CS253: Laboratory session 2, Typing tutor and Speed Test

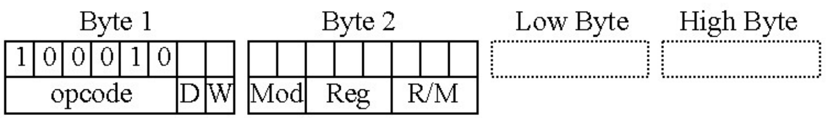
Preamble: Many instructions require extra data to operate. For example, the ADD AX, operation needs to know what to add to AX so as to work. The extra data (operand) can come from many sources. The sources include another register, data in the code following the instruction, data specified by an address and data identified by a register acting as a pointer. The various methods used by an instruction/CPU to locate data (for reading or writing) are known as addressing modes. The four main addressing modes available on most CPUs are as follows.

Immediate Addressing:	The operand (data) appears in the instruction. e.g. mov ax,10
Register Addressing:	The operand is contained in another register. e.g. mov ax,bx
Direct Addressing:	Specifies location in memory (address) where data is to be found. e.g. mov ax,DS:Count
Register Indirect Addressing:	A register is used to provide an offset address in a segment from which the data is obtained (or written to). e.g. mov ax,[bx]

Look at both the machine code and the MASM code of the following listing and located the operator and operand. Note the addressing mode, see code comments.

```
Microsoft (R) Macro Assembler Version 6.00
                              .model small
                              .stack
 0000                         .data
 0000 09                      ct1 db 9
 0001 0203                    ct2 dw 515

 0000                         .code
                              .startup

;Address  MachineCode         ASM code       ; Immediate Addressing
 001A  B8 05 68               back:  mov ax,568h
 001D  B2 08                         mov dl,8
 001F  72 F9                         jc back
 0021  B0 41                         mov al,'A'
 0023  83 F8 0A                      cmp ax,10
 0026  D1 DA                         rcr dx,1

                                            ; Register Addressing
 0028  8B C3                         mov ax,bx
 002A  22 C2                         and al,dl
 002C  03 C3                         add ax,bx

                                            ; Direct Addressing
 002E  2E: A0 0000 R                 mov al,ct1     ; Contains 9 at run time
 0032  A1 0001 R                     mov ax,DS:ct2  ; Contains 515 at run time

                                            ; Register Indirect
 0035  8B 07                         mov ax,[bx]
                                     .exit
                              END
Symbols:
                N a m e              Type      Value    Attr
back . . . . . . . . . . . . . .     L Near    001A     _TEXT
ct1  . . . . . . . . . . . . . .     Byte      0000     _TEXT
ct2  . . . . . . . . . . . . . .     Word      0001     _TEXT
```

Instruction Format: A binary number represents each machine code instruction, typically two bytes. At first sight it would appear that there is no correlation between the number and the instruction however this is not the case. Each binary digit in the number carries information about the instruction. The first 6 bits specifies the instruction type (e.g. mov, jmp, rol, etc). The following 12 bits specify the addressing mode, source or destination, data size (byte or word) etc.



opcode specifies the function in this case MOV
D gives direction, From Register D=0, To Register D=1
Reg: Used to identify the register e.g. dl,dx=010, ah,sp=100, dl,dx=010 etc.
W=0 To move a word (e.g. AX) W=1 To move a byte (e.g. BL)
Mod: Addressing mode, offset information
R/M: Addressing mode, the register

To aid understanding, look at the binary code of each of the instructions in the hello world programme listed below, note how the D and W columns work.

```
Machine Code (Binary)                                Machine Code(HEX) MASM Code
Opcode  D W  Mod Reg R/M
 [101110,1,1] [00,000,000] [0000,0000] [0000,0000]  BB 0000 R         mov bx,OFFSET msg1
 [100010,1,1] [00,010,111]                          8B 17      back: mov dx,[bx]
 [100000,0,0] [11,111,010] [0010,0100]              80 FA 24          cmp dl,'$'
 [011101,0,0] [00,000,111]                          74 07             jz  done
 [101101,0,0] [00,010,010]                          B4 02             mov ah,02h
 [110011,0,1] [00,100,001]                          CD 21             int 021h
 [010000,1,1]                                       43                inc bx
 [111010,1,1] [11,110,010]                          EB F2             jmp back
```

In the early days of assembly language programming you would write out the program in assembly language and then use a large table to look up the machine code (numbers) that corresponds to each line of assembly language (mnemonics). After that you would fill in all the absolute and relative jumps (operand data). Assemblers have simplified the task of creating machine code but have slightly distanced us from the code and how it works. Perhaps we should compile one program by hand sometime during the course.

ASCII: A 7-bit (0-127) number is used to represent the characters printed to the screen or returned by the keyboard. In most cases the number used is 8-bit and the character set is extended to 256 characters (0-255). ASCII (pronounced ASK-KEY) is the most popular standard for coding alphanumeric information. ASCII stands for American Standard Code for Information Interchange. Note that carriage return is coded as 13 decimal and line feed as 10 decimal and escape as 27 decimal, see table later in handout.

