NES Emulator

Release 0.1

Matyas Daniel

Aug 16, 2024

CONTENTS

This documentation details my implementation of an NES. However, as of now, only the CPU is in a usable condition.

The goal of this project is to develop a VHDL code which can be synthetised on an arbitrary FPGA chip to emulate the behavior of an NES.

CHAPTER

ONE

CENTRAL PROCESSING UNIT

The CPU of the NES was developed by Ricoh and it is called **RP2A03**. This chip is almost an identical copy of the CPU **6502** made by MOS Technology.

Binary Coded Decimal mode is not supported on 2A03 because 5 transistors were removed from the original 6502. One can still set and clear the decimal flag, but it will not have an effect electrically. This was almost certainly done because of copyright rules, since MOS Technology filed a patent for their parallel decimal number adder. [Inc75] [Qui11]

I do not have to worry about copyright laws (hopefully), so the goal of the project initially is just to implement a working 6502 on an FPGA board.

An important addition of Ricoh to the 6502 chip is the Audio Processing Unit (APU). The 2A03 has 4 internal sound channels which are able to generate semi-analog sound, and it has a sound channel capable of playing samples based on the memory map or using the 7-bit unsigned Pulse Code Modulator (PCM) directly.

Sound channels:

- 2 rectangle wave internal channels
- 1 triangle wave internal channel
- 1 random wavelength internal channel
- 1 external channel [Tay04]

1.1 6502 CPU

The 6502 microprocessor is a relatively simple 8 bit CPU with only a few internal registers capable of addressing at most 64Kb of memory via its 16 bit address bus. The processor is little endian and expects addresses to be stored in memory least significant byte first.

An instruction is executed by the processor in the following way: address of the next instruction is sent on the address bus and instruction is fetched. The ID (Instruction Decoder) decodes the instruction. Instruction is executed making use of the internal registers and the ALU (Arithmetic Logical Unit).

1.1.1 Arithmetic Logical Unit

This is a subunit of the CPU which executes arithmetic and logical operations. The input of the ALU is changed on rising edge of clock. The operations are executed immediately when a change in input is detected.

The flags from PSR are separated and returned only if the respective operation affects a given flag.

- c = Carry flag. Set if bit 8 in the extended result is set (overflow in bit 7).
- z = Zero flag. Set if alu_reg_out = 0.
- v = oVerflow flag. Set if signed overflow occurs, i.e., if adding two positive numbers results in a negative number, or adding two negative numbers results in a positive number.
- n = Negativity flag. Set if bit 7 is set.

alu_op	alu_reg_out	Affected flags	Description
ADC	alu_reg_a + alu_reg_b + c_in	c,z,v,n	Add with carry
ADD	alu_reg_a + alu_reg_b	c	This addition cannot ba accessed directly. Only *Microcode* can access it.
AND_OP	alu_reg_a & alu_reg_b	z,n	AND
CMP	alu_reg_a - alu_reg_b	c,z,n	CoMPare registers. The flags indicate which of the operands is greater.
EOR	alu_reg_a ^ alu_reg_b	z,n	Exclusive OR
LDA, LDX, TAX, TSX, TXA, TXS	alu_reg_b	z,n	LoaD or Transfer data
ORA	alu_reg_a | alu_reg_b	z,n	OR
SBC	alu_reg_a - alu_reg_b + (1 - c_in)	c,z,v,n	SuBtract with Carry
ASL	alu_reg_a << 1	c,z,n	Arithmetic Shift Left
LSR	alu_reg_a >> 1	c,z,n	Logical Shift Right. Carry will take the previous value of bit 0 in alu_reg_a.
ROL_OP	(alu_reg_a << 1) | c_in	c,z,n	ROtate Left
ROR_OP	(alu_reg_a >> 1) | (c_in << 7)	c,z,n	ROtate Right
DEC, DEX	alu_reg_a - 1	z,n	DECrement
INC	alu_reg_a + 1	z,n	INCrement
STA, STX, NOP	—	—	STore and No-OPeration. Do nothing.

1.1.2 Instruction decoder

The instructions are divided into 3 parts:

OOOA.AAGG

- O = operation
- A = addressing mode
- G = group

Masks are used to separate the respective parts and they are analyzed separately.

Group

There are three groups:

- 00 = group 3
- 01 = group 1
- 10 = group 2
- 11 = invalid instruction

Group 1 instructions are executed on the Accumulator and have the following addressing modes: Immediate, Zero Page, Zero Page Indexed by X, Absolute, Absolute Indexed by X, Absolute Indexed by Y, Indexed Indirect by X, Indirect Indexed by Y.

Group 2 has the following addressing modes: Zero Page, Zero Page Indexed (sometimes X and sometimes Y), Absolute, Absolute Indexed (sometimes X and sometimes Y). Most of these instructions are executed on memory, i.e., data is fetched from memory and it is written back to the same location. Group 2 can be divided two subgroups. **Group 2A** instructions include the shift and rotate operations, and these have a new addressing mode: Accumulator addressing, which operates directly on the Accumulator. Some of **group 2B** instructions are executed on register X. The common thing in them is that they lack Accumulator addressing. In group 2B some group 3-ish instructions (they do not have the required addressing modes by group 2) intervene for undefined addressing mode masks. These use Implied addressing.

Group 3 includes every remaining instruction. A commonality between these operations cannot be concluded.

Implementation note: From group 3 only JMP is implemented.

Addressing Mode

Addressing modes are detailed in another section.

Addressing mask	Group 1	Group 2A	Group 2B	Group 2-extra	Group 3
000	IDX_IND	—	—	—	
001	ZERO_PG	ZERO_PG	ZERO_PG	—	
010	IMM	A	—	IMPLIED	
011	ABSOLUTE	ABSOLUTE	ABSOLUTE	—	ABSOLUTE, IND_ABS (JMP); …
100	IND_IDX	—	—	—	
101	ZERO_PG	ZERO_PG	ZERO_PG_Y **if** op = (STX or LDX) **else** ZERO_PG_X	—	
110	ABS_IDX	—	—	IMPLIED	
111	ABS_IDX	ABS_IDX	ABS_IDX_Y **if** op = STX **else** ABS_IDX_Y	—	

Operation

Operation mask	Group 1	Group 2A	Group 2B	Group 2-extra	Group 3
000	ORA	ASL	—	—	
001	AND	ROL_OP	—	—	
010	EOR	LSR	—	—	JMP (ABSOLUTE), …
011	ADC	ROR_OP	—	—	JMP (IND_ABS), …
100	STA	—	STX	TXA, TXS	
101	LDA	—	LDX	TAX, TSX	
110	CMP	—	DEC	DEX	
111	SBC	—	INC	NOP	

[MOS Technology, Inc.76b]

1.1.3 Memory

The memory is structured into pages of 256 bytes. The first 256 byte page of memory ($0000-$00FF) is referred to as 'Zero Page' and is the focus of a number of special addressing modes that result in shorter (and quicker) instructions or allow indirect access to the memory. The second page of memory ($0100-$01FF) is reserved for the system stack. The only other reserved locations in the memory map are the very last 6 bytes of memory $FFFA to $FFFF which must be programmed with the addresses of the non-maskable interrupt handler ($FFFA/B), the power on reset location ($FFFC/D) and the BRK/interrupt request handler ($FFFE/F) respectively.

The 6502 does not have any special support of hardware devices so they must be mapped to regions of memory in order to exchange data with the hardware latches.

1.1.4 Registers

1. **Accumulator (reg_a)**: It is the main register of this CPU. By default the instructions use the accumulator.
2. **Index registers (reg_index_x, reg_index_y)**: These registers are used in indirect addressing, adding their value to a given address. There is a limited number of instructions which can operate directly on these registers: load, store, increment, decrement, exchange data.
3. **Program counter (reg_pc)**: This register always points to the next instruction to be executed. Originally this register was a pair of 8-bit registers, however, in the code a 16-bit register is used.
4. **Stack pointer (sp)**: Contains the low order byte on Page One where the stack is located. The reset value of this register is 0xFF. Every time data is pushed on the stack, the register is decremented. Every time data is pulled from the stack the register is incremented.
5. **Processor status register (reg_ps)**: This indicates the status of the CPU. Every single bit from this registers is a flag and is independent of the other bits.
    - BIT 7: N = negativity flag
    - BIT 6: V = signed overflow flag
    - BIT 5: - = not used
    - BIT 4: B = break command *(not implemented)*
    - BIT 3: D = decimal mode *(not implemented)*
    - BIT 2: I = interrupt DISABLE *(not implemented)*
    - BIT 1: Z = zero flag

- BIT 0: C = carry flag

6. **Instruction register (reg_i)**: This where the instruction op-code is stored temporarily after it is fetched. This is fed directly to the instruction decoder (ID). [Car84]

Implementation note: The exact execution of the instructions on *Microcode* level is not diclosed, so 1 additional register is used as buffer to the *Arithmetic Logical Unit* input (alu_reg_a) and 1 register to the output (alu_reg_out). Using these registers easy modification of ALU input and temporary storage of ALU ouput during microcode is possible.

The other register input of the ALU is connected directly to the *data bus*.

1.1.5 Instructions

There are 56 possible instructions. A detailed description of every one of them can be found at *Instruction decoder* documentation.

The instructions can be cathegorized into 3 groups.

The instructions can have 1, 2 or 3 bytes. The *first* byte always specifies the type of instruction and the addressing mode. The *second* byte can be a constant which is directly used, or it can be the low part of the address. The *third* byte is the high part of the address.

1.1.6 Addressing modes

When an instruction is executed the operands and the destination of the result is determined by the addressing mode. There are several addressing modes and the *state machine* is mostly dependent on these:

<table>
<tr><th>1-byte</th></tr>
<tr><td><b>Accumulator mode addressing (A)</b> : This is an implied addressing mode unique to the shift and rotate instructions. The data in the accumulator will be directly modified.
<i>Example</i>
<table>
<tr><th>Address</th><th>Data</th><th>Dissassembly</th><th>Result</th></tr>
<tr><td>$0010</td><td>2A</td><td>ROL A; A = 10</td><td>reg_a = 0x20</td></tr>
</table>
</td></tr>
<tr><td><b>Immediate addressing mode (IMM)</b>: Execute the instruction with the constant, that is, the second byte of the instruction.
<i>Example</i>
<table>
<tr><th>Address</th><th>Data</th><th>Dissassembly</th><th>Result</th></tr>
<tr><td>$0010</td><td>a9</td><td rowspan="2">LDA #$01</td><td rowspan="2">reg_a = 0x01</td></tr>
<tr><td>$0011</td><td>01</td></tr>
</table>
</td></tr>
<tr><td><b>Implied addressing (IMPLIED)</b>: The operand and destination is implied by the instruction. In this type of addressing neither an operand, nor a destination is necessarily specified.
<i>Example</i>
<table>
<tr><th>Address</th><th>Data</th><th>Dissassembly</th><th>Result</th></tr>
<tr><td>$0010</td><td>18</td><td>CLC</td><td>C = 0</td></tr>
</table>
</td></tr>
</table>

2-byte

Relative addressing mode (REL): It is used in branch instructions. The program counter points to the next instruction. If the condition for branching is satisfied, the second byte of the instruction is added to the program counter or subtracted from the program counter if bit 7 is set (using 2's complement).

Example 1

Address	Data	Dissassembly	Result

$0110	90	BCC $50	**if C = 1 :**
			reg_pc = 0112
			else :
			reg_pc = 0162

$0111	50		
$0001	01		

Example 2

Address	Data	Dissassembly	Result

$0110	90	BCC $C0	**if C = 1 :**
			reg_pc = 0112
			else :
			reg_pc = 00D2

$0111	C0		
$0001	01		

Zero page addressing mode (ZERO_PG): The operand is from page zero and only the low address byte specified.

Example

Address	Data	Dissassembly	Result
$0010	A5	LDA $01	reg_a = 0x01
$0011	01		
$0001	01		

Zero page indexed X and Y addressing mode (ZERO_PG_IDX): The low byte of the address from page zero is fetched and the value of X or Y is added to it. The result cannot cross the page. It will wrap around.

Example

Address	Data	Dissassembly	Result
$0010	B5	LDA $F3,X ; X = 33	reg_a = 0x01
$0011	F3		
$00F3	XX		
$0026	01		

Indirect indexed addressing mode (IND_IDX): In this mode the second byte indicates the memory address in page zero, where the low address byte is stored. The high address byte is stored sequetially at the next address. After constructing an absolute address **register Y** is added to it. From the resulting address the operand will be fetched.

Example

Address	Data	Dissassembly	Result
$0010	B1	LDA ($F3),Y ; Y = 33	reg_a = 0x01
$0011	F3		
$00F3	34		
$00F4	12		
$1234	XX		
$1267	01		

2-byte

Indexed indirect addressing mode (IDX_IND): In this mode the **register X** is added to the second byte. The location indicated by the result on page zero will contain the low address byte of the operand. The high address byte of the operand can be fetched from the next address sequentially.
Example

Address	Data	Dissassembly	Result
$0010	A1	LDA ($F3,X) ; X = 33	reg_a = 0x01
$0011	F3		
$00F3	XX		
$0026	12		
$0027	34		
$3412	01		

3-byte

Absolute addressing mode (ABSOLUTE): The operand is extracted from an absolute (16-bit) address from memory.
Example

Address	Data	Dissassembly	Result
$0010	AD	LDA $1123	reg_a = 0x01
$0011	23		
$0012	11		
$1123	01		

Indirect absolute addressing mode (IND_ABS): This is a subtype of absolute addressing, used only by JMP instruction. 2 sequential bytes at the provided absolute address are taken as low byte and high byte of an address. The resulting address points to the low and high byte of the program counter.
Example

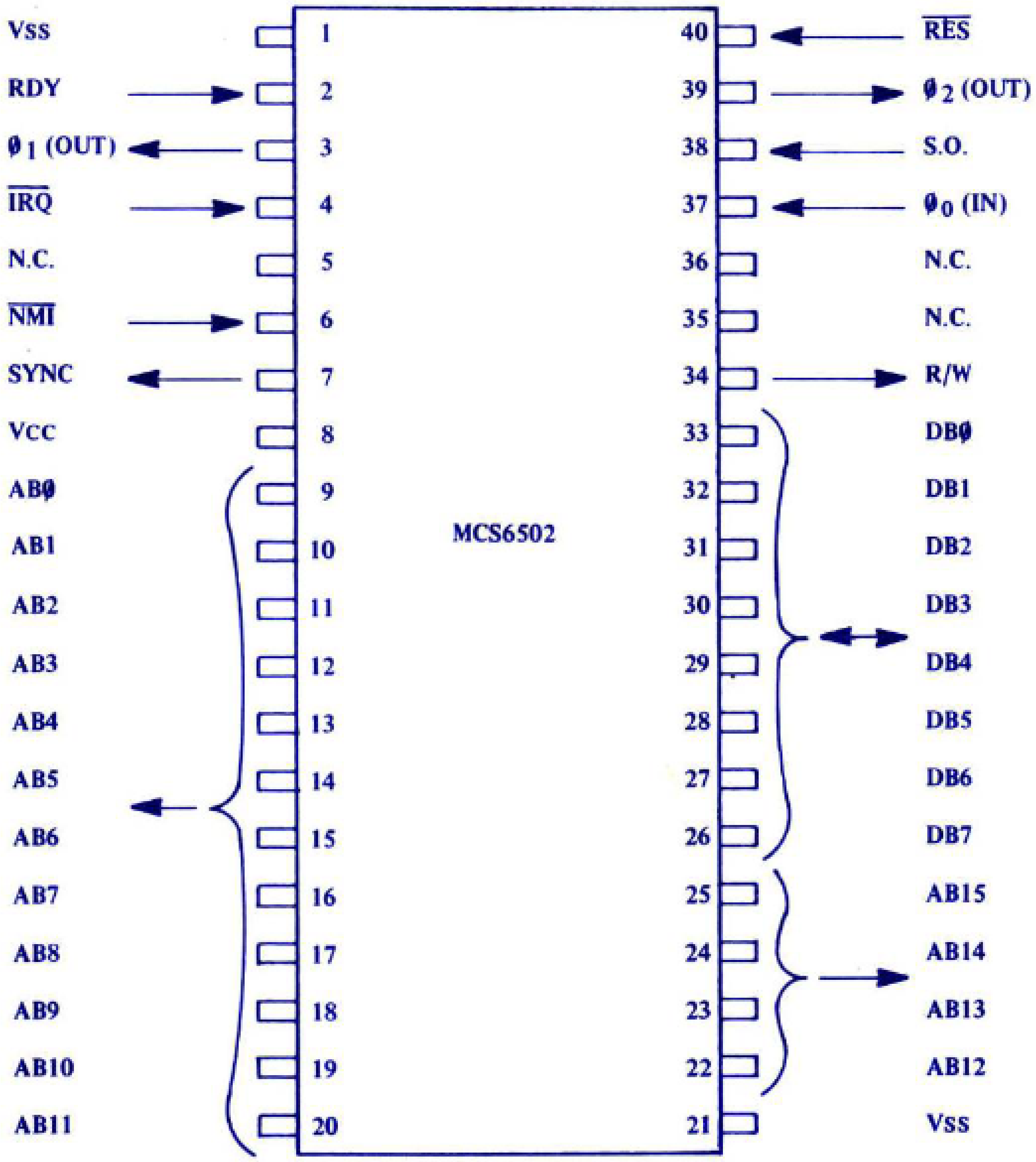
Address	Data	Dissassembly	Result
$0010	6C	JMP ($2300)	reg_pc = 1100
$0011	00		
$0012	23		
$2300	34		
$2301	12		
$1234	00		
$1235	11		

Absolute indexed X and Y addressing mode (ABS_IDX): The absolute address is fetched and it is added together with value of X or Y. The resulting address will contain the operator on which the operation is going to be executed.
Example

Address	Data	Dissassembly	Result
$0010	BD	LDA $1123,X ; X = 33	reg_a = 0x01
$0011	23		
$0012	11		
$1123	XX		
$1156	01		

[Car84] [MOS Technology, Inc.76b] [Mor12]

1.1.7 Pinout



N.C. = NO CONNECTION

MCS6502 Pinout Designations

- Vss = Negative Supply Voltage (not needed in VHDL).
- RDY (in) = Ready. Transitions take place during phase-1. The CPU checks ready line during phase-2. If ready line is low a wait state is introduced.
- Phi0 (in) = Input clock.
- Phi1 (out) and Phi2 (out) = Internally used phase-1 and phase-2 clocks. In the code only one input clock is used.

Phase-1 is equivalent to the rising edge of the input clock, and phase-2 is equivalent to the falling edge of the input clock.

- IRQ_n (in) = Interrupt Request. After completing the current instruction, the microprocessor will begin an interrupt sequence if and only if the interrupt mask flag is NOT set. The Program Counter and Status Register will be stored in the stack. The interrupt mask flag will be set high to prevent further interrupts. Program counter will be loaded from 0xFFFE (low) and 0xFFFF (high). Sampled during phase-2 if RDY is high and begins on phase-1.
- NMI_n (in) = Non-Maskable Interrupt. The same interrupt sequence is executed as for IRQ_n, but the state of the interrupt mask flag does not matter. The program counter is loaded from 0xFFFA (low) and 0xFFFB (high).
- SYNC (out) = This line goes high during phase-1 of fetch cycle.
- Vcc = Positive Supply Voltage (not needed in VHDL).
- AB (out) = Address Bus. Address is sent on phase-1.
- RES_n (in) = Reset. If this line is low, writing to or from the microprocessor is inhibited. If this line goes high program counter will be loaded from 0xFFFC (low) and 0xFFFD (high) while the interrupt mask is set. On the original CPU, the reset sequence took 6 clock cycles. I reduced it to 3.
- SO_n (in) = Set Overflow. If low the overflow flag will be set.
- R/W_n (out) = Read/Not Write.
- DB (inout) = Data Bus. Data is sent on phase-2 and it is read on phase-1.

[Car84] [MOS Technology, Inc.76a] [MOS Technology, Inc.75]

Implementation note: RDY, Phi0 (currenntly generated by testbench), IRQ_n, NMI_in, SO_n are not implemented or not implemented properly.

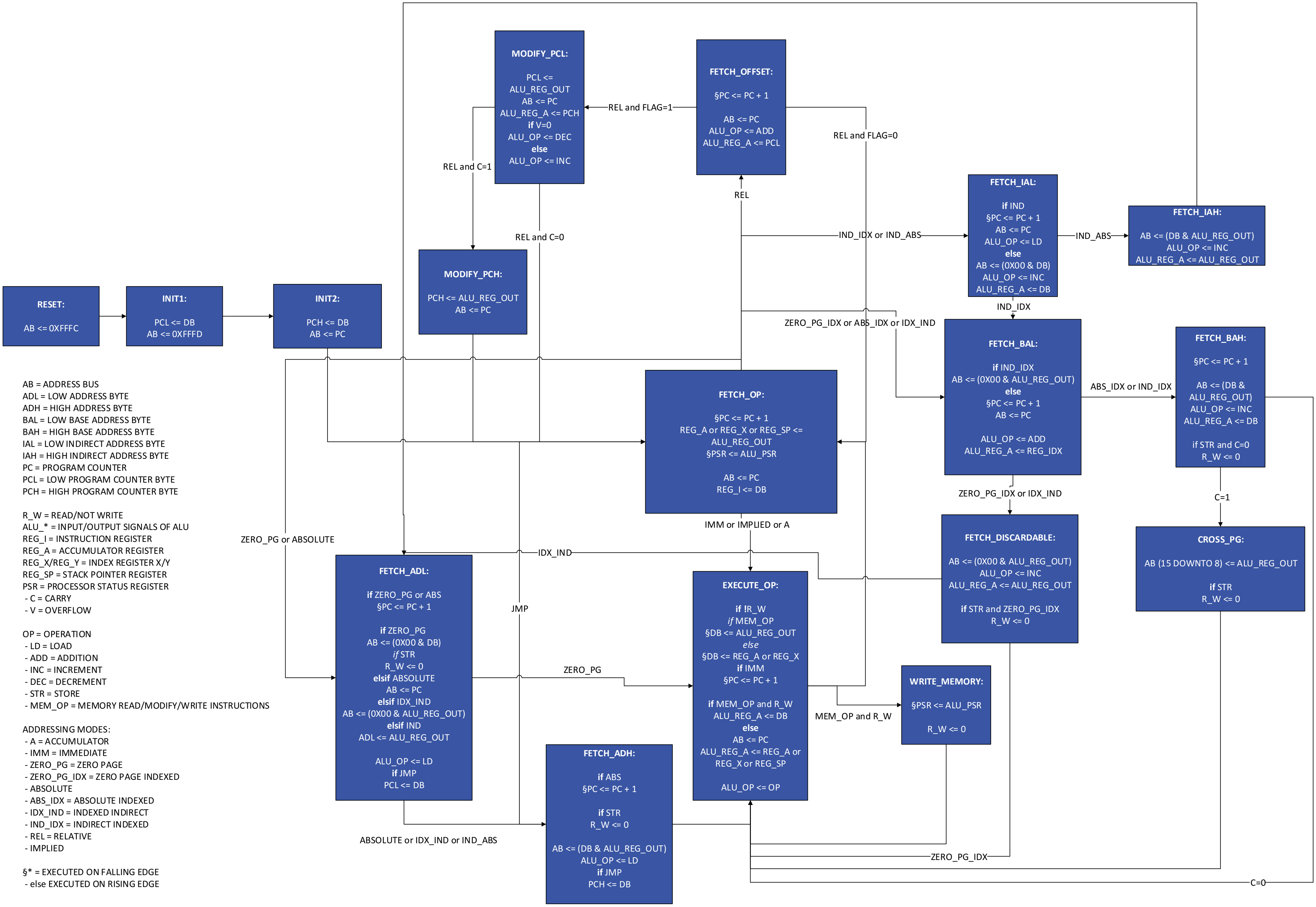
1.1.8 Microcode

Microcode is another name for a complete processor cycle. The term microcode is inspired by the first HDL implementation of 6502: Free-6502. [Kes99]

