Lec-7-HW-1-LC3-ISA-SOLN

PP, Chp 5, problems:

5.4 (#address bits, PC-relative offset size and value)
5.5 (addressing modes and operand locations)
5.7 (largest pos. immed. value)
5.9 (cc+br = nop?)
5.10 (given instr. bits, instr. diff?)
5.11 (immed. data limits)
5.13 (reg-reg transfer, sub, cc set, cc codes, clear reg)
5.14 (OR)
5.18 (#mem accesses LDR, STI, TRAP)
5.21 (#trap routines)

PP, Chp 6, problems:

6.4 (prog. to compare R1 and R2: GT, EQ, LT)

Reading:
PP, Chp. 5: The LC-3 ISA, datapath, and controller.
PP, Chp. 6.2: Using the LC-3 simulator

(NB--PP describe their Simulate.exe, which is very close to what we are using, PennSim.jar. The major difference is that PennSim will not execute an instruction with all zeroes, which is a flaw because such an instruction is a perfectly valid LC3 branch instruction.)

Chp 5/6 Sol'ns

(5.4) memory = 256 cells @ 16-bits

a. bits for 256 addresses
b. bits for pc-rel jump +- 20
c. BR @ 3: offset to 10?

(a.) address bits ⇒ $2^x = 256$, $x = 8$

(b.) pc-relative jmp ±20 : $2^x \geq \pm 20 \rightarrow \pm 32 \rightarrow \pm 2^5$

5-bits unsigned ⇒ 6 bits for 2's compl.

(c)

addr	content
00000011	BR <offset to loc. 10>
⋮	
00001010	

PC = 3 ⇒ PC+1 = 4

offset = +6

(5.5)

a. addr mode?
b. 3 places
c. lc3 5 modes
d. ADD in 5.1.2

(a) mode: method of finding operand data

(b) where: in IR (instruction), in Reg, in memory (in PC, in PSR)

(c) modes LC3: immediate, reg, direct mem, indirect memory, base-offset: dbl indirect

(d) reg

5.7 5-bit 2's comp. ⇒ 01111 ↑ pos. → $2^3+2^2+2^1+2^0$

largest +value in ADD?

⋮

00000 (0) $=(10000 - 1) = 16-1 = 15$

11111 (−1)

⋮ ↓ neg.

10000

5.9 is NOP?

(a) ADD R1, R1, #0 → ok?, but sets CC

(b) BR nzp <+1> → no, jumps to +2 from instr. (PC++)

(c) BR <+0> → ok, can not take BR (but would go same place anyway).

How different?

5.10 BR nzp <offset-9 = 101010101> } both offsets = as neg nums.

JSR <offset-11 = 11101010101>

BR always taken: both jump same location, but JSR loads R7 (and CC?)

1-instr sub 20?

5.11 Immed value for ADD is 5-bits ⇒ +15, −16, not enough bits for −20.

How?

5.13 (a) R3←R2 : ADD R3, R2, #0

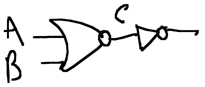
(b) R1 ← R2−R3: NOT R3, R3 ; ADD R3, R3, #1 ; ADD R1, R2, R3

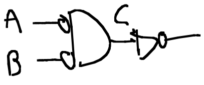
(c) set CC on R1: ADD R1, R1, #0 (or use AND)

(d) NZP = 110? NO, N ≠ Z BUS[15] must be 0 for P = 1, but must be 1 for N = 1.

(e) R2 ← 0 : AND R2, R2, #0

Do OR? fill in steps 2, 4

5.14 OR = NOT (NOR) → 

→ 

⇒ NOT A ; NOT B ; AND A, B ; NOT C

NOT R4, R1 $(R4 = \bar{A})$

→ NOT R5, R2 $(R5 = \bar{B})$

AND R6, R5, R4 $(R6 = C)$

→ NOT R3, R6 $(R3 \leftarrow \bar{C})$

5.18 # mem access? LDR, STI, TRAP

mem. acresses

$$PC \rightarrow \boxed{\text{LDR DR, BASER, offset-6}} : \left\{ \begin{array}{l} IR \leftarrow M[PC] \\ DR \leftarrow M[\text{BaseR} + \text{offset-6}] \end{array} \right\} \Rightarrow 2$$

$$\boxed{\text{STI SR, offset-9}} : \left\{ \begin{array}{l} IR \leftarrow M[PC] \\ MAR \leftarrow M[PC + \text{offset-9}] \\ SR \rightarrow M[M[PC + \text{offset-9}]] \end{array} \right\} \Rightarrow 3$$

$$\boxed{\text{TRAP vector}} : \left\{ \begin{array}{l} IR \leftarrow M[PC] \\ PC \leftarrow M[IR[\text{Vector}]] \end{array} \right\} \Rightarrow 2$$

5.21 # max TRAP routines?

1 trap per vector, 8-bit vector $\Rightarrow 2^8 = 256$ vectors $\Rightarrow$ 256 TRAPS

6.4

```
;---------------------------------------------------------------------------
;---- Set R0:
;----                R1 > R2 :  R0 <==  1
;----                R1 = R2 :  R0 <==  0
;----                R1 < R2 :  R0 <== -1
;----
;---- Algorithm in pseudo-code:
;----
;----    R0 <== (R1 - R2)
;----    switch( R0 )
;----        case N: R0 <== -1
;----        case Z: R0 <==  0
;----        case P: R0 <==  1
;----
;---------------------------------------------------------------------------
;----    R0 <== (R1 - R2)
;---------------------------------------------------------------------------
          NOT R2, R2        ;---- 2's comp R2
          ADD R2, R2, 1     ;---- 2's comp R2
          ADD R0, R1, R2  ;--- R0 <== R1 - R2

;---------------------------------------------------------------------------
;----    switch( R0 )
;---------------------------------------------------------------------------
          BRn neg
          BRz zero
          BRp pos

;---------------------------------------------------------------------------
;----    cases
;---------------------------------------------------------------------------
neg:
          AND R0, R0, 0
          ADD R0, R0, -1    ;--- R0 <== -1
          BRnzp done

zero:
          AND R0, R0, 0     ;--- R0 <== 0
          BRnzp done

pos:
          AND R0, R0, 0
          ADD R0, R0, 1     ;--- R0 <== 1
          BRnzp done


done:                         ;---- endSwitch
```

Sub in LC3   ADD, NOT

R3 ← R1 − R2   2's comp   not
                              ADD +1

```
NOT R2, R2
ADD R2, R2, #1
ADD R3, R1, R2
--------------
BrP Pos
BrZ Zero
BrN Neg

Pos:
    BRnzp done
Zero:
    BR done
Neg:
    BR done
done:
```

R1, R2 → { +1 if R1 > R2
           0   R1 = R2
           −1   R1 < R2 }

R1 − R2 result is P → R1 > R2
                  Z → R1 = R2
                  N → R1 < R2

R3 ← +1

```
AND R3, R3, #0
ADD R3, R3, #1
```

AND

```
AND
ADD R3, R3, #-1
```