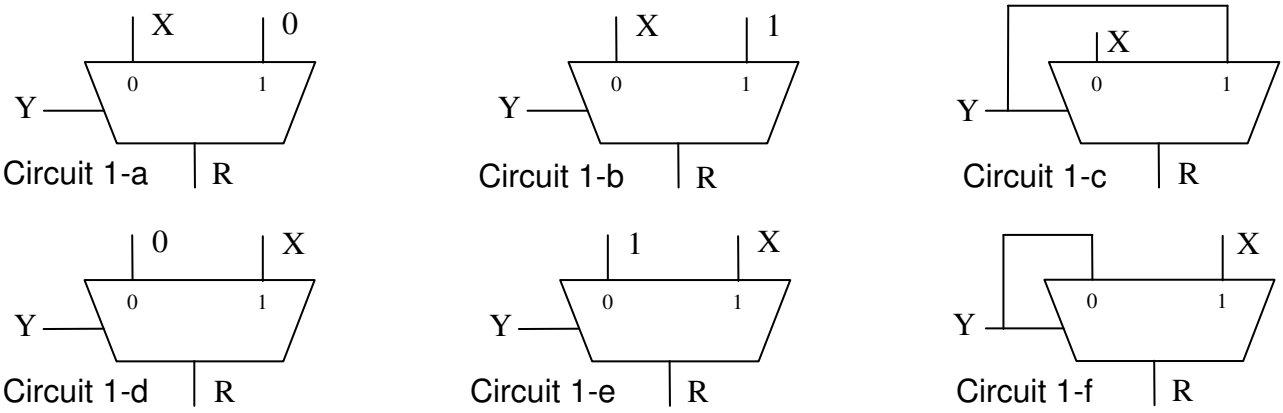
Université Joseph Fourier – Grenoble 1
Architecture des ordinateurs - MIAGE L3
Examen d'Architecture des ordinateurs - première session
Décembre 2015
Durée : 3h00 en tout (~1h00 Architecture des ordinateurs, ~1h Système, ~1h Réseau).
1 document A4 commun ASR autorisé. Sujet Architecture des ordinateurs sur 1 recto-verso.

1. Circuits combinatoires utilisant un multiplexeur (barème indicatif sur 7 points : 2 points)

Donnez les tables de vérité des circuits suivants et à partir de ces tables. Identifiez les fonctions « simples » associées, donnez leur schéma. Vous pouvez utiliser la symétrie des schémas pour gagner du temps.

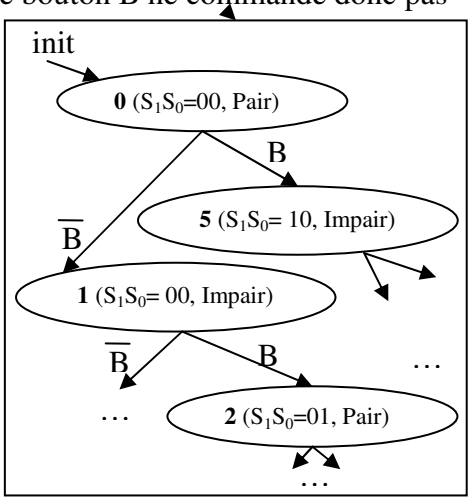


2. Boite à rythme binaire (barème indicatif sur 7 points : 3 points)

Objectif de l'exercice : **réaliser le circuit logique d'une mini boite à rythme binaire**.
L'objet prévu comporte une horloge H dont la fréquence peut être réglée par l'utilisateur et un haut parleur HP relié à deux sources de sons distinctes définies par l'utilisateur. Vous n'avez pas à vous occuper de ces parties du circuit. Vous considèrerez seulement que parmi les entrées de votre circuit vous avez un signal d'horloge H régulier et que parmi les sorties de votre circuit, vous devez fournir deux signaux S_0 et S_1 indiquant s'il faut jouer les sons 0 et 1.

Le rythme recherché étant binaire, il faut considérer séparément les sons produits sur les temps pairs et les sons produits sur les temps impairs. Un bit doit suffire pour mémoriser si le système est sur un temps pair ou sur un temps impair. Pour cette boite à rythme, l'objectif est de pouvoir jouer (éventuellement) le son S_0 sur les temps pairs et de pouvoir jouer (éventuellement) le son S_1 sur les temps impairs.

