CPEN 211 – Introduction to Microcomputers
Final Exam – December 2016

Time: 2.5 hours (150 minutes)

This examination consists of 14 pages. Please check that you have a complete copy. You may use both sides of each sheet if needed.

Surname First Name

Student Number

Signature

#	MAX	GRADE
1	10	
2	8	
3	9	
4	8	
5	10	
6	5	
7	10	
8	10	
9	8	
10	10	
TOTAL	88	

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.**
c) **Keep all your answers on the exam sheet. There is extra space on page 13 and you may write on the back of pages if you indicate the question number and add a note saying you did this on the page containing 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. 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 some ARM syntax on page 14.**

UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

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

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

```
L1:
    BGE L2
    ADD R1, R2, R3
L2:
```

(b) Briefly, explain why we use "`SUBS PC, LR, #4`" to return from an interrupt service routine (ISR) instead of the instruction we used for returning from a normal function.

(c) Encode -1.75 into IEEE single-precision floating-point and write it in hexadecimal.

(d) **[2 marks]** Consider adding virtual memory with the following properties to the Simple RISC Machine from the labs: 10-bit physical addresses, 64 byte pages, 12-bit virtual addresses. As in Lab 6 and 7 each memory location contains a 16-bit word. Assume disk addresses are not stored in the page table. What is the total size of the page table measured in 16-bit words for each process, assuming valid, protection, dirty, and use bits take a total of 4 bits and all virtual pages are in use?

(e) Briefly, explain what "`#(8)`" does in the following line of Verilog code:

```
foo #(8) bar(a, b, c);
```

(f) **[2 marks]** Briefly, explain what an inferred latch is (do ***not*** write Verilog).

(g) Draw a logic gate diagram with the same behavior as the following Verilog.

```
always @(*) begin
  Y = 1'b0;
  if (A == 1'b1)
    if (B == 1'b0)
      Y = 1'b0;
  else
    Y = 1'b1;
end
```

Question 2 [8 marks]: Design a four-bit decimal Fibonacci circuit. This circuit outputs a 1 if and only if its input is a Fibonacci number in the range 0 to 9 (i.e., 0, 1, 2, 3, 5, or 8). You must draw a truth table, Karnaugh map, identify a cover and draw a logic circuit for the function using AND, OR and NOT gates.

Question 3 [9 marks]: Draw a logic diagram for a circuit with behavior described by the Verilog below. Use ***only*** individual flip-flops, latches, AND gates, OR gates, NOT gates, and XOR gates. For greater clarity in showing connections between components each flip-flop and/or latch 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. NOTE: You ***must*** show logic required to compute the output `e` of module q3 but are otherwise free to optimize the circuit. If you do not know all the syntax used then draw logic only for the portions you understand.

```
module q3(a,b,c,d,e);
  input a, b, c, d;
  output reg e;
  reg r, s;
  reg [1:0] w;
  wire u;
  wire [3:0] t;
  always @*
    case( {a,b,c} )
      3'b011:  s = 1'b1;
      default: s = 1'b0;
    endcase
  assign u = |{s,t[2]};
  assign t = 1 << {a,c};
  always @(posedge d) begin
    w <= {(b ? 1'b0: (w[0] ^ w[1])), u};
    r = w[1];
  end
  always @(*)
    if (d == 1'b0)
      e = r;
endmodule
```

Question 4 [8 marks]: Write **synthesizable** Verilog for a combinational logic circuit with **n**-bit input **a** and **n**-bit output **s**. The output **s** should be a one-hot code indicating the ***second*** lowest bit-position of input **a** that is set to 1 on the input. If 0 or 1 input bits are set to 1 then **s** is all zeros. For example, if **n**=4 and **a**=4'b1111 then **s**=4'b0010, if **n**=4 and **a**=4'b1110 then **s**=4'b0100, if **n**=8 and **a**=8'b11100100 then **s**=8'b00100000, and if **n**=5 and **a**=5'b00010 then **s**=5'b00000. Your module, Nxt1, ***must*** be parameterized by **n** and work for **n** ≥ 2.

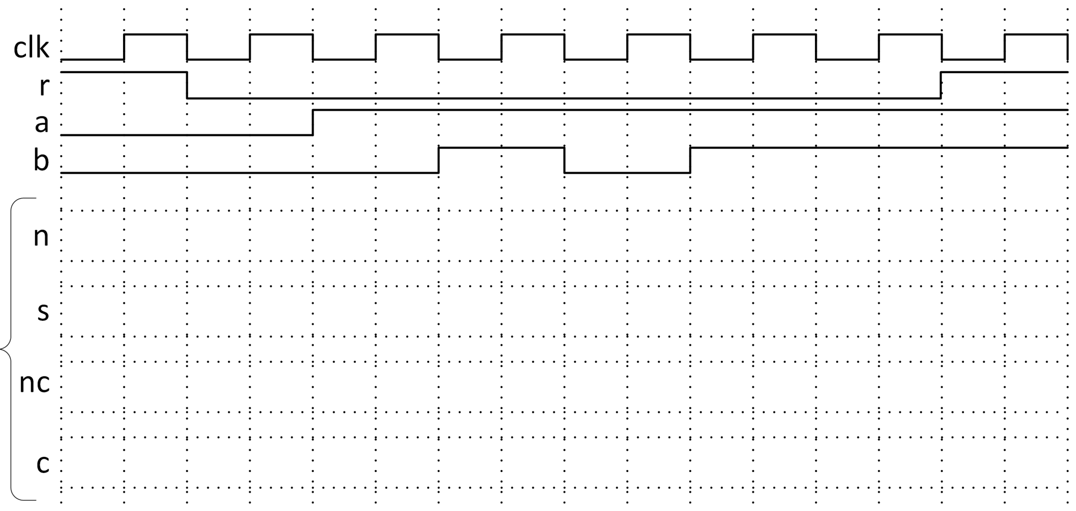
```
module Nxt1(a,s);
```

Question 5 [10 marks] Consider the following Verilog:

```
module q5(clk,r,a,b,c);
  input clk, r, a, b;
  output reg [1:0] c;
  reg  [1:0] s, n, nc;

  always @(posedge clk) begin
    s = n;
    c = nc;
  end

  always @* begin
    if (r==1'b1)
      n = 2'b00;
    else
      case (s)
        2'b00:   n = a ? 2'b01 : 2'b00;
        2'b01:   n = (c == 2'b10) ? 2'b10 : 2'b01;
        2'b10:   n = 2'b10;
        default: n = 2'bxx;
      endcase
  end

  always @*
    case (s)
      2'b00: nc = 0;
      2'b01: nc = c + {1'b0, b};
      2'b10: nc = c;
      default: nc = 2'bxx;
    endcase
endmodule
```

Fill in the waveform below for signals n, s, nc, and c for the code above given the inputs clk, r, a and b below. Indicate multibit signal values using binary.

Fill in waveforms for n, s, nc and c assuming the given waveforms for clk, r, a, b and the Verilog above.



Student number:______________

Question 6 [5 marks]: Draw a schematic using NFETs and PFETs for a restoring gate that implements the function: $f = \overline{a \vee \big(b \wedge (c \vee d)\big)}$

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 = g - A[B[3]+1];
```

Question 7 (b) [2 marks] Assuming R3 = 0x55555555 what is the value of R5 after the following sequence of instructions executes?

```
MOV R5, #0xBD
AND R5, R5, R3, LSL #1
```

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`, `sum` is a 64-bit double precision floating-point in register `D0`, and that array `A` is declared as "`double A[4][8];`".

```
sum = sum + A[i][j];
```

Question 7 (d) [2 marks] Write ARM assembly for the following C code assuming `p` is in `R0` and has type pointer to "`struct { int A; double B; int *C; }`" and that a variable of type `int` occupies 32-bits. Assume `tmp` is in `R1`.

```
tmp = p->C[5];
```

Question 8 [10 marks] Assuming the C function "`merge()`" is declared as follows

```
void merge(int a[], int i1, int j1, int i2, int j2);
```

write ARM assembly for the C function "`mergesort()`" given below. Your solution must follow the ARM calling conventions discussed in class. Assume `i` and `j` are ≥ 0. NOTE: You do ***not*** need to write code that implements the function "`merge()`".

```
void mergesort(int a[], int i, int j)
{
    int mid;

    if(i<j)
    {
        mid=(i+j)/2;
        mergesort(a,i,mid);
        mergesort(a,mid+1,j);
        merge(a,i,mid,mid+1,j);
    }
}
```

(additional space for this question on the next page)

(extra space for Question 8)

Question 9 [8 marks] Convert the following ARM assembly into equivalent C code.

```
      MOV R0, #0
L1:   CMP R0, R1
      BGE L5
      MOV R2, #0
      MOV R3, R0
L2:   CMP R3, R1
      BGE L4
      LDR R4, [R5, R3, LSL#2]
      CMP R4, #0
      BLE L3
      ADD R2, R2, R4
L3:   ADD R3, R3, #1
      B   L2
L4:   STR R2, [R5, R0, LSL#2]
      ADD R0, R0, #1
      B   L1
L5:
```

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 ***direct mapped cache*** holding two 8-byte blocks. 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 the value returned by an LDR instruction (R Value).

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

Question 10 (b) [4 marks] Complete the pipelining timing diagram assuming the five-stage pipeline studied in class and that forwarding paths required to reduce or eliminate stalls are available. Assume branches are resolved in the execute stage, do not have delay slots, and are predicted "not-taken" but that the BNE instruction is "taken". When forwarding is necessary, indicate where it occurs in your diagram using an arrow.

```
Loop: LDR R1, [R2,#4]
      ADD R3, R3, R1
      CMP R1, #0
      BNE Loop // assume this branch is "taken"
```

LDR R1, [R2,#4]																				
ADD R3, R3, R1																				
CMP R1, #0																				
BNE Loop																				
LDR R1, [R2,#4]																				

Question 10 (c) [2 marks] What fraction of a program must be parallel to achieve speedup of 3× on a processor with 4 cores?

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: stack pointer is R13, link register is R14, PC is R15; CPSR (status: bits 31-28 are N, Z, C, V flags)
Shifted Operands (shift): LSL (00), LSR (01)
***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
			Op (4)		**Target (24)**				
B<suffix>		cond	10		=$(PC_{target}-(PC_{branch}+8))/4$				
			Op (8)				**Rn (4)**	**(4)**	**Operand2 (12)**
CMP		cond	0x15				reg	0	reg
CMP (imm)		cond	0x35				reg	0	imm

***Shift Encoding** (Operand2 in instruction encoding)*

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

***Immediate Encoding**: value*=bits 0:7 as unsigned (0-255). If *r*=bits 8:11≠0 then also rotate this *value* right by 2×r. "Rotate right" means the least significant bit is copied to the most significant bit. E.g., 0001 rotated right by 1 is 1000 and rotated right by 2 is 0100.

***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