Unité d'enseignement : Systèmes séquentiels avancés (SSA)

MSS complexes

MSS: Machines séquentielles synchrones

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

ReDS
Reconfigurable & Embedded Digital Systems

Etienne Messerli
Institut REDS, HEIG-VD

Mise à jour le 14 mars 2013

Contenu de la présentation

- Complexe?
- On fractionne!
- Fractionnement hiérarchique
- Ebauche de méthode de travail

Une MSS devient vite complexe

- Si elle requiert plus de quelques états (↗16)
  ou comporte plus de quelques entrées (↗ 4)
- une MSS devient difficile à décrire par ses états (graphe ou table)
- Exception : structures modulaires telles que les registres et les compteurs

Description par les états

- Ne permet pas de distinguer
  - ✓ l'algorithme, d'une part
  - ✓ les données à traiter, d'autre part
- souvent le nombre d'états dépend de la taille (nombre de bits) des données à traiter
- Exemple : comparateur
- Ne facilite pas l'utilisation de fonctions standard

Art ou science?

- La conception d'une MSS complexe est autant un art (imagination, entraînement) qu'une science (méthodes, techniques)
- Nous allons étudier une méthode de travail et des techniques de réalisation
- et développer nos aptitudes par des exercices

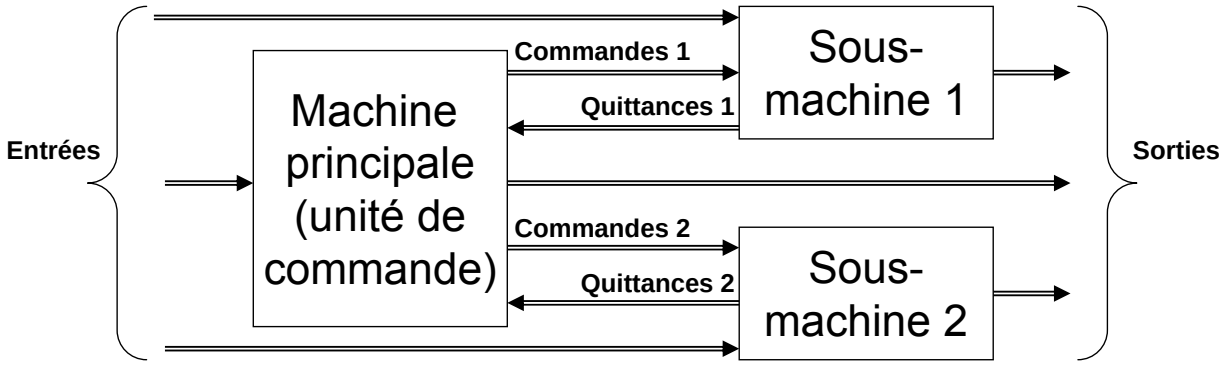
Trop complexe? Fractionnez-le!

- Méthode de travail universelle :
  Problème complexe => le décomposer en plusieurs sous-problèmes plus simples (fractionner, diviser, ..)
- Une MSS complexe sera découpée en plusieurs MSS simples qui interagissent
- Forme de découpage la plus fréquente :

Hiérarchique

Découpage hiérarchique

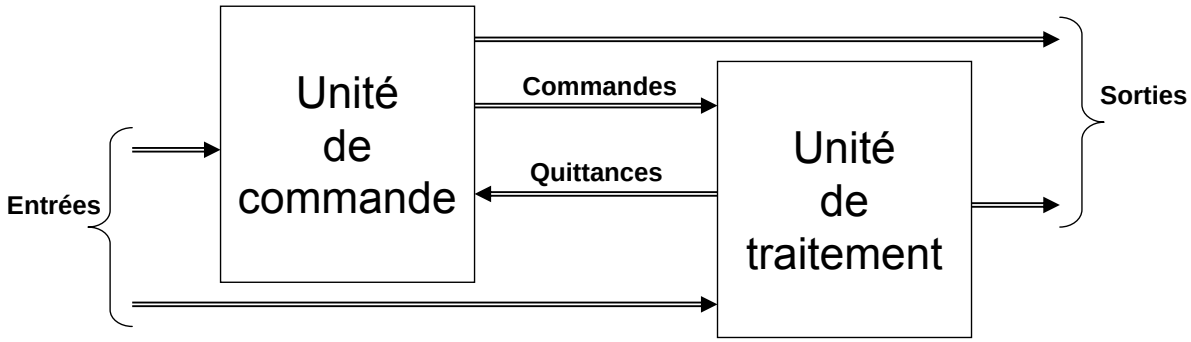
- Une tête qui commande (machine principale)
- des membres qui exécutent (sous-machines)
- et fournissent des quittances



Découpage hiérarchique

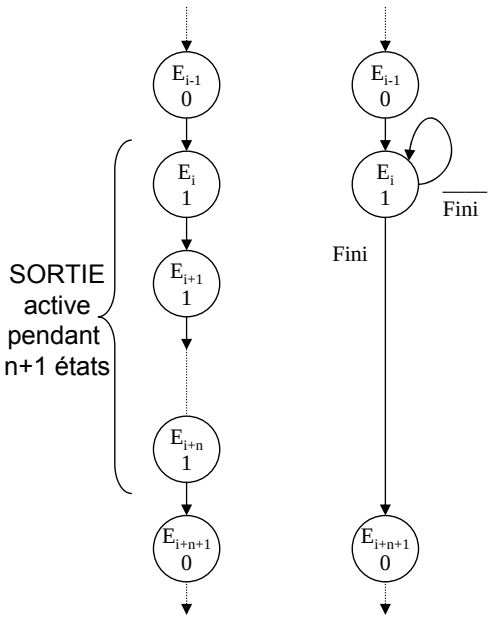
- Si encore trop complexe, chaque sous-machine peut elle-même être découpée hiérarchiquement
- Souvent, 2 niveaux suffisent
  - ✓ machine principale (unité de commande, niveau supérieur)
  - ✓ les sous-machines (exécute, niveau inférieur)
- Les sous-machines peuvent être regroupées en un seul bloc de niveau inférieur

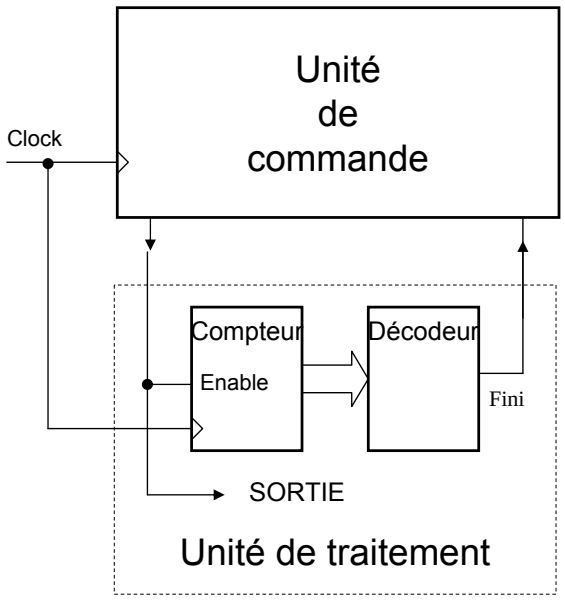
Partition à 2 niveaux hiérarchiques



- Machine principale : unité de commande (UC), unité de séquencement, séquenceur, "control unit", "sequencer"
- Sous-machines : unité de traitement (UT), unité d'exécution, "execution unit", "data unit"

Identification de sous-machines





La méthode en résumé

- Approche Top-Down
- Description d'un algorithme global
- Séquence d'opérations (au lieu d'états)
- Identification des sous-machines possibles (fonctions standard)
- Décomposition hiérarchique

Exercice d'introduction MSS

- Vous devez effectuer la comparaison de 2 nombres de 32 bits se trouvant chacun dans un registre à décalages. Décrivez un algorithme de comparaison bit à bit commençant par le poids faible. Proposez diverses façons d'implémenter cet algorithme.

Structures des MSS complexes

- Structures d'une unité de commande
  - ✓UC câblée
  - ✓UC micro-programmée
- Structures d'une unité de traitement
  - ✓UT spécialisée
  - ✓UT universelle
- Combinaisons UC - UT

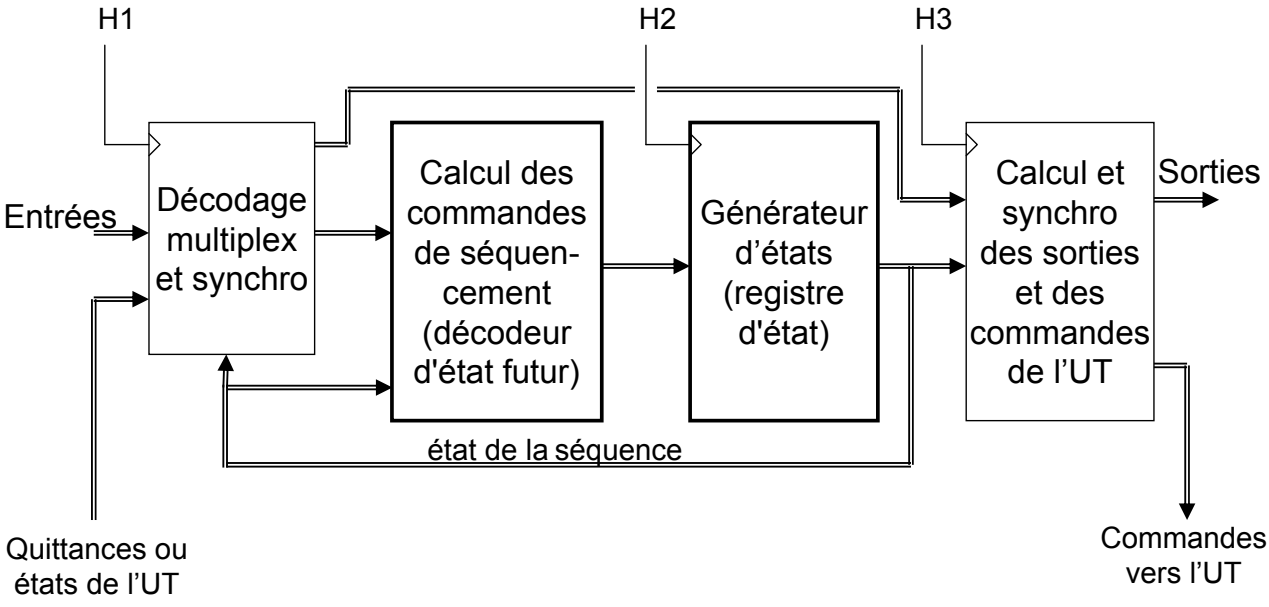
Unité de commande

- Séquencement des opérations exécutées par de l'unité de traitement
- Peut effectuer une partie du traitement
- C'est une machine séquentielle
- Structure:
  - ✓Si elle est simple : UC câblée (graphe, description VHDL)
  - ✓Si elle est trop complexe pour en faire un graphe ?
    séquence d'opérations => assimilée à un programme
    => UC micro-programmée (séquenceur + mémoire)

UC câblée

- UC réalisée comme une MSS simple
- Comportement déterminé par son câblage (schéma), d'où son nom
- Modifier comportement → modifier câblage
- Aujourd'hui : "simple" modification d'une description VHDL pour un FPGA
- Attention : timing à re-vérifier et banc de test à revoir !

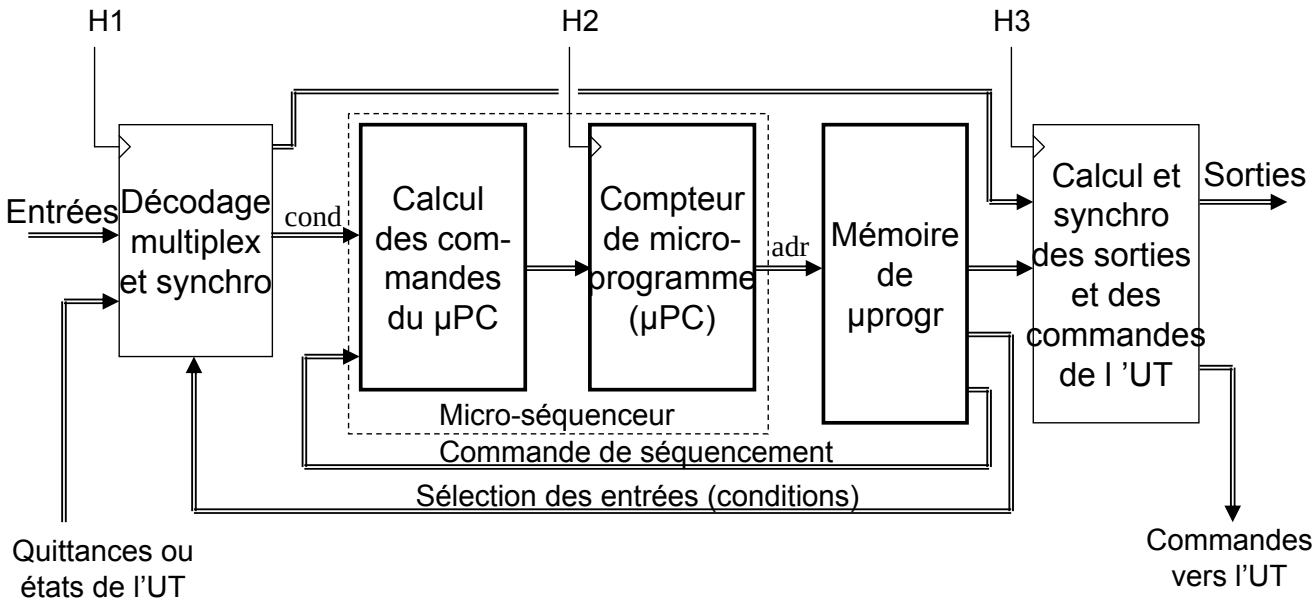
UC câblée : schéma bloc



UC (micro)programmée

- Schéma standard (câblage)
- Comportement déterminé par un programme dans une mémoire
- Modifications possibles sans changer le câblage tant que
  - ✓taille de la mémoire suffisante
  - ✓nombre d'entrées disponibles suffisant
  - ✓nombre de sorties disponibles suffisant

UC micro-programmée : schéma bloc



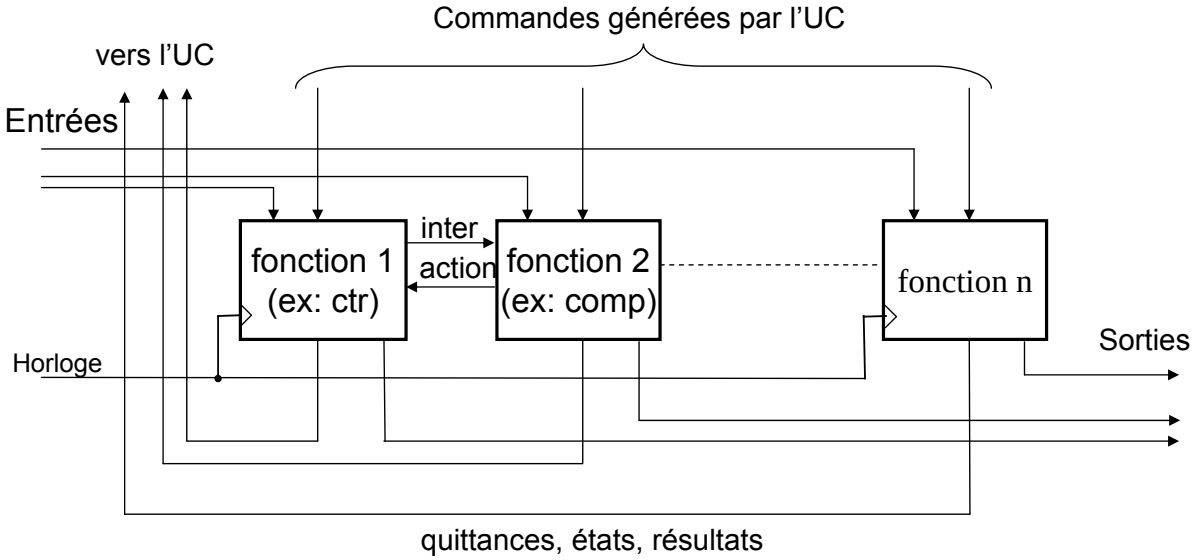
UT : 2 approches → 2 structures

- Chaque fonction est réalisée par un circuit spécifique :
  - ✓ comptage avec un compteur
  - ✓ comparaison avec un comparateur, etc.
  - ➔ UT spécialisée
- Chaque fonction est décomposée en une série de fonctions élémentaires standardisées (et, ou, inversion, addition, mémorisation)
  - ➔ UT universelle (ALU ou RALU)

UT spécialisée :

- Fonctions standard apparaissant dans l'algorithme (compteur, comparateur, addition, …)
- Fonctions spécialisées => descriptions VHDL
  - ✓ L'ensemble des fonctions travaillent en parallèles
  - ✓ performances maximum
  - ✓ volume, coûts et consommation proportionnels à la complexité
  - ✓ toute modification => correction description VHDL nécessite vérification du timing et une adaptation banc de test test
- Remarque :
  - ✓ un compteur correctement commandé peut aussi effectuer une addition

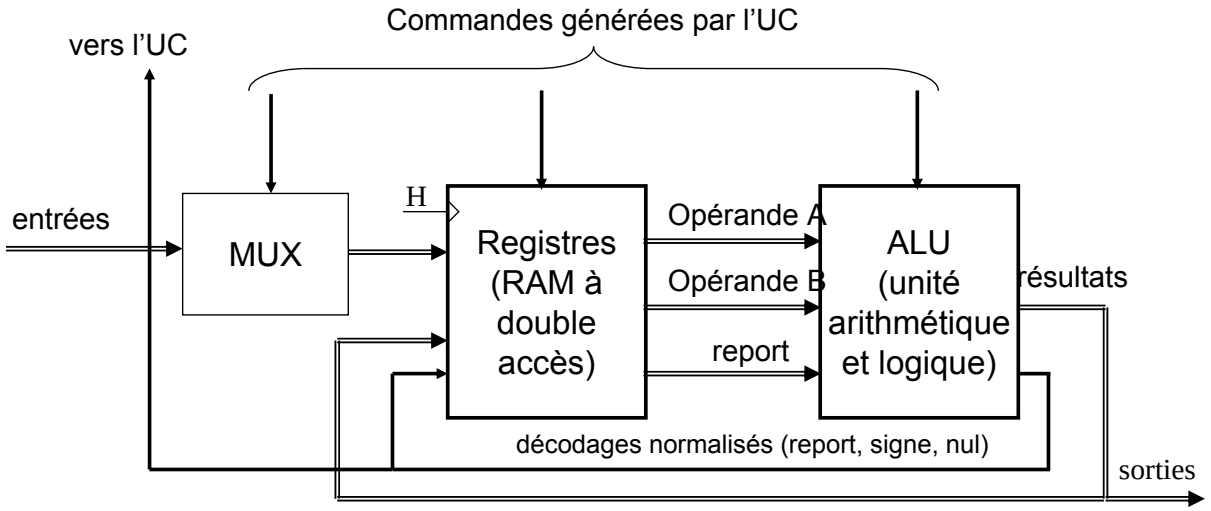
Schéma bloc d'une UT spécialisée



UT universelle : principe

- Toute fonction combinatoire peut être décomposée en une série (suite) de fonctions élémentaires *et, ou, inversion*
- Accélérer les calculs arithmétiques :
  => ajouter addition et soustraction
- Toute fonction séquentielle peut être décomposée en une fonction combinatoire et un registre

Schéma bloc d'une UT universelle



UT universelle : caractéristiques

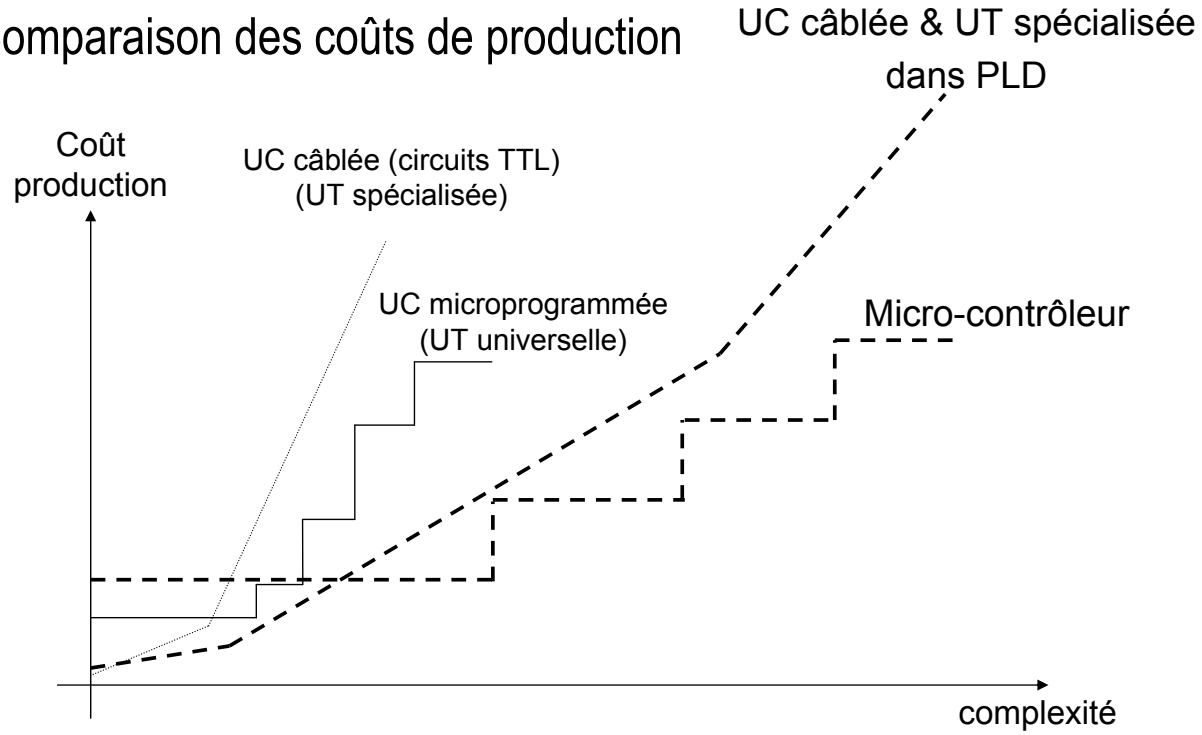
- Composée d'une ALU, de registres et de mux, câblés selon un schéma pouvant être standardisé (indépendant de l'application)
  - → performances inversement proportionnelles à la complexité du traitement (sérialisation)
  - → volume, coûts et consommation augmentant par escaliers
  - → ne requiert pas de modifications, celles-ci étant reportées sur l'UC (la séquence de commandes)

