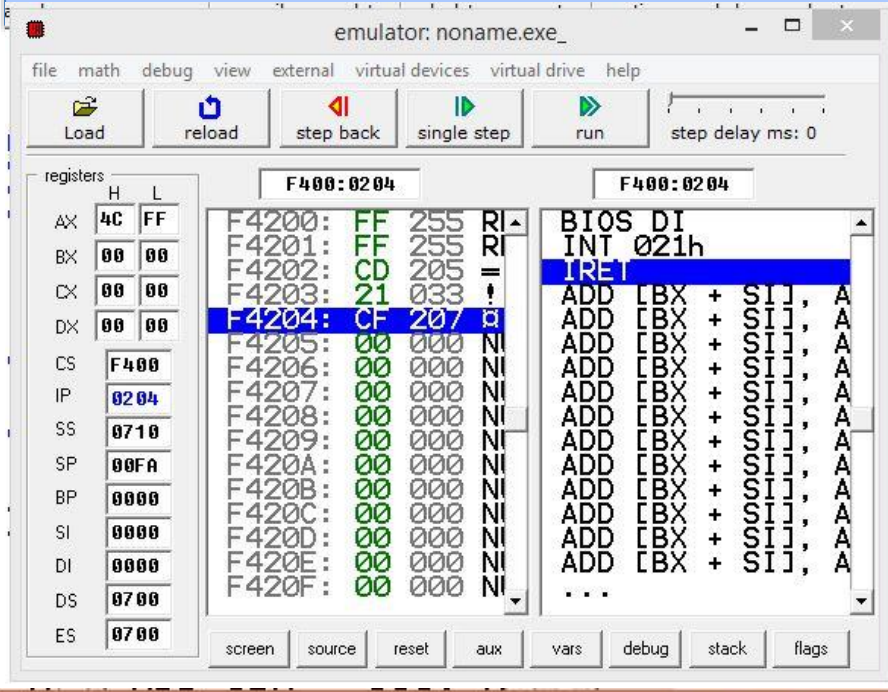
Islamic University of Technology (IUT)
Department of Electrical and Electronic Engineering

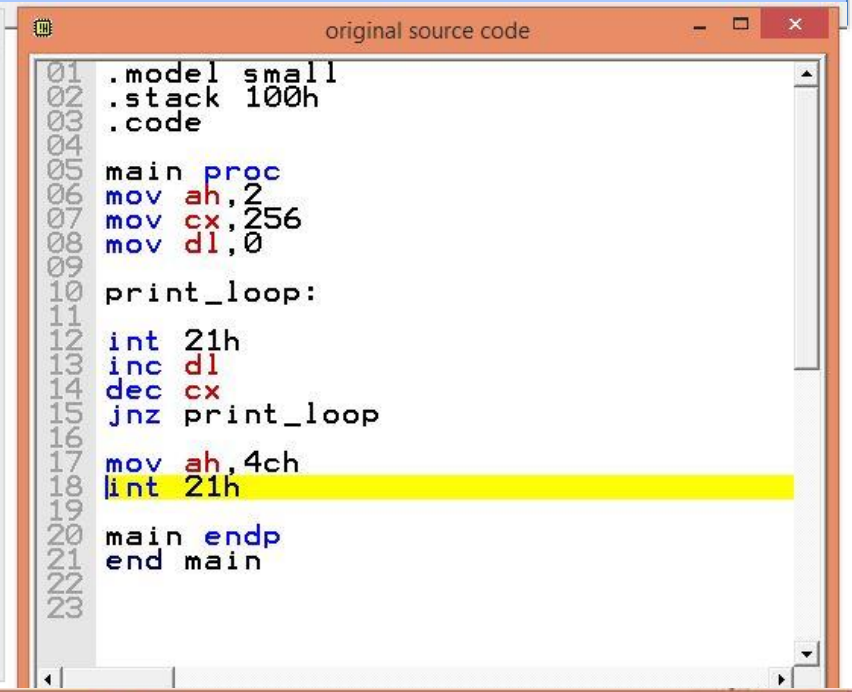
Course: EEE - 4616
Lab: 05
Flow Control Instructions

