Circuits Numériques et Éléments d'Architecture
Examen

ENSIMAG 1A

Année scolaire 2017–2018

- 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 cherche à réaliser un opérateur possédant 8 entrées de 1 bit qui produit un 0 si toutes les entrées sont à 1 et un 1 si au moins une des entrées n'est pas un 1. On fait l'hypothèse que toutes les entrées sont disponibles en même temps, et on veut que la sortie soit disponible le plus tôt possible. Donnez un schéma à base de portes logiques élémentaires à deux entrées qui réalise cette fonction. Donnez en le nombre de portes et le temps de traversée pour 8 entrées. Généralisez pour n entrées.

Question 2 Une bascule flip-flop échantillonne (plusieurs réponses possibles)

(i) sur front montant *ou* sur front descendant ;
(ii) sur front montant *et* sur front descendant ;
(iii) sur état.

Question 3 Soit un automate comprenant 8 états. Indiquer le nombre de bascules D nécessaires à la réalisation de l'automate : (répondre pour les deux cas)

(i) en codage logarithmique ;
(ii) en codage 1 parmi n.

Question 4 Soit un cache de 2k octets possédant des lignes de 4 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 combinatoire (2.5 pts)

On cherche à faire un circuit combinatoire qui indique sur sa sortie s_0 si un nombre compris entre 0 et 13 est un nombre de Lucas [1], et sur sa sortie s_1 si c'est un nombre de Fibonacci [2]. La table de vérité correspondante est donnée ci-dessous.

1. Édouard Lucas, mathématicien mort à 49 ans, en 1891, d'une infection due à une coupure par une assiette brisée lors d'un banquet !
2. Leonardo Fibonacci, mort autour de 1250. Les circonstances de sa mort nous sont inconnues.

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 et en utilisant des tables de Karnaugh, donnez les expressions simplifiées des s_i sous forme canonique (sommes de produits).

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

Ex. 3 : Synthèse d'automate (4 pts)

Pour assurer la sécurité des conducteurs, les voitures sont aujourd'hui équipées d'un système qui déclenche une alarme après quelques secondes (disons 5 secondes) si la voiture avance et que la ceinture n'est pas bouclée. On se propose de réaliser un automate qui implante cette fonctionnalité.
L'automate est doté de 3 entrées booléennes :

- `roule`, qui vaut 1 si la voiture avance, 0 sinon ;
- `ceinture`, qui vaut 1 si la ceinture du conducteur est bouclée, 0 sinon ;
- `expiré`, qui vaut 1 si le délai de 5 seconde est expiré, 0 sinon.

Et d'une sortie, `alarme`, qui vaut 1 pour activer l'alarme, 0 sinon. Cet automate possède 4 états :

- Éteint, qui est l'état initial de l'automate, qui indique que la voiture ne fait rien. Le délai d'attente est constamment réinitialisé à zéro dans cet état ;
- Attente, qui est atteint lorsque la voiture a démarré, et dans lequel on reste tant que (a) le délai n'est pas expiré, (b) la ceinture n'est pas attachée. Notez que même si l'on ne roule plus, le fait que l'on ait roulé implique que le compte à rebours est déclenché ;
- Alarme, représente le fait que plus de 5 secondes se sont écoulées depuis que l'on a démarré (et ce même si l'on s'est arrêté) sans avoir attaché sa ceinture. Le seul moyen de faire taire l'alarme est de boucler sa ceinture ;
- Normal, qui indique que l'on roule avec sa ceinture attachée. Si la ceinture devait être détachée pendant que l'on roule, alors, il faudrait faire tinter l'alarme immédiatement. Si on cesse de rouler, on retourne directement dans l'état Éteint, indépendamment de toute autre considération.

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 : Éteint (00), Attente (01), Alarme (11) et Normal (10). 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 de la sortie `alarme` en fonction des q_i et des entrées booléennes `roule`, `ceinture`, et `expiré`. *On ne demande pas les expressions simplifiées des différents signaux.*

Ex. 4 : Conception PC/PO d'un circuit de calcul de la racine carré entière (7 pts)