Combinaisons UC - UT

- UC : câblée / microprogrammée
- UT : spécialisée / universelle
  ➔ 4 combinaisons possibles
- A considérer:
  - ✓performances (vitesse de fonctionnement)
  - ✓coût et temps de développement
  - ✓souplesse (facilité de modification)
  - ✓coût de production

Combinaisons UC - UT

Comparaison des coûts de production



Combinaisons UC - UT

- UC câblée - UT spécialisée
  - ✓ performances maximum
  - ✓ coûts et temps de développement élevés
  - ✓ souplesse et coûts = améliorés par outils EDA et FPGA
  - ✓ coût de production élevé mais proportionnel à la complexité et aux performances
- Domaine d'utilisation:
  - ✓ Très utilisé actuellement pour ces performances très élevées
  - ✓ Solution compacte dans un seul circuit
  - ✓ Coût fortement diminué grâce aux FPGA *low-cost*
  - ✓ Utilisation d'IPs (design re-use)

Combinaisons UC - UT

- UC microprogrammée - UT spécialisée
  - ✓ performances moyennes
  - ✓ coût et temps de développement moyens à grands
  - ✓ souplesse élevée si seule la séquence doit être modifiée, sinon = celle d'un FPGA
  - ✓ coût de production élevé
- Domaine d'utilisation:
  - ✓ Plus utilisée tel quel!
  - ✓ Circuits actuellement utilisés:
    - → Microcontrôleur (8, 16 ou 32 bits) + FPGA

Combinaisons UC - UT

- UC câblée - UT universelle
  - ✓ performances faibles (meilleures qu'avec UC microprogrammée)
  - ✓ coût et temps de développement moyens
  - ✓ souplesse = celle d'un FPGA
  - ✓ coût de production moyen
  - ✓ convient aux systèmes à performances moyennes, de complexité moyenne
- Domaine d'utilisation:
  - ✓ utilisée dans les processeurs RISC

Combinaisons UC - UT

- UC microprogrammée - UT universelle
  - ✓ performances faibles
  - ✓ coût et temps de développement faibles
  - ✓ souplesse la plus élevée
  - ✓ coût de production faible mais avec un seuil de départ
  - ✓ convient aux systèmes à performances moyennes à faibles, de haute complexité
- Domaine d'utilisation:
  - ✓ utilisée dans les processeurs CISC
  - ✓ microcontrôleur (8, 16 ou 32 bits), DSP

Méthode de conception MSS complexe

1) Etablir les spécifications
2) Relations temporelles entrées ⇔ sorties, chronogrammes
3) Schéma-bloc grossier
   puis Algorithme avec phrase, texte => Organigramme grossier
4) Détailler progressivement => Organigrammes évolués
   choisir la partition entre l'UC et l'UT (lié aux points 5, 6 et 7)
5) Choisir la structure de l'unité de traitement (fcts standards)
6) Choisir la structure de l'unité de commande
7) Choisir les principaux composants
8) Réalisation UT : schémas, descriptions VHDL, …; puis test UT
9) Réalisation UC : org détaillé, graphe des états ou µprogramme; puis test UC
10) Test de l'ensemble, montage et test final

Documentation : à toutes les étapes

Dia volontairement laissé vide

Exemple : multiplication non signée …

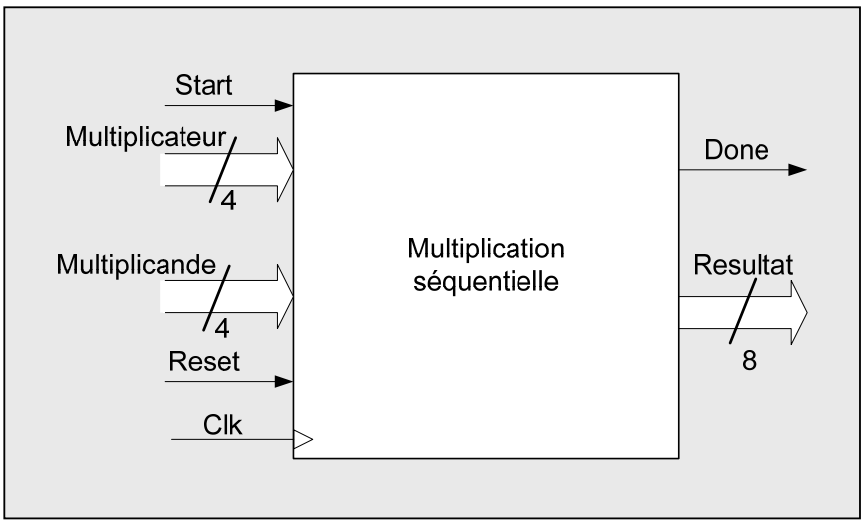
- Présentation de la démarche de conception d'une MSS complexe avec
  - ✓Multiplication d'entier non signé
  - ✓Algorithme déjà étudié au chapitre précédent (numération et arithmétique)

… multiplication non signée

- Etapes de la conception
  - ✓points 1 et 2 pas traité
  - ✓schéma bloc global
  - ✓algorithme grossier déjà étudié
  - ✓établir l'organigramme grossier
  - ✓partition UC / UT => organigrammes évolués
  - ✓schéma UT spécialisée
  - ✓UC câblée : organigramme détaillé => graphe des états

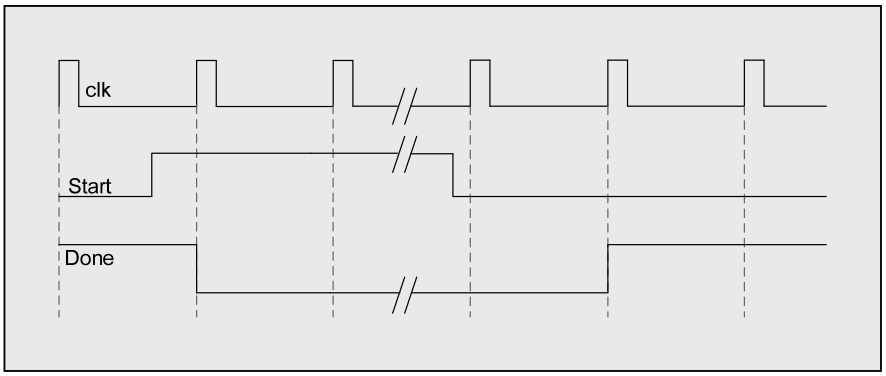
Schéma bloc pour la multiplication

Les signaux Start et Done permettront de contrôler le fonctionnement de notre MSS complexe. Les autres signaux viennent directement de l'algorithme



Chronogramme de fonctionnement

- Le chronogramme nous montre le fonctionnement des signaux de contrôle : Start et Done



Algorithme de la multiplication

Algorithme déjà étudié précédemment :

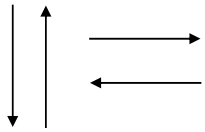
```
Resultat_Haut := 0;

for I=1 to Nombre_Bits loop
   if LSB(Multiplicateur) = 1 then
      Resultat_Haut := Resultat_Haut + Multiplicande;
      (mémorisation du report dans un flip-flop)
   else
      Resultat_Haut := Resultat_Haut+0;
   end if

   Décalage à droite(Report, Resultat_Haut, Multiplicateur);
end for;

Resultat_Bas := Multiplicateur;
Resultat := Resultat_Haut & Resultat_Bas;
```


Dia volontairement laissé vide

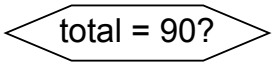
Organigramme

Description graphique d'un algorithme

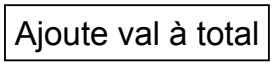
Symboles :



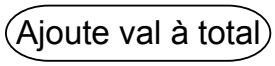
flèche : évolution (pointe vers l'opération suivante)



hexagone aplati : test déterminant l'évolution



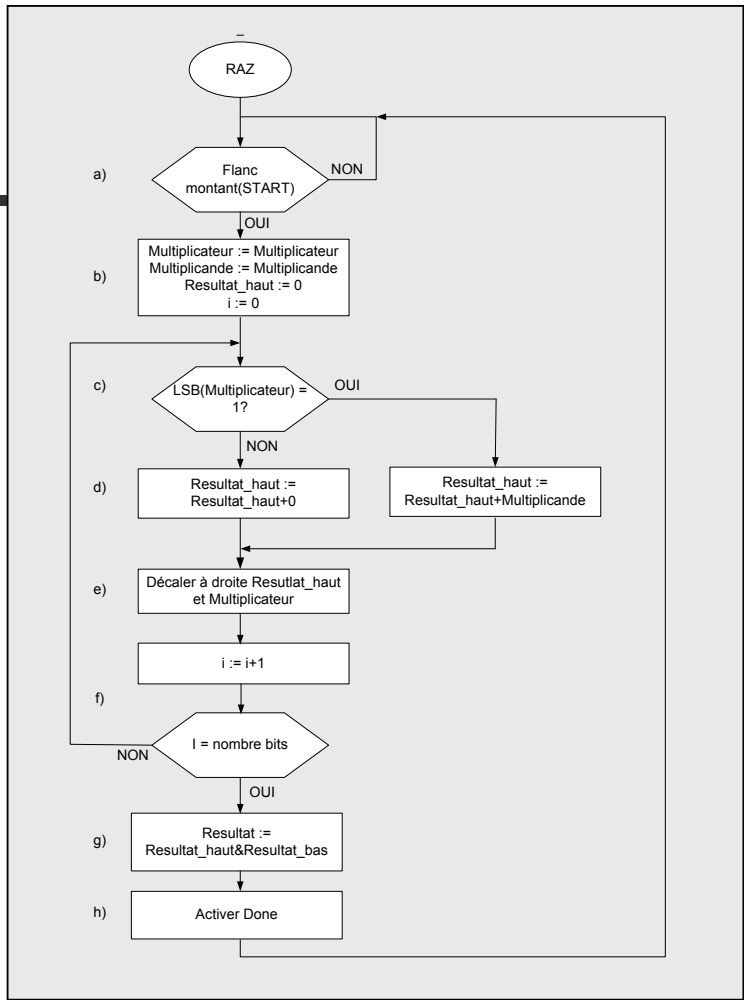
boîte rectangulaire : actions inconditionnelles



boîte oblongue : actions conditionnelles

Organigramme grossier

- Organigramme grossier correspondant à l'algorithme de la multiplication de nombres non-signés

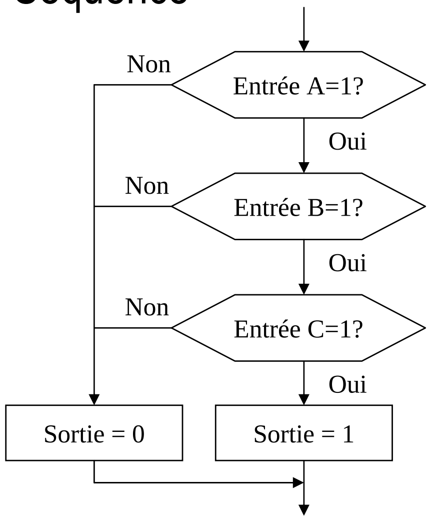


Partition UC / UT

Séquence ou fonction combinatoire?

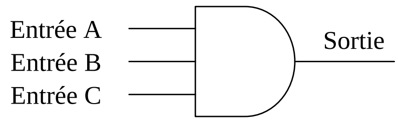
- Réaliser un ET logique avec trois signaux

- Séquence



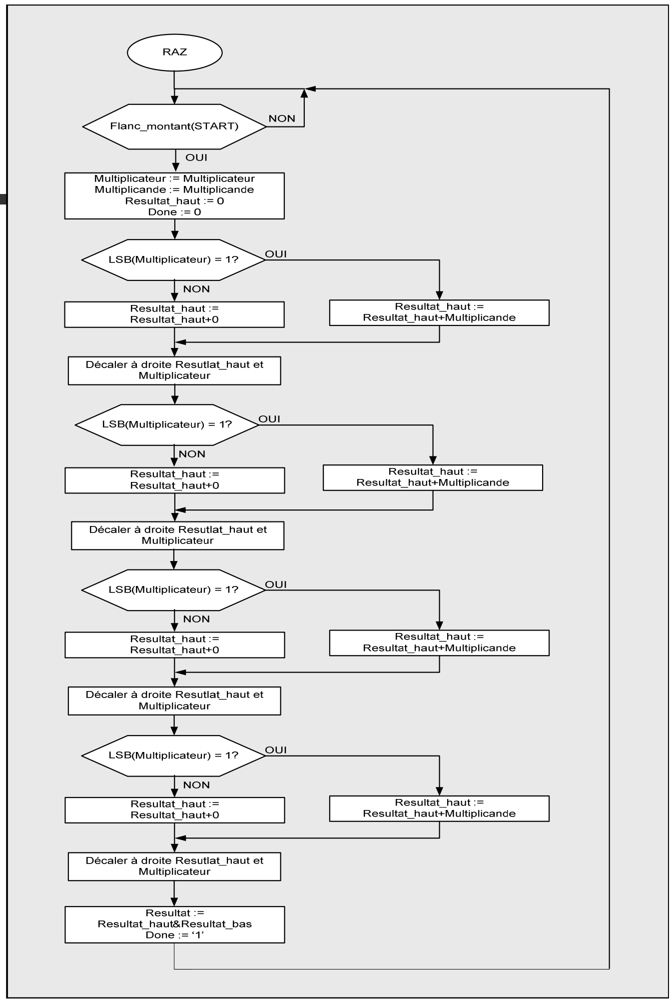
≡

- Porte combinatoire



Organigramme évolué

- Voici une 1ère évolution de l'organigramme
- Répétition de la séquence pour tous les bits du multiplicateur :
  - ✓ déroulée dans l'organigramme



Partition UC / UT

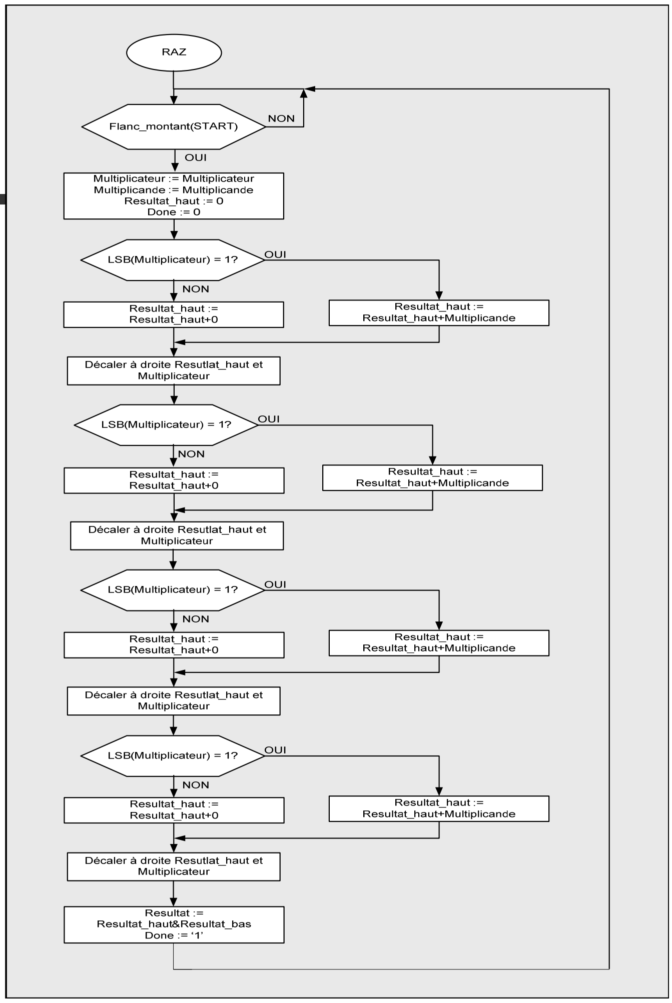
- Performance maximum → mettre toutes les fonctions identifiées dans une UT spécialisée
- Fonctions identifiées :
  - ✓mémorisation du résultat_haut, résultat_bas/multiplicateur, multiplicande
  - ✓test du LSB du multiplicateur : signal transmis à l'UC
  - ✓addition résultat_haut + (multiplicande ou 0)
  - ✓décalage Report, résultat_haut, résultat_bas/multiplicateur
  - ✓répéter l'algorithme pour tous les bits du multiplicateur

Unité de traitement UT

- L'unité de traitement comporte les composants suivants:
  - ✓2 registres, Result_H et Result_B_Mteur, avec chargement parallèle et décalage à droite
  - ✓1 multiplexeur 2 à 1 sur 4 bits pour sélectionner la valeur de chargement pour le registre Result_H
  - ✓1 registre, Mcande, avec chargement parallèle
  - ✓1 additionneur de 4 bits avec report
  - ✓1 bascule DFF pour mémoriser le report

Organigramme évolué

- Voici une 2ème évolution de l'organigramme
- Répétition de la séquence pour tous les bits du multiplicateur :
  - ✓ déroulée dans l'organigramme



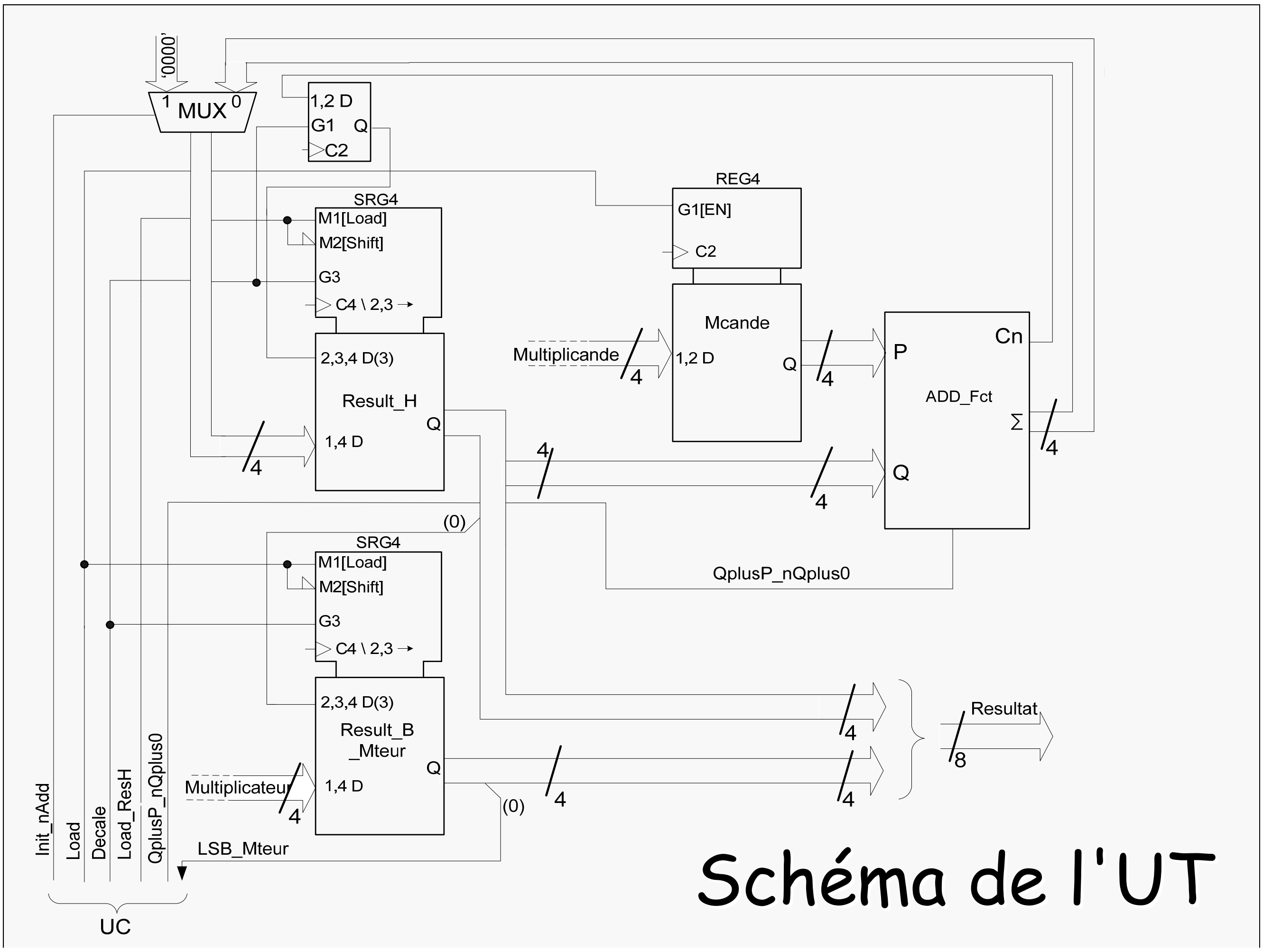


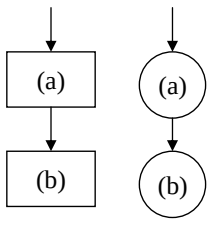
Schéma de l'UT

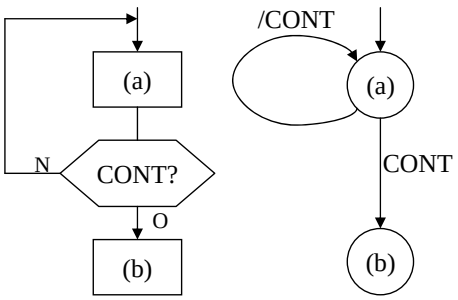
Passage organigramme => graphe d'états

- Graphe d'états :
  - ✓ évolution d'une MSS, période d'horloge par période d'horloge
  - ✓ Il faut adapter l'organigramme => version détaillée
- Organigramme détaillé :
  - ✓ Chaque boîte d'actions ne contient que les actions qui doivent / peuvent être déclenchées simultanément à ce stade de l'algorithme
  - ✓ Optimisation : mettre toutes les actions pouvant être simultanées dans la même boîte d'actions
  - ✓ L'organigramme ne contient que les noms des signaux connecté à l'UC

