A homebrew switching network (PBX) for four ATMs which share external telephone lines

Circuit Description

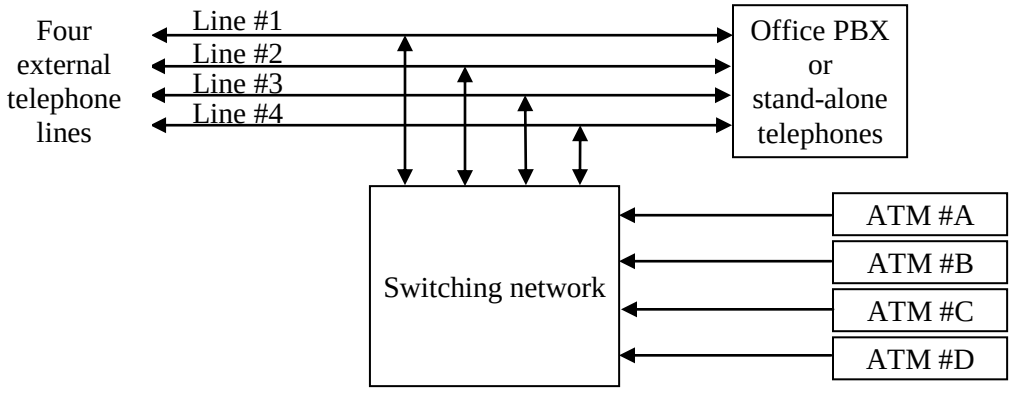
Design specifications

Consider a building complex that has four automated teller machines (ATMs), each of which has a dedicated external telephone line used solely for outgoing calls. When a customer enters his bank card to withdraw cash, the ATM makes a telephone call to its monitoring station for authorization. From time-to-time, each ATM also makes a telephone call to the monitoring station to tell the monitor that it is still alive (the so-called "heartbeat").

Assume that the building complex also has an office, which has several stand-alone telephones or a PBX ("private branch exchange"). Each stand-alone telephone has its own dedicated external telephone line or, alternatively, the PBX is serviced by several external telephone lines.

If the ATMs experience their heaviest use on evenings and weekends, when the office is closed, this configuration is wasteful. An external telephone line servicing an ATM has the same monthly tariff as the external telephone lines used by the office.

The design objective is to replace this configuration with a switching network whereby the ATMs can share the external telephone lines used by the office's telephones or PBX. This will allow the external telephone lines dedicated to the ATMs to be removed. We will assume that the switch should service up to four ATMs and will have available up to four external telephone lines. The following block diagram shows how the switching network will be connected.



Here, and elsewhere below, the external telephone are labeled numerically, as Line #1, …, Line #4. The ATMs are labeled alphabetically, as ATM #A, …, ATM #D. The external lines from the switching network ("external" meaning that they connect to the outside world) are connected in parallel with the external telephone lines which service the office PBX. We will see below that the switching network has a means to detect whether these external lines are in use. If an external line is not being used by the PBX, then the switching network will permit one of the ATMs to make its outgoing call on the line. If an external line is in use, then the switching network will examine the other external lines to find one which is free. Each ATM is an input (only) to the switching network. The switching network polls the ATMs to determine if and when one is starting to make an outgoing call. If so, the switching network identifies an external line which is not in use, and connects the ATM to that line.

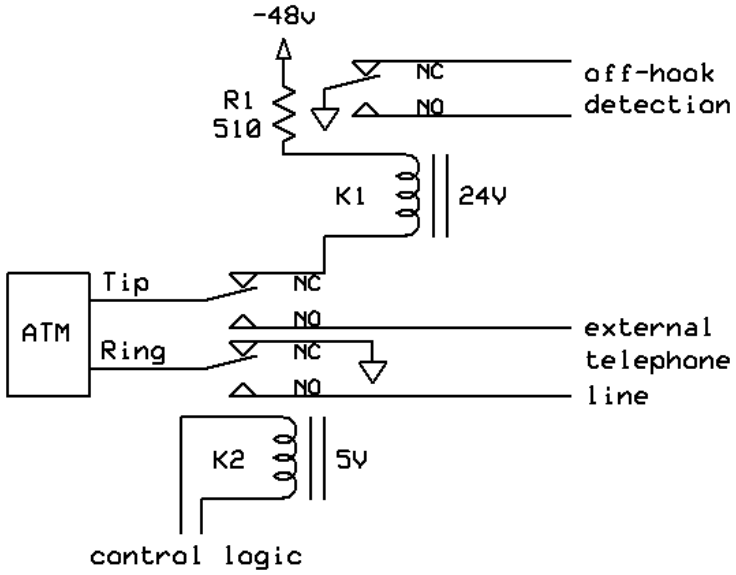
The switching network is not a full-function PBX *per se*. It merely switches lines. It does not generate dial tones or busy signals, nor does it need to. The ATMs already have the ability to control their communication on an external line – the switching network merely figures out which external line will suit. Furthermore, the switching network operates in one direction only, directing calls made by the ATMs. However, the switching network must provide the ATMs with the same standard -48V dc supply they would "see" if they were connected directly to the external telephone lines.

There are three schematic diagrams. Schematic #1 shows the logic circuits and the circuits that detect the status of the ATMs and the external telephone lines. Schematic #2 shows the array of switching relays. Schematic #3 shows the two dc power supplies, one at -48V (which is used in the telephony) and another at +5V (which is used to power the logic circuits and relays). The circuits in Schematic #1 are controlled by a PIC16F872 microcontroller (the component labeled U4). The circuits in Schematic #2 are controlled by a second PIC16F872 microcontroller (U7). The two microcontrollers communicate with each other using a four bit data bus and a fully-handshaken protocol. The device is constructed on a single printed circuit board (PCB) about ten inches high and eight inches wide.

Processing the inputs from the four ATMs

The internal, or local, lines from the four ATMs connect to the circuit through an 8-screw connector (J1), with one pair of screws for each ATM. The two lines for each ATM are identified as "Tip" (red wire) and "Ring" (green wire) in the customary way. Although an ATM should not be polarity-sensitive, it is good practice to hook them up as labeled.

As I have said, the four ATMs are labeled "ATM #A" through "ATM #D". The same suffixes "A" through "D" are also used to identify components which process signals from the ATM with the given suffix. The following figure shows the essential part of the circuit which handles the signal from one of the ATMs. This circuit is the same for each ATM.



The Tip and Ring wires of each ATM are connected to the two common terminals of the two poles of a double-pole double-throw relay (K2). When the ATM is not in use, the contacts of relay K2 are in their normally-closed (NC) state. In that state, the Ring wire is grounded and the Tip wire is connected in such a way that the voltage on the Tip wire is -48V. When the ATM is in use, relay K2 will be powered (+5V across its coil will close it). When K2 closes, both the Tip and Ring lines of the ATM will be

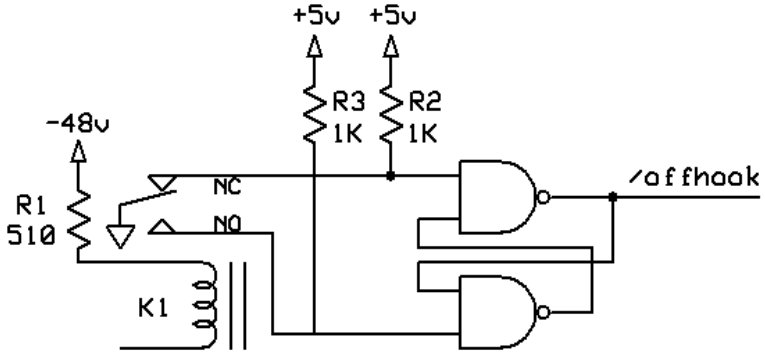
disconnected from the -48V supply circuit and will be connected through to an external telephone line. At no time will there be any galvanic connection between an external telephone line and the device's -48V supply.

Let me describe how the -48V supply is delivered to an ATM's Tip wire. When an ATM is not in use (described as being "on-hook", in the manner of the handset of an old rotary telephone resting in its cradle), its relay K2 will be open and there will be a series circuit from the -48V dc power supply to ground. The series circuit consists of a 510Ω resistor (R1), the coil of another relay K1 (+24V across the coil of K1 will close it) and the Tip-Ring resistance of the ATM. When the ATM is on-hook, the resistance between its Tip and Ring leads will be very high, so that no current will flow through K1's coil. When the ATM goes "off-hook" in preparation for making an outgoing telephone call, the dc resistance between its Tip and Ring lines will drop to about 150Ω. The dc voltage between the Tip and Ring lines when the ATM goes off-hook should be between 4V and 12V. I have chosen relay K1 and resistor R1 to be consistent with a 5.4V voltage drop across the 150Ω Tip-Ring resistance.

K1 is a 24V relay. 24V relays are manufactured with certain standard coil resistances. The standard coil resistance which is best for our purposes is 660Ω. Omron makes an SPDT 24V 660Ω resistance relay which Digikey sells as its part number Z1010-ND. I have used this relay for K1. Ohm's Law can be used to calculate the current which will flow through K1's coil when 24V is applied: I = V / R = 24V / 660Ω = 36mA. When this current flows through the series circuit, the voltage between the Tip and Ring lines of the ATM will be equal to V = IR = 36mA × 150Ω = 5.4V, which is within the 4V to 12V range we want. The voltage drop which remains over R1 can now be calculated as: V_{R1} = 48V – 24V – 5.4V = 18.6V. The resistance which makes this happen is equal to R = V / I = 18.6V / 36mA = 517Ω. A standard value resistance of 510Ω was used for R1.

There is a reason I chose to use a 24V relay for K1, as opposed to a 12V or 5V relay. So far, we have been looking only at the dc voltage between the Tip and Ring lines. However, once the ATM starts to communicate with its monitoring station, the same pair of wires will carry an audio signal. In the case of an ATM, the audio signal is a series of pulses at audio frequencies of several hundred Hz (called "DTMF" tones). The audio signal should have an amplitude of less than one volt, but the amplitude can on occasion be much larger. We want to ensure that the voltage over K1's coil is big enough that there is no possibility that the instantaneous voltage of the audio signal, when added to the dc voltage applied between the Tip and Ring lines, will cause the voltage over the coil to decrease to a level at which relay K1 will open.

In any event, when an ATM goes off-hook, the 48V drop will be shared among R1, the coil of relay K1 and the Tip-Ring resistance of the ATM in such a way that 36mA of current will flow, and relay K1 will close. K1 is a single-pole double-throw (SPDT) relay. Its contacts control the input to the off-hook detection circuit, which is shown in the following figure.



Relay K1's two non-common contacts are connected to a flip-flop constructed from two of the gates of a 74HC00 quad two-input NAND chip (U5 for ATMs #A and #B and U6 for ATMs #C and #D). K1 is wired so that its normally-closed contact is grounded. When the ATM is on-hook, the upper input line of the upper NAND gate will be grounded, so the output from the upper NAND gate will be high. This output is fed back as one of the two input lines of the lower NAND gate in the pair. The other input line of the lower NAND gate will be pulled high through 1KΩ resistor R3. Since both inputs of the lower NAND gate are high, the output of the lower NAND gate will be low. This is a stable configuration – both inputs of the upper NAND gate are low and both inputs of the lower NAND gate are high.

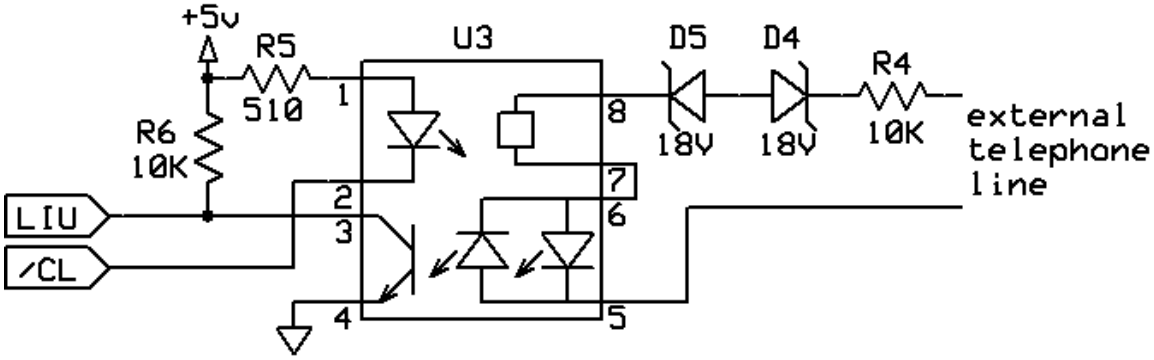
When the ATM goes off-hook, both NAND gates will change state and enter a new stable configuration. In the new configuration, the output of the upper NAND gate will be low and the output of the lower NAND gate will be high.

It is the output of the upper NAND gate (labeled "/offhook" in the figure), which is monitored by microcontroller U4. It is an inverted signal. This line is high when the ATM is on-hook and goes low when the ATM goes off-hook.

The off-hook states of the four ATMs are connected directly to four of the I/O pins of the microcontroller. ATM #A is connected to U4's RC4 line (pin 15), ATM #B is connected to U4's RC5 line (pin 16), ATM #C is connected to U4's RB4 line (pin 25) and ATM #D is connected to U4's RB6 line (pin 27). At any time, the microcontroller (U4) can read these lines to determine the status of the ATMs. If a line is high, the corresponding ATM is on-hook. If the line is low, the ATM is off-hook. In fact, polling these lines is the first part of U4's procedure for managing the ATMs. When U4 determines that an ATM has gone off-hook, it knows that the ATM wants to call in to the monitoring station.

Monitoring the external telephone lines

The first thing that U4 must do is determine which external telephone lines are available for use. The circuit which U4 uses for this purpose is shown here. There is one such circuit for each external line and all four circuits are identical.



A Clare TS117 multifunction telecom switch (U3) is the principal component in the external line detection circuit. The TS117 chip has two "sides" which are opto-coupled with each other and provide 3750 volts of galvanic isolation between the microcontroller side (the left-hand side of U3 as shown in the figure) and the external telephone line (the right-hand side of the component).

When the microcontroller (U4) wants to determine if an external line is available for use, it pulls the /CL line low ("CL" stands for CheckLine). The /CL line is connected directly to one of the I/O pins of microcontroller U4. U4 can activate the TS117 for each external line separately. The /CL control for Line #1 is connected to U4's RB7 line (pin 28), the /CL control for Line #2 is connected to U4's RB5 line

(pin 26), the /CL control for Line #3 is connected to U4's RB3 line (pin 24) and the /CL control for Line #4 is connected to U4's RB2 line (pin 23).

When /CL goes low, the infrared LED inside U3 between pins 1 and 2 will be forward-biased and will glow. The current flowing through this LED can be estimated from the datasheet for U3, which states that the forward voltage drop over this LED is about 1.2V. This will leave 3.8V over 510Ω resistor R5, which will allow current flow equal to I = V / R = 3.8V / 510Ω = 7.5mA. There are a couple of competing demands on the magnitude of this current. On the one hand, we do not want it to be too great, since the microcontroller must provide the power. It happens that U4's I/O pins can source and sink 25mA, so one pin can comfortably sink the 7.5mA. On the other hand, we want to "turn on" the TS117 as quickly as possible. The greater the current through the pin1-pin2 LED, the sooner U3 will produce a valid output signal. Two factors determine the intensity of the illumination of the pin1-pin2 LED: the amount of current flowing (more is better) and the length of time that one waits for the "bulb" to warm up (longer is better). The datasheet for U3 gives charts of these factors. The hardware was designed, as described above, so that 7.5mA would flow. The software, as described below, was written so that U4 waits 10ms after asserting /CL low before reading the LIU lines. This is a fairly conservative selection.

The glow from the pin1-pin2 LED will close the relay whose coil is connected to pins 7 and 8. This will allow current to flow (or not) through two more infrared LEDs inside U3, between pins 5 and 6. These two LEDs are wired in parallel, but in opposite directions, so that either one or the other will conduct. It does not matter, then, which polarity is used for the Tip and Ring lines of the external telephone line. Under some circumstances, current will flow through the two pin5-pin6 LEDs. In other circumstances, no current will flow. Whether current flows or not depends on the voltage between the Tip and Ring lines.

Let us examine first the case in which the external telephone line is not being used by any shared device. In that case, the voltage drop between the Tip and Ring lines will be approximately 48V. This exceeds the breakdown voltage of the 18V zener diodes D4 and D52. D4 and D5 are wired back-to-back, so that one of them will conduct in its zener region (with a voltage drop of 18V) and the other will operate in its forward-conduction region (with a voltage drop of about 0.7V). Therefore, the combined voltage drop over D4 and D5 will be 18.7V. U3's datasheet states that the voltage drop over the two pin5-pin6 LEDs will be about 1.2V, bringing the total series voltage drop to 19.9V. So long as the voltage drop between the Tip and Ring lines in greater than 19.9V, current will flow through 10KΩ resistor R4. If the voltage drop between the Tip and Ring lines is 48V, then the current flowing through R4 will be equal to I = V / R = (48V – 19.9V) / 10KΩ = 2.8mA. This is close to the typical current which U3's datasheet states is the current at which the glow from the pin5-pin6 LEDs will turn on the NPN transistor inside U3, connected between pins 3 and 4.

The magnitude of this current – 2.8mA – is, as is usually the case, subject to competing demands. We want it to be great enough to close the relay. But, we also want to keep it as small as possible because it is a load on Ma Bell's system. The batteries, or equivalent, in the Central Office ("C.O.") provide the -48V on-hook voltage and any current derived from it. In fact, it is the current drawn down the line which the C.O. monitors to determine if a telephone has gone off-hook. I stated above that a standard telephone has an off-hook resistance of about 150Ω and will, therefore, draw about 320mA. Ma Bell has a threshold current, above which it decides that the telephone on the line has gone off-hook. That threshold is probably in the range of 50 – 60mA, but can be lower. If only one TS117 is testing the line, the 2.8mA it uses is well below the threshold. But the designer must anticipate that there may be other devices which are also monitoring the same line. Too many monitors, each drawing a small or modest amount of current, can trick the C.O. in the mistaken decision that a telephone has gone off-hook.

Nor is the 50 – 60mA off-hook threshold the only current threshold your local Ma Bell may be using. Your local C.O. may also monitor the average dc current to determine if current is inadvertently being

drawn due to equipment failure or inappropriately drawn to power some non-telephone equipment. It is best, therefore, to draw as little current as possible and to test the line only when necessary. The alert reader will have noticed that the /CL line could be tied low or, alternatively, the /CL lines of all four U3 chips could be wired in parallel and controlled by just one of U4's I/O pins. That would be bad practice.

Now, let us continue to trace through what happens inside U3. When current flows through the pin5-pin6 LEDs, their illumination (or, more precisely, the illumination of whichever of the two LEDs is forward biased) will turn on the pin3-pin4 NPN transistor. The voltage drop between its collector (pin 3) and its emitter (pin 4) will be quite low, and the external 10KΩ resistor R6 will drop almost all of the applied voltage. This will be accompanied by the flow of current through R6 equal to I = V / R = 5V / 10KΩ = 0.5mA.

Pin 3 is the output line from U3. It is labeled as the LIU ("line-in-use") signal. The four output lines of the four U3 chips are connected directly to I/O pins of the microcontroller. LIU #1 is connected to U4's RC2 line (pin 13), LIU #2 is connected to U4's RC3 line (pin 14), LIU #3 is connected to U4's RC1 line (pin 12) and LIU #4 is connected to U4's RC0 line (pin 11). When U4 reads one of the LIU lines low, it knows that the voltage between the Tip and Ring lines is greater than about 20V, that is, that no other device or telephone is using the external line.

Let me describe quickly the opposite case, when the external line is in use when U3 tests it. The voltage between the Tip and Ring lines will be between 4V and 12V if and when the external line is being used by some other device. This voltage drop is less than the breakdown voltage of the zener diodes D4 and D5. No current will flow through them. The pin5-pin6 LED's will not glow and the pin3-pin4 NPN transistor will be cut off. When this transistor is cut off, its collector is free to float to whatever voltage suits the needs of the external components. In this case, resistor R6 will pull U3's pin 3 high. When U4 reads that the LIU line is high, it knows that the voltage between the Tip and Ring lines is less than about 20V, that is, that the external telephone line is in use by some other device.

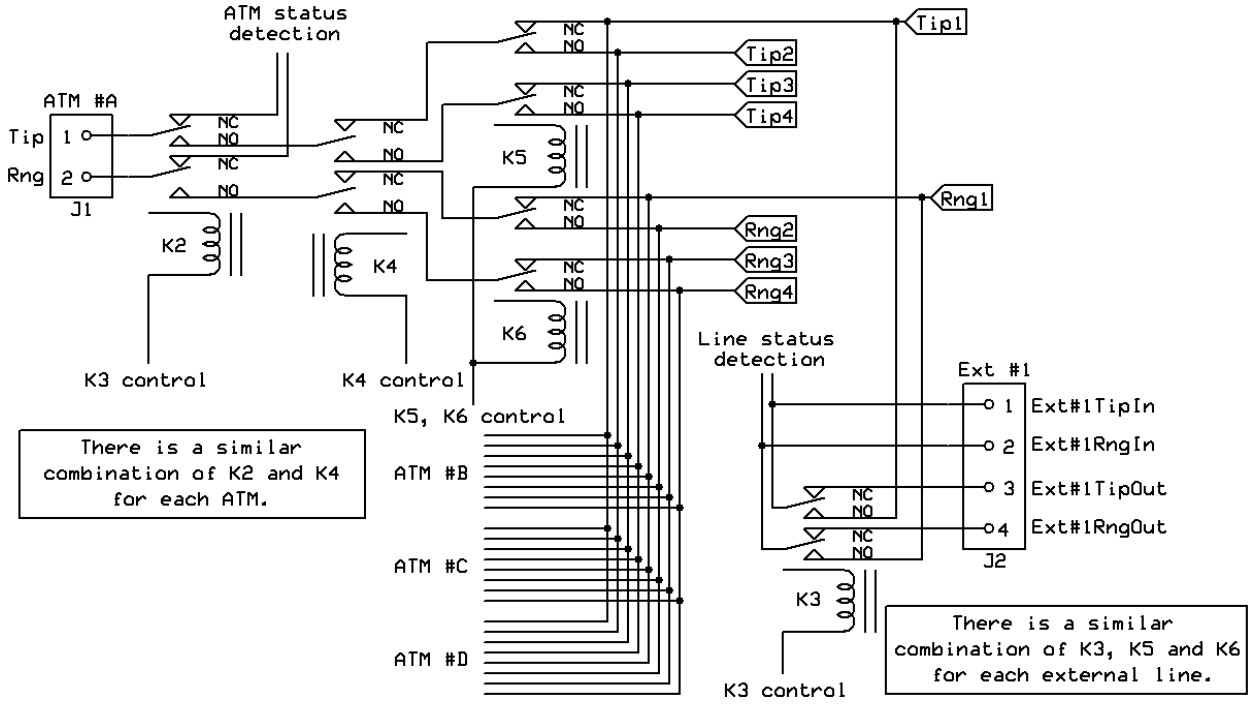
Microprocessor U4 polls the ATMs. When an ATM goes off-hook in preparation for making an outgoing call, U4's program runs through a procedure to find an available external line. The program tests the external lines one-by-one until it finds one that is free. However, the procedure avoids loading Ma Bell's system unnecessarily by not testing lines which it knows are already in use, which are the lines to which this device has already connected other ATMs.

One may ask: what happens if there is no available line? This happens very infrequently, but it can happen. If it does, U4 simply does not make a connection. The ATMs have the ability to handle a complete outgoing telephone call. The first thing they do after going off-hook is to listen for a dial tone. If they do not get a dial tone, and they will not if U4 does not make a connection to an external line, the ATMs will hang up and try again. Or the ATM's customer will re-insert his card and try again.

I have said that it is very rare for an external line to be available. That is true even if the number of ATMs is greater than the number of external telephone lines. Several factors contribute to this, the most important of which is the fact that the duration of the communication between an ATM and its monitoring station is only a fraction of the time a customer spends in front of the machine. A typical communication will take about 30 seconds, including dialing time. Even if four ATMs were in constant use, and customers did their business quickly, the average wait would be 30 seconds / 4 = 7.5 seconds before the ATM which started its communication first would be finished.

The relay array

When an ATM goes off-hook and U4 has selected an external telephone line, U4 must then connect the ATM's two lines to the external lines. For this purpose, it uses a network of electromechanical relays. We will look at that network of relays now. In this device, the relays are turned on and off by another microcontroller, U7. Once U4 has decided which ATM is to be connected to which external line, it communicates that pair to U7, which handles the details. We will look at U7 and the communication protocol in a later section. The relay array is shown in the following sub-schematic diagram.



In order to connect an ATM to an external telephone line, the microcontroller first closes the ATM's relay K2. This will connect the ATM's Tip and Ring lines to the common contacts of DPDT relay K4. All of the relays shown in this sub-schematic diagram have 5V coils. One end of each coil is connected to +5V (not shown) and the other end (the control line) will be pulled low when the relay is to be closed.

The three relays – K4, K5 and K6 – constitute a 1:4 switch by which the pair of lines from the ATM is connected to one of four output pairs of lines. The coils of K5 and K6 are wired in parallel so the two relays operate in synch, with K5 selecting one of the four output Tip lines and K6 selecting the corresponding Ring line. When K4 is in its closed position (NC), relays K5 and K6 are used to select either Line #1 or Line #2. When K4 is in its open position (NO), relays K5 and K6 are used to select either Line #3 or Line #4. The following table summarizes the logic of the relay network.

Coil of relay K4	Coil of relays K5 and K6	External line selected
Closed	Closed	#1
Closed	Open	#2
Open	Closed	#3
Open	Open	#4

The contacts of relays K5 and K6 are not, in fact, connected directly to the external telephone lines. There is one more relay, K3, involved. Relay K3 is another DPDT relay. When its contacts are in their normally-closed position (as they will be when the device is not powered up, for example), the external lines are not connected to the relay network at all. Instead, the two "incoming" lines, labeled Ext#1TipIn and Ext#1RingIn connect straight through to two "outgoing" lines, labeled Ext#1TipOut and Ext#1RingOut. Each line has a screw connector. This allows this device, and others like it, to be easily hooked up in series. (This is for convenience only, and does not prevent multiple devices from being hooked up, directly and in parallel, to the external telephone lines.)

Note that the leads to the circuit which detects the status of the external telephone lines are connected to the upstream side of K3's contacts. The status of the external lines can be checked at any time, without regard to the setting of K3's contacts.

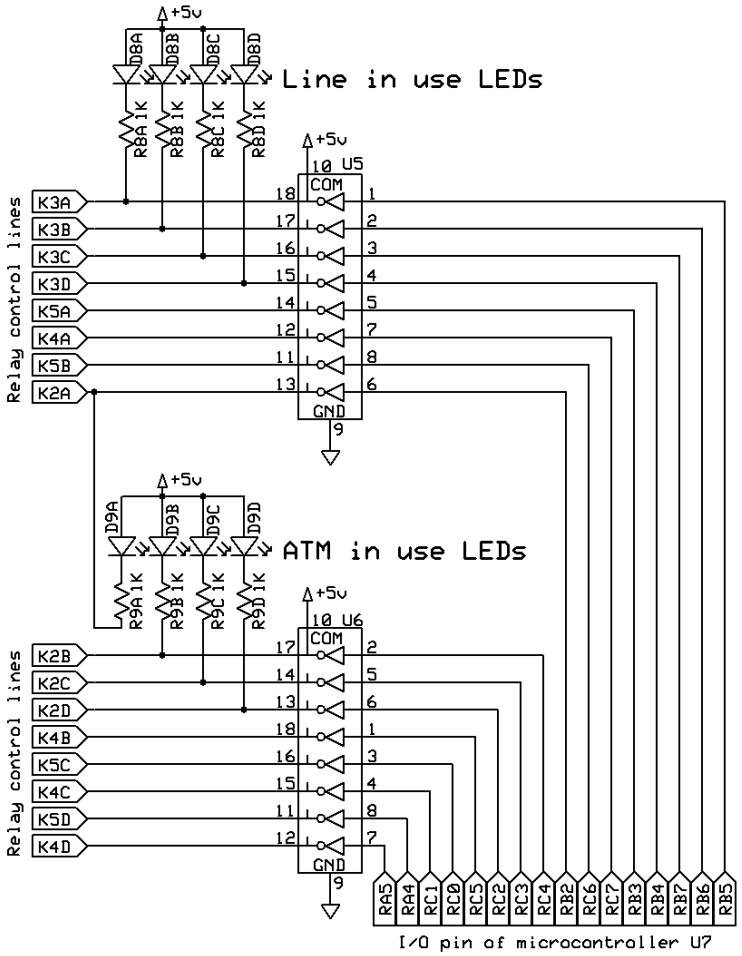
There are a couple of additional points to note:

- The sub-schematic diagram above shows only one set of relays. Dealing with four ATMs requires four sets of relays K2 and K4. Dealing with four external telephone lines requires four sets of relays K3, K5 and K6.
- An ATM is connected to an external line by separately connecting both of its Tip and Ring wires. There is no common ground between the external telephone lines and the logic circuits. And, there is no common ground between the ATMs' lines and the logic circuits.
- In a sense, relays K2 and K3 are isolation relays. An ATM is connected to the relay network if and only if its K2 relay is closed. Similarly, an external line is connected to the relay network if and only if its K3 relay is closed.
- ATM monitoring stations are very careful about the quality of the signals they receive from ATMs. "Clicks" on the line will be interpreted as a security breach (someone tapping in). They are avoided by making the complete connection, that is, setting all of the relays in the network, before closing relay K2 and allowing the ATM to commence dialing. Then, none of the relays is reset until after the communication has been completed.
- This raises the question: how does the device know when a communication has been completed? Once relay K2 is closed, the ATM status detection circuit (relay K1 and the NAND gates) is disconnected. But, microcontroller U4 can still monitor the external line using the TS117 chip. When the communication has been completed, the ATM monitoring station and the ATM will (presumably) both hang up. As a safety measure, U4's program includes a timing check – if a communication is not completed within five minutes, the connection is broken.

The driver transistors

One end of the coil of each of relays K2 through K6 is connected to the +5V power supply. The free ends, or control lines, are driven by two 18-DIP chips (U5 and U6) which each contain eight Darlington transistor drivers. The 16 drivers control the four K2 relays, the four K3 relays, the four K4 relays and the four K5-K6 relay pairs. Each of the drivers is connected to one of the I/O pins of microcontroller U7. Furthermore, the four drivers which control the K2 relays also power four LEDs which show the user the status of the four ATMs. And, the four drivers which control the K3 relays also power four LEDs which show the user the status of the four external lines, if they are under the control of this device.

The following extract from Schematic #2 shows the relay driver transistors.



The port labels along the left-hand side identify the relay controlled by each line. The port labels along the bottom identify the I/O pin of microcontroller U7. The transistors have open-collector outputs. When the input (base) is high, the transistor conducts and the output (collector) goes low. When the input is low, the transistor is cut-off and the voltage at the collector will float. In our circuit, a collector will float up to the +5V supply connected to the other end of the relay's coil, and no current will flow through the coil. Therefore, the relay control lines are "active high" in the sense that setting the voltage at the I/O pin high will close the relay and asserting the pin low will open the relay.

The chip used for U1 and U2 in our circuit is a ULN2803 (Digikey part # ULN2803APG-ND). It can handle 500mA of continuous output current and can dissipate up to 1W per Darlington (limited to 2.5W in total). This is far more than needed. The relays used draw a maximum current of 30mA or so when closed, so the current needed to close eight relays is only 240mA.

Visual indicators

The device has eight LEDs to display its status. Four of the LEDs are wired in parallel to the output lines from the transistor drivers which open and close the K2 relays. The output lines go low when the relays are to be closed, which cause the associated LED to glow. Closing the K2 relays connects an ATM to the relay network. So, when an ATM-in-use LED is illuminated, that ATM is being serviced. (Note that this is not quite the same thing as the ATM being off-hook. An ATM may be off-hook and still not be serviced.)

Similarly, the other four LEDs are wired in parallel to the output lines from the transistor drivers which open and close the K3 relays. These are the relays which connect the relay network to the external telephone lines. When one of these LEDs is illuminated, it indicates that the external line is being used by this device.

All eight LEDs have 1KΩ series resistors to limit the current flowing through the LEDs to approximately I = V / R = (5V – 2V) / 1K = 3mA.

Discharge diodes and bypass capacitors

The schematic diagrams above show the working components in the circuits described. But, there are ancillary components as well. Reference to the complete schematics (set out in the Appendices) will show that a small diode has been wired across the coil of each relay or, in the case of the K5-K6 pairs of relays, across the parallel coils of the pair. These diodes can be any small diode. They are wired against the polarity of the +5V supply voltage across the coils. During normal operations, therefore, these diodes do not conduct. They do not conduct when the contacts are open or when they are closed. They conduct only during a very short period of time when the relay makes the transition from being closed to being open. Here is why. A voltage drop is applied across the coil of a relay to close the contacts. The current which begins to flow through the coil builds up a magnetic field in and around the coil. It is this magnetic field which pulls the metal contacts into their closed position. The interesting thing happens when the relay is to be opened. To open the contacts, the applied voltage drop is removed. The magnetic field which had been built up consists of energy, and that energy must go somewhere. And, indeed, it does. To the best of its ability, the magnetic field will collapse at a rate which causes current to continue to flow at the same rate and in the same direction as it was flowing under the applied voltage drop. The flow of current represents energy draining away from the magnetic field. The current is really less of a "flow" and more of a "surge", because the magnetic field will collapse quite quickly. This surge of current will flow no matter what is connected to the coil. In practice, huge voltage spikes can develop as the current forces its way through the transistors and other components in the coil's circuit. Putting discharge diodes across the coil provides an easy path to handle this surge of current. It will flow through the diode and back into the coil, during which flow the resistance of the coil will sop up the energy and dissipate it as heat.

The complete schematic diagrams also show several small (1µF) capacitors between the +5V supply voltage and ground. In theory, these capacitors do not do anything special, being wired in parallel with the much bigger filter capacitors near the regulators. A review of the printed circuit board layout will show that these capacitors are located as near as possible to certain of the semiconductor chips. They provide a local reservoir of (electrical) charge in the immediate vicinity of the chips, which the chips can draw upon without having to wait for additional electrons to travel all the way from the power supply. In this manner, they reduce changes in voltage which would otherwise occur when the chips draw power in order to change state. They "bypass" rapid changes in voltage to ground, hence their name.

The power supplies

Before describing the microcontrollers, let me describe the power supply, which is shown in Schematic #3. The power supply delivers two voltages, which are produced by two different transformers. The -48V power supply has been designed to deliver 250mA at 48V. The +5V power supply, which powers the logic circuits and the relays, has been designed to deliver 1A at 5V.

The use of -48V in telephony is an artifact of telephone history. In the early days of telephony, signals were sent on a single wire with the earth used as the, uh, ground. It was observed that there was less corrosion of the copper wire if it was kept at a negative voltage with respect to the earth, rather than a

positive voltage. Furthermore, in those early days, 50V was the accepted threshold between voltages which were thought to be dangerous and those which were thought to be safe. 48V was selected to be just a bit less than this threshold.

120Vac is hooked up to the printed circuit board (PCB) via a recessed three-sided connector J3. A SPST toggle switch (S1) is mounted directly on the PCB. The incoming ac is delivered to the primary windings of both transformers. Both transformers have dual primary windings, and both primary windings in each transformer are wired in parallel.

Transformer TR1 is a Signal Transformers DPC-40-250 which has a 40Vrms center-tapped secondary winding rated at 250mA. The center-tap is ignored. The full 40Vrms is applied to the input terminals of a small bridge rectifier (BR1), which is rated at 1.5A and 600V. A 560µF capacitor (C7) provides a bit of filtering and a standard +12V voltage regulator (U8) provides the regulation. Note that a 36V zener diode (D10) is connected between the common terminal of the voltage regulator and "ground", so that the regulated voltage is the sum of 36V and 12V, or 48V. Another 560µF capacitor (C8) holds charge at the regulated voltage. Zener diode D10 is biased to operate in its zener region by a 1KΩ resistor (R9). The 12V voltage drop between pins 3 and 2 of U8 will cause current equal to 12mA to flow through R9. This current will also flow through D10, ensuring that it is always properly biased and the voltage drop over it is robust.

It is important to note that the "positive" side of the TR1 circuit is connected to the circuit ground and the "ground" side of the TR1 circuit actually constitutes the -48V supply.

Transformer TR2 is a Tamura PL10-16-130B dual 8Vrms transformer, with each sub-transformer rated at 625mA. The two secondary windings are connected in parallel, so this transformer can deliver 1.25A. A bridge rectifier (BR2) of the same type as used in the TR1 circuit provides rectification. This is followed by a common +5V positive voltage regulator (U8) with a 2200µF filter capacitor on both its input and output sides.

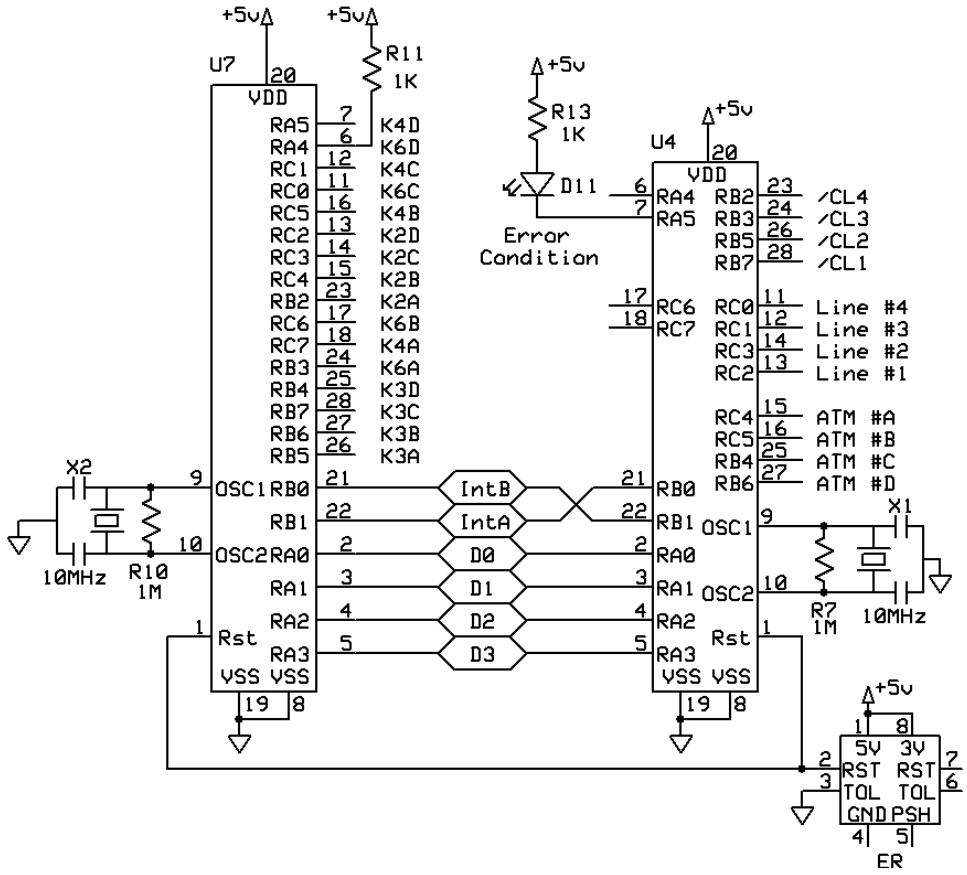
The "slave" microcontroller (U7)

The circuit uses two microcontrollers. In this section, I will describe U7, which controls the relay network through the transistor drivers. I have called this microcontroller a "slave" because its only task is to carry out a "make connection" or "break connection" instruction sent to it by the main microcontroller U4.

This microcontroller (in fact, both microcontrollers) is a PIC16F872 manufactured by Microchip Corporation. It has program memory of 2,064 14-bit words, 22 I/O lines, 128 eight-bit registers for program use and 64 eight-bit EEPROM storage bytes. We will use only a fraction of this tremendous capability.

As described above, 16 of U7's 22 I/O pins are used to control the 16 relays. These 16 lines are configured for output. U7 sets the voltages on these lines high or low depending on whether the associated relay is to be closed or open, respectively.

That leaves six I/O pins. All six are used to communicate with the main microcontroller. The following sub-schematic diagram shows the essential features of the hardware the two microcontrollers use to communicate. In the diagram, U7 is shown on the left and the main microcontroller (U4) is shown on the right. In the following descriptions, I will refer to the two microcontrollers as "U7" or "slave", on the one hand, or as "U4" or "main", on the other.



The communication is conducted using the six lines, IntA, IntB and D0 through D3. The lines D0 through D3 carry four bits of data, and constitute the "data bus". IntA and IntB are the two control lines. I have labeled the control lines with generic labels IntA and IntB, rather than IntReq (for interrupt request) and IntAck (for interrupt acknowledge). The latter usage is confusing, because one microcontroller's IntReq is the other's IntAck, and vice versa.

Before delving into the topic of communication, let me deal with a few administrative matters. Each microcontroller has its own external oscillator. Both oscillators have a frequency of 10MHz, so the clock period is 100ns long. Each instruction cycle of a PIC16F872 takes four clock cycles, so each instruction in the code takes 400ns (although a few instructions take twice as long.) A clean start-up when power is first applied is provided by a Dallas Econo-Reset chip (ER). The microcontrollers begin execution on a rising edge on pin 1. ER provides a short delay, during which the power supply stabilizes, before generating this low-to-high edge. As described above, U7 uses 16 I/O pins to control the relay network and U4 uses 12 I/O pins to do what it needs to do. I have attached an LED to U4's RA5 line (pin 7) which U4 can use to tell the user about an internal error or some other important event. Resistor R11 attached to U7's RA4 line (pin 6) is a pull-up resistor required because that particular I/O pin on a PIC16F872 is open-collector (or, rather, open drain). U4's RA4 line is not connected to anything, so it is configured for output, and no pull-up resistor is needed.

Each of these I/O lines is used for a single purpose. Therefore, they can be configured for input or output, as the case may be, when U4 and U7 begin execution. Their configurations do not change thereafter.

The same is true for the six lines used for communication. This is not generally the case but happens to be the case here because the communication is only one-way. When necessary, U4 initiates a

communication with U7 to tell the slave which ATM should be connected to or disconnected from which external telephone line. U7 never initiates a communication to U4. Accordingly, U4 configures the four lines of the data bus as outputs and U7 configures them as inputs. U4 controls the voltage on the IntA line, so it is an output of U4 and an input of U7. U7 controls the voltage on the IntB line, so it is an output of U7 and an input to U4. In the normal condition, when no communication is in progress, U4 holds the IntB line low and U7 holds the IntA line low.

The main microcontroller sends a nibble of information, consisting of four bits sent in parallel, to the slave using the following procedure.

1. To start a communication, U4 raises the voltage on the IntB line from zero to 5V. This is usually called a request for an interrupt or, simply, an "interrupt request".
2. When U7 realizes that this line has gone high, it acknowledges the request by raising the voltage on the IntA line from zero to 5V. This is usually called an "interrupt acknowledge". At this point, both microcontrollers have stopped whatever else they were doing and can concentrate on the communication.
3. When U4 realizes that U7 has acknowledged the interrupt, it proceeds to set the voltages on the four data lines to their appropriate values, 5V for a logic "1"bit and ground for a logic "0" bit. (Let me just note here that there would be one more step here in the usual case, when the two microcontrollers exchange data both ways. In that case, both microcontrollers would configure the four lines of the data bus into a high-impedance state, so that the voltage on the those lines would not be set by the microcontroller at either end. Then, when U4 recognized U7's interrupt acknowledgement, it would first reconfigure the four lines as outputs before setting the appropriate voltages. U4 would know that it could take control of the voltages on those lines without causing a conflict with voltages U7 was trying to impose. After all, by sending the interrupt acknowledgement, U7 acknowledged that it is ready to receive data.)
4. After having set the voltages on the data bus, U4 asserts the IntB control line low.
5. When U7 sees this transition, it knows that the voltages on the data bus are valid and can proceed to read them.
6. After having read the data, U7 asserts the IntA control line low.
7. Although it is not essential, it is good practice (and it is done here) for U4 to continue to monitor the IntA line even after it has asserted the IntB line low. If U4 sees the IntA line go low, it knows that U7 read the data. But, if the IntA line remains high, U4 should conclude that U7 may not have received the data.
8. At the end of the communication, the IntA and IntB lines are both low, as they were at the outset before the communication began. Because the whole procedure is symmetric, either microcontroller could initiate the next exchange of data. In our application, of course, only U4 needs to initiate a communication.

Note that the IntB line is connected to U7's RB0 line (pin 21). This is not by chance. The IntB line is the one U4 uses to start a communication. This is an important event. In a PIC16F872 microcontroller, the RB0 line can be configured as the trigger for an interrupt. When so configured, a low-to-high transition on the RB0 line can divert the flow of execution to memory address 0x0004 in the microcontroller's code. The diversion occurs automatically (from the programmer's point-of-view), and will commence within one or two instruction cycles after the voltage transition. The procedure which starts at address 0x0004 will be run when an interrupt occurs. This procedure is usually referred to as the "Interrupt Service Routine", or ISR, because it services interrupts when they occur.

U7 is configured and programmed so that an RB0 interrupt begins the communication process. The PIC16F872 is a versatile chip. RB0 interrupts are only one type of interrupt the chip can be configured to handle. Things can be arranged so that a change in any one of certain lines in portB triggers an interrupt. Or, things can be arranged so that an interrupt is triggered at certain times, as directed by either internal or external timers.

U7 is configured so that an internal timer (the Timer0 module) triggers an interrupt every 25ms or so. Like any interrupt trigger, program control passes to address 0x0004 when Timer0 generates its signal. The first thing the ISR must do, therefore, is determine whether it started up because of an RB0 interrupt sent by U4 or because of a Timer0 interrupt sent by the Timer0 module. If the interrupt was started by an RB0 interrupt, U7 starts the communication process.

What, then, is the purpose of the Timer0 interrupt? The Timer0 interrupt is used as a safety mechanism. It prevents U7 from getting "locked up" in a failed communication. At certain points in the communication process, U7 waits for a signal from U4. If, for some reason, that signal was not sent or was missed when it was sent, U7 could conceivably wait forever. A communication cycle should be complete within a few dozen microseconds at most. 25 milliseconds is much, much longer than required for a communication cycle. If a communication cycle is not completed within 25ms, then the Timer0 interrupt gives U7 the opportunity to stop waiting and take corrective action.

In the event of a communication timeout, the corrective action which U7 takes is: (i) to wait 500ms and then (ii) to restart itself. The thinking behind this recovery action is as follows. If a communication timeout occurs, it is likely that the main microcontroller will also detect it. When there is a failure of communication, the main microcontroller can no longer be certain that U7 has carried out its most recent instruction. Given the uncertainty about the state of the network relays, the prudent course for the main microcontroller is to start over again with a clean slate. It does this by restarting itself. When U4 restarts itself, the voltages on its I/O pins may do unpredictable things. So that U7 does not become confused by these voltage transitions, U7 waits for one-half second before restarting itself.

One last item remains to be described: the format of the data in the communication. Each communication from U4 to U7 consists of two four-bit nibbles, sent one immediately after the other. In fact, U7's ISR is programmed to read two nibbles. It does not exit from the ISR until it has received both nibbles.

The first nibble is called a Command. It can have one of three values:

- b'0001' means the following nibble describes a connection to make,
- b'0010' means the following nibble describes a connection to break and
- b'0011' means disconnect all ATMs from all external telephone lines.

The second, or following, nibble is called the Data nibble. It has the format b'ffgg', where b'ff' has the decimal value 0, 1, 2 or 3, and identifies the external telephone line and b'gg' has the decimal value 0, 1, 2 or 3, and identifies the ATM. For the external lines, b'ff'=0 corresponds to Line #1, b'ff'=1 corresponds to Line #2, b'ff'=2 corresponds to Line #3 and b'ff'=3 corresponds to Line #4. For the ATMs, b'gg'=0 corresponds to ATM #A, b'gg'=1 corresponds to ATM #B, b'gg'=2 corresponds to ATM #C and b'gg'=3 corresponds to ATM #D.

For Commands of type 3, U7 receives the Data nibble but ignores its contents.

The Appendices attached include a listing of the program for U7. It is written in Microchip's assembly language. Microchip's complete development kit is available on the web as a free download.

Timing delays required for the relays

U7 operates many, many times faster than relays can open and close. It will take only a few microseconds for U7 to decode Commands sent by U4 and to figure out exactly which relays need to be opened or closed. A few more microseconds will suffice for U7 to set the voltages on the appropriate I/O lines and for the driver transistors to change state. But, it will take thousands of times longer for the relays' contacts to settle into their new positions. We need to make the slave microcontroller wait after sending new voltage levels to the relays.

The datasheets for the K2 through K6 relays state that it takes between one and six milliseconds for these relays to open or close. To be safe, the software has been written so that U7 pauses for ten milliseconds after sending an open or close command to a relay.

An overview of U4's program

U4 uses five registers to store information about the state of the system:

offhookreg – a four-bit register (LSB = ATM #A through MSB = ATM #D) in which "1" means the corresponding ATM is off-hook and "0" means the corresponding ATM is on-hook. Note that the bit stored is the complement of the voltages from the NAND chips which U4 reads as "hookstatus" bits;

servicereg – a four-bit register (LSB = ATM #A through MSB = ATM #D) in which "1" means that the corresponding ATM has been connected to an external telephone line, that is, is being serviced, and "0" means that the corresponding ATM has not been connected to an external line (even if it is off-hook);

LIUreg – a four-bit register (LSB = Line #1 through MSB = Line #4) in which "1" means that the corresponding external telephone line is being used by an ATM and "0" means that the corresponding external telephone line is not being used by an ATM (even if it is being used by some other device connected to the external telephone line);

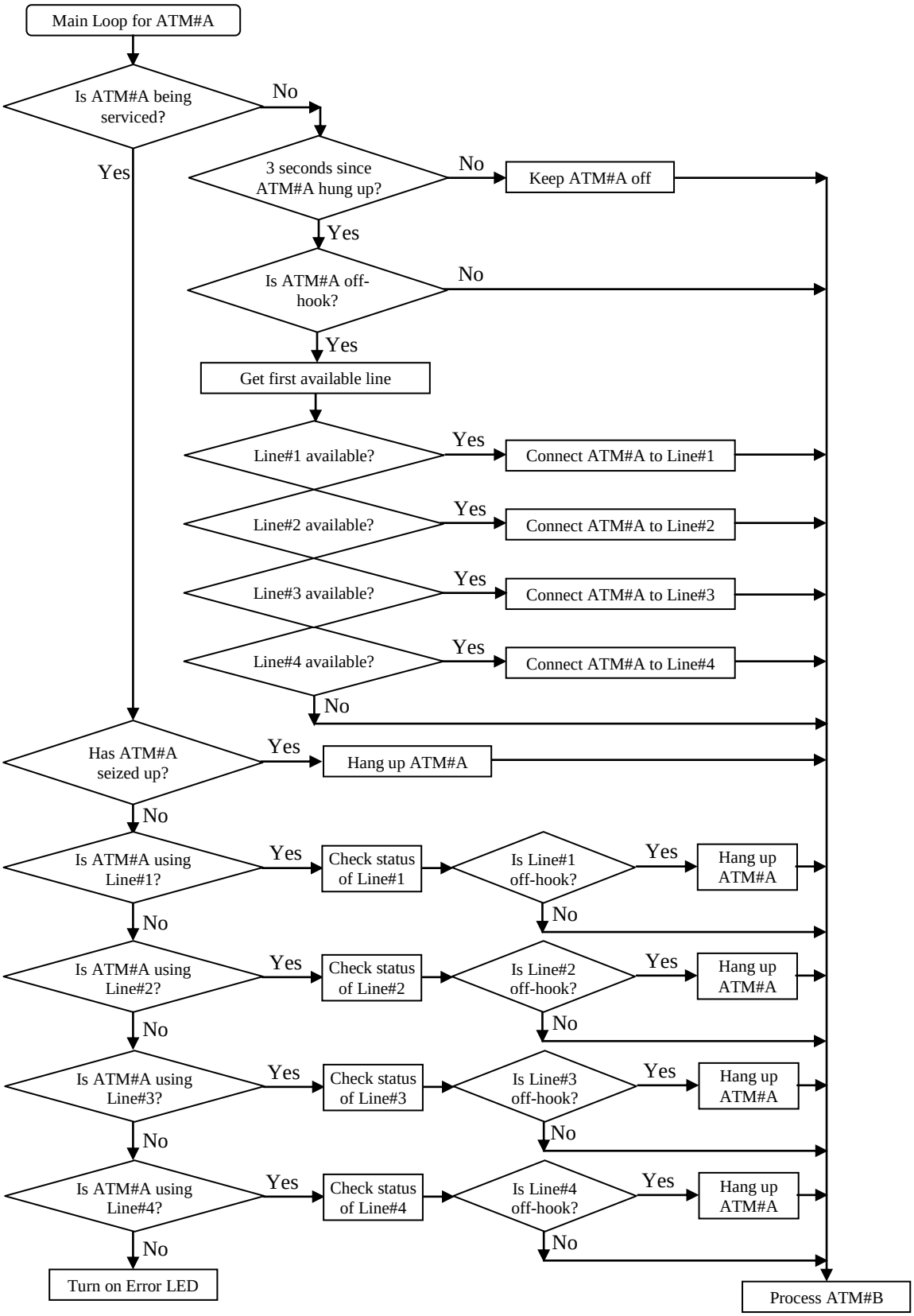
extlinereg – a four-bit register (LSB = Line #1 through MSB = Line #4) in which "1" means that the corresponding telephone line is in use, either by an ATM or by some other device and

ATMassign – an eight-bit register which holds four pairs of line assignments. The two low bits <0-1> record the number of the external line currently connected to ATM #A, if any. The values 0, 1, 2 and 3 correspond to the external lines #1, #2, #3 and #4, respectively. Bit pair <2-3> records the number of the external line currently connected to ATM #B, if any. Similarly, bit pair <4-5> records for ATM #C and bit pair <6-7> records for ATM #D.

The names which the program uses to identify U4's I/O pins are:

- CL1, CL2, CL3 and CL4 for the four CheckLine outputs,
- LIU1, LIU2, LIU3 and LIU4 for the four Line-in-use inputs,
- hookstatusA, hookstatusB, hookstatusC and hookstatusD for the off-hook status of the ATMs,
- D0, D1, D2 and D3 for the four bits of the data bus, and
- INT and ACK for the two lines which control communication with U7.

The following flowchart shows the procedure U4 uses to manage ATM #A. It uses a similar procedure to deal with each of the other three ATMs.


