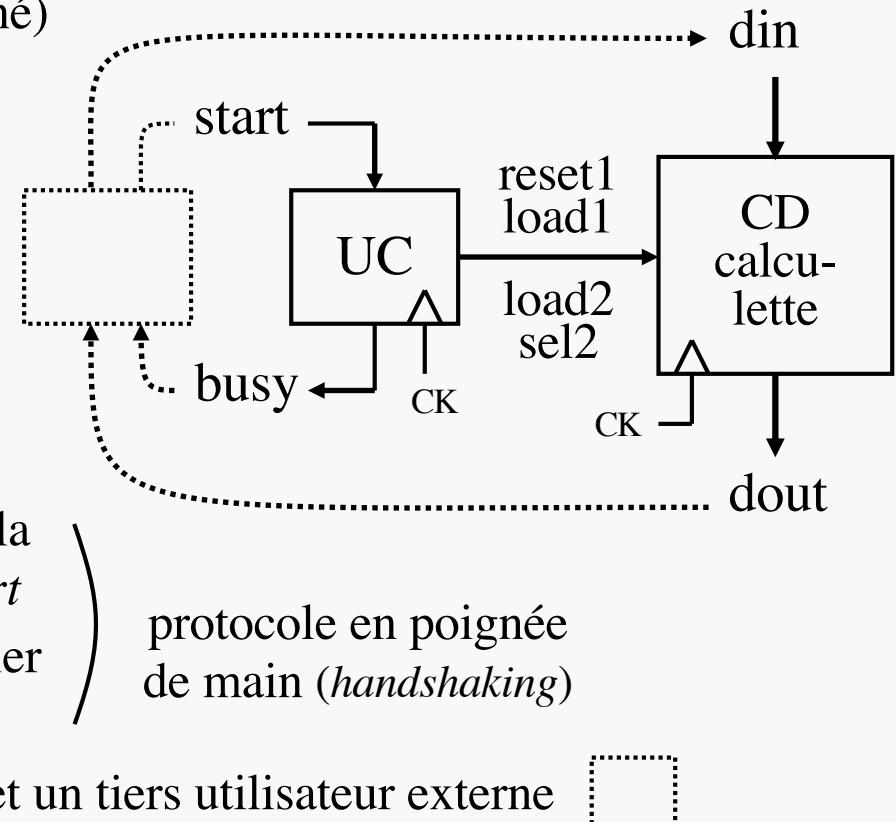
(pour y exécuter un calcul/algorithme donné)

→ Une *unité de commande* (*control unit*), alias UC  
circuit séquentiel indépendant du CD,  
mais également séquencé par CK,  
réalisant un certain *comportement*...

A suivre, projet d'UC un peu évoluée :

- capable de déclencher un calcul sur le CD à la demande, suivant la mise à 1 d'un signal *start*
- et de signaler que le CD est occupé (à dérouler un calcul) par un « voyant » / signal *busy*

→ cette UC est un intermédiaire entre le CD et un tiers utilisateur externe



Réalisation de l'UC :

- d'abord spécifiée sous forme de « diagramme algorithmique » ↘
- puis deviendra *automate*, synthétisé selon la méthodologie du CM6  
→ cas d'application approfondie exemplaire (cf. planches 13–18 + PC7/Exo2)

# DIAGRAMME ALGORITHMIQUE DE L'UC POUR $\times 5$ SUR LE CD

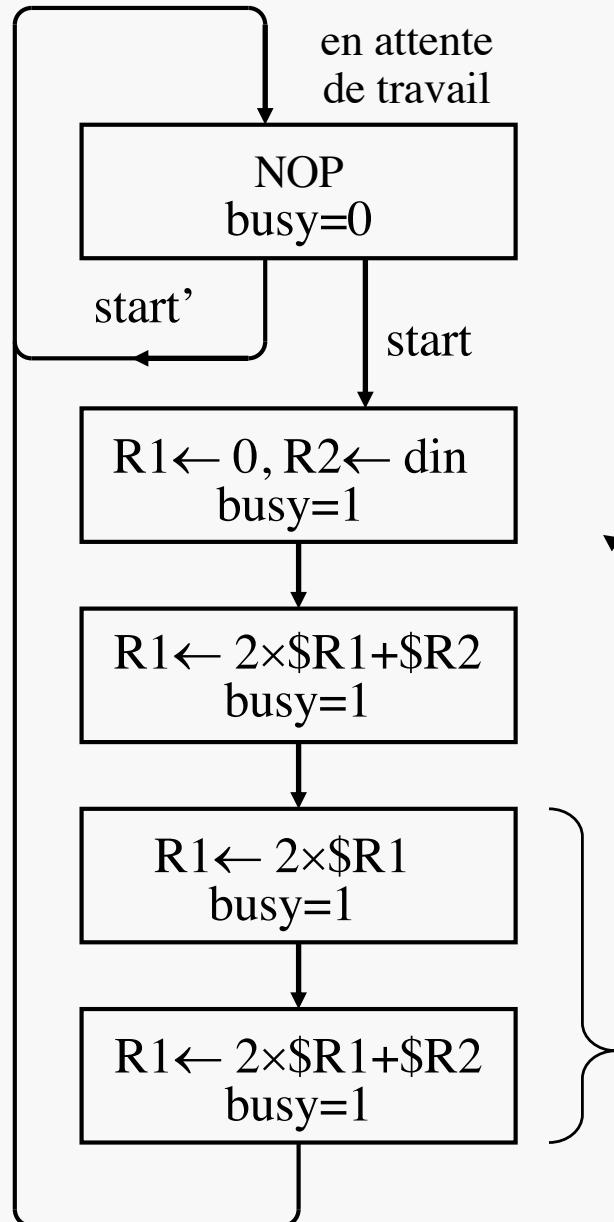
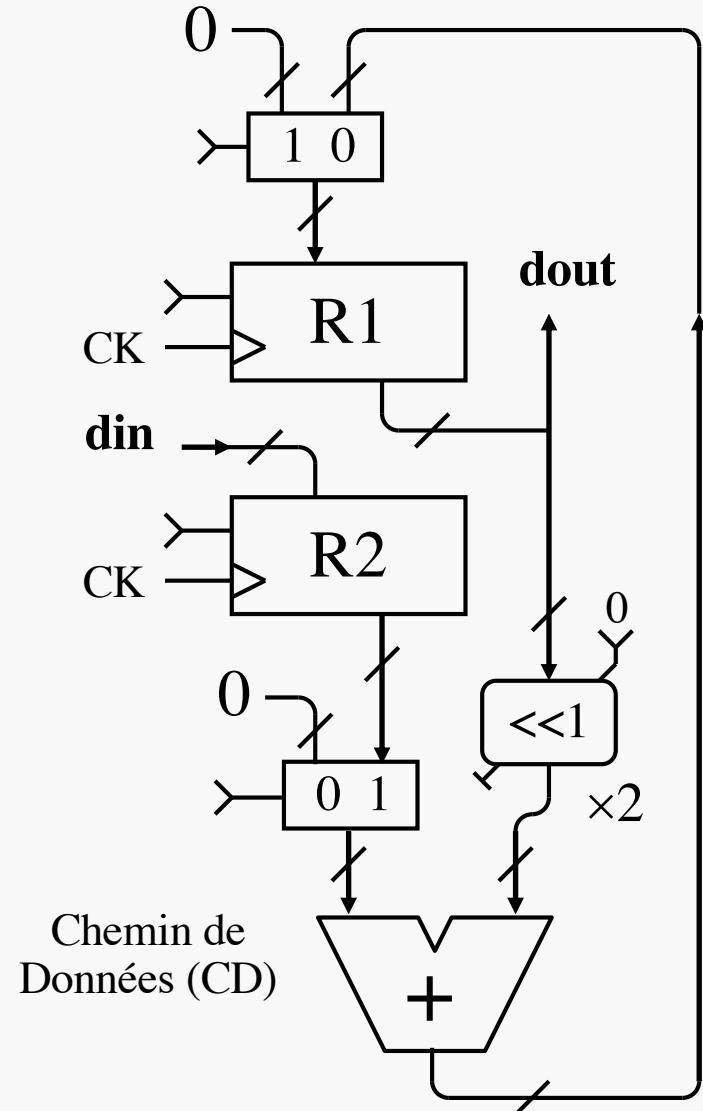


Diagramme algorithmique  
= spécification  
comportementale sous  
forme d'organigramme

Ressemble à un  
diagramme d'état,  
avec entrée (start)  
et sortie (busy).

