- מהו קוד מכונה? ה-ISA של המעבד.
  - אחד הקודים + ה-ISA.
  - 3 יח' - 100% מהציון הסופי.
  - 10% מטלות.

מבנה המעבד - 8086 של אינטל.

מעבדים משנה ה-ISA אך לא ה-OS:

8008 - Alter, אחד הראשונים. מעבד 8bit (=8 סיביות).
8080, 8bit ← 8086, 16bit ← 20 סיביות ← מיקרו-מחשב ראשון.

RISC - Reduced Instructor Set Computer - מעבד עם מעט פקודות, אבל הן מהירות.

CISC - " " " Complex " - מעבד עם הרבה פקודות יכולות ומסובכות.

- כל סוג יצרן מוסיף פקודות לשפת המכונה שלו ומשווק את המעבד שלו.

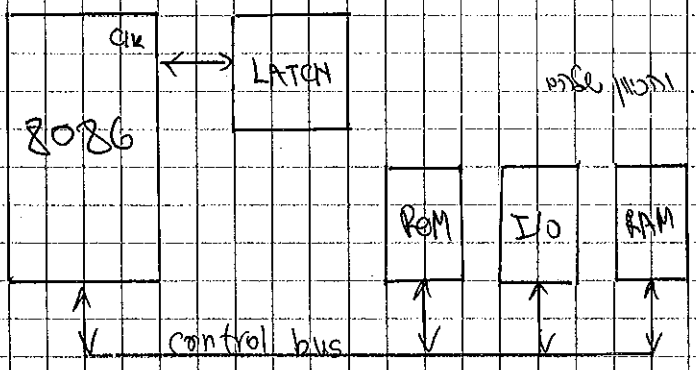
8086
80286
80386
80486 ← 32bit.
Pentium = 80586 ← 64 bit.
Multicore - כשיש כמה ליבות (cores) על שבב אחד. (אם ניתן היה לשים את כולם על שבב אחד, היה מהיר יותר. אבל יש בעיות של קירור וחשמל). ליבה היא יחידת עיבוד שלמה עם זיכרון מטמון משלה וכו'. → שבב שמכיל יותר מליבה אחת

שם המעבד לא נותן מידע מספיק לגבי ביצועיו, אלא רק מספר בערך.

- **ארכיטקטורת פון נוימן**: המעבד מחובר לזיכרון באמצעות 3 סוגי bus:
  - bus כתובת.
  - bus פקודה/נתון.
  - bus בקרה.

הזיכרון - מאוחסנים בתוכו הנתונים והפקודות של התוכנית (שפת מכונה).
מאופיין ביכולת לגשת לכל תא בזיכרון באותה מהירות ללא תלות במיקום התא בזיכרון. ← RAM (קיצור של Random Access Memory).

כל תא בזיכרון מזוהה ע"י כתובת ייחודית שלו. הכתובת היא מספר טבעי החל מ-0.
כל תא בזיכרון מכיל מספר בינארי בגודל קבוע. גודל תא בזיכרון נקרא גודל המילה.
גודל המילה - הוא גם גודל הנתונים הגדול ביותר שניתן לעבד בפקודה אחת.
בד"כ גודל המילה הוא 8bit - ניתן לעבד גם כאלה (short) integer 1 byte.
כל תא בזיכרון הוא בגודל של byte 1. כלומר, הכתובת הבאה אחרי תא מסוים היא תא שנמצא byte אחד אחריו.
הכתובת של הבית הראשון במילה היא הכתובת של המילה כולה.
המילה הוא מספק 2 שלמים כאשר כתובת הבית הראשון היא הכתובת של המילה.



- ליחה בין הכתובת לנתון אין הבדל שניהם נשלחים באותו אופן.
- כשיש פקודה - תלפף קודם את הפקודה ואחרי שנגמר אותה מביא את הנתונים.

ארכיטקטורה CISC היא היפוכה של ה RISC. (מתוך הנושאים שנלמדים).

מספרים שהם הופכים רבים שיש לעשות את ה RISC או CISC

Mnemonic
Opcode
Operands

Mov AX, 5 ; ← הערה

Mov EAX, 5 → Immediate Addressing.

- את בין המס מקום שלו מאבד הזיכרון.

Addressing Modes : סוגי איך

(סוגי פתרונות לאיך מביאים את הנתונים)

- Register Addressing : "אם הוראה" אומרת את מיקום הנתון
- Immediate Addressing : הנתון מיד ההוראה

- אך מה היתרון שיש לשלישי לשלישי?

Mov AX, BX ← מעתיק את הנתון מ BX ל AX

Mov AX, 5 ← מעתיק את הנתון 5 ל AX

Mov AX, [10h] ← הנתון בכתובת מיצוין הוא מעתיק ל AX.

כאשר הכתובת יכול להיות RAM / ROM.

שיכול להיות שכאשר

- Direct Addressing : כתובת הנתון היא המוחלט.

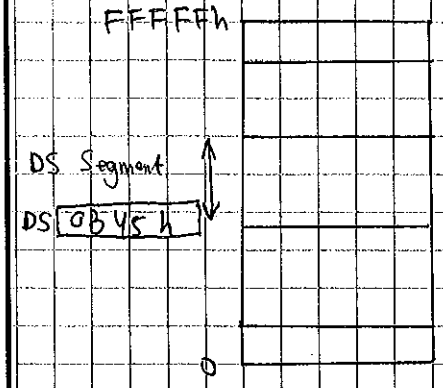
מעתיק כתובת בכתובת :

Mov [10h], AX

הכתובת הנתון היא מאוזן של 16 סיביות ← ניתן לגשת עד 2^{16}.

שיכול המחשבים מחלקת את הזיכרון הפנימי כסגמנטים:

קוד, נתונים, stack, data, code ... CS, DS, SS



Physical Address - הכתובת הפיזית

Mov AX, [13h]

כתובת הסגמנט - הכתובת ההתחלתית של הסגמנט בזיכרון

מיקום המשתנה בתוך הסגמנט - ההפרש בין כתובת המשתנה לכתובת הסגמנט

$$0B45h \times 10h = 0B450h$$

$$0B450h + 13h = 0B463h$$

כתובת המשתנה בזיכרון

27.02.11

שבוע

מרצה: אלכסנדר מנדלר

מתרגל: אלכסנדר מנדלר

מבחן מועד ב' 2.

CISC - ארכיטקטורה שנוצרה לפני שהיו מהדרים מוכנים.

מטרה: לתת הוראות שמאפשרות לכתוב בשפת מכונה בקלות.

- הוראה מקצרת שכן אופרנדים: אופרנד יכול
  - יכול להיות זיכרון
  - יכול להיות ישיר
  - יכול להיות עקיף
- הוראה יכולה לבצע הרבה פעולות.
- אורך הוראה משתנה.
- מספר הגישות לזיכרון משתנה.

RISC - ארכיטקטורה שנוצרה לפני שהיו מהדרים טובים.

מטרה: מהדר יוכל לייצר קוד יעיל.

- הוראות פשוטות שכל אחת עושה דבר אחד בלבד.
- אורך הוראה קבוע.
- מספר הגישות לזיכרון מוגבל.
- אופרנדים:
  - אוגרים
  - מיידי
  - ישיר
  - עקיף
- אין אופרנדים בזיכרון.
- מספר האוגרים גדול מאוד ביחס ל-CISC.
- מספר הפקודות קטן ביחס ל-CISC.

