IFT 1215 — Examen final — Hiver 2016
Introduction aux systèmes informatiques

Professeur : Michel Boyer

Date : Le 28 avril 2016, 15 : 30 — 16 : 25, 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 : _____

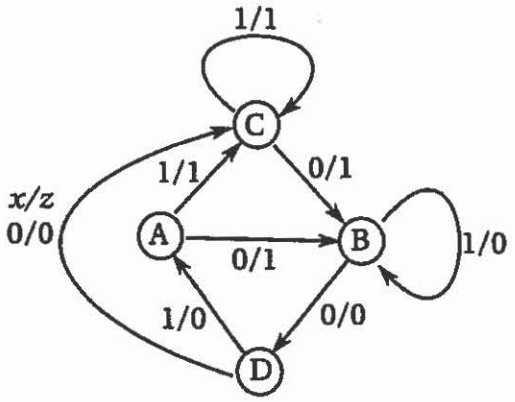
1. Circuits séquentiels (20 points)

a) (5 points) Codez sur le plus petit nombre de bits possible (sur un nombre fixe de bits) les états d'un compteur dont les sorties associées aux états sont 45, 32, 77, 82, 50.

Sortie	45	32	77	82	50
État codé					

b) (5 points) Donnez le ***diagramme d'états (automate)*** qui prend en entrée, séquentiellement, un à un, 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) (5 points) Le diagramme suivant décrit un automate avec les quatre états A, B, C et D, qui prend en entrée un bit x et retourne en sortie le bit z.



Complétez la table d'états (***table de transition***) de cet automate en spécifiant dans chaque case les valeurs s'/z où s' est le nouvel état et z la sortie :

$s \backslash x$	$x=0$	$x=1$
A		
B		
C		
D		

d) (5 points) La table de transition d'un automate avec états A et B qui prend une entrée de deux bits xy et qui retourne en sortie un bit z est donnée par

	$xy = 00$	$xy = 01$	$xy = 10$	$xy = 11$
A	A/0	B/1	B/0	A/1
B	B/0	A/1	B/1	A/0

Donnez d'abord un ***codage en binaire*** des états qui soit avec le nombre minimal de bits possible puis, en utilisant ce codage, donnez la ***table de vérité*** pour les transitions d'états d'une éventuelle implantation avec bascule(s) D ainsi que la table de vérité pour la sortie z ; mettez en partie gauche les entrées du circuit combinatoire et en partie droite une colonne pour D et une colonne pour z.

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) Supposons un programme automodifiant contenant à l'adresse 07 la ligne

```
INS  STO adr
```

où INS est l'étiquette. C'est un programme qui boucle et additionne régulièrement 1 au contenu de l'adresse 07, ce qui change l'adresse de stockage du STO. Ajoutez deux instructions aux adresses 08 et 09 qui auront pour effet de mettre en sortie l'adresse où a été stockée la valeur à l'instruction 07. Vous devrez utiliser une pseudo-instruction additionnelle (DAT) pour stocker une valeur utile et vous pouvez supposer que l'adresse 30 est libre (après insertion de vos deux instructions).

```
...

07      INS STA adr

08      ____________________________________

09      ____________________________________

...

..      HLT

30      ____________________________________
```

b) (3 points) Quelle est la sortie du programme suivant sur l'entree −99. Justifiez votre réponse

```
      IN
      STO next
next  DAT 000
      HLT
```

c) (7 points) Écrire en assembleur LMC un programme qui lit en entrée des entiers et qui retourne en sortie leur maximum et s'arrête dès qu'une entrée négative est tapée. Si la première entrée lue est négative, le programme doit s'arrêter et retournee -1 (cas spécial). Si les valeurs lues sont 5, 0, 7, 2, 3, -1, le programme retourne 7 et s'arrête.

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. Tous les codes sont en page 5 de ce questionnaire.

```
A     IN
      BRP B
      BR  E
B     STO T
      LDA B
      ADD C
      STO B
      BR A
E     HLT
C     DAT 001
T     DAT 000
```

Adresse	Code LMC
00	901

e) (5 points) Donnez les transferts de registres pour le fetch et l'exécution de l'instruction ADD de la LMC (telle que décrite dans le tableau en page 5 de cet examen).

