Unités : Systèmes logiques 2 (SysLog2)
Conception de systèmes numériques (CSN)

MSS complexes

MSS: Machines séquentielles synchrones

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Etienne Messerli
Institut REDS, HEIG-VD

Avril 2023

Contenu de la présentation

- Complexe ?
  => on fractionne !
- Fractionnement hiérarchique
- Ebauche de méthode de travail

- Documentation:
  - Electronique Numérique, 4ème tome
    Systèmes séquentiels avancés, MSS complexes

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 : systèmes avec fonctions standards telles que les registres et les compteurs
  - spécification via une table des fonctions synchrones

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
- Ne permet pas l'utilisation de fonctions standards sur les données

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

… puis développer nos aptitudes par des exercices

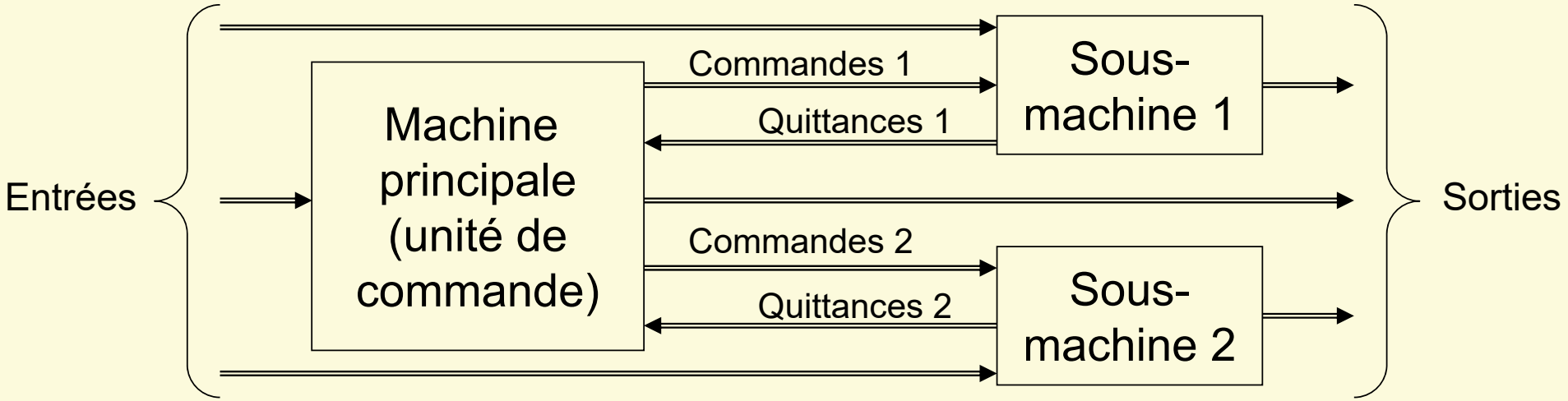
Trop complexe? Fractionnez-le!

- 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

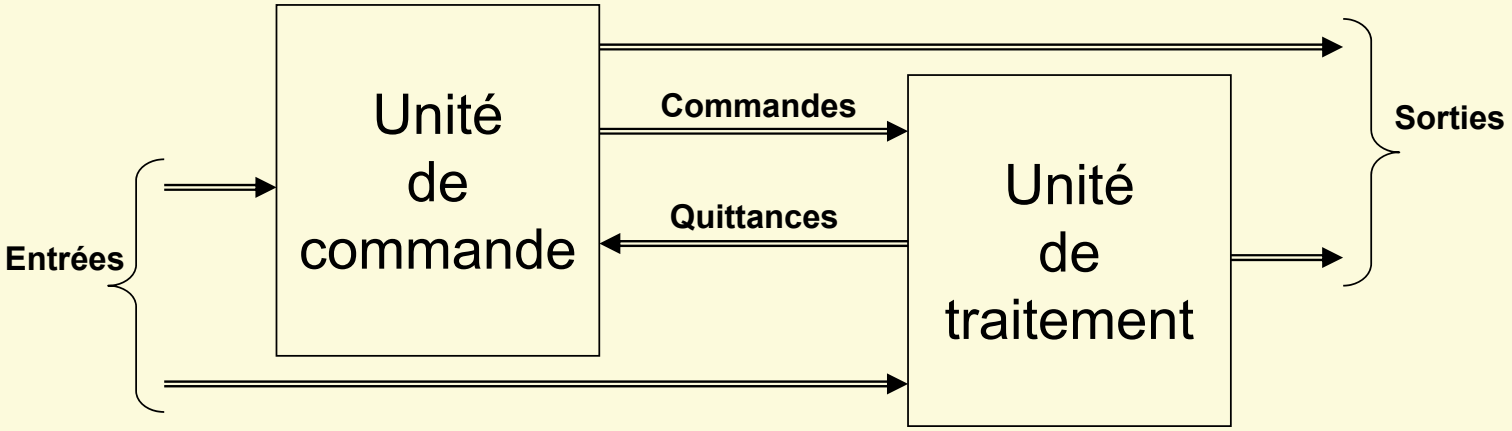
- Une tête qui commande (machine principale)
  - des membres qui exécutent (sous-machines)
  - et fournissent des quittances



Découpage hiérarchique

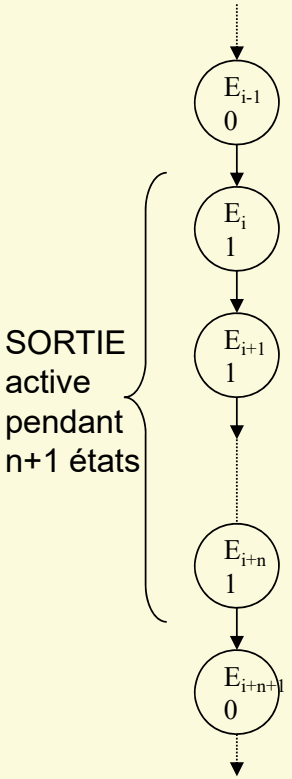
- Si encore trop complexe, chaque sous-machine peut elle-même être découpée hiérarchiquement
  - C'est le cas par exemple d'un ordinateur !
- Lors d'une décomposition, souvent, 2 niveaux suffisent :
  - machine principale (unité de commande, niveau supérieur)
  - les sous-machines (exécute, niveau inférieur)
- Les sous-machines peuvent être regroupées en un seul bloc de niveau inférieur

Partition à 2 niveaux hiérarchiques

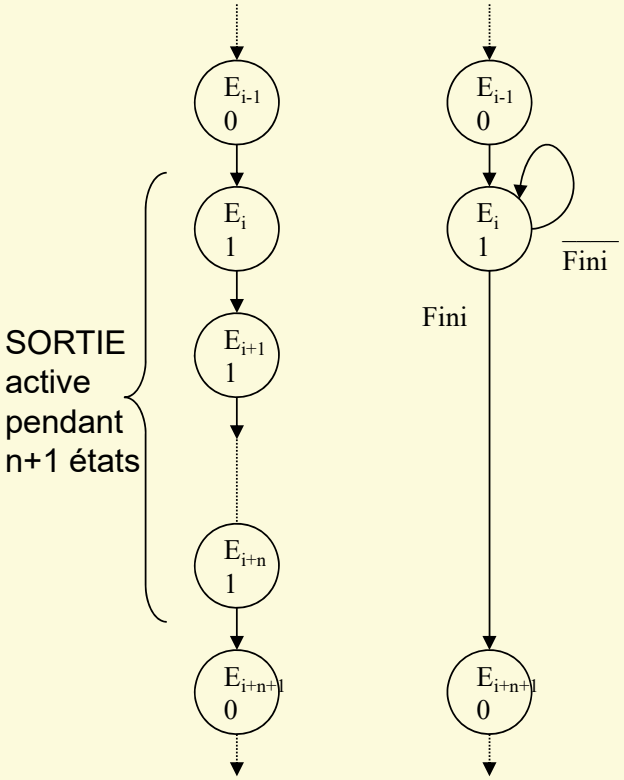


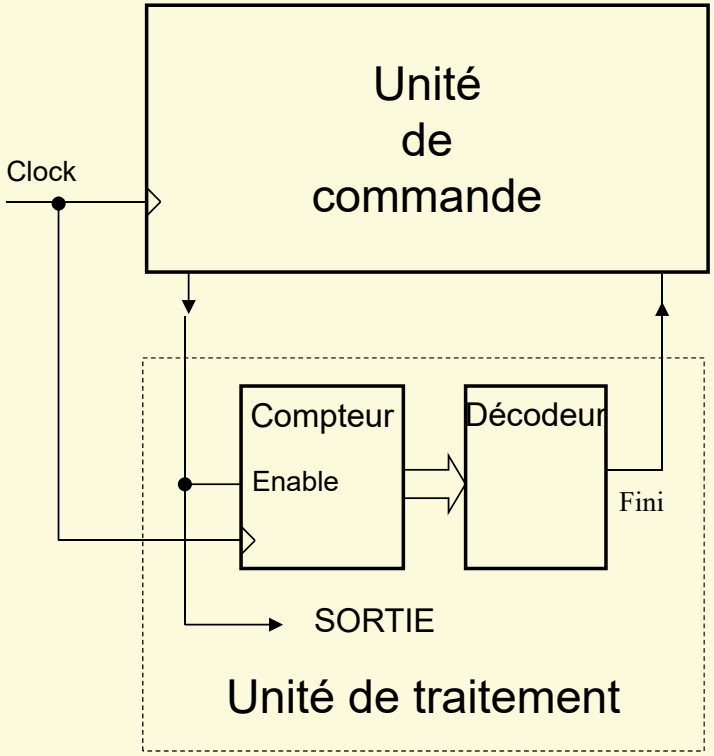
- Machine principale : unité de commande (UC),
  unité de séquencement, séquenceur, "control unit", "sequencer"
- Sous-machines : unité de traitement (UT),
  unité d'exécution, "execution unit", "data unit"

Identification de sous-machines



Identification de sous-machines





Méthodologie MSS complexes

- Approche Top-Down
- Description d'un algorithme global
- Séquence d'opérations (au lieu d'états)
- Identification des sous-machines possibles
  - identifier des fonctions standards
- Objectif globale de la méthode:
  - Maitriser complexité par une décomposition hiérarchique

Structures des MSS complexes

- Structures d'une unité de commande
  - UC câblée
  - UC micro-programmée (actuel µC)
- Structures d'une unité de traitement
  - UT spécialisée
  - UT universelle (RALU)
- Combinaisons UC – UT
  - 4 architectures possibles

Unité de commande

Fonctionnalité :

- Séquencement des opérations exécutées par l'unité de traitement
- Peut effectuer une partie du traitement
- C'est une machine séquentielle

Architecture :

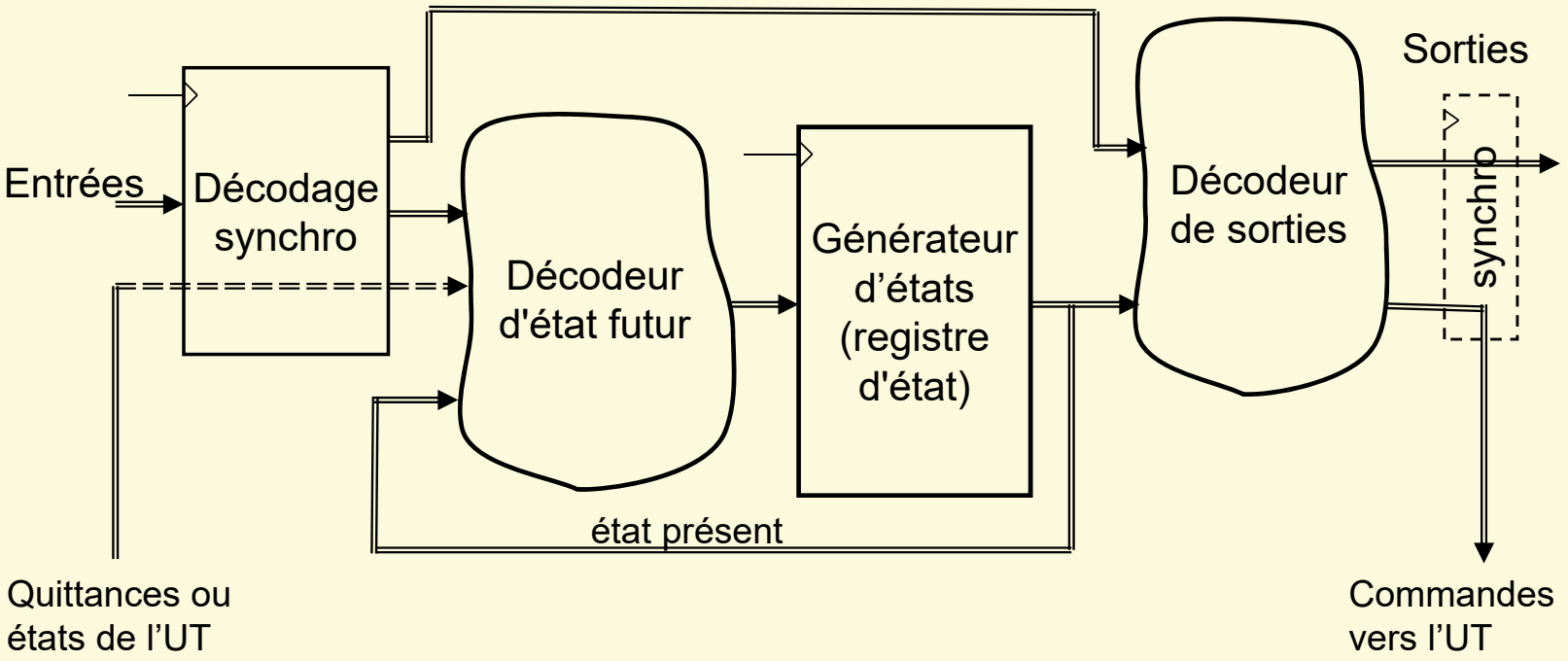
- Simple : UC câblée (graphe, description VHDL)
- Si trop complexe : séqu. opérations => c'est un programme
  - † *UC micro-programmée (séquenceur + mémoire)*
  - actuel : microcontrôleurs (µC + prog C)

UC câblée (graphe d'états)

Architecture câblée :

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

UC câblée : schéma bloc

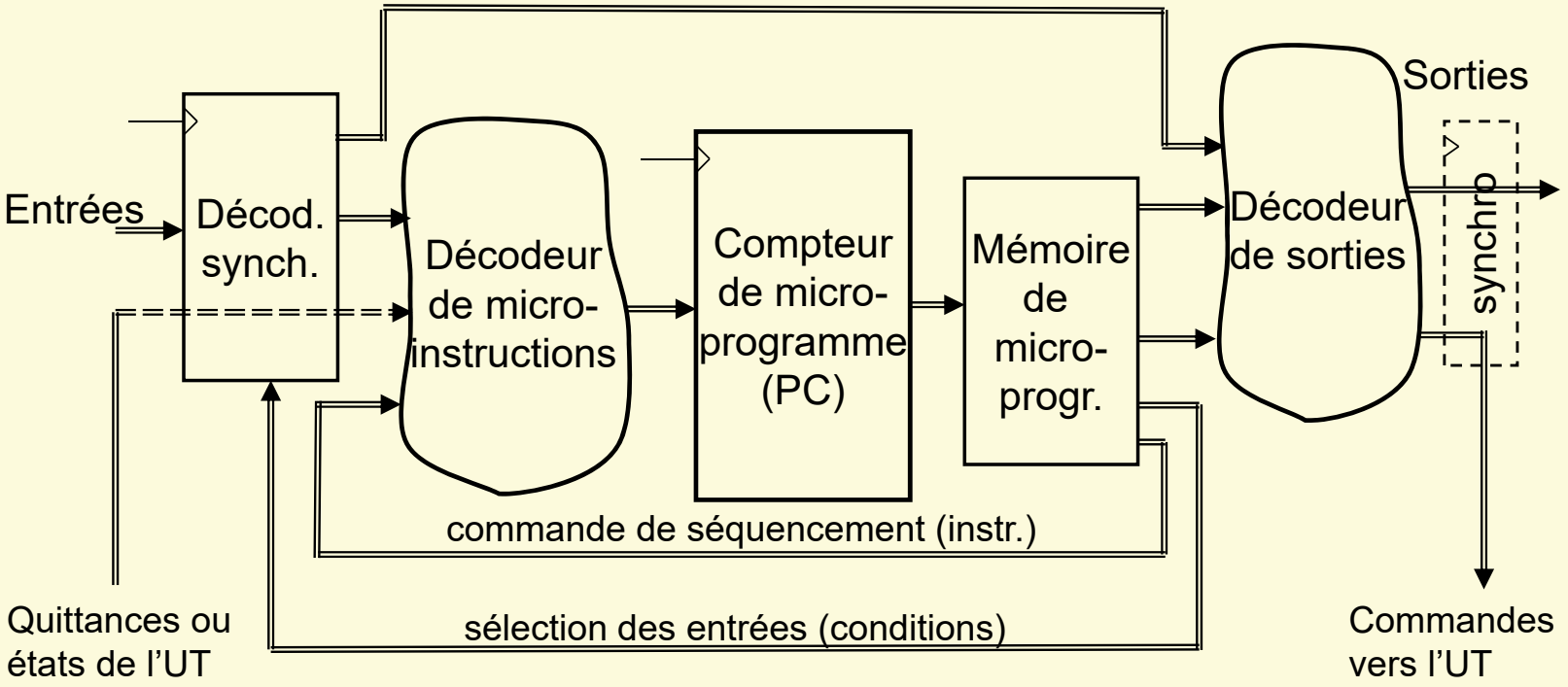


UC (micro)programmée

Architecture (micro) programmée :

- actuellement utilisation de microcontrôleur
- Comportement déterminé par un programme
- 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.

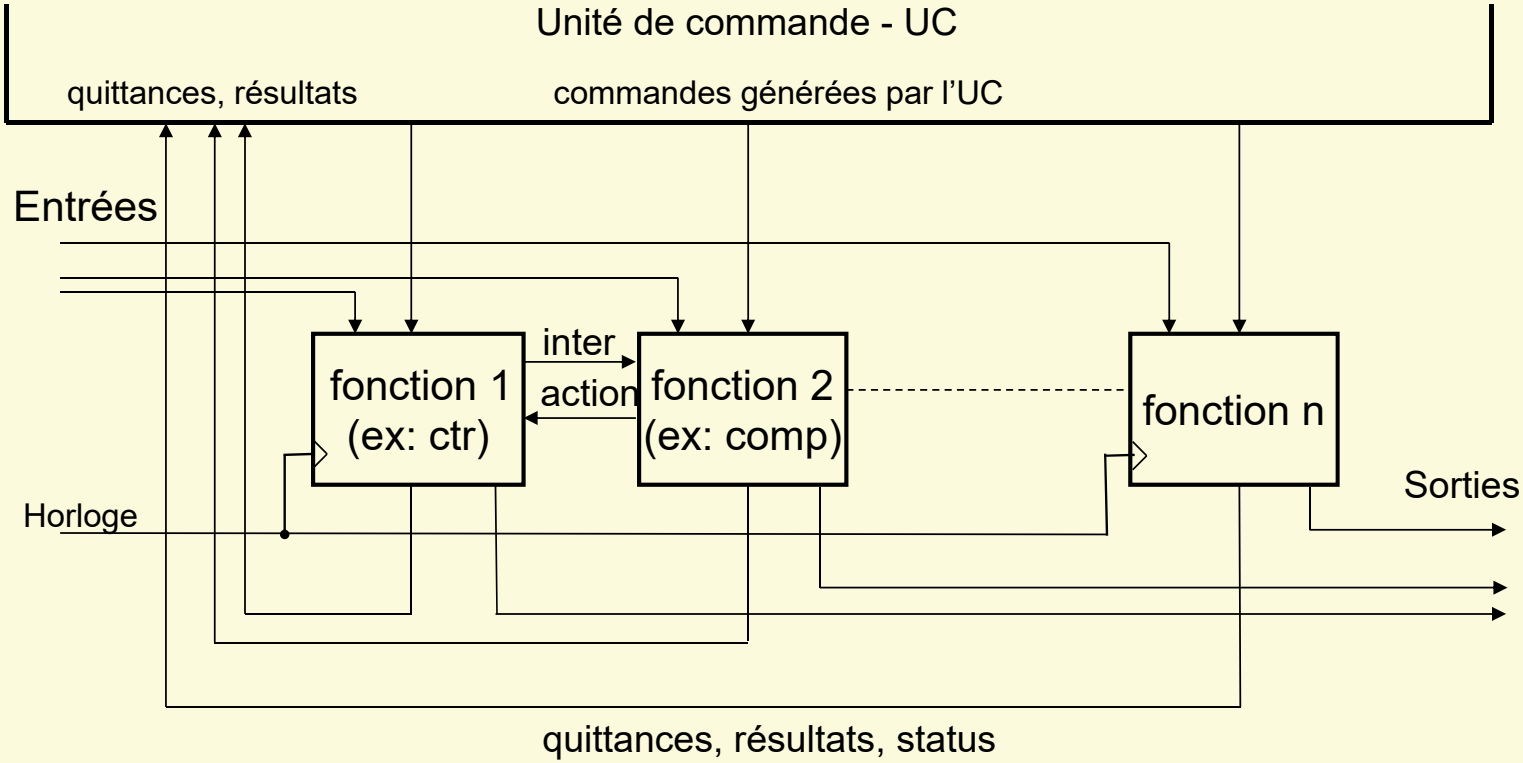
  ➔ UT spécialisée
- Chaque fonction est décomposée en une série de fonctions élémentaires standardisées (et, ou, inversion, addition, mémorisation)

  ➔ 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 maximales
  - volume/ coûts/ consommation : proportionnel à la complexité
  - modification => modification de descriptions VHDL
    nécessite: vérification du timing et une adaptation du banc de test

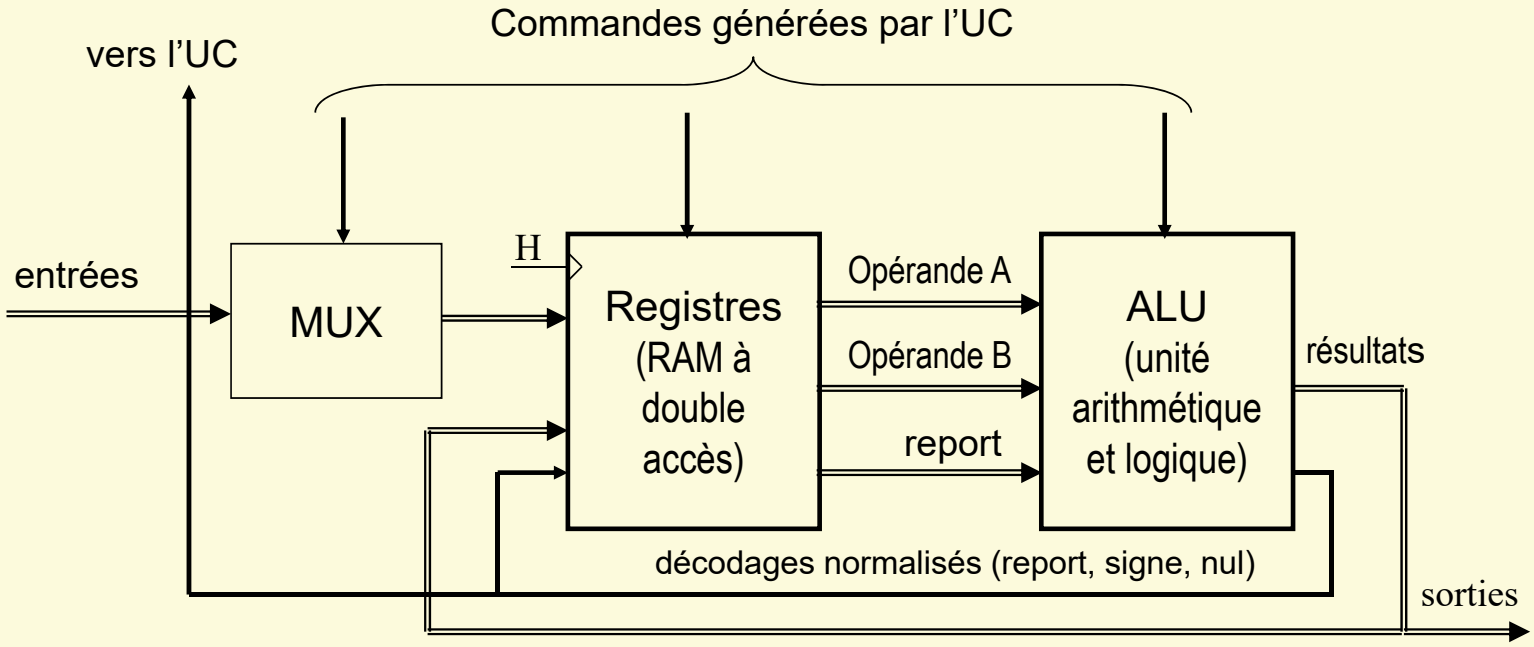
Schéma bloc d'une UT spécialisée



UT universelle : principe

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

Schéma bloc d'une UT universelle

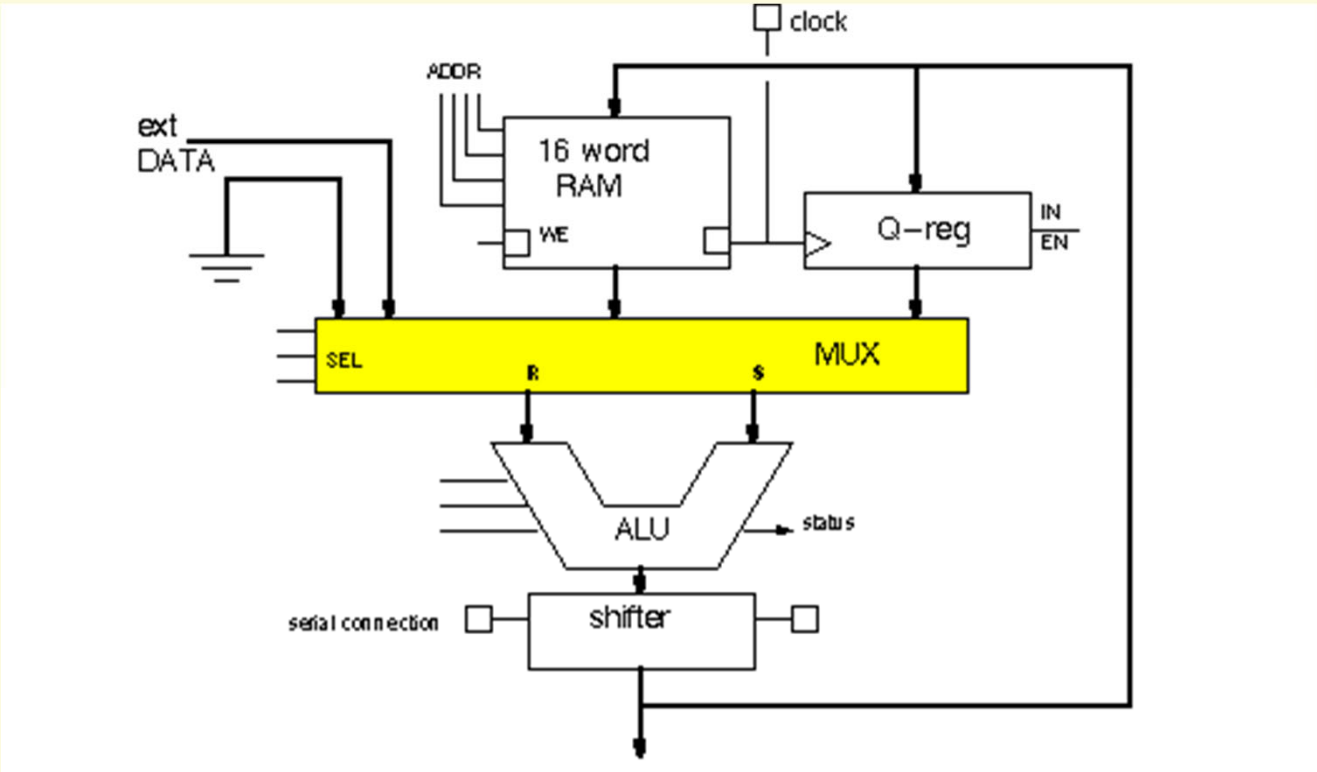


UT universelle : caractéristiques

- 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/ consommation : augmentation par palier
  - → changement d'application :
    - pas de modifications du matériel (ALU)
    - reporté sur l'UC (modification de la séquence/programme)

Schéma bloc d'une UT universelle

- Schéma de principe d'une RALU d'un processeur:



Combinaisons UC - UT

Il y a 4 combinaisons possible :

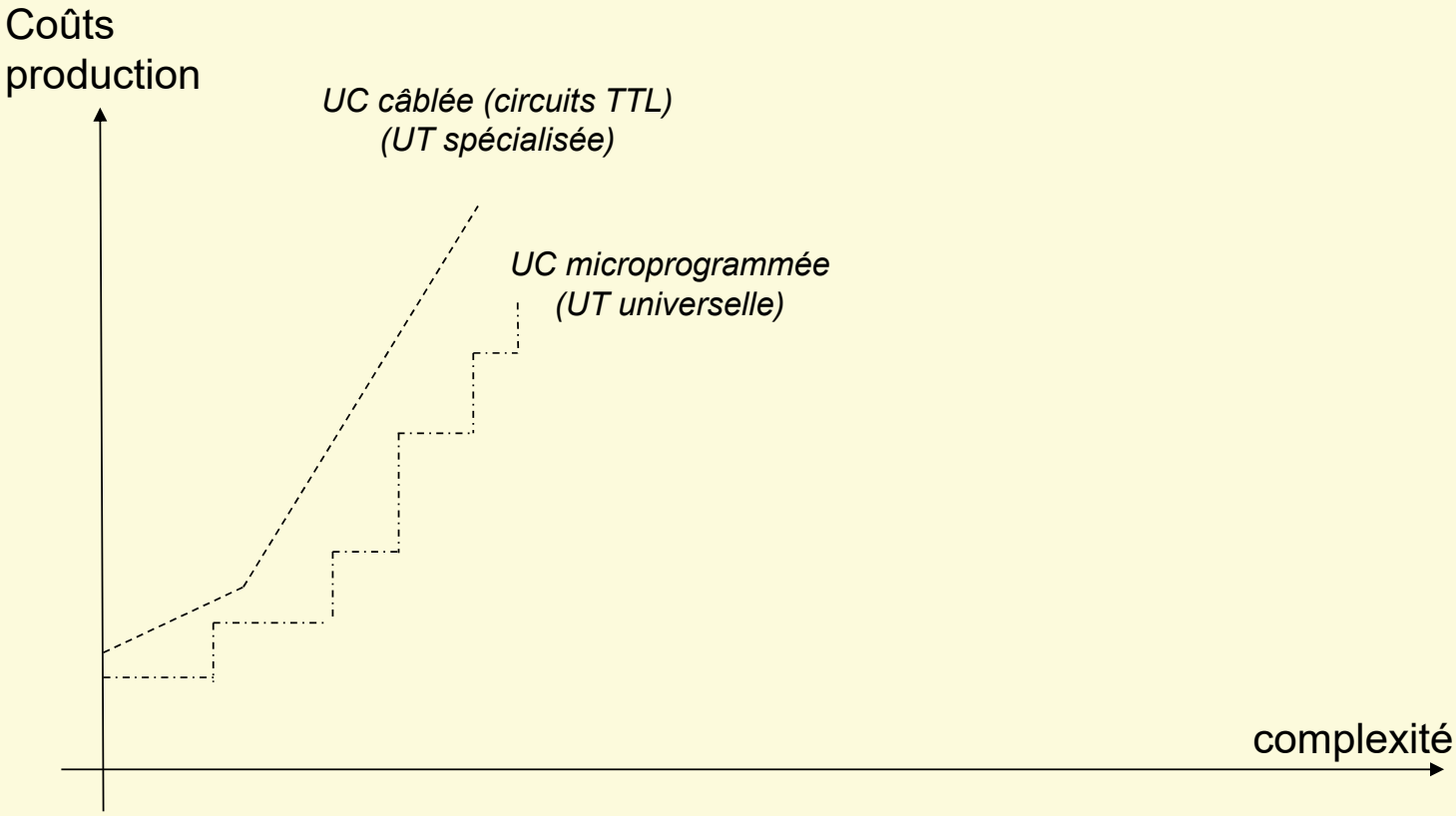
UC	câblée	câblée	(µ) programmée	(µ) programmée
UT	spécialisée	universelle	spécialisée	universelle