Contents

- Jump
- Conditional jump
- Unconditional jump
- Branching
- Looping

Jump


emulator: noname.exe_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
H L
AX 4C FF
BX 00 00
CX 00 00
DX 00 00
CS F400
IP 0204
SS 0710
SP 00FA
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
F400:0204
F400:0204
F4200: FF 255
F4201: FF 255
F4202: CD 205
F4203: 21 033
F4204: CF 207
F4205: 00 000
F4206: 00 000
F4207: 00 000
F4208: 00 000
F4209: 00 000
F420A: 00 000
F420B: 00 000
F420C: 00 000
F420D: 00 000
F420E: 00 000
F420F: 00 000
BIOS DI
INT 021h
IRET
ADD [BX + SI], A
...
screen
source
reset
aux
vars
debug
stack
flags


original source code
.model small
.stack 100h
.code
main proc
mov ah,2
mov cx,256
mov dl,0
print_loop:
int 21h
inc dl
dec cx
jnz print_loop
mov ah,4ch
int 21h
main endp
end main


emulator screen (80x25 chars)

original source code

```
01 .model small
02 .stack 100h
03 .code
04
05 main proc
06 mov ah,2
07 mov cx,256
08 mov dl,0
09
10 print_loop:
11
12 int 21h
13 inc dl
14 dec cx
15 jnz print_loop
16
17 mov ah,4ch
18 int 21h
19
20 main endp
21 end main
22
23
```

- Why “ah” is initialized to two?

➢ For single character display

- Which lines are used to display the characters?

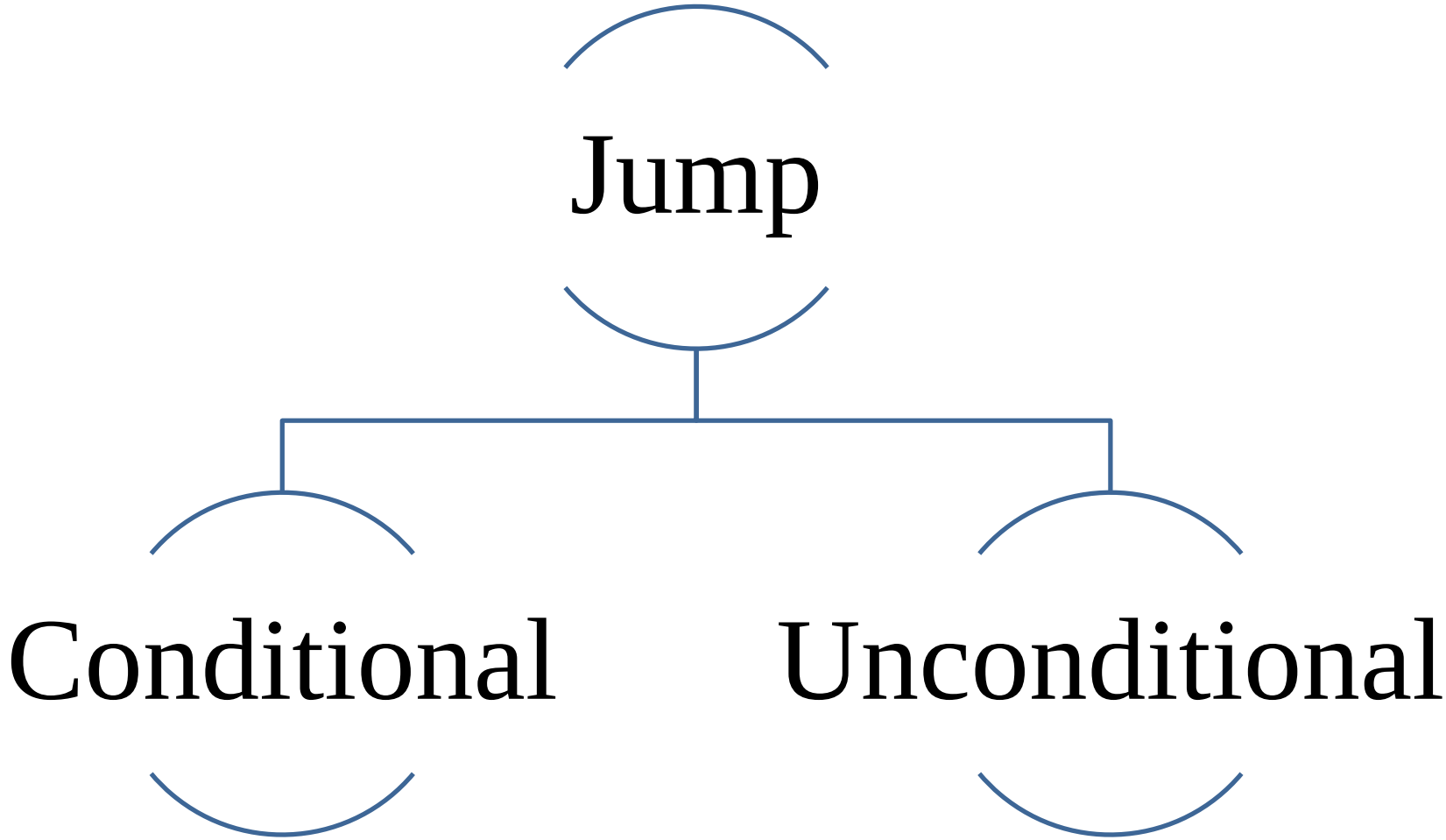
➢ Lines 10 to 15

- Which one is the loop counter?

➢ cx

Label

- Labels are needed in situations where one instruction refers to another
- Labels end with a colon.
- It is a good practice to represent labels in a single line


Jump
Conditional
Unconditional

Conditional Jump

- Syntax is

  Jxxx *destination_label*

- If the condition for the jump is true, the next instruction to be executed is the one at *destination_label*

- If the condition is false the instruction immediately following the jump is done next.

Jump instructions for unsigned numbers

Instruction	Description	Condition	Opposite Instruction
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JA , JNBE	Jump if Above (>). Jump if Not Below or Equal (not <=).	CF = 0 and ZF = 0	JNA, JBE
JB , JNAE, JC	Jump if Below (<). Jump if Not Above or Equal (not >=). Jump if Carry.	CF = 1	JNB, JAE, JNC
JAE , JNB, JNC	Jump if Above or Equal (>=). Jump if Not Below (not <). Jump if Not Carry.	CF = 0	JNAE, JB
JBE , JNA	Jump if Below or Equal (<=). Jump if Not Above (not >).	CF = 1 or ZF = 1	JNBE, JA

```
.model small
.stack 100h
.code

main proc
mov ah,2
mov dl,'='
mov bl,6
mov cl,4

print_loop:

int 21h
inc cl
dec bl
cmp bl,cl
je print_loop
mov dl,'x'
int 21h

mov ah,4ch
int 21h

main endp
end main
```

```
==x
```

```
.model small
.stack 100h
.code

main proc
mov ah,2
mov dl,'='
mov bl,7
mov cl,4

print_loop:

int 21h
inc cl
dec bl
cmp bl,cl
jna print_loop
mov dl,'x'
int 21h

mov ah,4ch
int 21h

main endp
end main
```

```
=x
```

```
.model small
.stack 100h
.code

main proc
mov ah,2
mov dl,'='
mov bl,7
mov cl,4

print_loop:

int 21h
inc cl
dec bl
cmp cl,bl
jb print_loop
mov dl,'x'
int 21h

mov ah,4ch
int 21h

main endp
end main
```

```
==x
```

