HAL 9000.1

Εργασία για το μάθημα : Εισαγωγή στους Μικροεπεξεργαστές των 8 Bits

Αλκαλάι Ερρίκος (3889)
Καλάργαρης Χαράλαμπος (3929)
Κωτσόγιαννης - Τεφτσόγλου Ιωάννης (3961)

2009

Τμημα Μηχανικων Η/Υ & Πληροφορικης

Περιεχόμενα

Προεσκόπιση

Η εργασία αυτή αποτελεί μέρος του μαθήματος “Εισαγωγή στους μικροεπεξεργαστές των 8 bits”. Σκοπός του μαθήματος ήταν μέσα από την θεωρητική παρουσίαση του τρόπου λειτουργίας ορισμένων μικροεπεξεργαστών να εφαρμόσουμε στην πράξη τον προγραμματισμό και τη λειτουργία ενός micro controller. Η επιλογή αυτού είναι ο PIC16F618 της MicroChip.

Στην μνήμη RAM ενός μικροϋπολογιστικού συστήματος βρίσκεται αποθηκευμένη μία απλή εικόνα 10Χ10 ασπρόμαυρων εικονοστοιχείων που μπορεί να περιέχει ένα απλό σχήμα: ευθύγραμμο τμήμα (Ε), κύκλος (Κ) ή τετράγωνο (Τ). Η εικόνα αυτή μπορεί να έχει ληφθεί μέσω Σειριακής Θύρας από ΗΥ. Ο μικροεπεξεργαστής θα πρέπει να διαβάσει την εικόνα από τη μνήμη και αφού ανιχνεύσει ποιος τύπος σχήματος υπάρχει στην εικόνα (μόνο ένα σχήμα) εντοπίζει τις διαστάσεις και τη θέση του. Στην περίπτωση Ευθύγραμμου τμήματος αποστέλλει μέσω Σειριακής θύρας στον ΗΥ τα στοιχεία Line-X1-Y1-X2-Y2 όπου τα σημεία (X1,Y1) και (X2,Y2) είναι οι συντεταγμένες των άκρων. Στην περίπτωση κύκλου αποστέλλονται τα στοιχεία Circle-X1-Y1-P όπου (X1,Y1) είναι οι συντεταγμένες του κέντρου και P η ακτίνα του. Στην περίπτωση ορθογωνίου αποστέλλονται τα στοιχεία Rectangle-X1-Y1-X2-Y2 όπου (X1,Y1), (X2,Y2) είναι οι διαγώνιες κορυφές του ορθογωνίου - γεωμετρικά αρκούν για να αναπαράγουμε το σχήμα.

Στην συνέχεια της παρούσας αναφοράς θα αναλύσουμε τις παραδοχές που κάναμε (μόνο και μόνο η προσπάθεια αναπαράστασης ενός κύκλου στην μνήμη είναι ένα θέμα αρκετά ασαφές) καθώς και θα παρουσιάσουμε εν συντομία τους αλγορίθμους αναγνώρισης/συμπίεσης ώστε να καταλάβει ο αναγνώστης τι είσοδο δέχονται αυτές αλλά κυρίως τι έξοδο βγάζουν. Σε δεύτερο επίπεδο θα αναλύσουμε εκτενώς τόσο την επικοινωνία μέσω σειριακής θύρας όσο και το κυρίως πρόγραμμα. Τέλος παρατίθεται μία εικονική εκτέλεση της ζητούμενης εργασίας (αποστολή αρχείου - επεξεργασία – αποστολή συμπιεσμένου αρχείου).

Η επικοινωνία του pic με τον υπολογιστή πάλι μπορεί να σπάσει σε δύο υποκατηγορίες, την αποστολή από τον Η/Υ του αρχείου με την εικόνα, και την αποστολή από τον pic του σχήματος που αναγνώρισε αλλά και του συμπιεσμένου αρχείου.

Κατά την αλγοριθμική διαδικασία, εκτελούνται δύο προγράμματα. Ένα για την αναγνώριση του σχήματος (κύκλος, τετράγωνο, ευθεία). Και άλλο ένα για την συμπίεση, το οποίο καλείται μόνο στις περιπτώσεις του τετραγώνου και του κύκλου, όπου κατά την εκτέλεση του γίνεται περαιτέρω επεξεργασία του αρχείου εισόδου ώστε να υπολογιστούν οι αναγκαίες τιμές για την συμπίεση των σχημάτων αυτών.

Στην συνέχεια της παρούσας αναφοράς θα αναλύσουμε τις παραδοχές που κάναμε (μόνο και μόνο η προσπάθεια αναπαράστασης ενός κύκλου στην μνήμη είναι ένα θέμα αρκετά ασαφές), μία εν συντομία παρουσίαση των αλγορίθμων αναγνώρισης/συμπίεσης ώστε να καταλάβει ο αναγνώστης τι είσοδο δέχονται αυτές αλλά κυρίως τι έξοδο βγάζουν. Σε δεύτερο επίπεδο θα αναφερθούμε εκτενώς στην επικοινωνία μέσω σειριακής θύρας και θα αναλύσουμε πλήρως τους αλγόριθμους που υλοποιήσαμε. Τέλος παρατίθεται μία εικονική εκτέλεση της εργασίας (αποστολή αρχείου - επεξεργασία - αποστολή συμπιεσμένου αρχείου).

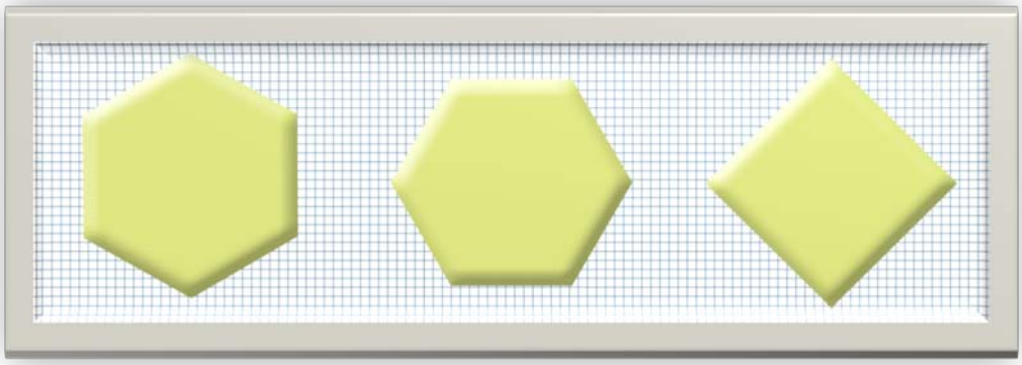
Παραδοχές

Όσον αφορά τα σχήματα :

Για τις ανάγκες αυτής της εργασίας θα θεωρήσουμε:

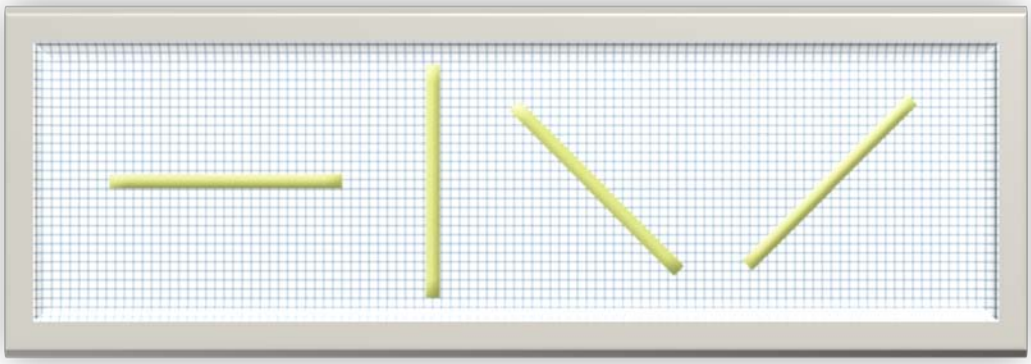
Κύκλοι

Οι κύκλοι αναγνωρίζονται ως παράγωγα αυτών των 3 αρχέτυπων κύκλων.



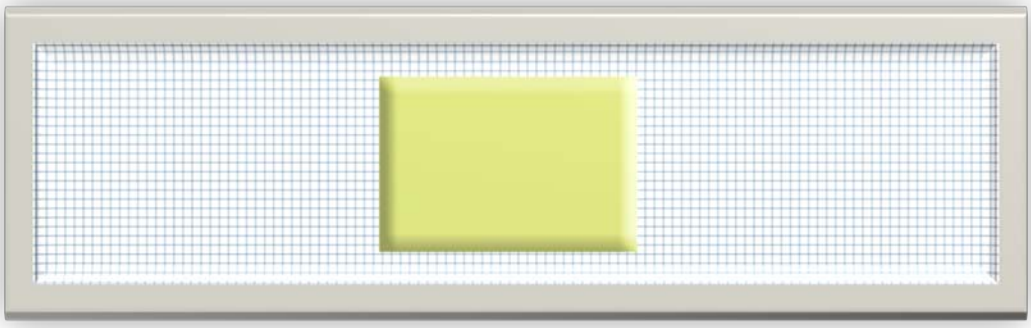
Ευθείες

Οποιαδήποτε ευθεία οριζόντια, κάθετη ή με κλίση 45°.



Παραλληλόγραμμα

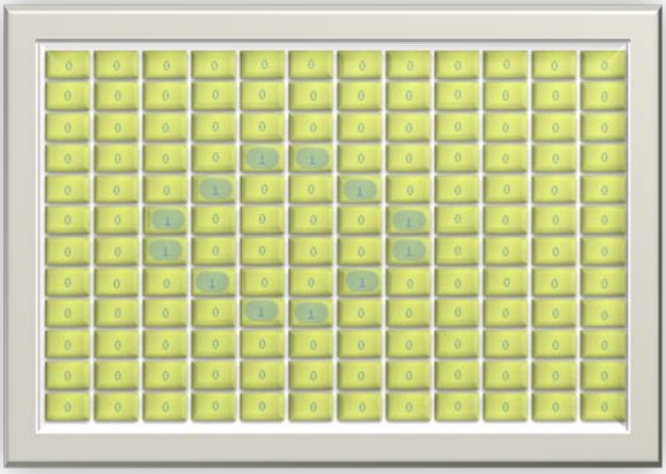
Όπως βλέπουμε θεωρούμε ρομβοειδή σχήματα σαν κύκλους, από την στιγμή που κάνουμε αυτήν την παραδοχή, γίνεται ευκόλως αντιληπτό πως δεν μπορούμε να αναγνωρίσουμε οποιοδήποτε παραλληλόγραμμο (αφού θα θεωρηθεί κύκλος), αλλά μόνο τα ορθογώνια παραλληλόγραμμα.



Έτσι θεωρούμε την οντότητα τετράγωνο - rectangle - και αναγνωρίζουμε μόνο αυτή, να αναφερθεί εδώ πως για την συμπίεση του τετραγώνου δεν χρειάζονται πλέον οι 3 κορυφές του, αλλά μόνο οι 2 διαγώνιες.

Όσον αφορά την μορφή του αρχείου :

Θεωρούμε μία NxN διαστάσεων εικόνα μία διαδοχή N^2 bit όπου το '0' αναπαριστά την απουσία χρώματος, ενώ το '1' αναπαριστά το "χρωματισμένο" pixel. Για μια εικόνα πχ. 12x12 pixels χρειαζόμαστε 144 bit ή αλλιώς 18 bytes.

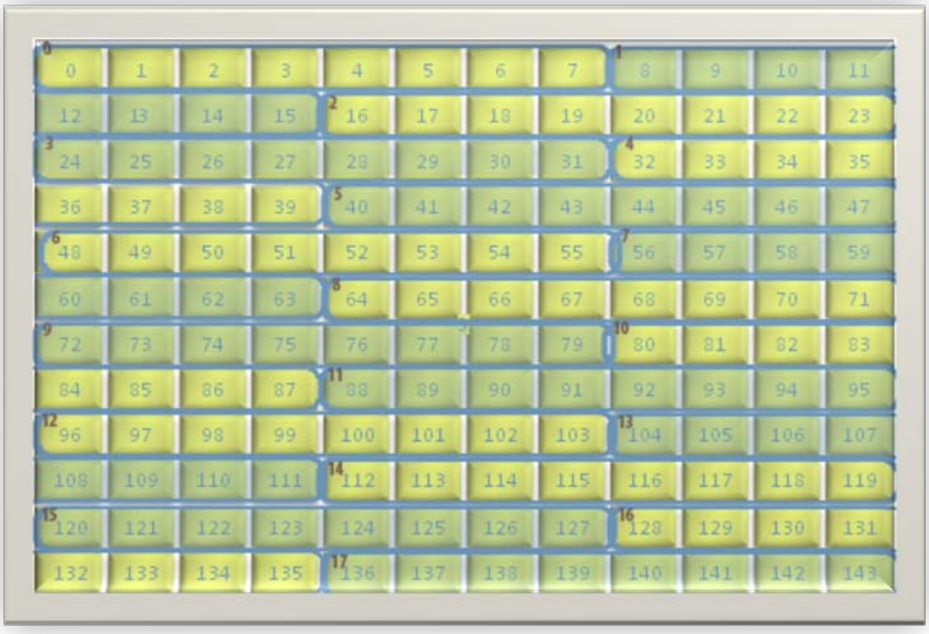


Έτσι το pixel στην θέση (12,3) έχει την μοναδική θέση:

$$\text{position} = 12 \cdot (\text{i} - 1) + \text{j} - 1 = 35$$

και το bit που το αναπαριστά βρίσκεται στο:

$$byte = \left\lfloor {}^{position}/_{8} \right\rfloor = 4, \text{ και } bit = 8 - position\%8 = 5$$

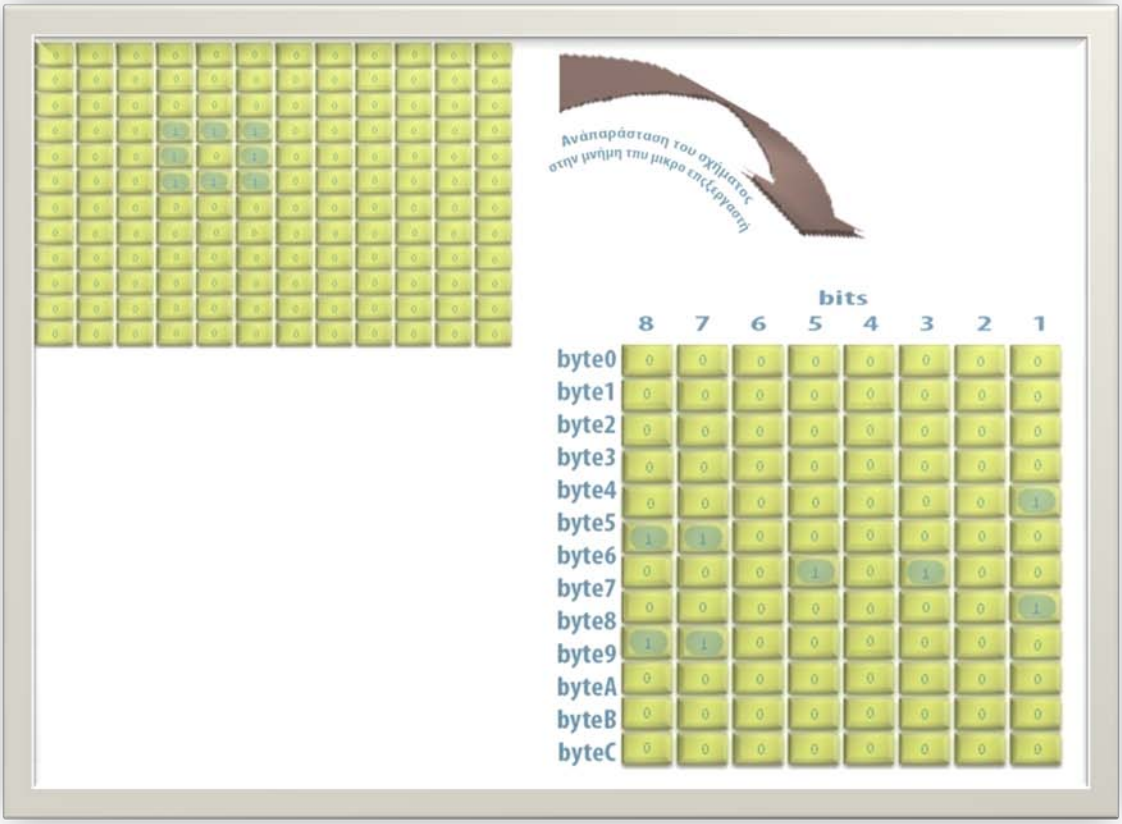


Αξίζει να σημειωθεί ότι η αρίθμηση του πίνακα και των byte ξεκινάει από το μηδέν, σε αντίθεση με αυτή των bits που ξεκινάει από το 8 και φτάνει μέχρι το μηδέν

$byte_i$							
bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1

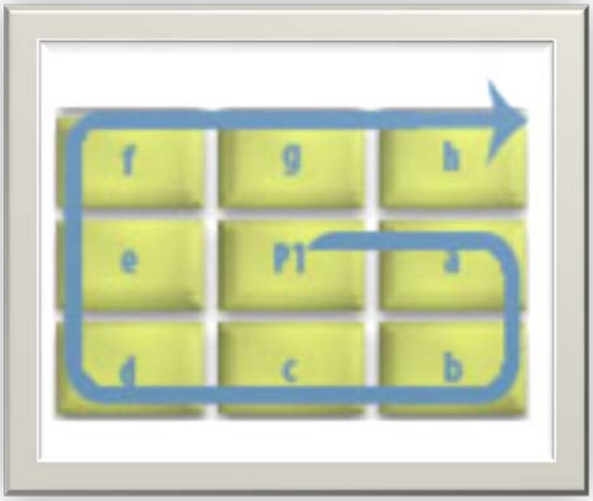
Για λόγους που θα γίνουν αντιληπτοί κατά την περιγραφή του αλγορίθμου αναγνώρισης θεωρούμε πως το αρχείο εικόνας που μας έστειλε ο υπολογιστής είναι ένα πλέγμα που τα εξωτερικά του στοιχεία είναι πάντα 0. Αυτό μπορεί να φαίνεται αρχικά σαν μία αχρείαστη σπατάλη μνήμης της τάξης του O(4N+4) (πχ, για να έχουμε "χρήσιμο" πλέγμα μεγέθους 10x10, θα πρέπει να στείλουμε ένα 12x12 όπου τα εξωτερικά bit θα είναι όλα 0 - για αυτόν τον λόγο κιόλας πριν αναφέρθηκε το 12x12 πλέγμα).

Ένα παράδειγμα :



Σύντομη παρουσίαση αλγορίθμων

Για να γίνει εύκολα αντιληπτή η αρχή λειτουργίας του μηχανισμού αναγνώρισης σχημάτων ας τον φανταστούμε σαν μία κεφαλή ανάγνωσης η οποία προσπελαύνει το αρχείο της εικόνας και ανανεώνει έναν καταχωρητή. Αρχικά η κεφαλή βρίσκεται στο πρώτο στοιχείο του αρχείου και μετατοπίζεται κατά μία θέση μέχρι να βρει "μαυρισμένο" pixel. Όταν βρει το πρώτο μαυρισμένο στοιχείο, αυτό σημαίνει πως βρήκαμε το πρώτο pixel που ανήκει στο σχήμα μας - έστω αυτό το pixel “p1”. Η κεφαλή τώρα αντί να συνεχίσει να προσπελαύνει γραμμικά το υπόλοιπο αρχείο θα ελέγξει μόνο τα γειτονικά pixel του p1 με την φορά που δείχνει το βέλος.



Σε περίπτωση που βρει κάποιο "μαυρισμένο" θα μεταπηδήσει σε αυτό και θα κάνει την ίδια δουλειά έως ότου γυρίσει στο startpixel ή έως ότου φτάσει σε pixel το οποίο δεν έχει "μαυρισμένα" γειτονικά. Κάθε φορά που η κεφαλή αλλάζει p1, στέλνει στην μνήμη την πληροφορία για το άλμα που έκανε, έτσι μπορούμε να έχουμε μέχρι και 8 διαφορετικά άλματα, όπου το καθένα από αυτά έχει και διαφορετική γεωμετρική σημασία.

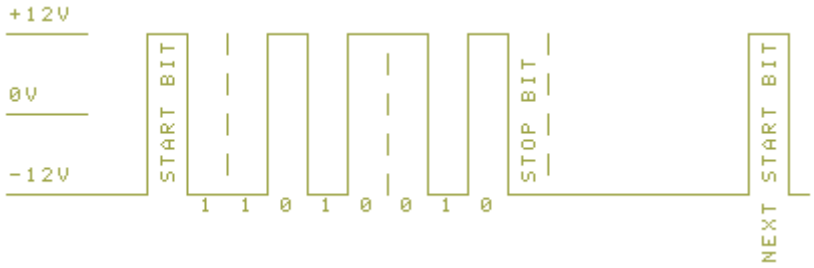
Έτσι έχουμε πετύχει δύο πράγματα, αφενός να προσπελάσουμε ολόκληρο το σχήμα και αφετέρου - και πιο σημαντικά - να καταγράψουμε την πορεία του μονοπατιού που ακολουθήθηκε για να προσπελάσουμε το σχήμα.

