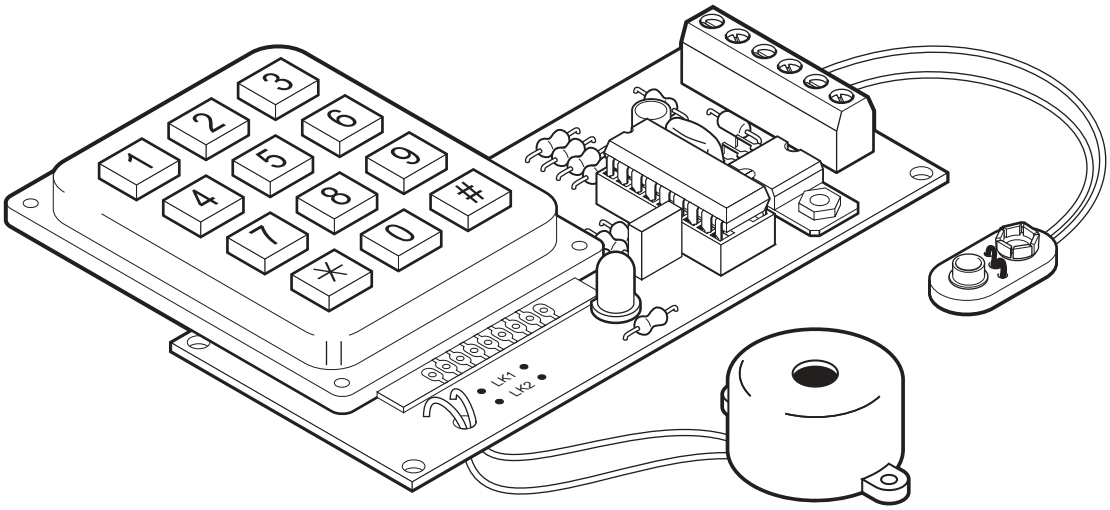
PIC Lock Self-Assembly Kit (v2)

Order Codes:

CHI008 PIC Lock Self-Assembly Kit



Features

- 12 key 'telephone' style keypad
- bicolour LED and piezo feedback devices
- optional FET driven output
- reprogrammable PIC16F627 microcontroller, programmed by direct PICAXE cable connection (cable AXE026) or by traditional programmer method.
- simple construction

Also required: - AA batteries
- soldering iron and solder
- side cutters and small cross-head screwdriver

Contents:

R1	1	4k7 carbon film 0.25W	*yellow violdet red gold*
R2	1	330R carbon film 0.25W	*orange orange brown gold*
R3	1	22k carbon film 0.25W	*red red orange gold*
RP1	1	5 pin 10k SIL resistor	*polarised - note dot at one end*
LK3	1	zero-ohm resistor link	*marked with single black line*
C1	1	100uF miniature electrolytic	*long leg is +ve*
C2	1	100nF polyester capacitor	*marked 104 - not polarised*
CT1	1	stereo PICAXE connector	*ensure 'snapped' onto pcb*
LED1	1	5mm bicolour LED	*align flat with ink image on pcb*
D1	1	1N4001 diode	*grey band faces 'up'*
FET	1	IRF530 FET	*text visible when FET in position*
PZ1	1	piezo transducer	*red wire to +ve*
SW1	1	miniature reset switch	*only fits one way around!*
IC1	1	18 pin IC socket	*use for PIC16F627*
IC1	1	PIC16F627 microcontroller	*pin 1 faces up*
CT	1	4 pole screw terminal block	*contacts face out*
BT1	1	AA battery box + clip	*red wire - V+*
	1	keypad	
	1	pcb	

Assembly Instructions

1. Solder all the resistors in position. The values of the resistors are shown on the pcb, and the colour codes are given in the table on page 1. The 'zero-ohm resistor' wire link (marked with a single black band) is connected across LK3
2. Using a wire off-cut from a resistor leg, connect *either* LK1 *or* LK2 (not both). Use LK1 to connect the piezo sounder to PIC pin RA4 (e.g. for the 'Chip Factory') or LK2 to connect the piezo sounder to PIC pin RB6 (e.g. for PICAXE or 'PIC-Logicator').
3. Solder the diode D1 in position. The grey bar should face away from the FET.
4. Solder the IC socket in position. Solder the 5pin SIL resistor in position, ensuring the dot marked on the resistor aligns with the dot on the PCB.
5. Solder the FET in position. The text should be visible on the FET when it is laid in position. Solder the PICAXE stereo connector in position
6. Solder the rectangular polyester capacitor C2 in position. It can be used either way around.
7. Solder the reset switch in position - it will only fit one way around.
8. Solder the cylindrical electrolytic capacitor C1 in position. The long leg is the +ve leg.
9. Solder the bicolour LED in position. The LED can be soldered directly to the pcb or connected via wires (not supplied). Make sure the flat on the LED aligns with the footprint on the pcb.
10. Solder the piezo sounder PZ in position. Before soldering, the two wires should be woven through the large hole to help strengthen the joint. The red wire connects to the + symbol.
11. Solder the 4 way screw terminal block in position. Make sure the contacts face out. Note that the 4 way terminal block may be supplied as two 2 way terminals which clip together.
12. Screw the battery clip into the bottom two contacts of the terminal block. The red wire is connected to the 6V contact, the black wire to the 0V contact.
13. The keypad may be mounted directly onto the pcb, or connected via an optional ribbon cable (not included). If mounting the keypad on the pcb use wire off-cuts from the resistor legs to solder the 9 keypad pads to the pcb.
14. Push the PICAXE-18 chip into it's socket. Make sure pin 1 faces the four resistors.
15. Insert the AA batteries (not supplied) into the battery pack and then connect to the battery clip.

DO NOT USE A 9V PP3 BATTERY WITH THIS PRODUCT.
ONLY USE THE 4.5V or 6V (AA CELLs) BATTERY BOX SUPPLIED.

Important Note:
The 100uF capacitor will keep the unit powered for approximately 20 seconds after the battery box is removed. Therefore to reset the unit press the reset switch.

Resonator position X1 is not used. It is provided for users who may wish to use the older PIC16F84A rather than the (cheaper) PIC16F627. In this case a 4Mhz 3-pin ceramic resonator should be soldered into the X1 position.

Initial Testing

The reprogrammable PICAXE-18 microcontroller should be programmed with the sample programs given at the rear of this datasheet. See the programming systems help files for further information on how to do this.

If the lock fails to operate check:

- The code is entered slowly, allowing the LED to flash between key presses.
- All solder joints are good, and there are no accidental solder bridges.
- All polarised components, including the microcontroller, are correctly inserted.
- The batteries are correctly inserted.
- The PICAXE18 is programmed correctly.

Input/Output Configuration

Keypad Column 1	- RA0	- Input 0
Keypad Column 2	- RA1	- Input 1
Keypad Column 3	- RA2	- Input 2
Keypad Row 1	- RB0	- Output 0
Keypad Row 2	- RB1	- Output 1
Keypad Row 3	- RB2	- Output 2
Keypad Row 4	- RB3	- Output 3
Piezo via LK1	- RA4	
Piezo via LK2	- RB6	
Bicolour LED	- RB4/5	- Output 4/5
FET Output	- RB7	- Output 7

FET Output Connections

The FET output connections are included for the user to expand the capabilities of the system. They do not require any connection to be used with the test program supplied in the pre-programmed PIC microcontroller. The FET output is connected across the two central terminal block contacts. Polarised components such as buzzers should have the positive (red) wire connected to the upper of the two contacts (which is marked +). Small DC motors should be suppressed with a 220nF polyester capacitor soldered across the motor contacts to prevent electrical noise affecting the operation of the microcontroller.

If desired the marked track on the pcb can be cut so that a separate output power supply can be connected. This enables the output to be powered at a different voltage than the board.

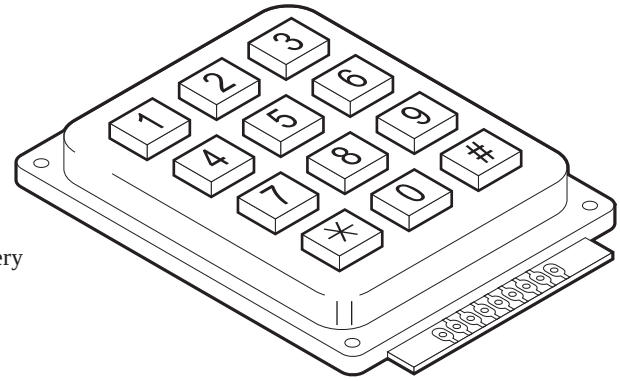
Other Notes

- The IC may be damaged if any power supply other than 6V DC (centre +ve) is applied to the power contacts. A 9V PP3 battery must never be used.
- Alkaline AA cells are recommended for this application.
- The PIC16F627 provided is pre-programmed to act as a PICAXE-18 microcontroller. However if re-programmed in a traditional ZIF socket programmer it will loose the PICAXE bootstrap code and no longer act as a PICAXE microcontroller.

Safety

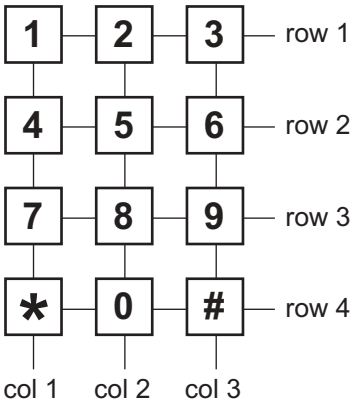
This product is designed as an educational teaching aid. It is not a toy and should not be handled by young children due to sharp edges and small parts.
THIS PRODUCT IS NOT A TOY.

SECURITY LOCKS



Introduction

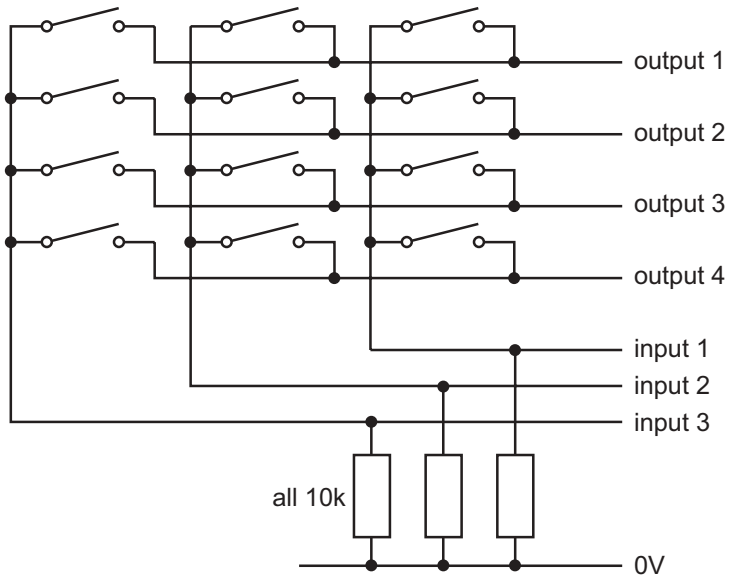
Many commercial alarm systems use a 'telephone style' keypad for entering secret PINs (personal identification numbers). These keypads are relatively cheap, are simple to physically mount, and can add a very professional 'look and feel' to a student project.



Almost all keypads, including computer keyboards, do not consist of individual switches, but are made up of a 'matrix' of switches in structured rows and columns. This is to reduce the number of interfacing lines required – a 12 key keypad only needs 7 interfacing lines (4 rows + 3 columns) as shown left. At first this can make the interfacing of a keypad seem daunting, but once you understand the 'scanning' process it can be seen that the actual electronic circuit is very simple.

Keypad Scanning

The illustration below shows the basic keypad electronic configuration. The four rows are connected to controller output lines, and the three columns are connected to controller input lines.



When no switches are pressed, there is no electrical connection between any row or column. Therefore all the inputs are at the logic level 0 (0V) status, as they are 'pulled low' by the 10k resistors connected to these lines. This means the input signal to the controller is 0-0-0 along the three input lines.

If all the output lines are switched off (at logic level 0) and you press a switch, the input signals will still remain at logic level 0. This is because there is no 'high' level signal anywhere on the keypad, and so the only possible input signal to the controller is still 0-0-0.

However if you switch one, and only one, output line high, and then press a switch on the corresponding row, a high input signal will be generated. For instance, if you press button '1' when row 1 is switched high, the input signal to the controller will be 1-0-0. As you have switched one, and only one, output row high, the controller therefore knows that the 1-0-0 pattern can only have been produced by switch 1. Switches 4,7, and * could not have produced the signal because their output rows are still low. This means that if you continuously scan the keypad, by switching each individual row high in turn, you can uniquely identify each single switch. This is the basic principle used in almost every keypad in the world!

Dedicated Keyboard Scanners

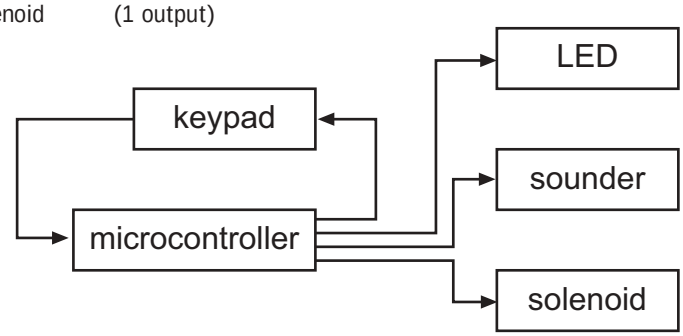
In the past, dedicated scanning chips, such as the MM74C922N have been used to achieve this scanning process. These are simple to use as they do all the scanning internally, and then output a simple 4-bit binary number corresponding to each switch. The disadvantage is that dedicated scanning chips are expensive (typically £5.00 or more).

Fortunately these dedicated scanner chips are now effectively obsolete – as a single microcontroller can easily reproduce their function. Amazingly the microcontroller is also under half the price of these dedicated ICs, and in addition also provides the control features for the other parts of your security device!

Security Lock System

The block diagram for the security lock system is shown below.

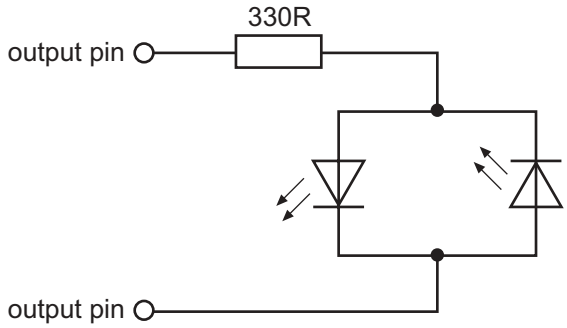
Keypad (3 inputs and 4 outputs)
Bi-colour LED (2 outputs)
Piezo sounder (1 output)
Solenoid (1 output)



When using a keypad, it is important to have some feedback to the user to acknowledge that a key has been pressed. Commonly this is achieved with a buzzer or sounder, and so in this project we will use a piezo-sounder. In microcontroller projects a piezo-sounder is always preferable to a buzzer as it is low cost, it can be directly connected without any interfacing circuits, and can be used to produce a variety of different sounds.

To show the status of the lock it can be useful to have a red (locked) and green (open) indicator LED. Naturally two separate LEDs could be used, but a single, bi-colour LED is a neater solution. This type of LED is often used on televisions or computer monitors to show 'standby' (red) or 'in use' (green) status.