A considérer :

- performances (vitesse de fonctionnement)
- coût et temps de développement
- souplesse (facilité de modification)
- coût de production

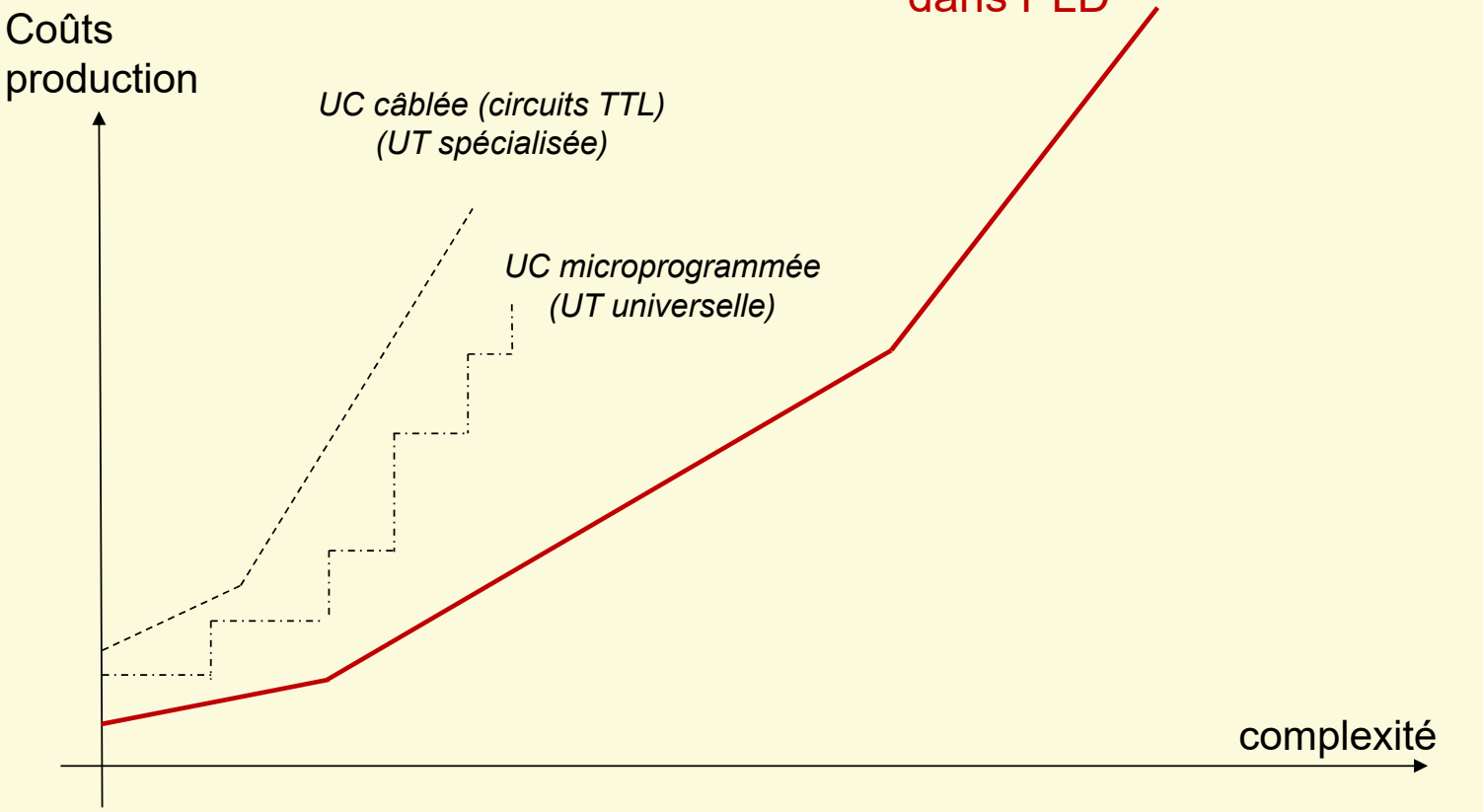
Combinaisons UC - UT

Coûts de production/ complexité :



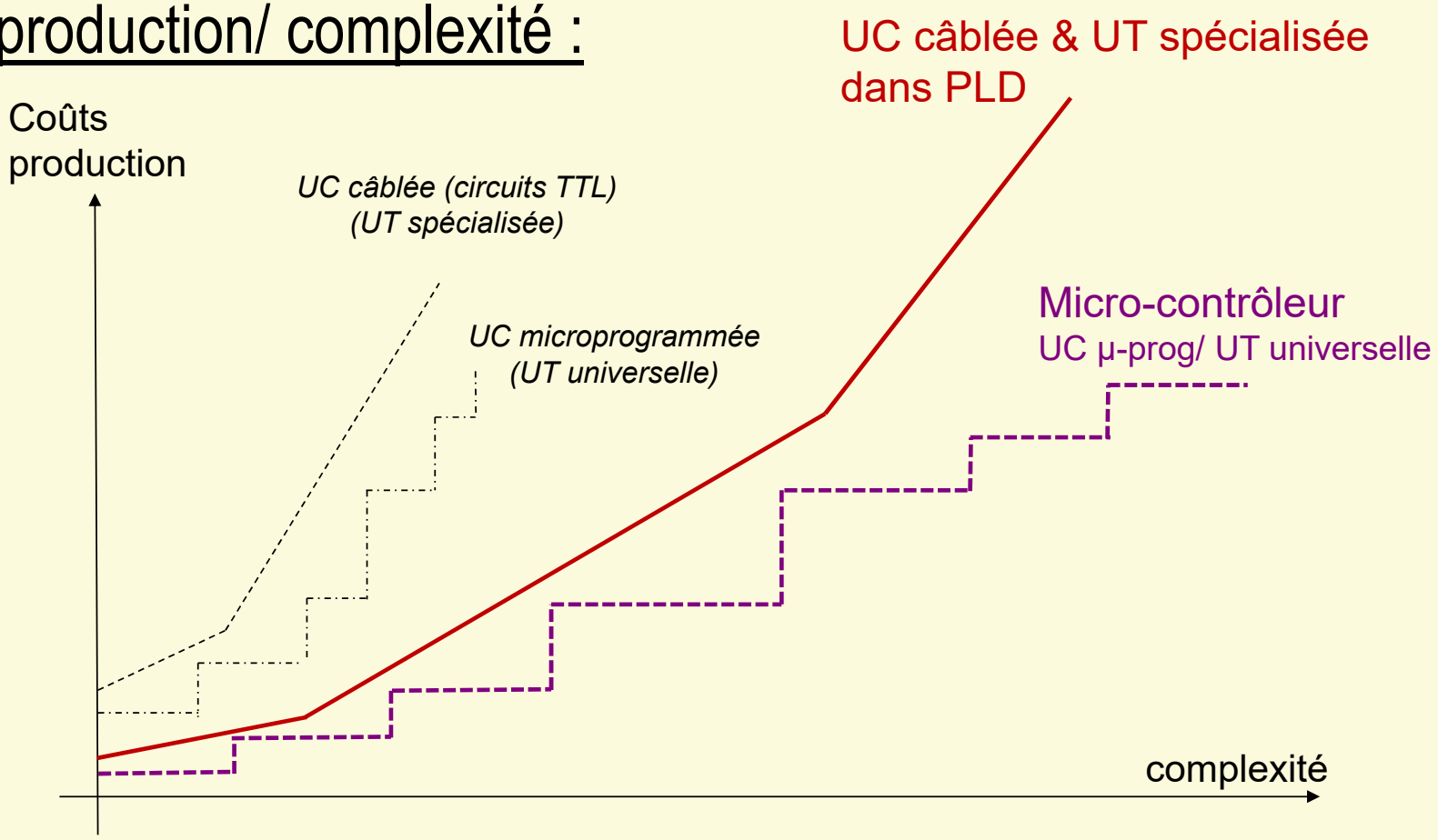
Combinaisons UC - UT

Coûts de production/ complexité :



Combinaisons UC - UT

Coûts de production/ complexité :



Combinaisons UC - UT

- **UC câblée - UT spécialisée**
  - performances maximums
  - coûts et temps de développement important
  - coûts de production proportionnel à la complexité et aux performances
  - souplesse et coûts = améliorés par outils EDA pour FPGA
- Domaine d'utilisation:
  - Applications à hautes et très hautes performances
  - Applications avec des fonctions particulières (non standards)
  - Intégration dans un seul circuit FPGA
  - Coût fortement diminué grâce aux FPGA *low-cost*
  - Utilisation d'IPs (design re-use)

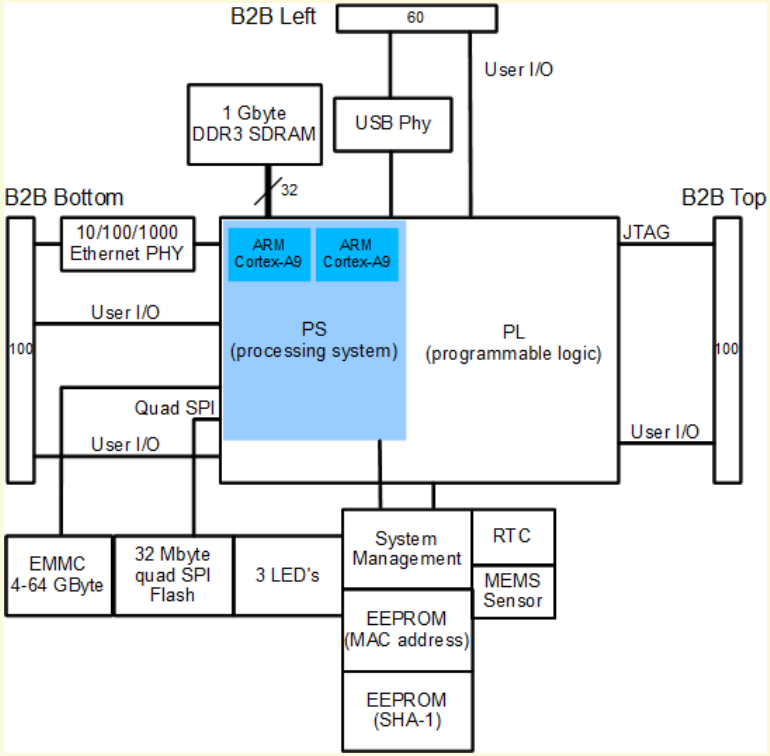
Combinaisons UC - UT

- **UC microprogrammée - UT spécialisée**
  - performances moyennes à élevées (nouveaux SoC-FPGA!)
  - coût et temps de développement moyens à grands
  - souplesse élevée si seul le programme est modifié
  - coût de production élevé
- Domaine d'utilisation:
  - Plus utilisée tel quel!
  - Solutions actuelles:
    - Microcontrôleur (8, 16 ou 32 bits) + FPGA
    - Nouveaux circuits SoC-FPGA

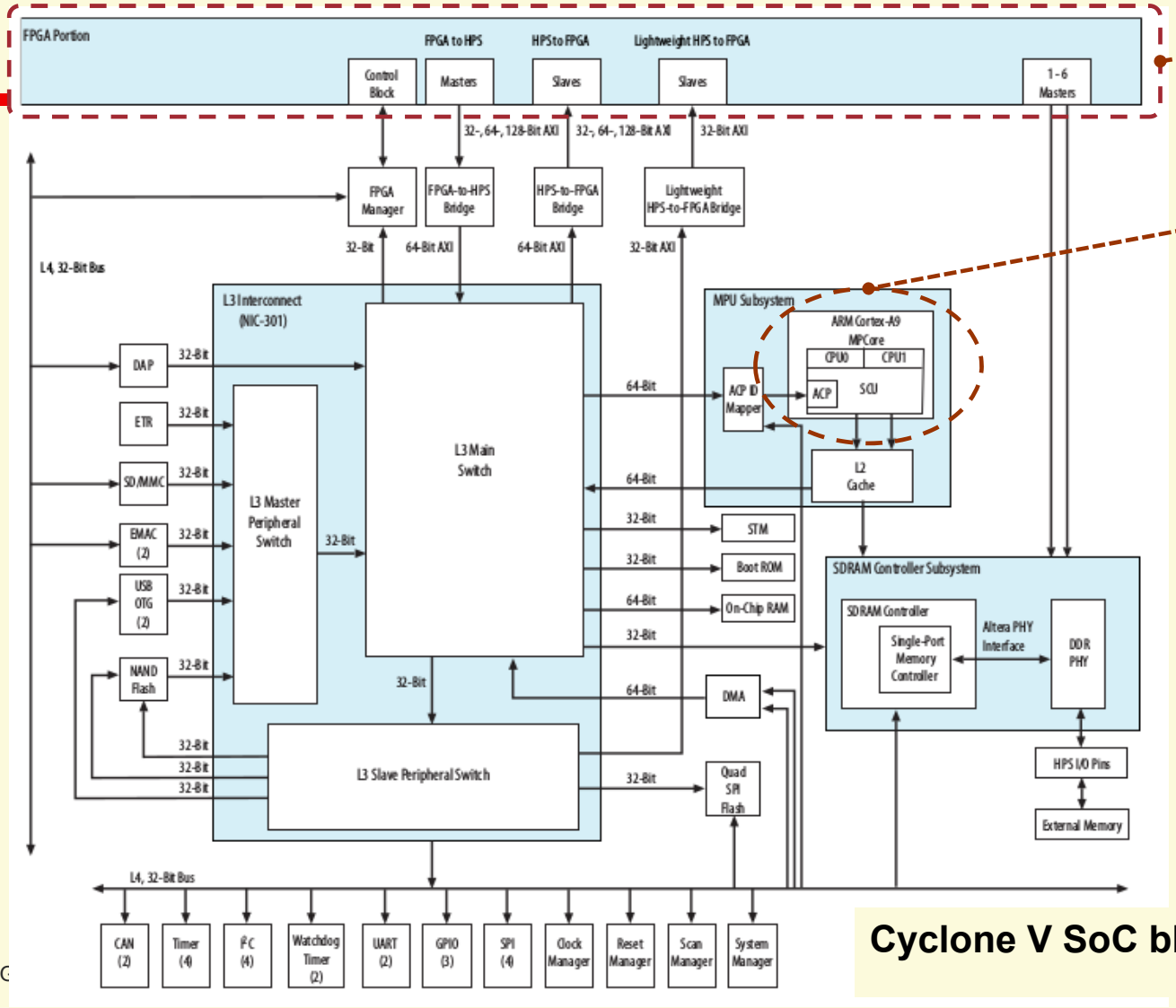
Combinaisons UC - UT

- **UC microprogrammée - UT spécialisée**
  - Arrivée de SoC comprenant: **µC + FPGA** (Zynq)

Source:
Trenz Electronic TE0720 Series (Z-7020)
(http://www.trenz-electronic.de/)



Combinaisons UC - UT



FPGA

MPCore
Dual Core ARM

Cyclone V SoC block diagram

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:
  - peu d'application
  - architecture des 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** de 8, 16, 32 ou 64 bits
  - processeur de traitement du signal **DSP**

Méthode de conception MSS complexe

1) Etablir les spécifications, entrées/sorties (symbole)
2) Relations temporelles entrées ⇔ sorties, chronogrammes
3) Schéma-bloc grossier, puis
   Algorithme avec phrase, texte => **Organigramme grossier**
4) Détailler progressivement => Organigrammes évolués,
   choix **partition entre UC - 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 composants (µC, FPGA, mémoires, …)
8) Design UT : schémas, descriptions VHDL, …; test UT
9) Design UC : org détaillé, graphe des états ou progr.; test UC
10) Test de l'ensemble, montage et test final

Documentation : à toutes les étapes

Exercice d'introduction MSS

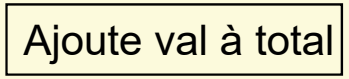
- Précédemment, nous avons réalisé un circuit pour la comparaison séquentielle de 2 nombres de 32 bits.
- Nous allons reprendre cet exemple afin d'appliquer la méthodologie de conception d'une MSS complexe :
  a) Etablir l'algorithme de comparaison bit à bit commençant par le poids faible.
  b) Identifier dans l'algorithme les fonctions nécessaires à implémenter dans le schéma de l'unité de traitement (UT).
  c) Etablir le graphe des états de l'unité de commande (UC) pour piloter l'UT afin de réaliser l'opération de comparaison.

  Voir présentation séparée et archive projet VHDL fourni (Cyberlearn

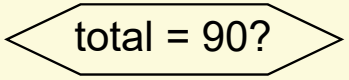
Exercice d'introduction MSS

- Organigramme :
  - établir l'organigramme sous forme graphique

ellipse : action asynchrone du reset



boîte rectangulaire : actions inconditionnelles

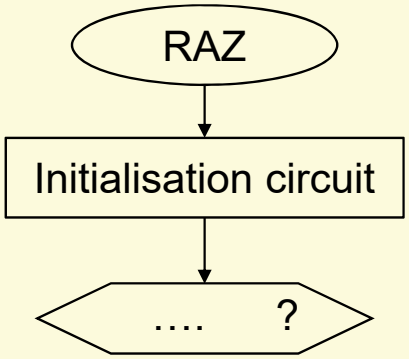


hexagone aplati : test déterminant l'évolution

flèche : évolution (pointe vers l'opération suivante)

Comparaison séquentielle

Etablir l'algorithme de comparaison bit à bit commençant par le poids faible



organigramme à compléter

- Voir présentation séparée pour l'exercice avec description VHDL :

  Comparaison séquentielle de 2 nombres

Comparaison séquentielle, UT

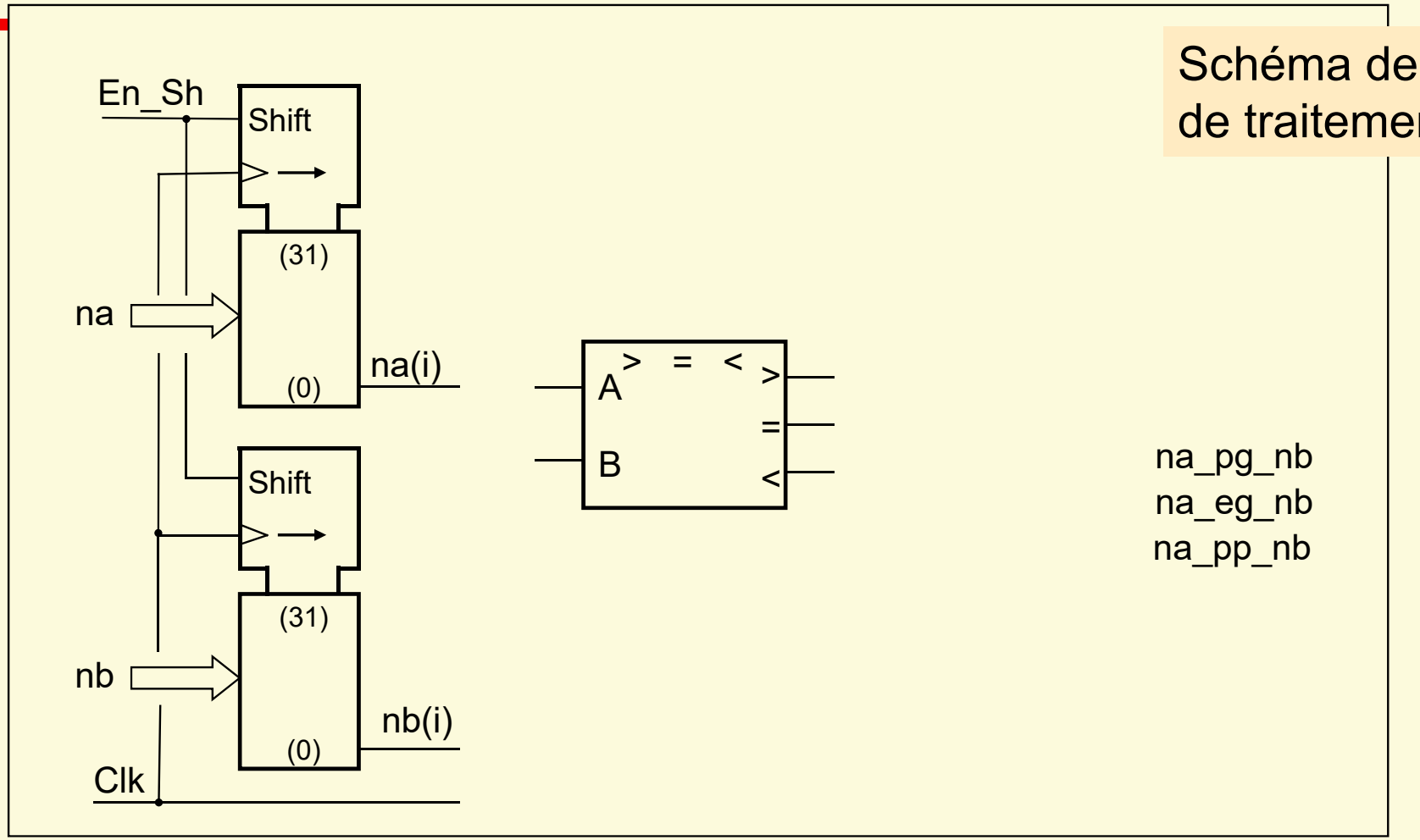


Schéma de l'unité de traitement UT

na_pg_nb
na_eg_nb
na_pp_nb

Exemple : multiplication non signée …

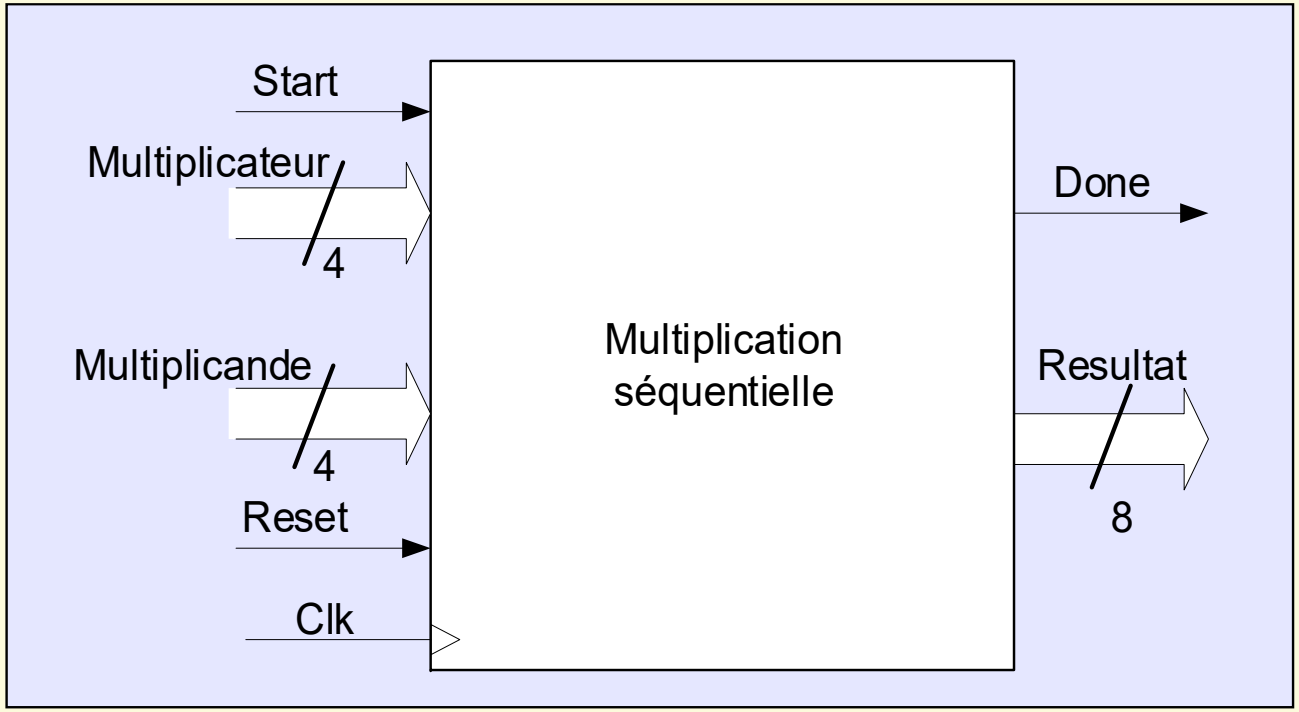
- Présentation de la démarche de conception d'une MSS complexe avec
  - Multiplication d'entier non signé
  - Algorithme déjà étudié au chapitre précédent (numération et arithmétique)

… 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

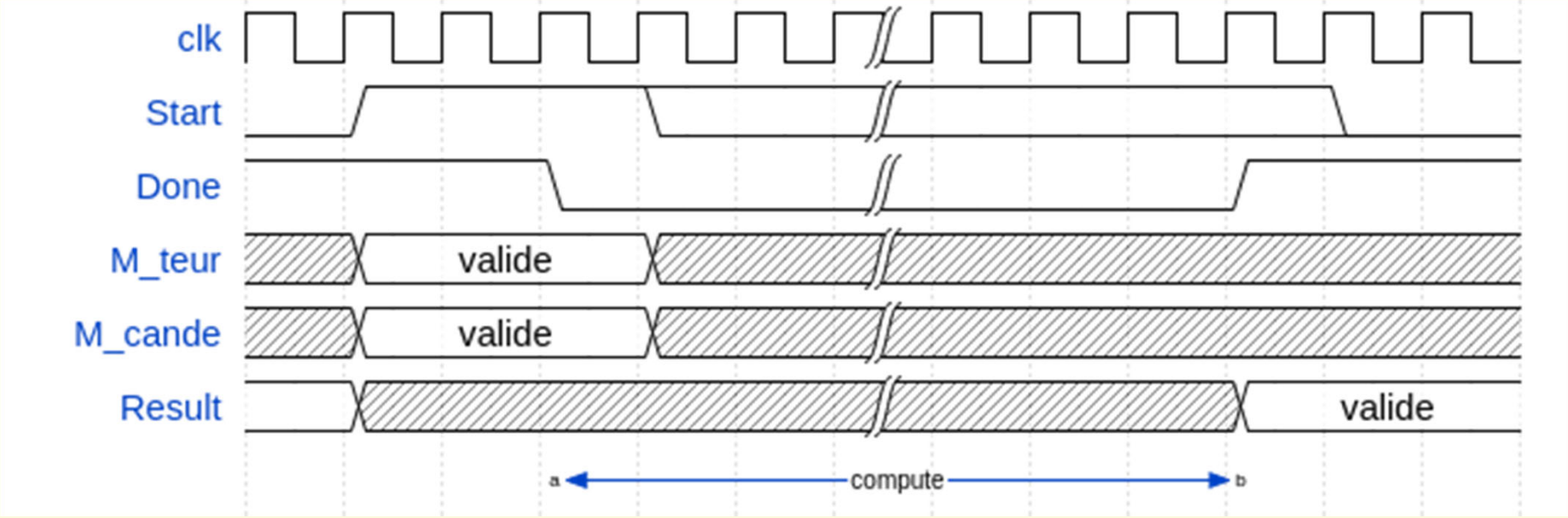
Symbole pour la multiplication

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

- Exemple de chronogramme du fonctionnement (externe) du multiplicateur avec l'évolution des signaux de contrôle : *Start* et *Done*



Algorithme de la multiplication

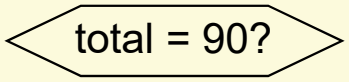
Algorithme déjà étudié précédemment :

```
Resultat_Haut := 0;
Resultat_Bas := Multiplicateur;

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 := Resultat_Haut & Resultat_Bas;
```

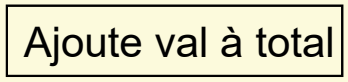
Remarque : nombre non signés,
dès lors Report = Carry

Organigramme

Description graphique d'un algorithme. Voici les symboles :



hexagone aplati : test déterminant l'évolution



boîte rectangulaire : actions inconditionnelles

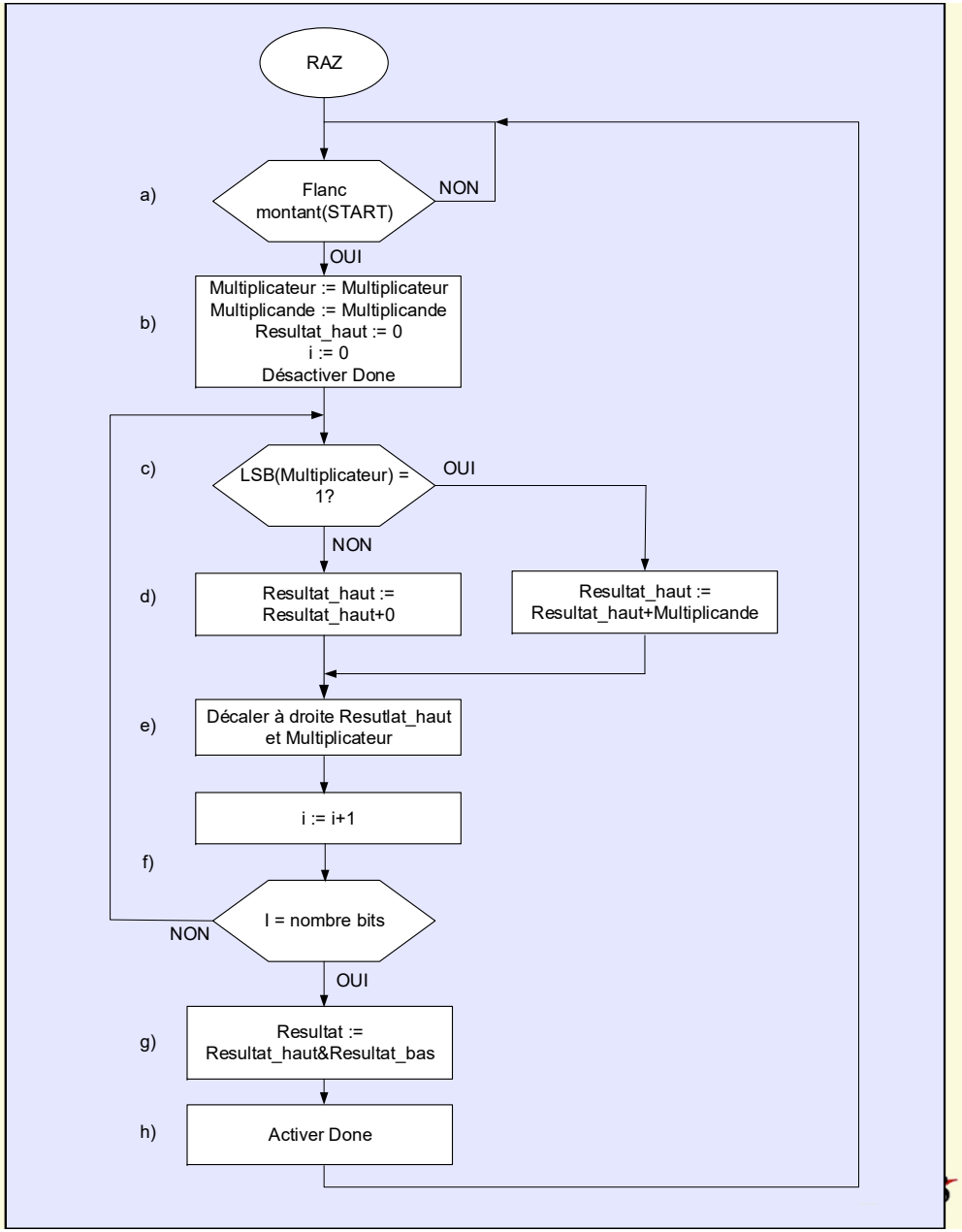
flèche : évolution (pointe vers l'opération suivante)

boîte oblongue : actions conditionnelles Pas recommandé

Organigramme grossier

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

Traduction graphique de l'algorithme textuel

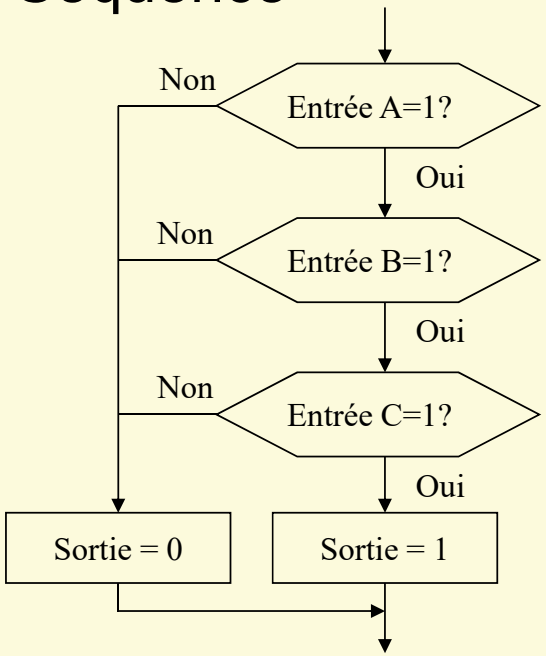


Partition UC / UT

Séquence (UC) ou fonction combinatoire (UT) ?

- Réaliser un ET logique avec trois signaux

- Séquence



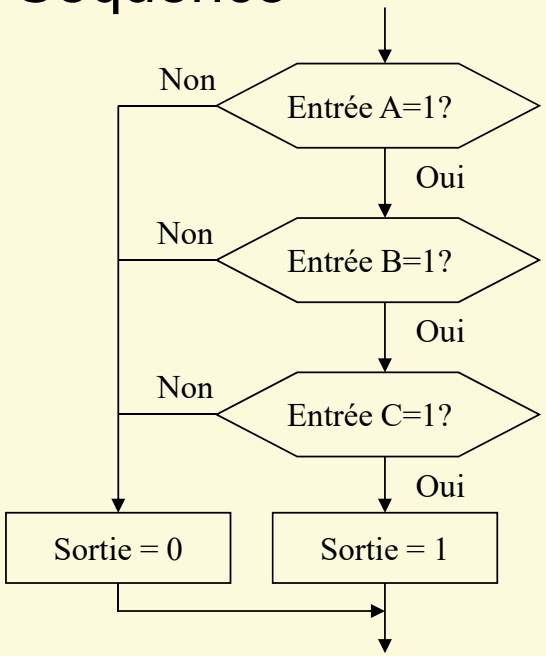
≡

Partition UC / UT

Séquence (UC) ou fonction combinatoire (UT) ?

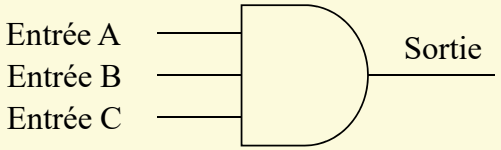
- Réaliser un ET logique avec trois signaux

- Séquence



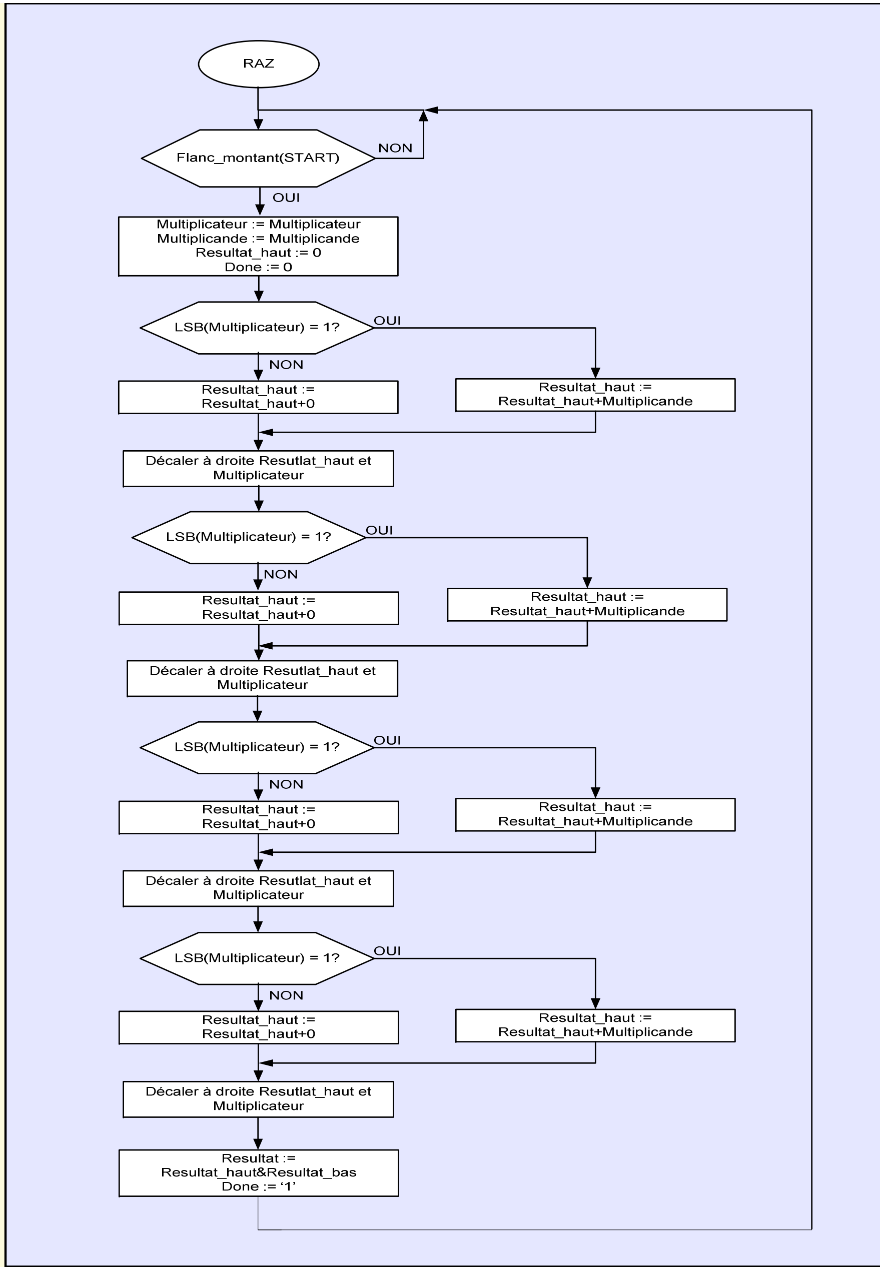
≡

- 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
  - suppression du compteur de boucle



Partition UC / UT

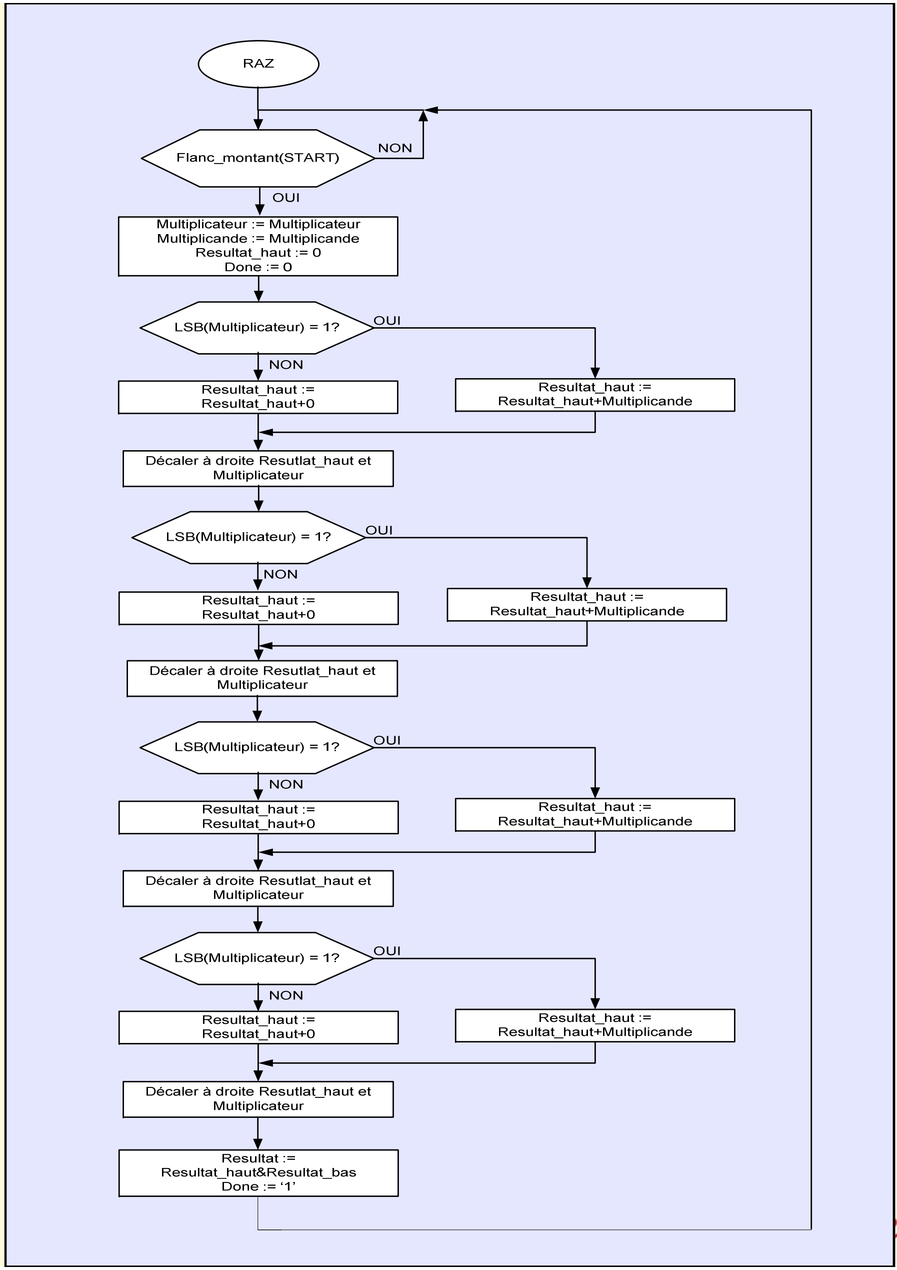
- Performance maximum → mettre toutes les fonctions identifiées dans une UT spécialisée
- Fonctions identifiées :
  - mémorisation du résultat_haut, résultat_bas/multiplicateur, multiplicande
  - test du LSB du multiplicateur : signal transmis à l'UC
  - addition résultat_haut + (multiplicande ou 0)
  - décalage Report, résultat_haut, résultat_bas/multiplicateur
    - Rem : cas de nombre non signés, dès lors Report = Carry
  - 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 carry (report)
  - 1 bascule DFF pour mémoriser le report

Organigramme évolué

- Rappel 1ère évolution
- Répétition de la séquence pour tous les bits du multiplicateur :
  - déroulée dans l'organigramme
  - solution pour un nombre de bits fixe



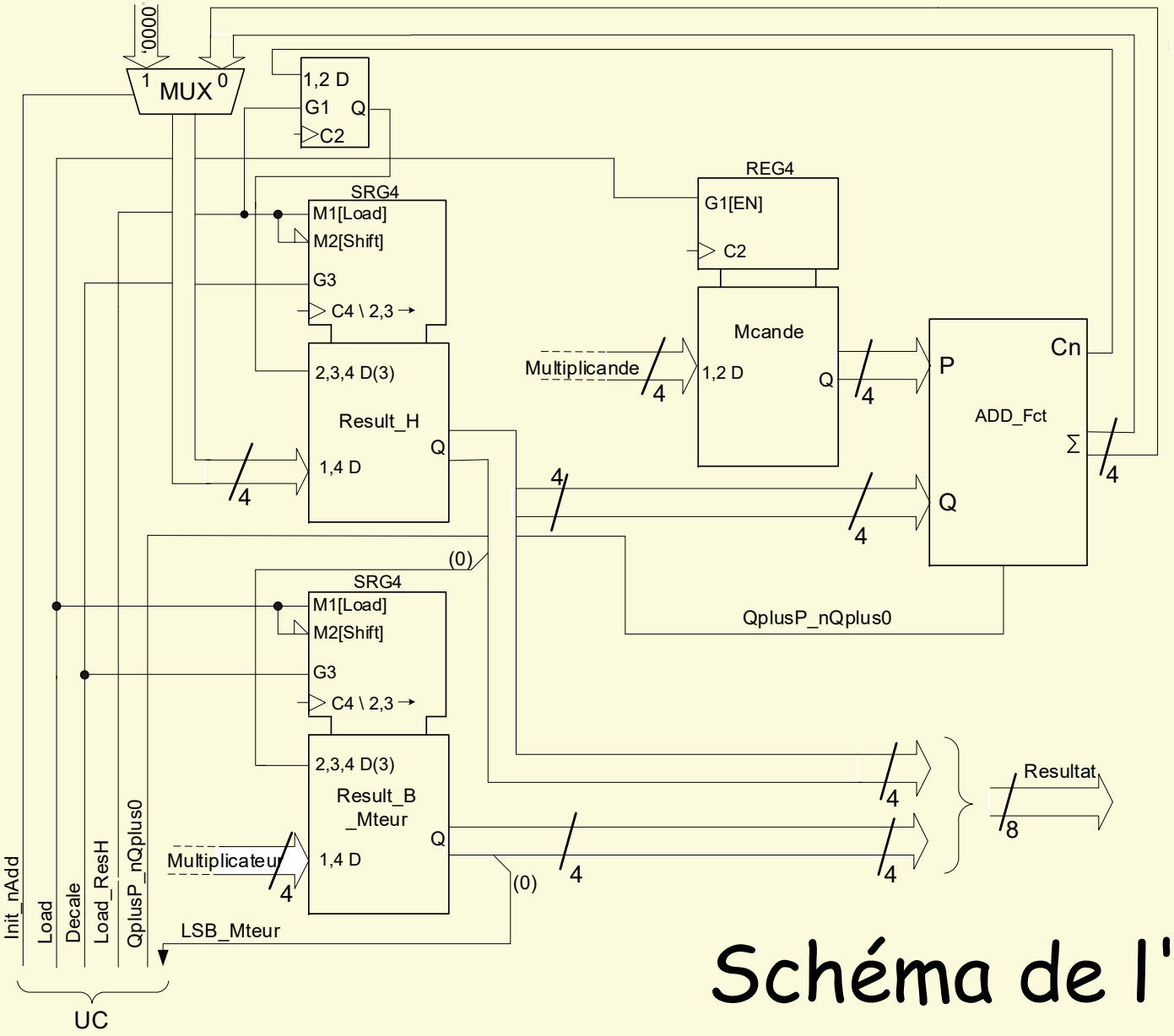


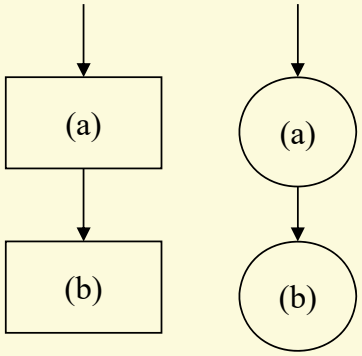
Schéma de l'UT

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

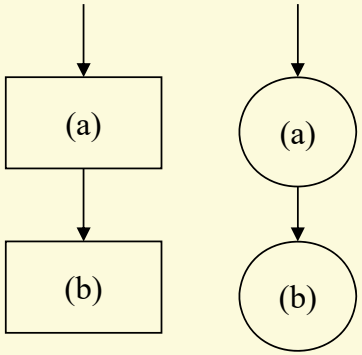
Organigramme ⇔ graphe

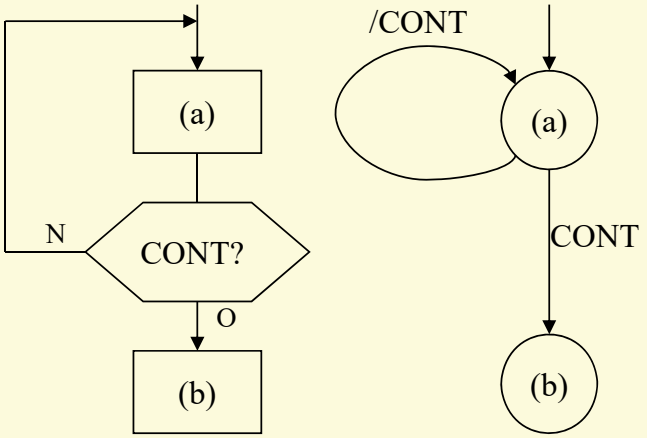
- boîte d'actions inconditionnelle = état dans le graphe
- conditions sur le(s) chemin(s) d'une boîte d'actions inconditionnelle à une suivante = condition de transition dans le graphe



Organigramme ⇔ graphe

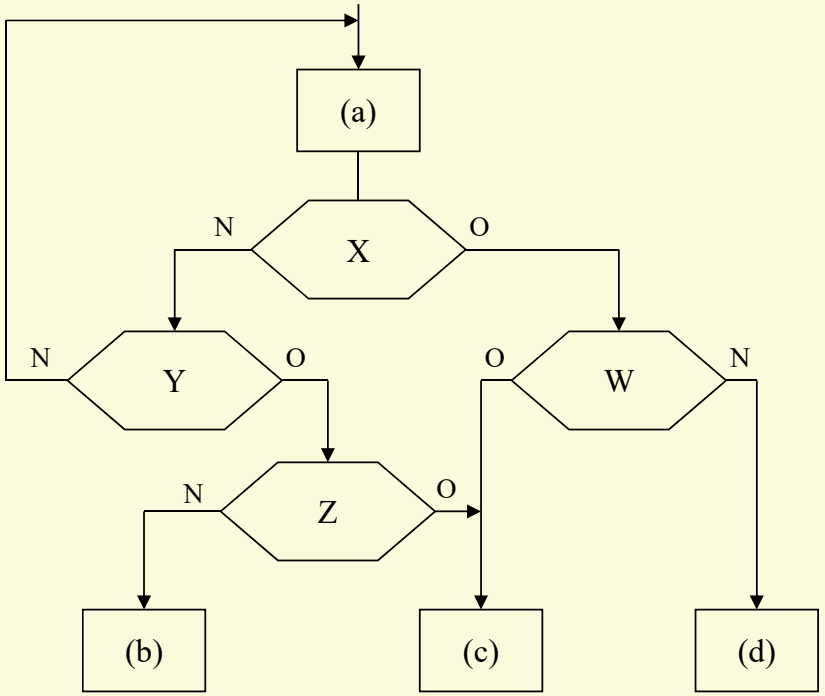
- boîte d'actions inconditionnelle = état dans le graphe
- conditions sur le(s) chemin(s) d'une boîte d'actions inconditionnelle à une suivante = condition de transition dans le graphe

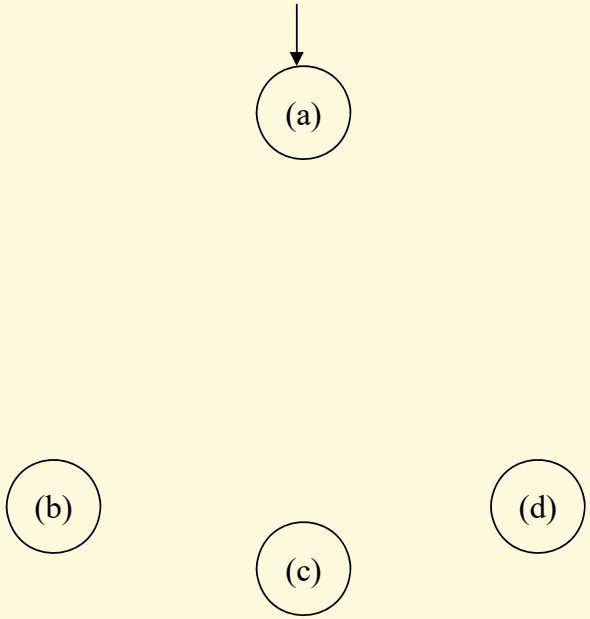




Organigramme ⇔ graphe

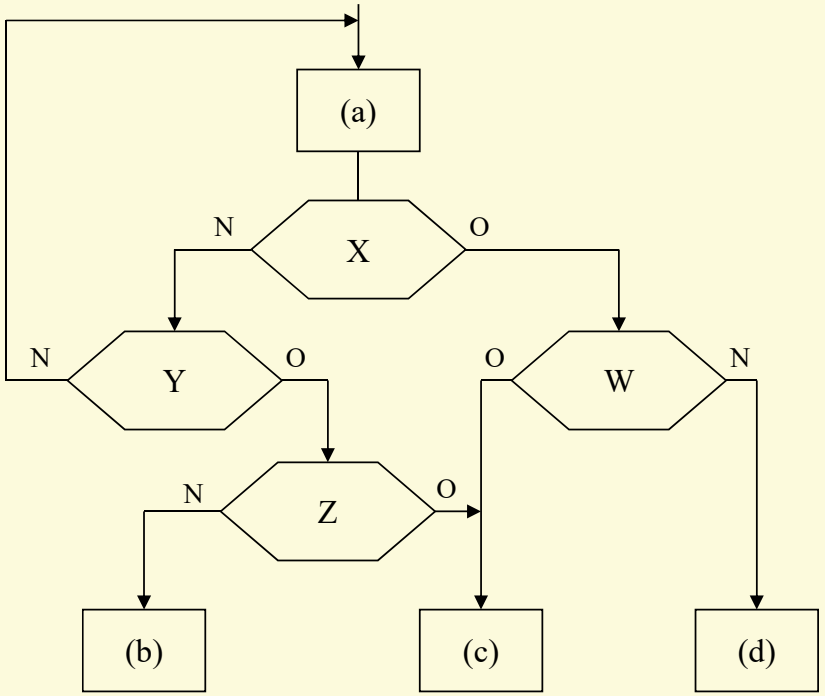
- Plusieurs chemins entre (a) et (c) sur un organigramme
  → une seule flèche sur le graphe

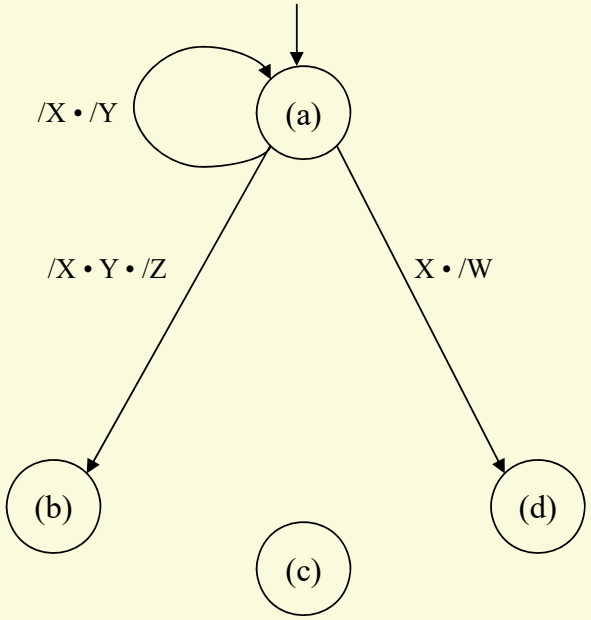




Organigramme ⇔ graphe

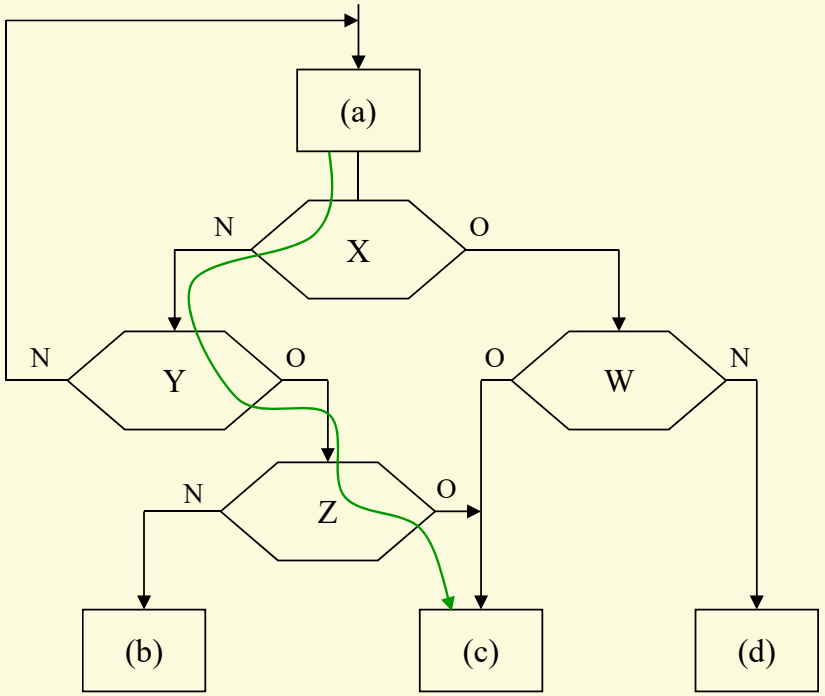
- Plusieurs chemins entre (a) et (c) sur un organigramme
  → une seule flèche sur le graphe

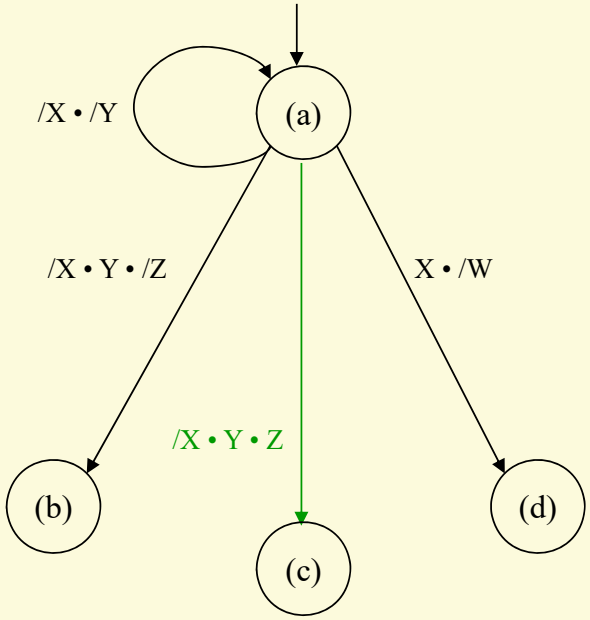




Organigramme ⇔ graphe

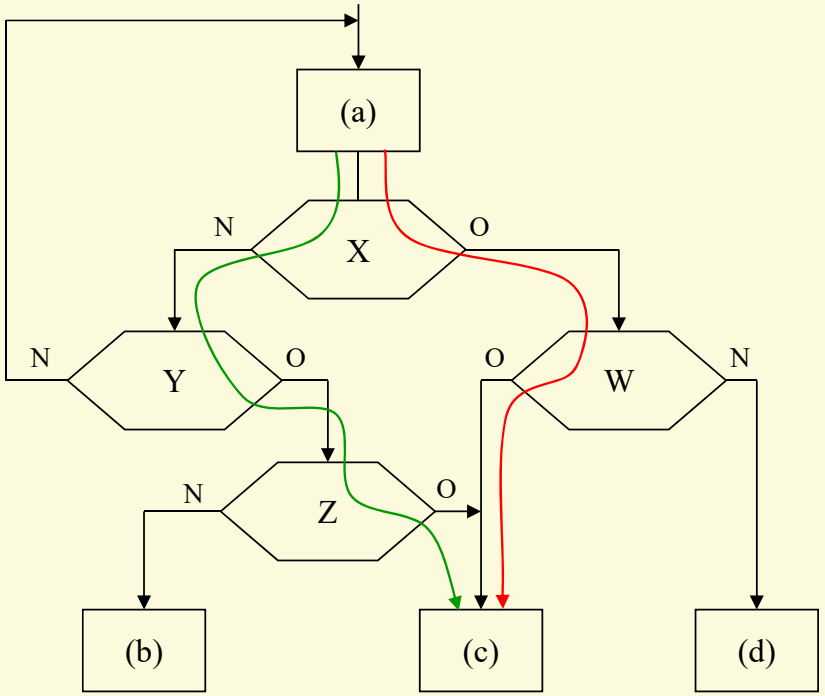
- Plusieurs chemins entre (a) et (c) sur un organigramme
  → une seule flèche sur le graphe

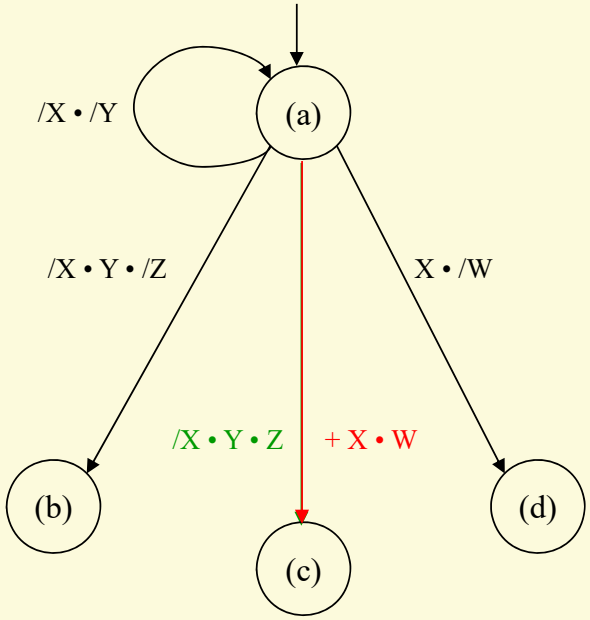




Organigramme ⇔ graphe

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





Organigramme détaillé et graphe

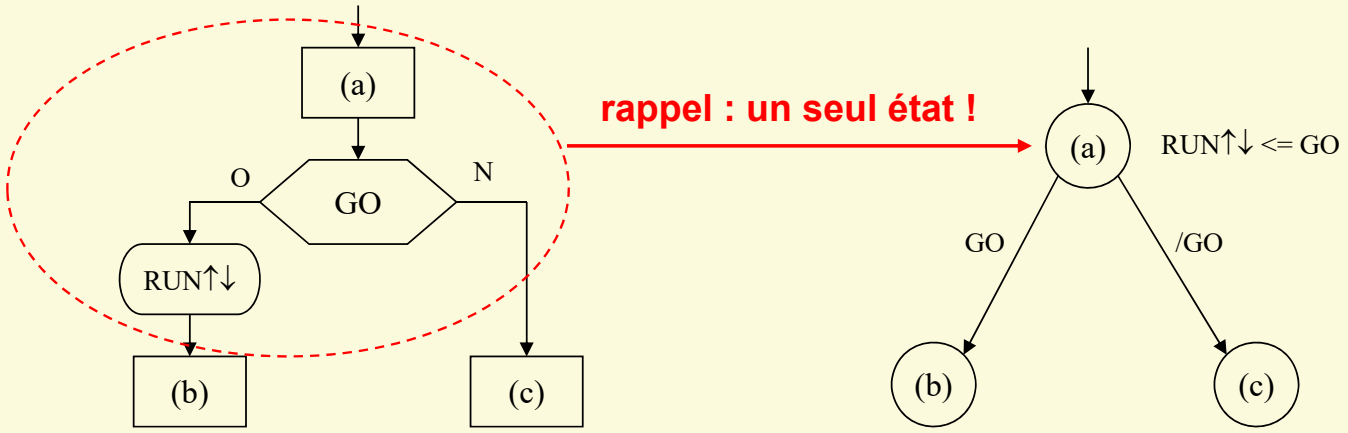
- Conventions
  - astérisque * indique test sur une entrée asynchrone
  - 2 astérisques * * indique des sorties devant être générées sans aléas
  - seuls les changements d'état des sorties seront indiqués dans les boîtes d'actions ou états, soit

↑ activation
↓ désactivation
↑↓ ou ↓↑ impulsion

} utiliser une logique positive !!