Part 1: Get the typing tutor code working using DOSBox at the start.

Enter the following code into a Notepad++ file called *typer.asm*. Save the file in your *X:\CS253* folder (or *C:\temp*), cut and paste could give problems with "'" etc.

```
           .MODEL medium
           .STACK     ; Stack default size 1024 bytes
           .DATA      ; Data segment (for variables)

           .CODE      ; Run-able code goes in code segment
           .STARTUP   ; Handover code from OS call to typer.exe

nextc:     mov ah,8   ; Call int21 with ah=8 returns with
           int 021h   ; al equal to the ASCII character pressed
           mov dl,al  ; dl is assigned value in al, dl=al

           mov ah,02h ; Call int21 with ah=2 prints ASCII
           int 021    ; character represented by value in dl

           cmp dl,'q' ; compare dl with ASCII 'q'=
           jnz nextc  ; if key pressed was not a 'q' go back

           .EXIT      ; Terminate and return control to OS
           END        ; End of file (for compiler)
```

Modify the typing tutor program so that it will only let you type characters 0 to 9 to the screen. You should try using *jc* and *jnc* instructions to achieve this goal, don't use Intel's greater than and less than instructions they prevent an understanding of the C (carry) and Z (zero) flags.

Many uP only have C and Z based jumps. Assembly language is different for each type of processor; it is not a portable language like C or Java that can run on most devices. We are learning a subset of the, x86, assembly language that will give you a good understanding of all the different assembly languages that exist.

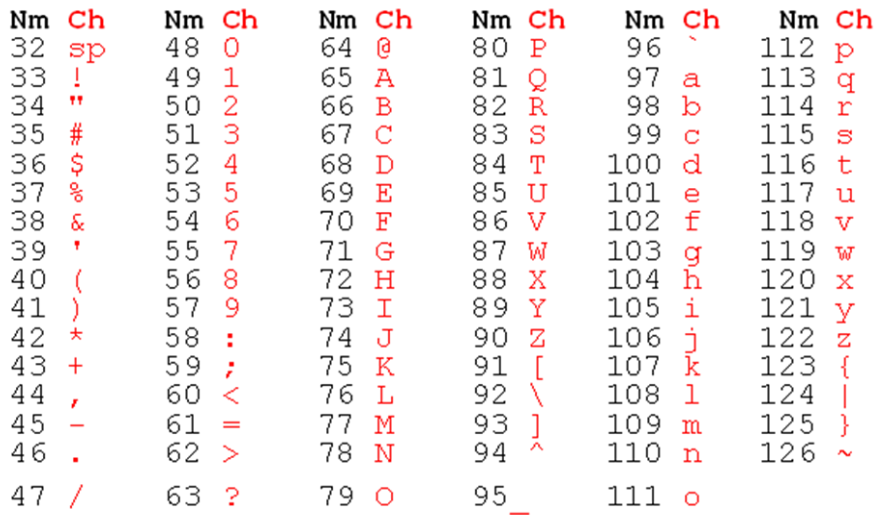
The *jnz* (jump if not zero) and *jz* (jump if zero) work with *cmp* (compare) to test equivalence (A==B or A!=B etc) and *jc* (jump if carry) and *jnc* (jump if no carry) work to test magnitude (A>B or A<=B etc). Make sure the program still terminates when you press 'q' without echoing it to the screen and that the code works correctly for "9" and "0".

Note the instruction *cmp dl,40* would take 40 away from the value in dl and then change the zero and carry flags based on the result of the calculation. The value in dl remains unchanged.

For the record the instruction *sub dl,40* takes 40 away from dl (i.e. it changes dl) and also changes the Z and C flags, The following algorithm could be used to achieve your goal.

```
1 Start
2 Read an ASCII character from the keyboard in to dl
3 If the character code is 133 or a 'q' then goto quit
4 Check if character is less than ASCII code 48, '0', go back to the start
5 Check if character is greater than ASCII code 57, '9', go back to the start
6 Print the character to the screen
7 Go back to the Start
8 Quit
```

The ASCII table shown below should be of help.



Nm	Ch	Nm	Ch	Nm	Ch	Nm	Ch	Nm	Ch	Nm	Ch
32	sp	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		

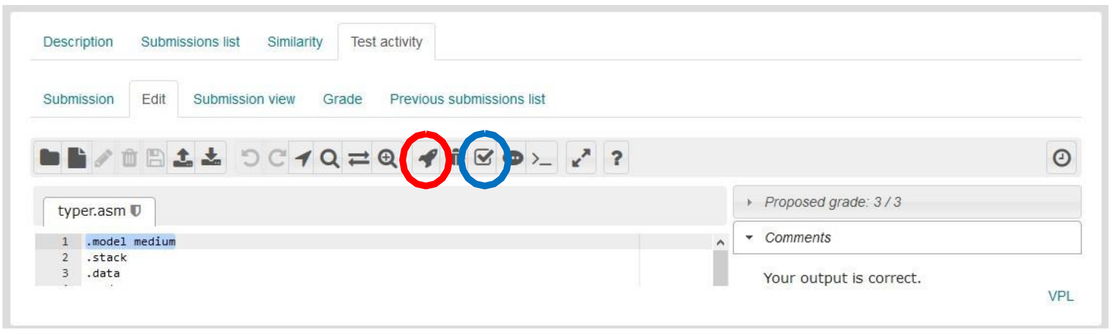
Note: <escape> is ASCII 27 decimal or 1B in HEX.

