Machines Algorithmiques

TP soutien sur machine salle sur ADE

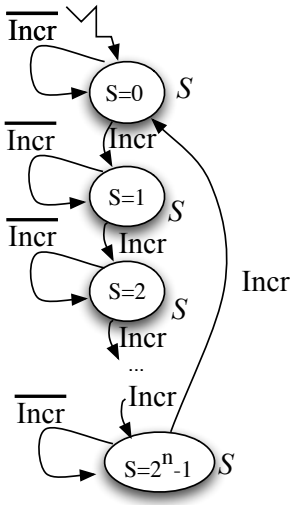
Introduction

- On connaît maintenant
  - **Circuits combinatoires** : réalisation de fonctions booléennes, permet de réaliser n'importe quel calcul
  - **Circuits séquentiels** : éléments de mémorisation, réalisation d'automate à parti de son graphe
- On voudrait réaliser un circuit à partir d'un algorithme
- Un circuit qui "exécute" un algorithme

Algorithme sous forme d'automate d'états finis

- On peut essayer de dessiner le graphe d'un automate modélisant un algorithme.
- Les états d'un tel automate sont nécessaires pour mémoriser les valeurs des variables à un instant donné de l'algorithme
- Exemple:
  - S un entier , Incr un booléen
  - S=0

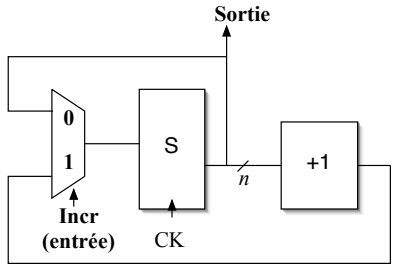
  Tant que vrai faire
  debut
  Lire (Incr)
  Si Incr alors S=S+1 modulo 2^n
  Délivrer S
  fin
- Il faut un état pour mémoriser chaque valeur de S



- On réalise ensuite le circuit à partir de l'automate
- Si n est grand, le nombre d'états devient vite très grand

Exemple de réalisation

- Essayons de réaliser l'algorithme sans "donner" le graphe de l'automate
- Une variable : un registre (dont la taille est fixée au préalable)
- Un calcul : un circuit combinatoire réalisant ce calcul
- Une entrée/sortie : pour l'instant on simplifie et on délivre/récupère une valeur sur un bus
- Un si alors sinon : un multiplexeur 2 vers 1
- Une boucle : un rebouclage sur les registres
- Un front sur l'horloge CK = une nouvelle affectation de la variable



Architectures de circuit

- Deux grands types d'architecture ont été utilisés pour réaliser les "circuits algorithmiques"
- **Flux de donnée** : (data flow)
  - Réalisation comme on vient de le voir sur l'exemple
  - Utilisé aujourd'hui dans la conception assistée par ordinateur
  - Peut devenir compliqué (boucle imbriquée)
- **Séparation du contrôle** (enchaînement des actions et affectations) et de l'exécution de ces **opérations**
  - Partie contrôle / partie opérative
  - Utilisé au début de la conception assistée car plus simple à "automatiser"
  - Moins d'unité de calcul mais en général un temps d'exécution plus grand que le flux de donnée

Exemple d'architecture "Flux de donnée"

La suite de Syracuse:

Algorithme:

U=U0 ;

Tant que vrai faire

debut

  Si (U modulo 2= 0) alors

    U= U div 2

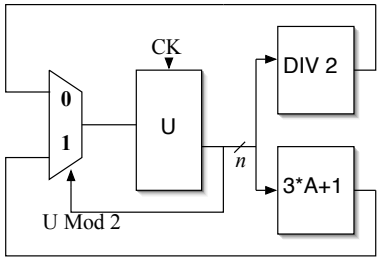
  Sinon U= 3*U + 1

  Délivrer (U)

fin

Calcul de la suite de Syracuse

• On ne s'intéresse pas au entrées/sorties, on suppose U initialisé à U0

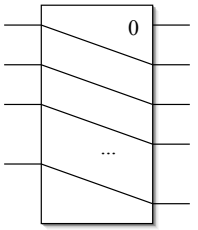


• U Mod 2: Bit de poids faible de U

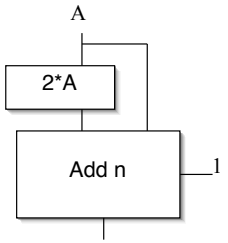
• Circuits combinatoires DIV 2 et 3A +1 ?

Calcul de la suite de Syracuse

- DIV2 : décalage de un bit vers les poids faibles



- 3*A + 1: possible simplement à l'aide d'un additionneur
  - 3*A+1= A+2*A+1



Exemple 2
d'architecture "Flux de donnée"

Multiplication à base d'additions

Algorithme :

R1 et R2 deux entiers ≥ 0

R3 le résultat entier R3=R1*R2

```
R3=0
Tant que R2>0 faire
debut
    R3=R3+R1 ;
    R2=R2-1;
fin
```

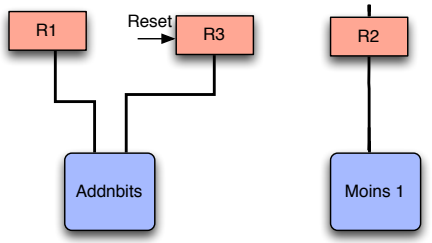

Réalisation d'un multiplieur algorithmique

On ne s'occupe pas de l'initialisation de R1 et R2 :
entrée/sortie vue plus tard

Une variable : un registre sur n bits

Un calcul : un circuit combinatoire réalisant ce calcul

```
R3=0
Tant que R2>0 faire
debut
    R3=R3+R1 ;
    R2=R2-1;
fin
```

