

USING AND PROGRAMMING THE 7 SEGMENT DISPLAY PROTOTYPE CIRCUIT

by Lewy Harrison

Submission date: 13-Jan-2025 12:06AM (UTC+0000)

Submission ID: 248160281

File name:

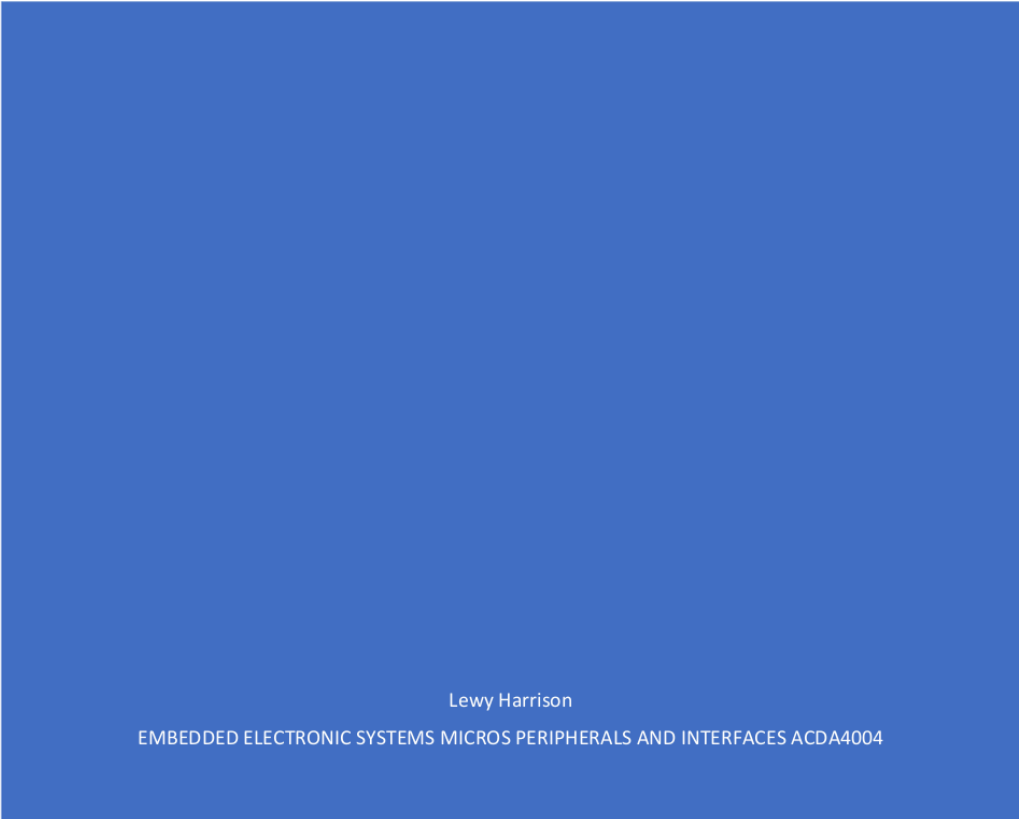
52191_Lewy_Harrison_USING_AND_PROGRAMMING_THE_7_SEGMENT_DISPLAY_PROTOTYPE_CIRCUIT_621295_407056532.docx
(473.23K)

Word count: 1623

Character count: 7833



USING AND PROGRAMMING THE 7 SEGMENT DISPLAY PROTOTYPE CIRCUIT.

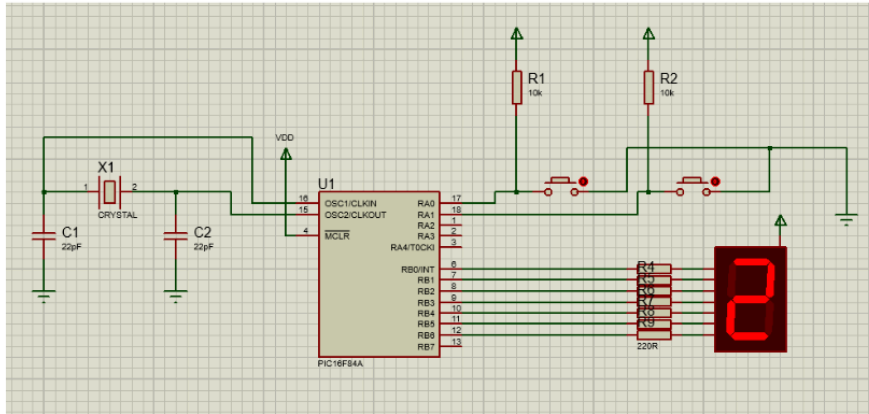


Lewy Harrison

EMBEDDED ELECTRONIC SYSTEMS MICROS PERIPHERALS AND INTERFACES ACDA4004

Section 1

Figure 1. Shown below is the proteus simulation of the original circuit.



