Cours #5
Conception de chemins des données

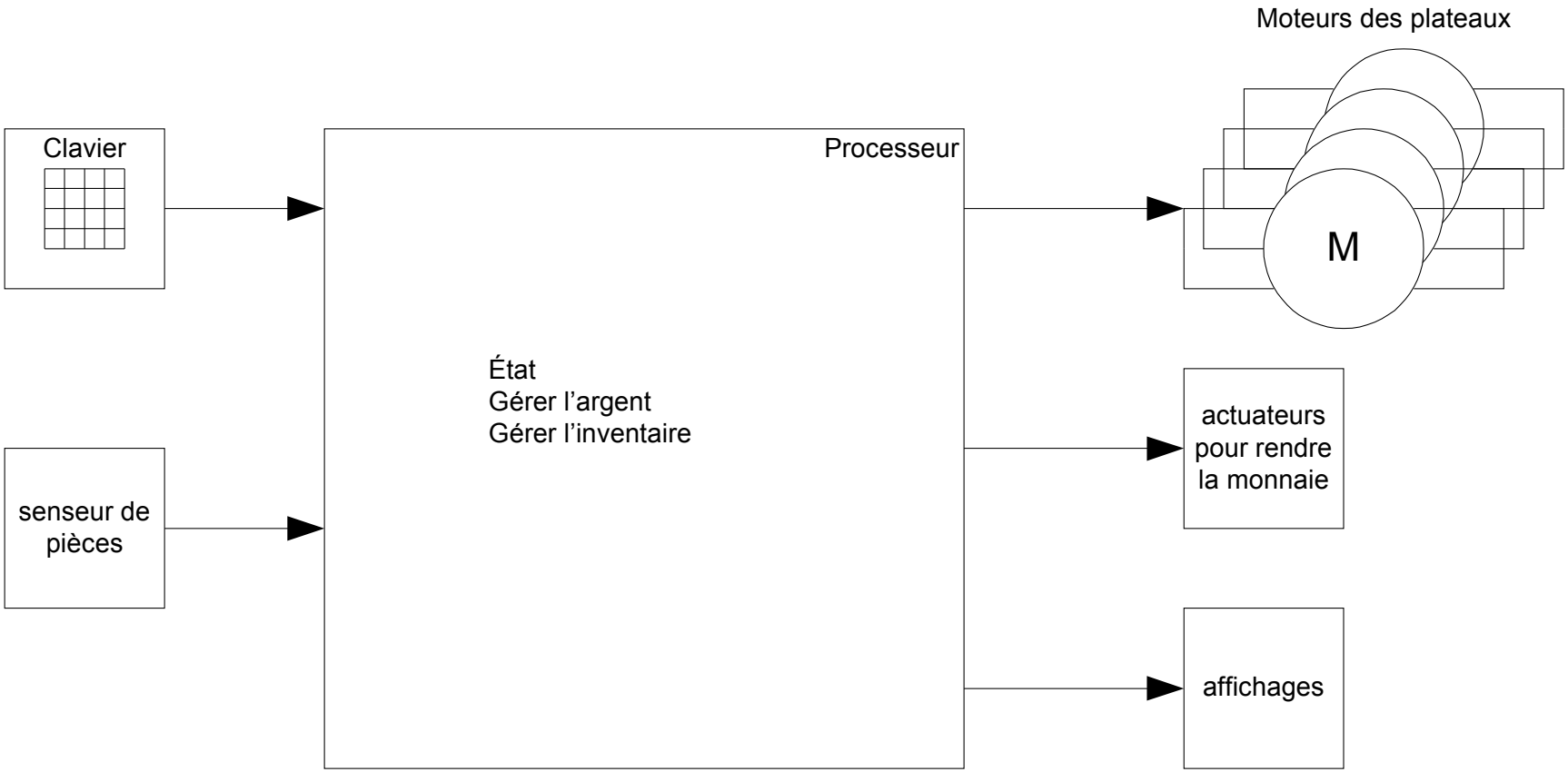
Plan

- Processeurs: introduction
- Modules combinatoires:
  - multiplexeurs, décodeurs et encodeurs
- Éléments à mémoire pour chemins des données:
  - registres à chargement parallèle et à décalage, bloc de registres, RAM
- Unités fonctionnelles:
  - unités arithmétiques, unités logiques, comparateurs, compteurs

Un exemple pour commencer

Source: Wikipédia

Un exemple pour commencer

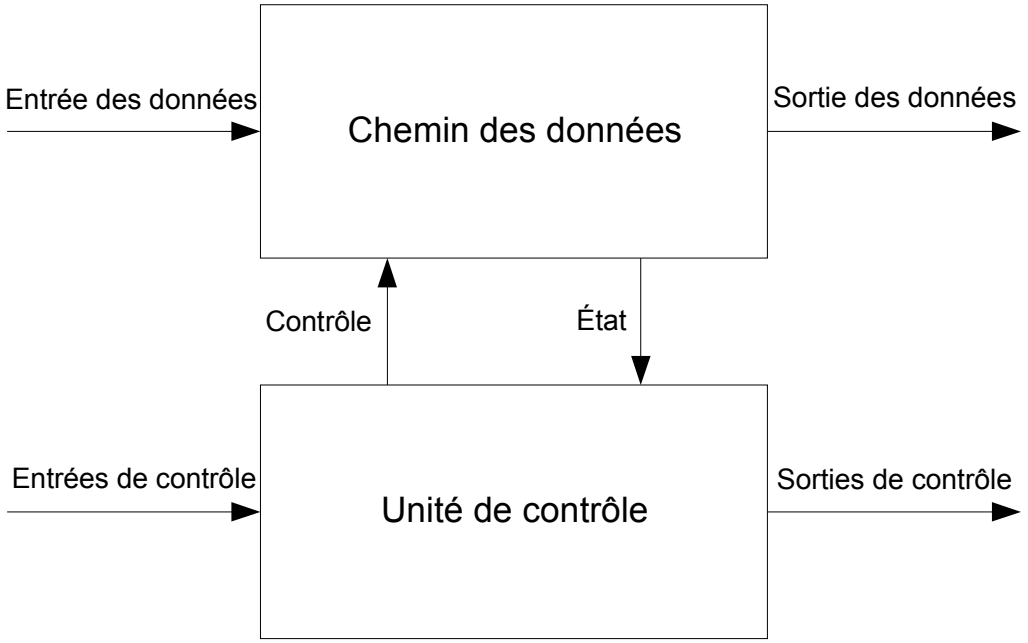


Deux types de processeurs

- Un processeur est un type spécial de système numérique dont le but est de traiter des données par une succession d'étapes simples, pouvant résulter en un traitement global complexe.
- Un processeur doit acquérir des données, les traiter, et produire un résultat sous forme numérique ou vidéo.
- On distingue deux types principaux de processeurs :
  - Les processeurs à usage général peuvent être programmés. Le programme exécuté par un processeur est gardé en mémoire sous la forme d'une liste d'instructions.
    - On réfère souvent à ce type de processeur par le nom de « microprocesseur ».
    - Un microcontrôleur est un cas particulier d'un processeur à usage général.
    - Un processeur spécialisé est un processeur à usage général auquel on a ajouté des instructions spéciales.
  - Les processeurs à usage spécifique sont des processeurs non programmables qui sont conçus dans le but de répondre à un besoin unique.
    - Ils sont plus simples et plus efficaces que les processeurs à usage général.
    - Ils ne peuvent pas en général être facilement reprogrammés.

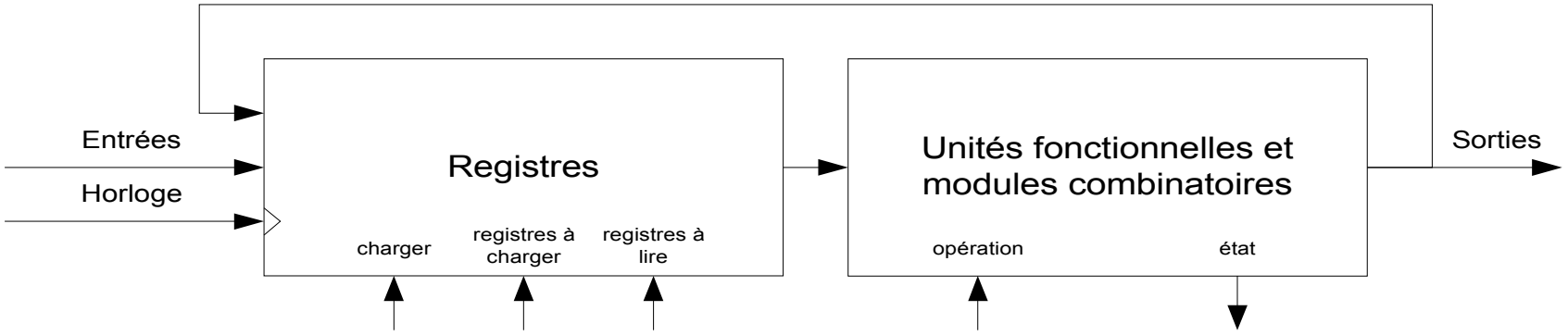
Parties d'un processeur

- Un processeur est composé de deux parties:
  - Le chemin des données (*datapath*) traite les données. Il inclut:
    - des registres;
    - des unités fonctionnelles (comme une unité arithmétique et logique)
    - un mécanisme de commutation pour transférer et manipuler les données.
  - L'unité de contrôle (*control unit*) est responsable du séquencement des opérations à exécuter par le chemin de données selon des entrées externes et le résultat des opérations.



Architecture d'un chemin des données

- Un chemin des données a deux parties principales:
  - un bloc de registres qui conserve les données à traiter et des résultats précédents, de façon à pouvoir combiner toutes ces valeurs dans de nouveaux calculs.
  - des unités fonctionnelles pour effectuer des opérations sur les données conservées dans les registres.
- Des modules combinatoires permettent de choisir, router et contrôler le flot d'information entre les registres et les unités fonctionnelles.



