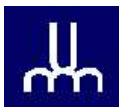


DIRO
IFT 1215

EXAMEN FINAL

Max Mignotte

DIRO, Département d'Informatique et de Recherche Opérationnelle, local 2377
Http : //www.iro.umontreal.ca/∼mignotte/ift1215/
E-mail : mignotte@iro.umontreal.ca

Date : *25/04/2008*

I	.....	Assembleur/LMC/Registres (57 pts)
II	.....	Mémoire micro-programmée (14 pts)
III	.....	Programmation Shell Script (23 pts)
IV	.....	Misc. (12 pts)
V	.....	Codes correcteurs (8 pts)
Total	.....	114 pts

Directives

- TOUTE DOCUMENTATION ET CALCULATRICE PERSONNELLE PERMISES

- Les réponses devront être clairement présentées et justifiées (elles peuvent être concises mais devront néanmoins contenir les résultats intermédiaires nécessaires permettant de montrer sans ambiguïté que vous êtes arrivés au résultat demandé).

- Si vous ne comprenez pas une question, faites en une interprétation, et proposez une réponse.

I. Assembleur/LMC/Registres (57 pts)

1. Programmation Machine

Implémenter en code d'instructions mnémoniques, avec le jeux d'instructions donné plus bas (cf. Fig. (1)) correspondant au modèle LMC vu en cours, et avec des adresses symboliques (i.e., étiquettes), un programme ou deux entiers positifs (ou nuls) a et b sont tout d'abord entrés en mémoire au début du programme (par l'instruction IN) et qui affiche ensuite en sortie le résultat de la multiplication entre a et b. Le résultat de cette multiplication ne devra se faire qu'en utilisant l'instruction machine ADD (i.e., en utilisant seulement l'addition).
Par exemple, si on entre 3 et 4, le programme devra calculer cette multiplication en faisant $3+3+3+3$ (i.e, en ajoutant 4 fois 3). Si on entre 2 et 5, le programme devra calculer cette multiplication en faisant $2+2+2+2+2$ (i.e, en ajoutant 5 fois 2), etc.
On supposera que le résultat est communiqué en sortie par l'instruction OUT.
<15 pts>

LDA	5xx	Load
STO	3xx	Store
ADD	1xx	Add
SUB	2xx	Subtract
IN	901	Input
OUT	902	Output
COB or HLT	000	Coffee break (or Halt)
BRZ	7xx	Branch if zero
BRP	8xx	Branch if positive or zero
BR	6xx	Branch unconditional
DAT		Data storage location

FIG. 1 – Jeu d'instructions du LMC

2. Langage LMC et Registres

Voici un programme en code d'instructions mnémoniques

```
        IN                  ...1
        STO     MN
        STO     MX
LOOP    LDA     CPT
        SUB     ONE
        STO     CPT
        BRZ     END
        IN
        STO     TMP
        LDA     MX          ...10
        SUB     TMP
        BRP     L1
        LDA     TMP
        STO     MX
L1      LDA     TMP
        SUB     MN
        BRP     L2
        LDA     TMP
        STO     MN
L2      BR      LOOP        ...20
END     LDA     MX
        SUB     MN
        OUT
        HLT
TMP     DAT     0
ONE     DAT     1
CPT     DAT     4
RES     DAT     0
MX      DAT     0
MN      DAT     0           ...30
```

(a) Indiquer ce que ce programme va faire (i.e., afficher en sortie) si on entre les séries de nombres suivant comme inputs à ce programme :

i. $(4, 7, 2, 10)$ (i.e., 4 pour le premier IN, 7 pour le second IN, etc.)

ii. $(8, 2, 1, 1)$

iii. Dans le cas général

<8 pts>

(b) Indiquer ce que ce programme va faire (i.e., afficher en sortie) si on remplace l'instruction à l'adresse symbolique L2 par **BRP** (au lieu de BR) et si on entre les séries de nombres suivant comme inputs à ce programme :

i. $(4, 7, 2, 10)$

ii. $(8, 2, 1, 1)$

<5 pts>

(c) On pourrait optimiser ce programme (dans le but qu'il aille un peu plus vite) en modifiant une des instructions machines. Laquelle ?

<3 pts>

(d) On pourrait optimiser ce programme (dans le but qu'il aille un peu plus vite) en ajoutant une instruction machine. Laquelle et à quelle endroit ?

<3 pts>

(e) Convertir les 12 premières instructions de ce programme en codes machines (codes d'opérations) en logeant ce programme à partir de l'adresse mémoire 00 et en précisant pour chaque instruction l'état du bit de relocation (i.e., l'indicateur de relocation) et en supposant que toutes les instructions et les données du programme sont le moins dispersé possible.

<7 pts>

(f) Donner la valeur des différents registres ACCU, IR, PC, MAR, MDR à la fin (de la phase exécution) de l'instruction **LDA CPT** (4-ième instruction) pour la première entrée dans la boucle pour l'input suivant $(4, 7, 2, 10)$.

<5 pts>

3. **Langage LMC et Tableaux**

Écrire en code d'instructions mnémoniques correspondant au modèle LMC vu en cours, et avec des adresses symboliques (vous pouvez utiliser aussi l'instruction STA), un programme permettant de trouver et d'afficher en sortie la plus grande donnée située entre l'adresse $adr1$ et l'adresse $adr2$ ($adr1$ et $adr2$ étant deux valeurs entrées en mémoire au début du programme par l'instruction IN).
Exemple : si on entre 50 et 54 en paramètre et que la mémoire stocke les 5 données ou instructions machines suivantes $625, 125, 765, 354, 154$ aux adresses respectifs 50, 51, 52, 53 et 54, le programme devra afficher en sortie 765.
On commentera intelligemment le programme de façon à ce que celui ci soit lisible et facilement compréhensible.

<12 pts>

Réponse

1.

```
        IN
        STO   INTA     | Enregistrement Entier A.
        IN
        STO   INTB     | Enregistrement Entier B.
LOOP    BRZ   END      | Fin Prog. après B addition de A
        LDA   RES
        ADD   INTA     | Addition de A ...
        STO   RES
        LDA   INTB
        SUB   ONE      |          ... B fois
        STO   INTB
        BR    LOOP
END     LDA   RES      | Fin Prog. Affichage Résultat
        OUT
        HLT
INTA    DAT   0        | Stockage Données
INTB    DAT   0
RES     DAT   0
ONE     DAT   1
```

<15 pts>

2a.

- Pour $(4, 7, 2, 10)$, le programme affichera en sortie 8.
  <2.5 pts>
- Pour $(8, 2, 1, 1)$, le programme affichera en sortie 7.
  <2.5 pts>
- Dans le cas général, le programme affichera en sortie $\max\{\text{Série de nombres}\} - \min\{\text{Série de nombres}\}$, c'est à dire l'amplitude max-min [i.e., (MX)-(MN)] de la série des 4 nombres entrées en paramètres.
  <3 pts>

2b.

Si on remplace l'instruction à l'adresse symbolique L2 par **BRP** (au lieu de BR), Cela ne changera rien si tous les nombres entrés sont positif et non-nuls (ce qui est le cas pour nos deux séries précédente de quatre nombres).

Nota : Les deux cas possibles sont soit suite à un BRP (cas : pas de nouveau min : le BR L2 ou BRP L2 ne change donc rien dans ce cas), soit après avoir assigné ÀMN la valeur de TMP qui est une des entrée (positive ou nul), dans ce cas, puisque l'accumulateur est positif ou nul, l'instruction BR L2 ou BRP L2 seront équivalentes.

<5 pts>

2c.

En modifiant la 17 ième instruction (BR L2) et en la remplaçant par BRP LOOP.

<3 pts>

Nota : Cela permettrait de gagner un cycle d'horloge pour chaque nouveau nombre de la série qui ne serait pas un nouveau min. Cela permettrait donc de sauver 2 cycles d'horloge pour le premier exemple et aucun cycle d'horloge pour le deuxième exemple.

2d.

Après avoir stocké le nouveau MAX (dans la case mémoire MX), il n'est pas nécessaire de savoir si ce nombre est un nouveau MIN ! et donc on peut le traduire en ajoutant entre la 14 ième et 15 ième instruction (après STO MX) l'instruction machine BR LOOP. Cela permettrait de gagner plusieurs cycles machines.

<3 pts>

Nota-1- : Plus précisément, cela permettrait de gagner 3×2 cycles d'horloge pour le premiers exemple et aucun pour le deuxième exemple.

Nota-2- : On pourrait gagner un cycle machine en ajoutant l'instruction BRZ END sous l'instruction SUB ONE aussi.

Nota-3- : On pourrait enlever la ligne RES DAT car elle ne sert a rien.

2e.

Les 12 premières conversions en code machine et son indicateur de relocation se trouvent immédiatement

Ad.	Code	Ind. Reloc.
00	901	0
01	329	1
02	328	1
03	526	1
04	225	1
05	326	1
06	720	1
06	901	0
07	324	1
08	528	1
09	224	1
10	814	1
11	324	1
...	...	

<7 pts>

2c.

	ACCU	IR	PC	MAR	MDR
Fin de **LDA CPT**	004	526	004	026	004
	<1 pt>	**<1 pt>**	**<1 pt>**	**<1 pt>**	**<1 pt>**

3a.

Un exemple de programme

```
            IN
            STO     AD1
            IN
            STO     AD2       On stocke les paramètres

            LDA     AD1
LOOP        STA     DTINST    On transforme l'instr. à l'adr. DTINST en LDA AD1+ième itér.
                              de boucle suivant que l'on soit dans la 1ère, 2ième,...,nième itér.
DTINST      LDA     TEM       LDA AD1+ième itér. de boucle : instruction a l'adr. (AD1+ième itér.) lu
            STO     TEM
            SUB     MAX
            BP      L1
            BR      L2

L1          LDA     TEM
            STO     MAX       Stockage nouveau Max

L2          LDA     AD1
            ADD     ONE
            STO     AD1       AD1+1 → AD1

            SUB     AD2
            BZ      END       Si l'adresse considéré > AD2 alors STOP

            LDA     AD1       Accumulateur=(AD+1+ième itération de boucle)
            BR      LOOP

END         LDA     MAX
            OUT
            HLT               Affichage MAX)

AD1         DAT     0
AD2         DAT     0
ONE         DAT     1
ZERO        DAT     0
TEM         DAT     0
MAX         DAT     0
```

<12 pts>

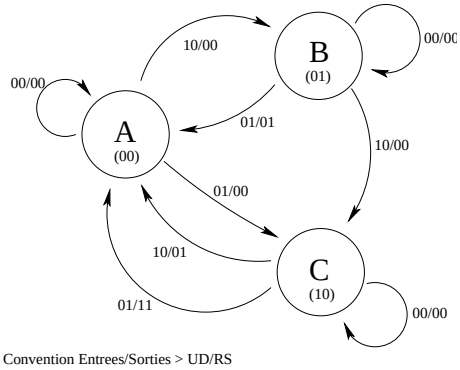
II. Mémoire micro-programmée (14 pts)

Soit le diagramme de transition d'états suivant (qui correspond à un distributeur automatique simplifié) dans lequel il y a deux signaux d'entrée U et D, deux signaux de sortie R et S et comprenant trois états A, B et C codée respectivement en binaire par les codes 00, 01 et 10.

On dispose d'une horloge et d'une petite mémoire permettant de stocker 16 Bytes (octets).

1. Indiquer quelles sont les différents micro-codes que l'on devra stocker dans cette mémoire (ainsi que leur adresse respectif) permettant ainsi de micro-programmer ce circuit séquentiel et de simuler ce diagramme de transition d'états.
   <9 pts>

2. Indiquer le dispositif microprogrammé (i.e., le schéma électrique) permettant de créer (avec cette mémoire et cette horloge) la FSM (machine a états fini) simulant ce diagramme de transition d'états. **<5 pts>**



Réponse

1.

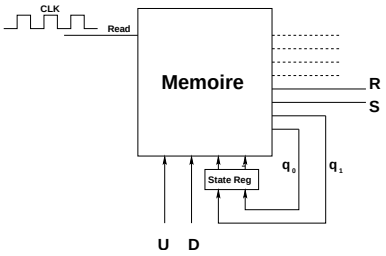
Le diagramme de transition d'états nous permet de trouver immédiatement la table de vérité qui lui est associé

ÉTATS PRÉSENTS	ENTRÉES	ÉTATS FUTURS	SORTIES
$q_1(t)\ q_0(t)$	U D	$q_1(t+1)\ q_0(t+1)$	R S
00	00	00	00
00	10	01	00
00	01	10	00
01	00	01	00
01	10	10	00
01	01	00	01
10	00	10	00
10	10	00	01
10	01	00	11

A chaque [ÉTATS PRÉSENTS : : ENTRÉES] différent (que l'on prendra comme adresses) correspond un [ÉTATS FUTURS : : SORTIES] (i.e., un ensemble de bits) que l'on enregistrera dans la mémoire.
<9 pts>

2.

et on considérera le dispositif suivant



<5 pts>

III. Programmation Shell Script (23 pts)

1. **Gestion de Fichiers**

   On vous demande de créer un script "CATTWOCOLUMNS.SH" passant en paramètre deux fichiers (qui ont tous les deux, une liste de N mots [un mot par ligne]) (pour une exécution du style CATTWOCOLUMNS.SH FILE1.DAT FILE2.DAT) et permettant d'afficher à l'écran sur une même ligne le premier mot du premier fichier suivi (sur la même ligne) du premier mot du deuxième

fichier, puis à la ligne suivante, le deuxième mot du premier fichier suivi du deuxième mot du deuxième fichier et ainsi de suite, etc.
<10 pts>

2. **Manipulation/Modification de Fichiers**

Vous trouverez en Fig. (2), le début du contenu d'un fichier (appelé FILE4.DAT). Ce fichier contient sur chaque ligne un nombre flottant (qui représente un score) et le nom d'une image (après traitement). L'expression "....." en fin de fichier signifie que le fichier continue avec la même syntaxe.

```
0.97718      167062_320.ppm_A22.pgm
0.8489      167083_320.ppm_A22.pgm
0.93004      169012_320.ppm_A22.pgm
0.82587      176035_320.ppm_A22.pgm
0.88758      176039_320.ppm_A22.pgm
0.85614      178054_320.ppm_A22.pgm
0.80595      209070_320.ppm_A22.pgm
0.93225      2092_320.ppm_A22.pgm
0.57884      210088_320.ppm_A22.pgm
0.75288      21077_320.ppm_A22.pgm
0.92189      216041_320.ppm_A22.pgm
0.92929      216053_320.ppm_A22.pgm
0.76427      216066_320.ppm_A22.pgm

.....
```

FIG. 2 – Fichier FILE4.dat

On vous demande de créer un script COMPUTE5MOY.SH permettant d'obtenir la moyenne (en flottants) des cinq scores associés aux images traitées suivantes 167062, 216081, 62096, 42049, 241004. Plus précisément, on aimerait que le stdout (i.e., terminal de l'ordinateur) affiche en sortie (avec le format suivant)

```
0.97718 ::167062 ::1
0.90639 ::216081 ::2
0.94578 ::62096 ::3
0.95831 ::42049 ::4
0.88073 ::241004 ::5

PRI .933678000000000000000
```

Ce script devra passer en paramètre le fichier FILE4.dat.dat, pour une exécution du style

COMPUTE5MOY.SH FILE4.DAT .

<13 pts>

Réponse

1.

Un exemple de script pourrait être

```
#! /bin/sh
# CatTwoColumns.sh script

nb=1
cat ${1} | while read line1
do

line2=`cat ${2} | head -n $nb | tail -n 1`

echo $line1 $line2
let nb=nb+1

done
```

<10 pts>

Il y a aussi la version ultra-short et concaténé pour gens pressés qui fonctionne tout aussi bien, i.e., la commande

```
nb=1 ; cat ${1} | while read line1 ; do echo $line1 `cat ${2} | head -n ${nb} | tail -n 1` ; let nb++ ; done
```

Barème des 10 Pts :
———————————

Recup de la 1ere Ligne : 2 Pts
Recup de la 2eme Ligne : 2 Pts
Affichage : 2 Pts
Reste Algo & Syntaxe : 4 Pts

2.

Un exemple de script pourrait être

```
#! /bin/sh
# Compute5Moy.sh script

#--Parametres
File=${1}
Ext="_320"


sum=0
cpt=1

for val in {167062,216081,62096,42049,241004}
do

pri=`grep " $val${Ext}" ${1} | cut -d " " -f 1`

echo $pri  ::$val ::$cpt

sum=`echo $sum+$pri | bc -l`


let cpt=cpt+1

done

echo ""
moy=`echo $sum/5 | bc -l`
echo PRI $moy
echo ""
```

<13 pts>

Barème des 13 Pts :
———————————

Extraction score : 4 Pts
Calcul Moyenne : 4 Pts
Affichage : 2 Pts
Reste Algo & Syntaxe : 3 Pts

IV. Misc. (12 pts)

Répondre aux questions suivantes

1. Vous trouverez dans la figure ci dessous le pseudo code de deux boucles qui sont censé faire la même chose dans un programme.

```
//version 1 (835 msec)

for(i=0;i<length;i++) for(j=0;j<width;j++)
   tmp+=Tab[i][j];


//version 2 (9368 msec)

for(j=0;j<width;j++) for(i=0;i<length;i++)
 tmp+=Tab[i][j];
```

Cependant, sur mon ordinateur

(a) pour length et width = 10000, la première boucle s'effectue en 835 msec et la seconde en 9368 msec.

(b) pour length et width = 1000, la première boucle s'effectue en 8 msec et la seconde en 18 msec.

(c) pour length et width = 300, la première boucle s'effectue en 1 msec et la seconde en 1 msec.

Expliquez (en trois-dix phrases) pourquoi. justifiez bien votre réponse.
<7 pts>

2. Soit la table des pages suivantes

page 0	-	xx
page 1	-	11
page 2	-	00
page 3	-	xx
page 4	-	xx
page 5	-	01
page 6	-	xx
page 7	-	10

Sachant que les pages physiques et virtuelles font 2 K octets, quelle est l'adresse mémoire (en hexa) correspondant a chacune des adresses virtuelles suivantes : $142A_{16}$ et $0AF1_{16}$?
<5 pts>

Réponse

1.

C'est grâce à la technique dite de *mémoire cache* que la première boucle ira plus vite que la deuxième. La mémoire cache est une mémoire ultra rapide d'accès (comparativement à la mémoire physique sous forme de barrettes) gravée directement dans le CPU qui va, lorsque le processeur demandera la donnée Tab[x][y] à la mémoire centrale, chargera (ou transférera) aussi dans la mémoire cache tout une ligne de données, précisément les données qui ont été stockés contiguement (longitudinalement dans le tableau Tab), i.e., Tab[x][y+1], Tab[x][y+2], Tab[x][y+3], ..., Tab[x][y+16] (i.e., dans le cas d'un bloc de données de 16 octets). De ce fait, la première boucle qui fera appel plus souvent à ces données stockés continûment (préalablement transféré dans la cache) sera plus rapide.

Dans le troisième exemple (tableau de taille 400×400), le tableau de la deuxième boucle est assez petit pour avoir été transféré totalement dans la cache (pas en une seule étape mais ligne après ligne). Cela explique que les deux boucles s'effectuent donc à peu près à la même vitesse.
<7 pts>

2.

Puisque 2 K octets est la taille des page virtuelles et physique, cela veut dire que les 11 (i.e., $2^{11} = 2048 = 2K$) derniers bits d'une adresses virtuelles seront l'adresse dans chacune des pages mémoire et les autres bits désigneront le numéro de page.

$142A_{16} = 0001\ \mathbf{0}\boxed{100\quad 0010\quad 1010}_2$ **<1 pt>**
est donc dans la page virtuelle 2, correspondant à la page mémoire 00_2 (0 ième page mémoire) et correspondant à l'adresse mémoire $\mathbf{0\ 0}100\ 0010\ 1010_2 = 42A_{16}$, (i.e., à l'adresse $42A_{16}$ dans cette première page mémoire).

$0AF1_{16} = 0000\ 1010\ 1111\ 0001_2$ **<1 pt>**
est donc dans la page virtuelle 1, correspondant à la page mémoire 11_2 (3 ième page mémoire) et correspond à l'adresse mémoire $\mathbf{1\ 1}010\ 1111\ 0001_2 = 1AF1_{16}$, (i.e., à l'adresse $2F1_{16}$ dans cette 3 ième page mémoire).
<3 pts>

V. Codes Correcteurs (8 pts)

1. **Codage CRC**
   On veut transmettre le message CB_{16} en utilisant la méthode CRC avec le polynôme générateur $G(x) = x^4 + x^2 + x$. Trouver le message à transmettre et le mettre en code hexadécimal.
   <8 pts>

Réponse

1.

• On convertit d'abord le message en binaire

$$CB_{16} \quad \blacktriangleright \quad 1100\,1011_2 \qquad \textbf{<1 pt>}$$

• Après avoir ajouté 4 zéros au message (4 correspondant au degré du polynôme générateur), on réalise ensuite la division modulo 2 suivante

1	1	0	0	1	0	1	1	0	0	0	0	|	1	0	1	1	0
													–	–	–	–	–
1	0	1	1	0								|					
–	–	–	–	–								|					
0	1	1	1	1	0							|					
	1	0	1	1	0							|					
	–	–	–	–	–							|					
	0	1	0	0	0	1						|					
		1	0	1	1	0						|					
		–	–	–	–	–						|					
		0	0	1	1	1	1	0				|					
				1	0	1	1	0				|					
				–	–	–	–	–				|					
				0	1	0	0	0	0			|					
					1	0	1	1	0			|					
					–	–	–	–	–	–		|					
					0	0	1	1	0	0	0	|					
							1	0	1	1	0	|					
							–	–	–	–	–	–					
							0	1	1	1	0	|					
												|					

<5 pts> **RJCF :2/5**

• On transmettra donc le message $\underbrace{11001011}_{M(x)}\underbrace{1110}_{R(x)} = CBE_{16}$. **<2 pts>**