3. Mémoire cache (15 points).

Dans cette question $K = 2^{10}$, $M = 2^{20}$, $G = 2^{30}$. Supposons une mémoire adressable de 4G octets (i.e. les adresses ont 32 bits) et une mémoire cache (cache direct) de 64K 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, 3E14532A, quels sont (en notation hexadécimale) le déplacement (offset), l'index et l'étiquette de cette adresse ?

c) Si l'accès à l'adresse 3E14532A ne cause pas de défaut de cache, donnez la liste explicite de toutes les adresses mémoire dont on peut être certain que leur accès ne causera pas de défaut de cache.

d) Si l'adresse 3E14532A est dans le cache et si le programme fait un accès mémoire à l'adresse 3F14532B, dire comment le cache sera modifié : si une étiquette change, dire laquelle, pour quel index, et donnez sa nouvelle valeur ; si une ligne change dire quel est son index, et donnez la liste de toutes les adresses mémoire qu'on y copie.

e) Si en d) on suppose qu'il y avait eu une écriture à l'adresse 3E14532A juste avant l'accès à l'adresse 3F14532B et si le cache ne fonctionne pas en mode d'écriture au travers (write through) mais utilise un "dirty bit", dire précisément quelles adresses mémoire doivent être mises à jour, et à partir de quelles données, avant que le contenu du cache ne soit modifié.

4. Adressage et format des instructions (10 points)

a) Soit une machine à deux adresses dont les opérations arithmétiques se font sur des registres généraux ; il y a 32 registres généraux. L'instruction `ADD R12, R25` a pour effet d'additionner les valeurs entières contenues dans les registres `R12` (Rdest) et `R25` (Rsrc) et de mettre le résultat dans le registre `R12` (Rdest). Les valeurs entières sont de 32 bits. Si on code l'instruction `ADD` dans le format | Opcod | ??? | Rdest | Rsrc | sur 32 bits, combien de bits dans l'instruction restent inutilisés (ou utilisables pour un déplacement) si l'opcod est de 5 bits ? (justifiez votre réponse).

b) Dans l'assembleur standard d'une machine à deux adresses, le premier argument est ici la destination. Suivant les conventions standard pour les modes d'adressage telles que précisées en classe, complétez le tableau suivant indiquant l'effet de chaque instruction ; notez M[x] le contenu de la mémoire à l'adresse x et R[a] le contenu du registre de numéro a ; le # représente un adressage immédiat.

`add R20,R12`	R[20] ← R[20] + R[12]
`add R20,#1143`	R[20] ←
`add R20,1143`	R[20] ←
`add R20,20(R4)`	R[20] ←

5. Infixe et postfixe (20 points).

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

(i) (3 points) Réécrire cette expression sous forme complètement parenthésée (par exemple $x+y+z$ donne $((x+y)+z)$ quand on rajoute toutes les parenthèses).

(ii) (3 points) Donnez sous forme arborescente la représentation de l'expression complètement parenthésée obtenue

(iii) (4 points) En déduire sa représentation en notation postfixe.

b) Soit la fonction K(x, y, z) telle que K(x, y, z) = F(y) + G$(z, x) - z$ (les entrées x, y et z ainsi que les valeurs retournées par F et G sont entières).

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

(ii) (7 points) 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 14).

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

a) Un programme pour la WKC contient les instructions suivantes aux adresses 97 à 102.

```
 97: 'PUSH(10)',
 98: 'PUSH(20)',
 99: 'PUSH(30)',
100: 'PUSH(500)',
101: 'CALL(3)',
102: 'PROUT()',
```

Si juste avant l'exécution de l'instruction à l'adresse 100, FP contient 1032, SP contient 1040, les adresses de 1032 à 1036 contiennent respectivement les valeurs 1, 2, 3, 4, 5 et les adresses de 1040 à 1044 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 3(FP) et 5(FP) (i.e. aux adresses FP+3 et FP+5) immédiatement après exécution de l'instruction à l'adresse 101 ?

PC:_____ FP:_____ 3(FP):_____ 5(FP):_____

b) Sur la WKC 16 bits vue en classe, 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 17658. Vous pouvez utiliser l'instruction CHBIT15() qui change le bit le plus significatif (le bit de signe) ainsi que PUSH avec un argument hexadécimal (par example PUSH(0x3456)). 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 (i.e. incrémente SP et met n à l'adresse dans SP)
PUSH_FP(n)	empile la valeur qui est contenue à l'adresse FP+n
POP_FP(n)	dépile et met la valeur dépilée dans l'adresse FP+n
CALL(n)	dépile l'adresse du code 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 l'adresse qui avait été dépilée.
RETFROM(n)	met dans PC le contenu de l'adresse FP+n+1 met dans SP un plus la valeur contenue dans FP met dans FP la valeur à l'adresse FP+n+2
CHBIT15()	inverse le bit le plus significatif (le bit de signe) de la valeur en dessus de pile.