Name: _____

First Name: _____

Student ID: _____

## 1st session examination

# Design of Digital Circuits SS2015

## (252-0014-00S)

Srdjan Capkun, Frank K. Gürkaynak

Examination Rules:

1. Written exam, 90 minutes in total.
2. No books, no calculators, no computers or communication devices. Six pages of hand-written 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.
9. Points for every part are indicated in the exam. They should correspond to the expected difficulty of the questions. Consider this when allocating your time.

| Question: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Total |
|---|---|---|---|---|---|---|---|---|
| Points: | 6 | 12 | 10 | 18 | 6 | 18 | 5 | 75 |
| Score: | | | | | | | | |

*This page intentionally left blank*

1. In this exercise we want to develop a *new* standard for representing Swiss postcodes in binary format. Swiss postcodes have four digits and ranges from 1000-9999.

   (a) (2 points) How many bits in total are needed if we want to represent the Swiss postcodes using **two's complement** binary format? Encode the postcode of Wilchingen, SH (8217)[1] using this format.

   > **Solution:** 15 bits are needed: there are 9'000 different possible postcodes (more or less), and you need at least 14 bits to represent them ($2^{14}$==16'536). You need another bit for the two's complement.
   > $(8217)_{10} = 8192 + 16 + 8 + 1 = (010\,0000\,0001\,1001)_2$

   (b) (2 points) How many bits in total are needed if we represent each decimal digit as a separate unsigned binary number? Encode the postcode of Wilchingen, SH (8217) using this format.

   > **Solution:** 16 bits are needed: For each decimal digit 4 bits will be needed. There are 4 digits, that makes 16 in total
   > $(8, 2, 1, 7)_{10} = (1000, 0010, 0001, 0111)_2$

   (c) (2 points) How many bits in total are needed if we represent two decimal digits at a time as a separate unsigned binary number? Encode the postcode of Wilchingen, SH (8217) using this format.

   > **Solution:** 14 bits are needed: For two decimal digits (00-99), 7 bits will be needed ($2^7$==128). There are two such numbers digits, that makes 14 in total
   > $(82, 17)_{10} = (101\,0010, 001\,0001)_2$

[1] We needed a *not so difficult* postcode for this exercise, otherwise there is no special reason for Wilchingen.

2. For this question, use the following truth table for a 4-input logic function called $Z$.

| **Input** | | | | **Output** |
|---|---|---|---|---|
| $A$ | $B$ | $C$ | $D$ | $Z$ |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

(a) (2 points) One of your friends has determined the following Boolean equation for $Z$:

$$Z = \overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot C + A \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} \cdot D$$

Does the equation match the truth table? If not, specify which terms in the equation are incorrect and what terms are missing, if any.

**Solution:** The equation does not match the truth table. There are the following problems:

- $\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$ is missing
- $\overline{A} \cdot \overline{B} \cdot C \cdot D$ is not in the truth table, but is included in the equation as part of the term $\overline{A} \cdot \overline{B} \cdot C$

(b) (2 points) The equation that your friend has written in 2a) also contains some redundant terms. Can you simplify it?

**Solution:**

$$Z = \overline{A} \cdot \overline{B} \cdot C + A \cdot C + \overline{A} \cdot B \cdot \overline{C} \cdot D$$

