Name: Waleed Akram

Roll no: 20p-0640

Section: 3B

Report#10

Computer Organizational & assembly Language

**Q#7:**

ASCII Codes:

The computer listens, sees, and speaks in numbers. Even a character is a  
number inside the computer. For example the keyboard is labeled with  
characters however when we press ‘A’, a specific number is transferred from  
the keyboard to the computer. Our program interprets that number as the  
character ‘A’. When the same number comes on display, the Video Graphics  
Adapter (VGA) in our computer shows the shape of ‘A’. Even the shape is  
stored in binary numbers with a one bit representing a pixel on the screen  
that is turned on and a zero bit representing a pixel that is not glowing. This  
example is considering a white on black display and no colors. This is the  
way a shape is drawn on the screen. The interpretation of ‘A’ is performed by  
the VGA card, while the monitor or CRT (cathode ray tube) only glows the  
pixels on and turns them off. The keyboard has a key labeled ‘A’ and  
pressing it the screen shows ‘A’ but all that happened inside was in  
numbers.  
An ‘A’ on any computer and any operating system is an ‘A’ on every other  
computer and operating system. This is because a standard numeric  
representation of all commonly used characters has been developed. This is  
called the ASCII code, where ASCII stands for American Standard Code for  
Information Interchange. The name depicts that this is a code that allows the  
interchange of information; ‘A’ written on one computer will remain an ‘A’ on  
another. The ASCII table lists all defined characters and symbols and their  
standardized numbers. All ASCII based computers use the same code. There  
are few other standards like EBCDIC and gray codes, but ASCII has become  
the most prevalent standard and is used for Internet communication as well.  
It has become the de facto standard for global communication. The character  
mode displays of our computer use the ASCII standard. Some newer  
operating systems use a new standard Unicode but it is not relevant to us in  
the current discussion.  
Standard ASCII has 128 characters with numbers assigned from 0 to 127.  
When IBM PC was introduced, they extended the standard ASCII and defined  
128 more characters. Thus extending the total number of symbols from 128  
to 256 numbered from 0 to 255 fitting in an 8-bit byte. The newer characters  
were used for line drawing, window corners, and some non-English  
characters. The need for these characters was never felt on teletype  
terminals, but with the advent of IBM PC and its full screen display, these  
semi-graphics characters were the need of the day. Keep in mind that at that  
time there was no graphics mode available.  
The extended ASCII code is just a de facto industry standard but it is not  
defined by an organization like the standard ASCII. Printers, displays, and all  
other peripherals related to the IBM PC understand the ASCII code. If the  
code for ‘A’ is sent to the printer, the printer will print the shape of ‘A’, if it is  
sent to the display, the VGA card will form the shape of ‘A’ on the CRT. If it is  
sent to another computer via the serial port, the other computer will  
understand that this is an ‘A’.  
The important thing to observe in the ASCII table is the contiguous  
arrangement of the uppercase alphabets (41-5A), the lowercase alphabets  
(61-7A), and the numbers (30-39). This helps in certain operations with  
ASCII, for example converting the case of characters by adding or subtracting  
0x20 from it. It also helps in converting a digit into its ASCII representation  
by adding 0x30 to it.

