ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΧΝΟΛΟΓΙΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΣΗΜΕΙΩΣΕΙΣ

Παναγιώτης Ε. Φουληράς

Διδάκτωρ Πληροφορικής Παν/μίου Λονδίνου

ΘΕΣΣΑΛΟΝΙΚΗ 1996

ΠΡΟΛΟΓΟΣ

Οι παρούσες σημειώσεις αποτελούν συρραφή προτεινομένων εργαστηριακών ασκήσεων του Εργαστηρίου του μαθήματος "Τεχνολογία Η/Υ". Σκοπό έχουν να παράσχουν πρακτική εμπειρία στους εξασκούμενους σπουδαστές του τμήματος Πληροφορικής του ΤΕΙ Θεσσαλονίκης, ώστε να είναι σε θέση:

1. Να κατανοήσουν βασικά στοιχεία από την αρχιτεκτονική μοντέρνων Η/Υ.

2. Να είναι σε θέση να προγραμματίσουν σχετικά απλές περιφερειακές συσκευές, όταν κανένα έτοιμο εργαλείο δεν είναι διαθέσιμο (π.χ. έτοιμες βιβλιοθήκες).

Η παρουσιαζόμενη θεωρία είναι επέκταση σε αντίστοιχα θέματα τα οποία ήδη παρουσιάζονται από τις σημειώσεις της Θεωρίας του μαθήματος "Τεχνολογία Η/Υ".

Το παρουσιαζόμενο υλικό σίγουρα δεν είναι πλήρες, αλλά όσο γίνεται απλούστερο, αλλά και ρεαλιστικό. Λόγω της εξέλιξης της τεχνολογίας, η περαιτέρω επέκταση και αναθεώρηση του υπάρχοντος υλικού είναι φυσικό επακόλουθο.

Για οποιεσδήποτε ελλείψεις ή λάθη φυσικά ο συγγραφέας του παρόντος είναι πλήρως υπεύθυνος. Υποδείξεις για τις σημειώσεις αυτές είναι πάντοτε ευπρόσδεκτες και μπορούν να αποστέλλονται στην παρακάτω διεύθυνση:

pef@alpha.it.teithe.gr

ΕΒΔΟΜΑΔΑ 1

Σκοπός του εργαστηρίου αυτού είναι να εφαρμόσετε τις γνώσεις σας για μετατροπή μη-προσημασμένων ακεραίων αριθμών, από το ένα αριθμητικό σύστημα στο άλλο, χρησιμοποιώντας τις αντίστοιχες δυνατότητες που σας παρέχει ο 8086.

1. Εισαγωγή

Για να μετατρέψετε έναν μη-προσημασμένο ακέραιο αριθμό από ένα αριθμητικό σύστημα με βάση Β σε ένα άλλο με βάση β, πρέπει πρώτα να γνωρίζετε το αριθμητικό σύστημα στο οποίο και πρόκειται να εργασθείτε. Εάν αυτό είναι το δεκαδικό, ο συνηθέστερος τρόπος μετατροπής, περιλαμβάνει πρώτα την μετατροπή από το σύστημα με βάση Β στο δεκαδικό και στην συνέχεια από το δεκαδικό στο σύστημα με βάση β.

Στην περίπτωση του 8086, μπορούμε να θεωρήσουμε ότι αυτός εργάζεται στο δυαδικό σύστημα, αλλά και τα παράγωγά του (δυνάμεις του 2), δηλαδή το οκταδικό και το δεκαεξαδικό. Όταν θέλουμε να μετατρέψουμε έναν αριθμό από ένα σύστημα **γνωστό** στον 8086 (π.χ. δεκαεξαδικό) σε ένα **άγνωστο** σε αυτόν (π.χ. δεκαδικό), εφαρμόζουμε την μέθοδο των διαιρέσεων, κατά την οποία εφαρμόζουμε διαδοχικές διαιρέσεις στον αριθμό που μας ενδιαφέρει, κρατώντας το εκάστοτε υπόλοιπο σαν το ψηφίο στο καινούργιο αριθμητικό σύστημα και το εκάστοτε πηλίκο για την επόμενη διαίρεση, έως ότου το τελευταίο μηδενισθεί.

Ο αριθμός στο καινούργιο σύστημα προκύπτει από την παράθεση των υπολοίπων, με **αντίστροφη σειρά** από εκείνη του υπολογισμού τους.

2. Εργαστηριακή Άσκηση

Παρακάτω βλέπετε ένα κομμάτι προγράμματος σε 8086 Assembly, που έχει γραφεί με την μορφή διαδικασίας. Όπως φαίνεται και από τον κώδικα, ένας μη-προσημασμένος ακέραιος αριθμός στο δεκαεξαδικό σύστημα περνάει ως παράμετρος στην διαδικασία αυτή μέσω του καταχωρητή DX. Η διαδικασία παίρνει αυτόν τον αριθμό και τον μετατρέπει στο δεκαδικό σύστημα, υπολογίζοντας διαδοχικά υπόλοιπα, τα οποία τα τοποθετεί κάθε φορά στην Σωρό.

Το πλεονέκτημα της χρήσης της Σωρού γίνεται κατανοητό, εάν θυμηθείτε ότι τα ψηφία που αντιστοιχούν στα υπόλοιπα πρέπει να τυπωθούν με την **αντίστροφη** σειρά του υπολογισμού τους.

Θυμηθείτε ότι **XOR** ενός καταχωρητή με τον εαυτό του, τον μηδενίζει και ότι η **DIV <καταχωρητής>** διαιρεί το περιεχόμενο του **AX** με τον καταχωρητή αυτόν, τοποθετώντας το πηλίκο της διαιρέσεως στον **AX** και το υπόλοιπο στον **DX**.

Όπως βλέπετε, αυτή η διαδικασία μετατρέπει έναν δεκαεξαδικό αριθμό στο δεκαδικό σύστημα και καλεί μία άλλη διαδικασία, που δεν έχει γραφεί για να τυπώσει ένα ψηφίο κάθε φορά. Εσείς πρέπει να γράψετε ένα πρόγραμμα για τα παρακάτω:

1. Να γράψετε (όχι κατ' ανάγκην σε μορφή διαδικασίας) την διαδικασία εκτύπωσης κάθε ψηφίου στην οθόνη.

2. Να θεωρήσετε ότι η διαδικασία μετατροπής, παίρνει ακόμη μία παράμετρο (το πώς αφήνεται στην κρίση σας), που της επιτρέπει την μετατροπή ενός αριθμού από το δεκαεξαδικό σε οποιοδήποτε αριθμητικό σύστημα μεταξύ του δυαδικού και του δεκαδικού.

```
; Αυτή η διαδικασία δέχεται έναν μη-προσημασμένο ακέραιο δεκαεξαδικό αριθμό,
; τον μετατρέπει στο δεκαδικό σύστημα και τον τυπώνει στην οθόνη
```

```
; Παράμετροι εισόδου:  DX = ο δεκαεξαδικός αριθμός
; Χρησιμοποιεί την διαδικασία εκτύπωσης        WRITE_DIGIT
; -------------------------------------------------------------------------------
CONVERT_NUM     PROC   NEAR
        push    ax                      ; Αποθήκευσε καταχωρητές που
        push    cx                      ; χρησιμοποιούνται σε αυτήν την διαδικασία
        push    dx
        push    si
        mov     ax, dx
        mov     si, 10                  ; SI κρατάει το 10 για διαίρεση με AX
        xor     cx, cx                  ; CX μετράει τον αριθμό των ψηφίων που
                                        ; θα μπουν στην Σωρό
NON_ZERO:
        xor     dx, dx                  ; DX = 0
        div     si                      ; AX / SI. Πηλίκο στον AX, υπόλοιπο DX
        push    dx                      ; Σπρώξε ψηφίο στην Σωρό
        inc     cx                      ; Αύξησε μετρητή ψηφίων
        or      ax, ax                  ; Πηλίκο είναι 0;
        jne     NON_ZERO                ; Όχι ακόμα, συνέχισε
WRITE_LOOP:
        pop     dx                      ; Βγάλε ψηφία από Σωρό (αντίστροφα!)
        call    WRITE_DIGIT     ; Κάποια διαδικασία εκτύπωσης στην οθόνη
        loop    WRITE_LOOP      ; Εξαρτάται από τον μηδενισμό η όχι του CX
LAST_PART:                              ; Έξοδος
        pop     si
        pop     dx
        pop     cx
        pop     ax
        ret
CONVERT_NUM     ENDP
```

ΕΒΔΟΜΑΔΑ 2

Σκοπός του εργαστηρίου αυτού είναι να θυμηθείτε την πολύ σημαντική χρήση των Διαδικασιών (Procedures) ή υπορουτινών, στον 8086. Όπως βλέπετε από το παρακάτω πρόγραμμα, σε περίπτωση χρήσης διαδικασιών, το κυρίως πρόγραμμα από όπου καλούνται οι υπόλοιπες διαδικασίες δηλώνεται και αυτό σαν μία διαδικασία. Η διαφορά έγκειται στο ότι υπάρχει μία οδηγία (directive) προς τον Assembler στο τέλος, που έχει την μορφή: **end <Όνομα Κυρίου Προγράμματος>.**

Στο πρόγραμμα αυτό παρατηρείστε ότι υπάρχει πέρασμα παραμέτρων (και αποτελεσμάτων) από μία διαδικασία στην άλλη, μέσω:

1. Συγκεκριμένης διεύθυνσης μνήμης (**Error_Flag**).

2. Της Σωρού, για να τυπωθεί από την ίδια διαδικασία, ανάλογα με την επιθυμία μας, το μήνυμα **Message** ή το μήνυμα **Error_mess**.

Όπως βλέπετε, αυτό το πρόγραμμα διαβάζει έναν θετικό, μονοψήφιο, ακέραιο αριθμό από το πληκτρολόγιο, ελέγχοντας για το εάν αυτός είναι μεταξύ 0 και 9. Εάν δεν είναι τότε τυπώνει ένα μήνυμα λάθους και βάζει τον αριθμό 1 στο byte **Error_Flag**. Αυτό που εσείς πρέπει να κάνετε είναι να πληκτρολογήσετε αυτό το πρόγραμμα, να το περάσετε από τον Assembler και να το τρέξετε δοκιμαστικά.

Εάν δουλεύει σύμφωνα με την παραπάνω περιγραφή, θα πρέπει να το επεκτείνετε (στο σημείο με τους αστερίσκους), ως εξής:

1. Όταν ο αριθμός είναι λανθασμένος, δεν αρκεί το πρόγραμμα να τυπώνει ένα μήνυμα λάθους και να σταματάει. Αντιθέτως θα πρέπει να ξαναζητάει σε αυτήν την περίπτωση την επαναεισαγωγή του αριθμού αυτού.

2. Επαναλάβετε την παραπάνω διαδικασία όχι για έναν, αλλά για δύο αριθμούς, τους οποίους θα πρέπει να προσθέσετε.

3. Τυπώστε το άθροισμα στην οθόνη. Καλέστε μία διαδικασία για τον σκοπό αυτό, που θα χρησιμοποιεί κάποια από τα **INT 21h** του DOS. Η δύσκολη περίπτωση προέρχεται από το γεγονός ότι η εισαγωγή και η εκτύπωση του αποτελέσματος πρέπει να είναι σε μορφή ASCII, ενώ η άθροιση γίνεται στο δεκαεξαδικό σύστημα. Ιδίως το αποτέλεσμα μπορεί να είναι **δύο** ψηφία (9+9=18).

4. Η διαδικασία που εκτελεί την πρόσθεση των δύο αριθμών θα πρέπει να τους διαβάζει σαν παραμέτρους μέσω της Σωρού (το πώς επιστρέφει το αποτέλεσμα αφήνεται στην διάθεσή σας).

```
soros segment para stack 'stack'
        DB  200h DUP(?)
soros ends

data  segment
        Message         db      'Dwste ton Arithmo n, me 0 <= n <= 9 : ',13, 10, '$'
        Error_mess      db      '  -> Invalid Number. Try again! ', 13, 10,'$'
        First_num       db      ?               ; Prwtos arithmos
        Second_num      db      ?               ; Deyteros arithmos
        Error_Flag      db      ?               ; 0 = No Error, 1 = Error
data  ends

kodikas segment
        assume ds:data, cs:kodikas, ss:soros

MAIN_PROG   PROC  NEAR
        mov   ax, data
        mov   ds, ax                            ; DS δείχνει στην αρχή του Data Segment
        mov   bx, offset Message
        push  bx                                ; Στην Σωρό η διεύθυνση του 1ου μηνύματος
        call    DISP_MESS
        pop   bx                                ; Η Σωρός επανέρχεται στην αρχική κατάσταση
```

```
        call    ENTER_NUM               ; Εάν σωστός αριθμός στον AL. Error_Flag
                                        ; δείχνει κατάσταση
        mov   ah, 4Ch
        int     21h
MAIN_PROG           endp

;-------------------------------------------------------------------------------------------
;Procedure: Display Message - Δεχεται παράμετρο από ένα push ατην Σωρό
;-------------------------------------------------------------------------------------------
DISP_MESS   PROC  NEAR
        push  ax
        pushf
        push  dx
        push  bp
        mov   bp, sp
        mov   ax, [bp+10]               ; Στον AX τώρα αντίγραφο της παραμέτρου
        mov  dx, ax
        mov  ah, 9h                     ; DOS Display message στον DX
        int     21h
        pop   bp
        pop   dx
        popf
        pop   ax
        ret
DISP_MESS           endp

;-------------------------------------------------------------------------------------------
; Procedure: Εισαγωγή Αριθμού n, με 0 ≤ n ≤ 9. Αποτέλεσμα στον AL
;-------------------------------------------------------------------------------------------
ENTER_NUM   PROC  NEAR
        push  bx
        push  dx
        mov   ah, 01h                   ; Είσοδος χαρακτήρα από πληκτρολόγιο
        int     21h                     ; ASCII κωδικός στον AL
        cmp   al, '0'
        jge     first_ok
```

```
;
; else
;
        mov   bx, offset Error_mess
        push  bx
        call  DISP_MESS
        pop   bx
        mov   dl, 1                          ; Υποδεικνύοντας κατάσταση λάθους
        mov   Error_Flag, dl
        jmp   end_enter_num

first_ok:
        cmp   al, '9'
        jle   second_ok
;
; else
;
        mov   bx, offset Error_mess
        push  bx
        call  DISP_MESS
        pop   bx
        mov   dl, 1                          ; Υποδεικνύοντας κατάσταση λάθους
        mov   Error_Flag, dl
        jmp   end_enter_num

second_ok:
        mov   dl, 0
        mov   Error_Flag, dl                 ; Υποδεικνύοντας ότι δεν υπάρχει λάθος

end_enter_num:
        pop   dx
        pop   bx
        ret
ENTER_NUM           endp

kodikas          ends
```

```
end     MAIN_PROG
```

Προαπαιτούμενα

Βασική γνώση της Assembly 8086. Ένας 8086 Assembler (MASM ή TASM). Κατάλογος κωδικών ASCII και Διακοπών MS-DOS. Σημειώστε ότι το παραπάνω πρόγραμμα έχει γραφεί για μορφή .EXE και όχι .COM όπως έχετε ίσως συνηθίσει. Σε αυτήν την περίπτωση (δημιουργία αρχείου .COM) θυμηθείτε να αλλάξετε και να προσθέσετε κάποια INT 21h σε INT 20h.

ΕΒΔΟΜΑΔΑ 3

Σκοπός του εργαστηρίου αυτού είναι να μάθετε να χρησιμοποιείτε το περιφερειακό ολοκληρωμένο 8253/8254, που υπάρχει σε κάθε IBM συμβατό Η/Υ.

1. Εισαγωγή

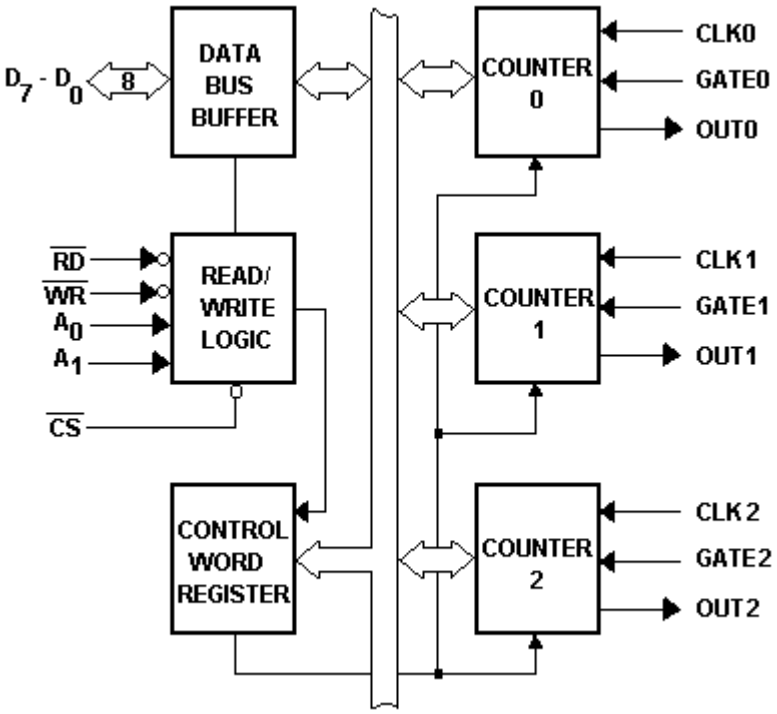
Οι πιο πολλοί μικροϋπολογιστές περιλαμβάνουν ένα ειδικό ολοκληρωμένο, το οποίο είναι σε θέση να προγραμματισθεί για να χρησιμοποιηθεί ως μετρητής σε πολλές εφαρμογές. Εάν, επιπλέον, είναι δυνατόν να προγραμματισθεί ώστε να μετράει σύμφωνα με τους παλμούς του ρολογιού του συστήματος, τότε μπορεί να παίξει και τον ρόλο του χρονομέτρη. Ένα τέτοιο ολοκληρωμένο είναι το 8254, το οποίο είναι μία ελαφρά πιο εξελιγμένη έκδοση του 8253 (και τα δύο της INTEL), που χρησιμοποιήθηκε στον προσωπικό Η/Υ της IBM.

Όπως για όλα τα περιφερειακά ολοκληρωμένα, που υπάρχουν σε έναν συμβατό IBM Η/Υ, έτσι και για το 8253/4, θα πρέπει να γνωρίζετε πλήρως τον τρόπο λειτουργίας του, προγραμματισμού του, καθώς και το πώς έχει ολοκληρωθεί στο υπολογιστικό σύστημα, προκειμένου να είστε (στην συνέχεια) σε θέση να το χρησιμοποιήσετε σωστά για τους σκοπούς σας.

Τα 8253 και 8254 είναι παρόμοια (μάλιστα συμβατά και ως προς τους ακροδέκτες τους), με τις παρακάτω κύριες διαφορές:

1. Η μέγιστη συχνότητα λειτουργίας του 8253 είναι 2,6 MHz, ενώ του 8254, 8-10 MHz.

2. Το 8254 έχει το επιπλέον χαρακτηριστικό *Read-back (Ανάγνωσης προς τα Πίσω),* που επιτρέπει την αποθήκευση των τρεχόντων περιεχομένων των καταχωρητών του, για την ανάγνωσή τους.



Σχ. 4.1 Εσωτερική Αρχιτεκτονική του 8253/4

Η εσωτερική αρχιτεκτονική του 8253/4 φαίνεται στο Σχ. 4.1. Όπως βλέπετε, υπάρχουν τρεις καταχωρητές των 16 bits, που παίζουν τον ρόλο μετρητή (Counter 0, 1 και 2). Κάθε ένας από αυτούς μπορεί να φορτωθεί με μία τιμή, να τον ξεκινήσετε ώστε να μετράει και να τον σταματήσετε, με εντολές από το πρόγραμμά σας. Μπορείτε να τροφοδοτήσετε τις εισόδους **CLK** με οποιοδήποτε σήμα μεταξύ 0 Hz (DC) και 8 MHz, και να ξεκινήσετε και να σταματήσετε τους μετρητές με εξωτερικό σήμα (**GATE**). Οι μετρητές αυτοί δίνουν το αποτέλεσμα-συχνότητα ή παλμό στην έξοδο OUT.

Όταν ξεκινάτε έναν Η/Υ συμβατό με IBM, το 8253 βρίσκεται σε απροσδιόριστη κατάσταση. Το βασικό πρόγραμμα που εκτελείται αυτόματα σε αυτήν την περίπτωση, ανάμεσα στα άλλα αρχικοποιεί και το 8253. Το τελευταίο είναι έτσι συνδεδεμένο, ώστε ο μετρητής (ή κανάλι) 0 να χρησιμοποιείται από το ρολόι του συστήματος, ο μετρητής 1 να ελέγχει την ανανέωση της δυναμικής μνήμης (μέσω του DMA) και ο μετρητής 2 να είναι συνδεδεμένος με το μεγάφωνο του Η/Υ για παραγωγή ήχων κατάλληλης συχνότητας.

Ειδικά στον συγκεκριμένο Η/Υ, η συχνότητα λειτουργίας του 8253 είναι 1,19318 MHz. Ανάλογα με την τιμή Χ που έχει τοποθετηθεί σε κάθε μετρητή, ο τελευταίος δημιουργεί στην έξοδό του, 1.193.180/Χ παλμούς το δευτερόλεπτο. Τοποθετώντας την τιμή Χ=0 είναι το ίδιο με το να έχει τοποθετηθεί η τιμή Χ=65.536.

Αφού ο μετρητής 0 έχει την τιμή Χ=0, διαιρώντας την συχνότητα με το 65536, προκύπτει ότι μία μονάδα του μετρητή 0 αντιστοιχεί σε 18,2065 Hz ή έναν παλμό κάθε 54,9 msec. Ο μετρητής 1 έχει την τιμή Χ=18, οπότε προκύπτει ότι παράγει έναν παλμό κάθε 15,086 μsec. Γενικά, δεν πρέπει να επηρεάζετε τους μετρητές 0 και 1, επειδή πολλές από τις λειτουργίες του συστήματος εξαρτώνται από αυτούς. Ο μετρητής 2 όμως μπορεί να χρησιμοποιηθεί ελεύθερα.

Υπάρχουν έξι διαφορετικές δυνατές καταστάσεις και επομένως τρόποι προγραμματισμού του 8253 (από 0 έως και 5), με τον μετρητή 0 να προγραμματίζεται στην κατάσταση 3 και τον μετρητή 1 στην κατάσταση 2.

2. Προγραμματισμός του 8253

Ο 8086 "βλέπει" το 8253 μέσω τεσσάρων Θυρών Ε/Ε (I/O Ports) - **40h**, **41h** και **42h** για τους τρεις μετρητές και **43h** για τον καταχωρητή εντολών (Control Word Register) του. Ουσιαστικά αυτό το οποίο προγραμματίζεται κάθε φορά είναι οι μετρητές του 8253. Αυτό επιτυγχάνεται με την αποστολή κάποιου byte ελέγχου στον καταχωρητή εντολών του 8253 - δηλαδή με μία:

OUT 43h, AL

όπου ο **AL** έχει φορτωθεί εκ των προτέρων με το byte ελέγχου.

Προκειμένου να προγραμματισθούν οι τρεις μετρητές απαιτούνται και τρία διαφορετικά byte, που πρέπει να σταλούν τον καταχωρητή εντολών, που - αν και ένας - εξυπηρετεί και τους τρεις μετρητές. Η δομή του byte ελέγχου φαίνεται στο Σχ. 4.2.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC – SELECT COUNTER:

SC1	SC0	
0	0	SELECT COUNTER 0
0	1	SELECT COUNTER 1
1	0	SELECT COUNTER 2
1	1	READ-BACK COMMAND (SEE READ OPERATIONS)

RW – READ/WRITE:

RW1	RW0	
0	0	COUNTER LATCH COMMAND (SEE READ OPERATIONS)
0	1	READ/WRITE LEAST SIGNIFICANT BYTE ONLY.
1	0	READ/WRITE MOST SIGNIFICANT BYTE ONLY.
1	1	READ/WRITE LEAST SIGNIFICANT BYTE FIRST, THEN MOST SIGNIFICANT BYTE.

M – MODE:

M2	M1	M0	
0	0	0	MODE 0 – INTERRUPT ON TERMINAL COUNT
0	0	1	MODE 1 – HARDWARE ONE-SHOT
X	1	0	MODE 2 – PULSE GENERATOR
X	1	1	MODE 3 – SQUARE WAVE GENERATOR
1	0	0	MODE 4 – SOFTWARE TRIGGERED STROBE
1	0	1	MODE 5 – HARDWARE TRIGGERED STROBE

BCD:

0	BINARY COUNTER 16-BITS
1	BINARY CODED DECIMAL (BCD) COUNTER (4 DECADES)

Σχ. 4.2 Δομή Byte Ελέγχου του 8253

Όπως φαίνεται, τα πρώτα bit (**SC1, SC0**) καθορίζουν τον επιθυμητό μετρητή, τα επόμενα δύο (**RW1, RW0**) το είδος λειτουργίας ως εξής:

00 - Μεταφορά τιμής μετρητή σε κάποιο Latch (μόνον για 8254)
01 - Διάβασμα/Γράψιμο υψηλού byte
10 - Διάβασμα/Γράψιμο χαμηλού byte
11 - Διάβασμα/Γράψιμο πρώτα χαμηλού και μετά υψηλού byte

Τα επόμενα τρία bit (**M2, M1, M0**) τον αριθμό κατάστασης (0-5) και το τελευταίο bit (**BCD**), εάν η μέτρηση γίνεται στο δυαδικό σύστημα (για τιμή bit 0) ή στο BCD (τιμή bit 1).

Αφού στείλετε το byte ελέγχου, στην συνέχεια τοποθετείτε (για την περίπτωση εγγραφής) την επιθυμητή τιμή στον αντίστοιχο μετρητή. Η τιμή αυτή είναι των 16 bits, αλλά για κάθε

μετρητή ο 8086 χρησιμοποιεί μία διεύθυνση θύρας (I/O port) των 8 bits. Για να λυθεί το πρόβλημα αυτό, το 8253 είναι σχεδιασμένο, ώστε πρώτα να σταλεί στην συγκεκριμένη διεύθυνση θύρας το πρώτο byte και μετά το δεύτερο, σύμφωνα και με τα bits RW που στάλθηκαν προηγουμένως.

ΠΡΟΣΟΧΗ: Θυμηθείτε ότι ο μετρητής 2 είναι συνδεδεμένος με το μεγάφωνο και το ρολόι του συστήματος. Από το Σχ. 4.1 παρατηρείστε ότι προκειμένου κάθε μετρητής να μετρήσει από την τιμή που θα του δοθεί (έως το μηδέν), θα πρέπει να του παρέχεται κάποιο σήμα στο αντίστοιχο **CLK** (συνήθως το ρολόι) και επίσης να είναι ενεργοποιημένο το αντίστοιχο **GATE** (τιμή "1"). Ειδικά για τον μετρητή 2, για να ενεργοποιηθεί με την τιμή που θα του σταλεί, θα πρέπει πρώτα να γραφθεί (με μία OUT) στην θύρα E/E 61h, ένα byte που να έχει το τελευταίο bit (bit 0) ίσο με 1. Εάν το προτελευταίο bit (bit 1) γίνει και αυτό "1", τότε ενεργοποιείται το μεγάφωνο του Η/Υ.

3. Κατάσταση Μετρητών 3 (Mode 3)

Αν και υπάρχουν έξι διαφορετικές καταστάσεις στις οποίες μπορεί να προγραμματισθεί ο κάθε μετρητής του 8253, η πιο σημαντική για τους σκοπούς του παρόντος εργαστηρίου είναι η κατάσταση 3. Εάν ένας μετρητής του 8253 προγραμματισθεί σε αυτήν την κατάσταση (π.χ. ο μετρητής 2), τότε θα παραχθεί στον αντίστοιχο ακροδέκτη (π.χ. **OUT2**) ένας *τετραγωνικό* κύμα, συχνότητας ανάλογης του αριθμού που τοποθετήθηκε στον μετρητή. Η μέτρηση (από τον αριθμό προς το 0) αρχίζει στον αμέσως επόμενο κύκλο ρολογιού από αυτόν της φόρτωσης του μετρητή με την αρχική του τιμή.

Με κάθε παλμό ρολογιού ο μετρητής μειώνεται κατά 2. Όταν ο μετρητής γίνει 2, τότε στέλνεται ένας παλμός στην έξοδό του ένας παλμός και αρχική τιμή επαναφορτώνεται στον μετρητή. Σε περίπτωση που ο αριθμός που τοποθετήθηκε στον μετρητή, δεν είναι ζυγός, τότε ο παλμός εξόδου δεν είναι τέλειος, διαρκώντας για έναν επιπλέον κύκλο ρολογιού. Γενικά θα πρέπει να τοποθετείτε ζυγούς αριθμούς στον μετρητή σας για την Κατάσταση 3.

Σε αυτήν την κατάσταση μπορείτε να χρησιμοποιήσετε τον μετρητή 2 του 8253, σαν χρονομέτρη. Αν και οι έννοιες δεν είναι ιδιαίτερα δύσκολες, ο προγραμματισμός του 8253 (όπως

και κάθε περιφερειακού ολοκληρωμένου) είναι επίπονες και μπορεί άνετα κάποιο λάθος να κάνει ασταθές το όλο υπολογιστικό σύστημα. Η γενική σειρά προγραμματισμού του μετρητή 2, είναι η εξής:

1. Στείλτε στην διεύθυνση Θύρας 61h το byte **XXXXXX11b** (ενεργοποίηση του μετρητή 2 και του μεγαφώνου).

2. Στείλτε στον καταχωρητή ελέγχου του 8253 (θύρα 43h) ένα κατάλληλο byte ελέγχου, που θα "λέει" ότι:

    α. Θα φορτώσετε τον μετρητή 2
    β. Θα γράψετε στέλνοντας πρώτα το χαμηλό και μετά το υψηλό byte της αριθμού μέτρησης του μετρητή 2.
    γ. Θα χρησιμοποιηθεί η κατάσταση 3
    δ. Θα χρησιμοποιηθεί το δυαδικό σύστημα.

3. Γράφετε την αρχική τιμή στον μετρητή 2 (με δύο διαδοχικά **OUT** στην θύρα **42h**).

4. Εκτελείτε το οποιοδήποτε τμήμα προγράμματος θέλετε να χρονομετρήσετε.

5. Στο τέλος στέλνετε στον καταχωρητή ελέγχου το ίδιο byte, ειδοποιώντας τον ότι θέλετε να μετρήσετε την τρέχουσα τιμή του μετρητή 2.

6. Διαβάζετε την τρέχουσα τιμή του μετρητή 2 (με δύο διαδοχικά **IN** από την θύρα **42h**).

7. Βλέπετε την διαφορά μεταξύ της αρχικής και της τρέχουσας τιμής του μετρητή 2, από την οποία μπορείτε να μετρήσετε τον ενδιάμεσο χρόνο.

8. Απενεργοποιείτε τον μετρητή 2 και το μεγάφωνο.

4. Εργαστηριακή Άσκηση

Στο πρόγραμμα το οποίο ακολουθεί βλέπετε τον προγραμματισμό του 8253, με εμφάνιση της τρέχουσας τιμής του μετρητή στην οθόνη, χρησιμοποιώντας το δεκαεξαδικό σύστημα. Ταυτόχρονα είναι ενεργοποιημένο και το μεγάφωνο του Η/Υ και θα πρέπει να ακούτε έναν χαρακτηριστικό ήχο. Αυτό που μεσολαβεί και υποτίθεται πως "μετράει" ο 8253 είναι ο χρόνος από δύο διαδοχικά loop. Αφού κατανοήσετε την λογική του προγράμματος, θα πρέπει να κάνετε τα εξής:

1. Φορτώστε τον μετρητή 2 με μία κατάλληλη τιμή και συγκρίνετε το χρονικό διάστημα που απαιτεί ο Η/Υ σας, προκειμένου:

   α) Να μεταφέρει μία τιμή 16 bits (σταθερά) στον **AX**.

