DIRO
IFT 1215

EXAMEN FINAL

Max Mignotte

DIRO, Département d'Informatique et de Recherche Opérationnelle, local 2377
Http : //www.iro.umontreal.ca/∼mignotte/ift1214/
E-mail : mignotte@iro.umontreal.ca

Date : *28/04/2006*

I	Assembleur/LMC/Registres (50 pts).
II	Programmation Shell Script (26 pts).
III	Misc. (12 pts).
IV	Codes correcteurs (22 pts).
Total	110 pts

Directives

- Toute documentation et calculatrice personnelle est permise

◦ 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 (50 pts)

1. **Programme -1-**

   Implémenter en code d'instructions mnémoniques, avec le jeux d'instruction donné plus bas (cf. Fig.) correspondant au modèle LMC vu en cours, et avec des adresses symboliques (i.e., étiquettes), un programme permettant d'afficher en sortie la somme d'une suite arithmétique dont la plus grande valeur n a été entré en mémoire au début du programme par l'instruction IN (i.e., affichage du résultat de $n + (n-1) + (n-2) + ... + 1, \quad \forall n > 0$). On supposera que le résultat est communiqué en sortie par l'instruction OUT.[1]
   <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

2. **Langage LMC et registres**

   Voici un programme en code d'instructions mnémoniques,

```
        IN
        STO   SMALL
        IN
        STO   BIG
BEGIN   LDA   SMALL
        ADD   ONE
        OUT
        STO   SMALL
        LDA   BIG
        SUB   ONE
        OUT
        BZ    END
        STO   BIG
        SUB   SMALL
        BZ    END
        BR    BEGIN
END     HLT
ONE     DAT   1
SMALL   DAT   0
BIG     DAT   0
```

   (a) Indiquer ce que ce programme va respectivement faire (i.e., afficher en sortie ; préciser par exemple les 20 premières sorties ou jusqu'à l'arrêt du programme) si on entre les séries de nombres suivant comme inputs à ce programme :

      i. $(10, 20)$ (i.e., 10 pour le premier IN et 20 pour le second IN)

      ii. $(11, 20)$

      iii. $(20, 10)$

      <6 pts>

1. Exemple ; si on entre, avec l'instruction IN, la valeur 10, le programme devra afficher en sortie le résultat de la somme $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$, soit 55 en sortie.

(b) Convertir les dix 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).
<6 pts>

(c) Donner la valeur des différents registres ACCU, IR, PC, MAR, MDR à la fin (de la phase exécution) des instructions **STO SMALL** (2 ième instruction) et **BR BEGIN** (16 ième instruction pour la première entrée dans la boucle BEGIN) pour les deux inputs suivant $(10, 20)$.
<5 pts>

(d) Indiquer quel devrait être la suite d'instructions (instructions mnémoniques avec adresses symboliques), que l'on devrait ajouter au programme, qui lui permettrait d'exécuter (i.e., afficher la même chose en sortie) si on lui donne comme inputs (IN= 20 , IN= 10) comparativement à (IN= 10 , IN= 20).
<5 pts>

3. **Langage LMC et tableaux**

Voici un programme en code d'instructions mnémoniques et adresses symboliques,

```
        IN
        STO   AD1
        IN
        STO   AD2
        LDA   AD1
LOOP    STA   DTINST

DTINST  LDA   TEM
        OUT
        LDA   AD1
        ADD   ONE
        STO   AD1
        SUB   AD2
        BZ    STOP
        LDA   AD1
        BR    LOOP

STOP    HLT

ADC     DAT   0
AD1     DAT   0
AD2     DAT   0
ONE     DAT   1
TEM     DAT   0
```

NOTA : Attention, la 5ième instruction de ce programme est bien l'instruction **STA** (et non pas STO !).

