

---

# **T34 Emulator Documentation**

***Release 0.1***

**Chezka Gaddi**

**Dec 08, 2019**



# CONTENTS

<b>1</b>	<b>T34 Emulator Tutorial</b>	<b>1</b>
1.1	Running the Application . . . . .	1
1.2	Functionality . . . . .	1
1.3	Load a Program . . . . .	1
1.4	Display the content of a specific memory address . . . . .	1
1.5	Display the content of a range of memory addresses . . . . .	2
1.6	Edit memory locations . . . . .	2
1.7	Run program starting as a specified address . . . . .	2
1.8	Exit the program . . . . .	2
<b>2</b>	<b>Documentation for the Code</b>	<b>3</b>
2.1	Emulator – auto members . . . . .	3
2.2	Instructions – auto members . . . . .	21
2.3	Memory – auto members . . . . .	21
<b>3</b>	<b>Testing the Program</b>	<b>23</b>
3.1	Test Emulator . . . . .	23
3.2	Test Memory . . . . .	23
3.3	Test Instructions . . . . .	24
<b>4</b>	<b>Indices and tables</b>	<b>29</b>
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>



## T34 EMULATOR TUTORIAL

This is the tutorial on how to use the T34 Emulator module.

### 1.1 Running the Application

### 1.2 Functionality

The monitor will have similar functionality as an OS. The T34 monitor has six functions;

1. *Load a Program*
2. *Display the content of a specific memory address*
3. *Display the content of a range of memory addresses*
4. *Edit memory locations*
5. *Run program starting as a specified address*
6. *Exit the program*

### 1.3 Load a Program

The machine can start in two modes. Either the user provided an object file (a program), if so, the program is loaded into the correct memory location, or the user just starts the emulator without any program. In both cases the monitor is started, and the user is provided with the monitor prompt (>).

To start the application with a program, run the application with the name of the object file.

```
$ python3 t34.py [filename]
```

### 1.4 Display the content of a specific memory address

By typing in the memory address in HEX at the Monitor prompt, the Monitor returns the byte (in HEX format) at that location.

```
> 200
200    A9
```

## 1.5 Display the content of a range of memory addresses

By typing in the starting address in HEX, followed by a period and finally the ending address in HEX at the Monitor prompt, the Monitor returns the bytes between those locations.

```
> 200.20F
200      A9 00 85 00 A5 00 8D 00
208      80 E6 00 4C 04 02 00 00
```

## 1.6 Edit memory locations

By typing in the starting address in HEX, followed by a colon, and then the new values for the memory locations at the Monitor prompt, the monitor updates the current locations.

```
> 300: A9 04 85 07 A0 00 84 06 A9 A0 91 06 C8 D0 FB E6 07
> 300.310
300      A9 04 85 07 A0 00 84 06
308      A9 A0 91 06 C8 D0 FB E6
310      07
```

## 1.7 Run program starting as a specified address

By typing in the starting address in HEX, followed by an R at the Monitor prompt. The monitor will execute all code starting at the address and up until the first BRK (opcode 00).

```
> 200R
PC  OPC  INS      AMOD OPRND  AC XR YR SP NV-BDIZC
200
```

## 1.8 Exit the program

The user should be able to exit the monitor (and python) in three ways:

1. Ctrl-C (keyboard interrupt)
2. Ctrl-D (EOF)
3. Type exit at the monitor prompt ( > exit)

## DOCUMENTATION FOR THE CODE

### 2.1 Emulator – auto members

**class** t34.Emulator.**Emulator** (*program\_name=None*)

Class to store an emulator and runs program files.

**access\_memory** (*address*)

Accesses the memory address and displays the contents.

**Parameters** **address** (*str*) – HEX address of the memory to be accessed.

**Returns** memory content

**Return type** string

**access\_memory\_range** (*begin, end*)

Accesses a memory range and displays all the contents.

**Parameters**

- **begin** (*str*) – beginning HEX address of the memory to be accessed.
- **end** (*str*) – end HEX address of the memory to be accessed.

**Return out** contents of the memory range.

**Return type** string

**adc\_abs** ()

This instruction adds the contents of a memory location to the accumulator together with the carry bit. If overflow occurs the carry bit is set, this enables multiple byte addition to be performed.

Processor Status after use:

C - Set if overflow in bit 7 Z - Set if A = 0 I - Not affected D - Not affected B - Not affected V - Set if sign bit is incorrect N - Set if bit 7 set

**adc\_imm** ()

This instruction adds the contents of a memory location to the accumulator together with the carry bit. If overflow occurs the carry bit is set, this enables multiple byte addition to be performed.

Processor Status after use:

C - Set if overflow in bit 7 Z - Set if A = 0 I - Not affected D - Not affected B - Not affected V - Set if sign bit is incorrect N - Set if bit 7 set

**adc\_zpg** ()

This instruction adds the contents of a memory location to the accumulator together with the carry bit. If overflow occurs the carry bit is set, this enables multiple byte addition to be performed.

Processor Status after use:

C - Set if overflow in bit 7 Z - Set if A = 0 I - Not affected D - Not affected B - Not affected V - Set if sign bit is incorrect N - Set if bit 7 set

### **and\_abs()**

A logical AND is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 set

### **and\_imm()**

A logical AND is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 set

### **and\_zpg()**

A logical AND is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 set

### **asl()**

This operation shifts all the bits of the accumulator or memory contents one bit left. Bit 0 is set to 0 and bit 7 is placed in the carry flag. The effect of this operation is to multiply the memory contents by 2 (ignoring 2's complement considerations), setting the carry if the result will not fit in 8 bits.