**Q#8 DISPLAY MEMORY FORMATION**We will explore the working of the display with ASCII codes, since it is our  
immediately accessible hardware. When 0x40 is sent to the VGA card, it will  
turn pixels on and off in such a way that a visual representation of ‘A’  
appears on the screen. It has no reality, just an interpretation. In later  
chapters we will program the VGA controller to display a new shape when  
the ASCII of ‘A’ is received by it.  
The video device is seen by the computer as a memory area containing the  
ASCII codes that are currently displayed on the screen and a set of I/O ports  
controlling things like the resolution, the cursor height, and the cursor  
position. The VGA memory is seen by the computer just like its own memory.  
There is no difference; rather the computer doesn’t differentiate, as it is  
accessible on the same bus as the system memory. Therefore if that  
appropriate block of the screen is cleared, the screen will be cleared. If the  
ASCII of ‘A’ is placed somewhere in that block, the shape of ‘A’ will appear on  
the screen at a corresponding place.  
This correspondence must be defined as the memory is a single  
dimensional space while the screen is two dimensional having 80 rows and  
25 columns. The memory is linearly mapped on this two dimensional space,  
just like a two dimensional is mapped in linear memory. There is one word  
per character in which a byte is needed for the ASCII code and the other byte  
is used for the character’s attributes discussed later. Now the first 80 words  
will correspond to the first row of the screen and the next 80 words will  
correspond to the next row. By making the memory on the video controller  
accessible to the processor via the system bus, the processor is now in  
control of what is displayed on the screen.  
The three important things that we discussed are.  
• One screen location corresponds to a word in the video memory  
• The video controller memory is accessible to the processor like its  
own memory.  
• ASCII code of a character placed at a cell in the VGA memory will  
cause the corresponding ASCII shape to be displayed on the  
corresponding screen location.  
**Display Memory Base Address**The memory at which the video controller’s memory is mapped must be a  
standard, so that the program can be written in a video card independent  
manner. Otherwise if different vendors map their video memory at different  
places in the address space, as was the problem in the start, writing software  
was a headache. BIOS vendors had a problem of dealing with various card  
vendors. The IBM PC text mode color display is now fixed so that system  
software can work uniformly. It was fixed at the physical memory location of  
B8000. The first byte at this location contains the ASCII for the character  
displayed at the top left of the video screen. Dropping the zero we can load  
the rest in a segment register to access the video memory. If we do something  
in this memory, the effect can be seen on the screen. For example we can  
write a virus that makes any character we write drop to the bottom of the  
screen.  
**Attribute Byte**The second byte in the word designated for one screen location holds the  
foreground and background colors for the character. This is called its video  
attribute. So the pair of the ASCII code in one byte and the attribute in the  
second byte makes the word that corresponds to one location on the screen.  
The lower address contains the code while the higher one contains the  
attribute. The attribute byte as detailed below has the RGB for the  
foreground and the background. It has an intensity bit for the foreground  
color as well thus making 16 possible colors of the foreground and 8 possible  
colors for the background. When bit 7 is set the character keeps on blinking  
on the screen. This bit has some more interpretations like background  
intensity that has to be activated in the video controller through its I/O  
ports.  
7 – Blinking of foreground character  
6 – Red component of background color  
5 – Green component of background color  
4 – Blue component of background color  
3 – Intensity component of foreground color  
2 – Red component of foreground color  
1 – Green component of foreground color  
0 – Blue component of foreground color  
**Display Examples**Both DS and ES can be used to access the video memory. However we  
commonly keep DS for accessing our data, and load ES with the segment of  
video memory. Loading a segment register with an immediate operand is not  
allowed in the 8088 architecture. We therefore load the segment register via a  
general purpose register. Other methods are loading from a memory location  
and a combination of push and pop.  
mov ax, 0xb800  
mov es, ax  
This operation has opened a window to the video memory. Now the  
following instruction will print an ‘A’ on the top left of the screen in white  
color on black background.  
mov word [es:0], 0x0741  
The segment override is used since ES is pointing to the video memory.  
Since the first word is written to, the character will appear at the top left of  
the screen. The 41 that goes in the lower byte is the ASCII code for ‘A’. The  
07 that goes in the higher byte is the attribute with I=0, R=1, G=1, B=1 for  
the foreground, meaning white color in low intensity and R=0, G=0, B=0 for  
the background meaning black color and the most significant bit cleared so  
that there is no blinking. Now consider the following instruction.  
mov word [es:160], 0x1230  
This is displayed 80 words after the start and there are 80 characters in  
one screen row. Therefore this is displayed on the first column of the second  
line. The ASCII code used is 30, which represents a ‘0’ while the attribute  
byte is 12 meaning green color on blue background.  
We take our first example to clear the screen.

; clear the screen

[org 0x0100]

mov ax, 0xb800 ; load video base in ax

mov es, ax ; point es to video base

mov di, 0 ; point di to top left column

nextchar: mov word [es:di], 0x0720 ; clear next char on screen

add di, 2 ; move to next screen location

cmp di, 4000 ; has the whole screen cleared

jne nextchar ; if no clear next position

mov ax, 0x4c00 ; terminate program

int 0x21

1 ; clear the screen

2 [org 0x0100]

3 00000000 B800B8 mov ax, 0xb800 ; load video base in ax

4 00000003 8EC0 mov es, ax ; point es to video base

5 00000005 BF0000 mov di, 0 ; point di to top left column

6 00000008 26C7052007 nextchar: mov word [es:di], 0x0720 ; clear next char on screen

7 0000000D 81C70200 add di, 2 ; move to next screen location

8 00000011 81FFA00F cmp di, 4000 ; has the whole screen cleared

9 00000015 75F1 jne nextchar ; if no clear next position

10 00000017 B8004C mov ax, 0x4c00 ; terminate program

11 0000001A CD21 int 0x21

**Q#9:**

Hello World in Assembly Language:

To declare a character in assembly language, we store its ASCII code in a  
byte. The assembler provides us with another syntax that doesn’t forces us to  
remember the ASCII code. The assembler also provides a syntax that  
simplifies declaration of consecutive characters, usually called a string. The  
three ways used below are identical in their meaning.  
db 0x61, 0x62, 0x63  
db 'a', 'b', 'c'  
db 'abc'  
When characters are stored in any high level or low level language the  
actual thing stored in a byte is their ASCII code. The only thing the language  
helps in is a simplified declaration.  
Traditionally the first program in higher level languages is to print “hello  
world” on the screen. However due to the highly granular nature of assembly  
language, we are only now able to write it in assembly language. In writing  
this program, we make a generic routine that can print any string on the  
screen.

[org 0x100]

jmp start

    message: db 'hello world' ; string to be printed

    length: dw 11 ; length of the string

; subroutine to clear the screen

    clrscr:

push es

push ax

push di

mov ax, 0xb800

mov es, ax ; point es to video base

mov di, 0 ; point di to top left column

    nextloc:

mov word [es:di], 0x0720 ; clear next char on screen

add di, 2 ; move to next screen location

cmp di, 4000 ; has the whole screen cleared

jne nextloc ; if no clear next position

pop di

pop ax

pop es

ret

; subroutine to print a string at top left of screen

; takes address of string and its length as parameters

printstr:

push bp

mov bp, sp

push es

push ax

push cx

push si

push di

mov ax, 0xb800

mov es, ax ; point es to video base

mov di, 0 ; point di to top left column

mov si, [bp+6] ; point si to string

mov cx, [bp+4] ; load length of string in cx

mov ah, 0x07 ; normal attribute fixed in al

nextchar:

mov al, [si] ; load next char of string

mov [es:di], ax ; show this char on screen

add di, 2 ; move to next screen location

add si, 1 ; move to next char in string

loop nextchar ; repeat the operation cx times

pop di

pop si

pop cx

pop ax

pop es

pop bp

ret 4

start:

call clrscr ; call the clrscr subroutine

mov ax, message

push ax ; push address of message

push word [length] ; push message length

call printstr ; call the printstr subroutine

mov ax, 0x4c00 ; terminate program

int 0x21

1 [org 0x100]

2 00000000 E95B00 jmp start

3 00000003 68656C6C6F20776F72- message: db 'hello world' ; string to be printed

4 0000000C 6C64

5 0000000E 0B00 length: dw 11 ; length of the string

6 ; subroutine to clear the screen

7 clrscr:

8 00000010 06 push es

9 00000011 50 push ax

10 00000012 57 push di

11 00000013 B800B8 mov ax, 0xb800

12 00000016 8EC0 mov es, ax ; point es to video base

13 00000018 BF0000 mov di, 0 ; point di to top left column

14 nextloc:

15 0000001B 26C7052007 mov word [es:di], 0x0720 ; clear next char on screen

16 00000020 81C70200 add di, 2 ; move to next screen location

17 00000024 81FFA00F cmp di, 4000 ; has the whole screen cleared

18 00000028 75F1 jne nextloc ; if no clear next position

19 0000002A 5F pop di

20 0000002B 58 pop ax

21 0000002C 07 pop es

22 0000002D C3 ret

23 ; subroutine to print a string at top left of screen

24 ; takes address of string and its length as parameters

25 printstr:

26 0000002E 55 push bp

27 0000002F 89E5 mov bp, sp

28 00000031 06 push es

29 00000032 50 push ax

30 00000033 51 push cx

31 00000034 56 push si

32 00000035 57 push di

33 00000036 B800B8 mov ax, 0xb800

34 00000039 8EC0 mov es, ax ; point es to video base

35 0000003B BF0000 mov di, 0 ; point di to top left column

36 0000003E 8B7606 mov si, [bp+6] ; point si to string

37 00000041 8B4E04 mov cx, [bp+4] ; load length of string in cx

38 00000044 B407 mov ah, 0x07 ; normal attribute fixed in al

39 nextchar:

40 00000046 8A04 mov al, [si] ; load next char of string

41 00000048 268905 mov [es:di], ax ; show this char on screen

42 0000004B 81C70200 add di, 2 ; move to next screen location

43 0000004F 81C60100 add si, 1 ; move to next char in string

44 00000053 E2F1 loop nextchar ; repeat the operation cx times

45 00000055 5F pop di

46 00000056 5E pop si

47 00000057 59 pop cx

48 00000058 58 pop ax

49 00000059 07 pop es

50 0000005A 5D pop bp

51 0000005B C20400 ret 4

52 start:

53 0000005E E8AFFF call clrscr ; call the clrscr subroutine

54 00000061 B8[0300] mov ax, message

55 00000064 50 push ax ; push address of message

56 00000065 FF36[0E00] push word [length] ; push message length

57 00000069 E8C2FF call printstr ; call the printstr subroutine

58 0000006C B8004C mov ax, 0x4c00 ; terminate program

59 0000006F CD21 int 0x21

**Q#10:**

Number Printing in Assembly Language:

Another problem related to the display is printing numbers. Every high  
level language allows some simple way to print numbers on the screen. As we  
have seen, everything on the screen is a pair of ASCII code and its attribute  
and a number is a raw binary number and not a collection of ASCII codes.  
For example a 10 is stored as a 10 and not as the ASCII code of 1 followed by  
the ASCII code of 0. If this 10 is stored in a screen location, the output will  
be meaningless, as the character associate to ASCII code 10 will be shown on  
the screen. So there is a process that converts a number in its ASCII  
representation. This process works for any number in any base. We will  
discuss our examples with respect to the decimal base and later observe the  
effect of changing to different bases.  
**Number Printing Algorithm**The key idea is to divide the number by the base number, 10 in the case of  
decimal. The remainder can be from 0-9 and is the right most digit of the  
original number. The remaining digits fall in the quotient. The remainder can  
be easily converted into its ASCII equivalent and printed on the screen. The  
other digits can be printed in a similar manner by dividing the quotient again  
by 10 to separate the next digit and so on.  
However the problem with this approach is that the first digit printed is the  
right most one. For example 253 will be printed as 352. The remainder after  
first division was 3, after second division was 5 and after the third division  
was 2. We have to somehow correct the order so that the actual number 253  
is displayed, and the trick is to use the stack since the stack is a Last In  
First Out structure so if 3, 5, and 2 are pushed on it, 2, 5, and 3 will come  
out in this order. The steps of our algorithm are outlined below.  
• Divide the number by base (10 in case of decimal)  
• The remainder is its right most digit  
• Convert the digit to its ASCII representation (Add 0x30 to the  
remainder in case of decimal)  
• Save this digit on stack  
• If the quotient is non-zero repeat the whole process to get the next  
digit, otherwise stop  
• Pop digits one by one and print on screen left to right  
**DIV Instruction**The division used in the process is integer division and not floating point  
division. Integer division gives an integer quotient and an integer remainder.  
A division algorithm is now needed. Fortunately or unfortunately there is a

|  |  |
| --- | --- |
| Computer Architecture & Assembly Language Programming | Course Code: CS401 |
| CS401@vu.edu.pk |  |