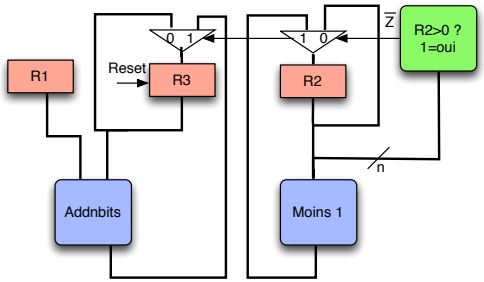


L'horloge activant le chargement des registres n'apparait pas sur les schémas

- Un tour de boucle : un rebouclage sur les registres plus un multiplexeur pour arrêter la boucle
  - C'est plus compliqué si il y a des affectations « dépendantes » dans la boucle

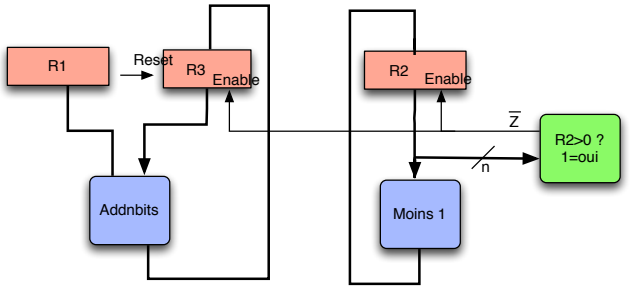
    Un circuit ne s'arrête pas
  - On fait en sorte que les registres contenant les variables de « sorties » ne changent plus
- Calcul des conditions du *tantque* réalisé par un circuit combinatoire

  R2>0 ? : R2 étant positif ou nul si R2>0 alors flag Z =0
  - Une porte OR suffit



Utilisation de l'entrée ***enable*** des registres pour commander le chargement des registres

- Economie des multiplexeurs
- Rappel :
  - si Enable =1 le registre est chargé au front de la clock
  - si Enable = 0 le contenu du registre est inchangé quelque soit la clock



Exemple 3
d'architecture "Flux de donnée"

Calcul du PGCD (algorithme d'Euclide)

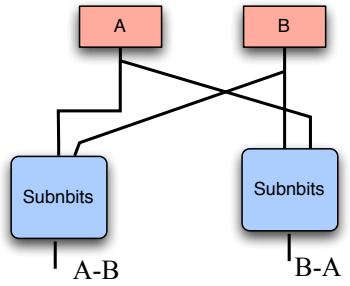
Algorithme :
A et B deux entiers ≥ 0
pgcd le résultat entier

```
Tant que A≠B faire
Si A>B alors
        A=A-B ;
    Sinon
        B=B-A ;
 pgcd=A ;
```


Calcul du PGCD

- On ne s'occupe de l'initialisation de A et B; et de la délivrance du résultat : entrée/sortie vue plus tard
- Une variable : un registre sur n bits
- Un calcul : un circuit combinatoire réalisant ce calcul

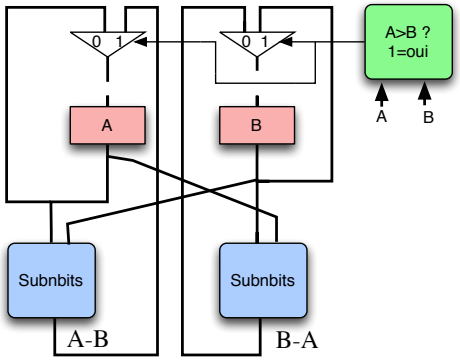
```
Tant que A≠B faire
Si A>B alors
        A=A-B ;
    Sinon
        B=B-A ;
 pgcd=A ;
```



Calcul du PGCD

- Un "si alors sinon" : un Multiplexeur par variable à modifier
- L'entrée de sélection est le résultat de la condition calculée dans un circuit combinatoire

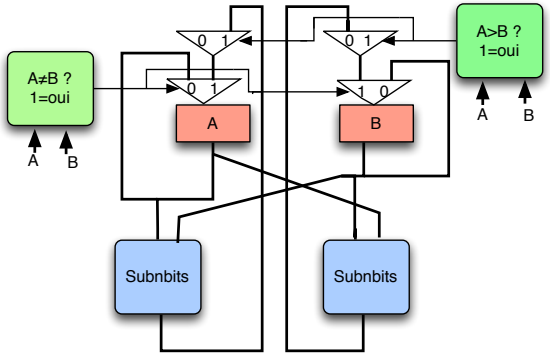
```
Tant que A≠B faire
Si A>B alors
        A=A-B ;
    Sinon
        B=B-A ;
 pgcd=A ;
```



Calcul du PGCD

- Un "tantque" : des multiplexeurs
- L'entrée de sélection est le résultat de la condition calculée dans un circuit combinatoire

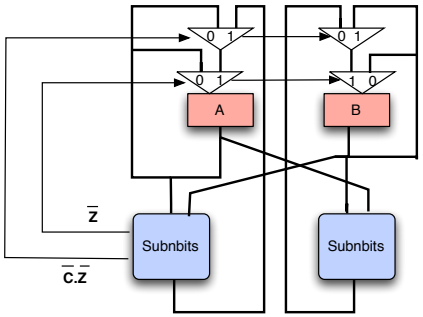
```
Tant que A≠B faire
Si A>B alors
        A=A-B ;
    Sinon
        B=B-A ;
 pgcd=A ;
```



Calcul du PGCD

- Le calcul des conditions peut se faire dans un des soustracteurs
- A et B toujours positifs ou nuls
- On code en base 2
- A>B: emprunt de la soustraction A-B égal à 0 et Z égal à 0
- A≠B : Z=0 après A-B

