Introduction à l'informatique II (CSI 1501)
Examen Final

Professeur: Marcel Turcotte

Avril 2004, durée: 3 heures

Identification

Nom, prénom: _____

Numéro d'étudiant: _____ Signature: _____

Consignes

1. Livres fermés;
2. Sans calculatrice ou toute autre forme d'aide;
3. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle;
4. Écrivez lisiblement, votre évaluation en dépend;
5. Commentez vos raisonnements;
6. Ne retirez pas l'agrafe;

Barème

Question	Maximum	Résultat
1	3	
2	3	
3	6	
4	12	
5	20	
6	16	
7	20	
8	15	
9	5	
Total	**100**	

Question 1 (3 points)

Sachant que $\oplus$ (xor — "ou-exclusif") est défini comme suit,

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

À l'aide d'une table de vérité, démontrez que l'équivalence suivante est vraie. Les tables de vérité des opérateurs AND ($\cdot$), OR ($+$) et NOT ($\neg$) sont présentées à l'appendice A.

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

Question 2 (3 points)

Étant donné **le produit canonique de sommes** (PCS) décrivant la fonction F:

$$F = (x + \overline{y} + z) \cdot (x + \overline{y} + \overline{z}) \cdot (\overline{x} + y + \overline{z}) \cdot (\overline{x} + \overline{y} + z)$$

Complétez la table de vérité de la fonction F.

x	y	z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Question 3 (6 points)

Cette question porte sur la représentation des entiers binaires signés en **complément à deux** encodés à l'aide de **huit (8) bits**.

A. Représentez l'entier décimal signé suivant $(+23)_{10}$ en complément à deux.

Réponse: [_____]$_2$

B. Représentez l'entier décimal signé suivant $(-70)_{10}$ en complément à deux.

Réponse: [_____]$_2$

C. Effectuez l'addition suivante en complément à deux. Vous devez effecter les opérations en binaire.

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

Suggestion: vérifiez vos réponses en faisant les calculs en base 10 aussi.

Question 4 (12 points)

Sur la page suivante, vous trouverez un programme (en langage machine) pour l'architecture TC-1101, telle que définie pour la question 4 du devoir 9; i.e. ce processeur possède une instruction `ADDi`. L'appendice B résume les informations nécessaires afin de répondre à cette question.

A. Donnez le contenu des registres suivants à la **fin** de chacun des trois (3) premiers cycles de l'exécution du programme se trouvant à la page suivante.

	PC	opCode	opAddr	MAR	MDR	A	Z	N
0								
1								
2								

B. Traduisez le programme machine en langage assembleur pour le TC-1101. Créez des symboles si nécessaire, une étiquette désignant une instruction devra avoir le format suivant, [1], [2], etc. et l'étiquette d'une variable, ou constante, aura la forme X, Y, etc.

(Question 4 suite)

Voici le contenu de la mémoire (à droite, en **caractères gras**) et les adresses (à gauche).

00 00	**15**
00 01	**00**
00 02	**06**
00 03	**15**
00 04	**00**
00 05	**01**
00 06	**91**
00 07	**00**
00 08	**05**
00 09	**61**
00 10	**00**
00 11	**03**
00 12	**19**
00 13	**00**
00 14	**21**
00 15	**17**
00 16	**00**
00 17	**21**
00 18	**15**
00 19	**00**
00 20	**41**
00 21	**91**
00 22	**00**
00 23	**04**
00 24	**99**
00 25	**00**
00 26	**05**
00 27	**39**
00 28	**00**
00 29	**04**
00 30	**91**
00 31	**00**
00 32	**05**
00 33	**83**
00 34	**02**
00 35	**39**
00 36	**00**
00 37	**05**
00 38	**15**
00 39	**00**
00 40	**09**
00 41	**01**
00 42	**00**
00 43	**04**
00 44	**64**

Question 5 (20 points)

Une liste associative, aussi appelée *alist*, est un type abstrait de données servant à sauvegarder des paires clé-valeur. Une liste associative sert, par exemple, à implémenter la table des symboles utilisée par un assembleur.

```
AList table = new AList();
table.put("Quot", new Address(44));
table.put("Rem", new Address(45));
table.put("[7]", new Address(7));
Address x = (Address) table.get("Quot");
System.out.println("x = " + x);
```

affichera,

```
x = 0044
```

la classe `Address` est définie à l'appendice C.

