HW 10 : Theme: Strings and Interrupts

*(All main questions carry equal weight.  Credit awarded to only those answers for which work has been shown.)*

1. [Interrupts] *What are hardware and software interrupts*? *Give examples of each.*

A hardware interrupt is an electronic alerting signal sent to the processor from an external device, like a disk controller or an external peripheral.

A software interrupt is a call to an operating system procedure. Most of these procedures, called interrupt handlers, provide input–output capability to application programs.

•Examples of hardware interrupts are:

•INTR

* An INTR line could be connected to
* Disk, printer, camera, etc
* It can be connected to a MUX such as 8279 which can then support more than one device interrupting.
* The external device raises the INTR line requesting an interrupt
* The CPU acknowledges the interrupt by lowering the INTA/ line
* The external device sends its device id on the address lines, which the CPU can sample to know which device has interrupted

•NMI or Non-Maskable Interrupt.

•RESET – triggered by the Power On/Off button

* Clears the registers and flags, the corresponding handler directs to the boot routines of the OS (residing in the kernel)

•There are specific pins on the CPU chip corresponding to each one of them(hardware interrupts).

•Reset and NMI are level-triggered whereas INTR is edge-triggered.

• Examples of software interrupts are:

• **INT Instruction**

* The INT instruction executes a software interrupt.
* The code that handles the interrupt is called an interrupt handler.
* Syntax: INT number (number = 0..FFh)

•**TRAP FLAG:**

•Trap is triggered by exceptions, such as divide by zero or memory bounds error

• One can use the overflow flag to also cause a software interrupt by executing the instruction INTO and writing an appropriate handler

•INT 21h, INT 10h

Software Interrupts are used for such tasks as the following:

• Displaying characters and strings

• Reading characters and strings from the keyboard

• Displaying text in color

• Opening and closing files

• Reading data from files

• Writing data to files

• Setting and retrieving the system time and date

*What are maskable and non-maskable interrupts?  Provide examples of each.*

A maskable interrupt is a hardware interrupt that can be disabled or ignored by the instructions of CPU.

* e.g. INT, INTR

A non-maskable interrupt is a hardware interrupt that cannot be disabled or ignored by CPU instructions. The interrupts are either edge-triggered or level-triggered.

* e.g. NMI, Reset

1. [Interrupts] Explain the process of interrupt vectoring.

Interrupt Vectoring

The CPU processes the INT instruction using the interrupt vector table, which, as we’ve mentioned, is a table of addresses in the lowest 1024 bytes of memory. Each entry in this table is a 32-bit segment-offset address that points to an interrupt handler. The actual addresses in this table vary from one machine to another.

• Step 1: The operand of the INT instruction is multiplied by 4 to locate the matching interrupt vector table entry.

• Step 2: The CPU pushes the flags and a 32-bit segment/offset return address on the stack, disables hardware interrupts, and executes a far call to the address stored at location (10h \* 4) in the interrupt vector table (F000:F065).

• Step 3: The interrupt handler at F000:F065 executes until it reaches an IRET (interrupt return) instruction.

• Step 4: The IRET instruction pops the flags and the return address off the stack, causing the processor to resume execution immediately following the INT 10h instruction in the calling program.

1. [Strings] Write a program that computes the number of characters in any string.  Test the robustness of your program using different strings including those of size 0.

**hw10-3.asm**

include irvine32.inc

;A program that computes the number of characters in any string.

;Test the robustness of your program using different strings including those of size 0.

.data

prompt BYTE "Enter a string: ", 0

SomeString1 BYTE ?

msg BYTE "Total number of characters: ", 0

MAX = 80

COUNT = 0

.code

main proc

mov edx, OFFSET prompt

call WriteString

call crlf

mov edx, OFFSET SomeString1

mov ecx, MAX

call ReadString

mov ecx, LENGTHOF SomeString1

inc ecx

mov edi, OFFSET SomeString1

mov eax, COUNT

L1: cmp BYTE PTR[edi], 0

je L2

inc edi

inc eax

jmp L1

L2:

call crlf

call WriteDec

invoke ExitProcess, 0

main endp

end main

**hw10-3.lst**

;A program that computes the number of characters in any string.

