Name: Matriculation Number:

Midterm Exam
General CS II (320201)

April 5, 2011

You have 75 minutes(sharp) for the test;
Write the solutions to the sheet.

The estimated time for solving this exam is 8 minutes, leaving you 67 minutes for revising your exam.

You can reach 4 points if you solve all problems. You will only need 31 points for a perfect score, i.e. -27 points are bonus points.

Different problems test different skills and knowledge, so do not get stuck on one problem.

	To be used for grading, do not write here		
prob.	1.1	Sum	grade
total	4	4	
reached			

Please consider the following rules; otherwise you may lose points:

- Always justify your statements. Unless you are explicitly allowed to, do not just answer "yes" or "no", but instead prove your statement or refer to an appropriate definition or theorem from the lecture.
- If you write program code, give comments!

1 Graphs

Problem 1.1 (Bipartite Graphs) 8pt

Let $G = \langle V, E \rangle$ be an undirected graph. Assume that we can split V to X and Y where $X \cap Y = \emptyset$ and every edge in E connects a node from X with a node from Y. Remember that we call such graphs *bipartite* graphs. Prove that if G is a bipartite graph with partite sets X and Y, then $\sum\limits_{v \in X} deg(v) = \sum\limits_{v \in Y} deg(v)$.

Solution: This can be proved via induction on the number of edges. Denote the number of elements of X as $\#(X)$. Suppose $\#(X) = r$ and $\#(Y) = s$ for some integers $r, s > 1$. Note that the case where X and Y have one vertex is trivial, as only one edge can be drawn. Take the subgraph of G which consists of only the vertices of G. Now we begin the induction: add one edge from any vertex in X to any vertex in Y. Then $\sum\limits_{v \in X} deg(v) = \sum\limits_{v \in Y} deg(v) = 1$. Now suppose this is true for $n - 1$ edges and add one more edge. Since this edge adds exactly 1 to both $\sum\limits_{v \in X} deg(v)$ and $\sum\limits_{v \in Y} deg(v)$, we have that this is true for all $n \in \mathbb{N}$.

---

2 Positional Number Systems and Circuits

Problem 2.1 (Positional Number Systems) 6ptin

Consider the 16-bit Two's Complement Number System.

1. Recall the definitions of overflow/underflow given in class. Give two examples when underflow and overflow occur and explain your choice.

2. Convert the numbers 31 and -17 in the above mentioned system.

Solution:

1. By the Structure Theorem in TCN, we know that an overflow/underflow occurs when the last two carry bits are different. Thus, any example that follows this rule is correct; take for instance, for underflow, $\langle\langle B(z_1)\rangle\rangle_{16}^{2s} = 1001010101011000$ and $\langle\langle B(z_2)\rangle\rangle_{16}^{2s} = 1101100101100010$. For overflow, one can consider the examples :

   $\langle\langle B(z_1)\rangle\rangle_{16}^{2s} = 0101010001011110$ and $\langle\langle B(z_2)\rangle\rangle_{16}^{2s} = 01101100100001010$

2. The solution of the problem is straightforward, if students know the Sign-Duplicate Theorem:

   $\langle\langle B(31)\rangle\rangle_{16}^{2s} = 0000000000011111$

   $\langle\langle B(-17)\rangle\rangle_{16}^{2s} = 1111111111101111$

Problem 2.2 (White Box Circuit) 4pt 8min

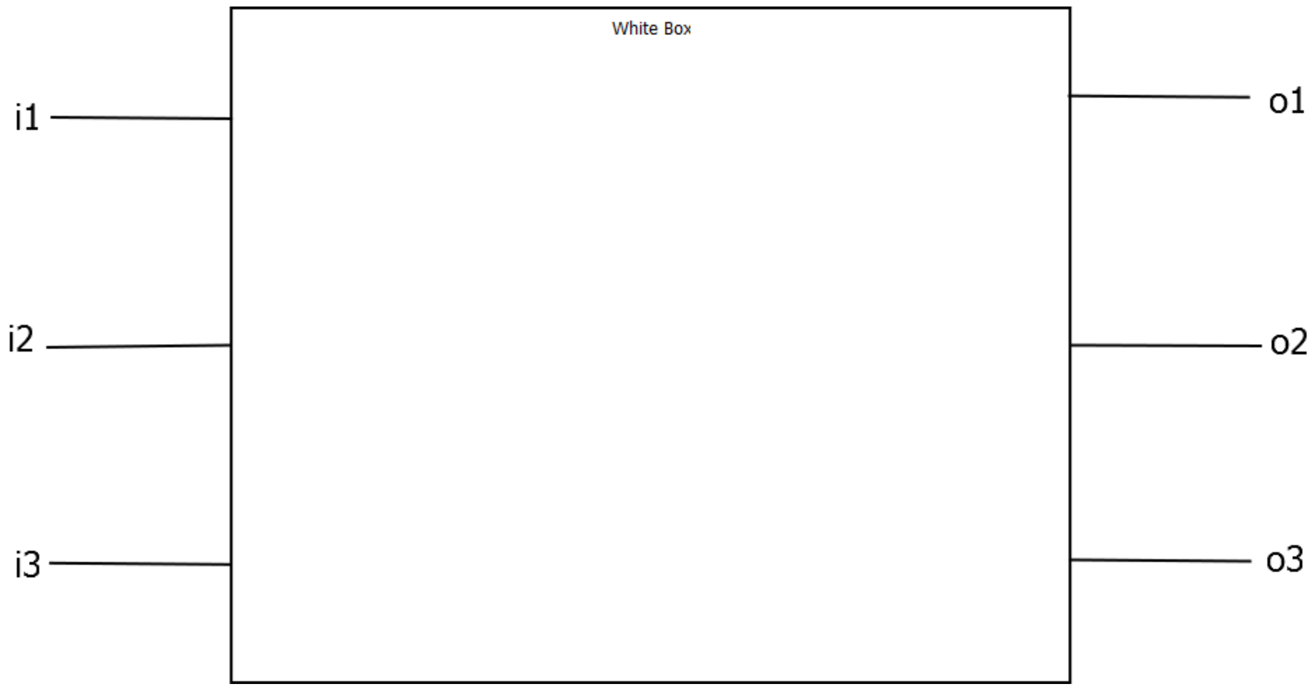
You are given a circuit element that has three inputs i_1, i_2 and i_3 and three outputs o_1, o_2 and o_3 known as a *white-box*. Such circuit elements also come with a specification table that describes what the outputs are for all possible inputs:

i_1	i_2	i_3	o_1	o_2	o_3
0	0	0	1	0	1
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	0

Your task is to find out the minimum number of circuit gates (AND, OR, NOT) this *white-box* contains and how they are interconnected by drawing the combinational circuit in the *white-box* below:

Hint:

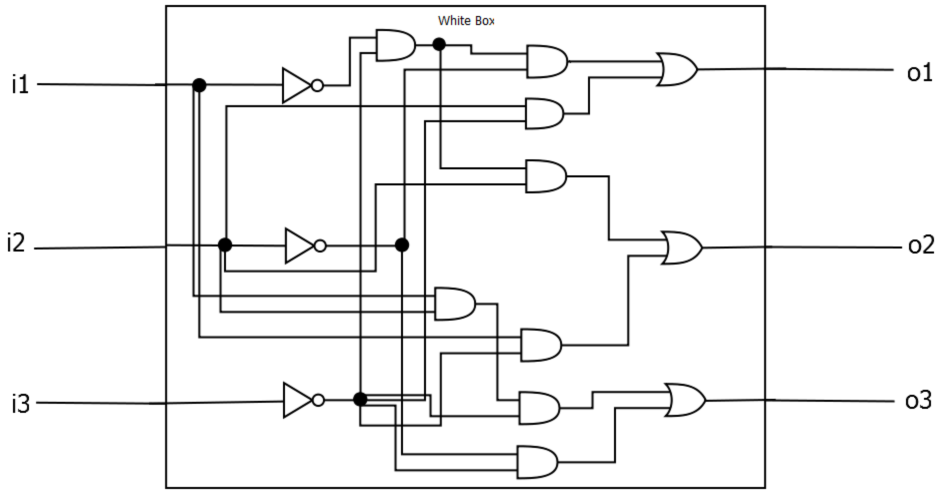
- Build the DNF for each of the outputs, minimizing the boolean polynomial through boolean equivalences
- Look for shared monomials and build the combinational circuit



Solution: The DNF for each output gives the following:

$$\begin{aligned} o_1 &= i_2 \wedge \neg i_3 \vee (\neg i_1 \wedge \neg i_2) \wedge \neg i_3 \\ o_2 &= i_1 \wedge \neg i_3 \vee (\neg i_1 \wedge i_2) \wedge \neg i_3 \\ o_3 &= \neg i_2 \wedge \neg i_3 \vee (i_1 \wedge i_2) \wedge \neg i_3 \end{aligned}$$

Hence the circuit for these functions can have the following representation:



Problem 2.3 (Reversible Counter) 7pt 14min

Build a circuit which simulates a 2-bit reversible binary counter. Your circuit will have 2 inputs and 2 outputs. Here are the input details:

- If the 1st input is 1, your counter will behave like a normal counter, outputting 00, 01, 10, 11 and looping back. However, if the input is 0, your counter will count backwards, i.e. 11, 10, 01, 00 and so on. Note that this value can change while testing your reversible binary counter.
- The 2nd input will be a standard clock signal of a constant frequency. This will be used to set the frequency of your counter.

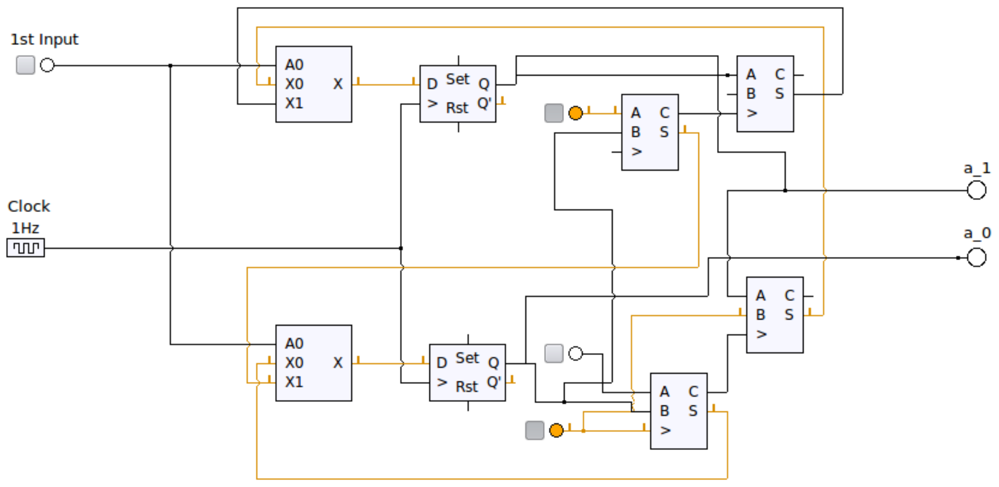
The output will be the standard 2 bits which represent the current value of the counter. You may use any logical setup which was taught in the slides, as long as you explain its input and output.

Example: If the 1st input is 1 for the first two runs of the clock, and 0 after wards, your output should be 01, 10, 01, 00, 11, 10, etc.

Hint: Use adders and subtracters to increment or decrement the counter.

Solution: First of all, we need two data flip-flops to store the current value of the counter. As we want our counter in sync with the input clock, we connect the clock to the enable inputs of our data flip-flops. Now, what we want is to either increment this counter or decrement the counter. To achieve this, we use an adder and a subtracter to compute the incremented, as well as the decremented value and then rely on a multiplexer to chose which result we want to feed back in the data flip-flops (depending on the 1st input).

Here is a diagram with the full circuit.



---

3 Machines

Problem 3.1 (Hofstadter Sequences) 10pt 5min

A Hofstadter sequence is an integer sequence defined by non-linear recurrence relations. The Hofstadter Male (M) and Female (F) sequences are defined as follows:

- $F(0) = 1$
- $M(0) = 0$
- $F(n) = n - M(F(n-1))$, for $n > 0$
- $M(n) = n - F(M(n-1))$, for $n > 0$

Create two static VM procedures for the computation of the Male and Female sequences. Each procedure should take an argument n, and return the n^{th} number from the respective Hofstadter sequence.
Then implement another procedure that calculates the sum of the first n elements of the following expression:

$$F(1) + M(2) + F(3) + M(4) + F(5) + ...$$

You may create auxiliary procedures that aid the computations if you wish.

