Short Notes Chapter 6

[Lecture 16-18 [first-half]]

ASCII:

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.

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 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.

DISPLAY MEMORY FORMATION

The three important things 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.

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**.

The attribute byte 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	6	5	4	3	2	1	0

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

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.**

The following operation opens a window to the video memory.

```
mov ax, 0xb800
mov es, ax
```

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 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.

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.

Character Printing:

To declare a character in assembly language, we store its ASCII code in a byte.

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'

Number Printing Algorithm:

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

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 8-bit operand, so the implied dividend is 16-bit and is stored in the AX register.

"DIV BX" has a 16-bit operand, so the implied dividend is 32-bit 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.

Divide overflow error:

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.

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.

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 y-position, and the attribute as parameters. The desired location on the screen can be calculated with the following formulae.
location = (hypos * 80 + epos) * 2