DIRO
IFT 1215

EXAMEN FINAL

Max Mignotte

DIRO, Département d'Informatique et de Recherche Opérationnelle, local 2377
Http: //www.iro.umontreal.ca/∼mignotte/ift1215/
E-mail: mignotte@iro.umontreal.ca

Date: *28/04/2022*

I	Mémoire micro-programmée (31 pts)
II	Assembleur/LMC/Registres (34 pts)
III	Programmation Shell script (25 pts)
IV	Codes correcteurs (15 pts)
V	Misc. (05 pts)
Total	110 pts

Directives

- TOUTE DOCUMENTATION ET CALCULATRICE PERSONNELLE PERMISES

- Les réponses devront être clairement présentées et justifiées (elles peuvent être concises mais devront néanmoins contenir les résultats intermédiaires nécessaires permettant de montrer sans ambiguïté que vous êtes arrivés au résultat demandé).
- Si vous ne comprenez pas une question, faites en une interprétation, et proposez une réponse.
- Utiliser le recto et verso de votre cahier.
- À la fin, remettez votre énoncé d'examen dans votre cahier d'examen.

I. Mémoire micro-programmée (31 pts)

Soit le diagramme de transition d'états suivants (cf. Fig. 1) qui correspond au circuit séquentiel demandé dans l'examen intra H22 correspondant à la conception d'un système de machine automatique permettant à l'utilisateur de se faire rembourser (automatiquement) la consigne (de bouteilles).

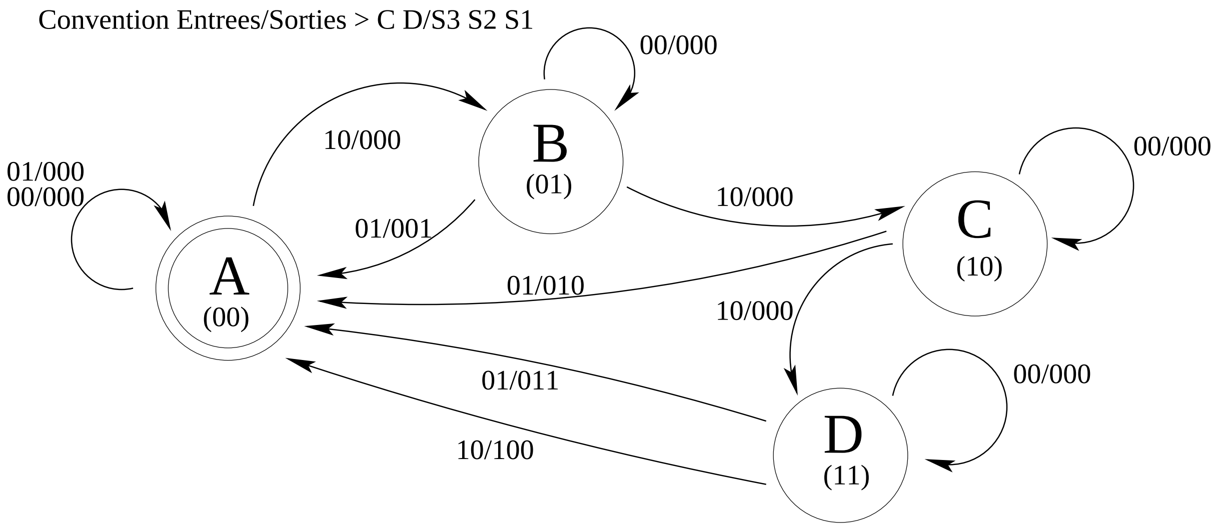


Figure 1: Diagramme de transition d'états de la machine de remboursement de consigne

On rappelle que cette machine vous donne 25¢ pour chaque bouteille en verre consignée. Pour que l'utilisateur reçoive son argent, il existe trois possibilités. Soit l'utilisateur presse le bouton **D**one après chaque retour de bouteille et la machine lui donnera automatiquement une pièce de 25¢. Soit l'utilisateur donne successivement quatre bouteilles, l'une après l'autre, et la machine lui donnera automatiquement, à la fin de cette opération, sans avoir à presser de bouton (*i.e.* sans avoir à appuyer sur D) une pièce de 1 \$. Soit l'utilisateur donne deux ou trois bouteilles, l'une après l'autre et appuie ensuite sur le bouton **D**one, à la fin de cette opération et la machine lui donnera automatiquement le nombre correct de pièces de 25¢ (*i.e.*, deux ou trois pièces de 25¢ dans notre exemple). Si le bouton **D**one est pressée sans que l'utilisateur ait rendu de bouteille en verre, alors aucune pièce de monnaie n'est donnée.

