T34 Emulator Documentation

Release 0.1

Chezka Gaddi

Dec 08, 2019

CONTENTS

CHAPTER

ONE

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)

CHAPTER

TWO

DOCUMENTATION FOR THE CODE

2.1 Emulator

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

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.

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

load_program()

Loads the program.

Returns successful read

Return type bool

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

start_emulator()
Starts the emulator and evaluates and executes commands.

2.2 Instructions

class t34.Instructions.**Instructions**
Class that handles all of the instructions to be executed by the T34.

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

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 >= 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 >= 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 >= 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

cpy_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 >= 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

cpy_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 >= 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

eor_abs()
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

eor_imm()
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

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

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

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

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

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

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

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

ror()
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

ror_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

ror_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

sbc_imm()
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

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

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

2.3 Memory

class t34.Memory.**Memory**
Memory class that maintains the T34 registers and memory.

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

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

initialize_registers()
Initialize registers to initial values.

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

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

overflow_isSet()
Checks if overflow bit is set.

Returns: Bool – status of overflow bit

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.

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

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

CHAPTER

THREE

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_jmp_abs()
Test jmp abs instruction.

test_jmp_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.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

L

M

N

O

P

R

S

T

U

W

Z