Virtual University of Pakistan 77  
DIV instruction available in the 8088 processor. There are two forms of the  
DIV instruction. The first form divides a 32bit number in DX:AX by its 16bit  
operand and stores the 16bit quotient in AX and the 16bit remainder in DX.  
The second form divides a 16bit number in AX by its 8bit operand and stores  
the 8bit quotient in AL and the 8bit remainder in AH. For example “DIV BL”  
has an 8bit operand, so the implied dividend is 16bit and is stored in the AX  
register and “DIV BX” has a 16bit operand, so the implied dividend is 32bit  
and is therefore stored in the concatenation of the DX and AX registers. The  
higher word is stored in DX and the lower word in AX.  
If a large number is divided by a very small number it is possible that the  
quotient is larger than the space provided for it in the implied destination. In  
this case an interrupt is automatically generated and the program is usually  
terminated as a result. This is called a divide overflow error; just like the  
calculator shows an –E– when the result cannot be displayed. This interrupt  
will be discussed later in the discussion of interrupts.  
DIV (divide) performs an unsigned division of the accumulator (and its  
extension) by the source operand. If the source operand is a byte, it is  
divided into the two-byte dividend assumed to be in registers AL and AH. The  
byte quotient is returned in AL, and the byte remainder is returned in AH. If  
the source operand is a word, it is divided into the two-word dividend in  
registers AX and DX. The word quotient is returned in AX, and the word  
remainder is returned in DX. If the quotient exceeds the capacity of its  
destination register (FF for byte source, FFFF for word source), as when  
division by zero is attempted, a type 0 interrupt is generated, and the  
quotient and remainder are undefined.

[org 0x0100]

jmp start

; subroutine to clear the screen

    clrscr:

        push es

        push ax

        push di

        mov ax, 0xb800

        mov es, ax ; point es to video base

        mov di, 0 ; point di to top left column

    nextloc:

        mov word [es:di], 0x0720 ; clear next char on screen

        add di, 2 ; move to next screen location

        cmp di, 4000 ; has the whole screen cleared

        jne nextloc ; if no clear next position

        pop di

        pop ax

        pop es

        ret

; subroutine to print a number at top left of screen

; takes the number to be printed as its parameter

    printnum:

        push bp

        mov bp, sp

        push es

        push ax

        push bx

        push cx

        push dx

        push di

        mov ax, 0xb800

        mov es, ax ; point es to video base

        mov ax, [bp+4] ; load number in ax

        mov bx, 10 ; use base 10 for division

        mov cx, 0 ; initialize count of digits

    nextdigit:

        mov dx, 0 ; zero upper half of dividend

        div bx ; divide by 10

        add dl, 0x30 ; convert digit into ascii value

        push dx ; save ascii value on stack

        inc cx ; increment count of values

        cmp ax, 0 ; is the quotient zero

        jnz nextdigit ; if no divide it again

        mov di, 0 ; point di to top left columnComputer Architecture & Assembly Language Programming Course Code

    nextpos:

        pop dx ; remove a digit from the stack

        mov dh, 0x07 ; use normal attribute

        mov [es:di], dx ; print char on screen

        add di, 2 ; move to next screen location

        loop nextpos ; repeat for all digits on stack

        pop di

        pop dx

        pop cx

        pop bx

        pop ax

        pop es

        pop bp

        ret 2

    start:

        call clrscr ; call the clrscr subroutine

        mov ax, 98763

        push ax ; place number on stack

        call printnum ; call the printnum subroutine

        mov ax, 0x4c00 ; terminate program

        int 0x21

1 ; number printing algorithm

2 [org 0x0100]

3 00000000 E95D00 jmp start

4 ;;;;; COPY LINES 008-025 FROM EXAMPLE 6.2 (clrscr) ;;;;;

5 ; subroutine to clear the screen

6 00000003 06 clrscr: push es

7 00000004 50 push ax

8 00000005 57 push di

9 00000006 B800B8 mov ax, 0xb800

10 00000009 8EC0 mov es, ax ; point es to video base

11 0000000B BF0000 mov di, 0 ; point di to top left column

12 0000000E 26C7052007 nextloc: mov word [es:di], 0x0720 ; clear next char on screen

