tretton37

8 bit assembly in 20 minutes

KRISTOFFER JÄLÉN

tretton37

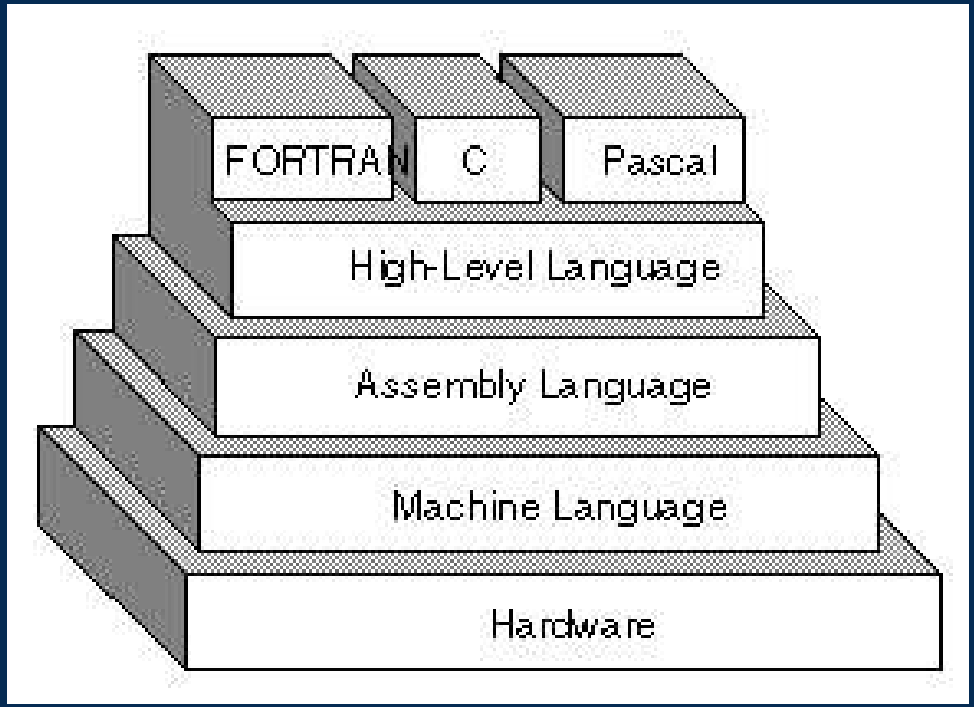
hello @simon.oskarsson!

tr∑tton37

_assembly language

- A low-level programming language
- Specific to a particular computer architecture
- Converted into executable machine code
- Uses a mnemonic to represent each low-level machine instruction or opcode

tretton37



ARM, MIPS, x86, Z80, 68000, 6502

6510

tretton37

_commodore 64?

- 8 bit computer
- Introduced in 1982
- Highest-selling computer model ever

tretton37

_what did it look like?

