CMSC 240 Principles of Computer Organization
Spring 2021
Exam 2

This exam contains 7 Questions on pages numbered 1-9.

Please fill in all answers in the space provided in the form. Be sure to save your filled out exam as **LastFirst.pdf (e.g. CassidyKim.pdf)**. The exam should be e-mailed before the time mentioned in the email you received with this exam. No credit will be given for exams submitted late. Your Professor will confirm the receipt of the exam by e-mail.

All resources (textbook, class notes, completed labs and assignments, etc.) are permitted, but no assistance from another person. Please, refrain from web searching for answers (Google, etc.), it will mostly lead to wasted time and possibly misleading answers. If you copy anything directly from anywhere, please provide an appropriate citation of the source.

Good Luck!

Declaration

Sign the following statement **after** you have completed the examination by typing your name in the box provided. Your exam will **not be graded** without your signature:

> I certify that my responses in this examination are solely the product of my own work and that I have fully abided by the Bryn Mawr College Academic Integrity policy and instructions stated above while taking this exam.

Name: SOLUTIONS

For administrative use only!

1	2	3	4	5	6	7	Total

Question 1 (10 points) For each of the following, either fill in the correct answer or answer **True** or **False** as appropriate:

1. Name of the basic circuit we have learned that is used to store 1-bit of information. Any pulse controlled latch

2. The unique number associated with each word in memory is called by this name. Address

3. The addressability of LC-3 is 2 bytes. (True/False) True

4. Write the size of the address space of LC-3's memory. 32768

5. The Gated D-Latch is a sequential logic circuit. (True/False) False

6. Finite State Machines are implemented using Combinational Logic Circuits. (True/False) False

7. The Finite State Machine of LC-3 is an asynchronous Finite State Machine False

8. Flip-Flops help to synchronize with the system clock. (True/False) True

9. What is another name for the Program Counter (or PC). Instruction Pointer

10. What is the word length of the LC-3 ALU? 16 bits

Question 2 (15 points) Write short answers to each of the following.

Part A: In LC-3 what are **MAR** and **MDR**? What are they used for? How many bits are there in the **MAR** and **MDR**?

MAR = Memory Address Register
MDR = Memory Data Register
While reading from memory, the address of the location to be read is first placed in the MAR. Then, the data from that memory location appears in the MDR.

While writing, the data to be stored is first placed in the MDR. Then the adddress where it is to be stored is placed in the MAR.

Part B: In LC-3 what are the condition code registers? How are they used?

N, Z, and P are the three condition codes in LC-3.
Every time a data is moved some place (registers or memory), or an operation is performed, the condition codes are set depending on the value.

Part C: In LC-3 what is the purpose of the **Instruction Register**?

Contains the fetched instruction from the memory location pointed to by the PC. This is the instruction to be decoded and executed.

Question 3 (10 points)

Part A: The LC-3 has 15 opcodes that define the instructions in its ISA. However, the instruction set charts of LC-3 (for example, see Figure 5.3 of your text) list 19 instructions. Explain.

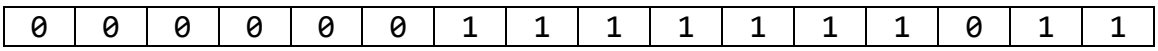
Because instructions like ADD and AND use two different addressing modes. Also, there are two Jump instructions use the same opcode. These four variations make up 15+4 = 19 instructions.

Part B: For each of the following categories, name all the LC-3 instructions that fall in that category (Select from: **ADD**, **AND**, **Branch**, **Jump**, **Load**, **NOT**, **Store**, **HALT**):

Operate Instructions	ADD, AND, NOT
Data Movement Instructions	Load, Store
Control Instructions	Branch, Jump, HALT

Question 4 (20 points) For each of the following, decode each instruction by listing the following: the opcode, all operands, the addressing mode used, and the task it performs using symbolic names (e.g. **Load**, **ADD**, **R3**, **R2**, **#11**, **R2 = R2 + 3**, etc.)):

A.



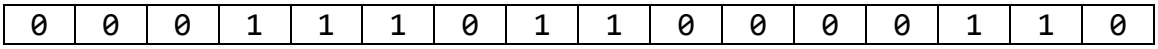
Opcode: 0000 (BR)

Operands: #-5

Addressing Mode: PC Relative mode

Task: if P then PC = PC -5

B.



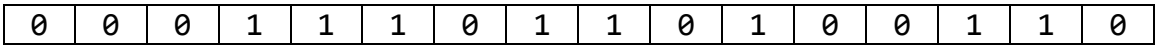
Opcode: 0001 (Add)

Operands: R6, R6, R6

Addressing Mode: Register mode

Task: R6 = R6 + R6

C.



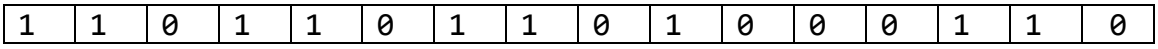
Opcode: 0001 (Add)

Operands: R6, R6, #6

Addressing Mode: Register+Immediate mode

Task: R6 = R6 + 6

D.



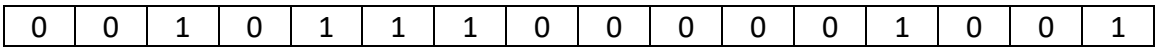
Opcode: 1101 (Reserved)

Operands: N/A

Addressing Mode: N/A

Task: Not an instruction

E.



Opcode: 0010 (Load)

Operands: R7, #9

Addressing Mode: Register+PC Relative

Task: R7 = M[PC+9]

Question 5 (10 points) Encode each of the following tasks into **an** equivalent LC-3 instruction. Where needed, the address of the current instruction is provided.:

A. **R6 = NOT(R6)**

```
1001  110  110  111111
```

B. **x600A R4 = M[x6000]**

```
0010  100 111 110 101   ; R4 = M[PC - 11]
```

C. **R3 = M[R0]**

```
0110  011  000  000000
```

D. **x600D Branch if Positive x6008**

```
0000  001 1111 1010   ; BR if P to PC-6
```

E. **R5 = R5 - 3**

```
0001  101  101 1 11101
```

Question 6 (15 points) Write a sequence of LC-3 machine language instructions to accomplish the tasks given (use comments to indicate what each instruction does):

A. R7 = R3 - R0

```
1001  110  000 111111   ; R6 = NOT(R0)
0001  110  110 1 00001  ; R6 = R6 + 1
;  Now R6 is -R0
0001  111  011 000 110  ; R7 = R3 + R6 (to get R3 - R0)
[Note: Not a good idea to change values in R3 or R0. Why?]
```

B. R7 = R6

```
0101  111  111  1  00000   ; First, set R7 =0
0001  111  111  000  110   ; Add R6 to R7

Alternately:
0001   111  110 1 00000   ; R7 = R6 + 0
```

C. R7 = R7 * 2

```
There is no multiply instruction. But multiplication by 2 can be achieved by adding the
number to itself.
0001  111  111  000  111   ; R7 = R7 + R7
```

D. Swap the contents of **R6** and **R7**.

```
; We will use R5 as temp
0001  101  110  1 00000   ; R5 = R6 + 0
0001  110  111  1 00000   ; R6 = R7 + 0
0001  111  101  1 00000   ; R7 = R5 + 0
```

E. R7 = R1 + R2 + R3

```
0001  111  001  0 00 010   ; R7 = R1 + R2
0001  111 111  0 00 011   ; R7 = R7 + R3
```

Question 7 (20 points) Write a complete LC-3 machine language program to perform integer division on two numbers, **A** and **B**. Assume that **A** is stored in location **x6000** and **B** is in **x6001**. The integer division should produce two results: a *quotient* (stored in **R0**) and a *remainder* (stored in **R1**). For example,

Suppose **A = 22** and **B = 7**. Then, after division **R0** will contain **3** and **R1** will contain **1**.

Suppose **A = 51** and **B = 9**. Then, after division **R0** will contain **5** and **R1** will contain **6**.

First, write the algorithm. Then, do register allocation. Then, write the algorithm again using registers and LC-3 equivalent instructions. Finally, code the entire program in LC3 machine language. Your program should be stored starting from memory location **x600A**.

```
Algorithm:
A = 22
B = 7
Q = 0
while A >= B do
   A = A - B
   Q = Q + 1
R = A

Use the registers: R0 = Q, R1 = R, A = R1 (why?), B = R2
Now we have:
R1 = A
R2 = B
R0 = 0
while R1 >= R2 do
   R1 = R1 - B
   R0 = R0 + 1
To optimize repeated subtraction of B, use R3 = -B

Then we have:
R1 = A
R2 = B
R3 = -B
R0 = 0
while R1 >= R2 do
   R1 = R1 + R3
   R0 = R0 + 1
```

```
Code:
0110 0000 0000 1010          ; start at x600A
0010  001 11111 0101         ; R1 = A, x6000 is -11 back
0010  010 11111 0101         ; R2 = B
1001  011  010  00000        ; R3 = -B (-R2), first R3 = NOT(R2)
0001  011  011  1 00001      ; then R3 = R3 + 1, now R3 = -R2 (B)
0101  000  000  1 00000      ; R0 = 0
; * while R1 > R2 do
   ; R4 = R1 - R2 (-R2 is in R3) i.e R4 = R1 + R3
   0001  100  001  0 00 011
   ; If negative then done, go to **
   0000 100  000 000 011 ; ** is PC+3
   0001  001  001  0 00 011   ; R1 = R1 + R3
   0001  000  000  1 00001    ; R0 = R0 + 1
   ; go to *
   0000 111 11111 1011       ; * = PC-5
; ** Done
1111 0000 0010 0101          ; HALT
```