



# LM888C

## HARDWARE REFERENCE MANUAL

# **LM80C COLOR COMPUTER & LM80C 64K COLOR COMPUTER HARDWARE REFERENCE MANUAL**

This release covers the  
LM80C Color Computer  
and  
LM80C 64K Color Computer

Latest revision on 30/03/2021

## Copyright notices:

The names “LM80C”, “LM80C Color Computer”, “LM80C BASIC”, and “LM80C DOS”, the “rainbow LM80C” logo, the LM80C schematics, the LM80C sources, and this work belong to Leonardo Miliani, from here on the “owner”.

The “rainbow LM80C” logo and the “LM80C/LM80C Color Computer/LM80C 64K Color Computer” names can not be used in any work without explicit permission of the owner. You are allowed to use the “LM80C” name only in reference to or to describe the LM80C/LM80C 64K Color Computer.

The LM80C and LM80C 64K schematics and source codes are released under the GNU GPL License 3.0 and in the form of "as is", without any kind of warranty: you can use them at your own risk. You are free to use them for any non-commercial use: you are only asked to maintain the copyright notices, to include this advice and the note to the attribution of the original works to Leonardo Miliani, if you intend to re-distribute them. For any other use, please contact the owner.

# Index

|  |    |
|--|----|
| Copyright notices:.....  | 3  |
| 1. THE LM80C COLOR COMPUTER.....                               | 5  |
| 1.1 Why this computer.....                                     | 5  |
| 1.2 LM80C: main features.....                                  | 5  |
| 1.3 LM80C 64K: main features.....                              | 5  |
| 1.4 The 64K version: bank switching.....                       | 6  |
| 1.5 System start-up.....                                       | 6  |
| 2. I/O PORTS.....  | 8  |
| 3. INTERRUPTS & JUMP VECTORS.....                              | 10 |
| 4. RAM REGISTERS.....  | 11 |
| 4.1 REGISTERS FOR LM80C AS PER FIRMWARE REVISION 3.23.....     | 11 |
| 4.2 REGISTERS FOR LM80C 64K AS PER FIRMWARE REVISION 1.16..... | 13 |
| 5. STORING DATA INTO MEMORY.....                               | 22 |
| 5.1 FLOATING POINT REPRESENTATION.....                         | 22 |
| 5.2 How variables and arrays are stored.....                   | 24 |
| 5.3 GOSUB and RETURN usage of the stack.....                   | 25 |
| 5.4 FOR and NEXT usage of the stack.....                       | 25 |
| 6. HOW A BASIC PROGRAM IS STORED IN MEMORY.....                | 26 |
| 7. SERIAL CONFIGURATION.....                                   | 27 |
| 8. VDP SETTINGS.....   | 28 |
| 9. Z80 DAISY CHAIN INTERRUPT PRIORITY.....                     | 29 |
| 10. STATUS LEDs.....   | 30 |
| 11. REFERENCES.....  | 31 |
| 12. USEFUL LINKS.....  | 32 |

# 1. THE LM80C COLOR COMPUTER

## 1.1 Why this computer

The LM80C Color Computer is a home-brew computer designed and programmed by me, Leonardo Miliani, an Italian retro-computer enthusiast, in an effort to have my own, old-style, 80s' computer. It is built upon the Z80, an 8-bit CPU developed in the '70s by Zilog. LM80C name stands for "L"eonardo "M"iliani (Z)"80" "C"olor. The "80" has the double meaning of Z80, to recall the CPU is built on, and 80s, to recall the years of my youth, when the 8-bit computers dominated the home computer market.

## 1.2 LM80C: main features

The LM80C might have been a good computer at that time. In fact, its features are as follow:

- CPU: Zilog Z80B@3.68 MHz
- RAM: 32 KB SRAM
- ROM: 32 KB EEPROM with built-in LM80C BASIC
- Video: TMS9918A with 16 KB VRAM, 256×192 pixels, 15 colors, and 32 sprites
- Audio: Yamaha YM2149F (or General Instruments AY-3-8910) with 3 analog channels, 2×8-bit I/O ports (used to read the external keyboard)
- Serial I/O: 1×Z80 SIO, with serial line up to 57,600 bps
- Parallel I/O: 1×Z80 PIO
- Timer: 1×Z80 CTC
- Compact Flash adapter for CF cards

**IMPORTANT NOTE: due to the better tech. Specifications & performances of the LM80C 64K, the LM80C Color Computer has been DISCONTINUED and it won't be developed anymore. New users should start using the LM80C 64K model while for current users of LM80C it is recommended to switch to the bigger model to get advantage of its features.**

## 1.3 LM80C 64K: main features

The LM80C 64K is almost identical to its little brother. The main difference, as its name reveals, is the amount of RAM, expanded to 64 KB. Another difference is the ability to use 2x 16K banks for VRAM, allowing the VDP to keep 2 different video pages into memory:

- CPU: Zilog Z80B@3.68 MHz
- RAM: 64 KB SRAM
- ROM: 32 KB EEPROM with built-in LM80C BASIC

- Video: TMS9918A with 32 KB VRAM (splitted into 2×16 KB banks), 256×192 pixels, 15 colors and 32 sprites
- Audio: Yamaha YM2149F (or General Instrument AY-3-8910) with 3 analog channels, 2×8-bit I/O ports (used to read the external keyboard)
- Serial I/O: 1×Z80 SIO, with serial line up to 57,600 bps
- Parallel I/O: 1×Z80 PIO
- Timer: 1×Z80 CTC
- Compact Flash adapter for CF cards

## 1.4 The 64K version: bank switching

The LM80C Color Computer version 2 has 64 KB of RAM and 32 KB of (EEP)ROM. Due to the 16-bit address bus limitations of the CPU, only 64 KB can be directly addressed. To go around this limit a sort of bank switching has been implemented.

ROM occupies the first 32 KB of the address space, from \$0000 to \$7FFF. The RAM is formed by 2×16 KB banks, lower and upper bank, or simply bank #0 and bank #1: bank #1 occupies the address space from \$8000 to \$FFFF while bank #0 occupies the same address space of the non-volatile memory but it's not enabled by default. When you power up the system or after a reset, the Z80 jumps to address \$0000, so the CPU have to find ROM memory at bank #0. Now comes in action the bank switching technique implemented: after the start, the CPU executes a little portion of code called “switcher”, that copies the whole firmware from ROM bank #0 to RAM bank #1. Then, it disables the ROM in bank #0 and enables the underlying RAM chip, switching to a real 64 KB RAM memory. After this, it copies back the BASIC firmware from RAM bank #1 to RAM bank #0, so that the firmware goes back to occupy its original location. Only the DOS remains in the higher portion of RAM, because it can be enabled/disabled at startup (see ch. 1.5).

Obviously, the entire RAM is available for user programs since he/she can overwrite the built-in firmware since it runs in a re-writable memory.

## 1.5 System start-up

At startup the Z80 loads the address \$0000 into the PC (Program Counter) and start initializing the HW of the computer. After each peripheral has been set up, the control passes to the bank switcher, which moves the BASIC interpreter and the whole firmware from RAM to ROM. After this, the BASIC interpreter checks if this is a cold start (i.e. after a power-up): if this isn't such case, it asks the user if he/she wants to perform a cold or warm start: the first one initializes the working space like at boot, deleting every possible program still resident in RAM and clearing every variable, while the latter preserves both these data.

If, while the logo has been showed at video, the user presses the **RUN/STOP** key, the firmware re-boot the system again and re-copy the firmware from the ROM into the RAM. This is useful if, some reason (“playing” with some **POKEs** around the system memory), the computer has began unstable.

Another key that can be pressed while the logo is on video is **CTRL:** by pressing it, the LM80C DOS will be disabled and the RAM occupied by its buffers will be freed up, recovering about 4 KB of space.

At the end of the startup process, the control is passed to the BASIC interpreter in direct mode: this means that the computer is able to execute commands as soon as they are entered.

## 2. I/O PORTS

Peripheral I/O chips have their I/O channels mapped at the following logical ports:

PIO:

- PIO data channel A: \$00
- PIO data channel B: \$01
- PIO control channel A: \$02
- PIO control channel B: \$03

CTC:

- CTC channel 0: \$10
- CTC channel 1: \$11
- CTC channel 2: \$12
- CTC channel 3: \$13

SIO:

- SIO data channel A: \$20
- SIO data channel B: \$21
- SIO control channel A: \$22
- SIO control channel B: \$23

VDP:

- VDP data port: \$30
- VDP control port:
  - 32K version: \$32
  - 64K version: \$31

PSG:

- PSG register port: \$40
- PSG data port: \$41

CF card:

- CF data (reg. #0): \$50 (R/W)
- CF error (reg. #1): \$51 (R)
- CF features (reg. #1): \$51 (W)
- CF sector count reg. (reg. #2): \$52 (R/W)
- CF LBA reg. 0 (reg. #3): \$53 (bits 0-7) (R/W)
- CF LBA reg. 1 (reg. #4): \$54 (bits 8-15) (R/W)
- CF LBA reg. 2 (reg. #5): \$55 (bits 16-23) (R/W)
- CF LBA reg. 3 (reg. #6): \$56 (bits 24-27) (R/W)



- CF status (reg. #7): \$57 (R)
- CF command (reg. #7): \$57 (W)

The user can control these chips directly by reading/writing from/to the ports listed above.

### 3. INTERRUPTS & JUMP VECTORS

Several interrupt & jump vectors are stored into ROM:

- \$0000 RESET: Z80 jumps here after a reset or at power-up
- \$0004 INT vector for SIO RX\_CHB\_AVAILABLE interrupt signal
- \$0006 INT vector for SPEC\_RXA\_CONDITION (special receive condition) interrupt signal
- \$0008 RST8: this restart calls a function that sends a char via serial
- \$000C INT vector for SIO RX\_CHA\_AVAILABLE interrupt signal
- \$000E INT vector for SIO SPEC\_RX\_CONDITION (special receive condition) interrupt signal
- \$0010 RST10: this restart jumps to a function that receives a char from the input buffer (serial and/or keyboard)
- \$0018 RST18: jumps to function that checks if a char is available in the input buffer
- \$0040 CTC CH0: jumps to CTC0IV (see below) - unused
- \$0042 CTC CH1: jumps to CTC1IV (see below) - unused
- \$0044 CTC CH2: jumps to CTC2IV (see below) - unused
- \$0046 CTC CH3: jumps to CTC3IV (see below) - used by system
- \$0066 NMI IRQ: jumps to NMIUSR (see below) - unused

## 4. RAM REGISTERS

The LM80C uses some RAM cells to store important information and data.<sup>(1)</sup> By manually writing into these locations the user can alter the functioning of the system, sometimes leading to crashes and/or non-predictable behaviors.

### 4.1 REGISTERS FOR LM80C AS PER FIRMWARE REVISION 3.23

|      |        |       |   |
|------|--------|-------|---|
| 805E | WRKSPC | (3)   | BASIC Work space                            |
| 8061 | NMIUSR | (3)   | NMI exit point routine                      |
| 8064 | USR    | (3)   | "USR (x)" jump                              |
| 8067 | OUTSUB | (1)   | "out p,n"                                   |
| 8068 | OTPORT | (2)   | Port (p)                                    |
| 806A | DIVSUP | (1)   | Division support routine                    |
| 806B | DIV1   | (4)   | <- Values                                   |
| 806F | DIV2   | (4)   | <- to                                       |
| 8073 | DIV3   | (3)   | <- be                                       |
| 8076 | DIV4   | (2)   | <-inserted                                  |
| 8078 | SEED   | (35)  | Random number seed                          |
| 809B | LSTRND | (4)   | Last random number                          |
| 809F | INPSUB | (1)   | INP A,(x) Routine                           |
| 80A0 | INPORT | (2)   | PORT (x)                                    |
| 80A2 | LWIDTH | (1)   | Terminal width                              |
| 80A3 | COMMAN | (1)   | Width for commas                            |
| 80A4 | NULFLG | (1)   | Null after input byte flag                  |
| 80A5 | CTLOFG | (1)   | Control "0" flag                            |
| 80A6 | CHKSUM | (2)   | Array load/save check sum                   |
| 80A8 | NMIFLG | (1)   | Flag for NMI break routine                  |
| 80A9 | BRKFLG | (1)   | Break flag                                  |
| 80AA | RINPUT | (3)   | Input reflection                            |
| 80AD | STRSPC | (2)   | Pointer to bottom (start) of string space   |
| 80AF | LINEAT | (2)   | Current line number                         |
| 80B1 | HLPLN  | (2)   | Current line with errors                    |
| 80B3 | KEYDEL | (1)   | delay before key auto-repeat starts         |
| 80B4 | AUTOKE | (1)   | Delay for key auto-repeat                   |
| 80B5 | FNKEYS | (128) | default text of FN keys                     |
| 8135 | BASTXT | (3)   | Pointer to start of BASIC program in memory |
| 8138 | BUFFER | (5)   | Input buffer                                |
| 813D | STACK  | (85)  | Initial stack                               |
| 8192 | CURPOS | (1)   | Character position on line                  |
| 8193 | LCRFLG | (1)   | Locate/Create flag for DIM statement        |
| 8194 | TYPE   | (1)   | Data type flag                              |
| 8195 | DATFLG | (1)   | Literal statement flag                      |
| 8196 | LSTRAM | (2)   | Last available RAM location usable by BASIC |
| 8198 | DOSBFR | (2)   | Pointer to start of temp. DOS buffer        |
| 819A | IOBUFF | (2)   | Pointer to start of I/O buffer used by DOS  |
| 819C | DOSER  | (1)   | DOS error                                   |
| 819D | TMPDBF | (36)  | Secondary buffer for DOS                    |
| 81C0 | TMSTPT | (2)   | Temporary string pointer                    |
| 81C2 | TMSTPL | (12)  | Temporary string pool                       |

