A homebrew telephone wardialer

The purpose of a "wardialer" is to dial a single telephone number repeatedly. One might think that would be a nuisance and, indeed, so it is. Here in Canada, there is a government-sponsored no-call list, which advertisers and telemarketers are supposed to respect. Perhaps they do. However, the no-call list does not apply to telemarketing by facsimile (fax). I designed and built this Wardialer to respond in kind to telemarketers who plague our fax machine.

There are three sheets for the schematic (see the Appendices). Sheet #1 shows the circuit that is the interface with the external telephone line. Sheet #2 shows the circuits that process signals on their way to or from the telephone line. Sheet #3 shows the microcontroller (a Microchip PIC 16F882) and the user interface.

Schematic sheet #3

Let me refer to Sheet #3 first. The user enters numerical data data through a 12-key (three columns by four rows) keypad (component Kypd). Pressing any key closes the contact which shorts the connection between that key's row and column. The keypad is interrogated whenever the PIC wants, or expects, an input. (The keypad does not generate interrupts.) To read the keypad, the PIC applies the supply voltage (+5V) to all rows (pins RB0-RB3 on the PIC) and then polls the three columns one after another to determine if any contacts have been shorted. Debouncing is done in software.

The four lines which deliver the voltage to the rows of the keypad have a dual use. They are also used to send four bits of data to the LCD display. Both uses are outputs from the point-of-view of the PIC, so buffering between the two uses is not needed. On the one hand, the LCD ignores the voltage on the common data lines unless and until the PIC manipulates the LCD's control lines in the prescribed manner. On the other hand, the voltages applied to the keypad's rows are irrelevant unless the PIC intends to read the columns.

For convenience, the three keypad columns have dedicated input pins on the PIC (pins RB4-RB6). The three 10K resistors R36 through R38 keep the input column-lines at ground level until a key press connects one of them to a row. Since the PIC treats all three lines as inputs, it is important that they not be floating when the PIC reads them.

The LCD I used is a two-row by 20 character display, with a backlight. It is a Lumex S02002DSF, which Newark sells as their part number 19J7674. Its "datasheet" is simply a dimensional drawing and is quite unsuitable for actually figuring out how to operate the device. Midas, a competing manufacturer, has a .pdf document I found on the internet titled *Specification – MC22005A6W-FPTLW*. It contains an excellent description of how to control an LCD, and the description can be applied with no trouble to the Lumex LCD. Note that I have use the 4-bit mode of operation, in which 8-bit character data is transmitted to the LCD in two consecutive nibbles.

The backlighting is entirely independent from the data circuit. It is just like a diode. Pins 15 and 16 on the LCD module are the anode (A+) and cathode (K-), respectively. Applying the 5V

supply voltage directly over these two pins will not burn out the backlight; at least, it did not burn out the one I have. But I don't think it is recommended. I wired a small 10Ω resistor into the anode-cathode circuit to reduce the level of illumination. This is resistor Rxxx in the schematic. If your 5V power supply doesn't supply enough current for the backlighting, it would be possible to draw power from upstream of the regulator.

I am running the PIC (component U5) with a 10MHz ceramic resonator (component X2). For timing purposes in the software, the instruction cycle time (one instruction takes four clock cycles) is 400 ns (0.4μs).

R25 and C36 comprise a cheap and dirty PIC reset. The R-C time constant for this pair is $10\text{K} \times 10\mu = 0.1\text{s}$. After the on-off switch is turned on, it will take about one-tenth of a second for the voltage on the capacitor to reach one-half of the supply voltage. This voltage is applied to the PIC's /MCLR pin, so the PIC will not begin operating until voltages throughout the rest of the circuit have had a chance to settle down.

Schematic sheet #1

Let me now turn to the hardware interface between the external telephone line and the rest of the circuit. The tip and ring lines from the telephone company's central office are screwed to a two-terminal connector (component Jtel) on the PCB (printed circuit board) and wired to the two contacts of a double-pole single-throw relay (component Rly).

The voltage drop on the external line is sampled by a TS117 telephone relay (component U1). Two back-to-back 18V zener diodes (components D1 and D2) allow current to flow through U1's input branch only when the voltage drop exceeds about 20V. This enables U1 to distinguish between the two cases: (i) when the line is free, and the voltage drop is 48V and (ii) when another telephone is using the line, and the voltage drop is, say, 5V. When the PIC want to determine if the telephone line is available for the Wardialer's use, it powers up the infrared LED between pins 1 and 2. This closes the relay between pins 7 and 8. If the voltage drop is 48V, then current will flow through U1's input branch. This will cause one of the infrared LEDs between pins 5 and 6 to glow, and enable the output transistor between pins 3 and 4. The PIC will read the voltage on pin 3. If it is low, then current is flowing through the input circuit and the line is not in use and is free for the Wardialer's use.

Back-to-back diodes D1 and D2 are used so that the tip and ring wires from the telephone company can be connected to connector Jtel in either polarity. Current can flow through the pair of diodes in either direction if the applied voltage is greater than the sum of the reverse breakdown voltage of one of the diodes (18V) and the forward voltage drop (about 0.7V) of the other diode. It is for this same reason that there are two parallel, but oppositely directed, infrared LEDs between pins 5 and 6 of the TS117 telephone relay. Whichever way the current flows, one of the LEDs will light up.

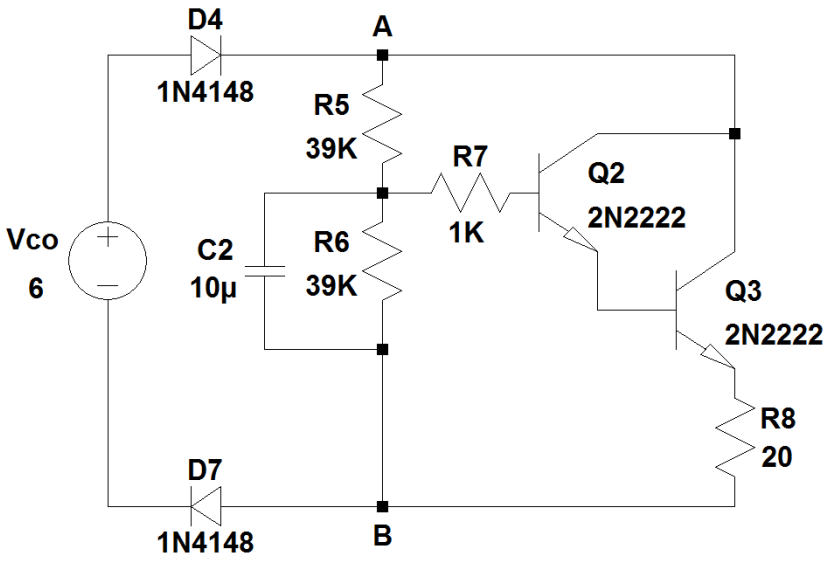
If the external line is free, then the PIC will connect it to the rest of the circuit by closing the contacts of the DPST relay Rly. The PIC does this by driving the base of transistor Q1 high.

Diode D3 exists to dissipate the magnetic field stored in the relay's coil when transistor Q1 is turned off.

The audio signal from the external telephone line is applied across the primary coil of a small 600Ω:600Ω telephone transformer (component TRtel). Like almost all modern versions of these transformers, it is a "dry" type. That means it is not designed to have any DC current flowing through the primary. Blocking capacitor C3 achieves that goal. Note that C3 has a voltage rating of 100V, sufficient to withstand the 48V that will prevail on the external line at the instant when the relay closes. A question to ask is whether C3 should actually have a higher voltage rating. The telephone company delivers a ring signal using a 90Vac waveform, which has a peak voltage of 127V. If the relay happened to be closed when a ring signal comes in, capacitor C3 would have to withstand the full 127V. That should never happen because of the subcircuit I will describe next. This subcircuit begins to operate at the instant the relay closes and, among other things, tells the telephone company that this telephone is now off-hook. The telephone company should never send a ring signal to a telephone that is off-hook.

The telephone company's central office monitors the line to determine when the parties hang up. As long as a minimum amount of DC current keeps flowing, the telephone company knows the conversation is continuing. Once the DC current stops, the telephone company releases the connection between the two customers' central offices. What constitutes a minimum amount of DC current varies. It will likely be between 40mA and 50mA. What we need to do is draw some DC current in from the external line. In the previous telephone circuits I have built, I have used a resistor to do this. For example, a 150Ω resistor will draw 40mA if the central office keeps the voltage drop on the external line at 6V.

For this device, I have not used a resistor, but a "gyrator" circuit. The following is an LTSpice schematic of this subcircuit on a stand-alone basis.



The 6V voltage source represents the DC voltage supplied by the telephone company. Diodes D4 and D7 are whichever pair of diodes in the bridge rectifier happen to be conducting. (If the tip and ring wires were connected with the other polarity, diodes D5 and D8 would do th conducting.) The bridge rectifier ensures that the positive end of the voltage drop is applied to the top of resistor R5.

The net voltage from point A to point B will be equal to the central office voltage (say, 6V) less two diode voltages drops. 1N4148 diodes are silicon diodes, so their forward voltage drop will

be about 0.7V each. Therefore, V_{AB} will be about $6 - 0.7 - 0.7 = 4.6V$. Since very little current will flow into the base of transistor Q2, resistors R5 and R6 will divide V_{AB} in half. If we ignore the tiny voltage drop over base resistor R7, then that mid-point voltage of 2.3V must be equal to the sum of: (i) the voltage drop over R8 plus (ii) the base-to-emitter voltage drops of transistors Q2 and Q3. They are silicon transistors, so their base-to-emitter voltage drops are going to be about 0.7V as well. This means the voltage drop over R8 must be equal to $4.6 - 0.7 - 0.7 = 0.9V$. Since R8 is a 20Ω resistor, it will allow $0.9V/0.02K = 45mA$ of current to flow through. Since very little current flows through the R5-R6 resistor chain compared to the transistor, the current through the diodes will be approximately equal to the current through resistor R8. We have got our 45mA draw of DC current which tells the central office that our telephone is off-hook.

The following table sets out the more exact values calculated by LTSpice for a range of central office voltages.

Voltage	Current
4V	4.56 mA
5V	20.95 mA
6V	41.14 mA
7V	62.44 mA

I did not experience any problems using a 20Ω resistor for R8. If problems do occur, it is probably because the off-hook voltage from the telephone company is on the low side, so the DC current drawn is too low for them to interpret correctly. One remedy is to reduce the value of R8. Another is to use germanium diodes for D5 through D8 instead of silicon diodes. The reduced voltage drop over germanium diodes will widen the voltage between points A and B in the subcircuit.

Before moving on from the gyrator, I should say a word about capacitor C2. It is a large value capacitor whose purpose is to stabilize the mid-point voltage of the resistor divider. It should be able to withstand high voltages, at least 50V. When the line goes off-hook, the central office will reduce the voltage from 48V to five or six volts. But that reduction will not happen instantaneously. Indeed, it is the purpose of the gyrator to help cause that transition. In any case, the gyrator must be able to withstand the normal resting line voltage.

There are two OpAmps on the inboard side of telecom transformer TRtel. Any general purpose OpAmp will do, since they are used for low-frequency audio signals. I happened to have some OPA340 chips on hand and used them. U3 (the upper one in the schematic) handles the outgoing DTMF tones of the telephone number being dialled. U2 (the lower one in the schematic) handles the incoming call progress tones, which include the dial tone, busy signal and ringback. The call progress tones are also DTMF waveforms.

DTMF is the acronym for “dual tone multi-frequency”. All of the DTMF tones are the sum of two sinusoidal waveforms with the same amplitudes. The frequencies of the sinusoidal waveforms lie between about 300 Hz and 600 Hz. Each DTMF tone corresponds to a separate pair of frequencies. It is an ingenious concept but its drawback is that the frequencies are spaced

very closely, so the hardware has to be put together with some care. One of the challenges is to make sure that the circuit that handles the outgoing signals and the circuit that handles the incoming signals do not interfere with each other.

Let me describe the outgoing, or "Transmit", OpAmp (U3) first.

C12 blocks the DC bias of the OpAmp from affecting the logic further upstream. At 500Hz, the capacitive reactance $(1/2\pi fC)$ of C12 is 3200Ω, a lot less than 47K resistor R17 with which it is in series. R14 and R15 form a voltage divider that sets the bias voltage at 2.5V. The OpAmp will source or sink current from its output pin 6 in whatever amount keeps the voltage at its negative input terminal (pin 2) equal to the bias voltage applied to its positive input terminal (pin 3). Resistors R16 and R17 set the gain to a factor of about two $(100\text{K}/47\text{K})$. The output from the OpAmp runs through a 300Ω resistor (R9) and a 20μF capacitor (C4, being two cheaper 10μF capacitors wired in parallel). At the frequencies of interest, say 500 Hz, the reactance of this capacitor is 16Ω. A large capacitor is desired, since it will have a lower reactance and therefore less influence on the 300Ω resistance of R9. The job of C4 is to block the DC bias at the OpAmp's output from causing a DC current to flow through the coil of the telecom transformer. (Note that the transformer runs both ways, so it is not accurate to call one coil the primary, as I did above, and the other coil the secondary.)

Taps are taken from both ends of resistor R9 to the input of the other OpAmp, which is the "Receive" OpAmp (U2) processing incoming signals. U2 has the same overall gain $(100\text{K}/47\text{K} \approx 2)$ as U3. It also has a DC blocking output capacitor C8 which keeps its DC bias from affecting the circuits which follow. The interesting part is how the DC bias voltage at the non-inverting input terminal of U2 (pin 3) is established. And why.

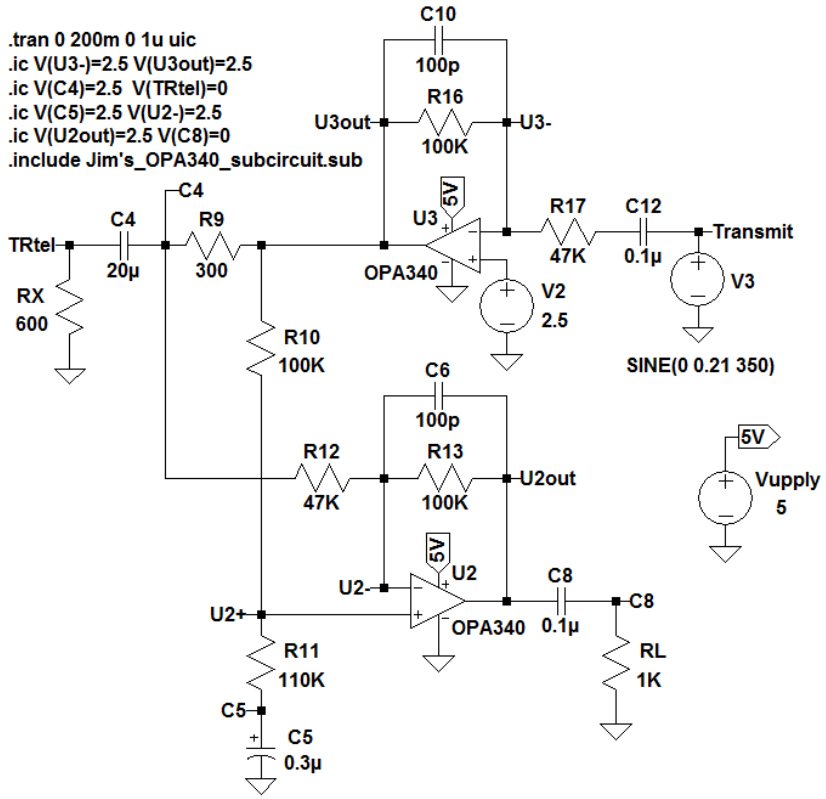
The bias level of the receive OpAmp U2 is not a fixed voltage like it is for the transmit OpAmp U3. Consider what would happen if it was a fixed voltage, and therefore produced a fixed amount of gain. Things would be fine with U2 when it processed signals coming from the transformer. But things would not be not be fine when U2 processed signals coming from U3. U3 already amplified them by a factor of two. U2 would amplify them by another factor of two. In other words, U2 would feed its following circuits (including the speaker) with unnecessarily high outgoing DTMF signals. It is more important for the user to be able to hear what is going on at the other end, and not to be blasted with signals generated by the Wardialer itself.

I set the following goal for the amplifier circuits: (i) to amplify audio voltages coming in from transformer TRtel by a factor of two, and (ii) to reduce Transmit audio voltages generated by the Wardialer by a factor of two. For this purpose, I have assumed that the peak amplitude of the audio voltage received from the external telephone line is 100mV and that the peak amplitude of the audio voltage from the Wardialer's DTMF generator is 210mV.

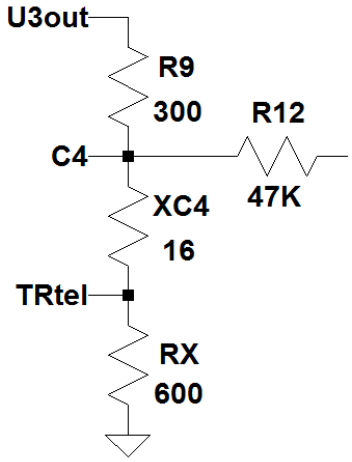
The following LTSpice model shows the Transmit (U3) and Receive (U2) OpAmps and the resistors which control the differential amplification. I used this model to look at what happens when the Wardialer is sending a DTMF telephone number onto the external line.

The audio signal to be transmitted is generated by voltage source V3, and is a 350Hz sinusoid with a peak amplitude of 210mV (described in more detail below). Transformer TRtel is treated as the load for the audio signal being transmitted. I have modeled the transformer simply as a 600Ω load. There are a lot of initial conditions, by which I set the capacitors' initial voltages to their steady-state values, just to avoid having the simulation waste time charging up capacitors.

Let me make a couple of observations about the initial conditions. The average bias voltage on U2's non-inverting terminal (node U2+) is indeed 2.5V, but it is not fixed at that voltage. An, the voltages over blocking capacitors C4 and C8 settle at 2.5V so the steady-state voltage drops over the two loads RX and RL are zero.



In operation, U3 will amplify the V3 sinusoid, producing at its output pin (node U3out) a sinusoid with a peak amplitude of about 420mV superimposed on the 2.5V DC level. This output signal is applied to the top of a voltage divider that has three series elements: R9, C4 and RX. The highlights of this voltage divider are shown at the left.



We can treat this as a simple three-element voltage divider because resistor R12 has such a large value that the current flowing through it can be ignored.

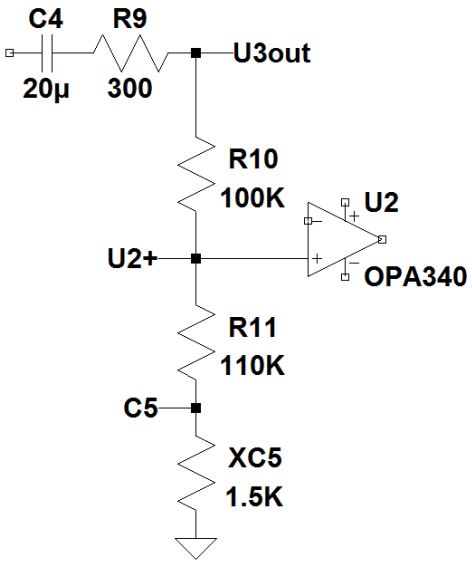
Note that I have put the reactance of C4 into the voltage divider, using the 16Ω value calculated above. Putting aside the fact that XC4 is a reactance (and so 90° out of phase with the two pure

resistances), the peak amplitude of the AC component of $\mathrm{Vac(U3out)} = 420\mathrm{mV}$ will be divided as follows:

$$\begin{aligned}\mathrm{Vac(C4)} &= \frac{16+600}{300+16+600} \times \mathrm{Vac(U3out)} &= 282\mathrm{mV}\\ \mathrm{Vac(TRtel)} &= \frac{600}{300+16+600} \times \mathrm{Vac(U3out)} &= 275\mathrm{mV}\end{aligned}$$

Of course, there will be some loss of power in transformer TRtel, in DC blocking capacitor C3, the contacts of relay Rly and so on, with the result that the peak AC voltage that actually reaches the external telephone line will be somewhat less than Vac(TRtel), probably 200mV or so.

The voltage which reaches the two input terminals of OpAmp U2 will be much less than this. The voltages at these two terminals are brought from the two ends of resistor R9. The voltage difference between the two ends is $\mathrm{Vac(U3out)} - \mathrm{Vac(C4)} = 420\mathrm{mV} - 282\mathrm{mV} = 138\mathrm{mV}$. And only half of that will reach U2 because of the action of a second voltage divider. Resistors R10 and R11 and capacitor C5 constitute another three-element voltage divider, shown here.



The non-inverting terminal of U2 will not draw current away from node U2+. Therefore, current will flow down through R10, R11 and C5 to ground. At the frequencies of interest, say, 350Hz, the reactance of C4 is 1.5K. The peak AC voltage at node U2+ will be:

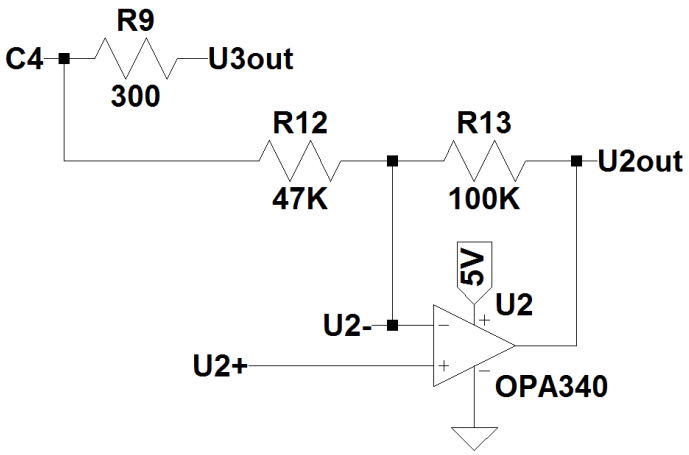
$$\begin{aligned}\mathrm{Vac(U2+)} &= \frac{110\mathrm{K}+1.5\mathrm{K}}{100\mathrm{K}+110\mathrm{K}+1.5\mathrm{K}} \times \mathrm{Vac(U3out)}\\ &= 221\mathrm{mV}\end{aligned}$$

It should be noted that the RC time constant of the R11-C5 pair is $\tau = 110\mathrm{K} \times 0.3\mu = 33\mathrm{ms}$. This is much longer than the 2.9ms period of the 350Hz signal. One concludes that the DC voltage level over capacitor C5 will not change very much, even though there is an alternating voltage at the non-inverting terminal.

The operation of OpAmp U2 is such that it will source or sink as much current as is necessary to keep the voltage at its inverting terminal (U2-) the same as the voltage at the non-inverting terminal. The bits of the circuit which are relevant to this equality are shown in the next schematic.

If we let i_{12} be the current (assumed to flow from left-to-right) through R12, then the equality can be written as follows:

$$\begin{aligned}&\mathrm{Vac(U2-)} = \mathrm{Vac(U2+)}\\ \rightarrow\quad &\mathrm{Vac(C4)} - \mathrm{R12}i_{R12} = \mathrm{Vac}(U2+)\end{aligned}$$



Substituting the expansions for $\mathrm{Vac(C4)}$ and $\mathrm{Vac(U2+)}$ from above gives:

$$\frac{16+600}{300+16+600}\times \mathrm{Vac(U3out)} - \mathrm{R12}i_{R12} = \frac{110\mathrm{K}+1.5\mathrm{K}}{100\mathrm{K}+110\mathrm{K}+1.5\mathrm{K}}\times \mathrm{Vac(U3out)}$$

$$\rightarrow \quad \mathrm{R12}i_{R12} = 0.145302 \times \mathrm{Vac(U3out)}$$

$$\rightarrow \quad i_{R12} = 0.00309 \times \mathrm{Vac(U3out)}, \text{measured in mA}$$

Since i_{R12} is algebraically positive, it confirms that the current does flow from left-to-right through R12 and that U2 is sinking that current. The voltage drop over the 100K feedback resistor R13 (from left-to-right) is given by:

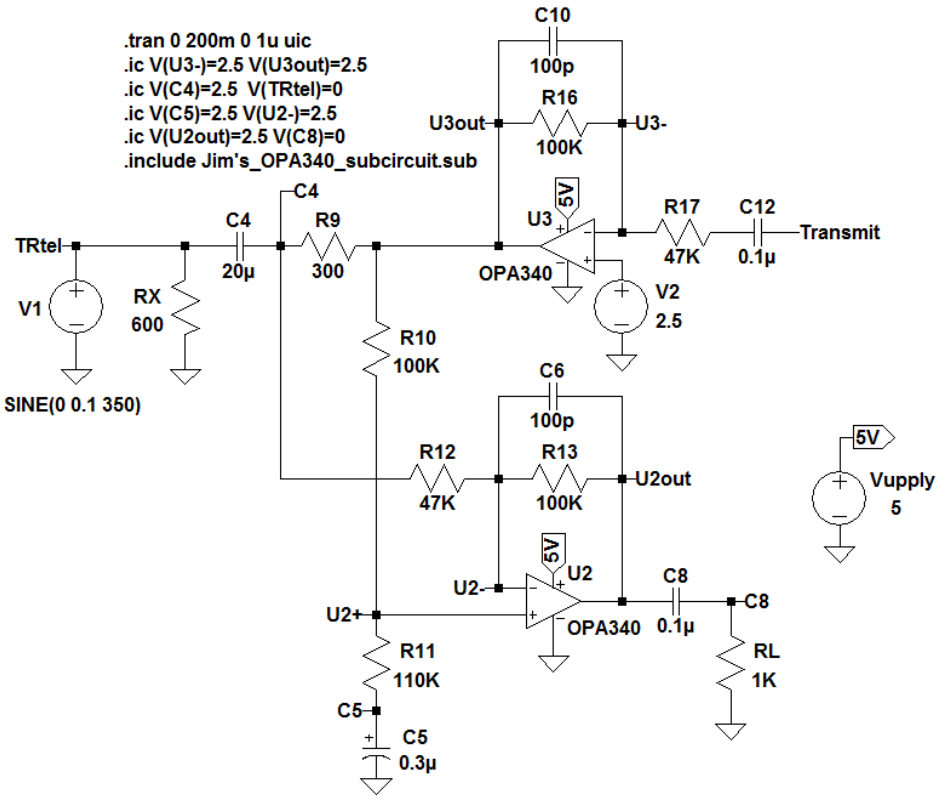
$$\Delta\mathrm{Vac(R13)} = 100\mathrm{K} \times i_{R12} = 0.309 \times \mathrm{Vac(U3out)}, \text{measured in Volts}$$

The voltage at node U2out, relative to ground, is the voltage at node U2+ less the differential voltage $\Delta\mathrm{Vac(R13)}$. This total is equal to:

$$\begin{aligned}\mathrm{Vac(U2out)} &= \mathrm{Vac(U2+)} - \Delta\mathrm{Vac(R13)} \\ &= \frac{110\mathrm{K}+1.5\mathrm{K}}{100\mathrm{K}+110\mathrm{K}+1.5\mathrm{K}}\times \mathrm{Vac(U3out)} - 0.309 \times \mathrm{Vac(U3out)} \\ &= 0.219 \times \mathrm{Vac(U3out)} \\ &= 0.219 \times 420\mathrm{mV} \\ &= 92\mathrm{mV}\end{aligned}$$

This is the AC component of U2's output voltage when it is processing a Transmit waveform with a 210mV amplitude. (This will, of course, be superimposed on the 2.5 DC bias.) This represents suppression by a factor of about one-half ($92/210 = 43.8\%$), which is the second goal I set out above. Execution of the LTSpice model confirms this outcome.

The following LTSpice model shows the other case of interest, when the input circuit is processing a DTMF signal received from the external telephone line.



V1 is the input sinusoid for this case. It is assumed to have a peak amplitude of 100mV. This time around, there is no AC component in the Transmit signal, so the output from OpAmp U3 (U3out) will be a constant 2.5V. The voltage drop over capacitor C5 will stabilize at the level where the reference voltage at U2's positive terminal will also be a constant 2.5V. The audio signal from the transformer will be doubled by the $100K/47K \approx 2$ gain of U2. This achieves the first goal I set out above.

The DTMF tone generator for dialling telephone numbers

I am going to move on to Sheet #2 of the schematic. This sheet includes the IC (U6) which generates the outgoing DTMF tones and the ICs (U9 and U10) which decode the incoming call progress tones.

U6 is a 14-pin DTMF tone generator (Holtek HT9200A). It has three control lines: (i) a chip enable line (pin 1), (ii) a clock input (pin 5) and (iii) an input data line (pin 6). It requires its own external oscillator, which is a 3.58MHz ceramic resonator (component X1). I wired U6 in its serial mode, in which it receives data bits one-by-one, but it can also be wired in a parallel mode with a 4-bit data bus. I used the serial mode to reduce the number of I/O lines that the PIC uses to control the various peripherals. The PIC sends information to U6 asynchronously as a stream of bits. There are two types of information packets: (i) 8-bit instruction bytes and (ii) 8-bit data bytes. The output from U6 (on pin 7) is the DTMF tone specified by the information packet most-recently sent to U6 by the PIC.

The datasheet states that the nominal amplitude of the DTMF output will be 150mV. This is an RMS voltage, so the nominal peak amplitude will be a factor of $\sqrt{2}$ greater, or 210mV. (This is the peak voltage of the audio Transmit signal I used in the previous section.)

In this application, U6 is used to generate the DTMF tones which correspond to the ten basic numerals 0, 1, ..., 8, 9 which make up telephone numbers. Although U6 can also generate a few other DTMF tones, they are not used in this application.

The protocol for delivering information to U6 is not very complicated, but it must be followed exactly. The datasheet has diagrams which describe the timing, and they should be compared with the assembly listing set out in the Appendices. Only one subroutine is needed to manage U6; in the source code it is named SendOneDTMFCode(). Unlike the LCD, U6 does not need a special initialization procedure. There are two points to note:

1. I wrote the source code so that the telephone number would be dialled relatively slowly. I send each numeric DTMF tone for 500ms, and leave a silent interval of 500ms before the next DTMF code. As long as each code (digit) is more than 100ms long, the telephone company should be able to detect it.

2. There are 16 frequency pairs in the basic set of DTMF tones. Each tone can therefore be represented by a single hexadecimal digit. A huge amount of wasted time can be avoided by observing that the hexadecimal digit for the numeral zero is not 0x00 = b'0000' as one might expect, but is 0x0A = '1010'.

The DTMF decoder for call progress tones

The telephone company sends certain DTMF signals to a telephone. There are perhaps a dozen such signals. They are called "call progress" tones. The most familiar ones are the dial tone, the busy signal, the ringback and the message-waiting signal.

Aside #1: The ringback is the ringing signal you hear after you have dialled a phone number and the telephone at the other end starts to ring. The ringback is generated by the telephone company's central office at your geographical end, and it is completely independent from the ring signal on the other end, which is generated by the central office at the callee's geographical end. The ringback pulses you hear have a timing that is not related to the ring pulses which the callee hears his telephone make.

Aside #2: The message-waiting signal is used to tell you that you have a message waiting on your answering machine. Although it is often called a "fast-busy" signal, that is not correct; the message-waiting signal uses different frequencies than the busy signal. You may hear a message-waiting signal instead of a dial tone when you pick up the handset in preparation for making a call. We need the Wardialer to recognize message-waiting signals, and to treat them in the same way as it treats normal dial tones.

Aside #3: Call progress tones can be distinguished from each other in two completely separate ways. One way is to sort out the two frequencies that comprise them. But, there is another way. Each type of call progress also has a distinct pattern of pulse lengths. The timing of the pulses and silences is called the "cadence". Either, or both, of the frequencies or the timing can be used to figure out what type of call progress signal is being received. For the Wardialer, I have chosen to make the distinction based solely on frequencies. It completely ignores the length of the pulses and inter-pulse silences. The reason is this –

Any procedure based on timing must wait through at least one complete pulse and one complete silent interval. Since the procedure is likely to begin partway through a pulse or a silent period, some additional wait time will be required before the first full pulse or silent period begins. As an example, the cadence of a ringback signal in Canada and the U.S. is a two-second pulse and a four-second silence. If the callee answers his telephone after the first pulse, the procedure will be confused. It will have to wait through the full four seconds before it realizes that there is no second call progress pulse and that, therefore, the callee must have picked up the handset.

Discriminating between call progress tones using only the frequencies avoids this awkward delay because it can be accomplished in a fraction of one pulse.

The following table sets out the component frequencies of the call progress tones the Wardialer must be able to decode. For reference purposes only, the table also set out the corresponding cadences.

