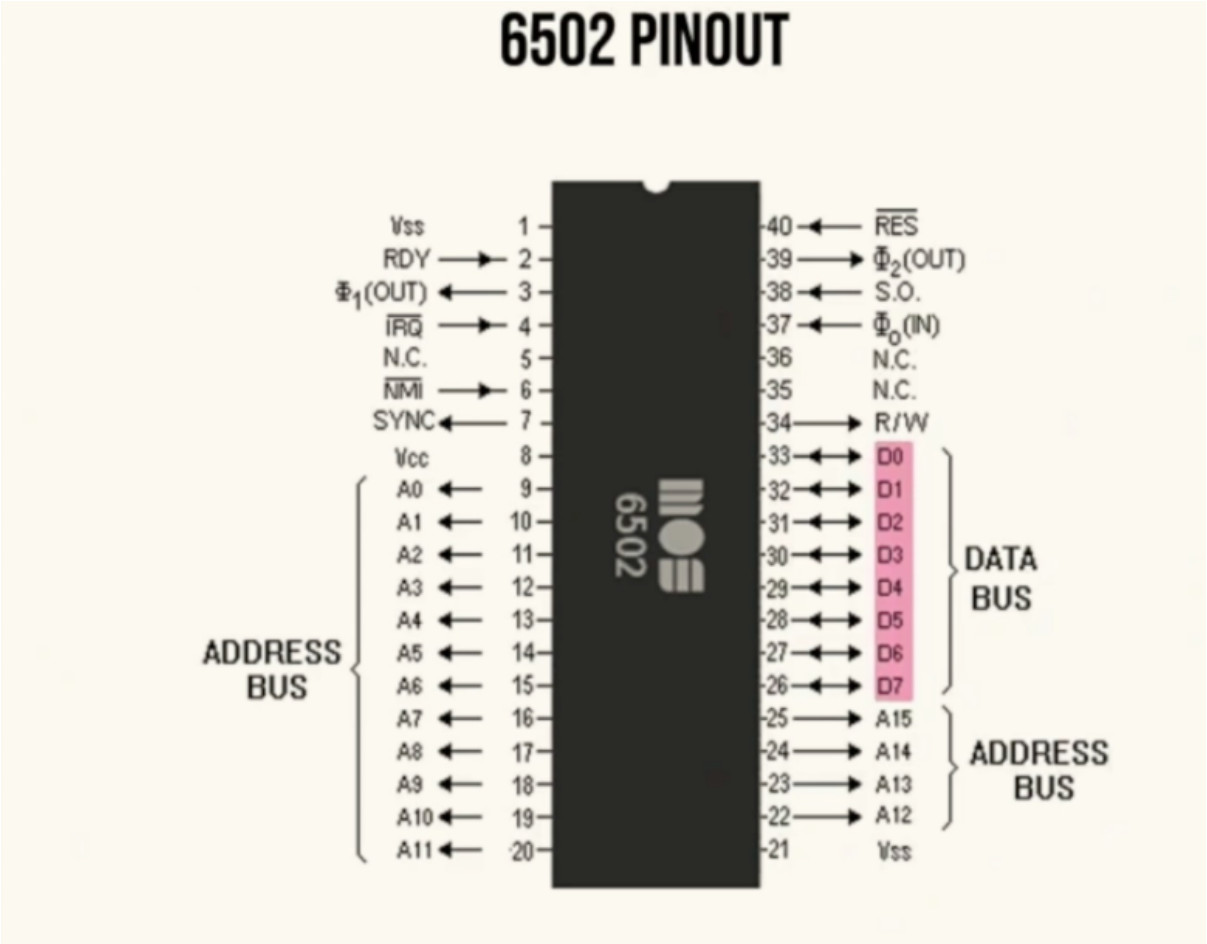
ATTARI 2600

CPU - 1.19MHz(**6507** processor)
Audio/Video - TIA Chip(Television Interface Adapter)
RAM - 128bytes **6532** RIOT chip
ROM - game cartridge- 4kB - Assembly Instruction is stored here
Input - joystick, paddle
Output - TV via RCA connector (NTSC, PAL, SECAM)

6502



6502 is a 8 bit machine because DATA BUS pins are 8 in number. Our machines are 64 bit

- DATA BUS - 8 bits (D0 to D7) sends/receives the data to be handled by the registers
- ADDRESS BUS - 16bits (A0 to A15) - stores value of memory address

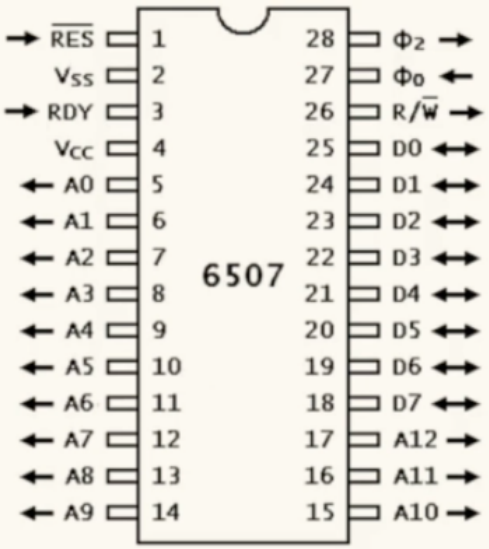
6507

Address pin = 13
Data pin = 8

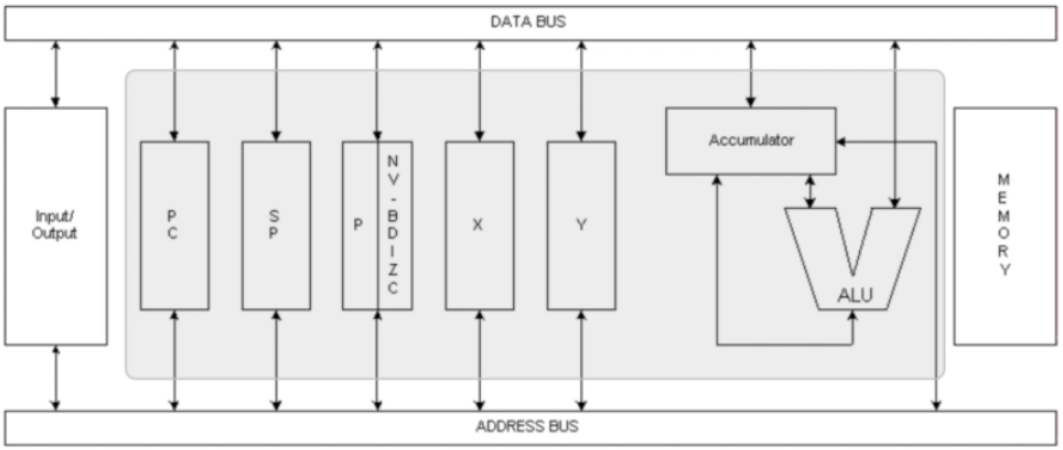
Total = 28 pins
No interrupt, NMI pins

6507

Essentially a 6502 in a cheaper 28-pin package.
A15 to A13 and other interruption lines are not accessible.

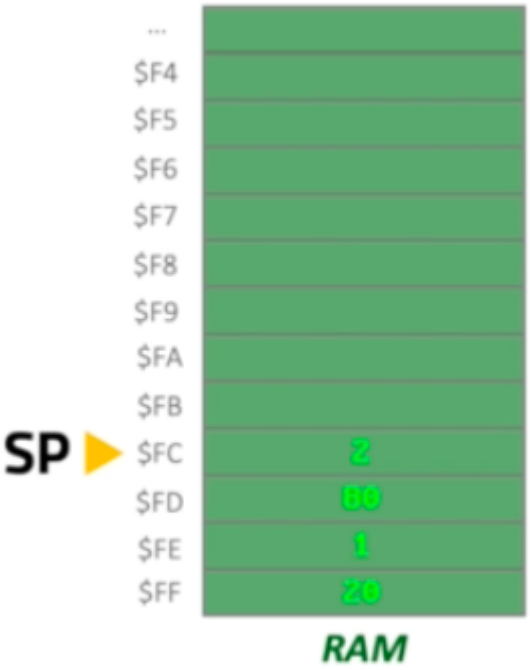


7 main Parts of 6507:

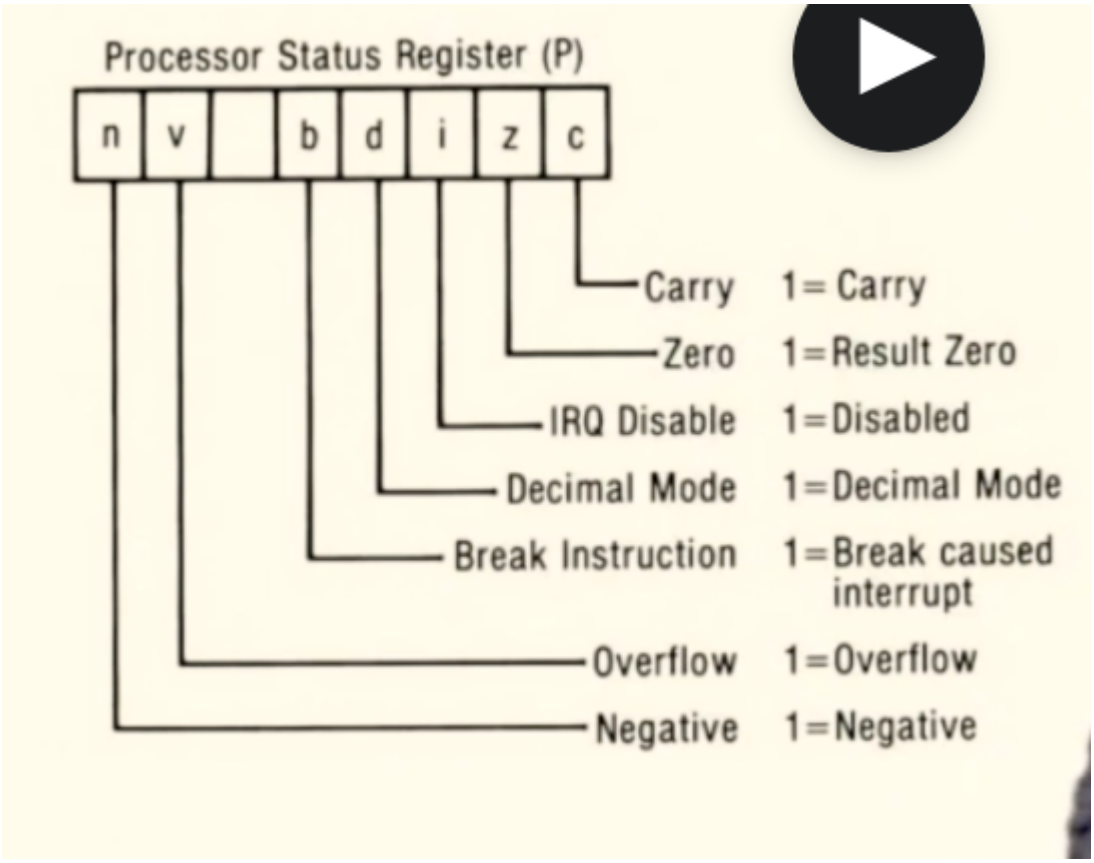


Grey part in the diagram is the processor

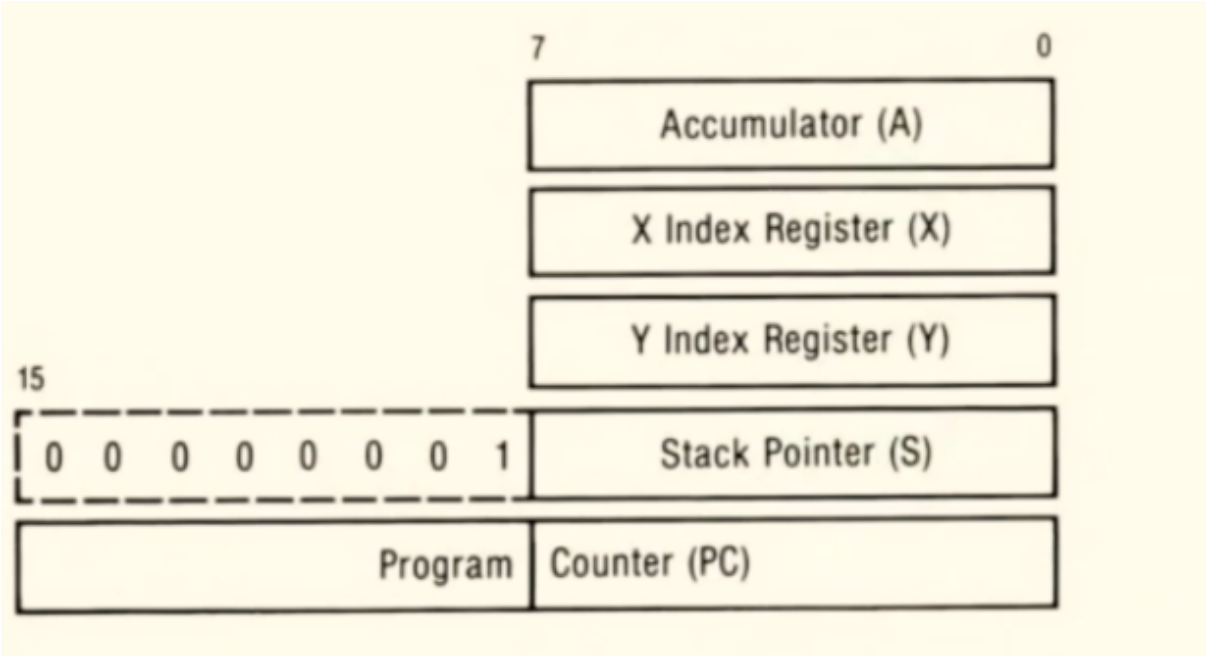
1. ALU (Arithmetic Logic Unit) - perform the calculations(add, sub, compare, etc). Gets value from both Accumulator and Data Bus. The result goes to the accumulator
2. Registers
    - PC (Program Counter) (16 bit register) - stores the address of the next instruction to be executed in the processor. This address is of ROM
    - SP (Stack Pointer) (16 bit register) - Points to the top of the stack where the data is available in RAM. Stack pointer holds the memory address of the stack in RAM and not the value. It shrinks(memory address decreases) on pushing values and grows(memory address increases) on popping values to the stack as shown below.



- P (Processor status register) - Stores flags based on what happened on the last execution on the processor, like positive, negative, zero, overflow status.

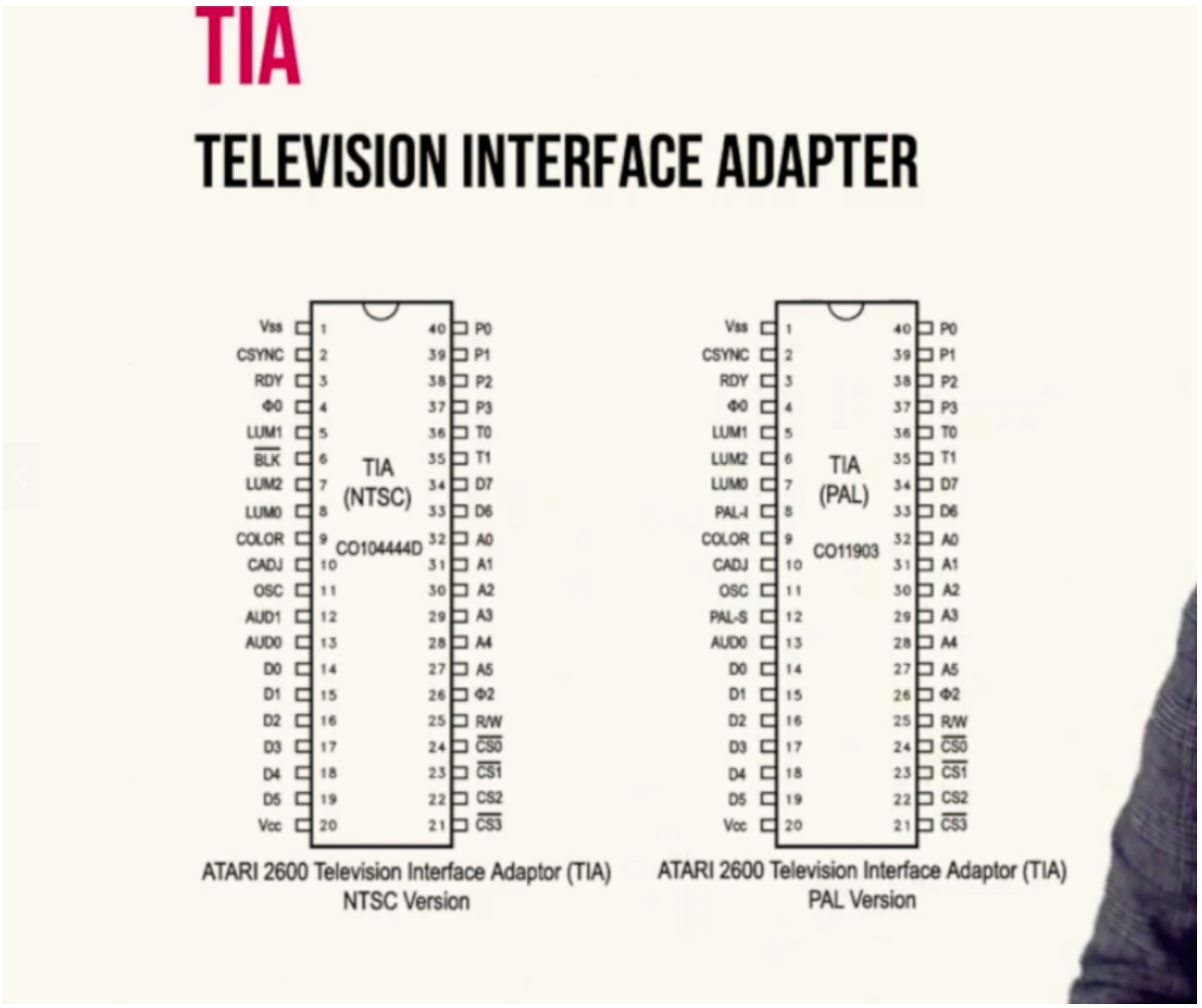


- X, Y(8 bit registers) - General purpose registers. Can store any values. For looping etc
- A (Accumulator) (8 bit register) - General purpose. For doing computation in ALU.Store value for the ALU to perform addition or subtraction. Also stores the result after ALU computation(addtion/subtraction result).

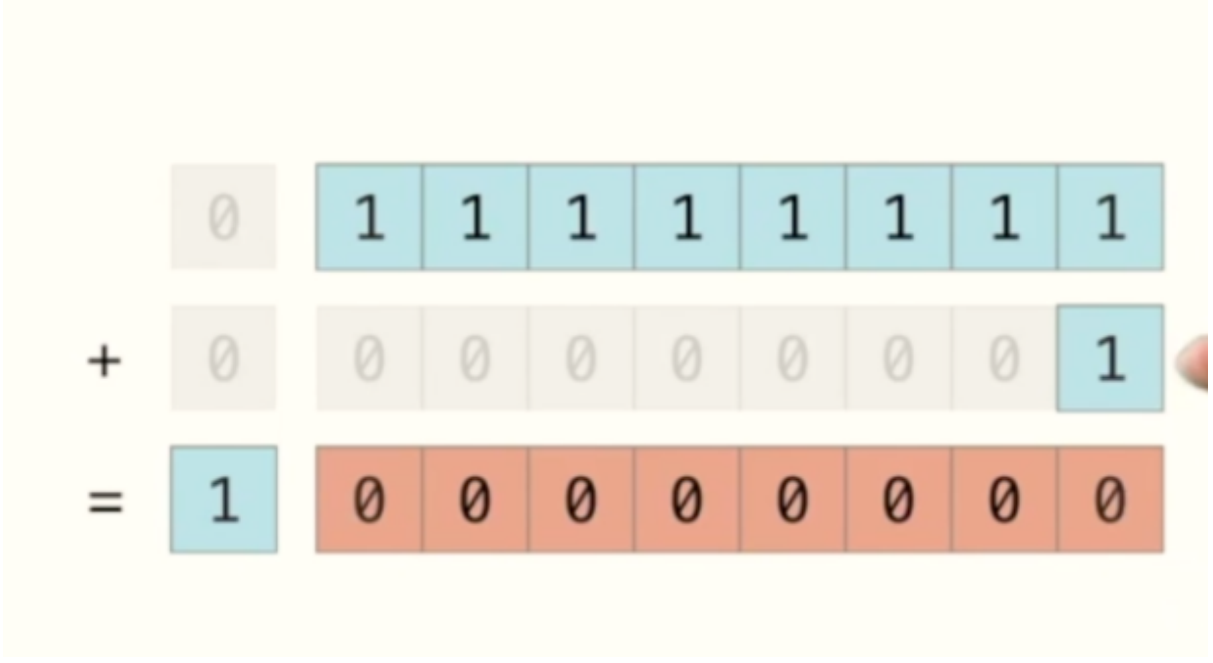


From the above diagram note that A, X and Y registers are 8 bit because they store value and S and PC are 16 bit registers because they need to store memory addresses of both RAM and ROM.
Also in the case of 6507 processor the manufacturer only allowed 9 bits for stack pointer (can be clearly seen on the above diagram as 00000001 11111111)

TIA chip



Carry flag and overflow flag

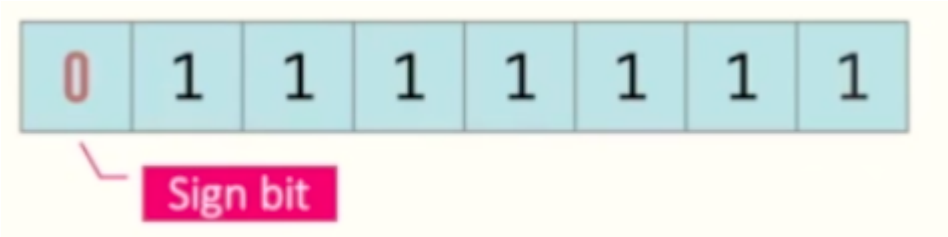


In the above example the 1 + 1 = 10 upon carry over finally gives the above answer where there is carry over of 1 at the MSB. This status is updated in the Process status register's C as 1.

n	v		b	d	i	z	c
0	0		0	0	0	0	1

Representation of negative numbers

- **Sign and Magnitude method**

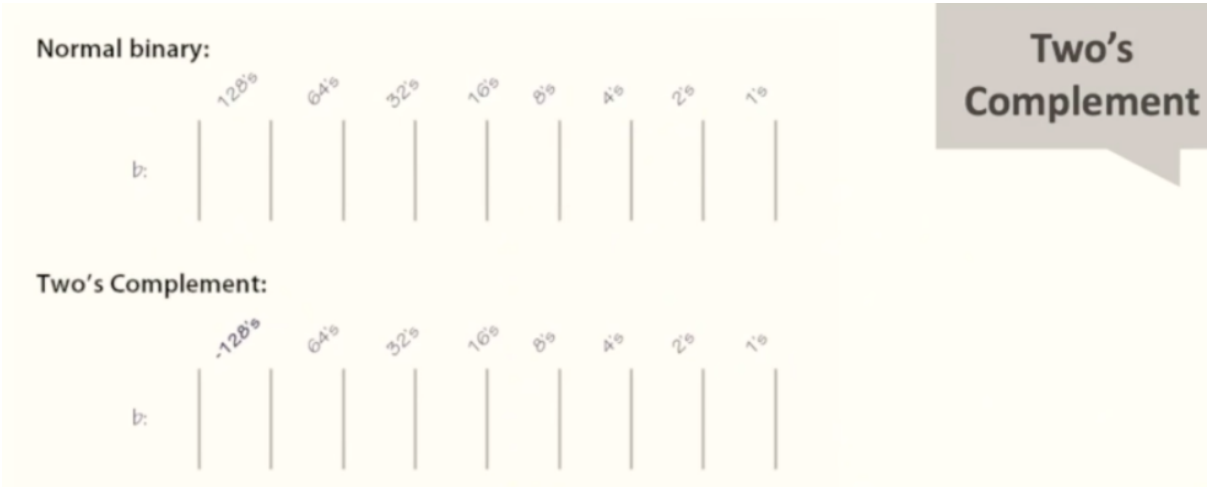


Left most bit is used to represent the sign bit.
1 - negative = -127 to 127 (2^7)
0 - positive = 255 (2^8)

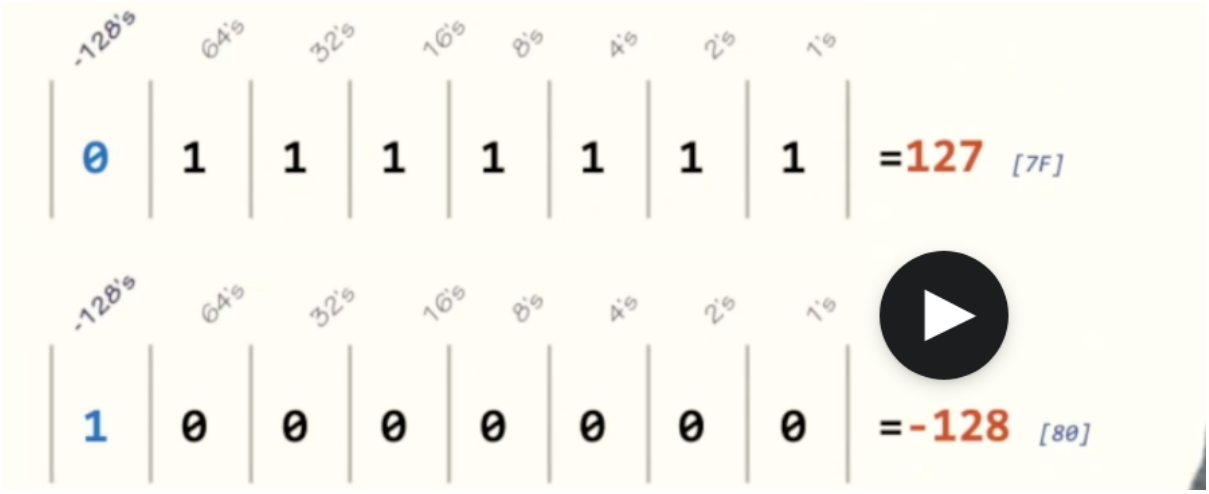
Drawback of this method is we can represent zero as both positive and negative as:
00000000 - positive zero
10000000 - negative zero

- **2's complement**

  Here left most bit alone represents -128 or +128



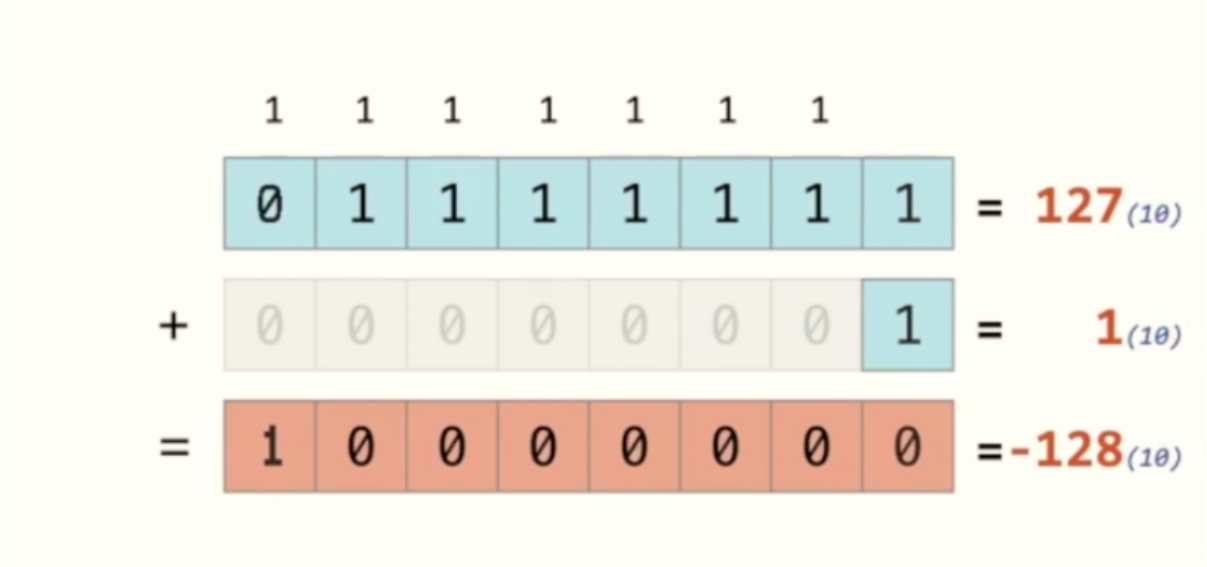
The most positive number that can be stored is 127
The least negative number that can be stored is -128,
As shown below.



In the below diagram we can see that -1 is represented. This is obtained by
-128 + (64+32+16+8+4+2+1)
-128 + 127 = -1

Two's complement overflow

Consider the below example of adding 127 + 1. This should give answer 128 but we get the answer as -128. This is considered as 2's complement overflow.



To disallow this and state it as positive +128 the processor status register is updated at the v(oveflow flag) as shown below.

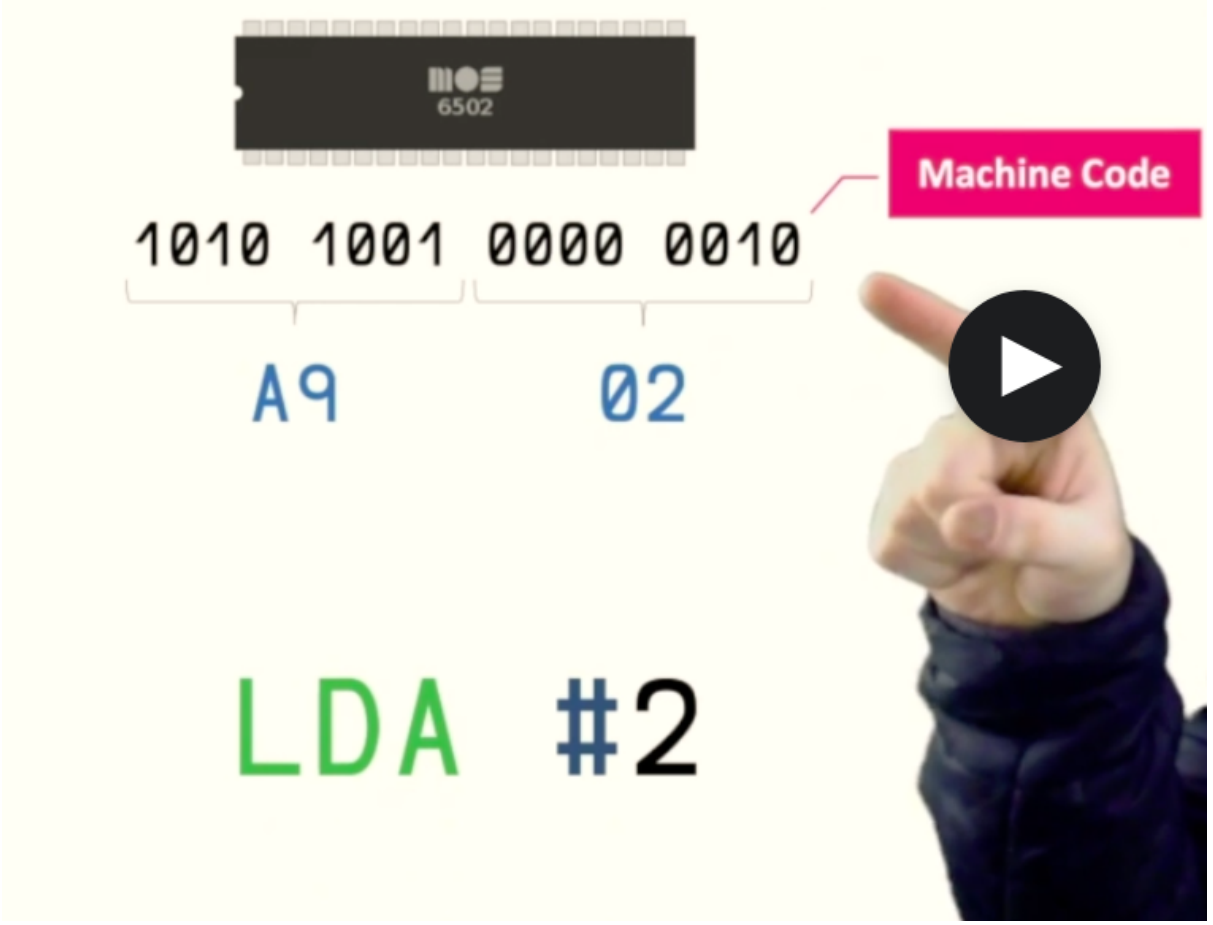
n	v		b	d	i	z	c
0	1		0	0	0	0	0

Assembler

Instructions to the processor
Example -

1. Load the decimal value 2 into the A register, can be represented as
   1010 1001 0000 0010 = A9 02 (OPCODE).
   Here A9 02 means load the register A with the value 02

   The assembly code for the above is LDA #2
   ⇒ Load the A register with binary value 2

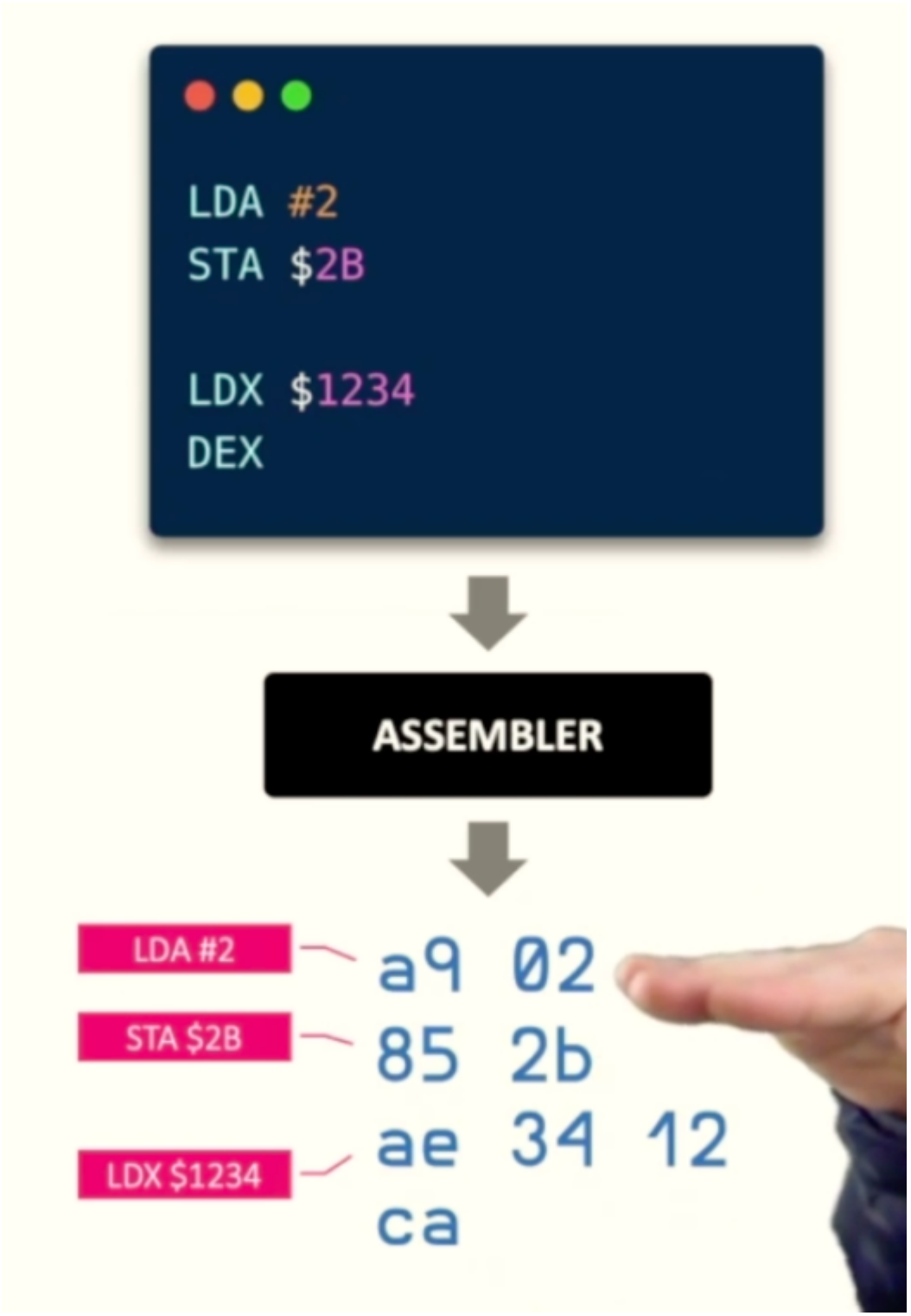


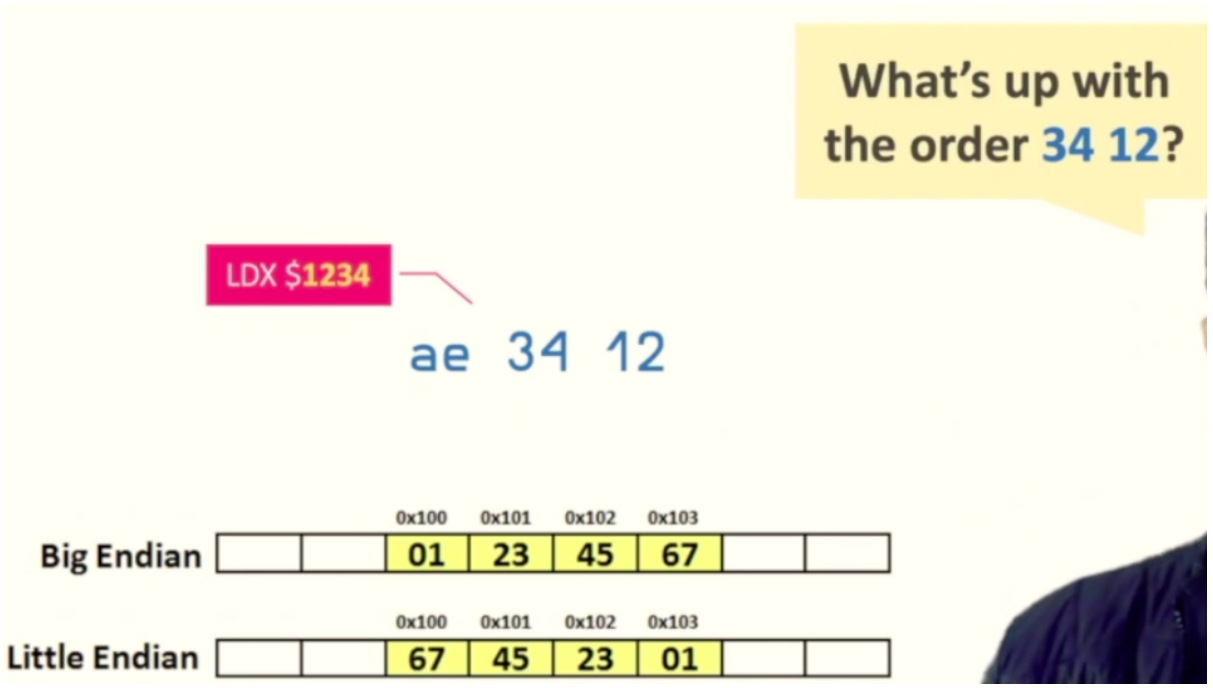
Definition: -The assembly code is converted to the binary with the help of assembler

Assembly flow:

Example:

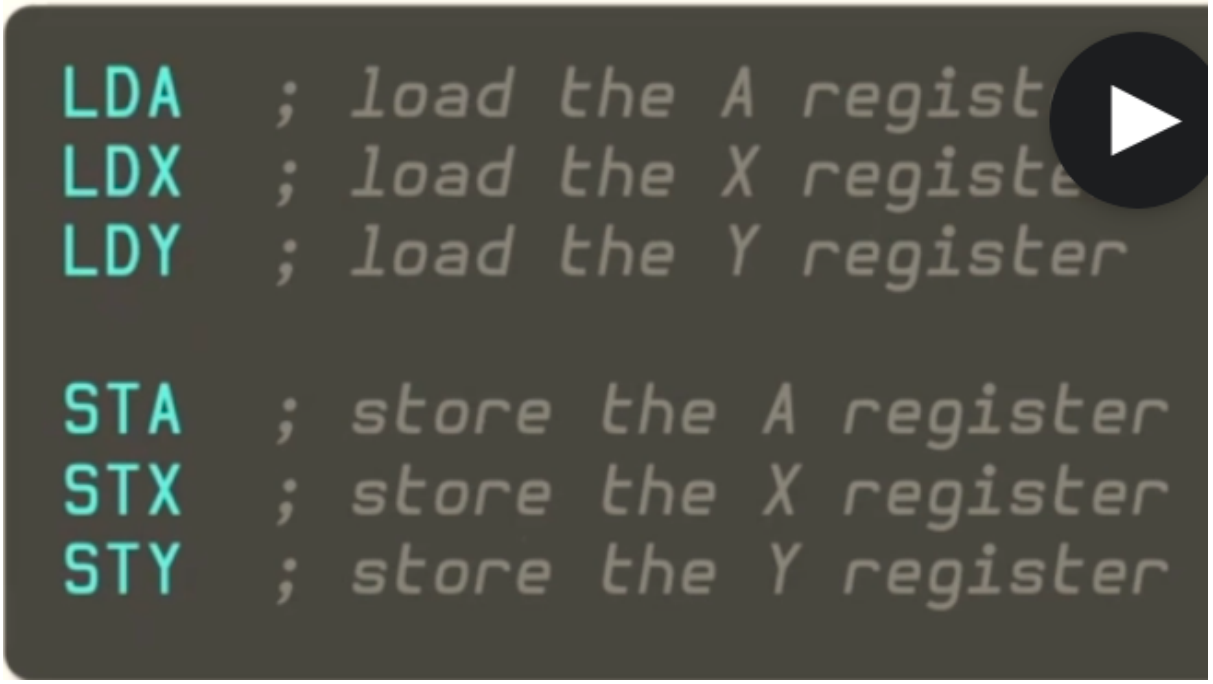
```
LDA #2        ;Load the A register with decimal value 2
STA $2B       ;Store the A register value(2) to the RAM of memory address 2B
LDX $1234     ;Load the value at the RAM of address 1234 to register X
DEX           ;Decrement the value of X register by 1; (X - -)
```


LDA #2
STA $2B
LDX $1234
DEX
ASSEMBLER
LDA #2
STA $2B
LDX $1234
a9 02
85 2b
ae 34 12
ca



Assembly Instructions

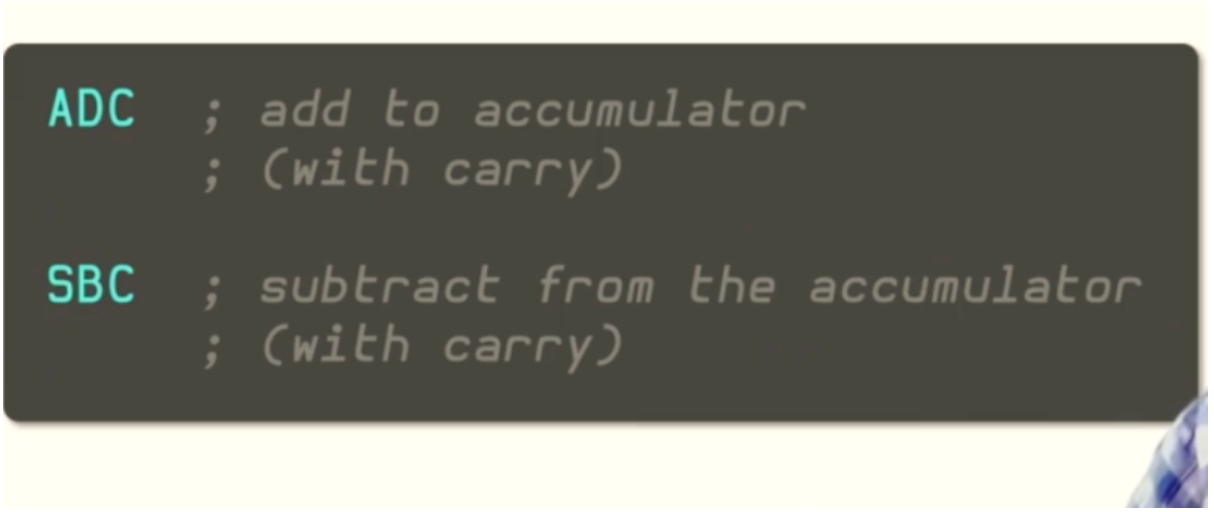
1. **Load and store instructions**:



Notes:

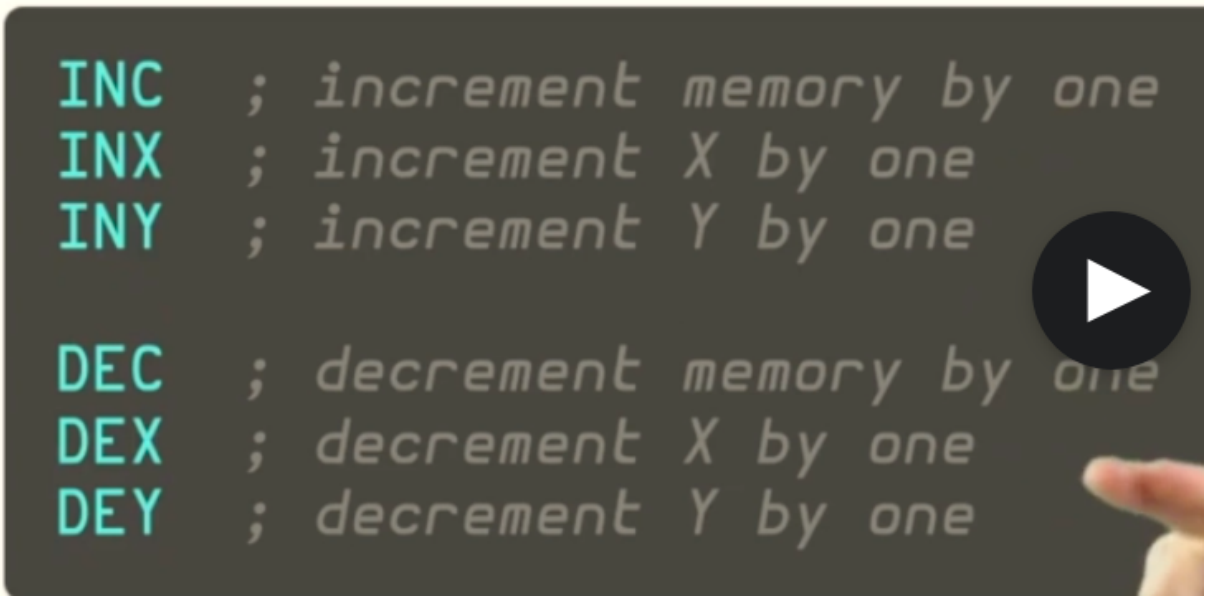
- LD is used to load to register and ST is used to store to RAM from register

2. **Arithmetic instructions**:



CLC - clears the carry flag, usually done before addition
SEC - set the carry flag, usually done before subtraction

3. **Increment and Decrement instructions**:



Increment and decrement instructions can set flags.

- If we decrement a register value and the result is zero then the Z flag is set as below.
  z=1 // for zero
  z=0 // for non zero

n	v		b	d	i	z	c
0	0		0	0	0	1	0

- Similarly if the result is negative i.e if the leftmost 7th bit is 1 then N flag is set
  N=1 // if negative; N=0 // is positive

n	v		b	d	i	z	c
1	0		0	0	0	0	0

4. **Jump and Branch Instructions**:

JUMP - jumps to the preferred location without conditions
Branch - takes place based on the status flag set based on the last instruction executed .

```
JMP   ; Jump to another location      GOTO

BCC   ; Branch on carry clear         C == 0
BCS   ; Branch on carry set           C == 1
BEQ   ; Branch on equal to zero       Z == 1
BNE   ; Branch on not equal to zero   Z == 0
BMI   ; Branch on minus               N == 1
BPL   ; Branch on plus                N == 0
BVC   ; Branch on overflow clear      V == 0
BVS   ; Branch on overflow set        V == 1
```

Looping with Assembly code

```
    ldy #100   ;y=100
Loop:
    dey        ;y--
    bne Loop   ;repeat until y==0
```

Here (Loop:) intended to the left represents the label.
bne Loop - go(branch) to the loop if Z(flag) == 0 which occurs until y - - => y == 0.

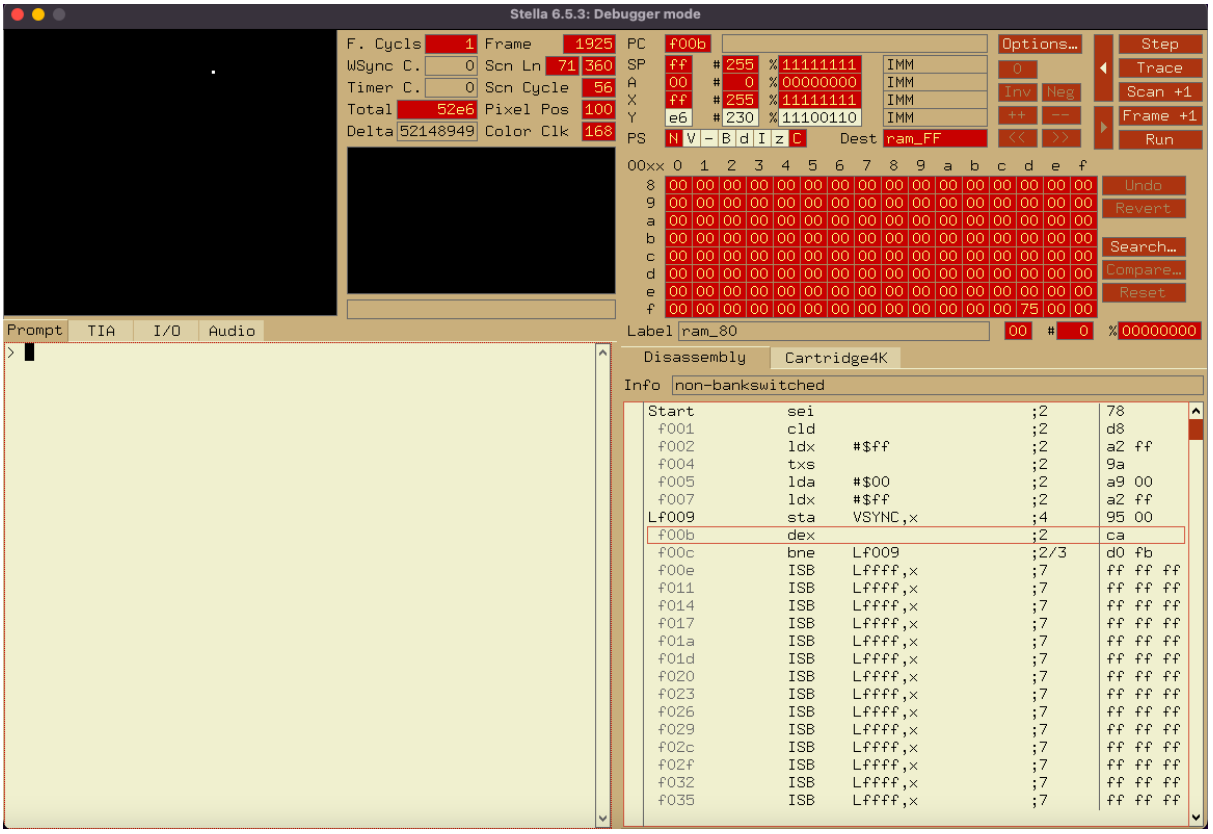
First Assembly Code

Clean memory:

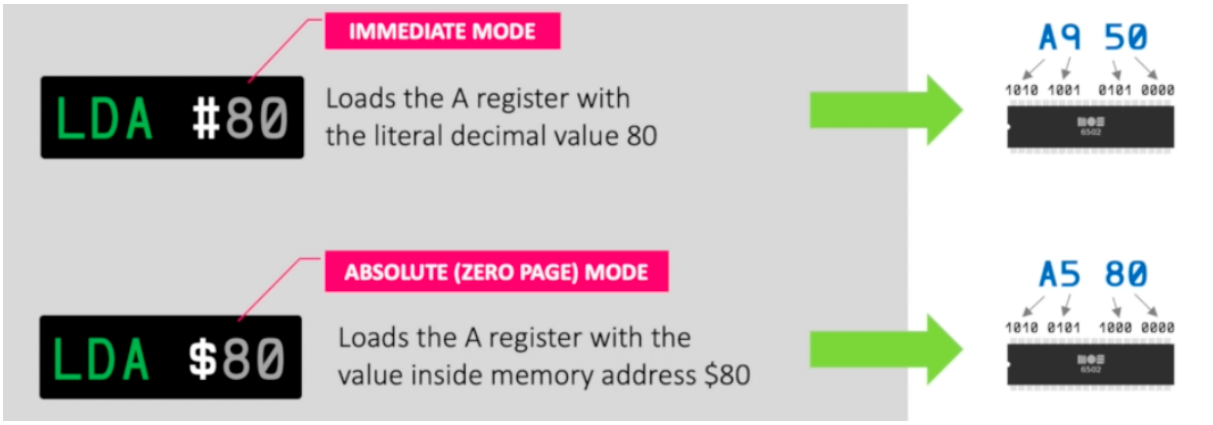
```
   processor 6502

   seg code
   org $F000 ;Define code origin at ROM $F000

Start:
   sei       ;Disable interrupt
   cld       ;Disable the BCD decimal math mode
   ldx #$FF  ;Loads the X register with #$FF // Done for setting the stack
pointer in the next step
   txs       ;Transfer the X register to the Stack Pointer

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;Clear the Page Zero ($00 to $FF) region of the
RAM;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;Clearing the Entire RAM and TIA
register;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;

   lda #0      ;A=0
   ldx #$FF    ;X=#$FF

MemLoop:
   sta $0,X    ;Store the value of A inside memory address $0+X
   dex         ;X - -
   bne MemLoop ;branch till X register is 0 or the Z flag is 1



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;
   ;; Fill the ROM size to 4KB

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;

   org $FFFC
   .word Start ;Reset vector $FFFC ROM where the program starts ;2 bytes
   .word Start ;interrupt vector at position $FFFE ROM (unused in the VCS)
;2bytes
```

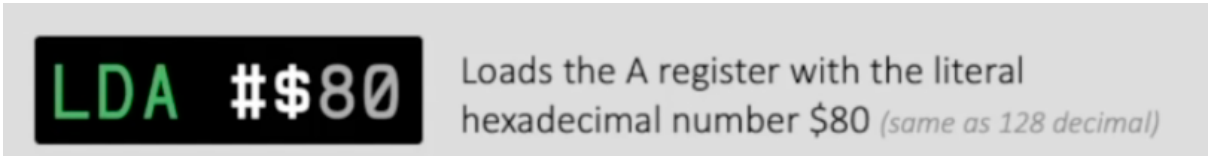
dasm cleanmem.asm -f3 -v0 -oclean.bin ⇒ command to execute the above program to binary

The representation of this in the Processor is shown below:



Addressing modes





Note: # is for decimal

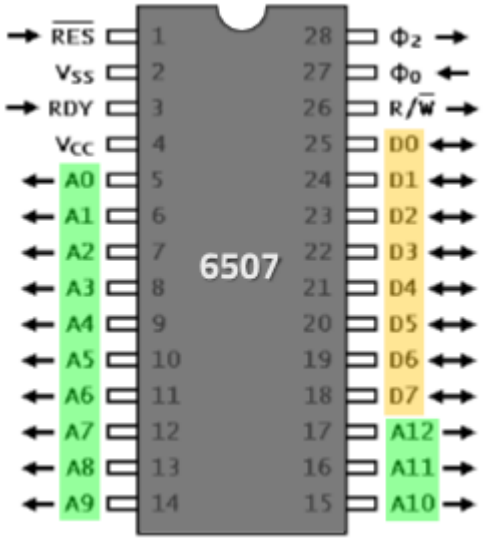
$ is for binary(memory address)

#$ for storing literal hexadecimal value

Sending data with instructions is also possible with the 6502 CPU. Here are some examples of instructions that also allow us to send data to be processed:

- `LDA #5` ; loads the literal decimal number 5 into the accumulator.
- `ADC #1` ; add the literal decimal value 1 to the accumulator, storing the result back into the accumulator.
- `STA $80` ; stores the value inside the accumulator into memory location $80.

And this whole discussion should also clarify something else about the processor. Remember how we have different pins for *data* and *address* in the 6507 processor?



The 8 ***data*** pins and the 13 ***address*** pins serve different purposes.

We know that the 6507 is an 8-bit processor, which means the registers are 8-bit in size and the CPU handles data eight-bits at a time. On the bus, these would be lines (D0, D1, D2, …, D7).

The other 13 lines (A0, A1, A2… A12) are used to choose the *address in memory* the CPU wants to read from or write to.

Speaking of read and write, do you see that pin on the top right that is labelled **R/W**? That is the pin that instructs the CPU if we are *reading from* or *writing to* the address. All these pins need to be correctly stimulated for the processor to correctly execute the instruction. Sending the correct assembled opcodes is all we need to do to make that happen.

Representation of values

Example problems

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
```

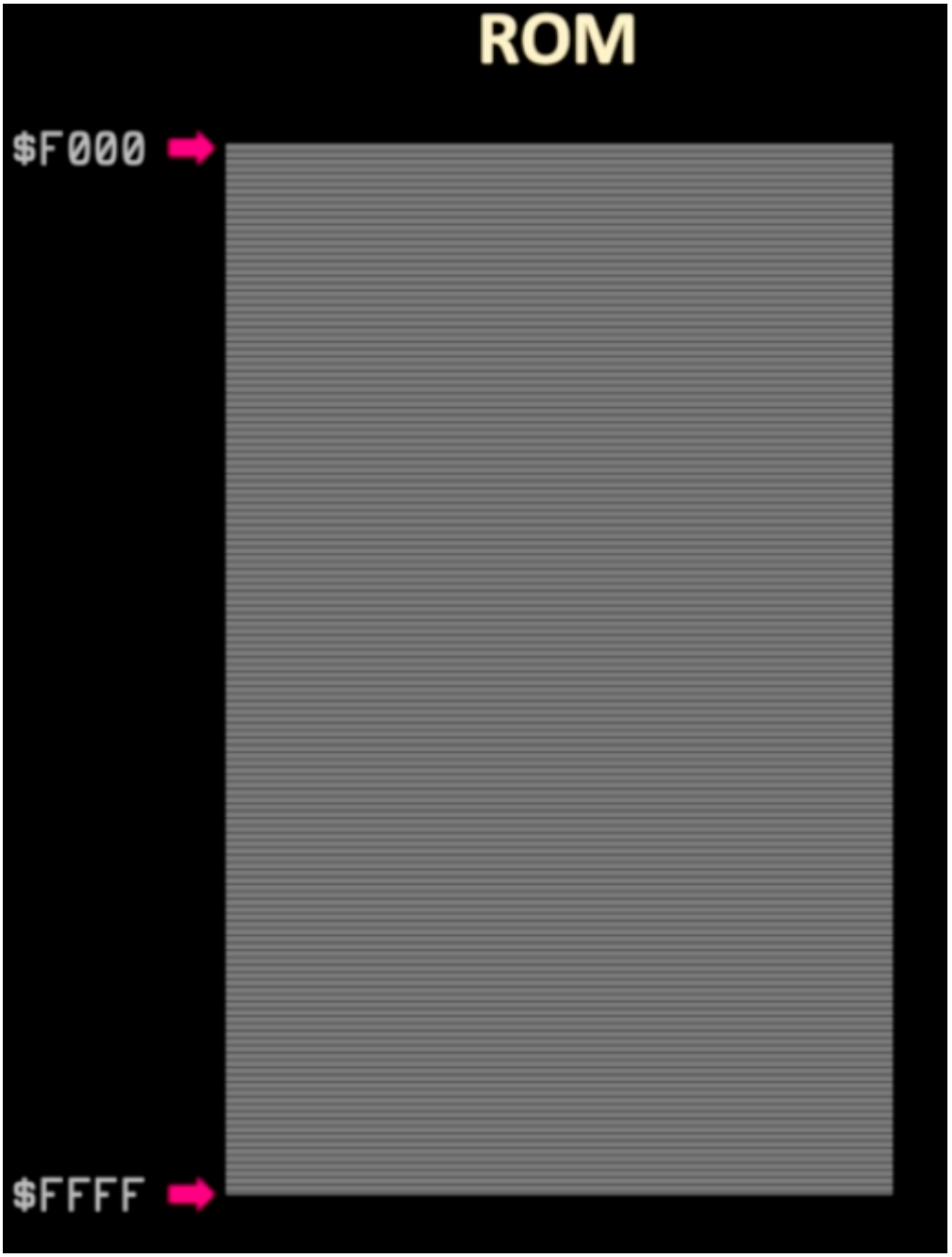
```
    ; EXERCISE 2

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

    jmp Start         ; Jump to the Start label to force an infinite loop


    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
```

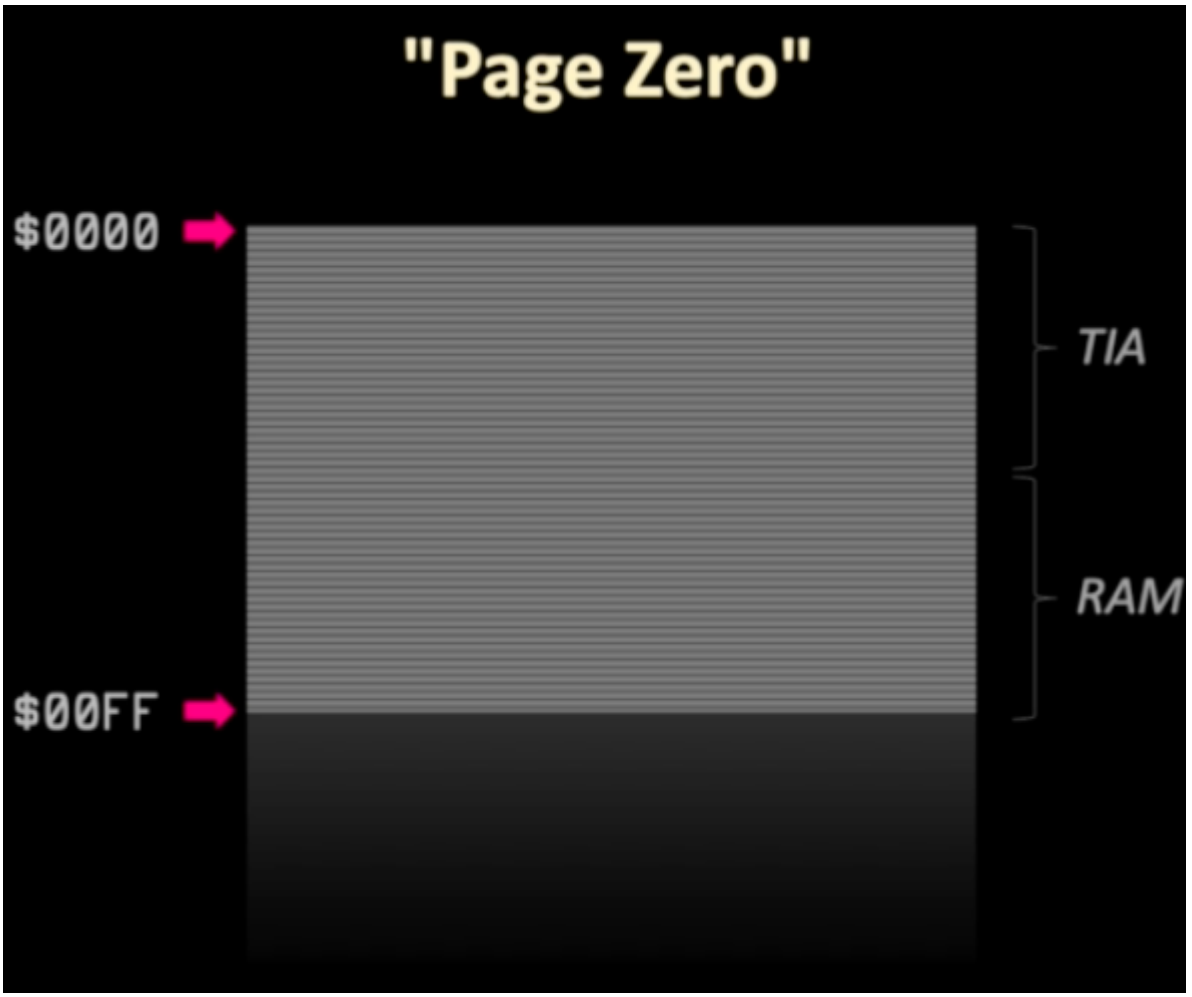
```
    ; EXERCISE 3

    processor 6502
    seg Code          ; Define a new assembler segment named "Code"
    org $F000         ; Define the origin of the ROM code at 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 is no TXY or TYX.
                      ; Therefore we can't transfer directly X to Y or Y to X
                      ; If we wish to do so, we must go through the A

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

    jmp Start         ; Jump to the Start label to force an infinite loop


    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
```

In 6502 adding is done with CLC(clear carry flag) followed by ADC.
Subtracting is always done with SEC(set carry flag) followed by SBC.

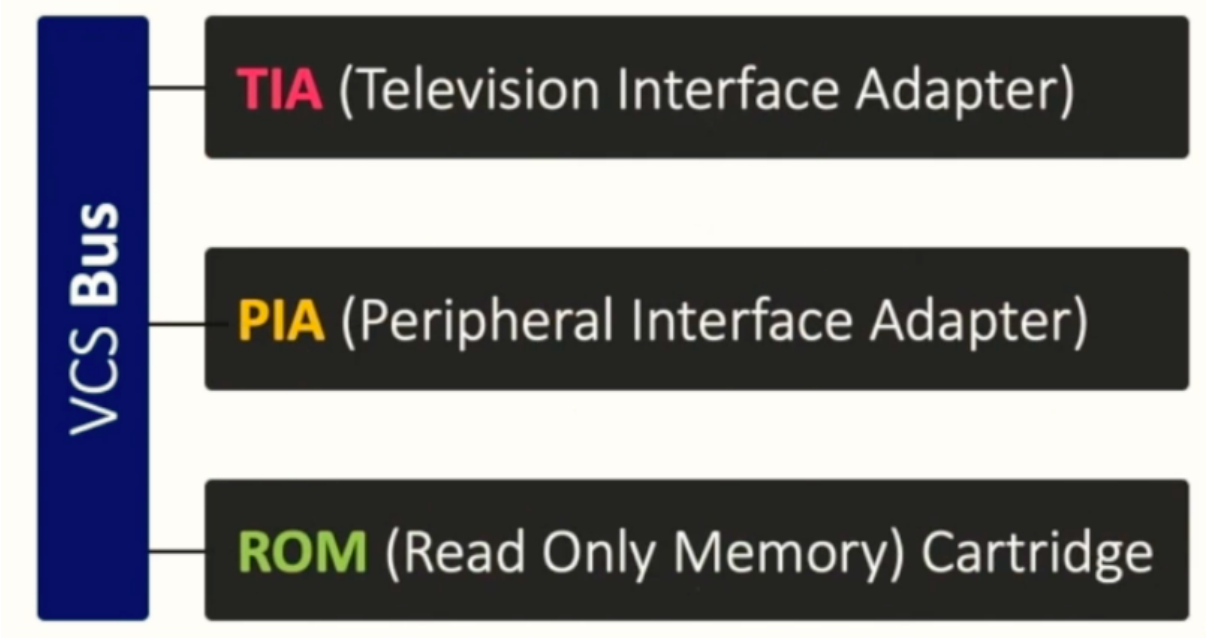
```
Start:
    lda #100        ; Load the A register with the literal decimal value 100

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

    sec             ; We always set the carry flag before calling SBC
    sbc #10         ; Subtract (with carry) the decimal #10 from accumulator
                    ; Register A should now contain the decimal 95 (or $5F in hex)
```

VCS Memory Map

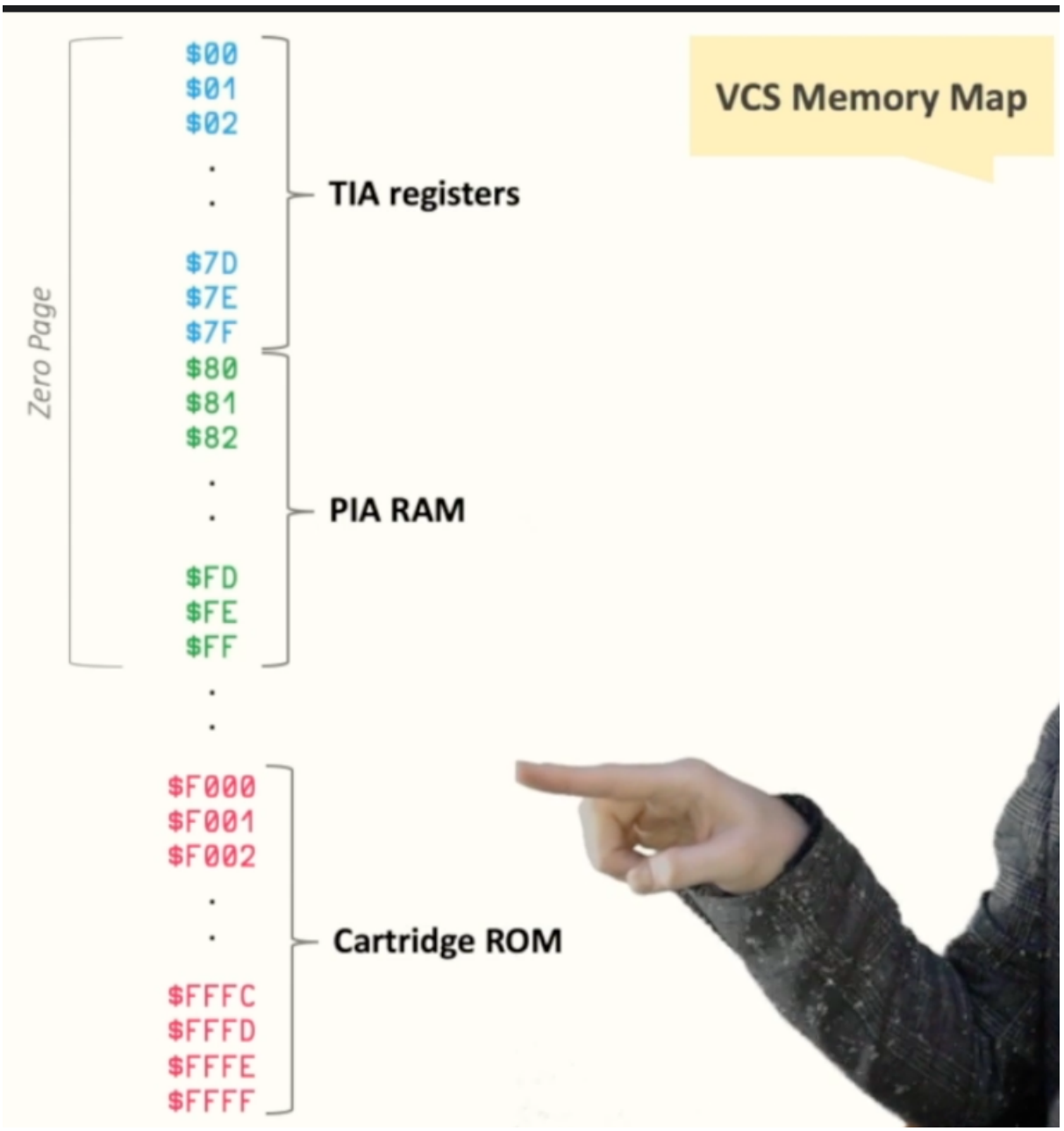




Address space



TIA - responsible for setting color of player, background etc
PIA - is the RAM
ROM - the cartridge



0000-002C – TIA (write)
0030-003D – TIA (read)
0080-00FF – RIOT (RAM)
0280-0297 – RIOT (I/O, Timer)
F000-FFFF – Cartridge (ROM)