דוגמאות: reg-reg, reg-imm, reg-memory, read-modify-write.

( ) דוגמאות לפקודות שלא יכולות להתקיים ב-RISC ויהיו ב-CISC.

- ADD [15h], 5 ← read modify write. ← מיידי, ישיר, ישיר
- MOV AX, 5 ← 16 סיביות
- MOV EAX, 5 ← 32 סיביות
- Segmentation ← Segments

מטרה: הגדלת הזיכרון הנגיש למעבד.

רעיון: לחלק את הזיכרון ל-Segments שמתחילים בכתובת שמתחלקת ב-16.

כתובת ה-Segment תהיה הכתובת שבה מתחיל ה-Segment בפועל.

- שם של אזור מסוים בזיכרון. Segment
- כל אזור יכול להיות בגודל שונה.

( ) - שמור הגדול ביותר את הספרה הקיצונית.

- כתובת ה-Segment - בת 16 ביט.
- כתובת ההזחה - בת 16 ביט.
- כתובת פיזית - בת 20 ביט.

⇐ כתובת ה-Segment - נקבע בזמן ריצה ע"י מערכת ההפעלה.

כתובת ההזחה - נקבעת ע"י המהדר.

⇐ כתובת ההזחה של הפקודה הבאה לביצוע נשמרת ב-IP - Instruction Pointer.

כתובת הפקודה הנוכחית נשמרת ב-IR - Instruction Register.

Fetch - הבאת הפקודה הנוכחית מהזיכרון.

Decode - פענוח הפקודה הנוכחית.

Execute - ביצוע הפקודה הנוכחית.

* בתוך המעבד יש רשימה של ההוראות הקיימות.

(ב- RISC ו- CISC ה ההוראות הן אותן אבל זמן ההרצה שונה.)

מחזורי הפס מתבצעים בהתאם לצורך המעבד.

ישנם 4 סוגים של מחזורי פס:

קריאה מהזיכרון
- * מחזור פס של הבאת הוראה (קריאת הוראה)
- מחזור פס של קריאת (נתון) - מתבצע בצורה דומה. הזיכרון מוסר את התוכן של פס הנתונים ופס הבקרה מוביל פעולה של מצב קריאה. (הפוינטר יהיה לנתון והרגיסטר יהיה מונה של נתונים.)

* מחזור כתיבה לזיכרון (המעבד מוציא גם את הכתובת אליו יכתוב הנתון, ה הבקרה שיופעל - memory read).

* קריאה מפלט/קלט - בדומה לקריאה מזיכרון. במקום הזיכרון יהיו רכיב של קלט/פלט וקו הבקרה יהיה - I/O read.

* כתיבה מפלט/קלט

( )

ב- 8086 - 20 קווי כתובת
- 16 קווי נתונים
-

לסיכום, מחזור הפס:

מעבד
- הבאת ההוראה מהזיכרון , IR.
-
-
-
-
° וחזרה לפעולה הראשונית

( ) * מעבד 8086 - מחולק לשני יחידות: יחידת ממשק ויחידת ביצוע.

ה יחידות עובדות במקביל.

הערה: אבל - ההוראות ברצף - שונה ⇏ הזמן של הרצת ה הוראה שונה.

מאוחסן בזכר

אוגרי המעבד:

char → byte.

byte → byte.

boolean → byte.

integer → word = 16 bites.

Word → word

Logg → DWord

Pointer → Dword.

8086 - סגמנטים בגודל של 64KB

לפי - AX ←
(ה אוגר הגדול הוא 16 סיביות)

Mov [BX] , AX

+BX
DS
x16

BX: 005ch
DS: 0041h

+ 5ch
410h
0046ch

- הוראות ה- 8086 הם על פני - AX, BX

- ניתן לפנות לאוגרים ב- 8 סיביות - AL, BL, ...

- ניתן לפנות לאוגרים ב- 32 סיביות - EAX, EBX, ...

- 64 סיביות - RAX, ...

- אוגרי מצביע:

דוגמה: DI (מצביע ל- SI)

* כל אוגרי מצביע יכולים לשמש כאוגרי

- הוראות לולאה: ... - mov

- הוראות אריתמטיות: add, sub, ...

- הוראות לוגיות - and, or, xor, ... (פועל על כל הביטים - ALU - Arithmetic Logic Unit)

- הוראות קלט / פלט - in al, 61h ... out 61h, al

- הוראות מחסנית - push, pop. Stack Segment - משמש לשמירת נתונים זמניים

- הוראות מחזור - LOOP - משמש לביצוע פעולות

- הוראות מעבד נוסף - מאפשר לשמור נתונים (זה לא חשוב)

(הוראות הספרייה של זה ניתן לחשוב שהוא לא חשוב בכלל)

- הוראות נוספות מהמעבד הפלואטי וכו': FADD - אריתמטי ל- floating point (זה לא חשוב)

XMM - מאפשר לשמור נתונים על אוגרים של מעבד גרפי

(כמעט ואף אחד לא משתמש בזה, וזה לא חשוב בכלל.
זה לא מעניין אותנו, וזה מסובך מדי.)

- הוראות לולאה (זה כן):

מבנה הלולאה:

```
mov cx, 10
start:
    ...
    loop start
```

זוהי לולאה שתרוץ 10 פעמים. בכל פעם שהיא מגיעה לסוף הלולאה, היא תחזור להתחלה.

הערה: אם נרצה לשנות את מספר הפעמים שהלולאה תרוץ, נשנה את הערך ב- cx.

הערה: אם נרצה לשנות את מספר הפעמים שהלולאה תרוץ, נשנה את הערך ב- cx.

שיטות מיעון ב- x86

- מיעון אוגר - MOV AX, DX
- מיעון ישיר - MOV [BX], AX ← (הפוינטר הוא הכתובת של הנתון) כתיבה לזכרון, אפשר כמובן להפוך לבצע גם קריאה מהזכרון.
- מיעון מיידי - MOV AX, 0A00h
- מיעון עקיף באמצעות אוגר - האוגר מכיל את הכתובת. קוראים אוגר הבסיס Index או Base
  SI, DI, BX   MOV AX, [SI]
- מיעון יחסי באמצעות אוגר - MOV al, [bx+si+30] ← כמו מיעון בו-מיידי (CISC).
  offset

* אי אפשר לעשות פעולה של מזיכרון לזיכרון (חוץ מפעולת מחרוזת) ← צריך 2 פעולות פס.
מחזור פס של כתיבה ומחזור פס של קריאה

Vector0 db 4,2,1 → מערך של 3 ערכים. כל ערך בגודל בייט.

מגדיר משתנה בגודל byte (define byte).

X db 2 ← הגדרת משתנה בשם x בגודל 2.

הקומפיילר מקצה מקום בשלב הקומפילציה - נוצרת ה-image של התוכנית עם כתובות יחסית. לכן בשלב הקומפילציה הקומפיילר מקצה כתובת יחסית לשם הזה. (בכל מקום שרשום x - הוא מחליף את הכתובת היחסית)

כשלא נותנים ערך התחלתי: בזיכרון יש ערכים הקודמים.

X dw ? → word

X dd ? → double word.

Hello db 'Shalom' → הקומפיילר מקצה אזור של מס' הבתים.

