Reading:
Patt & Patel

Chp 6.1.2-6.1.3 (control constructs and mechanisms)

Chp 6.2.1 up through Example 1. (Debugging, using the simulator/debugger.

Chp. 7.1-7.4 (LC-3 Assembly language: instruction syntax, labels, comments, assembler directives, 2-pass assembly, object files and linking, executable images).

Also, see in docs/:

LC3-assemblySyntax.html
LC3-IntructionsSummary.html
README-LC3tools.html
LC3-assemblyCheatSheet.asm
LC3-AssemblyManualAndExamples.pdf

Problems:

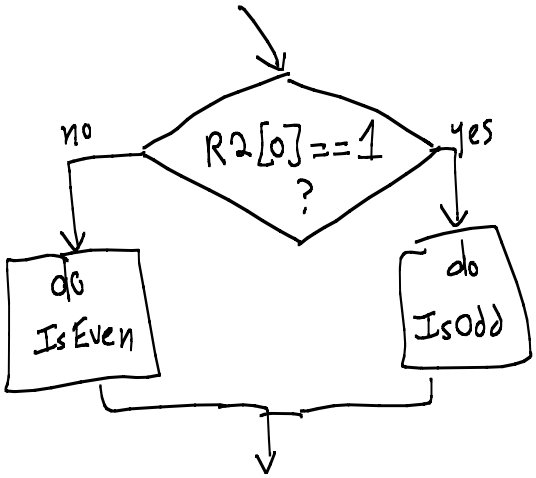
P&P, Chp. 6
6.10 (conditional, odd?)
6.12a (char echo)
6.15 (figure out instruction)

P&P, Chp. 7
7.1 (hand assemble) NB--Assume we mean "the location in the object file that will get loaded to memory location x3025" for the text's "location x3025 of the object file."
7.2 (hand assembly, symbol table entry)
7.4 (hand assembly, symbol table)
7.8 (hand assembly, program trace)
7.14 (debugging asm)
7.24 (looping control)
7.25 (too large constant in .FILL)

NB--Using a simulator/debugger in conjunction with an assembler can be helpful for these problems. Assembler is lc3as; simulator/debugger is PennSim.jar.

Chp 6/7 Solns

6.10) Content of R2 even or odd? Give machine language. Use conditional construct.




```
;-----------------------------------------------
;---- R2 even or odd
;---- If odd, bit_0 is 1; else even.
;-----------------------------------------------
        AND R0, R2, 1  ;--- result depends only on bit_0
        BRz IsEven
IsOdd:
        ...                     ;---- do odd stuff
        jmp Done
IsEven:
        ...                     ;---- do even stuff
Done:
```

(6.12) Echo one keyboard (kb) char to display.

```
;-------------------------------------------------------------------
;---- KBecho
;---- Get char from kb, send to display. We take the lazy route and use pre-coded OS functions.
;---- That's what they are there for, after all. "GETC" and "OUT" are assembler aliases
;---- for TRAP x20 and TRAP x21, respectively.
;-------------------------------------------------------------------
        TRAP x20    ;---- "GETC": Read char from kb into R0[7:0]
        TRAP x21    ;---- "OUT":   Send char in R0[7:0] to display.
```

GETC does polling until the kb controller signals it has new data from the keyboard. GETC then reads the content of kb's data register into R0. OUT polls until the display signals it can accept new data, then writes the content of R0 to the display's data register. Note that kb zeroes the high eight bits of its data register and puts an ASCII code in the low eight bits; the display reads only the low eight bits of its data register.

To poll a status register's ready bit (bit_15), read the device's status register into a register, then check whether the register's high bit is 1. We access kb's status register via memory address xFE00; its data register via memory address xFE02. For code clarity, we keep those addresses in memory words in our program:

```
KBSR:       .FILL xFE00
KBDR:       .FILL xFE02
```

and reference them as pointer variables via their labels, eg., LDI R0, KBSR

For code details of OUT and GETC, see lc3os code:

(1) using an LC3 debugger/simulator, find the addresses of the two trap routines in the trap vector table, then look in those memory locations to find the code. The trap vector for GETC is in memory location x0020; so, look at memory location x0020, write down the address found there, and then look at memory content at that address. Vector for OUT is at x0021. Realize that the code is in memory because the debugger/simulator loaded it to memory. That is something that would happen as part of system start-up, called "boot strapping" or booting for short;

(2) or see OS source code in src/lc3os.asm or versions thereof.

(6.15) Given reg and mem content, what instruction @ x3010 was executed?

There were no changes in reg content, only mem x3406 changed from x31BA to xE373. R2 contained xE373. Therefore, the instruction must have been a memory write of R2 via ST or STI or STR.

We try ST: ST R2, <offset>
We know the address must come out to be x3406 and the PC contained x3010+1; so
x3011 + <offset> = x3406
We get <offset> = x3F5, positive 2's complement, or 011 1111 0101 in as few bits as possible. That's 11 bits, but the offset field for ST is only 9 bits. ST won't work. We could try STI, but there isn't any memory location containing the address x3406; so, there's no point.

It must be STR: STR R2, R?, <offset>

STR's address arithmetic says:
R? + <offset> = x3406
where the offset is 6-bit 2's complement. The register value closest to x3406 is in R4, x33FF. The difference is +7. So, the offset is x07, or 00 0111 in 6 bits.

STR R2, R4, #7

(7.1)
What's in x3025 after assembly?

;-- in effect, code is,
.orig x3024
place: .FILL x45A7
LDI R3, place

assembly makes memory as:
x3024: x45A7
x3025: (bits for "LDI R3, -2")

result is:
x3024: 0100010110100111
x3025: 1010 011 111111110
or
x3024: x45A7
x3025: A7FE

(7.2)
ASCII LD R1, ASCII
What's in R1 after execution?

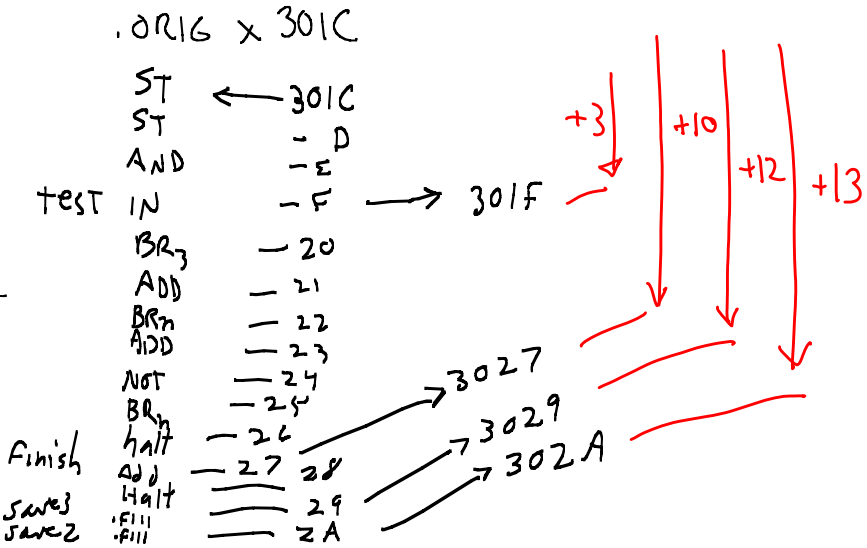
x4F08 : LD R1, <offset: PC+offset = x4F08> PC = x4F09 → offset = -1

⇒ 0010 001 1111 1111 → R1 = x23ff

-1

(7.4) ST for code?

ST	
SAVE3	3029
SAVE2	302A
TEST	301F
FINISH	3027



(7.8) Show REG_FILE content at breakpoint.
(xA400) this1: lea r0 this1
(xA401) this2: ld r1, this2
(xA402) this3: ldi r2, this5
(xA403) this4: ldr r3, r0, #2
(xA404) this5: .fill xA400

bp = xA404

R0 = x A400
R1 = x23ff ← LD R1, <-1>
0010 001 1 1111 1111
R2 = x E1ff ← LEA R0, <-1> } this5 points To xA400
1110 000 1 1111 1111
R3 = xA401 ← LDI R2, <+1> } R3 points To xA400 +2 = xA402
1010 010 0 0000 0001
all others = 0

(7.14) Assemble code

(a)

```
.ORIG x3000
    STI   R0, LABEL   → 1011 000 0 0000 0010   xB002  ①
    OUT               → 1111 0000 0010 0001    xF021  ②
    HALT              → 1111 0000 0010 0101    xF025  ③
LABEL                 → 0000 0000 0010 0101    x0025  ④
    .STRINGZ "%"      → 0000 0000 0000 0000    x0000  ⑤
    .END
```

(b) STI → LD

(c) STORES x3000 into TVT for trap x25, causes ③ to jmp to ①; ② outputs ascii NUL to display

(7.24) Fix bug in Lshift-by-4 program.

7.24 Lshift(R3, 4)

```
        .ORIG x3000
        AND  R2,R2,0
        ADD  R2,R2,4      ; R2 ← 4
Loop:   BRz  Done         ; (R2 == 0?) goto Done
A       ADD  R2,R2,-1     ; R2--
B       ADD  R3,R3,R3     ; R3 ← Lshift(R3)
        BRnzp Loop        ;
DONE    HALT
        .END
```

* "BR x" defined as "BRnzp x". See App. A.

BRz Branches on CC set by "ADD R3,R3,R3" (last reg load).
Should branch on value of R2. Swap A + B to fix.

7.25 .FILL xFF004

Assembler complains it cannot represent this value: it is 20-bits, and cannot fit into 16-bit word.