# מנגנון בסיסי לטיפול בפסיקות

לחומרת המעבד יש קו פסיקות

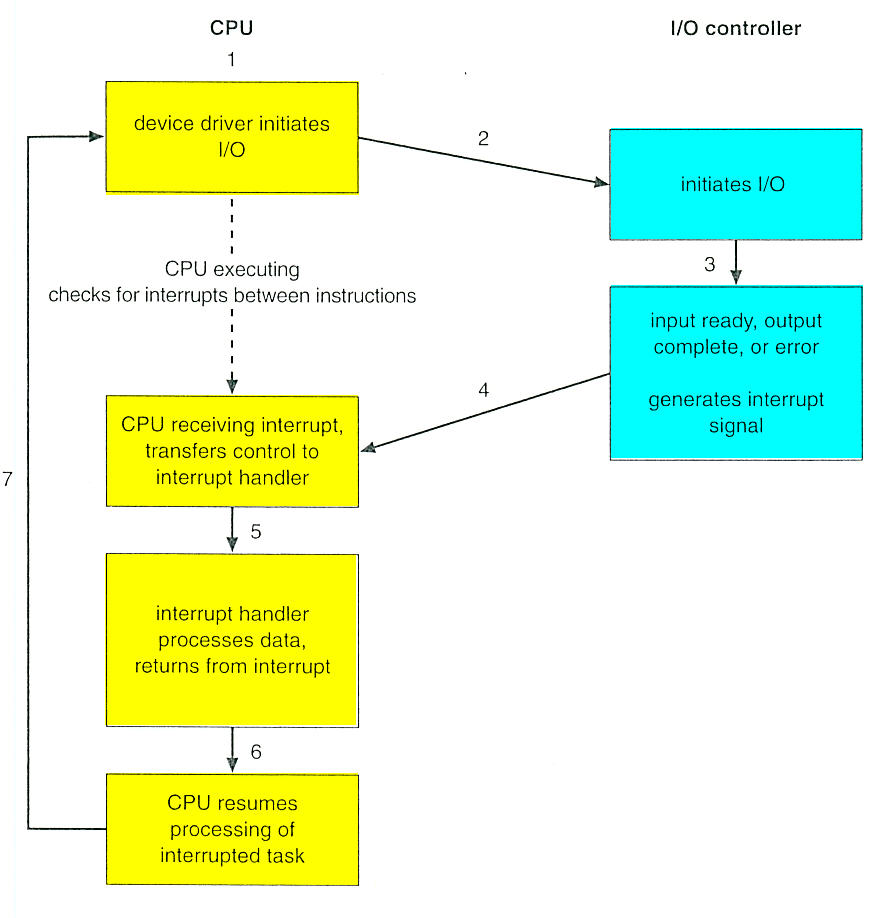
(**interrupt request line**) אותו היא בוחנת לאחר ביצוע כל פקודת מכונה. כאשר מאובחנת על קו הפסיקות פנייה מהתקן (פסיקה), נשמרת אינפורמציה מספקת בכדי לשחזר את המצב הנוכחי (בו רץ תהליך) ומאותרת שגרה אשר הוגדרה מראש לטיפול בפסיקה (המסויימת הזו)

שגרת הטיפול בפסיקה

(**ISR = interrupt service routine**) מבצעת את החישוב הנדרש לטיפול בפסיקה. השגרה מסתיימת ע"י ביצוע פקודת **return from interupt (iret)** אשר משחזרת את המצב לפני תחילת הטיפול בפסיקה. מנקודת מבטו של התהליך הנוכחי הטיפול בפסיקה אמור להיות שקוף לחלוטין

פסיקה אסינכרונית או פסיקה חיצונית - פסיקה הנגרמת על ידי רכיב חומרה, באופן שאינו תלוי בריצת התוכנית הנוכחית.

פסיקה סינכרונית או פסיקה פנימית - פסיקה הנוצרת על ידי החומרה או התוכנה במועד ידוע מראש עקב ביצוע פקודה מסוימת בתוכנית. פסיקה כזו קרויה "סינכרונית" משום שהיא צפויה - ניתן לחזות את התעוררותה כאשר יתבצע קטע הקוד המתאים.

סכימה כללית של מנגנון הפסיקות חומרה

**מנגנון הפסיקות הבסיסי שתואר מאפשר למעבד להגיב למאורעות אסינכרוניים (למשל בקר התקן ק/פ מוכן לשירות). כמובן, התיאור הנ"ל הנו כללי מאד והתעלמנו בו ממספר פרטים חשובים. למשל:**

**נרצה אפשרות לדחות הטיפול בפסיקות בעת עיבוד קטעים קריטיים, נרצה שאיתור שגרת הטיפול בפסיקה והפעלתה הנכונה יהיו יעילים**

**מ"ה מודרניות עשויות לדרוש אבחנה בין עדיפויות של פסיקות כך שהטיפול בפסיקות יהיה מותאם לרמת הדחיפות הנדרשת (multilevel interrupts)**

**לרוב המעבדים יש שני קווי פסיקות:**

**nonmaskable: שמור עבור מאורעות דחופים כמו חריגות זיכרון מסוכנות**

**maskable: ניתן לכיבוי (ע"י המעבד) למשל לפני ביצוע סידרת פקודות קריטיות**

**פסיקות חומרה הן אסינכרוניות**

# פסיקות מצב חריג

פסיקות מצב חריג (exception) משמשות לבקרת מצבים חריגים (ומסוכנים) היכולים להיווצר במערכת

דוגמאות למצבים חריגים: חלוקה ב- 0, ירידת מתח, גישה לכתובת מוגנת (או לא קיימת), ביצוע פקודה לא חוקית

המכנה המשותף לכל המצבים החריגים: נוצר מצב בו נדרשת תגובה דחופה

מטרת פסיקות מצב חריג היא, מצד אחד להבטיח שלא יגרם נזק למערכת, ומצד שני לחסוך זמן בדיקות של המשתמש ולעזור לו בהרצת ודיבוג תוכניות

הטכניקה של טפול בחריג דומה לאופן הטפול בפסיקות חומרה.

פסיקות מצב חריג הן בעלות עדיפות יותר גבוהה מפסיקות חומרה (הן **nonmaskable**)

**פסיקות מצב חריג הן סינכרוניות**

# פסיקות תוכנה

לקוד מערכת ההפעלה יש "זכויות יתר" אשר אין לקוד משתמש. לדוגמא קוד משתמש אינו יכול לגשת להתקנים חיצוניים (ברוב מערכות ההפעלה), קוד מערכת הפעלה יכול לגשת לקבצים אשר אינם מורשים למשתמשים אחרים, וכו,

למשל, הפקודה PS ב-UNIX, המותרת לכל משתמש, מחזירה מידע על כל התהליכים במערכת. המידע המבוקש שמור במקומות שאינם נגישים למשתמש

על מנת למנוע מהמשתמש קבלת זכויות יתר, אנו מגבילים את היכולת שלו, לפנות למערכת ההפעלה באופן "מסודר", באמצעות מנגנון פסיקות תוכנה. טכניקת הטיפול בפסיקות תוכנה דומה לטכניקת הטיפול בפסיקות חומרה או חריג.

כאשר תהליך משתמש מבצע **system call** הוא מעוניין בשרותי גרעין מ"ה. קוד ה-**system call** מארגן את הפרמטרים הדרושים להעברה לפונקציית הגרעין ואז יוזם פסיקת תוכנה (הנקראת גם **trap**). כתגובה לפסיקה נקראת שגרת טיפול אשר מפעילה את פונקציית הגרעין הדרושה עם פריווילגיות של מ"ה

**פסיקות תוכנה הן סינכרוניות**

# ההבדל בין פסיקה לקריאה לפונקציה

**ניתן להסתכל על פסיקות כמקבילה של קריאה לפונקציה, בהבדלים הבאים:**

**קריאה לפונקציה מתבצעת מתוך קוד התהליך רץ. הקריאה לפונקציית הטיפול בפסיקה איננה נמצאת בקוד התהליך, אלא מסומלצת ע"י מ"ה**