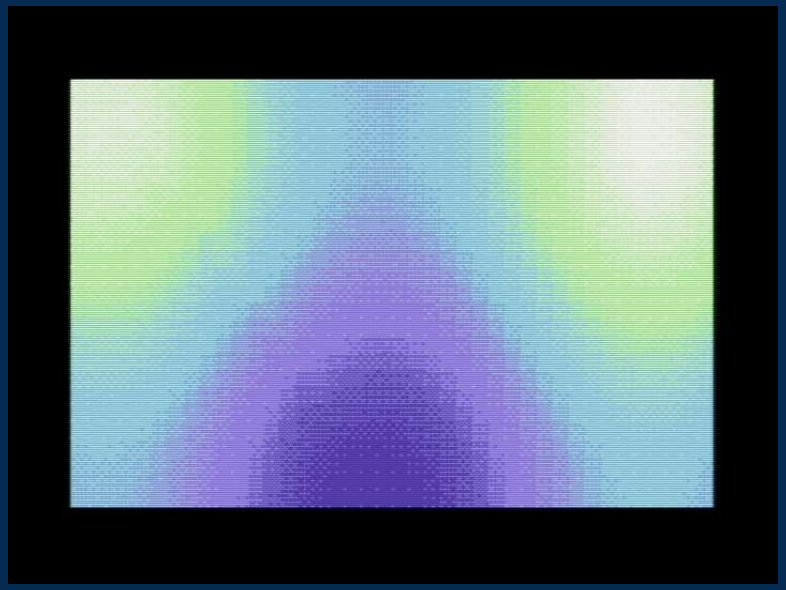

1UP 000125 TOP 000125 FALLS 04

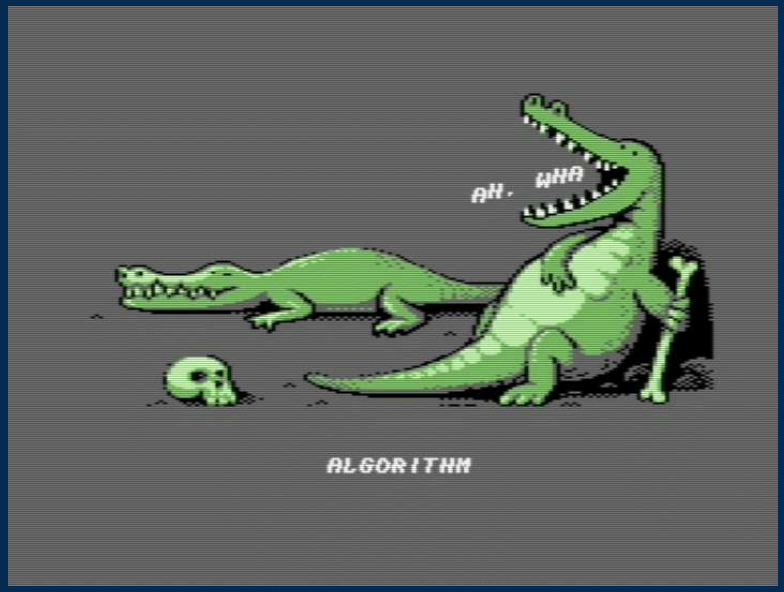

HOLLYWOOD
WERNER
OCEAN PACIFIC 1:15 0


EPYX


ENEMY
WOUNDS
USING
HOLDING
POWER
WEAPONRY
THE LAST NINJA

trtton37





tr3tton37

_memory-mapped i/o

- Each memory location corresponds to a **feature**
- Access features by **read/write** to memory addresses
- 64 kB RAM ($0000 - $ffff)

tr∑tton37

_memory-mapped i/o

$0400	Char at `y=1,x=1`
$0401	Char at `y=1,x=2`
$0428	Char at `y=2,x=1`
...	...
$d020	Border color (`$00-$0f`)

tretton37

the task

Output `Hello, world!` to screen

solution

- Screen coord = one memory location
- Write data to memory locations

tr∑tton37

instructions

- Three characters codes, e.g `LDA`, `JMP`
- Approx. 60 instructions
- Store data in memory and registers
- Increment/decrement
- Add/subtract
- Shift bits
- Jump and branch
- Compare

tr∑tton37

numeral system

One byte:

Decimal	`0` - `255`
Hexadecimal	`$00` - `$ff`
Binary	`%00000000` - `%11111111`

tretton37

registers

A (Accumulator)	Arithmetic and logic
X and Y	General purpose
S	Stack pointer
P	Processor status

Each register can hold one byte

tr∑tton37

store data

`STA $d020`	Store A in $d020
`LDA #$0f`	Load A with `$0f`
`STX $d020`	Store X in $d020
`LDX #$0f`	Load X with `$0f`
`STY $d020`	Store Y in $d020
`LDY #$0f`	Load Y with `$0f`

tr∑tton37

increment/decrement

`INC $d020`	Increase value in $d020 by 1
`DEC $d020`	Decrease value in $d020 by 1
`INX`	Increase value in X by 1
`DEX`	Decrease value in X by 1
`INY`	Increase value in Y by 1
`DEY`	Decrease value in Y by 1

tr3tton37

addressing modes

`LDA #$0f` Immediate addressing

`LDA $0f` Absolute addressing

```
lda #$0f      // Load A with value $0f (immediate)

lda $0f       // Load A with content of location $0f (absolute)
```

tr£tton37

indexed absolute addressing

Clear the screen

Screen is 40x25 chars ($0400-$07FF)

```
    lda #$20          // Character SPACE is $20
    ldx #$00          // Init counter to 0
loop:
    sta $0400,x       // Store A in $0400 + X
    sta $0500,x       // Store A in $0500 + X
    sta $0600,x       // Store A in $0600 + X
    sta $0700,x       // Store A in $0700 + X

    inx               // Increment X by 1

    bne loop          // Branch if X != 0, e.g. 256 iterations
```

tr3tton37

add and subtract

ADC Add with carry

```
lda $0400          // Read the byte value at $0400
clc                // No incoming carry; make sure carry is clear
adc #$05           // Add 5 to A
sta $0500          // Store A in $0500
```

SBC Subtract with carry

```
lda $0400          // Read the byte value at $0400
sec                // No incoming borrow; make sure carry is set
sbc #$05           // Subtract 5 from A
sta $0500          // Store A in $0500
```

tr2tton37

bit shifting

Shift bits to the left = multiply by two

Shift bits to the right = divide by two

ASL Bit shift left

```
lda #$02      // A = %0010 = $02
asl           // A = %0100 = $04
asl           // A = %1000 = $08
```

LSR Bit shift right

```
lda #$08      // A = %1000 = $08
lsr           // A = %0100 = $04
lsr           // A = %0010 = $02
```

tr∑tton37

bit masking

Set bits to 1

```
lda $d01d
ora %10000100     // Set bit 2 and 7
sta $d01d
```

Set bits to 0

```
lda $d01d
and %01111011     // Clear bit 2 and 7
sta $d01d
```

tr£tton37

branching

`CMP`	Compare memory and A
`BEQ` / `BNE`	Branch if equal / not equal

```
lda NumA        // Read value NumA
beq IsEqual     // Go to label IsEqual if NumA == 0
...             // Continue here if NumA != 0

lda NumA        // Read value NumA
cmp NumB        // Compare against NumB
beq IsEqual     // Go to label IsEqual if NumA == NumB
...             // Execution continues here if NumA != NumB
```

tr∑tton37

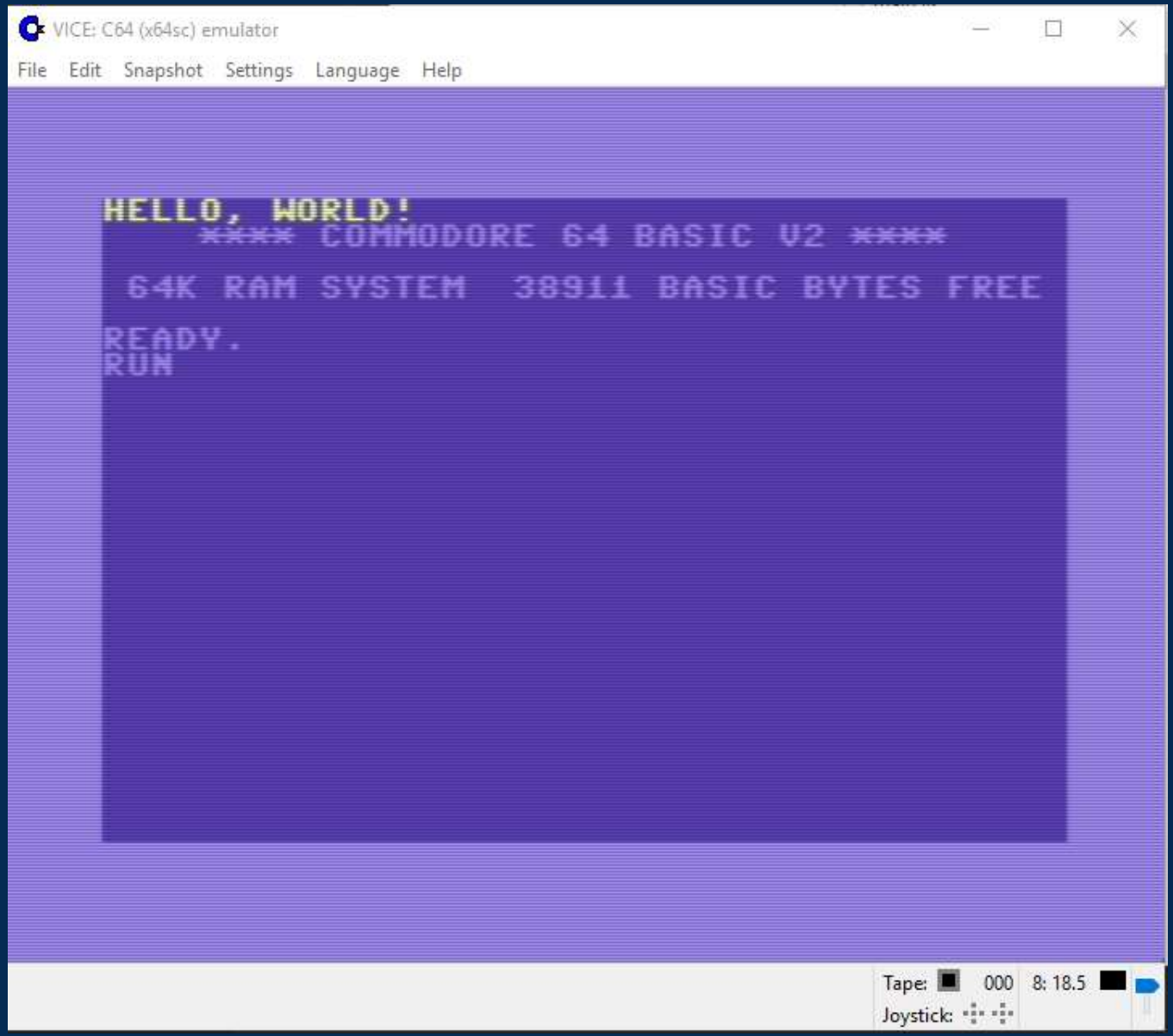
_rendering text

$0400 - $07e8	Screen memory
$d800 - $dbe7	Color memory

```
    ldx #$00
loop:
    lda message,x
    sta $0400,x     // Screen memory is at $0400-$07ff
    lda #$07        // $07 = yellow
    sta $d800,x     // Color memory is at $d800-$dbff
    inx
    cpx #13         // String length == 13
    bne loop
    jmp *           // Infinite loop

message:
    .text "hello, world!"
```

tretton37


VICE: C64 (x64sc) emulator
File Edit Snapshot Settings Language Help
HELLO, WORLD!
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY.
RUN
Tape: 000 8: 18.5
Joystick: