APPENDIX D

FLOWCHARTS AND PSEUDOCODE

OVERVIEW

This appendix provides an introduction to writing flowcharts and pseudocode.

Flowcharts

If you have taken any previous programming courses, you are probably familiar with flowcharting. Flowcharts use graphic symbols to represent different types of program operations. These symbols are connected together into a flowchart to show the flow of execution of a program. Figure D-1 shows some of the more commonly used symbols. Flowchart templates are available to help you draw the symbols quickly and neatly.

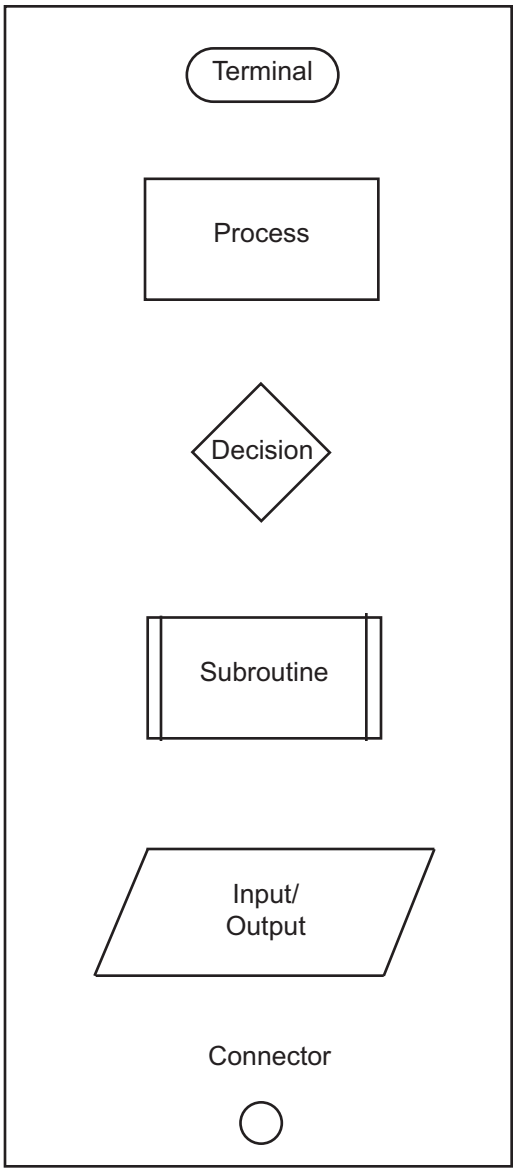

Figure D-1. Commonly Used Flowchart Symbols

Pseudocode

Flowcharting has been standard practice in industry for decades. However, some find limitations in using flowcharts, such as the fact that you can't write much in the little boxes, and it is hard to get the "big picture" of what the program does without getting bogged down in the details. An alternative to using flowcharts is pseudocode, which involves writing brief descriptions of the flow of the code. Figures D-2 through D-6 show flowcharts and pseudocode for commonly used control structures.

Structured programming uses three basic types of program control structures: sequence, control, and itera-

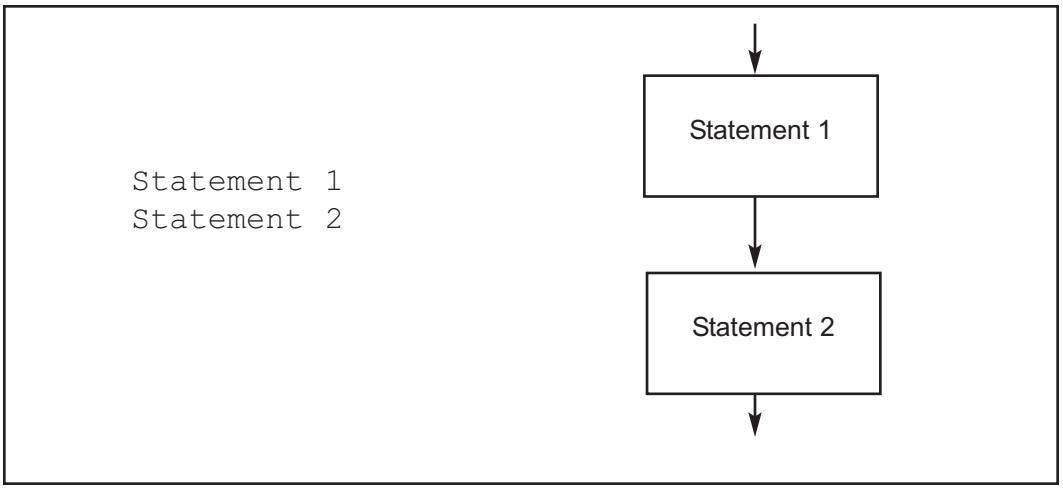

Figure D-2. SEQUENCE Pseudocode versus Flowchart

tion. Sequence is simply executing instructions one after another. Figure D-2 shows how sequence can be represented in pseudocode and flowcharts.

Figures D-3 and D-4 show two control programming structures: IF-THEN-ELSE and IF-THEN in both pseudocode and flowcharts.

Note in Figures D-2 through D-6 that "statement" can indicate one statement or a group of statements.

Figures D-5 and D-6 show two iteration control structures: REPEAT UNTIL and WHILE DO. Both structures execute a statement or group of statements repeatedly. The difference between them is that the REPEAT UNTIL structure always executes the statement(s) at least once, and checks the condition after each iteration, whereas the WHILE DO may not execute the statement(s) at all because the condition is checked at the beginning of each iteration.

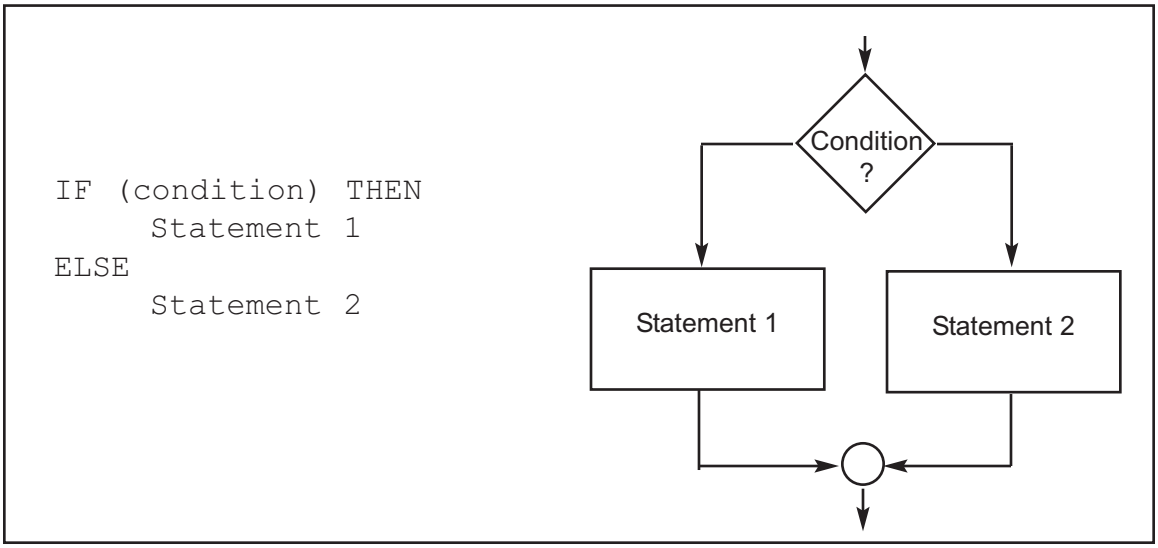

Figure D-3. IF THEN ELSE Pseudocode versus Flowchart

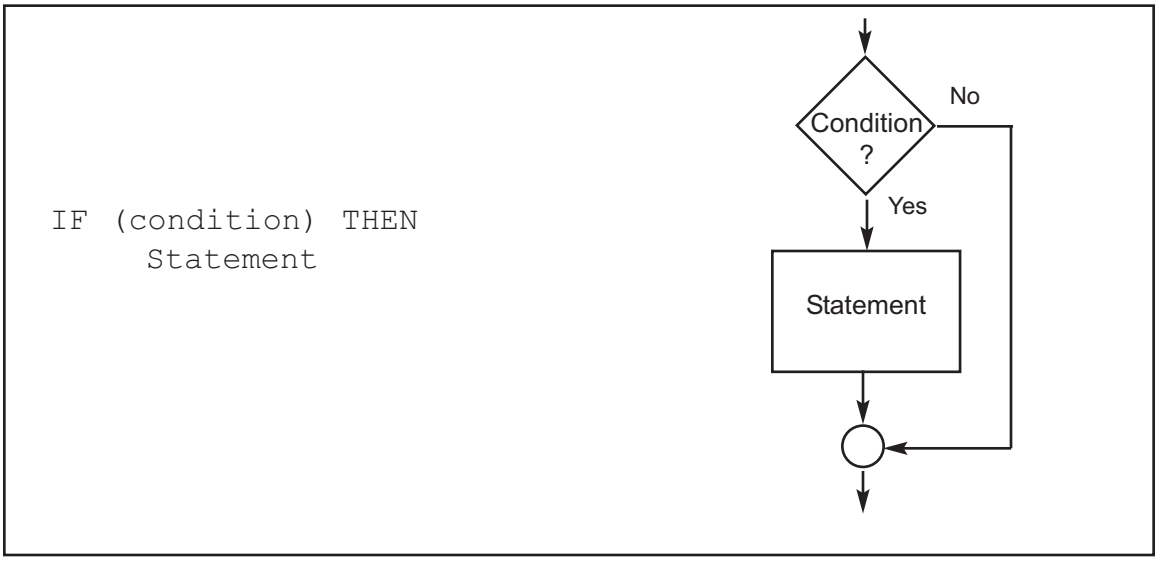

Figure D-4. IF THEN Pseudocode versus Flowchart

```
REPEAT
     Statement
UNTIL (condition)
```

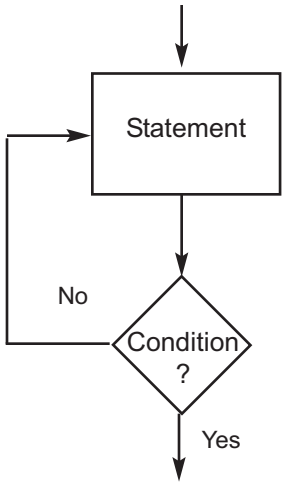

Figure D-5. REPEAT UNTIL Pseudocode versus Flowchart

```
WHILE (condition) DO
     Statement
```

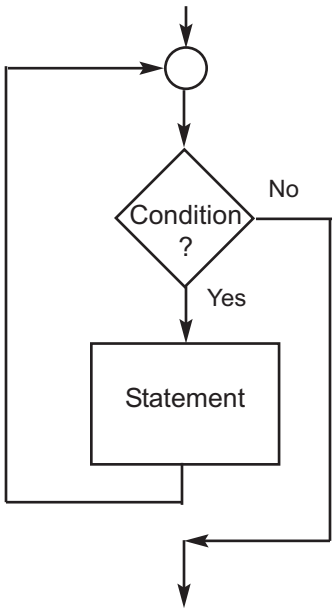

Figure D-6. WHILE DO Pseudocode versus Flowchart

Program D-1 finds the sum of a series of bytes. Compare the flowchart versus the pseudocode for Program D-1 (shown in Figure D-7). In this example, more program details are given than one usually finds. For example, this shows steps for initializing and decrementing counters. Another programmer may not include these steps in the flowchart or pseudocode. It is important to remember that the purpose of flowcharts or pseudocode is to show the flow of the program and what the program does, not the specific Assembly language instructions that accomplish the program's objectives. Notice also that the pseudocode gives the same information in a much more compact form than does the flowchart. It is important to note that sometimes pseudocode is written in layers, so that the outer level or layer shows the flow of the program and subsequent levels show more details of how the program accomplishes its assigned tasks.

```
Count = 5
Address = $140
Repeat
     Add next byte
     Increment address
     Decrement counter
Until Count = 0

Store Sum
```

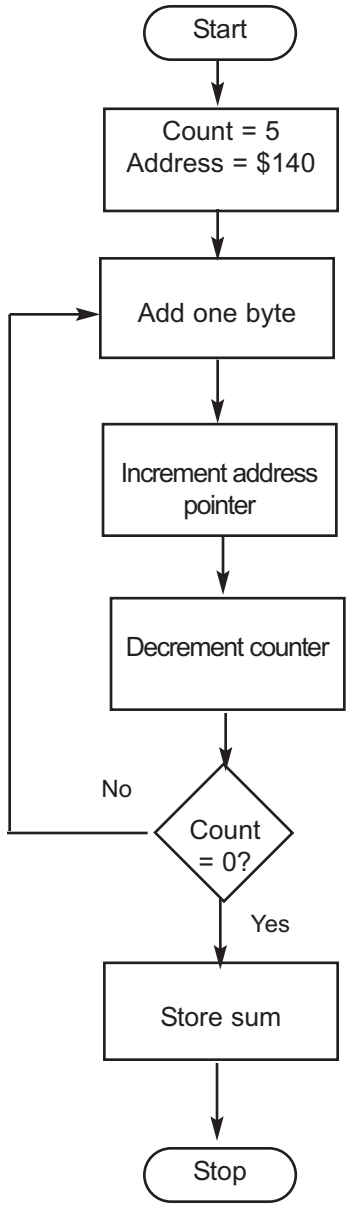

Figure D-7. Pseudocode versus Flowchart for Program D-1

```
#define     COUNTVAL    5      ;COUNT = 5
#define     COUNTER     R22
#define     SUM         R23
      LDI   COUNTER,COUNTVAL   ;R22 = 5
      CLR   SUM                ;SUM = 0
      LDI   R26,LOW($140)      ;load pointer to RAM data address
      LDI   R27,HIGH($140)
L1:   LD    R24,x+             ;copy RAM to R24 and increment pointer
      ADD   SUM,R24            ;add R24 to SUM
      DEC   COUNTER            ;decrement counter
      BRNE  L1                 ;loop until counter = zero
HERE: RJMP  HERE               ;stay here forever
```

Program D-1