Name: Matriculation Number:

Midterm Exam
General CS II (320201)

March 20, 2012

You have 75 minutes(sharp) for the test;
Write the solutions to the sheet.
The estimated time for solving this exam is 69 minutes, leaving you 6 minutes for revising your exam.
You can reach 109 points if you solve all problems. You will only need 100 points for a perfect score, i.e. 9 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	2.1	2.2	2.3	3.1	3.2	3.3	4.1	Sum	grade
total	15	10	10	10	15	17	17	15	109	
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, so that we can award you partial credits!
- You may use tags in your $\mathcal{L}(\texttt{VM})$ program to save some (counting) time. Use `<string>` for tags, where `string` is a string of lower-case english letters and place the tag before the instruction one would want to jump to when calling `jp` or `cjp`. For a jump place the tag after `jp` and `cjp` and omit writing the relative jump distance.
- Write your program clearly. Should you wish, you may write additional code in a higher level language (HLL) as a comment to help the grader understand what you are trying to do. HLL code without $\mathcal{L}(\texttt{VM})$ code will not give you points.

1 Graphs and Trees

Problem 1.1 (Graph Puzzles) 85 min

1. A complete graph is a graph for which every vertex is connected to all other vertices. Find the number of edges in such a graph, and the sum of the degrees of all vertices.
2. A balanced binary tree has 70 leaves. Find the height of the tree.

Solution:

1. Since every vertex is connected to all other vertices, we get that every vertex has degree $n-1$. Thus the total sum of degrees is $n \cdot (n-1)$. We know that the sum of all degrees equals $2*$ number of edges. Then the number of edges is $\frac{n \cdot (n-1)}{2}$.
2. Since we have a binary tree, by Lemma 400 (vertices of binary trees) the tree has $2 \cdot 70 - 1 = 139$ vertices. Then since the tree is balanced, by Lemma 399 (depth of a balanced binary tree) the depth of the tree is $\log_2(139) = 7$.

---

2 Combinatorial Circuits

Problem 2.1 (Binary Conversions) 10 pts

1. Convert the following decimal numbers into TCN and signed binary number representations:
   - 14
   - -24
2. How would you multiply an **unsigned** binary number by 2^{10} (10^2)?
3. What would be the effect of applying the above procedure directly to TCN and sign-bit binary numbers? Demonstrate using the example numbers from part 1.

Solution:

Problem 2.2 (Explaining Circuits) 5 10 min

Briefly explain what is the functionality/ application of the following circuit elements:

- multiplexer
- address decoder

Solution: Functionalities of the circuit elements:

1. A multiplexer is a circuit with n (in GenCS 2) input lines, a control input, and exactly one output line. The MUX^n is used to select exactly one of the input lines depending on the control.
2. An address decoder is used, as the name suggests, to resolve addresses. It has n input lines, where n corresponds to the number of bits in the address. There are 2^n output lines, one of which is selected, indicating the correct address.

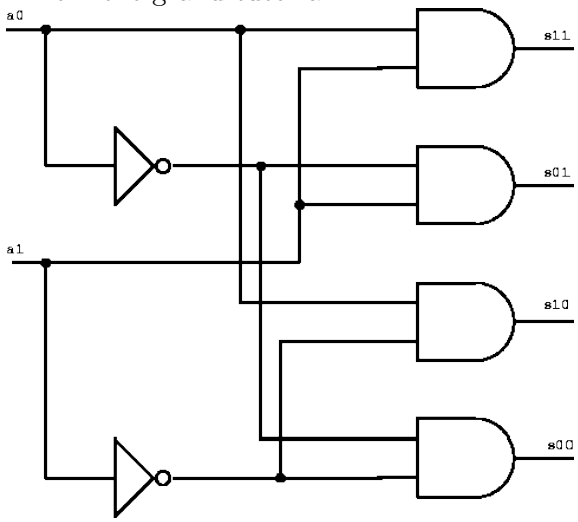
Problem 2.3 (Circuit Theory and Functional Completeness) 50pts

A functionally complete logic is a set of gates that can be used to realize all logic functions. You have seen examples of this in the slides: NAND gates are functionally complete, as well as NOR gates.

- Draw the circuit of a 2:4 address decoder and write the boolean equations describing its output lines.
- Is the set of gates consisting of inverters (NOT) and 2:4 address decoders functionally complete? Explain your reasoning.

Solution: Functionally complete logics:

- This is just the problem from the grand tutorial:



  $s00 = \overline{a0} * \overline{a1}$
  $s10 = a0 * \overline{a1}$
  $s01 = \overline{a0} * a1$
  $s11 = a0 * a1$
- The 2:4 address decoder already provides a NOR gate, i.e. $s00$. From this we can get all other gates: either only from the NOR gates, or together with the inverters, we can get OR gates; then from this, a simple application of De Morgan's laws will give us an AND gate. Therefore this set of gates is functionally complete.

---

3 Machine Programming

Problem 3.1 (Count the Contestants) 12pt 15min

You have just entered a new contest and got a decent score. However, you are afraid that you might not qualify to the next stage, so you quickly write an ASM program to help you figure how many contestants will advance.

You are given all the N scores (possibly negative), in a non-increasing order; you also know that only contestants that have:

1. a positive score
2. the score bigger than the score of contestant on place k

will advance.

You are given k in $D(1)$ and $N > 3$ values representing the scores of the participants in $D(2 \dots N+1)$ (N is not given, but you may rest assured that $N > K$). You must output the requested value in $D(0)$.

Example: $N = 8$, $K = 5$ and the scores $(10, 9, 8, 7, 7, 7, 5, 5)$. Your program should stop with $D(0) = 6$.

Hint: You are allowed to overwrite any position in memory, if needed!

Note: Write down your idea first! It will be more valuable than the code itself.

Solution: The following C snippet might clear up the solution:

```
int nr = 0; /* the number of contestants that advance, in D(0) */
int dk = D[k+1]; /* store this value in D(1) since we do not really
                    need k */
while ( D[nr+2] >= dk && D[nr+2] > 0 ) /* we iterate using nr */
     nr ++;
return nr;
```

Since there is no logical AND operation in ASM, we will use nested if constructs (plus comparisons with 0), such as:

```
if ( D[nr+2] - dk >= 0 )
  if ( D[nr+2] > 0 )
    /* ... */
```

The ASM code follows:

```
LOAD 1
MOVE ACC IN2
LOADI 0
STORE 0
LOAD 0
MOVE ACC IN1
LOADIN 2 1
STORE 1
LOADIN 1 2
SUB 1
JUMP(<) 7
LOADIN 1 2
```

```
JUMP(<=) 5
LOAD 0
ADDI 1
STORE 0
JUMP -12
STOP 0
```

Problem 3.2 (Divider in $\mathcal{L}$(VM)) 17pt

You are given a natural number $n > 1$. Write an $\mathcal{L}$(VM) program which finds the largest natural number x for which the following properties hold (simultaneously):

1. $x < n$
2. $n \bmod x = 0$

Since you are given n, assume your program can begin with "$VMconn$". Though this is not a valid $\mathcal{L}$(VM) set of instructions, we assume the user replaces the argument "n" by the decimal representation of the value of n before running the program through the virtual machine.

Assuming the stack was initially empty (before pushing the value of n), make sure that after the execution the stack only contains the result. This is a healthy precondition and we want you to learn good practice. Though it is mandatory, try not to stress this too much, and focus on the algorithm itself. You can deal with cleaning up the stack once you get to the result.

Solution: For ease of understanding, we write the solution in a higher level language:

```
var n = $n$
var a = n-1
var b = 1
<while> while ( b * a + 1 <= n ) {
  b = b+1
}
if (b*a <= n)
  return a
else
  a = a-1
  b = 1
  goto <while>  /* see what I did here? */
```

Now, we write the $\mathcal{L}$(VM) code (looks nice when put in parallel with the previous code):

```
con $n
con 1 peak 0 sub
con 1
<while> peak 0 con 1 peak 1 peak 2 mul add leq cjp <if>
  con 1 peak 2 add poke 2
jp <while>
<if> peak 0 peak 1 peak 2 mul leq cjp <else>
  poke 0  poke 0 halt
<else>
  con 1 peak 1 sub poke 1
  con 1 poke 2
  jp <while>
```

Problem 3.3 (Square root and $\mathcal{L}$(VMP)) 17pt

Design a $\mathcal{L}$(VMP) procedure which takes as argument a natural number n and returns $\lfloor\sqrt{n}\rfloor$.

Hint: Suppose you have the following C code for $\lfloor\sqrt{n}\rfloor$:

```
int square_root(int n)
{
    int i = 0;
    while (i * i <= n)
        i++;
    return i - 1;
}
```

For help with writing the $\mathcal{L}$(VMP) procedure, try to convert this iterative C function into a recursive one, which doesn't make use of local variables (remember you cannot use `peek` and `poke` in static procedures).

Solution: Write

```
int square_root_help(int n, int i)
{
    if (n == 0)
        return 0;
    else
        if (i * i <= n)
            return square_root_help(n, i + 1);
        else
            return i - 1;
}
```

The square root is now

```
square_root(n) = square_root_help(n, 0).
```

Write the procedures for these two functions,

```
proc 2 34
arg 1
cjp <nil>
arg 1
arg 2
arg 2
mul
leq
cjp <ret>
con 1
arg 2
add
arg 1
call 0
return
```

```
<ret> con 1
arg 2
sub
return
<nil> con 0
return
```

Then, square root is

```
proc 1 7
con 0
arg 1
call 0
return
```

---

4 Turing Machines

Problem 4.1 (Turing Machine) 10 min

Design a Turing machine, which on input consisting of an arbitrary sequence of ones (1) and zeros (0), surrounded by hashes (#) does the following:

- **Step 1:** XOR each input cell with the one right of it and output the result in the same cell;
- **Step 2:** delete the last input cell by replacing it with a #;
- **Step 3:** repeat steps 1 and 2 until the whole input is deleted, then halt.

Assume the head is position at the first input symbol. Provide a transition (action) table.

Sample run:
#1101001# $\mapsto$ #011101## $\mapsto$ #10011### $\mapsto$ #1010#### $\mapsto$ #111##### $\mapsto$ #00###### $\mapsto$ #0####### $\mapsto$ ########

Solution:

state	read	write	new state	move
S_i	1	1	S_{c1}	$\rightarrow$
S_i	0	0	S_{c0}	$\rightarrow$
S_{c1}	1	1	S_{p0}	$\leftarrow$
S_{c0}	0	0	S_{p0}	$\leftarrow$
S_{c1}	0	0	S_{p1}	$\leftarrow$
S_{c0}	1	1	S_{p1}	$\leftarrow$
S_{p0}	0/1	0	S_i	$\rightarrow$
S_{p1}	0/1	1	S_i	$\rightarrow$
S_{c1}/S_{c0}	#	#	$S_{p\#}$	$\leftarrow$
$S_{p\#}$	0/1	#	S_{ret}	$\leftarrow$
S_{ret}	0/1	0/1	S_{ret}	$\leftarrow$
S_{ret}	#	#	S_i	$\rightarrow$
S_i	#	#	$halt$	–