Dans ce système séquentiel, les sorties seront donc les trois variables $S3\,S2\,S1$ indiquant:

- ▷ Si $S3\,S2\,S1 = 000$, aucune pièce de 25¢ n'est rendue.
- ▷ Si $S3\,S2\,S1 = 001$, une pièce de 25¢ est rendue.
- ▷ Si $S3\,S2\,S1 = 010$, deux pièces de 25¢ sont rendues.
- ▷ Si $S3\,S2\,S1 = 011$, trois pièces de 25¢ sont rendues.
- ▷ Si $S3\,S2\,S1 = 100$, une piece de 1\$ est rendue.

et les entrées de ce système sont le bouton D (**D**one) ($D = 1$ pour pressée et $D = 0$ pour non pressée) et une autre entrée est C (pour consigne). Un capteur détecte le rendu d'une bouteille par $C = 1$. Lorsque aucune bouteille n'est détectée ou rendue: $C = 0$. Il n'y a pas de RESET sur cette machine. Le fait d'appuyer sur le bouton **D**one) permet de rendre la monnaie et de remettre le système dans l'état initial d'attente de réception de bouteille. Dans cette machine, D et C ne peuvent être simultanément à un.

On dispose d'une horloge et d'une petite mémoire ROM ou EPROM permettant de stocker 16 Bytes (octets).

1. Indiquer le dispositif microprogrammé (*i.e.*, le schéma électrique) permettant de créer (avec cette mémoire et cette horloge) la FSM (machine à états finis) simulant ce diagramme de transition d'états. Pour des raisons techniques on aimerait que les états soient le LSB (less significant bit) de l'adresse d'entrée (*input*)
   et de la valeur en sortie (*output*) de la mémoire.
   <6 pts>

2. Donner quels sont les différents micro-codes que l'on devra stocker dans cette mémoire (ainsi que leurs adresses respectives) *en hexadecimal* permettant ainsi de micro-programmer ce circuit séquentiel et de simuler ce diagramme de transition d'états.
   <7 pts>

3. Donner les avantages et inconvénients de cette méthode par rapport à un câblage par portes logiques (cf Intra H22) en essayant, autant que possible, de les classer par ordres d'importance.
   <4 pts>

4. Suite à une crise économique et à la réponse du gouvernement d'augmenter sa dette en imprimant des billets de banque et en donnant des subventions tout azimuth, la monnaie du pays vient de perdre 50% de sa valeur et du coup la valeur de la consigne d'une bouteille de verre vient subitement de monter à $50¢$ pour chaque bouteille en verre consignée. En tant qu'informaticien, on vous demande de re-programmer cette machine automatique pour que celle-ci donne $50¢$ (2 pièces de $25¢$) pour chaque bouteille en verre consignée après appui sur le bouton **D**one. Si l'utilisateur donne deux bouteilles, la machine lui donnera automatiquement, à la fin de cette opération, sans avoir à presser de bouton (*i.e.* sans avoir à appuyer sur **D**) une pièce de 1 $.
   Reprogrammer cette mémoire en changeant un minimum de mot (d'un octet) mémoire stocké en elle pour que la machine automatique soit modifiée pour suivre ses nouvelles directives.
   <8 pts>

5. Rappelons que la micro-programmation de la question 2., (programme initial de la Fig. 1) ne prend pas en compte le fait que, logiciellement, D et C puissent être simultanément à un, (*c-à-d*, que cette situation soit prise en compte dans le programme initial de la Fig. 1 (ou demandé a la question 2.) En effet, on remarquera qu'il n'y a aucune fleche sur laquelle il y a $11/xxx$ (convention $C\,D/S_3S_2S_1$).
   Cette particularité pourrait être utilisée par le programmeur qui fait la micro-programmation de la question 2 et plus précisément utiliser cela à son avantage en décidant d'**ajouter** au programme une micro-code (que seul lui connaîtra) et qui se situe dans l'une des adresses non utilisées (du micro-programme de la question 2.) et qui lui permettra, une fois que la machine se sera mis à l'état B (et donc une fois qu'une première bouteille aura été consigné), en appuyant sur **D** et en mettant en meme temps, une deuxième consigne de faire en sorte que la machine se rende directement à état initial A en lui rendant une pièce de 1 dollar (donc forçant la machine à lui donner deux fois plus d'argent qu'en cas normal). Trouver cette micro-instruction et l'adresse à laquelle on doit la mettre. Justifier bien votre réponse.
   <6 pts>

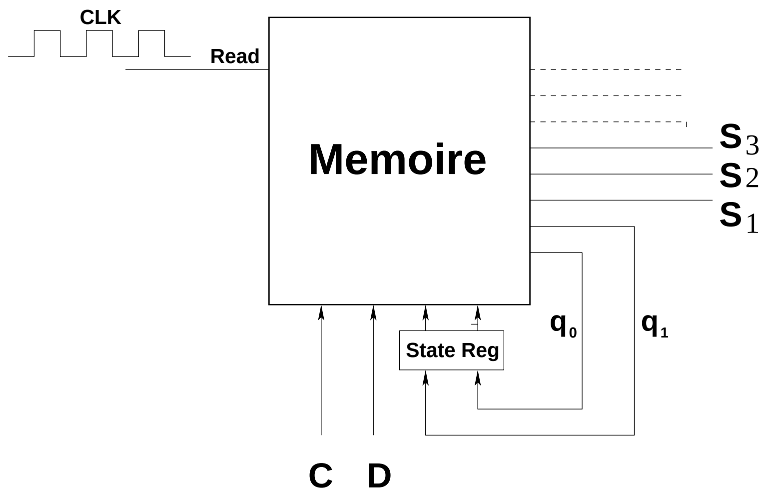
Réponse

$\boxed{\textbf{1.}}$

En respectant les consignes de l'énoncé,
dans lequel, on demande:

• que les états soient le LSB (less significant bit) de l'adresse d'entrée (*input*)
• et de la valeur en sortie (*output*) de la mémoire.

On doit donc considérer le dispositif suivant:



<6 pts>

$\boxed{2.}$

Le diagramme de transition d'états nous permet de retrouver immédiatement la table de vérité qui lui est associée. On s'assure de respecter les consignes de l'énoncé, dans lequel, on doit mettre les états dans la position du LSB de l'adresse d'entrée (*input*) et de la valeur en sortie (*output*) de la mémoire.

INPUT				**OUTPUT**					**HEX**
ENTRÉES		ÉTATS PRÉSENTS		SORTIES			ÉTATS FUTURS		input - output
C	D	$q_1(t)$	$q_0(t)$	S3	S2	S1	$q_1(t+1)$	$q_0(t+1)$	address - value
0	0	0	0	0	0	0	0	0	$00_{16} - 00_{16}$
0	0	0	1	0	0	0	0	1	$01_{16} - 01_{16}$
0	0	1	0	0	0	0	1	0	$02_{16} - 02_{16}$
0	0	1	1	0	0	0	1	1	$03_{16} - 03_{16}$
0	1	0	0	0	0	0	0	0	$04_{16} - 00_{16}$
0	1	0	1	0	0	1	0	0	$05_{16} - 04_{16}$
0	1	1	0	0	1	0	0	0	$06_{16} - 08_{16}$
0	1	1	1	0	1	1	0	0	$07_{16} - 0C_{16}$
1	0	0	0	0	0	0	0	1	$08_{16} - 01_{16}$
1	0	0	1	0	0	0	1	0	$09_{16} - 02_{16}$
1	0	1	0	0	0	0	1	1	$0A_{16} - 03_{16}$
1	0	1	1	1	0	0	0	0	$0B_{16} - 10_{16}$

À chaque [ENTRÉES::ÉTATS PRÉSENTS] différent codé en hexadécimal (que l'on prendra comme adresses) correspond une [SORTIES::ÉTATS FUTURS] (*i.e.*, un ensemble de bits codés en hexadécimal) que l'on enregistrera dans la mémoire. Ainsi la série des 12 micro-codes associées que l'on demande de donner en hexadécimal pour chaque adresse d'entrée de la mémoire, de l'adresse 0 à 11 ou 00_{16} à $0B_{16}$ (en hexadécimal) est:

$$\{\mathbf{00_{16} \quad 01_{16} \quad 02_{16} \quad 03_{16} \quad 00_{16} \quad 04_{16} \quad 08_{16} \quad 0C_{16} \quad 01_{16} \quad 02_{16} \quad 03_{16} \quad 10_{16}}\}$$

<6 pts>

NOTA : Pour plus de sécurité, on s'assurera de mettre dans cette mémoire, de l'adresse 13 (OD_{16}) à l'adresse 15 (OF_{15}) la valeur 00_{16}. Ainsi, si une petite perturbation (électrique ou électromagnétique) fait malencontreusement lire les mots mémoires situés entre l'adresse 13 et 15, on reviendra à l'état initial du système (état A ou 00 et aucune pièce de 25¢ ne sera rendue.

$\boxed{3.}$

L'avantage d'un séquenceur microprogrammé réside majoritairement dans sa souplesse et sa simplicité et surtout sa réutilisabilité. Ainsi, on pourrait facilement re-programmer (une partie de) la mémoire pour obtenir une modification

du fonctionnement de la FSM (machine à états finis) ou bien re-utiliser la carte électronique sur laquelle est cette mémoire pour la reprogrammer et ainsi l'utiliser pour le séquençage d'une nouvelle FSM simulant un autre diagramme de transition d'états. La réalisation est plus facile que le câblage par porte logique, bien sur aussi. Il est par contre plus lent qu'un séquenceur câblé mais pour le séquenceur d'une machine automatique de remboursement de consigne, cela n'a pas beaucoup d'importance.
<4 pts>

NOTA : On dirait que la plus petite EPROM commercialisable actuellement est une EPROM 8KBIT 1024×8 bits comme la M24C08-WDW6TP ou M24C08-RMN6TP vendu à \$0.185. Le prix n'est donc pas vraiment un avantage d'un câblage par portes logiques.

4.

Pour modifier le moins possible le diagramme, mais permettre de suivre les nouvelles consignes, l'état **A** reste le même mais sous l'effet de l'entrée $CD = 10$, on se retrouve directement en l'état **C**. L'état **D** n'existe plus. À l'état **C**, si on a $CD = 10$, alors on se retrouve a l'état **A** avec une pièce de 1 \$ et si on $CD = 01$, alors on se retrouve à l'état **A** avec 50¢ (*i.e.*, 2 pièces de 25¢).

INPUT				OUTPUT					HEX
ENTRÉES		ÉTATS PRÉSENTS		SORTIES			ÉTATS FUTURS		input - output
C	D	$q_1(t)$	$q_0(t)$	S3	S2	S1	$q_1(t+1)$	$q_0(t+1)$	address - value
0	0	0	0	0	0	0	0	0	$00_{16} - 00_{16}$
0	0	1	0	0	0	0	1	0	$02_{16} - 02_{16}$
0	1	0	0	0	0	0	0	0	$04_{16} - 00_{16}$
0	1	1	0	0	1	0	0	0	$06_{16} - 08_{16}$
1	0	0	0	0	0	0	1	0	$08_{16} - \underline{\mathbf{02_{16}}}$
1	0	1	0	1	0	0	0	0	$0A_{16} - \underline{\mathbf{10_{16}}}$

Il suffit donc de modifier les micro-codes stockés à l'adresse 08_{16} par 02_{16} et celui de l'adresse $0A_{16}$ par 10_{16} pour avoir la FSM modifiée adéquatement. Ainsi, la nouvelle liste de mots est:

$$\{\mathbf{00_{16}} \quad \mathbf{01_{16}} \quad \mathbf{02_{16}} \quad \mathbf{03_{16}} \quad \mathbf{00_{16}} \quad \mathbf{04_{16}} \quad \mathbf{08_{16}} \quad \mathbf{0C_{16}} \quad \underline{\mathbf{01_{16}} \Rightarrow \mathbf{02_{16}}} \quad \mathbf{02_{16}} \quad \underline{\mathbf{03_{16}} \Rightarrow \mathbf{10_{16}}} \quad \mathbf{10_{16}}\}$$
<8 pts>

5.

Pour cela, Il suffit que le programmeur crée une fleche supplémentaire au diagramme de la Fig 1 (que seul lui connaîtra !) qui partira de l'état B en se rendant à l'état A avec la condition Entrée/Sortie ($C\,D/S_3S_2S_1$) qui soit $11/100$. De ce fait, cela lui permettra, une fois qu'une première bouteille aura été consigné, et donc une fois que la machine se sera mis à l'état B, en appuyant sur **D** et en mettant en meme temps, une deuxième consigne (donc en mettant simultanément $C=1$ et $D=1$ pour cet état B), de faire en sorte que la machine se rende ensuite directement à l'état initial A en lui rendant une pièce de 1 dollar, donc forçant ainsi la machine à lui donner deux fois plus d'argent que dans le cas normal (2 consignes donnant normalement 2 pièces de 25¢).

Pour ajouter cette fleche (ou condition) supplémentaire au diagramme de transition d'états, on doit ajouter (cf. question 2.) la ligne ▷ [ENTRÉES::ÉTATS PRÉSENTS] qui correspond à l'adresse de la mémoire, la valeur (codée en hexadécimale) ▷ [SORTIES::ÉTATS FUTURS], c'est-à-dire, à l'adresse binaire [11 01], la valeur binaire [100 00], c'est-à-dire, à l'adresse hexadécimale OD_{16}, la valeur hexadécimale 10_{16}. On peut remarquer que cette adresse hexadécimale OD_{16} était inutilisé dans le programme initial.

NOTA : L'instruction supplémentaire est une sorte de *backdoor i.e.* ou une instruction cachée qui permet à celui qui la met de tirer un avantage (qui dans ce cas serait financier) de la machine (et qui serait en fait un acte de piraterie informatique, interdit et bien sûr contraire à l'éthique d'un bon informaticien !
<6 pts>

II. Assembleur/LMC/Registres (34 pts)

1. **Programmation Machine**

   Implémenter en code d'instructions mnémoniques, avec le jeux d'instructions donné plus bas (cf. Fig. (2)) correspondant au modèle LMC vu en cours, et avec des adresses symboliques (*i.e.*, étiquettes), un programme qui code le diagramme de transition d'états de la question 1 de cet examen, (cf. Fig. 1) associé au circuit séquentiel demandé dans l'examen intra H22 (codant les différents états d'une machine automatique de remboursement de consigne).
   Dans ce modèle séquentiel d'ENTRÉES/SORTIES, On supposera que les entrées C et D sont entrés dans le programme par l'instruction IN en hexadécimal (*e.g.*, IN 0_{16}, IN 1_{16} ou IN 2_{16} correspondant respectivement à IN 00_2, IN 01_2 ou IN 10_2, *i.e.*, $[C\!=\!0, D\!=\!0]$; ou $[C\!=\!0, D\!=\!1]$; ou $[C\!=\!1, D\!=\!0]$. (On rappelle que C, la consigne est considérée comme une entrée [un capteur détecte en fait le rendu d'une bouteille et communique cette information comme étant une entrée au programme par $C = 1$]). On supposera que la sortie est communiquée par l'instruction OUT en hexadécimal (*e.g.*, OUT 3_{16} signifie OUT 011_2, *i.e.*, ($S_3 = 0_2\, S_2 = 1_2\, S_1 = 1_2$).
   Pour simplifier le code, on supposera que C et D ne peuvent être simultanément à un, on n'aura donc que CD$=\{00, 01, 10\}$.
   L'ajout de quelques commentaires à coté du programme seront appréciés afin que celui-ci soit lisible et facilement compréhensible.
   <30 pts>

LDA	5xx	Load
STO	3xx	Store
ADD	1xx	Add
SUB	2xx	Subtract
IN	901	Input
OUT	902	Output
COB or HLT	000	Coffee break (or Halt)
BRZ	7xx	Branch if zero
BRP	8xx	Branch if positive or zero
BR	6xx	Branch unconditional
DAT		Data storage location

Figure 2: Jeu d'instructions du LMC

2. Donner les avantages et inconvénients de cette méthode par rapport à un câblage par portes logiques (cf Intra H22) et par rapport à une mémoire micro-programmée (question 1 de l'examen).
   <4 pts>

Réponse

$\boxed{\textbf{1.}}$

Un exemple possible, si on interprète directement et linéairement le diagramme de transition de la Fig. 1:

On va stocker aussi les données:

		0	STOCKAGE DONNÉES
ENT0	DAT	00	
ENT1	DAT	01	
ENT2	DAT	02	
SORT0	DAT	00	
SORT1	DAT	01	
SORT2	DAT	02	
SORT3	DAT	03	
SORT4	DAT	04	
ZERO	DAT	00	
ENTRY	DAT	00	

Le programme serait:

```
           BR     A-STATE
A-INIT     LDA    SORT0
           OUT
A-STATE    IN
           STO    ENTRY
           BRZ    A-INIT
           LDA    ENTRY      | Analyse des CD≠00 pour état A
           SUB    ENT2
           BRZ    B-INIT
           BR     A-INIT     | Bouclage sur A avec 01/000
B-INIT     LDA    SORT0
           OUT
B-STATE    IN
           STO    ENTRY
           BRZ    B-INIT
           LDA    ENTRY      | Analyse des CD≠00 pour état B
           SUB    ENT2
           BRZ    C-INIT
           LDA    SORT1
           OUT
           BR     A-STATE
C-INIT     LDA    SORT0
           OUT
C-STATE    IN
           STO    ENTRY
           BRZ    C-INIT
           LDA    ENTRY      | Analyse des CD≠00 pour état C
           SUB    ENT2
           BRZ    D-INIT
           LDA    SORT2
           OUT
           BR     A-STATE
D-INIT     LDA    SORT0
           OUT
D-STATE    IN
           STO    ENTRY
           BRZ    D-INIT
           LDA    ENTRY      | Analyse des CD≠00 pour état D
           SUB    ENT1
           BRZ    C-S001-A
           LDA    SORT4
           OUT
           BR     A-STATE
C-S001-A   LDA    SORT3
           OUT
           BR     A-STATE
```

<30 pts>

NOTA : Différemment, si on interprète ce que fait ce diagramme de transition d'états, on obtient un autre exemple de progamme:

```
ENTREE     IN
           BRZ    OUT0       | Si entrée=0 => OUT0
           SUB    ONE
           BRZ    E1         | Si entrée=1 => E1
           SUB    ONE
           BRZ    E2         | Si entrée=2 => E2
           BR     ENTREE

SEE-OUT    LDA    SORTIE     | Affichage sortie
           OUT
           LDA    ZERO
           STO    SORTIE     | Re-init sortie=0
           BR     ENTREE

OUT0       LDA    ZERO       | sortie=0
           OUT               | Affichage sortie=0
           BR     ENTREE

E1         LDA    STATE      | Gestion entrée=1
           BRZ    OUT0       | si état=A ne rien faire
           LDA    ZERO
           STO    STATE      | si état=B,C,D on re-init état=0
           BR     SEE-OUT    | et affichage sortie
```

```
E2     LDA   SORTIE     | Gestion entrée=2
       ADD   ONE        |
       STO   SORTIE     | si entrée=2 incrémente sortie
       LDA   STATE      |
       SUB   THREE      |
       BRZ   E2D        | si état D=3 => E2D
       LDA   STATE      |
       ADD   ONE        |
       STO   STATE      | sinon incrémente éta
       BR    OUT0       | affichage sortie=0
```

```
E2D    LDA   ZERO       | Gestion entrée=2 & état=D=3
       STO   STATE      | on re-init état à A=0
       LDA   SORTIE     |
       ADD   ONE        | incrémente sortie
       BR    SEE-OUT    | affichage sortie
```

Avec les données:

```
                    0   | STOCKAGE DONNÉES
STATE     DAT      00   | état (A=0, B=1, C=2, D=3)
OUTPUT    DAT      00   | sortie
ZERO      DAT      00   | constante 0
ONE       DAT      01   | constante 1
TWO       DAT      02   | constante 2
THREE     DAT      03   | constante 3
```

$\boxed{\textbf{3.}}$

Comme pour le séquenceur microprogrammé, le séquenceur programmé s sur un micro-contrôleur en language machine réside majoritairement dans sa réutilisabilité et la rapidité d'élaboration (pas besoin de faire une carte électronique sur laquelle sont soudés nos composants de porte électroniques.
<4 pts>

NOTA : On pourrait prendre pour cette programmation le plus petit des microprocesseur 8 bits existant sur le marché en 2022 (pour le prix de 3.95 $ US l'unité en 2022) https://www.jameco.com/z /Z80-CPU-Major-Brands-IC-Z80-CPU-2-5MHz-DIP-40-pin_35561.html.

III. Programmation Shell Script (20 pts)

1. Implémenter un script en Bash Shell, le diagramme de transition d'états de la question 1 de cet examen, (cf. Fig. 1) associé au circuit séquentiel (codant les différents états d'une machine automatique de remboursement de consigne [celui qui fut demandé dans l'examen intra H22]).
   Dans ce modèle séquentiel d'ENTRÉES/SORTIES, On supposera que les entrées C et D sont entrés dans le programme par la commande READ qui lit une ligne de texte à partir de l'entrée standard, (*e.g.*, si la valeur lue est respectivement 0, 1, 2, ceal voudra dire que respectivement que $[C=0, D=0]$; ou $[C=0, D=1]$; ou $[C=1, D=0]$.
   Pour simplifier le code, on supposera que C et D ne peuvent être simultanément à un, on n'aura donc que CD $=\{\mathbf{00}, \mathbf{01}, \mathbf{10}\}$**.**
   <25 pts>

Réponse

$\boxed{\textbf{1.}}$

Un exemple de script pourrait être: avec output = succéssivement à 0, 1, 2, 3 ou 4 pour aucune, une, deux, trois pièces de 25 sous et une pièce de 1 dollar rendue et en interprètant directement et linéairement le diagramme de transition d'états de la Fig. 1:
<25 pts>

```
#!/bin/bash
# FSMH22.sh script [Simulateur FSM Distributeur de consigne FinalH22]

while [ : ]
do

FlagReturnInStateA=0 ; Input=0 ; Output=0
#-[A-State]------
while [ $Input -lt 2 ]
  do
    echo Etat A ; echo Entrees ? ; read Input
    echo Entrees:$Input / Sorties:$Output
  done #-End-A---

#-[B-State]------
while [ : ]
  do
    echo Etat B ; echo Entrees ? ; read Input
    if [ $Input -gt 0 ]; then break;  fi
    echo Entrees:$Input / Sorties:$Output
  done
if [ $Input -eq 1 ]; then
 Output=1
 FlagReturnInStateA=1   ####-------B-->>Retour-En--A
 echo Entrees:$Input / Sorties:$Output
fi
if [ $Input -eq 2 ]; then
 Output=0
 echo Entrees:$Input / Sorties:$Output
 FlagReturnInStateA=0
fi #-End-B-------

#-[C-State]------
if [ $FlagReturnInStateA -eq 0 ]; then
while [ : ]
  do
    echo Etat C ; echo Entrees ? ; read Input
    if [ $Input -gt 0 ]; then break;  fi
    echo Entrees:$Input / Sorties:$Output
  done
if [ $Input -eq 1 ]; then
 Output=2
 FlagReturnInStateA=1   ####------C-->>Retour-En--A
 echo Entrees:$Input / Sorties:$Output
fi
fi #-End-C-------

#-[D-State]-
if [ $FlagReturnInStateA -eq 0 ]; then
while [ : ]
  do
    echo Etat D ; echo Entrees ? ; read Input
    if [ $Input -gt 0 ]; then break; fi
    echo Entrees:$Input / Entrees:$Input
  done
if [ $Input -eq 1 ]; then
 Output=3
 echo Entrees:$Input / Sorties:$Output
fi
if [ $Input -eq 2 ]; then
 Output=4
 echo Entrees:$Input / Sorties:$Output
fi
fi #-End-D-------

done
```

NOTA -1- : Plus simplement, si on interprète ce que fait ce diagramme de transition d'états, on obtient un autre exemple de script, beaucoup plus court et plus simple:

```
#!/bin/bash
# FSMH22_2.sh script [Simulateur FSM Dist. de consigne]

state=0
while [ : ]
do
echo -n Etat ${state} :: Entrees ? ; read Input

if [ $Input -eq 0 ]; then
   echo
fi
if [ $Input -eq 1 ]; then
   echo Etat ${state} et je rend ${state} piece(s) de 25 sous
   echo
   state=0
fi
if [ $Input -eq 2 ]; then
   let state=state+1
       if [ $state -eq 4 ]; then
       state=0
       echo je reviens a Etat ${state} apres avoir rendu 1 dollar
       fi
fi
done
```

NOTA -2- : Toujours en interprètant ce que fait ce diagramme de transition d’états, et pour les fans de la structure *case of*, on peut obtenir un autre exemple de script aussi très simple:

```
#!/bin/bash
# FSMH22_3.sh script [Simulateur FSM Dist. de consigne]

state=A
while [ : ]
do

case $state in
  A)
    echo -n Etat ${state} :: Entrees ? ; read Input
    case $Input in
      0) ;;
      1) ;;
      2) state=B ;;
    esac ;;

 B)
   echo -n Etat ${state} :: Entrees ? ; read Input
   case $Input in
      0) ;;
      1) state=A; echo -n je rend 1 piece de 25 sous " " ;;
      2) state=C ;;
   esac ;;

 C)
   echo -n Etat ${state} :: Entrees ? ; read Input
   case $Input in
      0) ;;
      1) state=A; echo -n je rend 2 pieces de 25 sous " " ;;
      2) state=D ;;
   esac ;;

D)
   echo -n Etat ${state} :: Entrees ? ; read Input
   case $Input in
      0) ;;
      1) state=A; echo -n je rend 3 pieces de 25 sous " " ;;
      2) state=A; echo -n je rend 1 piece de 1 dollar " " ;;
   esac ;;

esac
done
```

NOTA -3- : ou:

```
#!/bin/bash
# FSMH22_4.sh script [Simulateur FSM Dist. de consigne]

state=0
while [ : ]
do

echo -n Etat ${state} :: Entrees ? ; read Input
if [ $Input -eq 2 ]; then
   let state=state+1
fi
if [ $Input == 1 -o $state -eq 4 ]; then
   case $state in
        0) echo "je retourne aucune piece"          ;;
        1) echo "je retourne 1 piece de 25 sous"    ;;
        2) echo "je retourne 2 pieces de 25 sous"   ;;
        3) echo "je retourne 3 pieces de 25 sous"   ;;
        4) echo "je retourne 1 pieces de 1 dollar"  ;;
   esac

let state=0
fi

done
```

NOTA -2- : Et finalement, pour les informaticiens pressés, speedés, efficaces, qui n'aiment pas le verbiage inutile (*small is beautiful*) et qui n'aime pas écrire des lignes de code inutiles qui implémentent peu de chose ;-)

```
state=0; while [ : ]; do
echo -n Etat ${state} :: Entrees ? ; read Input
if [ $Input -eq 2 ]; then let state=state+1; fi
if [ $Input == 1 -o $state -eq 4 ]; then
echo "je retourne" $state "piece(s) de 25 sous"; let state=0; fi; done
```

IV. Codes Correcteurs (15 pts)

1. **Codage de Hamming**
   On veut envoyer le message suivant 452_8 avec un code de Hamming utilisant une parité paire. Retrouver la séquence binaire que l'on doit transmettre, et la convertir en Hexa.
   <7 pts>

2. **Codage CRC**
   On veut transmettre le message DA_{16} en utilisant la méthode CRC avec le polynôme générateur $G(x) = x^5 + x + 1$. Trouver le message à transmettre et le mettre en code hexadécimal.
   <8 pts>

Réponse

$\boxed{\textbf{1.}}$

On convertit d'abord le message en binaire: 452_8 ► $100\,101\,010_2$

Le message contient 9 bits informatifs, les bits de contrôle s'insèreront en position 1 (k_1), 2 (k_2), 4 (k_3) et 8 (k_4) (puissance de deux) de la façon suivante:

$$1\;0\;0\;\;1\;0\;k_4\;1\;\;0\;1\;k_3\;\;0\;k_2\;k_1$$

Utilisons la méthode de Hamming simplifiée qui consiste à calculer la somme modulo 2 (car la parité est paire dans cet exemple) des positions de bits à 1. La dernière ligne considérée étant la valeur de k_4, k_3, k_2, k_1. On obtient donc:

$$
\begin{array}{cccccc}
13 & \rightarrow & 1 & 1 & 0 & 1 \\
10 & \rightarrow & 1 & 0 & 1 & 0 \\
07 & \rightarrow & 0 & 1 & 1 & 1 \\
05 & \rightarrow & 0 & 1 & 0 & 1 \\
\hline
k_i & \rightarrow & \underbrace{0}_{k_4} & \underbrace{1}_{k_3} & \underbrace{0}_{k_2} & \underbrace{1}_{k_1}
\end{array}
$$

On obtient donc le message binaire à transmettre suivant; $1\ \ 0010\ \ 0101\ \ 1001_2\ \ (= 1259_{16})$

<7 pts>

$\boxed{\textbf{2.}}$

• On convertit d'abord le message en binaire: DA_{16} ▶ $1101\,1010_2$

• Après avoir ajouté 5 zéros au message (5 correspondant au degré du polynôme générateur), on réalise ensuite la division modulo 2 suivante:

$$
\begin{array}{ccccccccccccc|cccccccc}
1&1&0&1&1&0&1&0&0&0&0&0&0&1&0&0&0&1&1&&\\ \cline{14-19}
1&0&0&0&1&1&&&&&&&&1&1&0&1&0&0&0&1\\ \cline{1-6}
0&1&0&1&0&1&1&&&&&&&&&&&&&&\\
&1&0&0&0&1&1&&&&&&&&&&&&&&\\ \cline{2-7}
&0&0&1&0&0&0&0&0&&&&&&&&&&&&\\
&&&1&0&0&0&1&1&&&&&&&&&&&&\\ \cline{4-9}
&&&0&0&0&0&1&1&0&0&0&0&&&&&&&&\\
&&&&&&&1&0&0&0&1&1&&&&&&&&\\ \cline{8-13}
&&&&&&&0&1&0&0&1&1&&&&&&&&
\end{array}
$$

• On transmettra donc le message $\underbrace{1\ 1011\ 010}_{M(x)}\ \underbrace{1\ 0011}_{R(x)} = 1B53_{16}$.

<8 pts>

V. Misc. (05 pts)

Répondre aux questions suivantes:

1. Expliquez pourquoi la mémoire cache qui est aussi d'autant plus proche du processeur est d'autant plus petite. **<5 pts>**

Réponse

$\boxed{\textbf{1.}}$

Techniquement, plus la mémoire cache est proche du processeur et plus celle-ci est rapide, et la plus difficile à réaliser techniquement (souvent la gravure est plus petite ou elle est faite dans des composants qui lui assurent une plus grande rapidité et une meilleure performance) et de ce fait, aussi, plus elle est proche du processeur, plus celle-ci est couteuse.

En fait, dans un programme, on peut remarquer que 70%-90% du temps du programme concernent 10%-30% du code qui sont en fait des boucles dans les structures de données du type vecteur ou tableau n dimensions. Lorqu'on veut (ou lorsque le compilateur veut) optimiser efficacement ce code, il essaiera, autant que possible, de faire traiter ces 10%-30% du code dans une mémoire cache qui est plus proche du processeur (et plus petite) que celle utilisée pour le reste du programme.

<5 pts>