Name: Student ID:

## Final Examination

# Design of Digital Circuits (252-0028-00L)

# ETH Zürich, Spring 2017

Professors Onur Mutlu and Srdjan Capkun

| | |
|---|---|
| Problem 1 (70 Points): | |
| Problem 2 (50 Points): | |
| Problem 3 (40 Points): | |
| Problem 4 (40 Points): | |
| Problem 5 (60 Points): | |
| Problem 6 (60 Points): | |
| Total (320 Points): | |

**Examination Rules:**

1. Written exam, 90 minutes in total.
2. No books, no calculators, no computers or communication devices. Six 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. Clearly indicate your final answer for each problem. Answers will only be evaluated if they are readable.
5. Put your Student ID card visible on the desk during the exam.
6. If you feel disturbed, immediately call an assistant.
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. Please write your initials at the top of every page.

**Tips:**

- **Be cognizant of time.** Do not spend too much time on one question.
- **Be concise.** You may be penalized for verbosity.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.
- **Write legibly.** Show your final answer.

*This page intentionally left blank*

# 1 Potpourri

## 1.1 Processor Design [20 points]

Circle the lines including terms that are compatible with each other and it makes sense for a processor design to include both.

- superscalar execution — in-order execution
- superscalar execution — out-of-order execution
- single-cycle machine — branch prediction
- reservation station — microprogramming
- fine-grained multithreading — single-core processor
- Tomasulo's algorithm — in-order execution
- precise exceptions — out-of-order instruction retirement
- branch prediction — fine-grained multithreading
- direct-mapped cache — LRU replacement policy
- fine-grained multithreading — pipelining

## 1.2 Pipelining [6 points]

What are the three major causes of pipeline stalls?

## 1.3 Caches I [5 points]

Please reason about the following statements about a possible processor cache one can design.

Can a cache be 5-way set associative?

YES NO

Explain your reasoning. Be concise. Show your work.

## 1.4 Caches II [10 points]

Assume a processor where instructions operate on 8-byte operands. An instruction is also encoded using 8 bytes. Assume that the designed processor implements a 16 kilo-byte, 4-way set associative cache that contains 1024 sets.

How effective is this cache? Explain your reasoning. Be concise. Show your work.

## 1.5 Performance Analysis [15 points]

A multi-cycle processor executes *arithmetic instructions* in **5 cycles**, *branch instructions* in **4 cycles** and *memory instructions* in **10 cycles**. You have a program where 30% of all instructions are arithmetic instructions, 35% of *all instructions* are memory instructions, and the rest are branch instructions. You figured out that the processor cannot execute the program fast enough to meet your performance goals. Your goal is to reduce the execution time of this program by at least 10%. Hence, you decide to change the processor design to improve the performance of **arithmetic instructions**.

In the new processor design, **at most** how many cycles should the execution of **a single arithmetic instruction** take to reduce the execution time of the *entire program* by **at least** 10%? Show your work.

## 1.6 Microprogrammed Design [4 points]

In lecture, we discussed a design principle for microprogrammed processors. We said that it is a good design principle to generate the control signals for cycle $N + 1$ in cycle $N$.

Why is this a good design principle? Be concise in your answer.

## 1.7 Processor Performance [10 points]

Assume that we test the performance of two processors, A and B, on a benchmark program. We find the following about each:

- Processor $A$ has a CPI of 2 and executes 4 Billion Instructions per Second.
- Processor $B$ has a CPI of 1 and executes 8 Billion Instructions per Second.

Which processor has higher performance on this program? Circle one.
Recall that CPI stands for Cycles Per Instruction.

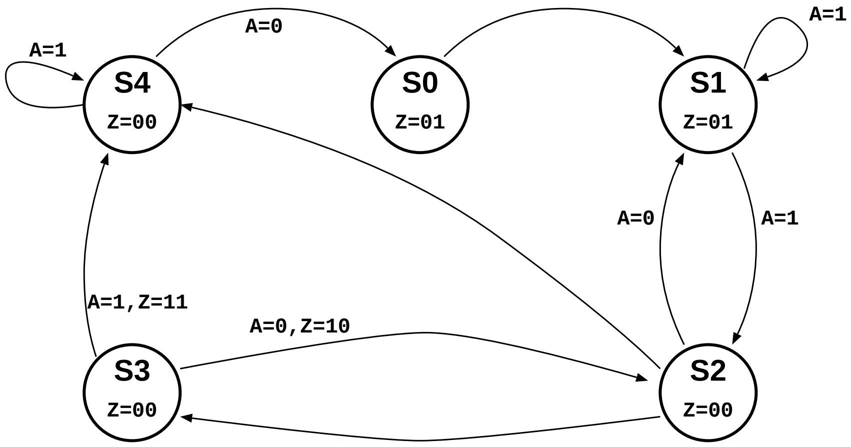
```
A. Processor A
B. Processor B
C. They have equal performance
D. Not enough information to tell
```

Explain concisely your answer in the box provided below. Show your work.