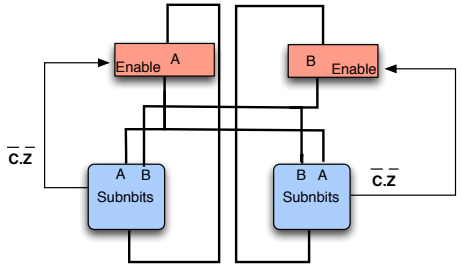
```
Tant que A≠B faire
Si A>B alors
        A=A-B ;
    Sinon
        B=B-A ;
pgcd=A ;
```



Calcul du PGCD

- On supprime les multiplexeurs en utilisant ***Enable***
- Transformation de l'algorithme et simplification des conditions

```
Tant que vrai faire
Début
   Si A>B alors
        A=A-B ;
   Si A<B alors
        B=B-A ;
Fin
pgcd=A ;
```



Les entrées/sorties

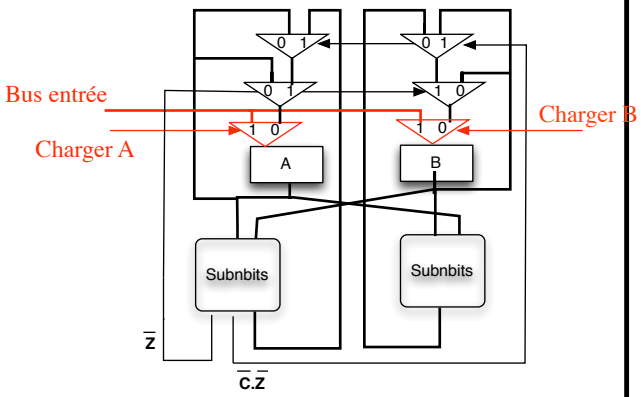
- Il faut un plusieurs bus d'entrée/sortie vers/depuis l'extérieur du circuit
- Il faut définir un protocole de communication avec l'extérieur
  - Quand est ce que la valeur initiale de A est sur le bus d'entrée ?
  - Quand est ce que les calculs commencent ?
  - Quand est ce que la valeur du résultat est sur la bus de sortie ?
- Il faut éventuellement des signaux supplémentaires pour définir précisément ce protocole
  - Exemple: Resultat_valide …

Un exemple d'entrées

Lire (A); lire (B);
Tant que A≠B faire
....

Délivrer (A);

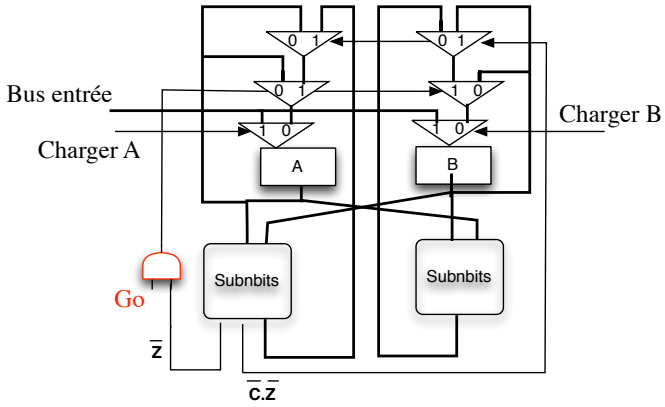
- Un bus d'entrée: l'extérieur met sur le bus d'entrée la valeur de A ou B;
- Il donne un signal charger A et charger B au "bon" moment



- Il faut que la valeur en entrée soit présente au moins pendant une période d'horloge
- Attention il ne faut que le registre change avant que l'algo démarre

Un exemple d'entrées

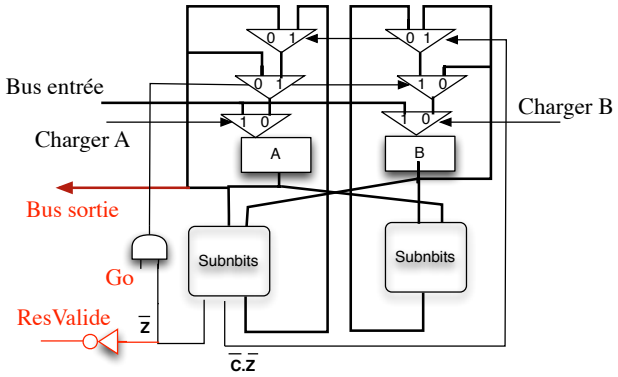
- L'utilisateur lance les calculs à l'aide d'un signal supplémentaire ***Go***
- A et B ne changent pas tant que le signal ***Go*** = 0
- En résumé: l'utilisateur met Go à 0 puis charge A et B puis met Go à 1



Un exemple de sortie

- ...

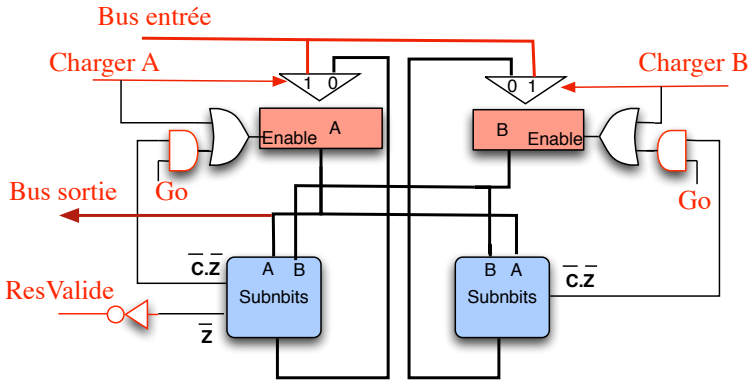
Tant que A≠B faire
....
Délivrer (A);



- Un bus de sortie: le circuit délivre un signal ***ResValide*** quand le résultat est disponible sur ce bus

Un exemple d'entrées/sorties

- Avec la version utilisant « Enable »



Architecture
Partie opérative/ Partie contrôle

- Architecture Flux de donnée peut devenir vite compliquée, surtout au niveau de l'enchaînement des calculs
- Architecture PC/PO plus facile à mettre en oeuvre "à la main"
- Utilisée au départ pour la conception des circuits
- Exemple le processeur 68000 (voir architecture microprogrammée étudiée au 2ème semestre)
- Principe: séparation du contrôle de l'enchaînement des calculs et des calculs eux même

Architecture
Partie opérative/ Partie contrôle

- **Partie opérative** :
  - -Mémorisation des variables et circuits combinatoire permettant d'effectuer les calculs
  - -Sans décider de l'enchaînement des calculs et affectations.
- **Partie contrôle** :
  - -décide de l'enchaînement des calculs, décision des structures algorithmique *tantque* et *sialorssinon*
- En comparaison avec le flux de donnée, souvent moins coûteux en terme de "surface" au détriment de la "vitesse"

Exemple PGCD

- Tant que A≠B faire

  Si A>B alors

  A=A-B ;

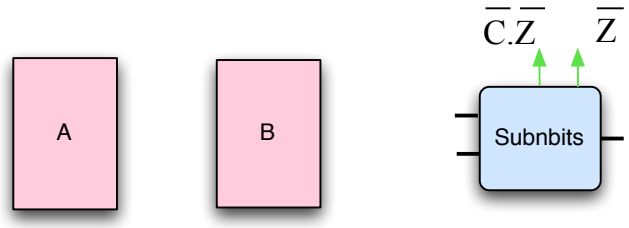
  Sinon

  B=B-A ;

  pgcd=A ;
- On répertorie les variables (A, B) et les calculs apparaissant dans l'algorithme: A-B et B-A: soustraction
- Il faut aussi répertorier les calculs nécessaires aux conditions: A≠B et A>B: une soustraction

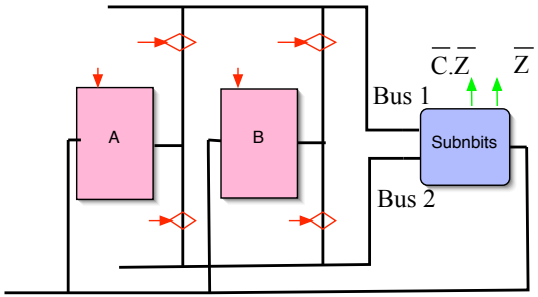
La partie opérative du PGCD

- Deux variables : deux registres
- Un circuit permettant de faire la soustraction et délivrant les conditions



La partie opérative du PGCD

- Il faut rajouter les bus permettant de faire les calculs et d'affecter les résultats
- A=A-B
- B=B-A

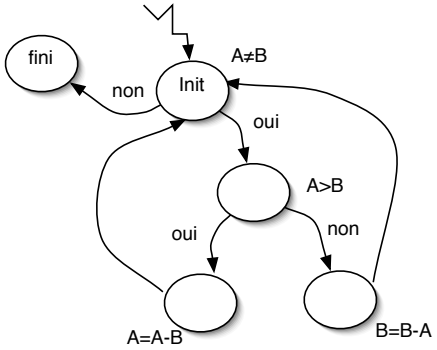


La partie contrôle

- Il faut décider de l'enchaînement des calculs
- On peut le décrire simplement à l'aide d'un automate d'états finis
- Similaire à la traduction de structure algorithmique à l'aide de branchements en assembleur

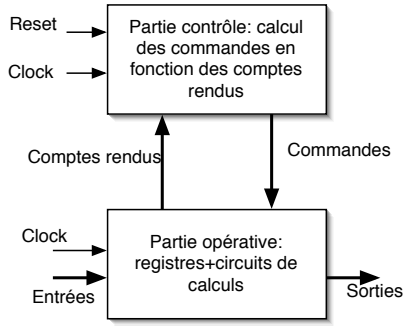
•Tant que A≠B faire

Si A>B alors
A=A-B ;
Sinon
B=B-A ;
pgcd=A ;



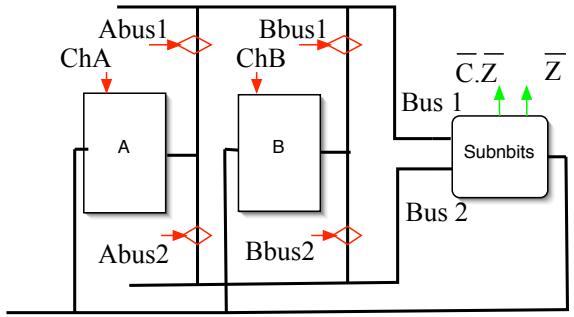
Le dialogue Partie contrôle/partie opérative

- La PC envoie à la PO les signaux permettant de décider du calcul à effectuer (et du chargement éventuel du résultat) à un moment donné
- La PO envoie à la PC , les comptes rendus des calculs des conditions (en général les valeurs des flags)