Sorties conditionnelles …

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



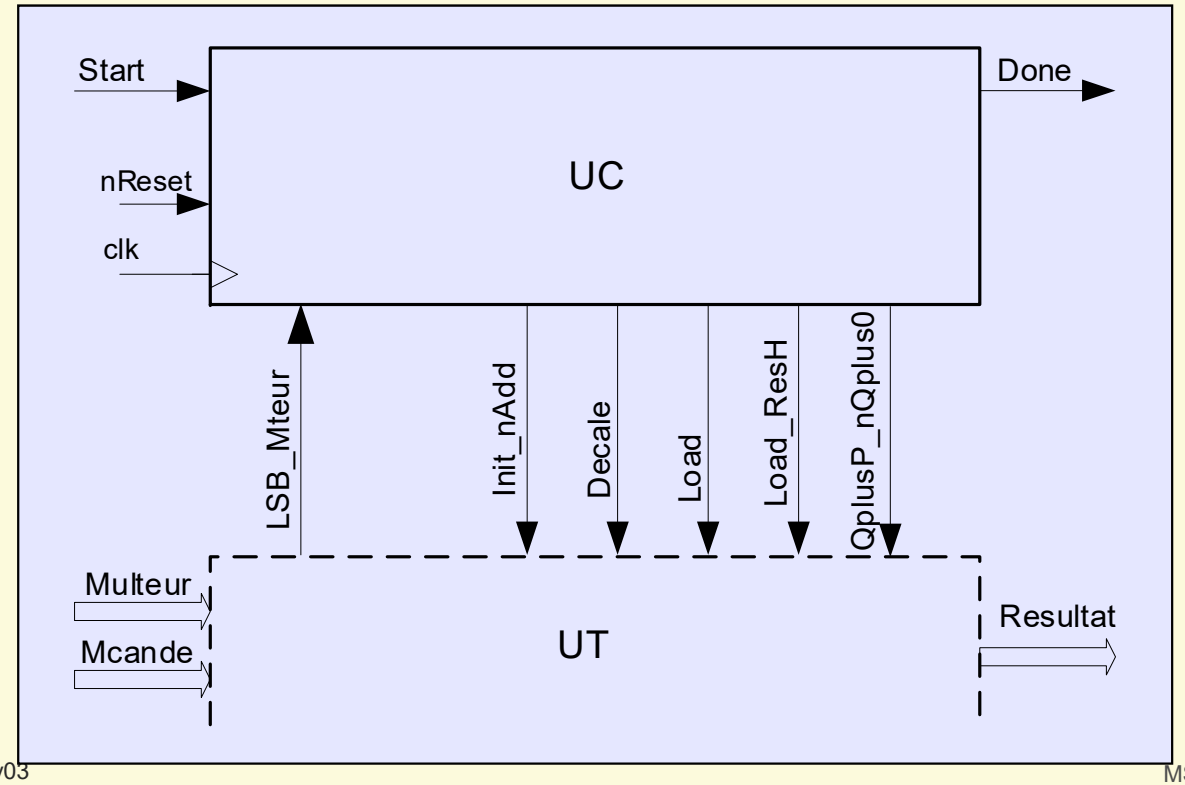
Remarque :
à éviter !

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 - UT

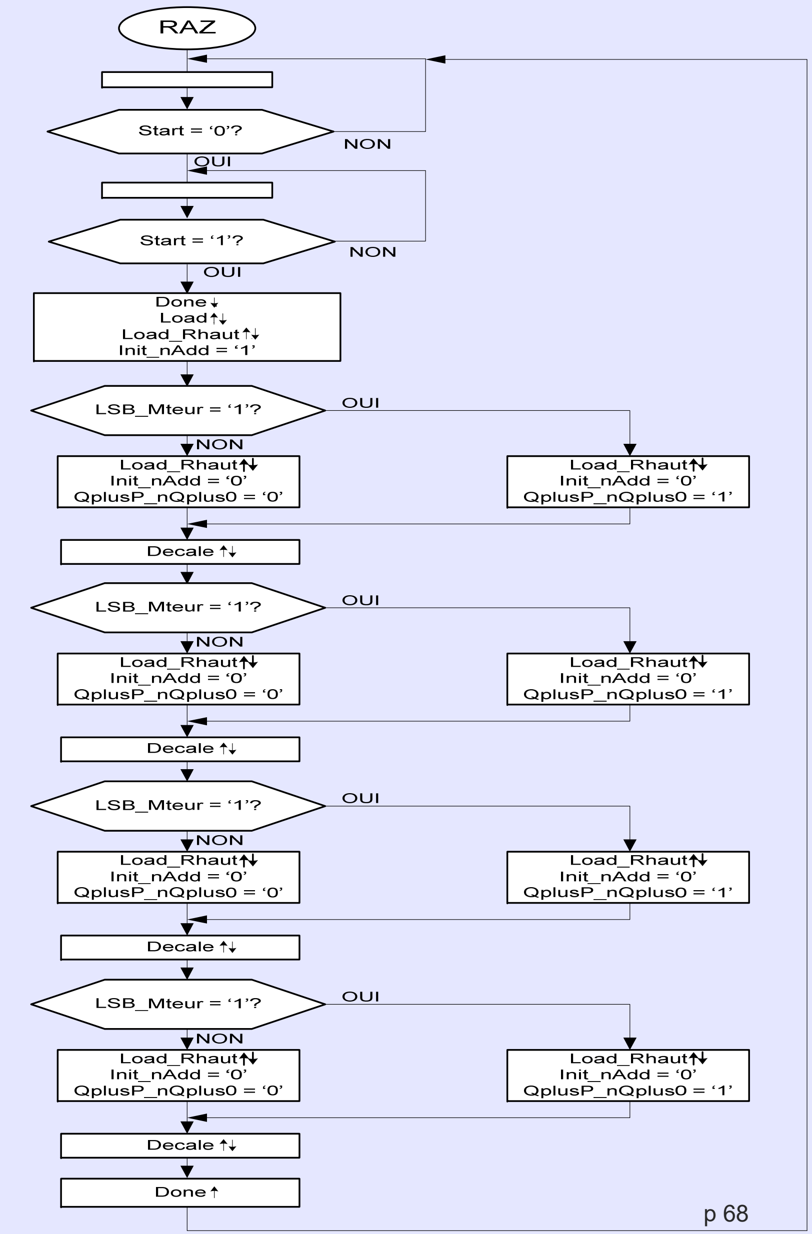
- Les signaux *Start* et *Done* sont directement connecté sur l'UC
- Les autres signaux sont connectés à l'UT



Organigramme détaillé

- Voici l'organigramme détaillé correspondant à l'organigramme évolué et à l'UT spécialisée
- Rappel :

  Répétition de la séquence pour tous les bits du multiplicateur déroulée dans l'organigramme

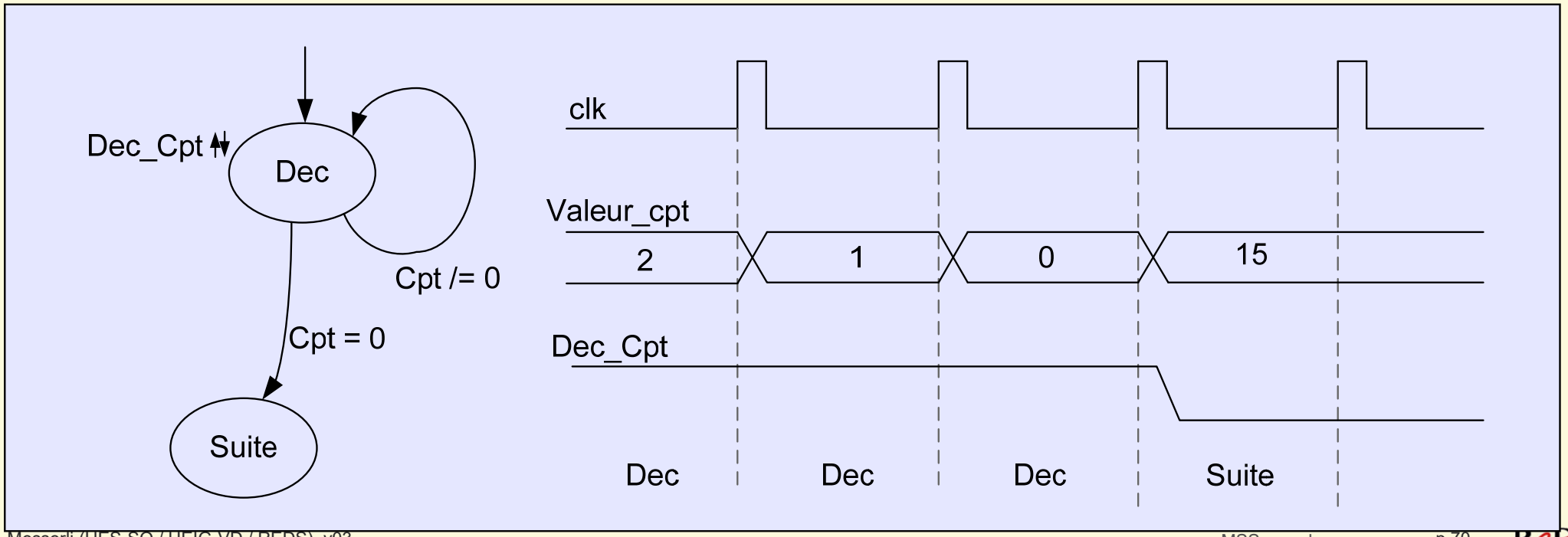


Structure hiérarchique : retard!

- Structure hiérarchique:
  - UC génère signaux de commande au cycle i
  - UT réagi aux commandes au cycle i+1
- Retard dans l'exécution des commandes, donc risque de fonctionnement incorrect

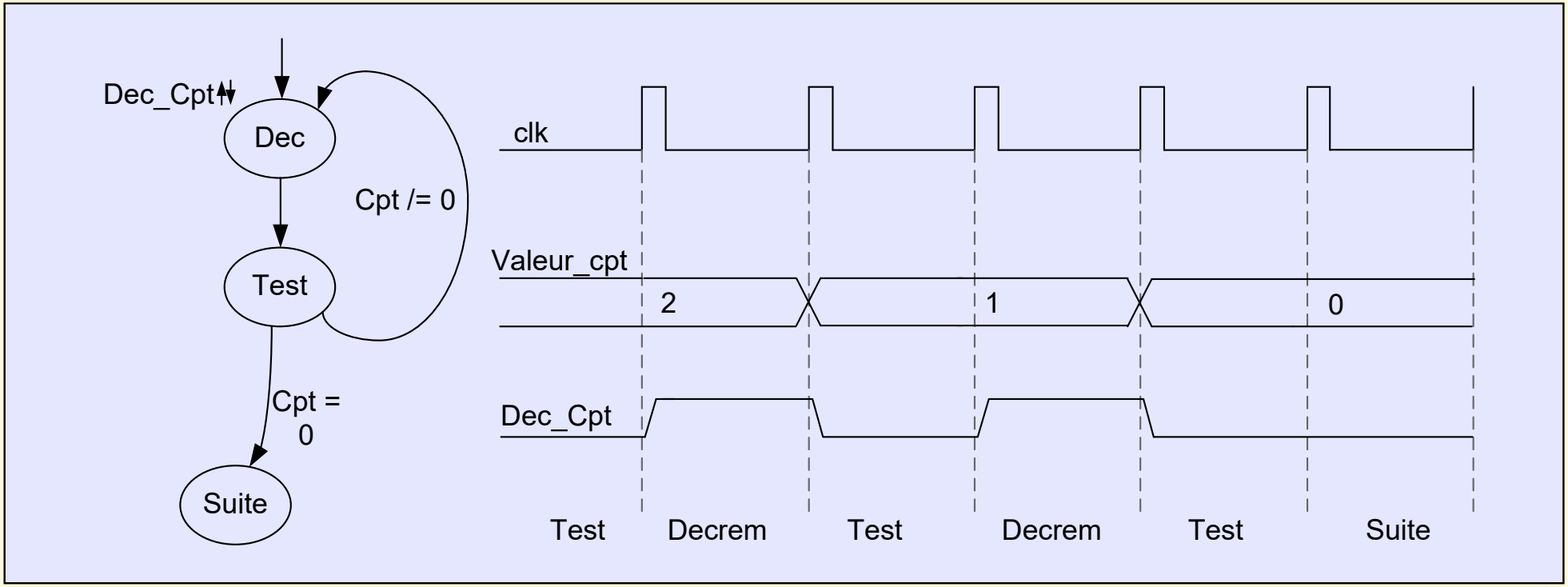
Fonctionnement incorrect

- Le graphe ci-dessous montre 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

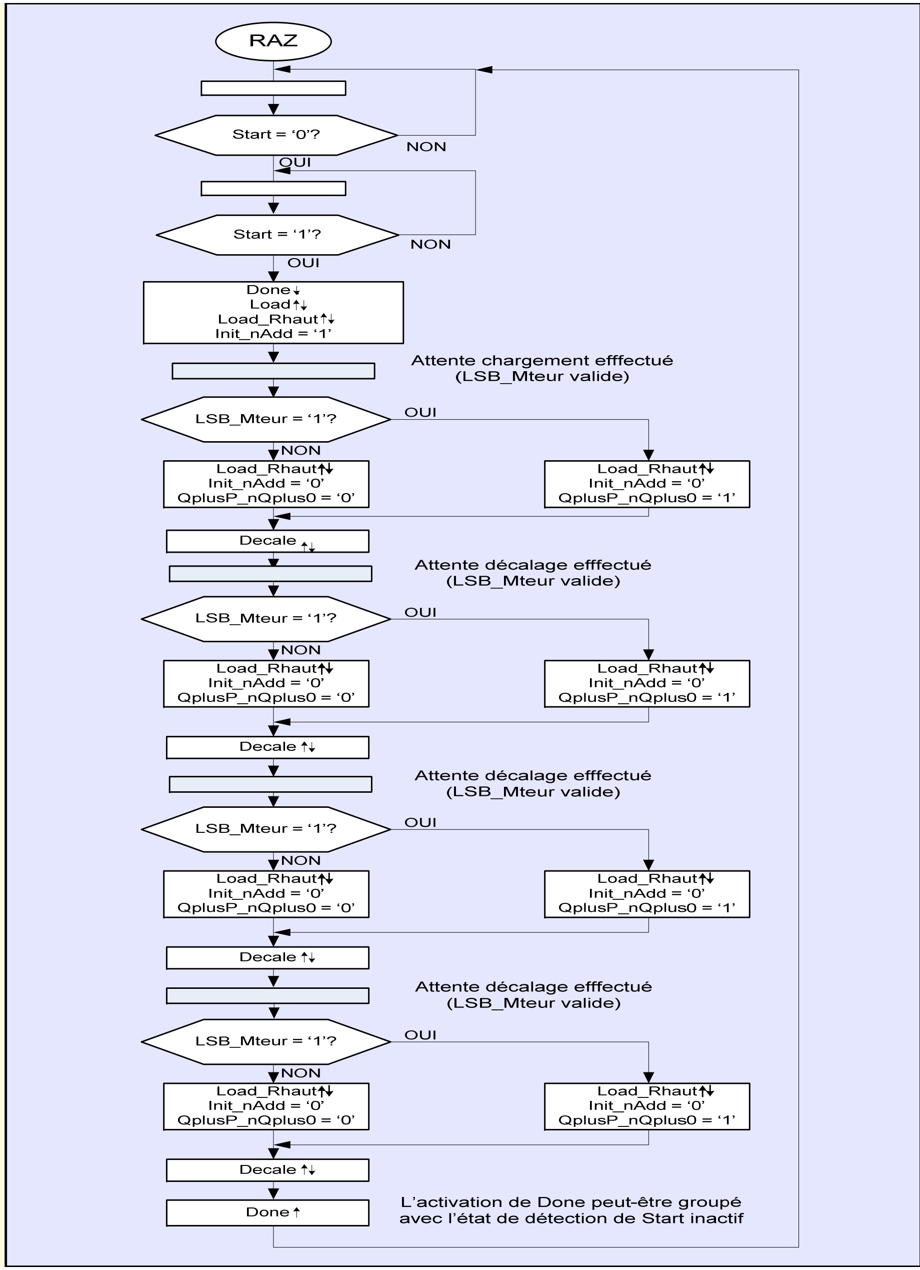
- Correction : rajouter un état pour tester l'état du compteur en désactivant le décomptage. Ajout état **Test**.



Organigramme détaillé final

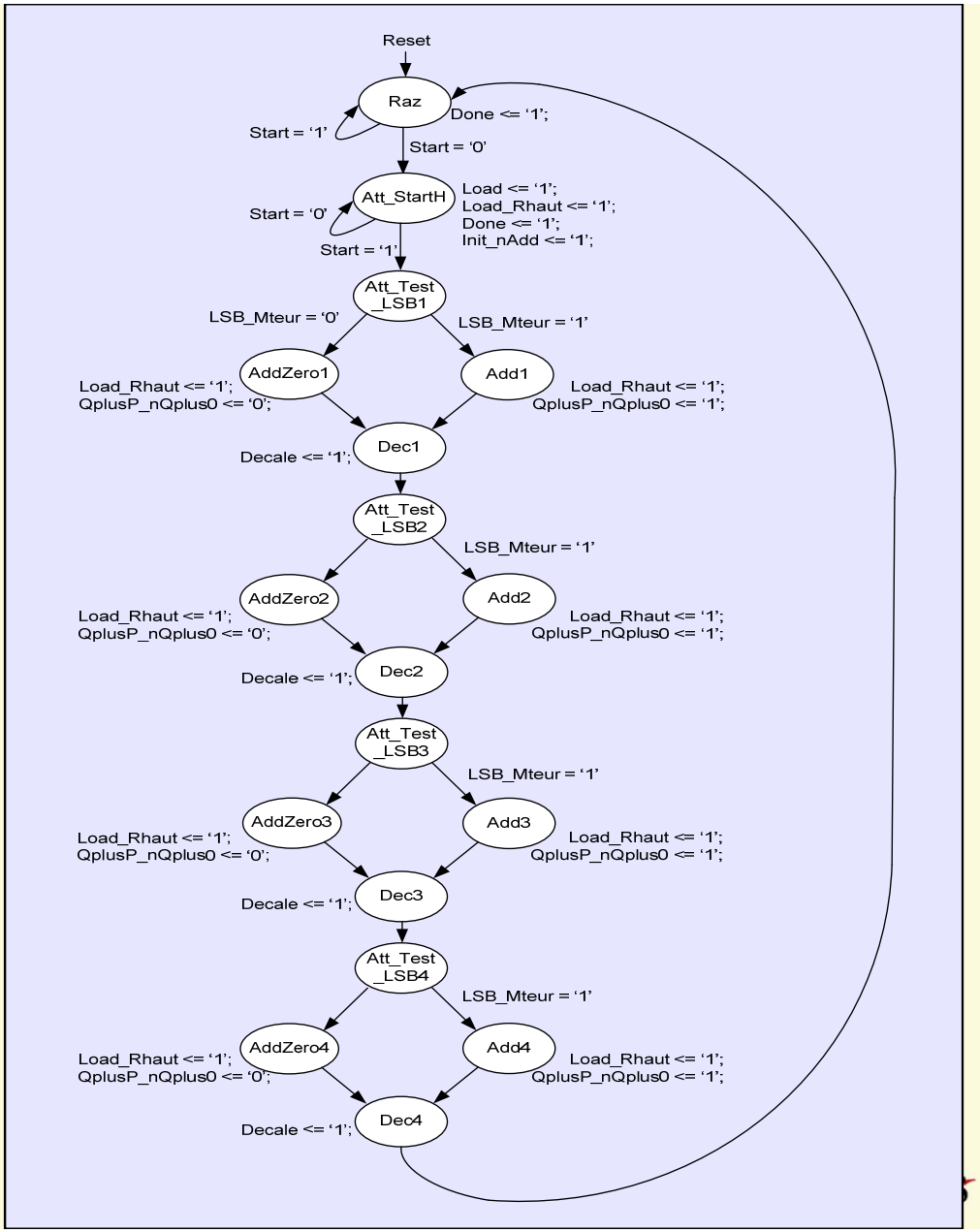
- Voici l'organigramme détaillé correspondant au schéma de l'UT spécialisé avec les corrections nécessaires au bon fonctionnement
- Rappel :

  Répétition de la séquence pour tous les bits du multiplicateur déroulée dans l'organigramme



Graphe des état 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



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 (I.1)

Exercices série I

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 même un troisième coup d'horloge avec l'attente pour le test du bit du multiplicateur.

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

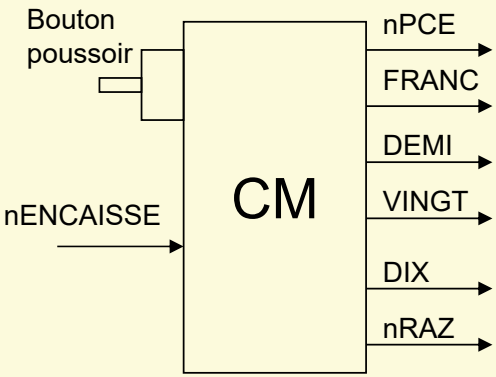
Exemple de MSS complexe

Distributeur automatique de billets pour la compagnie des Gyrobus Yverdonnois

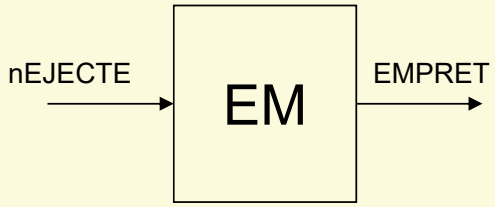
- Etapes de la conception du distributeur de billets
  - cahier des charges
  - schéma bloc global
  - algorithme (organigramme)
  - partition UC / UT (choix)
  - schéma bloc de l'UT
  - schéma de l'UT avec des fonctions standards

Exemple : Distributeur automatique de billets

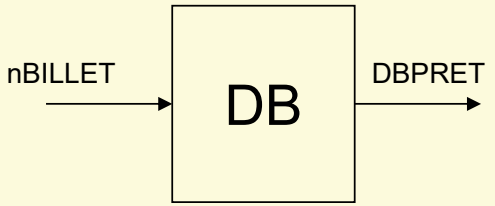
- Collecteur de Monnaie (CM)



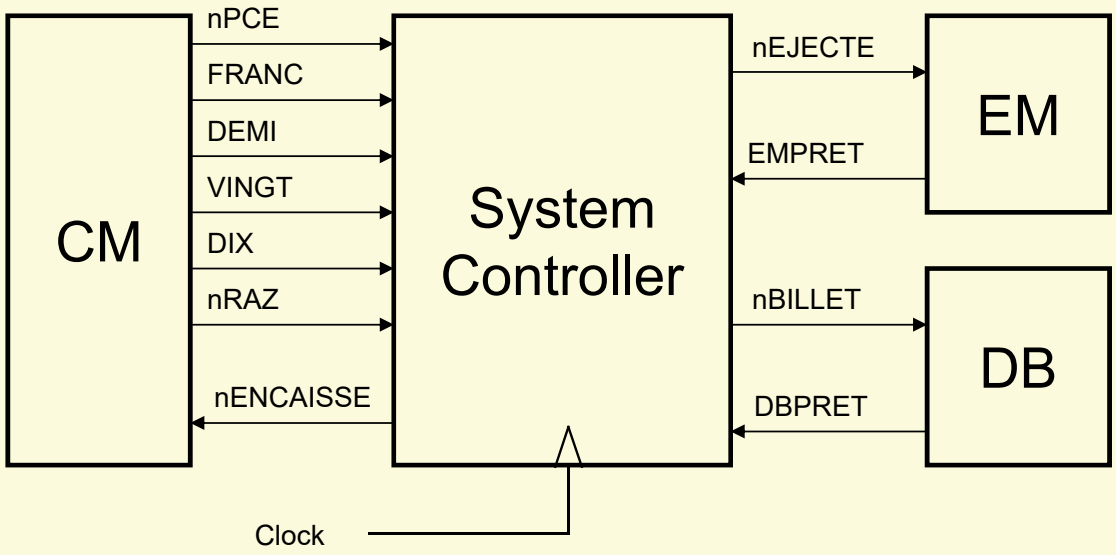
- Echangeur de Monnaie (EM)



- Distributeur de Billets (DB)



Distributeur de billets : schéma bloc



Distributeur de billets :

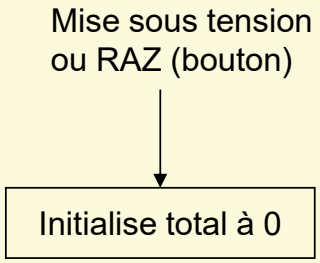
- Spécification détaillée : voir document pdf de la spécification
  Cours : Electronique numérique, Tome 4, chapitre 4

- Fonctionnement : démonstration Tcl/Tk
  fichier : distributeur_90cts.tcl

- Etablissement d'un organigramme grossier
  (1ère version)

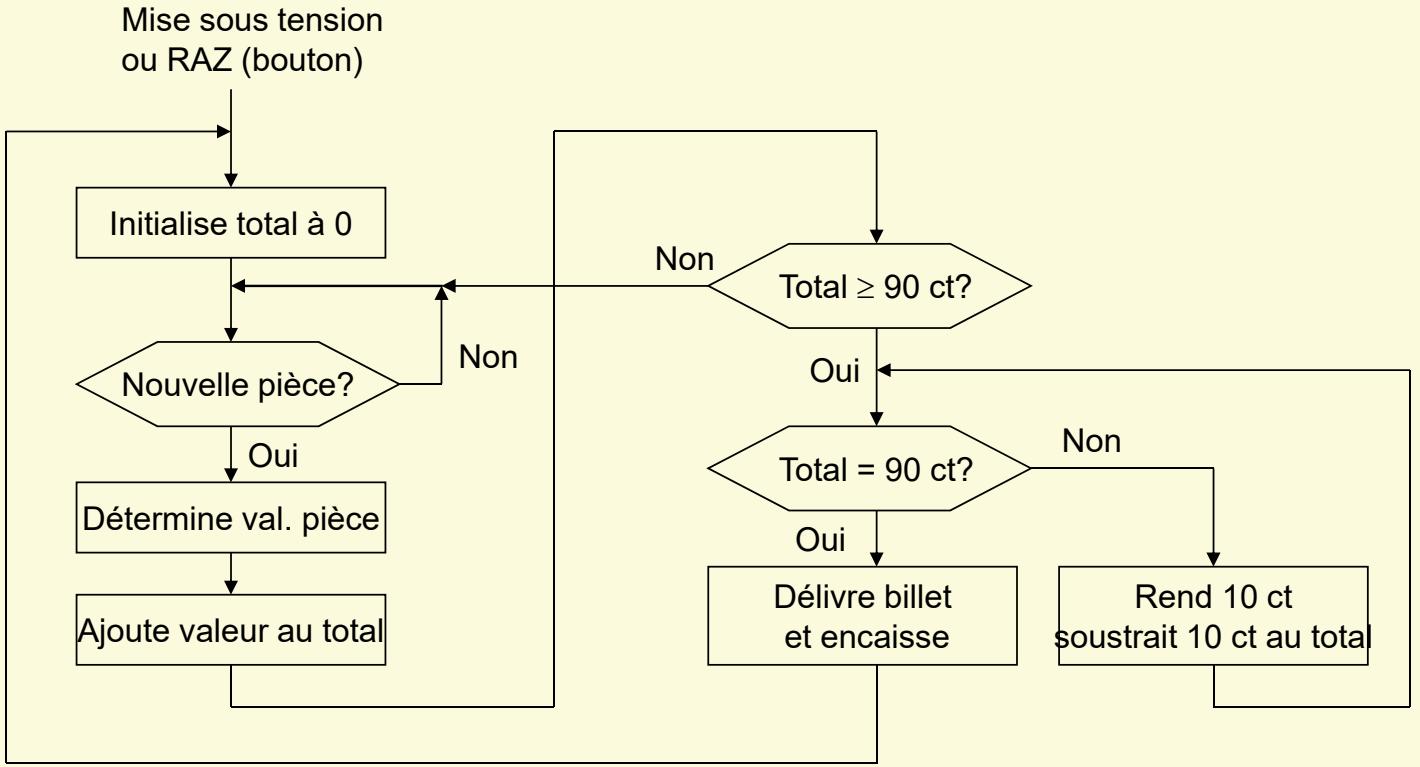
Distributeur de billets : algorithme global

1er organigramme grossier



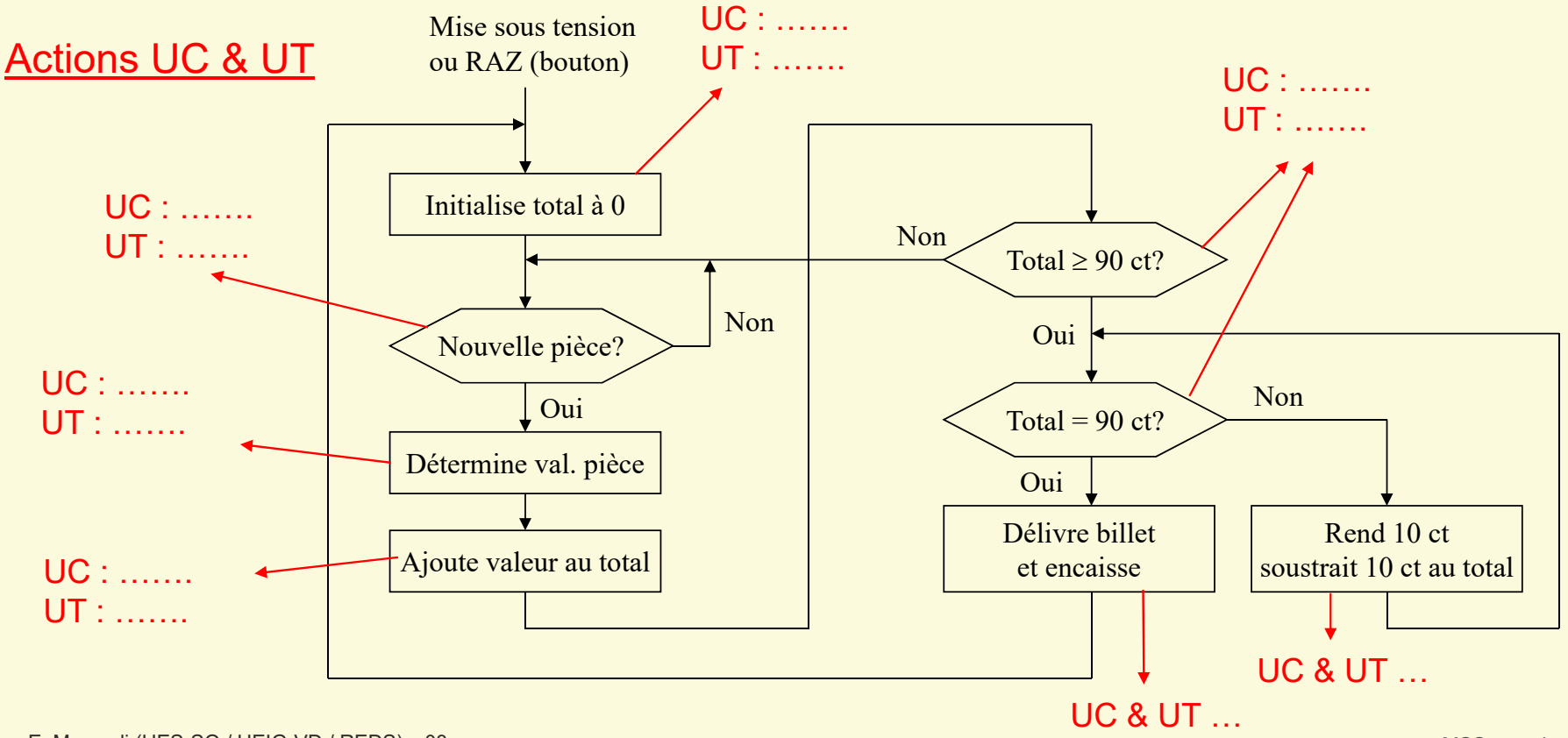
Distributeur de billets : algorithme global

1er organigramme grossier



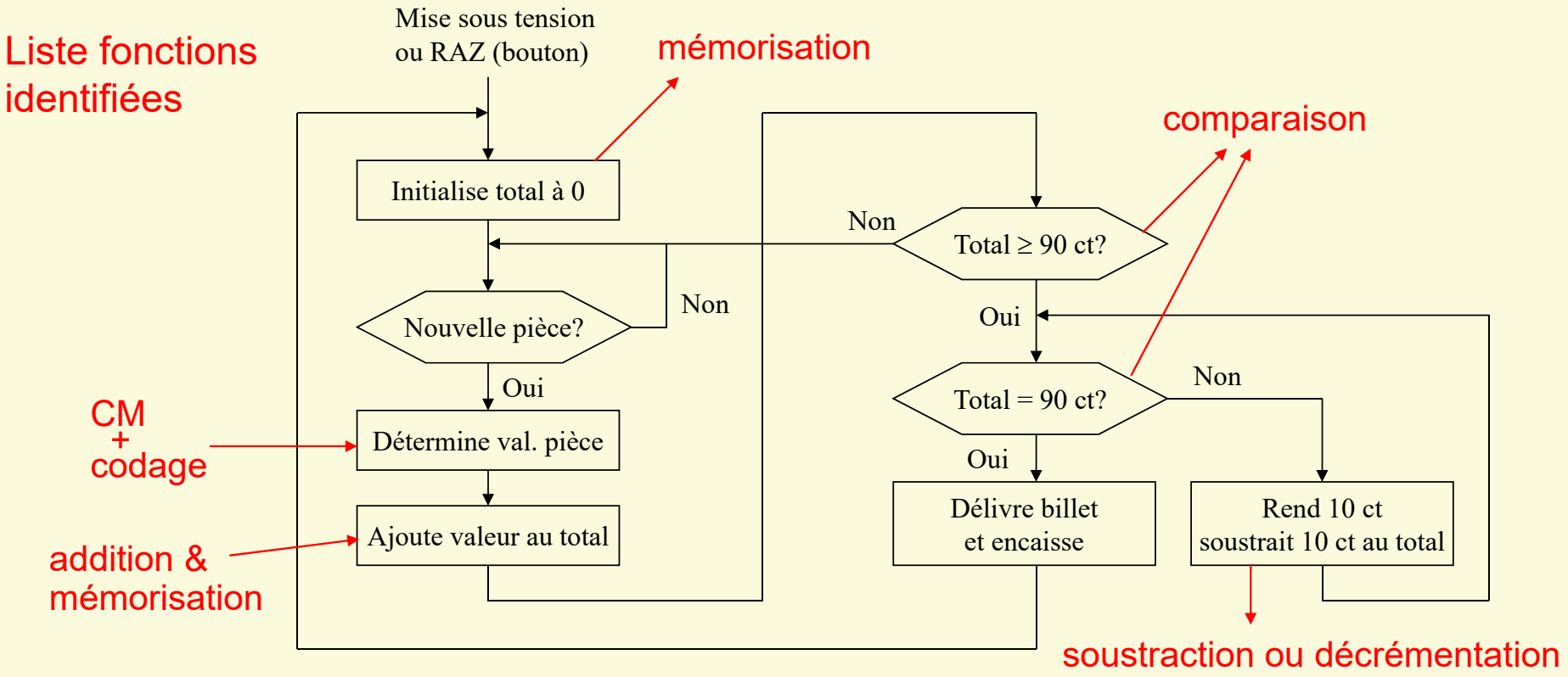
Identification des fonctions

- Identification fonctions UT et actions UC dans l'organigramme => choix partition UC/UT



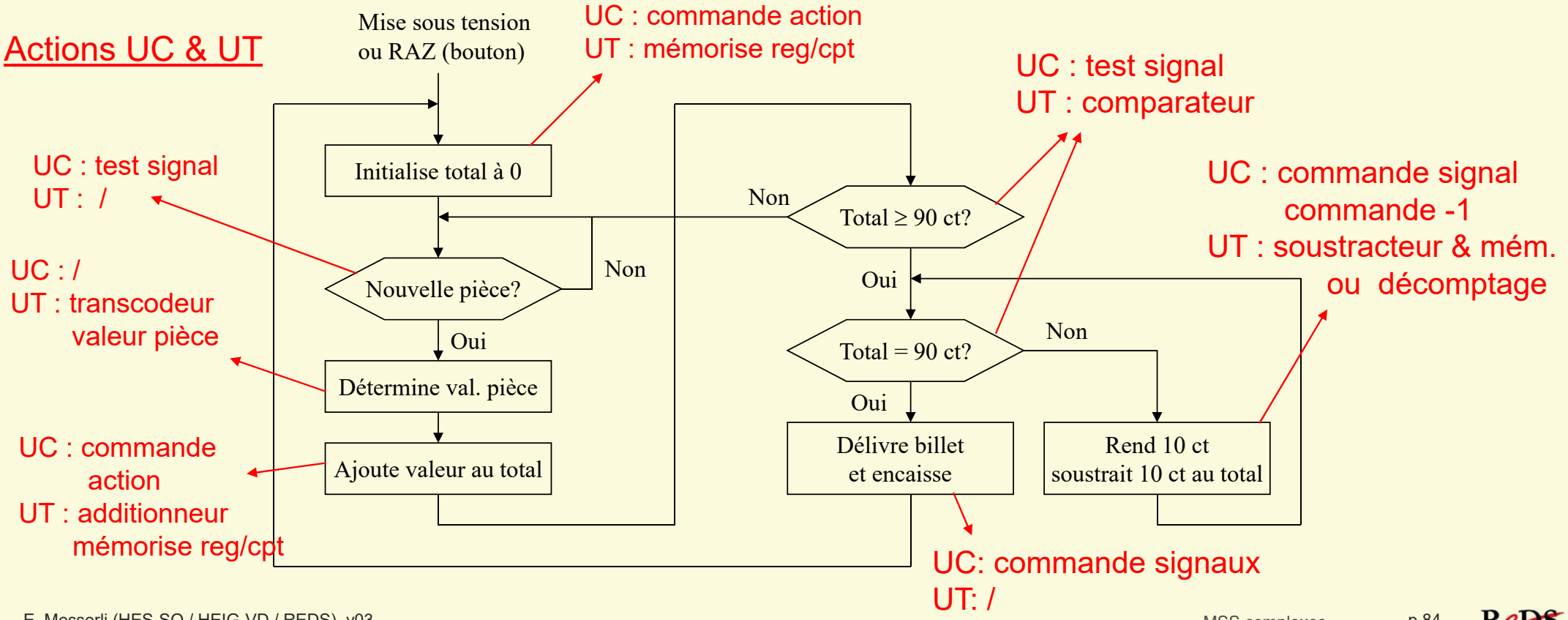
Identification des fonctions

- Identification fonctions UT et actions UC dans l'organigramme => choix partition UC/UT



Identification des fonctions

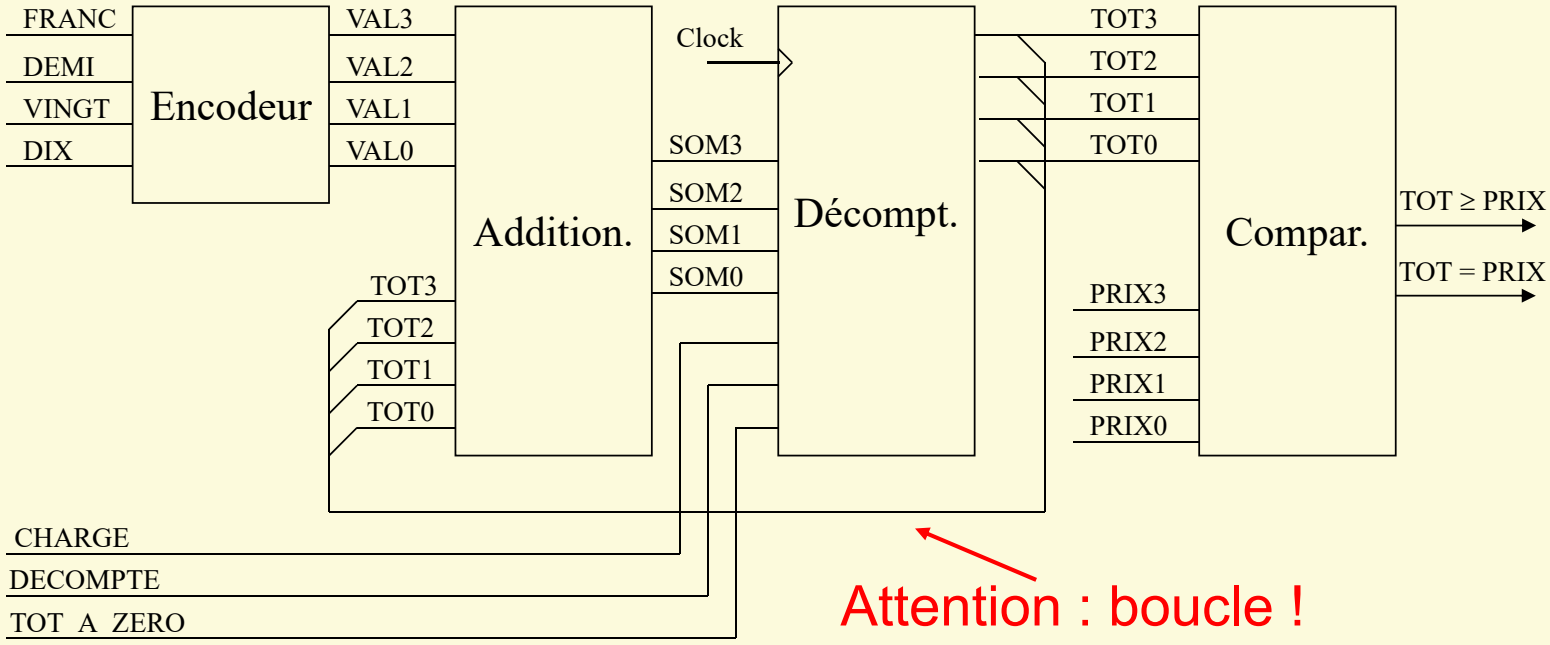
- Identification fonctions UT et actions UC dans l'organigramme => choix partition UC/UT



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
  - Addition => mémoriser résultat addtion
  - Soustraire 10 cts => décrémentation
- Additionneur 4 bits
- Comparateurs 4 bits

Schéma bloc de l'UT choisie



Réalisation avec fcts standard

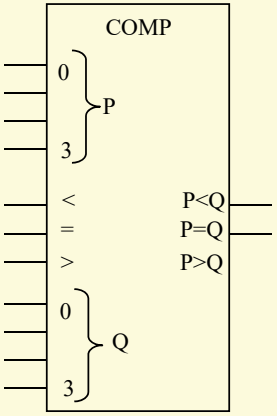
- Encodeur
  - Table de vérité (à compléter)

Franc	Demi	Vingt	Dix	Valeur
0	0	0	1	0001
0	0	1	0	0010
0	1	0	0	0101
1	0	0	0	1010
	autres			?

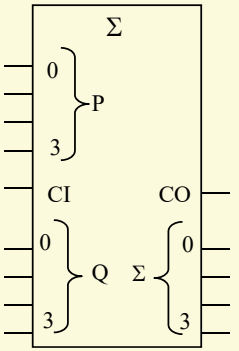
- Exercice: établir description VHDL optimale

Réalisation avec fcts standard

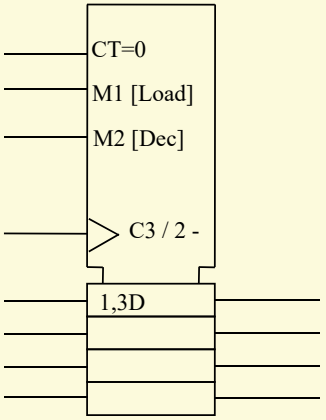
Comparateur 4 bits :

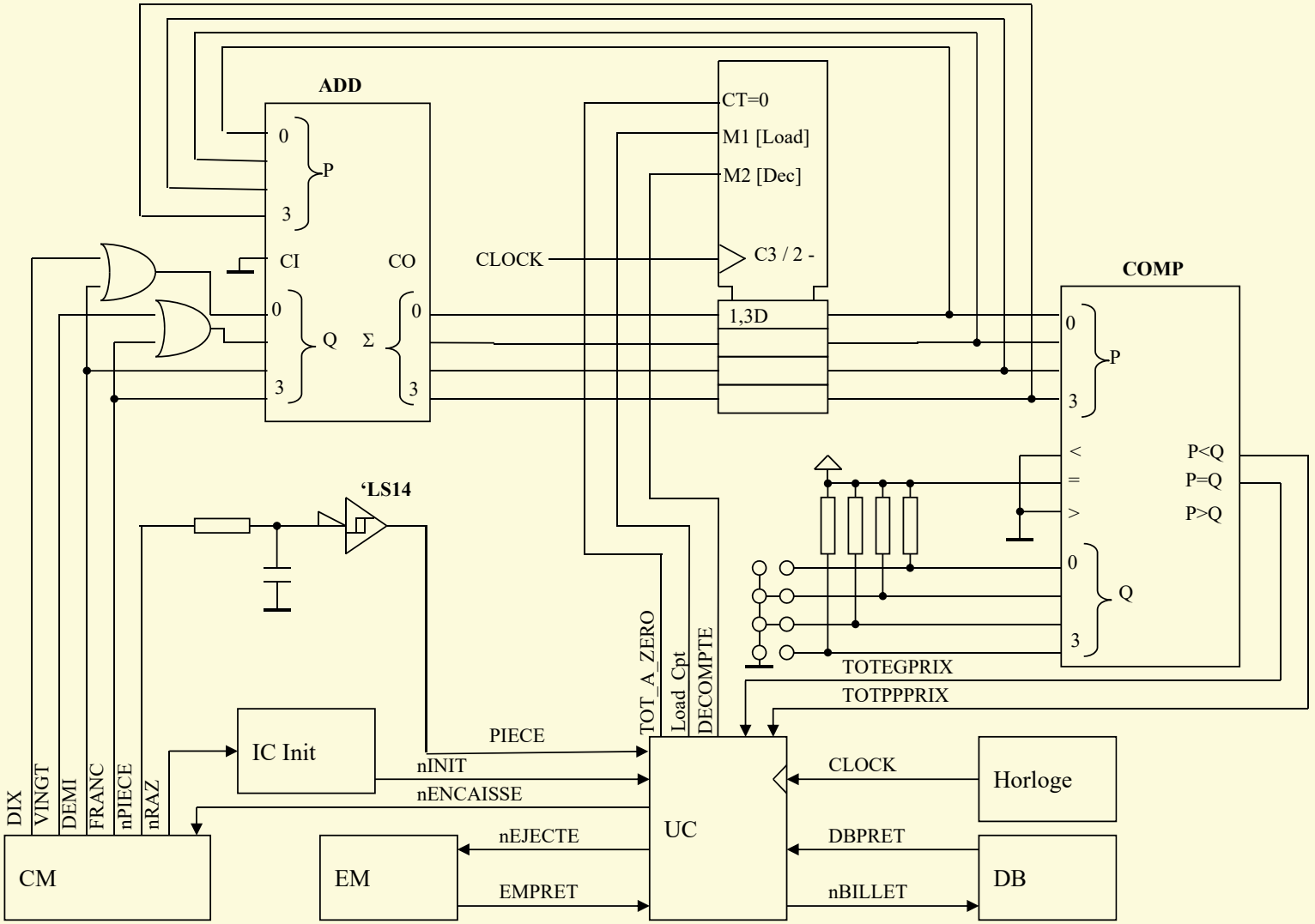


Additionneur 4 bits :



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




ADD
0
P
3
CI
CO
0
Q
Σ
0
3
CLOCK
CT=0
M1 [Load]
M2 [Dec]
C3 / 2 -
1,3D
COMP
0
P
3
<
=
>
P<Q
P=Q
P>Q
0
Q
3
'LS14
TOTEGPRIX
TOTPPPRIX
TOT_A_ZERO
Load Cpt
DECOMPTE
DIX
VINGT
DEMI
FRANC
nPIECE
nRAZ
IC Init
PIECE
nINIT
nENCAISSE
CLOCK
Horloge
UC
DBPRET
nBILLET
DB
CM
EM
nEJECTE
EMPRET

Exercices série II

II.1 Modifiez l'UT de la diapositive 77 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 Etablir une description optimale, en VHDL synthétisable, de l'encodeur.

II.3 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.4 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. Puis:

- Etablir le schéma de l'UT
- Etablir l'organigramme détaillé de l'UC
- Décrire le système en VHDL synthétisable (UT, UC et top)

Exercices série II

II.5 Modifiez l'organigramme grossier de la diapositive 71 et l'UT de la diapositive 77, de façon à calculer le montant payé par comptage plutôt que par addition. Puis:
- Etablir le schéma de l'UT
- Etablir l'organigramme détaillé de l'UC
- Décrire le système en VHDL synthétisable (UT, UC et top)

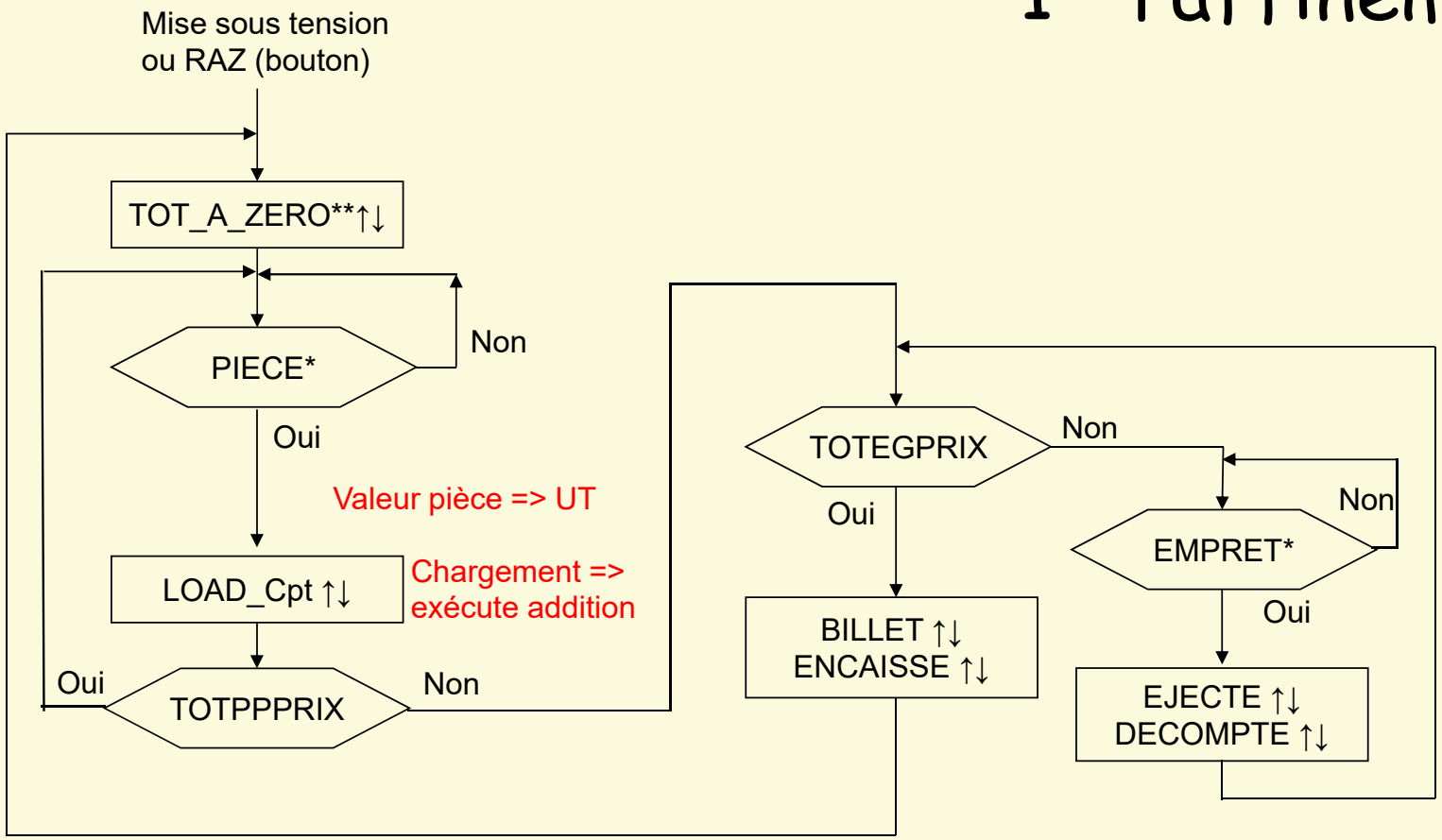
II.6 En faisant la synthèse des exercices précédents, réduisez l'UT spécialisée au strict minimum et adaptez (complétez) l'organigramme.

Organigramme détaillé

- Etablir l'organigramme avec **uniquement** les signaux connecté sur l'UC (entrées/sorties et signaux UT)
- 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.
  - Tenir compte du temps de réaction de l'UT
  - …

Organigramme détaillé de l'UC :

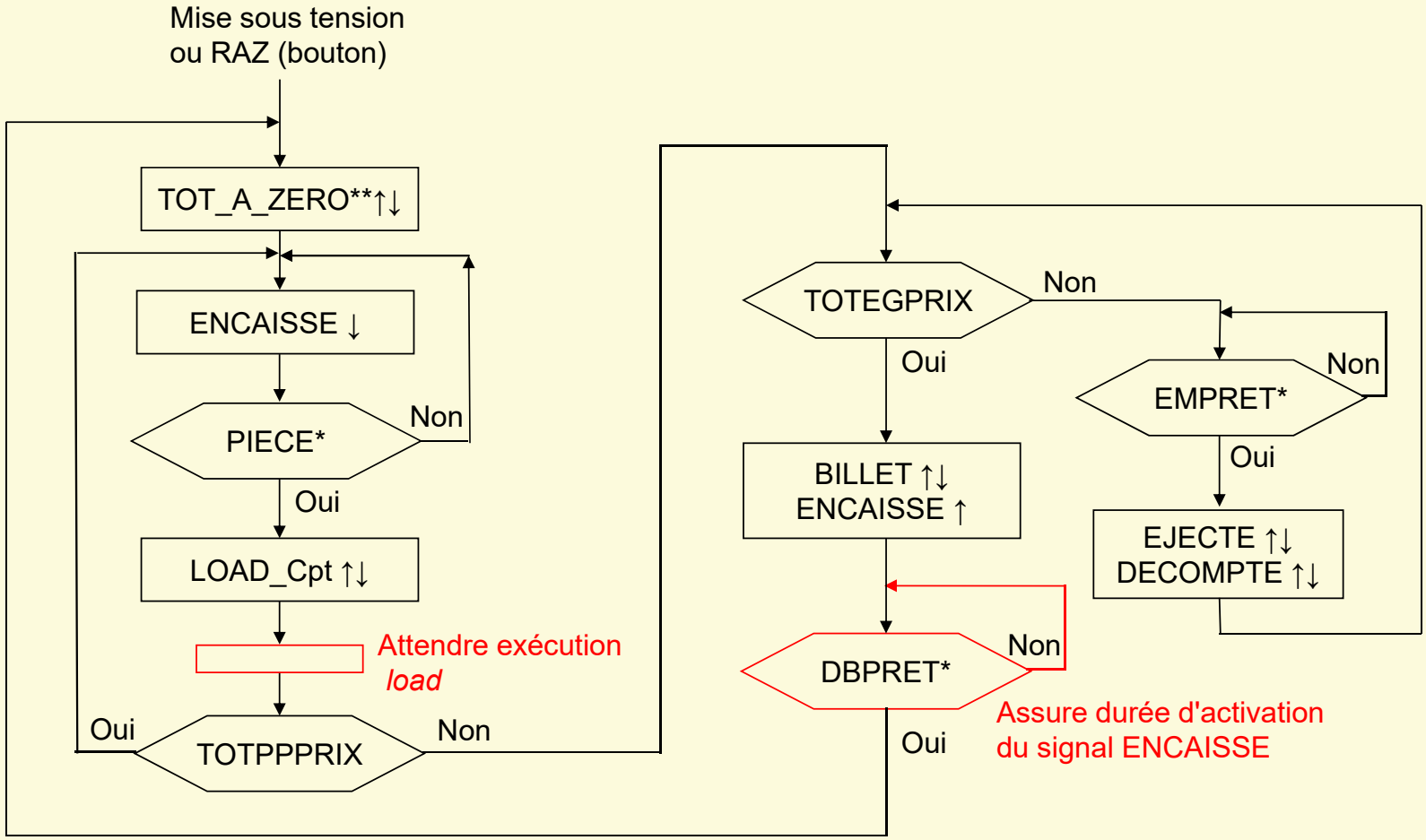
1er raffinement



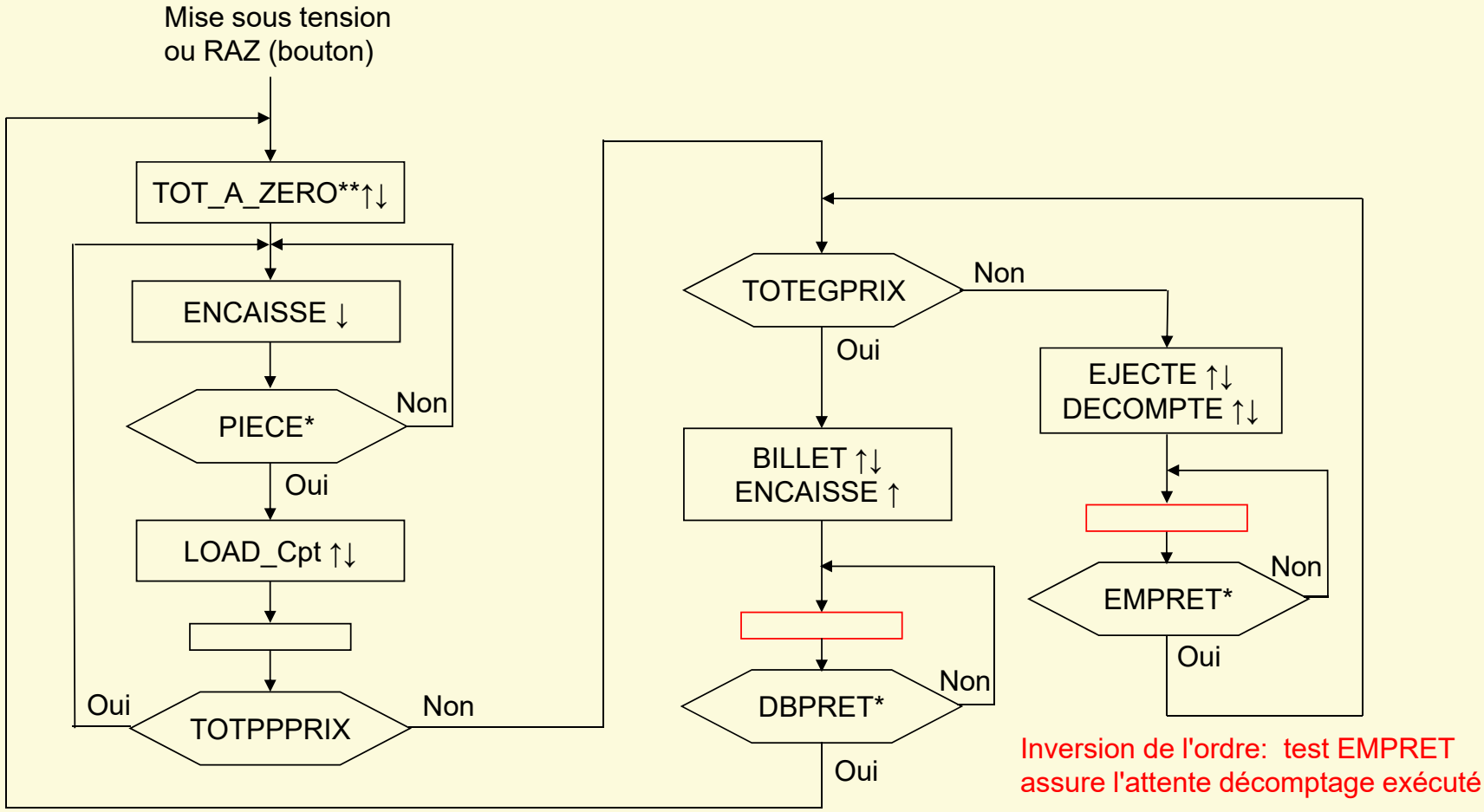
Organigramme détaillé 2ème raffinement

- 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



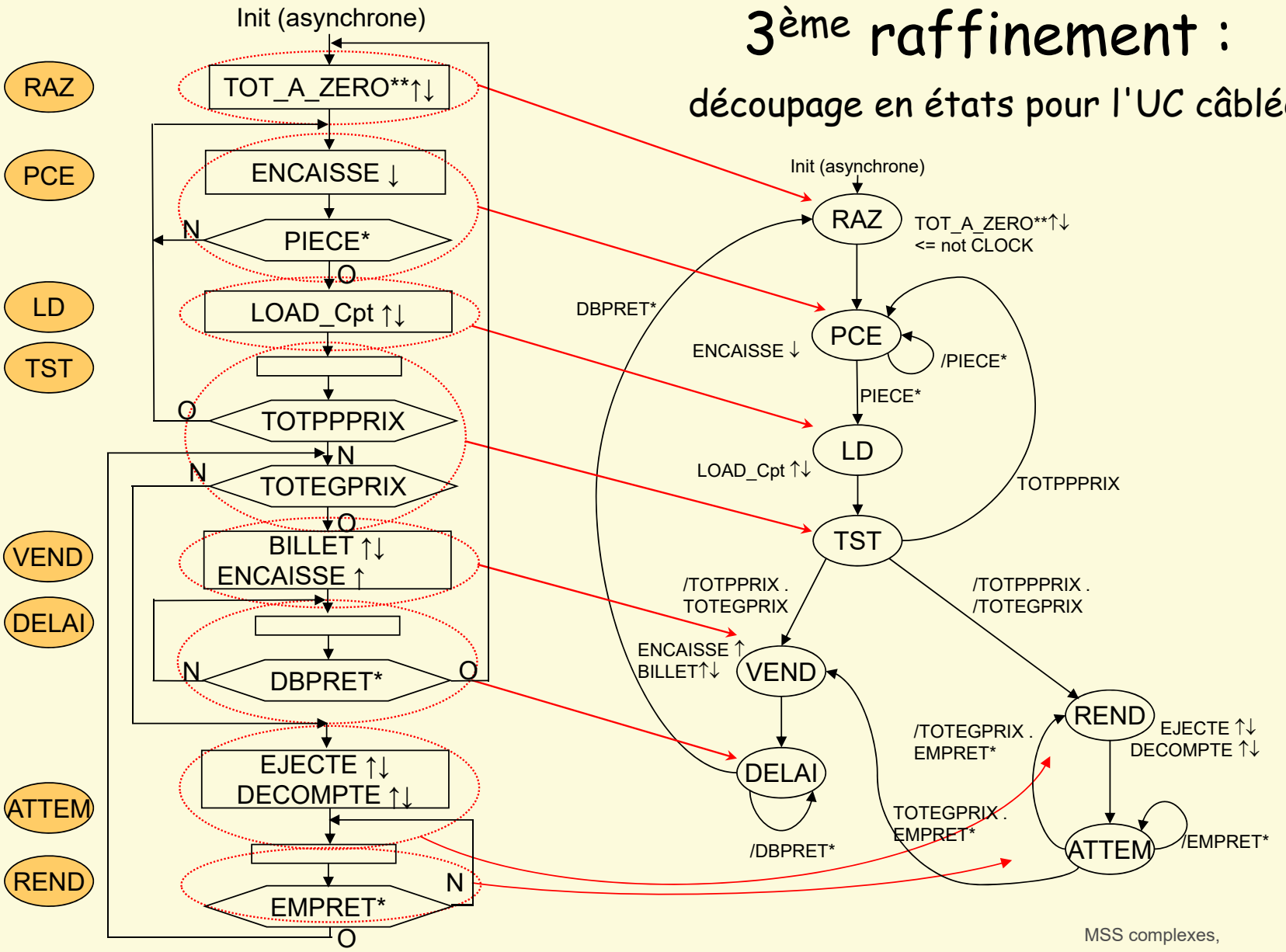
3ème raffinement : passage au graphe



Inversion de l'ordre: test EMPRET
assure l'attente décomptage exécuté

3ème raffinement :

découpage en états pour l'UC câblée



Exercices série III

III.1 Faites une description en VHDL synthétisable de l'UC câblée de la diapositive 86.

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 86 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 86 dans l'ordre numérique, avec 000 pour RAZ et 111 pour DELAI. Prenez un compteur 4bits (ayant les fcts load et Inc) comme générateur d'états. Etablissez la table de vérité des commandes à appliquer au compteur pour générer la séquence souhaitée.

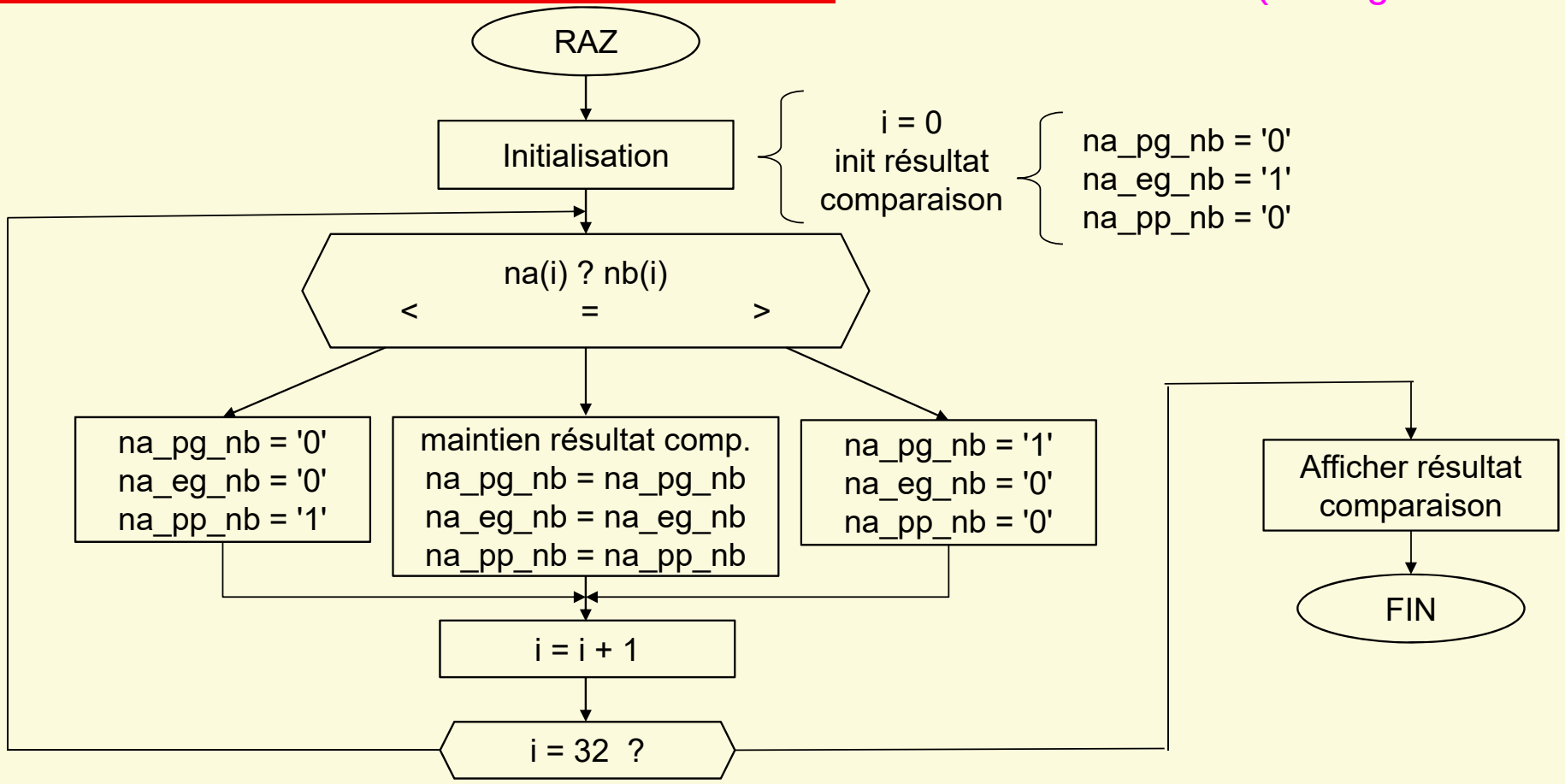
Exercices série III

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 85 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

Comparaison séquentielle, organigramme

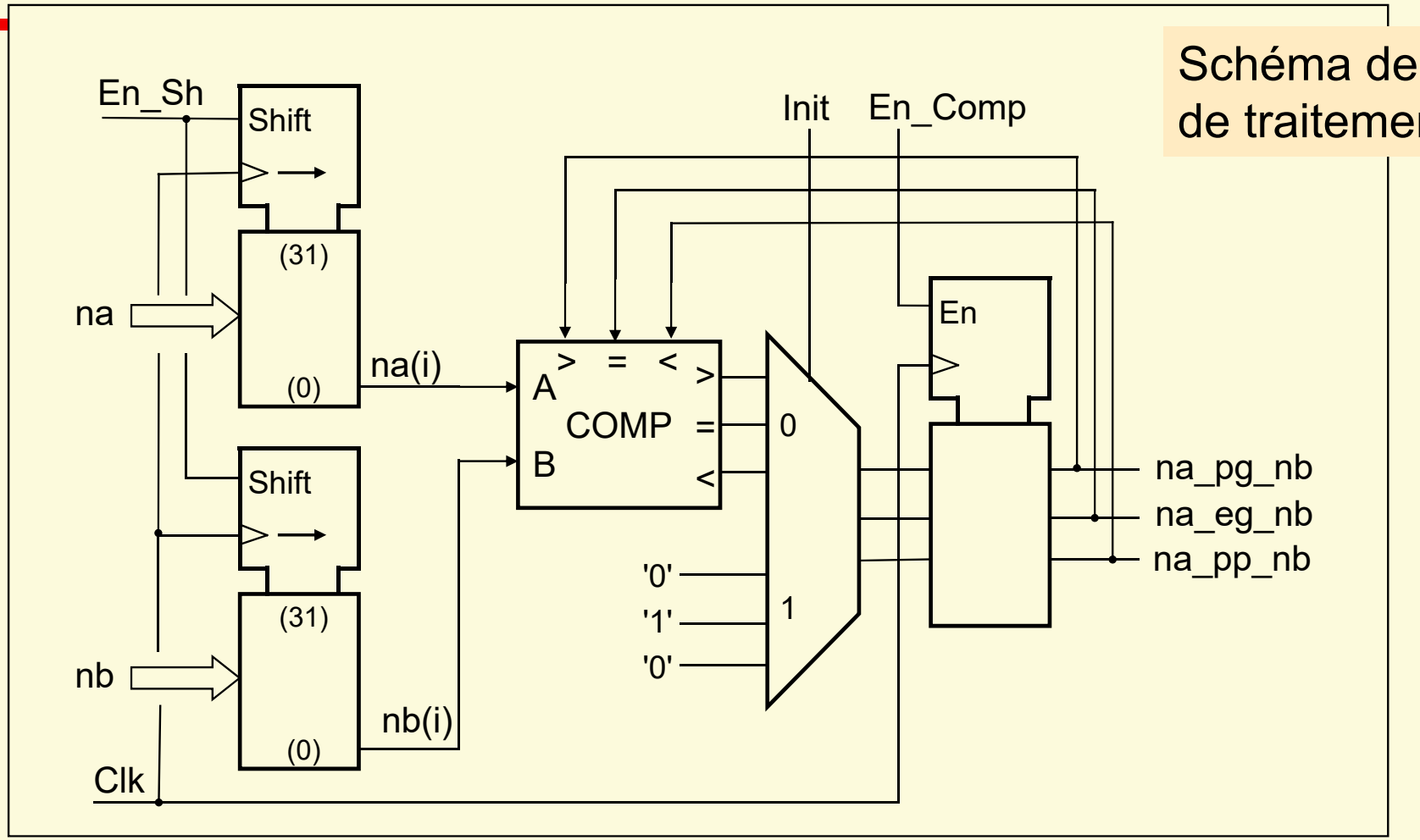
Solution (sans gestion d'un start)



Comparaison séquentielle, UT

Solution

Schéma de l'unité de traitement UT



Evolution vers UC microprogrammée

- Passage d'une UC câblée (graphe des états)

  vers une UC microprogrammée

Soit le passage vers le

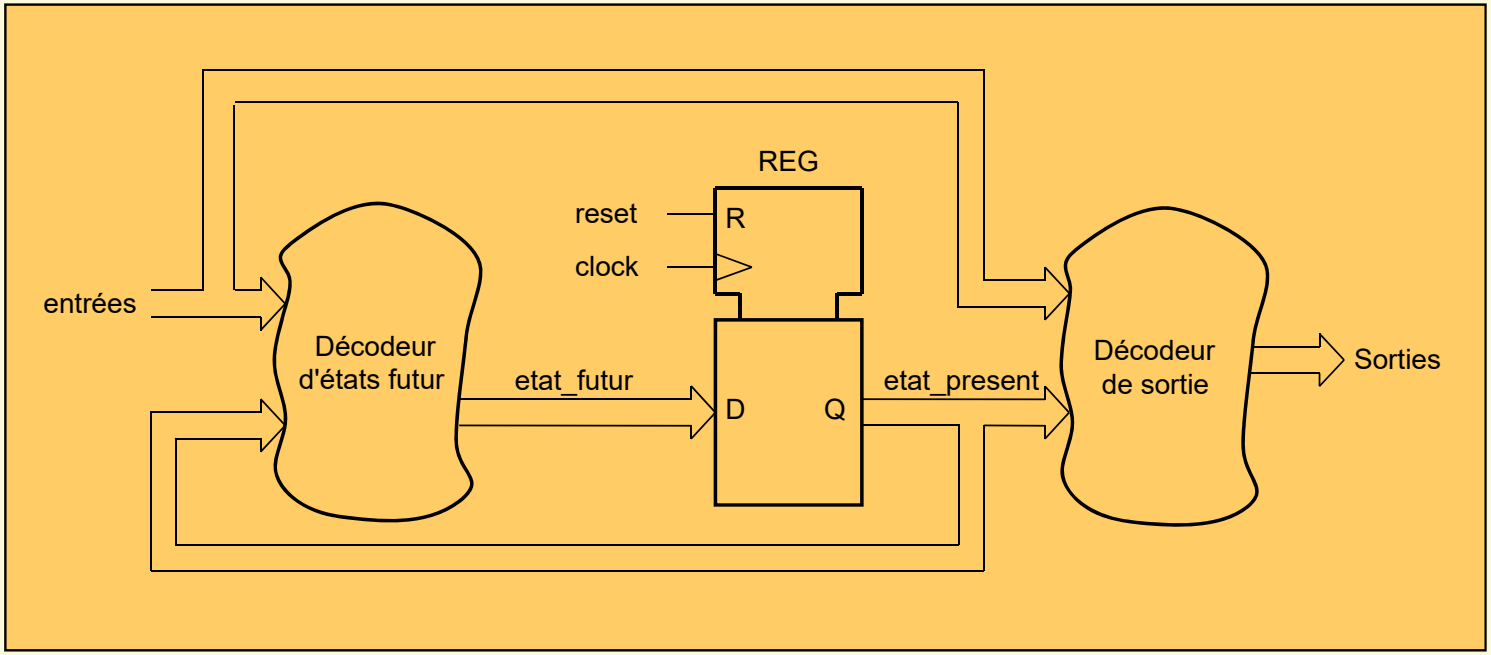
principe de l'architecture d'un processeur

Idées menant à une UC μprogrammée

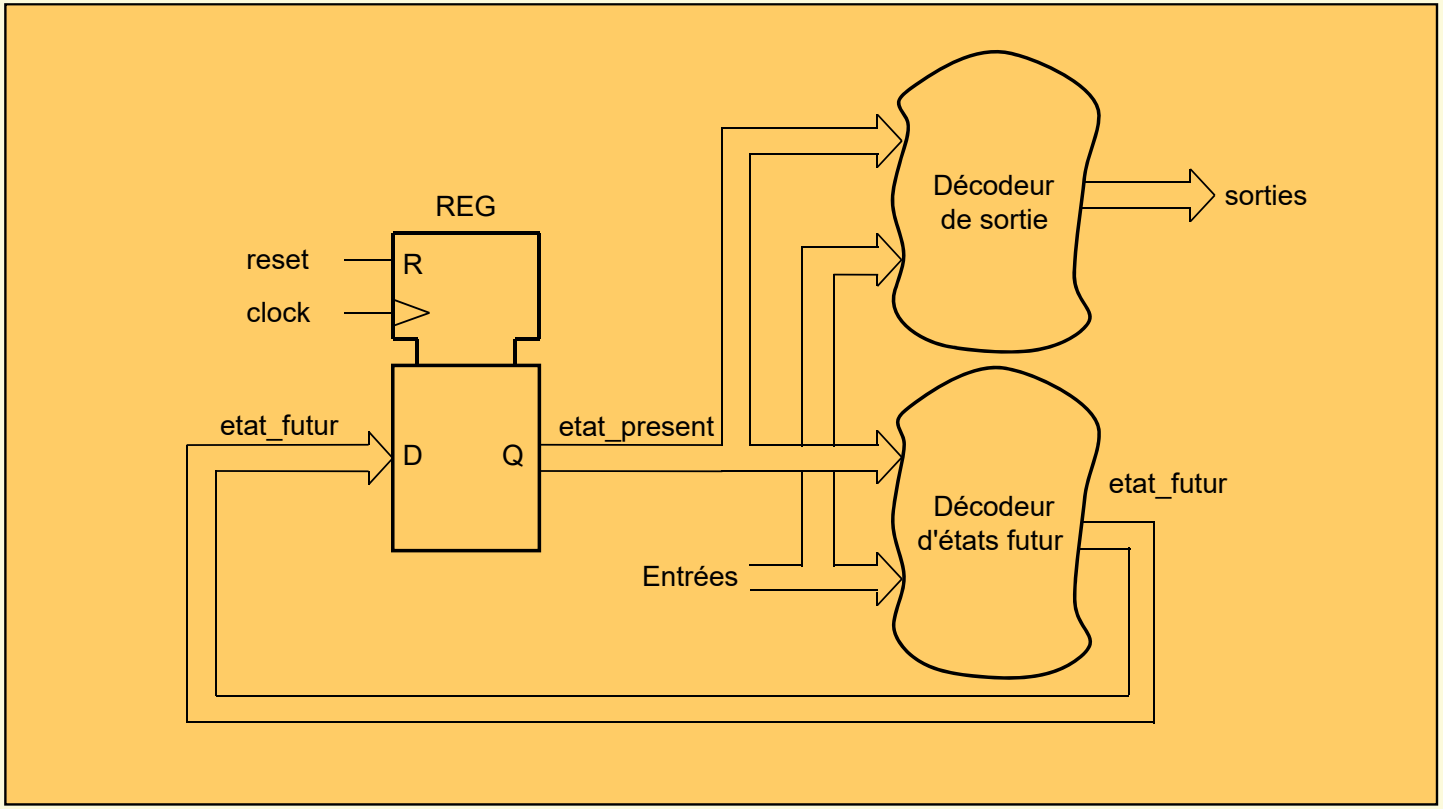
- Réaliser le décodeur d'état futur (décodeur des actions de séquencement) et le décodeur de sorties avec une ROM (mémoire)
  - conception facilitée (table de vérité)
  - réalisation modulaire
  - schéma standard
  - modifications faciles
  - haute intégration

Décomposition d'un système séquentiel

Décomposition d'une UC câblée. Cela correspond à la structure d'un système séquentiel, soit :

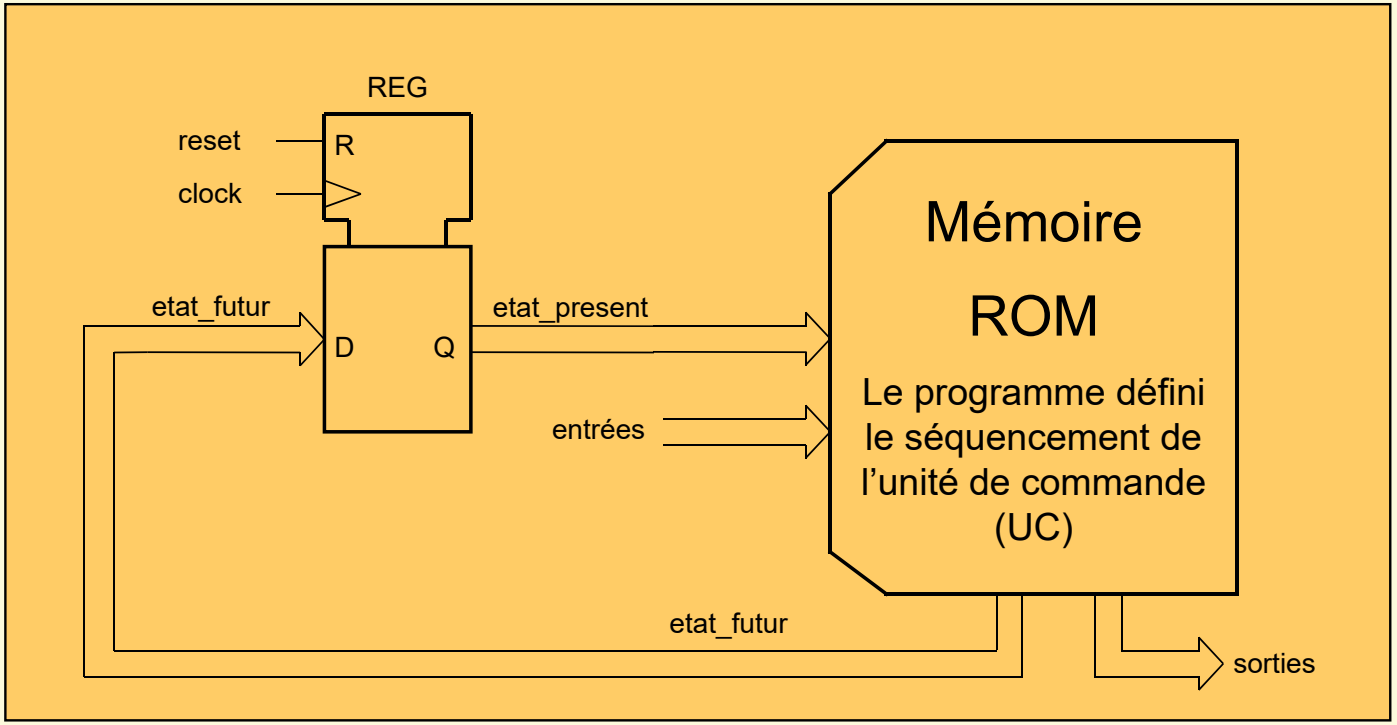


Autre représentation structure UC câblée



Utilisation d'une mémoire ROM

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

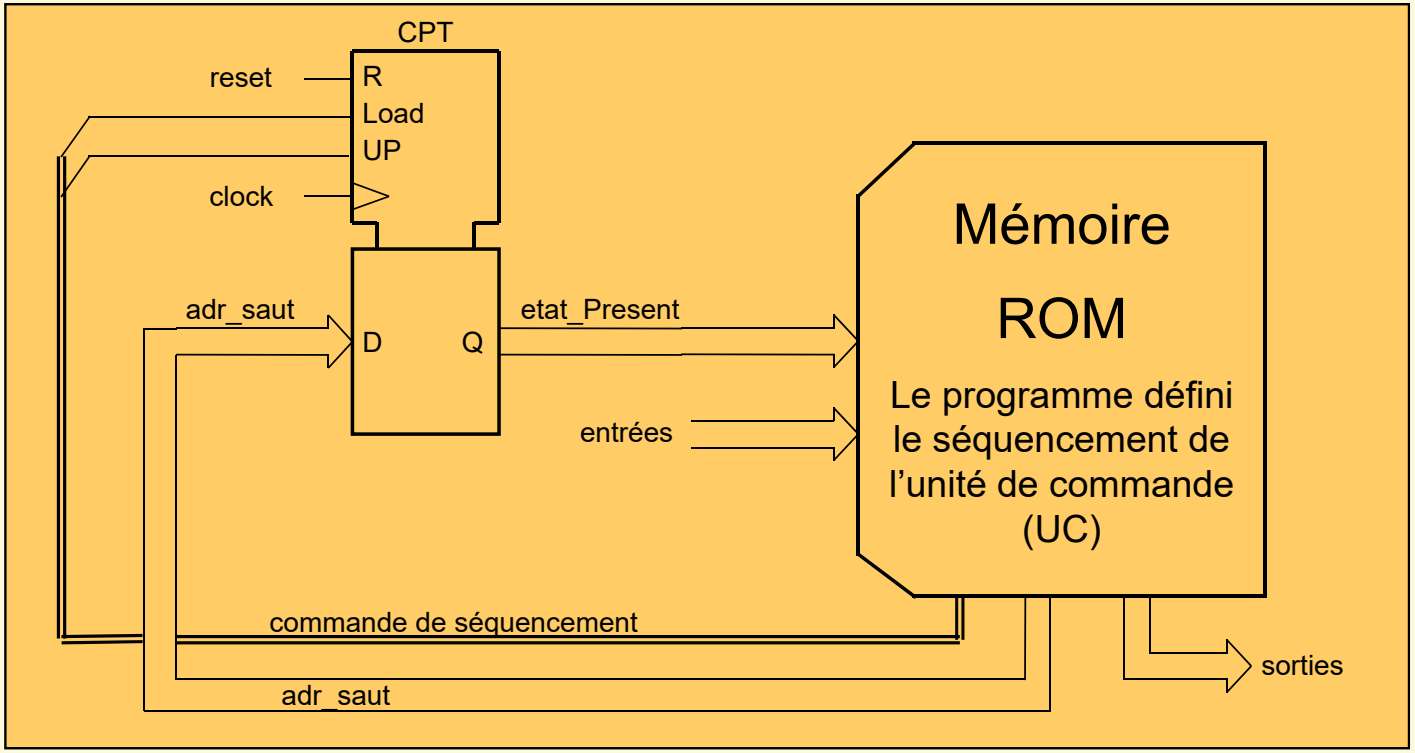


Idées menant à une UC programmée

- 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

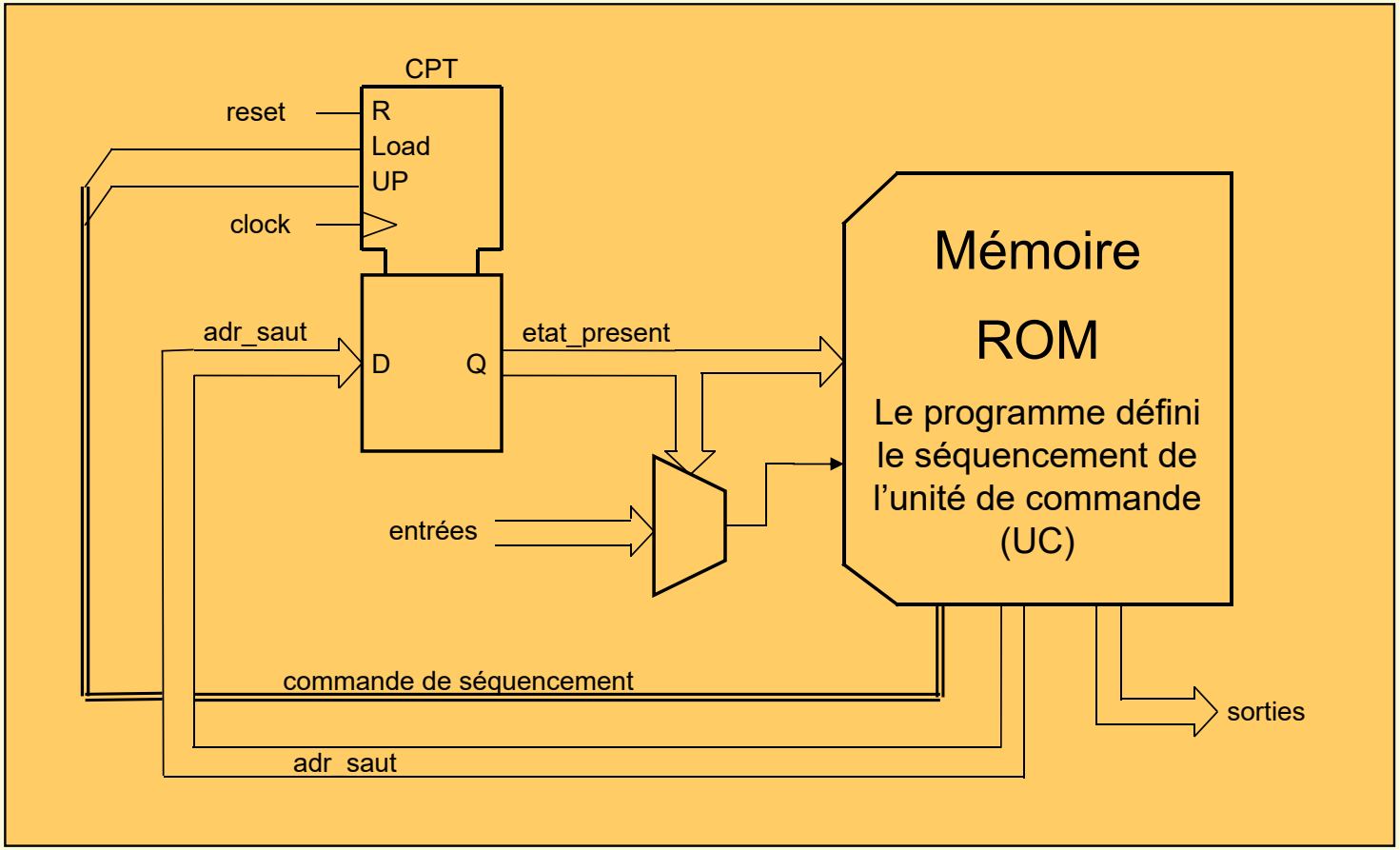
Si les états sont codés avec des codes représentants des valeurs croissantes : l'utilisation d'un compteur est préférable (incrém.)



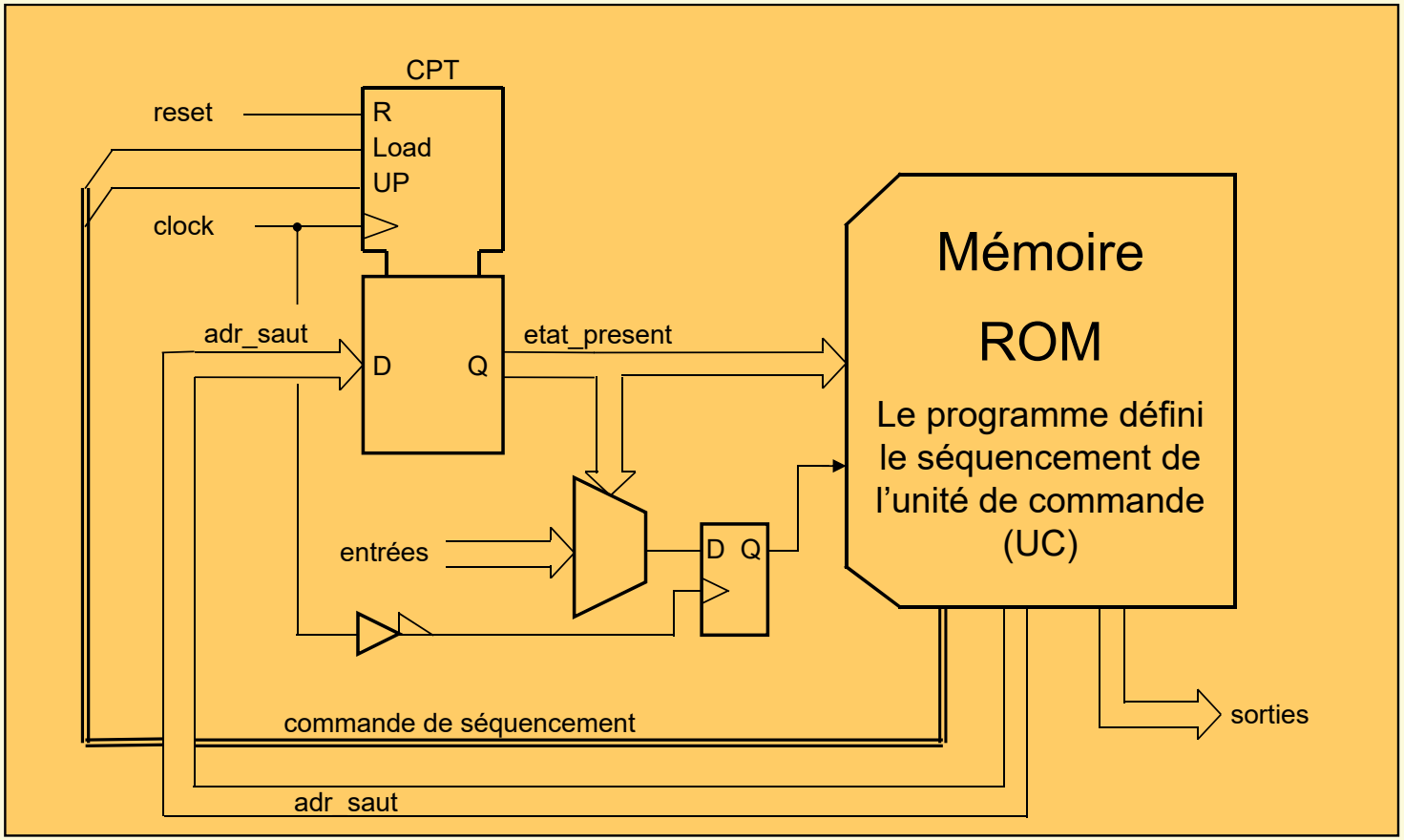
Idées menant à une UC μprogrammée

- 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, permet de diminuer la taille de la ROM
  - diminue le nombre de bascules de synchronisation
  - modulaire

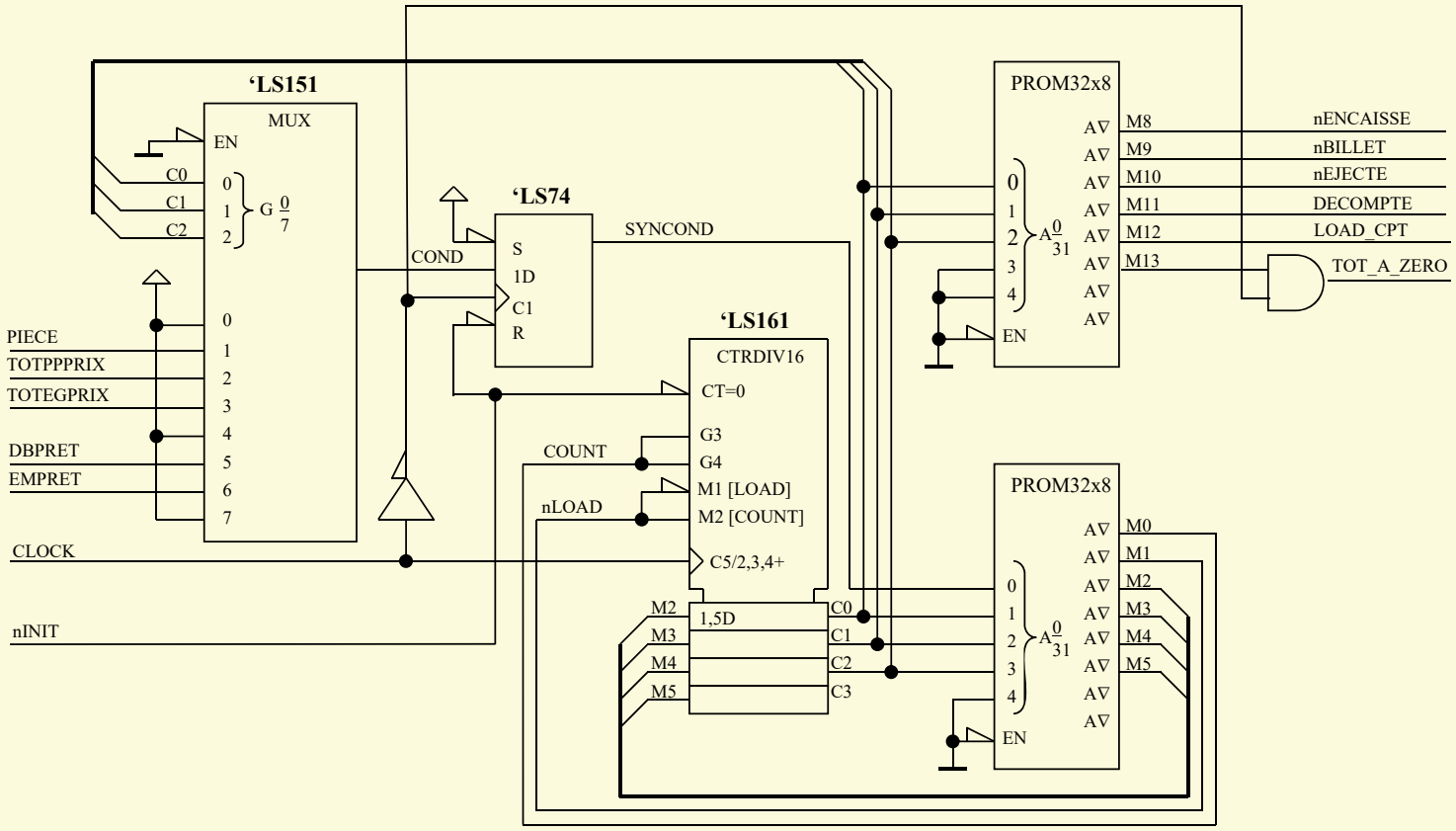
Diminution nbr adresses mémoire



Respect règle entrée asynchrone : obligatoire de synchroniser



1ère UC µprogrammée pour le vendeur de billets



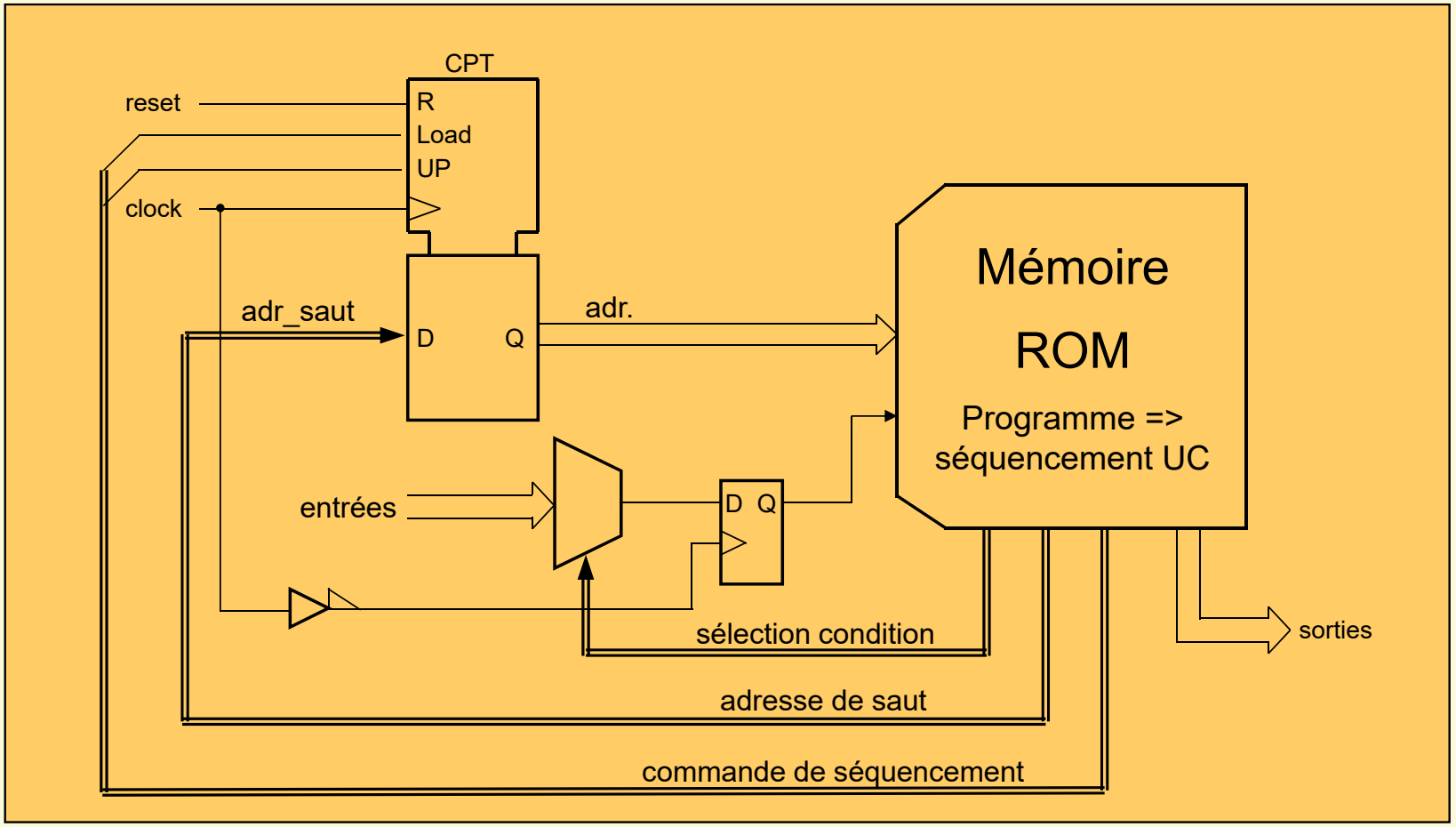
Inconvénients 1ère structure UC μprogr.

- Généralement :
  nb. d'états (μinstructions) >> nb. d'entrées (conditions distinctes)
  → mux d'entrées >> que nécessaire
- Une **modification du μprogramme** va impliquer 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

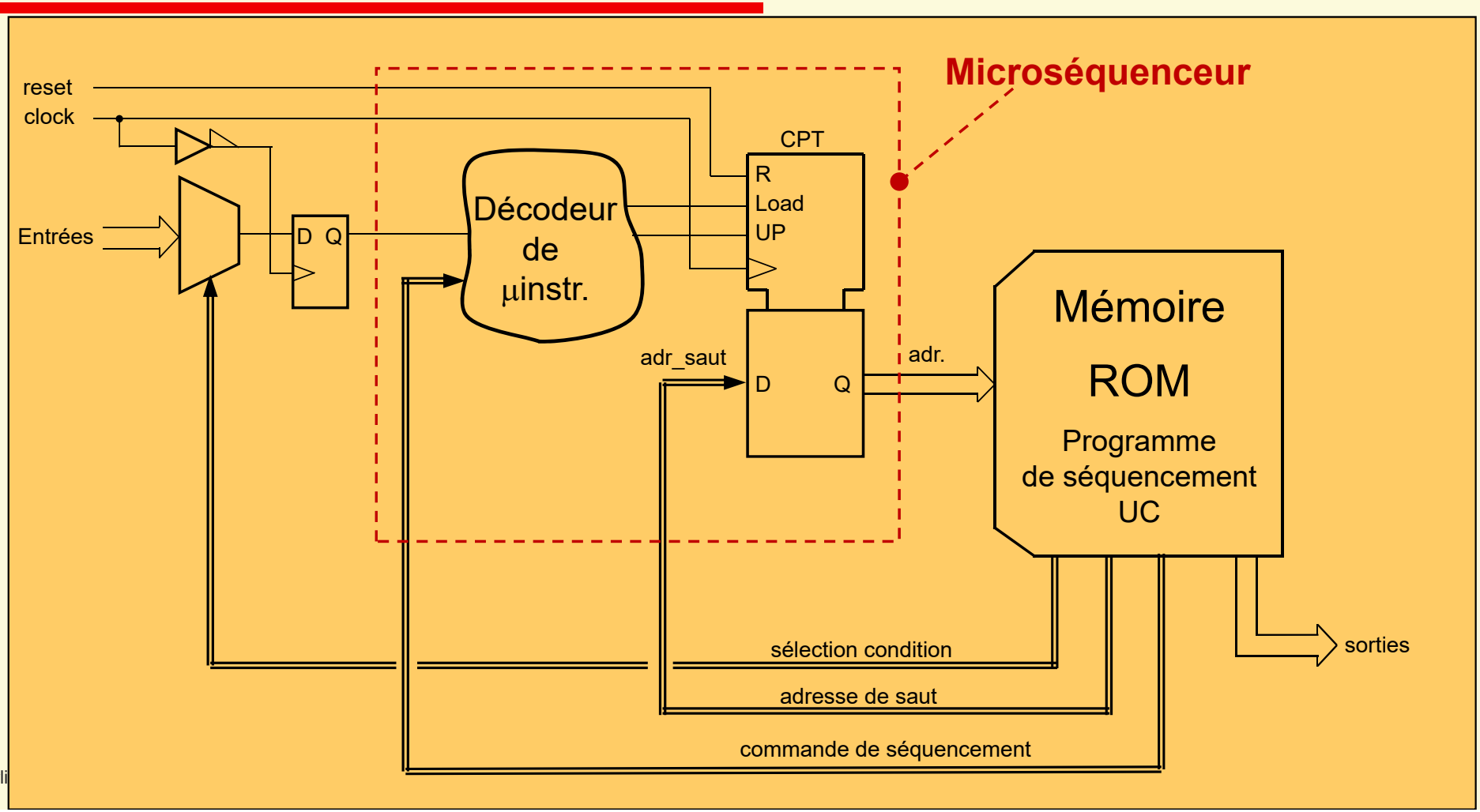
UC avec µprogr. horizontale



Améliorations

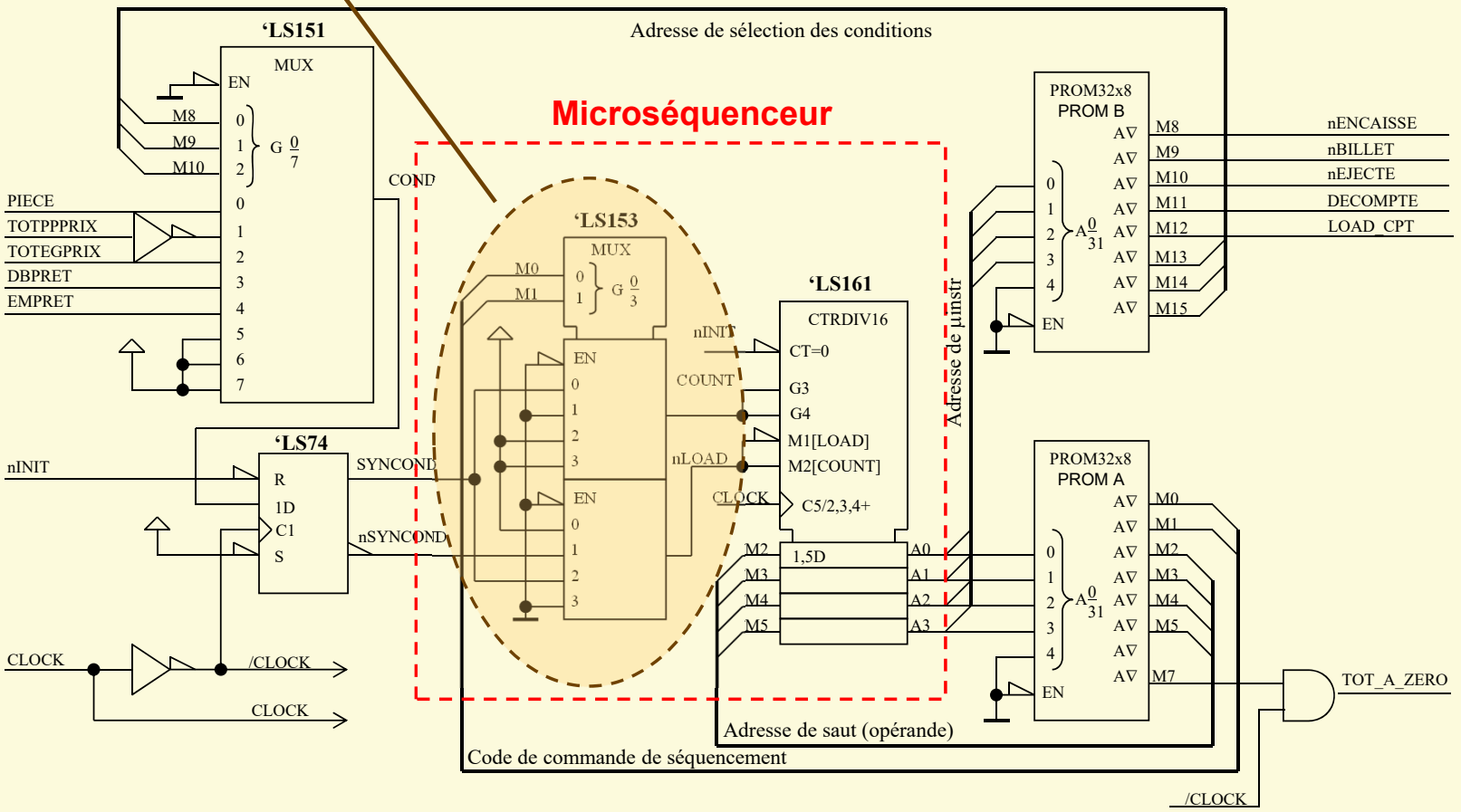
- Décomposer le décodeur d'actions de séquencement en 2 parties :
  - une partie fixe qui commande le compteur à partir d'un code d'opération et de la condition (combinée au compteur → μséquenceur)
  - une partie variable qui fournira le code de l'opération et qui sera placée dans la ROM

2ème UC avec µprogr. horizontale



2ème UC µprogrammée pour le vendeur de billets

Micro décodeur d'instruction



Commandes du µséquenceur

- Exemple de micro-jeu d'instruction

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

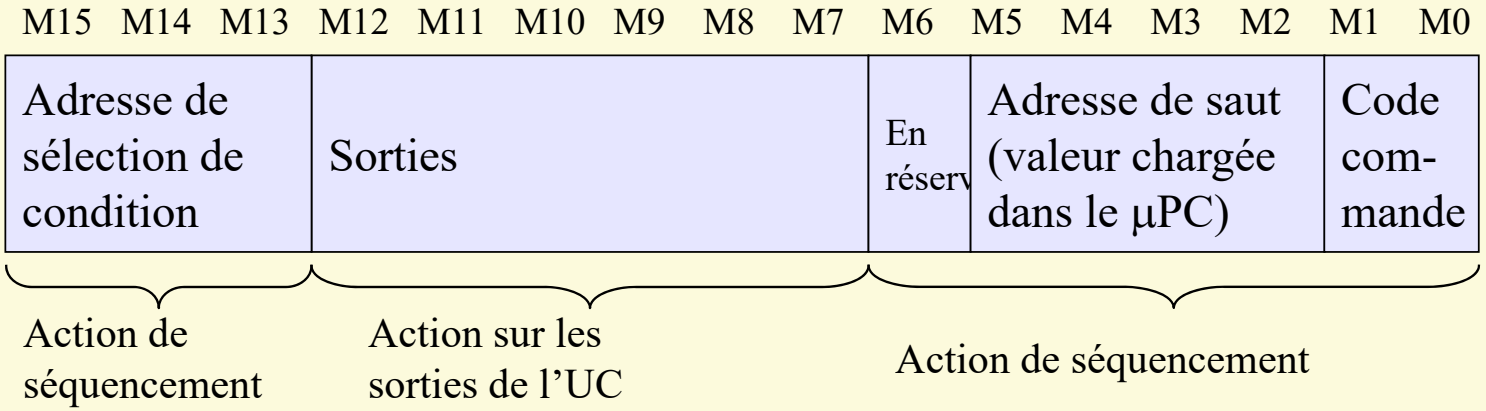
Table de vérité du décodeur de commandes

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)

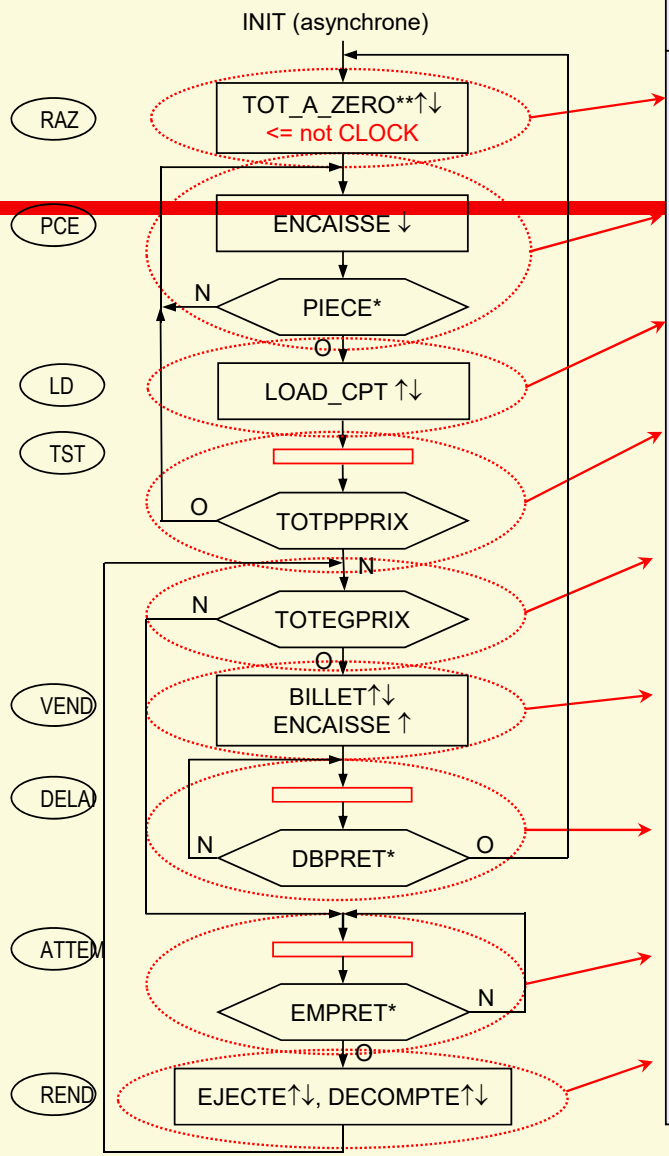
… format d'une μinstruction

- μprogrammation horizontale : chaque plage du format contient un seul type d'information
- 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 page 119 et des µinstructions page 120



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

Contenu de la présentation

- Programmation horizontale !
- Optimisation de la mémoire de μprogramme

  => μprogrammation verticale

  - Exemple : μséquenceurs à un seul opérande
  - Limitation des adresses de saut
  - Codage des plages
  - Nanoprogrammation
  - Codage des sorties

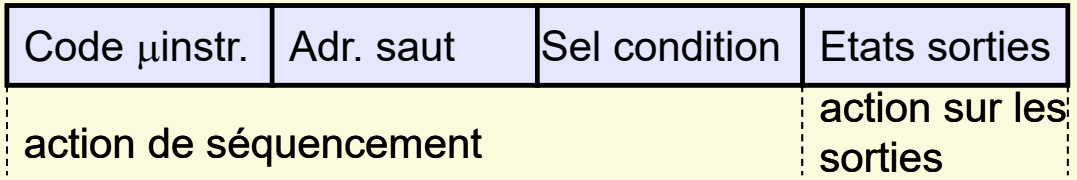
Microprogrammation horizontale

- 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

Nous avons vu un premier format, soit :

- 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

Code µinstr.	Opérande

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

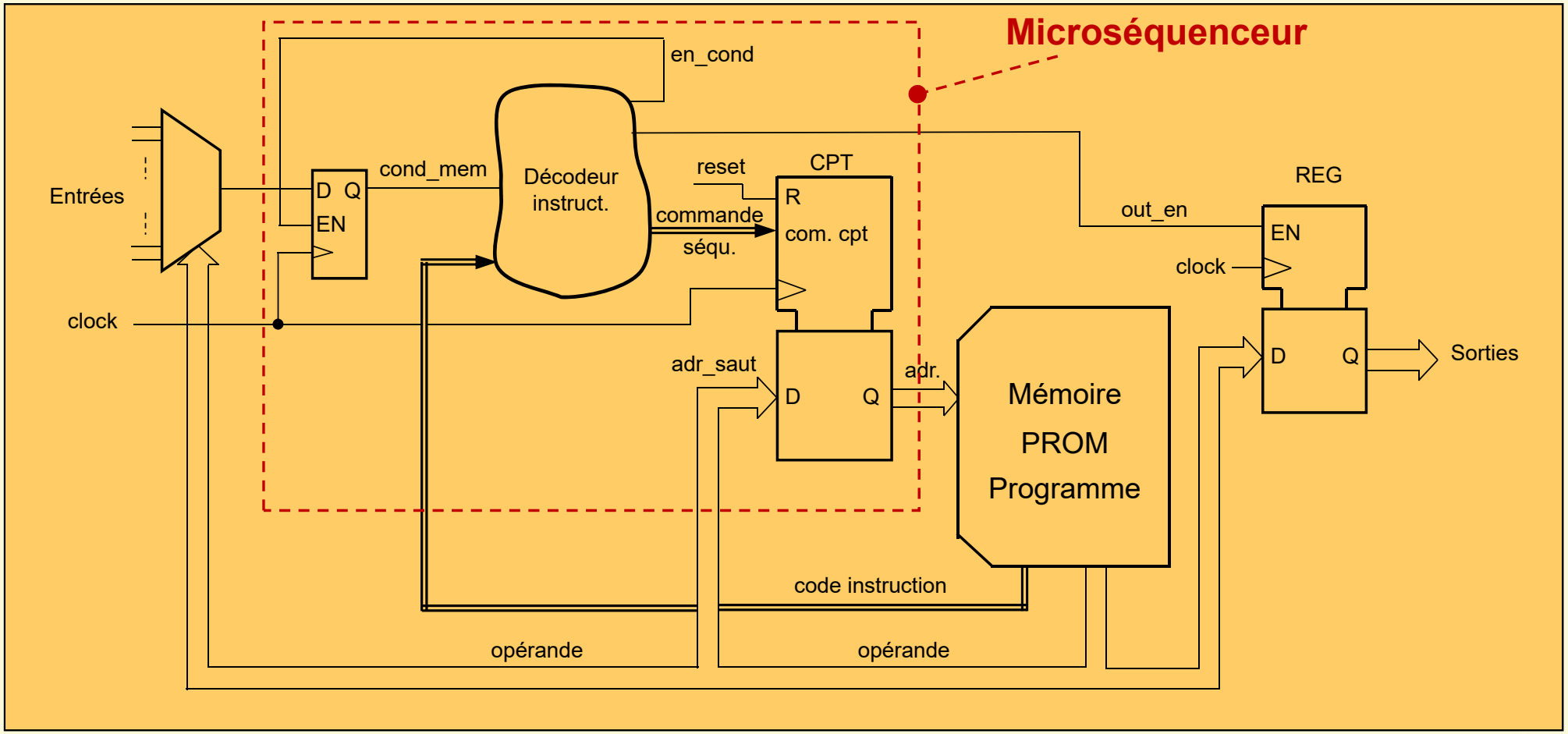

Il faudra mémoriser l'état des sorties


L 'opérande a plusieurs fonctions

Exemple : μséquenceurs à un seul opérande

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

UC µprogr. à un seul opérande

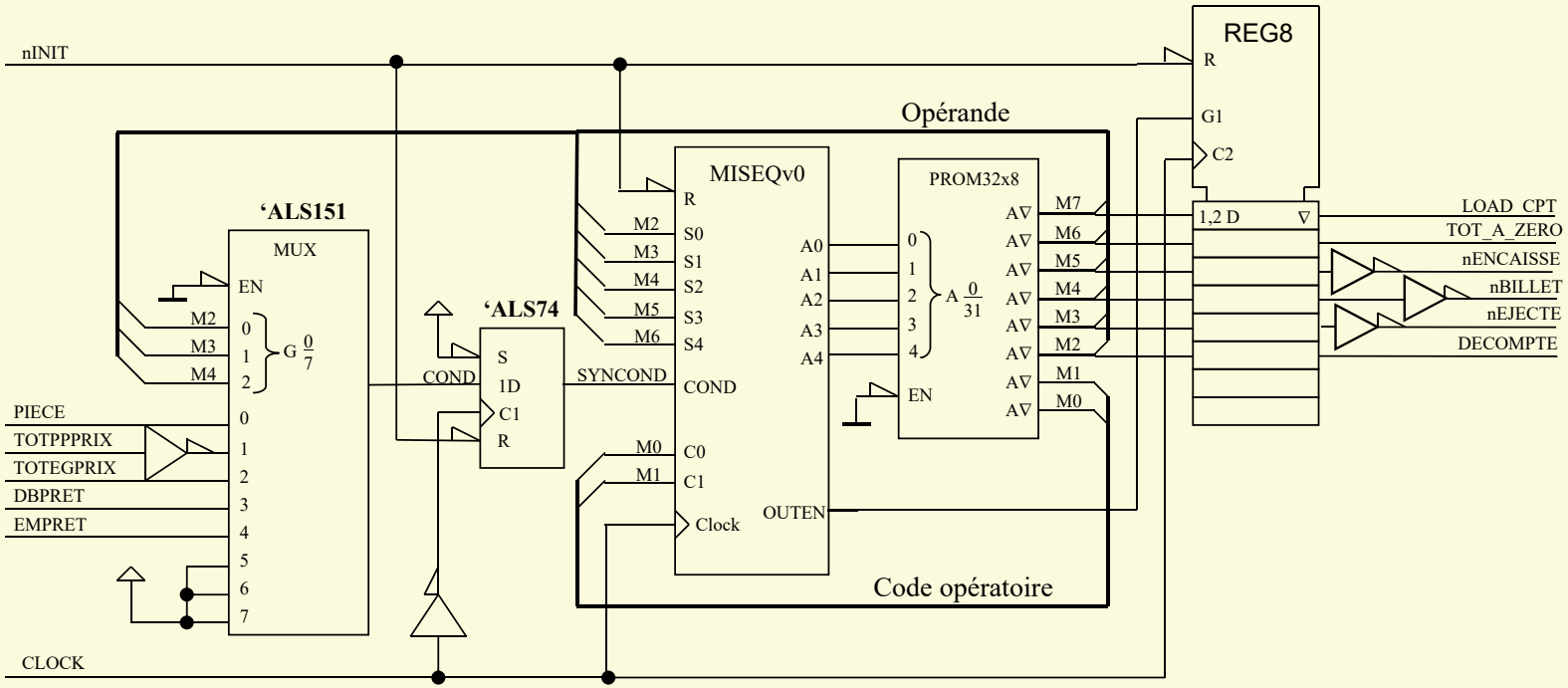


MISEQv0 : µséquenceur à un seul opérande

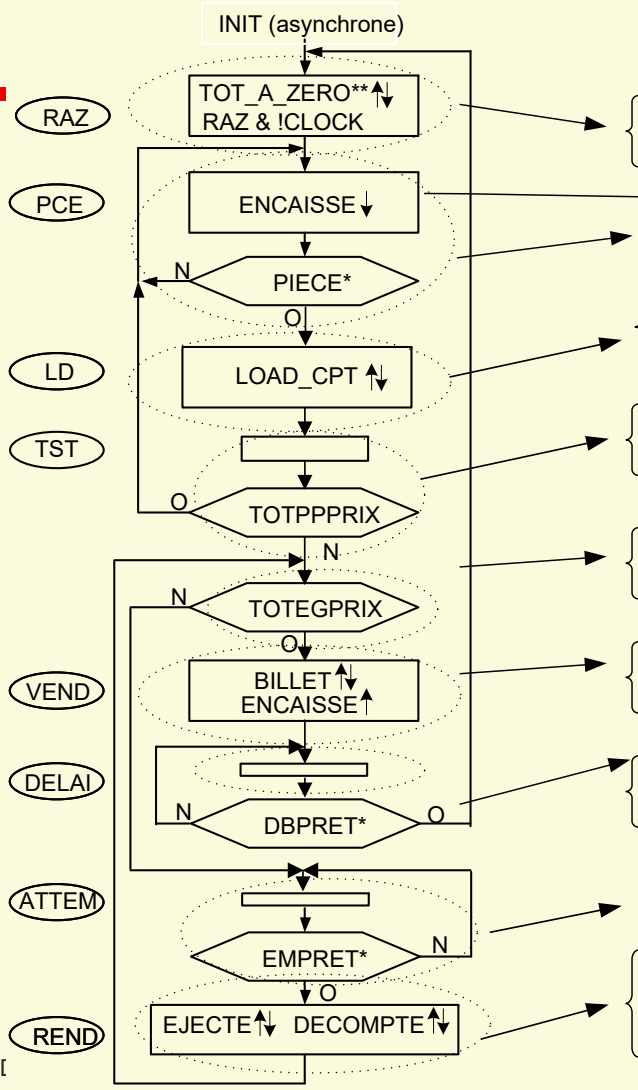
- Jeu de µinstructions MISEQV0 :

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

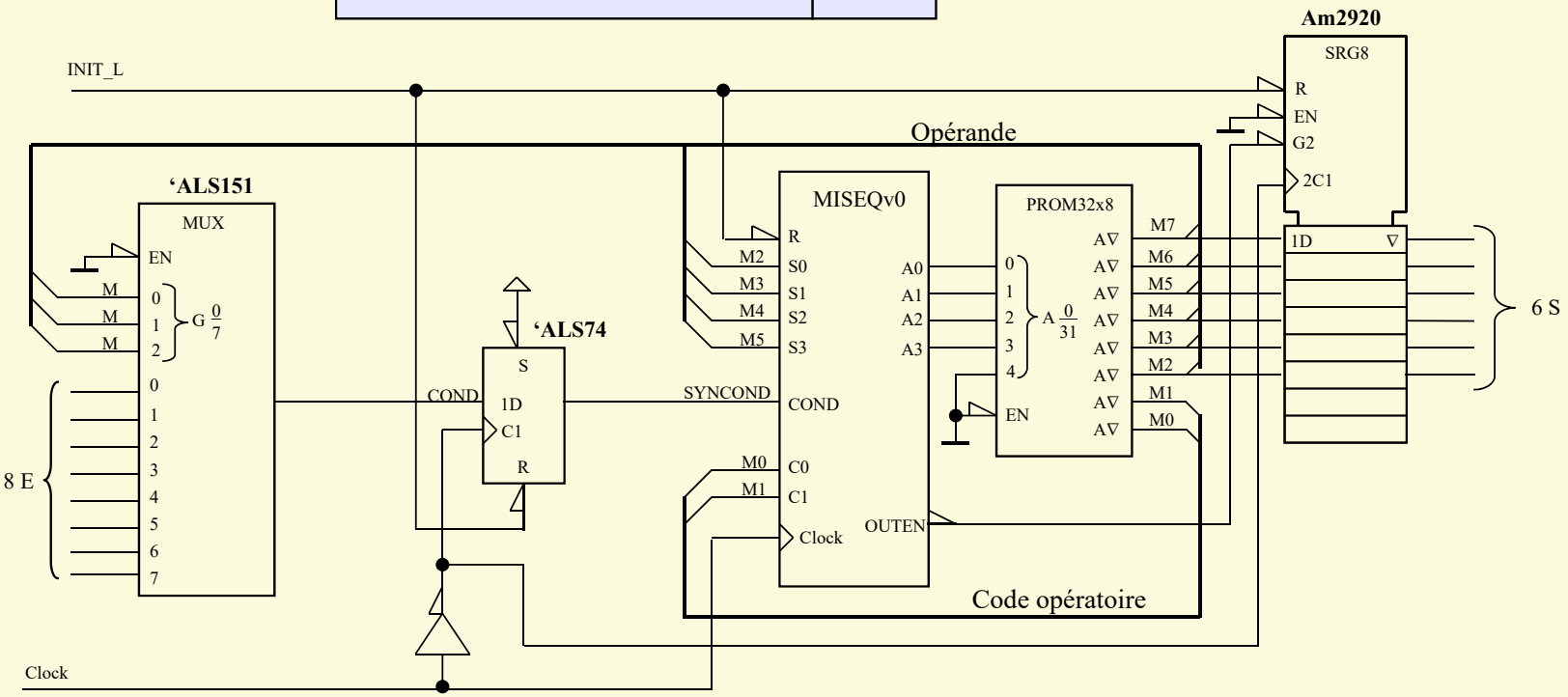
Exercices série V

V.1 Ajoutez un sélecteur de la polarité de la condition dans le schéma de la diapositive 124. Proposez un format pour les μinstructions WAIT et SKIP de MISEQv0, qui inclue le choix de la polarité.