מבנה תוכנית:

תוכנית מורכבת מהוראות שנועדו למעבד והוראות שנועדו לקומפיילר.

הוראה לקומפיילר - עוזרת לדעת איך לבנות את ה- image ואיפה מסתיימת הקומפילציה (end)

סיום התוכנית ב- CPU: החזרת השליטה למערכת ההפעלה:

```
mov ah, 4ch
int 21h
```

פסיקה בתוכנה מאפשר יציאה לפונ' מע' הפעלה - DOS.

```
. model small
. stack 100h
. data
. code
* * * * * *
mov ax, @data
mov ds, ax
end.
```

- הנחיה לקומפיילר לבנות image בגודל המתאים.
- גודל סגמנט המחסנית הוא 100.
- סגמנט הנתונים בגודל 0. כי לא * * * * (הגדרנו משתנים)

מבנה תוכנית:

הוראות שמשפיעות על הזיכרון שיוקצה:

CALL SUM — שם הפרוצדורה

SUM
RET

כשבצעת מעבר בין תוכניות אנחנו מכניסים את הכתובת של SUM ל IP.
שמים לו הוא שומר את התוכן של ה- IP (הוא מצביע לפקודה שאחרי הפרוצדורה).
הפעולה שמשפיעה על ה- IP נקראת JMP.
לפעמים שומרים אותו ב- IMP
כלומר כל שפה שמשתמשת בו שולח לתוכנית שמה שהיא צריכה.
אחרי שהוא לקח את התוכן הוא שולח את הכתובת.

* הפרדה בין סגמנטים:

ה- code וה- data מוגדרים על פחות = (אחד לשני) על סגמנט אחד.

• מבנה: .stack , .data , .code .

• ניתן להגדיר משתנים באופנים: dp ← 2 בתים
mp ← 4 בתים
bp ← 8 בתים

הדבר הוביל לזה שבמקום ה- segment data (כפי שהיינו רגילים עד עכשיו) הגדירו (לשם את השם שלו).

• לולאות: כדי לגרום לשינוי שיתקיים תוכנית.

הוראה שיכול את הקוד של הלולאה ופעם אחת היא בודקת אם ה
(בדיקה) ה משפיע שם את הלב הלולאה ⇐ אין אפשר לבדוק בתוך ⇐ זו הסיבה שאחרי ה
הוראה: jz label / jump if zero = 0

הרצת תוכנית אסמבלי:

File.exe
↑
Loader ⇐ Linker ⇐ File.obj ⇐ Assembler ⇐ File.asm ⇐ Editor
(notepad)

מכין את הזיכרון	מיקום קוד, נתונים, מחסנית	שם מקומפל	שגיאה משפה אסמבלי
מעתיק את הסגמנטים	header	שפה מכונה	שפה אסמבלי
לזיכרון	קובץ גוף שם כל		
	הוראות שאפשר להריץ.		
	לקבל כן אחרי ההפעלה.		
	הוראות שלו.		

* הוראות הדפסה הן לפי מקלדת. ב- x64 - אופציה שמופיעה אז אצל הסטודנט אחד ניתוק. אחרת.

- לינק קובץ רשימה של סמלים הנמצא יחד עם הקובץ שניתן לשימוש אחר.

add bx, 160 ⇐ מקום. אי"ב. נניח "160" יופיע בשם הסמלים ערכו 0A00.

MOV [bx], al ⇐ (שגוי) MOV [bx], ax :כך MOV

* הצהרה על קבועים - נשים לשים בקע עליון בתחילת. (יכול שם ה- data segment)

אבל רצוי לשים אותו לפחות בתחילת כדי שנדע היכן בערך לשים אותו

* MOV DX, RATE ⇐ מקום. שוי.

הקומפיילר מחליף את שם המשתנה בכתובת של המשתנה ב- data.

0000 8B 16 0002r (rate של כתובת היחסית)

* CX - משתנה מהצורה המיוחדת לפקודת loop.

כתוצאה הסבה של הבדל - 128 בתים ⇐ סוג הקפיצה היא מסוג short. (קרוב).

⇐ הקומפיילר מחשב את ההפרש מהפקודה אחת את ה- offset היחסי.

הסבה אוטומטית - אם שגוי, (נשים לב).

מיקרו - קוד משותף בשפת C/C++:

כי- C/C++ הוראה יכולה לגרום לאותה פעולה אחרת. ולכן לשים את - Prefix ⇐ קוד נוסף על הקיים.

לדוגמה- Rep Prefix - הוסף לפעולה שאחריו לחזור מספר פעמים.

מבנה כפוי של קוד מכונה:

OP code — קוד הפעולה — 1-2 byte

mod R/M — מידע נוסף — 1 byte.

SIB — מידע נוסף

Displacement — ↓

Immediate — הערכים

mov AX, [BX+SI+0Fh]

displacement

- כאשר מקמפלים לשפה - יש צורך לבנות את הקוד ולקשר בין
כל ה־ libraries הן הספרות הפנימיות - האחרונות בשפת־מכונה
(ב־0) מוסיפ מספרים הוראות שמתחילות את הקוד
• mov AL - שם פרמטר לשפת כתיבה אלא לא כתוב בקוד כי־כאן.
הבוב אותה הפרמטר ב־ Segment. הפרמטר ; קפיצה מרחקית
jmp far - קפיצה אחר ב־ Segment. הפרמטר ; קפיצה מרחקית
out of range)
short jump - מוגבל ל־ 127 ⇐ 128 - (אפשר לכתוב שתוכן
קפיצה מוחלטת מאפשרת לבצע קפיצה שניה מאחרי אחת אם
ב־ JE. כלל את מהלך הקפיצה של ה־ JE אם נמצא
3 סוגי jump שונים מוחלטת: jump far, jump near, short jump
3 בתים בתים בתים ← כתוב אחד.
הקומפיילר מחשב את ההפרש של ה־ offset.
• הוראות הקפיצה מוסיפות לפחות - if, while, jmp.
ה מאפשר לשנות קפיצה בתוך הקוד מחדש.
2. שימוש בשפת־מכונה מקלה על שינוי שפת־מכונה.
1. מהירות + תלות ההוראות מוצפנות (סודיות).
שיפור הביצועים:

Mov AL, BL :דוגמה

Opcode	Mode	Operand 1	Operand 2

שפת־מכונה מאפשרת לשפה עילית כתיבה נוחה.
הקומפיילר מתרגם את ההוראות לשפה עילית לשפת־מכונה
הנקראת "common inter... language".
אין לי מושג מהכל.
רמז: לכל הוראה ב־ hex יש אותה כמות בתים ויותר מזה "אותה הוראה".
• מהנדס ביולוגיה - רוצה לשנות מהנדס מחשבים ביטוח
הקומפיילר מתרגם את השפה לשפת־המכונה שלו.
• machine code ב־ שפת־מכונה.

הפקודה - Push (או Push - הכנסה של ערך למחסנית.
ב- 8086 זה יכול להיות גודל 8 או 16 סיביות. (תלוי בגודל של הפרמטר).
ניתן גם לבצע זאת עבור כתובת הזיכרון במחסנית.
(הכתובת = אותה כתובת שבה נמצא הערך).
• שני הפקודות הללו Push / Pop יבוצעו - רק אחרי שהיחידה מוכנה לכך.
• Push, Pop - מתייחסים לפעולה של פוש.