Pour l'implémentation partielle de la classe `AList` ci-bas, utilisant des noeuds simplement chaînés, écrivez la méthode d'instance `put` selon les spécifications suivantes, `Object put(Object key, Object value)`:

- Associe la valeur spécifiée (`value`) avec la clé spécifiée (`key`) dans cette liste associative;
- Si cette liste associative contenait déjà une association pour cette clé, alors l'ancienne valeur est remplacée la valeur spécifiée;
- Retourne la valeur de l'association précédente, ou `null` si la liste ne contenait aucune association pour cette clé;
- Ni la clé, ni la valeur ne peuvent être `null`, une exception de type `IllegalArgumentException` devra être lancée si tel est le cas.

```
public class AList {

    //  implementation des noeuds de la liste
    private static class Node {

        private Object key;
        private Object value;
        private Node next;

        public Node(Object key, Object value, Node next) {
            this.key = key;
            this.value = value;
            this.next = next;
        }
    } // Fin de Node

    private Node first ; // le premier noeud de la liste
```

(Question 5 suite)

```
    // representation de la liste vide
    public AList() {
        first = null;
    }

    // Les autres methodes, telles que get, seraient placees ici, cependant vous
    // ne pouvez les utiliser.  Ecrivez votre reponse dans l'espace ci-bas.
```

```
} // Fin de AList
```

Question 6 (16 points)

Écrivez une méthode de classe (`static`) `boolean equals(Queue a, Queue b)` retournant `true` si `a` et `b` sont deux files contenant les mêmes éléments, dans le même ordre. Il vous faudra retirer et comparer les éléments un à un. Pour ce faire, vous n'avez droit qu'aux piles (vous pouvez utiliser autant de piles que nécessaire). Vous trouverez ci-bas les deux interfaces liées à cette question. Pour cette question, il existe une implémentation de l'interface `Stack` nommée `LinkedStack`. Les files `a` et `b` doivent demeurer inchangées suite à un appel à la méthode `equals`.

```
public interface Queue {

    // Retourne vrai si cette file est vide
    public abstract boolean isEmpty();

    // Retire et retourne l'element avant
    public abstract Object dequeue();

    // Ajoute un element a l'arriere
    public abstract void enqueue(Object element);
}

public interface Stack {

    // Retourne vrai si cette pile est vide
    public abstract boolean isEmpty();

    // Retourne une reference sur l'element avant, sans
    // changer le contenu de la pile
    public abstract Object peek();

    // Retire et retourne l'element du dessus
    public abstract Object pop();

    // Ajoute un element sur le dessus
    public abstract void push(Object element);
}
```

(Question 6 suite)

```
public static boolean equals(Queue a, Queue b) {
```

```
} End of equals
```

Question 7 (20 points)

Écrivez l'implémentation de l'interface `Stack` ci-bas.

```
public interface Stack {

    // Ajoute un element sur le dessus
    public abstract void push(Object element);

    // Retire et retourne l'element du dessus
    public abstract Object pop() throws java.util.EmptyStackException;

    // Retourne vrai si cette pile est vide
    public abstract boolean isEmpty();
}
```

- Votre implémentation doit utiliser un tableau de taille fixe;
- Le constructeur de cette classe a un paramètre, qui indique la taille du tableau;
- Lorsque le tableau est rempli, la méthode `push` met à l'écart l'élément du bas afin de faire place au nouvel élément à insérer, ainsi cet élément sera perdu;
- Cependant, la méthode `push` ne doit pas déplacer les éléments qui se trouvent présentement dans la pile, cette solution est trop coûteuse, il faudra plutôt utiliser un tableau circulaire, comme vu en classe et au devoir 6;
- La valeur `null` est un argument valide à la méthode `push`.

