Appendix F

Selected Solutions

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.

6.3 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 )
```

6.5 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.

6.7 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.

6.9
```
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 )
```

6.11 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 )
```

6.13 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 )
```

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

6.17 `JSR #15`

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