הפקודה Push cx - הפעולה הזו מבצעת שתי פעולות על ה- stack.
כלומר: הוספת ערך של cx למחסנית ועדכון ה- sp.
ה- Push הוא פעולה שבה אנחנו מוסיפים ערך למחסנית.
כלומר: מוסיפים את הערך שנמצא ב- cx למחסנית.

Pop cx
• מוציא מהמחסנית ומעדכן את ה- cx בערך שהוא.
(מוציא את הערך הנמצא בראש המחסנית ל- cx).
Push cx. ⇐ sp←sp-2, [sp-2]←ch, [sp-1]←cl

הפקודה: Push - הכנסת למחסנית. } הפעולה תמיד תהיה אותה כמות.
Pop - הסרה מהמחסנית.

מחסנית: מבנה נתונים שבו האיבר שנכנס אחרון יצא ראשון.
LIFO. (Last in, first out).
המחסנית היא כמעט מבנה שבו האיבר שנכנס לאחרונה
יצא ראשון מהמחסנית - קרוי stack. יכול להיות
כאשר מתבצעת הקריאה לפונקציה מתבצע שמירת המצב הנוכחי:
(call, ret, push, pop, int

מחסנית:
מה) תוכנה? ה פונקציה: JO או JS.
sign flag overflow flag
(המחשב יודע לזהות בדיוק איזה מהדגלים).
את ההבדל תלוי רק אם נעשה שימוש בדגל. JE - ZF = 1
אחרת - הוא בודק אם אכן הוכנס (cx)=0 הדגל הזה מופעל ⇐ ZF = 1
ה- overflow אומרת כמה סיבות הזמן שהוא תהיה מוגדר.
כי לפעמים הסיבה היא מה אומרת ה- Push כשיש - overflow.

(ההבדל בין call-near ל- call-far נלמד בהמשך)

call = פקודה שדוחפת את ה- IP למחסנית ⇐ call-near

Push IP

הלוך שלב מחסנית בפקודת ה- return.

ret = Pop IP

• אין call לפני הגעה אבל אפשר לחזור לשם אז מגיע לסוף.

• ניתן גם לשנות מה יחזור בסוף. (ret)

(* כדאי לכתוב את ההערות: סימון מחשבתי ובסופו של יום זה לעזר, אבל מעדיפים שלא

סיכום:

סיכום ביניים (לא כשאתה מבין סיכום):

- סיכום מחלקה (פלט - int 21h)
- סיכום מחלקה (קלט - int 21h)

• קלט int 21h - סיכום מחלקה.

" System. call

(שאר בלי לב...) ← PUSHF,

CALL far [4*21h] → Push IP
so Push IP←[4*21h];CS

• דוגמה נוספת -

int 9 - הפסק חומרה.

זה השלב שמגיע ההפסק.

מה שלפני זה הוא הפסק שלנו.

שמירה 0 של הזיכרון שמלא ?פסיקות ; IVT (= Interrupt Vector Table)

מצד אפוך: IRet לפחות כמו pop לפחות).

- קיים וקטור פסיקה יחיד.
- פסיקה חוסמת פסיקה.
- האחריות לטיפול בפסיקה.
- מחזיר ערכים ל פסיקה- IRET

אופן פעולה של פסיקה:

* הערה - ניתן לשנות את הפסיקה בקוד- מה שלא ניתן לשנות זה את הוקטור.

פסיקה תוכנה ('int', 'שם') - אוסף של פקודות של שפת ההרכבה שנקרא בשם הפסיקה (מספר בין 0-255).

פסיקה חומרה חיצונית מסוימת שמחוברת לשבב ה- MPU (בשביל פסיקה 0=).

פסיקות חומרה - פסיקה שמגיעה ע"י חומרה חיצונית.

שהיו לנו לפני

* כשיש פסיקה חומרה היא לוקחת את שלב הביצוע, CS', IP מכניסה (PIC) שבה פסיקה- מגיעה היא ל- MPU.
אבל בפסיקה זו,
* ע"י הקוד מקבלים את הנתונים שיש ל- MPU.

FL - נעלם מ פסיקה פנימית.

ולקוד הפסיקה.
לו יש- data
רכיב I/O שמחבר בין מקלדת לבין ה- MPU. הוא יכול
ע"י הקוד חומרה ה ISR שלו.

יש לנו שליטה כי הפסיקה חומרה. בשביל- שלפני שהוא מבצע את הפסיקה הבאה הוא "בודק" לפעמים כי חומרה מחוברת הוא אי-סינכרוני. הוא לא "בודק" כל פעם (כל פקודה) האחד נכנס ל I/O ואז הוא מבצע את הפסיקה הבאה.)

• **הפסיקה החומרה**- אירוע של חומרה חיצונית שגורם למעבד לפנות לשגרת פסיקה ISR לטפל בה.

פסיקה חומרה מאפשר לטפל מהר יותר בהתקני ה I/O.

פסיקה = מקום שמאחר ויש בידנו התכנית לעבור ל"מבנה" שנקרא פסיקה". ISR.

אלו הוא מצב שבו שגרת פסיקה יכול מחליפה ו (jump far).

$$[4 \times n] \text{ JMP} \quad (\text{jump far}$$

$$n \text{ הוא } = u$$

$$u \quad \text{INT} \equiv$$

Push Flags

Push cs:IP

פסיקה תוכנה ⇐ מופיע בקוד ב- int' מספר שלה.

אין הבדל: פסיקה חומרה ⇐ מאחז תיבוע שיגיע מבחוץ מתבצע.

* Into

* INTO - הפסקה מסוג.

סוגי הפסקות - INT

- מספר הפסקה (סוג) מ-0 עד 255.
- מספר הפסקה - INN - הפסקה רגילה; כאשר אין שום פין INTR. ⇐ מאפשר שלב-שלב (single-step).
- מוגדר פנימי.
- מוגדר חיצוני.
- int - הפסקה תוכנה.
- תוכנה מכוונת.

NMI = non maskable interrupt - פסיקה חיצונית. הפסקה שאי אפשר לבטל אותה.
Trap - הפסקה שנוצרת כאשר מתגלה שגיאה.

- מפעיל את הקו החיצוני INTR.

הפסקה לא מתבצעת:

1. MOVS ⟵
2. MOVSB ⟵ — הפסקה לאחר כל פעולה.

Source Index Destination Index.

DS:[SI] ES:[DI]

הפסקה מתבצעת כאשר:

* הפסקה מסוג 3 או 4.
* מגיעים לסוף הסגמנט ב-FFFF.

MOVSW ⟵ ⟵ מגיעים לסוף הסגמנט ב-0.

הפסקה לא מתבצעת כאשר DF. או רוצים לגרום לכתיבה מאחור קדימה. DF מאופס את

* הקידום נעשה לפי גודל הפעולה.

⇐ מאפשר לשמור הפעולה על הנתונים בזיכרון.

mov 2 מספר 2 ל-CX.
inc 2

הפסקה זו אלה 2-4 הפעולות.

3. CMPS - מאפשר להשוות בין מחרוזות.
4. SCAN - מאפשר לבדוק האם מחרוזת מכילה.
5. STOS - לאחסן בזיכרון מחרוזת.
6. STOS, LODS - לקרוא בזיכרון מחרוזת.

REP MOVSW ⟵ prefix כל הפעולות של לפניה

מצב הפעולה ב- Real-Mode:

IF - ביט שמאפשר פסיקות. כלומר האם ניתן לבקש את הפסיקות.

