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 : *24/04/2009*

I	Assembleur/LMC/Registres (57 pts)
II	Programmation Shell Script (23 pts)
III	Misc. (16 pts)
IV	Codes correcteurs (15 pts)
Total	111 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 (possiblement nul pour a mais pas pour b) 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 division entière entre a et b (i.e., $\lfloor \frac{a}{b} \rfloor$ ou partie entiere de a/b).
   Par exemple, si on entre 3 et 4, le programme devra calculer $\lfloor \frac{3}{4} \rfloor = 0$. Par exemple, si on entre 13 et 4, le programme devra calculer $\lfloor \frac{13}{4} \rfloor = 3$. Par exemple, si on entre 13 et 7, le programme devra calculer $\lfloor \frac{13}{7} \rfloor = 1$, etc. On supposera que le résultat est communiqué en sortie par l'instruction OUT. L'ajout de quelques commentaires à coté du programme seront appréciés.
   <16 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              ...0
        STO   NA
        IN
        STO   NB
        SUB   NA
        BRZ   END
LOOP    LDA   NA
        STO   RST
        SUB   NB
        BRP   L1
        BR    END       ...10
L1      STO   NA
        BRP   LOOP
END     LDA   RST
        OUT
        HLT
RST     DAT   0
NB      DAT   0
NA      DAT   0         ...18
```

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

      ii. $(2, 4)$

      iii. $(13, 4)$

      iv. $(7, 3)$

      v. Dans le cas général

   <8 pts>

   (b) Convertir les 13 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 dispersées possible.
   <7 pts>

(c) Donner la valeur des différents registres ACCU, IR, PC, MAR, MDR à la fin (de la phase exécution) de l'instruction **STO RST** (8-ième instruction) pour la première entrée dans la boucle LOOP pour l'input suivant $(13, 4)$.
<**5 pts**>

(d) On pourrait optimiser ce programme (dans le but qu'il aille un peu plus vite) en modifiant et/ou enlevant une/plusieurs des instructions machines. La(les)quelle(s) ?
<**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 si vous le desirez [pour simplifier un peu ce programme]), un programme permettant de déplacer (i.e., copier) un bloc de donnés ou d'instructions situé entre $adr1$ et $adr2$ en $adr3$ et $adr4$.
$adr1$, $adr2$, $adr3$ et $adr4$ étant quatres valeurs entrées en mémoire au début du programme par l'instruction IN et on supposera que ces adresses entrées par IN seront toujours tel que :

(a) $99 > adr4 > adr3 > adr2 > adr1$

(b) $adr4 - adr3 = adr2 - adr1$

On commentera (un peu) et intelligemment le programme de façon à ce que celui ci soit lisible et facilement compréhensible.
<**16 pts**>

Réponse

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

Un exemple possible, valable $\forall\, a \geq 0$ et $b > 0$

```
        IN
        STO   INTA   | Enregistrement Entier A.
        IN
        STO   INTB   | Enregistrement Entier B.

LOOP    LDA   INTA   |
        BRZ   END    | Si [(INTA) == 0] => Fin Prog
        SUB   INTB   | On soustrait (INTB) a (INTA)
        STO   INTA   | autant de fois que le
        BRP   CONT   | résultat > 0 [I]

END     LDA   RES    | Fin Prog. Affichage Résultat
        OUT          |
        HLT          |

CONT    LDA   RES    |
        ADD   ONE    | On compte le nb de fois
        STO   RES    | qu' [I] a été réalisé => RES
        BR    LOOP   |

INTA    DAT   0      | Stockage Données
INTB    DAT   0      |
RES     DAT   0      |
ONE     DAT   1      |
```

<**16 pts**>

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

- Pour $(4, 4)$, le programme affichera en sortie 0.
  <**1.5 pts**>
- Pour $(2, 4)$, le programme affichera en sortie 2.
  <**1.5 pts**>
- Pour $(13, 4)$, le programme affichera en sortie 1.
  <**1.5 pts**>
- Pour $(7, 3)$, le programme affichera en sortie 1.
  <**1.5 pts**>

- Dans le cas général, ce programme affichera le reste de la division entière de NA par NB. Mathématiquement, il affichera ce qu'on appelle le modulo (%), i.e., $NA\%NB$.
  <**2 pts**>

2b.

Les 13 premières conversions en code machine et son indicateur de relocation se trouvent immédiatement

Ad.	Code	Ind. Reloc.
00	901	0
01	318	1
02	901	0
03	317	1
04	218	1
05	713	1
06	518	1
07	316	1
08	217	1
09	811	1
10	613	1
11	318	1
12	806	1
13	516	1
...	...	

<**5 pts**> + <**2 pts**> (Reloc)

2c.

	ACCU	IR	PC	MAR	MDR
Fin de **STO RST**	013	316	008	016	13
	<**1 pt**>	<**1 pt**>	<**1 pt**>	<**1 pt**>	<**1 pt**>

2d. Par ordre d'importance :

- On pourrait s'apercevoir que la variable RST est inutile, de ce fait l'instruction STO RST (6 ième instruction) pourrait être supprimée mais on devrait alors remplacer l'instruction LDA RST (à l'adresse END) par LDA NA.
- Une petite modification qui permettrait de gagner quelques cycles machines consisterait à mettre l'étiquette **LOOP** une ligne plus basse. On éviterait ainsi une instruction inutile qui se fait dans ce programme autant de fois que $\lfloor \frac{NA}{NB} \rfloor$ (où $\lfloor \frac{NA}{NB} \rfloor$ désigne la partie entiere de la division de NA par NB).
- Une petite modification consisterait à déplacer tout le bloc de trois instruction à partir de l'adresse END et le mettre à la place de l'instruction BR END ; ce qui permettrait de gagner une instruction machine en fin de programme.

<**5 pts**>

3.

Un exemple possible utilisant l'instruction **STA**

```
            IN
            STO     SRC_DEB
            IN
            STO     SRC_END
            IN
            STO     DES_DEB
            IN
            STO     DES_END    | Enr. & Stok. des 4 adrs
LOOP        LDA     SRC_DEB
            STA     READ       | †
            LDA     DES_DEB
            STA     WRIT       | ‡
READ        LDA     NUL        | convertit par † en LDA (SRC_DEB)
WRIT        STO     NUL        | convertit par ‡ en STO (DES_DEB)
            LDA     SRC_DEB
            ADD     ONE
            STO     SRC_DEB    | SRC_DEB+1 → SRC_DEB
            LDA     DES_DEB
            ADD     ONE
            STO     DES_DEB    | DES_DEB+1 → DES_DEB
            LDA     SRC_END
            SUB     SRC_DEB
            BRZ     END        | Test fin copiage bloc ?
            BR      LOOP
END         HLT                | Fin Prog.
SRC_DEB     DAT     0          | Stockage Données
SRC_END     DAT     0
DES_DEB     DAT     0
DES_END     DAT     0
ONE         DAT     1
NUL         DAT     0
```

<16 pts>

II. Programmation Shell Script (23 pts)

1. **Gestion de Fichiers**

   On vous demande de créer un script "ChgtNameSrcDest.sh passant en paramètre deux chaînes de caractères (pour une exécution du style ChgtNameSrcDest.sh A22 B30) et permettant de changer la première chaîne de caractères dans le nom de toute les images au format pgm existant dans le répertoire courant par la deuxième chaînes de caractères.
   Par exemple, si il y a l'ensemble des images suivantes (dans le répertoire courant) :

   ```
   210088_320.ppm_A22.pgm
   21077_320.ppm_A22.pgm
   216066_320.ppm_A22.pgm
   ...
   ```

   Une exécution de style ChgtNameSrcDest.sh A22 B30
   devra les renommer en

   ```
   210088_320.ppm_B30.pgm
   21077_320.ppm_B30.pgm
   216066_320.ppm_B30.pgm
   ...
   ```

   Le script verifiera qu'il n'existe pas de fichier du même nom (existant dans ce même répertoire) que le script pourrait effacer.
   <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 FindBadScore.sh permettant d'obtenir le nom de toute les images ayant un score (en flottants) strictement inférieur à celui spécifié en paramètre à ce script. Plus précisément, on aimerait que le stdout (i.e., terminal de l'ordinateur) affiche en sortie, un classement croissant du score

0.57884	210088_320.ppm_A22.pgm
0.75288	21077_320.ppm_A22.pgm
0.76427	216066_320.ppm_A22.pgm
...	...

pour une exécution du style (par exemple)

FindBadScore.sh FILE4.dat 0.80 ,

<13 pts>

Réponse

1.

Un exemple de script pourrait être

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

for NameImg in *${1}.pgm
do

ImgWhitoutExt=${NameImg%_*}

if [ ! -f ${ImgWhitoutExt}_${2}.pgm  ]; then
mv $NameImg ${ImgWhitoutExt}_${2}.pgm
fi

done
```

Barème des 10 Pts :

Recup des arguments : 2 Pts
Ligne For : 2 Pts
Enleve l'extension : 2 Pts
Test [fichier n'existe pas] : 2 Pts
Mv : 1 Pts
Reste Algo & Syntaxe : 1 Pts

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

```
for NameImg in *${1}.pgm ; do [ ! -f ${NameImg%_*}_${2}.pgm  ] && mv $NameImg ${NameImg%_*}_${2}.pgm; done
```

2.

Un exemple de script pourrait être

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

#Parametres
BelowVal=`echo ${2}*100 | bc -l | cut -d "." -f 1`

#Nettoyage
if [ -f tmp1.dat ]
then
rm tmp1.dat
fi

#Boucle
cat ${1} | while read line
do

val=`echo $line | cut -d " " -f 1`
newval=`echo $val*100 | bc -l | cut -d "." -f 1`

if [ $newval -lt $BelowVal ]
then
echo $line >> tmp1.dat
fi

done

#Afichage

sort tmp1.dat
```

Barème des 13 Pts :

Recup score : 3 Pts
Isole img dessous score : 3 Pts
Tri croisant : 2 Pts
Nettoyage : 1 pt
Astuce test entier : 3 Pts
Reste Algo & Syntaxe : 1 Pts

version ultra-short qui fonctionne tout aussi bien, i.e.

```
#! /bin/sh
cat ${1} | sort | while read line; do val=`echo $line | cut -d " " -f 1`
if [ `echo $val \< ${2} | bc` -gt 0 ];  then echo $line; fi; done
```

III. Misc. (16 pts)
Répondre aux questions suivantes

1. Expliquez, en utilisant vos connaissances d'architecte informaticien, quelles sont en fait les caractéristiques nécessaires qui entrent en compte (et/ou les différentes composantes qu'il faudrait ajouter) si on souhaite définir (ou acheter) un ordinateur (ou tout système informatique) qui soit performant en terme de vitesse d'exécution ?
   Classez les par ordre décroissant d'importance, en justifiant d'abord pour chacune d'entre elle, dans quel contexte et/ou pourquoi, celle ci permettrait à l'ordinateur d'augmenter sa performance en terme de rapidité.
   <8 pts>
2. Imaginez que vous soyez en train de réaliser le prototype (i.e., la première version) d'un programme informatique qui réalise un certain traitement numérique (par exemple le débruitage d'une image ou le calcul de la solution d'un système d'équations). Vous vous êtes assuré que le programme soit sans bug et vous constatez que celui-ci réalise ce qu'on lui demande en un temps x secondes.
   Vous décidez d'ajouter quelques fonctionnalités à ce programme (par exemple une interface graphique, des options, la possibilité d'utiliser plusieurs méthodes différentes pour réaliser le traitement demandé, etc.) et vous remarquez que le temps de traitement a été multiplié par deux ! (sans avoir changé la routine principale qui réalise le traitement demandé !). En tant qu'architecte informaticien, expliquez (en justifiant bien votre réponse) ce qui pourrait être responsable de cela dans les deux cas suivants
   (a) Vous avez rajouté de nombreuses structures de données (en plus de celle utilisé dans la routine principale) à votre programme (ou vous avez décidé d'utiliser de plus grande structure de données).
   (b) Vous n'avez pas vraiment ajouté plus de structure de données (ou pas modifié les structures de donné existantes) mais seulement ajouté plus de fonctionnalités au programme (interface graphique convivial, nombreuses options, etc.)

   <8 pts>

Réponse

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

1. (a) Cadencage du processeur (vitesse de l'horloge)
   (b) Architecture du processeur superscalaire mieux que scalaire mieux qu'un processeur nécessitant plusieurs période d'horloge pour la phase fetch et execution d'une instruction.
2. Présence et importance de mémoire cache rapide.
3. Rapidité et importance de la mémoire vive (importance pour éviter tout problème de swap).
4. Rapidité du bus (i.e.. carte maîtresse de qualité).
5. Rapidité du disque dur.
   (a) Efficacité du DMA.
   (b) Pour les jeux, présence d'une carte graphique

<8 pts>

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

$\boxed{\textbf{a}}$ 2 possibilités dans ce cas. En utilisant de plus grande structure de données, il est possible que l'ordinateur *swape* c'est à dire qu'il n'y ait pas assez de mémoire vive physique (sous forme de barettes RAM) dans l'ordinateur, L'ordinateur utilise donc une partie du disque dur comme mémoire virtuelle, et se met à échanger des pages de données avec la mémoire physique, ce qui ralentit le système. On peut facilement savoir si il s'agit de ce problème en regardant la led rouge du disque dur pour savoir si celui ci est activement sollicité.

La deuxième possibilité, vient du fait que les structures de données précédemment entraient souvent entièrement dans la mémoire cache, ce qui n'est plus le cas maintenant.

$\boxed{\textbf{b}}$ Dans ce cas, les deux hypothèses précédentes sont valables mais la première (SWAPPING) est tout de même la plus vraissemblable.

<8 pts>

IV. Codes Correcteurs (15 pts)

1. **Codage CRC**
   On veut transmettre le message AB_{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 octal.
   <8 pts>
2. Vérifiez que le message que vous trouvez ne vous donnera aucune erreur en réception.
   <4 pts>
3. Quel sont les avantages et inconvénients de ce code par rapport aux autres codes correcteurs vus en cours.
   <3 pts>

Réponse

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

• On convertit d'abord le message en binaire

$$A\,B_{16} \quad \blacktriangleright \quad 1010\,1011_2 \qquad \textbf{<1 pt>}$$

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

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

<5 pts> $\boxed{\textbf{RJCF :2/5}}$

• On transmettra donc le message $\underbrace{10101011}_{M(x)}\underbrace{01100}_{R(x)}$ $=156C_{16}$ (ou 12554_8). **<2 pts>**

$\boxed{2.}$

Dans ce cas, il suffit de refaire la division

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

<4 pts> **RJCF :2/4**

$\boxed{3.}$

Peut détecter plusieurs erreurs (erreurs groupés) mais ne peut les corriger.

<3 pts>