some history
1971

- First microprocessor on an integrated circuit: Intel 4004

- 4-bit central processing unit, 12 bit address space (4KB)
• 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
• 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
• Arithmetic: 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]
0800-09ff 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 → 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 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 from Accumulator: TXA, TXY

• Note: no TXY, TYX
Arithmetic

• Addition (to accumulator): ADC
  – ADC #$02 \rightarrow \text{add 2 to accumulator}
  – ADC $4050 \rightarrow \text{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

- 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

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

16 bit addition
• 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., \(11110000 \rightarrow 01111000\)

- **ASL** (Arithmetic Shift Left) / **LSR** (Logical Shift Right) use carry bit
  - **ASL**: \(11110000 \text{ (C=0)} \rightarrow 1110000 \text{ (C=1)}\)
  - **LSR**: \(11110000 \text{ (C=1)} \rightarrow 11111000 \text{ (C=0)}\)
Example: Multiplication

• Elementary school multiplication:

\[
\begin{array}{c}
10101 	imes 1101 \\
\hline
10101 \\
0 \\
10101 \\
10101 \\
\hline
100010001
\end{array}
\]

(in decimal: 23 \times 13 = 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

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