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 : *27/04/2007*

I	Assembleur/LMC/Registres (49 pts)
II	Programmation Shell Script (26 pts)
III	Misc. (18 pts)
IV	Codes correcteurs (15 pts)
Total	108 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 (49 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 permettant d'afficher en sortie le résultat (U_n) de la suite de Fibonacci donnée par la relation $U_{n+2} = U_{n+1} + U_n$ avec $U_0 = 0$ et $U_1 = 1$. Le programme devra afficher les valeurs de cette suite qui sont inférieur à n. n sera la première valeur (> 1) entrée en mémoire au début du programme par l'instruction IN.
   En résumé, si on rentre 14 en mémoire au début du programme, celui-ci devra afficher en sortie 0, 1, 1, 2, 3, 5, 8, 13 (cad les 2 premières valeurs qui sont toujours 0 et 1 puis $0+1 = 1$, $1+1 = 2$, $3+2 = 5$, $5 + 3 = 8$, 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

FIGURE 1 – Jeu d'instructions du LMC

2. **Langage LMC et Registres**

   Voici un programme en code d'instructions mnémoniques

```
        IN
        STO   NB
        IN
        STO   CMP
LOOP    LDA   RES
        OUT
        ADD   THREE
        OUT
        SUB   TWO
        OUT
        STO   RES
        SUB   CMP
        BRZ   END
        LDA   CPT
        ADD   ONE
        STO   CPT
        SUB   NB
        BRP   LOOP
END     HLT
NB      DAT   0
CMP     DAT   0
THREE   DAT   3
TWO     DAT   2
ONE     DAT   1
RES     DAT   1
CPT     DAT   0
```

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

      i. $(4, 7)$ (i.e., 4 pour le premier IN et 7 pour le second IN)

      ii. $(8, 2)$

      iii. $(8, 9)$

      iv. Dans le cas général
      <8 pts>

(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) et en supposant que toutes les instructions sont mises à la suite des autres.
<6.5 pts>

(c) Donner la valeur des différents registres ACCU, IR, PC, MAR, MDR à la fin (de la phase exécution) des instructions **LDA RES** (5-ième instruction) et **ADD THREE** (7-ième instruction pour la première entrée dans la boucle BEGIN) pour les deux inputs suivant $(4, 7)$.
<7.5 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 mais pas obligatoirement), un programme permettant d'initialiser toutes les cases mémoires situées 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. On commentera intelligemment le programme de façon à ce que celui ci soit lisible et facilement compréhensible.
<12 pts>

Réponse

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

Un exemple possible, valable $\forall\ n > 0$

	IN		
	STO	BORNESUP	Enregistrement Borne Sup.
	LDA	VALN	
	OUT		
	LDA	VALN+1	
	OUT		Affichage des 2 premiers termes
LOOP	LDA	VALN+1	
	ADD	VALN	
	OUT		Affichage $U_{n+1} = U_n + U_{n-1}$
	STO	TEMP	
	LDA	VALN+1	
	STO	VALN	U_n à l'adresse VALN
	LDA	TEMP	
	STO	VALN+1	U_{n+1} à l'adresse VALN+1
	LDA	BORNESUP	
	SUB	VALN+1	
	SUB	UN	
	BRP	LOOP	
END	HLT		Test si U_{n+1} > BORNESUP
VALN	DAT	0	
VALN+1	DAT	1	
TEMP	DAT	1	
BORNESUP	DAT	0	
UN	DAT	1	

0. Nota : La suite de Fibonacci, inventée par Leonardo Pisano (connu sous le pseudonyme Fibonacci (1170-1250)), est l'une des suites mathématiques les plus connues. Cette suite a été initialement inventée pour décrire la croissance mensuelle d'une population de lapins partant du principe que "Possédant initialement un couple de lapins, combien de couples obtient-t-on chaque mois si chaque couple engendre tous les mois un nouveau couple à compter du second mois de son existence ? " Dans ce modèle, on suppose que les lapins ne meurent jamais. En calculant les valeurs approchées des quotients de deux nombres successifs de la suite de Fibonacci, on trouve le célèbre nombre d'or) intervenant dans de nombreux domaines tels que que la géométrie, la peinture et la nature (fleur de tournesol, certains coquillages, le chou ou sur la pomme de pin). L. de Vinci le verra dans les proportions du corps humain.