Processor Status after use:

C - Set to contents of old bit 7 Z - Set if A = 0 I - Not affected D - Not affected B - Not affected V - Not affected N - Set if bit 7 of the result is set

### **asl\_abs()**

This operation shifts all the bits of the accumulator or memory contents one bit left. Bit 0 is set to 0 and bit 7 is placed in the carry flag. The effect of this operation is to multiply the memory contents by 2 (ignoring 2's complement considerations), setting the carry if the result will not fit in 8 bits.

Processor Status after use:

C - Set to contents of old bit 7 Z - Set if A = 0 I - Not affected D - Not affected B - Not affected V - Not affected N - Set if bit 7 of the result is set

### **asl\_zpg()**

This operation shifts all the bits of the accumulator or memory contents one bit left. Bit 0 is set to 0 and bit 7 is placed in the carry flag. The effect of this operation is to multiply the memory contents by 2 (ignoring 2's complement considerations), setting the carry if the result will not fit in 8 bits.

Processor Status after use:



C - Set to contents of old bit 7 Z - Set if A = 0 I - Not affected D - Not affected B - Not affected V - Not affected N - Set if bit 7 of the result is set

**bcc\_rel()**

BCC - Branch if Carry Clear

If the carry flag is clear then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**bcs\_rel()**

BCS - Branch if Carry Set

If the carry flag is set then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**beq\_rel()**

BEQ - Branch on Result Zero

If the zero flag is set then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**bit\_abs()**

A & M, N = M7, V = M6

This instructions is used to test if one or more bits are set in a target memory location. The mask pattern in A is ANDed with the value in memory to set or clear the zero flag, but the result is not kept. Bits 7 and 6 of the value from memory are copied into the N and V flags.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if the result if the AND is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Set to bit 6 of the memory value N Negative Flag Set to bit 7 of the memory value

**bit\_zpg()**

A & M, N = M7, V = M6

This instructions is used to test if one or more bits are set in a target memory location. The mask pattern in A is ANDed with the value in memory to set or clear the zero flag, but the result is not kept. Bits 7 and 6 of the value from memory are copied into the N and V flags.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if the result if the AND is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Set to bit 6 of the memory value N Negative Flag Set to bit 7 of the memory value

**bmi\_rel()**

BMI - Branch if Minus

If the negative flag is set then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **bne\_rel()**

BNE - Branch if Not Zero

If the zero flag is not set then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **bpl\_rel()**

BNE - Branch if Plus

If the negative flag is not set then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **brk()**

The BRK instruction forces the generation of an interrupt request. The program counter and processor status are pushed on the stack then the IRQ interrupt vector at \$FFFE/F is loaded into the PC and the break flag in the status set to one.

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Set to 1 V Overflow Flag Not affected N Negative Flag Not affected

### **bvc()**

BVC - Branch on Overflow Clear

If the overflow flag is not set then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **bvs()**

BVS - Branch if Overflow Set

If the overflow flag not set then add the relative displacement to the program counter to cause a branch to a new location.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **carry\_isSet()**

Checks if carry bit is set.

**Returns:** Bool – status of carry bit

**check\_negative** (*value: int*)

Checks sign of the value and sets the negative bit respectively.

**Arguments:** value {int} – value to check sign

**check\_negative\_sign** (*sign: int*)

Determines how to set the negative bit.

**Arguments:** sign {int} – sign of the number

**check\_zero** (*value: int*)

Checks is the value is zero and sets the zero bit respectively.

**Arguments:** value {int} – value to check

**clc** ()

C = 0

Set the carry flag to zero.

C Carry Flag Set to 0 Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**cld** ()

D = 0

Sets the decimal mode flag to zero.

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Set to 0 B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**cli** ()

I = 0

Clears the interrupt disable flag allowing normal interrupt requests to be serviced.

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Set to 0 D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**clv** ()

V = 0

Clears the overflow flag.

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Set to 0 N Negative Flag Not affected

**cmp\_imm** ()

Z,C,N = A-M

This instruction compares the contents of the accumulator with another memory held value and sets the zero and carry flags as appropriate.

Processor Status after use:

C Carry Flag Set if A >= M Z Zero Flag Set if A = M I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**cmp\_zpg** ()

Z,C,N = A-M

This instruction compares the contents of the accumulator with another memory held value and sets the zero and carry flags as appropriate.

Processor Status after use:

C Carry Flag Set if  $A \geq M$  Z Zero Flag Set if  $A = M$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**cpx\_imm()**

Z,C,N = X-M

This instruction compares the contents of the X register with another memory held value and sets the zero and carry flags as appropriate.

Processor Status after use:

C Carry Flag Set if  $X \geq M$  Z Zero Flag Set if  $X = M$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**cpx\_zpg()**

Z,C,N = X-M

This instruction compares the contents of the X register with another memory held value and sets the zero and carry flags as appropriate.

Processor Status after use:

C Carry Flag Set if  $X \geq M$  Z Zero Flag Set if  $X = M$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**cpx\_imm()**

Z,C,N = Y-M

This instruction compares the contents of the Y register with another memory held value and sets the zero and carry flags as appropriate.

Processor Status after use:

C Carry Flag Set if  $Y \geq M$  Z Zero Flag Set if  $Y = M$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**cpx\_zpg()**

Z,C,N = Y-M

This instruction compares the contents of the Y register with another memory held value and sets the zero and carry flags as appropriate.

Processor Status after use:

C Carry Flag Set if  $Y \geq M$  Z Zero Flag Set if  $Y = M$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**dec\_abs()**

M,Z,N = M-1

Subtracts one from the value held at a specified memory location setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if result is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**dec\_zpg()**

$M, Z, N = M - 1$

Subtracts one from the value held at a specified memory location setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if result is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**dex()**

$X, Z, N = X - 1$

Subtracts one from the X register setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if X is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of X is set

**dey()**

$Y, Z, N = Y - 1$

Subtracts one from the Y register setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if Y is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of Y is set

**edit\_memory(address, data)**

Edits the contents of a specific memory address.