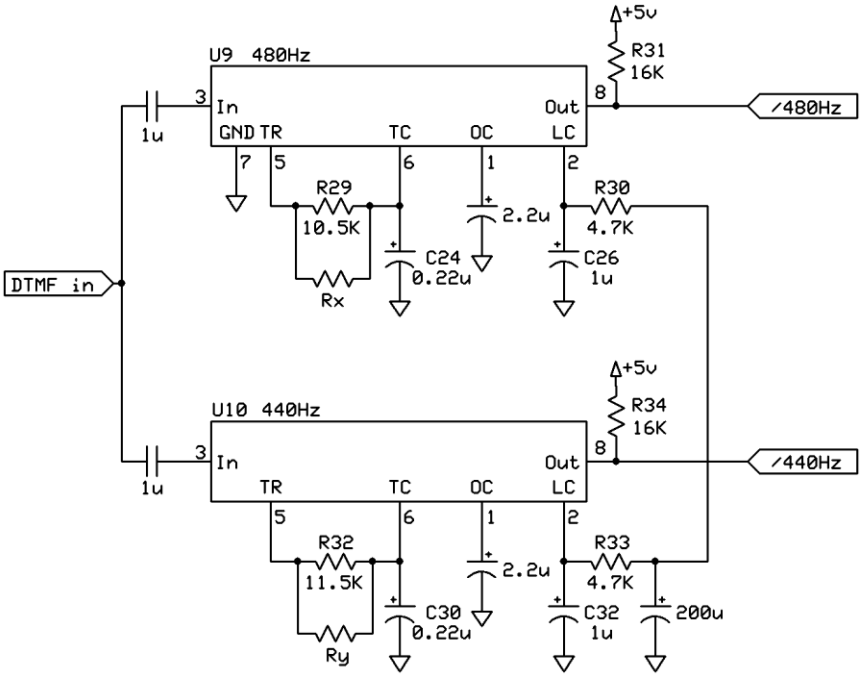
Call progress tone	**Frequency #1**	**Frequency #2**	**Cadence**
Dial tone	440Hz	350Hz	continuous
Message-waiting	440Hz	350Hz	250ms pulse; 250 ms silence
Busy signal	480Hz	620Hz	500ms pulse; 500ms silence
Ringback	480Hz	440Hz	2s pulse; 4s silence
Silence	none	none	no pulse

Note that the two frequencies for message-waiting are the same as for a dial tone. Detecting one is tantamount to detecting the other. The table includes four distinct frequencies, but it turns out that we can get away with looking at only two of them.

Call progress tone	**480Hz**	**440Hz**
Dial tone	Absent	Present
Message-waiting	Absent	Present
Busy signal	Present	Absent
Ringback	Resent	Present
Silence	Absent	Absent

The Wardialer uses two ICs, U9 and U10, to detect these two frequencies. Each one is an 8-pin LM567 tone decoder designed for applications like this one. They use external resistors and capacitors to set the set the central frequency and bandwidth of a filter. One is tuned to 480Hz, the other to 440Hz. Their outputs go to separate pins on the PIC (RC2 and RC3), so the PIC can read the outputs whenever it chooses. The outputs of U9 and U10 are logic level voltages. The default output is the power supply voltage (5V). When the incoming signal includes the central frequency, the output voltage goes low.

The two frequencies are closely spaced – only 40Hz separates them – so one must adhere closely to what is said in the datasheet to ensure reliable discrimination. The following diagram shows the bits of the schematic that are relevant here.



The central frequency of each filter is set by the values of the resistors and capacitors wired to pins 5 and 6. The formula is: $f_0 = 1.1/RC$. I chose the values so that:

U9	$f_0 = \dfrac{1.1}{10.5\text{K} \times 0.22\mu} = 476.2\text{Hz}$
U10	$f_0 = \dfrac{1.1}{11.5\text{K} \times 0.22\mu} = 434.8\text{Hz}$

These frequencies are a percent or two less than the target frequencies of 480Hz and 440Hz. That is by choice. I planned on doing some fine tuning. That is most easily done by adding some resistance in parallel to R29 and/or R32, which will reduce the denominator and increase the central frequency. These tuning resistors are labelled Rx and Ry in the schematic. Even though I planned to do some tuning, I still used precision components. I used 0.22µF capacitors (Newark part number 04J5180) which have a tolerance of 2%. And I used resistors with a tolerance of 1%.

When all was said and done, I got excellent results using a 270K resistor for Rx and a 500K resistor for Ry. Let me say a word or two about how I did the tuning, which I did after having made the printed circuit. When I soldered R29 and R32 to the PCB, I left them a small distance above the board. That made it easy to clip various test resistors in parallel while the board is powered up. I wrote a short test routine in the software, which was executed after the unit dialled a telephone number. The test routine continuously sampled the outputs from U9 and U10 and displayed their values as "HH", "HL", "LH" or "LL" directly on the LCD. The test routine loop continues until the telephone company puts an end to the call.

Among other things, the test routine showed a noticeable delay which occurs before the filters produce a clear result. As one expects, these filters (indeed, any filters) need several cycles of input signal before they can figure out what they've got. The datasheet for the LM576 states that the maximum turn on-turn off frequency is $f_0/20$. I'm not exactly sure what this frequency is, but it may be the time it takes for the chip to respond to a change in the incoming signal. If so, then $f_0/20$ implies that the response time is 20 cycles. The period of one cycle at 440Hz is 2.3 milliseconds, so 20 cycles takes 46 milliseconds. That is very consistent with the delays I saw as I watched the outputs from the filters on the LCD.

I learned another lesson from my tuning experiment. It is not enough to take just one sample of the filter outputs. Not only is there a delay before the signals settle, but the two filters will not settle at the same time. It is necessary to take several samples before reaching any conclusion. I wrote the code to take 11 samples at intervals of 100µs, to take the complements of the samples, and then to take the logical OR of the complements. If the central frequency was detected at least once during the one second examination, it was reported as being "Present".

Let me move on to control over the bandwidth of the filters in U9 and U10. The filters do not have infinite Q. They will recognize as valid a range of frequencies on either side of their central frequency. The resistors and capacitors connected to pin 2 of the LM576 give the designer some control over the width of the recognition band. The bandwidth is a function of the parameter f_0C, where C is the capacitance from pin 2 to ground. In our circuit, the capacitors for the two chips are C26 and C32, respectively. I used 1µF capacitors. The parameter f_0C is virtually the same for the two frequencies, being approximately $460\text{Hz} \times 1\mu = 460$ HzµF. Note the units of the parameter. They are unusual, but that is neither here nor there. It is simply that the manufacturer has found that the bandwidth is described most easily this way.

The datasheet describes two separate operating regions, distinguished by the amplitude of the incoming DTMF signal. If the incoming DTMF signal has an RMS voltage greater than 200mV, then Table 2 in datasheet states that a parameter f_0C of 460 will cause the bandwidth to be about 14% of the central frequency. If the incoming DTMF has an RMS voltage less than 200mV, then recourse must be made to Figure 5 in the datasheet. It shows how the bandwidth decreases as the signal strength decreases. At 50mVrms, and with $f_0C = 460$, the bandwidth will be down to about 10% of the central frequency. These bandwidth are pretty broad and could be cause for concern. At 480Hz, a 10% bandwidth is 48Hz, and extends far enough down to detect a 440Hz signal. One might be tempted to increase the capacitance – a bigger capacitance (and so, bigger parameter f_0C) tends to produce a smaller bandwidth. But, increasing C26 and C32 starts to

interfere with another recommendation made by the manufacturer: that the grounding capacitor wired to pin 1 be at least twice as large as the bandwidth capacitor.

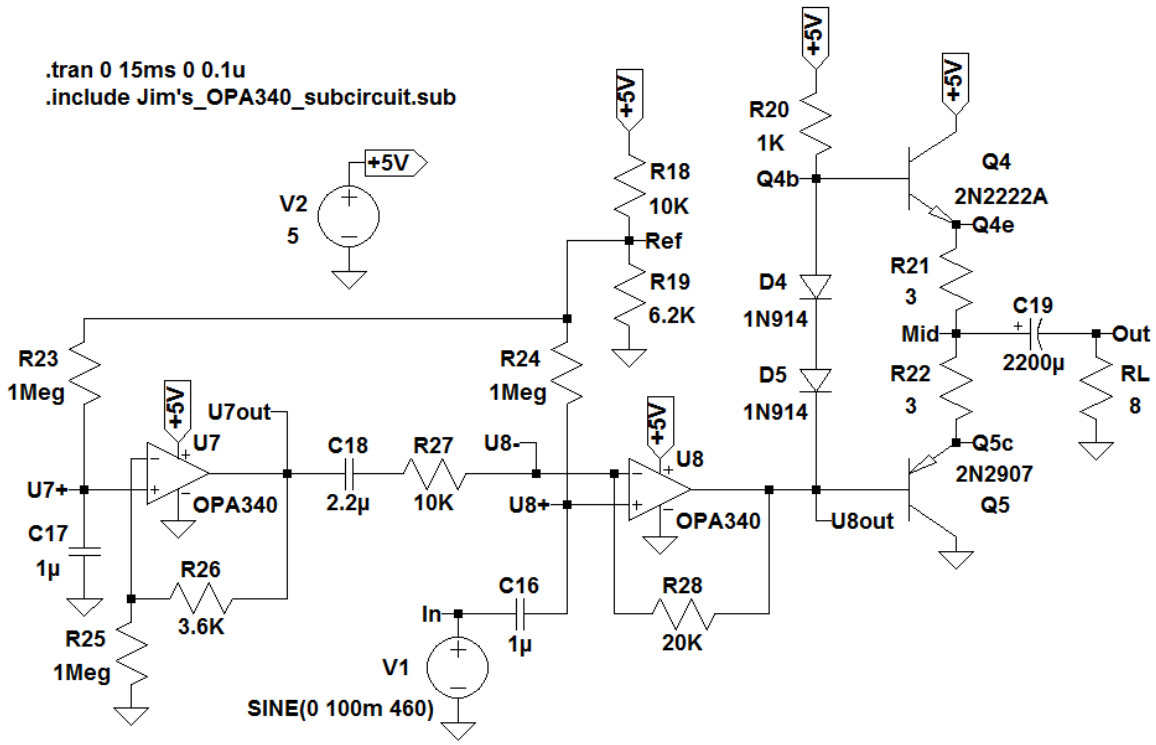
As luck would have it, I did not experience any problems with bandwidth using the values in the schematic, so I did not pursue my concerns about this any further.

Inside the LM567, the output circuit is an npn transistor with an open collector. The collector is tied high by an external resistor. R31 and R34 are the external pull-up resistors for U9 and U10, respectively. When the central frequency is detected, the output transistor is driven into saturation, which pulls output pin 8 low. In other words, when the desired frequency is present, the output from the filters is low. The software must take this into account. In the code, subroutine ReadCallProgress() reads the signals on RC2 (from U9) and RC3 (from U10) and complements them, so that a "1" corresponds to the presence of the desired frequency.

The audio amplifier

The Wardialer includes a small 8Ω speaker. A SPST toggle switch on the front panel allows the speaker to be turned on or off. It is useful to be able to listen to what is going on, particularly when a number is being attacked for the first time. Once things are found to be running smoothly, it's handy to be able to turn the speaker off and let the device run quietly.

The following schematic shows the details of the audio amplifier.



The Receive signal (generated by the circuits in the previous section) is applied to the positive input terminal of OPA340 OpAmp U8. The DC bias voltage at the positive input terminals of both OpAmps is set by a voltage divider comprised of R18 and R19. They divide the 5V supply

voltage by a fraction $6.2K/(10K + 6.2K)$ to give a DV bias of 1.914V. While the DC bias voltage is not critical to operation, setting it at this particular voltage gives the maximum voltage swing over the speaker before clipping sets in.

For design purposes, I have assumed the incoming signal, represented by voltage source V1, has a peak amplitude of between 100mV and 600mV. In combination with DC blocking capacitor C16, the incoming AC waveform is superimposed on the 1.914V DC bias and applied to the non-inverting terminal (node U8+) of OpAmp U8. U8 will work as hard as it can to keep the instantaneous voltage at both of its input terminals the same. For now, don't worry about how it does that, just note that the voltage at the right-hand end of resistor R27 (at node U8-) will be equal to the DC bias plus the incoming AC waveform.

OpAmp U7 will also be working as hard as it can, to keep the voltage at its two terminals the same. The voltage at its non-inverting terminal (node U7+) is also fixed at 1.914V. Smoothing capacitor C17 damps down voltage excursions at this terminal. The voltage at the left-hand end of resistor R27 will want to want to run up-and-down in synch with the voltage at its right-hand end, which would tend to pull both ends of resistor R26 up-and-down as well. U7 will try to prevent that. It will fire current out of its output terminal (node U7out). That current will flow through R26 and then down to ground through R25. Because R25 has such a high value (1Meg), a miniscule amount of current will be enough to adjust the voltage at the inverting terminal to keep it at 1.914V. Furthermore, since the value of the feedback resistor R26 is so low compared with R25, the drop in voltage from the output terminal of U7 (node U7out) back to the inverting terminal will also be miniscule. The result is that voltage at node U7out will be kept virtually equal to the DC bias voltage 1.914V.

We can therefore say that the voltage drop over resistor R27, from right-to-left, will be equal to the input AC signal voltage. The current flowing through R27 will be equal to the input AC signal voltage divided by R27's 10K resistance. That is the real purpose of U7 and R26: to create an AC current which is proportional to the input AC voltage.

Now let's look at U8. Since no current flows into or out of its inverting terminal (node U8-), any and all current which flows through R27 must also flow through R28. Since R28 has twice the value of R27 but carries the same current, the voltage drop from U8's output terminal (node U8out) to its inverting terminal (node U8-) will be twice the voltage drop over R27. Adding things up, the voltage at node U8out will be equal to three times the voltage of the incoming AC signal. As before, this AC voltage will be superimposed on the 1.914V DC bias.

This brings us to transistors Q4 and Q5. They are npn and pnp transistors, respectively, wired in a push-pull arrangement to drive the speaker. Each transistor is biased for class-B operation. The two transistors will conduct during alternate half cycles of the AC waveform.

Look at the pair of diodes which connect the bases of the two transistors. They are silicon diodes. When they are forward-biased, each will drop an almost-constant voltage of 0.7V. The two transistors are also made of silicon. When they are forward-biased, the voltage drops across their base-to-emitter junctions will also be about 0.7V. The diodes keep both transistors just on

the verge of their active operating regions. The better the diodes and transistors are matched, the less distortion will show up at the output.

The voltage drop between the bases of the two transistors will be approximately constant at 1.4V. Voltage swings at Q4's base will be absorbed by resistor R20. R20 will allow a greater or lesser amount of current to flow through so that the voltages at the bases can move up or down in lockstep. In the absence of an AC signal, the voltage drop over resistor R20 will be: (i) the 5V supply voltage, less (ii) the 1.4V drop over the diodes and less (iii) the 1.914V DC bias at the output of U8. That is 1.686V. The DC current which flows down through R20 will be $1.686\text{V}/1\text{K} = 1.686\text{mA}$.

The steady-state voltage at the point where the diodes meet can be calculated in two ways. From the top down, it is: (i) the 5V supply less (ii) the 1.686V drop over R20 and less (iii) the 0.7V drop over diode D4, giving a total of 2.614V. From the bottom up, it is: (i) the 1.914V DC bias at the output from U8 plus (ii) the 0.7V drop over diode D5, giving a total of 2.614V.

Let's take a quick look at the speaker, which I have modelled as an 8Ω resistor. DC blocking capacitor C19 has quite a high value: 2200μF. At the frequencies we are looking at, say 460Hz on average, the reactance ($1/2\pi fC$) of C19 is 0.16Ω. We need a large capacitor here to maximize the amount of power that reaches the speaker. Since transistors Q4 and Q5, and their emitter resistors, are all wired symmetrically with respect to the power supply, C19 will charge up to a DC voltage of 2.5V. When there is no AC signal, and with the two transistors on the verge of their active regions, the voltage at Q4's base (node Q4b) will be this DC voltage plus one base-to-emitter junction voltage drop. That is $2.5 + 0.7 = 3.3\text{V}$. Similarly, the DC voltage at Q5's base (node Q5b) will be $2.5 - 0.7 = 1.9\text{V}$. This configuration centers the output at 2.5V, and allows for the maximum swing in voltage before transistors Q4 and Q5 stop operating in their active regions. (It is for this reason that resistors R18 and R19 were selected above so that the DC bias of the OpAmps is 1.9V or, more precisely, 1.914V.)

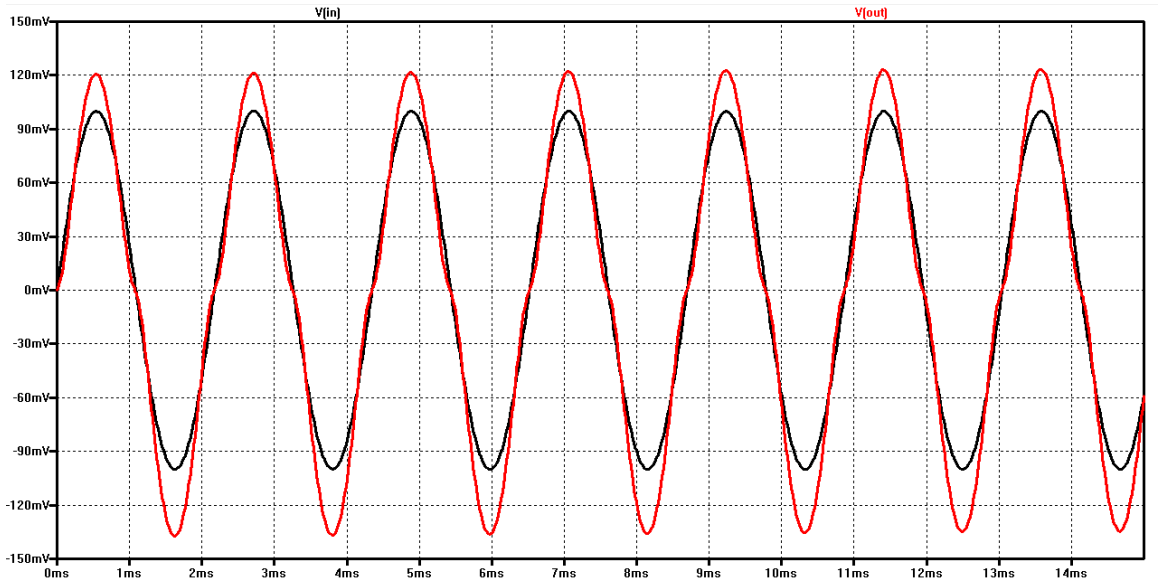
Consider the half-cycle of the input AC waveform during which the voltage at Q5's base (node U8out) rises above the DC level. The voltage at the base of Q4 will also rise above its DC level. pnp transistor Q5 will be cut off as the voltage drop over its base-to-emitter junction falls below 0.7V. But npn transistor Q4 will be driven into its active region. Current will flow down through Q4, its emitter resistor R21 and out into the speaker. The flow of current through R21 will cause the voltage at Q4's emitter to rise. However, the amount of the rise is limited. It cannot rise above the level that would take Q4 back out of its active region. The way it works itself out is that the voltage at Q4's emitter will rise by the same amount as voltage at its base. The peak magnitude of the rise will be three times the peak magnitude of the incoming AC signal. The current from Q4's emitter flows through a series resistance of 11Ω, of which 8Ω is the speaker. Therefore, the peak voltage over the speaker will be $3 \times 8/11 = 2.18\text{x}'\text{s}$ the peak voltage of the incoming AC signal. If the peak input voltage is 100mV, then the peak voltage over the speaker will be 218mV.

During the other half-cycle of the input AC waveform, transistor Q4 will be cut off and Q5 will be driven into its active region. Its emitter voltage will be drawn down in synch with the drawdown of its base voltage. Current will be pulled in from the speaker. The speaker will

experience $8/11$ths of the voltage swing. Again, the peak voltage (negative for this half-cycle) will be 218mV.

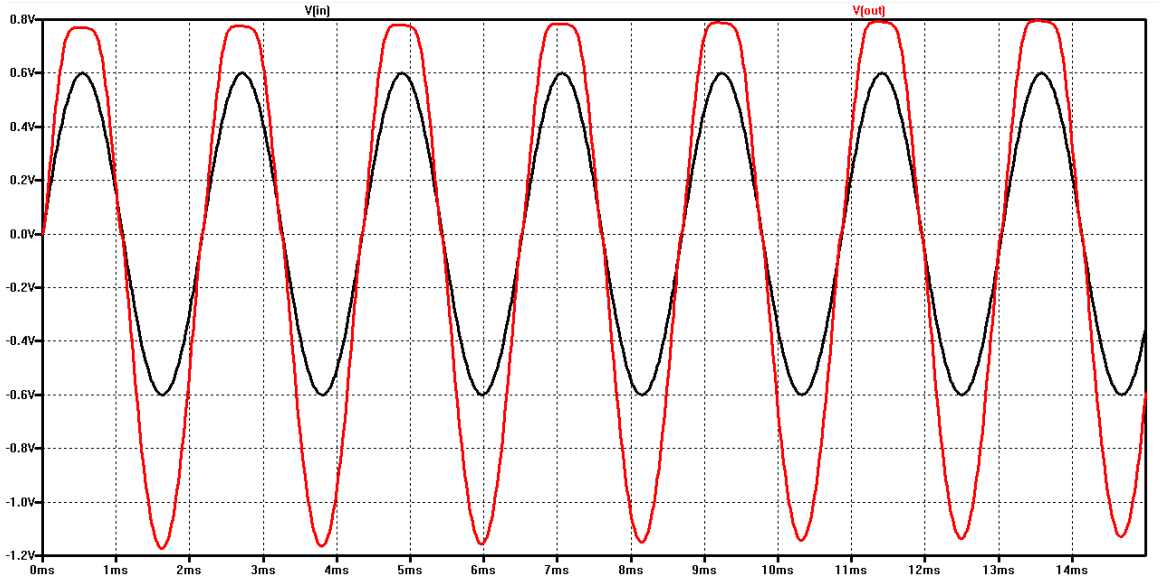
How much power does the speaker dissipate? If the peak AC voltage over the speaker is 218mV, then the RMS equivalent is $218/\sqrt{2} = 154\text{mV}$. The RMS current will be (using Ohm's Law) $154\text{mV}/8 = 19.25\text{mA}$. The product of voltage and current is the power: $154\text{mV} \times 19.25\text{mA} = 3.0\text{mW}$.

The following is a plot of the input and output voltage waveforms produced by the LTSpice simulation.



The input AC waveform is rendered in red. Its peak amplitude is 100mV. The voltage over the speaker is rendered in black. Its peak amplitude is about 120mV – quite a bit less than the 218mV we calculated. The output voltage also has about 10mV of asymmetry, biased towards the negative side. In addition, there is some distortion as the output voltage passes through zero. All three of these faults arise because real diodes and transistors are not ideal. In particular, the forward voltage over a diode is not constant, but does vary a little bit with the strength of the current. The base-to-emitter junction of a real transistor has exactly the same kind of variation. Furthermore, the boundary between a transistor's cut-off and active regions is not a sudden change, but a gradual one. All of these realities permit the components to partially compensate for externally-imposed changes by changing their own internal voltage drops. The combined effect is that only a part of the externally-imposed changes find their way all the through to the speaker.

Now, 100mV is about the minimum level of signal one would expect to come in from the telephone line. Signals can be much bigger. The following plot shows the output voltage when the input AC signal has a peak amplitude of 600mV. Once again, the input signal is shown in black and the output signal is shown in red.



We see here the onset of a different kind of distortion. The output peaks are showing some flatness, or "clipping". We are trying to get more amplification than the transistors can deliver. As the voltage swings get larger, the voltages at the transistors' bases skirt closer and closer to the limits of the power supply. The transistors begin to leave their active operating regions and enter saturation.

For the sinusoidal components of AC signals, the distortion on this most recent plot are not a problem. But, it should be noted that the amplifier is at its limit. With the peak-to-peak voltage of about 1.8V as shown in the plot, the speaker is dissipating about $\left(V/\sqrt{2}\right)^2/R = 1.8^2/16 = 203\text{mW}$ of power. I used a 150mW speaker (Digikey's part number 668-1136-ND), so this amount of amplification is a little too much for the speaker as well.

The PIC program – data entry

The source code listing is set out in the Appendices below. There are two main parts to the program. The first part runs when the Wardialer is powered up, and guides the user in entering the parameters to be used for the endeavour. The second part is execution of the user's instruction. This section describes the data entry procedure.

If you think through the steps of making a telephone call, it will become clear that several choices need to be made. How many times should the callee's phone be allowed to ring before deciding that the callee is "not at home"? If the callee's phone is busy, it makes sense to try again shortly. If it is busy on the second attempt, should a third attempt be made? How long should the Wardialer wait between calls? And so on.

The following table sets out the prompts the Wardialer displays on the LCD. It also sets out the names of the registers in which the user's entries are stored. The alphanumeric text in the messages is stored in the PIC's flash memory, not in EEPROM. The numerals shown for the prompts in the following table are related to their locations in the flash memory. The Wardialer

does some validation as the user enters the data. It displays appropriate error and warning messages which have the reference numbers which are missing from the following table.

Prompt		**Data is stored in**	**Purpose**
#1	Telephone number: _	TelNum1, ..., TelNum14	Target telephone number, up to 14 digits
#2	Test dial? (1=Y): _		Listen in on call progress
#3	Mins between calls, max (0-255): _	MaxWaitToCall	Maximum number of minutes to wait between making calls
#4	Mins between calls, min (1-max): _	MinWaitToCall	Minimum number of minutes to wait between making calls
#8	Secs between calls, max (1-255): _	MinWaitToCall	Maximum number of seconds to wait between making calls
#10	Number of telephone calls (10-255): _	MaxNumCalls	Maximum number of calls to make
#12	Number of ring pulses (3-10): _	MaxNumRingbacks	Number of ringback pulses to allow
#14	Number of busy tries (5-50): _	MaxNumBusyTries	Number of re-dial attempts if busy
#16	Max sec between busy tries (5-50): _	MaxWaitOnBusy	Maximum number of seconds to wait between re-dial attempts
#17	Min sec between busy tries (1-max): _	MinWaitOnBusy	Minimum number of seconds to wait between re-dial attempts
#20	Max sec to hold callee (2-240): _	MaxSecsToHold	Maximum number of seconds to wait before hanging up

There is a colon at the end of each of these prompts. After displaying the prompt, the cursor settles on the second position after the prompt and the PIC enters a loop while it awaits the user's keystrokes.

A word about the keypad is in order. The keypad is a normal 3-column by 4-row telephone-type keypad. Ten of the keys are taken up by the basic ten digits. But the "*" key at the lower left and the "#" key at the lower right have special meanings here. The "*" key is used as the backspace key and the "#" key is used as the ENTER key. The PIC does not really begin to process the user's entries until he presses the "#" (ENTER) key. Quite a lot of code is required to keep track of the display and the cursor position as the user types in an entry.

The PIC uses different subroutines to process the telephone number, on the one hand, and the other numeric entries, on the other hand. As the user enters telephone digits, the PIC simply adds them to a stack of 14 registers dedicated to their storage. If the user backspaces, the PIC simply moves down one item in the stack. When the user presses the "*" (ENTER) key, the program records the number of digits in the telephone number in a separate register named LenTelNum. The subroutine GetTelephoneNumber() is the procedure which, well, gets the telephone number. As the user types in the digits, the subroutine updates the LCD display. It re-

writes the entire number, not just the most-recently added digit. It calls another subroutine, named DisplayTelephoneNumber(), to display the current version of the telephone number.

The Wardialer does not validate the telephone number. It can include leading numerals to select an outside line if it is connected to a PBX. It can include long-distance codes. As soon as the user enters the telephone number, the next prompt is Test dial? (1=Y). If the user answers in the affirmative, the device immediately goes through the process of dialling the telephone number. If the speaker is turned on, the user can listen to the entire process. This is useful, not just to ensure that the correct number has been entered, but (and especially for a new target number) to ensure that the procedure unfolds as expected.

All of the other user entries, that is, the data other than the digits of the telephone number, are designed so that their numerical values are between zero and d'255', allowing them to be stored in single byte registers. A maximum of three characters are all that is needed to display any of the user's replies to these prompts. The user's keystrokes are saved in registers Digit1, Digit2 and Digit3 as the user enters them. Another register, named NumDigits, keeps track of how many digits the user has entered. If the user tries to enter a fourth digit, it will simply overwrite the third one. When the user presses the "#" (ENTER) key, the key calls subroutine ConvertNumericFieldToBinary() to convert the user's entry into a binary value, which is then stored in the appropriate register.

The program allows for a variable length of time between successive telephone calls. A variable length of time between calls makes the process more confusing, and disruptive, for a human callee. The user specifies maximum and minimum wait times. First, the PIC prompts the user to enter the maximum length of time between calls, which it assumes is stat in minutes. This number is stored in register MaxWaitToCall. If the user's entry is two or more, then the PIC prompts for the minimum length of time between calls (also in minutes), which it stores in register MinWaitToCall. The difference between these two extremes is calculated and stored in register DeltaWaitToCall. At the end of a telephone call during the main operating loop, the PIC will calculate how long to wait (in minutes) before starting the next call. The code includes a small 8-bit random number generator, in subroutine Random(). Calling subroutine Random() returns a random 8-bit sequence in the *w* accumulator. The PIC treats the random number as a binary fraction. The left-most bit (if present) represents one-half. The second bit from the left (if present) represents one-quarter. The third bit from the left (if present) represents one-eighth. And so on, all the way down to the right-most bit, which (if present) represents $1/2^8$, or one-two hundred fifty sixth.

The PIC calls subroutine Multiply() to multiply this 8-bit random fraction by the 8-bit value DeltaWaitToCall. Multiply() carries out a full 16-bit multiplication, but returns only the high order 8 bits. The product is added to MinWaitToCall. The result of the addition is a random integer somewhere between MinWaitToCall and MaxWaitToCall. The program waits this number of minutes and then calls again. I might just point out that the subroutine CalcAndDoWaitToCall() does the randomness calculation, and also waits through the calculated number of minutes or seconds.

For some target telephones, it is useful to call more frequently than this. The PIC manages things differently if the user enters either zero or one in reply to the prompt for MaxWaitToCall. It then assumes that the user wants to measure the inter-call period in seconds, not minutes. The PIC sets MaxWaitToCall to zero, and then displays the prompt "Secs between calls, max (1-255):". The user's entry is stored in register MinWaitToCall. The PIC can easily avoid confusion by the dual use of register MinWaitToCall. It uses MaxWaitToCall as a kind of flag. If MaxWaitToCall is not zero, then it and MinWaitToCall set the bounds of the desired duration in minutes. On the other hand, if MaxWaitToCall is zero, then MinWaitToCall is the number of seconds to wait. Well, not quite. More precisely, the program knows that MinWaitToCall is the maximum number of seconds to wait (see the wording of the prompt). It assumes that the minimum wait is one second. It uses the Random() and Multiply() subroutines to pick a random number of seconds up to the number entered by the user.

The next couple of user entries are straight-forward. MaxNumCalls is the number of calls the Wardialer will make during this session. After it has made these calls, the program will enter a do-nothing loop until the user switches the device off.

MaxNumRings is the number of ringback pulses the procedure will wait through after it dials a telephone number. In the event that the telephone at the far end is answered, the program will hang up after waiting MaxSecsToHold seconds.

If the telephone at the other end is busy, then the program relies on three other parameters. MaxNumBusyTries is the maximum number of times the program will attempt to re-dial in the face of a busy signal. If the program hears a busy signal, it will immediately hang up. But, it will then attempt to re-dial. Up to 50 re-dials are permitted; the user's selection is stored in register MaxNumBusyTries. Just to be clear about it, this is the maximum number of attempts to make during one telephone call. If the callee's phone is busy on every try, then the program records this as a failed call, increments the running total in register NumBusyCalls, and then starts to wait for the net call.

A question is: how long to wait between these attempted re-dials when the callee's telephone is busy? The user is prompted to enter a maximum and a minimum wait time, assumed to be in seconds. These two values are stored in registers MaxWaitOnBusy and MinWaitOnBusy. The program handles these two parameters in the same way as the parameters that govern the wait time between calls made in the ordinary course. It calculates the difference, which it stores in register DeltaWaitOnBusy. When necessary it calls subroutine Random() to get a random number, which it multiplies by the difference, and then adds to the minimum, to calculate the number of seconds to wait before attempting a re-dial when busy. The subroutine CalcAndDoWaitOnBusy() does the randomness calculation, and also waits through the calculated number of seconds. This whole re-dialling procedure is stopped if, on one of the re-dials, the callee's telephone does ring through.

During the data-entry procedure, the PIC can display other messages. The following table lists these other messages.

Message		Follow-up
#3	Turn on speaker	Remind user to turn on the speaker
#7	Min minutes > Max	Re-prompt with "Mins between calls, max (0-255): _"
#18	Min seconds > Max	Re-prompt with "Secs between calls, max (1-255): _"

The last two messages alert the user that the cardinal order of the maximum and minimum wait times he has entered cannot be correct. These messages are displayed for three seconds after which the original prompts are displayed once more. No error messages are displayed if the user tries to enter one of the one-byte parameters with a value outside of the prescribed range. The PIC keeps the prompt displays until the user types in a satisfactory value.

To help the hapless programmer (which is to say, me) debug the data-entry procedure, I prepared a flowchart for the procedure, which is set out in the Appendices.

The PIC program – operating loop

As soon as the user has entered the last of the parameters, the program will commence its main operating loop. There is no delay before the PIC makes the first call. The following table sets out the messages that the PIC may display during operations.

Message	Meaning and follow-up
Line in use	Our telephone line is being used; wait one minute and try again
No dial tone	Not receiving a dial tone; wait one minute and try again
Dialling ...	PIC is dialling the telephone number
Busy ...	Telephone at far end is busy
Ringing ...	Telephone at far end is ringing
Dialling try #xxx	Attempt to re-dial a line that has been busy
Progress Good:xxx Busy:yyy Rang:zzz	Progress report
All finished	Procedure is finished

The first two messages will appear if the device is having trouble getting access to telephone service. It is possible that some other device that uses the same external line, such as a fax machine, is using the line. Or, for some reason, the device simply does not get a dial tone after the DPDT relay is closed. In either case, the PIC will display the message for one minute, then try again. Failure to make a call does not affect the statistics. That is, failure at our end does not reduce the planned number of outgoing calls.

While it is dialling the telephone number, the PIC will display Dialling...

Once the callee's line begins to respond, the PIC will display either Busy ... or Ringing ..., depending on what it hears. It is possible that the callee will answer the phone before a busy signal or ringback is generated at our end. If that happens, the PIC will proceed with its procedure for answered calls, and neither of these messages will be displayed.

If the callee's line is busy, the PIC will enter its procedure for dealing with busy lines. As it makes each attempt to re-dial, it will display the message Dialling try #xxx. The number xxx will be incremented on each attempt.

In between normal-course calls, the PIC will display a progress report. A "Good" call is defined as one where the callee actually answered, or "picked up". A "Rang" call is one where the callee's telephone rang through the prescribed number of ringback pulses, but the telephone was not answered. This is not to say that a "Good" or "Rang" call happened on the first attempt. It may well be that several busy signals and re-dials were necessary before the call eventually rang through. A "Busy" call is one where the prescribed MaxNumBusyTries was made, but the callee's line was busy on every one of them. The program keeps track of these outcomes in the three registers NumGoodCalls, NumRangCalls and NumBusyCalls. When the procedure is all finished, the sum of these three numbers will be equal to the number of calls the user requested in parameter MaxNumCalls.

To process these messages, the PIC needs to be able to take an 8-bit byte and display it on the LCD as a three-digit decimal number. Here's how the PIC does that. A call to subroutine ConvertBinaryToCharacterField() converts the contents of the *w* accumulator into three decimal digits in registers char1, char2 and char3. Next, a call to subroutine DisplayNumericField() displays these three characters on the LCD. An important associated variable is stored in register StartAdd. This is the address on the LCD display at which the numeric sequence is to start. For reference purposes, note that address 0x00 is the start of the first line; address 0x40 is the start of the second line. The LCD uses the ASCII representation of characters. This is helpful, particularly for numerals. The ASCII representation of a numeral is 0x20 plus the binary representation of the numeral. For example, 0x05 = d'5' has an ASCII representation of 0x25. Converting the binary contents of the char* registers for transmission to the LCD is therefore a simple logical operation.

Let me say a word about the random number generator. The algorithm used here is a type of linear-feedback shift register, of which there are many descriptions on the internet. The following is subroutine Random(). At every call, it updates a random 8-bit user register named Random, and for convenience also returns the updated number in the *w* register. The result is not truly random, of course, but pseudo-random. 256 successive calls to Random() will return the 256 numbers d'0' through d'255' in what appears to be random order. Although not truly random, the lengths of the wait times that this subroutine returns are more than adequate to mislead the callee (even if AI) into believing that the caller is a human. Incidentally, register Random needs to be seeded before its first use. For no particular reason, I seeded it with the value 0x55.

```
Random
        rlf     random,w
        rlf     random,w
        btfsc   random,4
        xorlw   0x01
        btfsc   random,5
        xorlw   0x01
        btfsc   random,3
        xorlw   0x01
        movwf   random
        return
```

To help the hapless programmer (once gain, me) debug the operating loop, I prepared a flowchart, which is set out in the Appendices.

The PIC program – detecting call progress tones

Although the two LM576 filters provide the raw data for detecting DTMF call progress tones, some software is required as well. The presence or absence of 440Hz and 480Hz components in the call progress tones need to be interpreted into something more meaningful. Subroutine ReadCallProgress() does the heavy lifting. It return its findings by setting bits in user-register RetValues, whose name signifies that it holds values returned by subroutines. RetValues has separate bits to flag the presence of dial tones, busy tones, ringback tones, and silence. A flowchart for this subroutine is set out in the Appendices.

Construction

The circuit is built on two printed circuit boards (PCBs). The telephone line interface (Schematic #1) is built on one. The telephone line control circuits (Schematic #2) and the data entry and display circuits (Schematic #3) are both built on the second. Two PCBs were needed in order to make everything fit inside the enclosure. The latter PCB is the bigger of the two, and is mounted underneath the former in the enclosure. The second, third and fourth photographs in the Appenices show the arrangement.

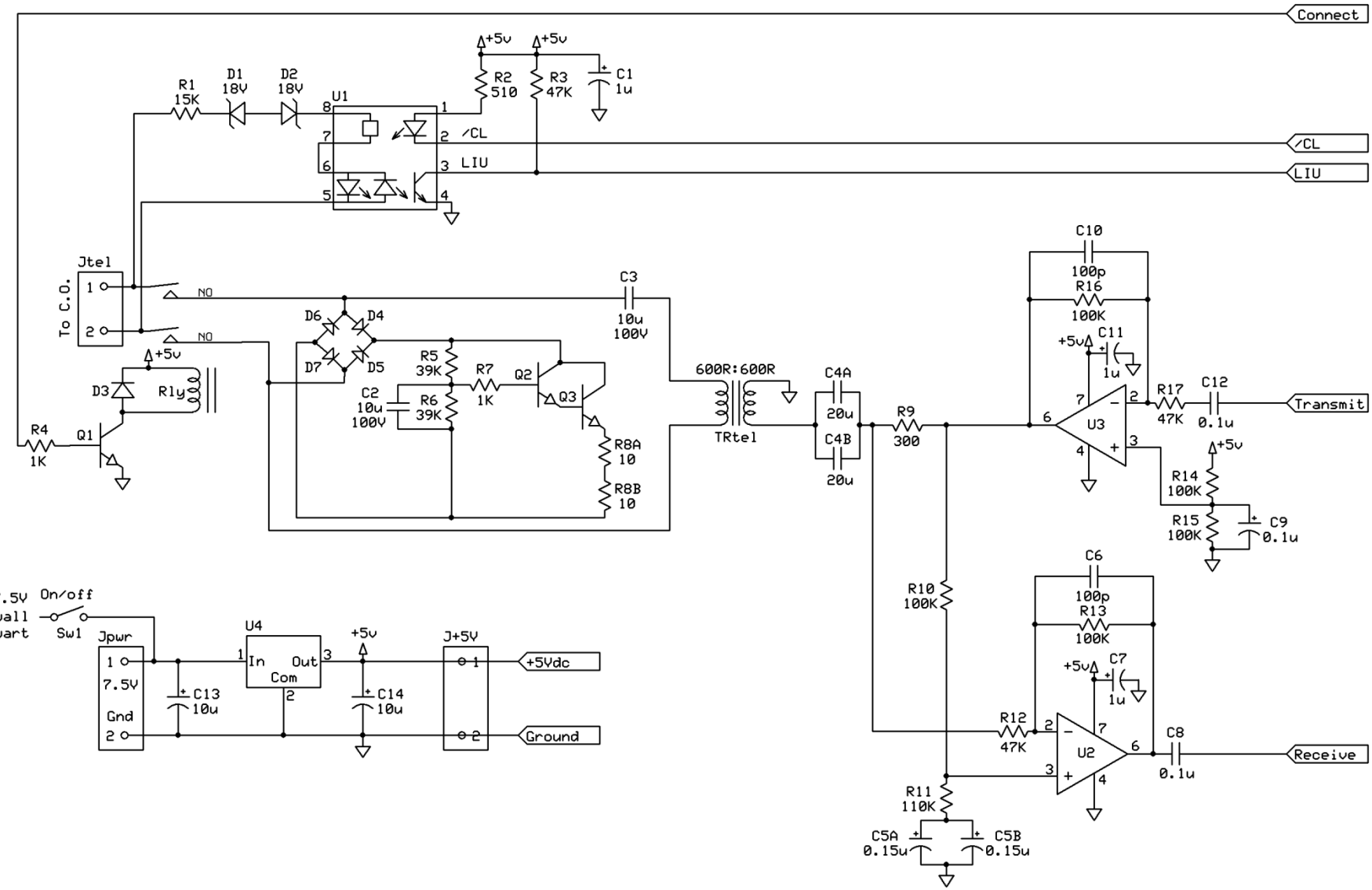
The lid of the enclosure is hinged on the left hand side. The LCD display, keypad and control switches are all mounted on and through the lid. The last photograph in the Appendices shows how access is gained to the rear of these controls.

February 2019

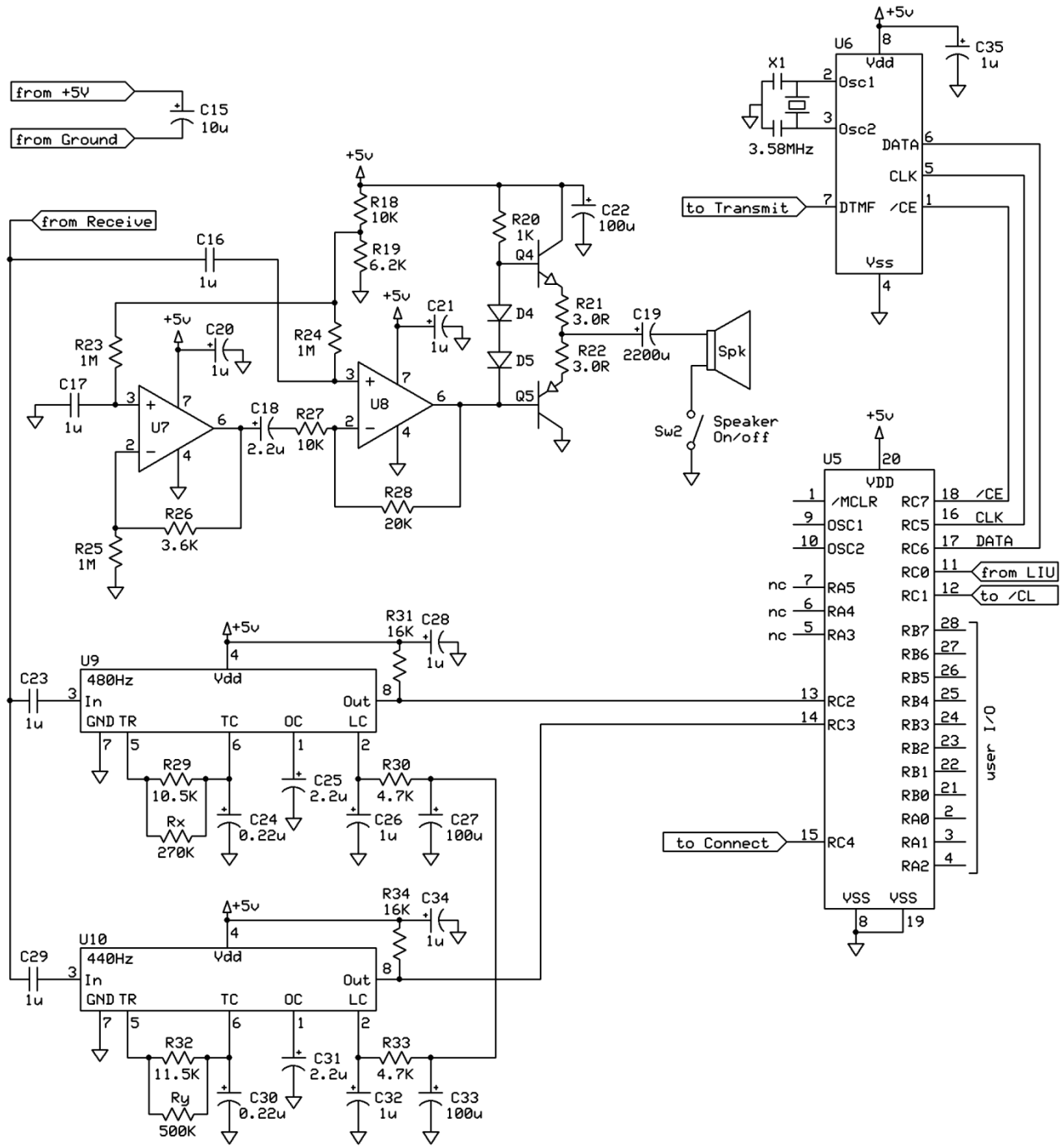
Jim Hawley
jim@jimhawley.ca or hawley@pathcom.com

(As always, an e-mail setting out errors or omissions would be welcome)

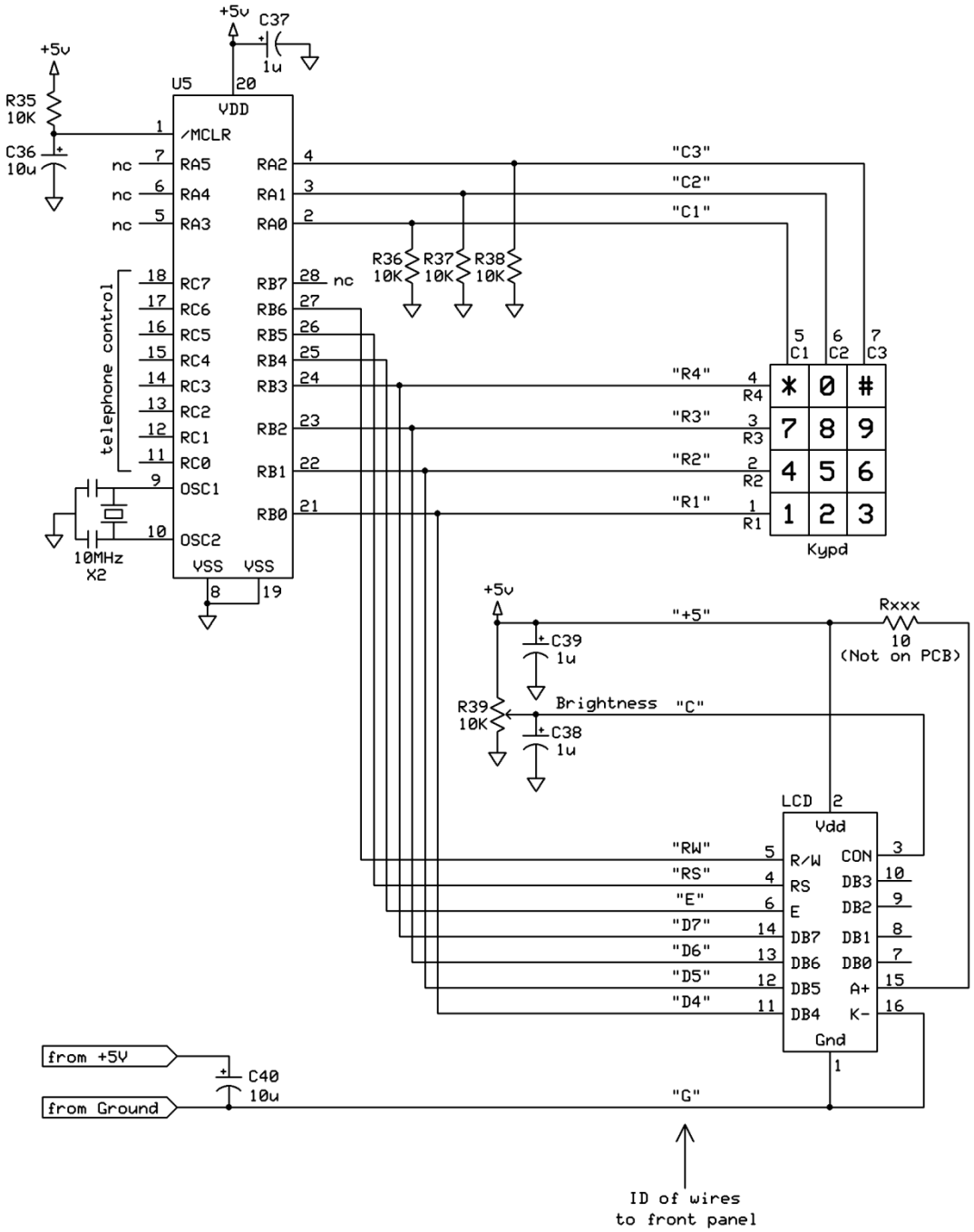
List of Appendices



Wardialer Schematic #1		
Telephone line interface		
Jim Hawley		25/07/16



Wardialer Schematic #2		
Telephone line control		
Jim Hawley		25/07/16



Wardialer Schematic #3		
Data entry and display		
Jim Hawley		25/07/16

Newark (NW) and Digikey (DK) part numbers

Components in Schematic #1
U1 = DK:CLA108-ND Clare TS117 multi-function telecom relay 8-DIP
U2,U3 = DK:OPA340PA-ND general purpose OpAmp
U4 = DK:497-1404-5-ND 5V 15A low-dropout voltage regulator TO-220
Q1-Q3 = DK:P2N2222AGOS-ND general purpose NPN transistor
D1,D2 = DK:1N5355BTPMSCT-ND 18V 5W zener diode DO-15
D3 = DK:1N914ACT-ND 100V 200mA diode DO-35
D4-D7 = DK:1N4001GOS-ND 50V 1A diode DO-41
Rly = DK:PB288-ND DPDT relay 5V 2A contacts
TRtel = DK:MT7207-ND 600:600 telecom transformer
C1,C7,C11 = DK:478-1835-ND 1uF 10% 35V tantalum
C2,C3 = NW:49W0224 10uF 5% 100V PET(polyester)
C4A,C4B = DK:445-173156-1-ND 10uF 10% 25V ceramic
C5A,C5B = DK:478-1848-ND 0.15uF 10% 35V tantalum
C6,C10 = DK:BC5223CT-ND 100pF 5% 50V ceramic
C8,C12 = DK:478-3188-ND 0.1uF 10% 50V ceramic
C9 = DK:478-1831-ND 0.1uF 10% 35V tantalum
C13-C14 = DK:478-4171-ND 10uF 10% 35V tantalum
Sw1 = DK:360-3237-ND miniature SPST toggle switch
Jtel,Jpwr,J+5V = assembled from DK:281-1435-ND 2-pin 0.2-inch terminal strips
Resistors all 1% 1/4W

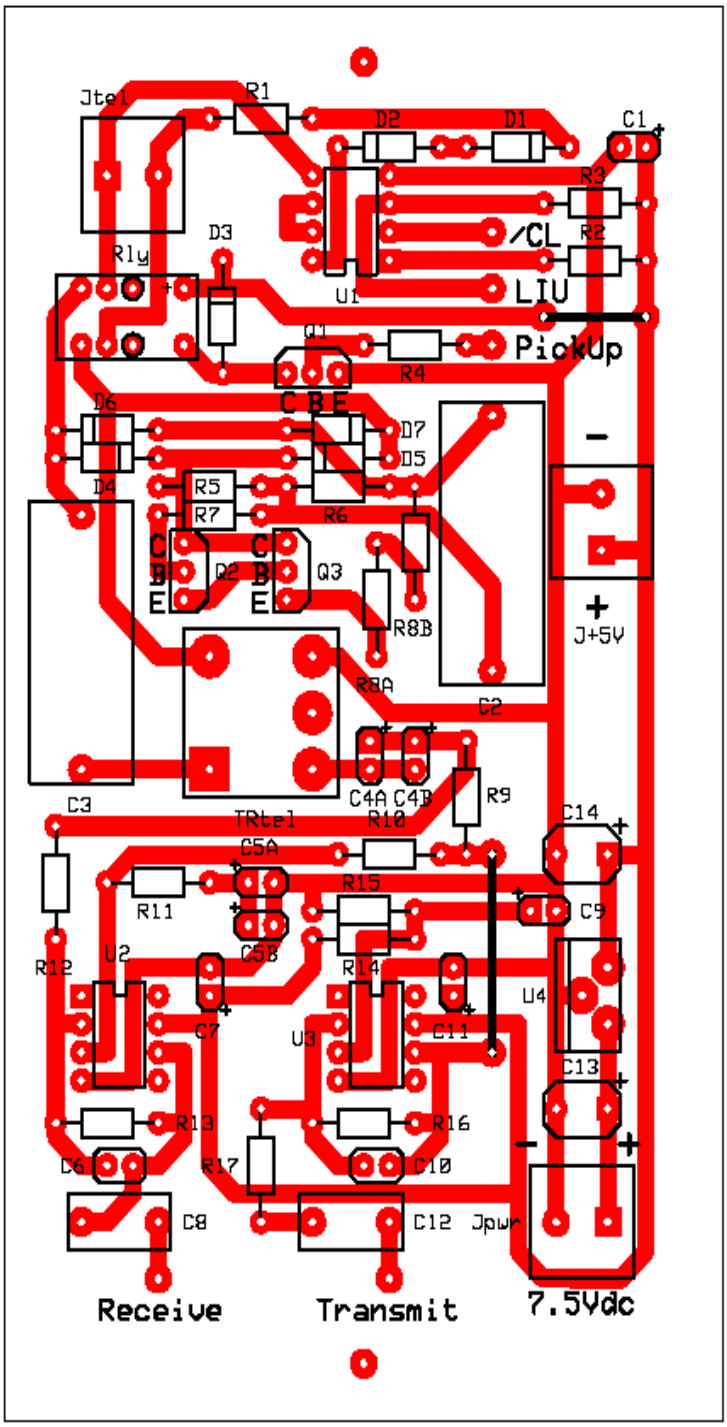
Components in Schematic #2
U5 = NW:12C1945 PIC16F882-I/SP 2Kx14 20MHz 22I/O E64x8 RAM128x8
U6 = NW:53M7848 Holtek HT9200A DTMF generator 8-SOP
U7,U8 = DK:OPA340PA-ND general purpose OpAmp
U9,U10 = DK:296-36350-5-ND LM567 tone decoder 8-DIP
X1 = DK:490-1207-ND 3.58MHz 0.2% ceramic resonator with built-in capacitors
D4-D5 = DK:1N4001GOS-ND 50V 1A diode DO-41
C15 = DK:478-4171-ND 10uF 10% 35V tantalum
C16,C17,C23,C29 = DK:445-173583-1-ND 1uF 10% 25V ceramic
C18,C25,C31 = DK:478-8819-ND 2.2uF 10% 16V tantalum
C19 = DK:478-5430-1-ND 2200uF 20% 6.3V tantalum surface mount 6030
C20,C21,C26,C28,C32,C34,C35 = DK:478-1835-ND 1uF 10% 35V tantalum
C22,C27,C33 = NW:88H3775 100uF 10% 10V tantalum
C24,C30 = DK:478-8967-1-ND 0.22uF 10% 35V tantalum
Sw2 = DK:360-3237-ND miniature SPST toggle switch
Spk = DK:668-1136-ND 8ohm 150mW 500Hz-20KHz
Resistors all 1% 1/4W

Components in Schematic #3
Kypd = NW:14M6876 3x4-key front panel mount
LCD = NW:19J7674 2x20-character backlit
X2 = DK:490-1213-ND 10MHz ceraminc resonator with built-in capacitors
C36,C40 = DK:478-4171-ND 10uF 10% 35V tantalum
C37-C39 = DK:478-1835-ND 1uF 10% 35V tantalum
R39 = NW:62J2249 10K one-turn trimpot top adjust
Resistors all 1% 1/4W

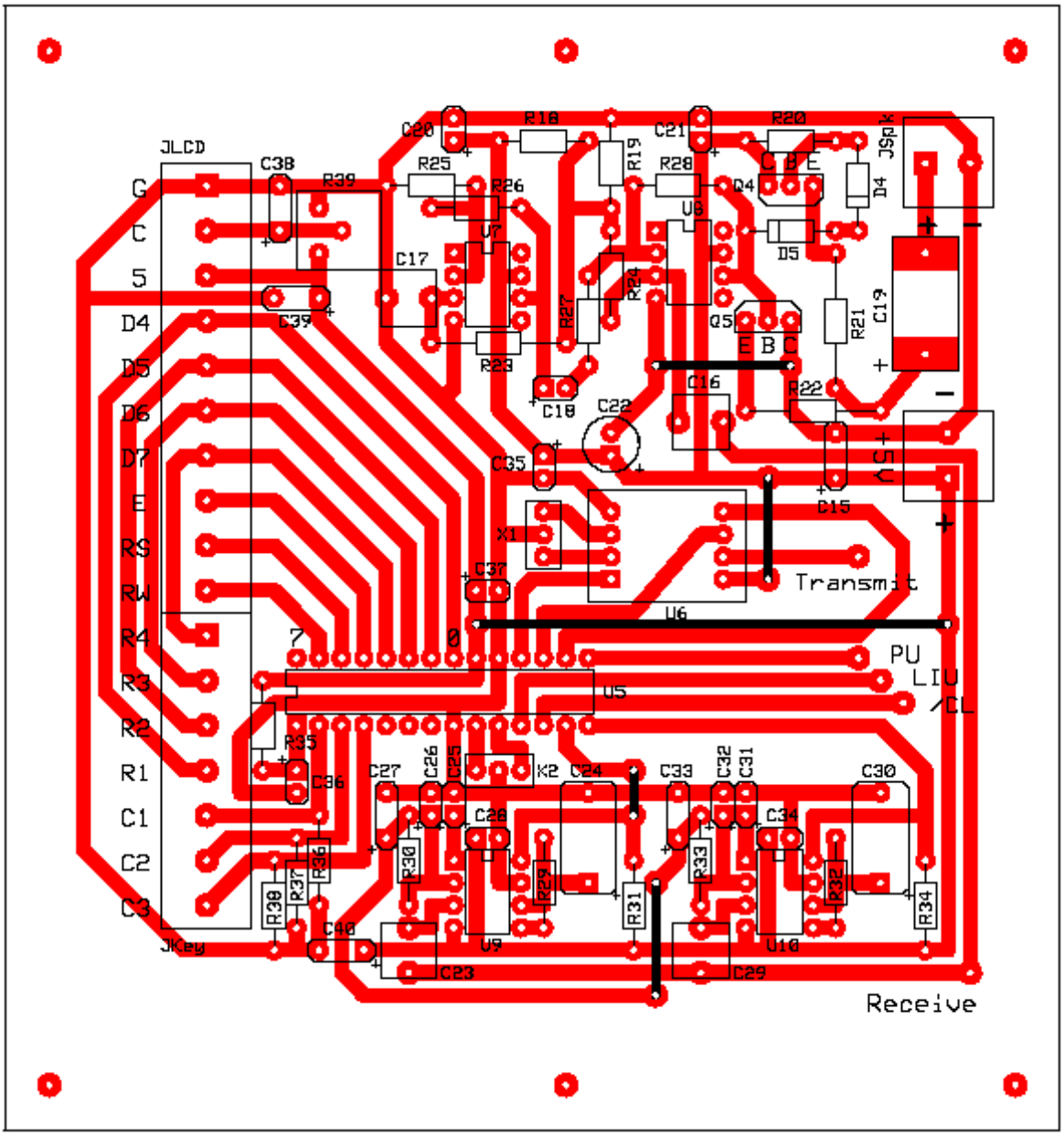
Other parts
Enclosure = DK:377-1765-ND 5.9 x 5.9 x 3.54 inches ABS with hinged door
Wall adapter = DK:237-1423-ND 7.5V 1A
29-pin DIP socket for U5 = DK:ED3328-ND

Wardialer		
Parts list		
Jim Hawley		2018 Jan 30

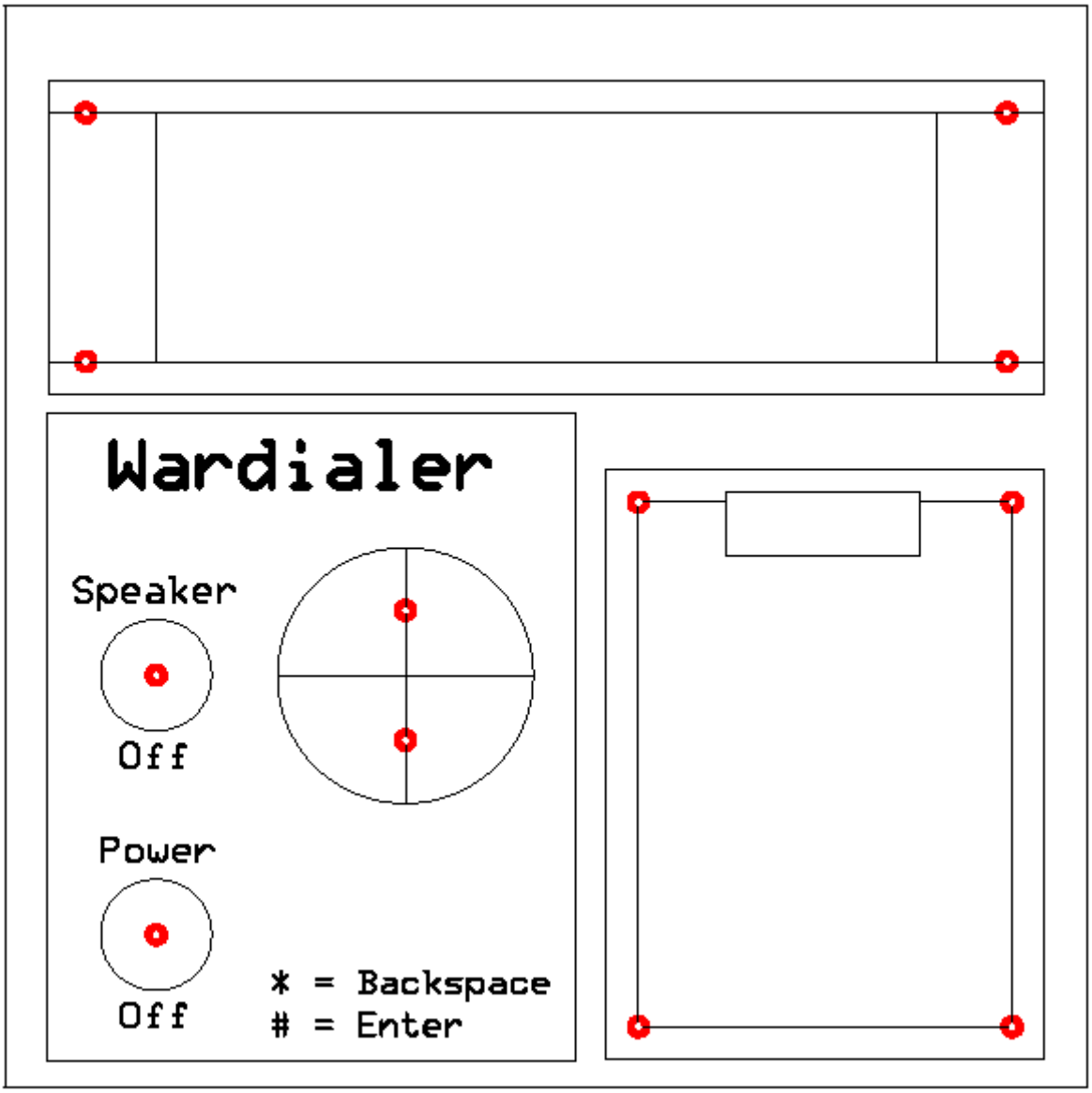
PCB#1 – 2.8” wide by 5.0” high
(Includes circuit in Schematic #1)



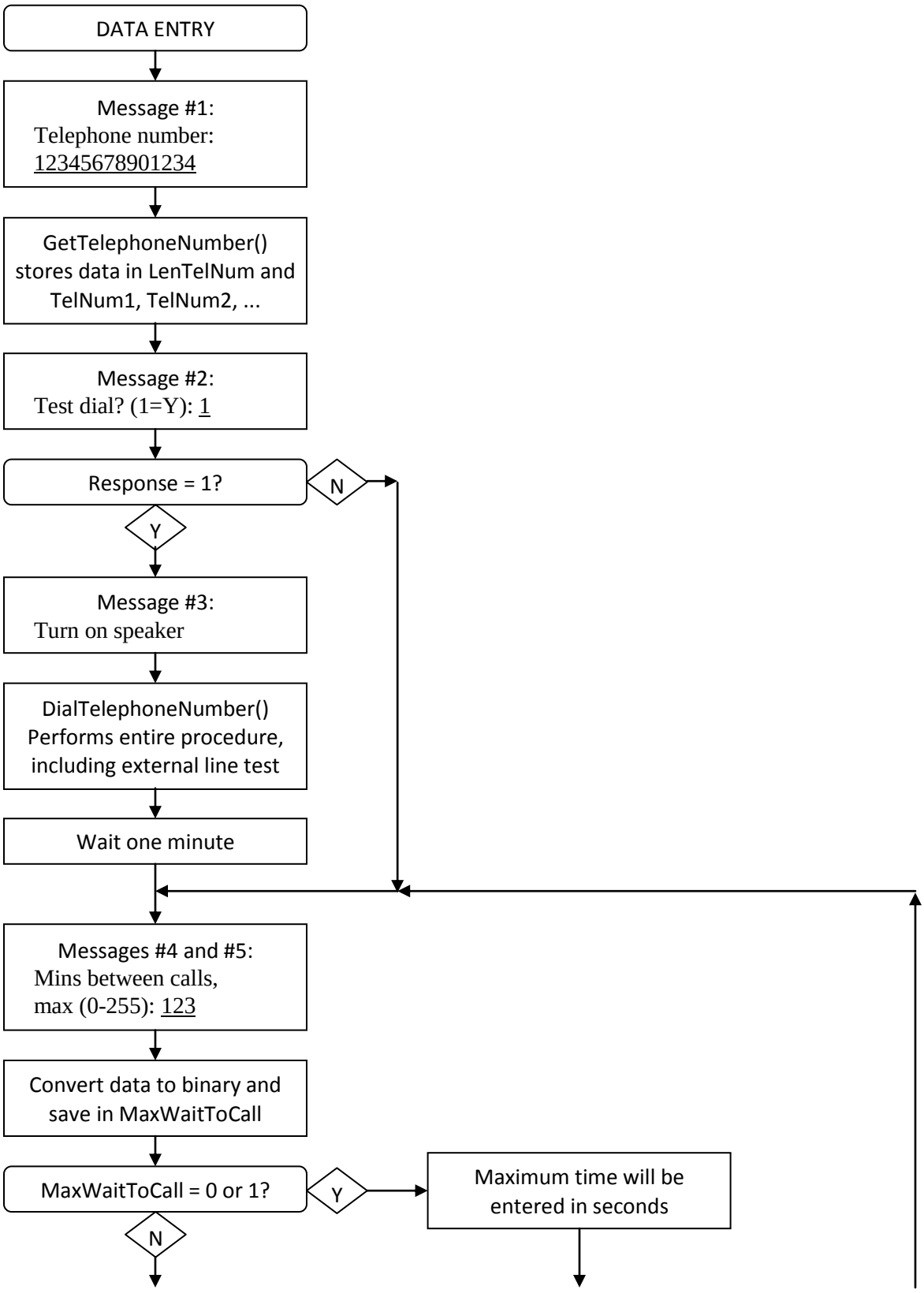
PCB#2 – 4.7” wide by 5.0” high
(Includes circuits in Schematics #1 and #2)

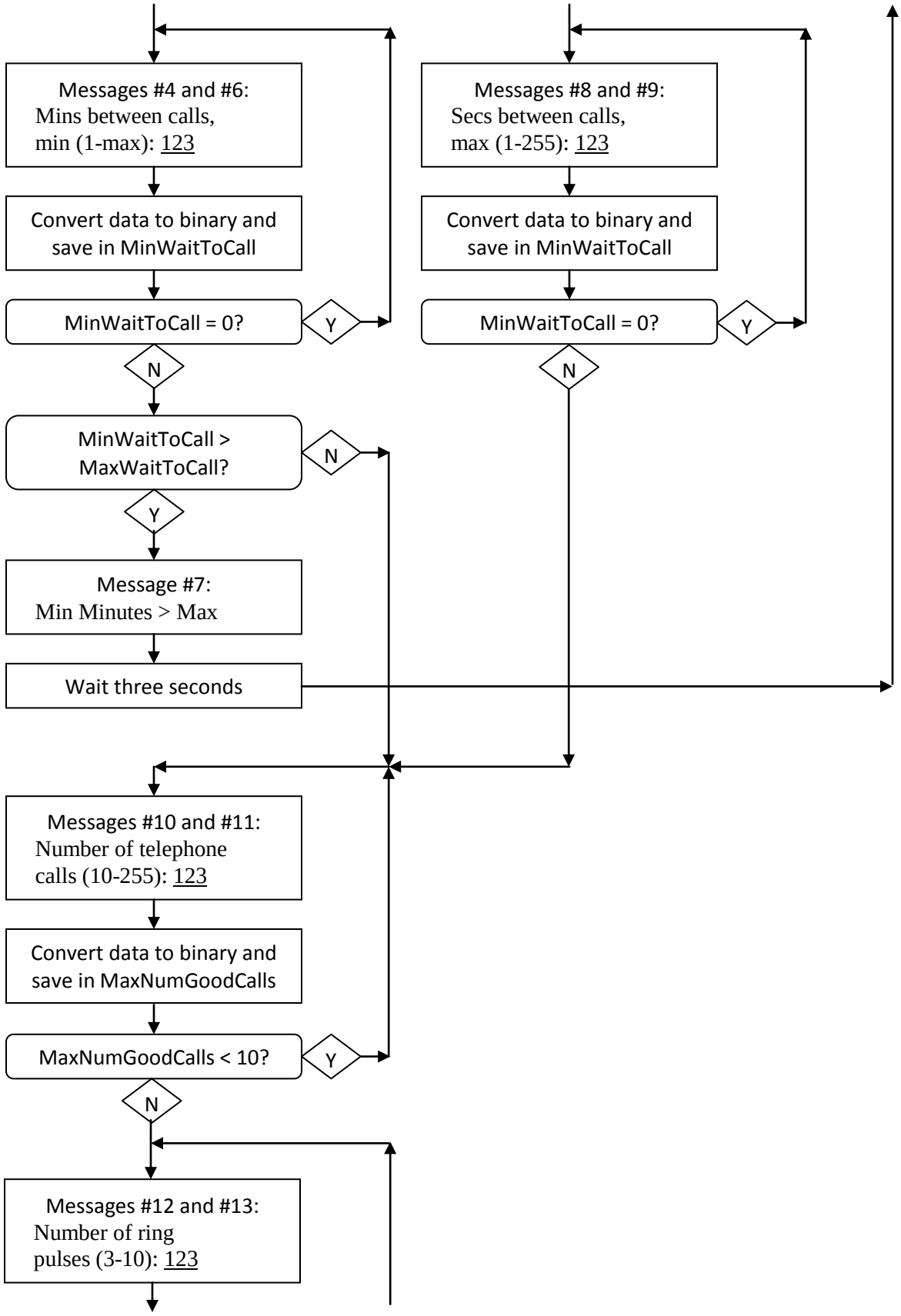


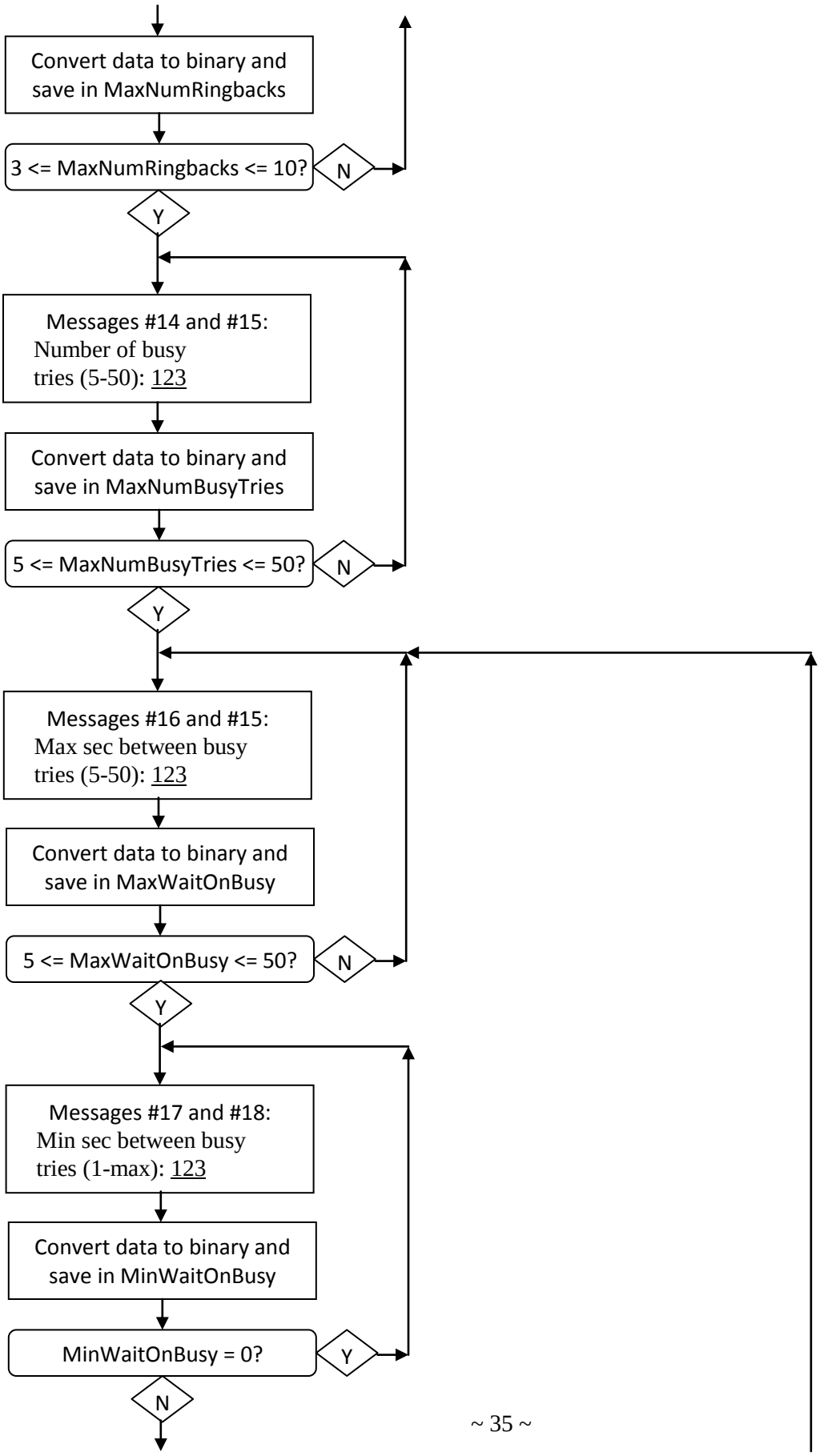
Front panel layout – 5.0” wide by 5.0” high

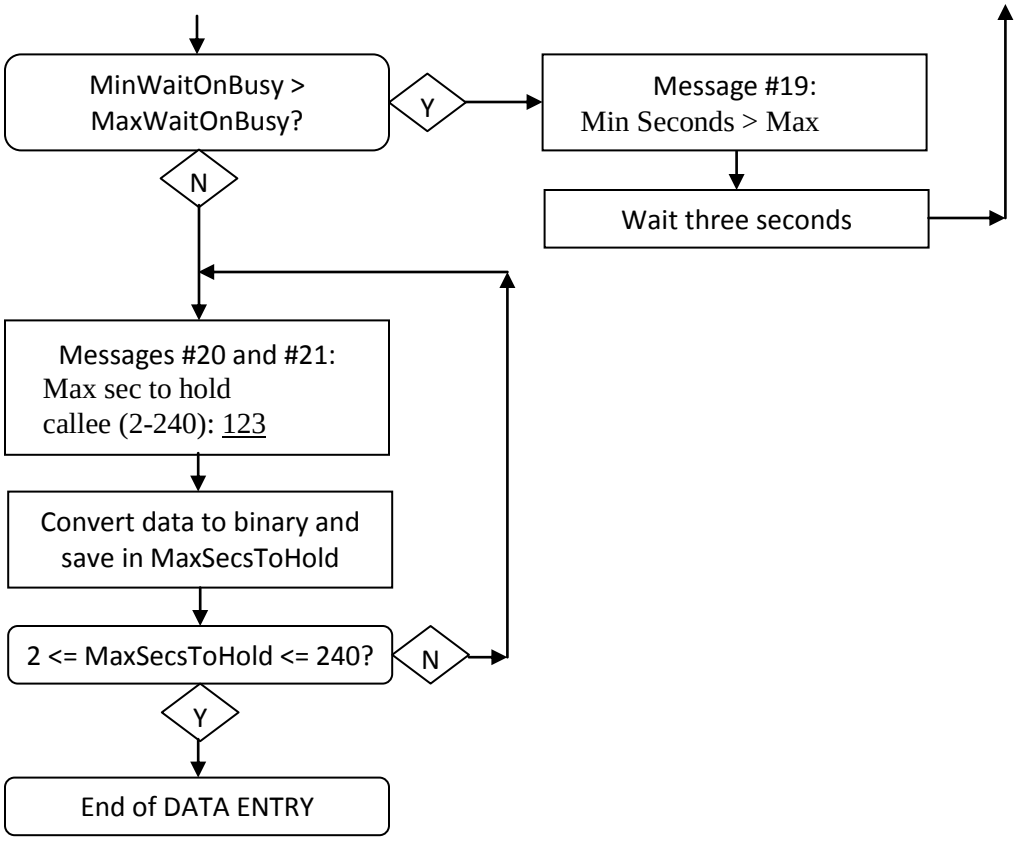


Flowchart for Data-Entry procedure

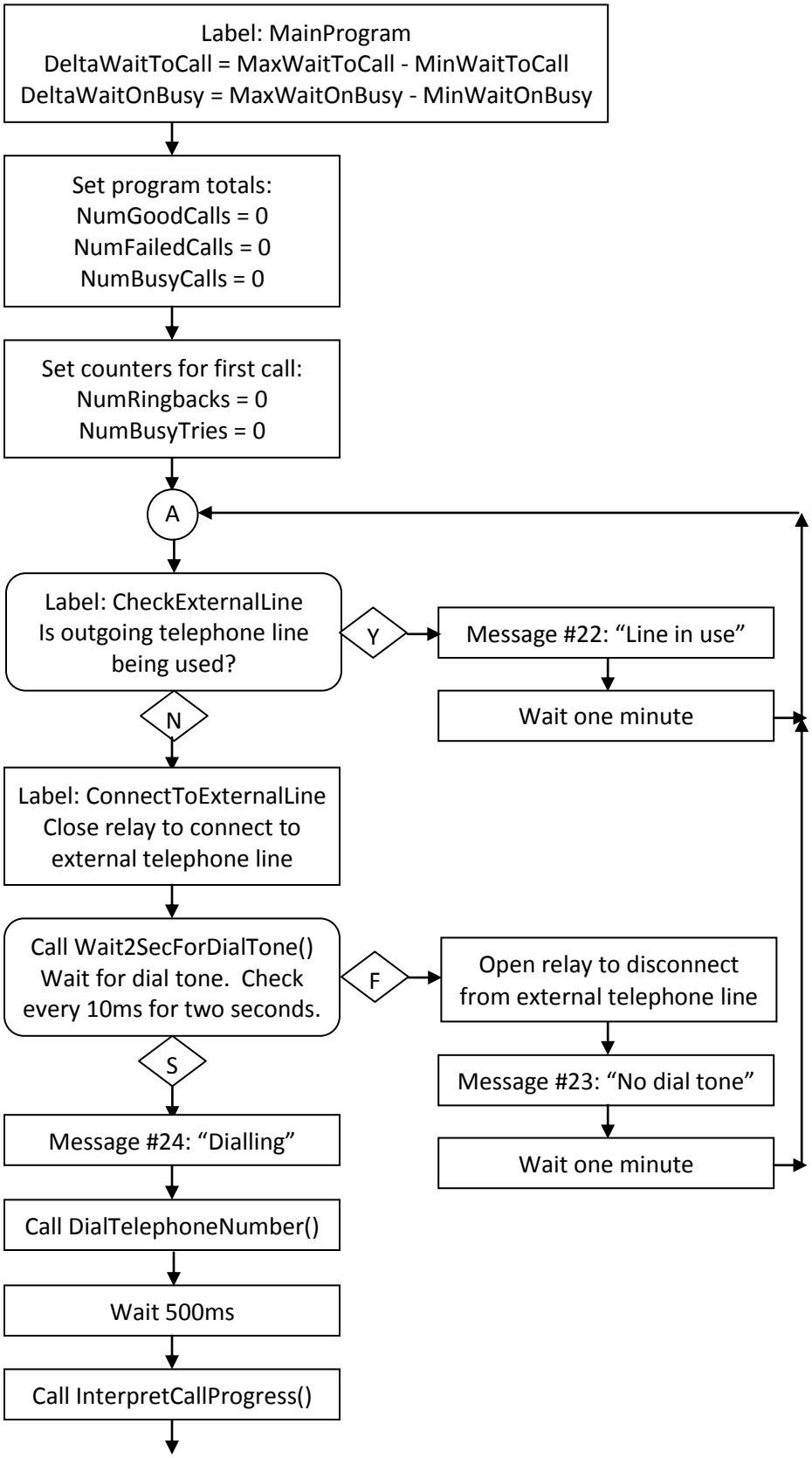


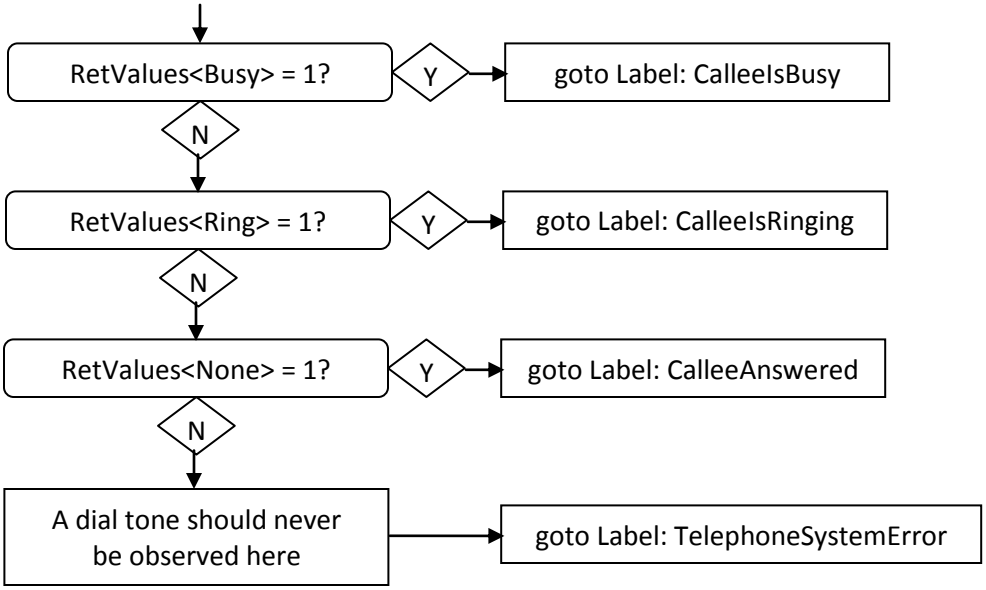

Messages #4 and #6:
Mins between calls,
min (1-max): 123
Convert data to binary and save in MinWaitToCall
MinWaitToCall = 0?
Y
N
MinWaitToCall > MaxWaitToCall?
N
Y
Message #7:
Min Minutes > Max
Wait three seconds
Messages #8 and #9:
Secs between calls,
max (1-255): 123
Convert data to binary and save in MinWaitToCall
MinWaitToCall = 0?
Y
N
Messages #10 and #11:
Number of telephone calls (10-255): 123
Convert data to binary and save in MaxNumGoodCalls
MaxNumGoodCalls < 10?
Y
N
Messages #12 and #13:
Number of ring pulses (3-10): 123

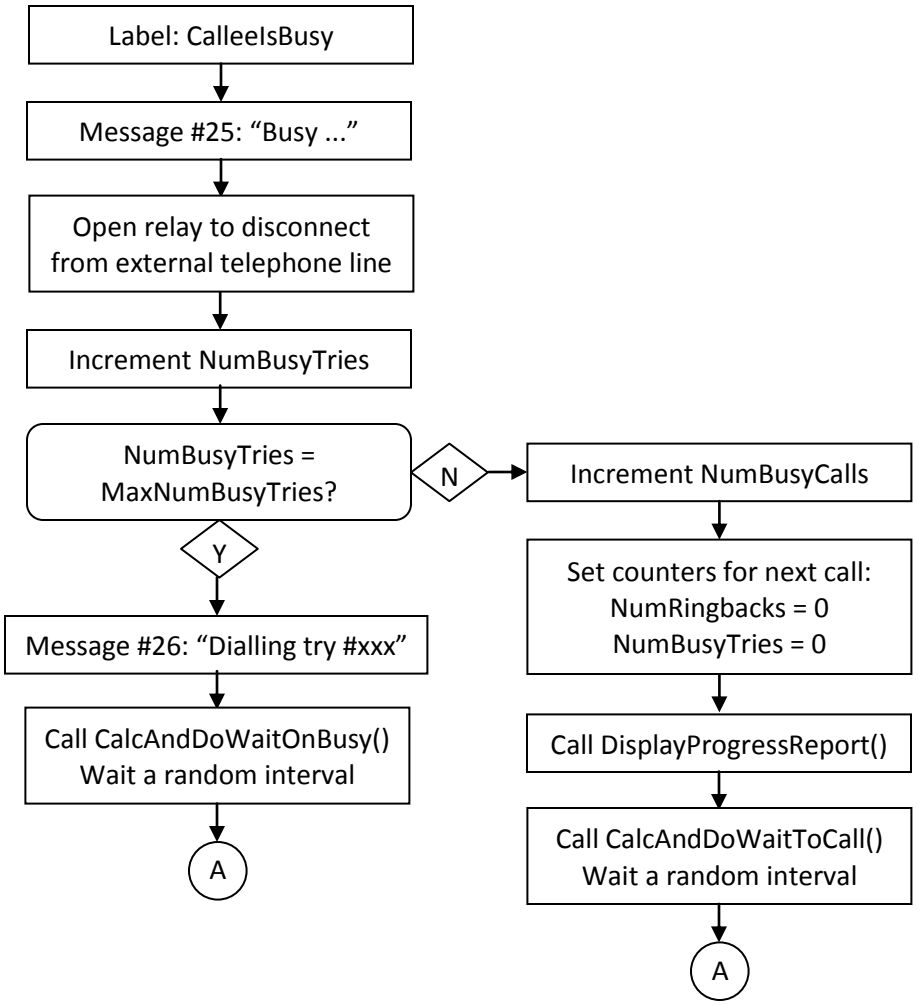

Convert data to binary and save in MaxNumRingbacks
3 <= MaxNumRingbacks <= 10?
N
Y
Messages #14 and #15: Number of busy tries (5-50): 123
Convert data to binary and save in MaxNumBusyTries
5 <= MaxNumBusyTries <= 50?
N
Y
Messages #16 and #15: Max sec between busy tries (5-50): 123
Convert data to binary and save in MaxWaitOnBusy
5 <= MaxWaitOnBusy <= 50?
N
Y
Messages #17 and #18: Min sec between busy tries (1-max): 123
Convert data to binary and save in MinWaitOnBusy
MinWaitOnBusy = 0?
Y
N

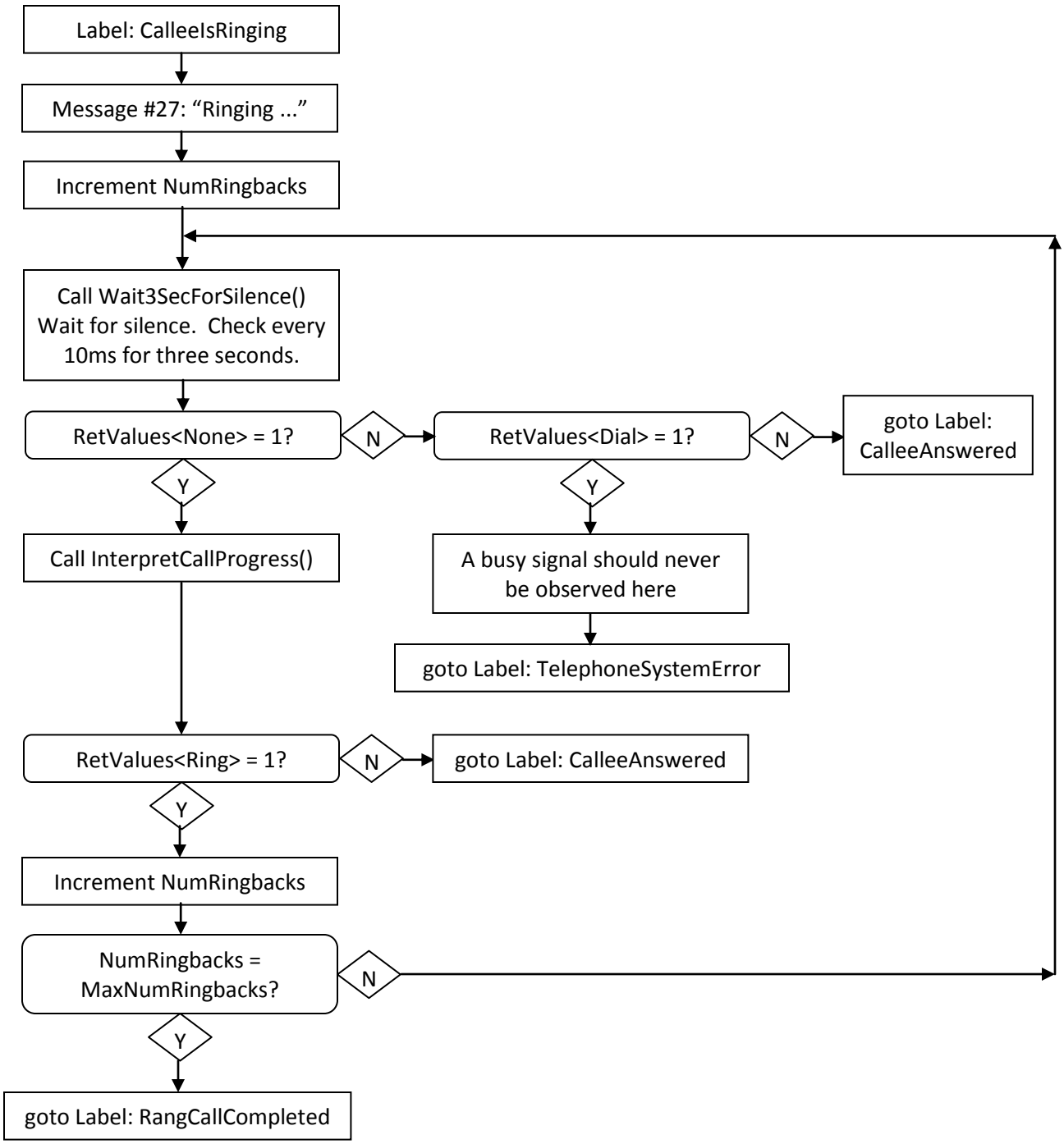

MinWaitOnBusy > MaxWaitOnBusy?
Y
Message #19: Min Seconds > Max
Wait three seconds
N
Messages #20 and #21: Max sec to hold callee (2-240): 123
Convert data to binary and save in MaxSecsToHold
2 <= MaxSecsToHold <= 240?
N
Y
End of DATA ENTRY

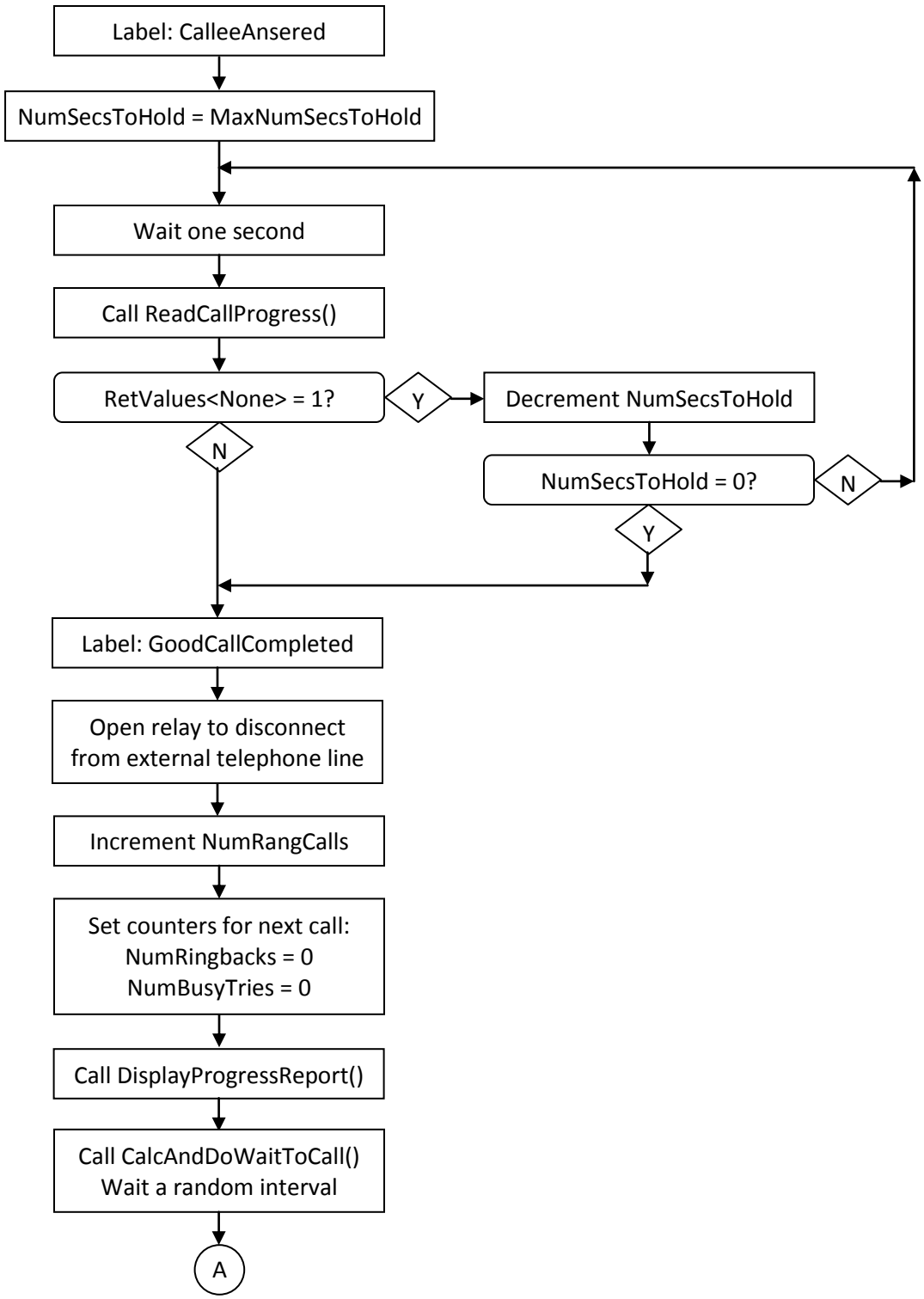
Flowchart for making a telephone call




RetValues<Busy> = 1?
Y
goto Label: CalleeIsBusy
N
RetValues<Ring> = 1?
Y
goto Label: CalleeIsRinging
N
RetValues<None> = 1?
Y
goto Label: CalleeAnswered
N
A dial tone should never be observed here
goto Label: TelephoneSystemError


Label: CalleeIsBusy
Message #25: “Busy ...”
Open relay to disconnect from external telephone line
Increment NumBusyTries
NumBusyTries = MaxNumBusyTries?
N
Increment NumBusyCalls
Set counters for next call:
NumRingbacks = 0
NumBusyTries = 0
Call DisplayProgressReport()
Call CalcAndDoWaitToCall()
Wait a random interval
A
Y
Message #26: “Dialling try #xxx”
Call CalcAndDoWaitOnBusy()
Wait a random interval
A


Label: CalleeIsRinging
Message #27: “Ringing ...”
Increment NumRingbacks
Call Wait3SecForSilence()
Wait for silence. Check every 10ms for three seconds.
RetValues<None> = 1?
N
RetValues<Dial> = 1?
N
goto Label: CalleeAnswered
Y
Y
Call InterpretCallProgress()
A busy signal should never be observed here
goto Label: TelephoneSystemError
RetValues<Ring> = 1?
N
goto Label: CalleeAnswered
Y
Increment NumRingbacks
NumRingbacks = MaxNumRingbacks?
N
Y
goto Label: RangCallCompleted

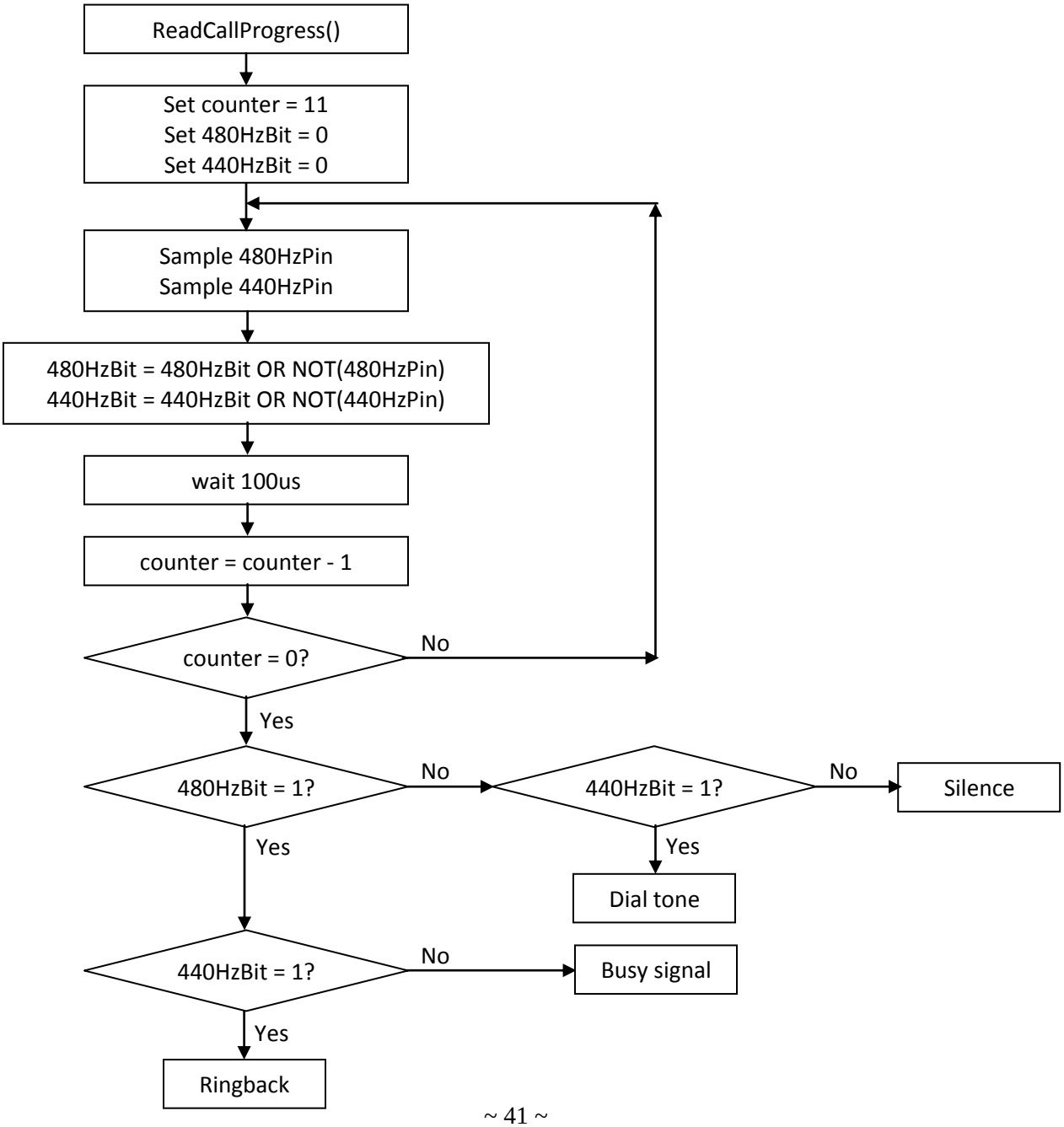

Label: CalleeAnsered
NumSecsToHold = MaxNumSecsToHold
Wait one second
Call ReadCallProgress()
RetValues<None> = 1?
Y
Decrement NumSecsToHold
N
NumSecsToHold = 0?
N
Y
Label: GoodCallCompleted
Open relay to disconnect from external telephone line
Increment NumRangCalls
Set counters for next call:
NumRingbacks = 0
NumBusyTries = 0
Call DisplayProgressReport()
Call CalcAndDoWaitToCall()
Wait a random interval
A

Flowchart for subroutine ReadCallProgress()

CP480HzPin = portC<2> input signal from 480Hz filter
CP 440HzPin = portC<3> input signal from 440Hz filter

CP480HzBit = tempRCP2<2> of user register tempRCP2
CP440HzBit = tempRCP2<3> of user register tempRCP2

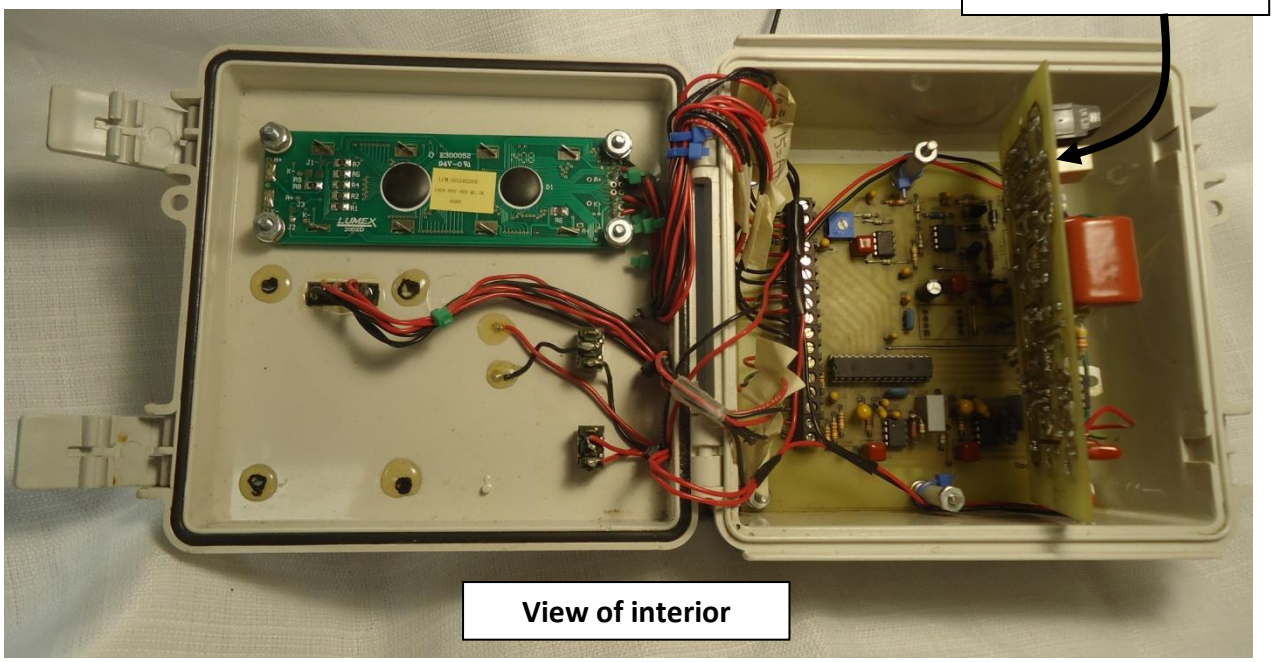
RetValues<7> = 1 denotes a busy signal
RetValues<6> = 1 denotes a ringback tone
RetValues<5> = 1 denotes a dial tone
RetValues<4> = 1 denotes silence



Photographs

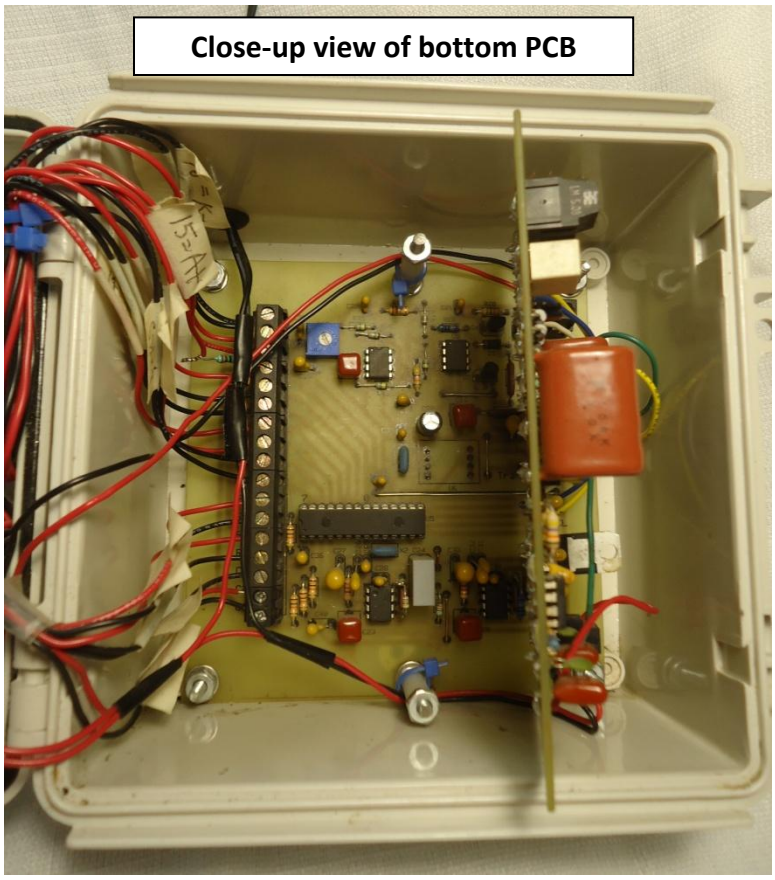
Finished unit during entry of telephone number

