CPSC 201: Introduction to Computer Science
Carsten Schürmann
Date: March 3, 2003

Midterm Examination

Name: ____________________________

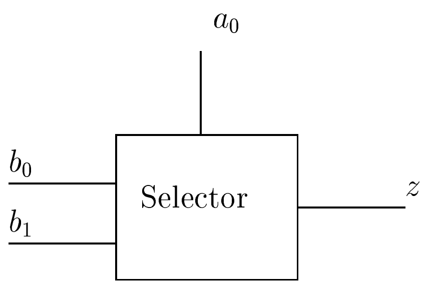
Instructions

- This is a closed book, closed notes, closed computer examination.
- There are 12 pages including 2 worksheets.
- This examination consists of 3 questions worth 100 points. The point value of each question is given with the question.
- Read each question completely before attempting to solve any part.
- Write your answers legibly in the space provided on the examination sheet. If you use the back of a sheet or a worksheet, indicate *clearly* that you have done so on the front.
- The worksheets attached to the end of this examination are for your own use; they will not be used in grading.

Problem	1	2	3	Total
Score				
Maximum	40	40	20	100

Problem 1: Boolean Logic/Circuits (40 points)

A 1-selector is a circuit that has three inputs a_0, b_0, and b_1 and one output z. If a_0 is set to 0, then the selector pushes through to z the level at b_0, otherwise that of b_1. Intuitively, you can think of this circuit to connect b_0 to z if $a = 0$ otherwise b_1.



Please answer the following four questions.

Question 1.1: (10 points)

Write out a truth table for the selector circuit.

a_0	b_0	b_1	z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Question 1.2: (10 points)

Write out Boolean expressions for z in terms of a_0, b_0,and b_1.

$$\begin{aligned} z &= (\neg a_0 \wedge b_0 \wedge \neg b_1) \vee (\neg a_0 \wedge b_0 \wedge b_1) \vee (a_0 \wedge \neg b_0 \wedge b_1) \vee (a_0 \wedge b_0 \wedge b_1) \\ &= (\neg a_0 \wedge b_0 \wedge (\neg b_1 \vee b_1)) \vee (a_0 \wedge (\neg b_0 \vee b_0) \wedge b_1) \\ &= (\neg a_0 \wedge b_0) \vee (a_0 \wedge b_1) \end{aligned}$$

Question 1.3: (10 points)

Sketch the circuit that implements the 1-selector using only NOR gates.

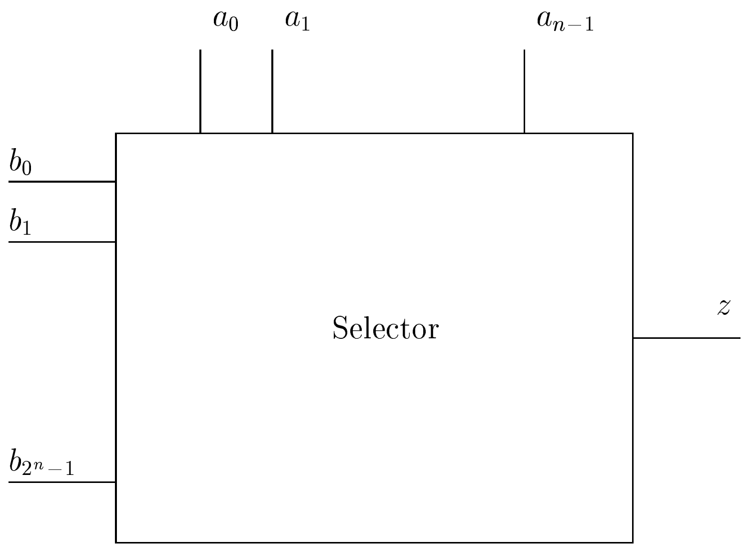
Let us write $\overline{\vee}$ as a symbole for nor.

$$\begin{aligned}(\neg a_0 \wedge b_0) \vee (a_0 \wedge b_1) &= \neg((\neg a_0 \wedge b_0)\overline{\vee}(a_0 \wedge b_1)) \\ &= \neg((a_0 \overline{\vee} \neg b_0)\overline{\vee}(\neg a_0 \overline{\vee} \neg b_1))\end{aligned}$$

where we define $\neg x$ as $x \overline{\vee} x$.

Question 1.4: (10 points)

An n-selector is a generalization of the 1-selector, in that it has n input signals $a_0, \ldots a_{n-1}$, and 2^n input signals $b_0 \ldots b_{2^n-1}$, but still only one output line z.



Implement a 2-selector circuit exclusively in terms of 1-selector circuits. Do not use any other gates.

