University of Toronto
Faculty of Applied Science and Engineering

Final Exam
December 2014

ECE253 – Digital and Computer Systems

Examiner – Prof. Stephen Brown

Print:

First Name ______________________ Last Name ______________________

Student Number ______________________

1. There are **6** questions and **18** pages. Do **all** questions. The duration of the exam is 2.5 hours.
2. **ALL WORK IS TO BE DONE ON THESE SHEETS.** You can use the blank pages included at the end of the exam (Pages 15 – 17) if you need more space. Be sure to indicate clearly if your work continues elsewhere.
3. Closed book. One 2-sided hand-written aid sheet is permitted.
4. No calculators are permitted.

1 [~~15~~] 17	
2 [8]	
3 [8]	
4 [12]	
5 [10]	
6 [~~12~~] 10	
Total [65]	

1. Short answers:

[2 marks] (a) Perform the following additions of 2's complement numbers.

[1 mark each]

```
i.    10000000        ii.   11111111
      01111111              11111110
      10000001              11111101
      01111110              11111100
    + 00000010            + 00001010
      00000000              00000000
```

[1 mark] (b) For the numbers in part (a) are the results you calculated correct 2's complement sums, or not?

[0.5 mark each]

Answer for (a) i. Yes

Answer for (a) ii. Yes

[3 marks] (c) Consider the ARM code fragment shown below. When this code is being executed, an interrupt occurs when the ARM processor is executing the instruction ADD R1, R2, R3. Assume that interrupts are enabled, and that the interrupt is generated by the timer that you used in Lab Exercise 10. Assume the following values for ARM registers: R1 = 1, R2 = 2, R3 = 3.

```
          .text
          .global   _start

_start:   BL        somesubroutine
          LDR       R10, =0xFF200040
          LDR       R6, [R10]
          ADD       R1, R2, R3
          ...
```

The definition of an "interrupted instruction" changed in the lecture notes this year to say that a "current instruction" is **completed** instead of **aborted** if an interrupt occurs. We still accepted answers using the previous years' definition.

Fill in the values that the registers listed below will have when the ARM processor reaches, but has not yet executed, the branch instruction at the IRQ exception vector. Assume that the main program is stored in the memory starting at address 0x40.

[1 mark each] **PC** 0x18 **LR** 0x54 **R1** 0x5

Will also accept: 0x50 0x1

[2 marks] (d) In part (c) of this question, you were told that the main program is stored in the memory starting at address 0x40. Would it be okay if this main program were stored in the memory starting at address 0 instead? Explain your reasoning.

Answer No, it would overwrite the exception vector table.

1 mark for "vector table"

1 mark for "no" or "overwrite"

[3 marks] (e) Consider the ARM code fragment shown below. This code finds the greatest common divisor (GCD) of the two numbers found in registers R0 and R1.

```
GCD:    CMP     R0, R1
        BEQ     DONE
        BLT     LESS
        SUB     R0, R0, R1
        B       GCD
LESS:   SUB     R1, R1, R0
        B       GCD
DONE:   B       DONE
```

In this code only branch instructions make use of conditions. But the ARM processor supports conditional execution of most instructions, including SUB. In the space below re-write the above code that finds the GCD using conditional SUB instructions. Your goal is to use as few instructions as possible, while still implementing the same GCD algorithm.

Answer

```
GCD:    CMP   R0, R1
        SUBGT R0, R1
        SUBLT R1, R0
        BNE   GCD

DONE:   B     DONE
```

1 mark to collapse SUBGT
1 mark to collapse SUBLT
1 mark to collapse BNE

[2 marks] (f) Given functions f and g, below, for what values of the inputs a, b, and c does $f \neq g$?

$$f = \overline{a}\,\overline{c} + ac + \overline{a}\overline{b}$$

$$g = \overline{b}c + ab + \overline{a}\,\overline{c}$$

Answer: (a, b, c) = (1, 1, 0)

2 marks

-1 for each additional (incorrect) response

abc	f	g
000	1	1
001	1	1
010	1	1
011		
100		
101	1	1
110		1
111	1	1

[2 marks] (g) Use Boolean algebra to minimize the following function. Show your work and specify which identity is used in each of your steps.

Identity

$x\overline{z} + (xy + x)z$

Identity	
13a	$x\overline{z} + xz$
14a	x
	1 mark each

[2 marks] (h) Use Boolean algebra to minimize the following function. Show your work and specify which identity is used in each of your steps.

Identity

$\overline{b}c + ab + \overline{a}\,\overline{c} + \overline{a}\,bc$

Identity	
12a	$\overline{b}c + ab + \overline{a}(\overline{c} + bc)$
16a	$\overline{b}c + ab + \overline{a}(\overline{c} + b)$
12a	$\overline{b}c + ab + \overline{a}\overline{c} + \overline{a}b$
14a	$\overline{b}c + b + \overline{a}\overline{c}$
16a	$c + b + \overline{a}\overline{c}$
16a	$c + b + \overline{a}$
	Roughly:
	0.5 marks for first two steps 0.25 for each of last four steps

[8 marks] 2. Karnaugh maps:

(i)

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	1	0	0
10	0	1	0	0

(ii)

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	1	0	0	0
10	1	0	1	1

(iii)

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	1	1	0	d
01	0	1	1	0
11	0	0	1	1
10	d	0	1	1

(iv)

$x_3x_4 \backslash x_1x_2$	00	01	11	10
00	0	1	0	1
01	0	0	0	0
11	1	1	1	1
10	1	0	1	0

(a) For the function in Karnaugh map (i) above list **all minimal products-of-sums** solutions:

[2]

$(\overline{x_2} + x_3)(x_2 + \overline{x_3})(\overline{x_1} + \overline{x_3})$ 1 mark

$(\overline{x_2} + x_3)(x_2 + \overline{x_3})(\overline{x_1} + \overline{x_2})$ 1 mark -0.5 per error/extra

(b) For the function in Karnaugh map (ii) above list **all prime implicants**:

[2.5]

$(\overline{x_2}\,\overline{x_4}),\ (\overline{x_1}\,\overline{x_2}x_3),\ (\overline{x_1}\,\overline{x_3}\,\overline{x_4}),\ (x_1\overline{x_2}\,\overline{x_3}),\ (x_1x_3\overline{x_4})$

0.5 mark per term -0.5 per error/extra

(c) For the function in Karnaugh map (iii) above list **all minimal sum-of-products** solutions:

[1.5]

$(x_1x_3) + (x_2\overline{x_3}x_4) + (\overline{x_1}\,\overline{x_3}\,\overline{x_4})$

0.5 mark per term -0.5 per error/extra

(d) For the function in Karnaugh map (iv) above, let $g = x_1 \oplus x_2$. Fill in the logic expression below. Make the simplest expression you can, using g as indicated.

[2]

$f = \overline{x_3 \oplus x_4} \cdot (g) +$ x_3 $\cdot(\overline{g})$

or

$\overline{x_3}\,\overline{x_4} + x_3x_4$

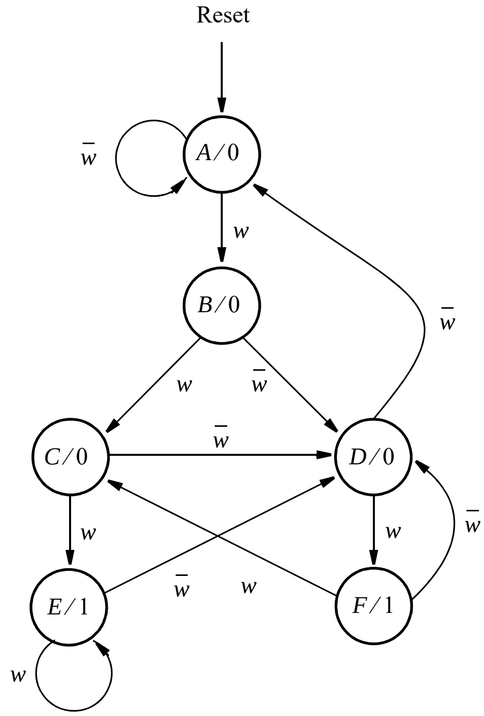
1 mark
or
0.5 each product term

1 mark

If two terms were swapped, 1 mark/2

3. Finite State Machines:

[8 marks] Consider the finite state machine below.



(a) What does this FSM "do"? That is, for what values of w does this FSM produce $z = 1$?

Answer:

1 mark Accepts a 111 or 101 sequence, overlaps allowed

(b) On the next page, write complete Verilog code for the above FSM. Assume that the FSM will be implemented on the DE1-SoC board, and connect the w input to switch SW_0, the reset input to KEY_0, and the clock signal to KEY_1. Display the current state on the *LEDR* outputs. Assign a state coding of your preference.

[7 marks] **Answer** for Verilog code for the FSM:

```
module FSM (
 input [1:0] KEY,
 input [0:0] SW,
 output [3:0] LEDR
);

 wire w = SW[0];
 wire reset = KEY[0];
 wire clk = KEY[1];
 wire z;

 parameter A=0,B=1,C=2,D=3,E=4,F=5;
 reg[2:0] state, next;

 always @(posedge clk)
  if (reset) state <= A;
  else state <= next;


 always@(*) begin
  case (state)
   A: next = w ? B : A;
   B: next = w ? C : D;
   C: next = w ? E : D;
   D: next = w ? F : A;
   E: next = w ? E : D;
   F: next = w ? C : D;
   default: next = 3'hx;
  endcase
 end

 assign z = (state == E) || (state == F);
 assign LEDR[3:0] = {z, state};

endmodule
```

1 mark: Port declarations/connections

2 marks: State flip-flops (1) with reset (1)

3 marks: State transitions

Quartus doesn't generate a latch even without a default case here. Omitting default case accepted.

We didn't ask for the output z to be connected. Oops.

1 mark: Output current state

[12 marks] 4. Trace an ARM Program:

Consider the ARM code shown below. Note that the address that each instruction would have in the memory is shown to the left of the code.

```
                        .text
                        .global   _start
            _start:
00000000                LDR       SP, 0x20000

00000004                LDR       R4, =N
00000008                LDR       R0, [R4], #4
0000000C                BL        HELLO

00000010                STR       R0, [R4]

00000014    END:        B         END              /* wait here */

00000018    HELLO:      PUSH      {R4, LR}
0000001C                MOV       R4, R0

00000020                CMP       R0, #1
00000024                BEQ       GOODBYE

00000028                SUB       R0, #1
0000002C                BL        HELLO

00000030                MUL       R0, R4, R0

00000034    GOODBYE:    POP       {R4, PC}

            N:          .word     3
            F:          .space    4

                        .end
```

(a) What does this code "do"? That is, for a given value of N, what result is stored in F?

Answer Computes factorial of value located at address N.

1 mark: Factorial
1 mark: Value came from address N

(b) If this program is executed on the ARM processor, what would be the values of the ARM registers shown below the **first** time the code reaches, but has not yet executed, the instruction at address 0x30. Also, show in the space below the contents of the stack in memory at this point in time (fill in the memory addresses on the left, and show the data stored in each location). For memory values that are not known, if any, write N/A in the corresponding box.

[3 marks]: 0.5 marks each

R0	1	R1	N/A	R4	2
R13	0x1FFF0	R14	0x30	R15	0x30

[7 marks]

Memory Address	Content		
	N/A		
	N/A	-1 for extra stuff	
1 mark: addresses	N/A		
1FFE8	0x2	1 mark	
1FFEC	0x30	1 mark	
1FFF0	0x3	1 mark	-1 if swapped {R4, LR} pairs
1FFF4	0x30	1 mark	
1FFF8	0x3C	1 mark	
1FFFC	0x10	1 mark	
20000	N/A	-1 if this box is missing and everything else shifted down	

5. Assembly Language Subroutine:

[10 marks] Consider the ARM *unsigned divide* assembly language instruction

UDIV Rd, Rn, Rm

The result is defined as the integer operation $Rd = Rn/Rm$. Since the *UDIV* instuction isn't implemented in the ARM A9 processor used in this course, you are to write a subroutine, called DIVIDE, that performs a division operation. Your DIVIDE subroutine should implement the equivalent of the instruction "UDIV R0, R0, R1". Thus, the inputs to DIVIDE should be passed to the subroutine in registers $R0$ and $R1$, and the result should be returned in register $R0$. The result should have the integer quotient of $R0/R1$ in the least-significant halfword of $R0$, and the remainder should be in the most-significant halfword of $R0$. Assume that $R0$ and $R1$ contain unsigned values, and that $R1 \neq 0$.

You need to provide a main program that calls your DIVIDE subroutine. Part of this main program is shown below; fill in the rest of the code. The values of arguments N and M that are used to produce the result $D = N/M$ are stored in memory as shown in the code—your main program needs to load these values from memory and pass them to the subroutine. After the subroutine returns, your main program should store the result into memory location *D*. Write the code for DIVIDE in the space on the next page. Include meaningful comments in your code!

```
            .text
            .global   _start
_start:

[3 marks]
                                      LDR  R4, =N
2 marks loading values                LDR  R0, [R4], #4 // R0 = N
           from memory                LDR  R1, [R4], #4 // R1 = M

0.5 function call                     BL   DIVIDE
0.5 storing result to memory          STR  R0, [R4]

END:        B         END              # wait here

            .data
N:          .word     10
M:          .word     3
D:          .space    4
            .end
```

Answer space for the DIVIDE subroutine.

```
        .global    DIVIDE
DIVIDE:
```

[5 marks]

```
        MOV   R3, #0   0.5: initialization
D_LOOP: CMP   R0, R1
        BLT   FINISH   2: compare and branch
        ADD   R3, #1        // increment quotient
        SUBS  R0, R1        // N = N - M   2: subtract and increment
        B     D_LOOP   0.5: loop branch
```

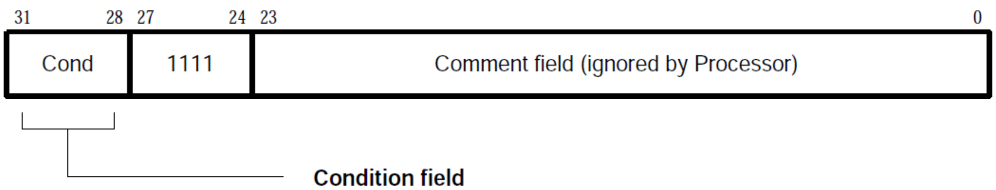
[2 marks]

```
FINISH: LSL   R0, #16       // R3 has quotient, R0 has remainder
        ORR   R0, R3        // form the final result
                  1: Packing result into one 4-byte word
        MOV   PC, LR      1: return from function
```

6. Exceptions:

[10 marks] In Question 5 you called your DIVIDE subroutine to implement unsigned division. Your DIVIDE subroutine is equivalent to the exact instruction "UDIV R0, R0, R1". If the operand registers are not exactly $R0$, $R0$, $R1$, then your DIVIDE subroutine can't be used. For this question you are to create a somewhat more general unsigned division operation that will work for any combination of registers $R0 - R12$. For example, it has to work with "UDIV R6, R4, R5", or "UDIV R8, R6, R7", or "UDIV R0, R1, R2", and so on.

You will implement the unsigned division operation by using the ARM SVC instruction. As discussed in lectures, when the SVC instruction is executed the ARM processor takes the SVC exception. The SVC instruction takes an immediate argument (maximum of 26 bits), and the assembler includes that argument in the machine code for the instruction. The machine-code format of the SVC instruction is shown below.



For example, the instruction "SVC #0" produces the machine code EF000000, and the instruction "SVC #FF" produces the machine code EF0000FF. In this question we use the SVC argument to specify which registers operands are involved. For example, "UDIV R6, R4, R5" is specified as "SVC 0b011001000101", and "UDIV R8, R6, R7" is specified as "SVC 0b100001100111".

You have to write an SVC exception handler. It has to examine the machine code that caused the exception to determine what register operands are involved. Assume that only registers in the range $R0$ to $R12$ will be used, and make the same assumptions as for Question 5 regarding the division operation ($Rd = Rn/Rm$, unsigned integers, argument $Rm \neq 0$, quotient and remainder in *Rd*).

An exception vector table and a main program that invokes two examples of the "UDIV" instruction is given on the next page. Complete the code shown for the SVC exception handler. Think about the comment that is shown in the code—it provides a *hint* about how your exception handler code can access the contents of registers *Rn*, *Rn*, and *Rd*. Include **meaningful comments** in your code!

```
                .section .vectors, "ax"

                B           _start                  // reset vector
                .word       0                       // undefined instruction vector
                B           SERVICE_SVC             // software interrrupt vector
                .word       0                       // aborted prefetch vector
                .word       0                       // aborted data vector
                .word       0                       // unused vector
                .word       0                       // IRQ interrupt vector
                .word       0                       // FIQ interrupt vector

                .text
                .global     _start
_start:
                // Main program
                MOV         SP, #0x20000            // initialize stack pointer
                MOV         R4, #10
                MOV         R5, #3
                SVC         0b011001000101          // execute "UDIV R6, R4, R5"
                // result is in R6
                MOV         R6, #20
                MOV         R7, #6
                SVC         0b100001100111          // execute "UDIV R8, R6, R7"
                // result is in R8
END:            B           END                     # wait here

                .end
```

Complete the *SERVICE_SVC* code on the following page.

```
        .global     SERVICE_SVC
SERVICE_SVC:  PUSH   {R0-R12}     // After executing this instruction, the contents of
                                  // registers specified for the "UDIV" operation can be
                                  // gotten from the stack. R0 is at the top of the stack,
                                  // R1 below that, . . ., and R12 is at the bottom of the stack
```

```
    LDR  R3, [LR, #-4]    // read the SVC ("DIV") machine code
    LSR  R2, R3, #8
    AND  R2, #0b1111      // R2 has index of Rd
    LSR  R0, R3, #4
    AND  R0, #0b1111      // R0 has index of Rn
    AND  R1, R3, #0b1111  // R1 has index of Rm

    LSL  R0, #2     // convert index N to word amount (x 4)
    LSL  R1, #2     // convert index M to word amount (x 4)
    LSL  R2, #2     // convert index D to word amount (x 4)

    LDR  R0, [SP, R0]   // get argument N
    LDR  R1, [SP, R1]   // get argument M

    MOV  R3, #0
D_LOOP:  CMP  R0, R1
    BLT  FINISH
    ADD  R3, #1      // increment quotient
    SUBS  R0, R1     // N = N - M
    B   D_LOOP

FINISH:  LSL  R0, #16   // R3 has quotient, R0 has remainder
    ORR  R0, R3         // form the final result

    STR  R0, [SP, R2]   // store result into Rd
    POP  {R0-R12}       // restore regs
    SUBS  PC, LR, #0    // return from SVC exception
```

3 marks:
Extract operand
from SVC imm24

1.5 marks:
Scale offset
to word size

1.5 marks:
Read from stack

1 mark:
Do a division (or BL DIVIDE)

3 marks:
Store result into stack,
Pop regs off stack
Return from SVC

Extra answer space for any question on the test, if needed:

Extra answer space for any question on the test, if needed:

Extra answer space for any question on the test, if needed:

Boolean Identities

$10a.$	$x \cdot y = y \cdot x$	*Commutative*
$10b.$	$x + y = y + x$	
$11a.$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	*Associative*
$11b.$	$x + (y + z) = (x + y) + z$	
$12a.$	$x \cdot (y + z) = x \cdot y + x \cdot z$	*Distributive*
$13a.$	$x + x \cdot y = x$	*Absorption*
$14a.$	$x \cdot y + x \cdot \overline{y} = x$	*Combining*
$15a.$	$\overline{x \cdot y} = \overline{x} + \overline{y}$	*DeMorgan's theorem*
$16a.$	$x + \overline{x} \cdot y = x + y$	
$17a.$	$x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z$	*Consensus*