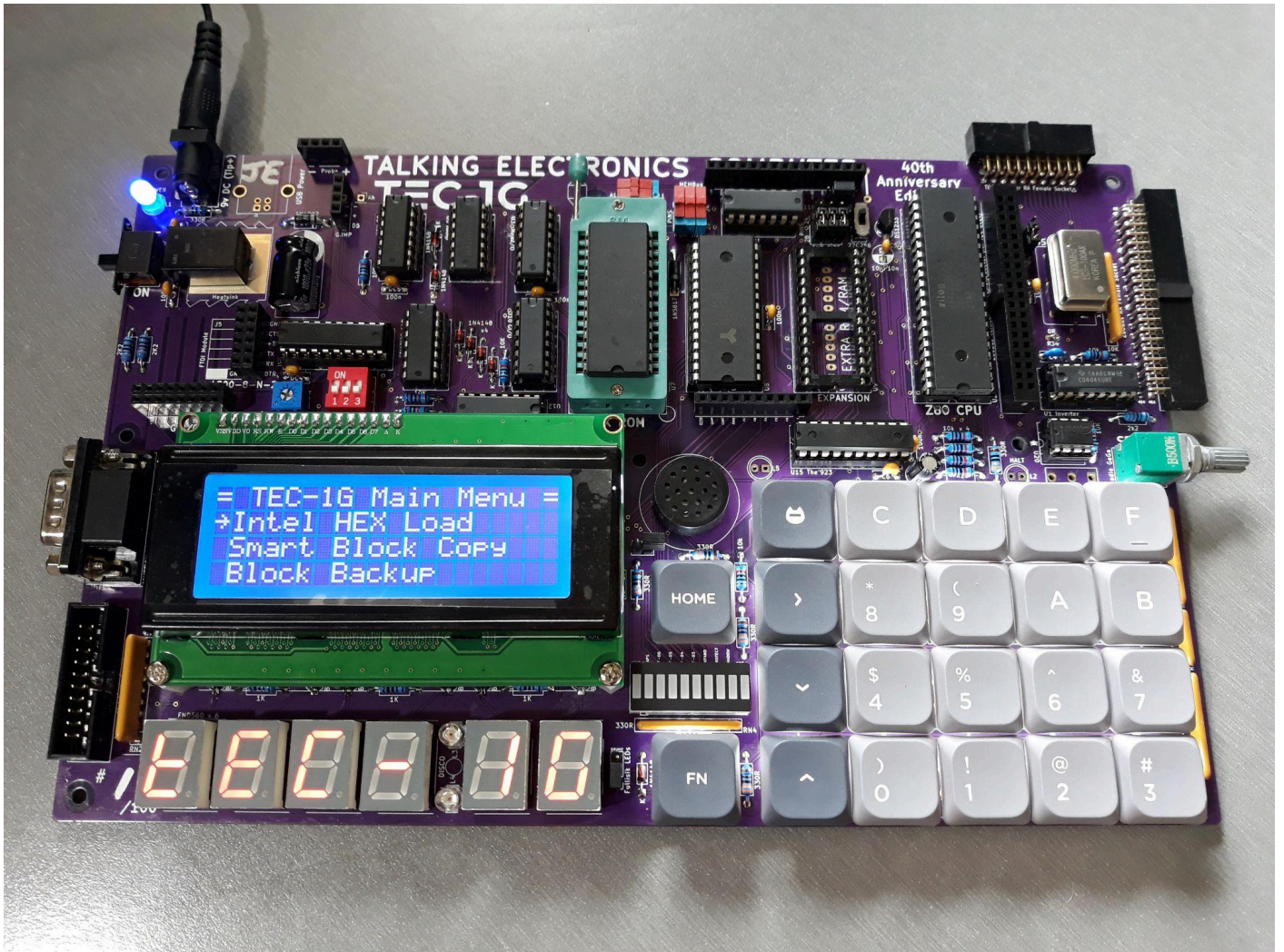


# TEC-1G MON3

## User guide

By Brian Chiha v1.4



Mon3 (Talking Electronics Computer Monitor version 3) is custom-built for the TEC-1G Single Board Z80 Computer. Mon3 is the heart of the TEC-1G. It brings the hardware to life. Consider it an Operating System that provides the ability to program the TEC. The monitor is jam-packed with features, designed for beginners who are just learning to code Z80 and rich enough for the advanced software developer.

# Table of Contents

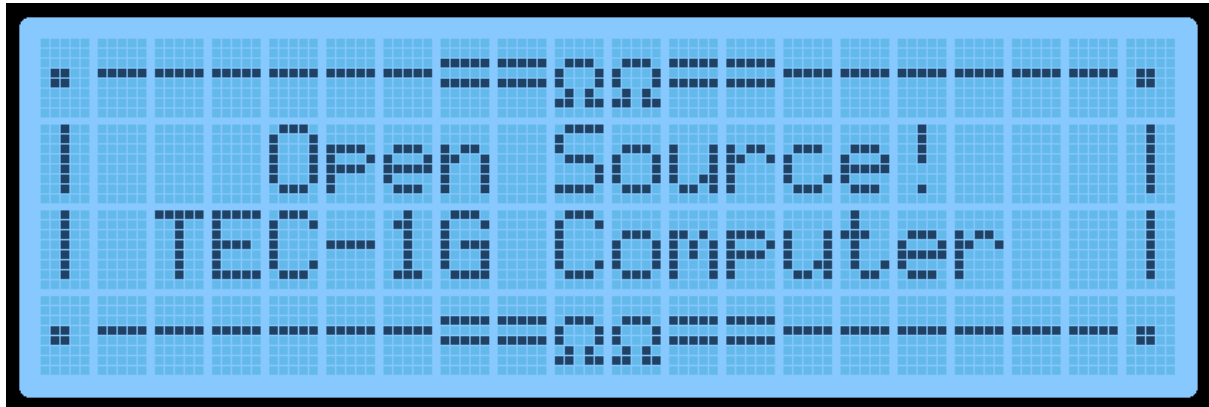
<b>Basic Operation</b> .....	<b>4</b>
Cold Reset.....	4
Warm Reset.....	4
<b>Main Menu</b> .....	<b>5</b>
Intel HEX Load.....	6
Smart Block Copy.....	6
Block Backup.....	7
Export Z80 Assembly.....	7
Export Raw Data.....	8
Export Hex Dump.....	8
Import Binary File.....	9
Tiny Basic.....	9
Music Routine.....	10
Terminal Monitor.....	11
Settings.....	11
Credits.....	11
<b>Memory Map</b> .....	<b>12</b>
<b>Data Entry Mode</b> .....	<b>13</b>
Basic Operation.....	13
LCD Screen.....	14
Function Keys.....	15
<b>Matrix Keyboard</b> .....	<b>17</b>
<b>Debugging Programs</b> .....	<b>18</b>
<b>Terminal Monitor</b> .....	<b>19</b>
Starting up TMON.....	19
Using TMON.....	19
The Command Prompt.....	20
DATA mode.....	20
TMON Commands.....	21
<b>TEC Magazine Code on the TEC-1G</b> .....	<b>24</b>
<b>Advanced Programming</b> .....	<b>25</b>
RST (Restart) commands.....	26
Interrupts.....	27
NMI (Non-Maskable Interrupts).....	27
API (Application Programming Interface) commands.....	29
General conventions.....	29
API Call List.....	30
API Utility Calls.....	31

API LCD Calls.....	34
API Input Calls.....	35
API Serial Data Transfer Calls.....	37
API Menu & Parameter Calls.....	40
API Sound Calls.....	43
API System Latch Calls.....	44
Miscellaneous Calls.....	45
Real Time Clock (RTC) Add-On Interface.....	46
RTC API Calls.....	47
Graphical LCD Add-On Interface.....	53
General Conventions.....	54
GLCD API Calls list.....	55
GLCD API Configure Calls.....	55
GLCD API Graphics Calls.....	57
GLCD API Text Calls.....	59
GLCD API Utility Calls.....	60
GLCD Examples.....	62
<b>Quick Start Programs.....</b>	<b>63</b>
<b>Appendix.....</b>	<b>69</b>
Ports.....	69
Serial Connection.....	70
LCD Cheatsheet.....	71
<b>Useful Links.....</b>	<b>74</b>
I/O Connectors.....	75

The version of this document corresponds to the version of the monitor binary file. IE: Version 1.2 of this document is for file **MON3-1G\_BC23-12.bin**. The 12 at the end of the file is the version number.

# Basic Operation

With the monitor loaded into the ROM socket and all the jumpers set correctly for the ROM used. Turn the TEC on. If all is working well, a welcome banner will be displayed on the LCD and a short tune will be heard.



## Cold Reset

When the TEC turns on after being powered down, a Cold Reset occurs. A Cold Reset signified with the display of the welcome banner and the short tune. A Cold Reset will configure the monitor for first-time use after powering it on. It will default monitor variables and configure the LCD for first use.

If the TEC isn't responding normally or something "weird" is occurring, a manual Cold Reset can be performed. Programs loaded in RAM will be retained when a manual Cold Reset is done. To do a manual Cold Reset, while pressing and releasing the **RESET** key, hold the **Fn** key down. The distinctive LCD Banner and music tone will indicate that the Cold Reset was successful. A manual Cold Reset on the HexPad will still work if the Matrix Keyboard is in use.

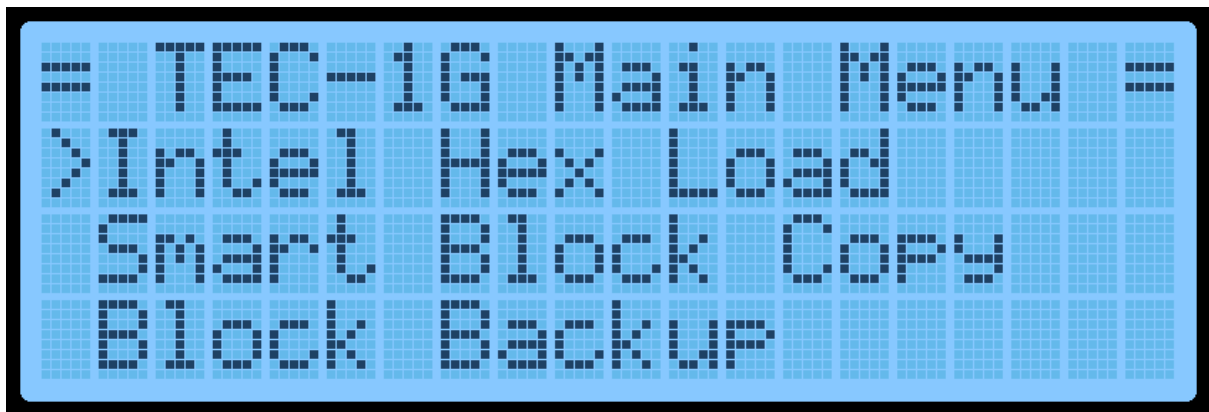
## Warm Reset

A Warm Reset occurs when pressing and releasing the **RESET** key. A warm reset returns the TEC to its initial editing location on a Cold Reset. It's a quick way to get back to the start of a code block.

# Main Menu

A menu is provided on the LCD screen to help with navigating the inbuilt routines that the monitor comes with. A menu will appear on Cold Reset.

Navigating the menu should be intuitive. Press the **Plus** or **Minus** keys to scroll down and up. Press **GO** to run the selected routine. A right-facing Arrow indicates which menu item is currently selected. One thing that might not be obvious is how to exit the menu and move into Data Entry mode. This is achieved by pressing the **AD** key. Once this is known, it's hard to forget it. Menus can be nested up to 3 deep. Pressing the **AD** key will exit to the parent menu or Data Entry mode if at the main menu.



The current items on the menu are

Menu Text	Description
Intel HEX Load	Receive data in Intel Hex File format via the FTDI connector
Smart Block Copy	Move a block of code AND update all 2-byte addresses that are within the block
Block Backup	Move a block of code
Export Z80 Assembly	Display Z80 Assembly to a Serial terminal via the FTDI connector
Export Raw Data	Send binary data via the FTDI connector
Export Hex Dump	Display a 16-byte per line HEX dump to a Serial terminal via the FTDI connector
Import Binary File	Receive data in binary format via the FTDI

	connector
Tiny Basic	Run Tiny Basic on a Serial terminal
Terminal Monitor	Serial terminal monitor interface
Music Routine	Play musical notes at a given address
Settings	Update monitor settings
Credit	Display the people who made the TEC-1G

## Intel HEX Load

Intel created a text file format that contains information on loading bytes into memory. When this routine is run, the TEC seven segments will go blank and wait for a file to be received. This is done via the FTDI connector and serial terminal. When data is transmitted, the rightmost segment will illuminate in a pattern. This indicates data is being read. Once the file has fully loaded, the letters "PASS" will display on the seven segments. This means that the load was successful. Press any key to exit. If the segments display the word "FAIL", then there is something wrong with the file or your serial connection.

## Smart Block Copy

This very clever routine shifts a program from one spot in memory to another and changes all absolute jumps and calls. Memory pointers are also altered if the memory pointers are within the start and end address of the program being relocated. Any reference to a location outside the start and end range is not altered.

The block copy treats Data bytes as instructions and might change data bytes as well. IE: `.db C3, 23, 01` could be seen as a `JP 0123` instruction.

When this routine is run, it will ask for a START, END and DESTINATION address. Type in the 16-bit address via the HEX PAD and use the **Plus** or **Minus** keys to change the selected parameter. Press **GO** to run the routine.

Here is an example of copying **4000H-4009H** to location **2000H**

Original		After Copy	
4000 11 09 40	<b>LD DE,4009</b>	2000 11 09 20	<b>LD DE,2009</b>
4003 E7	<b>RST 20</b>	2003 E7	<b>RST 20</b>
4004 FE 13	<b>CP 13</b>	2004 FE 13	<b>CP 13</b>
4006 C2 00 40	<b>JP NZ,4000</b>	2006 C2 00 20	<b>JP NZ,2000</b>
4009 C9	<b>RET</b>	2009 C9	<b>RET</b>

## Block Backup

This routine simply copies a data block from one address location to another. No bytes are altered when the copy is performed. This routine is also useful to copy data reference tables like music data for the music routine.

When this routine is run, it will ask for a START, END and DESTINATION address. Type in the 16-bit address via the HEX PAD and use the **Plus** or **Minus** keys to change the selected parameter. Press **GO** to run the routine.

Here is an example of copying **4000H-4009H** to location **2000H**

Original		After Copy	
4000 11 09 40	<b>LD DE,4009</b>	2000 11 09 40	<b>LD DE,4009</b>
4003 E7	<b>RST 20</b>	2003 E7	<b>RST 20</b>
4004 FE 13	<b>CP 13</b>	2004 FE 13	<b>CP 13</b>
4006 C2 00 40	<b>JP NZ,4000</b>	2006 C2 00 40	<b>JP NZ,4000</b>
4009 C9	<b>RET</b>	2009 C9	<b>RET</b>

## Export Z80 Assembly

If the TEC is connected to a serial terminal via an FTDI to USB adaptor, code that is stored or written on the TEC can be disassembled and sent to the terminal. This is a great way to view the code that is on the TEC in a readable format and could be passed into a Z80 compiler on a PC.

When this routine is run, it will ask for a START and END address. Type in the 16-bit address via the HEX PAD and use the **Plus** or **Minus** keys to change the selected parameter. Press **GO** to run the routine.

Here is an example of its output.

```
4000 3E 3F      LD A,3F
4002 D3 01      OUT (02),A
4004 3E 04      LD A,04
4006 D3 02      OUT (02),A
4008 CF        RST 08
4009 C9        RET
```

## Export Raw Data

This routine will send binary data from the TEC to a serial connection. It's a way of saving the code written on the TEC to a PC. As binary data is being sent, the data can only be properly viewed through a HEX file viewer or HEX dump routine.

When this routine is run, it will ask for a START and END address. Type in the 16-bit address via the HEX PAD and use the **Plus** or **Minus** keys to change the selected parameter. Press **GO** to run the routine.

## Export Hex Dump

This routine will display binary data in a readable format to a serial terminal connected via an FTDI to USB adaptor. It will display up to 16 bytes per line.

When this routine is run, it will ask for a START and END address. Type in the 16-bit address via the HEX PAD and use the **Plus** or **Minus** keys to change the selected parameter. Press **GO** to run the routine.

Here is an example of its output.

```
C100: 31 80 08 21 00 40 CD FC C5 AF D3 05 D3 06 DB 03
C110: 47 E6 10 C2 00 80 3A 9F 08 E6 04 0E 01 B1 D3 FF
C120: 32 9D 08 78 E6 02 32 9E 08 3A 9D 08 E6 01 28 0B
C130: 21 00 C0 11 00 00 01 00 01 ED B0 21 00 40 22 86
C140: 08 22 A0 08 DB 03 0F 38 06 DB 00 E6 20 18 08 CD
```



## Import Binary File

This routine will upload a binary file from a PC onto the TEC via an FTDI to USB adaptor. This is the opposite of the Export Raw Data routine and will load binary data to a given address on the TEC.

When this routine is executed, it will ask for a START and END address. This address range must match the size of the binary file being sent. Type in the 16-bit address via the HEX PAD and use the **Plus** or **Minus** keys to change the selected parameter. Press **GO** to run the routine. The TEC will wait for data to be received and will end when END-START+1 bytes are received.

## Tiny Basic

Mon3 comes with Tiny Basic installed. Tiny Basic is an easy-to-use BASIC programming language. At this stage, all interactions with BASIC are done on a serial terminal via an FTDI to USB adapter. Some extra commands have been implemented to interact with the TEC-1G hardware.

```
Z80 TINY BASIC 2.2b
TEC-1G VERSION BY B CHIHA, 2023

OK
>LIST
 5 REM ** FIBONACCI SEQUENCE **
10 PRINT "FIBONACCI SEQUENCE"
20 FOR I=1 TO 22
30 GOSUB 70
40 PRINT "F", I, F
50 NEXT I
60 STOP
70 LET A=0; LET B=1
80 FOR J=1 TO I
90 LET T=A+B; LET A=B; LET B=T
100 NEXT J
110 LET F=A
120 RETURN
```

For information on how to use Tiny Basic, go to this link:

[https://github.com/bchiha/BMon/wiki/tiny\\_basic](https://github.com/bchiha/BMon/wiki/tiny_basic).

## Music Routine

Use this routine to play some notes to the TEC speaker. It is based on John Hardy's Mon1 routine adjusted for a 4 Mhz clock speed. The routine uses similar input codes making it suitable for existing tunes to be used.

When this routine is executed, it will ask for a START address of the music data—type in the 16-bit address via the HEX PAD. Press **GO** to run the routine.

Two octaves are playable. Here is a reference to the note code and its musical note. A Pause is represented by **00** and any other note code that isn't listed will exit the routine.

Note	Code	Note	Code	Note	Code	Note	Code
G	01	C#	07	G	0D	C#	13
G#	02	D	08	G#	0E	D	14
A	03	D#	09	A	0F	D#	15
A#	04	E	0A	A#	10	E	16
B	05	F	0B	B	11	F	17
C	06	F#	0C	C	12	F#	18

Here are some examples tunes that can be typed in a played

### Bealach

06, 06, 0A, 0D, 06, 0D, 0A, 0D, 12, 16, 14, 12, 0F, 11, 12, 0F  
 0D, 0D, 0D, 0A, 12, 0F, 0D, 0A, 08, 06, 08, 0A, 0E, 0A, 0D, 0F  
 06, 06, 0A, 0D, 06, 0D, 0A, 0D, 12, 16, 14, 12, 0F, 11, 12, 0F  
 0D, 0D, 0D, 0A, 12, 0F, 0D, 0A, 08, 06, 08, 0A, 06, 12, 00, 1F

### Angels On High

0F, 0F, 0F, 0F, 0F, 0F, 12, 12, 12, 12, 12, 10, 0F, 0F, 0F, 0F  
 0F, 0F, 0D, 0D, 0F, 0F, 12, 12, 0F, 0F, 0F, 0D, 0B, 0B, 0B, 0B  
 0F, 0F, 0F, 0F, 0F, 0F, 12, 12, 12, 12, 12, 10, 0F, 0F, 0F, 0F  
 0F, 0F, 0D, 0D, 0F, 0F, 12, 12, 0F, 0F, 0F, 0D, 0B, 0B, 0B, 0B  
 12, 12, 12, 12, 14, 12, 10, 0F, 10, 10, 10, 10, 12, 10, 0F, 0D  
 0F, 0F, 0F, 0F, 10, 0F, 0D, 0B, 0D, 0D, 0D, 06, 06, 06, 06, 06  
 0B, 0B, 0D, 0D, 0F, 0F, 10, 10, 0F, 0F, 0F, 0F, 0D, 0D, 00, 00  
 00, 12, 12, 12, 12, 14, 12, 10, 0F, 10, 10, 10, 10, 12, 10, 0F  
 0D, 0F, 0F, 0F, 0F, 10, 0F, 0D, 0B, 0D, 0D, 0D, 06, 06, 06, 06  
 06, 0B, 0B, 0D, 0D, 0F, 0F, 10, 10, 0F, 0F, 0F, 0F, 0D, 0D, 0D  
 0D, 0B, 0B, 0B, 0B, 0B, 0B, 0B, 00, 00, 00, 00, 00, 00, 1F

# Terminal Monitor

Terminal Monitor or TMON gives the user the ability to interface with the TEC-1G via a serial terminal. There is an extensive chapter regarding the use of TMON below.

## Settings

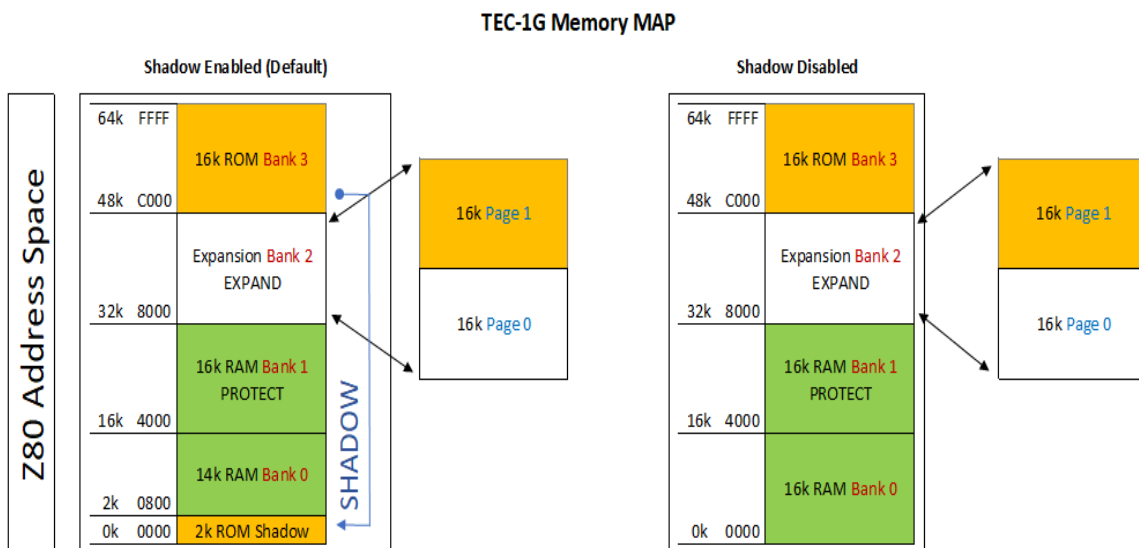
The settings allow the user to configure the monitor. Turning the power off from the TEC will return these settings to their default state. Some settings will be retained if an RTC Add-on board is connected with battery backup.

- Toggle Key Beep - Turn the keypress 'beep' indication on or off.
- Toggle Address Inc - Turn the automatic address increase after a byte has been keyed on or off.
- Configure RTC - Set Time/Date of RTC (if RTC Add-on is connected).
- Reset RTC PRAM - Reset RTC NVRAM (if RTC Add-on is connected).
- Toggle EXPAND - software controlled the expansion socket to toggle between lower and upper 16Kb memory for a 32Kb ROM/RAM chip.

## Credits

Display the people who developed and tested the TEC-1G

- Mark Jelic - Designer of the TEC-1G
- Brian Chiha - Mon3 Programmer
- Craig Hart - TECnical Expert
- Ian McLean - Tester and QA
- James Elphick - Tester and QA
- John Hardy & Ken Stone - The original designers



# Memory Map

The table below outlines how the full 64Kb of address space is allocated on the TEC-1G.

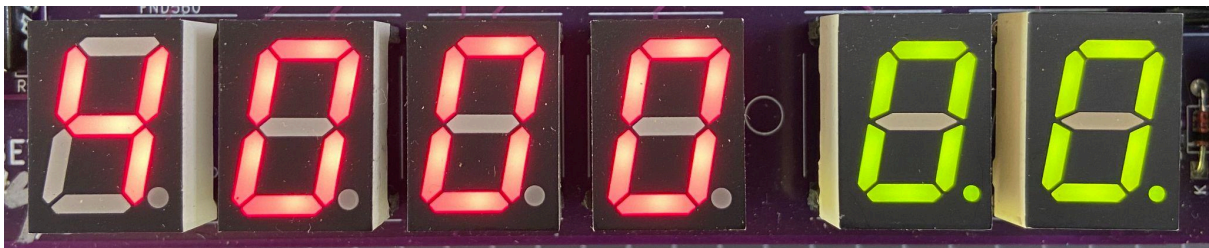
Address	Contents	Type
<b>0000H-00FFH</b>	Reserved for Z80 instructions	RAM
<b>0100H-07FFH</b>	Free RAM	RAM
<b>0800H-087FH</b>	Reserved for Hardware Stack	RAM
<b>0880H-0FFFH</b>	Reserved for Monitor RAM	RAM
<b>1000H-3FFFH</b>	Free RAM	RAM
<b>4000H-7FFFH</b>	Free RAM (Protected)	RAM
<b>8000H-BFFFH</b>	Expansion Socket	RAM/ROM
<b>C000H-FFFFH</b>	Monitor ROM	ROM

Some things to be considered are:

- Any RAM location can be updated, but it is highly recommended **not to** update Monitor **Reserved** RAM locations. This can/will cause undesirable effects on the running of the TEC. A Cold Reset will restore the TEC to its default running state (hopefully).
- The address range between **4000H-7FFFH** is a special area that can be made READ ONLY. This is called a Protected area. Protect mode can be switched on using the configuration 3-DIP switch. If protect is enabled and code is being executed. No RAM update can be done in this range. This feature is designed to protect keyed-in code from being inadvertently erased by a rogue routine.
- The Expansion Socket on the TEC can have a 32Kb ROM or RAM inserted. Only 16kb can be accessed at one time. To switch between high and low memory use the Expand switch on the configuration 3-DIP switch. The switch can also be overridden in software by toggling the Expand flag in the Settings menu or pressing Fn-E.
- If the monitor ROM is a legacy monitor, IE: Mon1, Mon2, JMon or BMon, The address range **0000H-07FFH** will be READ ONLY and will emulate the same addressing that is used for that particular ROM. Shadow mode will be active by default and will be indicated by an illuminated LED segment on the system latch BAR component.

# Data Entry Mode

Data Entry Mode allows the user to enter Z80 Op Codes directly into the TEC. To access Data Entry Mode from the Main Menu simply press the **AD** key. In this mode, the 4 left seven-segment displays will show the current editing address and the 2 right segments will display the byte at that address.



**Address**

**Data**

The decimal place LED on the segments indicates which part, Address or Data is currently enabled for direct updates. In the picture above, the dots are on the Data segments.

The initial starting address is **4000H**. This address was chosen as it's within the Protect RAM area.

## Basic Operation

To update a byte at an address, simply use the **0-F** keys on the keypad. After the byte has been entered, by default when the next byte is keyed, the current editing address will automatically move to the next address location. This saves the user from pressing the **Plus** key after each byte is added. This option can be switched off in the Settings menu.

To navigate to another address, press the **Plus** or **Minus** key. Or press the **AD** key. The decimal place dots will move to the address segments indicating that the address field is updatable. Key in a new 16-bit address by using the **0-F** keys. Press the **AD** key to move back to data updating mode.

And finally, to execute code, navigate to the address where the code starts and press the **GO** key. Protect mode will be honoured if switched on. If the code ends with a **RET** instruction (**C9**), execution will cleanly exit back to the monitor.

One thing to note is that while data is being entered, the decimal place LED on the data segments will change from displaying two lights to one. The one light will indicate which Nibble (half byte) has been entered. This will assist in knowing if the whole byte has been entered or not.

If a mistake is made during data entry and the byte is to be re-entered. To stop the address from automatically incrementing, press the **AD** key twice. This will reset the Nibble counter and allow a new byte to be entered.

If any key is held down, after a short period, the key will automatically repeat. This is mostly useful while holding down the Plus or Minus key to quickly move to a new address. But can also be used to populate memory with 00 or FF or anything else.

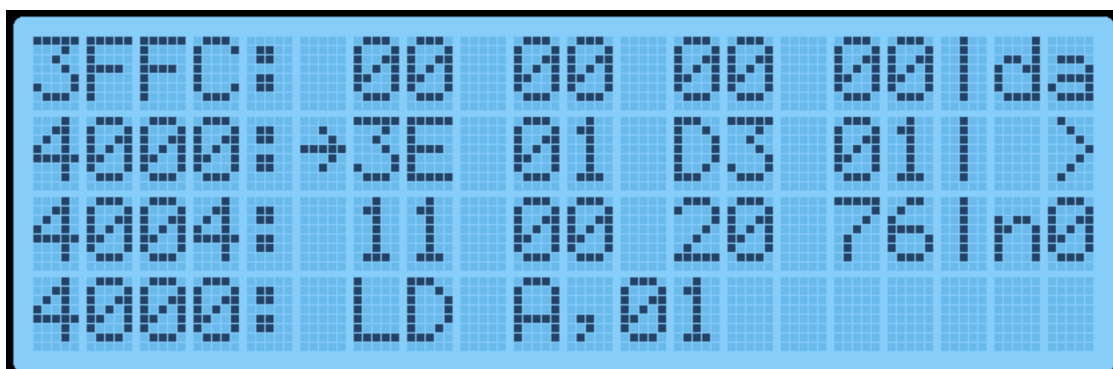
## LCD Screen

In Data Entry Mode the LCD Screen will display 12 bytes of data. 4 bytes before the current editing location and 8 bytes from the current editing location. These bytes are displayed in groups of 4 (3 lines). A right arrow indicates the byte at the current editing location.

Displayed on the right side of the screen is the current edit mode, **da**=Data, **ad**=Address, the current byte in LCD ASCII and the Nibble Counter. The picture below is showing: The current address is **4000**, Data mode, ">" = **3E** in ASCII and **0** nibble count.

On the 4th line of the LCD, the Z80 Assembly of the current OP Code(s) is shown. This can be useful to see what instruction is currently being keyed.

By displaying a range of bytes on the LCD, the user can check if the correct bytes have been entered without individually moving to each address.

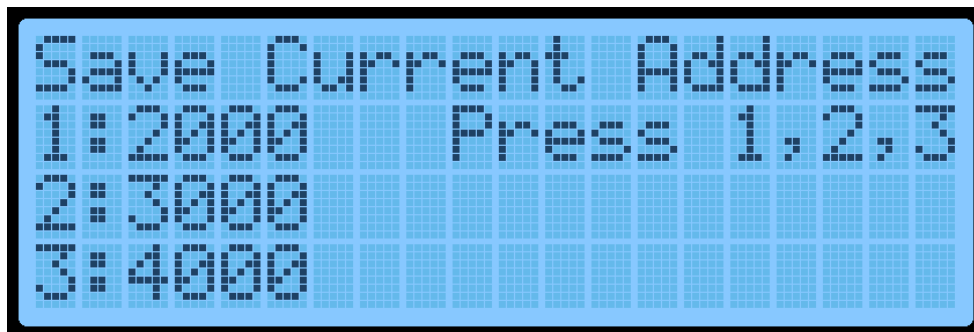


## Function Keys

Various extra options can be selected via the Function Key. To use these functions, hold the **Fn** key down and press one of the **O-F** keys.

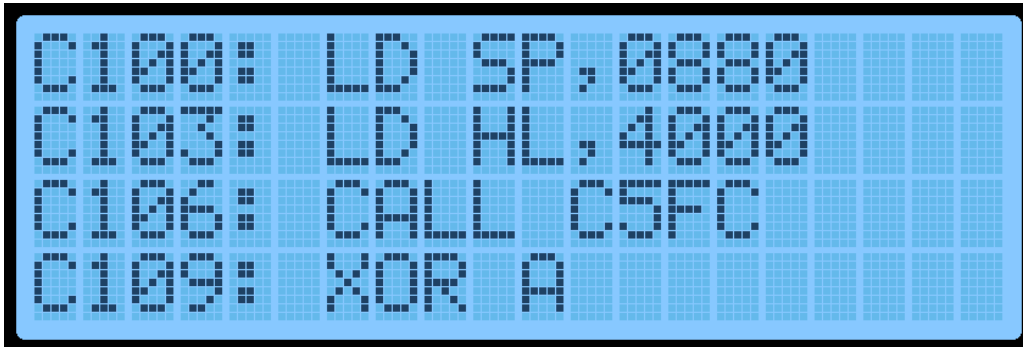
The routines attached to the Function Key are:

- **Fn-AD** - Display the Main Menu
- **Fn-0** - Save Current Address. Press 1,2 or 3 to save the current editing address in RAM to quickly jump to this location later. Three addresses can be saved. This is useful if code is in a location other than 4000H and the Reset button has been pressed. Press AD to exit the routine. The initial default address is 4000H.



- **Fn-1, 2, 3** - Quick jump to Address. This will move the monitor's current editing location to the saved address set by **Fn-0** above.
- **Fn-4** - Intel Hex Load. This is a shortcut to the Main Menu routine.
- **Fn-B** - Block Backup. This is a shortcut to the Main Menu routine.
- **Fn-C** - Smart Block Copy. This is a shortcut to the Main Menu routine.

- **Fn-D** - Switch between Data Entry View and Disassembly View. Disassembly View displays the next 4 Assembly instructions. To move through the instructions press the **Plus** or **Minus** keys. Data entry can still be done in this mode if desired.



```
C100: LD SP, 0880
C103: LD HL, 4000
C106: CALL C5FC
C109: XOR A
```

- **Fn-E** - Toggle the Expansion Socket Expand flag. This will switch between the upper and lower memory of the 32Kb ROM/RAM in the expansion socket.
- **Fn-Plus** - Insert an **NOOP** instruction at the current editing location AND move all bytes up to max RAM by one address upwards. It will also do a Smart Block Copy to all moved bytes. This routine can add a Breakpoint (**F7**) or missing opcodes to an existing program.
- **Fn-Minus** - Delete a byte from the current editing location AND move all bytes down by one address. It will also do a Smart Block Copy to all moved bytes.



# Matrix Keyboard

Mon3 will work with the TEC Matrix Keyboard Add-on. The Keyboard is connected to the Keyboard Socket on the lower left of the PCB. How your Keyboard PCB is designed might affect which pins can be connected. Please view the TEC-1G Schematic for information on pin configuration.



To activate the Keyboard, The Matrix switch on the 3-DIP switch is to be turned on. This activates the Matrix Keyboard and disables the onboard Hex Keypad (except Reset). Mon3 only maps keys present on the TEC-1G to the Matrix Keyboard.

The Keyboard map to Hex Keypad is as follows:

- **AD** - Esc
- **Plus** - Right Arrow
- **0-F, Fn** - 0-F, Fn keys
- **GO** - Enter
- **Minus** - Left Arrow
- **Reset** - Reset key if connected

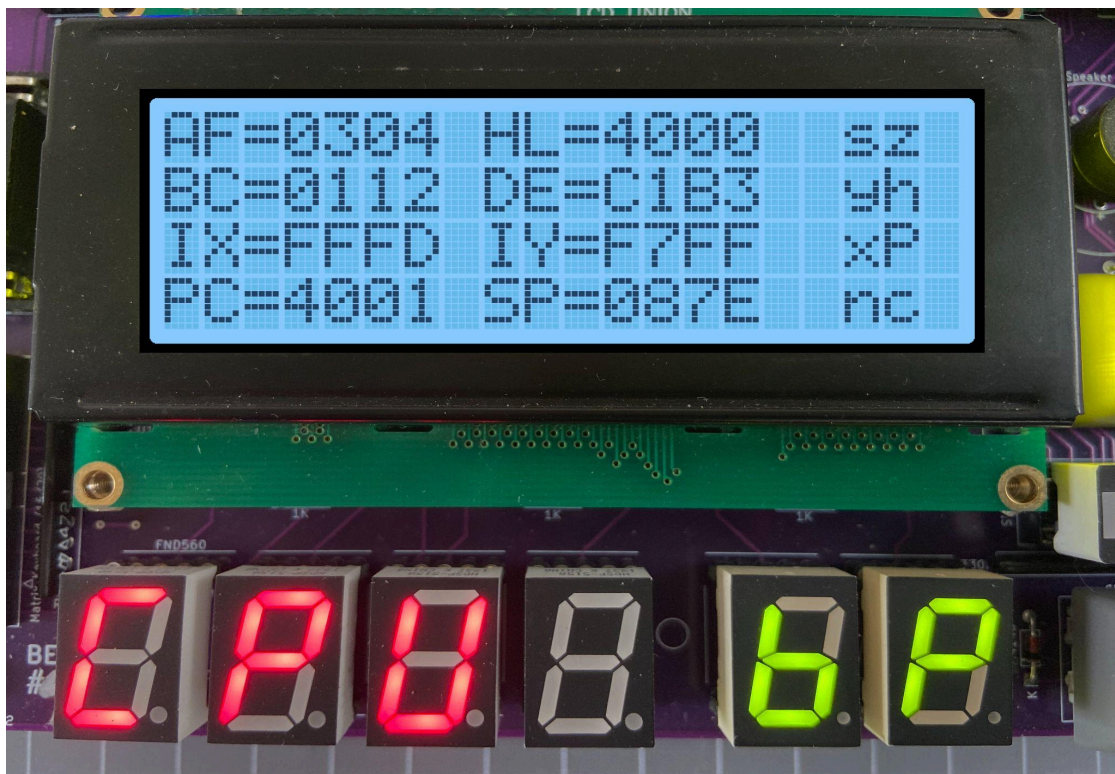
The full range of keys can be accessed and converted when developing programs via the `matrixScan` and `matrixToASCII` API routines.

# Debugging Programs

Breakpoints can be inserted within a program which can help with viewing the state of the CPU registers. To break the execution of your code, insert a **RST 30H** or **F7** at the current address where the break should occur.

An easy way to insert a byte into an existing program is to press **Fn-Plus**. This will insert a **NOP** instruction at the current address. Then change this byte to **F7**.

When the execution of code is interrupted with a breakpoint, the TEC will pause and display register information on the LCD screen.



The contents of the Z80 CPU registers AF, HL, BC, DE, IX, IY, the Program Counter and Stack Pointer are displayed. CPU Flags are also displayed. Flags that are set are in Capitals. To continue code execution press the **GO** key and to quit execution and return to the Monitor press the **AD** key. Finally, to remove an inserted Breakpoint press **Fn-Minus** at the address where the Breakpoint is. This will remove the breakpoint and adjust the code to its original state. **Note:** Breakpoints will be ignored if a connection is made between the + and the D5 pins on the **G.IMP** header. **Warning:** Do not connect the + to the - pin on the **G.IMP** header!!! This will short out the TEC!

# Terminal Monitor

The Terminal Monitor (TMON) is a complete serial port-based monitor for the TEC-1G, designed for users who prefer to interact with the TEC-1G via a terminal. TMON is written by Craig Hart

## Starting up TMON

Connect a serial terminal to the TEC-1G via the FTDI to USB connector. Then, select Terminal Monitor from the main menu by pressing GO and look at the serial terminal.

```
TMON for TEC-1G Version 1.0
MON-3 Version: 2023.11
RAM Found between 0000h and 3FFFh - 16384 bytes
1000 >
```

## Using TMON

TMON is an interactive tool that works with a serial terminal e.g. PuTTY or Tera Term on a PC, or a 'real' VT100 serial terminal such as a Wyse WY-60. The TEC-1G keypad and 7-seg displays are not used once the program starts, and do not do anything (except for the testing routines documented below).

Interactions with TMON are via the serial console. The user types commands interactively and the results are displayed on the terminal.

All interactions with TMON use HEX format - so a byte is 00 to FF, etc. The "h" or "0x" is omitted for brevity.

Typically, the ADDR key exits any interactive command, or by entering "Q" from the terminal.

The above text is the default display when TMON first starts. TMON is now awaiting input and commands from the Available Commands list can be entered.

## The Command Prompt

```
1000 >
```

The `1000` represents the CURRENT ADDRESS in HEX. Many commands default to their actions interacting with memory at this address. The CURRENT ADDRESS changes as with certain commands. e.g. inputting code and data, and can be set by the ADDR command. By default, TMON points to itself.

The command input editor is very simple. Invalid inputs are typically ignored and result in the user simply being returned to the command prompt. The maximum command length accepted is 40 characters, however, presently the longest valid command possible is 9 characters in length. When the user's input exceeds the maximum command length, the TEC will emit a beep tone to indicate this condition has been reached. Backspace is supported, to correct typos.

All data entered at all times is assumed to be HEX - 4 bytes for addresses, 2 bytes for data. Invalid data input is ignored.

## DATA mode

When the DATA command is given, TMON switches to interactive data entry mode. This is signified by the prompt changing as follows:

```
XXXX nn :
```

XXXX continues to represent the CURRENT ADDRESS however the nn represents the HEX byte stored at that address, which you are presently editing.

- Enter a **HEX byte** and it will be written to memory at CADDR; CURRENT ADDR is then incremented by one.
- **ENTER** increments CURRENT ADDRESS by one and leaves the existing value as-is. In this way, any bytes that don't need altering are skipped over.
- **-** decrements the CURRENT ADDRESS by one. This allows for correcting input errors by going back one address after erroneous input.
- **Q** exits data entry mode.

Invalid entries will be ignored.

The DATA entry system is very simple and will continue to be improved in future versions.

## TMON Commands

<b>HELP</b>	<b>?</b>	<b>EXIT</b>
<b>INTEL</b>	<b>BEEP</b>	<b>BELL</b>
<b>VER</b>	<b>STATE</b>	<b>CLS</b>
<b>RAMCHK</b>	<b>GO [xxxx]</b>	<b>DUMP [xxxx]</b>
<b>ADDR [xxxx]</b>	<b>DATA [xxxx]</b>	<b>INC</b>
<b>7SEG</b>	<b>SMON</b>	<b>HALT</b>
<b>DEBUG</b>	<b>KEYTEST</b>	<b>FILL xxxx yyyy nn</b>
<b>PRINT</b>		

Parameters marked with square brackets e.g. \[xxxx\] are optional.

### **HELP**

Displays help text

### **?**

Display the list of commands

### **EXIT**

Reboots the 1G back to MON3

### **INTEL**

Calls the Intel Hex file transfer routine built into MON-3

### **BEEP**

Beeps the 1G speaker

### **BELL**

Sents the BELL command to the remote console

**VER**

Displays the version number of TMON and MON-3

**STATE**

Displays the state of the 1G system - SHADOW, PROTECT, EXPAND, CAPS LOCK

**CLS**

Sends a clear screen sequence to the remote console

**RAMCHK**

Runs a simple test to determine how much RAM is installed, and at what momentary address(es). Uses whichever bank EXPAND is set to, but does not alter the EXPAND state. Supports multiple discontinuous RAM blocks, if fitted.

**GO xxxx**

Executes code from the CURRENT ADDRESS, or from xxxx if supplied.

**DUMP xxxx**

DUMP the contents of 64 bytes of memory; provides HEX and ASCII outputs so memory can be examined.

DUMP pauses at completion - space repeats the command (CADDR continues to increment if auto-increment is on; otherwise the same block repeats). This allows you to quickly run through larger blocks without needing to type commands repeatedly.

Q quits and returns to the command prompt.

**ADDR xxxx**

Set the CURRENT ADDRESS. If no address is supplied, display the CADDR instead.

**DATA xxxx**

Interactively Input data into memory. Input one hex byte at a time; the value input is stored in the CADDR memory location.

Enter Q to quit input mode. See full description of DATA mode, above.

## **INC ON/OFF**

Set auto-increment mode of CADDR. No parameter supplied = Display the current auto-increment mode. Sometimes turning auto-increment off is helpful for debugging or monitoring.

## **7SEG**

Displays the CADDR and byte of memory on the TEC 7-seg displays. + and - keys increment/decrement CADDR. Pressing the ADDR key exits to TMON.

## **SMON**

Serial data stream monitor. Accepts serial input from the terminal and displays the HEX bytes received on screen. Great for debugging terminal comms and understanding control codes received from the PC (e.g. VT100 sequences). This is a crude implementation but does display the limitations of the bit-bang serial in not being able to adequately buffer incoming bytes in real time (try pressing an arrow key or a PC function key).

Enter Q (capital) to exit SMON back to TMON.

If a terminal program such as Tera Term is used to add a small delay (e.g 20ms) between bytes transmitted from the PC, SMON can accurately show VT100 control codes such as a PC arrow or function key. Without the delay, the bit-bang serial normally gets the first byte only, or perhaps the first and fourth or fifth byte, hence demonstrating the limitations of the bit-bang interface.

## **HALT**

Executes a CPU HALT instruction - on TEC-1F, press any key to resume.

## **DEBUG**

Calls the MON-3 debugger/breakpoint tool to examine register contents.

## **KEYTEST**

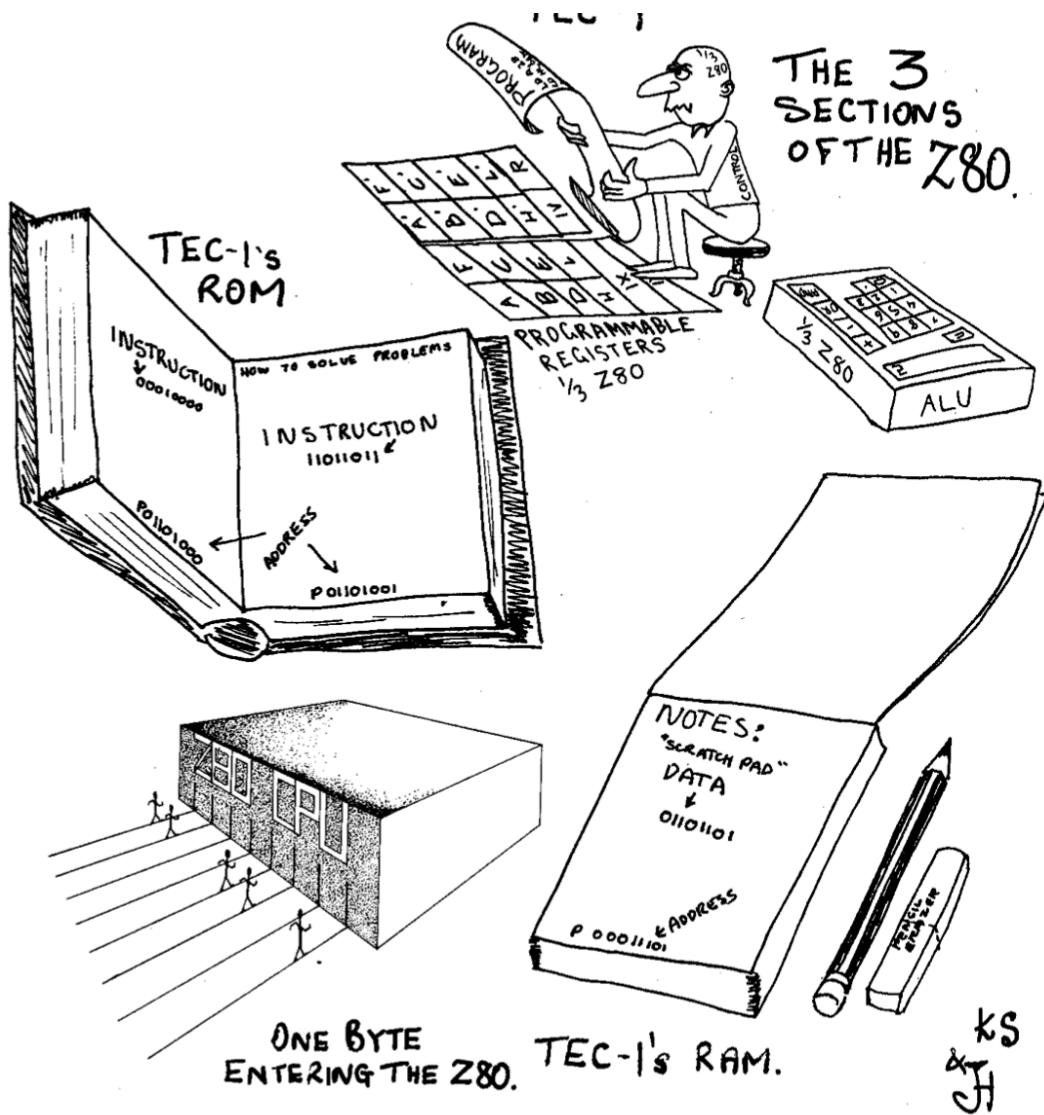
Tests the selected keyboard - the last pressed key's scancode will appear on the 7-segment displays. **Fn** is displayed by bit 5 set. Matrix keypad keys supported by MON3 (*NOT* the full matrix keyset) will be returned if MATRIX mode is enabled. Pressing the ADDR key exits to TMON.

**FILL xxxx yyyy nn**

Fill memory between address xxxx and yyyy with data nn. note: Fill range must be at least 2 bytes long. Does not do any checks for safety - use with caution, as you can overwrite any area of memory including the stack, program code or data. This does not apply if Protect Mode is on.

**PRINT your-text-here**

**your-text-here** is echoed back to the serial terminal.





# TEC Magazine Code on the TEC-1G

A great way to learn how to use the TEC-1G is to key in programs presented in the TE Magazines Issues 10 to 15. If the programs are keyed in directly, they probably won't work! This is because they usually start at addresses **0800H** or **0900H**. These addresses are reserved for Mon3. To get the code working, simply update all 2-byte address references to match the address location of the code on the 1G.

Keypad interactions are a bit more complicated. The old monitors use the register **I** and the NMI (Non-Maskable Interrupt) to trigger and save a keypad press. Mon3 uses 'Polling' instead and **RST/API** calls to do keypad reading. See the next chapter for more information on RST and API calls.

Below is a conversion table to help convert older code to work on Mon3 when a keypad press is required.

Old Command	Mon3 Replacement	Reason
<b>HALT</b>	<b>RST 08H</b>	<b>RST 08H</b> simulates a <b>HALT</b> command and sets register <b>A</b> with the key value pressed.
<b>LD A,I</b>	<b>LD C,10H</b> <b>RST 10H</b>	A <b>LD A,I</b> by itself is 'polling' for a key press. Call the scanKey API routine ( <b>10H</b> ) which sets register <b>A</b> with the key value pressed. If <b>LD A,I</b> is immediately after a <b>HALT</b> instruction, then just use <b>RST 08H</b> as described above.

Here is an example of magazine code at **0800H** with key input converted to use Mon3 at RAM address **4000H**. The code in **RED** has been modified.

<b>LD A,80</b>	<b>800</b>	<b>3E 80</b>	<b>LD A,80H</b>	4000	3E 80
<b>OUT (2),A</b>	<b>802</b>	<b>D3 02</b>	<b>OUT (2),A</b>	4002	D3 02
<b>LD B,03</b>	<b>804</b>	<b>06 03</b>	<b>LD B,03H</b>	4004	06 03
<b>LD A,B</b>	<b>806</b>	<b>78</b>	<b>LD A,B</b>	4006	78
<b>OUT (1),A</b>	<b>807</b>	<b>D3 01</b>	<b>OUT (1),A</b>	4007	D3 01
<b>HALT</b>	<b>809</b>	<b>76</b>	<b>RST 08H</b>	4009	CF
<b>LD A,I</b>	<b>80A</b>	<b>ED 57</b>	<b>CP 10H</b>	400A	FE 10
<b>CP 10</b>	<b>80C</b>	<b>FE 10</b>	<b>JP NZ,4014H</b>	400C	C2 14 40
<b>JP NZ 0816</b>	<b>80E</b>	<b>C2 16 08</b>	<b>RLC B</b>	400F	CB 00
<b>RLC B</b>	<b>811</b>	<b>CB 00</b>	<b>JP 4006H</b>	4011	C3 06 40
<b>JP 806</b>	<b>813</b>	<b>C3 06 08</b>	<b>CP 0CH</b>	4014	FE 0C
<b>CP C</b>	<b>816</b>	<b>FE 0C</b>	<b>JP NZ,4009H</b>	4016	C2 09 40
<b>JP NZ 809</b>	<b>818</b>	<b>C2 09 08</b>	<b>RRC B</b>	4019	CB 08
<b>RRC B</b>	<b>81B</b>	<b>CB 08</b>	<b>JP 4006H</b>	401B	C3 06 40
<b>JP 806</b>	<b>81D</b>	<b>C3 06 08</b>			

# Advanced Programming

To assist when developing Z80 programs, Mon3 contains inbuilt functionality that makes it easy to interface with the TEC-1G hardware.

## RST (Restart) commands

RST commands on the Z80 are one-byte call commands that execute code at certain address locations defined by the Z80. The following table outlines the routines.

Command	Op Code	Description
<b>RST 00H</b>	<b>C7</b>	Software monitor reset.
<b>RST 08H</b>	<b>CF</b>	Key wait and press routine. This simulates a HALT command where the TEC will wait for a key to be pressed and continue execution. If a key is currently being held down, the routine will wait first until the key is released and then detect the next key. The key that has been pressed will be stored in register A. EG:  <b>RST 08H</b> ; Wait for keypress <b>LD B,A</b> ; Load key to register B
<b>RST 10H</b>	<b>D7</b>	API entry call. Executes a monitor routine. See the API calls section below for more details.
<b>RST 18H</b>	<b>DF</b>	API 2 entry call. Graphical LCD routine entry. See the GLCD section below for more details.
<b>RST 20H</b>	<b>E7</b>	Scan Seven Segments and Key. Multiplex the seven-segment displays and check for a key press. It can be used to display information on the seven segments and check for a key to be pressed. It must be called in a loop until a key is pressed to maintain 7 segment persistence.. Returns Zero flag set when a key is pressed and Register A with the key value. Register DE points to the seven-segment data. See the first program in the Quick Start Programs chapter for an example.

<b>RST 28H</b>	<b>EF</b>	<p>LCD Busy Check. To be called prior to sending a command to the LCD if directly communicating with the LCD. The routine will only exit when the LCD Busy flag is not set. EG:</p> <pre> <b>RST 28H</b>      ; Check LCD busy flag <b>LD A,01H</b>     ; Load A with clear screen <b>OUT (04),A</b>   ; Send instruction to LCD </pre>
<b>RST 30H</b>	<b>F7</b>	<p>Breakpoint entry. Break execution of the code at the current address location. See the Debugging Programs chapter for more details.</p>
<b>RST 38H</b>	<b>FF</b>	<p>Maskable interrupt handler. Jumps here with Interrupts Enabled (<b>EI</b>), Interrupt Mode 1 (<b>IM 1</b>) and when the INT pin on the CPU goes low. Mon3 will do nothing when this happens. However, a user-defined routine can be used. See the Interrupt section below on how to do this.</p>

## Interrupts

The Z80 CPU has the ability to interrupt the execution of code, handle the interrupt and then resume code execution. This is done in software with Interrupts Enabled (**EI**) and Interrupt Mode 1 (**IM 1**) and by hardware when the INT line on the CPU goes low. Mon3 ignores interrupts, but a user-defined routine can be provided to handle the interrupt. To do this, the address of the interrupt routine is to be placed at RAM address **0892H**.

```

ei           ; Enable interrupts
im 1        ; Interrupt mode 1
ld hl,myINT ; Interrupt routine
ld (0892H),hl ; Save address in 0892H
... continue

myINT:
ld c,03H    ; Bell routine
rst 10H    ; Call API
reti       ; Exit Int routine

```

This code will sound a bell tone in the speaker when an interrupt occurs.

## NMI (Non-Maskable Interrupts)

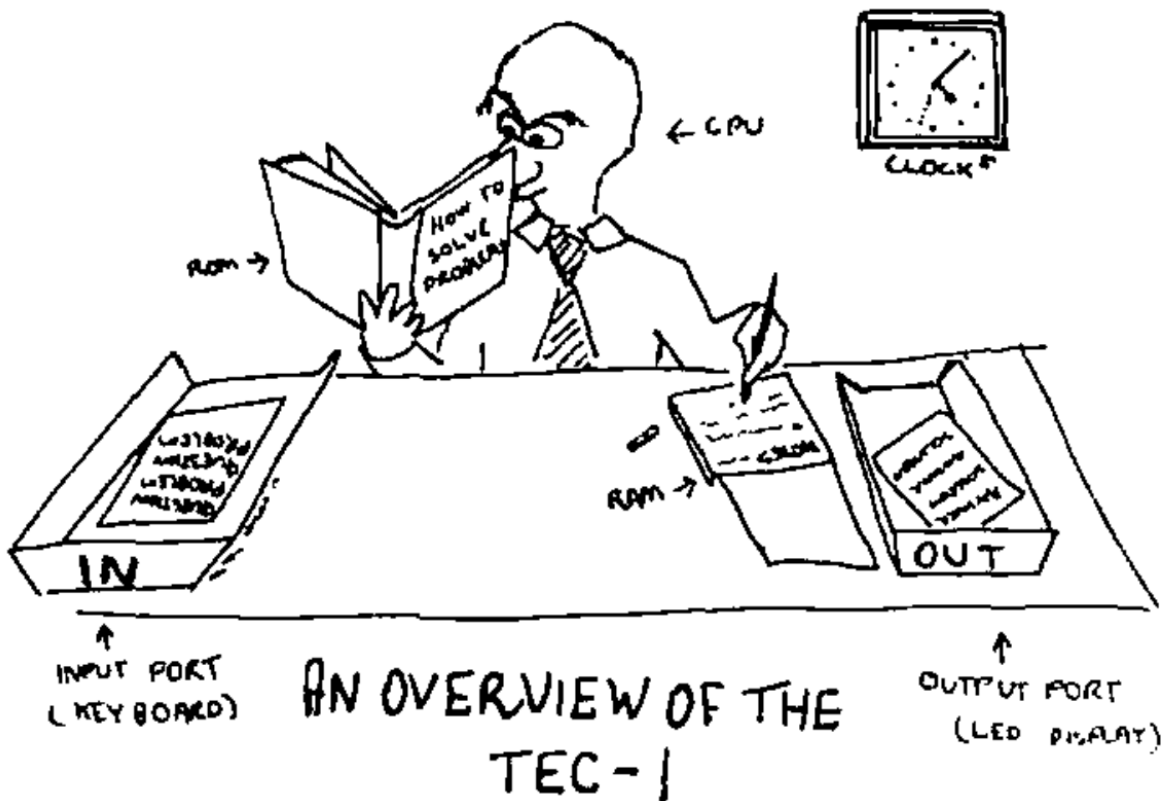
Non-Maskable Interrupts occur when the NMI line on the CPU goes low. These interrupts will always trigger. Mon3 ignores the NMI line, but a user-defined routine can be provided to handle the interrupt. To do this, the address of the interrupt routine is to be placed at RAM address **0894H**.

```
ld hl,myNMI    ; NMI routine
ld (0894H),hl  ; Save address in 0894H
... continue
```

**myNMI :**

```
ld c,03H      ; Bell routine
rst 10H       ; Call API
retn          ; Exit NMI routine
```

This code will sound a bell tone in the speaker when an NMI occurs. The TEC-1G has an NMI jumper that can set NMI to trigger on a Keypad press, a HALT instruction or externally (no jumper).



Credit: Ken Stone

## API (Application Programming Interface) commands.

The API on Mon3 exposes routines used by Mon3 which can be used in your own programs. No need to rewrite the world! But more importantly, it makes writing code quicker and easier with most of the complicated stuff removed.

### General conventions

The register **c** holds the API Call number. All other registers except the **IX** register can be used as parameters if needed. Executing a **RST 10H** or **D7** calls the API.

#### General Interface

```
ld c,[API Call Number]
rst 10H
```

#### Some Examples

```
                                ;Produce a short Beep from the speaker
0E 03      ld c,3                ;beep call number
D7         rst 10H

                                ;Display the letter 'G' on the LCD Screen
0E 0E      ld c,14               ;charToLCD call number
3E 47      ld a,"G"             ;parameter
D7         rst 10H

                                ;Wait for a period of time
0E 21      ld c,33               ;timeDelay call number
21 00 30   ld hl,3000H          ;parameter
D7         rst 10H
```

To assist with API call number references, the file `api_includes.z80`, in the GitHub repository, contains the API Call Number with its Text equivalent for use with your own code.

See <https://github.com/MarkJelic/TEC-1G/tree/main/ROMs/MON3/source>

## API Call List

Utility Calls	#	0x
softwareID	0	0
versionID	1	01
preInit	2	02
beep	3	03
convAToSeg	4	04
regAToASCII	5	05
ASCIIToSegment	6	06
stringCompare	7	07
HLToString_	8	08
AToString	9	09
scanSegments	10	0A
displayError	11	0B
checkStartEnd	30	1E

LCD Calls	#	0x
LCDBusy	12	0C
stringToLCD	13	0D
charToLCD	14	0E
commandToLCD	15	0F

Menu Calls	#	0x
menuDriver	31	1F
paramDriver	32	20
menuPop	47	2F

Serial Calls	#	0x
serialEnable	20	14
serialDisable	21	15
txByte	22	16
rxByte	23	17
intexHexLoad	24	18
sendToSerial	25	19
receiveFromSerial	26	1A
sendAssembly	27	1B
sendHex	28	1C
genDataDump	29	1D
stringToSerial	45	2D

Input Calls	#	0x
scanKeys	16	10
scanKeysWait	17	11
matrixScan	18	12
joystickScan	19	13
matrixScanASCII	53	35

Sound Calls	#	0x
playNote	34	22
playTune	35	23
playTuneMenu	36	24

System Latch Call	#	0x
getCaps	37	25
getShadow	38	26
getProtect	39	27
getExpand	40	28
setCaps	41	29
setShadow	42	2A
setProtect	43	2B
setExpand	44	2C

Misc. Calls	#	0x
timeDelay	33	21
RTCAPI	46	2E
random	49	31
setDisStart	50	32
getDisNext	51	33
getDisassembly	52	34

## API Utility Calls

### **softwareID #0**

Get Software ID String

- Input: nothing
- Return: HL = Pointer to SOFTWARE ASCII String
- Destroy: none

### **versionID #1**

Get Version Number and Version String

- Input: nothing
- Return: HL = Pointer to Release ASCII String  
BC = Release major version number  
DE = Release minor version number
- Destroys: none

### **preInit #2**

Performs a cold reset as if the TEC-1G had just been powered on. Returns to MON3 to its default state.

### **beep #3**

Makes a short beep tone to the TEC Speaker

- Input: nothing
- Destroys: A

### **convAToSeg #4**

Convert register A to Seven Segment display format

- Inputs: A = byte to convert  
DE = address to store segment values (2 bytes)
- Destroys: BC

### **regAToASCII #5**

Convert register A to ASCII. IE: **2CH** -> "2C"

- Input: A = byte to convert
- Output: HL = two-byte ASCII string
- Destroys: A

### **ASCIItoSegment #6**

ASCII to Segment. Converts an ASCII character to Seven Segment display format

- Input: A = ASCII character
- Return: A = Segment character or 0 if out of range
- Destroys: none

### **stringCompare #7**

Compare two string

- Input: HL = source pointer  
DE = target pointer  
B = #bytes to compare (up to 256)
- Output: Zero Flag Set = compare match
- Destroys: HL, DE, A, BC

### **HLToString #8**

Convert HL to ASCII string. IE: **2C0FH** -> "2COF"

- Input: HL = value to convert  
DE = address of string destination (4 bytes)
- Output: DE = address one after last ASCII entry
- Destroys: A

### **AToString #9**

Convert register A to ASCII string. IE: **2CH** -> "2C"

- Input: A = byte to convert  
DE = address of string destination (2 bytes)
- Output: DE = address one after last ASCII entry
- Destroys: A

### **scanSegments #10**

Multiplex the Seven Segment displays with the contents of DE. Must be called repetitively for segments to stay persistent.

- Inputs: DE = pointer to 6-byte location of segment data
- Destroys: A, B, DE = DE + 6

### **displayError #11**

Display ERROR on the Seven Segments and wait for keypress

- Input: none
- Destroys: all



### **checkStartEnd #30**

Check start and end address differences.

- Input: HL = address location of START value  
HL+2 = address location of END value
- Output: HL = start address  
BC = length of end-start  
Carry = set if end is less than start
- Destroys: DE

## API LCD Calls

### **LCDBusy #12**

LCD busy check. Checks the LCD busy flag and loops until LCD isn't busy

- Input: nothing
- Destroys: none

### **stringToLCD #13**

ASCII string to LCD. Writes a string (text) to the current cursor location on the LCD

- Input: HL = ASCII string terminated with a zero byte
- Destroy: A, HL (moves to end of the list)

```
TEXT: .db "HELLO TEC!",0
```

```
ld hl,TEXT
ld c,13
rst 10h
```

### **charToLCD #14**

ASCII character to LCD. Writes one character to the LCD at the current cursor location

- Input: A = ASCII character
- Destroy: none

```
ld a,"G"
ld c,14
rst 10h
```

### **commandToLCD #15**

Command to LCD. Sends an LCD instruction to the LCD

- Input: B = Instruction byte
- Destroy: none

```
ld b,01 ;clear LCD
ld c,15
rst 10h
```

## API Input Calls

### scanKeys #16

Universal Key input detection routine. Supports HexPad and Matrix. The routine does not wait for a key press the returns immediately. Only Hexpad keys are detected if using the Matrix Keyboard.

- Return: A = key value (if the following is met)
  - zero flag set if a key is pressed
  - carry flag set if press detected of a new key
  - carry flag not set for a key pressed and held or if no key has been pressed
- Destroys: DE if using Matrix Keyboard

#### Key mapping returned in register A

0-F	= 00-0F	Fn-0-F	= 20-2F (Bit 5 set)
Plus	= 10	Fn-Plus	= 30
Minus	= 11	Fn-Minus	= 31
G0	= 12	Fn-G0	= 32
AD	= 13	Fn-AD	= 33

### scanKeysWait #17

Generic Key input detection routine. Supports HexPad and Matrix. Waits until a key is pressed. The routine will only detect a key if all keys are released first. Only Hexpad keys are detected if using the Matrix Keyboard.

- Return: A = key value (if following are met)
  - zero flag set if a key is pressed
- Destroys: DE if using Matrix Keyboard

See table above for return values in register A

### joystickScan #19

Joystick port scan routines. This routine will return a value based on the movement/button of the joystick or any combination: IE: UP+DOWN = 03H, Routine must be called repetitively.

- Input: None
- Output: A = Joystick return value between 00H-5FH (0-95)
  - 01H = Up
  - 02H = Down
  - 04H = Left
  - 08H = Right
  - 10H = Fire 2
  - 20H = Comm2 (Pin 9)
  - 40H = Fire 1
  - 80H = Fire 3
  - zero flag set if no joystick value returned
- Destroy: none

### matrixScan #18

Key scan routine for the Matrix Keyboard. This routine detects up to two key presses at the same time. Key values stored in DE. The routine must be called repetitively.

- Input: None
- Output: E = Key pressed between 00H-3FH (0-63)  
D = Second key, FF=no key, 00=shift, 01=Ctrl, 02=Fn  
zero flag set if a key is pressed or combination valid

#### Key mapping returned in register E (note: some gaps are present)

Shift = 00	Esc = 0C	4 = 17	D = 27	0 = 32	Z = 3D
Ctrl = 01	Space = 0D	5 = 18	E = 28	P = 33	\ = 3F
Fn = 02	Single Qt = 0E	6 = 19	F = 29	Q = 34	
Up = 03	Comma = 0F	7 = 1A	G = 2A	R = 35	
Down = 04	Minus = 10	8 = 1B	H = 2B	S = 36	
Left = 05	F.Stop = 11	9 = 1C	I = 2C	T = 37	
Right = 06	/ = 12	; = 1E	J = 2D	U = 38	
Caps = 07	0 = 13	= = 20	K = 2E	V = 39	
Del = 08	1 = 14	A = 24	L = 2F	W = 3A	
Tab = 09	2 = 15	B = 25	M = 30	X = 3B	
Enter = 0A	3 = 16	C = 26	N = 31	Y = 3C	

### matrixScanASCII #53

Convert the output of the matrixScan routine to ASCII. matrixScan returns values between 0 and 64, these represent key presses on the keyboard. This routine will convert the output of matrixScan DE, to the actual key pressed in ASCII. If the key doesn't map to an ASCII character then the matrix key value is returned.

- Input: DE = value return from matrixScan.  
E = key, D = Secondary key
- Output: A = key pressed in ASCII
- Destroy: BC, HL

Example code on using matrixScanASCII can be found in the Quick Start Programs chapter below.

## API Serial Data Transfer Calls

### **serialEnable #20**

Activates the BitBang serial port for serial transmit. Disco LED's glow blue to indicate ready status.

- Input: none
- Destroy: A

### **serialDisable #21**

Deactivates the BitBang serial port for serial transmit. Disco LEDs turn off.

- Input: none
- Destroy: A

### **txByte #22**

Bit Bang FTDI USB transmit routine. Send one byte via the FTDI USB serial connection. It assumes a UART connection of 4800-8-N-2.

- Input: A = byte to transmit
- Output: nothing
- Destroy: none

### **rxByte #23**

Bit Bang FTDI USB receive routine. Receive one byte via the FTDI USB serial connection. It assumes a UART connection of 4800-8-N-2. Note routine will wait until a bit is detected.

- Input: nothing
- Return: A = byte received
- Destroy: none

### **intelHexLoad #24**

Load an Intel Hex file via the FTDI USB serial connection. Displays file progress on the segments and PASS or FAIL at the end of the load. Intel Hex file format is a string of ASCII with the following parts:

MARK	LENGTH	ADDRESS	RECORD TYPE	DATA	CHECKSUM	
:	10	2000	00	210621CD7D20CD98203A00213C320021AF		<- EXAMPLE LINE

MARK is a colon character, LENGTH is the number of bytes per line, ADDRESS is the 2-byte address of where the data is to be stored. RECORD TYPE is 00 for Data and 01 for EOF. DATA is the bytes to be stored. CHECKSUM is the addition of all bytes in one line.

- Input: nothing
- Output: nothing
- Destroy: HL,DE,BC,A

### **sendToSerial #25**

SIO Binary Dump. Transfer data on the TEC to a serial terminal. From address and Length of data is needed for input. Use **checkStartEnd** to get length if using From/To address.