**Parameters**

- **address** (*str*) – HEX address of the memory to be edited.
- **data** (*str*) – data to store into the memory address.

**eor\_abs()**

$A, Z, N = A \wedge M$

An exclusive OR is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if  $A = 0$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 set

**eor\_imm()**

$A, Z, N = A \wedge M$

An exclusive OR is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 set

**eor\_zpg()**

A,Z,N = A^M

An exclusive OR is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 set

**execute\_instruction(address)**

Gets the instruction stored in memory, decodes it and executes it.

**Parameters** **address** – Location of the command to be executed

**Return output** Contents of specific

**get\_AC()** → int

Retrieve current address stored in AC register.

**Returns:** int – address stored in AC

**get\_PC()** → int

Retrieve current address stored in PC register.

**Returns:** int – address stored in PC

**get\_SP()** → int

Retrieve current address stored in SP register.

**Returns:** int – address stored in SP

**get\_SR()** → int

Retrieve current address stored in SR register.

**Returns:** int – address stored in SR

**get\_X()** → int

Retrieve current address stored in X register.

**Returns:** int – address stored in X

**get\_Y()** → int

Retrieve current address stored in Y register.

**Returns:** int – address stored in Y

**inc\_abs()**

M,Z,N = M+1

Adds one from the value held at a specified memory location setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if result is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**inc\_zpg()**

M,Z,N = M+1

Adds one to the value held at a specified memory location setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if result is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**initialize\_registers()**

Initialize registers to initial values.

PC: 0 AC: 0 X: 0 Y: 0 SP: 0xFF SR: 0x20

**inx()**

X,Z,N = X+1

Adds one to the X register setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if X is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of X is set

**iny()**

Y,Z,N = Y+1

Adds one to the Y register setting the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if Y is zero I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of Y is set

**jmp\_abs()**

JMP - Jump

Sets the program counter to the address specified by the operand.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**jmp\_ind()**

JMP - Jump

Sets the program counter to the address specified by the operand.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**jsr()**

JSR - Jump to Subroutine

The JSR instruction pushes the address (minus one) of the return point on to the stack and then sets the program counter to the target memory address.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**lda\_abs()**

A,Z,N = M

Loads a byte of memory into the accumulator setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of A is set

**lda\_imm()**

A,Z,N = M

Loads a byte of memory into the accumulator setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of A is set

**lda\_zpg()**

A,Z,N = M

Loads a byte of memory into the accumulator setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of A is set

**ldx\_abs()**

X,Z,N = M

Loads a byte of memory into the x register setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of A is set

**ldx\_imm()**

X,Z,N = M

Loads a byte of memory into the X register setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if X = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of X is set

**ldx\_zpg()**

X,Z,N = M

Loads a byte of memory into the X register setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if X = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of X is set

**ldy\_abs()**

Y,Z,N = M

Loads a byte of memory into the Y register setting the zero and negative flags as appropriate.



C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of Y is set

**ldy\_imm()**  
Y,Z,N = M

Loads a byte of memory into the Y register setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if Y = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of Y is set

**ldy\_zpg()**  
Y,Z,N = M

Loads a byte of memory into the Y register setting the zero and negative flags as appropriate.

C Carry Flag Not affected Z Zero Flag Set if Y = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of Y is set

**load\_program()**  
Loads the program.

**Returns** successful read

**Return type** bool

**lsr()**  
LSR - Logical Shift Right  
A,C,Z,N = A/2 or M,C,Z,N = M/2

Each of the bits in A or M is shift one place to the right. The bit that was in bit 0 is shifted into the carry flag. Bit 7 is set to zero.

Processor Status after use:

C Carry Flag Set to contents of old bit 0 Z Zero Flag Set if result = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**lsr\_abs()**  
LSR - Logical Shift Right  
A,C,Z,N = A/2 or M,C,Z,N = M/2

Each of the bits in A or M is shift one place to the right. The bit that was in bit 0 is shifted into the carry flag. Bit 7 is set to zero.

Processor Status after use:

C Carry Flag Set to contents of old bit 0 Z Zero Flag Set if result = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**lsr\_zpg()**  
LSR - Logical Shift Right  
A,C,Z,N = A/2 or M,C,Z,N = M/2

Each of the bits in A or M is shift one place to the right. The bit that was in bit 0 is shifted into the carry flag. Bit 7 is set to zero.

Processor Status after use:

C Carry Flag Set to contents of old bit 0 Z Zero Flag Set if result = 0 I Interrupt Disable Not affected  
D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N  
Negative Flag Set if bit 7 of the result is set

**make\_address** (*mem\_address*) -> (<class 'int'>, <class 'int'>, <class 'int'>)

Make a full address from the values in memory and return pieces.

**Arguments:** int {mem\_address} – location in memory with the address

**Returns:** int – low, high, and address

**negative\_isSet** ()

Checks if negative bit is set.

**Returns:** Bool – status of negative bit

**nop** ()

NOP - No Operation The NOP instruction causes no changes to the processor other than the normal incrementing of the program counter to the next instruction.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag  
Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**ora\_abs** ()

A,Z,N = A|M

An inclusive OR is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag  
Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7  
set

**ora\_imm** ()

A,Z,N = A|M

An inclusive OR is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag  
Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7  
set

**ora\_zpg** ()

A,Z,N = A|M

An inclusive OR is performed, bit by bit, on the accumulator contents using the contents of a byte of memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag  
Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7  
set

**overflow\_isSet** ()

Checks if overflow bit is set.

**Returns:** Bool – status of overflow bit

**pha()**

PHA - Push Accumulator Pushes a copy of the accumulator on to the stack.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**php()**

PHP - Push Processor Status Pushes a copy of the status flags on to the stack.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**pla()**

PLA - Pull Accumulator Pulls an 8 bit value from the stack and into the accumulator. The zero and negative flags are set as appropriate.

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of A is set

**plp()**

PLP - Pull Processor Status Pulls an 8 bit value from the stack and into the processor flags. The flags will take on new states as determined by the value pulled.

Processor Status after use:

C Carry Flag Set from stack Z Zero Flag Set from stack I Interrupt Disable Set from stack D Decimal Mode Flag Set from stack B Break Command Set from stack V Overflow Flag Set from stack N Negative Flag Set from stack

**pop\_from\_stack** (*size: int*) → ByteString

Pop data from stack.

**Arguments:** size {int} – size of data

**push\_to\_stack** (*data: int, size: int*)

Push data onto the stack.

**Arguments:** data {int} – data to be stored into the stack size {int} – size of data

**read\_memory** (*start: Address, end: Address*)

Edits the contents of a specific memory address.

#### Parameters

- **address** (*str*) – HEX address of the memory to be edited.
- **data** (*str*) – data to store into the memory address.

**rol()**

ROL - Rotate Left

Move each of the bits in either A or M one place to the left. Bit 0 is filled with the current value of the carry flag whilst the old bit 7 becomes the new carry flag value.

Processor Status after use:

C Carry Flag Set to contents of old bit 7 Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**rol\_abs()**

ROL - Rotate Left

Move each of the bits in either A or M one place to the left. Bit 0 is filled with the current value of the carry flag whilst the old bit 7 becomes the new carry flag value.

Processor Status after use:

C Carry Flag Set to contents of old bit 7 Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**rol\_zpg()**

ROL - Rotate Left

Move each of the bits in either A or M one place to the left. Bit 0 is filled with the current value of the carry flag whilst the old bit 7 becomes the new carry flag value.

Processor Status after use:

C Carry Flag Set to contents of old bit 7 Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**rор()**

ROR - Rotate Right

Move each of the bits in either A or M one place to the right. Bit 7 is filled with the current value of the carry flag whilst the old bit 0 becomes the new carry flag value.

Processor Status after use:

C Carry Flag Set to contents of old bit 0 Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**rор\_abs()**

ROR - Rotate Right

Move each of the bits in either A or M one place to the right. Bit 7 is filled with the current value of the carry flag whilst the old bit 0 becomes the new carry flag value.

Processor Status after use:

C Carry Flag Set to contents of old bit 0 Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**rор\_zpg()**

ROR - Rotate Right

Move each of the bits in either A or M one place to the right. Bit 7 is filled with the current value of the carry flag whilst the old bit 0 becomes the new carry flag value.

Processor Status after use:

C Carry Flag Set to contents of old bit 0 Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of the result is set

**rts()**

RTS - Return from Subroutine

The RTS instruction is used at the end of a subroutine to return to the calling routine. It pulls the program counter (minus one) from the stack.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**run\_program(address)**

Start program at specific location in memory until end of program.

**Parameters address** – Location of the command to be executed.

**Return output** Contents of all the registers.

**Return type** string

**sbcmimm()**

A,Z,C,N = A-M-(1-C)

This instruction subtracts the contents of a memory location to the accumulator together with the not of the carry bit. If overflow occurs the carry bit is clear, this enables multiple byte subtraction to be performed.

Processor Status after use:

C Carry Flag Clear if overflow in bit 7 Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Set if sign bit is incorrect N Negative Flag Set if bit 7 set

**sec()**

SEC - Set Carry Flag C = 1

Set the carry flag to one.

C Carry Flag Set to 1 Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**sed()**

SED - Set Decimal Flag D = 1

Set the decimal mode flag to one.

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Set to 1 B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**sei()**

SEI - Set Interrupt Disable I = 1

Set the interrupt disable flag to one.

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Set to 1 D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**set\_carry()**

Set carry bit to 1

**set\_negative()**

Set negative bit to 1

**set\_overflow()**

Set overflow bit

### **set\_zero()**

Set zero bit to 1

### **sta\_abs()**

STA - Store Accumulator

$M = A$

Stores the contents of the accumulator into memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **sta\_zpg()**

STA - Store Accumulator

$M = A$

Stores the contents of the accumulator into memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **start\_emulator()**

Starts the emulator and evaluates and executes commands.

### **stx\_abs()**

STX - Store X

$M = X$

Stores the contents of the X register into memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **stx\_zpg()**

STX - Store X Register

$M = X$

Stores the contents of the X register into memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

### **sty\_abs()**

STY - Store Y

$M = Y$

Stores the contents of the Y register into memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**sty\_zpg()**

STY - Store X Register

$M = Y$

Stores the contents of the Y register into memory.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**tax()**

TAX - Transfer Accumulator to X  $X = A$

Copies the current contents of the accumulator into the X register and sets the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if  $X = 0$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of X is set

**tay()**

TAY - Transfer Accumulator to Y  $Y = A$

Copies the current contents of the accumulator into the Y register and sets the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if  $Y = 0$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of Y is set

**tsx()**

TSX - Transfer Stack Pointer to X  $X = S$

Copies the current contents of the stack register into the X register and sets the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if  $X = 0$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of X is set

**txa()**

TXA - Transfer X to Accumulator  $A = X$

Copies the current contents of the X register into the accumulator and sets the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if  $A = 0$  I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of A is set

**txs()**

TXS - Transfer X to Stack Pointer  $S = X$

Copies the current contents of the X register into the stack register.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Not affected I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Not affected

**tya()**

TYA - Transfer Y to Accumulator A = Y

Copies the current contents of the Y register into the accumulator and sets the zero and negative flags as appropriate.

Processor Status after use:

C Carry Flag Not affected Z Zero Flag Set if A = 0 I Interrupt Disable Not affected D Decimal Mode Flag Not affected B Break Command Not affected V Overflow Flag Not affected N Negative Flag Set if bit 7 of A is set

**unset\_carry()**

Set carry bit to 0

**unset\_negative()**

Set negative bit to 0

**unset\_overflow()**

Unset overflow bit

**unset\_zero()**

Unset zero bit

**write\_AC(value: int)**

Write to the AC register.

**Arguments:** value {int} – new AC

**write\_PC(value: int)**

Write to the PC register.

**Arguments:** value {int} – new PC

**write\_SP(value: int)**

Write to the SP register.

**Arguments:** value {int} – new SP

**write\_SR(value: int)**

Write to the SR register.

**Arguments:** value {int} – new SR

**write\_X(value: int)**

Write to the X register.

**Arguments:** value {int} – new X

**write\_Y(value: int)**

Write to the Y register.

**Arguments:** value {int} – new Y

**write\_memory(address: Address, data: ByteString)**

Writes data to a specific memory address.

**Parameters**

- **address** (*Address*) – HEX address of the memory to be edited.



- **data** (*ByteString*) – data to store into the memory address.

**zero\_isSet** ()

Checks if zero bit is set.

**Returns:** Bool – status of zero bit

## 2.2 Instructions – auto members

## 2.3 Memory – auto members



## TESTING THE PROGRAM

All of the functionality of the *Emulator* class is tested with the unittest found in the TestEmulator and TestInstruction and TestMemory modules. All tests could be run with the command

```
python3 -m unittest discover
```

### 3.1 Test Emulator

**class** tests.test\_emulator.**TestEmulator** (*methodName='runTest'*)

Unit testing class for all the functionality of the Emulator class.

**setUp** ()

Setup the Emulator object to be used for all the tests.

**test\_access\_memory** ()

Test access to a memory address.

**test\_access\_memory\_range** ()

Test access to a memory address range.

**test\_edit\_memory\_locations** ()

Test edit of a memory location.

### 3.2 Test Memory

**class** tests.test\_memory.**TestMemory** (*methodName='runTest'*)

Unit testing class for all the functionality of the Emulator class.

**setUp** ()

Setup the Emulator object to be used for all the tests.

**test\_get\_ac** ()

Test retrieving address from AC register.

**test\_get\_pc** ()

Test retrieving address from PC register.

**test\_get\_sp** ()

Test retrieving address from Y register.

**test\_get\_sr** ()

Test retrieving address from Y register.

**test\_get\_x()**  
Test retrieving address from X register.

**test\_get\_y()**  
Test retrieving address from Y register.

**test\_write\_ac()**  
Test writing to AC register.

**test\_write\_pc()**  
Test writing to PC register.

**test\_write\_sp()**  
Test writing to SP register.

**test\_write\_sr()**  
Test writing to SR register.

**test\_write\_x()**  
Test writing to X register.

**test\_write\_y()**  
Test writing to Y register.

### 3.3 Test Instructions

**class** tests.test\_instructions.**TestInstructions** (*methodName='runTest'*)  
Unit testing class for all instructions in the Instructions class.

**setUp()**  
Hook method for setting up the test fixture before exercising it.

**set\_ac()**  
Sets ac to 4

**test\_adc\_abs()**  
Test adc abs instruction with overflow and carry. -122+(-94)

**test\_adc\_imm\_carry()**  
Test adc imm with hanging carry. -22+(-43)+1

**test\_adc\_imm\_nc()**  
Test adc imm instruction with negative and carry. -22+(-43)

**test\_adc\_imm\_nv()**  
Test adc imm instruction with negative and overflow. 113+25

**test\_adc\_imm\_vc()**  
Test adc imm instruction with overflow and carry. -122+(-94)

**test\_and\_abs()**  
Test and abs instruction. 5&4

**test\_and\_imm()**  
Test and imm instruction. 5&4

**test\_asl()**  
Test asl instruction.

**test\_asl\_abs()**  
Test asl abs instruction.

**test\_bcc\_rel()**  
Test bcc rel instruction.

**test\_bcs\_rel()**  
Test bcs rel instruction.

**test\_beq\_rel()**  
Test beq rel instruction.

**test\_bit\_abs()**  
Test bit abs instruction.

**test\_bit\_zpg()**  
Test bit zpg instruction.

**test\_bmi\_rel()**  
Test bmi rel instruction.

**test\_bne\_rel()**  
Test bne rel instruction.

**test\_bpl\_rel()**  
Test bpl rel instruction.

**test\_bvc()**  
Test bvc instruction.

**test\_bvs()**  
Test bvs instruction.

**test\_clc()**  
Test clc instruction.

**test\_cld()**  
Test cld instruction.

**test\_cli()**  
Test cli instruction.

**test\_clv()**  
Test clv instruction.

**test\_dec\_abs()**  
Test dec abs instruction.

**test\_dec\_zpg()**  
Test dec zpg instruction.

**test\_dex()**  
Test dex instruction.

**test\_dex\_xnegative()**  
Test dex to negative instruction.

**test\_dey()**  
Test dey instruction.

**test\_dey\_ynegative()**  
Test dey to negative instruction.

**test\_eor\_abs()**  
Test eor abs instruction. 5^4

**test\_eor\_imm()**  
Test eor imm instruction. 5^4

**test\_eor\_zpg()**  
Test eor zpg instruction. 5^4

**test\_imm\_with\_cmp()**  
Test immediate with compare instructions.

**test\_immediate\_and\_zero()**  
Test immediate and zeropage instructions.

**test\_inc\_abs()**  
Test inc abs instruction.

**test\_inx()**  
Test inx instruction.

**test\_iny()**  
Test iny instruction.

**test\_jump\_abs()**  
Test jmp abs instruction.

**test\_jump\_ind()**  
Test jmp ind instruction.

**test\_jsr\_abs()**  
Test jsr abs instruction.

**test\_lda\_abs()**  
Test lda abs instruction.

**test\_lda\_imm()**  
Test lda imm instruction.

**test\_lda\_zpg()**  
Test lda zpg instruction.

**test\_ldx\_abs()**  
Test ldx abs instruction. 0 -> X, set zero flag, unset negative

**test\_ldx\_imm()**  
Test ldx imm instruction.

**test\_ldx\_zpg()**  
Test ldx zpg instruction.

**test\_ldy\_abs()**  
Test ldx abs instruction. 0 -> Y, set zero flag, unset negative

**test\_ldy\_imm()**  
Test ldy imm instruction.

**test\_ldy\_zpg()**  
-1 -> Y, set negative

**test\_lsr()**  
Test lsr instruction.

**test\_lsr\_abs()**  
5 -> 2, set carry

**test\_lsr\_carry()**  
Test lsr instruction with carry.

**test\_lsr\_zpg()**  
Test lsr zpg instruction.

**test\_ora\_abs()**  
Test ora abs instruction. 194 | 169 -> A

**test\_ora\_imm()**  
Test ora imm instruction.

**test\_ora\_zpg()**  
Test ora zpg instruction.

**test\_pha()**  
Test pha instruction.

**test\_php()**  
Test php instruction.

**test\_pla()**  
Test php instruction.

**test\_rol()**  
Test rol instruction.

**test\_rol\_abs()**  
Test rol abs instruction. 01 -> 02

**test\_rol\_zpg()**  
Test rol zpg instruction.

**test\_ror()**  
Test ror instruction.

**test\_ror\_abs()**  
Test ror abs instruction. 01 -> 00

**test\_ror\_zpg()**  
Test ror zpg instruction.

**test\_run\_program\_nop()**  
Test run program with no operand.

**test\_sec()**  
Test sec instruction.

**test\_sed()**  
Test sed instruction.

**test\_sei()**  
Test sei instruction.

**test\_sta\_abs()**  
Test sta abs instruction. FF -> M

**test\_sta\_zpg()**  
Test sta zpg instruction.

**test\_stx\_abs()**  
Test stx abs instruction. FF -> M

**test\_stx\_zpg()**  
Test stx zpg instruction.

**test\_sty\_abs()**  
Test sty abs instruction. FF -> M

**test\_sty\_zpg()**  
Test sty zpg instruction.

**test\_tax()**  
Test tax instruction.

**test\_tay()**  
Test tay instruction.

**test\_tsx()**  
Test tsx instruction.

**test\_txa()**  
Test txa instruction.

**test\_txs()**  
Test txs instruction.

**test\_tya()**  
Test tya instruction.

**test\_zeropage()**  
Test zeropage instructions.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### e

Emulator, [3](#)

### i

Instructions, [21](#)

### m

Memory, [21](#)

### t

t34.Emulator, [3](#)

t34.Instructions, [21](#)

t34.Memory, [21](#)

TestEmulator, [23](#)

TestInstructions, [24](#)

TestMemory, [23](#)

tests.test\_emulator, [23](#)

tests.test\_instructions, [24](#)

tests.test\_memory, [23](#)



## A

`access_memory()` (*t34.Emulator.Emulator method*), 3  
`access_memory_range()` (*t34.Emulator.Emulator method*), 3  
`adc_abs()` (*t34.Emulator.Emulator method*), 3  
`adc_imm()` (*t34.Emulator.Emulator method*), 3  
`adc_zpg()` (*t34.Emulator.Emulator method*), 3  
`and_abs()` (*t34.Emulator.Emulator method*), 4  
`and_imm()` (*t34.Emulator.Emulator method*), 4  
`and_zpg()` (*t34.Emulator.Emulator method*), 4  
`asl()` (*t34.Emulator.Emulator method*), 4  
`asl_abs()` (*t34.Emulator.Emulator method*), 4  
`asl_zpg()` (*t34.Emulator.Emulator method*), 4

## B

`bcc_rel()` (*t34.Emulator.Emulator method*), 5  
`bcs_rel()` (*t34.Emulator.Emulator method*), 5  
`beq_rel()` (*t34.Emulator.Emulator method*), 5  
`bit_abs()` (*t34.Emulator.Emulator method*), 5  
`bit_zpg()` (*t34.Emulator.Emulator method*), 5  
`bmi_rel()` (*t34.Emulator.Emulator method*), 5  
`bne_rel()` (*t34.Emulator.Emulator method*), 6  
`bpl_rel()` (*t34.Emulator.Emulator method*), 6  
`brk()` (*t34.Emulator.Emulator method*), 6  
`bvc()` (*t34.Emulator.Emulator method*), 6  
`bvs()` (*t34.Emulator.Emulator method*), 6

## C

`carry_isSet()` (*t34.Emulator.Emulator method*), 6  
`check_negative()` (*t34.Emulator.Emulator method*), 6  
`check_negative_sign()` (*t34.Emulator.Emulator method*), 7  
`check_zero()` (*t34.Emulator.Emulator method*), 7  
`clc()` (*t34.Emulator.Emulator method*), 7  
`cld()` (*t34.Emulator.Emulator method*), 7  
`cli()` (*t34.Emulator.Emulator method*), 7  
`clv()` (*t34.Emulator.Emulator method*), 7  
`cmp_imm()` (*t34.Emulator.Emulator method*), 7  
`cmp_zpg()` (*t34.Emulator.Emulator method*), 7  
`cpx_imm()` (*t34.Emulator.Emulator method*), 8

`cpx_zpg()` (*t34.Emulator.Emulator method*), 8  
`cpy_imm()` (*t34.Emulator.Emulator method*), 8  
`cpy_zpg()` (*t34.Emulator.Emulator method*), 8

## D

`dec_abs()` (*t34.Emulator.Emulator method*), 8  
`dec_zpg()` (*t34.Emulator.Emulator method*), 8  
`dex()` (*t34.Emulator.Emulator method*), 9  
`dey()` (*t34.Emulator.Emulator method*), 9

## E

`edit_memory()` (*t34.Emulator.Emulator method*), 9  
`Emulator` (*class in t34.Emulator*), 3  
`Emulator` (*module*), 3  
`eor_abs()` (*t34.Emulator.Emulator method*), 9  
`eor_imm()` (*t34.Emulator.Emulator method*), 9  
`eor_zpg()` (*t34.Emulator.Emulator method*), 10  
`execute_instruction()` (*t34.Emulator.Emulator method*), 10

## G

`get_AC()` (*t34.Emulator.Emulator method*), 10  
`get_PC()` (*t34.Emulator.Emulator method*), 10  
`get_SP()` (*t34.Emulator.Emulator method*), 10  
`get_SR()` (*t34.Emulator.Emulator method*), 10  
`get_X()` (*t34.Emulator.Emulator method*), 10  
`get_Y()` (*t34.Emulator.Emulator method*), 10

## I

`inc_abs()` (*t34.Emulator.Emulator method*), 10  
`inc_zpg()` (*t34.Emulator.Emulator method*), 10  
`initialize_registers()` (*t34.Emulator.Emulator method*), 11  
`Instructions` (*module*), 21  
`inx()` (*t34.Emulator.Emulator method*), 11  
`iny()` (*t34.Emulator.Emulator method*), 11

## J

`jmp_abs()` (*t34.Emulator.Emulator method*), 11  
`jmp_ind()` (*t34.Emulator.Emulator method*), 11  
`jsr()` (*t34.Emulator.Emulator method*), 11

## L

`lda_abs()` (*t34.Emulator.Emulator method*), 12  
`lda_imm()` (*t34.Emulator.Emulator method*), 12  
`lda_zpg()` (*t34.Emulator.Emulator method*), 12  
`ldx_abs()` (*t34.Emulator.Emulator method*), 12  
`ldx_imm()` (*t34.Emulator.Emulator method*), 12  
`ldx_zpg()` (*t34.Emulator.Emulator method*), 12  
`ldy_abs()` (*t34.Emulator.Emulator method*), 12  
`ldy_imm()` (*t34.Emulator.Emulator method*), 13  
`ldy_zpg()` (*t34.Emulator.Emulator method*), 13  
`load_program()` (*t34.Emulator.Emulator method*), 13  
`lsr()` (*t34.Emulator.Emulator method*), 13  
`lsr_abs()` (*t34.Emulator.Emulator method*), 13  
`lsr_zpg()` (*t34.Emulator.Emulator method*), 13

## M

`make_address()` (*t34.Emulator.Emulator method*), 14  
`Memory` (*module*), 21

## N

`negative_isSet()` (*t34.Emulator.Emulator method*), 14  
`nop()` (*t34.Emulator.Emulator method*), 14

## O

`ora_abs()` (*t34.Emulator.Emulator method*), 14  
`ora_imm()` (*t34.Emulator.Emulator method*), 14  
`ora_zpg()` (*t34.Emulator.Emulator method*), 14  
`overflow_isSet()` (*t34.Emulator.Emulator method*), 14

## P

`pha()` (*t34.Emulator.Emulator method*), 15  
`php()` (*t34.Emulator.Emulator method*), 15  
`pla()` (*t34.Emulator.Emulator method*), 15  
`plp()` (*t34.Emulator.Emulator method*), 15  
`pop_from_stack()` (*t34.Emulator.Emulator method*), 15  
`push_to_stack()` (*t34.Emulator.Emulator method*), 15

## R

`read_memory()` (*t34.Emulator.Emulator method*), 15  
`rol()` (*t34.Emulator.Emulator method*), 15  
`rol_abs()` (*t34.Emulator.Emulator method*), 16  
`rol_zpg()` (*t34.Emulator.Emulator method*), 16  
`ror()` (*t34.Emulator.Emulator method*), 16  
`ror_abs()` (*t34.Emulator.Emulator method*), 16  
`ror_zpg()` (*t34.Emulator.Emulator method*), 16  
`rts()` (*t34.Emulator.Emulator method*), 16  
`run_program()` (*t34.Emulator.Emulator method*), 17

## S

`sbc_imm()` (*t34.Emulator.Emulator method*), 17  
`sec()` (*t34.Emulator.Emulator method*), 17  
`sed()` (*t34.Emulator.Emulator method*), 17  
`sei()` (*t34.Emulator.Emulator method*), 17  
`set_ac()` (*tests.test\_instructions.TestInstructions method*), 24  
`set_carry()` (*t34.Emulator.Emulator method*), 17  
`set_negative()` (*t34.Emulator.Emulator method*), 17  
`set_overflow()` (*t34.Emulator.Emulator method*), 17  
`set_zero()` (*t34.Emulator.Emulator method*), 17  
`setUp()` (*tests.test\_emulator.TestEmulator method*), 23  
`setUp()` (*tests.test\_instructions.TestInstructions method*), 24  
`setUp()` (*tests.test\_memory.TestMemory method*), 23  
`sta_abs()` (*t34.Emulator.Emulator method*), 18  
`sta_zpg()` (*t34.Emulator.Emulator method*), 18  
`start_emulator()` (*t34.Emulator.Emulator method*), 18  
`stx_abs()` (*t34.Emulator.Emulator method*), 18  
`stx_zpg()` (*t34.Emulator.Emulator method*), 18  
`sty_abs()` (*t34.Emulator.Emulator method*), 18  
`sty_zpg()` (*t34.Emulator.Emulator method*), 18

