Circuits Numériques et Éléments d'Architecture
Examen

ENSIMAG 1A

Année scolaire 2018–2019

— durée : 3h ;
— résumé sur feuille A4 manuscrite et calculatrices autorisés ;
— le barème est donné à titre indicatif ;
— les exercices sont indépendants et peuvent être traités dans le désordre, et certaines questions au sein du même exercice peuvent aussi être traitées indépendamment ;
— pensez à indiquer votre **nom** et **numéro** de groupe sur chacune de vos copies.
— les annexes à compléter sont à rendre avec votre copie. N'oubliez pas de le faire et d'y noter votre nom et numéro de groupe. D'ailleurs, faites-le tout de suite, ça sera fait !

Ex. 1 : Questions de cours (2.5 pts)

Question 1 On veut réaliser un multiplexeur $8 \rightarrow 1$ encodé réalisé à l'aide de multiplexeurs $2 \rightarrow 1$ encodés.

(i) donnez le nombre de bits nécessaire au signal de sélection ;
(ii) dessinez le schéma du circuit.

Question 2 Pourquoi est-il préférable de recopier la valeur (en rebouclant avec un multiplexeur) plutôt que de « masquer » l'horloge (avec une porte « and ») pour réaliser l'autorisation d'écriture dans une bascule ?

(i) parce qu'un multiplexeur est plus petit qu'une porte logique « and » ;
(ii) parce que la recopie consomme moins d'énergie que le masquage de l'horloge ;
(iii) parce qu'un front non-synchronisé peut se produire lors du passage à 1 du signal d'autorisation d'écriture en cas de masquage.

Question 3 Précisez la différence entre un automate ayant uniquement des signaux de Moore et un automate ayant également de signaux de Mealy.

Question 4 Soit un cache de 4k octets (à correspondance directe, celui exposé en cours) possédant des lignes de 16 mots, sachant qu'un mot est constitué de 4 octets. Combien de lignes contient ce cache ? En supposant que l'adresse est sur 32 bits, sur combien de bits sont codés (a) le champ *offset*, (b) le champ *index* et (c) le champ *tag* ?

Ex. 2 : Circuits combinatoires (2.5 pts)

On cherche à réaliser le circuit combinatoire à deux sorties suivant :

— la sortie s_0 est vraie (1) si un nombre $n \in [\![0\,;\,15]\!]$ n'appartient pas à la séquence d'entiers produite par $m \in \{1,\ldots,8\}$, $\lfloor m/2 \rfloor \times \lceil m/2 \rceil$, elle est fausse (0) sinon [1] ;
— la sortie s_1 est vraie (1) si un nombre $n \in [\![0\,;\,11]\!]$ n'appartient pas à la séquence des nombres premiers [2].

La table de vérité correspondante est donnée ci-dessous.

1. *c.f.* https://oeis.org/A049068
2. *c.f.* https://oeis.org/A005171

Question 1 Sur combien de bits est codée l'entrée n ? On notera e_i le bit en position i de l'entrée.

Question 2 En tenant compte du fait que la fonction est incomplètement spécifiée pour s_1 et en utilisant des tables de Karnaugh, donnez les expressions simplifiées des s_i sous forme canonique (sommes de produits). On ne *demande pas* de schémas en portes logiques des fonctions booléennes calculées.

n	s_1	s_0
0	1	0
1	1	0
2	0	0
3	0	1
4	1	0
5	0	1
6	1	0
7	0	1
8	1	1
9	1	0
10	1	1
11	0	1
12	-	0
13	-	1
14	-	1
15	-	1

Ex. 3 : Synthèse d'automate (4 pts)

On veut contrôler un store qui produit de l'ombre sur une terrasse de café, pour assurer le confort et la sécurité des consommateurs. Le store est doté de trois capteurs. Le premier est un capteur solaire `s` qui indique, s'il vaut 1, qu'il faut dérouler le store. Le second est un anémomètre `a` (capteur de vitesse du vent), qui s'il est à 1, indique qu'il faut enrouler le store et le laisser enroulé, indépendamment de la présence de soleil. Le dernier est un taquet `t` qui indique que le store est complètement enroulé ou complètement déroulé, auquel cas il faut cesser d'actionner le store. Si le vent se lève alors que le store se déroule, l'opération est interrompue et le store se rétracte. Le store reste déroulé tant qu'il y a du soleil, si le vent le permet.
Le circuit dispose d'une sortie sur 2 bits (qui typiquement commande le moteur du store) qui a la signification suivante :

o_1	o_0	signification
0	0	ne fait rien
0	1	enroule
1	-	déroule

L'automate permettant de réaliser la commande du store possède 4 états :

