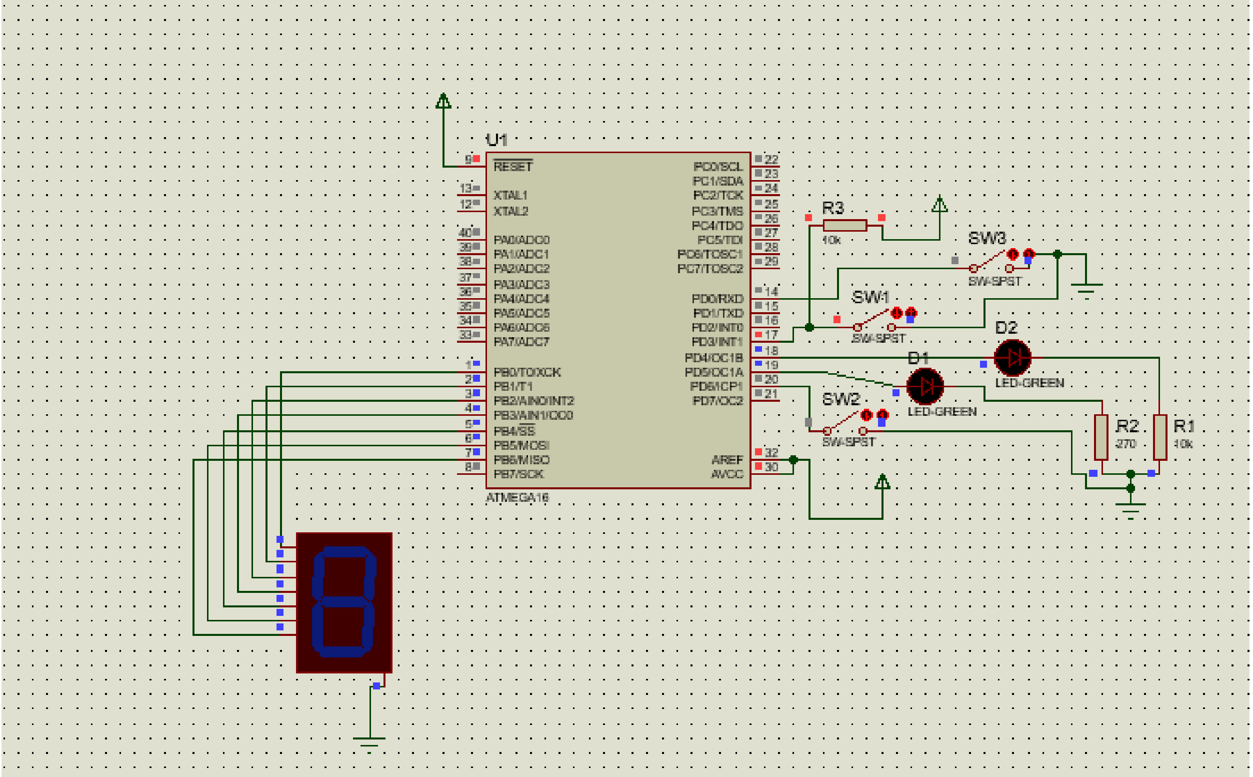
Yasaman Mirmohammad   9431022

گزارش تمرین شماره ی ۵ درس ریزپردازنده:

۱-

الف-میخواهیم با توجه به شکل، با یکبار فشرده شدن کلید sw1 و فعال شدن وقفه خارجیINT1 ، LED1 روشن شود و با فشردن همین کلید برای بار دوم، این LED خاموش شود.



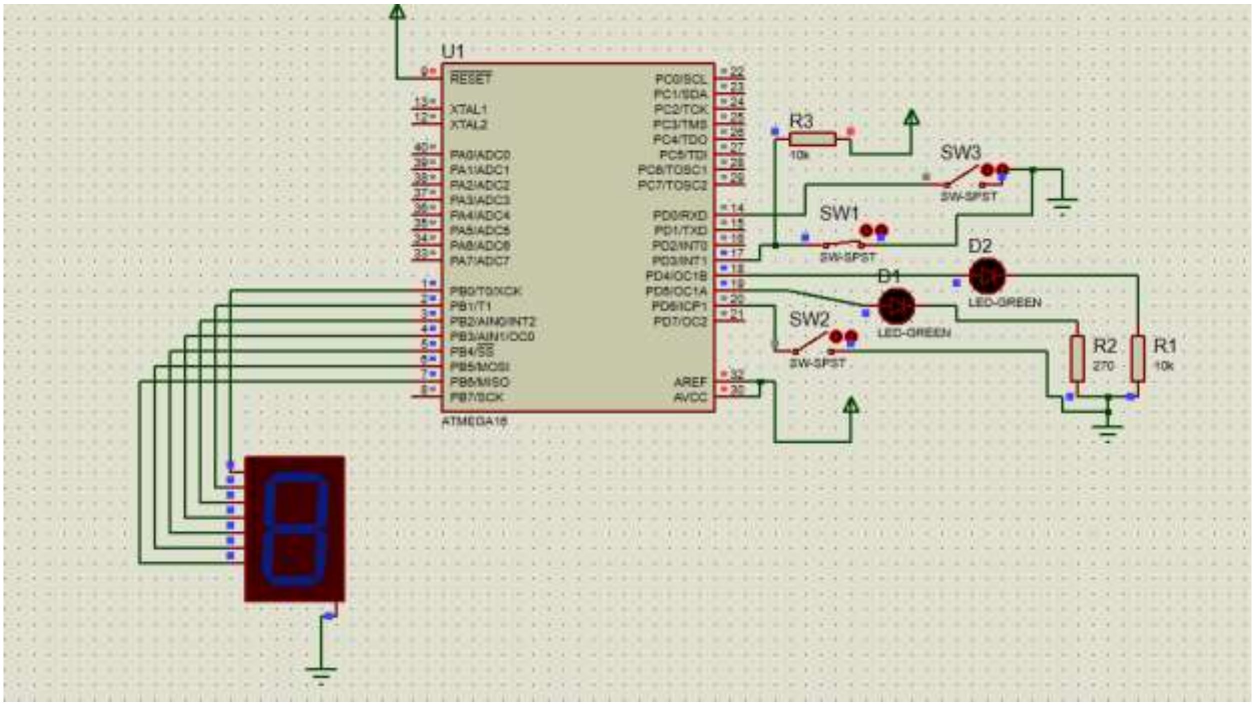
توضیحات:

مقادیررجیستر r17, r16 را لود کرده و یک زیرروال برای چشمک زدن دیود در نظر میگیریم.

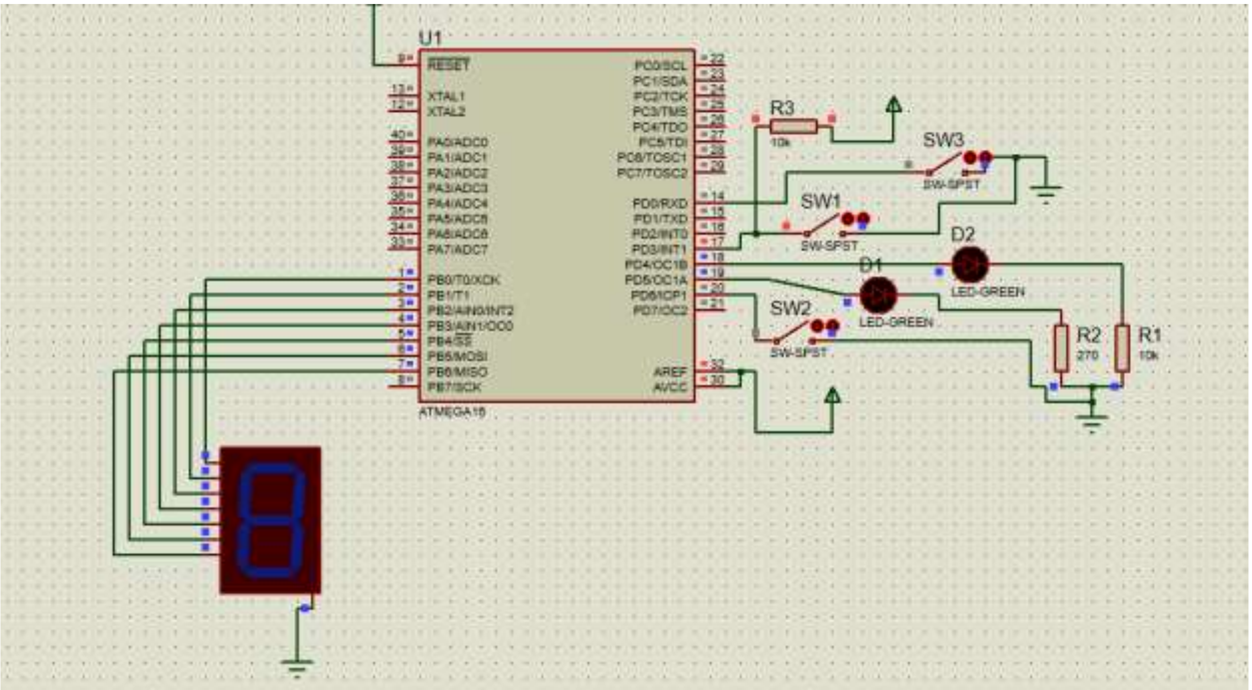
ابتدا پرش به زیرروال مربوط به چشمک زدن LED را درون آدرس مربوط به اینتراپت اکسترنال ۱ قرار میدهیم.