(a) Indiquer ce que ce programme va faire lorsqu'on lui présentera comme inputs la série de nombres $(40, 80)$, c'est à dire 40 pour le premier IN et 80 pour le second IN. Ajoutez quatre ou cinq lignes de commentaires à ce programme pour qu'il devienne facilement compréhensible (aux endroits importants).
<6 pts>

(b) Quel devrait être la valeur du bit de relocation pour la 7ième instruction ? (0, 1, ou ... les deux possibilités ne changeraient rien au fonctionnement du programme). Justifier votre réponse. Préciser quel est la nature de ce qui ce trouve à l'adresse symbolique DTINST (donnée, instruction ou les deux en même temps ?). Justifier votre réponse.
<3.5 pts>

(c) Indiquer ce que ce programme va faire si on change l'instruction à l'adresse symbolique DTINST par STO TEM lorsqu'on présentera à ce programme la série de nombres $(40, 80)$ comme inputs.
<3.5 pts>

Réponse

$\boxed{\textbf{1.}}$

Un exemple possible, valable $\forall\, n > 0$, dans lequel le comptage est fait à rebours,

```
        IN
        BZ    END
        STO   SUM
        STO   NB
L1      SUB   ONE
        BRZ   END
        ADD   SUM
        STO   SUM
        LDA   NB
        SUB   ONE
        STO   NB
        JMP   L1
END     LDA   SUM
        OUT
        HLT
ONE     DAT   1
SUM     DAT   0
NB      DAT   0
```

<**15 pts**>

$\boxed{\textbf{2a.}}$

- Pour $(10, 20)$, le programme affichera en sortie les nombres suivants :
  11 19 12 18 13 17 14 16 15 15,
  puis s'arrêtera.
  <**2 pts**>

- Pour $(11, 20)$, le programme affichera en sortie les nombres suivants :
  12 19 13 18 14 17 15 16 16 15 17 14 18 13 19 12 20 11.... ....30 1 31 0,
  puis s'arrêtera.
  <**2 pts**>

- Pour $(20, 10)$, le programme affichera en sortie les nombres suivants :
  21 9 22 8 23 7 24 6 25 5 26 4 27 3 28 2 29 1 30 0,
  puis s'arrêtera.
  <**2 pts**>

$\boxed{\textbf{2b.}}$

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

Ad.	Code	Ind. Relocation
00	901	0
01	318	1
02	901	0
03	319	1
04	518	1
05	117	1
06	902	0
07	318	1
08	519	1
09	118	1
...	...	

<**6 pts**>

2c.

	ACCU	IR	PC	MAR	MDR
Fin de **STO SMALL**	010	318	02	18	10
Fin de **BR BEGIN**	008	604	04	15	604
	<1 pt>	**<1 pt>**	**<1 pt>**	**<1 pt>**	**<1 pt>**

2d.

Dans le cas ou on présente l'inputs $(20, 10)$, il suffirait d'inverser au tout début du programme ces inputs pour que l'adresse de valeur minimale soit bien égale à 10 et que l'adresse de valeur maximale soit bien égale à 20. Cela pourrait se faire facilement avec la suite d'instruction mnémonique suivante ;

Ad.	Code	Mnémonique
...	...	
	IN	
	STO	*BIG*
	LDA	BIG
	SUB	SMALL
	BRP	BEGIN
	LDA	SMALL
	STO	TEMP
	LDA	BIG
	STO	SMALL
	LDA	TEMP
	STO	BIG
BEGIN	*LDA*	*SMALL*
	ADD	*ONE*
...	...	

<5 pts>

3a.

Le programme va afficher le contenu de toute les cases mémoires situées entre l'adresse 40 et l'adresse 80.

	IN		
	STO	AD1	
	IN		
	STO	AD2	► On stocke les deux entrés AD1=40 et AD2=80
	LDA	AD1	
LOOP	STA	DTINST	► On transforme l'instruction suivante en LDA 40+ième itération de boucle suivant que l'on soit dans la 1ère, 2ième,...,nième itération
DTINST	LDA	TEM	► (40+ième itération de boucle)-> accumulateur
	OUT		► On affiche le contenu de l'accumulateur
	LDA	AD1	
	ADD	ONE	
	STO	AD1	
	SUB	AD2	
	BZ	STOP	► Si l'adresse considéré > AD2 alors STOP
	LDA	AD1	► Accumulateur=(41+ième itération de boucle)
	BR	LOOP	
	...	...	

<6 pts>

3b.

Cela n'a aucune importance puisque l'instruction précédente changera l'opérande de notre code machine à l'adresse symbolique DTINST par la donnée (IN)+ième itération de boucle. On a donc à l'adresse DTINST, à la fois une donnée et une instruction (ou plus précisément la première partie de ce code est une instruction et la deuxième partie est une donnée).
<3.5 pts>

3c.

La première instruction/donnée exécute à l'adresse symbolique DTINST sera donc **STO** 40 avec Acc=40. Il stockera donc la donnée 40 à l'adresse 40. Pour la deuxième itération de boucle, on aura **STO** 41 avec Acc=41, etc.. Bref, ce programme stockera donc la donnée i à l'adresse i avec $40 \leq i < 80$ et affichera aussi en sortie les valeurs $40, 41, 42, \ldots, 79$.
<3.5 pts>

II. Programmation Shell Script (26 pts)

1. **Gestion de Fichiers**

   On vous demande de créer un script ChgtName.sh permettant de changer tous les noms de fichiers qui possèdent dans leur nom la chaîne de caractère passé en premier paramètre de ce script, puis remplacer cette chaîne de caractère par la chaîne de caractère passé en deuxième paramètre de ce script.
   Par exemple une exécution du script ChgtName.sh IFT1214 IFT1215
   devra changer tous les noms de fichiers dans le répertoire courant, i.e.,

Chap1IFT1214.pdf		Chap1IFT1215.pdf
Chap2IFT1214.pdf		Chap2IFT1215.pdf
...		...
Dem1IFT1214.pdf	par	Dem1IFT1215.pdf
Dem2IFT1214.pdf		Dem2IFT1215.pdf
...		...

   Le script devra vérifier qu'il y a bien deux (et seulement deux) paramètres passés en argument sinon ce dernier affichera à l'écran Mauvais nombre d'argument puis interrompera le script.
   <12 pts>

2. **Manipulation/Modification de Fichiers**

   Voici le début d'un fichier (appelé LISTE)

```
INTC:19.06:24.28
MOT:22.07:27.20
AMD:31.73:38.85
AAPL:67.04:89.22
TXN:35.34:37.69
MSFT:27.15:31.17
GOOG:437.1:476.55
IBM:81.66:96.06
YHOO:32.89:42.00
EBAY:35.09:45.36
SYMC:16.15:23.10
AMZN:36.03:37.85
ORCL:14.28:15.59
CSCO:20.68:23.14
NT:2.80:3.41
DELL:27.01:36.59
```

   On vous demande de créer un script ClassOrd.sh permettant d'obtenir la liste par ordre décroissant des gains en pourcentage (tronqué à l'entier le plus proche) de chaque couple de valeurs. Le gain en pourcentage étant affiché et calculé par la relation suivante :

$$\frac{\text{valeur de droite - valeur de gauche}}{\text{valeur de gauche}} \times 100$$

   et sauvegardé dans le fichier que l'on appellera LISTE-CLASSEE. Ce script devra passer en paramètre le fichier LISTE (pour une exécution du style ClassOrd.sh LISTE et, dans le cas de ce fichier LISTE, donner un fichier LISTE-CLASSEE, du style ;

```
43%: SYMC:16.15:23.10
35%: DELL:27.01:36.59
33%: AAPL:67.04:89.22
29%: EBAY:35.09:45.36
27%: YHOO:32.89:42.00
27%: INTC:19.06:24.28
23%: MOT:22.07:27.20
22%: AMD:31.73:38.85
21%: NT:2.80:3.41
17%: IBM:81.66:96.06
14%: MSFT:27.15:31.17
11%: CSCO:20.68:23.14
9%: ORCL:14.28:15.59
9%: GOOG:437.1:476.55
6%: TXN:35.34:37.69
5%: AMZN:36.03:37.85
```

<14 pts>

Réponse

1a.