Solution ajout polarité au MISEQv0

Exe V.1

	7 … 2	1 0
WAIT		0 0
SKIP		1 0



Limitation des adresses de saut

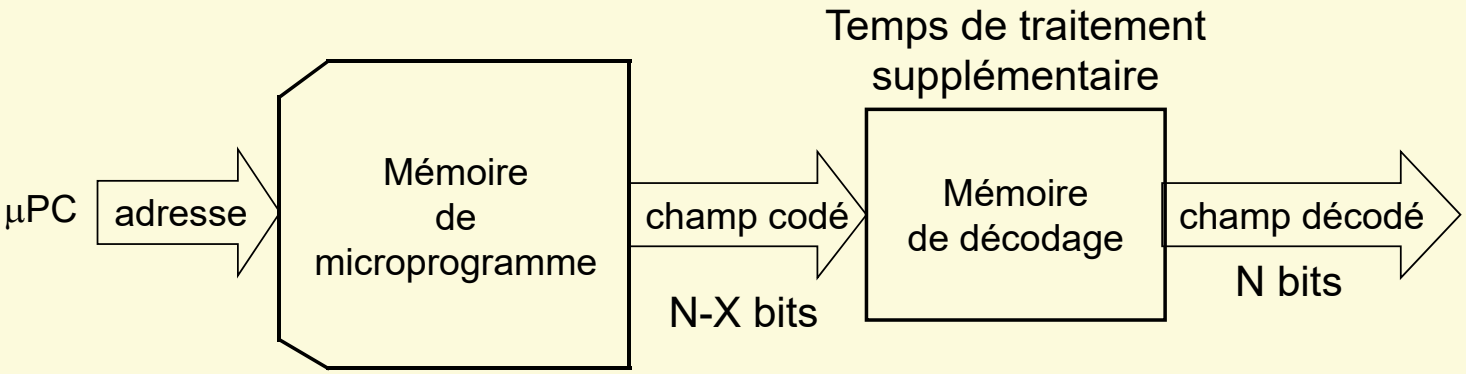
- Pour utiliser moins de bits dans les adresses de saut (opérande d'un saut)
  - ne sauter que vers certaines adresses (par ex. seulement les adresses paires)
  - utiliser l'écart entre l'adresse de départ et celle d'arrivée (adressage relatif), en le limitant
  - séparer l'adresse complète en 2 composantes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

Codage des champs

- Si le nombre de valeurs différentes utilisées par un champ est inférieur aux nombre combinaisons binaires, soit : nbr valeurs champ N bits << 2**N
  - coder ces valeurs sur N-X bits
  - X bits de moins dans la largeur des μinstructions
  - nécessite de décoder le champ à la suite de la mémoire
  - augmente le temps de propagation (diminue la fréquence d'horloge maximum)

Principe codage d'un champ

- Schéma de principe du décodage d'un champ codé



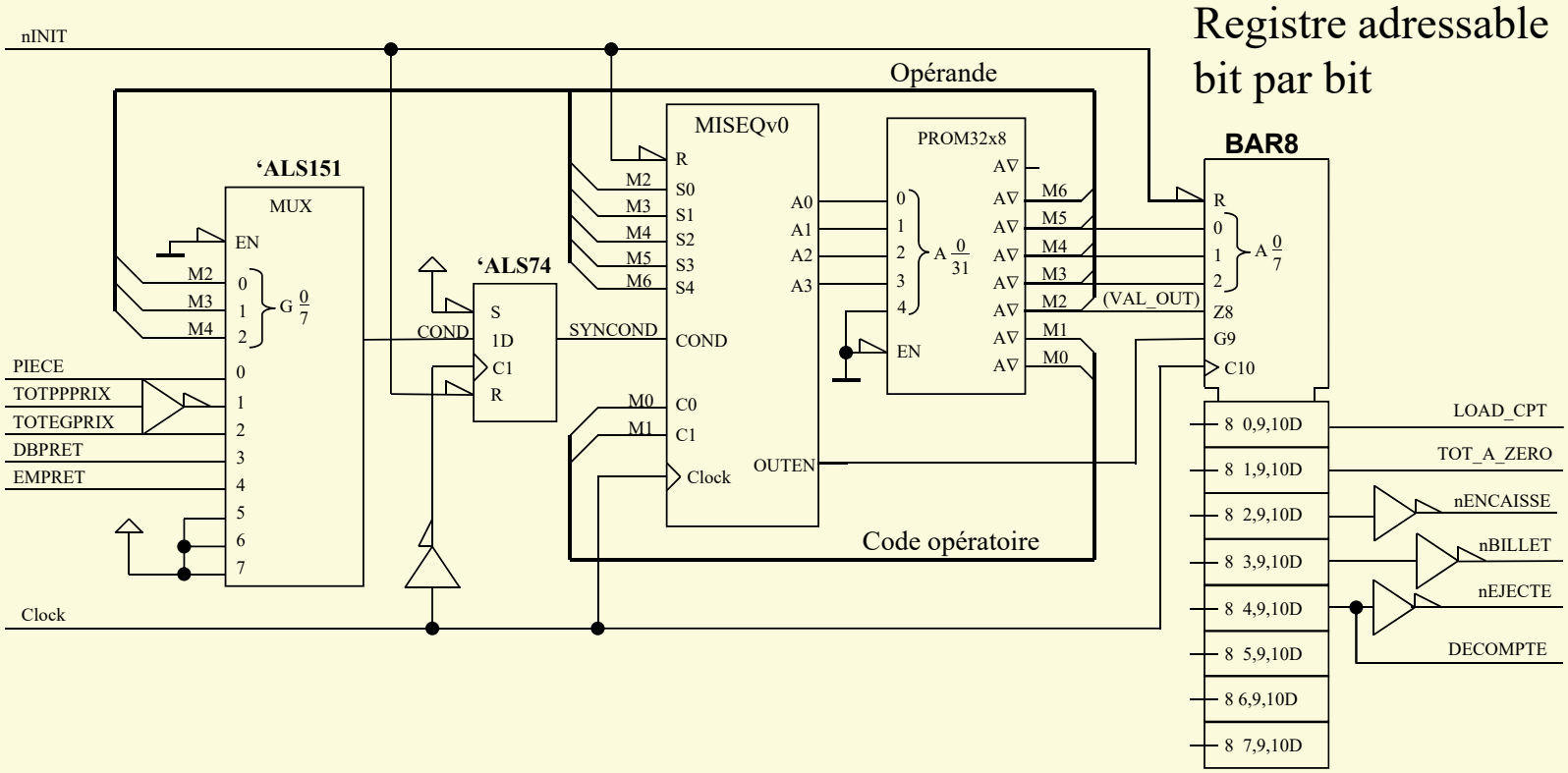
Codage des sorties

- 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)

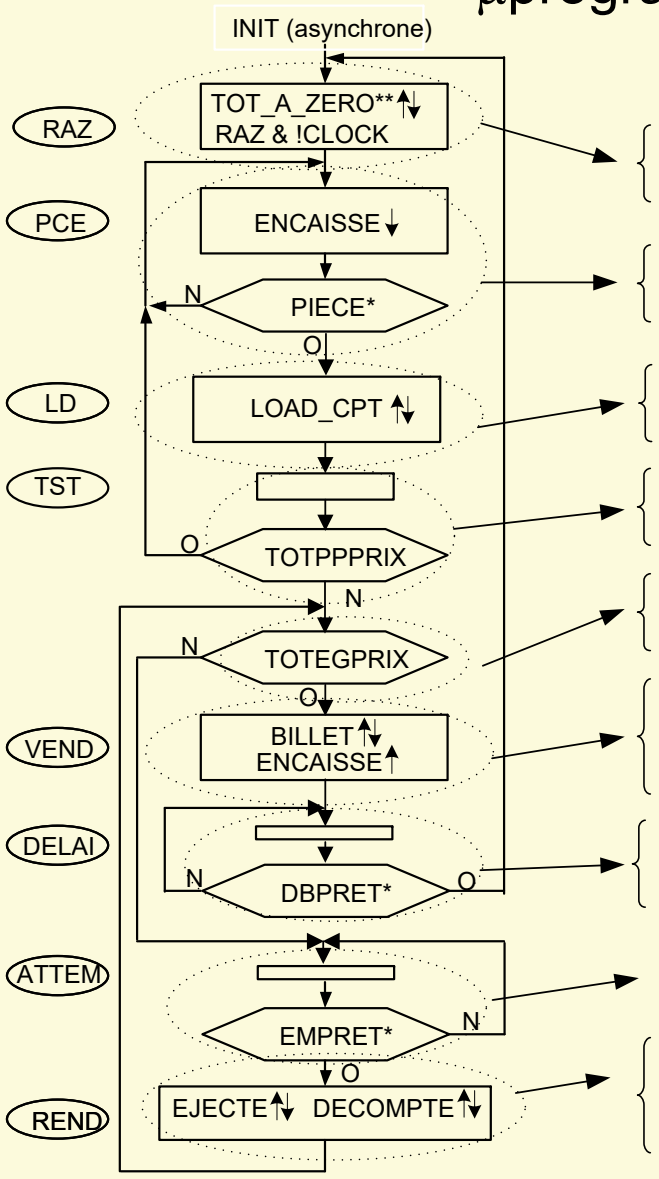
Exemple : affectation d'une seule sortie à la fois avec MISEQv0

- L'opérande pour la μinstruction OUT peut être composé de :
  - 1 bit pour la valeur à affecter
  - n bits pour désigner la sortie concernée (code, adresse)
  - inconvénient : modification d'une seule sortie simultanément
- 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 124



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

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

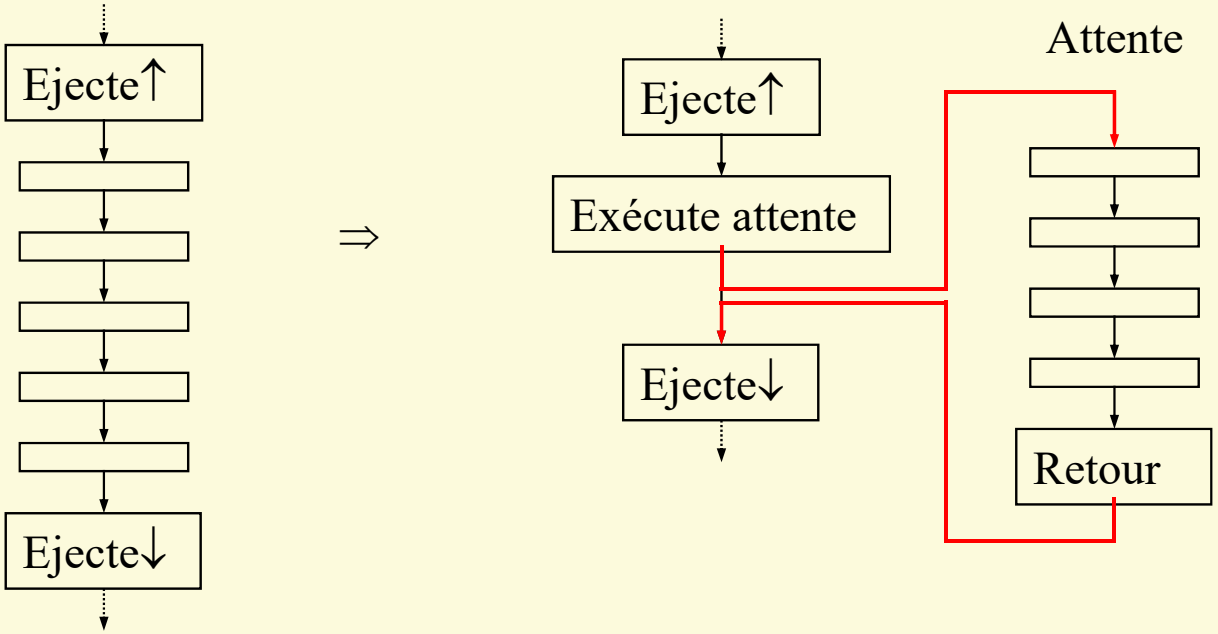
Pourquoi des sous-μprogrammes ?

- 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 :


Ejecte↑
Ejecte↓
⇒
Ejecte↑
Exécute attente
Ejecte↓
Attente
Retour

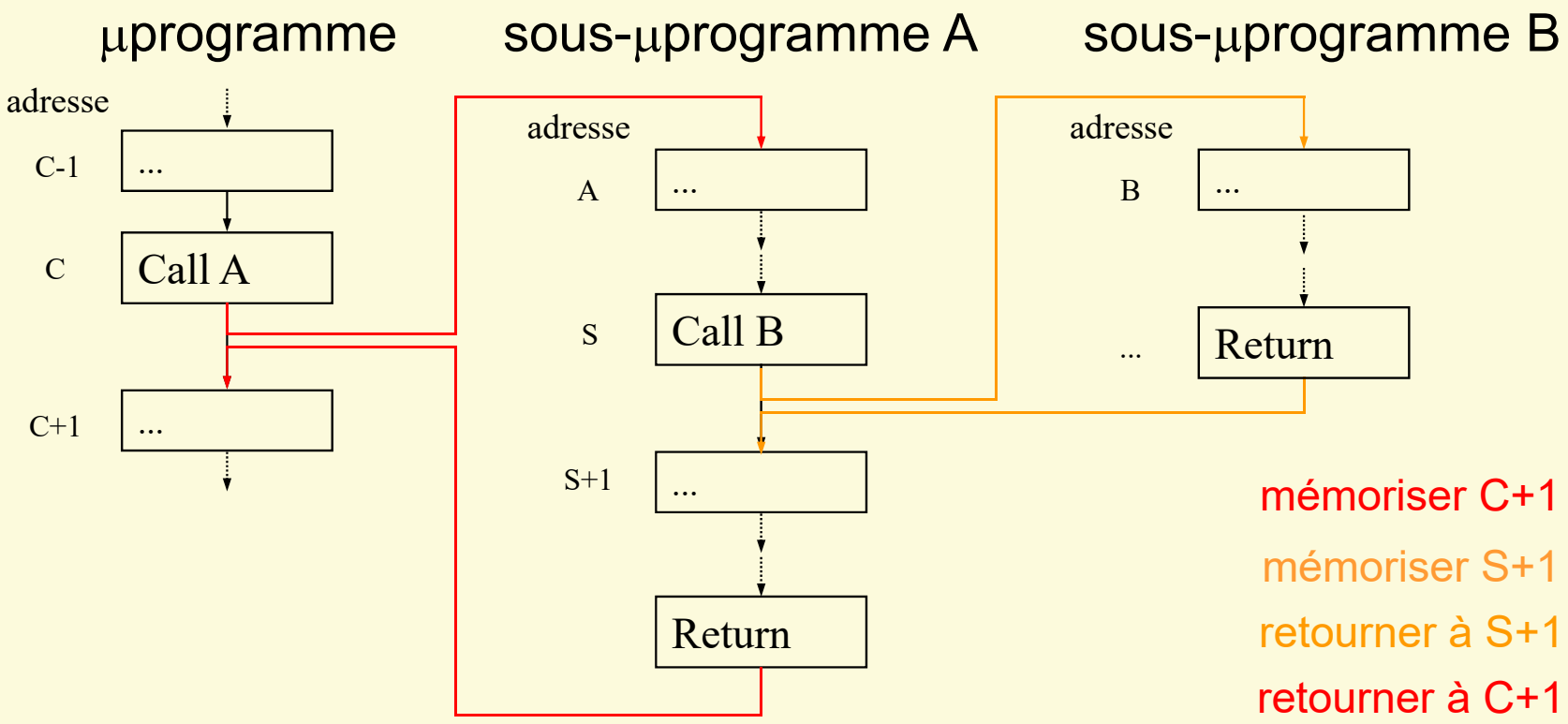
Sous-μprogrammes : définition et mécanisme

- 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 ?



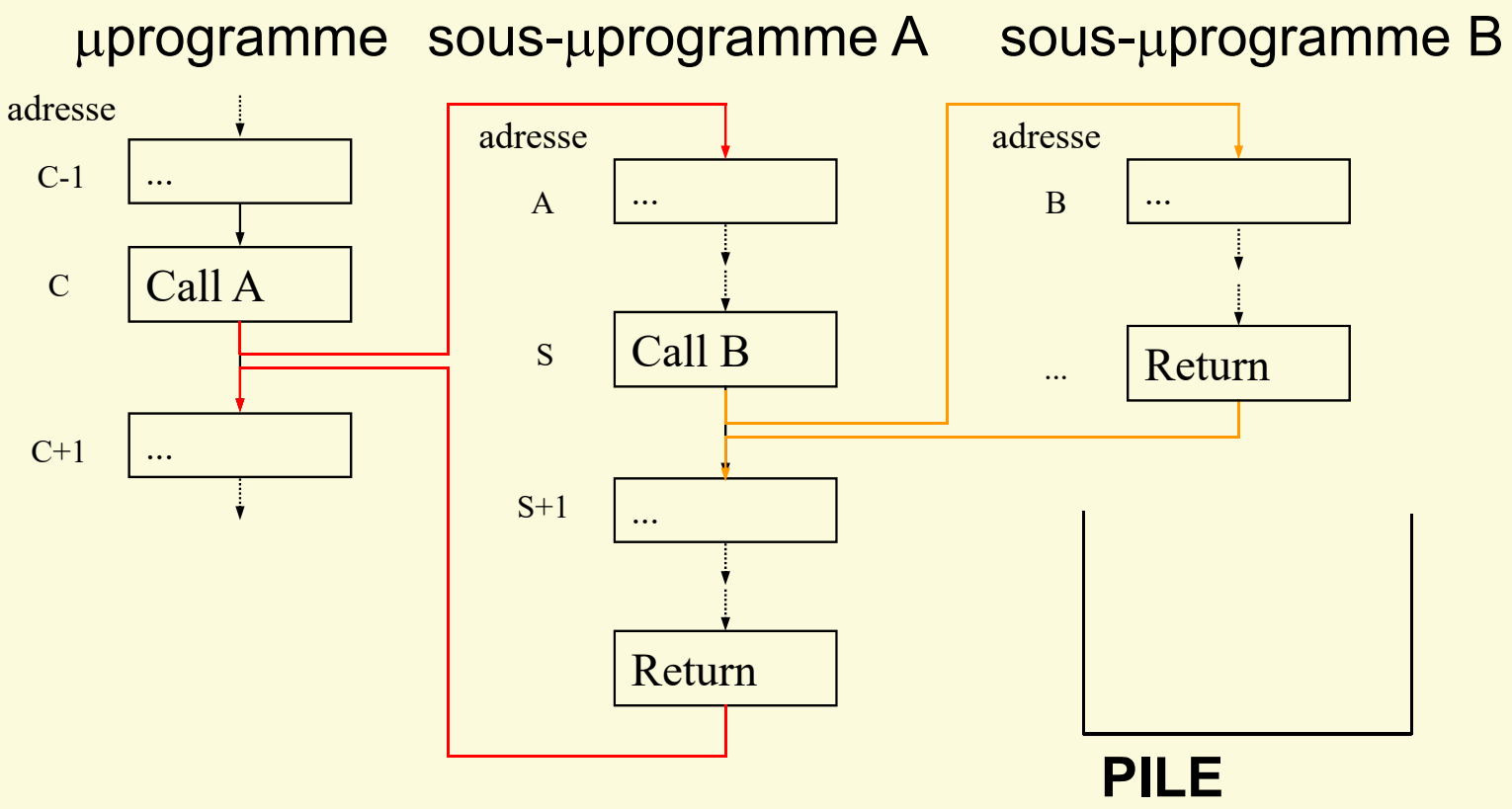
Pourquoi une pile ?

- 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 appels de saut
  - 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 ?



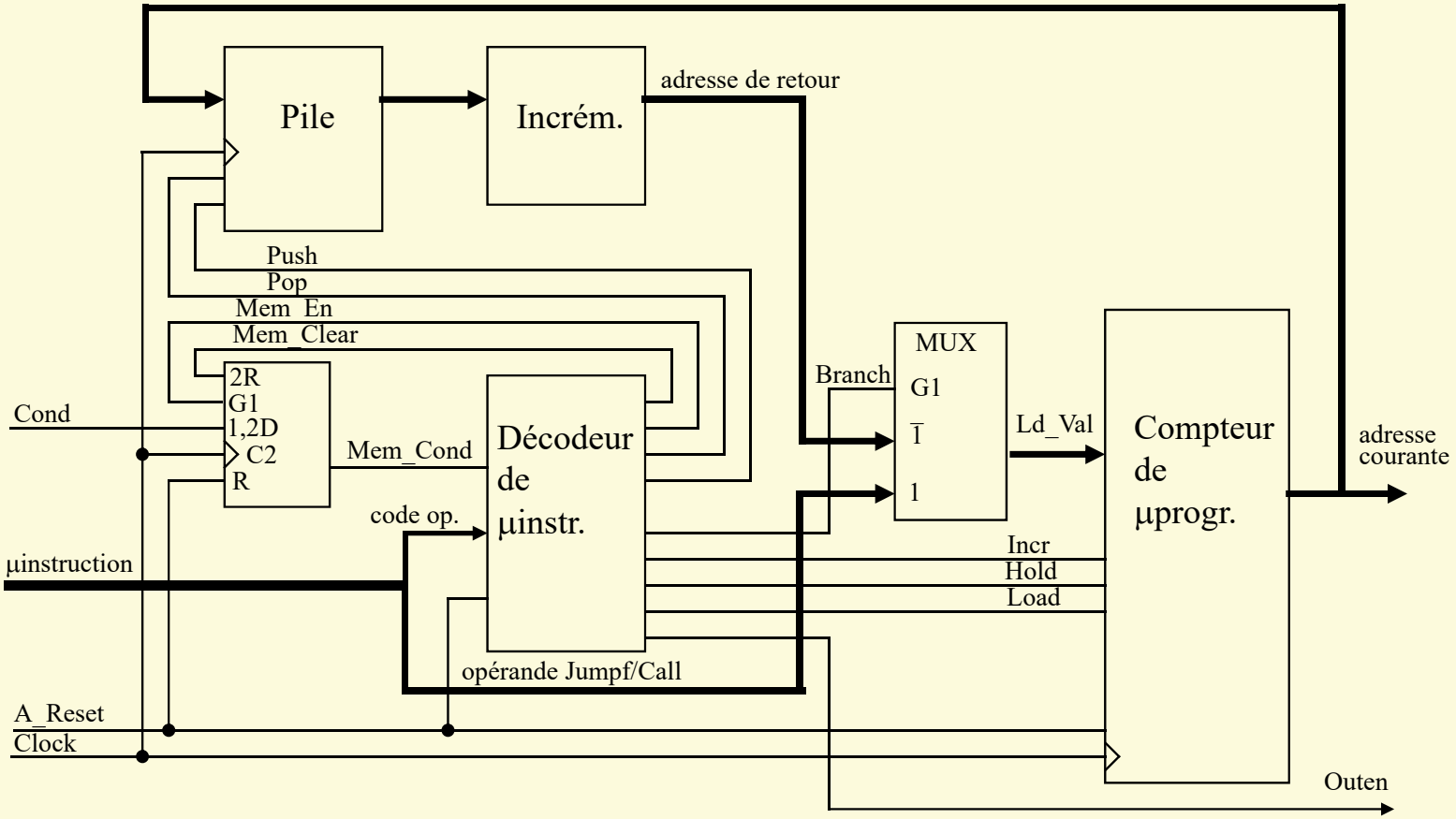
Exemple : MISEQv2 avec Call et Ret → MISEQv3

- 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

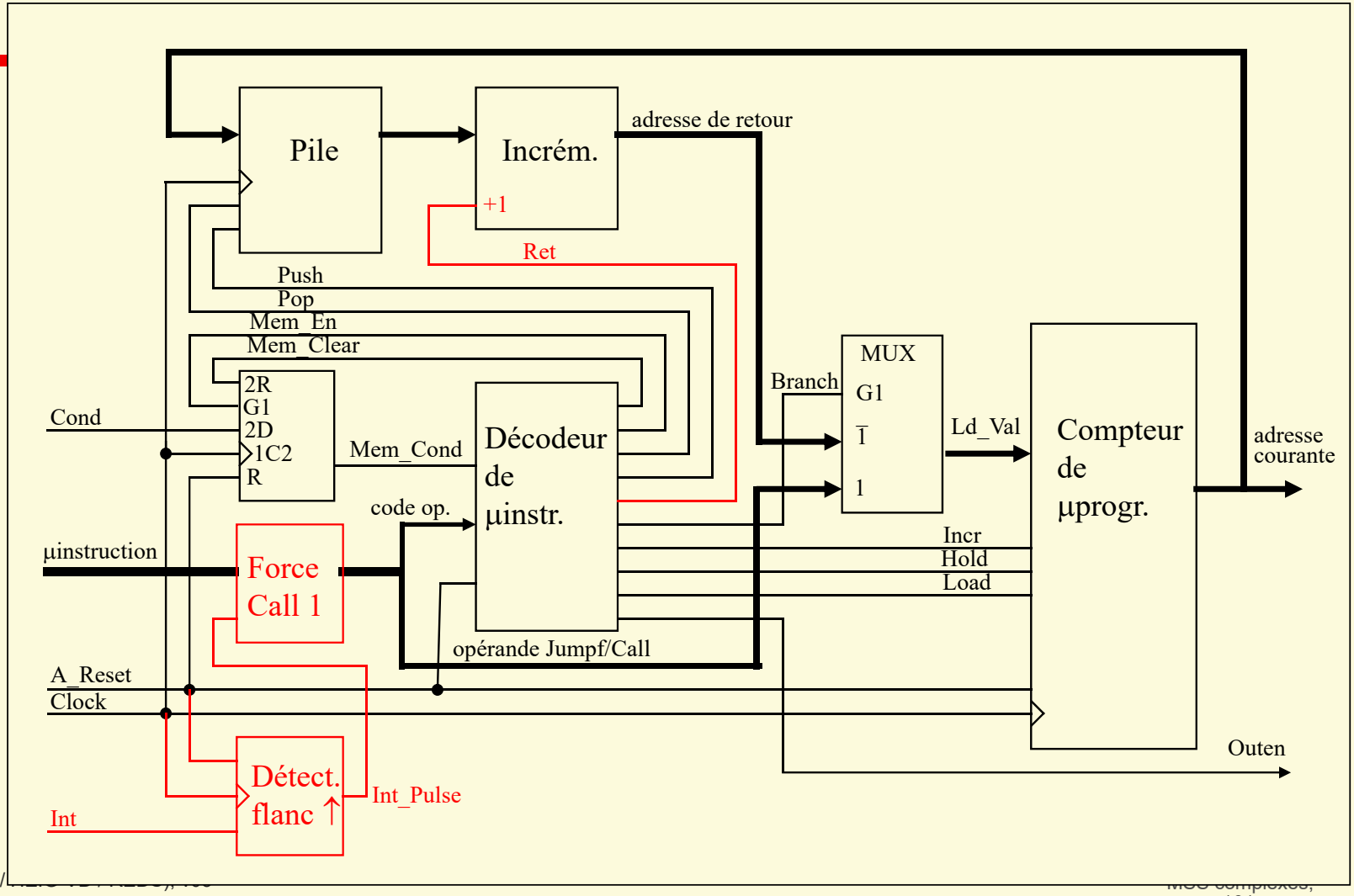
Interruption : définition et mécanisme

- 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.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 ?