You should be able to type a string such as "abc123ABC456xyq" into you program from the keyboard and it should display the following text on the screen "123456". Take care that it works properly for both '0' and '9'.

Submit the final working version of your code via the VPL (Virtual Programming Lab) interface.

Well done you have just worked out how assembly language manages magnitude and equivalence tests.

Use the VPL (Virtual programming Laboratory) to build and run your keyboard code. Cut and paste (or retype) the code into the VPL Edit window (maker sure the file is saved as *typer.asm*).

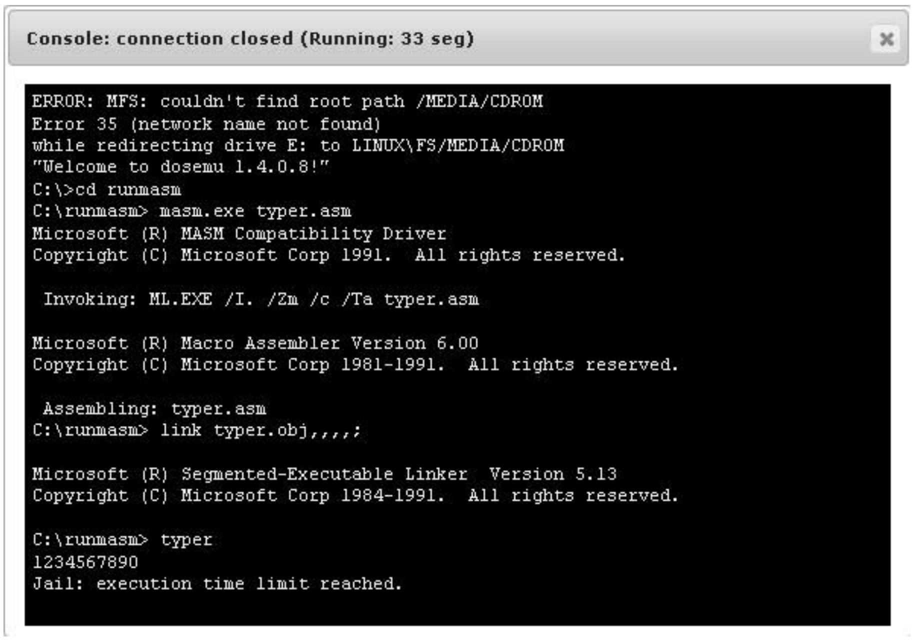


```
.model medium
.stack
.data
```

Your output is correct.

You can click the ***Run*** tab (rocket circled in red) and VPL will compile, link and run the program for you (I think this is easier than DOSBox). If you click inside the console window you can check your program by typing characters from the keyboard, your code is running on a server in the machine room. You can even quit and type *typer* and press <*return*> to restart the program. We have set a time limit of 60 seconds to do this before the server times out. You could re-run if you wish.

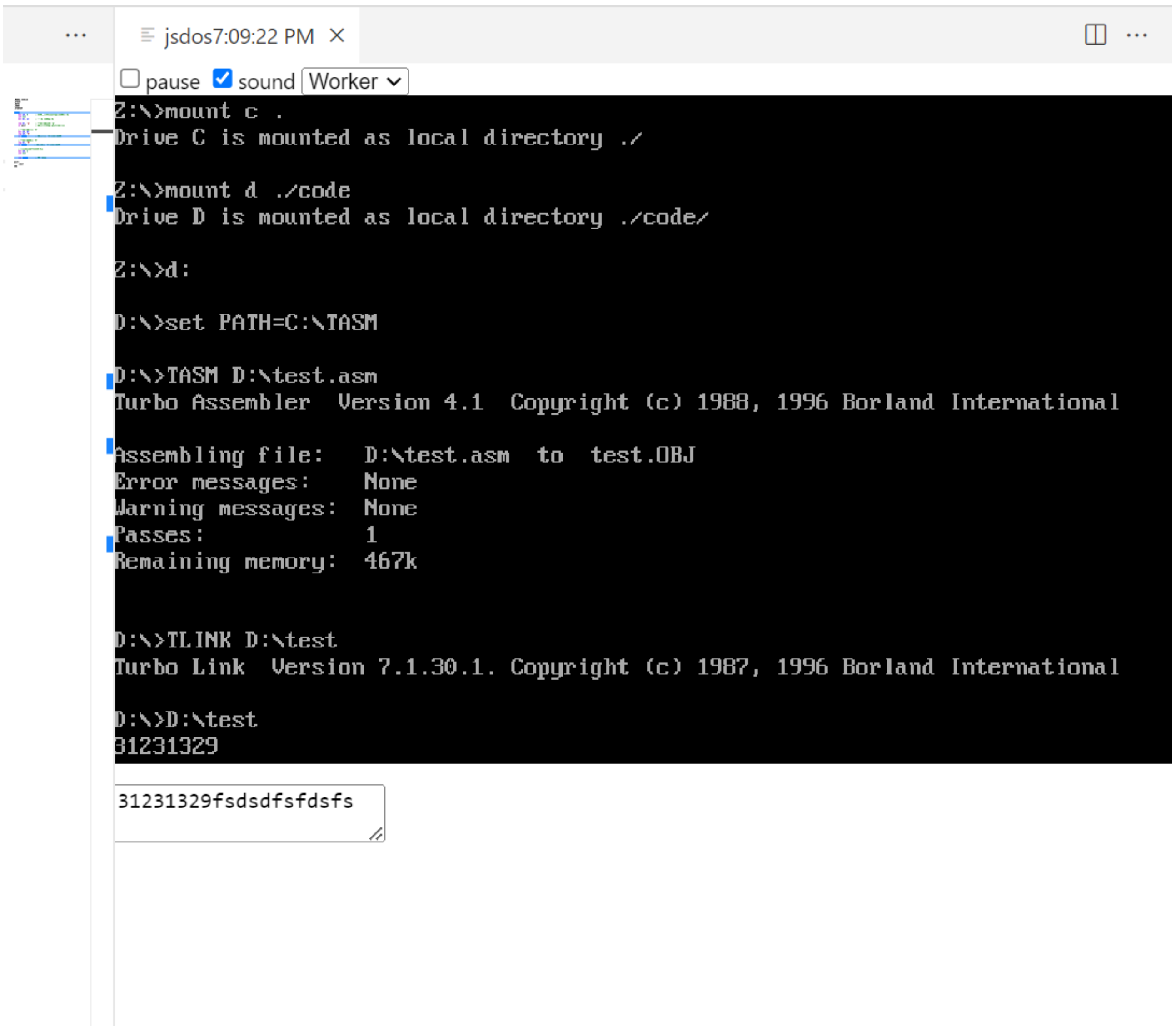
Click in this window and type "abc1234567890DEFq" from the keyboard to see what we mean; you should see something similar to that shown below.



```
Console: connection closed (Running: 33 seg)

ERROR: MFS: couldn't find root path /MEDIA/CDROM
Error 35 (network name not found)
while redirecting drive E: to LINUX\FS/MEDIA/CDROM
"Welcome to dosemu 1.4.0.8!"
C:\>cd runmasm
C:\runmasm> masm.exe typer.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1991.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta typer.asm

Microsoft (R) Macro Assembler Version 6.00
Copyright (C) Microsoft Corp 1981-1991.  All rights reserved.

 Assembling: typer.asm
C:\runmasm> link typer.obj,,,,;

Microsoft (R) Segmented-Executable Linker  Version 5.13
Copyright (C) Microsoft Corp 1984-1991.  All rights reserved.

C:\runmasm> typer
1234567890
Jail: execution time limit reached.
```

Use the VPL to obtain a grade for your assignment using ***Evaluate***, (tick box circled in blue). Evaluate uses the test case of "ADB0123456789cdeq" as an input to test your program. If you did not get full marks try modifying your code to meet all the criteria of the assignment, you may ***Evaluate*** your submission multiple times.

Answer:


jsdos7:09:22 PM

pause sound Worker

```
Z:\>mount c .
Drive C is mounted as local directory ./

Z:\>mount d ./code
Drive D is mounted as local directory ./code/

Z:\>d:

D:\>set PATH=C:\TASM

D:\>TASM D:\test.asm
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996 Borland International

Assembling file:   D:\test.asm  to  test.OBJ
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  467k


D:\>TLINK D:\test
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International

D:\>D:\test
31231329
```

```
31231329fsdsdfsfdsfs
```

```
.MODEL medium
.STACK
.DATA
.CODE
.STARTUP

nextc:
    mov ah, 8       ; 等待使用者按鍵, 結果儲存在 AL
    int 21h
    mov dl, al      ; 將 AL 儲存到 DL


    cmp dl, 'q'     ; 檢查是否按下 'q'
    je quit         ; 如果是, 跳到 quit, 不顯示


    ; 檢查是否小於 '0'
    mov ah, dl
    cmp ah, '0'
    jc nextc        ; 如果小於 '0', 跳回開頭


    ; 檢查是否大於 '9'
    cmp dl, '9'
    ja nextc       ; 如果大於 '9', 跳回開頭
    ; 顯示允許的字元（0~9）
    mov ah, 2
    int 21h


    jmp nextc       ; 重複輸入


quit:
    .EXIT
END
```

Part 2: Copy and paste your working typer.asm program to a new file called alpha.asm. Create this file in the second VPL assignment (if it doesn't exist already).

Modify the code so that it prints all characters typed to the screen. Alphabetical characters should only appear as capitals.

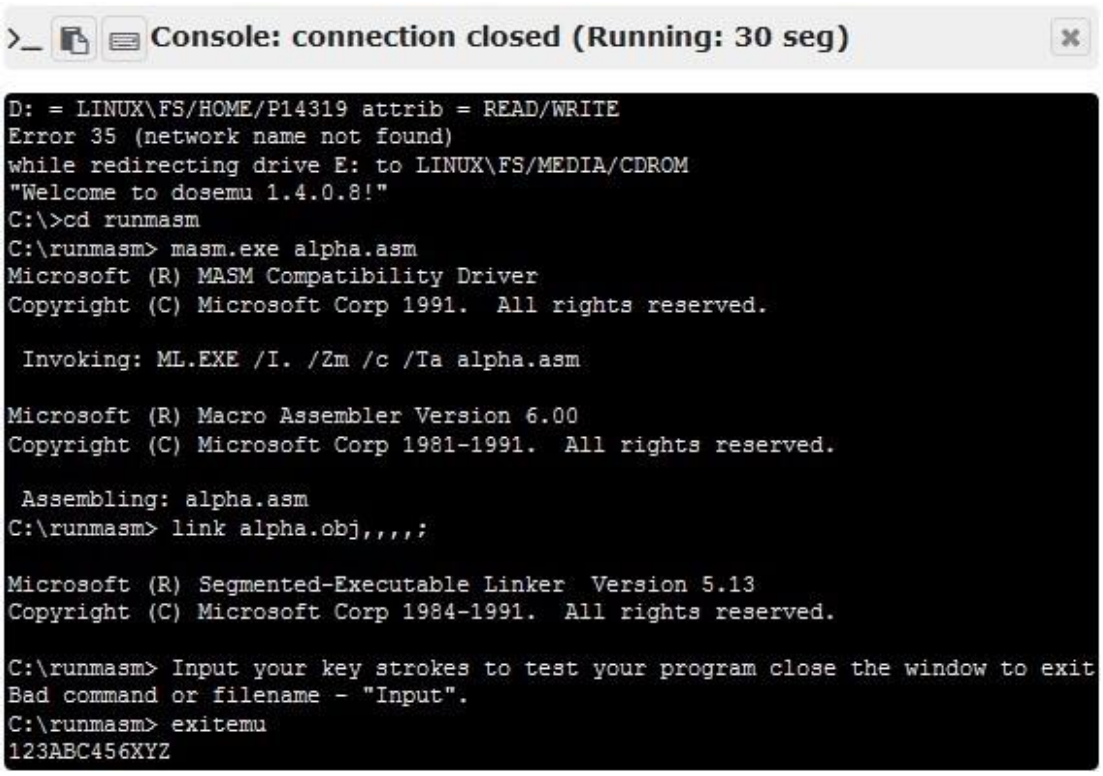
The character code for 'A' is 65 and the character code 'a' is 97. This means that to convert a lowercase character into an upper character you need to subtract (97-65)=32 from the character code.

The assembly language mnemonic for subtraction is `sub dl,123`.

Your program should terminate when the <escape> key is pressed. This key press should not be echoed to the screen.

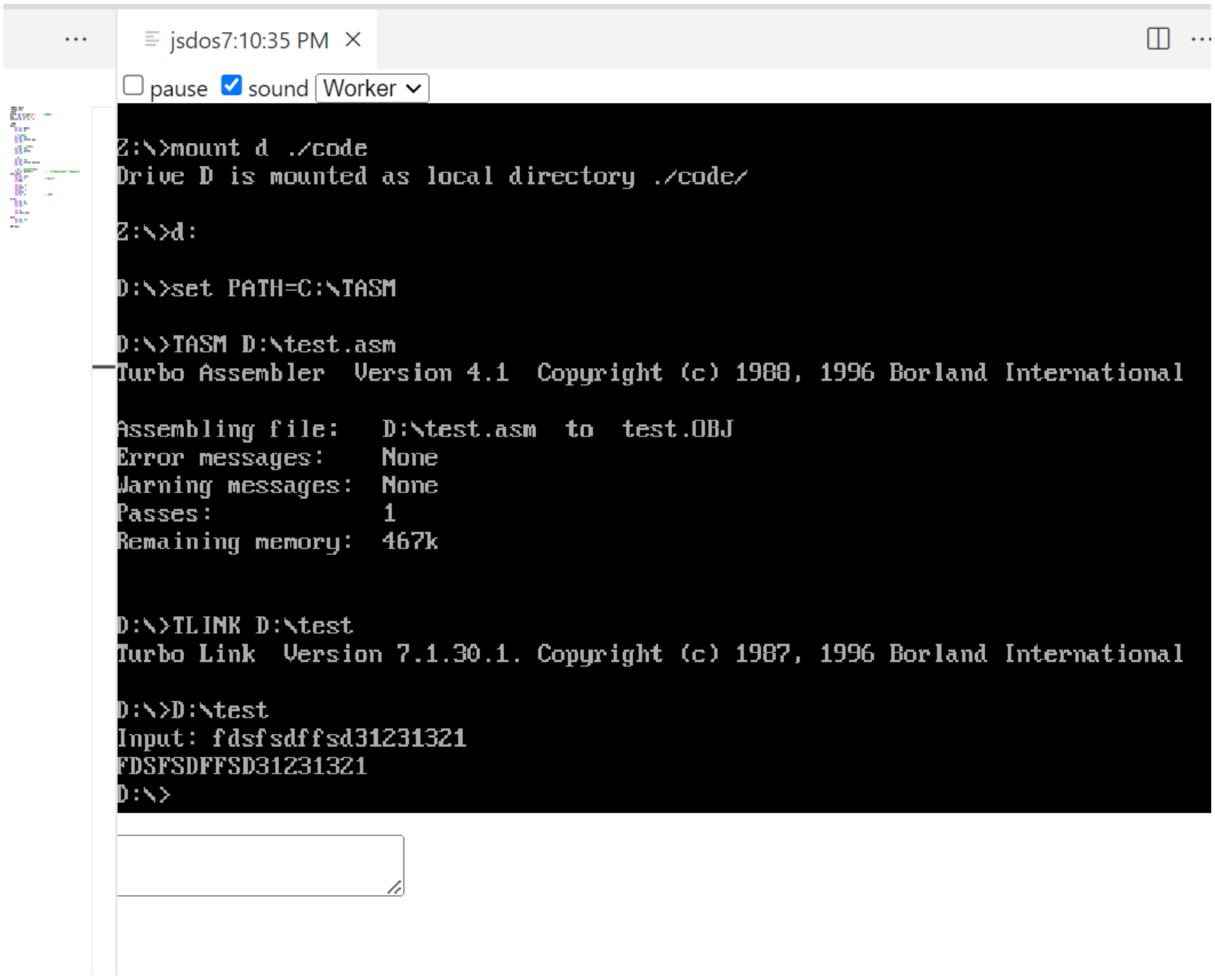
The following figure shows the code output when the following string was entered from the keyboard

123abc456XYZ<escape>



```
D: = LINUX\FS/HOME/P14319 attrib = READ/WRITE
Error 35 (network name not found)
while redirecting drive E: to LINUX\FS/MEDIA/CDROM
"Welcome to dosemu 1.4.0.8!"
C:\>cd runmasm
C:\runmasm> masm.exe alpha.asm
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1991.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta alpha.asm

Microsoft (R) Macro Assembler Version 6.00
Copyright (C) Microsoft Corp 1981-1991.  All rights reserved.

 Assembling: alpha.asm
C:\runmasm> link alpha.obj,,,,;

Microsoft (R) Segmented-Executable Linker  Version 5.13
Copyright (C) Microsoft Corp 1984-1991.  All rights reserved.

C:\runmasm> Input your key strokes to test your program close the window to exit
Bad command or filename - "Input".
C:\runmasm> exitemu
123ABC456XYZ
```

Answer:



```
Z:\>mount d ./code
Drive D is mounted as local directory ./code/

Z:\>d:

D:\>set PATH=C:\TASM

D:\>TASM D:\test.asm
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996 Borland International

Assembling file:   D:\test.asm  to  test.OBJ
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  467k


D:\>TLINK D:\test
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International

D:\>D:\test
Input: fdsfsdffsd31231321
FDSFSDFFSD31231321
D:\>
```

```
.MODEL small
.STACK 100h
.DATA
buffer db 100 dup('$')      ; 缓冲区
msg    db 'Input: $'
newline db 13, 10, '$'

.CODE
main:
    mov ax, @data
    mov ds, ax

    ; 显示提示
    mov ah, 09h
    mov dx, offset msg
    int 21h

    ; 读取一行字符串
    mov ah, 0Ah
    lea dx, buffer
    int 21h

    ; 换行
    mov ah, 09h
    mov dx, offset newline
    int 21h

    ; 输出转换后的字符串
    lea si, buffer+2         ; 第一个字符偏移（跳过长度信息）
next_char:
    mov al, [si]
    cmp al, 13              ; 回车结束
    je done

    cmp al, 'a'
    jb print
    cmp al, 'z'
    ja print
    sub al, 32              ; 转大写

print:
    mov dl, al
    mov ah, 02h
    int 21h

    inc si
    jmp next_char

done:
    mov ah, 4Ch
    int 21h

END main
```

Part 3: Bench mark your computer, MASM speed test.

Enter the following program into your computer and then compile it using MASM within DOSBox as before.

```
          .MODEL medium
          .STACK
          .DATA
          .CODE

          .STARTUP
          mov ah,02         ;Print S
          mov dl,'S'
          int 021h

          mov   bx, 65000   ;4
back2:    mov   cx, 64000   ;4
back1:    nop               ;3
          loop  back1       ;17^or5v

          dec   bx          ;2
          jnz   back2       ;16^or4v

          mov ah,02         ;Print F
          mov dl,'F'
          int 021h

          .EXIT
          END
```

Treat inner loop as a single instruction that takes C_I clock cycles to complete.

$$C_I = C_{Start} + N(C_{Back}) - \left(J_{Back} - J_{Forward}\right)$$

$$C_I = 4 + 64{,}000(17 + 3) - \left(17 - 5\right)$$

$$C_I = ?$$

Calculate total number of clock cycles required to complete code.

