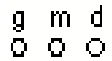
# ES102/PC8 : énoncé et corrigé

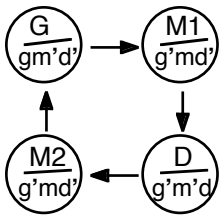
## 1) De spécifications comportementales en diagrammes d'état

*Maîtrisant désormais les étapes **2** (encodage d'états) et **3** (implantation) de la synthèse d'un circuit séquentiel synchrone, on pratique ici l'étape **1** : l'élaboration du diagramme d'état. Celui d'une unité de commande découle facilement du diagramme algorithmique. Mais en général, il faut travailler avec des spécifications comportementales bien moins formalisées…*




1a) Soit un barreau de 3 LEDs, commandées respectivement de gauche à droite par les signaux g, m et d (m=1 allume la diode du **m**ilieu). Proposer un diagramme d'état permettant de voir un point lumineux osciller entre les deux extrémités, ce point sautant d'une LED à sa voisine à chaque top d'horloge.

On peut d'abord penser y parvenir avec un diagramme à 3 états G, M et D aux sorties respectives g=1, m=1 et d=1 (tandis que les 2 autres LED sont éteintes). Mais quel état succède à M ? Il ne peut y en avoir qu'un. On réalise alors qu'il est indispensable de dédoubler M en M1 et M2, d'où le diagramme ci-contre. Ceci nous rappelle l'inégalité |Q| ≥ |Y|, presque toujours stricte sauf cas triviaux. *Avec une fréquence d'horloge en MHz, les états devront en pratique boucler sur eux-mêmes durant de nombreuses périodes, mais grâce à des moyens non précisés ici.*




1b) Le code Morse représente les lettres de l'alphabet sous la forme de chaînes de points (·) et de traits (-), de longueurs variées. Ainsi, la lettre A est codée par un point suivi d'un trait (·-).


L'arbre ci-contre montre le code de chaque lettre. Une arête représente respectivement un point ou un trait selon qu'elle descend vers la gauche (trait fin) ou vers la droite (trait épais). Le message « SOS » s'écrit donc : ··· --- ···.

Des espaces, appelés séparateurs, sont nécessaires entre codes de lettre (sinon on ne saurait pas si le message « SOS » débute par E, I ou S).

Il existe aussi un séparateur de mots, mais qu'on ignorera ici. On considère un signal Morse *m* synchrone (1 bit reçu à chaque top d'horloge) simplifié par rapport au standard, tel que :

- un point est représenté par un 1 isolé reçu entre deux 0 : 010 ;
- un trait est représenté par deux 1 consécutifs reçus entre deux 0 : 0110 ;
- au sein d'une même lettre, les 0 sont partagés entre points/traits successifs, donc isolés
- les lettres sont séparées par au moins deux 0 consécutifs (séparateur) ;
- la réception de plus de deux 1 consécutifs constitue une erreur.

Un message « SOS » seul serait donc reçu comme …0000010101001101101100101010 00…

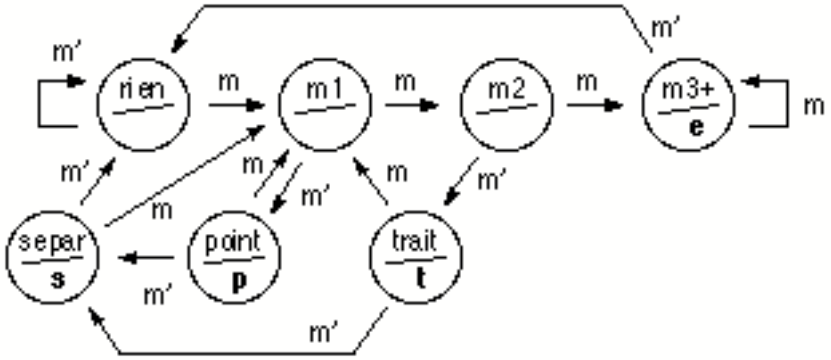
Proposer un diagramme d'état de circuit séquentiel ayant comme signal d'entrée *m* et comme signaux de sortie *p*, *t*, *s* et *e*. Ces derniers doivent indiquer respectivement l'apparition d'un point, d'un trait, d'un séparateur ou d'une erreur, en passant à 1 sur une unique période d'horloge et dès que possible. Un tel circuit constituerait un premier étage d'interprétation du signal *m*, à l'attention d'un deuxième reconnaissant les lettres elles-mêmes.