|      |             |      |   |
|------|-------------|------|---|
| 81CE | TMPSTR      | (4)  | Temporary string                          |
| 81D2 | STRBOT      | (2)  | Bottom of string space                    |
| 81D4 | CUR0PR      | (2)  | Current operator in EVAL                  |
| 81D6 | LOOPST      | (2)  | First statement of loop                   |
| 81D8 | DATLIN      | (2)  | Line of current DATA item                 |
| 81DA | FORFLG      | (1)  | "FOR" loop flag                           |
| 81DB | LSTBIN      | (1)  | Last byte entered                         |
| 81DC | READFG      | (1)  | Read/Input flag                           |
| 81DD | BRKLIN      | (2)  | Line of break                             |
| 81DF | NXTOPR      | (2)  | Next operator in EVAL                     |
| 81E1 | ERRLIN      | (2)  | Line of error                             |
| 81E3 | CONTAD      | (2)  | Where to CONTInue                         |
| 81E5 | TMRCNT      | (4)  | TMR counter for 1/100 seconds             |
| 81E9 | CTC0IV      | (3)  | CTC0 interrupt vector                     |
| 81EC | CTC1IV      | (3)  | CTC1 interrupt vector                     |
| 81EF | CTC2IV      | (3)  | CTC2 interrupt vector                     |
| 81F2 | CTC3IV      | (3)  | CTC3 interrupt vector                     |
| 81F5 | SCR_SIZE_W  | (1)  | screen width                              |
| 81F6 | SCR_SIZE_H  | (1)  | screen height                             |
| 81F7 | SCR_MODE    | (1)  | screen mode                               |
| 81F8 | SCR_NAM_TB  | (2)  | video name table address                  |
| 81FA | SCR_CURS_X  | (1)  | cursor X                                  |
| 81FB | SCR_CURS_Y  | (1)  | cursor Y                                  |
| 81FC | SCR_CUR_NX  | (1)  | new cursor X position                     |
| 81FD | SCR_CUR_NY  | (1)  | new cursor Y position                     |
| 81FE | SCR_ORG_CHR | (1)  | original char positioned under the cursor |
| 81FF | CRSR_STATE  | (1)  | state of cursor                           |
| 8200 | LSTCSRSTA   | (1)  | last cursor state                         |
| 8201 | PRNTVIDEO   | (1)  | print on video buffer                     |
| 8202 | CHR4VID     | (1)  | char for video buffer                     |
| 8203 | FRGNDCLR    | (1)  | foreground color                          |
| 8204 | BKGNDCLR    | (1)  | background color                          |
| 8205 | TMPBFR1     | (2)  | word for general purposes use             |
| 8207 | TMPBFR2     | (2)  | word for general purposes use             |
| 8209 | TMPBFR3     | (2)  | word for general purposes use             |
| 820B | TMPBFR4     | (2)  | word for general purposes use             |
| 820D | VIDEOBUFF   | (40) | temp. video buffer                        |
| 8235 | VIDTMP1     | (2)  | additional temp. video buffer             |
| 8237 | VIDTMP2     | (2)  | additional temp. video buffer             |
| 8239 | CHASNDDTN   | (2)  | sound Ch.A duration                       |
| 823B | CHBSNDDTN   | (2)  | sound Ch.B duration                       |
| 823D | CHCSNDDTN   | (2)  | sound Ch.C duration                       |
| 823F | KBDNPT      | (1)  | temp. location for keyboard inputs        |
| 8240 | KBTMP       | (1)  | temp. location used by keyboard scanner   |
| 8241 | TMPKEYBFR   | (1)  | temp. location used for last key pressed  |
| 8242 | LASTKEYPRSD | (1)  | code of last key pressed                  |
| 8243 | STATUSKEY   | (1)  | status key, used for auto-repeat          |
| 8244 | KEYTMR      | (2)  | timer used for auto-repeat key            |
| 8246 | CONTROLKEYS | (1)  | flags for control keys                    |
| 8247 | SERIALS_EN  | (1)  | serial ports status                       |
| 8248 | SERABITS    | (1)  | serial port A data bits                   |
| 8249 | SERBBITS    | (1)  | serial port B data bits                   |
| 824A | DOS_EN      | (1)  | DOS enable/disable (1/0)                  |
| 824B | PROGND      | (2)  | End of program                            |
| 824D | VAREND      | (2)  | End of variables                          |
| 824F | ARREND      | (2)  | End of arrays                             |

|      |        |                                  |
|------|--------|----------------------------------|
| 8251 | NXTDAT | (2) Next data item               |
| 8253 | FNRGNM | (2) Name of FN argument          |
| 8255 | FNARG  | (4) FN argument value            |
| 8259 | FPREG  | (3) Floating point register      |
| 825C | FPEXP  | (1) Floating point exponent      |
| 825D | SGNRES | (1) Sign of result               |
| 825E | PBUFF  | (13) Number print buffer         |
| 826B | MULVAL | (3) Multiplier                   |
| 826E | PROGST | (100) Start of program text area |
| 82D2 | STLOOK | Start of memory test             |

## 4.2 REGISTERS FOR LM80C 64K AS PER FIRMWARE REVISION 1.16

|      |        |   |
|------|--------|---|
| 53D8 | WRKSPC | (3) BASIC Work space                            |
| 53DB | NMIUSR | (3) NMI exit point routine                      |
| 53DE | USR    | (3) "USR (x)" jump                              |
| 53E1 | OUTSUB | (1) "out p,n"                                   |
| 53E2 | OTPORT | (2) Port (p)                                    |
| 53E4 | DIVSUP | (1) Division support routine                    |
| 53E5 | DIV1   | (4) <- Values                                   |
| 53E9 | DIV2   | (4) <- to                                       |
| 53ED | DIV3   | (3) <- be                                       |
| 53F0 | DIV4   | (2) <-inserted                                  |
| 53F2 | SEED   | (35) Random number seed                         |
| 5415 | LSTRND | (4) Last random number                          |
| 5419 | INPSUB | (1) #INP (x)" Routine                           |
| 541A | INPORT | (2) PORT (x)                                    |
| 541C | LWIDTH | (1) Terminal width                              |
| 541D | COMMAN | (1) Width for commas                            |
| 541E | NULFLG | (1) Null after input byte flag                  |
| 541F | CTLOFG | (1) Control "O" flag                            |
| 5420 | CHKSUM | (2) Array load/save check sum                   |
| 5422 | NMIFLG | (1) Flag for NMI break routine                  |
| 5423 | BRKFLG | (1) Break flag                                  |
| 5424 | RINPUT | (3) Input reflection                            |
| 5427 | STRSPC | (2) Pointer to bottom (start) of string space   |
| 5429 | LINEAT | (2) Current line number                         |
| 542B | HLPLN  | (2) Current line with errors                    |
| 542D | KEYDEL | (1) delay before key auto-repeat starts         |
| 542E | AUTOKE | (1) Delay for key auto-repeat                   |
| 542F | FNKEYS | (128) default text of FN keys                   |
| 54AF | BASTXT | (3) Pointer to start of BASIC program in memory |
| 54B2 | BUFFER | (5) Input buffer                                |
| 54B7 | STACK  | (85) Initial stack                              |
| 550C | CURPOS | (1) Character position on line                  |
| 550D | LCRFLG | (1) Locate/Create flag for DIM statement        |
| 550E | TYPE   | (1) Data type flag                              |
| 550F | DATFLG | (1) Literal statement flag                      |

|      |             |   |
|------|-------------|---|
| 5510 | LSTRAM      | (2) Last available RAM location usable by BASIC |
| 5512 | DOSER       | (1) DOS error                                   |
| 5513 | TMPDBF      | (36) Secondary buffer for DOS                   |
| 5537 | TMSTPT      | (2) Temporary string pointer                    |
| 5539 | TMSTPL      | (12) Temporary string pool                      |
| 5545 | TMPSTR      | (4) Temporary string                            |
| 5549 | STRBOT      | (2) Bottom of string space                      |
| 554B | CUOPR       | (2) Current operator in EVAL                    |
| 554D | LOOPST      | (2) First statement of loop                     |
| 554F | DATLIN      | (2) Line of current DATA item                   |
| 5551 | FORFLG      | (1) "FOR" loop flag                             |
| 5552 | LSTBIN      | (1) Last byte entered                           |
| 5553 | READFG      | (1) Read/Input flag                             |
| 5554 | BRKLIN      | (2) Line of break                               |
| 5556 | NXTOPR      | (2) Next operator in EVAL                       |
| 5558 | ERRLIN      | (2) Line of error                               |
| 555A | CONTAD      | (2) Where to CONTInue                           |
| 555C | TMRCNT      | (4) TMR counter for 1/100 seconds               |
| 5560 | CTC0IV      | (3) CTC0 interrupt vector                       |
| 5563 | CTC1IV      | (3) CTC1 interrupt vector                       |
| 5566 | CTC2IV      | (3) CTC2 interrupt vector                       |
| 5569 | CTC3IV      | (3) CTC3 interrupt vector                       |
| 556C | SCR_SIZE_W  | (1) screen width                                |
| 556D | SCR_SIZE_H  | (1) screen height                               |
| 556E | SCR_MODE    | (1) screen mode                                 |
| 556F | SCR_NAM_TB  | (2) video name table address                    |
| 5571 | SCR_CURS_X  | (1) cursor X                                    |
| 5572 | SCR_CURS_Y  | (1) cursor Y                                    |
| 5573 | SCR_CUR_NX  | (1) new cursor X position                       |
| 5574 | SCR_CUR_NY  | (1) new cursor Y position                       |
| 5575 | SCR_ORG_CHR | (1) original char under the cursor              |
| 5576 | CRSR_STATE  | (1) state of cursor (1=on, 0=off)               |
| 5577 | LSTCSRSTA   | (1) last cursor state                           |
| 5578 | PRNTVIDEO   | (1) print on video buffer                       |
| 5579 | CHR4VID     | (1) char for video buffer                       |
| 557A | FRGNDCLR    | (1) foreground color                            |
| 557B | BKGNDCLR    | (1) background color                            |
| 557C | TMPBFR1     | (2) word for general purposes use               |
| 557E | TMPBFR2     | (2) word for general purposes use               |
| 5580 | TMPBFR3     | (2) word for general purposes use               |
| 5582 | TMPBFR4     | (2) word for general purposes use               |
| 5584 | VIDEOBUFF   | (40) temp. video buffer                         |
| 55AC | VIDTMP1     | (2) temporary video word                        |
| 55AE | VIDTMP2     | (2) temporary video word                        |
| 55B0 | CHASNDDTN   | (2) sound Ch.A duration (in 1/100s)             |
| 55B2 | CHBSNDDTN   | (2) sound Ch.B duration (in 1/100s)             |
| 55B4 | CHCSNDDTN   | (2) sound Ch.C duration (in 1/100s)             |
| 55B6 | KBDNPT      | (1) temp. location for keyboard inputs          |
| 55B7 | KBTMP       | (1) temp. location used by keyboard scanner     |
| 55B8 | TMPKEYBFR   | (1) temp buffer for last key pressed            |

|      |             |                                      |
|------|-------------|--------------------------------------|
| 55B9 | LASTKEYPRSD | (1) last key code pressed            |
| 55BA | STATUSKEY   | (1) status key, used for auto-repeat |
| 55BB | KEYTMR      | (2) timer used for auto-repeat key   |
| 55BD | CONTROLKEYS | (1) flags for control keys           |
| 55BE | SERIALS_EN  | (1) serial ports status              |
| 55BF | SERABITS    | (1) serial port A data bits          |
| 55C0 | SERBBITS    | (1) serial port B data bits          |
| 55C1 | DOS_EN      | (1) DOS enable/disable (1/0)         |
| 55C2 | PROGND      | (2) End of program                   |
| 55C4 | VAREND      | (2) End of variables                 |
| 55C6 | ARREND      | (2) End of arrays                    |
| 55C8 | NXTDAT      | (2) Next data item                   |
| 55CA | FNRGNM      | (2) Name of FN argument              |
| 55CC | FNARG       | (4) FN argument value                |
| 55D0 | FPREG       | (3) Floating point register          |
| 55D3 | FPEXP       | (1) Floating point exponent          |
| 55D4 | SGNRES      | (1) Sign of result                   |
| 55D5 | PBUFF       | (13) Number print buffer             |
| 55E2 | MULVAL      | (3) Multiplier                       |
| 55E5 | PROGST      | (100) Start of program text area     |
| 5649 | STLOOK      | Start of memory test                 |

WRKSPC: workspace. This is a jump to “Warm start” routine

USR: user-defined function USR(X). Before to call it, locations USR+\$01 and USR+\$02 must be filled up with the address of the user routine. At startup, the default is a call to an “Illegal function call” error.

OUTSUB: out sub-routine. Since the i8080 didn’t have the “OUT(c),r” instruction, this was a skeleton for such statement. Now it’s a sub-routine to write to a specific output port.

OTPORT: output port “c” for the above routine.

DIVSUP: skeleton routine used for division. Since there aren’t enough register to store dividend, divisor and quotient, the divisor is loaded into this routine, to leave registers free for dividend and quotient.

SEED: seed for random number generator and table for floating point values used by RND function.

LSTRND: last random number is stored (available by RND(0)).

INPSUB: input sub-routine. Since the i8080 didn’t have the “IN r,(c)” instruction, this was a routine that read from an input port. Actually, just a sub-routine for “IN” statement.

INPORT: input port “c” for the above routine.

|          |   |
|----------|---|
| NULLS:   | nulls. Number of null chars printed after a carriage return. The original NASCOM BASIC had the command “NULL” to change this value, but now it can only be changed with a “POKE”.   |
| LWIDTH:  | terminal width, set by “WIDTH”.   |
| COMMAN:  | width of terminal for printing with commas. Since “WIDTH” does set LWIDTH but does NOT set COMMAN, the only way to get commas spacing work correctly is just using a POKE.  |
| NULFLG:  | null after input flag. Reminiscence of old terminal computers, where this command was used to set the number of null chars to send to teletype before printing any char.  |
| CTLOFH:  | flag for Control+”O”. When this flag is set, no output will be sent to the terminal.  |
| LINEESC: | lines counter. Initially loaded with the value of LINESM, it is decremented after every line. When it reaches zero, the BASIC interpreter stops waiting for a char from the keyboard.   |
| LINESN:  | lines number. Used for LINEESC. It can be changed with a POKE.  |
| CHKSUM:  | checksum used for array load/save (NASCOM BASIC legacy).  |
| NMIFLG:  | Non-Maskable Interrupt flag. This is used to inform the BASIC that it has been interrupted by an NMI.   |
| BRKFLG:  | break flag, to let BASIC know that the break key was pressed.   |
| RINPUT:  | Reflection for “INPUT” routine. When an “INPUT” instruction is encountered, BASIC jumps to the input routine pointed by this jump. Default is jump to “TTYLIN”, aka get a line by character. This can be changed to “CRLIN” (output CR/LF and get a line), or “GETLIN” (no CR/LF, get a line).  |
| STRSPC:  | start of string space pointer, the area where BASIC stores strings. By default, it is set 100 bytes below the end of memory but this value can be changed by the “CLEAR n” statement.   |
| LINEAT:  | current line number. This pair of bytes contains the value of the current line being executed. A value of -1 means that BASIC is executing a statement in direct mode. A value of -2 means that the computer has been reset and it’s executing the routine to calculate the memory size or, in case it can not determine the amount of RAM, the “Memory size?” routine. If the user doesn’t input a number, -2 instructs the error routine that an error has occurred and that a cold start must be executed. |
| HLPLN:   | help line. Current line that has raised an error during the execution of a BASIC program. This value is read by “HELP” statement, and reset after a program restart (“RUN”) or by another error in direct mode.   |
| KEYDEL:  | delay before the key auto-repeat starts repeating the pressed key.  |
| AUTOKE:  | Delay for key auto-repeat between two prints.   |



|         |   |
|---------|---|
| FNKEYS: | function keys. This area stores the text of function keys. 8 function keys are present and can be individually programmed with user defined statements, whose length can be up to 16 chars. Please refer to LM80C BASIC Reference Manual” to know the pre-configured texts.   |
| BASTXT: | BASIC program text. Pointer to where the BASIC is stored in memory. Usually this reflects the contents of PROGST and both point to the BASIC program area, but it can be changed by the user is a program is loaded elsewhere into RAM.   |
| BUFFER: | input buffer. 88 char buffer where the system stores all the input from the keyboard or the serial line.  |
| STACK:  | temporary stack used during boot of system.   |
| CURPOS: | cursor position. This value keeps the cursor position through the current line and it is incremented every time a new char has been inserted. It is the value returned by “POS(x)” statement . A press on the “RETURN” key resets this value.   |
| LCRFLG: | Locate/Create flag. Value used by the variable search routine to see if it’s in a “DIM” statement or not, so that it can determine if it has to locate or create the specified array.   |
| TYPE:   | type of data of the current expression. A zero value stands for a numeric type, while a non-zero value for a string.  |
| DATFLG: | literal statement flag. It’s used by the BASIC to know if it’s pointing at a literal statement such as a quoted string, a “REM” or a “DATA” statement.  |
| LSTRAM: | last available RAM pointer. Address of last location available to BASIC. It can be changed by “CLEAR n” statement.  |
| DOSBFR: | pointer to beginning of a 32-byte temporary buffer used by DOS in disk input/output operations. The DOS buffer is stored just below the I/O buffer.   |
| IOBUFF: | pointer to beginning of a 512-byte buffer used by DOS to store a sector loaded by disk or to assemble data to save into a sector. The I/O is stored in the highest portion of RAM (from \$FFFF downwards).  |
| DOSER:  | error returned by DOS functions.  |
| TMPDBF: | secondary 36-byte buffer used by DOS to execute statements.   |
| TMSTPT: | temporary string pointer used to point a string into the temporary string pool.   |
| TMSTPL: | temporary string pool. This pool contains 4 temporary strings that are created by string statements like “LEFT\$” and others.   |
| TMPSTR: | temporary string. This is a temporary area where BASIC stores blocks of bytes that reference to the strings being constructed. Every string block consists of 4 bytes: the first byte stores the length of string; the second byte is not used; the last two bytes form the pointer to the location in memory where the string is stored. |

|         |   |
|---------|---|
| STRBOT: | bottom of string space. This value points to the bottom of the string being used. Each time a new string is being formed, it is moved into the string area below this pointer that the pointer is decremented and moved below the new string. If there is not enough space for the new string, than a “garbage collector” is called to remove unused strings from the string space. If, after this cleaning, there is still not enough room, then an “Out of string space error” is raised. |
| CUROPR: | current operator address. Pointer to the current operator being evaluated in EVAL. This pointer is used to free the CPU register used to point to the current operator being analyzed and to avoid to store it into the stack, so that both can be used to simplify the evaluation of the operator.   |
| LOOPST: | loop start address. Pointer to the first statement in the FOR loop being constructed. This address is later moved into the FOR block on the stack .   |
| DATLIN: | data statement line number. This register contains the line number of the current DATA statement pointer. It’s used by DATSNR to report to the user the line where a DATA error has occurred.   |
| FORFLG: | “FOR”/”FN” flag. Flag to tell what GETVAR is expected to find:<br>\$00: a variable or array element<br>\$01: an array name<br>\$64: a variable only<br>\$80: an FN function   |
| LSTBIN: | last byte entered. Flag being set whenever any input is made into the BUFFER. The RETURN routine first checks this byte to see if a GOSUB was entered into direct mode, and if so checks this flag. If the flag is set, then this means that an INPUT statement has been encountered, and so after RETURN is executed, BUFFER contains garbage and the system returns into direct mode.   |
| READFG: | read/input flag. This flag tells the READ/INPUT routine what’s the source of the data being read. If the flag is zero, then an INPUT is being executed, otherwise it’s a READ reading from a DATA statement.  |
| BRKLIN: | break line pointer. Address of the line where a break occurred. This value is used by CONT to know where to continue the execution of the program.  |
| NXTOPR: | next operator address. Pointer into the expression being evaluated by EVAL to know where the execution is in the string.  |
| ERRLIN: | line number of break. This register contains the line number where a break occurred. Used by CONT to know the line number where to continue from.   |
| CONTAD: | continue address. Address of the statement where CONT will continue.  |

|              |   |
|--------------|---|
| TMRCNT:      | timer counter. This is a 32-bit hundredths of a second counter used as a sys-tick timer to temporize events such as sound duration, pauses and other. Its value is incremented every 1/100 s and can be read by the TMR statement.            |
| CTCxIV:      | CTC interrupt vectors. Interrupt vectors that can be changed to point the CTC interrupts to specifics interrupt service routines. CTC3IV is used by the kernel of the computer to temporize some jobs (see TMRCNT above).                     |
| SCR_SIZE_W:  | screen width. This register stores the screen width of the current screen mode. See chap. 8 for more details.   |
| SCR_SIZE_H:  | screen height. This register stores the screen height of the current screen mode. See chap. 8 for more details.   |
| SCR_MODE:    | screen mode. Current screen mode. This value reflects the mode set by SCREEN statement.   |
| SCR_NAM_TB:  | screen name table address. VRAM address of the name table being used by the current screen mode. See chap. 8 for more details.  |
| SCR_CURS_X:  | current cursor X coordinate. Keeps the current horizontal coordinate of the cursor in a text mode (screen 0, 1, & 4).   |
| SCR_CURS_Y:  | current cursor Y coordinate. Keeps the current vertical coordinate of the cursor in a text mode (screen 0, 1, & 4).   |
| SCR_CUR_NX:  | new cursor X coordinate. Keeps the horizontal coordinate of the video cell that the cursor is going to occupy when being moved.   |
| SCR_CUR_NY:  | new cursor Y coordinate. Keeps the vertical coordinate of the video cell that the cursor is going to occupy when being moved.   |
| SCR_ORG_CHR: | original char under the cursor. Register used to store the original char present in the video cell currently occupied by the cursor. It is used to restore the char during cursor flashing or when the cursor is moved into another location. |
| CRSR_STATE:  | cursor state. Flag used to tell BASIC if the cursor is visible (1) or not (0).  |
| LSTCSRSTA:   | last cursor state. Flag used to store the current cursor state, i.e. when executing PRINT statements or when the cursor is moved around the screen.   |
| PRNTVIDEO:   | print on video buffer flag. Used to tell the BASIC if keys being pressed can be echoed on the screen or not. Usually, printing on video is off when in indirect mode (i.e. when a program is being executed).                                 |
| CHR4VID:     | char for video buffer. Temporary buffer that stores a char that must be printed on the screen.  |
| FRGNDCLR:    | foreground color. Foreground color set by SCREEN statement. The default value can also be changed with COLOR statement. Used to print chars on  |

screen or to draw/plot figures/pixels on the graphic screen when no color is being specified.

|              |   |
|--------------|---|
| BKGNDCLR:    | background color. Background color set by SCREEN statement. The default value can also be changed with COLOR statement. Used for the background of the chars being printed on screen or to color the empty area of the graphic screen |
| TMPBFRx:     | temporary buffer. 4 words used by the kernel and BASIC as temporary buffers.  |
| VIDEOBUFF:   | temporary video buffer. 40 RAM cells used by the kernel and BASIC for video scrolling in text modes and as temporary buffer.  |
| VIDTMPx:     | temporary video buffer. 2 additional buffers used by the firmware and BASIC.  |
| CHASNDDTN:   | channel A sound duration. Duration of the sound being generated by ch. A of the PSG, in hundredths of a second. This value is set by the SOUND statement and decremented by the kernel.   |
| CHBSNDDTN:   | channel B sound duration. Same as above, but for ch. B.   |
| CHCSNDDTN:   | channel C sound duration. Same as above, but for ch. C.   |
| KBDNPT:      | keyboard input. Temporary buffer to store the char being input through the keyboard.  |
| KBTMP:       | temporary keyboard scanner. Register used by the keyboard scanner routine to store the key row of the key matrix when a key is being pressed.   |
| TMPKEYBFR:   | last key pressed buffer. Temporary buffer used to store the code of the last key being pressed.   |
| LASTKEYPRSD: | code of the last key pressed. Code of the last key being pressed.   |
| STATUSKEY:   | used by the key auto-repeat function. Stores the current state of the repeating (0 if no key is pressed, 1 if a key is being pressed for the first time, 2 if the key still continues to be pressed)                                  |
| KEYTMR:      | timer used by the key auto-repeat function to activate the auto-repeat function and the delay between 2 key prints.   |
| CONTROLKEYS: | flag for control keys. Flag used by the keyboard scanner to keep track of the control keys being pressed.   |
| SERIALS_EN:  | serial lines enabled. Status of the serial lines. Bit 0 reflects the status of serial line 0, while bit 1 reflects the status of serial line 1: 0 means line OFF, 1 means line ON.  |
| SERABITS:    | serial port A data bits. Register used to store the data configuration bits used to set the serial port A, used by the kernel.  |

|           |  |
|-----------|--|
| SERBBITS: | serial port B data bits. Register used to store the data configuration bits used to set the serial port B, used by the kernel.   |
| DOS_EN:   | I/O DOS enabled/disabled (1/0). When the computer's logo is shown at boot, by pressing the SHIFT key the user can disable the I/O buffer used for DOS operations, to recover 512 bytes of free RAM.  |
| PROGND:   | program end. Address of the byte after the end of the BASIC program text stored in RAM.  |
| VAREND:   | variables end. Address of the byte after the last variable stored in RAM.  |
| ARREND:   | arrays end. Address of the byte after the last array stored in RAM.  |
| NXTDAT:   | next data item. Address of the next DATA item to be read by READ.  |
| FNRGNM:   | FN argument name. Name of the argument of the current FN function. If an FN function calls another FN function, then this value is stored on the stack.  |
| FNARG:    | FN function argument. Floating point value of the argument of the current FN function. If an FN function calls another FN function, then this value is stored on the stack together with its name.   |
| FPREG:    | floating point register. Floating point value for the current value. These 3 bytes contains the mantissa.  |
| FPEXP:    | floating point exponent. Floating point value for the current value. This byte contains the exponent. See chap. 5 for floating point representation in memory.   |
| SGNRES:   | sign of the result. This register contains the sign of the result for multiplication. Both multiplicand and multiplier are tested and if their signs are different, then the product will be negative, otherwise it will be positive.  |
| PBUFF:    | number print buffer. Temporary buffer used by NUMASC to store a floating point that has to be converted into ASCII for PRINT or STR\$ statements   |
| MULVAL:   | multiplier. This 24-bit register contains the multiplier of a multiplication because there are not enough registers to store the multiplier, the multiplicand, and product at the same time.   |
| PROGST:   | program start. This is the byte before the first line of a program and it MUST be zero to tell the execution driver that the next line is to be executed. See chap. 6 to see how a BASIC program is stored into RAM.   |
| STLOOK:   | start of memory test. Address from which the memory test executed after a reset or when the system is powered up, start to look for the top of the RAM. This address is 100 bytes above the program text area so that the kernel can have at least some bytes to create the stack and store a very little program. |

## 5. STORING DATA INTO MEMORY

### 5.1 FLOATING POINT REPRESENTATION

Floating point numbers are represented in memory using the Microsoft Binary Format (MBF), introduced by Microsoft in its first Microsoft BASIC in 1975. It uses a 24-bit mantissa, of which 23 bits are used for the mantissa itself and 1 bit for its sign, and an 8-bit base-2 exponent, for a total of 32 bits (4 bytes). There is always a “1” bit implied to the left of the mantissa so that the exponent is encoded with a bias of 128, so that exponents from -127 to -1 are represented by values \$01~\$7F (1~127), while exponents 0~127 are represented by \$80~\$FF (128~255). Exponent set to zero is a special case, representing the whole number being zero.<sup>(1)(2)</sup>

MBF representation:

```
m= mantissa bits
s=mantissa sign
x=exponent
```

| Byte 1    | Byte 2     | Byte 3     | Byte 4     |
|-----------|------------|------------|------------|
| smmmmmmmm | mmmmmmmmmm | mmmmmmmmmm | xxxxxxxxxx |

Let's try to represent the 35.25 in MBF. Firstly, the number must be converted in binary format. To convert the integer part, divide the number repeatedly by 2, writing the remainder to the right, until the quotient becomes 0:

| Div. | Quot. | Rem. |
|------|-------|------|
| 35/2 | = 17  | 1    |
| 17/2 | = 8   | 1    |
| 8/2  | = 4   | 0    |
| 4/2  | = 2   | 0    |
| 2/2  | = 1   | 0    |
| 1/2  | = 0   | 1    |

Now write the remainders from bottom to top. The results is the binary representation of the integer part of the number. 35 in base-2 representation is:

100011

To convert the fractional part, multiply it repeatedly by 2 until it becomes 0. Stop after a number of steps, if the fractional part does not become zero:

|      |     |        |     |   |
|------|-----|--------|-----|---|
| 0.25 | x 2 | = 0.50 | → 0 |   |
| 0.50 | x 2 | = 1.00 | → 1 | ← stop here, as the fract. Part is now 0. |

From top to bottom, write the integer parts of the results to the fractional part of the base-2 number. So 0.25<sub>10</sub> becomes 0.01<sub>2</sub>. Finally, the complete number is:

35.25 → 100011.01

Now, consider the base-2 exponent, so that the above is the same as:

```
100011.01 * 2^00000000
```

The binary point is moved to the left most position, so that now precedes the first “1”.

This is the actual situation:

```
.10001101
```

Since the point was moved left 6 times dividing the number by  $2^6$ , now we must add 6 to the exponent to re-multiply by  $2^6$ :

```
.10001101 * 2^00000110
```

Since the bit to the right of the point is always 1, we can use this bit to store the sign of the number: 0 for a positive number, and 1 for a negative number. So +35.25 is stored as:

```
.00001101 * 2^00000110
```

In 24 bits the above is:

```
.000011010000000000000000 * 2^00000110
```

Now, 128 is added to the exponent so that overflows and underflows can be detected easily. So the number actually is:

```
00001101 00000000 00000000 10000110    binary
0   D   0   0   0   0   8   6            hex
```

The bytes of the mantissa are stored in reverse order. So, this gives the followings:

| Decimal | Binary                              | Hexadecimal |
|---------|-------------------------------------|-------------|
| +35.25  | 00001101 00000000 00000000 10000110 | 00 00 0D 86 |
| -32.25  | 10001101 00000000 00000000 10000110 | 00 00 8D 86 |

Other examples:

| Decimal       | Binary                              | Hexadecimal |
|---------------|-------------------------------------|-------------|
| 0             | 00000000 00000000 00000000 00000000 | 00 00 00 00 |
| 1             | 00000000 00000000 00000000 10000001 | 00 00 00 81 |
| 10            | 00100000 00000000 00001000 10000100 | 20 00 00 84 |
| PI/2 (1.5708) | 11011111 00001111 01001001 10000001 | DB 0F 49 81 |

## 5.2 How variables and arrays are stored

Variables such as AB, AB\$, and FN AB are all stored in the variable area of memory. The start address of such area is held in (\$PROGND) while the end address is held in (\$VAREND).

Let make some examples. Let's assume that AB=10, AB\$="HELLO", and FN AB(XY) have been defined by the user. The memory would then look like this:

```
AB
$42 $41      Name of AB in reverse ($42 $41 = "B" "A")
$00 $00 $20 $84 Floating point value for 10
```

```
AB$
$C2 $41      Name for AB$ in reverse ($C2 is "B" with bit 7 set)
$04          Length f string (4 chars)
??          This byte is unused fro strings
LL  HH      Address where "HELLO" is to be found (in Little Endian
fomat)
```

```
FN AB
$42 $C1      Name of FN AB in reverse ($C1 is "A" with bit 7 set)
LL  HH      Address of function (the portion after "=")
$59 $58     Argument name in reverse ($59 $58 = "Y" "X")
```

Arrays such as AB(1,3) are stored in the array area. The start address of this area is held in (\$VAREND) while the end address is held in (\$ARREND).

Let's assume that DIM AB(1,3), AB\$(3,1) had been encountered. Then the memory looks like this:

```
AB(1,3)
$42 $41      Name of array "AB" in reverse (see above)
$25 $00      Bytes used for array in Little Endian ($0025 = 37)
$02          2 dimensions
$04 $00      Size of 2nd dimension including zero element
$02 $00      Size of 1st element including zero element
$00 $00 $00 $00 AB(0,0)
$00 $00 $00 $00 AB(1,0)
$00 $00 $00 $00 AB(0,1)
$00 $00 $00 $00 AB(1,1)
$00 $00 $00 $00 AB(0,2)
$00 $00 $00 $00 AB(1,2)
$00 $00 $00 $00 AB(0,3)
$00 $00 $00 $00 AB(1,3)
```

```
AB$(1,3)
$C2 $41      Name of FN AB in reverse ($C1 is "A" with bit 7 set)
$25 $00      Bytes used for array ($0025 = 37)
$02          2 dimensions
$02 $00      Size of 2nd element including zero element
$04 $00      Size of 2nd element including zero element
$00 $00 $00 $00 AB(0,0)
```



```

$00 $00 $00 $00 AB(1,0)
$00 $00 $00 $00 AB(2,0)
$00 $00 $00 $00 AB(3,0)
$00 $00 $00 $00 AB(0,1)
$00 $00 $00 $00 AB(0,2)
$00 $00 $00 $00 AB(0,3)
$00 $00 $00 $00 AB(0,4)

```

## 5.3 GOSUB and RETURN usage of the stack

When a GOSUB statement is executed the BASIC interpreter push into the CPU stack the address of where to RETURN and the number of the line to RETURN as follows (from top of stack downwards):

```

LL HH      Address of where to RETURN to
LL HH      Line number to RETURN to
$8C        GOSUB token as marker

```

This block remains into the stack until a RETURN is executed, at which point the BASIC looks back through the stack until it finds a GOSUB block. Then it sets the stack there, recovers the line number and the address of the statement after the GOSUB and continues the execution of the program from such point.

When a GOSUB block is stored into the stack, it deactivates all active FOR loops which were set up inside the subroutine.

## 5.4 FOR and NEXT usage of the stack

Same behavior for a FOR statement. When it is executed, the address of the first statement of the loop, the line number of the loop statement, the TO value, the STEP value, and the sign of the STEP are stored into the stack as follows (from top of stack downwards):

```

LL HH      Address of 1st statement in loop
LL HH      Line number of loop statements
XX XX XX XX TO value in floating point (f.p.)
YY YY YY YY STEP value in f.p.
SS         Signf of STEP
LL HH      Address of index variable
$81        FOR token as marker

```

This FOR block remains into the stack until a matching NEXT is executed. When NEXT is executed, BASIC looks back through the stack to find the value of the index variable and the result is compared to the TO value. With the use of the TO value and the sign of the step, the interpreter knows if the loop has been completed or not. If it has not been completed, then the FOR block remains into the stack until the loop has been completed. The stack is set to point to this FOR block and effectively kills all FORs nested within this loop. When the completed, the FOR block is removed from the stack and the execution continues from after the NEXT instruction.

## 6. HOW A BASIC PROGRAM IS STORED IN MEMORY

A BASIC program is stored in memory following a specific scheme. The BASIC text area starts at the location \$PROGST (in the actual firmware revision this corresponds to address \$55E5). This location must contain \$00. This is a special marker used by the interpreter to mark the beginning of the BASIC space. From the following 2 locations over, the BASIC program is stored line by line: each program line begins with a 2-byte address to the address of the next line in memory, followed by a word (2 bytes) containing the current line number, then the program line itself, and finally a zeros to mark the end of line. The text of the line is stored in a mixed format: BASIC statements are stored using their tokenized form (a particular form that uses a 1-byte code from \$80 to \$FF to represent each single statement) while the rest of the line is stored using ASCII codes. A couple of zeros used as pointers to the next line identify the end of the program.

An example is shown below. The following single line program:

```
10 FOR A=1 TO 10
```

it's stored as:

| ADRS | TK | NOTE  |
|------|----|---|
| ---- | -- | -----   |
| 5346 | 00 | Marker for beginning of BASIC space                       |
| 5347 | 56 | Pointer to...   |
| 5348 | 53 | ...next line (\$5356 in little endian format (LSB/MSB))   |
| 5349 | 0A | Line...   |
| 534A | 00 | ...number (\$000A = 10 in little endian format (LSB/MSB)) |
| 534B | 81 | Token for "FOR"   |
| 534C | 20 | ASCII code for "space" (\$20 = 32)                        |
| 534D | 41 | ASCII code for "A" (\$41 = 65)                            |
| 534E | C8 | Token for "=" ("=" is interpreted as a STATEMENT)         |
| 534F | 31 | ASCII code for "1" (\$31 = 49)                            |
| 5350 | 20 | ASCII code for "space" (\$20 = 32)                        |
| 5351 | B7 | Token for "TO"  |
| 5352 | 20 | ASCII code for "space" (\$20 = 32)                        |
| 5353 | 31 | ASCII code for "1" (\$31 = 49)                            |
| 5354 | 30 | ASCII code for "0" (\$30 = 48)                            |
| 5355 | 00 | End of line   |
| 5356 | 00 | Pointer to...   |
| 5357 | 00 | ...next line (\$0000 = end of program)                    |

## 7. SERIAL CONFIGURATION

If you intend to connect the LM80C to a host computer through the serial port A, you have to use an FT232 module to adapt the RS232 serial lines of the LM80C to the USB port of modern systems. Moreover, to avoid serial issues during sending data to the LM80C, please configure the terminal emulator you are using (i.e. CoolTerm or TeraTherm) with these params:

PORT: choose the port your system has mounted the FT232 module to

BAUDRATE: 19,200/38,400 bps

DATA BITS: 8

PARITY: none (0)

STOP BITS: 1

FLOW CONTROL: CTS (optional, combine with TX delay)

SOFTWARE SUPPORT FOR FLOW CONTROL: yes

RTS AT STARTUP: on

HANDLE BS AND DEL CHARS: yes

HANDLE FF (FormFeed) CHAR: yes

USE TX DELAY: min. 5ms (increment it if you experience issues)

## 8. VDP SETTINGS

The VDP, aka Video Display Processor, is the video chip of the LM80C computer. It is a TMS9918A from Texas Instruments. It can visualize a video image of 256x192 pixels with 15 colors and 32 sprites. It has several graphics modes, each of them configured to store video data in particular areas of the VRAM. These are the main settings for the modes supported by LM80C. Before to proceed, a little explanation of the meaning of different areas:

- pattern table: it's the area where the patterns that compose the chars are stored;
- name table: this is a sort of look-up table. This area maps what's is shown by the VDP in each cell of the video. The VDP reads the byte stored into a particular cell and then looks into the pattern table to find the data needed to draw the corresponding char;
- color table: some graphics modes store the color of a particular cell into this table.
- sprite pattern table: similarly to the pattern table, this area stores the data needed to draw the sprites;
- sprite attribute table: this area contains the info needed by the VDP to locate and color the sprites.

| SCREEN MODE | SCREEN WIDTH | SCREEN HEIGHT | PATTERN TABLE | NAME TABLE    | COLOR TABLE   | SPRITE ATTRIBUTE TABLE | SPRITE PATTERN TABLE |
|-------------|--------------|---------------|---------------|---------------|---------------|------------------------|----------------------|
| Screen 0    | 40 cols.     | 24 rows       | \$0000-\$07FF | \$0800-\$0BBF | ----          | ----                   | ----                 |
| Screen 1    | 32 cols.     | 24 rows       | \$0000-\$07FF | \$1800-\$1AFF | \$2000-\$201F | \$1B00-\$1B7F          | \$3800-\$3FFF        |
| Screen 2    | 256 px.      | 192 px.       | \$0000-\$17FF | \$1800-\$1AFF | \$2000-\$37FF | \$1B00-\$1B7F          | \$3800-\$3FFF        |
| Screen 3    | 64 blks.     | 48 blks.      | \$0000-\$07FF | \$0800-\$0AFF | ----          | \$1B00-\$1B7F          | \$3800-\$3FFF        |
| Screen 4    | 32 cols.     | 24 rows       | \$0000-\$07FF | \$1800-\$1FFF | \$3800-\$3AFF | \$1800-\$1FFF          | \$3B00-\$3B7F        |

N.B: the addresses above are referred to Video RAM, so you must use the VPOKE and VPEEK statements to access it.

## 9. Z80 DAISY CHAIN INTERRUPT PRIORITY

Since the LM80C is set up to work in interrupt mode 2 (IM2), the Z80 CPU serves the interrupt signals following a priority schematic that is hard-wired in the computer itself. In IM2 interrupt signals with higher priority are served before others with lower priority. In LM80C computer:

- the highest priority periphery is the Z80 SIO, since data incoming over the serial must be collected as soon as they are available;
- then the Z80 CTC, also used for the system tick counter;
- lastly, the Z80 PIO that, at the moment, it's just used as on output periphery.

## 10. STATUS LEDs

Status LEDs are used by the operating system to communicate special conditions to the users. Their meaning is as follow:

|   |                            |
|---|----------------------------|
| 0: Bank 0 selector: 0=RAM, 1=ROM        | 4: Serial 1 buffer overrun |
| 1: VRAM bank # selector: 0=def., 1=alt. | 5: Serial 2 buffer overrun |
| 2: N.C.                                 | 7: Serial 1 line open      |
| 3: N.C.                                 | 8: Serial 2 line open      |

Please remember that LEDs #1 and #2 drive the bank switching mechanisms to switch between ROM and RAM in bank #0 of the main memory and between the two VRAM banks for the VDP, respectively. So, consider that you could get some unwanted side effects if you change them unintentionally.

## **11. REFERENCES**

1. NASCOM ROM BASIC DIS-ASSEMBLED – PART I – BY CARL LLOYD-PARKER
2. [https://en.wikipedia.org/wiki/Microsoft\\_Binary\\_Format](https://en.wikipedia.org/wiki/Microsoft_Binary_Format)

## 12. USEFUL LINKS

Project home page:

<https://www.leonardomiliani.com/en/lm80c/>

Github repository for source codes and schematics:

<https://github.com/leomil72/LM80C>

Hackaday page:

<https://hackaday.io/project/165246-lm80c-color-computer>



**LM80C**

**Color Computer**

**Enjoy home-brewing computers**

**Leonardo Miliani**