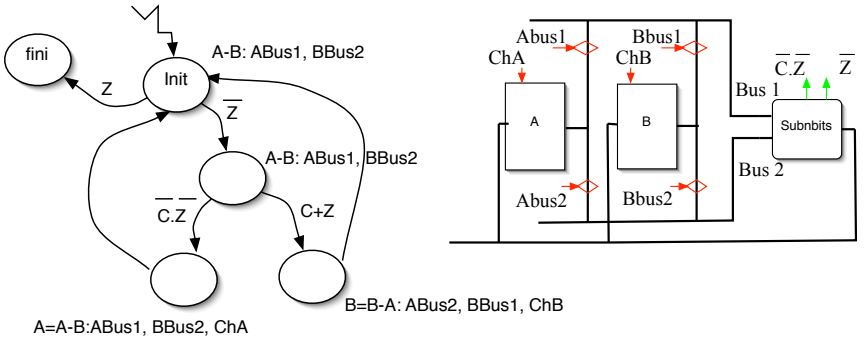
Automate de la partie de contrôle

- Les sorties : commandes de la PO
- Les entrées : compte des rendus (résultat du calcul des conditions)
- La PO avec ses signaux de commandes :

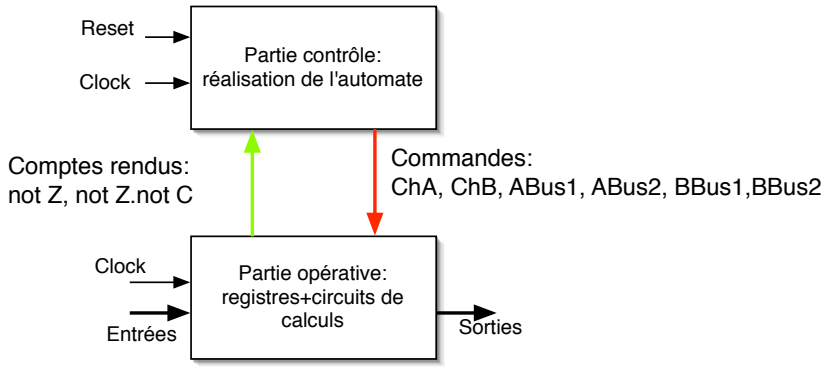


Automate de la partie de contrôle

- On traduit
  - Les instructions apparaissant sur l'automate en valeurs des signaux de commandes
  - Les valeurs des conditions en fonctions booléennes des flags
  - L'automate est ensuite réalisé "classiquement"



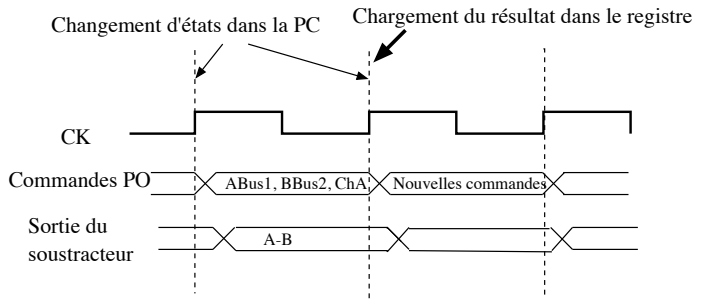
Architecture complète



- Un Reset pour démarrer
- Synchronisation de la PC et la PO à l'aide d'une même horloge : Clock

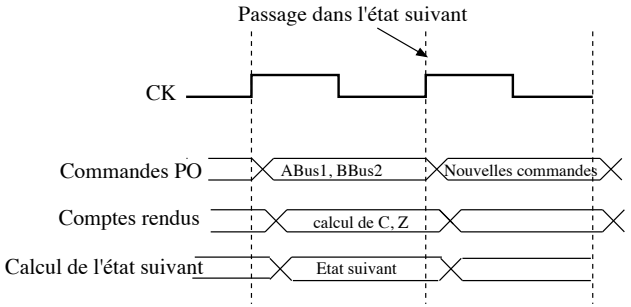
Fonctionnement dans le temps

- A chaque front montant de l'horloge de nouvelles commandes sont envoyées à la PO (changement d'état) pour effectuer le calcul voulu à ce moment là
- Le résultat est chargé au front montant de l'horloge suivant



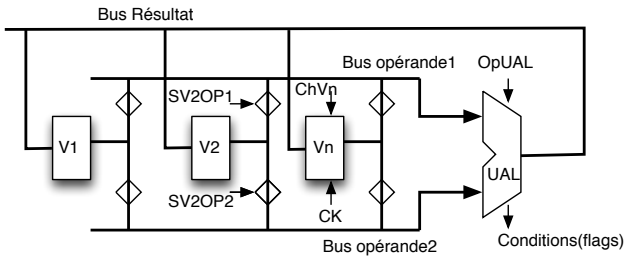
Fonctionnement dans le temps

- Les comptes rendus (les conditions des instructions, flags de l'UAL) sont calculés dans la PO et envoyés dans la PC pour qu'elle puisse prendre la décision du prochain calcul à effectuer
- Pendant une période d'horloge:
  - -Les conditions sont calculées dans l'UAL
  - -Le nouvel état est calculé à partir de ces conditions
  - -Au prochain front montant de l'horloge, l'automate change d'état



Généralisation de la PO

- Une UAL permettant de réaliser toutes les opérations apparaissant dans l'algorithme, y compris les calculs des conditions
- Deux bus opérandes : il faut ramener l'algorithme à des calculs à 2 opérandes
- Un bus résultat que l'on peut mémoriser dans n'importe quels registres
- Les fils de commandes décident de l'opération à effectuer (***SViOP1, SVjOP2, OpUAL***) et de l'affectation de la variable (***ChVi***). Ils viennent de la PC.