# 2 Finite State Machines

This question has three parts.

(a) [20 points] An engineer has designed a deterministic finite state machine with a one-bit input ($A$) and a two-bit output ($Z$). He started the design by drawing the following state transition diagram:



Although the exact functionality of the FSM is not known to you, there are **at least three mistakes** in this diagram. Please list **all** the mistakes.

(b) [25 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.

```verilog
module fsm (input CLK, RST, A, output [1:0] Z);

  reg [2:0] nextState, presentState;

  parameter start   = 3’b000;
  parameter flash1 = 3’b010;
  parameter flash2 = 3’b011;
  parameter prepare   = 3’b100;
  parameter recovery = 3’b110;
  parameter error = 3’b111;

  always @ (posedge CLK, posedge RST)
     if (RST)  presentState <= start;
     else      presentState <= nextState;

  assign Z = (presentState == recovery) ? 2’b11 :
             (presentState == error)    ? 2’b11 :
             (presentState == flash1)   ? 2’b01 :
             (presentState == flash2)   ? 2’b10 : 2’b00;

  always @ (presentState, A)
    case (presentState)
      start    : nextState <= prepare;
      prepare  : if (A) nextState <= flash1;
      flash1   : if (A) nextState <= flash2;
                 else   nextState <= recovery;
      flash2   : if (A) nextState <= flash1;
                 else   nextState <= recovery;
      recovery : if (A) nextState <= prepare;
                 else   nextState <= error;
      error    : if (~A) nextState <=start;
      default  : nextState <= presentState;
    endcase

endmodule
```

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

(c) [5 points] Is the FSM described by the previous Verilog code a Moore or a Mealy FSM? Why?

# 3 Verilog

Please answer the following four questions about Verilog.

(a) [10 points] Does the following code result in a sequential circuit or a combinational circuit? Please explain why.

```
module one (input clk, input a, input b, output reg [1:0] q);
 always @ (*)
   if (b)
      q <= 2'b01;
   else if (a)
      q <= 2'b10;
endmodule
```

Answer and concise explanation:

(b) [10 points] What is the value of the output `z` if the input `c` is `10101111` and `d` is `01010101`?

```
module two (input [7:0] c, input [7:0] d, output reg [7:0] z);
always @ (c,d)
  begin
    z = 8'b00000001;
    z[7:5] = c[5:3];
    z[4] = d[7];
    z[3] = d[7];
  end
endmodule
```

Please answer below. Show your work.

(c) [10 points] Is the following code correct? If not, please explain the mistake and how to fix it.

```
module mux2 ( input [1:0] i, input sel, output z);
  assign z= (sel) ? i[1]:i[0];
endmodule

module three ( input [3:0] data, input sel1, input sel2, output z);

  wire m;

  mux2 i0 (.i(data[1:0]), .sel(sel1), .z(m[0]) );
  mux2 i1 (.i(data[3:2]), .sel(sel1), .z(m[1]) );
  mux2 i2 (.i(m), .sel(sel2), .z(z) );

endmodule
```

Answer and concise explanation:

(d) [10 points] Does the following code correctly implement a multiplexer?

```
module four (input sel, input [1:0] data, output reg z);
  always@(sel)
  begin
    if(sel == 1’b0)
      z = data[0];
    else
      z = data[1];
  end
endmodule
```

Answer and concise explanation:

# 4 Boolean Logic and Truth Tables

You will be asked to derive the Boolean Equations for two 4-input logic functions, X and Y. Please use the Truth Table below for the following three questions.