Architecture d'un chemin des données
Exemple: processeur BlackFin

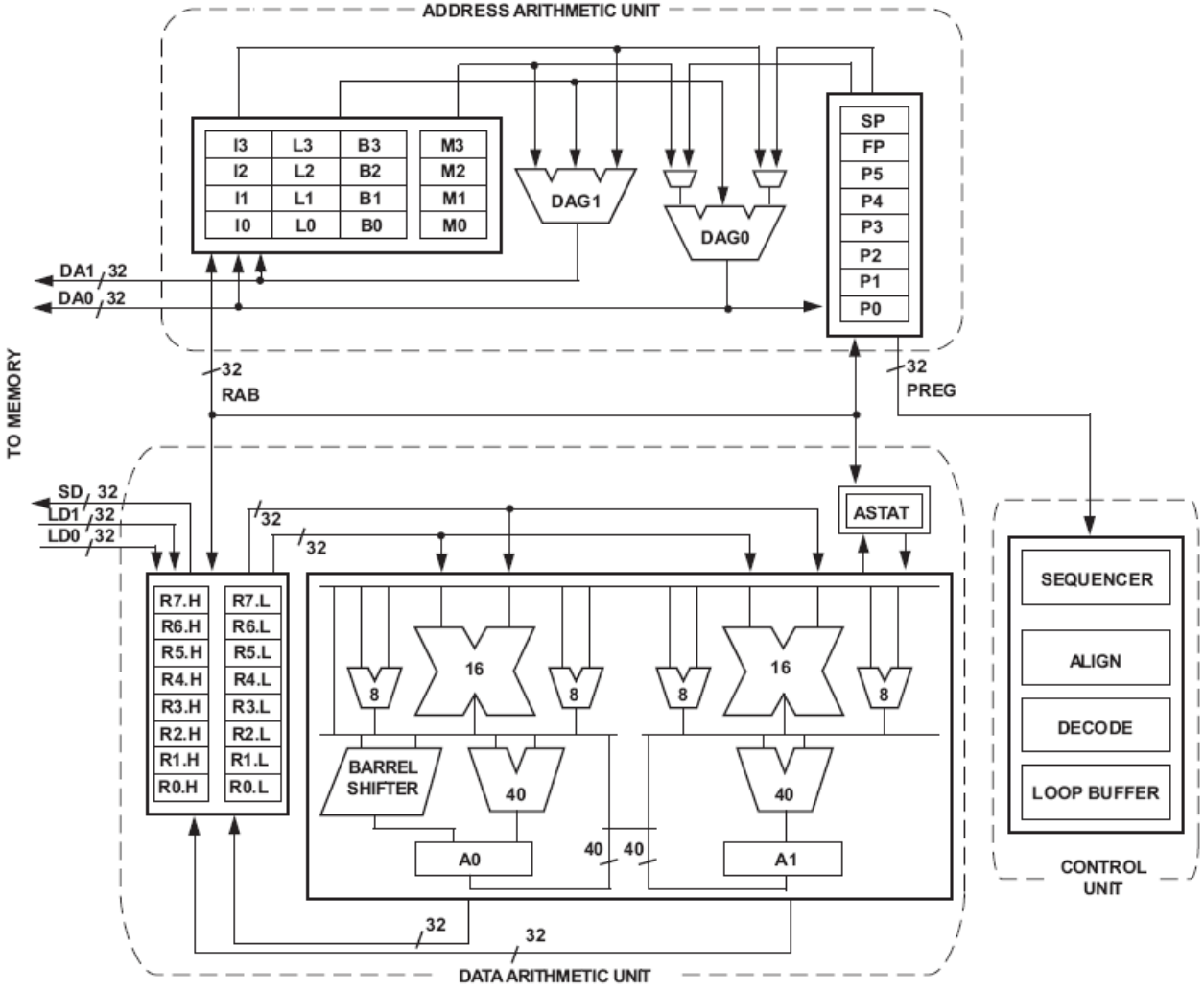


Figure 2. Blackfin Processor Core

Analog Devices, BlackFin Embedded Processor ADSP-BF534/ADSP-BF536/ADSP-BF537, 2009.

Conception de chemins des données avec l'approche RTL

- L'approche RTL (*Register Transfer Level*) est la plus populaire pour la conception de chemins des données. Elle s'accorde bien aux langages de description matérielle comme VHDL et Verilog.
- Dans l'approche RTL, le concepteur spécifie les registres du processeur, les transferts de données entre ces registres, les opérations à effectuer et les signaux de contrôle pour gérer ces activités.
- L'approche RTL peut être décomposée en quatre étapes :
  1. Analyse du problème afin de comprendre le flot des données à travers le processeur.
  2. Conception du chemin des données, identification des signaux de contrôle et d'état.
  3. Conception de l'unité de contrôle du processeur à l'aide d'une machine à états générant des signaux de contrôle.
  4. Vérification que le processeur résultant rencontre les spécifications.

Approche RTL et micro-opérations

- Une micro-opération est une opération élémentaire effectuée sur les données gardées dans des registres, en mémoire ou des données externes.
- La spécification d'une micro-opération inclut :
  - les opérandes (registres ou données externes);
  - la nature de la micro-opération à effectuer;
  - l'endroit où le résultat de la micro-opération doit être sauvegardé; et,
  - une condition à remplir pour que la micro-opération soit effectuée.
- On distingue quatre types principaux de micro-opérations :
  - les transferts entre registres;
  - les micro-opérations arithmétiques (addition, soustraction, multiplication, division, reste)
  - les micro-opérations logiques (NON, ET, OU, OUX, etc.); et,
  - le décalage.

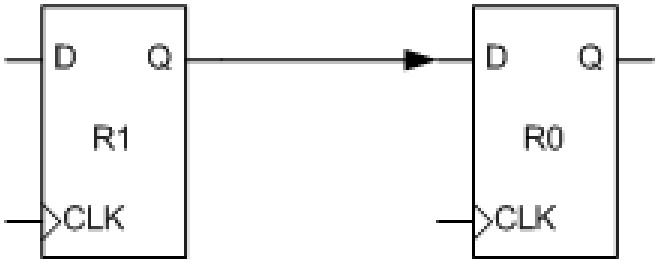
Approche RTL et micro-opérations

- La notation RTL est une forme de pseudocode.
- On peut utiliser des noms de registres (R0, R1, etc.) ou des identificateurs plus représentatifs.
- Le symbole ← indique une assignation de valeur.
- Le principe de la concurrence des opérations est intrinsèque à l'approche RTL.

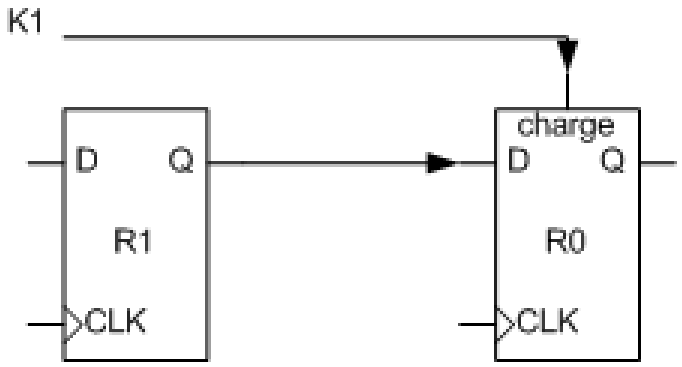
Micro-opération	Signification
R0 ← R1	Copier le contenu de R1 dans R0.
R2 ← R1 + R3	Placer la somme de R1 et de R3 dans R2.
R2 ← R2 ET R3	Placer le résultat de l'opération ET logique entre R2 et R3 dans R2.
K1 : R2 ← sll R1, 3 sll: « shift logical left »	Si le signal de contrôle K1 est actif, placer le résultat du décalage logique vers la gauche de 3 positions du registre R1 dans le registre R2; sinon, ne rien faire.
R2 ← R1 − R3, R4 ← R0	Simultanément, placer la différence entre R1 et R3 dans R2 et copier le contenu de R0 dans R4.

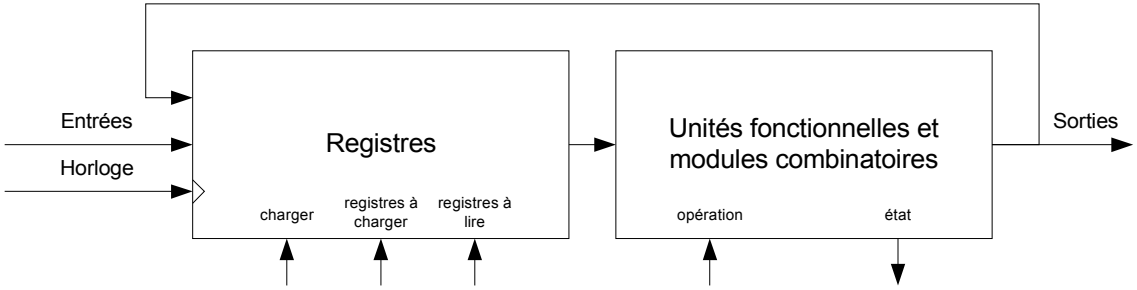
Exemples d'implémentation de micro-opérations

R0 ← R1



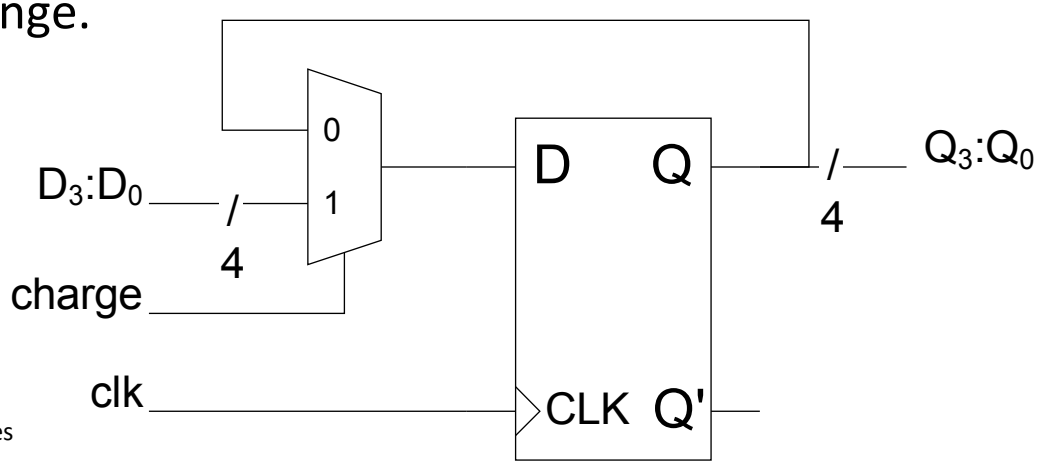
K1: R0 ← R1





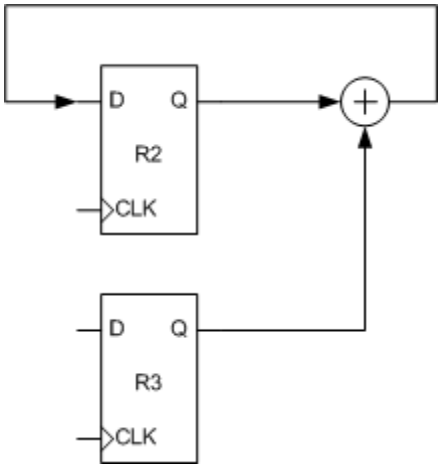
Éléments à mémoire pour chemins des données
Registre à chargement parallèle

- Un registre est l'élément à mémoire de base pour des données.
- Un registre est utilisé pour entreposer une information, encodée sur un groupe de bits, comme par exemple un octet de mémoire dans un ordinateur ou le contenu de l'accumulateur d'une calculatrice.
- Un registre est composé d'un groupe de bascules contrôlées par une horloge commune et dont les entrées et sorties partagent un identificateur commun. Chaque bascule du registre est différenciée des autres par un indice unique.
- Un registre à chargement parallèle comporte un signal de chargement qui permet de moduler le signal d'horloge. Quand ce signal est actif, le contenu du registre est modifié sur une transition de l'horloge. Dans le cas contraire, le contenu du registre reste inchangé.

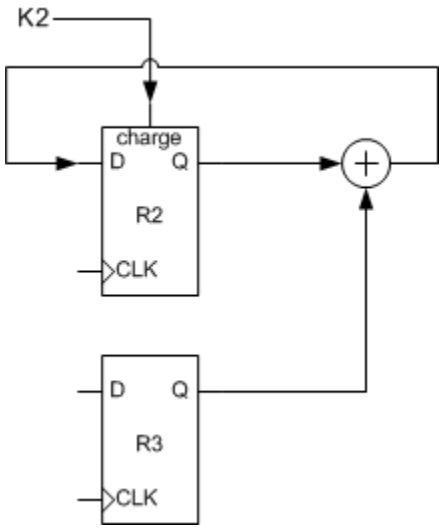


Exemples d'implémentation de micro-opérations

R2 ← R2 + R3

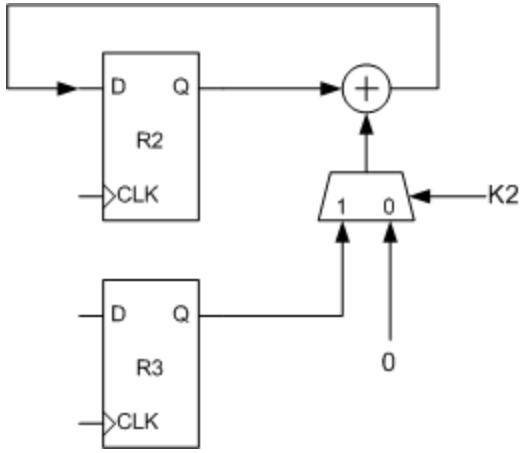


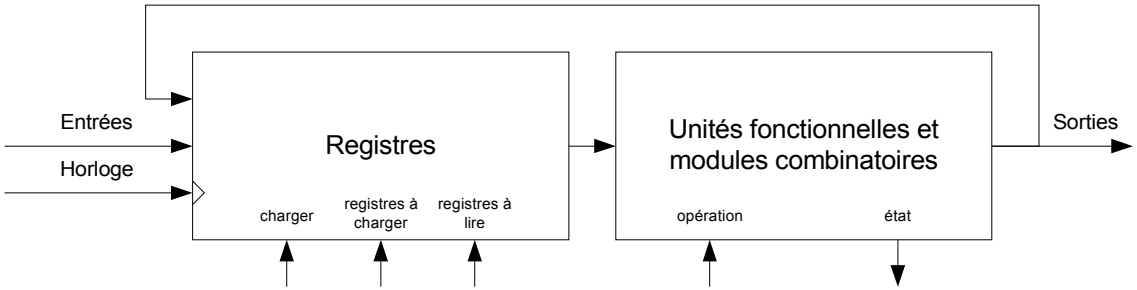
K2: R2 ← R2 + R3



K2: R2 ← R2 + R3

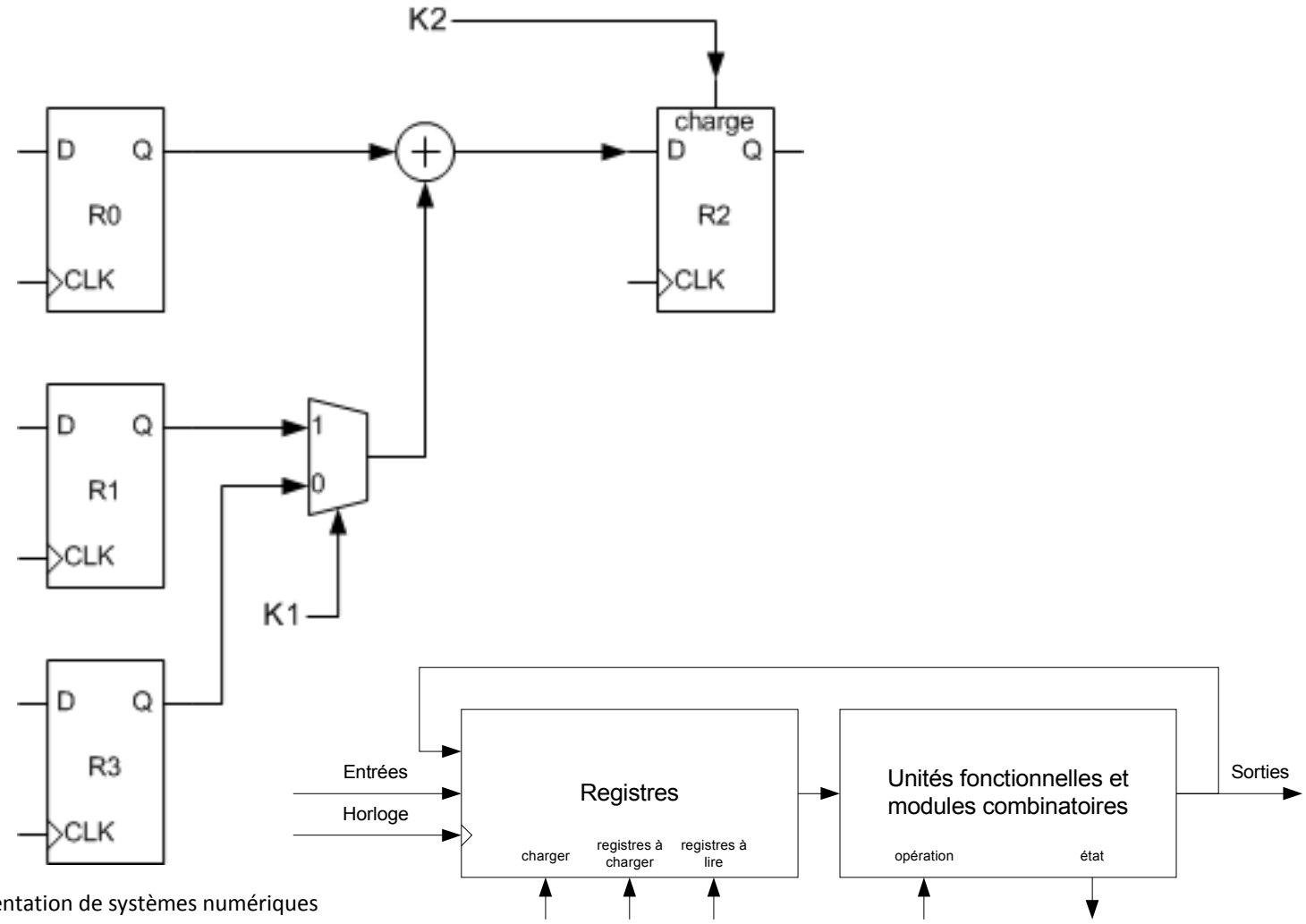
Pas recommandé: mieux vaut utiliser le signal de chargement d'une bascule.





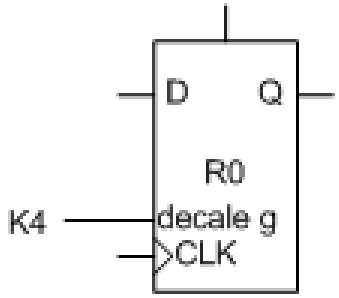
Exemple d'implémentation de micro-opérations

K2K1: R2 ← R0 + R1, K2K1': R2 ← R0 + R3

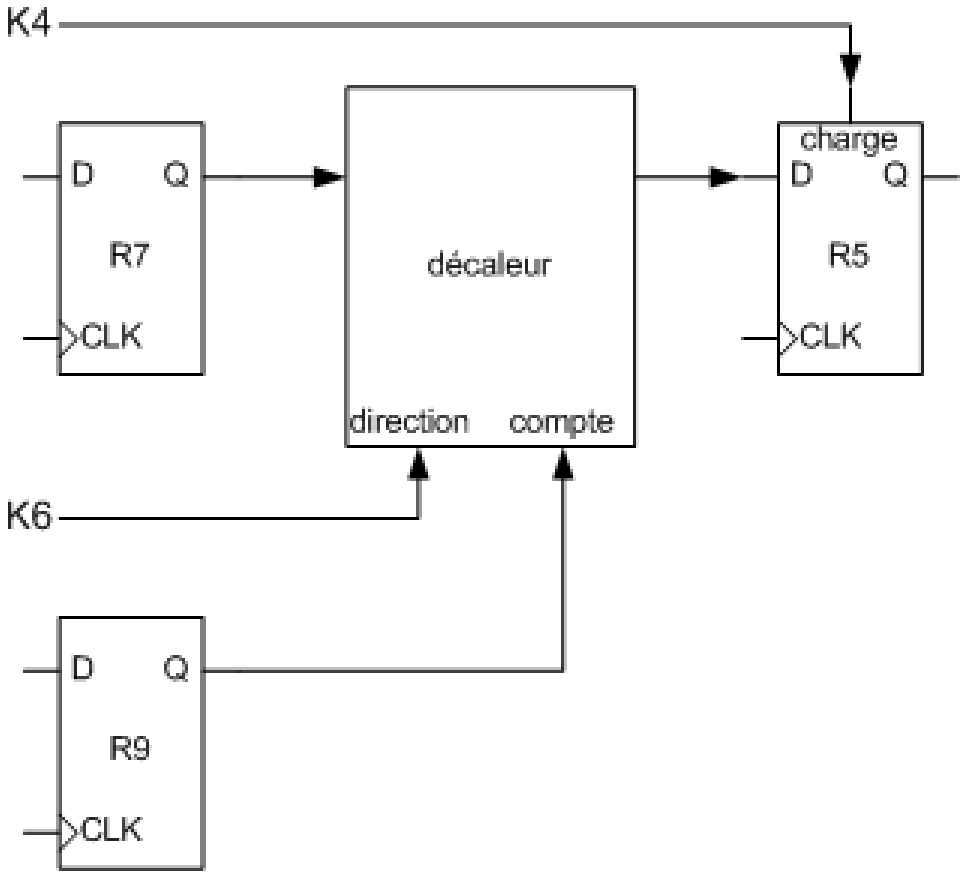


Exemples d'implémentation de micro-opérations

K4: R0 ← sll R2, 1



K4K6: R5 ← sll R7,R9 ;K4K6': R5 ← slr R7,R9



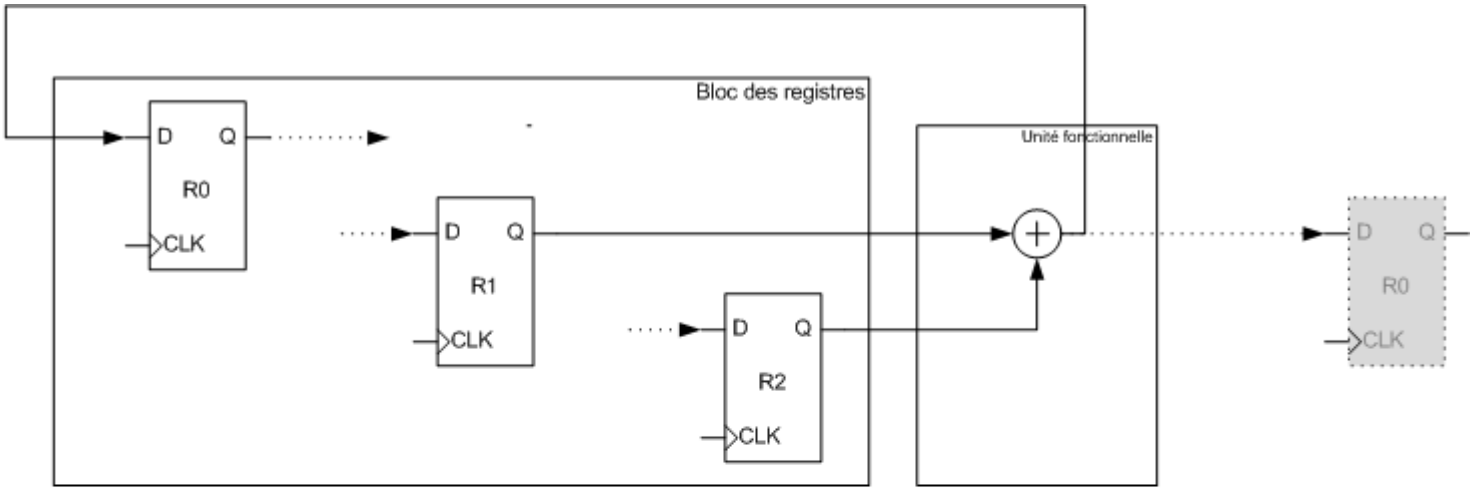
Synchronisation des opérations du chemin des données

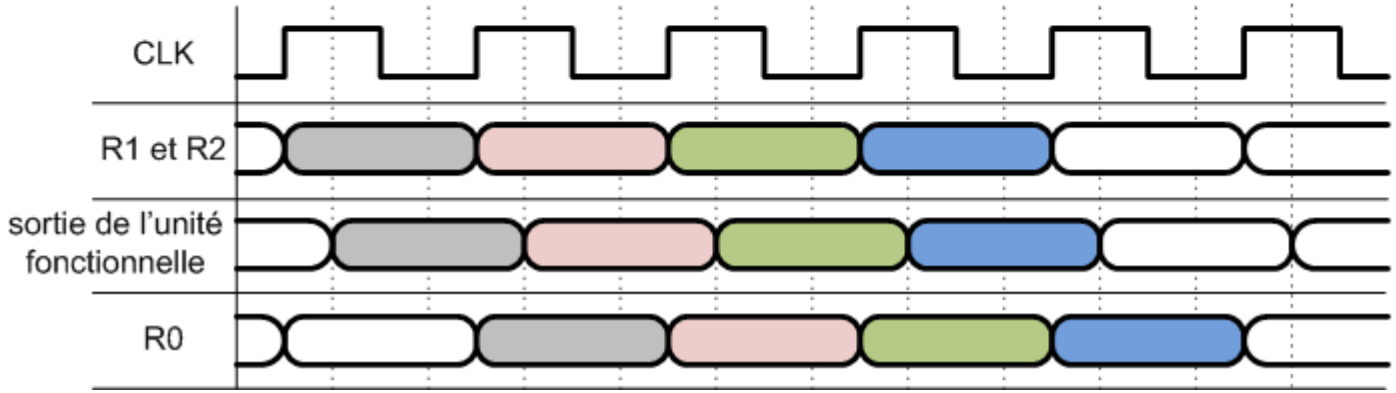
- Le bloc des registres est contrôlé par un signal d'horloge.
- À chaque coup d'horloge, le bloc des registres peut emmagasiner:
  - une nouvelle donnée provenant du port des entrées;
  - un résultat qui vient d'être calculé par l'unité fonctionnelle; ou bien,
  - effectuer un transfert entre deux registres (via l'unité fonctionnelle en général).
- Entre deux coups d'horloge, l'unité fonctionnelle:
  - lit des données provenant du bloc des registres;
  - effectue les calculs spécifiés par le code d'opération qui lui est appliqué; et,
  - applique les résultat à ses ports de sortie;

Une micro-opération ne prend effet que lors d'une transition active du signal d'horloge. Le bloc des registres ne saisit un résultat calculé par l'unité fonctionnelle que lors de la prochaine transition d'horloge.

La période d'horloge doit être suffisamment longue pour que l'unité fonctionnelle ait le temps d'effectuer son traitement.

Synchronisation des opérations du chemin des données



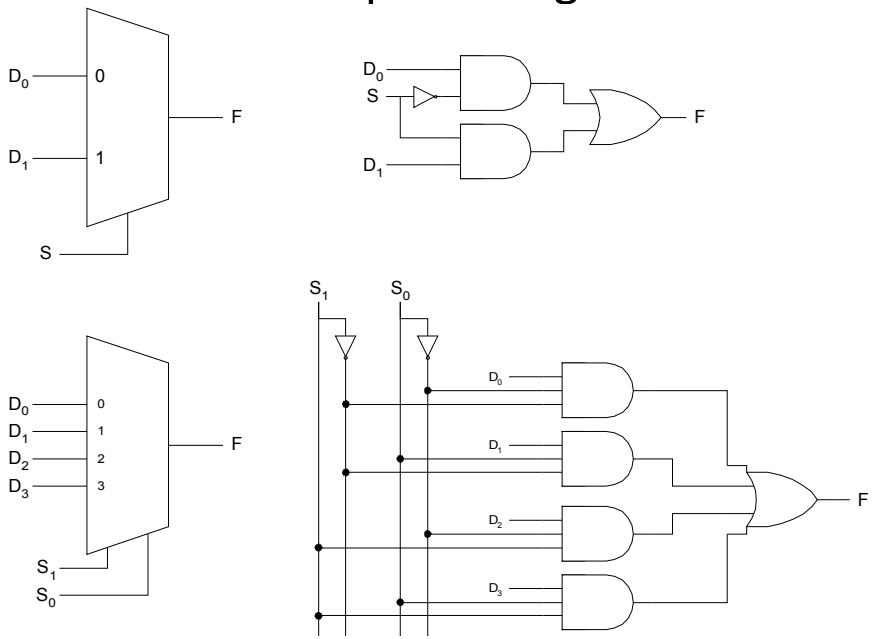


Plan

- Processeurs: introduction
- **Modules combinatoires**:
  - multiplexeurs, décodeurs et encodeurs
- Éléments à mémoire pour chemins des données:
  - registres à chargement parallèle et à décalage, bloc de registres, RAM
- Unités fonctionnelles:
  - unités arithmétiques, unités logiques, comparateurs, compteurs

Modules combinatoires utiles
Multiplexeur

- Un multiplexeur permet de choisir un seul signal à partir d'un ensemble de signaux, selon la valeur d'un signal de contrôle.
- Un multiplexeur a :
  - un groupe de signaux d'entrée *D*;
  - un groupe de signaux de contrôle *S* (pour *sélection*); et,
  - un signal de sortie *F*.
- Le signal de sortie est égal au signal d'entrée choisi par les signaux de contrôle.



Modules combinatoires utiles
Multiplexeur 2:1 en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity mux21 is
    port(D0, D1, S : in STD_LOGIC; F : out STD_LOGIC);
end mux21;

architecture flotDeDonnees of mux21 is
begin

   with S select
   F <= D0 when '0', D1 when others;

end flotDeDonnees;
```

Pas nécessaire de donner les équations booléennes.
Assignation choisie avec tous les cas couverts.

Modules combinatoires utiles
Multiplexeur général en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux is
    generic (
        n : positive := 3 -- nombre de signaux de contrôle
    );
    port (
        D : in std_logic_vector(2 ** n - 1 downto 0);
        S: in unsigned(n - 1 downto 0);
        F : out std_logic
    );
end mux;

architecture comportementale of mux is

begin

    process (D, S)
    begin
        F <= D(to_integer(S));
    end process;

end comportementale;
```

Description comportementale très compacte.

Normalement on ne décrirait pas un multiplexeur dans une entité séparée, on utiliserait plutôt le patron de code dans un module.

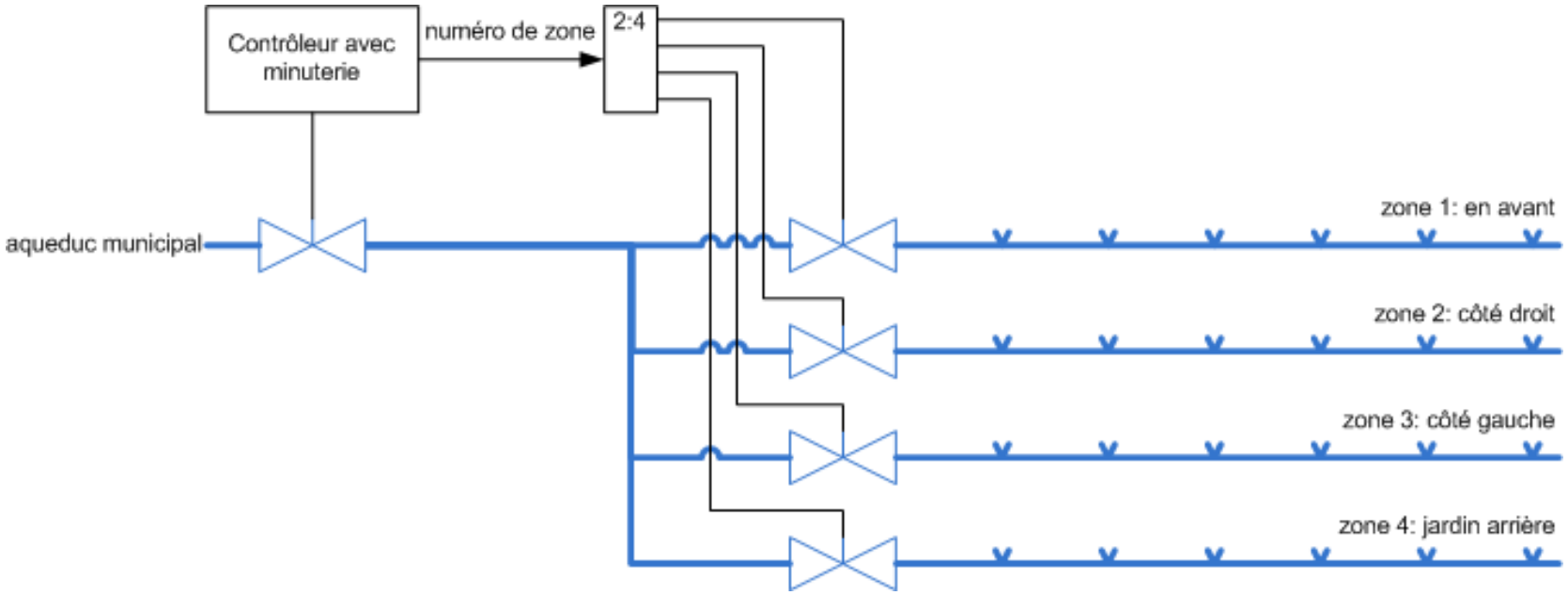
Rappel: loquet D en VHDL

- Un loquet D peut-être modélisé en VHDL par un énoncé if-then à l'intérieur d'un processus.
- Le processus doit avoir dans sa liste de sensitivité le signal de contrôle *G* et le signal de donnée *D*.
- Le signal *D* est assigné à la sortie *Q* quand le signal de contrôle *G* est actif (mode transparent)

```
library ieee;
use ieee.std_logic_1164.all;
entity loquetD is
    port (
        G : in STD_LOGIC;   -- contrôle
        D : in STD_LOGIC;   -- donnée
        Q : out STD_LOGIC
    );
end loquetD;
architecture loquetD of loquetD is
begin
    process(G, D) is
    begin
        if (G = '1') then
            Q <= D;
--      else   -- implicite, infère élément à mémoire
--             -- ne pas changer Q
        end if;
    end process;
end loquetD;
```

Patron de code spécial reconnu par les synthétiseurs pour signifier un loquet.

Modules combinatoires utiles
Décodeur: exemple d'utilisation



Exemple: système de gicleurs automatiques autour d'une résidence.
La pression de l'aqueduc municipal est insuffisante pour activer tous les gicleurs en même temps.
Les gicleurs sont divisés en quatre zones.
Une seule zone doit arroser à la fois.
Chaque zone est munie d'une valve.
Un décodeur permet d'ouvrir une seule valve à la fois.
Une valve principale doit aussi être ouverte.

Modules combinatoires utiles
Décodeur

- Un décodeur active un signal spécifique correspondant à un code numérique en particulier.
- Un décodeur a *n* signaux d'entrée et 2^n signaux de sortie.
  - Chacun des signaux de sortie correspond à un des mintermes et maxtermes composés des signaux d'entrée.
  - Une et une seule ligne de sortie est active à un moment donné.
  - Le numéro de la ligne active correspond à la valeur binaire appliquée aux lignes d'entrée.
  - Selon les décodeurs, la ligne active pourra être à une valeur 0 ou une valeur 1, et toutes les autres lignes seront à l'autre valeur.

#	A_2	A_1	A_0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	0	1	0	0
3	0	1	1	0	0	0	0	1	0	0	0
4	1	0	0	0	0	0	1	0	0	0	0
5	1	0	1	0	0	1	0	0	0	0	0
6	1	1	0	0	1	0	0	0	0	0	0
7	1	1	1	1	0	0	0	0	0	0	0

Table de vérité d'un décodeur 3:8

(A2, A1, A0) sont les entrées.

(F7, F6, …, F0) sont les sorties.

Modules combinatoires utiles
Décodeur 3:8 en VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity decodeur38 is
   port(
      A : in std_logic_vector(2 downto 0);
      F: out std_logic_vector(7 downto 0)
   );
end decodeur38;

architecture flotDeDonnees of decodeur38 is

begin
   with A select F <=
      "00000001" when "000",
      "00000010" when "001",
      "00000100" when "010",
      "00001000" when "011",
      "00010000" when "100",
      "00100000" when "101",
      "01000000" when "110",
      "10000000" when "111",
      (others => 'X') when others;

end flotDeDonnees;
```

Une assignation choisie spécifie les huit cas possibles du signal d'entrée F.
L'utilisation de la clause others permet de rendre le modèle plus robuste à la simulation. En effet, le type std_logic peut prendre des valeurs autres que '0' et '1' – voir notes de cours 3. Lors de la simulation, si le signal F prend une valeur comme « X1W », la sortie du décodeur sera un vecteur de 'X'.
L'expression (others => 'X') permet d'assigner la valeur 'X' à chacun des éléments du vecteur F.

Parenthèse: agrégats en VHDL

- En VHDL, un agrégat est une opération de base qui combine plusieurs valeurs en une valeur composée.

```
-- Exemple 1
variable Data_1 : BIT_VECTOR (0 to 3) := ('0','1','0','1'); -- assignation positionelle

-- Exemple 2
variable Data_2 : BIT_VECTOR (0 to 3) := (1=>'1',0=>'0',3=>'1',2=>'0'); -- assignation nommée

-- Exemple 3
signal Data_Bus : Std_Logic_Vector (15 downto 0);
. . .
Data_Bus <= (15 downto 8 => '0', 7 downto 0 => '1'); -- assignation positionelle avec gammes d'indices

-- Exemple 4
type Status_Record is record
     Code : Integer;
     Name : String (1 to 4);
end record;
variable Status_Var : Status_Record := (Code => 57, Name => "MOVE");

-- Exemple 5
signal Data_Bus : Std_Logic_Vector (15 downto 0);
. . .
Data_Bus <= (14 downto 8 => '0', others => '1'); -- utilisation du choix others (en dernier)

-- Exemple 6
signal Data_Bus : Std_Logic_Vector (15 downto 0);
. . .
Data_Bus <= (others => 'Z'); -- utilisation du choix others par lui-même
```

Modules combinatoires utiles
Décodeur général en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity decodeur is
   generic (
      n : positive := 3; -- nombre de signaux d'entrée
      valeurActive : std_logic := '1'
   );
   port(
      A : in std_logic_vector(n - 1 downto 0);
      F: out std_logic_vector(2 ** n - 1 downto 0)
   );
end decodeur;

architecture comportementale of decodeur is

begin

   process(A)
   begin
      F <= (others => not(valeurActive));
      F(to_integer(unsigned(A))) <= valeurActive;
   end process;

end comportementale;
```

On exploite le fait que les énoncés sont exécutés de façon séquentielle à l'intérieur d'un processus. Pour débuter, toutes les sorties sont désactivées.
Ensuite, seule celle dont le numéro correspond au signal d'entrée est activée.

.

Modules combinatoires utiles
Encodeur à priorité

- Un encodeur identifie un signal actif parmi un ensemble de signaux, et produit un code qui correspond à ce signal actif.
- Un encodeur fonctionne de façon contraire à un décodeur.
  - Il a *n* lignes de sortie et 2^n lignes d'entrée.
  - Le code à la sortie représente le numéro de la ligne qui est active.
  - Un signal de sortie spécial indique si au moins une des lignes en entrée est active.
- Un encodeur à priorité permet d'avoir plus d'une ligne d'entrée active à la fois. La priorité peut être accordée à la ligne ayant le plus grand ou le plus petit numéro, ou selon un autre choix.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0	V
0	0	0	0	0	0	0	0	-	-	-	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	-	0	0	1	1
0	0	0	0	0	1	-	-	0	1	0	1
0	0	0	0	1	-	-	-	0	1	1	1
0	0	0	1	-	-	-	-	1	0	0	1
0	0	1	-	-	-	-	-	1	0	1	1
0	1	-	-	-	-	-	-	1	1	0	1
1	-	-	-	-	-	-	-	1	1	1	1

Table de vérité partielle d'un encodeur 8:3

(D7, D6, …, D0) sont les entrées.

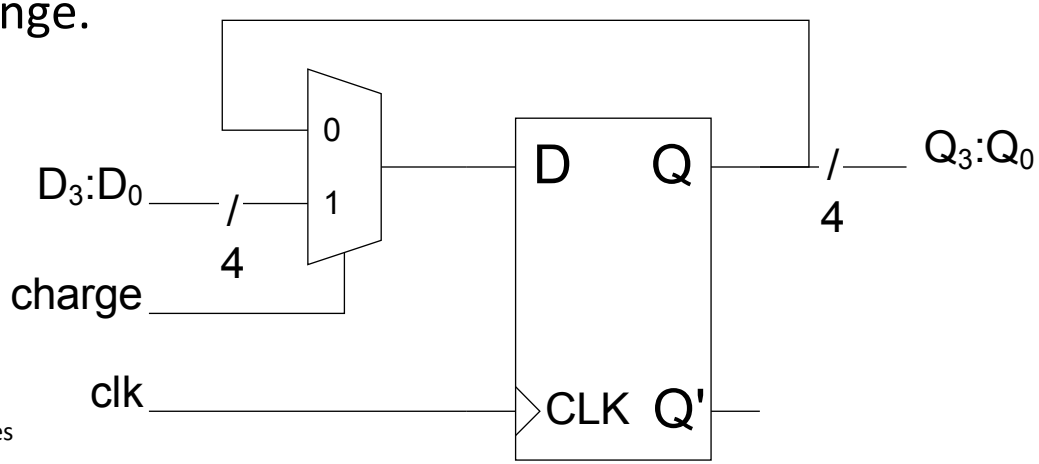
(A2, A1, A0 et V) sont les sorties.

Plan

- Processeurs: introduction
- Modules combinatoires:
  - multiplexeurs, décodeurs et encodeurs
- Éléments à mémoire pour chemins des données:
  - registres à chargement parallèle et à décalage, bloc de registres, RAM
- Unités fonctionnelles:
  - unités arithmétiques, unités logiques, comparateurs, compteurs

Éléments à mémoire pour chemins des données
Registre à chargement parallèle

- Un registre est l'élément à mémoire de base pour des données.
- Un registre est utilisé pour entreposer une information, encodée sur un groupe de bits, comme par exemple un octet de mémoire dans un ordinateur ou le contenu de l'accumulateur d'une calculatrice.
- Un registre est composé d'un groupe de bascules contrôlées par une horloge commune et dont les entrées et sorties partagent un identificateur commun. Chaque bascule du registre est différenciée des autres par un indice unique.
- Un registre à chargement parallèle comporte un signal de chargement qui permet de moduler le signal d'horloge. Quand ce signal est actif, le contenu du registre est modifié sur une transition de l'horloge. Dans le cas contraire, le contenu du registre reste inchangé.



