6502 Stack

Philipp Koehn

~~2 March 2018~~

20 Sept 2019

Midterm moved to
Friday Oct 4th

→ private msg to instructors on Piazza if you have a conflict

HW3 very soon
(later today)

c64 emulator

PEEK and POKE

- POKE: directly write into memory
- PEEK: directly read memory value
- Example: write into screen memory
  - POKE 1024,1
    writes letter A into top left corner
  - PRINT PEEK(1024)
    returns 1

Character Encoding in Screen Memory

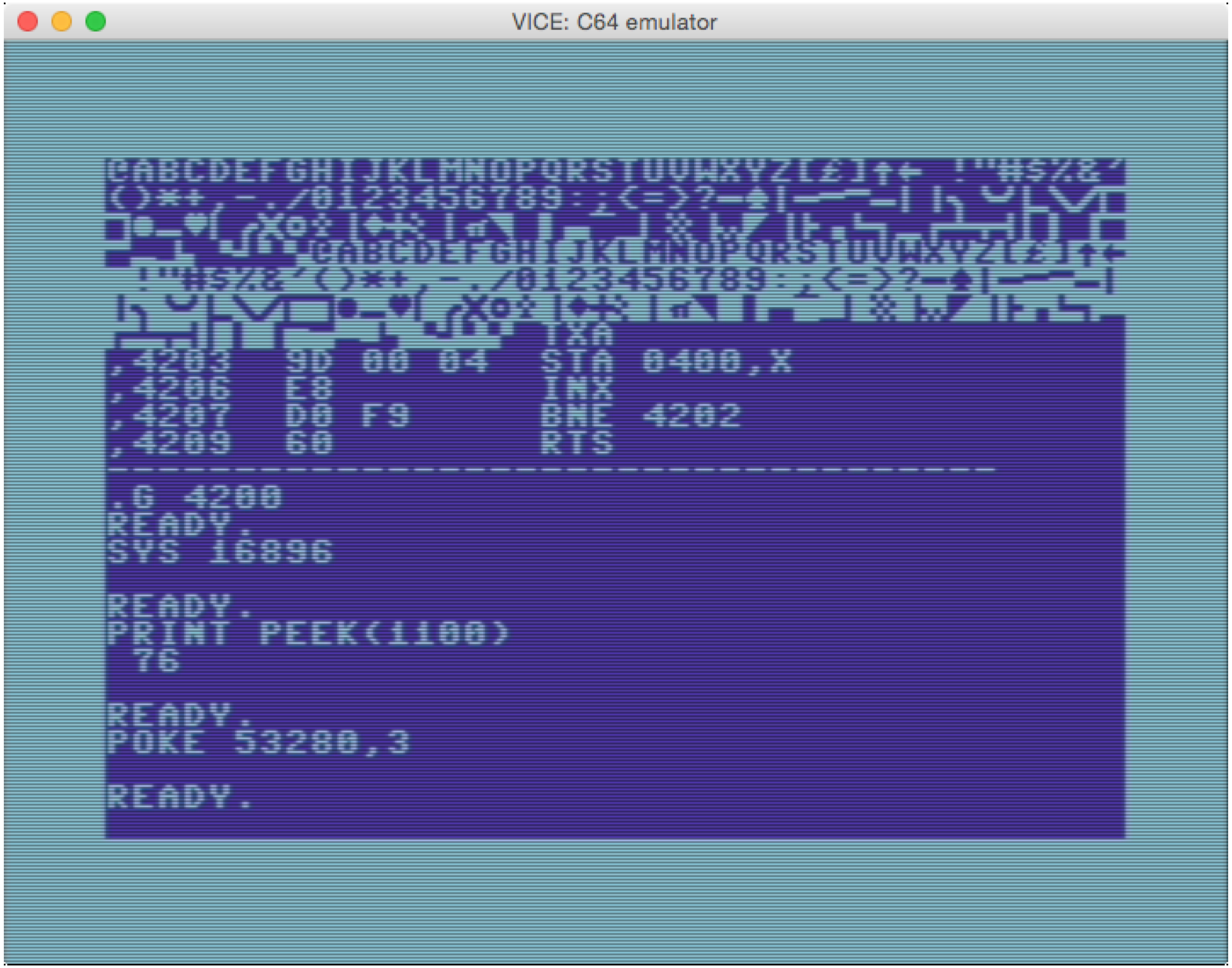
- What is the character encoding in screen memory?

- Let's write a program

Address	Bytes	Command
4200	A2 00	LDX #00
4202	8A	TXA
4203	9D 00 04	STA 0400,X
4206	E8	INX
4207	D0 F9	BNE 4202
4209	60	RTS

- Run from BASIC: SYS 16896

Screenshot



```
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[£]↑← !"#$%&'
()*+,-./0123456789:;<=>?
                   @ABCDEFGHIJKLMNOPQRSTUVWXYZ[£]↑←
 !"#$%&'()*+,-./0123456789:;<=>?
                                   TXA
,4203   9D 00 04    STA 0400,X
,4206   E8          INX
,4207   D0 F9       BNE 4202
,4209   60          RTS
-----------------------------------
.G 4200
READY.
SYS 16896

READY.
PRINT PEEK(1100)
 76

READY.
POKE 53280,3

READY.
```

stack

Stack

- Useful data structure

Stack

- Useful data structure
- LIFO: Last in, first out
    - PUSH 5

Stack

- Useful data structure
- LIFO: Last in, first out
  - PUSH 5
  - PUSH 2

Stack

- Useful data structure

- LIFO: Last in, first out
  - PUSH 5
  - PUSH 2
  - PULL → 2

a.k.a. "pop"

Stack

- Useful data structure

- LIFO: Last in, first out
  - PUSH 5
  - PUSH 2
  - PULL → 2
  - PULL → 5

6502 Stack in Memory

- 2nd page in memory reserved
  ("page" = 256 bytes)

- Stack pointer
  - current free position
  - an address, e.g., 01FF
  - register in CPU

⇒	01FF	
	01FE	
	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Stacks grow down

Example

- PUSH 0A

bottom

⇒	01FF	
	01FE	
	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- PUSH 0A
  - store 0A to 01FF
  - decrease stack pointer to 01FE

	01FF	0A
⇒	01FE	
	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- PUSH 0A
  - store 0A to 01FF
  - decrease stack pointer to 01FE
- PUSH 55

	01FF	0A
⇒	01FE	
	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- PUSH 0A
  - store 0A to 01FF
  - decrease stack pointer to 01FE

- PUSH 55
  - store 55 to 01FE
  - decrease stack pointer to 01FD

	01FF	0A
	01FE	55
⇒	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- PUSH 0A
  - store 0A to 01FF
  - decrease stack pointer to 01FE
- PUSH 55
  - store 55 to 01FE
  - decrease stack pointer to 01FD
- PULL

	01FF	0A
	01FE	55
⇒	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- PUSH 0A
  - store 0A to 01FF
  - decrease stack pointer to 01FE
- PUSH 55
  - store 55 to 01FE
  - decrease stack pointer to 01FD
- PULL
  - increase stack pointer to 01FE
  - retrieve 55 from 01FE

	01FF	0A
⇒	01FE	55
	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- PUSH 0A
  - store 0A to 01FF
  - decrease stack pointer to 01FE
- PUSH 55
  - store 55 to 01FE
  - decrease stack pointer to 01FD
- PULL
  - increase stack pointer to 01FE
  - retrieve 55 from 01FE
- PUSH 42

	01FF	0A
⇒	01FE	55
	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- PUSH 0A
  - store 0A to 01FF
  - decrease stack pointer to 01FE
- PUSH 55
  - store 55 to 01FE
  - decrease stack pointer to 01FD
- PULL
  - increase stack pointer to 01FE
  - retrieve 55 from 01FE
- PUSH 42
  - store 42 to 01FE
  - decrease stack pointer to 01FD

	01FF	0A
	01FE	42
⇒	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

6502 stack instructions

Basic Stack Manipulation

- Accumulator
  - PHA: push accumulator to stack
  - PLA: pull accumulator from stack

Basic Stack Manipulation

- Accumulator
  - PHA: push accumulator to stack
  - PLA: pull accumulator from stack

- Processor status (flags)
  - PHP: push processor status to stack
  - PLP: pull processor status from stack ← changes status flags

Example

- Stack is a good place to safely store register values
- Example

```
push A        PHA
push X        TXA      why not PHX?
              PHA         doesn't exist
push Y        TYA
              PHA
    (some code that changes registers)
              PLA      pop Y
              TAY
              PLA      pop X
              TAX
              PLA      pop A
  (all registers back to original state)
```

Stack Pointer Instructions

- Read out stack pointer
- TSX: transfer stack pointer to X register
- TXS: transfer X register to stack pointer

Warning

- Stack is not very big (256 bytes)
- Too heavy use may lead to stack overflow

sub routines

Subroutines

- Subroutines are small programs that do common things
  - for instance: write a character at current cursor position
  - this is in the C64 kernel at address FFD2

Subroutines

- Subroutines are small programs that do common things
  - for instance: write a character at current cursor position
  - this is in the C64 kernel at address FFD2

- Naive usage

```
LDA #41
JMP FFD2
```

- Why won't that work?

Subroutines

- Subroutines are small programs that do common things
  - for instance: write a character at current cursor position
  - this is in the C64 kernel at address FFD2

- Naive usage

```
LDA #41
JMP FFD2
```

- Why won't that work?

  Subroutine does not know where to return to

Solution

- Use the stack!

Solution

- Use the stack!

- Jump to subroutine
  - store current program counter to stack
  - jump to subroutine address

Solution

- Use the stack!

- Jump to subroutine
  - store current program counter to stack
  - jump to subroutine address

jsr addr

- Return from subroutine
  - retrieve return address from stack
  - jump to retrieved address

rts

6502 Subroutine Instructions

- JSR: Jump to subroutine

- RTS: Return from subroutine

Example

- `4400 LDA #41`
- `4402 JSR FFD2`

	01FF	0A
	01FE	55
⇒	01FD	
	01FC	
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
  - program counter is 4405
  - store program counter

	01FF	0A
	01FE	55
	01FD	44
	01FC	05
⇒	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
  - program counter is 4405
  - store programm counter

	01FF	0A
	01FE	55
	01FD	44
	01FC	05
⇒	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
    - program counter is 4405
    - store programm counter
- FFD2 JMP (0326)

	01FF	0A
	01FE	55
	01FD	44
	01FC	05
⇒	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
  - program counter is 4405
  - store programm counter
- FFD2 JMP (0326)
- F1CA ...

	01FF	0A
	01FE	55
	01FD	44
	01FC	05
⇒	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
  - program counter is 4405
  - store programm counter
- FFD2 JMP (0326)
- F1CA ...
- F207 RTS

	01FF	0A
	01FE	55
	01FD	44
	01FC	05
⇒	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
  - program counter is 4405
  - store programm counter
- FFD2 JMP (0326)
- F1CA ...
- F207 RTS

	01FF	0A
	01FE	55
	01FD	44
	01FC	05
⇒	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
  - program counter is 4405
  - store programm counter
- FFD2 JMP (0326)
- F1CA ...
- F207 RTS
  - retrieve program counter from stack
  - jump to retrieved address

	01FF	0A
	01FE	55
⇒	01FD	44
	01FC	05
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

Example

- 4400 LDA #41
- 4402 JSR FFD2
  - program counter is 4405
  - store programm counter
- FFD2 JMP (0326)
- F1CA ...
- F207 RTS
  - retrieve program counter from stack
  - jump to retrieved address
- 4405 ... back from subroutine

	01FF	0A
	01FE	55
⇒	01FD	44
	01FC	05
	01FB	
	01FA	
	01F9	
	01F8	
	01F7	
	01F6	
	01F5	
	01F4	
	01F3	
	01F2	
	...	...
	0100	

example

Recursion

- Recursively calling subroutines

- Canonical example: Fibonacci numbers

$f(0) = 0$
$f(1) = 1$

$x \geq 2$: `f(x) = f(x-1) + f(x-2)`

`0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...`

- $\lim_{x \to \infty} \frac{f(x+1)}{f(x)}$ is Golden Ratio parameter is X, return value is A

Code

```
START   TXA
        BNE M01       ; f(0) = 0?  no, continue
        RTS           ; yes
```

Code

```
START   TXA
        BNE M01     ; f(0) = 0?  no, continue
        RTS         ; yes
M01     DEX         ; prepare for f(x-1) call
        BNE M02     ; f(1) = 1?  no, continue
        RTS         ; yes
```

Code

```
START   TXA
        BNE M01     ; f(0) = 0?  no, continue
        RTS         ; yes
M01     DEX         ; prepare for f(x-1) call
        BNE M02     ; f(1) = 1?  no, continue
        RTS         ; yes
M02     TXA         ; save X on stack
        PHA
        JSR START
```

Code

```
START   TXA
        BNE M01     ; f(0) = 0?  no, continue
        RTS         ; yes
------------------------------------------------
M01     DEX         ; prepare for f(x-1) call
        BNE M02     ; f(1) = 1?  no, continue
        RTS         ; yes
------------------------------------------------
M02     TXA         ; save X on stack
        PHA
        JSR START   ; result of f(x-1) in accumulator
        TAY         ; let's put f(x-1) aside
```

Code

```
START   TXA
        BNE M01     ; f(0) = 0?  no, continue
        RTS         ; yes
M01     DEX         ; prepare for f(x-1) call
        BNE M02     ; f(1) = 1?  no, continue
        RTS         ; yes
M02     TXA         ; save X on stack
        PHA
        JSR START   ; result of f(x-1) in accumulator
        TAY         ; let's put f(x-1) aside
        PLA         ; get X back from stack
        TAX
```

Code

```
START   TXA
        BNE M01      ; f(0) = 0?  no, continue
        RTS          ; yes
M01     DEX          ; prepare for f(x-1) call
        BNE M02      ; f(1) = 1?  no, continue
        RTS          ; yes
M02     TXA          ; save X on stack
        PHA
        JSR START    ; result of f(x-1) in accumulator
        TAY          ; let's put f(x-1) aside
        PLA          ; get X back from stack
        TAX
        TYA          ; get f(x-1) back
        PHA          ; save that for now on stack
```

Code

```
START   TXA
        BNE M01      ; f(0) = 0?  no, continue
        RTS          ; yes
M01     DEX          ; prepare for f(x-1) call
        BNE M02      ; f(1) = 1?  no, continue
        RTS          ; yes
M02     TXA          ; save X on stack
        PHA
        JSR START    ; result of f(x-1) in accumulator
        TAY          ; let's put f(x-1) aside
        PLA          ; get X back from stack
        TAX
        TYA          ; get f(x-1) back
        PHA          ; save that for now on stack
        DEX          ; prepare f(x-2)
        JSR START
```

Code

```
START   TXA
        BNE M01      ; f(0) = 0?  no, continue
        RTS          ; yes

M01     DEX          ; prepare for f(x-1) call
        BNE M02      ; f(1) = 1?  no, continue
        RTS          ; yes

M02     TXA          ; save X on stack
        PHA
        JSR START    ; result of f(x-1) in accumulator
        TAY          ; let’s put f(x-1) aside
        PLA          ; get X back from stack
        TAX
        TYA          ; get f(x-1) back
        PHA          ; save that for now on stack
        DEX          ; prepare f(x-2)
        JSR START
        STA TEMP     ; store f(x-2) for addition
```

Code

```
START   TXA
        BNE M01      ; f(0) = 0?  no, continue
        RTS          ; yes
M01     DEX          ; prepare for f(x-1) call
        BNE M02      ; f(1) = 1?  no, continue
        RTS          ; yes
M02     TXA          ; save X on stack
        PHA
        JSR START    ; result of f(x-1) in accumulator
        TAY          ; let's put f(x-1) aside
        PLA          ; get X back from stack
        TAX
        TYA          ; get f(x-1) back
        PHA          ; save that for now on stack
        DEX          ; prepare f(x-2)
        JSR START
        STA TEMP     ; store f(x-2) for addition
        PLA          ; f(x-1) from stack
```

Code

```
START   TXA
        BNE M01     ; f(0) = 0?  no, continue
        RTS         ; yes
M01     DEX         ; prepare for f(x-1) call
        BNE M02     ; f(1) = 1?  no, continue
        RTS         ; yes
M02     TXA         ; save X on stack
        PHA
        JSR START   ; result of f(x-1) in accumulator
        TAY         ; let's put f(x-1) aside
        PLA         ; get X back from stack
        TAX
        TYA         ; get f(x-1) back
        PHA         ; save that for now on stack
        DEX         ; prepare f(x-2)
        JSR START
        STA TEMP    ; store f(x-2) for addition
        PLA         ; f(x-1) from stack
        CLC
        ADC TEMP    ; f(x-1) + f(x-2)
        RTS
```

some more instructions

Logic

- Standard Boolean operations
  - AND: bitwise AND
  - ORA: bitwise OR
  - EOR: bitwise XOR

- Operations impact negative and zero flag

- BIT: bitwise AND, but do not store result

Compare

- Compare register value
  - CMP: compare accumulator
  - CPX: compare X register
  - CPY: compare Y register
- Does not change register value
- Sets flags as in a subtraction
  (e.g., if values match, set zero flag)

some quirky things

Decimal Mode

- Decimal mode: pretend that hex numbers are really decimal numbers

```
LDA #07
CLC
ADC #04
```

- Normally results in 0B

Decimal Mode

- Decimal mode: pretend that hex numbers are really decimal numbers

```
LDA #07
CLC
ADC #04
```

- Normally results in 0B

- But in decimal mode: result 11

Decimal Mode

- Decimal mode: pretend that hex numbers are really decimal numbers

```
LDA #07
CLC
ADC #04
```

- Normally results in 0B
- But in decimal mode: result 11
- Instructions
  - SED: set decimal mode
  - CLD: clear decimal mode

NOP

- NOP: No Operation

- Does nothing

- Useful as filler

  e.g., when deleting instructions