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


if [ $# -ne 2 ]; then
 echo Mauvais nombre d arguments
 exit
fi

for File in *
do
FileNew=`echo $File | tr $1 $2`

mv $File $FileNew

done
```

<12 pts>

Barème des 12 Pts :

Traitement Nb d'arguments : 3 Pts
Listing Fichier : 2 Pts
Transformation : 3 Pts
Commande mv finale : 2 Pts
Reste & Syntaxe : 2 Pts

Une autre version possible n'utilisant pas TR,

```
#! /bin/sh
# ChgtName2.sh script


if [ $# -ne 2 ]; then
 echo Mauvais nombre d arguments
 exit
fi

for File in *${1}*
do
 newFile=${File%${1}*}${2}${File##*${1}}
 mv $File $newFile
done
```

1b.

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

cat $1 | while read line
do

Price=`echo $line | cut -d : -f2`
Price2=`echo $line | cut -d : -f3`

DiffPrice=`echo $Price2 - $Price | bc -l`
Gain=`echo $DiffPrice*100/$Price | bc -l`
GainInt=`echo $Gain | cut -d . -f 1`

echo $GainInt\%: $line >> LISTEtmp.dat

done

sort -n -r LISTEtmp.dat >  LISTE-CLASSEE.dat

rm LISTEtmp.dat
```

<14 pts>

Barème des 14 Pts :
————————————

Récup. de l'argument : 1 Pt
Récup. Prix : 4 Pts
Calcul Gain : 3 Pt
Troncature : 1 Pt
Ecriture Fichier : 1 Pt
Trie : 1 Pt
Reste & Syntaxe : 2 Pts

III. Misc. (12 pts)
Réponder aux questions suivantes ;

1. **Disque Dur**
   Quelles sont les techniques modernes utilisées permettant d'accélérer la vitesse de transfert des données d'un disque dur.
   <4 pts>
2. **Mémoire**
   Définir rapidement ce qu'est la mémoire cache et expliquer pour quelles raisons le principe de mémoire cache permet de diminuer le temps moyen d'accès mémoire.
   <4 pts>
3. **Système**
   Par quelle(s) technique(s) un ordinateur peut il faire tourner plusieurs programmes simultanément ?
   <4 pts>

Réponse

1. (a) Principalement grâce à de la mémoire cache (ou mémoire tampon) que l'on met en tampon, i.e., entre le disque dur et le CPU. Cette mémoire cache permet de conserver les données auxquelles le disque accède le plus souvent afin d'améliorer les performances globales de transfert. On peut aussi faire tourner les plateaux plus rapidement ou augmenter la densité des informations écrite sur le disque dur ou encore l'utilisation du DMA (Direct Memory Access) ou encore une connexion fibre optique permet aussi d'avoir quelques gains en transfert. Une defragmentation de temps en temps est utile pour gagner aussi du temps de transfert.
   <4 pts>

(b) La cache est une mémoire rapide entre le CPU et la mémoire principale. Elle contient une copie d'une zone de mémoire centrale et permet de diminuer les temps d'accès en accélérant le traitement des instructions par l'unité de commande. Elle est souvent sollicité (et donc le temps moyen d'accès mémoire s'en trouve diminué) grâce au principe de localité des donnés (TEMPORELLE [on fait souvent référence à la même zone mémoire] et SPATIALE [on utilise souvent des zones mémoires voisines]). Celui-ci est basé sur le fait que les instructions d'un programme et de même les données qui sont utilisés dans un programme sont proches les unes des autres (On a donc tout a gagner à amener des blocs de mots consécutifs dans une mémoire à accès rapide.)
<4 pts>

(c) En utilisant le time sharing ou multitasking (temps partagé ou multitaches) ; le CPU passe rapidement d'une tache à la suivante à intervalles régulier (quantum). Chaque programme ou chaque usager reçoit une fraction de la puissance de l'ordinateur et a l'illusion d'être le seul usager, De plus le dispatcher utilise les temps d'attente du CPU durant les opérations d'I/O d'un programme pour l'execution d'un autre programme.
<4 pts>

