תקציר מספר 7

פסיקות (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 במובן הזה, שכותב התוכנית (בדרך כלל) אינו מכיר את הרוטינה החלופית, אינו מעונין בעצירת התוכנית שלו ויכול להיות שהרוטינה החלופית אינה משרתת את מטרותיו 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 במצב זהה לרגע שבה היא הופסקה.

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

כאשר תוכנית רגילה מתבצעת, <u>הפקודה הבאה לביצוע</u> 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 בזיכרון מיד אחרי הפקודה המתבצעת הנוכחית.

בשני המקרים הפקודה הבאה לביצוע נמצאת בתוך התוכנית המתבצעת.

<u>פסיקה</u> 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 במהלך הריצה של התוכנית הנוכחית. חלוקה באפס (divide overflow) היא דוגמא קלאסית לכך. דוגמאות אחרות הם נסיון לבצע פקודת מכונה לא חוקית, נסיון לגשת לכתובת לא קיימת בזיכרון, 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 בזיכרון (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.

לגבי 1. 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 לזיכרון ומקבלת שליטה. הסתעפויות CALL מוגבלות לנקודות זיכרון הנמצאים ב"תמונה" הזו של קובץ ה-EXE בזיכרון. 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 לזיכרון. זה כמובן נכון גם עבור קוד מקורי שנכתב בתוכנית עצמה. רוטינות קלט / פלט כמו printf, fopen הינם קודים שנמצאים בקוץ 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 יכלות 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 255 .. 0. המערך טבלה הזה נקרא Interrupt Vector או IV. ב-8086 ובמצב real mode של ה-x86 המתקדמים יותר, הפוינטרים האלו הם כולם 32bit (16 ל-offset, 16 ל-segment) והמערך נמצא ב-1K הכתובות הפיזיות הראשונות (1023 .. 0) 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 39 - 36. ה-offset נמצא בכתובות 37 - 36, וה-segment בכתובות 38

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

<u>מה גורם לפסיקה?</u>

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

<u>בפסיקות תוכנה</u> - ע"י ביצוע של פקודת המכונה INT.

<u>מנגנון מימוש הפסיקות</u>

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.

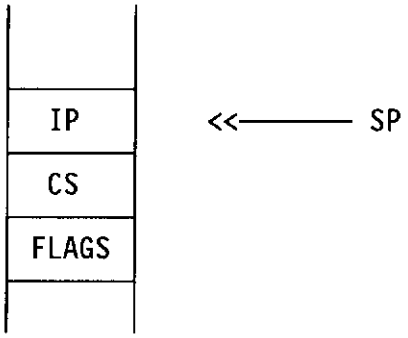
על כך נרחיב מאוחר יותר. כרגע נציין שהדבר מונע התרחשות פסיקות נוספות, תוך שהמחשב עסוק בטיפול בנוכחית.

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).

<u>הסתיגות</u>

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 <u>הפקודה שהופסקה</u>. 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-<u>Interrupt Flag</u> (IF)

בתוך אוגר הדגלים קיים דגל ה-Interrupt Flag או IF, השולט על מנגנון הפסיקות. כאשר IF = 0, מנגנון פסיקות <u>החומרה</u> <u>מושבת</u>. ה-CPU <u>מתעלם</u> מבקשות פסיקות <u>חומרה</u>. כאשר IF = 1, מנגנון פסיקות החומרה <u>פעיל</u>. 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.

<u>פקודות מכונה שקשורות למנגנון הפסיקות</u>

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 שחזור אוגר הדגלים (לערך שישנו במחסנית) ושינוי ה-CS:IP לערכים שנשלפו מהמחסנית.

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

ה- Trap Flag (TF)

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,100000000B
PUSH AX
POPF
```

קוד אופיני לכבוי ה-TF הוא כלהלן:

```
PUSHF
POP AX
AND AX,1111111011111111B
PUSH AX
POPF
```

ה-ROM-BIOS

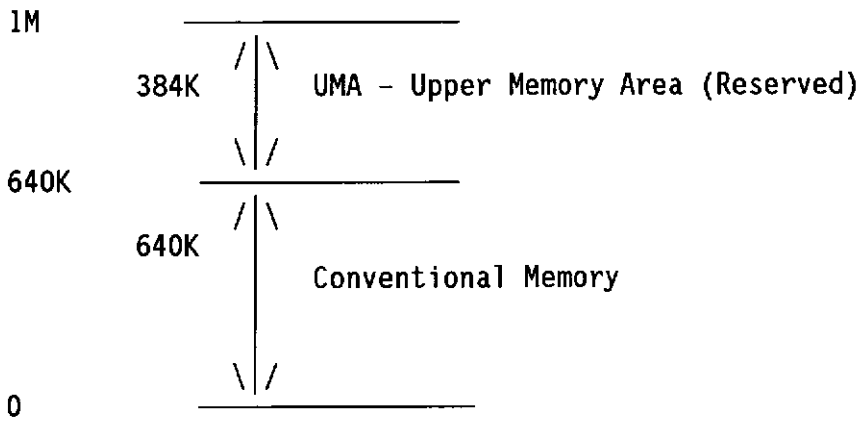
את ה- ROM-BIOS אפשר לכנות "תוכנה באדיבות היצרן".

ראשי התיבות של ROM-BIOS הוא:
.Read Only Memory, Basic Input Output System

מדובר ברוטינות אסמבלי (שעקרונית כל אחד יכול לכתוב), אשר מממומשות בצורת ISR -ים. זו בעצם <u>סיפרית רוטינות טיפול בפסיקות</u>.

המיקום שלהם מבחינת כתובות זיכרון הוא ב-384K של זיכרון 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 שממממשות 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 <u>מבלי לגשת ישירות לחומרה</u>. אחד היתרונות בכך, שלמרות שיש מספר גדול של יצרני PC ולמרות <u>שיש</u> הבדלים בין היצרנים, DOS עובד על <u>כולם</u>. הכותבים של DOS <u>לא</u> היו צריכים להביא בחשבון את ההבדלים בין היצרנים. בכדי ש-PC יהיה תואם, הוא <u>לא</u> חייב להיות זהה ליצרנים האחרים, מספיק ש<u>רוטינות ה-BIOS</u> 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 (8086, 286, 386 ...). 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 סיפריות 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 שמעונין 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
1Bh	(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' יכלות 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 המתחיב 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 נדרים. 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.

TABLE 5.2 INTEL DOS AND BIOS INTERRUPTS

Address				
SEG:OFF	20-bit	Type	Purpose	Description
00E3:3072	03EA2	Intel	Divide Overflow	This interrupt occurs when a divide overflow takes place. The interrupt vector varies with the DOS version.
0600:08ED	068ED	Intel	Single-step	This interrupt simulates single-step execution. IBM uses the Trace command in DEBUG to accomplish this task.
F000:E2C3 (F000:F85F – XT)	FE2C3	BIOS	Non-Maskable Interrupt	This interrupt cannot be prevented. It calls the BIOS NEAR procedure NMI_INT and results from memory errors on the system board (PARITY CHECK 1) or add-on boards (PARITY CHECK 2).
0600:08E6	068E6	Intel	Set Breakpoint	This interrupt stops the processing at a particular address.
0070:0147	00847	Intel	Interrupt If Overflow	This interrupt activates an INTO instruction return (IRET).
F000:FF54	FFF54	BIOS	Print Screen	This interrupt prints the screen under program control. The FAR procedure called is PRINT_SCREEN and the address 0050:0000 contains the status.
—	—	—	—	Not used.
—	—	—	—	Not used.
F000:FEA5	FFEA5	BIOS	Timer Interrupt	This routine handles the timer interrupt from channel 0 of the 8253 timer. There are approximately 18.2 interrupts/second. This handler maintains a count of the number of times it was called since power up. The FAR procedure is TIMER_INT and it calls an interrupt 1C H, in turn, which can contain a user routine.
F000:E987	FE987	BIOS	Keyboard Interrupt	This routine is FAR procedure KB_INT. It continues to address F000:EC32 and constitutes the keyboard interrupt. INT 16H is the keyboard I/O routine and is much more flexible.
—	—	—	—	Not used.
—	—	—	—	Not used.
—	—	—	—	Not used.
—	—	—	—	Not used.
F000:EF57	FEF57	BIOS	Floppy Diskette	This FAR procedure, DISK_INT, handles the diskette interrupt.
0070:0147	00847	DOS	INTO	This interrupt activates the same call as TYPE 4.
F000:F065	FF065	BIOS	Video Interface	This set of routines contained in the NEAR procedure VIDEO_IO provides the CRT interface. The use of this interrupt involves many options which are discussed in a subsequent table.
F000:F84D	FF84D	BIOS	Equipment	This procedure looks for the number of printers, any game I/O, the number of RS-232C cards, number of diskette drives, video mode, and RAM size.
F000:F841	FF841	BIOS	Memory Size	Determines the memory size from data.
F000:EC59 (C800:0256 – XT)	FEC59	BIOS	Diskette I/0	This procedure calls a series of routines that accomplish diskette I/O. Since a number of parameters are involved, this routine will be discussed separately.
F000:E739	FE739	BIOS	Communications Adapter	This procedure lets the user input/output data from the RS-232C communications port.
F000:F859	FF859	BIOS	Cassette I/O	This interrupt is used to control cassette I/O.
F000:E82E	FE82E	BIOS	Keyboard I/O	This interrupt manipulates AX to read the keyboard. It will be discussed separately
F000:FEFD2	FEFD2	BIOS	Printer I/O	This routine provides communication with the printer. It uses the AX and DX registers to setup parameters. We will discuss it later.
F600:000	F6000	BIOS	Cassette BASIC	This interrupt calls cassette BASIC.
F000:E6F2	FE6F2	BIOS	Bootstrap	Track 0 sector 1 of Drive A is read into the boot location. Control is transferred there.
F000:FE6E	FFE6E	BIOS	Time-Of-Day	This routine allows the clock to be set/read. CX contains the high portion of the count and DX the low portion.
0070:0140	00840	DOS	Ctrl-Break	This interrupt results when a keyboard interrupt is used.
F000:FF53	FFF53	BIOS	Dummy Return	This interrupt simply calls an IRET instruction.
