Atari 2600 Assembly

Martin Tromblee

11/13/2020

01 What is an ‘Atari 2600’?
02 What is ‘Assembly’?
03 Why? *(Its 2020!)*
04 Let’s Build Something

What is an ‘Atari 2600’?

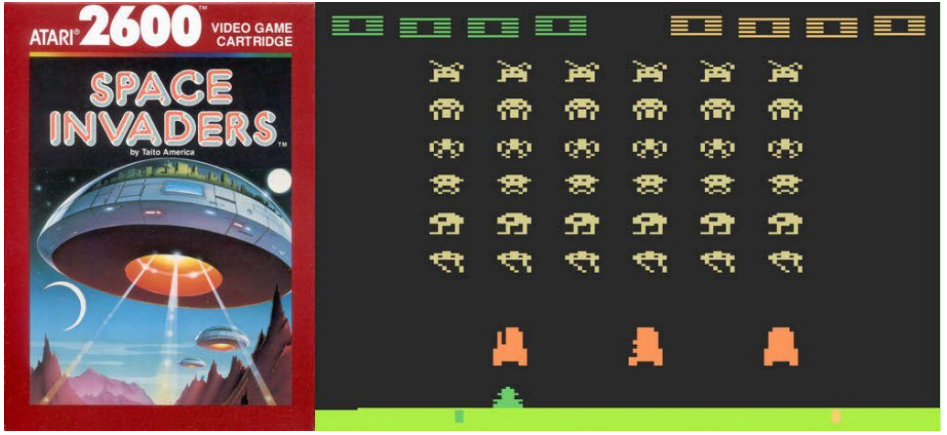
Released in 1977

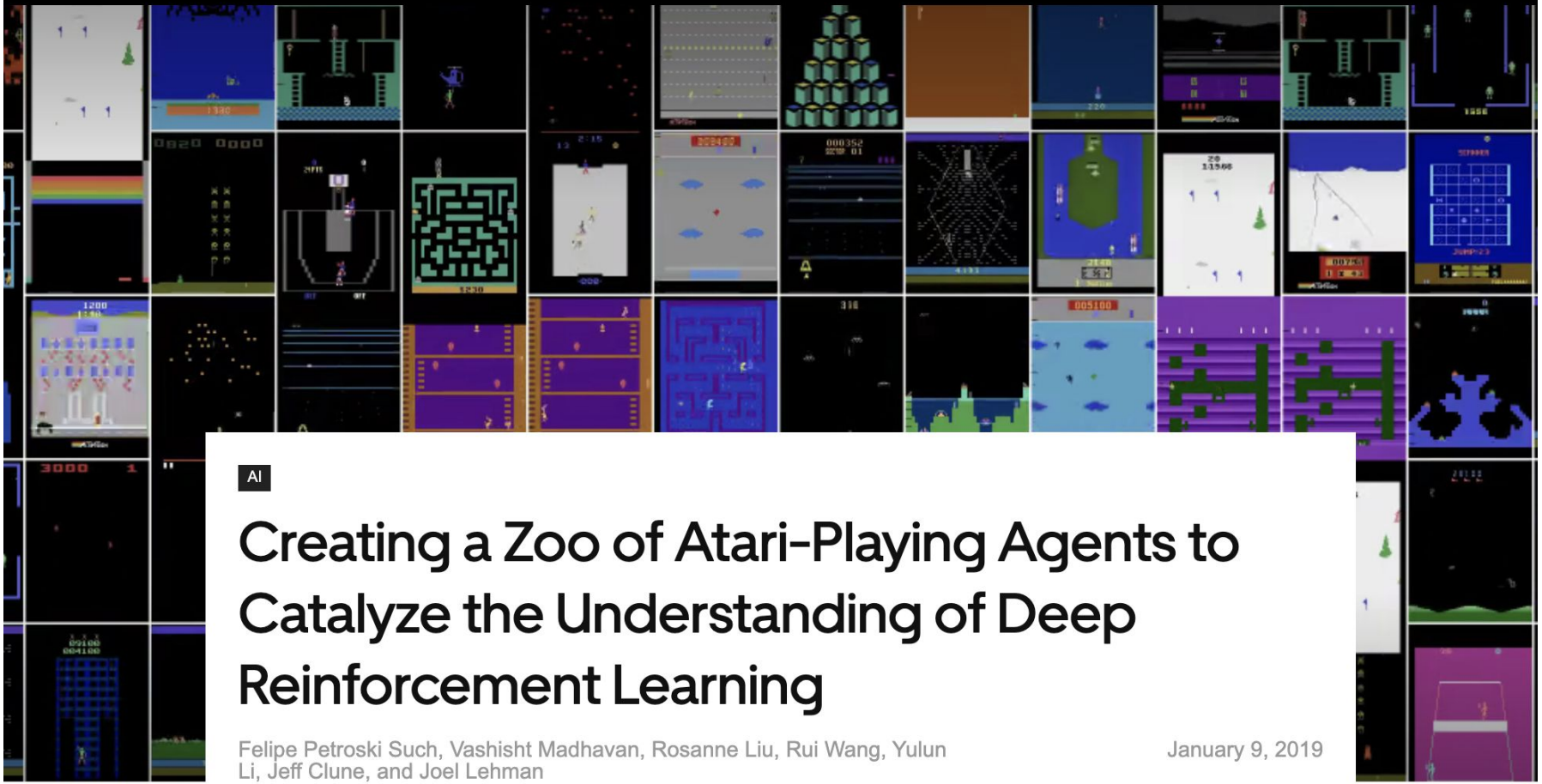
$199

30 million sold

Over 400 games

Connects directly to a TV





AI

Creating a Zoo of Atari-Playing Agents to Catalyze the Understanding of Deep Reinforcement Learning

Felipe Petroski Such, Vashisht Madhavan, Rosanne Liu, Rui Wang, Yulun Li, Jeff Clune, and Joel Lehman

January 9, 2019

01 What is an ‘Atari 2600’?
02 What is ‘Assembly’?
03 Why? *(Its 2020!)*
04 Let’s Building Something

What is ‘Assembly’?

What OS does the Atari 2600 use?

None ⇒ Need to talk directly to the chips (CPU & various controllers)

How?