Pour simplifier, vous pouvez supposer que l'horloge commence sur un temps pair. Au début de chaque temps (à la bascule de l'horloge), l'utilisateur de la boite à rythme peut décider s'il joue le son du temps ou pas à l'aide d'un bouton unique B. B est donc une entrée du dispositif. Au départ, aucun son n'est joué, le temps pair est donc silencieux. Si l'utilisateur appuie sur le bouton B au début du premier temps suivant alors le son S_1 du temps impair est joué. Si l'utilisateur appuie sur le bouton B au début du temps suivant (temps 2, pair), le son S_0 du pair est joué. S'il continue d'appuyer, les sons vont s'éteindre aux temps suivant (3 et 4), puis être à nouveau joué, et encore. Et ainsi de suite. Le bouton B ne commande donc pas directement les sons, mais inverse la commande des sons. Lorsque l'utilisateur n'appuie plus sur le bouton B, les séquences de sons joués se répètent à l'identique.
4 séquences différentes sont possibles : séquence avec aucun son ; avec seulement le son S_0 sur les temps pairs ; avec seulement le son S_1 sur les temps impairs ; avec les deux sons alternativement : le son S_0 sur les temps pairs, le son S_1 sur les temps impairs.



Q2-a. Modélisation. Modélisez par un automate **ou** par un programme le comportement de la boite à rythme. Expliquez vos choix, les éléments de votre modélisation.
Pour une modélisation par automate, vous pouvez complétez le diagramme ci-contre indiquant l'arbre des possibles en veillant à ne pas créer un nouvel état s'il y a un état identique déjà créé.
Pour une modélisation par programme, identifiez les variables du système parmi les entrées (H, B), les sorties (S_0, S_1) et les variables internes (Parité), puis transcrivez par programme les changements de valeurs.

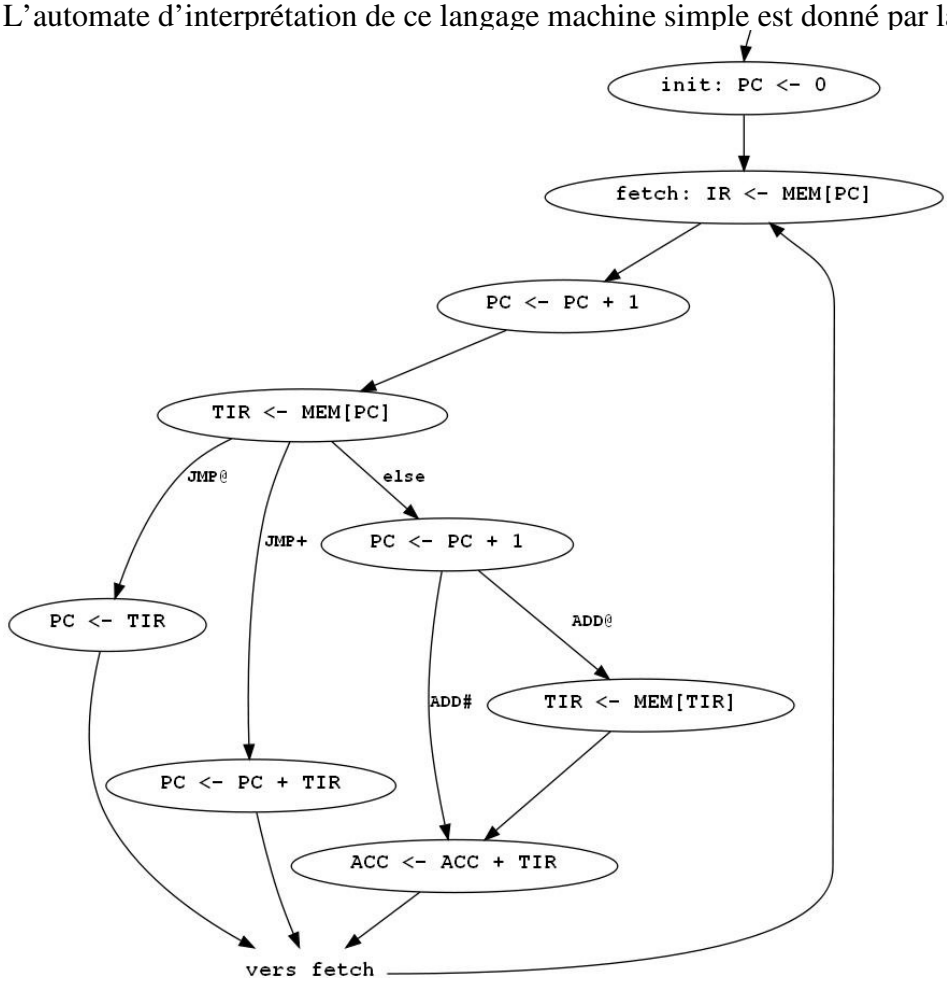
Q2-b. Réalisation. Donnez un circuit réalisant votre boite à rythme, ou une partie significative. Par exemple, pour une modélisation par automate : la table de transition, une formule logique et le dessin du circuit d'un bit du registre d'état, ainsi qu'un dessin global du circuit peut suffire.

3. Automate de contrôle (barème indicatif sur 7 points : 2 points)

L'automate d'interprétation d'un langage machine simple comportant seulement 2 instructions ADD et JMP, avec 2 modes d'adressage pour chaque instruction :

- Addition immédiate, ADD# codée sur un octet avec la valeur immédiate de l'addition sur l'octet suivant.
- Addition directe, ADD@) codée sur un octet avec l'adresse de l'opérande sur l'octet suivant.
- Saut absolu, JMP@ codé sur un octet avec la adresse absolue de branchement sur l'octet suivant.
- Branchement relatif, JMP+ codé sur un octet avec la valeur immédiate de déplacement par rapport à l'adresse PC courante stockée sur l'octet suivant.

Exemples : Add#3 signifie `ACC <- ACC + 3`, Add@3 signifie `ACC <- ACC + MEM[3]`, JMP@3 signifie `PC <- 3`.
L'automate d'interprétation de ce langage machine simple est donné par la figure suivante :



@	MEM
00	10 (= ADD#)
01	16
02	11 (= ADD@)
03	0D
04	41 (= JMP+)
05	FB
06	40 (= JMP@)
07	00
08	10 (= ADD#)
09	02
0A	00
0B	01
0C	02
0D	03
0E	04
…	…

Q3-a. Reprenez l'exécution du programme suivant qui est arrivée à l'étape « fetch » de l'automate avec les registres PC = 02 ; ACC = 30. Continuez cette exécution pour 3 instructions écrites en assembleur, en détaillant chaque étape de l'exécution de l'automate.

La partie opérative prévue pour l'automate comportait des registres constants valant 0 et 1 ainsi qu'une sélection fine des opérandes des opérations de l'UAL (Unité Arithmétique et Logique : +, -, opérande gauche, …) qui permettait `PC<-0`, `PC<-PC+1`, `PC<-PC+TIR`, … Cette partie opérative a été réduite. Les registres constants ont disparus, à la place deux fonctions constantes ont été ajoutées à l'UAL: la fonction constante 0 qui donne toujours 0, et la fonction constante 1. La sélection des opérandes des opérations a été simplifiée aussi, la seule opération binaire possible s'effectue avec un registre fixé à l'avance TMP (nouveau registre) comme second opérande pour n'avoir qu'un seul registre à indiquer globalement. La seule addition possible est donc du type : `Reg<-Reg+TMP` où `Reg` = `TMP, TIR, ACC` ou `PC` mais `Reg` est identique à gauche et à droite. Idem pour la soustraction. Les opérations unaires `Reg<-TMP` ou `TMP<-Reg`, l'accès à la mémoire `Reg<-MEM[Reg]` (avec `Reg` identique à gauche et à droite) ou `Reg<-MEM[TMP]` et `TMP<-MEM[Reg]` et les nouvelles opérations constantes `Reg<-0` et `Reg<-1` sont également possibles.

Q3-b. Transformez l'automate d'interprétation en fonction des ces changements pour qu'il puisse toujours exécuter le programme donné en mémoire, avec le même codage pour ADD(#/@), JMP(@/+).

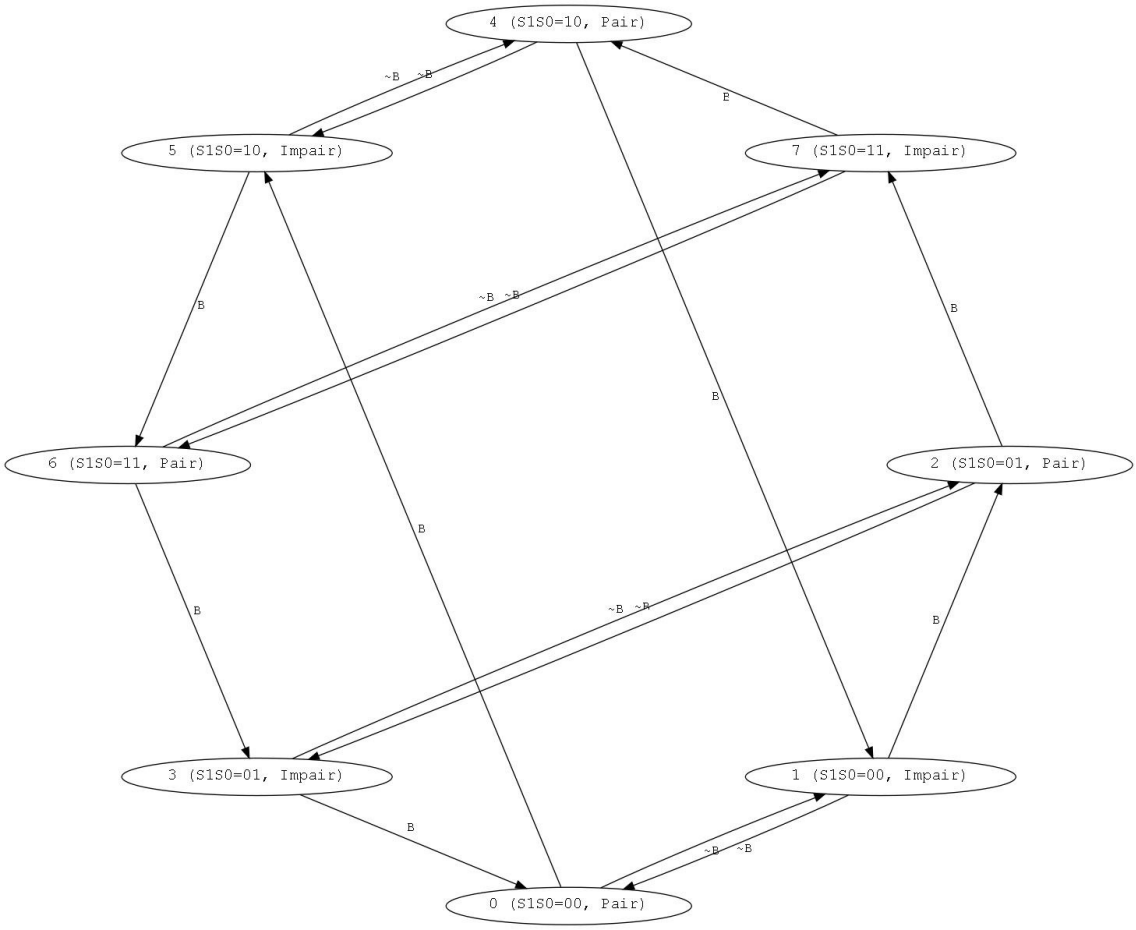
Éléments de corrections

Exercice 1. Circuits avec Multiplexeurs

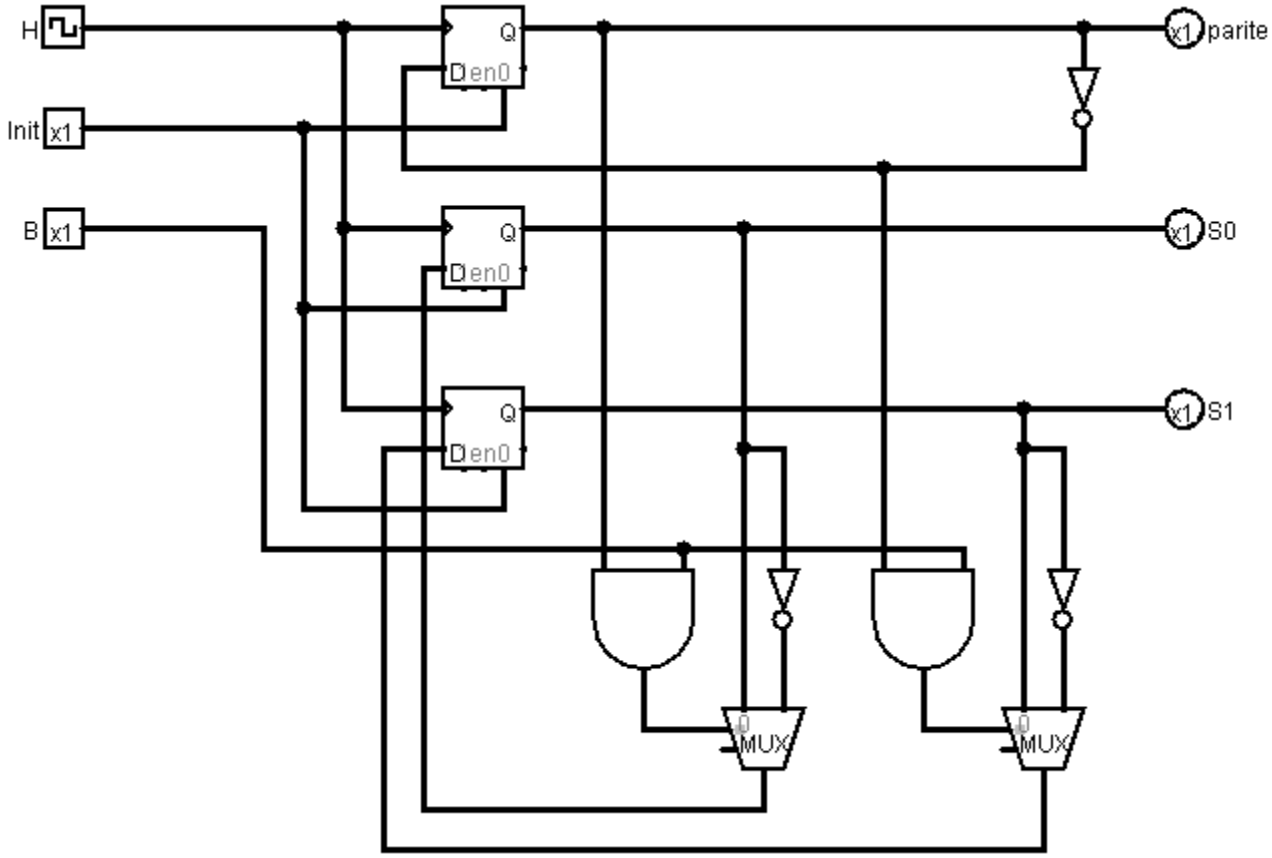
X	Y	R (1-a) = X et non Y = X > Y	R (1-b) = X ou Y	R (1-c) = X ou Y	R (1-d) = X et Y	R (1-e) = X ou nonY = X < Y	R (1-f) = X et Y
0	0	0	0	0	0	1	0
0	1	0	1	1	0	0	0
1	0	1	1	1	0	1	0
1	1	0	1	1	1	1	1

Exercice 2. Boite à rythme
2-a
Modélisation par programme :
Début
S0 ← Faux ;
S1 ← Faux ;
Parité ← Pair ;
Tant que vrai :
| Si (Parité = Pair) et B alors
| | S1 ← non S1
| finsi
| Si (Parité = Impair) et B alors
| | S0 ← non S0
| finsi
| Parité ← inverse (Parité)
fintq
Fin.

Modélisation par automate :



3-b, le circuit en découle (à partir du programme)



Exercice 3. Automate de contrôle

3-a

T	pc	ir	tir	acc
Init 1 2 3 4 5 6	02 03 04	 11 (add@)	 0D 03	30 33
7 8 9 10	 05 00	41 (jmp+)	 FB	
11 12 13 14 15	 01 02	10 (add#)	 16	 49

3-b, Il faut remplacer l'état `PC<-PC+1`, une solution consiste à introduire 2 états `TMP<-1`, puis `PC<-PC+TMP`.
Pour `PC<-TIR`, une solution consiste à introduire 3 états `TMP<-TIR` puis `PC<-0` et enfin `PC<-PC+TMP`.
Pour `PC<-PC+TIR`, une solution consiste à introduire 2 états `TMP<-TIR` puis `PC<-PC+TMP`.
Pour `IR<-MEM[PC]`, une solution consiste à introduire 2 états `TMP<-PC` puis `IR<-MEM[TMP]`.
Il reste aussi à remplacer `ACC<-ACC+TIR` et `TIR<-MEM[PC]` de la même manière.