(Question 7 suite)

(Question 7 suite)

Question 8 (15 points)

Pour la classe **BitList** ci-bas, écrivez une méthode d'instance **récursive** retournant une nouvelle liste qui est le complément de celle-ci (le complément de 0 est 1, et vice-versa). L'implémentation doit suivre le modèle vu en classe, tel que toute méthode récursive est composée d'une partie publique et d'une partie privée. La méthode publique initie le premier appel récursif. Si la liste est vide alors la méthode doit retourner une nouvelle liste vide. Si `a` désigne la liste suivante,

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

alors `a.complement()` retournera la (nouvelle) liste qui suit,

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

```
public class BitList {
    public static final int ZERO = 0;
    public static final int ONE = 1;
    // implementation des noeuds de la liste
    private static class Node {
        private int bit;
        private Node next;
        private Node(int bit, Node next) {
            this.bit = bit;
            this.next  = next;
        }
    }
    // variables d'instance
    private Node first;
    private Node last;
    // constructeur
    public BitList() {
        first = null;
    }
    public void addFirst(int bit) {
        first = new Node(bit, first);
        if (last == null)
            last = first;
    }
    public void addLast(int bit) {
        Node newNode = new Node(bit, null);
        if (first == null) {
            first = newNode;
            last = first;
        } else {
            last.next = newNode;
            last = last.next;
        }
    }
```

(Question 8 suite)

```
} // Fin de BitList
```

Question 9 (5 points)

De nos jours, on trouve des microprocesseurs à peu près partout. `UltimateGrill` est un nouveau produit qui sera mis sur le marché l'an prochain; disponible chez Mona Dépot. L'emphase est mise sur la sécurité. Ainsi, ce BBQ est doté de plusieurs senseurs et microprocesseurs. Java a été choisi comme langage d'implémentation parce qu'il possède des bons mécanismes pour traiter les situations d'erreur. Vous devez:

A. Implémentez la classe **BurnerException**. Elle doit être une sous-classe immédiate de la classe `Exception`. Cette classe n'a qu'un constructeur. Ce constructeur a un paramètre de type `String`, un message qui doit être passé au constructeur de la super-classe lors de la création d'une instance;

B. Pour la classe `UltimateGrill` ci-bas, effectuez tous les changements nécessaires à la méthode `start` afin que les erreurs soient traitées de la façon suivante. Le grille peut fonctionner si au moins 3 des 4 brûleurs sont fonctionnels, sinon le grille s'éteint automatiquement.

```
public class UltimateGrill {

    public static final int NB_BURNERS = 4;

    private Burner[] elems;

    UltimateGrill() {
        elems = new Burner[NB_BURNERS];
        for (int i=0; i<NB_BURNERS; i++) {
            elems[i] = new Burner();
        }
    }

    public void start() {
        for (int i=0; i<NB_BURNERS; i++) {
            elems[i].ignite();
        }
    }

    public void stop() {
        for (int i=0; i<NB_BURNERS; i++) {
            elems[i].turnOff();
        }
    }
}
```

Faites l'hypothèse que la classe `Burner` existe et possède les méthodes suivantes.

- `ignite()`: allume le brûleur ou lance une exception de type `BurnerException` si une erreur se produit;
- `turnOff()`: éteint le brûleur.

(Question 9 suite)

