Introduction à l'informatique II (CSI 1501)

Professeur: Marcel Turcotte

Session hiver: 2001, durée de l'examen: 3 heures

Identification

Nom: _____ prénom: _____

Numéro d'étudiant: _____ numéro du groupe de démonstration: _____

Nom du démonstrateur: _____

Consignes

1. Livres fermés,
2. Sans calculatrice, ou toute autre forme d'aide,
3. Il y a 11 questions. Répondez à toutes les questions.
4. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle.
5. Ne retirez pas l'agrafe.

Barème

1	2	3	4	5	6	7	8	9	10	11	**Total**
2	6	10	20	20	22	4	3	5	3	5	**100**

Question 1 (2 points)

Quelle est l'étiquette de l'instruction qui sera exécutée à la suite de l'instruction `RJMP` dans ce programme TC1101?
Notes:

- `RJMP` et `LDAI` sont les instructions de "branchement relatif" et "chargement d'une valeur immédiate" ajoutées au TC1101 dans le cadre du devoir 10.

```
[01]  CLA
[02]  INC
[03]  LDAI 4
[04]  CLA
[05]  INC
[06]  STA X
[07]  RJMP -5
[08]  LDA Y
[09]  INC
[10]  HLT
```

Question 2 (6 points)

Donnez la table des symboles du programme TC1101 suivant.
Notes:

- `LDAI` est une instruction qui "charge une valeur immédiate dans l'accumulateur" ajoutée au TC1101 lors du devoir 10.

```
    LDAI 15
    JMP  [1]
X   BYTE 7
[1] SUB  X
    JZ   [2]
    ADD  P
    JMP  [1]
[2] HLT
P   BYTE 3
```

Question 3 (10 points)

Sur cette page-ci et la suivante, vous trouverez une implémentation incomplète du simulateur TC1101, `Sim.java`, en particulier, les énoncés qui simulent l'exécution des instructions ont été remplacés par des commentaires de la forme "`/* code simulant ...*/`".

Effectuez toutes les modifications nécessaires au programme afin d'ajouter l'instruction suivante au TC1101:

`LDAS X`

Cette instruction charge la valeur se trouvant à l'adresse `X` de la mémoire dans l'accumulateur **et** met à jour les bits de statut `Z` et `N`.

Notes:

- utilisez l'opCode 99 pour cette instruction.

```
class Sim {

    public static final int MAX_ADDRESS = 9999; // les addresses ont 2 "octets"



    public static final int LDA = 91; // charge (load) une valeur de la mémoire dans l'accumulateur
    public static final int STA = 39; // sauve (store) le contenu de l'accumulateur en mémoire
    public static final int CLA =  8; // mise-à-zero (clear) de l'accumulateur
    public static final int INC = 10; // incrémente (ajoute 1) à l'accumulateur
    public static final int ADD = 99; // ajoute (add) à l'accumulateur
    public static final int SUB = 61; // soustrait de l'accumulateur
    public static final int JMP = 15; // branchement inconditionnel (jump, go to)
    public static final int JZ  = 17; // branchement si le bit de statut Zero est vrai
    public static final int JN  = 19; // branchement si le bit de statut Negatif est vrai
    public static final int DSP =  1; // affiche a l'écran (display)
    public static final int HLT = 64; // arrêt (halt)

    // la mémoire
    private static final int[] memory = new int[MAX_ADDRESS + 1];

    // les registres
    private static int pc;            // compteur de programme
    private static int a;             // accumulateur
    private static int opCode;        // l'opCode de l'instruction courante
    private static int opAddr;        // l'adresse de l'opérande de l'instruction courante
    private static boolean z;         // bit de statut "Zero"
    private static boolean n;         // bit de statut "Negatif"
    private static boolean h;         // bit de statut "Halt"

    private static int mar;           // registre adresse mémoire (Memory Address Register)
    private static int mdr;           // registre donnée mémoire (Memory Data Register)
    private static boolean rw;        // bit lecture/écriture (Read/Write)
                                      // lecture=true/écriture=false

    private static void accessMemory() {
        if (rw) {
            // rw=true signifie lecture, copie de la mémoire au processeur
            mdr = memory[mar];
        } else {
            // rw=false signifie écriture, copie du processeur à la mémoire
            memory[mar] = mdr;
        }
    }
```

```
public static void run() {
    pc = 0;       // l'exécution débute toujours à l'adresse 0
    h  = false;   // remise à faux du bit de statut Halt

    while (h == false) {
```

Indiquez à l'aide de flè
le code doit être inséré

Question 4 (20 points)

