UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

ECE253F – Digital and Computer Systems
Final Examination

December 18, 2023 9:30am - 12:00pm
Duration: 150 minutes

Examiners: Profs. N. Enright Jerger and M. Jeffrey

Please enter your name and student number in the spaces provided above as it appears on Quercus. It is important that your name exactly match the Quercus gradebook.

Exam Type D: Examiner specified aids: One single sheet of letter size paper (8.5 x 11 inch), both sides may be used. Can be handwritten, typed or mechanically reproduced. Rulers are permitted.

Calculator Type 4: No calculators or other electronic devices are allowed.

All questions are to be answered on the examination paper. Your answer **MUST** be fully contained on the same page as the question. **Any material written on the back of each page will be ignored.** Exams will be scanned – please write clearly in **pen or dark pencil**.

Please state any assumptions you make when answering a question.

The final two pages contains some reference material that may be useful.

The number of marks for each question are indicated. The exam has **26 pages**, including this one.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Total
5	4	15	10	16	12	12	5	12	16	107

Question 1 [5 Marks]

[3 marks] (a) Convert the following decimal numbers to **6-bit two's complement binary numbers** and add them; write the two's complement sum in the box provided. Indicate whether overflow occurs when the sum is limited to 6-bits.

(a) $(16)_{10}$ + $(9)_{10}$

$(16)_{10}$ in 2's complement **010000**

$(9)_{10}$ in 2's complement **001001**

Sum (in 2's complement): **011001**

Overflow (Y/N): **N**

(b) $(-19)_{10}$ + $(-22)_{10}$

$(-19)_{10}$ in 2's complement **101101**

$(-22)_{10}$ in 2's complement **101010**

Sum (in 2's complement): **1010111**

Overflow (Y/N): **Y**

(c) $(-26)_{10}$ + $(8)_{10}$

$(-26)_{10}$ in 2's complement **100110**

$(8)_{10}$ in 2's complement **001000**

Sum (in 2's complement): **101110**

Overflow (Y/N): **N**

[2 marks] (b) In class, you learned how to convert between decimal (base 10), binary (base 2) and hexadecimal (base 16) for unsigned numbers. Using the methods taught to you, this question asks you to convert to/from base 5.

(a) Convert $(17)_{10}$ to Base 5

$(32)_5$

(b) Convert $(23)_5$ to decimal

$(13)_{10}$

Question 2 [4 Marks]

[4 marks] Use Boolean algebra simplify the following expression:

$$f = \overline{x + \bar{x}y + \bar{x}\bar{y}} + \overline{x + \bar{y}}$$

For full marks, show your work and state the theorems used. You may use the numbers provided on the reference sheet on page 25 to refer to the theorems or you may write the simplified form (e.g., $x + x = x$).

f = $\overline{x + \bar{x}} + \overline{x + \bar{y}}$ 14

$\overline{1} + \overline{x + \bar{y}}$ 8

$\bar{x}\bar{\bar{y}}$ 15

$\bar{x}y$ 9

Question 3 [15 Marks]

RISC-V's Shift Right Arithmetic instruction (`sra`) is a faster alternative to division by a positive power of two. In this question, you must design a combinational logic circuit to implement a smaller version of `sra`. The circuit takes two binary numbers as input: 3-bit x (in two's complement) and 2-bit y (unsigned). The 3-bit output, f, is equal to x shifted right by y bits. The circuit uses sign extension: fill the most significant bits of f with the sign bit of x. In `C`, this is written `f = x >> y`.

[4 marks] (a) Complete the truth table for the Shift Right Arithmetic circuit. ('X' means "don't care").

x_2	x_1	x_0	y_1	y_0	f_2	f_1	f_0
0	0	0	X	X	0	0	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	0
0	0	1	1	0	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	0	0	1	0
0	1	0	0	1	0	0	1
0	1	0	1	0	0	0	0
0	1	0	1	1	0	0	0
0	1	1	0	0	0	1	1
0	1	1	0	1	0	0	1
0	1	1	1	0	0	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	1	1	1	0
1	0	0	1	0	1	1	1
1	0	0	1	1	1	1	1
1	0	1	0	0	1	0	1
1	0	1	0	1	1	1	0
1	0	1	1	0	1	1	1
1	0	1	1	1	1	1	1
1	1	0	0	0	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	0	1	1	1
1	1	0	1	1	1	1	1
1	1	1	X	X	1	1	1

[8 marks] (b) Derive the minimized boolean logic equations for f_2, f_1, and f_0 in sum-of-products form. (Hint: 4-variable k-maps could be useful for f_1 and f_0. Does f_1 depend on x_0? For f_0, you could write a separate 4-variable k-map for each of $x_2 = 0$ and $x_2 = 1$.)

$f_2 = x_2$
$f_1 = x_2 y_1 + x_2 y_0 + x_1 \bar{y_0} \bar{y_1}$
$f_0 = x_2 y_1 + x_0 \bar{y_0} \bar{y_1} + x_1 y_0 \bar{y_1}$

[3 marks] (c) Oh no! Your instructor's version of ModelSim does not support the >> nor >>> operators. Complete the following System Verilog module to implement the described `sra` circuit without using those operators.

```
module sra(input logic [2:0] x, input logic [1:0] y, output logic [2:0] f);


  // A direct translation from the previous page:
  assign f[2] = x[2];
  assign f[1] = x[1] & ~y[1] & ~y[0]
              | x[2] & x[1]
              | x[2] & y[1]
              | x[2] & y[0];
  assign f[0] = x[2] & y[1] | ~y[1] & ~y[0] & x[0] | ~y[1] & y[0] & x[1];

  // Alternatively one can use concatenation and replication
  always_comb
    case(y)
      0 : f = x;
      1 : f = {1{x[2]},x[2:1]};
      2 : f = {2{x[2]},x[2]};
      3 : f = {3{x[2]}};
      default: f = 3'b111;
    endcase

endmodule
```

Question 4 [10 Marks]

Consider a spelling/grammar checker that will correct simple errors. In this question, you are to specify a finite state machine (FSM) that will capitalize the personal pronoun I in certain instances if it is entered as a lower case i.

For example, 'i know i'll pass this exam' will be corrected to 'I know I'll pass this exam'.

Input to your FSM will be any sequence of characters from a standard keyboard. Your job is to replace the i with an I if

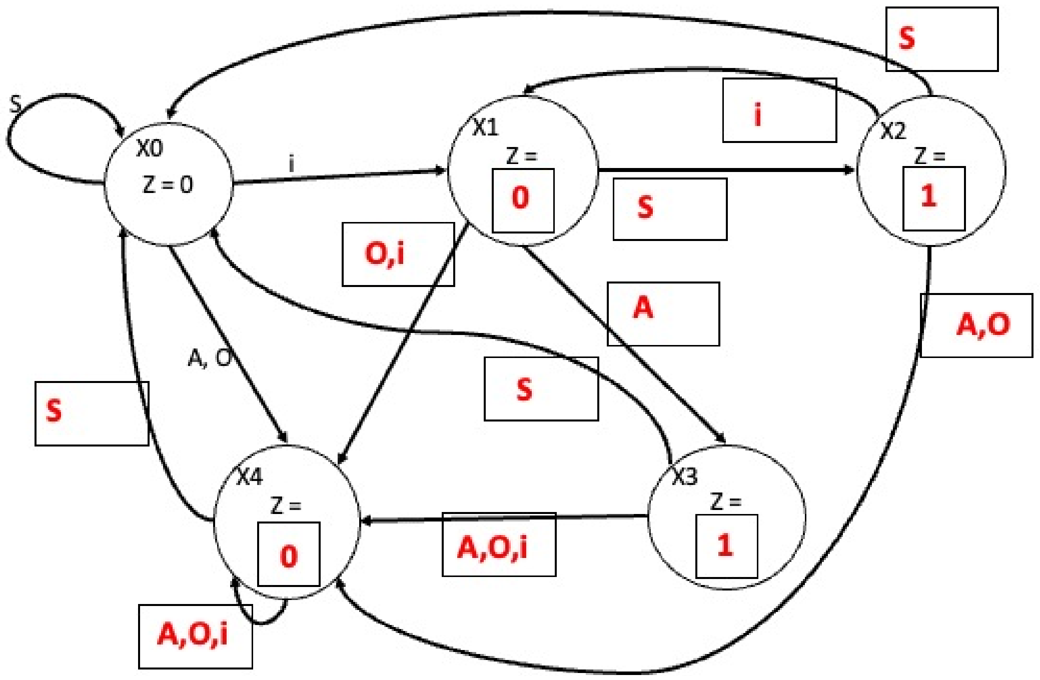
- the i is the first character input or is preceded by a **space**, and
- the i is followed by a **space** or by an **apostrophe (')**.

Shown below is an FSM with some of the inputs and some of the outputs unspecified. Your job is to complete the specification by filling in the boxes. States are labelled as X0-X4, with X0 being the initial state.

Inputs are from the set i, A, S, O, where A represents an apostrophe ('), S represents a space, O represents any character other than i, apostrophe or space. Note: in a real system, inputs would be mapped to binary values; for the purposes of this question, you can specify the inputs as i, A, S, O as already given on some of the arrows in the diagram.

The output Z corresponding to each state is 0 or 1, where 0 means "do nothing" and 1 means "change the most recent i to an I."

Fill in the boxes with the missing information.



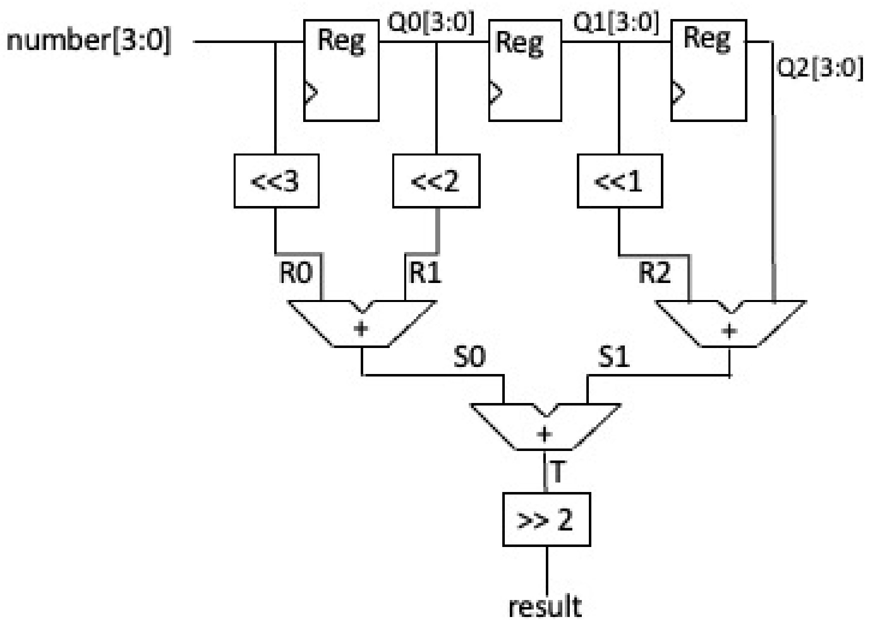
Question 5 [16 Marks]

In this question, you must design a circuit to calculate the *weighted average* of a series of numbers. The weighted average of 4 numbers (x0 - x3) is calculated as:

$WeightedAverage = floor(\frac{8\times X_0+4\times X_1+2\times X_2+X_3}{4})$

The circuit in the figure below implements this functionality. The circuit takes 4-bit unsigned numbers as inputs and outputs the weighted average every cycle. The multiplications by 2, 4, and 8 are implemented using left shift operations. Similarly, the division by 4 is implemented using a right shift.

Each instance of 'Reg' represents 4-bit registers each consisting of 4 D-Flip Flops. For clarity, clock and reset are not shown. However, all the flip-flops use the same clock and they all use an active-high synchronous reset.



[4 marks] (a) Fill in the table provided to indicate the width (number of bits) for each of the labelled wires in the figure.

Signal	R0	R1	R2	S0	S1	T	Result
Width	7	6	5	8	6	8	6

[8 marks] (b) Complete the following System Verilog code to implement the circuit from part (a).

```
module mov_avg(clock, reset, number, result);
   input logic clock, reset;
   input logic [3:0] number;
   logic [3:0] Q0, Q1, Q2;

// Assume that signals R0, R1, R2, S0, S1, T and result are created here,
// with the widths you specified in Part (a) above.

   always_ff @ (  posedge clock              ) // Fill in parenthesis

   begin
   // Add necessary code here

      if (reset) begin
         Q0 <= 4'b0000;
         Q1 <= 4'b0000;
         Q2 <= 4'b0000;
      end
      else begin
         Q0 <= number;
         Q1 <= Q0;
         Q2 <= Q1'
      end

   end

   always_comb
   begin
   // Add necessary code here

  R0 = number << 3;
  R1 = Q0 << 2;
  R2 = Q1 << 1;
  S0 = R0 + R1;
  S1 = R2 + Q2;
  T = S0 + S1;
  result = T >> 2;

   end
endmodule
```

[2 marks] (c) You must now write a test case in a DO file to test your circuit. Naturally, you want to write as little as possible, so you must write a single test case to test your design. Think about what single test case will give you the most confidence that your design is working correctly. For full marks, you must use the fewest lines possible.

```
    Vlib work
    Vlog movavg.sv
    Vsim movavg
    log {/*}
    add wave {/*}

    force clock 0 0ns, 1 5ns -r 10ns

    force reset 1
    run 10ns

    force reset 0
    run 10ns

# Add your test case here


force number 1111
run 40ns
```

[2 marks] (d) Briefly justify the test case selection from part (c) here:

Test max values to ensure signals have correct width. Run for 4 clock periods to see first full result.

Question 6 [12 Marks]

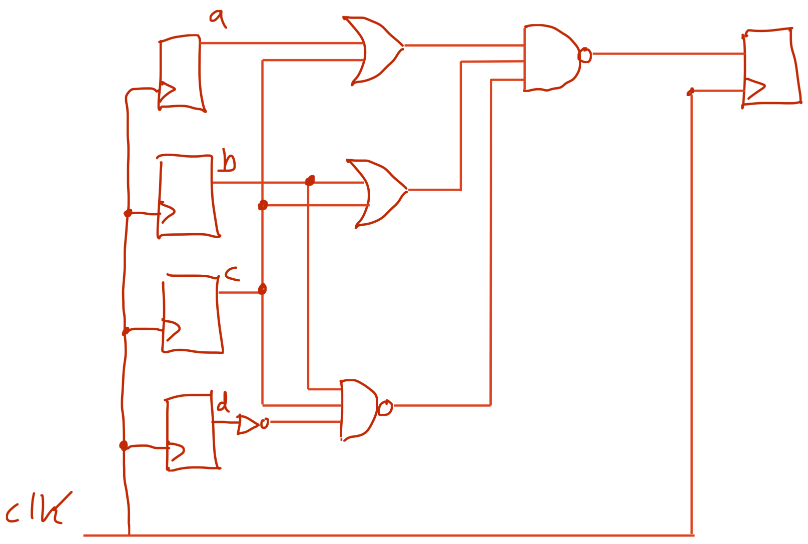
Consider the following System Verilog code:

```
module q6(input logic clk, in1, in2, in3, in4, output logic out);
  logic a, b, c, d, x;

  always_ff @(posedge clk)
  begin
    a <= in1;
    b <= in2;
    c <= in3;
    d <= in4;
  end

  assign x = (~a & b & ~c)
           | (~a & ~b & ~c)
           | (a & ~b & ~c)
           | (a & b & c & ~d)
           | (~a & b & c & ~d);

  always_ff @(posedge clk)
    out <= x;
endmodule
```

The table below lists the **only** gates and flip-flop that may be used in this question along with their timing values. (You may use as many of these as you need).

Positive edge-triggered D-type flip flop	t_{su}	0.5ns
	t_{cq}	1 ns
	t_{hold}	0.7ns
Inverter	t_{inv}	0.5 ns
NAND gate	t_{nand}	0.5 ns + 0.2ns $\times$ number of inputs
AND gate	t_{and}	1 ns + 0.2ns$\times$ number of inputs
OR gate	t_{or}	1 ns + 0.2ns $\times$ number of inputs

[7 marks] (a) Draw a schematic of a circuit that implements this code **that runs at the highest possible clock frequency.** You may alter the boolean algebra as long as it is logically equivalent to the given code. (Hint: do you remember bubble pushing and De Morgan's theorem?)



[3 marks] (b) Assuming there is no clock skew, what is the maximum operating frequency at which your circuit will operate correctly?

$$\begin{aligned} T_{min} &= t_{CQ} + t_{logic} + t_{su} \\ &= 1 + 0.5 + 0.5 + 0.2 \times 3 + 0.5 + 0.2 \times 3 + 0.5 \\ &= 4.2ns \end{aligned}$$

$$F_{max} = 1/4.2ns$$

[2 marks] (c) What is the value for t_{skew} at which a hold time violation will occur in your circuit?

Hold time violations are only relevant between each of the left hand FFs (`a,b,c,d`) and the right hand FF. The t_{logic_min} is from b through both 3-input NANDs (2.2ns) rather than from b through the 2-input OR then the 3-input NAND (2.5ns).

To avoid a hold-time violation,

$$\begin{aligned} t_h + t_{skew} &\leq t_{CQ} + t_{logic\_min} \\ 0.7 + t_{skew} &\leq 1 + 2.2 \\ t_{skew} &\leq 2.5 \end{aligned}$$

A hold-time violation occurs with $t_{skew} > 2.5ns$

Question 7 [12 Marks]

Consider the following RISC-V assembly language code with some memory addresses shown for aid in answering the question.

```
   .data
0x00000100: LIST: .byte 0x38, 0xF8, 0x17
0x00000103: LENGTH: .byte 3

   .text
   .global _start
            _start:
0x00000200: addi s0, zero, 1
0x00000204: la s1, LENGTH
0x00000208: lb s1, 0(s1)
            la s2, LIST
       ABC: bge s0, s1, DONE
            add s3, zero, s0
            add s4, s2, s3
       DEF: blez s3, GHI
            lb s5, -1(s4)
            lb s6, 0(s4)
0x00000228: ble s5, s6, GHI
            sb s5, 0(s4)
            sb s6, -1(s4)
            addi s3, s3, -1
            add s4, s2, s3
            j DEF
       GHI: addi s0, s0, 1
            j ABC
      DONE: ebreak
```

[6 marks] (a) Suppose this program is executed on a RISC-V processor. What are the values of the RISC-V registers shown below when the code reaches the ebreak instruction? Provide numeric values and specify the number format.

s0	3 (decimal)	s1	3 (decimal)	s2	0x00000100
s3	1 (decimal)	s4	0x00000101	s5	0xFFFFFFF8

[2 marks] (b) What are the values in memory at the following addresses when the code reaches the ebreak instruction. Provide numeric values and specify the number format.

Memory address	Value of byte in memory
0x00000100	0xF8
0x00000101	0x17
0x00000102	0x38
0x00000103	0x03

[2 marks] (c) Using ten words or less, what does this code "do" at a high level?

Sorts a list

[2 marks] (d) What would happen to the result of executing this code if the instruction at line 0x00000228 was changed from 'ble s5, s6, GHI' to 'bleu s5, s6, GHI'?

0xF8 would be interpretted as a positive number and the sorted list order would be 0x17, 0x38, 0xF8

Question 8 [5 Marks]

The strlen function from the C standard library counts the number of characters in a character array (a.k.a., string) up until the first null character (with integer value of zero). The first argument to strlen holds the memory address of the first element (byte) of the character array. The following RISC-V assembly implementation of strlen has errors.

```
strlen: addi s1, zero, 0 # Initialize counter to 0
LOOP:   lb s2, 0(a0)     # Load the current character
        beqz s2, END     # Check if null character
        addi s1, s1, 1   # Increment the counter
        addi a0, a0, 1   # Advance the character address
        j LOOP
END:    addi a0, s1, 0   # Return the result in a0
        addi sp, sp, 4
        jr ra
```

Rewrite strlen in RISC-V assembly to retain its functionality but fix the errors.

```
strlen: # Start your code below here
```

```
        addi t1, zero, 0
LOOP:   lb t2, 0(a0)
        beqz t2, END
        addi t1, t1, 1
        addi a0, a0, 1
        j LOOP
END:    addi a0, t1, 0
        jr ra
```

or

```
        addi sp, sp, -8
        sw s1, 0(sp)
        sw s2, 4(sp)
        addi s1, zero, 0
LOOP:   lb s2, 0(a0)
        beqz s2, END
        addi s1, s1, 1
        addi a0, a0, 1
        j LOOP
END:    addi a0, s1, 0
        lw s2, 4(sp)
        lw s1, 0(sp)
        addi sp, sp, 8
        jr ra
```

Question 9 [12 Marks]

[8 marks] (a) Consider the following code. Memory addresses for the data and instructions are provided in the left hand column to aid in answering the question.

```
     .data
0x10010000:     MY_STRING1: .byte ’a’, ’b’, ’c’, 0
0x10010004:     MY_STRING2: .byte 0, 0, 0, 0

    .global _start
    .text
_start:
0x00400000:    la a0, MY_STRING1
0x00400004:    la a1, MY_STRING2
0x00400008:    jal my_recursive_func
0x0040000C: END: ebreak

      my_recursive_func:
0x00400010:    addi sp, sp, -4
0x00400014:    sw ra, 0(sp)
0x00400018:    lbu t1, 0(a0)
0x0040001C:    addi sp, sp, -4
0x00400020:    sw t1, 0(sp)
0x00400024:    beqz t1, return
0x00400028:    addi a0, a0, 1
0x0040002C:    jal my_recursive_func
0x00400030:    lw t1, 0(sp)
0x00400034:    addi sp, sp, 4
0x00400038:    sb t1, 0(a1)
0x0040003C:    addi a1, a1, 1
0x00400040:    lw ra, 0(sp)
0x00400044:    addi sp, sp, 4
0x00400048: return: jr ra
```

On the next page, fill in the contents of the stack the first time the instruction at address 0x00400048 executes. When the code starts, the stack pointer (sp) is initially equal to 0x7FFFEFFC. This initial value has been filled into the table for you. Your answer may not require all boxes provided for the stack. You should fill in both the memory address and the value. For ASCII values, given as ’a’, ’b’, etc in the code above, you can just use the ASCII representation – you **do not** need to convert from ASCII to hexadecimal.

Question 9 continued . . .

Memory Address	Value in Memory
0x7FFFEFDC	0
0x7FFFEFE0	0x00400030
0x7FFFEFE4	’c’
0x7FFFEFE8	0x00400030
0x7FFFEFEC	’b’
0x7FFFEFF0	0x00400030
0x7FFFEFF4	’a’
0x7FFFEFF8	0x0040000C
0x7FFFEFFC	

[2 marks] (b) Describe in ten words or less what this recursive function does.

reverses a string

[2 marks] (c) Consider you have written a recursive program. Your program and any associated data section require 256 bytes of memory. Every time your recursive function is called, it saves 4 words on to the stack. Your system only contains 1024 bytes of memory. If your code and data start at address 0 and your stack pointer is initialized to 0x3FC (0x3FC = 1020 in decimal). How many recursive calls can be made before the stack clobbers the code/data of your program?

47

Question 10 [16 Marks]

Write a RISC-V assembly language program that polls the keyboard and displays a 10-bit binary value on the LEDs, depending on the key that is pressed. The value should initially be 512 (0b1000000000). Pressing key 1 should cause the displayed value to *decrease* by 1. Pressing key 2 should cause the displayed value to *increase* by 1. If the displayed value is 0, pressing key 1 should have no effect. If the displayed value is 1023 (0b1111111111), pressing key 2 should have no effect. **Pressing any other key resets the value to 512.**

The ASCII code for ~~1 is 0x50, 2 is 0x51~~ 1 is 0x31 (49 in decimal), 2 is 0x32 (50 in decimal).

The I/Os have the following memory map information:

Keyboard

(a) The ASCII code for the key is written to the Receiver Data register. This register is located in the lowest 8-bits of memory location 0xffff0004).

(b) The Ready bit is set to 1 in the Receiver Control register, which is the least significant bit of memory location 0xffff0000. The Ready bit is automatically reset to 0 when you read the Receiver Data register using a `lw` instruction.

LEDs

(a) The system has 10 LEDs located at 0xffff8000. The least significant bit is LED0, the next bit is LED1, etc.

(b) A 1 is written to the corresponding LED to turn it on.

The following code is provided to get you started.

```
    .data
KEYBOARD: .word 0xFFFF0000
LEDS: .word 0xFFFF8000

    .text
    .global _start
_start:
  li s1, 512 # will hold current LED value
  li s2, 0xFFFF0000
  li t1, 0x50 # compare for key1
  li t2, 0x51 # compare for key2
  li t3, 0x3FF # compare for 1023
  li s5, 0xFFFF8000
  j UPDATE
POLL: lw s3, 0(s2)
  andi s3, s3, 1
  beqz s3, POLL
  lw s3, 4(s2) # ascii value of key
  beq s3, t1, DECR
  beq s3, t2, INCR
  li s1, 512 # reset display value to 0b1000000000
  j UPDATE
DECR:
  beqz s1, POLL # do not decrement below zero
  addi s1, s1, -1
  j UPDATE
INCR:
  beq s1, t3, POLL # do not increment above 1023
  addi s1, s1, 1
UPDATE:
  sw s1, 0(s5) # update LED value
  j POLL # resume polling
END: ebreak
```

Question 10 continued …

This page is intentionally left blank. You may use it as scratch to solve exam problems but please be sure to write your answers in the spaces provided for each question. You must submit this page with your exam.

This page is intentionally left blank. You may use it as scratch to solve exam problems but please be sure to write your answers in the spaces provided for each question. You must submit this page with your exam.

The last two pages provides some reference material that you may find useful during the exam. You may detach them from the exam, but please turn them in with your exam paper at the end of the exam.

Table 1: **Boolean Algebra**

1	$0 \cdot 0 = 0$	$1 + 1 = 1$
2	$1 \cdot 1 = 1$	$0 + 0 = 0$
3	$0 \cdot 1 = 1 \cdot 0 = 0$	$1 + 0 = 0 + 1 = 1$
4	if $x = 0, \bar{x} = 1$	if $x = 1, \bar{x} = 0$
5	$x \cdot 0 = 0$	$x + 1 = 1$
6	$x \cdot 1 = x$	$x + 0 = x$
7	$x \cdot x = x$	$x + x = x$
8	$x \cdot \bar{x} = 0$	$x + \bar{x} = 1$
9	$\bar{\bar{x}} = x$	
10	$x \cdot y = y \cdot x$	$x + y = y + x$
11	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	$x + (y + z) = (x + y) + z$
12	$x \cdot (y + z) = x \cdot y + x \cdot z$	$x + (y \cdot z) = (x + y) \cdot (x + z)$
13	$x + x \cdot y = x$	$x \cdot (x + y) = x$
14	$x \cdot y + x \cdot \bar{y} = x$	$(x + y) \cdot (x + \bar{y}) = x$
15	$\overline{xy} = \bar{x} + \bar{y}$	$\overline{(x + y)} = \bar{x} \cdot \bar{y}$
16	$x + \bar{x} \cdot y = x + y$	$x \cdot (\bar{x} + y) = x \cdot y$

Table 2: **Branch Instructions**

beq rs1, rs2, label	branch if $=$
bne rs1, rs2, label	branch if $\neq$
blt rs1, rs2, label	branch if $<$
bge rs1, rs2, label	branch if $\geq$
bne rs1, rs2, label	branch if $\neq$
bltu rs1, rs2, label	branch if $<$ unsigned
bgeu rs1, rs2, label	branch if $\geq$ unsigned
beqz rs1, label	branch if $= 0$
bnez rs1, label	branch if $\neq 0$
blez rs1, label	branch if ≤ 0
bgez rs1, label	branch if ≥ 0
bltz rs1, label	branch if < 0
bgtz rs1, label	branch if > 0
ble rs1, rs2, label	branch if $\leq$
bgt rs1, rs2, label	branch if $>$
bleu rs1, rs2, label	branch if $\leq$ unsigned
bgtu rs1, rs2, label	branch if $>$ unsigned

RISC-V Instruction reference

Instruction	Description	Instruction	Description
lb rd, imm(rs1)	load byte	slli rd, rs1, imm	shift left logical immediate
lh rd, imm(rs1)	load halfword	srli rd, rs1, imm	shift right logical immediate
lw rd, imm(rs1)	load word	sll rd, rs1, rs2	shift left logical
lbu rd, imm(rs1)	load byte unsigned	srl rd, rs1, rs2	shift right logical
lhu rd, imm(rs1)	load halfword unsigned	sra rd, rs1, rs2	shift right arithmetic
sb rs2, imm(rs1)	store byte	srai rd, rs1, imm	shift right arithmetic immediate
sh rs2, imm(rs1)	store halfword	ori rd, rs1, imm	or immediate
sw rs2, imm(rs1)	store word	andi rd, rs1, imm	and immediate
addi rd, rs1, imm	add immediate	or rd, rs1, rs2	or

add rd, rs1, rs2	add	xori rd, rs1, imm	xor immediate
sub rd, rs1, rs2	subtract	and rd, rs1, rs2	and
lui rd, imm	load upper immediate	xor rd, rs1, rs2	xor
la rd, label	load address of global variable	not rd, rs1	not
li rd, imm	load 32-bit immediate	mv rd, rs1	move
jal label	jump and link	csrrw rd, csr, rs1	CSR read/write
j label	jump	csrrs rd, csr, rs1	CSR read/set
jr rs1	jump register	csrrwi rd, csr, rs1	CSR read/write immediate
csrrsi rd, csr, imm	CSR read/set immediate		

RISC-V register names and numbers

Name	Register Number	Use
zero	x0	Constant Value 0
ra	x1	Return address
sp	x2	Stack pointer
t0-t2	x5-7	Temporary registers
s0-s1	x8-x9	Saved registers
a0-a1	x10-x11	Function arguments/return values
a2-a7	x12-x17	Function arguments
s2-s11	x18-x27	Saved registers
t3-t6	x28-x31	Temporary registers