T34 Emulator Documentation

Release 0.1

Chezka Gaddi

Nov 16, 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 – 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

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

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

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

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.

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

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

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

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

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

read_memory(*start*, *end*)
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

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

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

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_negative()
Set negative bit to 1

set_zero()
Set zero bit to 1

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

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

write_memory(*address*, *data*)
Writes data to a specific memory address.

Parameters

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

2.2 Instructions – auto members

2.3 Memory – auto members

CHAPTER

THREE

TESTING THE PROGRAM

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

```
python3 -m unittest discover
```

3.1 Test Emulator

class t34.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 Instructions

class t34.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.

test_asl()
Test asl instruction.

test_clc()
Test clc instruction.

test_cld()
Test cld instruction.

test_cli()
Test cli instruction.

test_clv()
Test clv 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_inx()
Test inx instruction.

test_iny()
Test iny instruction.

test_lsr()
Test lsr instruction.

test_lsr_carry()
Test lsr instruction with carry.

test_pha()
Test pha instruction.

test_php()
Test php instruction.

test_pla()
Test php instruction.

test_rol()
Test rol instruction.

test_ror()
Test ror 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_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.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

W