;Test the robustness of your program using different strings including those of size 0.

00000000 .data

00000000 45 6E 74 65 72 prompt BYTE "Enter a string: ", 0

20 61 20 73 74

72 69 6E 67 3A

20 00

00000011 00 SomeString1 BYTE ?

00000012 54 6F 74 61 6C msg BYTE "Total number of characters: ", 0

20 6E 75 6D 62

65 72 20 6F 66

20 63 68 61 72

61 63 74 65 72

73 3A 20 00

= 00000050 MAX = 80

= 00000000 COUNT = 0

00000000 .code

00000000 main proc

00000000 BA 00000000 R mov edx, OFFSET prompt

00000005 E8 00000000 E call WriteString

0000000A E8 00000000 E call crlf

0000000F BA 00000011 R mov edx, OFFSET SomeString1

00000014 B9 00000050 mov ecx, MAX

00000019 E8 00000000 E call ReadString

0000001E B9 00000001 mov ecx, LENGTHOF SomeString1

00000023 41 inc ecx

00000024 BF 00000011 R mov edi, OFFSET SomeString1

00000029 B8 00000000 mov eax, COUNT

0000002E 80 3F 00 L1: cmp BYTE PTR[edi], 0

00000031 74 04 je L2

00000033 47 inc edi

00000034 40 inc eax

00000035 EB F7 jmp L1

00000037 L2:

00000037 E8 00000000 E call crlf

0000003C E8 00000000 E call WriteDec

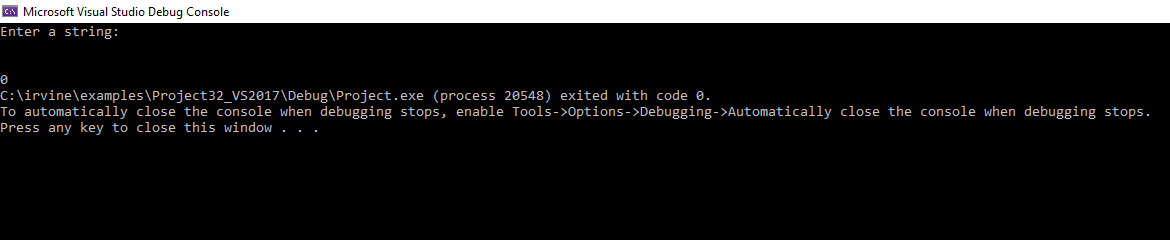
invoke ExitProcess, 0

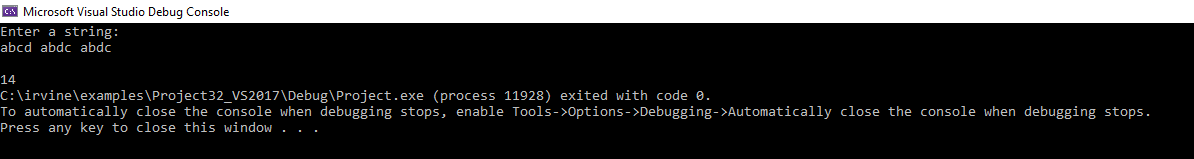
00000041 6A 00 \* push +000000000h

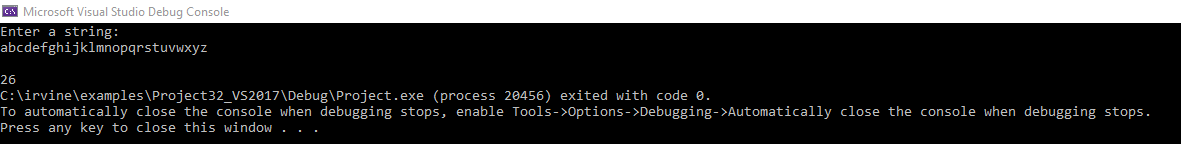
00000043 E8 00000000 E \* call ExitProcess

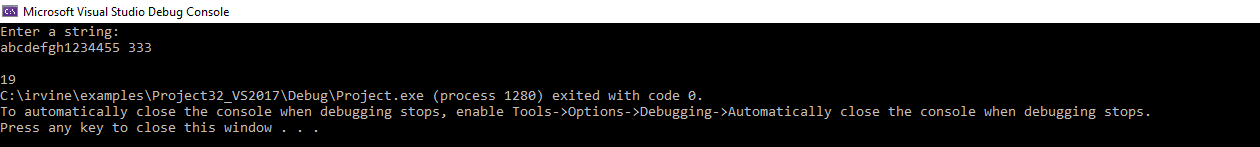
00000048 main endp

end main









1. [Structures] Using the structure example discussed in the book and slides, write a program that displays the *z*-coordinates of several points given as an array of coordinates in the data segment. Unlike the example in the book you should use 3-dimensional points. Test your program with various *(x, y, z)* Use base-indexed addressing to implement the program.

**hw10-4.asm**

include irvine32.inc

;Write a program that displays the z-coordinates of several points given as an array of coordinates in the data segment.

;Unlike the example in the book you should use 3-dimensional points.

;Test your program with various (x, y, z)

;Use base-indexed addressing to implement the program.

Point STRUCT

X SWORD ?

Y SWORD ?

Z SWORD ?

Point ENDS

.data

xyzCoords Point {-1,2,3}, {-3,4,-5}, {-5,7,9}, {6,8,0}, {7,3,-5}

.code

main PROC

mov esi, OFFSET xyzCoords

mov ax, (Point PTR[esi]).Z

movsx eax, ax

call WriteInt

mov ax, (Point PTR[esi + 6]).Z

movsx eax, ax

call WriteInt

mov ax, (Point PTR[esi + 12]).Z

movsx eax, ax

call WriteInt

mov ax, (Point PTR[esi + 18]).Z

movsx eax, ax

call WriteInt

mov ax, (Point PTR[esi + 24]).Z

movsx eax, ax

call WriteInt

invoke ExitProcess, 0

main endp

end main

**hw10-4.lst**

;Write a program that displays the z-coordinates of several points given as an array of coordinates in the data segment.

;Unlike the example in the book you should use 3-dimensional points.

;Test your program with various (x, y, z)

;Use base-indexed addressing to implement the program.

00000006 Point STRUCT

00000000 0000 X SWORD ?

00000002 0000 Y SWORD ?

00000004 0000 Z SWORD ?

Point ENDS

00000000 .data

00000000 FFFF 0002 0003 xyzCoords Point {-1,2,3}, {-3,4,-5}, {-5,7,9}, {6,8,0}, {7,3,-5}

FFFD 0004 FFFB

FFFB 0007 0009

0006 0008 0000

0007 0003 FFFB

00000000 .code

00000000 main PROC

00000000 BE 00000000 R mov esi, OFFSET xyzCoords

00000005 66| 8B 46 04 mov ax, (Point PTR[esi]).Z

00000009 0F BF C0 movsx eax, ax

0000000C E8 00000000 E call WriteInt

00000011 66| 8B 46 0A mov ax, (Point PTR[esi + 6]).Z

00000015 0F BF C0 movsx eax, ax

00000018 E8 00000000 E call WriteInt

0000001D 66| 8B 46 10 mov ax, (Point PTR[esi + 12]).Z

00000021 0F BF C0 movsx eax, ax

00000024 E8 00000000 E call WriteInt

00000029 66| 8B 46 16 mov ax, (Point PTR[esi + 18]).Z

0000002D 0F BF C0 movsx eax, ax

00000030 E8 00000000 E call WriteInt

00000035 66| 8B 46 1C mov ax, (Point PTR[esi + 24]).Z

00000039 0F BF C0 movsx eax, ax

0000003C E8 00000000 E call WriteInt

invoke ExitProcess, 0

00000041 6A 00 \* push +000000000h

00000043 E8 00000000 E \* call ExitProcess

00000048 main endp

end main

z-coordinates of the following points:

(-1,2,3) , (-3, 4, 5), (-5, 7, 9), (6, 8, 0), (7, 3, -5)