Top PCB swung up to show bottom PCB

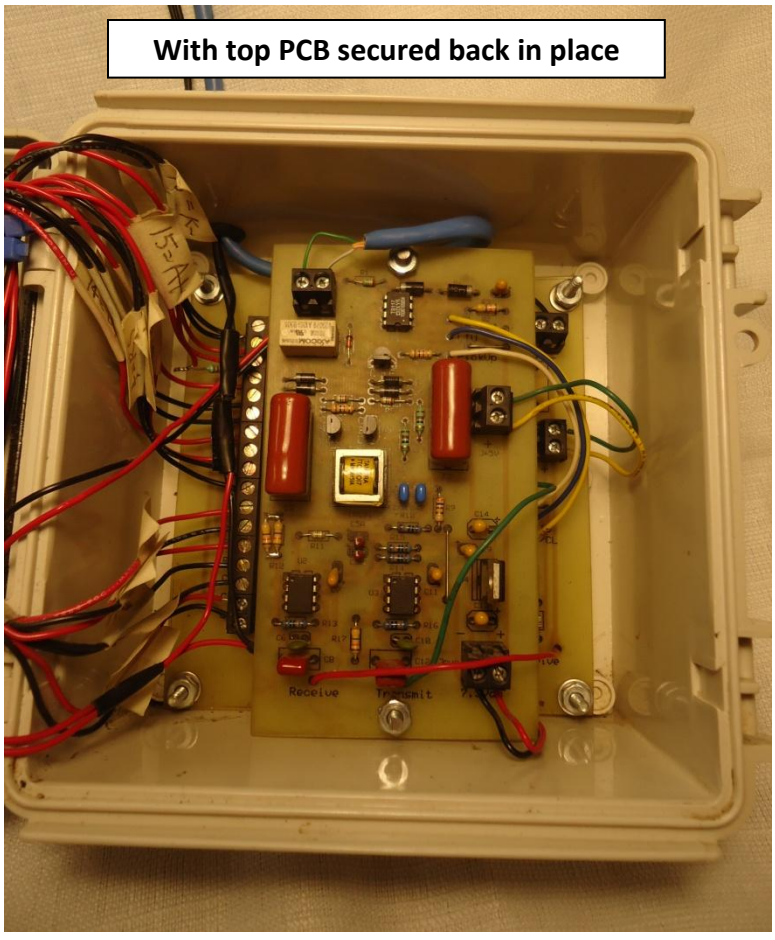


View of interior

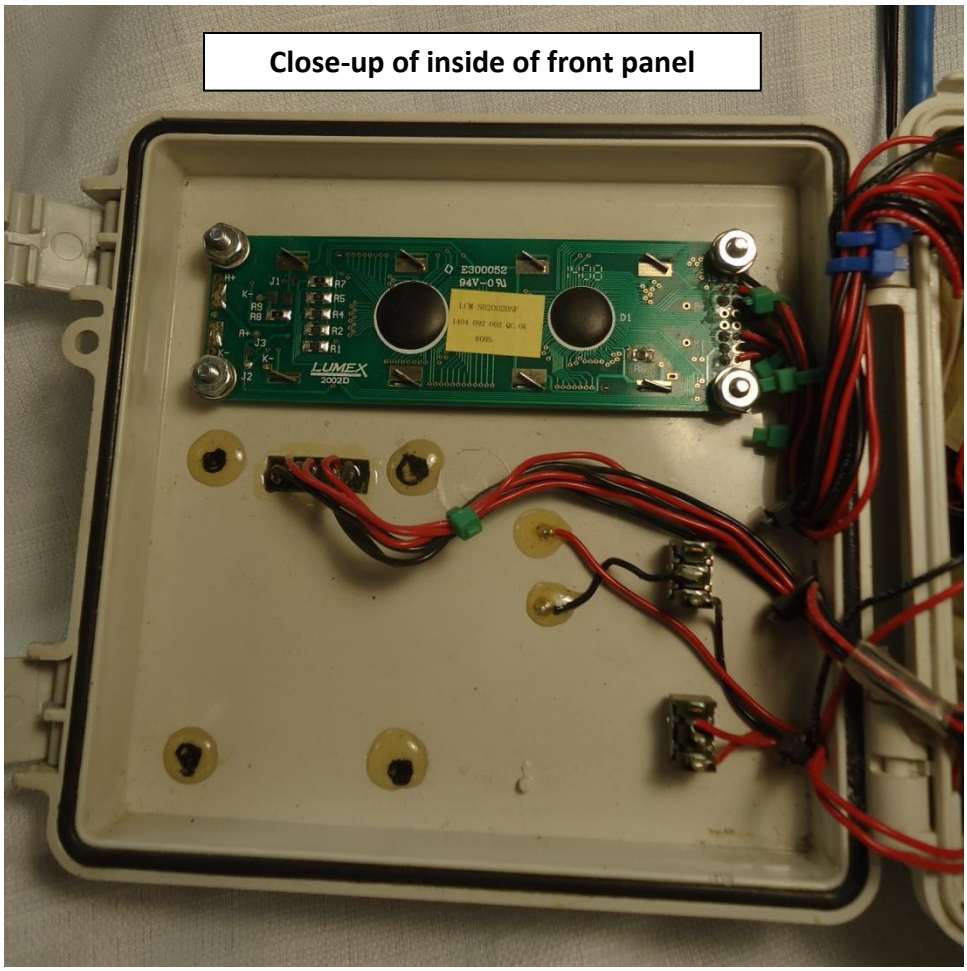
Close-up view of bottom PCB



With top PCB secured back in place



Close-up of inside of front panel



Listing of assembly code for PIC 16F882

```
; Program for Wardialer, for 16F882 microprocessor
;
; 1. The duration of DTMF tones and pauses is fixed at 500ms.  This is very
;    conservative, but little would be gained by speed dialling.
;
; 2. This application does not use interrupts.
;
; 3. The code has ten main blocks:
;    A.  The main program, which starts on power-up, and handles data entry
;    B.  The main operating program, which runs the dialler
;    C.  Routines needed to operate the LCD hardware
;    D.  Routines needed to operate the keypad hardware
;    E.  Routines needed to operate the telephone line hardware
;    F.  Routines which send messages to the LCD
;    G.  Routines which interpret key strokes
;    H.  Miscellaneous subroutines
;    I.  Routines used for debugging
;    J.  ASCII look-up tables with prompts and messages
;
; Configuration Words for 16F882
;       b<13>=1         Disable in-circuit debugger
;       b<12>=0         Disable Low-Voltage Programming
;       b<11>=0         Disable fail-safe clock monitor
;       b<10>=0         Disable internal/external switchover
;       b<9-8>=00       Disable brown-out reset
;       b<7>=1          Turn OFF EEPROM memory protection
;       b<6>=1          Turn OFF program memory protection
;       b<5>=1          Set standard /MCLR operation
;       b<4>=1          Disable power-up timer
;       b<3>=0          Disable watch-dog timer
;       b<2-0>=010      Set HS oscillator gain
  #include "p16F882.inc"
  processor   16F882
  __CONFIG _CONFIG1, 0x20F2 ; b'xx10 0000 1111 0010'
  __CONFIG _CONFIG2, 0x3FFF
;
; Crystal frequency is 10MHz, so the instruction cycle time is 400ns.
;
; *****************************************************************************
; Variable definitions - PIC 16F882 control registers
; *****************************************************************************
;
; Registers in Bank0
TMR0          equ   0x01        ; Timer0 count register
PCL           equ   0x02        ; Program counter, low byte
status        equ   0x03        ; Status register
carry         equ         0x00  ; carry from MSB occurred
zero          equ         0x02  ; result of operation is zero
page0         equ         0x05  ; register bank selector low bit
page1         equ         0x06  ; register bank selector high bit
portA         equ   0x05
portB         equ   0x06
portC         equ   0x07
```

```
PCLATH          equ   0x0A        ; Program counter, high byte
INTCON          equ   0x0B        ; Interrupt control register
T1CON           equ   0x10        ; Timer1 control register
SSPCON          equ   0x14        ; Synch serial port control register 1
CCP1CON         equ   0x17        ; Capture/compare/PWM control register 1
RCSTA           equ   0x18        ; Receive status and control register
CCP2CON         equ   0x1D        ; Capture/compare/PWM control register 2
ADCON0          equ   0x1F        ; Analogue-to-digital control register 0
;
; Registers in Bank1
OPTION_REG      equ   0x81        ; Option register
TRISA           equ   0x85        ; portA pin I/O direction
TRISB           equ   0x86        ; portB pin I/O direction
TRISC           equ   0x87        ; portC pin I/O direction
PCON            equ   0x8E        ; Power control register
WPUB            equ   0x95        ; Weak pull-up portB register
IOCB            equ   0x96        ; Interrupt-on-change portB
PSTRCON         equ   0x9D        ; Pulse steering control
;
; Registers in Bank2
CM1CON0         equ   0x107       ; Comparator C1 control register 0
CM2CON0         equ   0x108       ; Comparator C2 control register 0
CM2CON1         equ   0x109       ; Comparator C2 control register 1
;
; Registers in Bank3
ANSEL           equ   0x188       ; Analogue select low register
ANSELH          equ   0x189       ; Analogue select high register
;
f               equ         0x01  ; f and w identify destination register
w               equ         0x00
;
; ***************************************************************************
; Variable definitions - User RAM - Accessible only in bank0
; ***************************************************************************
;
; I/O ports
portAmirror     equ   0x20
Col1            equ         0x00  ; Input - Keypad column #1
Col2            equ         0x01  ; Input - Keypad column #2
Col3            equ         0x02  ; Input - Keypad column #3
ncRA3           equ         0x03  ; Output - n/c
ncRA4           equ         0x04  ; Output - n/c
ncRA5           equ         0x05  ; Output - n/c

portBmirror     equ   0x21
R1D4            equ         0x00  ; Output - Keypad row #1; LCD data #4
R2D5            equ         0x01  ; Output - Keypad row #2; LCD data #5
R3D6            equ         0x02  ; Output - Keypad row #3; LCD data #6
R4D7            equ         0x03  ; Output - Keypad row #4; LCD data #7
LCDE            equ         0x04  ; Output - LCD control line "E"
LCDRS           equ         0x05  ; Output - LCD control line "RS"
LCDRW           equ         0x06  ; Output - LCD control line "R/W"
ncRB7           equ         0x07  ; Output - n/c

portCmirror     equ   0x22
LIU             equ         0x00  ; Input - LOW if telephone line is free
CheckLine       equ         0x01  ; Output - LOW to check status of line
```

```
CP480Hz         equ         0x02  ; Input - LOW when 480Hz tone is present
CP440Hz         equ         0x03  ; Input - LOW when 440Hz tone is present
Connect         equ         0x04  ; Output - HIGH closes telephone relay
DTMFClock       equ         0x05  ; Output - DTMF clock
DTMFData        equ         0x06  ; Output - DTMF data (DTMF code)
DTMFEnable      equ         0x07  ; Output - LOW enables DTMF generator

; return values from calls to subroutines
RetValues       equ   0x23        ; bit = 1 if ...
KeyPress        equ         0x00  ; a key is being pressed/closed
LineInUse       equ         0x01  ; the telephone line is being used
ncRetValues     equ         0x02  ; not used
CP_error        equ         0X03  ; CP tone information is corrupt
CP_none         equ         0x04  ; no call progress tone is detected
CP_dial         equ         0x05  ; a dial tone is detected
CP_ring         equ         0x06  ; a ringback tone is detected
CP_busy         equ         0x07  ; a busy signal is detected

; registers used to process user entries, other than the telephone number
KeyPressed      equ   0x24        ; holds most-recently pressed key
NumDigits       equ   0x25        ; number of digits in user-entered field
Digit1          equ   0x26        ; digit #1 - leftmost/first digit
Digit2          equ   0x27        ; digit #2
Digit3          equ   0x28        ; digit #3 - rightmost/last digit
StartAdd        equ   0x29        ; starting address on LCD display
MaxValue        equ   0x2A        ; maximum numeric value for field
MinValue        equ   0x2B        ; minimum numeric value for field
CurValue        equ   0x2C        ; current numeric value for field

; variables used to access the table of messages
MSGstart        equ   0x2D        ; starting offset in MSGtable
MSGlength       equ   0x2E        ; length of message

; storage for the telephone number
LenTelNum       equ   0x2F        ; number of digits in telephone number
TelNum1         equ   0x30        ; digit #1 - leftmost/first digit
TelNum2         equ   0x31        ; digit #2
TelNum3         equ   0x32        ; digit #3
TelNum4         equ   0x33        ; digit #4
TelNum5         equ   0x34        ; digit #5
TelNum6         equ   0x35        ; digit #6
TelNum7         equ   0x36        ; digit #7
TelNum8         equ   0x37        ; digit #8
TelNum9         equ   0x38        ; digit #9
TelNum10        equ   0x39        ; digit #10
TelNum11        equ   0x3A        ; digit #11
TelNum12        equ   0x3B        ; digit #12
TelNum13        equ   0x3C        ; digit #13
TelNum14        equ   0x3D        ; digit #14 - rightmost/last digit

; registers used to store user entries
MaxWaitToCall         equ   0x3E        ; max num minutes between call attempts
MinWaitToCall         equ   0x3F        ; min num minutes between call attempts
DeltaWaitToCall       equ   0x40        ; difference: maximum - minimum
MaxNumGoodCalls       equ   0x41        ; max number of good calls to attempt
MaxNumRingbacks       equ   0x42        ; max number of rings before hanging up
MaxNumBusyTries       equ   0x43        ; max number of quick repeats when busy
```

```
MaxWaitOnBusy       equ   0x44        ; max num sec between quick busy repeats
MinWaitOnBusy       equ   0x45        ; min num sec between quick busy repeats
DeltaWaitOnBusy     equ   0x46        ; difference: maximum - minimum
MaxSecsToHold       equ   0x47        ; max num seconds to hold callee on line

; registers used during main operation to store CUMULATIVE number of ...
NumGoodCalls   equ   0x48        ; ... answered calls
NumRangCalls   equ   0x49        ; ... not-busy, but call not answered
NumBusyCalls   equ   0x4A        ; ... calls which never rang
NumBusyTries   equ   0x4B        ; ... attempts in a busy cycle
NumRingbacks   equ   0x4C        ; ... pulses in a ring cycle
NumSecsToHold  equ   0x4D        ; ... seconds to hold callee on line

; registers used during main operation to display number of calls
char1          equ   0x4E
char2          equ   0x4F
char3          equ   0x50

; register used by random number generator
random         equ   0x51        ; random 8-bit number

; counters used in timing delay subroutines
count1         equ   0x52        ; del100us()
count2         equ   0x53        ; del1ms() and del10ms()
count3         equ   0x54        ; del100ms() and del500ms()
count4         equ   0x55        ; del3sec(), del10sec() and del1min()

; temporary registers used in the subroutines indicated
tempWOIBTL     equ   0x56        ; WriteOneInstructionByteToLCD()
tempWODBTL     equ   0x57        ; WriteOneDataByteToLCD()
tempDispTN     equ   0x58        ; DisplayTelephoneNumber()
tempDispNF     equ   0x59        ; DisplayNumericField()
tempLTK        equ   0x5A        ; LoopThroughKeys()
tempWFK1       equ   0x5B        ; WaitForKey()
tempWFK2       equ   0x5C        ;     "
tempCNFTB1     equ   0x5D        ; ConvertNumericFieldTBinary()
tempCNFTB2     equ   0x5E        ;     "
tempCBTCF      equ   0x5F        ; ConvertBinaryToCharacterField()
tempCADWTC     equ   0x60        ; CalcAndDoWaitToCall()
tempCADWOB     equ   0x61        ; CalcAndDoWaitOnBusy()
tempM1H        equ   0x62        ; Multiply()
tempM1L        equ   0x63        ;     "
tempM2         equ   0x64        ;     "
tempM3H        equ   0x65        ;     "
tempM3L        equ   0x66        ;     "
tempM4         equ   0x67        ;     "
tempCLIU       equ   0x68        ; CheckLineInUse()
tempRCP1       equ   0x69        ; ReadCallProgress()
tempRCP2       equ   0x6A        ;     "
tempWFDT       equ   0x6B        ; Wait2SecForDialTone()
tempICP1       equ   0x6C        ; InterpretCallProgress()
tempICP2       equ   0x6D        ;     "
tempWFS        equ   0x6E        ; Wait3SecForSilence()
tempSODC       equ   0x6F        ; SendOneDTMFCode()
tempDialTN     equ   0x70        ; DialTelephoneNumber()
;
; ********************************************************************************
```

```
; Hard reset
; ********************************************************************************
;
   org   0x0000
   ;
   ; Select register bank 0
   bcf   status,page0
   bcf   status,page1
   ; INTCON=0 disables all interrupt activity (affects portB)
   clrf  INTCON
   ; T1CON=0 disables Timer1 (affects portC)
   clrf  T1CON
   ; SSPCON<5>=0 disables synchronous serial port (affects portA and portC)
   clrf  SSPCON
   ; CCP1CON=0 disables Enhanced C/C/P module (affects portB and portC)
   clrf  CCP1CON
   ; RCSTA=0 disables the serial port (affects portC)
   clrf  RCSTA
   ; CCP2CON=0 disables Capture/Compare/PWM module (affects portC)
   clrf  CCP2CON
   ; ADCON0=0 disables Analogue-to-digital converter (affects portA)
   clrf  ADCON0
   ;
   ; Select register bank 1
   bsf   status,page0
   ; Configure OPTION_REG (affects portB)
   ; <7>=1 disable PortB pull-up resistors
   ; <6>=1 RB0 interrupt on rising edge
   ; <5>=0 internal clock drives Timer0
   ; <4>=0 increment Timer0 on low-to-high
   ; <3>=0 assign prescalar to Timer0
   ; <2-0>=111 set Timer0 prescalar 256:1
   movlw 0xC7
   movwf OPTION_REG
   ; Configure RA0-RA2 for input; RA3-RA5 for output
   movlw 0x07
   movwf TRISA
   ; Configure all pins of portB for output
   clrf  TRISB
   ; Configure RC0,RC2-RC3 for input; RC1,RC4-RC7 for output
   movlw 0x0D
   movwf TRISC
   ; PCON<ULPWUE>=0 disables ultra low-power wake-up current on RA0
   ; PCON<SBOREN>=0 disables brown-out reset
   ; (affects portA)
   bcf   PCON,5
   bcf   PCON,4
   ; WPUB=0 disables weak pull-up resistors (affects portB)
   clrf  WPUB
   ; IOCB=0 disables Interrupt-on-change (affects portB)
   clrf  IOCB
   ; PSTRCON=0 zeroes the pulse steering pin assignments (affects portC)
   clrf  PSTRCON
   ;
   ; Select register bank 2
   bcf   status,page0
   bsf   status,page1
```

```
   ; Disable Comparator module 1 (affects pins on portA)
   clrf  CM1CON0
   ; Disable Comparator module 2 (affects pins on portA)
   clrf  CM2CON0
   ; Disable Comparator module 2 (affects pins on portA and portB)
   clrf  CM2CON1
   ;
   ; Select register bank 3
   bsf   status,page0
   ; Disable analogue-to-digital on A/D channels 0-7 (affects portA)
   clrf  ANSEL
   ; Disable analogue-to-digital on A/D channels 8-13 (affects portB)
   clrf  ANSELH
   ;
   ; Select register bank 0 for main program
   bcf   status,page0
   bcf   status,page1
InitializePorts
   clrf  portAmirror
   movf  portAmirror,w
   movwf portA
   clrf  portBmirror
   movf  portBmirror,w
   movwf portB
   clrf  portCmirror
   movf  portCmirror,w
   movwf portC
InitializeHardware
   call  InitializeLCD
   bsf   portCmirror,CheckLine   ; deactivate the TS117 chip
   bcf   portCmirror,Connect     ; open the telephone line relay
   bsf   portCmirror,DTMFEnable  ; disable the DTMF generator
   movf  portCmirror,w
   movwf portC
InitializeRandomNumberGenerator
   movlw 0x55              ; random seed
   movwf random
;
; ********************************************************************************
; Block A - The main program, which starts on power-up, and handles data entry
; ********************************************************************************
;
DataEntry
   call  ClearLCDDisplay
   movlw 0x00
   movwf MSGstart
   movlw 0x11
   movwf MSGlength
   call  DisplayMessageL               ; Message1 = "Telephone number:"
   call  PutCursorAtStartOfLine2InDisplay
   call  GetTelephoneNumber
TestDial
   call  ClearLCDDisplay
   movlw 0x11
   movwf MSGstart
   movlw 0x13
   movwf MSGlength
```

```
   call  DisplayMessageL               ; Message2 = "Test dial? (1-Yes):"
   call  PutCursorAtStartOfLine2InDisplay
   call  GetNumericField
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   xorlw 0x01
   btfss status,zero                   ; skip if CurValue=1
   goto  GetMaxMinutes
   call  ClearLCDDisplay
   movlw 0x24
   movwf MSGstart
   movlw 0x0F
   movwf MSGlength
   call  DisplayMessageL               ; Message3 = "Turn on speaker"
   call  HideCursor
   call  CloseRelayTel                 ; DO NOT CHECK FOR LINE-IN-USE ...
   call  del3sec                       ; ... just wait and then dial
   call  DialTelephoneNumber           ; Dial the telephone number
   call  del1min                       ; Listen to what happens for one minute
   call  OpenRelayTel                  ; disconnect and wait 500ms
GetMaxMinutes
   call  ClearLCDDisplay
   movlw 0x33
   movwf MSGstart
   movlw 0x13
   movwf MSGlength
   call  DisplayMessageL               ; Message4 = "Mins between calls,"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0x46
   movwf MSGstart
   movlw 0x0D
   movwf MSGlength
   call  DisplayMessageL               ; Message5 = "max (0-255): "
   movlw 0x4D
   movwf StartAdd
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MaxWaitToCall
   andlw 0xFE                          ; result=0 if MaxWaitToCall=0 or 1
   btfsc status,zero                   ; check if keypad entry is zero
   goto  GetMaxSeconds                 ; goto if MaxWaitToCall=0 or 1
GetMinMinutes
   call  ClearLCDDisplay
   movlw 0x33
   movwf MSGstart
   movlw 0x13
   movwf MSGlength
   call  DisplayMessageL               ; Message4 = "Mins between calls,"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0x53
   movwf MSGstart
   movlw 0x0D
   movwf MSGlength
   call  DisplayMessageL               ; Message6 = "min (1-max): "
   movlw 0x4D
   movwf StartAdd
```

```
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MinWaitToCall                 ; save MinWaitToCall
   btfsc status,zero                   ; check if keypad entry is zero
   goto  GetMinMinutes                 ; if zero, get another number
   subwf MaxWaitToCall,w               ; subtract Max - Min (2's comp)
   btfss status,carry                  ; carry = 1 if difference positive
   goto  Error_GetMinMinutes
   goto  GetMaxNumGoodCalls
Error_GetMinMinutes
   call  ClearLCDDisplay
   movlw 0x60
   movwf MSGstart
   movlw 0x11
   movwf MSGlength
   call  DisplayMessageL               ; Message7 = "Min Minutes > Max"
   call  HideCursor
   call  del3sec
   goto  GetMaxMinutes                 ; On error, get new Max and Min
GetMaxSeconds
   clrf  MaxWaitToCall                 ; wait time will be in seconds
   call  ClearLCDDisplay
   movlw 0x71
   movwf MSGstart
   movlw 0x13
   movwf MSGlength
   call  DisplayMessageL               ; Message8 = "Secs between calls,"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0x84
   movwf MSGstart
   movlw 0x0D
   movwf MSGlength
   call  DisplayMessageL               ; Message9 = "max (1-255): "
   movlw 0x4D
   movwf StartAdd
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   btfsc status,zero                   ; check if keypad entry is zero
   goto  GetMaxSeconds                 ; if zero, get another number
   movwf MinWaitToCall                 ; save MinWaitToCall
GetMaxNumGoodCalls
   call  ClearLCDDisplay
   movlw 0x91
   movwf MSGstart
   movlw 0x13
   movwf MSGlength
   call  DisplayMessageL               ; Message10 = "Number of telephone"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0xA4
   movwf MSGstart
   movlw 0x10
   movwf MSGlength
   call  DisplayMessageL               ; Message11 = "calls (10-255): "
   movlw 0x50
   movwf StartAdd
```

```
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MaxNumGoodCalls               ; save MaxNumGoodCalls
   sublw 0x09                          ; subtract d'9' - MaxNumGoodCalls
   btfsc status,carry                  ; carry = 0 if difference negative
   goto  GetMaxNumGoodCalls            ; re-start if MaxNumGoodCalls < 10
GetMaxNumRings
   call  ClearLCDDisplay
   movlw 0xB4
   movwf MSGstart
   movlw 0x0E
   movwf MSGlength
   call  DisplayMessageL               ; Message12 = "Number of ring"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0xC2
   movwf MSGstart
   movlw 0x0F
   movwf MSGlength
   call  DisplayMessageL               ; Message13 = "pulses (3-10): "
   movlw 0x4F
   movwf StartAdd
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MaxNumRingbacks               ; save MaxNumRingbacks
   sublw 0x02                          ; subtract d'2' - MaxNumRings
   btfsc status,carry                  ; carry = 1 if difference positive
   goto  GetMaxNumRings                ; re-start if MaxNumRings < 3
   movlw 0x0B                          ; 0x0B = d'11'
   subwf MaxNumRingbacks,w             ; subtract Max - d'11'
   btfsc status,carry                  ; carry = 0 if difference negative
   goto  GetMaxNumRings                ; re-start if MaxNumRings >= 11
GetMaxNumBusyTries
   call  ClearLCDDisplay
   movlw 0xD1
   movwf MSGstart
   movlw 0x0E
   movwf MSGlength
   call  DisplayMessageL               ; Message14 = "Number of busy"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0x00
   movwf MSGstart
   movlw 0x0E
   movwf MSGlength
   call  DisplayMessageU               ; Message15 = "tries (5-50): "
   movlw 0x4E
   movwf StartAdd
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MaxNumBusyTries               ; save MaxNumBusyTries
   sublw 0x04                          ; subtract d'4' - MaxNumBusyTries
   btfsc status,carry                  ; carry = 1 if difference positive
   goto  GetMaxNumBusyTries            ; re-start if MaxNumBusyTries < 5
   movlw 0x33                          ; 0x33 = d'51'
   subwf MaxNumBusyTries,w             ; subtract Max - d'51'
```

```
   btfsc status,carry                  ; carry = 0 if difference negative
   goto  GetMaxNumBusyTries            ; re-start if MaxNumBusyTries >= 51
GetMaxWaitOnBusy
   call  ClearLCDDisplay
   movlw 0x0E
   movwf MSGstart
   movlw 0x14
   movwf MSGlength
   call  DisplayMessageU               ; Message16 = "Max sec between busy"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0x00
   movwf MSGstart
   movlw 0x0E
   movwf MSGlength
   call  DisplayMessageU               ; Message15 = "tries (5-50): "
   movlw 0x4E
   movwf StartAdd
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MaxWaitOnBusy                 ; save MaxWaitOnBusy
   sublw 0x04                          ; subtract d'4' - MaxWaitOnBusy
   btfsc status,carry                  ; carry = 1 if difference positive
   goto  GetMaxWaitOnBusy              ; re-start if MaxWaitOnBusy < 5
   movlw 0x33                          ; d'51' = 0x33
   subwf MaxWaitOnBusy,w               ; subtract Max - d'51'
   btfsc status,carry                  ; carry = 0 if difference negative
   goto  GetMaxWaitOnBusy              ; re-start of MaxWaitOnBusy >= 51
GetMinWaitOnBusy
   call  ClearLCDDisplay
   movlw 0x22
   movwf MSGstart
   movlw 0x14
   movwf MSGlength
   call  DisplayMessageU               ; Message17 = "Min sec between busy"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0x36
   movwf MSGstart
   movlw 0x0F
   movwf MSGlength
   call  DisplayMessageU               ; Message18 = "tries (1-max): "
   movlw 0x4F
   movwf StartAdd
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MinWaitOnBusy                 ; save MinWaitOnBusy
   btfsc status,zero                   ; check if keypad entry is zero
   goto  GetMinWaitOnBusy              ; if zero, get another number
   subwf MaxWaitOnBusy,w               ; subtract Max - Min (2's comp)
   btfss status,carry                  ; carry = 1 if difference positive
   goto  Error_GetMinWaitOnBusy
   goto  GetMaxSecsToHold
Error_GetMinWaitOnBusy
   call  ClearLCDDisplay
   movlw 0x45
   movwf MSGstart
```

```
   movlw 0x11
   movwf MSGlength
   call  DisplayMessageU               ; Message19 = "Min seconds > Max"
   call  HideCursor
   call  del3sec
   goto  GetMaxWaitOnBusy              ; On error, get new Max and Min
GetMaxSecsToHold
   call  ClearLCDDisplay
   movlw 0x56
   movwf MSGstart
   movlw 0x13
   movwf MSGlength
   call  DisplayMessageU               ; Message20 = "Max seconds to hold"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0x69
   movwf MSGstart
   movlw 0x10
   movwf MSGlength
   call  DisplayMessageU               ; Message21 = "callee (2-240): "
   movlw 0x50
   movwf StartAdd
   call  GetNumericField               ; keypad entry will be 0-255
   call  ConvertNumericFieldToBinary
   movf  CurValue,w
   movwf MaxSecsToHold                 ; save MaxSecsToHold the callee on line
   sublw 0x01                          ; subtract d'1' - MaxSecsToHold
   btfsc status,carry                  ; carry = 1 if difference positive
   goto  GetMaxSecsToHold              ; re-start if MaxSecsToHold < 2
   movlw 0xF1                          ; 0xF1 = d'241'
   subwf MaxSecsToHold,w               ; subtract Max - d'241'
   btfsc status,carry                  ; carry = 0 if difference positive
   goto  GetMaxSecsToHold              ; re-start if MaxSecsToHold >= 241
;
; ***************************************************************************
; Block B - The main operating program, which runs the dialler
; ***************************************************************************
;
MainProgram
   movf  MinWaitToCall,w
   subwf MaxWaitToCall,w
   movwf DeltaWaitToCall               ; Max - Min --> DeltaWaitToCall
   movf  MinWaitOnBusy,w
   subwf MaxWaitOnBusy,w
   movwf DeltaWaitOnBusy               ; Max - Min --> DeltaWaitOnBusy
   clrf  NumGoodCalls                  ; clear the cumulative totals
   clrf  NumRangCalls
   clrf  NumBusyCalls
   clrf  NumRingbacks                  ; clear NumRingbacks for first call
   clrf  NumBusyTries                  ; clear NumBusyTries for first call
CheckExternalLine
   call  CheckLineInUse
   btfss RetValues,LineInUse           ; LineInUse=1 if line is in use
   goto  ConnectToExternalLine
Error_LineInUse
   call  ClearLCDDisplay
   movlw 0x7C
   movwf MSGstart
```

```
   movlw 0x0B
   movwf MSGlength
   call  DisplayMessageU              ; Message22 = "Line in use"
   call  HideCursor
   call  del1min                      ; wait one minute for line to clear
   goto  CheckExternalLine            ; try again after one minute
ConnectToExternalLine
   call  CloseRelayTel                ; connect and wait 500ms
   call  Wait2SecForDialTone          ; wait 2 seconds for dial tone
   btfsc RetValues,CP_dial            ; CP_dial=1 if there is a dial tone
   goto  StartDialing
Error_NoDialTone
   call  OpenRelayTel                 ; disconnect and wait 500ms
   call  ClearLCDDisplay
   movlw 0x87
   movwf MSGstart
   movlw 0x0C
   movwf MSGlength
   call  DisplayMessageU              ; Message23 = "No dial tone"
   call  HideCursor
   call  del1min                      ; wait one minute for a change
   goto  CheckExternalLine            ; try again after one minute
StartDialing
   call  ClearLCDDisplay
   movlw 0x93
   movwf MSGstart
   movlw 0x0C
   movwf MSGlength
   call  DisplayMessageU              ; Message24 = "Dialling ..."
   call  HideCursor
   call  DialTelephoneNumber
   call  del500ms                     ; wait 500ms before testing
InterpretCallProgressSignal
   call  InterpretCallProgress        ; interpret the callee's response
   btfsc RetValues,CP_busy            ; CP_busy=1 if line is busy
   goto  CalleeIsBusy
   btfsc RetValues,CP_ring            ; CP_ring=1 if phone is ringing
   goto  CalleeIsRinging
   btfsc RetValues,CP_none            ; CP_none=1 if phone picked up
   goto  CalleeAnswered               ; no tone -> callee picked up
   goto  TelephoneSystemError         ; should never be a dial tone here
CalleeIsBusy
   call  ClearLCDDisplay
   movlw 0x9F
   movwf MSGstart
   movlw 0x08
   movwf MSGlength
   call  DisplayMessageU              ; Message25 = "Busy ..."
   call  HideCursor
   call  OpenRelayTel                 ; disconnect and wait 500ms
   incf  NumBusyTries,f
   movf  NumBusyTries,w
   xorwf MaxNumBusyTries,w            ; zero if they are equal
   btfss status,zero
   goto  CalleeIsBusy_TryAgain
   goto  CalleeIsBusy_StopTrying
CalleeIsBusy_TryAgain
```

```
   call  ClearLCDDisplay
   movlw 0xA7
   movwf MSGstart
   movlw 0x0E
   movwf MSGlength
   call  DisplayMessageU               ; Message26 = "Dialling try #"
   movlw 0x8E                          ; DB7=b'1' in the instruction
                                       ; LCD address is 0x0E
   call  WriteOneInstructionByteToLCD
   movf  NumBusyTries,w
   call  ConvertBinaryToCharacterField
   movf  char1,w
   call  WriteOneDataByteToLCD
   movf  char2,w
   call  WriteOneDataByteToLCD
   movf  char3,w
   call  WriteOneDataByteToLCD
   call  HideCursor
   call  CalcAndDoWaitOnBusy           ; wait before trying again
   goto  CheckExternalLine             ; try dialling again
CalleeIsBusy_StopTrying
   incf  NumBusyCalls,f                ; increment the cumulative total
   clrf  NumRingbacks                  ; clear NumRingbacks for next call
   clrf  NumBusyTries                  ; clear NumBusyTries for next call
   call  DisplayProgressReport
   call  CalcAndDoWaitToCall           ; wait a random time for next call
   goto  CheckExternalLine             ; start next call attempt
CalleeIsRinging                        ; we are here at the start of the ...
   call  ClearLCDDisplay               ; ... first ringback pulse
   movlw 0xB5
   movwf MSGstart
   movlw 0x0B
   movwf MSGlength
   call  DisplayMessageU               ; Message27 = "Ringing ..."
   call  HideCursor
   incf  NumRingbacks,f                ; the first ringback is in progress
CallIsUnderway1                        ; we are here at the start of a ringback
   call  Wait3SecForSilence            ; wait up to 3 seconds for silence
   btfsc RetValues,CP_none             ; CP_none=1 if there is silence
   goto  CallIsUnderway2
   btfss RetValues,CP_dial             ; CP_dial=1 if there is a dial tone
   goto  CalleeAnswered                ; dial tone means the callee hung up
   goto  TelephoneSystemError          ; should never end up here (busy signal)
CallIsUnderway2                        ; we are here at the start of silence
   call  InterpretCallProgress         ; interpret the callee's response
   btfss RetValues,CP_ring             ; CP_ring=1 if line is ringing
   goto  CalleeAnswered
   incf  NumRingbacks,f                ; increment number of ringbacks
   movf  NumRingbacks,w
   xorwf MaxNumRingbacks,w             ; zero if they are equal
   btfss status,zero
   goto  CallIsUnderway1
   goto  RangCallCompleted             ; maximum number of ringbacks, no answer
CalleeAnswered
   ; Once the callee picks up the handset, we will wait until the earlier of:
   ; (i) the callee hangs up, or (ii) MaxSecsToHold seconds elapse.  When the
   ; callee hangs up, one would expect to hear a dial tone.  However, one could
```

```
   ; get a call-waiting fast busy, or perhaps something else.  I have decided to
   ; proceed as follows: after the callee answers, I will start monitoring the
   ; line for any call progress tone (not just a dial tone) and interpret the
   ; detection of any call progress tone as the result of the callee hanging up.
   ; The line is sampled once every second for this purpose.
   movf  MaxSecsToHold,w
   movwf NumSecsToHold                ; counter for seconds remaining
CalleeIsOnLine
   call  del500ms                     ; sample the line every second
   call  del500ms
   call  ReadCallProgress
   btfss RetValues,CP_none            ; if CP_none=1, then there is no CP ...
   goto  GoodCallCompleted            ; ... tone and the callee is still on
   decfsz NumSecsToHold,f
   goto  CalleeIsOnLine               ; since NumSecsToHold>0, keep holding
GoodCallCompleted
   call  OpenRelayTel                 ; disconnect and wait 500ms
   incf  NumGoodCalls,f               ; increment the cumulative total
   clrf  NumRingbacks                 ; clear NumRingbacks for next call
   clrf  NumBusyTries                 ; clear NumBusyTries for next call
   call  DisplayProgressReport
   call  CalcAndDoWaitToCall          ; wait a random time for next call
   decfsz MaxNumGoodCalls,f           ; decrement number of good calls to go
   goto  CheckExternalLine            ; either start the next call ...
   goto  FinalLoop                    ; ... or stop the procedure
RangCallCompleted
   call  OpenRelayTel                 ; disconnect and wait 500ms
   incf  NumRangCalls,f               ; increment the cumulative total
   clrf  NumRingbacks                 ; clear NumRingbacks for next call
   clrf  NumBusyTries                 ; clear NumBusyTries for next call
   call  DisplayProgressReport
   call  CalcAndDoWaitToCall          ; wait a random time for next call
   goto  CheckExternalLine            ; start next call attempt
TelephoneSystemError
   call  ClearLCDDisplay
   movlw 0xEA
   movwf MSGstart
   movlw 0x10
   movwf MSGlength
   call  DisplayMessageU              ; Message31 = "Tel system error"
   call  HideCursor
   goto  FinalLoop
FinalLoop
   call  ClearLCDDisplay              ; Alternate "All finished" with ...
   movlw 0xC0                         ; ... final progress report, ...
   movwf MSGstart                     ; ... showing each for one minute
   movlw 0x0C
   movwf MSGlength
   call  DisplayMessageU              ; Message28 = "All finished"
   call  HideCursor
   call  del1min
   call  DisplayProgressReport
   call  del1min
   goto  FinalLoop
;
; ********************************************************************************
; Block C - Routines needed to operate the LCD hardware
```

```
; The LCD is operated using the 4-bit interface mode.  No timing checks are
; done.  Instead, after each instruction is sent, a delay is introduced equal
; in length to the maximum required execution time.
; Subroutines:-
;    WriteOneInstructionNibbleToLCD()
;    WriteOneInstructionByteToLCD()
;    WriteOneDataNibbleToLCD()
;    WriteOneDataByteToLCD()
;    InitializeLCD()
;    PutCursorAtStartOfLine2InDisplay()
;    ClearLCDDisplay()
; ***************************************************************************
;
WriteOneInstructionNibbleToLCD
; Assumes: Data is stored in the low nibble of register w
; This subroutine includes all timing delays needed by the LCD
   movwf portBmirror                    ; store in portBmirror low nibble
   bcf   portBmirror,LCDE
   bcf   portBmirror,LCDRW              ; assert low for write
   movf  portBmirror,w
   movwf portB
   nop                                  ; delay 400ns
   bsf   portBmirror,LCDE               ; set E bit high
   movf  portBmirror,w
   movwf portB
   nop                                  ; delay 400ns
   bcf   portBmirror,LCDE               ; assert E bit low
   movf  portBmirror,w
   movwf portB
   call  del100us                       ; delay 100us
   return
;
WriteOneInstructionByteToLCD
; Assumes: Data is stored in register w
; This subroutine includes all timing delays needed by the LCD
   movwf tempWOIBTL                     ; save data in register tempWOIBTL
   swapf tempWOIBTL,w                   ; move high nibble into w low nibble
   andlw 0x0F
   call  WriteOneInstructionNibbleToLCD
   movf  tempWOIBTL,w                   ; move low nibble into w low nibble
   andlw 0x0F
   call  WriteOneInstructionNibbleToLCD
   return
;
WriteOneDataNibbleToLCD
; Assumes: Data is stored in the low nibble of register w
; This subroutine includes all timing delays needed by the LCD
   movwf portBmirror                    ; store in portBmirror low nibble
   bcf   portBmirror,LCDE
   bsf   portBmirror,LCDRS              ; <-- NB INSTRUCTION versus DATA
   bcf   portBmirror,LCDRW              ; assert low for write
   movf  portBmirror,w
   movwf portB
   nop                                  ; delay 400ns
   bsf   portBmirror,LCDE               ; set E bit high
   movf  portBmirror,w
   movwf portB
```

```
   nop                                  ; delay 400ns
   bcf   portBmirror,LCDE               ; assert E bit low
   movf  portBmirror,w
   movwf portB
   call  del100us                       ; delay 400ns
   return
;
WriteOneDataByteToLCD
; Assumes: Data is stored in register w
; This subroutine includes all timing delays needed by the LCD
   movwf tempWODBTL                     ; save data in register tempWODBTL
   swapf tempWODBTL,w                   ; move high nibble into w low nibble
   andlw 0x0F
   call  WriteOneDataNibbleToLCD
   movf  tempWODBTL,w                   ; move low nibble into w low nibble
   andlw 0x0F
   call  WriteOneDataNibbleToLCD
   return
;
InitializeLCD
; This subroutine includes all timing delays needed by the LCD
   call  del10ms                        ; wait at least 15ms
   call  del10ms
   movlw 0x03
   call  WriteOneInstructionNibbleToLCD
   call  del10ms                        ; wait at least 4.1ms
   movlw 0x03
   call  WriteOneInstructionNibbleToLCD
   call  del1ms                         ; wait at least 100us
   movlw 0x03
   call  WriteOneInstructionNibbleToLCD
   call  del100us                       ; wait at least 100us
   movlw 0x03
   call  WriteOneInstructionNibbleToLCD
   call  del100us                       ; wait at least 38us
   movlw 0x02                           ; set interface to 4 bits
   call  WriteOneInstructionNibbleToLCD
   call  del100us                       ; wait at least 38us
   movlw 0x2C                           ; set the functions
                                        ; DB<7-5>=b'001'
                                        ; DB4=0 for 4-bit interface
                                        ; DB3=1 for 2-line display
                                        ; DB2=1 for 5x11 font
                                        ; DB<1-0>=b'00'
   call  WriteOneInstructionByteToLCD
   call  del100us                       ; wait at least 38us
   movlw 0x0F                           ; set the display parameters
                                        ; DB<7-3>=b'00001'
                                        ; DB2=1 turns on the display
                                        ; DB1=1 turns on the cursor
                                        ; DB0=1 causes the cursor to blink
   call  WriteOneInstructionByteToLCD
   call  del100us                       ; wait at least 38us
   movlw 0x01                           ; clear the display
   call  WriteOneInstructionByteToLCD
   call  del10ms                        ; wait at least 1.52ms
   movlw 0x06                           ; set the entry mode
```

```
                                    ; DB<7-2>=b'000001'
                                    ; DB1=1 selects left-to-right entry
                                    ; DB0=0 prevents shifting display
   call  WriteOneInstructionByteToLCD
   call  del100us                   ; wait at least 38us
   return
;
PutCursorAtStartOfLine2InDisplay
; Assumes portB is configured for LCD write operations
; This subroutine includes all timing delays needed by the LCD.  It also places
; the start of the second line into register StartAdd.
   movlw 0xC0                       ; Set the DRAM address
                                    ; DB7=b'1'
                                    ; Address=0x40=b'1000000'
   call  WriteOneInstructionByteToLCD
   movlw 0x40
   movwf StartAdd
   return
;
HideCursor
; Assumes portB is configured for LCD write operations
; This subroutine moves the cursor to the 21st spot on the second line
; This subroutine includes all timing delays needed by the LCD
   movlw 0xD5                       ; Set the DRAM address
                                    ; DB7=b'1'
                                    ; Address=0x55=b'1010101'
   call  WriteOneInstructionByteToLCD
   return
;
ClearLCDDisplay
; Assumes portB is configured for LCD write operations
; This subroutine includes all timing delays needed by the LCD.  It also places
; the start of the first line into register StartAdd.
   movlw 0x01
   call  WriteOneInstructionByteToLCD
   call  del10ms                    ; wait at least 1.52ms
   clrf  StartAdd
   return
;
; ********************************************************************************
; Block D - Routines needed to operate the keypad hardware
; Subroutines:-
;  LoopThroughKeys()
;  WaitForKey()
; ********************************************************************************
;
LoopThroughKeys
; Returns:    1. If no key is pressed, RetValues<KeyPress>=0
;             2. If a key is pressed,
;                   a. register w holds the key value
;                   b. RetValues<KeyPress>=1
;             3. The key values are the LCD codes:
;                   0 = b'00110000' = 0x30
;                   1 = b'00110001' = 0x31
;                   9 = b'00111001' = 0x39
;                   * = b'00101010' = 0x2A
;                   # = b'00100011' = 0x23
```

```
; This routine does not do any debouncing
   movlw 0x01
   movwf portBmirror                    ; Activate row #1
   movwf portB
   call  del100us
   movf  portA,w                        ; Read all columns
   andlw 0x07                           ; Zero out all non-column bits ...
   movwf tempLTK                        ; ... and save in register tempLTK
   btfss tempLTK,Col1                   ; Test column #1
   goto  LTK1
   movlw 0x31                           ; The "1" key is pressed
   goto  LTK_pressed
LTK1
   btfss tempLTK,Col2                   ; Test column #2
   goto  LTK2
   movlw 0x32                           ; The "2" key is pressed
   goto  LTK_pressed
LTK2
   btfss tempLTK,Col3                   ; Test column #3
   goto  LTK3
   movlw 0x33                           ; The "3" key is pressed
   goto  LTK_pressed
LTK3
   movlw 0x02
   movwf portBmirror                    ; Activate row #2
   movwf portB
   call  del100us
   movf  portA,w                        ; Read all columns
   andlw 0x07                           ; Zero out all non-column bits ...
   movwf tempLTK                        ; ... and save in register tempLTK
   btfss tempLTK,Col1                   ; Test column #1
   goto  LTK4
   movlw 0x34                           ; The "4" key is pressed
   goto  LTK_pressed
LTK4
   btfss tempLTK,Col2                   ; Test column #2
   goto  LTK5
   movlw 0x35                           ; The "5" key is pressed
   goto  LTK_pressed
LTK5
   btfss tempLTK,Col3                   ; Test column #3
   goto  LTK6
   movlw 0x36                           ; The "6" key is pressed
   goto  LTK_pressed
LTK6
   movlw 0x04
   movwf portBmirror                    ; Activate row #3
   movwf portB
   call  del100us
   movf  portA,w                        ; Read all columns
   andlw 0x07                           ; Zero out all non-column bits ...
   movwf tempLTK                        ; ... and save in register tempLTK
   btfss tempLTK,Col1                   ; Test column #1
   goto  LTK7
   movlw 0x37                           ; The "7" key is pressed
   goto  LTK_pressed
LTK7
```

```
   btfss tempLTK,Col2                  ; Test column #2
   goto  LTK8
   movlw 0x38                          ; The "8" key is pressed
   goto  LTK_pressed
LTK8
   btfss tempLTK,Col3                  ; Test column #3
   goto  LTK9
   movlw 0x39                          ; The "6" key is pressed
   goto  LTK_pressed
LTK9
   movlw 0x08
   movwf portBmirror                   ; Activate row #4
   movwf portB
   call  del100us
   movf  portA,w                       ; Read all columns
   andlw 0x07                          ; Zero out all non-column bits ...
   movwf tempLTK                       ; ... and save in register tempLTK
   btfss tempLTK,Col1                  ; Test column #1
   goto  LTK10
   movlw 0x2A                          ; The "*" key is pressed
   goto  LTK_pressed
LTK10
   btfss tempLTK,Col2                  ; Test column #2
   goto  LTK11
   movlw 0x30                          ; The "0" key is pressed
   goto  LTK_pressed
LTK11
   btfss tempLTK,Col3                  ; Test column #3
   goto  LTK_nokey
   movlw 0x23                          ; The "#" key is pressed
   goto  LTK_pressed
LTK_nokey
   clrf  RetValues
   return
LTK_pressed
   clrf  RetValues
   bsf   RetValues,KeyPress
   return
;
WaitForKey
;  This subroutine waits indefinitely for any key to be pressed
;  It returns the key code in register KeyPressed
;  This routine uses 10ms debouncing on Open and Close
   call  LoopThroughKeys
   movwf tempWFK1                      ; save key, if any, in tempWFK1
   btfsc RetValues,KeyPress
   goto  WFK1                          ; jump if a key is pressed
   call  del10ms                       ; wait 10ms before checking again
   goto  WaitForKey
WFK1
   call  del10ms                       ; wait 10ms before confirming key
   call  LoopThroughKeys
   movwf tempWFK2                      ; save key, if any, in tempWFK2
   btfss RetValues,KeyPress
   goto  WaitForKey                    ; start over again if no key pressed
   movf  tempWFK1,w
   xorwf tempWFK2,w                    ; Compare new key with original
```

```
   btfss status,zero
   goto  WaitForKey                    ; Failure: a different key was read
WFK2
   call  del10ms                       ; wait 10ms before checking again
   call  LoopThroughKeys               ; Loop/wait until key is released
   btfsc RetValues,KeyPress
   goto  WFK2                          ; key is still pressed, keep waiting
   movf  tempWFK1,w
   movwf KeyPressed                    ; Place keystroke in KeyPressed
   return
;
; ***************************************************************************
; Block E - Routines needed to operate the telephone line
; Subroutines:-
;   CheckLineInUse() - determines if the telephone line is in use
;   OpenRelayTel() - opens Rly, disconnecting from the telephone line
;   CloseRelayTel() - closes Rly, connecting to the telephone line
;   ReadCallProgress() - samples the call progress tones at one instant
;   Wait2SecForDialTone() - waits up to 2 seconds for a dial tone
;   InterpretCallProgress() - determines type of callee response
;   Wait3SecForSilence() - waits up to 2 seconds for silence on the line
;   SendOneDTMFCode() - transmits one DTMF code and the following silence
;   DialTelephoneNumber() - dials a complete telephone number
; ***************************************************************************
;
CheckLineInUse
; This subroutine determines if the telephone line is in use
; Assumes:    1. Status of TS117 (U3) is unknown
; Returns:    1. TS117 is deactivated
;             2. If the line is in use, RetValues<LineInUse>=1
;             3. If the line is free, RetValues<LineInUse>=0
; Step #1 - Activate the TS117 chip
   bcf   portCmirror,CheckLine
   movf  portCmirror,w
   movwf portC
   call  del10ms
   ; Step #2 - Set the default return value to "in use"
   bsf   RetValues,LineInUse
   ; Step #3 - Read the line-in-use bit
   movf  portC,w
   movwf tempCLIU
   btfss tempCLIU,LIU                  ; if LIU bit is high, line is in use
   bcf   RetValues,LineInUse           ; if LIU bit is low, line is free
   ; Step #4 - Deactivate the TS117 chip
   bsf   portCmirror,CheckLine
   movf  portCmirror,w
   movwf portC
   call  del10ms
   return
;
OpenRelayTel
; This subroutine opens Rly, disconnecting from the telephone line.  It waits
; one-half second before returning.
   bcf   portCmirror,Connect
   movf  portCmirror,w
   movwf portC
   call  del500ms
```

```
   return
;
CloseRelayTel
; This subroutine closes Rly, connecting to the telephone line. It waits
; one-half second before returning.
   bsf   portCmirror,Connect
   movf  portCmirror,w
   movwf portC
   call  del500ms
   return
;
ReadCallProgress
; This subroutine samples the outputs of the 440Hz and 480Hz filters every
; 100us for 1ms, making 11 observations.  The results of the 11 samples are
; inverted and then inclusively OR-ed together.  When the desired frequencies
; are present, the filter outputs are zero, so their complements are one.
; Any single sample which includes a desired frequency will be recorded as a
; one in the accumulated OR-result.  In other words, one observation of a
; desired frequency will be enough to determine a positive result.  At the
; end of the sampling, the subroutine will set one of the five RetValues<CP_****>
; bits to correspond with what was heard.
;   A dial tone is characterized by: 480Hz not present; 440Hz is present.
;   A busy signal is characterized by: 480Hz is present; 440Hz is not present.
;   A ringback is characterized by both 480Hz and 440Hz are present.
; portC<CP480Hz = 2> is the 480Hz filter input line
; portC<CP440Hz = 3> is the 440Hz filter input line
   movlw 0x0B                          ; 0x0B = d'11'
   movwf tempRCP1                      ; use tempRCP1 as a loop counter
   clrf  tempRCP2                      ; use tempRCP2 to hold the result
RCP1
   movf  portC,w                       ; read the call progress port
   andlw 0x0C                          ; keep only the two filter bits
   xorlw 0x0C                          ; complement the two filter bits
   iorwf tempRCP2,f
   call  del100us                      ; loop delay is 100us
   decfsz tempRCP1,f                   ; repeat d'11'times
   goto  RCP1
   ; Now, set one of the RetValues<CP_****> bits
   movf  RetValues,w                   ; clear the five <CP_****> bits
   andlw 0x1F
   movwf RetValues
   movf  tempRCP2,w                    ; test tempRCP2 = 0x0C?
   xorlw 0x0C                          ; result is zero if exact match
   btfsc status,zero
   bsf   RetValues,CP_ring             ; both bits are high --> ringback
   movf  tempRCP2,w                    ; test tempRCP2 = 0x08?
   xorlw 0x08                          ; result is zero if exact match
   btfsc status,zero
   bsf   RetValues,CP_dial             ; 440Hz only --> dial tone
   movf  tempRCP2,w                    ; test tempRCP2 = 0x04?
   xorlw 0x04                          ; result is zero if exact match
   btfsc status,zero
   bsf   RetValues,CP_busy             ; 480Hz only --> busy signal
   movf  tempRCP2,w                    ; test tempRCP2 = 0x00?
   xorlw 0x00                          ; result is zero if exact match
   btfsc status,zero
   bsf   RetValues,CP_none             ; no tone --> silence
```

```
   return
;
Wait2SecForDialTone
; This subroutine checks for a dial tone every 10ms for two seconds.
; If a dial tone is heard, and heard again after 10ms, then the subroutine
; will return with bit RetValues<CP_dial> set high.  If the two seconds runs
; out without the dial tone being heard twice, then the subroutine will return
; with RetValues<CP_dial> set low.  Since a dial tone is a continuous tone, and
; since subroutine CloseRelayTel() waits one-half second after closing the
; relay, one would expect a dial tone to be heard within the two seconds
; allowed.  Furthermore, since the relay is closed only after the external line
; has been checked and confirmed to be available, the absence of a dial tone
; should be a very rare event.
   movlw 0xC8                          ; 0xC8 = d'200'
   movwf tempWFDT                      ; use tempWFDT as timeout counter
WFDT1
   call  ReadCallProgress
   btfss RetValues,CP_dial
   goto  WFDT2                         ; goto if it is not a dial tone
   ; Second call to ReadCallProgress to confirm a dial tone
   call  del10ms                       ; wait 10ms before confirming
   call  ReadCallProgress
   btfss RetValues,CP_dial
   goto  WFDT2                         ; goto if dial tone is not confirmed
   return                              ; return with <CP_dial> bit high
WFDT2    ; we are here if there is no dial tone (either silence or something else)
   call  del10ms                       ; keep waiting
   decfsz tempWFDT,f                   ; all finished when tempWFDT=0
   goto  WFDT1                         ; start a new test if tempWFDT<>0
   bsf   RetValues,CP_none             ; return with <CP_none> bit high
   bcf   RetValues,CP_dial             ; return with <CP_dial> bit low
   bcf   RetValues,CP_ring
   bcf   RetValues,CP_busy
   return
;
InterpretCallProgress
; This subroutine is called after the telephone number has been dialled.  It tries
; to determine the type of response by considering both the timing of the pulses
; and their frequency content.  It call subroutine ReadCallProgress() every 20ms
; until it can make a determination.  It returns with only one of the five
; RetValues<CP_****> bits set high.
   movf  RetValues,w
   andlw 0xC0                          ; 0xC0 = b'11000000'
   movwf RetValues                     ; clear all RetValues<CP_****> bits
   clrf  tempICP1                      ; tempICP1 is the counter
ICP1
   call  ReadCallProgress
   btfss RetValues,CP_none             ; RetValues<CP_none>=1 if silent
   goto  ICP2                          ; goto if not silence
   call  del10ms                       ; wait 20ms
   call  del10ms
   incf  tempICP1,f                    ; counter = counter + 1
   movlw 0xD7                          ; 0xD7 = d'215'
   subwf tempICP1,w                    ; w <-- counter - d'215'
   btfsc status,carry                  ; carry=1 if d'215' <= counter
   goto  ICP1                          ; keep waiting
   return                              ; return with RetValues<CP_none>=1
```

```
ICP2
  ; First transition from silence to a CP pulse (point "B" in flow chart)
  clrf  tempICP1                      ; set counter=0
ICP3
  call  ReadCallProgress
  btfsc RetValues,CP_none             ; RetValues<CP_none>=1 if silent
  goto  ICP6                          ; goto if CP pulse has stopped
  movf  RetValues,w
  movwf tempICP2                      ; tempICP2 is storage for CP_**** bit
  call  del10ms                       ; wait 20ms
  call  del10ms
  incf  tempICP1,f                    ; counter = counter + 1
  movlw 0x0A                          ; 0x0A = d'10'
  subwf tempICP1,w                    ; w <-- counter - d'10'
  btfsc status,carry                  ; carry=1 if d'10' <= counter
  goto  ICP3                          ; keep waiting
  movlw 0xD7                          ; 0xD7 = d'215'
  subwf tempICP1,w                    ; w <-- counter - d'215'
  btfsc status,carry                  ; carry=1 if d'215' <= counter
  goto  ICP5
  ; we are here if counter >= 215
  btfss tempICP2,CP_dial
  goto  ICP4
  bsf   RetValues,CP_dial
  return                              ; return with RetValues<CP_dial>=1
ICP4
  bsf   RetValues,CP_error
  return                              ; return with RetValues<CP_error>=1
ICP5
  ; we are here if counter < 215
  movlw 0x1D                          ; 0x1D = d'29'
  subwf tempICP1,w                    ; w <-- counter - d'29'
  btfsc status,carry                  ; carry=1 if d'29' <= counter
  goto  ICP2
  btfss tempICP2,CP_ring
  goto  ICP2
  bsf   RetValues,CP_ring
  return                              ; return with RetValues<CP_ring>=1
ICP6
  ; Definite transition from CP tone to silence (point "C" in flow chart)
  clrf  tempICP1                      ; set counter=0
ICP7
  call  ReadCallProgress
  btfss RetValues,CP_none             ; RetValues<CP_none>=1 if silent
  goto  ICP8                          ; goto if not silent
  call  del10ms                       ; wait 20ms
  call  del10ms
  incf  tempICP1,f                    ; counter = counter + 1
  movlw 0xD7                          ; 0xD7 = d'215'
  subwf tempICP1,w                    ; w <-- counter - d'215'
  btfsc status,carry                  ; carry=1 if d'215' <= counter
  goto  ICP7                          ; keep waiting if silence continues
  return                              ; return with <CP_ring> bit high
ICP8
  ; we are here if the first full silent period has ended
  movlw 0xD3                          ; 0xD3 = d'211'
  subwf tempICP1,w                    ; w <-- counter - d'211'
```

```
  btfss status,carry                ; carry=1 if d'211' <= counter
  goto  ICP9
  clrf  RetValues
  bsf   RetValues,CP_error
  return                            ; return with RetValues<CP_error)=1
ICP9
  movlw 0xAB                        ; 0xAB = d'171'
  subwf tempICP1,w                  ; w <-- counter - d'171'
  btfsc status,carry                ; carry=1 if d'171' <= counter
  goto  ICP10
  clrf  RetValues
  bsf   RetValues,CP_ring
  return                            ; return with RetValues<CP_ring>=1
ICP10
  movlw 0x1E                        ; 0x1E = d'30'
  subwf tempICP1,w                  ; w <-- counter - d'30'
  btfss status,carry                ; carry=1 if d'30' <= counter
  goto  ICP11
  clrf  RetValues
  bsf   RetValues,CP_error
  return                            ; return with RetValues<CP_error>=1
ICP11
  movlw 0x13                        ; 0x13 = d'19'
  subwf tempICP1,w                  ; w <-- counter - d'19'
  btfss status,carry                ; carry=1 if d'19' <= counter
  goto  ICP12
  clrf  RetValues
  bsf   RetValues,CP_busy
  return                            ; return with RetValues<CP_busy>=1
ICP12
  clrf  RetValues
  bsf   RetValues,CP_error
  return                            ; return with RetValues<CP_error>=1
;
Wait3SecForSilence
; This subroutines checks for silence every 100ms for three seconds.
; If a silent period starts, and repeated after 10ms, then the subroutine
; will return with bit RetValues<CP_none> set high.  If the three seconds run
; out without silence being heard twice, then the subroutine will return with
; RetValues<CP_none> set low.
  movlw 0x1E                        ; 0x1E = d'30'
  movwf tempWFS                     ; use tempWFS as timeout counter
WFS1
  call  ReadCallProgress
  btfss RetValues,CP_none
  goto  WFS2                        ; goto if there is not silence
  ; Second call to ReadCallProgress to confirm silence
  call  del10ms                     ; wait 10ms before confirming
  call  ReadCallProgress
  btfss RetValues,CP_none
  goto  WFS2                        ; goto if silence is not confirmed
  return                            ; return with <CP_none> bit high
WFS2    ; we are here if there is not silence
  call  del100ms                    ; keep waiting
  decfsz tempWFS,f                  ; all finished when tempWFS=0
  goto  WFS1                        ; start a new test if tempWFS<>0
  bcf   RetValues,CP_none
```

```
   bcf   RetValues,CP_dial
   bcf   RetValues,CP_ring
   bcf   RetValues,CP_busy
   return                              ; return with <CP_none> bit low
;
SendOneDTMFCode
; This subroutine sends one DTMF code
; Assumes:    1. The DTMF generator (U4) is enabled and in serial mode
;             2. The decimal digit is contained in the low nibble of register w
; This subroutine transmits the code for about 500ms and then waits silent
; for about 500ms before returning.
   ; Step #1 - convert the decimal digit "0" to the DTMF code "A"
   andlw 0x0F                          ; keep only the low nibble
   btfsc status,zero                   ; skip if digit is non-zero
   movlw 0x0A                          ; 0x0A is the DTMF code for zero
   movwf tempSODC                      ; save the DTMF code in tempSODC
   ; Step #2 - transmit the least-significant bit
   bsf   portCmirror,DTMFClock
   bcf   portCmirror,DTMFData
   rrf   tempSODC,f                    ; rotate right through the carry bit
   btfsc status,carry
   bsf   portCmirror,DTMFData
   movf  portCmirror,w
   movwf portC                         ; send data bit with the clock high
   call  del100us                      ; hold clock high for 100us
   bcf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC                         ; assert the clock low
   call  del100us                      ; hold clock low for 100us
   ; Step #3 - transmit the second least-significant bit
   bsf   portCmirror,DTMFClock
   bcf   portCmirror,DTMFData
   rrf   tempSODC,f                    ; rotate right through the carry bit
   btfsc status,carry
   bsf   portCmirror,DTMFData
   movf  portCmirror,w
   movwf portC                         ; send data bit with the clock high
   call  del100us                      ; hold clock high for 100us
   bcf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC                         ; assert the clock low
   call  del100us                      ; hold clock low for 100us
   ; Step #4 - transmit the second most-significant bit
   bsf   portCmirror,DTMFClock
   bcf   portCmirror,DTMFData
   rrf   tempSODC,f                    ; rotate right through the carry bit
   btfsc status,carry
   bsf   portCmirror,DTMFData
   movf  portCmirror,w
   movwf portC                         ; send data bit with the clock high
   call  del100us                      ; hold clock high for 100us
   bcf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC                         ; assert the clock low
   call  del100us                      ; hold clock low for 100us
   ; Step #4 - transmit the most-significant bit
   bsf   portCmirror,DTMFClock
```

```
   bcf   portCmirror,DTMFData
   rrf   tempSODC,f                    ; rotate right through the carry bit
   btfsc status,carry
   bsf   portCmirror,DTMFData
   movf  portCmirror,w
   movwf portC                         ; send data bit with the clock high
   call  del100us                      ; hold clock high for 100us
   bcf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC                         ; assert the clock low
   call  del100us                      ; hold clock low for 100us
   ; Step #5 - a fifth bit must be sent.  It is zero for all ten digits.
   bsf   portCmirror,DTMFClock
   bcf   portCmirror,DTMFData
   movf  portCmirror,w
   movwf portC                         ; send the zero with the clock high
   call  del100us                      ; hold clock high for 100us
   bcf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC                         ; assert the clock low
   call  del100us                      ; hold clock low for 100us
   ; Step #6 - return the clock high
   bsf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC
   ; Step #7 - send the DTMF tone for one-half second
   call  del500ms
   ; Step #8 - send b'11111' to stop transmitting the DTMF tone
   movlw 0x05
   movwf tempSODC                      ; use register tempSODC as a counter
   bsf   portCmirror,DTMFData
SODC1
   bsf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC                         ; send the one with the clock high
   call  del100us                      ; hold clock high for 100us
   bcf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC                         ; assert the clock low
   call  del100us                      ; hold clock low for 100us
   decfsz tempSODC,f
   goto  SODC1                         ; repeat five times
   ; Step #9 - return the clock high
   bsf   portCmirror,DTMFClock
   movf  portCmirror,w
   movwf portC
   ; Step #10 - Remain silent for one-half second
   call  del500ms
   return
;
DialTelephoneNumber
; Assumes:    1. LenTelNum holds the number of digits in the number
;             2. The decimal digits are stored in the low nibbles of
;                   TelNum1, TelNum2, TelNum3, etc.
;             3. portC is configured for output
;             4. The state of the DTMF generator (U4) is unknown
; Step #1 - Save the number of digits in tempDialTN for use as a counter
```

```
movf  LenTelNum,w
movwf tempDialTN
; Step #2 - Configure the DTMF generator
bcf   portCmirror,DTMFEnable
bsf   portCmirror,DTMFClock
bsf   portCmirror,DTMFData
movf  portCmirror,w
movwf portC
call  del10ms                         ; wait for the oscillator to steady
; Step #3 - Transmit the digits one-by-one.  Subroutine SendOneDTMFCode
; includes a half-second of silence after each tone.
movf  TelNum1,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum2,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum3,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum4,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum5,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum6,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum7,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum8,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum9,w
call  SendOneDTMFCode
decf  tempDialTN,f
btfsc status,zero
goto  DTNDone
movf  TelNum10,w
```

```
   call  SendOneDTMFCode
   decf  tempDialTN,f
   btfsc status,zero
   goto  DTNDone
   movf  TelNum11,w
   call  SendOneDTMFCode
DTNDone
   bsf   portCmirror,DTMFEnable
   bsf   portCmirror,DTMFClock
   bsf   portCmirror,DTMFData
   movf  portCmirror,w
   movwf portC
   return
;
; ********************************************************************************
; Block F - Routines which send messages to the LCD
; Subroutines:-
;    DisplayTelephoneNumber()
;    DisplayNumericField()
;    DisplayProgressReport()
; ********************************************************************************
;
DisplayTelephoneNumber
;   Assumes:   1.Number of digits is stored in register LenTelNum
;              2. Digits are stored in registers TelNum1, TelNum2, ...
; Step #1: Put the cursor at the correct position in the display
   call  PutCursorAtStartOfLine2InDisplay
   movf  StartAdd,w                    ; set the DRAM address
   xorlw 0x80                          ; DB7=b'1' in the instruction
   call  WriteOneInstructionByteToLCD
   call  del100us                      ; wait at least 38us
; Step #2: Display the digits one-by-one
   movf  LenTelNum,w
   movwf tempDispTN                    ; use tempDispTN to count LenTelNum
   movf  tempDispTN,f                  ; set the zero flag
   btfsc status,zero
   goto  DTNDoneDigits                 ; goto if LenTelNum=0
   movf  TelNum1,w                     ; else, display TelNum1
   call  WriteOneDataByteToLCD
   decf  tempDispTN,f                  ; this also sets the zero flag
   btfsc status,zero
   goto  DTNDoneDigits                 ; goto if LenTelNum=1
   movf  TelNum2,w                     ; else, display TelNum2
   call  WriteOneDataByteToLCD
   decf  tempDispTN,f
   btfsc status,zero
   goto  DTNDoneDigits                 ; goto if LenTelNum=2
   movf  TelNum3,w                     ; else, display TelNum3
   call  WriteOneDataByteToLCD
   decf  tempDispTN,f
   btfsc status,zero
   goto  DTNDoneDigits                 ; goto if LenTelNum=3
   movf  TelNum4,w                     ; else, display TelNum4
   call  WriteOneDataByteToLCD
   decf  tempDispTN,f
   btfsc status,zero
   goto  DTNDoneDigits                 ; goto if LenTelNum=4
```

```
  movf  TelNum5,w                   ; else, display TelNum5
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=5
  movf  TelNum6,w                   ; else, display TelNum6
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=6
  movf  TelNum7,w                   ; else, display TelNum7
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=7
  movf  TelNum8,w                   ; else, display TelNum8
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=8
  movf  TelNum9,w                   ; else, display TelNum9
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=9
  movf  TelNum10,w                  ; else, display TelNum10
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=10
  movf  TelNum11,w                  ; else, display TelNum11
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=11
  movf  TelNum12,w                  ; else, display TelNum12
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=12
  movf  TelNum13,w                  ; else, display TelNum13
  call  WriteOneDataByteToLCD
  decf  tempDispTN,f
  btfsc status,zero
  goto  DTNDoneDigits               ; goto if LenTelNum=13
  movf  TelNum14,w                  ; else, display TelNum14
  call  WriteOneDataByteToLCD
DTNDoneDigits
  ; Step #3 - Display a following blank, to cover up a Backspace
  movlw 0x20
  call  WriteOneDataByteToLCD
  ; Step #4 - Place the cursor after the last digit
  movf  StartAdd,w
  addwf LenTelNum,w
  xorlw 0x80                        ; DB7=b'1' in the instruction
  call  WriteOneInstructionByteToLCD
  call  del100us                    ; wait at least 38us
```

```
   return
;
DisplayNumericField
;   Assumes:  1. Initial cursor address is stored in register StartAdd
;             2. Number of digits in field is stored in register NumDigits
;             3. Digits are stored in registers digit1, digit2, ...
; Step #1: Put the cursor at the correct position in the display
   movf  StartAdd,w                    ; set the DRAM address
   xorlw 0x80                          ; DB7=b'1' in the instruction
   call  WriteOneInstructionByteToLCD
   call  del100us                      ; wait at least 38us
   ; Step #2: Display the digits one-by-one
   movf  NumDigits,w
   movwf tempDispNF                    ; use tempDispNF to count NumDigits
   movf  tempDispNF,f                  ; set the zero flag
   btfsc status,zero
   goto  DNFDoneDigits                 ; goto if NumDigits=0
   movf  Digit1,w                      ; else, display Digit1
   call  WriteOneDataByteToLCD
   decf  tempDispNF,f                  ; this also sets the zero flag
   btfsc status,zero
   goto  DNFDoneDigits                 ; goto if NumDigits=1
   movf  Digit2,w                      ; else, display Digit2
   call  WriteOneDataByteToLCD
   decf  tempDispNF,f
   btfsc status,zero
   goto  DNFDoneDigits                 ; goto if NumDigits=2
   movf  Digit3,w                      ; else, display Digit3
   call  WriteOneDataByteToLCD
DNFDoneDigits
   ; Step #3 - Display a following blank, to cover up a Backspace
   movlw 0x20
   call  WriteOneDataByteToLCD
   ; Step #4 - Place the cursor after the last digit
   movf  StartAdd,w
   addwf NumDigits,w
   xorlw 0x80                          ; DB7=b'1' in the instruction
   call  WriteOneInstructionByteToLCD
   call  del100us                      ; wait at least 38us
   return
;
DisplayProgressReport
   ; Step #1: Display the LCD template
   call  ClearLCDDisplay
   movlw 0xCC
   movwf MSGstart
   movlw 0x0F
   movwf MSGlength
   call  DisplayMessageU               ; Message29 = "Progress  Good:"
   call  PutCursorAtStartOfLine2InDisplay
   movlw 0xDB
   movwf MSGstart
   movlw 0x0F
   movwf MSGlength
   call  DisplayMessageU               ; Message30 = "Busy:     Rang:"
   ; Step #2: Put the cursor at the correct position for "Good"
   movlw 0x8F                          ; DB7=b'1' in the instruction
```

```
                                    ; LCD address is 0x0F
   call  WriteOneInstructionByteToLCD
   ; Step #3: Display NumGoodCalls
   movf  NumGoodCalls,w
   call  ConvertBinaryToCharacterField
   movf  char1,w
   call  WriteOneDataByteToLCD
   movf  char2,w
   call  WriteOneDataByteToLCD
   movf  char3,w
   call  WriteOneDataByteToLCD
   ; Step #4: Put the cursor at the correct position for "Busy"
   movlw 0xC5                       ; DB7=b'1' in the instruction
                                    ; LCD address is 0x45
   call  WriteOneInstructionByteToLCD
   ; Step #5: Display NumBusyCalls
   movf  NumBusyCalls,w
   call  ConvertBinaryToCharacterField
   movf  char1,w
   call  WriteOneDataByteToLCD
   movf  char2,w
   call  WriteOneDataByteToLCD
   movf  char3,w
   call  WriteOneDataByteToLCD
   ; Step #6: Put the cursor at the correct position for "Rang"
   movlw 0xCF                       ; DB7=b'1' in the instruction
                                    ; LCD address is 0x4F
   call  WriteOneInstructionByteToLCD
   ; Step #7: Display NumRangCalls
   movf  NumRangCalls,w
   call  ConvertBinaryToCharacterField
   movf  char1,w
   call  WriteOneDataByteToLCD
   movf  char2,w
   call  WriteOneDataByteToLCD
   movf  char3,w
   call  WriteOneDataByteToLCD
   call  HideCursor
   return
;
; ***************************************************************************
; Block G - Routines which interpret key strokes
; Subroutines:-
;    GetTelephoneNumber()
;    GetNumericField()
; ***************************************************************************
;
GetTelephoneNumber
; This subroutine reads a telephone number from the keypad.  It does not do any
; validation of the entry.  This subroutine updates the LCD display on the fly.
; A maximum of 14 digits can be entered.
; Assumes:    1. Configuration of portB is unknown
;             2. The LCD display is unknown
; Returns:    1. Number of digits in register LenTelNum
;             2. Digits are stored L-to-R in TelNum1, TelNum2, ...
   clrf  LenTelNum
GTN1
```

```
   ; Step #1: wait for the next key to be pressed
   call  WaitForKey
   ; Step #2: process the Enter (#=0x23) key
   movlw 0x23
   xorwf KeyPressed,w                  ; Compare new key  with Enter key
   btfsc status,zero                   ; skip if not the Enter key
   return                              ; return if Enter key was pressed
   ; Step #3: process numeric keys 0-9
   movlw 0x2A
   xorwf KeyPressed,w                  ; compare new key with Backspace key
   btfsc status,zero                   ; skip if not the Backspace key
   goto  GTNBackspace                  ; goto if Backspace key was pressed
   incf  LenTelNum,f                   ; increment the number of digits
   movlw 0x01                          ; save key code in register digit*
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN2                          ; goto GTN2 if LenTelNum <> 1, ...
   movf  KeyPressed,w                  ; ... else save in TelNum1
   movwf TelNum1
   goto  GTNUpdateDisplay
GTN2
   movlw 0x02
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN3                          ; goto GTN3 if LenTelNum <> 2, ...
   movf  KeyPressed,w                  ; ... else save in TelNum2
   movwf TelNum2
   goto  GTNUpdateDisplay
GTN3
   movlw 0x03
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN4                          ; goto GTN4 if LenTelNum <> 3, ...
   movf  KeyPressed,w                  ; ... else save in TelNum3
   movwf TelNum3
   goto  GTNUpdateDisplay
GTN4
   movlw 0x04
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN5                          ; goto GTN5 if LenTelNum <> 4, ...
   movf  KeyPressed,w                  ; ... else save in TelNum4
   movwf TelNum4
   goto  GTNUpdateDisplay
GTN5
   movlw 0x05
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN6                          ; goto GTN6 if LenTelNum <> 5, ...
   movf  KeyPressed,w                  ; ... else save in TelNum5
   movwf TelNum5
   goto  GTNUpdateDisplay
GTN6
   movlw 0x06
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN7                          ; goto GTN7 if LenTelNum <> 6, ...
```

```
   movf  KeyPressed,w                ; ... else save in TelNum6
   movwf TelNum6
   goto  GTNUpdateDisplay
GTN7
   movlw 0x07
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN8                        ; goto GTN8 if LenTelNum <> 7, ...
   movf  KeyPressed,w                ; ... else save in TelNum7
   movwf TelNum7
   goto  GTNUpdateDisplay
GTN8
   movlw 0x08
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN9                        ; goto GTN9 if LenTelNum <> 8, ...
   movf  KeyPressed,w                ; ... else save in TelNum8
   movwf TelNum8
   goto  GTNUpdateDisplay
GTN9
   movlw 0x09
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN10                       ; goto GTN10 if LenTelNum <> 9, ...
   movf  KeyPressed,w                ; ... else save in TelNum9
   movwf TelNum9
   goto  GTNUpdateDisplay
GTN10
   movlw 0x0A
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN11                       ; goto GTN11 if LenTelNum <> 10, ...
   movf  KeyPressed,w                ; ... else save in TelNum10
   movwf TelNum10
   goto  GTNUpdateDisplay
GTN11
   movlw 0x0B
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN12                       ; goto GTN12 if LenTelNum <> 11, ...
   movf  KeyPressed,w                ; ... else save in TelNum11
   movwf TelNum11
   goto  GTNUpdateDisplay
GTN12
   movlw 0x0C
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN13                       ; goto GTN13 if LenTelNum <> 12, ...
   movf  KeyPressed,w                ; ... else save in TelNum12
   movwf TelNum12
   goto  GTNUpdateDisplay
GTN13
   movlw 0x0D
   xorwf LenTelNum,w
   btfss status,zero
   goto  GTN14                       ; goto GTN14 if LenTelNum <> 13, ...
   movf  KeyPressed,w                ; ... else save in TelNum13
```

```
   movwf TelNum13
   goto  GTNUpdateDisplay
GTN14
   movf  KeyPressed,w                  ; do not store more than 14 digits
   movwf TelNum14
   movlw 0x0E                          ; set LenTelNum = 14
   movwf LenTelNum
GTNUpdateDisplay
   call  DisplayTelephoneNumber
   goto  GTN1
GTNBackspace
   ; Step #4: process the Backspace (*=0x2A) key
   movlw 0x00
   xorwf LenTelNum,w
   btfsc status,zero
   goto  GTN1                          ; do nothing if there are no digits
   decf  LenTelNum,f                   ; decrement the number of digits
   call  DisplayTelephoneNumber
   goto  GTN1
;
GetNumericField
; This subroutine reads a numeric entry from the keypad.  The numeric entry can
; be up to three decimal digits.  This subroutine does not do any validation of
; the field.  This subroutine updates the LCD display on the fly.
; Assumes:    1. Configuration of portB is unknown
;             2. A prompt message has been displayed on the LCD
;             3. Initial cursor address is stored in register StartAdd
; Returns:    1. Number of digits in field is stored in register NumDigits
;             2. Digits are stored in registers Digit1, Digit2, Digit3
   clrf  NumDigits                     ; set the default to no digits
GNF1
   ; Step #1: wait for the next key to be pressed
   call  WaitForKey
   ; Step #2: process the Enter (#=0x23) key
   movlw 0x23
   xorwf KeyPressed,w                  ; Compare new key with Enter key
   btfsc status,zero                   ; skip if not the Enter key
   return                              ; return if Enter key was pressed
   ; Step #3: process numeric keys 0-9
   movlw 0x2A
   xorwf KeyPressed,w                  ; compare new key with Backspace key
   btfsc status,zero                   ; skip if not the Backspace key
   goto  GNFBackspace                  ; goto if Backspace key was pressed
   incf  NumDigits,f                   ; increment the number of digits
   movlw 0x01
   xorwf NumDigits,w
   btfss status,zero
   goto  GNF2                          ; goto GNF2 if NumDigits <> 1, ...
   movf  KeyPressed,w                  ; ... else save in Digit1
   movwf Digit1
   goto  GNFUpdateDisplay
GNF2
   movlw 0x02
   xorwf NumDigits,w
   btfss status,zero
   goto  GNF3                          ; goto GNF3 if NumDigits <> 2, ...
   movf  KeyPressed,w                  ; ... else save in Digit2
```

```
   movwf Digit2
   goto  GNFUpdateDisplay
GNF3
   movf  KeyPressed,w                  ; do not store more than 3 digits
   movwf Digit3
   movlw 0x03                          ; set NumDigits = 3
   movwf NumDigits
   goto  GNFUpdateDisplay
GNFUpdateDisplay
   call  DisplayNumericField
   goto  GNF1
GNFBackspace
   ; Step #4: process the Backspace (*=0x2A) key
   movlw 0x00
   xorwf NumDigits,w
   btfsc status,zero
   goto  GNF1                          ; do nothing if there are no digits
   decf  NumDigits,f                   ; decrement the number of digits
   call  DisplayNumericField
   goto  GNF1
;
; ********************************************************************************
; Block H - Miscellaneous subroutines
; Subroutines:-
;    ConvertNumericFieldToBinary()
;    ConvertBinaryToCharacterField()
;    CalcAndDoWaitToCall() - calculates a new wait period and then waits
;    CalcAndDoWaitOnBusy() - calculates a new wait period and then waits
;    Random() - random number generator
;    Multiply() - multiplies register w by register random
;    del1min - uninterrupted delay of approximately one minute
;    del3sec - uninterrupted delay of approximately ten seconds
;    del3sec - uninterrupted delay of approximately three seconds
;    del500ms - uninterrupted delay of 503.082ms
;    del100ms - uninterrupted delay of 100.618ms
;    del10ms - uninterrupted delay of 10.0604ms
;    del1ms - uninterrupted delay of 1.018ms
;    del100us - uninterrupted delay of 100.4us
; ********************************************************************************
;
ConvertNumericFieldToBinary
; This subroutine converts numeric fields of up to d'255' into a single-byte
; binary value.  The input data is stored in the registers Digit1 (the left-most
; digit) through Digit3.  The number of digits which are relevant is stored in
; register NumDigits.  It is assumed that NumDigits is in the range 0 through 3.
; The digits themselves are stored with the values used by the LCD instructions.
; Therefore, the high-order nibble of each Digit* needs to be zeroed out.  This
; subroutine automatically converts overflows to d'255'.  It also automatically
; converts blank numeric fields (i.e., only the enter key was pressed) into
; d'1'.
; Assumes:    1. Data is in Digit1 (most-significant), Digit2 and Digit3
;             2. Number of relevant digits is in NumDigits
; Returns:    1. The binary value is stored in register CurValue
;             2. Digit1, Digit2 and Digit3 are left unchanged
   movlw 0x01
   movwf CurValue                      ; set CurValue=1 as default value
   movlw 0x00
```

```
   xorwf NumDigits,w
   btfsc status,zero
   return                              ; return if there are no digits
   ; Step #1 - Convert from LCD codes to binary
   movlw 0x0F
   andwf Digit1,f
   andwf Digit2,f
   andwf Digit3,f
   ; Step #2 - Process the most-significant digit
   movf  Digit1,w
   movwf CurValue
   movlw 0x01
   xorwf NumDigits,w
   btfsc status,zero
   return                              ; return if there is only 1 digit