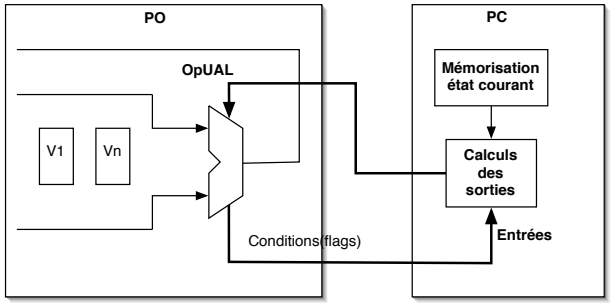


Optimisations de la PO

- A partir de cette PO “type”, on peut l’affiner en supprimant les chemins de données inutiles
- Autres possibilités d’optimisations : parallélisation d’instructions indépendantes dans un même état de la PC
- Cela implique en générale l’ajout d’opérateur supplémentaire
- On se dirige alors vers la solution « flux de donnée »
- Voir sur des exemples plus loin

Type de l’automate de la PC

- On ne peut pas utiliser un automate de Mealy, c’est forcément un automate de Moore
- Si la PC est un automate de Mealy cela ne fonctionne pas :
  - Il y a une **boucle** sur des circuits combinatoires
  - Les commandes dépendent alors des sorties de l’UAL qui dépendent elles-mêmes des commandes



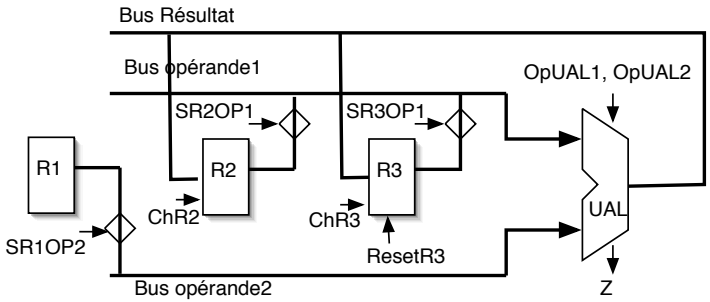
Exemple du multiplieur

```
R3=0
Tant que R2>0 faire
debut
     R3=R3+R1 ;
     R2=R2-1;
fin
```

- Trois variables R1, R2, R3
- Deux opérations : Add (R3+R1), -1 (R2-1)
- R3=0 peut se faire simplement à l’aide du reset du registre
- Calcul de la condition: R2>0 équivalent à R2≠0 donc Op1 et calculer Z
- UAL à 3 opérations: +1, add, op1

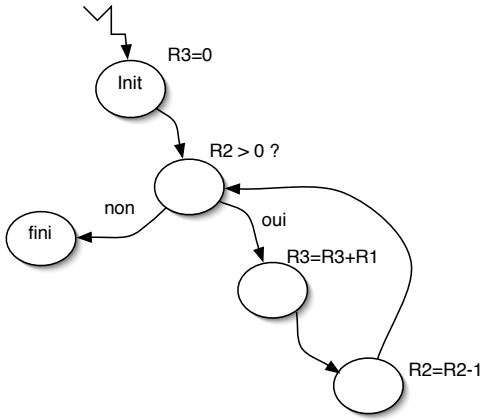
PO du multiplieur

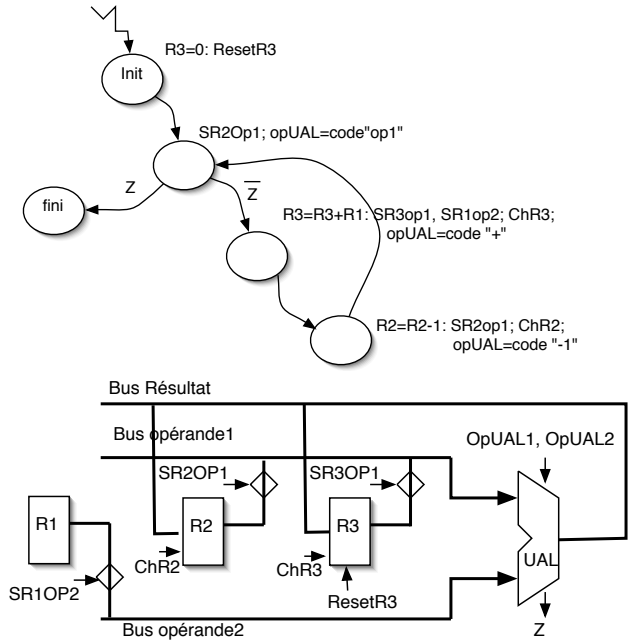
- R3=0 : ResetR3
- Condition R2>0 : R2 → op1 ; opUAL=code ”op1”
- R2=R2-1: R2 → Op1; Ch R2; opUAL=code ”-1”
- R3=R3+R1 : R3 → Op1; R1 → Op2; ChR3; opUAL=code a”+”



PC du multiplieur

R3=0
Tant que R2>0 faire
debut
    R3=R3+R1 ;
    R2=R2-1;
fin





Autre exemple

Calcul de la suite de Fibonacci

X : un entier >0 ;
Un-1 et **Un-2** : deux entiers > 0 ; { les données } ;
Un: un entier ≥ 0; { le résultat }

Algorithme :

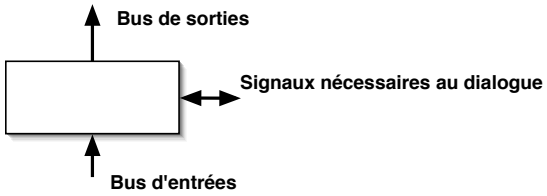
Un-1= **Un-2**= **Un**= 1
Tant que **Un** < X faire
Debut
    Un = Un-1 + Un-2
    Un-2 = Un-1
    Un-1 = Un
Fin

Les entrées / sorties

- Dialogue à instaurer avec l'extérieur du circuit : utilisateur humain, autres circuits (processeurs, mémoire …)
- Problème de synchronisation :
    - Quand est-ce que les entrées (ou sorties) sont présentes ?
    - Quand est-ce que le circuit a fini de lire une entrée ?
    - Quand est-ce que l'extérieur à effectivement lu une sortie ?
- Rappel : les entrées ne doivent pas changer au moment des fronts montants de l'horloge qui active les registres.