TF - ביט ה- Trap שמאפשר לנו לעשות את מה שנקרא step-by-step
(משפיע לנו ב- debuger).

ISI - כל בקשה את המעבד שמרים את תוכן הכתובת והקיצור
בתוך שטח ע"י הקצאת זיכרון אחרי ה- stack - שלנו.
סיום- IRET וחזרה לשחזור הכתובת (הוא הכתיר) של ה- stack - שלנו
המצב האחרון שהיה לנו הוא ה- step by step.
כל פעם שיש פסיקה ISI ע"י מקלדת למשל ← מפעיל את מנגנון הפסיקה
ה- DOS פועל ב- Real-Mode (אחד ה- modes שהיו לפני ה- protected mode).
protected mode - ה- real-mode של windows ו- DOS. ההבדל בין ה- protected mode
ל- real-mode: ב- DOS אין מישהו אחד שמנהל הכל.
ב- protected mode אפשר לנהל multi-tasking. שזה מנגנון ה- multi-tasking
ב- real-mode ע"י תוכנה כלשהי בזכרון תוכל לגשת את מה שרוצה אליו.
תלוי ההיגיון של המחשב. יש מחשבים שהם multi-tasking.
• המצב של שלנו מקורי (ברירת המחדל) הוא כאשר ה- real-mode

protected-mode

הגדרה: מצב שבו מקורי מקבל את כל הפסיקות. אם מקורי
רוצה לבצע פעולה שאינה מותרת לו, מערכת ההפעלה תבצע אותה עבורו.
ה- protected mode יש יתרונות מאוד גדולים.

hidden reg.

IDTR → IDT

privileged user ← administrator

superuser

• ב- Real-mode - מקורי הפסיקה שמגיע אל-0.

- ישנה עלה מקורי פסיקה

int 21 לפי תוכן האוגר 8 ביטים

כשהמקורי חיבורי שהמקלדת מכילה הוא יכול לבקש פסיקה כך שהוא
- כל מחשב מגיב לפסיקה אחרת (יכול לקרוא כך את זה).
- א-סינכרוני: (יכול לקרוא כך את זה).
יש מצבים שבהם נרצה למנוע פסיקה.
המחשב פסיקה - מערכת שלי פסיקה Halt (= הקשיב ומלא) ← פסיקה מאוחר

מ- protected mode; מצבי IDT, ומצבי כתובת תמיד 8 בתים (ולכן 4 הביט).

במעבדי 386, הוראת IRET מוסיף גם עדכון הדגלים כמו שמבצע הוראת IRET (הוראות 32 סיביות).

ה- IP מכיל 32 סיביות. IP(low) - החלק הנמוך של ה- IP בן 16 סיביות.

IP(high) - החלק הגבוה (16 סיביות).

הדרישות מופיעות בחלק זה; . . . (IDT ב

- IRETD; ההבדל העיקרי הוא שהוא מוחזר כי.

במעבדים שלפני ה- 386 ואילך ה- Stack היה מלא את

EIP ואת ה- EFLAGS (שניהם 32 סיביות).

5 int ⇐ פקודת Bound - אפשר לבדוק אם נתון מסוים (אבל במחסנית ערכים מסוימים, את הגבול שלו (אבל מחסנית ⇐ הפקודה מפעילה interrupt)
(בתנאי כי היא מופעלת ב real-mode)

( ) הוראות CBW/CWD:

- מיקום - מקור: אחד נתון מקור הוראות

- מיקום - מקור: מיקום למקור כתובת. למשל הוראה מקור CBW-CWD מפורש.

שהרי מקור שלם שהוא מעתיק. כיצד יודע שלקרוא מהזיכרון ולא מהמקור? ולכן הוראות אלו אין להן אופרנדים. מוכרח להיות אופרנד שהוא מקור (מהזיכרון או מרגיסטר) ואופרנד שהוא יעד (לרגיסטר או לזיכרון).

ב- 8088 - 2 פקודות CBW-CWD: IN AX, Port_Address (0-255)

Out Port_Address, AX
(0-255)

• אין משנה ה- Port שנבחר אי 255 ניתן לגשת

באמצעות DX.

( ) ⇐ חיסרון - אין בפקודה של כיב אחד כמה (בשונה מהזיכרון-שפיות) bit-Addressing.

הקוד הנכתב ב- protected mode שונה מה- real-mode

בגלל הכתובת ראשית שונה בגלל הכתובת ראשית
מתחילה מ- 0. שונה מתחילה מ- 0.

ב- protected mode: IDTR מצביע אל מקום הטבלה וגודל שלה וכך
מתבצע מיפוי = טבלה-שניה. (לוגית שניה - קצה הגוף)

בגלל הקוד הנכתב ב- Pentium:

יש type בתוך ה- descriptor.

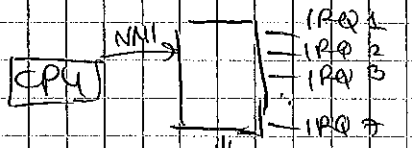
הוא type - כמו ב- descriptor.

הוא type יש סוגים שונים protected mode ב-

למשל שכתוב תמיד ב- segment stack.

הקוד יודע page fault - מה (virtual memory) (pages) תוכנה

(בסוף כתוב שזה קריטי לסוף.



הדרכה לפרויקט הבית:

timer ← מחובר ל- speaker.

צריך להבין בדיוק ל- מה הוא מתכוון שיהיה שלוש צלילים שונים:

אם מספיק לו טון פשוט של תדר - רואה שהוא מדבר בהמשך.

לכל אוקטבה יש תדירות שונה. ההפרש ביניהם הוא חצי חזקה של 2 וכך
(מעבר 8 יש לחשוב על שלוש הוא רוצה D : 587.33 .

ל- timer 3 קיבוצים (counter 0, counter 1, counter 2) מהם 2 counter

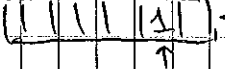
מחובר בדרך שהוא And ל- speaker?

כשהוא out 2 בדיוק לתוכו נכנסים בתוכו השעון.

הפתרון: Port 0x61, bit #1 = 1 - out-מחובר לשלוש שלוש בקר/בק.

out 61h, AL

AL = 02h



out, and, or, in. בקר-

ב- timer #0 מחובר out של 2 ב- מחובר ל- CLK 2 = 1193182 Hz

(לפעמים את מה שעניין ב- out הוא הפעם הראשונה שהוא עולה (בגלל שהוא מתחיל מ- 0).

LSB - בתוך MSB.

Port 64 = Port -> של הגדרה.

הכי פשוט שנעקוף את זה זה scan code.

כל לחיצת מקש זה בעצם לחיצה - שחרור שזה בעצם שני קודים של הלחצן (מיוצרים כאן)

mov es:[bx], ax ← הקוד של הלחצן שנלחץ.

:

lea ax, New_KeyB ← כתובת של הפונקציה שלנו

mov es:[bx], ax

mov es:[bx+2], cs

כדי לדעת שלחצו בסיכה?

יש שלם כזה בתבנית הלחצן

יש על הקו הפסוק 9 בתמונה 4000h

הקוד הישן של ה int 9 (כי הקוד שלנו יקרא לו בסוף)

קוד הפונקציה - ip

קוד הפונקציה - cs

כשעדיין אפשר רוצים שיהיה כאשר מחובר

שים לב: ICW1 מגדיר הכול:

1. מגדיר האם הבקר ICW1 → מגדיר לבקר שאחריו יגיעו ICW2 ו-ICW3 (אם יש).
   אם הבקר האחרון יחיד ← לא יגיעו.

שאפשר לחבר עד 8 בקרי פסיקות שיפעלו ביחד, אחד master והשני slave ← ע"י איזה מצוי

מגדיר בפועל יותר רק אם הוא master או slave.

* ICW1 :

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	X	X	X	1	LTIM	X	SNGL	IC4

← מספר הבקר.

- האם יהיה ICW4
- האם ההפסקה תהיה level triggered או edge triggered.

(
מדגיש שהפסקה מתרחשת
כאשר הפסקה היא בלתי מקשרת
לעומת edge triggered
)

level triggered - הפסיקה מתבצעת כל עוד מתח חוזר מ-"0" ל-"1".

edge triggered - הפסיקה יכולה להתרחש רק כאשר יש שינוי מ-"0" ל-"1".
(כלומר, כי כאשר יש שינוי
הפסיקה יכולה להתרחש רק
פעם אחת.)

* ICW4 - קובע את מקום וקיבוץ הפסיקות. מגדיר לבקר האם הוא מחובר למעבד או לא.

8 סיביות נמוכים של הכתובת, 4 סיביות גבוהות - real mode

* סיבית אם ב- protected mode

(
הערה - כי בשלב הזה אנחנו עדיין לא מגדירים את הפסיקות.
)

לאחר שהגדרנו את ICW1 ו-ICW2 נוכל לגשת אל OCW1 ו-OCW2 בכתובות (D_0, D_1, D_2).

⇐ זה נותן האפשרות ל- hardware.

דוגמה:

```
mov al, 00010000b ; שגיאה את
out 20h, al       ; ICW1
```

edge trigger ←

```
mov al, C8h = 11001000b ; שגיאה את ICW2
out 21h, al = 11100000b
```

הכתובת של OCW1: הכתובת של הפורט 21h (בד"כ)

מיועד לפסיקות:

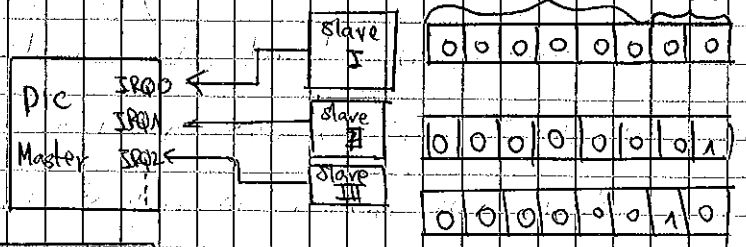
IRQ0 : 12h , 320h , 322h ...

IRQ1 : 13h , 324h , 326h

:

ICW3 = נקבע רק במידה ומקים מספר מעבדים

0- Master , מקים מערכת אחת בלבד מסוג זה. Slave



ICW3: 0 0 0 0 0 1 1 1

ICW4 = מאפשר לפרוץ מצבים חדשים ולא מוכרים.

Special Fully Nested Mode

None Special Fully :

Auto EOI - מצב שבו בקר הפסיקה (המעבד) עצמו מבצע את פעולת ה EOI (בסוף שגרת ה ISR)

- מאפשר שיוך כתובת לפסיקה:

```
Inter_vect_table  segment  at 0
Org   60*4  →   מקום לשגרת פסיקה 60
Vect_int          label    dword  →  מצביע 60 יכיל כתובת שגרת vect_int
Intr_vect_table   ends
```

OCW = מילת פיקוד המאפשרת בקרה על הפסיקה. 2

OCW1 = מאפשר למנוע מן הפסיקה מסוים. (מאפשר הפעלה חלקית של פסיקה

על ידי מיסוך הרצוי, שלא ירצה לעבוד)

OCW2 = מאפשר EOI (End of interrupt.

הפעל מאפס את סביבת ה ISR שהיה פעיל ומחזיר שליטה.

שים לב לאיזו כתובת הפסיקה הפעילה הסתיימה.

EOI Specific = סיום הפסיקה ע"י שליחת פקודת EOI עם מספר הרמה שבה הפסיקה טופלה.

EOI Non Specific = סיום הפסיקה ע"י שליחת פקודת EOI בלי לציין מספר רמה. (לוקח את הרמה הגבוהה ביותר)

EMCO = שליחת פקודה שתאפס את הביט המתאים ב-In-Service Register.

מאפס לפחות את הביט המתאים.

אחרי IRR + ISR ה- Input Line הרם. (אחרי הפסיקה)

מתחיל את מחזור הפסיקה!

- Interrupt from Hardware: פסיקה מהחומרה

כשיש אות מהמקלדת, הבקר מעביר את הפסיקה לרמה הבאה בתור.

לפעמים ⇐ יותר מפסיקה אחת לפני שהספקנו לטפל.

כשיש פסיקה, המעבד מפסיק את מה שהוא עושה ועובר לטפל בפסיקה.

- Trap: פסיקה שנוצרת מהתוכנה. היא נוצרת כתוצאה ממשהו שקרה בתוכנית.

int 21h, mov ah, [x2] ← מעבר לשורה הבאה בקוד

Trap: פסיקה שנוצרת מהתוכנה. (למשל: חלוקה באפס)

לפעמים ⇐ יותר מפסיקה אחת לפני שהספקנו לטפל.

כשיש פסיקה, המעבד מפסיק את מה שהוא עושה ועובר לטפל בפסיקה.

- Interrupt from Hardware: פסיקה מהחומרה

כשיש פסיקה, המעבד מפסיק את מה שהוא עושה ועובר לטפל בפסיקה.

לפעמים ⇐ יותר מפסיקה אחת לפני שהספקנו לטפל.

- Trap: פסיקה שנוצרת מהתוכנה. היא נוצרת כתוצאה ממשהו שקרה בתוכנית.

מעבר לשורה הבאה בקוד ← int 21h, mov ah, [x2]

Trap: פסיקה שנוצרת מהתוכנה. (למשל: חלוקה באפס)

- Abort: מצב שנוצרה שגיאה כלשהי שלא ניתן לתקן.

⇐ מצב כלשהו.

BSOD, DE - מסמלת לפעמים שגיאת זיכרון.

⇐ יש בעיה בזיכרון ואנחנו לא יכולים לעבוד איתו.

כשיש שגיאת זיכרון, מודיעים ל- Watchdog שיש בעיה ⇐ מאתחל ומתחיל מחדש.

⇐ מצב כלשהו:

Real Mode and Protected Mode

Protected Mode – המצב של המעבד שבו רוב התוכנות רצות כיום. (Real mode הוא מצב שנועד לתאימות לאחור)

למה צריך Protected mode? לאפשרות – הרבה יכולים לרוץ במקביל. ⇐

Multi-Tasking – היכולת ל-

(עוד – שם את המעבד במצב Protected mode)

מהו מנגנון הגנה? המעבד לוקח שליטה – מאפשר

מאפשר למערכת ההפעלה שליטה על המעבד להשתמש ב-multi-tasking, כלומר שיבוצ תהליכים – מתחלף

ביניהם במהירות (הוא יש מספיק שהמשתמש לא יבחין) כאילו כולם רצים במקביל. יש הגנה שתתן לתהליך אחד לדרוס את הזיכרון של תהליך אחר ויהיו מחוברים.

Real - Mode – שלב ההיסטוריה של המעבד. (אלף מיליון של טרנזיסטורים – 1979).

מה מיוחד במצב הזה – 8086. (כ- 29K טרנזיסטורים)

- שלח 4 טרנזיסטורים כ- Real-Mode ב- 8086 (כ- 29K טרנזיסטורים)

מה שונה היום מבחינת יכולת זו? מאפשר להשתמש בתוכנות (מבלבלים) של עד 4G

ואז אפשר לשמש בתוכנות חזקות

כתובת הזיכרון = (לפני הזיכרון – segment מבלבלים)

data segment :
stack :

ב- Real-Mode יש רק 2 סגמנטים: code segment

FS, GS, ← קיימים רק מ- 386.

אם מתמקדים בגודל הכתובת הזו אפשר לדבר על גודל הזיכרון שאפשר לגשת אליו.

$2^{32} = 4_{GB}$. ⇐ ב- Protected – אפשר לדבר על מרחב זיכרון שמאפשר לגשת אליו.

(אפשר לגשת עם מבלבלים EAX, EBX, ...)

* ב- Protected יש עוד מיפוי עבור סגמנט (Code, Data ...), לעומת ב- Real שאין.
* ב- Real – ניתן לגשת רק לסגמנטים (לפי מה שהוגדר ב- small-model).

כתובת הזיכרון = מאפשר לרוץ תוכנה שמאמינה, אבל לפועל בכתובות הפיזיות,

מאפשר גם לנהל לזיכרון (לחלקים segment-data).

⇐ מאפשר הגנה בזיכרון.

* כ- Protected - יש שני סוגי Segment = אזור בזיכרון. Descriptors

כדי שנוכל לגשת לזיכרון, המעבד צריך לדעת מיהו סוג הזיכרון, אורך הזיכרון

והרשאות הגישה לסגמנט.

GDT - Global

LDT - Local

המידע נשמר בטבלה מיוחדת בזיכרון - טבלת Descriptors.

כל תהליך מקבל את הטבלה שלו.

⇐ שני המעבדים יהיו שלוש טבלאות שונות.

⇐ כי ה- Protected-Mode מנהל יותר תהליכים.

כיצד עובדים ב- Protected Mode?

- כמו במצב ה- Real Mode מתבצעת הגישה לזיכרון.

- ב- Protected:

mov AL, byte ptr [EBX]

מציין שהמיקום מצביע אל הזיכרון.

Selector	offset

התוכן של ה- DS (16 סיביות)

מכיל מצביע (לא כתובת)

⇐ אינו מכיל כתובת, אלא מכיל קוד שמצביע על Descriptor

מתוך טבלת הסגמנטים.

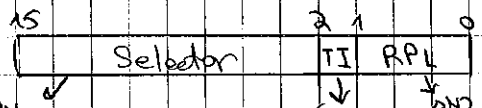
* כתובות מתקבלות ע"י ה- Selector (הקוד הבינארי

הנמצא בסיס הסגמנט. (שם ה- Descriptor)

⇐ פרט — מחברו שם כי אחרון מקבל אותו. (32 סיביות = Data Bus)

(לא כל קטעת מהזיכרון יש קוראת מהמעבד).

ה- Selector: הוא סוג סיבית שמצביע ל- Descriptor ובעזרת הסיביות הראשונות



מספר ה- Descriptor.

הסיבית ה- 2 קובעת אם ה- Descriptor נמצא ב- Global/Local Table.

- הגישה קדמה, כי ה- Descriptor נמצא בזיכרון הראשי ולכן הגישה ל- Selector

של ה- Descriptor ⇐ מתעכבת ב- Protected-Mode.

* ה- Descriptor מאורך של 4 מילים (= 32 סיביות), ולכן מחולק לכמה

כמות הסגמנטים, Type סוג הסגמנט. (כמה סיביות) ⇐ גודל מינימלי של סגמנט, 1MB = 2^{20}

limit - גודל. אומר לנו כמה הסגמנט. (כמה סיביות)

Descriptors:

Base ⇐ מיקום בתחילת הסגמנט בזיכרון הפיזי (בתים) ← 32 סיביות.

Limit ⇐ גודל הסגמנט (בתים) ← 20 סיביות.

Limit = 0 ⇐ הסגמנט ריק (אין שום בית שמור לו בזיכרון).

Limit = 4 ⇐ הסגמנט מכיל בית אחד שמור לו בזיכרון.

סיביות הרשאות: Access Right

A	DPL	S	E	ED/C	R/W	A

A = נגיש או לא נגיש לשימוש.

R/W, ED/C, E = שלושת הסיביות האחרונות מכוונות את הסוג של הסגמנט.

E קובע אם מדובר ב-Descriptor של Code או Data.

כשהוא 0- ⇐ מדובר בסגמנט נתונים.

ED - קובע כי הסגמנט הוא Data Segment ⇐ מדובר בסגמנט נתונים רגיל.
Stack Segment ⇐ מדובר בסגמנט המכיל את המחסנית.

E=1 ⇐ Code segment.

C ⇐ מאפשר לקוד הנוכחי גישה לסגמנטים בעלי DPL נמוך יותר.
Selector-ים

R=0 ⇐ לא מאפשר לקרוא את ה-Code Segment.
R=1 ⇐ כן ".

⇐ הקובץ שבו מוגדרים ה-Descriptor-ים של כל הסגמנטים בתוכנית.

GDT - הטבלה של

GDTR

יש לו תפקיד מיוחד בזיכרון ה-descriptors. (כי הוא מכיל את הכתובת בה נמצאת טבלת ה-descriptors).

ולכן מספר-הסדר שלו הוא 0 תמיד.

כאשר המעבד נמצא במצב Protected-mode.

Multi-Tasking - הכוונה ב-

Protected Mode - מצב הפעולה שאפשר את Multitasking.

- התפישה הזו מצריכה מערכת הפעלה שתנהל את ה- CPU שאפשר את הריבוי של התהליכים.

הוא מאפשר גישה לזיכרון מעל 1MB שיש לו.

אחד Segment - אבני הבניין (בשונה מהשיטה של קטעי הקוד כמו ב- real mode)

- בכל פעם שמבצעים תכנות/פיתוח - יש לפנות אל קריאות מהזיכרון Code Segment

(המעבד יהיה התואר CS).

יש 2 סיביות מיוחדות הקשורות לרמות ההרשאות של התוכנית, הנקראים RPL (יתואר להלן) סיביות ההרשאה של המשתמש (רשאי בפועל)

Global DT / local Des.. table - 13

- מבנה של Descriptor מתוך טבלאות ה- Des.

← ישנו מחשב נפרד של הקריאות ה Descriptor ← שלא מתבצע בכל פעם,

אלא הטבלה נשמרת על ה Memory cash

ניכנס מהיר (על הכיפ). מוען לזיכרון.

* ישנו GDT אחד ויכולים לנהל טבלאות נוספות (LDT) שמוגדרות

הראשי שמצביע על מיקום טבלאת ה- GDT = GDTR (והוא מוחבא - hidden).

ה- GDT מכיל מבנה נפרד טבלאת ה- LDT וה- LDTR הוא מצביע

מיקום טבלת ה- LDT (שני מצביעים).

מתניע Fetch ← פרט אשר ל- CS כפי שקראנו את ההוראה,

שלאחר מכן יש לבצע קריאה של הזיכרון.

מיוחדת (הרשאות)-

- RPL = שני הסיביות ה- Selector.

- DPL = Descriptor privilege level

שני סיביות של רמת ההרשאה שנשמרות בתוך טבלאת ה Descriptor.

- CPL = רמת ההרשאה של אותו התהליך שרץ כעת.

Code Privilege level.

כדי ה- CPU יוכל לגשת מקטע הרשאה מסוים שלפי "הסיווג" הנ"ל

$$DPL \geq \max(CPL, RPL)$$

רמת ההרשאה הכי גבוהה מבניהם

← מתאים גם אין הרשאה.

Segm- Linear Address נשמע כמותה הוירטואלי של המעבד.
(32 bits)

Virtual space = מרחב שבנוי רוב של מיכון אחד שבו אין בפועל.

Linear space = הכתובת הפיסית המלאה של הזיכרון.

שני מנגנונים שבונים אחד על השני:

Paging : ניהול אחסון של מידע.

HLT - הפסקת העבודה (הקפאת המעבד).

* נתון קטע מהמעבד ב- Protected שהיה בהשאה שלנו:

1 0F 01 90 90 90 B8

קטע - יותר אין אינו מיד!

* כתובת של אחד מ 16 סיביות:

הפקודה שיקחו: mov ax, 9090h (ביצוע של הפעולה לשם. קטע מאחר הפעולה האחרונה)

* כתובת של בת סיביות: 9090 - היא קוד אחרונה שהיה שם בקטע.

mov eax, 90909090h

כי, קטעים אלו- Real-Mode ו Protected -

שם לעשות נמצא הפקודה:

1. למצוא GDT (מבנה Descriptors שהוגדרו).
2. למצוא טבלה וקטורי הפסיקה (כי Real-Mode היה התחלה א-ס).
IDT
3. למחוק את הפסיקות.
4. להכניס את הכתובת GDTR (מצביע על התחלת מקום של GDT
→ יש אפשרות שלנו רק מצביע של הרשאה שלנו)
5. " " " IDTR.
6. לעבור למצב של Protected.
7. לעדכן מצבע של FAR. הפעלה הוירטואלים כמו, EIP.
8. מאתחלים את DS, SS (שהיו כאן קטע הבעיה).

* מהנדס ה- 8088 - שם תקוות רבות מאוד ורבים מעריכים היום שהוא היה יכול להיות, לפני הכל, מעבד ייעודי לפני מחשבי
* מעגל - Page - Addressing.
* תמיכה ב- Protected-Mode.
* הגדלת מהירות השעון.
* הוספת הפסקה מחומרה - רכיב שמעיר את ה- CPU לטיפול בהפסקה = PIC.
- הוא מנהל את מעגל הפסקה, הוא אחראי על הבחירה, Paging
בין הפסקה לשפר מהיר.
- הקומפיילר של האסמבלר הוא זה שכתב, כי יש הוראות ספציפיות
של האסמבלר. (בגלל שאסמבלר יודע לשפות בדיוק איזה שאומרים מתי מהיר).
- שפה האסמבלר מאפשרת לנהוג את האלגוריתמים
- מאפשר כתיבת קוד יעיל מאוד.
- הוא מהיר מאוד של מעבד מסוג (8088).
על מנת

סיכום:

* בדרגת הנמוך - הסרטנו של הדברים שאי אפשר את הפסיקות (השלמה).
כי יש לשלוף גם את תפקידם ואת ה- EIP, EFlags.

* ב- Protected Mode תמיד נסיים פונקציה ב- IRETD (ISR) ולא ב- IRET
ה- Descriptor של וקטור הפסקה- Trape Gate.

שלב וקטור הפסיקה ב- Protected Mode:

הכתובת הפיזיקלית - היא הכתובת שעליה נשלחת הבקשה

הכתובת הלוגית - היחידה של מרחב הזיכרון.

* הגדרה - Pipeline - מאפיין יותר מעבד מודרני ו יותר של Super-Scalar

מעבד שיש בו מספר שלבים - הביצוע חלוקה.
- מיבוץ הוראות
- סקירה של הוראה
- ...

יש אפשרות של הוראה שלישית אחרי 3-4 הוראות שהתקבלו כו-זמנית.

הדרישה המינימלית - מכל היותר שאיזשהו שלבים נמשכים יצא. (בפועל 9 - בא יתבע של שלבים שלבים).

* הגדרה מדויקת של אוטומט:

1. קבוצה של הסימנים שלנו. (Editor)
2. קבוצה של מצבים. הקומפיילר יכיר קובץ exe (אבל תמיד מתחיל בקובץ מקור).
   מתרגם כמה שאפשר מהקוד לשפת image של המחשב שניתן להריץ אותו.
   הקומפיילר לא מוצא יותר ← אין מילה וכתובה מסוימת אלא שגיאה כלשהי.
3. Linker ← מוסיף header לפני התוכנית שמאפשר למערכת הפעלה מה התוכנית מלכתחילה. מאפשר גם קישור של כמה קבצים ליצור התוכנית הסופית כמכלול.
4. Loader = מעתיק את הפונקציות של התוכנית. יוצר ע"י קובץ ההפעלה.
   קובץ הריצה. מלמטה מכל שהוא שהוא ← נוסף קובץ exe. (שהוא יותר ע"י התוכנית).

כל שפה יש לה הוראות שמקבלת מעתה פיזי.
הוראה מעתה מה שכתוב בזיכרון!

* כלי לבנה debug - מקבלים את ה- dll ומאפשרים לנו לראות את האופציה של שורה ב- debug.
  שורה בשלב store ← lb [xq]sp mov (מיזון מקף, ומשתנה מקבל אותה קבוצה מצד שני).

* מיזון - mov ax, 1 ; הוראת מיזון. הוראת מקף ההוראה.

כל מה שלא כתוב אחרי ; נחשב הערה ומתעלמים ממנו.

* הוראה add ax, (x+2) ← הפקודה של מחשבים כלשהו.

* הוראה jmp קובץ exe (מתאים שאין שכתוב המפעילים).

* כמה הוראות - מחשב יודע של SD, CS ואחרת נוספים שונים.

* הוראות תנאי - לשלב מצב: כמה:

שורה של מקור של לוגים).

* כל שורה של קוד (ה- list (LST) ב' ו שם הוספה שלב בתחילת שורה.

* הגדרה של המשתנה x הינו בעל הגדרה של התוכנית.

* מוצג לך לפניך את הפעולה שתבצע. (ניתן לעצור את הפעולה בכל שלב).
  את מבינה/ת מה קורה בכל שלב? אם לא, שאל/י שאלות.

* **הוראות כתיבה:**
  כתוב/כתבי את הפעולה שמוצגת לפניך, בשלבים. (שים/שימי לב לכך שכל שלב בפעולה מתחיל בפועל בלשון הציווי).

* שים/שימי לב, ייתכן שיהיו פעולות שאינך יודע/ת לבצע!

* **אין אפשרות:**
  - כתוב/כתבי הוראות
  - **לא**
  - ציור של הפעולה.
  - רשימת חומרים שתצטרך/י לביצוע.
  - תיאור מילולי של הפעולה.

משוב לתלמיד: