

CSI333 Lecture 7 Strategy & Tips for Proj 3

Project 3

Program LC-3 to:

1. Subtract two numbers.
FREE! I coded this in the template file
2. Test if two numbers are equal and store a 1 in a register if they are and a 0 in the same register otherwise.
3. Shift a number one bit to the left.
4. Shift a number one bit to the left the number of times given in a register.
5. Rotate a number one bit to the left.

Project 4

1. (Program LC-3 to) count the number of ways a **pattern bit string P** occurs within a **long (target) bit string L**.
2. (Program it to) do this counting for as many P's as the user put in memory, and store each count in the word under each given P.
3. (Don't worry, it is due in 2 weeks.)

Strategy & Tips

Project 3 Continued

- 6 Rotate a number one bit to the left the number of times given in a register.
- 7 Rotate a number one bit to the right the number of times given in a register.
- 8 Shift a number to the right the number of times given in a register.

REQUIRED FORM

1. Each must be an LC-3 **function**.
2. A test call for each must be coded, beginning at 0x3000.

Explanation of Shifting and Rotating

- ▶ Left shifting illustrated.
- ▶ Digression about barrel shifting hardware. A 32-bit shifter will have 5 stages connected serially: shift of 0 or 1, shift of 0 or 2, shift of 0 or 4, shift of 0 or 8, and shift of 0 or 16. The binary code for the amount of shifting (0 to 31) controls which stages are activated.
- ▶ Left rotation illustrated in a line and on a circle.
- ▶ Right rotation can be done by left rotation by an amount you can calculate. It is similar to a right 90 degree left rotation has the same effect as a 270 degree right rotation

Keeping Track of What the Top Bit was After Shifting it Out

Note that 1-bit left rotate differs from left shift only in that the top input bit shifted out must be copied into the bottom bit of the answer.

Therefore, that top bit value must be saved after the multiply by 2 is done.

Two strategies

1. Save the info in a register or a memory location.
2. Use a conditional and two sub-task code sequences, one for each case. (Flowcharts for sequential, branching and loop forms shown from PP Chapter 6.)

Tip 1: Scout out the battlefield

Your programming environment

1. Download our template from the web.
2. Open it with LC3Edit (on Windows PCs).
3. Click Binary to LC3 object conversion, get .obj file.
4. Run simulate.exe and load that .obj file into it.

LC3 features

1. Read Ch 5, 6, and Appendix A, in PP a little at a time, and practice deploying LC3 instructions.
2. Keep a copy of the ISA chart and Appendix A handy at all times.
3. Slowly and carefully study the details of an instruction (**only**) when you need to use it.
4. Analyze by hand the examples related to your current task.

Tip 2: Protect your handiwork

Establish habits to:

1. Save current and PREVIOUS versions, especially if previous versions work partially.
2. ALWAYS (for binary programming) write comments to remember your intentions.
3. KEEP the comments consistent with the binary code. 30sec. of time to edit them saved will cause you to lose 60min. of puzzlement when you try find a bug

Tips 3 and 4: Using Space

3. Spacing out

Adjust the spaces between the 0's and 1's to express the bit fields of the current instruction (as I did in the template).

4. Plan use of memory

1. The template gives zeros in a $0x100=2^8=256$ word memory region.
2. The testing code must begin at $0x3000$ (top of template).
3. It's hard to move machine code from one address to another.
4. Code each function in DIFFERENT AREAS with unused space in-between.

Tip 5: Leaving space to fix things: **NOT IN BOOKS**

- ▶ What does LC-3 instruction 0x0000 do?

Tip 5: Leaving space to fix things: **NOT IN BOOKS**

- ▶ What does LC-3 instruction 0x0000 do?
- ▶ NOTHING!

Tip 5: Leaving space to fix things: **NOT IN BOOKS**

- ▶ What does LC-3 instruction 0x0000 do?
- ▶ NOTHING!

0x0000 = OPCODE nzp PC9offset = 0000 000 000000000

Tip 5: Leaving space to fix things: NOT IN BOOKS

- ▶ What does LC-3 instruction 0x0000 do?
- ▶ NOTHING!
0x0000 = OPCODE nzp PC9offset = 0000 000 0000000000
- ▶ Opcode 0000 means BR (branch).
PC9offset=0000000000 means branch to PC+0. 0x0000 is abbreviated by NOP (NO-OP). (The nzp condition test codes are irrelevant, but nzp=000 means NEVER BRANCH.)

Tip 5: Leaving space to fix things: NOT IN BOOKS

- ▶ What does LC-3 instruction 0x0000 do?
- ▶ NOTHING!
0x0000 = OPCODE nzp PC9offset = 0000 000 000000000
- ▶ Opcode 0000 means BR (branch).
PC9offset=000000000 means branch to PC+0. 0x0000 is abbreviated by NOP (NO-OP). (The nzp condition test codes are irrelevant, but nzp=000 means NEVER BRANCH.) **Space out your instruction with 0's so you can insert new ones very easily.**

Consult the ISA reference manual: Appendix A of PP for LC-3

We projected the page for the ADD instruction. Note the binary code, the 2 or 3 register fields, formal C-like specification of what the instruction does, and explanation in English.

