pikuma.com

6502 Instruction Set

(Warm-Up Exercises)

The following exercises are designed to introduce you to the most common 6502 instructions that we'll use in the beginning of our course.

I am aware that these exercises will sound a bit pointless, and it might seem like we are simply moving values around the machine. But don't worry; very soon we'll put all these instructions together and create something more meaningful, like asking the PPU to paint pixels on the screen or keeping the score of a player or the number of enemies in our game.

For now, all I want is for you to get familiarized with the basic instructions of the 6502 CPU. These exercises will cover things like different addressing modes, loading values into registers, storing values in memory, checking processor flags, and creating loops.

Don't forget that you must always include the iNES header in all exercises if you want to run and debug them with an emulator. You must also always add the three vectors for NMI, Reset, and IRQ handlers at the end of the PRG-ROM at address $FFFA.

Exercise 1

Your goal here is to simply load the processor registers A, X, and Y with some values.

```
.segment "HEADER"   ; Don't forget to always add the iNES header to your ROM files
.org $7FF0
.byte $4E,$45,$53,$1A,$02,$01,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00

.segment "CODE"     ; Define a segment called "CODE" for the PRG-ROM at $8000
.org $8000

Reset:              ; TODO:
                    ; Load the A register with the literal hexadecimal value $82
                    ; Load the X register with the literal decimal value 82
                    ; Load the Y register with the value that is inside memory position $82

NMI:                ; NMI handler
    rti             ; doesn't do anything

IRQ:                ; IRQ handler
    rti             ; doesn't do anything

.segment "VECTORS"  ; Add addresses with vectors at $FFFA
.org $FFFA
.word NMI           ; Put 2 bytes with the NMI address at memory position $FFFA
.word Reset         ; Put 2 bytes with the break address at memory position $FFFC
.word IRQ           ; Put 2 bytes with the IRQ address at memory position $FFFE
```

Exercise 2

Your goal here is to just store some values into zero-page memory positions.

```
.segment "CODE"
.org $8000

Reset:              ; TODO:
                    ; Load the A register with the hexadecimal value $A
                    ; Load the X register with the binary value %11111111
                    ; Store the value in the A register into memory address $80
                    ; Store the value in the X register into memory address $81
```

Exercise 3

This exercise is about transferring values from registers to other registers.

```
.segment "CODE"
.org $8000

Reset:              ; TODO:
                    ; Load the A register with the literal decimal value 15
                    ; Transfer the value from A to X
                    ; Transfer the value from A to Y
                    ; Transfer the value from X to A
                    ; Transfer the value from Y to A

                    ; Load X with the decimal value 6
                    ; Transfer the value from X to Y
```

Exercise 4

This exercise is about adding and subtracting values. Adding and subtracting are math operations that are done by the processor ALU (arithmetic-logic-unit). Since the ALU can only manipulate values from the *(A)ccumulator*, all these additions and subtractions must be performed with the values in the A register.

```
.segment "CODE"
.org $8000

Reset:              ; TODO:
                    ; Load the A register with the literal decimal value 100

                    ; Add the decimal value 5 to the accumulator
                    ; Subtract the decimal value 10 from the accumulator
                    ; Register A should now contain the decimal 95 (or $5F in hexadecimal)
```

Exercise 5

The ADC and SBC instructions can also be used with different addressing modes. The above exercise used ADC with immediate mode (adding a literal value directly into the accumulator), but we can also ask ADC to add a value from a (zero page) memory position into the accumulator.

```
.segment "CODE"
.org $8000

Reset:
                    ; TODO:

                    ; Load the A register with the hexadecimal value $A
                    ; Load the X register with the binary value %1010

                    ; Store the value in the A register into (zero page) memory address $80
                    ; Store the value in the X register into (zero page) memory address $81

                    ; Load A with the decimal value 10
                    ; Add to A the value inside RAM address $80
                    ; Add to A the value inside RAM address $81
                    ; A should contain (#10 + $A + %1010) = #30 (or $1E in hexadecimal)

                    ; Store the value of A into RAM position $82
```

Exercise 6

This exercise covers the increment and decrement instructions of the 6502.

```
.segment "CODE"
.org $8000

Reset:
                    ; TODO:

                    ; Load the A register with the decimal value 1
                    ; Load the X register with the decimal value 2
                    ; Load the Y register with the decimal value 3

                    ; Increment X
                    ; Increment Y
                    ; Increment A

                    ; Decrement X
                    ; Decrement Y
                    ; Decrement A
```

Exercise 7

This exercise covers the increment and decrement using zero-page addressing mode. The zero-page addressing mode helps us directly increment and decrement values inside memory positions. The "zero page" in the 6502 are addresses between 0 and 255. These addresses are special for the 6502 processor because we can store them using only 1 byte (8 bits), which also means they can be performed relatively fast by the CPU.

```
.segment "CODE"
.org $8000

Reset:               ; TODO:
                     ; Load the A register with the decimal value 10
                     ; Store the value from A into memory position $80

                     ; Increment the value inside a (zero page) memory position $80
                     ; Decrement the value inside a (zero page) memory position $80
```

Exercise 8

Your goal here is to create a loop that counts down from 10 to 0. You should also fill the memory addresses from $80 to $8A with values from 0 to A.

$80	$81	$82	$83	$84	$85	$86	$87	$88	$89	$8A
0	1	2	3	4	5	6	7	8	9	A

```
.segment "CODE"
.org $8000

Reset:
    ldy #10          ; Initialize the Y register with the decimal value 10

Loop:
                     ; TODO:
                     ; Transfer Y to A
                     ; Store the value in A inside memory position $80+Y
                     ; Decrement Y
                     ; Branch back to "Loop" until we are done
```

Exercise 9

Your goal in this exercise is to create a simple loop that goes from 1 to 10. If possible, try using the CMP instruction. This instruction that can be used to compare the value of the accumulator with a certain literal number. Once the comparison is done, the processor flags will be set (zero if the compared values are equal, non-zero if different).

```
.segment "CODE"
.org $8000

Reset:
        lda #1      ; Initialize the A register with 1

Loop:
                    ; TODO:
                    ; Increment A
                    ; Compare the value in A with the decimal value 10
                    ; Branch back to "Loop" if the comparison was not equals (to zero)
```