Name: ______________________________

First Name: ___________________________

Student ID: ___________________________

1st session examination

Design of Digital Circuits SS2013

(252-0014-00S)

Markus Püschel, Frank K. Gürkaynak

Examination Rules:

1. Written exam, 90 minutes total.
2. No books, no calculators, no computers or communication devices. Five pages of handwritten notes are allowed.
3. Write all your answers on this document, space is reserved for your answers after each question. Blank pages are available at the end of the exam.
4. Put your Student ID card visible on the desk during the exam.
5. If you feel disturbed, immediately call an assistant.
6. Answers will only be evaluated if they are readable
7. Write with a black or blue pen (no pencil, no green or red color).
8. Show all your work. For some questions, you may get partial credit even if the end result is wrong due to a calculation mistake.

Question:	1	2	3	4	5	6	7	8	Total
Points:	5	10	15	10	12	5	12	6	75
Score:									

This page intentionally left blank

1. (a) (3 points) You receive the following 12-bit binary sequence:

`0000 0111 1100`

Which decimal numbers are encoded in this sequence, if you were told that the sequence contained:

Two 6-bit numbers using two's complement:	$1, -4$
A single 12-bit unsigned number:	124
Three 4-bit numbers using sign/magnitude:	$0, 7, -4$

(b) (2 points) In the lecture, it was explained that the two's complement was the better alternative to represent negative numbers. Name two main advantages of the two's complement representation over a sign/magnitude representation:

Solution:

1. Zero is represented only once
2. Standard binary addition works with two's complement numbers without additional effort
3. Associativity law holds

2. The following Verilog code defines a combinational circuit. We are interested in finding out the timing properties of this circuit.

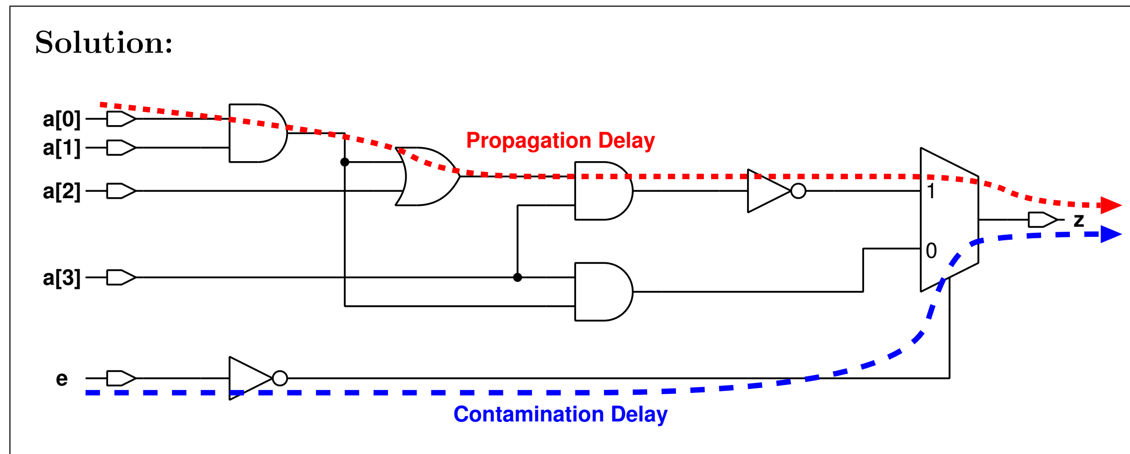
```
1  module gandalf ( input [3:0] a, input e, output z);
2
3    wire b,c,d;
4    reg f;
5
6    assign d = ~(a[3] & (a[2] | b));
7
8    always @ (*)
9      f <= a[3] & b;
10
11   assign z = (~e) ? d : f;
12   assign b = a[0] & a[1];
13
14 endmodule
```

The circuit is implemented using only the following basic logic building blocks: 2-input AND, 2-input OR, 2:1 Multiplexer, Inverter. The delay from any input to the output for each basic building block is given in the table below:

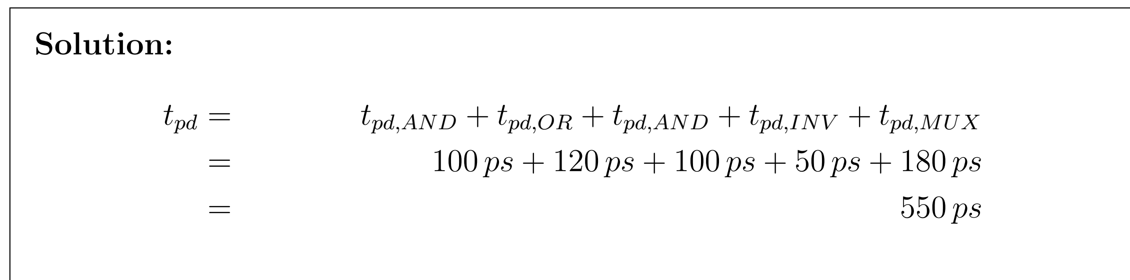
Description	**Delay** [ps]
2-input AND gate	100
2-input OR gate	120
Inverter	50
2:1 Multiplexer	180

Continue to the next page.

(a) (4 points) Draw a gate-level circuit diagram of the circuit using **only** the following basic logic gates: 2-input AND, 2-input OR, 2:1 Multiplexer, Inverter. *Note: there is no need for optimizations.*



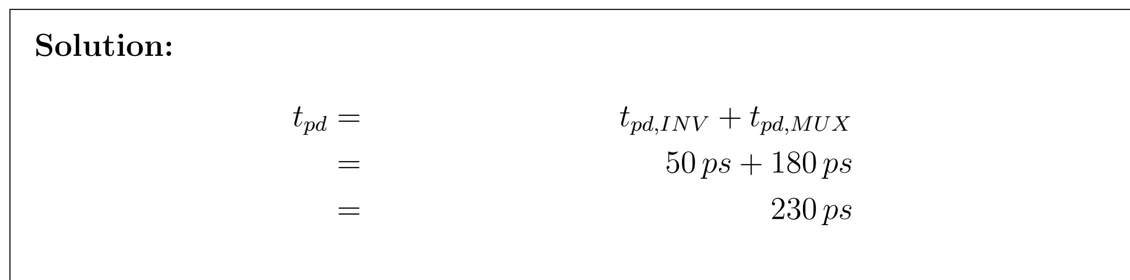
(b) (3 points) Determine the propagation delay of the circuit. Draw it on your schematic, and calculate the propagation delay using the delay values from the table.



Solution:

$$
\begin{aligned}
t_{pd} &= t_{pd,AND} + t_{pd,OR} + t_{pd,AND} + t_{pd,INV} + t_{pd,MUX} \\
&= 100\,ps + 120\,ps + 100\,ps + 50\,ps + 180\,ps \\
&= 550\,ps
\end{aligned}
$$

(c) (3 points) Determine the contamination delay of the circuit. Draw it on your schematic, and calculate the contamination delay using the delay values from the table.



Solution:

$$
\begin{aligned}
t_{pd} &= t_{pd,INV} + t_{pd,MUX} \\
&= 50\,ps + 180\,ps \\
&= 230\,ps
\end{aligned}
$$