The main component of this circuit is the PIC16F84A chip. This chip uses the Harvard architecture, which separates program memory from data memory for faster processing. It has 68 bytes of data memory (RAM) and operates at up to 20 MHz clock speed. It uses a voltage range of 2.0V to 5.5V, making it suitable for battery-powered applications [1]. Although it's an older model and has been surpassed by more advanced PIC microcontrollers, this PIC remains popular for its simplicity and reliability.

In this circuit, the microcontroller reads inputs from the switches, controls the 7-segment display by sending output signals to its pins and runs the programmed logic to determine what number to display.

The Crystal Oscillator provides a clock signal required for the microcontroller's operation. It uses the two capacitors either side of it to stabilize itself and filter noise.

The pull-up resistors' (R1 and R2) function is to maintain a high logic level on the switch input pins when the switches are not pressed. When the switch is closed, the pin reads '0' (ground) as the switch creates a direct connection to ground. The switches act as some form of user input.

The rest of the resistors are to limit the current flowing through the LEDs in the 7-segment display. This protects them from potential damage.

The 7-segment display shows numbers based on the signals from the microcontroller. A common cathode or common anode type display is used here. The 7 segments each take signal from a different pin in the B port.

Section 2

To get the circuit to display the hexadecimal characters 0-F on the 7-segment display, the first thing to do is to understand how the chip sends signals to the display. Each input on the display corresponds to a segment on the display.

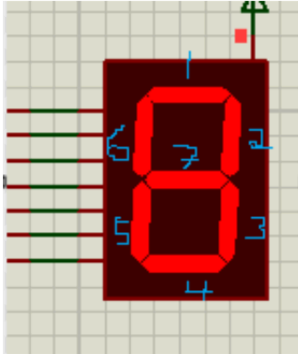


Figure 2: shows the 7-segment display labelled 1-7

The segment labelled one will correspond to the top wire input. The segment labelled 2 corresponds to the second wire and so on. This, along with all the pins from the display connecting only to the pins in the B port on the microcontroller, means that if an 8-bit binary input or a 2-digit hex input is given to the B port, it will show only chosen segments on the display. For example, if the binary input of 0000 0100 (also is 4) is given to the B port, the 3rd pin would be putting out a low signal (as all pins are by default high), then the segment labelled 3 above will be off. Using this base, it can be easily found out what binary string or hex digits are needed to display each number 0-9 and then letter a-f.

Unit	Binary	Hex
0	0100 0000	0x40
1	0111 1001	0x79
2	0010 0100	0x24
3	0011 0000	0x30
4	0001 1001	0x19
5	0001 0010	0x12
6	0000 0010	0x02
7	0111 1000	0x78
8	0000 0000	0x00
9	0001 0000	0x10
a	0000 1000	0x08
b	0100 0110	0x03
c	0100 0110	0x46
d	0010 0001	0x21
e	0000 0110	0x06
f	0000 1110	0x0E

Figure 3 A table that shows the conversion for each hexadecimal digit to be show correctly on the 7-segment display.

Section 3

Task 1

```
1 ;-----
2 ; DEFINITIONS
3 ;-----
4 #include p18f44a.inc           ; Include register definition file
5
6 ; Define PORTB register address
7 portb equ 0x06
8 ; Define TRISB register address
9 trisb equ 0x06
10
11 ; Define STATUS register address
12 status equ 0x03
13 ; Define bit 3 of STATUS register (RP0)
14 RP0 equ 3
15 ; Define bit 4 of STATUS register (RP1)
16 RP1 equ 4
17
18 ;-----
19 ; VARIABLES
20 ;-----
21 count equ 0x0C                ; Define memory location for "Count"
22 nCount equ 0x0D              ; Define memory location for "nCount"
23 mCount equ 0x0E              ; Define memory location for "mCount"
24
25 ;-----
26 ; RESET and INTERRUPT VECTORS
27 ;-----
28 ; Reset Vector
29 RST code 0x00                ; Define reset vector location
30 goto Start                   ; Jump to "Start" label after reset
31
32 ;-----
33 ; CODE SEGMENT
34 ;-----
35 ; Define code segment
36 Start
37 ; Clear RP0 bit to select bank 0
38 bcf status,RP0
39 ; Clear RP1 bit to select bank 0
40 bcf status,RP1
41 ; Set RP0 bit to select bank 1
42 movlw 0x00
43 movwf trisb
44 ; Set PORTB pins as output (TRISB = 0x00)
45 bcf status,RP0
46 ; Clear RP0 bit to return to bank 0
47 clrf portb
48 ; Clear "Count" variable
49 clrf count
50
51 ; Load W register with 0x00 (pattern for display)
52 movlw 0x00
53 movwf portb
54 ; Call delay subroutine "pause"
55 call pause
56
57 ;-----
58 ; same as above, just repeated for each digit 0-9 and a-f
59 ;-----
60 movlw 0x79
61 movwf portb
62 call pause
63
64 movlw 0x34
65 movwf portb
66 call pause
67
68 movlw 0x30
69 movwf portb
70 call pause
71
72 movlw 0x13
73 movwf portb
74 call pause
75
76 movlw 0x12
77 movwf portb
78 call pause
79
80 movlw 0x09
81 movwf portb
82 call pause
83
84 movlw 0x08
85 movwf portb
86 call pause
87
88 movlw 0x06
89 movwf portb
90 call pause
91
92 movlw 0x03
93 movwf portb
94 call pause
95
96 movlw 0x00
97 movwf portb
98 call pause
99
100 ;-----
101 ; Repeat the process indefinitely
102 ;-----
103 goto numbers
104
105 ;-----
106 ; Load W register with 0xFF (initial value for mCount)
107 ; Store 0xFF in mCount
108 movlw 0xFF
109 movwf mCount
110
111 ; Load W register with 0xFF (initial value for nCount)
112 ; Store 0xFF in nCount
113 movlw 0xFF
114 movwf nCount
115
116 ; Decrement nCount and skip next instruction if zero
117 decfz nCount,f
118 gotoz decn
119 ; Loop until nCount reaches zero
120
121 ; Decrement nCount and skip next instruction if zero
122 decfz nCount,f
123 gotoz loadn
124 ; Loop until nCount reaches zero
125
126 ; Return from subroutine
127 return
128
129 ;-----
130 ; END
131 ; End of program
132
```

Figure 4-6 shows the code required to complete task 1

Pseudocode

Definitions – defines necessary header files for the PIC16F84A microcontroller, defines constraints for key registers and their addresses (PORTB, TRISB, STATUS)

Variables - defines memory locations for variables used in the program (count, nCount, mCount)

Reset and interrupt vectors – defines the reset vector to jump to the 'start' section of the program.

Code segment:

- Configure the microcontroller – set the STATUS register to select the correct bank, configure PORTB as output by writing 0x00 to TRISB and clears PORTB and initializes variables.
- Enter the main loop 'numbers' – load a series of predefined values into PORTB to display the hex digits on the display, after each value call the 'pause' subroutine to introduce a delay and then repeat the sequence indefinitely by jumping back to the 'numbers' subroutine.
- Define the 'pause' subroutine – use nested loops to create a delay. The outer loop decrements mCount until it reaches 0, and the inner loop decrements nCount until it reaches 0. Then returns control to the calling function after the delay is complete.

Task 2

```

1  #include p16f84a.inc
2
3  ; Define registers
4  portB equ 0x06           ; PORTB address (7-segment display)
5  trisb equ 0x06           ; TRISB register
6  portA equ 0x05           ; PORTA address (Buttons)
7  trisa equ 0x05           ; TRISA register
8  status equ 0x03         ; STATUS register
9  RP0 equ 5               ; RP0 bit in STATUS register
10
11 dCount equ 0x0D          ; Temporary register for delay
12
13 ; Reset vector
14 RST code 0x00
15 goto Start              ; Jump to start of program
16
17 ; Main program
18 PGM code
19
20 Start:
21   ; Setup ports
22   bsf status,RP0        ; Switch to Bank 1
23   movlw 0x00             ; Set PORTB as output
24   movwf trisb
25   movlw 0x0F             ; Set lower 4 bits of PORTA as input
26   movwf trisa
27   bcf status,RP0        ; Switch back to Bank 0
28
29   ; Initialize variables
30   clrf portB             ; Ensure 7-segment display is OFF
31
32 main_loop:
33   movf portA, W          ; Read button states
34
35   ; Check Button 1 (RA0)
36   btfss portA, 0         ; If RA0 is pressed, skip next instruction
37   call display_L         ; Call display_L subroutine
38
39   ; Check Button 2 (RA1)
40   btfss portA, 1         ; If RA1 is pressed, skip next instruction
41   call display_r         ; Call display_r subroutine
42
43   goto main_loop        ; Repeat main Loop
44
45 display_L:
46   movlw 0x47             ; Load value for "L" on 7-segment display
47   movwf portB            ; Write to PORTB
48   call delay             ; Debounce delay
49   return
50
51 display_r:
52   movlw 0x4E             ; Load value for "r" on 7-segment display
53   movwf portB            ; Write to PORTB
54   call delay             ; Debounce delay
55   return
56
57 ; Simple delay for debounce
58 delay:
59   movlw 0xFF             ; Load delay counter
60   movwf dCount
61
62 delay_loop:
63   decfsz dCount, F       ; Decrement dCount
64   goto delay_loop        ; Loop until dCount reaches 0
65   return                ; Return from delay
66
67 END

```

Figures 7 and 8 show the code for task 2, which displays an L when one button is pressed and an R when the other button is pressed.

Pseudocode

BEGIN Program

INITIALIZE:

1. Define required constants and memory locations:

- PORTB for controlling the 7-segment display.
- PORTA for reading button inputs.
- STATUS register bits for switching memory banks.
- Variables for LED state and debounce delay.

2. Configure ports:

- Set PORTB as output for the 7-segment display.
- Set PORTA as input for buttons.
- Initialize LED_STATE to 0 (indicating no active display).
- Clear PORTB to ensure no initial output on the display.

MAIN LOOP:

1. Read the button states from PORTA.
2. Check if Button 1 (RA0) is pressed:
 - If pressed, call the subroutine `show\_L`.
3. Check if Button 2 (RA1) is pressed:
 - If pressed, call the subroutine `show\_R`.
4. Repeat the main loop indefinitely.

SUBROUTINE: show_L

1. Load the 7-segment display pattern for the letter 'L' into the W register.
2. Output this pattern to PORTB to display 'L'.
3. Update LED_STATE to indicate 'L' is being displayed.
4. Call the `delay` subroutine to provide a debounce delay.
5. Return to the main loop.

SUBROUTINE: show_R

1. Load the 7-segment display pattern for the letter 'r' into the W register.
2. Output this pattern to PORTB to display 'r'.
3. Update LED_STATE to indicate 'r' is being displayed.
4. Call the `delay` subroutine to provide a debounce delay.
5. Return to the main loop.

SUBROUTINE: delay

1. Initialize a counter for the debounce delay.
2. Create a loop that decrements the counter until it reaches zero.

3. Return to the calling function after the delay is completed.

END Program

Task 3

```
1 #include p16f84a.inc
2
3 ; Define registers
4 portB equ 0x06 ; PORTB address (LEDs)
5 trisB equ 0x06 ; TRISB register
6 portA equ 0x05 ; PORTA address (Buttons)
7 trisA equ 0x05 ; TRISA register
8 status equ 0x03 ; STATUS register
9 RPB equ 5 ; RPB bit in STATUS register
10
11 ; Variables
12 LED_STATE equ 0x0C ; Variable to hold LED state
13 dCount equ 0x0D ; Temporary register for delay
14
15 ; Reset vector
16 RST code 0x0 ; Jump to start of program
17 goto Start
18
19 ; Main program
20 PGM code
21
22 Start:
23 ; Setup ports
24 bcf status,RPB ; Switch to Bank 1
25 movlw 0x00 ; Set PORTB as output
26 movwf trisB
27 movlw 0x0F ; Set lower 4 bits of PORTA as input
28 movwf trisA
29 bcf status,RPB ; Switch back to Bank 0
30
31 ; Initialize variables
32 clrf LED_STATE ; Set initial LED state to OFF
33 clrf portB ; Ensure all LEDs are OFF
34
35 main_loop:
36 movf portA,W ; Read button states
37
38 ; Check ON button (RA0)
39 btfss portA,0 ; If RA0 is pressed, skip next instruction
40 call turn_on ; Call turn_on subroutine
41
42 ; Check OFF button (RA1)
43 btfss portA,1 ; If RA1 is pressed, skip next instruction
44 call turn_off ; Call turn_off subroutine
45
46 goto main_loop ; Repeat main loop
47
48 turn_on:
49 movlw 0xFF ; Load value to turn ON all LEDs
50 movwf portB ; Write to PORTB
51
52 ; Set LED_STATE to ON
53 movlw 0x01
54 movwf LED_STATE
55 call delay ; Debounce delay
56 return
57
58 turn_off:
59 clrf portB ; Clear PORTB to turn OFF all LEDs
60 clrf LED_STATE ; Set LED_STATE to OFF
61 call delay ; Debounce delay
62 return
63
64 ; Simple delay for debounce
65 delay:
66 movlw 0xFF ; Load delay counter
67 movwf dCount
68
69 delay_loop:
70 decfsz dCount,F ; Decrement dCount
71 goto delay_loop ; Loop until dCount reaches 0
72 return ; Return from delay
73
74 END
```

Pseudocode

BEGIN Program

INITIALIZE:

1. Include the required header file for the PIC16F84A microcontroller.
2. Define constants and memory locations:
 - 'PORTB' for controlling LEDs.
 - 'PORTA' for reading button inputs.

- `TRISB` and `TRISA` for port direction configuration.
- Variables `LED\_STATE` to track the current state of LEDs and `dCount` for delay timing.

3. Configure ports:

- Set `PORTB` as output.
- Set the lower 4 bits of `PORTA` as input.

4. Initialize variables:

- Clear `PORTB` to ensure LEDs are initially OFF.
- Clear `LED\_STATE` to indicate the OFF state.

MAIN LOOP:

1. Read the state of buttons from `PORTA`.

2. Check if Button 1 (RA0) is pressed:

- If pressed, call the `turn\_on` subroutine.

3. Check if Button 2 (RA1) is pressed:

- If pressed, call the `turn\_off` subroutine.

4. Repeat the main loop indefinitely.

SUBROUTINE: turn_on

1. Load `0xFF` into `PORTB` to turn ON all LEDs.

2. Set `LED\_STATE` to `0x01` to indicate the ON state.

3. Call the `delay` subroutine to debounce the button.

4. Return to the main loop.

SUBROUTINE: turn_off

1. Clear `PORTB` to turn OFF all LEDs.

2. Clear `LED\_STATE` to indicate the OFF state.

3. Call the `delay` subroutine to debounce the button.

4. Return to the main loop.

SUBROUTINE: delay

1. Initialize the delay counter `dCount` to `0xFF`.

2. Create a loop:

- Decrement `dCount` until it reaches zero.

3. Return to the calling subroutine after the delay.

END Program

Task 4

```
1  #include p16f84a.inc
2
3  ; Define registers
4  portB equ 0x06
5  triB equ 0x06
6  portA equ 0x05
7  triA equ 0x05
8  status equ 0x03
9  RP0 equ 5
10
11 ; Variables
12 count equ 0x0c ; Counter value
13 lastState equ 0x0d ; Previous button state
14 dCount equ 0x0e ; For delay
15
16 ; Reset vector
17 RST code 0x0b
18 goto Start
19
20 ; Main program
21 PGM code
22 Start
23 ; Setup ports
24 bsf status,RP0 ; Bank 1
25 movlw 0x00
26 movwf triB ; PORTB = outputs
27 movlw 0x0f
28 movwf triA ; PORTA Lower = inputs
29 bcf status,RP0 ; Bank 0
30
31 ; Initialize
32 movlw 0x05
33 movwf count
34 movwf lastState ; Initialize button state
35
36 main_loop
37 movf portA,W ; Get current button state
38 xorwf lastState,W ; Compare with last state
39 andlw 0x03 ; Only care about first two buttons
40 btfsc STATUS,Z ; Skip if buttons changed
41 goto display ; No change, just update display
42
43 movf portA,W ; Get button state again
44 movwf lastState ; Save for next time
45
46 btfss portA,0 ; Skip if left button pressed
47 call left_button
48
49 btfss portA,1 ; Skip if right button pressed
50
```

```

51     call right_button
52
53 display
54     movf count,W           ; Get current count
55     call table             ; Convert to display pattern
56     movwf portB           ; Output to display
57     call delay             ; Small delay
58     goto main_loop        ; Repeat
59
60 ; Increment counter (max 9)
61 right_button
62     movlw .9               ; Check if count = 9
63     subwf count,W
64     btfsc STATUS,Z
65     return                ; At max, don't increment
66     incf count,F           ; Increment
67     return
68
69 ; Decrement counter (min 0)
70 left_button
71     movf count,W           ; Check if count = 0
72     btfsc STATUS,Z
73     return                ; At min, don't decrement
74     decf count,F           ; Decrement
75     return
76
77 ; Delay for debounce
78 delay
79     movlw 0xff
80     movwf dCount
81 delay_loop
82     decfsz dCount,F
83     goto delay_loop
84     return
85
86 ; Display patterns for 7-segment (0-9)
87 table
88     addwf PCL
89     retlw 0xC0             ; 0
90     retlw 0xF9             ; 1
91     retlw 0x64             ; 2
92     retlw 0xB0             ; 3
93     retlw 0x99             ; 4
94     retlw 0x92             ; 5
95     retlw 0x82             ; 6
96     retlw 0xF8             ; 7
97     retlw 0x80             ; 8
98     retlw 0x90             ; 9
99
100    END

```

Pseudocode

BEGIN Program

INITIALIZE:

1. Include the required header file for the PIC16F84A microcontroller.

2. Define constants and memory locations:

- `PORTB` for controlling the 7-segment display.
- `PORTA` for reading button inputs.
- `TRISB` and `TRISA` for port direction configuration.
- Variables `count` to hold the current counter value, `lastState` to track the previous button state, and `dCount` for delay timing.

3. Configure ports:

- Set `PORTB` as output.
- Set the lower 4 bits of `PORTA` as input.

4. Initialize variables:

- Set `count` to 5 as the starting value.
- Set `lastState` to the initial button state read from `PORTA`.

MAIN LOOP:

1. Read the current state of buttons from `PORTA`.
2. Compare the current state with the `lastState`:
 - If no change is detected, skip to the display update.
 - If a change is detected, update `lastState` with the new state.
3. Check if Button 1 (RA0) is pressed:
 - If pressed, call the `left\_button` subroutine to decrement the counter.
4. Check if Button 2 (RA1) is pressed:
 - If pressed, call the `right\_button` subroutine to increment the counter.
5. Update the 7-segment display:
 - Call the `table` subroutine to convert the `count` value into the corresponding display pattern.
 - Output the pattern to `PORTB`.
6. Call the `delay` subroutine to provide a small delay for debounce.
7. Repeat the main loop indefinitely.

SUBROUTINE: right_button

1. Check if `count` is at its maximum value (9):
 - If it is, return without incrementing.
2. Increment `count` by 1.
3. Return to the main loop.

SUBROUTINE: left_button

1. Check if `count` is at its minimum value (0):
 - If it is, return without decrementing.
2. Decrement `count` by 1.
3. Return to the main loop.

SUBROUTINE: delay

1. Initialize the delay counter `dCount` to `0xFF`.
2. Create a loop:
 - Decrement `dCount` until it reaches zero.
3. Return to the calling subroutine after the delay.

SUBROUTINE: table

1. Use the value in `count` to calculate the address offset for the lookup table.
2. Retrieve the corresponding 7-segment display pattern for the current `count` value.
3. Return the pattern to the calling function.

END Program

USING AND PROGRAMMING THE 7 SEGMENT DISPLAY PROTOTYPE CIRCUIT

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

47 /100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

GRADING FORM: GJ_2019

LEWY HARRISON

233

ENGAGEMENT WITH LITERATURE

 Only one reference source, and it wasn't even generated in a visible list! **5**

KNOWLEDGE AND UNDERSTANDING

 Far more explanation is needed for the component/circuit operation. The lack of EWL severely limits this mark **50**
LED code generation explanation is fair however.

COGNITIVE AND INTELLECTUAL

 The core programming tasks have been attempted reasonably well, although they mostly use a very standard approach (apart from the last one which does use an array implementation) **60**

PRACTICAL APPLICATION

 Circuit doesn't include the additional components (reg and sounder). **60**

TRANSFERABLE SKILLS

 Listing FAR too small to read easily. More time could have been spent in producing a more professional layout. **58**
Needed more images etc in the explanation section.