Προγραμματισμός επικοινωνίας PIC/PC

Ένας από τους λόγους για τον οποίο επιλέχτηκε ο συγκεκριμένος controller ήταν γιατί διαθέτει περιφερειακό σειριακής επικοινωνίας (USART). Όμως για τον άριστο και εύκολο συγχρονισμό του controller με τον υπολογιστή θα πρέπει να χρησιμοποιηθεί το πρωτόκολλο RS232.

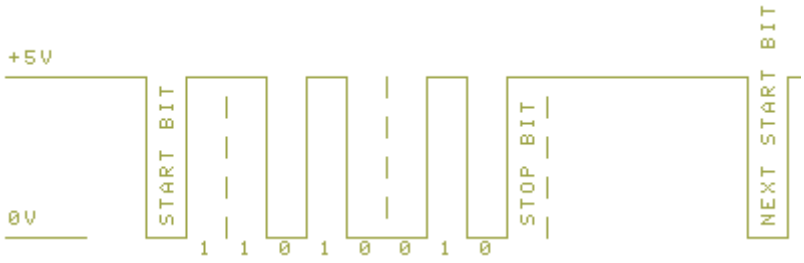
Το RS232 είναι ένα πρωτόκολλο ασύγχρονης σειριακής επικοινωνίας που χρησιμοποιείται ευρέως στους υπολογιστές. Η λέξη ασύγχρονη δηλώνει ότι δεν χρειάζεται κάποιο ρολόι για να συγχρονιστούν τα δεδομένα, αλλά γίνεται με χρήση start και stop παλμών. Τα σήματα που αντιλαμβάνεται ο υπολογιστής είναι της τάξης των 12 Volts για καλύτερη αξιοπιστία και αύξησης του διαθέσιμου εύρους τιμών για την επεξεργασία δεδομένων. Ο controller χρησιμοποιεί σήματα της τάξης των 5Volts και φυσικά αυτό δεν εμποδίζει να πραγματοποιηθεί η επικοινωνία του με τον υπολογιστή. Το ενδιάμεσο κύκλωμα που παρεμβάλλεται είναι το MAX232 και αναλαμβάνει την άμεση μετατροπή των σημάτων της τάξης των 5Volts σε αυτά των 12Volts και αντίθετα.

Το πρωτόκολλο χρησιμοποιεί διάφορες μορφές χειρισμού των δεδομένων και ταχύτητες. Το πιο γνωστό είναι το 8N1,που σημαίνει 8 bit δεδομένων χωρίς ψηφίο ισοτιμίας και το stop bit είναι υψηλό δυναμικό. Όσο αφορά την ταχύτητα επικοινωνίας θα χρησιμοποιηθεί 9600 BPS (Bits Per Second)



Αυτό είναι ένα τυπικό σήμα που υπάρχει πάνω στη γραμμή του RS232

Αυτό είναι το σήμα που υπάρχει πάνω στο ποδαράκι επικοινωνίας του PIC. Φαίνεται αντιστραμμένο αλλά στην πραγματικότητα είναι το σωστό λόγω της αρνητικής λογικής του controller



Ο προγραμματισμός του controller είναι πάρα πολύ απλός αρκεί να εφαρμοστούν τα παραπάνω ενημερώνοντας τους κατάλληλους καταχωρητές του. Αρχικά θα πρέπει να δηλωθεί η δεύτερη θύρα (RB1/RX) του portB σαν είσοδος και η τρίτη θύρα (RB2/TX) ως έξοδος .ώστε να επιτευχθεί η επικοινωνία με το MAX232.

Οι εντολές που θα χρησιμοποιηθούν είναι:

```
movlw b'00000100'
movwf PORTB
movlw b'11110010'
movwf TRISB
```

Για τον καθορισμό των παραπάνω χαρακτηριστικών θα μεταφέρουμε στους καταχωρητές που αφορούν το USART τις κατάλληλες τιμές

```
movlw 0x19              ; 0x19=9600 bps
movwf SPBRG

movlw b'00100100'       ; brgh = high (2)
movwf TXSTA             ; Ενεργοποίηση της ασύγχρονης
                        ; επικοινωνίας

bcf STATUS,RP0          ; RAM BANK 0
movlw b'10010000'       ; Ενεργοποίηση της ασύγχρονης λήψης
movwf RCSTA
```

Για την λήψη ενός χαρακτήρα η διαδικασία είναι να ελέγχεται συνεχώς το ψηφίο που υποδηλώνει ότι δεν είναι γεμάτος ο buffer της διαδικασίας της λήψης. Μόλις γίνει 1 αυτό το bit τότε ο buffer γέμισε και θα μπορεί ο controller να διαβάσει τον χαρακτήρα που δέχτηκε.

```
receive     btfss PIR1,RCIF         ; (5) Έλεγξε το
                                    ; ψηφίο για τον buffer
            goto receive

            movf RCREG,W            ;Αποθήκευσε τα δεδομένα
                                    ;στον W
            return
```

Στην περίπτωση που ο controller θέλει να στείλει έναν χαρακτήρα τότε θα μεταφέρει τα δεδομένα στον καταχωρητή TXREG και μετά θα περιμένει μέχρι να αδειάσει ο καταχωρητής ολίσθησης, δηλαδή να έχουν σταλεί όλα τα δεδομένα

```
send        movwf TXREG         ; Στείλε τα δεδομένα στον W

            bsf STATUS,RP0          ; RAM BANK 1

WtHere      btfss TXSTA,TRMT        ; (1) transmission is
                                    ;complete if hi
            goto WtHere

            bcf STATUS,RP0          ; RAM BANK 0
            return
```

Αναλυτική λειτουργία αλγορίθμων

Στην παρούσα ενότητα θα παρουσιαστεί ψευδοκώδικας που υλοποιεί τους αλγόριθμους αναγνώρισης σχήματος και συμπίεσης.

Αναγνώριση Σχήματος

```
shape_rec
step1 // Εύρεση πρώτου μαυρισμένου
pixel.
        i=0;
        while( img_file[i]!=1 )
                i++;
        startpx = i;
        p1 = i;
step2
        if ( img_file[p1 + 1] == 1 ) then
        {
                p1 = p1 + 1;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x80;
                goto step2;
        }

        if ( img_file[p1 + 13] == 1 ) then
        {
                p1 = p1 + 13;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x40;
                goto step2;
        }

        if ( img_file[p1 + 12] == 1 ) then
        {
                p1 = p1 + 12;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x20;
                goto step2;
        }

        if ( img_file[p1 + 11] == 1 ) then
        {
                p1 = p1 + 11;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x10;
                goto step2;
        }

        if ( img_file[p1 - 1] == 1 ) then
        {
                p1 = p1 - 1;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x08;
                goto step2;
        }

        if ( img_file[p1 - 13] == 1 ) then
        {
                p1 = p1 - 13;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x04;
                goto step2;
        }

        if ( img_file[p1 - 12] == 1 ) then
        {
                p1 = p1 - 12;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x02;
                goto step2;
        }

        if ( img_file[p1 - 11] == 1 ) then
        {
                p1 = p1 - 11;
                if (p1 == startpx )
                        goto END;
                shape_flag ^= 0x01;
                goto step2;
        }
        else endpixel = p1;

end

return(startpixel, endpixel, shape_flag);
```

Σε κάθε συνθήκη ελέγχου ο αλγόριθμος ελέγχει εάν το εκάστοτε γειτονικό στοιχείο είναι μαυρισμένο και εάν είναι να ενημερώσει το shape_flag και να μεταπηδήσει η "κεφαλή" σε εκείνο το στοιχείο. Ο αλγόριθμος σταματάει την αναδρομή του εάν φτάσει στο σημείο από το οποίο ξεκίνησε, ή εάν φτάσει σε στοιχείο το οποίο δεν έχει γειτονικό "μαυρισμένο".

Συμπίεση Σχήματος

shape_archive

```
if (shape_flag = line)
        return (startpx, endpixel)

if (shape_flag = rect)
        call (find_rect)

if (shape_flag = circle)
        call find_radius
```

Ειδικές Περιπτώσεις:

Ορθογώνιο

Η ρουτίνα find_rect απλά βρίσκει το κάτω διαγώνιο στοιχείο του σχήματος - το πάνω διαγώνιο το ξέρουμε ήδη, είναι το startpixel.

```
find_rect
 oldpath = 0;
newpath = 0;
p1 = startpixel
step2
if ( (oldpath==c)&&(newpath==e) )
{pointB = p1; goto END;}

else if ( img_file[p1 + 1] == 1 )
{
        p1 = p1 + 1;
        oldpath = newpath;
        newpath = a;
        goto step2;
}
if ( img_file[p1 + 13] == 1 ) then
{
        p1 = p1 + 13;
        oldpath = newpath;
        newpath = b;
        goto step2;
}
if ( img_file[p1 + 12] == 1 ) then
{
        p1 = p1 + 12;
        oldpath = newpath;
        newpath = c;
        goto step2;
}
if ( img_file[p1 + 11] == 1 ) then
{
        p1 = p1 + 11;
        oldpath = newpath;
        newpath = d;
        goto step2;
}
if ( img_file[p1 - 1] == 1 ) then
{
        p1 = p1 - 1;
        oldpath = newpath;
```

```
        newpath = e;
        goto step2;
}

if ( img_file[p1 - 13] == 1 ) then
{
        p1 = p1 - 13;
        oldpath = newpath;
        newpath = f;
        goto step2;
}
if ( img_file[p1 - 12] == 1 ) then
{
        p1 = p1 - 12;
        oldpath = newpath;
        newpath = g;
        goto step2;
}
if ( img_file[p1 - 11] == 1 ) then
{
        p1 = p1 - 11;
        oldpath = newpath;
        newpath = h;
        goto step2;
}
end

return (pointB, startpixel)
```

Κύκλος

Η ρουτίνα find_radius υπολογίζει την περίμετρο του σχήματος σαρώνοντας ένα ένα τα σημεία του, καθώς και ένα σημείο του σχήματος στο οποίο αλλάζει η "κλίση". Μόλις βρεθεί η περίμετρος εύκολα η γεωμετρία μας δίνει ότι $Περίμετρος = ακτίνα \cdot 2 \cdot \pi$. Όσον αφορά το κέντρο αρκεί να προχωρήσουμε κατά απόσταση ίση με την ακτίνα στο σημείο που αλλάζει η κλίση. Έτσι θα έχουμε βρει ένα προσεγγιστικό κέντρο

```
find_radius
p1 = startpixel;
circumference=0;
step2
if ( (oldpath==c)&&(newpath!=c) )
        {temp = p1;}
else
if ( img_file[p1 + 1] == 1 ) then
{
        p1 = p1 + 1;
        oldpath = newpath;
        newpath = a;
        circumference++;
        goto step2;
}
if ( img_file[p1 + 13] == 1 ) then
{
        p1 = p1 + 13;
        oldpath = newpath;
        newpath = b;
        circumference++;
        goto step2;
}
```

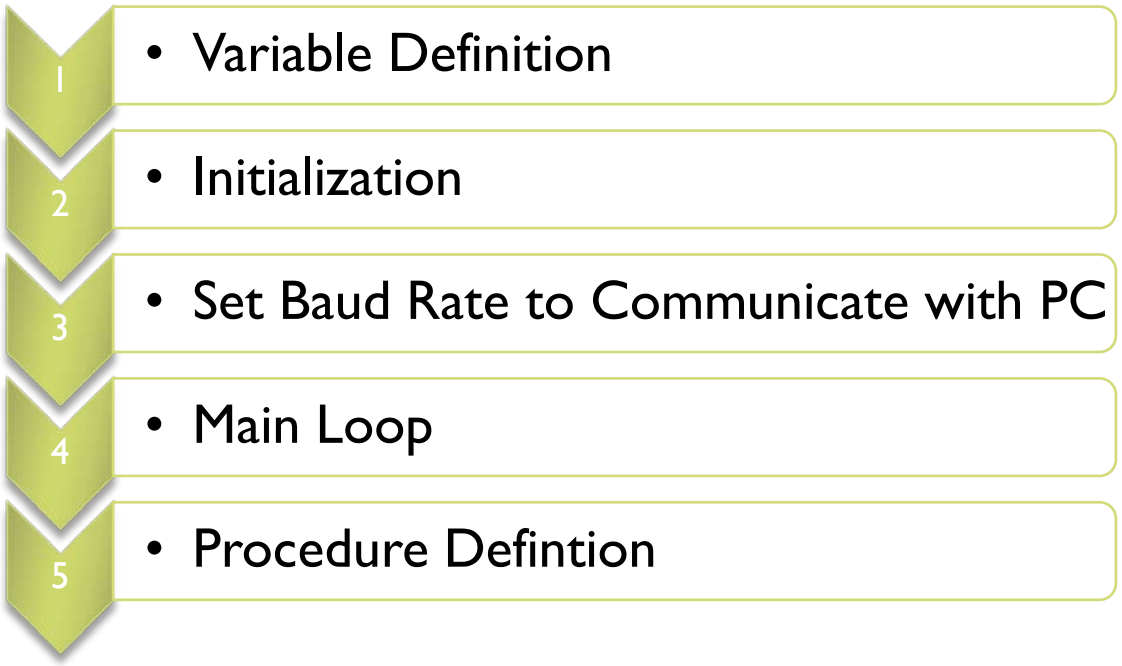
```
if ( img_file[p1 + 12] == 1 ) then
{
        p1 = p1 + 12;
        oldpath = newpath;
        newpath = c;
        circumference++;
        goto step2;
}
if ( img_file[p1 + 11] == 1 ) then
{
        p1 = p1 + 11;
        oldpath = newpath;
        newpath = d;
        goto step2;
}
if ( img_file[p1 - 1] == 1 ) then
{
        p1 = p1 - 1;
        oldpath = newpath;
        newpath = e;
        circumference++;
        goto step2;
}
if ( img_file[p1 - 13] == 1 ) then
{
        p1 = p1 - 13;
        oldpath = newpath;
        newpath = f;
        circumference++;
        goto step2;
}

if ( img_file[p1 - 12] == 1 ) then
{
        p1 = p1 - 12;
        oldpath = newpath;
        newpath = g;
        circumference++;
        goto step2;
}
```

```
if ( img_file[p1 - 11] == 1 ) then
{
        p1 = p1 - 11;
        oldpath = newpath;
        newpath = h;
        circumference++;
        goto step2;
}

step3
radius = circumference / 6.2;
centre = temp - radius

return (radius, centre)
```

Περιγραφή πλήρους προγράμματος

Για την καλύτερη δομή του προγράμματος και την κατανόηση του από τρίτους, αυτό έχει χωριστεί σε μικρές διαδικασίες και τμήματα.

Η δομή του προγράμματος έχει την παρακάτω μορφή:



1.Variable definition

Οι μεταβλητές που χρησιμοποιούμε είναι οι παρακάτω:

Όνομα	Θέση Μνήμης	Λειτουργία
cur_point	0x20	Περιέχει το offset της θέσης μνήμης του στοιχείου που ελέγχουμε.
cur_byte	0x21	Offset από την αρχή των bytes
cur_bit	0x22	Bit του byte που ελέγχουμε
startpx	0x23	Το πρώτο “έγχρωμο” pixel
endpx	0x24	Το τελευταίο pixel που βρίσκει ο αλγόριθμος
shape	0x25	
newflag	0x26	Μάσκες για τον αλγόριθμο συμπίεσης, αναπαριστούν την κλίση του σχήματος.
oldflag	0x27	
pointB	0x28	
ACC0	0x29	Μεταβλητές για αλγόριθμο πολλαπλασιασμού και διαίρεσης
ACC1	0x2A	
TEMP0	0x2B	
TEMP1	0x2C	
temp	0x2D	Προσωρινές μεταβλητές για εναλλαγές στον καταχωρητή W.
temp2	0x2E	
temp3	0x2F	
array	0x30 - 0x42	Σε αυτές τις θέσεις μνήμης αποθηκεύεται το σχήμα. 144 bit, δηλαδή 18 bytes
visit_flag	0x43	Κρατάει αν ο αλγόριθμος έχει επισκεφτεί το cur_point
temp_v	0x44	Προσωρινή μεταβλητή για εναλλαγές στον καταχωρητή W.
the_bit	0x45	Έχει την τιμή του cur_bit
temp_bit	0x46	Προσωρινές μεταβλητές για εναλλαγές στον καταχωρητή W.
temp_byte	0x47	
check_point	0x48	
p	0x49	Μεταβλητή που κρατάει το cur_point για τον αλγόριθμο check_neighbors
temp_v_bit	0x4A	Προσωρινή μεταβλητή για εναλλαγές στον καταχωρητή W.
the_byte	0x4B	Έχει την τιμή του cur_byte
temp_isv_bit	0x4C	
shape_rec	0x4D	Μεταβλητή που σε κάθε bit της αναπαριστά αν έχει ενεργοποιηθεί κάποιο flag αλλαγή κλίσης
shape_mask	0x4E	Μάσκα για πού δηλώνει ποιο flag ενεργοποιήθηκε
visited	0x50 - 0x62	Σε αυτές τις θέσεις μνήμης αποθηκεύεται ένας χάρτης που δηλώνει αν ποια στοιχεία έχουν επισκεφτεί από τον αλγόριθμο
cord_x	0x63	Συντεταγμένες x,y στον πραγματικό κόσμο (σχήμα 10x10)
cord_y	0x64	

2.Initialization

```
; --------------------------------
; SET ANALOG/DIGITAL INPUTS PORT A
; --------------------------------
      movlw 7
      movwf CMCON         ; CMCON=7 set comperators off

; ----------------
; INITIALIZE PORTS
; ----------------
      movlw b'00000000'   ; set up portA
      movwf PORTA
      movlw b'00000100'   ; RB2(TX)=1 others are 0
      movwf PORTB
      bsf STATUS,RP0            ; RAM BANK 1
      movlw 0xFF
      movwf TRISA         ; portA all pins input
      movlw b'11110010'   ; RB7-RB4 and RB1
                                ; RX)=input, others output
      movwf TRISB
```

3.Set Baud Rate to communicate with PC

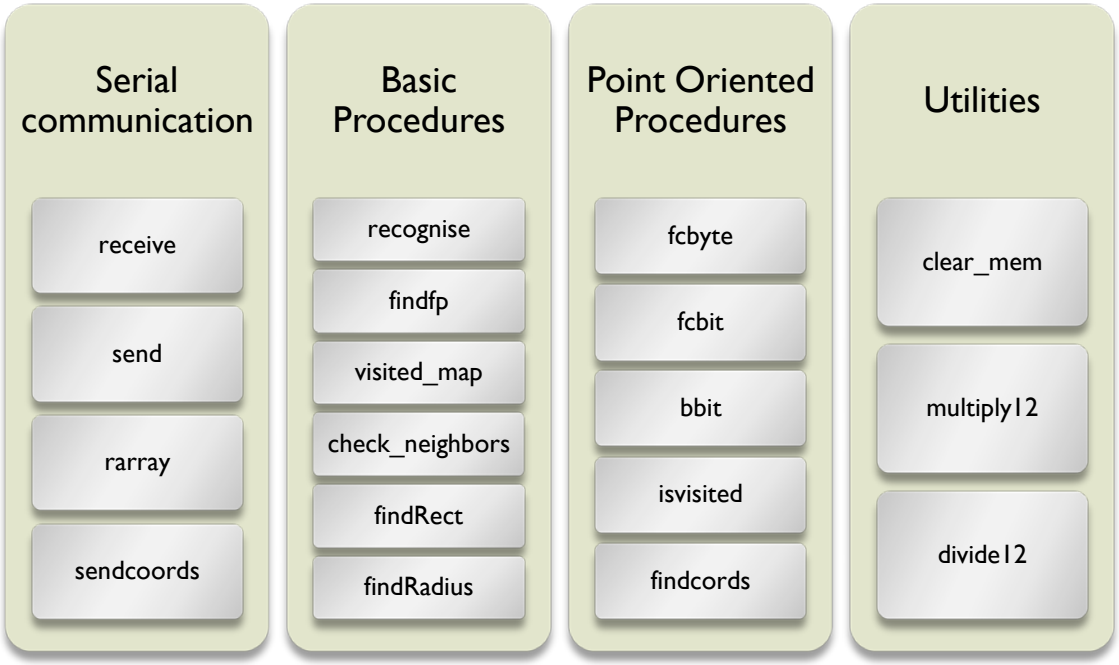
```
; ------------------------------------
; SET BAUD RATE TO COMMUNICATE WITH PC
; ------------------------------------

      movlw 0x19                ; 0x19=9600 bps (0x0C=19200 bps)
      movwf SPBRG
      movlw b'00100100'   ; brgh = high (2)
      movwf TXSTA         ; enable Async Transmission,
                                ; set brgh

      bcf STATUS,RP0            ; RAM BANK 0
      movlw b'10010000'   ; enable Async Reception
      movwf RCSTA
```

4.Main loop

Στην main καλούνται όλες οι βασικές διαδικασίες.

```
; ---------
; MAIN LOOP
; ---------
loop
      call clear_mem
      ;movlw 0x50

      call rarray

      call recognise

      call findShape

      goto endoff
```

5.Procedure Definition



Serial Communication

Η λειτουργία των send & receive εξηγήθηκαν σε παραπάνω ενότητα

```
; -----------------------------------------
; RECEIVE CHARACTER FROM RS232 AND STORE IN W
; -----------------------------------------
receive
       btfss PIR1,RCIF     ; (5) check for received ;data
       goto receive

       movf RCREG,W  ;save received data in W
       return

; ---------------------------------------------------------
; SEND CHARACTER IN W VIA RS232 AND WAIT UNTIL FINISHED
; SENDING
; ---------------------------------------------------------
Send
       movwf TXREG         ; send data in W

TransWt
       bsf STATUS,RP0              ; RAM BANK 1
WtHere
       btfss TXSTA,TRMT            ; (1) transmission is complete
                                   ; if hi
       goto WtHere

       bcf STATUS,RP0              ; RAM BANK 0
       return
```

Με έμμεση διευθυνσιοδότηση μεταβαίνουμε στην αρχή του πίνακα που είναι αποθηκευμένο το σχήμα και μετά κάνουμε 18 επαναλήψεις για να λάβουμε από τον υπολογιστή τα 18 bytes που αντιστοιχούν στην εικόνα.

```
;-------------
;Receive array
;-------------
rarray
        movlw array
        movwf FSR

        movlw 0x13
        movwf cur_byte

loopr
        decfsz cur_byte,f
        goto contr
        goto endr

contr
        call receive

        movwf INDF
        incf FSR,f
        goto loopr
endr
        return
```

Η λειτουργία αυτής της διαδικασίας είναι πολύ απλή. Το μόνο που κάνει είναι να στέλνει ένα ζευγάρι χαρακτήρων που αντιστοιχούν στις συντεταγμένες του σημείου. Η αποκωδικοποίηση γίνεται στον υπολογιστή.

```
;--------------------------
;Send the coordinates to PC
;--------------------------
sendcords
        movf cord_x,w
        addlw 0x30
        call send

        movlw 0x2D
        call send

        movf cord_y,w
        addlw 0x30
        call send
        return
```

Basic Procedures

Αρχικά θα πρέπει να κληθεί η διαδικασία recognize, αυτή αφού φωνάξει την διαδικασία findfp θα ξέρει που θα βρίσκεται το πρώτο “έγχρωμο” pixel του πίνακα. Θα καλεί ενημερώνει τον πίνακα visited καλώντας την διαδικασία visited_map και θα ελέγχει τα γειτονικά καλώντας την διαδικασία check_neighbors. Θα μεταφέρει τον νέο προς έλεγχο στοιχείο στον δείκτη cur_point και θα επαναλαμβάνει αυτές τις ενέργειες μέχρι να λάβει τον ειδικό κωδικό 0xFF ως επόμενο στοιχείο, το οποίο στην πραγματικότητα δεν αντιστοιχεί σε κάποιο σημείο του πίνακα.

```
;---------------
;Recognise Shape
;---------------
recognise
       ;search for first pixel
       call findfp

       movf startpx,w
       movwf cur_point

       call fcbyte
       call fcbit

       movlw 0x00
       movwf newflag
       movwf oldflag
       movwf radius
looprec
       incf radius, f
       movf oldflag, w
       sublw 0x20          ;C
       btfss STATUS, Z
              goto checkend

       movf newflag, w
       sublw 0x08          ;E
       btfss STATUS, Z
              goto checkend

       movlw 0x1
       addwf cur_point,w
       movwf pointB
checkend
       call visited_map
       call check_neighbors

       movf cur_point,w
       sublw 0xFF
       btfsc STATUS,Z
              goto endrec

       movf shape_mask,w
       IORWF shape_rec,w
       movwf shape_rec

       goto looprec

endrec
       return
```

Η διαδικασία αυτή απλά σαρώνει από την αρχή του πίνακα μέχρι να βρει το πρώτο “μαυρισμένο” pixel. Η λογική είναι ότι παίρνει τα bytes και τα ολισθαίνει προς τα δεξιά μέχρι να βρει τον πρώτο άσσο.

```
;--------------
;Find 1st pixel
;--------------
findfp
              movlw 0xD
              movwf cur_point
loopf
              call fcbyte
              call fcbit

                                        ;krata ta backup
              movf cur_byte,w
              movwf temp2
              movf cur_bit,w
              movwf temp3
                                        ;Indirect addressing
              movlw array
              addwf cur_byte,w
              movwf FSR
              movf INDF,w
              movwf temp
                                        ;vres to pixel steile sto temp
dof
              decfsz cur_bit,f
                     goto dof2
              goto rfff
dof2
              bcf STATUS,C
              rrf temp,f
              goto dof
rfff
              bcf STATUS,C
              rrf temp,f
                                        ;elegxe an einai mhden
              btfsc STATUS,C
                     goto not_clearf
                                        ;epanefere tis times twn
                                        ;byte,bit

              movf temp2,w
              movwf cur_byte
              movf temp3,w
              movwf cur_bit
              call visited_map
                                        ;pame sto epomeno pixel
              incf cur_point,f
              movf cur_point,w
                                        ;elegxe an einai to teleftaio
              sublw 0x82 ;130
              btfss STATUS,C
                     goto endf
              goto loopf
not_clearf
              movf cur_point,w
              movwf startpx
endf
              return
```

Εδώ απλά ενημερώνει την αντίστοιχη θέση του cur_point στον πίνακα visited. Βρίσκει το byte που πρέπει και μετά απλά κάνει OR το αντίστοιχο bit.

```
;----------------------
;Change the visited map
;----------------------
visited_map
              movf cur_bit,w
              movwf temp_v_bit

              bsf STATUS,C
              movlw 0x00
              movwf temp
dof3
              rlf temp,f
              BCF STATUS,C
              decfsz temp_v_bit,f
                     goto dof3

              movlw visited
              addwf cur_byte,w
              movwf FSR
              movf INDF,w

              IORWF temp,w
              movwf INDF
       return
```

Εδώ υλοποιείται ο αλγόριθμος που αναλύθηκε παραπάνω για την εύρεση των γειτονικών στοιχείων.

```
;-----------------------------------------------
;Checks all the neighbours of the current bit
;-----------------------------------------------
check_neighbors
       movf cur_point,w
       movwf temp
       movwf p

a      ;check_point = p+1
       movlw 0x80
       movwf shape_mask
       movf p,w
       addlw 1
       movwf cur_point

       call fcbyte
       call fcbit
       call bbit
       decfsz the_bit
              goto b_path

fa     call isvisited
       ;call visited_map
       decfsz visit_flag
              goto next

b_path ;check_point = p+13
       movlw 0x40
       movwf shape_mask
       movf p,w
       addlw 0xD
       movwf cur_point

       call fcbyte
       call fcbit
       call bbit
       decfsz the_bit
              goto c

fb
       call isvisited
       decfsz visit_flag
              goto next

c      ;check_point = p+12
       movlw 0x20
       movwf shape_mask
       movf p,w
       addlw 0xC
       movwf cur_point

       call fcbyte
       call fcbit
       call bbit
       decfsz the_bit
              goto d

fc     call isvisited
       decfsz visit_flag
              goto next
```

```
d       ;check_point = p+11
        movlw 0x10
        movwf shape_mask
        movf p,w
        addlw 0xB
        movwf cur_point

        call fcbyte
        call fcbit
        call bbit
        decfsz the_bit
               goto e
fd      call isvisited
        decfsz visit_flag
        goto next

e       ;check_point = p-1
        movlw 0x08
        movwf shape_mask
        movlw 0x01
        subwf p,w
        movwf cur_point

        call fcbyte
        call fcbit
        call bbit
        decfsz the_bit
               goto f

fe
        call isvisited
        decfsz visit_flag
               goto next
f       ;check_point = p-13
        movlw 0x04
        movwf shape_mask
        movlw 0xD
        subwf p,w
        movwf cur_point

        call fcbyte
        call fcbit
        call bbit
        decfsz the_bit
               goto g
ff
        call isvisited
        decfsz visit_flag
               goto next

g       ;check_point = p-12
        movlw 0x02

        movwf cur_point
        movf p,w
        movwf endpx

        return
```

```
        movwf shape_mask

        movlw 0xC
        subwf p,w
        movwf cur_point

        call fcbyte
        call fcbit
        call bbit
        decfsz the_bit
               goto h

fg      call isvisited
        decfsz visit_flag
               goto next

h       ;check_point = p-11
        movlw 0x01
        movwf shape_mask

        movlw 0xB
        subwf p,w
        movwf cur_point

        call fcbyte
        call fcbit
        call bbit
        decfsz the_bit
               goto fn

fh      call isvisited
        decfsz visit_flag
               goto next
        goto end_checks

fn
        movlw 0x00
        movwf shape_mask
        goto end_checks
next
        ;oldflag=newflag
        ;newflag = shape_mask

        movf newflag, w
        movwf oldflag
        movf shape_mask, w
        movwf newflag

        return
end_checks
        movlw 0xFF
```

Η διαδικασία στέλνει το startpx – πάνω αριστερά γωνιακό στοιχείο – και το pointB – κάτω δεξιά γωνιακό στοιχείο.

```
;-------------------------------------------
;Find the other 2 coordinates of the rectangle
;-------------------------------------------
findRect
       movf startpx,w
       movwf cur_point
       call findcords
       call sendcords

       movlw 0x2D ;-
       call send

       movf pointB,w
       movwf cur_point
       call findcords
       call sendcords
return
```

Η διαδικασία findRad επιστρέφει την ακτίνα και το κέντρο του κύκλου. Για την ακτίνα χρησιμοποιούμε τον γνωστό τύπο από την γεωμετρία.: $Circumference = 2 \cdot \pi \cdot radius$. Για το κέντρο προχωράμε από ένα γωνιακό στοιχείο κατά radius απόσταση προς το κέντρο του. Τέλος στέλνονται τα παραπάνω δεδομένα.

```
;-------------------------------
;Find Circle's Radius and Centre
;-------------------------------
findRad
       movf radius, w
       movwf ACC0
       call divide12
       bsf STATUS, C
       rlf ACC0,w
       movwf radius
       addlw 0x30
       movwf ACC0

       movf ACC0,w
       call send

       movlw 0x2D ;-
       call send

       incf radius, w
       movwf ACC0
       call multiply12
       movf startpx, w
       addwf ACC0, w
       movwf cur_point
       call findcords
       call sendcords
return
```

Pixel Oriented Procedures

Για να βρούμε το offset του byte στο οποίο εργαζόμαστε αρκεί να διαιρέσουμε το cur_point με το 8. Δηλαδή 3 ολισθήσεις προς τα δεξιά. Και επιπλέον εύκολα με έμμεση διευθυνσιοδότηση βρίσκουμε και την τιμή του cur_byte.

```
;-----------------
;Find current byte
;-----------------
fcbyte
       movf cur_point,w
       movwf temp
       bcf STATUS,C
       RRF temp,f
       bcf STATUS,C
       RRF temp,f
       bcf STATUS,C
       RRF temp,w
       MOVWF cur_byte

       movlw array
       addwf cur_byte,w
       movwf FSR
       movf INDF,w
       movwf the_byte
       return
```

Αντίστοιχα για να βρούμε το bit στο οποίο εργαζόμαστε θα πρέπει να πολλαπλασιάσουμε με το 8 ώστε να μεταφερθούμε στην 8άδα η οποία αντιστοιχεί στο cur_byte και μετά αρκεί να αφαιρέσουμε από το cur_point. Τέλος επειδή αριθμούμε ανάποδα τα bits αφαιρούμε από το 8.

```
;-----------------
;Find current bit
;-----------------
fcbit
       movf cur_byte,w
       movwf temp
       bcf STATUS,C
       RLF temp,f
       bcf STATUS,C
       RLF temp,f
       bcf STATUS,C
       RLF temp,w

       subwf cur_point,w
       SUBLW 0x8 ;(8+1)
       MOVWF cur_bit

       Return
```

Για την εύρεση της τιμής του cur_bit αρκεί να πάρουμε την τιμή του the_byte και να την ολισθήσουμε τόσες φορές όσες αναφέρει το cur_bit (απαριθμητής). Στο τέλος θα έχουμε στον STATUS register στο πεδίο του C, το κρατούμενο εξόδου του ολισθητή. Λόγω του RISC συνόλου εντολών πολλές φορές είναι αναπόφευκτα οι αναπηδήσεις μέσα στον κώδικα.

```
;-------------------------------------------------
;stores to the_bit register the current_bit value
;-------------------------------------------------
bbit
       movf cur_bit,w
       movwf temp_bit

       movf the_byte,w
       movwf temp_byte
dob
       decfsz temp_bit,f
              goto dob2
       goto rfb
dob2
       bcf STATUS,C
        rrf temp_byte,f
       goto dob
rfb
       bcf STATUS,C
       rrf temp_byte,f
       movlw 0x00
       btfsc STATUS,C
              movlw 0x01
       movwf the_bit
       return
```

Αυτή η διαδικασία ανατρέχει στον πίνακα visited και ελέγχει αν το cur_bit το έχει επισκεφτεί ο αλγόριθμος. Αρχικά ανακτάται το byte στο οποίο βρίσκεται αυτό bit και μετά με αντίστοιχες ολισθήσεις παίρνουμε την τιμή του.

```
;-----------------------------------------
;Checks if the current bit has been visited?
;-----------------------------------------
isvisited

       movlw visited
       addwf cur_byte,w
       movwf FSR
       movf INDF,w
       movwf temp_v

       movf cur_bit,w
       movwf temp_isv_bit
dov
       decfsz temp_isv_bit,f
              goto dov2
       goto rfv
dov2
       bcf STATUS,C
        rrf temp_v,f
       goto dov

rfv
       bcf STATUS,C
       rrf temp_v,f
       movlw 0x00
       btfsc STATUS,C
              movlw 0x01
       movwf visit_flag
       return
```

Εδώ γίνεται η αντιστοιχία του σημείου από την θέση που είναι αποθηκευμένο στην μνήμη, στην πραγματική του θέση στον πίνακα 10x10. Η μετατροπή θα γίνει με διαίρεση με το 12_H (πλήθος bytes) για να βρεθεί η κάθετη συντεταγμένη. Ύστερα αυτή πολλαπλασιαζόμενη με το 12 για να μεταφερθούμε στην y γραμμή και αφαίρεση του αποτελέσματος από το cur_point μας μεταφέρει και στην οριζόντια συντεταγμένη.

```
;---------------------------------------
;find the coordinates of the current point
;---------------------------------------
findcords
       movf cur_point,w
       movwf ACC0
       call divide12
       movf ACC0,w
       movwf cord_y

       movf cord_y,w
       movwf ACC0
       call multiply12
       movf ACC0,w
       subwf cur_point,w
       movwf cord_x

       return
```

Utilities

Η ακόλουθη διαδικασία απλά καθαρίζει όλους του καταχωρητές γενικής χρήσης που αφορούν τον χρήστη.

```
;------------
;Clear memory
;------------
clear_mem
              movlw 0x20
              movwf FSR

              movlw 0x4F
              movwf cur_byte

loopcm        decfsz cur_byte,f
                     goto contcm
              goto endcm

contcm
              movlw 0x00
              movwf INDF
              goto loopcm
endcm
       return
```

Επειδή το σύνολο εντολών του controller είναι αρκετά περιορισμένο και ως εκ τούτου δεν διαθέτει εντολές πολλαπλασιασμού και διαίρεσης. Παρακάτω υλοποιούνται διαδικασίες πολλαπλασιασμού και διαίρεσης με το 12. Οι παρακάτω κώδικες είναι computer-generated από την σελίδα του Nikolai Golovchenko.

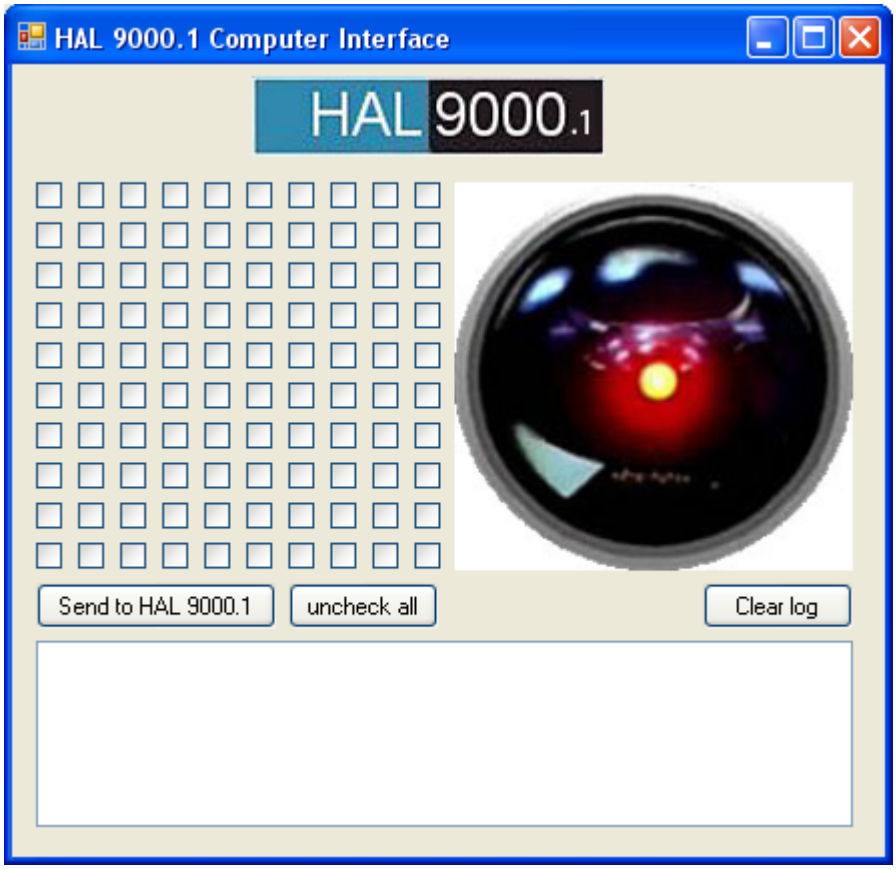
```
;--------------------
;Multiplication by 12
;--------------------
multiply12
              bcf STATUS,C
              rlf    ACC0, f
              clrf   ACC1
              rlf    ACC1, f
              rlf    ACC0, f
              rlf    ACC1, f
                                  ;copy accumulator to temporary
              movf   ACC1, w
              movwf  TEMP1
              movf   ACC0, w
              movwf  TEMP0
                                  ;shift temporary left 1 times
              bcf STATUS,C
              rlf    TEMP0, f
              rlf    TEMP1, f

                                  ;add temporary to accumulator
              movf   TEMP0, w
              addwf  ACC0, f
              movf   TEMP1, w
              btfsc STATUS,C
              incfsz TEMP1, w
              addwf  ACC1, f
       return

;--------------
;Division by 12
;--------------
divide12
              movf   ACC0, w
                                  ;shift accumulator right 2 times
              bcf STATUS,C
              rrf    ACC0, f
              bcf STATUS,C
              rrf    ACC0, f
                                  ;add temporary to accumulator
              addwf  ACC0, f
                                  ;shift accumulator right 2 times
              rrf    ACC0, f
              bcf STATUS,C
              rrf    ACC0, f
                                  ;add temporary to accumulator
              addwf  ACC0, f
                                  ;shift accumulator right 2 times
              rrf    ACC0, f
              bcf STATUS,C
              rrf    ACC0, f
                                  ;add temporary to accumulator
              addwf  ACC0, f
                                  ;shift accumulator right 4 times
              swapf  ACC0, w
              andlw  0x0F
              movwf  ACC0
              btfsc STATUS,C
              bsf    ACC0, 4
       return
```

Περιγραφή Software διασύνδεσης PC/PIC

Σκοπός της εφαρμογής είναι την άμεση αποστολή δεδομένων προς τον controller για την επεξεργασία και την λήψη των αποτελεσμάτων. Αποφεύχθηκε η απλή υλοποίηση μέσω ενός terminal διότι έτσι υπήρχε δυσκολία του χρήστη για την αποστολή της εικόνας καθώς θα απαιτούσε την μετατροπή του κάθε pixel σε αντίστοιχο bit του προς αποστολή χαρακτήρα.

Η εφαρμογή υλοποιήθηκε για χάριν απλότητας και αμέσων αποτελεσμάτων σε περιβάλλον Microsoft Visual Studio 2005 Professional Edition και γλώσσα προγραμματισμού Visual Basic 2005.

Το κεντρικό – και μοναδικό – παράθυρο της εφαρμογής είναι το παρακάτω:



Τα checkboxes αναπαριστούν το πλέγμα-εικόνα που επιθυμεί ο χρήστης για να στείλει προς τον controller. Προφανώς θεωρείται πως όταν μία θέση του πλέγματος είναι σημαδεμένη τότε θεωρείται πως το αντίστοιχο εικονοστοιχείο είναι “ μαυρισμένο ”.

Για την αποστολή των δεδομένων προς τον controller θα πρέπει να πατηθεί το κουμπί “Send to HAL 9000.1”. Τότε η εφαρμογή θα μετατρέψει την δοσμένη “εικόνα” σε μια ακολουθία από δεκαεξαδικούς αριθμούς (18 στο σύνολο) και θα προσπαθήσει να επικοινωνήσει με τον controller. Σε περίπτωση αποτυχίας εμφανίζει αντίστοιχο μήνυμα λάθους. Σε περίπτωση επιτυχίας τότε στέλνει τα δεδομένα και μετά περιμένει τα αποτελέσματα.

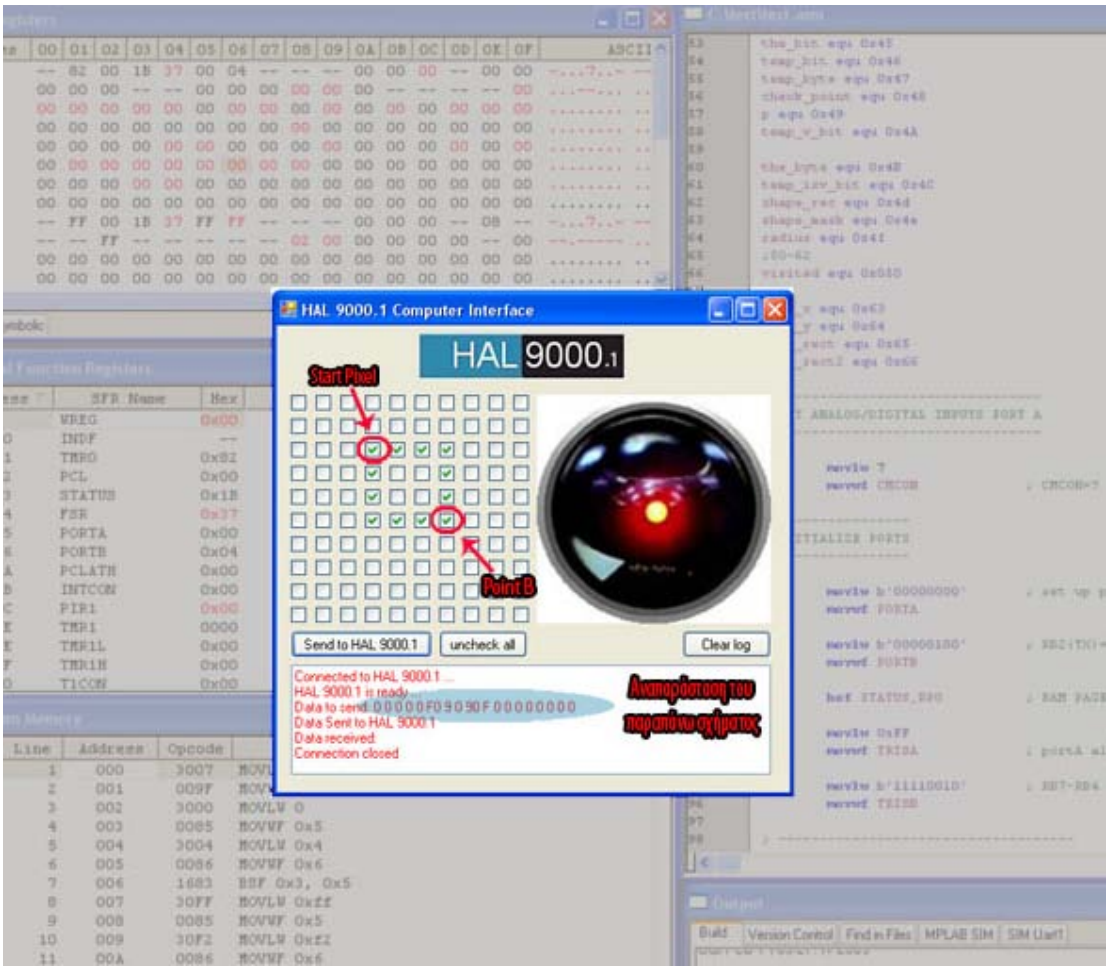
Για την ενημέρωση του χρήστη για την κατάσταση του συστήματος υπάρχει το TextBox που καταγράφει κάθε στιγμή τις παρακάτω πληροφορίες:

Μήνυμα	
`Connected to HAL 9000.1 ...`	**Έγινε επιτυχής σύνδεση με το Controller**
`HAL 9000.1 is ready ...`	**Ο controller είναι έτοιμος να δεχθεί δεδομένα**
`Data Sent to HAL 9000.1`	**Τα δεδομένα στάλθηκαν στον controller**
`Connection closed`	**Η σύνδεση έκλεισε.**

Εικονική εκτέλεση προγράμματος

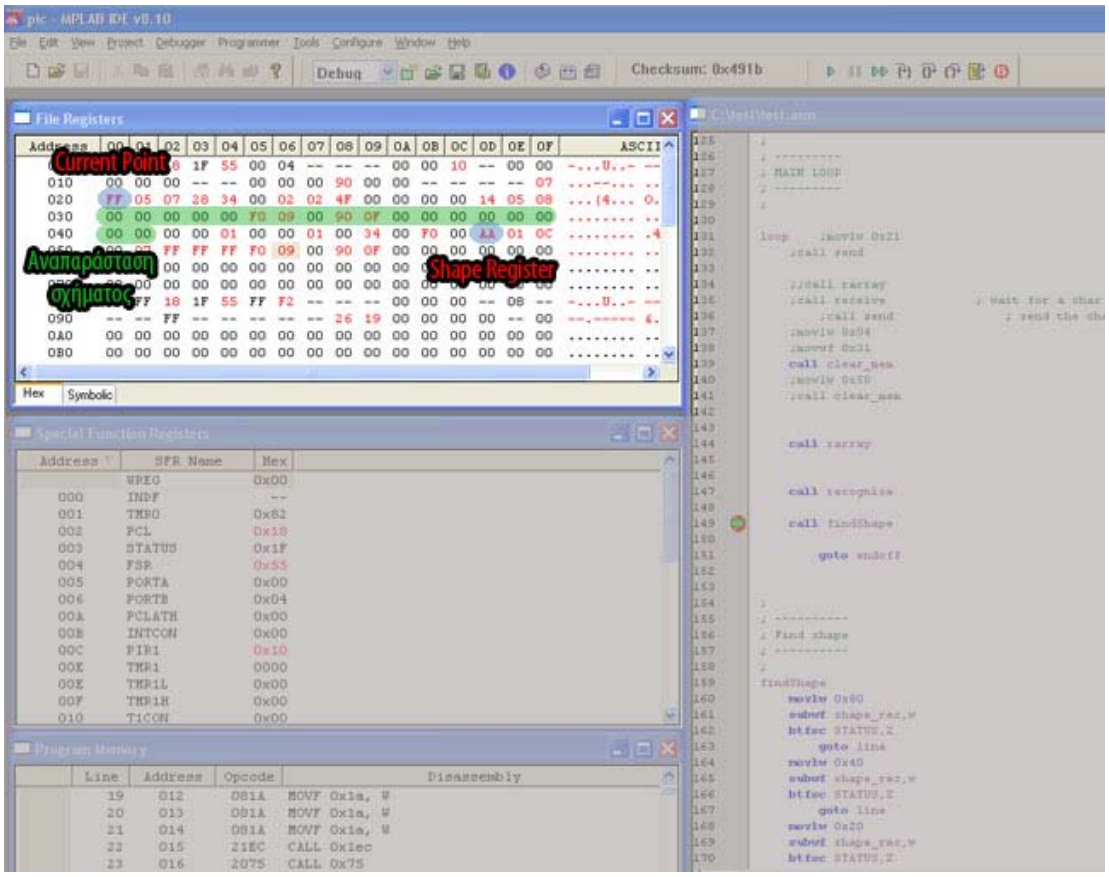
Παρακάτω παρουσιάζεται μια εκτέλεση του μικροϋπολογιστικού συστήματος στον Simulator της MicroChip MPLAB Integrated Development Environment V 8.10.00.00

Παραλαβή δεδομένων από σειριακή θύρα



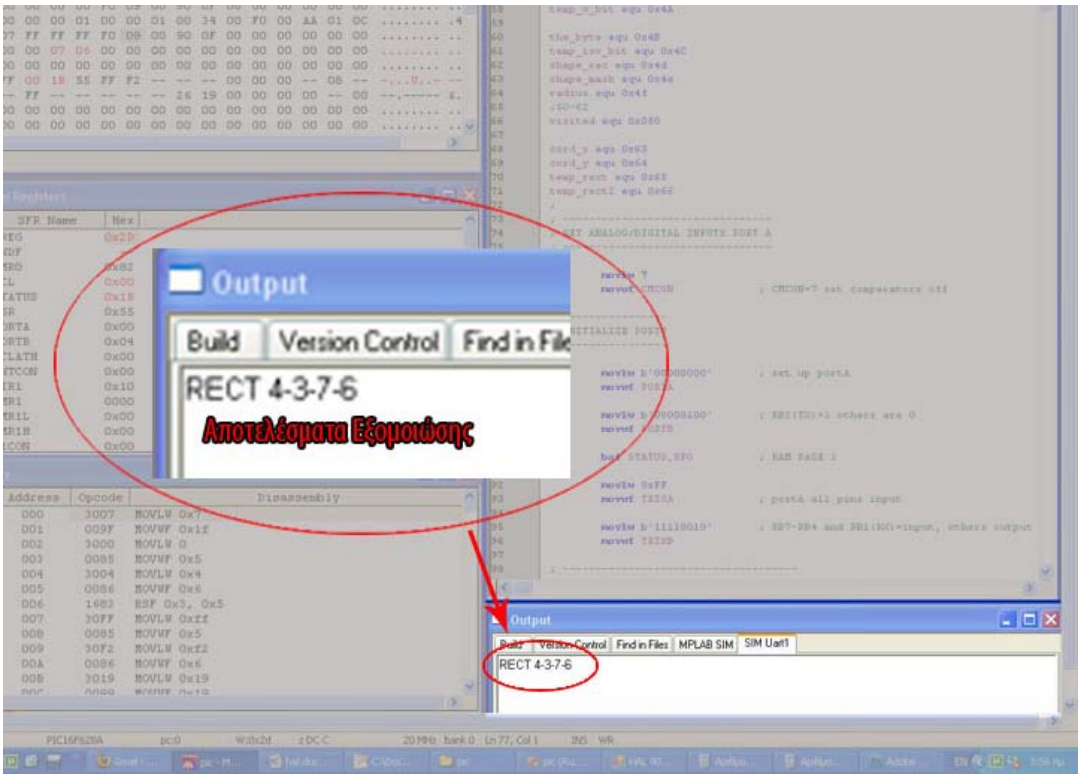
Στην παραπάνω εικόνα απεικονίζεται η αποστολή ενός αρχείου. Το συγκεκριμένο αρχείο περιέχει ένα τετράγωνο με διαγώνια στοιχεία (3,3) και το (7,6) και με γαλάζιο φόντο έχουμε την δεκαεξαδική του αναπαράσταση με την οποία θα αποσταλεί στον PIC.

Αποστολή και επεξεργασία αρχείου.



Οι θέσεις μνήμης με πράσινο φόντο είναι δεσμευμένες για την αποθήκευση του αρχείου, οπότε παρατηρούμε πως η αποστολή του αρχείου έγινε επιτυχώς. Επίσης σε αυτό το στάδιο έχει ήδη εκτελεστεί ο αλγόριθμος αναγνώρισης. Όπως παρατηρούμε ο καταχωρητής Current Point περιέχει την τιμή 0xFF, γεγονός που σημαίνει πως η σάρωση του αρχείου από τον αλγόριθμο τελείωσε. Επίσης ο καταχωρητής σχήματος (Shape Register) περιέχει την τιμή 0xAA που σημαίνει πως κατά την σάρωση ανιχνεύθηκαν τα μονοπάτια a-c-e-g μόνο και άρα πρόκειται για ένα ορθογώνιο.

Αποστολή και εμφάνιση αποτελεσμάτων.



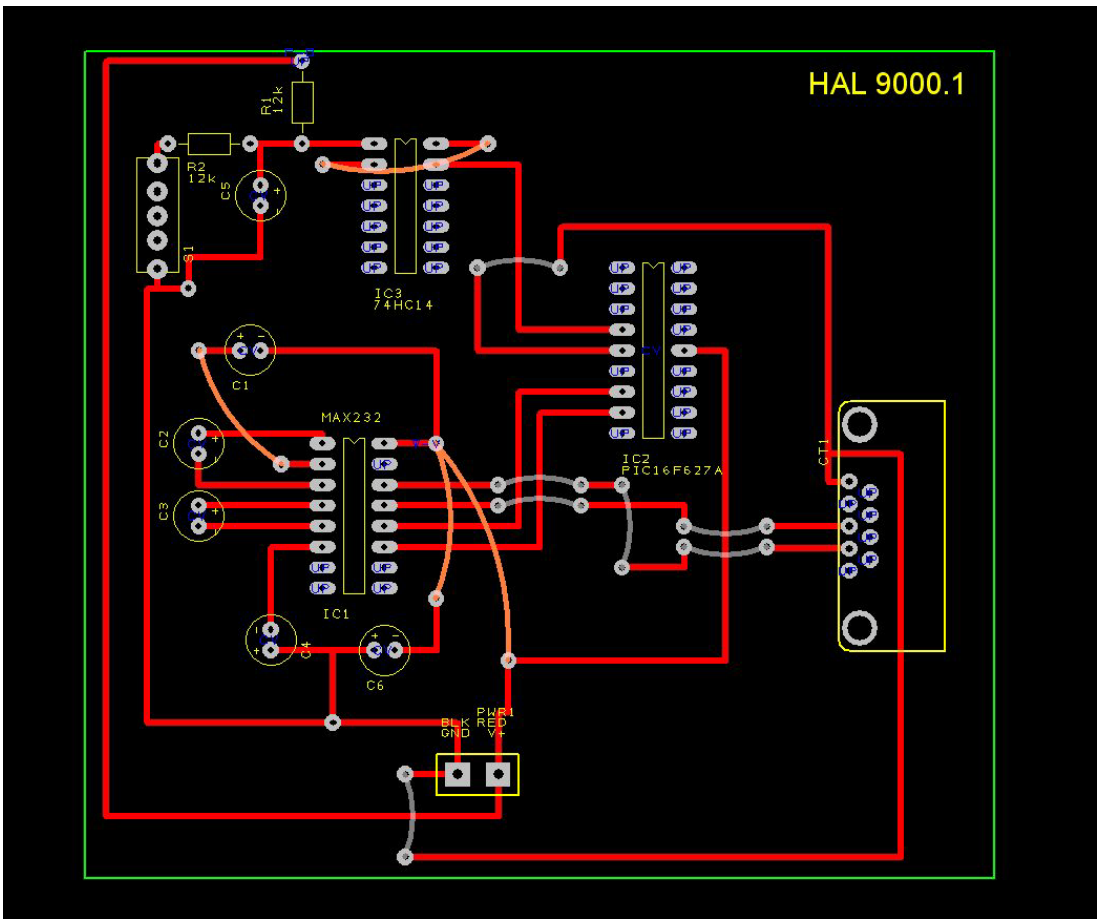
Σε αυτό το τελικό στάδιο το σύστημα προσπελαύνει τον καταχωρητή σχήματος και απέστειλε στην σειριακή έξοδο τα αντίστοιχα δεδομένα με την μορφή [ΣΧΗΜΑ]-[ΣΤΟΙΧΕΙΑ]. Στην συγκεκριμένη περίπτωση μας απέστειλε ορθά τα στοιχεία που αναμέναμε.

Περιγραφή του κυκλώματος

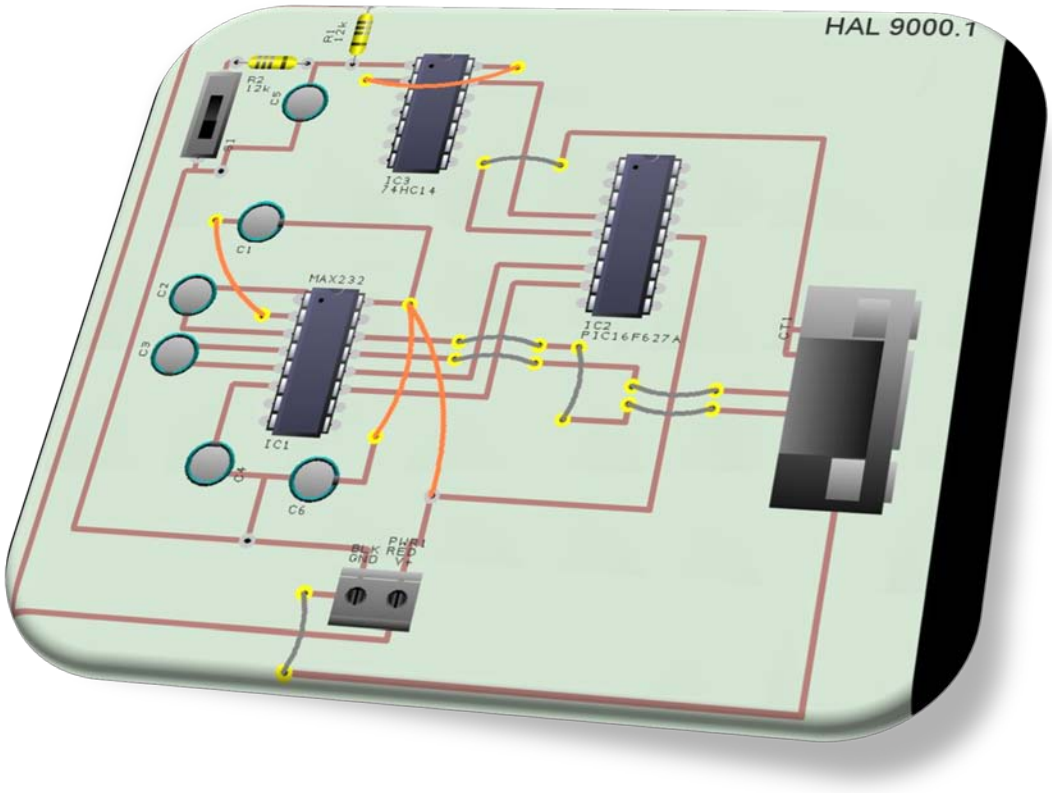
Για την υλοποίηση του συστήματος χρησιμοποιούνται τα παρακάτω υλικά και η συνδεσμολογία γίνεται όπως στο παρακάτω σχήμα.

Component	Ποσότητα	Περιγραφή
PIC16f628	1	Controller
MAX232	1	Κύκλωμα για την αποστολή δεδομένων στον υπολογιστή
74HC14	1	Schmitt *trigger* inverters
Πυκνωτές 1μF	6	Για την ομαλή φόρτιση και αποφόρτιση των ακροδεκτών
Αντιστάσεις 12KOhm	2	Για αποφυγή βραχυκυλωμάτων
Connector D9	1	Για σύνδεση σειριακού καλωδίου
Διακόπτης	1	Για επαναφορά του συστήματος

Σχηματική αναπαράσταση 1



Σχηματική αναπαράσταση 2



Παράρτημα

Περαιτέρω επεκτασιμότητα του συστήματος

Καταρχάς ας υπενθυμίσουμε στον αναγνώστη πως οποιαδήποτε γεωμετρικό σχήμα στην μνήμη μπορεί να αντιστοιχηθεί με μία συγκεκριμένη ακολουθία "μονοπατιών". Επίσης να αναφέρουμε πως ο αλγόριθμος ανίχνευσης σχημάτων εκτελεί αυτή ακριβώς την λειτουργία, δηλαδή ανιχνεύει τα μονοπάτια που του σχήματος και ενημερώνει έναν καταχωρητή.

Θεωρητικά λοιπόν με ενημέρωση περισσότερων καταχωρητών αλλά και με πολλαπλές συνθήκες ελέγχου να μετατρέψουν την πληροφορία των καταχωρητών σε κάποιο υπαρκτό γεωμετρικό σχήμα θα μπορούσαμε να ανιχνεύσουμε *οποιοδήποτε* σχήμα. Επίσης δεδομένης περισσότερης διαθέσιμης μνήμης εκτός από τον πιο εύκολο χειρισμό των pixel θα μπορούσαμε να αναγνωρίσουμε και διαφορετικά χρώματα. Έτσι υλοποιήσαμε την παρούσα λύση στο θέμα, η οποία παρόλο που δεν είναι τέλεια, μας παρέχει lossless συμπίεση στις περιπτώσεις ευθείας και τετραγώνου και μία αρκετά αξιόπιστη προσεγγιστική λύση στην περίπτωση του κύκλου.

Αντί επιλόγου

Κλείνοντας να παρατηρήσουμε πως για μία πλήρη λύση του προβλήματος, οι συντάκτες της παρούσας αναφοράς προτείνουν την χρήση ενός μικροϋπολογιστικού συστήματος αρχιτεκτονικής CISC, και με σαφώς περισσότερη μνήμη - ίσως της τάξης των KB - (η αντιστοιχία pixel ↔ bit δημιούργησε το πρόβλημα χειρισμού κάθε bit ξεχωριστά) ώστε να υπάρχει η απαραίτητη προγραμματιστική ευελιξία για την πλήρη λύση του προβλήματος.

Βιβλιογραφία

ASCII printable characters. (n.d.). Ανάκτηση Φεβρουάριος 2009, από Wikipedia: http://en.wikipedia.org/wiki/ASCII

Golovchenko, N. (n.d.). *Code Generation for Constant Multiplication/Division*. Ανάκτηση Φεβρουάριος 2009, από http://www.piclist.com/techref/piclist/codegen/constdivmul.htm

Microsoft Visual Basic .NET BHMA BHMA. Microsoft Press.

MIT - Microcomputer Project Laboratory - Spring 2009. (n.d.). Ανάκτηση Φεβρουάριος 2009, από http://web.mit.edu/6.115/www/pic.shtml

PIC Tutorial Seven - RS232. (n.d.). Ανάκτηση Ιανουάριος 2009, από http://www.winpicprog.co.uk/pic_tutorial7.htm

PIC16F628. (n.d.). Ανάκτηση Φεβρουάριος 2009, από http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010209

The Art of Assembly Language Programming. (n.d.). Ανάκτηση Φεβρουάριος 2009, από http://maven.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html

UART test program for 16F628. (n.d.). Ανάκτηση Φεβρούαριος 2009, από http://www.oz1bxm.dk/PIC/628uart.htm

Αλεξιου, Π. (2009). *Μικροεπεξεργαστές και σχεδιασμός μικρουπολογιστικών Συστημάτων.* Γκιουρδας.