CPEN 211 – Introduction to Microcomputers
Final Exam – December 2017

Time: 2.5 hours (150 minutes)

This examination consists of 15 pages. Please check that you have a complete copy.

Surname First Name

Student Number

Signature

#	MAX	GRADE
1	10	
2	7	
3	5	
4	4	
5	6	
6	3	
7	10	
8	10	
9	10	
10	10	
TOTAL	75	

READ THIS

IMPORTANT NOTES:

1. By signing above you agree to uphold the highest standards of academic integrity during this exam. It is the policy of UBC APSC that any student suspected of cheating during an exam must have their booklet confiscated ***immediately***, before the end of the test, and be reported to the Dean's office.

2. The announcement "stop writing" will be made at the end of the examination. Writing after this announcement will be considered cheating and handled as above.

3. Instructions:

a) **Read each question carefully before attempting to solve it.**
b) **Attempt to solve <u>all</u> questions. Do <u>NOT</u> use red pen.**
c) **Keep all your answers on the exam sheet. There is extra space on page 14 and you can write on the back of pages. If you do either of these you MUST indicate the question number AND you MUST also indicate where your solution can be found by adding a note in the original space for the question being answered.**
d) **You are allowed two 8.5x11" single sided (or one double-sided), hand-written aid sheets (not photocopied).**
e) **Absolutely NO communication or calculator devices allowed.**
f) **All exam booklet pages <u>must</u> be returned. You are NOT permitted to take any exam material from the exam hall. Removing pages is not recommended, but if you do remove a page write your student number at the bottom.**
g) **There is a summary of ARM encoding and calling conventions on page 15.**

Question 1 [10 marks]: Short answer. Each question is worth 1 mark unless specified.

(a) Encode the following ARM instruction and write it in hexadecimal.

```
FLDD D3, [R5,#8]
```

(b) In Lab 11 you were provided code for configuring performance counters. Part of that code included the instruction "MCR p15, 0, R0, c9, c12, 5". What does "p15" in this instruction mean?

(c) Determine the relative error of 7.3 represented in 4.4 fixed-point format assuming rounding to nearest.

(d) Briefly, explain <u>why</u> hardware masks interrupts when jumping to an ISR.

(e) What does the 32-bit value 0x40600000 represent when interpreted as an IEEE 32-bit floating-point number?

(f) Is it possible to build a sequential logic circuit using only two NAND gates? If "yes", illustrate such a circuit, if "no" briefly explain why it cannot be done.

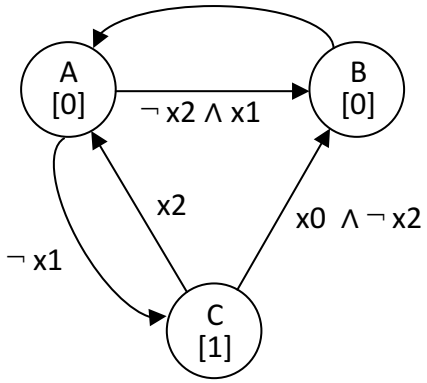
(g) [2 marks] Reduce the following Boolean expression to a minimum number of literals
$(x \wedge y) \vee \left(x \wedge \left((w \wedge z) \vee (w \wedge \bar{z})\right)\right)$

(h) List **two** rules you must follow to ensure a Verilog "always" block synthesizes to combinational logic.

(i) List the *two* 1-bit signals connected to the input of the 2-input AND gate that drives output `b[21]` of a 5:32 decoder with output `b[31:0]` built out of one 2:4 decoder with outputs `x[3:0]` and one 3:8 decoder with output `y[7:0]`.

Question 2 [7 marks]: Create an ***optimized*** digital circuit that implements the following finite state machine. In the figure below x0, x1 and x2 are inputs, transitions from a state to itself are not explicitly shown and the condition for an unlabeled edge is always true. The output f is 1-bit and is shown inside each state in square brackets. Put your final answer in the form of a logic diagram using AND, OR, NOT gates and flip-flops. Use the symbol we introduced in class for a flip-flop rather than showing how to build a flip-flop out of AND, OR, and NOT gates. When they are connected in your final design you must draw lines to show connections from the outputs of flip-flops to the inputs of next state and output logic. **You MUST use the state encoding A=01, B=10, C=11.** Show all steps used to arrive at your logic diagram. There is space on the next page if needed. **DO NOT WRITE Verilog for this question.**

☐ *Check this box if you are unsure how to simplify a Boolean expression with more than four inputs and, for a maximum of 4 marks, you will solve this question assuming* ***x0*** *is always equal to* ***x1****.*



 Student number:_____

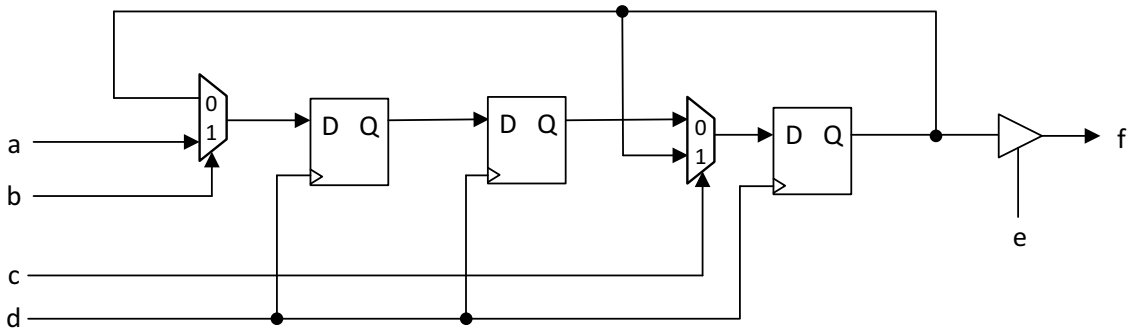
(extra space for Question 2)

Question 3 [5 marks]: Draw a logic diagram for a circuit with behavior described by the Verilog below. Use ***only*** individual D flip-flops, D latches, AND gates, OR gates, NOT gates, XOR gates, and/or multiplexers with 1-bit inputs (do not use busses). **For greater clarity in showing connections between components each flip-flop gate must use only a 1-bit D input and 1-bit Q output.** Where possible you ***must*** label wires in your diagram with names given in the Verilog. Complicated and/or unclear solutions will lose marks.

```
module circ(a,b,c,d,e,f,g,h);
  input [2:0] a, b;
  input c, d, e, f, g;
  output reg h;
  wire [3:0] t;
  reg [2:0] q;
  assign t = {(a ^ b) & t[2:0], 1'b1};
  wire s = |((1 << {c,d}) & t);
  always @(posedge e) begin
    {h,q[2:1]} = q[2:0];
    if (~f) // not a typo
    if (~g) q[0] = h;
    else q[0] = s;
  end
endmodule
```

 Student number:_____

Question 4 [4 marks]: Write **synthesizable** Verilog for the following logic diagram. For this question do NOT assume any modules are defined for you. If you want to use any modules in your solution you must define them. Solutions using modules that are not defined will receive a mark of at most 1 out of 4. All signals in the diagram below are one bit.




```
module circ(a,b,c,d,e,f);
```

 Student number:_____

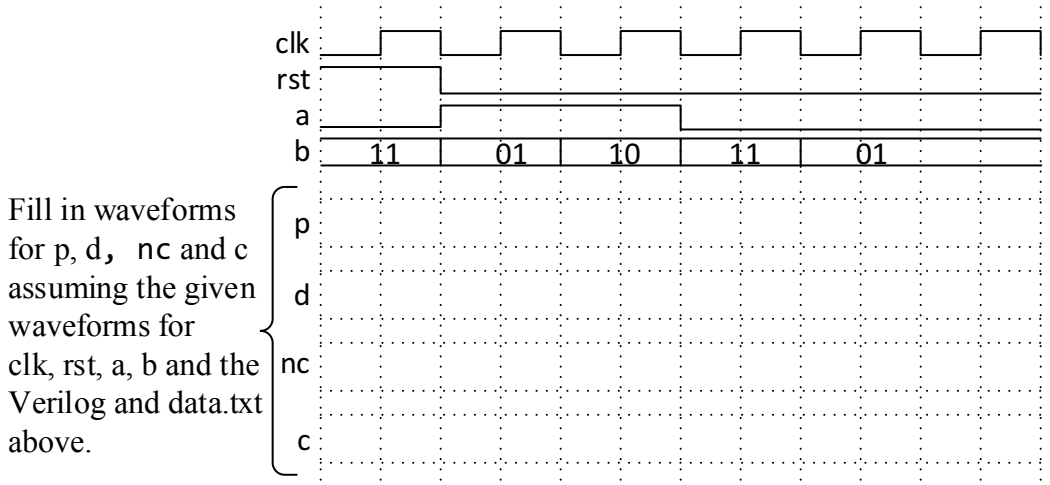
Question 5 [6 marks] Consider the following synthesizable Verilog code:

```
module system(clk, rst, a, b, c);
  input clk, rst, a;
  input [1:0] b;
  output reg [1:0] c;
  reg [1:0] m [3:0];
  reg p;
  initial $readmemb("data.txt",m);
  wire [1:0] d = m[ {p,a} ];
  wire [1:0] nc;
  wire nr = rst ? 1'b0 : d[1];
  always @(posedge clk)
    {p,c} <= {nr,nc};
  assign nc = {2{~rst}} & (({2{~d[0]}} & c) | ({2{d[0]}} & (b+c)));
endmodule
```

In addition, assume the contents of the file "data.txt" are:

00
10
01
11

Fill in the waveform below for signals `p`, `d`, nc and `c` for the code above given the inputs `clk`, `rst`, `a` and `b` below. Indicate multibit signal values using binary.



Fill in waveforms for p, d, nc and c assuming the given waveforms for clk, rst, a, b and the Verilog and data.txt above.

 Student number:_____

Question 6 [3 marks]: Draw a schematic using NFETs and PFETs for a restoring gate that implements the function: $f = \overline{(a \vee b) \wedge (c \vee d)}$

Question 7 (a) [3 marks] Assume that the variables `f` and `g` are assigned to registers `R0` and `R1`, respectively. Also, assume that the base address of the arrays `A` and `B` containing 32-bit integers are in registers `R2` and `R3`, respectively. Write ARM assembly code for the following C statement.

```
f = A[B[g]+4];
```

Question 7 (b) [1 marks] Assuming `R0` = 0x80000001 and `R1` = 0x70000002 what is the value of `R2` after the following instructions execute?

```
CMP   R0, R1
MOV   R2, #1
MOVLT R2, #2
MOVGE R2, #3
```

Question 7 (c) [3 marks] Write ARM assembly for the following C code. Assume the base of array `A` is in register `R0`, `i` is in `R1`, `j` is in `R2`, `value` is in register `D0`, and that array `A` is declared as "`double A[128][128];`".

```
value = A[i-1][j+1];
```

Question 7 (d) [3 marks] Write ARM assembly code that, when run, reverses the order of the bits initially in `R0`. For example, if `R0` initially contains 0x01000005, then after executing your code `R0` should contain 0xA0000080.

Question 8 [10 marks] Write ARM assembly for the C function "`func()`" given below. Your solution **must** follow the ARM calling conventions.

```
unsigned func(unsigned m, unsigned n)
{
  if(m==0)
    return n+1;
  else if(m>0 && n==0)
    return func(m-1,1);
  else
    return func(m-1, func(m,n-1) );
}
```

(additional space for this question on the next page)

(extra space for Question 8)

Question 9 [10 marks] Convert the following ARM assembly into equivalent C code. Assume the base of array `A` is in register `R0`, the base of array `B` is in register `R1`, `i` is in `R2`, `sum` is in `R3` and `N` is in `R4`.

```
      MOV R3, #0
      MOV R2, #0
      CMP R2, R4
      BGE L3
L1:   LDR R5, [R0, R2, LSL#2]
      ADD R6, R2, #1
      LDR R7, [R1, R6, LSL#2]
      CMP R5, R7
      BLE L2
      ADD R3, R3, R5
L2:   ADD R2, R2, #1
      CMP R2, R4
      BLT L1
L3:
```

Question 10 (a) [4 marks] The contents of memory are shown below. Addresses are 12-bits, and each entry in the "Data" column contains 16 bytes. The 16 bytes are shown as a pair of two 8-byte words, with the word at block offset 0 on the right. Assume a **little-endian** byte order.

Address	Data															
	Word 1								Word 0							
	B7	B6	B5	B4	B3	B2	B1	B0	B7	B6	B5	B4	B3	B2	B1	B0
0x100	2F	66	3E	FE	00	00	3D	A0	00	B0	00	3F	00	07	61	20
0x110	DE	AD	BE	EF	33	EA	05	30	01	1E	39	00	81	83	51	1F
0x120	87	99	15	98	16	56	80	17	02	20	05	01	11	10	70	70

Consider a ***two-way set associative cache*** with 1 index bit and block size of 8 bytes and LRU replacement. The tables below contain a sequence of memory references and the cycle on which they occur. **Fill in the following information in the table below:** Tag, Index, Offset, whether the access was a hit and, under "R Value", the value returned by a LDR instruction that reads from "Address".

Cycle	Address	Tag	Index	Offset	hit? (Y/N)	R Value
1	0x10C					
2	0x104					
3	0x118					
4	0x12C					
5	0x100					
6	0x108					

Question 10 (b) [4 marks] Complete the pipelining timing diagram assuming the five-stage pipeline studied in class assuming all forwarding paths required to reduce or eliminate stalls are available. When forwarding is necessary, indicate where it occurs in your diagram using an arrow from the pipeline stage the value is forwarded form to the pipeline stage the value is forwarded to. You may indicate an instruction is stalled by writing "s" and that it is squashed by circling the letter for the stage it is squashed in.

```
Loop: LDR R1, [R2,#4]
      STR R1, [R1,#8]
      ADD R3, R3, R1
```

LDR R1, [R2,#4]																				
STR R1, [R1,#8]																				
ADD R3, R3, R1																				

Question 10 (c) [2 marks] Assume you have discovered ways to make LDR, STR and ADD instructions much faster. Specifically, you know how to make LDR 30 times faster, STR 20 times faster and ADD 15 times faster. If 25% of your program is LDR instructions, 25% is STR instructions then what fraction of your program must be ADD instructions to achieve and overall speedup of 10?

Extra Space –clearly indicate Question number

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ARM Summary

Data Processing (op): ADD (4), SUB (2), AND (0), ORR (12), MVN (15), MOV (13)
Data Transfer (op): LDR (25), STR (24), LDR *shifted Rm* (57), STR *shifted Rm* (56)
Registers: R0-R15: SP is R13, LR is R14, PC is R15; CPSR (bits 31-28 are N, Z, C, V)

***Instruction Encodings**:*

Instruction	Format	Cond (4 bits)	F (2)	I (1)	Op (4/6)	S (1)	Rn (4)	Rd (4)	Operand2 (12)
ADD, etc…	DP	cond	0	imm?	op	0	reg	reg	imm / reg, shift
LDR, etc…	DT	cond	1	n.a.	op	n.a.	reg	reg	imm / reg, shift

Instruction	Cond	Op (4)	Target (24)
B<suffix>	cond	10	$=(PC_{target}-(PC_{branch}+8))/4$

Instruction		Cond (4 bits)	Op (8)	Rn (4)	(4)	Operand2 (12)
CMP		cond	0x15	reg	0	reg
CMP (imm)		cond	0x35	reg	0	imm
CMP (imm)		cond	0x35	reg	0	imm

Instruction	Cond (4 bits)	(4)				(1)	(3)			(4)	(4)	Coproc (4)	Imm (8)
FLDD	cond	1	1	0	1	U	0	0	1	Rn	Dd	11	imm
FMULD	cond	1	1	1	0	0	0	1	0	Dn	Dd	11	Dm
FADDD	cond	1	1	1	0	0	0	1	1	Dn	Dd	11	Dm
FSTD	cond	1	1	0	1	U	0	0	1	Rn	Dd	11	imm

***Shift Encoding** (Operand2)*
Shifted Operands (shift):
LSL (00), LSR (01)

11	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
shift_imm					*shift*		*0*	*Rm*			
Rs				*0*	*shift*		*1*	*Rm*			

***Immediate**:value*=bits 0:7 as unsigned (0-255). If *r*=bits 8:11≠0 rotate *value* right by 2×r.

***Condition codes** (use to determine value for "cond" field in instruction encoding):*

Suffix	Flags	Meaning	cond
EQ	Z	Equal	0
NE	~Z	Not Equal	1
HS	C	Unsigned >=	2
LO	~C	Unsigned <	3
MI	N	Negative	4
PL	~N	Positive	5
VS	V	Overflow	6
VC	~V	No overflow	7

Suffix	Flags	Meaning	cond
HI	C&~Z	Unsigned >	8
LS	~C | Z	Unsigned <=	9
GE	N = V	Signed >=	10
LT	N != V	Signed <	11
GT	~Z&N=V	Signed >	12
LE	Z | N!=V	Signed <=	13
AL	-	Always	14
NV	-	Reserved	15

ARM calling convention:

Name	Registers	Usage	Preserved on call?
a1-a2	R0-R1	Argument / return result / scratch register	no
a3-a4	R2-R3	Argument / scratch register	no
v1-v8	R4-R11	Variables for local routine	yes
ip	R12	Intra-procedure-call scratch register	no
sp	R13	Stack pointer	yes
lr	R14	Link Register (Return address)	yes
pc	R15	Program Counter	n/a

 Student number:_____