Machine language ← Assembly


```
JetYPos = 10
```

01 What is an ‘Atari 2600’?
02 What is ‘Assembly’?
03 Why? *(Its 2020!)*
04 Let’s Building Something

Why? *(Its 2020!)*

01 What is an ‘Atari 2600’?
02 What is ‘Assembly’?
03 Why? *(Its 2020!)*
04 Let’s Building Something

Live

CODE

enemy.asm

Tools

Assembly

be proficient

It’s about the Chips...

Specifically the CPU

MICROPROCESSOR
6507
A/X/Y

Accumulator (A)
All logical and arithmetic operations

Two Registers (X/Y)
Storage and increment/decrement a value

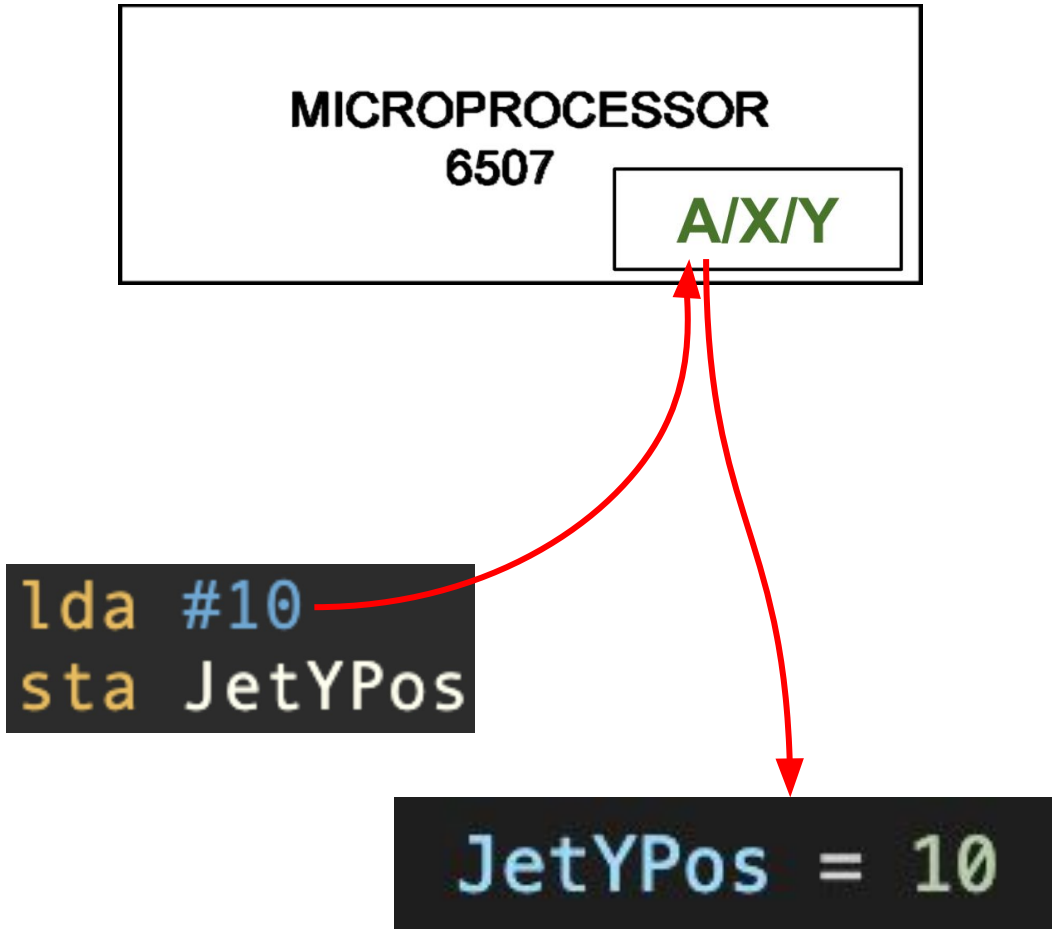
8-Bit CPU ⇒ All registers are 1 byte

ADC	add with carry	JSR	jump subroutine
AND	and (with accumulator)	LDA	load accumulator
ASL	arithmetic shift left	LDX	load X
BCC	branch on carry clear	LDY	load Y
BCS	branch on carry set	LSR	logical shift right
BEQ	branch on equal (zero set)	NOP	no operation
BIT	bit test	ORA	or with accumulator
BMI	branch on minus (negative set)	PHA	push accumulator
BNE	branch on not equal (zero clear)	PHP	push processor status (SR)
BPL	branch on plus (negative clear)	PLA	pull accumulator
BRK	break / interrupt	PLP	pull processor status (SR)
BVC	branch on overflow clear	ROL	rotate left
BVS	branch on overflow set	ROR	rotate right
CLC	clear carry	RTI	return from interrupt
CLD	clear decimal	RTS	return from subroutine
CLI	clear interrupt disable	SBC	subtract with carry
CLV	clear overflow	SEC	set carry
CMP	compare (with accumulator)	SED	set decimal
CPX	compare with X	SEI	set interrupt disable
CPY	compare with Y	STA	store accumulator
DEC	decrement	STX	store X
DEX	decrement X	STY	store Y
DEY	decrement Y	TAX	transfer accumulator to X
EOR	exclusive or (with accumulator)	TAY	transfer accumulator to Y
INC	increment	TSX	transfer stack pointer to X
INX	increment X	TXA	transfer X to accumulator
INY	increment Y	TXS	transfer X to stack pointer
JMP	jump	TYA	transfer Y to accumulator

Load memory → CPU
LD**A** →A
LD**X** →X
LD**Y** →Y

Store CPU → memory
ST**A** A→
ST**X** X→
ST**Y** Y→

Transfer CPU Register → CPU Register
T**AX** A→X
T**AY** A→Y
T**XA** X→A
T**YA** Y→A



Logic

AND - AND
ORA - OR
EOR - Exclusive OR

Shift

Bits move one position left or right
ASL Arithmetic Shift Left
LSR Logical Shift Right

ASL

0	0	0	0	0	1	1	0	6
0	0	0	0	1	1	0	0	12

Branch	
BEQ BNE	A==B A!=B
BCC (BMI) BCS (BPL)	A<B (signed) A>=B (signed)

```
lda JetYPos     ; boundry check
cmp #70
bpl CheckXPos   ; JetYPos >= 70
```