   ; Step #3 - Multiply CurValue by 10, by adding Digit1 9 more times
   movlw 0x09
   movwf tempCNFTB1                    ; tempCNFTB1 counts additions
CNFTB1
   movf  Digit1,w
   addwf CurValue,f
   decf  tempCNFTB1,f
   btfss status,zero
   goto  CNFTB1
   ; Step #4 - Add the second most-significant digit
   movf  Digit2,w
   addwf CurValue,f
   movlw 0x02
   xorwf NumDigits,w
   btfsc status,zero
   return                              ; return if there are only 2 digits
   ; Step #5 - Multiply CurValue by 10, by adding 10 times
   movlw 0x0A
   movwf tempCNFTB1                    ; tempCNFTB1 counts additions
   clrf  tempCNFTB2                    ; tempCNFTB2 holds the product
CNFTB2
   movf  CurValue,w
   addwf tempCNFTB2,f
   btfsc status,carry
   goto  CNFTBOverflow                 ; on overflow, goto CNFTBOverflow
   decf  tempCNFTB1,f
   btfss status,zero
   goto  CNFTB2
   ; Step #6 - Add the least-significant digit to tempCNFTB2
   movf  tempCNFTB2,w
   addwf Digit3,w
   btfsc status,carry
   goto  CNFTBOverflow                 ; on overflow, goto CNFTBOverflow
   movwf CurValue
   return
CNFTBOverflow
   movlw 0xFF
   movwf CurValue
   return
;
ConvertBinaryToCharacterField
; This subroutine converts single-byte binary value to a three-character field
```

```
; for display on the LCD.  numeric fields of up to d'255' into a single-byte binary
; value.  The input data is stored in the registers digit1 (the left-most digit)
; through digit3.  The number of digits which are relevant is stored in register
; NumDigits.  It is assumed that NumDigits is in the range 0 through 3.  The digits
; themselves are stored with the values used by the LCD instructions.  Therefore, the
; high-order nibble of each digit* needs to be zeroed out.
; Assumes:    1. The binary value is stored in register w
; Returns:    1. Field is in char1 (most-significant), char2 and char3
;             2. Leading zeroes are blanked out
;             3. The characters are in LCD-format
   movwf tempCBTCF
   movlw 0x30
   movwf char1
   movwf char2
   movwf char3
CBTCF100s
   movlw 0x64                          ; 0x64 = d'100'
   subwf tempCBTCF,w                   ; tempCBTCF - d'100' --> w
   btfss status,carry
   goto  CBTCF10s
   movwf tempCBTCF
   incf  char1,f
   goto  CBTCF100s
CBTCF10s
   movlw 0x0A                          ; 0x0A = d'10'
   subwf tempCBTCF,w                   ; tempCBTCF - d'10' --> w
   btfss status,carry
   goto  CBTCF1s
   movwf tempCBTCF
   incf  char2,f
   goto  CBTCF10s
CBTCF1s
   movf  tempCBTCF,w
   addwf char3,f
   movlw 0x30
   xorwf char1,w
   btfss status,zero
   return
   movlw 0x20
   movwf char1
   movlw 0x30
   xorwf char2,w
   btfss status,zero
   return
   movlw 0x20
   movwf char2
   return
;
CalcAndDoWaitToCall
; There are two cases.
; Case A:
;   If MaxWaitToCall is two or more, then it represents the maximum wait time in
;   minutes before making the next call.  MinWaitToCall is the minimum wait time
;   in minutes before making the next call.  The difference between these two
;   numbers was calculated right after the data input ended, and the difference
;   is stored in DeltaWaitToCall.  DeltaWaitToCall is multiplied by a random
;   number between 0 and 1, and the product added to MinWaitToCall, to get the
```

```
;   actual number of minutes to wait before calling again.
; Case B:
;   If MaxWaitToCall is equal to zero or one, then it is ignored and MinWaitToCall
;   represents the maximum wait time in seconds before making the next call.  The
;   minimum wait time in seconds is assumed to be one second.  MinWaitToCall is
;   multiplied by a random number between 0 and 1, and the product added to
;   one, to get the actual number of seconds to wait before calling again.
   movf  MaxWaitToCall,w
   andlw 0xFE                          ; 0xFE = b'11111110'.  The result ...
                                       ; ... is zero if MaxWaitToCall=0 or 1
   btfsc status,zero                   ; skip if MaxWaitToCall >= 2
   goto  CADWTC2                       ; goto if MaxWaitToCall=0 or 1
   call  Random                        ; here if the wait is measured in minutes
   movf  DeltaWaitToCall,w
   call  Multiply
   addwf MinWaitToCall,w               ; add the minimum wait time
   movwf tempCADWTC
CADWTC1
   call  del1min                       ; wait one minute
   decfsz tempCADWTC,f
   goto  CADWTC1                       ; repeat tempCADWTC times
   return
CADWTC2
   call  Random                        ; here if the wait is measured in seconds
   movf  MinWaitToCall,w
   call  Multiply
   addlw 0x01                          ; add the minimum wait time
   movwf tempCADWTC
CADWTC3
   call  del500ms                      ; wait one second
   call  del500ms
   decfsz tempCADWTC,f                 ; repeat tempCADWTC times
   goto  CADWTC3
   return
;
CalcAndDoWaitOnBusy
   call  Random
   movf  DeltaWaitOnBusy,w
   call  Multiply
   addwf MinWaitOnBusy,w
   movwf tempCADWOB
CADWOB1
   call  del500ms
   call  del500ms
   decfsz tempCADWOB,f
   goto  CADWOB1
   return
;
Random
; This subroutine returns an 8-bit random number in register random.
; The algorithm is a type of Linear-Feedback Shift Register.
   rlf   random,w
   rlf   random,w
   btfsc random,4
   xorlw 0x01
   btfsc random,5
   xorlw 0x01
```

```
   btfsc random,3
   xorlw 0x01
   movwf random
   return
;
Multiply
; This subroutine multiplies register random by register w and returns
; the result in register w.  Register random is not changed.  Note that
; register random is treated as a fraction between zero and one.  Therefore,
; there will not be an overflow of the total.  This subroutine returns only
; the high byte of the 16-bit product.
   movwf tempM1H                        ; tempM1H is multiplicand, high byte
   clrf  tempM1L                        ; tempM1L is multiplicand, low byte
   movf  random,w
   movwf tempM2                         ; tempM2 is multiplier
   clrf  tempM3H                        ; tempM3H is product, high byte
   clrf  tempM3L                        ; tempM3L is product, low byte
   movlw 0x07
   movwf tempM4                         ; tempM4 is a 7-loop counter
M1
   bcf   status,carry
   rrf   tempM1H,f                      ; divide multiplicand by 2, 4, 8 ...
   rrf   tempM1L,f
   rlf   tempM2,f                       ; check next multiplier MSB
   btfss status,carry
   goto  M2
   ; If carry=1, add shifted multiplicand to product
   ; If carry=0, leave product alone
   movf  tempM1L,w                      ; add the two low ...
   addwf tempM3L,f                      ; ... bytes together
   btfsc status,carry
   incf  tempM3H,f                      ; if carry, then increment high byte
   movf  tempM1H,w                      ; add the two high ...
   addwf tempM3H,f                      ; ... bytes together
M2
   decfsz tempM4,f
   goto  M1                             ; loop 7 times
   movf  tempM3H,w                      ; return only high byte of product
   return
;
del1min
; This subroutine is an uninterrupted delay of approximately one minute
   movlw 0x78                           ; 0x78 = d'120'
   movwf count4
del1min1
   call  del500ms                       ; call del500ms 120 times
   decfsz count4,f
   goto  del1min1
   return
;
del10sec
; This subroutine is an uninterrupted delay of approximately ten seconds
   movlw 0x14                           ; 0x14 = d'20'
   movwf count4
del10sec1
   call  del500ms                       ; call del500ms 20 times
   decfsz count4,f
```

```
   goto  del10sec1
   return
;
del3sec
; This subroutine is an uninterrupted delay of approximately three seconds
   movlw 0x06                          ; 0x06 = d'6'
   movwf count4
del3sec1
   call  del500ms                      ; call del500ms six times
   decfsz count4,f
   goto  del3sec1
   return
;
del500ms
; This subroutine is an uninterrupted delay of 503.082 milli-seconds
   movlw 0x32                          ; 1 cycle (Note that 0x32 = d'50'.)
   movwf count3                        ; 1
del500ms1
   call  del10ms                       ; 50 x 25,151 = 1,257,550
   decfsz count3,f                     ; (49 x 1) + (1 x 2) = 51
   goto  del500ms1                     ; 49 x 2 = 98
   return                              ; 2
;                                      ; CALL adds 2 = 1,257,705 cycles
;                                      ; 1,257,705 x 400ns = 503,082,000ns
;
del100ms
; This subroutine is an uninterrupted delay of 100.618 milli-seconds
   movlw 0x0A                          ; 1 cycle  (Note that 0x0A = d'10'.)
   movwf count3                        ; 1
del100ms1
   call  del10ms                       ; 10 x 25,151 = 251,510
   decfsz count3,f                     ; (9 x 1) + (1 x 2) = 11
   goto  del100ms1                     ; 9 x 2 = 18
   return                              ; 2
;                                      ; CALL adds 2 =- 251,545 cycles
;                                      ; 251,545 x 400ns = 100,618,000ns
;
del10ms
; This subroutine is an uninterrupted delay of 10.0604 milli-seconds
   movlw 0x63                          ; 1 cycle  (Note that 0x63 = d'99'.)
   movwf count2                        ; 1
del10ms1
   call  del100us                      ; 99 x 251 = 24,849
   decfsz count2,f                     ; (98 x 1) + (1 x 2) = 100
   goto  del10ms1                      ; 98 x 2 = 196
   return                              ; 2
;                                      ; CALL adds 2 = 25,151 cycles
;                                      ; 25,151 x 400ns = 10,060,400ns
;
del1ms
; This subroutine is an uninterrupted delay of 1.018 milli-seconds
   movlw 0x0A                          ; 1 cycle  (Note that 0x0A = d'10'.)
   movwf count2                        ; 1
del1ms1
   call  del100us                      ; 10 x 251 = 2,510
   decfsz count2,f                     ; (9 x 1) + (1 x 2) = 11
```

```
   goto  del1ms1                        ; 9 x 2 = 18
   return                               ; 2
;                                       ; CALL adds 2 = 2,545 cycles
;                                       ; 2,545 x 400ns = 1,018,000ns
;
del100us
; This subroutine is an uninterrupted delay of 100.4 micro-seconds
   movlw 0x52                           ; 1 cycle  (Note that 0x52 = d'82'.)
   movwf count1                         ; 1
del100us1
   decfsz count1,f                      ; (81 x 1) + (1 x 2) = 83
   goto  del100us1                      ; 81 x 2 = 162
   return                               ; 2
;                                       ; CALL adds 2 = 251 cycles
;                                       ; 251 x 400ns = 100,400ns
;
; ***************************************************************************
; Block I - Routines used for debugging
;
; Uncomment DisplayRawCPData() to use for debugging
;DisplayRawCPData
; ; Most useful if called immediately after a call to ReadCallProgress.
; ; This subroutine displays the setting of two bits on the LCD.  The
; ; two bits are RetValues<RV_440Hz> and RetValues<RV_480Hz>, in that
; ; order from left to right.  The display is "HH", "HL", "LH" or "LL"
; ; to signify High or Low bits.  This subroutine returns 100us after
; ; writing the characters to the LCD.
; call  ClearLCDDisplay
; movlw 0x20                           ; Leading blank leaves room for
; call  WriteOneDataByteToLCD          ; cursor after next ClearLCDDisplay
; movlw 0x48                           ; H
; btfss RetValues,RV_440Hz
; movlw 0x4C                           ; L
; call  WriteOneDataByteToLCD
; movlw 0x48                           ; H
; btfss RetValues,RV_480Hz
; movlw 0x4C                           ; L
; call  WriteOneDataByteToLCD
; call  del100us
; return
;
; Uncomment subroutine DisplayCPCodes() to use for debugging
;DisplayCPCodes
; ; Most useful if called immediately after a call to ReadCallProgress.
; ; This subroutine displays "DT", "RG", "BS" or "SL" depending on
; ; which CP code subroutine ReadCallProgress has determined.  These are
; ; acronyms for dial tone, ringing, busy signal and silence,
; ; respectively.  This subroutine returns 100us after writing the
; ; characters to the LCD.
; call  ClearLCDDisplay
; movlw 0x20                           ; Leading blank leaves room for
; call  WriteOneDataByteToLCD          ; cursor after next ClearLCDDisplay
; btfsc RetValues,CP_dial
; goto  DCPC1
; movlw 0x44                           ; D
; call  WriteOneDataByteToLCD
; movlw 0x4C                           ; L
```

```
;  call  WriteOneDataByteToLCD
;  call  del100us
;  return
;DCPC1
;  btfsc RetValues,CP_ring
;  goto  DCPC2
;  movlw 0x52                          ; R
;  call  WriteOneDataByteToLCD
;  movlw 0x47                          ; G
;  call  WriteOneDataByteToLCD
;  call  del100us
;  return
;DCPC2
;  btfsc RetValues,CP_busy
;  goto  DCPC3
;  movlw 0x42                          ; B
;  call  WriteOneDataByteToLCD
;  movlw 0x53                          ; S
;  call  WriteOneDataByteToLCD
;  call  del100us
;  return
;DCPC3
;  movlw 0x53                          ; S
;  call  WriteOneDataByteToLCD
;  movlw 0x4C                          ; L
;  call  WriteOneDataByteToLCD
;  call  del100us
;  return
;
; ****************************************************************************
; Block J - ASCII look-up tables
; Subroutines:-
;  DisplayMessageL() - accesses the lower table MSGtableL
;  DisplayMessageU() - accesses the upper table MSGtableU
;
;       ***** Messages in the lower table, which starts at org 0x0680 *****
;  Message1   = "Telephone number:"    start=0=0x00,   length=17
;  Message2   = "Test dial? (1=Yes):"  start=17=0x11,  length=19
;  Message3   = "Turn on speaker"      start=36=0x24,  length=15
;  Message4   = "Mins between calls,"  start=51=0x33,  length=19
;  Message5   = "max (0-255): "        start=70=0x46,  length=13
;       Message6   = "min (1-max): "   start=83=0x53,  length=13
;  Message7   = "Min minutes > Max"    start=96=0x60,  length=17
;  Message8   = "Secs between calls,"  start=113=0x71, length=19
;  Message9   = "max (1-255): "        start=132=0x84, length=13
;  Message10  = "Number of telephone"  start=145=0x91, length=19
;  Message11  = "calls (10-255): "     start=164=0xA4, length=16
;  Message12  = "Number of ring"       start=180=0xB4, length=14
;  Message13  = "pulses (3-10): "      start=194=0xC2, length=15
;  Message14  = "Number of busy"       start=209=0xD1, length=14
;  Free space at end of page           start=223=0xDF, length=33
;
;  ***** Messages in the upper table, which starts at org 0x0700 *****
;  Message15  = "tries (5-50): "       start=0=0x00,   length=14
;  Message16  = "Max sec between busy" start=14=0x0E,  length=20
;  Message17  = "Min sec between busy" start=34=0x22,  length=20
;  Message18  = "tries (1-max): "      start=54=0x36,  length=15
```

```
;  Message19  = "Min seconds > Max"    start=69=0x45,  length=17
;  Message20  = "Max seconds to hold"  start=86=0x56,  length=19
;  Message21  = "callee (2-240): "     start=105=0x69, length=16
;  Message22  = "Line in use"          start=124=0x7C, length=11
;  Message23  = "No dial tone"         start=135=0x87, length=12
;  Message24  = "Dialling ..."         start=147=0x93, length=12
;  Message25  = "Busy ..."             start=159=0x9F, length=8
;  Message26  = "Dialling try #"       start=167=0xA7, length=14
;  Message27  = "Ringing ..."          start=181=0xB5, length=11
;  Message28  = "All finished"         start=192=0xC0, length=12
;  Message29  = "Progress  Good:"      start=204=0xCC, length=15
;  Message30  = "Busy:     Rang:"      start=219=0xDB, length=15
;  Message31  = "Tel system error"     start=234=0xEA, length=16
;  Free space at end of page           start=250=0xFA, length=6
; ********************************************************************************
;
DisplayMessageL
   ; This subroutine writes a message to the LCD.  It does not clear the
   ; display, or the parts of the display before and after the message.
   ; The starting index in the MSGtable is stored in register MSGstart.
   ; The number of characters in the message is stored in register
   ; MSGlength.  The message is written to the LCD starting at the current
   ; cursor location.  Both input registers are changed during execution.
   movlw HIGH MSGtableL                ; ensure that the high byte ...
   movwf PCLATH                        ; ... of PCL will point to MSGtableL
   movf  MSGstart,w                    ; w <- index in MSGtableL
   call  MSGtableL                     ; w <- byte from MSGtableL
   call  WriteOneDataByteToLCD
   incf  MSGstart,f                    ; increment to next byte in table
   decf  MSGlength,f
   btfss status,zero
   goto  DisplayMessageL               ; MSGlength is not zero, keep going
   return
;
DisplayMessageU
   ; This subroutine processes messages from the upper table.
   movlw HIGH MSGtableU                ; ensure that the high byte ...
   movwf PCLATH                        ; ... of PCL will point to MSGtableU
   movf  MSGstart,w                    ; w <- index in MSGtableU
   call  MSGtableU                     ; w <- byte from MSGtableU
   call  WriteOneDataByteToLCD
   incf  MSGstart,f                    ; increment to next byte in table
   decf  MSGlength,f
   btfss status,zero
   goto  DisplayMessageU               ; MSGlength is not zero, keep going
   return
;
   org   0x0600      ; leaves 33 unused words at the end of the page
MSGtableL
   addwf PCL,f                ; w + PCL -> PCL
   retlw 0x54                 ; T             0 - Message1 start
   retlw 0x65                 ; e             1
   retlw 0x6C                 ; l             2
   retlw 0x65                 ; e             3
   retlw 0x70                 ; p             4
   retlw 0x68                 ; h             5
   retlw 0x6F                 ; o             6
```

```
retlw 0x6E                  ; n             7
retlw 0x65                  ; e             8
retlw 0x20                  ; blank         9
retlw 0x6E                  ; n            10
retlw 0x75                  ; u            11
retlw 0x6D                  ; m            12
retlw 0x62                  ; b            13
retlw 0x65                  ; e            14
retlw 0x72                  ; r            15
retlw 0x3A                  ; :            16 - Message1 end
retlw 0x54                  ; T            17 - Message2 start
retlw 0x65                  ; e            18
retlw 0x73                  ; s            19
retlw 0x74                  ; t            20
retlw 0x20                  ; blank        21
retlw 0x64                  ; d            22
retlw 0x69                  ; i            23
retlw 0x61                  ; a            24
retlw 0x6C                  ; l            25
retlw 0x3F                  ; ?            26
retlw 0x20                  ; blank        27
retlw 0x28                  ; (            28
retlw 0x31                  ; 1            29
retlw 0x3D                  ; =            30
retlw 0x59                  ; Y            31
retlw 0x65                  ; e            32
retlw 0x73                  ; s            33
retlw 0x29                  ; )            34
retlw 0x3A                  ; :            35 - Message2 end
retlw 0x54                  ; T            36 - Message3 start
retlw 0x75                  ; u            37
retlw 0x72                  ; r            38
retlw 0x6E                  ; n            39
retlw 0x20                  ; blank        40
retlw 0x6F                  ; o            41
retlw 0x6E                  ; n            42
retlw 0x20                  ; blank        43
retlw 0x73                  ; s            44
retlw 0x70                  ; p            45
retlw 0x65                  ; e            46
retlw 0x61                  ; a            47
retlw 0x6B                  ; k            48
retlw 0x65                  ; e            49
retlw 0x72                  ; r            50 - Message3 end
retlw 0x4D                  ; M            51 - Message4 start
retlw 0x69                  ; i            52
retlw 0x6E                  ; n            53
retlw 0x73                  ; s            54
retlw 0x20                  ; blank        55
retlw 0x62                  ; b            56
retlw 0x65                  ; e            57
retlw 0x74                  ; t            58
retlw 0x77                  ; w            59
retlw 0x65                  ; e            60
retlw 0x65                  ; e            61
retlw 0x6E                  ; n            62
retlw 0x20                  ; blank        63
```

```
retlw 0x63                  ; c            64
retlw 0x61                  ; a            65
retlw 0x6C                  ; l            66
retlw 0x6C                  ; l            67
retlw 0x73                  ; s            68
retlw 0x2C                  ; ,            69 - Message4 end
retlw 0x6D                  ; m            70 - Message5 start
retlw 0x61                  ; a            71
retlw 0x78                  ; x            72
retlw 0x20                  ; blank        73
retlw 0x28                  ; (            74
retlw 0x30                  ; 0            75
retlw 0x2D                  ; -            76
retlw 0x32                  ; 2            77
retlw 0x35                  ; 5            78
retlw 0x35                  ; 5            79
retlw 0x29                  ; )            80
retlw 0x3A                  ; :            81
retlw 0x20                  ; blank        82 - Message5 end
retlw 0x6D                  ; m            83 - Message6 start
retlw 0x69                  ; i            84
retlw 0x6E                  ; n            85
retlw 0x20                  ; blank        86
retlw 0x28                  ; (            87
retlw 0x31                  ; 1            88
retlw 0x2D                  ; -            89
retlw 0x6D                  ; m            90
retlw 0x61                  ; a            91
retlw 0x78                  ; x            92
retlw 0x29                  ; )            93
retlw 0x3A                  ; :            94
retlw 0x20                  ; blank        95 - Message6 end
retlw 0x4D                  ; M            96 - Message7 start
retlw 0x69                  ; i            97
retlw 0x6E                  ; n            98
retlw 0x20                  ; blank        99
retlw 0x6D                  ; m           100
retlw 0x69                  ; i           101
retlw 0x6E                  ; n           102
retlw 0x75                  ; u           103
retlw 0x74                  ; t           104
retlw 0x65                  ; e           105
retlw 0x73                  ; s           106
retlw 0x20                  ; blank       107
retlw 0x3E                  ; >           108
retlw 0x20                  ; blank       109
retlw 0x4D                  ; M           110
retlw 0x61                  ; a           111
retlw 0x78                  ; x           112 - Message7 end
retlw 0x53                  ; S           113 - Message8 start
retlw 0x65                  ; e           114
retlw 0x63                  ; c           115
retlw 0x73                  ; s           116
retlw 0x20                  ; blank       117
retlw 0x62                  ; b           118
retlw 0x65                  ; e           119
retlw 0x74                  ; t           120
```

```
retlw 0x77                  ; w           121
retlw 0x65                  ; e           122
retlw 0x65                  ; e           123
retlw 0x6E                  ; n           124
retlw 0x20                  ; blank       125
retlw 0x63                  ; c           126
retlw 0x61                  ; a           127
retlw 0x6C                  ; l           128
retlw 0x6C                  ; l           129
retlw 0x73                  ; s           130
retlw 0x2C                  ; ,           131 - Message8 end
retlw 0x6D                  ; m           132 - Message9 start
retlw 0x61                  ; a           133
retlw 0x78                  ; x           134
retlw 0x20                  ; blank       135
retlw 0x28                  ; (           136
retlw 0x31                  ; 1           137
retlw 0x2D                  ; -           138
retlw 0x32                  ; 2           139
retlw 0x35                  ; 5           140
retlw 0x35                  ; 5           141
retlw 0x29                  ; )           142
retlw 0x3A                  ; :           143
retlw 0x20                  ; blank       144 - Message9 end
retlw 0x4E                  ; N           145 - Message10 start
retlw 0x75                  ; u           146
retlw 0x6D                  ; m           147
retlw 0x62                  ; b           148
retlw 0x65                  ; e           149
retlw 0x72                  ; r           150
retlw 0x20                  ; blank       151
retlw 0x6F                  ; o           152
retlw 0x66                  ; f           153
retlw 0x20                  ; blank       154
retlw 0x74                  ; t           155
retlw 0x65                  ; e           156
retlw 0x6C                  ; l           157
retlw 0x65                  ; e           158
retlw 0x70                  ; p           159
retlw 0x68                  ; h           160
retlw 0x6F                  ; o           161
retlw 0x6E                  ; n           162
retlw 0x65                  ; e           163 - Message10 end
retlw 0x63                  ; c           164 - Message11 start
retlw 0x61                  ; a           165
retlw 0x6C                  ; l           166
retlw 0x6C                  ; l           167
retlw 0x73                  ; s           168
retlw 0x20                  ; blank       169
retlw 0x28                  ; (           170
retlw 0x31                  ; 1           171
retlw 0x30                  ; 0           172
retlw 0x2D                  ; -           173
retlw 0x32                  ; 2           174
retlw 0x35                  ; 5           175
retlw 0x35                  ; 5           176
retlw 0x29                  ; )           177
```

```
   retlw 0x3A                 ; :           178
   retlw 0x20                 ; blank       179 - Message11 end
   retlw 0x4E                 ; N           180 - Message12 start
   retlw 0x75                 ; u           181
   retlw 0x6D                 ; m           182
   retlw 0x62                 ; b           183
   retlw 0x65                 ; e           184
   retlw 0x72                 ; r           185
   retlw 0x20                 ; blank       186
   retlw 0x6F                 ; o           187
   retlw 0x66                 ; f           188
   retlw 0x20                 ; blank       189
   retlw 0x72                 ; r           190
   retlw 0x69                 ; i           191
   retlw 0x6E                 ; n           192
   retlw 0x67                 ; g           193 - Message12 end
   retlw 0x70                 ; p           194 - Message13 start
   retlw 0x75                 ; u           195
   retlw 0x6C                 ; l           196
   retlw 0x73                 ; s           197
   retlw 0x65                 ; e           198
   retlw 0x73                 ; s           199
   retlw 0x20                 ; blank       200
   retlw 0x28                 ; (           201
   retlw 0x33                 ; 3           202
   retlw 0x2D                 ; -           203
   retlw 0x31                 ; 1           204
   retlw 0x30                 ; 0           205
   retlw 0x29                 ; )           206
   retlw 0x3A                 ; :           207
   retlw 0x20                 ; blank       208 - Message13 end
   retlw 0x4E                 ; N           209 - Message14 start
   retlw 0x75                 ; u           210
   retlw 0x6D                 ; m           211
   retlw 0x62                 ; b           212
   retlw 0x65                 ; e           213
   retlw 0x72                 ; r           214
   retlw 0x20                 ; blank       215
   retlw 0x6F                 ; o           216
   retlw 0x66                 ; f           217
   retlw 0x20                 ; blank       218
   retlw 0x62                 ; b           219
   retlw 0x75                 ; u           220
   retlw 0x73                 ; s           221
   retlw 0x79                 ; y           222 - Message14 end
;
   org   0x0700      ; leaves 6 unused words at the end of the page
MSGtableU
   addwf PCL,f                ; w + PCL -> PCL
   retlw 0x74                 ; t             0 - Message15 start
   retlw 0x72                 ; r             1
   retlw 0x69                 ; i             2
   retlw 0x65                 ; e             3
   retlw 0x73                 ; s             4
   retlw 0x20                 ; blank         5
   retlw 0x28                 ; (             6
   retlw 0x35                 ; 5             7
```

```
retlw 0x2D                  ; -             8
retlw 0x35                  ; 5             9
retlw 0x30                  ; 0            10
retlw 0x29                  ; )            11
retlw 0x3A                  ; :            12
retlw 0x20                  ; blank        13 - Message15 end
retlw 0x4D                  ; M            14 - Message16 start
retlw 0x61                  ; a            15
retlw 0x78                  ; x            16
retlw 0x20                  ; blank        17
retlw 0x73                  ; s            18
retlw 0x65                  ; e            19
retlw 0x63                  ; c            20
retlw 0x20                  ; blank        21
retlw 0x62                  ; b            22
retlw 0x65                  ; e            23
retlw 0x74                  ; t            24
retlw 0x77                  ; w            25
retlw 0x65                  ; e            26
retlw 0x65                  ; e            27
retlw 0x6E                  ; n            28
retlw 0x20                  ; blank        29
retlw 0x62                  ; b            30
retlw 0x75                  ; u            31
retlw 0x73                  ; s            32
retlw 0x79                  ; y            33 - Message16 end
retlw 0x4D                  ; M            34 - Message17 start
retlw 0x69                  ; i            35
retlw 0x6E                  ; n            36
retlw 0x20                  ; blank        37
retlw 0x73                  ; s            38
retlw 0x65                  ; e            39
retlw 0x63                  ; c            40
retlw 0x20                  ; blank        41
retlw 0x62                  ; b            42
retlw 0x65                  ; e            43
retlw 0x74                  ; t            44
retlw 0x77                  ; w            45
retlw 0x65                  ; e            46
retlw 0x65                  ; e            47
retlw 0x6E                  ; n            48
retlw 0x20                  ; blank        49
retlw 0x62                  ; b            50
retlw 0x75                  ; u            51
retlw 0x73                  ; s            52
retlw 0x79                  ; y            53 - Message17 end
retlw 0x74                  ; t            54 - Message18 start
retlw 0x72                  ; r            55
retlw 0x69                  ; i            56
retlw 0x65                  ; e            57
retlw 0x73                  ; s            58
retlw 0x20                  ; blank        59
retlw 0x28                  ; (            60
retlw 0x31                  ; 1            61
retlw 0x2D                  ; -            62
retlw 0x6D                  ; m            63
retlw 0x61                  ; a            64
```

```
retlw 0x78                  ; x           65
retlw 0x29                  ; )           66
retlw 0x3A                  ; :           67
retlw 0x20                  ; blank       68 - Message18 end
retlw 0x4D                  ; M           69 - Message19 start
retlw 0x69                  ; i           70
retlw 0x6E                  ; n           71
retlw 0x20                  ; blank       72
retlw 0x73                  ; s           73
retlw 0x65                  ; e           74
retlw 0x63                  ; c           75
retlw 0x6F                  ; o           76
retlw 0x6E                  ; n           77
retlw 0x64                  ; d           78
retlw 0x73                  ; s           79
retlw 0x20                  ; blank       80
retlw 0x3E                  ; >           81
retlw 0x20                  ; blank       82
retlw 0x4D                  ; M           83
retlw 0x61                  ; a           84
retlw 0x78                  ; x           85 - Message19 end
retlw 0x4D                  ; M           86 - Message20 start
retlw 0x61                  ; a           87
retlw 0x78                  ; x           88
retlw 0x20                  ; blank       89
retlw 0x73                  ; s           90
retlw 0x65                  ; e           91
retlw 0x63                  ; c           92
retlw 0x6F                  ; o           93
retlw 0x6E                  ; n           94
retlw 0x64                  ; d           95
retlw 0x73                  ; s           96
retlw 0x20                  ; blank       97
retlw 0x74                  ; t           98
retlw 0x6F                  ; o           99
retlw 0x20                  ; blank      100
retlw 0x68                  ; h          101
retlw 0x6F                  ; o          102
retlw 0x6C                  ; l          103
retlw 0x64                  ; d          104 - Message20 end
retlw 0x63                  ; c          105 - Message21 start
retlw 0x61                  ; a          106
retlw 0x6C                  ; l          107
retlw 0x6C                  ; l          108
retlw 0x65                  ; e          109
retlw 0x65                  ; e          110
retlw 0x20                  ; blank      111
retlw 0x28                  ; (          112
retlw 0x32                  ; 2          113
retlw 0x2D                  ; -          114
retlw 0x32                  ; 2          115
retlw 0x34                  ; 4          116
retlw 0x30                  ; 0          117
retlw 0x29                  ; )          118
retlw 0x3A                  ; :          119
retlw 0x20                  ; blank      120 - Message21 end
nop ; spare location                     121
```

```
nop ; spare location                    122
nop ; spare location                    123
retlw 0x4C                ; L           124 - Message22 start
retlw 0x69                ; i           125
retlw 0x6E                ; n           126
retlw 0x65                ; e           127
retlw 0x20                ; blank       128
retlw 0x69                ; i           129
retlw 0x6E                ; n           130
retlw 0x20                ; blank       131
retlw 0x75                ; u           132
retlw 0x73                ; s           133
retlw 0x65                ; e           134 - Message22 end
retlw 0x4E                ; N           135 - Message23 start
retlw 0x6F                ; o           136
retlw 0x20                ; blank       137
retlw 0x64                ; d           138
retlw 0x69                ; i           139
retlw 0x61                ; a           140
retlw 0x6C                ; l           141
retlw 0x20                ; blank       142
retlw 0x74                ; t           143
retlw 0x6F                ; o           144
retlw 0x6E                ; n           145
retlw 0x65                ; e           146 - Message23 end
retlw 0x44                ; D           147 - Message24 start
retlw 0x69                ; i           148
retlw 0x61                ; a           149
retlw 0x6C                ; l           150
retlw 0x6C                ; l           151
retlw 0x69                ; i           152
retlw 0x6E                ; n           153
retlw 0x67                ; g           154
retlw 0x20                ; blank       155
retlw 0x2E                ; .           156
retlw 0x2E                ; .           157
retlw 0x2E                ; .           158 - Message24 end
retlw 0x42                ; B           159 - Message25 start
retlw 0x75                ; u           160
retlw 0x73                ; s           161
retlw 0x79                ; y           162
retlw 0x20                ; blank       163
retlw 0x2E                ; .           164
retlw 0x2E                ; .           165
retlw 0x2E                ; .           166 - Message25 end
retlw 0x44                ; D           167 - Message26 start
retlw 0x69                ; i           168
retlw 0x61                ; a           169
retlw 0x6C                ; l           170
retlw 0x6C                ; l           171
retlw 0x69                ; i           172
retlw 0x6E                ; n           173
retlw 0x67                ; g           174
retlw 0x20                ; blank       175
retlw 0x74                ; t           176
retlw 0x72                ; r           177
retlw 0x79                ; y           178
```

```
retlw 0x20                  ; blank       179
retlw 0x23                  ; #           180 - Message26 end
retlw 0x52                  ; R           181 - Message27 start
retlw 0x69                  ; i           182
retlw 0x6E                  ; n           183
retlw 0x67                  ; g           184
retlw 0x69                  ; i           185
retlw 0x6E                  ; n           186
retlw 0x67                  ; g           187
retlw 0x20                  ; blank       188
retlw 0x2E                  ; .           189
retlw 0x2E                  ; .           190
retlw 0x2E                  ; .           191 - Message27 end
retlw 0x41                  ; A           192 - Message28 start
retlw 0x6C                  ; l           193
retlw 0x6C                  ; l           194
retlw 0x20                  ; blank       195
retlw 0x66                  ; f           196
retlw 0x69                  ; i           197
retlw 0x6E                  ; n           198
retlw 0x69                  ; i           199
retlw 0x73                  ; s           200
retlw 0x68                  ; h           201
retlw 0x65                  ; e           202
retlw 0x64                  ; d           203 - Message28 end
retlw 0x50                  ; P           204 - Message29 start
retlw 0x72                  ; r           205
retlw 0x6F                  ; o           206
retlw 0x67                  ; g           207
retlw 0x72                  ; r           208
retlw 0x65                  ; e           209
retlw 0x73                  ; s           210
retlw 0x73                  ; s           211
retlw 0x20                  ; blank       212
retlw 0x20                  ; blank       213
retlw 0x47                  ; G           214
retlw 0x6F                  ; o           215
retlw 0x6F                  ; o           216
retlw 0x64                  ; d           217
retlw 0x3A                  ; :           218 - Message29 end
retlw 0x42                  ; B           219 - Message30 start
retlw 0x75                  ; u           220
retlw 0x73                  ; s           221
retlw 0x79                  ; y           222
retlw 0x3A                  ; :           223
retlw 0x20                  ; blank       224
retlw 0x20                  ; blank       225
retlw 0x20                  ; blank       226
retlw 0x20                  ; blank       227
retlw 0x20                  ; blank       228
retlw 0x52                  ; R           229
retlw 0x61                  ; a           230
retlw 0x6E                  ; n           231
retlw 0x67                  ; g           232
retlw 0x3A                  ; :           233 - Message30 end
retlw 0x54                  ; T           234 - Message31 start
retlw 0x65                  ; e           235
```

```
retlw 0x6C                  ; l           236
retlw 0x20                  ; blank       237
retlw 0x73                  ; s           238
retlw 0x79                  ; y           239
retlw 0x73                  ; s           240
retlw 0x74                  ; t           241
retlw 0x65                  ; e           242
retlw 0x6D                  ; m           243
retlw 0x20                  ; blank       244
retlw 0x65                  ; e           245
retlw 0x72                  ; r           246
retlw 0x72                  ; r           247
retlw 0x6F                  ; o           248
retlw 0x72                  ; r           249 - Message31 end

END                         ; end assembly
```