

מיקרו-מעבדים ושפת אסמבילר

קורס 83-255, תשפ"א

ד"ר רן גלס



שלום!

- סילבוס / שעות קבלה / לוגיסטיקה - הכל ב-Piazza
- ציון הקורס מורכב מ:
 - תרגילי בית (20%) - תרגילי תכנות ארוכים ו"כבדים"
 - מבחן מסכם (80%)
- דרישות קדם:
 - מערכות לוגיות ספרתיות (140-83); מבוא להנדסת תוכנה ומחשבים (120-83)
 - מומלץ גם: מבני נתונים ואלגוריתמים (19-83). מתשפ"ג-חוובה
- נהלי עבודה עצמאית

[hw1](#) [hw2](#) [hw3](#) [hw4](#) [hw5](#) [hw6](#) [ta_section](#) [exam](#) [logistics](#) [tips](#) [lectures](#)

[◀](#) Unread Updated Unresolved [⚙️](#)

[New Post](#) [🔍 Search or add a post...](#)

WEEK 8/12 - 8/18

שעת קבלה Instr 8/15/18
עופר טוב, מחר בשעה 15:00 מתוכננת שעת קבלה
הוּמָן (בכolumbia מינס 1 EnIcs 1 במעבדות). לרווחת המבנה

WEEK 7/22 - 7/28

פסיקות מתבקשות בו דמונת 2 7/24/18
במבחן לוגוגמא , שאלה 4 , סעיף ג, כשותתקנים .
מקרים פסיקה בו דמונת מה וורקאו ? איך אומן יודע
? האם פסיפה הטענה יבעג ואשונה ?

WEEK 7/15 - 7/21

תשובות למבחן בוחן 7/21/18
הרי רצוי לשאול אם יש תשובות למבחנים שהועלו
למהלך כפוי כן במבחן לוגוגמא שאלה 2 ג האם צריך
להתחשב בו שלמי הנקודה בהיבורו ?

תשובות לתרגיל בית 7/20/18
האם אפשר להעלות דוגמאות לפתרונות של שיעורי
mouse בתרגיל בית 4 merge sort ?
בתרגיל בית 6 שאלה 3 בתרגיל בית 6
תודה.

WEEK 7/1 - 7/7

... שאלה 1 - איך אני משנה את הצורה של הסמן 7/6/18
היה, רצוי בתרגיל 6 שאלה 1 כי צריך לשנות את
צורת הסמן , ולא ברור לי ממש איך אומן ורים לעשות
זאת... חורי כתוב שחשטו אמרו

שאלה 2, חזצת יומם 7/5/18
הPsiוקה שמתיחה את התאריך נקבעת 2 סוגיות שיכ
יום, יום ובשבוע (1-6) או יומם (1-31) איך מוסמ
צריך להציג או שווה לא משנה?

פונקציית דיבאג שאלה 1 7/5/18
כארפה פונקציה שיטולה לעוזר לכס בפתרון התרגיל
DX ונדבישה את `print_location` הפונקציה נקודות
החוודים מפשיטת העברת CX ואת

קבועת אופסט בזיכרון 7/4/18
שלום מתרגלים ומרצה נכבדים. רצויי לשאול היכן נקבע
אופסט של משתנים בזיכרון, האם נקבע בリンקר או
בלואדר? בפ' שהבנתי הדאטה טגנון

WEEK 6/24 - 6/30

...ית הגשה של תרגיל 6 ל 5.7.2018 6/26/18
את הגשת תרגיל 6 החזר בשבוע. תאריך הגשה מעדכו
5.7.2018 (יום חמישי של שבוע הבא) בנסוף יש גרסה
מעודכנת של התרגיל במודול

WEEK 6/17 - 6/23

סעיף ב' בתרגיל 6 6/20/18
א.האם ניתן להוכיח שהקפט והוויא 4 אובייקטים? או

Question History

! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

[question](#) [star](#)

תשובות לתרגיל בית

האם אפשר להעלות דוגמאות לפתרונות של שיעורי הפת

:בדgesch על

בתרגיל בית 4 merge sort 4

בתרגיל בית 6 mouse

? ושאלות 3 בתרגיל בית 6

תודה.

[ta_section](#)

[edit](#) · [good question](#) 0

[S](#) the students' answer, where students collectively construct a single answer

[edit](#) · [good answer](#) 0

[i](#) the instructors' answer, where instructors collectively construct a single answer

3_אלה.pdf

[edit](#) · [good answer](#) 0

followup discussions for fingering questions and comments

Start a new followup discussion

Compose a new followup discussion

הקדמה: הנדסת מחשבים



הקדמה: הנדסת מחשבים



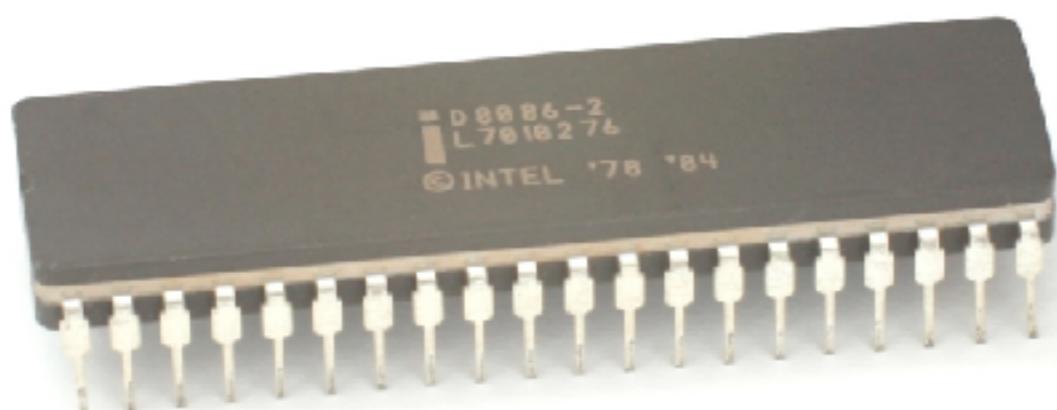
הקדמה: הנדסת מחשבים

מיקרו-מעבדים
ושפת אסמבלר



Intel 8086

- אונחנו נתמקד בمعالג Intel 8086
- יוצר בסוף שנות ה-70, 1979
- מעבד 16 ביט
- 5-10MHz
- מארז 40 פין





למה 6808?



- נשווה לⁱ⁷: Intel Core i7
- 3GHZ, 64Bit, מעורן 1366 פין
- מכיל 4-6 “ליבות”, רשת תקשורת זיכרון cache היררכי, בקר זיכרון פנימי
- מאות פקודות אפשריות (opcode)



למה 8086?

- 8086 מספיק פשוט
- בסיס להבנת מעבדים מתקדמים
- לומדים **עקרונות** ולא מעבד ספרטיפי
- עד 5 שנים יהיה מעבד אחר....

אייפה משתמשים במעבדים?

- "מחשב"
- מערכות
מושבצות מחשב
(Embedded)





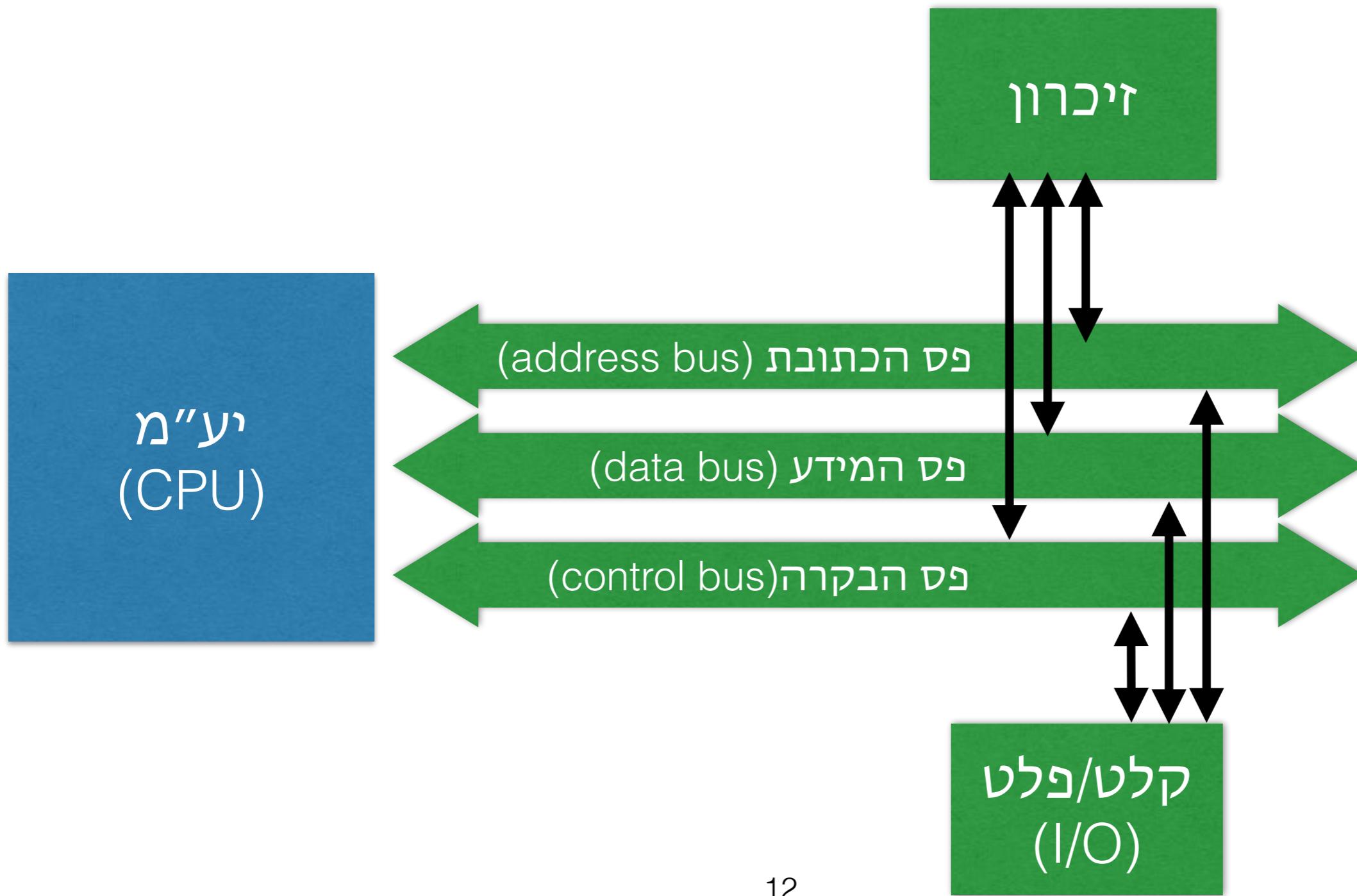
אוניברסיטת בר-אילן
Bar-Ilan University

מבנה "מחשב"

- איך בנוי מחשב?

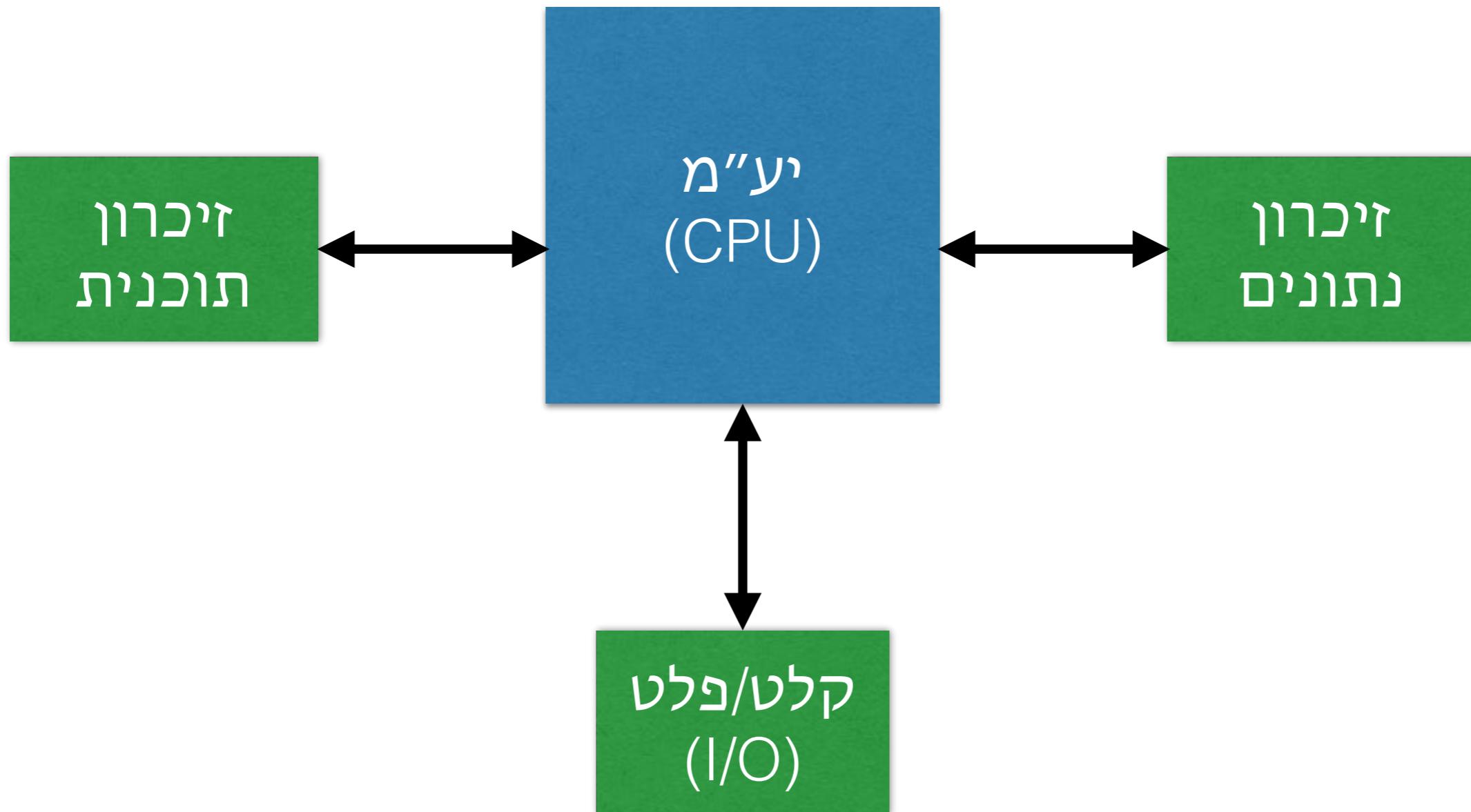
מבנה "מחשב"

ארQUITקטורת פון-נוימן



מבנה "מחשב"

ארQUITקטורת הרווארד



ארכיטקטורת מחשב

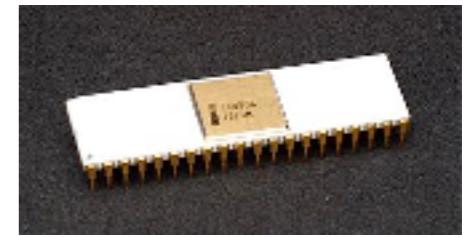
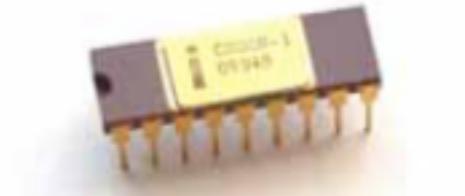
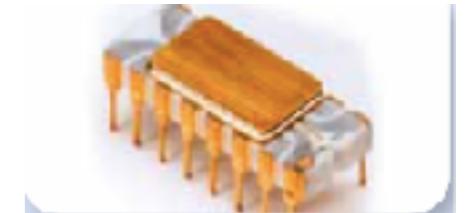
- **הרווארד:**
 - פס נפרד לזכרו המידע ולזכיר בו שמו התוכנית
 - פעולה "מקבילה" בין זכרו, התקני קלט/פלט והתקדמות התוכנית
- **פונ-ניומן:**
 - ביצוע פעולה ייחודית בכל פעם (צואר בקבוק - פס משותף)
 - כתיבה/קריאה מהזיכרון "מעכבת" התקדמות התוכנה
 - לרוב גישה לזכרו איטית מפעולת מעבד

מיקרו-בקרים

- מיקרו-בקר הוא שבב יחיד שעליו יש "מחשב" שלם - יחידת CPU, זכרון, ומספר התקני קלט פלט (למשל: שעון/טיימר, AD/DA, רכיב תקשורת)
- לרוב מיועדים למערכות Embedded (General-Purpose) שהוא "אימטני" (לעומת CPU "אימטני" שהוא נפוץ):
- ארכיטקטורת בקרים נפוצים:
 - 8051/8052 - 8ビット, ARC, פון-נוימן עם שינויים פותח בשנות 1980 (Intel) והיה קיים פחות או יותר בכל מקום (הרבה חברות ייצרו צ'יפים תואמים). הפסקת ייצור ב-2007 (!!)
 - ARM / Cortex - סדרה ארוכה של בקרים 32-64ビット, למשל ב-BeagleBoard ו-Raspberry Pi ArduinPi ו-embedded מערכות חדשות
 - AtmelAVR - ארכיטקטורת הרווארד. קיים ב-AtmelAVR FLASH. הראשון עם FLASH.

מעבדים על ציר הזמן

שם	שנה	מאפיינים
Intel 4004	1971	4 ביט, 740 קה"צ, 2,300 טרנזיסטורים
Intel 8008	1972	8 ביט (עד 16kb זכרון), 800kHz
Intel 8080	1974	8 ביט (עד 64kb זיכרון), 2MHz
Intel 8086	1978/9	16 ביט (עד 1MB זיכרון), 5MHz כ-29,000 טרנזיסטורים



מעבדים (אינטל) על ציר הזמן

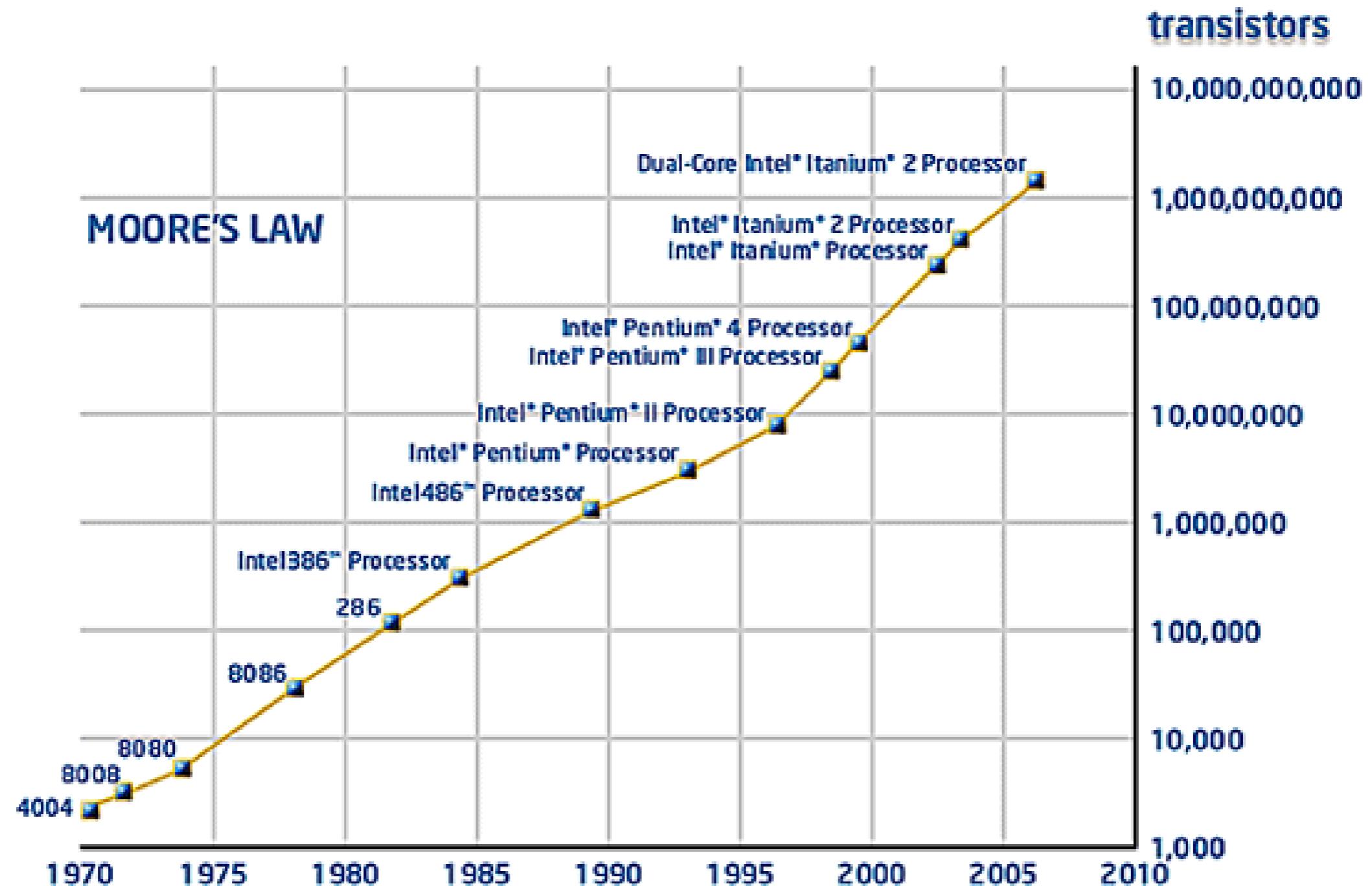
שם	שנה	מאפיינים
Intel 80286	1982	16 ביט, 25MHz, מרחב זיכרון 16MB 134,000 טרנזיסטורים
Intel 80386	1985	32 ביט, 40MHz, מרחב זיכרון עד 4GB 275,000 טרנזיסטורים
Intel 80486	1989	32 ביט, עד 150MHz, cache... 1.2 מיליון טרנזיסטורים, זיכרון...
Intel 80586 (Pentium)	1993	60-300MHz, כ-3 מיליון טרנזיסטורים...
Intel Core i3/i5/i7	~2010	64 ביט, מהירות שעון 3Ghz 4-8 ליביות עיבוד, 995 מיליון טרנזיסטורים



Moore's Law

2X transistors/Chip every 18 months

Performance is doubled every ~ 18 months



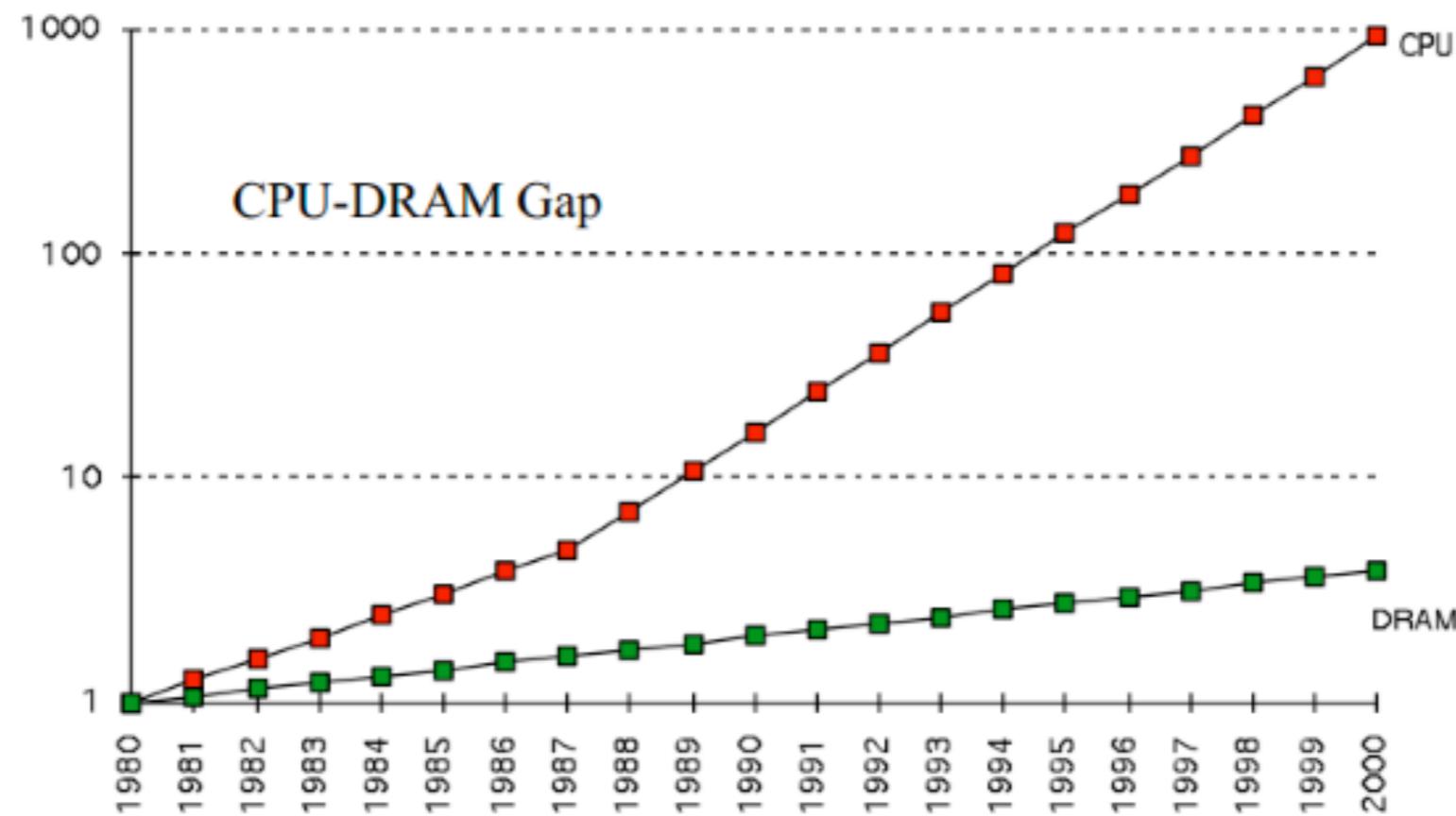
Graph taken from: <http://www.intel.com/technology/mooreslaw/index.htm>

אתגרים לחוק Moore

- צוואר הקבוק עובר לרכיבים אחרים (למשל: זיכרון)
- גודל טרנזיסטורים
- צריכת הספק: ה-Wall

מהירות מעבד מול זיכרו

- Processor vs Memory Performance



1980: no cache in microprocessor;

1995 2-level cache

פתרונות: הוספה זיכרון Cache מהיר (וקטן) על המעבד

Power Wall

$$P_{dyn} = CV^2 f$$

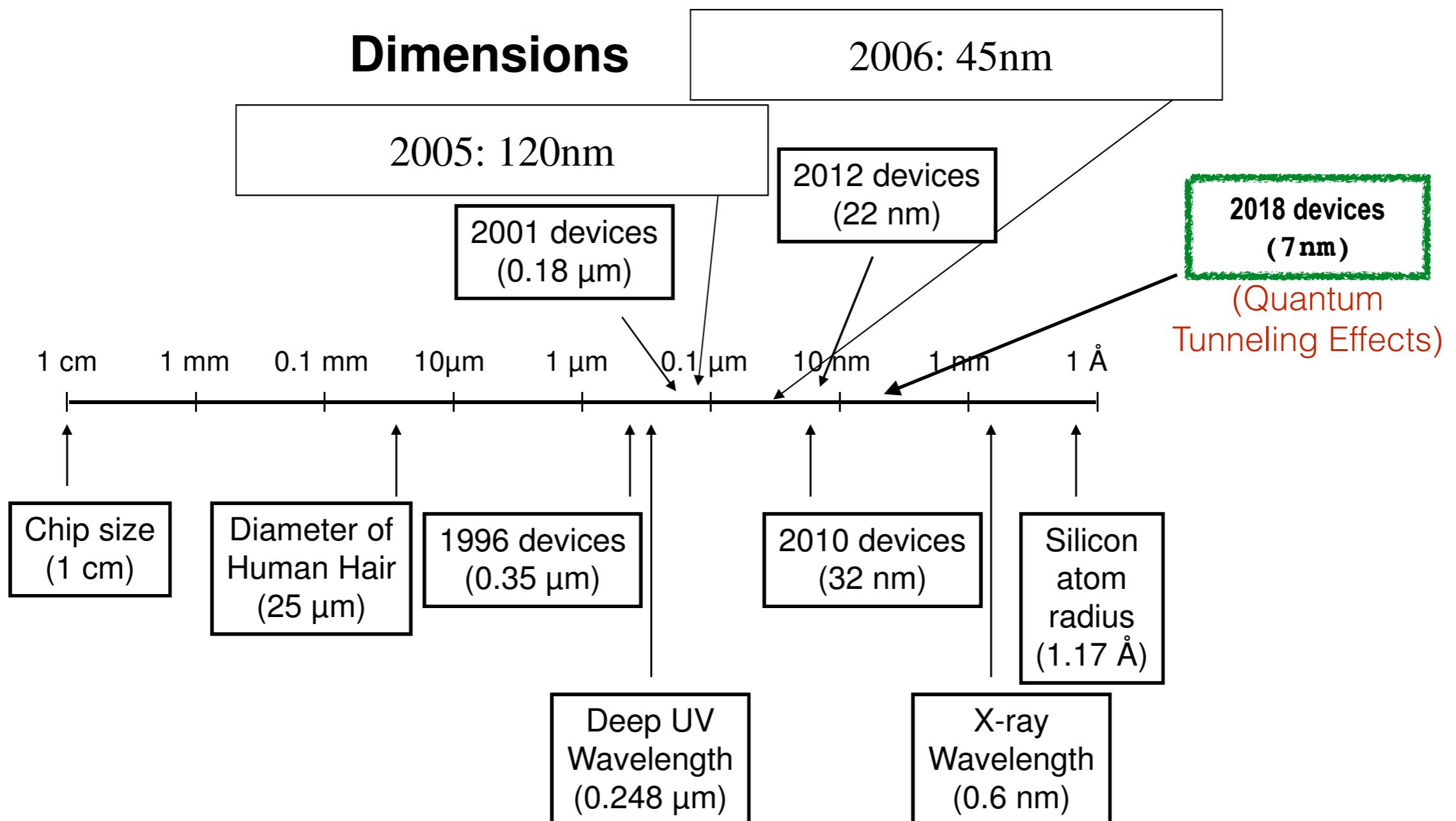
- הספק = עומס קיבול × מתח² × מהירות מיתוג

Processor	Clock rate	Power	Voltage
80286 (1982)	12.5Mhz	3.3w	5V
80386 (1985)	16Mhz	4.1W	5V
80486 (1989)	25Mhz	4.9W	5V
Pentium (1993)	66Mhz	10.1W	5V
Pentium pro (1997)	200Mhz	29.1W	3.3V
Pentium 4 Willamette (2001)	2Ghz	75.3W	1.75V
Pentium 4 Prescott (2004)	3.6Ghz	103W	1.25V
Core 2 Kentsfield (2007)	2.667Ghz	95W	1.1V
Core I7-37xx IVY Bridge (2012)	3.4Ghz	77W	1.1V

מעבר לריבוי
ליבנות →

קצב שעון
+
הורדת מתח
מתוך נמור
يُؤثِّر بعمليَّة
اللِّيجَة ورَجْيَشَة

הקטנת טרנזיסטורים



פתרונות: מעבר לטכנולוגיה אופטית (??)

The Intel 8086 MicroProcessor

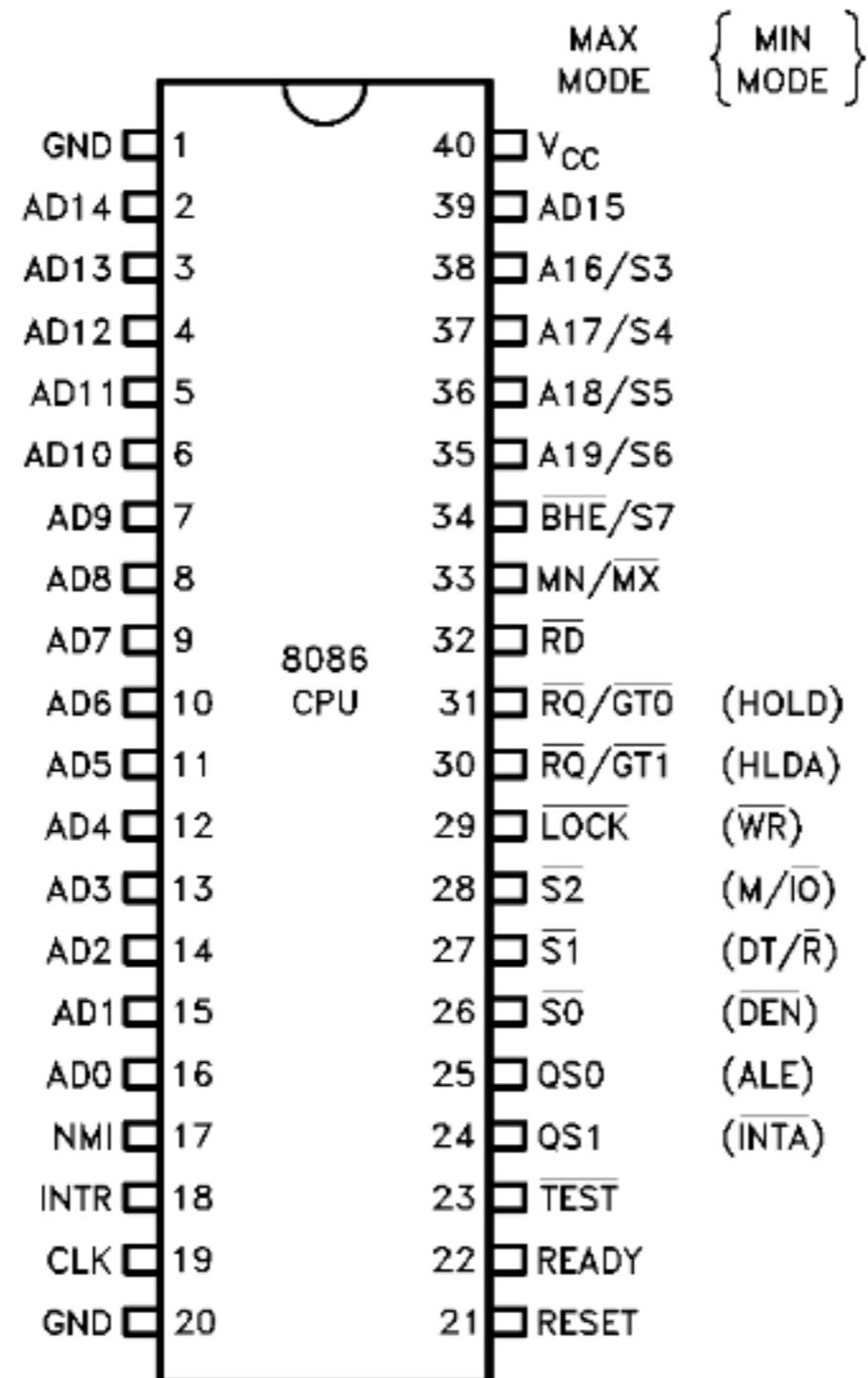
Intel 8086

- מעבד "16 בית", תדר מקסימלי 5Mhz, 40 רגילים.
- המעבד מותאם לארכ' פון-ניאומן
- זיכרון נתונים/פקודות חיצוני (לא cache)
- פסים חיצוניים (Bus, עורך, אפיק):
 - פס נתונים באורך 16 בית
 - פס כתובות באורך 20 בית (מרחב זיכרון של 1MB)
 - פס בקרה של כ-17 אותות



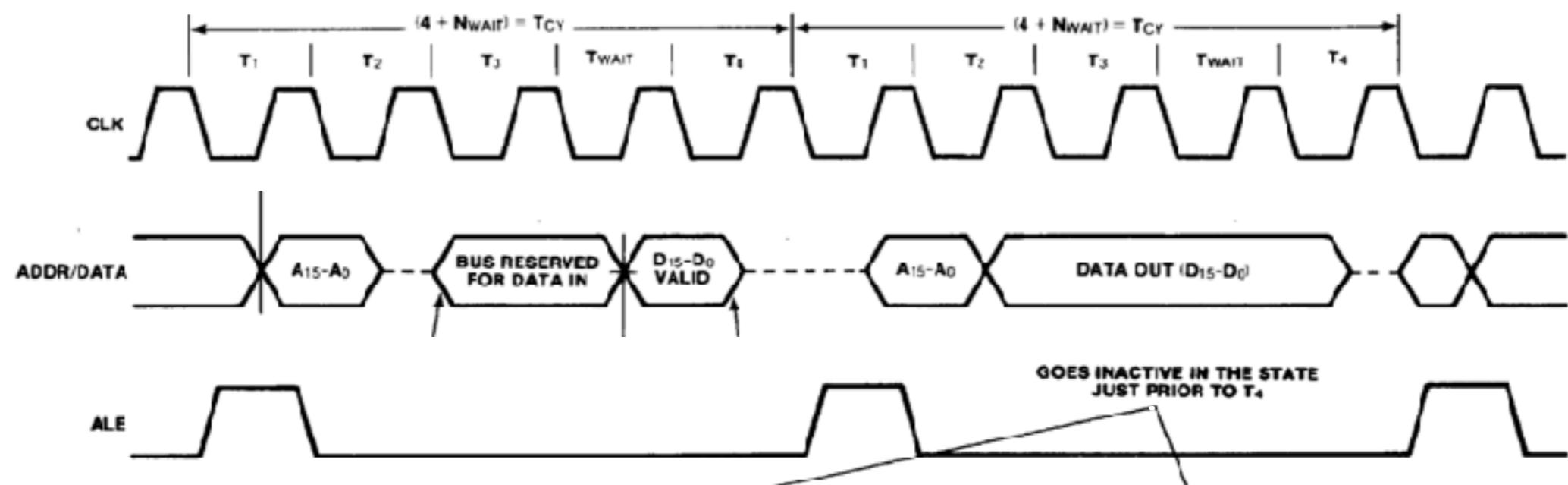
איך הכל נכנס???

8086 Pin Layout



תזמוו פסים

- ה-CPU מተזמן את תהליכי העברת מידע בין יחידות המחשב (זיכרון, O/I) ומפעיל אותם ע"י שליטה ב-BUS
- פסי הכתיבה/מידע/בקרה מרובבים ייחדיו (חלקית) ע"י שימוש באותם פינים, כאשר כל מידע מופיע בתזמון שונה (ריבוב בזמן)



גישה לזכרון

- איך המעבד ניגש (חטמלה) לזכרון?