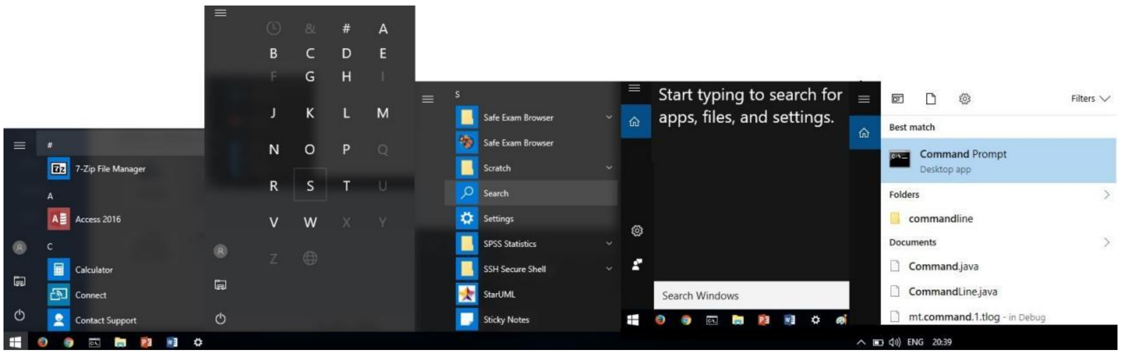
$$C_{Total} = C_{Start} + N(C_{Back}) - \left(J_{Back} - J_{Forward}\right)$$

$$C_{Total} = 4 + 65{,}000(16 + 2 + C_I) - \left(16 - 4\right)$$

$$C_{Total} = ?$$

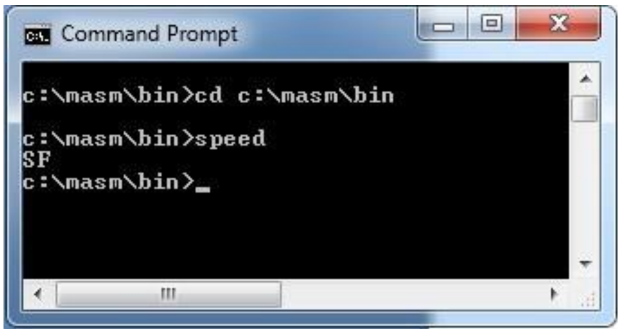
However this time we wish to run the code created on the Windows machine rather than the emulator (DosBox), which is slow and not representative of the machines true speed. Open a command prompt by clicking on the Windows start button then enter “command” into “Search program and files” and press return (or All program –

Accessories – Command prompt). On lab machines just click on command prompt (black rectangle icon) on start bar.



Click on Window button, then the #, then S, select Search, enter “Command”, then run Command prompt, also good to find (MS_Paint and the Control panel etc).

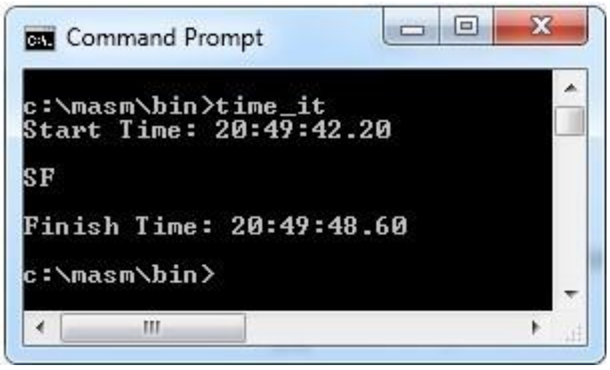
Change the path to the MASM bin folder and run the program by entering *speed* and pressing return.



```
c:\masm\bin>cd c:\masm\bin

c:\masm\bin>speed
SF
c:\masm\bin>_
```

Measure the time between the "S" (for Start) and "F" (for Finish) appearing on screen, use a stopwatch. If you want more precision, you should create the following batch file in Notepad++. The batch can be run by typing its name at the command prompt; it displays the time before and after the program runs; call the file *time_it.bat* when you save it. The batch file should be in same folder as *speed.exe*.

```
@echo off
echo Start Time: %time%
echo.
speed.exe
echo.
echo.
echo Finish Time: %time%
```



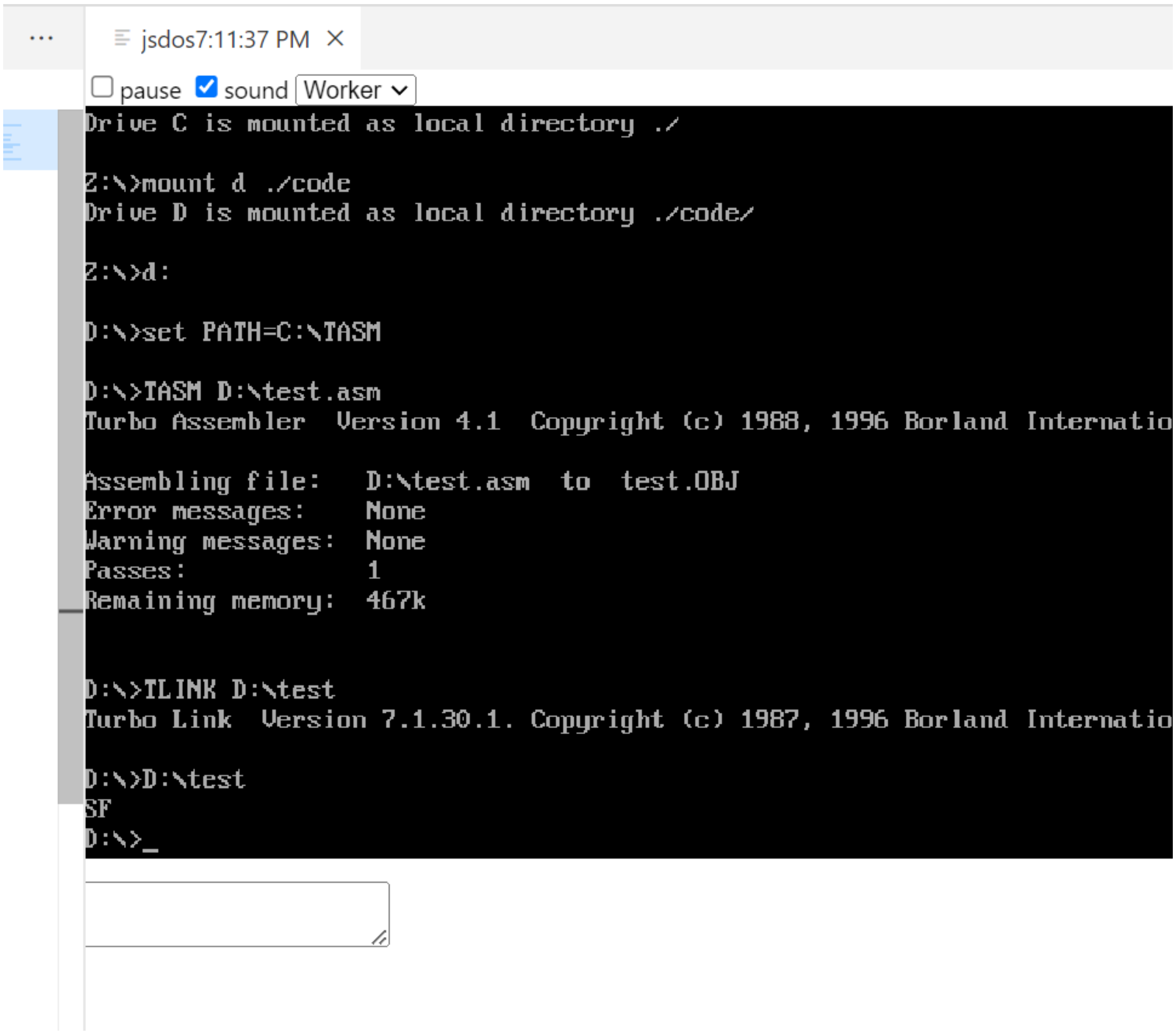
```
c:\masm\bin>time_it
Start Time: 20:49:42.20

SF

Finish Time: 20:49:48.60

c:\masm\bin>
```

Repeat your timing measurement about 5 times and calculate the average run time in seconds, T_{Avg}. Are the times consistent?

$$CPU\_Clock\_rate = \frac{C_{Total}}{T_{Avg}}$$

Note the computer in the laboratory contains an i3-4160 processor running at 3.6GHz but don't expect your answer to be the same.

To record your calculation of the speed of your PC answer the MASM speed test quiz



```
Drive C is mounted as local directory ./

Z:\>mount d ./code
Drive D is mounted as local directory ./code/

Z:\>d:

D:\>set PATH=C:\TASM

D:\>TASM D:\test.asm
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996 Borland Internatio

Assembling file:   D:\test.asm  to  test.OBJ
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  467k


D:\>TLINK D:\test
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland Internatio

D:\>D:\test
SF
D:\>_
```

```
.MODEL medium
.STACK
.DATA
.CODE
.STARTUP

    mov ah, 02h       ; 打印 'S' 表示开始
    mov dl, 'S'
    int 21h

    mov bx, 6500     ; 外循环次数
back2:
    mov cx, 6400    ; 内循环次数
back1:
    nop               ; 空操作, 占用时间
    loop back1        ; 内循环控制

    dec bx            ; 外循环控制
    jnz back2

    mov ah, 02h       ; 打印 'F' 表示结束
    mov dl, 'F'
    int 21h

.EXIT
END
```

$CPU\_Clock\_rate = 30\text{Mhz}$

Time : 27s

Part 4: Complete the general quiz.