6502 Program Example 04

Output a String Using Indexed Addressing

Here is a program that is designed to output a string of characters to the terminal screen. As you can see, we have to treat the string as an array of characters, which have to be output one by one. This is an ideal job for the index registers, so we will be making a lot of use of the indexed addressing mode.

This program is doing the same work as the **puts** command in C, so it is safe to assume that when you used **puts** in your C program, the C compiler produced some machine code that was fairly similar to this. You may also remember that in C, the end of a string was marked by an invisible string termination character called '\0'. This program uses the same technique.

```
; Program to output a string of characters
; Similar to the C command puts(nam)
; Putssim.65s

       .ORG $0200       ; Store machine code starting here

io_area = $E000
io_putc = io_area + 1

       LDX #$0          ; Initialise X register

la1:   LDA nam, X       ; Load character at base address + offset
       STA io_putc      ; Send to output screen
       INX              ; Increment offset
       CMP #$18         ; Test for string termination character.
       BNE la1          ; If not termination character, branch up to LA1
       BRK

nam:   .DB 'h'          ; This assembler allows me to
       .DB 'e'          ; use characters in quotes
       .DB 'l'          ; which saves me having
       .DB 'l'          ; to look up the ASCII codes for
       .DB 'o'          ; the characters that I
       .DB ' '          ; wish to hold in the string
       .DB 'w'
       .DB 'o'
       .DB 'r'
       .DB 'l'
       .DB 'd'
       .DB $18          ; This is the string termination character
```

Exercises

Step through the program. Keep an eye on the contents of the A and the X registers – and on the i/o screen.

Using $18 as a terminator character isn't a good idea – it shows up on the screen. Try to rearrange the program so that you can still use $18, but exit from the loop before it gets displayed.

Rewrite the program using $00 as a terminator character instead. This is what C does. Not only does it not show up on the screen, it enables you to do without the CMP instruction. Why?

Before each character code is output, lets say that we want to convert it into upper case.

The first thing that we need to do is look at the way ASCII codes were designed.

A	01000001	$41	Z	01011010	$5A
a	01100001	$61	z	01111010	$7A

You can see that the codes for a letter and it's lower case equivalent differ by only one bit, usually called bit 5 if we adhere to the convention of counting from the right hand end of the byte with the least significant bit numbered bit 0.

If bit 5 is cleared (0) then it is an upper case letter, and if it is set (1) it is lower case. This is not an accident – ASCII codes were designed with this sort of manipulation in mind.

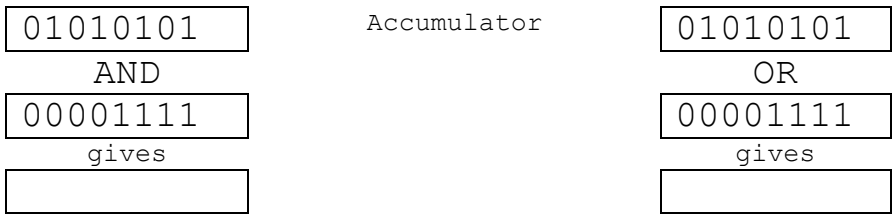
Exercise

The easiest way is to simply subtract $20 from each character code before it is output. Build this into your loop and see if it works. (Don't forget to set the carry flag first)

This would be ok, except that if the character is already upper case, subtracting $20 from it will mess things up. There is another method which will set bit 5, but it won't change the value of the rest of the character code.

It involves using one of the Logical Group instructions.

If the accumulator contains the code 01010101, what effect does doing a bit by bit AND operation with the code 00001111 have on the accumulator. What about the OR operation? Write in your answers.



Exercise

Using one of the logical operations in a certain way will enable you to set bit 5, without the danger of changing the code if it is already a 0. See if you can work it out.