גישה לזיכרון

U6264 8k x 8bit SRAM

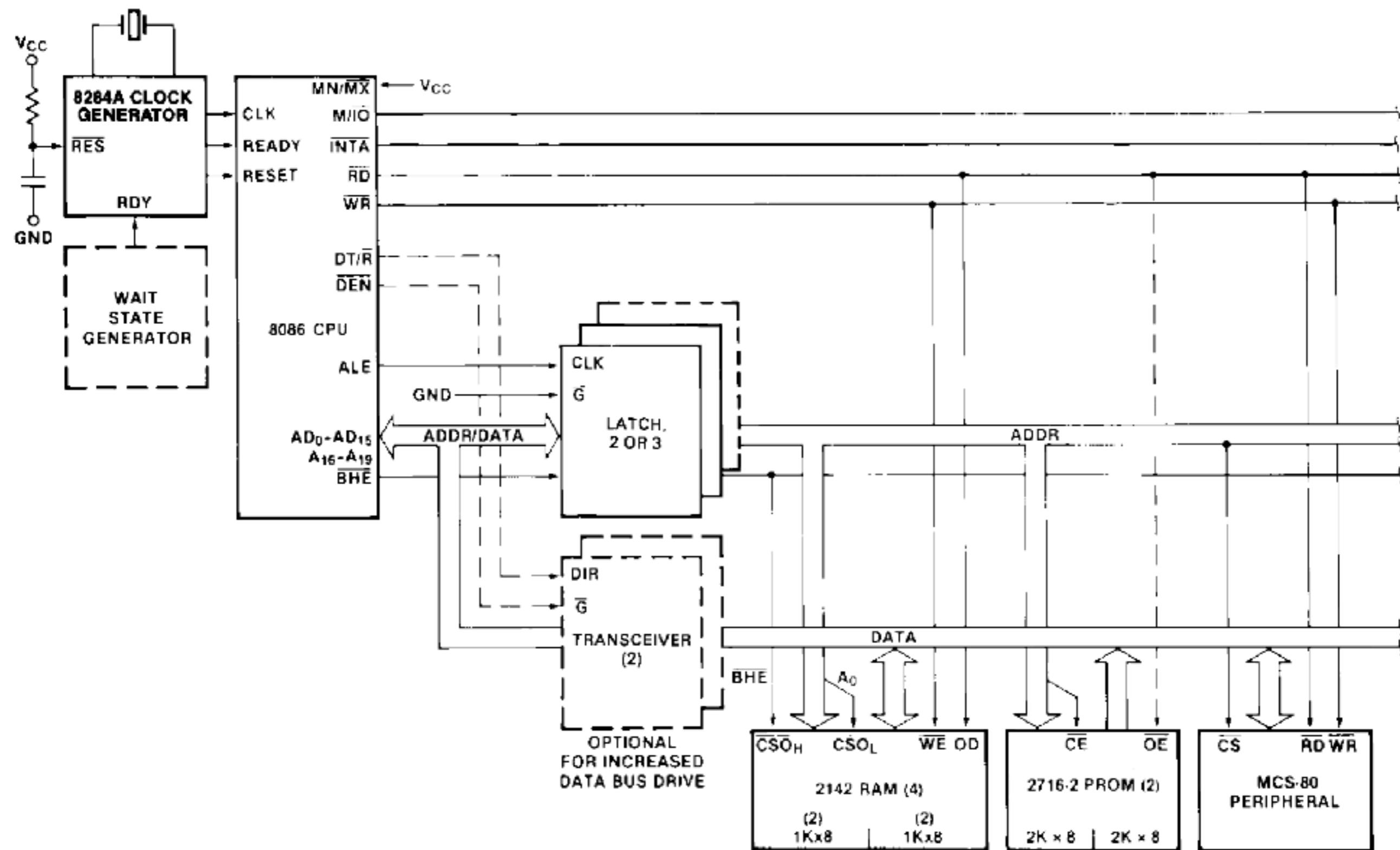
Pin Configuration

n.c.	1	28	VCC
A12	2	27	\overline{W} (\overline{WE})
A7	3	26	E2 (CE2)
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	PDIP	\overline{G} (\overline{OE})
A2	8	SOP	A10
A1	9	20	$\overline{E1}$ ($\overline{CE1}$)
A0	10	19	DQ7
DQ0	11	18	DQ6
DQ1	12	17	DQ5
DQ2	13	16	DQ4
VSS	14	15	DQ3

Top View

Pin Description

Signal Name	Signal Description
A0 - A12	Address Inputs
DQ0 - DQ7	Data In/Out
$\overline{E1}$	Chip Enable 1
E2	Chip Enable 2
\overline{G}	Output Enable
\overline{W}	Write Enable
VCC	Power Supply Voltage
VSS	Ground
n.c.	not connected



231455-5

Figure 4a. Minimum Mode 8086 Typical Configuration

מחזור פקדה בסיסי

תזכורת "תכו לוגי"

- עיבוד פקודה FETCH-EXECUTE מכיל 5 שלבים:
 - **FETCH** - היבאת פקודה מהזיכרון
 - **DECODE** - "פונוח" הפקודה
 - **READ** - גישה ל זיכרון (היבאת נתונים)
 - **EXECUTE** - ביצוע הפקודה
 - **WRITE**- גישה ל זיכרון (כתיבת תוצאה אם צרי)

8086 Fetch-Execute

- כתובת הפקודה הבאה לבייצוע שמור ב IP (אוגר מעבד)
- מחזור פקודה 8086:
 - היבאת הפקודה מכתובת שרשומה ב-IP
 - הגדלת IP לכתובת הפקודה הבאה
 - פענוח ובייצוע הפקודה
- מעבד multi-cycle - ביצוע פקודה לוקה מספר רב של מחזורי שעון פקודה אחד מבוצעת בכל פעם.
- קיים "pipeline": הפקודות הבאות מובאות ונשמרות **בthora**, בעוד פקודות קודמות מבוצעות....

סוגי פקודות

- **גישה ל זיכרון:** כתיבת / קריית נתון
- **"חישוב"** על נתונים שהובאו מהזיכרון
 - אритמטי: חיבור, חיסור, כפל, ...
 - לוגי: סיבוב, AND, OR, ...
- **בקה:** הרצה "לא לינארית" של התוכנית בזיכרון
 - קפיצה לכתובת מסויימת (והמשר הרצה משם)
 - לולאות, קריאה לחת-רוטינות, וכו'

RISC vs CISC

- קיימות שתי גישות לגבי מורכבות (וכן גודל) סט הפקודות
שהמעבד תומך:
 - Complex Instr. Set Comp. **CISC**
 - Reduced Instr. Set Comp. **RISC**

RISC

- גישת RISC טענת ש "פחות זה יותר"
 - עד ~ 100 פקודות בסיסיות
 - פקודה באורך קבוע
- דוגמאות:
 - MIPS™, ARM ™, Sparc™, Alpha™, PowerPC™
- יתרונות:
 - מעבדים יותר קלים לתוכנו, ויתר מודולריים
 - מחזור Fetch-Execute פשוט ו אחיד; פקודות פשוטות
 - מעבדים מתקדמים יותר נשאים עם אותו סט פקודות בדיק - תמינה לאחור ותמינה לפנים
- חסרונות:
 - הרבה פקודות נדרשות לביצוע "פעולה אחת";
פקודות "מוגבלות" (למשל, חישוב רק בין אוגרים)

CISC

- **תמיינה בריבוי פקודות** - כל פעולה "בסיסית" מקבלת פקודה נפרדת
- **שפת הסך הופכת חזקה וקרובה לשפה "עילית"**
- **דוגמאות:** מעבדי Intel, AMD
- **יתרונות:**
 - ביצוע פועלות "מורכבות" בפשטות (למשל, פעולה חישוב בין אוגר ו זיכרון, גישה עקיפה ל זיכרון)
 - תוכנית קצרה יותר (פחות פקודות לביצוע פועלות)
- **חסרונות:**
 - **פקודות מסובכות**, אורך פקודה משתנה, מהזור Fetch-Execute מורכב
 - **קומpileציה** יותר מורכבה

RISC vs CISC

“RISC”

```
LOAD A, ADDR_1
LOAD B, ADDR_2
PROD A, B
STORE ADDR_1, A
```

“CISC”

```
MUL ADDR_1,ADDR_2
```

- גישת CISC יותר דומיננטית בעולם (תוכנה קיימת)
- מעבדים רבים (>Pentium-2) מתרגמים פנימית פקודת RISC לאוסף פקודות CISC

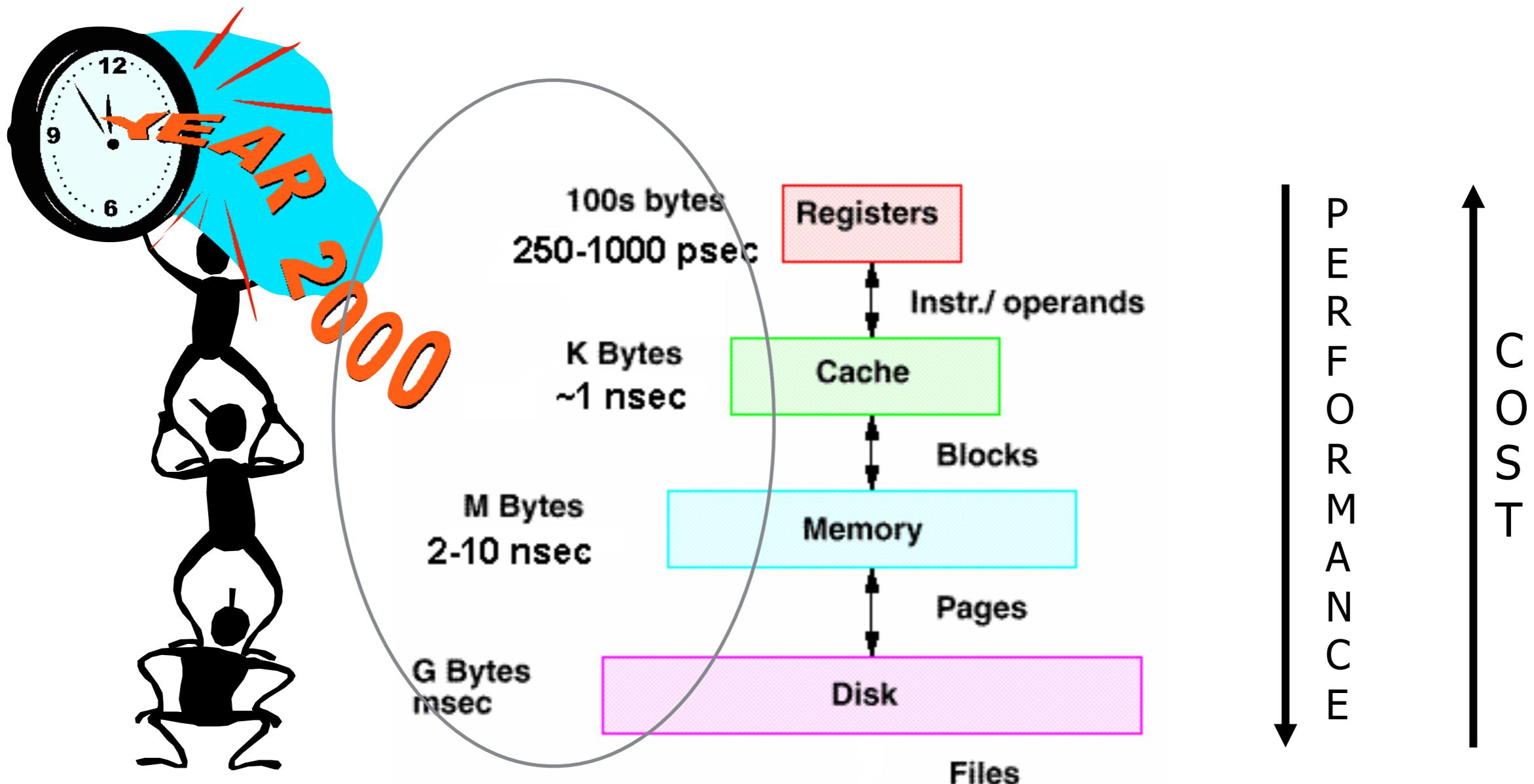
אורים: מקורות וזכויות יוצרים

- Wikipedia (Intel 8080,8086)
- Intel 8086 Specs, Intel, 1980
- U6264B 8k SRAM datasheet, ZMD, 2004
- <http://www.theinquirer.net/inquirer/feature/2124781/microprocessor-development>
- <https://www.extremetech.com/extreme/188776-how-l1-and-l2-cpu-caches-work-and-why-theyre-an-essential-part-of-modern-chips>

אוגרי המעבד ומבנה הזיכרו

מיקרומעבדים ושפט אסמלר 83-255

היררכיה זיכרון



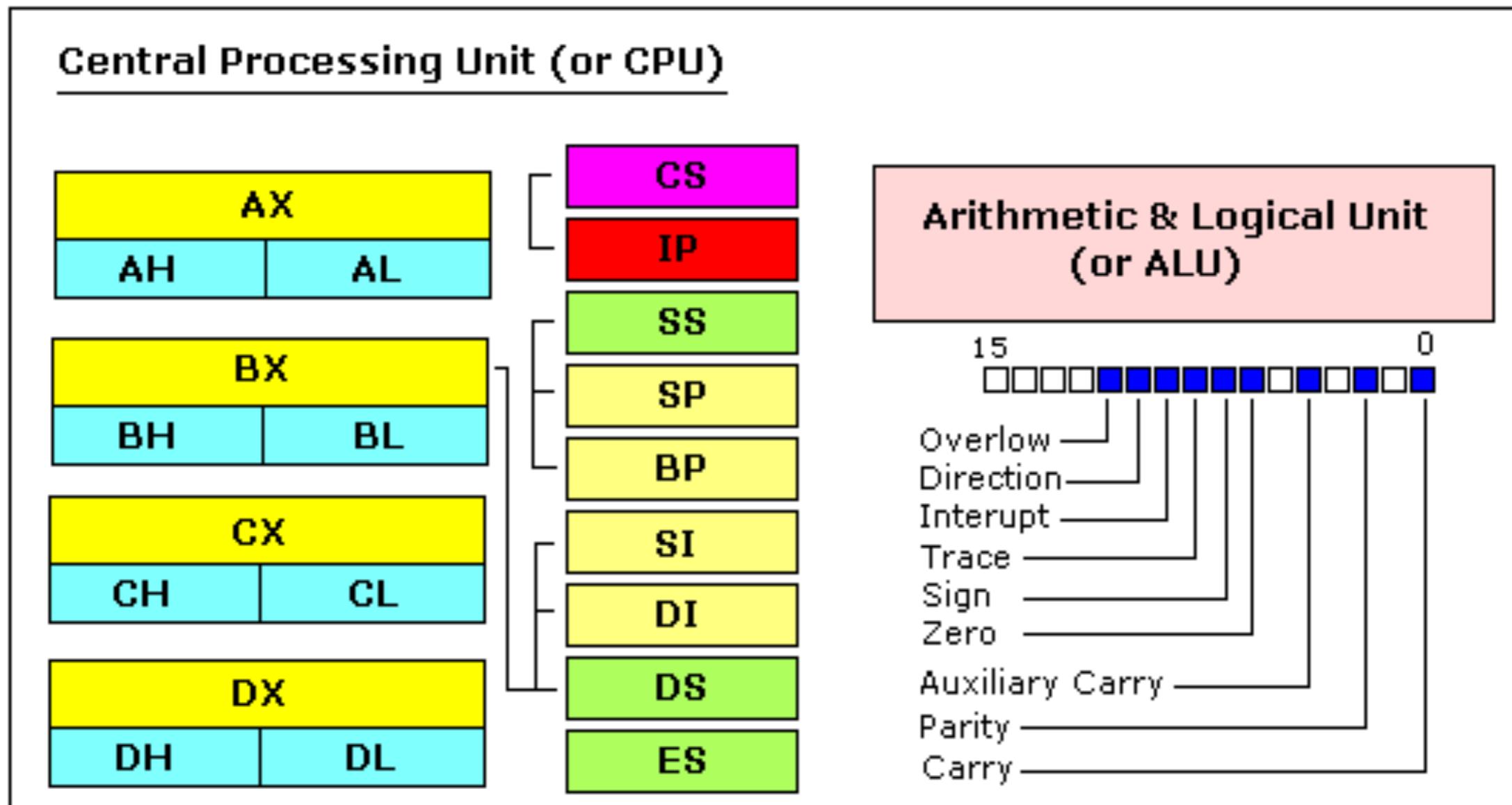
أוגרי המעבד

- האוגרים הם ייחדות זיכרונו "קטנות" ו" מהירות "
- מכילים מידע: נתונים, כתובות בזיכרון, וכו'.
- רוב פעולות המעבד מבוצעות על מידע הנשמר באוגרים (בדומה ל"משתנים" ב-C)
- ב986: כל אוגר מכיל 16 בית
- בחלק מהאוגרים ניתן לבצע פעולות על "חצאי" המידע: 8 בית עליוניים או 8 בית נמוכים.

אוגרי המעבד

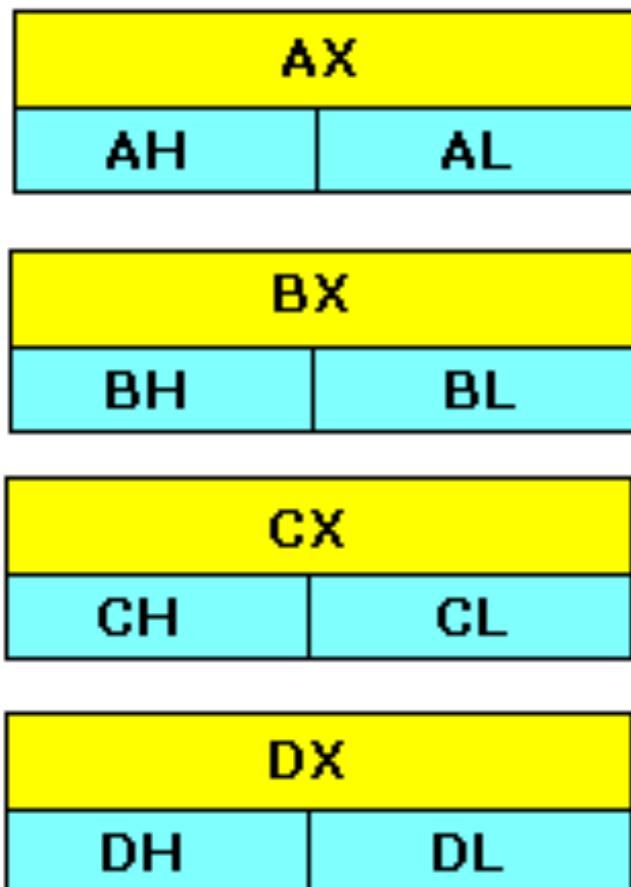
- 1 + 1 אוגרים:
 1. אוגרים כללים (4)
 2. אוגרי הצבעה / אינדקס לזכרו (5)
 3. אוגרי סגמנט זיכרו (4)
 4. אוגר "דגלי" סטוס

אוגרי המעבד



אוגרי נתונים

- 4 אוגרים כלליים, המשמשים להחזקת נתונים/כתובות:
AX, BX, CX, DX



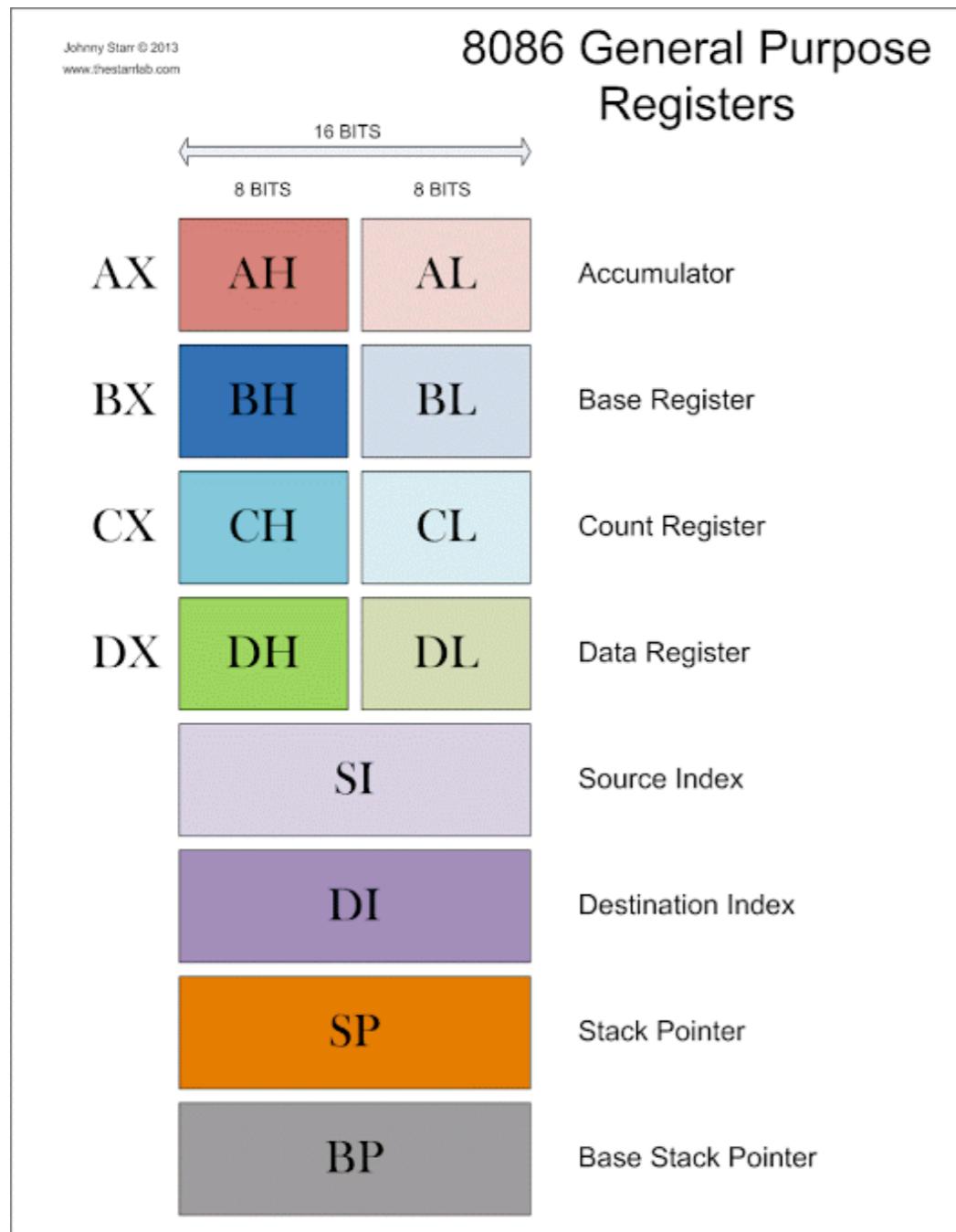
- ביצוע פעולות בין אוגרים
e.g., ADD AX, BX

$$\text{AX} := \text{AX} + \text{BX}$$

- ניתן לפנות רק ל"חצי אגר" 8 ביט
 $\text{AX} = (\text{AH}, \text{AL})$

e.g., MOV AL, 22h

אוגרי הצבעה לזכרון



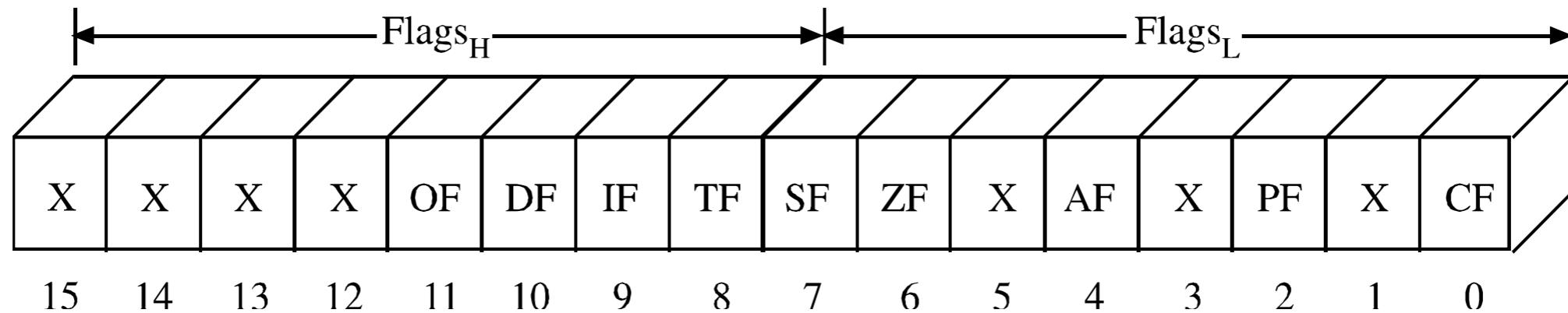
- אוגרים המשמשים בד"כ להצבעה **לכתובת זיכרון:** SI, DI, **SP**, BP, **IP**
- פועלות 16 ביט בלבד
- אוגרים מיוחדים (להשתמש בזיהירות):
- **IP** - כתובת למיקום הפקודה הבאה **לביצוע**
- **SP** - מצביע לנטוון האחרון במחסנית

אוגרי הסגמנט

- 4 אוגרי 16 ביט משמשים כמצבייע "סגמנט" לזכרון
(סביר בהמשך השיעור על מבנה זיכרון וסגמנטציה)
- CS, DS, SS, ES
- CS- מצביע לאיזור הזיכרון בו נשמר קוד התוכנית (code)
(קשר למצביע IP)
- SS- מצביע לאיזור הזיכרון בו נשמרת המחסנית (stack)
(קשר למצביע SP ולמצביע BP)
- DS- מצביע לאיזור זיכרון בו נשמרים נתונים (data)
(קשר לאינדקסים SI, DI, אוגר BX)
- ES מצביע "אקסטרה" לשימוש כללי (Extra)

אוגר הדגלים

- אוגר מיוחד השומר את "סטוס" המעבד
- 16 ביט - לכל בית משמעות שונה
- הדגלים משתנים בעקבות **פעולות אחרות המבצעות** במעבד או בעקבות פקודות הדלקה וכיבוי דגלים
- לא ניתן לגשת שירות לאוגר השלם



אוגר הדגלים

Flag L :



CF: Carry Flag $\begin{cases} \text{CF} = 0 : \text{No Carry (Add) or Borrow (SUB)} \\ \text{CF} = 1 : \text{high-order bit Carry/Borrow} \end{cases}$

PF: (Even) Parity Flag (even number of 1's in low-order 8 bits of result)

AF: Aux. Carry: Carry/Borrow on bit 3 (Low nibble of AL)

ZF: Zero Flag: (1: result is zero)

SF: Sign Flag: (0: positive, 1: negative)

אוגר הדגלים

Flag H :



TF: Trap flag (single-step after next instruction; clear by single-step interrupt)

IF: Interrupt-Enable: enable maskable interrupts

DF: Direction flag: auto-decrement (1) or increment(0) index on string (block move) operations

OF: Overflow: signed result cannot be expressed within #bits in destination operand

זיכרוו המחשב

זיכרון המחשב

מידע כתובות

00000h	00h
00001h	B2h
00002h	FFh
00003h	00h
00004h	15h
⋮	⋮

- **הזיכרון הוא מערכ גדול של "תאים"**
- **לכל תא יש כתובות, כל תא שומר 8 ביט**
- **מרחב כתובות של $1MB = 2^{20}$ כתובות.**
כתובת ראשונה 00000h אחרונה FFFFFh
- **ניתן לגשת לכל כתובות (זוגית/אי-זוגית)**
ולקרוא/לכטוב 8 או 16 ביט (Byte/Word)

זיכרון המחשב

מידע כתובת

00000h	00h
00001h	B2h
00002h	FFh
00003h	00h
00004h	15h
⋮	⋮

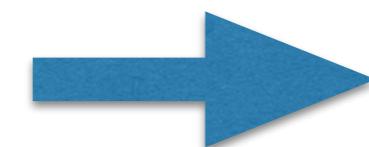
MOV AX, 1234h
MOV [0000h],AX



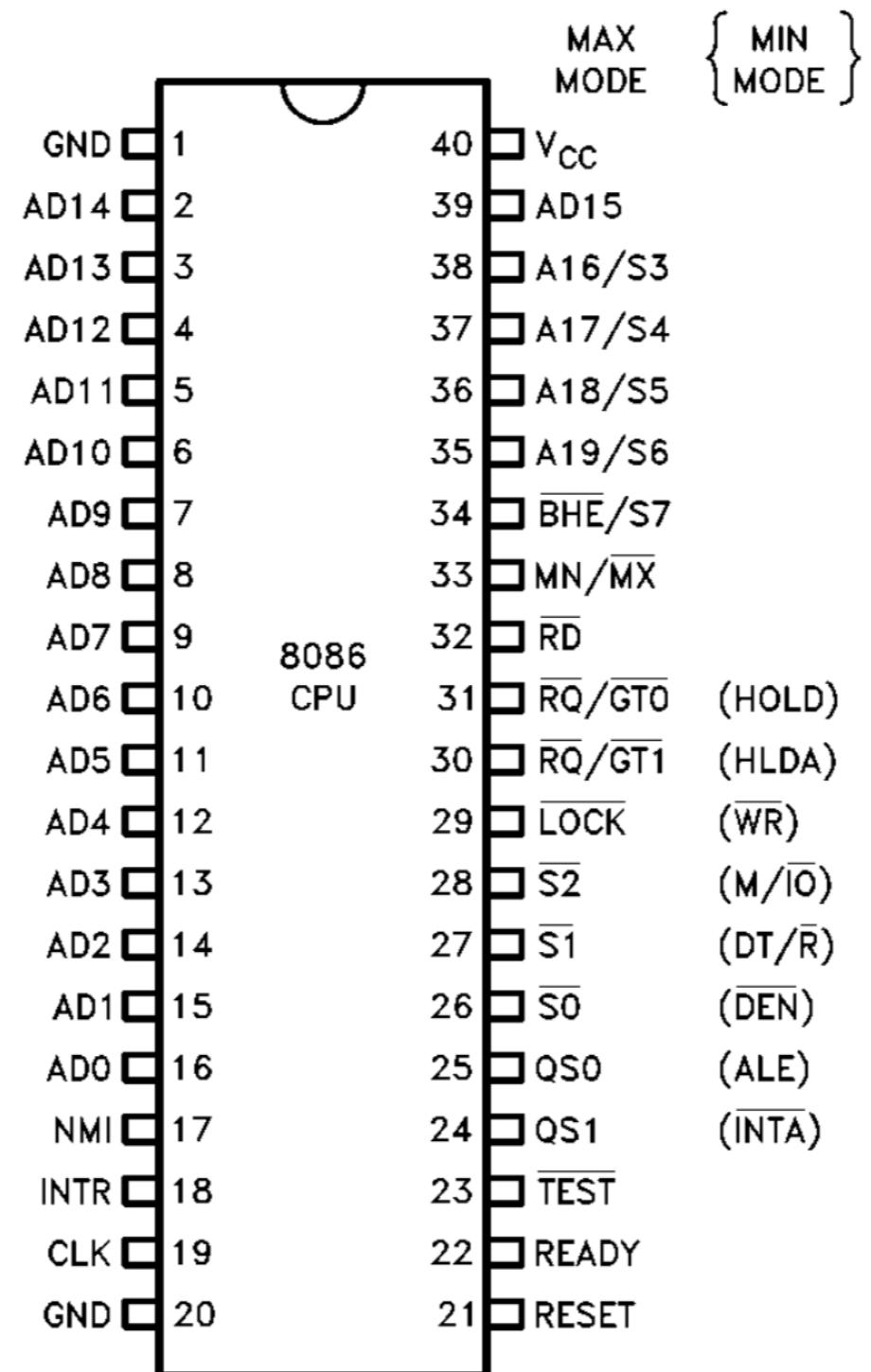
מידע כתובת

00000h	34h
00001h	12h
00002h	FFh
00003h	00h
00004h	15h
⋮	⋮

MOV AX, 4567h
MOV [0001h],AX



8086 Pin Layout



231455-2

40 Lead

בעירה?

- גודל כל אוגר הוא 16 ביט
(קטן מידי מכדי לשמר "כתובות" או מצביע לזכרון)
- איך ניתן לגשת למרחב כתובות של 20 ביט?

פתרון - סגמנטציה

- כדי לגשת לכתובת זיכרון המעבד עושה שימוש ב-2 אוגרים, אוגר **סגןט** וואוגר **הצבעה**
- כתובת בפועל =

כתבת סגןט × 10h + כתובת מצביע (היסט)
- למשל, BX=0050h , DS=1111h
כתבת בפועל:

פתרון - סגמנטציה

- כדי לגשת לכתובת $\text{BX} + \text{כתובת סegment}$ אבד עווה שימוש ב-2 אוגרים, אוגר סegment הנקרא "נתונים" (Data Segment)
- כתובת בפועל =

כתובת segment $\times 10h$ + כתובתBX (היסט)

- למשל, $\text{BX}=0050h$, $\text{DS}=1111h$:
כתובת בפועל:

11110h

+ 0050h

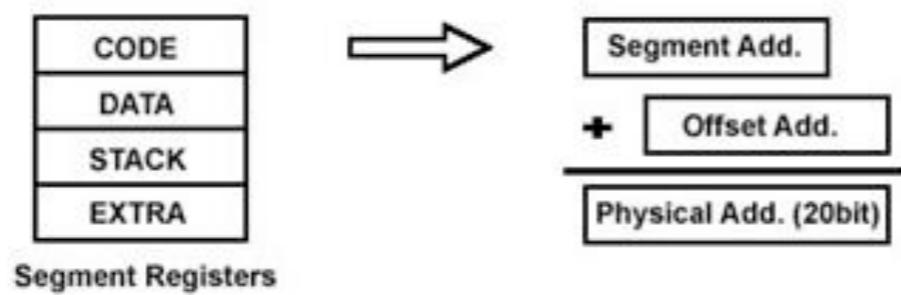
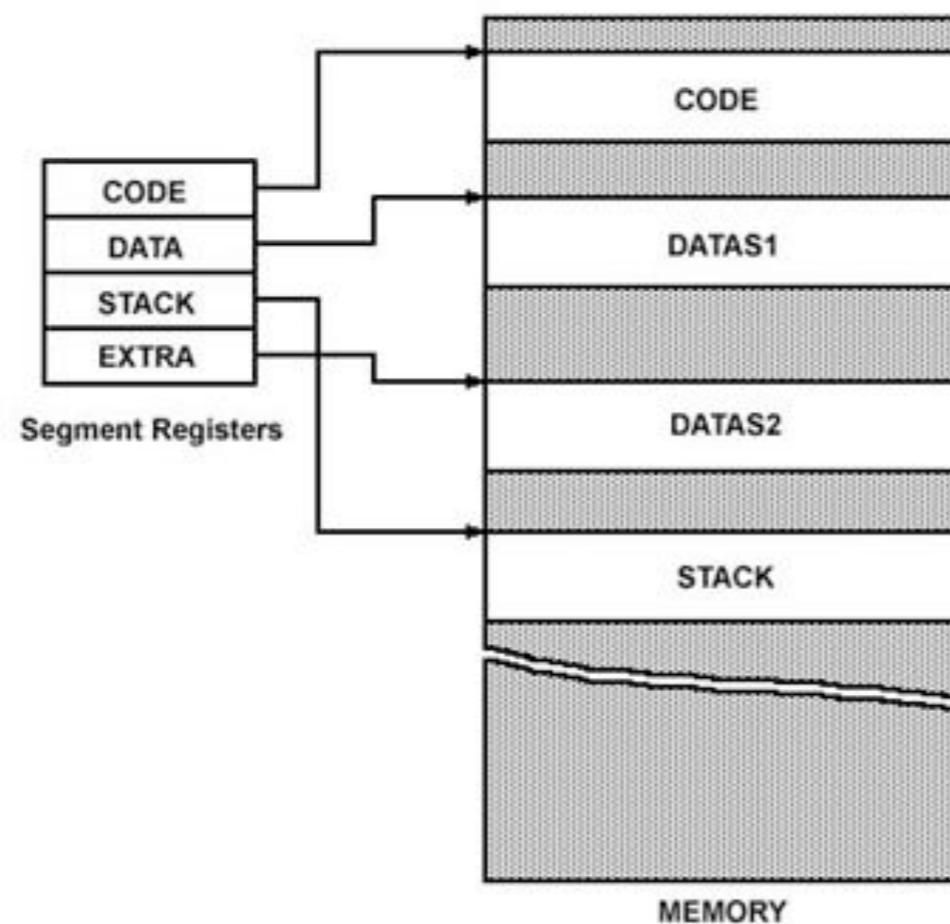
=====

11160h

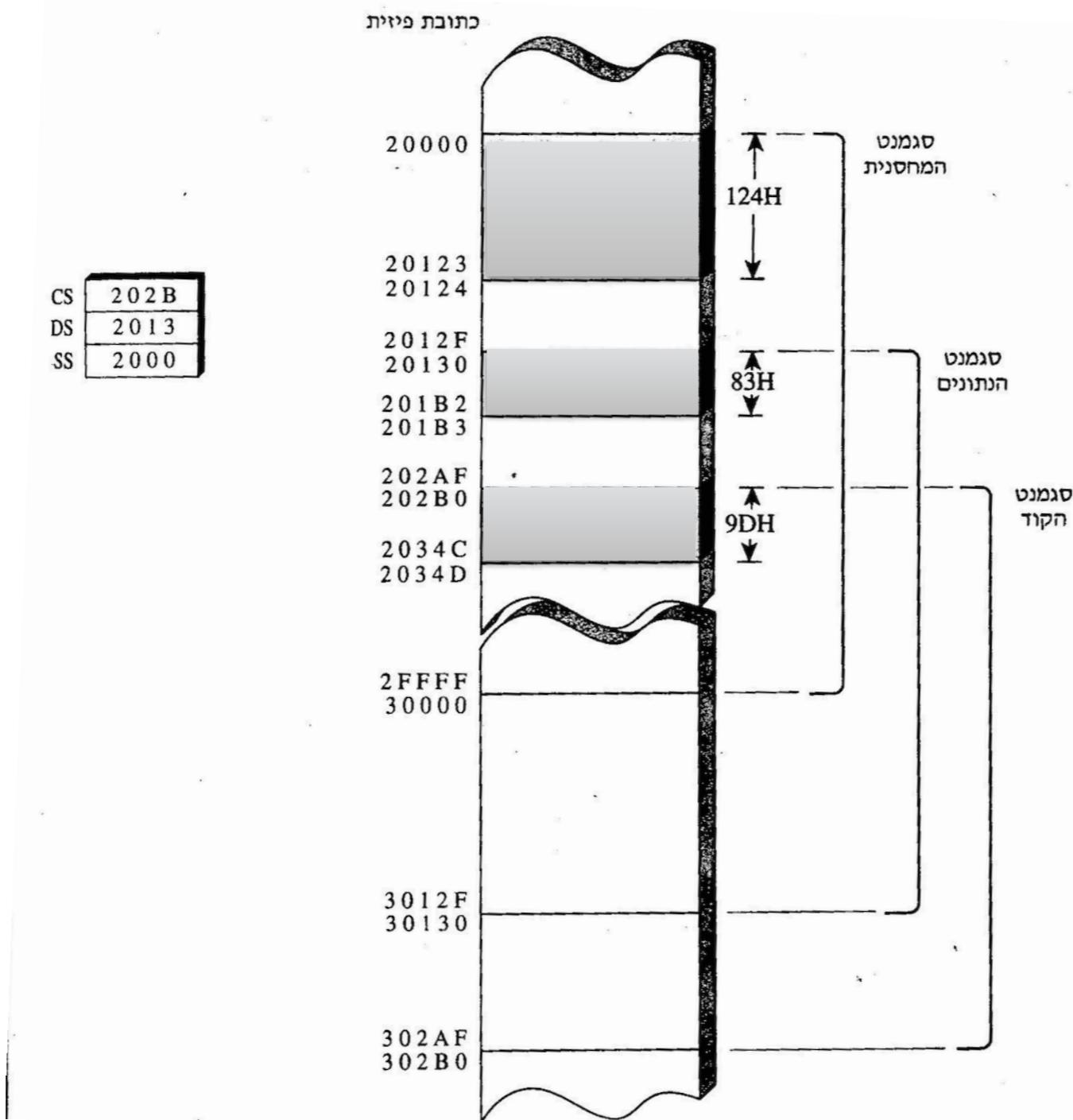
סגמנטציה

- כל מקטע הוא רצף של 64k כתובות (16 ביט היסט)
- קיימים 4 מצביעי סגמנט:
קוד (CS), מחסנית (SS), נתוניים (DS), אקסטרה (ES)
- סגמנטים יכולים להפוך:
למשל, $DS = 2500h$, $ES = 2000h$
חופפים ב-44k כתובות (בין $2FFFFh$ ל $25000h$)

סגןנטציה



סגןנטציה



הסגןטים השונים

- **מקטע הקוד:**
- מכיל את קוד התוכנית שרצה
- בכתובת IP:CS נמצאת הפקודה הבאה שהמעבד צריך לבצע.
- שינוי IP/CS מתבצע אוטומטית אחרי כל פקודה, “קפיצה” לרוטינה או חזרה מרוטינה, ופקודות JMP (אבל לא ניתן לשנות ישרות)

הסגןטים השונים

- **מקטע המחסנית:**
- מכיל "מחסנית" עבור נתונים שונים נשמרים בעת פעילות תקינה של התוכנה (נושא המחסנית יוסבר בעוד מספר שבועות)
- ראש המחסנית נמצא בכתובת SS:SP .
אוגר היסט הקשור לSS BP הוא.

