Finite State Machines

An FSM is a computational model that can be used to simulate sequential logic, or, in other words, to represent and control execution flow.

It is a mathematical model of computation that can be used to model the behavior of a system that can be in a finite number of states. The system can change state based on the input it receives.

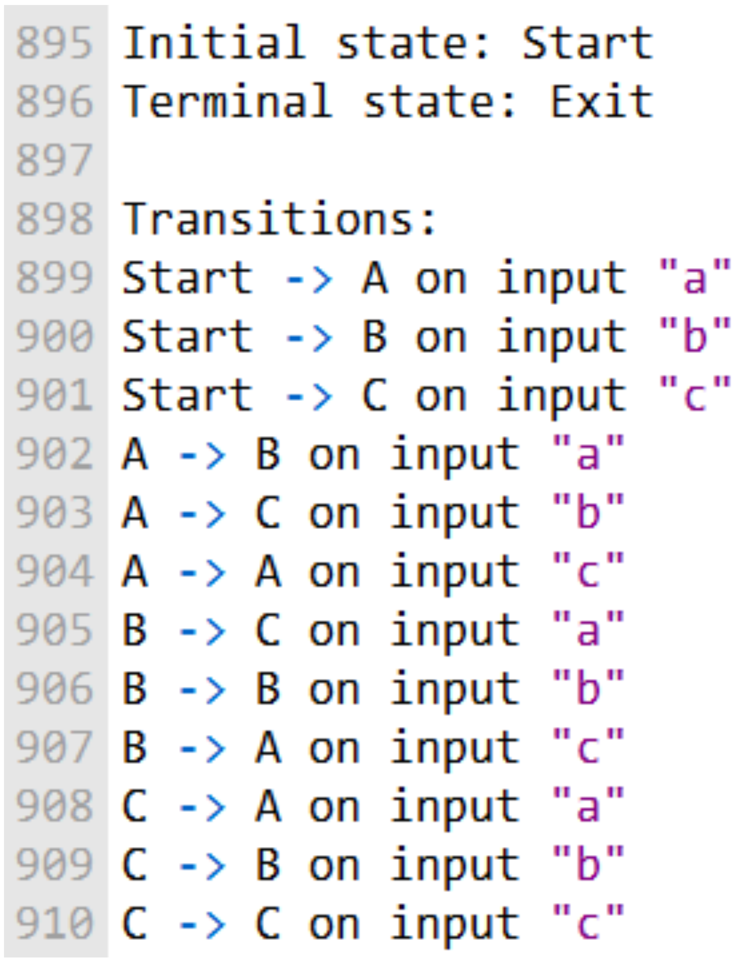
FSMs can be represented using a graph, where each node represents a state and each edge represents a transition from one state to another.

The edges are labeled with the input symbols that trigger the transitions. One node is designated as the initial state, and one or more nodes are designated as terminal states.

FSMs are used in a wide variety of applications, including:

- Traffic lights
- Vending machines
- Telephone systems
- Computer software
- Robotics

Here is a simple example of an FSM:

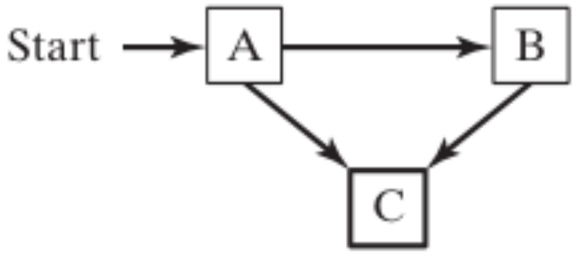


```
Initial state: Start
Terminal state: Exit

Transitions:
Start -> A on input "a"
Start -> B on input "b"
Start -> C on input "c"
A -> B on input "a"
A -> C on input "b"
A -> A on input "c"
B -> C on input "a"
B -> B on input "b"
B -> A on input "c"
C -> A on input "a"
C -> B on input "b"
C -> C on input "c"
```

- This FSM can be used to simulate the behavior of a traffic light.
- The FSM starts in the Start state.
- If the input is "a", the FSM transitions to the A state, which represents the green light.
- If the input is "b", the FSM transitions to the B state, which represents the yellow light.
- If the input is "c", the FSM transitions to the C state, which represents the red light.
- The FSM will continue to transition between states until it reaches the terminal state, the Exit state.
- This FSM will never reach the terminal state, because it is always possible to receive an input "a", "b", or "c".

FSMs can be used to model and control much more complex systems than a traffic light. For example, an FSM could be used to model and control the behavior of a vending machine, a telephone system, or a computer program.



Here is a more detailed explanation of the diagram:

The initial state is the Start state.

The three possible states are A, B, and C.

The arrows show the possible transitions between states. The terminal state is the Exit state. The FSM can be described in words as follows:

- The FSM starts in the Start state.
- If the FSM receives the input "a", it transitions to the A state.
- If the FSM receives the input "b", it transitions to the B state.
- If the FSM receives the input "c", it transitions to the C state.
- If the FSM is in the A state and receives the input "a", it transitions to the B state.
- If the FSM is in the A state and receives the input "b", it transitions to the C state.
- If the FSM is in the A state and receives the input "c", it remains in the A state.
- If the FSM is in the B state and receives the input "a", it transitions to the C state.
- If the FSM is in the B state and receives the input "b", it remains in the B state.
- If the FSM is in the B state and receives the input "c", it transitions to the A state.
- If the FSM is in the C state and receives the input "a", it transitions to the A state.
- If the FSM is in the C state and receives the input "b", it transitions to the B state.
- If the FSM is in the C state and receives the input "c", it remains in the C state.

The FSM will continue to transition between states until it reaches the terminal state, the Exit state.

===================================

Validating an Input String Programs

===================================

Here is a more detailed explanation of the example FSM in Figure 6-4:

Start state: **A**
Terminal state: **C**
Transitions: **A -> B on input "x"**

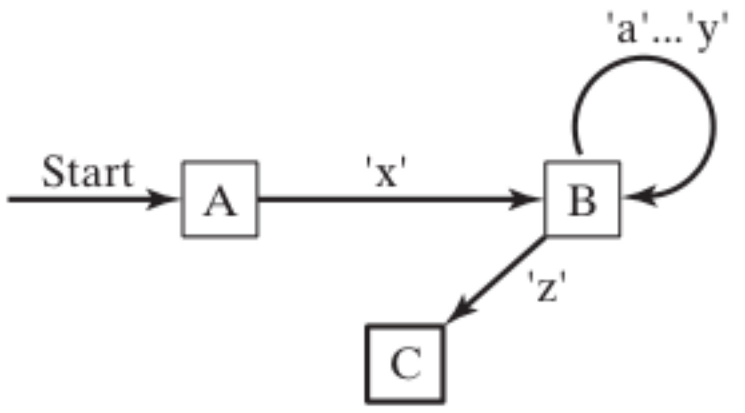
• B -> B on input any letter in the range {a, b, ..., y} B -> C on input "z" This FSM can be used to validate an input string according to the following two rules:
• The string must begin with the letter "x" and end with the letter "z". Between the first and last characters, there can be zero or more letters within the range {a, b, ..., y}.
• To validate an input string, the FSM starts in the A state.
• It then reads the input string character by character.
• If the next character is "x", the FSM transitions to the B state.
• If the next character is any letter in the range {a, b, ..., y}, the FSM remains in the B state.
• If the next character is "z", the FSM transitions to the C state.
• If the FSM reaches the C state, it means that the input string is valid.
• If the FSM reaches the end of the input string before it reaches the C state, it means that the input string is invalid.

Here is an example of a valid input string:

• xyz The FSM will start in the A state. It will then read the first character, which is "x". This will transition the FSM to the B state.
• The FSM will then read the second character, which is "y". This will cause the FSM to remain in the B state. The FSM will then read the third character, which is "z".
• This will transition the FSM to the C state. Since the FSM has reached the C state, it means that the input string is valid.

Here is another example of an invalid input string:

• xab The FSM will start in the A state. It will then read the first character, which is "x". This will transition the FSM to the B state.
• The FSM will then read the second character, which is "a". This will cause the FSM to remain in the B state.
• The FSM will then read the third character, which is "b". This will cause the FSM to remain in the B state.
• However, the FSM will not be able to transition to the C state, because the next input character is not "z". This means that the input string is invalid.



I have reviewed the notes again, and I understand them better now. The notes are describing how to use a FSM to validate an input string according to the following two rules:

The string must begin with the letter "x" and end with the letter "z". Between the first and last characters, there can be zero or more letters within the range {a, b, ..., y}.

The FSM diagram in the image shows the possible states and transitions of the FSM. The FSM starts in the A state.

If the next character in the input string is "x", the FSM transitions to the B state. If the next character in the input string is any letter in the range {a, b, ..., y}, the FSM remains in the B state.

If the next character in the input string is "z", the FSM transitions to the C state.

If the FSM reaches the C state, it means that the input string is valid. If the FSM reaches the end of the input string before it reaches the C state, it means that the input string is invalid.
Here is an example of a valid input string:

xyz The FSM will start in the A state. It will then read the first character, which is "x". This will transition the FSM to the B state.

The FSM will then read the second character, which is "y". This will cause the FSM to remain in the B state. The FSM will then read the third character, which is "z".

This will transition the FSM to the C state. Since the FSM has reached the C state, it means that the input string is valid.

Here is an example of an invalid input string:

xab The FSM will start in the A state. It will then read the first character, which is "x". This will transition the FSM to the B state.

The FSM will then read the second character, which is "a". This will cause the FSM to remain in the B state. The FSM will then read the third character, which is "b".

This will cause the FSM to remain in the B state. However, the FSM will not be able to transition to the C state, because the next input character is not "z". This means that the input string is invalid.

- If the end of the input stream is reached while the program is in state A or B, an error condition results because only state C is marked as a terminal state. This means that the input string must end with the letter "z" in order for it to be valid.
- The following input strings would be recognized by this FSM:

xaabcdefgz xz xyyqqrrstuvz

- All of these input strings begin with the letter "x" and end with the letter "z". There may be any number of letters in the range {a, b, ..., y} in between.
- Here is an example of an input string that would not be recognized by this FSM:

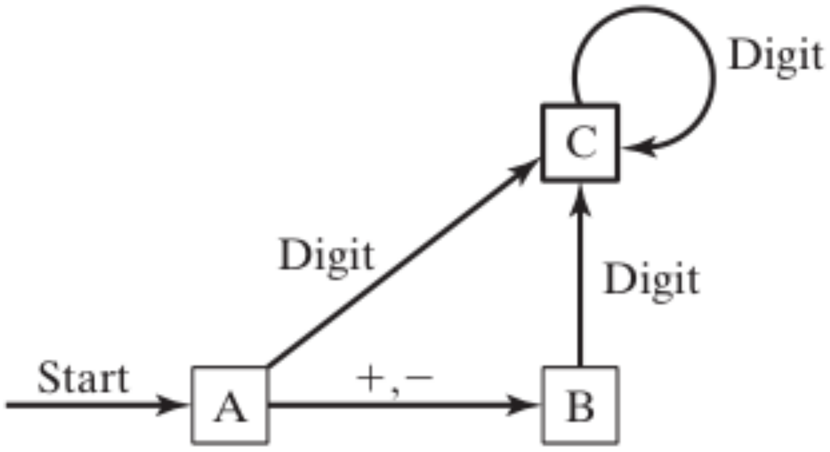
xab

• This input string begins with the letter "x" but does not end with the letter "z". Therefore, it is invalid.

=======================================

Validating Integers in Programs

=======================================



The FSM diagram in Figure 6-5 shows how to validate a signed integer. The FSM starts in the Start state.

If the next character in the input stream is a plus sign (+) or a minus sign (-), the FSM transitions to the Sign state.

If the next character in the input stream is a digit, the FSM transitions to the Digits state.

If the FSM is in the Sign state and the next character in the input stream is a digit, the FSM remains in the Sign state.

This is because the sign can be followed by any number of digits.

If the FSM is in the Digits state and the next character in the input stream is a digit, the FSM remains in the Digits state.

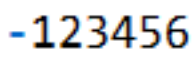
This is because the sequence of digits can be any length.

If the FSM is in the Digits state and the next character in the input stream is not a digit, the FSM transitions to the End state.

This is because the sequence of digits must end with a non-digit character.

The End state is a terminal state. This means that the input string is valid if the FSM reaches the End state.

Here is an example of a valid input string:


-123456

-123456 The FSM will start in the Start state. It will then read the first character, which is a minus sign (-).

This will transition the FSM to the Sign state. The FSM will then read the second character, which is the digit 1.

This will cause the FSM to remain in the Sign state. The FSM will then read the third character, which is the digit 2.

This will cause the FSM to remain in the Sign state. The FSM will then read the fourth character, which is the digit 3.

This will cause the FSM to remain in the Sign state. The FSM will then read the fifth character, which is the digit 4.

This will cause the FSM to remain in the Sign state. The FSM will then read the sixth character, which is the digit 5.

This will cause the FSM to remain in the Sign state. The FSM will then read the seventh character, which is the digit 6.

This will cause the FSM to transition to the Digits state. The FSM will then read the eighth character, which is not a digit.

This will cause the FSM to transition to the End state. Since the FSM has reached the End state, it means that the input string is valid.

Here is another example of an invalid input string:

```
-123456.78          ;invalid
```

-123456.78 The FSM will start in the Start state. It will then read the first character, which is a minus sign (-).

This will transition the FSM to the Sign state.

The FSM will then read the second character, which is the digit 1. This will cause the FSM to remain in the Sign state.

The FSM will then read the third character, which is the digit 2. This will cause the FSM to remain in the Sign state.

The FSM will then read the fourth character, which is the digit 3. This will cause the FSM to remain in the Sign state.

The FSM will then read the fifth character, which is the digit 4. This will cause the FSM to remain in the Sign state.

The FSM will then read the sixth character, which is the digit 5. This will cause the FSM to remain in the Sign state.

The FSM will then read the seventh character, which is the digit 6. This will cause the FSM to transition to the Digits state.

The FSM will then read the eighth character, which is a period (.). This is not a digit, so the FSM will transition to the End state.

However, the End state is not a terminal state. This means that the input string is invalid.

FSMs are a powerful tool for validating input strings and other types of data. They are used in a wide variety of applications, including programming language compilers, text editors, and network protocols.

=======================================

Validating Integers in Programs

=======================================

FSM for parsing a signed integer:

• This FSM diagram shows how to validate a signed integer. The FSM starts in the Start state. If the next character in the input stream is a plus sign (+) or a minus sign (-), the FSM transitions to the Sign state. If the next character in the input stream is a digit, the FSM transitions to the Digits state.

• If the FSM is in the Sign state and the next character in the input stream is a digit, the FSM remains in the Sign state. This is because the sign can be followed by any number of digits.

• If the FSM is in the Digits state and the next character in the input stream is a digit, the FSM remains in the Digits state. This is because the sequence of digits can be any length.

• If the FSM is in the Digits state and the next character in the input stream is not a digit, the FSM transitions to the End state. This is because the sequence of digits must end with a non-digit character.

• The End state is a terminal state. This means that the input string is valid if the FSM reaches the End state.

The following assembly language code implements the FSM diagram above:

```
StateA:
call Getnext ; read next char into AL
cmp al, '+' ; leading + sign?
je StateB ; go to State B
cmp al, '-' ; leading - sign?
je StateB ; go to State B
call IsDigit ; ZF = 1 if AL contains a digit
jz StateC ; go to State C
call DisplayErrorMsg ; invalid input found
jmp Quit ; exit program

StateB:
; ...

StateC:
; ...

End:
call Crlf
exit

main ENDP
```

This code works as follows:

- The label StateA marks the start of the FSM.
- The call to the Getnext procedure reads the next character from the console input into the AL register.
- The code checks for a leading plus (+) or minus (-) sign.
- If a leading sign is found, the code jumps to the label StateB.
- If a leading sign is not found, the code calls the IsDigit procedure to check if the character in AL is a digit. If the character is a digit, the code jumps to the label StateC.
- If the character in AL is not a digit or a leading sign, the code calls the DisplayErrorMsg procedure to display an error message on the console and then jumps to the label Quit.
- The label StateB marks the state where the FSM is after reading a leading sign.

- The code in StateB will check for the other possible transitions away from this state and take the appropriate action. The label StateC marks the state where the FSM is after reading a digit.
- The code in StateC will check for the other possible transitions away from this state and take the appropriate action. The label End marks the terminal state of the FSM.
- The code in End will perform any necessary cleanup and then exit the program. The main procedure simply calls the StateA procedure to start the FSM.
- This is just a basic example of how to implement a FSM in assembly language. More complex FSMs can be implemented using the same basic principles.

```
; Finite State Machine (Finite.asm)
INCLUDE Irvine32.inc

ENTER_KEY = 13

.data
InvalidInputMsg: db "Invalid input", 13, 10, 0

.code
main:
    ; Clear screen
    call Clrscr

    ; Start state
StateA:
        ; Read next character into AL
        call Getnext

        ; Check for leading + or - sign
        cmp al, '+'
        je StateB
        cmp al, '-'
        je StateB

        ; Check if AL contains a digit
        call IsDigit
        jz StateC

        ; Invalid input
        call DisplayErrorMsg
        jmp Quit

StateB:
        ; Read next character into AL
        call Getnext

        ; Check if AL contains a digit
        call IsDigit
        jz StateC

        ; Invalid input
        call DisplayErrorMsg
        jmp Quit

StateC:
        ; Read next character into AL
        call Getnext

        ; Check if AL contains a digit
        call IsDigit
        jz StateC

        ; Check if Enter key pressed
        cmp al, ENTER_KEY
        je Quit

        ; Invalid input
        call DisplayErrorMsg
        jmp Quit

Quit:
        ; Call Crlf to print a newline
        call Crlf
        exit

; Getnext procedure
; Reads a character from standard input
; Receives: nothing
; Returns: AL contains the character
Getnext:
        ; Input from keyboard
        call ReadChar

        ; Echo on screen
        call WriteChar

        ret

; DisplayErrorMsg procedure
; Displays an error message indicating that
; the input stream contains illegal input
; Receives: nothing
; Returns: nothing
DisplayErrorMsg:
        ; Push EDX onto the stack
        push edx
        ; Move the offset of the error message to EDX
        mov edx, OFFSET InvalidInputMsg

        ; Call WriteString to print the error message
        call WriteString

        ; Pop EDX from the stack
        pop edx

        ret
```

main procedure:

The main procedure is the entry point for the program. It starts by clearing the screen and then entering the StateA state.

StateA state:

The StateA state is the start state for the FSM. It reads the next character from the input stream and checks for a leading + or - sign. If a leading sign is found, the FSM transitions to the StateB state. If a leading sign is not found, the FSM checks if the character is a digit. If the character is a digit, the FSM transitions to the StateC state. Otherwise, the FSM calls the DisplayErrorMsg procedure to display an error message and then jumps to the Quit label to exit the program.

StateB state:

The StateB state is the state where the FSM is after reading a leading + or - sign. It reads the next character from the input stream and checks if it is a digit. If the character is a digit, the FSM transitions to the StateC state. Otherwise, the FSM calls the DisplayErrorMsg procedure to display an error message and then jumps to the Quit label to exit the program.

StateC state:

The StateC state is the state where the FSM is after reading a digit. It reads the next character from the input stream and checks if it is a digit. If the character is a digit, the FSM remains in the StateC state. Otherwise, the FSM checks if the Enter key was pressed. If the Enter key was pressed, the FSM transitions to the Quit label to exit the program. Otherwise, the FSM calls the DisplayErrorMsg procedure to display an error message and then jumps to the Quit label to exit the program.

Quit label:

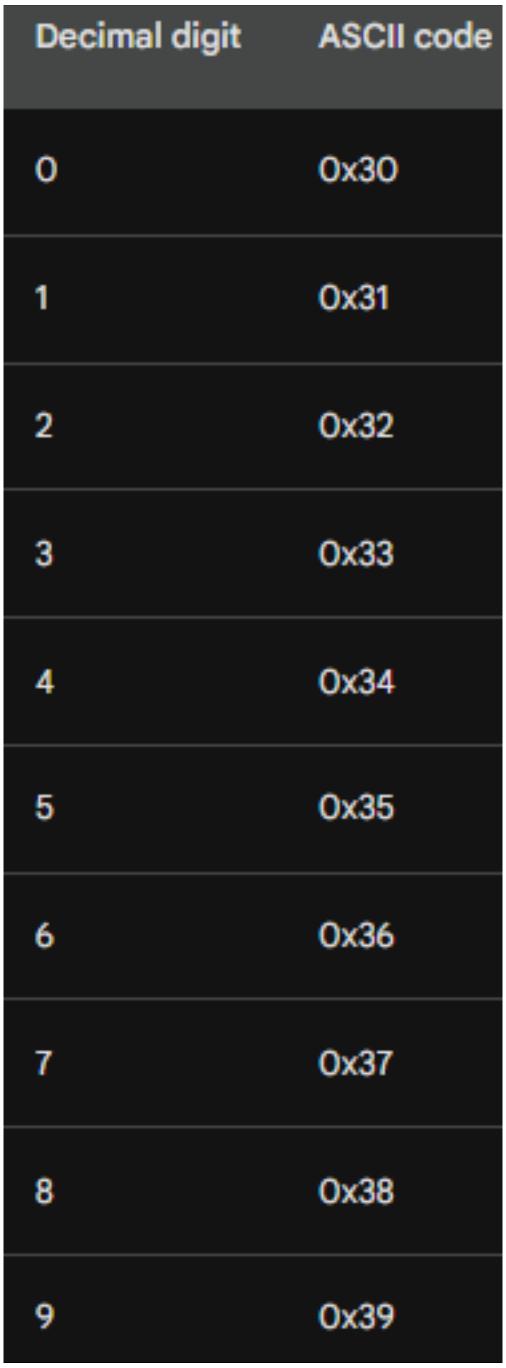
The Quit label is the exit point for the program. The main procedure jumps to the Quit label to exit the program when the Enter key is pressed or when an invalid character is encountered.

Getnext procedure:
The Getnext procedure reads a character from standard input and echoes it to the screen. It returns the character in the AL register.

DisplayErrorMsg procedure:
The DisplayErrorMsg procedure displays an error message indicating that the input stream contains illegal input. It receives nothing and returns nothing. This is a basic example of how to implement a FSM in assembly language.

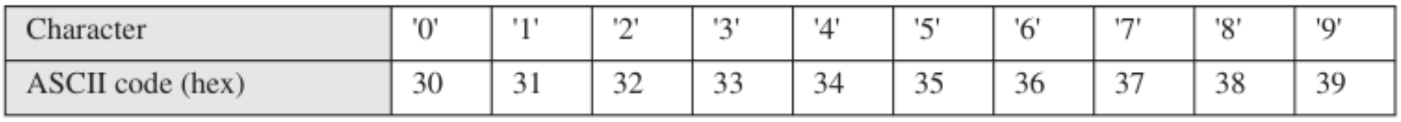
• The IsDigit procedure determines whether the character in the AL register is a valid decimal digit. It returns the setting of the Zero flag, which is 1 if the character is a valid decimal digit and 0 otherwise.
• The IsDigit procedure works by first comparing the character in AL to the ASCII code for the digit 0. If the character is less than 0, then it is not a valid decimal digit and the Zero flag is cleared.
• Next, the IsDigit procedure compares the character in AL to the ASCII code for the digit 9. If the character is greater than 9, then it is not a valid decimal digit and the Zero flag is cleared.
• Finally, the IsDigit procedure sets the Zero flag.

The following table shows the hexadecimal ASCII codes for decimal digits:



Decimal digit	ASCII code
0	0x30
1	0x31
2	0x32
3	0x33
4	0x34
5	0x35
6	0x36
7	0x37
8	0x38
9	0x39

As you can see, the ASCII codes for decimal digits are **contiguous.** This means that we only need to check for the starting and ending range values.



Character	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
ASCII code (hex)	30	31	32	33	34	35	36	37	38	39

The IsDigit procedure uses the CMP instruction to compare the character in the AL register to the ASCII codes for the digits 0 and 9. If the character is less than 0 or greater than 9, then the JB or JA instruction will jump to the label ID1, respectively.

The JB instruction jumps to a label when the Carry flag (CF) is set and the Zero flag (ZF) is clear. The JA instruction jumps to a label when the Carry flag (CF) is clear and the Zero flag (ZF) is clear.

If neither the JB nor the JA instruction jumps to the label ID1, then the TEST instruction is executed. The TEST instruction sets the Zero flag if the result of the AND operation is zero.

Since the JB and JA instructions jump to the label ID1 if the Zero flag is clear, the TEST instruction will only be executed if the character in the AL register is a digit.

Therefore, the IsDigit procedure returns the setting of the Zero flag, which is 1 if the character in the AL register is a digit and 0 otherwise.

Here is a more detailed explanation of the code:

```
; IsDigit procedure
; Determines whether the character in AL is a valid decimal digit.
; Receives: AL = character
; Returns: ZF = 1 if AL contains a valid decimal digit; otherwise, ZF = 0.
;-----------------------------------------------------------------
IsDigit PROC
; Compare AL to the ASCII code for the digit 0.
; If AL is less than 0, the JB instruction will jump to the label ID1.
cmp al,'0'
jb ID1
; ZF = 0 when jump taken
; Compare AL to the ASCII code for the digit 9.
; If AL is greater than 9, the JA instruction will jump to the label ID1.
cmp al,'9'
ja ID1
; ZF = 0 when jump taken
; If neither the JB nor the JA instruction jumps to the label ID1,
; then the character in AL must be a digit. Therefore, we set the Zero flag.
test ax,0
; set ZF = 1
; Return from the procedure.
ID1: ret
IsDigit ENDP
```

This is a very efficient way to implement the IsDigit procedure, because it takes advantage of the hardware characteristics of the CPU.