Microprocessor Programming

Dr. Tin Ni Ni Kyaw

Ph. D (Kumamoto University, Japan)

Associate Professor

Department of Computer Engineering and Information Technology

Yangon Technological University

Yangon, Myanmar

Microprocessor Programming

Lecture 12

Strings

Contents

- Introduction
- Strings
- String Instructions
- Array of Strings
- Complete Program
- Summary
- Practical Works
- Assignments

Introduction

- A string is an array of bytes as opposed to an array of signed double words.
- The strings are declared as byte instead of sdword.
- It is incremented only by one to account for the size of byte as opposed to the size of a sdword.
- So, it is possible to use all of the techniques for arrays introduced in the previous lecture with strings.

Strings

- The strings can be declared as follows.

```
string1 byte "Hello World!"
string2 byte 12 dup(?)
letter1 byte 'a'
letter2 byte ?
name1 byte "MaryJo"
name2 byte 6 dup(?)
name3 byte "Abe Lincoln"
name4 byte 6 dup(" ")
```

Strings (Cont.)

- To copy the contents of one string to another string, the ebx register could be used as an index for both strings.
- Note that the mov instructions are using only a 1-byte register, AL, instead of a 32-bit register eax.

Strings (Cont.)

- Here is a MASM code segment for copying the contents in string1 into string2 by using mov instruction with ebx register.

```
        .data
string1 byte "Hello World!"
string2 byte 12 dup(?)
        .code
        mov ecx,12              ;load ecx with 12
        mov ebx,0               ;load ebx with 0
        .repeat
        mov al,string1[ebx]     ;load al with string1[ebx]
        mov string2[ebx],al     ;store al in string2[ebx]
        inc ebx                 ;increment ebx by 1
        .untilcxz
```

Strings (Cont.)

- Besides using the ebx registers, the arrays can also be indexed by using the esi and edi registers.
- The esi can be used for string1 as the source of the transfer and the edi can be used for the destination to string2.

Strings (Cont.)

- This is a MASM code segment for copying the contents in string1 into string2 by using mov instruction with esi and edi registers.

```
        .data
string1 byte "Hello World!"
string2 byte 12 dup(?)
        .code
        mov ecx,12          ; load ecx with 12
        lea esi,string1     ; load esi with address of string1
        lea edi,string2     ; load edi with address of string2
        .repeat
        mov al,[esi]        ; load al with [esi]
        mov [edi],al        ; store al in [edi]
        inc esi             ; increment esi by 1
        inc edi             ; increment edi by 1
        .untilcxz
```

String Instructions

- There are a number of functions that need to be performed on strings.
- The following instructions can primarily be used for string processing.

Instruction	Meaning
movsb	Move string byte
cmpsb	Compare string byte
scasb	Scan string byte
storsb	Store string byte
lodsb	Load string byte

String Instructions

Moving Strings (movsb)

- The movsb instruction is used to move a string of bytes.
- In particular, the movsb instruction does two things.
- First, it moves the contents of the byte pointed at by the esi register to the byte in memory pointed at by the edi register.
- Then, it decrements the ecx register and either increments or decrements the esi and edi registers by 1.

String Instructions (Cont.)

- For example, if we want to move only a single byte, then we can write the following code segment:

```
        .data
letter1 byte 'a'
letter2 byte ?
        .code
        lea esi,letter1   ; load esi with address of letter1
        lea edi,letter2   ; load edi with address of letter2
        movsb             ; move string byte from [esi] to [edi]
```

String Instructions (Cont.)

- The way to determine which way esi and edi will be altered is based on the direction flag.
- The direction flag can be cleared or set by using the cld or std instructions which stand for clear direction flag or set direction flag, respectively.
- If the esi and edi registers need to be incremented, the direction flag then needs to be cleared (cld), otherwise the direction flag needs to set (std) to cause the registers to be decremented.

String Instructions (Cont.)

- The simple code using mov instruction is on the left and the modified code using movsb instruction is listed on the right.

```
mov ecx,12
mov esi,offset string1
mov edi,offset string2
.repeat
mov al,[esi]
mov [edi],al
inc esi
inc edi
.untilcxz
```

```
mov ecx,12
mov esi,offset string1
mov edi,offset string2
cld
.repeat
movsb
.untilcxz
```

String Instructions (Cont.)

- The above code can be further simplified by using a prefix.
- There are three prefixes that are useful as listed below:

Prefix	Meaning
`rep`	Repeat
`repe`	Repeat while equal
`repne`	Repeat while not equal

String Instructions (Cont.)

- The rep prefix works just like the .repeat-.untilcxz directives, where it decrements the ecx register until it reaches 0.
- Unlike the .repeat-.untilcxz directives which can have any number of instructions in the body of the loop, the rep prefix works only in conjunction with instructions like movsb.

String Instructions (Cont.)

- Thus, the modified code shown below on the left can be further simplified as follows on the right.

```
mov ecx,12
lea esi,string1
lea edi,string2
cld
.repeat
movsb
.untilcxz
```

```
mov ecx,12
lea esi,string1
lea edi,string2
cld
rep movsb
```

String Instructions (Cont.)

Scanning (scasb)

- The scasb instruction is a fairly useful scanning instruction when used with either the repe or repne prefixes.
- For example, consider for the case that we want to scan a string of bytes to find whether there was a particular character in a string, such as a blank.

String Instructions (Cont.)

Scanning (scasb)

- Using the repne prefix, the string will be scanned until the character is found or until the ecx is equal to 0:

```
        .data
name1   byte "Abe Lincoln"
        .code
        mov al, ' '                 ; load al with a space
        mov ecx,lengthof name1      ; load ecx with length
        lea edi,name1               ; load address of name1
        repne scasb
```

String Instructions (Cont.)

Storing (stosb)

- The stosb instruction is useful to store the contents of the AL register at the location in a string pointed at by the edi register.
- In this case, the edi register is incremented after its respective operations.

String instruction	Equivalent
`stosb`	`mov al,[edi]` `inc edi`

String Instructions (Cont.)

Loading (lodsb)

- The lodsb instruction loads the AL register with the contents of the string pointed at by the esi register.
- In this case, the esi register is incremented after its respective operations.

String instruction	Equivalent
`lodsb`	`mov [esi],al` `inc esi`

String Instructions (Cont.)

- As an example of how many of the above instructions might be used in which situations, we will consider the task of taking someone’s name in the normal first name followed by last name order and then reversing it so that the last name is first, followed by a comma, followed by a space, and then followed by the last name.

String Instructions (Cont.)

- This is the implementation of string processing to output the names in reverse order by using the string instructions.

```
        .data
name1   byte "Abe Lincoln"
name2   byte 12 dup (?)
        .code
        mov al,' '              ; load al with space
        mov ecx,lengthof name1 ; load length of name1
        lea edi,name1           ; load address of name1
        repne scasb             ; find space in name1
        push ecx                ; save ecx
        mov esi,edi             ; move edi to esi
        lea edi,name2           ; load adress of name2
        rep movsb               ; copy last name to name2
        mov al,', '             ; load al with comma
        stosb                   ; store comma in name2
        mov al,' '              ; load al with comma
        stosb                   ; store space in name2
        mov ecx,lengthof name2 ; store length of name2
        pop eax                 ; restore ecx into eax
        sub ecx,eax             ; sub length of last name
        dec ecx                 ; decrement ecx for space
        lea esi,name1           ; load address of name1
        rep movsb               ; copy first name to name2
```

String Instructions (Cont.)

- In above program, first the space needs to be found between the first and last name in name1 using scasb instruction.
- After ecx is saved for subsequent processing, the last name from name1 needs to be copied to name2 using the movsb instruction.
- Then, a comma and a space needs to be inserted into name2 using the stosb instruction.
- Then, using the previously saved value of ecx to determine the length of the first name, the first name can be copied from name1 to name2 using the movsb instruction.

String Instructions (Cont.)

Comparing Strings (cmpsb)

- The cmpsb instruction is used to compare a string of bytes.
- It performs two major tasks.
- First, it compares the two bytes pointed to by the esi and edi registers and sets the appropriate flags such as the zero and sign flags respectively.
- Then, it decrements the ecx register and increments or decrements the esi and edi registers as indicated by the direction flag just like with the movsb instruction.

String Instructions (Cont.)

Comparing Strings (cmpsb)

- For example, given the following declaration with equal length strings, cmpsb is used to compare the two names.

```
        .data
name1   byte "James"
name2   byte "James"
        .code
        mov ecx,lengthof name1   ; load ecx with length
        lea esi,name1            ; load address of name1
        lea edi,name2            ; load address of name2
        cld                      ; clear direction flag
        repe cmpsb               ; repeat while equal
```

String Instructions (Cont.)

- Modifying the previous code segment above to include input and output, the following complete program illustrates how this would work.

```
            .listall
            .386
            .model flat,c
            .stack 100h
scanf       PROTO arg2:Ptr Byte, inputlist:VARARG
printf      PROTO arg1:Ptr Byte, printlist:VARARG
            .data
msg1fmt     byte "%s",0
msg2fmt     byte 0Ah,"%s",0Ah,0Ah,0
in1fmt      byte "%s",0
msg1        byte "Enter a first name: ",0
msg2        byte "Enter another first name: ",0
msg3        byte "The names are not the same.",0
msg4        byte "The names are the same.",0
name1       byte 6 dup(" ")
name2       byte 6 dup(" ")
            .code
main        proc
            INVOKE printf, ADDR msg1fmt, ADDR msg1
            INVOKE scanf, ADDR in1fmt, ADDR name1
            INVOKE printf, ADDR msg1fmt, ADDR msg2
            INVOKE scanf, ADDR in1fmt, ADDR name2
            mov ecx,lengthof name1   ; load ecx with length
            lea esi,name1            ; load address of name1
            lea edi,name2            ; load address of name2
            cld                      ; clear direction flag
            repe cmpsb               ; compare while equal
            .if ecx > 0              ; check if ecx > 0
            INVOKE printf, ADDR msg2fmt, ADDR msg3
            .else
            dec esi                ; back up esi one position
            dec edi                ; back up edi one position
            mov al,[esi]           ; load al with [esi]
            .if al != [edi]        ; if not equal
            INVOKE printf, ADDR msg2fmt, ADDR msg3
            .else
            INVOKE printf, ADDR msg2fmt, ADDR msg4
            .endif
            .endif
            ret
main        endp
            end
```

Array of Strings

- Given the following array of strings, the strings can be viewed as single string or as an array of strings.

```
names1   byte "Abby","Fred","John","Kent","Mary"
```

names1

A	b	b	y	F	r	e	d	J	o	h	n	K	e	n	t	M	a	r	y

Strings viewed as a single string

names1

A	b	b	y
F	r	e	d
J	o	h	n
K	e	n	t
M	a	r	y

Strings viewed as an array of strings

Array of Strings (Cont.)

- This code segment is for moving string from names1 to names2 of the given array of strings.

```
            .data
names1      byte "Abby","Fred","John","Kent","Mary"
names2      byte 20 dup(?)

            .code
main        proc
            mov ecx,5           ; load ecx with 5
            lea esi,names1      ; load esi with address of names1
            lea edi,names2      ; load edi with address of names2
            cld                 ; clear direction flag
            .repeat
            push ecx            ; save ecx
            mov ecx,4           ; load ecx with 4
            rep movsb           ; move string from names1 to names2
            pop ecx             ; restore ecx
            .untilcxz
```

Complete Program: Searching an Array of Strings

- In order to accomplish individual processing of each string in the array, consider the problem of searching an array of fixed length strings sequentially.
- Note that since ecx is used for the loop control variable of the outer loop, its value must be pushed prior to the repe cmpsb instruction and then popped prior to the end of the outer loop.

Complete Program: Searching an Array of Strings

- This is the complete implementation of searching an array of strings in MASM.

```
        .listall
        .386
        .model flat,c
        .stack 100h
scanf   PROTO arg2:Ptr Byte, inputlist:VARARG
printf  PROTO arg1:Ptr Byte, printlist:VARARG
        .data
msg1fmt byte 0Ah,"%s",0
msg2fmt byte 0Ah,"%s",0Ah,0Ah,0
in1fmt  byte "%s",0
msg1    byte "Enter the state to be found: ",0
msg2    byte "The state was found.",0
msg3    byte "The state was not found.",0
arrystr byte "Illinois  ","Michigan  ","Iowa      ",
             "Missouri  ","Arkansas  ","Tennessee ",
             "Louisiana ","Arizona   ","Montana   ",
             "Ohio      "
n       sdword 10
string  byte 10 dup(?)
found   sdword ?
         .code
main    proc
        INVOKE printf, ADDR msg1fmt,ADDR msg1
        INVOKE scanf, ADDR in1fmt, ADDR string
        mov ecx,0                  ; initialize ecx to 0
        mov found,0                ; initialize found to 0
        lea edi,arrystr+0          ; load edi with address
        .while(ecx<n &&  found != -1)
        push ecx                   ; save ecx
        lea esi,string+0           ; load address of string
        cld                        ; clear direction flag
        mov ecx,lengthof string ; load length of string
        repe cmpsb                 ; compare while equal
        dec esi                    ; decrement es1
        dec edi                    ; decrement edi
        mov al,[esi]               ; load al with [esi]
        mov ah,[edi]               ; load ah with [edi]
        .if (al==0)&&(ah==" ") ; compare for 0 and space
        mov found,-1               ; if yes, found
        .endif
        inc edi                    ; increment edi back
        add edi,ecx                ; adjust edi to next string
        pop ecx                    ; restore ecx
        inc ecx                    ; increment ecx
        .endw
        .if (found ==-1)
        INVOKE printf, ADDR msg2fmt, ADDR msg2
        .else
        INVOKE printf, ADDR msg2fmt, ADDR msg3
        .endif
        ret
main    endp
        end
```

Summary

- The movsb instruction moves a string of bytes.
- The cmpsb instruction compares a byte in a string.
- The scasb instruction will scan a string for the character.
- The stosb instruction will store the character at the location pointed by the edi register.
- The lodsb instruction will load the character pointed by the esi register into the al register.

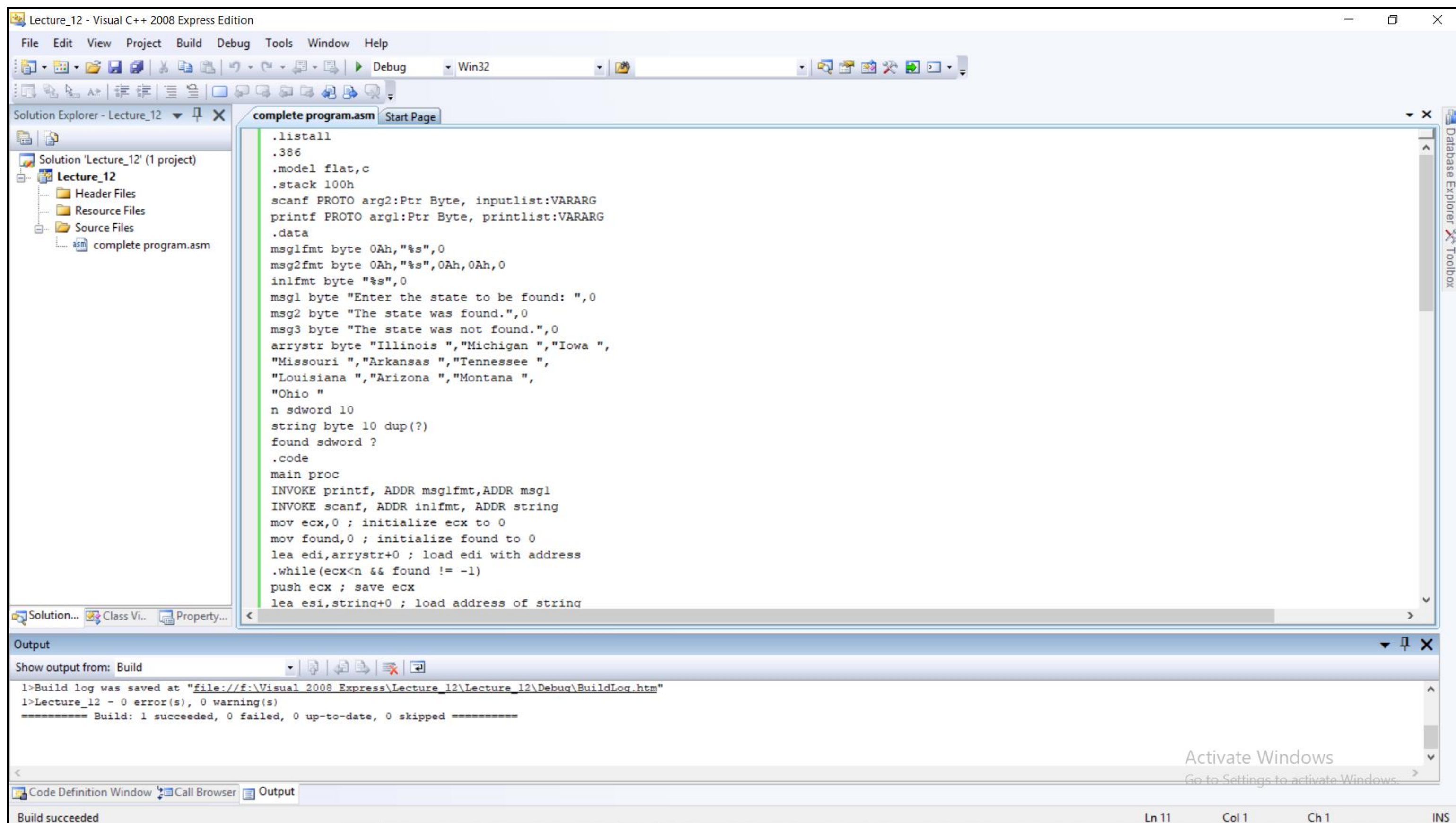
Microprocessor Programming

Practical Works

Searching an Array of Strings

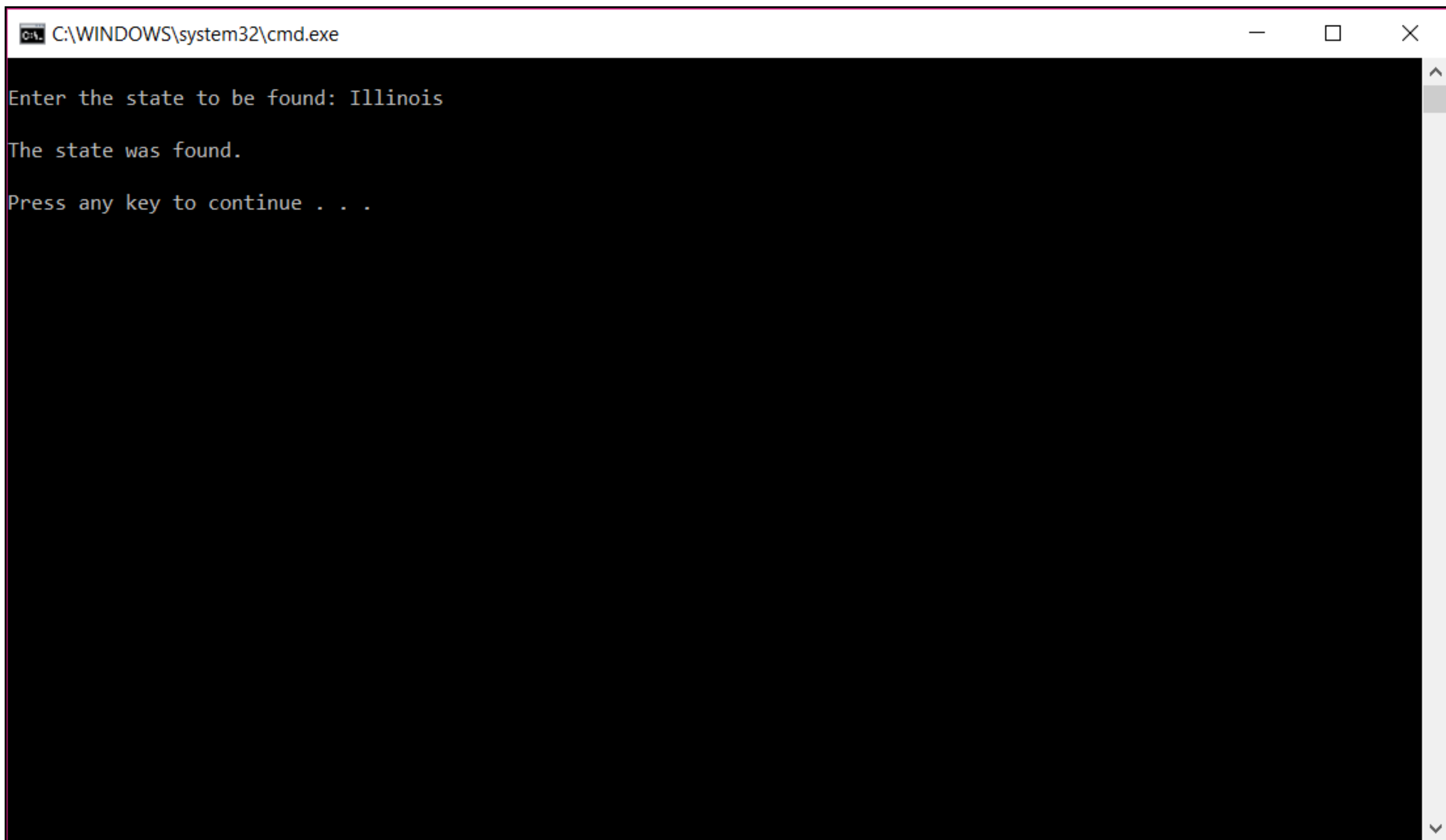
Searching an Array of Strings

- The implementation of the given MASM program is described below:



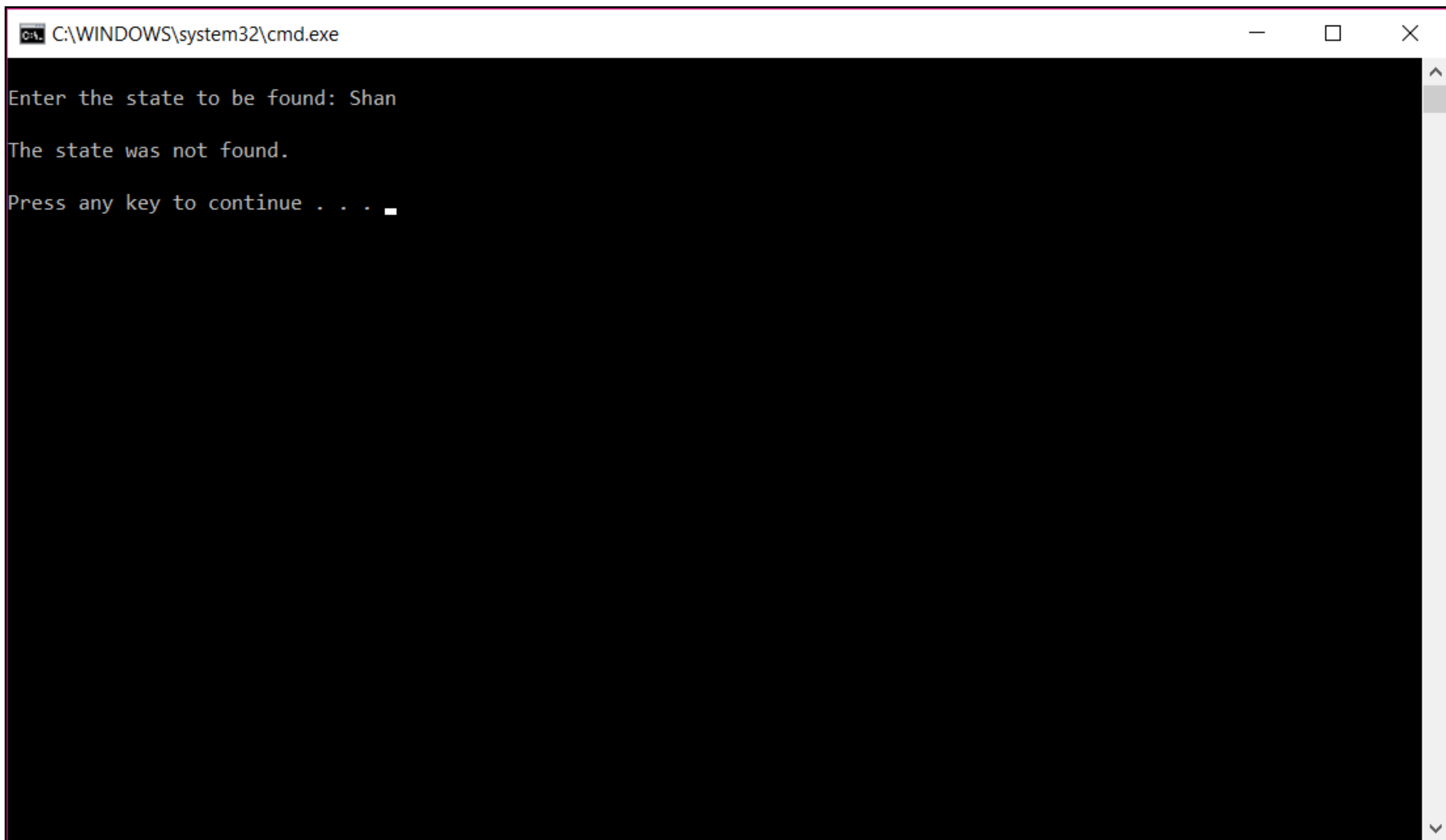
Searching an Array of Strings

- The output of the given program which the state is found is described below:



Searching an Array of Strings

- The output of the given program which the state is not found is described below:



Microprocessor Programming

Practical Assignments (Instructions)

Assignment 1

- Write a complete assembly program using string instructions for the following skeleton of the program:

1. Input name1 and name2,
2. Compare the two names,
3. Determine name1 is greater than or less than name2,
4. Output as The names are the same or,
5. The first name is less than the second or,
6. The first name is greater than the second.

Microprocessor Programming

Practical Assignments (Report)