13 00000013 81C70200 add di, 2 ; move to next screen location

14 00000017 81FFA00F cmp di, 4000 ; has the whole screen cleared

15 0000001B 75F1 jne nextloc ; if no clear next position

16 0000001D 5F pop di

17 0000001E 58 pop ax

18 0000001F 07 pop es

19 00000020 C3 ret

20 ; subroutine to print a number at top left of screen

21 ; takes the number to be printed as its parameter

22 00000021 55 printnum: push bp

23 00000022 89E5 mov bp, sp

24 00000024 06 push es

25 00000025 50 push ax

26 00000026 53 push bx

27 00000027 51 push cx

28 00000028 52 push dx

29 00000029 57 push di

30 0000002A B800B8 mov ax, 0xb800

31 0000002D 8EC0 mov es, ax ; point es to video base

32 0000002F 8B4604 mov ax, [bp+4] ; load number in ax

33 00000032 BB0A00 mov bx, 10 ; use base 10 for division

34 00000035 B90000 mov cx, 0 ; initialize count of digits

35 00000038 BA0000 nextdigit: mov dx, 0 ; zero upper half of dividend

36 0000003B F7F3 div bx ; divide by 10

37 0000003D 80C230 add dl, 0x30 ; convert digit into ascii value

38 00000040 52 push dx ; save ascii value on stack

39 00000041 41 inc cx ; increment count of values

40 00000042 3D0000 cmp ax, 0 ; is the quotient zero

41 00000045 75F1 jnz nextdigit ; if no divide it again

42 00000047 BF0000 mov di, 0 ; point di to top left columnComputer Architecture & Assembly Language Programming Course Code

43 0000004A 5A nextpos: pop dx ; remove a digit from the stack

44 0000004B B607 mov dh, 0x07 ; use normal attribute

45 0000004D 268915 mov [es:di], dx ; print char on screen

46 00000050 81C70200 add di, 2 ; move to next screen location

47 00000054 E2F4 loop nextpos ; repeat for all digits on stack

48 00000056 5F pop di

49 00000057 5A pop dx

50 00000058 59 pop cx

51 00000059 5B pop bx

52 0000005A 58 pop ax

53 0000005B 07 pop es

54 0000005C 5D pop bp

55 0000005D C20200 ret 2

56 00000060 E8A0FF start: call clrscr ; call the clrscr subroutine

57 00000063 B8B111 mov ax, 4529

58 00000066 50 push ax ; place number on stack

59 00000067 E8B7FF call printnum ; call the printnum subroutine

60 0000006A B8004C mov ax, 0x4c00 ; terminate program

61 0000006D CD21 int 0x21

**Q#11:**

Screen Location Calculation:

Until now our algorithms used a fixed attribute and displayed at a fixed  
screen location. We will change that to use any position on the screen and  
any attribute. For mapping from the two dimensional coordinate system of  
the screen to the one dimensional memory, we need to multiply the row  
number by 80 since there are 80 columns per row and add the column  
number to it and again multiply by two since there are 2 bytes for each  
character.  
For this purpose the multiplication routine written previously can be used.  
However we introduce an instruction of the 8088 microprocessor at this time  
that can multiply 8bit or 16bit numbers.  
**MUL Instruction**MUL (multiply) performs an unsigned multiplication of the source operand  
and the accumulator. If the source operand is a byte, then it is multiplied by  
register AL and the double-length result is returned in AH and AL. If the  
source operand is a word, then it is multiplied by register AX, and the  
double-length result is returned in registers DX and AX.  
**String Printing at Desired Location**We modify the string printing program to take the x-position, the yposition, and the attribute as parameters. The desired location on the screen  
can be calculated with the following formulae.  
location = ( hypos \* 80 + epos ) \* 2

[org 0x0100]

jmp start

    message:            db 'hello world' ; string to be printed

    length:     dw 11 ; length of the string

; subroutine to clear the screen

    clrscr:

        push es

        push ax

        push di

        mov ax, 0xb800

        mov es, ax ; point es to video base

        mov di, 0 ; point di to top left column

    nextloc:

        mov word [es:di], 0x0720 ; clear next char on screen

        add di, 2 ; move to next screen location

        cmp di, 4000 ; has the whole screen cleared

        jne nextloc ; if no clear next position

        pop di

        pop ax

        pop es

        ret

    printstr:

        push bp

        mov bp, sp

        push es

        push ax

        push cx

        push si

        push di

        mov ax, 0xb800

        mov es, ax ; point es to video base

        mov al, 80 ; load al with columns per row

        MUL byte [bp+10] ; multiply with y position

        add ax, [bp+12] ; add x position

        shl ax, 1 ; turn into byte offset

        mov di,ax ; point di to required location

        mov si, [bp+6] ; point si to string

        mov cx, [bp+4] ; load length of string in cx

        mov ah, [bp+8] ; load attribute in ah

    nextchar:

        mov al, [si] ; load next char of string

        mov [es:di], ax ; show this char on screen

        add di, 2 ; move to next screen locationComputer Architecture & Assembly Language Programming Course Code: CS401

        add si, 1 ; move to next char in string

        loop nextchar ; repeat the operation cx times

        pop di

        pop si

        pop cx

        pop ax

        pop es

        pop bp

        ret 10

    start:

        call clrscr ; call the clrscr subroutine

        mov ax, 30

        push ax ; push x position

        mov ax, 20

        push ax ; push y position

        mov ax, 1 ; blue on black attribute

        push ax ; push attribute

        mov ax, message

        push ax ; push address of message

        push word [length] ; push message length

        call printstr ; call the printstr subroutine

        mov ax, 0x4c00 ; terminate program

        int 0x21

1 ; hello world at desired screen location

2 [org 0x0100]

3 00000000 E96500 jmp start

4 00000003 68656C6C6F20776F72- message: db 'hello world' ; string to be printed

5 0000000C 6C64

6 0000000E 0B00 length: dw 11 ; length of the string

7 ; subroutine to clear the screen

8 clrscr:

9 00000010 06 push es

10 00000011 50 push ax

11 00000012 57 push di

12 00000013 B800B8 mov ax, 0xb800

13 00000016 8EC0 mov es, ax ; point es to video base

14 00000018 BF0000 mov di, 0 ; point di to top left column

15 nextloc:

16 0000001B 26C7052007 mov word [es:di], 0x0720 ; clear next char on screen

17 00000020 81C70200 add di, 2 ; move to next screen location

18 00000024 81FFA00F cmp di, 4000 ; has the whole screen cleared

19 00000028 75F1 jne nextloc ; if no clear next position

20 0000002A 5F pop di

21 0000002B 58 pop ax

22 0000002C 07 pop es

23 0000002D C3 ret

24 printstr:

25 0000002E 55 push bp

26 0000002F 89E5 mov bp, sp

27 00000031 06 push es

28 00000032 50 push ax

29 00000033 51 push cx

30 00000034 56 push si

31 00000035 57 push di

32 00000036 B800B8 mov ax, 0xb800

33 00000039 8EC0 mov es, ax ; point es to video base

34 0000003B B050 mov al, 80 ; load al with columns per row

35 0000003D F6660A MUL byte [bp+10] ; multiply with y position

36 00000040 03460C add ax, [bp+12] ; add x position

37 00000043 D1E0 shl ax, 1 ; turn into byte offset

38 00000045 89C7 mov di,ax ; point di to required location

39 00000047 8B7606 mov si, [bp+6] ; point si to string

40 0000004A 8B4E04 mov cx, [bp+4] ; load length of string in cx

41 0000004D 8A6608 mov ah, [bp+8] ; load attribute in ah

42 nextchar:

43 00000050 8A04 mov al, [si] ; load next char of string

44 00000052 268905 mov [es:di], ax ; show this char on screen

45 00000055 81C70200 add di, 2 ; move to next screen locationComputer Architecture & Assembly Language Programming Course Code: CS401

46 00000059 81C60100 add si, 1 ; move to next char in string

47 0000005D E2F1 loop nextchar ; repeat the operation cx times

48 0000005F 5F pop di

49 00000060 5E pop si

50 00000061 59 pop cx

51 00000062 58 pop ax

52 00000063 07 pop es

53 00000064 5D pop bp

54 00000065 C20A00 ret 10

55 00000068 E8A5FF start: call clrscr ; call the clrscr subroutine

56 0000006B B81E00 mov ax, 30

57 0000006E 50 push ax ; push x position

58 0000006F B81400 mov ax, 20

59 00000072 50 push ax ; push y position

60 00000073 B80100 mov ax, 1 ; blue on black attribute

61 00000076 50 push ax ; push attribute

62 00000077 B8[0300] mov ax, message

63 0000007A 50 push ax ; push address of message

64 0000007B FF36[0E00] push word [length] ; push message length

65 0000007F E8ACFF call printstr ; call the printstr subroutine

66 00000082 B8004C mov ax, 0x4c00 ; terminate program

67 00000085 CD21 int 0x21