

## מנגנון הפסיקות (Interrupts)

פסיקה היא מצב של העברת שליטה בכפיה של ה-CPU לרוטינה. דרך אחרת לחשוב על פסיקה היא הפסקה בכוח של ריצה שוטפת של תוכנית והסתעפות לרוטינה אחרת.

פסיקה היא למעשה (או יכולה להיות) תגובה לאיתות של החומרה שמחוץ ל-CPU.

מדובר למעשה בעצירת התוכנית הנוכחית, ושפעול רוטינה אחרת על פניה. הרוטינה הזו תהיה בדרך כלל (אבל לא תמיד) רוטינה בלתי תלויה בתוכנית שמהלכה השוטף נעצרה. הרוטינה הזו "בלתי תלויה" בתוכנית המופסקת כמובן הזה, שכותב התוכנית (בדרך כלל) אינו מכיר את הרוטינה החלופית, אינו מעוניין בעצירת התוכנית שלו ויכול להיות שהרוטינה החלופית אינה משרתת את מטרתיו כלל. בדרך כלל התוכניתן אפילו אינו מביא את האפשרות של פסיקה בחשבון. המהלך של פסיקה עשוי להתרחש, בדרך כלל, אחרי כל פקודת מכונה בתוכנית, ללא תאום כלשהוא איתה וללא אפשרות לחזות את נקודת הזמן או הפקודה מראש. כמובן שיוצא מזה שבדרך כלל, ה"רוטינות החלופיות" הללו, שבהמשך נקרא להם ISR-ים, נכתבות בצורה כזו שההפעלה שלהם לא תשפיע לרעה על התוכניות המופסקות, או שהשפעה שלילית כזו תהיה מינימלית. ה-ISR בדרך כלל ישתדל להחזיר את השליטה לתוכנית המופסקת במהירות המירבית, והתוכנית המופסקת תתחדש כמצב זהה לרגע שבה היא הופסקה.

אולי הדרך הטובה ביותר להבין את מושג הפסיקה הוא לראות איך הוא מתבטא ב-CPU:

כאשר תוכנית רגילה מתבצעת, הפקודה הבאה לביצוע הנקבעת על ידי הערך של הזוג CS:IP, מתעדכנת באופן שוטף לפי רצון התוכנית:

עבור פקודות בקרה, אותם פקודות שתפקידם לשנות את ה-IP או ה-CS:IP (כמו JMP, CALL), הערך החדש נקבע על פי ערכים שהפקודה מכילה.

עבור יתר הפקודות כמו ADD, MOV (ואילה הרוב) הפקודה העוקבת בזכרון מיד אחרי הפקודה המתבצעת הנוכחית.

בשני המקרים הפקודה הבאה לביצוע נמצאת בתוך התוכנית המתבצעת.

פסיקה גורמת (או עשויה לגרום) שינוי הזוג CS:IP לערכים שהם מחוץ לתוכנית המופסקת.

## סוגי פסיקות

מבחינה לוגית, פסיקות נבדלות לכמה סוגים:

1. Hardware Interrupts - פסיקות חומרה - תגובה לאיתות רכיב חומרה במחשב שמחוץ ל-CPU. פסיקות מסוג זה הם בדרך כלל לצורכי קלט / פלט - למשל כתוצאה של לחיצת מקש במקלדת או הזזת העכבר.

2. Exceptions - חריגות - פסיקות הנובעת ממצב לא תקין כמהלך הריצה של התוכנית הנוכחית. חלוקה באפס (divide overflow) היא דוגמה קלאסית לכך. דוגמאות אחרות הם נסיון לבצע פקודת מכונה לא חוקית, נסיון לגשת לכתובת לא קיימת בזכרון, גישה לכונן דיסקטים שאין בו דיסקט וכו'.

3. Software Interrupts - פסיקות תוכנה - פסיקות ביוזמת התוכנית - בדרך כלל ע"י הפקודה INT. פסיקות מסוג זה הם בדרך כלל שימוש ברוטינות שירות הקיימות בזכרון (אך מחוץ לתוכנית).

זאת כבר ראינו: כאשר השתמשנו בספרית הרוטינות של DOS (INT 21h).

