```

AVRASM ver. 2.2.7  C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\main.asm Thu Dec 05 18:18:29 2019

C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\main.asm(24): Including file  'C:/Program Files (x86)\Atmel\Studio\7.0\Packs\atmel\ATmega_DFP\1.3.300\avrasm\inc\m324adef.inc'
C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\main.asm(469): warning: Register r14 already defined by the .DEF directive
C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\main.asm(470): warning: Register r15 already defined by the .DEF directive
C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\main.asm(623): Including file  'C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\lcd_dog_asm_driver_m324a.inc'
C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\main.asm(24): Including file  'C:/Program Files (x86)\Atmel\Studio\7.0\Packs\atmel\ATmega_DFP\1.3.300\avrasm\inc\m324adef.inc'
C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\main.asm(623): Including file  'C:\Users\Seyi Olajuyi\Documents\Atmel Studio\7.0\ppg_IV_fsm\ppg_IV_fsm\lcd_dog_asm_driver_m324a.inc'


                                 ;***************************************************************************
                                 ;*
                                 ;* Title:   Simplified Table Driven FSM
                                 ;* Author:          Ken Short
                                 ;* Version:         2.0
                                 ;* Last updated:    11/09/15
                                 ;* Target:          ATmega16
                                 ;*
                                 ;* DESCRIPTION
                                 ;* This is a simplified version of the table driven FSM. It handles  only 255
                                 ;* or less input symbols.
                                 ;*
                                 ;* A sample table is included for a simple FSM. This table can be modified to
                                 ;* handle any FSM by equating the input symbols to byte values starting at
                                 ;* $00 and entering the appropriate next state and task subroutine  names.
                                 ;*
                                 ;*
                                 ;* VERSION HISTORY
                                 ;* 1.0 Original version
                                 ;* 2.0 Subroutines moved to end of file
                                 ;****************************************************************** *********
                                 .list

                                 .dseg  ;The variable below are in SRAM
000100                           burst_count_setting_bcd:        .byte 3; setting unpacked BCD ;THIS HAS THREE BTYE allocated to the variable name
000103                           burst_count:                    .byte 1; pulses left to generated in burst
000104                           keyvalue:                       .byte 1; stores the keyvalue into a variable
000105                           make_pulse:                         .byte 1;
000106                           is_burst_zero:                  .byte 1; Used to check if burst count is equal to zero. 1 means burst count is equal to zero
000107                           input:                          .byte 1; input


                                 ;burst_count_setting_bcd is right most digit and
                                 ; (burst_count_setting_bcd + 2) is the left most digit

                                 .cseg
                                 reset:
                                 .org RESET              ;reset interrupt vector
000000 c004                          rjmp start          ;program starts here at reset
                                 .org INT0addr           ;INT0 interrupt vector
000002 c0f7                          rjmp keypress_ISR
                                 .org INT1addr
000004 c11a                         rjmp pb_press_ISR

                                 start:
000005 ef0f                          ldi r16, LOW(RAMEND)    ;initialize SP to point to top of stack
000006 bf0d                          out SPL, r16
000007 e008                          ldi r16, HIGH(RAMEND)
000008 bf0e                          out SPH, r16

```

```
000009 e00f                          ldi r16, (1 << ISC00) | (1 << ISC01) | (1 << ISC10) | (1 << ISC 11)
00000a 9300 0069                     sts EICRA, r16
00000c e003                         ldi r16, $03         ; Enable interrupt request at INTO & INT1
00000d bb0d                          out EIMSK, r16

00000e ef0f                          ldi r16, $ff      ; load r16 with all 1s.
00000f b904                          out DDRB, r16     ; set portB = output

000010 e003                         ldi r16, $03         ; Set pin 0 & pin 1 to output, everyother pin is an input
000011 b90a                         out DDRD, r16

000012 9a0e                         sbi DDRA, 6          ;Set Pin 6 on PORTA (Buzzer)

000013 9a0f                         sbi DDRA, 7          ; Set pin 7 on PORTA to output (OUTPUT)

000014 9a2c                          sbi portB, 4      ; set /SS of DOG LCD = 1 (Deselected)

000015 d15a                          rcall init_lcd_dog        ; init display, using SPI serial inte rface
000016 d049                         rcall clr_dsp_buffs       ; clear all three buffer lines
000017 d176                         rcall update_lcd_dog      ; update the display

000018 e0d1                         ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
000019 e0c8                          ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
                                                               ; (dsp_buff_1 for now).

                                    ;put FSM in initial state
00001a e38a                          ldi pstatel, LOW(display)
00001b e090                          ldi pstateh, HIGH(display)

00001c 9478                         sei                  ;set global interrupt enable

                                 variable_reset:
                                    ; RESET THE VARIABLES WITH ZERO
00001d e010                         ldi r17, $00
```

```
00001e 9310 0102                    sts burst_count_setting_bcd + 2, r17
000020 9310 0101                    sts burst_count_setting_bcd + 1, r17
000022 9310 0100                    sts burst_count_setting_bcd + 0, r17

000024 9310 0105                    sts make_pulse, r17
000026 9310 0106                    sts is_burst_zero, r17

000028 9310 0103                    sts burst_count, r17

00002a 9310 0104                    sts keyvalue, r17


                                 test:
00002c 9100 0105                     lds r16, make_pulse
00002e ff00                         sbrs r16, 0             ; Skip the rjmp instruction if the make_pulse flag is set
00002f cffc                          rjmp test

000030 9100 0106                    lds r16, is_burst_zero
000032 3001                         cpi r16, 1
000033 f419                         brne gen_1_pulse

000034 940e 00b3                    call generate_a_pulse
000036 cff5                         rjmp test

                                 gen_1_pulse:
000037 940e 00a8                    call pulse_generator
000039 cff2                         rjmp test



                                 ;****************************************************************** *********
                                 ;*
                                 ;* "fsm" - Simplified Table Driven Finite State Machine
                                 ;*
```

```
;* Description:
;* This table driven FSM can handle 255 or fewer input symbols.
;*
;* Author:              Ken Short
;* Version:             2.0
;* Last updated:        11/09/15
;* Target:              ATmega16
;* Number of words:
;* Number of cycles:
;* Low regs modified:   r16, r18, r20, r21, r31, and r31
;* High registers used:
;*
;* Parameters:          present state in r25:r24 prior to call
;*                      input symbol in r16 prior to call
;*
;* Notes:
;*
;****************************************************************** *********

.def pstatel = r24 ;low byte of present state address
.def pstateh = r25 ;high byte of present state address

;input symbols for example finite state machine
.equ number = $00   ;input symbols equated to numerical values ;
.equ enter = $01
.equ clear = $02
.equ pushb = $03
                ;additional symbols would go here
.equ eol = $FF  ;end of list (subtable) do not change

;state table for example finite state machine
;each row consists of input symbol, next state address, task
;subroutine address

state_table:
```

```

00003a 0000
00003b 003a
00003c 0068                      display: .dw number,    display,    display_the_value
00003d 0001
00003e 0043
00003f 00cb                               .dw enter,     burst,      convert_to_Binary
000040 00ff
000041 003a
000042 00a2                               .dw eol,       display,    buzz

000043 0003
000044 0043
000045 00b8                      burst:   .dw pushb,     burst,      update_flags
000046 0002
000047 003a
000048 00c7                               .dw clear,     display,    clear_flags
000049 00ff
00004a 0043
00004b 00a2                               .dw eol,       burst,      buzz


                                 fsm:
                                 ;load Z with a byte pointer to the subtable corresponding to the
                                 ;present state
00004c 2fe8                          mov ZL, pstatel ;load Z pointer with pstate address * 2
00004d 0fee                          add ZL, ZL ;since Z will be used as a byte pointer with the lpm  instr.