Main Loop for ATM#A
Is ATM#A being serviced?
No
Yes
3 seconds since ATM#A hung up?
No
Keep ATM#A off
Yes
Is ATM#A off-hook?
No
Yes
Get first available line
Line#1 available?
Yes
Connect ATM#A to Line#1
Line#2 available?
Yes
Connect ATM#A to Line#2
Line#3 available?
Yes
Connect ATM#A to Line#3
Line#4 available?
Yes
Connect ATM#A to Line#4
No
Has ATM#A seized up?
Yes
Hang up ATM#A
No
Is ATM#A using Line#1?
Yes
Check status of Line#1
Is Line#1 off-hook?
Yes
Hang up ATM#A
No
Is ATM#A using Line#2?
Yes
Check status of Line#2
Is Line#2 off-hook?
Yes
Hang up ATM#A
No
Is ATM#A using Line#3?
Yes
Check status of Line#3
Is Line#3 off-hook?
Yes
Hang up ATM#A
No
Is ATM#A using Line#4?
Yes
Check status of Line#4
Is Line#4 off-hook?
Yes
Hang up ATM#A
No
Turn on Error LED
Process ATM#B

The procedure is "linear" – no programming tricks or shortcuts are used in the procedure. The main program is a loop. It examines the four ATMs one-by-one, taking the appropriate action for each ATM, and then starts the loop again. The machine code executes much faster the ATMs and external telephone lines change state, so there is no need for tricky code.

Furthermore, neither the ATMs nor the external telephone lines are processed "in parallel". It would be possible, for example, to read the status of all external lines at the same time in order to determine their availability. This is not done. In order to minimize the loading placed on Ma Bell, external lines are checked individually and only when necessary.

Three applications of the Timer0 module

Three events in U4's program are timed. They are:

1. Communication with the slave microcontroller must time out if not completed within 25ms or so. This is necessary so that U4 does not wait indefinitely if a communication with U7 fails.
2. After the ATM monitoring station hangs up on an ATM which had been dispending cash, the program waits three seconds before checking whether the ATM has gone off-hook again. This explicit delay is necessary to give the ATM time to hang up. As I have said, U4's program is much faster than the ATMs'. Without an explicit delay, U4 could misinterpret an ATM's being off-hook as the start of a new outgoing call to the monitoring station when, in fact, the ATM had not completed the preceding communication.
3. Once U4 connects an ATM to an external telephone lines, it starts counting the duration of the telephone call. If the call has not been completed within five minutes, U4 breaks the connection.

U4 uses the chip's internal Timer0 module, and an Interrupt Service Routine, to manage all three timing cycles. Since U4 is the originator of all communications with U7, it never receives RB0-type interrupts. The ISR of U4 is triggered only by the Timer0 module. The Timer0 module is configured so that a Timer0 interrupt occurs every 26.2144ms.

An aside: The configuration of the Timer0 module is set as follows. The Timer0 prescalar is set to 256:1, so that register timer0 increments once every 256 instruction cycles. Since each instruction cycle is comprised of four clock cycles, timer0 will incremented every 256 x 4 x 100ns = 102.4μs. A Timer0 interrupt will be generated when register timer0 overflows. Therefore, if timer0 is left free-running, it will overflow once every 256 x 102.4μs = 26.2144ms. Incidentally, U7's Timer0 module is configured in the same way.

U4's ISR uses the following 13 registers, or pairs of registers, to look after the separate timed events:

- counterTO – for communication time-outs;
- counterAH and counter AL – register pair for ATM #A counting,
- counterAon – flag for ATM #A's five-minute line-seizure test,
- counterAoff – flag for ATM #A's three-second keep-off period,
- counterBH and counter BL – register pair for ATM #B counting,
- counterBon – flag for ATM #B's five-minute line-seizure test,
- counterBoff – flag for ATM #B's three-second keep-off period,

…

…

- counterDH and counter DL – register pair for ATM #D counting,
- counterDon – flag for ATM #D's five-minute line-seizure test and
- counterDoff – flag for ATM #D's three-second keep-off period.

One counter (counterTO) is use to detect communication time-outs. Each ATM has a 16-bit counter consisting of two registers (counterXH and counterXL) and two flags (counterXon and counterXoff). The suffixes "H" and "L" identify the counter registers as being the high-order byte or the low-order byte.

The ISR is coded so that counterTO is decremented once per execution. At the start of a communication with U7, counterTO is set to d'2'. Since timer0 is never reset, timer0 will have some value between 0 and 255 when the communication starts. It will take somewhere between 0ms and 26.2144ms before register timer0 overflows, the ISR executes and counterTO is decremented by one from d'2' to d'1'. The second decrement will occur exactly 26.2144ms later. If the loops in the communication procedure are still executing when counterTO decrements to zero, they will be stopped and a time-out declared. So, the communication will be timed-out somewhere between 26.2144ms and twice that duration, which is satisfactory for our purposes.

Similarly, the ISR is coded so that the four ATM counter pairs are also decremented once per execution. As an example, let us consider ATM #B. (The other three ATMs are handled in the same way.) When ATM #B is first connected to an external telephone line, counterBH is set to d'45' and counterBL is set to zero. The decimal value of the 16-bit unsigned integer in the register pair is 256 x 45 = 11,520. The value will be reduced by one once per ISR execution. Each of the decrements will take exactly 26.2144ms, except for the first one, because the value of register timer0 is not know at the outset. So, the counter pair will reach zero somewhere between 11,519 and 11,520 Timer0 interrupts, or between 11,519 x 26.2144ms = 301.964s and 11,520 x 26.2144ms = 301.990s. This is close enough to five minutes (300s) for our purposes.

Similarly, when an ATM is disconnected from its external line, its counter pair is reset. To continue using ATM #B as the example, counterBH is set to zero and counterBL is set to d'115'. The ISR continues to decrement the pair once every Timer0 interrupt, and the counter pair will be decremented to zero, somewhere between 114 and 115 Timer0 interrupts. This is between 114 x 26.2144ms = 2988.44ms and 115 x 26.2144ms = 3014.66ms. This is close enough to three seconds (3000ms) for our purposes.

Note that the same pair of registers is used for both the five-minute line-seizure test and the three-second keep-off test. There is no conflict between the two uses – one timing event takes place while the ATM is connected to the external line and the other takes place after it has been disconnected. But, to distinguish between the two events, U4 uses flags. To continue using ATM #B as the example, counterBon is set high when ATM #B is first connected to an external line and counterBoff is set high when the connection is broken. U4's main program uses these two flags to determine whether an event is being timed and, if so, whether to examine the values of the counter pair for the corresponding ATM.

It should be noted that the ISR decrements the counters until they reach zero, but does not "decrement" them any further once they reach zero.

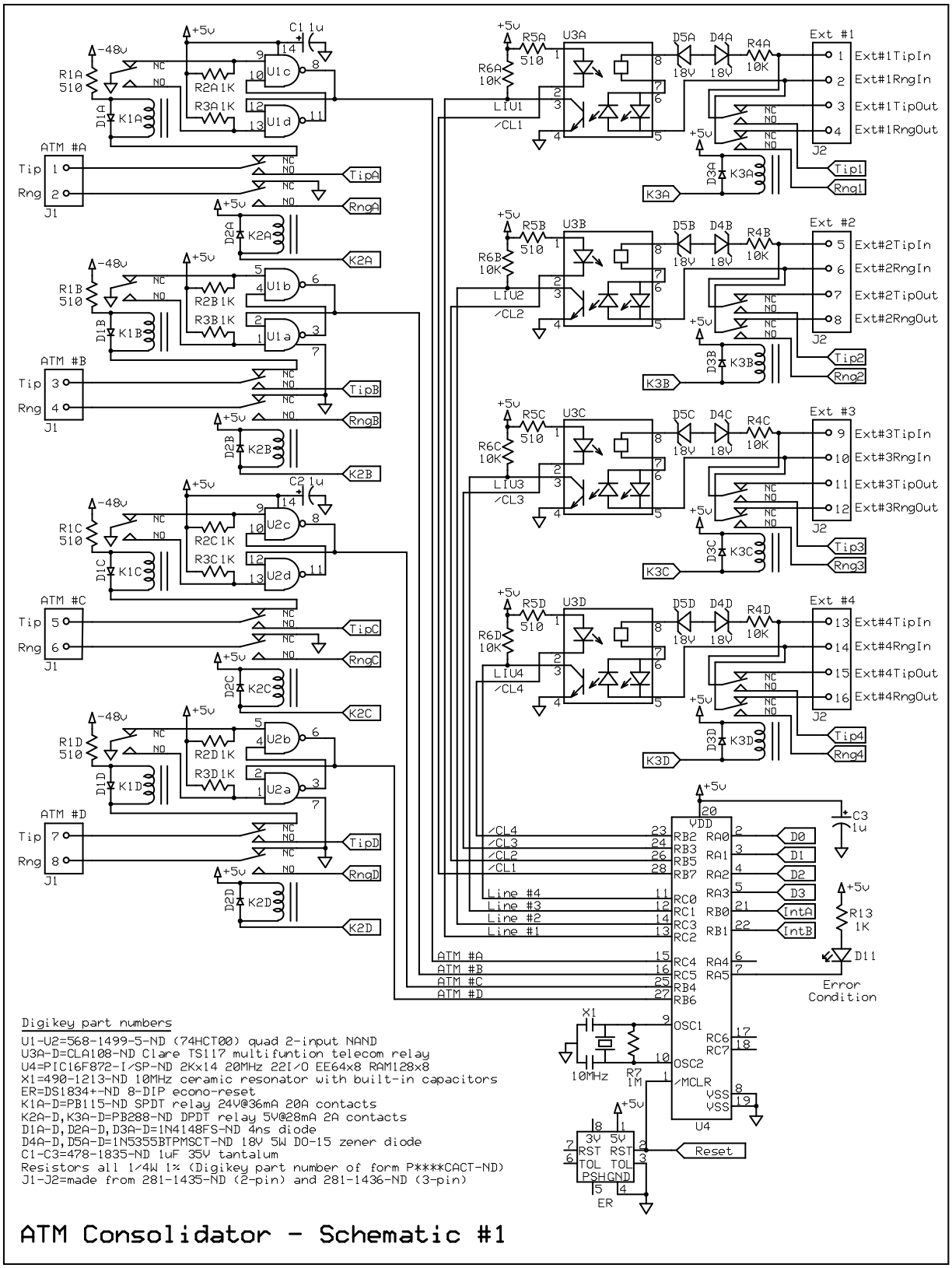
The attached Appendices contain the schematics and printed circuit board layouts for the device, and the assembly language listings of the two microcontroller programs.

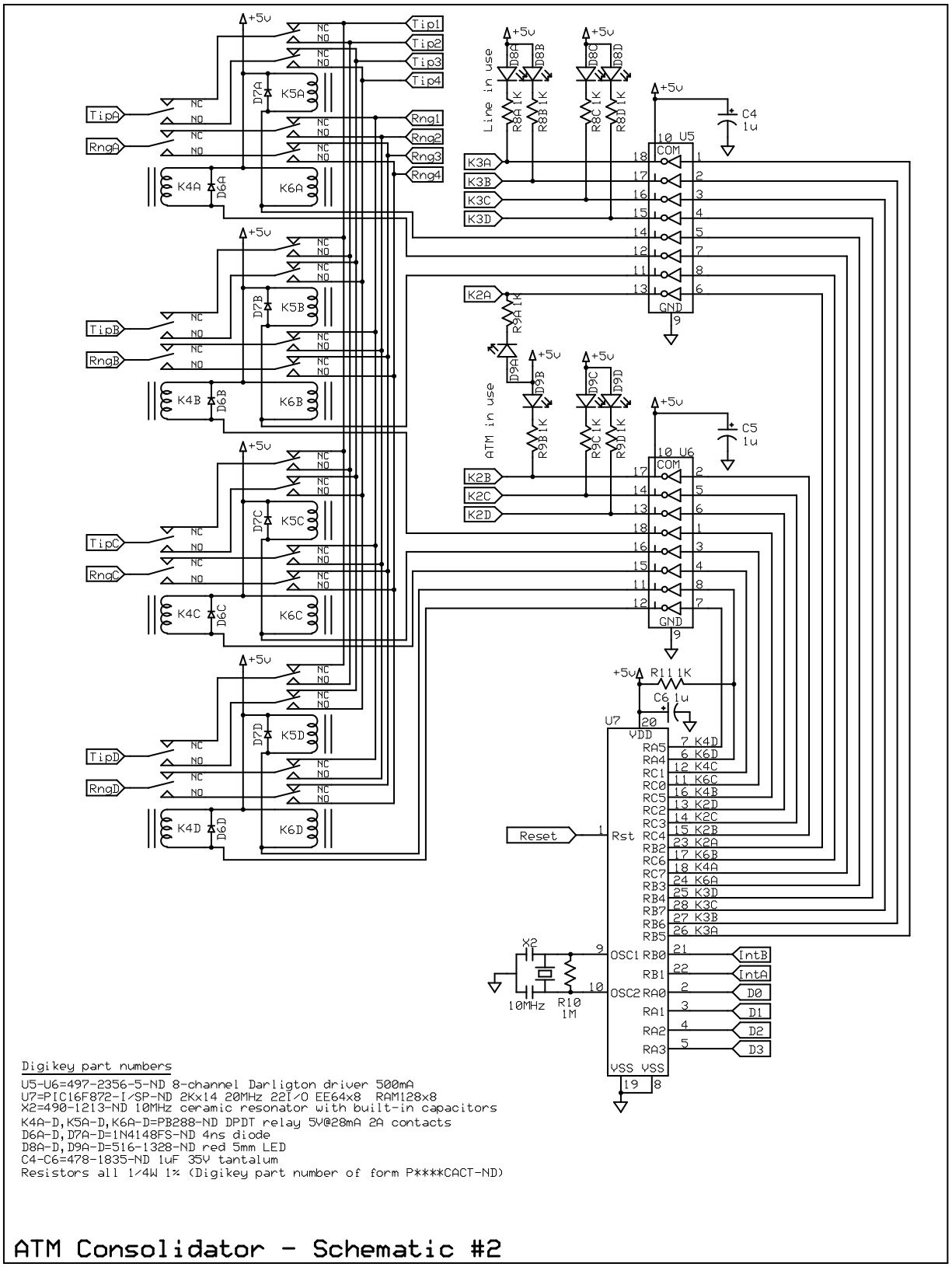
James Hawley
July 2012

An e-mail setting out errors and omissions would be appreciated.

Index of Appendices

The photograph shows two devices. Together, they connect five ATMs to four external telephone lines. Three of the external telephone lines are also used by a Nortel PBX. The fourth external line is also used by a fax machine and as a DSL internet line. Two of the ATMs are connected to the PCB at the top. The other three ATMs are connected to the PCB at the bottom.

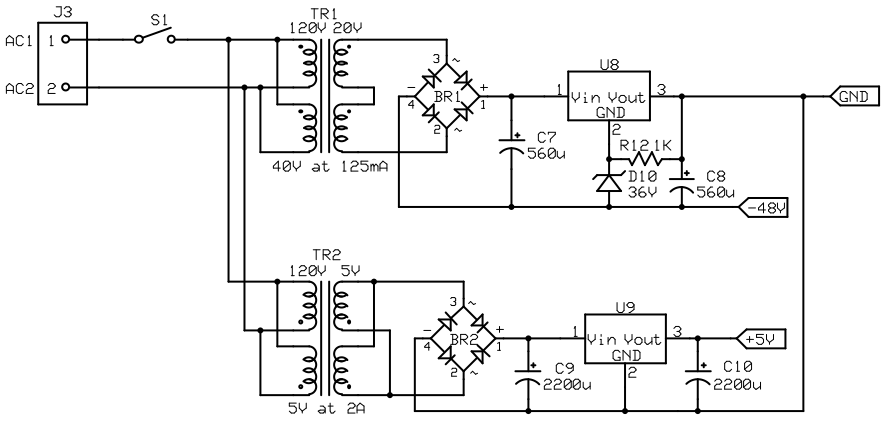

Digikey part numbers
U1-U2=568-1499-5-ND (74HCT00) quad 2-input NAND
U3A-D=CLA108-ND Clare TS117 multifuntion telecom relay
U4=PIC16F872-I/SP-ND 2Kx14 20MHz 22I/O EE64x8 RAM128x8
X1=490-1213-ND 10MHz ceramic resonator with built-in capacitors
ER=DS1834+-ND 8-DIP econo-reset
K1A-D=PB115-ND SPDT relay 24V@36mA 20A contacts
K2A-D,K3A-D=PB288-ND DPDT relay 5V@28mA 2A contacts
D1A-D,D2A-D,D3A-D=1N4148FS-ND 4ns diode
D4A-D,D5A-D=1N5355BTPMSCT-ND 18V 5W DO-15 zener diode
C1-C3=478-1835-ND 1uF 35V tantalum
Resistors all 1/4W 1% (Digikey part number of form P****CACT-ND)
J1-J2=made from 281-1435-ND (2-pin) and 281-1436-ND (3-pin)
ATM Consolidator - Schematic #1



Digikey part numbers

U5-U6=497-2356-5-ND 8-channel Darligton driver 500mA
U7=PIC16F872-I/SP-ND 2Kx14 20MHz 22I/O EE64x8 RAM128x8
X2=490-1213-ND 10MHz ceramic resonator with built-in capacitors
K4A-D,K5A-D,K6A-D=PB288-ND DPDT relay 5V@28mA 2A contacts
D6A-D,D7A-D=1N4148FS-ND 4ns diode
D8A-D,D9A-D=516-1328-ND red 5mm LED
C4-C6=478-1835-ND 1uF 35V tantalum
Resistors all 1/4W 1% (Digikey part number of form P****CACT-ND)

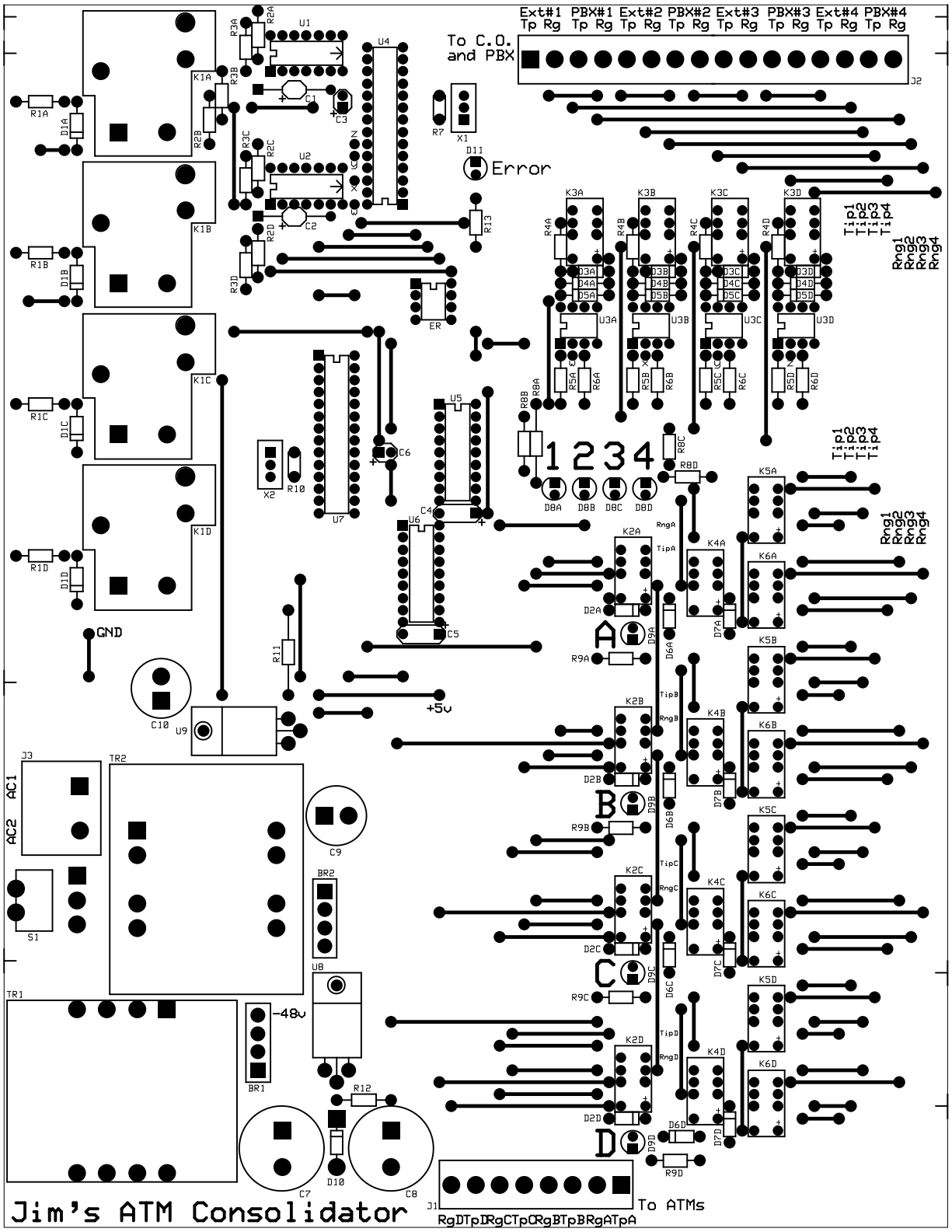
ATM Consolidator - Schematic #2



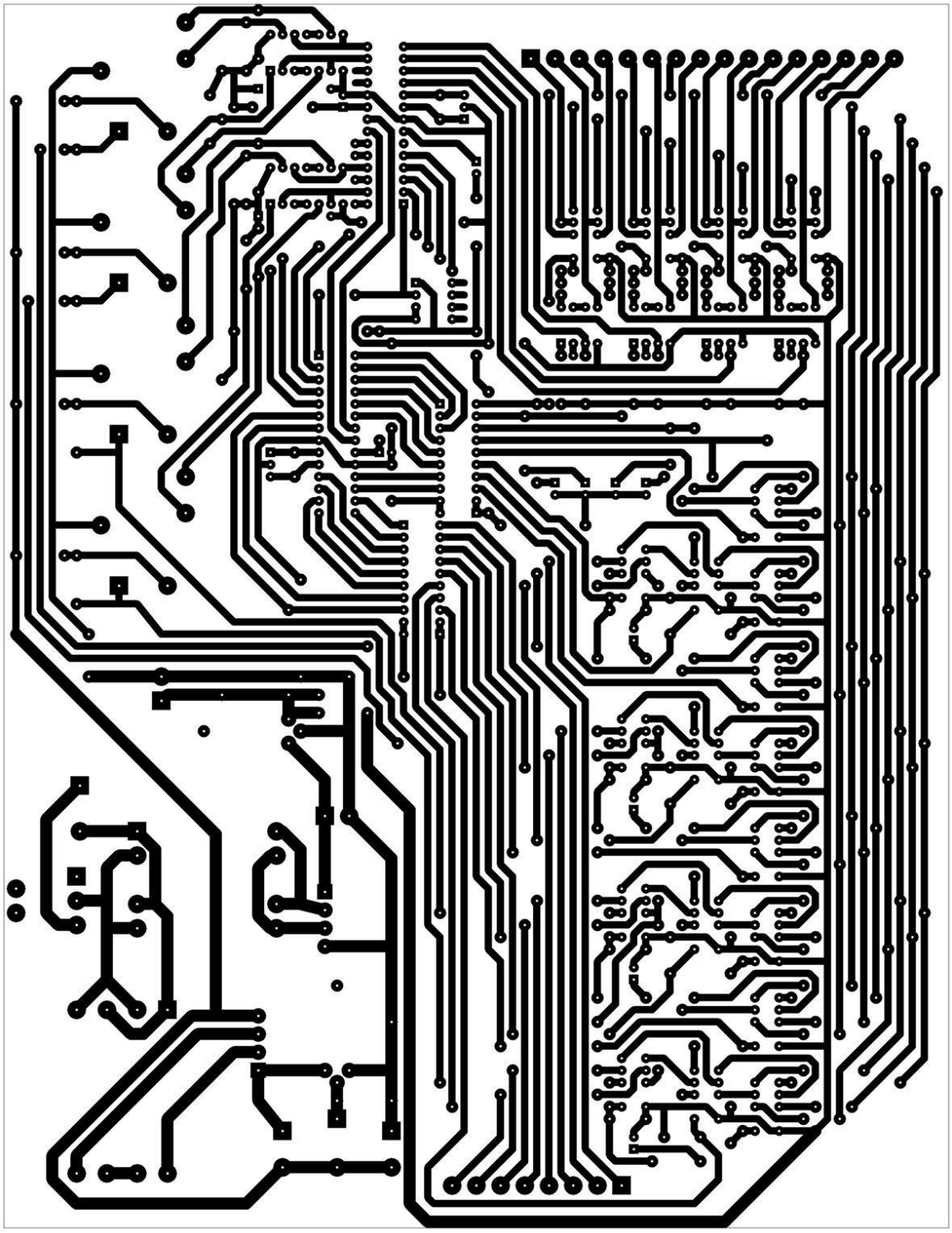
Digikey part numbers

S1=360-1877-ND SPDT PC-mount toggle switch
TR1=595-1157-ND 120V-to-dual 20V 500mA transformer
TR2=595-1008-ND 120V-to-dual 8V 620mA transformer
BR1-BR2=KBP2005GDI-ND 2A 50V bridge rectifier
U8=LM7812ACT-ND 12V 1A TO-220 voltage regulator
U9=LM7805CT-ND 5V 1A TO-220 voltage regulator
C7-C8=493-1986-ND 560uF 100V electrolytic capacitor
C9-C10=P5157-ND 2200uF 25V electrolytic capacitor
D10=1N5365BGOS-ND 36V 5W zener diode
Resistors all 1/4W 1% (Digikey part number of form P****CACT-ND)
J3=A98481-ND 2-pin 0.375-inch tri-barrier terminal strip

ATM Consolidator
Schematic #3 - Power Supply



(Scale to 7.8" wide by 10.1" high.)