Solution:

```
proc 1 26 // F(n)
con 0 arg 1 leq cjp 5
con 1 return
con 1 arg 1 sub
call 0          // F(n-1)
call 26         // M(F(n-1))
arg 1 sub       // n-M(F(n-1))
return


proc 1 26 // M(n)
con 0 arg 1 leq cjp 5
con 0 return
con 1 arg 1 sub
call 26         // M(n-1)
call 0          // F(M(n-1))
arg 1 sub       // n-F(M(n-1))
return
```

```
    proc 1 21 // auxiliary procedure: $n\%2$
con 1 arg 1 leq cjp 5
arg 1 return
con 2 arg 1 sub call 52
return


proc 1 40 // sum
con 1 arg 1 leq cjp 7
arg 1 call 0 return     // $F(1)$
arg 1 call 52 cjp 8     // checking $n\%2$
arg 1 call 0 jp 6       // if $n\%2 = 1$: $F(n)$
arg 1 call 26           // if $n\%2 = 0$: $M(n)$
con 1 arg 1 sub call 73 // $sum(n-1)$
add
return
```

Problem 3.2 (Zebra TM) 4pt

Design a Turing Machine that accepts an input of the form $w \in \{0,1\}^*$. You need to test whether the tape is an alternating sequence of 1's and 0's. Possible correct starting tapes include: 01010, 1010, 101. Should this not be the case, you need to output a $*$ character at the right of your tape. You should not use more than 6 states including the halting state, each extra state leads to a 0.5 point deduction.

Solution: The trivial solution is to basically traverse the tape once from left to right, zig-zag-ing between tapes if the input is correct. At the first error, one moves to another state (4 in the sample solution) which simply moves rightwards through the tape without changing anything except writing a $*$ when it reaches the end of the tape.

```
1, 1 2, 1, >
1, 0 3, 0, >
1, _ H, _
2, 0 1, 0, >
2, 1 4, 1, >
3, 1 5, 1, >
3, 0 4, 0, >
4, 0 4, 0, >
4, 1 4, 1, >
4, _ H, *
5, 0 1, 0, >
5, 1 4, 1, >
5, _ H, _
```

4 Search

Problem 4.1 (Passing a Bridge) 8pt 15min

There exists in Bremen an **one-way** bridge over the Weser. You are the administrator of this bridge and today you received a list of N bridge pass requests in form of pairs (t_i, d_i) (where t_i is the time of arrival and d_i is the direction, for example 1 or 2). You know that the bridge can be passed by any car in a time T and, as you wish to get home early today, you want to schedule the car-passes in such a way that the last car finishes passing at a minimum time.

For example:

```
N=3, T=5
Requests: (1,1), (4,1), (3,2)
```

That means that you have to manage two cars passing in direction 1, arriving at the bridge at times 1 and 4, and one car passing in direction 2 arriving at time 3.

Your tasks are the following:

- Give a formal description of the problem. Give an example of a state respecting your description. How do you define the transitions between states?
- Draw the search tree for the given example in such a way that it contains at least one goal state. **The solution does not have to be optimum / on the optimum path.**
- Pick **one** search strategy and clearly state **why it provides / why it does not provide an optimum solution**.

Note: Needless to say, the bridge can be passed at the same time by multiple cars in the same direction, but no two cars can pass at the same time in opposite directions.

Solution:

- The states $\mathcal{S}$ can be represented as $\langle (l_0 \ldots l_k), \tau \rangle$, where $(l_0 \ldots l_k)$ is a list of cars in the order they pass the brdige, and τ is the total pass time. For example (considering the problem example), we have states $\langle (0), 6 \rangle$ and $\langle (0,2), 11 \rangle$. The transition is made by appending a car index at the end of the list and updating the total pass time as follows:

$$\tau^{new} = \begin{cases} max(\tau^{old}, t_{l_{k+1}}) + T & \text{if } d_{l_{k+1}} \neq d_{l_k} \\ max(\tau^{old}, t_{l_{k+1}} + T) & \text{if } d_{l_{k+1}} = d_{l_k} \end{cases}$$

  The cost of the transition is $\tau^{new} - \tau^{old}$. There are invalid states, for example the states that contain duplicate cars. The goal state is a valid state that contains all the cars in the list. The optimum goal state is the one with a minimum τ.

- will be soon
- BFS will not provide optimum solution because it ignores the cost of transitions and will supply a non-optimum goal state. On the other hand, UCS will provide an optimum solution.