AS Electronics PIC Project - Calculator

Design Brief:
This project's goal is the development of a calculator which supports the four basic arithmetic operations. It is capable of displaying inputs, outputs and any errors that may arise during calculation on a decimal display. It accepts input via a standard numeric keypad.

Specification:

-Accurately Add, Subtract, Multiply and Divide integers using 16bits of storage

-Subtraction should accurately display results with negative and positive answers

-Division should display the integer part of the result and notify if there was a remainder

-The previous calculation's result will automatically be used as one of the next operations left operand, after floor and absolute value functions are applied

-Pressing Equals after a calculation will repeat the operation using the previous right operand and the previous result as the left operand

-16bit overflows and math errors will be reported on the display using the F and E characters respectively

-Dual stage reset: First press clears the currently displayed value but maintains values stored in memory. Second press clears all values.

Bill Of Materials:
-3x PIC16F627A
-2x 3-digit Kingbright BC56-12EWA numeric Display
-16x Push Buttons
-16x 330 Ohm Resistors
-6x 10k Ohm Resistors
-3x 12k Ohm Resistors

Michal Adamkiewicz

Principal of Operation:

Overview:

PIC #1 takes input from a single clear button and a 3x5 array of buttons in the shape of a keypad, using multiplexing on the three columns. The layout is shown to the left.

After preforming the calculation PIC #1, the desired output is sent out from PIC #1 to PIC #2 and PIC #3 using a serial bus. PIC #2 handles one of the 3-digit displays and PIC #3 handles the other 3-digit display. To accommodate for the limited number of pins on each PIC the displays are driven using multiplexing.

7	8	9
4	5	6
1	2	3
+	0	=
-	/	*

Keypad Layout

Button	7	6	5	4	3	2	1	0	Sum
-	1	0	0	0	0	1	0	0	132
/	1	0	0	0	0	0	1	0	130
*	1	0	0	0	0	0	0	1	129
+	0	1	0	0	0	1	0	0	68
0	0	1	0	0	0	0	1	0	66
=	0	1	0	0	0	0	0	1	65
1	0	0	1	0	0	1	0	0	36
2	0	0	1	0	0	0	1	0	64
3	0	0	1	0	0	0	0	1	33
4	0	0	0	1	0	1	0	0	20
5	0	0	0	1	0	0	1	0	18
6	0	0	0	1	0	0	0	1	17
7	0	0	0	0	1	1	0	0	12
8	0	0	0	0	1	0	1	0	10
9	0	0	0	0	1	0	0	1	9

Input:

To accommodate for the limited number of pins on a PIC16F627A the buttons inputs are multiplexed in 3 columns of 5. Each column is connected to one of the 3 least significant bits of PORTB, and each of the rows to the 5 most significant bits. PIC #1 repeatedly flashes a single column high while keeping the other column low. It then looks for changes on the 5 most significant bits of PORTB to indicate a button was pressed. Additionally before each cycle PIC#1 pools the clear button (A7). This happens in the main loop in the code.

Which 2 PORTB pins where high when this happens determine which button was pressed. Decoding of this information takes place in the button_decode subroutine, which redirects the program flow to an appropriate “act” subroutine. Placed to the left is the table of PORTB values for each button. The top row indicates pin numbers of the PORTB register.

Output:

PIC #2 and PIC #3 each control one 3-digit display, and run identical code. A display module refers to the PIC 3-digit display combo. Each display module has 3 input pins Prog (A5), Clock(A4) and Data(A6) and 1 output pin (A7). The first 3 are connected to PIC #1 pins A0, A2, A2 respectively. This allows us to daisy chain many (in the case of this project 2) display modules by connecting all Prog pins together, all Clock pins together, and plugging one module's Output pin into the next’s Data input. They behave much like 24bit shift registers with the additional function of multiplexing on 8 parallel outputs to the display while Prog pin is low. When Prog is high, a low pulse on the Clock pin shifts all values over by one bit, taking a new bit from Data pin and shifting out the old one to the Output pin. As

PICs #2,3 run at a much faster frequency of 4 MHz as opposed to PIC#1's 48 kHz no delay code is needed on PIC#1

Subroutine Descriptions:

PIC #1

main - loop in which PIC waits for user input from button array via multiplexing or via clear button. This loop is executed while idle, and is the first non-initialization subroutine run

Button_decode - decodes values of PORTB and determines which button does what. It redirects program flow to "act" subroutines which correspond 1 to 1 to the buttons

[0-9]_act - loads the respective digit value into store register and goes to shift_digits.

[add/sub/mul/div]_act - Runs any previously buffered calculation (equivalent to pressing equals before any other operator), remembers which operator was pressed in the operator register (0=add 1=sub 2=mul 3=div) and ensures the left operand is correct.

equ_act - Prepares registers for calculation, by ensuring operands are in correct places based on if the press is a repeat of the previous operation. Then It calls equ_function for calculation and write_value for display.

Shift_digits - updates value_hi and lo registers based on button press. It multiplies the current value by 10 and adds the value of store to append a digit the the end

equ_function - redirects program flow to subroutines preforming arithmetic functions based on the value of the operator register.

Actual_add - Implements 16bit addition using standard methods and carry bits

actual_mul - Implements 16bit multiplication using a long multiplication in base 2

actual_sub - Implements 16bit subtraction using standard methods and borrow bits

actual_div - Implements 16bit integer division using repeated subtraction

Write_value - Uses serial data transfer to communicate with PICs #2,3 to display the required value on the display

Segment_look - A lookup table containing he shapes of digits 0-9 on a numeric display

Multi_key - Handles the rare occurrence of the PIC begin unable to discern which press was first if two buttons are pressed simultaneously

Unexpected_error - Handles math undefined errors and previously unpredicted errors

