CPEN 211 – Introduction to Microcomputers
Final Exam – December 2018

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	18	
2	6	
3	7	
4	5	
5	6	
6	10	
7	10	
8	8	
TOTAL	70	

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 all questions. Do NOT 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 must be returned. You are NOT permitted to take any exam material from the exam hall. Temporarily 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.**

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Question 1 [18 marks]: Short answer. Each question is worth 1 mark unless specified.

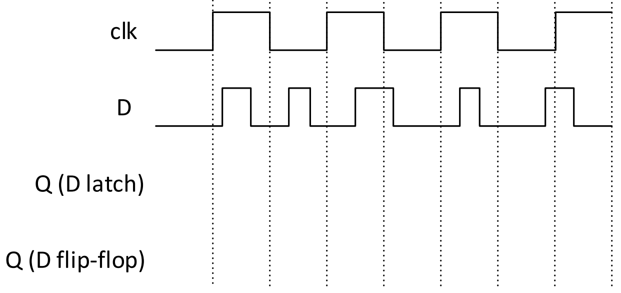
(a) What is combinational logic?

(b) [2 marks] A logic device encodes signals with levels proportional to its power supply voltage (V_{DD}) according to the table below. Suppose two such logic devices A and B send signals to one another and the supply of device A is V_{DDA} = 1.0V. Assuming that there are no other noise sources and that the two devices have a common ground (i.e., 0V is the same level in both devices), what is the range of supply voltages for device B, V_{DDB}, over which the system will operate properly?

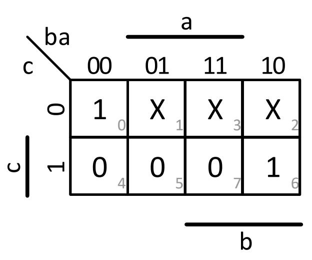
V_{OL}	0.1 V_{DD}
V_{IL}	0.4 V_{DD}
V_{IH}	0.6 V_{DD}
V_{OH}	0.9 V_{DD}

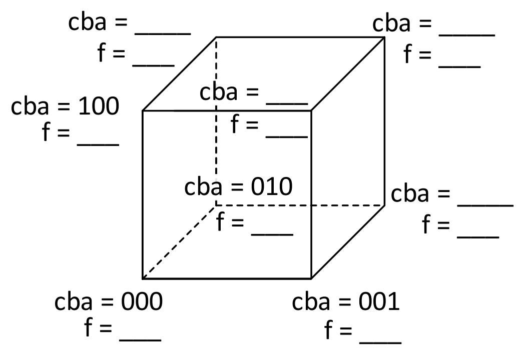
(c) Draw a schematic for the following unsimplified logic equation: $((x \wedge y) \vee z) \wedge (x \wedge \overline{z})$

(d) [2 marks] Draw the output of a D latch and D flip-flop given the following inputs:



(e) Consider the following KMAP for a Boolean function *f* with three inputs *c*, *b* and *a*. Label the vertices on the cube on the right including the missing values for cba and the value of f at each vertex. Also, highlight the edges or surfaces on the cube that connect minterms contained within each prime implicant.





Student number:______________

(f) Using full adders, design a circuit that accepts a seven-bit input and outputs the number of inputs that are 1 as a three-bit binary number.

(g) Using the symbols introduced in class for NFET and PFET transistors draw a CMOS circuit that implements a tri-state buffer.

(h) How many rows are required in a one-level page table if virtual addresses are 24-bits and page size is 8KB?

(i) Does a direct mapped cache require tags? If so why? If not why not?

(j) Does hardware or software mask interrupts when the interrupt input (IRQ) is set to 1 on an ARM processor?

(k) **[2 marks]** Why does ARM require a separate link register (LR_IRQ) to support interrupts? What value should be in this register after an ISR starts?

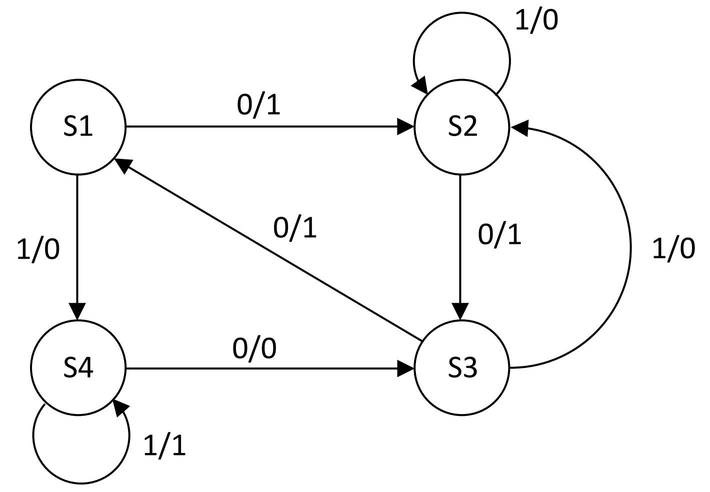
(l) What is the absolute error of 1.635 represented in 2.2 fixed-point format assuming rounding to nearest?

(m) Express the number -1025.175 in 32-bit IEEE format using hexadecimal.

(n) **[2 marks]** Encode the branch in the code below and write it in hexadecimal.

```
L1: ADD R1, R1, #1
    CMP R1, R2
    BLT L1
```

Question 2 [6 marks]: Create an ***optimized*** digital circuit that implements the following Mealy finite state machine. Each edge is labeled with an "input / output" combination. For example, when in state S1 if the input is 0 the output should be 1 and, on the rising edge of the clock, the next state should be S2. 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 S1=01, S2=10, S3=11, S4=00.** 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.**



(extra space for Question 2)

Question 3 [7 marks]: Draw a logic diagram for a circuit with behavior described by the Verilog below. For full marks do NOT use multiplexer symbols or adder blocks in your final answer (using either will result in a maximum mark of 4). Rather, use ***only*** individual (1-bit) flip-flops, latches, AND, OR, NOT and XOR gates. **For clarity in showing connections between components each gate MUST use only 1-bit inputs and outputs. Components should be connected by 1-bit wires** (marks will be deducted for any use of bus notation in your logic diagram). Where possible label wires with names given in the Verilog. Complicated and/or unclear solutions will lose marks. The contents of the file "data.txt" are shown inside the box on the right.

```
module q3(a,b,c,d);
  output reg [3:0] a;
  input b, c, d;
  reg [3:0] g [3:0];
  initial $readmemb("data.txt",g);
  reg [1:0] i;
  always @(posedge b) begin
    casex( {c,d} )
      2'b1x: i <= 2'b00;
      2'b00: i <= i;
      2'b01: i <= i + 2'b1;
      default: i <= 2'bxx;
    endcase
    a <= g[i];
  end
endmodule
```

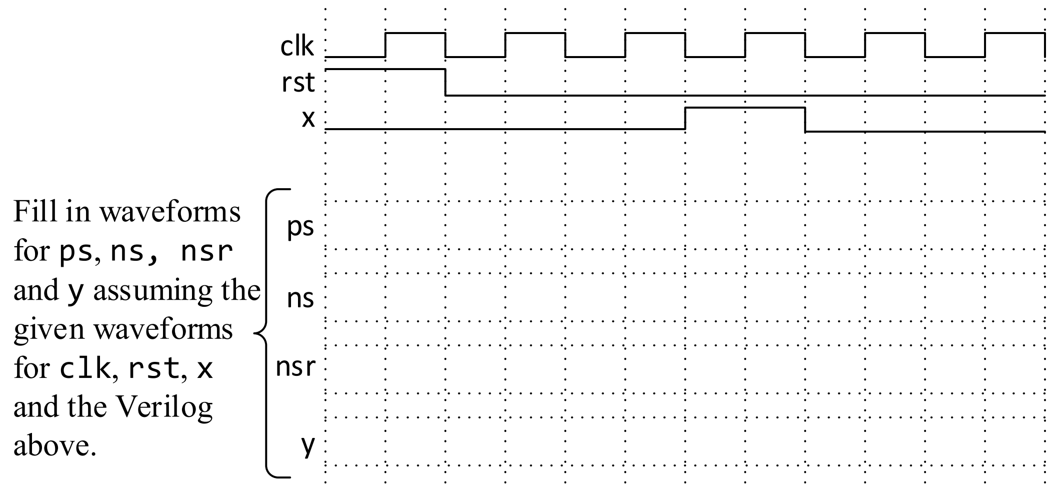
Contents of data.txt:

```
@0 0001
@1 0110
@2 0110
@3 1101
```

Question 4 [6 marks] Consider the following synthesizable Verilog code:

```
module q4(clk, rst, x, y);
  input clk, rst, x;
  output reg y;
  reg [2:0] ps, ns;
  wire [2:0] nsr = rst ? 3'b000 : ns;
  always @(*)
    case( ps )
      3'b000:  {ns,y} = {(x ? 3'b010 : 3'b001), 1'b0};
      3'b001:  {ns,y} = {(x ? 3'b010 : 3'b001), 1'b1};
      3'b010:  {ns,y} = {(x ? 3'b000 : 3'b100), 1'b0};
      3'b100:  {ns,y} = {(x ? 3'b100 : 3'b110), 1'b1};
      3'b110:  {ns,y} = 4'b0101;
      default: {ns,y} = 4'bxxxx;
    endcase
  always @(posedge clk)
    ps <= nsr;
endmodule
```

Fill in the waveform below for signals `ps, ns, nsr`, and `y` for the code above given the inputs `clk`, `rst`, `a` and `b` below. Indicate multibit signal values using binary.


clk
rst
x
Fill in waveforms for ps, ns, nsr and y assuming the given waveforms for clk, rst, x and the Verilog above.
ps
ns
nsr
y

Question 5 [5 marks]: On the following page write **synthesizable** Verilog for a computer that has two types of instructions "copy" and "jump". Your computer **must** instantiate and use the read-write memory module named RAM1P defined below:

```
module RAM1P(clk,addr,write,data);
  parameter address_width = 3;
  parameter data_width = 8;
  parameter filename = "data.txt";
  input clk, write;
  input [address_width-1:0] addr;
  inout [data_width-1:0] data;
  reg [data_width-1:0] mem [2**address_width-1:0];

  initial $readmemb(filename,mem);
  assign data = (~write) ? mem[addr] : {data_width{1'bz}};
  always @(posedge clk)
    if (write)
      mem[addr] = data;
endmodule
```

RAM1P differs from the RAM used in Lab 7 & 8: There is a single bus "`data`" for both input and output, read operations are combinational and there is only one address input. Inside mem.v the bus "`data`" is declared as "`inout`". We didn't talk about "`inout`" in class but you should still be able to solve this question! Declare the bus you connect to `data` when instantiating RAM1P as a multibit "`wire`" and read it normally but write it using a tri-state buffer. Your computer should reset its program counter (`pc`) to 0 if both `reset` is equal to `1'b1` and there is a rising edge on `clk` otherwise it should begin executing instructions when reset is set to `1'b0`. Use the default parameters when instantiating RAM1P.

The encoding for "`copy`" and "`jump`" are shown in the table below.

Assembly	***Encoding***								***Operation***
	7	6	5	4	3	2	1	0	

`copy #dst, #src`	`0`	`dst`				`src`			`if (0 <= dst < 8) {`
									`  mem[dst] = mem[src];`
									`} else {`
									`  out = mem[src];`
									`}`
									`pc = pc + 1;`

`jump #offset`	`1`	*unused*				`offset`			`pc = pc + offset`

(Space for Question 5)

Question 6 [10 marks] Write ARM assembly for the following C code. Assume the variables `i`, `j`, `tmp` are in registers `R0`, `R1`, and `R2` and the base of Array A is in `R3`.

```
for( int i=0; i < 10; i++ ) {
  int j = i + 1;
  while( j < 10 ) {
     if ( A[j] < A[i] ) {
        int tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;
     }
     j = j + 1;
  }
}
```

Question 7 [10 marks] Write ARM assembly for the C function "`func()`" given below. Your solution **must** follow the ARM calling conventions described in class.

```
int func(int *A, int n)
{
   int tmp = 0;
   for( int i=0; i+4 < n; i++ ) {
     tmp = tmp + g(A[i], A[i+1], A[i+2], A[i+3], A[i+4]);
   }
   return tmp;
}
```

All you know about the function "`g()`" is that it is declared as follows:

```
int g(int a, int b, int c, int d, int e);
```

(additional space for this question on the next page)

(extra space for Question 7)

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Question 8 (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 four 4-byte words, with word at offset 0 on the right. Assume a **little-endian** byte order. +0, +1, etc… refer to the address offset within a row. E.g., the byte value at address 0x100+0xF = 0x10F is **2F**.

Address	Data															
	Word 3				Word 2				Word 1				Word 0			
	+F	+E	+D	+C	+B	+A	+9	+8	+7	+6	+5	+4	+3	+2	+1	+0
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
0x130	40	43	03	00	50	30	60	07	10	80	90	02	FF	EE	DD	CC

Consider a ***three-way set associative cache*** with block size of 4 bytes and LRU replacement. Assume the total capacity of the cache is 48 bytes. The table below contains 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 an LDRB instruction that reads from "Address".

Cycle	Address	Tag	Index	Offset	hit? (Y/N)	R Value
1	0x104					
2	0x120					
3	0x117					
4	0x125					
5	0x122					
6	0x136					
7	0x104					
8	0x127					

Question 8 (b) [2 marks] The AMD Ryzen processor reportedly uses a hardware neural network to predict whether branch instructions will be taken or not taken. For this question assume this predictor guesses the correct next instruction to fetch for 95% of all branches. However, for 5% of the branches the predictor is wrong and the processor incurs a 20 cycle penalty. Assuming one in five instructions is a branch and that if all branches were predicted correctly the CPI would be 0.25 how much faster would the processor be if the predictor was 100% accurate?

Question 8 (c) [2 marks] You decide to re-write a portion of a program. This portion takes 80% of the time used by the program when run on a high-performance CPU. Specifically, you re-write this portion so that it runs 10 times faster on a special-purpose accelerator. However, you do not actually need to speed the original program up. Instead, you would like to save power by running the remaining 20% of the original program (the portion you did not re-write) on a low power CPU. When using the special purpose accelerator how much ***slower*** can this lower-power CPU be versus the high-performance CPU such that overall execution time does not increase when compared with running the original software on the high-performance CPU?

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