TIAS - a TI8x Assembler

Michael K. Pellegrino

January 8, 2021

1 Abstract

Why on Earth would I set out to create an Assembler, Debugger, Programs, and Disassembler for the TI8x calculators?

2 Using the Assembler and Disassembler

The assembler can be executed using the following command:

```
tias input.asm output.8xp
```

Additional options

3 Built-in Functions

- loop
- pusha
- popa
- mem_clear
- user_input, (dont_)check_for_decimal_points, (dont_)check_for_negatives
- store_op1
- hex_to_string
- degree_mode
- radian_mode
- disp_op1
- fully_clear_screen
- convop1b (this should *probably* be `conv_op1` but let's wait until this is all finished to start messing around with the details like that.)

3.1 loop

`loop`: arguments pushed onto the stack before `call &loop`: address of code to be looped, and the number of iterations

```
Example:

    ld hl, &mycode

    push hl

    ld bc, #0x000A

    push bc

    call &loop

    ret

mycode:

    push bc

    pop bc

    ret
```

3.2 pusha

`pusha`: a macro that pushes the registers `af`, `bc`, `de`, and `hl` (in that order) onto the stack.

3.3 popa

`popa`: a macro that pops the registers `hl`, `de`, `bc`, and `af` (in that order) off of the stack.

3.4 mem_clear

`mem_clear`: zeroes out a section of memory

```
Example:

    ld hl, &start_of_region

    ld (&function_mem_clear_address), hl

    ld a, 0x05

    ld (&function_mem_clear_number_of_bytes), a
    call &mem_clear

    ret

start_of_region:

    .db 0x01

    .db 0x02

    .db 0x03

    .db 0x04

    .db 0x05

    .db 0x06; <-- this byte won't get cleared
```

3.5 user_input

`user_input`: gets a numerical value from the user and stores it in the `OP1` and `&FP_user_input`. There are 4 helper functions that go along with this function.

- `dont_check_for_decimal_points`: forces the `user_input` function to stop allowing decimal points
- `check_for_decimal_points`: forces the `user_input` function to begin allowing decimal points (this is the dafault)
- `dont_check_for_negatives`: forces the `user_input` function to stop allowing negative signs
- `check_for_negatives`: forces the `user_input` function to begin allowing negative signs (this is the dafault)

```
Example:

    call &user_input

    call &disp_op1

    call &dont_check_for_decimal_points

    call &dont_check_for_negatives

    call &user_input
    call &disp_op1

    call &check_for_decimal_points

    call &check_for_negatives

    call &user_input

    call &disp_op1

    ret
```

3.6 store_op1

`store_op1`: stores floating point register `OP1` as a variable on the calculator. The 1 parameter is a token (the ASCII value of the letter that is the variable) stored in `&function_store_op1_variabletoken`. Variables A-Z use the hex values 0x41 - 0x5A and θ is 0x5B.

```
Example:

    call &user_input

    ld a, 0x46; 0x46 is the letter F so this

              ;will store the user input in the F variable

    ld (&function_store_op1_variabletoken), a

    call &store_op1

    ret
```

3.7 hex_to_string

`hex_to_string`: Converts a 2 byte hexadecimal value to a displayable string. The 1 parameter is a 2 byte hexadecimal word passed via the stack. The return value is the address of the string. It is returned via stack.

```
Example:

    ld hl, #0xAB12
    push hl

    call &hex_to_string

    pop hl

    bCall(PutS)

    ;;; to print without the 0x

    ld hl, #0xAB12

    call &hex_to_string

    pop hl

    inc hl ; increase hl by 2 bytes

    inc hl ; which put it past the 0x

    bCall(PutS)

    ret
```

3.8 degree_mode and radian_mode

`degree_mode`: puts the calculator into degree mode, and `radian_mode` puts the calculator into radian mode. In `degree_mode`, $\sin(45) = 0.7071...$ and in `radian_mode`, $\sin(\pi/4) = 0.7071....$

```
Example:

    ;;; find the sin(45)

    call &degree_mode

    ld hl, &forty_five

    bCall(Mov9ToOP1)

    bCall(Sin)

    call &disp_op1


    ;;; find the sin(pi/4)

    call &radian_mode
    ld hl, &pi

    bCall(Mov9ToOP1)

    bCall(TimesPt5)

    bCall(TimesPt5)

    bCall(Sin)

    call &disp_op1

    ret

;;; data for the example

.pi

forty_five:

    .db 0x00

    .db 0x81

    .db 0x45

    .db 0x00

    .db 0x00

    .db 0x00

    .db 0x00

    .db 0x00

    .db 0x00
```

3.9 disp_op1

`disp_op1`: Displays the value of `OP1` on the screen.

```
Example:

    ld hl, &pi

    bCall(Mov9ToOP1)
    call &disp_op1
    ret
.pi
```

3.10 fully_clear_screen

`fully_clear_screen`: Completely clears the screen. This function takes no parameters.

```
Example:
    call &fully_clear_screen
    call &disp_op1
    ret
```

3.11 convop1b

`convop1b`: Converts `OP1` into a 2 byte hexadecimal number. There is a system call provided by TI that also does this, but it has a limited input range. It is called `convop1`. This built-in function will allow all decimal values from 0 to 65535. It returns the value in `de`.

```
Example:
    call &user_input
    call &convop1b
    push de
    call &hex_to_string
    pop hl
    bCall(PutS)
    ret
```

4 Registers

- z80 registers
  - `af`
  - `bc`
  - `de`
  - `hl`
  - (also the shadow registers af', bc', de', and hl')
- Floating Point
  - `OP1`
  - `OP2`
  - `OP3`
  - `OP4`
  - `OP5`
  - `OP6`

5 Directives

- `.name`
- `.dw`
- `.db`
- `.str`
- `.chars`
- `.pi`
- `.e`
- `.fp`