Overflow - Handles cases in which numbers are to big to be stores in 16bits

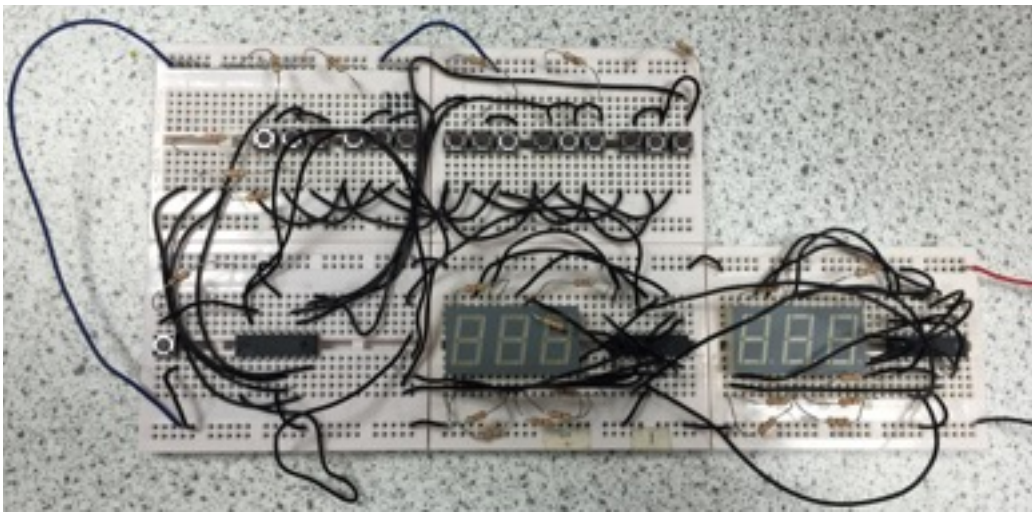
PIC# 2,3

main - Loop that multiplexes the 3 digits while no data is being sent

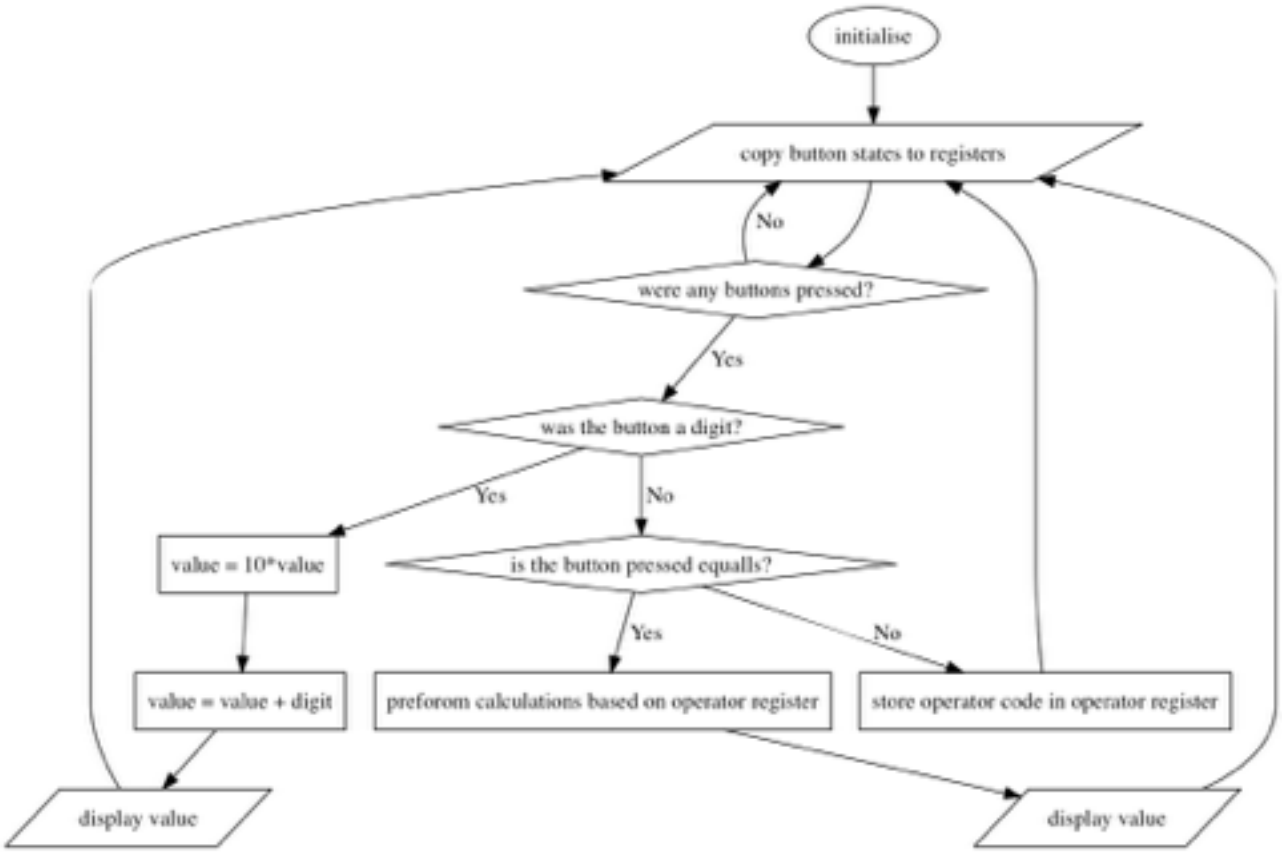
prog - Code that received serial data and stores in in dig_one dig_two dig_three registers

Delay - Time waisting loop which changes the multiplexing duty cycles allowing human eyes to correctly perceive the display

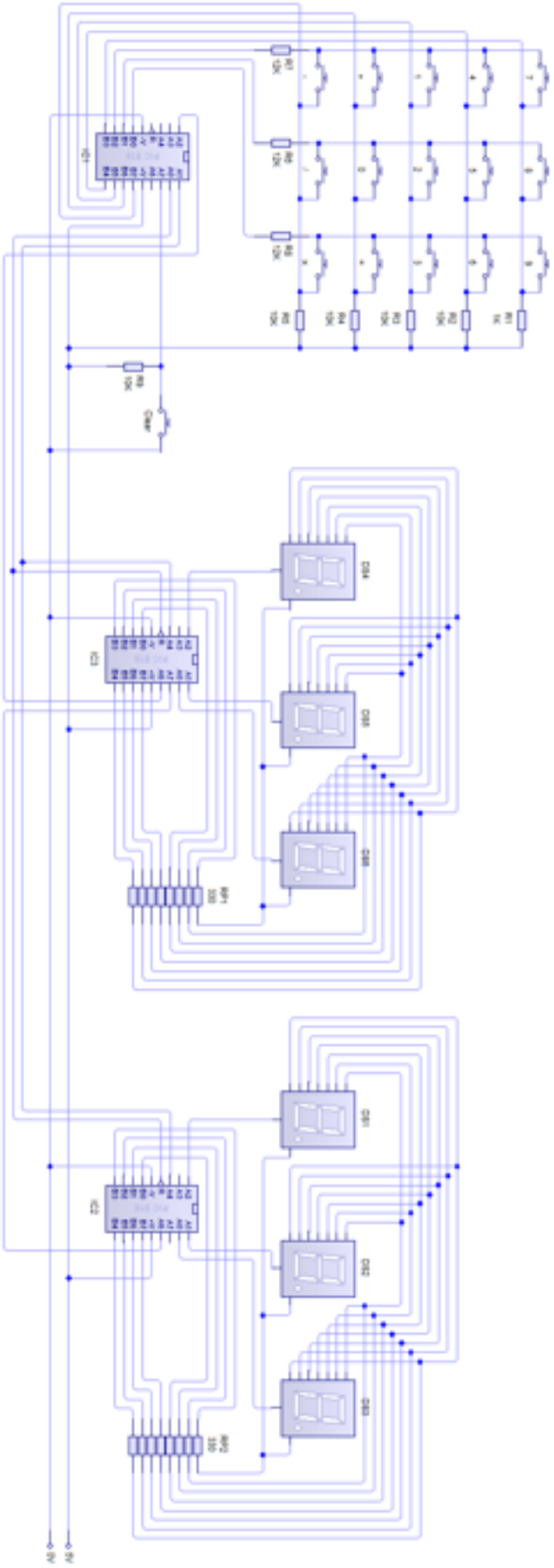
Picture of Breadboard layout:



FlowChart of PIC #1 Program:



Circuit Diagram:



Testing:

-Simple Addition:
Button order: 2 + 2 =
Expected Output: 4

-Exact Division:
Button order: 50 / 5 =
Expected Output: 10

-All Clear:
BO: 5 + 223 CLR CLR 2 =
Expected Output: 2

-Addition with carry:
Button order: 255 + 2 =
Expected Output: 267

-Remainder Division:
Button order: 51 / 5 =
Expected Output: 10r.

-Overflow:
Button order: 256 * 256 =
Expected Output: (First Character) F.

-Subtraction Positive:
Button order: 300 - 3 =
Expected Output: 297

-Maths Error:
Button order: 3 / 0 =
Expected Output: (First Character) E.

-Subtraction Negative:
Button order: 10 - 30 =
Expected Output: -20

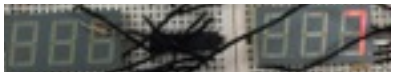
-Auto Evaluation:
Button order: 2 * 6 + 4 =
Expected Output: 16

-Multiplication:
Button order: 4 * 8 =
Expected Output: 32

-Repeat operation:
Button order: 4 + 3 ==
Expected Output: 10

-Multiplication with carry:
Button order: 201 * 199 =
Expected Output: 39999

-Clear:
BO: 5 + 223 CLR 2 =
Expected Output: 7



```
;-----------------------------------------------------------;
; PIC#1 Code                                                ;
;-----------------------------------------------------------;
; Program title: 4 operator calculator                      ;
;-----------------------------------------------------------;
; Written by: Michal Adamkiewicz                            ;
;-----------------------------------------------------------;
; Date:  19th September 2014                                ;
;-----------------------------------------------------------;
; Version:          1.0                                     ;
;-----------------------------------------------------------;
; Device:  PIC16F627A                                       ;
;-----------------------------------------------------------;
; Oscillator: Internal 48 kHz                               ;
;-----------------------------------------------------------;



 LIST  P=PIC16F627A
    ;select device
    ;Tells MPLAB what processor IC is being used

 INCLUDE  c:\program files (x86)\microchip\MPASM Suite
\P16F627A.inc
    ;include header file
    ;from default location
    ;tells the MPLAB where to find the files

 __config 0x3F10
    ;sets config to; internal  I/O, no watchdog,Power
    ;up timer on, master Reset off,
    ;no brown-out, no LV program, no read protect,
    ;no code protect

;-----------------------------------------------------------;
; DEFINE REGISTERS                                          ;
;-----------------------------------------------------------;

    cblock  0x20
temp
temp_value_lo
Temp_value_hi                ;temporary registers used in loop
                            counting and space for calculations
                            Operator
store                        ;stores code for the next operation
                            to execute
value_hi
value_lo
                             ;stores currently displayed value
old_value_hi
old_value_lo
                             ;stores non displayed operand
```

```
out_value_lo
out_value_hi
                              ;temporary registers used in
                         calculations
update_flag
                              ;contains various bit flags:
                             0-screen update 7-minus sign
                             6-temp storage 5-operator repeat
                              ;4-remainder note
                              3-all clear counter
                              2-multiplication overflow
                    flag
rep_value_lo
rep_value_hi
                              ;store values for repeated
                    operations
    endc
                              ;INITIALIZATION
MOVLW d'07'
MOVWF CMCON                   ;disable comparators
init    BSF STATUS, RP0       ;select bank1 for setup
    BCF PCON, OSCF            ;selects 48 kHz oscillator
    MOVLW b'11111000'         ;configure PORTB for button
                    multiplexing
    MOVWF TRISB
    MOVLW b'10000000'
    MOVWF TRISA               ;configure PORTA for clear of A7 and
                             A0,1,2 as display outputs
                              ;PORTA 0=clear 1=clock 2=data,
                         7=clear
    BCF STATUS, RP0           ;return to bank0 for program
                    operation
    BSF PORTA,0               ;prime display modules
                              ;clear registers to prevent ghosting
                             from previous runs
    CLRF temp
    CLRF temp_value_lo
    CLRF temp_value_hi
    CLRF store
    CLRF operator
    CLRF value_hi
    CLRF value_lo
    CLRF old_value_hi
    CLRF old_value_lo
    CLRF out_value_lo
    CLRF out_value_hi
    CLRF rep_value_lo
    CLRF rep_value_hi
    CLRF update_flag

main                          ;wait for till input loop -
                    multiplexes buttons
```

```
    BCF PORTA,0                ;display values
    CLRF PORTB                 ;sets all multiplex columns off
    BTFSC PORTA,7              ;check if clear button was pressed
    GOTO no_clear_calc
clear_debounce BTFSS PORTA,7
    GOTO clear_debounce
    BTFSC update_flag,3        ;test how many times clear button
                               was pressed
    GOTO all_clear
    BSF update_flag,3          ;count one button press
    CLRF value_lo              ;if yes clear all relevant registers
    CLRF value_hi
    CLRF operator
    BCF update_flag,0
    CALL write_value           ;will leave 0 character
    GOTO no_clear_calc
all_clear                      ;second time button was pressed
    CLRF value_lo              ;if yes clear all relevant registers
    CLRF value_hi
    CLRF old_value_lo
    CLRF old_value_hi
    CLRF rep_value_lo
    CLRF rep_value_hi
    CLRF operator
    CLRF update_flag
    CLRF store                 ;Artificially prepare for
                    write_value
    BSF PORTA,0
    CALL no_negative           ;Clear all characters
no_clear_calc                  ;no clear took place
    BSF PORTB,0                ;Enable left column for multiplexing
    MOVF PORTB,W               ;copy inputs to temp
    MOVWF temp
    XORLW D'1'
    BTFSC STATUS, Z            ;check if button was pressed
    GOTO bone                  ;button not pressed
bpone MOVF PORTB,W             ;debouncing loop- wait till unpress
    XORLW D'1'
    BTFSs STATUS, Z            ;button unpressed?
    GOTO bpone
    GOTO button_decode         ;decode button code functionality
bone BCF PORTB,0               ;Disable left column for
                    multiplexing
    BSF PORTB,1                ;Enable centre column for
                    multiplexing
    MOVF PORTB,W               ;copy inputs to W
    MOVWF temp
    XORLW D'2'
    BTFSC STATUS, Z            ;check if button was pressed
    GOTO btwo                  ;button not pressed
bptwo MOVF PORTB,W             ;debouncing loop- wait till unpress
    XORLW D'2'