Architectural Choice Example

IS making 0x0000 a NOP a good architectural idea??

Yes	No

Architectural Choice Example

IS making 0x0000 a NOP a good architectural idea??

Yes	No

Operating systems often initialize unused memory with 0. Failing memory regions (in small systems) often appear as regions of 0s (or FFF..s).

What is the shortest LC-3 loop?

0x0FFF= 0000 111 11111111

0000 means Branch

nzp=111 means branch if last data result is

neg. OR if pos. (> 0) OR if neg.

9PCoffset=11111111 = -1 means SUBTRACT 1 from
the PC

From page 528 of PP:

```
if ((n AND N) OR (z and Z) OR (p AND P))
```

```
    PC=PC^ + SEXT(PCoffset9);
```

^This is the incremented PC

What is the shortest LC-3 loop?

0x0FFF= 0000 111 11111111

0000 means Branch

nzp=111 means branch if last data result is

neg. OR if pos. (> 0) OR if neg.

9PCoffset=11111111 = -1 means SUBTRACT 1 from
the PC

From page 528 of PP:

```
if ((n AND N) OR (z and Z) OR (p AND P))
```

```
    PC=PC^ + SEXT(PCoffset9);
```

^This is the incremented PC

So, 111 means **take the branch ALWAYS**

What is the shortest LC-3 loop?

0x0FFF= 0000 111 11111111

0000 means Branch

nzp=111 means branch if last data result is

neg. OR if pos. (> 0) OR if neg.

9PCoffset=11111111 = -1 means SUBTRACT 1 from
the PC

From page 528 of PP:

```
if ((n AND N) OR (z and Z) OR (p AND P))
```

```
    PC=PC^ + SEXT(PCoffset9);
```

^This is the incremented PC

So, 111 means **take the branch ALWAYS**
and the **target** is the same LC3 word.

1. Subtract two numbers.

Strategy

1. Input numbers are in registers: **YOUR CHOICE! DOCUMENT it!**
2. Strategy: (1) Calculate 2-s complement of the number to be subtracted. (2) Add result of (1) to the other number.
3. Strategy for (1): (1.1) “flip” the bits. (1.2) Add 1.

Tactics: 1. Subtract 2 numbers

Tactics

1. Decide where to put the code. Write the comments FIRST.
2. Write the test code first. Prof. C says it must be at 0x3000.
3. Use the NOT instruction for (1.1).
4. Figure out which register to use. (It's ok to mess up the input numbers, so I used NOT R1,R1.)
5. Use the ADD instruction with an **immediate** operand of 1 for (1.2).
6. Since I wanted to use R0 for an input and output register, I had no other sensible choice: `ADD R0,R1,#1`
7. Use table in PP to edit the 0 bits

The test code.

Simple way to get test data in registers

1. Put 0 in a register with `AND Rd,Rd,#0` (This uses the immediate mode. See book for the machine code).
2. Put a number X between -32 and $+31$ in Rd with an `ADD Rd,Rd,#X` (This also uses the immediate mode.)

How to call a function

1. Use the JSR instruction to target the function body.
2. JSR **Clobbers** $R7$ by saving the **return address** in it.
3. JSR=0100 1 dddddddd
 - 3.1 Figure out the address (PC') of the NEXT word. Write it in Hex.
 - 3.2 Write the target address (TARG) in Hex.
 - 3.3 Calculate TARG-PC' in binary (2-s complement). Watch for overflow. Put 9 bits of result in dddddddd.

How to make a function return

Use the RET instruction = 1100 000 111 000000
(It is register-displacement jump with R7 and 0 displacement.)

Shift a number one bit to the left the number of times given in a register.

Use a loop!

Each round of the loop shifts left ONE bit by $X \leftarrow 2X = X + X$. Use LC-3 3-register ADD.

Loop tactics

1. Make one register Ra hold the NUMBER OF TIMES the body is YET TO RUN.
2. Test Ra for being 0. If so, you are done. If not, code the instructions one after the others,
 - 2.1 to decrement Ra (with ADD Ra, Ra, #-1)
 - 2.2 to perform the loop body.
 - 2.3 to unconditionally branch to the test instruction.
(You will have to figure out the PCoffset9 value to code that branch. It will be NEGATIVE.)

How to make a function A call a function B

1. **DANGER!** The **Caller** (first function) must **SAVE** R7, because its call will clobber R7.
2. Memory planned for A's use is a good place to save R7.
3. Learn to use ST and LD instructions for (1) Saving data in memory and (2) restoring saved data to R7.
4. Using ST and LD requires calculating a PCoffset9 value just like BR.