<15 pts>

IFT1215$_{H}07_{F}inal_{c}orrection.dvi$

2a.

- Pour $(4,7)$, le programme affichera en sortie les nombres suivants :
  1 4 2 puis s'arrêtera.
  <2 pts>

- Pour $(8,2)$, le programme affichera en sortie les nombres suivants :
  1 4 2 puis s'arrêtera.
  <2 pts>

- Pour $(8,9)$, le programme affichera en sortie les nombres suivants :
  1 4 2 puis s'arrêtera.
  <2 pts>

- On aura la suite 1 4 2 sauf si le premier chiffre à l'entrée (premier IN) est égale à un ou zéro (ou inferieur à zéro), dans ce cas, on aura la suite 1 4 2 si la deuxième entrée (deuxième IN) est 2 ou la suite 1 4 2 2 5 3 3 6 4 si la deuxième entrée est 4 ou la suite 1 4 2 2 5 3 3 6 4 4 7 5 si la deuxième entrée est 5 ou la suite 1 4 2 2 5 3 3 6 4 4 7 5 5 8 6 6 9 7 si la deuxième entrée est 7 etc. ...

  En pseudo code, avec <premier IN>=NB et <deuxième IN>=CMP, cela pourrait s'exprimer comme cela

Cpt=0	
Res=1	
Loop	Afficher Res=Res+3
	Afficher Res=Res-2
	Si [Res == CMP] –> END
	else ct=cpt+1
While [nb $\leq$ 1]	–> goto Loop

<2 pts>

2b.

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

Ad.	Code	Ind. Reloc.
00	901	0
01	319	1
02	901	0
03	320	1
04	524	1
05	901	0
06	121	1
07	901	0
08	222	1
09	901	0
...	...	

<6 pts>

2c.

	ACCU	IR	PC	MAR	MDR
Fin de **LDA RES**	001	524	005	024	001
Fin de **ADD THREE**	004	121	007	021	003
	<1.5 pts>	**<1.5 pts>**	**<1.5 pts>**	**<1.5 pts>**	**<1.5 pt>**

3a.

Un exemple de programme

Nota : On peut remplacer l'instruction **STA DTINST** par les deux instructions suivantes **ADD TROIS-CENT** et **STO DTINST** et en ajoutant à l'adresse symbolique **TROIS-CENT** la donnée 300.

L'énoncé ne nous précise pas avec quelle valeur on doit initialiser ce tableau ; on le fera donc avec des valeurs égales à zéro dans cet exemple.

	IN		
	STO	AD1	
	IN		
	STO	AD2	On stocke les deux entrés (adresses) AD1 et AD2
	LDA	AD1	
LOOP	STA	DTINST	On transforme l'instr. à l'adr. DTINST en STO AD1+ième itér. de boucle suivant que l'on soit dans la 1ère, 2ième,...,nième itér.
LOOP	LDA	ZERO	
DTINST	STO	TEM	STO AD1+ième itér. de boucle : 0 écrit à l'adr. (AD1+ième itér.)
	LDA	AD1	
	ADD	ONE	
	STO	AD1	AD1+1 → AD1
	SUB	AD2	
	BZ	STOP	Si l'adresse considéré > AD2 alors STOP
	LDA	AD1	Accumulateur=(AD+1+ième itération de boucle)
	BR	LOOP	
STOP	HLT		
AD1	DAT	0	
AD2	DAT	0	
ONE	DAT	1	
ZERO	DAT	0	
TEM	DAT	0	

<12 pts>

II. Programmation Shell Script (26 pts)

1. **Gestion de Fichiers**

   On vous demande de créer un script "CompressPS.sh" permettant de

   i. rechercher dans le répertoire où vous le lancer ainsi que tous ses sous répertoires et sous-sous répertoires etc. tous les fichiers ayant une extension .ps (i.e., fichiers postscripts)

   ii. de voir si ce fichier existe avec l'extension .dvi (cad "est du même nom et est au même endroit avec l'extension .dvi") et si cela s'avère être le cas d'écrire à l'écran Je pourrais comprimer le fichier <Nom du fichier>.ps.

<13 pts>

2. **Manipulation/Modification de Fichiers**

   Vous trouverez en Fig. (2), le début d'un fichier (appelé OUTPUT-PROG.DAT) qui est en fait l'enregistrement de l'output (i.e., sortie terminal) d'un programme (l'expression "**...** **...**" signifie que le programme continue avec la même syntaxe).

   On vous demande de créer un script EXTRACTGAM-SNR.SH permettant d'obtenir deux colonnes, l'une concernant la valeur de GAMMA et sur la deuxième colonne la valeur de BEST MAP-SNR associée pour chacun des tests effectués (TEST 0, TEST 1, etc.) (pour éventuellement afficher ensuite le graphe de la valeur de SNR en fonction de Gamma avec un logiciel).

```
>>>>>>>>>>>> TEST 0 <<<<<<<<<<<<
> Gamma = 0.08333

Boucle PE-EM
[0] >   MAP=864980608.000
[1]  >   MAP=680152.7500
        ... ...
[38]  >   MAP=4615.0396
[39]  >   MAP=4596.9536

Resultat Reconstruction
MMSE = 75604.2578125    SNR = 10.00 dB  [10.00][39 iters]
BEST-MAP = 9.18 dB


>>>>>>>>>>>> TEST 1 <<<<<<<<<<<<
> Gamma = 0.09436

Boucle PE-EM
[0] >   MAP=864977024.000
[1]  >   MAP=85937.7812
       ...   ...
[298]  >   MAP=2072.3049
[299]  >   MAP=2071.5813

Resultat Reconstruction
MMSE = 0.0872052    SNR = 9.35 dB  [10.69][43 iters]
BEST-MAP = 9.38 dB


>>>>>>>>>>>> TEST 2 <<<<<<<<<<<<
> Gamma = 0.10684

Boucle PE-EM
[0] >   MAP=864977024.000
[1]  >   MAP=86068.7812
      ...  ...
[298]  >   MAP=2072.9424
[299]  >   MAP=2072.2229


Resultat Reconstruction
MMSE = 0.0870771    SNR = 9.34 dB  [10.69][44 iters]
BEST-MAP = 9.55 dB


    ... ...          ... ...
```

FIGURE 2 – Fichier OUTPUT-PROG.dat

   En résumé, on aimerait que le stdout (i.e., terminal de l'ordinateur) affiche donc

0.083333	9.18
0.09436	9.38
0.10684	9.55
...	...

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

   EXTRACTGAM-SNR.SH OUTPUT-PROG.DAT .

   <13 pts>

Réponse

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

Un exemple de script pourrait être

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

find -name \*.ps | while read line; do
echo $line
file=${line##/*/}
dvifile=${file%.*}.dvi
if [ -f $dvifile ]; then
echo je pourrais compresser $file
fi
done
```

<13 pts>

Barème des 13 Pts :
——————————

Lecture Récursive répertoires : 2 Pts
Récup. chaque fichier : 2 Pts
Enlever Path : 2 Pts
Enlever/Mettre extension : 2 Pts
Condition DVI existe : 2 Pt
Reste & Syntaxe : 3 Pts

Une autre version n'utilisant pas les manipulation de chaîne de caractères (pour enlever le path et l'extension) pourrait être

```
#! /bin/sh
# CompressPS2.sh script

find -name \*.ps | while read line
do
nom=`echo $line | cut -d . -f2 | cut -d "/" -f 2-`.dvi
file=`ls $nom`
if [ $file ]; then echo je pourrais compresser $file
fi done
```

Nota : Dans ce script, on aurait put être moins concis en écrivant find -name .ps > tmp.file puis cat tmp.file | while read line au lieu de find -name .ps | while read line.

2.
Un exemple de script utilisant deux fichiers temporaires tmp.dat et tmp2.dat pourrait être

```
#! /bin/sh
# ExtractGAM-SNR.sh script

File=$1

if [ -f tmp.dat ]
then
rm tmp.dat
fi
if [ -f tmp2.dat ]
then
rm tmp2.dat
fi

cpt=1
grep "Gamma = " $File | cut -d "=" -f 2 > tmp.dat
grep "BEST-MAP" $File | cut -d "=" -f 2 | cut -d "d" -f 1 > tmp2.dat

cat tmp.dat | while read line
do
secondline=`head -n $cpt tmp2.dat | tail -n 1`
echo $line $secondline
let cpt=cpt+1
done
```

<13 pts>

Barème des 13 Pts :
———————————
Récup. de l'argument : 2 Pt
Récup. de Gamma : 3 Pt
Récup. de SNR : 3 Pts
Affichage Correcte : 2 Pt
Reste & Syntaxe : 3 Pts

En un peu plus court, mais faisant la même chose, on pourrait trouvé

```
#! /bin/sh
# ExtractGAM-SNR2.sh script

cat $1 | while read line; do
a1=`echo $line | grep Gamma  | cut -d = -f 2`
a2=`echo $line | grep BEST | cut -d = -f 2 | cut -d d -f 1`
if [ $a1 ]; then  echo -n $a1
fi
if [ $a2 ]; then echo " $a2"
fi
done
```

En version "*ultra short*" pour informaticien pressé, pressé, ...

```
cat $1 | while read line; do for nm in Gamma BEST; do coord=`echo $line | grep $nm  | cut -d = -f 2 | cut -d d -f 1`
if [ $coord ]; then echo -n " $coord" && if [ $nm == BEST ]; then echo ""
fi fi done done
```

III. Misc. (18 pts)

Répondre aux questions suivantes ;

1. **Mémoire**
   Quelles sont les techniques modernes utilisées permettant à un système informatique de faire fonctionner un programme qui demande plus de place mémoire que celle installée physiquement (sous forme de barrettes RAM) dans le système ? Expliquer et discuter les inconvénients de cette technique.
   <4 pts>
2. **Système**
   Par quelle(s) technique(s) un ordinateur peut il être multi-utilisateurs ? Expliquer brièvement comment cela marche.
   <4 pts>
3. **Code correcteur**
   Par quelle technique peut on détecter et corriger plusieurs erreurs avec le code de Hamming. Décrire brièvement et préciser dans votre exemple le nombre d'erreurs consécutifs que l'on peut détecter et corriger.
   <4 pts>
4. **Télé-Informatique**
   Si on transmet une communication série asynchrone avec un bit de start et deux bit de stop. De combien, au minimum, sera plus lent cette communication (exprimer le en pourcentage par exemple) comparativement à une communication série synchrone.
   <3 pts>
5. **Télé-Informatique**
   Si une catastrophe nucléaire détruisait la plupart des infrastructures de communication, serait il

possible qu'internet fonctionne ? Expliquez qu'elles seraient les raisons qui permettrait à internet, eventuellement, de rester fonctionnelle ?
<3 pts>

Réponse

1. Cette technique s'appelle le *paging* et consiste, après avoir divisé le programme en blocs de mémoire (ou pages) de transférer la page dont l'ordinateur a besoin dans la mémoire primaire (ou physique). Il existe ensuite tout un processus, géré par le système d'exploitation, permettant au programme de convertir une adresse du programme en une adresse physique et de recopier le bloc mémoire sous disque dur si ce bloc a été modifié durant l'exécution du programme. La contrepartie d'un tel dispositif est le *swapping*, cad l'échange nécessaire des données entre le disque dur et la mémoire qui ralentit considérablement l'ordinateur car l'accès disque est considérablement plus lent (facteur 1000) que l'accès d'une case mémoire physique.
<4 pts>
2. Le système informatique utilise la même technique que celle utilisée pour faire fonctionner plusieurs programmes en parallèle. Il utilise le *time sharing* ou *multitasking* (temps partagé ou multitaches) ; le CPU passe rapidement d'un usager au suivant par intervalles de temps régulier (tranche de temps appelée quantum). Chaque usager reçoit une fraction de la puissance de l'ordinateur et a l'illusion d'être le seul usager. Le dispatcher utilise aussi les temps d'attente du CPU durant les opérations d'I/O d'un programme d'un utilisateur pour l'exécution d'un autre programme d'un autre utilisateur.
<4 pts>
3. Il suffit par exemple d'associer le code de Hamming nécessaire pour chaque donnée que l'on désire envoyer (par exemple des caractères ASCII) et d'envoyer ensuite les bits colonne par colonne (i.e., premier bit du premier caractère, premier bit du deuxième caractère, ..., premier bit du dernier caractère puis deuxième bit du premier caractère, etc). Par cette technique, à la réception, une seule erreur par caractère sera présent (et donc détectable et corrigible) si il n'y a pas plus de *nb* erreurs consécutifs où *nb* est le nombre de données rassemblé en colonne.
<4 pts>
4. Au minimum de $\frac{8+3}{8} = 1.37$, cad 37% plus lente, borne minimal, puisqu'on est dans le cas ou il n'y a aucun temps mort entre les différents caractères envoyés.
<3 pts>
5. Internet a la capacité d'utiliser de véhiculer l'information vers les infrastructures qui seraient non détruites (par exemple après une catastrophe nucléaire, d'ailleur Internet a initialement été créé pour cette raison) permettant ainsi au réseau des réseaux de rester fonctionnel (d'où l'image de la toile d'araignée).
<3 pts>

IV. Codes Correcteurs (15 pts)

1. **Codage de Hamming**
On veut envoyer le message suivant 154_8 avec un code de Hamming utilisant une parité paire. Retrouver la séquence binaire que l'on doit transmettre, et la convertir en Hexa.
<7 pts>

2. **Codage CRC**
On veut transmettre le message $A0_{16}$ en utilisant la méthode CRC avec le polynôme générateur $G(x) = x^5 + x^3 + 1$. Trouver le message à transmettre et le mettre en code hexadécimal.
<8 pts>

Réponse

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

On convertit d'abord le message en binaire

$$154_8 \quad \blacktriangleright \quad 1\,101\,100_2 \qquad <\textbf{1 pt}>$$

Le message contient 9 bits informatifs, les bits de contrôle s'insèreront en position 1 (k_1), 2 (k_2), 4 (k_3) et 8 (k_4) (puissance de deux) de la façon suivante :

$$1\ 1\ 0\ k_4\ 1\ 1\ 0\ k_3\ 0\ k_2\ k_1$$

Utilisons la méthode de Hamming simplifiée qui consiste à calculer la somme modulo 2 (car la parité est paire dans cet exemple) des positions de bits à 1. La dernière ligne considérée étant la valeur de k_4, k_3, k_2, k_1. On obtient donc

$$\begin{array}{cccccc}
11 & \rightarrow & 1 & 0 & 1 & 1 \\
10 & \rightarrow & 1 & 0 & 1 & 0 \\
07 & \rightarrow & 0 & 1 & 1 & 1 \\
06 & \rightarrow & 0 & 1 & 1 & 0 \\
\hline
k_i & \rightarrow & 0 & 0 & 0 & 0
\end{array}$$

$<$**5 pts**$>$ **RJCF :2.5/5**

On obtient donc le message binaire a transmettre suivant ; $0110\,0110\,0000_2 (= 660_{16})$. $<$**1 pt**$>$

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

• On convertit d'abord le message en binaire

$$A\,0_{16} \quad \blacktriangleright \quad 1010\,0000_2 \qquad <\textbf{1 pt}>$$

• Après avoir ajouté 5 zéros au message (5 correspondant au degré du polynôme générateur), on réalise ensuite la division modulo 2 suivante

$$\begin{array}{ccccccccccccc|cccccc}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
 & & & & & & & & & & & & & – & – & – & – & – & – \\
1 & 0 & 1 & 0 & 0 & 1 & & & & & & & & & & & & & \\
– & – & – & – & – & & & & & & & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & & & & & & & & \\
 & & & & & 1 & 0 & 1 & 0 & 0 & 1 & & & & & & & & \\
 & & & & & – & – & – & – & – & – & & & & & & & & \\
 & & & & & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & & & & & & \\
 & & & & & & & 1 & 0 & 1 & 0 & 0 & 1 & & & & & & \\
 & & & & & & & – & – & – & – & – & – & & & & & & \\
 & & & & & & & 0 & 0 & 1 & 1 & 0 & 1 & & & & & & \\
 & & & & & & & & & & & & & & & & & &
\end{array}$$

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

• On transmettra donc le message $\underbrace{10100000}_{M(x)}\underbrace{01101}_{R(x)} = 140D_{16}$. $<$**2 pts**$>$