Sans message transmis sur la ligne, le signal *m* reste à 0 pendant un nombre arbitrairement grand de périodes d'horloge. Cette situation correspond à un état d'attente du système, que l'on

nomme « rien » (pour rien sur la ligne). On peut alors commencer à construire le diagramme d'états selon le nombre de 1 reçus successivement en partant de « rien ». D'où les états « m1 », « m2 » et « m3+ » ci-dessous. Une fois en « m1 » ou « m2 », la réception d'un 0 indique respectivement un point ou un trait, situations auxquelles on consacre 2 états nommés « point » et « trait ». Un 0 de plus reçu indique alors un séparateur de lettre, d'où l'état nommé « separ ». Il reste enfin à compléter de sorte que le successeur de chaque état soit bien déterminé quelle que soit la valeur de *m*. Aucun autre état ne s'avère finalement nécessaire pour réaliser le comportement souhaité (si un 0 est reçu en « m3+ », on décide de revenir à « rien »).

Par souci de lisibilité, on n'affiche que les signaux de sortie à 1 ci-dessous, tandis que les autres sont implicitement à 0. Ainsi, pour l'état « point », la sortie **p** signifie en réalité **pt's'e'**.

A part en « rien » et en « m3+ », on ne reste qu'une période d'horloge dans chaque état : ceci répond à l'exigence de ne mettre *p*, *t* et *s* à 1 qu'une seule fois pour chaque point, trait ou séparateur détecté.

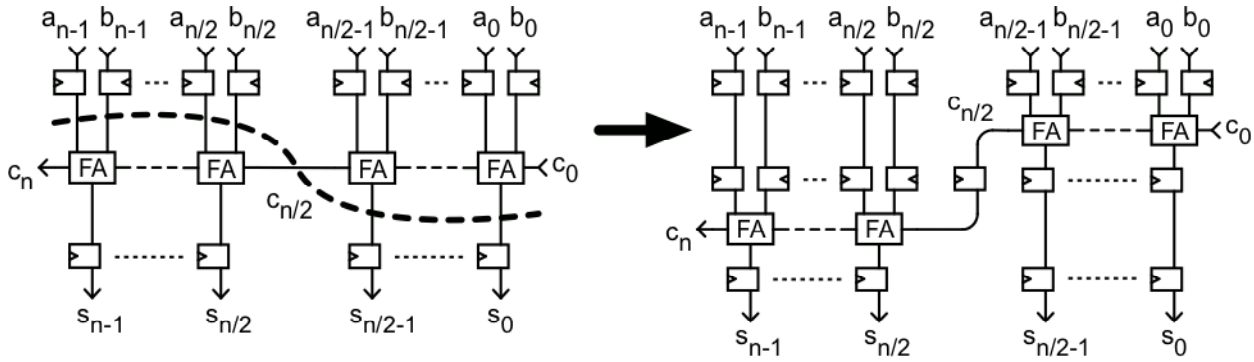



Remarque : quel que soit l'état d'initialisation, il faut au plus trois 0 consécutifs sur *m* pour se retrouver dans l'état « rien ». Un signal *reset* n'est donc pas forcément indispensable ici.

