Programming Basics

From NESdev Wiki

NES runs on the 6502 microprocessor, an 8-bit microprocessor with a 16-bit address bus. The 6502 has powered systems like Commodore 64, Apple II, Atari 2600 and Nintendo Entertainment System. The 6502 has different mnemonics than what some assembly programmers might be used to and some useful instructions like `mul` (used for multiplication) are not available in the 6502 and hence present the need to program them on our own.

Contents

Stack

Main article: Stack

Stack is a data structure used to store data which is very simple and much faster than a heap-based memory structure. It runs by the principle of **last-in-first-out**, where any new data that is to be entered, is put at the top of the stack (pushing data to the stack) and when removing data from the stack (popping data from the stack), the data that was entered last (the data that is at the top) will be removed first.

Instructions and Opcodes

Instructions

Instructions are actions that the processor performs. The 6502 has 56 of such instruction including instructions for operations such as addition, subtraction, AND, OR, ROR, etc. All the instructions are denoted by a 3-letter mnemonic and are then followed by their operands.

There are some instructions which perform operation on a specific register or need a specific register for its operation, such instructions contain the register mnemonic of the specific register in their instruction mnemonic only. For Example - `LDA` loads a byte of memory into the accumulator register denoted by `A`.

Opcodes

Opcodes (abbreviated from operation codes) are the part of instruction in machine language which specifies the operation to be performed by the processor. Operands are the data on which the operation is performed. The 6502 processor has a total of 256 possible opcodes, but only 151 were used originally, arranged into 56 instructions which the NES used.

Registers

The 6502 processor has six 8-bit registers, with the exception of the Program Counter, which is 16-bit. The registers are as follows:

1. Accumulator(A) - The accumulator can read and write to memory. It is used to store arithmetic and logic results such as addition and subtraction.
2. X Index(X) - The x index is can read and write to memory. It is used primarily as a counter in loops, or for addressing memory, but can also temporarily store data like the accumulator.
3. Y Index(Y) - Much like the x index, however they are not completely interchangeable. Some operations are only available for each register.
4. Flag(P) - The register holds value of 7 different flags which can only have a value of 0 or 1 and hence can be represented in a single register. The bits represent the status of the processor.
5. Stack Pointer(SP) - The stack pointer hold the address to the current location on the Stack (https://www.nesdev.org/wiki/Stack). The stack is a way to store data by pushing or popping data to and from a section of memory.
6. Program Counter(PC) - This is a 16-bit register unlike other registers which are only 8-bit in length, it indicates where the processor is in the program sequence.

Math Operations

The 6502 processor only has instruction for addition and subtraction, it unfortunately doesn't have an instruction for multiplication or division and hence puts the need to implement them on our own.

Simple operations

1. Addition: The 6502 processor has the instruction `ADC` for addition. It adds the value of an 8-bit number to the accumulator along with the carry bit.
2. Subtraction: The `SBC` instruction is used to subtract a value to the accumulator together with the `NOT` of the carry bit.

Complex operations

Multiplication

As multiplication is repeated addition, one can implement a simple loop to add the value of multiplicand (the quantity to be multiplied) to itself times the value of multiplier (the value multiplicand is to be multiplied with). This is a valid approach but a more efficient solution would be to use left shifts and additions which can significantly reduce the number of operations.

The following routine multiplies two unsigned 16-bit numbers, and returns an unsigned 32-bit value.

```
mulplr  = $c0          ; ZP location = $c0
partial = mulplr+2     ; ZP location = $c2
mulcnd  = partial+2    ; ZP location = $c4

_usmul:
  pha
  tya
  pha

_usmul_1:
  ldy #$10     ; Setup for 16-bit multiply
_usmul_2:
  lda mulplr   ; Is low order bit set?
  lsr a
  bcc _usmul_4

  clc          ; Low order bit set -- add mulcnd to partial product
  lda partial
  adc mulcnd
  sta partial
  lda partial+1
  adc mulcnd+1
  sta partial+1
;
; Shift result into mulplr and get the next bit of the multiplier into the low order bit of mulplr.
;
_usmul_4:
  ror partial+1
  ror partial
  ror mulplr+1
  ror mulplr
  dey
  bne _usmul_2
  pla
  tay
  pla
  rts
```

Here's an example of the above `_usmul` routine in action, which multiplies 340*268:

```
  lda #<340    ; Low byte of 16-bit decimal value 340  (value: $54)
  sta mulplr
  lda #>340    ; High byte of 16-bit decimal value 340 (value: $01) (makes $0154)
  sta mulplr+1
  lda #<268    ; Low byte of 16-bit decimal value 268  (value: $0C)
  sta mulcnd
  lda #>268    ; High byte of 16-bit decimal value 268 (value: $01) (makes $010C)
  sta mulcnd+1
  lda #0               ; Must be set to zero (0)!
  sta partial
  sta partial+1
  jsr _usmul   ; Perform multiplication
;
; RESULTS
;    mulplr    = Low byte of lower word  (bits 0 through 7)
;    mulplr+1  = High byte of lower word (bits 8 through 15)
;    partial   = Low byte of upper word  (bits 16 through 23)
;    partial+1 = High byte of upper word (bits 24 through 31)
;
```

Retrieved from "https://www.nesdev.org/w/index.php?title=Programming_Basics&oldid=21421"