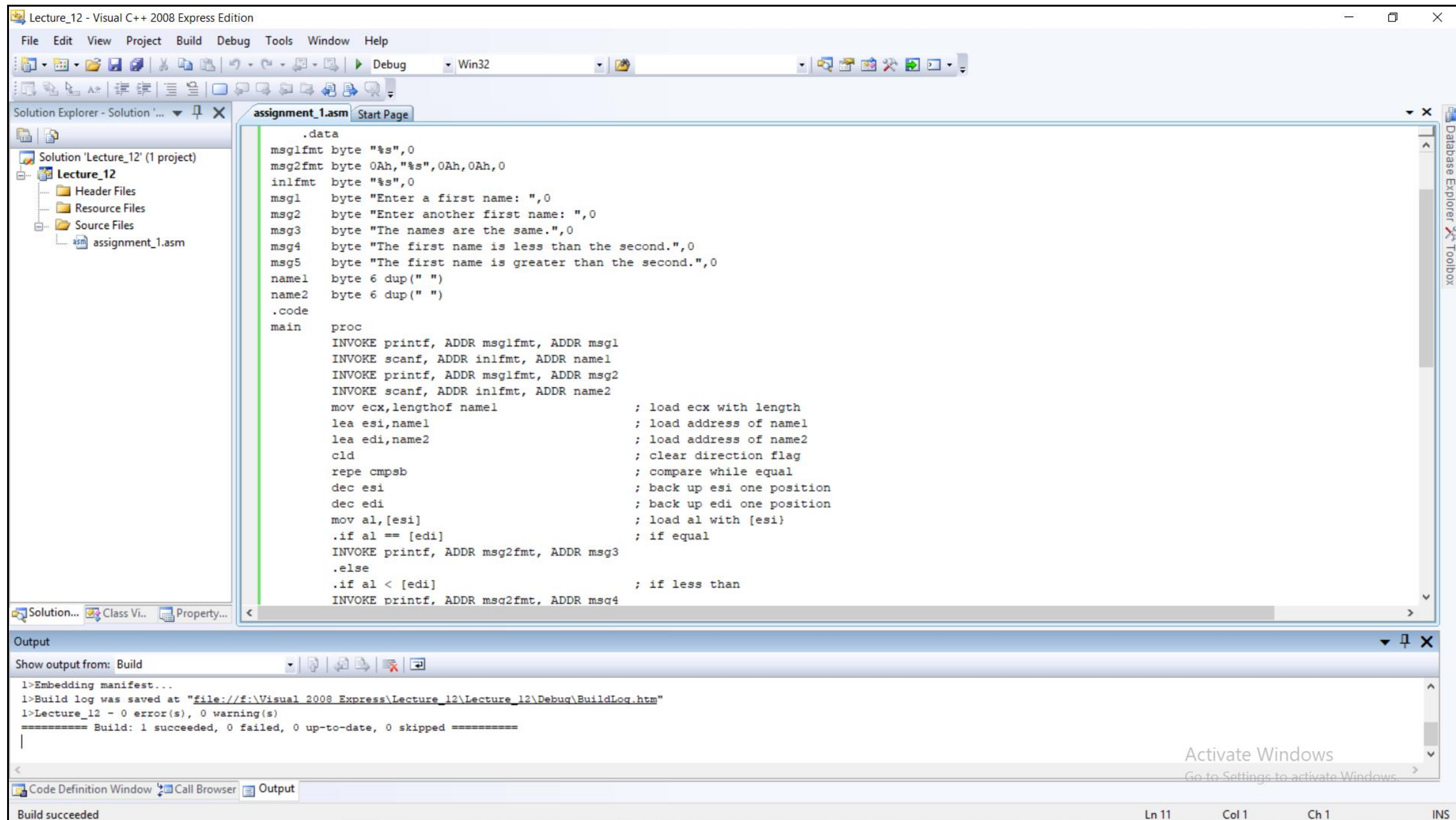
Assignment 1

- A complete assembly program is described below:

```
        .listall
        .386
        .model flat,c
        .stack 100h
scanf   PROTO arg2:Ptr Byte, inputlist:VARARG
printf  PROTO arg1:Ptr Byte, printlist:VARARG
        .data
msg1fmt byte "%s",0
msg2fmt byte 0Ah,"%s",0Ah,0Ah,0
in1fmt  byte "%s",0
msg1    byte "Enter a first name: ",0
msg2    byte "Enter another first name: ",0
msg3    byte "The names are the same.",0
msg4    byte "The first name is less than the second.",0
msg5    byte "The first name is greater than the second.",0
name1   byte 6 dup(" ")
name2   byte 6 dup(" ")
        .code
main    proc
        INVOKE printf, ADDR msg1fmt, ADDR msg1
        INVOKE scanf, ADDR in1fmt, ADDR name1
        INVOKE printf, ADDR msg1fmt, ADDR msg2
        INVOKE scanf, ADDR in1fmt, ADDR name2
        mov ecx,lengthof name1                  ; load ecx with length
        lea esi,name1                           ; load address of name1
        lea edi,name2                           ; load address of name2
        cld                                     ; clear direction flag
        repe cmpsb                              ; compare while equal
        dec esi                                 ; back up esi one position
        dec edi                                 ; back up edi one position
        mov al,[esi]                            ; load al with [esi}
        .if al == [edi]                         ; if equal
        INVOKE printf, ADDR msg2fmt, ADDR msg3
        .else
        .if al < [edi]                          ; if less than
        INVOKE printf, ADDR msg2fmt, ADDR msg4
        .else
        INVOKE printf, ADDR msg2fmt, ADDR msg5
        .endif
        .endif
        ret
main    endp
        end
```

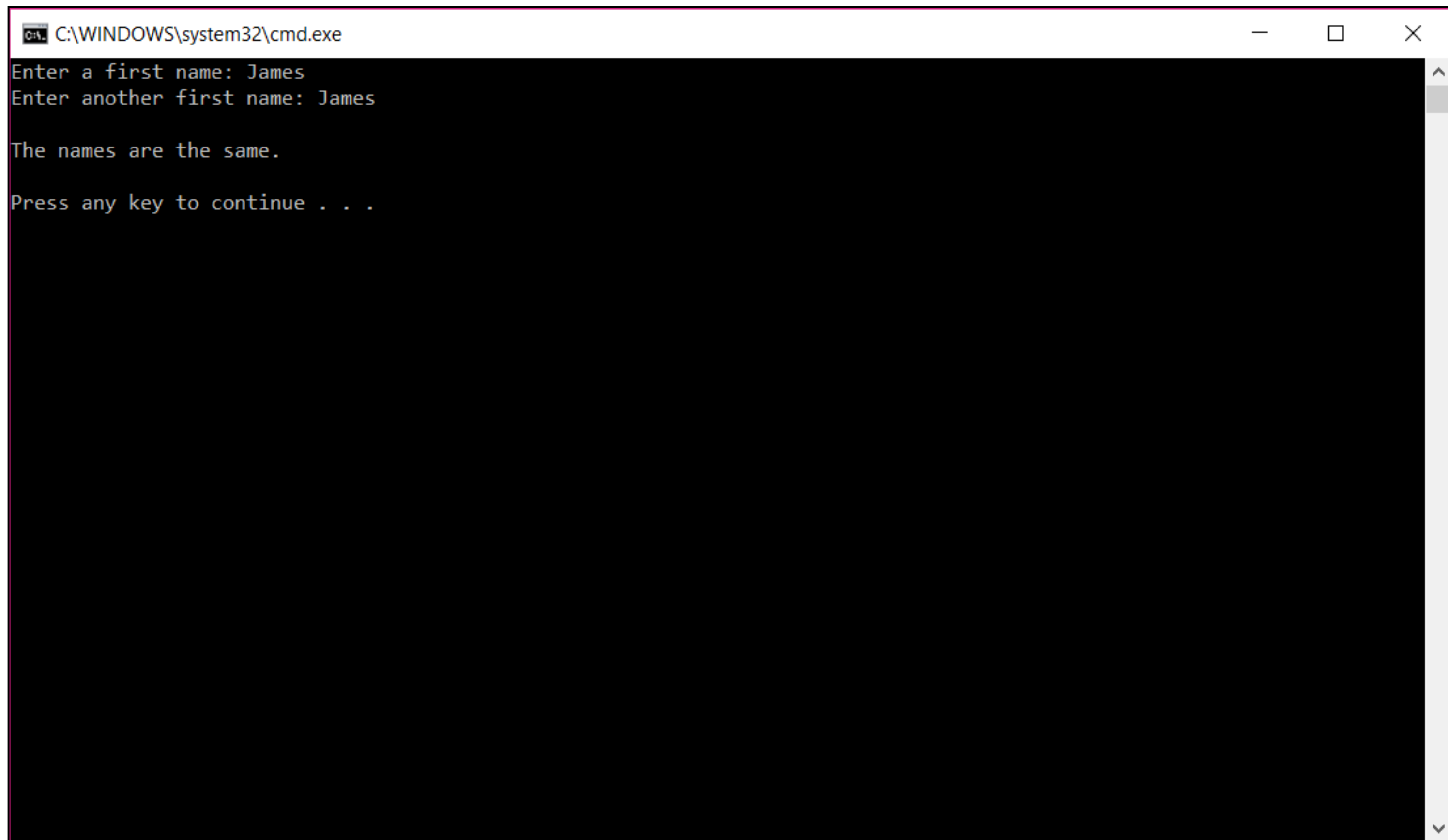
Assignment 1

- A complete assembly program is implemented as follow:



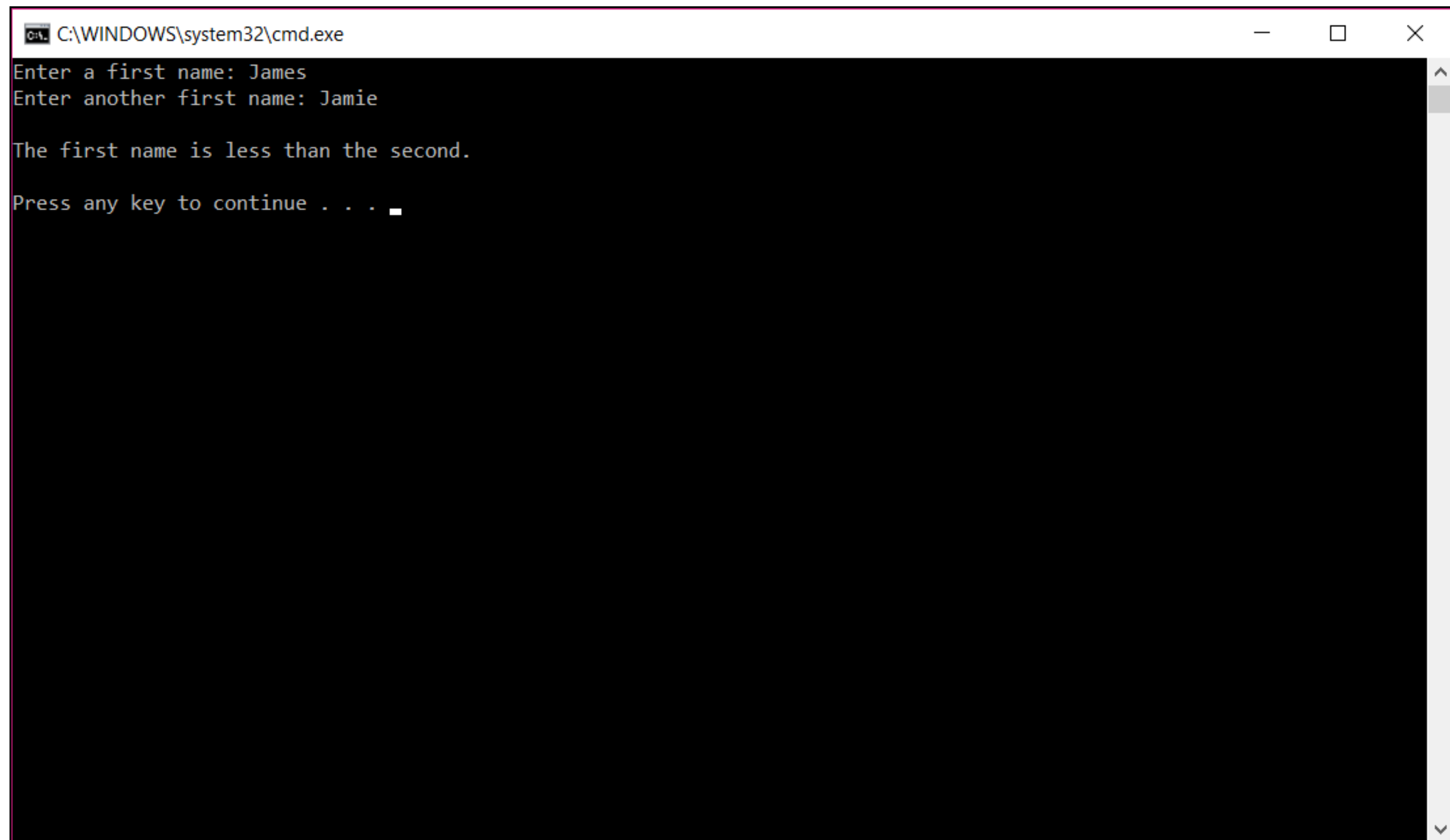
Assignment 1

- The output for the case of the names are the same is described below:



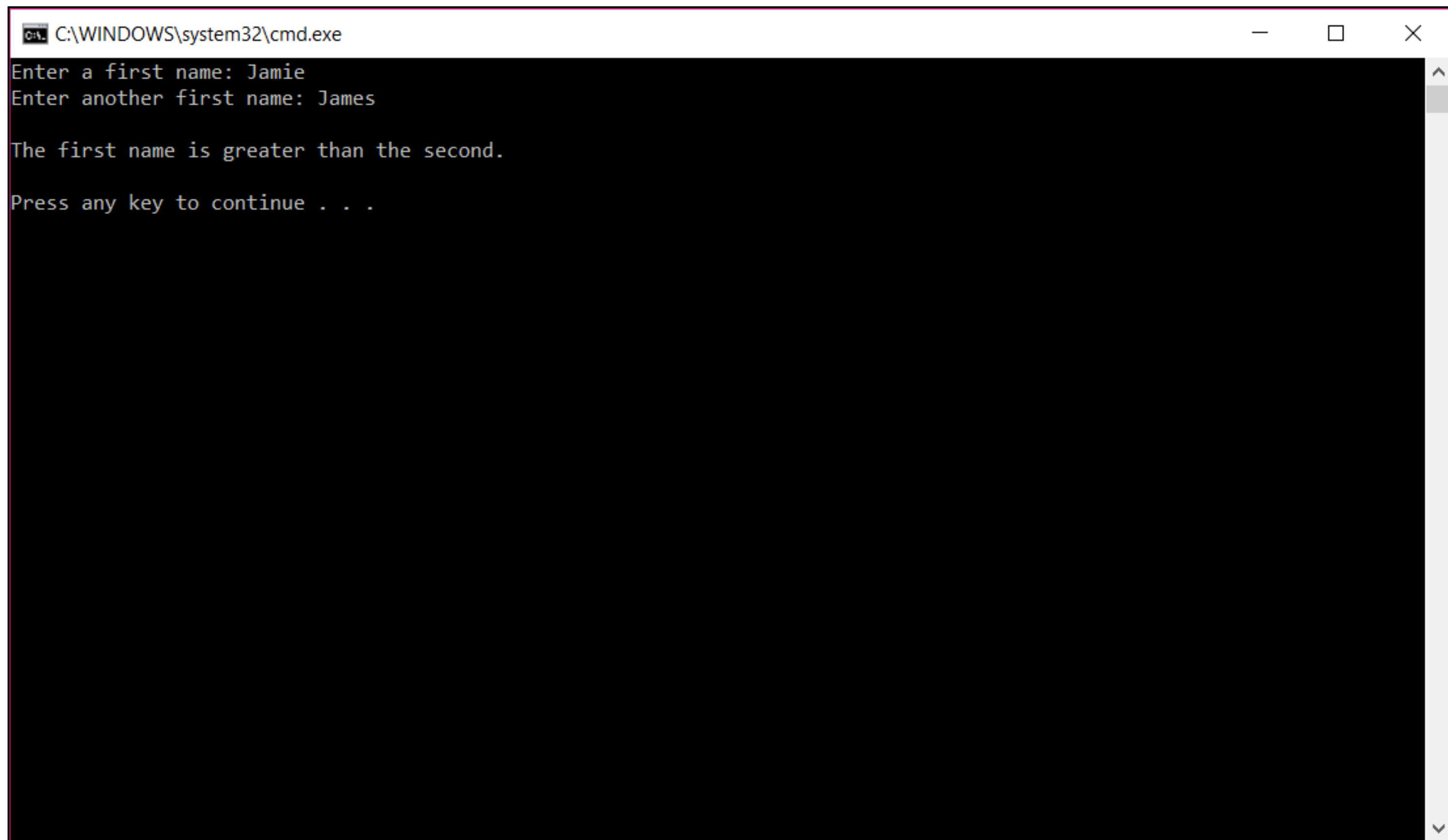
Assignment 1

- The output for the case of the first name is less than the second is described below:



Assignment 1

- The output for the case of the first name is greater than the second is described below:



Thank You