IV. Codes correcteurs (22 pts)

1. **Codage de Hamming**
   On reçoit le message suivant 16516_8 qui est encodé avec un code de Hamming utilisant 4 bits de contrôle et une parité impaire. Retrouver la séquence originale en base 2 et octal en corrigeant les erreurs si nécessaire et indiquer ensuite en binaire quel est le message corrigé.
   <7 pts>

2. **Codage CRC**
   (a) On veut transmettre le message 110_8 en utilisant la méthode CRC avec le polynôme générateur $G(x) = x^4 + x^2 + 1$. Trouver le message à transmettre et le mettre en code hexadécimal.
   <10 pts>
   (b) On reçoit le message $B60_{16}$ utilisant un code CRC avec le polynôme générateur $G(x) = x^4 + x^2 + 1$. Le message a t-il été corrompu ?
   <5 pts>

Réponse

$\boxed{\textbf{1.}}$

On convertit d'abord le message en binaire.

16516_8 ► $001\,110\,101\,001\,110_2$ **<1 pt>**

Utilisons la méthode de Hamming simplifiée qui consiste à générer un nombre impaire (car la parité est impaire dans cet exemple) de bits à 1 sur chaque colonne des différentes positions du message pour laquelle le bit est égale à 1. La dernière ligne considérée étant la position en binaire d'un éventuel bit erroné. On obtient donc,

13	$\rightarrow$	1	1	0	1
12	$\rightarrow$	1	1	0	0
11	$\rightarrow$	1	0	1	1
09	$\rightarrow$	1	0	0	1
07	$\rightarrow$	0	1	1	1
04	$\rightarrow$	0	1	0	0
03	$\rightarrow$	0	0	1	1
02	$\rightarrow$	0	0	1	0
A_i	$\rightarrow$	1	1	1	0

<**4 pts**> $\boxed{\text{RJCF :2/4}}$

Cette opération (respectivement 5 bits à 1 dans toute les colonnes), nous indique que le bit à la position $1110_2 = 14_{10}$ est erroné. Le message corrigé est donc $011\,110\,101\,001\,110_2 = 36516_8$ <**1 pt**>. En enlevant les bits de contrôle (position 1, 2, 4, 8), on obtient la message binaire informatif suivant ; $01111011001_2 (= 1731_8)$. <**1 pt**>

2a.

• On convertit d'abord le message en binaire.

$$110_8 \quad \blacktriangleright \quad 1\,001\,000_2$$ <**1 pt**>

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

$$\begin{array}{ccccccccccc|ccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
\cline{12-16}
1 & 0 & 1 & 0 & 1 & & & & & & & & & & & \\
\cline{1-5}
0 & 0 & 1 & 1 & 1 & 0 & 0 & & & & & & & & & \\
 & & 1 & 0 & 1 & 0 & 1 & & & & & & & & & \\
\cline{3-7}
 & & 0 & 1 & 0 & 0 & 1 & 0 & & & & & & & & \\
 & & & 1 & 0 & 1 & 0 & 1 & & & & & & & & \\
\cline{4-8}
 & & & 0 & 0 & 1 & 1 & 1 & 0 & 0 & & & & & & \\
 & & & & & 1 & 0 & 1 & 0 & 1 & & & & & & \\
\cline{6-10}
 & & & & & 0 & 1 & 0 & 0 & 1 & 0 & & & & & \\
 & & & & & & 1 & 0 & 1 & 0 & 1 & & & & & \\
\cline{7-11}
 & & & & & 0 & 0 & 0 & 1 & 1 & 1 & & & & &
\end{array}$$

<**7 pts**> $\boxed{\text{RJCF :2/7}}$

• On transmet le message $\underbrace{1001000}_{M(x)}\underbrace{0111}_{R(x)} = 2207_8 = 487_{16}$. <**2 pts**>

2b.

• On convertit d'abord le message en binaire.

$$B60_{16} \quad \blacktriangleright \quad 1011\,0110\,0000_2$$ <**1 pt**>

• On réalise ensuite la division modulo 2 suivante,

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

Le reste $R(x) = 1101$ est non nul donc le message reçu est corrompu. $<$**5 pts**$>$ $\boxed{\text{RJCF :2/5}}$