הסגמנטים השונים

- **מקטע הנתוניים:**
 - מכיל נתונים, מערכיים, קבועים ומשתנים, שהמוכנה משתמש כחלק מהתוכנה
 - אוגר BX משמש כהיסט למקטע זה, DS:BX
 - האוגרים AI,DI משמשים כאינדקס למקטע זה (לעתים יחד עם BX)
- **מקטע ה"אקסטרה":**
 - דומה למקטע הנתוניים, מאפשר גמישות נוספת בתוכנה

מקטע בירית מחדל

- אם קוד האסמבלר לא מצין סגמנט מפורשות, הסגמנט נקבע לפי בירית מחדל:

מצבי	SEGMENT BY READING MHDL
IP	CS
BP, SP	SS
BX, SI, DI <i>(כתובת מספרית)</i>	DS

- למשל
- | | | |
|----------------|----|-------------------|
| MOV [BX], CX | -> | MOV DS:[BX], CX |
| MOV AX, [BP] | -> | MOV AX, SS:[BP] |
| MOV [100h], AX | -> | MOV DS:[100h], AX |

כפיפות מקטע

- עבור המצביעים: BX, SP, BP ו- SI, DI וכותבת מספרית
ניתן “לכפות” את הsegment בו יעשה שימוש
- למשל

MOV ES:[BX], CX

MOV SS:[1000h], AX

MOV CS:[DI], AX

סיכום אוגרים ותפקיד

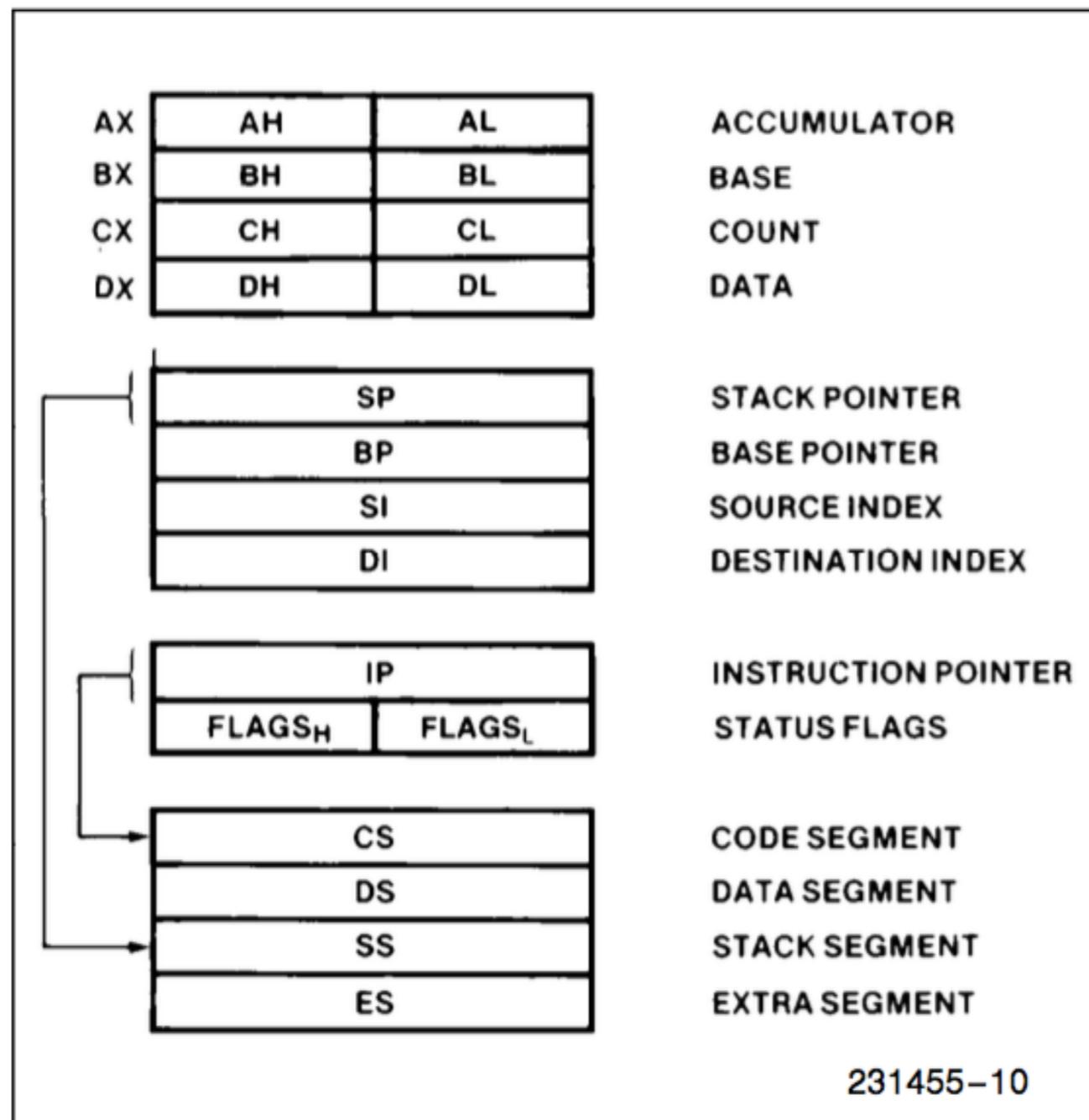


Figure 7. 8086 Register Model

מבוא לשפת סר (אסםבלר)

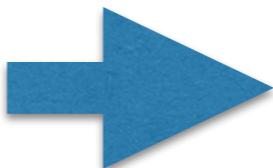
מ-ל C-ל ASM

```
/* TEST.C -- simple prog */

main()
{
    int i;
    for (i = 1; i < 5 ; i++){
        /* Do something */
        f();
    }

int f()
{
    return 0;
}
```

מהדר



01FD	PUSH	SI	i = 1
01FE	MOV	SI,0001	
0201	JMP	0207	
0203	CALL	020F	call f()
0206	INC	SI	i++
0207	CMP	SI,+05	i < 5
020A	JL	0203	
020C	POP	SI	
020D	POP	BP	
020E	RET		
020F	PUSH	BP	
0210	MOV	BP,SP	
0212	XOR	AX,AX	
0214	JMP	0216	f()
0216	POP	BP	
0217	RET		

מבנה תוכנית asm פשוטה

```
.model small
```

```
.data
```

```
INFO DW 1000h
```

תחילת מקטע נתונים ;

הגדרת משתנה בשם INFO ;

```
.stack 100
```

הקצת 100 בתים למחסנית ;

```
.code
```

תחילת מקטע קוד ;

```
START:
```

```
MOV AX, 0
```

שים 0 באוגר AX ;

```
MOV BX, INFO
```

שים את הערך של המשתנה "AINFO" באוגר BX ;

```
MOV DS, AX
```

העתק את אוגר AX לאוגר DS ;

```
MOV AH, 4CH
```

סיום התוכנה (חזרה למ"ה) ;

```
INT 21h
```

```
END START
```

הגדרת המקטעים

- שיטה ב':

SSEG **SEGMENT STACK**

... <def of stack>

SSEG **ENDS**

DSEG **SEGMENT**

... <def of constants>

DSEG **ENDS**

CSEG **SEGMENT**

ASSUME CS:CSEG, DS:DSEG, SS:SSEG

... <instructions>

CSEG **ENDS**

- שיטה א':

.stack <size>

.data

... <def constants>

.code

... <instructions>

הגדרת משתנים

- ניתן לתת "שם" לאיזור זיכרונו בגודל בית או מילה, ולבזוע ערך ראשוני שייטען לזכרונו עם טעינת התוכנה
- הגדרת משתנה בגודל בית ע"י **DB** (Define Byte)
הגדרת משתנה בגודל מילה ע"י **DW** (Define Word)

<name>	DB	<val 8bit>
<name>	DW	<val 16bit>

MASK	DB	08h
LENGTH	DW	1500

- **לדוגמה**

הגדות המשתנים

- ערד התחלה? יגדיר את הזיכרון ללא איתחולו

TEMP DB ?

- ניתן גם לתת שם לאזר זיכרונו גדול יותר ("מרקם"):

placeXYZ DB 0, 150, -34

- ניתן לאთחל מספר גדול של תאים ע"י ההנחיה DUP

GRADES DB 50h DUP (100)

יגדר 80 בתים מאוחלים ל 64ה

TEMP3 DW 50h DUP (?)

**יגדייר 80 מילים
לא איתחול**

הגדרת קבועים

- ניתן להגדיר שם קבוע ע"י הirection **EQU**
- לדוגמה:

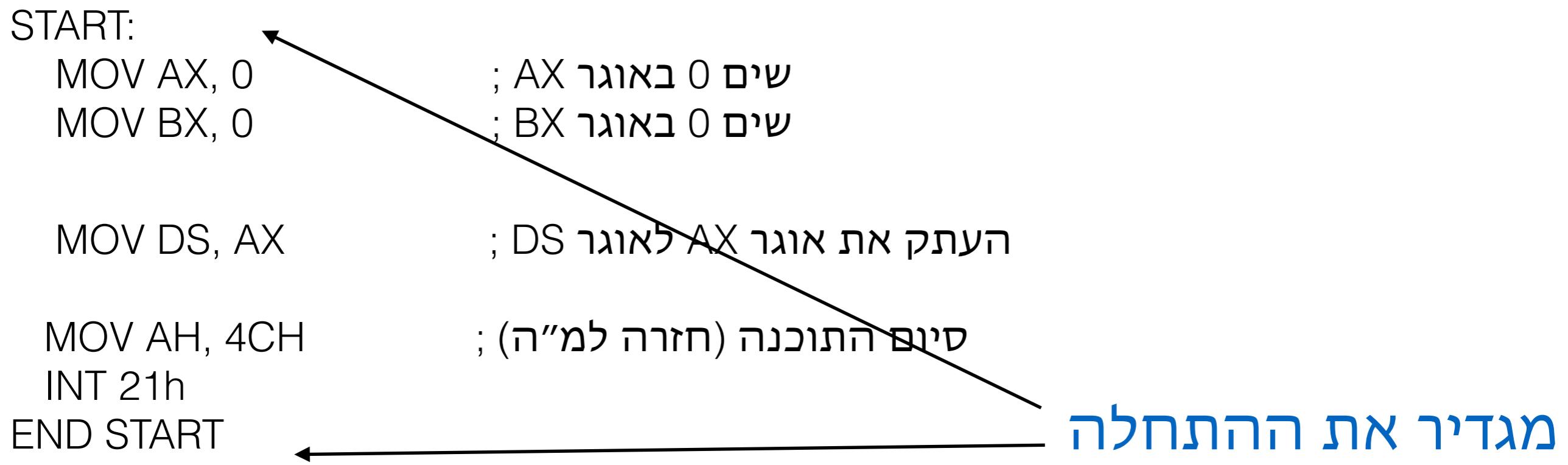
zero **EQU** 0

המהדר יחליף כל מופע של zero בערך 0

MOV AX, zero \iff MOV AX, 0

התחלת וסיום התוכנית

- התוכנית מסתיימת עם `END <LABEL>`
- התוכנית תתחיל לזרז מהפקודה הראשונה של `<LABEL>`



התחלת וסיום התוכנית

- להפסקת הריצה (וחזרה למ"ה) יש לכתוב פקודה מיוחדת

START:

MOV AX, 0 ; Shim 0 baogur AX
MOV BX, 0 ; Shim 0 baogur BX

MOV DS, AX ; העתק את אוגר AX לאוגר DS

MOV AH, 4CH ; סיום התוכנה (חזרה למ"ה)
INT 21h
END START

הנחיה מקוצרת:



.EXIT

התחלת וסיום התוכנית

- מה יקרה בסוף התוכנית אם לא נכתב פקודה כנ"ל?

התחלה וסיום התוכנית

.model small

.data
INFO DW 1000h ;
תחילת מקטע נתונים ;
הגדרת משתנה בשם INFO ;

.stack 100h ;
הказאת 100 בתים למחסנית ;

.code
תחילת מקטע קוד ;

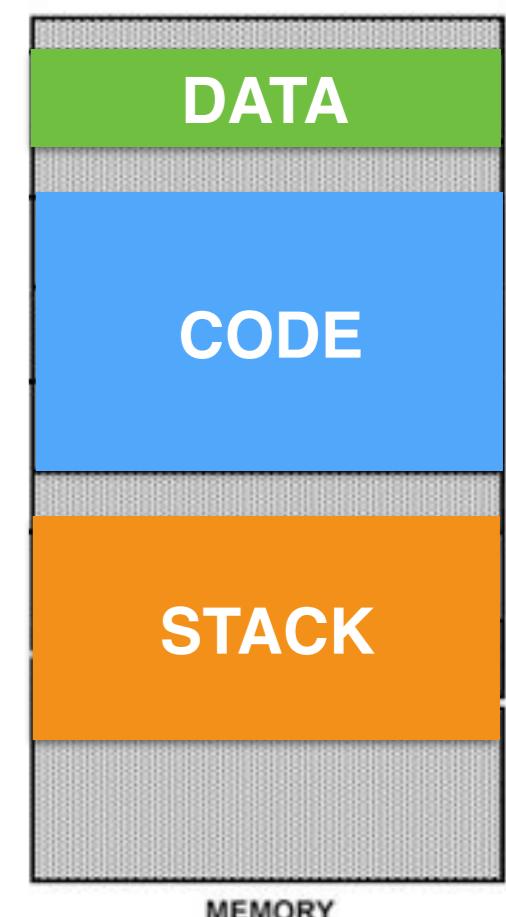
START:

MOV AX, 0 ; AX
MOV BX, 0 ; BX

MOV DS, AX ; DS
העתק את אוגר AX לאוגר DS ;

MOV AH, 4CH ;
INT 21h
סיום התוכנה (חזרה למ"ה) ;

END START



אורים: מקורות

- [http://www.yecd.com/os/
8086%20assembler%20tutorial%20for%20beginners%20\(part%201\).htm](http://www.yecd.com/os/8086%20assembler%20tutorial%20for%20beginners%20(part%201).htm)
- [http://www.cosc.brocku.ca/~bockusd/3p92/Local_Pages/
8086_architecture.htm](http://www.cosc.brocku.ca/~bockusd/3p92/Local_Pages/8086_architecture.htm)
- מחשבים ומיקרו מעבדים, מט"ח, האוניברסיטה הפתוחה

שיטת מייעון ומבנה הפקודה

מיקרומעבדים ושות אסמלר 83-255-83



סוגי פקודות

- **פקודות העברת מידע**
MOV, IN/OUT, PUSH, POP
 - **פקודות חישוב אריתמטיות/לוגיות**
ADD,SUB,MUL,AND,OR,XOR,...
 - **פקודות בקרת תוכנה ומעבד**
JMP, JZ,JNZ,..., CALL, RET, LOOP, STI, CLI,..
 - **פקודות על מחרוזת**
MOVS, LODS, STOS, REP
-
- (לא נלמד)

מבנה פקודת עקרוני

- פקודה יכולה להכיל 0, 1 או 2 אופרנדים

COMMAND

WAIT

COMMAND

operand

INC AX

COMMAND

operand1, operand2

ADD AX, BX

- כאשר יש שני אופרנדים, הראשון הינו היעד והשני המקור

COMMAND

op-DEST, op-SRC

הפקודה MOV

- פקודת mov "מיזה" (מעתיקה) מידע בין מקום למקום (אוגר, תא זיכרון, ...)
- מבנה הפקודה:


```
MOV  <dest>, <src>
```
- המידע שבמקור "مועתק" אל היעד.
המקור לא משתנה

הפקודה MOV

- דוגמא:

MOV AX, BX

יעתיק את המידע שבאוגר BX לטור AX

AX = 0000h
BX = 1212h

MOV AX, BX



AX = **1212h**
BX = 1212h

הפקודה MOV

- דוגמא:

MOV AL, AH

יעתיק את המידע שבאוגר AH לתוכן AL

AX = 00B1h
BX = 1212h

MOV AL, AH



AX = 0000h
BX = 1212h

הפקודה MOV

- דוגמא:

MOV [0000h], AX

עתיק את המידע שבאוגר AX לטור תא זיכרון 0000h
(בסגנון הנתונים)

AX = 0000h

BX = 1212h

DS = 1000h

MOV [0000h], AX

AX = 0000h

BX = 1212h

DS = 1000h



01	02	03
----	----	----

10000h 10001h 10002h

00	00	03
----	----	----

10000h 10001h 10002h

הפקודה MOV

- דוגמא:

MOV AX, [0000h]

AX = 0000h
BX = 1212h
DS = 1000h



AX = **0201h**
BX = 1212h
DS = 1000h

01	02	03
----	----	----

10000h 10001h 10002h

01	02	03
----	----	----

10000h 10001h 10002h

פקודת MOV

- נדרש כי המקור והיעד יהיה **תמיד** מאותו הגודל
- או שניהם 16 ביט, או שניהם 8 ביט

MOV AX, DX
MOV AL, AH



חוקי 16 ביט
חוקי 8 ביט

MOV AL, DS
MOV AX, BL



פקודה לא חוקית

סוגי מיעון

- mov register, register
- mov register, constant
- mov register, memory
- mov memory, register
- mov memory, constant

~~mov memory, memory~~

מייעו אוגר

mov register, register

- מעתיק מידע בין אוגרים באותו גודל

MOV AX, DX 16 bit

MOV BL, CL 8 bit

MOV SI,DS 16 bit - any general purpose reg

- לא ניתן להעביר בין שני אוגרי סגמנט

~~MOV DS, ES~~

- לא לשנות את CS ישירות

MOV CS, AX **בעייתן!**

מייעו מיד!

mov register, constant

- מעתיק מידע נתון (קבוע) אל אוצר

MOV AX, 0011h

MOV BL, 20h

MOV BL, 20

MOV SI,1100h

לשים לב, !!! $14_{16} = 20_{10}$

- לא ניתן להעביר קבוע אל אוצר סגמנט

~~MOV DS, 0000h~~

- במקום:

MOV AX, 0000h

MOV DS, AX

מייעו מיידי

- אין משמעות להעתיק אוגר קבוע

~~MOV 20h, AL~~

~~MOV 0000h, BX~~

מייענו ישיר (לזיכרונו)

mov register, memory

mov memory, register

- מעתיק תוכן אוגר לכתובת בזיכרון, או להיפך

MOV AX, DS:[0011h]

MOV BL, DS:[0013h]



AX ← 0201h
BL ← 03h

מייענו ישיר

mov register, memory

mov memory, register

- מעתיק תוכן אוגר לכתובת בזיכרון, או להיפך

MOV DS:[0010h], AL

MOV DS:[0010h], CX



AX = 0201

CX = 0304

AX = 0201h

CX = 0304h

מייעו רישיר

mov register, memory

mov memory, register

- כתובת בזיכרון יכולה להופיע גם בצורה סימבולית
- המהדר (קומפיאילר) מחליף את התוויות בכתובת המתאימה

מייענו ישיר

mov register, memory

mov memory, register

- **שימו לב!** אין העתקה מזיכרון לזמן

~~mov memory, memory~~

- להלופין?

MOV AX, DS:[FF21h]

MOV DS:[0000h], AX

- העתקה דרך אוגר

מייענו עקייף (לזיכרונו)

mov register, memory
 mov memory, register



mov register,[register (ptr)]
 mov [register (ptr)], register

- מעתיק תוכנו אוגר לכתובת בזיכרון המיוצגת על-ידי אוגר אחר (או להיפך)

MOV AX, DS:[BX]
 MOV BL, DS:[BX]



AX \leftarrow 0201h
 BL \leftarrow 01h

DS=0000, BX=0011h

מייעו עקייף (לזיכרונו)

mov register,[register (ptr)]

mov [register (ptr)], register

MOV [BX], AX

MOV [BX], AH

00	01	02	03
00010h	00011h	00012h	00013h



00	7A	7A	03
00010h	00011h	00012h	00013h

DS:BX=00011h

AX=7A33h

מייעו עקייף (לזיכרונו)

mov register,[register (ptr)]

mov [register (ptr)], register

- ניתנו להשתמש **رك** באוגרים:
 - AX או BP (מצביעים)
 - SI או DI (אינדקסים)
 - תזכורת: BP מצביע למקטע מחסנית (SS),
AX למקטע נתונים (.DS).
 - ניתנו לכפות סגמנט אחר:
- MOV ES:[BX], AX
MOV ES:[1000h], AX

מיומו עקיף - יחסית

mov register, memory

mov memory, register

- ניתן ליצג כתובות בזיכרון ע"י אוגר בסיס (BX או BP) + הسطה של 16 ביט

MOV [BX+4], AX

MOV DX, [BP-05h]

מייעו עקייף - יחסוי

mov register, memory

mov memory, register

DS=0010h

SS=1000h

BX=0000h

BP=0000h

AX=7A33h

מה מבצעת הפקודה?

MOV [BX+4], AX

תשובה:

DS:[BX+4] \leftarrow AX

[0010:0004] \leftarrow AX

מייעו עקייף - יחסוי

mov register, memory

mov memory, register

DS=0010h

SS=1000h

BX=0000h

BP=0000h

AX=7A33h

מה מבצעת הפקודה?

MOV DX, [BP-05h]

תשובה:

DX \leftarrow SS:[BP-05h]

DX \leftarrow [1000:FFF8h]

מייעו עקייף - יחסית/אינדקס

mov register, memory

mov memory, register

- ניתן לייצג כתובות בזיכרון גם ע"י

אוגר בסיס (BX או BP)

+ אוגר אינדקס (DI או SI)

הסטה של 16 ביט

MOV [BX+DI+4], AX

MOV DX, [BP+DI-5Fh]

- באופן כללי, ניתן להשמית בסיס או אינדקס או היסט

MOV [DI-40h], AX

MOV DX, [BP+SI]

מייעו עקייף - יחסית/אינדקס

mov register, memory

mov memory, register

- ניתן לייצג כתובות בזיכרון גם ע"י

אוגר בסיס (BX או BP)

+ אוגר אינדקס (DI או SI)

הסטה של 16 ביט

- **לא ניתן** להשתמש בשני בסיסים או בשני אינדקסים

~~MOV [BP+BX]~~

~~MOV [DI+SI]~~

מייעו עקיף - מיידי

mov **memory**, **constant**

- מעתיק מספר "קבוע" אל זיכרון (המיוצג ע"י בסיס/אינדקס/היסט)
- דוגמא:

MOV DS:[1000h], 0005h

MOV [BX+SI], 1122h

MOV [DI-2FFh], A4B5h

מייעו עקיף - מיידי

mov memory, constant

- מה הערך בפקודה הבאה?

MOV DS:[1000h], 05h

- הפקודה זו משמעית: האם הזיכרון הוא בית או מילה?

MOV DS:[1000h], 05h \longleftrightarrow MOV DS:[1000h], 0005h

- במיון זה נדרש להנחות את המהדר בצורה מפורשת:

(AX = 0005h)

MOV BYTE PTR [1000h], 5h

MOV WORD PTR [1000h], 5h



MOV [1000h], AL

MOV [1000h], AX

שיטות מייעו - סיכום

<u>יעד</u>	<u>חישוב כתובות</u>	<u>מקור</u>	<u>דוגמא</u>	
אוגר AX	$\xleftarrow{\hspace{1cm}}$	אוגר BX	MOV AX,BX	מייעו אוגר
אוגר CL	$\xleftarrow{\hspace{1cm}}$	קבוע 05	MOV CL, 05H	מיידי
זיכרון 11000h	$\xleftarrow{\hspace{1cm}}$ $DS \times 10h + 1000h$	אוגר AX	MOV DS:[1000h], AX	ישיר
זיכרון 10500h	$\xleftarrow{\hspace{1cm}}$ $DS \times 10h + BX$	אוגר CX	MOV [BX], CX	עקייר
זיכרון 105B0h	$\xleftarrow{\hspace{1cm}}$ $DS \times 10h + BX + SI$	אוגר DL	MOV [BX+SI], DL	עקייר+אינדקס
זיכרון 10400h	$\xleftarrow{\hspace{1cm}}$ $DS \times 10h + BX - 100h$	אוגר BX	MOV [BX-100h], BX	עקייר+היסט
זיכרון 105B2h	$\xleftarrow{\hspace{1cm}}$ $DS \times 10h + BX + SI + 2$	אוגר AH	MOV [BX+SI+2], AH	עקייר+אינדקס+ היסט

DS=1000h, BX=0500h, SI=00B0h

חוקי או לא?

- ✓ MOV AX, BX
 - ✗ MOV CL, DX mixing 8/16bit
 - ✓ MOV BP, SP
 - ✗ MOV DS, CS segment to segment
 - ✓ MOV ES:[BX], AL
 - ✗ MOV CS:[IP], DX can't use IP as base/index
 - ✗ MOV DS:[BP], 05h BYTE PTR / WORD PTR

מבנה הפקודה

היררכיה של שפות



שפת מכוна

- המהדר (compiler) מתרגם כל "פקודה" בשפת אסמבלי לרצף של פקודות מכונה
- פקודות המכונה נמצאות בזיכרון ונקראות ע"י המעבד
- כל פקודת-מכונה מתורגםת לפעולה אחת של המעבד

(שפה סר)

MOV AX, BX

(instruction)



8BC3

מבנה פקודת ב-8086

- ב-8086 קידוד כל פקודה מכיל בין בית אחד לשישה בתים
- כל פקודה מתחילה ב-OPCODE
- בית יחיד (בדרכ-כלל)
- מגדר את **סוג** הפקודה (mov, add, jmp)
- קיימים OPCODE-ים מיוחדים שתפקידם להשפיע על הפקודה שלאחריהם (פעולה זו יוצרת לכaura "PCODE" מורחב באורד 2 בתים)
- לעיתים ניתן לקוד את אותה פעולה במספר דרכים שונות בשפת מכונה

מבנה פקודת

(שפה סג')

MOV AX, BX

(instruction)

8B C3

opcode: use registers:
MOV 16b AX, BX

MOV AX, ES:[BX+SI+1122h]

26 8B 80 22 11

opcode: opcode:
ES mov 16

offset

use register AX
and indirect
addressing
BX, SI, offset

Table 2. Instruction Set Summary

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010 dw	mod reg r/m		
Immediate to Register/Memory	1100011 w	mod 000 r/m	data	data if w = 1
Immediate to Register	1011 w reg		data	data if w = 1
Memory to Accumulator	1010000 w		addr-low	addr-high
Accumulator to Memory	1010001 w		addr-low	addr-high
Register/Memory to Segment Register	10001110	mod 0 reg r/m		
Segment Register to Register/Memory	10001100	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	11111111	mod 110 r/m		
Register	01010 reg			
Segment Register	000 reg 110			
POP = Pop:				
Register/Memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment Register	000 reg 111			
XCHG = Exchange:				
Register/Memory with Register	1000011 w	mod reg r/m		
Register with Accumulator	10010 reg			
IN = Input from:				
Fixed Port	1110010 w	port		
Variable Port	1110110 w			
OUT = Output to:				
Fixed Port	1110011 w	port		
Variable Port	1110111 w			
XLAT = Translate Byte to AL	11010111			
LEA = Load EA to Register	10001101	mod reg r/m		
LDS = Load Pointer to DS	11000101	mod reg r/m		
LES = Load Pointer to ES	11000100	mod reg r/m		
LAHF = Load AH with Flags	10011111			
SAHF = Store AH into Flags	10011110			
PUSHF = Push Flags	10011100			
POPF = Pop Flags	10011101			

- **המפרט הטכני של המעבד מכיל את קידודי הפקודות הנתמכות ע"י המעבד**

מבנה פקודת נפוץ

MOV = Move:

Register/Memory to/from Register

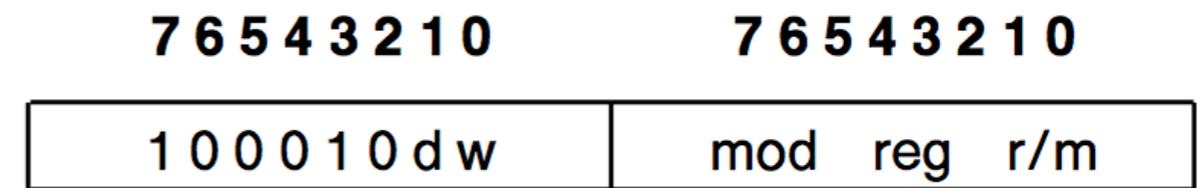


FIGURE 4–2 Byte 1 of many machine language instructions, showing the position of the D- and W-bits.

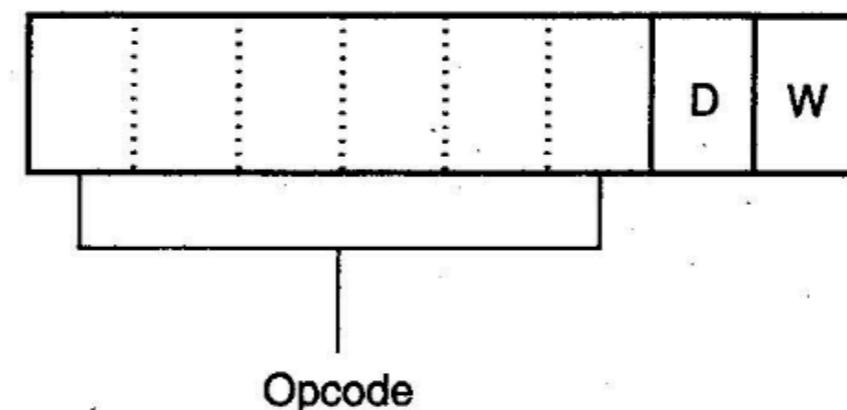
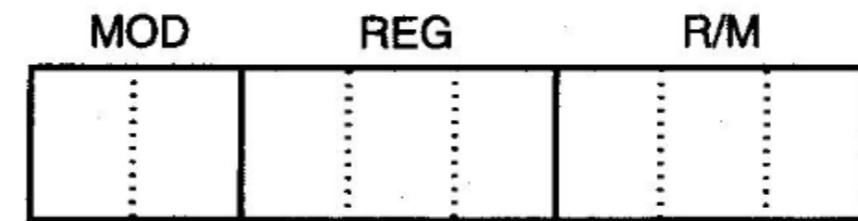


FIGURE 4–3 Byte 2 of many machine language instructions, showing the position of the MOD, REG, and R/M fields.



מבנה פקודת נפוץ

MOV = Move:

Register/Memory to/from Register

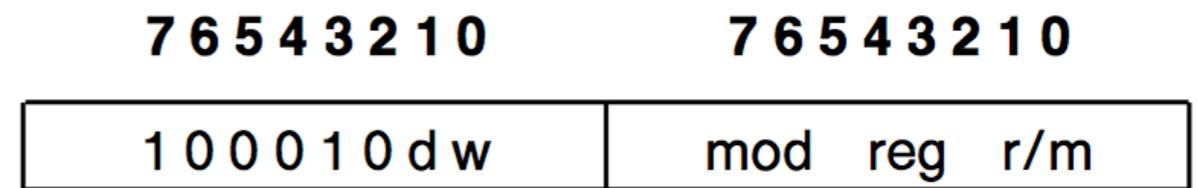


FIGURE 4–2 Byte 1 of many machine language instructions, showing the position of the D- and W-bits.

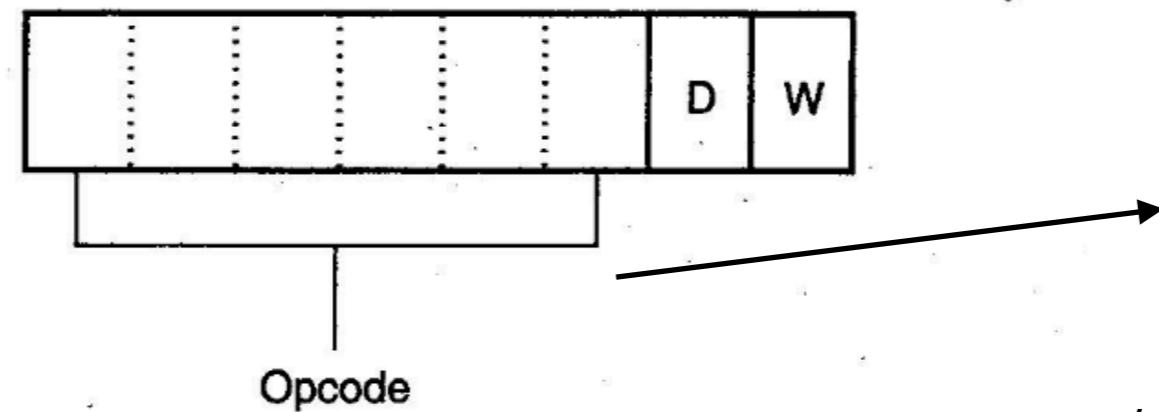
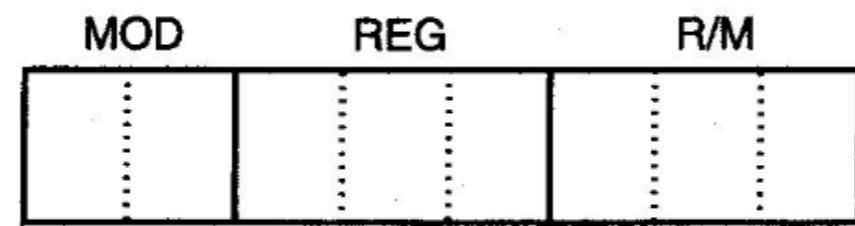


FIGURE 4–3 Byte 2 of many machine language instructions, showing the position of the MOD, REG, and R/M fields.



הopcode קובע את סוג הפקודה
למשל: העתק אוגר/זיכרון 100010
חיבור אוגר/זיכרון 000000

מבנה פקודת נפוץ

MOV = Move:

Register/Memory to/from Register

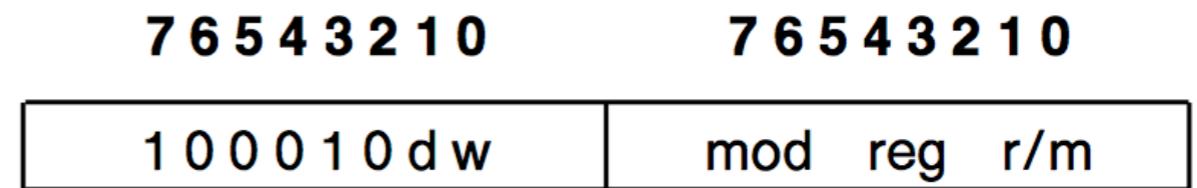
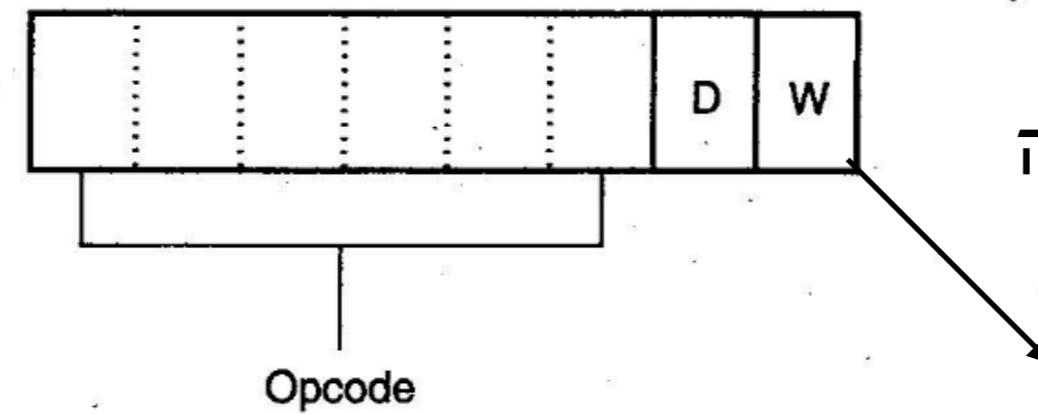


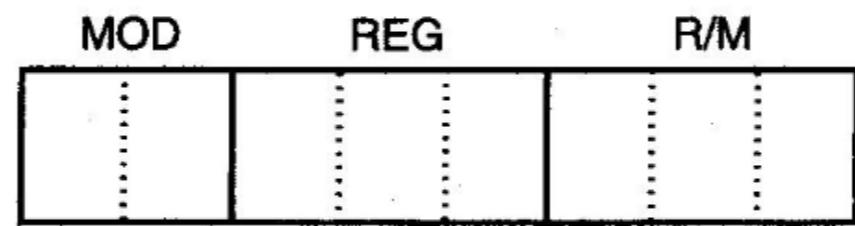
FIGURE 4–2 Byte 1 of many machine language instructions, showing the position of the D- and W-bits.



הビיט W קובע האם פעולה על בית או מילה

w=0 8bit
 w=1 16bit

FIGURE 4–3 Byte 2 of many machine language instructions, showing the position of the MOD, REG, and R/M fields.



מבנה פקודת נפוץ

MOV = Move:

Register/Memory to/from Register

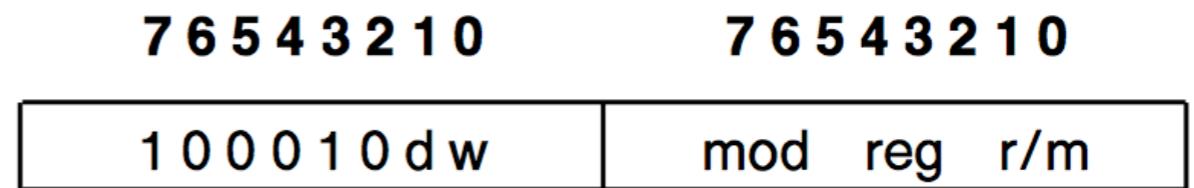


FIGURE 4–2 Byte 1 of many machine language instructions, showing the position of the D- and W-bits.

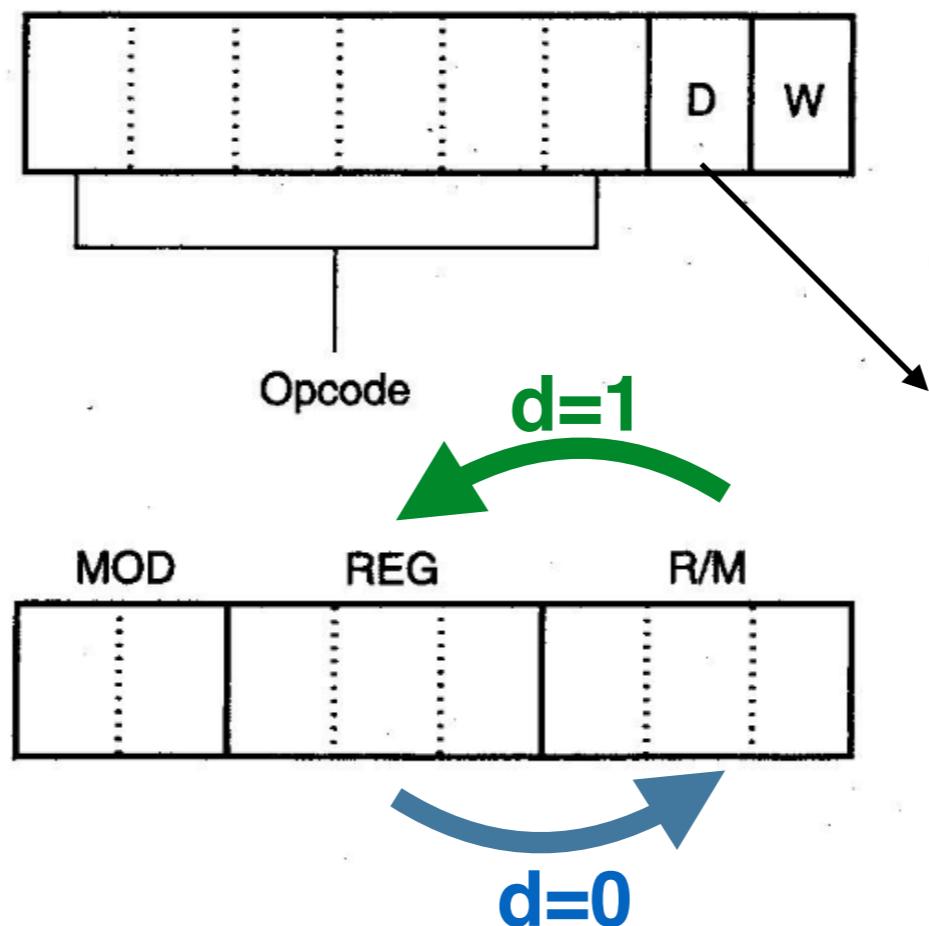


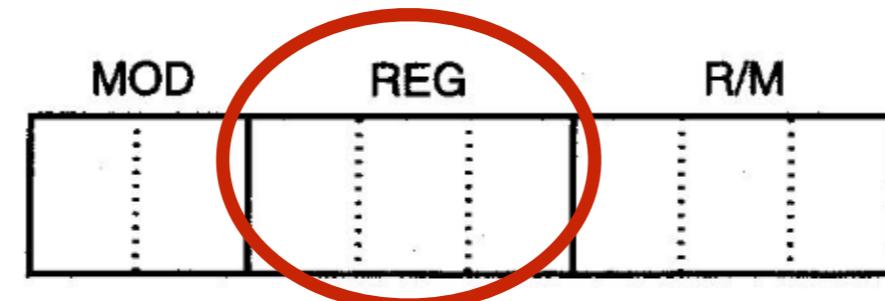
FIGURE 4–3 Byte 2 of many machine language instructions, showing the position of the MOD, REG, and R/M fields.

הביט d קובע כיוון
העברת הנתונים

בש **d=0**:
משדה reg לשדה r/m

בש **d=1**:
משדה r/m לשדה reg

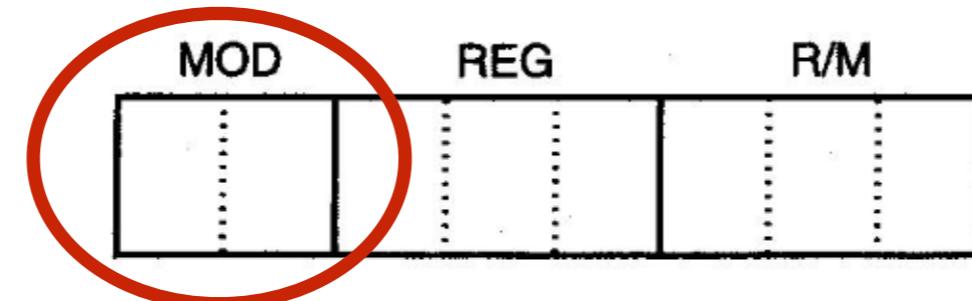
מבנה שדה REG



שדה REG מציין אוגר מסוים:

Code	W = 0 (Byte)	W = 1 (Word)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

מבנה שדה mod/reg/rm



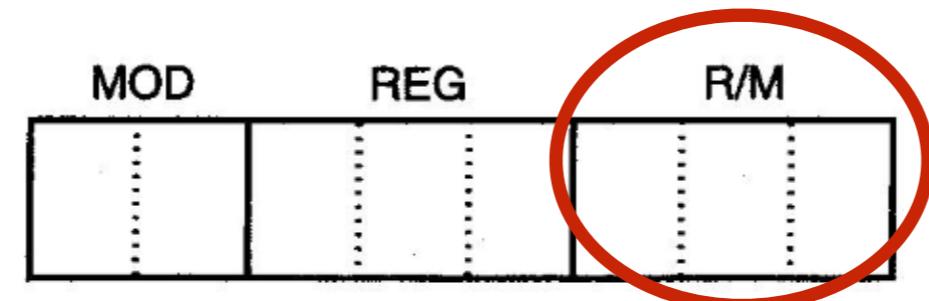
- שדה m/r יכול או לציין אוגר (reg), או לציין שיטת מיעון לזכורו (mem).
- תפקיד השדה m/r נקבע ע"י השדה mod

<i>MOD</i>	<i>Function</i>
00	No displacement
01	8-bit sign-extended displacement
10	16-bit signed displacement
11	R/M is a register

m/r הוא זיכרון

m/r הוא אוגר

מבנה שדה rm



- אם m/r מצין אוגר, האוגר הגרפי נקבע לפי טבלת reg

<i>R/M Code</i>	<i>Addressing Mode</i>
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]*
111	DS:[BX]

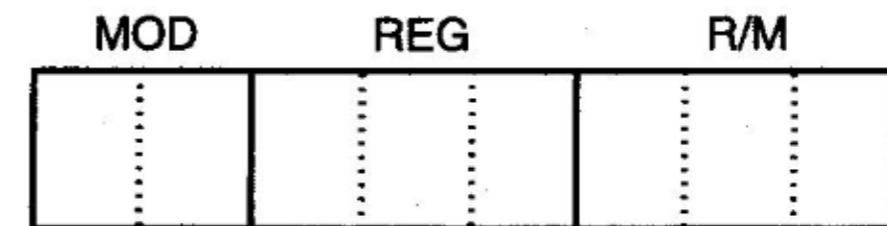
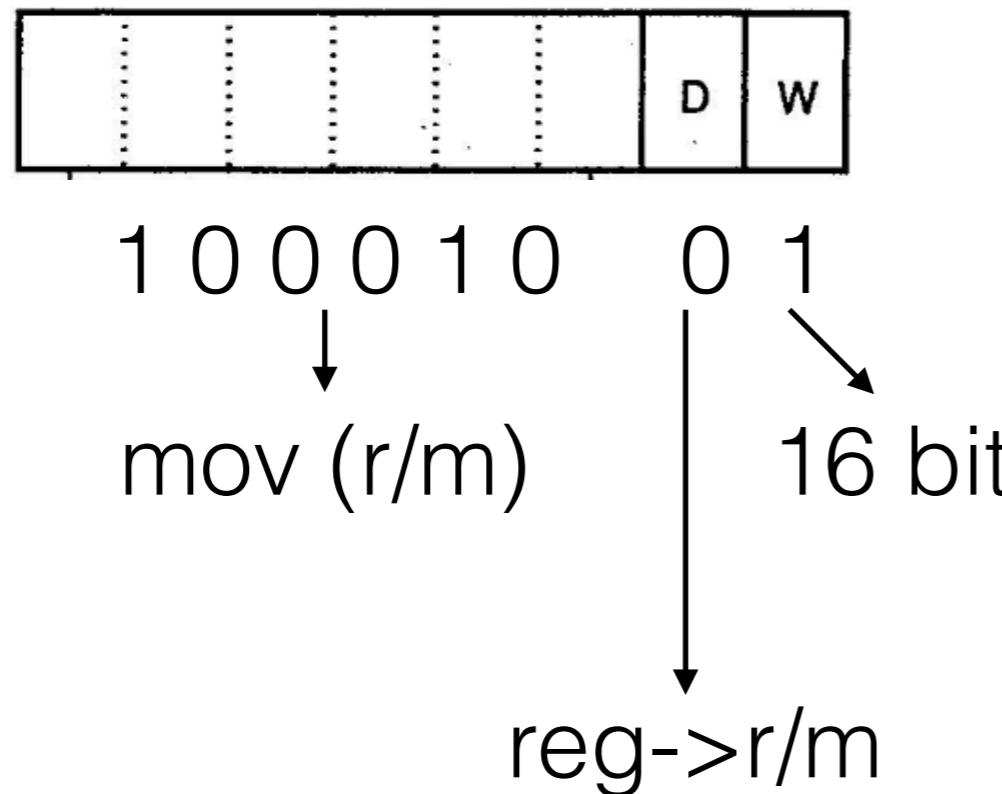


*Note: See text section, Special Addressing Mode.

- אם מצין מיעון לזכרון:
 - בשילוב שדה mod נקבע היחסט (אם קיים)
- מקרה מיוחד:**
תמיד מכיל היחסט!
(mod 00) יסמל מיעון ישיר

מייענו אוגר

MOV AX, BX 89 D8



MOD	Function
00	No displacement
01	8-bit sign-extended displacement
10	16-bit signed displacement
11	R/M is a register

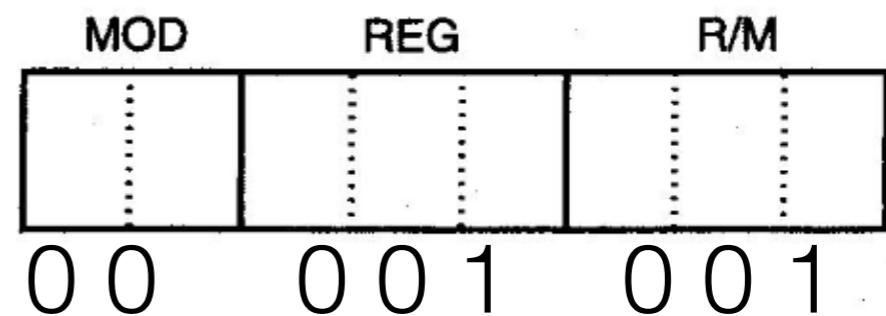
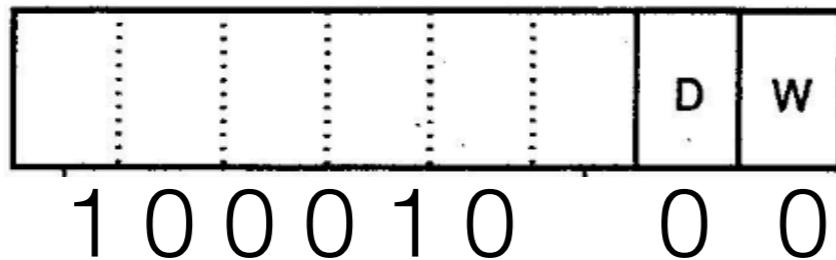
Code	W = 0 (Byte)	W = 1 (Word)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

מה פקודה המכונה של
 ? MOV BX,AX

מייעו עקייף

MOV [BX+DI], CL

מה משמעות הפקודה ? 8809



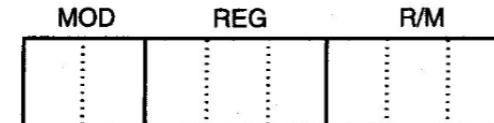
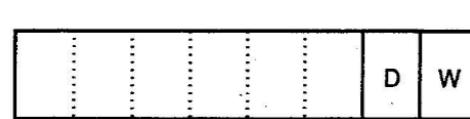
<i>R/M Code</i>	<i>Addressing Mode</i>
000	DS:[BX+SI]
001	DS:[BX+DI] (circled)
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]*
111	DS:[BX]

*Note: See text section, Special Addressing Mode.

<i>MOD</i>	<i>Function</i>
00	No displacement
01	8-bit sign-extended displacement
10	16-bit signed displacement
11	R/M is a register

<i>Code</i>	<i>W = 0 (Byte)</i>	<i>W = 1 (Word)</i>
000	AL	AX
001	CL (circled)	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

מיומו עקיף + היסט



<i>MOD</i>	<i>Function</i>
00	No displacement
01	8-bit sign-extended displacement
10	16-bit signed displacement
11	R/M is a register

- אם קיים היסט (מספרי), ערך היסט יופיע מיד לאחר שדה mod/reg/rm

88 09

ראיינו CL MOV [BX+DI], CL מקודד ב

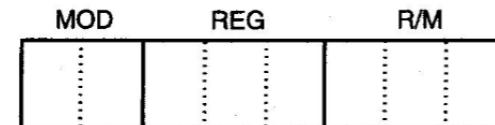
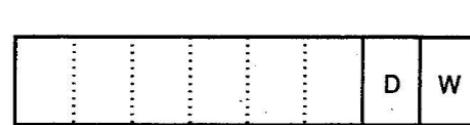
88 49 33

יקוד MOV [BX+DI+33h], CL •

88 89 33 10

יקוד MOV [BX+DI+1033h], CL •

מייעו עקייף + היסט



MOD	Function
00	No displacement
01	8-bit sign-extended displacement
10	16-bit signed displacement
11	R/M is a register

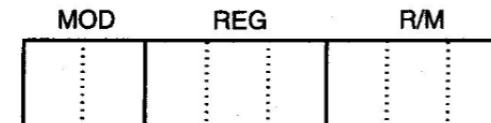
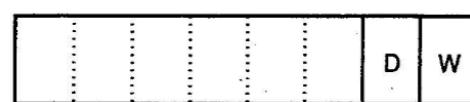
- מקרה מיוחד: עברו []
תמיד יש היסט

R/M Code	Addressing Mode
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]*
111	DS:[BX]

*Note: See text section, Special Addressing Mode.

- הפקודה MOV [BP],AX
תקודד ב MOV [BP+00],AX
שפת מכונה: 89 46 00

מייעון ישיר



Code W = 0 (Byte) W = 1 (Word)

000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

R/M Code Addressing Mode

000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]*
111	DS:[BX]

*Note: See text section, Special Addressing Mode.

MOD	Function
00	No displacement
01	8-bit sign-extended displacement
10	16-bit signed displacement
11	R/M is a register

- מייעון ישיר מקודד ע"י

MOD=00

בצירוף עם r/m=110

- למשל: MOV [1000h], DX
מקודד 89 16 00 10

מייעו מיידי

MOV = Move:

Register/Memory to/from Register

7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

Immediate to Register/Memory

1 0 0 0 1 0 d w	mod reg r/m		
1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
1 0 1 1 w reg	data	data if w = 1	

Immediate to Register

Memory to Accumulator

Accumulator to Memory

Register/Memory to Segment Register

Segment Register to Register/Memory

1 0 1 0 0 0 0 w	addr-low	addr-high
1 0 1 0 0 0 1 w	addr-low	addr-high
1 0 0 0 1 1 0	mod 0 reg r/m	
1 0 0 0 1 1 0 0	mod 0 reg r/m	

אם קיים היסט (מייעון עקייף+היסט)
 יופיע כאו - לפני המיידע

מייעו מיידי

Immediate to Register

1 0 1 1 w reg	data	data if w = 1
---------------	------	---------------

MOV DL, 05h

1011 0 010 0000 0101
B2 05

MOV DX, 1005h

1011 **1** 010 0000 0101 0001 0000
BA 05 10

Code	W = 0 (Byte)	W = 1 (Word)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

מייענו מיידי/עקייף

Immediate to Register/Memory

1100011w	mod 000 r/m	data	data if w = 1
----------	-------------	------	---------------

MOV BYTE PTR [SI], 05h C6 04 05

MOV WORD PTR [BP], 05h C7 46 05 00

00

היסט!!

R/M Code	Addressing Mode	MOD	Function	Code	W = 0 (Byte)	W = 1 (Word)
000	DS:[BX+SI]	00	No displacement	000	AL	AX
001	DS:[BX+DI]	01	8-bit sign-extended displacement	001	CL	CX
010	SS:[BP+SI]	10	16-bit signed displacement	010	DL	DX
011	SS:[BP+DI]	11	R/M is a register	011	BL	BX
100	DS:[SI]			100	AH	SP
101	DS:[DI]			101	CH	BP
110	SS:[BP]*			110	DH	SI
111	DS:[BX]			111	BH	DI

כפיהת סגמנט

- הוספת opcode מיוחד **לפני** פקודת MOV (וכו') יגרום לכפיהת סגמנט עבר שדה r/m.

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

- למשל

MOV [BX],AX
=> 89 07

MOV **ES**:[BX], AX
=> **26** 89 07

מייעו ישר ל-AX

MOV = Move:

Register/Memory to/from Register

7 6 5 4 3 2 1 0

1 0 0 0 1 0 d w	mod reg r/m
-----------------	-------------

Immediate to Register/Memory

7 6 5 4 3 2 1 0

1 1 0 0 0 1 1 w	mod 0 0 0 r/m
-----------------	---------------

7 6 5 4 3 2 1 0

data	data if w = 1
------	---------------

7 6 5 4 3 2 1 0

Immediate to Register

1 0 1 1 w reg	data
---------------	------

data if w = 1	
---------------	--

Memory to Accumulator

1 0 1 0 0 0 0 w	addr-low
-----------------	----------

addr-high	
-----------	--

Accumulator to Memory

1 0 1 0 0 0 1 w	addr-low
-----------------	----------

addr-high	
-----------	--

Register/Memory to Segment Register

1 0 0 0 1 1 1 0	mod 0 reg r/m
-----------------	---------------

Segment Register to Register/Memory

1 0 0 0 1 1 0 0	mod 0 reg r/m
-----------------	---------------

העברה עם אוגר סגמנט

MOV = Move:

Register/Memory to/from Register

7 6 5 4 3 2 1 0

1 0 0 0 1 0 d w

7 6 5 4 3 2 1 0

mod reg r/m

Immediate to Register/Memory

1 1 0 0 0 1 1 w

mod 0 0 0 r/m

7 6 5 4 3 2 1 0

data

7 6 5 4 3 2 1 0

data if w = 1

Immediate to Register

1 0 1 1 w reg

data

data if w = 1

Memory to Accumulator

1 0 1 0 0 0 0 w

addr-low

addr-high

Accumulator to Memory

1 0 1 0 0 0 1 w

addr-low

addr-high

Register/Memory to Segment Register

1 0 0 0 1 1 1 0

mod 0 reg r/m

Segment Register to Register/Memory

1 0 0 0 1 1 0 0

mod 0 reg r/m

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

אורים: מקורות

- B. Brey, The Intel Microprocessor, 2009
- Intel 8086 spec, INTEL, 1979

פקודות אריתמטיות לוגיות, ופקודות בקרה

מיקромעבדים ושפת אSEMBLER 83-255

סוגי פקודות

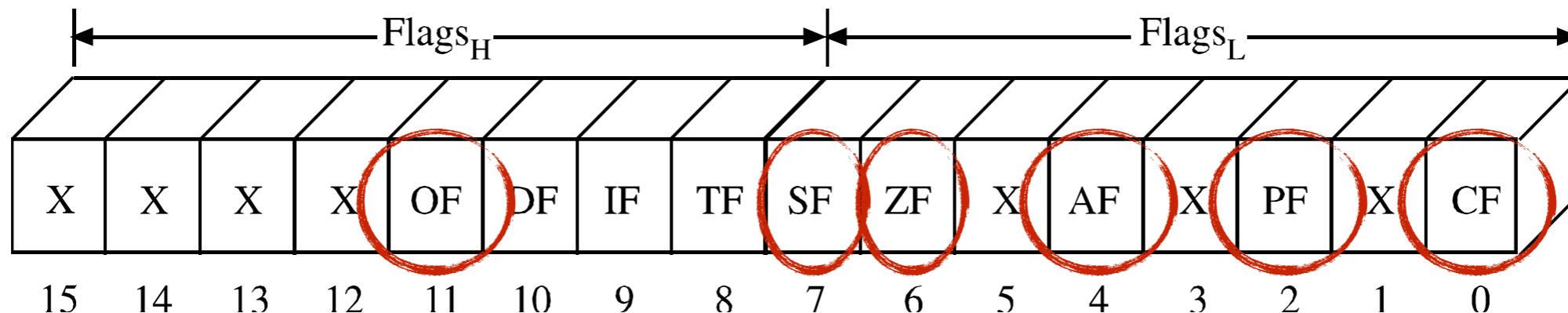
- פקודות העברת מידע
MOV, IN/OUT, PUSH, POP
 - פקודות חישוב אריתמטיות/לוגיות
ADD,SUB,MUL,AND,OR,XOR,...
 - פקודות בקרת תוכנה ומעבד
JMP, JZ,JNZ,..., CALL, RET, LOOP, STI, CLI,..
 - פקודות על מחרוזת
MOVS, LODS, STOS, REP
-
- (לא נלמד)

פקודות אריתמטיות/לוגיות

- פקודות המבצעות חישוב/עיבוד על המידע
- פקודות אריתמטיות מתיחסות לנตอน כאל **מספר**
(חיבור, חיסור, כפל...)
- פקודות לוגיות מתיחסות לנตอน כאל **אוסף ביטים**
(XOR , OR , AND)

פקודות אריתמטיות/לוגיות

- לעומת פקודת **MOV**, פעולות אריתמטיות-לוגיות
משנות את מצב **אוגר הדגלים**
- הדגלים משתנים על-פי התוצאה של החישוב
- כל פעולה יכולה לשנות דגלים מסוימים
- חלק מהדגלים עשויים להשנות באופן לא צפוי
(למשל: דגל *carry* בחישוב חילוק)



אוגר הדגלים

Flag L :



CF: Carry Flag $\begin{cases} \text{CF} = 0 : \text{No Carry (Add) or Borrow (SUB)} \\ \text{CF} = 1 : \text{high-order bit Carry/Borrow} \end{cases}$

PF: (Even) Parity Flag (even number of 1's in low-order 8 bits of result)

AF: Aux. Carry: Carry/Borrow on bit 3 (Low nibble of AL)

ZF: Zero Flag: (1: result is zero)

SF: Sign Flag: (0: positive, 1: negative)

OF: Overflow: signed result cannot be expressed within #bits in destination opera

פועלות אריתמטיות

פקודת INC

- מגדילה אוגר/זיכרון ב-1

`INC AX` \Rightarrow $AX := AX + 1$

`INC byte ptr [BX]` \Rightarrow $[BX] := [BX] + 1$

`INC word ptr [100]` \Rightarrow $[100,101] := [100,101] + 1$

- אם ערך מקסימום (FFFF או FF) אחרי ביצוע INC הערך יהיה 00 או 0000 , ודגל ZF ידلك

- דגל OF ידلك כאשר יוצג משלים ל-2 עבור מספר חיובי למספר שלילי

OF	SF	ZF	AF	PC	CF
±	±	±	±	±	

פקודת DEC

- מקטינה אוגר/זיכרון ב-1

`DEC AX` \Rightarrow `AX := AX - 1`

`DEC byte ptr [BX]` \Rightarrow `[BX] := [BX] - 1`

`DEC word ptr [100]` \Rightarrow `[100,101] := [100,101] - 1`

- אם ערךAPS אחרי ביצוע DEC הערך יהיה FF או FFFF
- דגל OF יידלק כאשר יוצג משלים ל-2 עובר ממספר שלילי לחובי

OF	SF	ZF	AF	PC	CF
±	±	±	±	±	

פקודת ADD

- חיבור שני נתונים

ADD AX, BX

$AX := AX + BX \pmod{16b}$

- מיעוניים אפשריים בדומה לפקודת MOV:

פעולה	דוגמא	סוג
$ax := ax + bx$	ADD AX, BX	add reg, reg
$dl := dl + 8$	ADD DL, 08h	add reg, const
$ax := ax + \text{mem}[bx+2,bx+3]$	ADD AX, [BX+2]	add reg,mem
$\text{mem}(50) := \text{mem}(50)+al$	ADD [50], AL	add mem,reg
$\text{mem}(si) = \text{mem}(si)+1$	ADD byte ptr [SI], 01h	add mem, const

OF	SF	ZF	AF	PC	CF
±	±	±	±	±	±

פקודת SUB

- חישור שני נתוניים

SUB AL, BH $AL := AL - BH \pmod{8b}$

- מיעונים אפשריים בדומה לפקודת MOV:

פעולה	דוגמא	סוג
$ax := ax - bx$	SUB AX, BX	sub reg, reg
$dl := dl - 8$	SUB DL, 08h	sub reg, const
$ax := ax - \text{mem}[bx+2,bx+3]$	SUB AX, [BX+2]	sub reg,mem
$\text{mem}(50) := \text{mem}(50) - al$	SUB [50], AL	sub mem,reg
$\text{mem}(si) = \text{mem}(si) - 1$	SUB byte ptr [SI], 01h	sub mem, const

OF	SF	ZF	AF	PC	CF
±	±	±	±	±	±

פקודות inc/dec/add/sub

INC = Increment:

Register/Memory

1111111 w	mod 0 0 r/m
01000 reg	

Register

DEC = Decrement:

Register/memory

1111111 w	mod 0 0 1 r/m
01001 reg	

Register

ADD = Add:

Reg./Memory with Register to Either

000000 d w	mod reg r/m		
100000 s w	mod 0 0 0 r/m	data	data if s: w = 01
0000010 w	data	data if w = 1	

Immediate to Register/Memory

Immediate to Accumulator

SUB = Subtract:

Reg./Memory and Register to Either

001010 d w	mod reg r/m		
100000 s w	mod 1 0 1 r/m	data	data if s w = 01
0010110 w	data	data if w = 1	

Immediate from Register/Memory

Immediate from Accumulator

פקודות inc/dec/add/sub

INC = Increment:

Register/Memory

Register

1111111w	mod 0 0 r/m
01000 reg	

DEC = Decrement:

Register/memory

Register

1111111w	mod 0 1 r/m
01001 reg	

ADD = Add:

Reg./Memory with Register to Either

Immediate to Register/Memory

Immediate to Accumulator

000000d w	mod reg r/m		
100000s w	mod 0 0 r/m	data	data if s: w = 01
0000010w	data	data if w = 1	

SUB = Subtract:

Reg./Memory and Register to Either

Immediate from Register/Memory

Immediate from Accumulator

001010d w	mod reg r/m		
100000s w	mod 1 0 r/m	data	data if s w = 01
0010110w	data	data if w = 1	

אפשר חיבור 8 ביט
לאוגר 16 ביט.

המספר המיידי יוארך
ל-16 ביט ע"י סיבית
הסימן

CMP פקודת

- **פקודת בקרה** לבדיקה שיוויון (compare)
- מבצעת בדיקת כמו SUB אבל בלי לשנות את האוגרים - רק הדגלים משתנים

CMP AL, BH

FLAGS change like in
SUB AL, BH

- דגל האפס יציין האם $AL=BH$ או לא, מבלי שינוי אותם

OF	SF	ZF	AF	PC	CF
±	±	±	±	±	±

פקודת MUL

- 8bit:
מכפילה אוגר/תא-זיכרון באוגר AL. התוצאה (16bit)
טופיע ב-AX.

MUL BH

MUL byte ptr [1000h]

$AX := AL * BH$

$AX := AL * [1000h]$

- 16bit:

מכפילה אוגר/זיכרון באוגר AX.

DX:AX התוצאה (32bit) טופיע ב AX

MUL BX

$DX:AX := AX * BX$

MUL word ptr [1000h]

$DX:AX := AX * [1000,1001]$

OF	SF	ZF	AF	PC	CF
±	?	?	?	?	±

כפל unsigned

פקודת iMUL

- כמו MUL אבל מתייחסת לאופרנדים כאל מספרים **signed**
- ל-MUL ו-iMUL, דגלי OF/CF מסמנים האם האוגר AL מספיק גדול להכיל את התוצאה (עבור 8ビט), או האם האוגר AX מספיק גדול להכיל את התוצאה (עבור 16ビט).
הדגלים יידלקו כאשר התוצאה торגת מוגדל האופרנד

OF	SF	ZF	AF	PC	CF
±	?	?	?	?	±

פקודת DIV/iDIV

- פעולה חילוק על מספרים signed / unsigned
- 8 ביט:
מחלקת את AX באופרנד. התוצאה מופיעה ב-AL והשארית ב-AH

DIV BH

$$\begin{aligned} AL &:= AX / BH \\ AH &:= AX \bmod BH \end{aligned}$$

- 16 ביט:
מחלקת את DX:AX באופרנד. התוצאה ב-AX והשארית ב-DX

DIV word ptr [BX]

$$\begin{aligned} AX &:= DX:AX / [BX,BX+1] \\ DX &:= DX:AX \bmod [BX,BX+1] \end{aligned}$$

OF	SF	ZF	AF	PC	CF
?	?	?	?	?	?

פקודת mul/div

MUL = Multiply (Unsigned)

IMUL = Integer Multiply (Signed)

1 1 1 1 0 1 1 w	mod 1 0 0 r/m
1 1 1 1 0 1 1 w	mod 1 0 1 r/m

DIV = Divide (Unsigned)

IDIV = Integer Divide (Signed)

1 1 1 1 0 1 1 w	mod 1 1 0 r/m
1 1 1 1 0 1 1 w	mod 1 1 1 r/m

פקודת NEG

- הפיכת מספר signed משלילי לחיובי, או להפר
(כזכור: במשלים ל-2 סקול להפיכת סיביות + 1)
- הפעלה על אוגר או זיכרון

NEG AL

NEG CX

NEG byte



$AL := 0 - AL$

$CX := 0 - CX$

$[1000h] := 0 - [1000h]$

OF	SF	ZF	AF	PC	CF
±	±	±	±	±	±

נק אם האופרנד 0 אז CF=0
נק אם האופרנד 80h אז OF=1

פקודות לוגיות

פקודת AND

- ביצוע bitwise-AND בין שני אופרנדים

AND CL, 08h

0	1	1	0	1	0	0	0
0	0	0	0	1	0	0	0

CL
08h

CL \leftarrow

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

- נועד כדי "לאפס" ביטים לא רצויים (ולשמור את ערכם של הביטים הרצויים)

- מיעוניים - בדומה לMOV

OF	SF	ZF	AF	PC	CF
0	\pm	\pm	?	\pm	0

פקודת OR

- ביצוע bitwise-OR בין שני אופרנדים

OR [BX], DL

0	1	1	0	1	0	0	0
1	0	0	0	1	0	1	0

[BX]
DL

[BX] ←

1	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

- נועד כדי "להדליק" ביטים רצויים (ולשמור את ערכם של שאר הביטים)

- מיעוניים - בדומה לMOV

OF	SF	ZF	AF	PC	CF
0	±	±	?	±	0

פקודת XOR

- ביצוע bitwise-XOR בין שני אופרנדים

XOR AX, FF00h

0	1	1	0	1	0	0	0
1	1	1	1	1	1	1	1
1	0	0	1	0	1	1	1

0	1	1	0	1	0	0	0
0	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0

- נועד כדי "להפוך" ערכם של ביטים רצויים (ולשמור את ערכם של שאר הביטים)

- מיעוניים - בדומה לMOV

OF	SF	ZF	AF	PC	CF
0	±	±	?	±	0

פקודת AND/OR/XOR

AND = And:

Reg./Memory and Register to Either

001000 d w	mod reg r/m		
1000000 w	mod 100 r/m	data	data if w = 1
0010010 w	data	data if w = 1	

Immediate to Register/Memory

Immediate to Accumulator

TEST = And Function to Flags, No Result:

Register/Memory and Register

1000010 w	mod reg r/m		
1111011 w	mod 000 r/m	data	data if w = 1
1010100 w	data	data if w = 1	

OR = Or:

Reg./Memory and Register to Either

000010 d w	mod reg r/m		
1000000 w	mod 001 r/m	data	data if w = 1
0000110 w	data	data if w = 1	

Immediate to Register/Memory

Immediate to Accumulator

XOR = Exclusive or:

Reg./Memory and Register to Either

001100 d w	mod reg r/m		
1000000 w	mod 110 r/m	data	data if w = 1
0011010 w	data	data if w = 1	

Immediate to Register/Memory

Immediate to Accumulator

פקודת NOT

- ביצוע היפוך של כלቢיטי האופרנד

NOT AL

0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

NOT byte ptr [BX]

NOT word ptr [1000h]

...



1	0	0	1	0	1	1	1
---	---	---	---	---	---	---	---

OF	SF	ZF	AF	PC	CF

פקודת NOT

- מה הבדלים בין NOT ל-NEG?
- NEG = “NOT + INC”
(0-operand אבל דגלים כמו)
- NOT לא משנה את הדגלים (חריג)
- NEG שומר על הערך מוחלט של הנתון

פקודת TEST

- **פקודת בקרה לבדיקת ביטים**
- מבצעת בדיקת כמו AND אבל בלי לשנות את האוגרים - רק הדגלים משתנים

TEST AL, BH

FLAGS change like in
AL AND BH

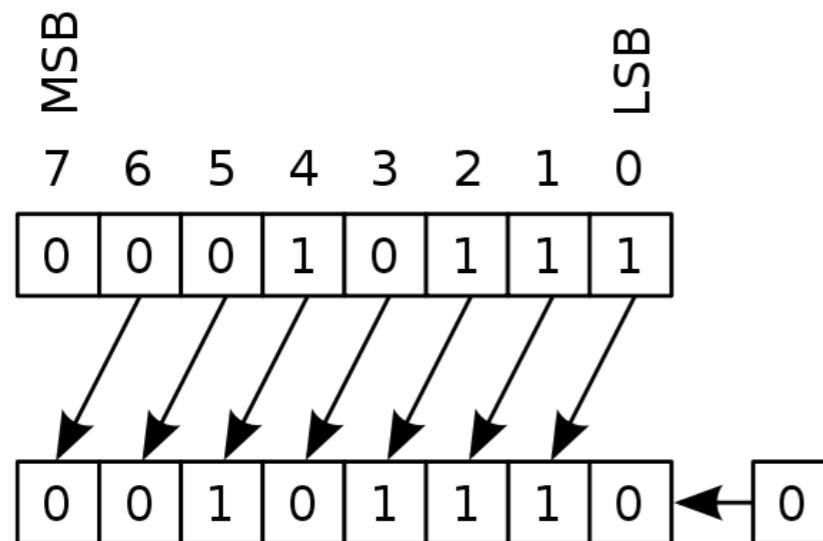
- למשל: כדי לבדוק אם ה-LSB דולק בלי לשנות את האוגר, נבצע `TEST AL, 01h`. דגל ZF יציין האם הבית דולק או לא

OF	SF	ZF	AF	PC	CF
0	±	±	?	±	0

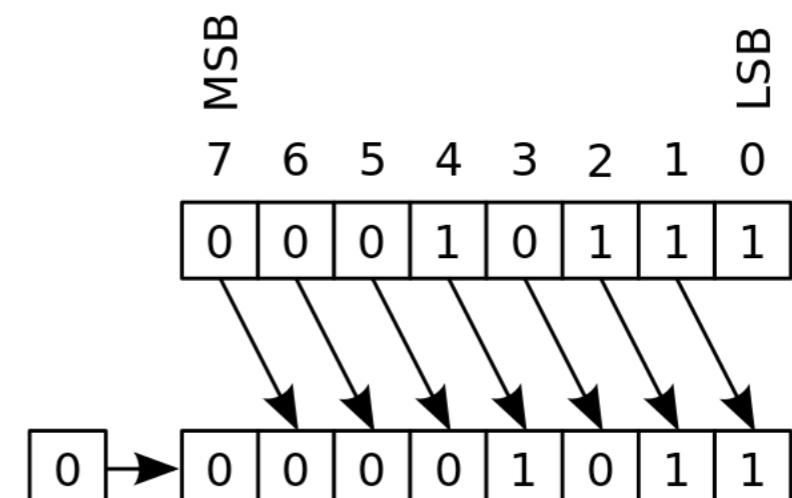
פקודות הזזה לוגית SHL , SHR

- מבצעת הזזה (shift) של הסיביות במקומות אחד ימינה או שמאלה.
- הסיבית ה"חדשה" תהיה 0

SHL AL, 1



SHR CL,1



פקודות הזהה לוגית SHL , SHR



- ניתן לבצע הזהה של יותר מצעד אחד ע"י שימוש ב-CL

SHL AX, CL

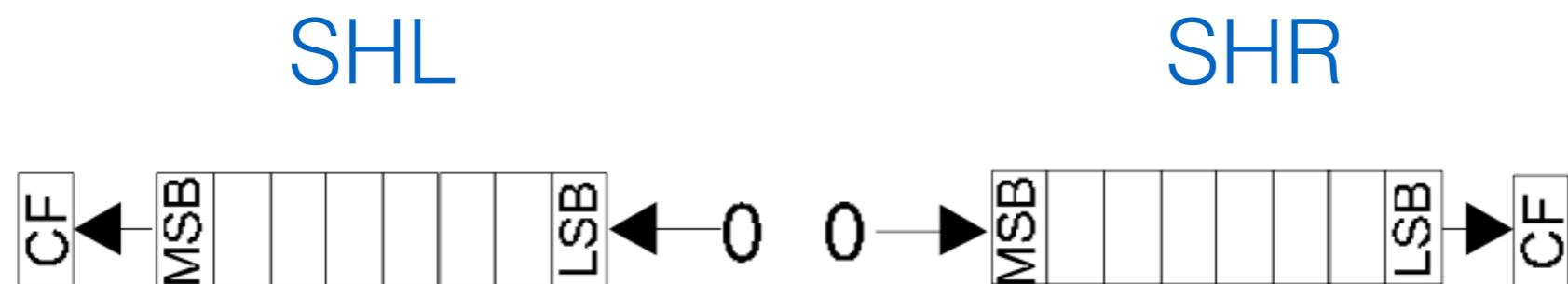
זהות AX שמאלה במספר צעדים
לפי הערך של CL

SHR word ptr [1000], CL

זהות המילה בתאים [1000,1001]
ימינה במספר צעדים לפי הערך של CL

פקודות הזזה לוגית SHL , SHR

- הסיבית ה"יוצאת" עוברת ל-CF



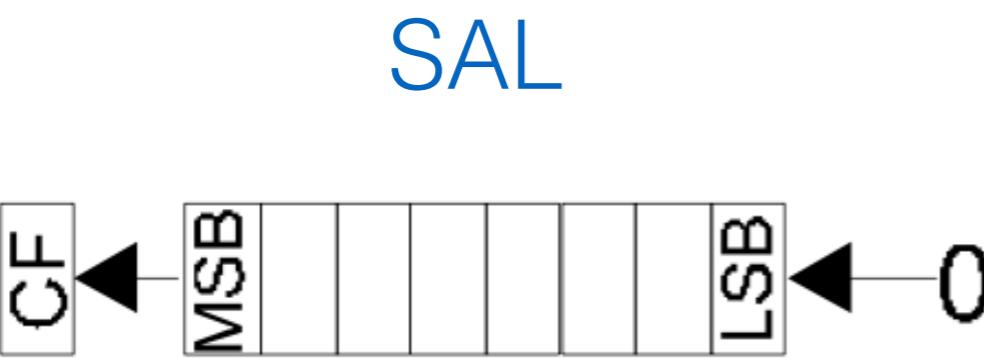
- OF נדלק במעבר בין חיובי לשילי

OF	SF	ZF	AF	PC	CF
±	±	±	?	±	±

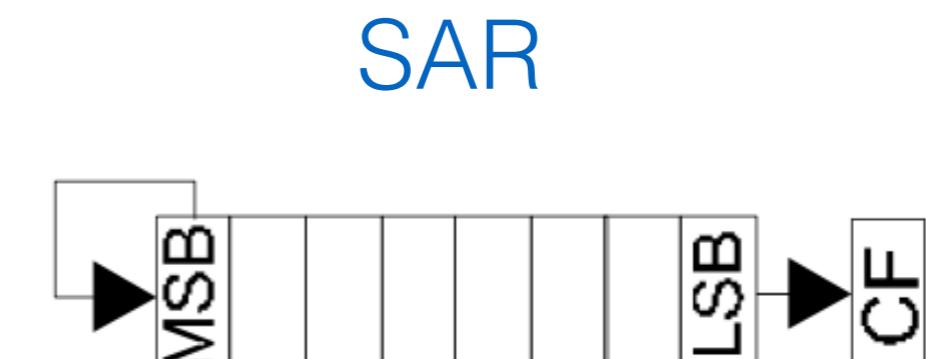
פקודות הזזה אריתמטית

SAL , SAR

- מבצעת הזזה (shift) של הסיביות במקומות אחד ימינה או שמאלה - **כך שהמספר signed שומר על סימנו**
- הזרת מספר צעדים ע"י CL



OF	SF	ZF	AF	PC	CF
±/0	±	±	?	±	±

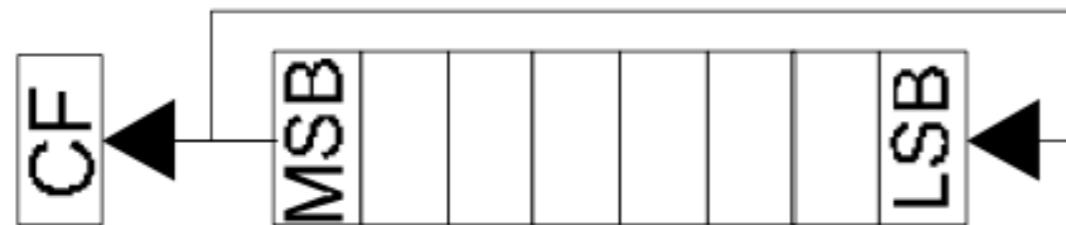


סיבית הסימן

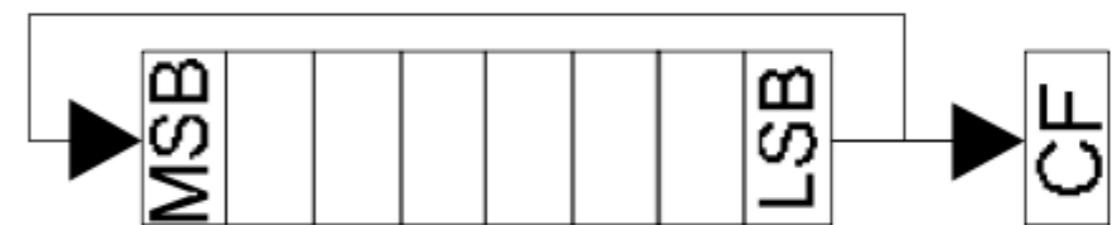
פקודות סיבוב אריתמטית

ROL/ROR, RCL/RCR

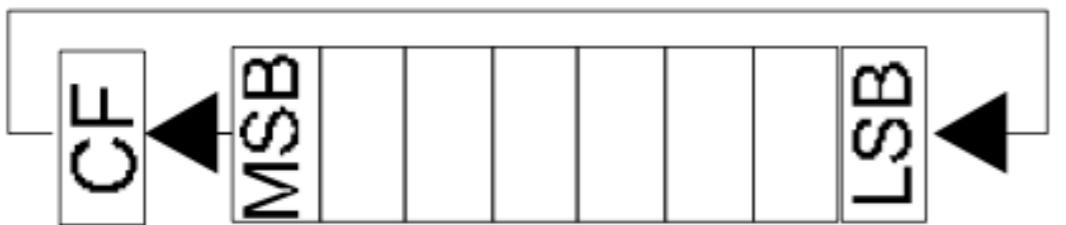
ROL



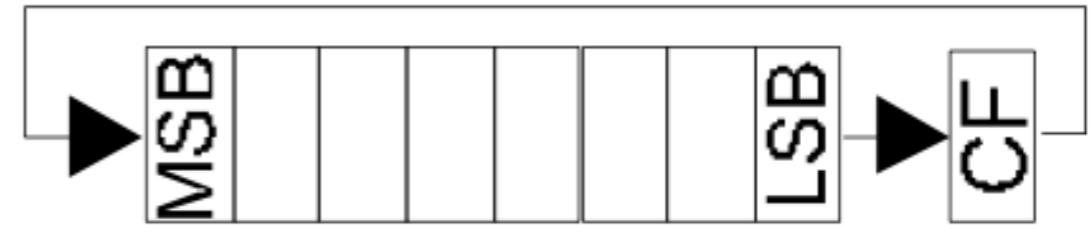
ROR



RCL



RCR



OF	SF	ZF	AF	PC	CF
±					±

פקודות סיבוב וNOT

NOT = Invert

SHL/SAL = Shift Logical/Arithmetic Left

SHR = Shift Logical Right

SAR = Shift Arithmetic Right

ROL = Rotate Left

ROR = Rotate Right

RCL = Rotate Through Carry Flag Left

RCR = Rotate Through Carry Right

1 1 1 1 0 1 1 w	mod 0 1 0 r/m
1 1 0 1 0 0 v w	mod 1 0 0 r/m
1 1 0 1 0 0 v w	mod 1 0 1 r/m
1 1 0 1 0 0 v w	mod 1 1 1 r/m
1 1 0 1 0 0 v w	mod 0 0 0 r/m
1 1 0 1 0 0 v w	mod 0 0 1 r/m
1 1 0 1 0 0 v w	mod 0 1 0 r/m
1 1 0 1 0 0 v w	mod 0 1 1 r/m

מה זה?
↑

תרגיל: פעולות ארית'-לוגיות

- מה עושה הקטע הבא:

XOR AX, AX

- שקול ל **MOV AX, 0** (אבל קצר יותר!)

- מה עושה הקטע הבא:

XOR BX,CX

XOR CX,BX

XOR BX,CX

- פקודה שcolaה : **XCHG BX,CX**

פקודות אריתמטיות/לוגיות:

סיכום

- **מגוון פקודות המבצעות "חישוב" בין שני נתונים**
- **לרוב החישוב בין שני אוגרים.** חלק מהפקודות אפשרות פעולה על **הזיכרון** לפי שיטות המיעוון השונות
- **ניתן להתייחס לנตอน בטור מספר unsinged, מספר signed או אוסף ביטים**
- **הדגלים משתנים בהתאם לתוכנת החישוב**
- **פקודת MOV אינה משנה דגלים!**

פקודות בקרה

בקרת זרימת התוכנית

- בחלק זה נלמד פקודות המאפשרות בקרת זרימת התוכנית
- למשל:
- כיצד ניתן לבדוק תנאי ..
`IF (AX == 0) DO ... ELSE ..`
- כיצד מבצעים לולאה
`WHILE (AX != 0) DO`

תוויות (Labels)

- לפני כל שורת פקודה ניתן לשים תוית:

[Label:] Command [OP1] [, OP2]

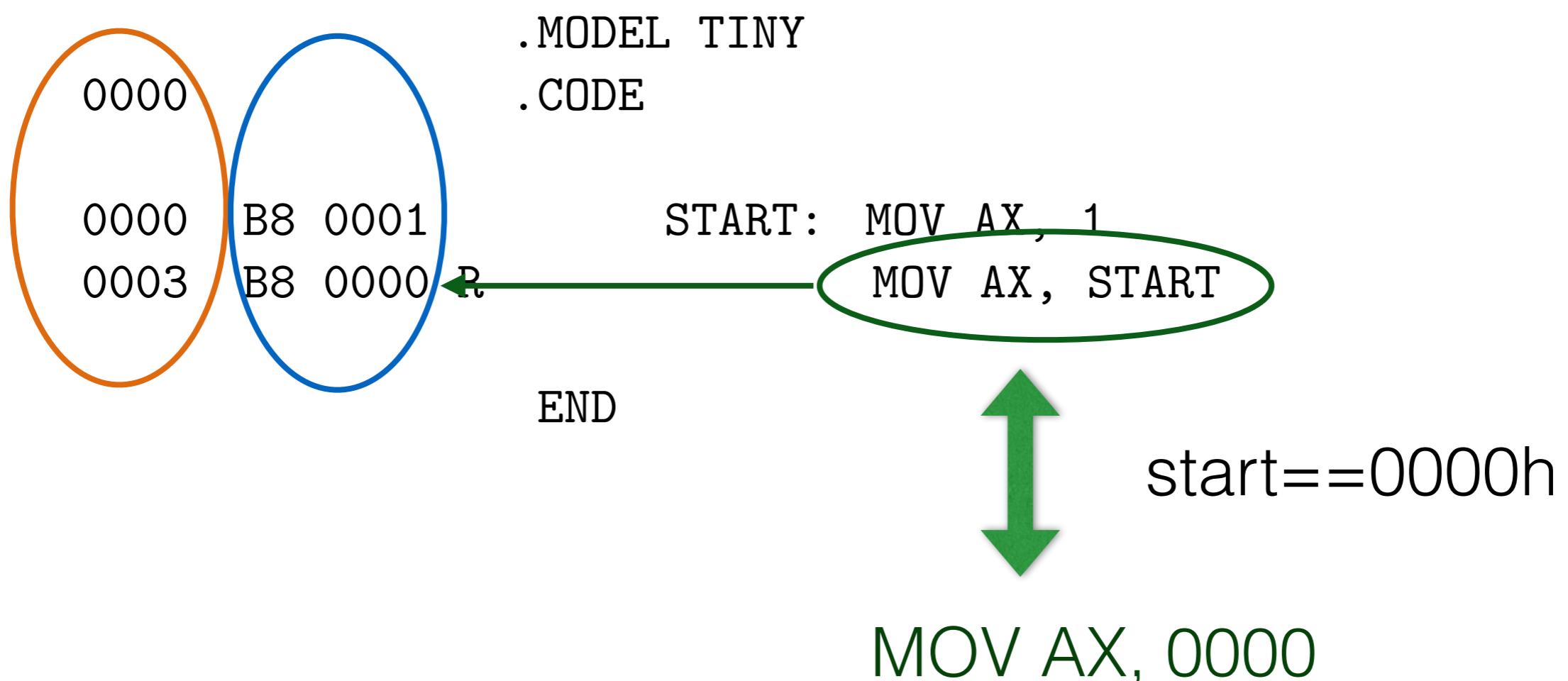
- תוית מורכבת מרצף אותיות+מספרים ולאחריהם נקודותיים (למעט מספר מילים שמורות, כמו DB או .CODE).

- הтоית שקופה ל" כתובת" בו נמצאת הפקודה
וניתן להשתמש בה כמו קבוע מספרי

• **למשל:**
START: MOV AX, 1
 MOV BX, START

תוויות (Labels)

**פקודה כתובה
(שפת מconaה)**



תוויות (Labels)

```

.MODEL TINY
.CODE
DB 100h DUP(?)
[

START: MOV AX, 1
MOV AX, START
END

```

0000 0100 [00]

0100 B8 0001 ← 0001

0103 B8 0100 ← R

- המהדר מחליף את התווית בערך הכתובת בה היא נמצאת
- ניתן לשימוש בתווית בכל מקום בו היינו משתמשים קבוע
- **המעבד אינו מכיר את התווית!!** זהו כלי עברור המתוכנת / קומפайлר!!

The list of words in this table are words that have special meaning to the assembler. You cannot use them for any purpose other than that defined by the assembler.

.186	.286	.286C	.286P	.287	.386
.386C	.386P	.387	.8086	.8087	ALIGN
.ALPHA	AND	ASSUME	AT	BYTE	CATSTR
.CODE	COMMENT	COMMON	.CONST	.CREF	.DATA
.DATA?	DB	DD	DOSSEG	DQ	DT
DUP	DW	DWORD	ELSE	ELSEIF	ELSEIFI1
ELSEIF2	ELSEIFB	ELSEIFDEF	ELSEIFDIF	ELSEIFDIFI	ELSEIFE
ELSEIFIDN	ELSEIFIDNI	ELSEIFNB	ELSEIFNDEF	END	ENDIF
ENDM	ENDP	ENDS	EQ	EQU	.ERR
.ERR1	.ERR2	.ERRB	.ERRDEF	.ERRDIF	.ERRDIFI
.ERRE	.ERRIDN	.ERRIDNI	.ERRNB	.ERRNDEF	.ERRNZ
EVEN	EXITM	EXTRN	FAR	.FARDATA	.FARDATA?
FWORD	GE	GROUP	GT	HIGH	IF
IF1	IF2	IFB	IFDEF	IFDIF	IFDIFI
IFE	IFIDN	IFIDNI	IFNB	IFNDEF	INCLUDE
INCLUDELIB	INSTR	IRP	IRPC	LABEL	.LALL
LE	LENGTH	.LFCOND	.LIST	LOCAL	LOW
LT	MACRO	MASK	MEMORY	MOD	.MODEL
NAME	NE	NEAR	NOT	OFFSET	OR
ORG	%OUT	PAGE	PAGE	PARA	PROC
PTR	PUBLIC	PUBLIC	PURGE	QWORD	.RADIX
RECORD	REPT	.SALL	SEG	SEGMENT	.SEQ
.SFCOND	SHL	SHORT	SHR	SIZE	SIZESTR
.STACK	STACK	STRUC	SUBSTR	SUBTTL	TBYTE
.TFCOND	THIS	TITLE	.TYPE	WIDTH	WORD
.XALL	.XREF	.XLIST	XOR		

In addition to the above words, TASM also reserves these:

ARG	%BIN	CODESEG	%COND\$	CONST	%CREF
%CREFALL	%CREFREF	%CREFUREF	%CTL\$	DATAPTR	DATASEG
%DEPTH	DISPLAY	DP	EMUL	ERRIF	ERRIF1
ERRIF2	ERRIFB	ERRIFDEF	ERRIFDIF	ERRIFDIFI	ERRIFE
ERRIFIDN	ERRIFIDNI	ERRIFNB	ERRIFNDEF	EVENDATA	FARDATA
FARDATA?	GLOBAL	IDEAL	%INCL	JUMPS	LARGE
%LINUM	%LIST	LOCAL\$	%MACS	MASM	MASM51
MODEL	MULTERRS	%NEWPAGE	%NOCOND\$	%NOCREF	%NOCTL\$
NOEMUL	%NOINCL	NOJUMPS	%NOLIST	NOLOCAL\$	%NOMACS
NOMASM51	NOMULTERRS	%NOSYMS	%NOTRUNC	NOWARN	P186
P286	P286N	P286P	P287	P386	P386N
P386P	P387	P8086	P8087	%PAGESIZE	%PCNT
PNO87	%POPLCTL	%PUSHLCTL	PWORD	QUIRKS	RADIX
SMALL	%SUBTTL	%SYMS	SYMTYPE	%TABSIZE	%TEXT
%TITLE	%TRUNC	UDATASEG	UFARDATA	UNION	UNKNOWN
WARN					

- **רשימת מילים שמורות (אסורות לשימוש כתוית)**
- **כמו כן לא ניתן להשתמש ב:**
 - ♦ **שמות של פקודות, (כגון LOOP)**
 - ♦ **שמות אוגרים**
 - ♦ **שמות משתנים שהוגדרו או שמות שגרות**
 - ♦ **לא ניתן להתחליל בספרה (מוסכמת: רק קבוע מתחיל בספרה)**

פקודה JMP

- הפקודה תגרום "קפיצה" אל מקום המצוין בפקודה:

0000	START:	MOV AX, 1	
0003		JMP START	; infinite loop...

- הפקודה "מעדכנת" את האוגר IP לערך .START.
הפקודה הבאה שתבוצע היא הפקודה במקומות .START.

JMP START → IP ← 0000

פקודת JMP

- 3 סוגי קפיצה:
 - short — קפיצה של 127/128+ כתובות ממיקום הנוכחי
 - near — קפיצה למקומות כלשהו בסגמנט הקוד הנוכחי
 - far — קפיצה לסגמנט אחר
- המדר/קשר משתמש בקפיצה המתאימה לפי מיקום התווית (אלא אם פקודה מפורשת ... JMP FAR ...)

פקודה JMP

- קפיצה — הפקודה מכילה היסט (8/16 ביט).

JMP = Unconditional Jump:

Direct within Segment

Direct within Segment-Short

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within Segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within Segment-Short	1 1 1 0 1 0 1 1	disp	

$$\text{IP} \leftarrow \text{IP} + \text{disp} \quad \bullet$$

- פקודת FAR מכילה את הסgement החדש + כתובה

Direct Intersegment

1 1 1 0 1 0 1 0	offset-low	offset-high
	seg-low	seg-high

$$\text{CS} \leftarrow \text{seg} ; \quad \text{IP} \leftarrow \text{offset} \quad \bullet$$

פקודת JMP

- קפיצה — הפקודה מכילה היסט (8/16 ביט).

JMP = Unconditional Jump:

Direct within Segment

Direct within Segment-Short

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
1 1 1 0 1 0 0 1	disp-low	disp-high
1 1 1 0 1 0 1 1	disp	

$$\text{IP} \leftarrow \text{IP} + \text{disp}$$

שים לב:
 ערך IP **לאחר** סיום
 קריאת פקודת JMP

- הפקודה EB 00 מבצעת "קפיצה" לפקודה הבא

- דוגמא:

מיעון עקיর JMP



אוניברסיטת בר-אילן
Bar-Ilan University

JMP = Unconditional Jump:

Direct within Segment

7 6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

1 1 1 0 1 0 0 1

disp-low

disp-high

Direct within Segment-Short

1 1 1 0 1 0 1 1

disp

Indirect within Segment

1 1 1 1 1 1 1 1

mod 1 0 0 r/m

Direct Intersegment

1 1 1 0 1 0 1 0

offset-low

offset-high

seg-low

seg-high

Indirect Intersegment

1 1 1 1 1 1 1 1

mod 1 0 1 r/m

JMP AX

IP \leftarrow AX

JMP [BX+20h]

IP \leftarrow (WORD) [BX+20]

JMP FAR PTR [100h]

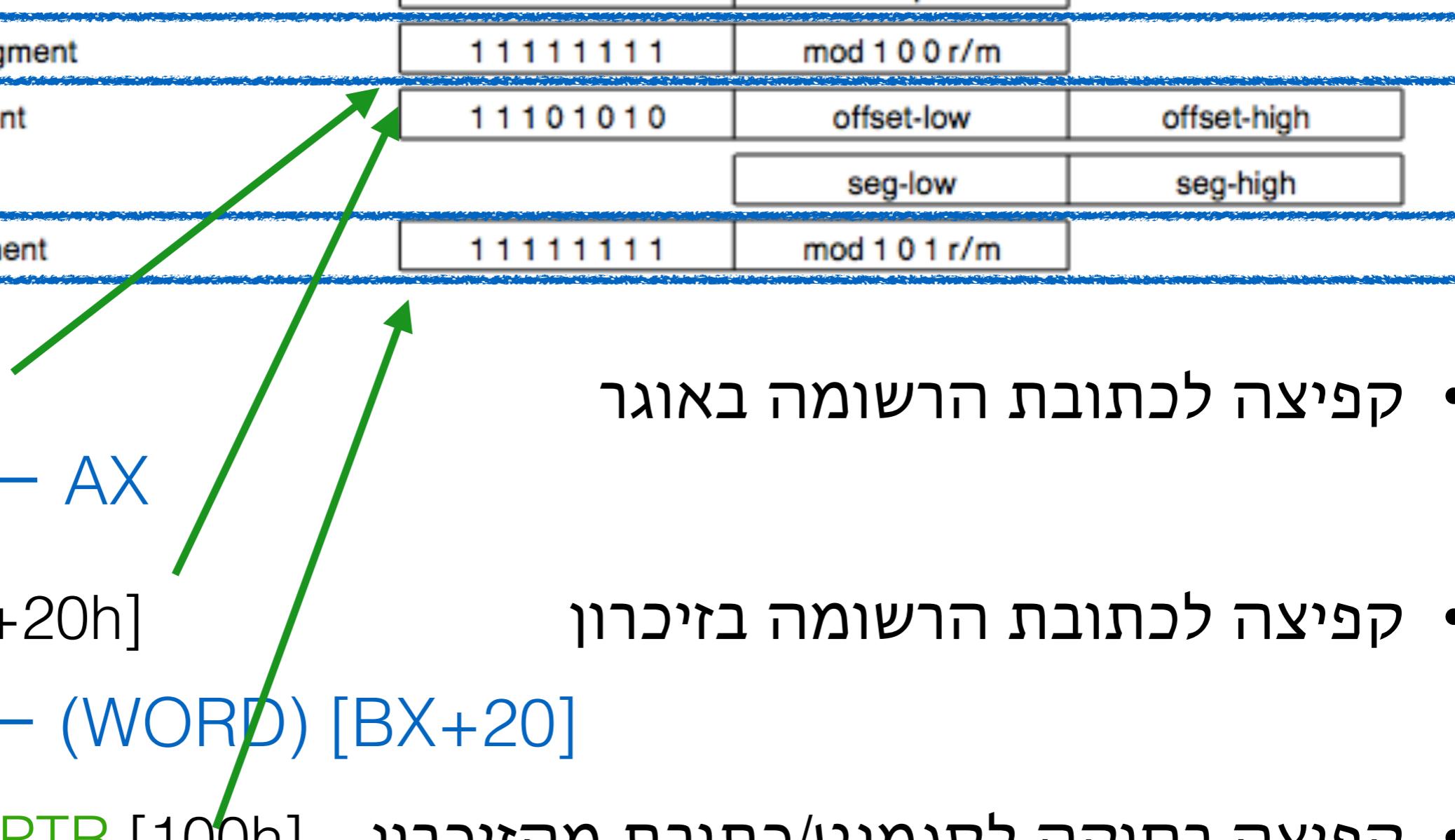
IP \leftarrow [100,101]

• קפיצה לכתובת הרשומה באוגר

• קפיצה לכתובת הרשומה בזיכרון

• קפיצה רוחקה לSEGMENT/כתובת מהזיכרון

CS \leftarrow [102,103]



קפיצה מבוקרת

- קיימים סט פקודות שמבצעות קפיצה בצורה מותנית -
רק אם הדגליםקיימים תנאי כלשהו
- אחרת - התוכנית **משיכה** לפקודה הבאה
- קפיצה מבוקרת היא **תמיד** מסוג short (8bit היסט)
- **מבנה** פקודה עקרוני

J-CMD label

פקודת JZ/JNZ

- ZJ מבצעת קפיצה רק אם דגל ZF דולק
- NZ מבצעת קפיצה רק אם דגל ZF כבוי
- **לדוגמה**

AGAIN: XOR AX,AX
JZ AGAIN

פקודת JZ/JNZ

- תרגיל: נכתוב לולאה שתבצע 5 פעמים בדיק

MOV CL,5

ЛОP5: <some inst.>

...

DEC CL

JNZ LOP5

<rest of program>

פקודת JZ/JNZ

- פקודת JZ נקראת גם JE (Jump Equal).
 - מה עושה הקוד הבא:
- DONT: CMP AX, BX
JZ DO
<some inst.>
...
JMP endProg
- DO: <some other inst.>
...
JMP endProg
[...]
- 
- ```
IF (AX==BX)
 <some other inst.>
ELSE
 <some inst.>
```

# קפיצה מותנת לפידג'ל

- לכל אחד מדגלי PF, CF, OF, SF קיימת קפיצה מותנית לפיקוחו של הדגל
- JC / JNC - קפיצה אם דגל carry דולק/כבוי
- JP / JNP - האם דגל הזוגיות דולק/כבוי
- JO / JNO - האם היה Overflow ..
- JS / JNS - האם דגל הסימן דולק..

# קפיצות מותנות ארית'

- קיים סט קפיצות מותנות לשווה את שני מספרים וקפיצה בהתאם למצבם: האם שווים, האם הראשון גדול יותר או קטן יותר, וכו'.
- מי גדול יותר ??? b11000 או ??? 1b0001 •
- תלוי
- אם  $1 > 1 < 13$  או unsigned •
- אם  $1 < 1 < -3$  או signed •

# קפיצות מותנות ארית'

cmp Operand1, Operand2

| מספרים Unsigned              | מספרים Signed                  | משמעות הפקודה                                       |
|------------------------------|--------------------------------|-----------------------------------------------------|
| JA - Jump if Above           | JG - Jump if Greater           | קפוץ אם האופrnd הראשון<br>גדול מהשני                |
| JB - Jump Below              | JL - Jump if Less              | קפוץ אם האופrnd הראשון<br>קטן מהשני                 |
| JE - Jump Equal              |                                | קפוץ אם האופrnd הראשון<br>והשני שווים               |
| JNE - Jump Not Equal         |                                | קפוץ אם האופrnd הראשון<br>והשני שונים               |
| JAE - Jump if Above or Equal | JGE - Jump if Greater or Equal | קפוץ אם האופrnd הראשון<br>גדול או שווה לאופrnd השני |
| JBE - Jump if Below or Equal | JLE - Jump if Less or Equal    | קפוץ אם האופrnd הראשון<br>קטן או שווה לאופrnd השני  |

# קפיצות מותנות ארית'

- לדוגמה, מה צריכים להיות הדגלים ע"מ לבצע קופצת JB  
(קפוץ אם האופרנד הראשון קטן מהשני, `unsigned`)
  - CMP OP1, OP2
- אם  $OP2 \geq OP1$  אז לא יהיה carry בחישור (borrow)
- אם  $OP2 < OP1$  אז יהיה carry
- לכן JB שקול לבדיקה !CF
- מה קורה עבור שני מספרים ? **signed**

# תנאי דגלים לkapיצה

| משמעות                     | קפיצה אם | הפקודה |
|----------------------------|----------|--------|
| כפוץ אם האופרנדים<br>שווים | ZF=1     | JE     |

## UNSIGNED

| משמעות               | קפיצה אם      | הפקודה |
|----------------------|---------------|--------|
| כפוץ אם פחות מ..     | CF=1          | JB     |
| כפוץ אם פחות או שווה | ZF=1 or CF=1  | JBE    |
| כפוץ אם יותר מ..     | CF=0 and ZF=0 | JA     |
| כפוץ אם יותר או שווה | CF=0          | JAE    |

# תנאי דגלים לקבעה

**SIGNED**

| משמעות                 | קבעה אם                | הפקודה |
|------------------------|------------------------|--------|
| קבע אמצעי מ..          | $SF \neq OF$           | JL     |
| קבע אמצעי קטן או שווה  | $SF \neq OF$ or $ZF=1$ | JLE    |
| קבע אמצעי גדול מ..     | $SF=OF$ and $ZF=0$     | JG     |
| קבע אמצעי גדול או שווה | $SF=OF$                | JGE    |

# פקודת LOOP

- פקודת לולאה מקוצרת: ממשת לולאת FOR כמספר הפעמים שרשום באוגר CX

```
MOV CX,5
AGN: <some inst.>
...
LOOP AGN
<rest of program>
```

```
MOV CX,5
AGN: <some inst.>
...
CX ← CX-1
JNZ AGN
<rest of program>
```

- בכל ביצוע פקודת LOOP, אוגר CX מופחת ב-1  
ומתבצעת קפיצה ZJN לתוית

# לולאות מקווננות

- כיצד ניתן לבצע לולאה בתוך לולאה?
- מה יקרה למשל בקטע הבא:

```
MOV CX, 10
```

```
LoopA:
```

```
 MOV CX, 5
```

```
LoopB: ... ; Some code for LoopB
```

```
 LOOP LoopB
```

```
 ... ; Some code for LoopA
```

```
 LOOP LoopA
```

# קפיצה מותנית - שפת מכונה

**JE/JZ** = Jump on Equal/Zero

**JL/JNGE** = Jump on Less/Not Greater or Equal

**JLE/JNG** = Jump on Less or Equal/Not Greater

**JB/JNAE** = Jump on Below/Not Above or Equal

**JBE/JNA** = Jump on Below or Equal/Not Above

**JP/JPE** = Jump on Parity/Parity Even

**JO** = Jump on Overflow

**JS** = Jump on Sign

**JNE/JNZ** = Jump on Not Equal/Not Zero

**JNL/JGE** = Jump on Not Less/Greater or Equal

**JNLE/JG** = Jump on Not Less or Equal/Greater

**JNB/JAE** = Jump on Not Below/Above or Equal

**JNBE/JA** = Jump on Not Below or Equal/Above

**JNP/JPO** = Jump on Not Par/Par Odd

**JNO** = Jump on Not Overflow

**JNS** = Jump on Not Sign

**LOOP** = Loop CX Times

**LOOPZ/LOOPE** = Loop While Zero/Equal

**LOOPNZ/LOOPNE** = Loop While Not Zero/Equal

**JCXZ** = Jump on CX Zero

|          |      |
|----------|------|
| 01110100 | disp |
| 01111100 | disp |
| 01111110 | disp |
| 01110010 | disp |
| 01110110 | disp |
| 01111010 | disp |
| 01110000 | disp |
| 01111000 | disp |
| 01110101 | disp |
| 01111101 | disp |
| 01111111 | disp |
| 01110011 | disp |
| 01110111 | disp |
| 01111011 | disp |
| 01110001 | disp |
| 01111001 | disp |
| 11100010 | disp |
| 11100001 | disp |
| 11100000 | disp |
| 11100011 | disp |

- כאמור, רק קפיצה SHORT.

- מה עושים אם נדרשת קפיצה רחוקה יותר?

# פקודות בקרה: סיכום

- בקרת זרימת התוכנית מבוצעת על-ידי פקודות "קפיצה" מותנות במצב הדגמים
- בשילוב עם פקודה CMP/TEST (CMP/TEST) ניתן לקודד לולאות, תנאים, וכו'.
- CMP מבצעת השוואה בין שני מספרים, وكפיצה לפי היחס ביניהם (בהתאם במצב הדגמים)
- קפיצה מותנית אפשרית רק בטווח של 8 ביט ממיקום הנוכחי
- פקודה LOOP מבצעת לולאה אוטומטית X פעמים.

# פקודות בקרה: תרגיל

- כיצד ניתן לבדוק תנאי ..  
`IF (AX == 0) DO ... ELSE ..`
- כיצד מבצעים לולאה  
`WHILE (AX != 0) DO ....`

# אורים: מקורות

- B. Brey, The Intel Microprocessor, 2009
- Intel 8086 spec, INTEL, 1990
- Wikipedia: Logical Shift, Arithmetic Shift
- ארגון המחשב ושפת ספ, ברק גונן, גבאים, מרכז הסייבר הצבאי 2015

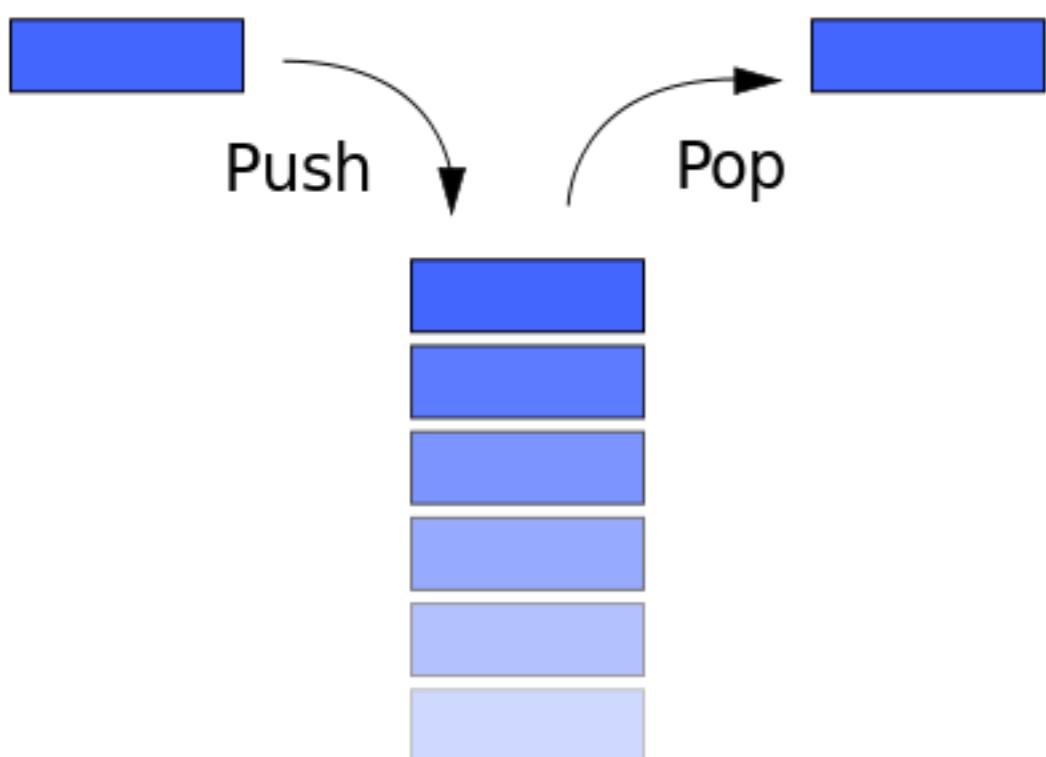
# המחסנית ותת-י-שגרות (רוטיניות)

מיקромעבדים ופתח אסמלר 83-255-83



# מחסנית

- **“מחסנית”** הינה מבנה נתונים מסווג  
**LIFO - Last In First Out**



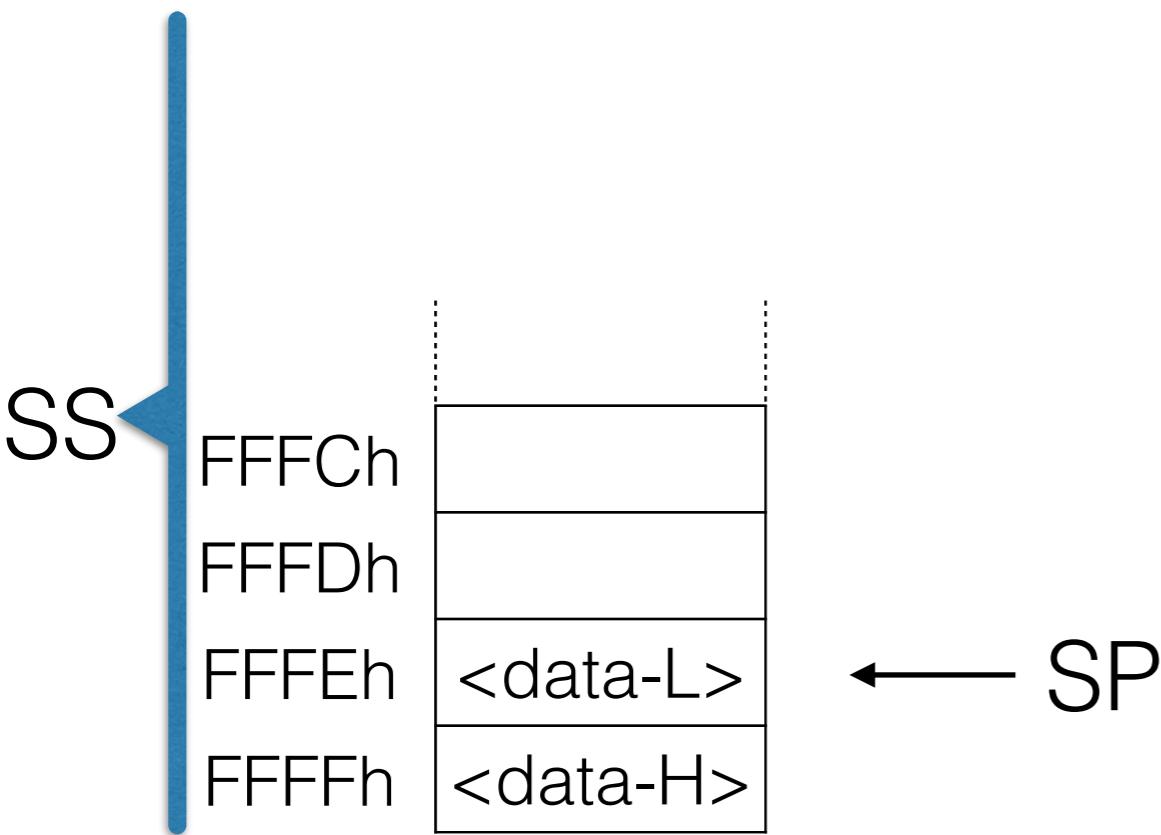
- תומכת בשתי פעולות:
  - PUSH - הכנסת נתון לראש המחסנית
  - POP - שילפת הנתון העליון מראש המחסנית

# מחסנית ב-8086

- איזור “**המחסנית**” הוא איזור בזיכרון המשמש לשימרת נתוניים **“זמןניים”**
- שימירה זמנית של **תוכן אוגרים** (למשל, ע”מ לבצע פעולה הדורשת שימוש במספר גדול של אוגרים)
- שימירת נתונים בעת **מעבר ל תת-שגרות** (למשל, כתובת החזרה, פרמטרים לשגרה...)

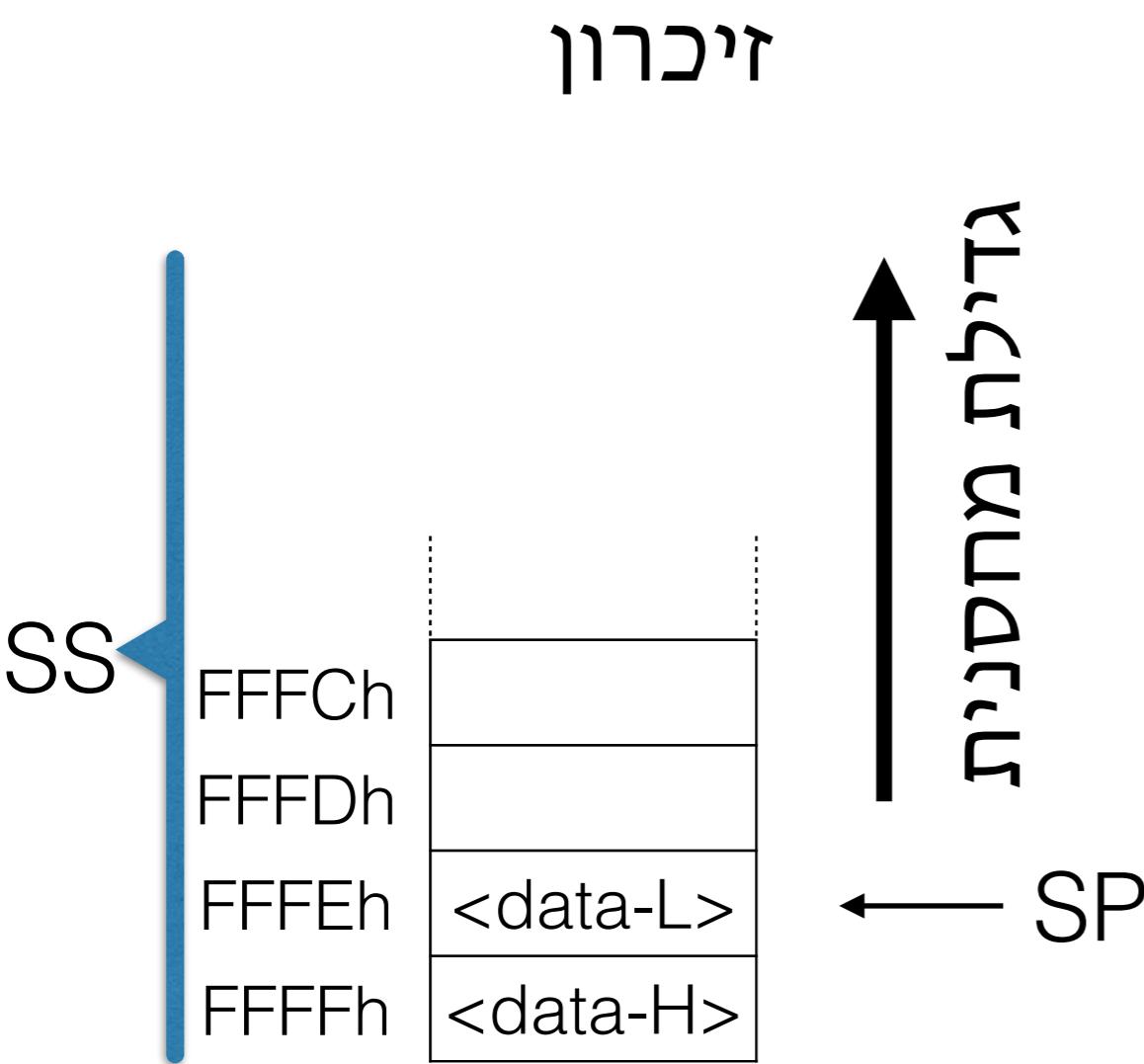
# מבנה המחסנית

זיכרון



- המחסנית נמצאת במקטע סגמנט SS
- גודל מחסנית  $> 64k$
- אוגר SP מצביע על ראש המחסנית - על הנתון החדש ביותר

# מבנה המחשבנית



- **שומרה 1:** המחשבנית גדרה לכיוון כתובות נמוכות
- **שומרה 2(א):** כל רשותה 16 ביט בלבד
- **שומרה 2(ב):** בית נמוך בכתובת נמוכה (סדר תואם לפקודה MOV של 16 ביט)

# שימוש במחסנית

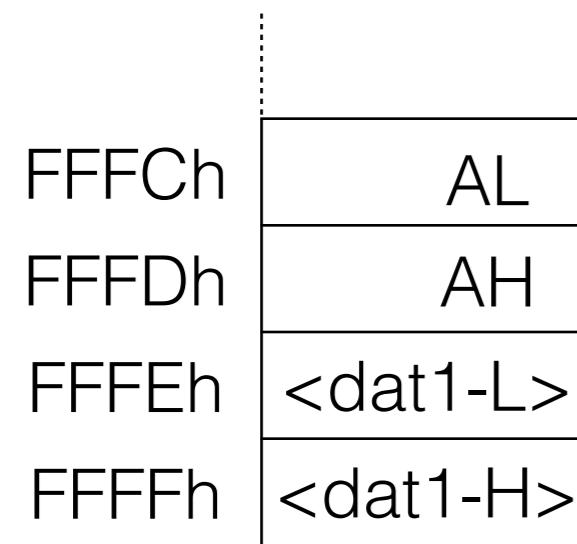
- הכנסה למחסנית:

PUSH <16b register>  
PUSH <16b Memory>

שליפה מהמחסנית:

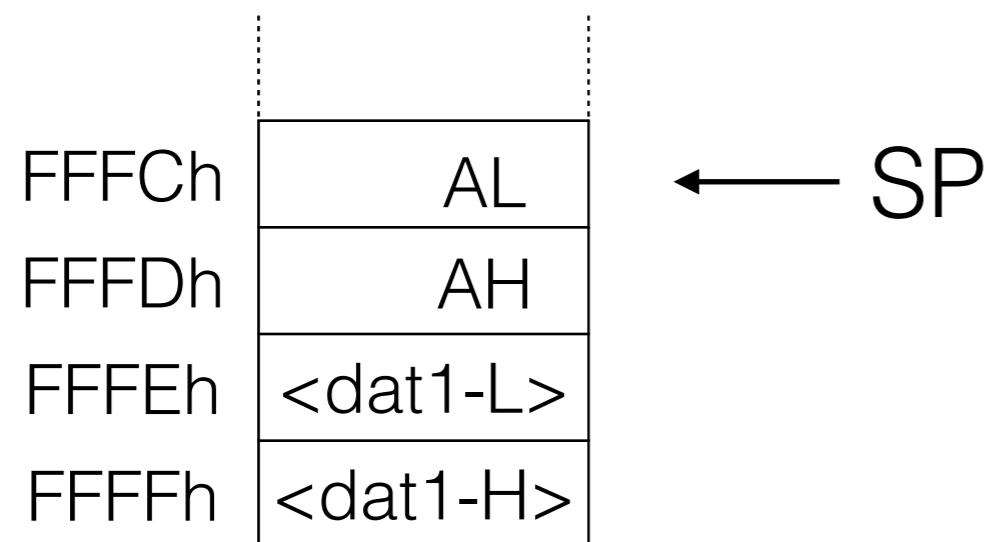
POP <16b register>  
POP <16b Memory>

# הכנסה למחסנית



- PUSH AX :  
 $sp \leftarrow sp - 1$  •  
 $[sp] \leftarrow ah$  •  
 $sp \leftarrow sp - 1$  •  
 $[sp] \leftarrow al$  •
- בעת המחסנית מכילה AX

# שליפה מהמחסנית



- POP BX יבצע:  
 $bl \leftarrow [sp]$  •  
 $sp \leftarrow sp + 1$  •  
 $bh \leftarrow [sp]$  •  
 $sp \leftarrow sp + 1$  •
- בעת BX יוכל את הערך השמור של AX.
- **שים לב:** הזיכרונו לא "מתנתקה"

# דוגמה: שימוש במחסנית לשמרות זמנית של נתונים

- ניתן לדחוף אוגרים למחסנית ע"מ "לשחרר" את האוגרים לשימוש חלופי

PUSH AX  
PUSH BX

DO\_SMTG:

<long code  
that uses  
AX,BX>

...

POP BX  
POP AX

- שים לב למבנה LIFO - האחרון שנדחק למחסנית הוא הראשון לשילפה

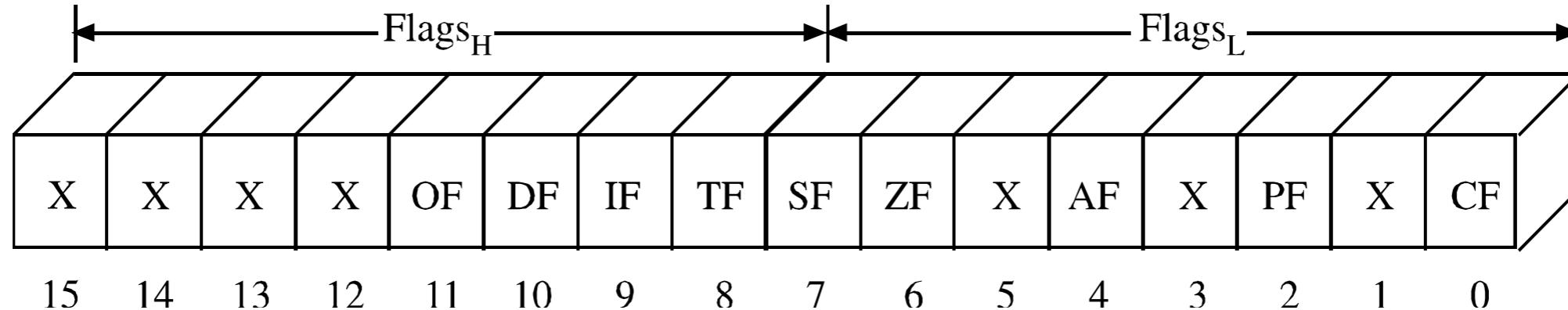
- למשל: שימרת CX בלולאה מקווננת

# שמירת דגלים: PUSHF/POPF

- ניתן לדחוף/לשלוף את אוגר הדגלים אל/מ המחסנית ע"י הפקודה

**PUSHF**

**POPF**



# דוגמא: בדיקת דגלים

- נניח שנרצה לבצע קוד מסויים אמ"מ  $SF=CF=OF=1$ .  
איך נבצע?
- פתרון א: ביצוע SJ ואחריו JC ואחריו OJ
- פתרון ב:

PUSHF

POP AX

AND AX, 0881h

CMP AX, 0881h

JE <yes>

# דוגמא: שינוי דגלים

- נניח שנרצה לשנות את דגל TF (ביט מס' 8)  
(הדלקתו מעביר את המעבד לモצב DEBUG)
- לא קיימת פקודה שמשנה את ערך הדגל ישירות
- איך נבעצע?

# הגדרת המחשבנית

- הגדרת המחשבנית מתבצעת ע"י הפקודה:

.stack <size>

- או באופן מפורש:  
(אפשר גם קביעת תוכו למחשבנית)

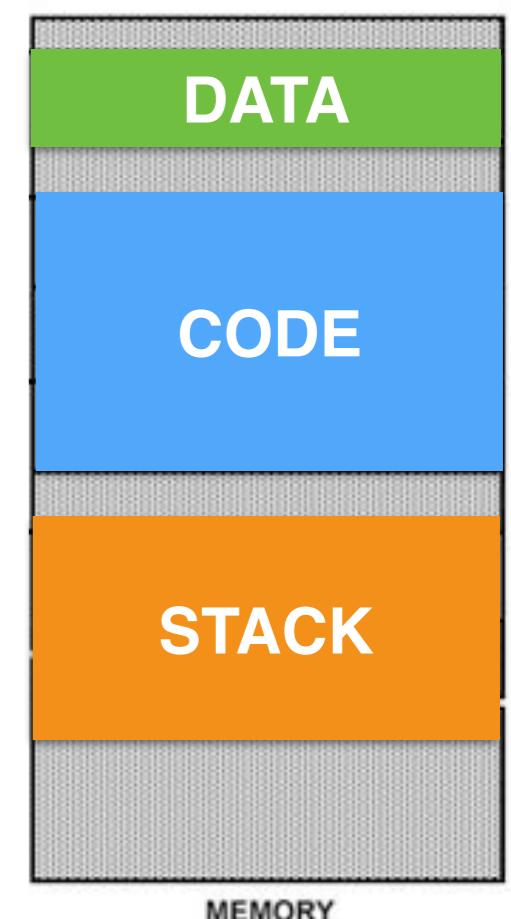
<name> **SEGMENT STACK**

... <def of stack> e.g.,  
segval DB 2500 DUP(?)

<name> **ENDS**

# גִּלְישָׁת מַחֲסָנִית

|             |          |                            |
|-------------|----------|----------------------------|
| .data       |          | תחילת מקטע נתונים ;        |
| INFO        | DW 1000h | הגדרת משתנה בשם INFO ;     |
| .stack      | 100      | הказאת 100 בתים למחסנית ;  |
| .code       |          | תחילת מקטע קוד ;           |
| START:      |          |                            |
| MOV AX, 0   |          | שים 0 באוגר AX ;           |
| MOV BX, 0   |          | שים 0 באוגר BX ;           |
| MOV DS, AX  |          | העתק את אוגר AX לאוגר DS ; |
| MOV AH, 4CH |          | סיום התוכנה (חזרה למ"ה) ;  |
| INT 21h     |          |                            |
| END START   |          |                            |



# גليسית מחסנית

- כאשר "דוחפים" למחסנית יותר נתונים ממה שהוקצה למקטע המחסנית - תבוצע **גليسית מחסנית**
- המחסנית עלולה "לדרוס" נתונים אחרים בזיכרון (או לדרוס את סוף המחסנית, למשל אם הוקזו 64k כתובות)
- שגיאה כנ"ל עשויה לגרום לקריסת התוכנית קשה לזיהוי...

# פקודות מכונה

**PUSH = Push:**

Register/Memory

|               |               |
|---------------|---------------|
| 1 1 1 1 1 1 1 | mod 1 1 0 r/m |
|---------------|---------------|

Register

|               |
|---------------|
| 0 1 0 1 0 reg |
|---------------|

Segment Register

|                 |
|-----------------|
| 0 0 0 reg 1 1 0 |
|-----------------|

**POP = Pop:**

Register/Memory

|               |               |
|---------------|---------------|
| 1 0 0 0 1 1 1 | mod 0 0 0 r/m |
|---------------|---------------|

Register

|               |
|---------------|
| 0 1 0 1 1 reg |
|---------------|

Segment Register

|                 |
|-----------------|
| 0 0 0 reg 1 1 1 |
|-----------------|

---

**PUSHF = Push Flags**

|                 |
|-----------------|
| 1 0 0 1 1 1 0 0 |
|-----------------|

**POPF = Pop Flags**

|                 |
|-----------------|
| 1 0 0 1 1 1 0 1 |
|-----------------|

# מחסנית: סיכום

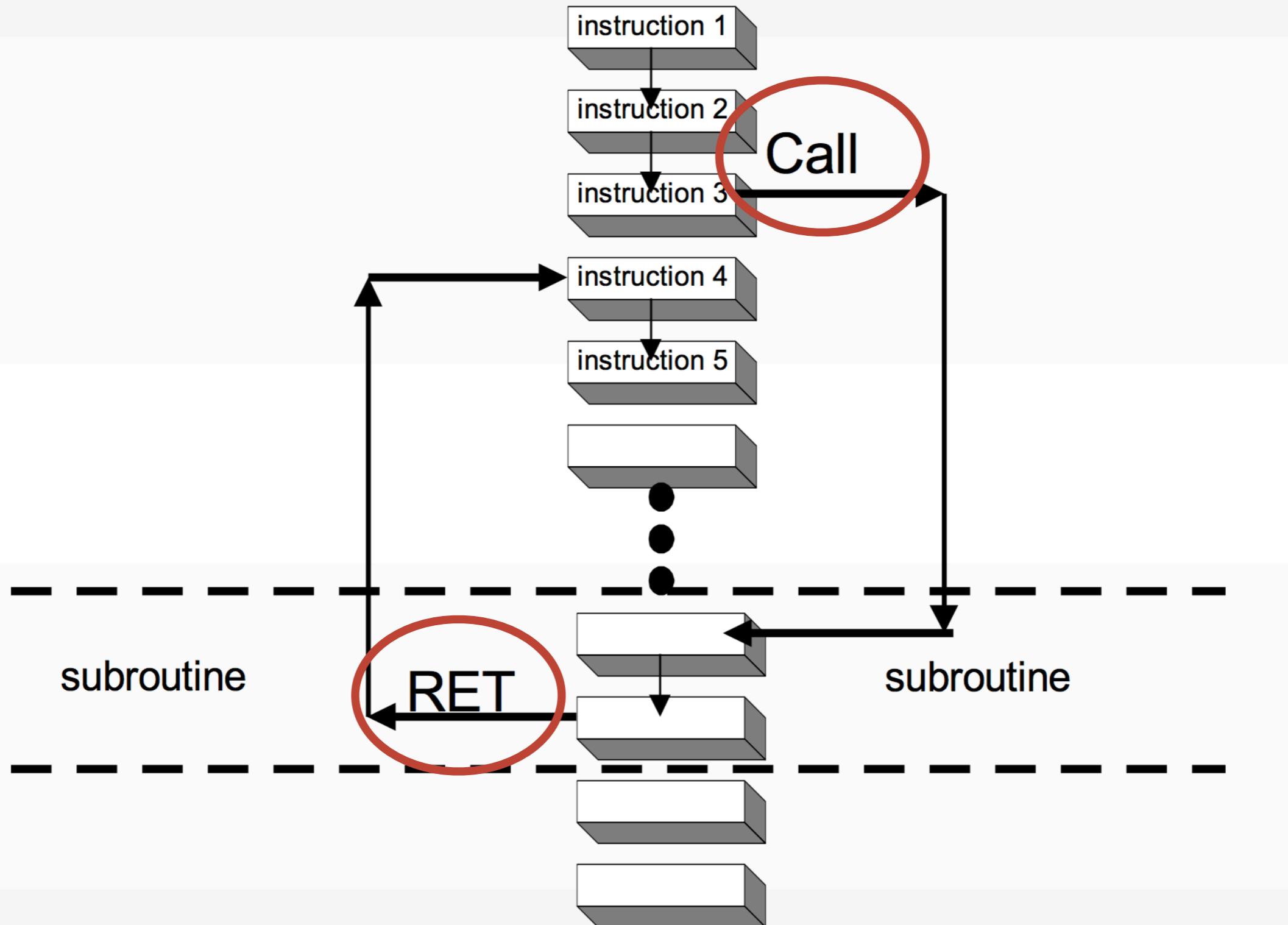
- **המחסנית** - אזור בזיכרון המשמש לשמירה זמנית של נתונים **בשיטת Last In First Out**
- האוגר SP מצביע לנตอน **בראש המחסנית**
- דחיפה/שליפה למחסנית ע"י הפקודות PUSH/POP
- רשומות בגודל **16 בית** בלבד!
- שמירת/שינוי דגלים ע"י PUSHF/POPF
- ביצוע יותר מדי POP (ללא PUSH) עלול לגרום **לגלישה**

# תתי שגרות

# תת-שגרה (subroutine)

- **תת-שגרה:**
- **אוסף פקודות המקבצות "יחדיו" לביצוע פעולה מסוימת**
- **שימושי כאשר צריך לחזור על אותה פעולה מספר פעמים, במקומם לשכפל את הקוד**
- **תכנות מודולרי - "קיבוץ" פקודות המבצעות פעולה מסוימת והтиיחסות אליהן כמקשה אחת**
- **שימוש באותו שגרה בתוכניות אחרות**
- **כתיבת שגרות ע"י צוות תוכנה גדול**

# שימוש בתתי-שגרות



# שימוש בתתי-שגרות

- דוגמא:  
הנח שגרת `DELAY` שיוצרת השניה של מספר שניות,  
לפי אוגר `AL`

MAIN: MOV AL, 5  
CALL DELAY

<do stuff>

MOV AL, 10  
CALL DELAY

# מבנה תת-שגרה

<name> **PROC** [NEAR/FAR]

<inst 1>

<inst 2>

...

**RET**

<name> **ENDP**

- לכל שגרה יש שם ייחודי  
שימוש בקריאה לשגרה
- השגרה תחומה בהוראות  
PROC/ENDP
- לרוב, הפקודה الأخيرة  
בשגרה תהיה RET  
שתפקידה לחזור לביצוע  
התוכנית הראשית

# דוגמא



**DELAY PROC NEAR**

<MUL 10> ; AX will hold #of 0.1 sec to wait

AGAIN : PUSH CX

XOR CX,CX

wait100msec: NOP

LOOP wait100msec ; takes about 0.1sec @5MHZ

POP CX

DEC AX

JNZ AGAIN

RET

**DELAY ENDP**

# מבנה תחת-שגרה

- ניתן להגדיר שגרת FAR או NEAR או
- מסמל למהדר האם השגרה נמצאת בסגמנט הקוד הנוכחי או בסגמנט אחד
- ברירת המחדל היא NEAR (גם אם לא נכתב במפורש)

# קריאה לשגרה

- קריאה לשגרה מתבצעת ע"י הפקודה

CALL <label>

- מיצעת:

1. דחיפת כתובת החזרה **למחסנית** (IP או IP:CS)
2. קפיצה לתחילת השגרה JMP <label>



FAR



NEAR

# חזרה מshort

RET

- חזרה מshort ע"י
- מבצעת קפיצה לכתובת **שבראש המחסנית**
- אם shortNEAR, מבצעת [SP]  
(וגם  $SP \leftarrow SP+2$ )
- אם shortFAR, מבצעת [SP]  
(כמו "POP IP" שלאחריו ".(POP CS")

# קריאה וחזרה משגרה

קריאה לשגרה:

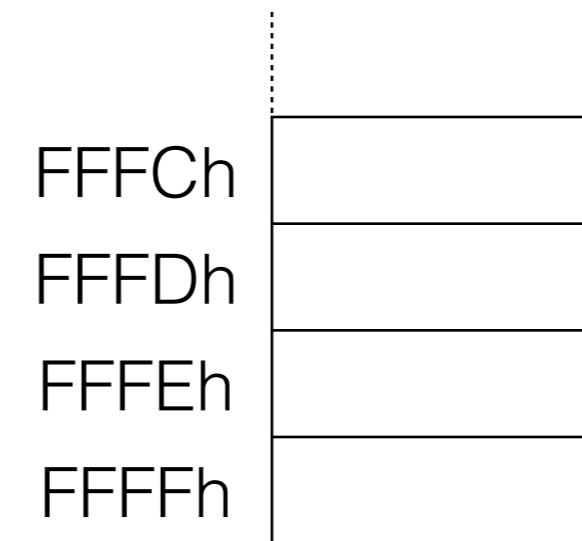
0000 CALL 2000h ← IP

0003 NOP

...

2000 NOP

2001 RET



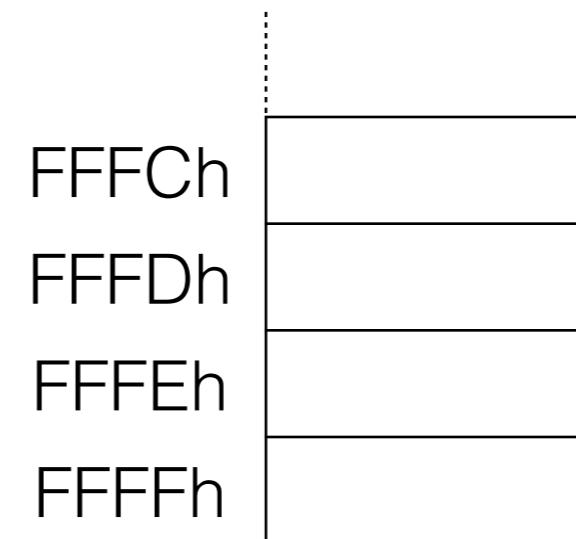
# קריאה וחזרה משגרה

קריאה לשגרה:

0000      CALL 2000h ← IP  
0003      NOP

...

2000      NOP  
2001      RET



← SP

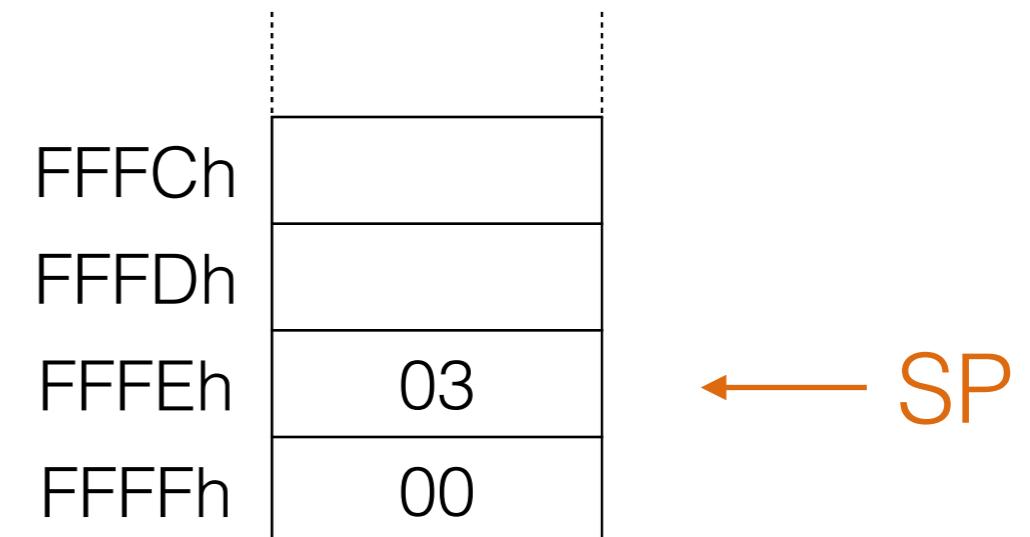
# קריאה וחזרה משגרה

קריאה לשגרה:

0000      CALL 2000h ← IP  
0003      NOP

...

2000      NOP  
2001      RET



# קריאה וחזרה משגרה

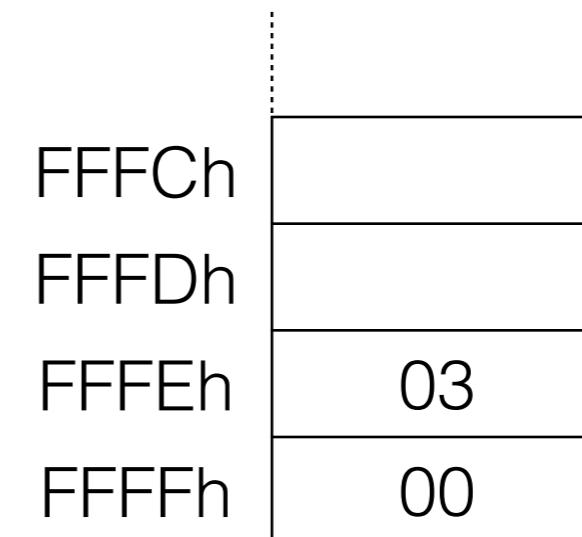
קריאה לשגרה:

|      |            |
|------|------------|
| 0000 | CALL 2000h |
| 0003 | NOP        |

...

|      |     |
|------|-----|
| 2000 | NOP |
| 2001 | RET |

← IP



# קריאה וחזרה משגרה

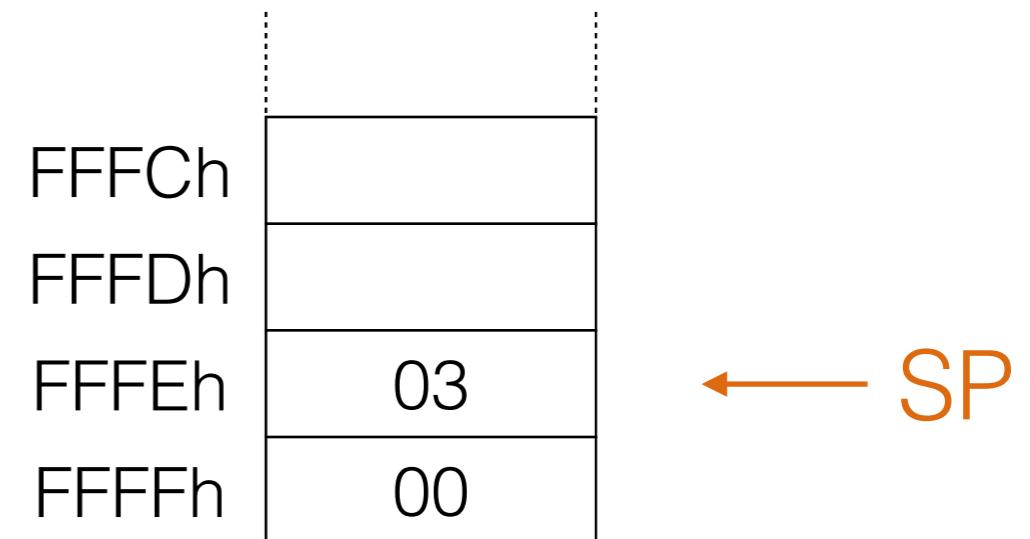
חזרה משגרה:

|      |            |
|------|------------|
| 0000 | CALL 2000h |
| 0003 | NOP        |

...

|      |     |
|------|-----|
| 2000 | NOP |
| 2001 | RET |

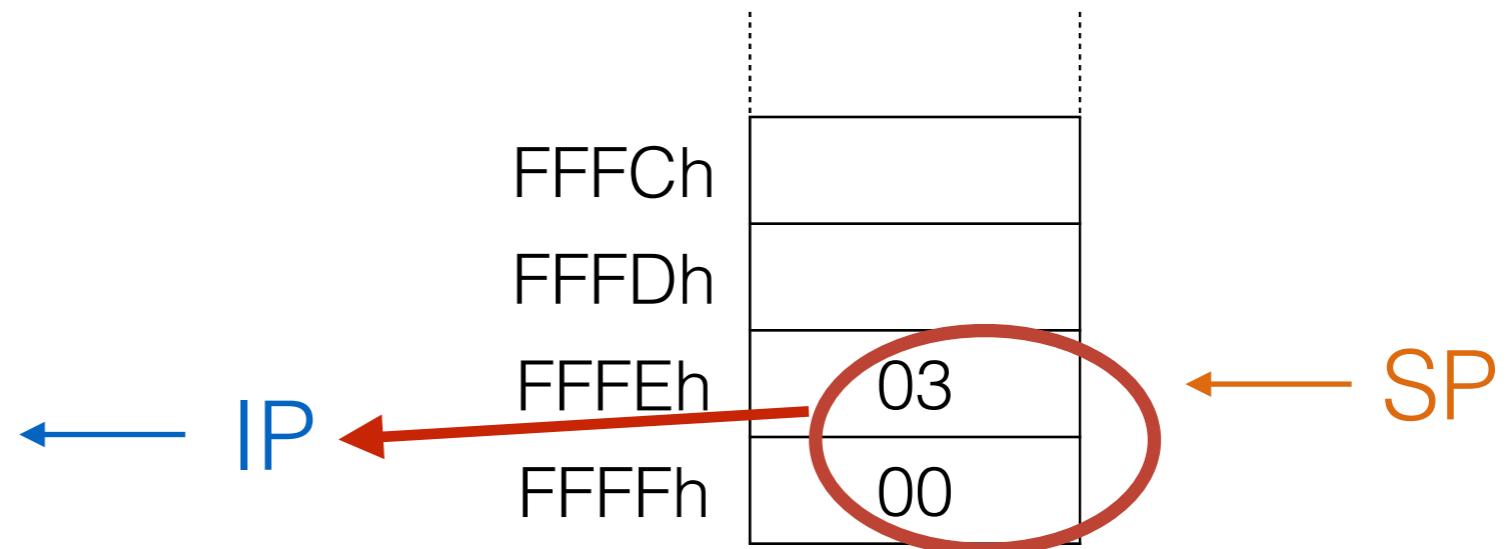
← IP



# קריאה וחזרה משגרה

חזרה משגרה:

|      |            |
|------|------------|
| 0000 | CALL 2000h |
| 0003 | NOP        |
| ...  |            |
| 2000 | NOP        |
| 2001 | RET        |



# קריאה וחזרה משגרה

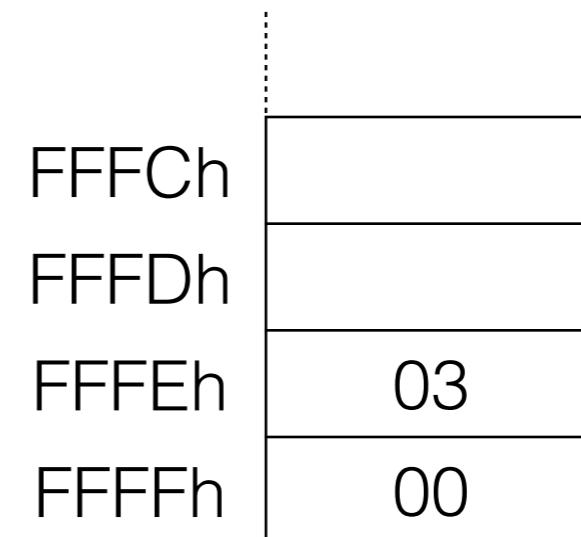
חזרה משגרה:

|      |            |
|------|------------|
| 0000 | CALL 2000h |
| 0003 | NOP        |

← IP

...

|      |     |
|------|-----|
| 2000 | NOP |
| 2001 | RET |



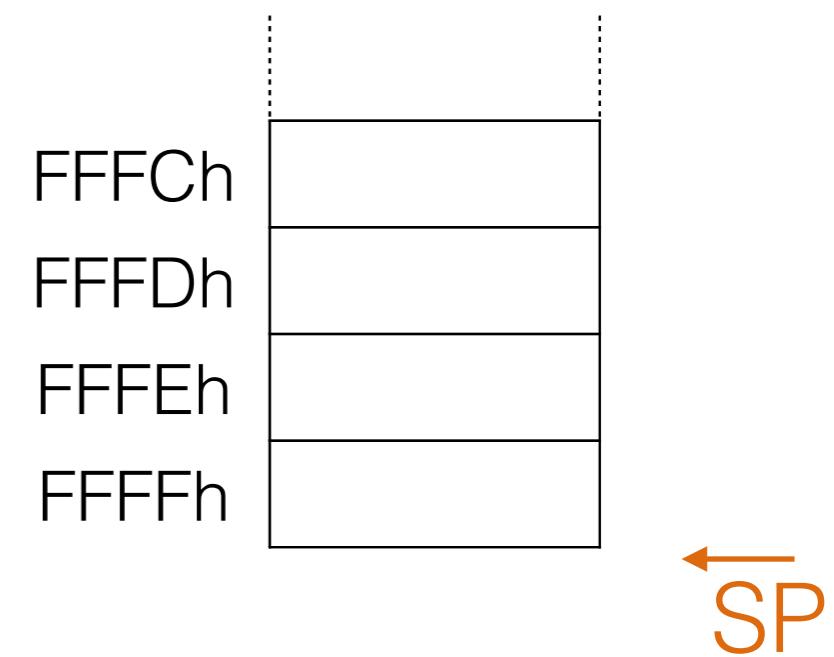
← SP

# קריאה וhzורה משגירה

בשגרת FAR הכתובת נשמרת בצורה CS:IP

|           |               |      |
|-----------|---------------|------|
| 1000:0000 | CALL ext_test | ← IP |
| 1000:0003 | NOP           |      |
| ...       |               |      |

|           |           |     |
|-----------|-----------|-----|
| F300:2000 | ext_test: |     |
|           |           | NOP |
| F300:2001 |           | RET |

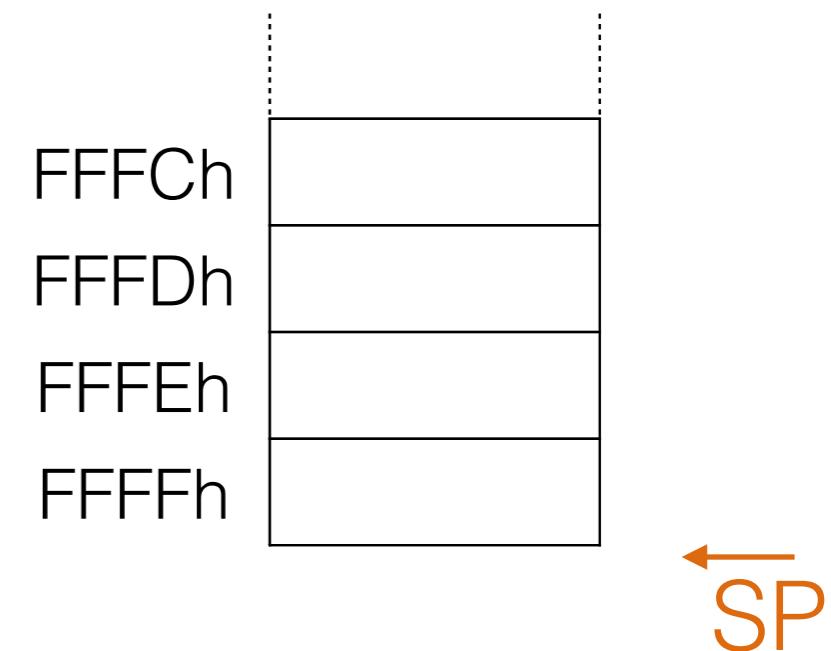


# קריאה וhzורה משגרה

בשגרת FAR הכתובת נשמרת בצורה CS:IP

1000:0000      CALL ext\_test      ← IP  
1000:0003      NOP  
  
...

F300:2000      ext\_test:  
F300:2001      NOP  
F300:2002      RET

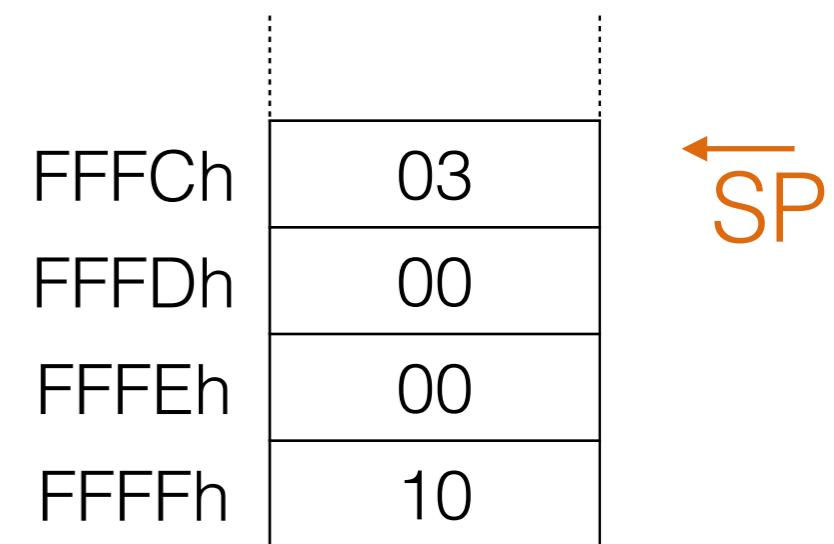


# FAR קרייה וחזרה משגרה

בשגרת FAR הכתובת נשמרת בצורה CS:IP

|           |               |      |
|-----------|---------------|------|
| 1000:0000 | CALL ext_test | ← IP |
| 1000:0003 | NOP           |      |
| ...       |               |      |

|           |           |     |
|-----------|-----------|-----|
| F300:2000 | ext_test: |     |
|           |           | NOP |
| F300:2001 |           | RET |



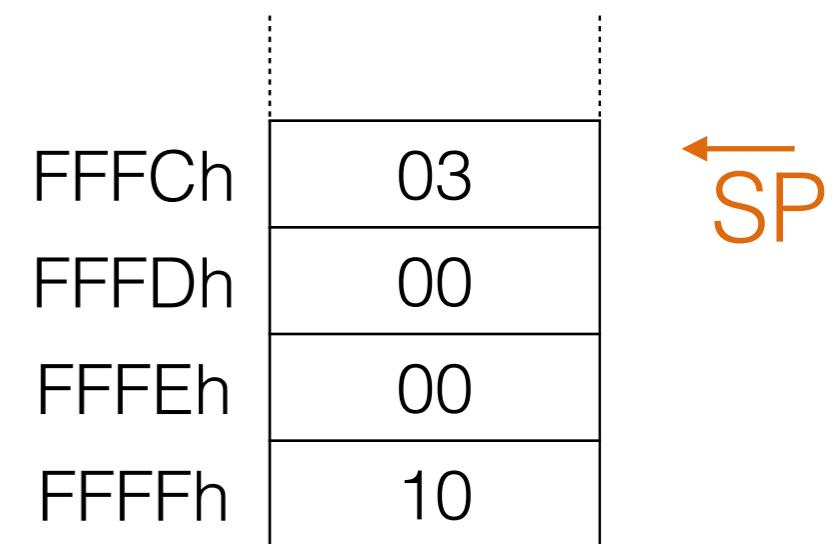
# FAR קרייה וחזרה משגרה

בשגרת FAR הכתובת נשמרת בצורה CS:IP

|           |               |
|-----------|---------------|
| 1000:0000 | CALL ext_test |
| 1000:0003 | NOP           |
| ...       |               |

|           |           |     |
|-----------|-----------|-----|
| F300:2000 | ext_test: |     |
|           |           | NOP |
|           |           | RET |

IP ←



# מייעו CALL/RET

- לפקודה CALL מייעונים כמו JMP

**CALL = Call:**

Direct within Segment

|          |          |           |
|----------|----------|-----------|
| 11101000 | disp-low | disp-high |
|----------|----------|-----------|

Indirect within Segment

|          |               |
|----------|---------------|
| 11111111 | mod 0 1 0 r/m |
|----------|---------------|

Direct Intersegment

|          |            |             |
|----------|------------|-------------|
| 10011010 | offset-low | offset-high |
|----------|------------|-------------|

Indirect Intersegment

|          |               |          |
|----------|---------------|----------|
|          | seg-low       | seg-high |
| 11111111 | mod 0 1 1 r/m |          |

- פקודת RET - קיים opcode שונה לNEAR מאשר FAR

**RET = Return from CALL:**

Within Segment

|          |
|----------|
| 11000011 |
|----------|

Intersegment

|          |
|----------|
| 11001011 |
|----------|

# שגרות: רב- כניסה (re-entrancy) והעברת פרמטרים

# (non-) Reentrancy

- שגרות יכולות לקרוא לשגרות אחרות או לעצמן
- האם ניתן לקרוא לשגרה במהלך הריצה של אותה השגרה?
- אם כן, השגרה נקראת **re-entrant**, אחרת - **non-reentrant**
- **הערה:**
  - בתוכנה סידرتית - קריאה חוזרת היא רק רקורסיבית
  - כישוף פסיקות/multitasking: מספר "תוכנות" רצות במקביל ויכולות כל-אותה להפעיל "עותק" של השגרה.

# (non-) Reentrancy

```

DELAY PROC NEAR
 <MUL 10> ← PUSH AX
AGAIN : PUSH CX
 XOR CX,CX
wait100msec: NOP
 LOOP wait100msec
 POP CX
 DEC AX
 JNZ AGAIN
 RET ← POP AX
DELAY ENDP

```

- דוגמא - האם השגרה ?reentrant הבאה לא!
- קרייה חוזרת תגרום לשיבוש ערך AX
- איך להפוך ל?reentrant

# תנאים ל-reentrancy

- על מנת ששגרה תהיה reentrant היא צריכה לקיים את כל שלושת "כלי האצבע" הבאים:
  1. שמירת כל האוגרים בהם עושים שימוש
  2. שימוש בזיכרון משותף בצורה אוטומית ("הכל או כלום") או: נמנעת שימוש בזיכרון משותף
  3. לא קוראת לשגרות non-reentrant

**למה זה  
בכל חשוב?**

# העברת פרמטרים לשגרה

- לעתים נרצה להעביר פרמטרים לשגרה, למשל, כמות זמן המתנה בשגרת `DELAY` שראינו לעיל
- שיטות העברת נתונים:
  - דרך אוגר
  - זיכרון נתונים
  - מחסנית

# העברת פרמטרים לשגרה

- **שיטת א':** העברת דרך אוגר
  - הקוד הקורא מעדכן את האוגרים בערך המתאים
  - לדוגמה: `DELAY AX,10` ו`MOV AX,[Y]` לשגרה
- **יתרונות:** קל לניהול, אוטומטי (בלעד מטבחעת שמירת אוגרים בשגרה או בקוד הקורא),
- **חסרונות:** כמות הפרמטרים מוגבלת

# העברת פרמטרים לשגרה

- **שיטה ב':** העברת פרמטרים בזיכרון הנתוניים.
- הפרמטרים נשמרים בהיסט קבוע ב-DS.

```
.data DelayCount DB ?
```

- **לדוגמה:**

```
...
```

```
.code
```

```
DELAY PROC
```

```
 PUSH AX
```

```
 MOV AX, DelayCount
```

```
 <delay code as before>
```

```
 POP AX
```

```
 RET
```

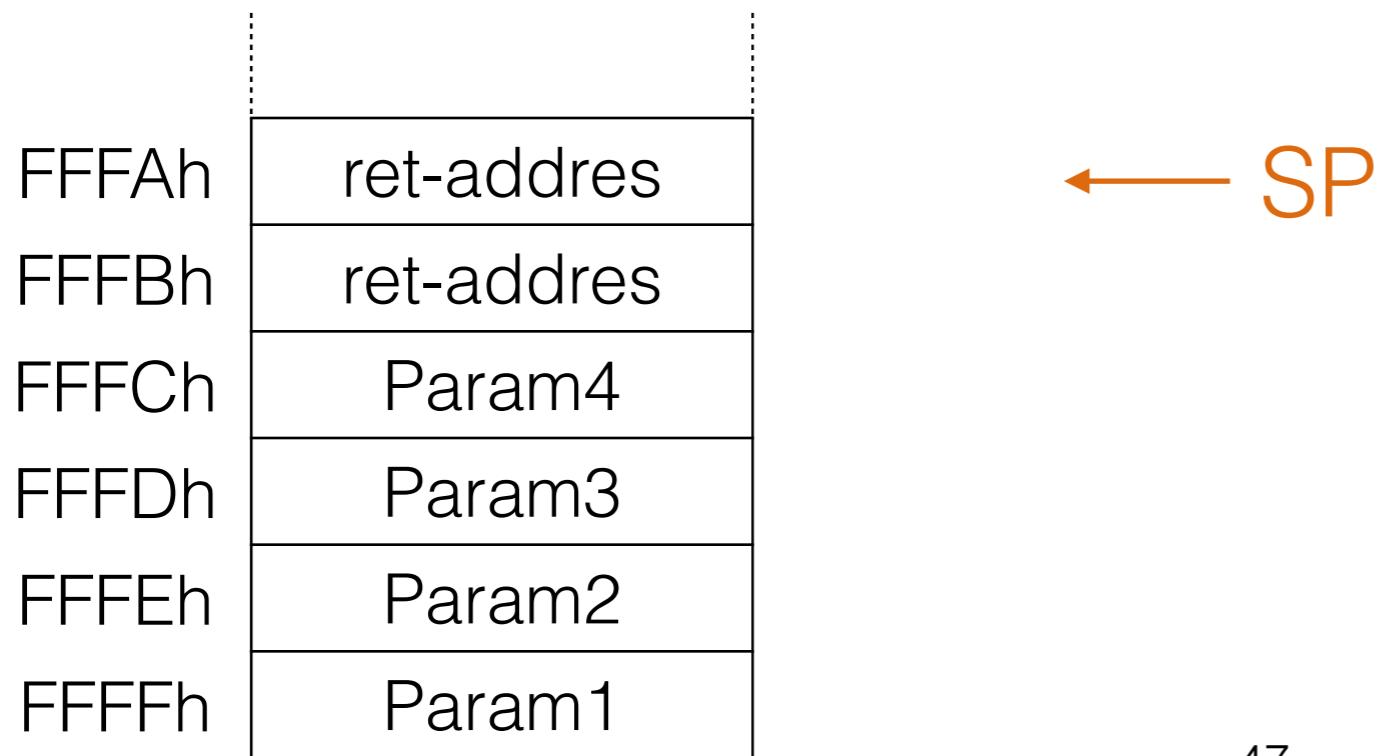
```
DELAY ENDP
```

# העברה פרמטרים לשגרה

- **שיטה ב':** העברת פרמטרים בזיכרון הנתונים.
- הפרמטרים נשמרים בהיסט קבוע ב-DS.
- **יתרונות:** אין הגבלה על כמות פרמטרים; נוח לשימוש ad-hoc
- **חסרונות:**
  - זיכרון משותף - עלול להפוך לשגרה ל-non-reentrant
  - פחות מודולרי, כותב התוכנה הראשית צריך לדעת את קוד השגרה ואיך לקרוא כל פרמטר
  - התנגשות בין שמות: שתי תת-שגרות שונות (שנכתבו ע"י אנשים שונים) עלולות להשתמש באותו label ...

# העברת פרמטרים לשגרה

- **שיטת ג':** העברת פרמטרים במחסנית
- לפני הקראיה לשגרה, הקוד הקורא דוחף את כל הפרמטרים למחסנית בסדר נتون



(מצב אחרי ביצוע CALL)

# העברת פרמטרים לשגרה

- **שיטת ג':** העברת פרמטרים במחסנית
- לפני הקריאה לשגרה, הקוד הקורא דוחף את כל הפרמטרים למחסנית בסדר נתון
- **יתרונות:**
  - קל לימוש וגישה
  - אוטומטי - כל שגרה פועלת עצמאית עם הנתונים שלה (אין התנגשות)
- **חרונות:** זהירות מגليسת מחסנית

# העברה פרמטרים לשגרה

- **שיטת ג':** העברת פרמטרים במחסנית
- העברה **לפי ערך** — המחסנית שומרת את הערך של הפרמטר שרוצים להעביר
- העברה **לפי reference** — המחסנית שומרת את כתובות הזיכרון בה נמצא ערך הפרמטר
- נדגים ע"י שגרה לחיבור 3 מספרים הניתנים כפרמטרים ושמירתם ב-X.

# העברה לפיעර

הקוד הקורא

```
.data
 i dw ?
 j dw ?
 k dw ?

.code
 PUSH i
 PUSH j
 PUSH k

 CALL ADD3
```

השגרה

```
ADD3 PROC
 POP DX ;; save return address
 POP AX ;; get k
 POP BX ;; get j
 ADD AX, BX ;; j+k
 POP BX ;; get i
 ADD AX, BX ;; i+j+k
 PUSH DX ;; put the return
 address back!

 RET
ADD3 ENDP
```

# העברה לפי ערד

- חסרונות השיטה לעיל:
- דרשו את X , BX .  
(אם ניתן היה לעשות להם PUSH לפניהם?)
- מצרייך ניהול IP (כתבת חזרה) באופנו פרטני
- שיטה יותר טובה -  
ע"י שימוש במצבייע בסיס המחסנית BP

# העברה לפיעර

הקוד הקורא

```
.data
i dw ?
j dw ?
k dw ?
```

השגרה

```
ADD3 PROC
 MOV BP,SP
```

```
.code
PUSH i
PUSH j
PUSH k
```

```
 MOV AX, [BP+2] ;; get k
 ADD AX, [BP+4] ;; add j
 ADD AX, [BP+6] ;; add i
 RET 6
ADD3 ENDP
```

CALL ADD3

# העברה לפיעර

הקוד הקורא

חזרה מהשגרה +  
שינוי SP בערך הרשום  
==

מחיקת הפרמטרים  
מהמחסנית בסיום  
הfonkציה

PUSH j  
PUSH k

CALL ADD3

השגרה

ADD3 PROC  
MOV BP,SP

MOV AX, [BP+2] ;; get k  
ADD AX, [BP+4] ;; add j  
ADD AX, [BP+6] ;; add i  
**RET 6**  
ADD3 ENDP

# העברה לפיעර

- יתרונות השיטה לעיל:
- אין ניהול של IP באופן פרטני
- אין התבസות על SP - יכול לשתנות (עקב דחיפה של נתונים אחרים למחסנית)

# העברה לפি reference

- הקוד הקורא מכמה מקומות בזיכרון לפרקטי, ו מעביר את כתובות הזיכרון (BS) לשגרה
- אפשר לשגרה לשנות את ערך תא הזיכרון (לעומת העברה לפி ערך)

# העברה לפי reference

מעביר את כתובות הנתון  
וללא את תוכנו

הקוד הקורא

.data

iParam dw 0

jParam dw 0

kParam dw 0

MOV AX, **offset** iParam

PUSH AX

...