Mais chaque état  
correspond à une  
instruction RTL, qui  
masque le détail des  
signaux de commande  
à appliquer au CD

algorithme de  
multiplication par 5



# SYNTHÈSE DE CIRCUITS SÉQUENTIELS

*Spécifications comportementales*



① *Elaborer un diagramme d'état*

entrées  $X$  - sorties  $Y$   
états naturels, t.q.  $q_{init}$   
spécifications t.q.  $g^*(q_{init}, \cdot)$



Mobiliser les états nécessaires au comportement voulu

Trouver  $Q$ ,  $f$  et  $g$  répondant aux spécifications



② *Encoder les états*

$q$  symbolique

changement de variables

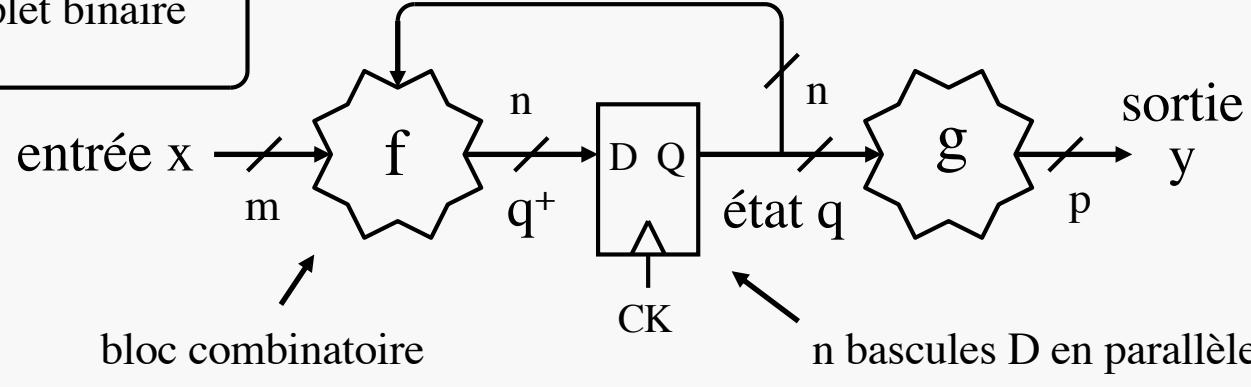
$|Q|$  désormais connu

Choisir  $n \geq \log_2(|Q|)$   
et  $\gamma$  injectif :  $Q \rightarrow \mathbb{B}^n$



③ *Implanter*

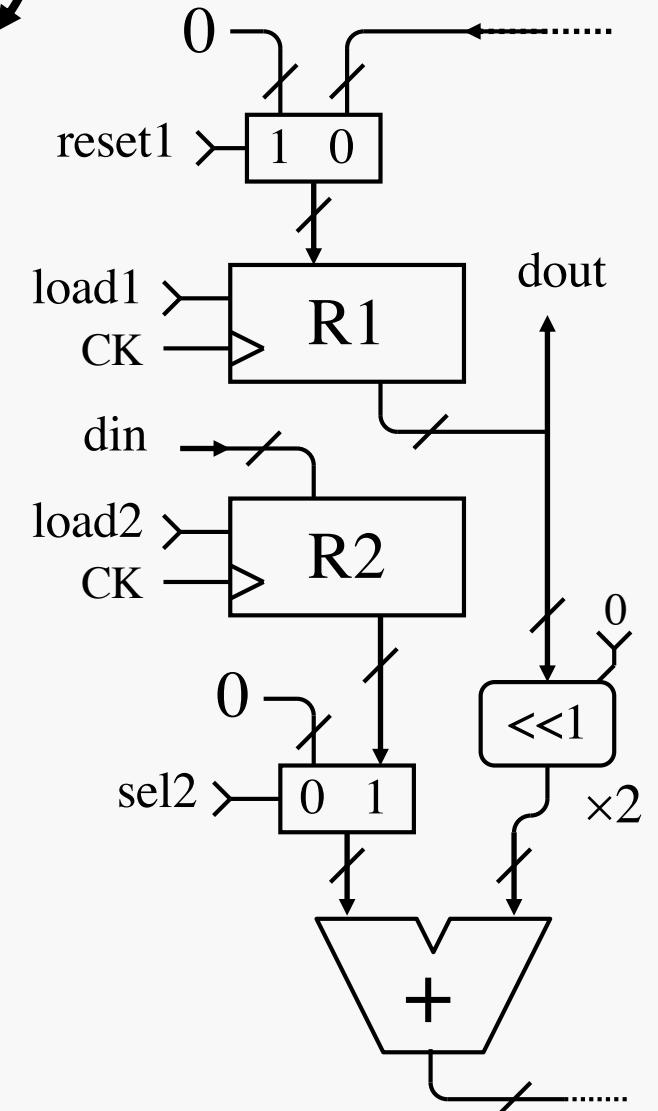
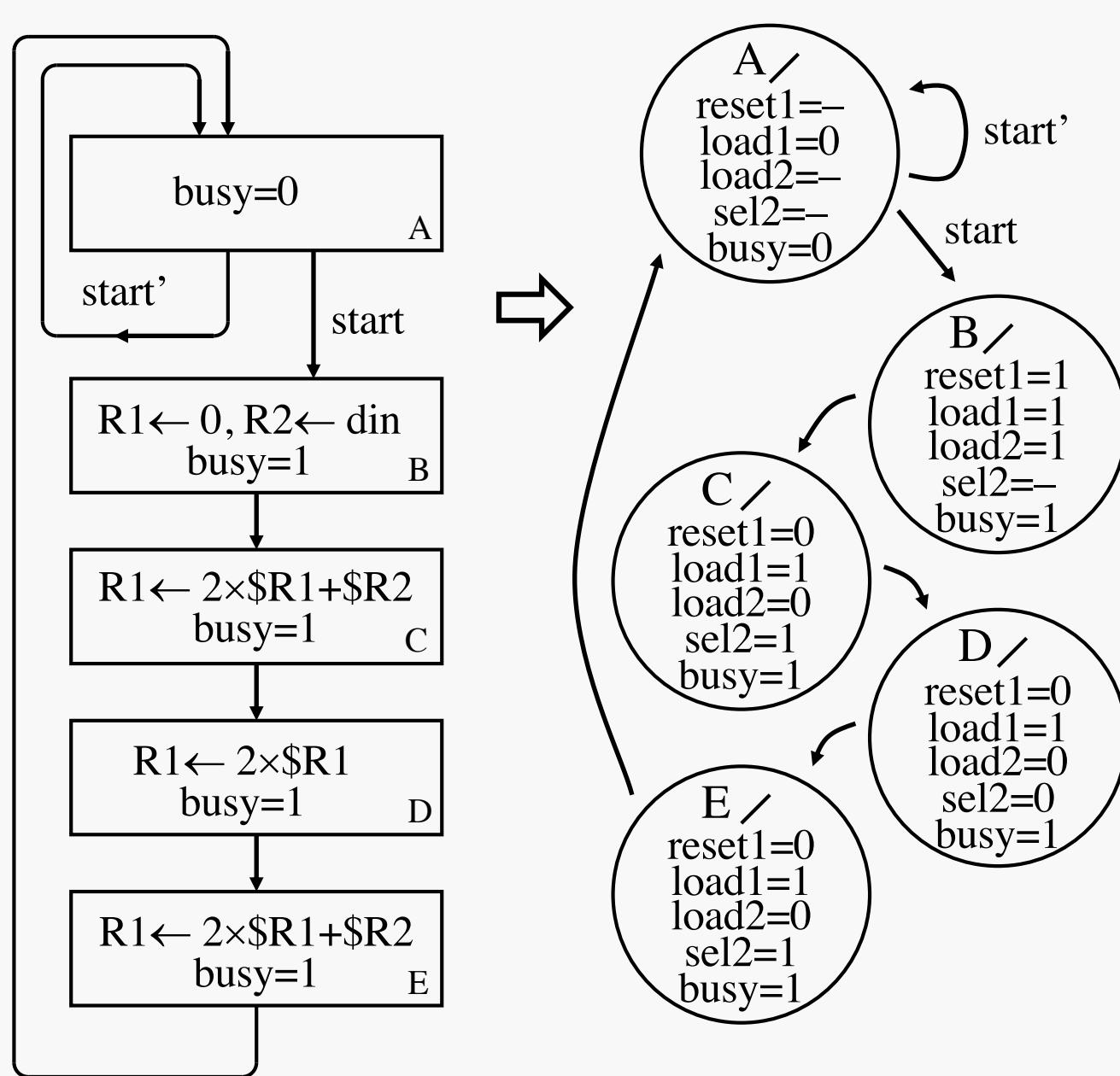
$q$  numérique =  
 $\gamma(q$  symbolique)  
 $n$ -uplet binaire

