F.5 Chapter 5 Solutions

5.1 (a) ADD
  - operate
  - register addressing for destination and source 1
  - register or immediate addressing for source 2

  (b) JMP
  - control
  - register addressing

  (c) LEA
  - data movement
  - immediate addressing

  (d) NOT
  - operate
  - register addressing

5.2 The MDR is 64 bits. The statement does not tell anything about the size of the MAR.

5.3 Sentinel. It is a special element which is not part of the set of allowable inputs and indicates the end of data.

5.4 (a) 8 bits
  (b) 6
  (c) 6

5.5 (a) Addressing mode: mechanism for specifying where an operand is located.
  (b) An instruction's operands are located as an immediate value, in a register, or in memory.
  (c) The 5 are: immediate, register, direct memory address, indirect memory address, base + offset address. An immediate operand is located in the instruction. A register operand is located in a register (R0 - R7). A direct memory address, indirect memory address and base + offset address all refer to operands locate in memory.
  (d) Add R2, R0, R1 => register addressing mode.

5.6 (a) 0101 011 010 1 00100
  AND R3, R2, #4
  (b) 0101 011 010 1 01100
  AND R3, R2, #12
  (c) 1001 011 010 111111
  NOT R3,R2 if zero, no machine is busy.
  (d) We cannot do this in only one instruction. We'd need to do an AND with 0000 0000 0100 0000, since the state of machine 6 is in bit [6:6]. This is impossible with the 5-bit immediate value. We could use a second instruction to load this value into a register, and then perform the AND.

5.7 01111 (decimal 15)

5.8 Increasing the number of registers to 32 will need 5 bits to denote the register number. Now, the minimum number of bits needed for the ADD instruction will be 4 (for the opcode) + 3 registers * 5 bits = 19 bits. This cannot fit in the 16-bits allocated for an lc-3 instruction.

5.9 (a) Add R1, R1, #0 => differs from a NOP in that it sets the CC's.

(b) BRnzp #1 => Unconditionally branches to one after the next address in the PC. Therefore no, this instruction is not the same as NOP.

(c) Branch that is never taken. Yes same as NOP.

5.10 A: BRnzp -171
B: JSR -171

Both A and B result in the PC being changed to (PC+1)-171.

However, B saves the linkage information in R7 and A does not affect R7.

5.11 No. We cannot do it in a single instruction as the smallest representable integer with the 5 bits available for the immediate field in the ADD instruction is -16. However this could be done in two instructions.

5.12 If R0[15] and R1[15] equal 0, but R2[15] equals 1, the result can be trusted. The other case causes an overflow, this one doesnt.

5.13 (a) 0001 011 010 1 00000 (ADD R3, R2, #0)

(b)
```
1001 011 011 111111  (NOT R3, R3 )
0001 011 011 1 00001 (ADD R3, R3, #1 )
0001 001 010 0 00011 (ADD R1, R2, R3 )
```

(c) 0001 001 001 1 00000 (ADD R1, R1, #0)
or
0101 001 001 1 11111 (AND R1, R1, #-1)

(d) Can't happen. The condition where N=1, Z=1 and P=0 would require the contents of a register to be both negative and zero.

(e) 0101 010 010 1 00000 (AND R2, R2, #0)

5.14 (2) : 1001 101 010 111111
(4) : 1001 011 110 111111

5.15
```
1110 001 000100000  ( LEA R1, 0x20    )  R1 <- 0x3121
0010 010 000100000  ( LD R2,  0x20    )  R2 <- Mem[0x3122] = 0x4566
1010 011 000100001  ( LDI R3, 0x20    )  R3 <- Mem[Mem[0x3123]] = 0xabcd
0110 100 010 000001 ( LDR R4, R2, 0x1 )  R4 <- Mem[R2 + 0x1] = 0xabcd
1111 0000 0010 0101 ( TRAp 0x25       )
```

5.16 (a) PC-relative mode

(b) Indirect Mode

(c) Base+offset mode

5.17 (a) LD: two, once to fetch the instruction, once to fetch the data.

(b) LDI: three, once to fetch the instruction, once to fetch the data address, and once to fetch the data.

(c) LEA: once, only to fetch the instruction.

5.18 LDR - 2 memory accesses
STI - 3 memory accesses
TRAP - 2 memory accesses

5.19 PC-64 to PC+63. The PC value used here is the incremented PC value.

5.20 7 bits will be required for the PC-relative offset.

5.21 The Trap instruction provides 8 bits for a trap vector. That means there could be 2^8 = 256 trap routines.

5.22 Please correct the LC-3 instructions to read:

```
x3010          1110 0110 0011 1111
x3011          0110 1000 1100 0000
x3012          0110 1101 0000 0000
```

These instructions cause the value x123B to be loaded into R6.

These three instructions could be replaced by the following instruction at location x3010:

```
x3010          1010 1100 0011 1111
```

5.23

```
x30ff 1110 0010 0000 0001  (LEA R1, #1) R1 <- 0x3101
x3100 0110 010 001 00 0010 (LDR R2, R1, #2) R2 <- 0x1482
x3101 1111 0000 0010 0101  (TRAP 0x25)
x3102 0001 0100 0100 0001
x3103 0001 0100 1000 0010
```

5.24 The largest address this instruction can load from is x4011 + x1F = x4030. This is specified by the following instruction: LDR R5, R4, x1F.

If the LDR offset were zero-extended, the largest address the LDR instruction can load from is x4011 + x3F = 0x4050 and the smallest address is x4011.

