



Engineer's Assistant Using a PIC16F84A

Author: *Voja Antonic*
PC Press

INTRODUCTION

This compact instrument is intended to be a digital laboratory tool for hardware and, in some cases, software debugging. It contains four instruments in one unit: logic probe, single channel logic state analyzer, frequency counter and serial code receiver.

The only chip used is a PIC16F84A running at 10 MHz. The display unit is a LCD dot matrix alphanumeric module with 2 rows of 20 characters. The LCD is used as the display device for all functions, except for the logic probe which indicates low, high and pulse logic states on individual LEDs. Mode select, parameter change, function execute and ON/OFF switching is activated by two keys.

The probe tip is the common input for all functions, and the GND cable is used for connection to Vss of the tested circuit.

Although there are a lot of functions integrated in a single chip unit, it did not increase the complexity of hardware, as all functions are implemented in software. This enables a very good price/performance ratio.

The power supply is obtained by four 1.2V/180 mAh or 250 mAh NiCd batteries of LR03 (AAA) size. The instrument also has a battery manager, which supports automatic battery discharging and charging.

The source code is written in MPASM. As it is highly optimized for code space, most of the code could not be written in a modular format. For the same reason, a lot of subroutines have more than one entry point and some of them are terminated by a GOTO instruction instead of using a RETURN instruction.

FEATURES

- Stand-alone hand-held instrument
- Single chip design
- Built-in rechargeable power supply
- Easy to assemble and ready to use, no adjustment needed
- User interface with LCD output and command input by two keys
- TTL or 5V CMOS input, or direct input from RS-232C +/-12V signals

SPECIFIC FEATURES FOR INDIVIDUAL FUNCTIONS

Logic Probe

The low and high logic levels are displayed by LEDs, which are OFF if the probe tip is floating or connected to a hi-impedance (>220k) output. A pulse transition is detected and is indicated by turning on the LED for 80 ms.

Logic State Analyzer

The analyzer fetches 300 single bit samples at a selectable rate (in 16 steps from 40 Hz to 1 MHz). It has a programmable start at High-to-Low or Low-to-High transition at input. Digital waveforms are displayed in a pseudographic mode on the LCD.

Serial Code Receiver

The Serial Code Receiver receives 42 bytes and displays them in both HEX and ASCII. The baud rate is selectable in 8 steps, from 1200 to 115200. The selectable format is 7 or 8 bits with or without a parity bit which is not displayed. Signal polarity is also selectable. Direct signal stealing from an RS-232 or an RS-232C interface is possible.

Frequency Counter

The Frequency Counter counts frequency and displays it in an 8-digit decimal format on the LCD with a refresh rate of 500 ms. There are four ranges, from 5 to 40 MHz, which affect the count resolution (from 4 to 32).

Battery Manager

The Battery Manager provides for discharging with an automatic switch that changes to charge mode at 4V battery voltage, charging with 18 mA of constant current and automatic power off after 14 hours. Any DC source between 10V and 30V, at any polarity, can be used for charging.

SYSTEM FUNCTIONS

User Interface

There are four modes of operation: Analyzer, Serial Code Receiver, Frequency Counter and Battery Manager. The logic probe function is transparent in all modes except in the Frequency Counter.

In all modes, submodes are listed in the lower row of the LCD. The submodes list can be cycled through by pressing the right key, which moves the cursor (blinking block) to the right. The left key activates the selected submode (executes a function or changes the parameter state/value).

The right-most submode (right arrow symbol) acts as a shortcut jump to the next mode. After power-on (by pressing any key), a mode is chosen by pressing the left key, then the submode by the right key, and then the eventual parameter change or command execution by the left key again. The only exception is the Logic Probe function, the only action needed is to switch the instrument on, and the logic probe is ready to use.

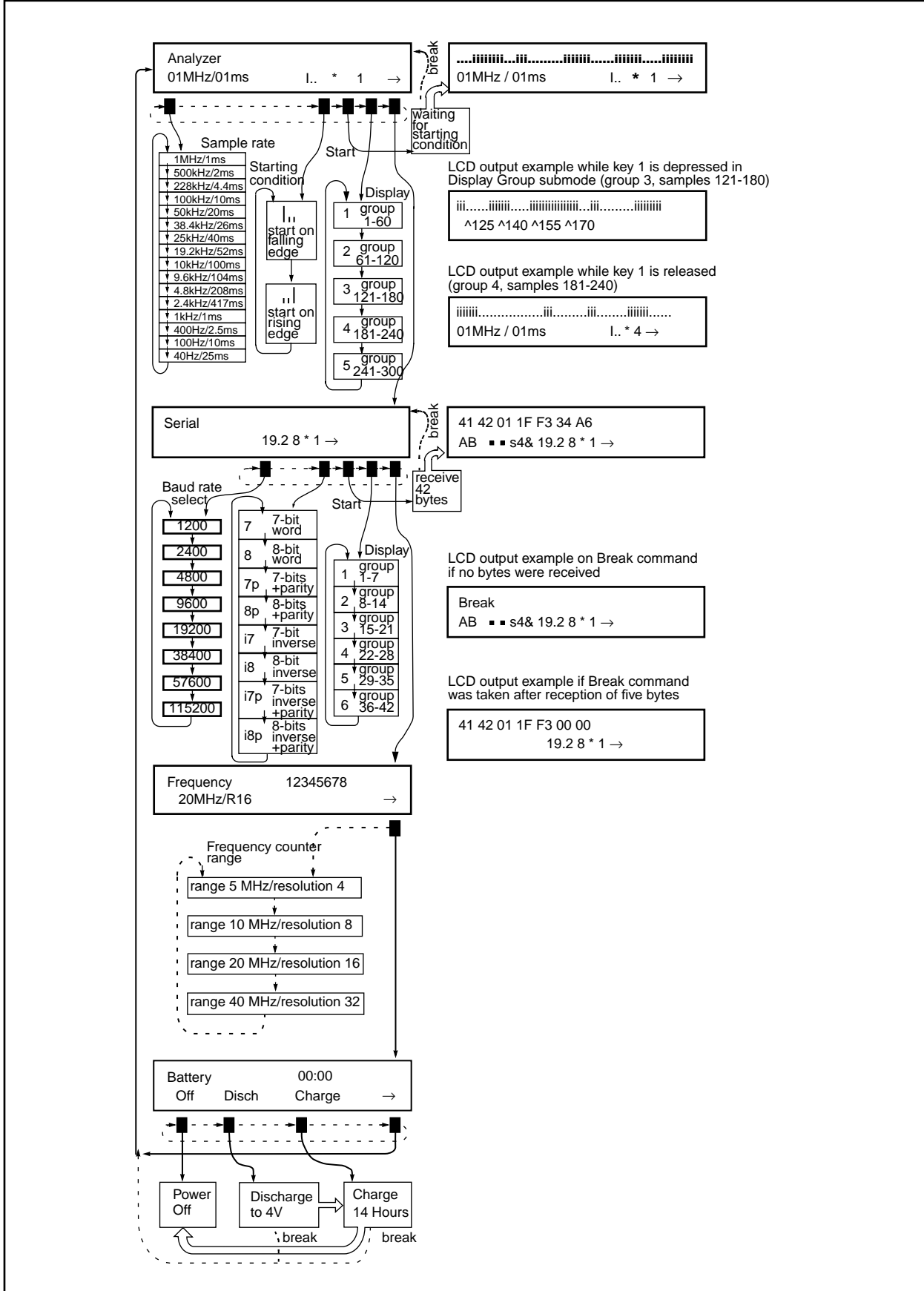
In Analyzer and Serial Code Receiver mode, the asterisk (*) is a special symbol for the "Start" command. When executed (left key pressed while the cursor is on asterisk), it causes the program to wait for a start condition or a start bit.

Although there is a manual Switch Off command (accessible in Battery mode), there is also the automatic power off after approximately 8 minutes of inactivity if no key is pressed. Note that the down counter for automatic power-off is "frozen", while the instrument is waiting for a start condition in analyzer mode and for the start bit in serial code receiver mode. Of course, the same applies to the discharging and charging processes, as another conditions are used to define the end of those processes.

Figure 1 represents the key functions diagram. The dotted line represents actions taken when the right key is pressed, and the solid line is for the left key. The cursor, which is the blinking block on the LCD, is represented as a solid block in Figure 1, but it is moved down on the drawing for clarity.

A variable (named REL) in the assembler source code, defines the position of the cursor on the LCD. If it is a '1' (default), the cursor will be placed on the first character of the command (or parameter). If it is a '0', the code will be assembled so that the cursor will be moved to the preceding location (if it exists) before the command.

FIGURE 1: HIGH LEVEL FUNCTION FLOWCHART



Logic Probe

The typical hardware solution for a logic probe is shown in Figure 2. Two inverters, for low and high indication, and two monostables, for pulse detection, are commonly used in most low-cost logic probes. This solution will display an unconnected probe tip as high logic

level. There are some better versions which can detect a floating input and turn all LEDs off if it is detected. Figure 3 represents the common solution for such functions, where two analog comparators are employed to detect the low, high and floating inputs.

FIGURE 2: TYPICAL LOGIC PROBE SCHEMATIC

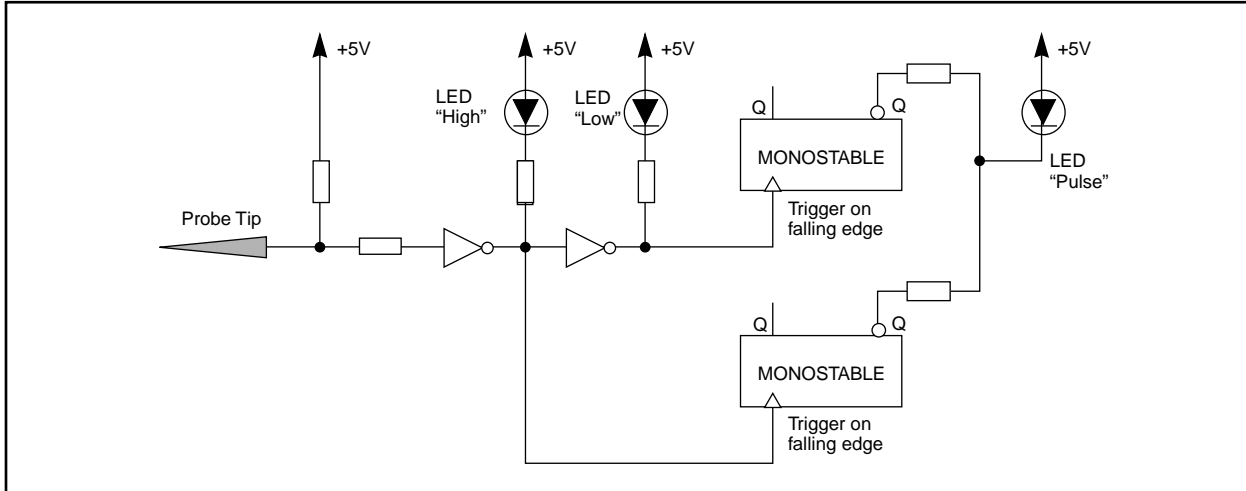
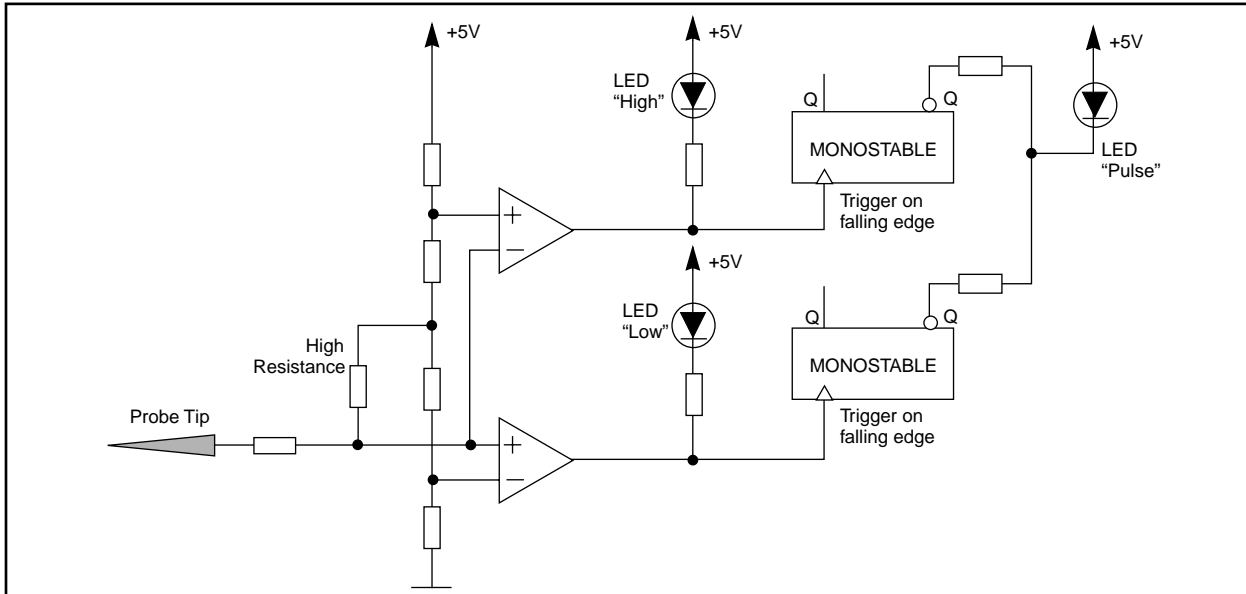


FIGURE 3: IMPROVED LOGIC PROBE SCHEMATIC



Instead of using such approaches, the logic probe function in this instrument is software aided, and the floating input is detected in a dynamic way instead of a static one. The equivalent hardware schematic diagram of this solution is shown in Figure 4 (Pulse detection circuit not shown). The hardware detail which supports the operation of the logic probe used in this unit is represented by Figure 5. The microcontroller polls the input tip and services LEDs L and H. If a transition is detected, LED P is switched ON and the down counter switches it OFF after 80 ms if no additional transition is detected.

This approach has two disadvantages. Logic state latching at a uniform rate may cause visible interference if the frequency of the monitored signal is near the latching rate. This problem is minimized by adding a self-variable extra delay in software, which makes the latching frequency unstable. This makes the range of critical frequencies much wider, but the interference appears as a very short burst of pauses in LED L or H activity, which is completely avoided by adding an extra debouncer of only 250 microseconds. Although unnoticeable, this delay helps prevent LED level instability while monitoring critical frequencies.

FIGURE 4: FUNCTIONAL SCHEMATIC OF PIC16F84A LOGIC PROBE

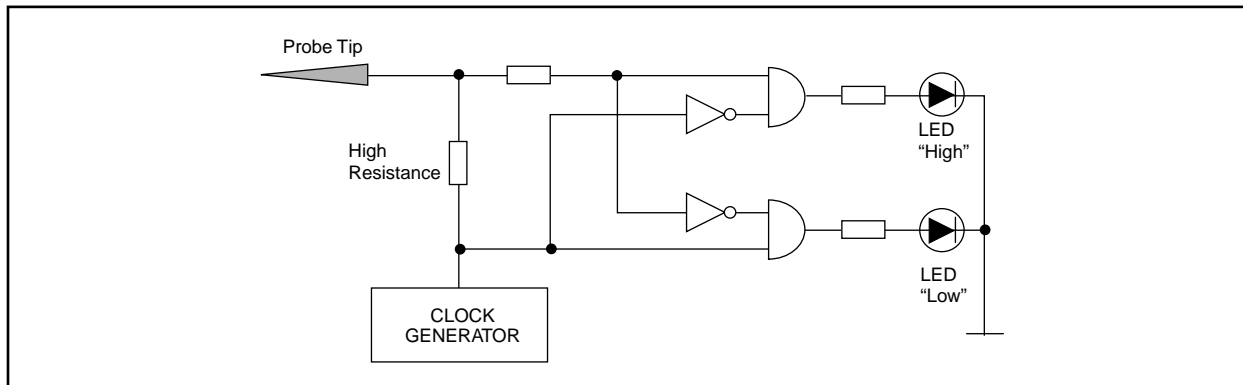
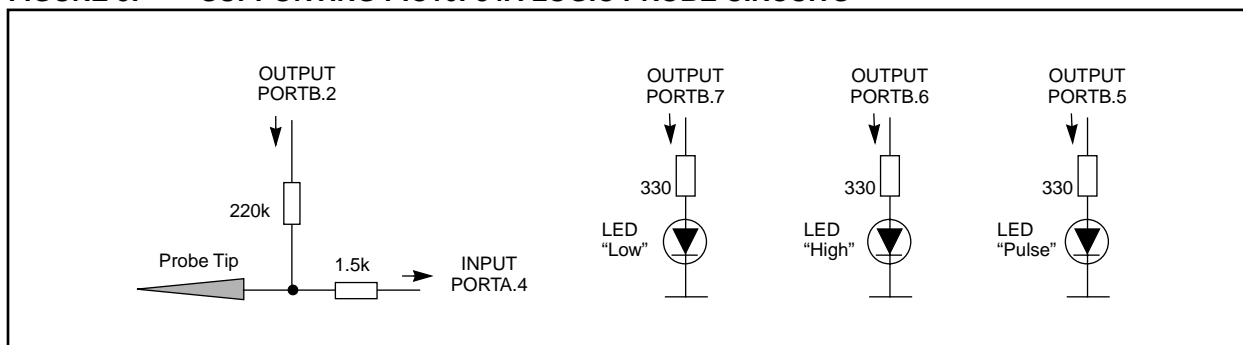


FIGURE 5: SUPPORTING PIC16F84A LOGIC PROBE CIRCUITS



Another disadvantage is related to pulse indication on LED P. In the case of a very short pulse, it is likely that the microcontroller, which polls the input, may omit it between two input reads. Instead of simple polling, the internal counter, `TMR0`, is used here so that instead of testing the logic state of the input, the state of `TMR0` is tested. In this way, pulses as short as 10 ns might be detected. In reality, the minimal pulse width is limited by resistor R6 and the input pin RA4 capacitance. The `T0SE` bit in the `OPTION_REG` register is properly updated at each pass, so that the first incoming transition will increment `TMR0`.

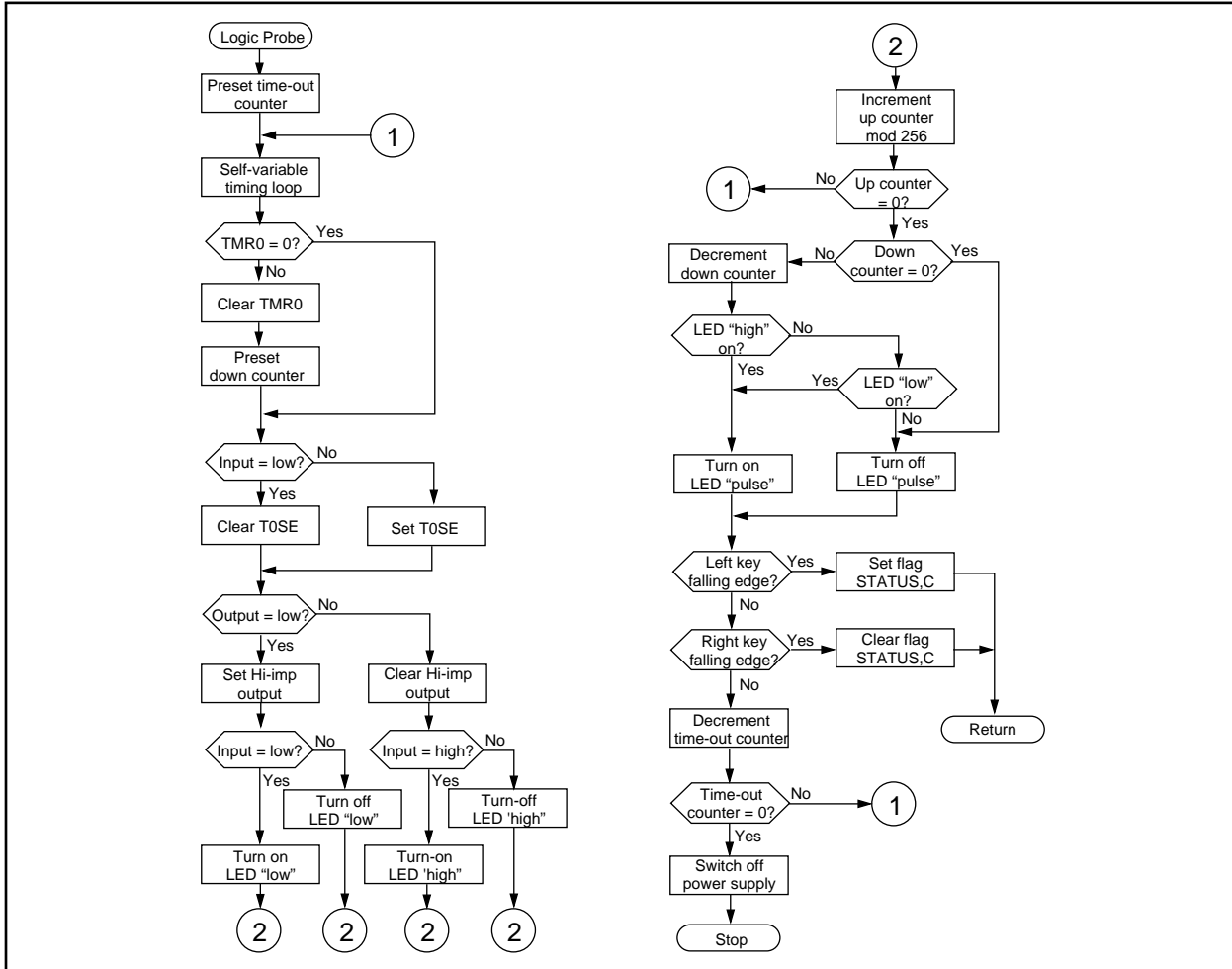
The logic probe software support is integrated in the keyboard routine. LEDs L, H and P are active only while the instrument is idle (doing nothing but waiting for some key to be pressed), which is all the time while the unit is ON, except in frequency counter mode, during battery discharging or charging, or if the START command is issued in analyzer or serial receiver mode and the job (300 samples fetched or all bytes received) is not yet finished.

Pin RB2 is the output which generates square-wave pulses. These pulses are fed through R5 to the probe tip. The resistance is high enough not to affect the tested circuit, except if the tested point is the floating input. However, in that case it will probably make the circuit unstable and thus help in locating the floating input. This pulse stream is also used by software to detect the floating probe tip, and in this case to switch

all LEDs off. This saves energy in batteries and helps to detect if the probe tip is validly connected to the point under test.

A simplified flow chart for the logic probe is represented by Figure 6. As this subroutine is an integral part of the key scan routine, the key (debouncers are not shown in detail) and time-out testing (which employs a 16-bit counter, "Time-out Counter") are also provided. "Up Counter" is the free running counter which enables execution of the second part of the subroutine to be performed at each 256th pass. "Down Counter" is the timing base for the LED Pulse. If the state of this counter is greater than zero and the LED Low or LED High is on, the LED Pulse will be turned on. The program exits only if some key is pressed (flag bit `STATUS,C` denotes which) or when the time-out counter reaches zero.

FIGURE 6: LOGIC PROBE FLOWCHART



Logic State Analyzer

The commonly used hardware concept for a logic analyzer design is represented in Figure 7. All those functions are realized in software, which is much easier to implement, but results in a loss of sampling speed. The software solution is briefly represented on the flow chart in Figure 8.

In analyzer mode, a sequence of 300 one-bit fetches is performed. Samples are stored in internal RAM (actually, 304 samples are read, but the last 4 are dummy reads). The upper row of the LCD is used to display the samples. As the LCD (Hitachi's LM032L) has no graphic capabilities (it is not possible to address a single dot), this is simulated by eight special user-defined characters (which are stored in the character generator RAM), each for a group of 3-bit samples. This enables a pseudo-graphic mode which, in this case, looks as if all pixels were individually addressed.

The display shows a window of 60 samples. One of five windows is selected by placing the cursor on the group number and advancing it by pressing the left key. While the key is pressed, the lower row displays the numeric

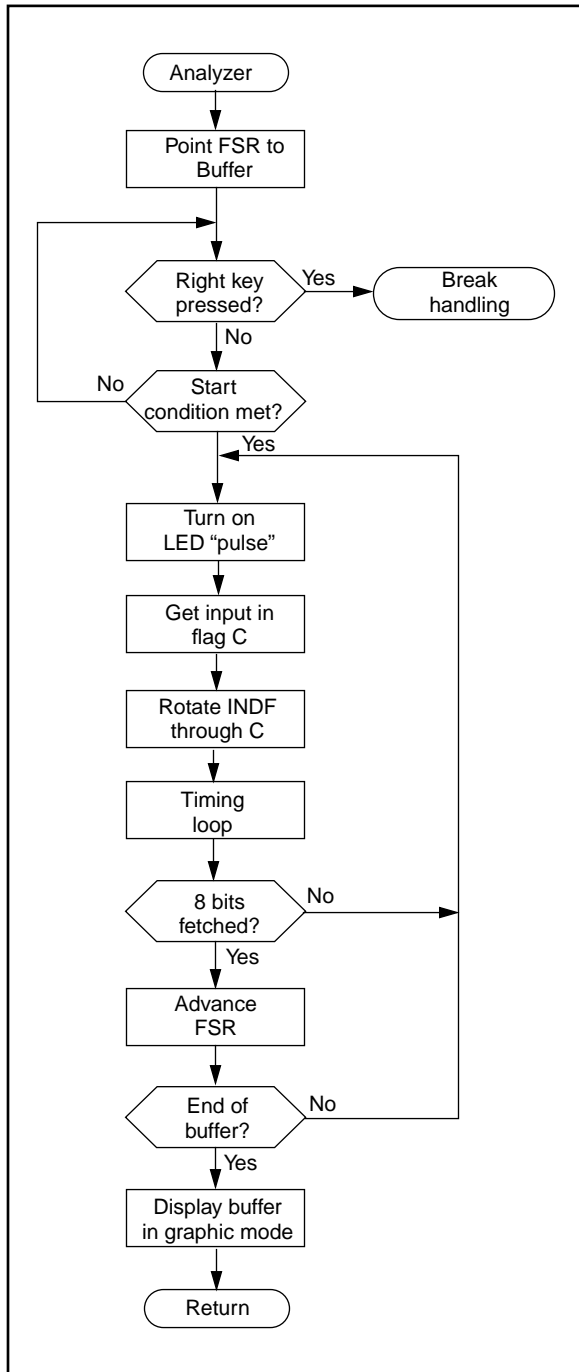
pointers, which help by counting the sample number and calculating the timings in the recorded sequence. When the key is released, the normal row 2 is restored.

A uniform clock, for sample rate, is internally generated. It is selectable to 16 steps. The frequency and period are both displayed. The following is a list of available sample rates:

1 MHz	50 kHz	10 kHz	1 kHz
500 kHz	38.4 kHz	9.6 kHz	400 Hz
228 kHz	25 kHz	4.8 kHz	100 Hz
100 kHz	19.2 kHz	2.4 kHz	40 Hz

The sampling sequence does not start immediately after the command is issued, but after the selected transition (L to H or H to L) is detected. While waiting for the transition to occur, the RB2 output is continuously held in the state which is opposite of the triggering logic level. This enables application on the wired-or logic, even if it is without pull-ups. If this condition never occurs, it is possible to escape by pressing the right key. In this case, message "Break" is displayed in the LCD upper row.

FIGURE 8: LOGIC ANALYZER FLOWCHART



All sample rates are generated by software, and the three highest ones use individual subroutines. The sample rate for 1 MHz, which is at the very beginning of the program, has to fetch and memorize a single bit sample by rotating it into the buffer, change the destination address after every 8 samples and exit the loop after 304 samples. All this while keeping uniform timing of 2 and 3 (alternated, which gives an average of 2.5) instruction cycles for one fetch. That could not be realized in a conventional manner, so it has a location-sensitive structure. Upon exit, it jumps to address 4Dh (which is far from the subroutine itself). If you modify anything in this program, take care not to affect this location.

The analyzer may have some unpredictable delays between an external starting event (rising or falling edge) and the first sample. In all cases, this delay may vary from 0 to 4 microseconds, so it may have some significance only in highest sample rates. One of the reasons for this delay is the time which the microcontroller requires for a key test, which enables the manual break if this event never comes. Also, there is some minor jitter at the 1 MHz analyzer sample rate. In the worst case, it might be 300 ns.

Serial Code Receiver

In this mode, a total of 42 bytes is received and displayed in both HEX and ASCII. The acceptable format is:

1 Start Bit / 7 or 8 Data Bits / 0 or 1 Parity Bit / 1 or more Stop Bits.

It is possible to connect the probe tip directly to the RS-232 or RS-232C +/- 12V voltage levels, to RS-422 or RS-485, or to +5V logic.

The available baud rates are:

1200	(1.2)
2400	(2.4)
4800	(4.8)
9600	(9.6)
19200	(19.2)
38400	(38.4)
57600	(57.6)
115200	(115)

7 or 8 data bits may be selected to adjust for the desired data format.

Parity or no parity bit (suffix "p"). This affects only the proper timing for this bit during reception. It is neither tested for validity nor displayed.

Standard RS232C or inverse polarity. If prefix "i" is displayed, then inverse polarity is active (low start bit, inverted data and optional parity bits and high stop bit). This is useful if the serial message must be fetched before the RS-232C TX drivers and after the RX buffers (which are both inverters).

Received bytes are displayed both in HEX and ASCII in 6 groups of 7 bytes each. ASCII representation is with bit 7 cleared, and the non-printable characters (00h-1Fh) are represented as dots. All other codes are standard ASCII.

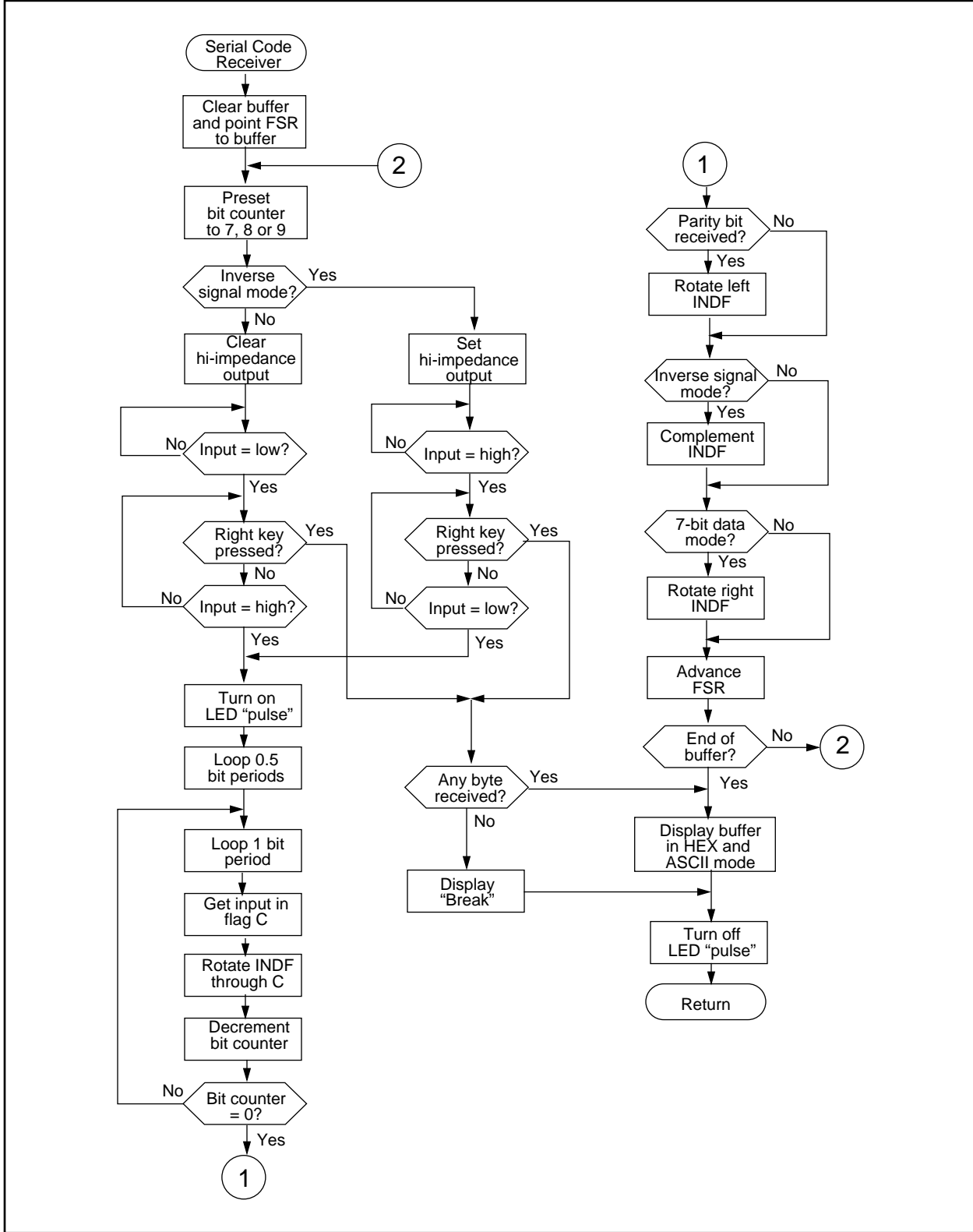
The string of received serial codes is 42 bytes long. If the string is shorter, the instrument will wait for next start bit, so it may look like it is stuck without any message. In that case, reception may be stopped by pressing the right key. If no bytes were received, the message "Break" will be displayed, but if at least one byte was received, the received sequence will be displayed with all unreceived bytes represented as zeros.

Similar to the analyzer mode, LED P will be turned ON when the first start bit is detected. This helps to detect sequences of less than 42 bytes in length.

No error test is performed during reception.

[Figure 9](#) represents the flowchart for the serial code receiver.

FIGURE 9: SERIAL CODE RECEIVER FLOWCHART



Frequency Counter

Figure 10 shows the standard structure of the hardware solution for a frequency counter. All this is substituted by software in the PICmicro[®] microcontroller (MCU) aided by the existing TMR0. All counters are binary, and the counter state is displayed after a 4-byte binary to 8-digit decimal conversion. The display refresh rate is 2 Hz.

The flow chart for the frequency counter is represented in Figure 11. As this is the real-time function, the existing keyboard subroutine might not be used, but separate key and time-out tests are written. The logic probe function is disabled in this mode.

There is no "Start" command here, as this function is active all the time while the instrument is in Frequency Counter mode. There is only one submode, range select, so pressing the right button is not used for stepping through submodes, but it changes the range immediately.

Internal counter TMR0 is used, and the program expands the width of the counter for an additional two bytes. The fourth byte is added after 500 ms of counting and multiplying the 24-bit counter state by a constant, which depends on which prescaler factor was used.

The prescaler also affects the counter resolution. Here are the counter ranges and corresponding resolutions:

- Range 5 MHz / Resolution 4
- Range 10 MHz / Resolution 8
- Range 20 MHz / Resolution 16
- Range 40 MHz / Resolution 32

The resolution surely affects the reading error of the frequency counter, but this error is still less than the error which is caused by the initial non-accuracy of industrial class quartz crystals.

FIGURE 10: FREQUENCY COUNTER SCHEMATIC

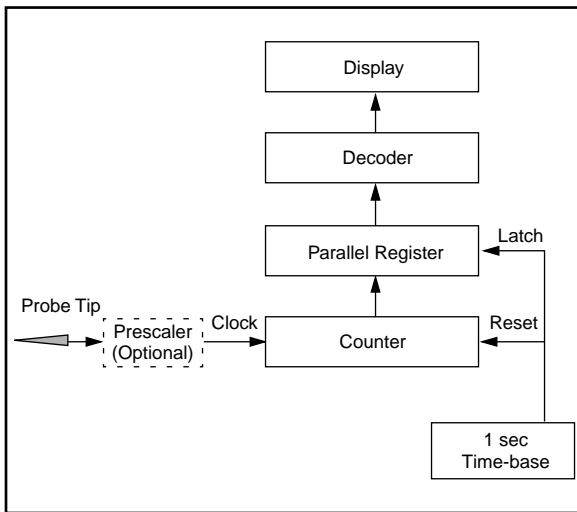
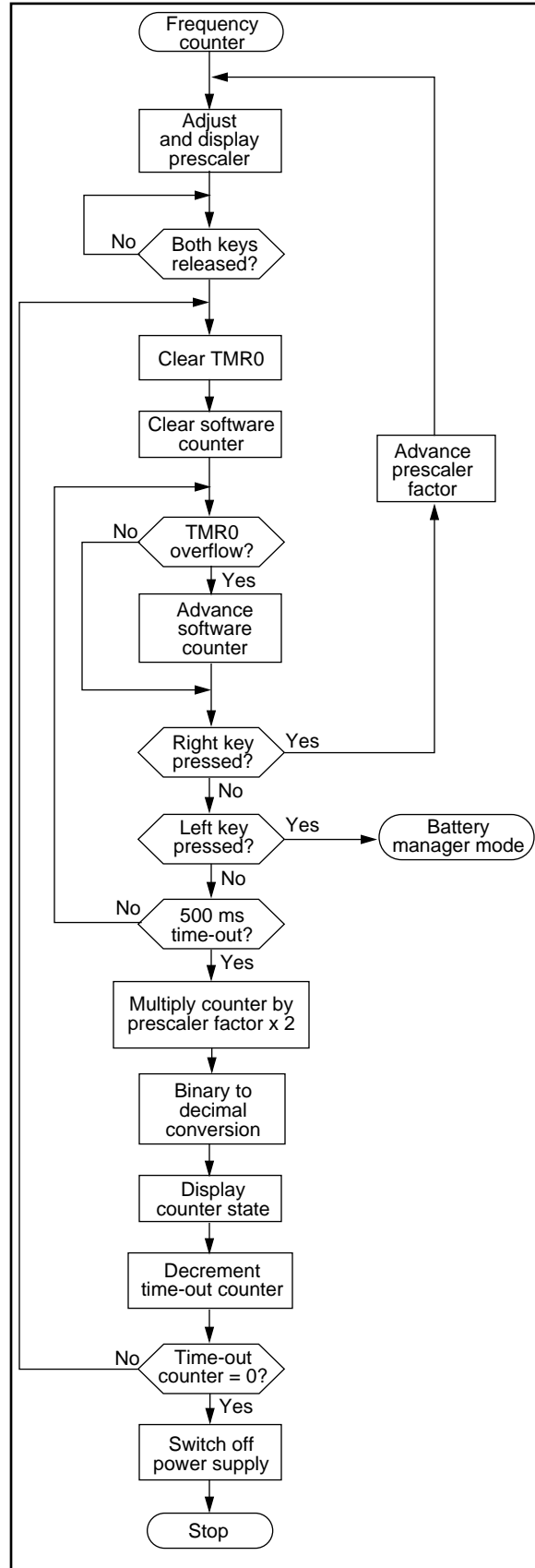


FIGURE 11: FREQUENCY COUNTER FLOWCHART



Battery Manager

The battery manager has three submodes. The first one is **manual power-off**, although there is also the automatic power off after approximately 8 minutes of inactivity (no key pressed).

The second submode is **discharging**. It is performed with 100 mA current through the resistors. The voltage monitor informs the PICmicro MCU if battery voltage is lower than 4V. If it is, the mode is automatically switched to charging. It is recommended that an external DC power supply be connected before the discharging command is issued. This will decrease the resulting discharging current to about 80 mA when the instrument is ON, and the DC supply is connected as the charging current flows independently of the mode selected.

Charge submode, when started, displays the time in HH:MM format, starting from 00:00, and switches off the instrument (and charging current also) at 14:00.

It is also possible to charge the NiCd battery even if it is not discharged, but this is not recommended, as unintentional overcharging may affect its capacity and life.

The unit is ready to charge the battery all the time if it is switched ON, even if the command charge is not active. It is enough to connect the external DC supply and to turn the instrument ON.

If the LCD were not counted, more than half of the hardware is used for discharging and charging. **Figure 12** explains the structure of the battery manager hardware in a simplified form, where transistors T1, T4 and T5 are replaced by switches, for clarity. The flow chart for the battery discharger and charger is shown in **Figure 13**.

FIGURE 12: BATTERY MANAGEMENT SCHEMATIC

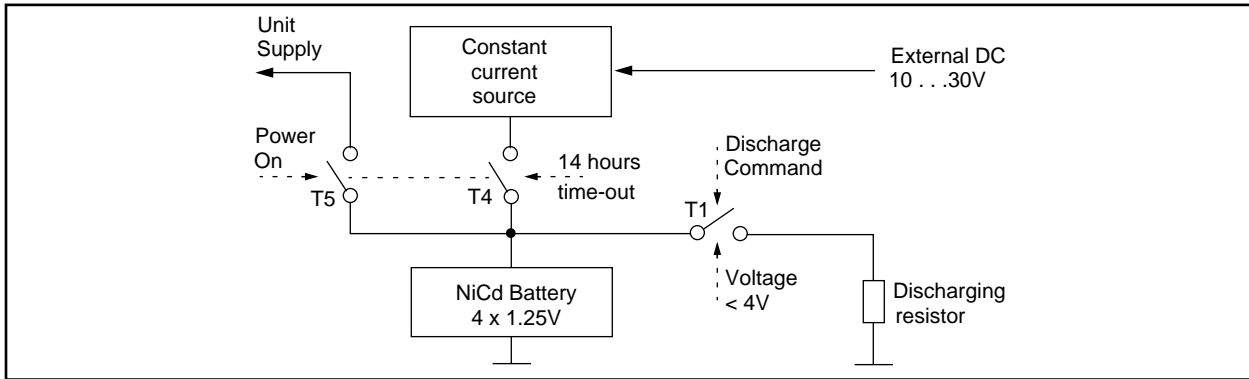
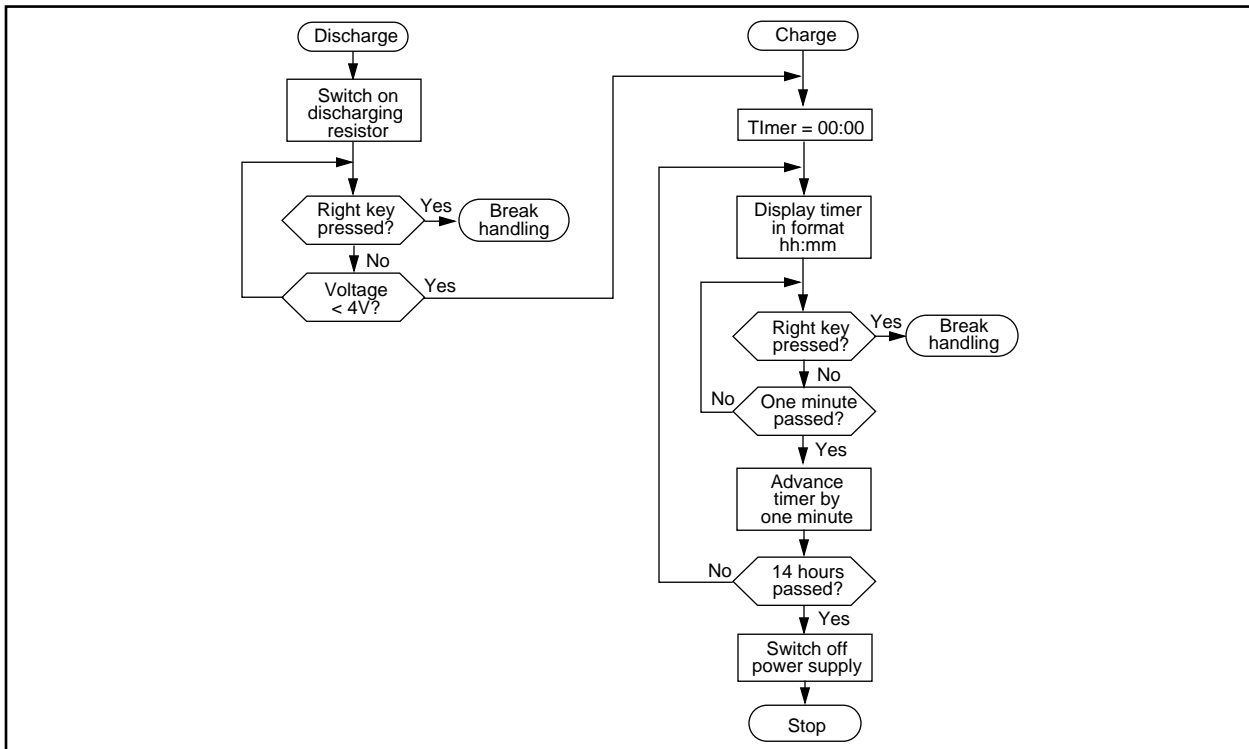


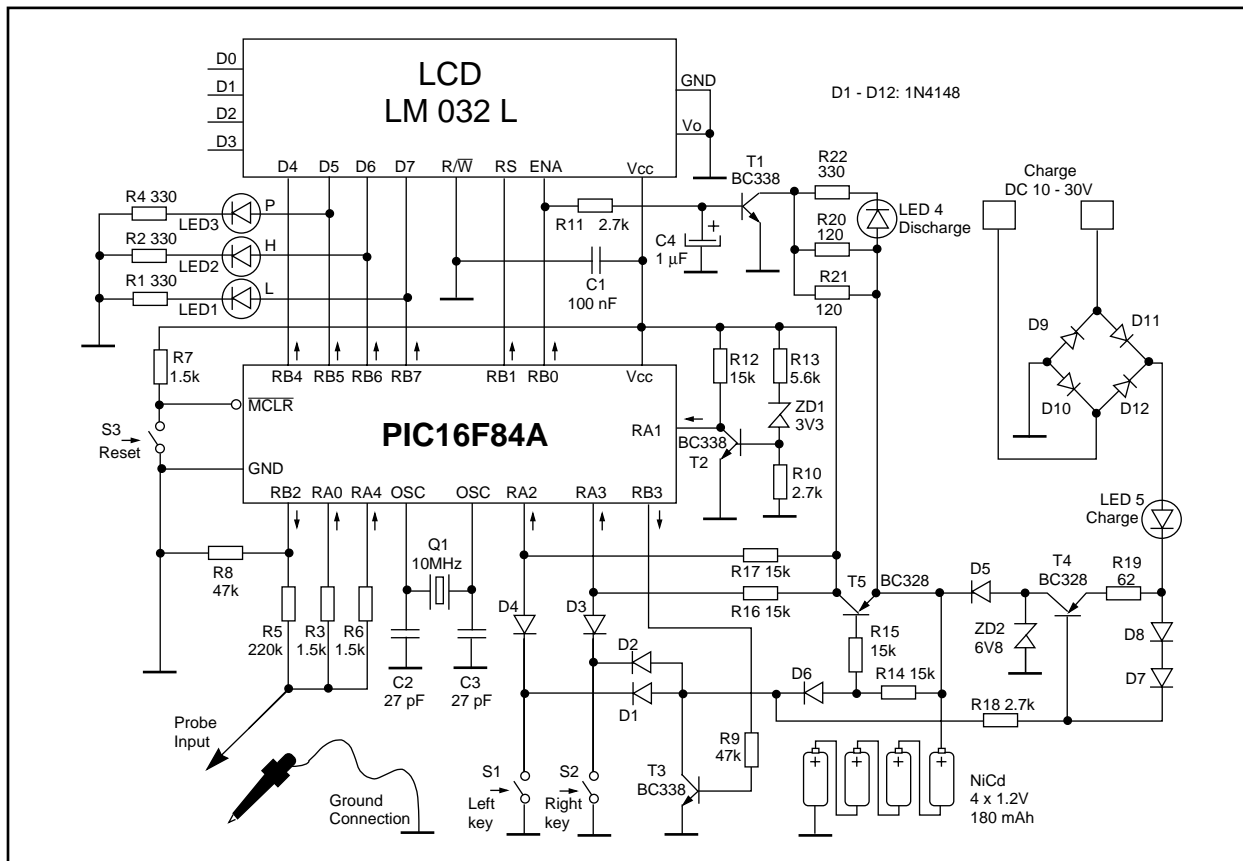
FIGURE 13: BATTERY MANAGER FLOWCHART



HARDWARE OVERVIEW

The complete schematic diagram is shown in Figure 14. The only IC used is the PIC16F84A-10/P, running at 10 MHz. It controls the intelligent LCD module via pins RB4-RB7, using two additional ports (RB0 and RB1) for the Enable signal and Register Select (Control/Data).

FIGURE 14: PIC16F84A PROBE SCHEMATIC



LCD module LM032L may be controlled in 4-bit parallel mode, which is used in this project. That is why pins 7-10 of the LCD are not used.

LEDs L, H and P share the same outputs with the LCD data bus. The only consequence is that some very minor and short LED flashing is visible while the PICmicro MCU is accessing the LCD (mostly when some key is pressed). **RB0** also has two functions. It controls the Enable signal for the LCD module and the transistor T1, which is used for the discharging function. To eliminate current spikes through T1 while accessing the LCD, capacitor C4 is added, which disables activating T1 by short Enable pulses. The main discharging resistors (R20 and R21) are intentionally located at two distant places on the PCB to minimize heat dissipation in a small area.

Inputs RA0 and RA4 are used to read the same signal, which was necessary because RA4 is the only port which may be directed to the `TMR0` counter, and RA0 had to be the "edge bit" in the port. The analyzer fetch

routine in the highest rate must rotate `PORTA` bits in a single instruction and perform bit transfers from the input pin to the `STATUS, C` flag.

Resistors R3 and R6 are used for signal voltage limitation, with internal reverse diodes in the PIC16F84A. This enables connecting the probe tip directly to the RS-232C connector, which has +/- 12V voltage range. Resistor R8 disables input floating while the unit is OFF.

The voltage monitor (transistor T2 with ZD1) acts as a comparator which holds input port RA1 low if the battery voltage is higher than 4V. This is used to automatically detect the end of the discharging process. The value of resistor R13 can be modified to fine trim this cut-off voltage to 4V.

Transistor T5 is the main ON-OFF switch. **RB3** controls this transistor through T3. It is also activated by pressing the left or right key. Diodes D3 and D4 disable false activation of T5 through internal diodes in the PICmicro MCU when it is powered OFF.

Transistor T4 is the constant current regulator, which enables the use of any DC supply between 10V and 30V for battery charging. **Diodes D9-D12** allow application of any voltage polarity. **Zener diode ZD2** is not the voltage stabilizer, but it protects the hardware from overvoltage if the battery contacts are not properly tied while charging.

S3 is the RESET key, which is mounted on the solder side of the PCB, and is accessible from the lower side of the instrument through the small hole at the bottom plane of the package. It is used if the MCU drops into a deadlock state for some reason or when the unit is switched ON for the first time after assembly.

Pin 3 (Vo) on the LCD connector is for LCD driving voltage. The manufacturer recommends the use of a potentiometer (10-20k) for voltage adjustment on this input and to fine trim the LCD contrast, but in all cases the contrast was optimal when the potentiometer was in its lower-most position (Vo shortened to GND). So it was rejected and pin 3 was connected to GND in the final version of this instrument.

Charging current will flow all the time while DC supply is connected and the unit is ON, even if the unit is not in Charge mode. When the unit is OFF (e.g. when the charging 14-hour process is finished), the charging current is stopped.

Both charging and discharging are indicated on individual LEDs.

Note: The LCD module used is Hitachi's LM032L. Type LM032LT may also be used, but it is not recommended, as it is a trans-reflective type and contains the integrated illumination (which may not be used in this case, as it requires high voltage). Do not use modules LM032H or LM032HT as they require a dual voltage supply (+5/-5V).

FIRST SWITCHING ON AFTER ASSEMBLING

The batteries should be connected last, as there is no easy way to disconnect them once they are soldered, also it is not recommended to assemble the hardware while the voltage is present. The best way is to test the instrument with some external 5V power supply, and when it is completely debugged and tested, the batteries may be soldered. Do not connect the external DC supply for charging batteries if the batteries are not safely in their places! Zener diode ZD2 will reduce the voltage to 6.8V, but avoid testing the efficiency of this protection if at all avoidable.

If the NiCd batteries are discharged to the point the PICmicro cannot operate, it will be necessary to keep the left or right key pressed for some time (while the DC supply is connected for charging), as the pressing of any key makes hardware bypass for charging current. After about one minute of such charging, the battery

voltage will be sufficient and then the unit will probably need to be reset by pressing S3. The normal charging process should then be used by executing the Charge command in Battery mode.

The contrast on the LCD is voltage-dependent, and as there is no voltage stabilizer, it appears to be a little darker immediately after a full charge. The battery voltage will be slightly over 5V. This will not affect readability. After a few minutes of operation, the battery voltage stabilizes and the LCD appears as normal.

Note: Data EEPROM is used for some lookup tables. This is read-only data and the Data EEPROM must be programmed before the unit is ready to use. The MCU will not affect data EEPROM contents. If your programmer does not support automatic loading of Data EEPROM contents from the HEX file, it must be loaded manually (a total of 61 bytes are used, and the last three bytes are don't care). The following will help in that case (all values are hexadecimal):

```

addr 00-07h:  88 01 01 98 F4 02 8C E4
addr 08-0Fh:  2C 88 64 0A 88 32 14 D8
addr 10-17h:  80 1A 88 19 28 C8 C0 34
addr 18-1Fh:  88 0A 64 C8 60 68 C8 30
addr 20-27h:  D0 C9 18 A1 80 01 01 14
addr 28-2Fh:  90 19 00 64 0A 00 28 19
addr 30-37h:  01 06 09 10 16 2E 30 64
addr 38-3Ch:  CC 05 0E 3B 95

```

MECHANICAL CONSTRUCTION

The components layout is shown in [Figure 16](#). All components are placed on the component side of the PCB, except key S3 (reset), which is on the solder side. So are the NiCd batteries, which are placed in the specially shaped PCB edges and soldered directly to the PCB.

The LCD module is placed on M3 spacers, 7 mm long, which are tightened to 5 mm long spacers at the bottom side of the PCB. This leaves enough room for batteries which are 10 mm in diameter. Key 3 should not be higher than 5 mm, and the recommended height for keys 1 and 2 is about 16 mm. As the keys listed in the parts list are 14.5 mm high, they should be mounted on an extra spacer about 1.5 mm thick, non-conductive material.

The probe tip is fixed using three wire loops soldered to the PCB and to the tip. If it is not possible to get a connector for the DC supply which fits to the PCB pads, it is also possible to cut the PCB (across the dotted line on the components layout) to make enough space for some other type of connector, which may be tightened to the package. Pads for wires, needed in this case, are provided on the PCB. The polarity is not significant.

It is possible to build the package of the same material which is used for printed circuit boards, as it can easily be cut and joints soldered. [Figure 15](#) shows the package detail.

FIGURE 15: CASE/PCB CONSTRUCTION

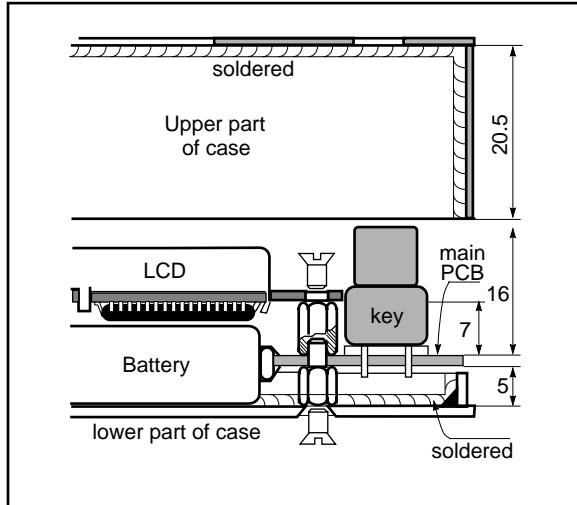
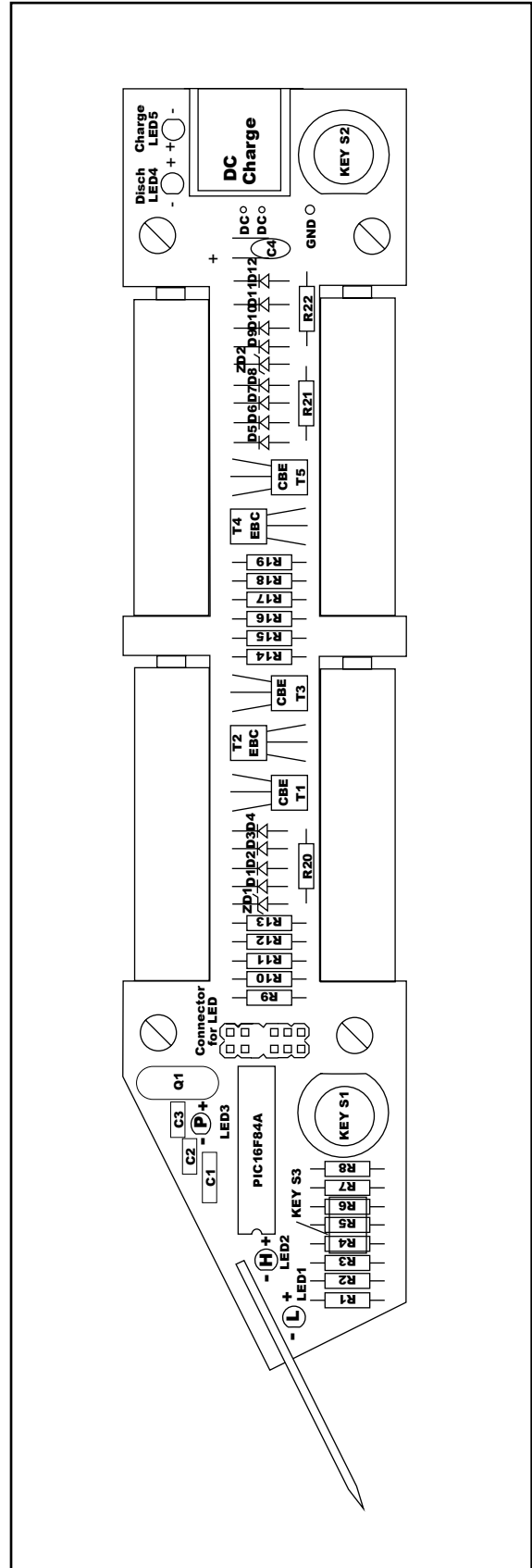


FIGURE 16: PCB PARTS LAYOUT



PARTS LIST

1.	LCD module type LM032L (Hitachi).....	1
2.	PCB.....	1
3.	Microcontroller PIC16F84A-10/P (Microchip).....	1
4.	Transistors:	
	BC338 (or any small signal silicon NPN in SOT-54, pinning CBE)3	BC328 (or any small signal silicon PNP in SOT-54, pinning CBE)2
5.	Diodes:	
	1N4148 (or any small signal silicon diode)12	ZPD 6V8 (or any low power 6.8V zener diode)1
	ZPD 3V3 (or any low power 3.3V zener diode)1	
6.	Resistors:	
	62R 1/4W axial1	5K6 1/4W axial1
	120R 1/4W axial 2	15K 1/4W axial5
	330R 1/4W axial 4	47K 1/4W axial2
	1K5 1/4W axial3	220K 1/4W axial1
	2K7 1/4W axial3	
7.	Capacitors:	
	27 pF ceramic2	1 uF tantal1
	100 nF ceramic1	
8.	Quartz:	
	10 MHz 1	
9.	LEDs:	
	red, 3 mm diameter 2	yellow, 3 mm diameter1
	green, 3 mm diameter2	
10.	I.C. socket:	
	18-pin 1	
11.	Keys:	
	typ ITT D 6 (raster 5*5 mm, 14.5 mm high) 2	typ SEL ET 5 (raster 5.5*3) or SEL ET 11 (raster 7.5*5)1
12.	Connectors:	
	2*7 pins male connector for PCB, raster 2.54 mm (100 mils)1	cable-end crocodil-grip for GND connection1
	2*7 pins female connector for PCB, raster 2.54 mm (100 mils)1	coaxial 3.5 mm female connector typ PG2031
13.	Mechanical parts:	
	spacer M3, 7 mm high 4	
	spacer M3, 5 mm high 4	

MPASM 02.15 Released PROBE.ASM 8-5-1998 13:51:00 PAGE 1

```

LOC  OBJECT CODE      LINE SOURCE TEXT
VALUE

00001 ;*****
00002 ;* Filename: PROBE.ASM
00003 ;*****
00004 ;* Author:   Voja Antonic
00005 ;* Company:  PC Press
00006 ;* Revision: RevA1
00007 ;* Date:    07-09-98
00008 ;* Assembled using MPASM rev 01.50
00009 ;*****
00010 ;* Include files:
00011 ;*         p16f84.inc
00012 ;*****
00013 ;* This is the program for multi-purpose laboratory instrument which
00014 ;* consists of logic probe, single-channel logic state analyzer,
00015 ;* serial code receiver and frequency counter. As this is single-chip
00016 ;* instrument, all functions are supported by software.
00017 ;* LCD module used is Hitachi's LM032L with 2 lines of 20 columns.
00018 ;*
00019 ;* Note: The code is optimized for code space, and for that reason the
00020 ;* most of code could not be written in modular format. For the same
00021 ;* reason a lot of subroutines have more than one entry point and some
00022 ;* of them are terminated by GOTO instead of RETURN.
00023 ;*
00024 ;* I/O port usage (all PORTA bits are inputs, all PORTB bits outputs)
00025 ;* (note: bits 0, 5, 6 and 7 in port B have two functions each):
00026 ;* PORTA,0 (input)  probe input
00027 ;* PORTA,1 (input)  voltage monitor (high if battery voltage < 4 V)
00028 ;* PORTA,2 (input)  left key (key 1) (low = key pressed)
00029 ;* PORTA,3 (input)  right key (key 2) (low = key pressed)
00030 ;* PORTA,4 (input)  probe input
00031 ;* PORTB,0 (output) enable LCD (high=select LCD),discharge (high=on)
00032 ;* PORTB,1 (output) register select LCD (low=instruction,high=data)
00033 ;* PORTB,2 (output) hi-imp output for probe pin
00034 ;* PORTB,3 (output) current hold for MCU supply (low = off)
00035 ;* PORTB,4 (output) LCD module D4
00036 ;* PORTB,5 (output) LCD module D5, LED P (high = on)
00037 ;* PORTB,6 (output) LCD module D6, LED H (high = on)
00038 ;* PORTB,7 (output) LCD module D7, LED L (high = on)
00039 ;*
00040 ;* External Clock Frequency: 10 MHz
00041 ;* Config Bit Settings: CP=OFF, PWRTE=ON, WDT=OFF, OSC=HS
00042 ;* Program Memory Usage: 1023 words
00043 ;* Data RAM Usage: 68 bytes
00044 ;* Data EEPROM Usage: 61 bytes
00045 ;* Note: This is read-only data, so the Data EEPROM must be programmed
00046 ;* before the unit is used. MCU will not affect data EEPROM contents.
00047 ;*****
00048         list p=16f84, f=inhx8m, n=0
00049         include "p16f84.inc"
00001         LIST
00002 ; P16F84.INC Standard Header File, V 2.00 Microchip Technology, Inc.
00136         LIST
00050
0000000C 00051 FLAG      equ      0ch      ; 1 by  flag register
0000000D 00052 RXBITS     equ      0dh      ; 1 by  bit0=parity,bit1=7/8 bits,bit 2=inverse
0000000E 00053 DJNZ      equ      0eh      ; 1 by  general purpose, e.g. loop counter
0000000F 00054 SCRATCH  equ      0fh      ; 1 by  general purpose scratchpad
00000010 00055 PCOUNT   equ      10h     ; 1 by  timing count for led P (monostable sim)
00000011 00056 SUBMODE  equ      11h     ; 1 by  submode (cursor horizontal position)
00000012 00057 DEBO1    equ      12h     ; 1 by  rotor for key 1 debouncing

```

```

00000013      00058 DEBO2  equ    13h    ; 1 by rotor for key 2 debouncing
00000014      00059 COUNT equ    14h    ; 1 by general purpose counter
00000015      00060 RATE  equ    15h    ; 1 by analyzer sample rate, 0...15
00000016      00061 CHARCOU equ   16h    ; 1 by char counter for fixed format display
00000017      00062 SHOWCOU equ   17h    ; 1 by 1-4, which group of 60 samples is shown
00000018      00063 DELAYL  equ   18h    ; 1 by delay for led P on when led L is on
00000019      00064 DELAYH  equ   19h    ; 1 by delay for led P on when led H is on
0000001A      00065 PRESC  equ    1ah    ; 1 by prescaler rate for frequency counter
0000001B      00066 TIMOUTL equ    1bh    ; 1 by timeout counter lo, for auto power off
0000001C      00067 TIMOUTH equ    1ch    ; 1 by timeout counter hi, for auto power off
0000001D      00068 RXRATE equ    1dh    ; 1 by rx baud rate, 0...7
0000001E      00069 BIN4  equ    1eh    ; 4 by arith buf bin value, lo byte first
00000022      00070 CMP4   equ    22h    ; 4 by arith buf for comparing, lo byte first
00000026      00071 BUFFER equ    26h    ;42 by 42 by receive buf for analyzer and RX
00000001      00072 REL    equ     1        ;=1 to put cursor on 1st char of command
00073          ;=0 to put cursor before the command
00074
00075 ; Bits definitions for FLAG register (bit 0 not used):
00000001      00076 DP    equ     1        ; decimal point in 3-digit bin2dec conv
00000002      00077 PTIP  equ     2        ; prev.state of probe input (for edge detect)
00000003      00078 RIPPLE equ     3        ; zero blanking bit
00000004      00079 XTOX  equ     4        ; analyzer start at: 1=rising, 0=falling edge
00000005      00080 LEDP  equ     5        ; led Pulse, 1=on
00000006      00081 LEDH  equ     6        ; led High, 1=on
00000007      00082 LEDL  equ     7        ; led Low, 1=on
00083
00084 ;*****
00085 ;* Reset vector
00086 ;*****
00087          goto    Start
00088
00089 ;*****
00090 ;* Get1MHz
00091 ;* This subroutine fetches 307 samples (last 7 will be ignored) from
00092 ;* PORTA.0 rotating through CARRY at 1 MHz rate - 2.5 instr. cycles
00093 ;* for each sample, realized mostly as 2 and 3 cycles alternatively,
00094 ;* at the following order:
00095
00096 ;* 4t-2t-2t (not in main loop, executed only once), and then
00097 ;* 2t-2t-3t-2t-3t-2t-3t-2t-3t-2t-3t-2t-3t-2t-4t (repeat 19 times)
00098 ;*
00099 ;* Call Common inits loop counter (COUNT) to make 19 cycles before
00100 ;* exiting (16 samples are fetched at each pass), and FSR to point to
00101 ;* BUFF. It also presets TOSE bit depending on XTOX bit (in FLAG reg)
00102 ;* to enable proper edge detect, as it will affect TMR0 state.
00103 ;* State of key 2 (Break) is tested while waiting for start condition.
00104 ;* Write ptr FSR is incremented after every 8 samples. COUNT initial
00105 ;* value is 01101101, after ANDing 0c0h and subtracting 33h from it,
00106 ;* makes 0dh, even if COUNT is incremented 18 times. After 19 passes,
00107 ;* COUNT is incremented to b'10000000', which after AND 0c0h and
00108 ;* SUB 33h makes 4dh. Those jumps are location sensitive, and it makes
00109 ;* the whole subroutine unrellocateable.
00110 ;* Between this subroutine and the instruction goto Finished (below),
00111 ;* which must be at loc. 4dh, there are 25 free locations. They are
00112 ;* used for tables DecTab and CurTab, which causes that those tables
00113 ;* must have the fixed length. If anything relocates here, take care
00114 ;* not to affect location of instruction goto Finished.
00115 ;* Input/Output variables: None
00116 ;*****
0001          org     1                ; this subroutine must start at addr 1
00118
0001          00119 Get1MHz                ; 2.5 t read cycle
0001 306D      00120          movlw    80h-.19    ; loop end in 19 cyc(38 by=304 smpls)
0002 226F      00121          call    Common    ; initialize COUNT, FSR, hi-imp out...
00122          ; ...bit XTOX and TOSE bit
0003          00123 GetEdge

```

```

0003 1D85      00124      btfss  PORTA,3      ; test status of key 2 and...
0004 28CC      00125      goto   Break        ; ...jump to Break routine if pressed
                   00126
0005 0801      00127      movf   TMRO,W       ; TMRO = logic level edge detector
0006 0C85      00128      rrf    PORTA,F      ; <--- ; the first sample is a little earlier
                   00129                        ; ... to compensate starting delay
0007 1903      00130      btfsc  STATUS,Z    ; test if there was egde...
0008 2803      00131      goto   GetEdge     ; ...and loop if not
                   00132
0009 0C80      00133      rrf    INDF,F       ; rotate bit into destination byte
000A 0C85      00134      rrf    PORTA,F      ; <--- get bit from input to C
000B 0C80      00135      rrf    INDF,F       ; rotate bit into destination byte
000C 0C85      00136      rrf    PORTA,F      ; <--- get bit from input to C
                   00137                        ; movwf PCL will jump here at 18 passes
                   00138                        ; ...@addr 0100 0000 (40h) - 33h = 0dh
000D 0C80      00139      rrf    INDF,F       ; rotate bit into destination byte
000E 0C85      00140      rrf    PORTA,F      ; <--- get bit from input to C
000F 0C80      00141      rrf    INDF,F       ; rotate bit into destination byte
0010 0C85      00142      rrf    PORTA,F      ; <--- get bit from input to C
0011 0C80      00143      rrf    INDF,F       ; rotate bit into destination byte
0012 0C85      00144      rrf    PORTA,F      ; <--- get bit from input to C
0013 0C80      00145      rrf    INDF,F       ; rotate bit into destination byte
                   00146
0014 0A94      00147      incf   COUNT,F     ; COUNT = loop counter
                   00148
0015 0C85      00149      rrf    PORTA,F      ; <--- get bit from input to C
0016 0C80      00150      rrf    INDF,F       ; rotate bit into destination byte
0017 0C85      00151      rrf    PORTA,F      ; <--- get bit from input to C
0018 0C80      00152      rrf    INDF,F       ; rotate bit into destination byte
                   00153
0019 0A84      00154      incf   FSR,F       ; must be exactly 8 read cycles apart
                   00155                        ; ... between FSR incrementing
001A 0C85      00156      rrf    PORTA,F      ; <--- get bit from input to C
001B 0C80      00157      rrf    INDF,F       ; rotate bit into destination byte
001C 0C85      00158      rrf    PORTA,F      ; <--- get bit from input to C
001D 0C80      00159      rrf    INDF,F       ; rotate bit into destination byte
                   00160
001E 0814      00161      movf   COUNT,W     ; COUNT = loop counter
                   00162
001F 0C85      00163      rrf    PORTA,F      ; <--- get bit from input to C
0020 0C80      00164      rrf    INDF,F       ; rotate bit into destination byte
0021 0C85      00165      rrf    PORTA,F      ; <--- get bit from input to C
0022 0C80      00166      rrf    INDF,F       ; rotate bit into destination byte
                   00167
0023 39C0      00168      andlw  0c0h        ; this will make first 18 jumps to 0dh,
                   00169                        ; ...and the 19th one to 4dh
0024 0C85      00170      rrf    PORTA,F      ; <--- get bit from input to C
0025 0C80      00171      rrf    INDF,F       ; rotate bit into destination byte
0026 0C85      00172      rrf    PORTA,F      ; <--- get bit from input to C
0027 0C80      00173      rrf    INDF,F       ; rotate bit into destination byte
                   00174
0028 3ECD      00175      addlw  -33h        ; this will make first 18 jumps to 0dh,
                   00176                        ; ...and the 19th one to 4dh
0029 0C85      00177      rrf    PORTA,F      ; <--- get bit from input to C
002A 0C80      00178      rrf    INDF,F       ; rotate bit into destination byte
002B 0C85      00179      rrf    PORTA,F      ; <--- get bit from input to C
002C 0C80      00180      rrf    INDF,F       ; rotate bit into destination byte
                   00181
002D 0A84      00182      incf   FSR,F       ; must be exactly 8 read cycles apart
                   00183                        ; ... between FSR incrementing
002E 0C85      00184      rrf    PORTA,F      ; <--- get bit from input to C
002F 0C80      00185      rrf    INDF,F       ; rotate bit into destination byte
0030 0C85      00186      rrf    PORTA,F      ; <--- get bit from input to C
0031 0C80      00187      rrf    INDF,F       ; rotate bit into destination byte
0032 0C85      00188      rrf    PORTA,F      ; <--- get bit from input to C
                   00189

```

```

0033 0082          00190      movwf   PCL          ; jumps to 0dh in first 18 passes
                   00191                      ; jumps to 4dh at 19th pass
                   00192
                   00193
;*****
                   00194 ;* This table is used for bin2ascii (4-byte to 8-digit) conversion
                   00195
;*****
0034          00196 DecTab
0034 3498          3496          3480          00197          dt          098h,096h,080h ;
decimal 10 000 000
0037 340F 3442 3440 00198      dt          00fh,042h,040h ; decimal 1 000 000
003A 3401 3486 34A0 00199      dt          001h,086h,0a0h ; decimal 100 000
003D 3400 3427 3410 00200      dt          000h,027h,010h ; decimal 10 000
0040 3400 3403 34E8 00201      dt          000h,003h,0e8h ; decimal 1 000
0043 3400 3400 3464 00202      dt          000h,000h,064h ; decimal 100
0046 3400 3400 340A 00203      dt          000h,000h,00ah ; decimal 10
                   00204
                   00205 ;*****
                   00206 ;* Cursor position table for all SUBMODEs in mode 4 (battery manager)
                   00207 ;*****
0049          00208 CurTab4
0049 34D3 34C0 34C4 00209      dt          0d2h+REL,0c0h,0c3h+REL,0cah+REL
                   34CB
                   00210
                   00211 ;*****
                   00212 ;* This is exit point for subroutine Get1MHz. Do not move this
                   00213 ;* instruction, it must be at address 4dh!
                   00214 ;*****
004D          00215      org          80h-33h          ; addr 1000 0000 (80h) - 33h = 4dh
                   00216                      ; Get1MHz jumps here
004D 2A6A          00217      goto       Finished
                   00218
                   00219 ;*****
                   00220 ;* Table for LCD module character generator redefinition, to enable
                   00221 ;* pseudographic representation of bit samples in analyzer mode. It
                   00222 ;* defines first 8 characters, which are stored in LCD module's RAM.
                   00223 ;*****
004E          00224 Graphs
                   00225 ;
004E 3400 3401 3404 00226      dt          00h, 01h, 04h, 05h, 10h, 11h, 14h, 15h
                   3405 3410 3411
                   3414 3415
                   00227
                   00228 ;*****
                   00229 ;* Cursor position table for all SUBMODEs in mode 1 (Analyzer)
                   00230 ;*****
0056          00231 CurTab1
0056 34D3 34C0 34CD0 00232      dt          0d2h+REL,0c0h,0cch+REL,0ceh+REL,0d0h+REL
                   34CF 34D1
                   00233
                   00234 ;*****
                   00235 ;* Cursor position table for all SUBMODEs in mode 2 (Serial rcvr)
                   00236 ;*****
005B          00237 CurTab2
005B 34D3 34C8 34CD 00238      dt          0d2h+REL,0c7h+REL,0cch+REL,0ceh+REL,0d0h+REL
                   34CF 34D1
                   00239
                   00240 ;*****
                   00241 ;* Prescaler table for resolution display in mode 3 (freq counter)
                   00242 ;*****
0060          00243 PrescTab
0060 3404 3408 3410 00244      dt          .4, .8, .16, .32
                   3420
                   00245
                   00246 ;*****

```

```

00247 ;* Range table for max. frequency display in mode 3 (freq counter)
00248 ;*****
0064 00249 RangeTab
0064 3405 340A 3414 00250 dt .5, .10, .20, .40
3428
00251 ;*****
00252 ;* Timing constants for serial code receiver
00253 ;*****
0068 00254 BaudRate
0068 34BC 345E 342E 00255 dt .188, .94, .46, .23, .11, .5, .3, .1
3417 340B 3405
3403 3401
00256
00257 ;*****
00258 ;* Text strings (terminator = last character with bit 7 set)
00259 ;*****
0070 00260 TxtHz
0070 344D 3448 347A 00261 dt "MHz", '/' + 80h
34AF
0074 00262 DisTxt
0074 344F 3466 3466 00263 dt "Off Disch. Chrg", 'e' + 80h
3420 3444 3469
3473 3463 3468
342E 3420 3443
3468 3461 3472
3467 34E5
0085 00264 Head1
0085 3441 346E 3461 00265 dt "Analyze", 'r' + 80h
346C 3479 347A
3465 34F2
008D 00266 Head2
008D 3453 3465 3472 00267 dt "Seria", 'l' + 80h
3469 3461 34EC
0093 00268 Head3
0093 3446 3472 3465 00269 dt "Frequenc", 'y' + 80h
3471 3475 3465
346E 3463 34F9
009C 00270 Head4
009C 3442 3461 3474 00271 dt "Batter", 'y' + 80h
3474 3465 3472
34F9
00A3 00272 BrkMes
00A3 3442 3472 3465 00273 dt "Brea", 'k' + 80h
3461 34EB
00274
00275 ;***** START
00276 ;*****
00277 ;* Power Up sequence: I/O port B defined as all outputs, PORTB, 3 set to
00278 ;* switch power supply on and the internal Data Ram is cleared
00279 ;*****
00A8 00280 Start
00A8 1683 00281 bsf STATUS, RP0
00A9 0186 00282 clrf TRISB ; portb: all bits outputs, port a: inputs
00AA 1283 00283 bcf STATUS, RP0
00AB 300C 00284 movlw 0ch ; start of RAM clr, & PORTB output byte
00AC 0086 00285 movwf PORTB ; switch power supply ON (set PORTB, 3)
00AD 23B8 00286 call ClrRam ; clear internal RAM and wait 33.8 ms
00287
00288 ;*****
00289 ;* LCD module initialization. 4-bit mode selected, display data RAM
00290 ;* cleared cursor set to blink mode, and the pseudographics character
00291 ;* for character set 00h-07h preset from table Graphs.
00292 ;*****
00AE 1086 00293 bcf PORTB, 1 ; rs lo (instruction)
00AF 3002 00294 movlw 2 ; 4-bit mode
00B0 23E9 00295 call Nibble ; write 4-bit mode command

```

```

00B1 3028          00296          movlw 28h          ; func set: 4 bit mode,2 lines,5*7 dots
00B2 23CF          00297          call  WrComL       ; write command and wait 130 us
00B3 3006          00298          movlw 06h         ; modeset: cursor moves right, no shift
00B4 23CF          00299          call  WrComL       ; write command and wait 130 us
00B5 300D          00300          movlw 0dh         ; disp on, no cursor,blink cursor pos
00B6 23CF          00301          call  WrComL       ; write command and wait 130 us
                   00302
00B7 3040          00303          movlw 40h         ; cg ram addr 0
00B8 23CF          00304          call  WrComL       ; write address and wait 130 us
00B9 304E          00305          movlw Graphs      ; start addr of graph set for lcd disp
00BA 008F          00306          movwf SCRATCH     ; move start address to pointer
00BB 3020          00307          movlw .8*.4       ; 8 special characters to define
                   00308          ; CHARCOU decremented in Char routine,
                   00309          ; ...that is why here is 8*4
00BC 0096          00310          movwf CHARCOU     ; loop counter for 8 special characters
00BD              00311          GoGraph
00BD 3005          00312          movlw .5          ; five rows are equal
00BE 0094          00313          movwf COUNT       ; counter for 5 rows
00BF              00314          FiveRows          ; inner loop: 5 rows are equal
00BF 23B5          00315          call  PclSub1     ; move SCRATCH to PCL
00C0 23D9          00316          call  CharNCC     ; rows 1-5 from the table
00C1 0B94          00317          decfsz COUNT,F    ; five passes over?
00C2 28BF          00318          goto  FiveRows    ; no, loop
                   00319
00C3 3015          00320          movlw 15h         ; 15h=b'10101'=dot-space-dot-space-dot
00C4 23D6          00321          call  CharBl      ; row 6:all dots set,row 7:all dots clr
00C5 23D7          00322          call  Blank       ; row 8: all dots cleared
00C6 0A8F          00323          incf  SCRATCH,F   ; inc ptr
00C7 0B96          00324          decfsz CHARCOU,F ; 8 characters defined?
00C8 28BD          00325          goto  GoGraph     ; no, loop 8 x
                   00326
00C9              00327          ;*****  USER INTERFACE
00C9              00328          ;*****
00C9              00329          ;* This is home point for mode 1(Analyzer): prints text"Analyzer" and
00C9              00330          ;* command line in line 2, w/cursor location set on variable SUBMODE.
00C9              00331          ;* Then the keyboard routine is called, where it waits for key to be
00C9              00332          ;* pressed.
00C9              00333          ;* Break entry point prints message Break in line 1 and redraws line 2
00C9              00334          ;*****
00C9 3084          00335          Model            ; Mode 1: Analyzer
00CA 235F          00336          movlw Head1-1    ; start address of string -1
00CB 28CD          00337          call  Headline    ; print "Analyzer"
00CC 23DB          00338          goto  Farm1       ; avoid "Break" message
00CC 2170          00339          Break           ; Break entry pt (if Break in Mode 1)
00CD 227D          00340          call  PrintBrk   ; print "Break"
00CE 2170          00341          Farm1           ;
00CE 2170          00342          call  PrintM1    ; print string in line 2
00CE 3056          00343          Farm1B          ;
00CF 2170          00344          movlw CurTab1    ; get cursor table addr in analyze mode
00CF 2170          00345          call  CurPosKb   ; place cursor on proper position
00CF 2170          00346          ; test keys / probe input,service leds
00CF 2170          00347          ; return if key press (C:key1,NC:key2)
00CF 2170          00348
00CF 2170          00349          ;*****
00D0 1803          00350          ;* If key 1 pressed (C),SUBMODE is advanced (range 0...4,then wrapt to 0
00D1 28D4          00351          ;* If key 2 pressed (NC), program vectors to corresponding routine
00D1 28D4          00352          ;*(except if SUBMODE=1,then the sample rate is advanced and displayed)
00D1 28D4          00353          ;*****
00D1 28D4          00354          btfsz STATUS,C   ; test which key was pressed
00D1 28D4          00355          goto  Key1A      ; jump if key 1
00D1 28D4          00356          ; continue if key 2 pressed
00D2 212A          00357          call  Range5     ; advance var SUBMODE in range 0...4
00D3 28CE          00358          goto  Farm1B     ; go wait next key
00D4              00359          Key1A           ; key 1 pressed
00D4 0B11          00360          decfsz SUBMODE,W ; test SUBMODE (set Z if SUBMODE=1)
00D5 28DF          00361          goto  NoRate     ; jump if SUBMODE <>1

```

```

00362                ; continue if SUBMODE=1
00D6 0A95           00363      incf   RATE,F      ; advance sample rate
00D7 1215           00364      bcf    RATE,4      ; RATE range 0...15
                   00365
00D8 22B7           00366      call   ClrRow1     ; prepare line 1 to print sample rate #
00D9 0A15           00367      incf   RATE,W      ; readjust RATE from 0...15 to 1...16
00DA 2370           00368      call   Print255    ; print serial # of sample rate 1...16
00DB 28CD           00369      goto   Farm1       ; go redraw row 2, wait next command
                   00370
00DC                00371 EdgeSet      ; Change start cond(L-2-H or H-2-L)
00DC 3010           00372      movlw  10h         ; bit 4 is flag XTOX
00DD 068C           00373      xorwf  FLAG,F      ; change flag
00DE 28CD           00374      goto   Farm1       ; go redraw row 2, wait next command
00DF                00375 NoRate
00DF 1811           00376      btfsc  SUBMODE,0   ; bit 0 will be set only if SUBMODE=3
00E0 2A0E           00377      goto   ModelGo     ; if SUBMODE=3
00E1 1891           00378      btfsc  SUBMODE,1   ; bit 1 will be set only if SUBMODE=2
00E2 28DC           00379      goto   EdgeSet     ; if SUBMODE=2
00E3 1911           00380      btfsc  SUBMODE,2   ; bit 2 will be set only if SUBMODE=4
00E4 29D4           00381      goto   ModelShow   ; if SUBMODE=4
                   00382                ; if SUBMODE=0, program drops to Mode2.
                   00383
00384 ;*****
00385 ;* This is home for mode 2 (RS232 receiver): prints text "Serial" and
00386 ;* command line in line 2,cursor placed depended on variable SUBMODE.
00387 ;* BrkRS entry point prints message Break in line 1 and redraws line 2
00388 ;* if 0 bytes are received, display first 7 bytes if any byte received
00389 ;*****
00E5                00390 Mode2        ; mode 2: Serial receiver
00E5 308C           00391      movlw  Head2-1     ; start address of string -1
00E6 235F           00392      call   Headline    ; print "Serial"
00E7 28ED           00393      goto   Farm2       ; avoid "Break" message
00E8                00394 BrkRS        ; Break entry pt (if Break in mode 2)
00E8 0804           00395      movf   FSR,W       ; FSR points to write next rcvd byte
00E9 3A26           00396      xorlw  BUFFER      ; if FSR=literal BUFF the 0 bytes rcvd
00EA 1D03           00397      btfss  STATUS,Z    ; test if FSR = literal BUFFER
00EB 2AC6           00398      goto   Show2       ; no -some bytes received,show them
00EC 23DB           00399      call   PrintBrk    ; yes -no bytes received, print "Break"
                   00400
00401 ;*****
00402 ;* Prints baud rate in KBaud on LCD
00403 ;*
00404 ;* Input variables: RXRATE in range 0...7
00405 ;* Output variables: CHARCOU decremented by num of characters printed
00406 ;*****
00ED                00407 Farm2
00ED 30C8           00408      movlw  0c8h        ; baud rate position on LCD
00EE 23CF           00409      call   WrComL      ; move cursor command
                   00410
00EF 108C           00411      bcf    FLAG,DP     ; no decimal point printing if RATE=0
00F0 019F           00412      clrf  BIN4+1       ; BIN4+1 is high byte for baudrate disp
                   00413
00F1 0A1D           00414      incf  RXRATE,W     ; move RXRATE from range 0...7 to 1...8
00F2 008E           00415      movwf  DJNZ        ; DJNZ = RXRATE+1
                   00416
00F3 3073           00417      movlw  .115        ; case RXRATE=7:then Baudrate=115 Kbaud
00F4 198E           00418      btfsc  DJNZ,3      ; test if DJNZ=8 (same as RXRATE=7)
00F5 2905           00419      goto   Lth256     ; yes, go case 115.2 (RXRATE=7)
                   00420
00F6 148C           00421      bsf    FLAG,DP     ; for rete 0...6 there is decimal point
00F7 149F           00422      bsf    BIN4+1,1    ; case RXRATE=6:is hi byte for 57.6
00F8 3040           00423      movlw  .576-.512   ; case RXRATE=6:is lo byte for 57.6
00F9 0A8E           00424      incf  DJNZ,F       ; DJNZ=RXRATE+2
00FA 198E           00425      btfsc  DJNZ,3      ; test if DJNZ=8 (same as RXRATE=6)
00FB 2905           00426      goto   Lth256     ; yes, go case 57.6 (RXRATE=6)
                   00427
00427

```

```

00FC 019F      00428      clrf    BIN4+1      ; for rates 0..5 hi byte is zero
00FD 3003      00429      movlw   .3          ; constant for rates 0...5
00FE 009E      00430      movwf   BIN4        ; BIN4 will be rotated (mult by 2)
                                00431      ; RXRATE+2 times to get 1.2 - 2.4 - 4.8
                                00432      ; - 9.6 - 19.2 - 38.4
00FF          00433      X2Loop
00FF 1003      00434      bcf     STATUS,C    ; clear bit C to get multiplying by 2
0100 0D9E      00435      rlf     BIN4,F      ; multiply low byte
0101 0D9F      00436      rlf     BIN4+1,F    ; multiply hi byte, that is 16-bit rotate
0102 0B8E      00437      decfsz DJNZ,F      ; test if RXRATE+2 times multiplied
0103 28FF      00438      goto   X2Loop      ; no, loop
0104 081E      00439      movf   BIN4,W      ; yes, get result to print it
0105          00440      Lth256
0105 2372      00441      call   PrintBR     ; print baud rate,incl. decimal point
0106 23D7      00442      call   Blank       ; to delete last # from previous rate
                                00443
00444 ;*****
00445 ;* Prints num bits to be received (7 or 8), with suffix "p" if parity
00446 ;* bit will be received (not written to RAM!), and with prefix "i" if
00447 ;* inverse input polarity is expected. Input variable RXBITS, bit0 set
00448 ;* if parity bit expected, bit 1 set if 8-bit word and bit 2 set if
00449 ;* inverse polarity (lo start bit,inverse data bits and high stop bit)
00450 ;*****
0107 30CC      00451      movlw  0cch        ; bit# pos (7/8/7p/8p/i7/i8/i7p/i8p) -1
0108 23CF      00452      call   WrComL     ; write command
                                00453
0109 3020      00454      movlw  ' '         ; space: true polarity
010A 190D      00455      btfsz  RXBITS,2   ; let it be space if RXBITS,2 cleared
010B 3069      00456      movlw  'i'        ; "i": inverse polarity
010C 23D8      00457      call   Char       ; print blank or "i"
                                00458
010D 3037      00459      movlw  '7'        ; "7": 7 bits
010E 1C8D      00460      btfsz  RXBITS,1   ; let it be 7 if RXBITS,1 set
010F 3038      00461      movlw  '8'        ; "8": 8 bits
0110 23D8      00462      call   Char       ; print "7" or "8" (bits)
                                00463
0111 3020      00464      movlw  ' '         ; space: no parity
0112 180D      00465      btfsz  RXBITS,0   ; let it be space if RXBITS,0 cleared
0113 3070      00466      movlw  'p'        ; "p": parity bit exists
0114 23D8      00467      call   Char       ; print "p" (parity bit) or blank
                                00468
00469 ;*****
00470 ;* This call prints number of group displayed and "*" (execution) symb
00471 ;*****
0115 22A5      00472      call   KaoAna     ; print rest of line - is the same as
                                00473      ; on mode 1 (analyzer)
                                00474
00475 ;*****
00476 ;* Places cursor on proper position (input variable SUBMODE) and calls
00477 ;* keyboard subroutine, where it will wait for key to be pressed
00478 ;*****
0116 305B      00479      movlw  CurTab2    ; table with cursor positions
0117 2170      00480      call   CurPosKb   ; place cursor on proper pos
                                00481      ; test keys / probe input,service leds
                                00482      ; return if key press (C:key1,NC:key2)
                                00483
00484 ;*****
00485 ;* If key1 pressed (C), SUBMODE is advanced (range 0..4, then wrap to 0
00486 ;* If key 2 pressed (NC), program vectors to corresponding routine
00487 ;* (except if SUBMODE=1, then the Baud rate is advanced and displayed)
00488 ;*****
0118 1803      00489      btfsz  STATUS,C    ; test which key was pressed
0119 291C      00490      goto   Key1B      ; jump if C set, means key 1 pressed
                                00491      ; key 2 pressed
011A 212A      00492      call   Range5     ; increment SUBMODE in range 0...4
011B 28ED      00493      goto   Farm2      ; go redraw row2, wait for next command

```

```

011C          00494      Key1B          ; key 1 pressed
011C 1911     00495      btfscc SUBMODE,2  ; bit 2 is set only if SUBMODE = 4
011D 2AC1     00496      goto     Mode2Show ; jump if SUBMODE = 4
              00497
011E 1C91     00498      btfss   SUBMODE,1 ; bit1 cleared only if SUBMODE = 0 or 1
011F 2925     00499      goto     Sub01     ; jump if SUBMODE = 0 or SUBMODE = 1
              00500
0120 1811     00501      btfscc SUBMODE,0  ; bit 0 is set here only if SUBMODE = 3
0121 2AEA     00502      goto     Mode2Go   ; jump if SUBMODE = 3
              00503
0122 0A8D     00504      incf    RXBITS,F   ; advance RXBITS (command)
0123 118D     00505      bcf     RXBITS,3   ; RXBITS cycle in range = 0..7
0124 28ED     00506      goto     Farm2     ; go redraw row2, wait for next command
0125          00507      Sub01
0125 1C11     00508      btfss   SUBMODE,0  ; bit 0 is set here only if SUBMODE = 0
0126 2B28     00509      goto     FreqEp    ; if SUBMODE =0,goto frequency entry pt
              00510
0127 0A9D     00511      incf    RXRATE,F   ; if SUBMODE = 1 then advance RXRATE
0128 119D     00512      bcf     RXRATE,3   ; RXRATE cycle in range 0..7
0129 28ED     00513      goto     Farm2     ; go redraw row2, wait for next command
              00514
00515 ;*****
00516 ;* This subroutine increments variable SUBMODE,and if the result is >4
00517 ;* it wraps to 0
00518 ;*****
012A          00519      Range5          ; increment SUBMODE in range 0..4
012A 0A91     00520      incf     SUBMODE,F ; advance SUBMODE
012B 1911     00521      btfscc SUBMODE,2  ; if SUBMODE,2 cleared then no overflow
012C 1C11     00522      btfscc SUBMODE,0  ; if SUBMODE,0 cleared then no overflow
012D 0008     00523      return          ; no overflow: return
012E 0191     00524      clrf     SUBMODE  ; SUBMODE cycle in range 0..4
012F 0008     00525      return          ; SUBMODE wrapped to 0, return
              00526
00527 ;*****
00528 ;* Mode4 is home point for mode 4 (off/discharge/charge): prints text
00529 ;* "Battery" and command line in line2, with cursor placed depended on
00530 ;* variable SUBMODE. Then keyboard routine is called, where it will
00531 ;* wait for key to be pressed
00532 ;* Break4 entry point prints message Break in line1 and redraws line2
00533 ;* ExitDis is the entry point if key 2 is pressed during discharging
00534 ;*****
0130          00535      ExitDis         ; exit disch entry point,if disch Break
0130 23BD     00536      call     DisEna30 ; turn off PORTB,0 discharge transistor
0131          00537      Break4         ; exit Chg entry point, if Charge Break
0131 23DB     00538      call     PrintBrk ; print "Break"
0132 2934     00539      goto     Contm4  ; avoid headline printing
0133          00540      Mode4         ; mode 4: discharge/charge
0133 2360     00541      call     Headline2 ; print "Battery"
0134          00542      Contm4
0134 23CE     00543      call     Row2    ; move cursor to line 2
0135 3073     00544      movlw   DisTxt-1 ; point to message -1
0136 23DD     00545      call     Write   ; print Off Disch Charge
0137          00546      Farm4
0137 3049     00547      movlw   CurTab4  ; point to cursor table for mode 4
0138 2170     00548      call     CurPosKb ; place cursor @ Off/Disch/Charge/-->
              00549
              00550      ; test keys / probe input,service leds
              00551      ; return if key press (C:key1,NC:key2)
00552 ;*****
00553 ;* If key1 pressed (C), SUBMODE is advanced (range 0..3,then wrap to 0
00554 ;* If key 2 pressed (NC), program jumps to corresponding routine
00555 ;*****
0139 1803     00556      btfscc STATUS,C   ; test which key was pressed
013A 293E     00557      goto     Key1D    ; C set: key 1 pressed
              00558      ; key 2 pressed
013B 0A91     00559      incf     SUBMODE,F ; advance SUBMODE

```

```

013C 1111      00560      bcf      SUBMODE,2 ; SUBMODE cycle in range 0...3
013D 2937      00561      goto     Farm4     ; go redraw row2, wait for next command
013E           00562      Key1D          ; key 1 pressed
013E 1891      00563      btfs    SUBMODE,1 ; bit 1 set only if Chg/Disch. submode
013F 2943      00564      goto     ChargDis  ; goto Charge or Discharge process
                        00565                        ; depended on bit 0 in variable SUBMODE
0140 1C11      00566      btfss   SUBMODE,0 ; SUBMODE,0 cleared here if SUBMODE=0
0141 28C9      00567      goto     Mod1      ; shortcut to mode 1 (Analyzer)
                        00568
0142 29BC      00569      goto     Suicide   ; manual power off
                        00570
00571 ;***** CHARGE/DISCHARGE
00572 ;*****
00573 ;* Charge/Disch entry point. If bit SUBMODE,0 set, then go to Charge
00574 ;* SUBMODE,1 is set in this point. (SUBMODE=2 or 3)
00575 ;*****
0143           00576      ChargDis
0143 22B7      00577      call     ClrRow1   ; in both cases row 1 must be cleared
0144 1811      00578      btfs    SUBMODE,0 ; test if SUBMODE,0 set, if so...
0145 2952      00579      goto     Charge    ; ...jump to Charge (SUBMODE=3)
                        00580                        ; ...else continue to disch (SUBMODE=2)
                        00581
00582 ;*****
00583 ;* Discharge starts here (SUBMODE=2)
00584 ;* Cursor moved to line 1 under text "Disch."
00585 ;* Then command 02h (Home Cursor) issued to LCD controller, but this
00586 ;* is dummy command - sense is to freeze it after first nibble, and
00587 ;* thus to leave PORTB,0 (ENA) in high state as long as discharging
00588 ;* lasts. After the discharging termination (if volt monitor detects
00589 ;* <4V or key 2 pressed), the command for LCD controller will be
00590 ;* completed, switching discharging transistor off.
00591 ;* If discharging is broken by key, program returns to user interface
00592 ;* for mode 4, if terminated by voltage monitor, charging takes place
00593 ;*****
0146 1086      00594      bcf     PORTB,1   ; pull LCD Reg Select low (=instr)
0147 0103      00595      clr     PORTB,1   ; high nibble of instruction 02h = 0h
0148 23F1      00596      call    Hinib_B   ; output W,4-7 to 4-bit LCD data bus
0149 23ED      00597      call    EnaLCD    ; generate En signal (1200us) for LCD
014A 3020      00598      movlw  20h       ; command 02h=home cursor(swap nibbles)
                        00599
014B 23F1      00600      call    Hinib_B   ; output W,4-7 to 4-bit LCD data bus
014C 1406      00601      bsf     PORTB,0   ; ENA activated (the command won't be
                        00602                        ; finished until Break or voltage < 4V)
014D           00603      DisLoop
014D 1D85      00604      btfs    PORTA,3   ; test key 2 status...
014E 2930      00605      goto     ExitDis  ; if low,disching manually broke by key
                        00606
014F 1C85      00607      btfs    PORTA,1   ; test voltage monitor...
0150 294D      00608      goto     DisLoop  ; if still >=4V, loop
                        00609                        ; disching terminated (voltage < 4V)
                        00610
0151 23BD      00611      call    DisEna30  ; switch off discahge transistor
                        00612
00613 ;*****
00614 ;* Charging starts here (by command or after successful discharging)
00615 ;* Minute and hour counters are init'd and counting process starts.
00616 ;* Clock(in format HH:MM) is displayed in line 1 under text "Charge".
00617 ;* If charging broken by key, program returns to user interface for
00618 ;* mode 4, if terminated by timeout (14 hours), the unit jumps to
00619 ;* SUICIDE (switches off the unit forcing the output PORTB,3 low).
00620 ;*****
0152           00621      Charge          ; Charge entry point
                        00622
0152 019B      00623      clrf    TIMOUTL   ; initialize minute counter 0...59
0153 019C      00624      clrf    TIMOUTH   ; initialize hour counter 0...13
0154           00625      ChLoop

```

```

0154 308B      00626      movlw  8bh          ; position of digital clock on LCD
0155 23CF      00627      call   WrComL      ; cursor to digital clock pos
0156 081C      00628      movf   TIMOUTH,W   ; TIMOUTH=hours in binary format
0157 2370      00629      call   Print255    ; print hour in format HH
0158 303A      00630      movlw  ':'          ; ":" = separator
0159 21D1      00631      call   PrintTL     ; print ":" and minute in format MM
                00632
015A 30E4      00633      movlw  .228        ; 228 x 263270.4 us = 60 sec
015B 008F      00634      movwf  SCRATCH     ; high byte loop counter for 1 min loop
015C          00635      Min1
015C 23D2      00636      call   GoLoop      ; 1283t (513.2us) inclusive ; 1283t
015D 23D2      00637      call   GoLoop      ; total 1026.4us ; 1283t
                00638
015E 1D85      00639      btfsz  PORTA,3     ; 2t test status of key 2...
015F 2931      00640      goto   Break4      ; - ...if low,chg manually terminated
                00641
0160 0B94      00642      decfsz COUNT,F    ; 1t low byte loop counter
0161 295C      00643      goto   Min1        ; 2t inner pass 1028.4 us
                00644
0162 0B8F      00645      decfsz SCRATCH,F  ; high byte loop counter for 1 minute
0163 295C      00646      goto   Min1        ; one pass 263270.4 us
                00647
0164 0A9B      00648      incf   TIMOUTL,F   ; advance minute counter
0165 081B      00649      movf   TIMOUTL,W   ; TIMOUTL = minute up counter
0166 3EC4      00650      addlw  -.60        ; test if 60 minutes of charging done..
0167 1C03      00651      btfsz  STATUS,C    ; if 60 minutes passed, C should be set
0168 296B      00652      goto   NotHour     ; not yet hour advance
0169 019B      00653      clrf   TIMOUTL     ; if 60 minutes done, clr minute cntr
016A 0A9C      00654      incf   TIMOUTH,F   ; ..and advance TIMOUTH=hour up counter
016B          00655      NotHour
016B 081C      00656      movf   TIMOUTH,W   ; TIMOUTH = hour up counter
016C 3EF2      00657      addlw  -.14        ; test if 14 hours of charging
016D 1C03      00658      btfsz  STATUS,C    ; if 14 hours passed, C should be set
016E 2954      00659      goto   ChLoop      ; ...if not yet 14 hours, loop
016F 29BC      00660      goto   Suicide     ; charging terminated after 14 h
                00661
00662 ;***** KEYBOARD AND PROBE
00663 ;*****
00664 ;* CurPosKb
00665 ;* This subroutine places cursor in line 2 at position taken from the
00666 ;* lookup table: table offset is addressed by W at input, and table
00667 ;* read location by variable SUBMODE.
00668 ;* High timeout counter (TIMOUTH) is initialized. This sets automatic
00669 ;* Power Off timing to about 8 minutes. TIMOUTL is of minor importance
00670 ;* here (it affects less than 2 secs of timing), so it was not worth
00671 ;* waisting one instruction word.
00672 ;* Bit DEBO,0 set to disable false recognizing of PORTA,3 low level as
00673 ;* falling edge (as if key 2 was just pressed). This could happen if
00674 ;* some function was broken by pressing key2, as those are simple port
00675 ;* tests without affecting debouncer.
00676 ;* This subroutine continues to keyboard scan subroutine.
00677 ;*
00678 ;* Input variables: SUBMODE, affects cursor position
00679 ;* Output variables: TIMOUTH=0, high timeout counter
00680 ;*****
00681 CurPosKb
0170 0711      00682      addwf  SUBMODE,W   ; add SUBMODE to lookup table offset
0171 23B6      00683      call   PclSub      ; get cursor position from table
0172 23CF      00684      call   WrComL      ; write new cursor position to LCD
0173 019C      00685      clrf   TIMOUTH
                00686
0174 1412      00687      bsf   DEBO1,0     ; set any bit in both debouncers...
0175 1413      00688      bsf   DEBO2,0     ; ...to disable false recognizing of...
                00689
                00689
                00690

```

```

00691 ;*****
00692 ;* GoKbd
00693 ;* Is main keyboard subroutine, in which program loops all the time
00694 ;* except in freq counter mode, or while wait for start condition or
00695 ;* executing some command. Program exits subrout only if some key is
00696 ;* just pressed (not if continuously pressed), or when timeout counter
00697 ;* (TIMOUTH, TIMOUTL) after 8 min reaches zero. If key 1 was pressed,
00698 ;* flag STATUS,C will be reset at exit, if key 2 was pressed, flag
00699 ;* STATUS,C will be set. If timeout detected, the routine SUICIDE is
00700 ;* executed (the unit is switched off forcing the output PORTB,3 low).
00701 ;* During keyboard scan, LEDs L, H and P are serviced. Logic state at
00702 ;* PORTA,4 affects LEDs L and H directly, and LED P is under control
00703 ;* of down counter PCOUNT. This counter is initialized at every logic
00704 ;* level transition at PORTA,4, and while counting down, if PCOUNT>0,
00705 ;* LED P is switched on.
00706 ;* Loop labeled "Unstable" adds the extra delay which timing is not
00707 ;* constant, but changes from 3 to 49t. This mins the interference
00708 ;* between the input scan and tested signal frequency.
00709 ;* Counter TMR0 is used for detecting of short pulses. At each
00710 ;* transition detected, TMR0 is cleared, then periodically tested if
00711 ;* counter state was incremented. If so, PCOUNT is initialized and LED
00712 ;* P turned on.
00713 ;* Register DJNZ is used as a freerunning counter, which divides loop
00714 ;* count by 256 and slows down PCOUNT countdown / keys scanning. Keys
00715 ;* are debounced and falling edge (pressing moment) detected by
00716 ;* rotating registers DEBO1 and DEBO2, and testing them if the key was
00717 ;* unpressed at least at 7 passes and then pressed at 1 pass.
00718 ;*
00719 ;* Input variables: none
00720 ;* Output variables: Bit STATUS,C reset if key 1 pressed, set if key 2
00721 ;*****
00722 GoKbd
0176 080E 00723      movf   DJNZ,W      ; 1t to avoid intrference,total avg 29t
0177 38F0 00724      iorlw  0f0h      ; 1t extra time range from -.16 and -1
0178 008F 00725      movwf  SCRATCH    ; 1t here SCRATCH varies 0f0h to 0ffh
0179      00726 UnStable
0179 0F8F 00727      incfsz SCRATCH,F    ; 1-17t (avg .9) adv extra timing count
017A 2979 00728      goto   UnStable    ; 2-32t (avg .17) loop loses extra time
017B 0801 00729
017C 1903 00730      movf   TMR0,W      ; TMR0 = hardware transition detector
017D 2980 00731      btfsc  STATUS,Z    ; test if transition at PORTA,4...
017E 1590 00732      goto   NoIniP     ; ...if not, do not affect LED P status
017F 0181 00733      bsf    PCOUNT,3   ; initialize PCOUNT for LED P timing
0180      00734      clr    TMR0       ; re-init hardware transition detector
0180      00735 NoIniP
0180 3028 00736      movlw  28h        ; bit 4 (TOSE) RESET: L-to-H transition
0181 1A05 00737      btfsc  PORTA,4    ; if probe tip low,leave TOSE reset...
0182 3038 00738      movlw  38h        ; if high, set TOSE: H-to-L transition
0183 1683 00739
0184 0081 00740      bsf    STATUS,RP0 ; select bank 1 of registers
0185 1283 00741      movwf  OPTION_REG ; set/reset TOSE
0185      00742      bcf    STATUS,RP0 ; reselect bank 0
0186 1019 00743
0187 1018 00744      bcf    DELAYH,0   ; init input bit in delay HI rotor
0188 3004 00745      bcf    DELAYL,0   ; init input bit in delay LOW rotor
0189 0686 00746      movlw  4          ; value 4 = bit 2 set
018A 1A05 00747      xorwf  PORTB,F    ; change state of PORTB.2 (square wave.
018B 2990 00748      ; generation probe tip hi-imp output
018C 1D06 00749      btfsc  PORTA,4    ; test probe input logic level...
018D 1418 00750      goto   InputHi    ; ...and jump if case 2: input high
018E 110C 00751      ; ...or continue if case 1: input low
018E      00752      btfss  PORTB,2    ; test hi-impedance output state and...
018F 2993 00753      bsf    DELAYL,0   ; ...turn on led L, hi-imp out was lo
018F      00754      bcf    FLAG,PTIP  ; reset flag to notify that previous...
018F      00755      ; ...state of probe tip was low
018F      00756      goto   ContInpX  ; jump to skip case 2

```

```

0190          00757 InputHi          ; entry point for case 2: input high
0190 1906     00758          btfsc  PORTB,2  ; test hi-impedance output state and...
0191 1419     00759          bsf    DELAYH,0  ; ...turn on led L, PORTB,2 was high
0192 150C     00760          bsf    FLAG,PTIP ; set flag to notify that previous...
          00761          ; ...state of probe tip was high
0193          00762 ContInpX        ; moves LHP leds from FLAG to PORTB
0193 138C     00763          bcf    FLAG,LEDL ; init input bit for led L delay rotor
0194 0898     00764          movf  DELAYL,F  ; test DEAYL status...
0195 1D03     00765          btfss STATUS,Z ; ...and skip if DELAYL=0
0196 178C     00766          bsf    FLAG,LEDL ; turn on led L if DELAYL rotor > 0
0197 0D98     00767          rlf    DELAYL,F  ; propagate bit thru DELAYL rotor
          00768
0198 130C     00769          bcf    FLAG,LEDH ; init input bit for led H delay rotor
0199 0899     00770          movf  DELAYH,F  ; test DEAYH status...
019A 1D03     00771          btfss STATUS,Z ; ...and skip if DELAYH=0
019B 170C     00772          bsf    FLAG,LEDH ; turn on led H if DELAYH rotor > 0
019C 0D99     00773          rlf    DELAYH,F  ; propagate bit thru DELAYH rotor
          00774
019D 23F0     00775          call  MoveLESD ; send leds status flag bits to PORTB
          00776
019E 0F8E     00777          incfsz DJNZ,F  ; test if this is 256th pass...
019F 2976     00778          goto  GoKbd   ; ...if not, loop
          00779
          00780 ;----- passes here each 256 pass (about 8.9ms)
          00781
01A0 128C     00782          bcf    FLAG,LEDP ; reset led P flag (set if PCOUNT>0)
01A1 0890     00783          movf  PCOUNT,F  ; test PCOUNT status...
01A2 1903     00784          btfsc STATUS,Z ; ...and skip jump if PCOUNT>0
01A3 29A9     00785          goto  PCount0 ; ...else jump if PCOUNT = 0
          00786
01A4 0390     00787          decf  PCOUNT,F  ; if PCOUNT>0, then decrement it
01A5 0818     00788          movf  DELAYL,W  ; W>0 if led L is on
01A6 0419     00789          iorwf DELAYH,W  ; W>0 if led L or led H is on
01A7 1D03     00790          btfss STATUS,Z ; ...skip if both L and H leds are off
01A8 168C     00791          bsf    FLAG,LEDP ; if PCOUNT>0, dec PCOUNT, & set led P
01A9          00792 PCount0
          00793          ; ----- key 2 test (right key)
01A9 0E05     00794          swapf PORTA,W  ; let key 1&2 status move to bits 6&7
01AA 008F     00795          movwf SCRATCH ; SCRATCH,7=key2, SCATCH,6=key1
01AB 098F     00796          comf  SCRATCH,F ; complement to set bit if key pressed
          00797
01AC 0D8F     00798          rlf    SCRATCH,F ; set C if key 2 pressed
          00799
01AD 0D93     00800          rlf    DEBO2,F  ; propagate key 2 bit thru rotor
01AE 1003     00801          bcf    STATUS,C ; reset C,notify at exit key 2 pressed
01AF 0313     00802          decf  DEBO2,W  ; DEBO2 = b'00000001' if just pressed
01B0 1903     00803          btfsc STATUS,Z ; skip return if key 2 not just pressed
01B1 0008     00804          return      ; *** exit 1: key 2 just pressed (NC)
          00805
          00806          ; ----- key 1 test (left key)
          00807
01B2 0D8F     00808          rlf    SCRATCH,F ; set C if key 1 pressed
          00809
01B3 0D92     00810          rlf    DEBO1,F  ; propagate key 1 bit thru rotor
01B4 1403     00811          bsf    STATUS,C ; set C, notify at exit key 1 pressed
01B5 0312     00812          decf  DEBO1,W  ; DEBO1 = b'00000001' if just pressed
01B6 1903     00813          btfsc STATUS,Z ; skip return if key 1 not just pressed
01B7 0008     00814          return      ; *** exit 2: key 1 just pressed (C)
          00815
01B8 0B9B     00816          decfsz TIMOUTL,F ; timeout lo counter...
01B9 2976     00817          goto  GoKbd   ; ... not yet zero: loop
          00818
01BA 0B9C     00819          decfsz TIMOUTH,F ; timeout hi counter...
01BB 2976     00820          goto  GoKbd   ; ... not yet zero: loop
          00821          ; *** exit 3:cont with timeout process
          00822

```

```

00823 ;*****
00824 ;* Power Off entry point
00825 ;* Wait until both keys off for 34 ms, and then switch power off.
00826 ;* PORTB,3, when low, switches the unit off.
00827 ;*****
01BC 00828 Suicide
01BC 21BF 00829      call    KeysOff      ; test keys off to avoid re-triggering
01BD 0186 00830      clr    PORTB        ; pull PORTB,3 low to switch power off
01BE 29BE 00831      goto    $            ; loop until power off
00832
00833 ;*****
00834 ;* KeysOff
00835 ;* Loop until both keys off for 34 ms, then exit
00836 ;*
00837 ;* Input variables: none
00838 ;* Output variables: TIMOUTH is cleared to 0
00839 ;*****
01BF 00840 KeysOff
01BF 019C 00841      clr    TIMOUTH      ; initialize pointer
01C0 00842 BothOff
01C0 1905 00843      btfsc  PORTA,2        ; skip if key 1 on
01C1 1D85 00844      btfss  PORTA,3        ; do not skip if key 2 on
01C2 29BF 00845      goto    KeysOff        ; reinitialize ptr if any key on
01C3 23D0 00846      call    loop130         ; loop 130us
01C4 0B9C 00847      decfsz TIMOUTH,F        ; test pointer
01C5 29C0 00848      goto    BothOff        ; loop 256xs to verify both keys off
01C6 0008 00849      return                ; both keys are off for at least 34 ms
00850
00851 ;***** ANALYZER
00852 ;*****
00853 ;* Pointer2
00854 ;* Writes 2 measuring points for analyzer reference. 1st=TIMOUTL*10+5,
00855 ;* then TIMOUTL is incremented by 2 and the second one is TIMOUTL*10+0
00856 ;*
00857 ;* Input variables: TIMOUTL will first be printed as TIMOUTL*10+5
00858 ;* Output variables: TIMOUTL incremented by 3 (pointer advanced by 30)
00859 ;*****
01C7 00860 Pointer2
01C7 21CF 00861      call    Pointer1        ; first two digits
01C8 3035 00862      movlw  '5'            ; third digit is 5 for odd pointer
01C9 0A9B 00863      incf  TIMOUTL,F        ; each incr advances pointer by 10
01CA 21CD 00864      call  AdvToCh          ; advance ptr and print "5"
01CB 21CF 00865      call  Pointer1        ; first two digits
01CC 304F 00866      movlw  '0'            ; third digit is 0 for even pointer
01CD 00867 AdvToCh
01CD 0A9B 00868      incf  TIMOUTL,F        ; each incr advances pointer by 10
01CE 2BD8 00869      goto  Char            ; print 0 or 5
00870
00871 ;*****
00872 ;* Pointer1
00873 ;* Writes blank, then symbol "^" and then converts TIMOUTL and prints
00874 ;* as 2 digits.
00875 ;*
00876 ;* Input variables: TIMOUTL, bin value which prints as 2-digits
00877 ;* Output variables: none
00878 ;*****
01CF 00879 Pointer1
01CF 23D7 00880      call  Blank            ; skip first 3 samples (one character)
01D0 305E 00881      movlw  '^'            ; "^" (pointing tool)
01D1 00882 PrintTL
01D1 23D8 00883      call  Char            ; print "^"
01D2 081B 00884      movf  TIMOUTL,W        ; TIMOUTL is the main pointer for...
01D3 2B70 00885      goto  Print255        ; ...first two digits
00886
00887 ;*****
00888 ;* ModelShow

```

```

00889 ;* Write measuring points at row 2, and wait until keys are released.
00890 ;* Then incr. SHOWCOU in range 0...4 and continue to Draw subroutine
00891 ;* This command, which executes when key 2 is pressed in analyzer mode,
00892 ;* while the cursor is on the number of 60-sample group, advances the
00893 ;* pointer.
00894 ;* It continues to Draw routine.
00895 ;*****
01D4      00896 ModelShow
01D4 23CE      00897      call   Row2      ; pointers are in row 2
01D5 1003      00898      bcf    STATUS,C    ; prepare for multiplying by 2
01D6 0D17      00899      rlf    SHOWCOU,W   ; W=SHOWCOU*2
01D7 009B      00900      movwf TIMOUTL     ; TIMOUTL=SHOWCOU*2
01D8 0D9B      00901      rlf    TIMOUTL,F   ; TIMOUTL-SHOWCOU*4
01D9 079B      00902      addwf TIMOUTL,F   ; TIMOUTL-SHOWCOU*6
                00903
01DA 21C7      00904      call   Pointer2   ; print 1st and 2nd pointer
01DB 21C7      00905      call   Pointer2   ; print 3rd and 4th pointer
                00906
01DC 21BF      00907      call   KeysOff    ; wait until key off
01DD 227D      00908      call   PrintM1    ; restore normal row 2
                00909
01DE 0A97      00910      incf   SHOWCOU,F  ; advance number of groups displayed
01DF 1917      00911      btfs   SHOWCOU,2  ; test if SHOWCOU=5: first test bit2...
01E0 1C17      00912      btfs   SHOWCOU,0  ; test if SHOWCOU=5: ...then test bit 0
01E1 29E3      00913      goto   Draw       ; skip wrapping if SHOWCOU<5
01E2 0197      00914      clrf   SHOWCOU    ; SHOWCOU cycle in range 0...4
                00915
00916 ;*****
00917 ;* Draw
00918 ;* This subroutine writes 20 pseudographic chars in line 1 in analyzer
00919 ;* mode. First, whole buffer is rotated 0/60/120/180/240 bit places to
00920 ;* the right (if SHOWCOU=0/1/2/3/4, respectively) to adj. the sequence
00921 ;* to be displayed to the start of the buffer. Then the string of 20
00922 ;* 3-bit groups is rotated right, and displayed as 20 special chars
00923 ;* (codes 0-7), defined at program setup (at loop labeled GoGraph).
00924 ;* Then the buffer is rotated again, to the total of 304 bit places,
00925 ;* so the buffer contents is unmodified on exit.
00926 ;*
00927 ;* Input variables:
00928 ;* SHOWCOU, denotes which group of 60 samples will be displayed
00929 ;* Output variables: none
00930 ;*****
01E3      00931 Draw
01E3 0817      00932      movf   SHOWCOU,W  ; prep to rotate buf SHOWCOU*60 times
01E4 1D03      00933      btfs   STATUS,Z   ; avoid rotating if SHOWCOU=0
01E5 21F9      00934      call   Carusel    ; rotate buffer SHOWCOU*60 times
                00935
01E6 22BF      00936      call   Row1       ; samples must be written in row 1
01E7 3014      00937      movlw  .20        ; .20 characters to write
01E8 0096      00938      movwf  CHARCOU    ; CHARCOU is the main character counter
01E9      00939 Go20Chars
01E9 018F      00940      clrf   SCRATCH    ; register for 3-bit code gen (0...7)
01EA 2202      00941      call   RRBuf      ; bit from buffer in C...
01EB 0D8F      00942      rlf    SCRATCH,F  ; ...bit from C in code register
01EC 2202      00943      call   RRBuf      ; bit from buffer in C...
01ED 0D8F      00944      rlf    SCRATCH,F  ; ...bit from C in code register
01EE 2202      00945      call   RRBuf      ; bit from buffer in C...
01EF 0D0F      00946      rlf    SCRATCH,W  ; ...bit from C in code reg and in W
01F0 23D9      00947      call   CharNCC    ; draw one char = 3 samples
01F1 0B96      00948      decfsz CHARCOU,F  ; 20 characters written?
01F2 29E9      00949      goto   Go20Chars ; no, loop
                00950
01F3 0817      00951      movf   SHOWCOU,W  ; prep to rotate to total of 304 bits
01F4 3C04      00952      sublw  .4         ; W=4-SHOWCOU
01F5 1D03      00953      btfs   STATUS,Z   ; avoid rotating if SHOWCOU=0
01F6 21F9      00954      call   Carusel    ; rotate buffer again to restore it

```

```

01F7 2200      00955      call   RRBuf4      ; four more times to get 304 times
01F8 28CD      00956      goto   Farml       ; done, go back to user interface
00957
00958 ;*****
00959 ;* Carusel
00960 ;* This subroutine rotates BUFFER right W*60 times
00961 ;* Note: if W=0, rotating will be performed 1024 times
00962 ;*
00963 ;* Input variables: W,how many (*60) x the buf is rotated right (W>0)
00964 ;* Output variables: none
00965 ;*****
01F9          00966 Carusel
01F9 008F      00967      movwf  SCRATCH     ; SCRATCH=W
01FA 0E8F      00968      swapf SCRATCH,F   ; SCRATCH=W*16
01FB 028F      00969      subwf  SCRATCH,F   ; SCRATCH=W*15
01FC          00970 RRLoop
01FC 2200      00971      call   RRBuf4     ; 4 rotates in every pass
01FD 0B8F      00972      decfsz SCRATCH,F  ; done W*15*4 times?
01FE 29FC      00973      goto   RRLoop     ; no, continue rotating BUFFER
01FF 0008      00974      return            ; done
00975
00976 ;*****
00977 ;* RRBuf4 executes RRBuf 4 times
00978 ;* RRBuf rotates buffer(38 bytes=304 bits) right for one bit position.
00979 ;* Bit STATUS,C is first loaded from the first bit in buffer, so
00980 ;* will rotating be completely performed at 304 bits, not through C.
00981 ;*
00982 ;* Input/Output variables: none
00983 ;*****
0200          00984 RRBuf4
0200 2201      00985      call   RRBuf2     ; rotate BUFFER 2* and then again 2*
0201          00986 RRBuf2
0201 2202      00987      call   RRBuf      ; rotate BUFF once and then again once
0202          00988 RRBuf
0202 304B      00989      movlw  BUFFER+.37 ; start with last byte to be rotated
0203 0084      00990      movwf  FSR        ; FSR = pointing register for rotating
0204 3026      00991      movlw  .38        ; 38 bytes total buffer
0205 0094      00992      movwf  COUNT      ; byte counter
00993
0206 1003      00994      bcf    STATUS,C   ; C rotated into BUFFER+.37,so it...
0207 1826      00995      btfsc  BUFFER,0   ; ...must be equal to bit BUFFER+0,0...
0208 1403      00996      bsf    STATUS,C   ; ...to perform non-destruct rotating
0209          00997 ByteLoop
0209 0C80      00998      rrf    INDF,F     ; byte rotated here
020A 0384      00999      decf   FSR,F     ; let FSR point to next byte
020B 0B94      01000     decfsz COUNT,F    ; COUNT is byte counter, done?
020C 2A09      01001     goto   ByteLoop   ; no, loop
020D 0008      01002     return            ; here the output bit is in STATUS,C
01003
01004 ;*****
01005 ;* ModelGo
01006 ;* This is entry point for analyzer start command (symbol * ). After
01007 ;* clearing line of LCD, the program vectors to routines which handle
01008 ;* different sampling rates. Three highest rates (1 MHz, 500 KHz and
01009 ;* 228 KHz are sampled at individual routines (Get1MHz, Get500KHz and
01010 ;* Get228KHz), andthe remining rates at routine GetSlowClk. All those
01011 ;* routines (except Get1MHz, which is location-sensitive (so it is at
01012 ;* the very beginning of the program), are listed here.
01013 ;*
01014 ;* Input/Output variables: none
01015 ;*****
020E          01016 ModelGo
01017
020E 22B7      01018     call   ClrRow1    ; clear LCD line 1 and turn LEDs off
01019
020F 0815      01020     movf   RATE,W     ; RATE = sample rate in range 0...15

```

```

0210 1903          01021      btfsc  STATUS,Z      ; skip if sample rate>0
0211 2801          01022      goto   Get1MHz      ; jump to individual routine if RATE=0
                   01023
                   01024          ; --- try 500 KHz rate
0212 008F          01025      movwf  SCRATCH      ; SCRATCH = RATE
0213 0B8F          01026      decfsz SCRATCH,F    ; test if RATE=1
0214 2A2B          01027      goto   Try228KHz   ; if not RATE=1, then try if RATE=2
                   01028          ; if RATE=1, then drop to Get500KHz
                   01029
                   01030 ;*****
01031 ;* Get500KHz
01032 ;* This subroutine fetches 304 samples at 2us rate (5 instr cycles
01033 ;* timing).
01034 ;* Call Common initializes loop counter (COUNT) to 38*8=304 samples
01035 ;* and FSR to point to BUFFER. It also presets T0SE bit depended on
01036 ;* XTOX bit (in FLAG register)to enable proper edge detecting, as it
01037 ;* will affect TMR0 state.
01038 ;* State of key 2(Break) tested while waiting for starting condition.
01039 ;*
01040 ;* Input/Output variables: none
01041 ;*****
0215              01042      Get500KHz      ; 5 t read cycle
0215 3026          01043      movlw  .38          ; 38 bytes * 8 bits = 304 samples
0216 226F          01044      call   Common      ; initialize COUNT, FSR, hi-imp out...
                   01045          ; ...bit XTOX and T0SE bit
0217              01046      Edge500
0217 1D85          01047      btfss  PORTA,3     ; test status of key 2 and...
0218 28CC          01048      goto   Break       ; ...jump to Break routine if pressed
                   01049
0219 0801          01050      movf   TMR0,W      ; TMR0 = logic level edge detector
021A 1903          01051      btfsc  STATUS,Z     ; test if there was egde...
021B 2A17          01052      goto   Edge500    ; ...and loop if not
021C 0384          01053      decf   FSR,F      ; adj pointer as it will be advanced...
                   01054          ; ... before data write
021D              01055      Get500Loop
021D 0C85          01056      rrf    PORTA,F     ; <-- move input status to C
021E 0A84          01057      incf   FSR,F      ; advance write pointer
021F 0C80          01058      rrf    INDF,F     ; write bit C in destination rotor
0220 3006          01059      movlw  .6          ; initialize count for 6 bits
0221 008E          01060      movwf  DJNZ       ; DJNZ = bit counter
0222              01061      Go6Bits
0222 0C85          01062      rrf    PORTA,F     ; <-- 6* move input status to C
0223 0C80          01063      rrf    INDF,F     ; write bit C in destination rotor
0224 0B8E          01064      decfsz DJNZ,F    ; DJNZ = bit counter
0225 2A22          01065      goto   Go6Bits    ; loop if not yet 6 bits fetched
                   01066
0226 0C85          01067      rrf    PORTA,F     ; <-- move input status to C
0227 0C80          01068      rrf    INDF,F     ; write bit C in destination rotor
0228 0B94          01069      decfsz COUNT,F   ; COUNT = byte counter
0229 2A1D          01070      goto   Get500Loop ; loop if not yet 38 bytes fetched
022A 2A6A          01071      goto   Finished   ; all bits fetched; go display them
                   01072
01073 ;*****
01074 ;* Test if register SCRATCH reaches 0 after decrementing(this happens
01075 ;* if RATE = 2), if so drop to Get228KHz else go to GetSlowClk
01076 ;*****
022B              01077      Try228KHz
022B 0B8F          01078      decfsz SCRATCH,F ; test RATE status (SCRATCH=RATE-1)
022C 2A3F          01079      goto   GetSlowClk ; jump if not RATE=2
                   01080
01081 ;*****
01082 ;* This subroutine fetches 304 samples at 4.4us rate (11 instruction
01083 ;* cycles timing).
01084 ;* Call Common304 initializes loop counter (COUNT) to 304 samples
01085 ;* and FSR to point to BUFFER. It also presets T0SE bit depended on
01086 ;* XTOX bit (in FLAG register)to enable proper edge detecting, as it

```

```

01088 ;* will affect TMR0 state.
01089 ;* State of key 2(Break) tested while waiting for starting condition.
01090 ;*
01091 ;* Input/Ouput variables: none
01092 ;*****
022D      01093 Get228KHz           ; 11 t read cycle
022D 226C      01094          call   Common304   ; init COUNT(lo), TIMOUTH(hi byte)...
01095          ; ...FSR, hi-imp outbit XTOX & TOSE bit
022E      01096 Edge228
022E 1D85      01097          btfs   PORTA,3   ; test status of key 2 and...
022F 28CC      01098          goto   Break     ; ...jump to Break routine if pressed
01099
0230 0801      01100          movf   TMR0,W    ; TMR0 = logic level edge detector
0231 1903      01101          btfs   STATUS,Z   ; test if there was egde...
0232 2A2E      01102          goto   Edge228   ; ...and loop if not
0233
0233 2A34      01104          goto   $+1       ; 2      two extra cycles to make 11t
0234          01105 Go228B
0234 0C85      01106          rrf    PORTA,F ; <--- ; 1      move input bit to C
0235 0C80      01107          rrf    INDF,F   ; 1      write bitC in destination rotor
01108
0236 0314      01109          decf   COUNT,W  ; 1      COUNT = bit counter
0237 3907      01110          andlw  7        ; 1      test if 8th pass...
0238 1903      01111          btfs   STATUS,Z   ; 1 (2) ...and skip if not
0239 0A84      01112          incf   FSR,F    ; 1 (0) ...else advance pointer
01113
023A 0B94      01114          decfsz COUNT,F  ; 1      COUNT = (lo byte) bit counter
023B 2A33      01115          goto   Go228    ; 2      loop if not yet = 0
01116
023C 0B9C      01117          decfsz TIMOUTH,F ; TIMOUTH = (hi byte) bit counter
023D 2A34      01118          goto   Go228B   ; this does not add extra cycles, as it
01119          ; ...jumps after goto $+1
023E 2A6A      01120          goto   Finished ; all bits fetched; go display them
01121
01122 ;*****
01123 ;* GetSlowClk
01124 ;* This subroutine fetches 304 samples at variable rate, depended on
01125 ;* RATE (SCRATCH=RATE-3). Timing constant is loaded from lookup table
01126 ;* located at DATA EEPROM (locations 30h-3ch).
01127 ;*
01128 ;* Call Common304 initializes 16-bit loop counter to 304 samples
01129 ;* (lo byte: COUNT=.304-.256, hi byte: TIMOUTH=.1)
01130 ;* and FSR to point to BUFFER. It also presets TOSE bit depended on
01131 ;* XTOX bit (in FLAG register)to enable proper edge detecting, as it
01132 ;* will affect TMR0 state.
01133 ;*
01134 ;* Rates 3-11 (100KHz-2.4KHz) have loop period of factor from EEPROM
01135 ;* table multiplied by 5 instruction cycles and adding 20 instruction
01136 ;* cycles (Factor*5T+20T), and rates 12-15 (1.2KHz-40Hz) multilied the
01137 ;* factor by 417 instruction cycles and adding 415 instruction cycle
01138 ;* (Factor*417T+415T)
01139 ;*
01140 ;* RATE = 3, factor: 1, T/cycle: 25, Freq: 100 KHz
01141 ;* RATE = 4, factor: 6, T/cycle: 50, Freq: 50 KHz
01142 ;* RATE = 5, factor: 9, T/cycle: 65, Freq: 38.4 KHz
01143 ;* RATE = 6, factor: 16, T/cycle: 100, Freq: 25 KHz
01144 ;* RATE = 7, factor: 22, T/cycle: 130, Freq: 19.2 KHz
01145 ;* RATE = 8, factor: 46, T/cycle: 250, Freq: 10 KHz
01146 ;* RATE = 9, factor: 48, T/cycle: 260, Freq: 9.6 KHz
01147 ;* RATE = 10, factor: 100, T/cycle: 520, Freq: 4.8 KHz
01148 ;* RATE = 11, factor: 204, T/cycle: 1040, Freq: 2.4 KHz
01149 ;* RATE = 12, factor: 5, T/cycle: 2500, Freq: 1 KHz
01150 ;* RATE = 13, factor: 14, T/cycle: 6253, Freq: 400 Hz
01151 ;* RATE = 14, factor: 59, T/cycle: 25018, Freq: 100 Hz
01152 ;* RATE = 15, factor: 149, T/cycle: 62548, Freq: 40 Hz
01153 ;*

```

```

01154 ;* State of key 2 (Break) is tested while waiting for start condition.
01155 ;* LED P is turned on while sampling, indicate sample period at slower
01156 ;* rates, in which it appears to be visible.
01157 ;*
01158 ;* Input variables: RATE, affects timing factor
01159 ;* Output variables: none
01160 ;*****
023F      01161 GetSlowClk
023F 030F      01162      decf   SCRATCH,W      ; W = RATE-3
0240 3E30      01163      addlw  .48              ; rate table in data eeprom @ addr .48
                                01164
0241 22AC      01165      call   AGet_EE           ; get time const. from dataeeprom table
0242 008F      01166      movwf  SCRATCH          ; time const for rates 3-15 -> SCRATCH
                                01167
0243 226C      01168      call   Common304        ; init COUNT(lo), TIMOUTH(hi byte)...
                                01169                      ; ...FSR, hi-imp outbit XTOX & T0SE bit
0244      01170 EdgeSlow
0244 1D85      01171      btfss  PORTA,3          ; test status of key 2 and...
0245 28CC      01172      goto   Break            ; ...jump to Break routine if pressed
                                01173
0246 0801      01174      movf   TMR0,W           ; TMR0 = logic level edge detector
0247 1903      01175      btfsc  STATUS,Z        ; test if there was egde...
0248 2A44      01176      goto   EdgeSlow        ; ...and loop if not
                                01177
0249 1686      01178      bsf    PORTB,5         ; turn led P on, notify sampling on
                                01179
024A      01180 GoSlow
024A 0C85      01181      rrf    PORTA,F ; <--- ; 1      move input bit to C
024B 0C80      01182      rrf    INDF,F       ; 1      write bitC in destination rotor
                                01183
024C 0314      01184      decf   COUNT,W         ; 1      COUNT = bit counter
024D 3907      01185      andlw  7              ; 1      test if 8th pass...
024E 1903      01186      btfsc  STATUS,Z        ; 1 (2) ...and skip if not
024F 0A84      01187      incf   FSR,F          ; 1 (0) ...else advance pointer
                                01188
0250 1995      01189      btfsc  RATE,3          ; 1 1    if bits2 and 3 cleared, that...
0251 1D15      01190      btfss  RATE,2          ; 2 1    ...means that rate<12...
0252 2A62      01191      goto   Not417         ; - 2    ...if so, jump to short timing
                                01192                      ; -----417 (RATE 12-15,bits 2,3 set)
0253 080F      01193      movf   SCRATCH,W       ; 1      SCRATCH = timing constant
0254 0096      01194      movwf  CHARCOU         ; 1      CHARCOU = loop counter
0255      01195 Loop417
0255 3052      01196      movlw  .82             ; 1 \
0256 008E      01197      movwf  DJNZ           ; 1 \
0257 23D3      01198      call   Loop7          ; 411 > total cyc here 417*SCRATCH-1
0258 0000      01199      nop                    ; 1 /
0259 0B96      01200      decfsz CHARCOU,F       ; 1 / 2t at exit only
025A 2A55      01201      goto   Loop417        ; 2 / 0t at exit only
                                01202
025B 0806      01203      movf   PORTB,W         ; 1
025C 39DF      01204      andlw  0dfh           ; 1      reset bit 5 (LED P)
025D 1E14      01205      btfss  COUNT,4        ; 1 2
025E 3820      01206      iorlw  20h            ; 1 0    set bit 5 (LED P) if COUNT,4 = 0
025F 0086      01207      movwf  PORTB          ; 1      blink LED P while sampling each
                                01208                      ; 16th pass
0260 304E      01209      movlw  .78             ; 1      constant for long timing
0261 2A64      01210      goto   SameAs5        ; 2      skip short timing
0262      01211 Not417
0262 2A63      01212      goto   $+1            ; 2      waist 2 cycles
0263 080F      01213      movf   SCRATCH,W       ; 1      SCRATCH = timing constant
0264      01214 SameAs5
0264 008E      01215      movwf  DJNZ           ; 1      DJNZ = loop counter
0265 23D2      01216      call   GoLoop         ; short time: 3T+5T*SCRATCH, long:393T
                                01217
0266 0B94      01218      decfsz COUNT,F        ; 1      COUNT = (lo byte) bit counter
0267 2A4A      01219      goto   GoSlow         ; 2      loop if not yet = 0

```

```

01220
0268 0B9C      01221      decfsz  TIMOUTH,F      ; TIMOUTH = (hi byte) bit counter
0269 2A4A      01222      goto    GoSlow        ; adds 2T extra ltime, after 48th pass
026A          01223      Finished
026A 0197      01224      clrf    SHOWCOU      ; reset pointer to 1st group of 60 samp
026B 29E3      01225      goto    Draw          ; all bits fetched; display them
01226
01227 ;*****
01228 ;* Common304
01229 ;* This subroutine initializes low byte loop counter (COUNT) to 304
01230 ;* samples (lo byte: COUNT=.304-.256, hi byte: TIMOUTH=.1+1)
01231 ;* and FSR to point to BUFFER. It also presets TOSE bit depended on
01232 ;* XTOX bit (in FLAG register)to enable proper edge detecting, as it
01233 ;* will affect TMR0 state.
01234 ;* Entry point Common allows subroutines Get1MHz and GetSlowClk to
01235 ;* preset COUNT to another values.
01236 ;*
01237 ;* Input variables:
01238 ;*   For entry point COMMON, register W is placed in COUNT
01239 ;* Output variables:
01240 ;*   COUNT is initialized to # of loop passes (W or low byte of 304)
01241 ;*   TMR0 is cleared
01242 ;*   TOSE and PORTB,2 are copied from FLAG,XTOX
01243 ;*****
026C          01244      Common304
026C 3002      01245      movlw   .2            ; hi byte=2 for reg lo byte value...
01246                        ; ...plus extra 256 passes
026D 009C      01247      movwf  TIMOUTH       ; hi byte loop counter for 304 passes
026E 3030      01248      movlw  .304-.256    ; lo byte value for 304 passes
026F          01249      Common
026F 0094      01250      movwf  COUNT         ; COUNT = loop counter
0270 3026      01251      movlw  BUFFER        ; first byte of destination
0271 0084      01252      movwf  FSR           ; FSR = destination pointer
01253
0272 3028      01254      movlw  28h          ; initialize TOSE fot L-to-H transition
0273 1106      01255      bcf    PORTB,2      ; clr hi-imp out if expecting rise edge
01256
0274 1A0C      01257      btfsz  FLAG,XTOX    ; test slctd edge for start condition
0275 2A78      01258      goto   ToOption     ; and skip falling edge if rise slctd
01259
0276 3038      01260      movlw  38h          ; initialize TOSE fot H-to-L transition
0277 1506      01261      bsf    PORTB,2      ; set hi-imp out if expecting fall edge
0278          01262      ToOption
0278 1683      01263      bsf    STATUS,RP0   ; select bank 1 of registers
0279 0081      01264      movwf  OPTION_REG   ; set/reset TOSE
027A 1283      01265      bcf    STATUS,RP0   ; reselect bank 0
027B 0181      01266      clrf  TMR0          ; initialize TMR0 as edge detector
027C 0008      01267      return             ; finished
01268
01269 ;*****
01270 ;* PrintM1
01271 ;* Print string at line 2 in analyzer mode.
01272 ;* At pos 0, rate in format XX[.]X[M|K]Hz/XX[.]X[u|m]s is printed.
01273 ;* Those values picked from table located in Data EEPROM, locations
01274 ;* 0-2Fh. Register CHARCOU is used to fill blanks up to pos 13 in line
01275 ;* 2, to disable phantom characters appearance when changing rate from
01276 ;* some long-string to short-string value.
01277 ;* Symbol for starting or rising edge for starting event is written at
01278 ;* pos 13, symbol "*" for start command at pos 15 and the number of
01279 ;* group displayed at pos. 17.
01280 ;*
01281 ;* Input variables:
01282 ;*   RATE will be printed in pos 0, row2
01283 ;*   bit FLAG,XTOX affects the printed symbol of rising/falling edge
01284 ;*   SHOWCOU (number of group displayed) is printed as numeric (+1)
01285 ;* Output variables: none

```

```

01286 ;*****
027D 01287 PrintM1
027D 23CE 01288 call Row2 ; move cursor to row 2
01289
027E 300D 01290 movlw .13 ; init counter for 13 char fix format
027F 0096 01291 movwf CHARCOU ; CHARCOU = character counter
01292
0280 0815 01293 movf RATE,W ; W = RATE
0281 0715 01294 addwf RATE,W ; W = 2 * RATE
0282 0715 01295 addwf RATE,W ; W = 3 * RATE (ea rate has 3 bytes
01296 ; in table)
0283 22AC 01297 call AGet_EE ; get 1st byte via table in data eeprom
0284 009B 01298 movwf TIMOUTL ; TIMOUTL = 1st byte from table
0285 0E1B 01299 swapf TIMOUTL,W ; move to bits 0-3, bits 4-7 are freq
01300
0286 22AD 01301 call Get_EE ; get 2nd byte via table in data eeprom
01302
0287 1B1B 01303 btfsc TIMOUTL,6 ; bit 6 = decimal point for frequency
0288 148C 01304 bsf FLAG,DP ; set decimal point bit if bit 6 set
01305
0289 2373 01306 call Print3 ; display sampling frequency
01307
028A 304D 01308 movlw 'M' ; for "MHz" display
028B 0895 01309 movf RATE,F ; if RATE=0...
028C 1D03 01310 btfss STATUS,Z ; ...then let it be MHz
028D 304B 01311 movlw 'K' ; for "KHz" display
028E 1B9B 01312 btfsc TIMOUTL,7 ; bit 7=KHz or MHz,skip 1st char if clr
028F 23D8 01313 call Char ; print "M" or "K" if bit 7 set
0290 3070 01314 movlw TxtHz-1+1 ; for "Hz" display
0291 23DD 01315 call Write ; print "Hz"
01316
0292 081B 01317 movf TIMOUTL,W ; TIMOUTL = 1st byte from table
01318
0293 22AD 01319 call Get_EE ; get 2nd byte via table in data eeprom
01320
0294 191B 01321 btfsc TIMOUTL,2 ; bit 2 = decimal point for period
0295 148C 01322 bsf FLAG,DP ; set decimal point bit if bit 2 set
01323
0296 2373 01324 call Print3 ; display digits for period
01325
0297 30E4 01326 movlw 0e4h ; Greek "micro"
0298 1D9B 01327 btfss TIMOUTL,3 ; if bit 3 set, let it be "micro"
0299 306D 01328 movlw 'm' ; ...if not, convert to "milli" (m)
029A 23D8 01329 call Char ; print "micro" or "m"
029B 3073 01330 movlw 's' ; s stands for seconds
029C 23D8 01331 call Char ; print "s"
029D 01332 XtraChar ; adds extra (CHARCOU) blanks
029D 3020 01333 movlw ' ' ; print blank to overprint prev string
029E 23D9 01334 call CharNCC ; print blank without affecting CHARCOU
029F 0B96 01335 decfsz CHARCOU,F ; CHARCOU=character counter
02A0 2A9D 01336 goto XtraChar ; loop if not yet pos 13
01337
01338 ; ---- here is start condition symbol
02A1 3001 01339 movlw 1 ; symbol of rising edge
02A2 1E0C 01340 btfss FLAG,XTOX ; let it be rising if XTOX set
02A3 3004 01341 movlw 4 ; symbol of falling edge
02A4 23D6 01342 call CharBl ; print symbol and blank
02A5 01343 KaoAna ; ---- here is "start" ("*") symbol
02A5 302A 01344 movlw '*' ; "start" symbol
02A6 23D6 01345 call CharBl ; print "start" symbol and blank
01346 ; --- here follows # of 60 smples group
02A7 0A17 01347 incf SHOWCOU,W ; SHOWCOU = number of 60 samples group
02A8 23C9 01348 call Num ; print it (incremented by 1)
02A9 23D7 01349 call Blank ; print blank
02AA 01350 Arrow
02AA 307E 01351 movlw 7eh ; 7Eh=right arrow in LM032L char set

```

```

02AB 2BD8      01352      goto      Char          ; print arrow in rightmost pos
01353
01354 ;*****
01355 ;* AGet_EE
01356 ;* Get_EE
01357 ;* This is routine for reading from Data EEPROM. Writing to BIN4+0 and
01358 ;* BIN4+1 is also integrated here, as those variables are used for bin
01359 ;* to decimal conversion.
01360 ;*
01361 ;* Input variables: W, data address at AGet_EE
01362 ;* Output variables: BIN4, binary data of rate display from DATA EEPROM
01363 ;*****
02AC          01364 AGet_EE
02AC 0089      01365      movwf    EEADR          ; initialize eeprom address pointer
02AD          01366 Get_EE
02AD 3903      01367      andlw   3              ; hi byte BIN4 for freq display
02AE 009F      01368      movwf   BIN4+1        ; BIN4+1 = hi byte for range 0...999
01369
02AF 1683      01370      bsf     STATUS,RP0    ; select bank 1 of registers
02B0 1408      01371      bsf     EECON1,RD     ; set handshaking bit for data ee read
02B1 1283      01372      bcf     STATUS,RP0    ; reselect bank 0
02B2 0808      01373      movf   EEDATA,W       ; reading from data eeprom
02B3 0A89      01374      incf   EEADR,F        ; adv address pointer for future read
02B4 009E      01375      movwf  BIN4           ; lo byte BIN4 for freq display
02B5 108C      01376      bcf     FLAG,DP       ; clear decimal point flag
02B6 0008      01377      return          ; finished
01378
01379 ;*****
01380 ;* ClrRow1
01381 ;* SameAs20
01382 ;* This subroutine clears line 1 of LCD and switches off LEDs. Entry
01383 ;* point SameAs20 allows clearing some other number of character
01384 ;* positions starting from line 1 of LCD. All LEDs are turned off,
01385 ;* flags for LEDs also Cursor pointer of LCD is restored to first pos
01386 ;* of line 1 at exit of subroutine.
01387 ;*
01388 ;* Input variables:
01389 ;*     entry point ClrRow1: none
01390 ;*     entry point SameAs20: W=number of characters to be cleared
01391 ;* Output variables:
01392 ;*     CHARCOU is decremented by the number of cleared characters
01393 ;*     SHOWCOU is cleared to 0
01394 ;*****
01395 ;* Row1
01396 ;* This subroutine moves cursor to pos 0 of row 1 on LCD.
01397 ;*
01398 ;* Input variables: none
01399 ;* Output variables: none
01400 ;*****
02B7          01401 ClrRow1
02B7 3014      01402      movlw  .20            ; 20 spaces (one row) to write
02B8          01403 SameAs20
02B8 0097      01404      movwf  SHOWCOU        ; SHOWCOU = space counter
02B9 22BF      01405      call   Row1           ; move cursor to row 1
02BA          01406 LoopClD
02BA 23D7      01407      call   Blank          ; print one space
02BB 0B97      01408      decfsz SHOWCOU,F      ; SHOWCOU = space counter
02BC 2ABA      01409      goto   LoopClD        ; loop if not yet 20 spaces
01410
02BD 301F      01411      movlw  1fh            ; bits 7,6,5 (LED flags) reset
02BE 058C      01412      andwf  FLAG,F         ; turn LED flags off
02BF          01413 Row1
02BF 3080      01414      movlw  080h           ; command for line 1
02C0 2BCF      01415      goto   WrComL         ; go write command
01416
01417 ;***** SERIAL CODE RECEIVER

```

```

01418 ;*****
01419 ;* Mode2Show
01420 ;* This subroutine advances SHOWCOU in range 0..5, and then continues
01421 ;* to subroutine Show2
01422 ;*****
02C1 Mode2Show
02C1 0A97 01424      incf    SHOWCOU,F      ; advance SHOWCOU
02C2 1917 01425      btfsc   SHOWCOU,2      ; bits 1 & 2 will be set if SHOWCOU...
02C3 1C97 01426      btfss   SHOWCOU,1      ; ...is equal to 5...
02C4 2AC6 01427      goto    Show2          ; ...if not, skip clearing
02C5 0197 01428      clrf   SHOWCOU       ; cycle show counter in range 0..5
01429
01430 ;*****
01431 ;* Show2
01432 ;* This subroutine prints the 7 bytes of Buffer (+0,+7,+14,+21,+28 or
01433 ;* +35) in hex mode at line 1, and the same bytes in ASCII at line 2.
01434 ;* ASCII representation is with bit 7 reset, and non-printables (<20h)
01435 ;* are printed as dots
01436 ;*
01437 ;* Input variables:
01438 ;*     SHOWCOU, denotes which group of 7 bytes will be displayed
01439 ;* Output variables:
01440 ;*     CHARCOU is decremented by the number of characters printed
01441 ;*****
02C6 Show2
02C6 3025 01443      movlw   BUFFER-1       ; source pointer for reading -1
02C7 1917 01444      btfsc   SHOWCOU,2      ; if bit 2 of SHOWCOU set...
02C8 3E1C 01445      addlw   .28           ; ...then add 28 (4 groups) to pointer
02C9 1897 01446      btfsc   SHOWCOU,1      ; if bit 1 of SHOWCOU set...
02CA 3E0E 01447      addlw   .14           ; ...then add 14 (2 groups) to pointer
02CB 1817 01448      btfsc   SHOWCOU,0      ; if bit 0 of SHOWCOU set...
02CC 3E07 01449      addlw   .7            ; ...then add 7 to pointer
02CD 0084 01450      movwf   FSR           ; FSR on (1st byte pos)-1 to display
01451
02CE 22BF 01452      call    Row1          ; move cursor to row 1
02CF 3007 01453      movlw   .7            ; bytes to display
02D0 008F 01454      movwf   SCRATCH       ; SCRATCH = byte display counter
02D1      01455      Hex7
02D1 0A84 01456      incf    FSR,F         ; adv pointer (it was x-1 at beginning)
02D2 0E00 01457      swapf  INDF,W         ; move hi nibble to display1. hex digit
02D3 23F8 01458      call   HexDigit       ; display 1st digit in hex mode
02D4 0800 01459      movf   INDF,W         ; move lo nibble to disp 2nd hex digit
02D5 23F8 01460      call   HexDigit       ; display 2nd digit in hex mode
02D6 23D7 01461      call   Blank          ; blank after hex number
01462
02D7 0B8F 01463      decfsz SCRATCH,F      ; SCRATCH = byte counter
02D8 2AD1 01464      goto   Hex7           ; loop if not yet 7 bytes
01465
02D9 0804 01466      movf   FSR,W          ; FSR = read pointer
02DA 3EF9 01467      addlw  -.7            ; restore it for ASCII mode printing
02DB 0084 01468      movwf  FSR            ; FSR on (1st byte pos)-1 to display
01469
02DC 23CE 01470      call   Row2           ; move cursor to row 2
02DD 3007 01471      movlw  .7             ; bytes to display
02DE 008F 01472      movwf  SCRATCH        ; SCRATCH = byte display counter
02DF      01473      Ascii7
02DF 0A84 01474      incf   FSR,F         ; adv pointer (it was x-1 at beginning)
02E0 0800 01475      movf   INDF,W         ; read byte
02E1 397F 01476      andlw  7fh           ; reduce ascii representation to 7 bits
01477
02E2 3EE0 01478      addlw  -20h          ; test if byte < 20h
02E3 3E20 01479      addlw  20h           ; restore previous value
02E4 1803 01480      btfsc  STATUS,C      ; C is set if byte < 20h
02E5 30A5 01481      movlw  0a5h          ; represent non-printables as dots
01482      ; (0a5h = dot)
02E6 23D8 01483      call   Char           ; display ascii char

```

```

01484
02E7 0B8F 01485      decfsz  SCRATCH,F      ; SCRATCH = byte counter
02E8 2ADF 01486      goto    Ascii7        ; loop if not yet 7 bytes
01487
02E9 28ED 01488      goto    Farm2         ; go back to user interface
01489
01490 ;*****
01491 ;* Mode2Go
01492 ;* This is entry point for Start command in mode 2 (serial code rcver)
01493 ;* Line 1 and first 7 positions (ASCII chars) of line 2 on LCD is clr.
01494 ;* Here, buffer is cleared and a sequence of 42 bytes are received and
01495 ;* written to buffer. Manual break (key 2) jumps to Break handling.
01496 ;* This protocol is used: High start bit, 7 or 8 data (true) bits, 0 or
01497 ;* 1 parity bit (not written to memory) and 1 low stop bit (not tested
01498 ;* for validity). If RXBITS,2 is set then the input is inverted.
01499 ;* Baud rates 1200-115200 are supported.
01500 ;* Note: no receive errors are detected nor indicated.
01501 ;*
01502 ;* Input variables:
01503 ;* RXRATE (range 0...7),which affects timing loaded via table BaudRate
01504 ;* RXBITS, bits 0-2 significant:bit0=parity,bit1=7/8 bits,bit2=inverse
01505 ;*
01506 ;* Output variables:
01507 ;* Buffer (42 bytes) loaded with bytes received, all unreceived bytes
01508 ;* represented as 00s.
01509 ;*****
02EA 01510 Mode2Go
02EA 1106 01511      bcf     PORTB,2      ; clear hi-imp probe output...
02EB 190D 01512      btfsc  RXBITS,2     ; ...let it be low if polarity bit clr
02EC 1506 01513      bsf     PORTB,2     ; set hi-imp output if polarity bit set
02ED 23B7 01514      call   ClrBuf       ; clear wholw buffer
02EE 302F 01515      movlw  .47          ; .47 blanks to clear displayed values
02EF 22B8 01516      call   SameAs20    ; clear displayed HEX and ASCII values
01517
02F0 3026 01518      movlw  BUFFER       ; start of buffer...
02F1 0084 01519      movwf  FSR          ; ...assigned to destination pointer
01520
02F2 081D 01521      movf   RXRATE,W     ; RXRATE = selected rate in range 0...7
02F3 3E68 01522      addlw  BaudRate     ; add to timing constant table offset
02F4 23B6 01523      call   PclSub       ; get rate to W
02F5 01524 RX42Bytes
02F5 008E 01525      movwf  DJNZ         ; baudrate timing factor to time cnter
01526
02F6 0194 01527      clrf   COUNT        ; this is to preset bit counter to 8...
02F7 1594 01528      bsf    COUNT,3     ; ...and not to disturb W
01529
02F8 1C0D 01530      btfss  RXBITS,0     ; RXBITS,0 is set if 7 bits selected
02F9 1C8D 01531      btfss  RXBITS,1     ; RXBITS,1 is set if parity bit select
02FA 0A94 01532      incf   COUNT,F     ; if not(RXBITS and 3 = 2) then COUNT=9
01533
02FB 180D 01534      btfsc  RXBITS,0     ; RXBITS,0 is set if 7 bits selected
02FC 188D 01535      btfsc  RXBITS,1     ; RXBITS,1 is set if parity bit select
02FD 0394 01536      decf   COUNT,F     ; ifnot(RXBITS and 3=1)reduce to 8 or 7
01537
02FE 1D0D 01538      btfss  RXBITS,2     ; RXBITS,2 set if inverse polar slctd
02FF 2B07 01539      goto   GetSp2       ; jump to true polarity if RXBITS,2 clr
01540
0300 01541 GetSp1
0300 1E05 01542      btfss  PORTA,4     ; test input status...
0301 2B00 01543      goto   GetSp1      ; ...and loop if still low
0302 01544 GetStart1
0302 1D85 01545      btfss  PORTA,3     ; test status of key 2...
0303 28E8 01546      goto   BrkRS       ; ...and jump to Break if presseed
0304 1A05 01547      btfsc  PORTA,4     ; test input status...
0305 2B02 01548      goto   GetStart1   ; ...and loop if still high
0306 2B0D 01549      goto   StartFound  ; falling edge detected: start receipt

```

```

01550
0307          01551 GetSp2                ; ---- true rx
0307 1A05          01552          btfsc   PORTA,4        ; test input status...
0308 2B07          01553          goto    GetSp2         ; ...and loop if still high
0309          01554 GetStart2
0309 1D85          01555          btfss   PORTA,3        ; test status of key 2...
030A 28E8          01556          goto    BrkRS         ; ...and jump to Break if presseed
030B 1E05          01557          btfss   PORTA,4        ; test input status...
030C 2B09          01558          goto    GetStart2     ; ...and loop if still low
01559          ; rising edge detected: start reception
030D          01560 StartFound
01561          ; 2-9 t from starting edge
030D          01562 HalfBit
030D 1686          01563          bsf     PORTB,5        ; 1*W led P on
030E 0000          01564          nop                    ; 1*W waist one cycle
030F 0B8E          01565          decfsz  DJNZ,F         ; 1*W DJNZ = timing constant counter
0310 2B0D          01566          goto    HalfBit      ; 2*W loop if not half bit timing passd
01567
0311          01568 RX8Bits
0311 008E          01569          movwf   DJNZ          ; 1      move timing constant to cnter
0312          01570 OneBit
0312 23A0          01571          call   Waist8T       ; 8 8 * W waist 8 t
0313 0B8E          01572          decfsz  DJNZ,F         ; 1 2 * W DJNZ=timing constant counter
0314 2B12          01573          goto    OneBit       ; 2 - * W loop if not 1 bit time passd
01574
0315 23A2          01575          call   Waist6T       ; 6      waist 6 t
01576
0316 0C85          01577          rrf     PORTA,F       ; 1 <--- move input status to C
0317 0C80          01578          rrf     INDF,F        ; 1      place C in input rotor
01579
0318 0B94          01580          decfsz  COUNT,F       ; 1      COUNT = bit counter
0319 2B11          01581          goto    RX8Bits      ; 2 total 22t + (w-1) * 11
01582          ;-----received byte @ INDF
031A 180D          01583          btfsc   RXBITS,0      ; test if parity bit selected...
031B 0D80          01584          rlf     INDF,F        ; ...and discard parity bit if received
01585
031C 1D0D          01586          btfss   RXBITS,2      ; test if inverse polarity selected...
031D 0980          01587          comf    INDF,F        ; ...and complement if true parity
01588
031E 1003          01589          bcf     STATUS,C      ; clear 8th bit for 7-bit mode
031F 188D          01590          btfsc   RXBITS,1      ; test if 7-bit mode selected and...
0320 0C80          01591          rrf     INDF,F        ; ..rotate 8th(zero) bit if 7bits slctd
01592
0321 0A84          01593          incf    FSR,F         ; advance destination pointer
0322 1B04          01594          btfsc   FSR,6         ; bits 4 and 6 will both be set if...
0323 1E04          01595          btfss   FSR,4         ; ... end of buffer+1 reached
0324 2AF5          01596          goto    RX42Bytes     ; loop if not yet FSR=50h
0325 2AC6          01597          goto    Show2         ; over: go show received bytes
01598
01599 ;***** FREQUENCY COUNTER
01600 ;*****
01601 ;* GoPresc
01602 ;* Prescaler factor (variable PRESC, in range 0...3) is advanced
01603 ;* (executes when key 2 is pressed in frequency counter mode)
01604 ;*****
0326          01605 GoPresc
0326 0A9A          01606          incf    PRESC,F       ; advance prescaler
0327 111A          01607          bcf     PRESC,2       ; and cycle prescaler in range 0...3
01608
01609 ;*****
01610 ;* FreqEp
01611 ;* Frequency counter entry point.
01612 ;* Subroutine WrParam does this: Displays message "Frequency" in row 1,
01613 ;* counter range (taken from table RangeTab) & resolution (taken from
01614 ;* table PrescTab) in row2.
01615 ;* LEDs are turned OFF, and the main counter (BIN4, 4 bytes) cleared.

```

```

01616 ;* The following is used to count pulses:
01617 ;* State of TMR0 is written to BIN4+0 and sequentially tested, and when
01618 ;* bit7 of current value detected as 0 and the previous one was 1, the
01619 ;* overflow is considered. In that case, state of BIN4+1 is advanced,
01620 ;* extended to BIN4+2. After 500ms, the 32-bit value of BIN4 is shifted
01621 ;* left in a total of PRESC+2 times, to get multiply by 4,8,16 or
01622 ;* 32. Then BIN4 (4 bytes) is converted to ASCII and printed on LCD.
01623 ;* This routine does not call keyboard routine, as accurate timing of
01624 ;* 500 ms must be generated for counting (high count register is
01625 ;* CHARCOU lo count SHOWCOU). Instead of this, there is the individual
01626 ;* routine for key 1 and key 2 test, and also the countdown timer for
01627 ;* automatic power-off (registers TIMOUTL, TIMOUTH).
01628 ;* If key 1 is pressed, mode 1 (analyzer) is entered. If key 2 is
01629 ;* pressed, prescaler value is advanced.
01630 ;*
01631 ;* Note: code from label "Loop500A" to comment "; 500 ms timeout" is
01632 ;* real time code. If the # of cycles is changed, then the literals
01633 ;* 0f4h+1 and 24h+1 written to CHARCOU and SHOWCOU must be readjusted.
01634 ;*
01635 ;* Input variables: PRESC (affects prescaler factor)
01636 ;* Output variables: none
01637 ;*****
0328          01638 FreqEp          ; mode 3: frequency counter
0328 2364      01639          call    WrParam      ; print "Frequency" and "xxMHz/Rxx"
0329 21BF      01640          call    KeysOff     ; test both keys off for 34 ms and
01641          ; initialize 8 min auto off sequence
032A          01642 Count500
032A 30D3      01643          movlw   0d2h+REL    ; right arrow position
032B 23CF      01644          call    WrComL     ; move cursor on right arrow
01645
032C 019F      01646          clrfl  BIN4+1     ; clear next counter byte
032D 01A0      01647          clrfl  BIN4+2     ; clear next counter byte
032E 01A1      01648          clrfl  BIN4+3     ; clear next counter byte
032F 018F      01649          clrfl  SCRATCH    ; initialize TMR0 overflow detector
01650
0330 30F5      01651          movlw   0f4h+1     ; high loop counter for 500 ms
0331 0096      01652          movwf  CHARCOU    ; CHARCOU = hi byte counter
0332 3025      01653          movlw   24h+1     ; 0f424h=.62500 cycles=1250000,T=500 ms
0333 0097      01654          movwf  SHOWCOU    ; SHOWCOU = lo byte counter
01655
0334 081A      01656          movf    PRESC,W     ; PRESC = prescaler factor selected
0335 3E20      01657          addlw  20h         ; for PRESC 0,1,2,3 w=20h,21h,22h,23h
0336 2278      01658          call   ToOption    ; here is clrfl TMR0 also
0337          01659 Loop500A
0337 1D85      01660          btfss  PORTA,3     ; 2 test key 2 status and
0338 2B26      01661          goto   GoPresc    ; - jump to "prescaler change" if hit
0339          01662 Loop500
0339 0801      01663          movf    TMR0,W     ; 1 1 1 only place where TMR0 is read
033A 009E      01664          movwf  BIN4        ; 1 1 1 rtcc ---> freq0
01665
033B 0D1E      01666          rlf    BIN4,W     ; 1 1 1 carry <--- TMR0.7
033C 0D8F      01667          rlf    SCRATCH,F  ; 1 1 1 rotor <--- carry
01668
033D 080F      01669          movf    SCRATCH,W  ; 1 1 1 SCRATCH=TMR0 overflow detect
033E 3903      01670          andlw  3          ; 1 1 1 mask 2 LSbs for edge detect
033F 3A02      01671          xorlw  2          ; 1 1 1 00000000 if 1 <--- 0
01672
0340 1D03      01673          btfss  STATUS,Z   ; 1 2 2 | skip if TMR0 overflow
0341 2B43      01674          goto   NotOvf1    ; 2 - - | if nz
01675          ; | 5T any case
0342 0A9F      01676          incf   BIN4+1,F   ; - 1 1 | nsb
0343 1903      01677 NotOvf1 btfsc  STATUS,Z   ; 2 2 1 | skip MSB adv ifnot overflow
0344 0AA0      01678          incf   BIN4+2,F   ; - - 1 | msb
01679
0345 309B      01680          movlw  Head4-1    ; 1 initialize "Battery" message
0346 1D05      01681          btfss  PORTA,2    ; 2 test key 1 status and...

```

```

0347 2933      01682      goto      Mode4          ; - got shortcut to mode 4 if pressed
                01683
0348 0B97      01684      decfsz    SHOWCOU,F      ; 1 (2)  lo loop counter
0349 2B37      01685      goto      Loop500A      ; 2 (-)  20T total
                01686
034A 0B96      01687      decfsz    CHARCOU,F     ; (1)   hi loop counter
034B 2B39      01688      goto      Loop500       ; (2)   20T total
                01689
                01690      ; --- 500 ms timeout here
034C 0A1A      01691      incf      PRESC,W       ; prepare for x2 multiply: PRESC incr
034D 0096      01692      movwf    CHARCOU       ; CHARCOU = multiply factor counter
034E 0A96      01693      incf      CHARCOU,F     ; and prescaler constant incr again
034F          01694      ShLoop    ; BIN4 = BIN4 * 2 total (PRESC+2) times
034F 1003      01695      bcf      STATUS,C      ; clear C to allow clean x2 multiply
0350 0D9E      01696      rlf      BIN4, F       ; low byte x2 multiply
0351 0D9F      01697      rlf      BIN4+1,F     ; next byte x2 multiply
0352 0DA0      01698      rlf      BIN4+2,F     ; next byte x2 multiply
0353 0DA1      01699      rlf      BIN4+3,F     ; highest byte x2 multiply
                01700
0354 0B96      01701      decfsz    CHARCOU,F     ; CHARCOU = multiply factor counter
0355 2B4F      01702      goto      ShLoop       ; loop if not PRESC*2 times multiplied
                01703
0356 308A      01704      movlw    8ah           ; frequency display position
0357 23CF      01705      call     WrComL        ; cursor to freq display position
0358 238C      01706      call     Print8        ; print the frequency in 8-digit ASCII
                01707
0359 0B9B      01708      decfsz    TIMOUTL,F    ; TIMOUTL=lo byte auto power off cnter
035A 2B2A      01709      goto      Count500     ; inner loop
035B 151B      01710      bsf      TIMOUTL,2     ; TIMOUTL is init to 4 passes instead
                01711      ; of 256, 4*256*500ms=512s=8.5min appr
                01712
035C 0B9C      01713      decfsz    TIMOUTH,F   ; TIMOUTH=hi byte auto power off cnter
035D 2B2A      01714      goto      Count500     ; loop if not yet 8.5 min
                01715
035E 29BC      01716      goto      Suicide      ; 8.5 min timeout - go switch power off
                01717
01718 ;*****  BINARY TO ASCII CONVERSION
01719 ;*****
01720 ;*  Headline
01721 ;*  Clears SHOWCOU (Headline2 skips this), Clears LCD, prints right
01722 ;*  arrow @ last pos of row2, prints message addressed by W+1 at page 0.
01723 ;*  Terminator is last character with bit 7 set.
01724 ;*
01725 ;*  Input variables: W+1 addresses string (on page 0) to be printed
01726 ;*  Output variables:
01727 ;*    CHARCOU is decremented by the number of characters printed
01728 ;*****
035F          01729      Headline
035F 0197      01730      clrf     SHOWCOU      ; initialize show group counter
0360          01731      Headline2
0360 008F      01732      movwf    SCRATCH      ; move input parameter to SCARTCH
0361 303B      01733      movlw    .59          ; .59 characters to clear
0362 22B8      01734      call     SameAs20     ; clear all but right arrow
0363 2BDE      01735      goto     GoWrite      ; print headline message on LCD
                01736
01737 ;*****
01738 ;*  WrParam
01739 ;*  Prints message "Frequency" in line 1 and parameters for frequency
01740 ;*  counter in line 2.
01741 ;*  Text XXMHz/RYY is printed, where XX is taken from RangeTab, and YY
01742 ;*  from PrescTab.
01743 ;*
01744 ;*  Input variables:
01745 ;*    PRESC, affects displayed frequency range and resolution
01746 ;*  Output variables:
01747 ;*    CHARCOU is decremented by the number of characters printed

```

```

01748 ;*****
01749 ;* Print255
01750 ;* Entry point Print255 converts 8-bit binary value (<100) in BIN4 to
01751 ;* 2-digit ASCII and prints it on LCD, without decimal point. Leading
01752 ;* zeros are printed.
01753 ;*
01754 ;* Input variables: W, binary number (0-99) to be converted and printed
01755 ;* Output variables:
01756 ;* CHARCOU is decremented by the number of characters printed
01757 ;*****
01758 ;* Print3
01759 ;* Entry point Print3 converts 16-bit binary value (<1000) in BIN4 to 2-
01760 ;* or 3-digit ASCII and prints it on LCD. Leading zero is skipped only
01761 ;* if value is <100. Decimal point is printed between tens and ones if
01762 ;* FLAG,DP is set, otherwise decimal point is omitted.
01763 ;*
01764 ;* Input variables: BIN4 (2 bytes, LSB first, in range 0-999) binary
01765 ;* number to be converted and printed
01766 ;* Output variables:
01767 ;* CHARCOU is decremented by the number of characters printed
01768 ;*****
01769 ;* PrintBR
01770 ;* Entry point PrintBR does the same as PRINT3, but the low byte value
01771 ;* is in W instead in BIN4+0. This is used for baud rate display.
01772 ;*
01773 ;* Input variables: W, BIN+1 (2 bytes, LSB in W, MSB in BIN+1, in range
01774 ;* 0-999) binary number to be converted and printed
01775 ;* Output variables:
01776 ;* CHARCOU is decremented by the number of characters printed
01777 ;*****
0364 01778 WrParam ; print xxMHz/Rxx
0364 3092 01779 movlw Head3-1 ; address of message "Frequency"
0365 2360 01780 call Headline2 ; print message
0366 23CE 01781 call Row2 ; move cursor to row 2
01782
0367 3064 01783 movlw RangeTab ; offset of max frequency range table
0368 236E 01784 call Presc255 ; print max frequency range
01785
0369 306F 01786 movlw TxtHz-1 ; address of message "MHz/"
036A 23DD 01787 call Write ; print message
036B 3052 01788 movlw 'R' ; "R" stands for "Resolution"
036C 23D8 01789 call Char ; print "R"
01790
036D 3060 01791 movlw PrescTab ; offset of resolution table
036E 01792 Presc255
036E 071A 01793 addwf PRESC,W ; add PRESC to offset
036F 23B6 01794 call PclSub ; read value from table
0370 01795 Print255
0370 108C 01796 bcf FLAG,DP ; clear decimal point enable flag
0371 019F 01797 clrf BIN4+1 ; clear hi byte (allow range 00-99)
0372 01798 PrintBR
0372 009E 01799 movwf BIN4 ; allow W as input parameter
0373 01800 Print3
0373 01A1 01801 clrf BIN4+3 ; clear hi byte
0374 01A0 01802 clrf BIN4+2 ; clear next byte
0375 01A5 01803 clrf CMP4+3 ; clear hi byte of temporary register
0376 01A4 01804 clrf CMP4+2 ; clear next byte of temporary register
0377 01A3 01805 clrf CMP4+1 ; clear next byte of temporary register
01806
0378 3064 01807 movlw .100 ; first digit constant
0379 2383 01808 call Times ; # of times goes in BIN4+1 and BIN4?
037A 1D03 01809 btfsz STATUS,Z ; skip printing if it goes zero times
037B 23C9 01810 call Num ; print digit (hundreds) if W>0
01811
037C 300A 01812 movlw .10 ; second digit constant
037D 2383 01813 call Times ; how many times it goes in BIN4?