Organigramme ⇔ graphe

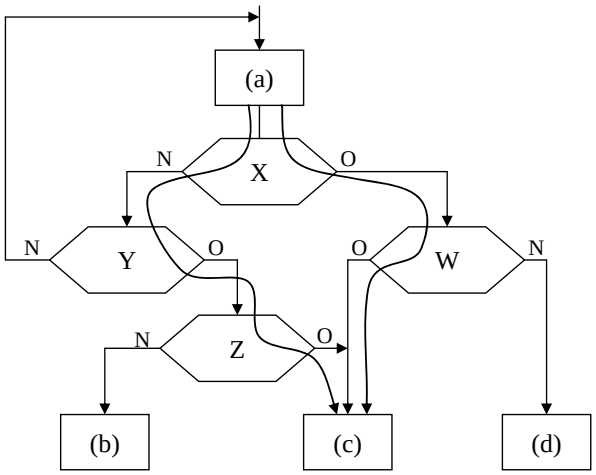
- boîte d'actions inconditionnelle = état dans un graphe
- conditions sur le(s) chemin(s) d'une boîte d'actions inconditionnelle à une suivante = condition de transition dans un graphe

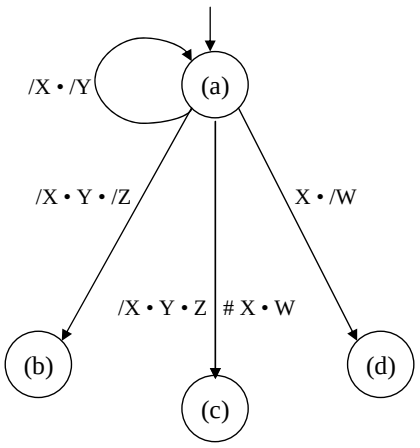




Organigramme ⇔ graphe

- Plusieurs chemins entre (a) et (c) sur un organigramme
  → une seule flèche sur le graphe



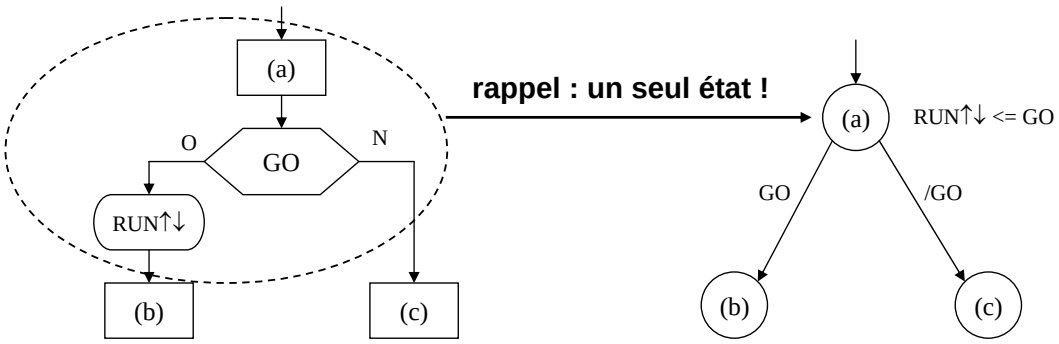


Organigramme et graphe détaillés

- Conventions
  - ✓les tests portant sur une entrée asynchrone seront signalés avec un astérisque *
  - ✓les sorties devant être générées sans transitoires ni aléas seront signalées avec 2 astérisques * *
  - ✓seuls les changements d'état des sorties seront indiqués (dans les boîtes d'actions ou états, 2 changements au maximum par sortie et par état)
  - ✓↑ : activation (attention : logique positive !), ↓ : désactivation, ↑↓ ou ↓↑ : impulsion

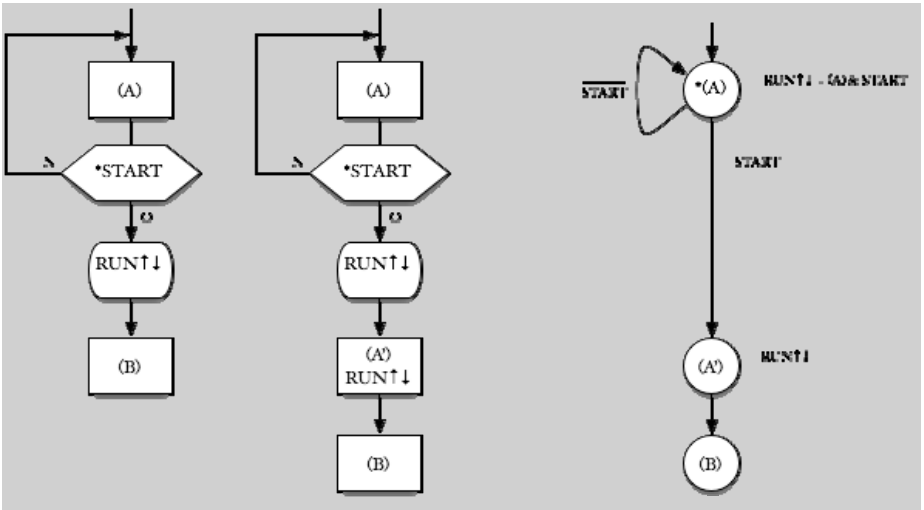
Sorties conditionnelles …

- Exemple :
  sortie RUN active dans l'état (a) si l'entrée GO est active



… sorties conditionnelles

- Attention aux sorties conditionnelles dépendant d'une entrée asynchrone
  - ✓ Garantir une durée supérieur à une période d'horloge

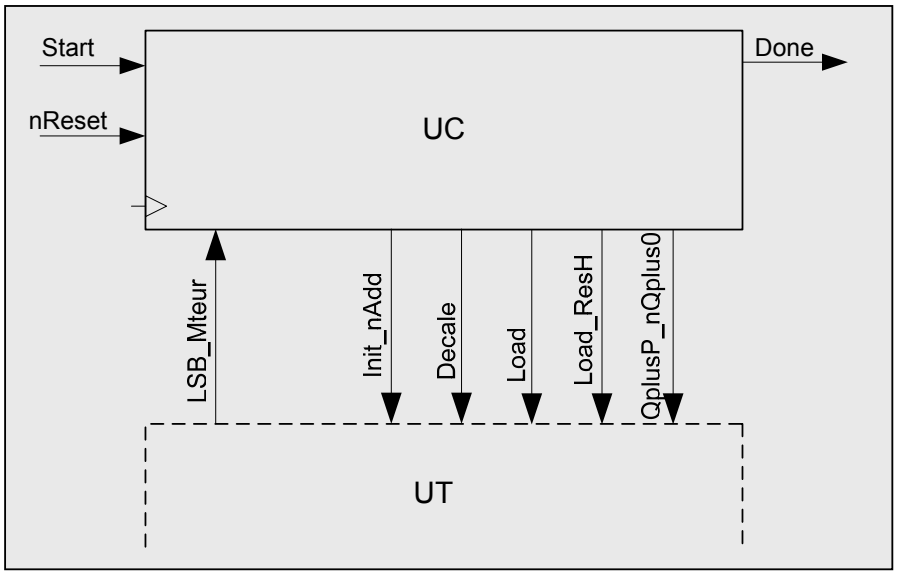


Organigramme détaillé de l'UC

- Décrit ce que doit faire l'UC (pas toute la MSS comme l'organigramme grossier)
- Tient compte de l'UT qui a été conçue
- Tient compte des contraintes temporelles
- Ses boîtes d'action ne comportent plus que des activations et désactivations des signaux de l'UC
- Accompagné de commentaires
- Obtenu par raffinements successifs

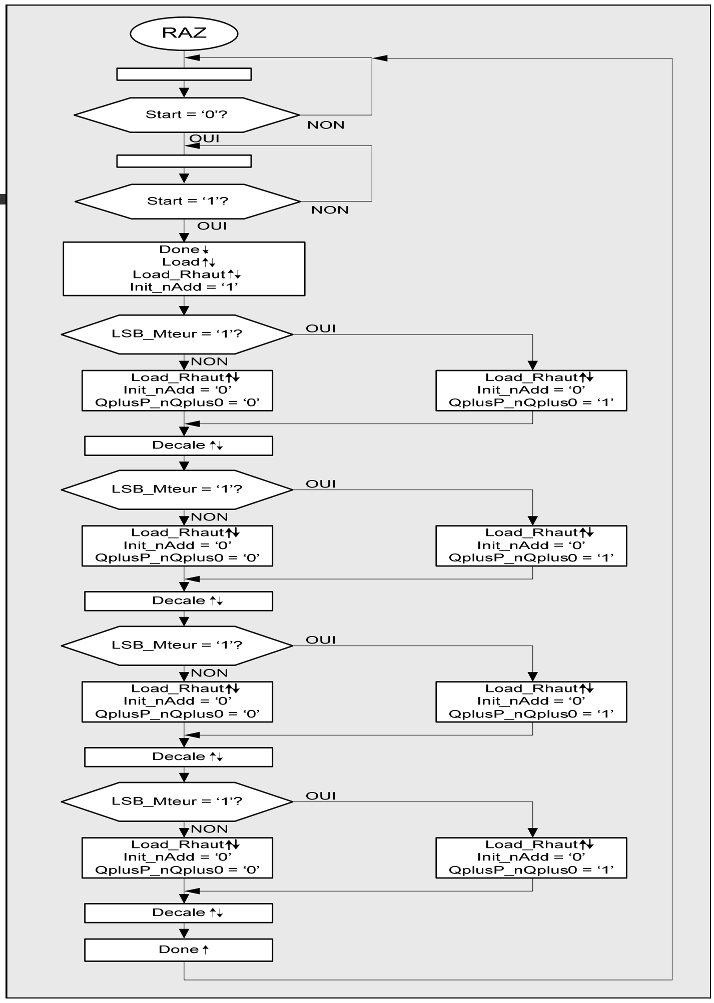
Schéma bloc UC

- Les signaux Start et Done sont directement connecté sur l'UC
- Les autres signaux sont ceux connecté à l'UT



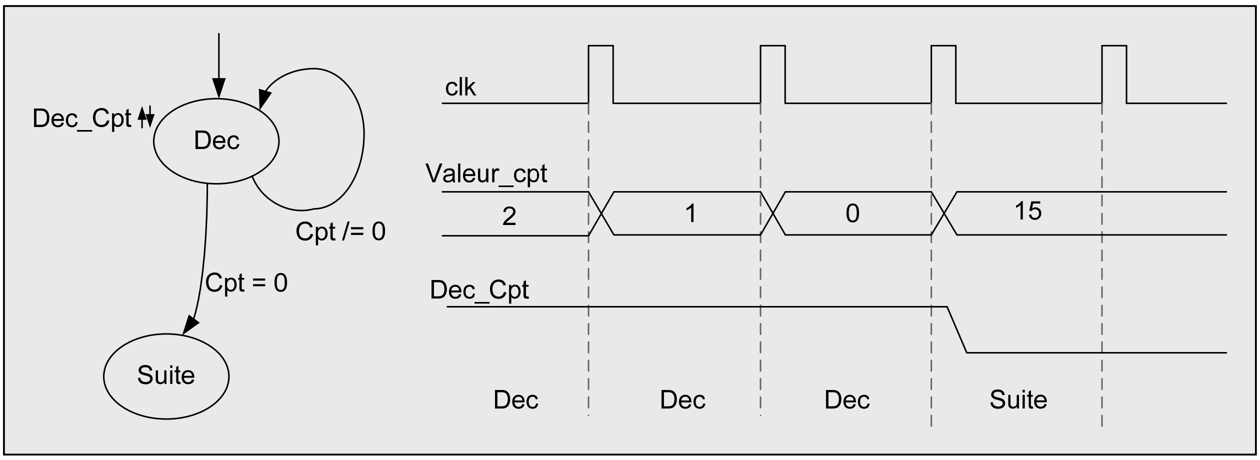
Organigramme détaillé

- Voici l'organigramme détaillé correspondant au schéma de l'UT spécialisé
- Répétition de la séquence pour tous les bits du multiplicateur :
  - ✓ déroulée dans l'organigramme



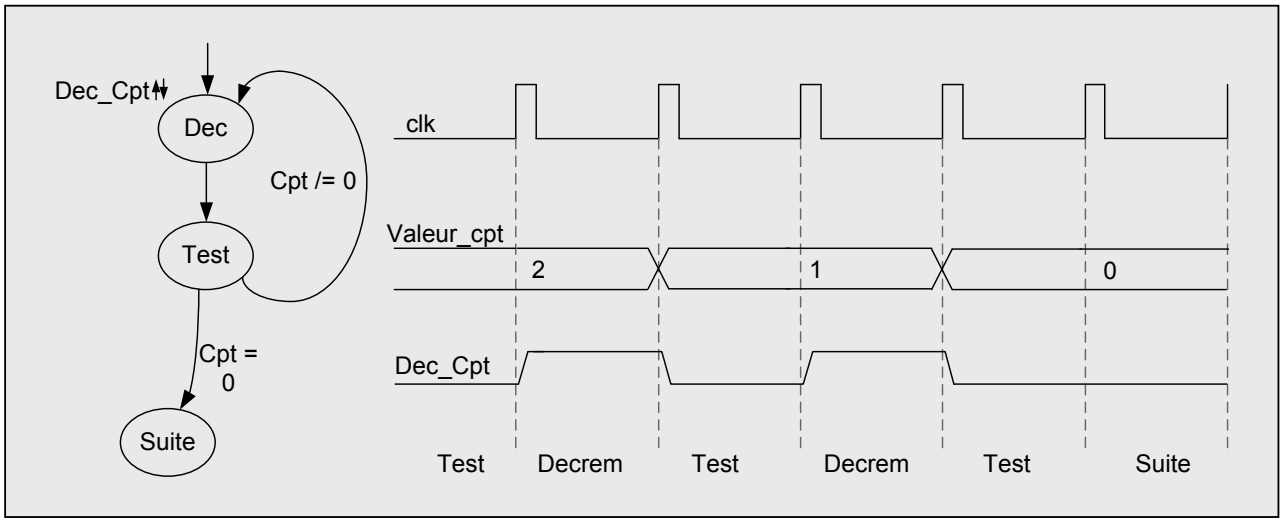
Fonctionnement incorrect

- Le graphe ci-dessous montre le cas d'un décomptage avec le test de l'état zéro.
- Lorsque l'état zéro est détecté, nous quittons l'état Dec. Mais le compteur est décrémenté encore une fois.



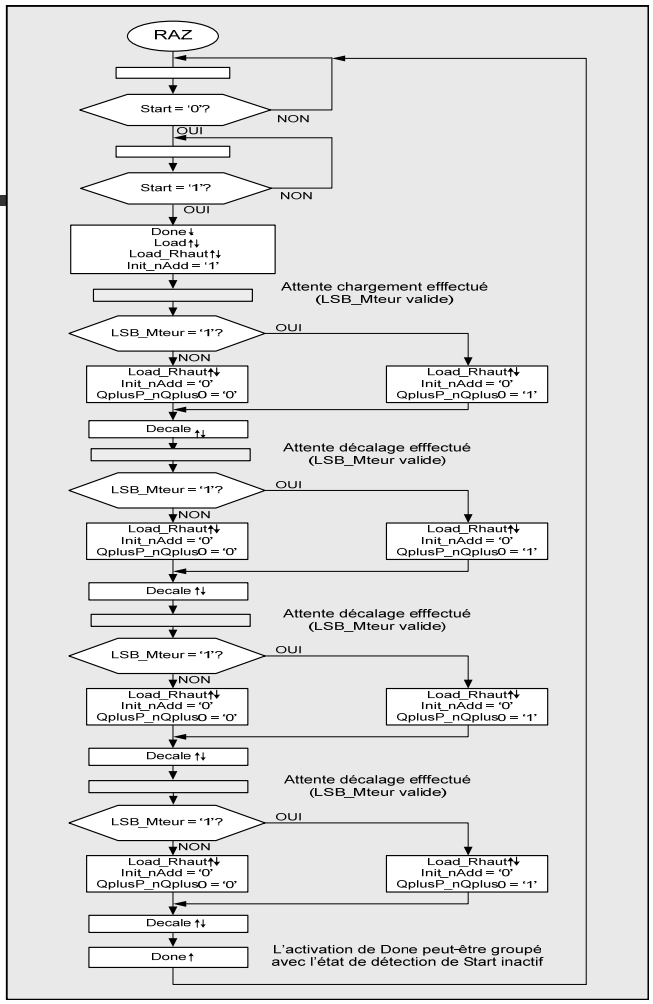
Correction du fonctionnement

- La correction est rajouter un état pour tester l'état du compteur. Dans cet état, le décomptage est désactivé.



Organigramme détaillé final

- Voici l'organigramme détaillé correspondant au schéma de l'UT spécialisé et aux corrections mentionnées préalablement
- Répétition de la séquence pour tous les bits du multiplicateur :
  - ✓ déroulée dans l'organigramme



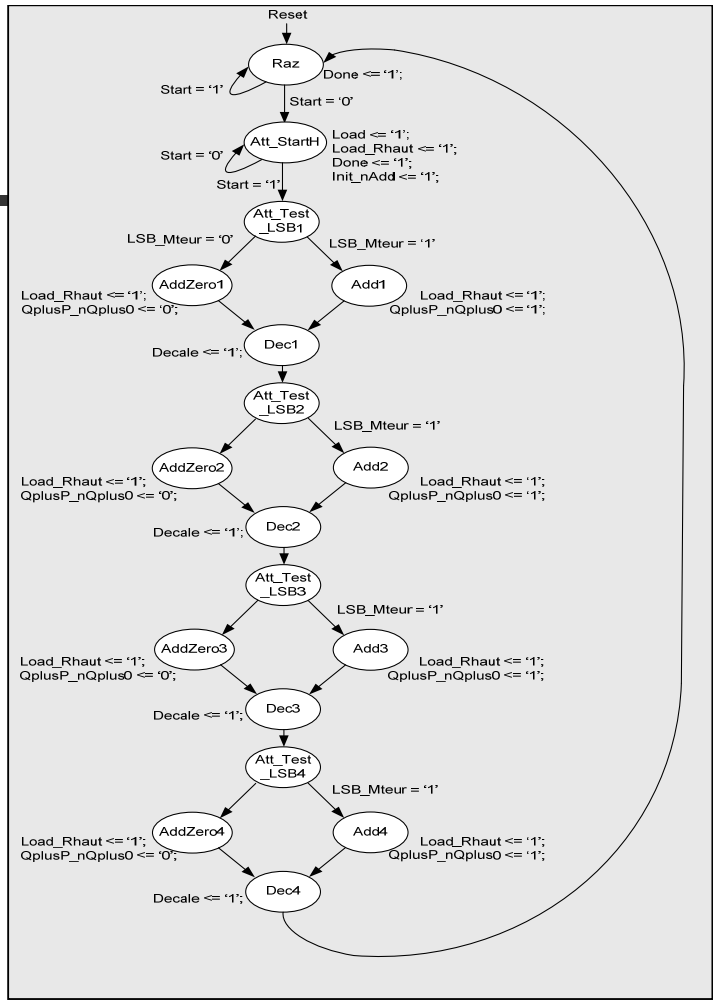
Graphe des états de l'UC

- Voici le graphe des états avec l'ajout d'un état pour tester le résultat de l'addition

  Optimisation:

  → L'initialisation de l'UT a été déplacée dans l'état Att_StartH

  → La désactivation Done est déplacé dans l'état Att_Test_LSB1



Dia volontairement laissé vide

Exercices série I

1. Réaliser une version permettant la multiplication de deux nombres de N bits.
   Proposer une modification de l'organigramme et de l'UT

2. Etablir le graphe des états correspondant à l'exercice précédent (1.)
   Proposer une optimisation du décodeur de sorties sachant que les signaux Init_nAdd et QplusP_nQplus0 ne sont pas toujours utilisés.

Exercices série I (suite)

3. Proposer une solution pour supprimer le coup d'horloge utilisé nécessaire pour le décalage.
   Dans la version proposée, il y a un coup d'horloge pour le calcul et un second pour le décalage. En fait, il y a un troisième coup d'horloge pour tester le bit du multiplicateur.

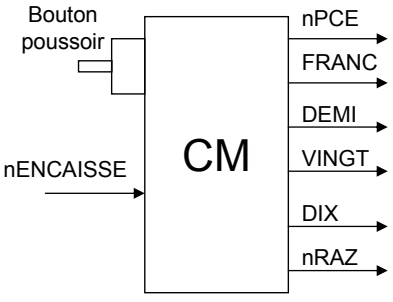
4. Etablir le graphe des états correspondant à l'exercice 3.

