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 painting pixels on the screen or keeping the score of a player or the number of enemies in our games.

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.

Exercise 1

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

```
    processor 6502
    seg Code         ; Define a new segment named "Code"
    org $F000        ; Define the origin of the ROM code at memory address $F000
Start:
    lda #$82         ; Load the A register with the literal hexadecimal value $82
    ldx #82          ; Load the X register with the literal decimal value 82
    ldy $82          ; Load the Y register with the value that is inside memory position $82

    org $FFFC        ; End the ROM by adding required values to memory position $FFFC
    .word Start      ; Put 2 bytes with the reset address at memory position $FFFC
    .word Start      ; Put 2 bytes with the break address at memory position $FFFE
```

Exercise 2

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

```
    processor 6502
    seg Code         ; Define a new segment named "Code"
    org $F000        ; Define the origin of the ROM code at memory address $F000
Start:
    lda #$A          ; Load the A register with the hexadecimal value $A
    ldx #%11111111   ; Load the X register with the binary value %11111111
    sta $80          ; Store the value in the A register into memory address $80
    stx $81          ; Store the value in the X register into memory address $81

    org $FFFC       ; End the ROM by adding required values to memory position $FFFC
    .word Start     ; Put 2 bytes with the reset address at memory position $FFFC
    .word Start     ; Put 2 bytes with the break address at memory position $FFFE
```

Exercise 3

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

```
    processor 6502
    seg Code         ; Define a new segment named "Code"
    org $F000        ; Define the origin of the ROM code at memory address $F000
Start:
    lda #15          ; Load the A register with the literal decimal value 15

    tax              ; Transfer the value from A to X
    tay              ; Transfer the value from A to Y
    txa              ; Transfer the value from X to A
    tya              ; Transfer the value from Y to A

    ; Important! There's no TXY or TYX, so we can't transfer directly X to Y or Y to X.
    ; If we want to transfer X to Y, we can first transfer X to A and then A to Y

    ldx #6           ; Load X with the decimal value #6
    txa              ; Transfer X to A
    tay              ; Transfer A to Y

    org $FFFC       ; End the ROM by adding required values to memory position $FFFC
    .word Start     ; Put 2 bytes with the reset address at memory position $FFFC
    .word Start     ; Put 2 bytes with the break address at memory position $FFFE
```

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 access the *(A)ccumulator* register, all these additions and subtractions must be performed with the values in the A register.

```
    processor 6502
    seg Code         ; Define a new segment named "Code"
    org $F000        ; Define the origin of the ROM code at memory address $F000

Start:
    cld              ; Remember to disable BCD decimal mode to avoid wrong arithmetic
    lda #100         ; Load the A register with the literal decimal value 100

    clc              ; We always clear the carry flag before addition in the 6502
    adc #5           ; Add (with carry) the decimal  value 5 to the accumulator
                     ; Register A should now contain the decimal value 105

    sec              ; We always set the carry flag before subtraction in the 6502
    sbc #10          ; Subtract (with carry) the decimal value 10 from accumulator
                     ; Register A should now contain the decimal value 95

    org $FFFC       ; End the ROM by adding required values to memory position $FFFC
    .word Start     ; Put 2 bytes with the reset address at memory position $FFFC
    .word Start     ; Put 2 bytes with the break address at memory position $FFFE
```

Exercise 5

ADC and SBC can also be invoked using 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.

```
    processor 6502
    seg Code        ; Define a new segment named "Code"
    org $F000       ; Define the origin of the ROM code at memory address $F000
Start:
    cld             ; Remember to disable BCD decimal mode to avoid wrong arithmetic
    lda #$A         ; Load the A register with the hexadecimal value $A
    ldx #%1010      ; Load the X register with the binary value %1010

    sta $80         ; Store the value in the A register into memory address $80
    stx $81         ; Store the value in the X register into memory address $81

    lda #10         ; Load A with the decimal value 10
    clc             ; Clear the carry flag before ADC
    adc $80         ; Add to A the value inside RAM address $80
    adc $81         ; Add to A the value inside RAM address $81
                    ; A should contain (#10 + $A + %1010) = #30 (or $1E in hexadecimal)
    sta $82         ; Store the value of A into RAM position $82

    org $FFFC      ; End the ROM by adding required values to memory position $FFFC
    .word Start    ; Put 2 bytes with the reset address at memory position $FFFC
    .word Start    ; Put 2 bytes with the break address at memory position $FFFE
```

Exercise 6

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

```
    processor 6502
    seg Code        ; Define a new segment named "Code"
    org $F000       ; Define the origin of the ROM code at memory address $F000
Start:
    lda #1          ; Load the A register with the decimal value 1
    ldx #2          ; Load the X register with the decimal value 2
    ldy #3          ; Load the Y register with the decimal value 3
    inx             ; Increment X
    iny             ; Increment Y
    clc
    adc #1          ; Add 1 to A, since the 6502 has no "inc A" instruction
    dex             ; Decrement X
    dey             ; Decrement Y
    sec
    sbc #1          ; Subtract 1 from A, since the 6502 has no "dec A" instruction

    ; The 6502 can directly increment and decrement X and Y, but not A.
    ; We can only fake an A increment with ADC #1 and fake a decrement with SBC #1.
    ; This makes X and Y good registers to control loops.

    org $FFFC       ; End the ROM always at position $FFFC
    .word Start     ; Put 2 bytes with reset address at memory position $FFFC
    .word Start     ; Put 2 bytes with break address at memory position $FFFE
```

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.

```
    processor 6502
    seg Code        ; Define a new segment named "Code"
    org $F000       ; Define the origin of the ROM code at memory address $F000

Start:
    cld             ; Remember to disable BCD decimal mode to avoid wrong arithmetic
    lda #10         ; Load the A register with the decimal value 10
    sta $80         ; Store the value from A into memory position $80

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

    org $FFFC       ; End the ROM always at position $FFFC
    .word Start     ; Put 2 bytes with reset address at memory position $FFFC
    .word Start     ; Put 2 bytes with break address at memory position $FFFE
```

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

```
    processor 6502
    seg Code        ; Define a new segment named "Code"
    org $F000       ; Define the origin of the ROM code at memory address $F000

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

Loop:
    tya             ; Transfer Y to A
    sta $80,Y       ; Store the value in A inside memory position $80+Y
    dey             ; Decrement Y
    bpl Loop        ; Branch if plus (only if result of last instruction was positive)

    org $FFFC       ; End the ROM always at position $FFFC
    .word Start     ; Put 2 bytes with reset address at memory position $FFFC
    .word Start     ; Put 2 bytes with break address at memory position $FFFE
```

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).

```
    processor 6502
    seg Code        ; Define a new segment named "Code"
    org $F000       ; Define the origin of the ROM code at memory address $F000

Start:
        lda #1      ; Initialize the A register with the decimal value 1

Loop:
    clc
    adc #1          ; Increment A (using ADC #1)
    cmp #10         ; Compare the value in A with the decimal value 10
    bne Loop        ; Branch back to loop if the comparison was not equals (to zero)


    org $FFFC       ; End the ROM always at position $FFFC
    .word Start     ; Put 2 bytes with reset address at memory position $FFFC
    .word Start     ; Put 2 bytes with break address at memory position $FFFE
```