Name: Matriculation Number:

Midterm Exam
General CS II (320201)

March 27, 2007

You have one hour(sharp) for the test;
Write the solutions to the sheet.

The estimated time for solving this exam is 60 minutes, leaving you 0 minutes for revising your exam.

You can reach 26 points if you solve all problems. You will only need 23 points for a perfect score, i.e. 3 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	1.2	2.1	2.2	3.1	3.2	Sum	grade
total	2	4	4	5	5	6	26	
reached								

Good luck to all students who take this test

1 Graphs

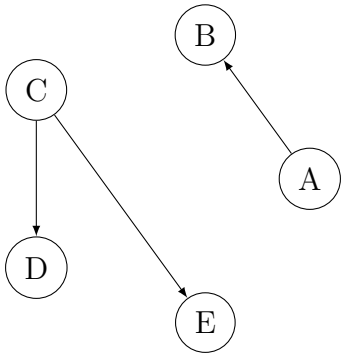
2pt
6min

Problem 1.1 (Directed Graphs)

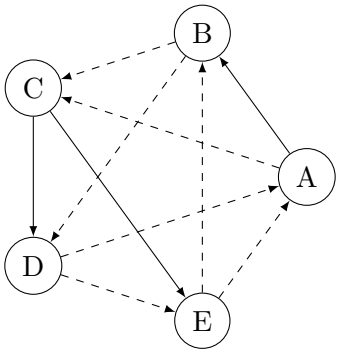
We call a directed graph (strongly) connected, iff for any two nodes $n_1 \neq n_2$ there is a path starting at n_1 and ending at n_2.

Complete the unconnected directed graph below by adding directed edges such that it becomes a strongly connected graph where each $indeg(n) = outdeg(n)$ for all nodes n.

How many initial and terminal nodes and how many paths does your final graph have?



Solution: This graph has neither an initial nor a terminal node and infinitely many paths since it is cyclic.



Problem 1.2 (Constructing Fully Balanced Binary Trees in SML) 10 10min

Write an SML function `MakeTree` that takes an integer $n \geq 0$ and returns a fully balanced binary tree with n nodes if one exists, and raises an exception `WrongInput` otherwise. The following datatype is used to construct binary trees:

```
datatype btree = leaf | parent of btree*btree;
```

Solution: A sample solution would look like this:

```
datatype btree = leaf | parent of btree*btree;

exception WrongInput;

fun construct 1 = leaf |
     construct n = let
      val temp = construct ((n-1) div 2)
       in
         parent(temp,temp)
      end;

fun check(1) = true |
    check(n) = (n mod 2 = 0) andalso check(n div 2);

fun MakeTree 0 = raise WrongInput |
     MakeTree n =
       if check(n+1) then construct(n)
       else raise WrongInput;
```

There is a simpler solution that does part of the check in every step:

```
fun MakeTree 1 = leaf |
    MakeTree n = if (n mod 2 = 0) then raise WrongInput
                 else parent(MakeTree(n div 2),MakeTree( n div 2));
```

---

2 Combinatorial Circuits

4pt

Problem 2.1 (Right and Left Shift on PNS) 12min

Consider for this problem the signed bit number system and the two's complement number system. Given a binary string $b = a_n \ldots a_0$. We define

1. the left shift function $lshift$ that maps the $n+1$-bit number $a_n \ldots a_0$ to the $n+2$-bit number $a_n \ldots a_0 0$
2. the right shift function $rshift$ that maps the $n+1$-bit number $a_n \ldots a_0$ to the n-bit number $a_n \ldots a_1$, discarding a_0.

Prove or refute the following two statements

- The $lshift$ function has the same effect in both number systems; i.e. for any integer z:
$$(\langle\!\langle lshift(B(z))\rangle\!\rangle^{-}) = \langle\!\langle lshift(B_n^{2s}(z))\rangle\!\rangle_{n+1}^{2s}$$
- The $rshift$ function has the same effect in both number systems; i.e. for any integer z:
$$(\langle\!\langle rshift(B(z))\rangle\!\rangle^{-}) = \langle\!\langle rshift(B_n^{2s}(z))\rangle\!\rangle_{n-1}^{2s}$$

Solution:

- $(\langle\!\langle rshift(B(z))\rangle\!\rangle^{-}) = \begin{cases} \lfloor \frac{z+1}{2} \rfloor & \text{if } z > 0 \\ , \lfloor \frac{z}{2} \rfloor & \text{if } z < 0 \end{cases}$.
- $(\langle\!\langle lshift(B(z))\rangle\!\rangle^{-}) = 2 * z$
- $\langle\!\langle rshift(B_n^{2s}(z))\rangle\!\rangle_{n-1}^{2s} = \lfloor \frac{z}{2} \rfloor$
- $\langle\!\langle lshift(B_n^{2s}(z))\rangle\!\rangle_{n+1}^{2s} = 2 * z$

Proof for the last equality:
$lshift(-a_n * 2^n + \sum_{k=0}^{n-1} a_k * 2^k) = -a_n * 2^{n+1} + \sum_{k=1}^{n} a_k * 2^k = 2 * (-a_n * 2^n + \sum_{k=0}^{n-1} a_k * 2^k)$

Problem 2.2 (A Binary Counter) 5pt 10min

Implement a 3-bit binary counter that counts from 111 down to 000 in steps of 1 and again starts from 111, doing one step with each clock pulse. You may use any combinational or sequential logic circuit that has been introduced in the lecture. Draw the circuit of your implementation with sufficient explanation.

Note: Assume that, initially, each storage element contains a 0.

Hint: Instead of building everything from elementary gates, first think of more complex circuits for doing arithmetics or storing values that have been introduced.

Solution: One possible solution uses D-flipflops to store the current number. The outputs, as well as the constant $111 = \langle\!\langle -1 \rangle\!\rangle_3^{2s}$, are fed into a 3-bit full adder, whose output is wired into the D-flipflops again. The clock signal is wired into the "enable" inputs of the D-flipflops.

Another solution uses a series of three toggling D-flipflops (where the inverted output is wired into the input); here, one could also use toggling JK-flipflops, but they have not been introduced in our lecture.

Note that we should assume edge-triggered D-flipflops in any case.

3 Machine Programming

5pt
10min

Problem 3.1 (Array Indexing in Assembler)

Given $n \geq 1$, stored in $P(0)$, and $a_1, \ldots, a_n$, stored in $P(1), \ldots, P(n)$, with $1 \leq a_i \leq n, i = 1, \ldots, n$, compute the "$n$-th order subscript" $a_{a_{\ldots a_n}}$ of a and store it in $P(1)$.

Note: With the above definition, the first-order subscript of a would be a_n, the second-order subscript would be a_{a_n}, and so on. An example initial setup for $n = 3$ could be:

i	0	1	2	3
$P(i)$	3	3	1	2

In this case, the program should compute $a_{a_{a_n}} = a_{a_{a_3}} = a_{a_2} = a_1 = 3$.

Solution:

P	instruction	comment
0	`LOAD` 0	
1	`MOVE` ACC $IN1$	store n in $IN1$
2	`JUMP`$_=$ $= 7$	if $n = 0$ then jump to 9
3	`LOADIN` 1 0	load the current result
4	`MOVE` ACC $IN1$	store it to be used as index
5	`LOAD` 0	decrement n
6	`SUBI` 1	
7	`STORE` 0	
8	`JUMP` -6	repeat loop
9	`MOVE` $IN1$ ACC	
10	`STORE` 1	
11	`STOP` 0	

Problem 3.2 (Static Procedure for Logarithm) 6 pt 12 min

Write a $\mathcal{L}(\texttt{VM})$ program that implements the `log` function for the integer logarithm defined as $\lfloor \log_b a \rfloor$ as a static procedure and calls that procedure to compute $\log_2 3$, as in the following μML listing (given in an SML-like syntax):

```
let
  fun log(b, a) =
    ...
in
  log(2, 3)
end
```

1. Complete the function in the above μML listing, using an an SML-like syntax.
2. Write down the $\mathcal{L}(\texttt{VM})$ program (in concrete, not abstract syntax) that results from compiling the μML program[1]. You may use any $\mathcal{L}(\texttt{VM})$ instruction except `peek` and `poke`.
3. Draw the evolution of the stack, including all intermediate steps.

Note: Assume a built-in `div` instruction that performs integer division. You may confine yourself to the cases $b > 1$ and $a > 0$.

Solution: The SML code:

```
let
  fun log(x, 1) = 0 |
      log(x, y) = 1 + log(x, div(y, x))
in
  log(2, 3)
end;
```

The μML code in abstract syntax (not part of the assignment, just for the sake of completeness!):

```
(
  [
    ("log", ["x", "y"], If(Leq("y", Con 1), Con 0,
       Add(Con 1, App("log", [Id "x", App("div", [Id "y", Id "x"])]))))
  ],
  App("log", [Con 2, Con 3])
)
```

The $\mathcal{L}(\texttt{VM})$ program (assuming that `div` is a procedure at address one; alternatively, we could assume `div` as a $\mathcal{L}(\texttt{VM})$ instruction):

```
proc 2 34
con 1
arg 2
leq
```

[1] You need not remember the exact definition of the compiler. Just give a $\mathcal{L}(\texttt{VM})$ program that computes the same function as the μML program and explain to which lines of the μML program the parts of the $\mathcal{L}(\texttt{VM})$ code relate.

```
cjp 5
con 0
return
con 1
arg 1
arg 2
call 1 (* div *)
add
arg 1
call 0
return
con 3
con 2
call 0
halt
```
