Introduction to Computer Science II (CSI 1101 A,B)
Final Examination

Instructor: Marcel Turcotte

April 2004, duration: 3 hours

Identification

Student name: _____

Student number: _____ Signature: _____

Instructions

1. This is a closed book examination;
2. No calculators or other aids are permitted;
3. Write comments and assumptions to get partial marks;
4. Beware, poor hand writing can affect grades;
5. Do not remove the staple holding the examination pages together;
6. Write your answers in the space provided. Use the back of the pages if necessary. You may **not** hand in additional pages;

Marking scheme

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

Question 1 (3 marks)

Knowing that $\oplus$ (called xor — "exclusive-or") is defined as follows,

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

With a truth table, show that the following equivalence is true. The Appendix A has the truth tables for the AND ($\cdot$), OR ($+$) and NOT ($\neg$).

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

Question 2 (3 marks)

Given the following **canonical product of sums** (CPOS) describing F:

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

Complete the following truth table for the boolean function 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 marks)

For this question, **eight (8) bits** are used to represent **signed** binary integers using **two's complement**.

A. Represent the signed decimal integer $(+23)_{10}$ in two's complement.

Answer: [_____]$_2$

B. Represent the signed decimal integer $(-70)_{10}$ in two's complement.

Answer: [_____]$_2$

C. Do the following addition in two's complement. The calculations must be carried out in base 2.

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

Hint: verify your answers by doing the calculations in decimal as well.

Question 4 (12 marks)

On the next page you will find a program (in machine code) for the TC-1101 computer, as defined for the question 4 of the assignment 9; i.e. the processor has an `ADDi` instruction. The Appendix B summarizes the necessary information for this question.

A. Give the content of the following registers at the **end** of each of the first three (3) cycles of the execution of the program found on the next page; the cycles refer to the FETCH-EXECUTE cycles.

	PC	opCode	opAddr	MAR	MDR	A	Z	N
0								
1								
2								

B. Convert the machine program to assembly language for the TC-1101. Create new symbols as needed, labels for the instructions must have the form [1], [2], etc. and the labels that represent either variables, or constants, must have the form, X, Y, etc.

(Question 4 continued)

The table below gives the memory content (right, **bold**) and the addresses (left).

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 marks)

An association list, *alist* for short, is an *abstract data type* to store key-value pairs. The operation for finding the value associated with a key is called a *lookup*. An association list can be used to implement a "symbol table", in an assembler program, for example.

```
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);
```

would display,

```
x = 0044
```

the class `Address` is defined in the Appendix C.

For the partial implementation of the class `AList` below, using linked nodes, write the instance method **put** according to the following specifications, `Object put(Object key, Object value)`:

- Associates the specified `value` with the specified `key` in this association list;
- If the association list previously contained an association for this `key`, the old `value` is replaced by the specified `value`;
- Returns the previous `value` associated with the specified `key`, or `null` if the `AList` contains no association for this `key`;
- Neither the `key` nor the `value` can be `null`, an exception of type `IllegalArgumentException` must be thrown if this is the case.

```
public class AList {

    //  private nested class
    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;
        }
    } // end of Node

    private Node first ; // first node
```

(Question 5 continued)

```
    // Representation of the empty list
    public AList() {
        first = null;
    }

    // The other methods, such as get, would go here, but no auxiliary method can
    // be used for your implementation. Write your answer in the space below.
```

```
} // End of AList
```

Question 6 (16 marks)

Write the static method `boolean equals(Queue a, Queue b)` that returns `true` if `a` and `b` are two queues containing the same elements in the same order. This will require taking elements out of the queues and comparing them one by one. The only auxiliary data structure that you are allowed to use to temporarily store these elements is a stack (you can use as many stacks as you need). Here are the two interfaces for this question. You may assume the existence of an implementation for `Stack`, say `LinkedStack`. The queues `a` and `b` must remain unchanged following a call to the method `equals`.

```
public interface Queue {

    // Tests if this Queue is empty
    public abstract boolean isEmpty();

    // Removes and returns the front element of the Queue
    public abstract Object dequeue();

    // Puts an element at the rear of this Queue
    public abstract void enqueue(Object element);
}

public interface Stack {

    // Tests if this Stack is empty
    public abstract boolean isEmpty();

    // Returns a reference to the top element; does not change
    // the state of this Stack
    public abstract Object peek();

    // Removes and returns the element at the top of this stack
    public abstract Object pop();

    // Puts an element onto the top of this stack
    public abstract void push(Object element);
}
```

(Question 6 continued)

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

```
} End of equals
```

Question 7 (20 marks)

Write an implementation for the interface `Stack` below.

```
public interface Stack {

    // Puts an element onto the top of this stack
    public abstract void push(Object element);

    // Removes and returns the element at the top of this stack
    public abstract Object pop() throws java.util.EmptyStackException;

    // Tests if this Stack is empty
    public abstract boolean isEmpty();
}
```

- Your implementation must use a fixed-size array;
- The constructor of the class has a parameter, which is the size of the array;
- Whenever the stack is full the method `push` should discard the bottom element to make room for the new element to be inserted, consequently elements will be lost;
- However, the method `push` should not move the elements that are currently stored in the stack, this solution is considered to costly, instead, the implementation should use a circular array, as seen in class and in assignment 6;
- `null` values are valid elements.

(Question 7 continued)

(Question 7 continued)

Question 8 (15 marks)

For the class **BitList** below write a **recursive** instance method that returns a new **BitList** that is the complement of this list (the complement of 0 is 1, and vice-versa). The implementation must follow the scheme presented in class, where a recursive method is made of a public part and a private recursive part, that we called the helper method. The public method is responsible for initiating the first call to the recursive method. If the list is empty the method must return a new empty list. Assuming that `a` designates an instance that contains the following list of bits,

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

`a.complement()` returns a new `BitList` as follows,

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

```
public class BitList {
    public static final int ZERO = 0;
    public static final int ONE = 1;
    // the implementation of the Nodes
    private static class Node {
        private int bit;
        private Node next;
        private Node(int bit, Node next) {
            this.bit = bit;
            this.next  = next;
        }
    }
    // instance variables
    private Node first;
    private Node last;
    // constructor
    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 continued)

```
} // End of BitList
```

Question 9 (5 marks)

Nowadays, microprocessors are found everywhere. The `UltimateGrill` is a new product that will be launched next year; available at Mona Dépot. The emphasis is placed on security. Accordingly, the BBQ is equipped with sensors and microprocessors. Java has been chosen as the implementation language because of its ability to handle exceptions. For this question you must:

A. Implement the exception class **BurnerException** as a direct descendant of the class **Exception**. This class has a single constructor that has a formal parameter of type `String`, which is the message to be passed to the constructor of the superclass upon creation;

B. In the class `UltimateGrill`, make all the necessary changes so that the method `start` handles error situations as follows; the grill is functional if at least 3 out of its 4 burners are functional, otherwise the grill is automatically turned off.

```
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();
        }
    }
}
```

Assume the existence of a class called `Burner` that has the following methods.

- `ignite()`: ignites the element or throws an exception of type `BurnerException` if an error occurs;
- `turnOff()`: turns off the burner.

(Question 9 continued)

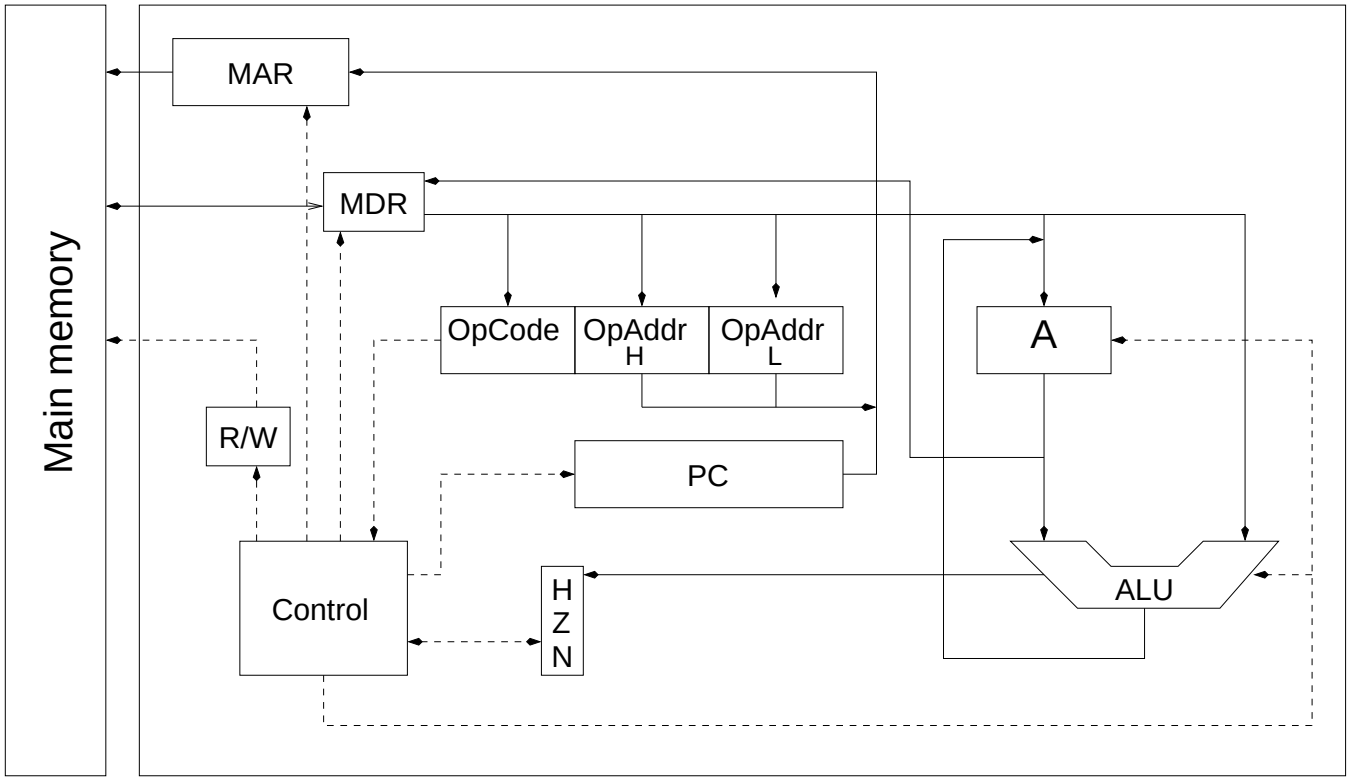
Appendix A: Switching Logic

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

Appendix B: TC-1101 architecture and assembly language



Mnemonic	opCode	Description
LDA	**91**	load x
STA	**39**	store x
CLA	**08**	clear the accumulator (a=0, z=true, n=false)
INC	**10**	increment accumulator (modifies z and n)
ADD	**99**	add x to the accumulator (modifies z and n)
ADDi	**83**	add a value to the accumulator (modifies z and n)
SUB	**61**	subtract x from the accumulator (modifies z and n)
JMP	**15**	unconditional branch to x
JZ	**17**	go to x if z==true
JN	**19**	go to x if n==true
DSP	**01**	display the content of the memory location x
HLT	**64**	halt
		where x is a memory location.

Appendix C: auxiliary classes

Here is a possible implementation for the class `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;
    }
}
```

(blank space)