— ENROULÉ, état initial de l'automate, qui indique que le store est totalement enroulé. Dans ce cas là, le taquet `t` est à 1, et passe à 0 dès que l'on sort de l'état ;

— EXTENSION, qui indique que le store est *en train* de se dérouler. À ce moment là, on ne se préoccupe plus de la présence de soleil tant que le store n'est pas complètement déroulé. En revanche, si le vent se lève le store se rétracte immédiatement ;

— RANGEMENT, qui indique que le store est *en train* de s'enrouler. Il finit son action indépendamment de la présence de soleil et de l'absence de vent ;

— DÉROULÉ, qui indique que le store est totalement déroulé. Dans ce cas là, le taquet `t` est à 1, et passe à 0 dès que l'on sort de l'état.

Question 1 Construisez le graphe de transitions de la machine d'état, en notant sur les arcs les conditions de transitions.

On code les états comme suit : ENROULÉ (00), EXTENSION (10), RANGEMENT (11), DÉROULÉ (01), On note q_i les bits du registre contenant l'état.

Question 2 Donnez la table de transition exprimant les d_i du registre d'état et des sorties o_1 et o_0 en fonction des q_i et des entrées booléennes `s`, `a`, et `t`. *On ne demande pas les expressions simplifiées des différents signaux.*

Ex. 4 : Conception PC/PO d'un circuit de calcul de l'exponentiation modulaire (6 pts)

L'algorithme écrit en `python` ci-dessous[3] permet d'effectuer l'exponentiation modulaire. Étant donnés une base b, un exposant e et un entier m (tous trois nombres positifs par hypothèse), il calcule efficacement $c \equiv b^e \pmod{m}$. La taille des variables est implicite, nous supposerons les opérateurs « assez grands » pour effectuer ces calculs sans troncature.

```
1  import sys
2
3  def expmod(base, exp, m):
4      result = 1
5      while exp > 0:
6          if (exp & 1) > 0:
7              result = (result * base) % m
8          exp = exp >> 1
9          base = (base * base) % m
10     return result
```

- on ne cherchera pas à comprendre ce que fait cet algorithme, on le prendra donc comme une entrée de l'exercice sans plus se préoccuper de sa fonction ;
- l'opération `v >> p` décale le mot binaire `v` à droite de `p` positions, les `p` bits de poids faible étant perdus et `p` '0' étant injectés à gauche ; il s'agit donc d'un décalage *logique* à droite ;
- l'opérateur `a & b` effectue le « et » bit à bit entre les mots binaires `a` et `b` ;
- l'opérateur $*$ est la multiplication entière classique, et l'opérateur `%` calcule le reste de la division entière (le modulo).

Pour mémoire, des instructions qui peuvent s'effectuer en parallèle peuvent être écrites sous forme d'affectations concurrentes (ex. $(a,b) = (x,y)$), et l'on peut aussi affecter conditionnellement une variable (ex. $n = x > y ? 1 : 0$).

Question 1 Proposez une implantation efficace de l'opération `exp = exp >> 1` et de l'opération `exp & 1`.

On cherche à optimiser le temps d'exécution des lignes 6 à 8 en faisant l'hypothèse que l'on a autant d'opérateurs que l'on veut. À cet effet, nous allons dans un premier temps nous intéresser à faire disparaître le **`if`** ligne 6.

Question 2 On considère que l'expression `(exp & 1) > 0` retourne un booléen qui vaut 1 si elle est vraie et 0 sinon. Exploitez cette information pour faire disparaître le **`if`** et écrire les lignes 6 et 7 comme une seule ligne (que nous appellerons 7 dans la suite).

On veut dans un second temps exécuter en parallèle autant de lignes que possibles dans le corps de la boucle.

