F.6 Chapter 6 Solutions

6.1 Yes, for example, an iterative block where the test condition remains true for each iteration. This procedure will never end and is therefore not finite and not an algorithm. The following is an example of a procedure that isn't an algorithm:

```
x3000 0101 000 000 1 00000 ( LOOP AND  R0, R0, #0 )
x3001 0000 010 111111110   (      BRz LOOP )
```

This is not an algorithm because the branch instruction is always taken and the program loops indefinitely.

6.2 Problem Statement: Subtract the value in R1 from the value in R2 and store the result in R0.

The following is the systematic decomposition of the problem concluding with an LC-3 program solving the problem.

(a) `Start -> Subtract R1 from R2; place result in R0 -> End`

(b) `Start -> Compute the complement of the value in R1 -> Add this result to R2 and store the result in R0 -> End`

(c) `Start -> Negate R1 -> Add 1 to R1 -> Add R1 and R2; store result in R0 -> End`

(d)
```
1001 001 001 111111 ( NOT R1, R1 )
0001 001 001 1 00001 ( ADD R1, R1, #1 )
0001 000 001 0 00 010 ( ADD R0, R1, R2 )
```

6.3 The following program uses DeMorgan's Law to set the appropriate bits of the machine busy register.

```
x3000 1010000000001110 (   LDI R0, S )
x3001 1010001000001110 (   LDI R1, I )
x3002 0101010010100000 (   AND R2, R2, #0 )
x3003 0001010010100001 (   ADD R2, R2, #1 )
x3004 0001001001111111 ( L ADD R1, R1, #-1 )
x3005 0000100000000010 (   BRn D )
x3006 0001010010000010 (   ADD R2, R2, R2 )
x3007 0000111111111100 (   BRnzp L )
x3008 0001001010100000 ( D ADD R1, R2, #0 )
x3009 1001000000111111 (   NOT R0, R0 )
x300a 1001001001111111 (   NOT R1, R1 )
x300b 0101000000000001 (   AND R0, R0, R1 )
x300c 1001000000111111 (   NOT R0, R0 )
x300d 1011000000000001 (   STI R0, S  )
```

```
x300e 1111000000100101 (   TRAP x25 )
x300e 0100000000000001 ( S .FILL x4001 )
x300f 0100000000000000 ( I .FILL x4000 )
```

6.4
```
x3000 0101000000100000 (     AND R0, R0, #0 )
x3001 1001011001111111 (     NOT R3, R1     )
x3002 0001011011100001 (     ADD R3, R3, #1 )
x3003 0001011011000010 (     ADD R3, R3, R2 )
x3004 0000010000000100 (     BRz done       )
x3005 0000100000000010 (     BRn neg        )
x3006 0001000000111111 (     ADD R0, R0, #-1)
x3007 0000111000000001 (     BRnzp done     )
x3008 0001000000100001 ( neg ADD R0, R0, #1 )
x3009 1111000000100101 (done halt           )
```

6.5 The three additions of 88 + 88 + 88 requires fewer steps to complete than the eighty eight additions of 3 + 3 + ... + 3. Because 88 + 88 + 88 requires fewer instructions to complete, it is faster and therefore preferable.

6.6 Problem Statement: Multiply two numbers together using repeated addition. Make the procedure efficient by comparing the two numbers per the result of Problem 6.4.

```
Start -> Multiply two numbers; put result in R3 -> End
Start -> Compare two numbers -> Multiply two numbers
by repeated addition;
Put result in R3 -> End
Start -> Compare two numbers -> Put smaller number
in R2, larger number in R1 -> Add R1 to itself while
decrementing R2 -> Store result of repeated addition
to R0 -> End
```

The program compares the numbers located in memory locations x3010 and x3011, places the larger of the two in R1 and the smaller in R2, then repeatedly adds R1 to itself R2 times. The program stores the result in register R3.

```
x3000 0010110000010000 ( LD R6, A )
x3001 0010111000010000 ( LD R7, B )
x3002 1001101110111111 ( NOT R5, R6 )
x3003 0001101101100001 ( ADD R5, R5, #1 )
x3004 0001101111000101 ( ADD R5, R7, R5 )
x3005 0000110000000011 ( BRnz ABIG )
x3006 0001001111100000 ( ADD R1, R7, #0 )
x3007 0001010110100000 ( ADD R2, R6, #0 )
x3008 0000111000000010 ( BRnzp NEXT )
x3009 0001001110100000 ( ABIG ADD R1, R6, #0 )
x300a 0001010111100000 ( ADD R2, R7, #0 )
```

```
x300b 0101000000100000 ( NEXT AND R0, R0, #0 )
x300c 0001000000000001 ( LOOP ADD R0, R0, R1 )
x300d 0001010010111111 ( ADD R2, R2, #-1 )
x300e 0000001111111101 ( BRp LOOP )
x300f 0001011000100000 ( ADD R3, R0, #0)
x3010 1111000000100101 ( TRAP x25 )
x3011 0000000001011000 ( A .FILL #88 )
x3012 0000000000000011 ( B .FILL #3 )
```

6.7 This program adds together the corresponding elements of two lists of numbers (vector addition). One list starts at address x300e and the other starts at address x3013. The program finds the length of the two lists at memory location x3018. The first element of the first list is added to the first element of the second list and the result is stored back to the first element of the first list; the second element of the first list is added to the second element of the second list and the result is stored back to the second element of the first list; and so on.

6.8 Were R2 not initially cleared, a specious count would be computed by the program because R2 would initially contain garbage. After running, R2 would equal the difference between the number of occurances and whatever random initial value R2 happened to contain.

6.9
```
x3100 0010 000 0 0000 0101 (   LD R0, Z   )
x3101 0010 001 0 0000 0101 (   LD R1, C   )
x3102 1111 0000 0010 0001  ( L TRAP x21   )
x3103 0001 001 001 1 11111 (   ADD R1, R1, #-1 )
x3104 0000 001 1 1111 1101 (   BRp L      )
x3105 1111 0000 0010 0101  (   TRAP x25   )
x3106 0000 0000 0101 1010  ( Z .FILL x5A  )
x3107 0000 0000 0110 0100  ( C .FILL #100 )
```

6.10 This program tests whether R2 is even or odd. This program branches to x3110 if the number in R2 is even and branches to x3120 if the number in R2 is odd.

```
x3100 0101 000 010 1 00001 ( AND R0, R2, #1 )
x3101 0000 010 0 0000 1110 ( BRz Even )
x3102 0000 101 0 0001 1101 ( BRnp Odd )
```

6.11 This program increments each number in the list of numbers starting at address A and ending at address B. The program tells when it's done by comparing the last address it loaded data from with the address B. When the two addresses are equal, the program stops incrementing data values.

```
x3000 0010 000 011111111    (      LD R0, x3100 )
x3001 0010 001 011111111    (      LD R1, x3101 )
x3002 0001 001 001 1 00001  (      ADD R1, R1, #1 )
x3003 1001 001 001 111111   (      NOT R1, R1 )
x3004 0001 001 001 1 00001  (      ADD R1, R1 #1 )
```

```
x3005 0001 011 000 0 00 001 ( l    ADD R3, R0, R1 )
x3006 0000 010 000000101    (      BRz Done )
x3007 0110 010 000 000000   (      LDR R2, R0, #0 )
x3008 0001 010010100001     (      ADD R2, R2, #1 )
x3009 0111 010000000000     (      STR R2, R0, #0 )
x300a 0001 000000100001     (      ADD R0, R0, #1 )
x300b 0000 111111111001     (      BRnzp l )
x300c 1111 000000100101     ( Done HALT )
```

6.12 (a)
```
x3100 1111 0000 00100011 ( IN )
x3101 1111 0000 00100001 ( OUT )
x3102 0000 111 111111101 ( BRnzp x3100 )
```

(b)
```
x3000 0010 001 00001 0001  ( LD r1, nl )
x3001 1001 001 001 111111  ( NOT r1, r1 )
x3002 0001 001 001 1 00001 ( ADD r1, r1, #1 )
x3003 0010 010 0 0000 1111 ( LD r2, strg )
x3004 1111 0000 00100011   (iput IN )
x3005 0111 000 010 000000  ( STR r0, r2, #0 )
x3006 0001 011 001 0 00 000( ADD r3, r1, r0 )
x3007 0000 010 0 0000 0010 ( BRz oput )
x3008 0001 010 010 1 00001 ( ADD r2, r2, #1 )
x3009 0000 111 1 1111 1010 ( BRnzp iput )
x300a 0010 010 0 0000 1000 (oput LD r2, strg )
x300b 0110 000 010 000000  (oputl LDR r0, r2, #0 )
x300c 1111 0000 00100001   ( OUT )
x300d 0001 011 001 0 00 000 ( ADD r3, r1, r0 )
x300e 0000 010 0 0000 0010  ( BRz done )
x300f 0001 010 010 1 00001  ( ADD r2, r2, #1 )
x3010 0000 111 1 1111 1010  ( BRnzp oputl )
x3011 1111 0000 00100101    (done HALT )
x3012 0000 0000 0000 1010   (nl .FILL x0A )
x3013 0100 0000 0000 0000   (strg .FILL x4000 )
```

6.13 Memory location x3011 holds the number to be right shifted. The strategy here is to implement a one bit right shift by shifting to the left 15 bits. The most significant bit must be carried back to the least significant bit when it's shifted out (a circular left shift). The data to be shifted is stored at x3013. R1 is a counter to keep track of how many left shifts remain to be done.

```
x3000 0010000000010010 (      LD R0, NUM )
x3001 0101001001100000 (      AND R1, R1, #0 )
x3002 0001001001101111 (      ADD R1, R1, #15 )
x3003 0001000000100000 ( LOOP ADD R0, R0, #0 )
x3004 0000100000000001 (      BRn NEG )
x3005 0000001000000101 (      BRp POS )
```

```
x3006 0001000000000000 ( NEG  ADD R0, R0, R0 )
x3007 0001000000100001 (      ADD R0, R0, #1 )
x3008 0001001001111111 (      ADD R1, R1, #-1 )
x3009 0000110000000101 (      BRnz DONE )
x300a 0000111111111000 (      BRnzp LOOP )
x300b 0001000000000000 ( POS  ADD R0, R0, R0 )
x300c 0001001001111111 (      ADD R1, R1, #-1 )
x300d 0000110000000001 (      BRnz DONE )
x300e 0000111111110100 (      BRnzp LOOP )
x300f 0010001000000100 (DONE  LD R1, MASK )
x3010 0101000000000001 (      AND R0, R0, R1 )
x3011 0011000000000001 (      ST R0, NUM )
x3012 1111000000100101 (      HALT )
x3013 1000010000100001 ( NUM  .FILL x8421 )
x3014 0111111111111111 ( MASK .FILL x7FFF )
```

6.14
```
0x3000 0101 010 010 1 00000 ( AND R2, R2, #0 )
0x3001 0001 001 001 1 11111 ( ADD R1, R1, #-1 )
0x3002 0001 001 001 1 11111 ( ADD R1, R1, #-1 )
0x3003 0001 001 001 1 11111 ( ADD R1, R1, #-1 )
0x3004 0000 100 000000010 ( BRn x3007 )
0x3005 0001 010 010 1 00001 ( ADD R2, R2, #1 )
0x3006 0000 111 111111010 ( BRnzp x3001 )
0x3007 1111 0000 00100101 ( TRAP 0x25 )
```

The possible values for R1 are: 9, 10, and 11.

6.15
```
0111 010 100 000111 ( STR R2, R4, #7 )
```

6.16
```
x3000 0010 000 0 0000 0100 ( LD R0, #4 )
x3002 1011 000 0 0000 0011 ( STI R0, #3 )
x3003 1111 0000 0010 0001  ( TRAP x21 )
x3006 0100 0000 0000 0001
```

6.17
```
JSR #15
```

6.18
```
x3000 0101 0000 0010 0000 (       ADD R0, R0, #0 )
x3001 1010 0010 0000 1011 (       LDI R1, PDIVIDEND )
x3002 1010 0100 0000 1011 (       LDI r2, PDIVISOR  )
x3003 1001 0110 1011 1111 (       NOT R3, R2 )
x3004 0001 0110 1110 0001 (       ADD R3, R3, #1 )
x3005 0001 0010 0100 0011 ( LOOP  ADD R1, R1, R3 )
x3006 0000 1000 0000 0010 (       BRn REMN )
x3007 0001 0000 0010 0001 (       ADD R0, R0, #1 )
x3008 0000 1111 1111 1100 (   BRnzp LOOP )
x3009 0001 0010 0100 0010 ( REMN  ADD R1, R1, R2
x300a 1011 0000 0000 0100 (       STI R0, PQUO )
```

```
x300b 1011 0010 0000 0100 (       STI R1, PREM )
x300c 1111 0000 0010 0101 (       HALT )
x300d 0100 0000 0000 0000 ( PDIVIDEND .FILL x4000)
x300e 0100 0000 0000 0001 ( PDIVISOR  .FILL x4001)
x300f 0101 0000 0000 0000 ( PQUO  .FILL x5000)
x3010 0101 0000 0000 0001 ( PREM  .FILL x5001)
```

6.19 The bugs are:

1. The instruction at x3000 should be 0010 0000 0000 1010
2. The instruction at x3004 should be 0001 0100 1010 0100
3. The instruction at x3008 should be 0000 1111 1111 1001
4. The instruction at x3009 should be 0111 0100 0100 0000

6.20 Dividing signed numbers is more complicated than dividing positive numbers. The following rule is followed :
Dividend = Quotient x Divisor + Remainder

The sign of the quotient is set to MINUS if the divisor and the dividend are of different signs.The sign of the remainder is set according to the sign of the dividend.

```
x3000 0101 0000 0010 0000 (     and r0, r0, #0 ; quotient)
x3001 0101 1001 0010 0000 (     and r4, r4, #0 )
x3002 0101 1011 0110 0000 (     and r5, r5, #0 )
x3003 1010 0010 0001 1110 (     ldi r1, pnuma )
x3004 0000 1000 0000 0001 ( brn aneg )
x3005 0000 0110 0000 0100 (  brzp bld )
x3006 0001 1001 0010 0001 (aneg add r4, r4, #1 )
x3007 0001 1011 0110 0001 ( add r5, r5, #1 )
x3008 1001 0010 0111 1111 ( not r1, r1 )
x3009 0001 0010 0110 0001 (     add r1, r1, #1 )
x300a 1010 0100 0001 1000 ( bld ldi r2, pnumb )
x300b 0000 1000 0000 0001 ( brn bneg )
x300c 0000 0110 0000 0011 ( brzp cont )
x300d 0001 1001 0010 0001 (bneg add r4, r4, #1)
x300e 1001 0100 1011 1111 ( not r2, r2 )
x300f 0001 0100 1010 0001 ( add r2, r2, #1 )
x3010 1001 0110 1011 1111 ( cont not r3, r2 )
x3011 0001 0110 1110 0001 ( add r3, r3, #1 )
x3012 0001 0010 0100 0011 (loop add r1, r1, r3 )
x3013 0000 1000 0000 0010 ( brn remn )
x3014 0001 0000 0010 0001 (  add r0, r0, #1 )
x3015 0000 1111 1111 1100 (  brnzp loop )
x3016 0001 0010 0100 0010 (remn add r1, r1, r2 )
x3017 0101 1001 0010 0001 (done and r4, r4, #1 )
```

```
x3018 0000 0100 0000 0010 (  brz checkremsign )
x3019 1001 0000 0011 1111 (  not r0, r0 )
x301a 0001 0000 0010 0001 ( add r0, r0, #1 )
x301b 0101 1011 0110 0001 ( checkremsign and r5, r5, #1 )
x301c 0000 0100 0000 0010 (  brz storeres )
x301d 1001 0010 0111 1111 ( not r1, r1 )
x301e 0001 0010 0110 0001 ( add r1, r1, #1 )
x301f 1011 0000 0000 0100 (storeres sti r0, pquo )
x3020 1011 0010 0000 0100 (   sti r1, prem )
x3021 1111 0000 0010 0101 (  halt )
x3022 0100 0000 0000 0000 ( pnuma   .fill x4000 )
x3023 0100 0000 0000 0001 ( pnumb   .fill x4001 )
x3024 0101 0000 0000 0000 ( pquo    .fill x5000 )
x3025 0101 0000 0000 0001 ( prem    .fill x5001 )
```