- Input: HL = start address  
DE = length in bytes of data to send
- Destroys: A,HL,DE,BC

### **receiveFromSerial #26**

SIO receives binary data. Receive binary data from FTDI. From address and Length of data is needed for input. Use **checkStartEnd** to get length if using From/To address.

- Input: HL = start address  
DE = length in bytes of data to receive
- Destroys: A,HL,DE,BC

### **sendAssembly #27**

Send Assembly instructions to the serial port. Print out the disassembled code that is on the TEC in readable assembly language on the serial terminal. From address and Length of data is needed for input. Use **checkStartEnd** to get length if using From/To address.

- Input: HL = start address  
DE = length in bytes of data to disassemble
- Destroys: A,HL,DE,BC

### **sendHex #28**

Send a traditional HEX dump as text to the serial terminal. Up to 16 bytes are displayed per line. From address and Length of data is needed for input. Use **checkStartEnd** to get length if using From/To address.

- Input: HL = start address  
DE = length in bytes of data to send as Hex
- Destroys: A,HL,DE,BC

### **genDataDump #29**

Generate data dump in ASCII. Print the Address and then B number of bytes. This routine is a subroutine in the \_sendHex routine.

- Input: B = number of bytes to display
  - HL = start address of data dump
  - DE = address of string destination
  - Output: DE = zero terminated address one after last ASCII entry
- ```
          IE: "4000: 23 34 45 56 78 9A BC DE",0
```
- Destroys: A, HL (moves to next address after last byte)

### **stringToSerial #45**

ASCII string to FTDI Serial Port. Writes a string (text) to the serial port

- Input: HL = ASCII string terminated with a zero byte
- Destroy: A, HL (moves to end of the list)

```
TEXT: .db "HELLO TEC!",0
```

```
      ld hl,TEXT
      ld c,55
      rst 10h
```

## API Menu & Parameter Calls

### **menuDriver #31**

Menu driver for user programs. Creates a selectable custom menu/list. Keys: **Go** = Select menu item, **AD** = Exit Menu, **Plus/Minus** = Navigate menu. If a menu item is selected by pressing **Go**, a jump is performed to the menu routine address (see example below). If the user routine ends with a RET instruction, control will be brought back to the menu. There is no need to call the menuDriver again after the routine returns.

If after the RET the menu is to be removed or popped off, then call the **menuPop** routine prior to the RET. This will return control to the previous menu or enter Data Entry mode.

The menu can also be used as a selectable List. Use **menuPop** to close the list once the item has been selected. See an example below on how to do this.

- Input: HL = Pointer to Menu configuration.
- Destroys: A, HL

All strings are ZERO terminated! Except the 7 Segment Text must be ASCII of exactly 6 bytes. Menu configuration is as follows.

```
<# Menu Entries>, <7 Segment Text>, <Menu Text Title>,  
[<Menu Text Label>, <Menu Routine Address>]+  
EG: .db 2                ; Two menu items  
    .db "MyGame"         ; 7 segment text (6 bytes)  
    .db "Games",0        ; Menu title  
    .db "TEC Invaders",0 ; Text and Routine  
    .dw invaders  
    .db "TEC Maze",0     ; Text and routine  
    .dw maze
```



### **paramDriver #32**

Parameter data entry driver. Creates a list of editable two-byte parameters. Keys: **Go** = Continue, **AD** = Exit, **Plus/Minus** = Navigate, **0-F** = enter values

- Input: HL = Pointer to Parameter configuration.

Once the **Go** key is pressed, code will continue after the API call. The parameter view on the LCD will automatically be removed and the LCD will display the prior view to the parameter call. There is no need to call **menuPop** to restore the previous LCD view.

Parameter text can be no longer than 14 characters. Parameters entered will be stored in the Param RAM Address locations of two-bytes each. All strings are ZERO terminated! Except the 7 Segment Text must be ASCII of exactly 6 bytes. Parameter configuration is as follows.

```
<No. of Entries>, <7 Segment Text>, <Parameter Title  
Text>, [<Param Text Label>, <Param RAM Address>]+  
EG: .db 3 ; Three parameters  
    .db "Params" ; 7 segment text (6 bytes)  
    .db "= Enter Parameters =",0 ; Parameter title  
    .db "Start Address:",0 ; Text and Address  
    .dw RAM_LOC_1  
    .db "End Address:",0 ; Text and Address  
    .dw RAM_LOC_2  
    .db "Dest. Address:",0 ; Text and Address  
    .dw RAM_LOC_3
```

### **menuPop #47**

Replace the current menu with its parent menu if any. If menus have been nested, then the parent menu will become the active menu. This is the same as pressing the **AD** key but done in software. If no parent menu exists then the Monitor mode is changed to Data Entry View. Useful if using the menu as a Select List where execution of code is to be continued.

- Input: none.
- Destroys: A

## Menu and Parameter Driver Example

Create a Menu with 3 items. The first item jumps to a routine which is the standard way to use the menu. The second item displays a selectable list that saves a value in RAM and returns to the menu. The last item will create a parameter entry list of four 2-byte items.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> <b>MENUDRIVER</b> .<b>EQU</b> <b>1FH</b> ;Menu API <b>PARAMDRIVER</b> .<b>EQU</b> <b>20H</b> ;Param API <b>MENUPOP</b> .<b>EQU</b> <b>2FH</b> ;Menu Pop API  <b>PROGRAM1</b> .<b>EQU</b> <b>1000H</b> ;Program 1 <b>BAUD</b> .<b>EQU</b> <b>2008H</b> ;Baud value <b>PARAM1</b> .<b>EQU</b> <b>2000H</b> ;two bytes <b>PARAM2</b> .<b>EQU</b> <b>2002H</b> ;per param <b>PARAM3</b> .<b>EQU</b> <b>2004H</b> <b>PARAM4</b> .<b>EQU</b> <b>2006H</b>  ;Create Menu <b>0E 1F</b> <b>ld c</b>,<b>MENUDRIVER</b> <b>21 00 30</b> <b>ld hl</b>,<b>menuCFG</b> ;config <b>D7</b> <b>rst 10H</b> ;API call ;Code continues in menu routines  ;Create Selectable List <b>setBaud:</b> <b>0E 1F</b> <b>ld c</b>,<b>MENUDRIVER</b> <b>21 00 30</b> <b>ld hl</b>,<b>baudCFG</b> ;config <b>D7</b> <b>rst 10H</b> ;API call ;Code continues in menu routines  ;Baud rate saving code <b>baud12:</b> <b>21 00 12</b> <b>ld hl</b>,<b>1200H</b> ;baud rate <b>18 0D</b> <b>jr saveBaud</b> ;cont.. <b>baud24:</b> <b>21 00 24</b> <b>ld hl</b>,<b>2400H</b> ;baud rate <b>18 08</b> <b>jr saveBaud</b> ;cont.. <b>baud48:</b> <b>21 00 48</b> <b>ld hl</b>,<b>4800H</b> ;baud rate <b>18 03</b> <b>jr saveBaud</b> ;cont.. <b>baud96:</b> <b>21 00 96</b> <b>ld hl</b>,<b>9600H</b> ;baud rate <b>saveBaud:</b> <b>22 08 20</b> <b>ld (BAUD)</b>,<b>hl</b> ;save baud <b>0E 2F</b> <b>ld c</b>,<b>MENUPOP</b> <b>D7</b> <b>rst 10H</b> ;API call <b>C9</b> <b>ret</b> ;Return to Main Menu </pre> | <pre> ;Create Parameter Entry <b>createParam:</b> <b>0E 20</b> <b>ld c</b>,<b>PARAMDRIVER</b> <b>21 80 30</b> <b>ld hl</b>,<b>paramCFG</b> ;config <b>D7</b> <b>rst 10H</b> ;API call ...Parameter code continues <b>C9</b> <b>ret</b> ;Return to Main Menu  ;Main Menu Configuration <b>menuCFG:</b> <b>.db 3</b> ;three entries <b>.db "-Menu-"</b> <b>.db "= MENU TITLE =",0</b> <b>.db "Run Program",0</b> <b>.dw PROGRAM1</b> <b>.db "Set Baud Rate",0</b> <b>.dw setBaud</b> <b>.db "Parameters",0</b> <b>.dw createParam</b>  ;Selectable List Configuration <b>baudCFG:</b> <b>.db 4</b> ;four entries <b>.db "BAUDrt"</b> <b>.db "= Select Baud =",0</b> <b>.db "1200",0</b> <b>.dw baud12</b> <b>.db "2400",0</b> <b>.dw baud24</b> <b>.db "4800",0</b> <b>.dw baud48</b> <b>.db "9600",0</b> <b>.dw baud96</b>  ;Parameter Entry Configuration <b>paramCFG:</b> <b>.db 4</b> ;four entries <b>.db "Input "</b> <b>.db "= PARAM TITLE =",0</b> <b>.db "Start Address",0</b> <b>.dw PARAM1</b> <b>.db "End Address",0</b> <b>.dw PARAM2</b> <b>.db "Copy Address",0</b> <b>.dw PARAM3</b> <b>.db "Backup Address",0</b> <b>.dw PARAM4</b> </pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## API Sound Calls

### **playNote #34**

Play a note. Play a note with a given frequency and wavelength

- Input: HL = frequency (01-7F)  
          B = wavelength (00-FF)
- Destroys: HL, BC, A

### **playTune #35**

Play a series of notes. To play a note use a reference between 01H and 18H. Where 01H is the lowest frequency and 18H is the highest frequency. Use 00H for a pause and any value greater than 18H to exit. A single pause can be used to separate notes.

Note reference table is as follows:

|    |     |    |     |    |     |    |     |
|----|-----|----|-----|----|-----|----|-----|
| G  | 01H | C# | 07H | G  | 0DH | C# | 13H |
| G# | 02H | D  | 08H | G# | 0EH | D  | 14H |
| A  | 03H | D# | 09H | A  | 0FH | D# | 15H |
| A# | 04H | E  | 0AH | A# | 10H | E  | 16H |
| B  | 05H | F  | 0BH | B  | 11H | F  | 17H |
| C  | 06H | F# | 0CH | C  | 12H | F# | 18H |

- Input: DE = Address of first note
- Destroy: A, B, DE, HL

### **playTuneMenu #36**

Play a series of notes with the \_playTune routine, but the address of the first note is selected via a parameter menu.

- Input: none
- Destroy: A, B, DE, HL

## API System Latch Calls

### **getCaps #37**

Get Caps lock state

- Input: none
- Output: A = caps lock state; 0 = off, 80H = on

### **getShadow #38**

Get SHADOW state

- Input: none
- Output: A = shadow state; 0 = off, 01H = on

### **getProtect #39**

Get PROTECT state

- Input: none
- Output: A = protect state; 0 = off, 02H = on

### **getExpand #40**

Get EXPAND state

- Input: none
- Output: A = expand state; 0 = off, 04H = on

### **setCaps #41**

Set Caps lock state

- Input: A = Desired caps lock state; 0 = off, 80H = on
- Destroy: A

### **setShadow #42**

Set Shadow state

- Input: A = Desired shadow state; 0 = off, 01H = on
- Destroy: A

### **setProtect #43**

Set Protect state

- Input: A = Desired protect state; 0 = off, 02H = on
- Destroy: A

### **setExpand #44**

Set Expand state

- Input: A = Desired expand state; 0 = off, 04H = on
- Destroy: A

## Miscellaneous Calls

### **timeDelay #33**

A 16-bit delay routine. An input delay of **2000H** is approximately 50ms.

- Input: HL = delay amount
- Destroys: none

### **random #49**

Random number generator. Return a random number between 00H-FFH

- Input: none
- Output: A = pseudo-random number
- Destroy: B

### **setDisStart #50**

Set Disassembly start address. Set the first address for disassembly output

- Input: HL = start address
- Output: none
- Destroy: none

### **getDisNext #51**

Get Disassembly next address. The new start address for the next output.

- Input: none
- Output: HL = start address
- Destroy: none

### **getDisassembly #52**

Generate Disassembly line. Must call **setDisStart** prior. Only need to call **setDisStart** once as the next address is automatically increased.

- Input: none
- Output: HL = pointer to disassembly ASCII, zero terminated
- Destroy: none

### **RCTAPI #46**

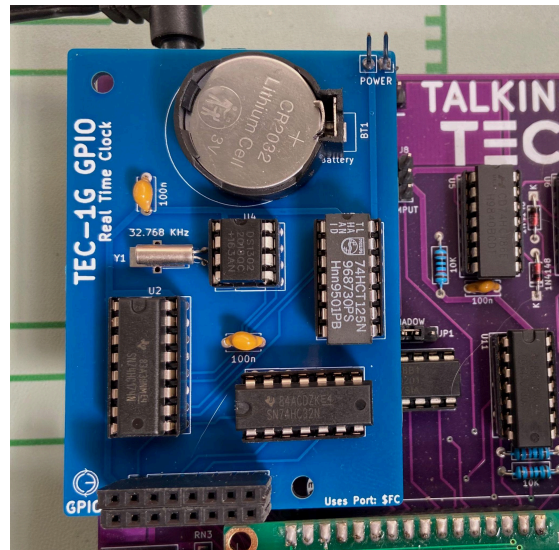
Call a Real Time Clock (RTI) routine for the RTC add on board. See the RTC chapter below for detailed information on this add-on.

- Input: B = RTC routine number  
Other = Depends on the RTC routine

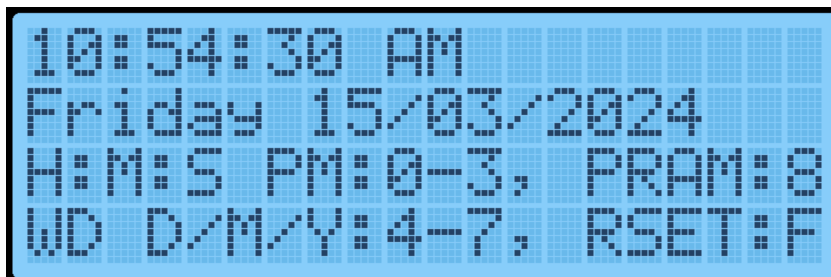
## Real Time Clock (RTC) Add-On Interface

A RTC add-on board that connects to the General Purpose IO port on the TEC-1G can be interfaced with Mon3. The board uses the **DS1302** Real Time Clock chip. The RTC chip is designed to respond on port **FCH**.

The **DS1302** supports 12 and 24 hour clock modes, a 100 year calendar (2000-2099) with leap year support, and **31** bytes of general purpose nonvolatile RAM. The TEC Designers have called the NVRAM, "Parameter RAM" or PRAM.



To initially set the RTC, a convenient RTC Setup routine has been provided in the Settings item in the Main Menu. Select "**Configure RTC**". Press the following keys to update the time/date: **0** = Hour, **1** = Minute, **2** = Second, **3** = 12/24h, **4** = Day of week, **5** = Day, **6** = Month, **7** = Year, **8** = View RTC PRAM, **F** = Reset RTC, **AD** = Exit. When viewing RTC PRAM data, **Plus** = Move Down, **Minus** = Move Up, **AD** = Exit back to RTC Setup.



Mon3 will automatically utilise the internal PRAM to retain some settings when the TEC-1G is powered down. 14 Free bytes are

available to be used by the user. The reserved Mon3 PRAM slots are:

| Slot | Reserved for             |
|------|--------------------------|
| 0-5  | Quick Jump Addresses     |
| 6-11 | Start/End/Dest Addresses |

| Slot  | Reserved for    |
|-------|-----------------|
| 12-15 | Mon3 Future Use |
| 16-29 | User Free RAM   |

| Slot | Reserved for  |
|------|---------------|
| 30   | Mon3 Checksum |

When the RTC board is first used, TEC-1G settings are saved to the PRAM during power on. Manual resetting of the PRAM can also be achieved by selecting the "**Reset RTC PRAM**" option in the Settings item in the Main Menu. This will only reset Mon3 reserved values.

## RTC API Calls

The RTC API uses the standard **rst 10H** call with the addition of the **B** register to specify which RTC API function is required. In this way, all RTC functions only occupy a single Mon3 API call.

### General Interface

```
ld c,2EH          ;RTC API call number
ld b,[RTC Call Number]
rst 10H
```

### Some Examples

```
                                ;Get the current time
01 2E 02 ld bc,022EH          ;getTime + RTC API
D7      rst 10H

                                ;Set the current time to 10:24:46
01 2E 03 ld bc,032EH          ;setTime + RTC API
21 00 30 ld hl,1024H          ;10 hours, 24 minutes
16 46    ld d,46H             ;46 seconds
D7      rst 10H

                                ;Write a byte to the RTC NV Ram
01 2E 0C ld bc,0C2EH          ;writeRTCByte + RTC API
11 FF 02 ld de,02FFH          ;Save FF in position 02
D7      rst 10H
```

| RTC Routine | # | Ox |
|-------------|---|----|
| checkDS1302 | 0 | 0  |
| resetDS1302 | 1 | 01 |
| getTime     | 2 | 02 |
| setTime     | 3 | 03 |
| getDate     | 4 | 04 |
| setDate     | 5 | 05 |
| getDay      | 6 | 06 |

| RTC Routine  | #  | Ox |
|--------------|----|----|
| setDay       | 7  | 07 |
| get1224Mode  | 8  | 08 |
| set12HrMode  | 9  | 09 |
| set24HrMode  | 10 | 0A |
| readRTCByte  | 11 | 0B |
| writeRTCByte | 12 | 0C |
| burstRTCRead | 13 | 0D |

| RTC Routine | #  | Ox |
|-------------|----|----|
| BCDToBin    | 14 | 0E |
| binToBCD    | 15 | 0F |
| formatTime  | 16 | 10 |
| formatDate  | 17 | 11 |
| RTCSetup    | 18 | 12 |

### **checkDS1302 #0**

Check if a DS1302 is detectable, by verifying that the DS1302's registers return expected results.

- Input: none
- Output: Carry flag set = no RTC add-on board present
- Destroy: A

### **resetDS1302 #1**

Resets the DS1302 to a known state - clears existing Time and Calendar. Does not clear RTC RAM. Sets DS1302 to 01:00.00 AM, 01/01/2000.

- Input: none
- Destroy: none

Note: To be used **only** when the RTC requires a settings reset e.g. if it's not "ticking". Use **checkDS1302** to "reset" the DS1302 to a ready state, as part of program initialization.

### **getTime #2**

Get time from RTC. Time is formatted in either 12 or 24 hour mode, depending on selected mode.

- Input: none
- Output: H = hour, bit 5=am/pm flag (in 12hr mode). 1=PM  
L = minute  
D = second
- Destroy: A

Note that all returned registers are BCD coded, so 10:24:36 results in HL=1024h, D=36h

### **setTime #3**

Sets the time in the RTC chip. Time is formatted in either 12 or 24 hour mode, depending on selected mode.

- Input: H = hour, bit 5=am/pm flag (in 12hr mode). 1=PM  
L = minute  
D = second
- Destroy: A,E

The 12/24 hour mode flag is preserved. Note that all registers are BCD coded, so 10:24:36 is formatted as HL=1024h, D=36h



#### **getDate #4**

Returns the present Calendar date, month, year.

- Input: none
- Output: H = date  
          L = month  
          DE = year
- Destroy: A

Note that values returned are BCD coded.

#### **setDate #5**

Sets the Calendar to a specified date/month/year. Invalid dates may be accepted e.g. **30 February** as the **DS1302** does not validate dates as programmed; it simply rolls over at midnight.

- Input: H = date  
          L = month  
          DE = year 2000-2099, D is assumed to be 20h
- Destroy: A

Note that values returned are BCD coded.

### **getDay #6**

Gets the Day of the week i.e. "Monday", "Tuesday", etc. 01 = Monday, 07 = Sunday.

- Input: none
- Output: D = 01-07 (Day of week)  
HL = address of zero terminated DOW string
- Destroy: A

The names of the days of the week are stored in the Mon3 ROM; HL points to the correct string for that day.

### **setDay #7**

Sets the Day of the week. 01 = Monday, 07 = Sunday.

- Input: D = 01-07 (Day of week)
- Output: Carry Flag set = invalid value supplied
- Destroy: A

### **get1224Mode #8**

Reports if the RTC is presently in 12 or 24 hour mode.

- Input: none
- Output: A = 00H (24hr), 80H (12hr), Zero flag set
- Destroy: none

### **set12HrMode #9**

Sets the RTC to 12 hour mode. That is, the hour is subsequently returned as 01-12, and an AM/PM flag.

- Input: none
- Output: Carry Flag set = already in 12 hr mode
- Destroy: A,D

### **set24HrMode #10**

Sets the RTC to 24 hour mode (also known as Military Time). That is, the hour is subsequently returned as 00-23.

- Input: none
- Output: Carry Flag set = already in 24 hr mode
- Destroy: A,D

### **readRTCByte #11**

Reads a byte from the RTC PRAM.

- Input: D = memory slot to return 0–30
- Output: A = value stored in memory
- Destroy: none

### **writeRTCByte #12**

Writes a byte to the RTC PRAM.

- Input: D = memory slot to write to 0–30  
E = value to store
- Destroy: A

### **burstRTCRead #13**

Reads all 31 RTC PRAM bytes and fills a user-supplied buffer with that data.

The user buffer should be 31 bytes long.

- Input: HL = location to write to (31 bytes)
- Output: HL = moved to address after last byte
- Destroy: A

### **binToBcd #14**

Converts the value in register A from BCD encoded, to binary. i.e. "23h" becomes "23" decimal.

- Input: A = BCD Value to convert
- Output: A = Binary value of BCD
- Destroy: none

### **bcdToBin #15**

Converts the value in register A from binary to BCD. i.e. "52" decimal becomes "52h".

- Input: A = Binary Value to convert
- Output: A = BCD value of Binary
- Destroy: none

### **formatTime #16**

Takes a time and fills a user-supplied buffer with an ASCIIZ string formatted as human-readable text. The user-supplied buffer should be at least 12 bytes long.

Bits 7 and 5 of the hour is used to format the time, if it is a 12hr mode timestamp - AM or PM is appended accordingly.

- Input: H = hour (bit 7 = 12/24hr, 1=12hr mode)  
(bit 5 = am/pm flag, 1=PM)
  - L = minute
  - D = second
  - IY = address of user supplied buffer
- Output: IY = moved to address after last byte
- Destroy: A

### **formatDate #17**

Takes a date and fills a user-supplied buffer with an ASCIIZ string formatted as human-readable text. The user-supplied buffer should be at least 11 bytes long.

Dates are output as DD/MM/YYYY

- Input: H = Date
  - L = Month
  - DE = Year (2000 - 2099)
  - IY = address of user supplied buffer
- Output: IY = moved to address after last byte
- Destroy: A

### **RTCSetup #18**

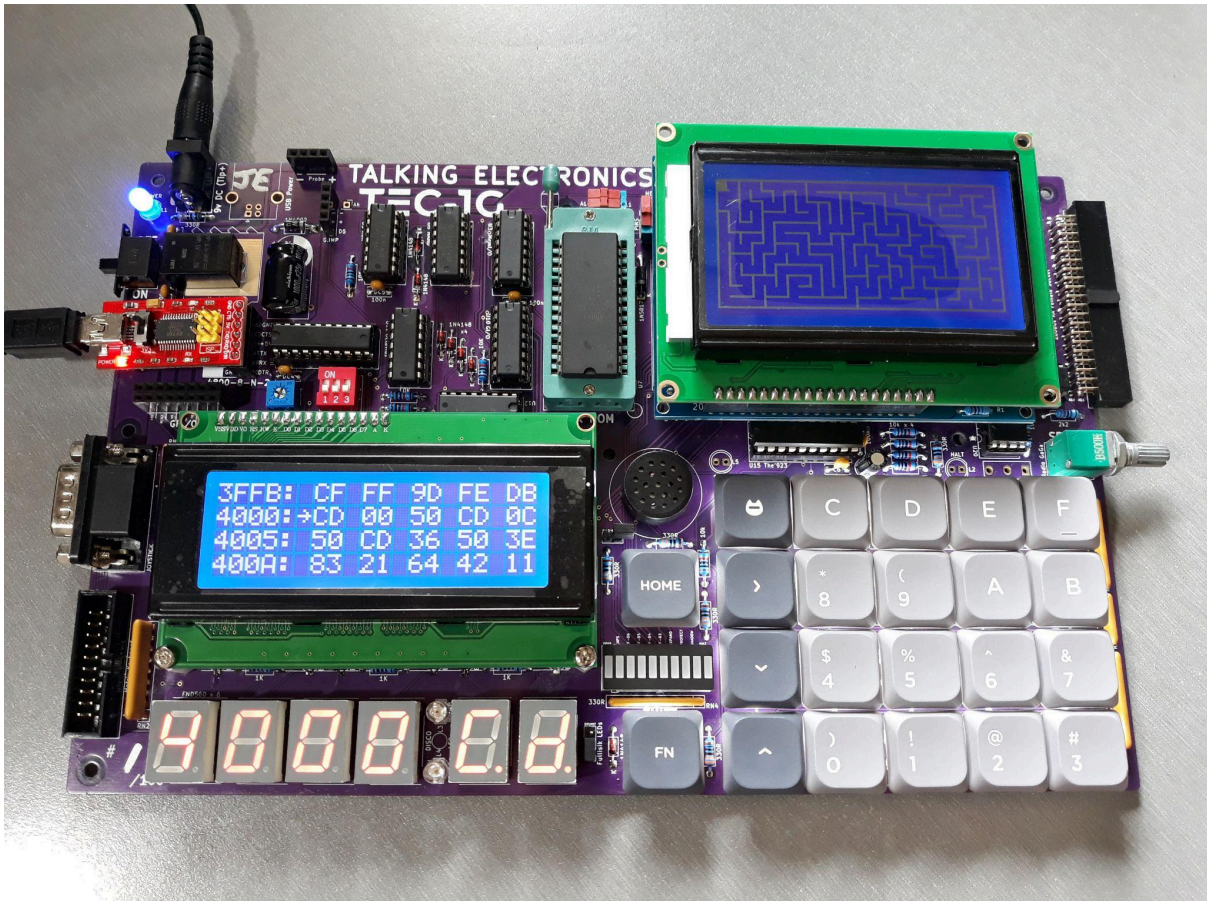
Standalone application that assists with configuring the RTC for initial use. The LCD displays the current RTC time and date along with the instructions.

Keys: **0** = Hour, **1** = Minute, **2** = Second, **3** = 12/24h, **4** = Day of week, **5** = Day, **6** = Month, **7** = Year, **8** = View RTC PRAM, **F** = Reset RTC, **AD** = Exit.

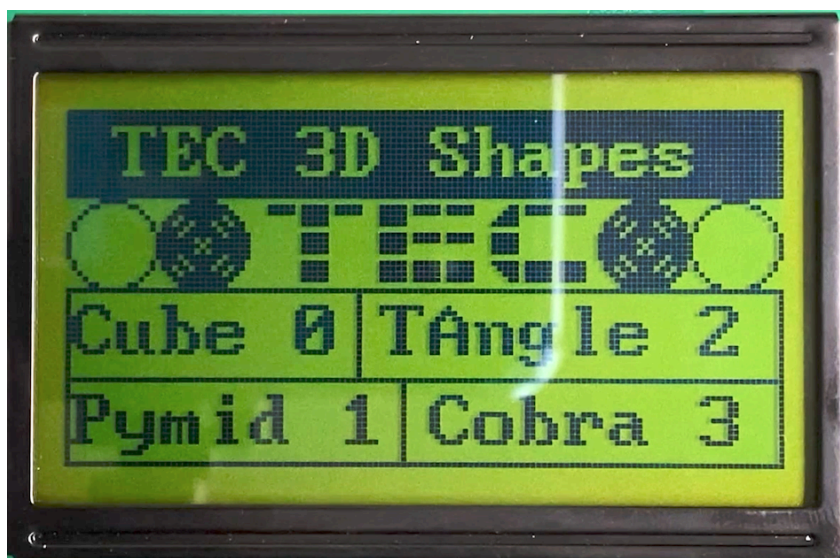
When viewing RTC RAM data, **Plus** = Move Down, **Minus** = Move Up, **AD** = Exit back to RTC Setup.

## Graphical LCD Add-On Interface

Mon3 includes a Graphical LCD (GLCD) library that will work with the TEC-DECK Graphical LCD PCB Add-On. If the Graphical LCD is installed on the TEC-1G via the TEC-DECK headers, special GLCD API calls can be used to interface with the GLCD. The library is for GLCDs with the **ST7920** chip.



The GLCD library contains a variety of routines that can produce simple shapes and lines, these include text, lines, rectangles, circles and pixels.



## General Conventions

The register **a** holds the API Call number. All other registers except the **ix** register can be used as parameters if needed. Executing a **RST 18H** or **DF** calls the GLCD API.

### General Interface

```
ld a,[API Call Number]
rst 18H
```

The following code will draw a box and write text to the GLCD

```
                ; Initialise and set to Graphics Mode
3E 00          ld a,0                ; Initialise GLCD
DF            rst 18H
3E 04          ld a,4                ; Graphics Mode
DF            rst 18H

                ; Draw Box - Box Outline Example
01 20 00      ld bc,0020H           ; X0, Y0
11 3F 7F      ld de,7F3FH           ; X1, Y1
3E 06          ld a,6                ; Draw a outline box from X0,Y0 to X1,Y1
DF            rst 18H

                ; Plot Graphics to LCD Screen (must do)
3E 0C          ld a,12               ; Plot To LCD
DF            rst 18H

                ;Write Text to the Screen
3E 05          ld a,5                ; Text Mode
DF            rst 18H
0E 01          ld c,01H              ; Row 1
3E 0D          ld a,13               ; Print String
DF            rst 18H
54 45 43 2D 31 47 00 .db "TEC-1G",0
```

**initLCD** must be called at the start of every program. The GLCD has two modes, Text and Graphics. Both Text and Graphics can be displayed at the same time. These modes must be selected prior to the drawing or text routine. Also, **plotToLCD** must be called to display any graphics drawn to the screen. The above example displays these to principals.

## GLCD API Calls list

| Routine     | #  | 0x |
|-------------|----|----|
| initLCD     | 0  | 0  |
| clearGBUF   | 1  | 01 |
| clearGrLCD  | 2  | 02 |
| clearTxtLCD | 3  | 03 |
| setGrMode   | 4  | 04 |
| setTxtMode  | 5  | 05 |
| drawBox     | 6  | 06 |
| drawLine    | 7  | 07 |
| drawCircle  | 8  | 08 |
| drawPixel   | 9  | 09 |
| fillBox     | 10 | 0A |

| Routine       | #  | 0x |
|---------------|----|----|
| fillCircle    | 11 | 0B |
| plotToLCD     | 12 | 0C |
| printString   | 13 | 0D |
| printChars    | 14 | 0E |
| delayUS       | 15 | 0F |
| delayMS       | 16 | 10 |
| setBufClear   | 17 | 11 |
| setBufNoClear | 18 | 12 |
| clearPixel    | 19 | 13 |
| flipPixel     | 20 | 14 |

## GLCD API Configure Calls

### **initLCD #0**

Initialise the LCD Screen. This routine is to be called before any other routine.

- Input: nothing
- Destroy: All

### **clearGBUF #1**

Clear the Graphics Buffer. The Graphics Buffer or GBUF is the internal memory area that contains pixel data for the LCD. The drawing routines write data to the GBUF. Once all pixels are set, this buffer is then plotted to the LCD with the **plotToLCD** Routine. Clearing the GBUF is a good way to ensure the pixel area is empty.

- Input: nothing
- Destroy: All

### **clearGrLCD #2**

Clear the Graphics LCD Screen. This routine clears the GDRAM or Graphics screen on the LCD.

- Input: nothing
- Destroy: All

### **clearTxtLCD #3**

Clear the Text LCD Screen. This routine clears the DDRAM or Text screen on the LCD.

- Input: nothing
- Destroy: All

### **setGrMode #4**

Set the LCD to Graphics Mode. This routine puts the LCD in Graphics mode (Extended Instructions) and any further instructions to the LCD will be for the graphics screen. It only needs to be called once if multiple graphics routines are used.

- Input: nothing
- Destroy: AF,DE

### **setTxtMode #5**

Set the LCD to Text Mode. This routine puts the LCD in Text mode (Basic Instructions) and any further instructions to the LCD will be for the text screen. It only needs to be called once if multiple text routines are used.

- Input: nothing
- Destroy: AF,DE



## GLCD API Graphics Calls

### **drawBox #6**

Draws a single-line rectangle between two points X1, Y1 and X2, Y2.

- Input: B = X1-coordinate (0-127)  
C = Y1-coordinate (0-63)  
D = X2-coordinate (0-127)  
E = Y2-coordinate (0-63)
- Destroy: AF,HL

```
ld bc,0020H    ;X0, Y0
ld de,7F3FH    ;X1, Y1
ld a,6         ;drawBox
rst 18H
```

### **drawLine #7**

Draws a straight line between X1, Y1 and X2, Y2. Uses the Bresenham Line drawing algorithm. <http://members.chello.at/~easyfilter/bresenham.html>

- Input: B = X1-coordinate (0-127)  
C = Y1-coordinate (0-63)  
D = X2-coordinate (0-127)  
E = Y2-coordinate (0-63)
- Destroy: All

```
ld bc,0010H    ;X0, Y0
ld de,7F30H    ;X1, Y1
ld a,7         ;drawLine
rst 18H
```

### **drawCircle #8**

Draws a circle from a midpoint to a radius.

- Input: B = Mid-X-coordinate (0-127)  
C = Mid-Y-coordinate (0-63)  
E = Radius (1-63)
- Destroy: All

```
ld bc,0818H    ;Mid X, Mid Y
ld e,08H       ;Radius
ld a,8         ;drawCircle
rst 18H
```

### **drawPixel #9**

Draws a single Pixel.

- Input: B = X-coordinate (0-127)  
          C = Y-coordinate (0-63)
- Destroy: AF,HL

```
ld bc,4020H    ;X,Y
ld a,9         ;drawPixel
rst 18H
```

### **fillBox #10**

Draws a filled rectangle between X1, Y1 and X2, Y2.

- Input: B = X1-coordinate (0-127)  
          C = Y1-coordinate (0-63)  
          D = X2-coordinate (0-127)  
          E = Y2-coordinate (0-63)
- Destroy: AF,HL

```
ld bc,0020H    ;X0, Y0
ld de,7F3FH    ;X1, Y1
ld a,10        ;fillBox
rst 18H
```

### **fillCircle #11**

Draws a filled circle from a midpoint to a radius. This routine iteratively calls the **drawCircle** routine increasing the radius until it equals the register E. There might be gaps in the filled circle, but hey it looks just like what you get on a BASIC program.

- Input: B = Mid-X-coordinate (0-127)  
          C = Mid-Y-coordinate (0-63)  
          E = Radius (1-63)
- Destroy: All

```
ld bc,1018H    ;Mid X, Mid Y
ld e,08H       ;Radius
ld a,11        ;fillCircle
rst 18H
```

## plotToLCD #12

This routine draws the Graphics Buffer or GBUF to the Graphics LCD screen. It is usually called after one of the drawing routines is called. This routine must be called for any graphics to appear on the GLCD. After plotting the GBUF is cleared. Use **setBufNoClear** to retain the GBUF.

- Input: nothing
- Destroy: All

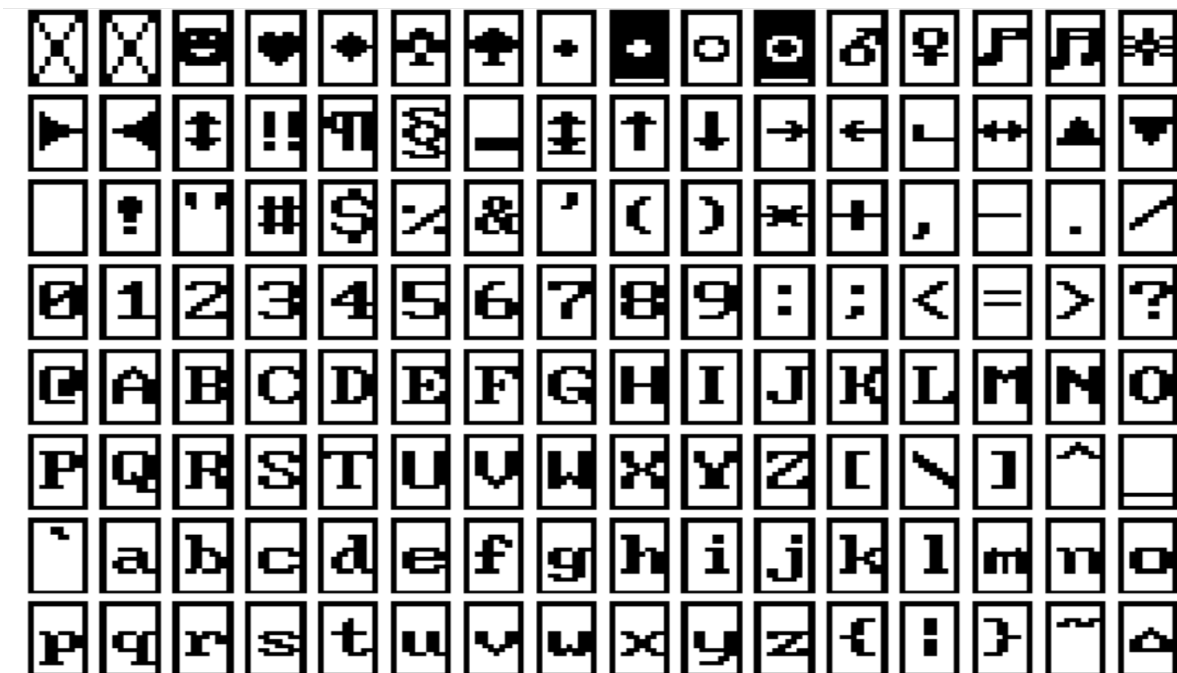
## GLCD API Text Calls

### printString #13

Prints ASCII text on a given row. There are 4 text rows on the LCD screen. The text is to be defined directly **after** the RST 18H routine and is to be terminated with a zero.

- Input: C = row number (0-3)  
Text = "String" on the next line, terminate with 0
- Destroy: All

```
ld c,02H      ;Row 2
ld a,13      ;printString
rst 18H
.db 02H, " This Text ", 1BH ,00H
```



There are 128 characters that are available from 00H-7FH. Conveniently, Alphanumeric characters align with the ASCII table.

### **printChars #14**

Print Characters on the screen in a given row and column. This routine is similar to the one above but character row *and* column placement can be made. Characters to be printed are to be terminated with a zero.

Even though there are 16 columns, only every second column can be written to and two characters are to be printed. IE: if one character is to be printed in column 2, then set B=0 and print " x", putting a space before the character.

- Input: B = column (0-7)  
C = row (0-3)  
HL = start address of text data
- Destroy: All. (HL will be at the end of the text data)

```
ld hl,TEXT_DATA
ld bc,0102H      ;Column 1, Row 2
ld a,14          ;printChars
rst 18H
...
TEXT_DATA:
.db "Hello!",0
```

## GLCD API Utility Calls

### **delayUS #15**

Delay loop for LCD to complete its instruction. Every time a command is sent to the LDC, it requires a small amount of time to complete that operation. IE: setting extended instruction mode. The time needed for most operations is defined in the LDC specification. It is usually around 72us. This routine is used internally, but can also be used directly. The delay time depends on how fast the CPU is running.

- Input: nothing
- Destroy: AF,DE

```
ld a,02H      ;Home instruction
out (07),a    ;send instruction to GLCD
ld a,15       ;delayUS
rst 18H
```

### **delayMS #16**

This is the same as the above routine, but the delay can be software controlled.

- Input: DE = delay value
- Destroy: AF,DE

```
ld a,01H      ;Clear instruction
out (07),a    ;send instruction to GLCD
ld de,0050H   ;longer delay
ld a,16       ;delayMS
rst 18H
```

### **setBufClear #17**

On every **plotToLCD** call, clear the graphics buffer GBUF. Calling this routine will clear the graphics buffer on every draw to the LCD. This is useful if doing animation that requires a new drawing to be displayed on every plot or frame.

- Input: none
- Destroy: AF

### **setBufNoClear #18**

Do not clear the graphics buffer on every **plotToLCD**. Calling this routine will not clear the graphics buffer on every draw to LCD. This is useful for adding graphics data to an existing drawing.

- Input: none
- Destroy: AF

### **clearPixel #19**

Removes or clears a single Pixel from the LCD.

- Input: B = X-coordinate (0-127)  
          C = Y-coordinate (0-63)
- Destroy: AF,HL

```
ld bc,4020H   ;X,Y
ld a,19       ;clearPixel
rst 18H
```

## flipPixel #20

Inverts a single Pixel. If the Pixel is on, it will turn off and if the Pixel is off, it will turn on.

- Input: B = X-coordinate (0-127)  
          C = Y-coordinate (0-63)
- Destroy: AF,HL

```
ld bc,4020H      ;X,Y
ld a,20          ;flipPixel
rst 18H
```

## GLCD Examples

Provided in the TEC-1G GitHub repository are three GLCD programs. The programs have already been converted to Intel Hex files and are ready to load onto the TEC. All programs start at address **2000H**. Source code for all programs are provided and can be changed and studied.

The TEC-1G GitHub account is here: <https://github.com/MarkJelic/TEC-1G> and the GLCD examples are in the TEC-Deck/Graphical\_LCD directory.

### lcd\_3d\_demo

Draw 3D wireframe graphics and rotate them. This program requires keypad input to rotate the objects. Buttons **4,8** and **C** rotate the object in the 3-axis. **Plus** and **Minus** will zoom the object in and out. **0** will return to the main menu. Pressing **GO** will exit the program

### lcd\_mad\_program

Draw the face of Alfred E. Neuman. This program draws lines between two points and creates the face of the Mad Magazine mascot. It draws one line at a time, similar to how it would display on an Apple ][. But if the program is run at **2022H** it will generate instantly. <https://meatfighter.com/mad/>

### lcd\_maze\_gen

Create a maze. This program generates a maze using a recursive backtracking algorithm. Watch the maze slowly generate before your eyes.

Some easy-to-type examples have also been provided in the Quick Start Programs chapter below.

# Quick Start Programs

Who wants the TEC-1G to say Hello? Here are three different ways the TEC can do this. Only a summary of the programs has been provided, making the examples a good exercise for learning how they work. The programs utilise Mon3 API routines as discussed in the Advanced Programming chapter.

This routine is the shortest. It will display the data at **4009** using **RST 20** to multiplex and key scan. If the **AD** key is pressed the routine will exit. Data at **4009** is hardcoded to display HELLO on the seven segments

```
4000 11 09 40    LD DE,4009
4003 E7          RST 20
4004 FE 13       CP 13
4006 20 F8       JR NZ,4000
4008 C9          RET
4009 6E C7 C2    .db 6E C7 C2
400B C2 EB 18    .db C2 EB 18
```

This routine will display HELLO on the LCD Screen. It first clears the LCD by calling **commandToLCD** and then calling **stringToLCD** to display a zero-terminated ASCII string. Press the **AD** key to exit.

```
4000 06 01       LD B,01
4002 0E 0F       LD C,0F
4004 D7          RST 10
4005 21 11 40    LD HL,4011
4008 0E 0D       LD C,0D
400A D7          RST 10
400B CF          RST 08
400C FE 13       CP 13
400E 20 FB       JR NZ,400B
4010 C9          RET
4011 48 45 4C    .db "HEL"
4014 4C 4F 21 00 .db "LO!",0
```

This routine will convert the ASCII "HELLO!" to seven segment code using the **ASCIItoSegment** routine. Then it will use **RST 20** to multiplex and key scan. Change the ASCII at **401A** to display something different.

```
4000 21 1A 40    LD HL,401A
4003 11 20 20    LD DE,2020
4006 06 06       LD B,06
4008 0E 06       LD C,06
400A 7E          LD A,(HL)
400B D7          RST 10
400C 12          LD (DE),A
400D 23          INC HL
400E 13          INC DE
400F 10 F9       DJNZ 400A
4011 11 20 20    LD DE,2020
4014 E7          RST 20
4015 FE 13       CP 13
4017 20 F8       JR NZ,4011
4019 C9          RET
401A 48 45 4C    .db "HEL"
401D 4C 4F 21    .db "LO!"
```

## Matrix Keyboard echo to the Serial Terminal

This program demonstrates how to read in key presses from the Matrix Keyboard, convert the keys to ASCII, handle key bounce and send the ASCII to a serial terminal. *Fun Task:* Modify the program to display on the LCD.

```
MATRIXSCAN      .EQU 12H
SERIALENABLE    .EQU 14H
TXBYTE          .EQU 16H
TOGGLECAPS     .EQU 2DH
MATRIXSCANASCII .EQU 35H
KEY_VALUE       .EQU 2000H ;RAM location of key value

4000 0E 14      LD C,SERIALENABLE ;set serial to send bytes
4002 D7         RST 10H           ;API call
4003 0E 12      LD C,MATRIXSCAN  ;Scan the keyboard
4005 D7         RST 10H           ;API call
4006 28 06      JR Z,400E         ;valid key has been pressed
4008 AF         XOR A            ;reset last key pressed
4009 32 00 20   LD (KEY_VALUE),A
400C 18 F5      JR 4003           ;get next key
400E 3E 07      LD A,07H         ;is the key CAPS LOCK?
4010 BB         CP E
4011 20 09      JR NZ,401C        ;no, then skip caps toggle
4013 3A 00 20   LD A,(KEY_VALUE) ;was the previous key CAPS?
4016 BB         CP E
4017 28 03      JR Z,401C        ;yes, then skip caps toggle
4019 0E 2D      LD C,TOGGLECAPS  ;toggle caps lock flag
401B D7         RST 10H           ;API call
401C 0E 35      LD C,MATRIXSCANASCII ;convert to ASCII
401E D7         RST 10H           ;API call
401F FE 03      CP 03H           ;ignore Shift, Ctrl or Fn if
4021 38 E0      JR C,4003        ;first key pressed
4023 4F         LD C,A
4024 3A 00 20   LD A,(KEY_VALUE) ;ignore key if its the same
4027 B9         CP C            ;as the previous key
4028 28 D9      JR Z,4003
402A 79         LD A,C
402B 32 00 20   LD (KEY_VALUE),A ;store new key pressed
402E 0E 16      LD C,TXBYTE      ;send key pressed to serial
4030 D7         RST 10H           ;API call
4031 18 D0      JR 4003           ;loop back to matrixScan
```



## Seven Segment Scroller via the Serial Terminal

This program reads in text from the serial terminal and scrolls the text on the Seven Segment Displays. Pressing Enter (Carriage Return) will start the scroll. It uses **ASCIITOSEGMENT** to convert ASCII to Seven Segment Display format. *Fun Task:* Modify the program to display text on the LCD.

```
ASCIITOSEGMENT    .EQU 06H
SERIALENABLE      .EQU 14H
TXBYTE           .EQU 16H
RXBYTE           .EQU 17H
START_STR        .EQU 2000H    ;Start of string address
ASCII_STR        .EQU 2002H    ;RAM location of ASCII text

4000 0E 14        LD C,SERIALENABLE ;set serial to send bytes
4002 D7          RST 10H          ;API call
4003 11 02 20    LD DE,ASCII_STR    ;set DE to store ASCII
4006 0E 17        LD C,RXBYTE      ;get a byte from terminal
4008 D7          RST 10H          ;API call
4009 FE 0D        CP 0DH          ;is the byte a CR?
400B 28 0A        JR Z,4017        ;yes jump to scroll routine
400D 0E 16        LD C,TXBYTE      ;echo byte back to terminal
400F D7          RST 10H          ;API call
4010 0E 06        LD C,ASCIITOSEGMENT ;convert ASCII to 7-Seg
4012 D7          RST 10H          ;API call
4013 12          LD (DE),A        ;save modified ASCII
4014 13          INC DE          ;move to next RAM location
4015 18 EF        JR 4006          ;loop for more input
4017 3E FF        LD A,0FFH       ;place FF at end of string
4019 12          LD (DE),A
401A 21 02 20    LD HL,ASCII_STR    ;scroll loop starts here
401D 22 00 20    LD (START_STR),HL ;reset to start of string
4020 26 00        LD H,00H        ;set timer to zero
4022 ED 5B 00 20 LD DE,(START_STR) ;point to start of string
4026 E7          RST 20H          ;scan segments & scan keys
4027 C8          RET Z           ;if key is pressed, exit
4028 25          DEC H           ;delay for full 256 bytes
4029 20 F7        JR NZ,4022      ;repeat multiplex
402B 1A          LD A,(DE)       ;check to see if FF is
402C 3C          INC A           ;the next char to display
402D 28 EB        JR Z,401A       ;it is, go back to begining
402F 21 00 20    LD HL,START_STR  ;shift start by one address
4032 34          INC (HL)        ;(max 254 characters!)
4033 18 EB        JR 4020         ;display scroll again
```

Two GLCD demos are provided to demonstrate how to use the GLCD API calls. The first example is a circle animation that uses graphics mode and the second displays all known fonts on the GLCD which uses text mode.

## Making Bubbles

This program first sets up the LCD to use Graphics and ensures that on every plotToLCD the internal graphics buffer is cleared. This makes the circle animate. Then a circle is expanded until it reaches the end of the screen. A beep is played and the code is repeated.

```

INITLCD           .EQU 0
SETGRMODE        .EQU 4
DRAWCIRCLE       .EQU 8
PLOTTOLCD       .EQU 12
SETBUFCLEAR     .EQU 17
BEEP            .EQU 3
TIMEDELAY       .EQU 33

4000 3E 00          LD A,INITLCD           ;Initialise the GLCD
4002 DF            RST 18H
4003 3E 04          LD A,SETGRMODE        ;Set Graphics Mode
4005 DF            RST 18H
4006 3E 11          LD A,SETBUFCLEAR      ;Set Gr Buffer to Clear
4008 DF            RST 18H
4009 0E 03          LD C,BEEP            ;Play a Beep
400B D7            RST 10H
400C 1E 01          LD E,1                ;Set initial radius to 1
400E 01 20 40       LD BC,4020H          ;Set X,Y to mid screen
4011 C5            PUSH BC                ;Save BC/DE
4012 D5            PUSH DE
4013 3E 08          LD A,DRAWCIRCLE       ;Draw Circle
4015 DF            RST 18H
4016 3E 0C          LD A,PLOTTOLCD        ;Output to LCD
4018 DF            RST 18H
4019 0E 21          LD C,TIMEDELAY        ;Wait a bit
401B 21 00 40       LD HL,4000H
401E D7            RST 10H
401F D1            POP DE                 ;Restore BC/DE
4020 C1            POP BC
4021 1C            INC E                  ;Increase radius by 1
4022 CB 6B          BIT 5,E                ;Check if bubble hits edge
4024 20 E3          JR NZ,4009            ;Yes, reset radius
4026 18 E9          JR 4011                ;No, redraw circle

```

## GLCD Font Display

This program cycles through all stored fonts on the GLCD. Characters on the GLCD are stored in the Character Generator ROM (CGROM). The program sets up the LCD for text mode and displays characters on the screen. Press any key to continue. The code also uses the GLCD ports directly, skipping the API. This is perfectly fine to do. See the ST7920 manual on how to send instructions directly to the GLCD.

```
INITLCD      .EQU 0
SETTXTMODE   .EQU 5
PRINTSTRING  .EQU 13
DELAYUS      .EQU 15

4000 3E 00    LD A,INITLCD      ;Initialise the GLCD
4002 DF      RST 18H
4003 3E 05    LD A,SETTXTMODE   ;Set Text Mode
4005 DF      RST 18H
4006 3E 0D    LD A,PRINTSTRING ;Display Text
4008 DF      RST 18H
4009 20 50 72 65 .DB " Press Any Key",0
400D 73 73 20 41
4011 6E 79 20 4B
4015 65 79 00
4018 0E 00    LD C,0          ;Character Counter
401A CF      RST 08H      ;Wait for key press
401B 06 40    LD B,40H        ;64 Characters per screen
401D 3E 80    LD A,80H        ;row 1 on LCD
401F CD 47 40 CALL 4047      ;Set Row on LCD
4022 79      LD A,C          ;Get Character
4023 CD 4B 40 CALL 404B      ;Display Character on LCD
4026 0C      INC C          ;Next Character
4027 CB 79    BIT 7,C        ;Is C=80H
4029 20 04    JR NZ,402F     ;Yes, display chinese chars
402B 10 F5    DJNZ 4022     ;No, display next character
402D 18 EB    JR 401A      ;Page done, next page
402F 21 40 A1 LD HL,A140H    ;Point to Chinese ROM
4032 CF      RST 08H      ;Wait for key press
4033 06 20    LD B,20H        ;32 Characters per screen
4035 3E 80    LD A,80H        ;row 1 on LCD
4037 CD 47 40 CALL 4047      ;Set Row on LCD
403A 7C      LD A,H          ;Get Character High Byte
403B CD 4B 40 CALL 404B      ;Display Character on LCD
403E 7D      LD A,L          ;Get Character Low Byte
403F CD 4B 40 CALL 404B      ;Display Character on LCD
4042 23      INC HL        ;Next Character
4043 10 F5    DJNZ 403A     ;Display next character
4045 18 EB    JR 4032      ;New Page
4047 D3 07    OUT (07H),A    ;Send instruction to LCD
4049 18 02    JR 404D      ;Do Delay
404B D3 87    OUT (87H),A    ;Send data to LCD
404D 3E 0F    LD A,DELAYUS    ;Set Delay
404F DF      RST 18H
4050 C9      RET
```

## Display a Clock on the Seven Segments

This program requires the RTC Add-on board and will display the current time set on the RTC Board on the Seven Segments.. A check for 12/24 hour mode is done to determine how the Hours are displayed. If in 12 hour mode, Bit 5 is cleared and a decimal point is inserted. Pressing **AD** will quit the program.

```
RTCPRESENT      .EQU 00H
GETTIME         .EQU 02H
GET1224MODE     .EQU 08H
CONVATOSEG      .EQU 04H
RTCAPI         .EQU 46H
DISP_BUFF       .EQU 2000H      ;7 Segment Display Buffer
4000 0E 2E      LD C,RTCAPI      ;RTC API Entry
4002 06 00      LD B,RTCPRESENT  ;Is RTC Board Installed?
4004 D7         RST 10H         ;API call
4005 D8         RET C          ;Carry Set = No, Just Exit
4006 0E 2E      LD C,RTCAPI      ;RTC API Entry
4008 06 02      LD B,GETTIME     ;Get Current RTC Time
400A D7         RST 10H         ;API call
400B 7A         LD A,D          ;Get Seconds
400C 11 04 20   LD DE,DISP_BUFF+4 ;point DE to seconds buffer
400F 0E 04      LD C,CONVATOSEG   ;Convert A to 7 Segment
4011 D7         RST 10H         ;API call saves in DE
4012 7D         LD A,L          ;Get Minutes
4013 11 02 20   LD DE,DISP_BUFF+2 ;point DE to minutes buffer
4016 0E 04      LD C,CONVATOSEG   ;Convert A to 7 Segment
4018 D7         RST 10H         ;API call saves in DE
4019 0E 2E      LD C,RTCAPI      ;RTC API Entry
401B 06 08      LD B,GET1224MODE ;Check if 12 or 24 Hour
401D D7         RST 10H         ;API call
401E 28 0A      JR Z,402A       ;24 Mode, skip AM/PM setup
4020 CB AC      RES 5,H         ;Remove AM/PM Flag (Bit 5
4022 3A 03 20   LD A,(DISP_BUFF+3) ;Get 4th segment value
4025 F6 10      OR 10H         ;Set Decimal Point Segment
4027 32 03 20   LD (DISP_BUFF+3),A ;Save back to segment
402A 7C         LD A,H          ;Get Hour
402B 11 00 20   LD DE,DISP_BUFF ;point DE to hour buffer
402E 0E 04      LD C,CONVATOSEG   ;Convert A to 7 Segment
4030 D7         RST 10H         ;API call saves in DE
4031 11 00 20   LD DE,DISP_BUFF ;point to start of buffer
4034 E7         RST 20H         ;Scan Segments & Key Press
4035 FE 13      CP 13H         ;Is key press "AD" key?
4037 20 CD      JR NZ,4006     ;No, Loop Main Display
4039 C9         RET            ;Exit back to Monitor
```

# Appendix

## Ports

| Port | Direction | Description                                                                                                                                                                                                                                 |
|------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 00H  | In        | Keypad press encoder<br>→ Bit 0-4 HexPad<br>→ Bit 5 Function Key (Active Low)<br>→ Bit 6-7 N/A                                                                                                                                              |
| 01H  | Out       | Seven segment digits switch<br>→ Bit 0-1 Data Segments<br>→ Bit 2-5 Address Segments<br>→ Bit 6 FTDI Rx (Out), Disco LED's<br>→ Bit 7 Speaker                                                                                               |
| 02H  | Out       | Seven segment LED switch<br>→ Bit 0 G segment<br>→ Bit 1 F segment<br>→ Bit 2 C segment<br>→ Bit 3 D segment<br>→ Bit 4 E segment<br>→ Bit 5 DP segment<br>→ Bit 6 B segment<br>→ Bit 7 A segment                                           |
| 03H  | In        | System Input<br>→ Bit 0 Matrix Keyboard (DIP-3)<br>→ Bit 1 Protect Mode (DIP-3)<br>→ Bit 2 Expand Mode (DIP-3)<br>→ Bit 3 Expand Status<br>→ Bit 4 Cartridge Flag<br>→ Bit 5 General Input<br>→ Bit 6 Keypress Flag<br>→ Bit 7 FTDI Tx (In) |
| 04H  | In/Out    | LCD Instruction                                                                                                                                                                                                                             |
| 05H  | Out       | LED 8x8 Matrix Horizontal (TEC Expander)                                                                                                                                                                                                    |
| 06H  | Out       | LED 8x8 Matrix Vertical (TEC Expander)                                                                                                                                                                                                      |
| 07H  | Out       | Graphical LCD Instruction                                                                                                                                                                                                                   |

| Port | Direction | Description                                                                                                                                                                                                                                                                                                            |
|------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 84H  | In/Out    | LCD Data                                                                                                                                                                                                                                                                                                               |
| 87H  | Out       | Graphical LCD Data                                                                                                                                                                                                                                                                                                     |
| F8H  | In/Out    | Spare (TEC Expander & I/O Bus)                                                                                                                                                                                                                                                                                         |
| F9H  | In/Out    | Spare (TEC Expander & I/O Bus)                                                                                                                                                                                                                                                                                         |
| FAH  | In/Out    | Spare (I/O Bus)                                                                                                                                                                                                                                                                                                        |
| FBH  | In/Out    | Spare (General I/O & I/O Bus)                                                                                                                                                                                                                                                                                          |
| FCH  | In/Out    | RTC (Real Time Clock) (General I/O & I/O Bus)                                                                                                                                                                                                                                                                          |
| FDH  | In/Out    | SD (Secure Digital) Flash Card (General I/O)                                                                                                                                                                                                                                                                           |
| FEH  | In        | Matrix Keyboard                                                                                                                                                                                                                                                                                                        |
| FFH  | Out       | System Latch <ul style="list-style-type: none"> <li>→ Bit 0 Shadow (Active Low)</li> <li>→ Bit 1 Protect</li> <li>→ Bit 2 Expand</li> <li>→ Bit 3 FF-D3 (Mem Bus)</li> <li>→ Bit 4 FF-D4 (Mem Bus)</li> <li>→ Bit 5 FF-D5 (Mem Bus)</li> <li>→ Bit 6 FF-D6 (Mem Bus)</li> <li>→ Caps Lock (Matrix Keyboard)</li> </ul> |

## Serial Connection

| Constant                        | Value                                                                                                                                         |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| FTDI to USB Serial Transmission | 4800-8-N-2 <ul style="list-style-type: none"> <li>→ Baud 4800</li> <li>→ 8 Packet Bits</li> <li>→ No Parity</li> <li>→ 2 Stop bits</li> </ul> |

# LCD Cheatsheet

Z80 instructions to communicate with the LCD screen are given as direct commands. IE: OUT (04),A. Mon3 also provides API routines that do the same but also check for the LCD busy state. If using direct port instructions, the LCD busy flag is to be checked prior to the instruction call. The example code provided uses the API routines.

## Instruction Register One Byte commands to configure the LCD Screen

### OUT (04), A

When LCD turns on or resets, screen defaults with 0x01, 0x06, 0x08 and 0x30

| Hex  | Description                | Hex  | Description                     | Hex  | Description                           |
|------|----------------------------|------|---------------------------------|------|---------------------------------------|
| 0x01 | Clear Screen, Cursor reset | 0x0F | Display On, Cursor On and Blink | 0x40 | Set CGRAM Address Pos 1               |
| 0x02 | Return Cursor to top left  | 0x10 | Move Cursor one to the left     |      | (Address from 40-7F)                  |
| 0x04 | Decrement Cursor on write  | 0x14 | Move Cursor one to the right    |      |                                       |
| 0x06 | Increment Cursor on write  | 0x18 | Shift Display to the left       | 0x80 | Set Row 1, Col 1 DDRAM Address        |
| 0x05 | Display to Shift Right     | 0x1C | Shift Display to the right      | 0xC0 | Set Row 2, Col 1 DDRAM Address        |
| 0x07 | Display to Shift Left      | 0x30 | 8-Bit, 1 Line, 5x8 dots         | 0x94 | Set Row 3, Col 1 DDRAM Address (20x4) |
| 0x08 | Display Off, Cursor Off    | 0x34 | 8-Bit, 1 Line, 5x11 dots        | 0xD4 | Set Row 4, Col 1 DDRAM Address (20x4) |
| 0x0C | Display On, Cursor Off     | 0x38 | 8-Bit, 2 Line, 5x8 dots         |      | (Address from 80-A7, C0-E7)           |
| 0x0E | Display On, Cursor On      | 0x3C | 8-Bit, 2 Line, 5x11 dots        |      | See Screen Layout Below               |

**OUT (84), A**  
to write a character to DDRAM

Only write while LCD is not busy

## Screen Layout

### DDRAM Address Counter with Bit 7 set

| Pos   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Row 1 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 |
| Row 2 | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 | D2 | D3 |
| Row 3 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 |
| Row 4 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 |

20x4

**IN A, (84)**  
to read a character from DDRAM

**IN A, (04)**  
read busy flag (bit 7)

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| BF | AC | AC | AC | AC | AC | AC | AC |
|----|----|----|----|----|----|----|----|

| Pos   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | Off Screen |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|
| Row 1 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 89 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90-A7      |
| Row 2 | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0-E7      |

16x2

Writing a character to the screen will increase/decrease the Address Counter automatically

To move the cursor to Row 2, Column 10 do LD A, 0xC9 / OUT (04), A  
For IN A, (04), If Bit 7 is set, then LCD is Busy. Other bits are the current Address Counter

# Character Table

| Lower 4 Bits | Upper 4 Bits | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111                                                                                  |
|--------------|--------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|---------------------------------------------------------------------------------------|
| xxxx0000     | CG RAM (1)   |      |      | 0    | a    | P    | `    | P    |      |      |      | -    | 9    | ≡    | α    | ρ    |                                                                                       |
| xxxx0001     | (2)          |      | !    | 1    | A    | Q    | a    | 9    |      |      |      | μ    | 7    | ≠    | ∠    | ä    | q                                                                                     |
| xxxx0010     | (3)          |      | "    | 2    | B    | R    | b    | r    |      |      |      | 「    | イ    | ツ    | ×    | ρ    | θ                                                                                     |
| xxxx0011     | (4)          |      | #    | 3    | C    | S    | c    | s    |      |      |      | 」    | ウ    | テ    | ε    | ∞    |                                                                                       |
| xxxx0100     | (5)          |      | \$   | 4    | D    | T    | d    | t    |      |      |      | 、    | イ    | ト    | ト    | μ    | Ω                                                                                     |
| xxxx0101     | (6)          |      | %    | 5    | E    | U    | e    | u    |      |      |      | ・    | オ    | 十    | 一    | ε    | Ü                                                                                     |
| xxxx0110     | (7)          |      | &    | 6    | F    | V    | f    | v    |      |      |      | ヲ    | カ    | ニ    | ヨ    | ρ    | Σ                                                                                     |
| xxxx0111     | (8)          |      | '    | 7    | G    | W    | g    | w    |      |      |      | フ    | キ    | ヌ    | ウ    | g    | π                                                                                     |
| xxxx1000     | (1)          |      | (    | 8    | H    | X    | h    | x    |      |      |      | イ    | ウ    | ネ    | リ    | γ    | ×                                                                                     |
| xxxx1001     | (2)          |      | )    | 9    | I    | Y    | i    | y    |      |      |      | ウ    | ケ    | ル    | ル    | "    | y                                                                                     |
| xxxx1010     | (3)          |      | *    | :    | J    | Z    | j    | z    |      |      |      | エ    | コ    | ン    | レ    | j    | ≠                                                                                     |
| xxxx1011     | (4)          |      | +    | ;    | K    | L    | k    | l    |      |      |      | オ    | サ    | ヒ    | ロ    | *    | π                                                                                     |
| xxxx1100     | (5)          |      | ,    | <    | L    | ¥    | l    | l    |      |      |      | ト    | シ    | フ    | ワ    | φ    | π                                                                                     |
| xxxx1101     | (6)          |      | -    | =    | M    | I    | m    | )    |      |      |      | ユ    | ズ    | ハ    | ン    | ト    | ÷                                                                                     |
| xxxx1110     | (7)          |      | .    | >    | N    | ^    | n    | ÷    |      |      |      | ヨ    | セ    | ホ    | ハ    | ñ    |                                                                                       |
| xxxx1111     | (8)          |      | /    | ?    | O    | _    | o    | +    |      |      |      | ッ    | リ    | マ    | "    | ö    |  |

Note: The user can specify any pattern for character-generator RAM.

## Creating Custom Characters

| CGRAM Address | Character In DDRAM |
|---------------|--------------------|
| 40            | 0                  |
| 48            | 1                  |
| 50            | 2                  |
| 58            | 3                  |
| 60            | 4                  |
| 68            | 5                  |
| 70            | 6                  |
| 78            | 7                  |

|   |   |   |   |   |   |   |   |   |      |
|---|---|---|---|---|---|---|---|---|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0x01 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x03 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0x05 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0x09 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0x09 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0x0B |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |   | 0x1B |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |   | 0x18 |

Up to 8 Custom Characters can be programmed

Use Bits 0 to 4 only

After each byte is written CGRAM Address increases

To display character use 0-7 in DDRAM

Use OUT (04), A to set the CGRAM address, where A is between 40h-7Fh  
Then OUT (84), A to write one 5 pixel row

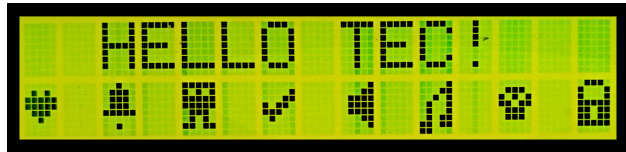


## Example Using CGRAM and DDRAM

```

_stringToLCD .equ 13
_charToLCD .equ 14
_commandToLCD .equ 15
; LCD Setup
ld c,_commandToLCD 4000 0E 0F ;LCD Instruction API routine
ld b,01H 4002 06 01 ;Clear display
rst 10H 4004 D7 ;call API routine
ld b,38H 4005 06 38 ;8-Bit, 2 Lines, 5x8 Characters
rst 10H 4007 D7 ;call API routine
; Tell the LCD that next data will be to CGRAM
ld b,40H 4008 06 40 ;CGRAM entry
rst 10H 400A D7 ;call API routine
; Save multiple characters to CGRAM using lookup table
ld b,40H 400B 06 40 ;8 Characters (8 bytes each)
ld c,_charToLCD 400D 0E 0E ;LCD Data API routine
ld hl,403FH 400F 21 3F 40 ;LCD custom character table
loop1:
ld a,(hl) 4012 7E ;get custom character byte
inc hl 4013 23 ;move to next item in table
rst 10H 4014 D7 ;call API routine
djnz loop1 4015 10 FB ;continue for all 64 char bytes
; Display first line of text
ld c,_commandToLCD 4017 0E 0F ;LCD Instruction API routine
ld b,82H 4019 06 82 ;Move Cursor to Row 1, Col 3
rst 10H 401B D7 ;call API routine
ld hl,4034H 401C 21 34 40 ;ASCII text
ld c,_stringToLCD 401F 0E 0D ;LCD String API routine
rst 10H 4021 D7 ;call API routine
; Display customer characters
ld c,_commandToLCD 4022 0E 0F ;LCD Instruction API routine
ld b,0C0H 4024 06 C0 ;Move Cursor to Row 2, Col 1
rst 10H 4026 D7 ;call API routine
ld b,08H 4027 06 08 ;8 Characters
ld c,_charToLCD 4029 0E 0E ;LCD Data API routine
loop2:
ld a,b 402B 78 ;set A to current character
rst 10H 402C D7 ;call API routine
ld a,20H 402D 3E 20 ;space character
rst 10H 402F D7 ;call API routine
djnz loop2 4030 10 F9 ;continue for all 8 characters
; All Done, what for key press and exit
rst 08H 4032 CF ;key wait and press (HALT)
ret 4033 C9 ;exit

```



```

TEXT TABLE: 4034 48 45 4C 4C 4F 20 54 45 43 21 00 ; "HELLO TEC!"
CHAR TABLE: 403F 00 0A 1F 1F 0E 04 00 00 ; Heart
4047 04 0E 0E 0E 1F 00 04 00 ; Bell
404F 1F 15 1F 1F 0E 0A 1B 00 ; Alien
4057 00 01 03 16 1C 08 00 00 ; Tick
405F 01 03 0F 0F 0F 03 01 00 ; Speaker
4067 01 03 05 09 09 0B 1B 18 ; Note
406F 00 0E 15 1B 0E 0E 00 00 ; Skull
4077 0E 11 11 1F 1B 1B 1F 00 ; Lock

```

# Useful Links

TEC-1G GitHub Repository

<https://github.com/MarkJelic/TEC-1G>

TEC-1 Facebook Page

<https://www.facebook.com/groups/tec1z80>

Z80 Instruction Set Reference

<https://clrhome.org/table/>

Online Z80 Compiler and Debugger

<https://www.asm80.com/>

Rodney Zaks Programming the Z80

<https://archive.org/details/ptz80>

TEC Seven Segment Value Calculator

<https://slartibartfastbb.itch.io/seven-segment-calculator>

Ready? Z80 YouTube Channel (TEC related content)

<https://www.youtube.com/@ReadyZ80>

Mon3 video demonstration

<https://youtu.be/0peIG2HKX3Q>

TEC-1 GitHub Group

<https://github.com/tec1group/>

Talking Electronics Website including original TEC related magazines

[https://www.talkingelectronics.com/te\\_interactive\\_index.html](https://www.talkingelectronics.com/te_interactive_index.html)

# I/O Connectors

