1. Convert the following LC3 Assembly instructions to their binary representation (2 points each):

   a. \texttt{ADD R4, R3, #12}  
      \[0001 \text{100} 011 1 01100 = 0001 1000 1110 1100 = \times \text{18EC}]

   b. \texttt{LDR R2, R3, #\text{-5}}  
      \[5 = 000101 \quad \text{-5} = 111011  
      0110 010 011 111011 = 0110 0100 1111 1011 = \times 64FB]

   c. \texttt{BRNP #6}  
      \[0000 101 0 0000 0110 = 0000 1010 0000 0110 = \times 6A06]

   d. \texttt{JSR #\text{-14}}  
      \[14 = 000 0000 1110 \quad \text{-14} = 1111111010  
      0100 1 111 1111 0010 = 0100 1111 1111 0010 = \times 4FF2]

   e. \texttt{ADD R4, R4, R4}  
      \[0001 \text{100} 100 0 00 100 = 0001 1001 0000 0100 = \times 1904]

2. Dis-assemble the following LC3 binary instructions Express offsets as literal constants because no symbol table is provided. (2 points each):

   a. \texttt{0x3428}  
      \[0011 0100 0010 1000  
      0011 010 0 0010 1000 = \text{ST R2, #40}]

   b. \texttt{0x1260}  
      \[0001 0010 0110 0000  
      0001 001 001 1 0 0000 = \text{ADD R1, R1, #0}]

   c. \texttt{0x0FF7}  
      \[0000 1111 1111 0111  
      0000 111 1 1111 0111 = \text{BRNZP #-9}]

   d. \texttt{0x6280}  
      \[0110 0010 1000 0000  
      0110 001 010 0 0000 = \text{LDR R1, R2, #0}]

   e. \texttt{0xC1C0}  
      \[1100 0001 1100 0000  
      1100 000 111 0 0000 = \text{RET}]

---

Name: ________________________________
3. Describe the errors in the following lines of LC3 code (5 points each):

a. ADD R0,R1,#-23
   #-23 is too large to fit in imm5, which holds at the least, -16.

b. JSR ZERO_R7
   ...
   ZERO_R7: AND R7,R7,0
   RET
   The RET instruction depends on R7 unmodified or stored and restored after a JSR instruction. In this case, R7 is zeroed, which will cause RET to branch to location 0, rather than the correct return address.

c. ADD R3,R3,0
   BRNZ ISPOS
   HALT
   .BLKW 512
   ISPOS NOT R3,R3
   ...
   The PCoffset of label ISPOS, 512, is greater than the largest value that will fit in PCoffset9, 255, in the BR instruction.

d. ADD R6,R7,R8
   There is no R8 in LC3 assembler.
   LEA R3,MY_DATA
   MY_DATA .FILL xC222
   After the LEA instruction is executed, LC3 will try to execute xC222, which is not a valid instruction.

4. Answer the following questions in one or two sentences... (3 points each)

a. What is the Condition Code (CC) in LC3 assembler and how is it set to a value?
   The condition code is a 3 bit internal register, NZP for negative, zero, or positive; that is set by all instructions which write to a register, and used to determine if a conditional branch is taken or not.

b. Why does LC3 use both registers and addressable memory to save data?
   Main memory is slower, but virtually unlimited memory, accessible only through load and store. Registers are faster than main memory, and the ALU can access only registers, but can only hold 8 words.

c. What is the difference between Load and Store instructions?
   Load reads data from memory and saves it in a register. Store takes data from a register and saves it in memory.

d. Describe two addressing modes in LC3 assembler.
   PCoffset addressing computes the address by adding PCoffset to the PC. Base/Offset addressing computes the address by adding the Offset to the value of the base register.
5. Write the LC3 Assembler code to add all integers from 1 to 100. Do not make any assumptions about the initial value of any registers. (10 points)

```
LD R1,HUNDRED
AND R0,R0,0
LOOP ADD R0,R0,R1
ADD R1,R1,#-1
BRP LOOP
HALT
HUNDRED .fill #100
```

6. Write LC3 Assembler code to perform the function in the following pseudo-code: (Note: Assume the value of R1 is unknown at entry to your code, but the value of R2 has been set already. You may use any general purpose register in your code.) (10 points)

```
for(R1=0; R1<10; R1=R1+2) {
  if ( R2 < 50 ) then { R2 = R2 + R1 }
  else { R2 = R2 - R1 }
}

AND R1,R1,0 ; For loop initialization
LD R3,MINUSFIFTY ; Save -50 in r3
FORLOOP ADD R4,R1,#-10 ; Set CC for “for” condition
  BRZP ENDFOR ; if (r1-10>=0) we’re done here
  ADD R4,R2,R3 ; Set CC for “if” condition
  BRZP ELSE ; if (r2-50>=0) go to else block
  ADD R2,R2,R1 ; if block
  BRNZP ENDIF ; skip else block
ELSE NOT R4,R1 ; r4=-r1
  ADD R4,R4,#1; ditto
  ADD R2,R2,R4 ; else block
ENDIF ADD R1,R1,2 ; For loop iteration
  BRNZP FORLOOP ; Try it again
ENDFOR HALT
MINUSFIFTY .fill #-50
```
7. Write the LC3 assembler code for a function that multiplies the value of R3 by the value of R4, and puts the product in R5. Your function may change the values of R3 and R4. You may assume R3 is positive or zero, but R4 may be negative, zero, or positive. (10 points)

```
MULTR3R4 : AND R5,R5,0 ; Zero product
LOOP:   ADD R5,R5,R4 ; Add R4
        ADD R3,R3,-1
        BRP LOOP
        RET
```

8. Write the LC3 assembler code to compute the cross-product of two vectors of numbers. (10 points)
Assume there are five numbers in each vector. You may assume that you have access to the multiply function you wrote in problem 7. The pseudo-code to compute the cross-product is as follows:

```
cp=0
for(i=0;  i<vector_count; i=i+1) {
    cp = cp + ( vec1[i] x vec2[i] )
}
```
Assume the following values have been defined for you:

```
vec1 .fill #10
   .fill #14
   .fill #28
   .fill #36
   .fill #48
vec2 .fill #18
   .fill #23
   .fill #32
   .fill #41
   .fill #10
```

```
AND R0,R0,0 ; R0=cp=0
ADD R1,R0,5 ; R1=count ... num left
LEA R2,VEC1 ; R2->VEC1[0]
LEA R6,VEC2 ; R6->VEC2[0]
CPLOOP LDR R3,R2,0 ; R3=vec1[i]
    LDR R4,R6,0 ; R4=vec2[i]
    JSR MULTR3R4 ; R5=vec1[i] x vec2[i]
    ADD R0,R0,R5 ; cp = cp + r5 (1750)
    ADD R2,R2,1 ; r2->vec1[i+1]
    ADD R6,R6,1 ; r6->vec2[i+1]
    ADD R1,R1,-1 ; decrement count
    BRP CPLOOP ; if more left, loop
```

10. Whose birthday is it today (Thursday July 26)? (3 points)
    Henry’s!