```

```
    BTFSs STATUS, Z           ;button unpressed?
    GOTO bptwo
    GOTO button_decode        ;decode button code functionality
btwo BCF PORTB,1              ;Disable centre column for
                    multiplexing
    BSF PORTB,2               ;Enable centre column for
                    multiplexing
    MOVF PORTB,W              ;copy inputs to W
    MOVWF temp
    XORLW D'4'
    BTFSC STATUS, Z           ;check if button was pressed
    GOTO bthr                 ;button not pressed
bpthr MOVF PORTB,W            ;debouncing loop- wait till unpress
    XORLW D'4'
    BTFSs STATUS, Z           ;button unpressed?
    GOTO bpthr
    GOTO button_decode        ;decode button code functionality
bthr BCF PORTB,2              ;Disable right column for
                    multiplexing
    GOTO main                 ;loop while awaiting input

segment_look ADDWF PCL,F      ;Appearances of characters on screen
                              ;7-Decimal Point
    RETLW b'01011111'         ;0
    RETLW b'00000101'         ;1
    RETLW b'01110110'         ;2
    RETLW b'01110101'         ;3
    RETLW b'00101101'         ;4
    RETLW b'01111001'         ;5
    RETLW b'01111011'         ;6
    RETLW b'01000101'         ;7
    RETLW b'01111111'         ;8
    RETLW b'01111101'         ;9
    GOTO unexpected_error     ;something unpredicted happened

button_decode                 ;subroutine decodes button codes
    BCF update_flag,3         ;resets clear counter
    BTFSS update_flag,0       ;test if previous button press was
                             operator and value needs clearing
    GOTO no_update
    BCF update_flag,0         ;clear flag
    CLRF value_hi             ;clear value
    CLRF value_lo
no_update
    MOVF temp,w
    XORLW D'132'
    BTFSC STATUS,Z            ;does temp match the buttons code?
    GOTO sub_act
    MOVF temp,w
    XORLW D'130'
    BTFSC STATUS,Z            ;does temp match the buttons code?
    GOTO div_act
```

```
    MOVF temp,w
    XORLW D'129'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO mul_act
    MOVF temp,w
    XORLW D'68'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO add_act
    MOVF temp,w
    XORLW D'66'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO zero_act
    MOVF temp,w
    XORLW D'65'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO equ_act
    MOVF temp,w
    XORLW D'36'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO one_act
    MOVF temp,w
    XORLW D'34'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO two_act
    MOVF temp,w
    XORLW D'33'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO three_act
    MOVF temp,w
    XORLW D'20'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO four_act
    MOVF temp,w
    XORLW D'18'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO five_act
    MOVF temp,w
    XORLW D'17'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO six_act
    MOVF temp,w
    XORLW D'12'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO seven_act
    MOVF temp,w
    XORLW D'10'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO eight_act
    MOVF temp,w
    XORLW D'9'
    BTFSC STATUS,Z          ;does temp match the buttons code?
    GOTO nine_act
```

```
    GOTO multi_key            ;more than 1 button was pressed,
                          display error code

equ_act
    BTFSC update_flag,5       ;was equals pressed twice?
    GOTO equ_act_rep
    BSF update_flag,5         ;notify that previous button press
                              was equals
    MOVFW value_lo            ;store current argument for future
                              double press
    MOVWF rep_value_lo
    MOVFW value_hi
    MOVWF rep_value_hi
    CALL equ_function         ;run appropriate calculation
    CALL write_value          ;push value to display
    GOTO main                 ;return to main loop
equ_act_rep                   ;Equals double press
    MOVFW value_lo            ;use value as old argument
    MOVWF old_value_lo
    MOVFW value_hi
    MOVWF old_value_hi
    MOVFW rep_value_lo        ;used rep_value as current argument
    MOVWF value_lo
    MOVFW rep_value_hi
    MOVWF value_hi
    CALL equ_function         ;run appropriate calculation
    CALL write_value          ;push value to display
    GOTO main                 ;RETURN to main loop

mul_act
    BTFSC update_flag,5       ;stops in auto-calculating repeated
                              operation
    GOTO mul_rep_skip
    CALL equ_function         ;If yes calculate them
    CALL write_value          ;And display them
mul_rep_skip
    BCF update_flag,5         ;Resets repetition flag
    MOVLW D'2'                ;remember which operator is active
                              in register
    MOVWF operator
    MOVF value_hi, W          ;move value to old_value to be used
                              a previous argument
    MOVWF old_value_hi
    MOVF value_lo, W
    MOVWF old_value_lo
    BSF update_flag,0         ;clear value next time button is
                          presed
    GOTO main                 ;wait for button input

div_act
    BTFSC update_flag,5       ;stops in auto-calculating repeated
                              operation
```

```
    GOTO div_rep_skip
    CALL equ_function         ;If yes calculate them
    CALL write_value          ;And display them
div_rep_skip
    BCF update_flag,5         ;Resets repetition flag
    MOVLW D'3'                ;remember which operator is active
                              in register
    MOVWF operator
    MOVF value_hi, W          ;move value to old_value to be used
                              a previous argument
    MOVWF old_value_hi
    MOVF value_lo, W
    MOVWF old_value_lo
    BSF update_flag,0         ;clear value next time button is
                          pressed
    GOTO main                 ;wait for button input

sub_act
    BTFSC update_flag,5       ;stops in auto-calculating repeated
                              operation
    GOTO sub_rep_skip
    CALL equ_function         ;If yes calculate them
    CALL write_value          ;And display them
sub_rep_skip
    BCF update_flag,5         ;Resets repetition flag
    MOVLW D'1'                ;remember which operator is active
                              in register
    MOVWF operator
    MOVF value_hi, W          ;move value to old_value to be used
                              a previous argument
    MOVWF old_value_hi
    MOVF value_lo, W
    MOVWF old_value_lo
    BSF update_flag,0         ;clear value next time button is
                          pressed
    GOTO main                 ;wait for button input

add_act
    BTFSC update_flag,5       ;stops in auto-calculating repeated
                              operation
    GOTO add_rep_skip
    CALL equ_function         ;If yes calculate them
    CALL write_value          ;And display them
add_rep_skip
    BCF update_flag,5         ;Resets repetition flag
    MOVLW D'0'                ;remember which operator is active
                              in register
    MOVWF operator
    MOVF value_hi, W          ;move value to old_value to be used
                              a previous argument
    MOVWF old_value_hi
    MOVF value_lo, W
```

```
    MOVWF old_value_lo
    BSF update_flag,0          ;clear value next time button is
                          pressed
    GOTO main                  ;wait for button input
one_act MOVLW D'1'             ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
two_act MOVLW D'2'             ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
three_act MOVLW D'3'           ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
four_act MOVLW D'4'            ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
five_act MOVLW D'5'            ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
six_act MOVLW D'6'             ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
seven_act MOVLW D'7'           ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
eight_act MOVLW D'8'           ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
nine_act MOVLW D'9'            ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits
zero_act MOVLW D'0'            ;copy digit to store and append to
                               end of value base10
    MOVWF store
    GOTO shift_digits

shift_digits
                               ;Multiply by 10 by Multiplying by
                               2,8 and adding
                               ;Multiply by 2, by register rotation
    CLRF temp
    RLF value_lo, F            ;2*
    BCF value_lo,0             ;ensures zero is shifted in
```

```
    BTFSC STATUS, C           ;uses temp as carry flag holder
    BSF temp,0
    RLF value_hi, F           ;2*
    BCF value_lo,0            ;ensures zero is shifted in
    BTFSC STATUS,C            ;test to ensure number is in 16bit
                              range
    GOTO overflow
    BTFSC temp,0              ;carry bit in temporary flag
    BSF value_hi,0            ;
                              ;keep temp_value for later addition
    MOVF value_lo, W
    MOVWF temp_value_lo
    MOVF value_hi, W
    MOVWF temp_value_hi
                              ;Multiply by 4, by register rotation
    CLRF temp
    RLF value_lo, F           ;2*
    BCF value_lo,0            ;ensures zero is shifted in
    BTFSC STATUS, C           ;uses temp as carry flag holder
    BSF temp,0
    RLF value_hi, F           ;2*
    BCF value_hi,0            ;ensures zero is shifted in
    BTFSC STATUS,C            ;test to ensure number is in 16bit
                              range
    GOTO overflow
    BTFSC temp,0              ;carry bit in temporary flag
    BSF value_hi,0
                              ;Multiply by 8, by register rotation
    CLRF temp
    RLF value_lo, F           ;2*
    BCF value_lo,0            ;ensures zero is shifted in
    BTFSC STATUS, C           ;uses temp as carry flag holder
    BSF temp,0
    RLF value_hi, F           ;2*
    BCF value_hi,0            ;ensures zero is shifted in
    BTFSC STATUS,C            ;test to ensure number is in 16bit
                              range
    GOTO overflow
    BTFSC temp,0              ;carry bit in temporary flag
    BSF value_hi,0
                              ;Add x*2 + x*8 to make 10*x
    MOVF temp_value_lo,W
    ADDWF value_lo,F          ;Add lowest bits
    BTFSS STATUS,C            ;Test for carry to higher bits
    GOTO shift_digits_no_overflow;No carry
    INCF temp_value_hi, F     ;increment hi from carry
    BTFSC STATUS,Z            ;if carry out occurred
              temp_value_hi==0
    GOTO overflow             ;display overflow error
shift_digits_no_overflow
    MOVF temp_value_hi,W      ;Add higher byte
    ADDWF value_hi,F
```

```
    BTFSC STATUS,C            ;test for overflow
    GOTO overflow
                              ;Add digit stored in store
    MOVF store,W              ;Add store to value
    ADDWF value_lo,F
    MOVLW d'0'                ;Trick for conditional increment and
                             test without labels
    BTFSC STATUS,C            ;If carry, add 1 instead of 0
    MOVLW d'1'
    ADDWF value_hi,F          ;Add 1 or 0 to execute or not
                         execute carry
    BTFSC STATUS,C            ;Test for overflow
    GOTO overflow
    CALL write_value          ;update displayed value
    GOTO main                 ;RETURN to main loop

equ_function                  ;Actually does the execution of
                         calculating functions
    MOVF operator,W           ;Tests if operator matches + code
    XORLW D'0'
    BTFSC STATUS, Z           ;If yes go to designated subroutine
    GOTO actual_add
    MOVF operator,W           ;Tests if operator matches - code
    XORLW D'1'
    BTFSC STATUS, Z           ;If yes go to designated subroutine
    GOTO actual_sub
    MOVF operator,W           ;Tests if operator matches * code
    XORLW D'2'
    BTFSC STATUS, Z           ;If yes go to designated subroutine
    GOTO actual_mul
    MOVF operator,W           ;Tests if operator matches / code
    XORLW D'3'
    BTFSC STATUS, Z           ;If yes go to designated subroutine
    GOTO actual_div
    GOTO unexpected_error     ;Abnormal operator code, display
                         error code
back_to_equ_function RETURN   ;After calculations RETURN to CALL
                             point

actual_add                    ;subroutine for adding old_value and
                             value
    MOVF old_value_lo,W       ;Add low bytes
    ADDWF value_lo,F
    BTFSS STATUS, C           ;Test if carry out of low byte
                         occurred
    GOTO add_no_carry
    INCF old_value_hi, F      ;Execute carry
    BTFSC STATUS,Z            ;If hi byte carried out it==0
    GOTO overflow             ;display overflow code
add_no_carry
    MOVF old_value_hi,W       ;Add hi bytes
    ADDWF value_hi,F
```

```
    BTFSC STATUS, C           ;Test if overflow occurred
    GOTO overflow
    GOTO back_to_equ_function ;equ_function for display and
                         further RETURN

actual_mul                    ;Multiplication via binary algorithm
    MOVFW value_lo            ;keep value in temp_value for
                         manipulation
    MOVWF temp_value_lo
    MOVFW value_hi
    MOVWF temp_value_hi
    CLRF value_hi             ;clear value - it will be used as
                             output
    CLRF value_lo
    CLRF temp                 ;temp will count number of bit
                         shifts executed - target 16
    BCF update_flag,2
mul_loop_bit
    INCF temp,F               ;increment loop counter
    BTFSS temp_value_lo,0     ;Tests lowest bit to see if value is
                             to be added
    GOTO mul_no_add
                             ;Adds old_value to value
    BTFSC update_flag,2
    GOTO overflow
    MOVF old_value_lo,W       ;Adds lower bytes
    ADDWF value_lo,F
    BTFSS STATUS, C           ;Tests for carry
    GOTO mul_no_carry
    INCF value_hi, F          ;Executes carry to hi byte
    BTFSC STATUS,Z            ;Tests if 16bit overflow occurred
    GOTO overflow
mul_no_carry
    MOVF old_value_hi,W       ;Added high bytes
    ADDWF value_hi,F
    BTFSC STATUS, C           ;Tests if 16bit overflow occurred
    GOTO overflow
mul_no_add
                             ;Multiplies old_value by 2
    RLF old_value_lo,F        ;rotate lower byte
    RLF old_value_hi,F        ;rotate higher byte - carry handled
                             by STATUS,C
    BTFSC STATUS,C            ;Test if 16bit overflow occurred
    BSF update_flag,2
    BCF old_value_lo,0        ;Ensure new bit is==0
    RRF temp_value_hi,F       ;Prepare new bit for conditional
                         addition
    RRF temp_value_lo,F
    BCF temp_value_hi,7       ;Ensure new bit is=0-not necessary
                             but keeps code nice
    MOVFW temp                ;Testing if loop run 16 times
    XORLW d'16'
```

```
    BTFSS STATUS,Z              ;If no repeat
    GOTO mul_loop_bit
end_mul GOTO back_to_equ_function
                                ;equ_function for display and
                            further RETURN

actual_sub
    BCF update_flag,7           ;clears the minus sign flag
    MOVF value_lo,W             ;subtracts low bytes
    SUBWF old_value_lo, F
    BTFSC STATUS, C             ;tests if borrow occurred
    GOTO sub_no_carry
    DECF old_value_hi, F        ;borrows bit from high byte
    MOVFw old_value_hi
    XORLW d'255'                ;Tests if borrow there was something
                                to borrow from
    BTFSC STATUS,Z
    BSF update_flag,7           ;If no then set minus sign
sub_no_carry
    MOVF value_hi,W             ;Subtracts high bytes
    SUBWF old_value_hi, F
    BTFSS STATUS, C             ;If borrow occurred - nothing to
                            borrow from
    BSF update_flag,7           ;Set minus sign
    BTFSS update_flag,7         ;Test if numbers need to be adjusted
                                for minus
    GOTO sub_no_neg
                                ;Execute two's compliment conversion
                                if minus is present
    MOVFW old_value_lo          ;XOR with 11111111 == flip all bits
    XORLW d'255'                ;this is twos compliment notation
    MOVWF old_value_lo
    MOVFW old_value_hi          ;XOR with 11111111 == flip all bits
    XORLW d'255'                ;this is twos compliment notation
    MOVWF old_value_hi
    INCF old_value_lo,F         ;Compensate for zero not being
                            negative by adding 1
    BTFSC STATUS,Z
    INCF old_value_hi
sub_no_neg
    MOVF old_value_lo, W        ;Store result in value so it will be
                                displayed
    MOVWF value_lo
    MOVF old_value_hi, W
    MOVWF value_hi
sub_end GOTO back_to_equ_function
                                ;equ_function for display and
                            further RETURN

actual_div                      ;Division via repeated subtraction
    MOVFW value_lo              ;Test if lower byte of right operand
                                is ==0
```

```
    XORLW d'0'
    BTFSS STATUS,Z
    GOTO no_div_zero
    MOVFW value_hi            ;Test if higher byte of right
                          operand is ==0
    XORLW d'0'
    BTFSC STATUS,Z
    GOTO unexpected_error     ;Output an Error as division by 0 is
                             undefined

no_div_zero
    MOVFW value_lo            ;copy value to temp_value for
                          manipulation
    MOVWF temp_value_lo
    MOVFW value_hi
    MOVWF temp_value_hi
    CLRF value_lo             ;value will be used as output
    CLRF value_hi
div_sub_loop
    MOVFW temp_value_lo       ;Subtract lower bytes
    SUBWF old_value_lo,F
    BTFSC STATUS,C            ;Check for a borrow from high byte
    GOTO div_no_borrow
    DECF old_value_hi,F       ;Borrow from high byte
    MOVFW old_value_hi
    XORLW d'255'              ;Test if high byte went negative
    BTFSC STATUS,Z
    GOTO div_sub_end          ;If yes GOTO ending
div_no_borrow
    MOVFW temp_value_hi       ;Subtract higher bytes
    SUBWF old_value_hi,F
    BTFSS STATUS,C            ;Check if result went negative -
                          borrow from non existent byte
    GOTO div_sub_end
    INCF value_lo,F           ;Increment counter on number of
                          successful subtractions
    BTFSC STATUS,Z
    INCF value_hi,F           ;Carry into high byte
    GOTO div_sub_loop
div_sub_end
    MOVFW temp_value_lo       ;fix offset in low byte for one too
                             many subtractions
    ADDWF old_value_lo,F
    BTFSC STATUS,C            ;carry addition to high byte
    INCF old_value_hi,F
    MOVFW temp_value_hi       ;fix offset in high byte for one too
                             many subtractions
    ADDWF old_value_hi,F
    BCF update_flag,4         ;Remainder flag=0
    MOVFW old_value_lo
    XORLW d'0'                ;Test if division was exact in low
                             byte
```

```
    BTFSS STATUS,Z
    BSF update_flag,4          ;Remainder flag=1
    MOVFW old_value_hi
    XORLW d'0'                 ;Test if division was exact in high
                               byte
    BTFSS STATUS,Z
    BSF update_flag,4
end_div GOTO back_to_equ_function

write_value                    ;writes value_hi,value_lo to the
                          display
    BSF PORTA,0                ;prepare display modules for data
                               transfer
    MOVF value_hi, W           ;use temp_value to prevent changing
                               value as it will be used in
                     calculations
    MOVWF temp_value_hi
    MOVF value_lo, W
    MOVWF temp_value_lo
    CLRF store                 ;store will count the number of
                          characters already displayed target-6
    BTFSS update_flag,4        ;Was division result exact or with a
                               remainder?
    GOTO no_remainder
    INCF store,F               ;Increment number of characters
                          displayed
    MOVLW b'10100010'          ;Display code for reminder -looks
                               like a r.
    MOVWF out_value_lo
    CLRF temp                  ;Temp will count number of bits
                          shifted - target 8 (7 segments +
                     decimal point)
shift_loop_remainder
    INCF temp,F                ;Increment counter of bits shifted
                               out
    BCF PORTA,2                ;Set data pin to 0
    BTFSC out_value_lo,0       ;If data pin is meant to be 1
    BSF PORTA,2                ;make it 1
    BCF PORTA,1                ;raise the clock - shift value
    RRF out_value_lo,F         ;prepare to shift out next bit
    BSF PORTA,1                ;fall the clock
    MOVFW temp                 ;Tests if all 8 bits were shifted
                               out
    XORLW d'8'
    BTFSS STATUS,Z             ;If yes continue to next step
    GOTO shift_loop_remainder
no_remainder
    CLRF out_value_lo          ;out_value will store integer part
                               of division by 10
    CLRF out_value_hi          ;temp will store remainder of
                          division by 10
write_loop
```

```
    INCF out_value_lo, F      ;counts the number of 10s subtracted
    BTFSC STATUS,Z            ;carry when counting 10s
    INCF out_value_hi, F      ;Division ensures no 16bit overflow
    MOVLW D'10'
    SUBWF temp_value_lo,F     ;actually subtracts the 10 from temp
    BTFSC STATUS,C            ;checks if a borrow occurred
    GOTO write_loop           ;Subtract another 10
    MOVF temp_value_hi,W
    BTFSC STATUS,Z            ;If all temp_value is <0 division
                              finishes. Must be zero at one point
                              as 10<255
    GOTO write_digit_calc     ;Process done, move to next step
    DECF temp_value_hi,F      ;borrow from hi byte
    GOTO write_loop           ;Subtract another 10
write_digit_calc
    MOVLW D'10'               ;Compensates for 10 subtracted to
                              make temp_value negative
    ADDWF temp_value_lo,F
    DECF out_value_lo,F       ;Compensates counter for added 10
    MOVFW out_value_lo
    XORLW d'255'              ;check borrow from hi byte
    BTFSC STATUS,Z
    DECF out_value_hi,F       ;Execute borrow from high
    INCF store,F              ;Tracks number of characters written
                              to display
    MOVFW temp_value_lo       ;Look up code for calculated
                    character
    CALL segment_look         ;Access table
    MOVWF temp_value_lo
    CLRF temp                 ;Temp will count number of bits
                         shifted - target 8 (7 segments +
                    decimal point)
shift_loop_main
    INCF temp,F               ;Count number of bits shifted out
    BCF PORTA,2               ;Copy temp_value_lo,0 into PORTA,2
    BTFSC temp_value_lo,0
    BSF PORTA,2
    BCF PORTA,1               ;Raise the clock - shift value
    RRF temp_value_lo,F       ;Prepare next bit in temp_value_lo
                              while waiting for display value
    BSF PORTA,1               ;Fall the clock
    MOVFW temp                ;Test if 8 bits already shifted out
    XORLW d'8'
    BTFSS STATUS,Z
    GOTO shift_loop_main      ;If no shift out next bit
    BCF update_flag,6         ;Clean bit for temporary bit flag
    MOVF out_value_hi,W
    MOVWF temp_value_hi       ;Copy low byte of integer part of
                              division by 10 for next division
    BTFSC STATUS, Z           ;If ==0 to copy remember in
                    temporary flag
    BSF update_flag,6
```

```
    MOVF out_value_lo, W
    MOVWF temp_value_lo       ;Copy hi byte of integer part of
                          division by 10 for next division
    BTFSS STATUS, Z           ;Test if ==0 for finishing condition
    GOTO no_remainder         ;Run division loop again
    BTFSS update_flag,6       ;If both hi and low byte ==0
                    continue to next section
    GOTO no_remainder         ;Run division loop again
    BTFSS update_flag,7       ;If minus sign flag clear then
    GOTO no_negative          ;Don't display a minus
    INCF store,F              ;Count number of characters written
    MOVLW b'00100000'         ;Display code for a minus sign -
    MOVWF temp_value_lo
    CLRF temp                 ;Temp will count number of bits
                         shifted - target 8 (7 segments +
                    decimal point)
shift_loop_neg
    INCF temp,F               ;Count number of bits shifted out
    BCF PORTA,2               ;Copy temp_value_lo,0 into PORTA,2
    BTFSC temp_value_lo,0
    BSF PORTA,2
    BCF PORTA,1               ;Raise the clock - shift value
    RRF temp_value_lo,F       ;Prepare next bit in temp_value_lo
                             while waiting for display value
    BSF PORTA,1               ;Fall the clock
    MOVFW temp                ;Test if 8 bits already shifted out
    XORLW d'8'
    BTFSS STATUS,Z
    GOTO shift_loop_neg       ;If no shift out next bit
no_negative
    BCF update_flag,7         ;Clear minus sign flag
    BCF update_flag,4         ;Clear reminder flag
    MOVFW store               ;Tests if all 6 characters are
                         written to
    XORLW d'6'
    BTFSC STATUS,Z
    RETURN                    ;If yes: RETURN to CALL point
    INCF store,F              ;Count number of characters written
                             to display
    BCF PORTA,2               ;Shift outs b'00000000' - an empty
                             character
    CLRF temp                 ;Temp will count number of bits
                         shifted - target 8 (7 segments +
                    decimal point)
shift_loop_null
    INCF temp,F               ;Count number of bits shifted out
    BCF PORTA,1               ;Raise the clock - shift value
                             ;Delay not need as display PIC runs
                             with a faster clock speed
    BSF PORTA,1               ;Fall the clock
    MOVFW temp                ;Test if 8 bits already shifted out
    XORLW d'8'
```

```
    BTFSS STATUS,Z
    GOTO shift_loop_null      ;If no shift out next bit
    GOTO no_negative
                              ;Subroutines for handling exceptions
                             in the code

multi_key                     ;triggers when two buttons are
                         pressed simultaneously and PIC can't
                         distinguish them
    BSF PORTA,0               ;Prepare display modules for data
    MOVLW b'10001111'         ;multi key press code. Two Parallel
                             line followed by a period
    MOVWF temp_value_lo
    CLRF temp                 ;Temp will count number of bits
                         shifted - target 8 (7 segments +
                    decimal point)
shift_loop_multi
    INCF temp,F               ;Count number of bits shifted out
    BCF PORTA,2               ;Copy temp_value_lo,0 into PORTA,2
    BTFSC temp_value_lo,0
    BSF PORTA,2
    BCF PORTA,1               ;Raise the clock - shift value
    RRF temp_value_lo,F       ;Move to next bit in temp_value_lo
    BSF PORTA,1               ;Fall the clock
    MOVFW temp                ;Test if 8 bits already shifted out
    XORLW d'8'
    BTFSS STATUS,Z
    GOTO shift_loop_multi     ;If no shift out next bit
    GOTO main                 ;resume normal operation
overflow                      ;Indicates program encountered a
                         16bit overflow while preforming large
                         calculation
    BSF PORTA,0               ;Prepare display modules for data
    MOVLW b'11101010'         ;Display a F. symbol to indicate
                         operation Failed
    MOVWF temp_value_lo
    CLRF temp                 ;Temp will count number of bits
                         shifted - target 8 (7 segments +
                    decimal point)
shift_loop_overflow
    INCF temp,F               ;Count number of bits shifted out
    BCF PORTA,2               ;Copy temp_value_lo,0 into PORTA,2
    BTFSC temp_value_lo,0
    BSF PORTA,2
    BCF PORTA,1               ;Raise the clock - shift value
    RRF temp_value_lo,F       ;Move to next bit in temp_value_lo
    BSF PORTA,1               ;Fall the clock
    MOVFW temp                ;Test if 8 bits already shifted out
    XORLW d'8'
    BTFSS STATUS,Z
    GOTO shift_loop_overflow  ;If no shift out next bit
    GOTO main                 ;resume normal operation
```

```
unexpected_error              ;Signifies a unclassified error
                          occurred. EG division by 0
    BSF PORTA,0               ;Prepare display modules for data
    MOVLW b'11111010'         ;Display a E. symbol to indicate and
                              Error
    MOVWF temp_value_lo
    CLRF temp                 ;Temp will count number of bits
                          shifted - target 8 (7 segments +
                    decimal point)
shift_loop_error
    INCF temp,F               ;Count number of bits shifted out
    BCF PORTA,2               ;Copy temp_value_lo,0 into PORTA,2
    BTFSC temp_value_lo,0
    BSF PORTA,2
    BCF PORTA,1               ;Raise the clock - shift value
    RRF temp_value_lo,F       ;Move to next bit in temp_value_lo
    BSF PORTA,1               ;Fall the clock
    MOVFW temp                ;Test if 8 bits already shifted out
    XORLW d'8'
    BTFSS STATUS,Z
    GOTO shift_loop_error     ;If no shift out next bit
    GOTO main                 ;resume normal operation

    END                       ;Opcode need for compilation. PIC
                              should never reach this
```

```
;-------------------------------------------------------------;
; PIC#2,3 Code                                                ;
;-------------------------------------------------------------;
; Program title: 8-bit Multiplexing Shift Register            ;
;-------------------------------------------------------------;
; Written by: Michal Adamkiewicz                              ;
;-------------------------------------------------------------;
; Date:  19th September 2014                                  ;
;-------------------------------------------------------------;
; Version:          1.0                                       ;
;-------------------------------------------------------------;
; Device:  PIC16F627                                          ;
;-------------------------------------------------------------;
; Oscillator: Internal 4 MHz                                  ;
;-------------------------------------------------------------;

 LIST  P=PIC16F627A ;select device
    ;Tells MPLAB what processor IC is being used

 INCLUDE  c:\program files (x86)\microchip\MPASM Suite
\P16F627A.inc
    ;include header file
    ;from default location
    ;tells the MPLAB where to find the files


 __config 0x3F10
    ;sets config to; internal  I/O, no watchdog,Power
    ;up timer on, master Reset off,
    ;no brown-out, no LV program, no read protect,
    ;no code protect
;-------------------------------------------------------------;
; DEFINE REGISTERS                                            ;
;-------------------------------------------------------------;

cblock  0x20
     dig_one             ;Three registers store what to be
               displayed on the 3 characters
     dig_two
     dig_three
     loop                ;This stores the current loop time
endc



init
     MOVLW d'07'
     MOVWF CMCON         ;Disable comparators
     BSF STATUS, RP0     ;select bank1 for setup
     BSF PCON, OSCF      ;select 4 MHz
     MOVLW b'01110000'
     MOVWF TRISA         ;set PortA as inputs on designated pins
```

```
     MOVLW b'00000000'
     MOVWF TRISB          ;set PortB all outputs
     BCF STATUS, RP0      ;return to bank0 for program operation

     CLRF dig_one         ;clear registers to prevent ghosting
     CLRF dig_two
     CLRF dig_three

main
     BTFSC PORTA,5        ;Test if Programing mode (shift in) was
                          activated
     GOTO prog

     BCF PORTA,2          ;Prepare pin to sink current from diodes
     MOVFW dig_three      ;Move stored values onto PORTB
     MOVWF PORTB
     MOVLW b'00011111'    ;Delay amount established experimentally
     MOVWF loop
     CALL delay           ;Delay ensures display is visible to
                    human eye
     BSF PORTA,2

     BCF PORTA,1          ;Prepare pin to sink current from diodes
     MOVFW dig_two        ;Move stored values onto PORTB
     MOVWF PORTB
     MOVLW b'00011111'    ;Delay amount established experimentally
     MOVWF loop
     CALL delay           ;Delay ensures display is visible to
                    human eye
     BSF PORTA,1

     BCF PORTA,0          ;Prepare pin to sink current from diodes
     MOVFW dig_one        ;Move stored values onto PORTB
     MOVWF PORTB
     MOVLW b'00011111'    ;Delay amount established experimentally
     MOVWF loop
     CALL delay
     BSF PORTA,0          ;Delay ensures display is visible to
                    human eye
     GOTO main            ;Loop back to top

prog                 ;Programing mode - PIC accepts serial data
     BTFSS PORTA,5        ;Test if Programing mode disabled
     GOTO main            ;return to multiplexing
     BTFSC PORTA,4        ;Wait for Clock pin to go low
     GOTO prog
wait BTFSS PORTA,4  ;Wait for it to go back to high
     GOTO wait

     RRF dig_three, F     ;Shift register over by one bit
     BCF dig_three,7      ;Ensures register has bit
     BTFSC dig_two,0      ;carried from next register
```

```
      BSF dig_three,7

      RRF dig_two, F      ;Shift register over by one bit
      BCF dig_two,7       ;Ensures register has bit
      BTFSC dig_one,0     ;carried from next register
      BSF dig_two,7

      RRF dig_one, F      ;Shift register over by one bit
      BCF dig_one,7       ;Take the new bits value from
      BTFSC PORTA,6       ;the Data Pin
      BSF dig_one,7

      BCF PORTA,7         ;Copy last bit to Output pin
      BTFSC dig_three,0   ;for chaining to other modules
      BSF PORTA,7

      GOTO prog           ;Wait for another bit shift

delay DECFSZ loop, F      ;Count down until zero
       GOTO delay
      RETURN              ;end delay

      END                 ;Opcode need for compilation. PIC should
                         never reach this
```