

Introduction

An alarm clock can be a vital part of some people's working lives guaranteeing that they do not sleep in, alternatively they can be used to ensure the user is not late for whatever they are doing that day be that getting a train or meeting a deadline. Alarm clocks can help in these aspects by acting as notifications that it is a certain time telling the user what they had to do, i.e. The alarm goes off at 07:00 telling the user that it is time to get out of bed or the alarm goes off ten minutes before they have to leave for the train, reminding them that they have ten minutes to finish getting ready and leave. Such uses can allow a person to be more productive.

Specification

- ◆ Runs off a $5 \pm 1\text{v}$ power supply
- ◆ Uses a PIC 16f88
- ◆ Outputs to four red seven segment LED displays
- ◆ Displays 24 hour time
- ◆ Takes input via three tactile switches
- ◆ All buttons are debounce
- ◆ Displays have no visible flickering
- ◆ Outputs sound through a piezo transducer
- ◆ The time is incremented every minute $\pm .5\text{s}$

I will put the circuit diagram here

Meeting the specification

The clock runs off a +5-0Volt power supply counts in one second increments on four multiplexed seven-segment displays that display one of 20 digits, the first ten are human readable numbers 0-9 and the last ten are the same digits only with the decimal point of the displays active, the decimal point is used to show which display is currently selected when the circuit is taking an input from the user.

The first digit is incremented every minute, once this reaches ten it resets to zero and the second digit is incremented, once the second digit is incremented to six it resets to zero and increments to 0. The Third digit will reset and increment the fourth once it reaches ten if the fourth digit is zero or one however if the Fourth digit is two the third will rollover at five, this prevents the clock from displaying illegal times such as 27:00 while still allowing it to display legal times where the Third digit is over four such as 17:00.

Theoretically the active display switches every 576ms as set by a precale of 1:32 and an operating clock speed of 8MHz, this results in a visually smooth display that doesn't flicker, however these numbers are not exact due to the inaccuracy of the internal RC circuit used for timing. The clock operates on 24 hour time meaning it counts to 2400 before resetting to 0000.

The three buttons connected to bits 4-6 of port A are debounced through the software running on the PIC instead of through a physical circuit, this is done by making the

program wait for 200ms before executing the code attached to that input. The buttons are debounced this way because it is easier to build and design to produce the same effect, as a secondary benefit it also reduces the material cost of the circuit as there is no need for a dedicated RC circuit or any other hardware solutions.

Research

Most alarm clocks use a crystal oscillator instead of an RC circuit

- ◆ Operated from a 5 ±1
- ◆ Increments every minute ± .5s
- ◆ Operated with three tactile buttons
- ◆ All buttons debounced
- ◆ Digits are displayed on four RED seven segment LED displays
- ◆ Digits are displayed without visible flicker
- ◆ Uses Piezo Transducer as a speaker
- ◆ The program is run on a PIC 16f88

User Guide

After powering the system the user will be prompted to input the time they would like the alarm to go off, they can do this by using the leftmost button (Button6) select a digit, the currently selected digit is indicated by the decimal point. The user can then increment the selected digits as necessary using the middle button (Button5), after the user has set their desired alarm they can confirm their input by pressing the rightmost button (Button4), this will move them onto the next portion of the program setting the current time. The user will set the current time by using the same controls mentioned above, after the current time is set the user can start the clock and the alarm will sound at the set time.

Limitations

The displays cannot be photographed with a standard shutter speed and most video will show the displays cycling meaning that it is hard to read the output from a video.

The user also cannot set multiple alarms at the same time, a common feature that potentially allows for a better user experience, this could be added to the current system without a major re-design however it would result in a far larger program size.

While not a major problem there is a small inaccuracy caused by inherent inaccuracy present in the internal RC circuit that is used for timing. This can be fixed by using an external crystal oscillator to clock the timing. To do this I would change bit7 of PORTB to an input after the user has set their alarm and the time, I would do this because there are no spare pins in my design and bit7 serves no purpose after the user has finished their inputs as the decimal points are not used when the clock is keeping time.

The alarm cannot be silenced without turning off the clock, resetting the time and the alarm this could become annoying for the user as they would have to set the time

and the alarm every time the alarm goes off.

The alarm is arguably too loud and the volume cannot be adjusted.

There is no snooze feature, silencing the alarm and re-sounding it after some time. This reduces the utility of the system.

Testing Logs

Starting 14/05/24

Display

1. 09:20 display subroutine written but does not give an output, presumed cause to be the digits incrementing past nine too quickly.
2. 10:32 display still has no output, display routine found to be at fault
3. 10:42 display Subroutine re-written, still no output

17/05/24

1. 11:49 display Subroutine re-written, still no output.
2. 11:56 moved display code to main program for testing, no output but the external interrupt light now shows on displays 1 and 4 instead of just display 4, not sure why this happens.
3. 12:00 disabled external interrupt, light now only shows on display 4.
4. 13:21 after re-writing the display subroutine it gives no output but the external interrupt light now moves between displays.
5. 13:29 - After clearing the digits before they are set nothing has changed

21/05/24

1. 09:31 - replaced the look-up table with test code to turn on all LEDs, upon running only displays 1 and 3 were lit.
2. 09:36 - set Current display to 4 at the start of the program, once run displays 3 and 4 displayed an 8 (equivalent to b'01s111111') despite being set to b'11111111'
3. 09:51 - after reverting changes from a test only display four has an output.
4. 09:58 - all displays now work, however the wrong numbers are displayed, 1011 instead of 4321.
5. 10:04 - all displays now give an output.
6. 10:40 - added code to increment numbers before they are displayed, this broke everything
7. 11:06 - changed code to display four fives and then increment them, the fives display but do not increment
8. 11:11 - added argument to "incf", this has changed nothing
9. 11:17 - changed program to manually change the digits, this has changed nothing, the PIC loves the number 5.

24/05/24

10. 12:16 - changed display subroutine, now all displays light up and display a digit properly but the digit does not change between
11. 12:29 - changed display subroutine, display now works

Clock

04/06/24

1. 10:15 - Digit 1 now increments correctly
2. 10:30 - Digit 1 increments but the rest do not
3. 10:40 - All digits now increment, but this can only display 20 hours of 24
4. 11:21 - code to determine when digit 3 should increment does not work :(, should increment digit3 to 10 until digit4 is 2 and then clear both when digit3 is 4.

11/06/24

1. 09:47 - Digit4 now resets at 2, all other digits working correctly
2. 09:52 - All digits function correctly
- 3.

Input

11/06/24

1. 11:20 Select code is unresponsive, just displays all zeroes

14/06/24

1. 11:42 - Program starts by displaying all zeroes (this is intended), but no way of showing the selected digit. When button 6 is pressed a seemingly random display shows a seemingly random number and all other displays turn off. The start button does work however.
2. 12:03 - Pressing button6 now always displays zero on display4
3. 12:20 - Button6 now increments the digits but is polled far too quickly, button5 does not do anything.
4. 12:26 - After re-writing the selected display logic there has been no change.
5. 13:19 - After changing the select display subroutine to decrement the selected display Button5 still does not work.
6. 13:30 - Button5 now changes the display just too quickly, Digits1 and 3 can also be incremented to ten, represented by a zero with a decimal point
7. 13:51 - after debouncing the buttons they no-longer work.

18/06/24

1. 09:21 - buttons five and six are now debounced and increment digits at an acceptable rate
2. 09:25 - In input mode only display4 is on and clock mode flickers with display4 being significantly brighter than the rest.
3. 09:57 - incrementing digits no-longer works and the decimal point is not removed when the display is no longer selected, the clock mode works again.
4. 10:06 - decimal point now deactivates until digit1 when the displays break.
5. 10:42 - decimal point now functions as intended however incrementing a digit leaves it on zero and deactivates all other digits.
6. 11:04 - if you change the selected digit after incrementing the display breaks.
7. 11:10 - if digit2 is selected clock mode is activated prematurely
8. 11:17 - you can now change the selected digits as intended but if the display is selected after the digit is incremented the display breaks

21/06/24

1. 12:00 - can now change digit after incrementing but if the timer is started after a digit is incremented the display breaks.
2. 12:07 - The timer will now start with incremented digits but all displays will quickly turn off, when changing the selected digit in input mode if you loop the selected digit back to the start display4 will flicker then dim instead of enabling the decimal point, if the display is incremented again displays 4 and 3 will have the decimal point while display4 is still noticeably dimmer than the rest, if the selected digit is looped again all displays will turn off and the program will become unresponsive.

3. 13:34 - can now loop the selected digit as intended but if the timer is started digit4 will be changed to a dimmer copy of digit3 and the display will break when it rolls over, if digit2 is incremented the display will break when it tries to roll-over and if digit1 is incremented all displays will turn off when it tries to roll over.
4. 13:43 - The Time set function seems to be working as intended
- 5.

Change Logs

- ◆ 17/05/24
 - ◆ 10:01 - Re-Written selection logic of the display sub-routine so that it now uses the zero bit instead of the decfsz command.
 - ◆ 10:34 - Added notation to display subroutine.
 - ◆ 11:43 - commas are illegal characters.
 - ◆ 13:18 - Re written display subroutine so that it no longer copies CurrentDisplay to another file register.
 - ◆ 13:48 - set prescaler to 1:32
- ◆ 21/05/24
 - ◆ 09:13 - changed "movf PORTB" to "movwf PORTB" in the display subroutine
 - ◆ 09:40 - changed prescaler to 1:64
 - ◆ 09:51 - reverted changes from a test that meant the timer interrupt was not calling display
 - ◆ 09:56 - removed the code to set CurrentDisplay to 4 from the main loop, now happens before the main loop so that all displays are used
- ◆ 24/05/24
 - ◆ 09:21 - added return commands to subdisplay4 and the main display subroutine.
- ◆ 03/06/24
 - ◆ 13:33 - added code to increment digit 1
- ◆ 04/06/24
 - ◆ 10:17 - added code to increment digits 2 - 4
 - ◆ 10:39 - fixed code to increment all digits
 - ◆ 10:43 - added subroutine to wait a minute (Time is sped up to make testing easier)
- ◆ 11/06/24
 - ◆ 11:19 - added code to select time
- ◆ 14/06/24
 - ◆ 11:37 - changed code to input time so that it polls PORTA instead of PORTB
 - ◆ 11:57 - called display in the input-time subroutine in an attempt to keep the displays cycling after a digit is incremented
 - ◆ 12:13 - added code to clear the digits if they are above they need to be reset, i.e. they are above the point at which they increment.
 - ◆ 12:25 - re-written code to change the selected display
 - ◆ 13:16 - code to change the selected display to decrement the current display instead of increment.
 - ◆ 13:51 - debounced switches five and six by adding wait100ms subroutines after they
- ◆ 18/06/24
 - ◆ 09:16 - debounced switches five and six by adding wait100ms subroutines before functions of the buttons are executed

- ◆ 09:24 - added code to show which display is selected
- ◆ 11:04 - added code to remove decimal point when entering clock mode
- ◆ 21/06/24
 - ◆ 11:55 - changed sublw to subwf to stop negative numbers being fed into the digit table and breaking the display
 - ◆ 14:06 - fixed time set subroutine and added alarm set subroutine,
- ◆ 28/06/24
 - ◆ 09:44 - Fixed spelling mistakes in "GetAlarm" that were causing errors, fixed spelling mistakes in "Compare" that were causing errors, compare now calls buzz when all digits are the same as the alarm digits.
- ◆ 08/07/24
 - ◆ 12:16 added a subroutine to display alarm digits, now will not enter clock mode

Source Code

I will not have the code typed out like this on the final report because it has to be printed but I was unsure of how to appropriately screenshot it or otherwise reduce the number of lines. Can you please clarify how the code should be attached.

```

;***
; TEMPLATE PROVIDED BY CENTRE

; TITLE: ##### Alarm Clock #####
; AUTHOR: ##### Hamish Lindsay #####
; DATE: ##### 07/05/2024 #####
;

; **
; PROGRAM DESCRIPTION:
;
; ##### The Program takes the current time from the user and lets the user set an
alarm, when the alarm is reached the piezo buzzer will sound at 500hz.#####
;
; *
; DEFINITIONS
; *
list p=16F88 ; tells the assembler which PIC chip to program for
radix dec ; set default number radix to decimal
;radix hex uncomment this to set radix to hex
config h'2007', 0x3F50 ; internal oscillator, RA5 as i/o, wdt off
config h'2008', 0x3FFF
errorlevel -302 ; hide page warnings

W EQU h'00' ; pointer to Working register
F EQU h'01' ; pointer to file

; ** REGISTER USAGE **

; For PIC16F88, user RAM starts at h'20'. The following definitions
; will be found useful in many programs.

```

```

; Register page 1
PCL EQU h'02'
TRISA EQU h'85' ; data direction registers
TRISB EQU h'86'
OSCCON EQU h'8F' ; internal oscillator speed
ANSEL EQU h'9B' ; ADC port enable bits

; Register page 0
STATUS EQU h'03' ; status
PORTA EQU h'05' ; input / output ports
PORTB EQU h'06'
INTCON EQU h'0B' ; interrupt control
ADRESH EQU h'1E' ; ADC result
ADCON0 EQU h'1F' ; ADC control
OPTION_REG EQU h'81'

B0 EQU h'20' ; general use byte registers B0 to B27
B1 EQU h'21'
B2 EQU h'22'
B3 EQU h'23'
B4 EQU h'24'
B5 EQU h'25'
B6 EQU h'26'
B7 EQU h'27'
B8 EQU h'28'
B9 EQU h'29'
B10 EQU h'2A'
B11 EQU h'2B'
B12 EQU h'2C'
B13 EQU h'2D'
B14 EQU h'2E'
B15 EQU h'2F'
B16 EQU h'30'
B17 EQU h'31'
B18 EQU h'32'
B19 EQU h'33'
B20 EQU h'34' ; used in interrupt routine
B21 EQU h'35' ; used in interrupt routine
B22 EQU h'36'
B23 EQU h'37'
B24 EQU h'38'
B25 EQU h'39'
B26 EQU h'3A'
B27 EQU h'3B'

WAIT1 EQU h'3C' ; counters used in wait delays
WAIT10 EQU h'3D'
WAIT100 EQU h'3E'
WAIT1000 EQU h'3F'
ADCTEMP EQU h'40' ; adc loop counter
Digit1 EQU h'41' ;minutes (will be sped up for testing)
Digit2 EQU h'42' ;ten minutes
Digit3 EQU h'43' ;hours
Digit4 EQU h'44' ;ten hours

```

```

save1 EQU h'45' ;register to copy digit1 to
save2 EQU h'46' ;register to copy digit2 to
save3 EQU h'47' ;register to copy digit3 to
save4 EQU h'48' ;register to copy digit4 to
CurrentDisplay EQU h'49' ;variable to indicate the current display
SaveDis EQU h'4A'
AlarmDig1 EQU h'4C' ;minutes used to compare the alarm to
AlarmDig2 EQU h'4D' ;ten minutes used to compare the alarm to
AlarmDig3 EQU h'4E' ;hours used to compare the alarm to
AlarmDig4 EQU h'4F'
count EQU h'55'
SelectDis EQU h'56' ;Display selected for input
buzzCount EQU h'57' ;Used for buzz loop to time how long the piezo will sound
;** REGISTER BITS **

```

```

C EQU h'00' ; carry flag
Z EQU h'02' ; zero flag
RP0 EQU h'05' ; register page bit
INT0IF EQU h'01' ; interrupt 0 flag
INT0IE EQU h'04' ; interrupt 0 enable
GIE EQU h'07' ; global interrupt enable
TMR0IE EQU h'05' ; Timer interrupt enable
TMR0IF EQU h'02' ; Timer interrupt flag

```

```

;*
; VECTORS
;*

```

```

;The PIC16F88 reset vectors

```

```

ORG    h'00'           ; reset vector address
goto   start           ; goes to first instruction on reset/power-up
ORG    h'04'           ; interrupt vector address
goto   interrupt

```

```

;
;*
; SUBROUTINES
;*
; Predefined wait subroutines - wait1ms, wait10ms, wait100ms, wait1000ms

```

```

wait1ms ; (199 x 5) + 5 instructions = 1000us = 1ms @ 4MHz resonator
movlw d'199' ; 1
movwf WAIT1 ; 1
loop5ns
clrwdt ; 1 this loop 1+1+1+2 = 5 instructions
nop ; 1
decfsz WAIT1,F ; 1
goto loop5ns ; 2
nop ; 1
return ; 2
wait10ms
movlw d'10' ; 10 x 1ms = 10ms

```



```

movwf WAIT10
loop10ms
call wait1ms
decfsz WAIT10,F
goto loop10ms
return

wait100ms
movlw d'100' ; 100 x 1ms = 100ms
movwf WAIT100
loop100ms
call wait1ms
decfsz WAIT100,F
goto loop100ms
return
wait1min
clrf count
minloop ; waits for one minute used to time the increments
call wait1000ms
incf count
movf count, 0
sublw d'60'
btfss STATUS, Z
goto minloop
return
wait1000ms
movlw d'10' ; 10 x 100ms = 1000ms
movwf WAIT1000
loop1000ms
call wait100ms
decfsz WAIT1000,F
goto loop1000ms
return
wait500ms
call wait100ms
call wait100ms
call wait100ms
call wait100ms
call wait100ms
return
wait200ms
call wait100ms
call wait100ms
return

; Predefined ADC subroutines - readadc0, readadc1, readadc2

readadc0
movlw b'00000001' ; setup mask for pin A.0
call readadc ; do the adc conversion
movwf B0 ; save result in B0
return
readadc1
movlw b'00000010' ; setup mask for pin A.1
call readadc ; do the adc conversion

```

```

movwf B1 ; save result in B1
return
readadc2
movlw b'00000100' ; setup mask for pin A.2
call readadc ; do the adc conversion
movwf B2 ; save result in B2
return

readadc
; Generic sub routine to read ADC 0, 1 or 2 (pass appropriate mask in W)
; To start conversion we need mask (001, 010, 100) in ANSEL bits 0-2
; but the actual channel number (0, 1, 2) in ADCON0 channel select bits
; Then set the ADCON0, GO bit to start the conversion

```

```

bsf          STATUS,RP0      ; select register page 1
movwf  ANSEL          ; move mask value 001,010,100 into ANSEL
bcf          STATUS,RP0      ; select register page 0
movwf  ADCTEMP ; 00000???    get mask value
rlf          ADCTEMP,F        ; 0000???x      rotate twice
rlf          ADCTEMP,W        ; 000???xx
andlw  b'00011000'      ; 000??000    mask off the unwanted bits
iorlw  b'00000001'      ; 000??001    set the 'ADC on' bit
movwf  ADCON0 ; move working into ADCON0
          movlw          d'10'          ; 10 x 3 = 30us acquisition time
          movwf  ADCTEMP          ; re-use ADC1 register as a counter

```

```

loopacq
decfsz ADCTEMP,F ; loop around to create short delay
goto loopacq ; each loop is 1+2 = 3 instructions = 3us @ 4MHz
bsf ADCON0,2 ; now start the conversion
loopadc
clrwdt ; pat the watchdog
btfsc ADCON0,2 ; is conversion finished?
goto loopadc ; no, so wait a bit more
movf ADRESH,W ; move result into W
return ; return with result in W

```

; NOTE for PICAXE users: the following four specific subroutines and two instructions are not supported by PICAXE compiler

```

readtemp1:
readtemp2:
readtemp3:
debug:
lcd:
clrw ; instruction not supported by this template
return ; instruction not supported by this template

```

```

table ;lookup table for seven segment displays.
addwf PCL
retlw b'00111111'
retlw b'00000110'
retlw b'01011011'
retlw b'01001111'

```

```

retlw b'01100110'
retlw b'01101101'
retlw b'01111101'
retlw b'00000111'
retlw b'01111111'
retlw b'01101111'
retlw b'10111111' ;table repeated with the decimal point lit.
retlw b'10000110'
retlw b'11011011'
retlw b'11001111'
retlw b'11100110'
retlw b'11101101'
retlw b'11111101'
retlw b'10000111'
retlw b'11111111'
retlw b'11101111'

```

Display ;select the display to turn on

```

movf CurrentDisplay, 0
sublw d'4'
btfsc STATUS, Z
goto SubDisplay1 ;Subtract four from CurrentDisplay, if result is 0 goto SubDisplay1
movf CurrentDisplay, 0
sublw d'3'
btfsc STATUS, Z
goto SubDisplay2
movf CurrentDisplay, 0
sublw d'2'
btfsc STATUS, Z
goto SubDisplay3
movf CurrentDisplay, 0
sublw d'1'
btfsc STATUS, Z
goto SubDisplay4

```

```

SubDisplay1
bcf PORTA, 3
bsf PORTA, 0
movf Digit1, 0
call table
movwf PORTB ;Display value of Digit 1 on display 1
decf CurrentDisplay, 1
return ;Return to display selection
SubDisplay2 ;Repeat for other displays
bcf PORTA, 0
bsf PORTA, 1
movf Digit2, 0
call table
movwf PORTB
decf CurrentDisplay, 1
return
SubDisplay3
bcf PORTA, 1
bsf PORTA, 2

```

```

movf Digit3, 0
call table
movwf PORTB
decf CurrentDisplay, 1
return
SubDisplay4
bcf PORTA, 2
bsf PORTA, 3
movf Digit4, 0
call table
movwf PORTB
movlw d'4'
movwf CurrentDisplay
return

Compare
movf Digit1, 0
subwf AlarmDig1, 0
btfsc STATUS, Z ;Compare Digit1 of current time and digit =1 of the alarm, if they
are the same check digit2, if they are not return to program
goto Compare2
return
Compare2
movf Digit2, 0
subwf AlarmDig2, 0
btfsc STATUS, Z ;Compare digit2 fo current time to digit2 of the alarm, if they are
the same check digit3, if they are not return to program
goto Compare3
return
Compare3
movf Digit3, 0
subwf AlarmDig3, 0
btfsc STATUS, Z ;Compare digit3 o fcurrent time to digit3 of alarm, if they are the
same check digit4, if they are not return to program
goto Compare4
return
Compare4
movf Digit4, 0
subwf AlarmDig4, 0
btfsc STATUS, Z ;Compare digit4 of current time to digit4 of the alarm, if they are
the same sound the peizo, if they are not return to program
goto test
return
Buzz
buzzLoop
bsf PORTA, 7
call wait1ms
call wait1ms
bcf PORTA, 7
call wait500ms
goto buzzLoop
GetAlarm
clrf Digit1 ;Set all digits to 0 and set Selectdis to 4
clrf Digit2

```

```

clrf Digit3
clrf Digit4
movlw d'4'
movwf SelectDis
Alarmloop
movf SelectDis, 0 ;if display4 is selected enter the loop to poll changes to digit4
sublw d'4'
btfsc STATUS, Z
goto AlarmSelectIs4
movf SelectDis, 0 ;if display3 is selected enter the loop to poll changes to digit3
sublw d'3'
btfsc STATUS, Z
goto AlarmSelectIs3
movf SelectDis, 0 ;if display2 is selected enter the loop to poll changes to digit2
sublw d'2'
btfsc STATUS, Z
goto AlarmSelectIs2
movf SelectDis, 0 ;if display1 is selected enter the loop to poll changes to digit1
sublw d'1'
btfsc STATUS, Z
goto AlarmSelectIs1
AlarmSelectIs4 ;add the decimal point to the selected digit
movf Digit4, 0
addlw d'10'
movwf Digit4
AlarmSelectLoop4 ;poll the input bits and go to their according subroutines if they
are high
btfsc PORTA ,6
goto Alarminc4
btfsc PORTA ,5
goto DecDis4
btfsc PORTA ,4
goto Alarmcontinue4
goto AlarmSelectLoop4
AlarmSelectIs3 ;add the decimal point to the selected digit
movf Digit3, 0
addlw d'10'
movwf Digit3
AlarmSelectLoop3 ;poll the input bits and go to their according subroutines if they
are high
btfsc PORTA ,6
goto Alarminc3
btfsc PORTA ,5
goto AlarmDecDis3
btfsc PORTA ,4
goto Alarmcontinue3
goto AlarmSelectLoop3
AlarmSelectIs2 ;add the decimal point to the selected digit
movf Digit2, 0
addlw d'10'
movwf Digit2
AlarmSelectLoop2 ;poll the input bits and go to their according subroutines if they
are high

```

```

btfsc PORTA ,6
goto Alarminc2
btfsc PORTA ,5
goto AlarmDecDis2
btfsc PORTA ,4
goto Alarmcontinue2
goto AlarmSelectLoop2
AlarmSelectIs1 ;add the decimal point to the selected digit
movf Digit1, 0
addlw d'10'
movwf Digit1
AlarmSelectLoop1 ;poll the input bits and go to their according subroutines if they
are high
btfsc PORTA ,6
goto Alarminc1
btfsc PORTA ,5
goto AlarmSetDis
btfsc PORTA ,4
goto Alarmcontinue1
goto AlarmSelectLoop1
Alarminc4 ;increment the digit4, if it is equal to 13 (three with a decimal point)
set it to 10 (zero with a decimal point).
call wait200ms
incf Digit4, 1
movf Digit4, 0
sublw d'13'
btfss STATUS, Z
goto AlarmSelectLoop4
movlw d'10'
movwf Digit4
goto AlarmSelectLoop4
Alarminc3 ;increment the digit3, if it is equal to 20 (this would display an
unintended digit) set it to 10 (zero with a decimal point).
call wait200ms
incf Digit3, 1
movf Digit3, 0
sublw d'20'
btfss STATUS, Z
goto AlarmSelectLoop3
movlw d'10'
movwf Digit3
goto AlarmSelectLoop3
Alarminc2 ;increment the digit2, if it is equal to 16 (six with a decimal point) set
it to 10 (zero with a decimal point).
call wait200ms
incf Digit2, 1
movf Digit2, 0
sublw d'16'
btfss STATUS, Z
goto AlarmSelectLoop2
movlw d'10'
movwf Digit2
goto AlarmSelectLoop2

```

```

Alarminc1 ;increment the digit3, if it is equal to 20 (this would display an
unintended digit) set it to 10 (zero with a decimal point).
call wait200ms
incf Digit1, 1
movf Digit1, 0
sublw d'20'
btfss STATUS, Z
goto AlarmSelectLoop1
movlw d'10'
movwf Digit1
goto AlarmSelectLoop1
AlarmDecDis4 ;remove the decimal point and move to the subroutine to change the
selected display
movlw d'10'
subwf Digit4, 1
goto AlarmDecDis
AlarmDecDis3 ;remove the decimal point and move to the subroutine to change the
selected display
movlw d'10'
subwf Digit3, 1
goto AlarmDecDis
AlarmDecDis2 ;remove the decimal point and move to the subroutine to change the
selected display
movlw d'10'
subwf Digit2, 1
goto AlarmDecDis
AlarmDecDis ;remove the decimal point and move to the subroutine to change the
selected display
call wait200ms
decf SelectDis, 1
goto Alarmloop
AlarmSetDis ;remove the decimal point and change the selected display to display
four.
movlw d'10'
subwf Digit1, 1
call wait200ms
movlw d'4'
movwf SelectDis
goto Alarmloop
Alarmcontinue1 ;remove the decimal point and move to the exit point of the
subroutine
movlw d'10'
subwf Digit1, 1
Goto AlarmReturn
Alarmcontinue2 ;remove the decimal point and move to the exit point of the
subroutine
movlw d'10'
subwf Digit2, 1
goto AlarmReturn
Alarmcontinue3 ;remove the decimal point and move to the exit point of the
subroutine
movlw d'10'
subwf Digit3, 1

```

```

goto AlarmReturn
AlarmContinue4 ;remove the decimal point and move to the exit point of the
subroutine
movlw d'10'
subwf Digit4, 1
goto AlarmReturn
AlarmReturn ;save the time on the displays to AlarmDig1-4 before exiting the
subroutine.
movf Digit1, 0
movwf AlarmDig1
movf Digit2, 0
movwf AlarmDig2
movf Digit3, 0
movwf AlarmDig3
movf Digit4, 0
movwf AlarmDig4
return
GetTime
clrf Digit1
clrf Digit2
clrf Digit3
clrf Digit4
movlw d'4'
movwf SelectDis
TimeLoop
movf SelectDis, 0
sublw d'4'
btfsc STATUS, Z
goto SelectIs4
movf SelectDis, 0
sublw d'3'
btfsc STATUS, Z
goto SelectIs3
movf SelectDis, 0
sublw d'2'
btfsc STATUS, Z
goto SelectIs2
movf SelectDis, 0
sublw d'1'
btfsc STATUS, Z
goto SelectIs1
SelectIs4
movf Digit4, 0
addlw d'10'
movwf Digit4
SelectLoop4
btfsc PORTA, 6
goto inc4
btfsc PORTA, 5
goto DecDis4
btfsc PORTA, 4
goto continue4
goto SelectLoop4

```



```

SelectIs3
movf Digit3, 0
addlw d'10'
movwf Digit3
SelectLoop3
btfsc PORTA ,6
goto inc3
btfsc PORTA ,5
goto DecDis3
btfsc PORTA ,4
goto continue3
goto SelectLoop3
SelectIs2
movf Digit2, 0
addlw d'10'
movwf Digit2
SelectLoop2
btfsc PORTA ,6
goto inc2
btfsc PORTA ,5
goto DecDis2
btfsc PORTA ,4
goto continue2
goto SelectLoop2
SelectIs1
movf Digit1, 0
addlw d'10'
movwf Digit1
SelectLoop1
btfsc PORTA ,6
goto inc1
btfsc PORTA ,5
goto SetDis
btfsc PORTA ,4
goto continue1
goto SelectLoop1
inc4
call wait200ms
incf Digit4, 1
movf Digit4, 0
sublw d'13'
btfss STATUS, Z
goto SelectLoop4
movlw d'10'
movwf Digit4
goto SelectLoop4
inc3
call wait200ms
incf Digit3, 1
movf Digit3, 0
sublw d'20'
btfss STATUS, Z
goto SelectLoop3

```

```

movlw d'10'
movwf Digit3
goto SelectLoop3
inc2
call wait200ms
incf Digit2, 1
movf Digit2, 0
sublw d'16'
btfss STATUS, Z
goto SelectLoop2
movlw d'10'
movwf Digit2
goto SelectLoop2
inc1
call wait200ms
incf Digit1, 1
movf Digit1, 0
sublw d'20'
btfss STATUS, Z
goto SelectLoop1
movlw d'10'
movwf Digit1
goto SelectLoop1
DecDis4
movlw d'10'
subwf Digit4, 1
goto DecDis
DecDis3
movlw d'10'
subwf Digit3, 1
goto DecDis
DecDis2
movlw d'10'
subwf Digit2, 1
goto DecDis
DecDis
call wait200ms
decf SelectDis, 1
goto TimeLoop
SetDis
movlw d'10'
subwf Digit1, 1
call wait200ms
movlw d'4'
movwf SelectDis
goto TimeLoop
continue1
movlw d'10'
subwf Digit1, 1
call wait200ms
return
continue2
movlw d'10'

```

```

subwf Digit2, 1
call wait200ms
return
continue3
movlw d'10'
subwf Digit3, 1
call wait200ms
return
continue4
movlw d'10'
subwf Digit4, 1
call wait200ms
return
test
bsf PORTA,7
goto test
ShowAlarm
movf AlarmDig1, 0
movwf Digit1
movf AlarmDig2, 0
movwf Digit2
movf AlarmDig3, 0
movwf Digit3
movf AlarmDig4, 0
movwf Digit4
call wait1000ms
return

;*
; MAIN PROGRAM
;*

;** INITIALISATION **
start
bsf STATUS,RP0 ; select register page 1
movlw b'01100000' ; set to 4MHz internal operation
movwf OSCCON
clrf ANSEL ; disable ADC (enabled at power-up)
bcf STATUS,RP0 ; select register page 0

;The data direction registers TRISA and TRISB live in the special register set. A
'1' in
;these registers sets the corresponding port line to an Input, and a
;'0' makes the corresponding line an output.

Init
clrf PORTA ; make sure PORTA output latches are low
clrf PORTB ; make sure PORTB output latches are low
bsf STATUS,RP0 ; select register page 1
;Modify the next line to correspond with your input output requirements
movlw b'01110000' ; set port A data direction (0 = output bit, 1 = input bit)
movwf TRISA ;
;Modify the next line to correspond with your input output requirements
movlw b'00000000' ; set port B data direction (0 = output bit, 1 = input bit)

```

```

movwf TRISB ;

movlw b'11000011' ;set timer prescaler to 1:32
movwf OPTION_REG

bcf STATUS,RP0 ; select register page 0
clrf AlarmDig1
clrf AlarmDig2
clrf AlarmDig3
clrf AlarmDig4
;** MAIN PROGRAM **
;* remove semicolons from next two lines to enable interrupt routine****
;bsf INTCON,INT0IE ; set external interrupt enable
bsf INTCON,GIE ; enable all interrupts
bsf INTCON, TMR0IE ;enable timer0 interrupt
main
movlw d'4'
movwf CurrentDisplay
call GetAlarm
call GetTime
Loop1
call Compare
call wait1min
incf Digit1, 1
movf Digit1, 0
sublw d'10'
btfss STATUS, Z
goto Loop1
clrf Digit1
Loop2
call Compare
incf Digit2, 1
movf Digit2, 0
sublw d'6'
btfss STATUS, Z
goto Loop1
clrf Digit2
Loop3
call Compare
movf Digit4, 0
sublw d'2'
btfsc STATUS, Z
goto Dig4is2 ;if digit4 is currently two, increment digit3 to four otherwise
increment digit3 to ten.
incf Digit3, 1
movf Digit3, 0
sublw d'10'
btfss STATUS, Z
goto Loop1
clrf Digit3
goto Loop4
Dig4is2
call Compare

```

```

incf Digit3, 1
movf Digit3, 0
sublw d'4'
btfss STATUS, Z
goto Loop1
clrf Digit3
Loop4
call Compare
incf Digit4, 1
movf Digit4, 0
sublw d'3'
btfss STATUS, Z
goto Loop1
clrf Digit4
goto Loop1
;*
;*
; INTERRUPT SERVICE ROUTINE
;*

W_SAVE EQU B20 ; backup registers used in interrupts
interrupt
movwf W_SAVE ; Copy W to save register

btfss INTCON,TMR0IF ; check correct interrupt has occurred
retfie ; no, so return and re-enable GIE
;**The interrupt service routine (if required) goes here*
call Display
bcf INTCON,TMR0IF ; clear interrupt flag
movf W_SAVE,W ; restore W
retfie ; return and re-set GIE bit
END ; all programs must end with this

```