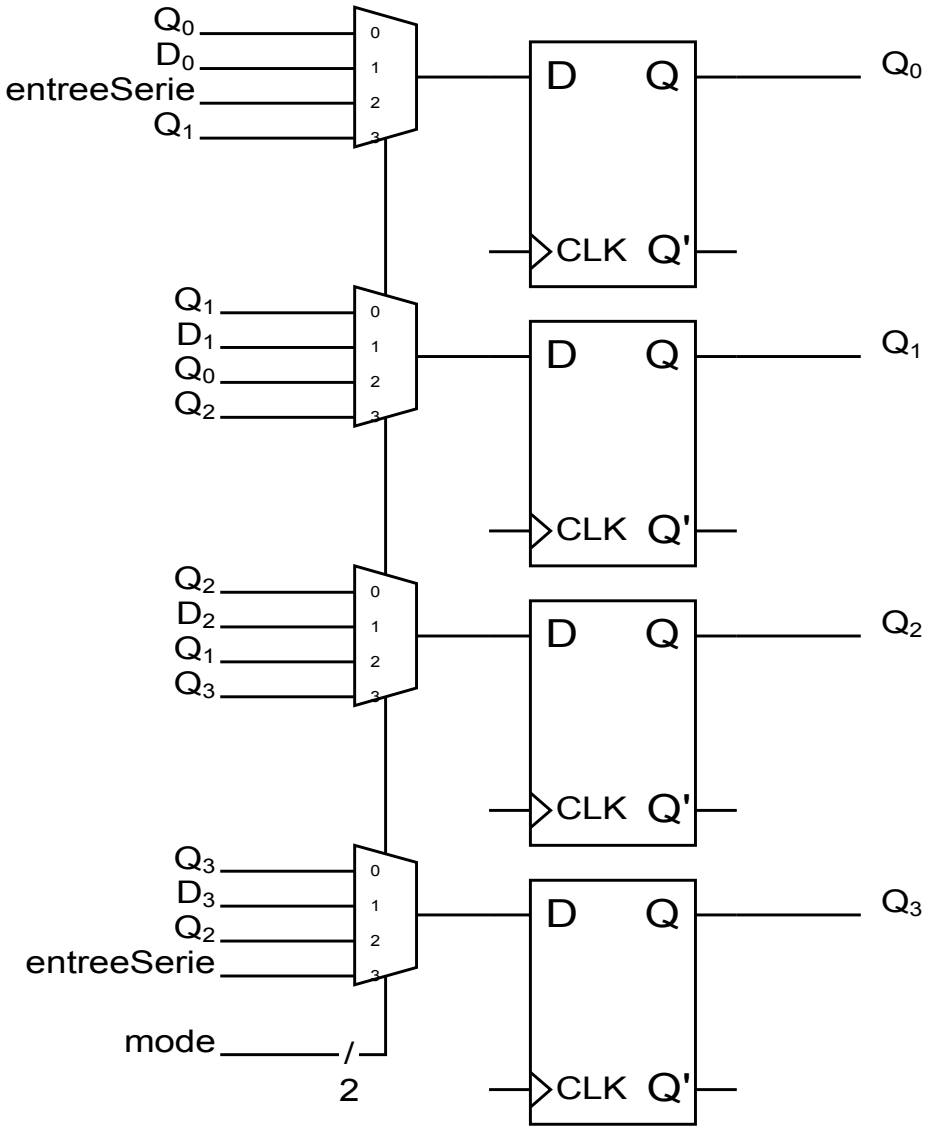
Registre à chargement parallèle

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity registre is
   generic (
      W : integer := 8
   );
   port(
      reset : in STD_LOGIC;
      CLK : in STD_LOGIC;
      charge : in STD_LOGIC;
      D : in STD_LOGIC_VECTOR(W - 1 downto 0);
      Q : out STD_LOGIC_VECTOR(W - 1 downto 0)
   );
end registre;
architecture arch of registre is
begin
   process (CLK, reset)
   begin
      if reset='0' then
         Q <= (others => '0');
       elsif CLK='1' and CLK'event then
         if charge='1' then
            Q <= D;
         end if;
      end if;
   end process;
end arch;
```

Éléments à mémoire pour chemins des données
Registre à décalage

- Un registre à décalage peut décaler ses bits vers la gauche ou la droite.
- En général, un registre à décalage ne décale son contenu que d'une position par coup d'horloge.
- Un registre à décalage de 1 position peut être construit par une cascade de bascules D dont la sortie est reliée à l'entrée de la bascule suivante.
- Un registre à décalage peut être utilisé pour faire une conversion entre les formats série et parallèle et est donc une composante fondamentale de plusieurs circuits de communication.
- Un registre à décalage permet aussi d'effectuer:
  - la multiplication par un puissance de deux, qui correspond à un décalage de bits vers la gauche;
  - la division par une puissance de deux, qui correspond à un décalage vers la droite.

Registre de 4 bits à décalage de 1 position et à chargement parallèle



Les multiplexeurs partagent le même signal de contrôle ‘mode’.

Ce module est très polyvalent, il a quatre modes de fonctionnement.

- mémoire
- chargement parallèle
- décalage vers la gauche (Q0 vers Q3)
- décalage vers la droite (Q3 vers Q0)

Registre de 4 bits à décalage et à chargement parallèle Modèle VHDL

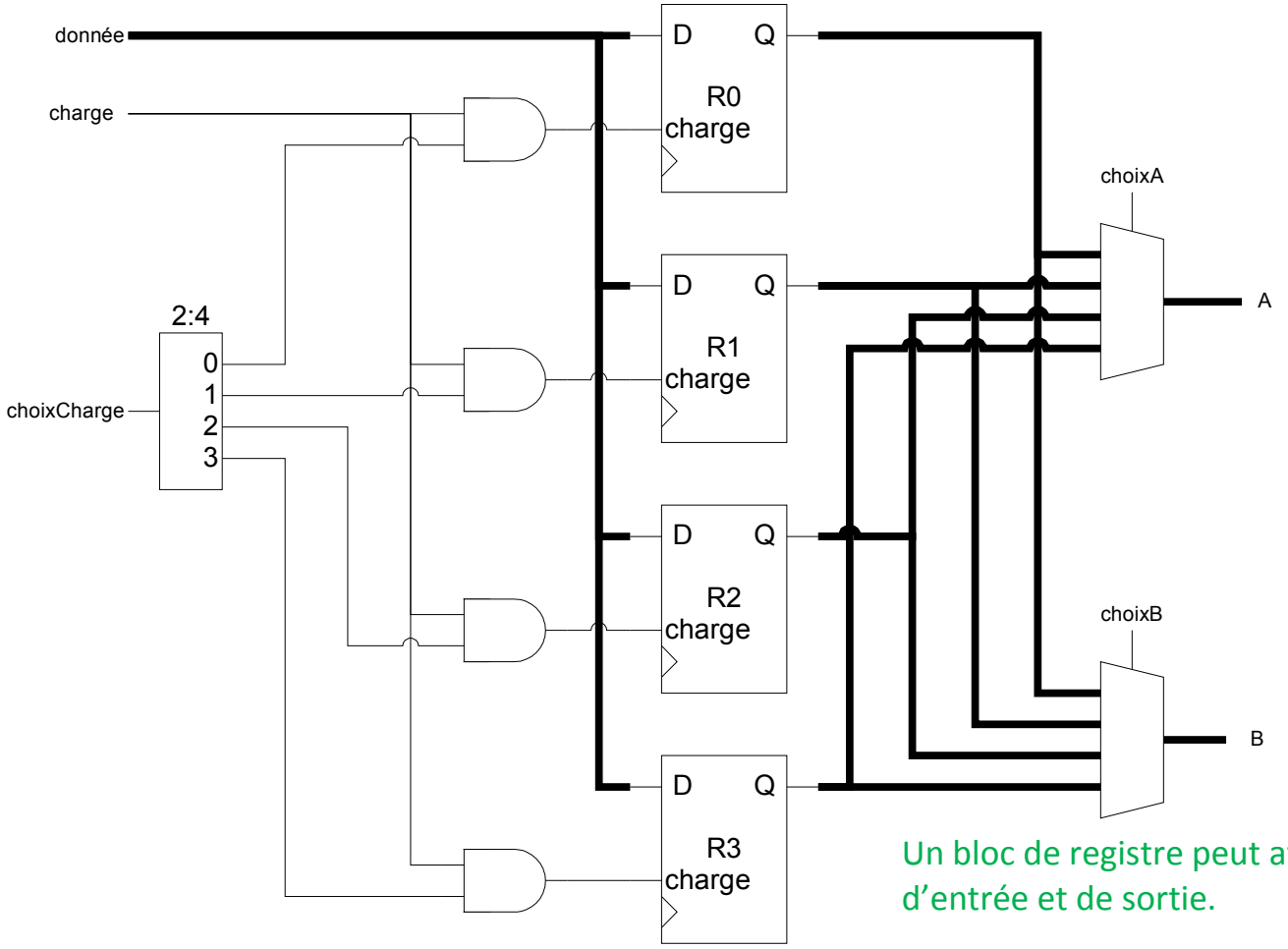
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity registreadecallage is
    generic (
        W : integer := 8 -- nombre de bits du registre
    );
    port(
        reset : in STD_LOGIC;
        CLK : in STD_LOGIC;
        mode : in STD_LOGIC_VECTOR(1 downto 0); -- mode
        entreeSerie : in STD_LOGIC; -- entree serielle
        D : in STD_LOGIC_VECTOR(W - 1 downto 0);
        Q : out STD_LOGIC_VECTOR(W - 1 downto 0)
    );
end registreadecallage;
architecture arch of registreadecallage is
begin
    process (CLK, reset)
    variable Qinterne : STD_LOGIC_VECTOR(W - 1 downto 0);
    begin
        if reset='0' then
            Qinterne := (others => '0');
            Q <= (others => '0');
         elsif CLK='1' and CLK'event then
             case mode is
               when "00" => -- garde
                   Qinterne := Qinterne;
               when "01" => -- charge
                   Qinterne := D;
               when "10" => -- decale gauche
                   Qinterne := Qinterne(W - 2 downto 0)  & entreeSerie;
               when "11" => -- decale droite
                   Qinterne := entreeSerie & Qinterne(W - 1 downto 1);
               when others =>
                   Qinterne := Qinterne;
            end case;
            Q <= Qinterne;
        end if;
    end process;
end arch;
```

Utilisation d'une variable Qinterne parce qu'on ne peut pas lire un port de sortie (Q) en VHDL.