00004e 2ff9                          mov ZH, pstateh
00004f 1fff                          adc ZH, ZH

                                 ;search subtable rows for input symbol match
                                 search:
000050 9124                          lpm r18, Z ;get symbol from state table
000051 1720                          cp r18, r16 ;compare table entry with input symbol
000052 f021                          breq match
```

```

                                 ;check input symbol against eol
                                 check_eol:
000053 3f2f                          cpi r18, eol ;compare low byte of table entry with eol
000054 f011                          breq match

                                 nomatch:
000055 9636                          adiw ZL, $06 ;adjust Z to point to next row of state table
000056 cff9                          rjmp search ;continue searching

                                 ;a match on input value to row input value has been found
                                 ;the next word in this row is the next state address
                                 ;the word following that is the task subroutine's address
                                 match:
                                    ;make preseent state equal to next state value in row
                                    ;this accomplishes the stat transition
000057 9632                          adiw ZL, $02 ;point to low byte of state address
000058 9185                          lpm pstatel, Z+; ;copy next state addr. from table to preseent  stat
000059 9195                          lpm pstateh, Z+

                                    ;execute the subroutine that accomplihes the task associated
                                    ;with the transition
00005a 9145                          lpm r20, Z+ ;get subroutine address from state table
00005b 9154                          lpm r21, Z ;and put it in Z pointer
00005c 2fe4                          mov ZL, r20
00005d 2ff5                          mov ZH, r21
00005e 9509                          icall ;Z pointer is now used as a word pointer
00005f 9508                          ret


                                 ;************************
                                 ;NAME:      clr_dsp_buffs
                                 ;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
                                 ;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
                                 ;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
                                 ;RETURNS:   nothing.
                                 ;MODIFIES:  r25,r26, Z-ptr
                                 ;CALLS:     none
                                 ;CALLED BY: main application and diagnostics
                                 ;****************************************************************** **
                                 clr_dsp_buffs:
000060 e390                           ldi R25, 48            ; load total length of both buffer.
000061 e2a0                           ldi R26, ' '               ; load blank/space into R26.
000062 e0f1                           ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
000063 e0e8                           ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.

                                     ;set DDRAM address to 1st position of first line.
                                 store_bytes:
000064 93a1                           st  Z+, R26       ; store ' ' into 1st/next buffer byte and
                                                        ; auto inc ptr to next location.
000065 959a                           dec  R25          ;
000066 f7e9                           brne store_bytes  ; cont until r25=0, all bytes written.
000067 9508                           ret


                                 ;**********************************************
                                 ;SUBROUTINE FOR DISPLAYING THE INPUT TO LCD
                                 ;**********************************************
                                 display_the_value:
000068 e0d1                         ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
000069 e0c8                          ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
                                                               ; (dsp_buff_1 for now).

00006a e60e                         ldi r16, 'n'
00006b 9309                         st Y+, r16
00006c e200                         ldi r16, ' '
00006d 9309                         st Y+, r16
00006e e30d                         ldi r16, '='
00006f 9309                         st Y+, r16
000070 e200                         ldi r16, ' '
```

```
000071 9309                         st Y+, r16

000072 e310                         ldi r17, $30                             ; Load $30 into r16
                                    ; store the ascii representation of the digit in the buffer
000073 9100 0102                    lds r16, (burst_count_setting_bcd + 2)           ; Store the leftmost keyvalue into r16

000075 2b01                         or r16, r17                              ; Adds $30 to the keyvalue, which turn the keyvalue into ASCII
000076 9309                         st Y+, r16                                       ; Put the value into the display buffer

000077 9100 0101                    lds r16, (burst_count_setting_bcd + 1)           ;
000079 2b01                         or r16, r17                              ; Adds $30 to the keyvalue, which turn the keyvalue into ASCII
00007a 9309                         st Y+, r16

00007b 9100 0100                    lds r16, (burst_count_setting_bcd + 0)           ; Store the rightmost keyvalue into r16
00007d 2b01                         or r16, r17                              ; Adds $30 to the keyvalue, which turn the keyvalue into ASCII
00007e 9309                         st Y+, r16                                       ; Put the value into the display buffer

00007f 940e 018e                    call update_lcd_dog                              ; update the display
000081 9508                         ret

                                 ;***********************************************
                                 ;SUBROUTINE FOR STORING THE VALUE INTO THE Variable
                                 ;**********************************************
                                 store_value:
                                    ;r18 is the value read by the input
000082 9120 0104                    lds r18, keyvalue
000084 9100 0101                    lds r16, burst_count_setting_bcd + 1     ; Load r16 with the middle digit
000086 9300 0102                    sts burst_count_setting_bcd + 2, r16     ; Put the middle digit into the
```

```
  leftmost digit

000088 9100 0100                    lds r16, burst_count_setting_bcd + 0     ; Load r16 with the Rightmost digit
00008a 9300 0101                    sts burst_count_setting_bcd + 1, r16     ; Put the rightmost digit into the
  middle digit

00008c 9320 0100                    sts burst_count_setting_bcd + 0, r18     ; Store the new number into the
  rightmost digit
00008e 9508                         ret

                                 ;**********************************
                                 ;SUBROUTINE FOR RETRIEVING INPUT(PART 2)
                                 ;*********************************
                                 get_key_value:
00008f b129                         in r18, PIND             ; Store the Input into r18
000090 7f20                         andi r18, $F0            ; Clear the low nibble of r18
000091 9522                         swap r18                 ; Swap the nibble
000092 940e 0095                    call keycode2keyvalue    ; Convert the input into HEXVALUES (NOT ASCII)
000094 9508                         ret

                                 ;********************************
                                 ;SUBROUTINE FOR LOOKUP TABLE
                                 ;******************************
                                 keycode2keyvalue:
                                 lookup:
000095 e0f2                         ldi ZH, high (keytable * 2)      ;set Z to point to start of table
000096 e6e6                         ldi ZL, low (keytable * 2)
000097 e000                         ldi r16, $00                     ;add offset to Z pointer
000098 0fe2                         add ZL, r18                      ;originally r18
000099 0ff0                         add ZH, r16
00009a 9124                         lpm r18, Z
00009b 9508                         ret

                                 ;**************************
                                 ;SUBROUTINE FOR DELAY
                                 ;************************
                                 var_delay: ;delay for ATmega324 @ 1MHz = r16 * 0.1 ms
                                 outer_loop:
00009c e210                         ldi r17, 32
                                 inner_loop:
00009d 951a                         dec r17
00009e f7f1                         brne inner_loop
00009f 950a                         dec r16
0000a0 f7d9                         brne outer_loop
0000a1 9508                         ret


                                 ;**********************************
                                 ;SUBROUTINE FOR BUZZER
                                 ;*********************************
                                 buzz:
0000a2 9a16                         sbi PORTA, 6
0000a3 ef0f                         ldi r16 , 255   ; For delay
0000a4 940e 009c                    call var_delay
0000a6 9816                         cbi PORTA, 6
0000a7 9508                         ret


                                 ;**********************************************
                                 ;SUBROUTINE FOR PULSE GENERATOR
                                 ;**********************************************
                                 pulse_generator:
0000a8 9a17                         sbi PORTA, 7                         ; set bit for pulse
0000a9 dff2                         rcall var_delay
0000aa e00a                         ldi r16, 10                          ; pulse width delay
0000ab 9817                         cbi PORTA, 7                         ; clear bit for pulse
0000ac dfef                         rcall var_delay
0000ad e00a                         ldi r16, 10                          ; time between pulses delay
0000ae 953a                         dec r19                              ; decrement the binary value
0000af f7c1                         brne pulse_generator
```

```

0000b0 940e 00c7                    call clear_flags

0000b2 9508                         ret


                                 ;**********************************************
                                 ;SUBROUTINE FOR GENERATING A PULSES
                                 ;**********************************************
                                 generate_a_pulse:
0000b3 e00a                         ldi r16, 10                          ; pulse width
0000b4 9a17                         sbi PORTA, 7                         ; set bit for pulse
0000b5 dfe6                         rcall var_delay
0000b6 9817                         cbi PORTA, 7                         ; clear bit for pulse
0000b7 9508                         ret


                                 ;**********************************************
                                 ;SUBROUTINE FOR ASSIGNING FLAGS
                                 ;**********************************************
                                 update_flags:
0000b8 e001                         ldi r16, 1                   ; Set the make_pulse flag
0000b9 9300 0105                    sts make_pulse, r16

0000bb 9100 0103                    lds r16, burst_count
0000bd 3000                         cpi r16, $00
0000be f021                         breq burst_is_zero

0000bf e000                         ldi r16, 0
0000c0 9300 0106                    sts is_burst_zero, r16

                                 please_go_here:
0000c2 9508                         ret

                                 burst_is_zero:
```

```
0000c3 e001                         ldi r16, 1
0000c4 9300 0106                    sts is_burst_zero, r16
0000c6 cffb                         rjmp please_go_here


                                 ;**********************************************
                                 ;SUBROUTINE FOR CLEARING FLAGS
                                 ;**********************************************
                                 clear_flags:
0000c7 e000                         ldi r16, 0
0000c8 9300 0105                    sts make_pulse, r16                 ; Reset the make_pluse to zero
0000ca 9508                         ret


                                 ;**********************************************
                                 ;SUBROUTINE FOR CONVERTING UNPACKED BCD TO BINARY
                                 ;**********************************************
                                 convert_to_Binary:
0000cb 9100 0100                    lds r16, burst_count_setting_bcd        ; Retrieve the value store in the
  FIRST byte of burst_count_setting_bcd   and store it in r16
0000cd 9110 0101                    lds r17, burst_count_setting_bcd + 1    ; Retrieve the value store in the
  SECOND byte of burst_count_setting_bcd and store it in r17
0000cf 9120 0102                    lds r18, burst_count_setting_bcd + 2    ; Retrieve the value store in the
  THIRD byte of burst_count_setting_bcd and store it in r18

0000d1 9512                         swap r17                                ; Swap the nibble in r17

0000d2 2b01                         or r16, r17                             ; Or r16 & r17, Combine the two
  contents of two registers into one register (r16)
0000d3 702f                         andi r18, $0F                           ; AND r18 & $0F, clear the high
  nibble of r18
0000d4 2f12                         mov r17, r18                            ; Move the content of r18 into r17
0000d5 e020                         ldi r18, $00                            ; Load r18 with zero, this will be
  useful when we are trying to convert
                                                                            ; Packed BCD into a 16-bit      R16
```

```
                            0x0a    byte{registers}@R16bianry value
                              ;This converts the Packed BCD into the 16-bit binary
0000d6 940e 00ee                 call BCD2bin16

0000d8 2d3e                      mov r19, r14                        ; Moves the low byte of the 16-bit binary value into r17
0000d9 9330 0103                 sts burst_count, r19                ; Store the value of r17 into burst_count_bin
0000db 9508                      ret


                              ;****************************************************************** *********
                              ;*
                              ;* "BCD2bin16" - BCD to 16-Bit Binary Conversion
                              ;*
                              ;* This subroutine converts a 5-digit packed BCD number represented  by
                              ;* 3 bytes (fBCD2:fBCD1:fBCD0) to a 16-bit number (tbinH:tbinL).
                              ;* MSD of the 5-digit number must be placed in the lowermost nibble  of fBCD2.
                              ;*
                              ;* Let "abcde" denote the 5-digit number. The conversion is done by
                              ;* computing the formula: 10(10(10(10a+b)+c)+d)+e.
                              ;* The subroutine "mul10a"/"mul10b" does the multiply-and-add opera tion
                              ;* which is repeated four times during the computation.
                              ;*
                              ;* Number of words  :30
                              ;* Number of cycles     :108
                              ;* Low registers used   :4 (copyL,copyH,mp10L/tbinL,mp10H/tbinH)
                              ;* High registers used   :4 (fBCD0,fBCD1,fBCD2,adder)
                              ;*
                              ;****************************************************************** *********

                              ;***** "mul10a"/"mul10b" Subroutine Register Variables

                              .def    copyL    =r12        ;temporary register
                              .def    copyH    =r13        ;temporary register
```

```
                                 .def   mp10L   =r14         ;Low byte of number to be multiplied by 10
                                 .def   mp10H   =r15         ;High byte of number to be multiplied by 10
                                 .def   adder   =r19         ;value to add after multiplication

                                 ;***** Code

                                 mul10a:     ;***** multiplies "mp10H:mp10L" with 10 and adds "adder" high nibble

0000dc 9532                         swap     adder
                                 mul10b:     ;***** multiplies "mp10H:mp10L" with 10 and adds "adder" low nibble
0000dd 2cce                         mov copyL,mp10L ;make copy
0000de 2cdf                         mov copyH,mp10H
0000df 0cee                         lsl mp10L       ;multiply original by 2
0000e0 1cff                         rol mp10H
0000e1 0ccc                         lsl copyL       ;multiply copy by 2
0000e2 1cdd                         rol copyH
0000e3 0ccc                         lsl copyL       ;multiply copy by 2 (4)
0000e4 1cdd                         rol copyH
0000e5 0ccc                         lsl copyL       ;multiply copy by 2 (8)
0000e6 1cdd                         rol copyH
0000e7 0cec                         add mp10L,copyL ;add copy to original
0000e8 1cfd                         adc mp10H,copyH
0000e9 703f                         andi     adder,0x0f  ;mask away upper nibble of adder
0000ea 0ee3                         add mp10L,adder ;add lower nibble of adder
0000eb f408                         brcc     m10_1       ;if carry not cleared
0000ec 94f3                         inc mp10H       ;    inc high byte
0000ed 9508                      m10_1: ret

                                 ;***** Main Routine Register Variables

                                 .def   tbinL   =r14         ;Low byte of binary result (same as mp10L)
                                 .def   tbinH   =r15         ;High byte of binary result (same as mp10H)
                                 .def   fBCD0   =r16         ;BCD value digits 1 and 0
                                 .def   fBCD1   =r17         ;BCD value digits 2 and 3
                                 .def   fBCD2   =r18         ;BCD value digit 5

                                 ;***** Code

                                 BCD2bin16:
0000ee 702f                         andi    fBCD2,0x0f  ;mask away upper nibble of fBCD2
0000ef 24ff                         clr mp10H
0000f0 2ee2                         mov mp10L,fBCD2 ;mp10H:mp10L = a
0000f1 2f31                         mov adder,fBCD1
0000f2 dfe9                         rcall   mul10a      ;mp10H:mp10L = 10a+b
0000f3 2f31                         mov adder,fBCD1
0000f4 dfe8                         rcall   mul10b      ;mp10H:mp10L = 10(10a+b)+c
0000f5 2f30                         mov adder,fBCD0
0000f6 dfe5                         rcall   mul10a      ;mp10H:mp10L = 10(10(10a+b)+c)+d
0000f7 2f30                         mov adder,fBCD0
0000f8 dfe4                         rcall   mul10b      ;mp10H:mp10L = 10(10(10(10a+b)+c)+d)+e
0000f9 9508                         ret


                                 ;****************************************************************** *********
                                 ;*
                                 ;* "keypress_ISR" - Check Interrupts at INT0
                                 ;*
                                 ;* Description: Get the keyvalue if the key is pressed, the keyvalu e is stored if the key is a number
                                 ;*
                                 ;* Author:                  Seyi Olajuyi & Bassel El Amine
                                 ;* Version:
                                 ;* Last updated:            11/21/19
                                 ;* Target:                  ATmega324A
                                 ;* Number of words:
                                 ;* Number of cycles:        N/A
                                 ;* Low registers modified:  none
                                 ;* High registers modified: none
                                 ;*
                                 ;* Parameters:
```

```
                                 ;* Notes:
                                 ;*
                                 ;****************************************************************** *********

                                     ;INT0 interrupt service routine
                                 keypress_ISR:
0000fa 932f                         push r18
0000fb 930f                          push r16            ;save r16
0000fc b70f                         in r16, SREG        ;save SREG
0000fd 930f                          push r16

0000fe e001                         ldi r16, (1 <<INTF0)
0000ff bb0c                         out EIFR, r16

000100 df8e                         rcall get_key_value
000101 302a                         cpi r18, $0A
000102 f068                         brlo skip_line_1

000103 f099                         breq input_clear

000104 302c                         cpi r18, $0C
000105 f0a9                         breq input_enter

000106 ef0f                         ldi r16, $FF
000107 9300 0107                    sts input, r16

                                 restore_values_1:
000109 940e 004c                    call fsm

00010b 910f                          pop r16             ;restore SREG
00010c bf0f                          out SREG,r16
00010d 910f                         pop r16             ;restore r16
00010e 912f                         pop r18             ;restore r18

00010f 9518                          reti                ;return from interrupt
```

```

                                 skip_line_1:
000110 9320 0104                    sts keyvalue, r18                       ; if key value is a number

000112 e000                         ldi r16, $00                            ; input is assign as a number
000113 9300 0107                    sts input, r16

000115 df6c                         rcall store_value
000116 cff2                         rjmp restore_values_1

                                 input_clear:
000117 e002                         ldi r16, $02
000118 9300 0107                    sts input, r16
00011a cfee                         rjmp restore_values_1

                                 input_enter:
00011b e001                         ldi r16, $01
00011c 9300 0107                    sts input, r16
00011e cfea                         rjmp restore_values_1



                                 ;****************************************************************** *********
                                 ;*
                                 ;* "pb_press_ISR" - Check Interrupts at INT1
                                 ;*
                                 ;* Description: Checks if the push button is pressed
                                 ;*
                                 ;* Author:                  Ken Short
                                 ;* Version:
                                 ;* Last updated:            11/21/19
                                 ;* Target:                  ATmega324A
                                 ;* Number of words:
                                 ;* Number of cycles:        16
                                 ;* Low registers modified:  none
                                 ;* High registers modified: none
                                 ;*
                                 ;* Parameters:  Uses PORTB register to hold the count and drive LED s
                                 ;* connected to that port.
                                 ;*
                                 ;* Notes:
                                 ;*
                                 ;****************************************************************** *********

                                     ;INT1 interrupt service routine
                                 pb_press_ISR:
00011f 930f                         push r16             ;save r16
000120 b70f                         in r16, SREG         ;save SREG
000121 930f                          push r16

                                 wait_for_bounce_1:
000122 994b                         sbic PIND, 3
000123 cffe                         rjmp wait_for_bounce_1
000124 e604                         ldi r16, 100
000125 df76                         rcall var_delay
000126 994b                         sbic PIND, 3
000127 cffa                         rjmp wait_for_bounce_1

000128 e002                         ldi r16, (1 <<INT1)
000129 bb0c                         out EIFR, r16

00012a e003                         ldi r16 , $03                        ; Set polling_for_button
00012b 9300 0107                    sts input, r16       ; Use to find out if the button was pressed

                                 restore_value_2:
00012d 940e 004c                    call fsm
00012f 910f                          pop r16              ;restore SREG
000130 bf0f                          out SREG,r16
000131 910f                         pop r16              ;restore r16

```

```
000132 9518                          reti                ;return from interrupt




000133 0201
000134 0f03
000135 0504
000136 0e06
000137 0807
000138 0d09                      keytable: .db $01, $02, $03, $0F, $04, $05, $06, $0E, $07, $08, $09 , $0D
000139 000a
00013a 0c0b                             .db $0A, $00, $0B, $0C


                                 .list



RESOURCE USE INFORMATION
------------------------

Notice:
The register and instruction counts are symbol table hit counts,
and hence implicitly used resources are not counted, eg, the
'lpm' instruction without operands implicitly uses r0 and z,
none of which are counted.

x,y,z are separate entities in the symbol table and are
counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

"ATmega324A" register use summary:
x  :   0 y  :   7 z  :  10 r0 :   0 r1 :   0 r2 :   0 r3 :   0 r4 :   0
```

```
r5 :   0 r6 :   0 r7 :   0 r8 :   0 r9 :   0 r10:   0 r11:   0 r12:   5
r13:   5 r14:   6 r15:   5 r16: 121 r17:  21 r18:  21 r19:  10 r20:  10
r21:   2 r22:   2 r23:   2 r24:   7 r25:   5 r26:   2 r27:   0 r28:   2
r29:   2 r30:  12 r31:  10
Registers used: 21 out of 35 (60.0%)

"ATmega324A" instruction use summary:
.lds  :   0 .sts  :   0 adc   :   2 add   :   5 adiw  :   2 and   :   0
andi  :   4 asr   :   0 bclr  :   0 bld   :   0 brbc  :   0 brbs  :   0
brcc  :   1 brcs  :   0 break :   0 breq  :   5 brge  :   0 brhc  :   0
brhs  :   0 brid  :   0 brie  :   0 brlo  :   1 brlt  :   0 brmi  :   0
brne  :  11 brpl  :   0 brsh  :   0 brtc  :   0 brts  :   0 brvc  :   0
brvs  :   0 bset  :   0 bst   :   0 call  :   9 cbi   :   6 cbr   :   0
clc   :   0 clh   :   0 cli   :   0 cln   :   0 clr   :   1 cls   :   0
clt   :   0 clv   :   0 clz   :   0 com   :   0 cp    :   1 cpc   :   0
cpi   :   5 cpse  :   0 dec   :  10 eor   :   0 fmul  :   0 fmuls :   0
fmulsu:   0 icall :   1 ijmp  :   0 in    :  12 inc   :   1 jmp   :   0
ld    :   3 ldd   :   0 ldi   :  66 lds   :  12 lpm   :   9 lsl   :   4
lsr   :   0 mov   :  13 movw  :   0 mul   :   0 muls  :   0 mulsu :   0
neg   :   0 nop   :   2 or    :   4 ori   :   0 out   :  12 pop   :  11
push  :  11 rcall :  48 ret   :  23 reti  :   2 rjmp  :  15 rol   :   4
ror   :   0 sbc   :   0 sbci  :   0 sbi   :  12 sbic  :   2 sbis  :   0
sbiw  :   0 sbr   :   0 sbrc  :   0 sbrs  :   3 sec   :   0 seh   :   0
sei   :   1 sen   :   0 ser   :   0 ses   :   0 set   :   0 sev   :   0
sez   :   0 sleep :   0 spm   :   0 st    :   8 std   :   0 sts   :  22
sub   :   0 subi  :   0 swap  :   3 tst   :   0 wdr   :   0
Instructions used: 40 out of 113 (35.4%)

"ATmega324A" memory use summary [bytes]:
Segment   Begin    End      Code   Data   Used    Size   Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x000366    814     52    866   32768   2.6%
[.dseg] 0x000100 0x000138      0     56     56    2048   2.7%
[.eseg] 0x000000 0x000000      0      0      0    1024   0.0%

```

```
Assembly complete, 0 errors, 2 warnings

```