F000:F0A4	FF0A4	BIOS	Video Parameters	This is simply a table of byte values and routines for setting up various graphics parameters.
0000:0522	00522	DOS	Floppy Table	
00E3:0B07	01937	DOS	Graphics Table	Used with DOS 3.0
PSP:0000	—	DOS	Program Terminate	This interrupt is issued by DOS to exit from a program. It is the first address in the Program Segment Prefix area.
relocatable	—	DOS	Function Request	This interrupt has many options and will be discussed later.
PSP:000A	—	DOS	Terminate Address	Control transfers to the address specified at this interrupt location when the program terminates. Do not issue this address directly.
PSP:000E	—	DOS	Ctrl-Break Exit Address	This interrupt is issued in response to a Ctrl-Break from the standard input.

(Continued)

TABLE 5.2 *(Continued)*

Int.	Address SEG:OFF	Address 20-bit	Type	Purpose	Description
24	PSP:0012	—	DOS	Critical Error Handler Vector	This interrupt is called when a critical error occurs within DOS such as a disk error.
25	relocatable	—	DOS	Absolute Disk Read	This interrupt transfers control to the device driver for a read.
26	relocatable	—	DOS	Absolute Disk Write	This interrupt transfers control to the device driver for a write.
27	relocatable	—	DOS	Terminate But Stay Resident	This vector is used by programs to remain resident after DOS regains control.
28-2E		RESERVED FOR DOS			
2F	relocatable	—	DOS	Multiplex Interrupt	This interrupt defines a general interface between 2 processes. Each handler is assigned a specific number in AH and the function of the handler in AL.
30-3F		RESERVED FOR DOS			

Table 5.2 is rather extensive and contains most of the interrupt vectors for the IBM PC and XT [2]. Most of these interrupts will not be useful from a programming viewpoint; however, the interrupts indicated in the description as affecting such things as I/O can make the programmer's job much simpler. Three types of interrupts are discussed: Intel, BIOS, and DOS. The Intel interrupts are common to most Intel processors in the 8086 family. The BIOS interrupts are specific to the IBM Basic Input Output System and the DOS interrupts are, of course, specific to the IBM Disk Operating System [3]. The BIOS interrupts are explained in the BIOS program listing discussed above, and the DOS interrupts are described in the *DOS Technical Reference* manual [4].

5.1.1 Interrupts 0–0FH

Interrupts 0–4 are not particularly useful to the assembler programmer because they tend to call system oriented functions. INT 5 can be used to print the contents of the screen Cathode Ray Tube (CRT) under program control. If, for example, a program generates output to the screen, it might be desirable to automatically save this output and INT 5 is useful for this purpose. INT 8 is the Timer Interrupt and returns a time count in locations 0040:006C and 0040:006E where these are the low and high count, respectively. This 32-bit time count can be used to time events in the program. INT 9 activates when keys are pressed or released and simply stops the processing. This interrupt is more of a system interrupt. The INT 16H keyboard I/O interrupt will be discussed in detail rather than focusing on INT 9. Both address the keyboard operation; however, INT 16H has considerable flexibility over INT 9. Interrupts 0AH to 0FH tend to be either unused at present or system oriented and will not be discussed further. INT 10H, however, is the video I/O routine already mentioned. This interrupt has many options as discussed in the program listing in the *IBM Technical Reference* manual. Essentially all screen I/O under program control can be implemented using this interrupt. The following discussion addresses this interrupt in some depth.

5.1.2 Interrupt 10H: Video I/O

The type 10H interrupt has 16 basic options controlled by the AH register at the time the interrupt is called. Table 5.3 describes these options and the registers that must be set to control output. The mode option (AH = 0) determines the screen resolution and what graphics are available. To handle the increased memory requirements for the graphics modes, additional video memory must be provided and is usually supplied with the graphics card purchased. The following discussion illustrates the use of this video interrupt and how to achieve screen graphics.

All standard screen modes involve the use of a raster scan which simply means the electron beam moves vertically downward after scanning across for each line from left to right. The resolution of a dot in one of the graphics modes is determined by the number of lines vertically down the screen and by the number of columns horizontally across the screen. In the 640 × 200 graphics mode, for example, the height of the screen can be thought of as 200 dots high. Similarly, the screen width would be 640