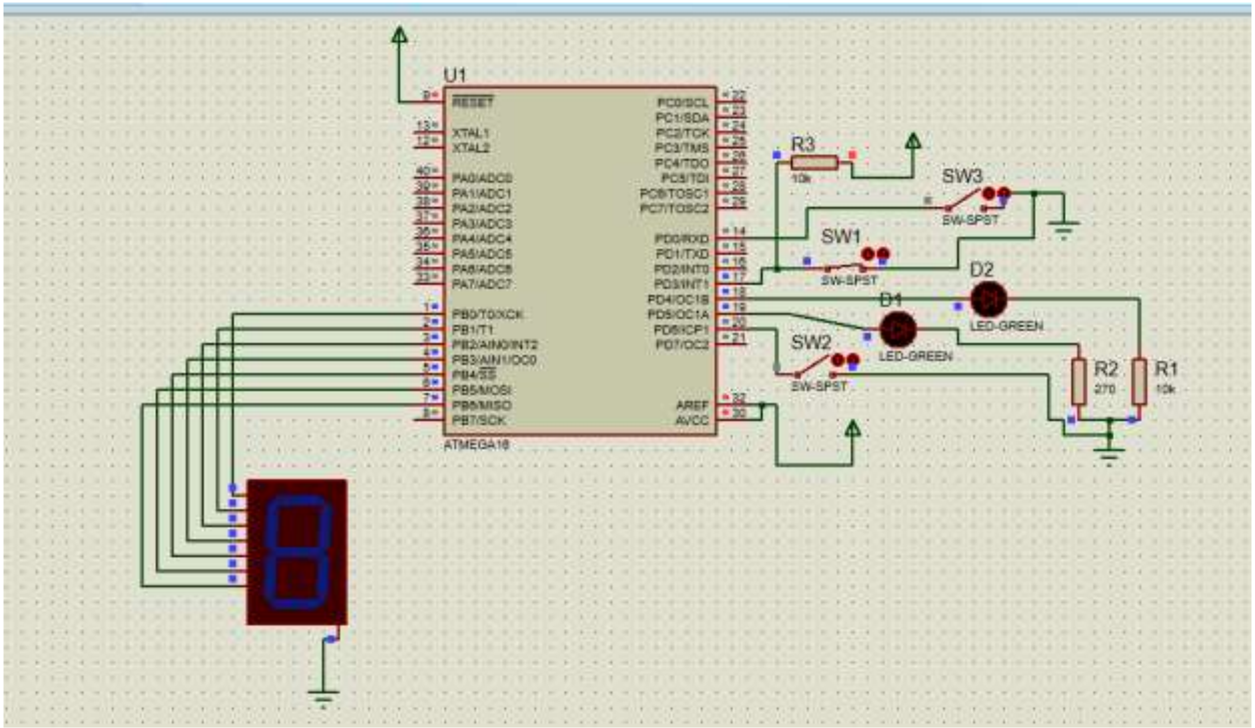
پس از تعیین ورودی/خروجی پایه های میکروکنترلر و ست کردن اینتراپت، بار هر بار صدا زده شدن زیرروال مربوط به چشمک زدن LED، یک رجیستر را complement میکنیم(۰ ها را ۱ و ۱ ها را ۰ میکنیم) و به پورت D میدهیم.

در نتیجه با هر بار صدازده شدن اینتراپت، ال ای دی به ترتیب روشن و خاموش شده و چشمک میزند. در حالت falling edge و Rising edgeهر بار فشردن کلید LED را خاموش و روشن میکند و در حالت any logical change با هر بار رها کردن کلید LED هم تغییر وضعیت میدهد.


U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA16
R3
10k
SW3
SW-SPST
SW1
SW-SPST
D2
LED-GREEN
D1
LED-GREEN
SW2
SW-SPST
R2
270
R1
10k

;


U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA16
R3
10k
SW3
SW-SPST
SW1
SW-SPST
D2
LED-GREEN
D1
LED-GREEN
SW2
SW-SPST
R2
270
R1
10k



```
;====================================================================
; Main.asm file generated by New Project wizard
;
; Created:   Tue Apr 24 2018
; Processor: ATmega16
; Compiler:  AVRASM (Proteus)
;====================================================================
.include "m16def.inc"


.def temp = r16
.def blinky_boy = r17


jmp reset
.org INT1addr
jmp handle_blink
```

```
reset:
  ;; init stack
  ldi temp, low(RAMEND)
  out SPL, temp
  ldi temp, high(RAMEND)
  out SPH, temp

  ldi temp , (1<<PD5)
  out DDRD , temp

  ldi temp , (0<<PD5)
  mov blinky_boy , temp

  ldi temp, (1 << ISC11) | (0 << ISC10)
  out MCUCR, temp

  in temp, GICR
  ori temp, (1<<INT1)
  out GICR, temp

  sei

Loop:
rjmp Loop

handle_blink:
  com blinky_boy
  out PORTD , blinky_boy
  reti
```

ب-

برای راه اندازی صفحه کلید، در روال وقفهINT0،زیرروالی به نام keyfind را صدا بزنید که شماره کلید را محاسبه و آنرا در ثبات r0 برگرداند.

```
*******************************************************************************;
m8_LCD_4bit.asm      File:           ;
ATmega8 driver for LCD in 4-bit mode (HD44780)         Title:      ;
AVR assembler/AVR Studio    Assembler:  ;
۱/۰       Version:    ;
April 5th, 2004      Created:    ;
ATmega8       Target:     ;
*******************************************************************************;


:Some notes on the hardware ;
ATmega8 (clock frequency doesn't matter, tested with 1 MHz to 8 MHz);
PORTA.1 -> LCD RS (register select) ;
PORTA.2 -> LCD RW (read/write) ;
PORTA.3 -> LCd E (Enable) ;
PORTA.4 ... PORTA.7 -> LCD data.4 ... data.7 ;
.the other LCd data lines can be left open or tied to ground ;




۱ =LCD_RS      equ.
۲ =           LCD_RW      equ.
۳ =   LCD_E      equ.

r16 =    temp       def.
temp2   = r24       def.
argument for calling subroutines;              argument= r17       def.
```

```
return value from subroutines;                r18 =   return       def.


org 0.

rjmp reset



org $002 ; INT0addr is the address of EXT_INT0.

jmp handle_pb0

org $004 ; INT1addr is the address of EXT_INT1.

jmp handle_pb1




:reset

temp, low(RAMEND)         ldi

SPL, temp        out

temp, high(RAMEND)         ldi

SPH, temp        out



LCD after power-up: ("*" means black bar);

|****************|;

|                         |;


LCD_init      rcall



:LCD now;

(cursor, blinking :&) |                      &|;
```

```
|                   |;

LCD_wait      rcall
;write 'A' to the LCD char data RAM    argument, 'A'         ldi
LCD_putchar      rcall

|                 &A|;
|                   |;

LCD_wait      rcall
now let the cursor go to line 0, col 0 (address 0);argument, 0x80         ldi
for setting a cursor address, bit 7 of the commands has to be set;  LCD_command      rcall

(cursor and A are at the same position!) |                    A|;
|                   |;

LCD_wait      rcall
now read from address 0;     LCD_getchar      rcall

 | (cursor is also incremented after read operations!!!)                  A&|;
|                   |;

save the return value (the character we just read!);              return     push

LCD_delay      rcall
restore the character;          argument      pop
and print it again;     LCD_putchar      rcall

 | (A has been read from position 0 and has then been written to the next pos.)                AA&|;
|                    |;

rcall D_INIT
rcall C_INIT
ldi temp, 0b00110000   Input/Output state;



   :loop






       rjmp loop


;used for init (we need some 8-bit commands to switch to 4-bit mode!)          lcd_command8:
we need to set the high nibble of DDRA while leaving;              temp, DDRA        in
.the other bits untouched. Using temp for that;
set high nibble in temp;      temp, 0b11110000      sbr
write value to DDRA again;               DDRA, temp      out
then get the port value;                temp, PortA        in
and clear the data bits;      temp, 0b11110000      cbr
then clear the low nibble of the argument; argument, 0b00001111      cbr
so that no control line bits are overwritten;
then set the data bits (from the argument) in the;                temp, argument       or
Port value;
.and write the port value;               PortA, temp      out
now strobe E;              PortA, LCD_E      sbi
nop
nop
```

```
nop
PortA, LCD_E        cbi
get DDRA to make the data lines input again;                  temp, DDRA          in
clear data line direction bits;        temp, 0b11110000        cbr
and write to DDRA;                    DDRA, temp        out
ret

:lcd_putchar
save the argmuent (it's destroyed in between);                       argument     push
get data direction bits;                   temp, DDRA          in
set the data lines to output;        temp, 0b11110000        sbr
write value to DDRA;                    DDRA, temp        out
then get the data from PORTA;                   temp, PortA          in
clear ALL LCD lines (data and control!);        temp, 0b11111110        cbr
we have to write the high nibble of our argument first; argument, 0b00001111        cbr
so mask off the low nibble;
now set the argument bits in the Port value;                         temp, argument          or
and write the port value;                   PortA, temp        out
now take RS high for LCD char data register access;               PortA, LCD_RS        sbi
strobe Enable;                  PortA, LCD_E        sbi
nop
nop
nop
PortA, LCD_E        cbi
...restore the argument, we need the low nibble now;                       argument       pop
clear the data bits of our port value;        temp, 0b11110000        cbr
we want to write the LOW nibble of the argument to;                       argument     swap
!the LCD data lines, which are the HIGH port nibble;
clear unused bits in argument; argument, 0b00001111        cbr
```

```
and set the required argument bits in the port value;              temp, argument      or
write data to port;            PortA, temp     out
again, set RS;          PortA, LCD_RS      sbi
strobe Enable;           PortA, LCD_E      sbi
nop
nop
nop
PortA, LCD_E      cbi
PortA, LCD_RS      cbi
temp, DDRA       in
data lines are input again;     temp, 0b11110000     cbr
DDRA, temp     out
ret

!;same as LCD_putchar, but with RS low  lcd_command:
argument    push
temp, DDRA       in
temp, 0b11110000     sbr
DDRA, temp     out
temp, PORTA       in
temp, 0b11111110     cbr
argument, 0b00001111     cbr
temp, argument      or

PORTA, temp     out
PORTA, LCD_E      sbi
nop
nop
nop
```

```
PORTA, LCD_E        cbi
argument      pop
temp, 0b11110000       cbr
argument    swap
argument, 0b00001111       cbr
temp, argument          or
PORTA, temp       out
PORTA, LCD_E        sbi
nop
nop
nop
PORTA, LCD_E        cbi
temp, DDRA          in
temp, 0b11110000       cbr
DDRA, temp       out
ret

:LCD_getchar
make sure the data lines are inputs;                temp, DDRA          in
so clear their DDR bits;        temp, 0b00001111      andi
DDRA, temp       out
we want to access the char data register, so RS high;            PORTA, LCD_RS        sbi
we also want to read from the LCD -> RW high;                    PORTA, LCD_RW        sbi
while E is high;               PORTA, LCD_E        sbi
nop
we need to fetch the HIGH nibble;                  temp, PinD          in
mask off the control line data;        temp, 0b11110000      andi
and copy the HIGH nibble to return;                return, temp      mov
now take E low again;               PORTA, LCD_E        cbi
```

```
wait a bit before strobing E again;                              nop
    nop
same as above, now we're reading the low nibble;         PORTA, LCD_E      sbi
nop
get the data;              temp, PinD       in
and again mask off the control line bits;      temp, 0b11110000     andi
temp HIGH nibble contains data LOW nibble! so swap;                 temp    swap
and combine with previously read high nibble;          return, temp       or
take all control lines low again;          PORTA, LCD_E       cbi
PORTA, LCD_RS       cbi
PORTA, LCD_RW       cbi
the character read from the LCD is now in return;                                   ret

;works just like LCD_getchar, but with RS low, return.7 is the busy flag    LCD_getaddr:
temp, DDRA        in
temp, 0b00001111     andi
DDRA, temp       out
PORTA, LCD_RS       cbi
PORTA, LCD_RW       sbi
PORTA, LCD_E       sbi
nop
temp, PinD        in
temp, 0b11110000     andi
return, temp     mov
PORTA, LCD_E       cbi
nop
nop
PORTA, LCD_E       sbi
nop
```

```
temp, PinD          in

temp, 0b11110000      andi

temp     swap

return, temp          or

PORTA, LCD_E         cbi

PORTA, LCD_RW         cbi

ret


;read address and busy flag until busy flag cleared                                        LCD_wait:

LCD_getaddr       rcall

return, 0x80      andi

LCD_wait      brne

ret




:LCD_delay

r2          clr

:LCD_delay_outer

r3          clr

:LCD_delay_inner

r3        dec

LCD_delay_inner      brne

r2        dec

LCD_delay_outer      brne

ret


:LCD_init


control lines are output, rest is input;         temp, 0b00001110          ldi
```

```
DDRA, temp      out


first, we'll tell the LCD that we want to use it;                    LCD_delay      rcall
.in 4-bit mode;             argument, 0x20        ldi
LCD is still in 8-BIT MODE while writing this ;                      LCD_command8      rcall
!!!command


LCD_wait      rcall
!NOW: 2 lines, 5*7 font, 4-BIT MODE;             argument, 0x28        ldi
;              LCD_command      rcall


LCD_wait      rcall
now proceed as usual: Display on, cursor on, blinking;             argument, 0x0F        ldi
LCD_command      rcall


LCD_wait      rcall
clear display, cursor -> home;             argument, 0x01        ldi
LCD_command      rcall


LCD_wait      rcall
auto-inc cursor;             argument, 0x06        ldi
LCD_command      rcall
ret


:handle_pb0


rcall keyfind


mov temp, r0
```

```
'CPI temp, '1

BRNE next_2

rjmp seg_1


:next_2


'CPI temp, '2

BRNE next_3

rjmp seg_2


:next_3


'CPI temp, '3

BRNE next_4

rjmp seg_3


:next_4


'CPI temp, '4

BRNE next_5

rjmp seg_4


:next_5


'CPI temp, '5

BRNE next_6

rjmp seg_5
```

```
:next_6

'CPI temp, '6
BRNE next_7
rjmp seg_6

:next_7

'CPI temp, '7
BRNE next_8
rjmp seg_7

:next_8

'CPI temp, '8
BRNE next_9
rjmp seg_8

:next_9

'CPI temp, '9
BRNE end_seg
rjmp seg_9

:seg_1

ldi temp, 0b00000110
out PORTB, temp
```

```
rjmp end_seg


:seg_2
ldi temp, 0b01011011
out PORTB, temp
rjmp end_seg


:seg_3
ldi temp, 0b01001111
out PORTB, temp

rjmp end_seg


:seg_4

ldi temp, 0b01100110
out PORTB, temp

rjmp end_seg

:seg_5

ldi temp, 0b01101101
out PORTB, temp

rjmp end_seg

:seg_6
```

```
ldi temp, 0b01111101
out PORTB, temp

rjmp end_seg

:seg_7

ldi temp, 0b00000111
out PORTB, temp

rjmp end_seg

:seg_8

ldi temp, 0b01111111
out PORTB, temp

rjmp end_seg

:seg_9

ldi temp, 0b01101111
out PORTB, temp

rjmp end_seg
```

```
:end_seg

reti

:handle_pb1
in temp, PORTD
SBRC temp, 5
rjmp set_zero
rjmp set_one

:set_zero
andi temp, 0b11011111
rjmp end_pb1

:set_one
ori temp, 0b00100000
rjmp end_pb1

:end_pb1
out PORTD, temp
reti

:B_INIT
ldi temp, 0b11111111
out DDRB, temp

ldi temp, 0b00000000
out PORTB, temp
```

```
:C_INIT

ldi temp, 0b11110000 ;; (1« PORTC1 ) | (1 « PORTC2)
out DDRC,temp

ldi temp,0b00001111
out PORTC,temp

ret


:D_INIT

ldi temp, 0b00110000
out DDRD, temp
ldi temp, 0b01001111
out PORTD, temp

in temp, MCUCR
ldi temp2, 0b00001010
or temp, temp2
out MCUCR, temp

in temp, GICR
ldi temp2, 0b11000000
or temp, temp2
out GICR, temp

sei
```

```
ret

:keyfind

in temp, PINC

SBRS temp, 0
rjmp column_3
SBRS temp, 1
rjmp column_2
rjmp column_1

:column_1

in temp, PINC
ori temp, 0b00010000
out PINC, temp
in temp2, PINC
SBRC temp2, 2
rjmp A_1

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b00100000
out PINC, temp
in temp2, PINC
```

```
SBRC temp2, 2
rjmp B_1

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b01000000
out PINC, temp
in temp2, PINC
SBRC temp2, 2
rjmp C_1

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b10000000
out PINC, temp
in temp2, PINC
SBRC temp2, 2
rjmp D_1


:column_2

in temp, PINC
ori temp, 0b00010000
out PINC, temp
```

```
in temp2, PINC
SBRC temp2, 1
rjmp A_2

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b00100000
out PINC, temp
in temp2, PINC
SBRC temp2, 1
rjmp B_2

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b01000000
out PINC, temp
in temp2, PINC
SBRC temp2, 1
rjmp C_2

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b10000000
```

```
out PINC, temp
in temp2, PINC
SBRC temp2, 1
rjmp D_2


:column_3
in temp, PINC
ori temp, 0b00010000
out PINC, temp
in temp2, PINC
SBRC temp2, 0
rjmp A_3

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b00100000
out PINC, temp
in temp2, PINC
SBRC temp2, 0
rjmp B_3

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b01000000
```

```
out PINC, temp

in temp2, PINC

SBRC temp2, 0

rjmp C_3


ldi temp, 0b00001111

out PORTC, temp


in temp, PINC

ori temp, 0b10000000

out PINC, temp

in temp2, PINC

SBRC temp2, 0

rjmp D_3



:A_1

'ldi temp, '1

mov r0, temp

rjmp end_key


:A_2

'ldi temp, '2

mov r0, temp

rjmp end_key



:A_3

'ldi temp, '3
```

```
mov r0, temp
rjmp end_key

:B_1
'ldi temp, '4
mov r0, temp
rjmp end_key

:B_2
'ldi temp, '5
mov r0, temp
rjmp end_key

:B_3
'ldi temp, '6
mov r0, temp
rjmp end_key

:C_1
'ldi temp, '7
mov r0, temp
rjmp end_key

:C_2
'ldi temp, '8
mov r0, temp
rjmp end_key

:C_3
```

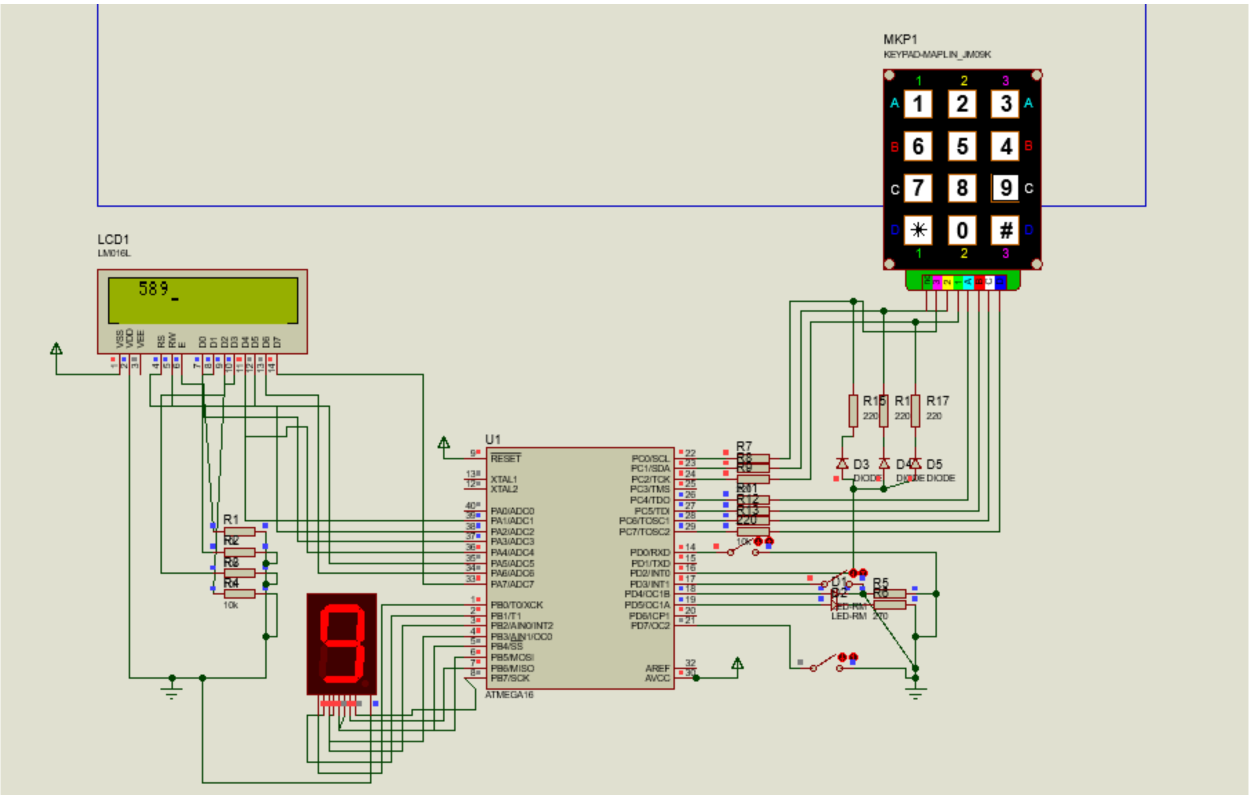
```
'ldi temp, '9
mov r0, temp
rjmp end_key

:D_1
rjmp end_key
:D_2
rjmp end_key
:D_3
rjmp end_key

:end_key

ldi temp, 0b00001111
out PORTC, temp

LCD_wait     rcall
write 'A' to the LCD char data RAM;     argument, r0     mov
LCD_putchar     rcall

ret
```

ج-

برنامه ای بنویسید که شماره کلید فشرده شده را بر روی seven segment نمایش دهد.

در زیرروال مربوط به keypad ابتدا ۴ سطر را به ترتیب یک کرده و ستون ها را میخوانیم . هر ستونی که یک شد چک میکنیم کدام سطر یک شده بوده است و در نتیجه سطر و ستون کلید فشرده و شده و نتیجتا عدد فشرده شده را پیدا میکنیم. سپس زیر روال هایی برای نشان دادن هر عدد روی ۷ سگمنت تعریف کردیم که مطابق جدول صدا زدن هر زیر روال عدد مربوطه روی آن نمایش داده بشود .

خروجی keypad را در یک رجیستر ذخیره کرده و از آن برای نمایش روی seven segment استفاده کرده ایم.[برای نمایش روی LCD مطابق کد اماده ای که در لینک بود، استفاده کردیم].



```
;***********************************************************************
;         File:     m8_LCD_4bit.asm
;    Title:         ATmega8 driver for LCD in 4-bit mode (HD44780)
; Assembler:        AVR assembler/AVR Studio
;   Version:        1.0
;   Created:        April 5th, 2004
;    Target:        ATmega8
;***********************************************************************
```

```
;ATmega8 (clock frequency doesn't matter, tested with 1 MHz to 8 MHz)
; PORTA.1 -> LCD RS (register select)
; PORTA.2 -> LCD RW (read/write)
; PORTA.3 -> LCd E (Enable)
; PORTA.4 ... PORTA.7 -> LCD data.4 ... data.7
; the other LCd data lines can be left open or tied to ground.



.equ    LCD_RS = 1
.equ    LCD_RW          = 2
.equ    LCD_E   = 3

.def    temp    = r16
.def    temp2   = r24
.def    argument= r17           ;argument for calling subroutines
.def    return  = r18           ;return value from subroutines

.org 0
rjmp reset


LCDTABLE: .db 3, 'A', 'L', 'I'

.org $002 ; INT0addr is the address of EXT_INT0
jmp handle_pb0
.org $004 ; INT1addr is the address of EXT_INT1
jmp handle_pb1
```

```
reset:
        ldi     temp, low(RAMEND)
        out     SPL, temp
        ldi     temp, high(RAMEND)
        out     SPH, temp


;LCD after power-up: ("*" means black bar)
;|****************|
;|                |

        rcall   LCD_init


;LCD now:
;|&               | (&: cursor, blinking)
;|                |

        rcall   LCD_wait
        ldi     argument, 'A'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar

;|A&              |
;|                |

        rcall   LCD_wait
        ldi     argument, 0x80;now let the cursor go to line 0, col 0 (address 0)
        rcall   LCD_command ;for setting a cursor address, bit 7 of the commands has to be set

;|A             | (cursor and A are at the same position!)
;|              |

        rcall   LCD_wait
        rcall   LCD_getchar     ;now read from address 0

;|A&            | (cursor is also incremented after read operations!!!)
;|              |

        push    return          ;save the return value (the character we just read!)

        rcall   LCD_delay
        pop     argument        ;restore the character
        rcall   LCD_putchar     ;and print it again

;|AA&           | (A has been read from position 0 and has then been written to the next pos.)
;|              |

         rcall D_INIT
         rcall C_INIT
         ;ldi temp, 0b00110000   Input/Output state


loop:
```

```
rjmp loop


print_from_memory:
    ldi    zl,low(LCDTABLE << 1)
    ldi    zh,high(LCDTABLE << 1)
    lpm    r20,z+ ;Load Number of Characters.


word_loop:
    lpm    r19,z+
    mov    r1, r19


    rcall LCD_wait
    movargument, r1     ;write 'A' to the LCD char data RAM
    rcall LCD_putchar


    SUBI R20, 1
    CPI R20, 0
    BREQ end_write
    RJMP word_loop


end_write:


ret




lcd_command8:            ;used for init (we need some 8-bit commands to switch to 4-bit mode!)
            in         temp, DDRA                ;we need to set the high nibble of DDRA while leaving
                                                 ;the other bits untouched. Using temp for that.
```

```
        sbr     temp, 0b11110000        ;set high nibble in temp
        out     DDRA, temp              ;write value to DDRA again
        in      temp, PortA             ;then get the port value
        cbr     temp, 0b11110000        ;and clear the data bits
        cbr     argument, 0b00001111 ;then clear the low nibble of the argument
                                        ;so that no control line bits are overwritten
        or      temp, argument                  ;then set the data bits (from the argument) in the
                                        ;Port value
        out     PortA, temp             ;and write the port value.
        sbi     PortA, LCD_E            ;now strobe E
        nop
        nop
        nop
        cbi     PortA, LCD_E
        in      temp, DDRA              ;get DDRA to make the data lines input again
        cbr     temp, 0b11110000        ;clear data line direction bits
        out     DDRA, temp              ;and write to DDRA
ret

lcd_putchar:
        push    argument                ;save the argmuent (it's destroyed in between)
        in      temp, DDRA              ;get data direction bits
        sbr     temp, 0b11110000        ;set the data lines to output
        out     DDRA, temp              ;write value to DDRA
        in      temp, PortA             ;then get the data from PORTA
        cbr     temp, 0b11111110        ;clear ALL LCD lines (data and control!)
        cbr     argument, 0b00001111 ;we have to write the high nibble of our argument first
                                        ;so mask off the low nibble
        or      temp, argument                  ;now set the argument bits in the Port value
        out     PortA, temp             ;and write the port value
        sbi     PortA, LCD_RS           ;now take RS high for LCD char data register access
        sbi     PortA, LCD_E            ;strobe Enable
        nop
        nop
        nop
        cbi     PortA, LCD_E
        pop     argument                ;restore the argument, we need the low nibble now...
        cbr     temp, 0b11110000        ;clear the data bits of our port value
        swap    argument                ;we want to write the LOW nibble of the argument to
                                        ;the LCD data lines, which are the HIGH port nibble!
        cbr     argument, 0b00001111 ;clear unused bits in argument
        or      temp, argument                  ;and set the required argument bits in the port value
        out     PortA, temp             ;write data to port
        sbi     PortA, LCD_RS           ;again, set RS
        sbi     PortA, LCD_E            ;strobe Enable
        nop
        nop
        nop
        cbi     PortA, LCD_E
        cbi     PortA, LCD_RS
        in      temp, DDRA
        cbr     temp, 0b11110000        ;data lines are input again
        out     DDRA, temp
ret

lcd_command:    ;same as LCD_putchar, but with RS low!
        push    argument
        in      temp, DDRA
        sbr     temp, 0b11110000
        out     DDRA, temp
        in      temp, PORTA
        cbr     temp, 0b11111110
        cbr     argument, 0b00001111
        or      temp, argument

        out     PORTA, temp
        sbi     PORTA, LCD_E
        nop
        nop
        nop
        cbi     PORTA, LCD_E
        pop     argument
        cbr     temp, 0b11110000
        swap    argument
        cbr     argument, 0b00001111
        or      temp, argument
        out     PORTA, temp
        sbi     PORTA, LCD_E
        nop
        nop
        nop
        cbi     PORTA, LCD_E
        in      temp, DDRA
        cbr     temp, 0b11110000
        out     DDRA, temp
ret
```

```
LCD_getchar:
          in      temp, DDRA              ;make sure the data lines are inputs
          andi    temp, 0b00001111        ;so clear their DDR bits
          out     DDRA, temp
          sbi     PORTA, LCD_RS           ;we want to access the char data register, so RS high
          sbi     PORTA, LCD_RW                     ;we also want to read from the LCD -> RW high
          sbi     PORTA, LCD_E            ;while E is high
          nop
          in      temp, PinD              ;we need to fetch the HIGH nibble
          andi    temp, 0b11110000        ;mask off the control line data
          mov     return, temp            ;and copy the HIGH nibble to return
          cbi     PORTA, LCD_E            ;now take E low again
          nop                             ;wait a bit before strobing E again
          nop
          sbi     PORTA, LCD_E            ;same as above, now we're reading the low nibble
          nop
          in      temp, PinD              ;get the data
          andi    temp, 0b11110000        ;and again mask off the control line bits
          swap    temp                    ;temp HIGH nibble contains data LOW nibble! so swap
          or      return, temp            ;and combine with previously read high nibble
          cbi     PORTA, LCD_E            ;take all control lines low again
          cbi     PORTA, LCD_RS
          cbi     PORTA, LCD_RW
ret                                       ;the character read from the LCD is now in return

LCD_getaddr:  ;works just like LCD_getchar, but with RS low, return.7 is the busy flag
          in      temp, DDRA
          andi    temp, 0b00001111
          out     DDRA, temp
```

```
            cbi     PORTA, LCD_RS
            sbi     PORTA, LCD_RW
            sbi     PORTA, LCD_E
            nop
            in      temp, PinD
            andi    temp, 0b11110000
            mov     return, temp
            cbi     PORTA, LCD_E
            nop
            nop
            sbi     PORTA, LCD_E
            nop
            in      temp, PinD
            andi    temp, 0b11110000
            swap    temp
            or      return, temp
            cbi     PORTA, LCD_E
            cbi     PORTA, LCD_RW
ret

LCD_wait:                               ;read address and busy flag until busy flag cleared
            rcall   LCD_getaddr
            andi    return, 0x80
            brne    LCD_wait
            ret



LCD_delay:
            clr     r2
```

```
          LCD_delay_outer:
          clr       r3
                    LCD_delay_inner:
                    dec       r3
                    brne      LCD_delay_inner
          dec       r2
          brne      LCD_delay_outer
ret


LCD_init:


          ldi       temp, 0b00001110      ;control lines are output, rest is input
          out       DDRA, temp

          rcall     LCD_delay             ;first, we'll tell the LCD that we want to use it
          ldi       argument, 0x20        ;in 4-bit mode.
          rcall     LCD_command8                    ;LCD is still in 8-BIT MODE while writing this
command!!!

          rcall     LCD_wait
          ldi       argument, 0x28        ;NOW: 2 lines, 5*7 font, 4-BIT MODE!
          rcall     LCD_command           ;

          rcall     LCD_wait
          ldi       argument, 0x0F        ;now proceed as usual: Display on, cursor on, blinking
          rcall     LCD_command

          rcall     LCD_wait
          ldi       argument, 0x01        ;clear display, cursor -> home
```

```
        rcall   LCD_command


        rcall   LCD_wait
        ldi     argument, 0x06          ;auto-inc cursor
        rcall   LCD_command
ret

handle_pb0:


rcall keyfind


mov temp, r0


CPI temp, '1'
BRNE next_2
rjmp seg_1

next_2:


CPI temp, '2'
BRNE next_3
rjmp seg_2

next_3:


CPI temp, '3'
BRNE next_4
rjmp seg_3
```

```
next_4:


CPI temp, '4'

BRNE next_5

rjmp seg_4


next_5:


CPI temp, '5'

BRNE next_6

rjmp seg_5


next_6:


CPI temp, '6'

BRNE next_7

rjmp seg_6


next_7:


CPI temp, '7'

BRNE next_8

rjmp seg_7


next_8:


CPI temp, '8'

BRNE next_9

rjmp seg_8
```

```
next_9:


CPI temp, '9'

BRNE end_seg

rjmp seg_9


seg_1:


ldi temp, 0b00000110

out PORTB, temp


rjmp end_seg


seg_2:

ldi temp, 0b01011011

out PORTB, temp

rjmp end_seg


seg_3:

ldi temp, 0b01001111

out PORTB, temp


rjmp end_seg


seg_4:


ldi temp, 0b01100110

out PORTB, temp
```

```
rjmp end_seg

seg_5:

ldi temp, 0b01101101
out PORTB, temp

rjmp end_seg

seg_6:

ldi temp, 0b01111101
out PORTB, temp

rjmp end_seg

seg_7:

ldi temp, 0b00000111
out PORTB, temp

rjmp end_seg

seg_8:

ldi temp, 0b01111111
out PORTB, temp
```

```
rjmp end_seg


seg_9:


ldi temp, 0b01101111
out PORTB, temp

rjmp end_seg





end_seg:


reti


handle_pb1:
   in temp, PORTD
   SBRC temp, 5
   rjmp set_zero
   rjmp set_one

   set_zero:
   andi temp, 0b11011111
   rjmp end_pb1

   set_one:
   ori temp, 0b00100000
   rjmp end_pb1
```

```
    end_pb1:
    out PORTD, temp
reti

B_INIT:
ldi temp, 0b11111111
out DDRB, temp

ldi temp, 0b00000000
out PORTB, temp

C_INIT:

ldi temp, 0b11110000 ;; (1« PORTC1 ) | (1 « PORTC2)
out DDRC,temp

 ldi temp,0b00001111
 out PORTC,temp

ret


D_INIT:

ldi temp, 0b00110000
out DDRD, temp
ldi temp, 0b01001111
out PORTD, temp
```

```
in temp, MCUCR
ldi temp2, 0b00001010
or temp, temp2
out MCUCR, temp

in temp, GICR
ldi temp2, 0b11000000
or temp, temp2
out GICR, temp

sei

ret

keyfind:

in temp, PINC

SBRS temp, 0
rjmp column_3
SBRS temp, 1
rjmp column_2
rjmp column_1

column_1:

in temp, PINC
ori temp, 0b00010000
```

```
out PINC, temp
in temp2, PINC
SBRC temp2, 2
rjmp A_1

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b00100000
out PINC, temp
in temp2, PINC
SBRC temp2, 2
rjmp B_1

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b01000000
out PINC, temp
in temp2, PINC
SBRC temp2, 2
rjmp C_1

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
```

```
ori temp, 0b10000000
out PINC, temp
in temp2, PINC
SBRC temp2, 2
rjmp D_1


column_2:

in temp, PINC
ori temp, 0b00010000
out PINC, temp
in temp2, PINC
SBRC temp2, 1
rjmp A_2

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b00100000
out PINC, temp
in temp2, PINC
SBRC temp2, 1
rjmp B_2

ldi temp, 0b00001111
out PORTC, temp
```

```
in temp, PINC
ori temp, 0b01000000
out PINC, temp
in temp2, PINC
SBRC temp2, 1
rjmp C_2

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b10000000
out PINC, temp
in temp2, PINC
SBRC temp2, 1
rjmp D_2



column_3:
in temp, PINC
ori temp, 0b00010000
out PINC, temp
in temp2, PINC
SBRC temp2, 0
rjmp A_3

ldi temp, 0b00001111
out PORTC, temp
```

```
in temp, PINC
ori temp, 0b00100000
out PINC, temp
in temp2, PINC
SBRC temp2, 0
rjmp B_3

ldi temp, 0b00001111
out PORTC, temp


in temp, PINC
ori temp, 0b01000000
out PINC, temp
in temp2, PINC
SBRC temp2, 0
rjmp C_3

ldi temp, 0b00001111
out PORTC, temp

in temp, PINC
ori temp, 0b10000000
out PINC, temp
in temp2, PINC
SBRC temp2, 0
rjmp D_3


A_1:
```

```
ldi temp, '1'
mov r0, temp
rjmp end_key

A_2:
ldi temp, '2'
mov r0, temp
rjmp end_key


A_3:
ldi temp, '3'
mov r0, temp
rjmp end_key

B_1:
ldi temp, '4'
mov r0, temp
rjmp end_key

B_2:
ldi temp, '5'
mov r0, temp
rjmp end_key

B_3:
ldi temp, '6'
mov r0, temp
rjmp end_key
```

```
C_1:
ldi temp, '7'
mov r0, temp
rjmp end_key

C_2:
ldi temp, '8'
mov r0, temp
rjmp end_key

C_3:
ldi temp, '9'
mov r0, temp
rjmp end_key

D_1:
rjmp end_key
D_2:
rjmp end_key
D_3:
rjmp end_key

end_key:

ldi temp, 0b00001111
out PORTC, temp
```

ret

-۲

الف-

Hello world را روی LCD نمایش بدهید.

ب-یک زیرروال به نامLCD که کاراکتر هایی را که در یک بلوک حافظه Flash ، به آدرس شروع LCDTABLE قرار دارند را نمایش بدهد.

توضیحات:

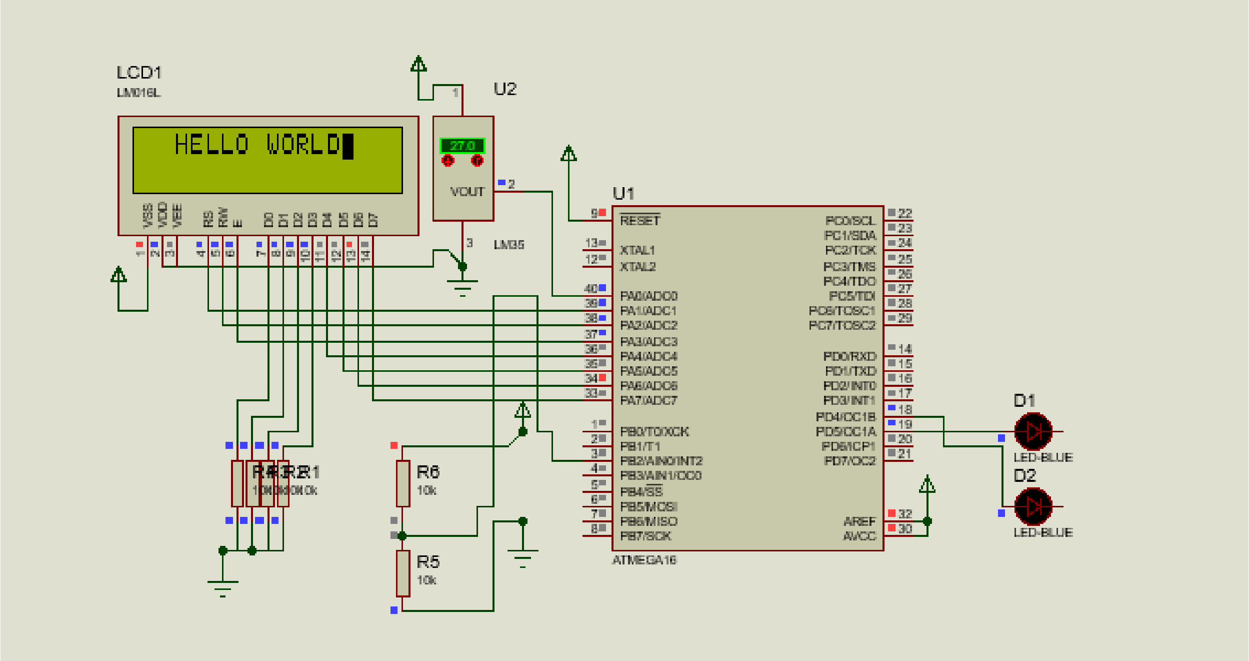
نمایش دادن عبارتHello World که طبق روند کدنویسی عادی انجام میشود و فقط لازم است نام

: ابتدا حروف کلمه ی اسم خود را در حافظه فلش ذخیره میکنیم

’LCDTABLE: .db 7, 'Y', 'A','S','A','M’,’A’,’N’

سپس آدرس زیرروال LCDTABLE را در رجیستر z ذخیره میکنیم و در رجیستر r20 تعداد کاراکترها را ذخیره میکنیم.

سپس در یک حلقه با تعداد تکرار رجیستر r20 یکی یکی از z خوانده و درr19 ریخته وr19 را بر روی LCD نمایش میدهیم.



;***

```
;           File:      m8_LCD_4bit.asm
;     Title:           ATmega8 driver for LCD in 4-bit mode (HD44780)
;  Assembler:    AVR assembler/AVR Studio
;    Version:       1.0
;    Created:       April 5th, 2004
;     Target:        ATmega8
;*******************************************************************************


; Some notes on the hardware:
;ATmega8 (clock frequency doesn't matter, tested with 1 MHz to 8 MHz)
; PORTA.1 -> LCD RS (register select)
; PORTA.2 -> LCD RW (read/write)
; PORTA.3 -> LCd E (Enable)
; PORTA.4 ... PORTA.7 -> LCD data.4 ... data.7
; the other LCd data lines can be left open or tied to ground.


;.include "c:\program files\atmel\avr studio\appnotes\m8def.inc"


.equ	LCD_RS	= 1
.equ	LCD_RW		= 2
.equ	LCD_E	= 3


.def	temp	= r16
.def	argument= r17		;argument for calling subroutines
.def	return	= r18		;return value from subroutines


.org 0
rjmp reset
```

```
reset:
        ldi     temp, low(RAMEND)
        out     SPL, temp
        ldi     temp, high(RAMEND)
        out     SPH, temp

;LCD after power-up: ("*" means black bar)
;|****************|
;|                |

        rcall   LCD_init

;LCD now:
;|&               | (&: cursor, blinking)
;|                |

        rcall   LCD_wait
        ldi     argument, 'A'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar

;|A&              |
;|                |

        rcall   LCD_wait
        ldi     argument, 0x80;now let the cursor go to line 0, col 0 (address 0)
        rcall   LCD_command ;for setting a cursor address, bit 7 of the commands has to be set

;|A               | (cursor and A are at the same position!)
;|                |
        rcall   LCD_wait
        rcall   LCD_getchar     ;now read from address 0

;|A&            | (cursor is also incremented after read operations!!!)
;|              |

        push    return          ;save the return value (the character we just read!)

        rcall   LCD_delay
        pop     argument        ;restore the character
        rcall   LCD_putchar     ;and print it again

;|AA&           | (A has been read from position 0 and has then been written to the next pos.)
;|              |
        rcall   LCD_wait
        ldi     argument, 'H'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar

rcall   LCD_wait
        ldi     argument, 'E'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar

rcall   LCD_wait
        ldi     argument, 'L'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar

rcall   LCD_wait
        ldi     argument, 'L'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar


rcall   LCD_wait
        ldi     argument, 'O'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar


rcall   LCD_wait
        ldi     argument, ' '   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar


rcall   LCD_wait
        ldi     argument, 'W'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar


rcall   LCD_wait
        ldi     argument, 'O'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar


rcall   LCD_wait
        ldi     argument, 'R'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar


rcall   LCD_wait
        ldi     argument, 'L'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar



        rcall   LCD_wait
        ldi     argument, 'D'   ;write 'A' to the LCD char data RAM
        rcall   LCD_putchar
loop:


        rjmp loop


lcd_command8:           ;used for init (we need some 8-bit commands to switch to 4-bit mode!)
        in      temp, DDRA              ;we need to set the high nibble of DDRA while leaving
                                        ;the other bits untouched. Using temp for that.
        sbr     temp, 0b11110000        ;set high nibble in temp
        out     DDRA, temp              ;write value to DDRA again
        in      temp, PORTA             ;then get the port value
        cbr     temp, 0b11110000        ;and clear the data bits
        cbr     argument, 0b00001111 ;then clear the low nibble of the argument
                                        ;so that no control line bits are overwritten
        or      temp, argument                  ;then set the data bits (from the argument) in the
                                        ;Port value
        out     PORTA, temp             ;and write the port value.
        sbi     PORTA, LCD_E            ;now strobe E
        nop
        nop
        nop
        cbi     PORTA, LCD_E
        in      temp, DDRA              ;get DDRA to make the data lines input again
        cbr     temp, 0b11110000        ;clear data line direction bits
        out     DDRA, temp              ;and write to DDRA
ret


lcd_putchar:
        push    argument                ;save the argmuent (it's destroyed in between)
```

```
in     temp, DDRA           ;get data direction bits
sbr    temp, 0b11110000     ;set the data lines to output
out    DDRA, temp           ;write value to DDRA
in     temp, PORTA          ;then get the data from PORTA
cbr    temp, 0b11111110     ;clear ALL LCD lines (data and control!)
cbr    argument, 0b00001111 ;we have to write the high nibble of our argument first
                            ;so mask off the low nibble
or     temp, argument               ;now set the argument bits in the Port value
out    PORTA, temp          ;and write the port value
sbi    PORTA, LCD_RS        ;now take RS high for LCD char data register access
sbi    PORTA, LCD_E         ;strobe Enable
nop
nop
nop
cbi    PORTA, LCD_E
pop    argument             ;restore the argument, we need the low nibble now...
cbr    temp, 0b11110000     ;clear the data bits of our port value
swap   argument             ;we want to write the LOW nibble of the argument to
                            ;the LCD data lines, which are the HIGH port nibble!
cbr    argument, 0b00001111 ;clear unused bits in argument
or     temp, argument               ;and set the required argument bits in the port value
out    PORTA, temp          ;write data to port
sbi    PORTA, LCD_RS        ;again, set RS
sbi    PORTA, LCD_E         ;strobe Enable
nop
nop
nop
cbi    PORTA, LCD_E
cbi    PORTA, LCD_RS
```

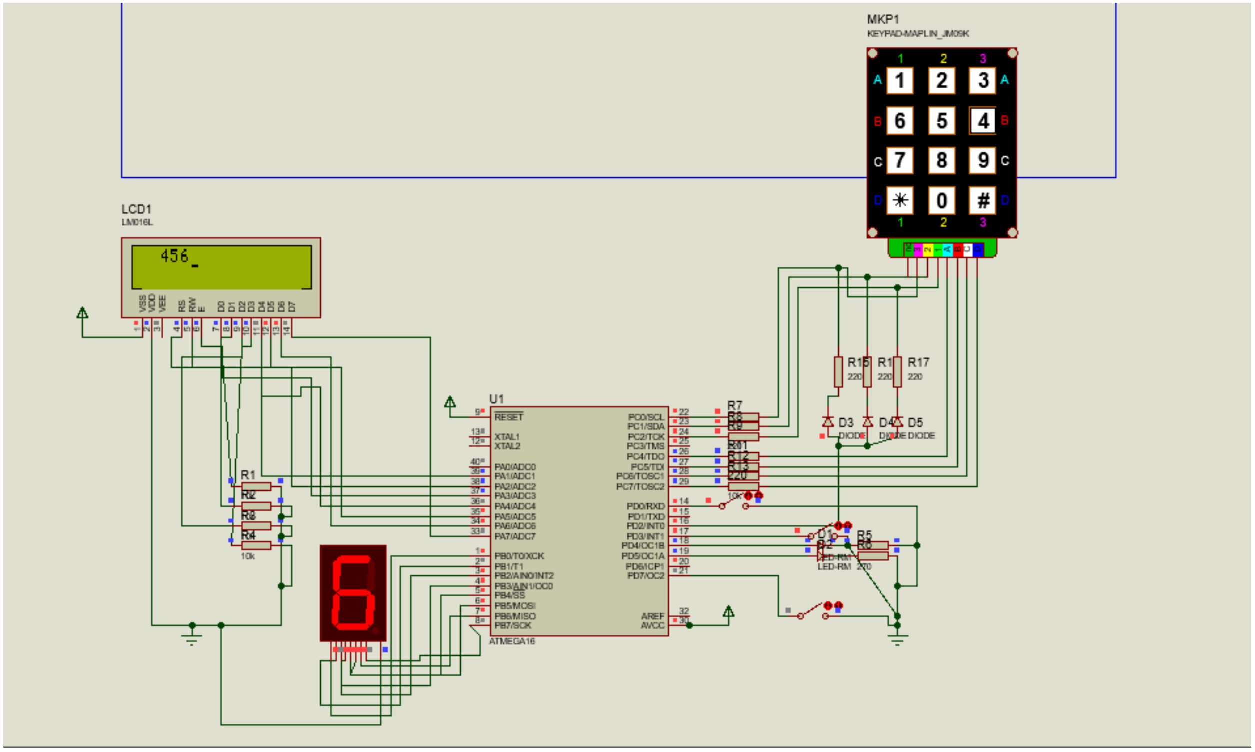
```
        in      temp, DDRA
        cbr     temp, 0b11110000        ;data lines are input again
        out     DDRA, temp
ret

lcd_command:  ;same as LCD_putchar, but with RS low!
        push    argument
        in      temp, DDRA
        sbr     temp, 0b11110000
        out     DDRA, temp
        in      temp, PORTA
        cbr     temp, 0b11111110
        cbr     argument, 0b00001111
        or      temp, argument

        out     PORTA, temp
        sbi     PORTA, LCD_E
        nop
        nop
        nop
        cbi     PORTA, LCD_E
        pop     argument
        cbr     temp, 0b11110000
        swap    argument
        cbr     argument, 0b00001111
        or      temp, argument
        out     PORTA, temp
        sbi     PORTA, LCD_E
        nop
        nop
        nop
        cbi     PORTA, LCD_E
        in      temp, DDRA
        cbr     temp, 0b11110000
        out     DDRA, temp
ret

LCD_getchar:
        in      temp, DDRA              ;make sure the data lines are inputs
        andi    temp, 0b00001111        ;so clear their DDR bits
        out     DDRA, temp
        sbi     PORTA, LCD_RS           ;we want to access the char data register, so RS high
        sbi     PORTA, LCD_RW                   ;we also want to read from the LCD -> RW high
        sbi     PORTA, LCD_E            ;while E is high
        nop
        in      temp, PinD              ;we need to fetch the HIGH nibble
        andi    temp, 0b11110000        ;mask off the control line data
        mov     return, temp            ;and copy the HIGH nibble to return
        cbi     PORTA, LCD_E            ;now take E low again
        nop                             ;wait a bit before strobing E again
        nop
        sbi     PORTA, LCD_E            ;same as above, now we're reading the low nibble
        nop
        in      temp, PinD              ;get the data
        andi    temp, 0b11110000        ;and again mask off the control line bits
        swap    temp                    ;temp HIGH nibble contains data LOW nibble! so swap
        or      return, temp            ;and combine with previously read high nibble
        cbi     PORTA, LCD_E            ;take all control lines low again
        cbi     PORTA, LCD_RS
        cbi     PORTA, LCD_RW
ret                                     ;the character read from the LCD is now in return

LCD_getaddr:    ;works just like LCD_getchar, but with RS low, return.7 is the busy flag
        in      temp, DDRA
        andi    temp, 0b00001111
        out     DDRA, temp
        cbi     PORTA, LCD_RS
        sbi     PORTA, LCD_RW
        sbi     PORTA, LCD_E
        nop
        in      temp, PinD
        andi    temp, 0b11110000
        mov     return, temp
        cbi     PORTA, LCD_E
        nop
        nop
        sbi     PORTA, LCD_E
        nop
        in      temp, PinD
        andi    temp, 0b11110000
        swap    temp
        or      return, temp
        cbi     PORTA, LCD_E
        cbi     PORTA, LCD_RW
ret

LCD_wait:                               ;read address and busy flag until busy flag cleared
```

```
        rcall   LCD_getaddr
        andi    return, 0x80
        brne    LCD_wait
        ret



LCD_delay:
        clr     r2
        LCD_delay_outer:
        clr     r3
                LCD_delay_inner:
                dec     r3
                brne    LCD_delay_inner
        dec     r2
        brne    LCD_delay_outer
ret

LCD_init:

        ldi     temp, 0b00001110        ;control lines are output, rest is input
        out     DDRA, temp

        rcall   LCD_delay               ;first, we'll tell the LCD that we want to use it
        ldi     argument, 0x20          ;in 4-bit mode.
        rcall   LCD_command8                    ;LCD is still in 8-BIT MODE while writing this
command!!!

        rcall   LCD_wait
        ldi     argument, 0x28          ;NOW: 2 lines, 5*7 font, 4-BIT MODE!
        rcall   LCD_command             ;

        rcall   LCD_wait
        ldi     argument, 0x0F          ;now proceed as usual: Display on, cursor on, blinking
        rcall   LCD_command

        rcall   LCD_wait
        ldi     argument, 0x01          ;clear display, cursor -> home
        rcall   LCD_command

        rcall   LCD_wait
        ldi     argument, 0x06          ;auto-inc cursor
        rcall   LCD_command
ret
```

ج-

اطلاعات خوانده شده و نمایش داده شده روی seven segmentدر سوال اول، روی ال سی دی هم چاپ شود.

(فقط به کد سوال ۱ قسمت ب، یک قسمت اضافه شده است.)



```
ldi temp, 0b00001111

out PORTC, temp


rcall       LCD_wait

            mov      argument, r0     ;write  to the LCD char data RAM

            rcall    LCD_putchar
```