or

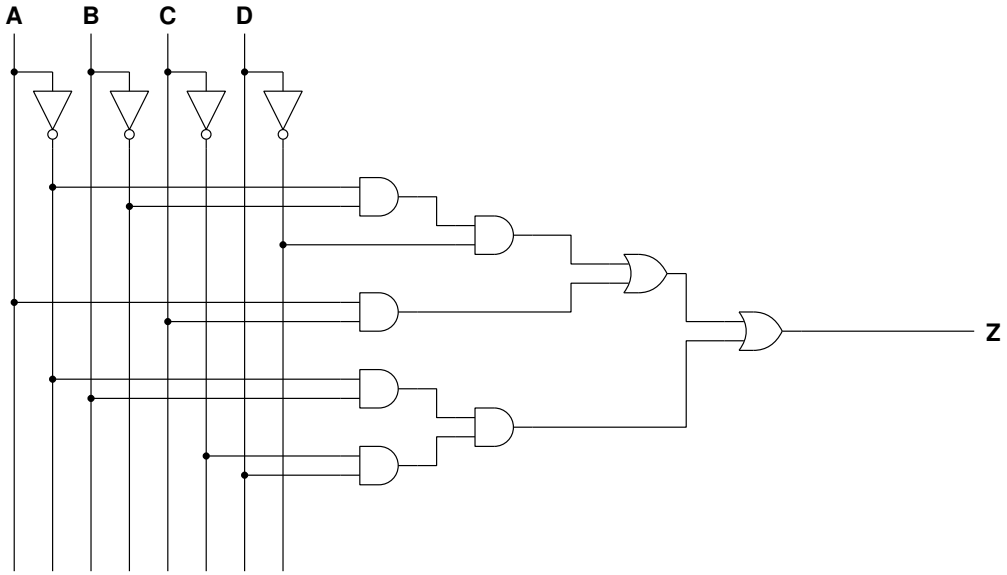
$$Z = \overline{B} \cdot C + A \cdot B \cdot C + \overline{A} \cdot B \cdot \overline{C} \cdot D$$

(c) (2 points) Derive an optimized correct Boolean equation for the same truth table. (*Hint: use a Karnaugh map*)

**Solution:**

$$Z = \overline{A} \cdot \overline{B} \cdot \overline{D} + A \cdot C + \overline{A} \cdot B \cdot \overline{C} \cdot D$$

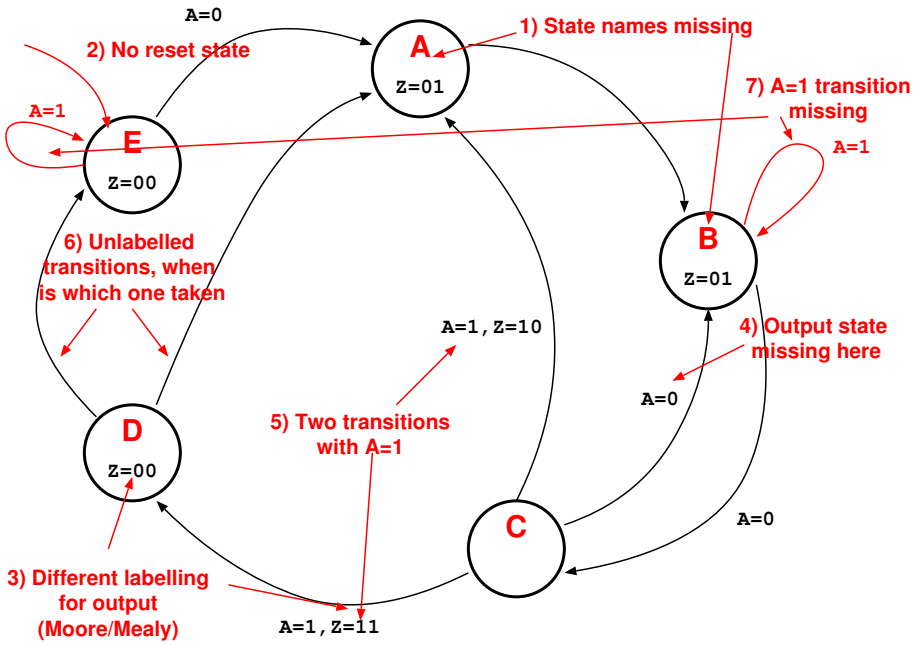
(d) (4 points) Draw a gate-level schematic that realizes the function $Z$ using **only** 2-input AND, OR gates. Assume that you have all the inputs $(A, B, C, D)$ and their complements $(\overline{A}, \overline{B}, \overline{C}, \overline{D})$ available.




(e) (2 points) Assume that all the gates (AND, OR, NOT) in the previous diagram have a propagation delay of 100 ps and a contamination delay of 50 ps. What is the delay of the longest (*critical*) path and the *shortest* path of this circuit?

**Solution:** In the solution above, the *critical* path goes through one inverter, 2 AND gates, and 2 OR gates and is $(5 \times t_{pd} ==)$ 500 ps. The *short* path goes through one AND gate and two OR gates and equals to $(3timest_{cd} ==)$150 ps.
Note: Depending on how the circuit is drawn the numbers could change slightly.

3. (a) (5 points) One of your colleagues is designing a finite state machine with a one-bit input ($A$) and a two-bit output ($Z$). He has started designing a state transitioning diagram as given below:




Even if you do not know the exact functionality of the FSM, there are some obvious problems with this diagram. Help your colleague by identifying the problems in this diagram. List *as many mistakes as you can find* in this diagram.

**Solution:** There are many problems with this diagram

1. The states do not have identifying names or labels
2. There is no reset state
3. Most states have a Moore labelling (output state in the bubble), one has a Mealy type labelling (output given with input transitions)
4. The output state is missing for the transition from the unlabelled state with $a = 0$.
5. There are two different transitions both with $a = 1$ from the same state
6. There are two different transitions from state $Z = 00$ that are not labelled
7. For two states only $a = 0$ makes a transition. What will happen with $a = 1$ is not shown, presumably will remain in the same state.

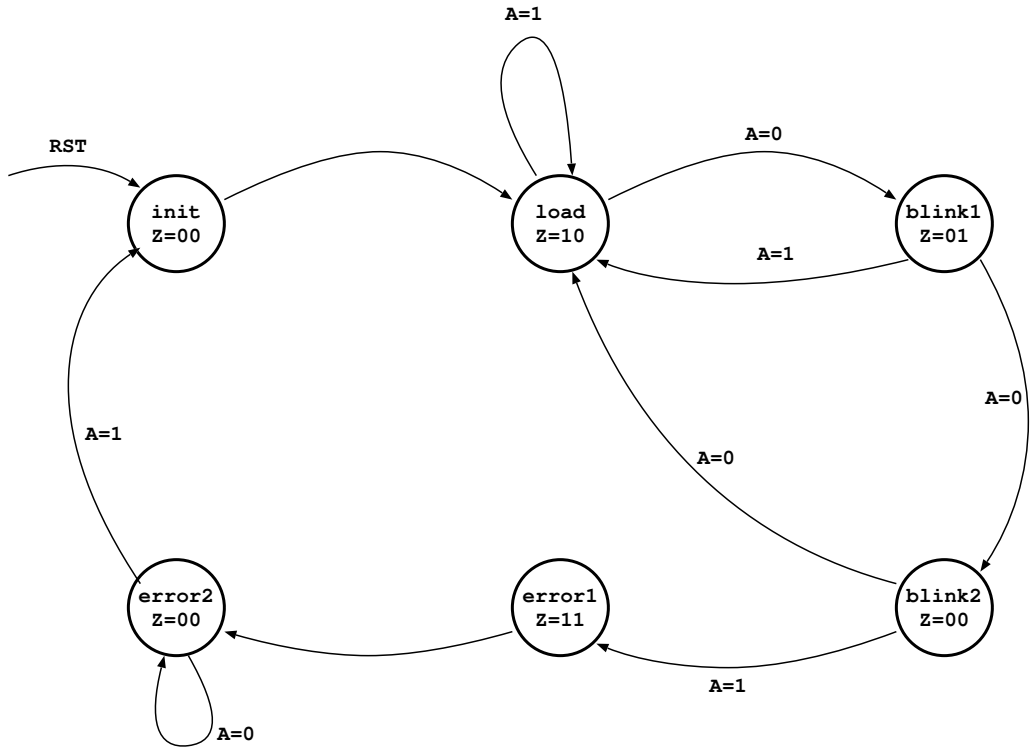
For every correctly spotted mistake you will get points, you will get full points if you have spotted 5 of the above.

(b) (4 points) After learning from his mistakes, your colleague has proceeded to write the following Verilog code for a much better (and different) FSM. The code has been verified for syntax errors and found to be OK.

```
1  module fsm (  input CLK, RST, A,
2                output [1:0] Z);
3
4    reg [2:0] next, present;
5
6    parameter init   = 3'b000;
7    parameter blink1 = 3'b010;
8    parameter blink2 = 3'b011;
9    parameter load   = 3'b100;
10   parameter error1 = 3'b110;
11   parameter error2 = 3'b111;
12
13   assign Z = (present == error1) ? 2'b11 :
14              (present == blink1) ? 2'b01 :
15              (present == load)   ? 2'b10 : 2'b00;
16
17   always @ (present, A)
18     case (present)
19       init   : next <= load;
20       load   : if (~A) next <= blink1;
21       blink1 : if (A) next <= load;
22                else   next <= blink2;
23       blink2 : if (A) next <= error1;
24                else   next <= load;
25       error1 : next <= error2;
26       error2 : if (A) next <=init;
27       default: next <= present;
28     endcase
29
30   always @ (posedge CLK, posedge RST)
31      if (RST)  present <= init;
32      else      present <= next;
33
34 endmodule
```

Draw a proper state transition diagram that corresponds to the FSM described in this Verilog code.

**Solution:**




(c) (1 point) Is the FSM described by the previous Verilog code a Moore or a Mealy FSM? Why?

**Solution:** Moore, the output $Z$ only depends on the state (*present*) and not on the input ($A$).

For questions 4,5 and 6 we will discuss a real-time graphics application where each picture element (pixel) is stored as an array of 32-bit values in the memory. In this example the 32-bits will hold the color information for each individual pixel in the following way:

**Bits 31:24** all zeroes

**Bits 23:16** an unsigned 8-bit value for the Red value

**Bits 15:8** an unsigned 8-bit value for the Green value

**Bits 7:0** an unsigned 8-bit value for the Blue value

The array is a continuous memory region arranged in 1024 rows containing 1024 pixels each. It occupies 4 Mbytes of memory starting from `0x2000 0000` and ending at `0x203F FFFC`

In this exercise we will want to apply a *special dimming function* to the picture. The *red* and *blue* color values will be **halved**, while the *green* component which was found to be too dominant will be **quartered**, i.e. divided by four. (*Note: divisions by powers of two can be performed by simple right shifts*).

An example input value (`0x00FF C436`) and how the values get modified in hexadecimal, binary and decimal notations is shown below. (*Note that the first 8 bits are always 0, and need not be processed.*)

| Bit | | 31 EMPTY | RED | GREEN | BLUE 0 |
|---|---|---|---|---|---|
| Original | (HEX) | 0 0 | F F | C 4 | 3 6 |
| Input | (BIN) | 0000 0000 | 1111 1111 | 1100 0100 | 0011 0110 |
| | (DEC) | 0 | 255 | 196 | 54 |
| Dimmed | (HEX) | 0 0 | 7 F | 6 2 | 1 B |
| Output | (BIN) | 0000 0000 | 0111 1111 | 0011 0001 | 0001 1011 |
| | (DEC) | 0 | 127 | 49 | 27 |
| Operation | | none | HALVE | QUARTER | HALVE |

4. In this question you will write a small MIPS program that halves/quarters the RGB color values of the entire image as described in the page before.

   (a) (10 points) Let us first develop a small MIPS assembly routine that will halve/quarter each 8-bit color component (Red, Green, Blue) individually and combine them together. The problem is that you can not just divide the 32-bit number that keeps the combined RGB value, you have to separate the values first, divide them individually and combine them again. (*Hint: consider using 'and', 'or' and 'shift' operators*)

   Assume that the 32-bit input value is located in the register $a0. Please provide the dimmed value in the register $v0.

```
1        addi $t0, $t0, 255    # initialize a mask
2
3        # GET B
4        and  $t1, $a0, $t0    # mask only the B into $t1
5        srl  $t1, $t1, 1      # divide B by 2 (unsigned number)
6
7        # GET G
8        srl  $a0, $a0, 8      # shift the number by 8 bits
9        and  $t2, $a0, $t0    # mask only the G into $t2
10       srl  $t2, $t2, 2      # divide G by 4 (unsigned number)
11
12       # GET R
13       srl  $a0, $a0, 8      # shift the number by 8 bits
14       and  $v0, $a0, $t0    # mask only the R into $v0
15       srl  $v0, $v0, 1      # divide R by 2 (unsigned number)
16
17       # ASSEMBLE
18       sll  $v0, $v0, 8      # shift the result by 8 bits left
19       or   $v0, $v0, $t2    # or the divided value of G in $t2
20       sll  $v0, $v0, 8      # shift the result by 8 bits left
21       or   $v0, $v0, $t1    # or the divided value of B in $t1
```

   We use the mask in $t0 to pick out exactly the 8 LSB in each step. We first start with B, mask it to $t1, and divide it. Next we shift the entire number by 8 to the right. This way the next color value is ready to be processed (G). We mask and shift it by 2 to divede by 4 in $t2. The last value (R) can then be already copied to $v0. Now we move back and 'or' the calculated values in the reverse order first G in $t2 and then B in $t1.

   There are many alternative solutions that would work (i.e. using `lb` and `sb` which was not covered in class).

(b) (2 points) We want to convert the routine from the previous exercise into a subroutine named **dim_pixel** that can be called from a main program. Make the necessary modifications (additions and/or changes, if necessary) to your code from the previous part so that it can be a proper subroutine.

```
1
2 dim_pixel: # function label
3
4        ### Your code from the previous part
5        ### no need to replicate it if no changes
6        ### are needed to the program
7
8        jr $ra   # jump back to the main ($ra)
```

Since you will not be jumping out of this subroutine again, there is technically no need to save anything on stack. You will not lose points if you do so.

(c) (6 points) Now we need to make one program that will loop over the entire picture, load the color values, call the function **dim_pixel** and write the result back again. Write this program using MIPS assembly.

```
1        # initializations
2        lui  $s0, 0x2000          # load start address
3        ori  $s0, $s0, 0x0000     # 0x2000 0000
4        lui  $s1, 0x2040          # load end address
5        ori  $s1, $s1, 0x0000     # 0x2040 0000
6
7  loop: lw   $a0, 0($s0)      # load one value
8        jal  dim_pixel        # call subroutine
9        sw   $v0, 0($s0)      # store back value
10       addi $s0, $s0, 4      # next pixel address
11       beq  $s0, $s1, done   # end of loop?
12       j    loop
13
14 done: # finished execution
```

The solution has to be consistent with your answer to 4a. i.e. if you assumed `$a0` contains the address of the pixel, and not the value, the address needs to be loaded at this point.

5. In this question we will calculate how fast our program from question 4 will run.

   (a) (3 points) How many instructions will be necessary to calculate the dimming operation on the *whole* image in Question 4c? If you have not answered Question 4a or 4b, assume that you need 10 instructions for one **dim_pixel** subroutine. If you make further assumptions please state them clearly. (*Note: you can use approximations to simplify calculations.*)

   **Solution:**
   The inner loop in question 4c has 6 instructions. One of these instructions calls the subroutine that has 14 further instructions (including the `jr`). Thus per pixel we will need 20 instructions. For the entire image this loop will be executed 1'024 times 1'024 == 1'048'576 (approximately 1 million). The initial instructions can be ignored.
   In total we will need slightly more than 20 million instructions (20'971'520). The answer will depend on your answer to question 4 (or your assumption).

   (b) (3 points) For a real-time video application, you realize that you can afford at most 10 ms of time until this operation is completed. Assuming that you are using an *ideal single cycle MIPS architecture*, what is the minimum *clock frequency* that your processor has to run at, so that your program finishes within 10 ms?

   **Solution:**
   $$T = N \cdot CPI \cdot \frac{1}{f}$$
   Where $T$ is the time to finish the operation, in this case 10 ms, $N$ was calculated in the previous exercise to be 20 million, $CPI$ for an ideal single cycle processor is 1. From here we can calculate the clock frequency (f) to be 2 GHz.

*Hint:*
$1\,s = 1000\,ms = 1'000'000\,\mu s = 1'000'000'000\,ns = 1'000'000'000'000\,ps$
$1\,Hz = 1\,\frac{1}{s}$; $1\,kHz = 1'000\,\frac{1}{s}$; $1\,MHz = 1'000'000\,\frac{1}{s}$; $1\,GHz = 1'000'000'000\,\frac{1}{s}$;

6. In this question, we will examine if we can make the dimming operation from question 4 run faster if we design a specific instruction **dim** just for this purpose.

   (a) (5 points) Design a simple *combinational* circuit that takes a 32-bit input $A$ and performs the dimming operation described in question 4. Complete the following Verilog code for this operation, or if you prefer, draw a detailed circuit schematic that explains the circuit.

```
module dim (  input  [31:0] A,
              output [31:0] Z);

  assign Z= {8'b0000_0000,
             1'b0,  A[23:17],
             2'b00, A[15:10],
             1'b0,  A[ 7:1]};

endmodule
```

   (b) (1 point) What is the propagation delay of this circuit? Assume all logic gates will have a propagation delay of 100 ps. *(Caution: tricky question.)*

> **Solution:** There are no active parts to this circuit, it contains only wiring and will virtually have no delay.

(c) (6 points) Now that we have the **dim** block designed, we want to enhance our processor with an instruction that uses this block. The *Control Unit* of MIPS has already been modified to generate a signal *ExecuteDim* that will be set to '1' whenever the new **dim** instruction has been decoded.

Two of your colleagues make two different suggestions.

- Josh says: "*Let us make **dim** an R-type instruction. The block **dim** can be placed in parallel with the ALU and it will take its input from SrcA. An additional multiplexer can select between the output of the ALU and the output of the **dim** block depending on the value of ExecuteDim. You execute **dim** by running* `dim $dest, $src`"
- Sandra says: "*We can combine the* `sw` *operation with **dim** and make it an I-type instruction. We can use the WriteData input of the RAM as an input to the **dim** block. A multiplexer can then select between the WriteData and the output of the **dim** block depending on the value of ExecuteDim. You execute **dim** by running* `dim $src, offset($addr)`*. The value in $src, will be dimmed and written to the address $addr+offset.*"

Does Josh's idea work? Does Sandra's idea work? Whose idea do you think is better? Why?

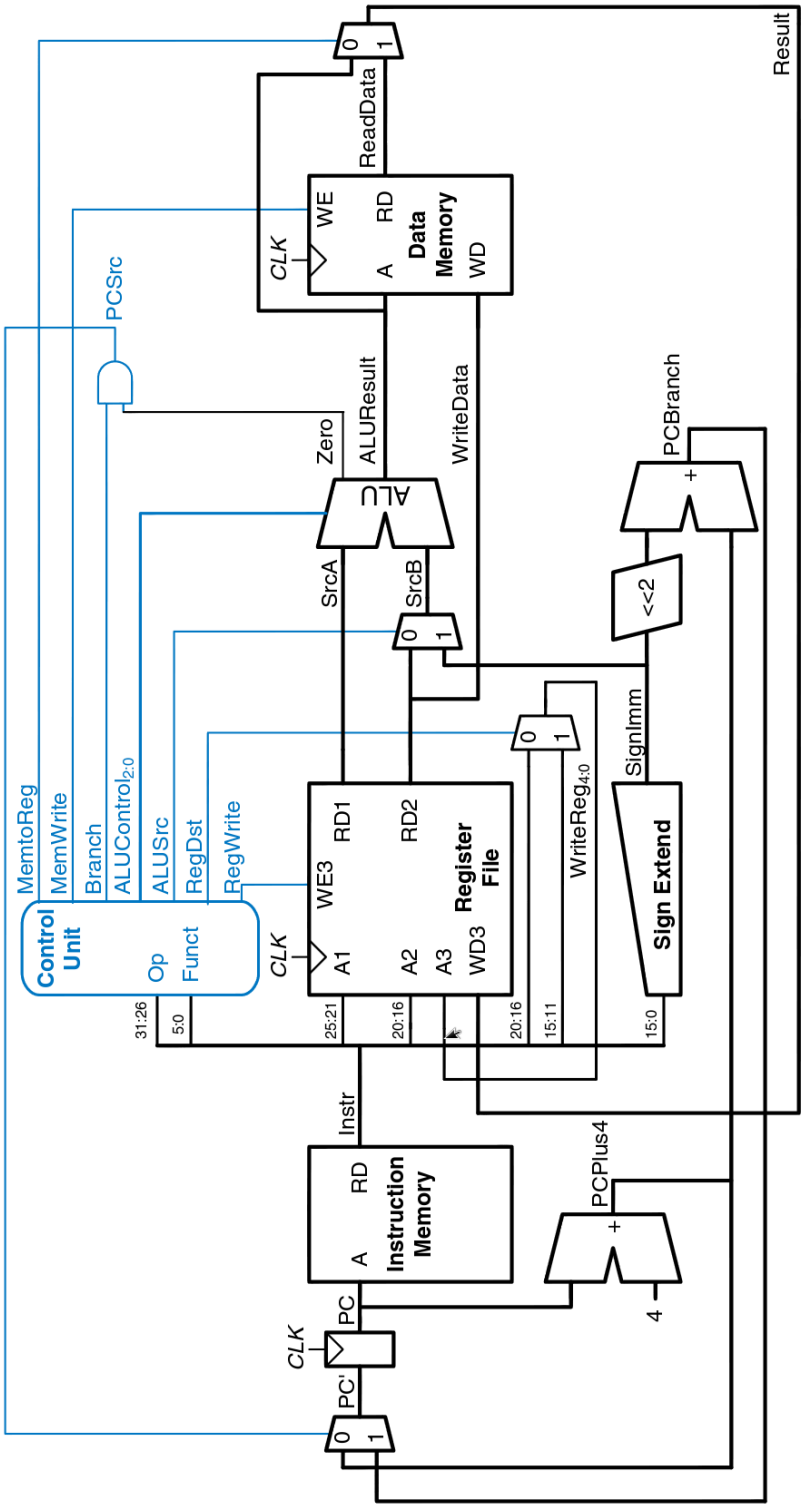
**Solution:** Both ideas would work. Sandra has a better idea, as it will further reduce the number of instructions by one.

```
# Josh version
loop:  lw   $a0, 0($s0)      # load one value
       dim  $a0, $a0         # execute the 'dim' function
       sw   $a0, 0($s0)      # dim store back value
       addi $s0, $s0, 4      # next pixel address
       beq  $s0, $s1, done   # end of loop?
       j    loop
```

```
# Sandra version
loop:  lw   $a0, 0($s0)      # load one value
       dim  $a0, 0($s0)      # dim and store back value
       addi $s0, $s0, 4      # next pixel address
       beq  $s0, $s1, done   # end of loop?
       j    loop
```

As long as you can make a reasonable case for it, answers defending the idea of Josh will also be accepted as correct. The correct answer also depends on the hardware implementation you came up with. In 6a/b, the expected answer only has wiring, therefore does not add much additional delay apart from a multiplexer. This is the reason why Sandra's idea works better. If the dim function had non-negligible delay, it could be argued that Josh's idea keeps this delay in parallel to the ALU and does not slow down the circuit.

(d) (6 points) Modify the standard single cycle block diagram given below so that it will allow the new **dim** instruction based on one of the ideas mentioned above. *Please explicitly state whose idea you implement.*



*This page intentionally left blank*

7. (a) (5 points) When considering the performance of caches, what are compulsory cache misses? Is there any way to reduce them? How?

**Solution:**

Compulsory misses occur when a data is accessed for the first time in a program. Since there were no prior accesses this data will not be located in the cache and will have to be fetched.

A larger block size can be used to reduce the compulsory misses. This takes advantage of spatial locality. When there is a cache miss, not only the required data is loaded into the cache, adjacent words are also automatically loaded into the cache, reducing the possibility of further compulsory misses.