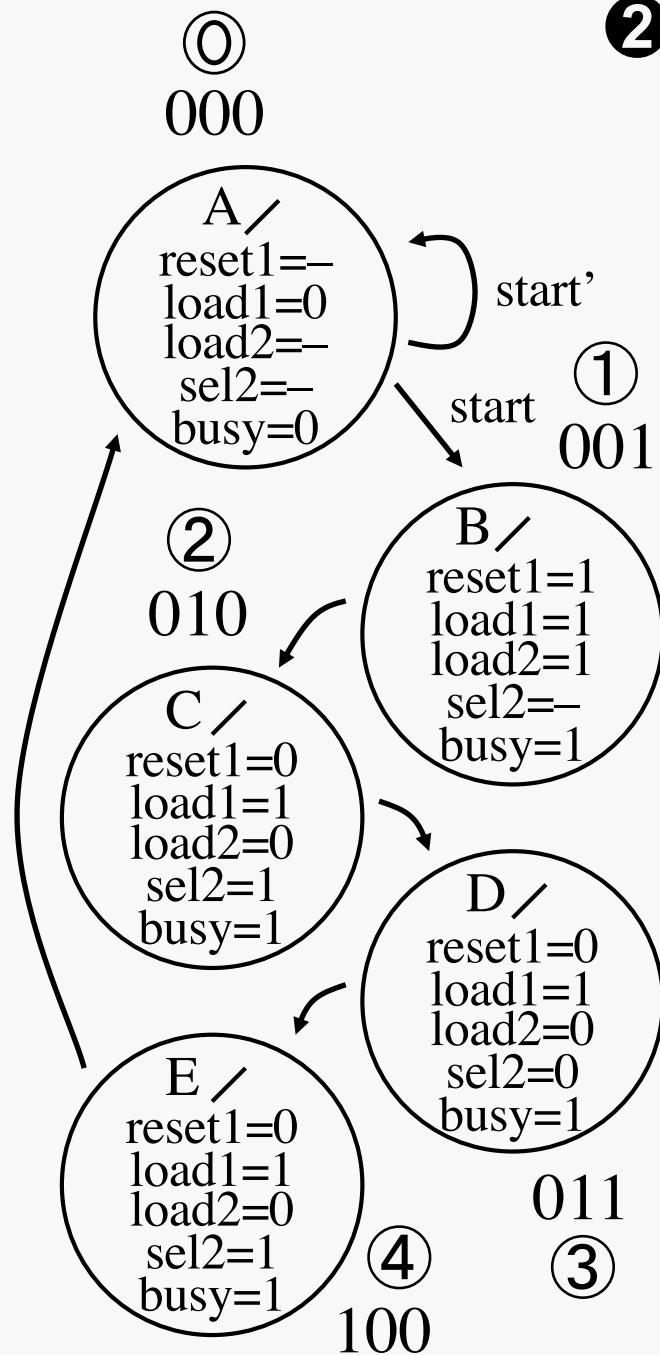


CM6

→ PC 6, 7 &8

# ① : DIAGRAMME D'ÉTAT DE L'UC



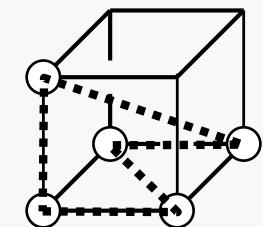


## ② : ENCODAGE D'ÉTAT

Choix d'un encodage naïf : état représenté par son numéro d'ordre partant de A, codé comme entier non signé sur 3 bits

- sorte de compteur *tronqué*
- un encodage plus adjacent pourrait être meilleur...

→PC7



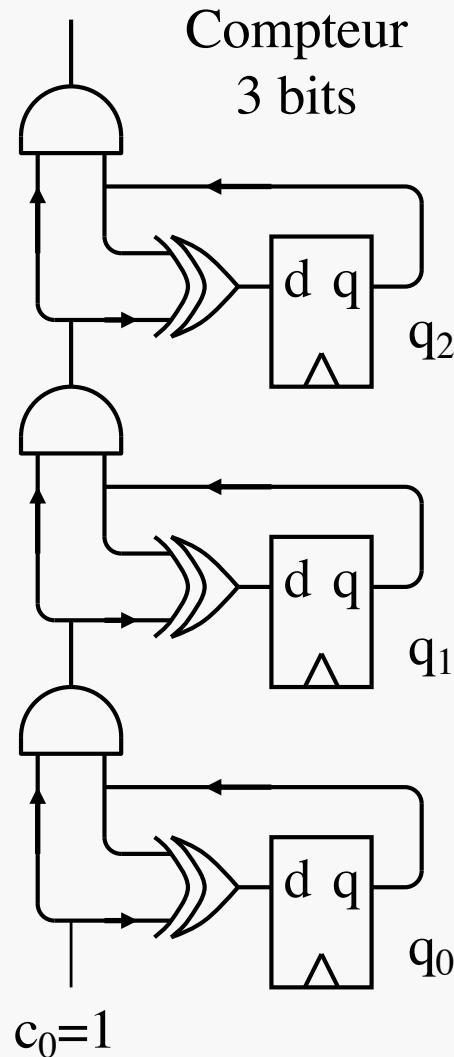
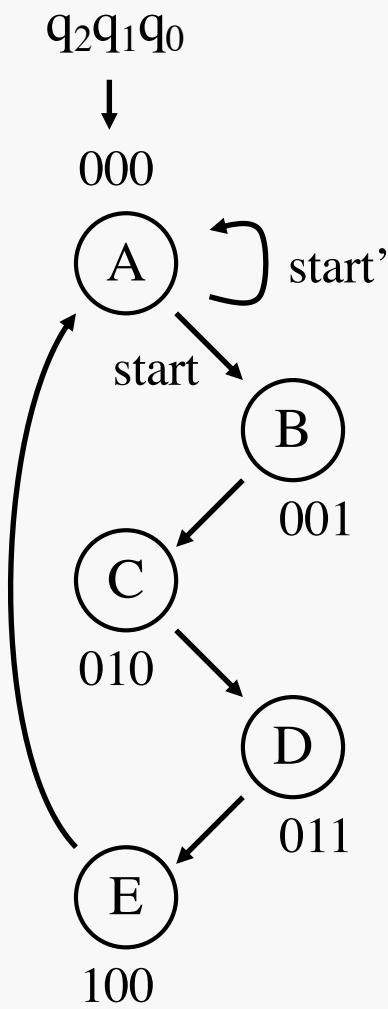
## ③ : IMPLANTATION

Logique de transition ( $f$ ) établie ci-après par :

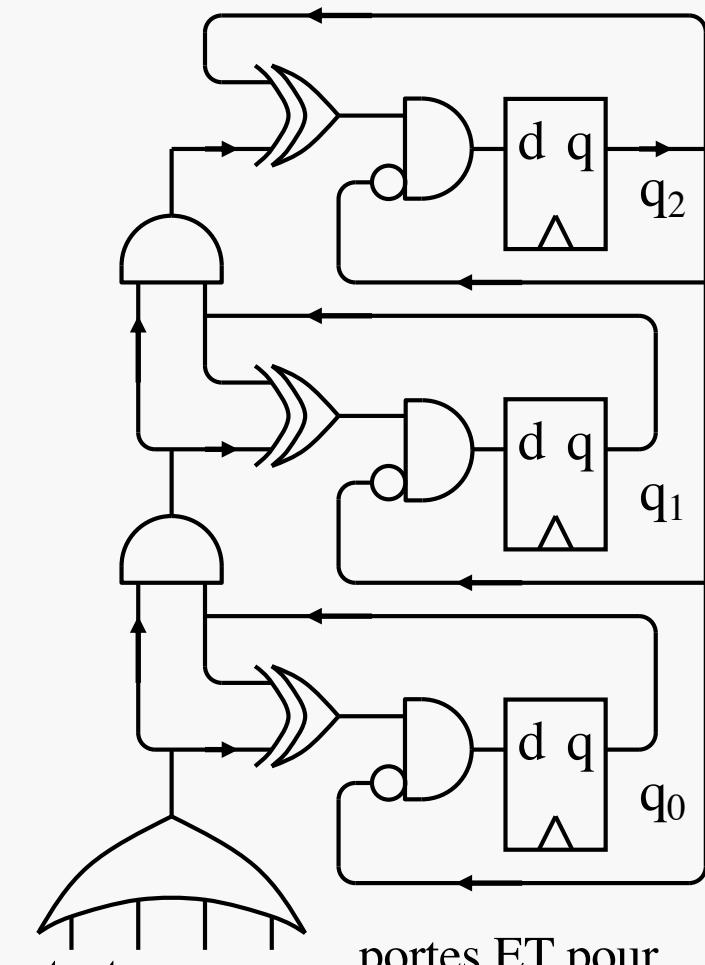
- approche artisanale
- méthode tabulaire vectorielle
- méthode algébrique

*méthode de référence*

### 3a : APPROCHE ARTISANALE



Compteur astucieusement bricolé pour implanter la logique de transition

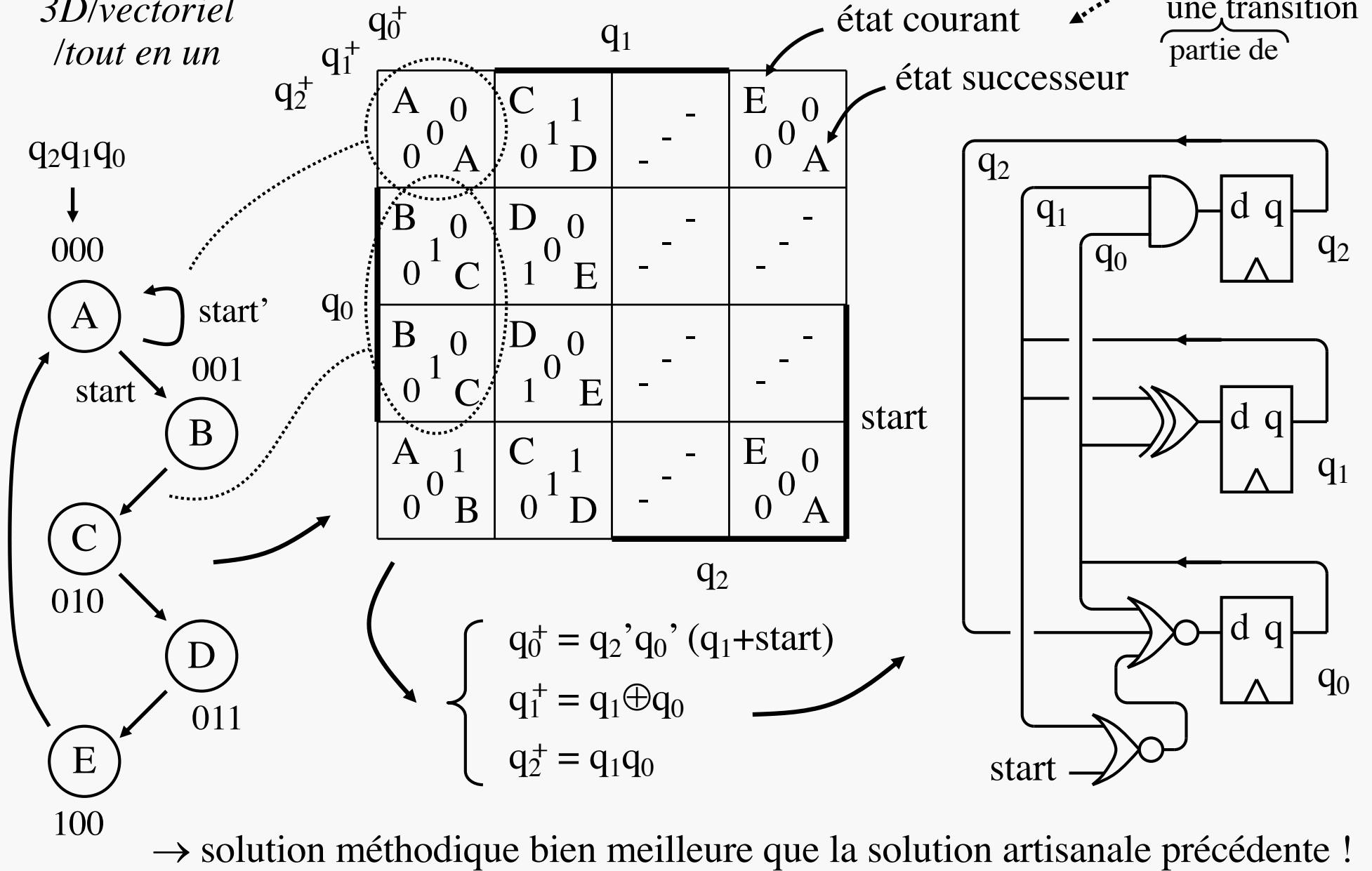


Avec l'affectation des états ci-dessus, l'automate ressemble à un compteur cyclique de 0 à 4

portes ET pour transition  $E \rightarrow A$

### 3b : MÉTHODE TABULAIRE

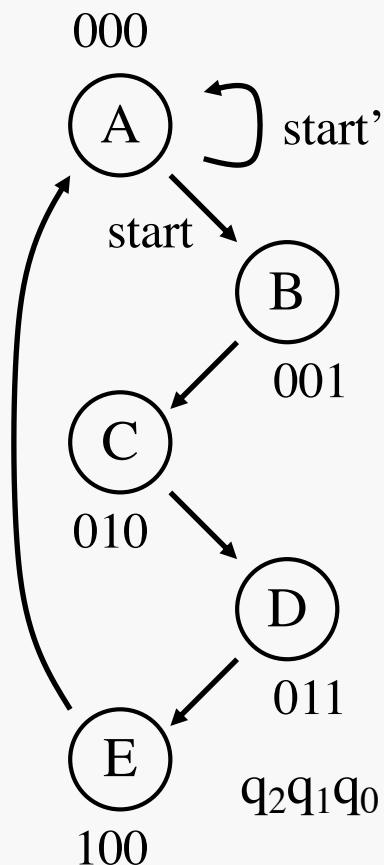
*Tableau  
3D/vectoriel  
tout en un*



### ③c : MÉTHODE ALGÉBRIQUE

A, B, C, D et E : variables binaires valant 1 lorsque l'état correspondant est occupé

$A^+, B^+, C^+, D^+$  et  $E^+$  : autres variables binaires, valant 1 lorsque l'état correspondant sera occupé à la prochaine période