Éléments à mémoire pour chemins des données
Bloc de registres

- Un bloc de registres (*register file*) regroupe plusieurs registres d'un chemin des données.
- C'est effectivement une petite mémoire qui rassemble plusieurs des données du circuit.
- Un bloc de registres composé de plusieurs registres de largeurs identiques qui partagent des ports d'entrée et de sortie.
- Le nombre de registres peut varier de 1 seul à 1024, et leur largeur peut varier de 4 à 128 bits.
- Par exemple, pour un microprocesseur de 32 ou 64 bits, le nombre réfère à la largeur des registres du bloc des registres.
- Le bloc des registres a en général plusieurs ports d'entrée et de sortie indépendants. Des signaux de contrôle permettent de choisir quel registre est dirigé à chacune des sorties, et quels registres doivent être chargés.

Éléments à mémoire pour chemins des données
Bloc de registres

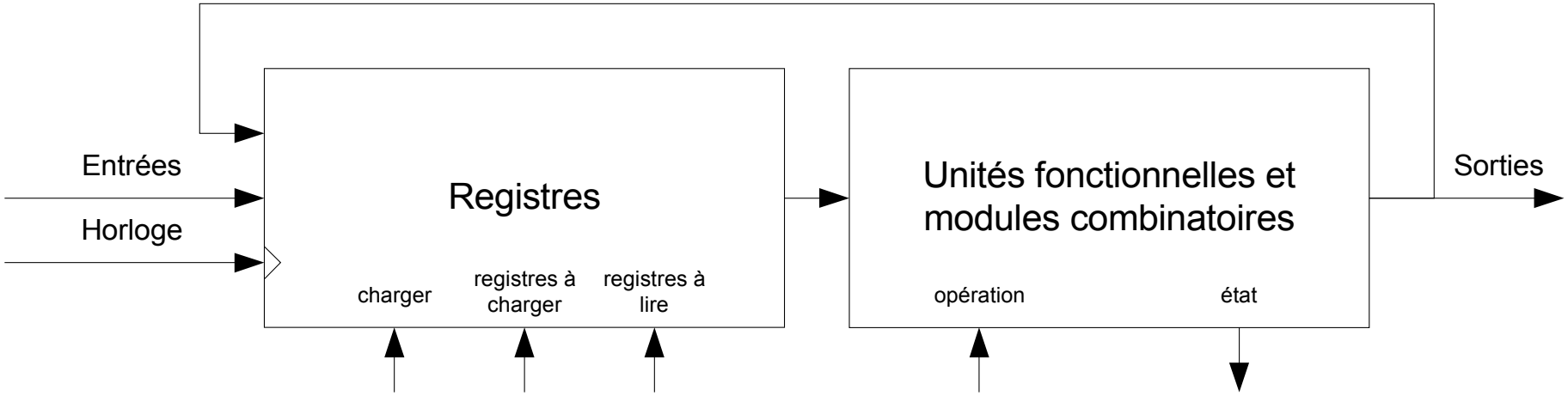


Un bloc de registre peut avoir plusieurs ports d'entrée et de sortie.

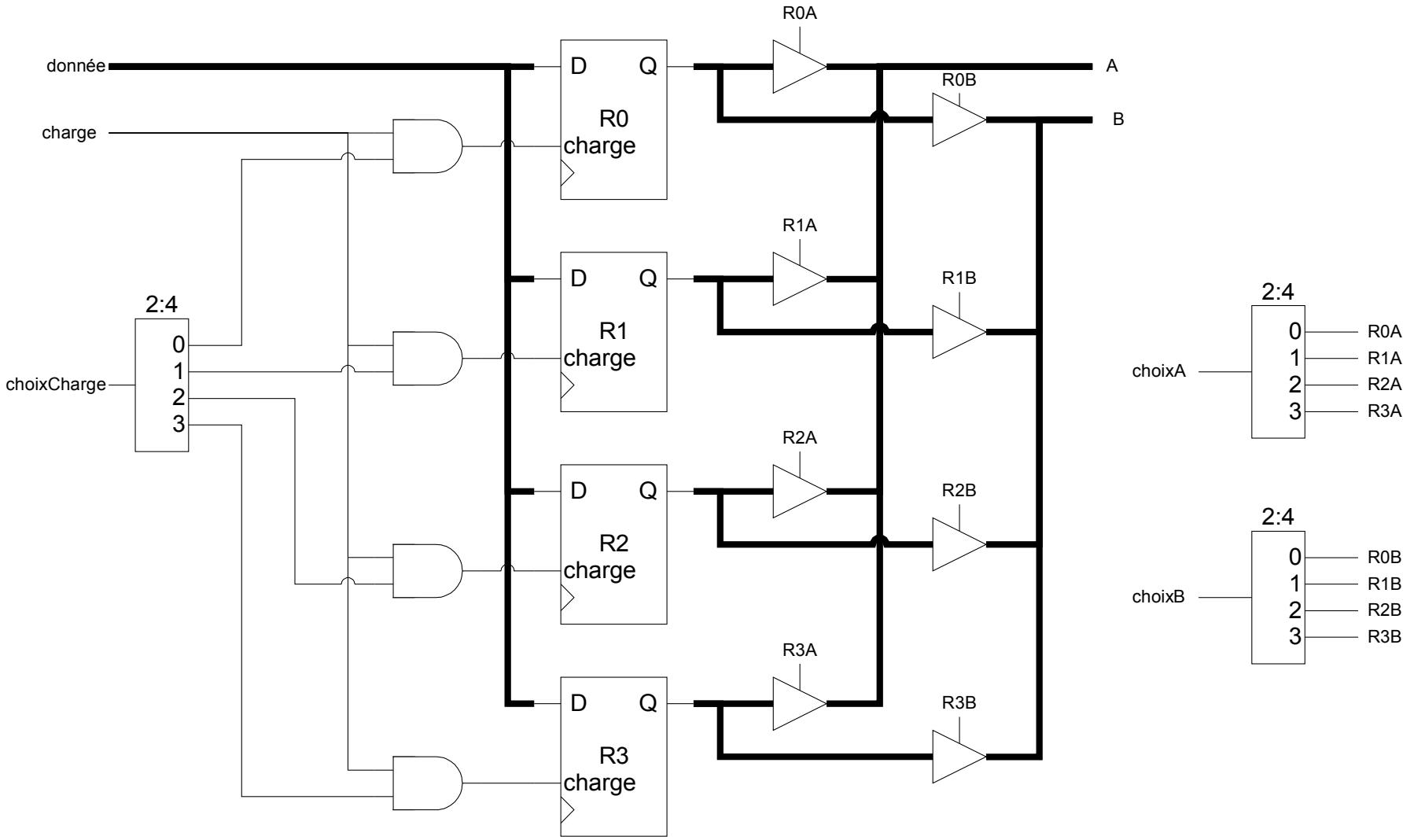
Un registre peut à la fois être la cible d'une écriture et la source pour l'un des ports de sortie.

Version coûteuse en multiplexeurs!

Rappel: architecture d'un chemin des données



Éléments à mémoire pour chemins des données
Bloc de registres: version avec tampons à trois états



Éléments à mémoire pour chemins des données
Bloc de registres: code VHDL

```
-- dans la partie déclarative de l'architecture
type lesRegistres_type is array(0 to Nreg - 1) of signed(Wd - 1 downto 0);
signal lesRegistres : lesRegistres_type;
signal A : signed(Wd - 1 downto 0);
signal choixA : integer range 0 to Nreg - 1;
signal B : signed(Wd - 1 downto 0);
signal choixB : integer range 0 to Nreg - 1;
signal donnee : signed(Wd - 1 downto 0);
signal choixCharge : integer range 0 to Nreg - 1;
signal charge : std_logic;

-- dans le corps de l'architecture
   process (CLK, reset)
   begin
      if rising_edge(CLK) then
         if reset = '1' then
            lesRegistres <= (others => (others => '0'));
         else
            if charge = '1' then
               lesRegistres(choixCharge) <= donnee;
            end if;
         end if;
      end if;
   end process;

   -- signaux de sortie du bloc des registres
   A <= lesRegistres(choixA);
   B <= lesRegistres(choixB);
```

Ici: un port d'écriture, deux ports de lecture

Implémentation par multiplexeurs ou tampons à trois états?

Expression (others=>(others=>'0')) parce que le bloc des registres est une structure de données à deux dimensions.

Wd et Nd sont des `generic` de l'entité.

Éléments à mémoire pour chemins des données
Mémoire vive

- Une mémoire vive peut être vue comme un bloc de registres de très grande taille, avec:
  - un (parfois deux) port de sortie
  - un (parfois deux) port d'entrée
  - un (parfois deux) port d'adresse
  - un port de contrôle de l'opération.
- La description d'une mémoire des données en VHDL peut prendre plusieurs formes, selon une multitude de paramètres. Ceux-ci incluent, entre autres :
  - le nombre de ports d'entrée et de sortie;
  - le fait que les sorties soient synchrones ou asynchrones;
  - le nombre de cycles nécessaires à la mémoire pour déplacer des données;
  - la présence de signaux d'activation; et,
  - la spécification de valeurs initiales.

Il faut consulter la documentation du synthétiseur pour adopter le bon patron de code VHDL correspondant aux paramètres de la mémoire RAM désirée.