למעשה התאור שבראשית סיכום זה מתאים רק לפסיקות מסוג 1 ו-2 (פסיקות חומרה וחריגות). פסיקות מסוג 1 ו-2 שונים מאוד מפסיקות מסוג 3 (פסיקות תוכנה). פסיקות תוכנה הם כמעט כמו הסתעפות לרוטינה: המתכנת מכיר את הרוטינות הנקראות ופונה אליהם ביוזמתו.

### מה מבדיל בין פסיקת תוכנה להסתעפות לרוטינה (call)?

יש שני הבדלים עיקריים:

1. הסתעפות לרוטינה היא לקוד הנמצא בתוך קוד ה-EXE של התוכנית הרצה.

2. פסיקת תוכנה נעשית דרך מבנה נתונים גלובלי של המחשב.

1. לגבי במערכות מוגנות (UNIX, WINDOWS NT) לא ניתן בעזרת CALL לעשות מה שעושים בפסיקת תוכנה: הסתעפות לקוד הנמצא מחוץ לקובץ ה-EXE. אולי יותר נכון לומר שדואגים לכך שהדבר יהיה בלתי אפשרי. כאשר תוכנית מורצת, היא

מועתקת לזכרון ומקבלת שליטה. הסתעפויות CALL מוגבלות לנקודות זכרון הנמצאים ב"תמונה" הזו של קובץ ה-EXE בזכרון. כאשר תוכנית ב-C למשל קוראת לרוטינות סטנדרטיות כמו strcpy, sin, qsort (כל רוטינה שאינה קלט / פלט ואינה משפיעה / מסתיעת במערכת ההפעלה) מוכלל בתוך קובץ ה-EXE קוד בינארי (מתוך סיפריות סטנדרטיות) המישם את הפעולות הללו ונטען עמה לזכרון. זה כמובן נכון גם עבור קוד מקורי שנכתב בתוכנית עצמה. רוטינות קלט / פלט כמו printf, fopen הינם קודים שנמצאים בקובץ ה-EXE אבל מסתעים במערכת ההפעלה ע"י פסיקות תוכנה.

תחת DOS אפשר לומר ש-CALL הוא כמעט תמיד להסתעפות לקוד בתוך קובץ ה-EXE למרות שאין מניעה לעשות אחרת. זה עינין של נוהג יותר מאשר עינין טכני. תחת DOS ניתן להסתעף לרוטינות חיצוניות כאילו ע"י פקודת CALL (ע"י שימוש בפוינטר לפונקציה) אבל בשפות עילית שימוש כזה ב-CALL הוא נדיר מאד ונעשה רק במצבים מיוחדים.

לגבי 2., קריאה לפסיקת תוכנה היא דרך משאב (מבנה נתונים) גלובלי - טבלת הפסיקות IV (שיתואר להלן). המשאב הזה מוגבל (ל-256 פוינטרים) השימוש בו דורש מיומנות ואחריות ומה שאולי חשוב מכל - במערכות מוגנות (UNIX, NT) הוא גם מוגן - תוכניות רגילות לא יכולות לשנות אותו.

#### מנגנון מימוש פסיקות

#### טבלת הפסיקות Interrupt Vector או IV.

במחשב PC קיימים 256 (מ-0 עד 255) מספרי פסיקות שניתן להקצות למטרות כאלו ואחרות, לא כולם מנוצלות. המספר 256 הינו מאפין של המחשב. אין אפשרות לשנות זאת. קיים מערך של 256 פוינטרים מלאים (segment + offset) שהם הפוינטרים "לאן להסתעף" עבור 256 פסיקות אפשריות ממוספרות 0 .. 255. המערך טבלה הזו נקרא Interrupt Vector או IV. ב-8086 ובמצב real mode של ה-86x המתקדמים יותר, הפוינטרים האלו הם כולם 32bit (16 ל-offset, 16 ל-segment) והמערך נמצא ב-1K הכתובות הפיזיות הראשונות (0 .. 1023) של המחשב. עבור פסיקה מספר K, 4 הבתים בכתובות  $K*4$  עד  $K*4 + 3$  הינם פוינטר מלא - קודם offset (בתים בכתובות  $K*4, K*4+1$ ) ואחר כך segment (בתים בכתובות  $K*4+2, K*4+3$ ) - שהוא כתובת היעד "לאן להסתעף" בהתרחש פסיקה מספר K. היעד הזה הוא לרוטינה המיוחדת שתפקידה לבצע את המצופה מהפסיקה הזו, והרוטינה הזו נקראית רוטינת הטיפול בפסיקה Interrupt Service Routine או בקיצור ISR. לדוגמא, עבור פסיקה מספר 9 הפוינטר ל-ISR של הפסיקה נמצא בכתובות 36 - 39. ה-offset נמצא בכתובות 37 - 36, וה-segment בכתובות 38 - 39.

