קלט / פלט ברמת המכונה ב-x86

עד כה עשינו קלט / פלט דרך ה-BIOS (ישירות או בעקיפין דרך DOS). למען האמת העברנו בקשות קלט / פלט + אינפורמציה לתוכניות אסמבלי שנכתבו ע"י היצרן. כל תוכניתן אסמבלי יכול לכתוב את הרוטינות הללו, אם כי הרוטינות עשויות להיות תלויות בפלטפורמה הספציפית שבה מדובר, למרות שבפירוש יש נסיון לממש סטנדרטיזציה גם כאן. תוכנית ה-keyboard המובאת להלן, keyb6.asm, היא תוכנית שלקחה על עצמה את תפקידי רוטינות ה-BIOS, והתנהגות שלה היתה מעט שונה על מקלדות שונות. ביחוד בכל הקשור למקשי החיצים, Home, Ins וכו'. בטרמינולוגיה מקצועית אפשר לומר שהתוכנית מכילה בתוכה Device Driver של המקלדת.

שיטות קלט / פלט

התיעוד של חברת Intel קובע שיש שתי שיטות קלט / פלט עיקריות:

- קלט / פלט דרך ports (ערוצים?)

- Memory Mapped I/O (קלט / פלט ממופה לכתובות).

לדעתי אפשר להוסיף קטגוריה שלישית: Direct Memory Access (DMA).
אנשי Intel כנראה קטלגו שיטה זו במסגרת הקטגוריה של ports.

ports הם מעין מרחב כתובות (נפרדת מהזכרון) המיצגים (בעיני התוכניתן לפחות) מעין חוטים עם 2 קצוות: קצה ב-CPU וקצה שני בהתקן ק/פ (בדרך כלל chip אחר). עד כמה שהבנתי, יתכן שלפחות היסטורית זה היה פעם גם המימוש הפיזי - יתכן שחלק מהקוים הכסופים על ה-mother board הינם (או היו) מימושים של ports. השאלה איך מתבטא ports בחומרה הוא חסר חשיבות לתוכניתן - הדבר מן הסתם השתנה בין גירסת PC אחד למשנהו, ובין יצרן ליצרן. מרחב הכתובות של ה-ports הוא בין 0 ל-65535 (רובם לא בשימוש).

בכל מקרה אם port מחובר למשהו או לא - זה עיניין של החומרה - אי אפשר בתוכנה "לחבר" port למשהו.

יש לשים לב: מספרי port הם חיבורים ישירים וגלויים ל-CPU. ישנם חיבורים במחשב בין chips שאינם ה-CPU והם אינם נכללים בקטגוריה הזו, שכן אינם גלויים ל-CPU (או, אם תרצו, לתוכניתן האסמבלי).

Memory Mapped I/O הוא מציאות שבה כתובות זכרון מסוימות מנותות להתקני ק/פ במקום לזכרון. הגישה הזו מחייבת מימוש בחומרה. שום תוכנה אינה יכולה "לממש" Memory Mapped I/O בכוחות עצמה. היצרן בונה את המחשב כך שכתיבה / קריאה לכתובות מסוימים משמעותם למעשה ק/פ. הדוגמא הקלאסית הוא כתיבה / קריאה לזכרון המסך: הסגמנטים A000h (כתובת פיזית 640k), B000h (כתובת פיזית 704k), B800h (כתובת פיזית 736k) הם כתובות המנותות לזכרון של כרטיס המסך של המחשב. תוכנית הכותבת / קוראת ל"שטחי זכרון" הללו למעשה כותבת / קוראת לזכרון של כרטיס המסך, והדבר מתבטא בשינוי התצוגה של המסך.

DMA הוא מאין שילוב של שני הקודמים: דרך ports ה-CPU מנחה chip של DMA (Intel 8237A) לקרוא ממקום מסוים בהתקן (נניח Track מסוים בדיסק) ליעד מסוים בזכרון (או להיפך). המידע אינו עובר דרך ה-CPU, דבר שהיה אמור לחסוך overhead ולאפשר ל-CPU לעסוק כביכול בדברים אחרים (במקביל ל-DMA).

<u>פקודות מכונה לקריאה / כתיבה ל-ports</u>:

יש מספר כאילו, אבל הם למען האמת גירסאות של הפקודות IN, OUT.

<u>מבנה הפקודה IN</u>:

IN AL, מספר port

או

IN AX, DX

או

IN EAX, DX (386 ואילך)

קורא בית או מילה או מילה כפולה מ-port לאוגר האקומולטור (AL, AX, EAX). על משמעות של קריאה של יותר מבית אחד נראה בהמשך.

מספר port הוא <u>קבוע</u> בין 0 ל-255.

מבנה הפקודה OUT:

OUT מספר port או DX או AL

או

AX

או

EAX (386 ואילך)

כותב בית או מילה או מילה כפולה מהאקומולטור (AL, AX ,EAX) ל-port. על משמעות של כתיבה של יותר מבית אחד נראה בהמשך.

מספר port הוא קבוע בין 0 ל-255.

דוגמאות:

```
IN AL,60h
```

קריאת תו מהמקלדת

```
MOV DX,378h
OUT DX,AL
```

שליחת תו למדפסת.

מספר port המופיע במפורש בפקודת IN אן OUT חייב להיות קבוע בין 0 ל-255. אפשר לגשת לכל מספר port דרך הגירסאות של הפקודות המשתמשות בתוכן DX לציון מספר ה-port. אולם ל-port עם מספר גדול יותר מ-255 אפשר לגשת רק ע"י גירסאות ה-DX של הפקודות.
מספר ה-port המירבי ב-DX הוא 65535. אין מספרי port גדולים יותר (אין תמיכה בגירסה של הפקודה עם EDX, גם לא בפנטיום).

הגדלים של האופרנדים אינם בהכרח שווים כמובן (שכן מספר ה-port או DX הם מעין פוינטרים).

כאשר משתמשים ב-AX או EAX בפקודות IN ו-OUT, ה-bytes המשמעותיים יותר של האוגר נכתבים למספרי ports עוקבים, כאילו היו כתובות. לדוגמא, אם מתבצע:

```
OUT 200,AX
```

אז AL נכתב לתוך port מספר 200, AH נכתב לתוך port מספר 201.

אי אפשר סתם לשלוח תוים ל-port.

יש צורך לעקוב אחרי פרוטוקול של בדיקות סטטוס ואישרור קבלה.

רוב מספרי ה-port אינם מנוצלים, ואינם מחוברים לדבר. שום תקלה אינה נגרמת כאשר קוראים זבל מ-port שאינו מחבר לכלום או כותבים ל-port שאינו מחבר לכלום.

Support Chips

Support Chips הם מעין מיקרו פרוסטורים קטנים ל"משימות מיוחדות". הם מורידים משימות מה-CPU על מנת שלפשט את מימוש ה-CPU. הם פשוטים יותר מה-CPU ובדרך כלל ניתנים "לתכנות" בצורה מוגבלת - אפשר לשנות את התנהגותם ע"י כתיבה לאוגרים שלהם דרך ports. דוגמא ל- Support Chip הוא ה-Timer ששולח פולס ל-CPU 18.207 פעמים בשניה, ומאפשר ל-CPU לממש את פסיקה מספר 8. החסרון של שימוש ב-Support Chips הוא שהדבר מאיט את המערכת ככלל.

נתרכז בשני Support Chips אחרים: בקר הפסיקות ובקר ההתקנים החיצוניים.

דוגמא לקריאת התקן דרך ports: המקלדת (פסיקה מספר 9).

פסיקה מספר 9 הוא פסיקת התומרה של המקלדת.
המערכת של המחשב בנויה כך שכל לחיצה / שחרור של מקש במקלדת גורם לפסיקת חומרה 9. הקריאה של איזה מקש מדובר נעשה דרך ports. למעשה תפקידה של פסיקה 9 לשמש מנגנון העברת אינפורמציה של המקלדת לתוכנה. ה-ISR של 9 מעביר את תוכן המקש לזכרון של ה-BIOS, ופסיקת התוכנה 16h לוקחת אותו משם.

איך בדיוק הרעיון הזה מיושם תלוי מאד באיזה דגם של מחשב מדובר. התאור הבא מתאים ל-8086. בדגמים שבאו אחר כך חלו שינויים, לא נעמוד עליהם כאן.

התהליך של העברת מידע מהמקלדת הוא הבא:
בתוך המקלדת יש מיקרו-פרוסטור קטן המבחין בכל לחיצה ובכל שחרור של מקש. הוא מאותת דרך חיבור מיוחד למחשב שיש לו מידע למחשב (החיבור הזה אינו גלוי ל-CPU, לכן אין לו מספר port). החיבור הזה הוא ל-chip שמשמש בקר התקנים חיצוניים. המקלדת אינה, כמובן, ההתקן היחיד שמחובר לבקר הזה.

בקר ההתקנים מחובר ל-CPU (בין השאר) ע"י ה-ports 60h, ו-61h.

port מספר 60h משמש להעברת ה-scan code, שבקר ההתקנים מקבל מהמקלדת ל-CPU. ה-scan code הוא מספר קטן או שווה ל-127. הביט המשמעותי ביותר ב-port הוא איפוא פנוי והוא משמש להבדיל בין לחיצה לשחרור. 0 = לחיצה, 1 = שחרור.

port מספר 61h משמש את ה-CPU "להודיע" לבקר ההתקנים שהתו הנוכחי נקרא ע"י ה-CPU.
ה-port הזה הוא דו-סיטרי: בקר ההתקנים מעביר קוד בין 0 ל-127 (למעשה מספר סידורי) ל-CPU, ה-CPU מעביר אותו בחזרה לבקר ההתקנים תוך הדלקת הביט המשמעותי ביותר, ואחר כך שוב את אותו הקוד עם הביט המשמעותי ביותר כבוי.

אחרי ההודעה מהמקלדת, בקר ההתקנים מבקש מבקר הפסיקות לגרום לפסיקה מספר 9 ב-CPU. ה-CPU מחובר לבקר הפסיקות (בין השאר) דרך port 20h. כחלק מהטיפול בפסיקה 9 (או כל פסיקת חומרה אחרת) על ה-CPU לאותת "סיום טיפול בפסיקה" ע"י כתיבת הערך 20h לתוך port 20h. השיוון בין מספר ה-port והתוכן הוא, עד כמה שידוע לי, מקרי.

לפיכך התהליך של טיפול בפסיקה 9 ע"י ה-ISR, כאשר הסדר מחייב, הוא כלהלן:

א. קרא את התו דרך port 60h

ב. הודע לבקר ההתקנים שהתו נקרא, דרך port 61h.

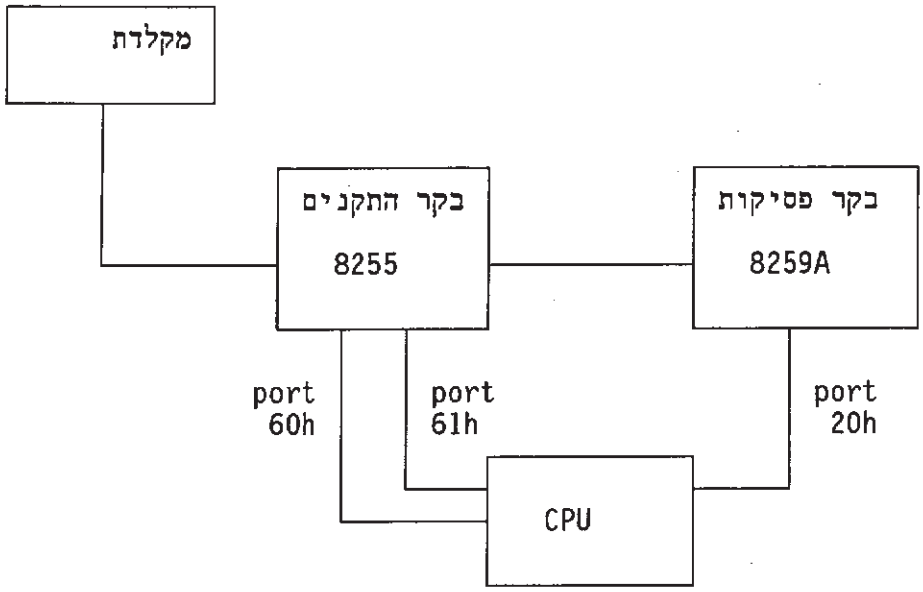
הדרך לעשות זאת הינו <u>לקרוא</u> את תוכן port 61h, <u>להדליק</u> את הביט העליון של המספר 8-ביט הזה, <u>לשגר</u> אותו חזרה ל- port 61h, <u>לכבות</u> את הביט העליון, <u>ולשגר</u> שוב את המספר עם הביט העליון כבוי. ראה קטע קוד III בתוכנית.

ג. הודע לבקר הפסיקות שהפסיקה טופלה, דרך port 20h.

הדרך לעשות זאת הינו לשגר את המספר 20h לתוך port 20h. ראה קטע קוד V בתוכנית.

ב-8086 שימש ה-Intel 8259A בתור בקר הפסיקות, וה-Intel 8255 בתור בקר ההתקנים. במחשבים שבאו יותר מאוחר ה-chips הללו הוחלפו ב-chips אחרים. בכלל, עם הזמן היה איחוד של Support Chips מסוימים, החלפתם באחרים ואפילו שילובם בתוך ה-CPU. אבל הפונקציונליות של ה-ports נשמר בדרך כלל, משיקולי "תאימות לאחור". לכן המודל שהיה ל-8086 רלוונטי במידה מסוימת גם היום.

לפיכך ב-8086 היו ה-CPU, שני ה-chips הנוספים, ה-8259A וה-8255 והמקלדת מחוברים בצורה הבאה:



שימו לב שרק בחיבורים ל-CPU יש מספרי port.

אלו כמובן לא כל החיבורים שיש: רק אילו שמעורבים בפסיקה 9. לדוגמא, ישנו port 21h בין ה-CPU ל-8259A המאפשר ל-CPU להשפיע על תפקוד ה-8259A - למשל להשבית פסיקות ספציפיות כמו פסיקה 8 או פסיקה 9.

תוכנית דוגמא keyb6.asm

התוכנית הזו קוראת מקשים מהמקלדת ומדפיסה אותם, אבל ללא שום סיוע לא ממערכת ההפעלה ולא מה-BIOS, אלא קוראת את הנתונים ישירות מהחומרה. במושגים מקצועיים התוכנית הזו מכילה device driver משלה למקלדת. בתוכנית מופיעים מספר בלוקים שנחשבים לעיקר התוכנית.

התוכנית מקבלת מהחומרה את ה-scan code של המקשים שנלחצים ומתמירה את הנתונים ל-Ascii code בעצמה. היא עושה את ההמרה תמיד לאותיות גדולות (Upper Case). היא מתעלמת מהמקשים שאין להם קוד Ascii וגם למקשי ה-Shift וה-Caps Lock לא יהיה שום ההשפעה עליה.

ההמרה נעשית באמצעות הטבלה ScanTable שבו עבור המקשים שיש להם קודי Ascii התו ה-i-יי בטבלה הוא קוד ה-Ascii של המקש שה-scan code שלו הוא i. שים לב שלטבלה יש את מבנה העקרוני של הפריסה של מקשים על המקלדת. מקשים שאין להם קודי Ascii הערך בטבלה הוא אפס.

ההמרה עצמה נעשית על ידי פקודת המכונה XTALB שנקראית Traslate byte פקודה ללא אופרנדים שנועדה להמרות מסוג זה. הפקודה מחשבת את הכתובת BX+AL ואת הבית שהיא מוצאת שם היא מציבה חזרה לתוך AL. אם תרצו משהו כמו

AL = [BX+AL] תוכן

במילים אחרות אם BX מצביע לטבלת המרה מהסוג הזה, אז XLATB תבצע מעין תרגום של AL דרך הטבלה. בתוכנית עצמה בבלוק IV מופיע קטע קוד

```
MOV BX,OFFSET SscanTable
XLATB
```

התוכנית הראשית מבצעת השתלטות על פסיקה מספר 9 ולאחר מכן נכנסת ללולאה שבה היא ממתינה למידע על מקשים ומדפסה אותם עד שנודע לה שנלחץ המקש Esc. בבלוק I היא משתלטת על פסיקה 9 ובבלוק II היא משחזרת לפני חזרה ל-DOS. החזרה ל-DOS נעשה ע"י הדרך הישנה - הסתעפות ל-INT 20h בראש ה-PSP כפי שמתואר בתקציר מספר 10. הדפסה של התו נעשה ע"י קריאה לפרוצדורה DispChar הקוראת מצידה לפסיקת BIOS INT 10h אופציה AH = 14. התוכנית הראשית מקבלת ממטפל הפסיקה את תוכן המקשים באמצעות המשתנה גלובלי Buffer ומסמן שיש אינפורמציה חדשה באמצעות המשתנה הגלובלי Buff_Flag. כל זה נעשה בתוך הרוטינה Kbget הנקראית ע"י

התוכנית הראשית. אינני חושב שיש צורך מיוחד להתעקב על כל אלה משום שהם אינם מה שאנחנו רוצים להמחיש כאן.

מה שחשוב יותר הוא רוטינת הטיפול בפסיקה 9 (ה-device driver עצמו) הנקרא Kbint. בבלוק III הוא קורא את המקש ומאותת לבקר ההתקנים 8255 שהמקש נקרא.

בפקודה

```
IN AL,60h
```

הרוטינה קוראת מה-8255 את תוכן לחיצת המקש, כולל הביט המשמעותי ביותר המבדיל בין לחיצה (0) לשחרור (1) ומשמרת אותו מיד לאחר כך ע"י פקודת PUSH.

בפקודה

```
IN AL,61h
```

התוכנית קוראת מה-8255 את תוכן מספר port 61h ובפקודות

```
OR AL,80h
OUT 61h,AL
AND AL,7Fh
OUT 61h,AL
```

היא מודיעה ל-8255 על קריאת המקש ע"י כותבת את המספר שקראה מה-port חזרה לתוכו פעם עם הביט המשמעותי דלוק ולאחר מכן כאשר הוא כבוי.

בבלוק IV הרוטינה בודקת אם מדובר בלחיצה או שחרור ע"י הפקודה

```
TEST AL,80h
```

פקודת המכונה TEST מבצע AND לוגי על שני האופרנדים מבלי לשנות אותם - TEST היא גירסה של AND המשפיעה רק על אוגר הדגלים. ביצוע TEST של בית עם הקבוע 80h פירשו ZF = 1 אם הביט המשמעותי ביותר של הבית הוא אפס ZF = 0 אם הביט המשמעותי ביותר הוא 1. במקרה ש-ZF = 1 מדובר בשחרור מקש והתוכנית מתעלמת ממנו. אחרת היא מתמירה את הקוד ל-Ascii דרך הטבלה. אם מדובר במקש שאין לו קוד Ascii (תוצאת ההמרה היא אפס)

זו תהיה עוד מקרה שהתוכנית מתעלמת מהמקש. אחרת היא מציבה את תוצאת ההמרה במשתנה הגלובלי Buffer ומדליקה את משתנה הדגל הגלובלי Buff_Flag.

בבלוק V מתבצע ההודעה לבקר הפסיקות 8259A שהפסיקה טופלה. הדבר נעשה ע"י צמד הפקודות

```
MOV AL,20h
OUT 20h,AL
```

לאחר מכן הרוטינה משחזרת אוגרים ומבצעת IRET.

```
;
; keyb6.asm
; Example of custom keyboard support software
;
Stack1   SEGMENT PARA STACK 'STACK'
         DB 256 DUP(?)                ; 256d bytes of stack space
Stack1   ENDS
;
Data     SEGMENT  PARA PUBLIC 'Data'
Buffer   DB  ?                ; 1 byte keyboard buffer
Buff_Flag DB ?                 ; 0 - Buffer empty, 1 - Char in
                              ; Buffer
Msg3 DB ' PRESS ANY KEY AND YOU WILL SEE IT ON THE'
     DB ' SCREEN (PRESS ESC TO QUIT)',10,13,'$'
Oldbios_Off DW 0
Oldbios_Seg DW 0

; ScanTable converts scan codes received from the keyboard
; into their corresponding ascii character codes:
;
ScanTable DB 0,1,'1234567890-=',8,0
          DB 'QWERTYUIOP[]',0Dh,0
          DB 'ASDFGHJKL;',0,0,0,0
          DB 'ZXCVBNM,./',0,0,0
          DB ' ',0,0,0,0,0,0,0,0,0,0,0,0
          DB '789-456+1230.'
Data      ENDS
;
Code      SEGMENT PARA PUBLIC 'Code'
Start     PROC FAR
;
;  Standard program prologue
;
    ASSUME CS:Code
      PUSH DS       ; Save PSP  SED addr
      MOV  AX,0
      PUSH AX       ; Save RET addr offset (PSP+0)
      MOV  AX,Data
      MOV  DS,AX    ; Establish Data SEG addressability
    ASSUME DS:Data
;
;  Part1: Setup our own keyboard interrupt service routine
;
      MOV AH,09
      MOV DX,OFFSET Msg3
      INT 21h
;
```

```
;=============================================================================
;                                          ###                               ||
;                                           #                                ||
;                                           #                                ||
        CLI             ;                   ###                              ||
        MOV AX,0        ;                                                    ||
        MOV ES,AX       ; Point extra segment at the ...                     ||
;                         ... interrupt service routine address table        ||
        MOV DI,36     ; Offset of entry for type code 09h                    ||
        MOV AX,ES:[DI]          ;save old bios offset                        ||
        MOV Oldbios_Off,AX      ;in Oldbios_Off                              ||
        MOV WORD PTR ES:[DI],OFFSET Kbint  ;                                 ||
                                    ; Offset of our service rout             ||
        MOV AX,ES:[DI+2]         ;save old bios segment                      ||
        MOV Oldbios_Seg,AX      ;in Oldbios_Seg                              ||
        MOV  ES:[DI+2],CS       ; SEG of our service routine                 ||
        STI                     ; Enable interrupts to the CPU               ||
;                                                                            ||
;=============================================================================
;  Part2: Read from keyboard and display chars on screen
;
Forever:
        CALL Kbget              ; Wait for a character from the keyboard
        CMP AL,01               ; check for "esc"=quit =01
        JZ Quit
        PUSH AX                 ; Save the character
        CALL DispChar           ; Display the character received
        POP  AX                 ; Restore the character
        CMP  AL,0Dh             ; Was it carrige return?
        JNZ  Forever            ; Repeat loop if not
        MOV  AL,0Ah             ; Yes it was, we must also display ...
        CALL DispChar           ;  ... aline feed.
        JMP  Forever            ; Stay in this loop forever.
;
;=============================================================================
;                                        ### ###                             ||
;                                         #   #                              ||
;                                         #   #                              ||
;                                        ### ###                             ||
Quit:                           ;                                            ||
        CLI                     ;disable interrupt                           ||
        MOV AX,0                ;restore the old bios keyboard routine       ||
        MOV ES,AX               ;                                            ||
        MOV DI,36               ;                                            ||
        MOV AX,Oldbios_Off      ;                                            ||
        MOV ES:[DI],AX          ;                                            ||
        MOV AX,Oldbios_Seg      ;                                            ||
        MOV ES:[DI+2],AX        ;                                            ||
        STI                     ;                                            ||
;                                                                            ||
;  Return to DOS                                                             ||
;                                                                            ||
        RET                     ;                                            ||
;                                                                            ||
;=============================================================================
```

```
;
; Call Kbget to wait for a character to be received from
;   the keyboard.  The character is returned in reg AL.
;
Kbget   PROC    NEAR
        CLI                 ; Disable interrupts
        CMP   Buff_Flag,0   ; Is buffer empty?
        JNZ   Kbget2        ; ->No
        STI                 ; Re-enable interrupts
        JMP   Kbget         ; Wait until something in buffer
;  There is something in the buffer, get it:
Kbget2:
        MOV AL,Buffer       ; Get char at buffer start
        MOV Buff_Flag,0     ; Signal Buffer empty
        STI                 ; Re-enable interrupts
        RET                 ; Return from Kbget
Kbget   ENDP
;
;  Kbint is our own interrupt seviice routine
;
Kbint   PROC  FAR
        PUSH  DS  ; save all altered registers
        PUSH  BX
        PUSH  AX
;
;  Establish addresability of our data segment:
;
        MOV AX,Data
        MOV DS,AX
;
;=============================================================================
;                              ###  ###  ###                                 ||
;                               #    #    #                                  ||
;                               #    #    #                                  ||
;                              ###  ###  ###                                 ||
;                                                                            ||
; Read the keyboard data and send acknowlege signal                          ||
;                                                                            ||
        IN    AL,60h    ; Read keyboard input                                ||
        PUSH  AX        ; Save keyboard input                                ||
        IN    AL,61h    ; Read 8255 port pb                                  ||
        OR    AL,80h    ; Set keyboard acknowlege signal                     ||
        OUT   61h,AL    ; Send keyboard acknowlege signal                    ||
        AND   AL,7Fh    ; Reset keyboard acknowlege signal                   ||
        OUT   61h,AL    ; Restore original 8255 port pb                      ||
;                                                                           ||
;============================================================================
;
```

```
;=================================================================
;                                ###  #     #                     ||
;                                 #    #   #                      ||
;                                 #     # #                       ||
;                                ###     #                        ||
;                                                                 ||
; Decode the scan code received:                                  ||
;                                                                 ||
        POP   AX                      ; Regain the keyboard input (AL)  ||
        TEST  AL,80h                  ; Is it a key being released      ||
        JNZ   Kbint2                  ; Branch if yes, we ignore these  ||
        MOV   BX,OFFSET ScanTable     ; Scan code - ASCII table         ||
        XLATB                         ; Convert the scan code to        ||
                                      ;  an ASCII char                  ||
        CMP   AL,0                    ; Is it a valid ASCII key         ||
        JZ    Kbint2                  ; Branch if not                   ||
;                                                                 ||
; Place the ASCII character into the buffer:                      ||
;                                                                 ||
        MOV   Buffer,AL               ; Place char in buffer            ||
        MOV   Buff_Flag,1             ; Signal char in buffer           ||
;                                                                 ||
;=================================================================
;
;
;=================================================================
;                                #     #                          ||
;                                 #   #                           ||
;                                  # #                            ||
;                                   #                             ||
;                                                                 ||
; Now indicate "END OF INTERRUPT" to the interrupt controller:    ||
;                                                                 ||
Kbint2:                               ;                           ||
        MOV   AL,20h                  ; Send "EOI" command ...          ||
        OUT   20h,AL                  ; ... to  8259 command register  ||
;                                                                 ||
;=================================================================
;
        POP   AX                      ; Restore all altered registers
        POP   BX
        POP   DS
        IRET                          ; Return from interrupt
Kbint   ENDP
;
; Subroutine to display a character on the screen.
; Enter with AL = character to be displayed
; Uses video interface in BIOS
;
DispChar  PROC  NEAR
          PUSH  BX      ; Save BX register
          MOV   BX,0    ; Select display page 0
          MOV   AH,14   ; Function code for write
          INT   10h     ; Call video driver in BIOS
          POP   BX      ; Restore BX register
          RET           ; Return to caller of 'DispChar'
DispChar  ENDP
;
Start     ENDP
Code      ENDS
     END  Start
```

```
E:\>keyb6.exe
 PRESS ANY KEY AND YOU WILL SEE IT ON THE SCREEN (PRESS ESC TO QUIT)
USER ECHO
IS ALWAYS UPPER CASE

E:\>
```

תוכנית דוגמא b800h.asm

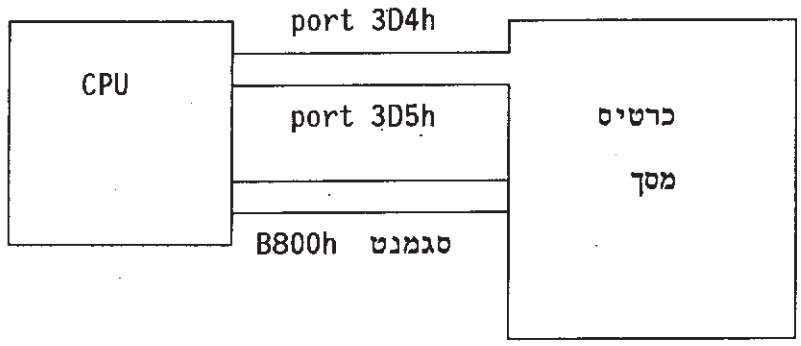
התוכנית הזו נועדה להמחיש את נושא ה-Memory Mapped I/O ופלט למסך כולל ה-Hardware Scrolling.

תוכנית זו משתמשת במסך הגרפי במוד טקסט צבעוני 25x40.

בתנאים אלו כתיבה לתצוגה היא דרך סגמנט B800h (זכרון פיזי 736k).

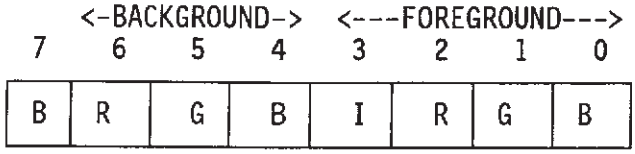
התוכנית כותבת 'A' -ים, 'B' -ים, 'C' -ים, 'D' -ים בצבעים שונים למסך דרך סגמנט B800h, ומדפדפת ביניהם ע"י כתיבה ל-ports 3D4h, 3D5h מה שקרוי דיפדוף חומרה (Hardware Scrolling).

לצורך נקודת ההשקפה שלנו המודל של החיבורים נראה כך:



פורמטים של אינפורמציה

במוד טקסט המערכת מפרשת את תוכן סגמנט B800h כצמדים של בתים המתארים כל אחד משבצת אחת במסך. הבית הנמוך (היסטים זוגיים) מפורש כקוד ה-ASCII של הבית שאותו צריך להציג. הבית בכתובת הגבוהה (היסטים איזוגיים) מפורש כבית של מאפיינים (Attribute byte):



B - Blink (1 - Foreground character blinks)
R - Red
G - Green
B - Blue
I - Intensity (0 - Low, 1 - High)

כמו כן ה-CPU מחובר למסך דרך זוג ה-ports 3D5h - 3D4h, שדרכם קובעים (בין השאר) את דף התצוגה, ומיקום ה-cursor. אנו קובעים את מוד המסך ע"י פסיקת תוכנה BIOS 10h.

בתוכנית הזו כרטיס המסך מקבלת את ההנחיות הבאות דרך פורטים 3D4h - 3D5h:

לפי התוכן הנכתב לפורט 3D4h הכרטיס מפרש / מחבר את פורט 3D5h לאוגר הפנימי שלו המתאים.

הערכים שאנחנו נשתמש (יש יותר):

תוכן 10 בפורט 3D4h: פורט 3D5h מפורש כגבול עליון לתצוגת ה-cursor (קו 0 עד 14).

תוכן 11 בפורט 3D4h: פורט 3D5h מפורש כגבול תחתון לתצוגת ה-cursor (קו 1 עד 15).

תוכן 12 בפורט 3D4h: פורט 3D5h מפורש כבית עליון לנקודת ההתחלה של תצוגת המסך (היסט בבתים מתחילת סגמנט B800h).

תוכן 13 בפורט 3D4h: פורט 3D5h מפורש כבית תחתון לנקודת ההתחלה של תצוגת המסך (היסט בבתים מתחילת סגמנט B800h).

תוכן 14 בפורט 3D4h: פורט 3D5h מפורש כבית עליון למיקום ה-cursor (היסט בבתים מתחילת סגמנט B800h).

תוכן 15 בפורט 3D4h: פורט 3D5h מפורש כבית תחתון למיקום ה-cursor (היסט בבתים מתחילת סגמנט B800h).

עם הרצת התוכנית הזו המסך יציג סידרה 50 שורות של 'A' -ים בצבע תכלת שבהדרגה יוחלפו ב-50 שורות של 'B' סגולים שיוחלפו בהדרגה ב-'C'-ים חומים שייוחלפו ב-50 שורות של 'D'-ים לבנים. ה-cursor נמצא בתחילת אחת השורות תחת אחד ה-'B' -ים. עם ההגעה לסוף ה-'D'-ים יתחיל תהליך הפוך של דפדוף חזרה עד לתחילת ה-'A' -ים וחוזר חלילה. לחיצת מקש כלשהוא (למעט Esc) יגרום לעצירת הדפדוף ולחיצה נוספת של מקש כלשהוא יחדש אותו. לחיצת Esc יגרום לסיום התוכנית.

בפועל התוכנית כותבת לזכרון המסך יותר משהמסך יכול להציג. ע"י שינוי נקודת ההתחלה של התצוגה בכל פעם מוצגת חלק אחר של זכרון כרטיס המסך. הפעולה הזו, של שינוי התצוגה לא ע"י כתיבת ערכים חדשים אלא ע"י שינוי נקודת ההתחלה נקרא Hardware Scrolling.

מהלך התוכנית היא

- להעביר את המסך למוד טקסט 25x40 - 25 שורות של 40 תוים לשורה (בניגוד ל-80 תוים לשורה בדרך כלל) ע"י רוטינת ה-BIOS INT 10h אופציה AH = 0, AL = 1. לא ניכנס לפרטים בנושא INT 10h הזה כאן. נאמר רק שבמוד הזה מסך מציג את התוכן של סגמנט B800h.

- מילוי התוכן הרצוי לזכרון המסך. הדבר נעשה בארבעה שלבים ע"י הצבת הערך הרצוי ל-AX וכתיבת הערך 2000 פעם. למשל כתיבת ה-'A'-ים נעשה ע"י

```
MOV AL,'A'
MOH AH,03h
MOV CX,2000
Loop1:
    MOV ES:[DI],AX
    ADD DI,2
    LOOP Loop1
```

לכאורה הפקודה MOV ES:[DI],AX היא פקודת כתיבה לזכרון אבל כתיבה לסגמנט B800h משמעותו פעולת פלט דומה יותר ל-OUT מאשר כתיבה לזכרון. המשמעות כאן היא כתיבה לזכרון המסך של 50 שורות של 'A'-ים בצבע תכלת.

- קביעת מיקום ה-cursor. הדבר נעשה ע"י הפקודות

```
MOV BX,40*63
MOV DX,3D4h
MOV AL,14
MOV AH,BH
OUT DX,AX
;
MOV AL,15
MOV AH,BL
OUT DX,AX
```

ה-OUT הראשון יהיה הבית המשמעותי יותר וה-OUT השני הבית המשמעותי פחות. הפקודות הללו ממקמים את ה-cursor בעמוד הראשון בשורה ה-64 של זכרון המסך.

מהשלב הזה התוכנית תכנס ללולאה של

- המתנה
- בדיקת לחיצת מקש
- שינוי נקודת ההתחלה של התצוגה של המסך

שינוי נקודת ההתחלה של תצוגת המסך נעשה בשני שלבים:

הבית המשמעותי יותר נכתב ע"י הפקודות

```
MOV DX,3D4h
MOV AL,12
MOV AH,BH
OUT DX,AX
```

הבית המשמעותי פחות נכתב ע"י הפקודות

```
MOV AL,13
MOV AH,BL
OUT DX,AX
```

```
;
; B800h.ASM
; This program demonstrates the use of hardware scrolling.
;
Stack               SEGMENT PARA STACK 'STACK'
                    DB      256 DUP(0)
Stack               ENDS
;
Data                SEGMENT PARA PUBLIC 'Data'
Count               DB      0       ; Number of lines scrollled down
Base                DW      0
;
Direction           DB      0       ; The scroll direction
;                                     0=Down    1=Up
Data                ENDS
Code                SEGMENT PARA PUBLIC 'Code'
Start               PROC    FAR
;
;Standard program prologue
;
  ASSUME            CS:Code
  PUSH              DS              ; Save PSP segment address
  MOV               AX,0
  PUSH              AX              ; Save RET address offset (PSP+0)
  MOV               AX,Data
  MOV               DS,AX           ; Establish Data segment addressability
  ASSUME            DS:Data
;
;Part1 : Initialize the display adapter
;
  MOV               AH,0            ; Select function = 'SET MODE'
  MOV               AL,1            ; 40 BY 25 Color image
  INT               10H             ; Adapter initilizedD. Page 0 displayed
;
  MOV               AX,0B800h       ; Segment address of memory of color adapter
                                    ; in text mode
  MOV               ES,AX           ; Set up extra segment register
  MOV               DI,0            ; Initial offset address into segment
  MOV               AL,'A'          ; Character A to fill adapter memory
  MOV               AH,03h          ; Attribute byte: BLUE + GREEN = LIGHT BLUE
;
  MOV               CX,2000
Loop1:
       MOV ES:[DI],AX
       ADD DI,2
       LOOP Loop1
;
  MOV               AL,'B'          ; Character B to fill adapter memoory
  MOV               AH,05h          ; Attribute byte: BLUE + RED = PURPLE
;
  MOV               CX,2000
Loop2:
       MOV ES:[DI],AX
       ADD DI,2
       LOOP Loop2
```

```
;
   MOV              AL,'C'          ; Character C to fill adapter memory
   MOV              AH,06h          ; Attribute byte: GREEN + RED = BROWN
;
   MOV              CX,2000
Loop3:
        MOV ES:[DI],AX
        ADD DI,2
        LOOP Loop3
;
   MOV              AL,'D'          ; Character D to fill adapter memory
   MOV              AH,07h          ; Attribute byte: GREEN + RED + BLUE = WHITE
;
   MOV              CX,2000
Loop4:
        MOV ES:[DI],AX
        ADD DI,2
        LOOP Loop4
;
;
; Set the cursor address registers
;
   MOV              BX,40*63        ; Cursor position: 40th row, Col 0
   MOV              DX,3D4h         ; 3D4h - 3D5h: Display adapter ports
   MOV              AL,14           ; 14 - Cursor address high byte register
   MOV              AH,BH           ; Load AH with desired high byte value
   OUT              DX,AX           ; Output AL to port 3D4h, AH to port 3D5h
;
   MOV              AL,15           ; 15 - Cursor address low byte register
   MOV              AH,BL           ; Load AH with desired low byte value
   OUT              DX,AX           ; Output AL to port 3D4h, AH to port 3D5h
;
;
;PART 2 : Scroll the display every second until a key is hit
;
Delay:
   ; Delay AL*CX operations
   MOV               AL,200
Tenth:
   MOV               CX,60000
Dloop:
   LOOP             Dloop
   DEC              AL
   CMP              AL,0
   JNE              Tenth
;
   MOV              AH,1            ; User pressed key?
   INT              16H
   JZ               Scroll          ; No, scroll
   MOV              AH,0            ; Yes, read the key
   INT              16H
   CMP              AH,1            ; Is it Esc?
   JE               ToReturn        ; Yes - return to DOS
   MOV              AH,0            ; No: Wait for another key
   INT              16H
   CMP              AH,1            ; Was it Esc?
   JE               ToReturn        ; Yes - return to DOS
```

```
;                               ; No - Continue
Scroll:
   MOV          BX,Base         ; Retrieve present location
   CMP          Direction,0     ; Backwards or Forwards?
   JNE          Backwards       ; Direction = 1: Backwards
;                                 Direction = 0: Forwards
   INC          Count           ; Advance 1 line
   CMP          Count,160       ; End of 4 pages?
   JB           Down_Ok         ; No
;
   MOV          Direction,1     ; Yes, reverse direction
Down_Ok:
   ADD          BX,40           ; No: Advance 1 line
   MOV          Base,BX         ; Store location
;
   JMP          SHORT  Update
;
Backwards:
   DEC          Count           ; Backtack 1 line
   CMP          Count,0         ; Reached first line?
   JNE          Up_Ok           ; No
;
   MOV          Direction,0     ; Yes, reverse direction
Up_Ok:
   SUB          BX,40           ; Backtack 1 line
   MOV          Base,BX         ; Store location
;
Update:
   MOV          DX,3D4h         ; 3D4h - 3D5h: Display adapter ports
   MOV          AL,12           ; Display position high byte register
   MOV          AH,BH           ; Move desired high byte value to AH
   OUT          DX,AX           ; Output AL to port 3D4h, AH to port 3D5h
   MOV          AL,13           ; Display position low byte register
   MOV          AH,BL           ; Move desired low byte value to AH
   OUT          DX,AX           ; Output AL to port 3D4h, AH to port 3D5h
;
   JMP          Delay             ; Repeat the process
;
ToReturn:
                MOV  AX,3     ; Restore adapter mode
                INT  10h
                RET           ; Return to DOS
Start           ENDP
Code            ENDS
   END          Start
```

תוכנית דוגמא cursor2.asm

התוכנית הזו נועדה להדגים אספקטים נוספים של ניהול המסך. מה שהתוכנית הזו עושה היא להציג את התו 'A' באמצע המסך כאשר מלבדו המסך ריק וה-cursor מהבהב מתחתיו. כל לחיצה עת מקש כלשהוא (למעט Esc) יגרום ל-cursor לשנות צורה, מקו תחתון ל-'A' עד ל-cursor המירבי של כיסוי כל המשבצת וחזרה. לחיצת Esc יביא לסיום התוכנית.

הגודל של ה-cursor נקבע לפי גבול עליון וגבול תחתון. הקו העליון ביותר ממוספר 0 והתחתון ביותר הוא 14. התוכנית יכולה לבחור גבול עליון מ-1 ל-14 וגבול תחתון כל מספר עד 15 מתחתיו. לפיכך ה-cursor המירבי הוא מ-0 ל-15 ומינימלי "הרגיל": הוא מ-14 ל-15.

הגבול העליון של ה-cursor נקבע לפי תוכן port 3D5h בתנאי שב-port 3D4h ישנו הערך 10 (Ah).

הגבול התחתון של ה-cursor נקבע לפי תוכן port 3D5h בתנאי שב-port 3D4h ישנו הערך 11 (Bh).

לפיכך קביעת ה-cursor המירבי נעשה ע"י הפקודות

```
MOV DX,3D4h
MOV AX,000Ah
OUT DX,AX
MOV AX,0F0Bh
OUT DX,AX
```

וקביעת ה-cursor המינימלי נעשה ע"י הפקודות

```
MOV DX,3D4h
MOV AX,0E0Ah
OUT DX,AX
MOV AX,0F0Bh
OUT DX,AX
```

מחיקת תוכן המסך נעשה ע"י הפקודות

```
MOV AX, 0B800h
MOV ES,AX
MOV DI,0
MOV AL,' '
MOV AH,0Eh
MOV CX,1000
CLD
REP STOSW
```

שמשמעותו הגדרת המסך כרווחים בצבע צהוב מודגש (למעשה הכל שחור) על פני 1000 התוים הראשונים (25 שורות ראשונות או מסך שלם) של המסך.
STOSW היא פקודת מחרוזת המציבה את התוכן של AX בכתובת ES:[DI] וקידום DI אוטומטית ב-2. משמעות שלושת הפקודות

```
MOV CX,1000
CLD
REP STOSW
```

משמעותו "כתוב את תוכן AX לתוך 1000 בתים החל מכתובת ES:[DI]".
זוהי אפשרות יותר יעילה מאשר מימוש רגיל שקול מבחינת התוצאה של

```
MOV CX,1000
Loop1:
   MOV ES:[DI],AX
   ADD DI,2
   LOOP Loop1
```

משמעות ה-REP STOSW היא בדרך כלל פעולת כתיבה לזכרון לכל דבר אבל בסגמנט המסוים מאד B800h פירושו Memory Mapped I/O לכרטיס המסך.

שינוי המשבצת האמצעית במסך ל-'A' צהוב על רקע שחור נעשה ע"י הפקודה

```
MOV BYTE PTR ES:[2*(12*40+20)],'A'
```

משמעותו תוכן התו מספר 20 (ה-21 מהקצה השמאלי) בשורה מספר 12 (ה-13 מהקצה העליון) של המסך יהיה 'A'. הצבע הצהוב נקבע קודם לכן ע"י ה-REP MOVSW.

מיקום ה-cursor נעשה בשני שלבים כמו בתוכנית b800h.asm ע"י הפקודות

```
MOV BX,12*40+20
MOV DX,3D4H
MOV AL,14
MOV AH,BH
OUT DX,AX
MOV AL,15
MOV AH,BL
OUT DX,AX
```

כאשר DX = 3D4h ו- AL = 14 ו-AH פירושו קביעת בית עליון של מיקום ה-cursor, כאשר DX = 3D4h ו- AL = 15 ו-AH פירושו קביעת בית תחתון של מיקום ה-cursor.

```
;
; Cursor2.asm
;
;This program demonstrates cursor manipulation.
;
;
Stak                SEGMENT PARA STACK 'STACK'
DB                   256 DUP(0)
Stak                ENDS
;
Data                 SEGMENT PARA PUBLIC 'Data'
CursorPos           DW      0      ; Number of lines scrolled down
Base                DW      0
;
Data                ENDS
Code                SEGMENT PARA PUBLIC 'Code'
                    .386   ; Enable 386 commands
;
Start               PROC    FAR
;
;STANDARD PROGRAM PROLOGUE
;
  ASSUME            CS:Code
  PUSH              DS            ; Save PSP segment address
  MOV               AX,0
  PUSH              AX            ; Save INT 20h address offset (PSP+0)
  MOV               AX,Data
  MOV               DS,AX         ; Establish Data segment addressability
  ASSUME            DS:Data
;
;Part1 : Initialize the display adapter
;
  MOV               AH,0            ; Select function = 'Set mode'
  MOV               AL,1            ; 40 by 25 color image
  INT               10h             ; Adapter initialized. Page 0 displayed
;
  MOV               AX,0B800h       ; Segment address of memory on color adapter
;
  MOV               ES,AX           ; Set up extra segment register
  MOV               DI,0            ; Initial offset address into segment
  MOV               AL,' '          ; Character space to fill adapter memory
  MOV               AH,0Eh          ; Attribute byte : Intense yellow
  MOV               CX,1000         ; Initialize count, 1 Screen
  CLD                               ; Write forward
  REP               STOSW           ; Write
;
; Write 'A' in mid screen
;
  MOV               BYTE PTR ES:[2*(12*40+20)],'A'
;
; Set the cursor address registers
;
  MOV               BX,12*40+20 ; Set BX to Desired Cursor position
  MOV               DX,3D4H  ; Point to 3D4H - 3D5H port pair
  MOV               AL,14    ; Address of cursor register pos high byte
  MOV               AH,BH    ; Get desired value of cursor pos high byte
  OUT               DX,AX    ; Port(3D4h) = 14, Port(3D5h) = Value of BH
;
  MOV               AL,15    ; Address of cursor register pos low byte
  MOV               AH,BL    ; Get desired value pf cursor pos low byte
  OUT               DX,AX    ; Port(3D4h) = 15, Port(3D5h) = Value of BL
```

```
;
;PART 2 : Wait for key strike
;
; Wait for key
;
NextLoop:
  MOV              AH,0        ; Wait and read key
  INT              16h         ;
  CMP              AH,1        ; Is it Esc?
  JE               ToReturn    ; Yes - Return to DOS
;
;  Not esc key - change cursor
;
  ; 3D4H  Graphics adapter address register port
  ; 3D5H  Graphics adapter data register port
;
    MOV             DX,3D4h   ; Point TO 3D4h - 3D5h port pair
    MOV             AX,010Ah  ; Cursor start address (0Ah) - Value 1 (01h)
    OUT             DX,AX     ; Port(3D4h) = 0Ah, Port(3D5h) = 01h
    MOV             AX,0E0Bh  ; Cursor end address - Value 14 (0Eh)
    OUT             DX,AX     ; Port(3D4h) = 0Bh, Port(3D5h) = 0Eh
;
; Wait for key
;
  MOV              AH,0
  INT              16h
  CMP              AH,1        ; Is it Esc?
  JE               ToReturn    ; Yes - Return to DOS
;
    MOV             DX,3D4h    ; Point to 3D4H - 3D5H port pair
    MOV             AX,0D0Ah   ; Cursor start address (0Ah) - Value 13 (0Dh)
    OUT             DX,AX      ; Port(3D4h) = 0Ah, Port(3D5h) = 01h
    MOV             AX,0E0Bh   ; Cursor end address - Value 14 (0Eh)
    OUT             DX,AX      ; Port(3D4h) = 0Bh, Port(3D5h) = 0Eh
;
  JMP              NextLoop    ; Repeat main loop
;
ToReturn:
                   MOV  AX,2
                   INT  10h
                   RET
Start              ENDP
Code               ENDS
  END              Start
```