Microcontroller pins can source ('give out'), and sink ('take in'), electric current. This means that if you connect a bi-colour LED as shown in figure 5, the LED will light when either output pin is high as current fill flow out of the 'high' pin and into the 'low' pin. The LED will go out when both output pins are in the same condition (both high or both low) as no current can flow. Which pin is which colour is best found by experimentation! A 330R current limiting resistor should be used in series with the LED.

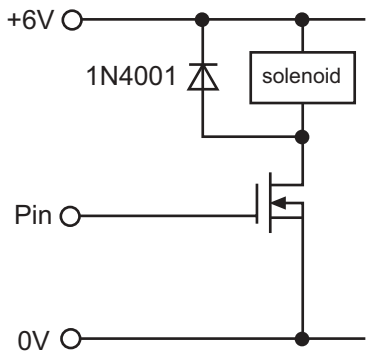


The final electronic interfacing is the solenoid bolt – in real applications this would normally be a 12V electronic door strike plate, but as these cost around £40 we will use a 6V solenoid for modelling purposes.

The instinct of many teachers would be to use a relay for this interfacing. However this is an expensive, and complicated, way of achieving such a simple interfacing task. A Field Effect Transistor (FET) is much more appropriate in this situation, as it can be directly wired to the microcontroller output pin.

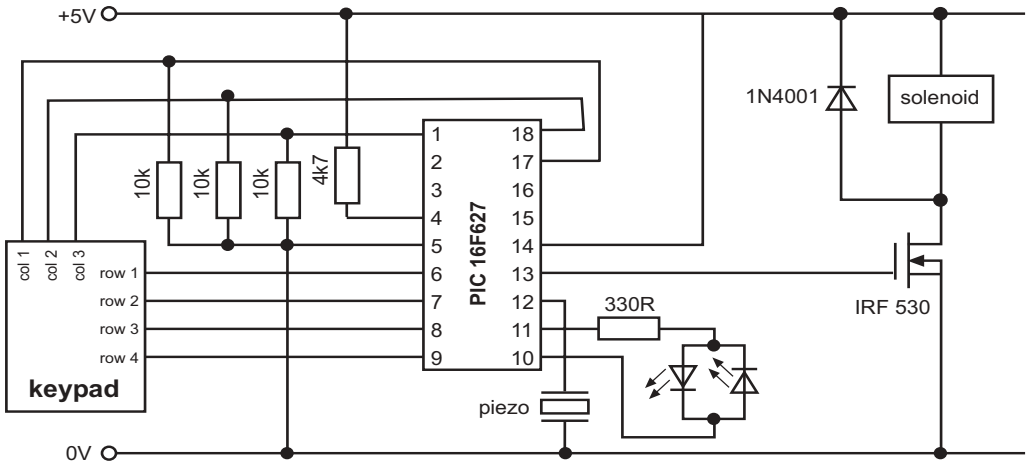
For those unfamiliar with FETs, they are most commonly used as electronic switches. The drain and source connections act as a 'switch', which is activated when the gate connector 'sees' a voltage above 2V. Unlike bipolar transistors FETs are not dependent on the size of the 'base' current, and can also generally switch much higher currents. Whenever possible I would always use a FET as opposed to, for instance, a Darlington driver or relay circuit, as it is generally cheaper and always much easier to wire up!

A suitable FET for most school projects is the IRF530 type. The connections for this device are shown right. Note that a 1N4001 diode is also required to protect the FET from the back EMF generated when the solenoid switches off.



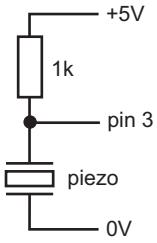
Full Circuit Diagram

The full circuit diagram is shown below. The only extras here are a 4k7 pull-up resistor to disable the microcontroller 'RESET' pin. The optional reset switch is positioned between the RESET pin and 0V.



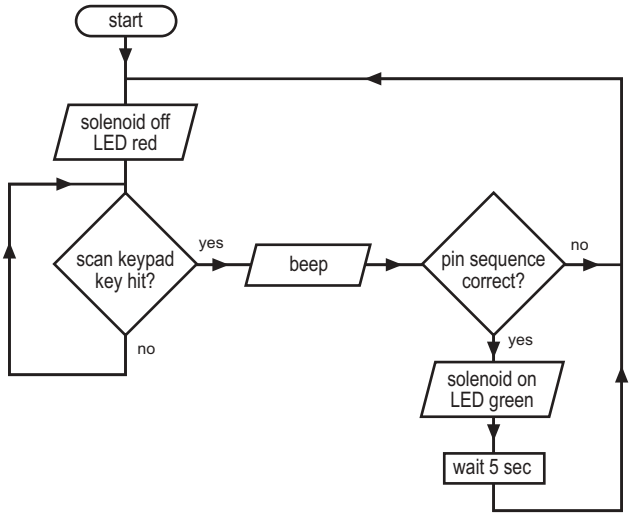
For clarity the PICAXE-18 download circuit (22k resistor and stereo socket) is not shown - see PICAXE manual for further details. 10k resistors are provided in SIL format to reduce pcb size.

NB. For the Chip Factory system the piezo is connected to pin 3.



Software Program

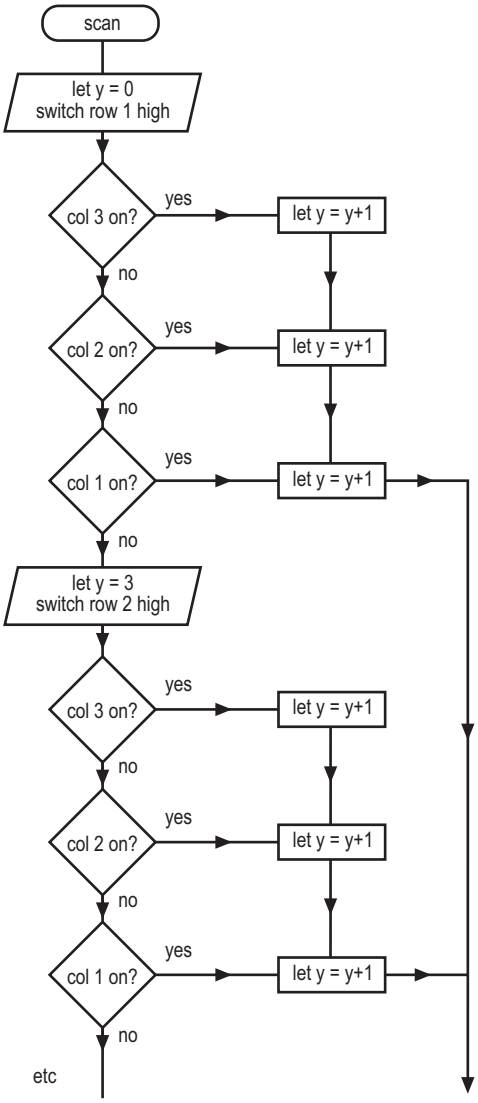
The programming of the lock can be illustrated by the simplified flowchart below. The two main software tasks are the 'keypad scanning' and the 'PIN testing'.



Scanning the Keypad

As explained earlier, the secret of scanning a keypad is to make each output row high in turn, and then to record the value of the input columns if any of them are then switched high by a key-press. The easiest way to do this in software is to use a variable, which we will label Y, as a counter. The flowchart below shows how this is achieved for rows 1 and 2 - rows 3 and 4 follow an identical pattern. The variable Y is preloaded with the value 0, 3, 6, and 9 in turn as each row is scanned. If a column input is detected the number 1, 2 or 3 is added to the existing value of Y, depending on which column has been detected. The means a different number will be generated in Y for every switch.

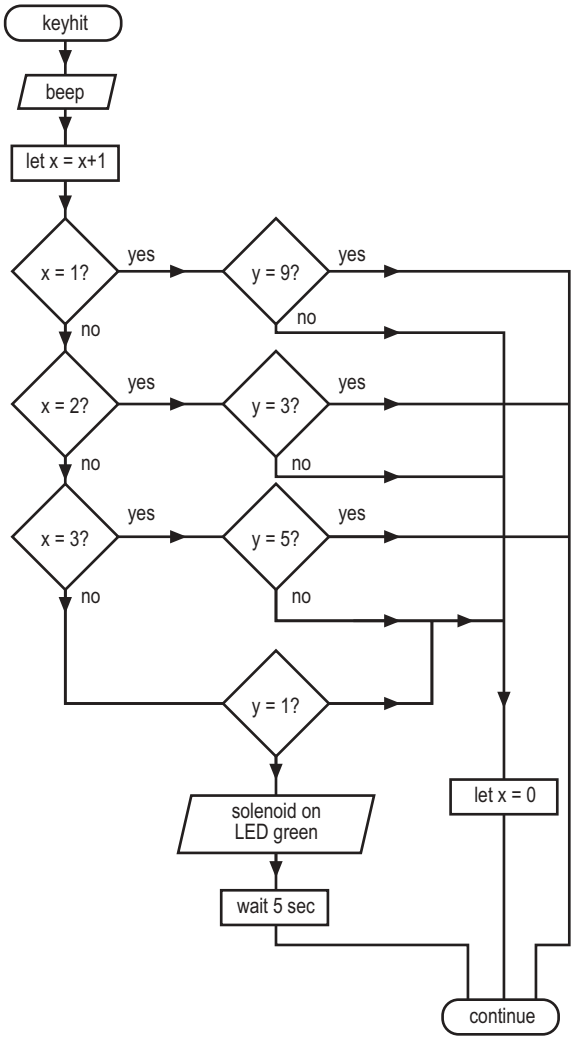
Note from the flowchart that the value '3' (for column 3) is actually added to Y as 1+1+1. This is a very common trick that is used when programming microcontrollers.



As soon as a hit is recorded, scanning ceases and a 'beep' sound is generated to create audible user feedback. The value of Y can then be tested against the PIN number to see if it is the correct value.

Checking the PIN code

The second major software task is to check the recorded key hit against the PIN sequence – in our example the code 9-3-5-1. To do this we need to know the current position within the PIN sequence – first digit, second digit etc. Therefore another variable is used as a position counter (variable X). At the start of the program variable X is given the value 0. When a key hit is recorded the value of X is incremented by 1, as shown below. In the first instance this means X now has a value of 1, representing the first position in the PIN sequence. Therefore we need to compare the value of Y against our first PIN digit value (i.e. the value '9').



If the value of Y is not '9', X is reset back to 'position' 0, so that the whole sequence is restarted. If Y is the correct value, X is left at 'position' 1. This means the next time we hit a switch the value of X will now be incremented to 'position' 2, which in turn means 'compare the value of the second PIN digit'. Once again if the value of Y is wrong variable X will be reset to 'position' 0, so the whole sequence has to start again. This method ensures that only the sequence 9-3-5-1 will work, any other combination will fail. It also protects against codes such as 9-3-2-5-1, which would work if the variable X was not reset on the wrong digit '2' in the middle of the sequence. When the correct code has been entered the solenoid opens and the LED glows green for a period of five seconds. The system the resets ready for the next use.

Chip Factory Program

A full program listing for the Chip Factory programming system is as follows:

```
00    let x    =    000              ' reset counter X
01    let y    =    000              ' preload Y value
02                  out   033         ' switch row 1 on
03    if 0 on       goto  25          ' test column 1
04    if 1 on       goto  24          ' test column 2
05    if 2 on       goto  23          ' test column 3
06    let y    =    003              ' preload Y value
07                  out   034         ' switch row 2 on
08    if 0 on       goto  25          ' test column 1
09    if 1 on       goto  24          ' test column 2
10    if 2 on       goto  23          ' test column 3
11    let y    =    006              ' preload Y value
12                  out   036         ' switch row 3 on
13    if 0 on       goto  25          ' test column 1
14    if 1 on       goto  24          ' test column 2
15    if 2 on       goto  23          ' test column 3
16    let y    =    009              ' preload Y value
17                  out   040         ' switch row 4 on
18    if 0 on       goto  25          ' test column 1
19    if 1 on       goto  24          ' test column 2
20    if 2 on       goto  23          ' test column 3
21                  goto  01          ' scan again
22
23    let y =  y    plus  001         ' add 3 to Y (1+1+1)
24    let y =  y    plus  001         ' add 2 to Y (1+1)
25    let y =  y    plus  001         ' add 1 to Y
26                  beep  150         ' make a noise
27    let x =  x    plus  001         ' always add 1 to X
28    if x =   001  goto  34          ' PIN digit 1 test
29    if x =   002  goto  36          ' PIN digit 2 test
30    if x =   003  goto  38          ' PIN digit 3 test
31    if x =   004  goto  40          ' PIN digit 4 test
32                  goto  00          ' back to start
33
34    if y = 009    goto  01          ' PIN is 9 - continue
35                  goto  00          ' not 9 so reset X
36    if y = 003    goto  01          ' PIN is 3 - continue
37                  goto  00          ' not 3 so reset X
38    if y = 005    goto  01          ' PIN is 5 - continue
39                  goto  00          ' not 5 so reset X
40    if y = 001    goto  43          ' PIN is 1 - finished
41                  goto  00          ' not 1 so reset X
42
43                  out   144         ' solenoid on
44                  wait  050         ' wait 5 seconds
45                  goto  00          ' back to start
```

PICAXE-18 Program

A full program listing for the PICAXE-18 programming system is as follows, (this file is saved in the PICAXE samples sub-folder as CHI008 - Keypad.bas):

```
' Keypad Lock For PICAXE-18
symbol b1 = key_pos
symbol b2 = key_value

' *** reset position to zero ***

init:
      let key_pos = 0

' *** now scan each row in turn ***
' *** by setting only 1 row (and LED) high ***
' *** if a switch is hit jump call score sub below ***

scan:
      let key_value = 0
      let pins = %00010001
      gosub key_test

      let key_value = 3
      let pins = %00010010
      gosub key_test

      let key_value = 6
      let pins = %00010100
      gosub key_test

      let key_value = 9
      let pins = %00011000
      gosub key_test

      goto scan

' *** Score sub procedure. ***
' *** return straight away if no key pressed ***
key_test:
      if pin0 = 1 then add1
      if pin1 = 1 then add2
      if pin2 = 1 then add3
      return

' *** key value will already be 0, 3, 6, or 9 ***
' *** so add 1, 2 or 3 to this value ***

add3:  let key_value = key_value + 1
add2:  let key_value = key_value + 1
add1:  let key_value = key_value + 1

' *** Make a beep ***
      sound 6,(60,50)       '(continued overleaf)
```

```
' *** Now increase position counter by 1 ***
' *** and test for 1st, 2nd 3rd or 4th push ***

      let key_pos = key_pos + 1

      if key_pos = 1 then test1
      if key_pos = 2 then test2
      if key_pos = 3 then test3
'     if key_pos = 4 then test4

' *** Now test the value for each position individually ***
' *** If value is wrong restart, if correct loop until ***
' *** fourth go. If fourth is correct open lock! ***

' *** Key code is set to 9-3-5-1 ***

test4: if key_value = 1 then open
      goto reset

test3: if key_value = 5 then continue
      goto reset

test2: if key_value = 3 then continue
      goto reset

test1: if key_value = 9 then continue
      goto reset


' *** Got here so open lock and set LED green for 5 sec ***

open:  let pins = %10100000
      wait 5
      goto reset

' *** Not correct value so reset position counter then return ***

reset:
      let key_pos = 0

' *** Okay so continue by returning back to main loop ***

continue:
      return
```