6502 Program Example 05

Use of Subroutines and the Stack

Here is a program that is designed to display the following pattern of characters in the terminal window:

```
X++++X++++X++++X++++X++++
```

The X characters are output in the main body of the program, using a loop which is controlled by the contents of the Y register. The + characters are output by a subroutine, which is called from inside the loop of the main program.

```
; Program to Demonstrate Use of a Subroutine (and the stack)
; Subroutine3.65s

       .ORG $0220

io_area = $E000
io_putc = io_area + 1

       LDA #'X'       ; Initialise acc with first print character
       LDY #$00       ; Initialise Y index reg (for a change) with count

lop:   STA io_putc    ; Output print character

       JSR star       ; Jump to the subroutine called star

       INY            ; Increment count
       CPY #$05       ; Have we output 5 X's yet?
       BNE lop        ; If not, jump up to top of loop.

       BRK            ; End of program.



; Start of subroutine section

star:  PHA            ; Push first print char onto stack for safekeeping

       LDA #'+'       ; Load acc with second print character
       STA io_putc    ; Output it 4 times
       STA io_putc
       STA io_putc
       STA io_putc

       PLA            ; Pop first print character off stack back into acc

       RTS            ; Return to the instruction immediately following
                      ; the subroutine call (JSR) in the main program.
```

When high level programs are divided up into functions or procedures, this structure is usually reflected in the low level code that is generated when they are compiled.

We need to understand what the JSR and RTS instructions are doing.

JSR aaaa Jump to Subroutine

This will cause control to be passed to a different part of the program. In other words, the address of the instruction being jumped to will be placed in the program counter register. Before we can do that, the processor needs to save the address of the current instruction, so that it knows where to jump back to, after the subroutine has finished executing. The section of the memory known as the stack is where the processor stores the return address.

When a JSR instruction is executed, the following steps are carried out:

1. The High End half of the address of the last byte of the JSR instruction is pushed onto the stack.
2. The stack pointer is decremented, so that it now points to the next free space.
3. The Low End half of the address of the last byte of the JSR instruction is pushed onto the stack.
4. The stack pointer is decremented, so that it now points to the next free space.
5. The address aaaa specified in the JSR instruction is loaded into the program counter register.

The instruction at location aaaa will therefore be the next one to be fetched and executed.

RTS Return from Subroutine

This will cause control to be passed back to the main program, to the instruction immediately after the JSR instruction (the subroutine call). This means that the return address has to be retrieved from the stack.

When a RTS instruction is executed, the following steps are carried out:

1. The stack pointer is incremented.
2. The Low End of the return address is loaded into the low end of the program counter register.
3. The stack pointer is incremented.
4. The High End of the return address is loaded into the high end of the program counter register.
5. The contents of the program counter are incremented by one, so that instead of referring to the last byte of the JSR instruction, the program counter now contains the address of the first byte of the next instruction.

The stack is not just used to store return addresses, as you can see. It can also be used to store the values of local variables (such as the contents of the acc in the subroutine above: '+'), or the values of parameters that need to be passed to the subroutine.

Exercises

Trace through the program above. Make sure that you have got the Stack window open, although you can see the values being placed on the stack in the ordinary memory window – remember that the stack starts at location $01FF and grows downwards towards $0100. Also keep an eye on the program counter register during the JSR and RTS instructions.

Add a further subroutine which displays two exclamation marks. It should be jumped to after the second '+' has been output, but before the third.

Add another which displays one equal sign '='. It should be jumped to after the first exclamation mark has been output.

So the final output should be: `X++!=!++X++!=!++X++!=!++X++!=!++X++!=!++`

Trace through it. Watch the contents of the stack and the program counter change as you go into and come out of the subroutines.

The point of all this is to show how the stack, with its first on, last off accessing, is the ideal structure in which to store the return addresses and local variables of nested subroutines. This is how high level functions and procedures are implemented at low level.