5.25

```
1001 100 011 111111  ;(NOT R4, R3)
0001 100 100 1 00001 ;(ADD R4, R4, #1)
0001 001 100 0 00 010 ;(ADD R1, R4, R2)
0000 010 000000101   ; (BRz Done)
0000 100 000000001   ; (BRn Reg3)
0000 001 000000010   ; (BRp Reg2)
0001 001 011 1 00000 ; (Reg3 ADD R1, R3, #0)
```

```
0000 111 000000001   ; (BRnzp Done)
0101 001 010 1 00000 ; (Reg2 ADD R1, R2, #0)
1111 0000 0010 0101  ; (Done TRAP 0x25)
```

5.26 NOTE: Please refer to the errata for the new problem statement.

(a) Yes, there is a problem. With 5 bits of opcode, and 4 bits each for DR and SR1 registers, and one steering bit to differentiate immediate vs. a second source register, there are only 2 bits left for the second source register. Consequently, ADD and AND can not be specified as in the original LC-3.

One possible way around this is to use the DR field to specify both the destination and one of the source registers, as are done in many ISAs, the x86, for example. If we adopted this approach (called a 2-address machine), we could specify the ADD and AND format as:

```
 -----------------------------------------------
|  '  '  '  '  |  '  '  '  | |  '  '  '  '  '  |
|    opcode        DR      I   imm or SR        |
|_______________________________________________|
```

(b) 64K = 2^{16} means 16 bits per address

(c) Since we shift the trap vector left 5 bits before zero-extending, a minimum of 32 bytes are allocated to each trap routine. Since we wish to accommodate 2^7 trap routines, a minimum of 4KB of memory is required.

(d) Look at the Add instruction to discover the size of the immediate:

```
[opcode] [destreg] [srcreg] [steeringbit] [imm]
   4b        2b       2b          1b          ?b
```

This leaves 16 - 4 - 2 - 2 - 1 = 7 bits for the immediate. The largest immediate value then is 2^6 - 1 = 63.

5.27 Four different values: xAAAA, x30F4, x0000, x0005

5.28 Instructions at x3002 and x3003 below replace the store indirect instruction in the original code. Note that as a new instruction had to be inserted, the offsets increased by 1.

```
x3000 0010 0000 0000 0011
x3001 0010 0010 0000 0011
x3002 0111 0000 0100 0000
x3003 1111 0000 0010 0101
x3004 0000 0000 0100 1000
x3005 1111 0011 1111 1111
```

5.29 (a) LDR R2, R1, #0 ;load R2 with contents of location pointed to by R1
STR R2, R0, #0 ;store those contents into location pointed to by R0

(b) The constituent micro-ops are:
MAR $<-$ SR
MDR $<-$ Mem[MAR]
MAR $<-$ DR
Mem[MAR] $<-$ MDR

5.30 If the conditional branch is taken, it is known that R0 and R1 must contain the same value.

5.31 0x1000: 0001 101 000 1 11000

5.32 Please correct the LC-3 instructions to read:

```
x3050          0000 0010 0000 0010
x3051          0101 0000 0010 0000
x3052          0000 1110 0000 0010
x3053          0101 0000 0010 0000
x3054          0001 0000 0011 1111
```

The resulting condition codes are N=1, Z=0, P=0.

5.33 It can be inferred that R5 has exactly 5 of the lower 8 bits = 1.

5.34 The Reg File and the ALU in Fig 5.18 implement the NOT instruction, alongwith NZP and the logic which goes with it.

5.35 The IR, SEXT unit, SR2MUX, Reg File and ALU implement the ADD instruction, alongwith NZP and the logic which goes with it.

5.36 Memory, MDR, MAR, IR, PC, Reg File, the SEXT unit connected to IR[8:0], ADDR2MUX, ADDR1MUX set to PC, alongwith the ADDER they connect to, and MAXMUX and GateMARMUX implement the LD instruction, alongwith NZP and the logic which goes with it.

5.37 Memory, MDR, MAR, IR, PC, Reg File, the SEXT unit connected to IR[8:0], ADDR2MUX, ADDR1MUX set to PC, alongwith the ADDER they connect to, and MAXMUX and GateMARMUX implement the LDI instruction, alongwith NZP and the logic which goes with it.

5.38 Memory, MDR, MAR, IR, Reg File, the SEXT unit connected to IR[5:0], ADDR2MUX is set to 0, ADDR1MUX is set to SR1 out, alongwith the ADDER they connect to, and MAXMUX and GateMARMUX implement the LDR instruction, alongwith NZP and the logic which goes with it.

5.39 IR, PC, Reg File, the SEXT unit connected to IR[8:0], ADDR2MUX, ADDR1MUX set to PC, alongwith the ADDER they connect to, and MAXMUX and GateMARMUX implement the LEA instruction, alongwith NZP and the logic which goes with it.

5.40 The signal A indicates if the instruction in IR is a taken branch instruction.

5.41 (a) Y is the P Condition code.

(b) Yes. The error is that the logic should not have the logic gate A. X should be one whever the opcode field of the IR matches the opcodes which change the condition code registers. The problem is that X is 1 for the BR opcode (0000) in the given logic.

5.42 (d) MUL is the most useful instruction out of the choices given. All the others can be accomplished using it or existing instructions.

(a) MOVE can be accomplished using either the ADD or AND instructions.
(b) NAND can be accomplished by doing an AND followed by a NOT.
(c) SHFL can be accomplished by using MUL.