Alors, on note que :

$$q_2^+ = E^+ = D = q_2'q_1q_0$$

$q_2$  vaudra 1  
ssi on est en E  
à la prochaine  
période

on sera en E  
ssi on est en D  
maintenant

on est en D  
maintenant ssi  
 $q_2'q_1q_0=1$

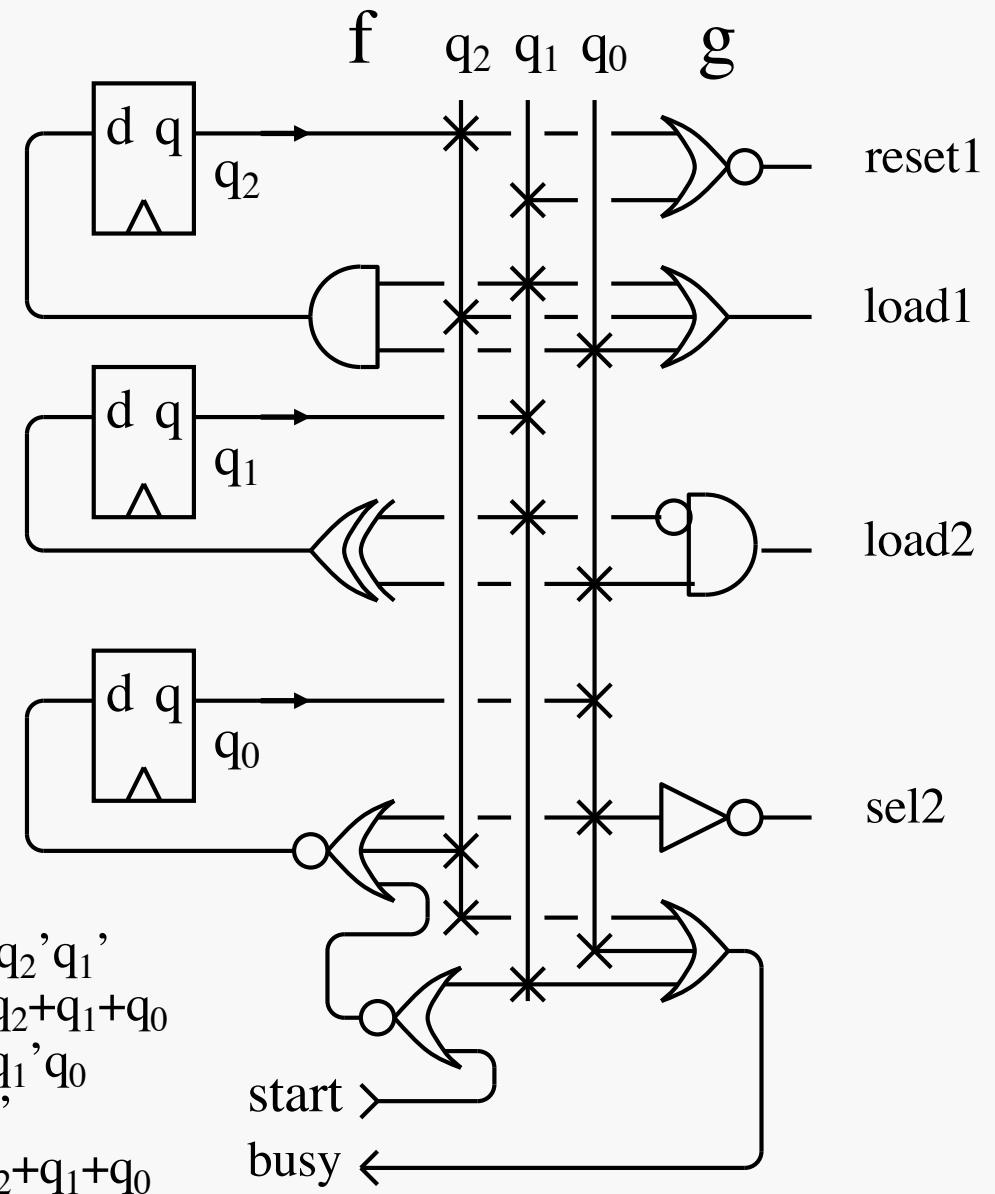
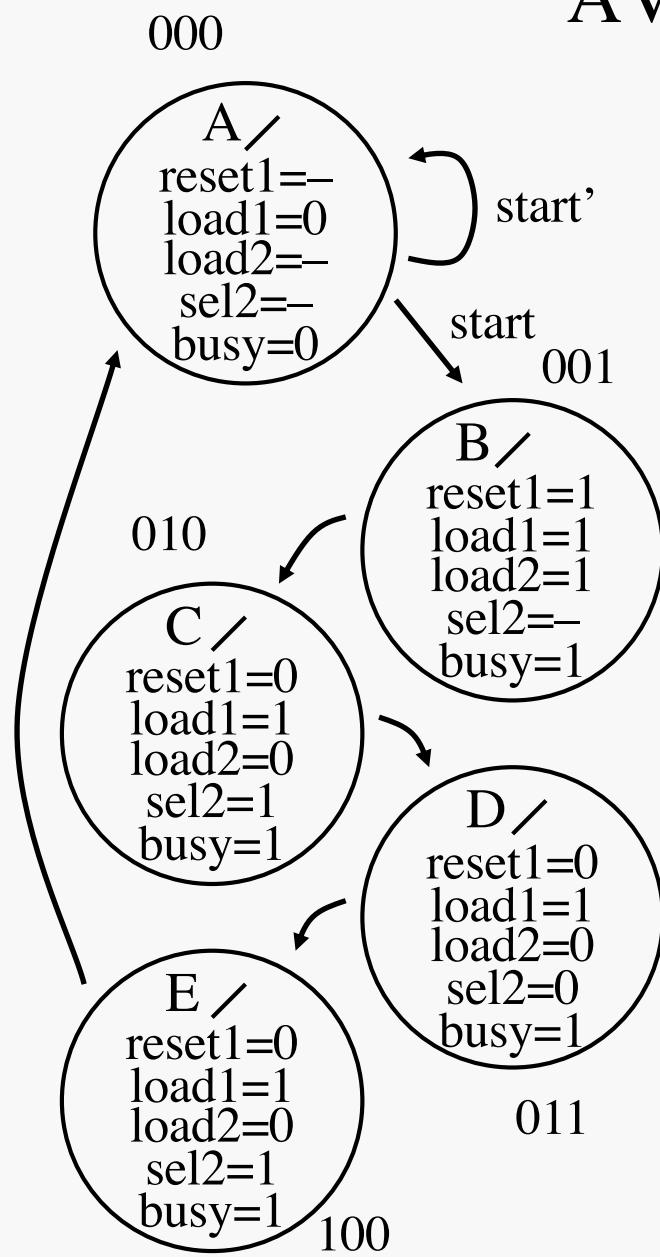
à la prochaine  
période, on sera en B  
ssi on est en A maintenant  
et que  $start=1$

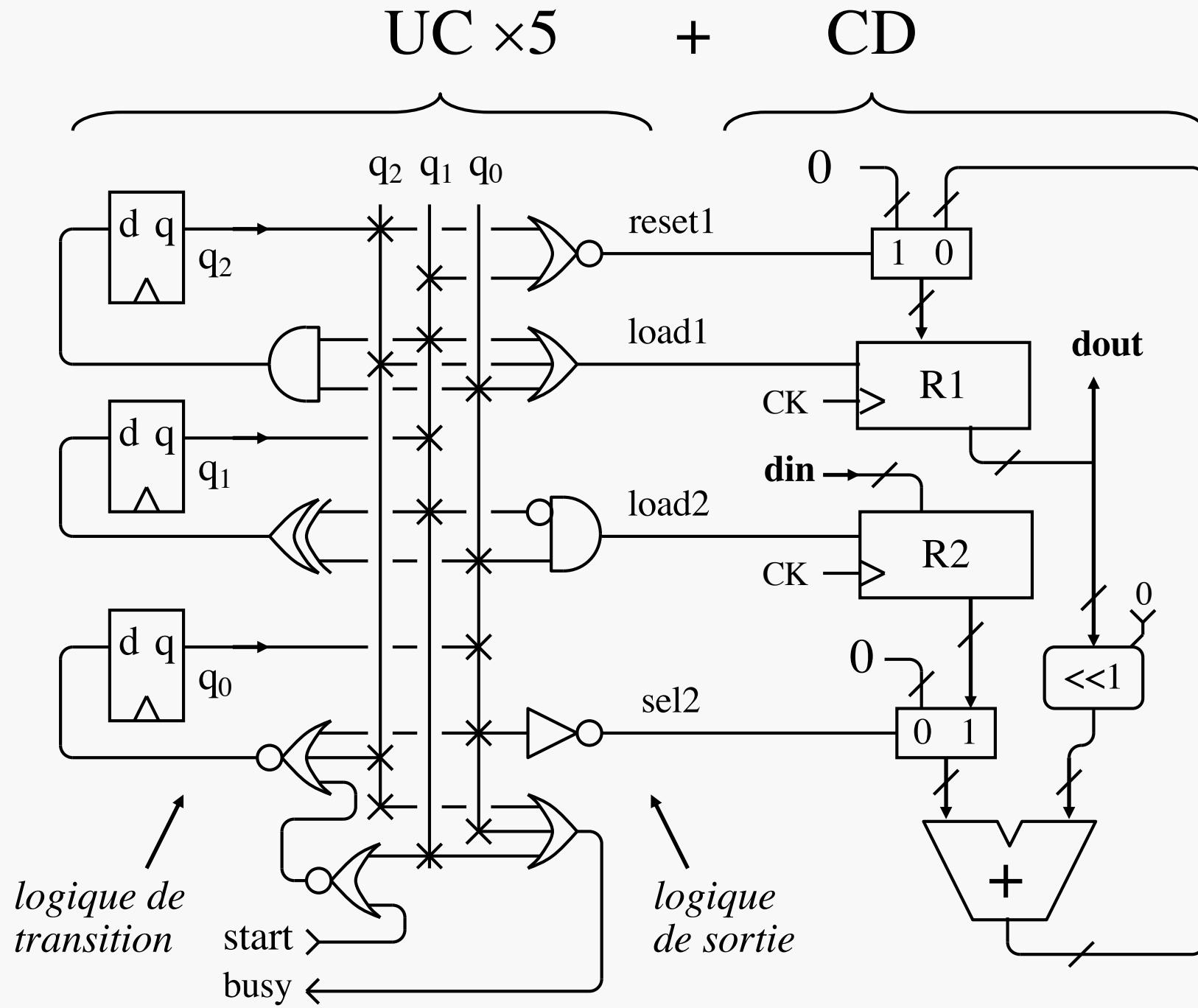
égalité correcte,  
mais forme  
non minimale  
( $q_2'$  inutile,  
en profitant des  
indéterminations)

$$\begin{aligned}
 q_0^+ &= (B^+ + D^+) \\
 &= start \cdot A + C \\
 &= start q_2'q_1'q_0' + q_2'q_1q_0' \\
 &= (start + q_1) q_2'q_0' \quad \checkmark
 \end{aligned}$$

→ méthode rapide mais résultats non minimaux (ne profitant pas des indéterminations)

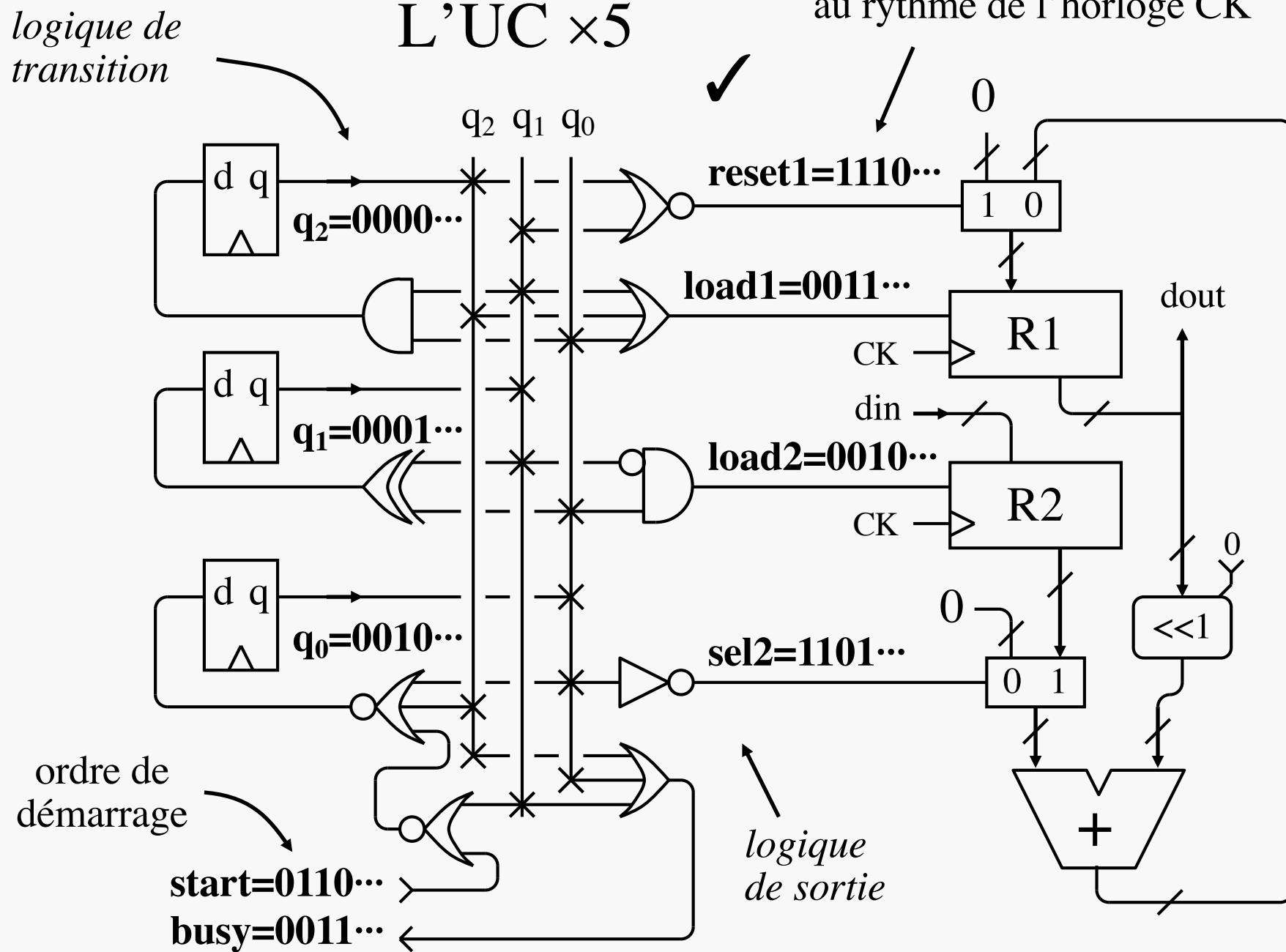
# ×5 : UC COMPLÈTE, AVEC LOGIQUE DE SORTIE





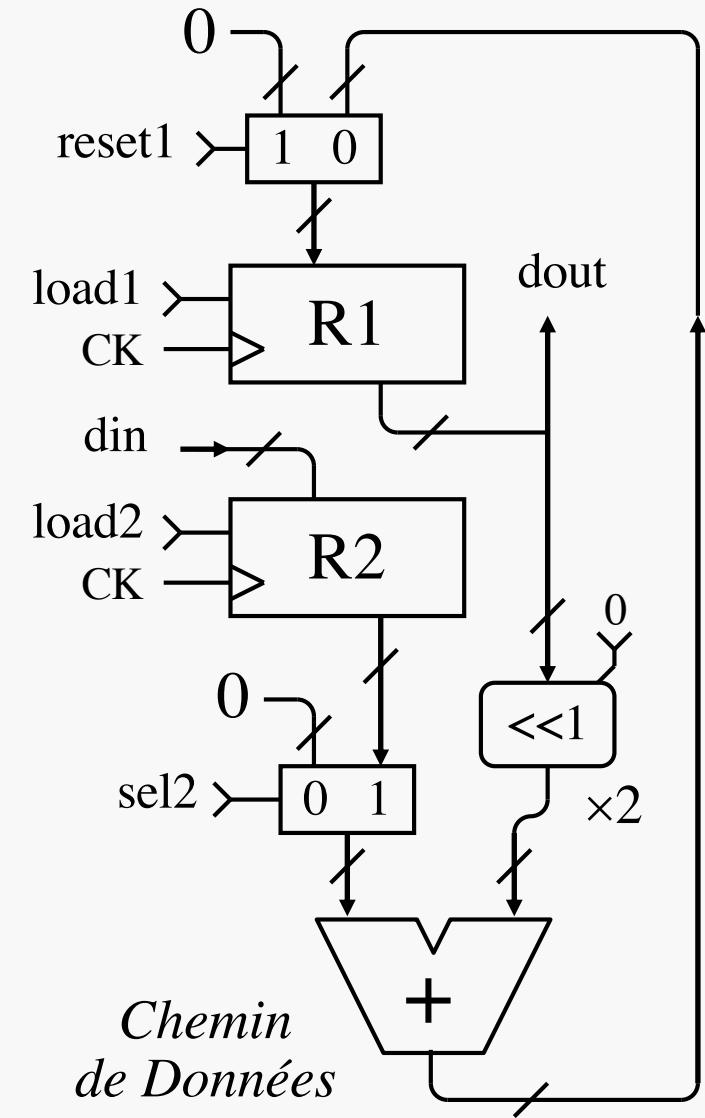
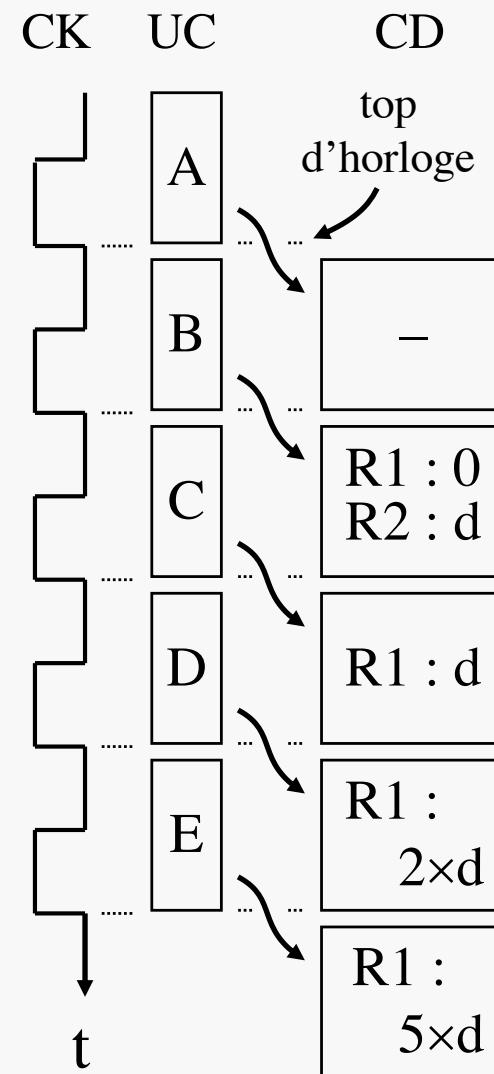
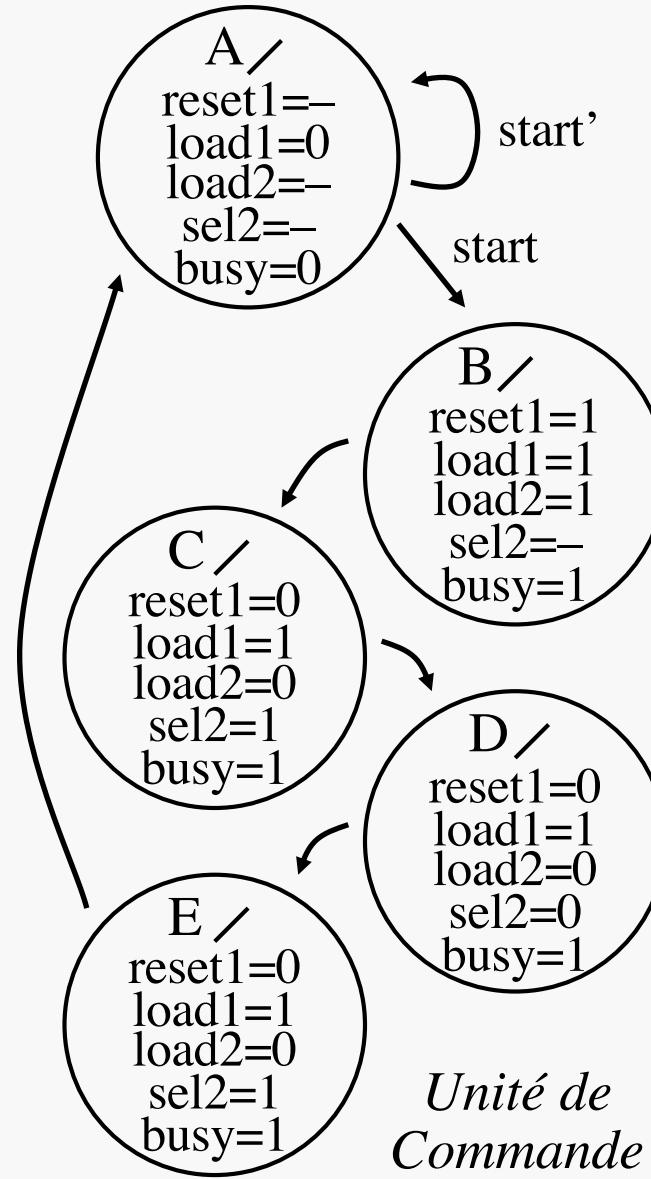
# ESSAI DE L'UC ×5

## succession temporelle des valeurs au rythme de l'horloge CK

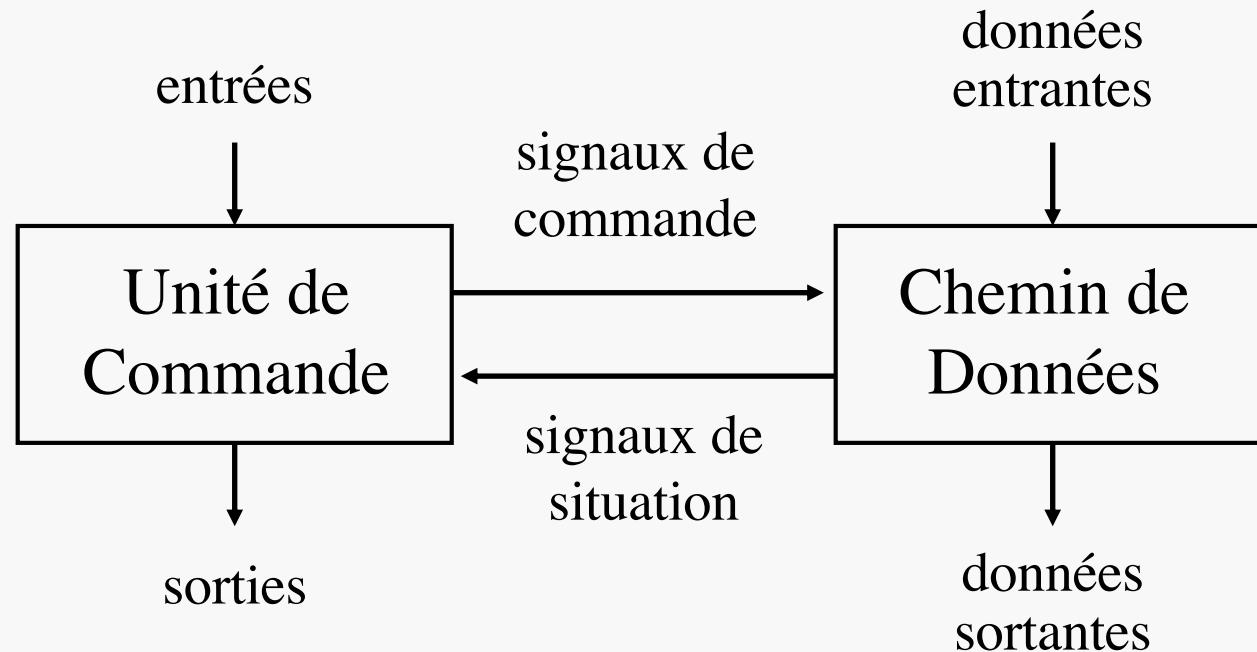


# RETARD COMMANCE-ACTION

Les sorties de l'UC n'agissent sur le CD qu'à la période suivante...



# UN MODÈLE ARCHITECTURAL GÉNÉRAL ...



- qui porte en germe la notion de programmation
  - ici, programmation matérielle (*hard*) du CD par l'UC
- retrouvailles prochaines dans l'architecture d'un microprocesseur → CM9
- bascules D présentes de chaque côté, dans des rôles distincts :
  - stocker des nombres dans les registres du CD
  - porter l'état de l'UC

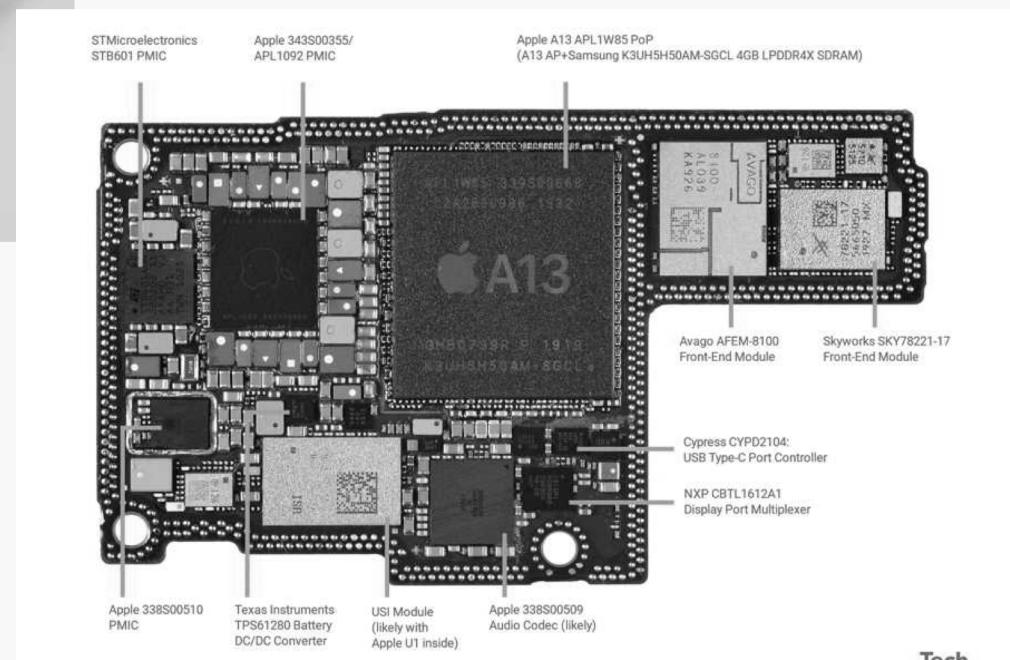
<b>y</b>	<b>Q</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	
reset1	–	1	0	0	0	0	
load1	0	1	1	1	1	1	
load2	–	1	0	0	0	0	
sel2	–	–	1	0	0	1	→ CM9
busy	0	1	1	1	1	1	instruction

## ÉMERGENCE LOGICIELLE...

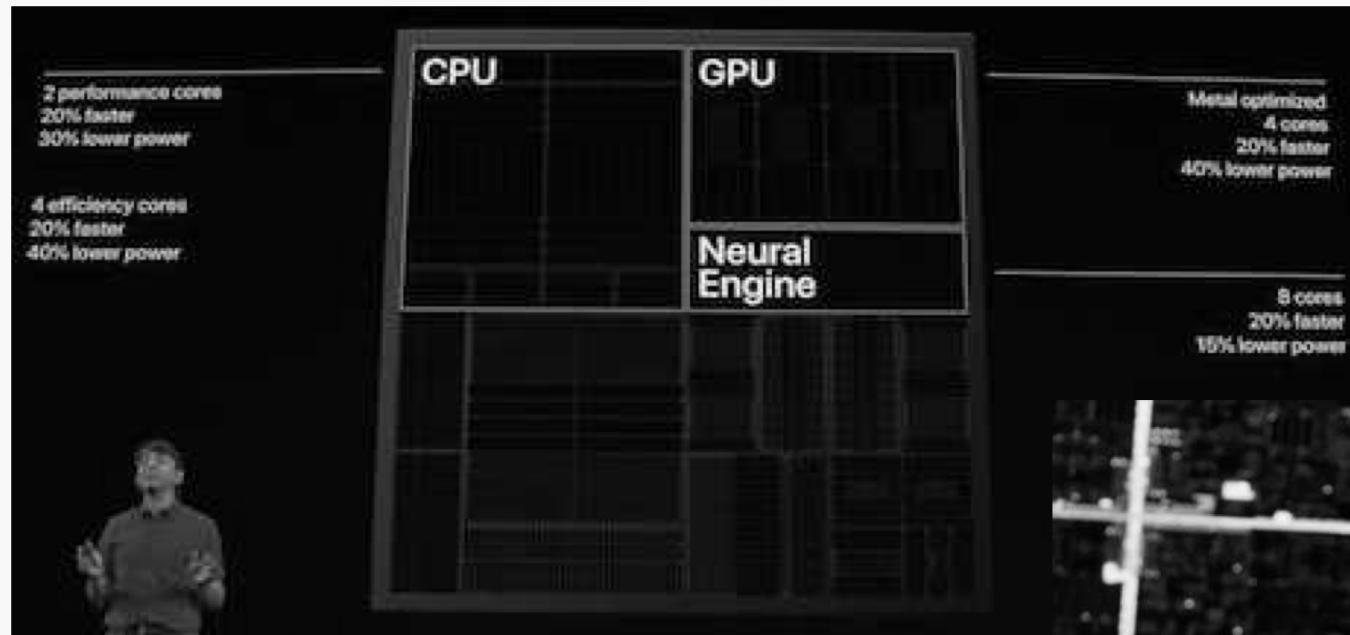
- Les valeurs de signaux de commande pour un calcul/algorithme donné pourraient être stockés dans une mémoire tableau, comme ci-dessus
- Celle-ci remplacerait la logique de sortie de l'UC, pourvu qu'elle soit équipée d'un mécanisme d'accès au contenu de chaque colonne ci-dessus, au choix
  - un tel contenu constitue une *instruction* pour le CD
  - accès via un *décodeur d'adresse* (PC4/Exo2), avec le n° de colonne comme adresse
- Intérêts :
  - contenu du tableau facile à modifier : c'est ainsi qu'on passera *du hard au soft* !
  - organisation similaire pour chaque calcul/algo → rationalisation & économies



# iPhone 11 Pro Max



# SoC\* A13



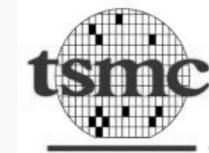
09/2019

98mm<sup>2</sup> 8,5Gt  
7nm.v2 FinFET



Sri Santhanam, Vice president of silicon engineering, 10/09/2019

« At Apple, we have the benefit of owning the entire vertical stack »  
(the software, the system hardware, and the chip design)



**ARM**  
v8.3-A