| **Inputs** | | | | **Outputs** | |
|---|---|---|---|---|---|
| $A_3$ | $A_2$ | $A_1$ | $A_0$ | $X$ | $Y$ |
| 0 | 0 | 0 | 0 | | |
| 0 | 0 | 0 | 1 | | |
| 0 | 0 | 1 | 0 | | |
| 0 | 0 | 1 | 1 | | |
| 0 | 1 | 0 | 0 | | |
| 0 | 1 | 0 | 1 | | |
| 0 | 1 | 1 | 0 | | |
| 0 | 1 | 1 | 1 | | |
| 1 | 0 | 0 | 0 | | |
| 1 | 0 | 0 | 1 | | |
| 1 | 0 | 1 | 0 | | |
| 1 | 0 | 1 | 1 | | |
| 1 | 1 | 0 | 0 | | |
| 1 | 1 | 0 | 1 | | |
| 1 | 1 | 1 | 0 | | |
| 1 | 1 | 1 | 1 | | |

(a) [15 points] The output $X$ is *one* when the input does **not** contain 3 consecutive 1's in the word $A_3, A_2, A_1, A_0$. The output $X$ is *zero*, otherwise. **Fill in the truth table on the previous page** and **write the Boolean equation in the box below** for $X$ using the *Sum of Products* form. (*No simplification needed.*)

(b) [15 points] The output $Y$ is *one* when no two adjacent bits in the word $A_3, A_2, A_1, A_0$ are the same (e.g., if $A_2$ is 0 then $A_3$ and $A_1$ cannot be 0). The output $Y$ is *zero*, otherwise (for example 0000). **Fill in the truth table on the previous page** and **write the Boolean equation in the box below** for $Y$ using the *Sum of Products* form. (*No simplification needed.*)

(c) [10 points] Please represent the circuit of $Y$ using only 2-input XOR and AND gates.

# 5 Tomasulo's Algorithm

In this problem, we consider an in-order fetch, out-of-order dispatch, and in-order retirement execution engine that employs Tomasulo's algorithm. This engine behaves as follows:

- The engine has four main pipeline stages: Fetch (F), Decode (D), Execute (E), and Write-back (W).
- The engine can fetch one instruction per cycle, decode one instruction per cycle, and write back the result of one instruction per cycle.
- The engine has two execution units: 1) an adder for executing ADD instructions and 2) a multiplier for executing MUL instructions.
- The execution units are fully pipelined. The adder has two stages (E1-E2) and the multiplier has four stages (E1-E2-E3-E4). Execution of each stage takes one cycle.
- The adder has a two-entry reservation station and the multiplier has a four-entry reservation station.
- An instruction always allocates the first available entry of the reservation station (in top-to-bottom order) of the corresponding execution unit.
- Full data forwarding is available, i.e., during the last cycle of the E stage, the tags and data are broadcast to the reservation station and the Register Alias Table (RAT). For example, an ADD instruction updates the reservation station entries of the dependent instructions in E2 stage. So, the updated value can be read from the reservation station entry in the next cycle. Therefore, a dependent instruction can potentially begin its execution in the next cycle (after E2).
- The multiplier and adder have separate output data buses, which allow both the adder and the multiplier to update the reservation station and the RAT in the same cycle.
- An instruction continues to occupy a reservation station slot until it finishes the Write-back (W) stage. The reservation station entry is deallocated after the Write-back (W) stage.

## 5.1 Problem Definition

The processor is about to fetch and execute *six* instructions. Assume the *reservation stations (RS)* are all initially empty and the initial state of the *register alias table (RAT)* is given below in Figure (a). Instructions are fetched, decoded and executed as discussed in class. At some point during the execution of the six instructions, a snapshot of the state of the RS and the RAT is taken. Figures (b) and (c) show the state of the RS and the RAT at the snapshot time. A dash (–) indicates that a value has been cleared. A question mark (?) indicates that a value is unknown.

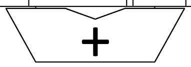
| Reg | Valid | Tag | Value |
|---|---|---|---|
| R0 | 1 | – | 1900 |
| R1 | 1 | – | 82 |
| R2 | 1 | – | 1 |
| R3 | 1 | – | 3 |
| R4 | 1 | – | 10 |
| R5 | 1 | – | 5 |
| R6 | 1 | – | 23 |
| R7 | 1 | – | 35 |
| R8 | 1 | – | 61 |
| R9 | 1 | – | 4 |

(a) Initial state of the RAT

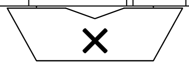
| Reg | Valid | Tag | Value |
|---|---|---|---|
| R0 | 1 | ? | 1900 |
| R1 | 0 | Z | ? |
| R2 | 1 | ? | 12 |
| R3 | 1 | ? | 3 |
| R4 | 1 | ? | 10 |
| R5 | 0 | B | ? |
| R6 | 1 | ? | 23 |
| R7 | 0 | H | ? |
| R8 | 1 | ? | 350 |
| R9 | 0 | A | ? |

(b) State of the RAT at the snapshot time

| ID | V | Tag | Value | V | Tag | Value |
|---|---|---|---|---|---|---|
| A | 1 | ? | 350 | 1 | ? | 12 |
| B | 0 | A | ? | 0 | Z | ? |



| ID | V | Tag | Value | V | Tag | Value |
|---|---|---|---|---|---|---|
| – | – | – | – | – | – | – |
| T | 1 | ? | 10 | 1 | ? | 35 |
| H | 1 | ? | 35 | 0 | A | ? |
| Z | 1 | ? | 82 | 0 | H | ? |

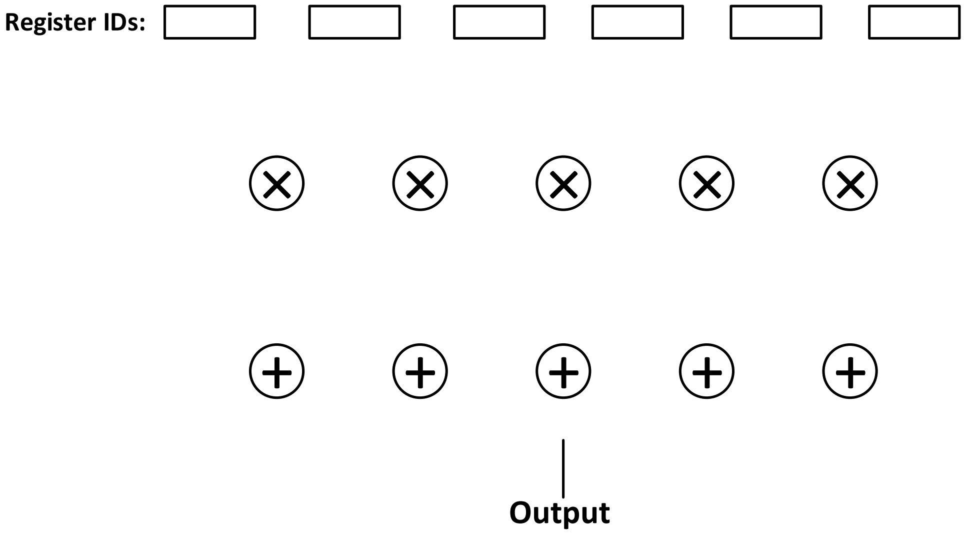


(c) State of the RS at the snapshot time

## 5.2 Questions

### 5.2.1 Data Flow Graph [40 points]

Based on the information provided above, identify the instructions and complete the dataflow graph below for the six instructions that have been fetched. Please appropriately connect the nodes using edges and specify the direction of each edge. Label each edge with the destination architectural register and the corresponding Tag. *Note that you may **not** need to use all registers and/or nodes provided below.*



### 5.2.2 Program Instructions [20 points]

Fill in the blanks below with the six-instruction sequence in program order. When referring to registers, please use their architectural names (R0 through R9). Place the register with the smaller architectural name on the left source register box.
For example, `ADD R8` ⇐ `R1, R5`.

| | | | | | |
|---|---|---|---|---|---|
| | | ⇐ | | , | |
| | | ⇐ | | , | |
| | | ⇐ | | , | |
| | | ⇐ | | , | |
| | | ⇐ | | , | |
| | | ⇐ | | , | |

# 6 GPUs and SIMD

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program. As we saw in lecture and practice exercises, the SIMD utilization of a program is computed across the *complete run* of the program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A, B, and C are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 6 instructions in each thread.) A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU. Please assume that all values in array B have magnitudes less than 10 (i.e., $|B[i]| < 10$, for all i).

```
for (i = 0; i < 1024; i++) {
    A[i] = B[i] * B[i];
    if (A[i] > 0) {
        C[i] = A[i] * B[i];
        if (C[i] < 0) {
            A[i] = A[i] + 1;
        }
        A[i] = A[i] - 2;
    }
}
```

Please answer the following five questions.

(a) [5 points] How many warps does it take to execute this program?

(b) [5 points] What is the maximum possible SIMD utilization of this program?

(c) [20 points] Please describe what needs to be true about array B to reach the maximum possible SIMD utilization asked in part (b). (Please cover all cases in your answer)

B:

(d) [10 points] What is the minimum possible SIMD utilization of this program?

(e) [20 points] Please describe what needs to be true about array B to reach the minimum possible SIMD utilization asked in part (d). (Please cover all cases in your answer)

B: