Examen final — IFT 1215 — Aut 2015
Introduction aux systèmes informatiques

Professeur : Michel Boyer

Date : Le 18 décembre 2015, 13 : 30 — 16 : 20, local P-310, Pavillon Roger-Gaudry.

Directives :

- *Aucune documentation permise.*
- *Aucun appareil électronique autorisé (calculatrice, téléphone, iPod, etc).*
- *Répondre sur le questionnaire dans l'espace disponible. Utiliser le verso si nécessaire.*
- *Attention, les questions ne sont pas nécessairement en ordre de difficulté.*
- *Total : 100 × 0.4 = 40% de la note finale.*

Nom _____

Prénom _____

Matricule _____

Question 1	/20
Question 2	/25
Question 3	/15
Question 4	/10
Question 5	/20
Question 6	/10
Total	/100

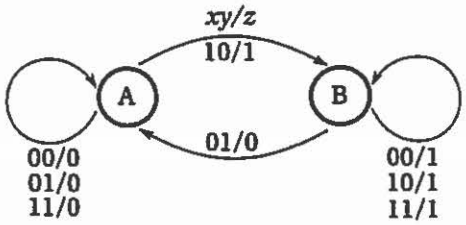
Signature : _____

1. Circuits séquentiels (20 points)

a) (3 pts) Quel est le nombre minimum de bascules D nécessaire pour implanter un compteur dont les sorties (hexadécimales) sont (cycliquement) 130, 012, 108, 201, 200, 115 (le suivant de 115 étant 130) ? Justifiez votre réponse.

b) (7 pts) Donnez le diagramme d'états (automate) qui prend en entrée, séquentiellement, des bits et met 1 en sortie lorsque les deux derniers bits lus sont 01 ou 10, et qui met 0 en sortie sinon. Il faut bien sûr mettre 0 en sortie quand moins de deux bits ont été lus.

c) (10 pts) Le diagramme suivant décrit un automate avec les deux états A et B qui prend en entrée deux bits xy et retourne en sortie le bit z.



(i) Donnez la table d'états (*table de transition*) de cet automate avec les états en lignes et les entrées en colonnes.

(ii) Après avoir encodé les états en binaire, donnez la *table de vérité* pour les transitions d'états d'une éventuelle implantation avec bascules D (uniquement la table est demandée, pas le circuit).

2. La machine LMC (25 points)

La machine LMC a ses instructions et valeurs numériques (en complément à 10) codées avec trois décimales, ses adresses sont de deux décimales, de 00 à 99. Si on désigne par M[xy] le contenu de la cellule mémoire d'adresse xy (xy de 00 à 99), le mnémonique de l'assembleur, le code opération et l'action effectuée par chaque instruction est alors comme suit :

Mnémonique	Code	Action
ADD	$1xy$	M[xy] est additionné au contenu de A
SUB	$2xy$	M[xy] est soustrait du contenu de A
STO	$3xy$	le contenu de A est mis dans M[xy]
LDA	$5xy$	M[xy] est mis dans A
BR	$6xy$	Brancher sans condition à l'adresse xy
BRP	$8xy$	Brancher à l'adresse xy si A ≥ 0
BRZ	$7xy$	Brancher à l'adresse xy si A est égal à 0
IN	901	L'entrée est mise dans A
OUT	902	Le contenu de A est copié en sortie.
HLT	000	L'exécution s'arrête
DAT		Met une valeur numérique à l'adresse courante.

L'exécution utilise les registres A (accumulateur), MAR (memory address register), MDR (memory data register), PC (compteur ordinal) et IR (registre instruction) ; on note IR[adr] la partie adresse (contenant la valeur xy) du registre instruction. Attention, tous les programmes débutent à l'adresse 00.

a) (5 points) Ecrire un programme en assembleur LMC qui lit en entrée un nombre positif de trois chiffres en complément à 10, le mémorise, exécute l'instruction encodée par ses trois chiffres, puis s'arrête.

b) (3 pts) Si on enlève la restriction que l'entrée est positive (en complément à 10 sur 3 chiffres), que se passe-t-il avec votre programme si l'entrée est −399 (entrée qui est permise avec l'émulateur utilisé en classe) ?

c) (7 pts) Écrire en assembleur LMC un programme qui lit en entrée des entiers tant qu'ils sont positifs ou nuls et qui retourne en sortie leur somme dès qu'une entrée négative est tapée.

d) (5 points) Traduisez dans le langage machine de la LMC le code assembleur suivant en indiquant bien en première colonne les adresses où sont mises les instructions (les programmes débutent toujours à l'adresse 00) ; les entrées de la colonne "Code LMC" sont chacune trois chiffres décimaux.

```
A     LDA C
      BRZ B
      OUT
      SUB D
      STO C
      BR  A
B     HLT
C     DAT 010
D     DAT 001
```

Adresse	Code LMC
00	

e) (5 points) Donnez les transferts de registres pour le fetch et l'exécution de l'instruction BR de la LMC.

3. Mémoire cache (15 points).

Dans cette question $K = 2^{10}$ et $M = 2^{20}$. Supposons une mémoire adressable de 256M octets (i.e. les adresses ont 28 bits) et une mémoire cache (cache direct) de 4K octets au total, et dont les lignes sont de 16 octets.

a) Combien la mémoire cache contient-elle de lignes et combien de bits ont respectivement l'index et l'étiquette (tag) ?

b) Soit l'adresse mémoire hexadécimale, 3CE4FE2, quels sont (en notation hexadécimale) le déplacement (offset), l'index et l'étiquette de cette adresse ?

c) Si l'accès à l'adresse 3CE4FE2 ne cause pas de défaut de cache, quelles sont les autres adresses mémoire dont on peut être certain que leur accès ne causera pas pas de défaut de cache.

d) Si l'adresse 3CE4FE2 est dans le cache et si le programme fait un accès mémoire à l'adresse 3C14FE3, dire comment le cache sera modifié : si une étiquette change, dire laquelle, pour quel index, et sa nouvelle valeur, si une ligne change dire quel est son index, et listez les adresses mémoire qu'on y copie.

e) Si en d) on suppose qu'il y avait eu une écriture à l'adresse 3CE4FE2 juste avant l'accès à l'adresse 3C14FE3 et si le cache ne fonctionne pas en more d'écriture au travers (write through) mais utilise un "dirty bit", dire précisément quelles adresses mémoire doivent être mises à jour, et de quelle façon, avant la mise à jour du cache.

4. Modes d'adressage (10 points)

Dans l'assembleur standard d'une machine à trois adresses, le premier argument est ici la destination. Les adresses immédiates sont précédées d'un # ; un nombre non précédé d'un # donne lieu à un adressage direct, sauf s'il sert de déplacement pour un registre. Suivant ces conventions, complétez le tableau suivant indiquant l'effet de chaque instruction en notant M[x] le contenu de la mémoire à l'adresse x et R[a] le contenu du registre de numéro a.

`add R3,R1,R2`	R[3] ← R[1] + R[2]
`add R3,R5,20`	R[3] ←
`add R3,R5,#20`	R[3] ←
`add R3,R5,20(R4)`	R[3] ←
`add R3,#32,20(R4)`	R[3] ←

5. Infixe et postfixe (20 points).

a) Soit l'expression $(x-y)*(x+z)-(x+y)*(z-y)-(x+y)*(x-z)$.

(i) (5 pts) Donnez-en la représentation sous forme arborescente.

(ii) (5 pts) En déduire sa représentation en notation postfixe.

b) Soit la fonction K(x, y, z) qui retourne la valeur de F(y, x)+G(y, z, x) (les entrées et sorties ayant toutes une valeur entière).

(i) (3 pts) Traduisez en postfixe l'expression F(y, x) + G(y, z, x)

(ii) (7 pts) Écrivez de code de K pour la machine WKC à partir de l'adresse 400 en supposant que le code de F est à l'adresse 500 et celui de G à l'adresse 550 (les instructions de la WKC sont en page 12).

6. La machine WKC, implantation et exécution (10 points).

a) Un programme pour la WKC contient les instructions suivantes aux adresses 100, 101 et 102.

```
100: 'PUSH(500)',
101: 'CALL(3)',
102: 'PROUT()',
```

Si juste avant l'exécution de l'instruction à l'adresse 100, FP contient 1020, SP contient 1035, les adresses de 1021 à 1025 contiennent respectivement les valeurs 1, 2, 3, 4, 5 et les adresses de 1031 à 1035 contiennent respectivement les valeurs 11, 12, 13, 14, 15, quelles seront ***les valeurs contenues*** dans les registres PC et FP et ***les valeurs contenues*** dans les adresses 4(FP) et 5(FP) (i.e. aux adresses FP+4 et FP+5) immédiatement après exécution de l'instruction à l'adresse 101 ?

PC :

FP :

4(FP) :

5(FP) :

b) Sur une WKC 16 bits proche de celle dont vous avez utilisé l'émulateur, l'instruction PUSH a un bit de code opération, le 1, et 15 bits pour coder un nombre signé en complément à 2, permettant de faire un PUSH(-30) par exemple. Les arguments immédiats du PUSH vont donc de $-2^{14} = -16384$ à $2^{14} - 1 = 16383$. Écrivez pour cette WKC 16 bits deux instructions aux adresses 15 et 16 pour que le branchement en 17 se fasse à l'adresse 16388. Vous pouvez utiliser l'instruction CHBIT15() qui change le bit le plus significatif (le bit de signe). Si vous n'y arrivez pas en deux instructions, faites le en 3 instructions (à partir de 14, avec pénalité de 2 points).

15:

16:

17: 'BR()',

Instructions de la WKC

ADD()	dépile deux données et empile leur somme
MUL()	dépile deux données et empile leur produit
DIV()	depile y, depile x, puis empile x/y
SUB()	depile y, depile x, puis empile x-y
AND()	depile deux valeurs (0 ou 1) et empile leur AND
OR()	depile deux valeurs (0 ou 1) et empile leur OR
NOT()	remplace le dessus de pile (0 ou 1) par sa négation
NEQ()	dépile deux valeurs, empile 1 si elles sont inégales, sinon empile 0
EQ()	dépile deux valeurs, empile 1 si elles sont égales, sinon empile 0
LT()	dépile y, dépile x, empile 1 si x < y, sinon empile 0
GT()	dépile y, dépile x, empile 1 si x > y, empile 0 sinon
BRIF()	dépile une adresse, dépile un "booléen" (idéalement 0 ou 1), branche à l'adresse si le "booléen" n'est pas 0.
BR()	dépile une adresse et y branche, i.e. met dans le registre PC l'adresse dépilée
READIN()	lit en entrée et empile la valeur lue
PROUT()	dépile une valeur et l'imprime en sortie
NOOP()	ne fait rien
PUSH(n)	empile l'entier n
PUSHFP(n)	empile la valeur qui est contenue à l'adresse n(FP)
POPFP(n)	dépile et met la valeur dépilée dans l'adresse n(FP)
CALL(n)	dépile l'adresse de la fonction où l'on va brancher empile la valeur contenue dans le registre PC (déjà incrémenté) empile la valeur contenue dans le registre FP met dans le registre FP la valeur SP - n - 2 met dans le registre PC la valeur qui a été dépilée.
RETFROM(n)	met SP égal à FP+1 met dans PC et FP les valeurs conservées dans le bloc d'activation lors du CALL
CHBIT15()	inverse le bit le plus significatif (le bit de signe) de la valeur en dessus de pile.