Exemple : distributeur de billets

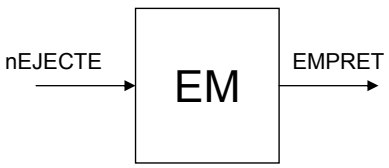
- Etapes de la conception du distributeur de billets
  - ✓cahier des charges
  - ✓schéma bloc global
  - ✓algorithme (organigramme)
  - ✓partition UC / UT (choix)
  - ✓schéma bloc de l'UT
  - ✓schéma de l'UT avec ICs standard

Exemple : distributeur automatique de billets

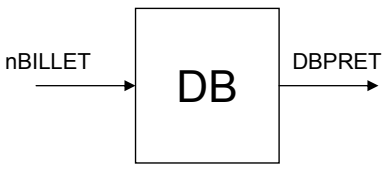
- Collecteur de Monnaie (CM)



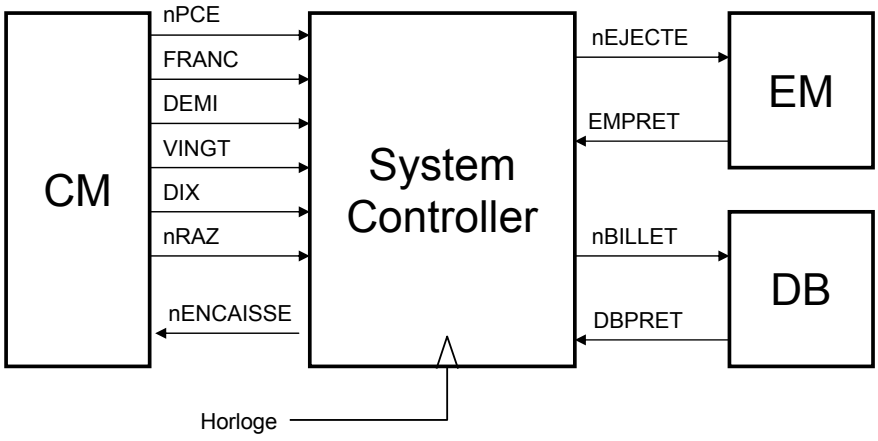
- Echangeur de Monnaie (EM)



- Distributeur de Billets (DB)



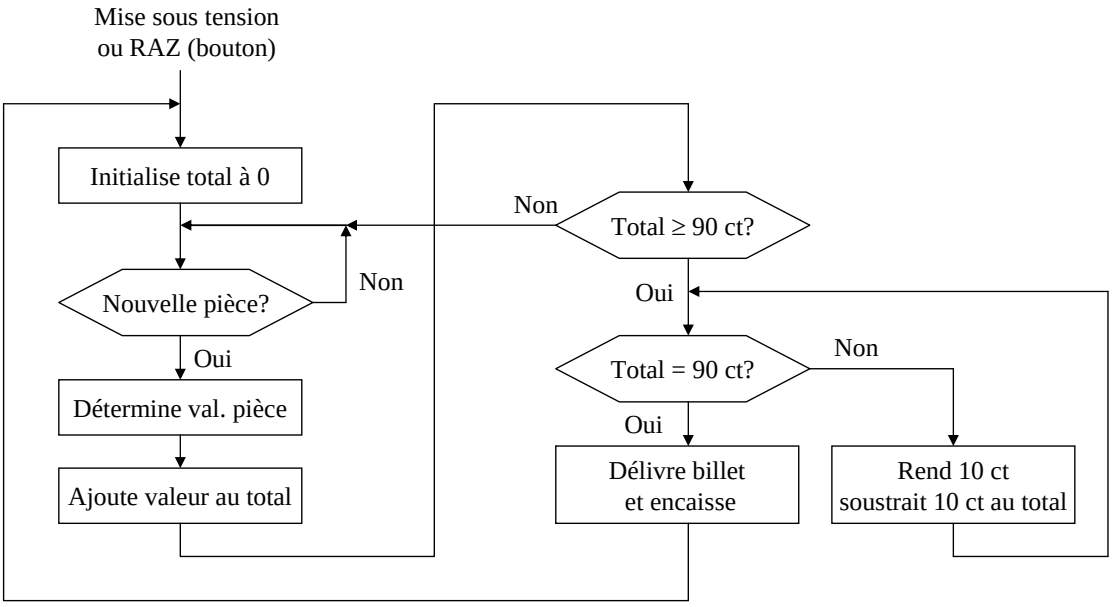
Distributeur de billets : schéma bloc



Distributeur de billets :

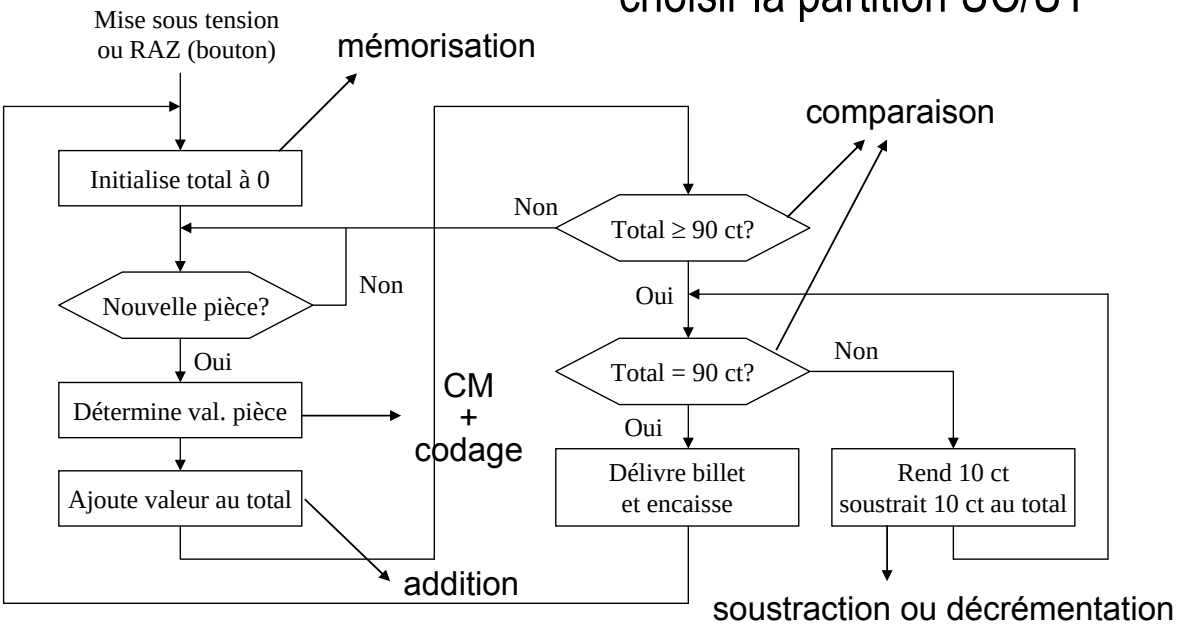
- Spécification détaillée : voir document
  Cours : Electronique numérique. Tome 4
- Fonctionnement : voir démonstration Tcl/Tk
  Distributeur_90cts.tcl
- Etablissement d'un organigramme grossier
  (1ère version)

Distributeur de billets : algorithme global (organigramme)



Identification des fonctions

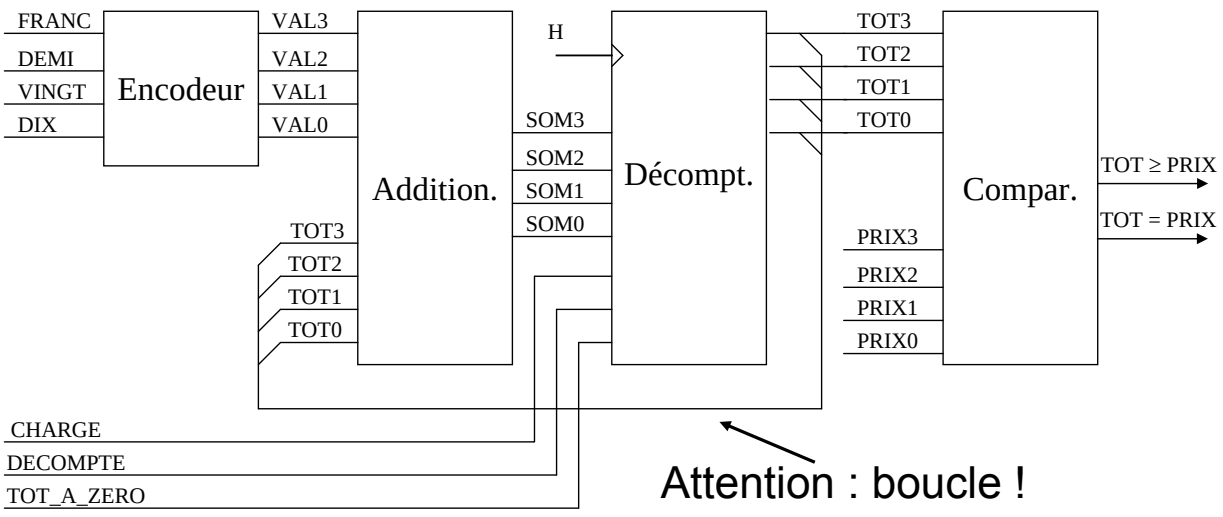
- Identification de fonctions standard dans l'algorithme pour choisir la partition UC/UT



Choix et spécification des fonctions

- Encodage de la valeur : nombre équivalent de pièces de 10 ct.
  - ✓unité des entiers utilisés dans le système : 10 cts
- Mémorisation et décrémentation du total dans un décompteur 4 bits
  - ✓Initialiser total à 0 => initialisation à 0
  - ✓Mémorisation => charger
  - ✓Soustraire 10 cts => décrémentation
- Additionneur 4 bits
- Comparateur 4 bits

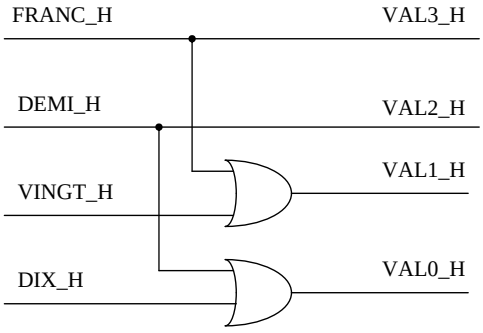
Schéma bloc de l'UT choisie



Réalisation avec fcts standard

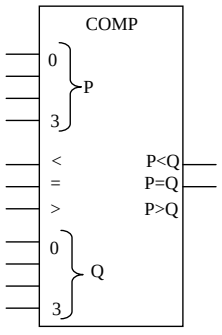
- Encodeur

```
Val(3) <= '1' when Franc = '1'  else '0';
Val(2) <= '1' when Demi = '1'   else '0';
Val(1) <= '1' when (Franc or Vingt) = '1' else '0';
Val(0) <= '1' when (Demi or Dix) = '1'    else '0';
```

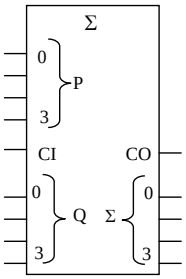


Réalisation avec fcts standard

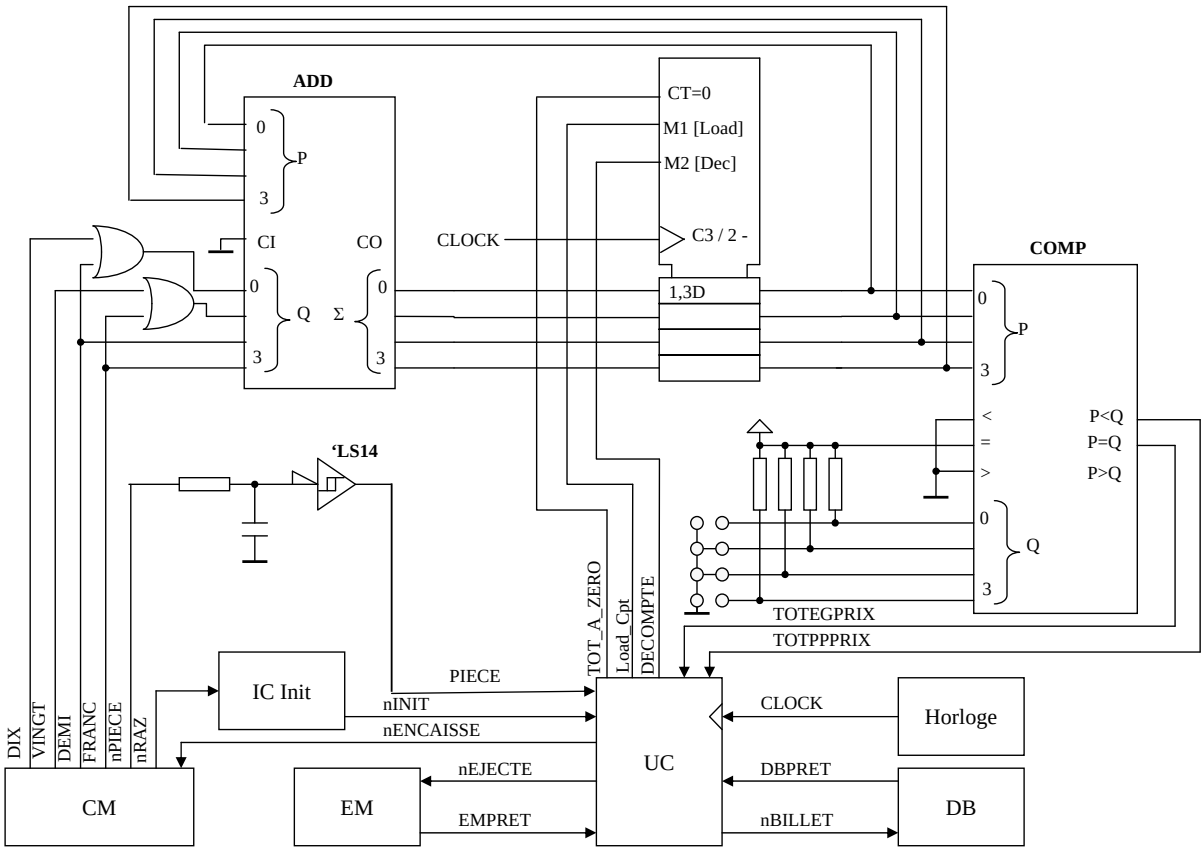
Comparateur 4 bits :



Additionneur 4 bits :



Décompteur 4 bits avec chargement synchrone (suprime la boucle) et remise à 0 :



Exercices série II

II.1 Modifiez l'UT de la diapositive 73 de façon à ce que le vendeur de billets fonctionne correctement lorsqu'un client introduit une pièce de 1 Fr après avoir déjà introduit 80 ct. Adaptez l'organigramme général s'il y a lieu.

II.2 Adaptez le schéma de l'UT et affinez l'organigramme général, de façon à ce que les comparaisons ne soient plus faites dans l'UT.

Exercices série II

II.3 Elaborez un algorithme dans lequel on calcule le solde à payer et non la somme payée. Dessinez l'organigramme général, identifiez les fonctions qui peuvent être réalisées dans une UT spécialisée et décrivez celle-ci (schéma bloc, VHDL ou schéma de détail avec ICs standard).

Exercices série II

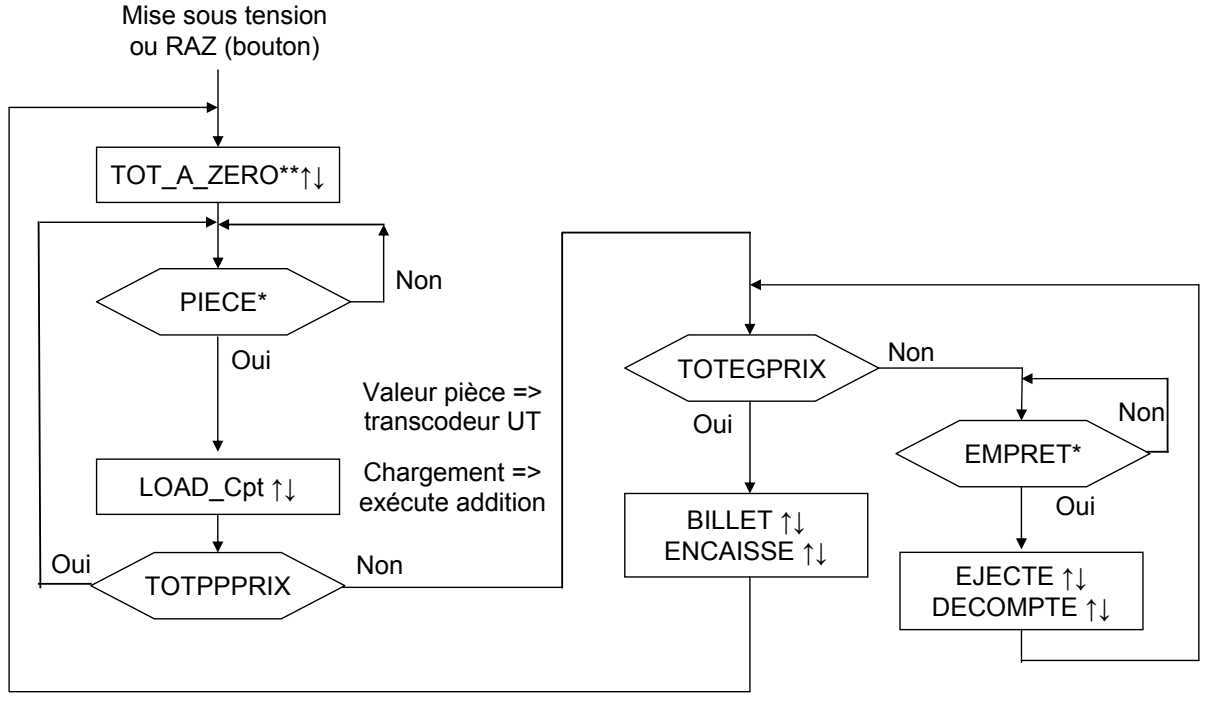
II.4 Modifiez l'organigramme de la diapositive 67 et l'UT de la diapositive 73, de façon à calculer le montant payé par comptage plutôt que par addition.

II.5 En faisant la synthèse des exercices précédents, réduisez l'UT spécialisée au strict minimum et adaptez (complétez) l'organigramme.

Organigramme détaillé

- Refaire l'organigramme avec uniquement les signaux de l'UC
- Tenir compte de la nécessité de respecter le fonctionnement des périphériques
  - ✓Avant de rendre une pièce, il faut être sûr que l'échangeur de monnaie soit prêt. Rajouter ce test.

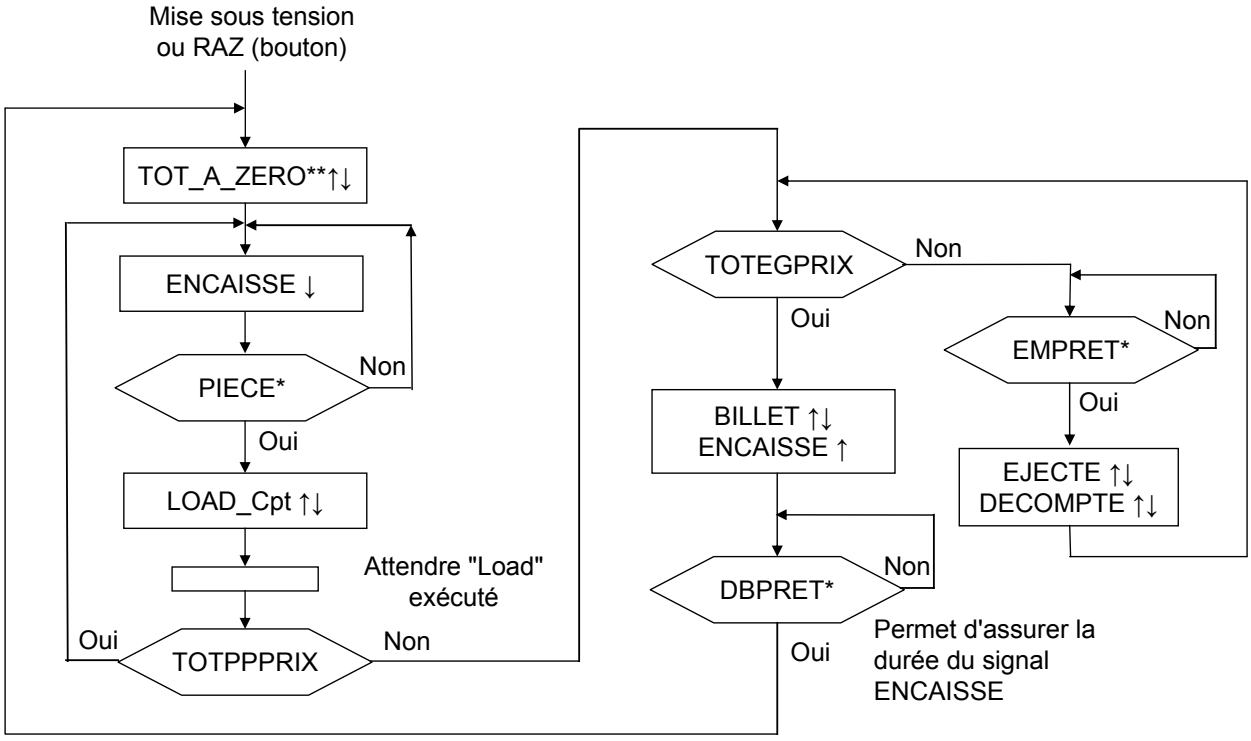
Organigramme détaillé de l'UC : 1er raffinement



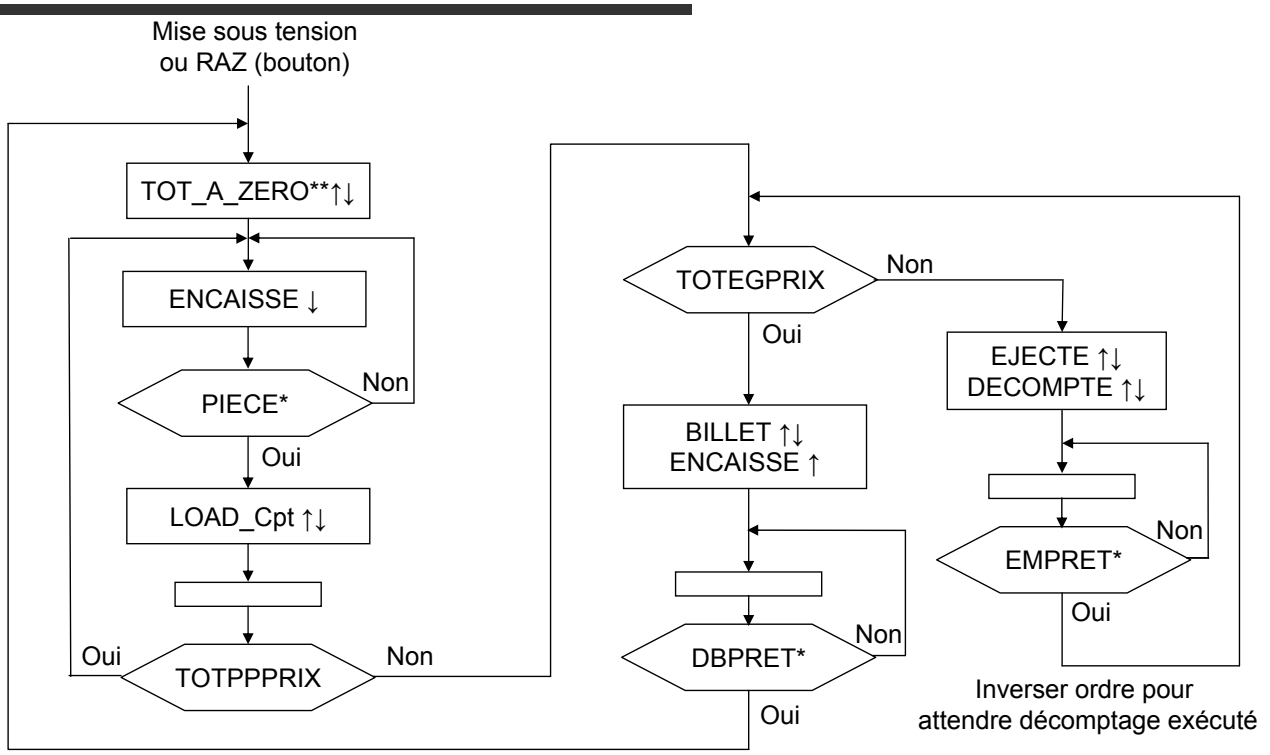
Organigramme détaillé 2ème raffinement

- Tenir compte des timings de fonctionnement du système.
- Choix de l'horloge du système
- L'organigramme suivant est prévu pour une horloge entre 10 et 20 Hz, d'où une période de 100 à 50 ms

2ème raffinement : Tclock = 50 à 100 ms

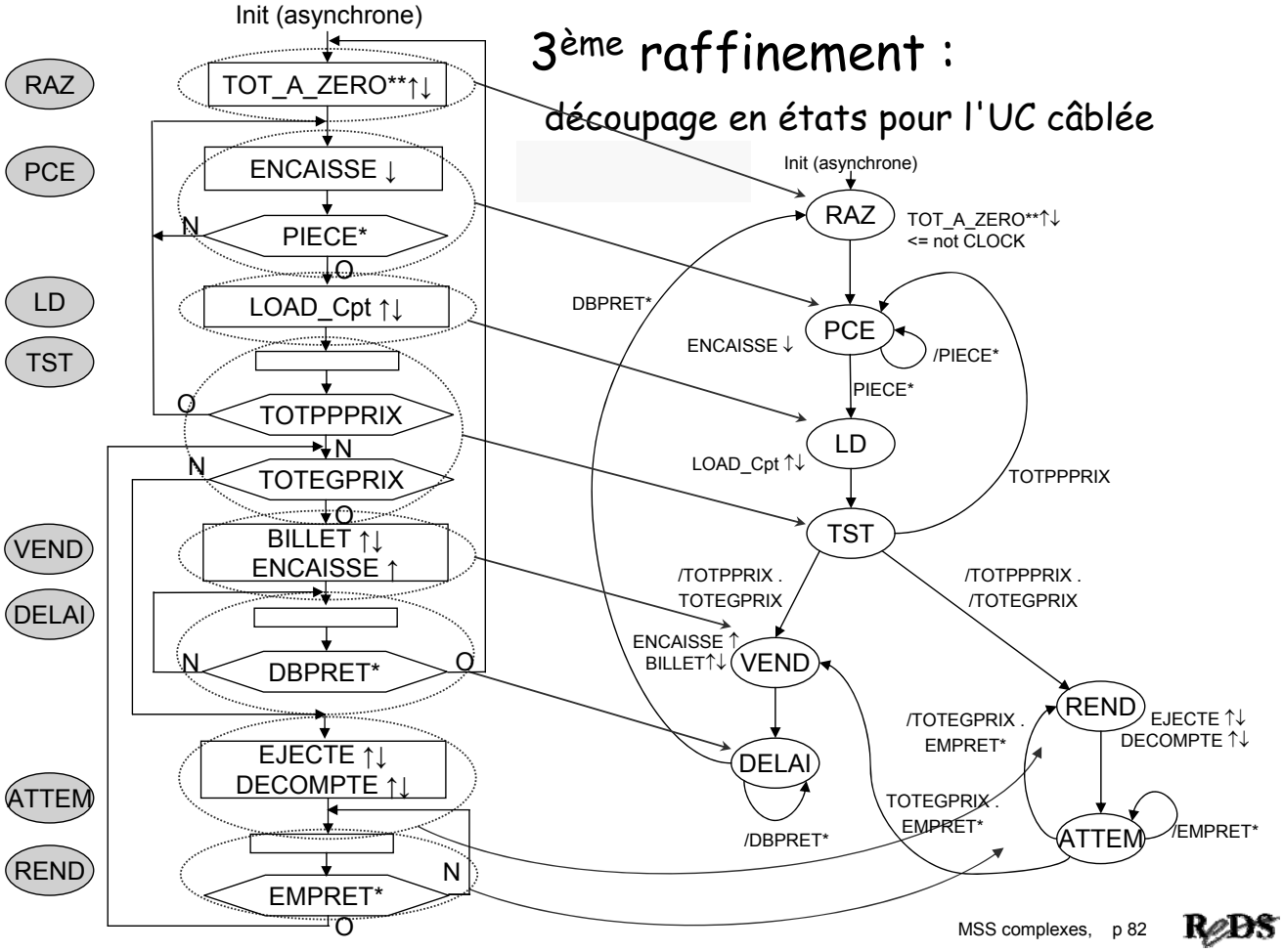


3ème raffinement : passage au graphe



3ème raffinement :

découpage en états pour l'UC câblée



Exercices série III

III.1 Faites une description VHDL de l'UC câblée de la diapositive 79.

III.2 Pour chacune des unités de traitement développées dans la série d'exercices précédente (série II), dessinez un organigramme détaillé et un graphe pour l'UC câblée correspondante.

III.3 Réalisez l'UC câblée de la diapositive 79 à l'aide d'une mémoire et d'un registre parallèle-parallèle.

Exercices série III

III.4 Codez les états du graphe de la diapositive 79 dans l'ordre numérique, avec 000 pour RAZ et 111 pour DELAI. Prenez un compteur 4bits (fcts load et Inc) comme générateur d'états. Etablissez la table de vérité des commandes à appliquer au compteur pour générer la séquence souhaitée.

Exercices série III

III.5 Le codage précédent ne convient pas avec les entrées asynchrones Piece, EMPret et DBPret : il faudra les synchroniser. Le déroulement de la séquence dépend parfois de deux entrées. Il faut modifier l'organigramme du dia 79 afin de tester une seule entrée à la fois. Ensuite, il est possible de sélectionner l'entrée concernée en fonction de l'état de l'UC, et de n'utiliser qu'une seule bascule de synchronisation. Réalisez une UC en exploitant ce qui précède.

Dia volontairement laissé vide

Evolution vers UC microprogrammée

- Idées menant à une UC microprogrammée
- 1ère et 2ème UC microprogrammée pour le vendeur de billets
- Notre premier microséquenceur
- Microprogramme pour l'UC du vendeur de billets

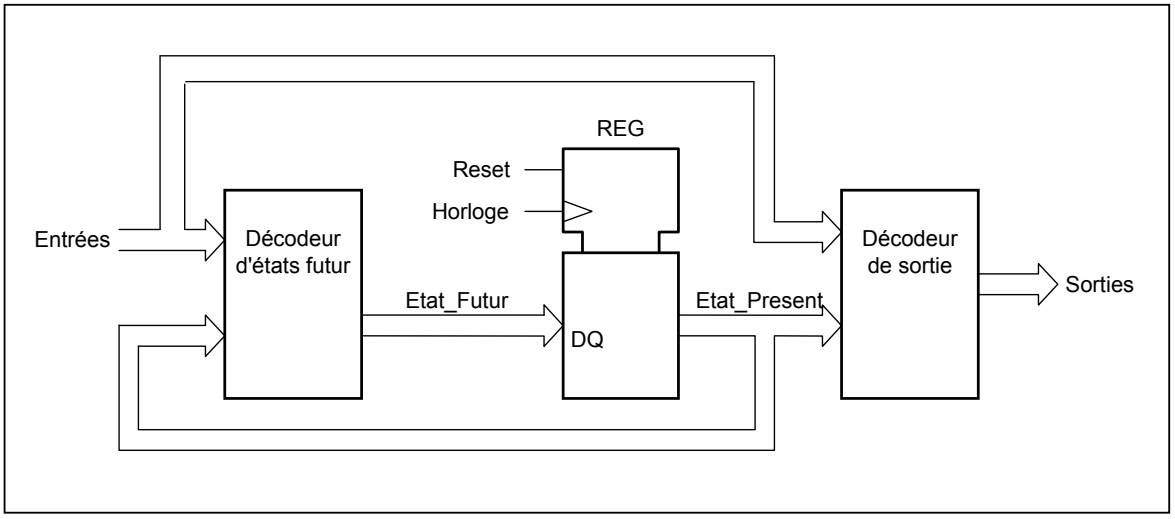
Objectif : arriver à la structure d'un processeur

Idées menant à une UC μprogrammée

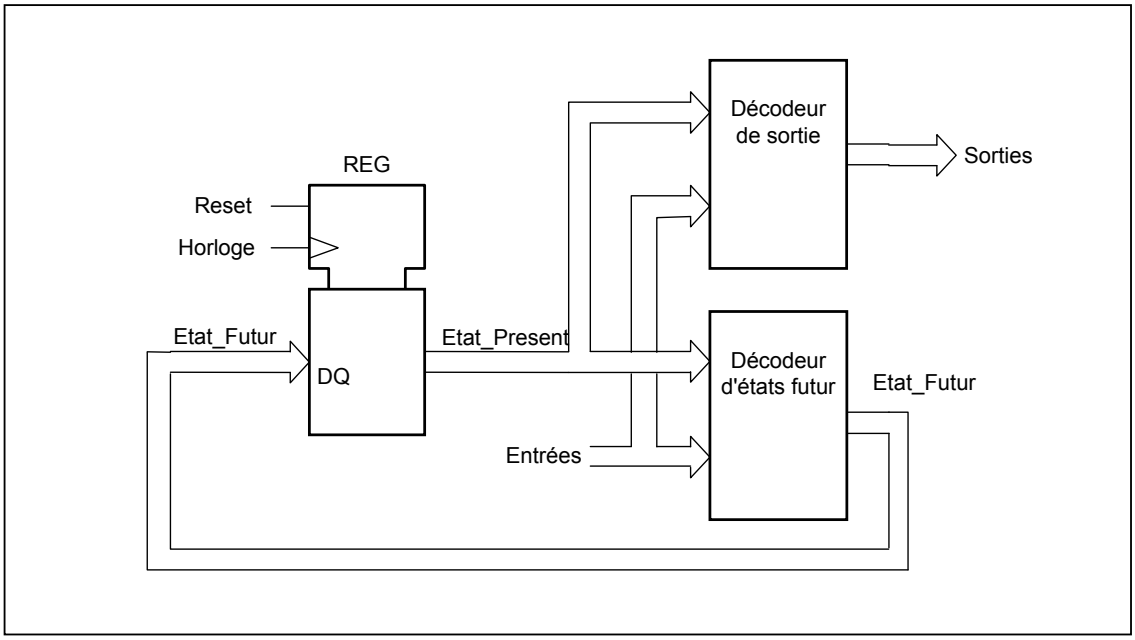
- Réaliser le décodeur d'état futur (décodeur des actions de séquencement) et le décodeur de sorties avec une ROM (mémoire)
  - ✓conception facilitée (table de vérité)
  - ✓réalisation modulaire
  - ✓schéma standard
  - ✓modifications faciles
  - ✓haute intégration

Décomposition d'un système séquentiel

UC câblée correspond à cette décomposition :

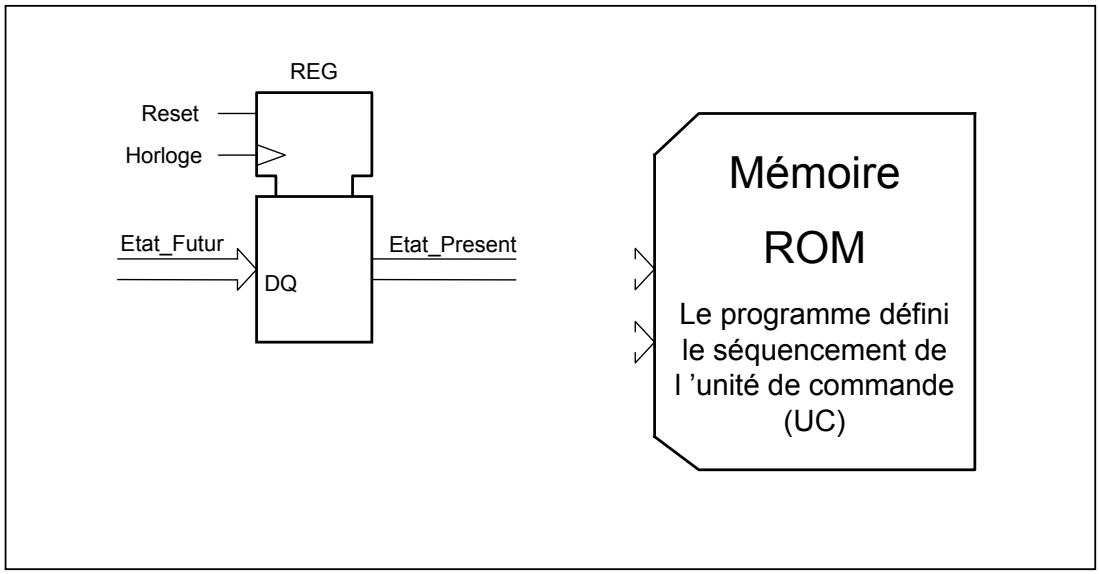


Autre représentation structure UC câblée



Utilisation d'une mémoire ROM

La mémoire est un système combinatoire. Elle est utilisée pour remplacer les deux décodeurs.

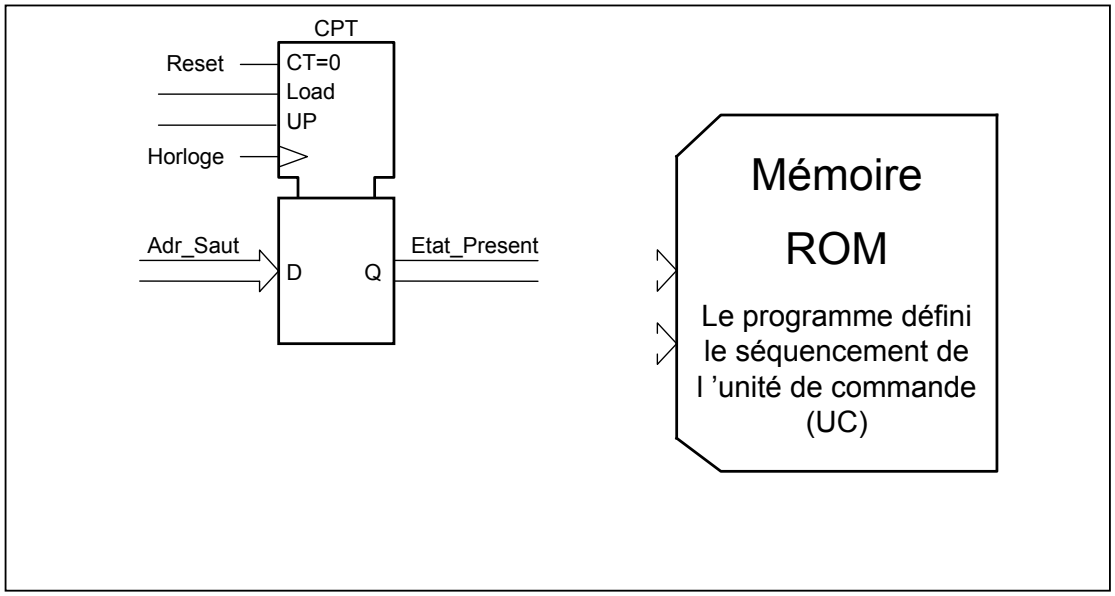


Idées menant à une UC programmée

- Utiliser un compteur comme générateur d'états
  - ✓fréquemment le code des états se suivent
  - ✓permet de passer à l'état suivant avec une commande très simple (mode, 2 bits), prenant peu de place dans la ROM
  - ✓modulaire
  - ✓règle no5 pas respectée => synchroniser entrées

Utilisation : mémoire ROM + compteur

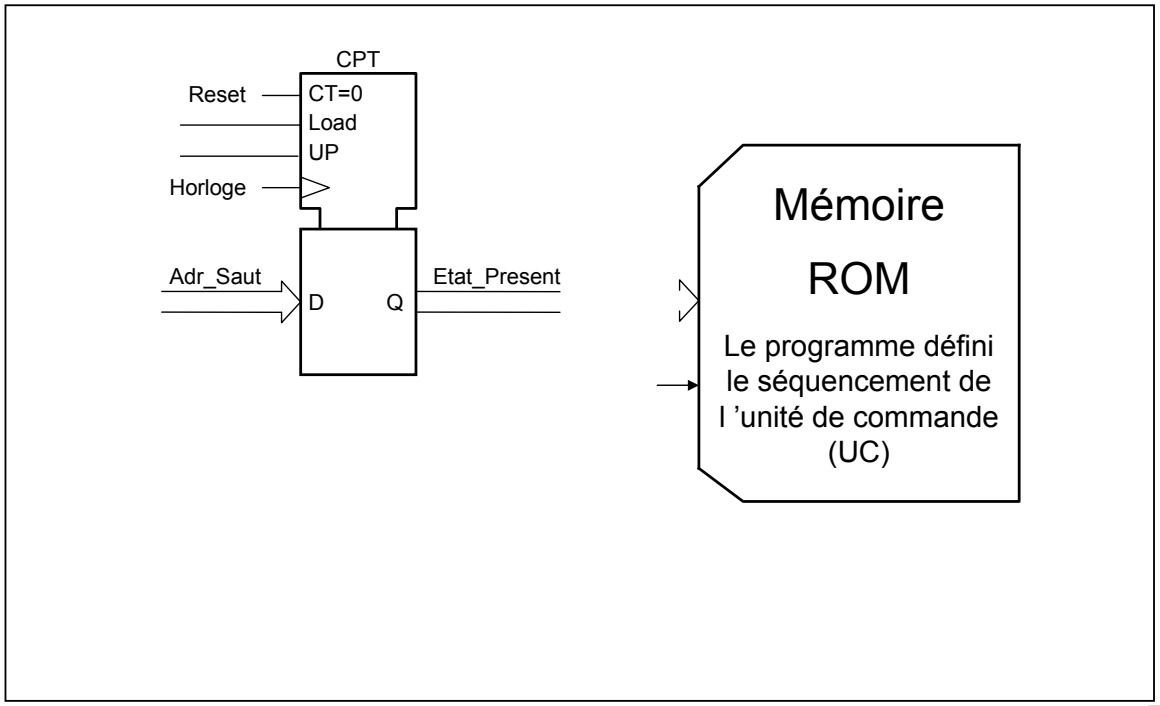
Si les états sont codés avec des codes représentants des valeurs croissantes : l 'utilisation d 'un compteur est préférable (incrém.)



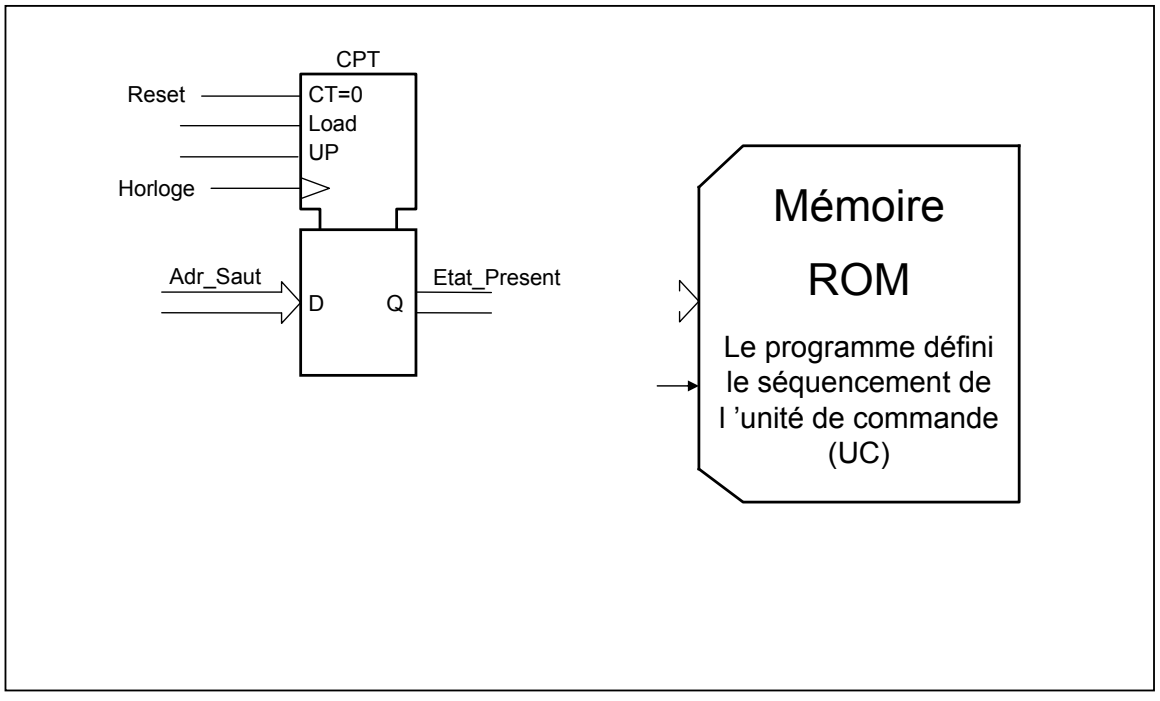
Idées menant à une UC μprogrammée

- Multiplexer les entrées de condition
  - ✓l'évolution du graphe ne dépend pas de TOUTES les entrées
  - ✓diminue le nombre d'entrées dans le décodeur d'actions de séquencement (la taille de la ROM)
  - ✓diminue le nombre de bascules de synchronisation
  - ✓modulaire
  - ✓schéma standard

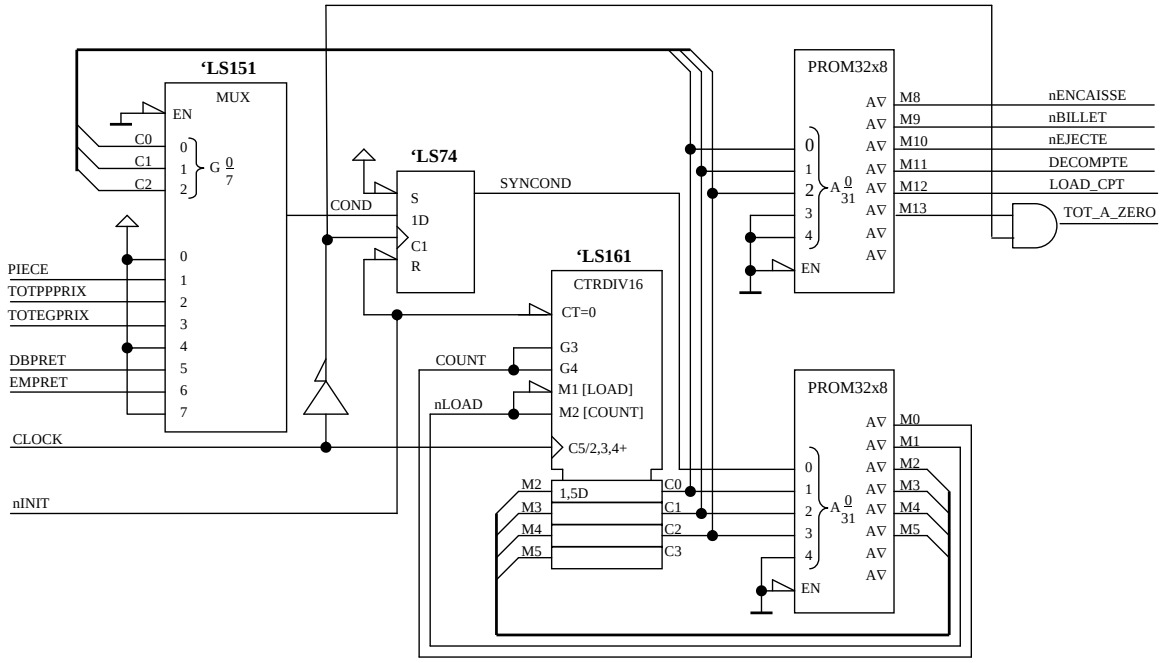
Diminution nbr adresses mémoire



Respect de la règle no 5 : synchroniser



1re UC μprogrammée pour le vendeur de billets



Inconvénients cette 1ère structure d'UC μprogrammée

- Généralement : nb. d'états (μinstructions) >> nb. d'entrées (conditions distinctes)