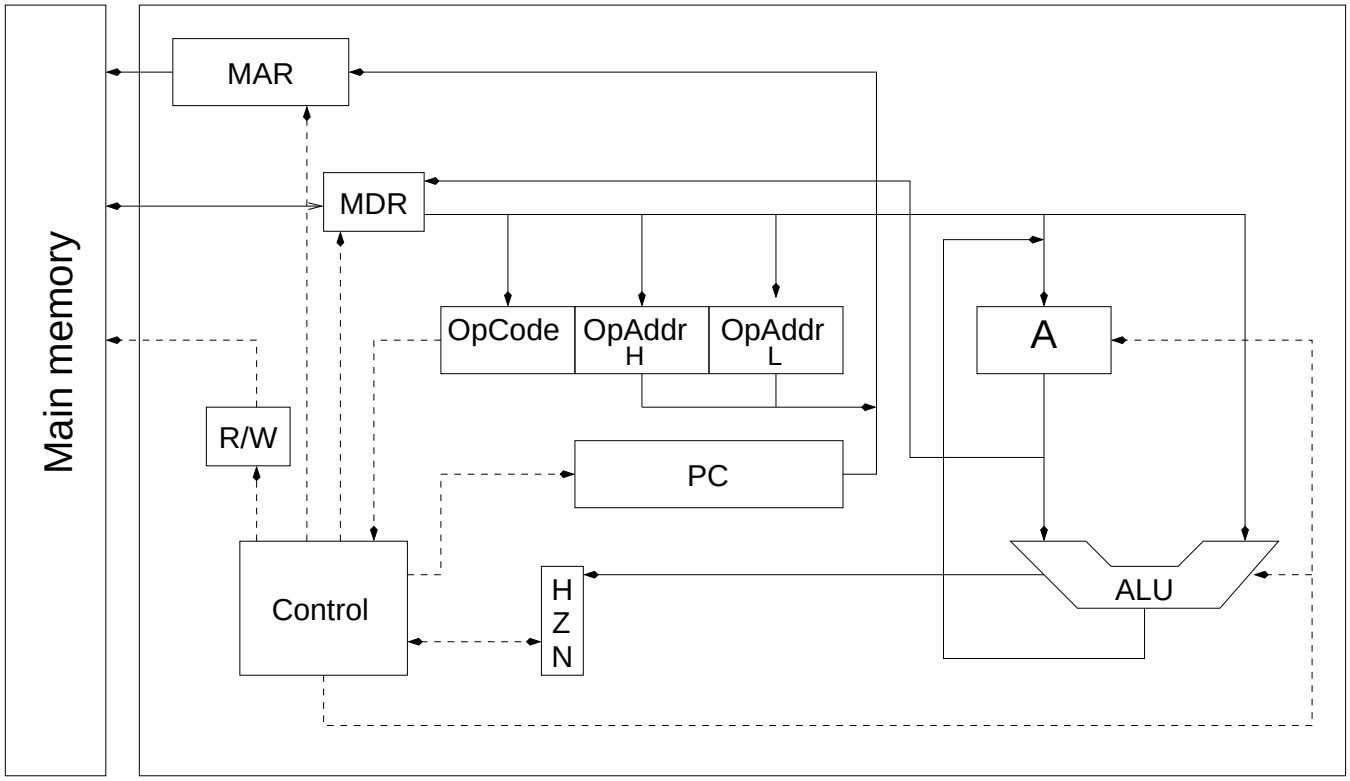
Appendice A: Algèbre logique

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A
0	1
1	0

Appendice B: architecture et assembleur du TC-1101



Mnémonique	**opCode**	Description
LDA	**91**	charge x dans l'accumulateur
STA	**39**	sauvegarde de l'accumulateur à l'adresse x
CLA	**08**	remise-à-zéro (a=0, z=vrai, n=faux)
INC	**10**	incrémente l'accumulateur (modifie z et n)
ADD	**99**	ajoute x à l'accumulateur (modifie z et n)
ADDi	**83**	ajoute une valeur à l'accumulateur (modifie z et n)
SUB	**61**	retranche x de l'accumulateur (modifie z et n)
JMP	**15**	branchement inconditionnel vers x
JZ	**17**	branchement sur x si z==vrai
JN	**19**	branchement sur x si n==vrai
DSP	**01**	affiche la valeur se trouvant à l'adresse x
HLT	**64**	fin

ou x est une adresse mémoire.

Appendice C: classe(s) auxilliaire(s)

Voici une implémentation possible de la classe `Address`.

```
public class Address {

    // addresses are 2 "bytes"
    public static final int  MAX_ADDRESS = 9999;
    private int address;

    public Address(int address) {
        if ((address < 0) || (address > MAX_ADDRESS))
            throw new IllegalArgumentException(Integer.toString(address));
        this.address = address;
    }
    public int getValue() {
        return address;
    }
    public String toString() {
        String sAddr = Integer.toString(address);
        while (sAddr.length() < 4) {
            sAddr = "0" + sAddr;
        }
        return sAddr;
    }
}
```

(page blanche)