(Scale to 7.8” wide by 10.1” high.)

```
; ATM Consolidator Program for PIC#1 (ATMPIC#1, compiles to ATMPIC#1.HEX)
;
; All communications from PIC#1 to PIC#2 take the form of two nibbles, sent one immediately
; after the other, so that PIC#2 receives them in the same interrupt service routine.  The
; first nibble contains an instruction code; the second nibble carries data.  PIC#2 does not
; send any information to PIC#1.  Therefore, it is possible to leave the data bus (portA<3-0>)
; configured for output by PIC#1 at all times.
;
; The commands which PIC#1 sends to PIC#2 have the following protocols:-
;   b'cccc'=1 means "make the following connection"
;             Data nibble of form b'ffgg', where
;             ff=0-3 is the external line (Line#1=0 ... Line#4=3)
;             gg=0-3 is the ATM (ATM #A=0 ... ATM #D=3) to connect to the external line
;   b'cccc'=2 means "break the following connection"
;             Data nibble of form b'ffgg', where
;             ff=0-3 is the external line (Line#1=0 ... Line#4=3)
;             gg=0-3 is the ATM (ATM #A=0 ... ATM #D=3) to disconnect from the external line
;   b'cccc'=3 means disconnect all ATMs from all external lines
;             Data nibble is ignored by PIC#2
;
; Three procedures needs to be timed: (i) every communication with PIC#2 must time out
; gracefully if not completed within, say, 25ms; (ii) if any ATM keeps control of an
; external line for more than, say, five minutes, the connection must be physically
; broken by software; and (iii) after any ATM-external line connection is broken, the
; ATM must be kept physically disconnected for, say, three seconds, in order to prevent it
; from immediately re-connecting to another external line before the ATM's circuitry has
; time to process the disconnect.
;
; Register timer0 and its control module, Timer0, are used to catch communication timeouts.
; The Timer0 prescalar is set to 256:1, so timer0 increments every 256 x 400ns = 102.4us.
; (Remember that, with a clock at 10MHz, each clock cycle takes 100ns and each
; instruction cycle, consisting of four clock cycles, takes 400ns.)  Therefore, timer0 will
; count from 0 to 0 (256 steps) in 256 x 102.4us = 26.2144ms.  The Timer0 module is set to
; run continuously from the time PIC#1 starts up.  Register timer0 is never reset to zero.
;
; When a communication to PIC#2 starts, register counterTO is set to d'2'.  It is
; decremented at every Timer0 overflow-initiated interrupt.  If and when it reaches zero,
; a communication timeout is declared.  Since the exact contents of register timer0 will
; not be known at the start of this timing cycle, it will take between d'1' and d'2'
; timer0 overflows for register counterTO to reach zero.  Therefore, communications with
; PIC#2 will time out between 1 x 26.2144ms = 26.2144ms and 2 x 26.2144ms = 52.4288ms.
; This is sufficiently close to the 25ms design time for our purposes.
;
; What to do in the event of a communication timeout with PIC#2?  PIC#2 has a similar timer
; to detect communication timeouts with PIC#1.  If PIC#2 is still functioning, it will also
; detect the timeout, in which case it will HardReset itself.  When PIC#2 restarts itself,
; all connections between the ATMs and the external telephone lines will be cut.  Cutting
; the connections will mean that all information PIC#1 has about the connections will be
; irrelevant.  The simplest solution is to have both PICs HardReset themselves, and begin
; afresh, in the event of a communication timeout.
;
; The Timer0 module and the timer0-triggered interrupt are also used to time the procedures
; described in clauses (ii) and (iii) above.  These two durations are much longer than the
; communication timeout, being measured in minutes or seconds instead of in milli-seconds.
; In order to measures these durations, it is necessary to use additional registers which
; count the timer0 interrupts as they occur.
;
; Let us deal with the type (ii) interrupt first.  At the start of every connection between
; ATM #A and an external line, register counterAH is set to d'45' and register counterAL is
; set to d'0'.  Register pair <counterAH, counterAL> is decremented at every Timer0
; overflow-initiated interrupt.  As before, register timer0 is not reset at the start of
; counting.  256 x d'45' = 11,520.  Therefore, it will take between 11,519 and 11,520 Timer0
; overflow interrupts before register pair <counterAH, counterAL> reaches zero.  This
; will take a length of time equal to between 11,519 x 301,963.6736ms = 301.96 seconds and
; 11,520 x 26.2144ms = 301,989.888ms = 301.99 seconds, both being very close to five minutes.
; If and when this occurs, it is reasonable to declare that ATM #A has ceased operating, and
; to sever its connection with the external line.  A flag register, counterAon, is set high
; when ATM #A is first connected, for use by the main program in recognizing that register
; pair <counterAH, counterAL> needs to be monitored for five minutes.  A similar register
; pair and flag is used for each of the other three ATMs.
;
```

```
; Now, let us deal with type (iii) Timer0 interrupts.  At the end of every connection
; between ATM #A and an external line, register counterAH is set to d'0' and register
; counterAL is set to d'115'.  Note that this is the same register pair which is used
; when ATM #A first makes a connection with an external line.  We can use the same register
; because there is no overlap between the two operating modes.  Once again, the register
; pair is decremented at every Timer0 overflow-initiated interrupt.  And, once again,
; since the value of register timer0 will not be known at the start of this timing cycle,
; it will take between d'114' and d'115' timer0 overflows for register pair <counterAH,
; counterAL> to reach zero.  This will take a length of time equal to between
; 114 x 26.2144ms = 2.99 seconds and 115 x 26.2144ms = 3.01 seconds, both being very close
; to three seconds.  ATM #A is prevented by software from making any new connection during
; this interval.  A flag register, counterAoff, is set high when ATM #A is first
; disconnected, for use by the main program in recognizing that register pair <counterAH,
; counterAL> needs to be monitored for three seconds.  Note that this is a different flag
; than used above.  A similar flag is used for each of the other three ATMs.
;
; Configuration Word
;    Protect off              ; b<13>=b<12>=b<8>=b<5>=b<4>=1
;    Debugger disabled        ; b<11>=1
;    b<10> is unimplemented   ; b<10>=1
;    WRT enabled              ; b<9>=1
;    LVP off                  ; b<7>=0
;    BOR disabled             ; b<6>=0
;    PUT disabled             ; b<3>=1
;    WDT disabled             ; b<2>=0
;    OSC HS                   ; b<1>=1; b<0>=0
; Crystal frequency = 10MHz
;
  processor       16F872
  __config        0x3F3A     ; b'0011 1111 0011 1010'
;
; Variable definitions - PIC 16F87x control registers
;
timer0       equ   0x01
status       equ   0x03
carry        equ      0x00
zero         equ      0x02
page0        equ      0x05
page1        equ      0x06
portA        equ   0x05
portB        equ   0x06
portC        equ   0x07
INTCon       equ   0x0B       ; interrupt control register
RB0IF        equ      0x01    ; RB0 interrupt flag
Tmr0IF       equ      0x02    ; Timer0 interrupt flag
RB0IE        equ      0x04    ; RB0 interrupt enable
Tmr0IE       equ      0x05    ; Timer0 interrupt enable
GIE          equ      0x07    ; global interrupt enable
T1CON        equ   0x10       ; controls use of portC<1-0>
CCP1CON      equ   0x17       ; controls use of portC<2>
optionreg    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
ADCON1       equ   0x9F       ; controls use of portA
f            equ      0x01    ; f and w identify the destination register
w            equ      0x00
;
; Variable definitions - User RAM - Accessible only in bank0
;
; I/O port registers
portAmirror  equ   0x20       ; portA<4> is not used
D0           equ      0x00    ; portA<3-0> is four-bit data bus with PIC#2
D1           equ      0x01
D2           equ      0x02
D3           equ      0x03
Err          equ      0x05    ; open drain, active low, used to drive an "Error" LED
portBmirror  equ   0x21
INT          equ      0x00    ; interrupt request input from PIC#2
ACK          equ      0x01          ; interrupt acknowledge output to PIC#2
CL4          equ      0x02    ; check external line #4, active low
```

```
CL3          equ     0x03    ; check external line #3, active low
hookstatusC  equ     0x04    ; hookstatus input bits are low if ATM is offhook, else high
CL2          equ     0x05    ; check external line #2, active low
hookstatusD  equ     0x06
CL1          equ     0x07    ; check external line#1, active low
portCmirror  equ   0x22      ; portC<7-6> are not used
LIU4         equ     0x00    ; Line-in-use bits are low if line is available, high if ...
LIU3         equ     0x01    ; ... (i) in use, or (ii) no line is connected
LIU1         equ     0x02
LIU2         equ     0x03
hookstatusA  equ     0x04    ; hookstatus input bits are low if ATM is offhook, high if ...
hookstatusB  equ     0x05    ; ... (i) onkook, or (ii) no ATM is connected
; registers for transmitting to PIC#2
Scommand1    equ   0x23      ; first nibble of command to be sent to PIC#2
Scommand2    equ   0x24      ; second nibble of command to be sent to PIC#2
; registers for dealing with ATMs
offhookreg   equ   0x25      ; high when ATM is offhook, low if onhook
ATMA         equ     0x00    ; ATM #A
ATMB         equ     0x01    ; ATM #B
ATMC         equ     0x02    ; ATM #C
ATMD         equ     0x03    ; ATM #D
servicereg   equ   0x26      ; high when an ATM is being serviced
; registers for dealing with external lines
extlinereg   equ   0x27      ; low if line is available, high if in use (offhook)
Line1        equ     0x00
Line2        equ     0x01
Line3        equ     0x02
Line4        equ     0x03
LIUreg       equ   0x28      ; high if line is in use by an ATM, low if not in use
; registers for matching ATMs with external lines
ATMassign    equ   0x29      ; <1,0>=line connected to ATM #A (line #1=0 ... line #4=3)
                             ; <3,2>=line connected to ATM #B (line #1=0 ... line #4=3)
                             ; <5,4>=line connected to ATM #C (line #1=0 ... line #4=3)
                             ; <7,6>=line connected to ATM #D (line #1=0 ... line #4=3)
; registers for timeout counters
counterTO    equ   0x2A      ; counter for communication timeouts with PIC#2
counterAH    equ   0x2B      ; counter for ATM #A duration checks
counterAL    equ   0x2C
counterAon   equ   0x2D      ; flag for five-minute ATM #A line seizure test
counterAoff  equ   0x2E      ; flag for three-second ATM #A out-of-service requirement
counterBH    equ   0x2F      ; counter for ATM #B duration checks
counterBL    equ   0x30
counterBon   equ   0x31      ; flag for five-minute ATM #B line seizure test
counterBoff  equ   0x32      ; flag for three-second ATM #B out-of-service requirement
counterCH    equ   0x33      ; counter for ATM #C duration checks
counterCL    equ   0x34
counterCon   equ   0x35      ; flag for five-minute ATM #C line seizure test
counterCoff  equ   0x36      ; flag for three-second ATM #C out-of-service requirement
counterDH    equ   0x37      ; counter for ATM #D duration checks
counterDL    equ   0x38
counterDon   equ   0x39      ; flag for five-minute ATM #D line seizure test
counterDoff  equ   0x3A      ; flag for three-second ATM #D out-of-service requirement
; registers for temporary use
temp         equ   0x3B      ; temporary register
tempSCP2     equ   0x3C      ; temporary register for subroutine SendCommandToPIC2
tempTest     equ 0x3D        ; temporary register used in all test programs
tempTest5A   equ   0x3E      ; temporary register used in TestProgram5
tempTest5B   equ   0x3F      ; temporary register used in TestProgram5
; registers for counting delay cycles
count1       equ   0x40      ; counters for delay subroutines
count2       equ   0x41
count3       equ   0x42
;
; Variable definitions - User RAM - Accessible in all banks
;
_wTmr0       equ   0x70      ; temp storage for the w register during Timer0 interrupts
_sTmr0       equ   0x71      ; temp storage for the status register during Timer0 interrupts
```

```
;
; Location 0x0000
;
     org     0x0000
     goto    HardReset               ; goto HardReset on power-up
     nop
     nop
     nop
;
; ********************************************************************************************
; ********************************************************************************************
; Interrupt Service Routine starts at address 0x0004
; There is only one source of interrupt: by Timer0.  At each interrupt, certain counter
; registers are decremented.  The ISR itself does not take any action if a counter reaches
; zero.  Instead, the main program monitors the counter registers of interest and takes
; whatever action is appropriate when one of them reaches zero.  Note that the counter
; registers are decremented to zero, but no further.  Once they reach zero, they remain
; zero.  This allows the main program to detect zero conditions at its leisure.
; ********************************************************************************************
; ********************************************************************************************
;
     org     0x0004
ISR
     bcf     INTCon,GIE              ; disable global interrupts, but do not clear flags
     bcf     INTCon,Tmr0IE           ; disable Timer0 interrupts
     movwf   _wTmr0                  ; save the w register during Timer0 interrupts
     movf    status,w
     movwf   _sTmr0                  ; save the status register during Timer0 interrupts
ISRTestTO
     movf    counterTO,w             ; set the zero flag based on the contents of counterTO
     btfss   status,zero             ; skip if counterTO is already zero
ISRDecrTO
     decf    counterTO,f             ; decrement register counterTO
ISRTestA
     movf    counterAL,w             ; set the zero flag based on the contents of counterAL
     btfss   status,zero             ; skip if counterAL is already zero
     goto    ISRDecrA                ; since counterAL is not zero, goto to decrement
     movf    counterAH,w             ; set the zero flag based on the contents of counterAH
     btfsc   status,zero             ; skip if counterAH is not zero
     goto    ISRTestB                ; <counterAH, counterAL> is identically zero, so goto
ISRDecrA
     decf    counterAL,f             ; decrement register pair <counterAH, counterAL>
     movf    counterAL,w
     xorlw   0xFF                    ; if counterAL=b'11111111', then xorAL=0
     btfsc   status,zero             ; if xorAL=0, then ...
     decf    counterAH,f             ; ... decrement counterAH
ISRTestB
     movf    counterBL,w             ; set the zero flag based on the contents of counterBL
     btfss   status,zero             ; skip if counterBL is already zero
     goto    ISRDecrB                ; since counterBL is not zero, goto to decrement
     movf    counterBH,w             ; set the zero flag based on the contents of counterBH
     btfsc   status,zero             ; skip if counterBH is not zero
     goto    ISRTestC                ; <counterBH, counterBL> is identically zero, so goto
ISRDecrB
     decf    counterBL,f             ; decrement register pair <counterBH, counterBL>
     movf    counterBL,w
     xorlw   0xFF                    ; if counterBL=b'11111111', then xorBL=0
     btfsc   status,zero             ; if xorBL=0, then ...
     decf    counterBH,f             ; ... decrement counterBH
ISRTestC
     movf    counterCL,w             ; set the zero flag based on the contents of counterCL
     btfss   status,zero             ; skip if counterCL is already zero
     goto    ISRDecrC                ; since counterCL is not zero, goto to decrement
     movf    counterCH,w             ; set the zero flag based on the contents of counterCH
     btfsc   status,zero             ; skip if counterCH is not zero
     goto    ISRTestD                ; <counterCH, counterCL> is identically zero, so goto
ISRDecrC
     decf    counterCL,f             ; decrement register pair <counterCH, counterCL>
     movf    counterCL,w
     xorlw   0xFF                    ; if counterCL=b'11111111', then xorCL=0
     btfsc   status,zero             ; if xorCL, then ...
```

```
     decf    counterCH,f             ; ... decrement counterCH
ISRTestD
     movf    counterDL,w             ; set the zero flag based on the contents of counterDL
     btfss   status,zero             ; skip if counterDL is already zero
     goto    ISRDecrD                ; since counterDL is not zero, goto to decrement
     movf    counterDH,w             ; set the zero flag based on the contents of counterDH
     btfsc   status,zero             ; skip if counterDH is not zero
     goto    ISRDone                 ; <counterDH, counterDL> is identically zero, so goto
ISRDecrD
     decf    counterDL,f             ; decrement register pair <counterDH, counterDL>
     movf    counterDL,w
     xorlw   0xFF                    ; if counterDL=b'11111111', then xorDL=0
     btfsc   status,zero             ; if xorDL=0, then ...
     decf    counterDH,f             ; ... decrement counterDH
ISRDone
     movf    _sTmr0,w
     movwf   status                  ; retrieve the original status register
     movf    _wTmr0,w                ; retrieve the orginal w register
     bcf     INTCon,Tmr0IF           ; clear Timer0 interrupt flag
     bsf     INTCon,Tmr0IE           ; re-enable Timer0 interrupts
     bsf     INTCon,GIE              ; re-enable global interrupts
     retfie                          ; return from the interrupt
;
; ********************************************************************************************
; ********************************************************************************************
; Hard reset
; ********************************************************************************************
; ********************************************************************************************
;
HardReset
     bcf     INTCon,GIE              ; disable global interrupts (until needed)
     bcf     INTCon,RB0IE            ; disable RB0 interrupts (permanent)
     bcf     INTCon,Tmr0IE           ; disable Timer0 interrupts (until needed)
     bcf     INTCon,Tmr0IF           ; clear Timer0 interrupt flag
     clrf    portA                   ; clear all I/O latches
     clrf    portB
     clrf    portC
     movlw   0x00                    ; set T1CON=0 to disable Timer1 and ...
     movwf   T1CON                   ; ... release portC<1-0> for digital I/O
     movlw   0x00                    ; set CCP1CON=0 to disable Capture/Compare/PWM and ..
     movwf   CCP1CON                 ; ... release portC<2> for digital I/O
     bsf     status,page0            ; select register bank 1
     bcf     status,page1
     movlw   0x06                    ; set ADCON1<3-1>=b'011' to ...
     movwf   ADCON1                  ; ... configure portA for digital I/O
     movlw   0x00                    ; configure portA<5-0> for output
     movwf   TRISA
     movlw   0x51                    ; configure portB<7,5,3-1> for O, portB<6,4,0> for I
     movwf   TRISB
     movlw   0x3F                    ; configure portC<7-6> for output, portC<5-0> for input
     movwf   TRISC
     bsf     optionreg,7             ; <7>=1 disables PortB pull-up resistors
     bsf     optionreg,6             ; <6>=1 triggers RB0 interrupts on the rising edge
     bcf     optionreg,5             ; <5>=0 uses the internal clock to increment Timer0
     bcf     optionreg,4             ; <4>=0 increments timer0 on low-to-high transitions
     bcf     optionreg,3             ; <3>=0 assigns the prescalar to the Timer0 module
     bsf     optionreg,2             ; <2-0>=111 sets the prescalar for timer0 to 256:1
     bsf     optionreg,1
     bsf     optionreg,0
     bcf     status,page0            ; select register bank 0
     bcf     status,page1
InitializeRegistersAndSettings
     clrf    portAmirror
     clrf    portBmirror
     clrf    portCmirror
     bcf     portBmirror,ACK         ; assert interrupt request to PIC#2 low
     bsf     portBmirror,CL1         ; deactivate TS117 relay for external line #1
     bsf     portBmirror,CL2         ; deactivate TS117 relay for external line #2
     bsf     portBmirror,CL3         ; deactivate TS117 relay for external line #3
     bsf     portBmirror,CL4         ; deactivate TS117 relay for external line #4
     movf    portBmirror,w
```

```
     movwf    portB
     clrf     offhookreg              ; offhook input bits set low; all ATMs are onhook
     clrf     servicereg              ; service flag bits set low; no ATMs are in service
     clrf     LIUreg                  ; LIU input bits set low; all lines are available
     clrf     ATMassign               ; no ATMs are in service
     clrf     counterAon
     clrf     counterAoff
     clrf     counterBon
     clrf     counterBoff
     clrf     counterCon
     clrf     counterCoff
     clrf     counterDon
     clrf     counterDoff
     call     TurnOffErrorLED         ; start up with the Error Condition LED off
BeginExecution
     call     del500ms                ; wait one second for circuits to settle, while ...
     call     del500ms                ; ... while PIC#2 only waits one-half second
     bcf      INTCon,Tmr0IF           ; clear Timer0 interrupt flag
     bsf      INTCon,Tmr0IE           ; enable Timer0 interrupts
     bsf      INTCon,GIE              ; enable global interrupts
     goto     MainLoop                ; begin the main program
;    goto     TestProgram1            ; to execute test programs, compile with desired goto
;    goto     TestProgram2
;    goto     TestProgram3
;    goto     TestProgram4
;    goto     TestProgram5
;
; ********************************************************************************************
; ********************************************************************************************
; Start of main loop
MainLoop
     call     del10ms                 ; 10ms delay between loops for circuits to settle
;
; ********************************************************************************************
; Part A: Deal with ATM#A
MainLoopATMA
     btfss    servicereg,ATMA
     goto     ATMANotInService
     goto     ATMAInService
ATMANotInService
     movf     counterAoff,w           ; set zero flag based on contents of register counterAoff
     btfss    status,zero
     goto     TestATMAOffState        ; counterAoff is activated - still within 3 seconds of stop
     goto     TestATMAStatus          ; counterAoff if not activated, so ATM#A can be serviced
TestATMAOffState
     movf     counterAH,w
     iorwf    counterAL,w             ; result is zero if counterA has counted down to zero
     btfss    status,zero
     goto     MainLoopATMB            ; counterA is not zero - ATM#A must be kept off
     clrf     counterAoff             ; counterA is zero, so counterAoff can be cleared
TestATMAStatus
     call     GetATMAStatus           ; returns offhookreg<ATMA>=1 if ATM#A has gone offhook
     btfss    offhookreg,ATMA
     goto     MainLoopATMB            ; ATM#A is still on hook - no service is required
ServiceATMA                           ; ATM#A is requesting service
     call     GetFirstAvailableLine
     movwf    temp                    ; register w holds the first available line
     xorlw    0x00
     btfsc    status,zero             ; the zero flag is set if ATM#A should use Line#1
     goto     AToUseLine1
     movf     temp,w
     xorlw    0x01
     btfsc    status,zero             ; the zero flag is set if ATM#A should use Line#2
     goto     AToUseLine2
     movf     temp,w
     xorlw    0x02
     btfsc    status,zero             ; the zero flag is set if ATM#A should use Line#3
     goto     AToUseLine3
     movf     temp,w
     xorlw    0x03
     btfsc    status,zero             ; the zero flag is set if ATM#A should use Line#4
```

```
     goto    AToUseLine4
     goto    MainLoopATMB           ; no external lines are available for ATM#A
ATMAInService
     movf    counterAH,w
     iorwf   counterAL,w            ; result is zero if counterA has counted down to zero
     btfsc   status,zero
     goto    HangUpATMA             ; line-seizure timer has expired - force ATM#A to hang up
     movf    ATMassign,w
     andlw   0x03                   ; ATMassign<1,0> is the telephone line which ATM#A is using
     movwf   temp                   ; save line assignment in register temp
     xorlw   0x00
     btfsc   status,zero            ; the zero flag is set if ATM#A is using Line#1
     goto    AIsUsingLine1
     movf    temp,w
     xorlw   0x01
     btfsc   status,zero            ; the zero flag is set if ATM#A is using Line#2
     goto    AIsUsingLine2
     movf    temp,w
     xorlw   0x02
     btfsc   status,zero            ; the zero flag is set if ATM#A is using Line#3
     goto    AIsUsingLine3
     movf    temp,w
     xorlw   0x03
     btfsc   status,zero            ; the zero flag is set if ATM#A is using Line#4
     goto    AIsUsingLine4
     call    TurnOnErrorLED         ; we should never be here but, if we are, turn on the ...
     goto    MainLoopATMB           ; ... Error Condition LED and deal with ATM#B
AIsUsingLine1
     call    GetLine1Status         ; returns extlinereg<Line1>=1 if Line#1 is still in use
     btfsc   extlinereg,Line1
     goto    MainLoopATMB           ; ATM#A is still using Line#1 - continue with service
     goto    HangUpATMA             ; Line#1 has disconnected from ATMA#A
AIsUsingLine2
     call    GetLine2Status         ; returns extlinereg<Line2>=1 if Line#2 is still in use
     btfsc   extlinereg,Line2
     goto    MainLoopATMB           ; ATM#A is still using Line#2 - continue with service
     goto    HangUpATMA             ; Line#2 has disconnected from ATMA#A
AIsUsingLine3
     call    GetLine3Status         ; returns extlinereg<Line3>=1 if Line#3 is still in use
     btfsc   extlinereg,Line3
     goto    MainLoopATMB           ; ATM#A is still using Line#3 - continue with service
     goto    HangUpATMA             ; Line#3 has disconnected from ATMA#A
AIsUsingLine4
     call    GetLine4Status         ; returns extlinereg<Line4>=1 if Line#4 is still in use
     btfsc   extlinereg,Line4
     goto    MainLoopATMB           ; ATM#A is still using Line#4 - continue with service
     goto    HangUpATMA             ; Line#4 has disconnected from ATMA#A
HangUpATMA
     clrf    counterAon
     movf    ATMassign,w
     andlw   0x03                   ; ATMassign<1,0> is the line which ATM#A was using
     movwf   temp                   ; save line assignment in register temp
     xorlw   0x00
     btfsc   status,zero            ; the zero flag is set if ATM#A was using Line#1
     goto    AWasUsingLine1
     movf    temp,w
     xorlw   0x01
     btfsc   status,zero            ; the zero flag is set if ATM#A was using Line#2
     goto    AWasUsingLine2
     movf    temp,w
     xorlw   0x02
     btfsc   status,zero            ; the zero flag is set if ATM#A was using Line#3
     goto    AWasUsingLine3
     movf    temp,w
     xorlw   0x03
     btfsc   status,zero            ; the zero flag is set if ATM#A was using Line#4
     goto    AWasUsingLine4
     call    TurnOnErrorLED         ; we should never be here but, if we are, turn on the ...
     goto    MainLoopATMB           ; ... Error Condition LED and deal with ATM#B
```

```
;
; ********************************************************************************************
; Part B: Deal with ATM#B
MainLoopATMB
     btfss    servicereg,ATMB
     goto     ATMBNotInService
     goto     ATMBInService
ATMBNotInService
     movf     counterBoff,w          ; set zero flag based on contents of register counterBoff
     btfss    status,zero
     goto     TestATMBOffState       ; counterBoff is activated - still within 3 seconds of stop
     goto     TestATMBStatus         ; counterBoff if not activated, so ATM#B can be serviced
TestATMBOffState
     movf     counterBH,w
     iorwf    counterBL,w            ; result is zero if counterB has counted down to zero
     btfss    status,zero
     goto     MainLoopATMC           ; counterB is not zero - ATM#B must be kept off
     clrf     counterBoff            ; counterB is zero, so counterBoff can be cleared
TestATMBStatus
     call     GetATMBStatus          ; returns offhookreg<ATMB>=1 if ATM#B has gone offhook
     btfss    offhookreg,ATMB
     goto     MainLoopATMC           ; ATM#B is still on hook - no service is required
ServiceATMB                          ; ATM#B is requesting service
     call     GetFirstAvailableLine
     movwf    temp                   ; register w holds the first available line
     xorlw    0x00
     btfsc    status,zero            ; the zero flag is set if ATM#B should use Line#1
     goto     BToUseLine1
     movf     temp,w
     xorlw    0x01
     btfsc    status,zero            ; the zero flag is set if ATM#B should use Line#2
     goto     BToUseLine2
     movf     temp,w
     xorlw    0x02
     btfsc    status,zero            ; the zero flag is set if ATM#B should use Line#3
     goto     BToUseLine3
     movf     temp,w
     xorlw    0x03
     btfsc    status,zero            ; the zero flag is set if ATM#B should use Line#4
     goto     BToUseLine4
     goto     MainLoopATMC           ; no external lines are available for ATM#B
ATMBInService
     movf     counterBH,w
     iorwf    counterBL,w            ; result is zero if counterB has counted down to zero
     btfsc    status,zero
     goto     HangUpATMB             ; line-seizure timer has expired - force ATM#B to hang up
     movf     ATMassign,w
     andlw    0x0C                   ; ATMassign<3,2> is the telephone line which ATM#B is using
     movwf    temp                   ; save line assignment in register temp
     xorlw    0x00
     btfsc    status,zero            ; the zero flag is set if ATM#B is using Line#1
     goto     BIsUsingLine1
     movf     temp,w
     xorlw    0x04
     btfsc    status,zero            ; the zero flag is set if ATM#B is using Line#2
     goto     BIsUsingLine2
     movf     temp,w
     xorlw    0x08
     btfsc    status,zero            ; the zero flag is set if ATM#B is using Line#3
     goto     BIsUsingLine3
     movf     temp,w
     xorlw    0x0C
     btfsc    status,zero            ; the zero flag is set if ATM#B is using Line#4
     goto     BIsUsingLine4
     call     TurnOnErrorLED         ; we should never be here but, if we are, turn on the ...
     goto     MainLoopATMC           ; ... Error Condition LED and deal with ATM#C
BIsUsingLine1
     call     GetLine1Status         ; returns extlinereg<Line1>=1 if Line#1 is still in use
     btfsc    extlinereg,Line1
     goto     MainLoopATMC           ; ATM#B is still using Line#1 - continue with service
     goto     HangUpATMB             ; Line#1 has disconnected from ATMA#B
```

```
BIsUsingLine2
     call    GetLine2Status          ; returns extlinereg<Line2>=1 if Line#2 is still in use
     btfsc   extlinereg,Line2
     goto    MainLoopATMC            ; ATM#B is still using Line#2 - continue with service
     goto    HangUpATMB              ; Line#2 has disconnected from ATMA#B
BIsUsingLine3
     call    GetLine3Status          ; returns extlinereg<Line3>=1 if Line#3 is still in use
     btfsc   extlinereg,Line3
     goto    MainLoopATMC            ; ATM#B is still using Line#3 - continue with service
     goto    HangUpATMB              ; Line#3 has disconnected from ATMA#B
BIsUsingLine4
     call    GetLine4Status          ; returns extlinereg<Line4>=1 if Line#4 is still in use
     btfsc   extlinereg,Line4
     goto    MainLoopATMC            ; ATM#B is still using Line#4 - continue with service
     goto    HangUpATMB              ; Line#4 has disconnected from ATMA#B
HangUpATMB
     clrf    counterBon
     movf    ATMassign,w
     andlw   0x0C                    ; ATMassign<3,2> is the line which ATM#B was using
     movwf   temp                    ; save line assignment in register temp
     xorlw   0x00
     btfsc   status,zero             ; the zero flag is set if ATM#B was using Line#1
     goto    BWasUsingLine1
     movf    temp,w
     xorlw   0x04
     btfsc   status,zero             ; the zero flag is set if ATM#B was using Line#2
     goto    BWasUsingLine2
     movf    temp,w
     xorlw   0x08
     btfsc   status,zero             ; the zero flag is set if ATM#B was using Line#3
     goto    BWasUsingLine3
     movf    temp,w
     xorlw   0x0C
     btfsc   status,zero             ; the zero flag is set if ATM#B was using Line#4
     goto    BWasUsingLine4
     call    TurnOnErrorLED          ; we should never be here but, if we are, turn on the ...
     goto    MainLoopATMC            ; ... Error Condition LED and deal with ATM#C
;
; ********************************************************************************************
; Part C: Deal with ATM#C
MainLoopATMC
     btfss   servicereg,ATMC
     goto    ATMCNotInService
     goto    ATMCInService
ATMCNotInService
     movf    counterCoff,w           ; set zero flag based on contents of register counterCoff
     btfss   status,zero
     goto    TestATMCOffState        ; counterCoff is activated - still within 3 seconds of stop
     goto    TestATMCStatus          ; counterCoff if not activated, so ATM#C can be serviced
TestATMCOffState
     movf    counterCH,w
     iorwf   counterCL,w             ; result is zero if counterC has counted down to zero
     btfss   status,zero
     goto    MainLoopATMD            ; counterC is not zero - ATM#C must be kept off
     clrf    counterCoff             ; counterC is zero, so counterCoff can be cleared
TestATMCStatus
     call    GetATMCStatus           ; returns offhookreg<ATMC>=1 if ATM#C has gone offhook
     btfss   offhookreg,ATMC
     goto    MainLoopATMD            ; ATM#C is still on hook - no service is required
ServiceATMC                          ; ATM#C is requesting service
     call    GetFirstAvailableLine
     movwf   temp                    ; register w holds the first available line
     xorlw   0x00
     btfsc   status,zero             ; the zero flag is set if ATM#C should use Line#1
     goto    CToUseLine1
     movf    temp,w
     xorlw   0x01
     btfsc   status,zero             ; the zero flag is set if ATM#C should use Line#2
     goto    CToUseLine2
     movf    temp,w
     xorlw   0x02
```

```
     btfsc    status,zero            ; the zero flag is set if ATM#C should use Line#3
     goto     CToUseLine3
     movf     temp,w
     xorlw    0x03
     btfsc    status,zero            ; the zero flag is set if ATM#C should use Line#4
     goto     CToUseLine4
     goto     MainLoopATMC           ; no external lines are available for ATM#C
ATMCInService
     movf     counterCH,w
     iorwf    counterCL,w            ; result is zero if counterC has counted down to zero
     btfsc    status,zero
     goto     HangUpATMC             ; line-seizure timer has expired - force ATM#C to hang up
     movf     ATMassign,w
     andlw    0x30                   ; ATMassign<5,4> is the telephone line which ATM#C is using
     movwf    temp                   ; save line assignment in register temp
     xorlw    0x00
     btfsc    status,zero            ; the zero flag is set if ATM#C is using Line#1
     goto     CIsUsingLine1
     movf     temp,w
     xorlw    0x10
     btfsc    status,zero            ; the zero flag is set if ATM#C is using Line#2
     goto     CIsUsingLine2
     movf     temp,w
     xorlw    0x20
     btfsc    status,zero            ; the zero flag is set if ATM#C is using Line#3
     goto     CIsUsingLine3
     movf     temp,w
     xorlw    0x30
     btfsc    status,zero            ; the zero flag is set if ATM#C is using Line#4
     goto     CIsUsingLine4
     call     TurnOnErrorLED         ; we should never be here but, if we are, turn on the ...
     goto     MainLoopATMD           ; ... Error Condition LED and deal with ATM#D
CIsUsingLine1
     call     GetLine1Status         ; returns extlinereg<Line1>=1 if Line#1 is still in use
     btfsc    extlinereg,Line1
     goto     MainLoopATMD           ; ATM#C is still using Line#1 - continue with service
     goto     HangUpATMC             ; Line#1 has disconnected from ATMA#C
CIsUsingLine2
     call     GetLine2Status         ; returns extlinereg<Line2>=1 if Line#2 is still in use
     btfsc    extlinereg,Line2
     goto     MainLoopATMD           ; ATM#C is still using Line#2 - continue with service
     goto     HangUpATMC             ; Line#2 has disconnected from ATM#C
CIsUsingLine3
     call     GetLine3Status         ; returns extlinereg<Line3>=1 if Line#3 is still in use
     btfsc    extlinereg,Line3
     goto     MainLoopATMD           ; ATM#C is still using Line#3 - continue with service
     goto     HangUpATMC             ; Line#3 has disconnected from ATMA#C
CIsUsingLine4
     call     GetLine4Status         ; returns extlinereg<Line4>=1 if Line#4 is still in use
     btfsc    extlinereg,Line4
     goto     MainLoopATMD           ; ATM#C is still using Line#4 - continue with service
     goto     HangUpATMC             ; Line#4 has disconnected from ATMA#C
HangUpATMC
     clrf     counterCon
     movf     ATMassign,w
     andlw    0x30                   ; ATMassign<5,4> is the line which ATM#C was using
     movwf    temp                   ; save line assignment in register temp
     xorlw    0x00
     btfsc    status,zero            ; the zero flag is set if ATM#C was using Line#1
     goto     CWasUsingLine1
     movf     temp,w
     xorlw    0x10
     btfsc    status,zero            ; the zero flag is set if ATM#C was using Line#2
     goto     CWasUsingLine2
     movf     temp,w
     xorlw    0x20
     btfsc    status,zero            ; the zero flag is set if ATM#C was using Line#3
     goto     CWasUsingLine3
     movf     temp,w
     xorlw    0x30
     btfsc    status,zero            ; the zero flag is set if ATM#C was using Line#4
```

```
     goto    CWasUsingLine4
     call    TurnOnErrorLED          ; we should never be here but, if we are, turn on the ...
     goto    MainLoopATMD            ; ... Error Condition LED and deal with ATM#D
;
; ********************************************************************************
; Part D: Deal with ATM#D
MainLoopATMD
     btfss   servicereg,ATMD
     goto    ATMDNotInService
     goto    ATMDInService
ATMDNotInService
     movf    counterDoff,w           ; set zero flag based on contents of register counterDoff
     btfss   status,zero
     goto    TestATMDOffState        ; counterDoff is activated - still within 3 seconds of stop
     goto    TestATMDStatus          ; counterDoff if not activated, so ATM#D can be serviced
TestATMDOffState
     movf    counterDH,w
     iorwf   counterDL,w             ; result is zero if counterD has counted down to zero
     btfss   status,zero
     goto    MainLoop                ; counterD is not zero - ATM#D must be kept off
     clrf    counterDoff             ; counterD is zero, so counterDoff can be cleared
TestATMDStatus
     call    GetATMDStatus           ; returns offhookreg<ATMD>=1 if ATM#D has gone offhook
     btfss   offhookreg,ATMD
     goto    MainLoop                ; ATM#D is still on hook - no service is required
ServiceATMD                          ; ATM#D is requesting service
     call    GetFirstAvailableLine
     movwf   temp                    ; register w holds the first available line
     xorlw   0x00
     btfsc   status,zero             ; the zero flag is set if ATM#D should use Line#1
     goto    DToUseLine1
     movf    temp,w
     xorlw   0x01
     btfsc   status,zero             ; the zero flag is set if ATM#D should use Line#2
     goto    DToUseLine2
     movf    temp,w
     xorlw   0x02
     btfsc   status,zero             ; the zero flag is set if ATM#D should use Line#3
     goto    DToUseLine3
     movf    temp,w
     xorlw   0x03
     btfsc   status,zero             ; the zero flag is set if ATM#D should use Line#4
     goto    DToUseLine4
     goto    MainLoop                ; no external lines are available for ATM#D
ATMDInService
     movf    counterDH,w
     iorwf   counterDL,w             ; result is zero if counterD has counted down to zero
     btfsc   status,zero
     goto    HangUpATMD              ; line-seizure timer has expired - force ATM#D to hang up
     movf    ATMassign,w
     andlw   0xC0                    ; ATMassign<7,6> is the telephone line which ATM#D is using
     movwf   temp                    ; save line assignment in register temp
     xorlw   0x00
     btfsc   status,zero             ; the zero flag is set if ATM#D is using Line#1
     goto    DIsUsingLine1
     movf    temp,w
     xorlw   0x40
     btfsc   status,zero             ; the zero flag is set if ATM#D is using Line#2
     goto    DIsUsingLine2
     movf    temp,w
     xorlw   0x40
     btfsc   status,zero             ; the zero flag is set if ATM#D is using Line#3
     goto    DIsUsingLine3
     movf    temp,w
     xorlw   0xC0
     btfsc   status,zero             ; the zero flag is set if ATM#D is using Line#4
     goto    DIsUsingLine4
     call    TurnOnErrorLED          ; we should never be here but, if we are, turn on the ...
     goto    MainLoop                ; ... Error Condition LED and deal with ATM#A
DIsUsingLine1
     call    GetLine1Status          ; returns extlinereg<Line1>=1 if Line#1 is still in use
```

```
     btfsc    extlinereg,Line1
     goto     MainLoop               ; ATM#D is still using Line#1 - continue with service
     goto     HangUpATMD             ; Line#1 has disconnected from ATMA#D
DIsUsingLine2
     call     GetLine2Status         ; returns extlinereg<Line2>=1 if Line#2 is still in use
     btfsc    extlinereg,Line2
     goto     MainLoop               ; ATM#D is still using Line#2 - continue with service
     goto     HangUpATMD             ; Line#2 has disconnected from ATMA#D
DIsUsingLine3
     call     GetLine3Status         ; returns extlinereg<Line3>=1 if Line#3 is still in use
     btfsc    extlinereg,Line3
     goto     MainLoop               ; ATM#D is still using Line#3 - continue with service
     goto     HangUpATMD             ; Line#3 has disconnected from ATMA#D
DIsUsingLine4
     call     GetLine4Status         ; returns extlinereg<Line4>=1 if Line#4 is still in use
     btfsc    extlinereg,Line4
     goto     MainLoop               ; ATM#D is still using Line#4 - continue with service
     goto     HangUpATMD             ; Line#4 has disconnected from ATMA#D
HangUpATMD
     clrf     counterDon
     movf     ATMassign,w
     andlw    0xC0                   ; ATMassign<7,6> is the line which ATM#D was using
     movwf    temp                   ; save line assignment in register temp
     xorlw    0x00
     btfsc    status,zero            ; the zero flag is set if ATM#D was using Line#1
     goto     DWasUsingLine1
     movf     temp,w
     xorlw    0x40
     btfsc    status,zero            ; the zero flag is set if ATM#D was using Line#2
     goto     DWasUsingLine2
     movf     temp,w
     xorlw    0x80
     btfsc    status,zero            ; the zero flag is set if ATM#D was using Line#3
     goto     DWasUsingLine3
     movf     temp,w
     xorlw    0xC0
     btfsc    status,zero            ; the zero flag is set if ATM#D was using Line#4
     goto     DWasUsingLine4
     call     TurnOnErrorLED         ; we should never be here but, if we are, turn on the ...
     goto     MainLoop               ; ... Error Condition LED and restart the MainLoop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to connect ATM#A to the external lines
;
AToUseLine1
     bsf      servicereg,ATMA        ; record in servicereg that ATM#A is being serviced
     bsf      LIUreg,Line1           ; record in LIUreg that Line#1 is in use by an ATM
     bcf      ATMassign,1            ; record in ATMassign<1-0> that Line#1 (=0) is running
ATM#A
     bcf      ATMassign,0
     movlw    0x01                   ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x00                   ; ff=00 (Line#1) and gg=00 (ATM#A)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms               ; wait one second for the connection to be made.  It takes
     call     del500ms               ; a lot longer than 10ms for the Bell system to react.
     call     StartAonTimer          ; start timer for ATM#A line seizure check
     goto     MainLoopATMB           ; goto Part B of the Main Loop
AToUseLine2
     bsf      servicereg,ATMA        ; record in servicereg that ATM#A is being serviced
     bsf      LIUreg,Line2           ; record in LIUreg that Line#2 is in use by an ATM
     bcf      ATMassign,1            ; record in ATMassign<1-0> that Line#2 (=1) running ATM#A
     bsf      ATMassign,0
     movlw    0x01                   ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x04                   ; ff=01 (Line#2) and gg=00 (ATM#A)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms               ; wait one second for the connection to be made
```

```
     call     del500ms
     call     StartAonTimer           ; start timer for ATM#A line seizure check
     goto     MainLoopATMB            ; goto Part B of the Main Loop
AToUseLine3
     bsf      servicereg,ATMA         ; record in servicereg that ATM#A is being serviced
     bsf      LIUreg,Line3            ; record in LIUreg that Line#3 is in use by an ATM
     bsf      ATMassign,1             ; record in ATMassign<1-0> that Line#3 (=2) running ATM#A
     bcf      ATMassign,0
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x08                    ; ff=10 (Line#3) and gg=00 (ATM#A)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartAonTimer           ; start timer for ATM #A line seizure check
     goto     MainLoopATMB            ; goto Part B of the Main Loop
AToUseLine4
     bsf      servicereg,ATMA         ; record in servicereg that ATM#A is being serviced
     bsf      LIUreg,Line4            ; record in LIUreg that Line#4 is in use by an ATM
     bsf      ATMassign,1             ; record in ATMassign<1-0> that Line#4 (=3) running ATM#A
     bsf      ATMassign,0
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x0C                    ; ff=11 (Line#4) and gg=00 (ATM#A)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartAonTimer           ; start timer for ATM #A line seizure check
     goto     MainLoopATMB            ; goto Part B of the Main Loop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to connect ATM#B to the external lines
;
BToUseLine1
     bsf      servicereg,ATMB         ; record in servicereg that ATM#B is being serviced
     bsf      LIUreg,Line1            ; record in LIUreg that Line#1 is in use by an ATM
     bcf      ATMassign,3             ; record in ATMassign<3-2> that Line#1 (=0) running ATM#B
     bcf      ATMassign,2
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x01                    ; ff=00 (Line#1) and gg=01 (ATM#B)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartBonTimer           ; start timer for ATM#B line seizure check
     goto     MainLoopATMC            ; goto Part C of the Main Loop
BToUseLine2
     bsf      servicereg,ATMB         ; record in servicereg that ATM#B is being serviced
     bsf      LIUreg,Line2            ; record in LIUreg that Line#2 is in use by an ATM
     bcf      ATMassign,3             ; record in ATMassign<3-2> that Line#2 (=1) running ATM#B
     bsf      ATMassign,2
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x05                    ; ff=01 (Line#2) and gg=01 (ATM#B)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartBonTimer           ; start timer for ATM#B line seizure check
     goto     MainLoopATMC            ; goto Part C of the Main Loop
BToUseLine3
     bsf      servicereg,ATMB         ; record in servicereg that ATM#B is being serviced
     bsf      LIUreg,Line3            ; record in LIUreg that Line#3 is in use by an ATM
     bsf      ATMassign,3             ; record in ATMassign<3-2> that Line#3 (=2) running ATM#B
     bcf      ATMassign,2
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
```

```
     movlw    0x09                    ; ff=10 (Line#3) and gg=01 (ATM#B)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartBonTimer           ; start timer for ATM#B line seizure check
     goto     MainLoopATMC            ; goto Part C of the Main Loop
BToUseLine4
     bsf      servicereg,ATMB         ; record in servicereg that ATM#B is being serviced
     bsf      LIUreg,Line4            ; record in LIUreg that Line#4 is in use by an ATM
     bsf      ATMassign,3             ; record in ATMassign<3-2> that Line#4 (=3) running ATM#B
     bsf      ATMassign,2
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x0D                    ; ff=11 (Line#4) and gg=01 (ATM#B)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartBonTimer           ; start timer for ATM#B line seizure check
     goto     MainLoopATMC            ; goto Part C of the Main Loop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to connect ATM#C to the external lines
;
CToUseLine1
     bsf      servicereg,ATMC         ; record in servicereg that ATM#C is being serviced
     bsf      LIUreg,Line1            ; record in LIUreg that Line#1 is in use by an ATM
     bcf      ATMassign,5             ; record in ATMassign<5-4> that Line#1 (=0) running ATM#C
     bcf      ATMassign,4
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x02                    ; ff=00 (Line#1) and gg=10 (ATM#C)
     movwf    Scommand2
     call     SendCommandToPIC2
     all      del500ms                ; wait one second for the connection to be madecall
     del500ms
     call     StartConTimer           ; start timer for ATM#C line seizure check
     goto     MainLoopATMD            ; goto Part D of the Main Loop
CToUseLine2
     bsf      servicereg,ATMC         ; record in servicereg that ATM#C is being serviced
     bsf      LIUreg,Line2            ; record in LIUreg that Line#2 is in use by an ATM
     bcf      ATMassign,5             ; record in ATMassign<5-4> that Line#2 (=1) running ATM#C
     bsf      ATMassign,4
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x06                    ; ff=01 (Line#2) and gg=10 (ATM#C)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartConTimer           ; start timer for ATM#C line seizure check
     goto     MainLoopATMD            ; goto Part D of the Main Loop
CToUseLine3
     bsf      servicereg,ATMC         ; record in servicereg that ATM#C is being serviced
     bsf      LIUreg,Line3            ; record in LIUreg that Line#3 is in use by an ATM
     bsf      ATMassign,5             ; record in ATMassign<5-4> that Line#3 (=2) running ATM#C
     bcf      ATMassign,4
     movlw    0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf    Scommand1
     movlw    0x0A                    ; ff=10 (Line#3) and gg=10 (ATM#C)
     movwf    Scommand2
     call     SendCommandToPIC2
     call     del500ms                ; wait one second for the connection to be made
     call     del500ms
     call     StartConTimer           ; start timer for ATM#C line seizure check
     goto     MainLoopATMD            ; goto Part D of the Main Loop
CToUseLine4
     bsf      servicereg,ATMC         ; record in servicereg that ATM#C is being serviced
     bsf      LIUreg,Line4            ; record in LIUreg that Line#4 is in use by an ATM
```

```
     bsf     ATMassign,5             ; record in ATMassign<5-4> that Line#4 (=3) running ATM#C
     bsf     ATMassign,4
     movlw   0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf   Scommand1
     movlw   0x0E                    ; ff=11 (Line#4) and gg=10 (ATM#C)
     movwf   Scommand2
     call    SendCommandToPIC2
     call    del500ms                ; wait one second for the connection to be made
     call    del500ms
     call    StartConTimer           ; start timer for ATM#C line seizure check
     goto    MainLoopATMD            ; goto Part D Of the Main Loop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to connect ATM#D to the external lines
;
DToUseLine1
     bsf     servicereg,ATMD         ; record in servicereg that ATM#D is being serviced
     bsf     LIUreg,Line1            ; record in LIUreg that Line#1 is in use by an ATM
     bcf     ATMassign,7             ; record in ATMassign<7-6> that Line#1 (=0) running ATM#D
     bcf     ATMassign,6
     movlw   0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf   Scommand1
     movlw   0x03                    ; ff=00 (Line#1) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     call    del500ms                ; wait one second for the connection to be made
     call    del500ms
     call    StartDonTimer           ; start timer for ATM#D line seizure check
     goto    MainLoop                ; restart Main Loop
DToUseLine2
     bsf     servicereg,ATMD         ; record in servicereg that ATM#D is being serviced
     bsf     LIUreg,Line2            ; record in LIUreg that Line#2 is in use by an ATM
     bcf     ATMassign,7             ; record in ATMassign<7-6> that Line#2 (=1) running ATM#D
     bsf     ATMassign,6
     movlw   0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf   Scommand1
     movlw   0x07                    ; ff=01 (Line#2) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     call    del500ms                ; wait one second for the connection to be made
     call    del500ms
     call    StartDonTimer           ; start timer for ATM#D line seizure check
     goto    MainLoop                ; restart Main Loop
DToUseLine3
     bsf     servicereg,ATMD         ; record in servicereg that ATM#D is being serviced
     bsf     LIUreg,Line3            ; record in LIUreg that Line#3 is in use by an ATM
     bsf     ATMassign,7             ; record in ATMassign<7-6> that Line#3 (=2) running ATM#D
     bcf     ATMassign,6
     movlw   0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf   Scommand1
     movlw   0x0B                    ; ff=10 (Line#3) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     call    del500ms                ; wait one second for the connection to be made
     call    del500ms
     call    StartDonTimer           ; start timer for ATM#D line seizure check
     goto    MainLoop                ; restart Main Loop
DToUseLine4
     bsf     servicereg,ATMD         ; record in servicereg that ATM#D is being serviced
     bsf     LIUreg,Line4            ; record in LIUreg that Line#4 is in use by an ATM
     bsf     ATMassign,7             ; record in ATMassign<7-6> that Line#4 (=3) running ATM#D
     bsf     ATMassign,6
     movlw   0x01                    ; 0x01 is the command to PIC#2 to make a connection
     movwf   Scommand1
     movlw   0x0F                    ; ff=11 (Line#4) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     call    del500ms                ; wait one second for the connection to be made
     call    del500ms
     call    StartDonTimer           ; start timer for ATM#D line seizure check
```

```
     goto    MainLoop                ; restart Main Loop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to disconnect ATM#A from the external lines
;
AWasUsingLine1
     movlw   0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x00                    ; ff=00 (Line#1) and gg=00 (ATM#A)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMA         ; record in servicereg that ATM#A is not being serviced
     bcf     LIUreg,Line1            ; record in LIUreg that Line#1 is not in use by an ATM
     bcf     ATMassign,1             ; reset ATMassign<1-0> to zero
     bcf     ATMassign,0
     call    del10ms                 ; wait 10ms for circuits to settle.  Note that
                                     ; the Bell system has already disconnected.
     call    StopAonTimer            ; stop timer for ATM#A line seizure check
     call    StartAoffTimer          ; start timer for ATM#A out-of-service requirement
     goto    MainLoopATMB            ; goto Part B of the Main Loop
AWasUsingLine2
     movlw   0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x04                    ; ff=01 (Line#2) and gg=00 (ATM#A)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMA         ; record in servicereg that ATM#A is not being serviced
     bcf     LIUreg,Line2            ; record in LIUreg that Line#2 is not in use by an ATM
     bcf     ATMassign,1             ; reset ATMassign<1-0> to zero
     bcf     ATMassign,0
     call    del10ms                 ; wait 10ms for circuits to settle
     call    StopAonTimer            ; stop timer for ATM#A line seizure check
     call    StartAoffTimer          ; start timer for ATM#A out-of-service requirement
     goto    MainLoopATMB            ; goto Part B of the Main Loop
AWasUsingLine3
     movlw   0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x08                    ; ff=10 (Line#3) and gg=00 (ATM#A)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMA         ; record in servicereg that ATM#A is not being serviced
     bcf     LIUreg,Line3            ; record in LIUreg that Line#3 is not in use by an ATM
     bcf     ATMassign,1             ; reset ATMassign<1-0> to zero
     bcf     ATMassign,0
     call    del10ms                 ; wait 10ms for circuits to settle
     call    StopAonTimer            ; stop timer for ATM#A line seizure check
     call    StartAoffTimer          ; start timer for ATM#A out-of-service requirement
     goto    MainLoopATMB            ; goto Part B of the Main Loop
AWasUsingLine4
     movlw   0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x0C                    ; ff=11 (Line#4) and gg=00 (ATM#A)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMA         ; record in servicereg that ATM#A is not being serviced
     bcf     LIUreg,Line4            ; record in LIUreg that Line#4 is not in use by an ATM
     bcf     ATMassign,1             ; reset ATMassign<1-0> to zero
     bcf     ATMassign,0
     call    del10ms                 ; wait 10ms for circuits to settle
     call    StopAonTimer            ; stop timer for ATM#A line seizure check
     call    StartAoffTimer          ; start timer for ATM#A out-of-service requirement
     goto    MainLoopATMB            ; goto Part B of the Main Loop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to disconnect ATM#B from the external lines
;
BWasUsingLine1
     movlw   0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
```

```
     movlw    0x01                    ; ff=00 (Line#1) and gg=01 (ATM#B)
     movwf    Scommand2
     call     SendCommandToPIC2
     bcf      servicereg,ATMB         ; record in servicereg that ATM#B is not being serviced
     bcf      LIUreg,Line1            ; record in LIUreg that Line#1 is not in use by an ATM
     bcf      ATMassign,3             ; reset ATMassign<3-2> to zero
     bcf      ATMassign,2
     call     del10ms                 ; wait 10ms for circuits to settle
     call     StopBonTimer            ; stop timer for ATM#B line seizure check
     call     StartBoffTimer          ; start timer for ATM#B out-of-service requirement
     goto     MainLoopATMC            ; goto Part C of the Main Loop
BWasUsingLine2
     movlw    0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf    Scommand1
     movlw    0x05                    ; ff=01 (Line#2) and gg=01 (ATM#B)
     movwf    Scommand2
     call     SendCommandToPIC2
     bcf      servicereg,ATMB         ; record in servicereg that ATM#B is not being serviced
     bcf      LIUreg,Line2            ; record in LIUreg that Line#2 is not in use by an ATM
     bcf      ATMassign,3             ; reset ATMassign<3-2> to zero
     bcf      ATMassign,2
     call     del10ms                 ; wait 10ms for circuits to settle
     call     StopBonTimer            ; stop timer for ATM#B line seizure check
     call     StartBoffTimer          ; start timer for ATM#B out-of-service requirement
     goto     MainLoopATMC            ; goto Part C of the Main Loop
BWasUsingLine3
     movlw    0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf    Scommand1
     movlw    0x09                    ; ff=10 (Line #3) and gg=01 (ATM #B)
     movwf    Scommand2
     call     SendCommandToPIC2
     bcf      servicereg,ATMB         ; record in servicereg that ATM#B is not being serviced
     bcf      LIUreg,Line3            ; record in LIUreg that Line#3 is not in use by an ATM
     bcf      ATMassign,3             ; reset ATMassign<3-2> to zero
     bcf      ATMassign,2
     call     del10ms                 ; wait 10ms for circuits to settle
     call     StopBonTimer            ; stop timer for ATM#B line seizure check
     call     StartBoffTimer          ; start timer for ATM#B out-of-service requirement
     goto     MainLoopATMC            ; goto Part C of the Main Loop
BWasUsingLine4
     movlw    0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf    Scommand1
     movlw    0x0D                    ; ff=11 (Line#4) and gg=01 (ATM#B)
     movwf    Scommand2
     call     SendCommandToPIC2
     bcf      servicereg,ATMB         ; record in servicereg that ATM#B is not being serviced
     bcf      LIUreg,Line4            ; record in LIUreg that Line#4 is not in use by an ATM
     bcf      ATMassign,3             ; reset ATMassign<3-2> to zero
     bcf      ATMassign,2
     call     del10ms                 ; wait 10ms for circuits to settle
     call     StopBonTimer            ; stop timer for ATM#B line seizure check
     call     StartBoffTimer          ; start timer for ATM#B out-of-service requirement
     goto     MainLoopATMC            ; goto Part C of the Main Loop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to disconnect ATM#C from the external lines
;
CWasUsingLine1
     movlw    0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf    Scommand1
     movlw    0x02                    ; ff=00 (Line#1) and gg=10 (ATM#C)
     movwf    Scommand2
     call     SendCommandToPIC2
     bcf      servicereg,ATMC         ; record in servicereg that ATM#C is not being serviced
     bcf      LIUreg,Line1            ; record in LIUreg that Line#1 is not in use by an ATM
     bcf      ATMassign,5             ; reset ATMassign<5-4> to zero
     bcf      ATMassign,4
     call     del10ms                 ; wait 10ms for circuits to settle
     call     StopConTimer            ; stop timer for ATM#C line seizure check
     call     StartCoffTimer          ; start timer for ATM#C out-of-service requirement
```

```
     goto    MainLoopATMD           ; goto Part D of the Main Loop
CWasUsingLine2
     movlw   0x02                   ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x06                   ; ff=01 (Line#2) and gg=10 (ATM#C)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMC        ; record in servicereg that ATM#C is not being serviced
     bcf     LIUreg,Line2           ; record in LIUreg that Line#2 is not in use by an ATM
     bcf     ATMassign,5            ; reset ATMassign<5-4> to zero
     bcf     ATMassign,4
     call    del10ms                ; wait 10ms for circuits to settle
     call    StopConTimer           ; stop timer for ATM#C line seizure check
     call    StartCoffTimer         ; start timer for ATM#C out-of-service requirement
     goto    MainLoopATMD           ; goto Part D of the Main Loop
CWasUsingLine3
     movlw   0x02                   ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x0A                   ; ff=10 (Line#3) and gg=10 (ATM#C)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMC        ; record in servicereg that ATM#C is not being serviced
     bcf     LIUreg,Line3           ; record in LIUreg that Line#3 is not in use by an ATM
     bcf     ATMassign,5            ; reset ATMassign<5-4> to zero
     bcf     ATMassign,4
     call    del10ms                ; wait 10ms for circuits to settle
     call    StopConTimer           ; stop timer for ATM#C line seizure check
     call    StartCoffTimer         ; start timer for ATM#C out-of-service requirement
     goto    MainLoopATMD           ; goto Part D of the Main Loop
CWasUsingLine4
     movlw   0x02                   ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x0E                   ; ff=11 (Line#4) and gg=10 (ATM#C)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMC        ; record in servicereg that ATM#C is not being serviced
     bcf     LIUreg,Line4           ; record in LIUreg that Line#4 is not in use by an ATM
     bcf     ATMassign,5            ; reset ATMassign<5-4> to zero
     bcf     ATMassign,4
     call    del10ms                ; wait 10ms for circuits to settle
     call    StopConTimer           ; stop timer for ATM#C line seizure check
     call    StartCoffTimer         ; start timer for ATM#C out-of-service requirement
     goto    MainLoopATMD           ; goto Part D of the Main Loop
;
; ********************************************************************************************
; ********************************************************************************************
; Procedures to disconnect ATM#D from the external lines
;
DWasUsingLine1
     movlw   0x02                   ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x03                   ; ff=00 (Line#1) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMD        ; record in servicereg that ATM#D is not being serviced
     bcf     LIUreg,Line1           ; record in LIUreg that Line#1 is not in use by an ATM
     bcf     ATMassign,7            ; reset ATMassign<7-6> to zero
     bcf     ATMassign,6
     call    del10ms                ; wait 10ms for circuits to settle
     call    StopDonTimer           ; stop timer for ATM#D line seizure check
     call    StartDoffTimer         ; start timer for ATM#D out-of-service requirement
     goto    MainLoop               ; restart Main Loop
DWasUsingLine2
     movlw   0x02                   ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x07                   ; ff=01 (Line#2) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMD        ; record in servicereg that ATM#D is not being serviced
     bcf     LIUreg,Line2           ; record in LIUreg that Line#2 is not in use by an ATM
     bcf     ATMassign,7            ; reset ATMassign<7-6> to zero
```

```
     bcf     ATMassign,6
     call    del10ms                 ; wait 10ms for circuits to settle
     call    StopDonTimer            ; stop timer for ATM#D line seizure check
     call    StartDoffTimer          ; start timer for ATM#D out-of-service requirement
     goto    MainLoop                ; restart Main Loop
DWasUsingLine3
     movlw   0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x0B                    ; ff=10 (Line#3) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMD         ; record in servicereg that ATM#D is not being serviced
     bcf     LIUreg,Line3            ; record in LIUreg that Line#3 is not in use by an ATM
     bcf     ATMassign,7             ; reset ATMassign<7-6> to zero
     bcf     ATMassign,6
     call    del10ms                 ; wait 10ms for circuits to settle
     call    StopDonTimer            ; stop timer for ATM#D line seizure check
     call    StartDoffTimer          ; start timer for ATM#D out-of-service requirement
     goto    MainLoop                ; restart Main Loop
DWasUsingLine4
     movlw   0x02                    ; 0x02 is the command to PIC#2 to break a connection
     movwf   Scommand1
     movlw   0x0F                    ; ff=11 (Line#4) and gg=11 (ATM#D)
     movwf   Scommand2
     call    SendCommandToPIC2
     bcf     servicereg,ATMD         ; record in servicereg that ATM#D is not being serviced
     bcf     LIUreg,Line4            ; record in LIUreg that Line#4 is not in use by an ATM
     bcf     ATMassign,7             ; reset ATMassign<7-6> to zero
     bcf     ATMassign,6
     call    del10ms                 ; wait 10ms for circuits to settle
     call    StopDonTimer            ; stop timer for ATM#D line seizure check
     call    StartDoffTimer          ; start timer for ATM#D out-of-service requirement
     goto    MainLoop                ; restart Main Loop
;
; ********************************************************************************************
; Index of subroutines
;    del500ms
;    del10ms
;    del1ms
;    del100us
;    GetATMAStatus - returns status (1=offhook) of ATM#A in offhookreg<ATMA>
;    GetATMBStatus - returns status (1=offhook) of ATM#B in offhookreg<ATMB>
;    GetATMCStatus - returns status (1=offhook) of ATM#C in offhookreg<ATMC>
;    GetATMDStatus - returns status (1=offhook) of ATM#D in offhookreg<ATMD>
;    GetLine1Status - returns status (1=in use) of external line #1 in extlinereg<Line1>
;    GetLine2Status - returns status (1=in use) of external line #2 in extlinereg<Line2>
;    GetLine3Status - returns status (1=in use) of external line #3 in extlinereg<Line3>
;    GetLine4Status - returns status (1=in use) of external line #4 in extlinereg<Line4>
;    GetFirstAvailableLine - returns first available external line (0-3) or h'F' if none
;    SendCommandToPIC2 - sends a two-nibble command to PIC#2
;    StartTOTimer - starts timing a 25ms communication timeout with PIC#2
;    StartAonMaxTimer - timers which check for 5-minute line seizure
;    StopAonMaxTimer
;    StartBonTimer
;    StopBonTimer
;    StartConTimer
;    StopConTimer
;    StartDonTimer
;    StopDonTimer
;    StartAoffTimer - timers which keep ATMs out-of-service for 3 seconds
;    StopAoffTimer
;    StartBoffTimer
;    StopBoffTimer
;    StartCoffTimer
;    StopCoffTimer
;    StartDoffTimer
;    StopDoffTimer
;    TurnOnErrorLED
;    TurnOffErrorLED
; ********************************************************************************************
```

```
;
; ********************************************************************************************
; Subroutine del500ms delays for 503.082ms
;                                   ; Cycles
del500ms
     movlw    0x32                  ; 1 (Note that 0x32 = d'50'.)
     movwf    count3                ; 1
del500msa
     call     del10ms               ; 50 x 25,151 = 1,257,550
     decfsz   count3,f              ; (49 x 1) + (1 x 2) = 51
     goto     del500msa             ; 49 x 2 = 98