Problem 2: Programming in ML (40 points)

Consider the following two functions. The not so familiar `fold` function

```
fun fold f b nil = b
  | fold f b (h::l) = f (h, fold f b l)
```

and the familiar `map` function:

```
fun map f nil = nil
  | map f (h::l) = (f h) :: map f l
```

Question 2.1: (10 points)

What is the type of `fold`?

```
fold : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
```

Question 2.2: (10 points)

Write out in little steps what

```
fold (fn (x, y) => x + y) 5 [3,2,1];
```

evaluates to. Use the $\stackrel{*}{\Longrightarrow}$ notation we have been using in class.

```
    fold (fn (x, y) => x + y) 5 [3,2,1]
=*=> (fn (x, y) => x + y) (3, fold (fn (x, y) => x + y) 5 [2,1])
=*=> 3 + fold (fn (x, y) => x + y) 5 [2,1]
=*=> 3 + (fn (x, y) => x + y) (2, fold (fn (x, y) => x + y) 5 [1])
=*=> 3 + 2 + fold (fn (x, y) => x + y) 5 [1]
=*=> 3 + 2 + (fn (x, y) => x + y) (1, fold (fn (x, y) => x + y) 5 [])
=*=> 3 + 2 + 1 + fold (fn (x, y) => x + y) 5 []
=*=> 3 + 2 + 1 + 5
=*=> 3 + 2 + 6
=*=> 3 + 8
=*=> 11
```

Question 2.3: (20 points)

Prove formally that for all functions `f` and `g`, for all values `b` and for all lists `L` the following hold:

```
fold f b (map g L) ≡ fold (fn (x, y) => f (g x, y)) b  L
```

Hint: The proof proceeds by induction on the structure of the list L, and $P_1 \equiv P_2$ is defined as $P_1 \stackrel{*}{\Longrightarrow} V$ iff $P_2 \stackrel{*}{\Longrightarrow} V$ for some value V.

Case: `L` = `nil`: The left hand side evaluates as follows:

```
      fold f b (map g nil)
 =*=> fold f b nil
 =*=> b
```

And the right hand side evaluates as follows:

```
      fold (fn (x, y) => f (g x, y)) b  nil
 =*=> b
```

Case: `L= h: L'`: The induction hypothesis has the follwing form:

```
fold f b (map g L') ≡ fold (fn (x, y) => f (g x, y)) b L'
```

Let's begin with the left hand side.

```
fold f b (map g (h :: L'))
=*=> fold f b ((g h) :: map g L')                          by definition of map
=*=> f (g h, fold f b (map g L'))                          by definition of fold
≡ f (g h, fold (fn (x, y) => f (g x, y)) b L')             by induction hypothesis
```

The right hand side is not much more difficult.

```
 fold (fn (x, y) => f (g x, y)) b (h :: L')
=*=> (fn (x, y) => f (g x, y)) (h, fold (fn (x, y) => f (g x, y)) b L')
                                                           by definition of fold
=*=> f (g h, fold (fn (x, y) => f (g x, y)) b L')          by function appliction
```

And again they are both the same and we are done. □

Problem 3: Programming Machine Code (20 points)

The goal of this problem is to implement a delay function that keeps the processor busy for n seconds (where n natural number) with doing nothing. All memory related instructions, such as `LOAD`, `LOADI`, `STORE`, `STOREI` take 0.4 seconds, `JMP` and `CJMP` do not take any time at all, and all others take 0.3 seconds. You can expect the wait time preloaded in register R_1.

`LOAD` A R_i: Load the contents of memory at address A into register R_i.

`LOADI` R_i R_j : Load the contents of memory at the address contained in register R_i into register R_j.

`MOVE` R_i R_j: Move the contents of register R_i to register R_j.

`STORE` R_i A: Store the contents of register R_i into memory at address A.

`STOREI` R_i R_j : Store the contents of register R_i into memory at the address contained in register R_j.

`ADD` R_i R_j: Add contents of registers R_i and R_j and store the result in register R_0.

`MULT` R_i R_j: Multiply contents of registers R_i and R_j and store the result in register R_0.

`CONST` C R_i: Moves constant C to register R_i.

`JMP` A: Jump to memory address A and continue program execution from that location.

`CJMP` A: Jump to memory address A iff register R_0 is zero.

`OUT` R_i: Output contents of register R_i.

`HALT`: Halt the program.

Please answer the following question.

Question 3.1: (20 points)

Implement the function that computes the delay function. *Hint: You may need much less the* 19 *instructions.*

100 CONST (∼1) 2		0.3 sec
101 ADD 1 2		0.3 sec
102 CJMP 107		0,0 sec
103 ADD 0 2		0.3 sec
104 CONST (∼1) 2		0.3 sec
105 STORE 42 99		0.4 sec
106 JMP 102		0.0 sec
107 STORE 42 99		0.4 sec
108 JMP 200		0.0 sec
⋮		
200 HALT		0.0 sec

Worksheet

Worksheet