3. The following Verilog code defines a Finite State Machine (FSM).

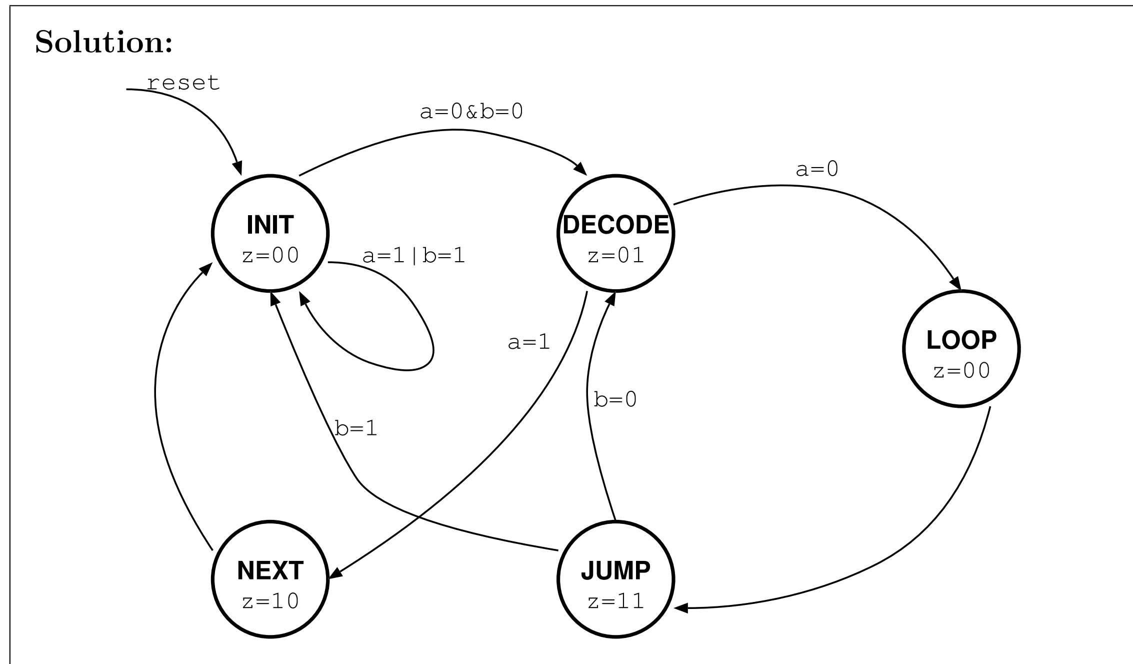
```
1  module fsm ( input a , input b , output [1:0] z,
2               input clk, input reset);
3
4  reg [2:0] state, nextstate;
5
6  parameter INIT   = 3'b000;
7  parameter DECODE = 3'b001;
8  parameter LOOP   = 3'b100;
9  parameter JUMP   = 3'b111;
10 parameter NEXT   = 3'b010;
11 //        DEF1   = 3'b011;
12 //        DEF2   = 3'b101;
13 //        DEF3   = 3'b110;
14
15 // next state calculation
16  always @( * )
17   case (state)
18     INIT:    if ((a==1'b0) & (b==1'b0) ) nextstate = DECODE;
19              else                        nextstate = INIT;
20     DECODE:  if (a) nextstate = NEXT;
21              else   nextstate = LOOP;
22     LOOP:    nextstate = JUMP;
23     JUMP:    if (b) nextstate = INIT;
24              else   nextstate = DECODE;
25     NEXT:    nextstate = INIT;
26     default: nextstate = INIT;
27   endcase
28
29 // state register
30  always @ (posedge clk, negedge reset)
31     if   (reset == 1'b0) state <= INIT;
32     else                 state <= nextstate;
33
34 // output logic
35   assign z = state[1:0];
36
37 endmodule
```

(a) (1 point) Is this a Moore or a Mealy FSM? Briefly explain.

Solution: Moore, outputs depend only on the present state and nothing else.

(b) (4 points) Draw the State Transition Diagram corresponding to the Verilog code given above.

Solution:



(c) (5 points) Complete the following state transition table for the FSM described by the Verilog code. To make writing easier, denote the `state` bits by S_2, S_1, S_0 and the `nextstate` bits by N_2, N_1, N_0. Note that the default behavior for the `nextstate` is to move to the `INIT` state. Since only five states have been defined, there are three additional states which we named `DEF1`, `DEF2`, `DEF3`. As an example, entries for these three default states and the `NEXT` state have been entered.

State				**Inputs**		**Next State**			
name	S_2	S_1	S_0	A	B	name	N_2	N_1	N_0
INIT	0	0	0	0	0	DECODE	0	0	1
INIT	0	0	0	0	1	INIT	0	0	0
INIT	0	0	0	1	0	INIT	0	0	0
INIT	0	0	0	1	1	INIT	0	0	0
DECODE	0	0	1	0	X	LOOP	1	0	0
DECODE	0	0	1	1	X	NEXT	0	1	0
LOOP	1	0	0	X	X	JUMP	1	1	1
JUMP	1	1	1	X	0	DECODE	0	0	1
JUMP	1	1	1	X	1	INIT	0	0	0
NEXT	0	1	0	X	X	INIT	0	0	0
DEF1	0	1	1	X	X	INIT	0	0	0
DEF2	1	0	1	X	X	INIT	0	0	0
DEF3	1	1	0	X	X	INIT	0	0	0

Note that there are different ways of writing this table to represent the same result.

(d) (1 point) For describing the `nextstate` is it better to use Products-of-Sums (POS) or Sums-of-Products (SOP)? Briefly explain.

Solution: Sums-of-Products, there are fewer entries with a 1 in them, and SOP requires one entry for each

(e) (3 points) Write down the Boolean Equations for the `nextstate` bits N_2, N_1, N_0, in either POS or SOP.
You don't need to minimize the equations.

Solution:
$N_0 = \overline{S_2}\,\overline{S_1}\,\overline{S_0}\,\overline{A}\,\overline{B} + S_2\,\overline{S_1}\,\overline{S_0} + S_2\,S_1\,S_0\,\overline{B}$

$N_1 = \overline{S_2}\,\overline{S_1}\,S_0\,A + S_2\,\overline{S_1}\,\overline{S_0}$

$N_2 = \overline{S_2}\,\overline{S_1}\,S_0\,\overline{A} + S_2\,\overline{S_1}\,\overline{S_0}$

(f) (1 point) Briefly explain how the output `z` could be obtained in an actual circuit implementation, what kind of logic circuit would be needed?

Solution: The output is directly obtained from the 2 least significant bits of the `state`, or $S_1\,S_0$.

4. (10 points) There are four Verilog code snippets in this section. Only one of these codes is syntactically correct. All others have a problem with the **syntax**. For each code, first state whether or not there is a mistake. If there is a mistake explain how to correct it. *Note: Assume that the behavior as described, is correct*

(a)

```
1 module one (input [1:0] sel, input [3:0] data, output z);
2
3   assign z = sel[1] ? (sel[0] ? data[0] : data[3])
4                     : (sel[0] ? data[2] : data[1]);
5 endmodule
```

Solution: This code is correct. The distribution of the data bits may seem strange, but we are not checking for behviour.

(b)

```
1  module mux2 ( input [1:0] i, input sel, output z);
2
3    assign z= (sel) ? i[1]:i[0];
4
5  endmodule
6
7  module two ( input [3:0] data, input sel1, input sel2, output z);
8
9    mux2 i0 (.i(data[1:0]), .sel(sel1), .z(m[0]) );
10   mux2 i1 (.i(data[3:2]), .sel(sel1), .z(m[1]) );
11   mux2 i2 (.i(m), .sel(sel2), .z(z) );
12
13 endmodule
```

Solution: This code has mistakes. In `module two` there is an additional signal `m` used. This has not been declared, it should be declared as `wire [1:0] m;`.

(c)

```
1  module three (input [1:0] sel, output reg [7:0] z);
2
3    always @ (sel)
4      if      (sel = 2'b01) z=8'b01010101;
5      else if (sel = 2'b10) z=8'b10101010;
6      else                  z=8'b00000000;
7
8  endmodule
```

Solution: This code has mistakes. The condition checking for `sel` has been written as = which is an assignment. It should be == in both instances.

(d)

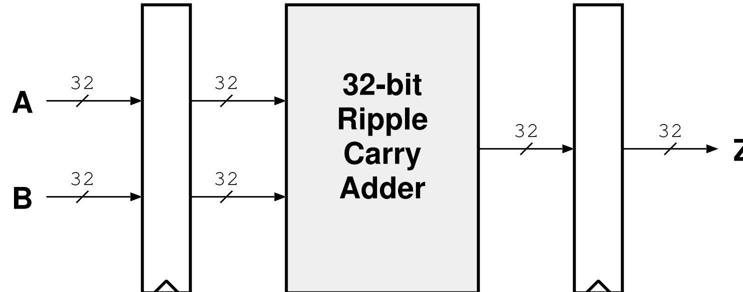
```
1  module four (input [1:0] sel, input neg, output reg [3:0] z);
2
3    always @ (neg, sel)
4        if (neg)    z = 4'b1111;
5        else        z = 4'b0000;
6        if (sel[1]) z = 4'b0001;
7        if (sel[0]) z = 4'b0010;
8
9  endmodule
```

Solution: This code has mistakes. There are 3 seperate `if` statements following `always`. These should be within a `begin ... end` block. Note that, it would not be correct to have three separate `always` statements as this would mean driving the signal `z` from three different processes.

5. In this section, you will compare three structures to add 32-bit binary numbers in terms of Latency, Throughput, Area and Maximum Operating Frequency. Assume the following performance numbers for the components in the question. *Note that the registers are considered ideal for timing: no propagation delay and no setup delay*

Description	**Delay** [ns]	**Area** [µm^2]
32-bit Ripple Carry Adder	4.0	4'000
16-bit Ripple Carry Adder	2.0	2'000
32-bit Carry Lookahead Adder	2.5	6'000
64-bit register	0.0	670
49-bit register	0.0	500
32-bit register	0.0	330

(a) (4 points) Consider the following 32-bit ripple carry adder pipeline stage and answer the following questions:



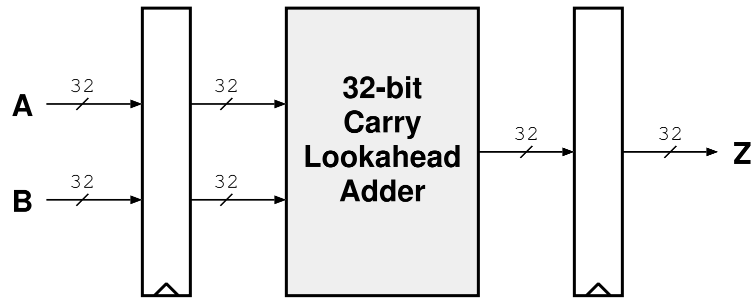
- What is the area occupied by the entire pipeline?
- How long does it take to compute one addition?
- What is the maximum operating frequency (in GHz) of this pipeline?
- How many additions can be completed in 1000 ns?

Solution:

$$
\begin{aligned}
Area &= A_{FF,64} + A_{RCA,32} + A_{FF,32} \\
&= 670 + 4000 + 330 \\
&= 5000 \\
Latency &= 4\,ns \\
MaxFrequency &= 1/4\,ns \\
&= 0.250\,GHz \\
Throughput &= 1000/4\,ns \\
&= 250\ additions\ per\ 1000\,ns
\end{aligned}
$$

Hint: $\frac{1}{1\,ns} = 1\,GHz$*, a clock with 1 GHz has a period of 1 ns.*

(b) (2 points) Consider the following 32-bit carry lookahead pipeline stage and answer the following questions:

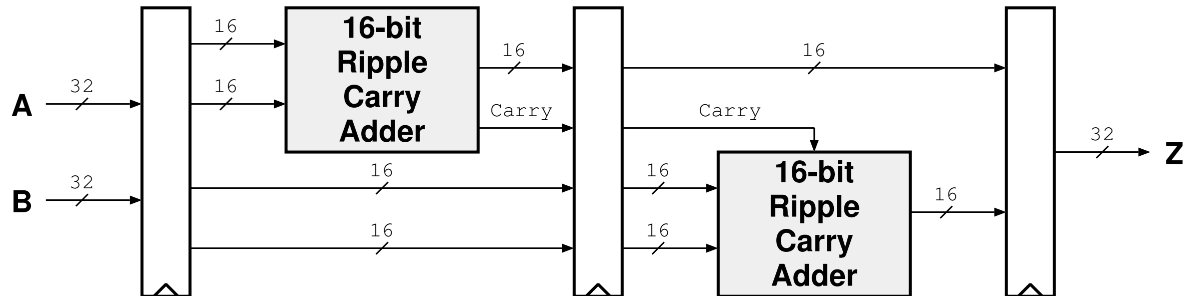


- What is the area occupied by the entire pipeline?
- How long does it take to compute one addition?
- What is the maximum operating frequency (in GHz) of this pipeline?
- How many additions can be completed in 1000 ns?

Solution:

$$\begin{aligned} Area &= A_{FF,64} + A_{CLA,32} + A_{FF,32} \\ &= 670 + 6000 + 330 \\ &= 7000 \\ Latency &= 2.5\,ns \\ MaxFrequency &= 1/2.5\,ns \\ &= 0.400\,GHz \\ Throughput &= 1000/2.5\,ns \\ &= 400\ additions\ per\ 1000\,ns \end{aligned}$$

(c) (2 points) Consider the following 32-bit adder with a 2 stage pipeline built out of two 16-bit ripple carry adders and answer the following questions:



- What is the area occupied by the entire pipeline?
- How long does it take to compute one addition?
- What is the maximum operating frequency (in GHz) of this pipeline?
- How many additions can be completed in 1000 ns?

Solution:

$$
\begin{aligned}
Area &= A_{FF,64} + A_{RCA,16} + A_{FF,49} + A_{RCA,16} + A_{FF,32} \\
&= 670 + 2000 + 500 + 2000 + 330 \\
&= 5500 \\
Latency &= 4\,ns \\
MaxFrequency &= 1/2\,ns \\
&= 0.500\,GHz \\
Throughput &= 1000/2\,ns \\
&= 500\ additions\ per\ 1000\,ns
\end{aligned}
$$

(d) (4 points) The *Latency* is the time it takes to calculate one addition, whereas the *Throughput* is the number of additions that can be calculated per unit time. It is obvious that the throughput will increase if you can reduce the latency. Is it possible to increase the throughput, even if you cannot reduce the latency? Briefly explain.

Solution: Yes. One solution is to introduce pipelining it is possible to improve the throughput as seen in the section c) of this question. Pipelining does not reduce latency, the computation of one data item still takes the same amount of time, however, the operation is broken down into smaller pieces, and as soon as the first part is completed, a new data item can be accepted, this improves the throughput. Another solution is to increase parallelism by, for example, duplicating the hardware.

6. (5 points) As covered in class, the execution speed of a program on a processor can be given as:

$$Execution\ Time = N \times CPI \times 1/f$$

Where N is the number of instructions, CPI is clocks per instruction and f is the clock frequency. $Execution\ Time$ will improve by either reducing N and CPI, or increasing f (or a combination thereof). List at least **five** improvements that can be made in order to improve the $Execution\ Time$.

Solution: Any five of the following could be accepted:

- **Reduce number of instructions**
  - adopt CISC, that uses instructions that can do more
  - improve the compiler so that it produces more optimized code
- **Reduce clocks per instruction**
  - adopt RISC, simpler instructions can be executed faster
  - add parallel execution units, do more per clock cycle
- **Increase clock frequency**
  - migrate to a more modern manufacturing technology
  - adopt pipelining
  - redesign and improve timing critical components in the circuit (adders, alu etc)
  - *Could also be accepted*: overclock the system (use higher voltage, clock frequency)

7. (12 points) Consider the following MIPS program. For clarity the addresses have been written using only 4 hexadecimal digits. Leading hexadecimal digits are all zeroes (the real start address is `0x00003000`).

```
0x3000  start:    addi $s0, $0,   4
0x3004            xor  $s1, $s1,  $s1
0x3008            addi $s2, $0,   10
0x300C            sw   $s2, 0($s1)
0x3010            addi $s2, $s2,  6
0x3014            add  $s1, $s1,  $s0
0x3018            sw   $s2, 0($s1)

0x301C            addi $a0, $0,   11
0x3020            sll  $t1, $a0,  1
0x3024            and  $a1, $a0,  $t1
0x3028            jal  absdiff
0x302C            sw   $v0, 4($s1)

0x3030            lw   $a0, 0($0)
0x3034            lw   $a1, 0($s0)
0x3038            jal  absdiff
0x303C            lw   $t3, 8($0)
0x3040            sub  $t2, $t3,  $v0

0x3044  done:     j    done

0x3048  absdiff:  sub  $t1, $a0,  $a1
0x304C            slt  $t2, $t1,  $0
0x3050            beq  $t2, $0,   pos
0x3054            sub  $t1, $a1,  $a0
0x3058  pos:      add  $v0, $0,   $t1
0x305C            jr   $ra
```

We are interested in determining the value of some registers at the end of the program execution when the program reaches line 0x3044. Fill in the following table, writing the value of the indicated registers at the end of the program, and at which line these values have been written into these registers.

As an example: at the end of execution the register $s0 will have the value 4. This value has been written into the register while executing line 0x3000.

Register	**Value**	**Assigned on line**
$s0	4	0x3000
$s2	16	0x3010
$t1	6	0x3054
$t2	3	0x3040
$t3	9	0x303C
$ra	0x303c	0x3038

8. You are involved in designing a computing system using a cache (256 kbyte, 4-way set associative cache using 1 kbyte blocks). Your first design has some cache performance problems. Your colleagues made the following suggestions. For each suggestion, first state whether or not the idea will work, and then *briefly* explain why. If the idea works explain under what conditions.

   (a) (2 points) Alain: "We have too many cache misses due to *conflicts*. We need to reduce the degree of associativity, so that we reduce conflict misses in the cache":

   Solution:
   This idea **will not work**. Just the opposite: increasing set associativity gives data more possibilities to be stored in the cache without replacing other data. This reduces conflict misses.

   (b) (2 points) Beatrice: "There are many *compulsory* cache misses. To combat this, we should increase our block size"

   Solution: This idea **could work**. A larger block size will take advantage of spatial locality and assume that nearby data items will also be accessed by the program. If the program has such accesses, the first data access will result in a compulsory miss, but the subsequent accesses will find data in the cache.

(c) (2 points) Cathy: "Our cache has many *capacity* misses. Instead of using a set associative cache, we should convert it to a direct mapped cache of the same size. This will allow more sets to be stored in the cache, hence reducing capacity misses"

Solution: This idea **will not work**. The organization of the cache does not change its capacity. The capacity miss occurs because data that is needed can not fit into the cache.