L'interface avec le monde extérieur

- Des bus d'entrées ou de sorties
- On peut aussi utiliser un seul bus qui permet les entrées et les sorties
  - Exemple: Bus adresses et bus données du processeur
- Des signaux supplémentaires nécessaires à la synchronisation

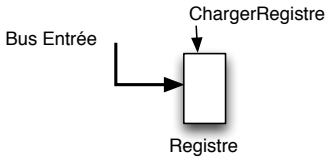


Un protocole universel: La poignée de main

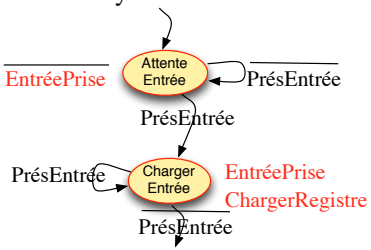
- Entrées et sorties : même protocole
- Deux visions du même protocole : les entrées vers le circuit sont les sorties du monde extérieur et réciproquement
- Si certaines hypothèses sont vérifiées, on peut simplifier ce protocole
- Par exemple si le circuit est supposé beaucoup plus rapide que l'extérieur

Automate pour une entrée

- Pour les **entrées** du circuit:
  - Un signal ***EntréePrise*** qui permet au circuit de signaler à l'extérieur qu'il a récupéré l'entrée
  - Un signal ***PresenceEntrée*** qui permet à l'extérieur de signaler qu'une nouvelle entrée est présente sur le bus d'entrée
  - Le chargement du bus d'entrée dans un registre se fait quand on sort de l'état **ChargerEntrée**

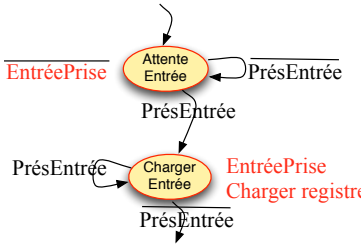


Automate de synchronisation:

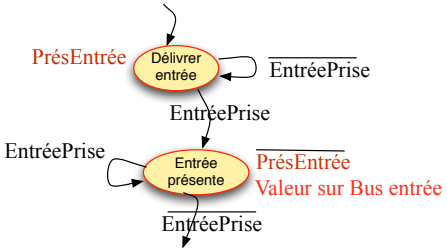


Une entrée du circuit

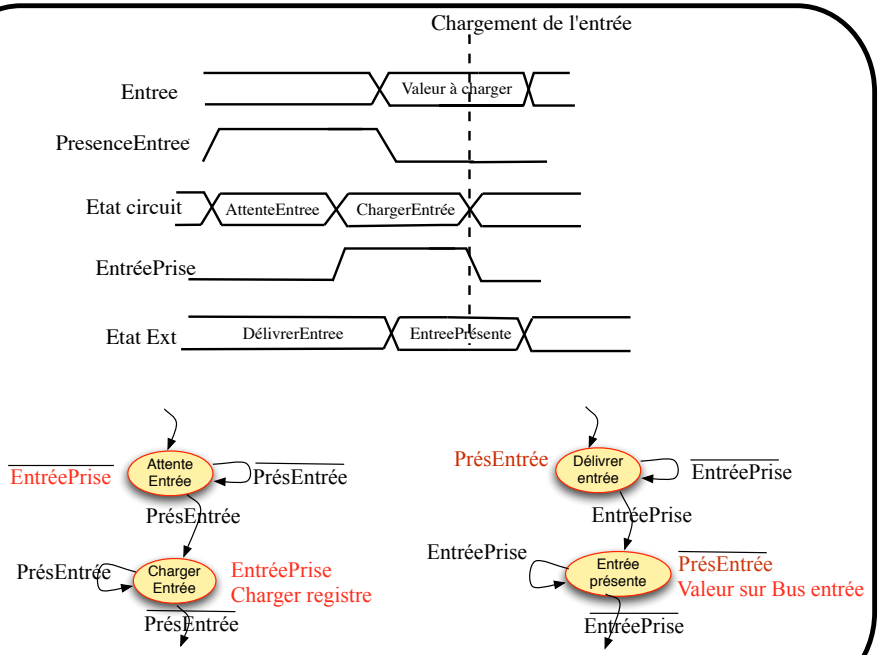
•Entrée pour le circuit



•Sortie pour l'extérieur

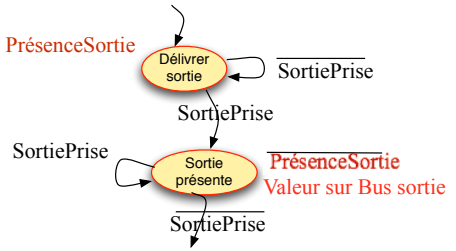


•La valeur doit être présente sur le bus d'entrée pendant l'état **EntréePrésente**



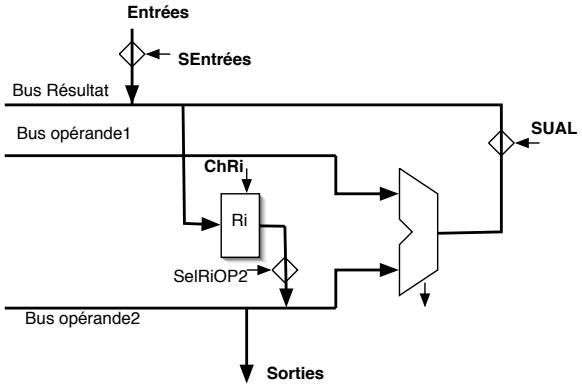
Automate pour une sortie

- Pour les **sorties** du circuit:
  - Un signal ***PrésenceSortie*** qui permet au circuit de signaler à l'extérieur qu'il y a une nouvelle sortie présente sur le bus de sortie
  - Un signal ***SortiePrise*** qui permet à l'extérieur de signaler qu'il a récupéré la sortie présente sur le bus de sortie
  - La valeur a délivrer doit être présente sur le bus de sortie pendant l'état ***SortiePrésente***
- Automate de synchronisation:



Les entrées/sorties dans l'architecture PCPO

- Dans la PO:
  - Plusieurs solutions possibles (un ou plusieurs bus...)
  - Sur le bus résultat, sur le bus opérande...
- Exemple :



Les entrées/sorties dans la PC

- On rajoute à l'automate les deux états de la poignée de main pour chaque E/S à l'endroit voulu
- Avec la PO précédente:
  - Dans l'état ***ChargEntrée***, on rajoute les signaux vers la PO ***SEntrées***=1, ***SUAL***=0, ***ChRi*** où Ri est le registre à charger par le bus Entrée
  - Dans l'état SortiePrésente on rajoute ***SelRiOp2***=1 suivant le registre à délivrer en sortie

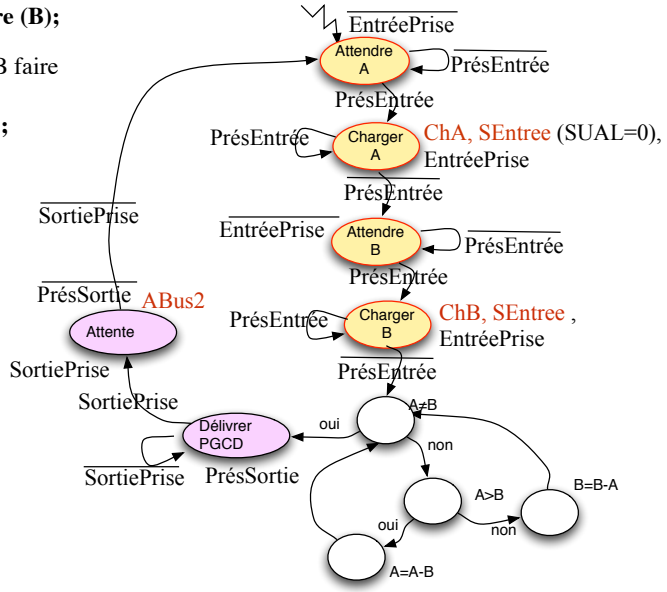
Exemple d'entrée/sortie

- **Lire (A); lire (B);**

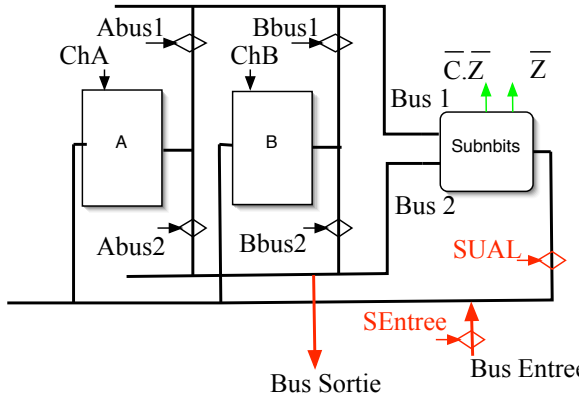
  Tant que A≠B faire

  ....

  Délivrer (A);



Exemple d'entrée/sortie
La PO associée



Un exemple: le processeur et la mémoire

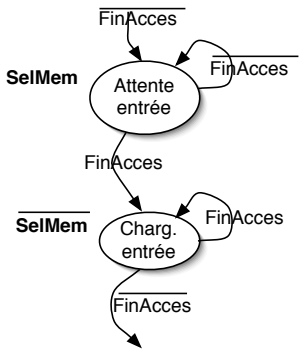
- Un bus de sortie : le bus adresse
- Un bus d'entrée et de sortie : le bus donnée
- Signaux de dialogue : ***FinAcces***, ***SelMem***, ***RWbarre***
- Le signal ***FinAcces*** sert à la mémoire
  - -à signaler la présence de donnée pour une lecture (***PresenceEntrée***)
  - -et aussi à l'écriture effective en mémoire (***SortiePrise***)
- De même le signal ***SelMem*** permet au processeur
  - -de signaler à la mémoire qu'une donnée est présente sur le bus donnée (***SortiePresente***)
  - -de signaler à la mémoire que la donnée est effectivement lue (***EntreePrise***)

Dialogue entre le processeur et la mémoire

- Vue côté processeur :
- Quand Ecriture Mémoire (***RWb***=0)
  - C'est une sortie du processeur
  - Données Prêtes sur le Bus donnée, Adresses Prêtes sur le Bus adresse
  - ***SelMem*** signale à l'extérieur que le bus donnée est valide (***PresenceSortie***)
  - -***FinAcces*** délivré par la mémoire signale la prise en compte des données, la fin de l'écriture en mémoire (***SortiePrise***)
- Quand lecture Memoire (***Rwb***=1)
  - C'est une entrée
  - ***FinAcces*** délivré par la mémoire signale la présence des données sur le bus données (***PresenceEntree***)
  - Le processeur signale qu'il a récupéré l'entrée par ***SelMem*** à 0

Dialogue entre le processeur et la mémoire

Lecture mémoire



Ecriture Mémoire