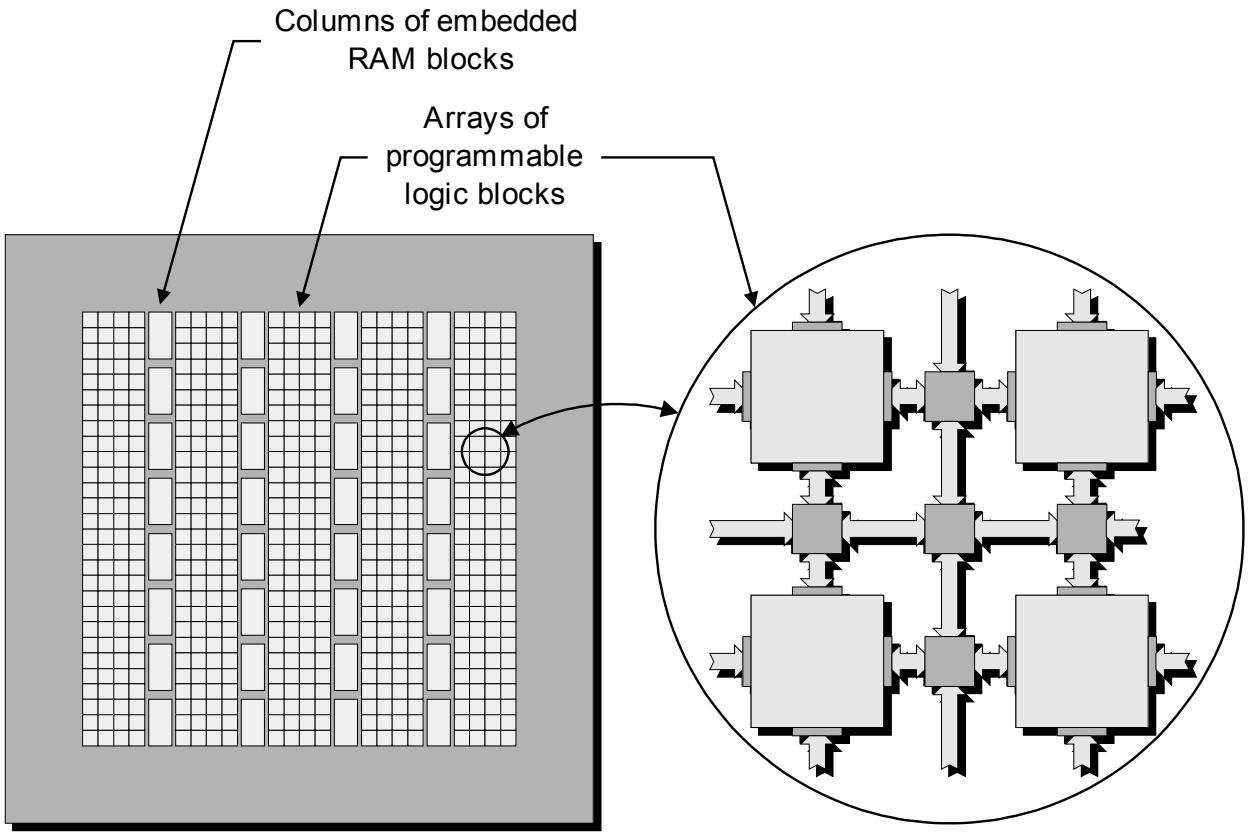
Éléments à mémoire pour chemins des données
Mémoire vive

```
-- dans la partie déclarative de l'architecture
type memoireDonnees_type is array(0 to 2 ** Md - 1) of signed(Wd - 1 downto 0);
signal memoireDonnees : memoireDonnees_type;
signal sortieMemoireDonnees : signed(Wd - 1 downto 0);
signal adresseMemoireDonnees : integer range 0 to 2 ** Md - 1;
signal lectureEcritureN : std_logic;


-- dans le corps de l'architecture
   -- mémoire des données
   process (CLK)
   begin
      if rising_edge(CLK) then
         if lectureEcritureN = '0' then
            memoireDonnees(adresseMemoireDonnees) <= B;
         end if;
      end if;
   end process;

   sortieMemoireDonnees <= memoireDonnees(adresseMemoireDonnees);
```

Important: pas de reset!
Si on utilise un reset, l'outil de synthèse implémentera la mémoire avec des bascules (faible densité).
Sans reset, l'outil de synthèse pourra implémenter la mémoire dans des blocs RAM dédiés du FPGA (haute densité).

Comment se synthétise la mémoire dans un FPGA?

- Colonnes de blocs de mémoire intégrées à travers les CLBs. (Xilinx: *Block RAM*)
- LUTs des CLBs (Xilinx: *Distributed RAM*)



Plan

- Processeurs: introduction
- Modules combinatoires:
  - multiplexeurs, décodeurs et encodeurs
- Éléments à mémoire pour chemins des données:
  - registres à chargement parallèle et à décalage, bloc de registres, RAM
- Unités fonctionnelles:
  - unités arithmétiques, unités logiques, comparateurs, compteurs

Unités fonctionnelles
Unité arithmétique

- Une unité arithmétique effectue des opérations arithmétiques entre des opérandes sous le contrôle d'un code d'opération.
- Les unités arithmétiques sont au cœur de tous les microprocesseurs.
- Les synthétiseurs de VHDL sur le marché reconnaissent les opérations arithmétiques d'addition, soustraction et multiplication et infèrent correctement des structures matérielles pour les implémenter. Les opérateurs correspondants sont '+', '-' et '*'.
- Les opérations de division, reste et modulo sont supportées uniquement lorsque le deuxième opérande est une constante égale à une puissance de deux.
- Ce support permet au concepteur d'élever de façon notable le niveau d'abstraction de modélisation de circuits arithmétiques en évitant d'avoir à penser aux menus détails de leur implémentation.
- Il est utile de bien comprendre les complexités relatives de ces opérations de façon à pouvoir faire des choix de conception éclairés.

Unités fonctionnelles
Unité arithmétique

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity unitearithmetique is
   generic (
      W : positive := 8 -- largeur des opérandes
   );
   port(
      A, B : in signed(W - 1 downto 0); -- les opérandes
      choix : in std_logic_vector(2 downto 0); -- le sélecteur d'opération
      F : out signed(W - 1 downto 0) -- le résultat
   );
end unitearithmetique;
architecture arch of unitearithmetique is
begin
   process(A, B, choix)
   variable t : signed(2 * W - 1 downto 0);
   begin
      t := A * B;
      case to_integer(unsigned(choix)) is
         when 0 => F <= A + B;
         when 1 => F <= A - B;
         when 2 => F <= A + B + 1;
         when 3 => F <= A + 1;
         when 4 => F <= abs(A);
         when 5 => F <= -A;
         when 6 => F <= t(2 * W - 1 downto W);
         when 7 => F <= t(W - 1 downto 0);
         when others => F <= (others => 'X');
      end case;
   end process;
end arch;
```

Unités fonctionnelles
Unité arithmétique: quels types utiliser?

- **`real`: ça dépend**
  - pour des valeurs constantes: ok
  - pour des registres et valeurs intermédiaires: non! trop précis et pas synthétisable
- **`integer`, `natural`, `positive`: acceptables**
  - Bien supportés par les synthétiseurs pour les opérations arithmétiques.
  - Bonne abstraction par rapport à un vecteur de bits.
  - Important de spécifier la gamme de valeurs possibles de façon à contraindre les ressources matérielles utilisées pour les représenter.
- **`signed`, `unsigned`: acceptables**
  - Définis dans le package normalisé numeric_std, comme des tableaux de std_logic.
  - Bien supportés par les outils de synthèse.
  - Le package numeric_std redéfinit les opérateurs de VHDL pour ces deux types.

Unités fonctionnelles
Unité logique

- Les unités logiques effectuent une opération logique sur des opérandes sous le contrôle d'un signal externe.
- Elles sont au cœur de tout microprocesseur.
- Par exemple, on peut vouloir effectuer une des opérations ET, OU, OUX, ou NON-ET entre deux vecteurs de façon dynamique.

Pour les types, on peut utiliser std_logic_vector, signed ou unsigned.

Le package std_logic_1164 redéfinit les opérateurs logiques and, nand, or, nor, xor, xnor et not pour les objets de type std_logic et std_logic_vector.

Le package numeric_std fait de même pour les types unsigned et signed.

Unités fonctionnelles
Unité logique

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity unitelogique is
   generic (
      W : positive := 8 -- largeur des opérandes
   );
   port(
      A, B : in std_logic_vector(W - 1 downto 0); -- les opérandes
      choix : in std_logic_vector(2 downto 0); -- le sélecteur d'opération
      F : out std_logic_vector(W - 1 downto 0) -- le résultat
   );
end unitelogique;
architecture arch of unitelogique is
begin
   process(A, B, choix)
   begin
      case to_integer(unsigned(choix)) is
         when 0 => F <= A and B;
         when 1 => F <= A or B;
         when 2 => F <= A nand B;
         when 3 => F <= A nor B;
         when 4 => F <= A xor B;
         when 5 => F <= A xnor B;
         when 6 => F <= not(A);
         when 7 => F <= not(B);
         when others => F <= (others => 'X');
      end case;
   end process;
end arch;
```

Unités fonctionnelles
Comparateurs

- Un comparateur permet de comparer les grandeurs relatives de deux valeurs et d'identifier leur égalité éventuelle.
- Ce type de circuit est essentiel dans un microprocesseur pour pouvoir effectuer des branchements conditionnels.
- Les opérateurs de VHDL pour la comparaison sont =, /=, <, <=, >, et >=. Dans chaque cas le résultat de la comparaison est de type boolean.
- Comme pour les opérations arithmétiques, le type des opérandes est critique et peut déterminer la valeur de la comparaison.

Unités fonctionnelles
Comparateur: code VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity comparateur is
  generic (
    W : positive := 8 -- largeur des opérandes
  );
  port(
    A, B : in signed(W - 1 downto 0);
    eq, neq, gt, lt, ge, le : out std_logic
  );
end comparateur;

architecture arch of comparateur is
begin
eq <= '1' when A = B else '0';
neq <= '1' when A /= B else '0';
gt <= '1' when A > B else '0';
lt <= '1' when A < B else '0';
ge <= '1' when A >= B else '0';
le <= '1' when A <= B else '0';
end arch;
```

Unités fonctionnelles
Compteurs

- Un compteur compte le nombre d'occurrences d'un événement.
- Un compteur est habituellement composé d'un registre couplé à un circuit combinatoire qui calcule la prochaine valeur du compte en fonction de sa valeur présente.
- Il y a plusieurs types de compteurs:
  - Compteur binaire. Progression monotone : 000, 001, 010, 011, …, 101, 110, 111, 000, 001, etc. Un compteur binaire à *n* bits a 2*n* états différents.
  - Compteur modulo-*n*. Ce compteur est réinitialisé à zéro dès qu'une valeur spécifiée est atteinte. Cas particulier: compteur BCD: 0000, 0001, … 1000, 1001, 0000, …;
  - Compteur à anneau. 0001, 0010, 0100, 1000, 0001, 0010, etc. Peut entrer dans une séquence d'états interdits si une erreur se produit.
  - Compteur Johnson. 0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000, 0000, 0001, …
  - Compteur à séquence arbitraire. L'utilisateur détermine la séquence, comme par exemple 0, 3, 1, 4, 2, 6, 0, 3, 1, etc.

Unités fonctionnelles
Compteur: code VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;

entity compteurSynchrone is
   generic (
      W : integer := 4 -- nombre de bits du compteur
   );
   port(
      reset : in STD_LOGIC;
      CLK : in STD_LOGIC;
      mode : in unsigned(1 downto 0);
      D : in unsigned(W - 1 downto 0);
      Q : out unsigned(W - 1 downto 0)
   );
end compteurSynchrone;

architecture comportementale of compteurSynchrone is
begin
   process (CLK, reset)
   variable Qinterne : unsigned(W - 1 downto 0);
   begin
      if reset='0' then
         Qinterne := (others => '0');
      elsif CLK='1' and CLK'event then
         case mode is
            when "01" => Qinterne := Qinterne + 1;
            when "10" => Qinterne := Qinterne - 1;
            when "11" => Qinterne := D;
            when others => Qinterne := Qinterne;
         end case;
      end if;
      Q <= Qinterne;
   end process;
end comportementale;
```

Unités fonctionnelles
Compteurs

- En plus de la séquence suivie, les compteurs peuvent être caractérisés par :
  - La valeur de réinitialisation (souvent 0).
  - La direction du compte (le haut, le bas ou les deux).
  - Le chargement parallèle d'une valeur de compte.
  - Une entrée ou une sortie sérielle.