Question 3 Peut-on paralléliser les lignes 7, 8 et 9 ? Justifiez en deux mots (c'est une allégorie, ...). Utilisez la syntaxe de l'affectation concurrente pour montrer le degré de parallélisation possible.

Étant donné la complexité des opérateurs $*$ et `%`, on se dote en réalité *d'un seul* de chacun de ces opérateurs dans la suite de l'exercice (*i.e.* à partir de maintenant !).

Question 4 Réécrivez la parallélisation du code des lignes 7, 8 et 9 en prenant en compte cette nouvelle contrainte.

Vous allez maintenant construire la partie opérative du circuit réalisant cet algorithme.

3. Basé sur le code présenté par Bruce Schneier dans son livre *Applied Cryptography*.

Question 5

— listez les registres du circuit ;

— listez les opérateurs du circuit et leur nombre ;

— dessinez le schéma incluant les opérateurs, les multiplexeurs et les interconnexions. Les opérateurs non-standard seront simplement représentés comme des boîtes avec le bon nombre d'entrées/sorties.

Question 6 Proposez un automate d'états qui pilote cette partie opérative sous forme de graphe d'état (au sein de chaque état on précisera le nom et la description RTL). Spécifiez la valeur des différents signaux de commande des registres et des multiplexeurs pour chaque état, comme fait en TD. L'autorisation d'écriture d'un registre `A` est notée `wea`, le signal de sélection des multiplexeurs à l'entrée d'un registre `A` est noté `ma`, et le signal de sélection du multiplexeur devant un operateur `#` est noté `s#`.

Ex. 5 : Conception de processeur (5 pts)

L'objectif de cet exercice est d'ajouter deux nouvelles instructions au processeur 2-adresses étudié durant les TD 9, 10 et 11. Pour mémoire, le jeu d'instructions, la partie opérative ainsi que la partie contrôle (sans signaux et conditions) sont rappelés en annexe. La première instruction, `ctz rd`, compte le nombre de zéros entre le bit de poids faible d'un nombre et le premier 1 [4]. Le registre rd est un registre source et destination pour cette instruction. On suppose que l'on dispose d'un opérateur matériel combinatoire qui fait ce calcul, donc capable de donner cette information en un cycle d'horloge.
La deuxième instruction effectue l'échange entre une donnée de 8 bits en mémoire et le contenu d'un registre. L'instruction, notée `swp rd, AD`, effectue $mem[AD] \leftarrow rd$ et, *en même temps*[5] $rd \leftarrow mem[AD]$.

Question 1 Proposez un encodage des instructions `ctz` et `swp`. Précisez pour l'instruction `swp` où se trouve l'adresse.

Question 2 On s'intéresse à l'instruction `ctz rd`. On suppose que l'on dispose d'un opérateur « count trailing zeros » avec une entrée `i` et une sortie `o`. Ajoutez dans la partie opérative (à compléter sur l'annexe à rendre avec votre copie) cet opérateur ainsi que les multiplexeurs et les connexions permettant de réaliser cette instruction.

Question 3 Ajoutez dans la partie contrôle (à compléter sur l'annexe et à rendre avec votre copie) les états nécessaires à l'exécution de l'instruction `ctz rd` en précisant la valeur des différents signaux (valeurs des éventuels nouveaux signaux pour chaque état et valeur de tous les signaux dans les éventuels nouveaux états).

On passe maintenant à l'instruction `swp rd, AD`.

Question 4 Que faut-il ajouter dans la partie opérative pour supporter cette instruction ? Aucun autre registre que `rd` ne doit être modifié par l'exécution de l'instruction. Modifiez l'automate d'états pour prendre en compte cette nouvelle instruction.

Question 5 Ajoutez dans la partie contrôle (à compléter sur l'annexe et à rendre avec votre copie) les états nécessaires à l'exécution de l'instruction `swp rd, AD` en précisant la valeur des différents signaux (valeurs des éventuels nouveaux signaux pour chaque état et valeur de tous les signaux dans les éventuels nouveaux états).

4. `https://en.wikipedia.org/wiki/Trailing_zero`, étonnant, non ?
5. © Palais de l'Elysée.