     return                         ; 2
;                                   ; CALL adds2 -- 1,257,705 cycles x 400ns = 503,082,000ns
;
; ********************************************************************************************
; Subroutine del10ms delays for 10.0604ms
;                                   ; Cycles
del10ms
     movlw    0x63                  ; 1 (Note that 0x63 = d'99'.)
     movwf    count2                ; 1
del10msa
     call     del100us              ; 99 x 251 = 24,849
     decfsz   count2,f              ; (98 x 1) + (1 x 2) = 100
     goto     del10msa              ; 98 x 2 = 196
     return                         ; 2
;                                   ; CALL adds 2 -- 25,151 cycles x 400ns = 10,060,400ns
;
; ********************************************************************************************
; Subroutine del1ms delays for 1.018ms
;                                   ; Cycles
del1ms
     movlw    0x0A                  ; 1 (Note that 0x0A = d'10'.)
     movwf    count2                ; 1
del1msa
     call     del100us              ; 10 x 251 = 2,510
     decfsz   count2,f              ; (9 x 1) + (1 x 2) = 11
     goto     del1msa               ; 9 x 2 = 18
     return                         ; 2
;                                   ; CALL adds 2 -- 2,545 cycles x 400ns = 1,018,000ns
;
; ********************************************************************************************
; Subroutine del100us delays for 100.4us
;                                   ; Cycles
del100us
     movlw    0x52                  ; 1 -- (Note that 0x52 = d'82'.)
     movwf    count1                ; 1
del100usa
     decfsz   count1,f              ; (81 x 1) + (1 x 2) = 83
     goto     del100usa             ; 81 x 2 = 162
     return                         ; 2
;                                   ; CALL adds 2 -- 251 cycles x 400ns = 100,400ns
;
; ********************************************************************************************
; Subroutine GetATMAStatus determines the offhook status of ATM#A.  The result is
; placed in register offhookreg<ATMA>.  The bit is set high if ATM#A is offhook,
; asserted low if it is onhook.
GetATMAStatus
     movf     portC,w               ; read portC and ...
     movwf    temp                  ; ... save in register temp
     btfss    temp,hookstatusA      ; portC<hookstatusA> is low if ATM#A is offhook
     goto     GATMASOffhook
     goto     GATMASOnhook
GATMASOffhook
     bsf      offhookreg,ATMA
     return
GATMASOnhook
     bcf      offhookreg,ATMA
     return
```

```
;
; ********************************************************************************************
; Subroutine GetATMBStatus determines the offhook status of ATM#B.  The result is
; placed in register offhookreg<ATMB>.  The bit is set high if ATM#B is offhook,
; asserted low if it is onhook.
GetATMBStatus
     movf    portC,w                 ; read portC and ...
     movwf   temp                    ; ... save in register temp
     btfss   temp,hookstatusB        ; portC<hookstatusB> is low if ATM#B is offhook
     goto    GATMBSOffhook
     goto    GATMBSOnhook
GATMBSOffhook
     bsf     offhookreg,ATMB
     return
GATMBSOnhook
     bcf     offhookreg,ATMB
     return
;
; ********************************************************************************************
; Subroutine GetATMCStatus determines the offhook status of ATM#C.  The result is
; placed in register offhookreg<ATMC>.  The bit is set high if ATM#C is offhook,
; asserted low if it is onhook.
GetATMCStatus
     movf    portB,w                 ; read portB and ...
     movwf   temp                    ; ... save in register temp
     btfss   temp,hookstatusC        ; portB<hookstatusC> is low if ATM#C is offhook
     goto    GATMCSOffhook
     goto    GATMCSOnhook
GATMCSOffhook
     bsf     offhookreg,ATMC
     return
GATMCSOnhook
     bcf     offhookreg,ATMC
     return
;
; ********************************************************************************************
; Subroutine GetATMDStatus determines the offhook status of ATM#D.  The result is
; placed in register offhookreg<ATMD>.  The bit is set high if ATM#D is offhook,
; asserted low if it is onhook.
GetATMDStatus
     movf    portB,w                 ; read portB and ...
     movwf   temp                    ; ... save in register temp
     btfss   temp,hookstatusD        ; portB<hookstatusD> is low if ATM#D is offhook
     goto    GATMDSOffhook
     goto    GATMDSOnhook
GATMDSOffhook
     bsf     offhookreg,ATMD
     return
GATMDSOnhook
     bcf     offhookreg,ATMD
     return
;
; ********************************************************************************************
; Subroutine GetLine1Status activates the TS117 multi-function relay for external
; Line #1.  The result is placed in bit extlinereg<Line1>.  The bit is asserted low if
; the line is available, set high if it is in use.
GetLine1Status
     bcf     portBmirror,CL1         ; assert low the CL for external Line #1
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
     movf    portC,w                 ; read portC and ...
     movwf   temp                    ; ... save in register temp
     bsf     portBmirror,CL1         ; set high the CL for external Line #1
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
     btfss   temp,LIU1               ; check status of Line #1 (low if available)
     goto    GL1SAvailable
     goto    GL1SNotAvailable
GL1SAvailable
```

```
     bcf     extlinereg,Line1
     return
GL1SNotAvailable
     bsf     extlinereg,Line1
     return
;
; ********************************************************************************************
; Subroutine GetLine2Status activates the TS117 multi-function relay for external
; Line #2.  The result is placed in bit extlinereg<Line2>.  The bit is asserted low if
; the line is available, set high if it is in use.
GetLine2Status
     bcf     portBmirror,CL2         ; assert low the CL for external Line #2
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
     movf    portC,w                 ; read portC and ...
     movwf   temp                    ; ... save in register temp
     bsf     portBmirror,CL2         ; set high the CL for external Line #2
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
     btfss   temp,LIU2               ; check status of Line #2 (low if available)
     goto    GL2SAvailable
     goto    GL2SNotAvailable
GL2SAvailable
     bcf     extlinereg,Line2
     return
GL2SNotAvailable
     bsf     extlinereg,Line2
     return
;
; ********************************************************************************************
; Subroutine GetLine3Status activates the TS117 multi-function relay for external
; Line #3.  The result is placed in bit extlinereg<Line3>.  The bit is asserted low if
; the line is available, set high if it is in use.
GetLine3Status
     bcf     portBmirror,CL3         ; assert low the CL for external Line #3
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
     movf    portC,w                 ; read portC and ...
     movwf   temp                    ; ... save in register temp
     bsf     portBmirror,CL3         ; set high the CL for external Line #3
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
     btfss   temp,LIU3               ; check status of Line #3 (low if available)
     goto    GL3SAvailable
     goto    GL3SNotAvailable
GL3SAvailable
     bcf     extlinereg,Line3
     return
GL3SNotAvailable
     bsf     extlinereg,Line3
     return
;
; ********************************************************************************************
; Subroutine GetLine4Status activates the TS117 multi-function relay for external
; Line #4.  The result is placed in bit extlinereg<Line4>.  The bit is asserted low if
; the line is available, set high if it is in use.
GetLine4Status
     bcf     portBmirror,CL4         ; assert low the CL for external Line #4
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
     movf    portC,w                 ; read portC and ...
     movwf   temp                    ; ... save in register temp
     bsf     portBmirror,CL4         ; set high the CL for external Line #4
     movf    portBmirror,w
     movwf   portB
     call    del10ms                 ; wait 10ms
```

```
     btfss    temp,LIU4               ; check status of Line #4 (low if available)
     goto     GL4SAvailable
     goto     GL4SNotAvailable
GL4SAvailable
     bcf      extlinereg,Line4
     return
GL4SNotAvailable
     bsf      extlinereg,Line4
     return
;
; ********************************************************************************************
; Subroutine GetFirstAvailableLine checks the external lines one-by-one until it finds one
; which is available for use by an ATM.  Two principles govern the search.
; 1. The lines are polled in the order Line #1, Line #2, Line #3 and, lastly, Line #4.
; 2. Lines which are already in use by an ATM (the LIUreg bit is high) are not polled.
; The result is returned in register w.  As always, the external lines are referred to by
; the numbers 0 through 3.  If no external line is available, this procedure returns h'F'.
GetFirstAvailableLine
GFALCheck1
     btfsc    LIUreg,Line1
     goto     GFALCheck2              ; Line #1 is already in use (LIU bit is high)
     call     GetLine1Status
     btfsc    extlinereg,Line1
     goto     GFALCheck2              ; Line #1 is in use by something else (Line1 bit is high)
     movlw    0x00                    ; use Line #1
     return
GFALCheck2
     btfsc    LIUreg,Line2
     goto     GFALCheck3              ; Line #2 is already in use (LIU bit is high)
     call     GetLine2Status
     btfsc    extlinereg,Line2
     goto     GFALCheck3              ; Line #2 is in use by something else (Line2 bit is high)
     movlw    0x01                    ; use Line #2
     return
GFALCheck3
     btfsc    LIUreg,Line3
     goto     GFALCheck4              ; Line #3 is already in use (LIU bit is high)
     call     GetLine3Status
     btfsc    extlinereg,Line3
     goto     GFALCheck4              ; Line #3 is in use by something else (Line3 bit is high)
     movlw    0x02                    ; use Line #3
     return
GFALCheck4
     btfsc    LIUreg,Line4
     goto     GFALNoLines             ; Line #4 is already in use (LIU bit is high)
     call     GetLine4Status
     btfsc    extlinereg,Line4
     goto     GFALNoLines             ; Line #4 is in use by something else (Line4 bit is high)
     movlw    0x03                    ; use Line #4
     return
GFALNoLines
     movlw    0x0F
     return
;
; ********************************************************************************************
; Subroutine SendCommandToPIC2 sends a two-nibble command to PIC#2
; Called with:-
;   data to send is in register Scommand1<3-0> and Scommand2<3,0>
;   RB0 interrupts disabled
;   interrupt request line (RB0) low
;   interrupt acknowledge line (RB1) presumed low, if PIC#2 is ready
;   data bus portA<3-0> configured for output
; Returns with:-
;   RB0 interrupts disabled
;   interrupt request line (RB0) low
;   interrupt acknowledge line (RB1) low
;   data bus portA<3-0> configured for output
;
SendCommandToPIC2
     bsf      portBmirror,ACK         ; interrupt PIC#2 by setting ACK line high
     movf     portBmirror,w
```

```
     movwf   portB
     call    StartTOTimer            ; begin 25ms communication timeout test
SCP2A
     movf    counterTO,w             ; check value of register counterTO
     btfsc   status,zero             ; if counterTO counts to zero, then the zero flag ...
     goto    HardReset               ; ... is set and the communication has timed out
     movf    portB,w                 ; read INT bit (do not use direct bit read)
     movwf   tempSCP2
     btfss   tempSCP2,INT            ; wait until PIC#2 acknowledges by setting INT line high
     goto    SCP2A
     movf    portAmirror,w
     andlw   0x30                    ; keep only the two high bits of portA
     iorwf   Scommand1,w             ; or-in the four low bits from register Scommand1
     movwf   portAmirror             ; save changes in portAmirror
     movwf   portA                   ; also write to portA
     bcf     portBmirror,ACK         ; request end-of-communication by asserting ACK line low
     movf    portBmirror,w
     movwf   portB
SCP2B
     movf    counterTO,w             ; check value of register counterTO
     btfsc   status,zero             ; if counterTO counts to zero, then the zero flag ...
     goto    HardReset               ; ... is set and the communication has timed out
     movf    portB,w                 ; read INT bit (do not use direct bit read)
     movwf   tempSCP2
     btfsc   tempSCP2,INT            ; wait until PIC#2 ends comm by asserting INT line low
     goto    SCP2B
     call    del100us                ; wait 100us before sending the second nibble
     bsf     portBmirror,ACK         ; interrupt PIC#2 by setting ACK line high
     movf    portBmirror,w
     movwf   portB
SCP2C
     movf    counterTO,w             ; check value of register counterTO
     btfsc   status,zero             ; if counterTO counts to zero, then the zero flag ...
     goto    HardReset               ; ... is set and the communication has timed out
     movf    portB,w                 ; read INT bit (do not use direct bit read)
     movwf   tempSCP2
     btfss   tempSCP2,INT            ; wait until PIC#2 acknowledges by setting INT line high
     goto    SCP2C
     movf    portAmirror,w
     andlw   0x30                    ; keep only the two high bits of portA
     iorwf   Scommand2,w             ; or-in the four low bits from register Scommand2
     movwf   portAmirror             ; save changes in portAmirror
     movwf   portA                   ; also write to portA
     bcf     portBmirror,ACK         ; request end-of-communication by asserting ACK line low
     movf    portBmirror,w
     movwf   portB
SCP2D
     movf    counterTO,w             ; check value of register counterTO
     btfsc   status,zero             ; if counterTO counts to zero, then the zero flag ...
     goto    HardReset               ; ... is set and the communication has timed out
     movf    portB,w                 ; read INT bit (do not use direct bit read)
     movwf   tempSCP2
     btfsc   tempSCP2,INT            ; wait until PIC#2 ends comm by asserting INT line low
     goto    SCP2D
     return
;
; ********************************************************************************************
; Subroutine StartTOTimer initializes register counterTO to detect a
; communication timeout.
StartTOTimer
     movlw   0x02                    ; 0x02=d'2'
     movwf   counterTO
     clrf    timer0                  ; reset timer0
     return
;
; ********************************************************************************************
; Subroutine StartAonTimer initializes register pair <counterAH, counterAL> to detect
; when ATM #A seizes an external line.
StartAonTimer
     movlw   0x2D                    ; 0x2D=d'45'
     movwf   counterAH
```

```
     clrf     counterAL
     bsf      counterAon,0
     return
StopAonTimer
     clrf     counterAon
     return
;
; ********************************************************************************************
; Subroutine StartBonTimer initializes register pair <counterBH, counterBL> to detect
; when ATM #B seizes an external line.
StartBonTimer
     movlw    0x2D                    ; 0x2D=d'45'
     movwf    counterBH
     clrf     counterBL
     bsf      counterBon,0
     return
StopBonTimer
     clrf     counterBon
     return
;
; ********************************************************************************************
; Subroutine StartConTimer initializes register pair <counterCH, counterCL> to detect
; when ATM #C seizes an external line.
StartConTimer
     movlw    0x2D                    ; 0x2D=d'45'
     movwf    counterCH
     clrf     counterCL
     bsf      counterCon,0
     return
StopConTimer
     clrf     counterCon
     return
;
; ********************************************************************************************
; Subroutine StartDonTimer initializes register pair <counterDH, counterDL> to detect
; when ATM #D seizes an external line.
StartDonTimer
     movlw    0x2D                    ; 0x2D=d'45'
     movwf    counterDH
     clrf     counterDL
     bsf      counterDon,0
     return
StopDonTimer
     clrf     counterDon
     return
;
; ********************************************************************************************
; Subroutine StartAoffTimer initializes register pair <counterAH, counterAL> to detect
; when ATM #A should be in its out-of-service period.
StartAoffTimer
     clrf     counterAH
     movlw    0x73                    ; 0x73=d'115'
     movwf    counterAL
     bsf      counterAoff,0
     return
StopAoffTimer
     clrf     counterAoff
     return
;
; ********************************************************************************************
; Subroutine StartBoffTimer initializes register pair <counterBH, counterBL> to detect
; when ATM #B should be in its out-of-service period.
StartBoffTimer
     clrf     counterBH
     movlw    0x73                    ; 0x73=d'115'
     movwf    counterBL
     bsf      counterBoff,0
     return
StopBoffTimer
     clrf     counterBoff
     return
```

```
;
; ********************************************************************************************
; Subroutine StartCoffTimer initializes register pair <counterCH, counterCL> to detect
; when ATM #C should be in its out-of-service period.
StartCoffTimer
     clrf     counterCH
     movlw    0x73                    ; 0x73=d'115'
     movwf    counterCL
     bsf      counterCoff,0
     return
StopCoffTimer
     clrf     counterCoff
     return
;
; ********************************************************************************************
; Subroutine StartDoffTimer initializes register pair <counterDH, counterDL> to detect
; when ATM #D should be in its out-of-service period.
StartDoffTimer
     clrf     counterDH
     movlw    0x73                    ; 0x73=d'115'
     movwf    counterDL
     bsf      counterDoff,0
     return
StopDoffTimer
     clrf     counterDoff
     return
;
; ********************************************************************************************
; Subroutine TurnOnErrorLED turns on the Error Condition LED.
TurnOnErrorLED
     bcf      portAmirror,Err         ; LED is active low
     movf     portAmirror,w
     movwf    portA
     return
;
; ********************************************************************************************
; Subroutine TurnOffErrorLED turns off the Error Condition LED.
TurnOffErrorLED
     bsf      portAmirror,Err         ; LED is active low
     movf     portAmirror,w
     movwf    portA
     return
;
; ********************************************************************************************
;
; Test programs used for debugging purposes only
;
; ********************************************************************************************
;
; TestProgram1 turns the ErrorLED on and off every 500ms.
TestProgram1
     bcf      INTCon,GIE              ; disable interrupts - TestProgram1 does not use interrupts
TP1A
     call     TurnOnErrorLED
     call     del500ms
     call     TurnOffErrorLED
     call     del500ms
     goto     TP1A
;
; ********************************************************************************************
;
; TestProgram2 cycles through the input hookstatus lines of the four ATMs, spending 500ms
; on each.  If the ATM is off hook (hookstatus line low), the ErrorLED is turned on for
; the 500ms interval.
TestProgram2
     bcf      INTCon,GIE              ; disable interrupts - TestProgram2 does not use interrupts
TP2A
     movf     portC,w
     movwf    tempTest
     call     TurnOffErrorLED
     btfss    tempTest,hookstatusA
```

```
     call     TurnOnErrorLED
     call     del500ms
     movf     portC,w
     movwf    tempTest
     call     TurnOffErrorLED
     btfss    tempTest,hookstatusB
     call     TurnOnErrorLED
     call     del500ms
     movf     portB,w
     movwf    tempTest
     call     TurnOffErrorLED
     btfss    tempTest,hookstatusC
     call     TurnOnErrorLED
     call     del500ms
     movf     portB,w
     movwf    tempTest
     call     TurnOffErrorLED
     btfss    tempTest,hookstatusD
     call     TurnOnErrorLED
     call     del500ms
     goto     TP2A
;
; ********************************************************************************************
;
; TestProgram3 cycles through the input line-in-use lines of the four external telephone
; lines, spending 500ms on each.  If the external line is available (line-in-use line low),
; the ErrorLED is turned on for the 500ms interval.
TestProgram3
     bcf      INTCon,GIE              ; disable interrupts - TestProgram3 does not use interrupts
TP3A
     bcf      portBmirror,CL1         ; assert low the CL for external Line #1
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     movf     portC,w                 ; read portC and ...
     movwf    tempTest                ; ... save in register temp
     bsf      portBmirror,CL1         ; set high the CL for external Line #1
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     call     TurnOffErrorLED
     btfss    tempTest,LIU1
     call     TurnOnErrorLED
     call     del500ms
     bcf      portBmirror,CL2         ; assert low the CL for external Line #2
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     movf     portC,w                 ; read portC and ...
     movwf    tempTest                ; ... save in register temp
     bsf      portBmirror,CL2         ; set high the CL for external Line #2
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     call     TurnOffErrorLED
     btfss    tempTest,LIU2
     call     TurnOnErrorLED
     call     del500ms
     bcf      portBmirror,CL3         ; assert low the CL for external Line #3
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     movf     portC,w                 ; read portC and ...
     movwf    tempTest                ; ... save in register temp
     bsf      portBmirror,CL3         ; set high the CL for external Line #3
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     call     TurnOffErrorLED
     btfss    tempTest,LIU3
     call     TurnOnErrorLED
```

```
     call     del500ms
     bcf      portBmirror,CL4         ; assert low the CL for external Line #4
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     movf     portC,w                 ; read portC and ...
     movwf    tempTest                ; ... save in register temp
     bsf      portBmirror,CL4         ; set high the CL for external Line #4
     movf     portBmirror,w
     movwf    portB
     call     del10ms                 ; wait 10ms
     call     TurnOffErrorLED
     btfss    tempTest,LIU4
     call     TurnOnErrorLED
     call     del500ms
     goto     TP3A
;
; ********************************************************************************************
;
; TestProgram4 tests interrupts from PIC#1 to PIC#2.  For each of the 16 combinations of
; Scommand1 = d'0' through d'16', PIC#1 sends the 16 combinations of Scommand2 = d'0' through
; d'16'.  PIC#2 should be running its TestProgram6, which displays the two nibbles as received.
; (Note that PIC#2 should not be running its main program, which would not recognize the
; first nibble as a command and would not, in any event, be able to make multiple
; connections.)  Each nibble pair is sent 500ms after its predecessor.  For convenience
; in programming below, the numeric sequences are actually sent in reverse order, as
; "countdowns".
TestProgram4
     bcf      INTCon,GIE
     movlw    0x0F
     movwf    Scommand1               ; Scommand1 starts as b'00001111'
     movlw    0x0F
     movwf    Scommand2               ; Scommand2 starts as b'00001111'
TP4A
     call     SendCommandToPIC2
     call     del500ms
     movf     Scommand2,w             ; set the zero status flag if register Scommand2=0
     btfsc    status,zero
     goto     TP4B                    ; after Scommand2=d'0' is sent, goto TP4B
     decf     Scommand2,f
     goto     TP4A
TP4B
     movf     Scommand1,w             ; set the zero status flag if register Scommand1=0
     btfsc    status,zero
     goto     TestProgram4            ; after Scommand1=d'0' is sent, restart TestProgram4
     decf     Scommand1,f
     movlw    0x0F
     movwf    Scommand2               ; reset Scommand2 to b'00001111' and ...
     goto     TP4A                    ; ... restart the TP4A Scommand2 loop
;
; TestProgram5 is very similar to TestProgram4 but, in TestProgram5, only the 16 valid
; connection pairs of ATMs to external lines are sent by PIC#1 to PIC#2.  The data
; is sent in the normal interrupt format, with Scommand1 being d'0' or d'1' to make
; or break a connection, respectively, and with Scommand2 being b'ffgg', where b'ff'
; is the selected external line and b'gg' is the selected ATM.  PIC#2 should be running
; its TestProgram7, which makes the connections as instructed.  (Note that PIC#2 should
; not be running its main program, which would require that the external lines be available.)
; Each nibble pair is sent one second after its predecessor.  For convenience in programming,
; the numeric sequences are actually sent in reverse order, as "countdowns".
; This test program, together with PIC#2 running its TestProgram7, are useful for
; observing the quality of signal transmission bewteen the ATM input connector (J1) and
; the external line connector (J2).
TestProgram5
     bcf      INTCon,GIE
     movlw    0x03
     movwf    tempTest5A              ; tempTest5A, for external lines, starts as b'00000011'
     movlw    0x03
     movwf    tempTest5B              ; tempTest5B, for ATMs, starts as b'00000011'
TP5A
     movf     tempTest5A,w
     movwf    tempTest                ; construct Scommand2 in register tempTest
```

```
     rlf     tempTest,f             ; rotate the external line bits into the appropriate places
     rlf     tempTest,w
     andlw   0x0C                   ; keep only the two external line bits
     iorwf   tempTest5B,w           ; or-in the two ATM bits
     movwf   Scommand2              ; save the connection pair in register Scommand2
     movlw   0x01
     movwf   Scommand1
     call    SendCommandToPIC2      ; instruct PIC#2 to make the connection
     call    del500ms               ; hold the connection for one second
     call    del500ms
     movlw   0x02
     movwf   Scommand1
     call    SendCommandToPIC2      ; instruct PIC#2 to break the connection
     call    del10ms                ; hold the disconnection for 10ms
     movf    tempTest5B,w           ; set the zero status flag if tempTest5B=0 has been sent
     btfsc   status,zero
     goto    TP5B                   ; after tempTest5B=0 is sent, goto TP5B
     decf    tempTest5B,f
     goto    TP5A
TP5B
     movf    tempTest5A,w           ; set the zero status flag if tempTest5A=0 has been sent
     btfsc   status,zero
     goto    TestProgram5           ; after tempTest5A=0 is sent, restart TestProgram5
     decf    tempTest5A,f
     movlw   0x03
     movwf   tempTest5B             ; reset tempTest5B to b'00000011' and ...
     goto    TP5A                   ; ... restart the TP5A Scommand2 loop

     END                            ; end assembly
```

```
; ATM Consolidator Program for PIC#2 (ATMPIC#2, compiles to ATMPIC#2.HEX)
;
; All communications from PIC#1 to PIC#2 take the form of two nibbles, sent one immediately
; after the other, and received by PIC#2 in the same interrupt service routine.  The first
; nibble contains an instruction code; the second nibble carries data.  PIC#2 does not send
; any information to PIC#1.  Therefore, it is possible to leave the data bus (portA<3-0>)
; configured for input from PIC#1 at all times.
;
; The commands which PIC#1 sends to PIC#2 have the following protocols:-
;   b'cccc'=1 means "make the following connection"
;             Data nibble of form b'ffgg', where
;             ff=0-3 is the external line (Line#1=0 ... Line#4=3)
;             gg=0-3 is the ATM (ATM#A=0 ... ATM#D=3) to connect to the external line
;   b'cccc'=2 means "break the following connection"
;             Data nibble of form b'ffgg', where
;             ff=0-3 is the external line (Line#1=0 ... Line#4=3)
;             gg=0-3 is the ATM (ATM#A=0 ... ATM#D=3) to disconnect from the external line
;   b'cccc'=3 means disconnect all ATMs from all external lines
;             Data nibble is ignored by PIC#2
;
; Only one procedure needs to be timed: every communication with PIC#1 must time out
; gracefully if not completed within, say, 25ms.
;
; Register timer0 and its control module, Timer0, are used to catch communication timeouts.
; The Timer0 prescalar is set to 256:1, so timer0 increments every 256 x 400ns = 102.4us.
; (Remember that, with a clock at 10MHz, each clock cycle takes 100ns and each instruction
; cycle, consisting of four clock cycles, takes 400ns.)  Therefore, register timer0 will
; count from 0 to 0 (256 steps) in 256 x 102.4us = 26.2144ms or from 1 to 0 (255 steps) in
; 255 x 102.4us = 26.112ms.
;
; When a communication from PIC#1 starts, register timer0 is set to d'1'.  When register
; timer0 reaches zero, a Timer0 interrupt will be trigerred.  PIC#2 will interpret this
; interrupt as a communication timeout.  The interrupt will occur 26.112ms after timer0
; is set to d'1'.  This is sufficiently close to the 25ms design time for our purposes.
;
; What to do in the event of a communication timeout with PIC#1?  PIC#1 has a similar timer
; to catch communication timeouts with PIC#2.  If PIC#1 is still functioning when the
; communication fails, it will also detect the timeout, in which case it will HardReset
; itself.  If something is not done to prevent it, PIC#2 will become confused as PIC#1's
; interrupt line bounces during the restart process.  There is a choice: PIC#2 can wait
; for, say, 500ms, while PIC#1 recovers from the timeout, or PIC#2 can HardReset itself
; as well.  The latter approach is used here.  The reason is this: when PIC#1 restarts
; itself, it will not remember the state of the connections between the ATMs and the
; external telephone lines.  Whatever information PIC#2 has about those connections will
; be irrelevant.  The simplest solution is to have both PICs HardReset themselves, and
; begin afresh, in the event of a communication timeout.
;
; Configuration Word
;             Protect  off             ; b<13>=b<12>=b<8>=b<5>=b<4>=1
;             Debugger disabled        ; b<11>=1
;             b<10> is unimplemented   ; b<10>=1
;             WRT enabled              ; b<9>=1
;             LVP off                  ; b<7>=0
;             BOR disabled             ; b<6>=0
;             PUT disabled             ; b<3>=1
;             WDT disabled             ; b<2>=0
;             OSC HS                   ; b<1>=1; b<0>=0
; Crystal frequency = 10MHz
;
  processor       16F872
  __config        0x3F3A             ; b'0011 1111 0011 1010'
;
; Variable definitions - PIC 16F87x control registers
;
timer0       equ   0x01
status       equ   0x03
carry        equ     0x00
zero         equ     0x02
page0        equ     0x05
page1        equ     0x06
portA        equ   0x05
```

```
portB         equ   0x06
portC         equ   0x07
INTCon        equ   0x0B        ; interrupt control register
RB0IF         equ      0x01     ; RB0 interrupt flag
Tmr0IF        equ      0x02     ; Timer0 interrupt flag
RB0IE         equ      0x04     ; RB0 interrupt enable
Tmr0IE        equ      0x05     ; Timer0 interrupt enable
GIE           equ      0x07     ; global interrupt enable
T1CON         equ   0x10        ; controls use of portC<1-0>
CCP1CON       equ   0x17        ; controls use of portC<2>
optionreg     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
ADCON1        equ   0x9F        ; controls use of portA
f             equ      0x01     ; f and w identify the destination register
w             equ      0x00
;
; Variable definitions - User RAM – Accessible only in bank0
;
; *** I/O ports ***
portAmirror   equ   0x20
D0            equ      0x00     ; portA<3-0> is the four-bit data bus with PIC#1
D1            equ      0x01
D2            equ      0x02
D3            equ      0x03
K6D           equ      0x04     ; line K* controls replay K*. K* active high closes relay K*.
K4D           equ      0x05
portBmirror   equ   0x21
INT           equ      0x00     ; interrupt request input from PIC#1
ACK           equ      0x01     ; interrupt acknowledge output to PIC#1, which interrupts PIC#1
K2A           equ      0x02
K6A           equ      0x03
K3D           equ      0x04
K3A           equ      0x05
K3B           equ      0x06
K3C           equ      0x07
portCmirror   equ   0x22
K6C           equ      0x00
K4C           equ      0x01
K2D           equ      0x02
K2C           equ      0x03
K2B           equ      0x04
K4B           equ      0x05
K6B           equ      0x06
K4A           equ      0x07
; *** program registers ***
Rcommand1     equ   0x23        ; first nibble of command received from PIC#1
Rcommand2     equ   0x24        ; second nibble of command received from PIC#1
tempGCP1      equ   0x25        ; temporary register for subroutine GetCommandFromPIC1
tempTest7A    equ 0x26          ; temporary register used in TestProgram7
tempTest7B    equ   0x27        ; temporary register used in TestProgram7
tempTest7C    equ   0x28        ; temporary register used in TestProgram7
count1        equ   0x29        ; counters for delay subroutines
count2        equ   0x2A
count3        equ   0x2B
;
; Variable definitions - User RAM – Accessible in all banks
;
_wRB0         equ   0x70        ; temporary storage for the w register during RB0 interrupts
_sRB0         equ   0x71        ; temporary storage for the status register during RB0 interrupts
```

```
;
; Location 0x0000
;
     org     0x0000
     goto    HardReset               ; goto HardReset on power-up
     nop
     nop
     nop
;
; ********************************************************************************************
; ********************************************************************************************
; Interrupt Service Routine starts at address 0x0004
; There are two sources of interrupt: by Timer0 and by PIC#1.
; 1.  Conflict between the two sources of interrupt is not relevant.  If a Timer0 interrupt
;     occurs, PIC#2 will be restarted and the state of an RB0 interrupt, if one is under
;     way, is irrelevant.
; 2.  The PIC#1 interrupt reads a two-nibble command sent by PIC#1.  The command is executed
;      within the ISR, so there is no possibility that a second command will be received
;     before a previous command has finished executing.
; ********************************************************************************************
; ********************************************************************************************
;
     org     0x0004
ISR
     bcf     INTCon,GIE              ; disable global interrupts, but do not clear flags
     btfss   INTCon,RB0IF            ; INTCon<RB0IF> is set if RB0 caused the interrupt
     goto    HardReset
     movwf   _wRB0                   ; save the w register during PIC#1 interrupts
     movf    status,w
     movwf   _sRB0                   ; also save the status register during PIC#1 interrupts
     call    StartTOTimer            ; begin 25ms communication timeout test
     call    GetCommandFromPIC1      ; command from PIC#1 is returned in Rcommand1 and Rcommand2
     call    StopTOTimer             ; end 25ms communication timeout test

                                     ; ***comment-out the following line to run TestProgram6***
                                     ; ***comment-out the following line to run TestProgram7***
     call    ExecuteCommandFromPIC1  ; execute the command received from PIC#1

     movf    _sRB0,w
     movwf   status                  ; retrieve the original status register
     movf    _wRB0,w                 ; retrieve the original w register
     bcf     INTCon,RB0IF            ; clear RB0 interrupt flag
     bsf     INTCon,GIE              ; re-enable global interrupts
     retfie
;
; ********************************************************************************************
; ********************************************************************************************
; Hard reset
; ********************************************************************************************
; ********************************************************************************************
;
HardReset
     bcf     INTCon,GIE              ; disable global interrupts (until needed)
     bcf     INTCon,RB0IE            ; disable RB0 interrupts (until needed)
     bcf     INTCon,Tmr0IE           ; disable Timer0 interrupts (until needed)
     bcf     INTCon,RB0IF            ; clear RB0 interrupt flag
     bcf     INTCon,Tmr0IF           ; clear Timer0 interrupt flag
     clrf    portA                   ; clear all I/O latches
     clrf    portB
     clrf    portC
     movlw   0x00                    ; set T1CON=0 to disable Timer1 and ...
     movwf   T1CON                   ; ... release portC<1-0> for digital I/O
     movlw   0x00                    ; set CCP1CON=0 to disable Capture/Compare/PWM and ..
     movwf   CCP1CON                 ; ... release portC<2> for digital I/O
     bsf     status,page0            ; select register bank 1
     bcf     status,page1
     movlw   0x06                    ; set ADCON1<3-1>=b'011' to ...
     movwf   ADCON1                  ; ... configure portA for digital I/O
     movlw   0x0F                    ; configure portA<5-4> for output, portA<3-0> for input
     movwf   TRISA
    movlw    0x01                    ; configure portB<7-1> for output, portB<0> for input
```

```
     movwf    TRISB
     movlw    0x00                    ; configure portC<7-0> for output
     movwf    TRISC
     bsf      optionreg,7             ; <7>=1 disables PortB pull-up resistors
     bsf      optionreg,6             ; <6>=1 triggers RB0 interrupts on the rising edge
     bcf      optionreg,5             ; <5>=0 uses the internal clock to increment Timer0
     bcf      optionreg,4             ; <4>=0 increments Timer0 on low-to-high transitions
     bcf      optionreg,3             ; <3>=0 assigns the prescalar to the Timer0 module
     bsf      optionreg,2             ; <2-0>=111 sets the prescalar for Timer0 to 256:1
     bsf      optionreg,1
     bsf      optionreg,0
     bcf      status,page0            ; select register bank 0
     bcf      status,page1
InitializeRegistersAndSettings
     clrf     portAmirror
     clrf     portBmirror
     clrf     portCmirror
     clrf     Rcommand1
     clrf     Rcommand2
     bcf      portBmirror,ACK         ; assert interrupt line to PIC#1 low
     movf     portBmirror,w
     movwf    portB
     call     DisconnectAllfromAll
BeginExecution
     call     del500ms                ; wait one-half second for circuits to settle
     bcf      INTCon,RB0IF            ; clear RB0 interrupt flag
     bcf      INTCon,Tmr0IF           ; clear Timer0 interrupt flag
     bcf      INTCon,Tmr0IE           ; disable Timer0 interrupts (enabled upon interrupt)
     bsf      INTCon,RB0IE            ; enable RB0 interrupts
     bsf      INTCon,GIE              ; enable global interrupts
     goto     MainLoop                ; begin the main program
;    goto     TestProgram1            ; to execute test programs, compile with desired goto
;    goto     TestProgram2
;    goto     TestProgram3
;    goto     TestProgram4
;    goto     TestProgram5
;    goto     TestProgram6
;    goto     TestProgram7
;
; ********************************************************************************************
; Main loop
;
MainLoop
     nop                              ; Main loop is a loop.
     nop                              ; Commands from PIC#1 are executed within
     nop                              ; the ISR at the time they are received.
     nop
     nop
     nop
     nop
     nop
     nop
     nop
     goto     MainLoop
;
; ********************************************************************************************
; ********************************************************************************************
; Index of subroutines
;    CloseK2A - no waiting            ; "no waiting" means that the subroutine returns
;    OpenK2A - no waiting             ; immediately after the voltage over the relay coil
;    CloseK2B - no waiting            ; is changed and without waiting a grace period for
;    OpenK2B - no waiting             ; the relay contacts to react.
;    CloseK2C - no waiting
;    OpenK2C - no waiting
;    CloseK2D - no waiting
;    OpenK2D - no waiting
;    CloseK3A - no waiting
;    OpenK3A - no waiting
;    CloseK3B - no waiting
;    OpenK3B - no waiting
;    CloseK3C - no waiting
```

```
;    OpenK3C - no waiting
;    CloseK3D - no waiting
;    OpenK3D - no waiting
;    CloseK4A - no waiting
;    OpenK4A - no waiting
;    CloseK4B - no waiting
;    OpenK4B - no waiting
;    CloseK4C - no waiting
;    OpenK4C - no waiting
;    CloseK4D - no waiting
;    OpenK4D - no waiting
;    CloseK6A - no waiting
;    OpenK6A - no waiting
;    CloseK6B - no waiting
;    OpenK6B - no waiting
;    CloseK6C - no waiting
;    OpenK6C - no waiting
;    CloseK6D - no waiting
;    OpenK6D - no waiting
;    ConnectAto1 - waits 10ms
;    DisconnectAfrom1 - no waiting
;    ConnectAto2 - waits 10ms
;    DisconnectAfrom2 - no waiting
;    ConnectAto3 - waits 10ms
;    DisconnectAfrom3 - no waiting
;    ConnectAto4 - waits 10ms
;    DisconnectAfrom4 - no waiting
;    ConnectBto1 - waits 10ms
;    DisconnectBfrom1 - no waiting
;    ConnectAto2 - waits 10ms
;    DisconnectBfrom2 - no waiting
;    ConnectBto3 - waits 10ms
;    DisconnectBfrom3 - no waiting
;    ConnectBto4 - waits 10ms
;    DisconnectBfrom4 - no waiting
;    ConnectCto1 - waits 10ms
;    DisconnectCfrom1 - no waiting
;    ConnectCto2 - waits 10ms
;    DisconnectCfrom2 - no waiting
;    ConnectCto3 - waits 10ms
;    DisconnectCfrom3 - no waiting
;    ConnectCto4 - waits 10ms
;    DisconnectCfrom4 - no waiting
;    ConnectDto1 - waits 10ms
;    DisconnectDfrom1 - no waiting
;    ConnectDto2 - waits 10ms
;    DisconnectDfrom2 - no waiting
;    ConnectDto3 - waits 10ms
;    DisconnectDfrom3 - no waiting
;    ConnectDto4 - waits 10ms
;    DisconnectDfrom4 - no waiting
;    DisconnectAllfromAll - no waiting
;    del500ms
;    del10ms
;    del1ms
;    del100us
;    ExecuteCommandFromPIC1
;    GetCommandFromPIC1 - completes the ISR to read two nibbles from PIC#1
;    StartTOTimer - starts timing a 25ms communication timeout with PIC#1
;    StopTOTimer - stops timing a 25ms communication timeout with PIC#1
; ********************************************************************************************
; ********************************************************************************************
;
; ********************************************************************************************
; Subroutine CloseK2A closes relay K2A - no waiting
;
CloseK2A
     bsf     portBmirror,K2A
     movf    portBmirror,w
     movwf   portB
     return
```

```
;
; ************************************************************************************************
; Subroutine OpenK2A opens relay K2A - no waiting
;
OpenK2A
     bcf     portBmirror,K2A
     movf    portBmirror,w
     movwf   portB
     return
;
; ************************************************************************************************
; Subroutine CloseK2B closes relay K2B - no waiting
;
CloseK2B
     bsf     portCmirror,K2B
     movf    portCmirror,w
     movwf   portC
     return
;
; ************************************************************************************************
; Subroutine OpenK2B opens relay K2B - no waiting
;
OpenK2B
     bcf     portCmirror,K2B
     movf    portCmirror,w
     movwf   portC
     return
;
; ************************************************************************************************
; Subroutine CloseK2C closes relay K2C - no waiting
;
CloseK2C
     bsf     portCmirror,K2C
     movf    portCmirror,w
     movwf   portC
     return
;
; ************************************************************************************************
; Subroutine OpenK2C opens relay K2C - no waiting
;
OpenK2C
     bcf     portCmirror,K2C
     movf    portCmirror,w
     movwf   portC
     return
;
; ************************************************************************************************
; Subroutine CloseK2D closes relay K2D - no waiting
;
CloseK2D
     bsf     portCmirror,K2D
     movf    portCmirror,w
     movwf   portC
     return
;
; ************************************************************************************************
; Subroutine OpenK2D opens relay K2D - no waiting
;
OpenK2D
     bcf     portCmirror,K2D
     movf    portCmirror,w
     movwf   portC
     return
;
; ************************************************************************************************
; Subroutine CloseK3A closes relay K3A - no waiting
;
CloseK3A
     bsf     portBmirror,K3A
     movf    portBmirror,w
     movwf   portB
```

```
     return
;
; ************************************************************************************************
; Subroutine OpenK3A opens relay K3A - no waiting
;
OpenK3A
     bcf      portBmirror,K3A
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine CloseK3B closes relay K3B - no waiting
;
CloseK3B
     bsf      portBmirror,K3B
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine OpenK3B opens relay K3B - no waiting
;
OpenK3B
     bcf      portBmirror,K3B
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine CloseK3C closes relay K3C - no waiting
;
CloseK3C
     bsf      portBmirror,K3C
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine OpenK3C opens relay K3C - no waiting
;
OpenK3C
     bcf      portBmirror,K3C
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine CloseK3D closes relay K3D - no waiting
;
CloseK3D
     bsf      portBmirror,K3D
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine OpenK3D opens relay K3D - no waiting
;
OpenK3D
     bcf      portBmirror,K3D
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine CloseK4A closes relay K4A - no waiting
;
CloseK4A
     bsf      portCmirror,K4A
     movf     portCmirror,w
```

```
     movwf   portC
     return
;
; ********************************************************************************************
; Subroutine OpenK4A opens relay K4A - no waiting
;
OpenK4A
     bcf     portCmirror,K4A
     movf    portCmirror,w
     movwf   portC
     return
;
; ********************************************************************************************
; Subroutine CloseK4B closes relay K4B - no waiting
;
CloseK4B
     bsf     portCmirror,K4B
     movf    portCmirror,w
     movwf   portC
     return
;
; ********************************************************************************************
; Subroutine OpenK4B opens relay K4B - no waiting
;
OpenK4B
     bcf     portCmirror,K4B
     movf    portCmirror,w
     movwf   portC
     return
;
; ********************************************************************************************
; Subroutine CloseK4C closes relay K4C - no waiting
;
CloseK4C
     bsf     portCmirror,K4C
     movf    portCmirror,w
     movwf   portC
     return
;
; ********************************************************************************************
; Subroutine OpenK4C opens relay K4C - no waiting
;
OpenK4C
     bcf     portCmirror,K4C
     movf    portCmirror,w
     movwf   portC
     return
;
; ********************************************************************************************
; Subroutine CloseK4D closes relay K4D - no waiting
;
CloseK4D
     bsf     portAmirror,K4D
     movf    portAmirror,w
     movwf   portA
     return
;
; ********************************************************************************************
; Subroutine OpenK4D opens relay K4D - no waiting
;
OpenK4D
     bcf     portAmirror,K4D
     movf    portAmirror,w
     movwf   portA
     return
;
; ********************************************************************************************
; Subroutine CloseK6A closes relay K5A/K6A - no waiting
;
CloseK6A
     bsf     portBmirror,K6A
```

```
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine OpenK6A opens relay K5A/K6A - no waiting
;
OpenK6A
     bcf      portBmirror,K6A
     movf     portBmirror,w
     movwf    portB
     return
;
; ************************************************************************************************
; Subroutine CloseK6B closes relay K5B/K6B - no waiting
;
CloseK6B
     bsf      portCmirror,K6B
     movf     portCmirror,w
     movwf    portC
     return
;
; ************************************************************************************************
; Subroutine OpenK6B opens relay K5B/K6B - no waiting
;
OpenK6B
     bcf      portCmirror,K6B
     movf     portCmirror,w
     movwf    portC
     return
;
; ************************************************************************************************
; Subroutine CloseK6C closes relay K5C/K6C - no waiting
;
CloseK6C
     bsf      portCmirror,K6C
     movf     portCmirror,w
     movwf    portC
     return
;
; ************************************************************************************************
; Subroutine OpenK6C opens relay K5C/K6C - no waiting
;
OpenK6C
     bcf      portCmirror,K6C
     movf     portCmirror,w
     movwf    portC
     return
;
; ************************************************************************************************
; Subroutine CloseK6D closes relay K5D/K6D - no waiting
;
CloseK6D
     bsf      portAmirror,K6D
     movf     portAmirror,w
     movwf    portA
     return
;
; ************************************************************************************************
; Subroutine OpenK6D opens relay K5D/K6D - no waiting
;
OpenK6D
     bcf      portAmirror,K6D
     movf     portAmirror,w
     movwf    portA
     return
;
; ************************************************************************************************
; Subroutine ConnectAto1 connects ATM#A to external line#1
;
ConnectAto1
```

```
     call     CloseK3A
     call     OpenK6A
     call     OpenK4A
     call     del10ms
     call     CloseK2A
     return
;
; ************************************************************************************************
; Subroutine DisconnectAfrom1 disconnects ATM#A from external line#1
;
DisconnectAfrom1
     call     OpenK3A
     call     OpenK6A
     call     OpenK4A
     call     OpenK2A
     return
;
; ************************************************************************************************
; Subroutine ConnectAto2 connects ATM#A to external line#2
;
ConnectAto2
     call     CloseK3B
     call     CloseK6A
     call     OpenK4A
     call     del10ms
     call     CloseK2A
     return
;
; ************************************************************************************************
; Subroutine DisconnectAfrom2 disconnects ATM#A from external line#2
;
DisconnectAfrom2
     call     OpenK3B
     call     OpenK6A
     call     OpenK4A
     call     OpenK2A
     return
;
; ************************************************************************************************
; Subroutine ConnectAto3 connects ATM#A to external line#3
;
ConnectAto3
     call     CloseK3C
     call     OpenK6A
     call     CloseK4A
     call     del10ms
     call     CloseK2A
     return
;
; ************************************************************************************************
; Subroutine DisconnectAfrom3 disconnects ATM#A from external line#3
;
DisconnectAfrom3
     call     OpenK3C
     call     OpenK6A
     call     OpenK4A
     call     OpenK2A
     return
;
; ************************************************************************************************
; Subroutine ConnectAto4 connects ATM#A to external line#4
;
ConnectAto4
     call     CloseK3D
     call     CloseK6A
     call     CloseK4A
     call     del10ms
     call     CloseK2A
     return
;
; ************************************************************************************************
```

```
; Subroutine DisconnectAfrom4 disconnects ATM#A from external line#4
;
DisconnectAfrom4
     call     OpenK3D
     call     OpenK6A
     call     OpenK4A
     call     OpenK2A
     return
;
; ************************************************************************************************
; Subroutine ConnectBto1 connects ATM#B to external line#1
;
ConnectBto1
     call     CloseK3A
     call     OpenK6B
     call     OpenK4B
     call     del10ms
     call     CloseK2B
     return
;
; ************************************************************************************************
; Subroutine DisconnectBfrom1 disconnects ATM#B from external line#1
;
DisconnectBfrom1
     call     OpenK3A
     call     OpenK6B
     call     OpenK4B
     call     OpenK2B
     return
;
; ************************************************************************************************
; Subroutine ConnectBto2 connects ATM#B to external line#2
;
ConnectBto2
     call     CloseK3B
     call     CloseK6B
     call     OpenK4B
     call     del10ms
     call     CloseK2B
     return
;
; ************************************************************************************************
; Subroutine DisconnectBfrom2 disconnects ATM#B from external line#2
;
DisconnectBfrom2
     call     OpenK3B
     call     OpenK6B
     call     OpenK4B
     call     OpenK2B
     return
;
; ************************************************************************************************
; Subroutine ConnectBto3 connects ATM#B to external line#3
;
ConnectBto3
     call     CloseK3C
     call     OpenK6B
     call     CloseK4B
     call     del10ms
     call     CloseK2B
     return
;
; ************************************************************************************************
; Subroutine DisconnectBfrom3 disconnects ATM#B from external line#3
;
DisconnectBfrom3
     call     OpenK3C
     call     OpenK6B
     call     OpenK4B
     call     OpenK2B
     return
```

```
;
; ************************************************************************************************
; Subroutine ConnectBto4 connects ATM#B to external line#4
;
ConnectBto4
     call     CloseK3D
     call     CloseK6B
     call     CloseK4B
     call     del10ms
     call     CloseK2B
     return
;
; ************************************************************************************************
; Subroutine DisconnectBfrom4 disconnects ATM#B from external line#4
;
DisconnectBfrom4
     call     OpenK3D
     call     OpenK6B
     call     OpenK4B
     call     OpenK2B
     return
;
; ************************************************************************************************
; Subroutine ConnectCto1 connects ATM#C to external line#1
;
ConnectCto1
     call     CloseK3A
     call     OpenK6C
     call     OpenK4C
     call     del10ms
     call     CloseK2C
     return
;
; ************************************************************************************************
; Subroutine DisconnectCfrom1 disconnects ATM#C from external line#1
;
DisconnectCfrom1
     call     OpenK3A
     call     OpenK6C
     call     OpenK4C
     call     OpenK2C
     return
;
; ************************************************************************************************
; Subroutine ConnectCto2 connects ATM#C to external line#2
;
ConnectCto2
     call     CloseK3B
     call     CloseK6C
     call     OpenK4C
     call     del10ms
     call     CloseK2C
     return
;
; ************************************************************************************************
; Subroutine DisconnectCfrom2 disconnects ATM#C from external line#2
;
DisconnectCfrom2
     call     OpenK3B
     call     OpenK6C
     call     OpenK4C
     call     OpenK2C
     return
;
; ************************************************************************************************
; Subroutine ConnectCto3 connects ATM#C to external line#3
;
ConnectCto3
     call     CloseK3C
     call     OpenK6C
     call     CloseK4C
```

```
     call     del10ms
     call     CloseK2C
     return
;
; ************************************************************************************************
; Subroutine DisconnectCfrom3 disconnects ATM#C from external line#3
;
DisconnectCfrom3
     call     OpenK3C
     call     OpenK6C
     call     OpenK4C
     call     OpenK2C
     return
;
; ************************************************************************************************
; Subroutine ConnectCto4 connects ATM#C to external line#4
;
ConnectCto4
     call     CloseK3D
     call     CloseK6C
     call     CloseK4C
     call     del10ms
     call     CloseK2C
     return
;
; ************************************************************************************************
; Subroutine DisconnectCfrom4 disconnects ATM#C from external line#4
;
DisconnectCfrom4
     call     OpenK3D
     call     OpenK6C
     call     OpenK4C
     call     OpenK2C
     return
;
; ************************************************************************************************
; Subroutine ConnectDto1 connects ATM#D to external line#1
;
ConnectDto1
     call     CloseK3A
     call     OpenK6D
     call     OpenK4D
     call     del10ms
     call     CloseK2D
     return
;
; ************************************************************************************************
; Subroutine DisconnectDfrom1 disconnects ATM#D from external line#1
;
DisconnectDfrom1
     call     OpenK3A
     call     OpenK6D
     call     OpenK4D
     call     OpenK2D
     return
;
; ************************************************************************************************
; Subroutine ConnectDto2 connects ATM#D to external line#2
;
ConnectDto2
     call     CloseK3B
     call     CloseK6D
     call     OpenK4D
     call     del10ms
     call     CloseK2D
     return
;
; ************************************************************************************************
; Subroutine DisconnectDfrom2 disconnects ATM#D from external line#2
;
DisconnectDfrom2
```

```
     call     OpenK3B
     call     OpenK6D
     call     OpenK4D
     call     OpenK2D
     return
;
; ***********************************************************************************************
; Subroutine ConnectDto3 connects ATM#D to external line#3
;
ConnectDto3
     call     CloseK3C
     call     OpenK6D
     call     CloseK4D
     call     del10ms
     call     CloseK2D
     return
;
; ***********************************************************************************************
; Subroutine DisconnectDfrom3 disconnects ATM#D from external line#3
;
DisconnectDfrom3
     call     OpenK3C
     call     OpenK6D
     call     OpenK4D
     call     OpenK2D
     return
;
; ***********************************************************************************************
; Subroutine ConnectDto4 connects ATM#D to external line#4
;
ConnectDto4
     call     CloseK3D
     call     CloseK6D
     call     CloseK4D
     call     del10ms
     call     CloseK2D
     return
;
; ***********************************************************************************************
; Subroutine DisconnectDfrom4 disconnects ATM#D from external line#4
;
DisconnectDfrom4
     call     OpenK3D
     call     OpenK6D
     call     OpenK4D
     call     OpenK2D
     return
;
; ***********************************************************************************************
; Subroutine DisconnectAllfromAll disconnects all ATMs from all external lines
;
DisconnectAllfromAll
     call     OpenK3A
     call     OpenK6A
     call     OpenK4A
     call     OpenK2A
     call     OpenK3B
     call     OpenK6B
     call     OpenK4B
     call     OpenK2B
     call     OpenK3C
     call     OpenK6C
     call     OpenK4C
     call     OpenK2C
     call     OpenK3D
     call     OpenK6D
     call     OpenK4D
     call     OpenK2D
     return
;
; ***********************************************************************************************
```

```
; Subroutine del500ms delays for 503.082ms
;                                   ; Cycles
del500ms
     movlw    0x32                  ; 1 (Note that 0x32 = d'50'.)
     movwf    count3                ; 1
del500msa
     call     del10ms               ; 50 x 25,151 = 1,257,550
     decfsz   count3,f              ; (49 x 1) + (1 x 2) = 51
     goto     del500msa             ; 49 x 2 = 98
     return                         ; 2
;                                   ; CALL adds 2 -- 1,257,705 cycles x 400ns = 503,082,000ns
;
; ********************************************************************************************
; Subroutine del10ms delays for 10.0604ms
;                                   ; Cycles
del10ms
     movlw    0x63                  ; 1 (Note that 0x63 = d'99'.)
     movwf    count2                ; 1
del10msa
     call     del100us              ; 99 x 251 = 24,849
     decfsz   count2,f              ; (98 x 1) + (1 x 2) = 100
     goto     del10msa              ; 98 x 2 = 196
     return                         ; 2
;                                   ; CALL adds 2 -- 25,151 cycles x 400ns = 10,060,400ns
;
; ********************************************************************************************
; Subroutine del1ms delays for 1.018ms
;                                   ; Cycles
del1ms
     movlw    0x0A                  ; 1 (Note that 0x0A = d'10'.)
     movwf    count2                ; 1
del1msa
     call     del100us              ; 10 x 251 = 2,510
     decfsz   count2,f              ; (9 x 1) + (1 x 2) = 11
     goto     del1msa               ; 9 x 2 = 18
     return                         ; 2
;                                   ; CALL adds 2 -- 2,545 cycles x 400ns = 1,018,000ns
;
; ********************************************************************************************
; Subroutine del100us delays for 100.4us
;                                   ; Cycles
del100us
     movlw    0x52                  ; 1 (Note that 0x52 = d'82'.)
     movwf    count1                ; 1
del100usa
     decfsz   count1,f              ; (81 x 1) + (1 x 2) = 83
     goto     del100usa             ; 81 x 2 = 162
     return                         ; 2
;                                   ; CALL adds 2 -- 251 cycles x 400ns = 100,400ns
;
; ********************************************************************************************
; Subroutine ExecuteCommandFromPIC1 executes a command received from PIC#1
ExecuteCommandFromPIC1
     movf     Rcommand1,w
     xorlw    0x01
     btfsc    status,zero           ; if Rcommand=d'1', then the zero flag is set and ...
     goto     MakeConnection        ; ... this is a "make connection" command
     movf     Rcommand1,w
     xorlw    0x02
     btfsc    status,zero           ; if Rcommand1=d'2', then the zero flag is set and ...
     goto     BreakConnection       ; ... this is a "break connection" command
     movf     Rcommand1,w
     xorlw    0x03
     btfsc    status,zero           ; if Rcommand1=d'3', then the zero flag is set and ...
     goto     BreakAll              ; ... this is a "break all connections" command
     return                         ; We should never be here.  If we are, ignore the command
MakeConnection
     movf     Rcommand2,w           ; Rcommand2 has the form b'ffgg', where
     xorlw    0x00                  ;   ff=0-3 is the external line and
     btfsc    status,zero           ;   gg=0-3 is the ATM
     call     ConnectAto1
```

```
     movf    Rcommand2,w
     xorlw   0x01
     btfsc   status,zero
     call    ConnectBto1
     movf    Rcommand2,w
     xorlw   0x02
     btfsc   status,zero
     call    ConnectCto1
     movf    Rcommand2,w
     xorlw   0x03
     btfsc   status,zero
     call    ConnectDto1
     movf    Rcommand2,w
     xorlw   0x04
     btfsc   status,zero
     call    ConnectAto2
     movf    Rcommand2,w
     xorlw   0x05
     btfsc   status,zero
     call    ConnectBto2
     movf    Rcommand2,w
     xorlw   0x06
     btfsc   status,zero
     call    ConnectCto2
     movf    Rcommand2,w
     xorlw   0x07
     btfsc   status,zero
     call    ConnectDto2
     movf    Rcommand2,w
     xorlw   0x08
     btfsc   status,zero
     call    ConnectAto3
     movf    Rcommand2,w
     xorlw   0x09
     btfsc   status,zero
     call    ConnectBto3
     movf    Rcommand2,w
     xorlw   0x0A
     btfsc   status,zero
     call    ConnectCto3
     movf    Rcommand2,w
     xorlw   0x0B
     btfsc   status,zero
     call    ConnectDto3
     movf    Rcommand2,w
     xorlw   0x0C
     btfsc   status,zero
     call    ConnectAto4
     movf    Rcommand2,w
     xorlw   0x0D
     btfsc   status,zero
     call    ConnectBto4
     movf    Rcommand2,w
     xorlw   0x0E
     btfsc   status,zero
     call    ConnectCto4
     movf    Rcommand2,w
     xorlw   0x0F
     btfsc   status,zero
     call    ConnectDto4
     return                          ; We should never be here.  If we are, ignore the command
BreakConnection
     movf    Rcommand2,w             ; Rcommand2 has the form b'ffgg', where
     xorlw   0x00                    ;   ff=0-3 is the external line and
     btfsc   status,zero             ;   gg=0-3 is the ATM
     call    DisconnectAfrom1
     movf    Rcommand2,w
     xorlw   0x01
     btfsc   status,zero
     call    DisconnectBfrom1
     movf    Rcommand2,w
```

```
     xorlw    0x02
     btfsc    status,zero
     call     DisconnectCfrom1
     movf     Rcommand2,w
     xorlw    0x03
     btfsc    status,zero
     call     DisconnectDfrom1
     movf     Rcommand2,w
     xorlw    0x04
     btfsc    status,zero
     call     DisconnectAfrom2
     movf     Rcommand2,w
     xorlw    0x05
     btfsc    status,zero
     call     DisconnectBfrom2
     movf     Rcommand2,w
     xorlw    0x06
     btfsc    status,zero
     call     DisconnectCfrom2
     movf     Rcommand2,w
     xorlw    0x07
     btfsc    status,zero
     call     DisconnectDfrom2
     movf     Rcommand2,w
     xorlw    0x08
     btfsc    status,zero
     call     DisconnectAfrom3
     movf     Rcommand2,w
     xorlw    0x09
     btfsc    status,zero
     call     DisconnectBfrom3
     movf     Rcommand2,w
     xorlw    0x0A
     btfsc    status,zero
     call     DisconnectCfrom3
     movf     Rcommand2,w
     xorlw    0x0B
     btfsc    status,zero
     call     DisconnectDfrom3
     movf     Rcommand2,w
     xorlw    0x0C
     btfsc    status,zero
     call     DisconnectAfrom4
     movf     Rcommand2,w
     xorlw    0x0D
     btfsc    status,zero
     call     DisconnectBfrom4
     movf     Rcommand2,w
     xorlw    0x0E
     btfsc    status,zero
     call     DisconnectCfrom4
     movf     Rcommand2,w
     xorlw    0x0F
     btfsc    status,zero
     call     DisconnectDfrom4
     return                          ; We should never be here.  If we are, ignore the command
BreakAll
     call     DisconnectAllfromAll
     return
;
; ********************************************************************************************
; Subroutine GetCommandFromPIC1 completes the ISR to read two nibbles from PIC#1
; Called with:-
;   RB0 interrupts disabled
;   interrupt request line (RB0) is still high; PIC#1 has not been answered
;   interrupt acknowledge line (RB1) presumed low, since PIC#1 initiated the interrupt
;   data bus portA<3-0> configured for input
; Returns with:-
;   data received in Rcommand1 and Rcommand2
;   RB0 interrupts disabled
;   interrupt request line (RB0) low
```

```
;   interrupt acknowledge line (RB1) low
;   data bus portA<3-0> configured for input
;
GetCommandFromPIC1
     bsf     portBmirror,ACK         ; acknowledge interrupt by setting ACK line high
     movf    portBmirror,w
     movwf   portB
GCP1A
     movf    portB,w                 ; read INT bit (do not use direct bit read)
     movwf   tempGCP1
     btfsc   tempGCP1,INT            ; wait until PIC#1 asserts the INT line low
     goto    GCP1A
     movf    portA,w                 ; read the data bus
     andlw   0x0F                    ; keep only the low four bits
     movwf   Rcommand1               ; save the data into register Rcommand1
     bcf     portBmirror,ACK         ; end communication by asserting ACK line low
     movf    portBmirror,w
     movwf   portB
GCP1B
     movf    portB,w                 ; read INT bit (do not use direct bit read)
     movwf   tempGCP1
     btfss   tempGCP1,INT            ; wait until PIC#1 interrupts by setting INT line high
     goto    GCP1B
     bsf     portBmirror,ACK         ; acknowledge interrupt by setting ACK line high
     movf    portBmirror,w
     movwf   portB
GCP1C
     movf    portB,w                 ; read INT bit (do not use direct bit read)
     movwf   tempGCP1
     btfsc   tempGCP1,INT            ; wait until PIC#1 asserts the INT line low
     goto    GCP1C
     movf    portA,w                 ; read the data bus
     andlw   0x0F                    ; keep only the low four bits
     movwf   Rcommand2               ; save the data into register Rcommand2
     bcf     portBmirror,ACK         ; end communication by asserting ACK line low
     movf    portBmirror,w
     movwf   portB
     return
;
; ********************************************************************************************
; Subroutine StartTOTimer initializes register timer0 to detetct a communication timeout.
StartTOTimer
     movlw   0x01
     movwf   timer0
     bsf     INTCon,Tmr0IE
     return
;
; ********************************************************************************************
; Subroutine StopTOTimer stops detetction of a communication timeout.
StopTOTimer
     bcf     INTCon,Tmr0IE
     return
;
; ********************************************************************************************
;
; Test programs used for debugging purposes only
;
; ********************************************************************************************
;
; TestProgram1 closes relays K2A through K2D, each in turn, for 500ms.
TestProgram1
     bcf     INTCon,GIE              ; disable interrupts - TestProgram1 does not use interrupts
     call    OpenK2A
     call    OpenK2B
     call    OpenK2C
     call    OpenK2D
TP1A
     call    CloseK2A
     call    del500ms
     call    OpenK2A
     call    CloseK2B
```

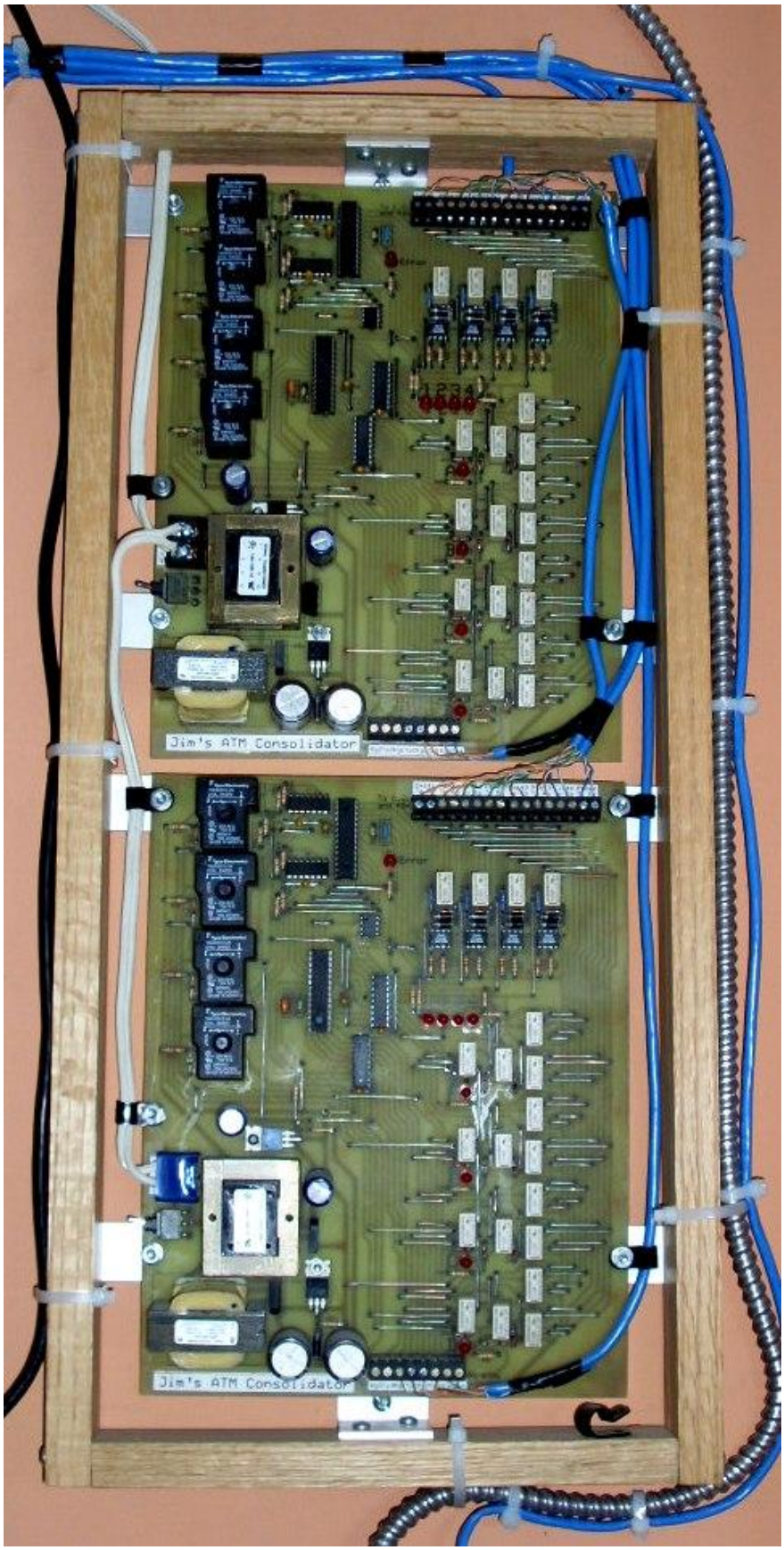
```
     call     del500ms
     call     OpenK2B
     call     CloseK2C
     call     del500ms
     call     OpenK2C
     call     CloseK2D
     call     del500ms
     call     OpenK2D
     goto     TP1A
;
; TestProgram2 closes relays K3A through K3D, each in turn, for 500ms.
TestProgram2
     bcf      INTCon,GIE              ; disable interrupts - TestProgram2 does not use interrupts
     call     OpenK3A
     call     OpenK3B
     call     OpenK3C
     call     OpenK3D
TP2A
     call     CloseK3A
     call     del500ms
     call     OpenK3A
     call     CloseK3B
     call     del500ms
     call     OpenK3B
     call     CloseK3C
     call     del500ms
     call     OpenK3C
     call     CloseK3D
     call     del500ms
     call     OpenK3D
     goto     TP2A
;
; TestProgram3 closes relays K4A through K4D, each in turn, for 500ms.
TestProgram3
     bcf      INTCon,GIE              ; disable interrupts - TestProgram3 does not use interrupts
     call     OpenK4A
     call     OpenK4B
     call     OpenK4C
     call     OpenK4D
TP3A
     call     CloseK4A
     call     del500ms
     call     OpenK4A
     call     CloseK4B
     call     del500ms
     call     OpenK4B
     call     CloseK4C
     call     del500ms
     call     OpenK4C
     call     CloseK4D
     call     del500ms
     call     OpenK4D
     goto     TP3A
;
; TestProgram4 closes relays K5A/K6A through K5D/K6D, each in turn, for 500ms.
TestProgram4
     bcf      INTCon,GIE              ; disable interrupts - TestProgram4 does not use interrupts
     call     OpenK6A
     call     OpenK6B
     call     OpenK6C
     call     OpenK6D
TP4A
     call     CloseK6A
     call     del500ms
     call     OpenK6A
     call     CloseK6B
     call     del500ms
     call     OpenK6B
     call     CloseK6C
     call     del500ms
     call     OpenK6C
```

```
     call     CloseK6D
     call     del500ms
     call     OpenK6D
     goto     TP4A
;
; TestProgram5 goes through all 16 connection pairs of ATMs to external lines.
; First, the connection is made for 500ms, then it is disconnected for 500ms before
; proceeding to the next pair.  This test program is useful for testing the quality
; of the connections using a signl generator and an oscilloscope.  Be sure to test
; the connections between the external lines and their downstream PBXs as well as the
; connections between the external lines and the ATMs.
TestProgram5
     bcf      INTCon,GIE              ; disable interrupts - TestProgram5 does not use interrupts
     call     ConnectAto1
     call     del500ms
     call     DisconnectAfrom1
     call     del500ms
     call     ConnectAto2
     call     del500ms
     call     DisconnectAfrom2
     call     del500ms
     call     ConnectAto3
     call     del500ms
     call     DisconnectAfrom3
     call     del500ms
     call     ConnectAto4
     call     del500ms
     call     DisconnectAfrom4
     call     del500ms
     call     ConnectBto1
     call     del500ms
     call     DisconnectBfrom1
     call     del500ms
     call     ConnectBto2
     call     del500ms
     call     DisconnectBfrom2
     call     del500ms
     call     ConnectBto3
     call     del500ms
     call     DisconnectBfrom3
     call     del500ms
     call     ConnectBto4
     call     del500ms
     call     DisconnectBfrom4
     call     del500ms
     call     ConnectCto1
     call     del500ms
     call     DisconnectCfrom1
     call     del500ms
     call     ConnectCto2
     call     del500ms
     call     DisconnectCfrom2
     call     del500ms
     call     ConnectCto3
     call     del500ms
     call     DisconnectCfrom3
     call     del500ms
     call     ConnectCto4
     call     del500ms
     call     DisconnectCfrom4
     call     del500ms
     call     ConnectDto1
     call     del500ms
     call     DisconnectDfrom1
     call     del500ms
     call     ConnectDto2
     call     del500ms
     call     DisconnectDfrom2
     call     del500ms
     call     ConnectDto3
     call     del500ms
```

```
     call    DisconnectDfrom3
     call    del500ms
     call    ConnectDto4
     call    del500ms
     call    DisconnectDfrom4
     call    del500ms
     goto    TestProgram5
;
; TestProgram6 is used in conjunction with PIC#1's TestProgram4.  For each of the 16
; values of Rcommand1 = d'0' through d'16', PIC#1 sends the 16 values of
; Rcommand2 = d'0' through d16'.  Each nibble pair is sent 500ms after its predecessor.
; For convenience in PIC#1's programming, the numeric sequences are actually sent in
; reverse order, as "countdowns".  PIC#2 displays the first nibble on the four Line-in-use
; LEDs, with Line#1 being the most significant bit.  It displays the second nibble on the
; four ATM-in-use LEDs, with ATM#A being the least significant bit.  This program consists
; of a loop, which continually sets the LEDs according to the current values of Rcommand1
; and Rcommand2, which are changed whenever an interrupt occurs.  Note: For TestProgram6
; to operate properly, it is necessary to comment-out the instruction in the ISR which
; executes commands received from PIC#1.
TestProgram6
     call    OpenK3A                 ; Line-in-use LEDs display Rcommand1 on LEDs
     btfsc   Rcommand1,3             ; which show the status of relays K3*.
     call    CloseK3A                ; For the best visual presentation,
     call    OpenK3B                 ;  Rcommand1<3> is shown on the left-most LED, for Line #1
     btfsc   Rcommand1,2             ;  ...
     call    CloseK3B                ;  Rcommand1<0> is shown on the right-most LED, for Line #4
     call    OpenK3C
     btfsc   Rcommand1,1
     call    CloseK3C
     call    OpenK3D
     btfsc   Rcommand1,0
     call    CloseK3D
     call    OpenK2A
     btfsc   Rcommand2,0             ; ATM-in-use LEDs display Rcommand2 on LEDs
     call    CloseK2A                ; which show the status of relays K2*.
     call    OpenK2B                 ; For the bext isual presentation,
     btfsc   Rcommand2,1             ;  Rcommand2<3> is shown on the bottom-most LED, for ATM #D
     call    CloseK2B                ;   ...
     call    OpenK2C                 ;   Rcommand2<0> is shown on the top-most LED, for ATM #A
     btfsc   Rcommand2,2
     call    CloseK2C
     call    OpenK2D
     btfsc   Rcommand2,3
     call    CloseK2D
     goto    TestProgram6
;
; TestProgram7 is used in conjuncion with PIC#1's TestProgram5, to test the interpretation
; of commands received from PIC#1.  PIC#1 sends the data in the normal interrupt format,
; with Rcommand1 being d'1' or d'2' to make or break a connection, respectively, and with
; Rcommand2 being b'ffgg', where b'ff' is the selected external line and b'gg' is the
; selected ATM.  Each nibble pair is sent one second after its predecessor.  For convenience
; in programming PIC#1, the numeric sequences are actually sent in reverse order, as
; "countdowns".  This test program, together with PIC#1 running its TestProgram5, are useful
; for observing the quality of signal transmission bewteen the ATM input connector (J1)
; and the external line connector (J2).  This program consists of a loop, which continually
; compares the current values of Rcommand1 and Rcommand2 (which are changed exogenously
; by interrupt) with their former values, which are preserved in registers tempTest7A and
; tempTest7B, respectively.  When the values change, TestProgram7 uses subroutine
; ExecuteCommandFromPIC1 to make or break the connection as instructed.  Note: For
; TestProgram7 to operate properly, it is necessary to comment-out the instruction in the
; ISR which executes commands received from PIC#1.
TestProgram7
     call    DisconnectAllfromAll
     clrf    tempTest7A
     clrf    tempTest7B
TP7A                                 ; check for changes in Rcommand1 and Rcommand2
     movf    tempTest7A,w
     xorwf   Rcommand1,w
     btfss   status,zero
     goto    TP7B                    ; goto if tempTest7A <> Rcommand1
     movf    tempTest7B,w
```

```
     xorwf    Rcommand2,w
     btfss    status,zero
     goto     TP7B                    ; goto if tempTest7B <> Rcommand2
     goto     TP7A
TP7B                                  ; here if a change has been detected
     call     ExecuteCommandFromPIC1
     movf     Rcommand1,w
     movwf    tempTest7A              ; update tempTest7A for next interrupt
     movf     Rcommand2,w
     movwf    tempTest7B              ; update tempTest7B for next interrupt
     goto     TP7A

     END                             ; end assembly
```


Jim's ATM Consolidator
Jim's ATM Consolidator