Name: Matriculation Number:

# Midterm Exam
# General CS II (320102)

April 8, 2013

**You have 75 minutes(sharp) for the test**;
Write the solutions to the sheet.
The estimated time for solving this exam is 71 minutes, leaving you 4 minutes for revising your exam.
You can reach 71 points if you solve all problems. You will only need 60 points for a perfect score, i.e. 11 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 | 5 | 6 | 6 | 8 | 12 | 12 | 10 | 12 | 71 | |
| 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

5pt
5min

**Problem 1.1 (Alternative Definition of a Tree)**

Prof. Simplovsky approaches Prof. Kohlhase at a conference in Saskatchewan and suggests a different definition of trees which he claims is simpler than Prof. Kohlhase's (in the slides) and yet it fully captures the notion of trees too:

*"A tree is a directed, connected acyclic graph with exactly one node with in-degree zero, which we call the root node."*

Is Prof. Simplovsky right? Explain your answer and prove the equivalence of the respective definitions or give an example that differentiates them.

---

**Note:** We call a directed graph connected, iff for any two nodes $n_1$ and $n_2$ there is a path in the underlying *undirected* graph (that we get by disregarding all directions of the edges) starting at $n_1$ and ending at $n_2$. (If this property holds for the directed graph itself, it is called *strongly* connected instead.)

---

**Solution:** Prof. Simplovsky is wrong; counter-example:

$a \to b$
$a \to c$
$b \to d$
$c \to d$

That is, we need the additional condition that every node except the root has an in-degree of 1.

(Thanks to Darko Pesikan for suggesting this problem.)

---

# 2 Positional Number Systems

**Problem 2.1 (Two's Complement Arithmetic)** 6pt 6min

1. What is the minimum number of bits which can be used to encode the following numbers: $A = 1023$, $B = 1024$, $C = -1025$, $D = -1024$? Explain, then perform the conversion.
2. Consider two numbers in decimal representation $A = 15$ and $B = 13$. Calculate $A + B$ and $A \cdot B$ in TCN and verify the result by converting the result back into decimal.

**Hint:** The product in TCN is done similar to the one in decimal. Don't forget to extend the input integers (to twice as many bits).

**Solution:**

# 3 Combinatorial Circuits

6pt

**Problem 3.1 (CSA and CCA)** 6min

Draw the basic building blocks for the following circuit elements:

1. n-bit Carry Chain Adder
2. n-bit Conditional Sum Adder

---

**Hint:**

No need to draw the diagram to the gate-level depth. Only the basic structure is expected.
In the drawing, you only need to show the structure to a level where the characteristic differences become apparent, you do not need to expand to gate level.

---

For both of the circuit elements given above, state their

- cost and
- depth

in Landau notation.

---

**Solution:** Check notes for GenCS-II

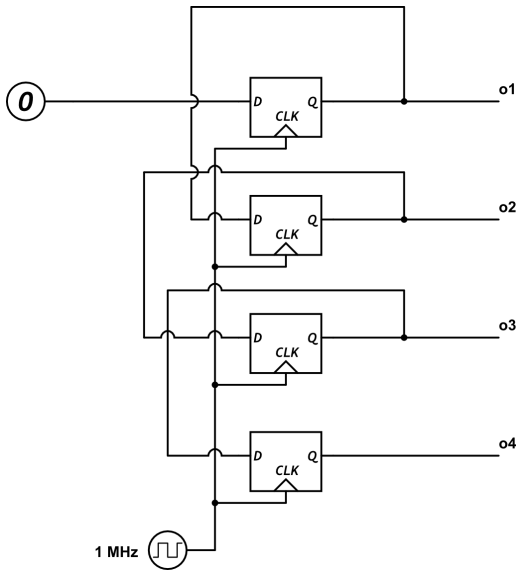
---

**Problem 3.2 (Doubler Circuit)** 8pt 8min

We have seen examples of binary counter. Now build a similar circuit which simulates a 4-bit binary doubler (i.e. a circuit that successively doubles a binary number). This means that given a binary number $x$ as input, your circuit would have the double as output and feed it into the input again. For example, if the input value is 1, then for the output: you first get 2, then 4, then 8 (in binary), and then 0s (overflow).

---

**Solution:** Doubling is fairly straightforward. You just do one bit shifting. The following diagram works:




---

# 4 Machine Programming

12pt
12min

**Problem 4.1 (Assembler Prime checking)**

Write an `ASM` program that checks whether a given natural number $n \geq 2$ is prime. You can find the number $n$ in $P(0)$. You should use $P(1)$ as the output where 0 is false and 1 is true and not alter the rest of the storage.

---

**Hint:** To simplify the program, you may use an extended set of jump instructions. `jump(>)` jumps if the accumulator has a number greater than zero and `jump(!=)` does it it has non-zero content.

---

**Solution:**

```
load 2
move acc in2
loadi 1
store 2
loadi 0
store 1
load 0
move acc in1
move in1 acc
sub 2
move acc in1
jump(>) -3
jump(!=) 4
loadi 1
add 1
store 1
jump 4
loadi 0
add 1
store 1
load 2
addi 1
store 2
sub 0
jump(>) 2
jump -19
loadi 0
store 2
move in2 acc
store 2
load 1
subi 2
jump(!=) 4
loadi 1
store 1
jump 3
loadi 0
store 1
stop 0
```

TODO: make solution nicer, it is going through too many numbers but works.

---

**Problem 4.2 (Arithmetic mean in $\mathcal{L}(\texttt{VM})$)** 12pt 12min

You are given a natural number $n > 1$ in $\mathcal{S}(0)$ and a sequence of $n$ non-negative integers stored in $\mathcal{S}(2) \dots \mathcal{S}(n+1)$. $\mathcal{S}(1)$ is available for intermediate calculations. Write an $\mathcal{L}(\texttt{VM})$ program which finds the average (in terms of arithmetic mean) of these numbers: $\frac{\sum_{i=2}^{n+1} \mathcal{S}(i)}{n}$. Of course, we are again dealing with integer division here. Make sure that after the execution, the stack contains only the result, in $\mathcal{S}(0)$.

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:**

```
peek 0 poke 1
peek 1 con 2 leq cjp 12
add
con 1 peek 1 sub poke 1
jp -15

con 0 poke 1
peek 2 peek 1 peek 0 mul leq cjp 11
con 1 peek 1 add poke 1 jp -17

con 1
peek 1
sub
poke 2
poke 1
poke 0
```

**Problem 4.3 (Fibonacci numbers and $\mathcal{L}$(VMP))** 10 min

Write a $\mathcal{L}$(VMP) static procedure which takes as argument a natural number $n$ and returns the $n^{\text{th}}$ Fibonacci number. Recursive definition of the Fibonacci numbers: $f(0) = 0$, $f(1) = 1$, $f(n) = f(n-1) + f(n-2)$.

To receive full points, the procedure has to run in linear time, i.e. for any input $n$, at most $cn$ operations are executed where $c$ is some constant. Otherwise, a solution that correctly computes the Fibonacci numbers will receive 70% of the points for this problem.

---

**Hint:** You should write a helper procedure.

A simple solution can be written which for input $n$ does exactly $n + 1$ procedure calls.

---

**Solution:** Define the following functions:

```
fibhelp(a,b,n) = if n<=0 then a else fibhelp(b,a+b,n-1);
fib(n) = fibhelp(0,1,n);
```

Write now the procedures:

```
<fibhelp> proc 3 #instr
con 0
arg 3
leq
cjp 5
arg 1
return
con 1
arg 3
sub
arg 2
arg 1
add
arg 2
call <fibhelp>
return

<fib> proc 1 #instr
arg 1
con 1
con 0
call <fibhelp>
return
```

---

# 5 Turing Machines

**Problem 5.1 (Encoding Turing Machines)** 12pt 12min

1. Design a TM that flips all the bits of an input and then doubles the result. If an overflow occurs, increase the length of the binary number.

**Hint:** Remember that a TMs tape has an infinite number of # around the input word

2. Now devise an encoding of this TM so that a suitable universal TM (you do not have to implement it) can take this encoding as an input word and use it to act like the encoded TM. Use the alphabet $\{0, 1, \#\}$.
3. Then actually encode the TM from 1.

**Solution:**

1.

| q | r | w | s | n |
|---|---|---|---|---|
| $q_0$ | 0 | 1 | $q_0$ | $\rightarrow$ |
| $q_0$ | 1 | 0 | $q_0$ | $\rightarrow$ |
| $q_0$ | # | # | $q_{c0}$ | $\leftarrow$ |
| $q_{c0}$ | 0 | 0 | $q_{c0}$ | $\leftarrow$ |
| $q_{c0}$ | 1 | 0 | $q_{c1}$ | $\leftarrow$ |
| $q_{c0}$ | # | 0 | $h$ | - |
| $q_{c1}$ | 0 | 1 | $q_{c0}$ | $\leftarrow$ |
| $q_{c1}$ | 1 | 1 | $q_{c1}$ | $\leftarrow$ |
| $q_{c1}$ | # | 1 | $h$ | - |

2. Encode the states with 2 bits: $q_0 \rightarrow 00$
   $q_{c0} \rightarrow 10$
   $q_{c1} \rightarrow 11$
   $h \rightarrow 01$

   Encode the read write bits with 2 bits: $0 \rightarrow 00$
   $1 \rightarrow 11$
   $\# \rightarrow 10$

   Encode the move arrows with 2 bits: $\rightarrow\rightarrow 01$
   $\leftarrow\rightarrow 10$
   $- \rightarrow 00$

   separate states by ##
   separate commands by #

3. ...###00#00#11#00#01##00#11#00#00#01##00#10#10#10#10##
   10#00#00#10#10##10#11#00#11#10##10#10#00#01#00##11#00#11#
   10#10##11#11#11#11#10##11#10#11#01#00###...