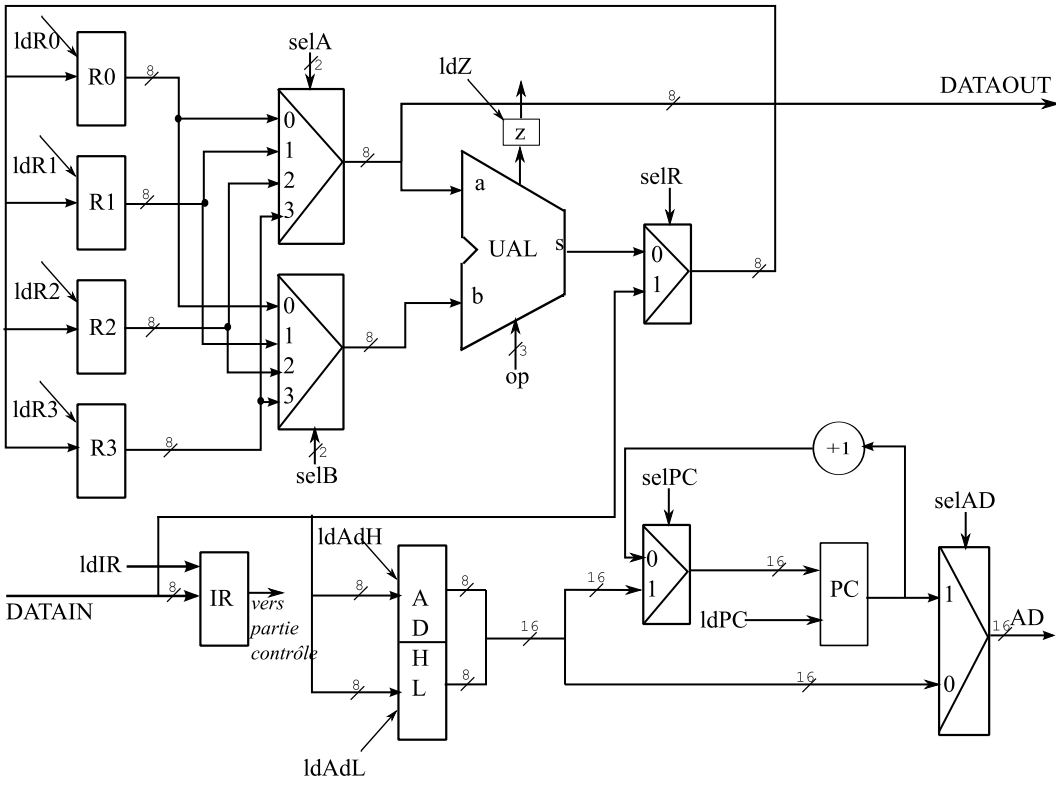
Annexe

NOM : PRENOM : GROUPE :

Jeu d'instructions et encodage

Instruction	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Opérations de la forme : $rd := rd\ op\ rs$								
`or rs, rd`	0	0	0	0	rs_1	rs_0	rd_1	rd_0
`xor rs, rd`	0	0	0	1	rs_1	rs_0	rd_1	rd_0
`and rs, rd`	0	0	1	0	rs_1	rs_0	rd_1	rd_0
`add rs, rd`	0	1	0	0	rs_1	rs_0	rd_1	rd_0
`sub rs, rd`	0	1	0	1	rs_1	rs_0	rd_1	rd_0
Opérations de la forme : $rd := op\ rd$								
`not rd`	0	0	1	1	0	0	rd_1	rd_0
`shl rd`	0	1	1	0	0	0	rd_1	rd_0
`shr rd`	0	1	1	1	0	0	rd_1	rd_0
Chargement : $rd := MEM(AD)$								
`ld AD, rd`	1	0	0	0	0	0	rd_1	rd_0
	ADH							
	ADL							
Stockage : $MEM(AD) := rs$								
`st rs, AD`	1	1	0	0	0	0	rs_1	rs_0
	ADH							
	ADL							
Branchement inconditionnel : $PC := AD$								
`jmp AD`	1	0	0	0	0	1	0	0
	ADH							
	ADL							
Branchements conditionnels : si Z=1 alors $PC = AD$								
`jz AD`	1	0	0	0	0	1	0	1
	ADH							
	ADL							

Partie opérative



Partie Contrôle