## 2) Additionneur à retenues propagées (ARP) *pipeliné*

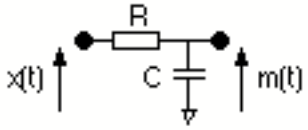
Pipeliner un ARP pour pouvoir doubler (enfin presque) sa puissance de calcul.

La situation type (cf. CM8/P8) est celle d'une unité de calcul trop lente au sein d'un chemin de données. Placée entre 2 registres (réduits ici à de simples bascules D numériques), elle limite la fréquence d'horloge applicable. L'unité considérée ici est un additionneur à retenues propagées (ARP), représentée ci-dessous à gauche, entre 2 rangées de bascules D donc. Il s'avère possible d'accélérer d'un facteur presque 2 en découpant l'unité de calcul considérée en 2 parties chaînées homogènes en délai et en les séparant par des bascules supplémentaires (qui constitueront une bascule D multi-bit). Un ARP se coupe en 2 au niveau de la retenue $c_{n/2}$ ou, plus exactement, selon la courbe en pointillé ci-dessous à gauche. Chaque fil intersecté par cette courbe donne alors lieu à l'insertion d'une bascule D ci-dessous à droite.



La fréquence de fonctionnement peut alors effectivement être presque doublée : pas tout à fait car les bascules D consomment elles-mêmes un petit délai incompressible, qui constitue une portion accrue de la période d'horloge raccourcie. Au cours d'une période d'horloge sont alors calculés les bits de poids faible de S=A+B, d'indice entre 0 et n/2-1, ainsi que $c_{n/2}$. Les autres bits, de poids fort, sont calculés au cours de la période suivante. Il faut réaliser qu'il y a en fait une nouvelle paire d'opérandes A et B présentée au montage à chaque période d'horloge. Du coup, deux calculs ont lieu simultanément : tandis que le calcul de la somme s'achève (poids forts) pour une paire, il débute (poids faibles) pour la paire suivante.

## 3) Système détecteur d'instabilité : 2 versions




Le petit montage analogique ci-contre est régi par l'équation différentielle (*ED*) **dm/dt=(x-m)/RC**. Il réalise un filtre passe-bas entre les signaux *x* et *m* ; et entre les signaux *x* et *x-m* (tension aux bornes de la résistance), un filtre passe-haut. Ce dernier permet de repérer des discontinuités affectant le signal *x*. Analytiquement, le signal *m* résulte de la convolution du signal *x* par un noyau positif unilatéral de type exponentiel décroissant : *m* est ainsi une *moyenne mobile* de *x*.

Une version en temps discret de l'ED ci-dessus est la loi d'évolution $\mathbf{m^+ = m + \varepsilon(x\text{-}m)}$ où ε, appelé « coefficient d'actualisation », est positif et typiquement petit devant 1. Les valeurs des signaux *x* et *m* peuvent aussi être discrétisées, sans altérer les propriétés de filtrage. On va donc les représenter sous forme numérique, avec d'éventuels chiffres après la virgule, indispensables en tout cas pour exprimer ε.

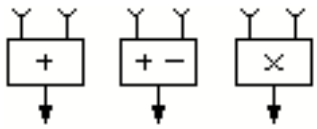
Un processus physique présentant un danger d'instabilité est observé via un capteur qui fournit à une fréquence $f_{capt}$ la mesure numérique *x* d'une grandeur physique caractéristique. L'instabilité se manifeste par des hausses relatives soudaines de *x*, alors que ce signal varie lentement en temps normal. On va justement détecter ces instabilités :

- en tenant à jour une moyenne *m* de *x* grâce à la loi d'évolution précédente, avec $\boldsymbol{\varepsilon \approx 5\%}$ ;
- en signalant le système observé comme instable chaque fois que $\mathbf{(x\text{-}m) > \beta m}$, avec $\boldsymbol{\beta \approx 0{,}3}$.

Ceci demande d'effectuer des calculs à chaque nouvelle mesure fournie par le capteur, c'est-à-dire à la fréquence $f_{capt}$, d'où un caractère nativement séquentiel.

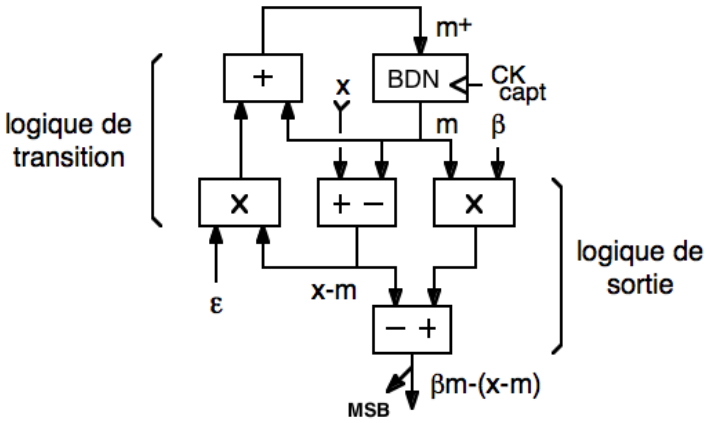
3a) On dispose d'un signal d'horloge $CK_{capt}$ de fréquence $f_{capt}$ et *x* est supposé synchrone par rapport à celui-ci. Rappeler ce que cela signifie.

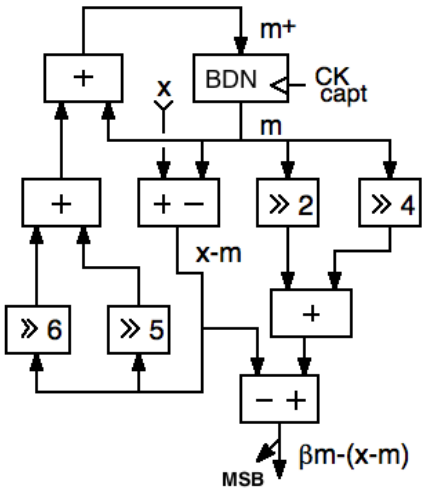
Cela signifie que, à chaque période de l'horloge $CK_{capt}$, les bits constituant le signal numérique *x* se stabilisent assez longtemps avant le prochain top pour que des blocs combinatoires les utilisant en entrée aient le temps de faire leur calcul et présenter des sorties stables lors du prochain top pour échantillonnage par les bascules D.




3b) Proposer un système séquentiel répondant directement au besoin exprimé, à un niveau numérique. Se servir notamment des briques combinatoires ci-contre (dont un multiplieur) en faisant comme si ε et β étaient des entiers.

L'état *m* devant être mis à jour à chaque top de $CK_{capt}$, une simple bascule D numérique (désignée par BDN ci-dessous) suffit pour le porter, cadencée par $CK_{capt}$ (pas besoin d'un registre et de son signal de chargement *load*). L'implantation numérique est présentée ci-dessous, à gauche. La loi d'évolution $m^+ = m + \varepsilon(x\text{-}m)$ est implantée par la logique de transition, à gauche de BDN. Profitant de ce que x-m est déjà calculé, la logique de sortie, au-dessous de BDN, calcule la différence βm-(x-m). Celle-ci est négative ssi (x-m)>βm. Le MSB de ce nombre sera donc égal à 1 en cas d'instabilité du processus physique. Il s'agit donc de **la** sortie binaire du système.



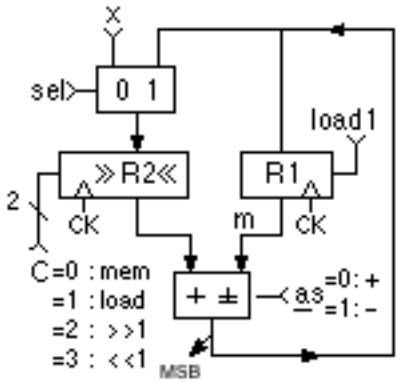




3c) Mais ε et β ne sont pas des entiers. Utiliser des décaleurs *n* bits (>>n) et des additionneurs pour réaliser approximativement les multiplications correspondantes.

Vu les précisions des valeurs décimales indiquées pour ε et β, on peut les approximer ainsi sous forme binaire : $\varepsilon = 5\% \approx 3/64 = (0{,}000011)_2$ et $\beta = 0{,}3 \approx 5/16 = (0{,}0101)_2$. Là où la « calculette » du CM7 décalait vers la gauche, il faut ici décaler vers la droite. D'où le schéma ci-dessus à droite. Les décalages étant réalisés simplement avec des fils, on ne se donne pas la peine de les « factoriser ».

3d) Quel est le coût en transistors de ce détecteur d'instabilité si *x* et *m* sont sur 32 bits ?

Par tranche, il faut donc une bascule D (24t) pour BDN, 5 FA (28t chacun, chiffre non divulgué jusque là) pour les additionneurs ou soustracteurs et 2 inverseurs (2t chacun) pour soustraire au lieu d'additionner, soit 168 transistors. Pour 32 tranches, cela fait 5376 transistors.




Ci-dessus, les calculs relatifs à la détection d'instabilité ont été réalisés de façon combinatoire, l'horloge $CK_{capt}$ servant seulement à mettre à jour la moyenne mobile *m* peu après chaque arrivée d'une nouvelle mesure *x*. On envisage désormais de réaliser ces calculs sur le chemin de données (CD) ci-contre, pour gagner en flexibilité. Ceci implique une séquentialisation plus fine en temps, par une horloge CK dont la fréquence *f* sera un multiple assez grand de $f_{capt}$ (supposée modérée). Ce CD comporte deux registres cadencés par CK : R1 (entrée *load1*) servira à mémoriser la moyenne mobile *m*, tandis que R2 sera au cœur des calculs, équipé de décaleurs 1 bit vers la droite ou la gauche (cf. CM7/P7) avec commande C sur 2 bits (voir détails sur la figure). L'additionneur/soustracteur permet en outre de calculer au choix \$R2+\$R1 si as=0 ou \$R2-\$R1 si as=1 (*a* pour **a**ddition, *s* pour **s**oustraction). Le résultat peut être dirigé vers R1 (load1=1) ou R2 (sel=1 et C=1). Ce petit CD et son UC pourront être *embarqués* à bord du capteur pour en faire un *capteur intelligent*.

3e) A priori, pourquoi R1, porteur de *m*, ne peut-il se réduire à une simple bascule D numérique ?

Désormais, $f/f_{capt}$ périodes de l'horloge CK vont s'écouler entre 2 arrivées de valeurs de *x* fournies par le capteur. Au cours de ces $f/f_{capt}$ périodes vont être réalisés séquentiellement les calculs qui l'étaient combinatoirement auparavant, où l'additionneur-soustracteur va fournir des valeurs destinées tantôt à R1, tantôt à R2. Dans le second cas, il faudra bien maintenir le contenu de R1, qui ne peut donc être qu'un registre.

3f) Ce chemin de données coûte-t-il plus ou moins cher que la solution combinatoire précédente ?

La différence, par tranche, entre les deux est de 1 bascule D, 5 MUX (dont 1 dans R1 et 3 dans R2 pour le MUX 4 vers 1) et un XOR (±) côté CD, contre 4 FA et 2 inverseurs (–) dans la solution précédente ; soit un petit avantage d'une vingtaine de transistors pour la tranche du CD (il a fallu payer des MUX dans le CD, là où de simples fils suffisaient à réaliser les décalages, précédemment). Par ailleurs, ce CD est-il suffisamment équipé avec seulement 2 registres ?

3g) Partant d'un top d'horloge où une nouvelle valeur de $x$ est fournie par le capteur, établir la séquence d'opérations nécessaire pour implanter la loi d'évolution de $m$.

La séquence est décrite cycle par cycle (cycle = période d'horloge). C'est un petit *programme*. Le résultat obtenu (colonne de droite) l'est après le prochain top d'horloge.

| n° | opération | commande | résultat (à la période suivante) | |
|---|---|---|---|---|
| 01 | R2 ← x | sel=0, C=1, load1=0, as=– | $R2=x | |
| 02 | R2 ← $R2–$R1 | sel=1, C=1, load1=0, as=1 | $R2=x-m | |
| 03 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=1/2·(x-m) | |
| 04 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=1/4·(x-m) | |
| 05 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=1/8·(x-m) | |
| 06 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=1/16·(x-m) | |
| 07 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=1/32·(x-m) | |
| 08 | R1 ← $R2+$R1 | sel=–, C=0, load1=1, as=0 | $R1=m+1/32·(x-m) | |
| 09 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=1/64·(x-m) | ε = 5% |
| 10 | R1 ← $R2+$R1 | sel=–, C=0, load1=1, as=0 | $R1=m+3/64·(x-m) | ≈ 3/64 |

*Remarque : la paire d'instructions successives 08&09 ci-dessus pourrait être fusionnée en une seule (avec* C=2, load1=0, as=0*). On ne l'a pas fait, dans un souci de lisibilité.*

3h) Poursuivre la « programmation » du CD pour pouvoir détecter s'il y a instabilité.

Il s'agit de calculer βm-(x-m) et d'obtenir son signe, via son MSB. Mais après la séquence ci-dessus de mise à jour de $m$, les valeurs initiales de $m$ et (x-m) ont disparu. Seules demeurent disponibles celles de $m^+$ dans R1 et 1/64·(x-m) dans R2. Les 6 bits de poids faible de x-m ont donc été perdus : c'est un souci. Par ailleurs, vouloir remonter à $m$ paraît absurde puisqu'il faudrait détruire $m^+$ pour cela. Et si l'on calculait plutôt βm-(x-m) *avant* de mettre $m$ à jour ?

Mais ne va-t-on pas là aussi détruire de l'information et rendre ensuite impossible le calcul de $m^+$. A cet instant, le doute s'installe et l'on se dit que ce petit CD avec ses 2 registres est peut-être trop limité. Essayons quand même. On commence donc par installer (x-m) dans R2, $m$ occupant R1, comme précédemment. Puis on reconsidère le calcul de βm-(x-m). Cela se présente mal.

La valeur (x-m) étant installée du côté '+' de l'additionneur/soustracteur, on ne pourra pas la soustraire. Soit, calculons plutôt (x--m)-βm. Lorsque le MSB de cette différence vaudra 0, cela indiquera que (x-m)≥βm. Le fait de passer ainsi à une inégalité au sens large n'a guère d'importance.

Mais comment multiplier $m$ par β (≈ 5/16) alors que R1 n'est pas équipé de décaleur ? Et même si l'on pouvait, ne perdrait-on pas de l'information sur $m$ à cette occasion ? Une solution à ces deux obstacles apparaît : elle consiste à calculer 16·[(x-m)-βm] ≈ 16(x-m)-5m sur R2, en multipliant itérativement x-m par 2 et en retranchant $m$ à 2 reprises. On récupérera ainsi le signe de (x-m)-βm et l'on pourra ensuite revenir à (x-m) sans perte. Il faudra toutefois avoir mis en place des registres assez larges pour que 4 décalages vers la gauche de leur contenu ne les fassent pas déborder. Une perte de bits de poids fort à 1 serait évidemment catastrophique.

*La programmation de ces calculs, fastidieuse et d'un intérêt limité suite à la question précédente, ne sera probablement pas détaillée en séance, faute de temps. Elle l'est cependant ci-après pour aller au bout de la démarche.*

Les 8 premiers cycles ci-dessous servent à obtenir la valeur 16(x-m)-5m. Un détail : une fois obtenue, on ne la range pas dans un registre. Son MSB, qui sera stable au top d'horloge suivant, sera alors pris en compte par l'unité de commande qui gère ce CD.

Les cycles 9 à 13 servent à défaire ce que l'on a fait pour faire réapparaître (x-m) dans R2. A noter que, si la valeur de x (founie par le capteur) est restée inchangée en entrée, il est plus rapide de recalculer x-m. La séquence de mise à jour de *m* élaborée dans la question précédente (à partir de l'étape 03) peut ensuite être déroulée, entre les cycles 14 et 21.

| n° | opération | commande | résultat (à la période suivante) | |
|---|---|---|---|---|
| 01 | R2 ← x | sel=0, C=1, load1=0, as=– | $R2=x | |
| 02 | R2 ← $R2–$R1 | sel=1, C=1, load1=0, as=1 | $R2=x-m | |
| 03 | R2 ← $R2×2 | sel=–, C=3, load1=0, as=– | $R2=2(x-m) | |
| 04 | R2 ← $R2×2 | sel=–, C=3, load1=0, as=– | $R2=4(x-m) | |
| 05 | R2 ← $R2–$R1 | sel=1, C=1, load1=0, as=1 | $R2=4(x-m)-m | |
| 06 | R2 ← $R2×2 | sel=–, C=3, load1=0, as=– | $R2=8(x-m)-2m | |
| 07 | R2 ← $R2×2 | sel=–, C=3, load1=0, as=– | $R2=16(x-m)-4m | β=0,3 |
| 08 | ø ← $R2–$R1 | sel=–, C=0, load1=0, as=1 | MSB=0 ⇔16(x-m)-5m ≥ 0 | ≈5/16 |
| 09 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=8(x-m)-2m | |
| 10 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=4(x-m)-m | |
| 11 | R2 ← $R2+$R1 | sel=1, C=1, load1=0, as=0 | $R2=4(x-m) | |
| 12 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=2(x-m) | |
| 13 | R2 ← $R2/2 | sel=–, C=2, load1=0, as=– | $R2=x-m | |

14-21 idem 03-10 de la question précédente

*Ainsi, pour s'en sortir, il aura fallu compenser la pauvreté fonctionnelle du CD par des astuces, coûteuses en nombre d'instructions. La dissymétrie entre les 2 registres s'est avérée gênante en particulier. On parle de* manque d'orthogonalité*, un défaut que l'on veillera à ne pas reproduire dans l'architecture d'un microprocesseur*. En revanche, les astuces utilisées seraient valables quelles que soient les valeurs de ε et β.

3i) A quoi ressemblerait le diagramme d'état de l'UC nécessaire pour piloter le CD considéré ?

Il comporterait nécessairement la chaîne de 21 états correspondant au code précédent. Cela peut suffire si UC et CD sont cadencés à une fréquence $f=21 \cdot f_{capt}$. Mais générer une telle fréquence n'est pas forcément facile, ni synchroniser le capteur avec l'UC. Il serait plus simple de choisir une fréquence f supérieure à $21 \cdot f_{cap}$ et demander à disposer d'un signal d'entrée *new* indiquant lorsqu'une nouvelle valeur de *x* est disponible en sortie du capteur. Dans ce cas, le diagramme comporterait en plus un état d'attente, bouclant sur lui-même tant que new=0, exactement comme avec le *start* de la calculette du CM7. Il faut aussi gérer la sortie MSB de l'additionneur/soustracteur du CD. Si elle doit être utilisée en tant qu'entrée de l'UC, cela en modifiera le diagramme d'état, impliquant l'apparition d'une fourche conditionnée par MSB. Mais il est probable que ce MSB serait plutôt utilisé en tant qu'entrée d'un autre circuit séquentiel, pour activer une alarme typiquement, qui ne serait levée qu'après action.