Jump instructions for signed numbers

Instruction	Description	Condition	Opposite Instruction
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JG , JNLE	Jump if Greater (>). Jump if Not Less or Equal (not <=).	ZF = 0 and SF = OF	JNG, JLE
JL , JNGE	Jump if Less (<). Jump if Not Greater or Equal (not >=).	SF <> OF	JNL, JGE
JGE , JNL	Jump if Greater or Equal (>=). Jump if Not Less (not <).	SF = OF	JNGE, JL
JLE , JNG	Jump if Less or Equal (<=). Jump if Not Greater (not >).	ZF = 1 or SF <> OF	JNLE, JG

EXPLANATION

original sourc

```
01  .model small
02  .stack 100h
03  .code
04
05  main proc
06  mov ah,2
07  mov dl,'='
08  mov bl,-7
09  mov cl,-1
10
11  print_loop:
12
13  int 21h
14  inc bl
15  dec cl
16  cmp bl,cl
17  jle print_loop
18  mov dl,'x'
19  int 21h
20
21  mov ah,4ch
22  int 21h
23
24  main endp
25  end main
26
```

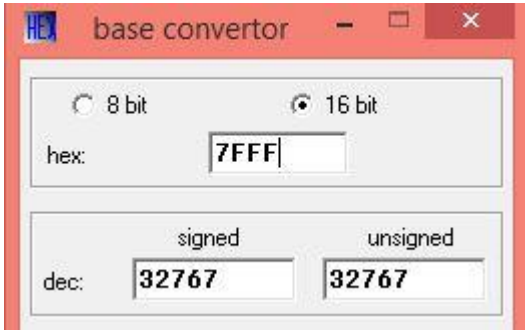
Signed vs Unsigned jump

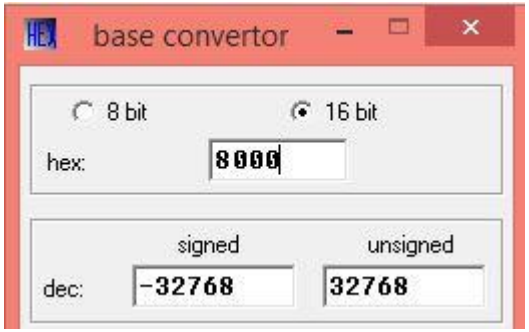
original source

```
.model small
.stack 100h
.code

main proc
mov ah,2
mov dl,'='
mov bx,7FFFH
mov cx,8000H

print_loop:

int 21h
cmp bx,cx
jg print_loop
mov dl,'x'
int 21h

mov ah,4ch
int 21h

main endp
end main
```

```
==========
==========
```

WHY?





original

```
.model small
.stack 100h
.code

main proc
mov ah,2
mov dl,'='
mov bx,7FFFH
mov cx,8000H

print_loop:

int 21h
cmp bx,cx
ja print_loop
mov dl,'x'
int 21h

mov ah,4ch
int 21h

main endp
end main
```

```
=x
```

Jump instructions that test single flag

Instruction	Description	Condition	Opposite Instruction
JZ , JE	Jump if Zero (Equal).	ZF = 1	JNZ, JNE
JC	Jump if Carry	CF = 1	JNC
JS	Jump if Sign.	SF = 1	JNS
JO	Jump if Overflow.	OF = 1	JNO
JPE, JP	Jump if Parity Even.	PF = 1	JPO, JNP
JNZ , JNE	Jump if Not Zero (Not Equal).	ZF = 0	JZ, JE
JNC	Jump if Not Carry	CF = 0	JC
JNS	Jump if Not Sign.	SF = 0	JS
JNO	Jump if Not Overflow.	OF = 0	JO
JPO, JNP	Jump if Parity Odd (No Parity).	PF = 0	JPE, JP

Limitation and way around

- The structure of the machine code of a conditional jump requires that *destination_label* must precede the jump instruction by no more than 126 bytes or follow it by no more than 127 bytes.
- Solution: Use unconditional jump
- Syntax: JMP *destination_label*

```
TOP:

;body of the loop

DEC CX
JNZ TOP
MOV AX, BX
```

But the body of the loop contains so many instructions that the label TOP is out of the range for JNZ.

```
TOP:

;body of the loop

DEC CX
JNZ  BOTTOM
JMP EXIT

BOTTOM:
        JMP TOP

EXIT:
        MOV AX, BX
```

Jump for characters

- In working with the standard ASCII character set, either signed or unsigned jumps may be used, because the sign bit of a byte containing a character code is always zero.

- For extended ASCII characters (codes 80h to FF) , unsigned jumps should be used

Branching

- In high level languages, branching structures enable a program to take different paths depending on conditions. The same thing can be accomplished in assembly language as well.

IF-THEN

PSEUDOCODE ALGORITHM

```
IF AX < 0
        THEN
        REPLACE AX by –AX
END_IF
```

CODE in ASSEMBLY LANGUAGE

```
CMP AX, 0     ; basically executes AX-0
JNL END_IF    ; jump if not less

; if the condition is false, following is executed

NEG AX ; now replace AX by -AX

END_IF:
```

IF-THEN-ELSE

PSEUDOCODE ALGORITHM

```
IF AL <= BL
   THEN
            DISPLAY THE CHARACTER IN AL
   ELSE
            DISPLAY THE CHARACTER IN BL
END_IF
```

CODE in ASSEMBLY LANGUAGE

```
MOV AH, 2 ; to display

CMP AL, BL ; is AL<BL?
JNBE ELSE_ ; no, i.e BL<AL, so jump

MOV DL, AL; yes, so display AL

JMP DISPLAY

ELSE_:
MOV DL, BL; display BL

DISPLAY:
INT 21H
```

Case

PSEUDOCODE ALGORITHM

```
CASE AX
        < 0; PUT -1 IN BX
        = 0; PUT 0 IN BX
        > 1; PUT 1 IN BX
END_CASE
```

CODE IN ASSEMBLY LANGUAGE

```
CMP AX, 0 ;is AX < 0?
        JL NEGATIVE ;yes, AX < 0
        JE ZERO ; no, AX = 0
        JG POSITIVE ; no, AX >0
NEGATIVE:
        MOV BX, -1
        JMP END_CASE
ZERO:
        MOV BX, 0
        JMP END_CASE
POSITIVE:
        MOV BX, 1
        JMP END_CASE
END_CASE:
```

Use of ‘or’ in case

PSEUDOCODE ALGORITHM

```
CASE AL
        1,3; DISPLAY ‘o’
        2,4: DISPLAY ‘e’
END_CASE
```

CODE IN ASSEMBLY LANGUAGE

```
CMP AL, 1
        JE ODD
CMP AL, 3
        JE ODD
CMP AL, 2
        JE EVEN
CMP AL, 4
        JE EVEN
        JMP END_CASE
ODD:
        MOV DL, ‘o’
        JMP DISPLAY
EVEN:
        MOV DL, ‘e’
        JMP DISPLAY
DISPLAY:
        MOV AH, 2
        INT 21H
END_CASE:
```

Use of ‘and’ in assembly language

PSEUDOCODE ALGORITHM

Read a character (into AL)

IF (‘A’ <= character) and (character <= ‘Z’)
display character

END_IF

CODE IN ASSEMBLY LANGUAGE

```
MOV AH, 1
INT 21H

CMP AL, ‘A’
JNGE END_IF

CMP AL, ‘Z’
JNLE END_IF

MOV DL, AL

MOV AH, 2
INT 21H
END_IF:
```

Looping structure

- A loop is a sequence of structures that is repeated.
- The number of times to repeat may be known in advance, or it may depend on conditions.
- Two loops will be discussed
  - For loop
  - While loop

For loop

- Syntax : LOOP *destination_label*
- The counter for the LOOP is the CX register which is initialized to loop_count.
- Execution of the LOOP instruction causes CX to be decremented automatically
- If CX is not 0, control transfers to *destination_label*.
- *destination_label* must precede the LOOP instruction by no more than 126 bytes.

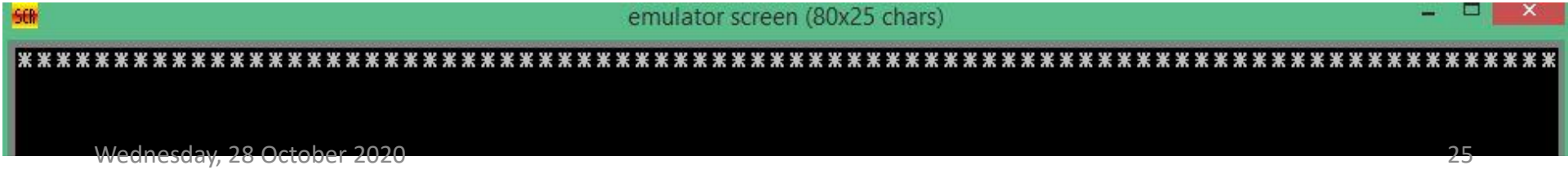
For loop

PSEUDOCODE ALGORITHM

FOR 80 times DO
DISPLAY ‘*’
END_FOR

original so

```
01  .MODEL SMALL
02  .STACK 100H
03  .CODE
04
05  MAIN PROC
06
07  MOV CX, 80
08  MOV DL, '*'
09  MOV AH, 2
10  JCXZ SKIP
11
12  TOP:
13
14  INT 21H
15  LOOP TOP
16
17  SKIP:
18
19  MOV AH, 4CH
20  INT 21H
21
22  MAIN ENDP
23  END MAIN
24
```



While loop

PSEUDOCODE ALGORITHM

INITIALIZE COUNT TO 0
READ A CHARACTER
WHILE CHARACTER <>
CARRIAGE_RETURN DO
COUNT=COUNT+1
END_WHILE

original source co

```
01  .MODEL SMALL
02  .STACK 100H
03  .CODE
04
05  MAIN PROC
06
07  MOV DL, 0
08  MOV AH, 1
09  INT 21H
10
11  WHILE_:
12
13  CMP AL, 0DH
14  JE END_WHILE
15  INC DL
16  INT 21H
17  JMP WHILE_
18
19  END_WHILE:
20
21  MOV BL,DL
22
23  MOV AH,2
24  MOV DL, 0DH
25  INT 21H
26
27  MOV AH,2
28  MOV DL, 0AH
29  INT 21H
30
31  MOV DL,BL
32  INT 21H
33
34  MAIN ENDP
35  END MAIN
36
37
```

SELF STUDY

- Repeat loop
- Repeat vs While loop
- Programming with high level structures

Tasks

1. Write a program to display a “?”, read two capital letters, and display them on the next line in alphabetical order.

2. Write a program to display the extended ASCII characters (ASCII codes 80h to FFh). Display 10 characters per line, separated by blanks. Stop after the extended characters have been displayed once.

3. Write a program that will prompt the user to enter a hex digit character (“O"....“9" or “A”... “F”), display it on the next line in decimal, and ask the user if he or she wants to do it again. If the user types “y” or “Y", the program repeats; if the user types anything else, the program terminates. If the user enters an illegal character, prompt the user to try again.

Sample:

ENTER A HEX DIGIT: 9
IN DECIMAL IS IT 9
DO YOU WANT TO DO IT AGAIN? y
ENTER A HEX DIGIT: c
ILLEGAL CHARACTER - ENTER 0..9 OR A..F: C
IN DECIMAL IT IS 12
DO YOU WANT TO DO IT AGAIN? N

Tasks

4. Do the previous problem again, except that if the user fails to enter a hex-digit character in three tries, display a message and terminate the program.

5. Write a program that reads a string of capital letters, ending with a carriage return, and displays the longest sequence of consecutive alphabetically increasing capital letters read.

Sample:

ENTER A STRING OF CAPITAL LETTERS:
FGHADEFGHC
THE LONGEST CONSECUTIVELY INCREASING STRING IS:
DEFGH