```
$modde0cv

CSEG at 0
ljmp START

dseg at 30h
x: ds 4 ; 32-bits for variable ‘x’
y: ds 4 ; 32-bits for variable ‘y’
target: ds 4 ; 32 bit for variable 'target'
bcd: ds 5 ; 10-digit packed BCD (each byte stores 2 digits)
bseg
mf: dbit 1 ; Math functions flag
$include(math32.asm)

CSEG

    ; Look-up table for 7-seg displays
seg_table:
    DB 0C0H, 0F9H, 0A4H, 0B0H, 099H        ; 0 TO 4
    DB 092H, 082H, 0F8H, 080H, 090H        ; 4 TO 9

showBCD MAC
    ; Display LSD
    movA, %0
    anl a, #0fh
    movcA, @A+dptr
    mov %1,A
    ; Display MSD
    movA, %0
    swap a
    anl a, #0fh
    movcA, @A+dptr
    mov %2,A
ENDMAC

Display:
    mov dptr, #seg_table
    showBCD(bcd+0, HEX0, HEX1)
    showBCD(bcd+1, HEX2, HEX3)
    showBCD(bcd+2, HEX4, HEX5)
    ret

MYRLC MAC
    mov a, %0
    rlc a
    mov %0, a
ENDMAC

Shift_Digits:
    mov R0, #4 ; shift left four bits
Shift_Digits_L0:
    clr c
    MYRLC(bcd+0)
    MYRLC(bcd+1)
    MYRLC(bcd+2)
    MYRLC(bcd+3)
    MYRLC(bcd+4)
    djnz R0, Shift_Digits_L0
    ; R7 has the new bcd digit
    mov a, R7
    orl a, bcd+0
    mov bcd+0, a
```

```
    ; bcd+3 and bcd+4 don't fit in the 7-segment displays so make them zero
    clr a
    mov bcd+4, a
    ret

Wait50ms:
;33.33MHz, 1 clk per cycle: 0.03us
    mov R0, #30
L3: mov R1, #74
L2: mov R2, #250
L1: djnz R2, L1 ;3*250*0.03us=22.5us
    djnz R1, L2 ;74*22.5us=1.665ms
    djnz R0, L3 ;1.665ms*30=50ms
    ret

; Check if SW0 to SW9 are toggled up.  Returns the toggled switch in
; R7.  If the carry is not set, no toggling switches were detected.
ReadNumber:
    mov r4, SWA; Read switches 0 to 7
    mov a, SWB ; Read switches 8 to 9
    anl a, #0000001B ; Only two bits of SWB available
    mov r5, a
    mov a, r4
    orl a, r5
    jz ReadNumber_no_number
    lcall Wait50ms ; debounce
    mov a, SWA
    clr c
    subb a, r4
    jnz ReadNumber_no_number ; it was a bounce
    mov a, SWB
    anl a, #0000001B
    clr c
    subb a, r5
    jnz ReadNumber_no_number ; it was a bounce
    mov r7, #16 ; Loop counter
ReadNumber_L0:
    clr c
    mov a, r4
    rlc a
    mov r4, a
    mov a, r5
    rlc a
    mov r5, a
    jc ReadNumber_decode
    djnz r7, ReadNumber_L0
    sjmp ReadNumber_no_number
ReadNumber_decode:
    dec r7
    setb c
ReadNumber_L1:
    mov a, SWA
    jnz ReadNumber_L1
ReadNumber_L2:
    mov a, SWB
    jnz ReadNumber_L2
    ret
ReadNumber_no_number:
    clr c
    ret

START; Called once on start
```

```
    mov SP,#7FH
    ; set everything to 0
    clr a
    mov LEDRA, a
    mov LEDRB, a
    mov bcd+0, a
    mov bcd+1, a
    mov bcd+2, a
    mov bcd+3, a
    mov bcd+4, a
    lcall Display

    mov b, #1 ; b = 1 for addition
    setb LEDRA.0 ; Turn LEDR0 on to indicate addition

LOOP: ; Called forever
    mov LEDRA, b            ; Display Function select on the LEDs

    ; Check if number is being loaded into the calculator
    lcall ReadNumber
    jnc func_button ; If nothing loaded, skip
    lcall Shift_Digits
    lcall Display

    ;This handles cycling the function

func_button:
    jb KEY.3, load_button       ; If 'Function' key not pressed, skip
        jnb KEY.3, $        ; Jumps to itself until key is relased
        mov a, b            ; When the key is pressed, we double B (multiply by 2) to bitshift left one
        mov b, #02
        mul ab
        mov b, a            ; B is used to store the current function
        cjne a, #10000000B, LOOP ; If B is high enough it needs to overflow back down to 1
        mov b,  #00000001B          ; B back down to 1
        ljmp LOOP               ; Restart the loop


load_button:
    jb KEY.2, equal_button      ; If 'Load' key not pressed, skip
        jnb KEY.2, $        ; Jumps to itself until key is relased
        lcall bcd2hex       ; Convert the BCD number to hex in x
        lcall copy_xy       ; Copy x to y
        Load_X(0)           ; Clear x
        lcall hex2bcd       ; Convert result in x to BCD
        lcall Display       ; Display the new BCD number
        ljmp LOOP       ; Restart the loop

equal_button:
    jb KEY.1, LOOP      ; If 'equal' key not pressed, skip
        jnb KEY.1, $        ; Jumps to itself until key is relased
        lcall bcd2hex       ; Convert the BCD number to hex in x
        mov a, b

; Check for function depending on state
Addition:
    CJNE a, #00000001B, Subtraction
        lcall add32     ; x = x + y
        ljmp DISP_ANSWER

Subtraction:
    CJNE a, #00000010B, Multiplication
```

```
        lcall sub32     ; x = x - y
        ljmp DISP_ANSWER

Multiplication:
    CJNE a, #00000100B, Division
        lcall mul32     ; x = x * y
        ljmp DISP_ANSWER

Division:
    CJNE a, #00001000B, Remainder
        lcall div32     ; x = x / y
        ljmp DISP_ANSWER

Remainder:
    CJNE a, #00010000B, Percentage
        lcall mod32 ; x = x % y
        ljmp DISP_ANSWER

Percentage:
    CJNE a, #00100000B, Square_root
        lcall perce32       ; x = (x * y) / 100
        ljmp DISP_ANSWER

Square_root:
        lcall square_root32     ; x = sqrt(x)
        ljmp DISP_ANSWER

DISP_ANSWER:
    lcall hex2bcd       ; Convert result in x to BCD
    lcall Display       ; Display the result
    ljmp LOOP       ; Go check for more input

end
```