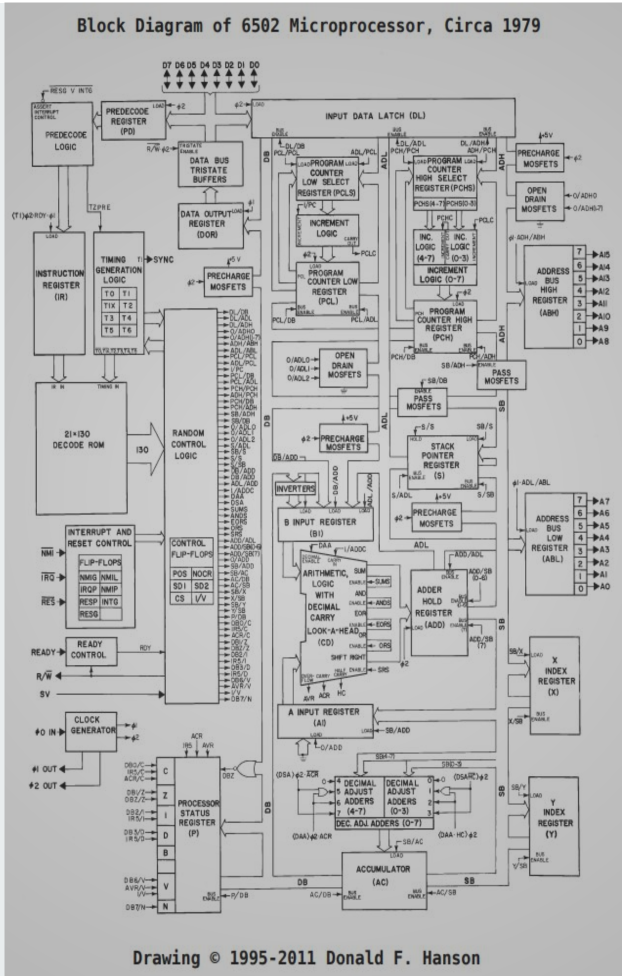
Emu6502

65c02 emulator in Forth

2023.01 update

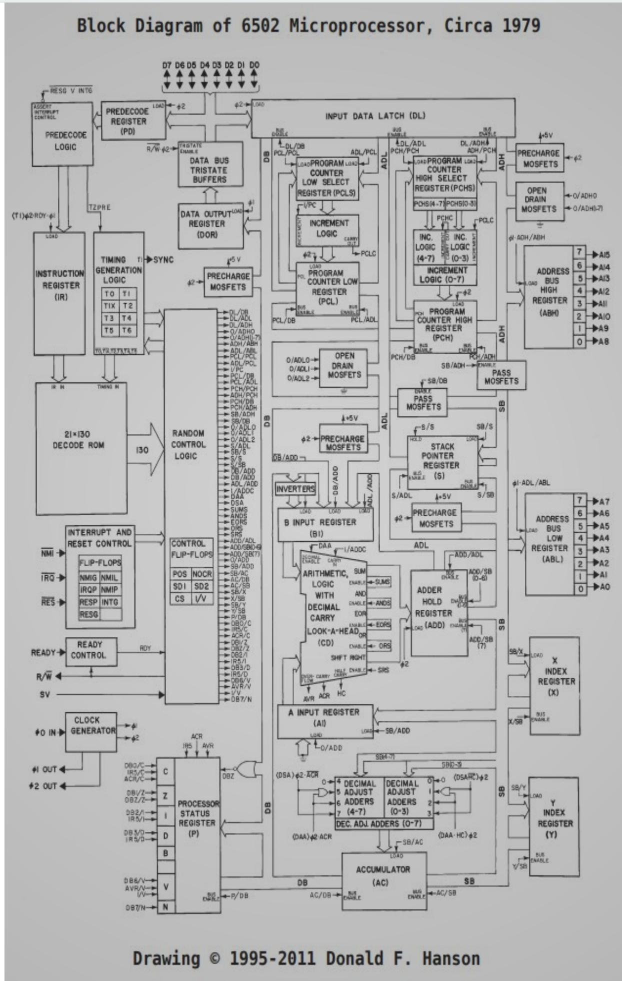
Alexandre Dumont
@adumont

#FORTH2020
Jan 21st, 2023


Block Diagram of 6502 Microprocessor, Circa 1979
INPUT DATA LATCH (DL)
PREDECODE REGISTER (PD)
PREDECODE LOGIC
DATA BUS TRISTATE BUFFERS
DATA OUTPUT REGISTER (DOR)
INSTRUCTION REGISTER (IR)
TIMING GENERATION LOGIC
PROGRAM COUNTER LOW SELECT REGISTER (PCLS)
PROGRAM COUNTER HIGH SELECT REGISTER (PCHS)
INCREMENT LOGIC
PROGRAM COUNTER LOW REGISTER (PCL)
PROGRAM COUNTER HIGH REGISTER (PCH)
PRECHARGE MOSFETS
OPEN DRAIN MOSFETS
ADDRESS BUS HIGH REGISTER (ABH)
ADDRESS BUS LOW REGISTER (ABL)
21×130 DECODE ROM
RANDOM CONTROL LOGIC
PASS MOSFETS
STACK POINTER REGISTER (S)
INVERTERS
B INPUT REGISTER (BI)
ARITHMETIC, LOGIC WITH DECIMAL CARRY LOOK-A-HEAD (CD)
ADDER HOLD REGISTER (ADD)
INTERRUPT AND RESET CONTROL
CONTROL FLIP-FLOPS
READY CONTROL
CLOCK GENERATOR
A INPUT REGISTER (AI)
X INDEX REGISTER (X)
Y INDEX REGISTER (Y)
PROCESSOR STATUS REGISTER (P)
DECIMAL ADJUST ADDERS (4-7)
DECIMAL ADJUST ADDERS (0-3)
ACCUMULATOR (AC)
Drawing © 1995-2011 Donald F. Hanson

Content

- Introduction
- Overall Approach
- Main components
- Instructions definitions