These cycles are also called microcode, because the processor executes operations which do not coincide necessarily with the operation decoded from the instruction. The next section provides a more detailed explanation of possible microcodes.

1.1.9 State machine

An explicit state machine is not available online. In the software manual there is a rough explanation of how many cycles a given addressing mode needs and what happens in those cycles. The state machine is constructed using these guidelines. [MOS Technology, Inc.76b]



AB = ADDRESS BUS
ADL = LOW ADDRESS BYTE
ADH = HIGH ADDRESS BYTE
BAL = LOW BASE ADDRESS BYTE
BAH = HIGH BASE ADDRESS BYTE
IAL = LOW INDIRECT ADDRESS BYTE
IAH = HIGH INDIRECT ADDRESS BYTE
PC = PROGRAM COUNTER
PCL = LOW PROGRAM COUNTER BYTE
PCH = HIGH PROGRAM COUNTER BYTE

R_W = READ/NOT WRITE
ALU_* = INPUT/OUTPUT SIGNALS OF ALU
REG_I = INSTRUCTION REGISTER
REG_A = ACCUMULATOR REGISTER
REG_X/REG_Y = INDEX REGISTER X/Y
REG_SP = STACK POINTER REGISTER
PSR = PROCESSOR STATUS REGISTER
- C = CARRY
- V = OVERFLOW

OP = OPERATION
- LD = LOAD
- ADD = ADDITION
- INC = INCREMENT
- DEC = DECREMENT
- STR = STORE
- MEM_OP = MEMORY READ/MODIFY/WRITE INSTRUCTIONS

ADDRESSING MODES:
- A = ACCUMULATOR
- IMM = IMMEDIATE
- ZERO_PG = ZERO PAGE
- ZERO_PG_IDX = ZERO PAGE INDEXED
- ABSOLUTE
- ABS_IDX = ABSOLUTE INDEXED
- IDX_IND = INDEXED INDIRECT
- IND_IDX = INDIRECT INDEXED
- REL = RELATIVE
- IMPLIED

§* = EXECUTED ON FALLING EDGE
- else EXECUTED ON RISING EDGE

1.2 Audio Processing Unit

CHAPTER

TWO

PICTURE PROCESSING UNIT

BIBLIOGRAPHY

[MOS Technology, Inc.75] *MCS6501 - MCS6505 MICROPROCESSORS*. MOS Technology, Inc., 1975.

[MOS Technology, Inc.76a] *MCS6500 MICROCOMPUTER FAMILY HARDWARE MANUAL*. MOS Technology, Inc., 1976.

[MOS Technology, Inc.76b] *MCS6500 MICROCOMPUTER FAMILY SOFTWARE MANUAL*. MOS Technology, Inc., 1976.

[Car84] Joseph J. Carr. *6502 User's Manual*. Reston Pub. Co., 1984.

[Inc75] Mos Technology Inc. Integrated circuit microprocessor with parallel binary adder having on-the-fly correction to provide decimal results. 1975. URL: https://patents.google.com/patent/US3991307.

[Kes99] David Kessner. Free-6502. 1999. URL: https://www.sprow.co.uk/dump/index.htm#Free6502.

[Mor12] Nick Morgan. Easy 6502. 2012. URL: https://skilldrick.github.io/easy6502/.

[Qui11] Quietust. Chip images. 2011. URL: https://www.qmtpro.com/~nes/chipimages/index.php.

[Tay04] Brad Taylor. 2a03 technical reference. 2004. URL: https://www.nesdev.org/2A03%20technical%20reference.txt.