  → mux d'entrées >> que nécessaire
- Une modification du μprogramme va généralement demander un changement de câblage du mux
- Le décodeur d'actions de séquencement a une entrée de plus que le décodeur de sorties

  (→ mémoire profondeur double)

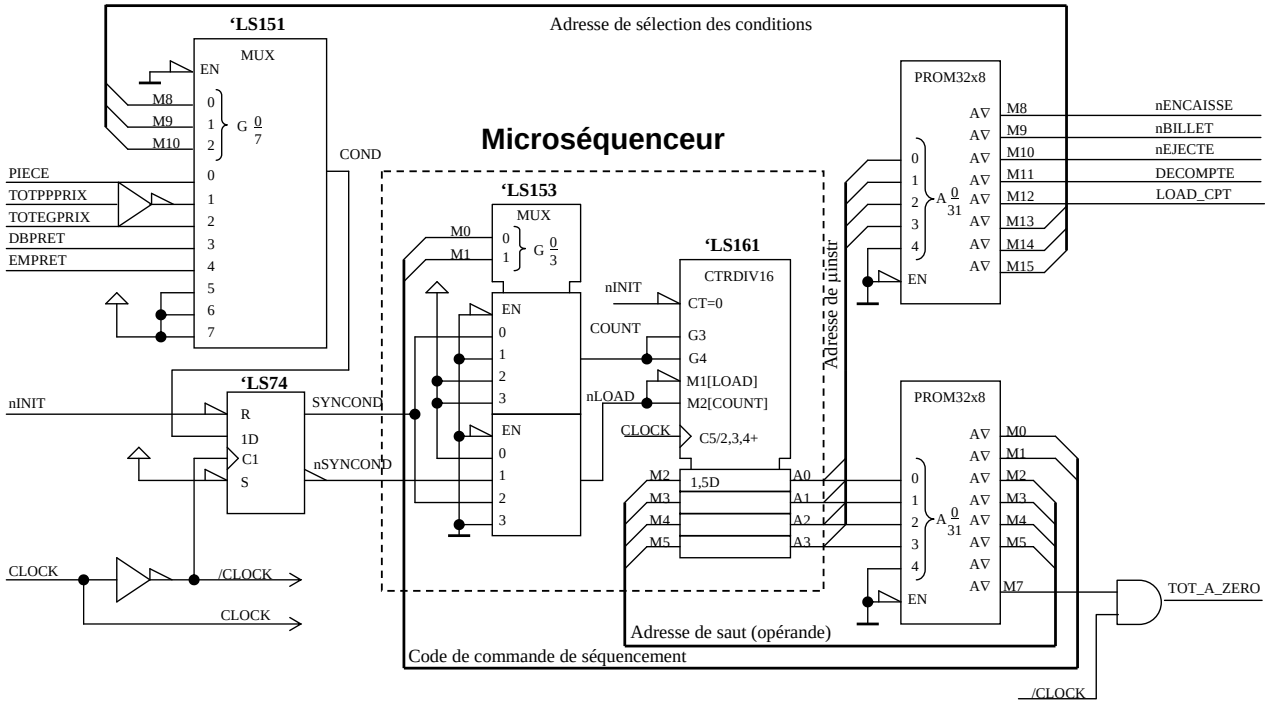
Améliorations

- Choisir un mux en fonction du nombre d'entrées (conditions distinctes) et le câbler sans tenir compte de l'algorithme
- Sélectionner de la condition à partir de la μinstruction (modifiable) et non à partir de l'état

Améliorations

- Décomposer le décodeur d'actions de séquencement en 2 parties :
  - ✓une partie fixe qui commande le compteur à partir d'un code d'opération et de la condition (combinée au compteur → μséquenceur)
  - ✓une partie variable qui fournira le code de l'opération et qui sera placée dans la ROM

2ème UC μprogrammée pour le vendeur de billets



Commandes du μséquenceur

Action	Mnémonique	Code (M1,M0)	COUNT	nLOAD
Compte si Condition ou Maintient μPC <= μPC+1 si condition vraie, μPC sinon	CCM	00	SYNCOND	1
Saute si Condition ou Maintient μPC <= Adr. (saut) si cond. fausse, μPC sinon	SCM	01	0	nSYNCOND
Compte si Condition ou Saute μPC <= μPC+1 si cond. vraie, Adr. Sinon	CCS	10	1	SYNCOND
Saute Inconditionnellement μPC <= Adr. (saut inconditionnel)	SI	11	-	0

Table de vérité du décodeur de commandes

Format d'une μinstruction

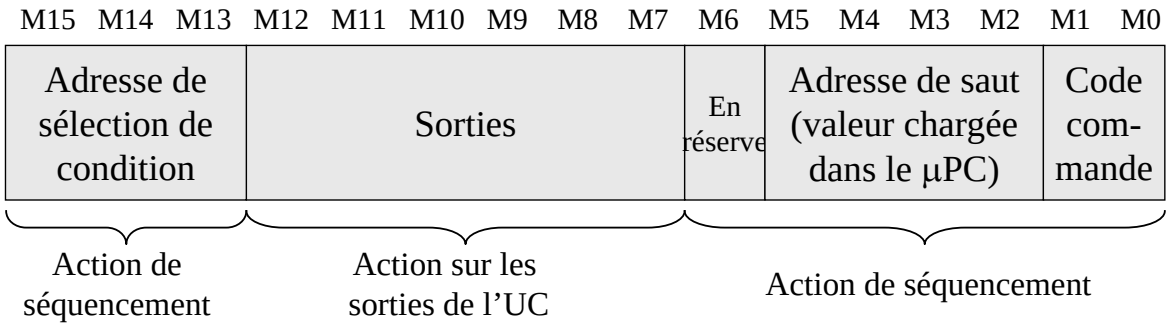
- (code d'une) μinstruction : assemblage de plusieurs plages contenant des commandes et des paramètres, en un mot mémoire
- format d'une μinstruction : définit la position, la taille et le contenu des différentes plages
- format fixe : toutes les μinstructions ont le même format (sinon : format variable)

Format d'une μinstruction

- μprogrammation horizontale : chaque plage du format ne peut contenir qu'un seul type d'information
- exemple : format des μinstructions de l'UC de la diapo 101 (μprogrammation horizontale)

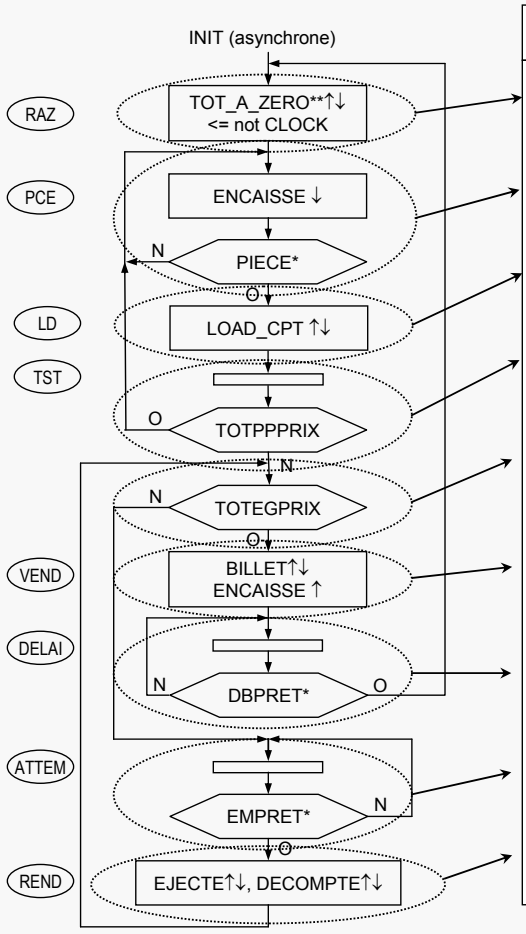
Format d'une μinstruction

- exemple : format des μinstructions de l'UC de la diapo 101 (μprogrammation horizontale)



μprogramme

- μprogramme :
  - ✓liste ordonnée de μinstructions
  - ✓traduction de l'organigramme détaillé, dans le langage de l'UC μprogrammée
  - ✓en texte (mnémoniques) = en langage d'assemblage
  - ✓en binaire = code (en langage machine)
- exemple : μprogramme pour le vendeur de billets avec l'UC de la diapo 101



Adr	Sorties actives	Sélection Cond	Adresse saut	Cmd séq.	PROM B	PROM A
0000	TOT_A_ZERO ENCAISSE	VRAIE	-	CCM (CCS)	11100110	1-----00
0001	aucune	PIECE	-	CCM	00000111	0-----00
0010	LOAD_CPT	VRAIE	-	CCM	11110111	0-----00
0011	aucune	TOTPPPRIX	0001 (PCE)	CCS	00100111	0-000110
0100	aucune	TOTEGPRIX	0111 (ATTEM)	CCS	01000111	0-011110
0101	BILLET ENCAISSE	VRAIE	-	CCM (CCS)	11100100	0-----00
0110	ENCAISSE	DBPRET	0000 (RAZ)	SCM	01100110	0-000001
0111	aucune	EMPRRET	-	CCM	10000111	0-----00
1000	EJECTE DECOMPTE	VRAIE	0100 (TST2)	SCM	11101011	0-010001

assembleur | code

Exercices série IV

IV.1 Dans l'UC de la diapositive 101, synchronisez la condition sur le flanc montant de l'horloge. Adaptez l'organigramme détaillé et le μprogramme.

Exercices série IV

IV.2 Modifiez le μséquenceur de la diapositive 101 de façon à ce qu'il exécute le jeu de commandes suivant :

Action	Mnémo.	Code
Compte inconditionnellement	INC	00
Compte si la condition est vraie, maintient l'état sinon	WAIT	01
Saute si la condition est vraie, compte sinon	JUMPT	10
Saute si la condition est fausse, compte sinon	JUMPF	11

Exercices série IV

IV.3 Adaptez le μprogramme de la diapositive 107 et le schéma de l'UC au μséquenceur développé dans l'exercice IV.2 .

IV.4 Dans l'UC de la diapositive 101, générez les sorties à l'aide d'un décodeur en lieu et place de la mémoire.

IV.5 Faites une description VHDL du μséquenceur de l'exercice IV.2 .

Contenu de la présentation

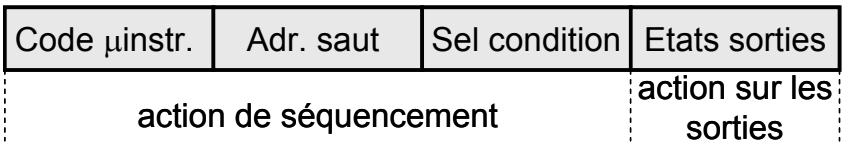
- Programmation horizontale !
- Optimisation de la mémoire de μprogramme
  - ✓Microprogrammation verticale
  - ✓Exemple : μséquenceurs à un seul opérande
  - ✓Limitation des adresses de saut
  - ✓Codage des plages
  - ✓Nanoprogrammation
  - ✓Codage des sorties

Microprogrammation horizontale

- Constat : toutes les μinstructions n'utilisent pas toutes les plages
- Existe-t-il une solution pour réduire la largeur de la mémoire de μprogramme ?

μinstructions horizontales

- μprogrammation horizontale



+ Plusieurs actions simultanées

(saut, sélection condition, activation sorties)

– Tous les champs ne sont pas toujours utilisés

Microprogrammation verticale

- Constat : peu de μinstructions horizontales utilisent toutes les plages
- En réduisant le nombre de plages des μinstructions on peut réduire la largeur de la mémoire de μprogramme
- Attribuer plusieurs rôles à une même plage :

μprogrammation verticale

μinstructions verticales

- μprogrammation verticale

+ μInstruction plus courte (largeur mémoire) \
– Plusieurs actions nécessitent plusieurs μinstr.

⚠ Il faudra mémoriser l 'état des sorties

⚠ L 'opérande a plusieurs fonctions

Exemple : μséquenceurs à un seul opérande

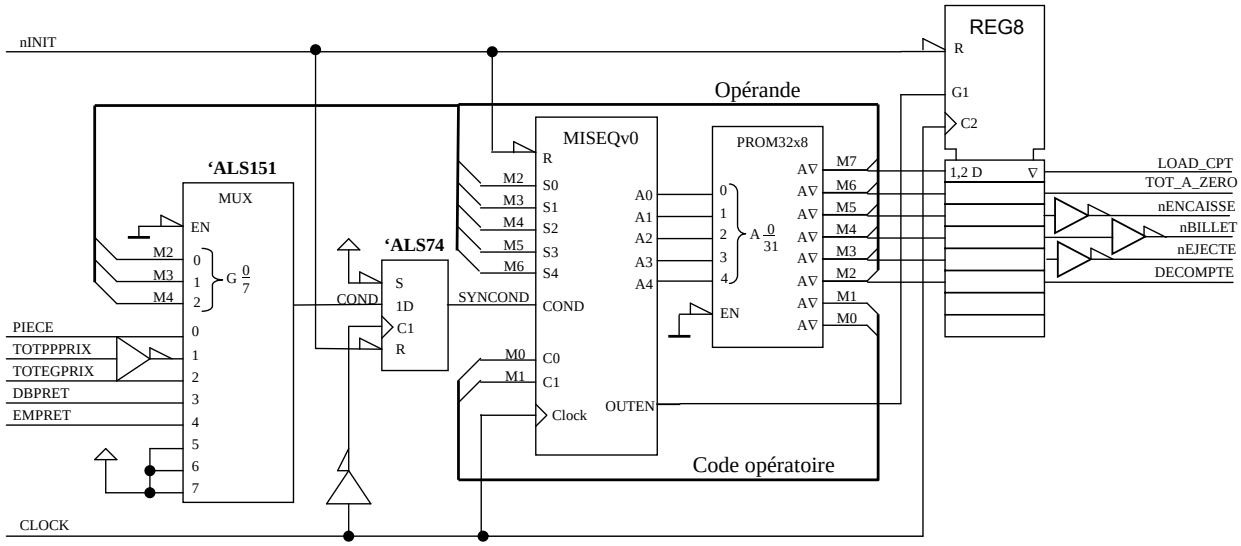
- 3 opérandes pour une seule plage :
  - ✓adresse de saut
  - ✓adresse de condition (éventuellement : polarité)
  - ✓état des sorties
- un seul opérande par μinstruction
  - ✓opérations de saut, de décision et d'affectation des sorties : mutuellement exclusives
  - ✓une seule de ces opérations par μinstruction
  - ✓entre 2 affectations, mémoriser les sorties

MISEQv0 : μséquenceur à un seul opérande

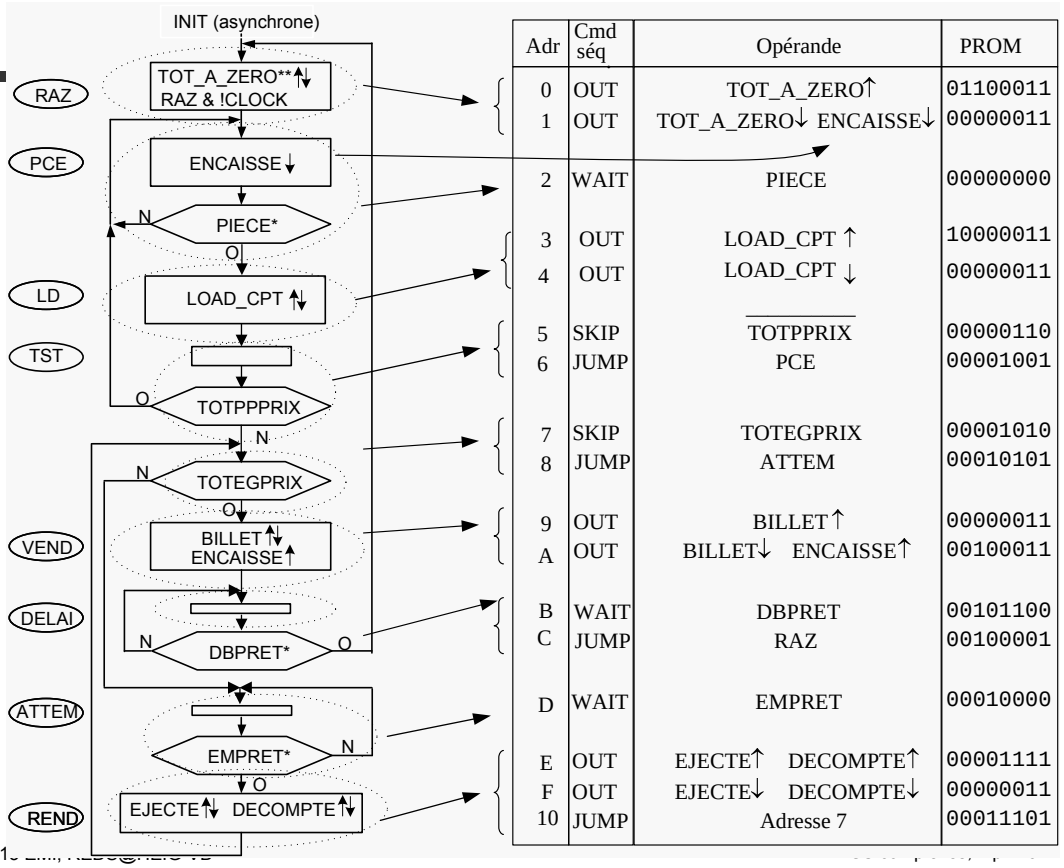
- Jeu de μinstructions MISEQV0 :

Action	Mnémo.	Code	Opérande
Attend condition μPC <= μPC+1 si condition vraie, μPC sinon	WAIT	00	Adresse de sélection de la condition
Saute Inconditionnellement μPC <= Adr. saut	JUMP	01	Adresse de saut
Saute μinstr. suivante si cond. vraie μPC <= μPC +2 si cond. vraie, μPC +1 sinon	SKIP	10	Adresse de sélection de la condition
Affecte sorties et continue μPC <= μPC+1	OUT	11	Désignation des sorties et des valeurs affectées

Schéma d'une UC avec MISEQv0



μprogramme, UC vendeur billets, MISEQv0



Adr	Cmd séq.	Opérande	PROM
0	OUT	TOT_A_ZERO↑	01100011
1	OUT	TOT_A_ZERO↓ ENCAISSE↓	00000011
2	WAIT	PIECE	00000000
3	OUT	LOAD_CPT ↑	10000011
4	OUT	LOAD_CPT ↓	00000011
5	SKIP	$\overline{\text{TOTPPRIX}}$	00000110
6	JUMP	PCE	00001001
7	SKIP	TOTEGPRIX	00001010
8	JUMP	ATTEM	00010101
9	OUT	BILLET↑	00000011
A	OUT	BILLET↓ ENCAISSE↑	00100011
B	WAIT	DBPRET	00101100
C	JUMP	RAZ	00100001
D	WAIT	EMPRET	00010000
E	OUT	EJECTE↑ DECOMPTE↑	00001111
F	OUT	EJECTE↓ DECOMPTE↓	00000011
10	JUMP	Adresse 7	00011101

Limitation des adresses de saut

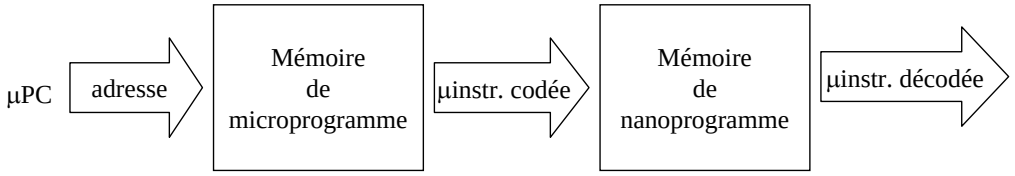
- Pour utiliser moins de bits dans les adresses de saut (opérande d'un saut)
  - ✓ne sauter que vers certaines adresses (par ex. seulement les adresses paires)
  - ✓utiliser l'écart entre l'adresse de départ et celle d'arrivée (adressage relatif), en le limitant
  - ✓séparer l'adresse complète en 2 composantes fournies par 2 μinstructions distinctes
  - ✓généralement, il faudra plus de μinstructions pour le même algorithme, donc plus de temps

Codage des plages

- Si nombre de valeurs différentes utiles que peut prendre une plage de N bits << 2**N
  - ✓coder ces valeurs sur N-X bits
  - ✓X bits de moins dans la largeur des μinstructions
  - ✓mais il faut ajouter un décodeur derrière la mémoire
  - ✓augmente le temps de propagation (diminue la fréquence d'horloge maximum)

Nanoprogrammation

- Nanoprogrammation = codage des μinstructions complètes



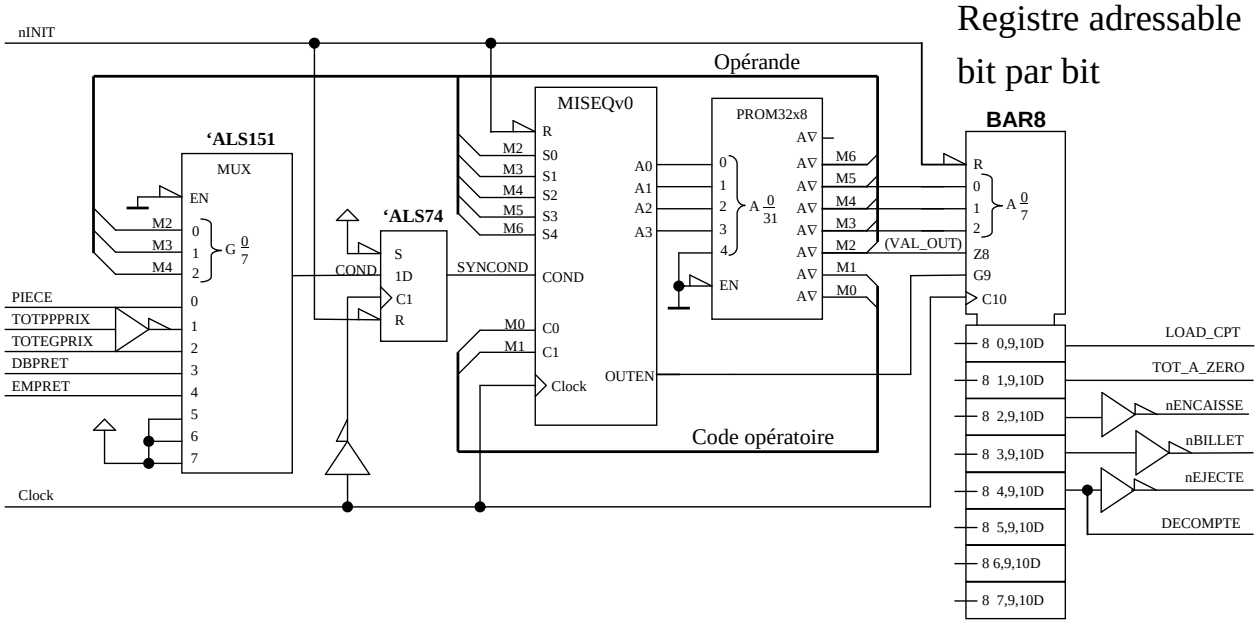
Codage des sorties

- A chaque période d'horloge, il n'y a qu'un petit nombre de sorties qui changent d'état
  - ✓n'indiquer que les sorties concernées (avec un code) et les valeurs qu'elles doivent prendre
  - ✓maintenir la valeur entre 2 changements, avec des bascules
  - ✓enregistrer les valeurs dans les bascules correspondant aux sorties concernées (démultiplexage)

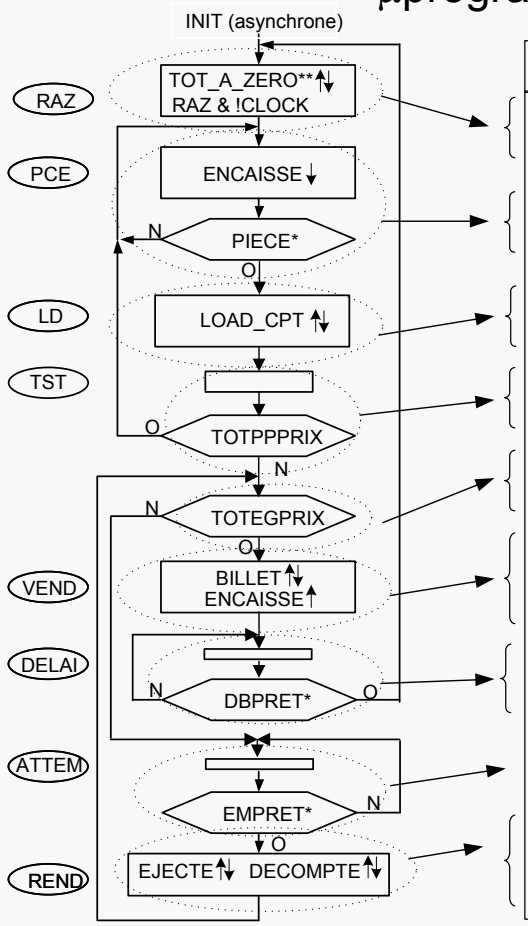
Exemple : affectation d'une seule sortie à la fois avec MISEQv0

- L'opérande de la μinstruction OUT peut être composé de
  - ✓1 bit pour la valeur à affecter
  - ✓n bits pour désigner la sortie concernée (code, adresse)
- Pour l'UC du vendeur de billets, il suffit de 2 bits pour désigner les 4 sorties distinctes (Ejecte = Decompte)

Schéma d'une UC avec MISEQv0 et des sorties codées bit par bit



μprogramme, UC vendeur billets diapo 121



Adr	Cmd séq.	Opérande	PROM
0	OUT	TOT_A_ZERO↑	11001111
1	OUT	TOT_A_ZERO↓	11001011
2	OUT	ENCAISSE↓	11010011
3	WAIT	PIECE	11100000
4	OUT	LOAD_CPT ↑	11000111
5	OUT	LOAD_CPT ↓	11000011
6	SKIP	$\overline{\text{TOTPPRIX}}$	11100110
7	JUMP	PCE + 1	10001101
8	SKIP	TOTEGPRIX	11101010
9	JUMP	ATTEM	10111101
A	OUT	BILLET↑	11011111
B	OUT	BILLET↓	11011011
C	OUT	ENCAISSE↑	11010111
D	WAIT	DBPRET	11101100
E	JUMP	ATTEM	10111001
F	WAIT	EMPRET	11110000
10	OUT	EJECTE↑ / DECOMPTE↑	11100111
11	OUT	EJECTE↓ / DECOMPTE↓	11100011
12	JUMP	Adresse 8	10100001

Exercices série V

V.1 Ajoutez un sélecteur de la polarité de la condition dans le schéma de la diapositive 118. Proposez un format pour les μinstructions WAIT et SKIP de MISEQv0, qui inclue le choix de la polarité.

V.2 Suite à de nouvelles spécifications, l'UC doit comporter 4 sorties supplémentaires. Quelles modifications faut-il apporter aux schémas des diapos 101, 118 et 125 ?

Exercices série V

V.3 Décrivez en VHDL un μséquenceur n bits ayant les mêmes spécifications que MISEQv0.

V.4 Décrivez en VHDL un registre adressable bit par bit similaire au BAR8, celui-ci est composé de bascules actives sur flanc.
Commencez par établir un schéma bloc.

Exercices série V

V.5 Décrivez en VHDL un μséquenceur ayant 7 bits d'adresse nommé MISEQv2 exécutant le jeu de μinstructions ci-dessous.

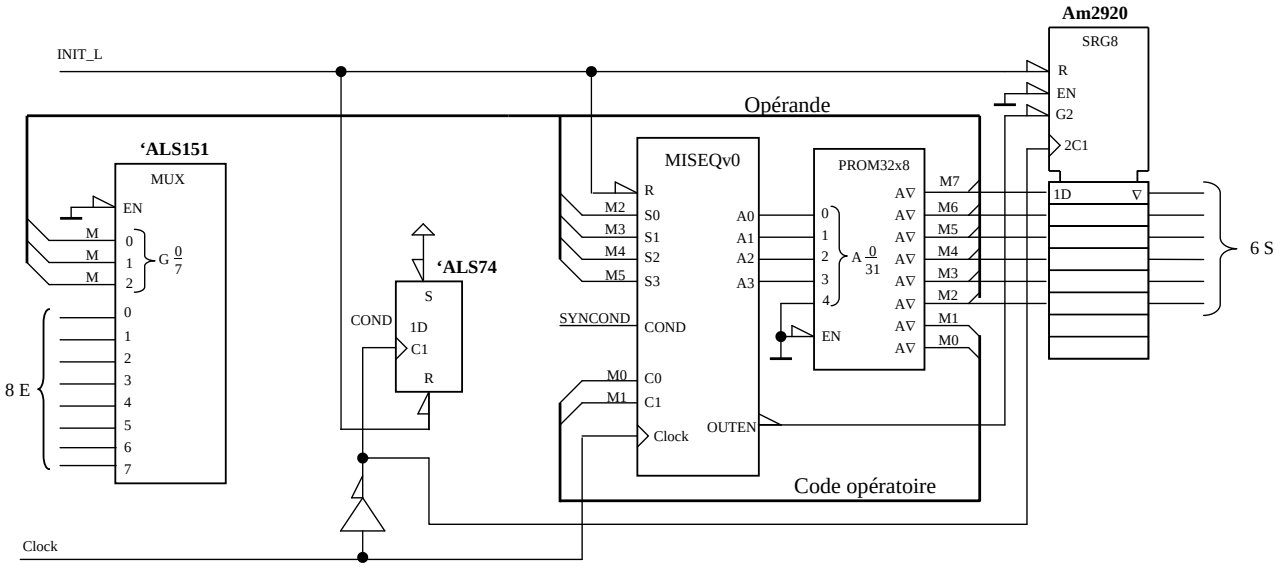
Action	Mnémo.	Code et format 8 b.	Opérandes
Mémorise la condition, inversée ou non μPC <= μPC+1	Mem	ccccci011	cccc = adresse de la condition i = inversion de la condition
Attend que la cond. mémorisée soit vraie, mémorise la condition à chaque clock (inv. ou non) μPC <= μPC+1 si cond. mém.vraie, μPC sinon	Wait	ccccci111	cccc = adresse de la condition i = inversion de la condition
Saute si la condition mémorisée est fausse, met à 0 la condition mémorisée (dans tous les cas) μPC <= Adr. si cond. fausse, μPC+1 sinon	Jumpf	aaaaaaa0	aaaaaaa = adresse de saut
Active la sortie /Outen de MISEQv2, met à 0 la condition mémorisée μPC <= μPC+1	Out	sssssv01 (par ex.)	sssss = sélection de la sortie v = valeur à affecter à la sortie

Exercices série V

V.6 Ecrivez le microprogramme correspondant à l'organigramme de la page 119 pour le MISEQv2 avec un registre de sortie BAR8

Solution ajout polarité au MISEQv0

Exe V.1



Dia volontairement laissé vide

Contenu de la présentation

- Sous-μprogrammes
  - ✓utilité
  - ✓définition et mécanisme
  - ✓notion de pile, définition et mécanisme
  - ✓exemple de μséquenceur avec sous-μprogrammes
- Interruptions
  - ✓utilité
  - ✓définition et mécanisme
  - ✓exemple de μséquenceur avec interruption

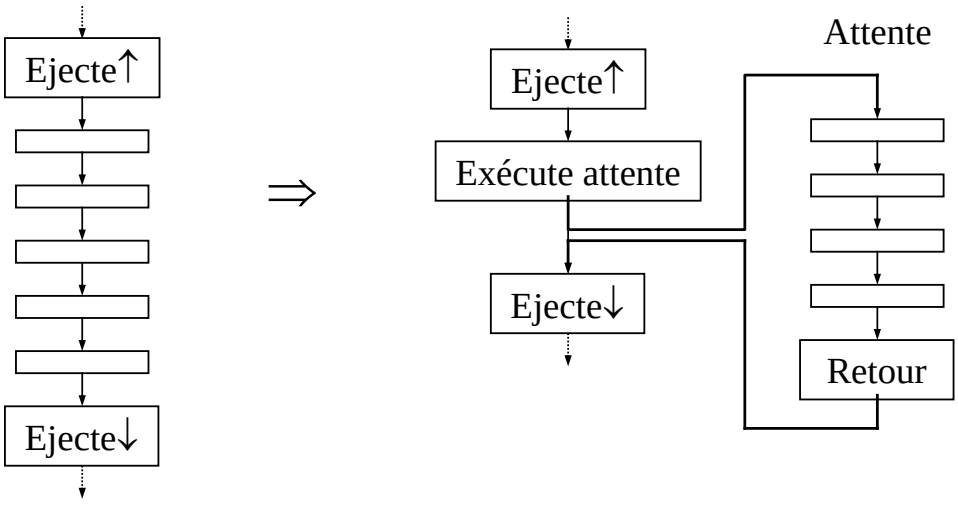
Pourquoi des sous-μprogrammes ?

- Constat : souvent, des séquences identiques (ou très semblables) d'opérations apparaissent à plusieurs reprises dans divers points d'un algorithme
- Par exemple : attente passive de 5 périodes d'horloge entre activation et désactivation d'une sortie, demandant à chaque fois 5 états (5 μinstructions)

Pourquoi des sous-μprogrammes ?

- La séquence réalisant cette attente peut être implémentée une seule fois et être exécutée à plusieurs reprises dans divers points d'un algorithme
  - ✓algorithme plus compact (μprogramme plus court)
  - ✓conception structurée, hiérarchique, réutilisable
- Il faut un mécanisme pour appeler l'exécution de cette séquence depuis n'importe où (Call) et retourner ensuite au point de départ (Return)

Exemple



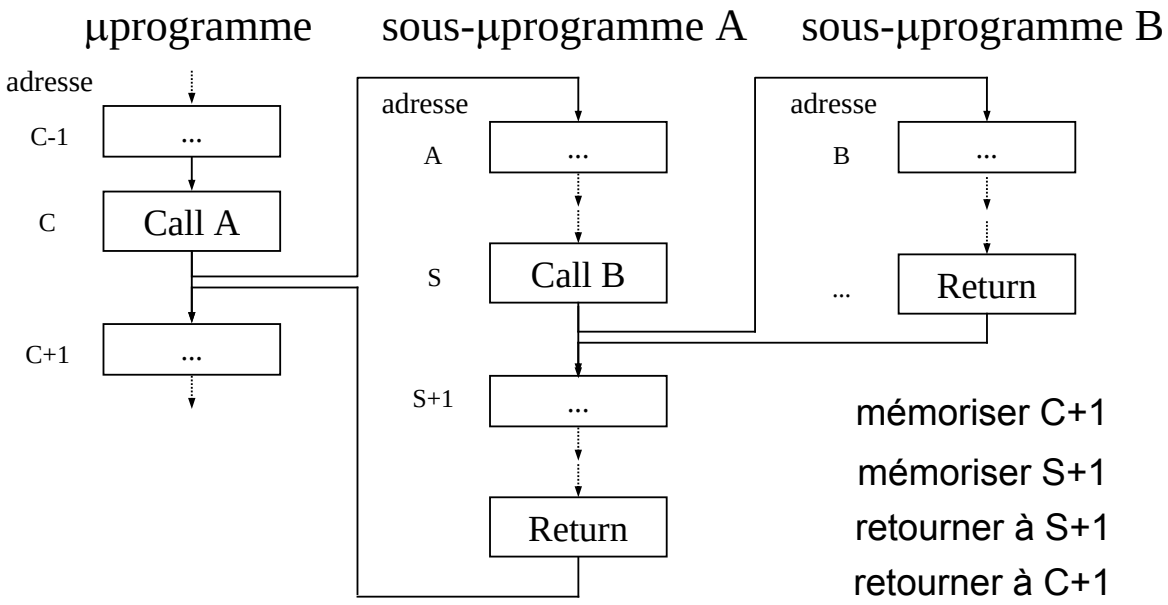
Sous-μprogrammes : définition et mécanisme

- Suite de μinstructions dont l'exécution est
  - ✓déclenchée à l'aide d'un saut spécial : une μinstruction "appel" (call, subroutine branch, branch and link)
  - ✓terminée par un retour à la μinstruction qui suit celle de l'appel (return)
- Exécution déclenchée par des appels situés à divers endroits de l'algorithme → l'adresse de retour doit être mémorisée lors de l'appel

Pourquoi une pile ?

- Un μprogramme peut appeler un sous-μprogramme A, qui appelle à son tour un sous-μprogramme B
  - ✓lors de l'appel de A, une 1re adresse de retour est mémorisée
  - ✓lors de l'appel de B, une 2ème adresse de retour est mémorisée
  - ✓à la fin du μprogramme B il faut retourner à l'adresse mémorisée lors de son appel (la 2ème)
  - ✓à la fin du μprogramme A il faut retourner à l'adresse mémorisée lors de son appel (la 1re)

Pourquoi une pile ?



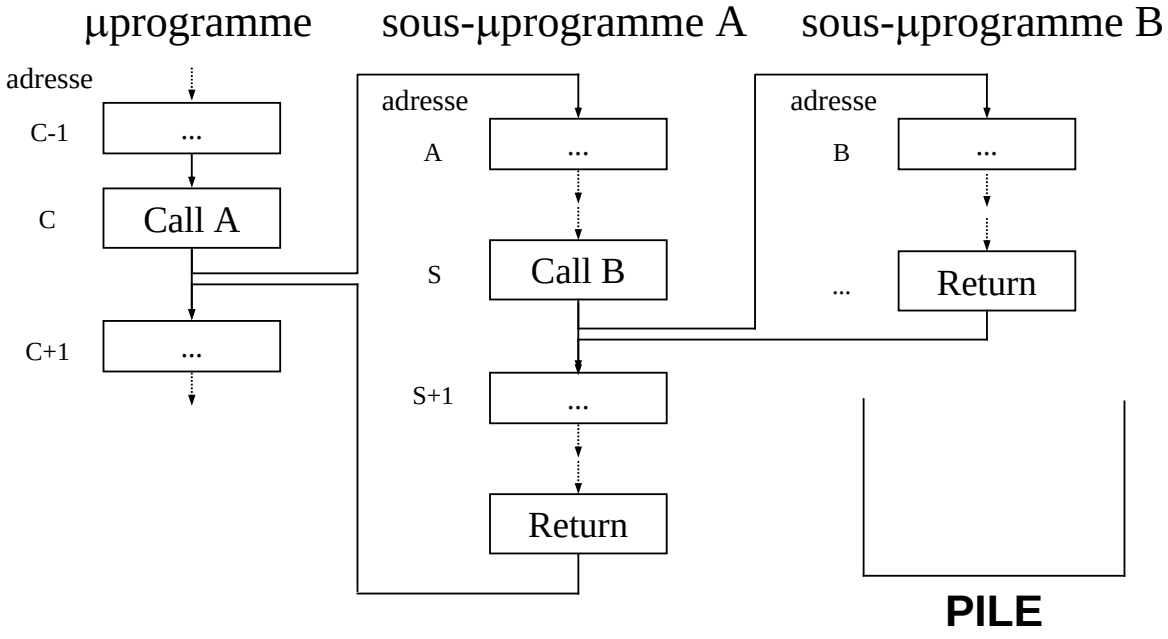
Pourquoi une pile ?

- Lors des sauts vers des sous-µprogrammes imbriqués, les adresses de retour sont mémorisées dans l'ordre des sauts
- Les retours utilisent les adresses mémorisées en ordre inverse : la dernière mémorisée est la première utilisée
- Les adresses de retour peuvent être
  - ✓empilées lors des sauts
  - ✓désempilées lors des retours

Pile : définition et mécanisme

- Pile (stack) : structure de données à accès séquentiel, où la dernière information écrite sera la première lue (LIFO, last in - first out)
- Accès par 2 opérations
  - ✓écriture d'une donnée sur le haut de la pile (Push) → augmentation de la pile
  - ✓lecture et suppression de la donnée se trouvant en haut de la pile (Pop) → diminution de la pile

Fonctionnement d 'une pile ?



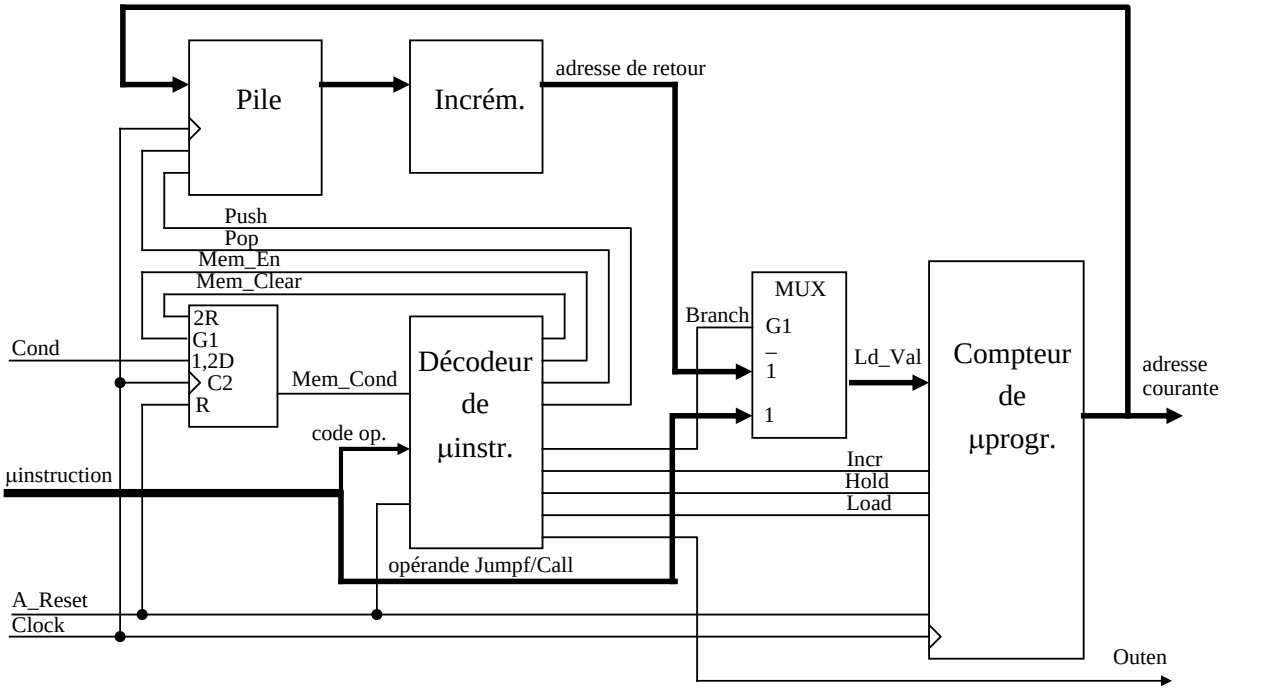
Exemple : MISEQv2 avec Call et Ret → MISEQv3

- Ajoutons 2 μinstructions à MISEQv2
- Call : provoque le saut à l'adresse fournie (opérande), sauve l'adresse courante sur la pile; ne modifie pas la condition mémorisée
- Ret : charge le compteur de μprogramme avec l'adresse se trouvant sur le haut de la pile + 1; ne modifie pas la condition mémorisée; active Outen; l'opérande est utilisé pour affecter les sorties

Exemple : MISEQv2 avec Call et Ret → MISEQv3

- Matériel à ajouter à MISEQv2
  - ✓pile
  - ✓incrémenteur (+1 à la sortie de la pile)
  - ✓MUX pour choisir la source de l'adresse chargée dans le compteur de μprogramme : pile à travers incrémenteur pour Ret, opérande de la μinstruction pour Call et Jumpf

Schéma bloc de MISEQv3



Jeu d'instructions du MISEQv3 …

séquenceur avec 5 bits d'adresse

Action	Mnémo.	Code et format 8 b.	Opérandes
Mémorise la condition, inversée ou non µPC <= µPC+1	Mem	cccci011	cccc = adresse de la condition i = inversion de la condition
Attend que la cond. mémorisée soit vraie, mémorise la condition à chaque clock (inv. ou non) µPC <= µPC+1 si cond. mém.vraie, µPC sinon	Wait	cccci001	cccc = adresse de la condition i = inversion de la condition
Saute si la condition mémorisée est fausse, met à 0 la condition mémorisée (dans tous les cas) µPC <= Adr. si cond. fausse, µPC+1 sinon	Jumpf	aaaaa000	aaaaa = adresse de saut
Saute à l'adresse spécifiée et sauve l'adresse actuelle sur la pile. La condition mémorisée est maintenue µPC <= Adr. ; Pile <= µPC (valeur actuelle)	Call	aaaaa100	aaaaa = adresse de saut

… jeu d'instructions du MISEQv3

séquenceur avec 5 bits d'adresse

Action	Mnémo.	Code et format 8 b.	Opérandes
Active la sortie /Outen de MISEQv2, met à 0 la condition mémorisée µPC <= µPC+1	Out	ssssv010 (par ex.)	ssss = sélection de la sortie v = valeur à affecter à la sortie
Charge µPC avec l'adresse fournie par la pile+1. Fonctionne comme Out sauf que la condition mémorisée n'est pas modifiée µPC <= AdressePile +1	Ret	ssssv110 (par ex.)	ssss = sélection de la sortie v = valeur à affecter à la sortie
Idem que Ret, mais l'adresse prise sur la pile n'est pas incrémentée µPC <= AdressePile	IRet	ssssv111 (par ex.)	ssss = sélection de la sortie v = valeur à affecter à la sortie
Fonctionnement non défini	NUsed	-----101	
Une interruption est obtenue en forçant un Call depuis l'extérieur. Dans ce cas, le sous-microprogramme doit se terminer par un IRet Le preset asynchrone met le compteur de microprogramme a 0x1F, la condition mémorisée a 1, empêche l'activation de Outen			

Pourquoi des interruptions ?

