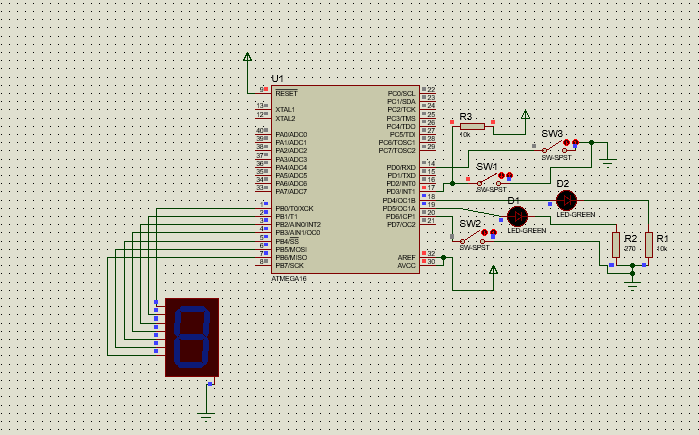
**گزارش تمرین شماره ی 5 درس ریزپردازنده:**

**1-**

**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.**



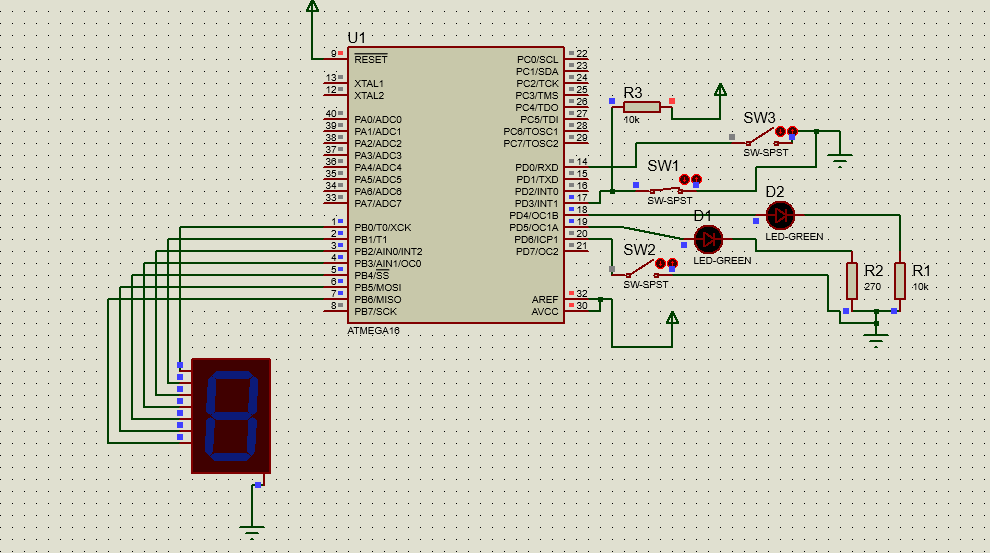
**توضیحات:**

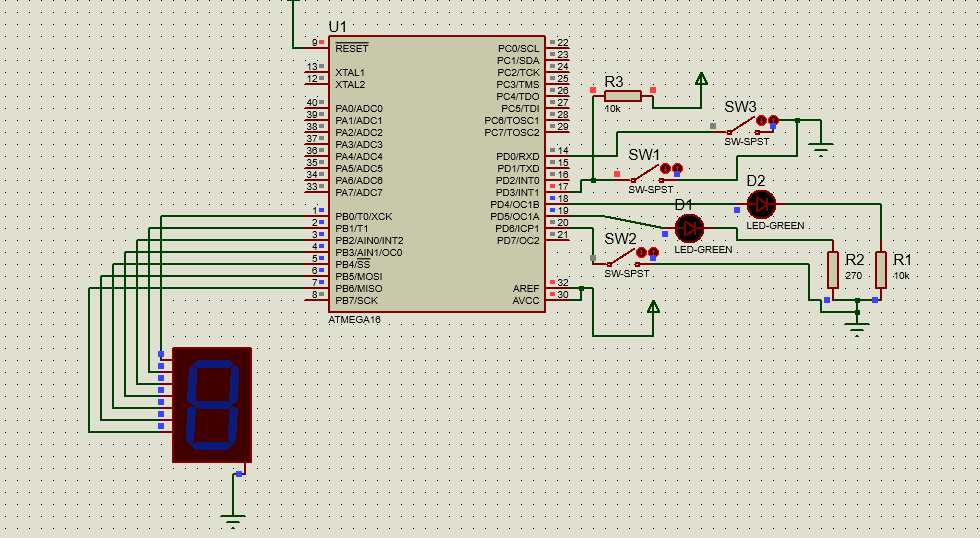
**مقادیررجیستر r17, r16 را لود کرده و یک زیرروال برای چشمک زدن دیود در نظر میگیریم.**

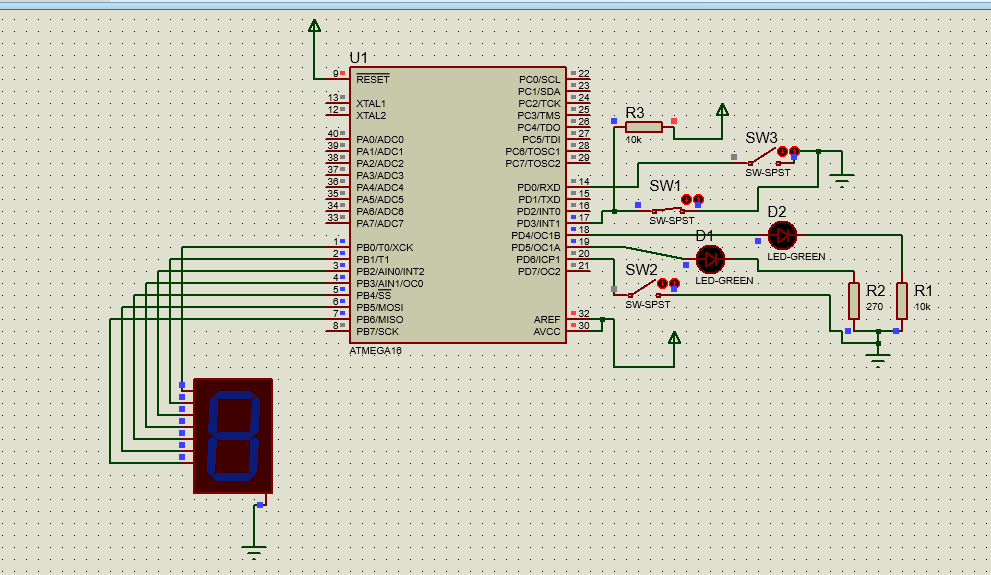
**ابتدا پرش به زیرروال مربوط به چشمک زدن LED را درون آدرس مربوط به اینتراپت اکسترنال ۱ قرار میدهیم.**

**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(0 ها را 1 و 1 ها را 0 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 هم LED تغییر وضعیت میدهد.**



**;** 



**;====================================================================**

**; 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.**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**; 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.**

**.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**

**.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**

**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**

**rcall LCD\_wait**

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

**rcall LCD\_putchar**

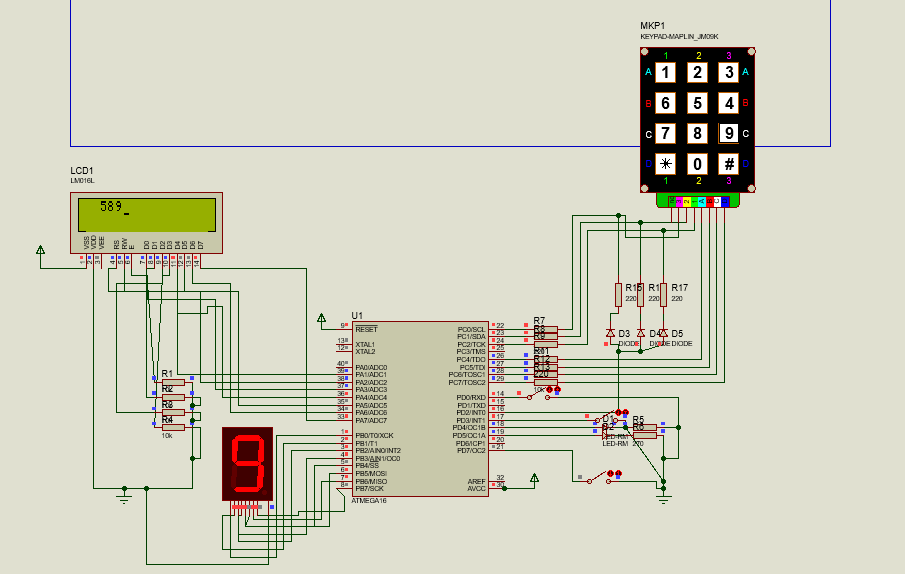
**ret**

**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 یشود .**

**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**

**mov argument, 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**

**2-**

**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:**

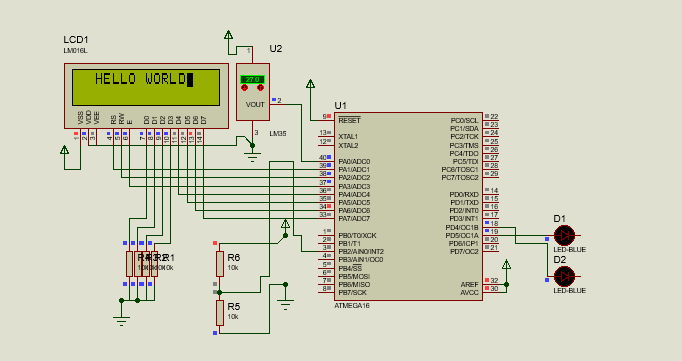
**نمایش دادن عبارت Hello World که طبق روند کدنویسی عادی انجام میشود و فقط لازم است نام**

**: 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**

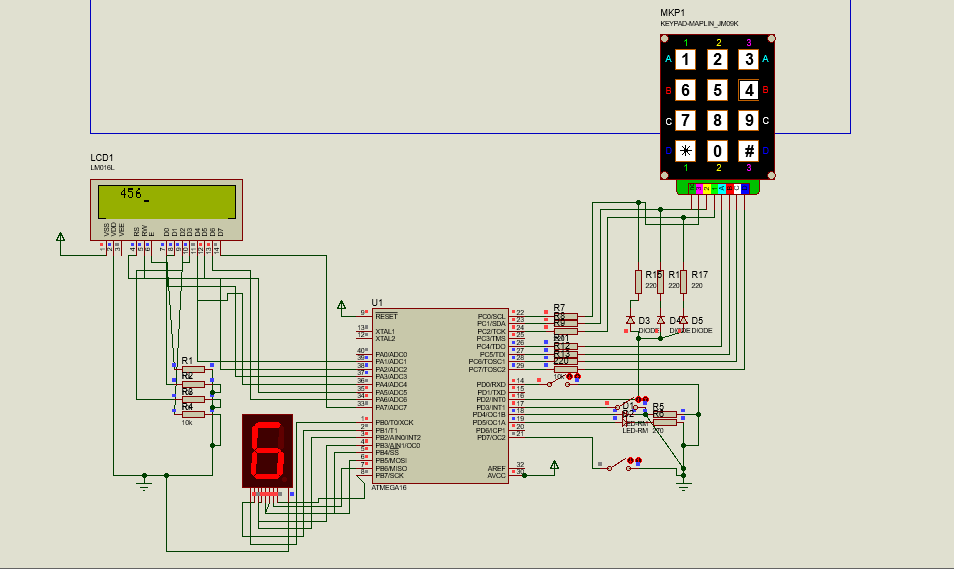
**ldi argument, 0x06 ;auto-inc cursor**

**rcall LCD\_command**

**ret**

**ج-**

**اطلاعات خوانده شده و نمایش داده شده رویseven segment در سوال اول، روی ال سی دی هم چاپ شود.**

**(فقط به کد سوال 1 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**