6502 Introduction

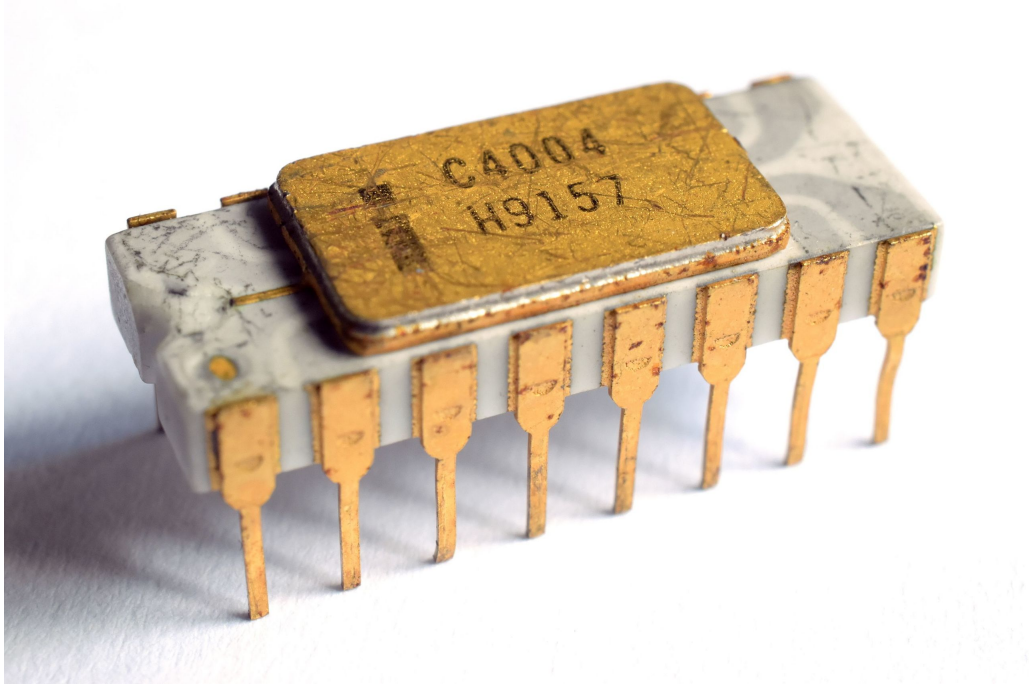
Philipp Koehn

28 February 2018

some history

1971

- First microprocessor on an integrated circuit: Intel 4004



- 4-bit central processing unit, 12 bit address space (4KB)

1975

- MOS Technology 6502

- Dominant CPU in home computers for a decade
  (Atari, Apple II, Nintendo Entertainment System, Commodore PET)

1977

- Atari 2600

- Video game console: Pong, Pac Man, ... connected to TV

1980

- Commodore VIC20

- 1 MHz, 5KB RAM, BASIC, 3.5KB RAM, 176x184 3 bit color video

1982

- Commodore C64

- 64KB RAM, 320x200 4 bit color video

Commodore C64

- BASIC programming language, but serious programs written in assembly
- No fancy stuff like multi-process, user accounts, virtual memory, etc.
- Machine itself had no mass storage - had to buy tape drive, then floppy disk drive, machine was obsolete once hard drives came around

BASIC Demo

- Commands get executed (just like Python interpreter)

```
PRINT "HELLO WORLD"
HELLO WORLD
```

- Program with line numbers

```
10 PRINT "HELLO WORLD"
20 GOTO 10
```

- List program

```
LIST
```

- Execute program

```
RUN
```

- Another example (takes about 1 second to run)

```
20 FOR I = 1 TO 1000
30 NEXT
```

6502 specification

6502 Specification

- 8-bit processor, using 16 bit address space (up to 64KB RAM)
- 3 registers: accumulator, X register, Y register
- Status register: contains flags
- Operating system in ROM (read only memory)
- Stack -- more on that later
- Interrupts -- more on that later

Assembly Code Instructions

- Load and store from A, X, and Y register
- Transfer between registers
- Arithmetric: add, subtract, increment, decrement
- Shift and rotate, e.g., 00001111 $\rightarrow$ 00011110
- Logic: AND and OR
- Compare and test
- Branch (conditional jump)
- Set and clear flag values
- Jump and subroutines
- Interrupt: cause interrupt, return from interrupt
- Stack operations

Memory Organization

0000-00ff Zero page: used for variables

0100-01ff Stack

0200-03ff More variables [C64]

0400-07ff Screen memory (characters) [C64]

0800-9fff BASIC RAM [C64]

a000-bfff BASIC ROM [C64]

c000-cffff Upper RAM Area [C64]

d000-dfff Character shape ROM / Video and audio RAM [C64]

e000-ffff Kernel ROM [C64]

Can switch to RAM under ROM

Load and Store

- 3 Registers: Accumulator, X, Y
- Load from memory: LDA, LDX, LDY
- Store to memory: STA, STX, STY

Addressing Modes

- Immediate: load specified value
  LDA #$22 $\rightarrow$ accumulator has now value $22 (hex)

Addressing Modes

- Immediate: load specified value
  LDA #$22 $\rightarrow$ accumulator has now value $22 (hex)

- Absolute: load value from specified address
  LDA $D010 $\rightarrow$ accumulator has now value store in memory position $D010

Addressing Modes

- Immediate: load specified value
  LDA #$22 $\rightarrow$ accumulator has now value $22 (hex)

- Absolute: load value from specified address
  LDA $D010 $\rightarrow$ accumulator has now value store in memory position $D010

- Zero page: as above, but for memory addresses 0000-00FF
  LDA $6A $\rightarrow$ accumulator has now value store in memory position $006A

Addressing Modes

- Immediate: load specified value
  LDA #$22 → accumulator has now value $22 (hex)

- Absolute: load value from specified address
  LDA $D010 → accumulator has now value store in memory position $D010

- Zero page: as above, but for memory addresses 0000-00FF
  LDA $6A → accumulator has now value store in memory position $006A

- Relative: relative to current program counter
  BCC $06 → jump 6 memory positions forward, if carry flag clear

Indexed Addressing Modes

- X and Y registers can be used as indexes for memory lookup

Indexed Addressing Modes

- X and Y registers can be used as indexes for memory lookup

- Indexed with X register
  - example: LDA $0400,X
  - add value of register X to $0400 (say, X=$05 $\rightarrow$ $0405)
  - load value from that memory position ($0405)

Indexed Addressing Modes

- X and Y registers can be used as indexes for memory lookup
- Indexed with X register
  - example: LDA $0400,X
  - add value of register X to $0400 (say, X=$05 $\rightarrow$ $0405)
  - load value from that memory position ($0405)
- Variants: Y register, zero page

Indexed Addressing Modes

- X and Y registers can be used as indexes for memory lookup

- Indexed with X register
  - example: LDA $0400,X
  - add value of register X to $0400 (say, X=$05 → $0405)
  - load value from that memory position ($0405)

- Variants: Y register, zero page

- Zero Page Indexed Indirect
  - example: LDA ($15,X)
  - add value of register X to $15 (say, X=$02 → $0017)
  - treat resulting memory position as pointer
    (say, $0017 contains $E0, $0018 contains $FF)
  - load value from that address ($FFE0)

Transfer Between Registers

- 3 Registers: Accumulator, X, Y
- Transfer from Accumulator: TAX, TAY
- Transfer to Accumulator: TXA, TXY
- Note: no TXY, TYX

Arithmetic

- Addition (to accumulator): ADC
  - ADC #$02 → add 2 to accumulator
  - ADC $4050 → add value in memory at address $4050 to accumulator
- Subtraction (from accumulator): SBC
- Increment by 1: INC, INX, INY
- Decrement by 1: DEC, DEX, DEY
- Sets carry, overflow, zero flag

Flags

- Carry: set iff
  - addition/increase results in value >255
  - subtraction/decrease results in value <0

Flags

- Carry: set iff
  - addition/increase results in value >255
  - subtraction/decrease results in value <0
- Overflow (V): same under assumption that numbers are signed

Flags

- Carry: set iff
  - addition/increase results in value >255
  - subtraction/decrease results in value <0
- Overflow (V): same under assumption that numbers are signed
- Zero: set iff result of operation/load/transfer is 0

Flags

- Carry: set iff
  - addition/increase results in value >255
  - subtraction/decrease results in value <0
- Overflow (V): same under assumption that numbers are signed
- Zero: set iff result of operation/load/transfer is 0
- Negative: set iff result of operation/load/transfer sets bit 7

Flags

- Carry: set iff
  - addition/increase results in value >255
  - subtraction/decrease results in value <0
- Overflow (V): same under assumption that numbers are signed
- Zero: set iff result of operation/load/transfer is 0
- Negative: set iff result of operation/load/transfer sets bit 7
- Other flags: Break, Interrupt, Decimal (more on these later)

Flags

- Carry: set iff
  - addition/increase results in value >255
  - subtraction/decrease results in value <0
- Overflow (V): same under assumption that numbers are signed
- Zero: set iff result of operation/load/transfer is 0
- Negative: set iff result of operation/load/transfer sets bit 7
- Other flags: Break, Interrupt, Decimal (more on these later)
- Clear flags: CLC, CLV, CLI, CLD
- Set flags: SEC, SED, SEI

Example Program

```
Address   Bytes       Command
  4000    65 1C       (data:  number 1)
  4002    A0 9E       (data:  number 2)
  4004    00 00       (data:  sum)
  4006    AD 00 40    LDA 4000
  4009    18          CLC
  400A    6D 02 40    ADC 4002
  400D    8D 04 40    STA 4004
  4010    AD 01 40    LDA 4001
  4013    6D 03 40    ADC 4003
  4016    8D 05 40    STA 4005
  4019    00          BRK
```

Example Program

```
Address   Bytes       Command
  4000    65 1C       (data:  number 1)
  4002    A0 9E       (data:  number 2)
  4004    00 00       (data:  sum)
  4006    AD 00 40    LDA 4000
  4009    18          CLC
  400A    6D 02 40    ADC 4002
  400D    8D 04 40    STA 4004
  4010    AD 01 40    LDA 4001
  4013    6D 03 40    ADC 4003
  4016    8D 05 40    STA 4005
  4019    00          BRK
```

16 bit addition

Branch

- Simple jump: JMP

- Flags can be used for conditional jump ("branch")

```
BCC  Branch if carry flag clear
BCS  Branch if carry flag set
BEQ  Branch if zero flag set
BMI  Branch if negative flag set
BNE  Branch if zero flag clear
BPL  Branch if negative flag clear
BVC  Branch if overflow flag clear
BVS  Branch if overflow flag set
```

Shift and Rotate

- Rotate bits by one position
  - ROL: Rotate left, i.e., 11110000 $\rightarrow$ 11100001
  - ROR: Rotate right, i.e., 11110000 $\rightarrow$ 01111000

Shift and Rotate

- Rotate bits by one position
  - ROL: Rotate left, i.e., 11110000 → 11100001
  - ROR: Rotate right, i.e., 11110000 → 01111000

- ASL (Arithmetric Shift Left) /
  LSR (Logical Shift Right) use carry bit
  - ASL: 11110000 (C=0) → 1110000 (C=1)
  - LSR: 11110000 (C=1) → 11111000 (C=0)

Example: Multiplication

- Elementary school multiplication:

10101 x 1101

Example: Multiplication

- Elementary school multiplication:

```
      10101 x 1101
    ----------------
      10101
```

Example: Multiplication

- Elementary school multiplication:

```
      10101 x 1101
-----------------
      10101
         0
```

Example: Multiplication

- Elementary school multiplication:

```
      10101 x 1101
-----------------
     10101
        0
   10101
```

Example: Multiplication

- Elementary school multiplication:

```
      10101 x 1101
-----------------
     10101
        0
   10101
  10101
```

Example: Multiplication

- Elementary school multiplication:

```
        10101 x 1101
    ----------------
        10101
           0
      10101
     10101
    ----------------
    100010001
```

(in decimal: 23x13 = 299)

Example: Multiplication

- Elementary school multiplication:

```
      10101 x 1101
-----------------
     10101
        0
   10101
  10101
-----------------
100010001
```

(in decimal: 23x13 = 299)

- Idea
  - shift second operand to right (get last bit)
  - if carry: add first operand to sum
  - rotate first operand to left (multiply with binary 10)

Code

```
Address   Bytes       Command
  4100    03          (data:  number 1)
  4101    06          (data:  number 2)
  4102    00          (data:  product)
  4103    A9 00       LDA #00
  4105    A2 08       LDX #08
  4107    4E 01 41    LSR 4101
  410A    90 00 41    BCC 4110
  410C    18          CLC
  410D    6D 00 41    ADC 4100
  4110    2E 00 41    ROL 4100
  4113    CA          DEX
  4114    D0 07 41    BNE 4107
  4116    8D 02 41    STA 4102
  4119    00          BRK
```