

MARKING CRITERIA AND STUDENT FEEDBACK

This section details the assessment criteria. The *extent* to which these are demonstrated by you determines your mark. The marks available for each criterion are shown. Lecturers will use the space provided to comment on the achievement of the task(s), including those areas in which you have performed well and areas that would benefit from development/improvement.

Common Assessment Criteria Applied	Marks available	Marks awarded
1. Research-informed Literature Extent of research and/or own reading, selection of credible sources, application of appropriate referencing conventions.		
you have clearly read well. whilst you have included references, this could be more extensive.	10	8
2. Knowledge and Understanding of Subject Extent of knowledge and understanding of concepts and underlying principles associated with the discipline.		
you now quite a good understanding although there are still some areas for improvement.	30	20
3. Analysis Analysis, evaluation and synthesis; logic, argument and judgement; analytical reflection; organisation of ideas and evidence		
There is clear evidence of analysis	10	8
4. Practical Application and Deployment Deployment of methods, materials, tools and techniques; application of concepts; formulation of innovative and creative solutions to solve problems.		
The report contains quite good narrative and flow charts. However the flow charts should be to normal standard, and contain errors. The programmes are quite close however there are errors which more extensive testing would reveal. It is a pity you did not complete the BCD display work.	45	35
5. Skills for Professional Practice Attributes in professional practice: individual and collaborative working; deployment of appropriate media; presentation and organisation.		
Overall a very good report	5	4

Assignment Mark Overall (Assessment marks are subject to ratification at the Exam Board. These comments and marks are to give feedback on module work and are for guidance only until they are confirmed.)

Late Submission Penalties (tick if appropriate)

Up to one-week late

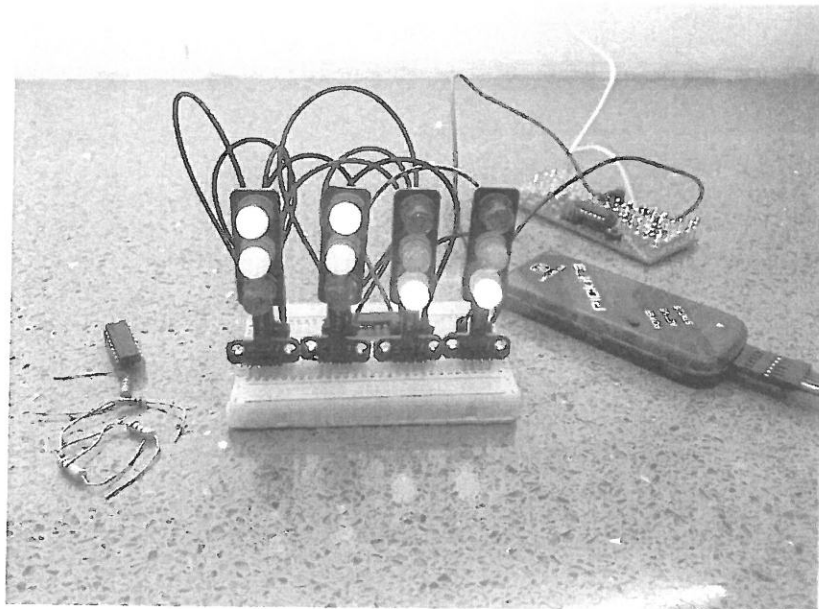
Over one week late

75 %

Microcontrollers

Practical

Assignment



Nicholas Davies

Tycoch College, Swansea

Year 2, HNC Electronic Engineering

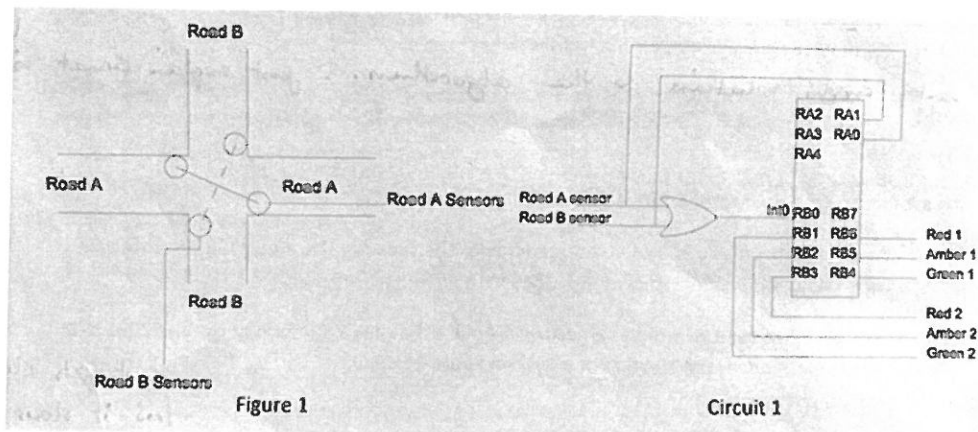
Contents Page

3.	Traffic Lights Part A
4.	Traffic Lights assembly code
7.	Calculating TMRO using Excel
9.	Traffic lights Part B + Algorithm (inc. PIR sequence)
10.	Traffic Lights assembly code including PIR detectors
14.	Traffic Light possible improvements
16.	Pulse Width Modulation for a DC motor control module
17.	PWM with BCD assembly code
22.	Speed setting on BCD
24.	BCD Algorithm
25.	Direction switch
25.	Overriding enable control
25.	Possible adjustments
26.	References

Traffic Lights Part A

The brief is to provide traffic flow control at an intersection of two roads using a predetermined sequence of lights shown in Figure 1 below.

This sequence of lights is to repeat indefinitely controlled by a PIC 16F84A Microchip with programming designed and debugged through the assembly program MPLAB v. 4.05



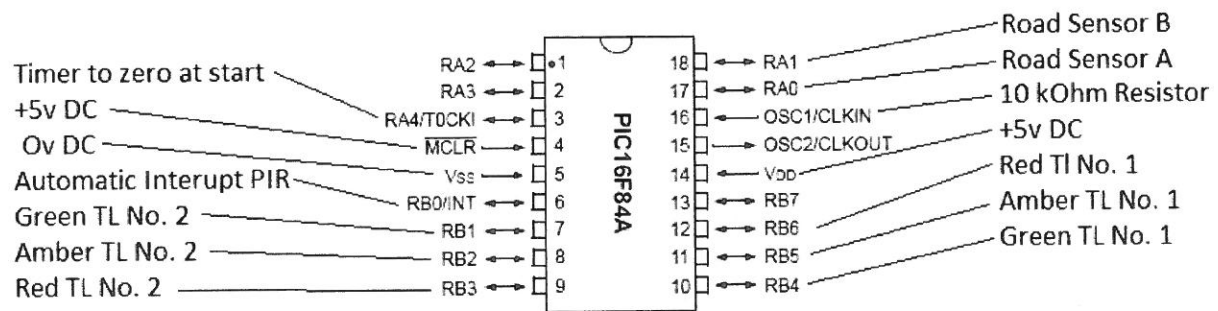
		Road 1			Road 2		
Sequence No.	Duration (Seconds)	Red 1	Amber 1	Green 1	Red 2	Amber 2	Green 2
1	40						
2	5						
3	5						
4	1						
5	40						
6	5						
7	5						
8	1						

The microcontroller chip used in this project is a 16F84A. It has many features making it suitable for use with this project.

The more basic 8 bit which has been setup (or equated) to Hexidecimal is more than sufficient for this level of assembly programming. It has an 8 bit architecture with an on board timer and interrupt functionality and the TMRO pin has an 8 bit programmable prescaler for being able to reduce the number of lines required to write a delay.

More importantly for simplicity it is able to provide a high current on the port B outputs to directly drive the LED outputs

It has 18 pins (or legs) as shown in the diagram below.



Assembly code

```

;
; Author: Nick B Davies.
;
; Date: February 2018
;
; Version: 4.05
;
; Title: Traffic light sequence
;
; Description: No control measures ;

```

```
include <p16f84a.inc>
```

```
_config _RC_OSC & _WDT_OFF & _PWRTE_ON
```

```
radix hex
```

```
; equates
```

```
portA equ 0x05
```

```
portB equ 0x06
```

```
count equ 0x0C
```

```
countx equ 0x0D
```

```
cycle equ 0x0E
```

```
pir equ 0x0F
```

```
countxx equ 0x10
```

```
timer equ 0x11
```

```
prescalar equ 0x12
```

```
org 0x000
```

```
goto start
```

```
org 0x04 ; dummy interrupt routine
```

```

        btfss    INTCON, 1
        goto     ISR
        retfie

start    bsf      0x03, 5          ; go to bank 1
        movlw    0xFF             ; b11111111
        movwf    0x85             ; portA all input
        movlw    0x01             ; b00000001
        movwf    0x86             ; portB outputs apart from RB0 input
        movlw    0xC7             ; binary 11000111 C7- maximum prescale, interrupt
                                     ; on rising edge, assigned to TMR0
        movwf    OPTION_REG       ; load into timer setup register
        movlw    0xB0             ; 10110000
        movwf    INTCON           ; global, timer and RB0 interrupt enabled
        bcf      0x03, 5          ; go to bank 0
        clrf     portB            ; initialize portA outputs to zero
        clrf     TMR0             ; initialize the timer to zero
                                     ; main
sq1      movlw    0x02             ; identify cycle
        movwf    cycle
        movlw    0x42             ; sq1 1red 2green
        movwf    portB
        call     delay40
sq2      movlw    0x00             ; identify cycle
        movwf    cycle
        movlw    0x44             ; sq2 1red 2amber
        movwf    portB
        call     delay5
sq3      movlw    0x68             ; sq3 1red/amber 2red
        movwf    portB
        call     delay5
sq4      movlw    0x01             ; identify cycle
        movwf    cycle

```

```

        movlw      0x28                ; sq4 1amber 2red
        movwf      portB
        call       delay1
sq5     movlw      0x18                ; sq5 1green 2red
        movwf      portB
        call       delay40
sq6     movlw      0x00                ; identify cycle
        movwf      cycle
        movlw      0x28                ; sq6 1amber 2red
        movwf      portB
        call       delay5
sq7     movlw      0x46                ; sq7 1red 2red/amber
        movwf      portB
        call       delay5
sq8     movlw      0x02                ; identify cycle
        movwf      cycle
        movlw      0x44                ; sq8 1red 2amber
        movwf      portB
        call       delay1
        goto       sq1                ; repeat forever

;FUNCTIONS-----
;timer - count to 18 with tmr0 preset to 39, 256 prescalar with 4MHz clock for 1 sec
delay1   MOVLW      0x06    ; count to 18 hex 12
        MOVWF      count
        MOVLW      0x27    ; set timer to 39
        MOVWF      TMR0
        BCF        INTCON, 2    ; clear over flow flag bit 2
        BTFSS     INTCON, 2    ; wait for flag set
        GOTO       $-1
        DECFSZ     count
        GOTO       $-6
        RETURN
delay5   movlw      0x05    ; call delay_1 5 times

```

```

        movwf countx
loop2    call    delay1
        decfsz  countx, 1
        goto   loop2
        return
delay40  movlw   0x28      ; call delay_1 5 times
        movwf  countx
loop3    call    delay1
        decfsz  countx, 1
        goto   loop3
        return
end

```

This basic traffic light sequence has been set up using the internal timer of the 16f84a microcontroller chip allowing delay routines to be accurately measured without necessarily holding up the main program flow with the calling of delay routines and sub routines counting multiple loops.

Working on the assumption that the chip has a 4Mhz frequency the time was calculated for a second on an Excel spreadsheet

	A	B	C	D	E
1	time	clock (osc)	tmr0	prescaler	count
2	1	4000000	39	256	18.0012
3					
4					
5					
6					
7					

The TMR0 has been set to 39 to give a more accurate count time

And using this following formula. It is divided by 4 to get 1Mhz as there is not an external oscillator- which is why the 10KΩ resistor is used.

$$f_{out} = \frac{f_{clk}}{4 * \text{Prescaler} * (256 - \text{TMR0}) * \text{Count}} \quad \text{where} \quad T_{out} = \frac{1}{f_{out}}$$

PIC TIMER0 formula for internal clock

F_{out}– The output frequency after the division.

T_{out} – The Cycle Time after the division.

4 - The division of the original clock (4 MHz) by 4, when using internal crystal as clock (and not external oscillator).

Count - A numeric value to be placed to obtain the desired output frequency - *Fout*.
(256 - *TMR0*) - The number of times in the timer will count based on the register *TMR0*.

This one second delay is then called, once, five or forty times depending on which stage of the sequence the assembly program is at. Example below:

```
delay5      movlw    0x05      ; call delay_1 5 times
            movwf    countx
loop2        call     delay1
            decfsz    countx, 1
            goto     loop2
            return
```

Here the *Delay_1* is called 5 times. It then runs through Loop 2 and each time it counts down (decrements) from 5, 4, 3, 2, 1. When zero is reached it skips the GOTO command for LOOP 2 and advances to RETURN- it then returns to the point of program where the DELAY was called.

Eg. call delay5

The section of code below shows how the A and B Ports have been set up

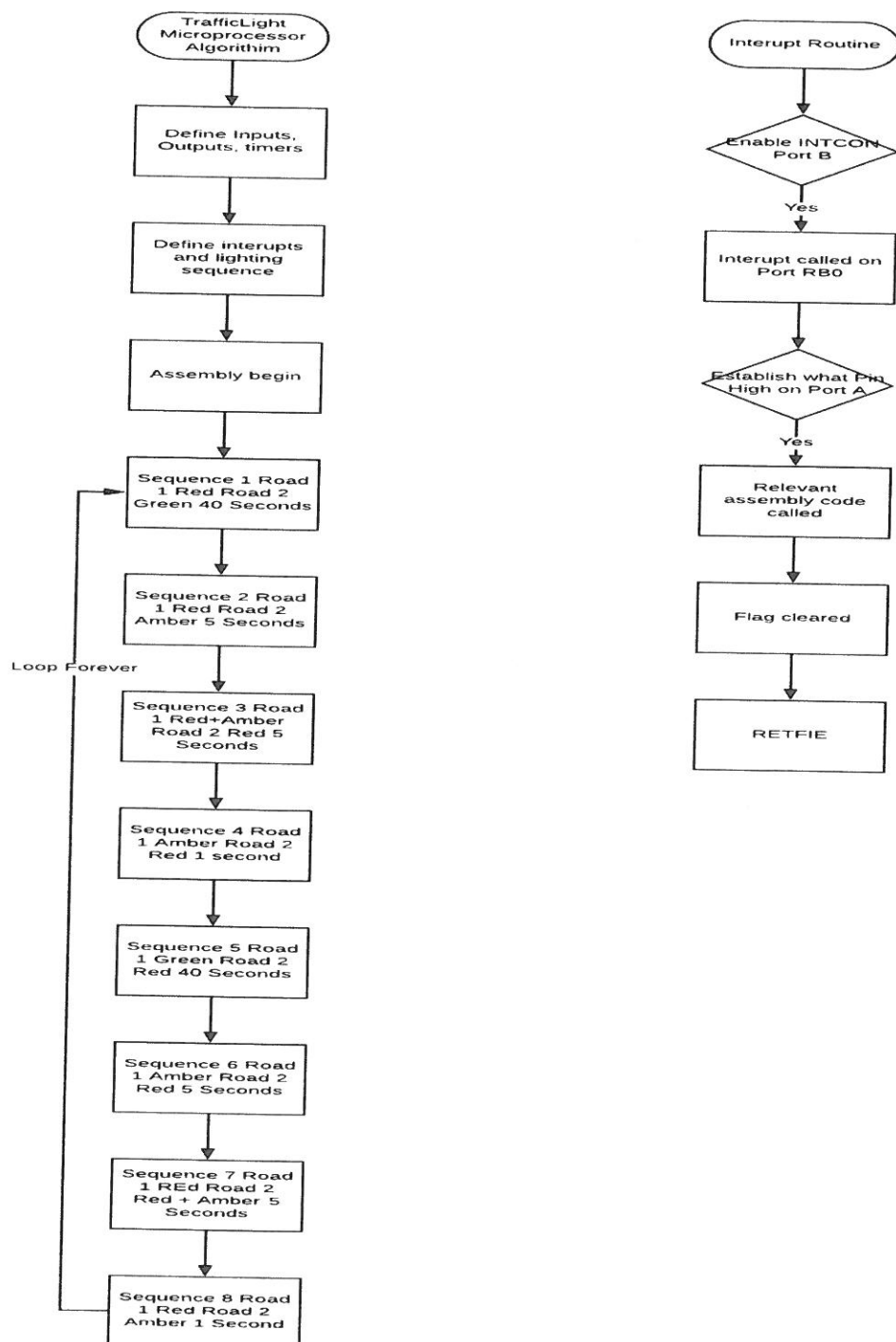
```
movlw       0xFF              ; b11111111
movwf       0x85              ; portA all input
movlw       0x01              ; b00000001
movwf       0x86              ; portB outputs apart from RB0 input
```

As it explains the Port A were all set up as inputs hence the binary b11111111 – all 8 BITS have been set to 1's

Port B have all been set to Outputs with the exception of RB0 which is being used as the global interrupt for the next part of the assembly program- This has been set now for convenience. As above in binary the code is b00000001 and for the Hexidecimal setting 0x01.

Traffic lights Part B

To make the traffic light operation more realistic and allow the free flow of cars if nothing is coming from the adjacent road PIR (passive infrared detectors) are to be fitted to the lights. They are intended to monitor the flow and should there be no traffic, priority is given to the adjacent road. The flow chart below shows the normal intended sequence and the global interrupts that the PIRS will initiate.



The assembly code for this program required three inputs. Two on Port A and one on Port B.

The input on Port A RA0 comes from the input logic (Input A) of the OR gate originating from the Road A PIR and RA1 comes from Road B PIR (Input B). The two roads have different inputs so that the program can distinguish between the two and give priority.

The input from Port B, RBO activates the global interrupt BCF INTCON, 3 (Bit 3) and sets this to high to enable all unmasked interrupts. The external interrupt flag is then cleared (Bit 1).CLEAR?????

When the 40 seconds has passed and the PIR's output is low, BSF INTCON, 3 is enabled to allow it to monitor constantly in the back ground.

The full assembly code is written below.

```
.....
; Author: Nick B Davies. ;
; Date: March 2018 ;
; Version: 4.05 ;
; Title: Traffic light sequence ;
; Description: PIR on lights ;
.....

include <p16f84a.inc>

__config _RC_OSC & _WDT_OFF & _PWRTE_ON

radix hex

;-----
; equates

portA equ 0x05

portB equ 0x06

count equ 0x0C

countx equ 0x0D

cycle equ 0x0E

pir equ 0x0F

countxx equ 0x10

timer equ 0x11

prescalar equ 0x12

;-----

org 0x000

goto start
```

```

    org 0x04          ; dummy interrupt routine

    btfss    INTCON, 1
    call     ISR
    retfie    ✓

start    bsf    0x03, 5          ; go to bank 1
        movlw  0xFF              ; b11111111
        movwf  0x85              ; portA all input
        movlw  0x01              ; b00000001
        movwf  0x86              ; portB outputs apart from RB0 input
        movlw  0xC7              ; binary 11000111 C7- maximum prescale, interrupt on rising edge,
                                   ; assigned to TMR0

        movwf  OPTION_REG        ; load into timer setup register
        movlw  0xB0              ; 10110000
        movwf  INTCON            ; global, timer and RB0 interrupt enabled

        bcf    0x03, 5          ; go to bank 0
        clrf   portB            ; initialize portA outputs to zero
        clrf   TMR0             ; initialize the timer to zero main

sq1      movlw  0x02              ; identify cycle
        movwf  cycle
        movlw  0x42              ; sq1 1red 2green
        movwf  portB
        call   delay40

sq2      movlw  0x00              ; identify cycle
        movwf  cycle
        movlw  0x44              ; sq2 1red 2amber
        movwf  portB
        call   delay5

sq3      movlw  0x68              ; sq3 1red/amber 2red
        movwf  portB
        call   delay5

sq4      movlw  0x01              ; identify cycle
        movwf  cycle
        movlw  0x28              ; sq4 1amber 2red

```

```

        movwf portB
        call delay1
sq5      movlw 0x18      ; sq5 1green 2red
        movwf portB
        call delay40
sq6      movlw 0x00      ; identify cycle
        movwf cycle
        movlw 0x28      ; sq6 1amber 2red
        movwf portB
        call delay5
sq7      movlw 0x46      ; sq7 1red 2red/amber
        movwf portB
        call delay5
sq8      movlw 0x02      ; identify cycle
        movwf cycle
        movlw 0x44      ; sq8 1red 2amber
        movwf portB
        call delay1
        goto sq1        ; repeat forever

;FUNCTIONS-----
;timer - count to 18 with tmr0 preset to 39, 256 prescalar with 4MHz clock for 1 sec
delay1    MOVLW 0x06      ; count to 18 hex 12
          MOVWF count
          MOVLW 0x27      ; set timer to 39
          MOVWF TMR0
          BCF     INTCON, 2 ; clear over flow flag bit 2
          BTFSS  INTCON, 2 ; wait for flag set
          GOTO   $-1
          DECFSZ count
          GOTO   $-6
          RETURN
delay5    movlw 0x05      ; call delay_1 5 times
          movwf countx

```

```

loop2    call  delay1
         decfsz countx, 1
         goto  loop2
         return

delay40  movlw  0x28      ; call delay_1 5 times
         movwf  countx

loop3    call  delay1
         decfsz countx, 1
         goto  loop3
         return

ISR      BCF     INTCON, 7      ; Disable all interrupts inside interrupt service routine
         BCF     INTCON, 1      ; Clear external interrupt flag bit
         btfss  portA, 0        ; car at road A
         call   roadA
         btfss  portA, 1        ; car at road B
         call   roadB
         BSF     INTCON, 7      ; Enable all interrupts on exit
         return

roadA    btfsc  cycle, 0        ; if cycle bit 0 is low(0) goto sequence 3, otherwise carry on as normal
         goto   sq3
         return

roadB    btfsc  cycle, 1
         goto   sq7
         return

end

```

with newin get Back.

The interrupt routine in more detail.

The assembly script needs to be told to monitor for an interrupt and once it flags high then to follow a CALL command before coming back and returning to the program where it left off (RETFIE).

```

btfss    INTCON, 1      ; global interrupt flag bit
         call    ISR      ; goto interrupt sub routine
         retfie           ; return from interrupt

```

The following part outlines the interrupt service routine code

```

ISR      BCF     INTCON, 7      ; Disable all interrupts inside interrupt service routine

```

```

BCF    INTCON, 1      ; Clear external interrupt flag bit

btfss  portA, 0        ; car at road A.

call   roadA

btfss  portA, 1        ; car at road B

call   roadB

BSF    INTCON, 7      ;Enable all interrupts on exit

Return

```

The BTFSS (Bit test f, skip if set), instruction tells the program to execute the next instruction if the input is low or, if the input is high the next instruction is skipped and a NOP (no operation) is executed instead. This means that if it sees a car at road A the 'call road A' will be skipped and it will then 'call road B'.

The current sequence position is tracked in the variable 'CYCLE' bits 0 and 1. From this the service routine knows where to go when an interrupt is encountered.

```

roadA    btfsc  cycle, 0 ; if cycle bit 0 is low(0) goto sequence 3, otherwise carry on as normal

        goto   sq3

        return

roadB    btfsc  cycle, 1

        goto   sq7

        return

```

The BTFSC (Bit test, set if clear), is used to skip the next line (as a NOP) if the input is low from the PIR. Once the time has passed from the delay it then returns.

Traffic Lights Part C

After simulating the assembly instructions it has become apparent that there is the potential to improve some aspects. These include:

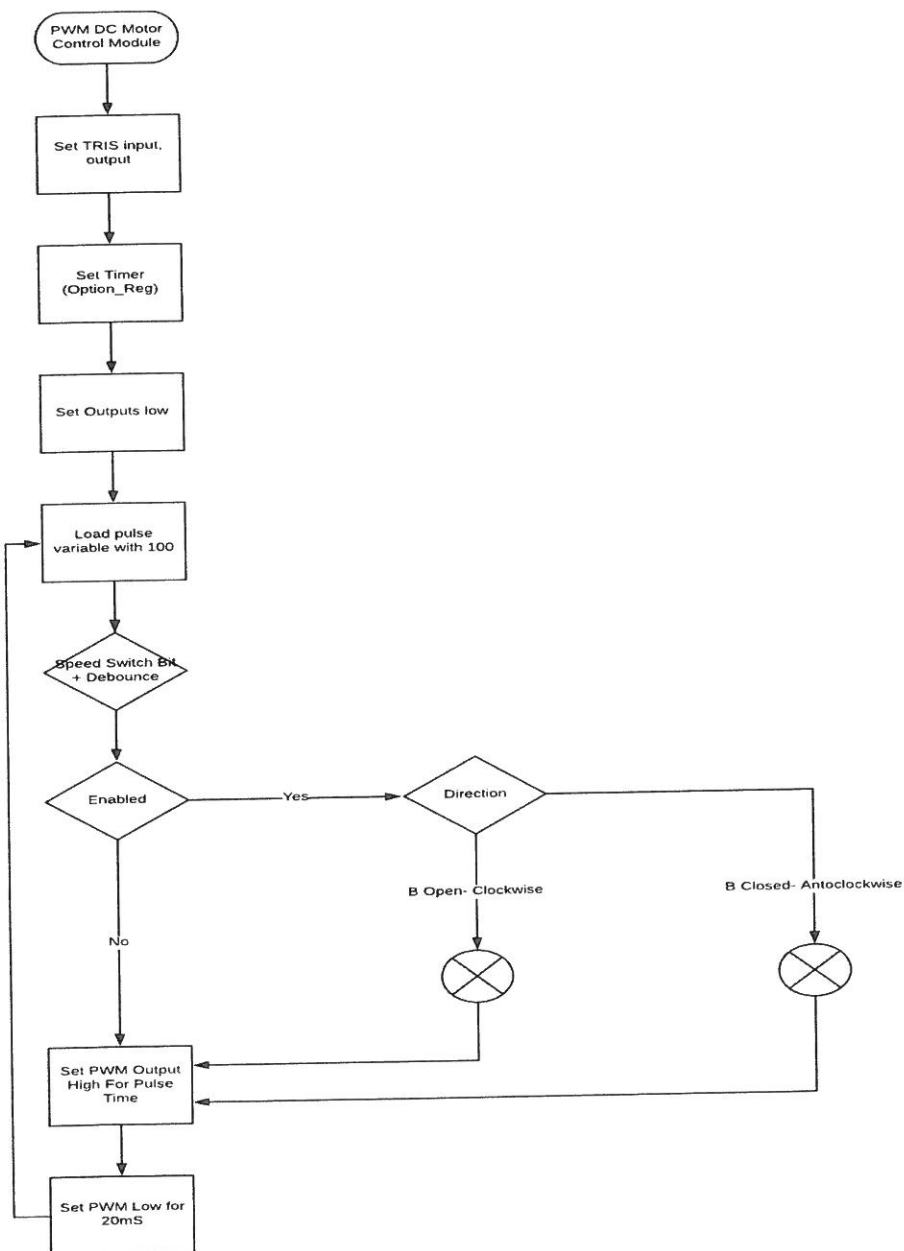
1. If both PIRs are activated setting up a priority so that which ever experiences the higher volume of traffic (through a counter) or a longer tailback (pressure sensors in the road) will have the green light first.
2. Amending the timing sequence so that it matches the ability of the traffic to safely stop and accelerate away
3. The lights will jump straight to Red on the road to be stopped. For safety the amber should illuminate first to allow motorists the opportunity to begin to slow if it is safe to do so.
4. The interrupt could cause the lights to operate in an unpredictable manner causing motorists to unexpectedly brake harshly leaving themselves or others open to collisions. Once the interrupt has cleared the lights should resume from where they are, not from the start of the routine.
5. The timer should give accurate timings as long as appropriate prescalar, count and TMR0 values are selected.

Pulse Width Modulation

A motor control module is required to operate at predefined speeds and rotational direction from the operation of three buttons. These are 'enable' (latching), 'direction' (latching) and 'speed' (momentary).

An assembly code is required to interpret the signals from the operation buttons and relay this to the DC motor control module in such a way that it can be controlled. A current speed setting is also needed on a twin BCD display that is attached to the motor control module.

The algorithm below shows the setup of the assembly code for making the Protractor disc rotate



The code for the PWM is detailed below

```
.....
;      Author:  Nick B Davies.      ;
;      Date:    March 2018         ;
;      Version: 4.05                ;
;      Title:   PWM DC motor control module ;
;      Description: PWM with BCD    ;
.....

; PIC16F84A Configuration Bit Settings
; Assembly source line config statements

#include "p16f84a.inc"

; CONFIG

; __config 0xFFFF

__CONFIG _FOSC_EXTRC & _WDTE_OFF & _PWRTE_OFF & _CP_OFF

        radix hex

;-----

;REGISTERS

;PORTS

portA      EQU    0x05
portB      EQU    0x06

;INPUTS

SPD         EQU    0x00      ;SWA is triggering RA0
DIR         EQU    0x01      ;SWB is triggering RA1
ENA        EQU    0x02      ;SWC is triggering RA2
PWM         EQU    0x00      ;PWM output

;VARIABLES

PULSE       EQU    0x0C      ;Store temporary speed value
SPEED       EQU    0x0D      ;Pulse adjustment value
COUNT      EQU    0x0E      ;Counters
COUNT1     EQU    0x0F
COUNT2     EQU    0x10
DISPLAY     EQU    0x11

;-----
```

	ORG	0	;Setting Reset vector, reset vector address is 000h in PIC16F84
	GOTO	SETUP	;Jump to RESET procedure when boot-up PIC device.
SETUP			;Program setup starts
	BSF	STATUS, 5	;Jump to bank 1 of PIC16F84 to access special registers.
	MOVLW	0x07	;Config Port A, RA0 -> RA2 are inputs.
	MOVWF	TRISA	;Set I/O configuration for PORTA
	MOVLW	0x00	;Config Port B RA7...RA0 are outputs.
	MOVWF	TRISB	;Set I/O configuration for PORTB
	MOVLW	0xC0	; Binary 11000000 C7- prescale 1:2, interrupt on rising edge, assigned to TMRO
	MOVWF	OPTION_REG	; load into timer setup register
	MOVLW	0xA0	; Binary 10100000
	MOVWF	INTCON	; Global and timer interrupt enabled
	BCF	STATUS,5	; Jump back to bank 0 of PIC.
	MOVLW	0x00	; binary 00000000
	MOVWF	DISPLAY	; Set DISPLAY default value
	MOVWF	PORTB	; Set outputs low (BCD will display 0)
	GOTO	LOOP	
LOOP			;Main program start
	MOVLW	0x64	;Load W with D'100'
	MOVWF	PULSE	;Move to PULSE
			; Cannot use interrupts on portA so use inline function
			;call(not ideal, better to use portB)
	BTFSF	portA, SPD	;Speed switch pressed
	CALL	SPEED_SWITCH	
	BTFSF	portA, ENA	;Motor enabled
	CALL	ADJUST_PWM	
	CALL	ON_PULSE	
	CALL	OFF_PULSE	
	GOTO	LOOP	

;FUNCTIONS-----

;Timer0 delay routine 0.01mS

```

delay1      MOVLW      0x05          ; Count to 5
            MOVWF      COUNT
            MOVLW      0xFF          ; Set timer to 255
            MOVWF      TMR0
            BCF        INTCON, 2     ; Clear over flow flag bit 2
            BTFSS      INTCON, 2     ; Wait for flag set
            GOTO        $-1
            DECFSZ     COUNT
            GOTO        $-6
            RETURN

; Call delay routine 2000 times 20mS

delay2000    MOVLW      0x64          ; 100 into count1
            MOVWF      COUNT1
            MOVLW      0x14          ; 20 into count2
            MOVWF      COUNT2
            CALL        delay1
            DECFSZ     COUNT1, 1
            GOTO        $-2
            DECFSZ     COUNT2, 1
            GOTO        $-4
            RETURN

ADJUST_PWM  MOVLW      SPEED          ; Load W with speed adjustment value
            BTFSS      portA, DIR     ; Direction switch closed, Anticlockwise
            SUBWF      PULSE, 1       ; Subtract speed adjustment from default value,
                                       ; store in PULSE
            BTFSC      portA, DIR     ; Direction switch open, Clockwise
            ADDWF      PULSE, 1       ; Add speed adjustment to default value, store in PULSE
            RETURN

SPEED_SWITCH CALL        delay2000    ; Debounce
            BTFSS      portA, SPD     ; Speed switch pressed
            CALL        SPEED_INC     ; Increment speed valve
            RETURN

SPEED_INC   MOVLW      0x31          ; Set W to 50

```

	SUBLW	SPEED	; If 50 - SPEED is positive
	SKPC		;Skip on carry
	INCF	SPEED	;Update display
	MOVLW	SPEED	; Load SPEED value to W
	MOVWF	DISPLAY	; Load into DISPLAY reg
	MOVLW	DISPLAY	; PWM will always be low at this point in circuit, bit
			; shifting DISPLAY will always leave bit 0 low, so
			;simply transfer DISPLAY into portB
	MOVWF	portB	; Update outputs
	RETURN		
ON_PULSE	BSF	portB, PWM	; Switch motor output high
loop3	CALL	delay1	; Delay PULSE times
	DECFSZ	PULSE, 1	
	GOTO	loop3	
	RETURN		
OFF_PULSE	BCF	portB, PWM	; Switch motor output low
	CALL	delay2000	; Delay 20mS
	RETURN		
	END		;End of source code.

To explain the program:

All inputs are the switches A, B and C

The variable PULSE has the default value of 100. This is multiplied by the delay time of 0.01mS to give the default ON pulse and the speed is controlled by the number of times the SPD button is pressed. Depending whether the direction is clockwise or anticlockwise it will increment from 1.0mS to 1.5mS or decrement from 1.0mS to 0.5mS.

Each time the low output will remain at 20mS.

The ORG is where the program starts and the SETUP is where the A and B ports and Global Interrupts are defined.

MOVLW	0x07	;Config Port A, RA0 -> RA2 are inputs.
MOVWF	TRISA	;Set I/O configuration for PORTA

Here Port A is configured in the W register and RA0, RA1 and RA2 are set to inputs (In Hexidecimal it is shown as 0x07 and in binary b00000111). It is then moved to the W register and set as TRISA. This needs to be done as upon power on reset, they are all configured as inputs and read as '0'.

The timer has been set with the prescaler as 1:2 in the OPTION register. This is the lowest it can be set to. (b11000000). The last three bits denote 1:2 in the TMRO rate.

```

                                MOVLW      0xC0      ; Binary 11000000 C7- prescale 1:2

delay1  MOVLW      0x05      ; Count to 5
        MOVWF      COUNT
        MOVLW      0xFF      ; Set timer to 255
        MOVWF      TMRO
        BCF        INTCON, 2    ; Clear over flow flag bit 2
        BTFSS      INTCON, 2    ; Wait for flag set
        GOTO       $-1
        DECFSZ     COUNT
        GOTO       $-6
        RETURN

```

The loop is the main program which calls all the subroutines as and when they are required. A delay subroutine of 0.01mS is established which is the smallest possible.

This delay is used for all timings in the script and is called a multitude of times depending on the length of delay required.

For example, Delay1 counts two clock ticks to give 0.01mS and the 20mS delay calls it 20x100=2000 times.

The default for the Protractor disk 1mS high and then 20mS low, looped until the Speed button is pressed. To do this the ADJUST_PWM is called that will adjust the pulse time dependant on the direction switch and value in the SPEED register.

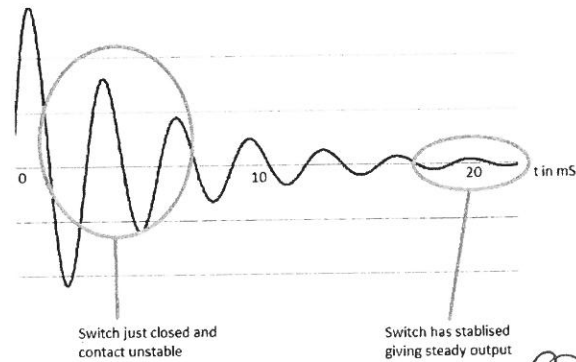
If anticlockwise, the value in SPEED is deleted from the default pulse value, if clockwise it is added.

```

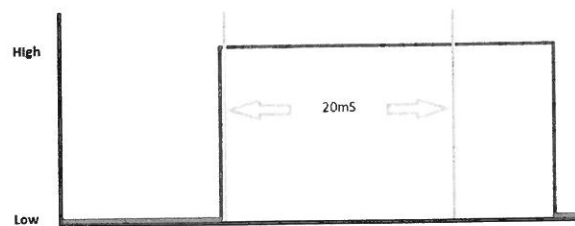
ADJUST_PWM  MOVLW      SPEED      ; Load W with speed adjustment value
            BTFSS      portA, DIR    ; Direction switch closed, Anticlockwise
            SUBWF      PULSE, 1      ; Subtract speed adjustment from default value,
                                     ;store in PULSE
            BTFSC      portA, DIR    ; Direction switch open, Clockwise
            ADDWF      PULSE, 1      ; Add speed adjustment to default value, store in PULSE
            RETURN

```

A debounce has been added to the momentary speed switch to ensure it doesn't count multiple times by accident. This can happen as the switch jitters back and forth until it settles and a steady high is obtained. The diagram below explains what happens when a switch closes.



The debounce uses the 0.01mS delay previously described and calls it 2000 times. Once 20mS have passed and switch jitter has settled, the program continues.



The working register is set to 49 and deleted from the value in SPEED.. As 50 is the maximum number of times the speed button can be pressed a value above 50 causes the result to be negative. The following command SKPC, causes the following (increment) statement to be skipped, keeping SPEED at a maximum of 50.

The current speed setting

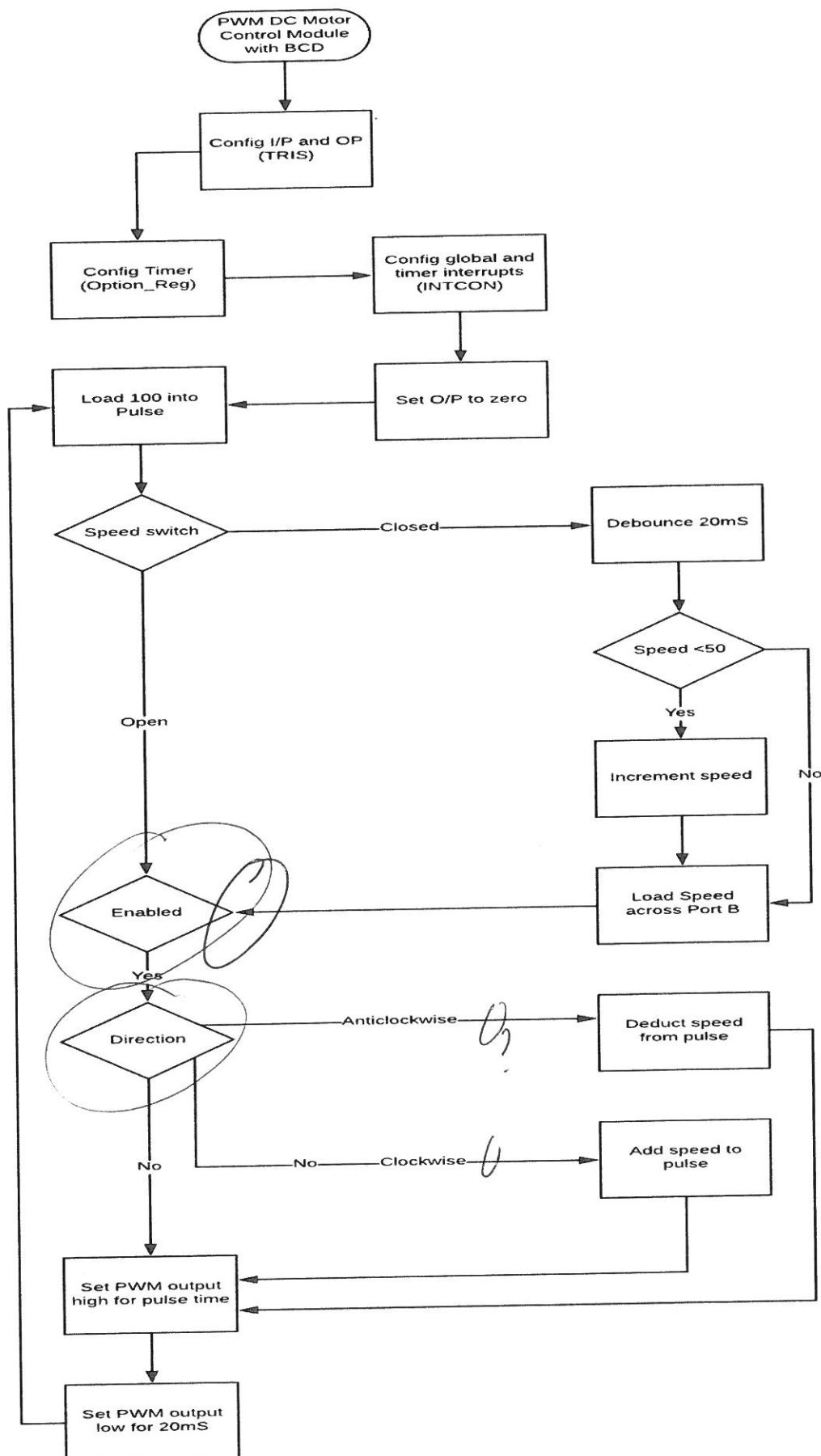
The speed is to be shown on two BCD displays from 0 to 50. Port B pins 0 to 6 control the display. (Three for each display).

The speed is transferred into the W register and put into Port B. It is only calling this function when the pin is set to low so the display can connect straight to Port B. It only becomes an issue when using an interrupt as you wouldn't know if the pulse was high. In this instance you would need to use a mask in conjunction with AND and OR logic.

SPEED_INC	MOVLW	0x31	; Set W to 50
	SUBLW	SPEED	; If 50 - SPEED is positive
	SKPC		;Skip on carry
	INCF	SPEED	;Update display
	MOVLW	SPEED	; Load SPEED value to W
	MOVWF	DISPLAY	; Load into DISPLAY reg
	MOVLW	DISPLAY	; PWM will always be low at this point in circuit, bit
			; shifting DISPLAY will always leave bit 0 low, so
			; simply transfer DISPLAY into portB

```
MOVWF    portB    ; Update outputs  
RETURN
```

The algorithmic chart below shows how the BCD counter operates



Switch B controlling direction

```
ADJUST_PWM    MOVLW        SPEED        ; Load W with speed adjustment value
               BTFSS        portA, DIR    ; Direction switch closed, Anticlockwise
               SUBWF        PULSE, 1      ; Subtract speed adjustment from default value,
                                           ; store in PULSE
               BTFSC        portA, DIR    ; Direction switch open, Clockwise
               ADDWF        PULSE, 1      ; Add speed adjustment to default value, store in PULSE
               RETURN
```

Here the W register is loaded with the new adjustment value from the default 1.0mS .

If the direction switch is closed (anticlockwise) then it will subtract the value.

1.0 to 0.99 to 0.98mS all the way to 0.5mS (maximum speed)

If the switch is open then the opposite will happen upto 1.5mS (maximum speed)

The BTFSS and BTFSC are used – Bit Test Skip if Clear/Set

In between each of these new adjustment values the 20mS off pulse is called for.

Overriding enable control

When the enable is not pressed it will default to 1mS on pulse, 20mS off pulse thereby stopping the motor.

```
BTFSS          portA, ENA      ;Motor enabled
               CALL          ADJUST_PWM
               CALL          ON_PULSE
               CALL          OFF_PULSE
               GOTO          LOOP
```

If not enabled, the pulse adjustment will not be performed, defaulting to the motors stopped state.

Possible improvements

As the three inputs are on Port A an interrupt can not be used.

This means that the part of the assembly code watch for a high input on RA0 (speed) is having to be called in every loop. Using an interrupt for this on Port B would mean that a direct interrupt command could go straight to

```
BTFSS          portA, SPD      ;Speed switch pressed
```

In the loop main program.

References

Calculating the TMRO

<http://www.microcontrollerboard.com/pic-timer0-tutorial.html>

In depth reading of the PIC 16f84a microprocessor

<http://www.mouser.com/ds/2/268/35007b-75235.pdf>- specification of 16f84a