- More Advanced stuff
- Demo


Block Diagram of 6502 Microprocessor, Circa 1979
INPUT DATA LATCH (DL)
PREDECODE REGISTER (PD)
PREDECODE LOGIC
DATA BUS TRISTATE BUFFERS
DATA OUTPUT REGISTER (DOR)
INSTRUCTION REGISTER (IR)
TIMING GENERATION LOGIC
21×130 DECODE ROM
RANDOM CONTROL LOGIC
INTERRUPT AND RESET CONTROL
READY CONTROL
CLOCK GENERATOR
PROCESSOR STATUS REGISTER (P)
PROGRAM COUNTER LOW SELECT REGISTER (PCLS)
INCREMENT LOGIC
PROGRAM COUNTER LOW REGISTER (PCL)
PROGRAM COUNTER HIGH SELECT REGISTER (PCHS)
PROGRAM COUNTER HIGH REGISTER (PCH)
STACK POINTER REGISTER (S)
B INPUT REGISTER (BI)
A INPUT REGISTER (AI)
ARITHMETIC, LOGIC WITH DECIMAL CARRY LOOK-A-HEAD (CD)
ADDER HOLD REGISTER (ADD)
DECIMAL ADJUST ADDERS
ACCUMULATOR (AC)
X INDEX REGISTER (X)
Y INDEX REGISTER (Y)
ADDRESS BUS HIGH REGISTER (ABH)
ADDRESS BUS LOW REGISTER (ABL)
PRECHARGE MOSFETS
OPEN DRAIN MOSFETS
PASS MOSFETS
Drawing © 1995-2011 Donald F. Hanson

Introduction

- Emu6502 is a **65C02 emulator written in FORTH**
    - Development started in late Dec. 2022
    - Started in AlexForth (on 6502) then moved to gForth
    - Idea is to adapt to ESP32Forth next

- **Objective:** run 65C02 binary rom
    - It runs my AlexForth and AlexMon (monitor) binary roms!