**פסיקה יכולה להיות סינכרונית (חריג+תוכנה) או א-סינכרונית (חומרה).**

**הפסיקה מעבירה את החומרה ממצב משתמש למצב מערכת בו יש למערכת זכויות נוספות**

**פסיקות הן בקשות לשירות שמתורגמות לקריאה לפונקציה ואינן קריאות ישירות**

**ביציאה מטיפול בפסיקה המערכת חוזרת בדיוק למצבה לפני הפסיקה (אם מבוצע חלוף תהליכים, התהליך החוזר מוחזר בדיוק למצב בו הופסק)**

# מנגנון הפסיקות ב-xinu

כללית, XINU מממשת את המנגנון שתארנו. מתאימה את עצמה לחומרת ה-PC ולארכיטקטורת Intel

לכל פסיקה יש מספר זיהוי בגודל בית אחד (כלומר 0-255)

מספר זה משמש כאינדקס לטבלת הפסיקות (וקטור הפסיקות) אשר מכילה את הכתובות של הפרוצדורות אשר מטפלות בכל פסיקה (**ISR**)

1024 הבתים הראשונים בזיכרון מוקדשים לווקטור הפסיקות

כל כניסה בווקטור הפסיקות הינה בת 4 בתים (32 ביט) עבור **segment:offset**

הכתובת לטיפול בפסיקה מספר N נמצאת בארבעת הבתים שמתחילים בכתובת N\*4

הערה: XINU תוכננה לארכיטקטורת 8086, אולם מבנה וקטור הפסיקות שונה ב- 486. לכן היא משתמשת במודל Intel הישן של ווקטור פסיקות.

# מנגנון הפסיקות

כאשר התקן חיצוני רוצה להתקשר עם המעבד הוא שולח סימן.המעבד בודק את קו הפסיקות בין כל שתי פקודות מכונה (במידה ודגל הפסיקות **IF** מורם)

המעבד מכבה את דגל הפסיקות ומבקש מההתקן אות זיהוי . ההתקן מחזיר בקו הנתונים את סוג הפסיקה (מספר N בין 0 ל 255)

ערך רגיסטר ה- **flags** הנוכחי (שלפני כיבוי דגל הפסיקות) וכן כתובת הפקודה הבאה לביצוע (**CS:IP**) נדחפים על המחסנית

המעבד טוען את תוכן תא זכרון **4\*N** לתוך רגיסטרים הפקודה (**CS:IP**)

המעבד ממשיך לבצע פקודות החל מהכתובת **CS:IP** ( פקודות לטיפול בפסיקה)

בגמר ביצוע הקוד לטיפול בפסיקה חוזר המעבד לפקודה שעמד לבצע לפני הפסיקה ע"י פקודת המכונה **iret**.

פקודת **iret** שונה מפקודת החזרה הרגילה **ret** שכן היא צריכה לשחזר גם את **flags** (בנוסף ל- **CS:IP**)

פסיקות תכנה

**XINU** משתמשת באותו מנגנון גם עבור פסיקות תוכנה

במקרה כזה הפסיקה מתבצעת באופן סינכרוני המזכיר יותר קריאה לפונקציה

דגל הפסיקות אינו משפיע על ביצוע פסיקת תכנה (**nonmaskable**) פסיקת תכנה מתבצעת ע"י פקודת המכונה **int n**

| פסיקות תוכנה | פסיקות חומרה |
| --- | --- |
| פסיקת תוכנה מס’ n מתבצעת ע”י פקודת המכונה  **int n**  גם בפסיקות תוכנה, המעבד מכבה את דגל הפסיקות ! | התקן המבקש פסיקת חומרה מסמן למעבד ע”י שינוי מתח על קו הפסיקות (אחת הכניסות למעבד).  בין כל שתי פקודות מכונה כאשר **1=IF** בודק המעבד את קו הפסיקות. אם מתבקשת פסיקה:  1. מכבה את **IF**.  2. שולח **ack** להתקן.  בתגובה ההתקן שולח את מספר הפסיקה**n** על קווי הנתונים.  פקודת המכונה הבאה תתבצע אחרי החזרה מהפסיקה. |
| המעבד שומר על המחסנית את **Flags** (שלפני כבויו של דגל הפסיקות) ואת**CS:IP** , טוען את **CS:IP** מווקטור הפסיקות (החל מכתובת **n4**), וממשיך בביצוע מ-**CS:IP** החדש. | |
| החזרה מהפסיקה מתבצעת ע”י פקודת המכונה **iret** המשחזרת את ערכי **Flags** ו-**CS:IP** מהמחסנית (בחזרה מפונקציה רגילה, אין צורך לשחזר את **Flags**). | |
| פסיקת תוכנה זה כמו קריאה לפונקציה, ושמירת הרגיסטרים נעשית לפי מוסכמה מה לשמור ומה לא צריך לשמור. | באחריות הקוד של פסיקת חומרה לשמור את ערכי כל הרגיסטרים כך שלאחר סיום הפסיקה ערכי כולם ישוחזרו למצבם הקודם. |
| פסיקת תוכנה נוצרת ע”י התהליך המתבצע ויכולה להחזיר לו ערכים. | פסיקת חומרה שקופה לתהליך המתבצע. |
| פסיקת תוכנה מתבצעת תמיד כשהיא מתבקשת. | פסיקת חומרה נמנעת (נדחית) כאשר **IF** מכובה. |

בציור מצויינת דוגמה של הצבעה על פסיקות השעון המקלדת. שאר הפסיקות מוצבעות בדרך דומה.

קוד לפסיקת שעון

קוד פסיקת המקלדת

וקטור הפסיקות

1K

כניסת פסיקת השעון

כניסת פסיקת המקלדת

0

וקטור הפסיקות מאותחל בזמן עליית מערכת ההפעלה. עד אתחול וקטור הפסיקות אין לאפשר ביצוע פסיקות, ודגל הפסיקות צריך להיות כבוי.

בזמן אתחול וקטור הפסיקות מוצבת הכתובת של קודי הפסיקה השונים לתוך הכניסות המתאימות בוקטור הפסיקות.

## עדכון וקטור הפסיקות

לעתים נרצה לשנות את קודי הפסיקה של פסיקות מסוימות בזמן ריצת מערכת ההפעלה. למשל, מ”ה המאפשרות הטענת **DRIVER**-ים חדשים להתקנים או החלפת ישנים בזמן ריצת המערכת, צריכות לשנות את ההצבעה של הפסיקות המתאימות בווקטור הפסיקות כדי שיצביעו על ה-**DRIVER**-ים החדשים (במקום על הישנים).

שינוי ההצבעה בוקטור הפסיקות נקרא: **Revectoring**.

במקרים אחרים נרצה להוסיף עוד פעולה לביצוע בזמן פסיקה מסוימת, אך לא לבטל את הפעולות המתבצעות בקוד הפסיקה הקיים. בדר”כ נרצה שהקוד הנוסף שלנו יתבצע מיד לאחר הקוד הקיים (בפרט אם הוא משתמש בפלט של הקוד הקיים).

לשם כך, נבצע **Revectoring** לקוד הנוסף, והקוד הנוסף יהיה אחראי לקרוא לקוד הישן (בדומה לקריאה לפונקציה).

כתובת הקוד הישן תשמר בזמן ביצוע ה-**Revectoring** כדי שהקוד החדש יוכל לקרא לקוד הישן.

## Revectoringדוגמאות ל-

קוד פסיקה מקורי

קוד פסיקה תחליפי

וקטור הפסיקות

**ההצבעה המקורית**

קוד פסיקה מקורי

קוד פסיקה תחליפי

וקטור הפסיקות

**ההצבעה לאחר ה-Revectoring**

## Revectoringדוגמאות ל-

קוד פסיקה מקורי

קוד פסיקה תחליפי

וקטור הפסיקות

**ההצבעה לאחר ה-Revectoring עם קריאה לקוד הישן**

הקריאה לקוד הישן נעשית כקריאה לפונקציה ולא באמצעות פסיקה!

