Self-modifying code puzzler

Write a program, **P**, in LC3 ISA machine code which alters its first two instructions by adding integers A and B to these instruction's opcodes:

instr_0.opcode <== instr_0.opcode + A

instr_1.opcode <== instr_1.opcode + B

"instr_0", is the first instruction after the data items A and B, as shown at right. "instr_1" immediately follows it. The bit fields of an instruction depend on the instruction's type; however, the opcode field for LC3 instructions is always the left-most, high-order four bits of the 16-bit instruction word. The notation, "instr_0.opcode", means the opcode field of the instruction, "instr_0".

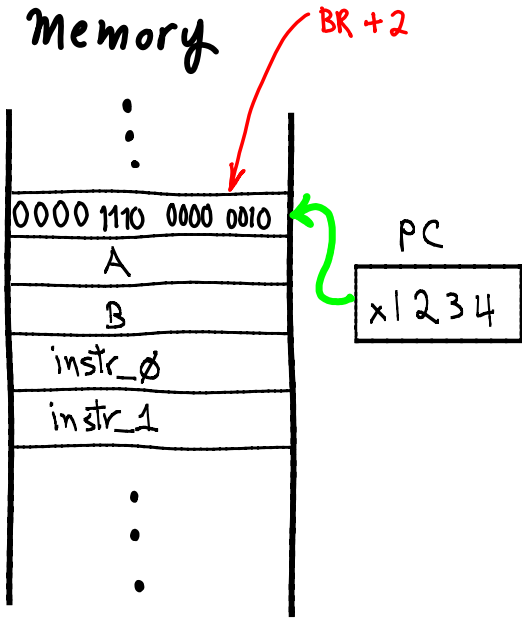
The code to do this should work correctly regardless of where the program is located in memory. Assume the PC initially contains the address of P's first instruction, which is just above A. (In the example at right, it is in the memory cell whose address is x1234.)

Don't worry about what happens when the end of the program is reached.

Shown just above the memory word holding the 16-bit integer A is an instruction showing all its 16 bits. That happens to be a BR instruction which will cause instr_0 to be the next instruction executed. The remainder of the program is assumed to follow directly after instr_1.

See questions below.

Extra Credit: write the program so that it alters all its instructions (requires looping, i.e., branches).



HINT

Use PennSim.jar and try different instructions to figure out the various parts of your program.

To change the value of a register or memory location, double click on it. Set the PC this way, too. Then click somewhere else for it to take effect. Use "Step" to execute.

Unfortunately, values are all in hex; so, you have to translate from bits to hex by hand.

Q.1 Suggest roughly what P will need to do in what order. That is, produce a rough outline of an algorithm. NB--We will not use branching. We only use the LC3's operate instructions (NOT, ADD, AND, ADDi*, ANDi*), and LC3's load/store instructions (LD, ST, LDI, STI, LDR, STR).

Notes:

* ADDi and ANDi are "immediate mode" versions of ADD and AND; that is, they have bit 5 turned on: Think, reg-mode using only a part of the IR, which gets sign-extended.

** LDR and STR are "register indirect" versions of LD and ST: Think, derefencing a pointer held in a register.

*** LDI and STI are "memory indirect" versions of LD and ST: Think, dereferencing a pointer variable held in memory, where the memory address of the variable is (PC + offset).

Q.2 The first thing we want to do is get instr_0 into a register so we can manipulate its opcode using LC3's ALU instructions. Explain what registers need to be altered, with what content, gotten from where, and in what sequential order these operations should occur. Show which instructions are needed to do this and the values of the various instruction fields. How can P get the address of instr_0? [Hint: use LEA early in P and use pointer de-referencing (register-indirect mode).

Q.3 With that in place, we need to get the data "A" into a register. Show the instructions to do that.

Q.4 At this point, both instr_0 and data A are in registers. A is a 4-bit integer expressed in 16 bits. That is, A is in the form, 000000000000wxyz, where wxyz is a 4-bit binary number. Suppose the opcode of instr_0 is 1010. Then we want to do the addition,

```
    xyzw
+ 1010
```

and have the result wind up in the opcode field of instr_0. We will ignore any carry. As above, explain the needed sequence of events, and give LC3 instructions to carry it out. Remember, the code must modify the memory content at instr_0's address.