- Certains événements peuvent exiger une réaction de l'UC *quasi* immédiate
  - ✓chute de tension
  - ✓erreur de fonctionnement
  - ✓détection d'un danger
  - ✓synchronisation avec une cadence d'échantillonnage
- Comment faire pour que l'UC réagisse au plus vite ? Interrompre le travail courant et sauter à l'exécution du travail urgent

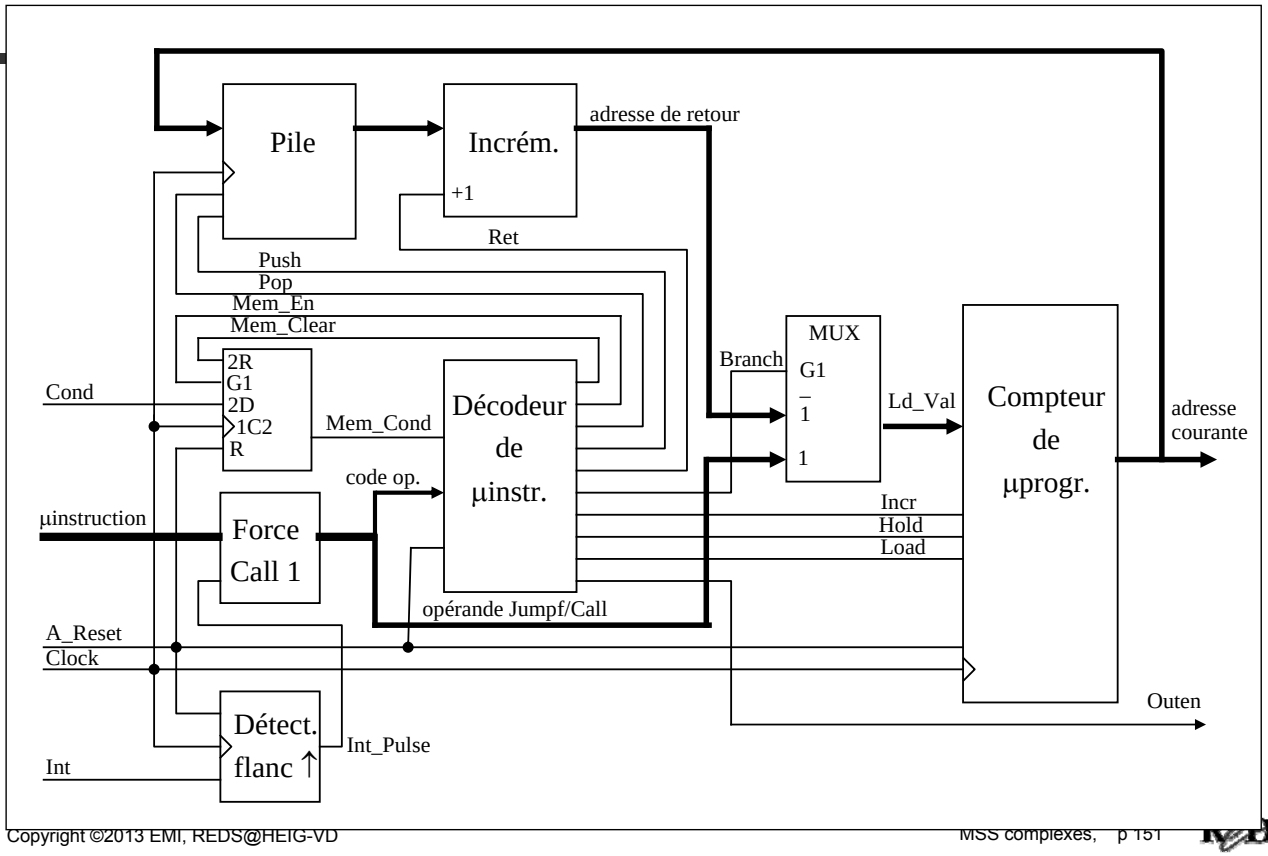
Interruption : définition et mécanisme

- Interruption : arrêt "immédiat" du travail A en cours, déclenché par un événement extérieur (pas par une μinstruction), afin d'exécuter un autre travail B
- A la fin de l'exécution du travail B, retour à l'exécution du travail A, là où elle avait été interrompue
- Mécanisme similaire à Call et Ret, mais saut vers sous-μprogramme déclenché par un signal d'entrée de l'UC

Exemple : MISEQv3 avec interruption → MISEQv4

- Idée : dès qu'une interruption est déclenchée, on force le μséquenceur à effectuer un Call à l'adresse du sous-μprogramme souhaité
  - ✓cette adresse peut être fixe (plus simple)
  - ✓commander le forçage durant une période d'horloge lorsqu'une interruption est déclenchée
  - ✓détecter un flanc du signal de déclenchement
  - ✓l'adresse de retour est celle sauvée sur la pile, sans incrémentation → nouvelle μinstruction de retour : Iret (idem Ret mais sans incrémentation)

Schéma bloc de MISEQv4



Exercices série VI

VI.1 Faites une description VHDL des blocs suivants de MISEQv4 (diapo 151)

- ✓la pile, en lui attribuant 4 niveaux
- ✓le décodeur de µinstructions
- ✓le bloc "Force Call 1"

VI.2 Avec MISEQv3, est-il possible de passer un paramètre booléen d'un µprogramme à un sous-µprogramme, et dans le sens inverse ? Si oui, de quelle(s) façon(s) ?

Exercices série VI

VI.3 MISEQv4 ne sauvegarde pas la condition mémorisée lorsque survient une interruption. Cela produira des erreurs lorsqu'une interruption survient après l'exécution d'une µinstruction Mem ou pendant celle de Wait

- ✓expliquez pourquoi
- ✓décrivez les modifications à apporter à MISEQv4 de façon à sauvegarder Mem_Cond lors d'une interruption et la récupérer lors d'un Iret
- ✓y a-t-il une solution du côté logiciel ?

Exercices série VI

VI.4 La prise en compte de la valeur des pièces est faite dans l'UC, qui commande le décom-ptage du solde effectué dans l'UT (par ex. : on décompte 10 fois pour une pièce de 1 Fr.). La fréquence d'horloge a été fixée à 100 Hz.

- ✓écrivez un sous-µprogramme pour générer une attente pouvant varier entre 1 et 10 périodes
- ✓écrivez un µprogramme complet de l'UC qui utilise ce sous-µprogramme

Exercices série VI

VI.5 L'arrivée d'une nouvelle pièce déclenche une interruption dans une UC basée sur MISEQv4 (même UT que pour l'exercice précédent)

- ✓Ecrivez, un sous-µprogramme d'interruption qui traite la valeur d'une pièce (provoque le décomptage adéquat du solde)
- ✓Ecrivez le µprogramme complet de l'UC
- ✓Dans certaines parties de ce µprogramme, il n'est pas possible de traiter une pièce. Comment l'éviter?
- ✓Comment éviter une interruption?

Dia volontairement laissé vide

Contenu de la présentation

- Unité de traitement universelle
  - ✓motivation
  - ✓opérations élémentaires
  - ✓indicateurs ("flags")
  - ✓unité arithmétique et logique
  - ✓registre(s) accumulateur(s)
  - ✓aiguillage des opérandes
- UT universelle pour le vendeur de billets
- UC µprogrammée pour cette UT universelle

Pourquoi une UT "universelle" ?

- Dans une UT spécialisée
  - ✓la quantité de matériel est proportionnelle à la complexité du traitement à effectuer
  - ✓le fonctionnement est déterminé par le choix des fonctions (circuits) et le câblage
  - ✓les modifications sont difficiles à réaliser
  - ✓l'adaptation à d'autres applications est irréaliste

Pourquoi une UT "universelle" ?

- Objectifs : créer une UT
  - ✓utilisable pour de nombreuses applications
  - ✓avec des modifications minimes et faciles à réaliser
  - ✓pouvant donc être standardisée
  - ✓modulaire (puissance de traitement extensible)
- Solution : créer une UT pouvant
  - ✓effectuer toutes les opérations élémentaires
  - ✓conserver les résultats intermédiaires

Opérations élémentaires

- Toute fonction combinatoire peut être décomposée en sommes de produits, donc
  - ✓en fonctions élémentaires ET, OU et inversion
  - ✓se succédant dans un certain ordre (plusieurs ordres possibles)
  - ✓cet ordre pouvant être défini par un schéma, mais aussi par une séquence (graphe d'états ou organigramme)

Opérations élémentaires

- Les opérations arithmétiques sont très utilisées
  - ✓UT universelle plus performante avec opérations élémentaires d'addition et soustraction (câblées)
- Autres opérations élémentaires utiles :
  - ✓OU exclusif
  - ✓masquage (A • /B)
  - ✓décalages et rotations
  - ✓incrémentation et décrémentation
  - ✓complément à 2
  - ✓comparaison

Indicateurs ("flags")

- Souvent, l'UC effectue des tests basés sur des soustractions et des masquages
- Indicateurs (appelés aussi bits d'état, signaux d'état, "flags") : caractéristiques d'un résultat
- Les indicateurs ci-dessous facilitent les tests
  - ✓résultat nul : A-B=0 $\rightarrow$ A=B
  - ✓résultat négatif : A-B<0 $\rightarrow$ A<B (avec signe)
  - ✓report : A-B et Cy_o=1 $\rightarrow$ A$\geq$B (sans signe)
  - ✓dépassement de capacité (overflow) : erreur

Unité arithmétique et logique

- Bloc logique exécutant un jeu d'opérations élémentaires arithmétiques et logiques, une seule à la fois
- Sigle habituel : ALU, pour "Arithmetic & Logic Unit"

Exemple : IC standard 'F382

- Entrées : opérandes A et B (4 bits), report, code de sélection de l'opération
- Jeu d'opérations (code de sel. : opération) :
  - ✓0 : Clear
  - ✓ 1 : B Minus A
  - ✓2 : A Minus B
  - ✓ 3 : A + B (addition)
  - ✓4 : A ⊕ B
  - ✓ 5 : A # B
  - ✓6 : A • B
  - ✓ 7 : Preset
- Sorties : résultat (4 bits), report, débordement
- Symbole CEI : insuffisant

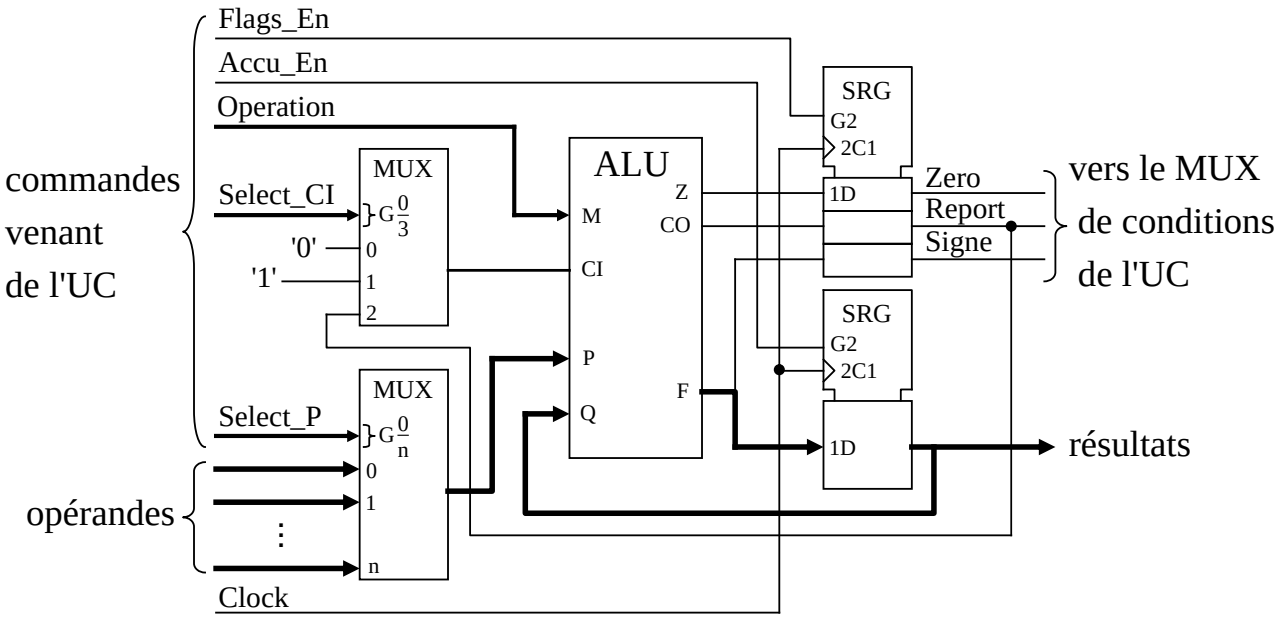
Registre(s) accumulateur(s)

- La décomposition de fonctions complexes en fonctions élémentaires génère des résultats intermédiaires qu'il faut mémoriser
- Un résultat intermédiaire est utilisé comme opérande dans une opération suivante
- Registre accumulateur : registre pouvant
  - ✓fournir un des opérandes d'entrée de l'ALU
  - ✓et recevoir le résultat généré
- Calculette 4 opérations : 1 seul accumulateur

Aiguillage de opérandes

- Les données à traiter doivent être acheminées vers l'une des entrées de l'ALU : multiplexage si plus d'une source par entrée
- Au moins une des entrées d'opérande doit pouvoir recevoir des données venant de l'extérieur de l'UT
- Le report d'entrée dépend de l'opération à effectuer, et du report généré par une opération précédente (add / sub "en tranches")

Schéma bloc d'une UT "universelle" simple



UT "universelle" pour le vendeur de billets, avec un 'F382

- Résultat intermédiaire à conserver : le total
- Opérations souhaitées :
  - ✓résultat ⇐ 0 : pour mettre total à 0
  - ✓addition 4 bits : total + valeur pièce
  - ✓comparaisons : total < prix et total = prix
  - ✓soustraction de 1 (décrémentation) : total - 1
- Pas de comparaison dans le 'F382, mais
  - ✓A<B lorsque A-B<0
  - ✓A=B lorsque A-B=0

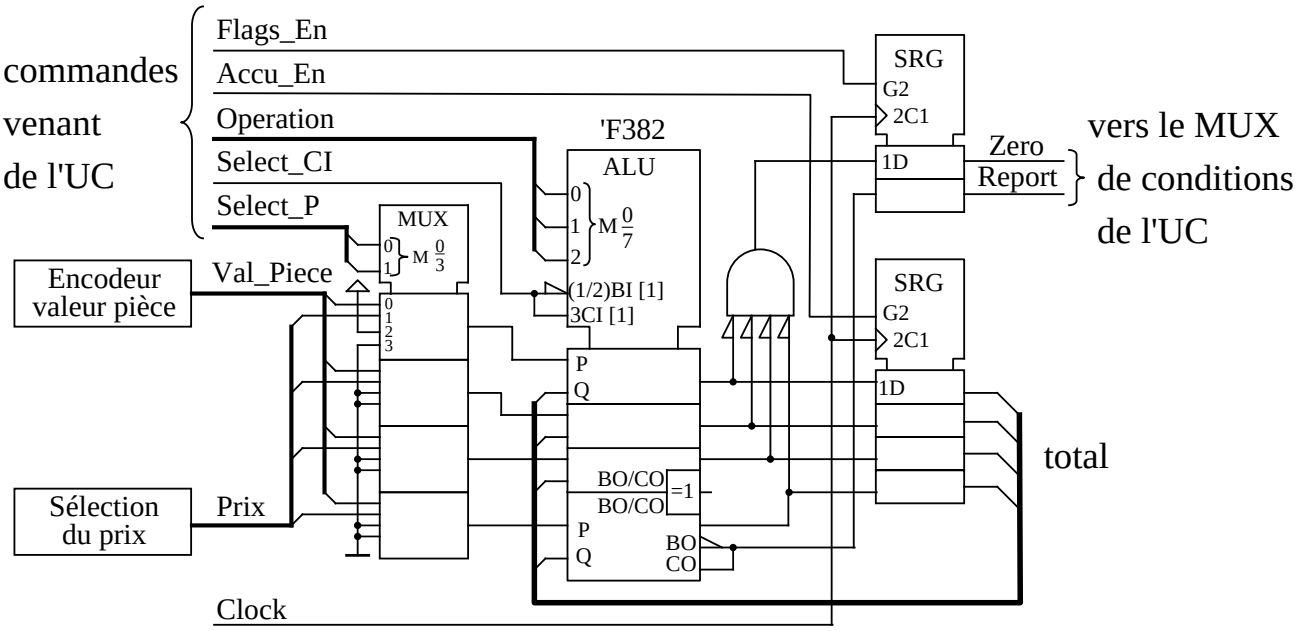
UT "universelle" pour le vendeur de billets, avec un 'F382

- Un seul accumulateur suffit (pour le total)
- Les autres opérandes seront multiplexés sur la 2ème entrée du 'F382 :
  - ✓valeur de la pièce
  - ✓prix
  - ✓1 (ou 0)
- Report d'entrée du 'F382
  - ✓forcé à 0 pour une addition
  - ✓forcé à 1 pour une soustraction

UT "universelle" pour le vendeur de billets, avec un 'F382

- Le report de sortie du 'F382 sera utilisé pour déterminer le signe du résultat de l'opération (calculs effectués sur 4 bits)
- Une détection de résultat nul est nécessaire pour les test d'égalité
- Report et détection de résultat nul doivent être mémorisés lorsqu'ils sont obtenus, pour être testés plus tard

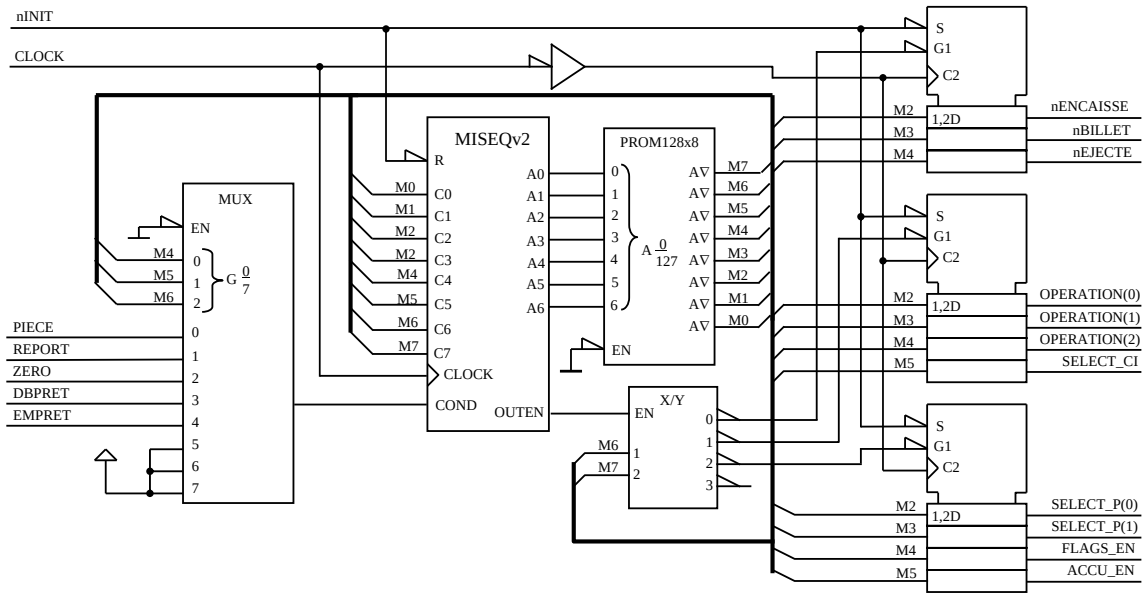
UT "universelle" pour le vendeur de billets, avec un 'F382



UC avec MISEQv2 pour l'UT universelle du vendeur de billets

- Il faut 8 sorties pour commander l'UT
  - ✓3 pour l'opération
  - ✓2 pour l'opérande
  - ✓1 pour le report d'entrée
  - ✓2 pour activer les mémorisations (accu et flags)
- Une PROM de 8 bits de large permet d'affecter les sorties par groupes de 4
- Format microinstruction OUT : ssvvvv01
  - ✓ss = sélection du groupe de 4 sorties
  - ✓vvvv = valeur affectée au groupe de 4 sorties

UC avec MISEQv2 pour l'UT universelle du vendeur de billets



Exercices VII

VII.1 Dessinez un organigramme détaillé pour un vendeur de billets réalisé avec l'UT de la diapo 164

VII.2 Traduisez cet organigramme en un µprogramme pour l'UC de la diapo 166