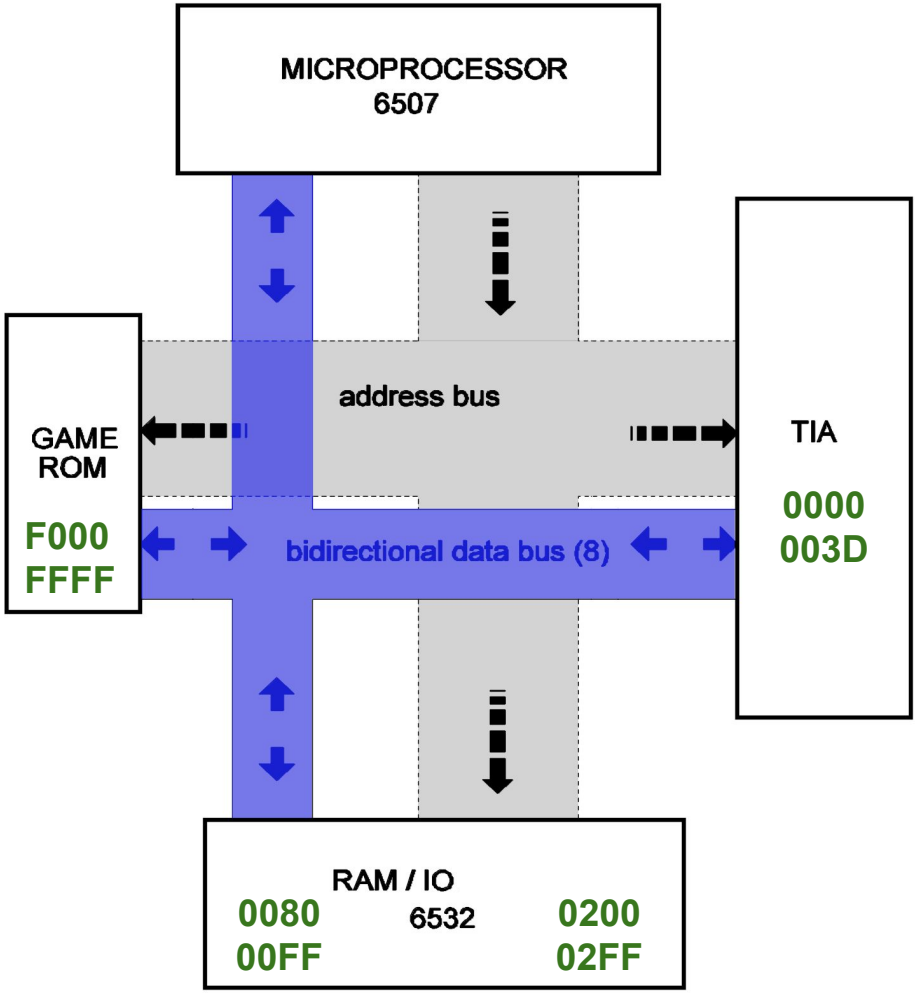
Compare

CMP Accumulator
CPX X
CPY Y

Environment

Atari 2600

where we build



Bus

Data - “What” (8-bit)

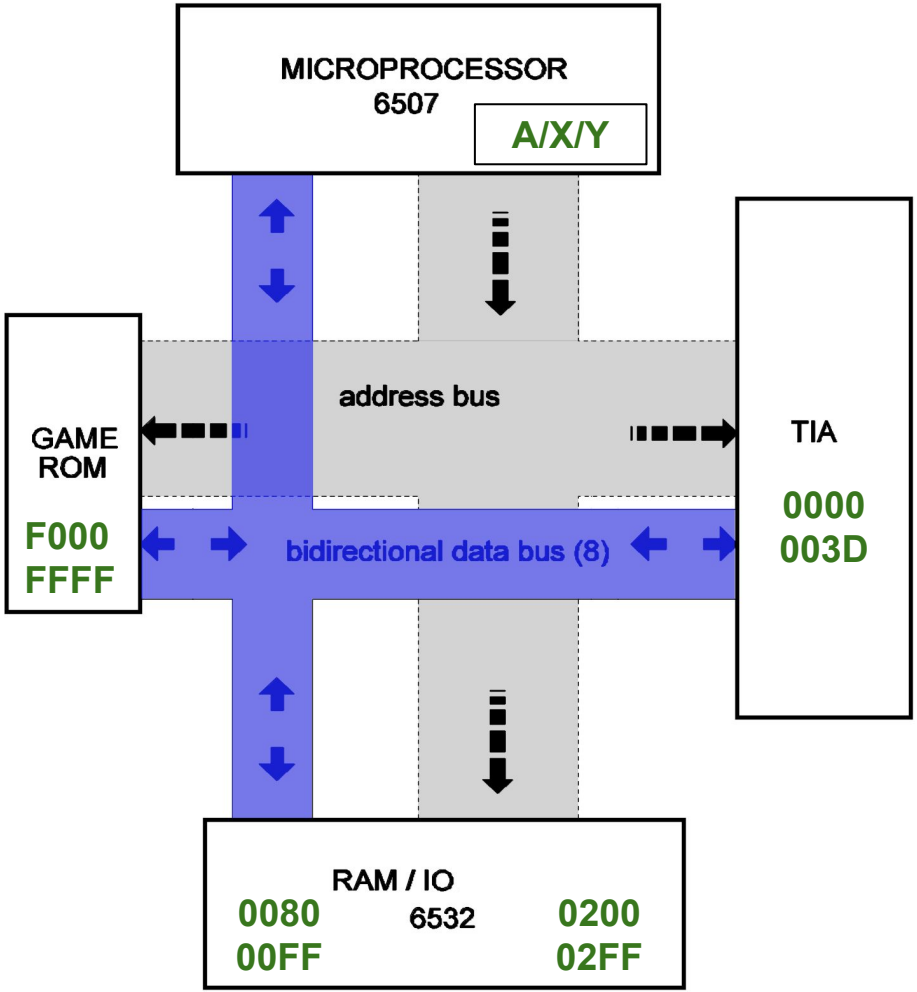
moves the data

Address - “Where” (16-bit)

Memory-mapped I/O

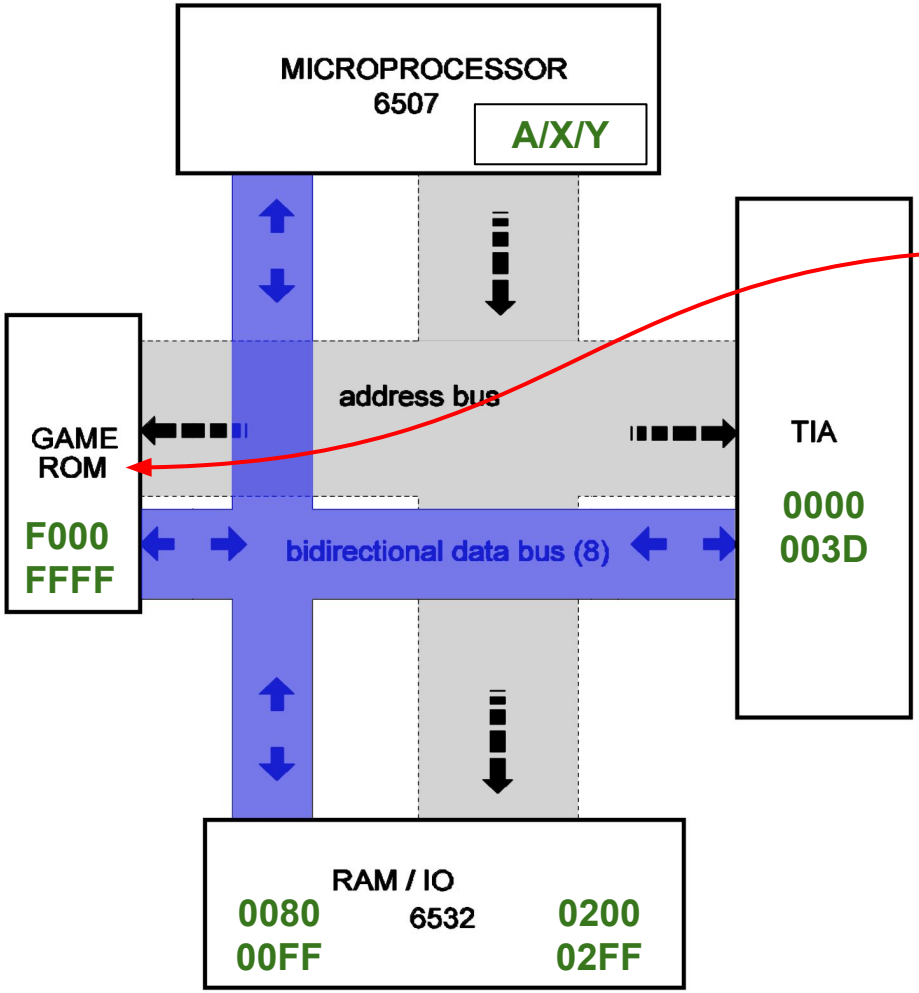
Same address space for both memory and I/O devices

Memory Address Range	
$0000 - $003D	TIA (TV)
$0080 - $0FF	RAM (128 bytes)
$0200 - $02FF	I/O (Joysticks)
$F000 - $FFFF	ROM (Our code)



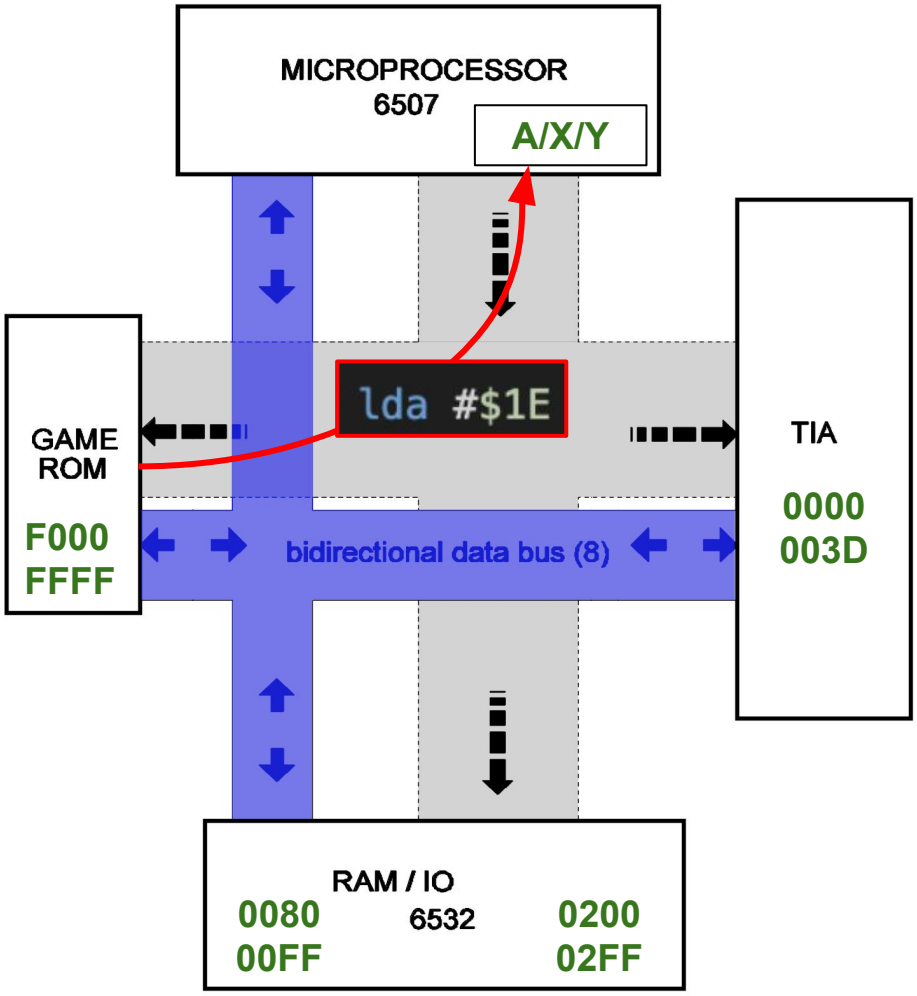
Screen.background = Color.yellow

```
    processor 6502

    seg code
    org $F000   ; defines origin of ROM


START:

    lda #$1E    ; color Yellow
    sta $09     ; address for background



    jmp START

    org $FFFC   ; Reset Vector
    .word START ;
    .word START ;
```



Screen.background = Color.yellow

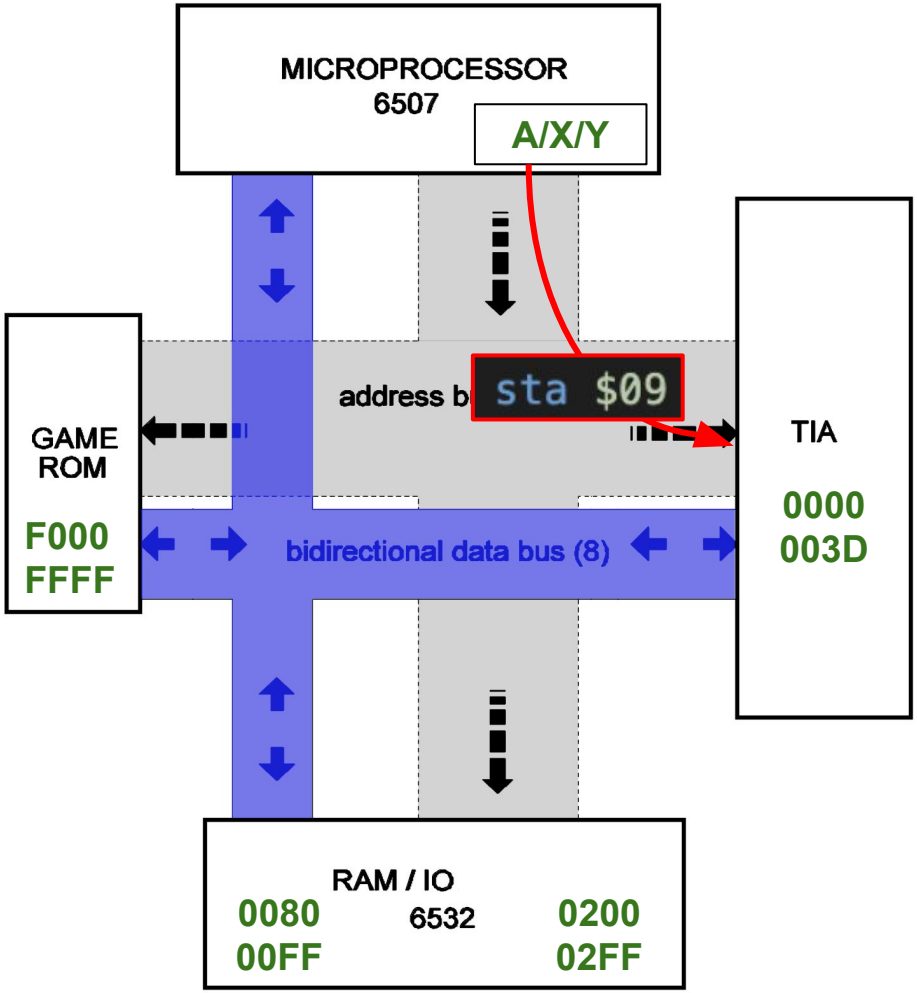
```
    processor 6502

    seg code
    org $F000   ; defines origin of ROM


START:

    lda #$1E    ; color Yellow
    sta $09     ; address for background



    jmp START

    org $FFFC   ; Reset Vector
    .word START ;
    .word START ;
```



Screen.background = Color.yellow

```
    processor 6502

    seg code
    org $F000   ; defines origin of ROM


START:

    lda #$1E    ; color Yellow
    sta $09     ; address for background
```

luminance/hue	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
2																
4																
6																
8																
10																
12																
14																



Screen.background = Color.yellow

```
    processor 6502

    seg code
    org $F000   ; defines origin of ROM


START:

    lda #$1E    ; color Yellow
    sta $09     ; address for background



    jmp START

    org $FFFC   ; Reset Vector
    .word START ;
    .word START ;
```

Code

```
JetSprite:
    .byte #%00000000
    .byte #%10000010
    .byte #%11000110
    .byte #%11111110
    .byte #%01111100
    .byte #%00111000
    .byte #%00111000
    .byte #%00010000
    .byte #%00010000
```

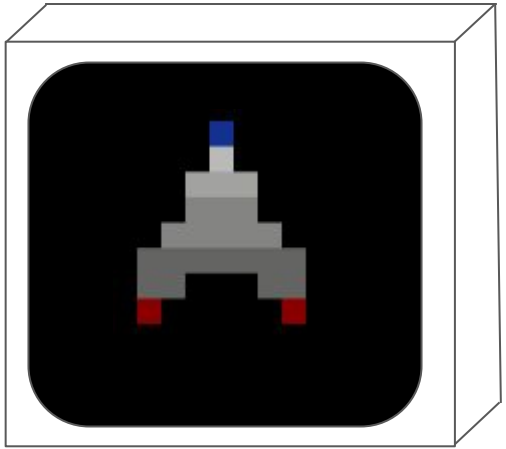
```
JetColor:
    .byte #$00
    .byte #$40
    .byte #$04
    .byte #$04
    .byte #$06
    .byte #$06
    .byte #$08
    .byte #$0A
    .byte #$92
```

Framebuffer

160x192 @ 7 bit = 27K RAM
$50/K ⇒ **$1,400**

0	0	0	92	0	0	0
0	0	0	A	0	0	0
0	0	8	8	8	0	0
0	0	6	6	6	0	0
0	6	6	6	6	6	0
4	4	4	4	4	4	4
4	4	0	0	0	4	4
E	0	0	0	0	0	E

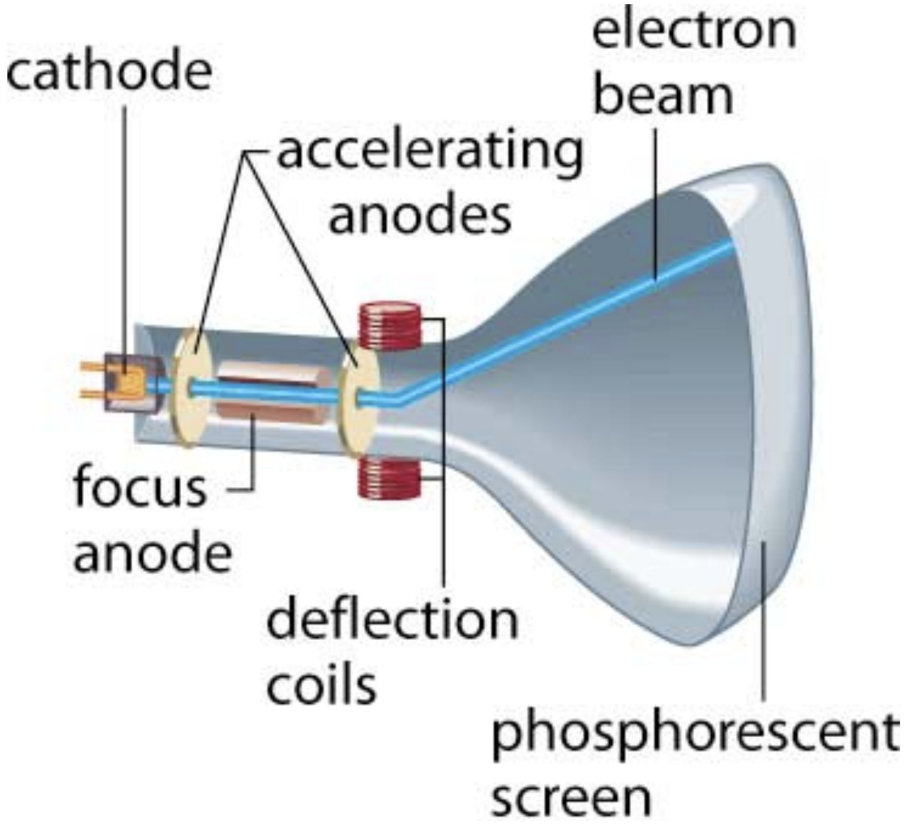
Display



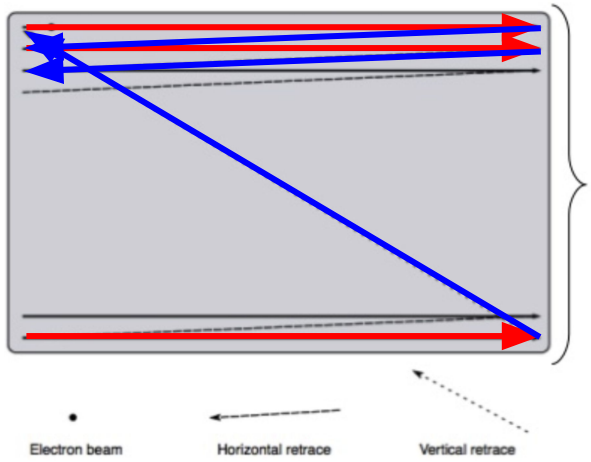
Environment

TV

constraint - memory cost



Scanlines



60 frames per second (NTSC standard)


Flex

Code

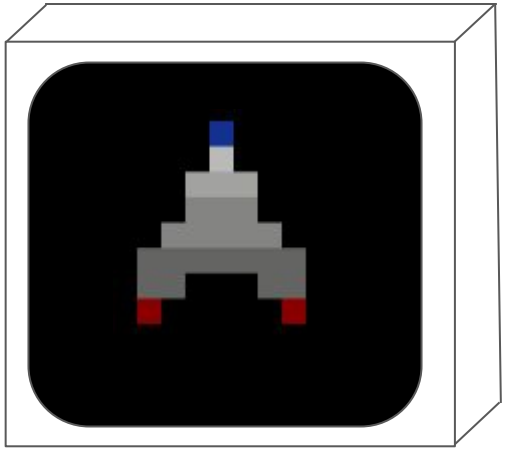
```
JetSprite:
    .byte #%00000000
    .byte #%10000010
    .byte #%11000110
    .byte #%11111110
    .byte #%01111100
    .byte #%00111000
    .byte #%00111000
    .byte #%00010000
    .byte #%00010000
```

```
JetColor:
    .byte #$00
    .byte #$40
    .byte #$04
    .byte #$04
    .byte #$06
    .byte #$06
    .byte #$08
    .byte #$0A
    .byte #$92
```

Framebuffer

0	0	0	92	0	0	0
0	0	0	A	0	0	0
0	0	8	8	8	0	0
0	0	6	6	6	0	0
0	6	6	6	6	6	0
4	4	4	4	4	4	4
4	4	0	0	0	4	4
E	0	0	0	0	0	E

Display



Code

```
JetSprite:
    .byte #%00000000
    .byte #%10000010
    .byte #%11000110
    .byte #%11111110
    .byte #%01111100
    .byte #%00111000
    .byte #%00111000
    .byte #%00010000
    .byte #%00010000
```

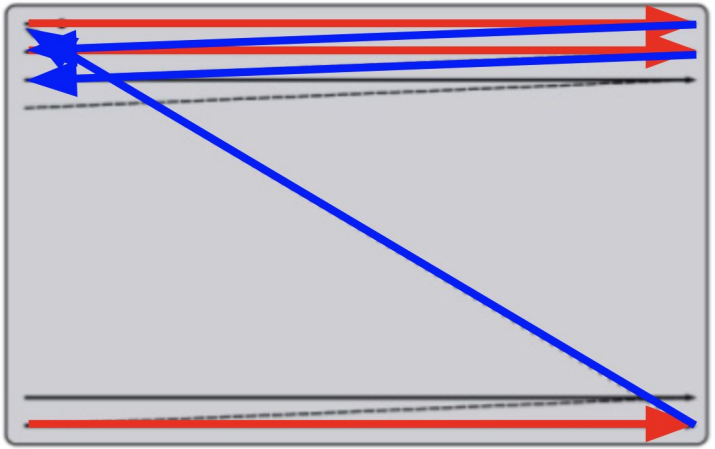
```
JetColor:
    .byte #$00
    .byte #$40
    .byte #$04
    .byte #$04
    .byte #$06
    .byte #$06
    .byte #$08
    .byte #$0A
    .byte #$92
```

