Lec-3-HW-2

Self-modifying code puzzler

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

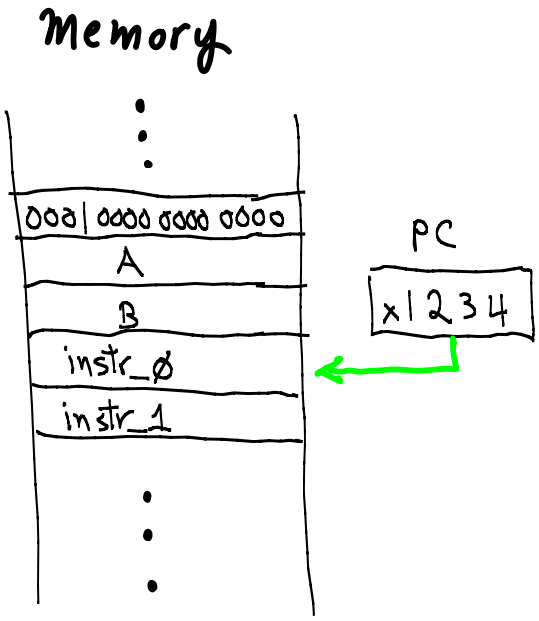
instr_0.opcode <== instr_0.opcode + A

instr_1.opcode <== instr_1.opcode + B

The notation, "instr_0", means the first instruction of the program; "instr_1" means the second instruction of the program. The bit fields of the instructions depend on the type of instruction, but the opcode field for LC3 instructions is always the left-most, high-order four bits of the 16-bits. The "instr_0.opcode" notation indicates 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 the memory word containing instr_0. (In the example at left, it is in the memory cell whose address is x1234.) Data A and B are in consecutive memory words as part of program. Don't worry about what happens after the end of the program is reached.

An example layout of the program in memory is shown at right. Shown just above the memory word holding the 16-bit integer A is an example instruction showing all its 16 bits. That happens to be an ADD instruction. The remainder of the program, P, is assumed to follow directly after instr_1.



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 be able to use branching. We can 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 the first instruction, instr_0, into a register so we can manipulate its opcode using LC3's ALU instructions. With that goal in mind, explain the sequence of events we need to have happen. That is, explain what registers need to be altered, with what content, gotten from where, in the sequential order needed. Show which instructions are needed to do this and the values of the various instruction fields.

Q.3 With that in place, we need to get the data "A" into a register. Show the instructions to do that, given that they immediately preceed P's instructions as shown above.

Q.4 At this point, both instr_0 and data A are in registers. We can assume A is a 4-bit integer expressed in 16 bits. That is, A is in the form, 000000000000xyzw, where x, y, z, and w are each 0 or 1. 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. How can P get the address of instr_0? [Hint: use LEA early in P and use pointer dereferencing (register-indirect mode).