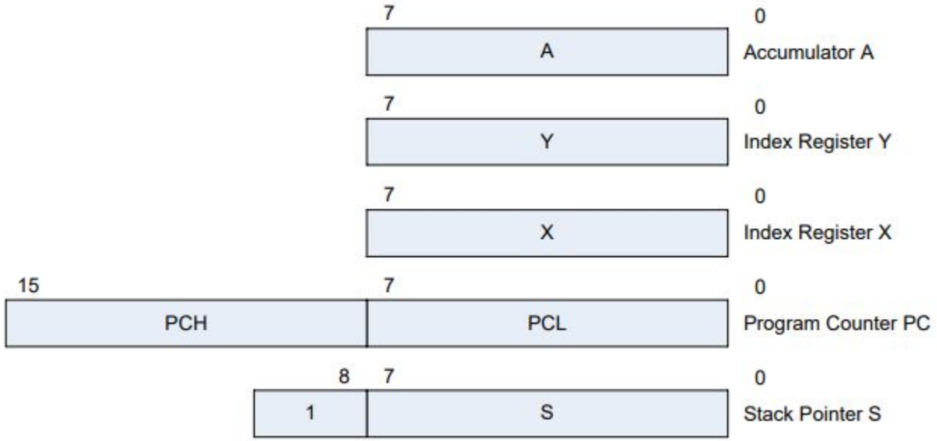
Overall approach

- Use the WDC 65C02 datasheet as the reference
- Replicate the 6502 main components in Forth
- Start with simple instructions like `lda` and `sta`
- Use a very systematic and mechanical approach
    - Critical for the instruction words definition
    - Grouping: same instructions with several addressing modes
    - Helps generating clean code
- Don't try to factor too early so I can see patterns emerge clearly

Emu6502 main components

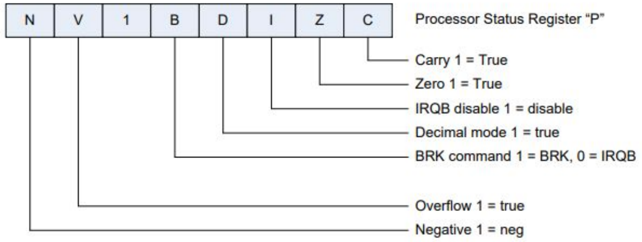
6502 registers

- Registers, A, X, Y, SP, P and PC as Forth variables:



```
CREATE _A    0  C,
CREATE _X    0  C,
CREATE _Y    0  C,
CREATE _SP   0  C,
CREATE _PC   0   ,
CREATE _P  $30  C,
```

6502 processor flags



```
\ -- Processor Flags masks
%10000000 VALUE 'N    \ Negative flag
%01000000 VALUE 'V    \ Overflow flag
%00010000 VALUE 'B    \ Break flag
%00001000 VALUE 'D    \ Decimal flag.
%00000100 VALUE 'I    \ Interrupt disabled
%00000010 VALUE 'Z    \ Zero flag
%00000001 VALUE 'C    \ Carry flag

: CLEAR ( mask -- ) NOT _P C@ AND _P C! ;
: SET   ( mask -- )     _P C@ OR  _P C! ;
: UPDATE-FLAG ( b/f mask -- ) SWAP IF SET ELSE CLEAR THEN ;
```

```
: >N   ( b -- b ) DUP          'N AND 'N UPDATE-FLAG ; \ non-droppy
: >Z   ( b -- b ) DUP $FF AND     0= 'Z UPDATE-FLAG ; \ non-droppy
: >V   ( f --   )                    'V UPDATE-FLAG ; \ droppy
: >C   ( f --   )                    'C UPDATE-FLAG ; \ droppy
: >NZ >N >Z ;

: C>   (   -- f ) _P C@ 'C AND ;
: D>   (   -- f ) _P C@ 'D AND ;
```

Target Memory

- Target memory is a 64KB array
- Initial version (without mapped IO):

```
CREATE RAM $10000 ALLOT \ Full 6502 memory 64KB space

\ Target RAM operations words
: TC@ ( addr -- byte ) RAM + C@ ;
: TC! ( byte addr -- ) RAM + C! ;

: T@  ( addr -- word )
  DUP TC@         \ LO
  SWAP 1+ TC@     \ HI
  $100 * + \ LO HI --> HILO  ;
```

Program Counter

- PC register points to the next instruction opcode

```
CREATE _PC   0   ,
: _PC! ( addr -- ) $FFFF AND _PC ! ;

\ Fetch a BYTE and advance PC by 1
: BYTE@ ( -- byte )
  _PC @ DUP 1+ _PC! TC@
;

\ Fetch a WORD and advance PC by 2
: WORD@ ( -- byte )
  _PC @ DUP 2+ _PC! T@
;
```

Opcodes table and **Processor cycle**

- The 65C02 has 212 valid 1-byte opcodes
- I allocate an array of 256 cells, to store the XT of up to 256 Forth words:

```
CREATE OPCODES #256 CELLS ALLOT
```

- The opcode is the index into the array

- A **processor cycle** now becomes clear:
  - Fetch 1 byte at PC
  - Decode the instruction
  - Execute the corresponding word

```
: NEXT
  ( FETCH   ) BYTE@
  ( DECODE  ) CELLS OPCODES + @
  ( EXECUTE ) EXEC
;
```

Instruction definitions approach

Defining instructions and **binding to opcodes**

- `BIND`: assigns a word to an opcode:

```
: BIND ( xt opcode -- )   CELLS OPCODES + ! ; \ saves XT in OPCODES table
```

- Example (simplest instruction): `NOP`, opcode is $EA

```
:NONAME ( NOP ) ;  $EA  BIND
```

Defining LDA and STA

- LDA immediate ($A9)

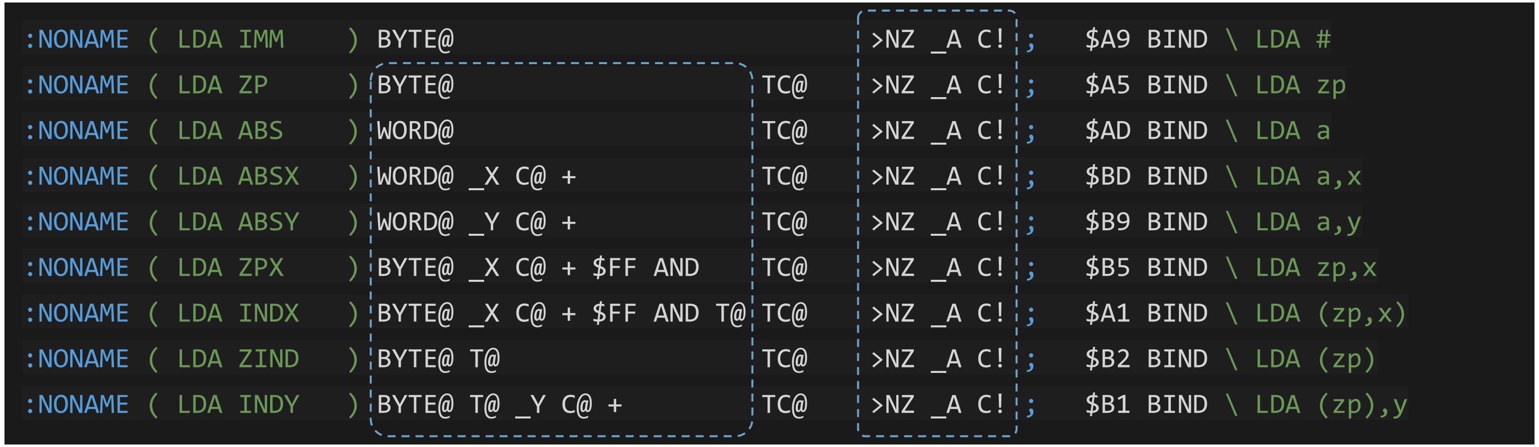
```
\ A9 XX                         Fetch byte |  Set flags   | Store in A    | Store the XT in OPCODES table
:NONAME ( LDA IMM       )          BYTE@          >NZ             _A C!     ; ( xt )   $A9 BIND
```

- STA absolute ($8D)

```
\ 8D LO HI                         Get A    |  Fetch Addr | Store to Mem | Store the XT in OPCODES table
:NONAME ( STA ABS       )           _A C@        WORD@                TC!     ; ( xt )   $8D BIND
```

Addressing modes

- The WDC 65C02 has 16 addressing modes.
  - Not all instructions use the 16 modes
- Example of `LDA`:



```
:NONAME ( LDA IMM    ) BYTE@                           >NZ _A C! ;   $A9 BIND \ LDA #
:NONAME ( LDA ZP     ) BYTE@                    TC@    >NZ _A C! ;   $A5 BIND \ LDA zp
:NONAME ( LDA ABS    ) WORD@                    TC@    >NZ _A C! ;   $AD BIND \ LDA a
:NONAME ( LDA ABSX   ) WORD@ _X C@ +            TC@    >NZ _A C! ;   $BD BIND \ LDA a,x
:NONAME ( LDA ABSY   ) WORD@ _Y C@ +            TC@    >NZ _A C! ;   $B9 BIND \ LDA a,y
:NONAME ( LDA ZPX    ) BYTE@ _X C@ + $FF AND    TC@    >NZ _A C! ;   $B5 BIND \ LDA zp,x
:NONAME ( LDA INDX   ) BYTE@ _X C@ + $FF AND T@ TC@    >NZ _A C! ;   $A1 BIND \ LDA (zp,x)
:NONAME ( LDA ZIND   ) BYTE@ T@                 TC@    >NZ _A C! ;   $B2 BIND \ LDA (zp)
:NONAME ( LDA INDY   ) BYTE@ T@ _Y C@ +         TC@    >NZ _A C! ;   $B1 BIND \ LDA (zp),y
```

- Patterns start to emerge that show opportunities for factoring

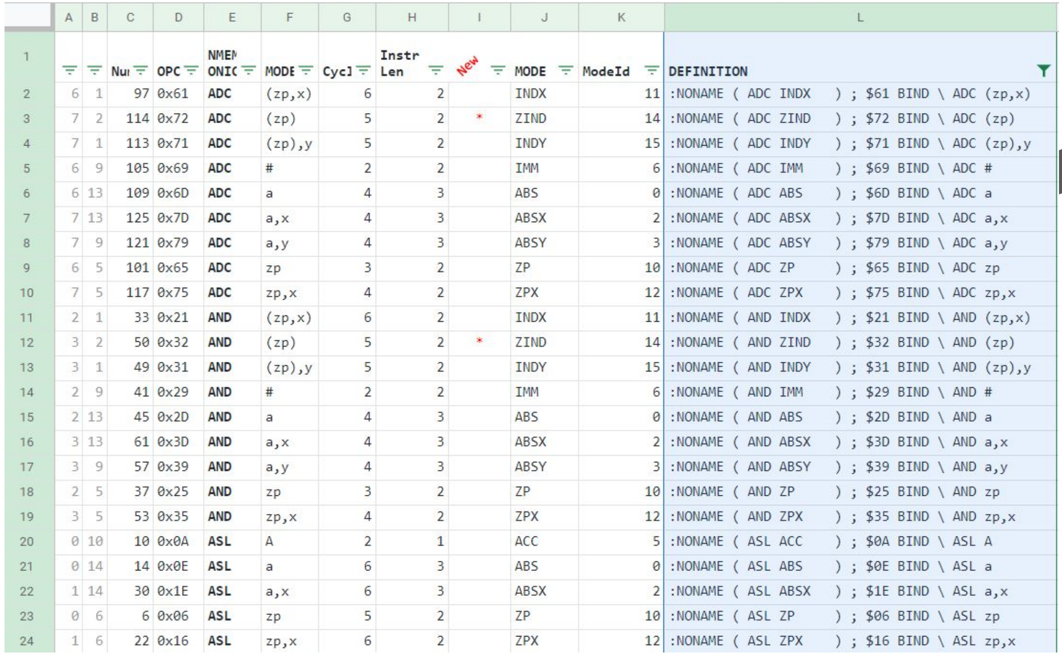
LDA and STA after factoring

```
\ Addressing modes words
: 'ZP   ( -- addr ) BYTE@ ;                          \ zp
: 'ABS  ( -- addr ) WORD@ ;                          \ Absolute a
: 'ABSX ( -- addr ) WORD@ _X C@ + ;                  \ a,x
: 'ABSY ( -- addr ) WORD@ _Y C@ + ;                  \ a,y
: 'ZPX  ( -- addr ) BYTE@ _X C@ + $FF AND ;          \ zp,x
: 'ZPY  ( -- addr ) BYTE@ _Y C@ + $FF AND ;          \ zp,y
: 'INDX ( -- addr ) BYTE@ _X C@ + $FF AND T@ ;       \ (zp,x)
: 'ZIND ( -- addr ) BYTE@ T@ ;                       \ (zp)
: 'INDY ( -- addr ) BYTE@ T@ _Y C@ + ;               \ (zp), y
```

```
: LDA ( byte -- ) >NZ _A C! ;
:NONAME ( LDA IMM     ) BYTE@       LDA ; $A9 BIND \ LDA #
:NONAME ( LDA ABS     ) 'ABS   TC@  LDA ; $AD BIND \ LDA a
:NONAME ( LDA ZP      ) 'ZP    TC@  LDA ; $A5 BIND \ LDA zp
:NONAME ( LDA ABSX    ) 'ABSX  TC@  LDA ; $BD BIND \ LDA a,x
:NONAME ( LDA ABSY    ) 'ABSY  TC@  LDA ; $B9 BIND \ LDA a,y
:NONAME ( LDA ZPX     ) 'ZPX   TC@  LDA ; $B5 BIND \ LDA zp,x
:NONAME ( LDA INDX    ) 'INDX  TC@  LDA ; $A1 BIND \ LDA (zp,x)
:NONAME ( LDA ZIND    ) 'ZIND  TC@  LDA ; $B2 BIND \ LDA (zp)
:NONAME ( LDA INDY    ) 'INDY  TC@  LDA ; $B1 BIND \ LDA (zp),y
```

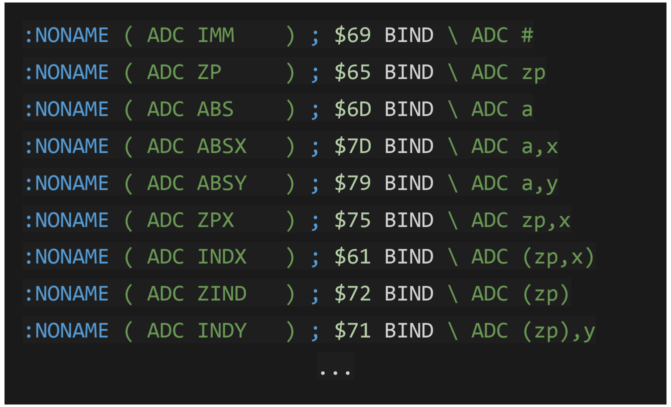
```
: STA ( addr -- ) _A C@ SWAP TC! ;
:NONAME ( STA ABS     ) 'ABS        STA ; $8D BIND \ STA a
:NONAME ( STA ZP      ) 'ZP         STA ; $85 BIND \ STA zp
:NONAME ( STA ABSX    ) 'ABSX       STA ; $9D BIND \ STA a,x
:NONAME ( STA ABSY    ) 'ABSY       STA ; $99 BIND \ STA a,y
:NONAME ( STA ZPX     ) 'ZPX        STA ; $95 BIND \ STA zp,x
:NONAME ( STA INDX    ) 'INDX       STA ; $81 BIND \ STA (zp,x)
:NONAME ( STA ZIND    ) 'ZIND       STA ; $92 BIND \ STA (zp)
:NONAME ( STA INDY    ) 'INDY       STA ; $91 BIND \ STA (zp),y
```

Accelerated code template generation



	A	B	C	D	E	F	G	H	I	J	K	L
1			Nu	OPC	NMEM ONIC	MODE	Cycl	Instr Len	New	MODE	ModeId	DEFINITION
2	6	1	97	0x61	ADC	(zp,x)	6	2		INDX	11	:NONAME (ADC INDX) ; $61 BIND \ ADC (zp,x)
3	7	2	114	0x72	ADC	(zp)	5	2	*	ZIND	14	:NONAME (ADC ZIND) ; $72 BIND \ ADC (zp)
4	7	1	113	0x71	ADC	(zp),y	5	2		INDY	15	:NONAME (ADC INDY) ; $71 BIND \ ADC (zp),y
5	6	9	105	0x69	ADC	#	2	2		IMM	6	:NONAME (ADC IMM) ; $69 BIND \ ADC #
6	6	13	109	0x6D	ADC	a	4	3		ABS	0	:NONAME (ADC ABS) ; $6D BIND \ ADC a
7	7	13	125	0x7D	ADC	a,x	4	3		ABSX	2	:NONAME (ADC ABSX) ; $7D BIND \ ADC a,x
8	7	9	121	0x79	ADC	a,y	4	3		ABSY	3	:NONAME (ADC ABSY) ; $79 BIND \ ADC a,y
9	6	5	101	0x65	ADC	zp	3	2		ZP	10	:NONAME (ADC ZP) ; $65 BIND \ ADC zp
10	7	5	117	0x75	ADC	zp,x	4	2		ZPX	12	:NONAME (ADC ZPX) ; $75 BIND \ ADC zp,x
11	2	1	33	0x21	AND	(zp,x)	6	2		INDX	11	:NONAME (AND INDX) ; $21 BIND \ AND (zp,x)
12	3	2	50	0x32	AND	(zp)	5	2	*	ZIND	14	:NONAME (AND ZIND) ; $32 BIND \ AND (zp)
13	3	1	49	0x31	AND	(zp),y	5	2		INDY	15	:NONAME (AND INDY) ; $31 BIND \ AND (zp),y
14	2	9	41	0x29	AND	#	2	2		IMM	6	:NONAME (AND IMM) ; $29 BIND \ AND #
15	2	13	45	0x2D	AND	a	4	3		ABS	0	:NONAME (AND ABS) ; $2D BIND \ AND a
16	3	13	61	0x3D	AND	a,x	4	3		ABSX	2	:NONAME (AND ABSX) ; $3D BIND \ AND a,x
17	3	9	57	0x39	AND	a,y	4	3		ABSY	3	:NONAME (AND ABSY) ; $39 BIND \ AND a,y
18	2	5	37	0x25	AND	zp	3	2		ZP	10	:NONAME (AND ZP) ; $25 BIND \ AND zp
19	3	5	53	0x35	AND	zp,x	4	2		ZPX	12	:NONAME (AND ZPX) ; $35 BIND \ AND zp,x
20	0	10	10	0x0A	ASL	A	2	1		ACC	5	:NONAME (ASL ACC) ; $0A BIND \ ASL A
21	0	14	14	0x0E	ASL	a	6	3		ABS	0	:NONAME (ASL ABS) ; $0E BIND \ ASL a
22	1	14	30	0x1E	ASL	a,x	6	3		ABSX	2	:NONAME (ASL ABSX) ; $1E BIND \ ASL a,x
23	0	6	6	0x06	ASL	zp	5	2		ZP	10	:NONAME (ASL ZP) ; $06 BIND \ ASL zp
24	1	6	22	0x16	ASL	zp,x	6	2		ZPX	12	:NONAME (ASL ZPX) ; $16 BIND \ ASL zp,x

I used a spreadsheet with all the 65C02 instructions, with their **opcodes** and **addressing modes**, to **quickly generate an empty template** code for all the 212 instructions, **just waiting to be defined.**



```
:NONAME ( ADC IMM    ) ; $69 BIND \ ADC #
:NONAME ( ADC ZP     ) ; $65 BIND \ ADC zp
:NONAME ( ADC ABS    ) ; $6D BIND \ ADC a
:NONAME ( ADC ABSX   ) ; $7D BIND \ ADC a,x
:NONAME ( ADC ABSY   ) ; $79 BIND \ ADC a,y
:NONAME ( ADC ZPX    ) ; $75 BIND \ ADC zp,x
:NONAME ( ADC INDX   ) ; $61 BIND \ ADC (zp,x)
:NONAME ( ADC ZIND   ) ; $72 BIND \ ADC (zp)
:NONAME ( ADC INDY   ) ; $71 BIND \ ADC (zp),y
                      ...
```

More advance stuff

Handling I/O

- 6502 I/O is memory mapped
- We can easily hook into `TC@` / `TC!` to add support for IO devices
- Example: simple char input/output

```
$F004 CONSTANT IN_CHAR
$F001 CONSTANT OUT_CHAR

\ Target RAM operations
: TC@ ( addr -- byte ) DUP IN_CHAR  = IF DROP GETC EXIT THEN RAM + C@ ;
: TC! ( byte addr -- ) DUP OUT_CHAR = IF DROP EMIT EXIT THEN RAM + C! ;
```

`NEXT`, `RUN` , `BREAKPOINT`

- Load a 65(C)02 rom into target memory
- Run step by step (`NEXT`) or call `RUN` to run up to the next breakpoint
- By default breakpoint is when code run a `BRK`. You can also define custom breakpoints (`BREAKPOINT` is a deferred word):

```
\ Ex. breakpoint on A=1
:NONAME ( -- flag ) _A C@ 1 = ; IS BREAKPOINT
```

Demo

Limits / Whats next

Limitations

- No interrupt support for the moment (`STP`, `WAI`, IRQ/NMI)
- Not clock-cycle accurate

Possible ideas for future updates

- Adapt to **ESP32Forth** (and basic IO/GPIO capability)
- Add some **interrupts** mechanism
- Develop a GUI? (when running in Gforth)
  - Step by step
  - Viewing/editing registers
  - Viewing/editing memory content,...

Thank you!

Links

Emu6502 repository on Github: https://github.com/adumont/emu6502

My web page with links to all my projects (and these slides): https://adumont.github.io/

Interact with me on Twitter: @adumont https://twitter.com/adumont

Forth2020 meetings archive, recordings and how to join us:
https://github.com/forth2020/zoom-presentations

Alexandre Dumont
@adumont