```
7.19:         .ORIG x3005; start location
              LEA R2, DATA; takes address from x000B and offsets PC by that address, writes result to R2
              LDR R4, R2, #0; store value of address of R2 + 0 into R4 (data at address of result just written into R2 + 0  stored into R4)
LOOP          ADD R4, R4, #-3; R4: mem(R4-3), deincrements R4 by 3
              BRzp LOOP; loop if 0 or positive
             TRAP x25; stop process
DATA         .FILL x000B; any specified register will have value x000B (11)
             .END; end program
```

4 times

```
7.21:          .ORIG x3000; start location
               AND R0, R0, #0; initializing R0 to 0
               ADD R2, R0, #10; R2: mem(R0+10), stores value R0 plus 10 in R2
               LD R1, MASK; R1 is ptr, point to data starting at x8000
               LD R3, PTR1; R3 is ptr, point to data starting at x4000
LOOP           LDR R4, R3, #0; R4: mem(0 + R3), put into R4 data from address (R3+0)
              AND R4, R4, R1; AND R4 w/ R1, R4 set flag
              BRz NEXT; next line if zero
             ADD R0, R0, #1; R0: mem(R0+1), increments R0 by 1
NEXT         ADD R3, R3, #1; R3: mem(R3+1), increments ptr R3 by 1
             ADD R2, R2, #-1; R2: mem(R2-1), deincrements ptr R2 by 1
             BRp LOOP; loop if positive
             STI R0, PTR2; mem(mem(PTR2)) into R4 (storing this address into memory)
             HALT; stop
MASK        .FILL x8000; any specified register will have value in x8000
```

```
PTR1        .FILL x4000; any specified register will have value in x4000
PTR2        .FILL x5000; any specified register will have value in x5000
            .END; end program
```

See above for what the program does.

```
7.23:                 .ORIG x3000; start location
                      LD R0, PTR; R0 is ptr, point to data in x4000
                      ADD R1, R0, #0; R1: mem(R0+0), stores value of R0 + 0 in R1
AGAIN                 LDR R2, R1, #0; R4: mem(R1 + 0), put into R2 data from address R2+0
                      BRz CONT; go to "CONT" line if zero
                      ADD R1, R1, #1; R1: mem(R1+1), increments R1 by 1
                      BRnzp AGAIN; go to "AGAIN" line
CONT                  --------------(a)
LOOP                  LDR R3, R0, #0; R3: mem(0 + R0), put into R3 data from address (R0+0)
                      --------------(b)
                      NOT R4, R4; flip bits in R4 to make negative of R4, store in R4
                      ADD R4, R4, #1; R4: mem(R4+1), increments R4 by 1
                      ADD R3, R3, R4; add R3 with R4, store value in R3
                      BRnp NO; go to "NO" line if not 0
                      --------------(c)
                      --------------(d)
                      NOT R2, R0; flip bits in R0 to make negative of R0, store in R2
                      ADD R2, R2, #1; R2: mem(R2+1), increments R2 by 1
                      ADD R2, R1, R2; add R1 with R2, store value in R2
                      BRnz YES; go to "YES" line if negative or 0
                      --------------(e)
```

```
YES        AND R5, R5, #0; initialize R5 to 0

           ADD R5, R5, #1; R5: mem(R5+1), increments R5 by 1

           BRnzp DONE; if R3 = negative, 0, positive then stop program

NO         AND R5, R5, #0; initialize R5 to 0

DONE       HALT; stop

PTR        .FILL x4000; any specified register will have value in x4000

           .END; end program
```

a.:
b.:
c.:
d.:
e.:

```
7.24:      .ORIG x3000; start location

           AND R2, R2, #0; initialize R2 to 0

           ADD R2, R2, #4; R2: mem(R2+4), increments R2 by 4

LOOP       BRz DONE; halt program if R2 = 0

           ADD R2, R2, #-1; R2: mem(R2-1), deincrements ptr R2 by 1

           ADD R3, R3, R3; add R3 to itself, store value in R3 (double R3, store in R3)

           BR LOOP; loop regardless of register value

DONE       HALT; stop

           .END; end program
```

The error is that the ADD function on line 6 isn't adding the value in R2 to R3, and is instead just adding R3 to itself. To fix it, replace one of the last two R3s on that line with R2.