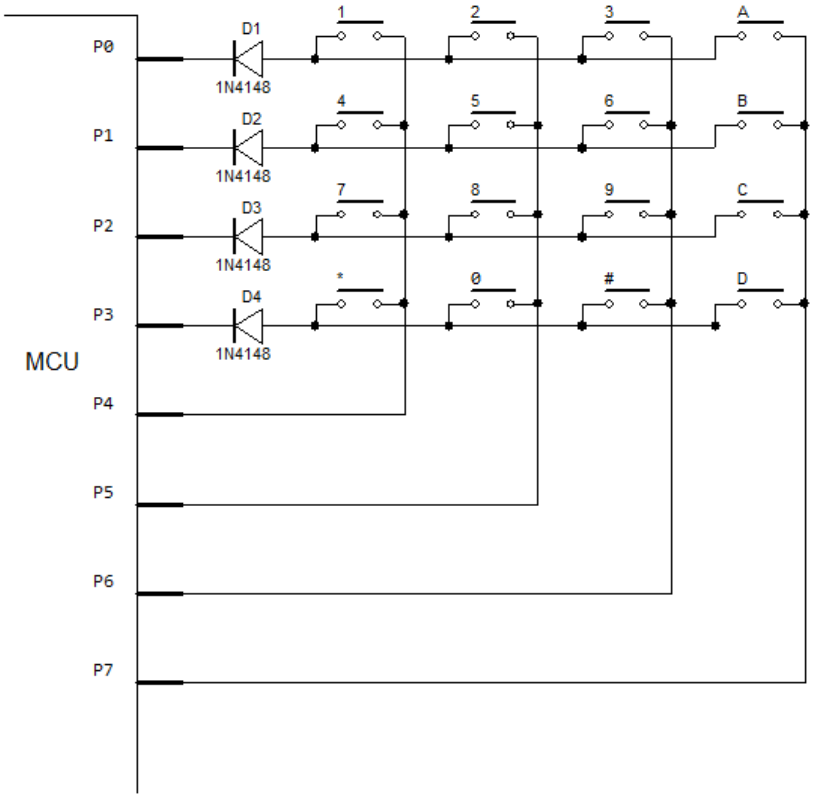
10. Πληκτρολόγιο matrix 4x4



Το πληκτρολόγιο matrix 4x4 αποτελείται από 16 πλήκτρα – διακόπτες τα οποία είναι συνδεδεμένα μεταξύ τους ανά 4 σε τέτοια διάταξη ώστε, με το ένα άκρο τους να σχηματίζουν 4 σειρές και με το άλλο 4 στήλες. Στο θεωρητικό κύκλωμα του σχήματος παρουσιάζεται η σύνδεση ενός πληκτρολογίου matrix 4x4 στο port ενός μικροελεγκτή. Οι 4 σειρές συνδέονται στη μισή θύρα του μικροελεγκτή P0 – P3, η οποία λειτουργεί ως έξοδος με αρχική κατάσταση 1111 και οι 4 στήλες στην άλλη μισή θύρα P4 – P7, η οποία λειτουργεί ως είσοδος με ενεργοποιημένες τις pull up αντιστάσεις σε κάθε pin. Οι έξοδοι P0 έως P3 περνούν μια – μια διαδοχικά σε λογικό 0 για ένα χρονικό διάστημα της τάξεως ms σαρώνοντας τις σειρές. Οι είσοδοι P4 – P7 ανιχνεύουν τη χαμηλή λογική σε περίπτωση που πατηθεί κάποιο πλήκτρο από τις στήλες. Ο συνδιασμός στήλης – σειράς καθορίζει το πλήκτρο που έχει πατηθεί. Αν για παράδειγμα πατηθεί το πλήκτρο 6, η είσοδος P6 θα δεχτεί λογικό 0 όταν η έξοδος P1 περάσει σε λογικό 0. Ο καταχωρητής κατάστασης της θύρας τη στιγμή εκείνη θα γίνει 10111101. Το byte αυτό αντιστοιχίζεται στη συνέχεια στο νούμερο 6 ή σε κάποια άλλη λειτουργία. Η αντιστοίχιση μπορεί να γίνει με μια εντολή **switch**.

Επειδή κατά το κλείσιμο και κατά το άνοιγμα της επαφής του διακόπτη δημιουργούνται παρασιτικοί παλμοί, εισάγονται κάποιες καθυστερήσεις από το λογισμικό, ώστε να αποφεύγονται λανθασμένες πληροφορίες. Οι δίοδοι D1 έως D4 είναι απαραίτητες για προστασία από βραχυκύκλωμα σε περίπτωση που πατηθούν ταυτόχρονα δύο πλήκτρα που βρίσκονται στην ίδια στήλη. Αν συμβεί αυτό η έξοδος που είναι σε λογικό 0 θα κλείσει κύκλωμα με την άλλη έξοδο που είναι σε λογικό 1. Η δίοδος τότε που βρίσκεται στη δεύτερη έξοδο πολώνεται ανάστροφα και προστατεύεται το κύκλωμα. Στη συνέχεια παρουσιάζεται ένα παράδειγμα κώδικα σε γλώσσα C που μπορεί να χρησιμοποιηθεί για αναγνώριση πληκτρολογίου:

```
#include <avr/io.h>
#include <util/delay.h>
uint8_t line;
uint8_t data;

void Keyb (void)                                    //Keyboard rutine
{
    line = 1;
    for (uint8_t i1 = 0; i1 < 4; i1++)
    {
        PORTA = ~line;
        line = line << 1;
        //_delay_ms(1);
        while (PINA < 0b11101111)
        {
            _delay_ms (5);
            switch (PINA)
            {
                case 0b11101110:                    //Button 1 pressed
                data = 1;
                while (PINA == 0b11101110);         //Button still pressed
                break;

                case 0b11101101:                    //Button 4 pressed
                data = 4;
                while (PINA == 0b11101101);
                break;

                case 0b11101011:                    //Button 7 pressed
                data = 7;
                while (PINA == 0b11101011);
                break;

                case 0b11100111:                    //Button * pressed
                data = 14;
                while (PINA == 0b11100111);
                break;

                case 0b11011110:                    //Button 2 pressed
                data = 2;
                while (PINA == 0b11011110);
                break;

                case 0b11011101:                    //Button 5 pressed
```

```
                data = 5;
                while (PINA == 0b11011101);
                break;

                case 0b11011011:                //Button 8 pressed
                data = 8;
                while (PINA == 0b11011011);
                break;

                case 0b11010111:                //Button 0 pressed
                data = 0;
                while (PINA == 0b11010111);
                break;

                case 0b10111110:                //Button 3 pressed
                data = 3;
                while (PINA == 0b10111110);
                break;

                case 0b10111101:                //Button 6 pressed
                data = 6;
                while (PINA == 0b10111101);
                break;

                case 0b10111011:                //Button 9 pressed
                data = 9;
                while (PINA == 0b10111011);
                break;

                case 0b10110111:                //Button # pressed
                data = 15;
                while (PINA == 0b10110111);
                break;

                case 0b01111110:                //Button A pressed
                data = 10;
                while (PINA == 0b01111110);
                break;

                case 0b01111101:                //Button B pressed
                data = 11;
                while (PINA == 0b01111101);
                break;

                case 0b01111011:                //Button C pressed
                data = 12;
                while (PINA == 0b01111011);
                break;

                case 0b01110111:                //Button D pressed
                data = 13;
                while (PINA == 0b01110111);
                break;
            }
        }
    }
}

int main(void)
{
    DDRA = 0b00001111;      //Port A upper inputs, lower outputs
    PORTA = 0b11111111;     //Enable pull up on inputs, send 1 to outputs
```

```
    while (1)
    {
            /* Your code here */
        Keyb ();            //Check keyboard
            /* Your code here */
    }
}
```

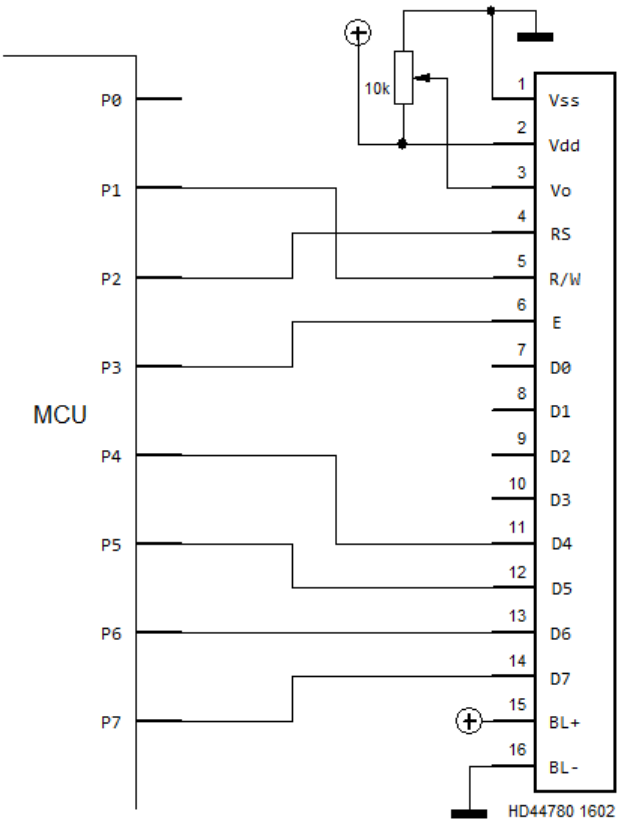
Η ρουτίνα του πληκτρολογίου **Keyb()** καλείται μέσα από το κυρίως πρόγραμμα και αποθηκεύει στη μεταβλητή **data** την πληροφορία από το πλήκτρο που πατήθηκε. Στον κώδικα που παρουσιάστηκε αποθηκεύεται η αριθμητική τιμή του πλήκτρου. Συνήθως το πάτημα ενός πλήκτρου διαρκεί 100 – 200ms, συνεπώς θα πρέπει στο διάστημα αυτό να έχει ολοκληρωθεί ο βρόχος του κυρίως προγράμματος ώστε να γίνει η κλήση της ρουτίνας του πληκτρολογίου στο χρόνο που διαρκεί το πάτημα του πλήκτρου, διαφορετικά δε θα αναγνωριστεί. Αν ο βρόχος του κυρίως προγράμματος διαρκεί αρκετό χρόνο θα πρέπει η ρουτίνα του πληκτρολογίου να καλείται με διακοπή από έναν Timer του μικροελεγκτή ή να καλείται η ρουτίνα σε διάφορα σημεία του προγράμματος. Σε αυτές τις περιπτώσεις χρειάζεται προσοχή στη χρήση των δεδομένων της **data** ώστε να είναι ενημερωμένη όταν διαβάζεται.

Όταν γίνει η κλήση της ρουτίνας **Keyb()** δίνεται αρχική τιμή **0b00000001** στη μεταβλητή **line** και στη συνέχεια ξεκινάει ένας βρόχος με μια **for** που επαναλαμβάνεται 4 φορές για τη σάρωση των γραμμών του πληκτρολογίου. Ξεκινώντας ο βρόχος δίνεται τιμή στο **PORTA** το συμπλήρωμα της **line** δηλαδή **0b11111110**, ώστε να ελεγχθεί η πρώτη γραμμή και γίνεται αριστερή ολίσθηση στην **line** για να προετοιμαστεί για τον επόμενο κύκλο. Σε αυτό το σημείο μπορεί να εισαχθεί μια καθυστέρηση της τάξεως του ms αν η συχνότητα λειτουργίας του μικροελεγκτή είναι υψηλή (στο παράδειγμα υπάρχει ως σχόλιο). Ακολουθεί ένας βρόχος με μια **while** όπου η συνθήκη της ικανοποιείται μόνο όταν υπάρχει πατημένο πλήκτρο σε αυτή τη γραμμή. Αν δεν υπάρχει πατημένο πλήκτρο το πρόγραμμα συνεχίζει με τον επόμενο κύκλο της **for**. Δίνεται ξανά τιμή στο **PORTA** το συμπλήρωμα της **line**, ώστε να ελεγχθεί αυτή τη φορά η δεύτερη γραμμή[1] και γίνεται αριστερή ολίσθηση στην **line** για να προετοιμαστεί για τον επόμενο κύκλο. Αν υπάρχει πατημένο πλήκτρο το πρόγραμμα εισέρχεται στην **while**. Στην **while** εκτελείται αρχικά μια εντολή καθυστέρησης για τους παρασιτικούς παλμούς και στη συνέχεια με την **switch** γίνεται αναγνώριση του πλήκτρου και αποθήκευση της πληροφορίας στην **data**. Πριν την έξοδο από την **switch** με την **break** σε κάθε **case** υπάρχει μια **while** η οποία έχει ως συνθήκη ισότητα του **PINA** με τη σταθερά της αντίστοιχης **case** που βρίσκεται. Ο ρόλος της **while** είναι να σταματάει το πρόγραμμα εκεί για όσο χρόνο είναι

[1] *Η **line** έχει ολισθήσει κατά αριστερά στον προηγούμενο κύκλο και συνεπώς το συμπλήρωμά της είναι **0b11111101**.*

πατημένο το πλήκτρο, ώστε να μην καταχωρείται ως νέο πάτημα κάθε φορά που καλείται η ρουτίνα. Το πρόγραμμα συνεχίζει τη λειτουργία του με την απελευθέρωση του πλήκτρου. Αν είναι δύο πλήκτρα πατημένα στην ίδια γραμμή δεν ικανοποιείται καμιά **case** και συνεπώς βγαίνει το πρόγραμμα από τη **switch** χωρίς να κάνει καμία ενέργεια. Αν είναι πατημένα δύο πλήκτρα στην ίδια στήλη θα αναγνωριστεί το πλήκτρο που η γραμμή του είναι σε χαμηλή λογική κατά τη σάρωση. Όταν ολοκληρωθεί η σάρωση και των τεσσάρων γραμμών το πρόγραμμα εξέρχεται από τη ρουτίνα του πληκτρολογίου.

11. Οθόνη LCD 2x16 (4x20)



Οι οθόνες LCD 2x16 ή 4x20 είναι πολύ εύκολες στη σύνδεση και στη χρήση τους σε μικροελεγκτές. Χρειάζονται ελάχιστες γραμμές κώδικα για την προετοιμασία τους (initialize) και για τις εντολές λειτουργίας τους. Στο θεωρητικό κύκλωμα του σχήματος παρουσιάζεται η σύνδεση μιας οθόνης στο port ενός μικροελεγκτή. Διαθέτουν 8 bit data bus αλλά είναι δυνατό να προγραμματιστούν για λειτουργία με 4 bit, ώστε μαζί με τα 3 σήματα ελέγχου να μπορούν να συνδεθούν σε ένα port. Όταν είναι προγραμματισμένες για λειτουργία 4 bit στέλνεται πρώτα το ανώτερο μέρος του byte (upper) και στη συνέχεια το κατώτερο (lower). Έχουν έναν καταχωρητή 80 byte (DD RAM) για αποθήκευση των δεδομένων που εμφανίζονται, ο οποίος είναι εγγραφής – ανάγνωσης. Στις οθόνες με 2 σειρές, τα 40 πρώτα byte αφορούν την πρώτη σειρά με διευθύνσεις στην DD RAM **0x00** έως **0x27** και τα επόμενα 40 τη δεύτερη με διευθύνσεις **0x40** έως **0x67**. Στις οθόνες με 4 σειρές, τα 20 πρώτα αφορούν την πρώτη σειρά με διευθύνσεις στην DD RAM **0x00** έως **0x13**, τα επόμενα 20 (21 – 40) την τρίτη σειρά με διευθύνσεις **0x14** έως **0x27**, τα επόμενα (41 – 60) τη δεύτερη σειρά με διευθύνσεις **0x40** έως **0x53** και τα τελευταία 20 (61 – 80) την τέταρτη με διευθύνσεις **0x54** έως **0x67**. Αν κάθε σειρά

έχει 16 θέσεις για εμφάνιση χαρακτήρων, εμφανίζονται τα δεδομένα που είναι αποθηκευμένα στις 16 πρώτες θέσεις του καταχωρητή, ενώ τα υπόλοιπα μπορούν να εμφανιστούν με ολίσθηση χρησιμοποιώντας την αντίστοιχη εντολή της οθόνης. Υπάρχει δυνατότητα να εμφανίζεται ή όχι ο κέρσορας ή να αναβοσβήνει, επίσης υπάρχει δυνατότητα να ολισθαίνει αυτόματα ο κέρσορας ή η οθόνη καθώς στέλνονται χαρακτήρες. Οι οθόνες διαθέτουν, ακόμα, οπίσθιο φωτισμό (back light) στα pin BL+, BL- και ρύθμιση αντίθεσης στο pin Vo.

Τα δεδομένα στέλνονται στην οθόνη σε κωδικοποίηση χαρακτήρων ASCII και εμφανίζονται όλοι οι λατινικοί χαρακτήρες και τα σύμβολα που υπάρχουν στις 128 πρώτες θέσεις. Για τις επόμενες 128 θέσεις του ASCII η ROM της οθόνης περιέχει πληροφορίες για εμφάνιση ειδικών χαρακτήρων και ανάλογα με την έκδοση της ROM μπορούν να εμφανιστούν Ελληνικοί, Κυριλλικοί, Κινέζικοι ή Γιαπωνέζικοι χαρακτήρες και κάποια μαθηματικά σύμβολα[1]. Επίσης υπάρχει η δυνατότητα να δημιουργηθούν pixel – pixel 8 χαρακτήρες, οι οποίοι αποθηκεύονται σε έναν καταχωρητή (CG RAM) και μπορούν να εμφανιστούν, όμως διαγράφονται με απώλεια τροφοδοσίας.

Η οθόνη δέχεται εντολές με **0** στο pin RS. Οι εντολές είναι για τη θέση του κέρσορα, τις ιδιότητες του κέρσορα, εντολές ολίσθησης, καθαρισμού της οθόνης, δημιουργίας χαρακτήρων και ανάγνωσης και έχει διαφορετικό χρόνο εκτέλεσης η κάθε μια. Η ανάγνωση χαρακτήρων γίνεται με **1** στο pin R/W. Οι πληροφορίες λαμβάνονται από την οθόνη με **1** στο pin E είτε αυτές είναι δεδομένα, είτε εντολές. Για την initialization της οθόνης πρέπει να εκτελεστεί αποστολή 3 φορές η πληροφορία **0b0011** στο **D7:D4** της οθόνης με διαφορετικό ενδιάμεσο χρόνο και στη συνέχεια να σταλεί η εντολή για λειτουργία 4 bit (μετά την εκκίνησή της η οθόνη είναι σε λειτουργία 8 bit, επειδή όμως δεν έχουμε συνδεδεμένα τα 4 LSB τα pin βλέπουν **0**). Στη συνέχεια παρουσιάζεται ένα παράδειγμα κώδικα σε γλώσσα C για initialization οθόνης:

```
#define lcdport PORTA                   // Define LCD PORT
#define lcdddr DDRA                     // Define LCD DDR
#define En 0b00001000                   // E signal

void InitLCD (void)                     // Start up, initialization rutine
{
    lcdddr |= ~(1<<0);                  // PORTx1:7 outputs (LCD)
    lcdport &= (1<<0);
    _delay_ms (20);                     // Power on delay 20ms
    out = 0b00110000;
    lcdport = (lcdport | out);
    _delay_us (100);
    lcdport = (lcdport | En);           // Init
```

[1] *Η πιο συνηθισμένες ROM είναι οι PN, PS, PM με English – Japanese Font.*

```
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_ms (5);

    lcdport = (lcdport | En);            // Init
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_us (200);

    lcdport = (lcdport | En);            // Init
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_ms (5);

    lcdport &= (1<<0);
    out = 0b00100000;                    // 4 bit interface
    lcdport = (lcdport | out);
    _delay_us (100);
    lcdport = (lcdport | En);
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_us (50);
}
```

Για την αποστολή δεδομένων ή εντολών στην οθόνη μπορούμε να γράψουμε ρουτίνες ώστε να τις καλούμε στο κυρίως πρόγραμμα. Στη συνέχεια παρουσιάζεται ένα παράδειγμα ρουτίνας αποστολής εντολών και η κλήση της ρουτίνας για αποστολή εντολής:

```
void WrIn (uint8_t tmp)              // Write Instruction routine
{
    lcdport &= (1<<0);
    out = tmp & 0b11110000;          // Upper
    lcdport = (lcdport | out);
    _delay_us (100);
    lcdport = (lcdport | En);
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_us (100);
    lcdport &= (1<<0);
    out = tmp << 4;                  // Lower
    lcdport = (lcdport | out);
    _delay_us (100);
    lcdport = (lcdport | En);
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_us (50);
}

#define CDRC 0b00000001              // Clear Display and Reset Cursor (2ms delay)

WrIn (0b00101100);                   // 001, 4 bit, 2 lines 2x16, 5x11 dots, xx
WrIn (0b00001111);                   // 00001, Display on, cursor on, blinking on
WrIn (0b00000110);                   // 000001, Cursor increase, display not shift
WrIn (CDRC);                         // Clear display, reset cursor (2ms delay)
_delay_ms (2);
```

Στη συνέχεια παρουσιάζεται ένα παράδειγμα ρουτίνας αποστολής δεδομένων και η κλήση της ρουτίνας για αποστολή του γράμματος C:

```
#define Da 0b00000100              // RS, data signal (0 for instruction)

void WrDa (uint8_t tmp)            // Write Data routine
{
    lcdport &= (1<<0);
    out = tmp & 0b11110000;                // Upper
    lcdport = (lcdport | out | Da);
    _delay_us (100);
    lcdport = (lcdport | En);
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_us (100);
    lcdport &= (1<<0);
    out = tmp << 4;                        // Lower
    lcdport = (lcdport | out | Da);
    _delay_us (100);
    lcdport = (lcdport | En);
    _delay_us (100);
    lcdport = (lcdport & ~En);
    _delay_us (50);
}

WrDa ('C');
```

Με παρόμοιο τρόπο μπορούμε να γράψουμε ρουτίνα εντολής μετακίνησης του κέρσορα ώστε να αποθηκευτούν τα δεδομένα σε συγκεκριμένη θέση. Στη συνέχεια παρουσιάζεται ένα παράδειγμα ρουτίνας και η κλήση για μετακίνηση στη δεύτερη σειρά πρώτη θέση:

```
void SeCu (uint8_t addr)                   // Send cursor to a specific address
{
    WrIn (0b10000000 | addr);
}

SeCu (0x40);
```