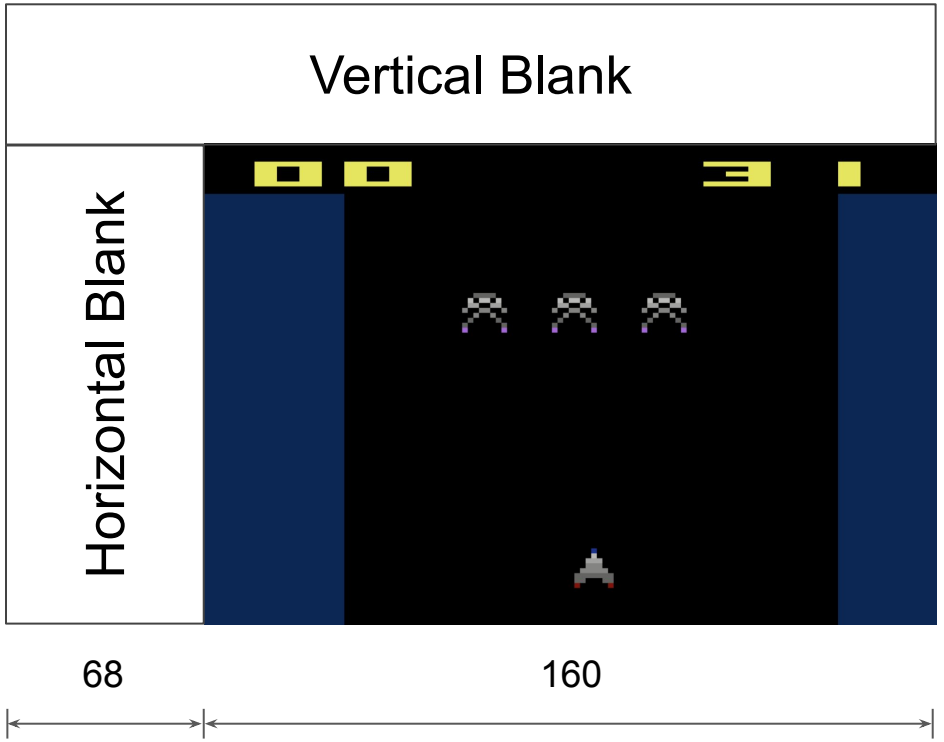
TIA

0000

003D

Display





160x192
Pixel=Clock cycle

HBlank = Newline Reset

VBlank = Frame Reset

TIA 3.58 MHz
CPU 1.19MHz

1 CPU = 3 TIA

Sprite Display

```
JetXPos          byte
JetYPos          byte
JetSpritePtr     word
JetColorPtr      word

; Constants
JET_HEIGHT = 9
```

```
; init
lda #80
sta JetYPos
lda #68
sta JetXPos
```

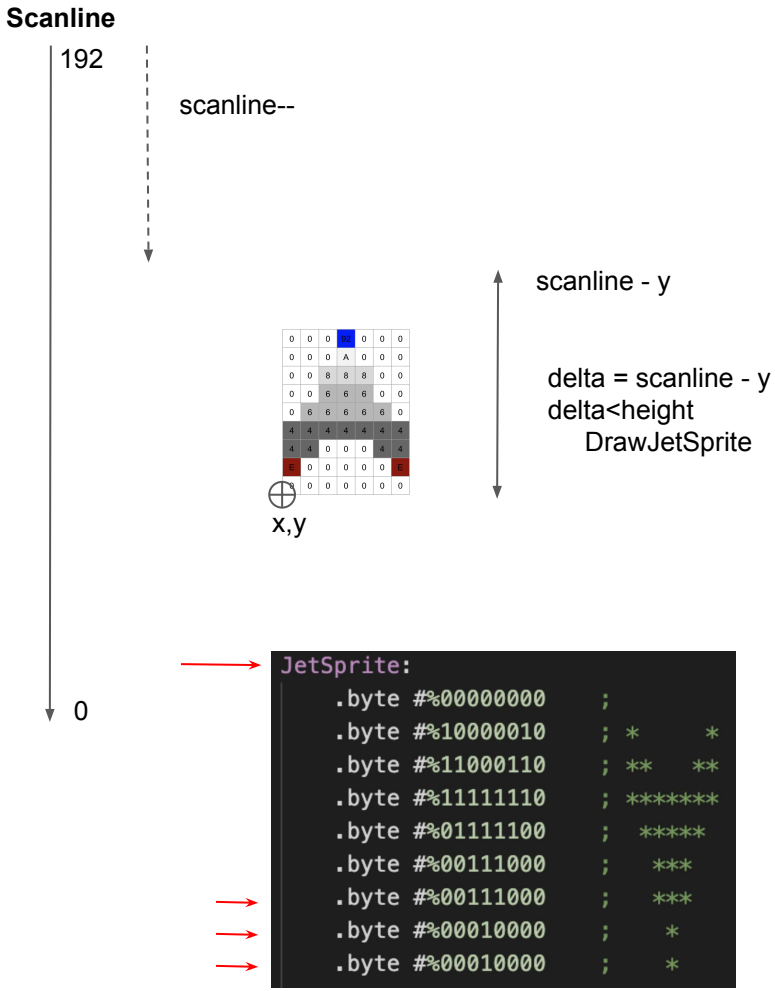
Pointers
Memory location⇒ 16-bit

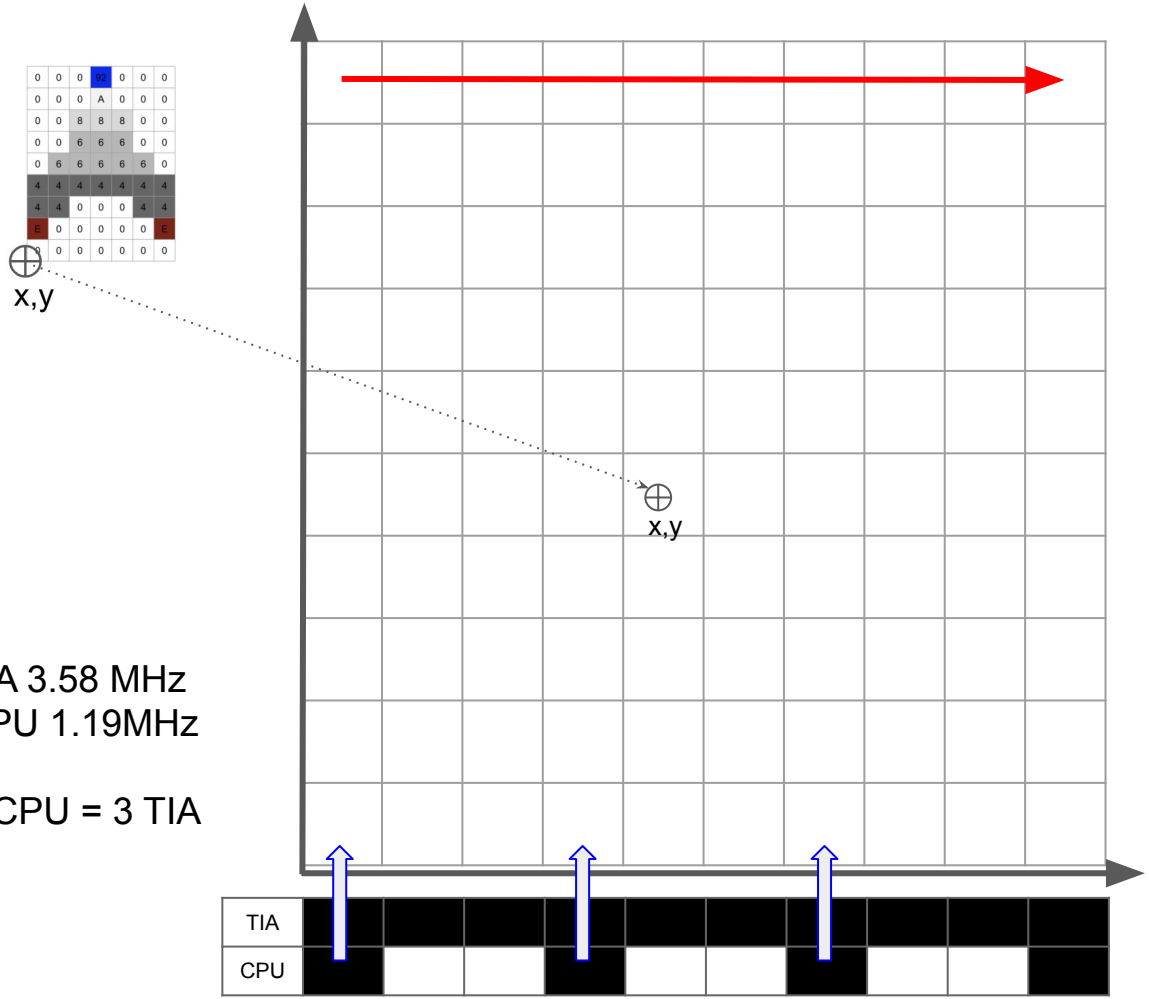
6507 little endian processor
In multi-byte, least significant byte is first

0	0	0	92	0	0	0
0	0	0	A	0	0	0
0	0	8	8	8	0	0
0	0	6	6	6	0	0
0	6	6	6	6	6	0
4	4	4	4	4	4	4
4	4	0	0	0	4	4
E	0	0	0	0	0	E
0	0	0	0	0	0	0

```
JetSprite:
    .byte #%00000000    ;
    .byte #%10000010    ; *     *
    .byte #%11000110    ; **    **
    .byte #%11111110    ; *******
    .byte #%01111100    ;  *****
    .byte #%00111000    ;   ***
    .byte #%00111000    ;   ***
    .byte #%00010000    ;    *
    .byte #%00010000    ;    *
```

```
    ldx #192
Scanline:

.InsideJetSprite:
    txa                 ; Scanline count (X) --> A
    sec                 ; set Carry Flag
    sbc JetYPos         ; A - PlayerYPos
    cmp JET_HEIGHT      ; w/i Height?
    bcc .DrawJetSprite  ; yes, then display bitmap
    lda #0              ; else, don't show

.DrawJetSprite:
    ; Bitmap
    clc
    tay
    lda (JetSpritePtr),Y ; load bitmap slice
    sta WSYNC
    sta GRP0        ; P0 graphics
    ; Color
    lda (JetColorPtr),Y
    sta COLUP0

    dex
    bne Scanline
```





TIA 3.58 MHz
CPU 1.19MHz

1 CPU = 3 TIA

Missed ⇒ Fine Grain setting
go back 2 positions

Want the remainder
X%3

```
.Div3Loop:
    sbc #3
    bcs .Div3Loop
```

sbc - 3 CPU cycles
bcs - 2 CPU cycles

5 CPU == 15 TIA cycles
X%15

```
.Div15Loop:
    sbc #15
    bcs .Div15Loop
```

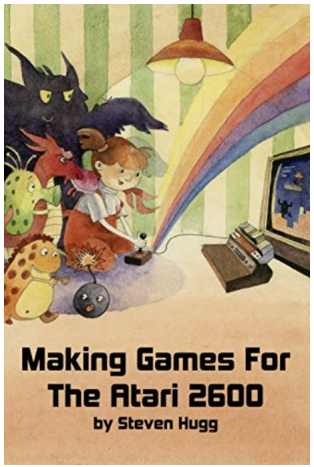
Live

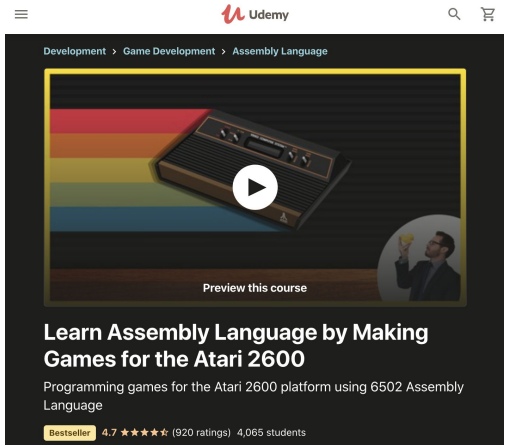
CODE

horizontalposition.asm

References

Playable [link](https://github.com/tromblee/UFBrownBag-11-2020)
https://github.com/tromblee/UFBrownBag-11-2020
https://8bitworkshop.com/v3.7.0/
https://www.randomterrain.com/atari-2600-memories.html#assembly_language
https://www.amazon.com/Making-Games-Atari-2600-Steven-ebook/dp/B01N4DSRIZ/ref=sr_1_1
https://www.udemy.com/course/programming-games-for-the-atari-2600/
https://www.slideserve.com/kane/design-of-the-first-action-adventure-video-game-adventure-for-the-atari-2600





Recap

As Builders we always have **constraints** (regardless of 1970's or 2020's)

The quality of our solutions depends on our

Proficiency of **Tools**
and
Understanding of **Environments**

Separations of Concerns is important
It significantly impacts the complexity of our environments

PRE-ORDER ATARI VCS 800 BLACK WALNUT ALL-IN BUNDLE

$389.99

Shipping 2020!
Technical Specifications >