בזמן ה-**revectoring** נשמרת כתובת הקוד הישן לשם איפשור הקריאה לו, לאחר מכן.

## XINUוקטור הפסיקות ב-

**ROM BIOS**

**וקטור פסיקות (cs:ip)**

בזמן אתחול **XINU** וקטור הפסיקות כבר מצביע על שירותי ה-**BIOS-ROM** השונים. אנו מעונינים להחליף חלק משירותים אלו בפונקציות **XINU** כדי לאפשר ביצוע פעולות **XINU** מתוך פסיקות.

בעיה: לא ניתן להצביע על פונקציה ב-C מתוך וקטור הפסיקות, שכן בפסיקה **Flags** נטען על המחסנית, אך בפונקציה לא, ולכן הפונקציה לא תדע לשחזר את **Flags** ולחזור למקום ממנו נקראה הפסיקה.

# שיגור פסיקות ב-XINU

ב-**XINU** קיימת פונקצית אסמבלר המופעלת בעת הפסיקות, והיא מבצעת את הקריאה לפונקציות ה-**C**. היא מאפשרת כתיבת קוד הפסיקות בשפת **C**, במקום לכתוב את כל קוד הפסיקה באסמבלר. פונקצית האסמבלר נקראת **intcom**.

בעיה: קוד ה-BIOS-ROM מבצע שירותים שונים שאין אנו מעונינים לבטלם ע”י ביטול ביצועם בזמן הפסיקה, אך מצד שני אין אנו מעונינים לכתוב הקוד מחדש בפונקציות הפסיקה של **XINU**.

פתרון: בפסיקות **XINU** בהן יש צורך בביצוע קוד ה-**BIOS-ROM**, הוא יקרא לפני ביצוע פונקצית ה-**XINU**, מתוך פונקצית האסמבלר. פונקצית האסמבלר תקרא הן לקוד ה-BIOS-ROM שהוצבע במקור מתוך וקטור הפסיקות וכן לקוד הפסיקה של **XINU**. בפסיקות מסוימות לא נרצה לקרא לקוד ה- **BIOS** (למשל כאשר מוקש **ctrl-break**).

היישום:המערך **sys\_imp** כולל את המידע הנדרש לפסיקות **XINU**. לכל פסיקה מוקצית רשומה הכוללת את המידע הבא:

* פקודת המכונה **call intcom** .
* ההצבעה של קוד ה-**BIOS** שהוצבע במקור מוקטור הפסיקות.
* הצבעה על פונקצית הפסיקה של **XINU**. מספר הפסיקה בוקטור הפסיקות.
* זיהוי פנימי של ההתקן.
* דגל המסמן אם לקרא לקוד ה-**BIOS** לפני הקריאה לקוד **XINU**.

call intcom

call

intcom

call intcom

**וקטור פסיקות**

**מערך sys\_imp**

**קוד XINU**

**ROM-BIOS**

מצביע מקורי

**intcom** (פונקציית אסמבלר)

**intcom מבצעת את הקריאה לקוד ה-ROM-BIOS ולקוד הפסיקה של XINU.**

**io.h**

**#define INTVECI inint /\* input interrupt dispatch routine \*/**

**#define INTVECO outint /\* output interrupt dispatch routine \*/**

**extern int INTVECI();**

**extern int INTVECO();**

**#define NMAPS 0x20 /\* number of intmap entries \*/**

**struct intmap { /\*device-to-interrupt routine mapping \*/**

**char ivec; /\* interrupt number \*/**

**char callinst; /\* the call instruction \*/**

**word intcom; /\* common interrupt code \*/**

**word oldisr\_off; /\* old int. service routine offset \*/**

**word oldisr\_seg; /\* old int. service routine segment \*/**

**int (\*newisr)();/\*pointer to the new int. ser. routine \*/**

**word mdevno;/\* minor device number \*/**

**word iflag;/\*if nonzero, call the old isr \*/**

**};**

**/\* NOTE: The intmap structure takes a total of 7 words or 14 bytes per record. \*/**

**extern struct intmap far \*sys\_imp; /\*pointer to intmap table \*/**

**extern int nmaps;/\*number of active intmap entries \*/**

**#define isbaddev(f) ( (f)<0 || (f)>=NDEVS )**

**/\* In-line I/O procedures \*/**

**#define getchar() getc(CONSOLE)**

**#define putchar(ch) putc(CONSOLE,(ch))**

**#define fgetc(unit) getc((unit))**

**#define fputc(unit,ch) putc((unit),(ch))**

**extern int \_doprnt(); /\* output formatter \*/**

# איך פועלת intcom

וקטור הפסיקות מצביע על פקודת call intcom. לכן בעת פסיקה תקרא **intcom** (כפונקציה) מתוך הפסיקה. **intcom** היא פונקציה כללית המשרתת את כל הפסיקות. איך היא תדע לאיזה שגרות פסיקה היא צריכה לקרא? הפתרון: ע”י מיקום פקודת **call intcom.** מיקום פקודה זו (למעשה כתובת הבית שאחריה) נמצא על המחסנית ככתובת החזרה.

SP לפני הפסיקה

Flags

CS

IP

IP

bp

ax

bx

cx

dx

si

di

ds

es

SP בתחילת הפסיקה

bp בזמן פעולת intcom

לפני הקריאה ל-intcom

כתובת חזרה מ-intcom

ivec

call

intcom

IP

CS

new isr

mdevno

iflag

old isr

**המחסנית**

**רשומת הפסיקה ב-sys\_imp:**

**intcom מקבל את רשומת הפסיקה הכוללת**

וקטור הפסיקות

**את שגרות הפסיקה ע”י התיחסות ישירה לכתובת החזרה שלו. בהנתן רשומת הפסיקה ניתן לקרא לפי הצורך ל-oldisr ול- newisr.**

## שליפת כתובות קודי הפסיקה לביצוע

**כאמור, כתובת החזרה של intcom הנמצאת על המחסנית מצביעה לתוך רשומת הפסיקה. כתובת החזרה נמצאת בכתובת** 2 + ss:bp **במחסנית. ע”י ביצוע** [2 + bp], bx mov **אנו מקבלים ב- bx את “כתובת החזרה” המצביעה לתוך רשומת הפסיקה. כתובת שגרת הפסיקה המקורית נמצאת היא** ([bx]:cs):([2 +bx]:cs) **כתובת שגרת הפסיקה של XINU היא** [4 +bx]:cs **כתובת mdevno היא** [6 +bx]:cs **וכתובת iflag המציין אם לקרא לשגרה המקורית היא** [8 +bx]:cs

## intcom החזרה מ-

intcom **נקרא כפונקציה מתוך פסיקה. אולם כתובת החזרה שלו אינו מצביעה על פקודות מכונה לביצוע ולכן אסור לו לבצע ret.**

**הפתרון הפשוט: להוסיף לאחר פקודת** **call intcom ברשומה** **פקודת iret מהפסיקה. כלומר**

ivec

call

intcom

iret

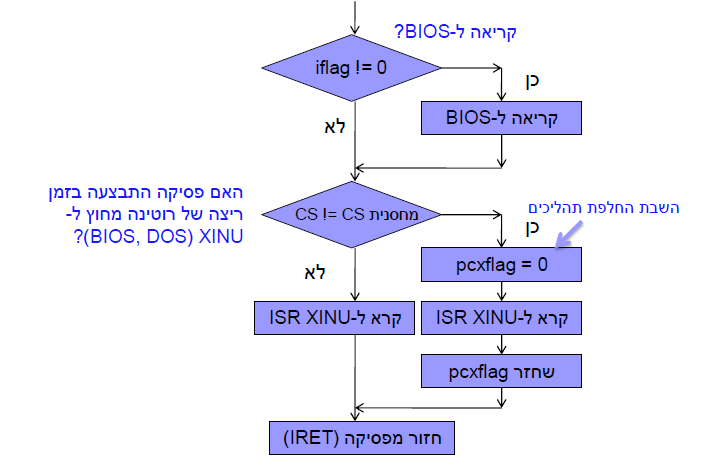
**מה יתבצע במקרה כזה?**

**intcom תבצע ret: הוצאת כתובת החזרה מהמחסנית ומעבר הביצוע לכתובת החזרה (כלומר ביצוע iret).**

**ביצוע iret: הוצאת IP:CS ו- Flags מהמחסנית וטעינתם ברגיסטרים (קפיצה ל- IP:CS).**

**בסך הכל אנו מוציאים את כתובת החזרה של intcom מהמחסנית ומבצעים iret, שכן ידוע שהפקודה לאחר החזרה תהיה iret, ולכן האפקט של ret הוא הוצאת 2 בתי כתובת החזרה מהמחסנית וביצוע iret.**

**בקיצור ניתן לבצע זאת ע”י הוצאה מהמחסנית וביצוע iret ישירות מ- intcom, וזאת ע”י add sp,2 וקריאה ל- iret.**

****

**intmap.asm**

**; low-level interrupt transfer table and dispatcher**

**include dos.asm**

**tblsize equ 20h ;define max size of intmap table**

**stksize equ 100h ; max size of system stack**

**dseg**

**public \_sys\_imp**

**intstack db stksize dup (?);interrupt stack**

**topstack label byte**

**\_sys\_imp dd far ptr intmap**

**endds**

**pseg**

**public pcxflag, sssave, spsave**

**pcxflag dw 1 ; zero when rescheduling disabled**

**spsave dw ? ; saved stack pointer register**

**sssave dw ? ; saved stack segment register**

**; intmap -- interrupt dispatch table**

**intmap label byte**

**rept tblsize**

**db ? ; ivec - interrupt vector number**

**call intcom**

**dd ? ;oldisr-old isr from bios (seg:off)**

**dw -1 ; newisr - new isr code address**

**dw ? ; mdevno - minor device number**

**dw ? ; iflag - interrupt flag**

**endm**

**ASSUME DS:NOTHING**

**; intcom -- common interrupt dispatcher**

**; This procedure is interupt handling code that is common to all interupt service routines.**

**intcom proc near**

**push bp**

**mov bp,sp**

**push ax ; push registers**

**push bx**

**mov bx,[bp+2] ; get pointer to intmap data**

**mov ax,cs:[bx+8] ; get interrupt flag**

**cmp ax,0 ; zero?**

**je short nobios;yes,skip the call to the BIOS**

**pushf ; push flags to simulate interrupt**

**call cs:dword ptr[bx] ; call BIOS ISR**

**cli ; be \*sure\* interrupts are back off**

**nobios:**

**push cx ; save rest of registers**

**push dx**

**push si**

**push di**

**push ds**

**push es**

**mov cs:sssave,ss ; save stack environment**

**mov cs:spsave,sp**

**mov cx,cs ; get code segment**

**; bp+6 points to code segment where interrupt occurred**

**cmp cx,[bp+6] ; check if we own interrupt**

**jne short newstack**

**; time to do our ISR, since the stack and data segments are OK**

**push cs:word ptr[bx+6] ; pass minor dev. no.**

**call cs:word ptr[bx+4]; call C ISR(saves si, di)**

**add sp,2 ;deallocate parameter(C convention)**

**jmp short popregs**

**newstack:**

**; now set up temporary stack in DGROUP and do our ISR**

**mov ax,DGROUP**

**mov ds,ax ; set ds to DGROUP**

**ASSUME DS:DGROUP**

**mov ss,ax ;and set up temporary stack in DGROUP**

**mov sp,offset topstack**

**xor ax,ax ;clear pcxflag to prevent reschedules**

**xchg ax,cs:pcxflag**

**push ax ; save for later**

**push cs:word ptr[bx+6] ; pass minor dev. no.**

**call cs:word ptr[bx+4] ; call our routine (saves si, di)**

**add sp,2 ;deallocate parameter(C convention)**

**pop cs:pcxflag ; restore pcxflag**

**mov ss,cs:sssave ; restore old stack**

**mov sp,cs:spsave**

**ASSUME DS:NOTHING**

**popregs:**

**pop es ; restore all registers**

**pop ds**

**pop di**

**pop si**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**pop bp**

**add sp,2 ; remove pointer to intmap area**

**iret**

**intcom endp**

**endps**

**end**

## אתחול וקטור הפסיקות – Revectoring

**אתחול רשומות הפסיקה ב-sys\_imp נעשה בנפרד לכל פסיקה בה XINU משתמש. האתחול כולל שינוי הכתובת בוקטור הפסיקות והגדרת רשומת הפסיקה ב-sys\_imp. אתחול זה נעשה ע”י הפונקציה mapinit.**

**עיקר האתחול:**

**שמירת כתובת ה- isr הקודם ברשומת הפסיקה.**

**שינוי וקטור הפסיקות שיצביע על פקודת call intcom ברשומת הפסיקות.**

**אתחול שדות נוספים ובהם iflag.**

**בסיום פעולת XINU יש לשחזר את הכתובות של הפסיקות בוקטור הפסיקות כדי לאפשר ל-MSDOS לחזור ולפעול. פעולה זו נעשית ע”י הפונקציה maprestore השולפת את הכתובות הקודמות מתוך רשומות הפסיקה ושותלת אותן בוקטור הפסיקות.**

**map.c**

**mapinit** **-- fill in an intmap table entry**

**int mapinit(vec,newisr,mdevno)**

**int vec; /\* interrupt vector no. \*/**

**int (\*newisr)();/\* addr. of new service routine \*/**

**int mdevno; /\* minor device number \*/**

**{**

**int i; /\* intmap entry \*/**

**word far \*addr; /\* far address pointer \*/**

**struct intmap far \*imp;/\*pointers to intmap\*/**

**int flag;/\* upper byte of vector \*/**

**i = nmaps;**

**if ( i >= NMAPS )**

**return(SYSERR);**

**nmaps++;**

**imp=&sys\_imp[i];/\*point to our intmap entry\*/**

**flag = (vec>>8) & 0xff;/\*pick up flag byte \*/**

**vec=vec & 0xff;/\*only low-order byte counts\*/**

**/\***

**#define FP\_SEG(fp) (\*((unsigned \*)&(fp) + 1))**

**#define FP\_OFF(fp) (\*((unsigned \*)&(fp)))**

**\*/**

**FP\_SEG(addr) = 0; /\*interrupts are on page 0\*/**

**FP\_OFF(addr) = vec \* 4;/\* offset of this interrupt no.\*/**

**/\* set up the input intmap entry \*/**

**imp->iflag = flag; /\* deposit flag byte in iflag\*/**

**imp->oldisr\_off = \*addr;/\* offset \*/**

**imp->oldisr\_seg = \*(addr + 1);/\* segment\*/**

**/\* the following is highly machine dependent \*/**

**\*addr = FP\_OFF(imp)+1;/\* point to call instruction \*/**

**\*(addr+1) = FP\_SEG(imp);/\*this code segment\*/**

**imp->newisr = newisr; /\*our input handler\*/**

**imp->mdevno = mdevno; /\*minor device no.\*/**

**imp->ivec = (char) vec;/\*interrupt vector\*/**

**return(OK);**

**}**

**// maprestore -- restore all old interrupt vectors from the intmap**

**int maprestore()**

**{**

**int i; /\* intmap entry number \*/**

**word far \*addr; /\* far address pointer \*/**

**struct intmap far \*imp;/\*pointers to intmap\*/**

**if ( nmaps > NMAPS )**

**nmaps = NMAPS;/\* just to be sure \*/**

**for ( i=0; i<nmaps; i++) {**

**imp = &sys\_imp[i]; /\* point to this intmap entry \*/**

**if ( (int)(imp->newisr) == -1 )**

**continue;/\* if unused entry \*/**

**FP\_SEG(addr) = 0; /\* interrupts are on page 0 \*/**

**FP\_OFF(addr) = imp->ivec \* 4; /\* offset to the vector \*/**

**\*addr = imp->oldisr\_off;/\* offset \*/**

**\*(addr+1) = imp->oldisr\_seg;/\*segment \*/**

**}**

**}**