L'algorithme écrit en `python` ci-dessous [3] permet de calculer la racine carrée entière d'un nombre et le reste correspondant. Bien qu'en `python` les tailles des variables soient implicites, nous spécifions en commentaire dans l'algorithme le nombre de bits de chacune des variables.

```
1  def isqrt(d):  # d(15:0) Valeur dont on extrait la racine carrée
2     q = 0       # q(8:0)  Résultat courant
3     r = 0       # r(15:0) Reste courant
4     f = 0       # f(8:0)  Variable intermédiaire
5     i = 14      # i(?:?)  Compteur du nombre de tours
6     while i >= 0:
7        r = (r << 2) | ((d >> i) & 3)
8        if r >= ((f << 1) | 1):
9           q = (q << 1) | 1
10          f = ((f + (f & 1)) << 1) | 1
11       else:
12          q = (q << 1) | 0;
13          f = ((f + (f & 1)) << 1) | 0;
14       r = r - (f * (f & 1));
15       i = i - 2
16    return (q, r)
```

- 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'opération `v << p` décale le mot binaire `v` à gauche de `p` positions, les `p` bits de poids fort étant perdus et `p` '0' étant injectés à droite ; il s'agit donc d'un décalage à gauche ;
- l'opération `a | b` effectue le « ou » bit à bit entre les mots binaires `a` et `b` et l'opérateur `a & b` effectue le « et » bit à bit entre les mots binaires `a` et `b` ;
- lors des opérations `|` et `&` avec des constantes, les `0` sur le bon nombre de bits de poids fort sont implicites, mais donc bien présents. Ainsi, par ex. ligne 13, `(f & 1)` s'interprète comme `(f(8:0) & 0_0000_0001)`.

Pour mémoire, des instructions qui peuvent s'effectuer en parallèle peuvent être écrites sous forme d'affectations concurrentes. Par ex. les lignes 2, 3, 4, et 5 de l'algorithme peuvent s'écrire `(q, r, f, i) = (0, 0, 0, 14)`. On peut aussi affecter conditionnellement une variable. Par ex. les lignes 8, 9 et 12 peuvent s'écrire ainsi : `q = (r >= ((f << 1) | 1)) ? ((q << 1) | 1) : ((q << 1) | 0)`.

Question 1

1. sur combien de bits est codée la variable `i` ?
2. comment vous proposez-vous de tester la condition du **`while`** ? Attention, la réponse n'est pas aussi triviale qu'elle semble être, …

Question 2 Plusieurs expressions de l'algorithme contiennent l'opération `(x << 1) | b`, où `x(8:0)` est un vecteur de 9 bits et `b` un unique bit. Proposez une implantation de cet opérateur, qui possède donc deux entrées `x(8:0)` et `b` et une sortie que nous noterons `y(8:0)`.

Question 3 Ligne 14, quelles sont les différentes valeurs que peut prendre l'opération `f * (f & 1)` ? Réécrivez l'expression `r = r - (f * (f & 1))` en utilisant la syntaxe de l'affectation conditionnelle.

On cherche à optimiser l'exécution des lignes 8 à 13.

3. Extrait (avec peine, et sans réussir à éviter des restes négatifs, *no comment*, …) du papier "A Novel Fixed-Point Square Root Algorithm and Its Digital Hardware Design" par Rachmad Vidya Wicaksana Putra.

Question 4 Réécrivez les lignes 9, 10 et 12, 13 et 14 et 15 en utilisant la syntaxe de l'affectation concurrente.

Nous allons maintenant nous intéresser à faire disparaître le **`if`** ligne 8 en exploitant intelligemment le calcul de sa condition.

Question 5 On considère que l'expression `r >= ((f << 1) | 1)` retourne un booléen qui vaut 1 si elle est vraie et 0 sinon. Exploitez cette information pour faire disparaître le **`if`** et écrire ces deux expressions comme une seule.

Question 6 Réécrivez le code de l'algorithme en prenant en compte les optimisations précédentes.

On cherche maintenant à construire la partie opérative du circuit réalisant cet algorithme. Vous trouverez en annexe sur la figure 1 une partie opérative qui contient l'ensemble des opérateurs nécessaires au déroulement de l'algorithme mais aucune des connexions permettant d'effectuer les opérations. Le signal `e` issu du comparateur indique que la condition de la boucle **`while`** n'est plus vraie. Le signal `n` au dessus du soustracteur indique que le résultat du calcul est négatif, *i.e.* il vaut `1` si le résultat du calcul est négatif, et `0` s'il est positif ou nul.

Question 7 Complétez sur l'annexe les connexions entre opérateurs pour réaliser le comportement de l'algorithme tel que trouvé à la question 6, et précisez la largeur des fils associées aux points d'interrogation. Notez que des entrées de certains opérateurs peuvent être des constantes, il faut alors en préciser la valeur sur le schéma. De même, l'entrée de sélection du multiplexeur qui suit le registre `f` sur le schéma est commandée directement par un fil de donnée (*c.f.* question 3). Indiquer lequel.

Question 8 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 dans le tableau fourni en annexe (Table 1), comme fait en TD. L'autorisation d'écriture du registre `x` est notée `wex`, le signal de sélection des multiplexeurs à l'entrée du registre `x` est noté `sx`, et le signal de sélection du multiplexeur devant le soustracteur est noté `ss`.

Ex. 5 : Conception de processeur (4 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 est un calcul de racine carrée entière. On suppose disposer d'un opérateur racine carrée combinatoire, donc capable de donner la racine carrée d'une entrée sur 8 bits dans le cycle de l'horloge. L'instruction se note `sqr rd` et effectue le calcul $rd \leftarrow \sqrt{rd}$.
La seconde instruction permet de faire un saut relatif au pc, en y additionnant une constante sur 8 bits contenue dans l'instruction. L'instruction, notée `jpc cst`, effectue $pc \leftarrow pc + cst$.

Question 1 Proposez un encodage des instructions `sqr` et `jpc`. Précisez pour l'instruction `jpc` où se trouve la constante.

Question 2 On s'intéresse à l'instruction `sqr rd`. On suppose que l'on dispose d'un opérateur « racine carrée » avec une entrée `d` et une sortie `q`. 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 `sqr rd` en précisant la valeur des différents signaux dans un tableau comme en TD.

On passe maintenant à l'instruction `jpc cst`.

Question 4 Que faut-il ajouter dans la partie opérative pour supporter cette instruction ? Attention à bien considérer la taille des opérandes pour cette instruction, ainsi que le fait que l'on puisse sauter aussi bien en avant (jusqu'à $+127$) qu'en arrière (jusqu'à -128) dans le programme. 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 `jpc cst` en précisant la valeur des différents signaux dans un tableau comme en TD.

Question 6 (question bonus) On s'interdit à présent d'enchaîner une lecture mémoire et une addition, afin de limiter le temps de cycle du processeur. Indiquez ce que cela change sur les deux questions précédentes.

Annexe

NOM : PRENOM : GROUPE :

PC/PO

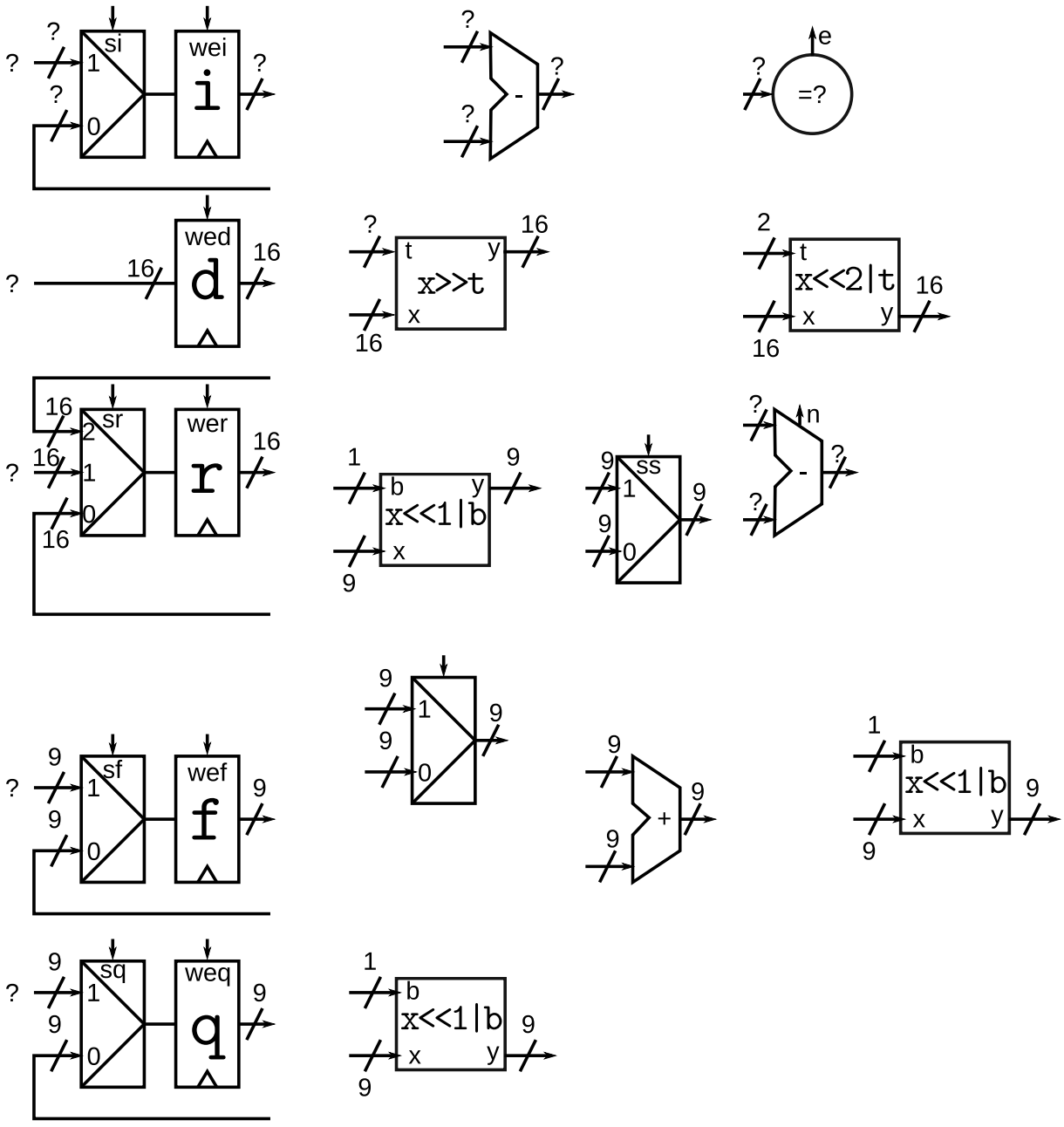


Figure 1 – Partie opérative à compléter pour le calcul de la racine carrée entière

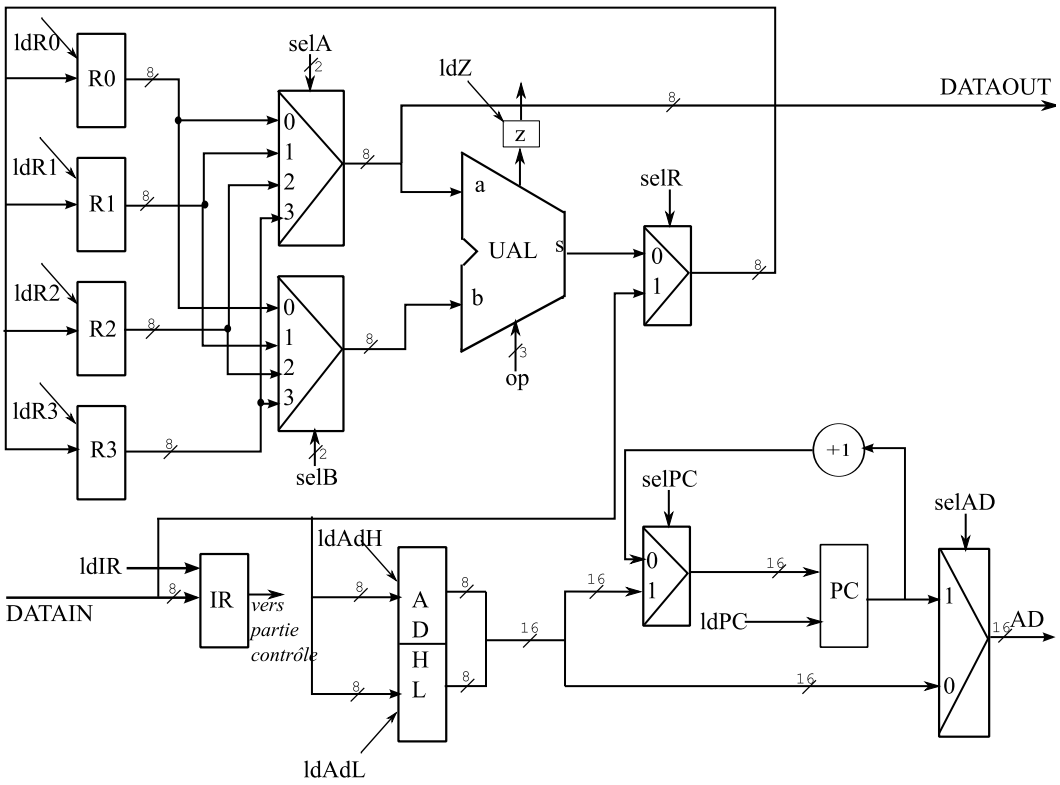
TABLE 1 – Table des signaux à remplir. Le nombre de lignes est arbitraire, vous pouvez avoir moins d'états qu'il y a de lignes dans la table.

	signaux									
état	si	wei	wed	sr	wer	sf	wef	sq	weq	ss
init										

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