## T

`t34.Emulator` (*module*), 3  
`t34.Instructions` (*module*), 21  
`t34.Memory` (*module*), 21  
`tax()` (*t34.Emulator.Emulator method*), 19  
`tay()` (*t34.Emulator.Emulator method*), 19  
`test_access_memory()` (*tests.test\_emulator.TestEmulator method*), 23  
`test_access_memory_range()` (*tests.test\_emulator.TestEmulator method*), 23  
`test_adc_abs()` (*tests.test\_instructions.TestInstructions method*), 24  
`test_adc_imm_carry()` (*tests.test\_instructions.TestInstructions method*), 24  
`test_adc_imm_nc()` (*tests.test\_instructions.TestInstructions method*), 24  
`test_adc_imm_nv()` (*tests.test\_instructions.TestInstructions method*), 24  
`test_adc_imm_vc()` (*tests.test\_instructions.TestInstructions method*), 24  
`test_and_abs()` (*tests.test\_instructions.TestInstructions method*), 24

`test_and_imm()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_asl()` (*tests.test\_instructions.TestInstructions* method), 24  
`test_asl_abs()` (*tests.test\_instructions.TestInstructions* method), 23  
`test_bcc_rel()` (*tests.test\_instructions.TestInstructions* method), 24  
`test_bcs_rel()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_beq_rel()` (*tests.test\_instructions.TestInstructions* method), 23  
`test_bit_abs()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_bit_zpg()` (*tests.test\_instructions.TestInstructions* method), 24  
`test_bmi_rel()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_bne_rel()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_bpl_rel()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_bvc()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_bvs()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_clc()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_cld()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_cli()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_clv()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_dec_abs()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_dec_zpg()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_dex()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_dex_xnegative()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_dey()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_dey_ynegative()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_edit_memory_locations()` (*tests.test\_emulator.TestEmulator* method), 23  
`test_eor_abs()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_eor_imm()` (*tests.test\_instructions.TestInstructions* method), 25  
`test_eor_zpg()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_get_ac()` (*tests.test\_memory.TestMemory* method), 26  
`test_get_pc()` (*tests.test\_memory.TestMemory* method), 26  
`test_get_sp()` (*tests.test\_memory.TestMemory* method), 26  
`test_get_sr()` (*tests.test\_memory.TestMemory* method), 26  
`test_get_x()` (*tests.test\_memory.TestMemory* method), 26  
`test_get_y()` (*tests.test\_memory.TestMemory* method), 26  
`test_imm_with_cmp()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_immediate_and_zero()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_inc_abs()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_inx()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_iny()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_jump_abs()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_jump_ind()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_jsr_abs()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_lda_abs()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_lda_imm()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_lda_zpg()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_ldx_abs()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_ldx_imm()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_ldx_zpg()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_ldy_abs()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_ldy_imm()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_ldy_zpg()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_lsr()` (*tests.test\_instructions.TestInstructions* method), 26  
`test_lsr_abs()` (*tests.test\_instructions.TestInstructions* method), 26

method), 26  
 test\_lsr\_carry() (tests.test\_instructions.TestInstructions  
 method), 26  
 test\_lsr\_zpg() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_ora\_abs() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_ora\_imm() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_ora\_zpg() (tests.test\_instructions.TestInstructions  
 method), 27  
 test pha() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_php() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_pla() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_rol() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_rol\_abs() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_rol\_zpg() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_ror() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_ror\_abs() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_ror\_zpg() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_run\_program\_nop()  
 (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_sec() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_sed() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_sei() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_sta\_abs() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_sta\_zpg() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_stx\_abs() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_stx\_zpg() (tests.test\_instructions.TestInstructions  
 method), 27  
 test\_sty\_abs() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_sty\_zpg() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_tax() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_tay() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_tsx() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_txa() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_txs() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_tya() (tests.test\_instructions.TestInstructions  
 method), 28  
 test\_write\_ac() (tests.test\_memory.TestMemory  
 method), 24  
 test\_write\_pc() (tests.test\_memory.TestMemory  
 method), 24  
 test\_write\_sp() (tests.test\_memory.TestMemory  
 method), 24  
 test\_write\_sr() (tests.test\_memory.TestMemory  
 method), 24  
 test\_write\_x() (tests.test\_memory.TestMemory  
 method), 24  
 test\_write\_y() (tests.test\_memory.TestMemory  
 method), 24  
 test\_zeropage() (tests.test\_instructions.TestInstructions  
 method), 28  
 TestEmulator (class in tests.test\_emulator), 23  
 TestEmulator (module), 23  
 TestInstructions (class in tests.test\_instructions),  
 24  
 TestInstructions (module), 24  
 TestMemory (class in tests.test\_memory), 23  
 TestMemory (module), 23  
 tests.test\_emulator (module), 23  
 tests.test\_instructions (module), 24  
 tests.test\_memory (module), 23  
 tsx() (t34.Emulator.Emulator method), 19  
 txa() (t34.Emulator.Emulator method), 19  
 txs() (t34.Emulator.Emulator method), 19  
 ty a() (t34.Emulator.Emulator method), 20

## U

unset\_carry() (t34.Emulator.Emulator method), 20  
 unset\_negative() (t34.Emulator.Emulator  
 method), 20  
 unset\_overflow() (t34.Emulator.Emulator  
 method), 20  
 unset\_zero() (t34.Emulator.Emulator method), 20

## W

write\_AC() (t34.Emulator.Emulator method), 20  
 write\_memory() (t34.Emulator.Emulator method),  
 20  
 write\_PC() (t34.Emulator.Emulator method), 20  
 write\_SP() (t34.Emulator.Emulator method), 20  
 write\_SR() (t34.Emulator.Emulator method), 20  
 write\_X() (t34.Emulator.Emulator method), 20  
 write\_Y() (t34.Emulator.Emulator method), 20



## Z

`zero_isSet()` (*t34.Emulator.Emulator method*), [21](#)