Pour cette question, présumez l'existence d'une classe nommée `List` dont les éléments sont de type double et qui contient **seulement** les méthodes suivantes, telle qu'au devoir 7 mais sans la méthode `toString()`, ni les exceptions:

```
public List() // ce constructeur crée une liste vide
public boolean isEmpty()
public double head()
public void deleteFirst()
public void insertAtFront(double newElement)
public List split()
public void join(List toBeJoined)
```

Écrivez une méthode de classe (static) **récursive**, `separate(double x, List l)`, qui retire de `l` tous les éléments dont la valeur est inférieure à `x` **et** qui retourne ces éléments dans une nouvelle liste. S'il n'y a aucune valeur inférieure à `x` alors la méthode retourne la liste vide sans changer le contenu de `l`. Étant donné les valeurs suivantes pour `l` et `x`:

```
l -> 5 -> 7 -> 3 -> 2 -> 6 -> 2
x = 5
```

suite à l'appel, `separate(x, l)`, le contenu de la liste `l` est le suivant:

```
l -> 5 -> 7 -> 6
```

et la valeur de retour est:

```
-> 3 -> 2 -> 2
```

Notes:

- l'ordre des éléments de la nouvelle liste est sans importance,
- cette méthode est définie à l'extérieur de la classe `List`,
- les éléments de la liste sont de type double,
- vous n'avez pas à traiter le cas ou la valeur de `l` est `null`.

(si nécessaire utilisez la page qui suit pour répondre)

(espace supplémentaire pour répondre à la question 4)

Question 5 (20 points)

Cette question nécessite l'utilisation:

- d'une interface nommée `Iterator` (exactement la même qu'au devoir 9),

```
public interface Iterator {
    public int      next()              throws InvalidIteratorOperation,
                                               InvalidIteratorException;
    public boolean  hasNext()           throws InvalidIteratorException;
    public Iterator copy()              throws InvalidIteratorException;
    public void     add(int newElement) throws InvalidIteratorException;
    public void     remove()            throws InvalidIteratorOperation,
                                               InvalidIteratorException;
    public boolean  hasPrevious()       throws InvalidIteratorException;
    public boolean  isValid();
    public boolean  equals(Iterator i)  throws InvalidIteratorException;
}
```

- d'une classe nommée `List`.
  Chaque élément de la liste est soit 0 ou 1, la classe `List` n'a que les 2 méthodes suivantes:

  ```
  public List() // un constructeur qui crée une liste vide
  public Iterator createIterator()
  ```

  Si `l` est de type `List` alors `l.createIterator()` crée un itérateur positionné devant le premier élément de la liste.
- d'une classe nommée `BinaryNumber` (consultez la page qui suit pour les détails). Comme pour le devoir 9, une instance de la classe `BinaryNumber` utilise une liste afin de représenter un nombre (entier) binaire sans signe de taille illimitée, telle que chaque noeud de la liste contient un bit du nombre. La liste de bits représente le nombre dans l'ordre "droite à gauche" — c'est-à-dire que le bit de poids le moins significatif (bit de droite) est placé dans le premier noeud de la liste. Par exemple, le nombre binaire 10011 sera représenté par la liste suivante:

  ```
  -> 1 -> 1 -> 0 -> 0 -> 1
  ```

  Notez que la liste vide représente la valeur 0 et que votre solution doit traiter ce cas correctement.

Pour la classe `BinaryNumber`, dont la définition se trouve sur la page qui suit, écrivez une méthode d'instance, `increment()`, telle que `b.increment()` **modifie** `b`, de sorte que sa valeur à la suite de l'appel soit un de plus.
Si `b` contient la liste de bits suivante (représentant le nombre binaire 10011):

```
-> 1 -> 1 -> 0 -> 0 -> 1
```

alors `b.increment()` modifie la liste de bits de `b` comme suit:

```
-> 0 -> 0 -> 1 -> 0 -> 1
```

(représentant le nombre binaire $10100 = 10011 + 1$)
Notes:

- la méthode ne doit pas être récursive.

(utilisez la page qui suit pour répondre)

(suite de la question 5)

```
class BinaryNumber {
  // variable d'instance
  private List bits; // liste de 0s et de 1s

  // constructeur, initialise BinaryNumber à la liste vide,
  // ce qui représente la valeur 0
  public BinaryNumber() {
    bits = new List();
  }

  // Répondez dans l'espace ci-bas.









  } // fin de la classe BinaryNumber
```

Question 6 (22 points)

Étant donné l'implémentation d'une liste chaînée donnée ci-bas, écrivez une méthode d'instance:

`removeDuplicates()`

Si `l` est une instance de la classe `List` alors `l.removeDuplicates()` modifie `l` de sorte qu'elle ne contienne qu'**une** seule occurrence de chaque valeur.
Étant donné `l`:

```
l -> 5 -> 7 -> 5 -> 3 -> 7 -> 5 -> 0 -> 1 -> 5 -> 5
```

voici 2 solutions possibles, garder la première occurence, ou garder la dernière:

```
l -> 5 -> 7 -> 3 -> 0 -> 1
l -> 3 -> 7 -> 0 -> 1 -> 5
```

Notes:

- peu importe quelle occurrence vous gardez, la première, la dernière ou tout autre, ce qui importe, c'est de n'en garder qu'une seule.

```
class List  {
    private class Node { // classe intérieure privée
        public int   value;
        public Node  next;

        public Node (int value, Node next) {
           this.value = value;
           this.next  = next;
        }
    }

    private Node first; // premier noeud
    private Node last;  // dernier noeud
    private int  modCount; // tel qu'au devoir 9

    // les autres méthodes sont ici, mais vous ne devez pas les utiliser.

    // la méthode removeDuplicates sera inserée ici.
}
```

(utilisez la page qui suit pour répondre)

Répondez à la question 6 dans l'espace ci-bas.

Question 7 (4 points)

Convertissez en hexadécimal le nombre binaire sans signe suivant:

$$(111011.10101)_{\text{binaire}} = (\qquad\qquad\qquad\qquad)_{\text{hexa}}$$

Question 8 (3 points)

Donnez le produit canonique de sommes de la fonction f définie par la table de vérité suivante, ne simplifiez pas votre réponse.

A	B	C	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Question 9 (5 points)

Pour cette question, les nombres binaires sont représentés à l'aide de 4 bits.

(a) Faites l'addition des entiers signés suivants, représentés en **complément à un** suivants:

$$\begin{array}{rcccc} & 1 & 1 & 1 & 0 \\ + & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

(b) Faites l'addition des entiers signés suivants, représentés en **complément à deux**:

$$\begin{array}{rcccc} & 1 & 1 & 1 & 0 \\ + & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

Question 10 (3 points)

6 bits sont utilisés afin de représenter les entiers signés, donnez la représentation binaire de l'entier décimal $(-24)_{10}$,

(a) en complément à 1.

(b) en complément à 2.

(c) en représentation "signe et valeur absolue".

Question 11 (5 points)

Pour une algèbre de boole, définissons l'opérateur binaire $\oplus$ comme suit,

$$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

(pour une algèbre logique, cet opérateur est connu sous le nom de "ou-exclusif").

En n'utilisant que les axiomes et théorèmes de l'algèbre de boole, dont vous trouverez les définitions à la dernière page du questionnaire, démontrez l'égalité suivante:

$$\overline{A \oplus B} = A \oplus \overline{B}$$

Annexe

Les axiomes et théorèmes de l'algèbre booléenne que vous devez utiliser pour la question 11.
Étant donné $a, b, c \in \mathcal{B}$, les opérateurs: $+, \cdot$ et $\neg$, vérifient les axiomes suivants:

Axiome 1 *fermeture:*

$$a + b \in \mathcal{B}$$

$$a \cdot b \in \mathcal{B}$$

$$\overline{a} \in \mathcal{B}$$

Axiome 2 *commutativité:*

$$a + b = b + a$$

$$a \cdot b = b \cdot a$$

Axiome 3 *associativité:*

$$(a + b) + c = a + (b + c)$$

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

Axiome 4 *distributivité:*

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

Axiome 5 *éléments neutres, il existe* $0, 1 \in \mathcal{B}$ *tels que:*

$$a + 0 = a$$

$$a \cdot 1 = a$$

Axiome 6 *complémentation, pour tout* $a \in \mathcal{B}$*, il existe un* $\overline{a}$ *tel que:*

$$a + \overline{a} = 1$$

$$a \cdot \overline{a} = 0$$

Les théorèmes suivants peuvent être démontrés à partir des axiomes:

idempotence	$a \cdot a = a$	$a + a = a$
élément absorbant	$a \cdot 0 = 0$	$a + 1 = 1$
absorption	$a \cdot (a + b) = a$	$a + a \cdot b = a$
de Morgan	$\overline{a \cdot b} = \overline{a} + \overline{b}$	$\overline{a + b} = \overline{a} \cdot \overline{b}$
involution	$\overline{(\overline{a})} = a$	

Vous trouverez aussi dans la littérature, U pour signifier 1 et ϕ pour signifier 0.