```

```

037E 23C9      01814      call      Num          ; print digit (tens)
                01815
037F 302E      01816      movlw    '.'          ; decimal point
0380 188C      01817      btfsc   FLAG,DP      ; test decimal pnt flag, skip if reset
0381 23D8      01818      call    Char         ; print decimal point if DP set
0382 2B9B      01819      goto    NumBin4      ; print last digit (ones)
                01820
01821 ;*****
01822 ;* Times
01823 ;* Counts # of times CMP4 (32-bit value) "goes" in BIN4 (32-bit value).
01824 ;* BIN4 is sequentially subtracted by CMP4 and counter COUNT advanced.
01825 ;* When borrow is detected, BIN4 is restored to the last positiv value
01826 ;* (by ADDING CMP4 again), COUNTer decremented and written to W.
01827 ;*
01828 ;* Input variables: CMP4 (32-bit value), BIN4 (32-bit value)
01829 ;* Output variables:
01830 ;*   BIN4 (32-bit value) modified to mod(CMP4)
01831 ;*   W (in range 0...9) = BIN4 (32-bit value) / CMP4 (32-bit value)
01832 ;*****
0383           01833 Times
0383 00A2      01834      movwf   CMP4         ; place input param in CMP4 to compare
0384 0194      01835      clrf   COUNT        ; clear result counter
0385           01836 GoTD
0385 0A94      01837      incf   COUNT,F      ; advance result counter
0386 239D      01838      call   Sub4         ; BIN4=BIN4-CMP4  nc if result <0
0387 1803      01839      btfsc  STATUS,C     ; test did it "go"?
0388 2B85      01840      goto   GoTD         ; loop if so
0389 23A5      01841      call   Add4         ; BIN4=BIN4+CMP4  c set if ovf
038A 0314      01842      decf   COUNT,W     ; W=# of times CMP4 goes in BIN4(32bit)
038B 0008      01843      return          ; result in W
                01844
01845 ;*****
01846 ;* Print8
01847 ;* This subroutine converts 32-bit value in BIN4 (low byte first), to
01848 ;* 8-dig ASCII and prints to LCD. Leading zeros are printed as blanks.
01849 ;* Table DecTab (21 words, must be at page 0 if PCLATH=0) used in conv.
01850 ;*
01851 ;* Input variables: BIN4 (32-bit value, <.100,000,000)
01852 ;* Output variables:
01853 ;*   CHARCOU is decremented by the number of characters printed
01854 ;*****
038C           01855 Print8
038C 3033      01856      movlw  DecTab-1     ; inici tab ptr
038D 008F      01857      movwf  SCRATCH     ; SCRATCH = tab ptr
038E 118C      01858      bcf    FLAG,RIPPLE ; zeros initialy print as blanks,until
01859 ; ...first non-zero appears
038F           01860 Cif7
038F 01A5      01861      clrf   CMP4+3     ; clear CMP4+3, it is =0 in all cases
0390 23B4      01862      call   PclSub2     ; get constant from table
0391 00A4      01863      movwf  CMP4+2     ; load dec. const from table in CMP4+2
0392 23B4      01864      call   PclSub2     ; get constant from table
0393 00A3      01865      movwf  CMP4+1     ; load dec. const from table in CMP4+1
0394 23B4      01866      call   PclSub2     ; get constant from table
0395 2383      01867      call   Times       ; how many times CMP4 goes in BIN4?
0396 23C3      01868      call   NZNum       ; print if w>0 or RIPPLE=1, else blank
                01869
0397 080F      01870      movf   SCRATCH,W   ; SCRATCH = table pointer
0398 3EB9      01871      addlw  .237-DecTab ; test if end of table
0399 1C03      01872      btfss  STATUS,C   ; C set if end of table
039A 2B8F      01873      goto   Cif7       ; if not end of table loop (will be 7x)
039B           01874 NumBin4
039B 081E      01875      movf   BIN4,W     ; last digit is in BIN4
039C 2BC9      01876      goto   Num        ; last digit must be printed always
                01877
01878 ;*****
01879 ;* Sub4

```

```

01880 ;* Subtract CMP4 (32-byte value) from 32-bit value in BIN4, lo byte 1st
01881 ;* This is performed as adding of negative value of CMP4. Negating is
01882 ;* performed as complementing and incrementing by 1.
01883 ;* Note: Incrementing by 1 is performed on least significant byte only,
01884 ;* without 32-bit extension, for code space saving. This will not cause
01885 ;* error in this case, as the number of all possible values for CMP4+0
01886 ;* is limited and none of them is equal to 0FFh before incrementing
01887 ;* (all possible values are taken from table DecTab, & are: 0ah, 64h,
01888 ;* 0e8h, 10h, 0a0h, 40h and 80h, and their negative values).
01889 ;* However, this is valid if this subroutine is used for decimal
01890 ;* conversion only, and if it is used for some other application,
01891 ;* extension to 32-bit should be added after incrementing.
01892 ;*
01893 ;* Input variables: BIN4 (32-bit value), CMP4 (32-bit value)
01894 ;* Output variables:
01895 ;*     BIN4 (32-bit value)
01896 ;*     STATUS,C denotes the sign of result: if cleared, output value
01897 ;*         is negative (there is borrow)
01898 ;*
01899 ;* Note: Entry points Waist8T and Waist6T are used only by some
01900 ;* real-time routines, in that case the instructions are dummy
01901 ;*****
039D      01902 Sub4                ; 32-bit sub: BIN4 = BIN4 - CMP4
01903                ; NC if result<0
039D 239F  01904      call    NegCmp        ; negate CMP (32 bits) first
039E 23A5  01905      call    Add4          ; add as negative value
039F      01906 NegCmp
039F 09A2  01907      comf    CMP4+0,F        ; complement low byte
03A0      01908 Waist8T
03A0 09A3  01909      comf    CMP4+1,F        ; complement next byte
03A1 09A4  01910      comf    CMP4+2,F        ; complement next byte
03A2      01911 Waist6T
03A2 09A5  01912      comf    CMP4+3,F        ; complement high byte
03A3 0AA2  01913      incf    CMP4+0,F        ; neg = complement + 1
01914                ; (no need test overflow here, it will
01915                ; ...never reach 0 after incrementing)
03A4 0008  01916      return                ; finished
01917
01918 ;*****
01919 ;* Add4
01920 ;* Add CMP4 (32-byte value) to BIN4 (32-byte value). 4-instr. groups
01921 ;* (movf-btfdc-incfsz-addwf) used instead of non-existing ADD W/ CARRY.
01922 ;* Input variables: BIN4 (32-bit value), CMP4 (32-bit value)
01923 ;* Output variables: BIN4 (32-bit value), 33th bit in STATUS,C
01924 ;*****
03A5      01925 Add4                ; 32-bit add: BIN4 = BIN4 + CMP4
03A5 0822  01926      movf    CMP4,W            ; low byte
03A6 079E  01927      addwf   BIN4,F            ; low byte add
01928
03A7 0823  01929      movf    CMP4+1,W          ; next byte
03A8 1803  01930      btfdc   STATUS,C          ; skip to simple add if C was reset
03A9 0F23  01931      incfsz  CMP4+1,W          ; add C if it was set
03AA 079F  01932      addwf   BIN4+1,F          ; next byte add if NZ
01933
03AB 0824  01934      movf    CMP4+2,W          ; next byte
03AC 1803  01935      btfdc   STATUS,C          ; skip to simple add if C was reset
03AD 0F24  01936      incfsz  CMP4+2,W          ; add C if it was set
03AE 07A0  01937      addwf   BIN4+2,F          ; next byte add if NZ
01938
03AF 0825  01939      movf    CMP4+3,W          ; high byte
03B0 1803  01940      btfdc   STATUS,C          ; skip to simple add if C was reset
03B1 0F25  01941      incfsz  CMP4+3,W          ; add C if it was set
03B2 07A1  01942      addwf   BIN4+3,F          ; high byte add if NZ
01943
03B3 0008  01944      return                ; finished
01945

```

```

01946 ;*****
01947 ;* PclSub is used for indirect addressing
01948 ;* PclSub1 uses SCRATCH instead of W as input parameter
01949 ;* PclSub2 advances pointer SCRATCH before executong
01950 ;*
01951 ;* Note: PCLATH=0 in all cases. So all tables pointed by this routine
01952 ;* are on page 0
01953 ;*****
03B4 01954 PclSub2
03B4 0A8F 01955      incf    SCRATCH,F      ; advance table pointer
03B5 01956 PclSub1
03B5 080F 01957      movf    SCRATCH,W      ; move table pointer to W
03B6 01958 PclSub
03B6 0082 01959      movwf   PCL            ; jump to address pointed by PCLATH,W
01960
01961 ;*****
01962 ;* ClrBuf
01963 ;* ClrRam
01964 ;* Subroutine ClrBuf clears BUFFER (42 bytes)
01965 ;* Entry point ClrRam allows some other start point for claring. It
01966 ;* clears internal RAM from address in W to the location 7Fh.
01967 ;* (locations 50h-7Fh, which do not exist in 16F84, are dummy).
01968 ;*
01969 ;* Both entry points continue to disabling Enable signal for LCD and
01970 ;* 33.8 ms timing loop
01971 ;*
01972 ;* Input variables:
01973 ;* W is the start addr if area to be cleared (ClrRam entry point only)
01974 ;* Output variables: none
01975 ;*****
03B7 01976 ClrBuf
03B7 3026 01977      movlw   BUFFER          ; get start address of buffer
03B8 01978 ClrRam
03B8 0084 01979      movwf   FSR            ; FSR = dest pointer for clearing
03B9 01980 Zeros
03B9 0180 01981      clrf    INDF           ; clear one byte
03BA 0A84 01982      incf    FSR,F          ; advance dest pointer
03BB 1F84 01983      btfsz   FSR,7          ; test if end of RAM...
03BC 2BB9 01984      goto    Zeros         ; ...if not, loop - else move LEDs
01985
01986 ;*****
01987 ;* Entry point DisEna30: Remove enable and discharge signal, and
01988 ;* refresh LEDs. Then loop 33.8 ms
01989 ;* Entry point Wait30: Loop 33.8 ms
01990 ;* Input variables: none
01991 ;* Output variables: none
01992 ;*****
03BD 01993 DisEna30
03BD 23EF 01994      call    DisEna         ; disable discharging output signal
03BE 01995 Wait30
03BE 0194 01996      clrf    COUNT        ; COUNT=time loop cnter,to wait 33.8ms
03BF 01997 GoWait30
03BF 23D0 01998      call    loop130       ; waist 130 us
03C0 0B94 01999      decfsz  COUNT,F       ; COUNT = time loop counter
03C1 2BBF 02000      goto    GoWait30      ; loop if not yet 256 passes
03C2 0008 02001      return   ; timing over
02002
02003 ;*****
02004 ;* NZNum
02005 ;* Num
02006 ;* Print numeric value in W (in range 0...9) on LCD. If FLAG,RIPPLE is
02007 ;* cleared, 0 is printed as blank. If non-zero numeric is printed, it
02008 ;* automatically sets FLAG,RIPPLE.
02009 ;* 0 (30h) prints as capital O (4Fh), for improved readability, as 0
02010 ;* may easily be substituted by 8 on LCD. This changing 0 to O is not
02011 ;* performed only in ASCII representation of recorded bytes in serial

```

```

02012 ;* code receiver.
02013 ;* Entry point NUM prints numeric unconditionally, undependently of bit
02014 ;* FLAG,RIPPLE.
02015 ;*
02016 ;* Input variables:
02017 ;* W (0...9), number to be printed at current cursor position of LCD
02018 ;* Output variables:
02019 ;* CHARCOU is decremented by the number of characters printed
02020 ;*****
03C3 02021 NZNum ; same aso Num, only blank instead of 0
03C3 198C 02022 btfsc FLAG,RIPPLE ; test if RIPPLE bit set...
03C4 2BC9 02023 goto Num ; ...if RIPPLE set, no more blanks
03C5 3E00 02024 addlw 0 ; set Z flag if W=0
03C6 1903 02025 btfsc STATUS,Z ; is it = 0 ?
03C7 2BD7 02026 goto Blank ; if so, jump to space routine
03C8 158C 02027 bsf FLAG,RIPPLE ; if>0,clr RIPPLE bit, no more blanks
03C9 02028 Num
03C9 390F 02029 andlw 0fh ; isolate low nibble
03CA 1903 02030 btfsc STATUS,Z ; is it = 0 ?
03CB 301F 02031 movlw '0'-30h ; if so, initialize capital 0
03CC 3E30 02032 addlw 30h ; adjust ASCII for numeric
03CD 2BD8 02033 goto Char ; print digit
02034
02035 ;***** LCD ROUTINES
02036 ;*****
02037 ;* All these entry ponts of this subroutine are used in program:
02038 ;*
02039 ;* Row2: Issues command to move cursor to row2 of LCD,and loops 130
02040 ;* us,to allow time for LCD controller to execute command
02041 ;* WrCom1: Issues command in W to the LCD, and loops 130 us, to allow
02042 ;* time to LCD controller to execute the command
02043 ;* loop130: Loops 130 us including call and return
02044 ;* GoLoop: Loops W*2 us
02045 ;* Loop7: Same as GoLoop, only 2t shorter (for smpl rate routine)
02046 ;*****
03CE 02047 Row2
03CE 30C0 02048 movlw 0c0h ; command for line 2
02049
03CF 02050 WrComL ; issues command in W
03CF 23E5 02051 call WrCom ; write command in LCD
03D0 02052 loop130 ; * waist 130 us
03D0 018E 02053 clrf DJNZ ; this is to init DJNZ to 40h and...
03D1 170E 02054 bsf DJNZ,6 ; not to disturb W
03D2 02055 GoLoop
03D2 2BD3 02056 goto $+1 ; 2 waist 2 t
03D3 02057 Loop7
03D3 0B8E 02058 decfsz DJNZ,F ; 1 DJNZ = timing loop counter
03D4 2BD2 02059 goto GoLoop ; 2 64x5=320t (128 us)
03D5 0008 02060 return ; 2 finished
02061
02062 ;*****
02063 ;* All these entry ponts of this subroutine are used in program:
02064 ;*
02065 ;* CharBl: print character in W, blank on LCD and decrement CHARCOU
02066 ;* Blank: print blank (32h) on LCD and decrement CHARCOU
02067 ;* Char: print character in W on LCD and decrement CHARCOU
02068 ;* CharNCC: print character in W on LCD without affecting CHARCOU
02069 ;*
02070 ;* Note: CHARCOU is used to print fixed format message on LCD, as the
02071 ;* calling routine will add N-CHARCOU blanks to fill area N chars long
02072 ;*
02073 ;* Input variables:
02074 ;* all entry points except Blank: W = character to be printed
02075 ;* Output variables:
02076 ;* all entry points except CharNCC: CHARCOU is decremented by 1
02077 ;*****

```

```

03D6          02078 CharBl          ; print char, then blank
03D6 23D8     02079          call   Char          ; print char first
03D7          02080 Blank          ; print blank
03D7 3020     02081          movlw  ' '          ; print blank
03D8          02082 Char          ; print W
03D8 0396     02083          decf   CHARCOU,F      ; decrement character counter
03D9          02084 CharNCC        ; print W without affecting CHARCOU
03D9 1486     02085          bsf   PORTB,1      ; pull RS hi (data register select)
03DA 2BE6     02086          goto   Skrl          ; continue 4-bit mode writing to LCD
02087
02088 ;*****
02089 ;* PrintBrk
02090 ;* Print message "Break" in row 1, pos 0
02091 ;*
02092 ;* Input variables: none
02093 ;* Output variables:
02094 ;* CHARCOU is decremented by the number of characters printed
02095 ;*****
02096 ;* Write
02097 ;* Print message addressed by W+1 in line 1
02098 ;* Note: Terminator is last character in string with bit 7 set
02099 ;*
02100 ;* Input variables: W points to string (RETLWs) decremented by 1
02101 ;* Output variables:
02102 ;* CHARCOU is decremented by the number of characters printed
02103 ;*****
03DB          02104 PrintBrk
03DB 22BF     02105          call   Row1          ; move cursor to row 1
03DC 30A2     02106          movlw  BrkMes-1      ; message "Break" address-1
02107
03DD          02108 Write          ; write string addressed by W, end w/ 0
03DD 008F     02109          movwf  SCRATCH       ; SCRATCH = source pointer
03DE          02110 GoWrite        ; write string at (SCRATCH+1), wnd w/ 0
03DE 23B4     02111          call   PclSub2       ; advance pointer and read pointed byte
03DF 3E80     02112          addlw  80h           ; this is to test if bit 7 was set...
03E0 1803     02113          btfsc STATUS,C      ; ...if so, C will be set
03E1 2BD8     02114          goto   Char          ; last character was with bit 7 set
03E2 397F     02115          andlw  7fh          ; restore initial character value
03E3 23D8     02116          call   Char          ; print one character
03E4 2BDE     02117          goto   GoWrite       ; loop
02118
02119 ;*****
02120 ;* Wrcom
02121 ;* Write command in W to LCD, then loop 130 us
02122 ;* Skrl
02123 ;* Allows data write to LCD, if PORTB,1 is set previously
02124 ;* Nibble
02125 ;* Write one nibble (W,0-3) to LCD data bus
02126 ;*
02127 ;* Note: all entry ponits are terminated by 130us timing loop, to allow
02128 ;* LCD controller to execute accepted command/data.
02129 ;*
02130 ;* Input variables: command in W
02131 ;* Output variables: none
02132 ;*****
03E5          02133 WrCom
03E5 1086     02134          bcf   PORTB,1      ; rs lo (command)
03E6          02135 Skrl
03E6 008E     02136          movwf  DJNZ          ; save W in DJNZ for lo nibble writing
03E7 23F1     02137          call   Hinib_B      ; outputs w,4-7 to PORTB,4-7
03E8 23ED     02138          call   EnaLCD        ; generate enable signal for hi nibble
03E9          02139 Nibble
03E9 0E0E     02140          swapf DJNZ,W        ; restore init value of W and swap it
03EA 23F1     02141          call   Hinib_B      ; outputs w,0-3 to PORTB,4-7
03EB 23ED     02142          call   EnaLCD        ; generate enable signal for low nibble
03EC 2BD0     02143          goto   loop130      ; wait 130 for LCD to crunch command

```

```

02144
02145 ;*****
02146 ;* Generate Enable signal (1200us) for LCD controller,and refresh LEDs.
02147 ;* Entry point DisEna: Remove enable & dischg signal,and refresh LEDs.
02148 ;*****
03ED 02149 EnaLCD
03ED 1406 02150      bsf      PORTB,0      ; enable LCD controller
03EE 2BEF 02151      goto    $+1          ; wait 2 cycles,make signal 1.2us long
03EF 02152 DisEna
03EF 1006 02153      bcf      PORTB,0      ; terminate enable signal LCD control
02154
02155 ;*****
02156 ;* MoveLEDs
02157 ;* Entry point MoveLEDs: transfer FLAG,LEDP FLAG,LEDH and FLAG,LEDL to
02158 ;* PORTB,5 PORTB,6 and PORTB,7 to service LEDs.
02159 ;*
02160 ;* Input variables:
02161 ;* FLAG, bits LEDP, LEDH, LEDH will affect LED1, LED2, LED3
02162 ;* Output variables: none
02163 ;*****
02164 ;* Hinib_B
02165 ;* Entry point Hinib_B: output W,4-7 to 4-bit LCD data bus
02166 ;*
02167 ;* Input variables:
02168 ;* hi nibble of W is copied to LCD data bus
02169 ;* Output variables: none
02170 ;*****
03F0 02171 MoveLESD      ; writes states LEDs L,H,P to PORTB,5-7
03F0 080C 02172      movf     FLAG,W      ; FLAG bits 5,6,7 are LED bits
03F1 02173 Hinib_B      ; outputs w,4-7 to PORTB,4-7
03F1 1206 02174      bcf     PORTB,4      ; clear high nibble of PORTB
03F2 1286 02175      bcf     PORTB,5      ; clear high nibble of PORTB
03F3 1306 02176      bcf     PORTB,6      ; clear high nibble of PORTB
03F4 1386 02177      bcf     PORTB,7      ; clear high nibble of PORTB
03F5 39F0 02178      andlw   0f0h      ; mask for hi nibble of W
03F6 0486 02179      iorwf   PORTB,F      ; write H nibble W to H nibble PORTB
03F7 0008 02180      return    ; finished
02181
02182 ;*****
02183 ;* HexDigit
02184 ;* This subroutine prints low nibble of W on LCD as hexadecimal digit.
02185 ;* Zero (30h) is printed as capital O (7Fh)
02186 ;*
02187 ;* Input variables: W in range 0..0fh, hex number to be printed
02188 ;* Output variables: CHARCOU is decremented by two
02189 ;*****
03F8 02190 HexDigit      ; hex W (lo nibble) to LCD, change 0...
02191      ; ...to capital O
03F8 390F 02192      andlw   0fh      ; isolate low nibble of W...
03F9 009E 02193      movwf   BIN4      ; ...and put it in BIN4
03FA 3EF6 02194      addlw   -0ah      ; test if input number > 9
03FB 1C03 02195      btfs    STATUS,C      ; Is it > 9 ?
03FC 2B9B 02196      goto    NumBin4      ; ..if not, just print it as-is
03FD 3E41 02197      addlw   .7+3ah      ; 3ah...3fh to 'A'...'F' correction
03FE 2BD8 02198      goto    Char        ; print ASCII adjusted hex value a...f
02199
02200 ;***** DATA EEPROM
02201 ;*****
02202 ;* This table is located in data eeprom
02203 ;* It contains numerical data for 16 sample frequencies period display
02204 ;* for analyzer. Last 13 bytes are timing constants used by subroutine
02205 ;* GetSlowClk to generate internal timing (three fastest rates need no
02206 ;* constants from the table, as they are treated as special cases)
02207 ;*****
02208 ;* ----- TABLE 1 (00h-2Fh): Rate display table for analyzer
02209 ;*

```

```

02210 ;* ****First byte: Flags. Bits in this byte have the following functs:
02211 ;* bit 7 = 0: Frequency in Hz
02212 ;*          = 1: Frequency in Mhz or Khz
02213 ;* bit 6 = 0: Frequency does not contain decimal point
02214 ;*          = 1: Frequency contains decimal point before last digit
02215 ;* bits 5,4: Bits 9 and 8 for frequency display, respectively
02216 ;* bit 3 = 0: Period in us (microseconds)
02217 ;*          = 1: Period in ms (miliseconds)
02218 ;* bit 2 = 0: Period does not contain decimal point
02219 ;*          = 1: Period contains decimal point before last digit
02220 ;* bits 1,0: Bits 9 and 8 for period display, respectively
02221 ;* **** Second byte: low significant byte for frequency
02222 ;* **** Third byte: low significant byte for period
02223 ;*****
02224 ;* ----- TABLE 2 (30h-3Ch): Timing constant table for analyzer
02225 ;*
02226 ;* Timing constant factors to all sample rates generated by subroutine
02227 ;* GetSlowClk (all except 1MHz, 500KHz and 228KHz)
02228 ;*****
02229 ;* Note: This is read-only data, so the Data EEPROM must be programmed
02230 ;* before the unit is ready to use. MCU will not affect data EEPROM
02231 ;* contents. If your programmer does not support automatic loading of
02232 ;* Data EEPROM contents from the HEX file, it must be loaded manually.
02233 ;* This will help in that case (all values are hexadecimal):
02234 ;*
02235 ;* addr 00-07: 88 01 01 98 F4 02 8C E4
02236 ;* addr 08-0f: 2C 88 64 0A 88 32 14 D8
02237 ;* addr 10-17: 80 1A 88 19 28 C8 C0 34
02238 ;* addr 18-1f: 88 0A 64 C8 60 68 C8 30
02239 ;* addr 20-27: D0 C9 18 A1 80 01 01 14
02240 ;* addr 28-2f: 90 19 00 64 0A 00 28 19
02241 ;* addr 30-37: 01 06 09 10 16 2E 30 64
02242 ;* addr 38-3c: CC 05 0E 3B 95
02243 ;*
02244 ;* Total bytes used in Data EEPROM: 61 (the last 3 bytes don't care)
02245 ;*****
2100 02246          org          2100h
02247 ;
02248          constant T/sample Hz      s      RATE
2100 0088 0001 0001 02249          de      b'10001000', .1, .1 ; -      2.5      1M      1u      0
2103 0098 00F4 0002 02250          de      b'10011000', .244, .2 ; -      5      500K      2u      1
2106 008C 00E4 002C 02251          de      b'10001100', .228, .44 ; -      11      228K      4.4u      2
2109 0088 0064 000A 02252          de      b'10001000', .100, .10 ; 1      25      100K      10u      3
210C 0088 0032 0014 02253          de      b'10001000', .50, .20 ; 6      50      50K      20u      4
210F 00D8 0080 001A 02254          de      b'11011000', .128, .26 ; 9      65      38.4K      26u      5
2112 0088 0019 0028 02255          de      b'10001000', .25, .40 ; 16      100      25K      40u      6
2115 00C8 00C0 0034 02256          de      b'11001000', .192, .52 ; 22      130      19.2K      52u      7
2118 0088 000A 0064 02257          de      b'10001000', .10, .100 ; 46      250      10K      100u      8
211B 00C8 0060 0068 02258          de      b'11001000', .96, .104 ; 48      260      9.6K      104u      9
211E 00C8 0030 00D0 02259          de      b'11001000', .48, .208 ; 100      520      4.8K      208u      10
2121 00C9 0018 00A1 02260          de      b'11001001', .24, .161 ; 204      1040      2.4K      417u      11
2124 0080 0001 0001 02261          de      b'10000000', .1, .1 ; 5      2500      1K      1m      12
2127 0014 0090 0019 02262          de      b'00010100', .144, .25 ; 14      6253      400      2.5m      13
212A 0000 0064 000A 02263          de      b'00000000', .100, .10 ; 59      25018      100      10m      14
212D 0000 0028 0019 02264          de      b'00000000', .40, .25 ; 149      62548      40      25m      15
02265
02266 ;          timing constants table
2130 0001 0006 0009 02267          de      .001, .006, .009, .016, .022, .046, .048
0010 0016 002E
0030
2137 0064 00CC 0005 02268          de      .100, .204, .005, .014, .059, .149
000E 003B 0095
02269
02270          end

```

All other memory blocks unused.

Program Memory Words Used: 1023
Program Memory Words Free: 1

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 4 reported, 0 suppressed



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: <http://www.microchip.com>

Atlanta

Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston

Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago

Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas

Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton

Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit

Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles

Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York

Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto

Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong

Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing

Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India

Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471-6166 Fax: 81-45-471-6122

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai

Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore

Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C

Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom

Arizona Microchip Technology Ltd.
505 Eskdale Road
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

Denmark

Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hof 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France

Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

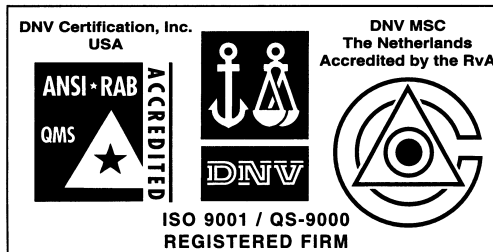
Germany

Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy

Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and water fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOC® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

All rights reserved. © 1999 Microchip Technology Incorporated. Printed in the USA. 11/99 Printed on recycled paper.

Information contained in this publication regarding device applications and the like is intended for suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights. The Microchip logo and name are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.