

CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Instructor: Rahul Nayar

TAs: Annie Lin, Mohit Verma

Examination 2

In Class (50 minutes)

Wednesday, March 8, 2017

Weight: 17.5%

NO: BOOK(S), NOTE(S), CALCULATORS OR ELECTRONIC DEVICES OF ANY SORT.

The exam has **ten pages**. You **must turn in the pages 1-9**. **Circle your final answers**. Plan your time carefully since some problems are longer than others. Use the blank sides of the exam for scratch work.

LAST NAME: _____

FIRST NAME: _____

ID#: _____

Problem	Maximum Points	Points Earned
1	4	
2	6	
3	3	
4	5	
5	4	
6	2	
7	4	
8	5	
9	5	
Total	38	

- Figure 1 shows the output of a transistor-level circuit connected to the select line of a 2-input multiplexer.

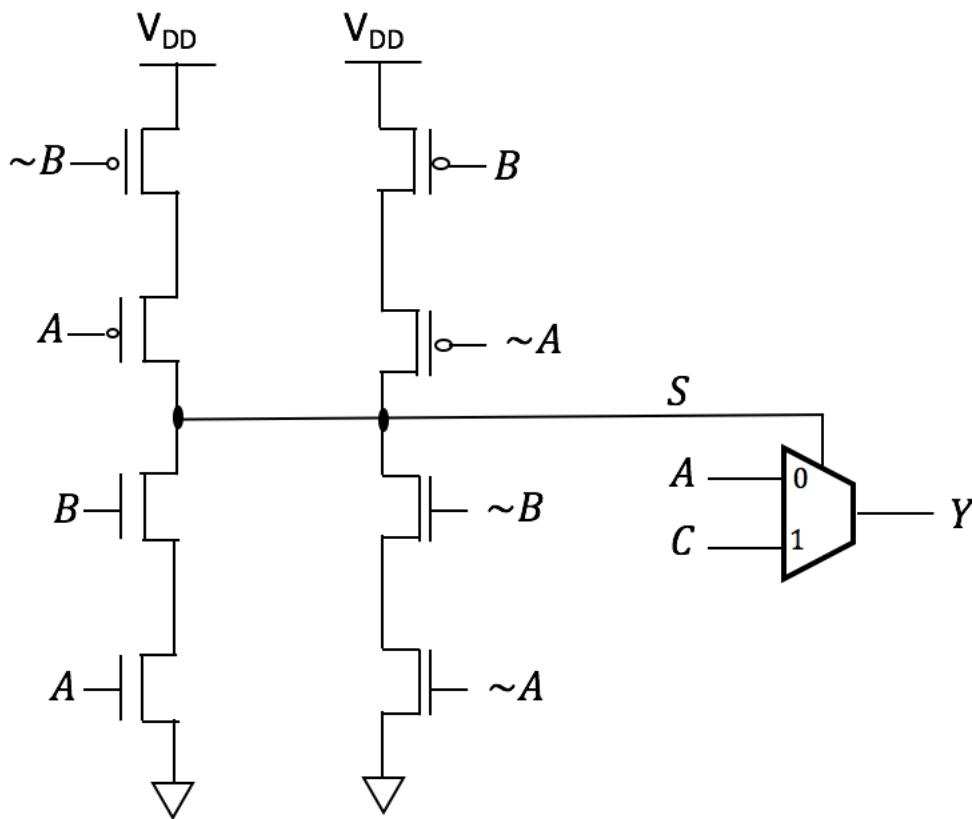


Figure 1

Fill out the following truth table for the overall circuit. (Note: $\sim A$ means NOT(A)). Fill all 8 rows. (4 points)

A	B	C	S	Y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

2. Consider the logic equation

$$Z = \text{NOT} (A \text{ OR } \text{NOT} (B \text{ AND } \text{NOT} (\text{NOT } A \text{ AND } \text{NOT } C)))$$

- a) Using DeMorgan's law, simplify the logic equation for Z to **reduce** the number of NOT gates to one. Show your work for full credit. (Solutions using more than one NOT gate will receive only partial credit.) (3 points)

- b) Fill out the following truth table for Z. (2 points)

A	B	C	Z
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

- c) Draw the logic gate diagram for the simplified equation obtained in part a) above using NOT and AND gates **only**. You can use AND gates with any number of inputs. (1 point)

3. a) An architecture has the following format for a 16-bit ISA. Given that all the **fields** are of **equal length**, what is the maximum number of Opcodes the ISA can represent? (1 point)

Opcode	Destination Register	Source Register 1	Source Register 2
--------	----------------------	-------------------	-------------------

- b) The PC-relative load instruction of a certain LC-3 like 16-bit ISA has the following format:

Load Opcode (4 bits)	Destination Register (3-bits)	PC-offset (9-bits)
----------------------	-------------------------------	--------------------

If this instruction resides in memory at location 0x3000, what is the range of memory locations that can it can address when it is executed? Assume 2's complement representation. (2 points)

4. Answer the short answer questions below.

- a) In LC3, what is the difference between a *jump* and a *branch* instruction? Describe at least one way that they are the same and one way that they are different. (1 point)
- b) True or false: a single LD instruction can be used to load **from** a memory location that is 100 locations away from the current PC. (1 point)

- c) Match each of the descriptions below to the appropriate terms. Each definition will have only one answer. Possible terms: *IR, ALU, Fetch Instruction, MDR, Execute Operation, Decode Instruction, Driver, Algorithm, Abstraction* (2 points)

	Register that contains the data to write to memory or read from memory
	Instruction processing state that identifies the opcode and other operands
	PC is always incremented in this instruction processing state
	Program that controls access to a device

- d) A decoder has 1024 outputs. How many inputs does it have? (1 point)

5. In the von Neumann model, the control unit orchestrates execution of a program



Figure 2

- a) What do **IR** and **PC** stand for? What do they do? (2 points)

- b) Given the six steps of instruction processing, put them into the correct order. Write the numbers 1-6 next to the appropriate instructions, with 1 being the first and 6 being the last. (2 points)

	Fetch instruction from memory
	Fetch operands from memory
	Store result
	Execute operation
	Decode instruction
	Evaluate address

6. Figure 3 shows a 1-bit half adder, and a 2-to-4 decoder with the same inputs A and B. You have one 3-input OR gate, and one 2-input AND gate. Label the **inputs** of the given OR and AND gates in terms of the **output of the decoder** (C, D, E or F), to realize a 1-bit half adder. Clearly **label the outputs** of the two gates to show which gate outputs “Sum” and which gate outputs “C_{out}”. (Note: You can also use the signals 0 and 1 as any input, if needed). (2 points)

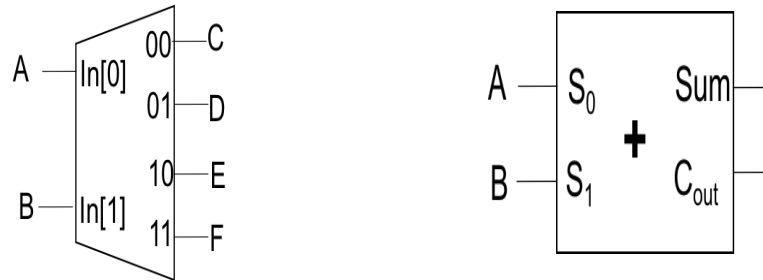
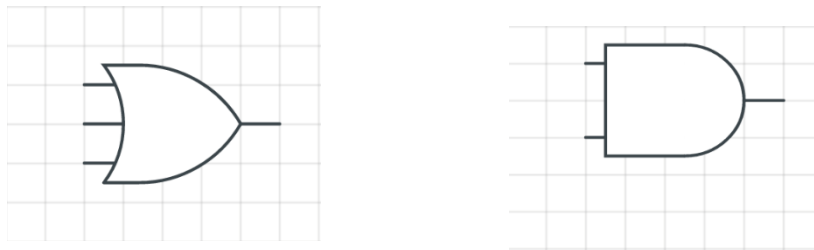


Figure 3



7. a) An R-S latch is shown in Figure 4. Given that **out** is initially set to 0, fill out the following table. (2 points)

Time	0	1	2	3	4	5	6
R	NONE	1	1	0	1	1	0
S	NONE	1	0	1	1	1	1
OUT	0						

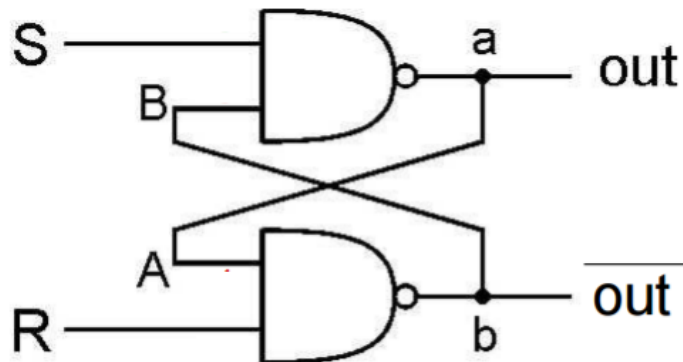
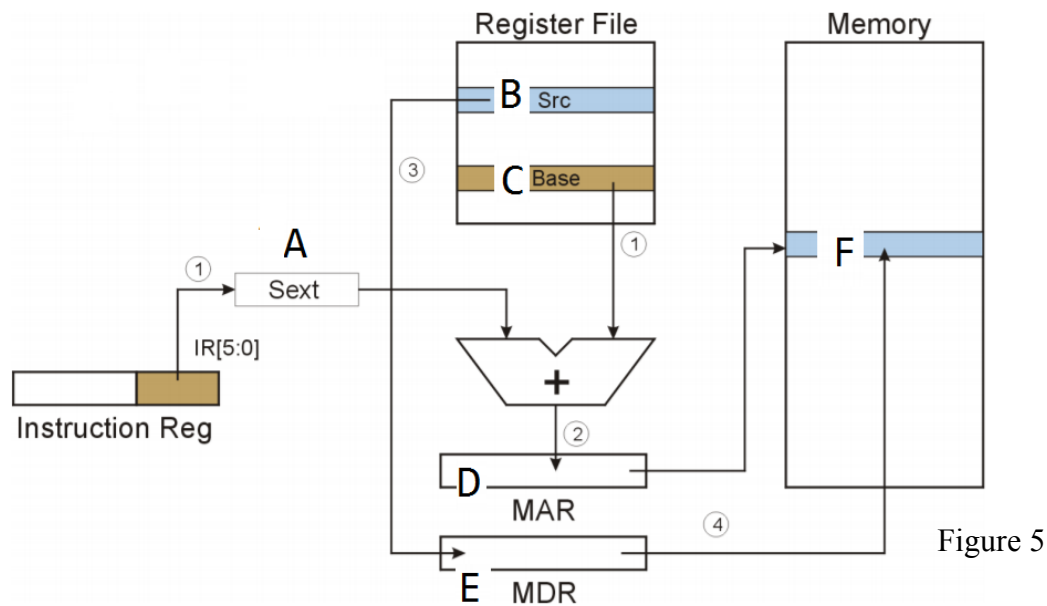


Figure 4

- b) A **gated D-latch** adds two inputs, D and WE, to an RS-latch to decide when to set or keep memory. Describe how do we combine multiple gated D-latches to make a **register**? (2 points)

8. Figure 5 shows the steps required to execute a certain LC-3 instruction.



- a) What is the instruction (ex. ADD, LD)? (1 point)
- b) Suppose that we know **A** has a value of xFFFD, **B** holds a value of x1234, and **C** has a value of x2345. Is it possible to write out the instruction in its LC-3 **16-bit binary** format? If yes, write the instruction. If not, explain why. (1 point)
- c) Using the values from part (b), what are the final values stored in **D**, **E**, and **F**? Write your answers in hex. Explain your answers for full credit. (3 points)

9. a) Draw a finite state machine (FSM) that has four states (00, 01, 10, and 11) and a 1-bit input. It should have the following transitions: (3 points)

- i) Input of 0 changes state from 00 to 01.
- ii) Input of 1 at state 00 does not change the state.
- iii) Input of 1 changes state from 01 to 10.
- iv) Input of 0 at state 01 does not change the state.
- v) Input of 1 changes state from 10 to 11.
- vi) Input of 0 changes state from 11 to 10.

b) Add extra transitions in your FSM above to satisfy the following condition:

Given that the start state is 00, your FSM should finish at state 11 at the end of ALL the following bit sequences: (2 points)

- i) 011, 0111, 01111.
- ii) 011, 011001, 011001001.

Clearly indicate the additional lines you added in step b).

(Entered by Mark D. Hill on 03/14/2007; last update 03/15/2007)

PC': incremented PC. setcc(): set condition codes N, Z, and P. mem[A]:memory contents at address A.

SEXT(immediate): sign-extend immediate to 16 bits. ZEXT(immediate): zero-extend immediate to 16 bits.

```

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
| 0 0 0 1 | DR | SR1 | 0 0 0 | SR2 |
+-----+
DR ← SR1 + SR2 also setcc()

+-----+
ADD DR, SR1, imm5 ; Addition with Immediate
| 0 0 0 1 | DR | SR1 | 1 | imm5 |
+-----+
DR ← SR1 + SEXT(imm5) also setcc()

+-----+
AND DR, SR1, SR2 ; Bit-wise AND
| 0 1 0 1 | DR | SR1 | 0 0 0 | SR2 |
+-----+
DR ← SR1 AND SR2 also setcc()

+-----+
AND DR, SR1, imm5 ; Bit-wise AND with Immediate
| 0 1 0 1 | DR | SR1 | 1 | imm5 |
+-----+
DR ← SR1 AND SEXT(imm5) also setcc()

+-----+
BRx, label (where x={n,z,p,zp,np,nz,nzp}); Branch
| 0 0 0 0 | n | z | p | PCoffset9 |
+-----+
if(GO is true) then PC ← PC' + SEXT(PCoffset9)

+-----+
JMP BaseR ; Jump
| 1 1 0 0 | 0 0 0 | BaseR | 0 0 0 0 0 0 |
+-----+
PC ← BaseR

+-----+
JSR label ; Jump to Subroutine
| 0 1 0 0 | 1 | PCoffset11 |
+-----+
R7 ← PC', PC ← PC' + SEXT(PCoffset11)

+-----+
JSRR BaseR ; Jump to Subroutine in Register
| 0 1 0 0 | 0 0 0 | BaseR | 0 0 0 0 0 0 |
+-----+
temp ← PC', PC ← BaseR, R7 ← temp

+-----+
LD DR, label ; Load PC-Relative
| 0 0 1 0 | DR | PCoffset9 |
+-----+
DR ← mem[PC' + SEXT(PCoffset9)] also setcc()

+-----+
LDI DR, label ; Load Indirect
| 1 0 1 0 | DR | PCoffset9 |
+-----+
DR ← mem[mem[PC' + SEXT(PCoffset9)]] also setcc()

+-----+
LDR DR, BaseR, offset6 ; Load Base+Offset
| 0 1 1 0 | DR | BaseR | offset6 |
+-----+
DR ← mem[BaseR + SEXT(offset6)] also setcc()

+-----+
LEA, DR, label ; Load Effective Address
| 1 1 1 0 | DR | PCoffset9 |
+-----+
DR ← PC' + SEXT(PCoffset9) also setcc()

+-----+
NOT DR, SR ; Bit-wise Complement
| 1 0 0 1 | DR | SR | 1 1 1 1 1 1 |
+-----+
DR ← NOT(SR) also setcc()

+-----+
RET ; Return from Subroutine
| 1 1 0 0 | 0 0 0 | 1 1 1 | 0 0 0 0 0 0 |
+-----+
PC ← R7

+-----+
RTI ; Return from Interrupt
| 1 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
+-----+
See textbook (2nd Ed. page 537).

+-----+
ST SR, label ; Store PC-Relative
| 0 0 1 1 | SR | PCoffset9 |
+-----+
mem[PC' + SEXT(PCoffset9)] ← SR

+-----+
STI, SR, label ; Store Indirect
| 1 0 1 1 | SR | PCoffset9 |
+-----+
mem[mem[PC' + SEXT(PCoffset9)]] ← SR

+-----+
STR SR, BaseR, offset6 ; Store Base+Offset
| 0 1 1 1 | SR | BaseR | offset6 |
+-----+
mem[BaseR + SEXT(offset6)] ← SR

+-----+
TRAP ; System Call
| 1 1 1 1 | 0 0 0 0 | trapvect8 |
+-----+
R7 ← PC', PC ← mem[ZEXT(trapvect8)]

+-----+
; Unused Opcode
| 1 1 0 1 |
+-----+
Initiate illegal opcode exception
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

```