ב-Protected mode יש הבדלים לגבי ה-IV לעומת המצב ב-Real mode. לא ניכנס לזה כאן.

### מה גורם לפסיקה?

בפסיקות חומרה - הפסיקה נגרמת ע"י איתותים של רכיבי חומרה (כמו המקלדת למשל).

בפסיקות תוכנה - ע"י ביצוע של פקודת המכונה INT.

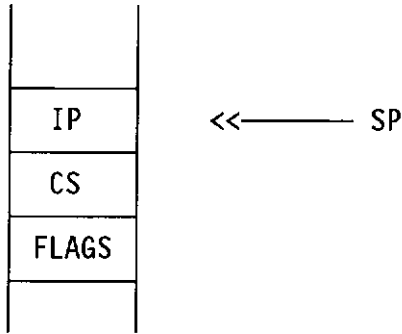
### מנגנון מימוש הפסיקות

המנגנון מימוש הפסיקות משותף לפסיקות תוכנה וחומרה.

כאשר מתרחשת פסיקה מספר K מתרחש התהליך הבא:

- א. ה-CPU גומר את פקודת המכונה שהוא מבצע כרגע,
  - ב. מיד לאחר מכן הוא שומר את אוגר הדגלים (FLAGS) במחסנית (פעולת PUSH).
  - ג. הוא מאפס את הדגל Interrupt Flag (IF) באוגר הדגלים.
- על כך נרחיב מאוחר יותר. כרגע נציין שהדבר מונע התרחשות פסיקות נוספות, תוך שהמחשב עסוק בטיפול בנוכחית.
- ד. הוא מאפס את הדגל Trap Flag (TF) באוגר הדגלים.
- זהו דגל שקשור בעיקר למימוש debuggers. על כך נרחיב מעט בהמשך.
- ה. במחסנית נשמרים (במעין PUSH) "כתובת הפקודה הבאה" ה-CS הנוכחי ולאחר מכן ה-IP.

לפיכך תוכן ראש המחסנית היא כעת



ו. מתוך הכתובות  $K*4$  עד  $K*4+3$  של ה-IV נשלפים הערכים החדשים של ה-IP  $(K*4, K*4+1)$  ו- $CS(K*4+2, K*4+3)$ .

### הסתיגות

לא תמיד הכתובת CS:IP הנשמרת במחסנית היא הכתובת של הפקודה הבאה לביצוע. עבור חריגות מסוימות (כמו ה-divide overflow, החלוקה באפס) הכתובת הנשמרת היא לא הפקודה הבאה לביצוע אלא דווקא הכתובת של הפקודה שהופסקה. השיקול כאן הוא כנראה להקל על איתור תקלות.

לסיכום, מתרחש כאן תהליך דומה במידה רבה לביצוע פקודה המכונה CALL: ההבדלים העיקריים הם מקור כתובת היעד (ה-IV), שמירת אוגר הדגלים ואיפוס ה-IF.

### ה-Interrupt Flag (IF)

בתוך אוגר הדגלים קיים דגל ה-Interrupt Flag או IF, השולט על מנגנון הפסיקות. כאשר  $IF = 0$ , מנגנון פסיקות החומרה מושבת. ה-CPU מתעלם מבקשות פסיקות חומרה. כאשר  $IF = 1$ , מנגנון פסיקות החומרה פעיל. ה-IF לא משפיע על פסיקות התוכנה (הפקודה INT). הדבר נחוץ משום שיש מצבים שבהם התוכנה לא יכולה להרשות לעצמה התרחשות פסיקה, מצבים שעוד נראה.

### פקודות מכונה שקשורות למנגנון הפסיקות

CLI - איפוס ה-IF ( $IF = 0$ ).

STI - הדלקת ה-IF (IF = 1).

INT k - k תמיד מספר קבוע - גורם לפסיקת תוכנה k.

IRET - חזור מטיפול בפסיקה - שליפה (POP) מהמחסנית את ה-IP, CS, FLAGS מהמחסנית.

שימו לב שעם ביצוע ה-IRET, ה-IF וה-TF ישתחזרו באופן אוטומטי לערכם המקורי לפני הפסיקה עם שיחזור הדגלים.

### Trap Flag (TF) -ה

ה-TF הוא דגל מיוחד באוגר הדגלים שכאשר הוא דלוק, ה-CPU מתפקד בצורה מיוחדת מאד: הוא מבצע פסיקה (למעשה חריגה) של פסיקה מספר 1 אחרי כל ביצוע של פקודת מכונה. פסיקה מספר 1 נקראית בשל כך פסיקת ה-single step. שים לב, שמכיון שה-TF מכובה אוטומטית ע"י מנגנון הפסיקות, התופעה הזו לא תתרחש ב-ISR של פסיקה 1 עצמה. מנגנון זה נועד בעיקר למימוש תוכנות debugger-ים תחת DOS. ה-debugger היה לוקח לעצמו את השליטה על פסיקה מספר 1 ואחרי ביצוע של כל פקודת מכונה של התוכנית הנבדקת ה-ISR שלו של פסיקה 1 היה בודק את מצב ה-CPU והתוכנית ולפי זה היה מחליט איזה פעולות לנקות.

ל-TF אין פקודות מיוחדות להדליקה / כבוי שלו, כפי שיש ל-IF (CLI, STI) למשל. ההדלקה או כיבוי הדגל נעשה בעקיפין בעזרת הפקודות POPF או IRET. בדרך כלל הדבר יעשה על בסיס הערך הנוכחי של אוגר הדגלים.

קוד אופיני להדלקת ה-TF הוא כלהלן:

```
PUSHF
POP AX
OR AX,10000000B
PUSH AX
POPF
```

קוד אופיני לכבוי ה-TF הוא כלהלן:

```
PUSHF
POP AX
AND AX,111111011111111B
PUSH AX
POPF
```

### ROM-BIOS-ה

את ה- ROM-BIOS אפשר לכנות "תוכנה באדיבות היצרן".

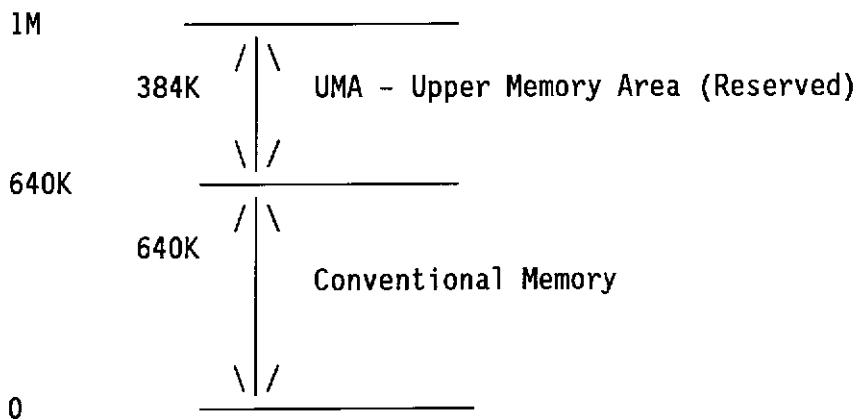
ראשי התיבות של ROM-BIOS הוא:

Read Only Memory, Basic Input Output System

מדובר ברוטינות אסמבלי (שעקרונית כל אחד יכול לכתוב), אשר מממשות בצורת ISR-ים. זו בעצם סיפריית רוטינות טיפול בפסיקות.

המיקום שלהם מבחינת כתובות זכרון הוא ב-384K של זכרון בין ה-640K הראשונים ל-1M או ה-Upper Memory Area או ה-UMA. אותו שטח מיוחד ששמור ל-SYSTEM כלומר לא האפליקציה וגם לא מערכות ההפעלה פועלים שם.

מרחב הכתובות 1M - 0 הראשונים ב-PC נראים כך:



הרוטינות של ה-BIOS מהוים סיפריית רוטינות שממשות פונקציות קלט / פלט בסיסיות. רובם ISR-ים של פסיקות תוכנה אך חלקם ISR-ים של פסיקות חומרה. רוטינות ה-BIOS מאפשרות לכותבי מערכות ההפעלה (DOS במקרה שלנו) לכתוב קוד שמבצע קלט / פלט בסיסי מבלי לגשת ישירות לחומרה. אחד היתרונות בכך, שלמרות שיש מספר גדול של יצרני PC ולמרות שיש הבדלים בין היצרנים, DOS עובד על כולם. הכותבים של DOS לא היו צריכים להביא בחשבון את ההבדלים בין היצרנים. בכדי ש-PC יהיה תואם, הוא לא חייב להיות זהה ליצרנים האחרים, מספיק שרוטינות ה-BIOS שלו (השונות מיצרן ליצרן) יעמדו בסטנדרטים מסוימים. אלמלא היה הדבר כך, לא רק שהיו בעיות בין יצרנים, אלא גם בעיות בין מודלים שונים של אותו יצרן (386, 286, 8086...). בכדי לקבל תחושה מה היה קורה אלמלא היה BIOS, שימו לב מה קורה כאשר מתקינים מדפסת או מודם או כרטיס רשת חדש למחשב: צריך להתקין תוכנה מיחדת הספציפית ליצרן של המכשיר החדש (הנקרא Device driver). מערכות הפעלה היום מסופקות עם דיסקים של סיפריית של

drivers של מאות יצרנים ומכשירים. זה היה המצב עבור כל סוג של קלט / פלט, אלמלא ה-BIOS.

יש לשים לב: בניגוד ל-INT 21h, שהם חלק ממערכת ההפעלה (DOS) רוטינות ה-BIOS אינם חלק משום מערכת ההפעלה. הם חלק מהמחשב והם משרתים כל מערכת הפעלה שמעוניין בשירותם. מערכת הפעלה אינה חייבת, כמובן, להשתמש בהם, חלקם או כולם, ואכן מערכות הפעלה כמו LINUX או NT בודאי מתעלמות מחלק מהם, משיקולים שונים.



רשימת חלקית של פסיקות חשובות

מספר פסיקה	שם	סוג הפסיקה	אחריות
0	Divide overflow	חריגה	DOS
5	Print-Screen	חומרה	BIOS
8	Timer	חומרה	BIOS
9	Keyboard	חומרה	BIOS
10h (16)	Video	תוכנה	BIOS
13h (19)	Floppy	תוכנה	BIOS
16h (22)	Keyboard	תוכנה	BIOS
18h (27)	Crtl-Break	חומרה	DOS
21h (33)	Function Request	תוכנה	DOS

הערות:

פסיקה מספר 21h הוא כמובן פסיקת התוכנה שהשתמשנו עד עכשיו ל-קלט/פלט דרך DOS (INT 21h).

פסיקות 9 ו-16h שניהם פסיקות מקלדת (Keyboard). זו אינה כפילות. פסיקה 9 היא פסיקת החומרה של המקלדת (מתרחשת עם כל לחיצה / שחרור של מקש). לעומת זאת פסיקה 16h היא פסיקת התוכנה של המקלדת - רוטינה שתוכניות קוראות לה ביוזמתם ע"י הפקודה INT 16h - על מנת לקבל קלט מהמקלדת, בדומה ל-INT 21h. למעשה INT 21h אופציה AH=1 משתמש ב-INT 16h בכדי לבצע את המשימה שלו.

צריך להיות ברור, למשל, שה-ISR של פסיקת ה-Ctrl-Break הוא פסיקה שהטיפול בה הוא באחריות DOS, שכן מדובר בחזרה למערכת ההפעלה, ורק מערכת ההפעלה יכולה לדעת לאן חוזרים. ה-BIOS לא יכול לדעת. כנ"ל לגבי Divide Overflow.

לעומת זאת, ה-ISR-ים של פסיקות Keyboard, Timer וכו' יכולות להיות BIOS, כי מדובר בגישה להתקני חומרה, שהם חלק מהמחשב בלי קשר לשאלה, איזה מערכת

הפעלה מותקנת.

### שימוש עקיף של ISR-ים של ה-BIOS

לפעמים מערכת הפעלה עומדת בפני דילמה: היא צריכה לקחת לעצמה את הטיפול בפסיקה מסוימת (פסיקת חומרה בדרך כלל) אך עדיין מעוניינת להסתמך על ה-ISR של ה-BIOS בכדי לתקשר עם החומרה. דרך אפשרית להתמודד עם הדילמה הזו היא לקרוא ל-ISR ה-BIOS מתוך ה-ISR החדש. הדרך הפשוטה ביותר לעשות זאת היא ע"י שימוש ב-CALL של פוינטר לפונקציה מסוג FAR. נראה דוגמא לכך בהמשך. באופן כללי השימוש באמצעי הזה נראה כך:

```
PUSHF  
CALL משתנה 32 ביט
```

כאשר המשתנה 32 ביט מכיל את הכתובת המלאה של ה-ISR של ה-BIOS.

### מימוש רוטינות טיפול בפסיקה ISR

בכל הקשור לכתיבת פסיקות תוכנה, אין הרבה הבדל בין פסיקת תוכנה לפרוצדורה. ההבדל היחיד המתחייב מהעובדה שמדובר בפסיקת תוכנה הוא, שהחזרה היא דרך הפקודה IRET (ולא RET).

בכתיבת פסיקת חומרה המצב שונה מאד. פסיקת חומרה מפסיקה (בדרך כלל) תוכנית שאינה קשורה לפסיקה. זה בודאי יכול לקרות לכל ISR חומרה. לכן ה-ISR צריך לדאוג לכך שהתוכנית המופסקת לא תושפע ע"י הפסיקה - התנהגותה צריכה להיות זהה למקרה שבו לא היתה פסיקה. זה כמובן למעט מקרים נדירים. בפועל הקריטריון הזה מתבטא, בראש ובראשונה, בשימור ערכים של כל האוגרים כולל אוגר הדגלים. זאת משום שהתוכנית המופסקת מסתמכת על ערכי האוגרים (מפקודת מכונה אחת לשניה) ואין אפשרות להמנע מכך. מאחר ו-IP, CS, ואוגר הדגלים נשמרים במחסנית בזמן הפסיקה עצמה, ה-IRET הוא זה שמשחזר אותם יחד, בפועל, עם אוגר ה-SP. יתר האוגרים חייבים להשמר ולהשתחזר ע"י צמדים של פקודות PUSH ו-POP בתחילת הטיפול בפסיקה ובסופה.

## פסיקת המקלדת INT 16h

בפסיקה מספר 16h הינה פסיקת התוכנה של המקלדת. תוכניות שמעונינות באינפורמציה מהמקלדת פונות אליה, בדרך כלל בעקיפין דרך מערכת ההפעלה והיא כאילו התוכנית הסטנדרטית לקרוא מידע מהמקלדת. זוהי פסיקת תוכנה של ה-BIOS ובעיקרו של דבר היא מהווה מעין קוד המכיר את מבנה הנתונים של ה-BIOS ומתפקדת התאם.

בשטח זכרון מיוחד בהתחלה של הזכרון של המחשב (מיד אחרי השטח של IV, כלומר מכתובת 1024 ואילך) ישנו שטח זכרון של ה-BIOS שמכיל בין השאר שטח המשקף את מצב המקלדת, איזה מקשים נלחצו וטרם נקראו ע"י שום תוכנית, מצב הנורות (Caps Lock, Insert, Scroll, Num), מצב מקשי הסטטוס (Shift, Alt), וכו'. למשל בבתים בכתובות מוחלטות 417h (1041) ו-418h (1042) ישנם בתים המהווים משתני דגלים של מצב הנורות ומקשי הסטטוס. נוסף לכך יש שם, חוצץ לשמירת מידע על עד ל-20 לחיצות מקשים שלא נקראו עדיין (מעבר לכך התגובה ללחיצות נוספות יהיה צפוף). השטח הזה מתחזק ומעודכן בעיקר ע"י ה-ISR של פסיקה מספר 9, פסיקת החומרה של המקלדת. המידע שנשמר שם הוא על לחיצות על (כמעט) כל המקשים של המקלדת, כולל המקשים F1 - F12, Page Down, Esc, Ins, Del ... וכו'.

כאשר תוכנית מבקשת מ-INT 16h אינפורמציה על לחיצת מקש, האינפורמציה מגיעה בשתי צורות: קוד Ascii וקוד Scan. קוד ה-Ascii הוא בדרך כלל מה שאנחנו צריכים, למשל אם נלחץ על המקש המסומן A נקבל או את הקוד Ascii 'a' או 'A' בהתאם למצב נורת ה-Caps Lock ומקשי ה-Shift וכו'. קוד ה-Scan הוא מספר יחודי לכל מקש, בתחום 1 - 127. צריך לזכור שיש מקשים שאין להם קוד Ascii (כמו F1, Esc, Page Down) ויש קודי Ascii שיש להם יותר ממקש אחד (למשל הספרות העשרוניות, הסימנים +, -, \*, /).

ל-INT 16h אופציות רבות, הנבחרות לפי הערך של האוגר AH כרגע הקריאה, השימושיות ביותר מבחינתנו הם:

INT 16h,

AH = 0, קריאת מקש:

קריאה ל-INT 16h כאשר AH = 0 פירושו בקשה לקבלת מידע על לחיצת מקש של המשתמש. אם אין מידע מסוג זה בהמתנה, החזרה מהקריאה תתעקב עד שחיצה כזו תתבצע (כלומר התוכנית תעצר לזמן בלתי מוגבל). עם החזרה (כין עם אחרי המתנה ובין שלא) יהיו באוגר AX האינפורמציה המבוקשת: ב-AL יהיה קוד ה-Ascii של המקש שנלחץ ואילו ב-AH יהיה קוד Scan של המקש. אותה לחיצת מקש שהמידע עליה מוחזרת לקורא ל-INT 16h נחשבת ל"נקראה" קרי "מבוטלת" כלומר שקריאה נוספת ל-INT 16h אופציה AH = 0 לא תתיחס אליה ותקרא את האינפורמציה

על הלחיצה הכאה. זה נראה מובן מאליו, אבל זה לא נכון לאופציה הכאה.

,INT 16h

,AH = 1, עיון במקש:

האופציה הזו דומה לאופציה AH = 0, אבל במספר הבדלים מהותיים. ראשית, הקריאה תמיד חוזרת מיד לקורא ל-INT 16h, גם אם אין אינפורמציה על לחיצת מקש שטרם נקראה. התוכנית הקוראת יכולה להבחין אם היתה לחיצת מקש או לא לפי הערך של הדגל ZF, ZF = 1 אם לא היתה אינפורמציה של לחיצת מקש בהמתנה לקריאה, אם היתה ZF = 0. במידה והיתה אינפורמציה ללחיצת מקש ממתינה, היא תחזור ב-AH כמו ב-INT 16h אופציה AH = 0, אך בניגוד למקרה הקודם הקריאה ל-INT 16h אופציה AH = 1 אינה "מבטלת" את הלחיצה. הקריאה הכאה ל-INT 16h באופציה AH = 0 או AH = 1 יקראו את אותו מקש עצמו. כלומר קריאה ל-INT 16h אופציה AH = 1 היא קריאה "לא מחייבת" של המקש, רק מעין הצצה לבדוק מה יש שם, אם בכלל. למשל, אם נרצה לבדוק אם יש אינפורמציה על לחיצת מקש ממתינה ובמידה ויש לקרוא אותה אבל מבלי להמתין ללחיצה כזו במידה ואין, אנחנו נקרא קודם ל-INT 16h אופציה AH = 1, נבדוק ש-ZF = 0 ובמידה וזה המצב, נקרא ל-INT 16h אופציה AH = 0.

הקוד יכול להראות כך:

```
MOV AH,1
INT 16h
JZ No_Inf01
MOV AH,0
INT 16h
JMP With_Inf01
No_Inf01:
```

....

```
With_Inf01:
```

,INT 16h

AH = 2, סטטוס המקלדת. מחזיר ב-AL אחד משני בתי הדגלים של המקשים המיוחדים (נורות ה-Caps Lock, Num, Scroll, מצב מקשי ה-Shift, Alt, Ctrl). מידע נוסף ניתן לקרוא מכתובת מוחלטת 418h או ע"י קריאה ל-INT 16h אופציה AH = 12h.

,INT 16h

AH = 5, הדמית מקש. אפשר בתוכנית ליצור מצב שבו "כאילו" נלחץ מקש כלשהוא, כאשר האינפורמציה של לחיצת המקש אותו רוצים לדמות מועברת דרך CX: CL יהיה קוד ה-Ascii ו-CH קוד ה-Scan.