   β) Να διαβάσει αυτήν την τιμή από μία θέση μνήμης. Κάθε μία από αυτές τις λειτουργίες θα επαναληφθεί 65.535 φορές και θα συγκρίνετε τους αντίστοιχους συνολικούς χρόνους. Βλέπετε καμία διαφορά και εάν ναι ποια;

2. Στον μετρητή 2, τοποθετήστε σαν αρχική τιμή το 65.000, ακούστε τον ήχο και επαναλάβετε την ίδια διαδικασία για τον αριθμό 64.500, κ.ο.κ. έως και τον αριθμό 500 (βήμα μείωσης 500).

```
SOROS SEGMENT PARA STACK 'STACK'
        DB    200h DUP(?)
SOROS ENDS

DATA  SEGMENT
        test_w   dw   0
        counter dw   0
DATA  ENDS

CODE  SEGMENT
        ASSUME ds:DATA, cs:CODE, ss:SOROS
```

```
MAIN_PROG   PROC  NEAR
        mov   ax,  data
        mov   ds,  ax                   ; DS δείχνει στην αρχή του Data Segment

        inal,  61h
        or      al,  00000011b          ; bit 1 = ενεργοποίηση μεγαφώνου
                                        ; bit 0 = ενεργοποίηση 8253, Μετρηττή 2
        out    61h,  al

        mov   dx,  43h                  ; Control Register 8253
        mov   al,  10110110b            ; Counter 2, Πρώτα Low μετά High Byte,
                                        ; Mode 3, Binary
        out    dx,  al
        mov   al,  0F0h
        out    42h,  al
        out    42h,  al                 ; Ο αριθμός 0F0F0h στον Μετρητή 2

        mov   si,   0FFFFh              ; Test loop για καθυστέρηση
out_loop:
        mov   cx,  0FFFFh
inner_loop:
        mov   bp,  sp
        mov   ax,  test_w
        loop   inner_loop
        dec    si
        cmp   si, 0
        jg      out_loop                ; Loop από 65535*65535 των δύο mov εντολών

        mov   al,  10110110b            ; Control word για ανάγνωση τρέχουσας τιμής
                                        ; Μετρητή 2
        out    dx,  al                  ; DX ήδη περιέχει 43h (Διεύθυνση Control
                                        ; Register )
        inal,  42h                      ; Διάβασε το πρώτο (low) byte
        mov   bl,  al                   ; Αποθήκευσέ το στον BL
        inal,  42h                      ; Διάβασε το δεύτερο (high) byte
```

```
        mov   bh,  al                   ; Αποθήκευσέ το στον BH
        mov   cx, 4                     ; Θα χρειασθούμε 4 ομάδες ολισθήσεων προς τα
                                        ; αριστερά (για 4 δεκαεξαδικά ψηφία)
get_digit:
        push  cx
        mov   cx,  4                    ; 4 απλές ολισθήσεις προς τα αριστερά για κάθε
                                        ; ψηφίο
        xor   dl,  dl                   ; DL = 0
shift4:
        shl   bx,  1                    ; Ολίσθησε το BX προς τα αριστερά, άρα το high
                                        ;  bit μέσα στο CF
        rcl   dl,  1                    ; Πάρε το low bit μέσα στον DL από το CF
        loop  shift4
        call  DISP_DIGIT                ; Εμφάνισέ το στην οθόνη
        pop   cx
        loop  get_digit

        mov   dl, 'h'                   ; Εμφάνισε τον χαρακτήρα 'h' στο τέλος
        mov   ah, 02h
        int   21h

        inal,  61h                      ; Αποενεργοποίησε Μετρητή 2 και Μεγάφωνο
              ; του PC
        and   al,  11111100b
        out   61h,  al
        mov   ah,  4Ch                  ; Έξοδος στο MS-DOS
        int   21h
MAIN_PROG   ENDP

; ---------------------------------------------------------------------------------------------------------------
```

```
; Γρήγορη & "Βρώμικη" Διαδικασία για Εμφάνιση ενός δεκαεξαδικού ψηφίου στην οθόνη.
; Υποθέτει ότι το προς εμφάνιση ψηφίο (όχι ASCII) είναι στον DL
; ----------------------------------------------------------------------------------------------------------------
DISP_DIGIT  PROC  NEAR
        cmp   dl,  10
        jl      print_it
        add   dl,  'A'-10
        sub   dl,  '0'
print_it:
        add   dl,  '0'
        mov   ah,  02h
        int     21h
        ret
DISP_DIGIT ENDP

CODE  ENDS
END    MAIN_PROG
```

ΕΒΔΟΜΑΔΑ 4

Σκοπός του εργαστηρίου αυτού είναι να έχετε μία εμπειρία από πρόσβαση στην κάρτα γραφικών ενός συμβατού με IBM Η/Υ.

1. Εισαγωγή

Προκειμένου να γίνει εγγραφή κάποιας πληροφορίας στην οθόνη ενός Η/Υ είναι ανάγκη αυτή να κωδικοποιηθεί σε μία κατάλληλη μορφή σημάτων προς την οθόνη. Η περιφερειακή συσκευή που αναλαμβάνει αυτήν την εργασία ονομάζεται σήμερα *Κάρτα Γραφικών*, επειδή έχει την μορφή ηλεκτρονικής κάρτας που τοποθετείται σε κάποια από τις διαθέσιμες θύρες επέκτασης του Η/Υ.

Ανάλογα με το είδος της επιθυμητής απεικόνισης (απλό κείμενο σε 80 στήλες επί 25 γραμμές ή γραφικά σε ανάλυση 1280 επί 1024 στα 256 χρώματα), υπάρχουν και διαφορετικές απαιτήσεις σε μνήμη (της κάρτας γραφικών), που χρειάζεται για να αποθηκεύσει αυτές τις πληροφορίες. Για μεγαλύτερη ταχύτητα μεταφοράς πληροφοριών απεικόνισης μεταξύ Η/Υ και μνήμης της κάρτας γραφικών, ο επεξεργαστής "βλέπει" την μνήμη της κάρτας γραφικών σαν ένα τμήμα της *κύριας* μνήμης του Η/Υ και όχι σαν ένα σύνολο από θύρες Ε/Ε.

Το πρόβλημα που δημιουργείται εδώ όμως είναι η μορφή του 8086 και του MS-DOS, εφ όσον το ποσό της απαιτούμενης μνήμης είναι μεγαλύτερο των 64KB - κάτι που δεν θα μας απασχολήσει στην περίπτωση αυτή (για περισσότερες πληροφορίες ο ενδιαφερόμενος αναγνώστης μπορεί να απευθυνθεί στις σημειώσεις θεωρίας του μαθήματος "Τεχνολογία Η/Υ").

Στην περίπτωση που θέλουμε να απεικονίσουμε μόνον χαρακτήρες στην οθόνη του Η/Υ, οι οποίοι ακολουθούν το σύστημα ASCII και καταλαμβάνουν προκαθορισμένο χώρο από κουκίδες, μπορούμε να χρησιμοποιήσουμε τον απλούστερο δυνατό τρόπο, κατά τον οποίο η οθόνη είναι οργανωμένη σε 25 γραμμές των 80 χαρακτήρων η κάθε μία.

Εφ όσον απαιτούνται 25x80 χαρακτήρες, ένα byte είναι αρκετό για την κωδικοποίηση του κάθε ενός. Επειδή όμως είναι φυσικό να θέλουμε και διαφορετικούς τρόπους εμφάνισης του χαρακτήρα (π.χ. να αναβοσβήνει), απαιτείται και ένα δεύτερο byte ιδιοτήτων ("attribute byte"), ανεβάζοντας στα 4.000 bytes το σύνολο της απαιτούμενης μνήμης.

2. Εγγραφή Χαρακτήρων στην Οθόνη (Text Mode, 80x25)

Η κατάσταση της κάρτας γραφικών στο Κείμενο, με 80 χαρακτήρες ανά γραμμή και 25 γραμμές, είναι η συνηθισμένη κατάσταση στην οποία βρίσκεται κάθε Η/Υ, όταν πρωτοξεκινάει. Οι πληροφορίες που πρέπει να γνωρίζει επομένως κανείς περιορίζονται στην γνώση της αρχής της ομάδας των 4.000 παραπάνω bytes, καθώς και στο νόημα των bits που απαρτίζουν κάθε byte χαρακτηριστικών (attribute byte).

Το πρώτο byte της παραπάνω ομάδας βρίσκεται στην θέση **B800:0000h** για κατάσταση κειμένου (Text Mode), προκειμένου για κάρτες γραφικών τύπου CGA και άνω. Για την περίπτωση προγραμματισμού σε κατάσταση γραφικών, το πρώτο byte (από EGA και άνω) βρίσκεται στην διεύθυνση **A000:0000h**. Αφού εδώ θα ασχοληθούμε μόνον με την απεικόνιση απλού κειμένου μόνον η πρώτη από τις παραπάνω αρχικές διευθύνσεις μας ενδιαφέρει.

Όπως είπαμε και παραπάνω, κάθε χαρακτήρας απεικονίζεται βάσει δύο διαδοχικών bytes, εκ των οποίων το πρώτο περιέχει τον ASCII κωδικό που αντιστοιχεί στον χαρακτήρα, ενώ το αμέσως επόμενο τις "ιδιότητες" του χαρακτήρα. Αφού χρησιμοποιούνται δύο bytes για κάθε χαρακτήρα, μπορούμε να βρούμε εύκολα την διεύθυνση του χαρακτήρα από την αρχή (**B000:0000**), ανάλογα με την επιθυμητή θέση του στην οθόνη (συντεταγμένες **<Στήλη>, <Γραμμή>**), σύμφωνα με τον παρακάτω τύπο:

Διεύθυνση από την Αρχή = (Γραμμή * 80 + Στήλη) * 2

Όπως είναι φανερό, ο πρώτος χαρακτήρας βρίσκεται στις συντεταγμένες (0, 0) και εκτυπώνεται στην επάνω αριστερή θέση της οθόνης.

📖 Προκειμένου να εκτελέσετε οποιαδήποτε άμεση ενέργεια απεικόνισης, είναι πρώτα σκόπιμο να γνωρίζετε τον τύπο κάρτας γραφικών που υπάρχει στον Η/Υ. Για να βρείτε εάν είναι μονόχρωμου τύπου - MDA, HGA (Hercules) - ή εγχρώμου (από CGA και άνω), δεν έχετε παρά να διαβάσετε το περιεχόμενο της Θύρας Ε/Ε **0463h**. Εάν η αποθηκευμένη τιμή είναι 03B4h, τότε η κάρτα γραφικών είναι MDA ή HGA. Εάν η αποθηκευμένη τιμή είναι 03D4h, τότε η κάρτα γραφικών σας είναι οποιαδήποτε άλλη.

Σημειώστε ότι με το πρόγραμμα DEBUG σας είναι δυνατόν να δείτε (από την διεύθυνση **C000:0000h**) την αρχή του BIOS της κάρτας γραφικών και μάλιστα και το μήνυμα Copyright του κατασκευαστή της κάρτας, προκειμένου για σύγχρονες κάρτες.

2.1 Το Byte Ιδιοτήτων

Το δεύτερο byte για τον κάθε χαρακτήρα αντιπροσωπεύει τις διάφορες ιδιότητες απεικόνισης. Προκειμένου για μονόχρωμη κάρτα γραφικών, αυτές είναι (κατά bit):

Bit	Νόημα	
7	0 = Χωρίς Αναβόσβησμα,	1 = Με Αναβόσβησμα
6,5,4	000 = Μαύρο Φόντο,	111 = Άσπρο Φόντο
3	0 = Κανονική Ένταση,	1 = Υψηλή Ένταση
2,1,0	001 = Υπογράμμιση σε Άσπρο,	
	111 = Άσπρο Βασικό, 000 = Μαύρο Βασικό	

Προκειμένου για έγχρωμη κάρτα γραφικών, έχουμε:

Bit	Νόημα
7	0 = Χωρίς Αναβόσβησμα, 1 = Με Αναβόσβησμα
6,5,4	Χρώμα Φόντου
3,2,1,0	Βασικό Χρώμα και Φωτεινότητα (εάν bit 3 = 1)

Προκειμένου λοιπόν να εμφανίσετε έναν χαρακτήρα στην οθόνη χρειάζεστε μία διαδικασία που θα δέχεται τέσσερις παραμέτρους:

1. Στήλη (0-79)
2. Γραμμή (0-24)
3. ASCII κωδικός του χαρακτήρα
4. Byte ιδιοτήτων

Μία τέτοια διαδικασία θα πρέπει κανονικά να αγνοεί όλες τις διακοπές συστήματος για να αποφευχθεί η εμφάνιση "χιονιού" και να πραγματοποιεί την εγγραφή στην μνήμη της κάρτας γραφικών όταν η τελευταία δεν διαβάζεται από τον ειδικό ελεγκτή που στέλνει τα σήματα απεικόνισης στην οθόνη.

Το πρώτο κατορθώνεται με την εντολή **CLI**, που θα πρέπει στο τέλος της διαδικασίας να ακολουθείται από μία **STI**, ώστε να επιτρέπονται ξανά οι διακοπές συστήματος.

🕮 **ΠΑΛΑΙΟΤΕΡΕΣ ΚΑΡΤΕΣ ΜΟΝΟΝ:** Το δεύτερο είναι πιο δύσκολο και επιτυγχάνεται με την ανάγνωση της Θύρας Ε/Ε **0469h** και ελέγχου στο πιο χαμηλό της bit (bit 0). Εάν αυτό είναι ίσο με 1, τότε έχουμε οριζόντια σάρωση και η εγγραφή κάποιου byte στην μνήμη της κάρτας γραφικών είναι ασφαλής, διαφορετικά μπορεί να δημιουργηθεί πρόβλημα εμφάνισης "χιονιού" στην οθόνη. Άρα θα πρέπει:

1. Να περιμένουμε (διαβάζοντας διαρκώς το bit αυτό) να γίνει 0, πράγμα που σημαίνει ότι τελείωσε η οριζόντια σάρωση.

2. Να περιμένουμε (διαβάζοντας διαρκώς το bit αυτό) να γίνει 1, πράγμα που σημαίνει ότι ξαναρχίζει μία οριζόντια σάρωση.

3. Να γράψουμε τα δύο byte για τον χαρακτήρα μας. Με τα προηγούμενα δύο βήματα είμαστε βέβαιοι ότι γράφουμε κατά την αρχή και όχι ίσως το τέλος ενός κύκλου οριζόντιας σάρωσης.

---

3. Τι πρέπει να κάνετε

Θα γράψετε την παραπάνω διαδικασία με τις 4 παραμέτρους. Θα πρέπει να ελέγχετε εάν η στήλη και η γραμμή είναι αποδεκτές τιμές, να απαγορεύσετε τις διακοπές συστήματος, να βεβαιωθείτε ότι είσαστε στην αρχή ενός κύκλου οριζόντιας σάρωσης, να γράψετε τον χαρακτήρα, να επιτρέψετε τις διακοπές συστήματος και να επιστρέψετε στο κυρίως πρόγραμμα.

Βάσει της διαδικασίας αυτής (που θα την ονομάσετε **PRINT_CHAR**), γράψετε ένα πρόγραμμα που θα τυπώνει στην οθόνη, στην περιοχή (παράθυρο) από (10,10) έως και (20,20), οποιαδήποτε συμβολοσειρά πληκτρολογήσουμε (**INT21h**, υπηρεσία **0Ah**). Εάν το μήκος της συμβολοσειράς είναι μεγαλύτερο του 10, θα πρέπει κατά την εκτύπωση να *"διπλώνεται"* στο παραπάνω παράθυρο.

ΕΒΔΟΜΑΔΑ 5

Σκοπός του εργαστηρίου αυτού είναι να έχετε μία πρακτική εξάσκηση με τις εντολές επεξεργασίας συμβολοσειρών (string manipulation instructions), που σας παρέχει ο 8086.

1. Εργαστηριακή Άσκηση

Χρησιμοποιώντας κατάλληλες εντολές επεξεργασίας συμβολοσειρών του 8086, γράψτε ένα πρόγραμμα σε Assembly, το οποίο θα "τρέχετε", αφού πρώτα εκτελέσετε στην γραμμή εντολών του MS-DOS την εντολή:

DIR C:

Η οθόνη του Η/Υ σας θα γεμίσει από τα περιεχόμενα του ευρετηρίου "\" του σκληρού δίσκου "C", παίρνοντας την παρακάτω μορφή:

<Γραμμή 1>
<Γραμμή 2>
.
.
<Γραμμή 25>

Αμέσως θα εκτελείτε το πρόγραμμά σας το οποίο και θα αρχίσεί να "ολισθαίνει" κυκλικά τα περιεχόμενα αυτά κατά γραμμή από επάνω προς τα κάτω. Έτσι, μετά την πρώτη "ολίσθηση", η οθόνη σας θα παρουσιάζει την παρακάτω μορφή:

<Γραμμή 25>
<Γραμμή 1>
.
.

<Γραμμή 24>

Για να το επιτύχετε, θα πρέπει να χρησιμοποιήσετε τουλάχιστον έναν buffer των 160 bytes (80 χαρακτήρες + 80 attribute bytes), για την πρώτη ολίσθηση από την τελευταία γραμμή σε αυτόν και τελικά αντιγραφή από αυτόν στην κορυφαία γραμμή της οθόνης.

Η δυσκολία για εσάς είναι βασικά στον εντοπισμό της μετατόπισης (offset) του byte που αντιστοιχεί στον πρώτο χαρακτήρα της κάθε γραμμής της οθόνης. Από εκεί και μετά χρειάζεστε μία απλή επανάληψη (loop).

ΕΒΔΟΜΑΔΑ 6

Σκοπός του εργαστηρίου αυτού είναι να έχετε μία πρακτική εξάσκηση με τον σχεδιασμό απλών γραφικών στην οθόνη του Η/Υ σας.

1. Εισαγωγή

Εκτός από την δυνατότητα απεικόνισης μίας συγκεκριμένης ομάδας αλφαριθμητικών χαρακτήρων (το γνωστό σύνολο extended ASCII με 256 χαρακτήρες), είναι δυνατή και η σχεδίαση κουκίδων (pixels) στην οθόνη. Οι τελευταίες αποτελούν τους θεμέλιους λίθους για οποιαδήποτε γραφική αναπαράσταση.

Για να επιτευχθεί όμως η παραπάνω δυνατότητα, θα πρέπει πρώτα να τεθεί η κάρτα γραφικών του Η/Υ σε μία κατάλληλη *κατάσταση γραφικών (graphics mode)*. Αν και υπάρχουν διαφορετικοί τρόποι με τους οποίους μπορεί κανείς να επιτύχει, ο πλέον χρησιμοποιούμενος (και απλούστερος) είναι να χρησιμοποιήσει μία υπηρεσία του BIOS.

Πιο συγκεκριμένα, αρκεί να εκτελέσετε τα παρακάτω:

```
mov AL, <mode>
mov AH, 0
int 10h
```

και η κάρτα γραφικών θα βρεθεί στην κατάσταση που επιθυμείτε.

Προκειμένου όμως να είναι δυνατή η διαφοροποίηση των διαφόρων καταστάσεων, κάθε μία από αυτές έχει και έναν ξεχωριστό κωδικό αριθμό ταυτότητας. Για να είστε σε θέση να ξέρετε ποιος είναι ο κωδικός αριθμός που σας ενδιαφέρει, εννοείται πως θα πρέπει να ανατρέξετε στο εγχειρίδιο αναφοράς της κάρτας γραφικών που διαθέτετε στον υπολογιστή σας.

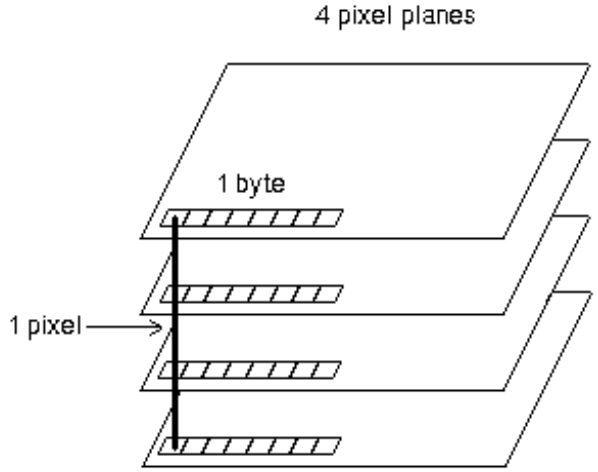
Ορισμένες όμως από αυτές τις καταστάσεις είναι κοινές για όλες τις κάρτες γραφικών που υποστηρίζουν κάποια στάνταρ. Έτσι, η κατάσταση 2 αντιστοιχεί σε Κατάσταση Κειμένου 80x25 χαρακτήρων για κάρτες τύπου VGA.

Ενώ όμως είναι σχετικά εύκολη η απεικόνιση αλφαριθμητικών χαρακτήρων στην οθόνη, δεν συμβαίνει και το ίδιο με τον σχεδιασμό κουκίδων (pixels). Το πρόβλημα προέρχεται από το γεγονός της περιορισμένης μνήμης που είναι άμεσα προσπελάσιμη από τον 8086 (συνήθως μόνον 64K) σε σχέση με το απαραίτητο ποσό πληροφορίας για όλες τις κουκίδες. Έτσι, εάν υποθέσουμε 320x200 κουκίδες με 256 χρώματα για την κάθε μία, απαιτούνται συνολικά 320x200 = 64.000 bytes, για 640x480 κουκίδες με 16 χρώματα, απαιτούνται συνολικά 640x480/2 = 153.600 bytes που προφανώς δεν χωράν σε 64K μνήμης.

Σε αυτήν την περίπτωση ο 8086 μπορεί να προσπελάσει μόνον 64K bytes κάθε φορά από το απαιτούμενο σύνολο (1 bank) και πρέπει να γίνεται αλλαγή (bank switching) για προσπέλαση στις υπόλοιπες πληροφορίες. Η προσπέλαση αυτή προκειμένου για Καταστάσεις Γραφικών (Graphics Modes) γίνεται από την διεύθυνση **A000:0000** και μετά (για 64K συνήθως).

Η πιο απλή περίπτωση είναι η Κατάσταση (Γραφικών) 13h. Αυτή αντιπροσωπεύει σχεδιασμό κουκίδων με ανάλυση 320 στηλών επί 200 γραμμές, με 256 χρώματα (και ένα byte) ανά κουκίδα. Επειδή συνολικά απαιτούνται 64.000 bytes, τα πλεονεκτήματά της είναι ότι δεν χρειάζεται αλλαγή bank, αλλά και ότι διαδοχικές κουκίδες αναπαρίστανται με byte τοποθετημένα διαδοχικά στην μνήμη. Έτσι ο προγραμματισμός είναι εξαιρετικά εύκολος για απλούς σχεδιασμούς.

Αντιθέτως, στην περίπτωση 16 χρωμάτων ανά κουκίδα (π.χ. mode 4) κάθε ένα από τα 4 bits βρίσκεται σε διαφορετικό *επίπεδο*, με ένα byte να περιέχει από ένα bit για 8 διαφορετικές κουκίδες όπως φαίνεται και από το παραπάνω σχήμα.



Σχ. 6.1 Μορφή Αποθήκευσης για 16 Χρώματα ανά Κουκίδα

Γενικά τα βήματα τα οποία θα πρέπει να ακολουθηθούν για απλό σχεδιασμό στην οθόνη, βάσει των παραπάνω πληροφοριών είναι:

1. Τοποθετείστε την κάρτα γραφικών στην κατάσταση **13h**

2. Σχεδιάστε ο,τιδήποτε επιθυμείτε με κατάλληλο συνδυασμό κουκίδων

3. Τοποθετείστε την κάρτα γραφικών στην κατάσταση **2h**

2. Εργαστηριακή Άσκηση

Κάνοντας χρήση των παραπάνω πληροφοριών, γράψτε ένα πρόγραμμα που θα "γεμίζει" το ορθογώνιο που ορίζεται από τις συντεταγμένες (50, 50) και (300, 100). Θυμηθείτε ότι τώρα έχετε 320 κουκίδες στον άξονα των X και 200 στον άξονα των Y, ενώ η πρώτη κουκίδα βρίσκεται σε SEGMENT **A000h** και OFFSET **0000**. Μπορείτε να επιλέξετε οποιοδήποτε από τα διαθέσιμα 256 χρώματα.

Θα πρέπει πρώτα να γράψετε μία διαδικασία **DRAW_PIXEL (X, Y, COLOUR),** όπου **X** και **Y** οι αντίστοιχες συντεταγμένες και **COLOUR** ο κωδικός χρώματος. Αυτές οι παράμετροι θα πρέπει να "περνάν" στην διαδικασία μέσω της σωρού. Η διαδικασία αυτή απλά υπολογίζει το OFFSET που αντιστοιχεί σε αυτές τις συντεταγμένες (με βάση του Segment το **A000h**) και τοποθετεί σε αυτήν το byte που παίρνει μέσω της **COLOUR**.

Στην συνέχεια θα πρέπει να γράψετε μία **διαδικασία DRAW_HLINE (X, Y, LENGTH, COLOUR),** όπου **X** και **Y** οι συντεταγμένες της πρώτης κουκίδας (από "αριστερά") της οριζόντιας γραμμής που σχεδιάζει αυτή η διαδικασία, **LENGTH** το μήκος της οριζόντιας γραμμής σε κουκίδες και **COLOUR** το χρώμα της γραμμής. Προφανώς η διαδικασία αυτή καλεί την **DRAW_PIXEL** επανειλημμένα για να σχεδιάσει την οριζόντια γραμμή.

Τέλος, θα έχετε μία διαδικασία ονόματι **FILL_RECT(X1, Y1, X2, Y2, COL),** όπου **X1** και **Y1** οι συντεταγμένες της "επάνω αριστερής" κουκίδας, **X2, Y2** οι συντεταγμένες της "κάτω δεξιάς κουκίδας" και **COL** ο κωδικός του επιθυμητού χρώματος.

Στην συνέχεια το κυρίως πρόγραμμά σας θα έχει την εξής μορφή:

1. Πήγαινε σε mode **13h**
2. **call FILL_RECT(50, 50, 300, 100, <χρώμα>)**
3. Περίμενε χαρακτήρα από πληκτρολόγιο
4. Πήγαινε σε mode **2h**

Το βήμα <3> σκοπό έχει να σταματήσει προσωρινά τον Η/Υ σας, ώστε να προλάβετε να δείτε το ορθογώνιο να σχεδιάζεται στην οθόνη σας.

ΕΒΔΟΜΑΔΑ 6Α

Σκοπός του εργαστηρίου αυτού είναι να έχετε μία εμπειρία από την χρήση εντολών του 8086, που εφαρμόζονται σε σειρές χαρακτήρων (συμβολοσειρές). Χρησιμοποιώντας κατάλληλες εντολές επεξεργασίας συμβολοσειρών και την εκφώνηση της άσκησης της προηγούμενης εβδομάδος, γράψτε ένα πρόγραμμα, το οποίο:

1. Έχει δύο περιοχές **A** και **B** στην κύρια μνήμη ικανές να αποθηκεύσουν τα περιεχόμενα του παραθύρου από (0, 0) έως και (79, 11).

2. Κάθε φορά που εισάγετε μία συμβολοσειρά, αποθηκεύει τα περιεχόμενα του παραθύρου στην περιοχή **A** ή **B** της κύριας μνήμης, επιλέγοντας εκείνην που έχει χρησιμοποιηθεί πιο παλιά, και μετά τυπώνει την συμβολοσειρά που έχετε εισαγάγει από το πληκτρολόγιο.

3. Εάν όμως η συμβολοσειρά, που έχετε εισαγάγει, έχει μόνον το γράμμα "T", τότε στο παράθυρο τοποθετούνται τα περιεχόμενα της περιοχής **A** ή **B** (επιλέγεται η πιο παλιά), ενώ η άλλη περιοχή μαρκάρεται πλέον ως η πιο παλιά, ώστε εάν ξαναεισάγετε το γράμμα "T", να εκτυπωθούν στο παράθυρο τα περιεχόμενα της δεύτερης περιοχής.

4. Εισαγωγή συμβολοσειράς που να περιέχει μόνον το γράμμα "C", έχει ως αποτέλεσμα τον καθαρισμό του παραπάνω παραθύρου (και ενημέρωση των περιεχομένων της αντίστοιχης περιοχής μνήμης).

ΕΒΔΟΜΑΔΑ 7

Σκοπός του εργαστηρίου αυτού είναι να έχετε μία εμπειρία από την δυνατότητα σύνδεσης προγραμμάτων σε γλώσσες υψηλού επιπέδου (π.χ. PASCAL ή C), με προγράμματα γραμμένα σε Assembly 8086.

1. Εισαγωγή

Η δημιουργία προγραμμάτων γραμμένων σε Assembly έχει τα πλεονεκτήματα της μεγαλύτερης δυνατής ταχύτητας και την πλήρη κατανόηση του υλικού (επεξεργαστής) που βρίσκεται στο συγκεκριμένο υπολογιστικό σύστημα. Όπως όμως είναι φυσικό, η δημιουργία προγραμμάτων σε μία γλώσσα χαμηλού επιπέδου (Assembly) παρουσιάζει μειονεκτήματα που κάνουν την χρήση της να περιορίζεται μόνον σε εξειδικευμένες εφαρμογές και γενικά όπου απαιτείται η μέγιστη δυνατή ταχύτητα.

Για αυτόν τον λόγο, είναι απαραίτητο να γνωρίζετε τον τρόπο δημιουργίας συνθέτων προγραμμάτων που αποτελούνται από ένα τμήμα γραμμένο σε Assembly και το υπόλοιπο σε κάποια γλώσσα ανωτέρου επιπέδου.

Γενικά, μπορεί κανείς να θεωρήσει το αρχείο σε Assembly ως ένα σύνολο από παρεχόμενες υπηρεσίες προς το πρόγραμμα της γλώσσας ανωτέρου επιπέδου. Είναι επόμενο λοιπόν να πρέπει αυτές οι υπηρεσίες να οργανωθούν υπό την μορφή διαδικασιών (procedures), οι οποίες και γίνονται γνωστές στο πρόγραμμα σε γλώσσα ανωτέρου επιπέδου, αλλά όχι και οι λεπτομέρειές τους.

Για αυτόν τον σκοπό, υπάρχουν στην Assembly 8086, δύο οδηγίες (directives):

1. **PUBLIC** **<Name>, ..., <Name>**

2. **EXTRN** **<Name>:<Type>, ..., <Name>:<Type>**

Η **PUBLIC** σκοπό έχει να δηλώσει τις παραμέτρους που την ακολουθούν ως ονόματα που καθίστανται γνωστά (και άρα μπορούν να χρησιμοποιηθούν) σε οποιαδήποτε άλλα αρχεία χρησιμοποιούνται για την δημιουργία του τελικού προγράμματος. Οι παράμετροι αυτοί είναι οποιεσδήποτε Μεταβλητές ή Ετικέτες, εκτός από αυτά το ονόματα που έχουν ορισθεί με την βοήθεια των οδηγιών **EQU** και =. Συνήθως όμως είναι ονόματα Διαδικασιών (Procedures).

Η **EXTRN** αντιθέτως, σκοπό έχει να δηλώσει στον Assembler ότι για το υπόλοιπο αρχείο που ο τελευταίος επεξεργάζεται, όλες οι παράμετροι με το όνομα **<Name>** είναι ονόματα είτε:

1. Διαδικασιών ή Ετικετών και τα αντίστοιχα **<Type>** είναι τότε **NEAR** ή **FAR**.

2. Ονόματα Μεταβλητών και τα αντίστοιχα **<Type>** είναι **BYTE**, **WORD**, **DWORD** ή οποιοδήποτε όνομα έχει προηγουμένως δηλωθεί με μία **EQU**.

Από τα παραπάνω είναι προφανές ότι η **EXTRN** απαιτείται οποτεδήποτε έχουμε περισσότερα από ένα αρχεία γραμμένα σε Assembly τα οποία χρησιμοποιούν οντότητες ορισμένες αλλού.

Το ειδικό πρόγραμμα που αναλαμβάνει να κάνει την παραπάνω σύνδεση είναι ο *Συνδέτης (Linker)*. Για να μπορέσει να γίνει αυτή η σύνδεση όμως θα πρέπει να ακολουθούνται ορισμένοι κανόνες, διαφορετικοί για κάθε γλώσσα και Compiler, κάτι που μπορεί κανείς να πληροφορηθεί ανατρέχοντας στα αντίστοιχα εγχειρίδια αναφοράς.

Εδώ θα εξετάσουμε την περίπτωση κλήσης μίας συνάρτησης ("συνάρτηση" επειδή επιστρέφει μία τιμή), ονόματι **SumAsm(A, B),** όπου **A** και **B** είναι ακέραιοι (των 16 bit). Η συνάρτηση αυτή είναι γραμμένη (ορισμένη) στο αρχείο **SumAsm.asm** και επιστρέφει ως αποτέλεσμα το άθροισμα των δύο παραπάνω παραμέτρων της, που είναι και αυτό ένας ακέραιος των 16 bits, στον Συσσωρευτή (**AX**).

Στην συνέχεια θα εξετάσουμε τις περιπτώσεις σύνδεσης με αντίστοιχο αρχείο σε PASCAL και σε C.

2. Σύνδεση με αρχείο σε PASCAL

Έστω το παρακάτω αρχείο γραμμένο σε PASCAL, ονόματι **SumPas.pas**. Οι κανόνες σύνταξής του για τους σκοπούς μας είναι οι εξής:

1. Κάθε συνάρτηση που δεν ορίζεται αλλά χρησιμοποιείται σε αυτό, πρέπει να δηλώνεται ως **external**.

2. Οι παράμετροι περνιούνται μέσω της Σωρού. Η σειρά με την οποία σπρώχνονται στην τελευταία είναι πρώτα η πρώτη παράμετρος, μετά η δεύτερη, κ.ο.κ.

3. Χρησιμοποιείται διαφορετικό Τμήμα Κώδικα για τον κώδικα των δύο αρχείων, άρα πρέπει όλες οι διαδικασίες που χρησιμοποιεί το **SumPas.pas** και είναι δηλωμένες στο **SumAsm.asm**, πρέπει να δηλωθούν ως **FAR** (εδώ μόνον η **SumAsm**).

4. Το αρχείο Assembly είναι στο τέλος υπεύθυνο για την εκκαθάριση της Σωρού από τις παραμέτρους.

5. Ειδικά για την Turbo Pascal, το Τμήμα Κώδικα των εξωτερικών αρχείων-αντικειμένων (external object files) πρέπει να ονομάζεται "**CODE**" ή "**CSEG**".

Παρακάτω φαίνεται ένα παράδειγμα για τα αντίστοιχα αρχεία - πρώτα το **SumPas.pas** και στην συνέχεια το **SumAsm.asm**:

```
{$F+}                                   (* Υποχρεωτικά κλήσεις τύπου FAR    *)
{$L SumAsm}                             (* Link με SumAsm                    *)

Function        SumAsm (A, B: integer):     integer; external;

var
        A, B, Result    : integer;
        Code            : integer;
        InputString     : string;

begin
        repeat
                Write (‘Enter A:’);
                Readln (InputString);
                val (InputString, A, Code);
                if InputString <> ‘’ then
                        begin
                                Write (‘Enter B:’);
                                Readln (InputString);
                                val (InputString, B, Code);     (* Στο Code γράφεται ο κωδικός
                                                                   λάθους εάν υπήρξε κάτι τέτοιο *)
                                Result := SumAsm (A, B);
                                Writeln (‘Result =’, Result);
                                Writeln(‘’);
                        end;
        until InputString = ‘’
end.
```

Αρχείο SumPas.PAS

```
;       Title SumAsm.asm
;
DOSSEG                                  ;       MS-DOS style for segment arrangement
        PUBLIC      PASCAL      SumPas
CODE  SEGMENT  WORD  PUBLIC 'CODE'
        ASSUME CS:CODE

SumAsm PROC  FAR
        push    bp
        mov     bp, sp

        mov     ax, [bp+8]              ;       Πάρε το A
        add     ax, [bp+6]              ;       Πάρε το B πρόσθεσέ το στο A
                                        ;       Αποτέλεσμα στον AX
        pop     bp
        ret     4
SumAsm ENDP

CODE  ENDS
END
```

Αρχείο SumAsm.ASM

3. Σύνδεση με Αρχείο σε C

Οι κανόνες σύνταξης των αντίστοιχων αρχείων είναι τώρα κάπως διαφορετικοί. Οι διαφορές είναι οι εξής:

1. Ο Compiler της C προσθέτει ένα "_" μπροστά από κάθε εξωτερικό σύμβολο, άρα θα πρέπει το όνομα της κάθε συνάρτησης που εξάγεται από το αρχείο Assembly να ορίζεται με τον χαρακτήρα "_" στην αρχή του.

2. Η σειρά με την οποία σπρώχνονται οι παράμετροι στην Σωρό είναι αντίθετη: Πρώτα η τελευταία παράμετρος, μετά η προτελευταία, κ.ο.κ.

3. Το αποτέλεσμα επιστρέφεται και πάλι στον AX για την περίπτωσή μας.

4. Τώρα είναι το αρχείο σε C υπεύθυνο στο τέλος για την εκκαθάριση της Σωρού από τις παραμέτρους.

5. Εξ ορισμού η C υποθέτει το μοντέλο μνήμης SMALL, άρα οι διαδικασίες στο αντίστοιχο αρχείο Assembly πρέπει να ορισθούν ως τύπου **NEAR**.

```
#include        <stdio.h>
#include        <stdlib.h>
main()
        {
        int             A, B, Result;
        char            InputString[30];
        extern int      SumC (int A, int B);
        do
                {
                printf ("Enter A: ");
                gets (InputString);
                A = atoi (InputString);

                if (*InputString != '\0')
                        {
                        printf ("Enter B: ");
                        gets (InputString);
                        B = atoi (InputString);
                        Result = SumC (A, B);
                        printf ("Result = %i\n\n", Result);
                        }
                }
        while (*InputString != '\0');
        }
```

Αρχείο SumC.c

```
        DOSSEG
        .MODEL  SMALL

        PUBLIC          _SumC

        .CODE

_SumC PROC  NEAR
        push    bp
        mov     bp, sp

        mov     ax, [bp+4]              ;       get A
        add     ax, [bp+6]              ;       get B and add it to AX -> result in AX

        pop     bp
        ret
_SumC ENDP

END
```

Αρχείο SumAsm.asm

4. Εργαστηριακή Άσκηση

Χρησιμοποιώντας την άσκηση της προηγούμενης εβδομάδας, τροποποιήστε τον Assembly κώδικα για την συνάρτηση **FILL_RECT(X1, Y1, X2, Y2, COLOUR),** έτσι ώστε να εισάγετε αρχικές τιμές για τις παραμέτρους από ένα αρχείο σε PASCAL ή C (όποια γλώσσα επιθυμείτε) και από εκεί καλείται (μετά την επιτυχή σύνδεση) η **FILL_RECT**. Το πρόγραμμά σας θα πρέπει να ελέγχει εάν οι τιμές των παραμέτρων είναι σωστές ή όχι (στο αρχείο της PASCAL ή C).

ΕΒΔΟΜΑΔΑ 8

Σκοπός του εργαστηρίου αυτού είναι να αποκτήσετε μερικές απλές γνώσεις στην χρήση και προγραμματισμό ενός ολοκληρωμένου για παράλληλη μεταφορά δεδομένων μεταξύ επεξεργαστή και περιφερειακών. Ένα τέτοιο ολοκληρωμένο είναι το 8255, το οποίο χρησιμοποιείται κατά κόρον ως ελεγκτής περιφερειακών συσκευών γενικού σκοπού. Τέτοιες περιπτώσεις είναι εκείνες όπου δεν είναι αναγκαίος μεγάλος βαθμός εξειδίκευσης, όπως ο έλεγχος πληκτρολογίου, κλπ.

1. Εισαγωγή

Προγραμματιστικά το 8255 δεν διαφέρει από οποιοδήποτε άλλο ολοκληρωμένο. Ως προς την ΚΜΕ, εμφανίζεται σαν ένα σύνολο από διευθύνσεις Θυρών Εισόδου / Εξόδου (Input/Output Ports). Γενικά υπάρχουν τέσσερις διαφορετικές περιπτώσεις για την ταυτότητα και σκοπό της κάθε Θύρας, οι οποίες μπορεί να είναι οι εξής:

1. Η Θύρα αποτελεί σημείο πρόσβασης στον Καταχωρητή Ελέγχου του ολοκληρωμένου, μέσω του οποίου γίνεται ο προγραμματισμός του.

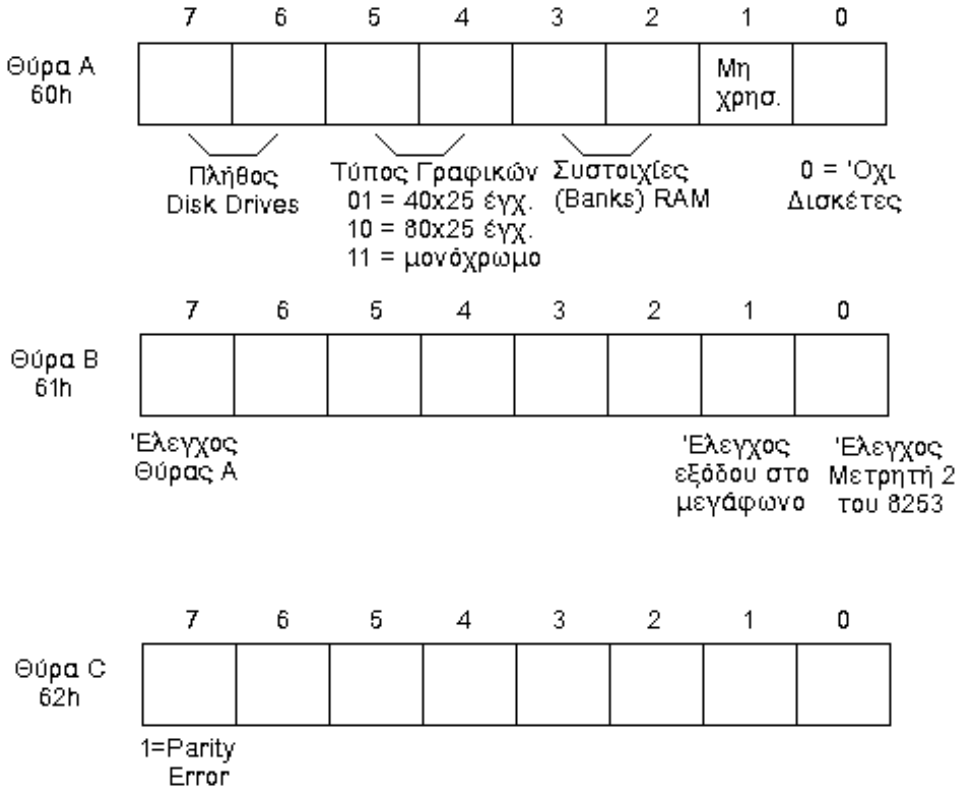
2. Η Θύρα αποτελεί σημείο Εισόδου δεδομένων από την οποιαδήποτε ελεγχόμενη συσκευή προς το ολοκληρωμένο.

3. Η Θύρα αποτελεί σημείο Εξόδου δεδομένων από το ολοκληρωμένο προς την οποιαδήποτε ελεγχόμενη από αυτό συσκευή.

4. Η Θύρα αποτελεί συνδυασμό των παραπάνω περιπτώσεων (2) και (3).

Ειδικότερα στον IBM PC το 8255 χρησιμοποιείται και για την αποθήκευση πληροφοριών που σχετίζονται με την κατάσταση του Η/Υ. Το 8255 έχει τρεις θύρες που είναι γνωστές με τα ονόματα **A, B, C** και εμφανίζονται ως προς τον 8086 στις διευθύνσεις θυρών **60h**, **61h** και

62h, αντίστοιχα. Από αυτές τις τρεις θύρες μπορείτε *μόνον* να διαβάσετε, εκτός από την θύρα Β, στην οποία μπορείτε *και* να γράψετε. Σημειώστε ότι το μέγεθος της κάθε θύρας είναι 8 bits. Όπως φαίνεται και από το παρακάτω σχήμα, κάνοντας 1 το bit 7 της θύρας **61h**, φορτώνονται κάποιες πληροφορίες για το σύστημα στην θύρα **60h**.



Σχ. 8.1 Κυριότερα Περιεχόμενα Θυρών 60h, 61h και 62h

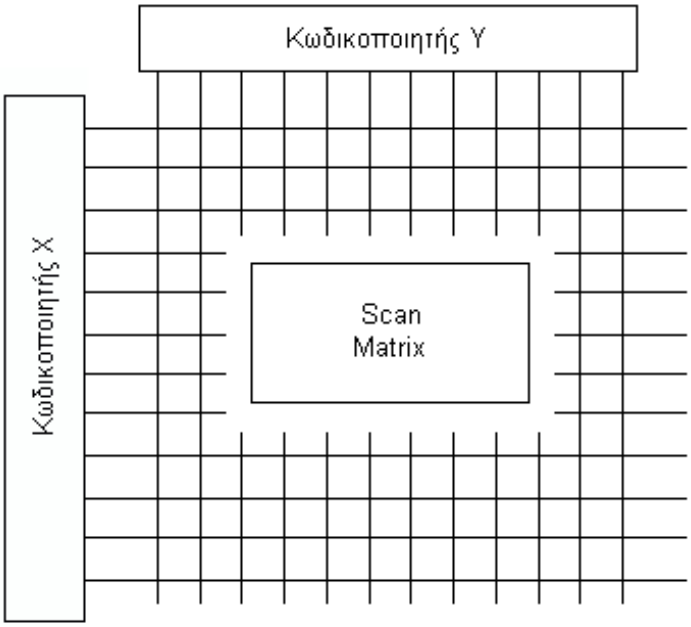
Εάν όμως το bit αυτό είναι 0, τότε η θύρα **A**, περιέχει τον *Κωδικό Σάρωσης (Scan Code)* του πλήκτρου που πατήθηκε. Ο τελευταίος διαφέρει από τον ASCII κωδικό των χαρακτήρων, δεδομένου ότι χρησιμοποιείται για την αναγνώριση και πλήκτρων ειδικών λειτουργιών (π.χ. έτσι μπορεί να αναγνωρισθεί εάν πατήθηκε το πλήκτρο Αριστερό-SHIFT).

Στον αρχικό IBM PC χρησιμοποιούνταν για τον σκοπό αυτό το 8255, με ένα πρωτόκολλο *χειραψίας* (handshaking), κατά το οποίο ο 8088 έκανε το bit 7 της θύρας **61h** ίσο με 0, για να διαβάσει τον κωδικό σάρωσης του πλήκτρου που πατήθηκε και αμέσως μετά 1 για να δείξει ότι ήδη διάβασε την τιμή αυτή, ώστε ο 8255 να τοποθετήσει στην θύρα 60h τον επόμενο κωδικό σάρωσης.

1.2 Πληκτρολόγιο

Στον αρχικό IBM PC/XT δεν ήταν δυνατόν να αποσταλούν εντολές ελέγχου στον ελεγκτή πληκτρολογίου (8048) και έτσι να είναι δυνατός ο προγραμματισμός του πληκτρολογίου, π.χ. για αλλαγή της συχνότητας επαναλήψεως (repetition rate) πλήκτρων. Τα παραπάνω άλλαξαν με την εμφάνιση του IBM PC/AT και συμβατών. Εδώ υπάρχει ο ελεγκτής 8741/2 και άλλα συμβατά ολοκληρωμένα.

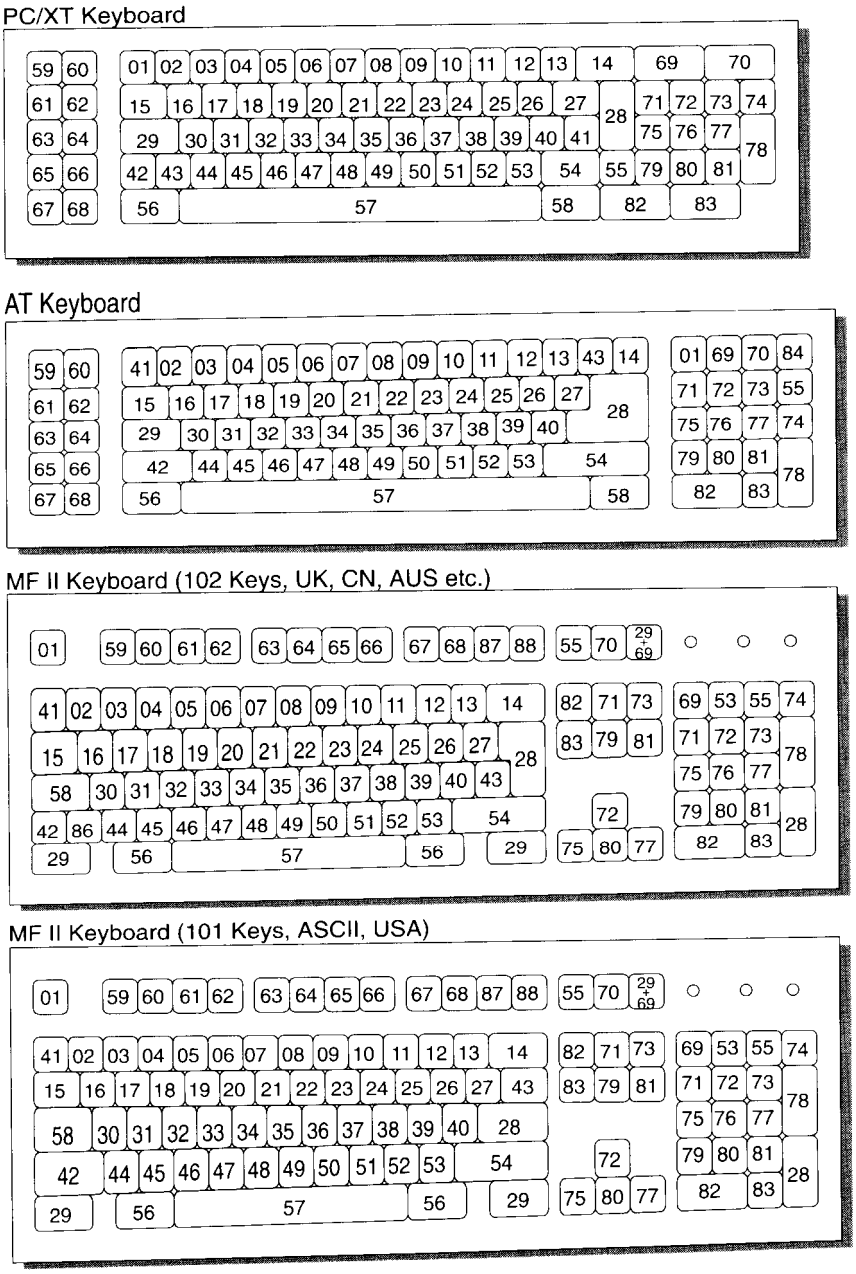
Το ζήτημα είναι όμως πώς λειτουργεί κατά βάση το πληκτρολόγιο. Αν και έχετε μάθει να αναφέρεστε σε διάφορους χαρακτήρες χρησιμοποιώντας το σύστημα ASCII, σε ένα τυπικό πληκτρολόγιο υπάρχουν περισσότερα πλήκτρα. Τι συμβαίνει λοιπόν;



Σχ. 8.2 Βασική Δομή Πληκτρολογίου

Φανταστείτε έναν δισδιάστατο πίνακα, που σχηματίζεται από την συμβολή οριζοντίων και κατακόρυφων γραμμών οι οποίες όμως δεν συνδέονται μεταξύ τους (βλ. Σχ. 8.2). Όταν πιέσετε κάποιο πλήκτρο ενεργοποιείται κάποια γραμμή και στήλη του πίνακα, που αντιστοιχούν στο συγκεκριμένο πλήκτρο. Σαρώνοντας τις γραμμές και στήλες του πίνακα αυτού είναι δυνατόν να βρεθεί πότε πιέσθηκε και πότε αφέθηκε κάποιο συγκεκριμένο πλήκτρο. Οι τιμές που προκύπτουν άμεσα, ονομάζεται συλλογικά *Κωδικοί Σάρωσης (Scan Codes).*

Υπάρχουν επομένως δύο περιπτώσεις:



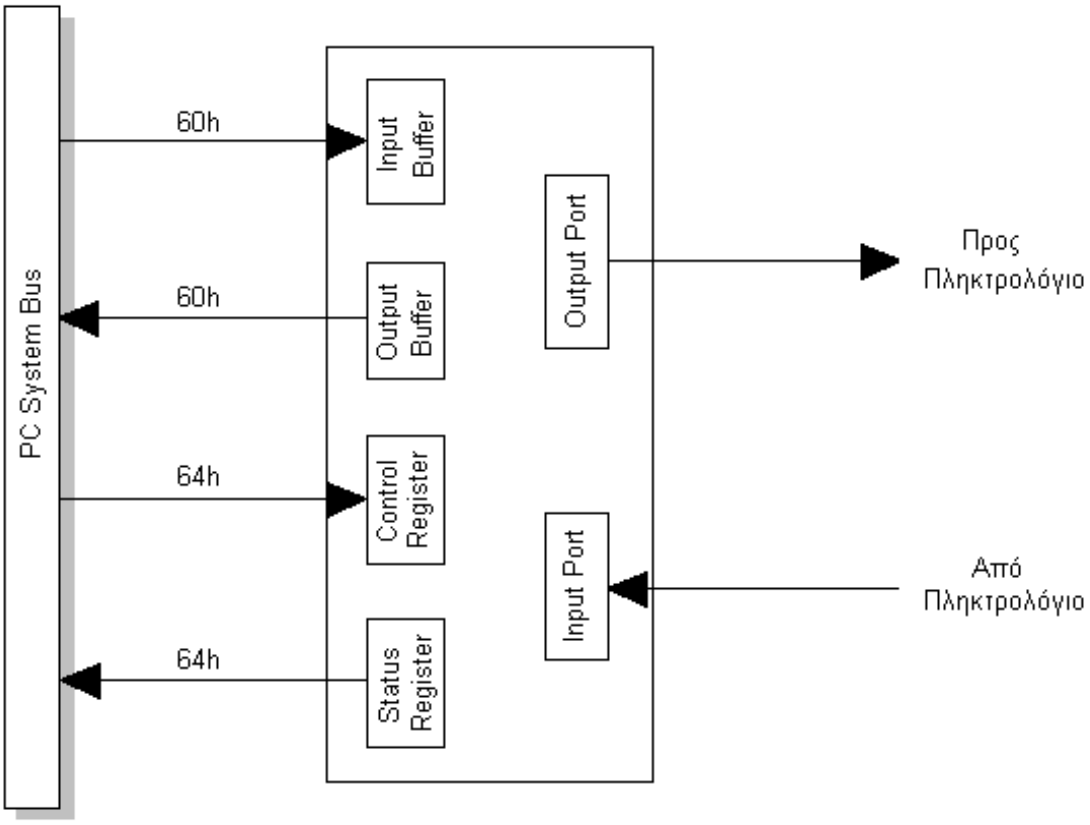
Σχ. 8.3 Κωδικοί Σάρωσης για Διάφορους Τύπους Πληκτρολογίων (Δεκαδικό Σύστημα)

Κάποιο πλήκτρο **πιέζεται**, οπότε το πληκτρολόγιο γεννά έναν κωδικό σάρωσης (1 byte), που ονομάζεται *"make-code"* και μεταφέρεται στο interface του PC. Η μεταφορά αυτή γεννά με την σειρά της μία διακοπή (09h) στο BIOS, η διαδικασία εξυπηρέτησης της οποίας με την σειρά της διαβάζει τον κωδικό αυτό από το interface του PC. Εάν ο κωδικός αυτός αντιστοιχεί σε κάποιο "ήσυχο" πλήκτρο (π.χ. SHIFT), είναι δουλειά του οδηγού πληκτρολογίου (keyboard driver) να εκτελέσει συνδυασμούς δύο make-codes. Ο κωδικός σάρωσης (άρα και

make-code) κάθε πλήκτρου φαίνονται στο Σχ. 8.3 (στο Δεκαδικό Σύστημα) για διάφορους τύπους πληκτρολογίων.

Στην δεύτερη περίπτωση κάποιο πλήκτρο **αφήνεται**. Εδώ γεννάται ένας *"break-code"*, ο οποίος είναι ίδιος με εκείνον της προηγούμενης περίπτωσης, με την διαφορά όμως ότι το πιο σημαντικό bit (bit 7) έχει την τιμή 1. Άρα ένας τέτοιος κωδικός διαφέρει από έναν make-code κατά 128 (στο Δεκαδικό Σύστημα).

Με τα καινούργια πληκτρολόγια (MFII) υπάρχει όμως σειρά σοβαρών προβλημάτων με την εισαγωγή καινούργιων πλήκτρων. Για τον σκοπό αυτό, κάθε φορά που πιέζεται το πλήκτρο PAUSE, πρώτα αποστέλλεται το byte **e1h** και μετά ο make- ή break-code. Για τα υπόλοιπα καινούργια πλήκτρα (π.χ. δεξί CTRL και δεξί ALT) πρώτα αποστέλλεται το byte **e0h** και μετά ο make- ή break-code.



Σχ. 8.4 Έλεγχος Πληκτρολογίου και 8255

2. Προγραμματισμός του Πληκτρολογίου

Μπορείτε να προγραμματίσετε τους καινούργιους τύπους πληκτρολογίων όπως προγραμματίζετε και άλλες περιφερειακές συσκευές. Εδώ θα εστιάσουμε την προσοχή μας στον προγραμματισμό πληκτρολογίων καινούργιου τύπου. Οι διευθύνσεις θυρών εισόδου / εξόδου που μας ενδιαφέρουν είναι οι **60h** και **64h**. Αυτές αντιστοιχούν σε τέσσερις καταχωρητές του 8255 (άλλο συμβατό ολοκληρωμένο σε σημερινούς Η/Υ πλέον), όπως φαίνεται στο Σχ. 8.4.

Όπως βλέπετε υπάρχουν τέσσερις καταχωρητές στο 8255 που σχετίζονται με τον έλεγχο του πληκτρολογίου: Δύο καταχωρητές εισόδου και δύο καταχωρητές εξόδου στις παραπάνω δύο διευθύνσεις θυρών, οι οποίοι αντιστοιχούν σε δύο διευθύνσεις Θυρών Εισόδου / Εξόδου. Η διαφοροποίηση μεταξύ τους γίνεται και με την βοήθεια των εντολών IN και OUT - π.χ. η IN 60h προκαλεί ανάγνωση των περιεχομένων του Input Buffer. Προσοχή στην διαφορά μεταξύ Input Buffer - Input Port και Output Buffer - Output Port: Η ανάγνωση ενός κώδικα σάρωσης γίνεται με την τοποθέτησή του από το πληκτρολόγιο στο Input Port και από εκεί στον Output Buffer. Η προσπέλαση όλων αυτών γίνεται μέσω εντολών προς τους Status Register και Control Register. Το μόνο ερώτημα που απομένει να απαντηθεί είναι το format του byte του Status Register (Καταχωρητής Κατάστασης) και του byte του Control Register (Καταχωρητής Ελέγχου).

2.1 Καταχωρητής Κατάστασης

Η μορφή του byte του Status Register (Καταχωρητής Κατάστασης) φαίνεται παρακάτω. Κάθε ένα από τα 8 bit του καταχωρητή αυτού έχει και δικό του όνομα.

7	6	5	4	3	2	1	0
PARE	TIME	AUXB	KEYL	C/D	SYSF	INPB	OUTB

PARE: Parity Error του τελευταίου byte από το πληκτρολόγιο

1 = τελευταίο byte είχε parity error 0 = χωρίς parity error

TIM: Γενικό time-out

1 = λάθος 0 = δεν υπάρχει time-out λάθος

AUXB: Buffer εξόδου για βοηθητική συσκευή (PS/2 μόνον - ποντίκι)

1 = περιέχει (ο buffer) δεδομένα για την βοηθητική συσκευή

0 = περιέχει δεδομένα για το πληκτρολόγιο

KEYL: Κατάσταση Κλειδώματος Πληκτρολογίου

1 = Πληκτρολόγιο Ελεύθερο 0 = Πληκτρολόγιο κλειδωμένο

C/D: Command / Data

1 = Byte εντολής που γράφθηκε μέσω θύρας 64h

0 = Byte δεδομένο που γράφθηκε μέσω θύρας 60h

SYSF: System Flag

1 = self-test successful 0 = Power-on reset

INPB: Input Buffer Status

1 = Δεδομένα από CPU στον buffer εισόδου

0 = Buffer εισόδου είναι άδειος

OUTB: Output Buffer Status

1 = Δεδομένα από ελεγκτή πληκτρολογίου στον buffer εξόδου

0 = Buffer εξόδου είναι άδειος

Παρακάτω βλέπετε ένα παράδειγμα όπου διαπιστώνετε εάν ο buffer εισόδου είναι γεμάτος (έχει διαβάσει κάποιο byte από την CPU για το πληκτρολόγιο):

```
start:
        IN    al,   64h
        TEST  al,   02h   ; Ελέγχει εάν bit 1  = 1 (input buffer είναι γεμάτος)
        JNZ   start ; Κάποιο byte στον input buffer ⇒ πίσω στο start
```

2.2 Καταχωρητής Ελέγχου

Αποτελείται από 8 bits, που έχουν το όνομα Command bit και έναν αριθμό ενδεικτικό της θέσης τους, ήτοι: C_7 - C_0. Φυσικά είναι δυνατή μόνον η εγγραφή εντολών σε αυτόν, μέσω της θύρας εξόδου 64h.

Οι εντολές που μπορούν να σταλούν είναι κωδικοποιημένες σαν τιμές μεγέθους ενός byte. Στην συνέχεια παρατίθεται μία συνοπτική περιγραφή κάποιων από αυτές.

Κωδικός	**Εντολή**	**Περιγραφή**
AAh	Self-Test	Ο ελεγκτής πληκτρολογίου εκτελεί έναν αυτοέλεγχο και γράφει 55h στον buffer εξόδου, εάν κανένα σφάλμα
Abh	Check Keyboard Interface	Ο ελεγκτής πληκτρολογίου ελέγχει το interface του πληκτρολογίου και γράφει το αποτέλεσμα στον buffer εξόδου (00 = no error, 01 = clock line low, 02 = clock line high, 03 = data line low, 04 = data line high, FFh = general error)
Adh	Disable Keyboard	Απενεργοποιεί το πληκτρολόγιο
AEh	Enable Keyboard	Ενεργοποιεί το πληκτρολόγιο
C0h	Read Input Port	Διαβάζει την θύρα εισόδου και μεταφέρει τα δεδομένα στον buffer εξόδου
C1h	Read out Input Port (low)	Διαβάζει τα bit 3-0 της θύρας εισόδου επαναληπτικά και μεταφέρει τα δεδομένα στα bit 7-4 του καταχωρητή κατάστασης, έως ότου το bit INPB = 1 στον καταχωρητή κατάστασης
C2h	Read out Input Port (high)	Διαβάζει τα bit 7-4 της θύρας εισόδου επαναληπτικά και μεταφέρει τα δεδομένα στα bit 7-4 του καταχωρητή κατάστασης, έως ότου το bit INPB = 1 στον καταχωρητή κατάστασης
D0h	Read Output Port	Διαβάζει την θύρα εξόδου και μεταφέρει τα δεδομένα στον καταχωρητή εξόδου
D1h	Write Output Port	Γράφει το επόμενο byte δεδομένο στην θύρα εξόδου
D2h	Write Keyboard Output Buffer	Γράφει το επόμενο byte δεδομένο στον buffer εξόδου και καθαρίζει το bit AUXB στον καταχωρητή κατάστασης

Για παράδειγμα, έστω ότι θέλετε να γράψετε το byte **12h** στον buffer εξόδου. Αυτό μπορείτε να το επιτύχετε με τις παρακάτω εντολές:

```
        OUT   64h,   D1h        ; Η εντολή "write output port"
WAIT:
        IN    AL,    64h        ; Διάβασε τον καταχωρητή κατάστασης
        TEST  AL,    02h        ; Ελέγχουμε εάν ο buffer εισόδου είναι γεμάτος
        JNZ   WAIT              ; Ο buffer εισόδου είναι γεμάτος, άρα περίμενε
        OUT   60h,   12h        ; Γράψε την τιμή 12h στον buffer εξόδου
```

Ενώ αντίθετα, εάν ότι θέλετε να διαβάσετε το byte **12h** στον buffer εξόδου (προοριζόμενο για τον ελεγκτή του πληκτρολογίου). Αυτό μπορείτε να το επιτύχετε με τις παρακάτω εντολές:

```
        OUT   64h,   D1h        ; Η εντολή "write output port"
WAIT:
        IN    AL,    64h        ; Διάβασε τον καταχωρητή κατάστασης
        TEST  AL,    02h        ; Ελέγχουμε εάν ο buffer εισόδου είναι γεμάτος
        JNZ   WAIT              ; Ο buffer εισόδου είναι γεμάτος, άρα περίμενε
        OUT   60h,   12h        ; Γράψε την τιμή 12h στον buffer εξόδου
```

2.3 Ανάγνωση Κωδικών Σάρωσης από το Πληκτρολόγιο

Σύμφωνα με την παραπάνω παρουσίαση, η ανάγνωση κωδικών σάρωσης από το πληκτρολόγιο είναι σχετικά εύκολη για τα περισσότερα πλήκτρα. Κάθε τέτοιος κωδικός (συνήθως ένα μόνον byte) αποστέλλεται από τον ελεγκτή του πληκτρογίου στο Input Port και από εκεί στον Output Buffer από όπου η CPU μπορεί να το διαβάσει (Σχ. 8.4).

Ο 8255 καταλαβαίνει αυτόματα ότι μία ανάγνωση έλαβε χώρα, ώστε να ειδοποιήσει τον ελεγκτή του πληκτρολογίου για την παραλαβή του επόμενου byte. Επομένως η CPU πρέπει απλά πρώτα να ελέγξει εάν υπάρχει κάποιο byte στον Output Buffer (ελέγχοντας το bit 1 του

Status Register) και μόνον εάν αυτό είναι αλήθεια (το bit αυτό είναι ίσο με 1), προχωρά στην ανάγνωση του υπάρχοντος byte στον Output Buffer:

```
START:
        IN      AL,     64h     ;       Διάβασε Status Register
        TEST    AL,     01h     ;       Έλεγξε OUTB bit
        JZ      START           ;       Δεν υπάρχει καινούργιο byte

        IN      AL,     60h     ;       Διάβασε byte από Output Buffer
```

3. Εργαστηριακή Άσκηση

Γράψετε ένα πρόγραμμα, το διαβάζει 100 διαδοχικά bytes τα οποία αντιστοιχούν σε κωδικούς σάρωσης σύμφωνα με τον τρόπο που σας υποδεικνύεται παραπάνω. Για κάθε περίπτωση θα πρέπει να τυπώνετε στην οθόνη τα εξής προκαθορισμένα μηνύματα:

(α)	"Key Pressed" ή "Key Released"	Εάν ο κωδικός σάρωσης αντιστοιχεί σε πλήκτρο που μόλις πατήθηκε ή απελευθερώθηκε.
(β)	"Left Shift", "Right Shift" ή "Other Key"	Εάν ο κωδικός σάρωσης αντιστοιχεί στο Αριστερό Shift, Δεξιό Shift ή οποιοδήποτε άλλο πλήκτρο.

ΕΒΔΟΜΑΔΑ 9

Σκοπός του εργαστηρίου αυτού είναι να εφαρμόσετε την σημαντική έννοια των Διακοπών (Interrupts), που σας είναι ήδη γνωστή, χρησιμοποιώντας τις αντίστοιχες δυνατότητες που σας παρέχει ο 8086. Επειδή ίσως δεν έχετε διδαχθεί τις κατηγορίες και τον τρόπο εξυπηρετήσεως των διακοπών στον 8086, πρώτα γίνεται μία σύντομη εισαγωγή και μετά παρουσιάζεται η εργαστηριακή άσκηση.

1. Εισαγωγή

Γενικά, ένας Η/Υ είναι ανάγκη να εκτελεί κάποιες ειδικές διαδικασίες, οποτεδήποτε κάποιες συνθήκες υπάρχουν σε ένα πρόγραμμα ή συνηθέστερα σε ένα υπολογιστικό σύστημα. Πιο συχνά τέτοιες συνθήκες παρουσιάζονται στην περίπτωση ύπαρξης περιφερειακών συσκευών, που συνδέονται με τον Η/Υ, αλλά είναι πιο αργές από αυτόν (π.χ. πληκτρολόγιο, οδηγοί δισκετών, κλπ.). Όποτε είναι ανάγκη να εξυπηρετηθεί μία τέτοια συσκευή (ή ειδική συνθήκη), ο Η/Υ και επομένως η Κ.Μ.Ε. θα πρέπει να είναι σε θέση να εκτελέσει την αντίστοιχη ειδική διαδικασία εξυπηρέτησης.

Γενικά υπάρχουν δύο τρόποι για κάτι τέτοιο, όπως γνωρίζετε και από την θεωρία του μαθήματος αυτού: Η Κ.Μ.Ε. ελέγχει διαρκώς εάν κάποια περιφερειακή συσκευή χρειάζεται εξυπηρέτηση (polling), ή η Κ.Μ.Ε. εξυπηρετεί κάποια συσκευή μόνον όταν η τελευταία χρειάζεται εξυπηρέτηση, μέσω μίας κατάλληλης *διακοπής* (interrupt). Είναι προφανές ότι ο δεύτερος τρόπος είναι πολύ αποδοτικότερος από τον πρώτο, αφού η Κ.Μ.Ε. ασχολείται με κάθε συσκευή (ή ειδική συνθήκη) *μόνον* όταν υπάρχει ανάγκη.

Υπάρχουν δύο κατηγορίες διακοπών και επομένως και διαδικασίες εξυπηρέτησης, ανάλογα με την πηγή τους: *Εσωτερικές* και *Εξωτερικές* Διακοπές. Οι πρώτες σχετίζονται με την κατάσταση της ίδιας της Κ.Μ.Ε. ή με το αποτέλεσμα μίας εντολής. Οι δεύτερες προκαλούνται από κάποιο ειδικό σήμα, το οποίο αποστέλλεται στην Κ.Μ.Ε. από κάποιο άλλο σημείο του Η/Υ (π.χ. περιφερειακή συσκευή). Επειδή οι εξωτερικές διακοπές έχουν να κάνουν με τις θύρες Εισόδου/Εξόδου (Ι/Ο), δεν θα ασχοληθούμε εδώ περισσότερο με αυτές. Ο ενδιαφερόμε-

νος αναγνώστης μπορεί να ανατρέξει στις σημειώσεις της θεωρίας του μαθήματος ή την πλούσια βιβλιογραφία επάνω σε αυτό το θέμα. Απλά σημειώνεται ότι οι διαδικασίες εξυπηρέτησης διακοπών είναι ίδιες και για τις δύο κατηγορίες διακοπών.

Μία διαδικασία (ή *ρουτίνα*) διακοπών είναι παρόμοια με οποιαδήποτε κοινή διαδικασία, αφού και για τις δύο η εκτέλεση του προγράμματος διακλαδώνεται προς τον κώδικα της διαδικασίας και στο τέλος της επιστρέφει στο πρόγραμμα αυτό. Η διαφορά έγκειται στο ότι η διαδικασία διακοπών πρέπει να γραφθεί έτσι, ώστε (μετά την εκτέλεση της διαδικασίας αυτής) το διακοπέν πρόγραμμα να συνεχίσει σαν να μην έχει συμβεί τίποτε. Οι επιπτώσεις αυτής της απαίτησης είναι ότι οι διάφορες σημαίες και καταχωρητές που χρησιμοποιούνται σε μία τέτοια διαδικασία, θα πρέπει να αποθηκευθούν, ώστε να μπορούν να επαναφερθούν στην αρχική τους κατάσταση (την στιγμή της διακοπής) και το πρόγραμμα να συνεχίσει την εκτέλεσή του από το σημείο διακοπής.

Επειδή μία διακοπή μπορεί να συμβεί σε οποιαδήποτε χρονική στιγμή και εκτελώντας οποιοδήποτε πρόγραμμα, η διαδικασία εξυπηρέτησής της δεν είναι άμεσα συνδεδεμένη με κάποιο πρόγραμμα. Συνεπώς είναι συνηθισμένο να τοποθετείται σε ένα σαφώς καθορισμένο (απόλυτο) χώρο στην μνήμη.

2. 8086

Στην περίπτωση του 8086 υπάρχουν δύο είδη διακοπών, ανάλογα με το εάν είναι δυνατόν να μπλοκαρισθούν (και επομένως να αγνοηθούν) ή όχι: Οι *Αγνοήσιμες* (Maskable) και οι *Μη-Αγνοήσιμες* (Non-Maskable). Οι πρώτες μπορούν να αγνοηθούν εάν ο 8086 προγραμματισθεί με έναν συγκεκριμένο τρόπο, ενώ οι δεύτερες δεν μπορούν να αγνοηθούν ποτέ.

Εάν μία διακοπή δεν έχει απενεργοποιηθεί, τότε ο 8086 ενεργεί ως εξής:

1. Ολοκληρώνει την εκτέλεση της εντολής που εκτελούσε την στιγμή της διακοπής.

2. Σπρώχνει στην Σωρό τα περιεχόμενα του Καταχωρητή Σημαιών (Flag Register), **CS** και **IP**.

3. Τοποθετεί στους CS και IP τα περιεχόμενα μίας Διπλής Λέξης (Double Word = 4 bytes), της οποίας η διεύθυνση καθορίζεται από τον τύπο της διακοπής.

4. Καθαρίζει (μηδενίζει) τις σημαίες **IF** και **TF**, απενεργοποιώντας τις αγνοήσιμες διακοπές.

Κατά την επιστροφή, τα παλιά περιεχόμενα των καταχωρητών Σημαιών, **CS** και **IP** παίρνονται από την Σωρό και τοποθετούνται σε αυτούς.

Η Διπλή Λέξη που περιέχει τα καινούργια περιεχόμενα των **CS** και **IP**, όταν εξυπηρετείται η διακοπή, ονομάζεται *Δείκτης* ή *Διάνυσμα Διακοπής (Interrupt Vector)*. Κάθε τύπος διακοπής χαρακτηρίζεται από έναν αριθμό μεταξύ 0 και 255, ενώ η διεύθυνση στην μνήμη όπου είναι αποθηκευμένο το αντίστοιχο Διάνυσμα Διακοπής μπορεί να βρεθεί πολλαπλασιάζοντας τον αριθμό της διακοπής με το 4. Π.χ., εάν ο αριθμός της διακοπής είναι 9, τότε το Διάνυσμα Διακοπής θα βρίσκεται στα bytes των διευθύνσεων από **00024h** έως και **00027h**. Για αυτόν τον λόγο σε κάθε Η/Υ με 8086, τα πρώτα 1.024 bytes της Κύριας Μνήμης καταλαμβάνονται από τα διανύσματα διακοπών (256 διακοπές x 4 bytes = 1.024 bytes).

Από αυτές τις 256 διακοπές οι πέντε πρώτες (0 - 4) είναι δεσμευμένες από τον 8086 και μερικές άλλες μπορεί να χρησιμοποιούνται από το υπάρχον λειτουργικό σύστημα (π.χ. MS-DOS). Οι υπόλοιπες είναι διαθέσιμες στον προγραμματιστή εφαρμογών.

Επίσης είναι δυνατόν να χρησιμοποιηθεί η εντολή **INT <n>**, όπου **n** ο αριθμός της διακοπής, προκειμένου να εκτελεσθεί η αντίστοιχη διαδικασία εξυπηρέτησης με τον τρόπο που αναφέρθηκε προηγουμένως. Το πλεονέκτημα της χρήσης αυτής της εντολής είναι ότι μπορούν να αποσφαλματωθούν και ρουτίνες εξυπηρέτησης διακοπών που δεν έχουν λάβει χώρα στην πραγματικότητα.

Ο προγραμματιστής διαδικασιών ρουτινών εξυπηρέτησης δεν θα πρέπει να ξεχνάει ότι κατά την κλήση της ρουτίνας του στην Σωρό έχουν τοποθετηθεί όχι μόνον τα παλιά περιεχόμενα

του **CS** και του **IP**, αλλά και του καταχωρητή Σημαιών. Επομένως ένα **RET** στο τέλος της διαδικασίας δεν είναι αρκετό, αλλά απαιτείται μία ειδική εντολή, η **IRET**. Η τελευταία επαναφέρει από την Σωρό τις παλιές τιμές *και* των τριών καταχωρητών.

Τέλος όπως φαίνεται παραπάνω από την τέταρτη ενέργεια του 8086, η σημαία IF μηδενίζεται, πράγμα που σημαίνει ότι πλέον όλες οι αγνοήσιμες διακοπές μπλοκάρονται, μέχρι την επιστροφή από την ρουτίνα εξυπηρέτησης της διακοπής. Εν συντομία, μία **INT <n>** ισοδυναμεί με τις παρακάτω εντολές:

CLI	;	Μπλοκάρισμα των αγνοήσιμων διακοπών
PUSHF	;	Τοποθέτηση του καταχωρητή σημαιών στην Σωρό
CALL DWORD PTR 4*n	;	Κλήση της ρουτίνας εξυπηρέτησης με το διάνυσμα διακοπής να βρίσκεται από την διεύθυνση μνήμης *4 * n* θέσεις.

3. Συλλαμβάνοντας Διακοπές

Εφ' όσον η διεύθυνση του ανύσματος της κάθε διακοπής είναι γνωστή, είναι δυνατόν να αλλάξουμε το περιεχόμενο των αντιστοίχων τεσσάρων bytes (για τους **CS** και **IP**), ώστε να εκτελεσθεί κάποια άλλη διαδικασία εξυπηρέτησης αυτής της διακοπής στην θέση της αρχικής. Πιο συχνά όμως, αυτό που μας ενδιαφέρει είναι να παρεμβάλουμε ένα τμήμα δικού μας κώδικα, πριν συνεχίσουμε με την κανονική (παλιά) διαδικασία εξυπηρέτησης.

Για τον σκοπό αυτό, θα πρέπει να εκτελέσουμε τα παρακάτω βήματα:

1. Να αποθηκεύσουμε το παλιό διάνυσμα της διαδικασίας εξυπηρέτησης της διακοπής.

2. Να τοποθετήσουμε στην θέση του παλιού διανύσματος, ένα καινούργιο που να δείχνει στην καινούργια διαδικασία εξυπηρέτησης της διακοπής.

3. Η καινούργια διαδικασία εξυπηρέτησης διακοπής να καλεί στο τέλος της την παλιά διαδικασία εξυπηρέτησης, με τις εντολές:

   PUSHF
   CALL <Διεύθυνση_Παλιάς_Διαδικασίας_Εξυπηρέτησης>
   (γιατί;)

4. Η αμέσως επόμενη (και τελευταία) εντολή να είναι η **IRET**.

Εννοείται ότι η καινούργια διαδικασία εξυπηρέτησης θα πρέπει να έχει δηλωθεί ως **FAR** (γιατί;). Υπάρχουν πλέον τέσσερα προβλήματα.

Το **πρώτο** είναι πώς να αποθηκεύσουμε το παλιό διάνυσμα διευθύνσεως. Αυτό λύνεται με τον ορισμό κάποιας μεταβλητής στο τμήμα δεδομένων (μνήμη) του προγράμματος, τύπου Double Word (=4 bytes), κάτι που μπορεί να επιτευχθεί με την οδηγία **DD**, π.χ.:

```
OLD_VECTOR      DD     ?
```

Το **δεύτερο** πρόβλημα είναι το πώς να διαβάσουμε το παλιό διάνυσμα, ώστε να το αποθηκεύσουμε στην παραπάνω μεταβλητή. Για τον σκοπό αυτό μπορούμε να χρησιμοποιήσουμε την διακοπή **21h** του MS-DOS, υπηρεσία **35h** (άρα **AH = 35h**) και με τον **AL** να περιέχει τον *αριθμό διακοπής* που μας ενδιαφέρει. Σαν αποτέλεσμα, τα πρώτα δύο bytes του παλιού διανύσματος (που θα αντιστοιχούσαν στον **IP**) τοποθετούνται στον **BX**, και τα δύο επόμενα (που θα αντιστοιχούσαν στον **CS**) τοποθετούνται στον **ES**.

Η τοποθέτηση των περιεχομένων των **BX** και **ES** στην μεταβλητή **OLD_VECTOR** γίνεται με δύο **MOV**:

```
MOV  WORD PTR OLD_VECTOR,      BX
MOV  WORD PTR OLD_VECTOR[2],   ES
```

Το **τρίτο** πρόβλημα είναι η εγγραφή της διεύθυνσης της καινούργιας διεύθυνσης εξυπηρέτησης στην θέση του παλιού ανύσματος. Αυτό γίνεται με την διακοπή **21h** του MS-DOS, υπη-

ρεσία **25h** (άρα **AH = 25h**) και με τον **AL** να περιέχει τον αριθμό διακοπής που μας ενδιαφέρει. Εδώ χρειαζόμαστε και την διεύθυνση της καινούργιας διεύθυνσης εξυπηρέτησης, με τον **DS** και τον **DX** την μετατόπιση από την αρχή του κώδικα. Αυτό μπορεί να γίνει με τις παρακάτω εντολές (υποθέτοντας ότι το όνομα της καινούργιας διαδικασίας είναι **INTERCEPT_IT**):

```
PUSH DS                          ; Αποθήκευσε τιμή DS
PUSH CS                          ; DS = CS
POP  DS                          ; εάν όλα, στο ίδιο τμήμα Κώδικα
                                 ; MOV AX, seg INTERCEPT_IT
                                 ; MOV DS, AX
                                 ; εάν σε διαφορετικό τμήμα Κώδικα
MOV  DX, offset CS:INTERCEPT_IT
MOV  AH, 25h
MOV  AL, n                       ; n = αριθμός διακοπής που μας
                                 ; ενδιαφέρει
INT  21h
POP  DS                          ; DS ξαναδείχνει στο τμήμα δεδομένων
```

Το **τέταρτο** πρόβλημα είναι ότι η καινούργια διαδικασία εξυπηρέτησης δεν είναι κατ' ανάγκην συνεχώς διαθέσιμη, μετά την λήξη εκτέλεσης του προγράμματός μας. Άρα θα πρέπει, είτε να επαναφέρουμε την διεύθυνση της παλιάς διαδικασίας διακοπής στην θέση του ανύσματος διακοπής, είτε να την κάνουμε πάντοτε διαθέσιμη μέσω μίας υπηρεσίας του MS-DOS, μετατρέποντάς την σε **TSR** (= Terminate and Stay Resident). Το τελευταίο επιτυγχάνεται μέσω των παρακάτω εντολών:

```
MOV  AH,  31h
MOV  DX,  100h      ; DX = Απαιτούμενη μνήμη σε παραγράφους
                    ;      με 1 παράγραφο = 16 bytes
                    ;      εδώ 100h παράγραφοι για σιγουριά
INT  21h
```

Τέλος το πρόγραμμά σας θα πρέπει να είναι **.COM**, ώστε να καταλαμβάνει τον μικρότερο δυνατό χώρο στην μνήμη. Αυτό γίνεται με την χρήση **INT 20h** στο τέλος του προγράμματός μας και επιλογή της κατάλληλης παραμέτρου του Linker.

4. Εργαστηριακή Άσκηση

Προκειμένου να κάνετε μία προσθήκη σε μία ήδη υπάρχουσα ρουτίνα εξυπηρέτησης, θα πρέπει να γράψετε ένα πρόγραμμα, που εκτελεί τα παρακάτω βήματα:

1. Διαβάζει το αντίστοιχο διάνυσμα διακοπής από την μνήμη και το αποθηκεύει σε μία άλλη ξεχωριστή θέση μνήμης (στο Τμήμα Δεδομένων).

2. Διαβάζει το αντίστοιχο διάνυσμα διακοπής από την μνήμη και το αποθηκεύει σε μία άλλη ξεχωριστή θέση μνήμης (στο Τμήμα Δεδομένων), για την διακοπή μετά από μία Υπερχείλιση (Overflow), μέσω της **INTO**.

3. Εκτελεί κάποια πράξη, που προκαλεί υπερχείλιση.

4. Εκτελεί αμέσως μετά την **INTO**.

5. Εάν η Σημαία **OF** = 1, τότε εκτελείται η διαδικασία εξυπηρέτησης.

6. Επιστρέφει στο κυρίως πρόγραμμα και τυπώνει ένα ανάλογο μήνυμα.

7. Επιστρέφει στο MS-DOS, αφού πρώτα τοποθετήσει την διεύθυνση της παλιάς ρουτίνας εξυπηρέτησης.

```
stack segment para stack
        db 200 dup(0)
        top_of_stack  label    BYTE
stack ends

data  segment
        return_flag     db      0               ; 0 εάν OK, 1 εάν έλαβε χώρα διακοπή
        old_int         label   DWORD
        old_ip  dw      ?
        old_cs  dw      ?
        OK_message      db      "Everything OK", 13, 10, "$"
data  ends

kodikas segment  public
        assume ds:data, cs:kodikas, ss:stack

MAIN_PROG   PROC  FAR
   mov      ax, stack                       ; Άχρηστο
   mov      ss,  ax                         ; Άχρηστο
   mov      sp,  offset top_of_stack        ; Άχρηστο
   mov      ax,  data
   mov      ds,  ax                         ; DS δείχνει στην αρχή του Data Segment

   mov      al,  04                         ; Πάρε παλιό διάνυσμα για INT 4
   mov      ah,  35h
   int      21h                             ; ES:BX περιέχουν τώρα το παλιό διάνυσμα

   mov      old_ip, bx
   mov      ax,  es
   mov      old_cs, ax
   push     ds                              ; Αποθήκευσε το DS

   push     cs
   pop      ds                              ; DS = CS
   mov      dx,  offset CS:INTERCEPT_IT     ; offset (IP) του INTERCEPT_IT
```

```
   mov      al,  04
   mov      ah,  25h
   int      21h                          ; Τοποθέτησε καινούργιο διάνυσμα για INT 4

   pop      ds                           ; Επαναφορά του DS

   mov      bl,  120
   mov      al,  120
   mul      bl
   INTO                                  ; Εκτέλεσε INT 4, εάν OF = 1

   mov      ax,  old_cs
   push     ds
   mov      ds,  ax
   mov      dx,  old_ip
   mov      al,  04
   mov      ah,  25h                     ; Επαναφορά παλιού διανύσματος για INT 4
   int      21h
   pop      ds
   mov      dl,  return_flag
   add      dl,  '0'
   mov      ah,  02h
   int      21h                          ; Τύπωσε την τιμή της return_flag
   mov      ah,  4Ch                     ; Έξοδος στο MS-DOS
   int      21h
MAIN_PROG   endp

;-------------------------------------------------------------------------------------------------
; Procedure: INTERCEPT_IT       Η διαδικασία εξυπηρέτησης της διακοπής 4
;-------------------------------------------------------------------------------------------------
INTERCEPT_IT    PROC  FAR
   mov      return_flag, 1
   iret
INTERCEPT_IT  endp

kodikas ends
```

```
end   MAIN_PROG
```

ΕΒΔΟΜΑΔΑ 10

Σκοπός του εργαστηρίου αυτού είναι να έχετε μία εμπειρία από τον προγραμματισμό κατάλληλων ολοκληρωμένων περιφερειακών για σειριακή επικοινωνία μεταξύ Η/Υ.

1. Εισαγωγή

Η σημασία της επικοινωνίας μεταξύ Η/Υ είναι ένα θέμα, η σπουδαιότητα του οποίου είναι γνωστή και κατανοητή από όλους. Υπάρχουν πολλοί τρόποι για να καταστεί δυνατή αυτή η επικοινωνία, αλλά κατά βάση, αυτό το οποίο αποστέλλεται από τον έναν Η/Υ στον άλλο είναι μία μονάδα πληροφορίας, βάσει κάποιου προσυμφωνημένου πρωτοκόλλου.

Όπως ήδη γνωρίζετε από άλλα σχετικά μαθήματα, με τον όρο *Πρωτόκολλο* αναφερόμαστε σε ένα σύνολο από αποδεκτούς κανόνες και ενέργειες οι οποίες πρέπει να λάβουν χώρα, προκειμένου να ολοκληρωθεί κάποια επιθυμητή λειτουργία.

Ανάλογα με το μέγεθος της αποστελλόμενης μονάδας πληροφορίας, διακρίνουμε δύο είδη επικοινωνίας:

1. **Σειριακή:** Η μονάδα της αποστελλόμενης πληροφορίας είναι το bit.

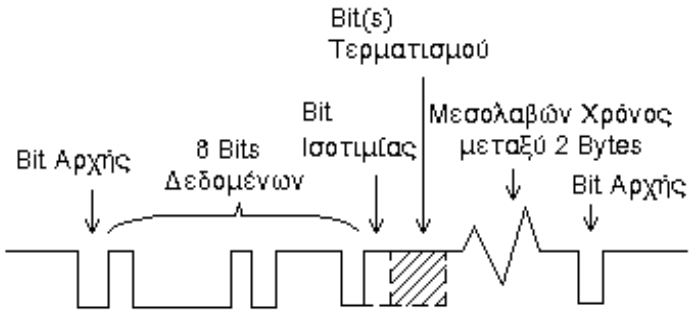
2. **Παράλληλη:** Η μονάδα της αποστελλόμενης πληροφορίας είναι το byte.

Το πλεονέκτημα του δεύτερου τύπου επικοινωνίας είναι ότι ανά κάθε χρονική στιγμή αποστέλλεται ποσό πληροφορίας οκταπλάσιο, από ό,τι στον πρώτο τύπο. Το μειονέκτημα φυσικά είναι ότι απαιτείται τουλάχιστον οκταπλάσιο πλήθος από συνδέουσες γραμμές μεταξύ των επικοινωνούντων Η/Υ. Ειδικότερα για απομακρυσμένους Η/Υ το μειονέκτημα αυτό γίνεται εξαιρετικά σημαντικός παράγων, λόγω κόστους. Για αυτόν τον λόγο, σήμερα χρησιμοποιείται μόνον μέθοδοι σειριακής επικοινωνίας όταν πρόκειται για απομακρυσμένους Η/Υ.

Εδώ όμως παραμένει ένα κλασικό πρόβλημα, που υπάρχει κατά την επικοινωνία οποιωνδήποτε δύο οντοτήτων (είτε μηχανήματα, είτε διεργασίες, ακόμα και άνθρωποι): **Συγχρονισμός**.

Εάν υποθέσουμε ότι οι δύο Η/Υ είναι ακριβώς ίδιοι και ότι το μέσον επικοινωνίας (π.χ. καλώδια) δεν δημιουργούν ποτέ προβλήματα θορύβου, κλπ, τότε μπορούμε να χρησιμοποιήσουμε κάποια από τις πολλές μεθόδους **Σύγχρονης** επικοινωνίας. Εδώ αποστέλλεται συνήθως κάποια προσυμφωνημένη ομάδα από bits, για να δηλωθεί η έναρξη επικοινωνίας και κάποια άλλη ομάδα στο τέλος για να δηλωθεί το πέρας της επικοινωνίας.

Στην πράξη όμως τέτοιες συνθήκες είναι δύσκολο να υπάρξουν - ιδίως σε φθηνά μέσα μετάδοσης, όπως οι τηλεφωνικές γραμμές. Για αυτόν τον σκοπό, έχουν επινοηθεί μέθοδοι **Ασύγχρονης** επικοινωνίας, οι οποίες σκοπό έχουν να καταστήσουν αξιόπιστο (ως ένα σημείο) ένα - κατά τα άλλα - αναξιόπιστο μέσο μετάδοσης. Ο χρονισμός μεταξύ δύο διαδοχικών bytes είναι άνευ σημασίας, αλλά ο χρονισμός της ακολουθίας των bits που συνθέτουν το byte είναι σημαντικός. Παράδειγμα από σειριακή μετάδοση ενός byte φαίνεται στο Σχ. 1.



Σχ. 10.1 Σειριακή Μετάδοση ενός Byte

Όπως φαίνεται από το Σχ. 10.1, υπάρχουν δύο στάθμες που αντιστοιχούν σε λογικό 0 και λογικό 1. Συνήθως χρησιμοποιείται θετική λογική και επομένως η υψηλή στάθμη αντιστοιχεί σε λογικό 1 ή Marking και η χαμηλή σε λογικό 0 ή Spacing. Όταν δεν αποστέλλονται δεδομένα, η γραμμή μετάδοσης διατηρείται σε υψηλή κατάσταση (λογικό 1).

Το πρωτόκολλο μετάδοσης δεδομένων έχει τότε ως εξής:

1. Όταν πρόκειται να αποσταλεί ένα byte δεδομένων, το σήμα πέφτει στο λογικό 0, ώστε έτσι να αποσταλεί το Bit Αρχής (Start Bit).

2. Μετά αποστέλλονται τα bits δεδομένων (7 ή 8 συνήθως).

3. Στην συνέχεια ακολουθεί ένα προαιρετικό bit Ισοτιμίας (Parity bit).

4. Αυτό ακολουθείται από ένα ή δύο bits Τερματισμού (Stop bits).

Επειδή ο συγκεκριμένος τρόπος μετάδοσης είναι ασύγχρονος, είναι δυνατόν να μεσολαβήσει οποιοδήποτε μεταβλητό χρονικό διάστημα, έως την αποστολή του επόμενου byte, οπότε επαναλαμβάνονται τα παραπάνω βήματα.

Βέβαια, τα παραπάνω αποτελούν μέρος μόνον του όλου προβλήματος, δεδομένου ότι και οι δύο σταθμοί (εκπομπής και μετάδοσης) πρέπει να χρησιμοποιούν το ίδιο πρωτόκολλο (π.χ. ίδιο αριθμός bits ανά byte, ίδια ταχύτητα, ίδιος αριθμός stop bits, ίδιο parity bit, κλπ). Επίσης, πρέπει να λαμβάνεται υπ' όψιν και η πιθανότητα δημιουργίας σφαλμάτων λόγω θορύβου στην γραμμή μετάδοσης, κ.ο.κ.

2. 8250 - Ένα Τυπικό UART

Η σειριακή επικοινωνία είναι τόσο πολύπλοκη, ώστε να έχουν σχεδιασθεί ειδικά ολοκληρωμένα τα οποία και να αναλαμβάνουν να φέρουν αυτήν την λειτουργία εις πέρας. Ένα τέτοιο ολοκληρωμένο ονομάζεται **Γενικός Ασύγχρονος Δέκτης / Εκπομπός (UART = Universal Asynchronous Receiver / Transmitter)**. Ειδικότερα, ο αρχικός IBM PC χρησιμοποιούσε το 8250 της INTEL. Αυτό σήμερα έχει εν πολλοίς αντικατασταθεί από το 16550, αλλά είναι δυνατόν να καταλάβετε κάποια πράγματα, αναφερόμενοι στο 8250.

Το MS-DOS υποστήριζε αρχικά δύο θύρες επικοινωνίας (που έγιναν στην συνέχεια τέσσερις), με την επωνυμία **COM1** και **COM2** - άρα και δύο UART. Οι διευθύνσεις **0040:0000**

και **0040:0002** αντιστοιχούν σε δύο words, της περιοχής όπου το BIOS κρατάει κάποια δεδομένα του. Πιο συγκεκριμένα, η πρώτη λέξη περιέχει το I/O port address για το **COM1** και η δεύτερη το I/O port address για το **COM2**. Οι τιμές αυτών των λέξεων αρχικά είναι **3F8h** και **2F8h**, αντίστοιχα. Εάν επιθυμούμε να εναλλάξουμε αυτές τις αντιστοιχίες, απλά εναλλάσσουμε τα περιεχόμενα αυτών των λέξεων.

Το 8250 έχει δέκα προγραμματιζόμενους καταχωρητές του ενός byte, με τους οποίους και ελέγχει την αντίστοιχη σειριακή θύρα. Αυτοί όμως φαίνονται από τον 8086 σε επτά μόνον διευθύνσεις θυρών (I/O port addresses) και για αυτό χρησιμοποιείται το bit 7 της θύρας **3FBh** ώστε να επιλεγούν κάποιοι από αυτούς τους καταχωρητές. Προκειμένου για το **COM1** αυτοί οι καταχωρητές είναι:

3F8h Καταχωρητής Κράτησης Εκπομπού (ΕΞΟΔΟΣ, bit 7=0 στο **3FBh**)
3F8h Καταχωρητής Δεδομένων Δέκτη (ΕΙΣΟΔΟΣ, bit 7=0 στο **3FBh**)
3F8h Διαιρέτης Ταχύτητας Baud (Low Byte) (ΕΞΟΔΟΣ, bit 7=1 στο **3FBh**)
3F9h Διαιρέτης Ταχύτητας Baud (High Byte) (ΕΙΣΟΔΟΣ, bit 7=1 στο **3FBh**)
3F9h Καταχωρητής Ενεργοποίησης Διακοπών (ΕΞΟΔΟΣ, bit 7=0 στο **3FBh**)
3FAh Καταχωρητής Αναγνώρισης Διακοπής (ΕΙΣΟΔΟΣ)
3FBh Καταχωρητής Ελέγχου Γραμμής (ΕΞΟΔΟΣ)
3FCh Καταχωρητής Ελέγχου Modem (ΕΞΟΔΟΣ)
3FDh Καταχωρητής Κατάστασης Γραμμής (ΕΙΣΟΔΟΣ)
3FEh Καταχωρητής Κατάστασης Modem (ΕΙΣΟΔΟΣ)

Από τους παραπάνω καταχωρητές, μόνον ένα υποσύνολο είναι απαραίτητο για απλή σειριακή επικοινωνία. Οι καταχωρητές που έχουν σχέση με modem φυσικά έχουν σημασία μόνον για την περίπτωση που χρησιμοποιείται ανάλογη συσκευή, ενώ εκείνοι που έχουν σχέση με διακοπές μόνον όταν έχουμε προγράμματα που χρησιμοποιούν διακοπές για την εξυπηρέτηση σειριακής επικοινωνίας. Ο Καταχωρητής Κράτησης Εκπομπού κρατάει το byte που πρόκειται να σταλεί, ενώ ο Καταχωρητής Δεδομένων Δέκτη το byte δεδομένων που παραλήφθηκε πιο πρόσφατα.

Ο Διαιρέτης Ταχύτητας Baud είναι ένας αριθμός Χ, που διαιρεί την συχνότητα του ρολογιού του Η/Υ (11900 Hz) για να πάρουμε την επιθυμητή τελική ταχύτητα: 11900/Χ. Αυτός ο α-

ριθμός περιγράφεται με δύο bytes σε αντίστοιχους από τους παραπάνω καταχωρητές. Για ταχύτητα 1200 baud τα αντίστοιχα bytes προφανώς σχηματίζουν την λέξη **0060h**.

Είναι επίσης προφανές ότι υπάρχει και ένας *Καταχωρητής Ολίσθησης (Shift Register)* για σειριοποίηση κάθε byte πριν την εκπομπή και ένας για αποσειριοποίηση κάθε byte μετά την παραλαβή του τελευταίου.

3. Εργαστηριακή Άσκηση

Όπως έχει ήδη καταστεί σαφές, η σειριακή επικοινωνία δεν είναι κάτι το απλό. Για τον λόγο αυτό θα υποθέσετε ότι η επιθυμητή σειριακή θύρα (εδώ η **COM1**) έχει ήδη αρχικοποιηθεί. Εσείς το μόνο που έχετε να κάνετε είναι να διαβάζετε διαρκώς το byte το οποίο και παραλαμβάνεται από τον αντίστοιχο καταχωρητή του UART. Για να υπάρχει νόημα στην άσκηση, θα ελέγχετε εάν το byte που διαβάσατε (8086) από τον παραπάνω καταχωρητή είναι διαφορετικό από αυτό που διαβάσατε αμέσως πριν, και εάν ναι, τότε και μόνον τότε θα το δεχόσαστε και θα το τυπώνετε στην οθόνη σαν χαρακτήρα αφού προσθέσετε σε αυτό τον ASCII κωδικό του μηδενός. Για μεγαλύτερη ευκρίνεια, στο τέλος θα τυπώνετε και τους χαρακτήρες CR και LF, ώστε να αλλάξετε γραμμή.

Επειδή δε χρειαζόσαστε κάποια συσκευή η οποία να στέλνει δεδομένα στο **COM1**, θα χρησιμοποιήσετε για αυτόν τον σκοπό το ποντίκι του Η/Υ σας. Θα πρέπει τότε να βλέπετε έναν (όχι κατ' ανάγκην ίδιο) χαρακτήρα στην οθόνη όταν μετακινείτε το ποντίκι σας. Αντιθέτως, δεν θα πρέπει να τυπώνεται τίποτε στην οθόνη, εάν το ποντίκι δεν μετακινείται. Για να προλάβετε να ελέγξετε την συμπεριφορά του προγράμματός σας, θα ρυθμίσετε τα παραπάνω να εκτελούνται 500 φορές.