Université Joseph Fourier – Grenoble 1
ALM MIAGE L3
**Examen d'Architecture des ordinateurs - première session**
Décembre 2016
Durée : 3h00 en tout (~1h00 Architecture des ordinateurs, ~1h Système, ~1h Réseau).
1 document A4 commun ASR autorisé. Sujet Architecture des ordinateurs sur 1 recto-verso.

## I. Sur les automates de Moore
**(barème indicatif : 1.5 points)**

**Q1.** Donner un automate de Moore déterminant le nombre de bits à 1 parmi les deux dernières valeurs lues sur l'unique entrée. (un dessin avec les états↑sorties et les flèches de transition suffit, inutile de donner le circuit)
**Q2.** Dénombrer le nombre maximum d'automates de Moore réalisables avec 4 bits de mémoire, 5 bits d'entrée et 6 bits de sortie. Rappel : un automate est défini par l'association d'une fonction de transition et d'une fonction de sortie. Il suffit donc de dénombrer le nombre de possibilités pour chacune de ces deux fonctions et de les combiner afin de trouver le nombre d'automates possibles. Donner et expliquer une formule de calcul, évaluer sa valeur en binaire et en décimal.

## II. Algorithme de Luhn
**(barème indicatif : 2.5 points)**

*« L'algorithme de Luhn est une simple formule de somme de contrôle utilisée pour valider une variété de numéros de comptes, comme les numéros de cartes bancaires. Elle fut développée dans les années 1960 par un ingénieur allemand de chez IBM, Hans Peter Luhn, comme méthode de validation d'identification de nombres. Elle n'a pas été conçue pour être une fonction de hachage sécurisée cryptologiquement ; elle protège contre les erreurs aléatoires. »*
D'après l'article Formule de Luhn de Wikipédia.

Dans cet exercice, nous utiliserons un algorithme adapté aux nombres binaires très librement inspiré de l'algorithme de Luhn :
En entrée, un nombre A sur 16 bits ($A_{16}$ … $A_0$), et un entier S sur 2 bits ($S_1S_0$), initialisées à 0,
En sortie, le nombre S sur 2 bits.

1. Prendre le nombre constitué des 2 bits de poids faibles ($A_1A_0$) et le multiplier par 2, si le résultat R dépasse 3 (R>3), prendre R ←7-R
2. Ajouter R à S, ne conserver que les 2 bits de poids faible
3. Ajouter le nombre constitué des 2 bits ($A_3A_2$) à S
4. Diviser A par 16
5. Recommencer à partir de l'étape 1. (répéter 3 fois, i.e. jusqu'à ce que A soit nul)

**Q1.** Définir un circuit combinatoire L réalisant l'étape 1. avec A en entrée et R en sortie. Donner le schéma de ce circuit avec des portes et/ou/non.
**Q2**. Réaliser l'ensemble de l'algorithme avec un circuit <u>combinatoire</u> (utiliser autant de cellules L définie en Q1, et d'additionneur 2 bits que nécessaire)
**Q3**. Réaliser l'ensemble de l'algorithme de Luhn avec un circuit <u>séquentiel</u> n'utilisant que 2 additionneurs 2 bits et une cellule L.

## III. Test and Set
**(barème indicatif : 3 points)**

*« L'instruction test-and-set est une instruction atomique utilisée en informatique pour écrire dans un emplacement mémoire et retourner la valeur d'origine de cet emplacement. Elle permet de protéger un espace de la mémoire en cas d'accès concurrents : si plusieurs processus tentent d'accéder à la même mémoire et si un processus est en train d'effectuer un test-and-set sur cette même mémoire, alors aucun autre processus ne peut commencer un autre test-and-set jusqu'à ce que le premier processus soit terminé. Exemple d'implémentation en C :*

```
int test_and_set (int *verrou){
   int old = *verrou;
   *verrou = 1;
   return old;}
```

» D'après l'article Test-and-Set de Wikipédia.

Deux processus ($P_0$ et $P_1$) sont en activité sur une machine décrite par l'automate d'interprétation et la mémoire donnée plus loin. Le jeu d'instructions de cette machine n'est pas présenté dans sa totalité pour simplifier le problème posé. Chaque processus possède un environnement d'exécution (contexte propre

comportant les registres PC, IR, ACC, AUX, TIR, … : le processus $P_0$ possède le contexte $PC_0$, $IR_0$, $ACC_0$, $AUX_0$, $TIR_{0,\ \ldots;}$ le processus $P_1$ possède le contexte $PC_1$, $IR_1$, $ACC_1$, $AUX_1$, $TIR_{1,\ldots}$). $P_0$ et $P_1$ partagent la même mémoire (il n'y a qu'une version de la mémoire). Le système d'exploitation réparti le temps d'exécution entre ces deux processus à l'aide d'un mécanisme non explicité ici permettant de basculer de l'un à l'autre des processus (par bascule entre les deux environnements d'exécution). Cette bascule peut se faire uniquement au début de l'étape « fetch » de l'automate de contrôle.

**Q1.** Simuler l'exécution de la machine en supposant que les deux processus sont au départ avec PC=0 ; ACC = 1 ; AUX = 2 ($PC_0$=0, $ACC_0$=1, $AUX_0$=2 ; $PC_1$=0, $ACC_1$=1, $AUX_1$=2) ; que le système d'exploitation alterne entre les 2 processus à chaque instruction assembleur, i.e. à chaque passage à l'étape « fetch » le système alterne entre le processus $P_0$ et le processus $P_1$, la machine commence avec $P_0$ (déroulement : $P_0$ ; $P_1$ ; $P_0$ ; $P_1$ ; $P_0$ ; …). Exécuter 2 instructions assembleur pour chaque processus.

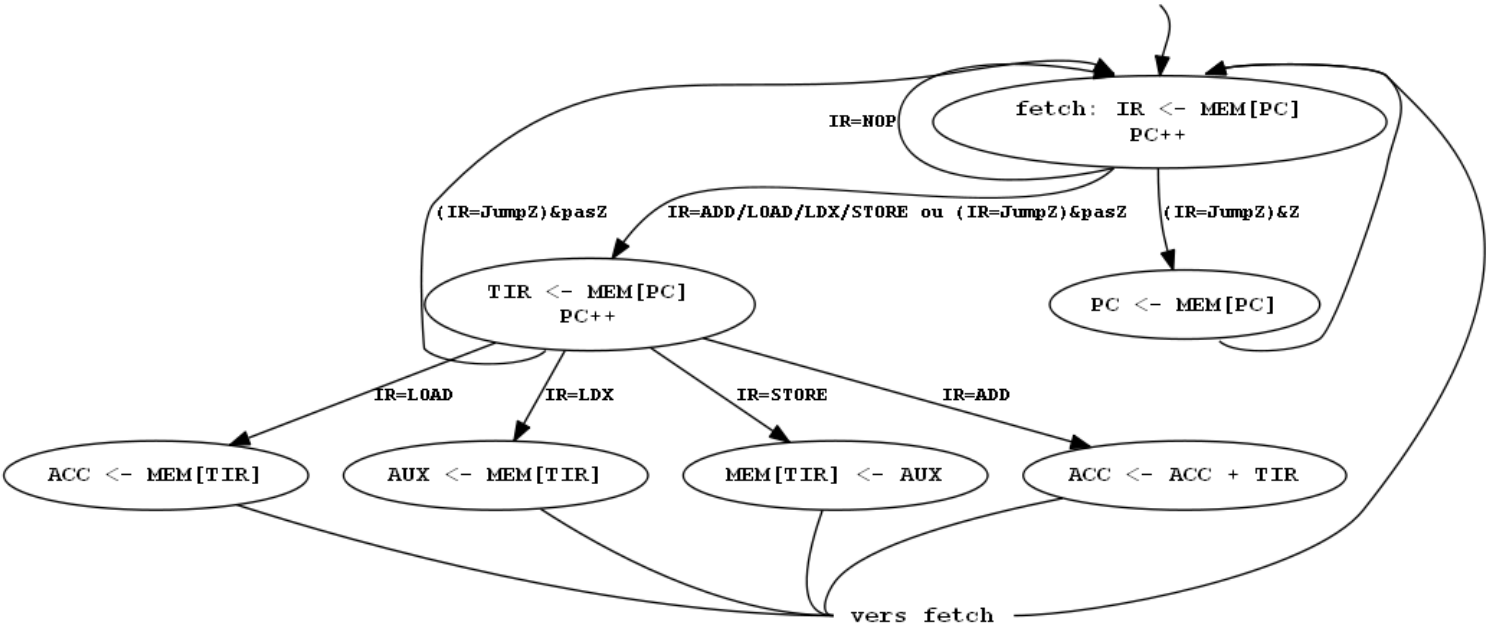
**Q2.** Est-ce que le résultat de l'exécution aurait été différent si le système d'exploitation avait alterné entre les 2 processus non pas à chaque instruction, mais toutes les deux instructions, i.e. deux fois moins souvent (déroulement : $P_0$ ; $P_{0\,;}$ $P_{1\,;}$ $P_1$ ; $P_0$ ; $P_0$ ; …)

**Q3.** Pour réaliser une action de type « test-and-set », modifier le graphe de l'automate de contrôle pour introduire une instruction LoadAndStore qui effectue en un seul cycle, de fetch à fetch, l'équivalent des deux instructions Load suivie de Store, i.e. l'instruction LoadAndStore @ doit faire l'équivalent de Load @, suivie de Store @ sans repasser par fetch.

Mémoire :

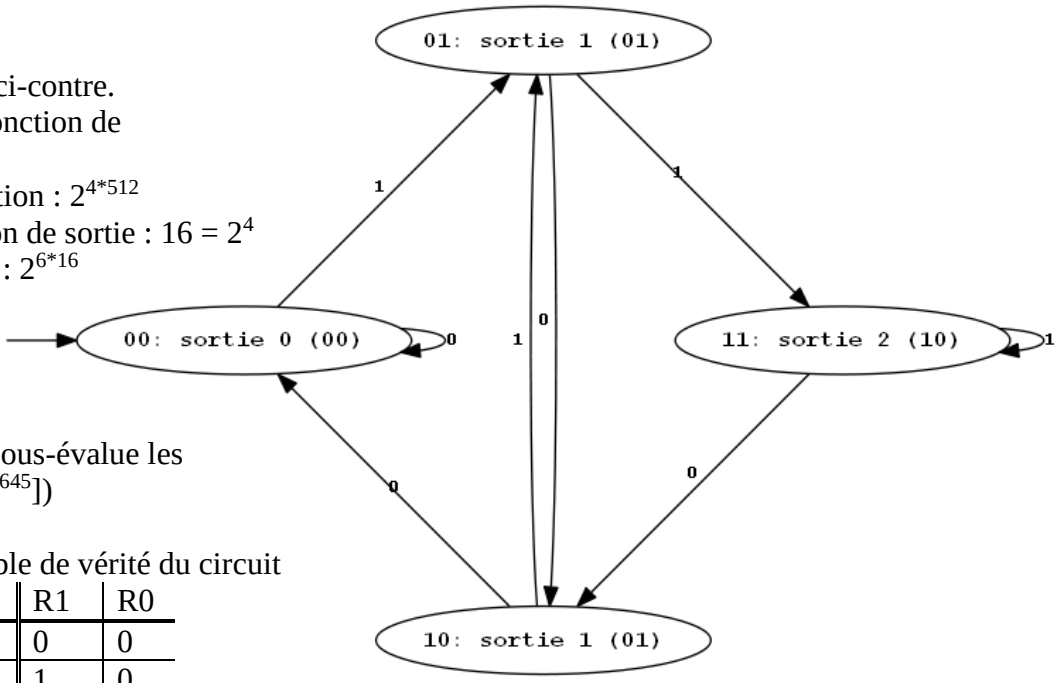
| Adresse | Contenu |
|---|---|
| 00 | LOAD |
| 01 | 0A |
| 02 | STORE |
| 03 | 0A |
| 04 | XXX |
| 05 | 08 |
| 06 | YYY |
| 07 | 07 |
| 08 | 06 |
| 09 | 05 |
| 0A | 04 |
| 0B | 03 |
| 0C | … |

Automate de contrôle :



Eléments de correction

**Automate de Moore. Q1.** Cf ci-contre.

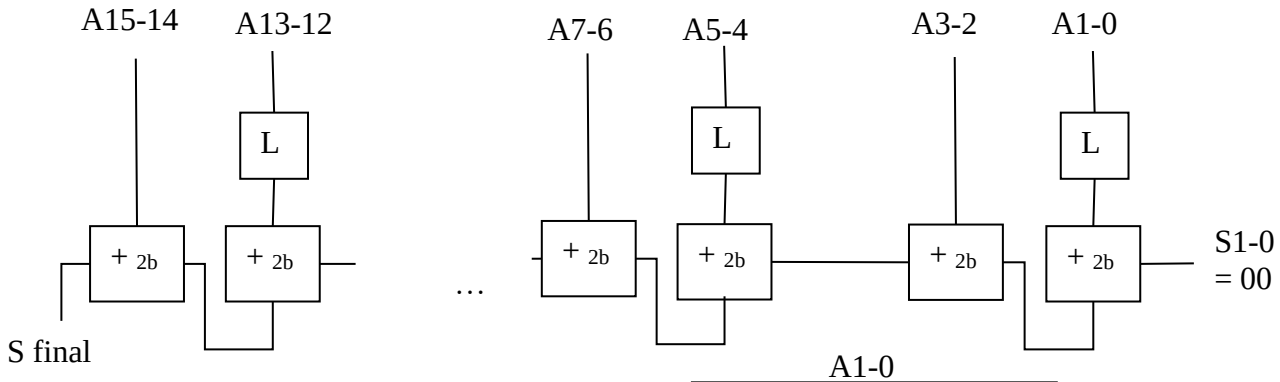



**Q2.** Nombre de lignes de la fonction de transition : $512 = 2^{4+5}$
Nombre de fonctions de transition : $2^{4*512}$
Nombre de lignes de la fonction de sortie : $16 = 2^4$
Nombre de fonctions de sortie : $2^{6*16}$
Nombre max d'automates :
$2^{4*512} * 2^{6*16} = 2^{4*512+6*16} =$
$2^{2144}$ (de l'ordre de $10^{644}$
[avec les grands nombres, l'approximation « $2^{10} = 10^3$ » sous-évalue les nombres ; le résultat est $2.* 10^{645}$])

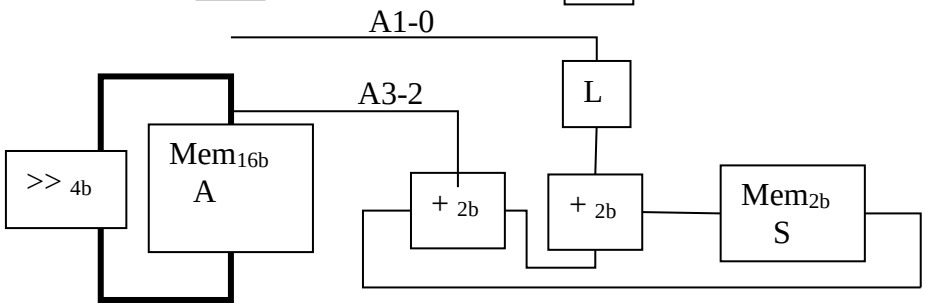
**Algorithme de Luhn**. **Q1.** Table de vérité du circuit

| A1 | A0 | R1 | R0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |

**Q2**. (chaque nappe de fils comporte deux fils)




**Q3**. (en gras des nappes de 16 fils)




**Test and Set. Q1.**

| $PC_0$ | $IR_0$ | $ACC_0$ | $AUX_0$ | $TIR_0$ | $PC_1$ | $IR_1$ | $ACC_1$ | $AUX_1$ | $TIR_1$ | MEM[0A] |
|---|---|---|---|---|---|---|---|---|---|---|
| 00 | ? | 01 | 02 | ? | 00 | ? | 01 | 02 | ? | 04 |
| | LOAD | | | | | | | | | |
| 01 | | | | | | | | | | |
| | | | | 0A | | | | | | |
| 02 | | | | | | | | | | |
| | | 04 | | | | | | | | |
| | | | | | | LOAD | | | | |
| | | | | | 01 | | | | | |
| | | | | | | | | | 0A | |
| | | | | | 02 | | | | | |
| | | | | | | | 04 | | | |
| | STORE | | | | | | | | | |
| 03 | | | | | | | | | | |
| | | | | 0A | | | | | | |
| 04 | | | | | | | | | | |
| | | | | | | | | | | 02 |
| | | | | | | STORE | | | | |
| | | | | | 03 | | | | | |
| | | | | | | | | | 0A | |
| | | | | | 04 | | | | | |
| | | | | | | | | | | 02 |

**Q2.** Avec cette exécution, à la fin, $ACC_1$ a la valeur 02.

**Q3.** LoadAndStore peut emprunter les transitions de Load (remplacer Load par Load/LoadAndStore dans les transitions de l'automate), puis, avant de retourner à fetch passer par la fin du Store (ajouter une transition (ACC<MEM[TIR]) ➔ (MEM[TIR]<AUX)).