CALL ADD3

# העברה לפי reference

| הקוד הקורא                   | השגרה                         |
|------------------------------|-------------------------------|
| .data                        |                               |
| iParam dw 0                  | ADD3 PROC                     |
| jParam dw 0                  | MOV BP,SP                     |
| kParam dw 0                  |                               |
| MOV AX, <b>offset</b> iParam | MOV BX, [BP+2] ; get k's addr |
| PUSH AX                      | MOV AX, [BX] ; add k          |
| ...                          | ...                           |
|                              | <b>RET 6</b>                  |
| CALL ADD3                    | ADD3 ENDP                     |

# meshutanim lokomiyim bishagrah

- בכתיבה מודולרית, כותב קוד השגרה לא יודע באיזה תוכנית היא תרוץ, ומה מבנה הסגמנטים של התוכנית הקוראת
  - במקום להקצות זיכרון data לשגרה (**סטטי!**), ניתן להקצות זיכרון מקומי על המחשבנית (**динامي**).
  - **למשל ע"י**

**MOV BP, SP** שמירת בסיס המחסנית;  
**SUB SP, 8** ג-8 בתים על המחסנית ;

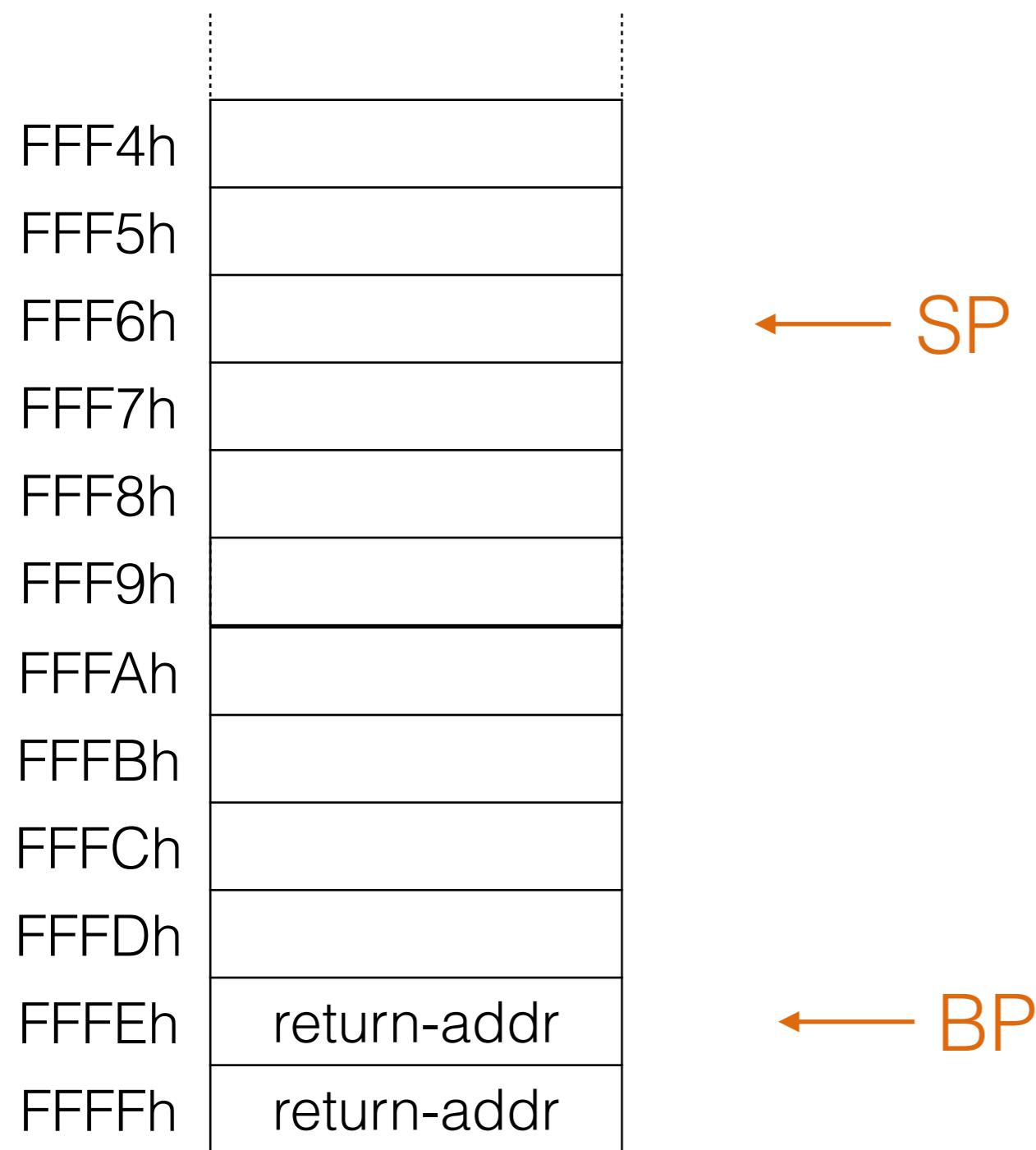
# משתנים מקומיים בשגרה

- בכתיבה מודולרית, כותב קוד השגרה לא יודע באיזה תוכנית היא תרוץ, וכך מוגדרים המשתנים של התוכנית הקורט.
- **הגדנו את המחסנית באופן פיקטיבי, ניתן להשתמש בשטח לצרכי השגרה**
- במתוך השגרה ניתן לרשום ערך קבוע להvariable.
- **למשל ע"י**

MOV BP, SP  
SUB SP, 8

שמירת בסיס המחסנית;  
הקצת מקום ל-8 בתים על המחסנית ;

# משתנים מקומיים בשגרה



- לאחר הקצת השטח, נשתמש בזיכרון מייעון עקיף לבסיס המחסנית

MOV AX, [BP-2]  
INC [BP-8]  
PUSH ...  
...  
MOV SP, BP  
RET

- שים לב: השגרה יכולה להמשיך לדוחף למחסנית ללא הפרעה (נתונים ייתווסף מעל השטח המוקצה!)

# תתי-שגרות: סיכום

- שגרה היא קטע קוד מודולרי המשמש לפעולות חוזרות
- קריאה לשגרה ע"י **CALL** שומרת כתובת חזורה במחסנית ו קופצת אל השגרה. **RET** מבצעת ההפך
- העברת פרמטרים לשגרה - עדיף על **המחסנית**.
- שימוש ב- **BP** כמצבייע בסיס ע"מ לא להפריע לפעולות המחסנית התקינה
- ניתן להעביר **ערך** (קריאה בלבד) או **מצבייע** (קריאה/כתיבה)
- יש להזהר בעת הפעלה חוזרת של שגרה מתוך עצמה:  
שגרה non-reentrant תתנהג באופן לא צפוי

# תת-שגרה: תרגיל

- בראצוני לחשב את מספר פיבונאצ'י ה-50.
- מה **הבעיות** בשגרה הבאה לחישוב מספר פיבונאצ'י?
- **תזכורת:**

$$\text{fib}(2) = \text{fib}(1) = 1$$

$$\text{fib}(N) = \text{fib}(N-1) + \text{fib}(N-2)$$

קוד התוכנית הקוראת:

```
MOV BX, 50
CALL FIB
```

# תת-שגרה: תרגיל

```
FIB PROC ; takes N in BX. outputs fib(N) in AX
 PUSH DX
 DEC BX ; BX has N-1
 [CMP BX,2 | JB end1 |..] ; assume base case works well!
 CALL FIB ; AX -> fib(N-1)

 MOV DX, AX ; remember mid result
 DEC BX ; BX has N-2
 CALL FIB ; AX -> fib(N-2)

 ADD AX, DX ; AX -> fib(N-2)+fib(N-1)
 POP DX ; restore DX
 RET
FIB ENDP
```

קוד התוכנית הקוראת:

```
MOV BX, 50
PUSH BX
CALL FIB
```

# תת-שגרה: תרגיל

```
FIB PROC ; takes N on stack. outputs fib(N) in AX
 PUSH BX
 PUSH DX
 MOV BX, [SP+2] ; BX keeps N
 DEC BX ; BX has N-1
 [CMP BX,2 | JL end1 |..] ; deal with base case
 PUSH BX ; put N-1 on stack
 CALL FIB ; AX -> fib(N-1)
 MOV DX, AX ; remember AX
 DEC BX ; BX has N-2
 PUSH BX
 CALL FIB ; AX -> fib(N-2)
 ADD AX, DX ; AX -> fib(N-2)+fib(N-1)
 POP DX
 POP BX
 RET 2
FIB ENDP
```

קוד התוכנית הקוראת:

```
MOV BX, 50
PUSH BX
CALL FIB
```

# תת-שגרה: תרגיל

```
FIB PROC ; takes N on stack. outputs fib(N) in AX
 PUSH BX
 PUSH DX
 MOV BX, [SP+6] ; BX keeps N (really, must use BP!)
 DEC BX ; BX has N-1
 [CMP BX,2 | JL end1 |..] ; deal with base case
 PUSH BX ; put N-1 on stack
 CALL FIB ; AX -> fib(N-1)
 MOV DX, AX ; remember AX
 DEC BX ; BX has N-2
 PUSH BX
 CALL FIB ; AX -> fib(N-2)
 ADD AX, DX ; AX -> fib(N-2)+fib(N-1)
 POP DX
 POP BX
 RET 2
FIB ENDP
```

# תת-שגרה: תרגיל 2

- מה יקרה בהרצת הקוד הבא? (הניחו,  $N=500$ )

```
REC PROC ; takes N on stack, recurse N times
 PUSH BP
 MOV BP, SP
 PUSH CX
 MOV CX, [BP+4] ; retrieve N
 SUB SP, CX ; make space of N bytes on stack
 DEC CX [N bytes implicitly used, e.g., sort]
 JZ stop
 PUSH CX ; recurse on N-1
 CALL REC
stop: ADD SP, [BP+4] ; restore the stack pointer
 POP CX
 POP BP
 RET 2
REC ENDP
```

# אורים: מקורות

- B. Brey, The Intel Microprocessor, 2009
- Intel 8086 spec, INTEL, 1990
- **מחסנית(מבנה נתונים)**: Wikipedia
- ארגון המחשב ושפת סר, ברק גונן, גבהים, מרכז הסייבר הצבאי 2015
- CSE 307- Microprocessor, Mohd. Moinul Hoque, Lecturer, CSE, AUST

# קלט-פלט ופסיכות

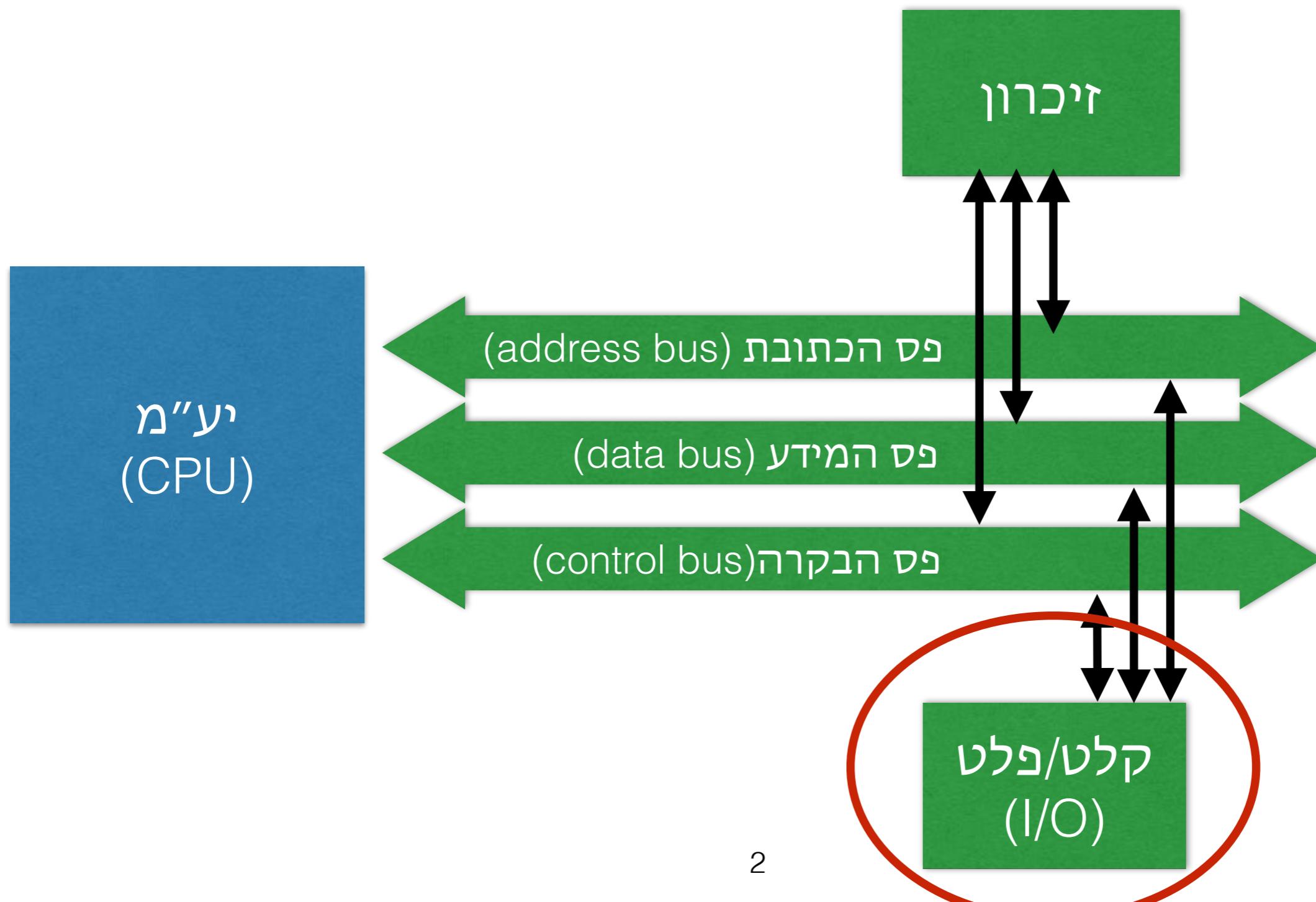
מיקромעבדים ושות אסמלר 83-255-83



אוניברסיטת בר-אילן  
Bar-Ilan University

# תזכורת: מבנה "מחשב"

## ארQUITטורת פון-נוימן



# תזכורת: מבנה "מחשב"

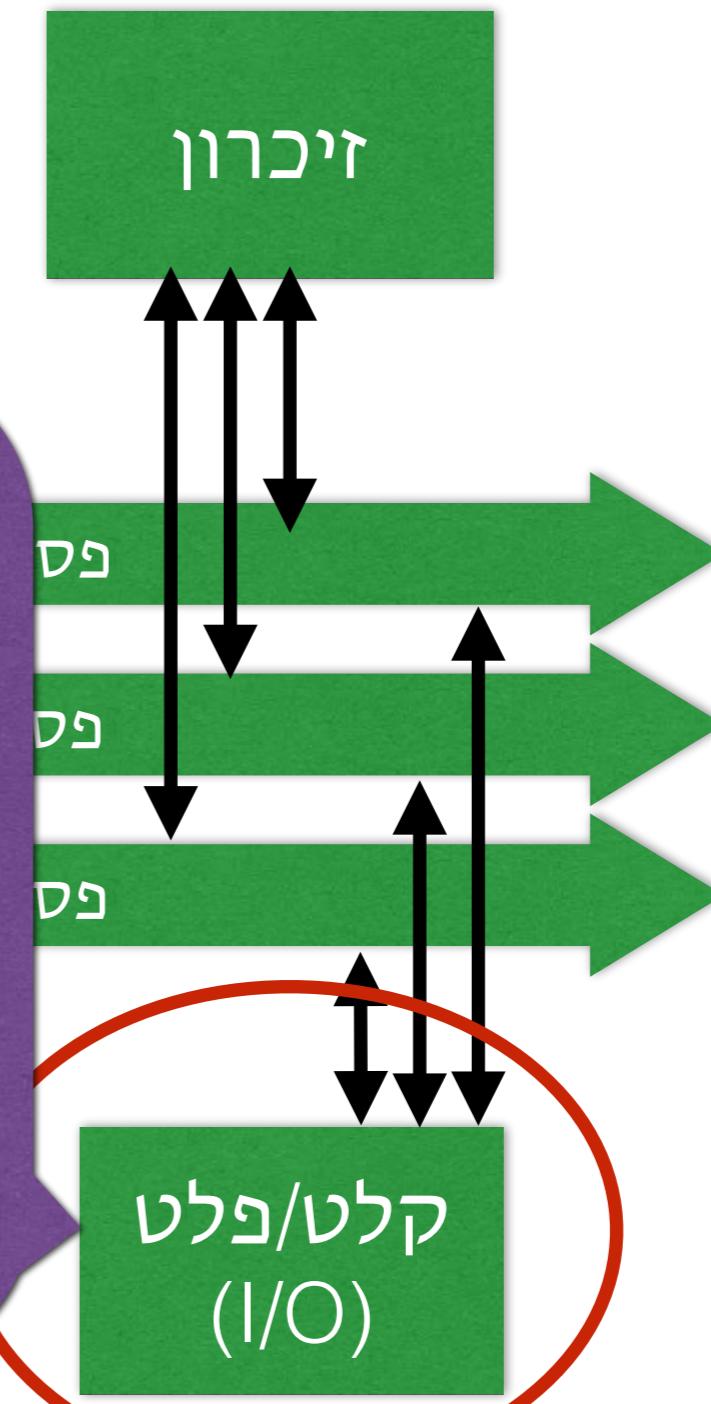
## ארQUITטורת פון-נוימן

**אצעי קלט/פלט ב"מחשב":**

- מקלדת, עכבר, צג, מדפסת, כרטיס רשת, USB, ...

**:Embedded אצעי קלט/פלט במערכת**

- סנзорים, מנוע, USB, LED, ...
- אנטנה(מודם), A/D, ...



# אמצעי קלט/פלט

- המעבד יכול להתmeshק לרכיבים היקפיים (פריפריאליים)
- לכל רכיב היקפי תמכנו התנהגות אחרת וממשק פקודות שונה
- לרוב, התקן היקפי יוכל בקר נפרד המאפשר ביצוע קריאה/כתיבה של המעבד המרכזי אל הבקר
- התקן **קלט**: "קריأت" הקלט, "כתיבת" הקלט, "קריאת" פקודות הפעלה
- התקן **פלט**: "כתיבת" הפלט, "קריأت" סטטוס

# אמצעי קלט/פלט

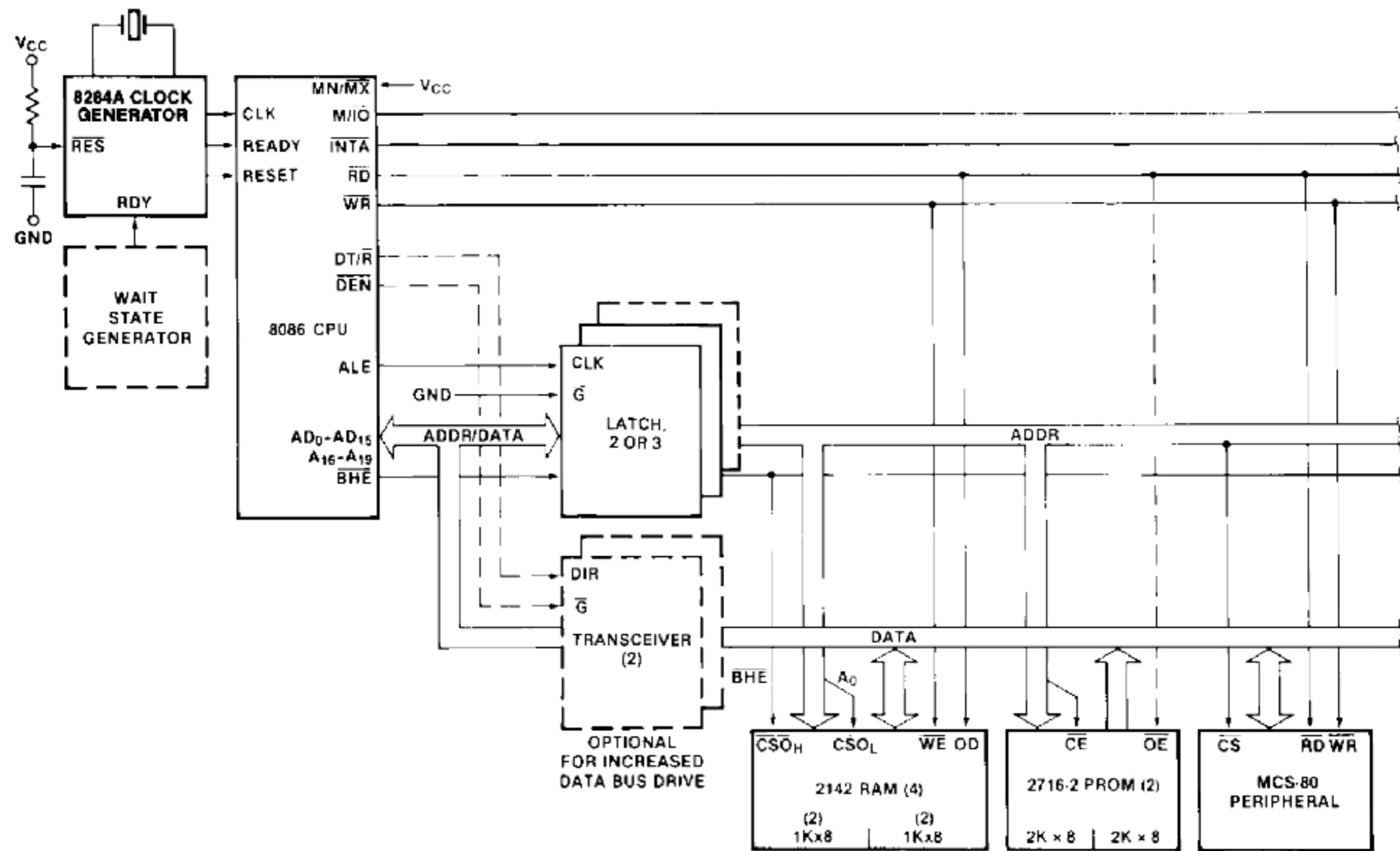
- שתי גישות מרכזיות לחבר התקן היקפי:
  - **מייפוי זיכרון**
  - **מייפוי קלט/פלט ישיר**

# מיפוי זיכרונו

- התקן היקפי המחבר במיפוי זיכרונו:
- "מחوط" אל אישור מסויים של זיכרונו (למשל במחשב ישן / סימולטור: אישור הזיכרונו 0000:0000:0000:0000 מופה אל כרטיס הוידאו)
- גישה לתקן מבוצעת ע"י גישה לאישור הזיכרונו - פקודות VOM וכל שאר הפקודות ממונעות הזיכרונו.  
(זהירות: הכתובות הנ"ל לאו דווקא מתנהגות כמו "זיכרונו")
- חסרו: "טופס" חלק מרחב הזיכרונו יתרונו: גישה נוחה ומודולרית

# מייפוי ישיר

- התקן היקפי המחבר במייפוי ישיר:
- "מחוט" אל port קלט/פלט ייעודי,  
(למשל במחשב ישן / סימולטור: פורט 60 משמש את המקלדת)
- גישה לתקן מבוצעת ע"י פקודת ייעודית -  
**IN** לקריאה מהפורט, **OUT** לכתיבה אל הפורט
- יתרון: מגדיל את מרחב הכתובות לשימוש בלי להפריע  
לזיכרנו.



231455-5

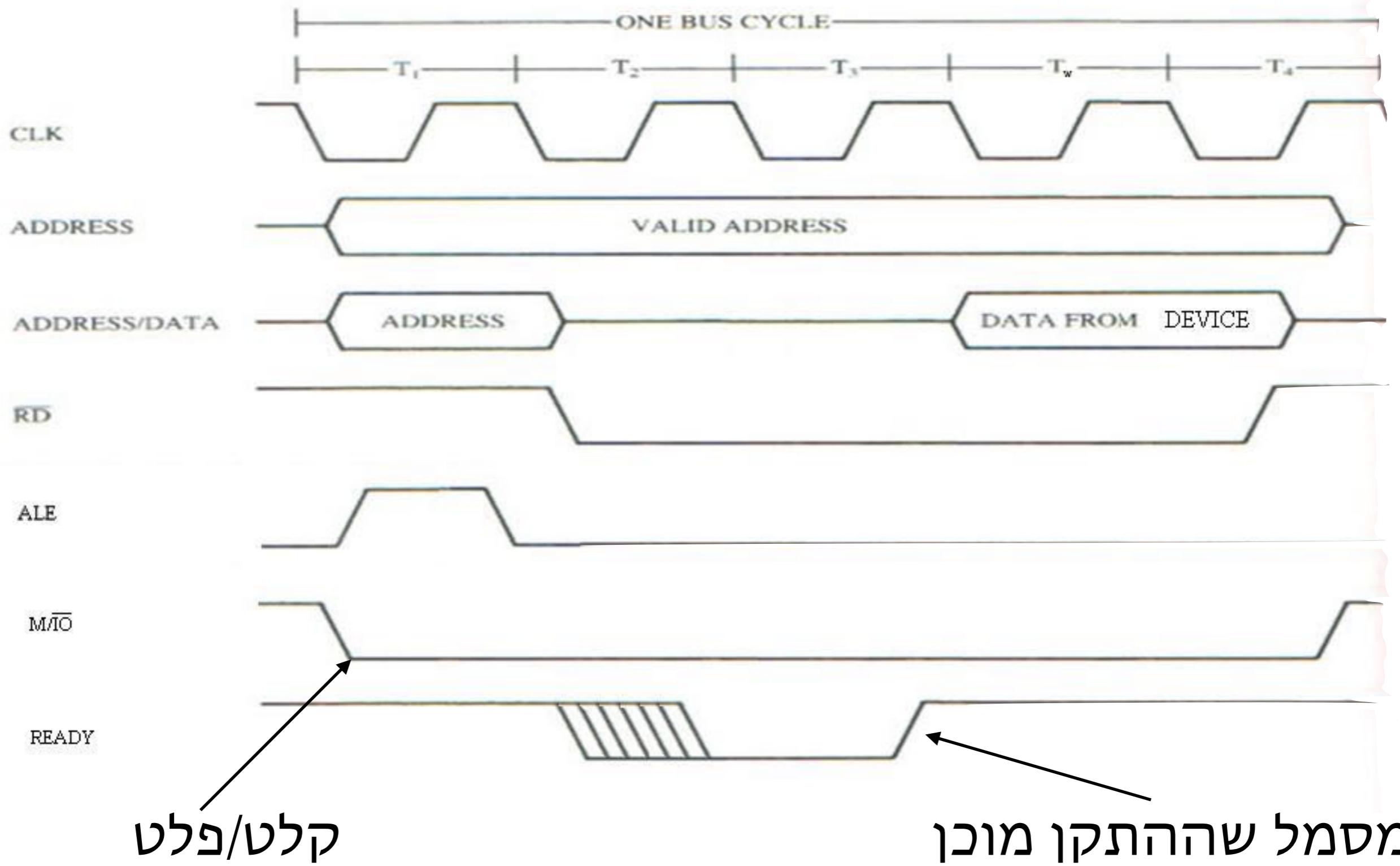
Figure 4a. Minimum Mode 8086 Typical Configuration

# 8086 Pin Layout

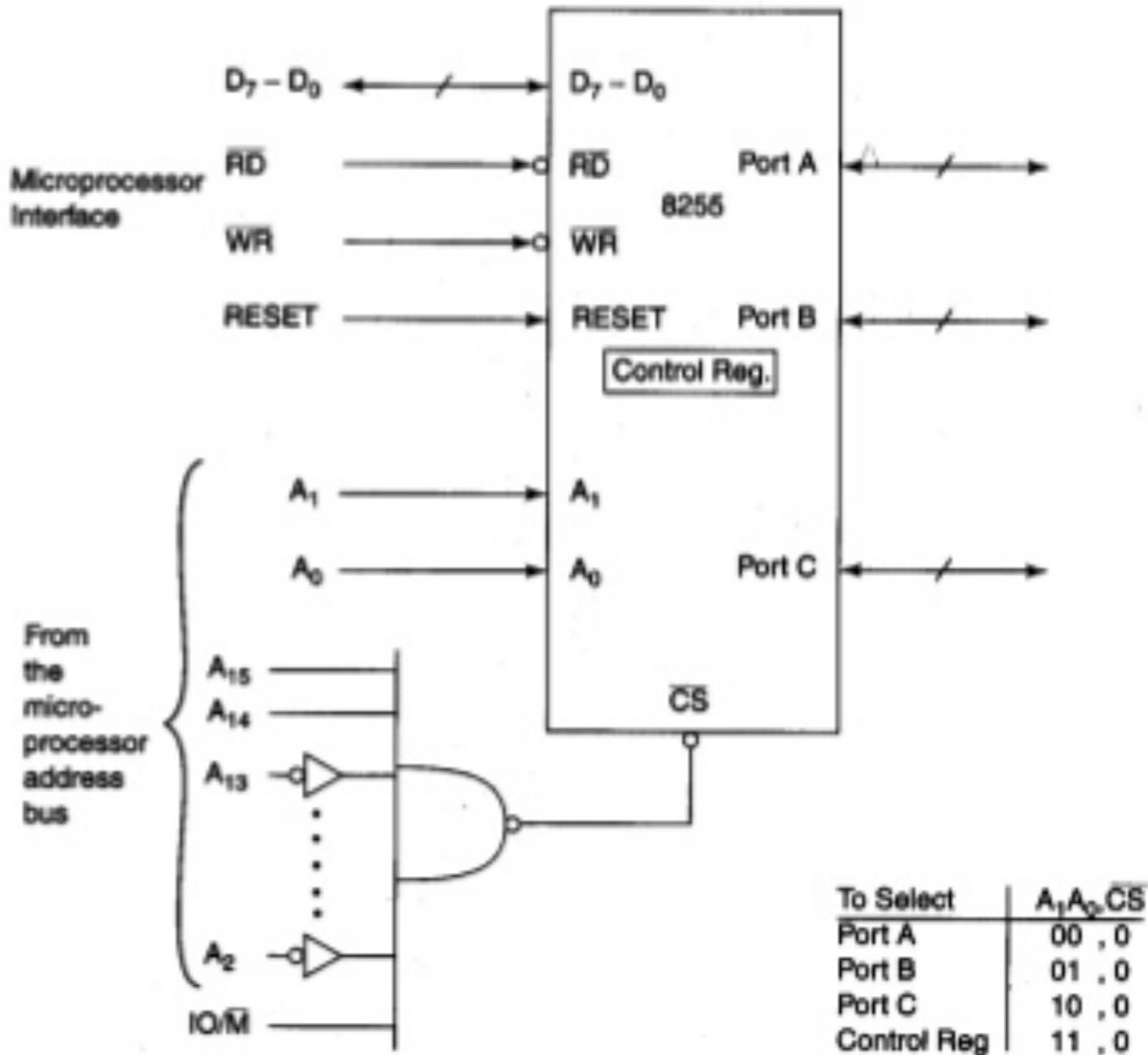
|      | MAX<br>MODE | { MIN<br>MODE } |
|------|-------------|-----------------|
| GND  | 1           | V <sub>CC</sub> |
| AD14 | 2           | AD15            |
| AD13 | 3           | A16/S3          |
| AD12 | 4           | A17/S4          |
| AD11 | 5           | A18/S5          |
| AD10 | 6           | A19/S6          |
| AD9  | 7           | BHE/S7          |
| AD8  | 8           | MN/MX           |
| AD7  | 9           | RD              |
| AD6  | 10          | CPU             |
| AD5  | 11          | RQ/GT0 (HOLD)   |
| AD4  | 12          | RQ/GT1 (HLDA)   |
| AD3  | 13          | LOCK (WR)       |
| AD2  | 14          | S2 (M/IO)       |
| AD1  | 15          | S1 (DI/R)       |
| AD0  | 16          | S0 (DEN)        |
| NMI  | 17          | QS0 (ALE)       |
| INTR | 18          | QS1 (INTA)      |
| CLK  | 19          | TEST            |
| GND  | 20          | READY           |
|      |             | RESET           |

- רجل  $M/\overline{IO}$  מסמלת האם:  
• ה" כתובת" היא במרחב הזיכרון  
(פקודת MOV ..)  
או במרחב הקלט/פלט  
(פקודת IN/OUT)

# מחזור קריאה כתיבה



# דוגמא: ממשך 8255



# קלט/פלט ב6808

- מרחב כתובות קלט פלט של 16bits = 64 פורטים
- אוגר מקור יעד: **תמיד AX** (או AL ל-8bits)
- מיעון מיידי ל-255 פורטים ראשוניים, מיעון עקיף (DX) לשאר הפורטים:
  - IN AX/AL, <port 0-255>
  - IN AX/AL, DX
  - OUT <port 0-255>, AX/AL
  - OUT DX, AX/AL
- קריית נתון
- כתיבת נתון

# קלט/פלט ב8088

8/16bit

- מבנה הפקודה

**IN** = Input from:

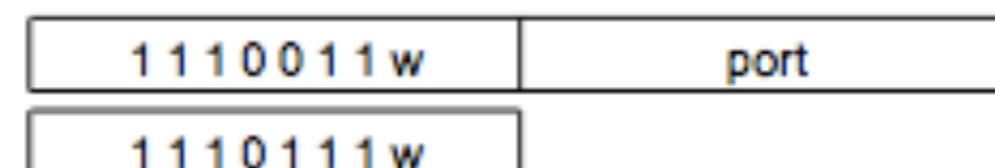
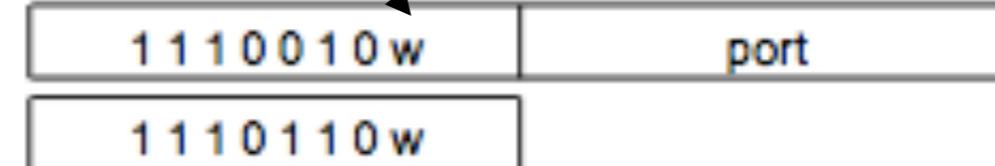
Fixed Port

Variable Port

**OUT** = Output to:

Fixed Port

Variable Port



- אין צורך לציין אוגרים : תמיד AX/DX/AX/...

# דוגמא: גישה למקלדת

- המקלדת (במחשב מבוסס 8086/אמולטור DOSBOX) מוחברת דרך בקר-משנה בפורט **60h** ופורט **64h**
- קריית 64h: **סטטוס מקלדת** ביט LSB מצב האם נלחץ כפתור
- קריית 60h: **קוד המקש שנלחץ**

הרחבה ב

<http://www.plantation-productions.com/Webster/www.artofasm.com/DOS/ch20/CH20-2.html#HEADING2-1>

# דוגמא: גישה למקלדת

- דוגמא: בדיקת סטוס המקלדת והבאת המKeySpec הנלחץ

```
chk_kbd: IN AL, 64h
 TEST AL, 01h
 JZ chk_kbd
```

```
IN AL, 60h
<use key>
```

# שיטות גישה להתקון

- **שיטה משאל** (Polling)  
המעבד ניגש שוב ושוב אל הפורט, ובודק האם קיים מידע חדש
- **שיטה פסיקה** (Interrupt)  
המעבד מבצע פעילות אחרת. ברגע שהייה מידע חדש, עבדתו תיפסק והוא יטפל במידע החדש
- הودעה כנ"ל למעבד נקראת "פסיקה"  
שכו, היא מפסיקת את עבדתו השותפת

# שיטות גישה להתקן

- האם הקוד שראינו מבצע **משאל** או **פסקה**?

```
chk_kbd: IN AL, 64h
 TEST AL, 01h
 JZ chk_kbd
```

```
IN AL, 60h
<use key>
```

# שיטות גישה להתקון

- **שיטה משאל** (Polling)
- חסרונות: המעביר "עסוק" בביצוע המשאל, ואיןו יכול לבצע פעולות אחרות
- **שיטה פסיקה** (Interrupt)
- יתרונות: המעבד פניו לביצוע משימות אחרות
- חסרונות: נדרשת תמיכה חומרתית של המעבד; פולה מרכיבת יותר; מה קורה כישר הרבה פסיקות בבית אחת

# סיכום: קלט-פלט

- **שימוש בתקנים חיצוניים**
- **מייפוי זיכרון** - התקן מתנהג כמו אзор בזיכרון  
(הברת מידע ע"י פקודת MOV)
- **מייפוי ישיר** - מרחב כתובות נפרד של 64k פורטים  
(הברת מידע ע"י TZ/OUT/IN)
- **ניתור התcano:**
  - משאל** - המעבד עסוק בניטור.
  - פסקה** - התקן מודיעע למעבד متى משתנה הסטוס

# לפני שמעמיקים בפסיקות...

- שאלה פילוסופית:
- איך המחשב יכול לבצע "מספר תוכניות בבית אחת"?
- האם תמיינה בריבוי משימות יכול להתבצע בשיטת  
משאל? בשיטת פסיקה?

# **פְּסִיקוֹת:**

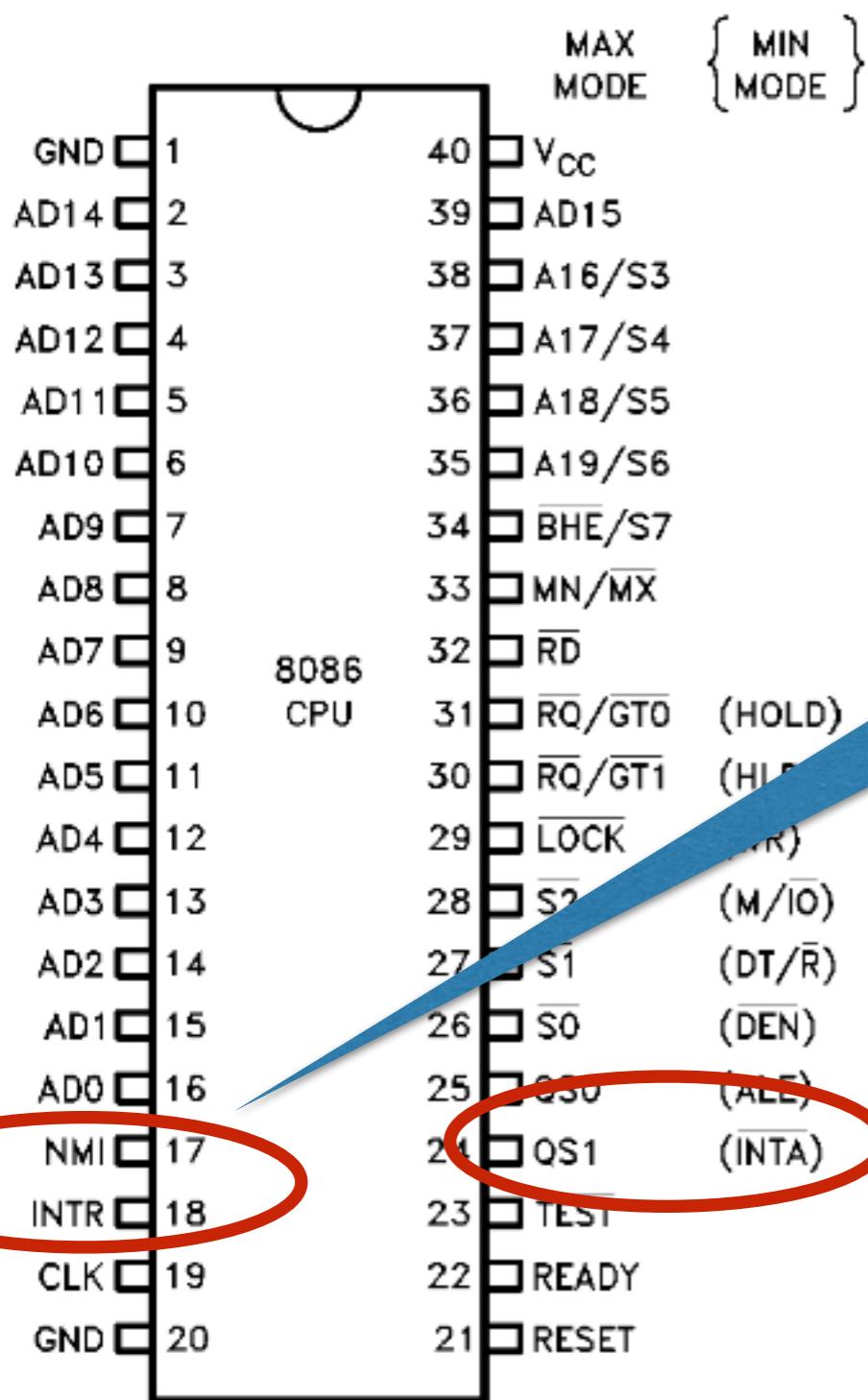
**פְּסִיקוֹת תּוֹכָנָה, חֲרִיגּוֹת, וֶפְּסִיקוֹת חֻמְרָה**

**חלק א: פְּסִיקוֹת חֻמְרָה, וֶפְּסִיקוֹת לְא-מְמוֹסָכוֹת**

# פזיקת חומרה

- אוט חשמלי המתקבל במעבד מהתקו חיצוני, ומודיע על מאורע כלשהו שקרה במערכת
- לכל פזיקה יש "**מספר**" מזהה. המעבד מטפל בפזיקה ע"י הרצת קטע קוד המשוויך **לאותו המספר**.
- פזיקות (חומרה) נפוצות לדוגמא:
- שעון (ב608 - פזיקה כל 55 מילישניות)
- פזיקת מקלדת, עבר (PS/2), ...
- יציאה טורית (COM1/COM2) ומקבילית (LPT1)

# 8086 Pin Layout



עלית האות החשמלי ברגל NMI או INT<sub>I</sub> תגרום למעבד להכנס ל" מצב פסיקה " ותחילה הטיפול בפסקה

בעזרת רgel INT<sub>A</sub> המעבד מודיע שהוא קיבל את הפסקה, ומקבל נתונים (דרך פס המידע) על הפסקה, סוג (מספר) הפסקה למשל, סוג (מספר) הפסקה

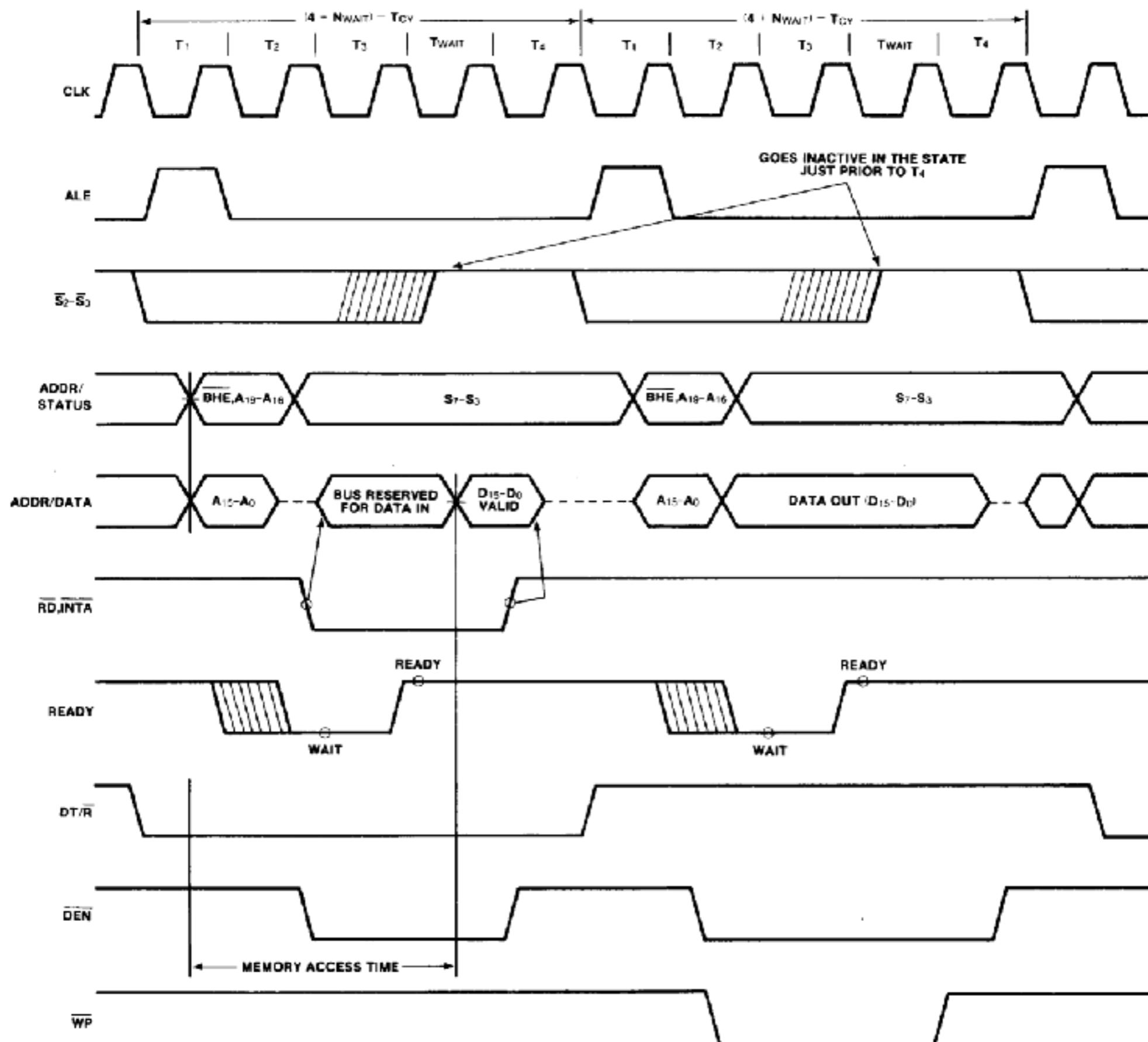


Figure 5: Basic System Timing

# טבלת פסיקות

- לכל **מספר פסיקה** שונה נרצה להריץ **קוד** שונה.  
איך מתבצע התרגום?
- **פתרון א:** נשמר את **הקוד** במיקום קבוע בזכרון.
- למשל, קוד השגירה של פסיקה 0, בכתבובת 00000.  
הקוד של פסיקה 1 בכתבובת 00010 וכך הלאה.
- חסרונו: מה אם הקוד "גדול" ולא נכנס?
- **פתרון ב:** טבלת "תרגום" המכילה **מצבייעים** לשגרות הפסיקה
  - בכתבובת 00000 נשמר מצביע לכתבובת של הקוד של פסיקה 0  
בכתבובת 00004 נשמר מצביע לכתבובת של הקוד של פסיקה 1

# פსיקת חומרה ב-8086

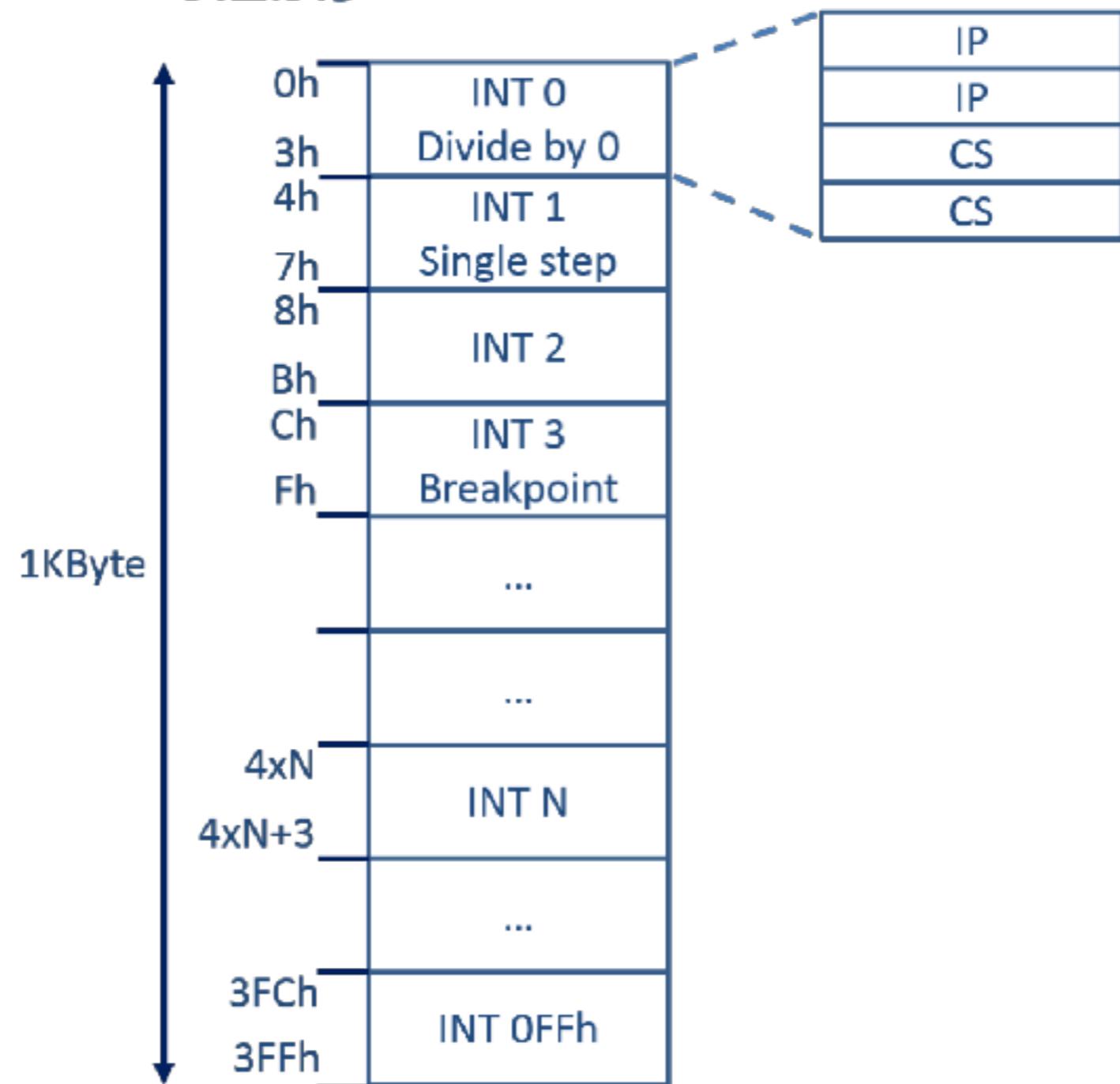
- **טיפול בפסקית חומרה:**
- חומרת המעבד מSIGA את מספר הפסקה XX
- מצב אוגר הדגלים נשמר במחסנית (STACK ← FLAGS)
- איפוס דגל הפסקה (F) ודגל trap (TF)
- מיקום התוכנית הנוכחי נשמר (IP ← CS:IP)
- חישוב IP:CS של הקוד המתאים לפסקה מס' XX לפי **טבלת תרגום**
- **ביצוע קוד הפסקה המתאים למספר XX**
- **בסיום הפסקה**, שחזור מצב הדגלים וחזרה ל-IP:CS שנשמר במחסנית

# פזיקת חומרה

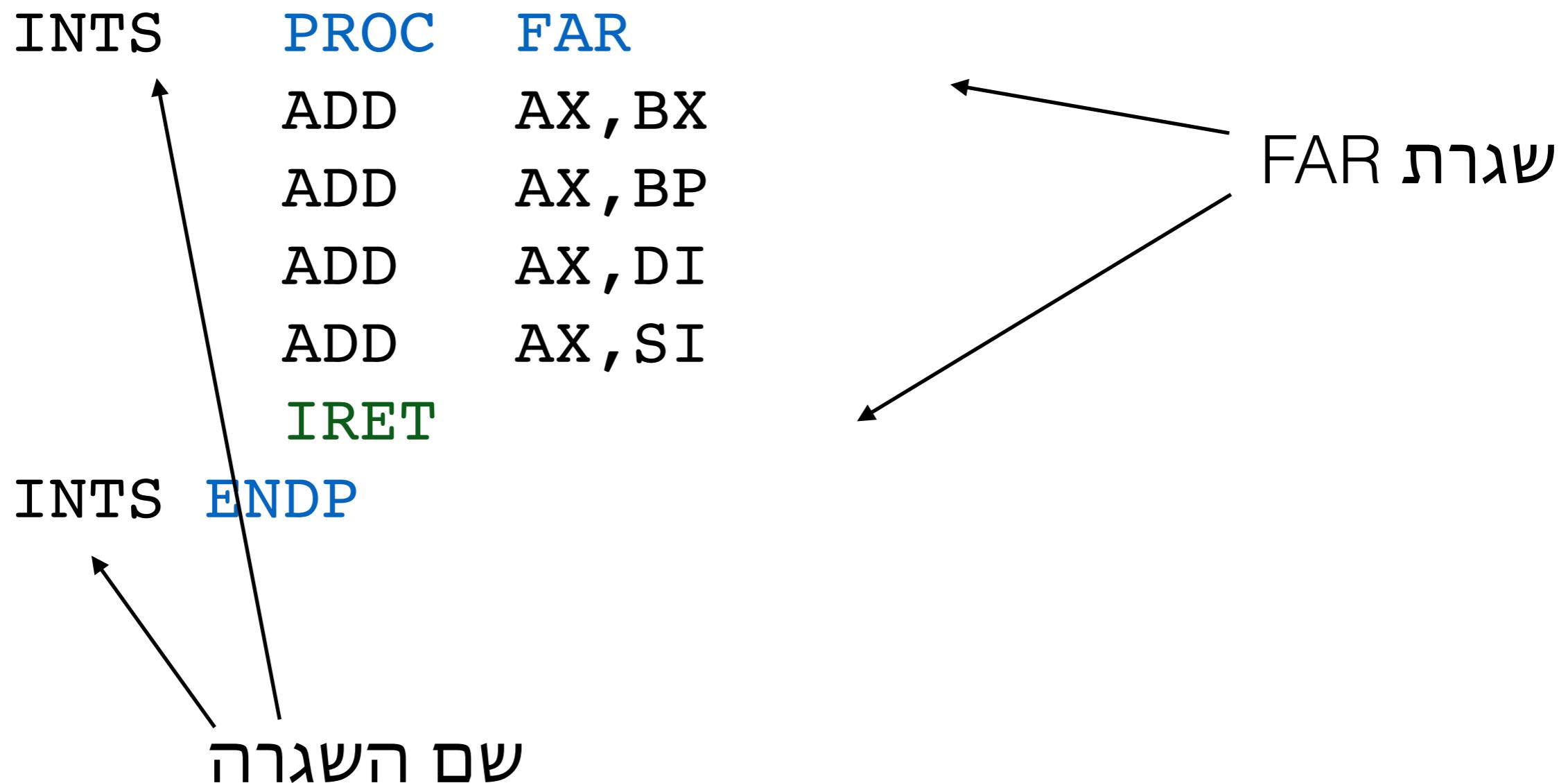
- חישוב CS:IP של שגרת הפסיקה:
- המעבד תומך בעד 256 פזיקות שונות
- כתובת h0000:0000 עד 0000:03FFh בזיכרון הראשי מכילות טבלה של 256 רשומות (כל רשומה בגודל 4 בית)
- רשומה מספר XX מכילה את CS:IP של שגרת הפסיקה מספר XX
- טבלה זו נקראת IVT - Interrupt Vector Table
- ISR - Interrupt Service Routine נקראת שגרת פזיקה

# מבנה ה-IVT

## כתובת



# מבנה שגרת פסיקה ISR



# מבנה שגרת פסיקה ISR

| INTS | PROC | FAR     |
|------|------|---------|
|      | ADD  | AX , BX |
|      | ADD  | AX , BP |
|      | ADD  | AX , DI |
|      | ADD  | AX , SI |
|      | IRET |         |
| INTS | ENDP |         |



קוד השgraה עצמה

# מבנה שגרת פסיקה ISR

| INTS | PROC | FAR     |
|------|------|---------|
|      | ADD  | AX , BX |
|      | ADD  | AX , BP |
|      | ADD  | AX , DI |
|      | ADD  | AX , SI |
|      | IRET |         |
| INTS | ENDP |         |

חזרה משגרת פסיקה

IRET מבצעת:

pop IP-CS; מהמחסנית;

(2) שחזור דגלים IP-CS; POPF ; (3) קפיצה ל-

# דגל הפסיקה (IF)

- במהלך הטיפול בפסקה, המעבד מכבה את דגל הפסיקות (IF).
- כאשר דגל זה כבוי, המעבד "מتعلם" פסיקות חדשות:  
עליה חשמלית של רgel INTR לא מבצעת דבר
- ניתן לשלוט על הדגל בעזרת הפקודות CLI / STI
- איפוס הדגל נקרא **מייסור פסיקות** — בעת שהדגל כבוי פסיקות חדשות ממוסכבות ולא יטופלו
- לרוב, קיים "בקר פסיקות" חיצוני ששומר את הפסיקות עד שדגל הפסיקות מחודש; אחראי גם לשולח פסיקות למאבד לפי עדיפות, אם כמה פסיקות מתרכשות בו בזמןית.

# דגל ההחלטה (IF)

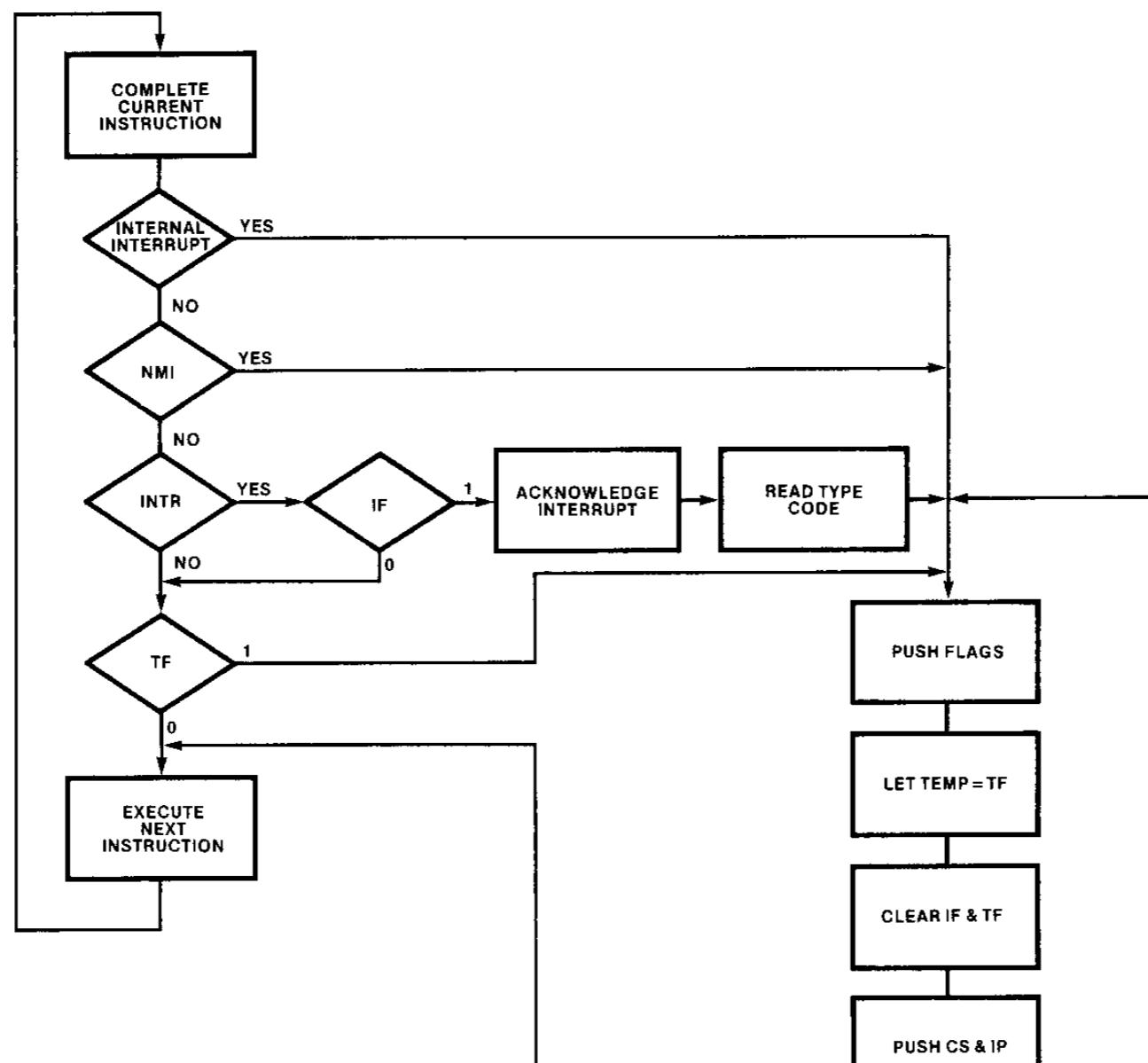
- בעת סיום ריצת השגרה משוחזר דגל ההחלטה
- (למה ואיך?)

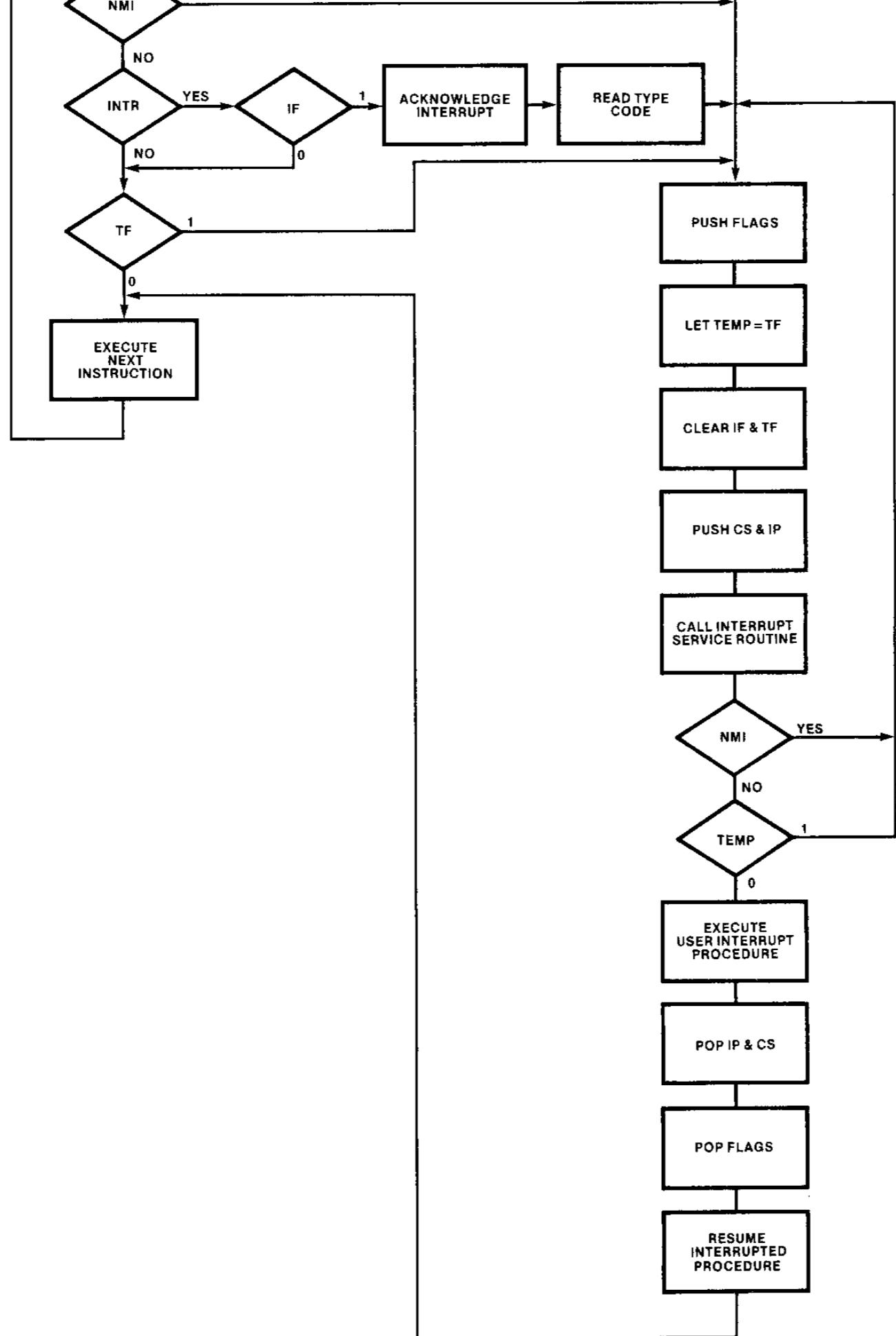
# פְּסִיקַת NMI

- מעבד קיימת רgel נוספת היוצרת פסיקה NMI
  - $NMI = \text{Non-Maskable Interrupt}$   
**פסיקה *שaina* ניתנת למיסוך**
- פסיקה זו תתקבל גם אם דגל FI כבוי
- מיועד לקרים קיצוניים: למשל, ירידה במתה החשמלי, או שגיאה קריטית שחיבבים לטפל בה.
- פסיקת NMI מופגה תמיד לרשותה מספר 2 בוקטור הפסיקות

# תהליך פסיקה: סיכום

## 8086 AND 8088 CENTRAL PROCESSING UNITS





# **פְּסִיקוֹת:**

**פְּסִיקוֹת תּוֹכָנָה, חַרְיגּוֹת, וֶפְּסִיקוֹת חֻמְרָה**

**חָלֵק ב: פְּסִיקוֹת תּוֹכָנָה, וֶחַרְיגּוֹת**

# פסקיות תוכנה וחריגות

- קריאה לפסקה יכולה לגרום ע"י מאורע במעבד ולאו דווקא מאורע חיצוני
- קריאה מפורשת של התוכנה - **פסקית תוכנה**
- קריאה לאור תקלה במעבד - **חריגה** (exception) -

# פסיכות תוכנה וחריגות

- דוגמאות לחריגות:
- **חילוק באפס.** יגרור את הפעלת שגרת פסיקה מס' 0
- **чисוב עם overflow:** הפקודה INTO ת לבדוק אם היה overflow ותפעיל את פסיקה מס' 4, בהתאם
- **Stack Exception** - פוליה על המחסנית גרמה ל-SP לעבור את כתובת FFFF - תגרור פסיקה מס' 12 (ב-286 ומעלה..)

# פსיקות תוכנה וחריגות

- בנוסר, המשתמש יכול להפעיל פסיקות כרצונו.  
**הפעלה זו נקראת פסיקת תוכנה**
- הפעלת פסיקת תוכנה ע"י הפקודה

INT #num <#num = 0..255>

- מתנהלת בדיק כמו פסיקת חומרה עם מס' XX:  
שמירת דגלים, שמירת IP:CS, איפוס IF/TF וקפייה  
לכתובת רשומה בוקטור הפסיקות מס' XX.

# פְּסִיקוֹת תּוֹכָנָה וּחֶרְיגָׁות

- פְּסִיקוֹת תּוֹכָנָה מִשְׁמֻשָׂת לְעִיתִים עַבְור הַפָּעֵלָת  
פּוֹנְקָצִיות מִעֲרָכָת שׁוֹנוֹת
- שָׁגָרוֹת BIOS או שָׁגָרוֹת מִעֲרָכָת הַפָּעֵלָה DOS
- הַמְשַׁתְמֵשׁ יִכְלֶن לְכַתְּבּ ISR וְלְהַשְׁתָּמֵשׁ בָּה בַּמְקוּם  
קְרִיאָה לְשָׁגָרָה

# פסיקות תוכנה וחריגות

- **פסיקות תוכנה VS שגרות**
- לפסיקת תוכנה "מקום" קבוע, ושיטת הפעלה קבועה (למשל, המשתמש לא צריך לדעת את שם השגרה; נוח כאשר הפסיקה נכתבת ע"י מישחו אחר, למשל microsoft)
- תוכן שגרת הפסיקה יכול להתעדכו לפי תצורת המערכת **בזמן הריצה**
- למשל: שגרת טיפול שונה למקלדת 101/102 מקלים
- איך היינו מבצעים אם הייתה כתובה **שגרה** ולא **פסיקה**?

# פ Sikoth DOS ו-BIOS

- DOS ו-BIOS מספקות פ סיקות תוכנה להפעלת חומרת ה"מחשב". מספר דוגמאות שימושיות:
- BIOS: (פ סיקות F1 - 00, נכתבו ע"י יצרן החומרה)
- 10h - פ סיקת וידאו (לדוגמה:)
- שירות 00=AH אוגר AL קובע את מצב הווידאו
- שירות 02=AH קובע את מיקום הסמן (cursor)
- שירות AH=0Ch כתיבת פיקסל במיקום CX,DX

# פسيקות DOS ו-BIOS

- BIOS: (פסיקות F - 00 - 09)
- פסיקת מקלדת (חומרה) -  
תרחשת בכל לחיצה על המקלדת
- 16h - פסיקת תוכנה לשירות מקלדת (לדוגמה):
- שירות 00=AH מחזירה מקש נקלט
- שירות 05=AH מסמלצת לחיצת מקש

# פسيקות DOS ו-BIOS

- **DOS:** (שגרות שנכתבו ע"י מ"ה, פסיקה h(21h)
- **שרות AH=4Ch סיום תוכנית (וחזרה למ"ה)**
- **שרות AH=02h הצגת תו על המסך**
- **שרות 01=AH קליטת תו מהמקלדת+הציגתו**
- **שרות AH=25h/35h קריית/שינוי רשומה ב-IVT**

<http://80864beginner.com/8086-interrupt-list/>

<http://spike.scu.edu.au/~barry/interrupts.html>

הרחבה:

הרחבה:

# IVT במחשב PC יישו



| Type  | Function                    | Comment                          |
|-------|-----------------------------|----------------------------------|
| 0     | Divide Error                | Processor - zero or overflow     |
| 1     | Single Step (DEBUG)         | Processor - TF=1                 |
| 2     | Nonmaskable Interrupt Pin   | Processor - NMI Signal           |
| 3     | Breakpoint                  | Processor - Similar to Sing Step |
| 4     | Arithmetic Overflow         | Processor - into                 |
| 5     | Print Screen Key            | BIOS - Key Depressed             |
| 6     | Invalid Opcode              | Processor - Invalid Opcode       |
| 7     | Coprocessor Not Present     | Processor - no FPU               |
| 8     | Time Signal                 | BIOS - From RT Chip (AT - IRQ0)  |
| 9     | Keyboard Service            | BIOS - Gen Service (AT - IRQ1)   |
| A - F | Originally Bus Ops (IBM PC) | BIOS - (AT - IRQ2-7)             |
| 10    | Video Service Request       | BIOS - Accesses Video Driver     |
| 11    | Equipment Check             | BIOS - Diagnostic                |
| 12    | Memory Size                 | BIOS - DOS Memory                |
| 13    | Disk Service Request        | BIOS - Accesses Disk Driver      |
| 14    | Serial Port Service Request | BIOS - Accesses Serial Port Drvr |
| 15    | Miscellaneous               | BIOS - Cassette, etc.            |
| 16    | Keyboard Service Request    | BIOS - Accesses KB Driver        |

# IVT במחשב PC יישו



| Type  | Function                       | Comment                         |
|-------|--------------------------------|---------------------------------|
| 17    | Parallel Port LPT Service      | BIOS - Printer Driver           |
| 18    | ROM BASIC                      | BIOS - BASIC Interpreter in ROM |
| 19    | Reboot                         | BIOS - Bootstrap                |
| 1A    | Clock Service                  | BIOS - Time of Day from BIOS    |
| 1B    | Control-Break Handler          | BIOS - Keyboard Break           |
| 1C    | User Timer Service             | BIOS - Timer Tick               |
| 1D    | Pointer to Video Parm Table    | BIOS - Video Initialization     |
| 1E    | Pointer to Disk Parm Table     | BIOS - Disk Subsystem Init.     |
| 1F    | Pointer to Graphics Fonts      | BIOS - CGA Graphics Fonts       |
| 20    | Program Terminate              | DOS - Clear Memory, etc.        |
| 21    | Function Call                  | DOS - Transfer Control          |
| 22    | Terminate Address              | DOS - program Terminate handler |
| 23    | Control-C Handler              | DOS - For OS Use                |
| 24    | Fatal Error Handler            | DOS - Critical Error            |
| 25    | Absolute Disk Read             | DOS - Disk Read                 |
| 26    | Absolute Disk Write            | DOS - Disk Write                |
| 27    | Terminate                      | DOS - TSR Usage                 |
| 28    | Idle Signal                    | DOS - Idle                      |
| 2F    | Print Spool                    | DOS - Cassette, etc.            |
| 70-77 | Hardware Interrupts in AT Bios | DOS - (AT - IRQs 8-15)          |

# INT 1Ah Timer Interrupt

Posted By : Murugan Andezuthu Dharmaratnam    Posted On :    Keywords :



אוניברסיטת בר-אילן  
Bar-Ilan University

## INT 1Ah Timer Interrupt

The INT 1Ah software interrupt handles the time of day I/O services. A Carry flag set on exit may indicate the clock is not operating.

### Function 00h Read current time

Input : none

Output:

CX High word of tick count  
DX Low word of tick count  
AL 00h = Day rollover has not occurred  
(Timer count is less than 24 hours since last power on or reset)

### Function 01h Set current time

Input:

CX High word of tick count  
DX Low word of tick count

Output: none

### Function 02h Read real time clock

Input: none

Output:

CH BCD hours  
CL BCD minutes  
DH BCD seconds  
DL 00 = Standard Time  
01h = Daylight Savings

קוד דוגמא:

```
mov ah, 02
int 1Ah
;;cx/dx now
;; hold the time
```

# פסיקות: מבנה פקודת

**INT** = Interrupt

Type Specified

Type 3 (breakpoint)

**INTO** = Interrupt on Overflow

**IRET** = Interrupt Return

|                 |      |
|-----------------|------|
| 1 1 0 0 1 1 0 1 | type |
| 1 1 0 0 1 1 0 0 |      |
| 1 1 0 0 1 1 1 0 |      |
| 1 1 0 0 1 1 1 1 |      |

# כתיבת ISR ו שינוי T7

- ביכולתנו לשנות את הקוד המופעל בעת פסיקה כלשהי
- למשל לשנות את הטיפול במקלדת - פס'09
- למשל לכתב פסיקה חדשה למאורע חדש במערכת הנתונה שלנו (מנוע/חישון/...)
- על מנת לבצע קוד משלנו בעת פסיקה צריך:
  1. לכתב קוד שגרת פסיקה (ISR)  
**בזמן הריצה!**
  2. לעדכן את טבלת ה-T7ו שתציביע אל הקוד החדש

# כתיבת ISR ושינוי IVT

מ"ה MS-DOS תעדכן את  
הIVT עבורנו, ע"י הפסיקה  
AH=25h  
INT 21h

שמירת מקום שגרת הפסיקה הקודמת ?

- בעת שינוי IVT :
- חובה למסך פסיקות ע"י CLI  
(אחרת?)
- שמירת מקום שגרת הפסיקה הקודמת ?
- נדרש שקד השגרה לא יימחק מהזיכרון - גם לאחר סיום התוכנית  
(TSR)
- שיקולים בכתיבת ISR:
  - שמירת מצב המערכת - קיימת תוכנית ב"רקע" ...
  - קצהה ויעילה - במיוחד אם נקרהת הרבה פעמים (למשל פסיקת שעון!)
  - האם להדליק IF ? - Reentrancy

# דוגמא: החלפת פסיקה 0

- פסיקה 0 פועלת כאשר המעבד מבצע חילוק ב-0  
(divide by zero exception)
- בד"כ גורמת ליצירת התוכנית
- נרצה לכתוב שגרת פסיקה שקדום מדפיסה למסך הודעת שגיאה ואח"כ קוראת לשגרת פסיקה המקורית להמשך הטיפול בחיריגה

# דוגמא: החלפת פסיקה 0

```
.code
 oldOff dw 0 ; place to hold IP of old int
 oldSeg dw 0 ; place to hold CS of old int
 msg db 'OH NO! Divide by 0$' ; msg to print

.startup ; start running here
push CS
pop DS ; NOTE: assuming throughout that DS=CS

; Part I - save the CS:IP of the old interrupt
mov bx,0
mov es,bx ; now es:bx = 0000:0000 = place of int 0

mov bx, es:[0000] ; save the old-ISR IP
mov OldOff, bx
mov bx, es:[0002] ; save the old-ISR CS
mov OldSeg, bx
```

# דוגמא: החלפת פסיקה 0

```
; Part II - set the new ISR in IVT

cli ; can't allow interrupts during the change

mov es:[0000], offset NewISR0 ; the new proc
mov es:[0002], seg NewISR0

sti ; IVT is up-to-date, restore ints

<rest of the program to run... >
```

# דוגמא: החלפת פסיקה 0

; Part III - the new ISR code

NewISR0:

```
 mov dx, offset msg ; load msg address
 mov ah, 09h
 int 21h ; print string from DS:[DX] ...
```

JMP DWORD PTR [oldOff]; JMP to the old int0

;;; NO IRET!!! why?

;;; (cf. CALL DWORD PTR [oldOff])

# פסיכות: סיכום

- פסיכות מאפשרות למדע "להפסיכ" את פועלתו, ולבור לטפל במאורע אחר שהתרחש במערכת
- שני סוגי פסיכות:
  - **פסיכת תוכנה** - נראית במפורש ע"י המתכנתה (הרחבת של קריאה לשגרה)
  - **פסיכת חומרה/חריגה** - נראית ללא ידיעת התוכנה) עקב מאורע חיצוני או שגיאה

# פסיכות: סיכום

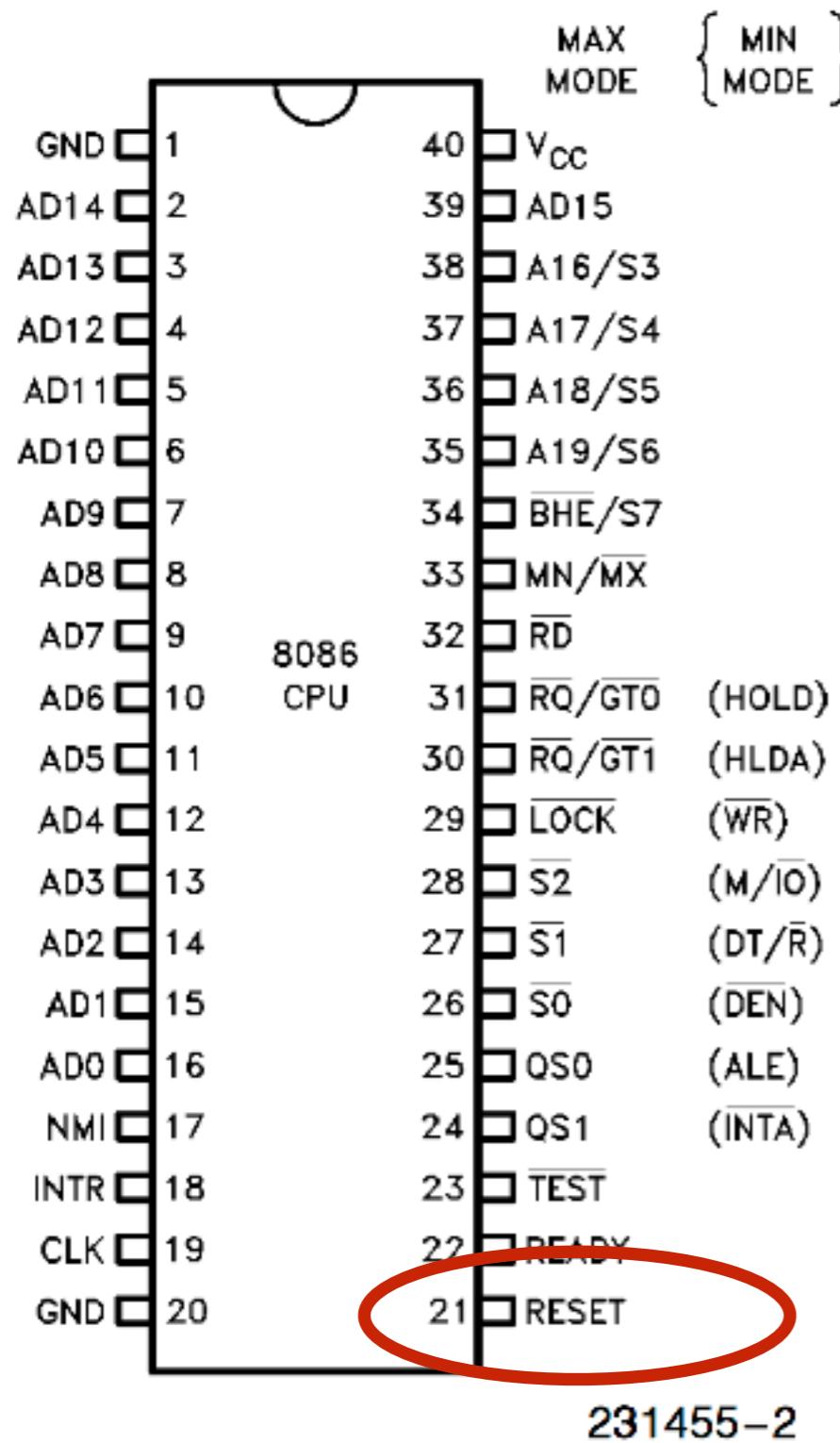
- שני סוגי פסיכות חומרה:
  - **ברת-מיסוך** (ע"י דגל F)
  - **אינה ברת-מיסוך**, וNM
- קריאה לשגרת פסיקה מתנהלת כמו קריאה לשgraה (רחוקה) עד כדי שמרית אוגר הדגלים ואיפוס IF/TF
- עד 256 פסיקות, טבלת פסיקות בזיכרון 00000-FF003

# תהליך הדלקה ואיתוח חול

# הבדיקה ואיתחול

- מהו תהליך ה"התעוררות" של המעבד?
- מה קורה בעת חיבור מתח למעבד, או בעת ביצוע איתחול (RESET)?

# הדלקה ואיתחול



- מתח גבוה ברגל reset תגרום למעבד לבצע את **תהליך ההתעוררות**:
- המעבד מפסיק את **הפעולה הנוכחית**.
- בעת ירידת המתח **יתחיל תהליך התעוררות**

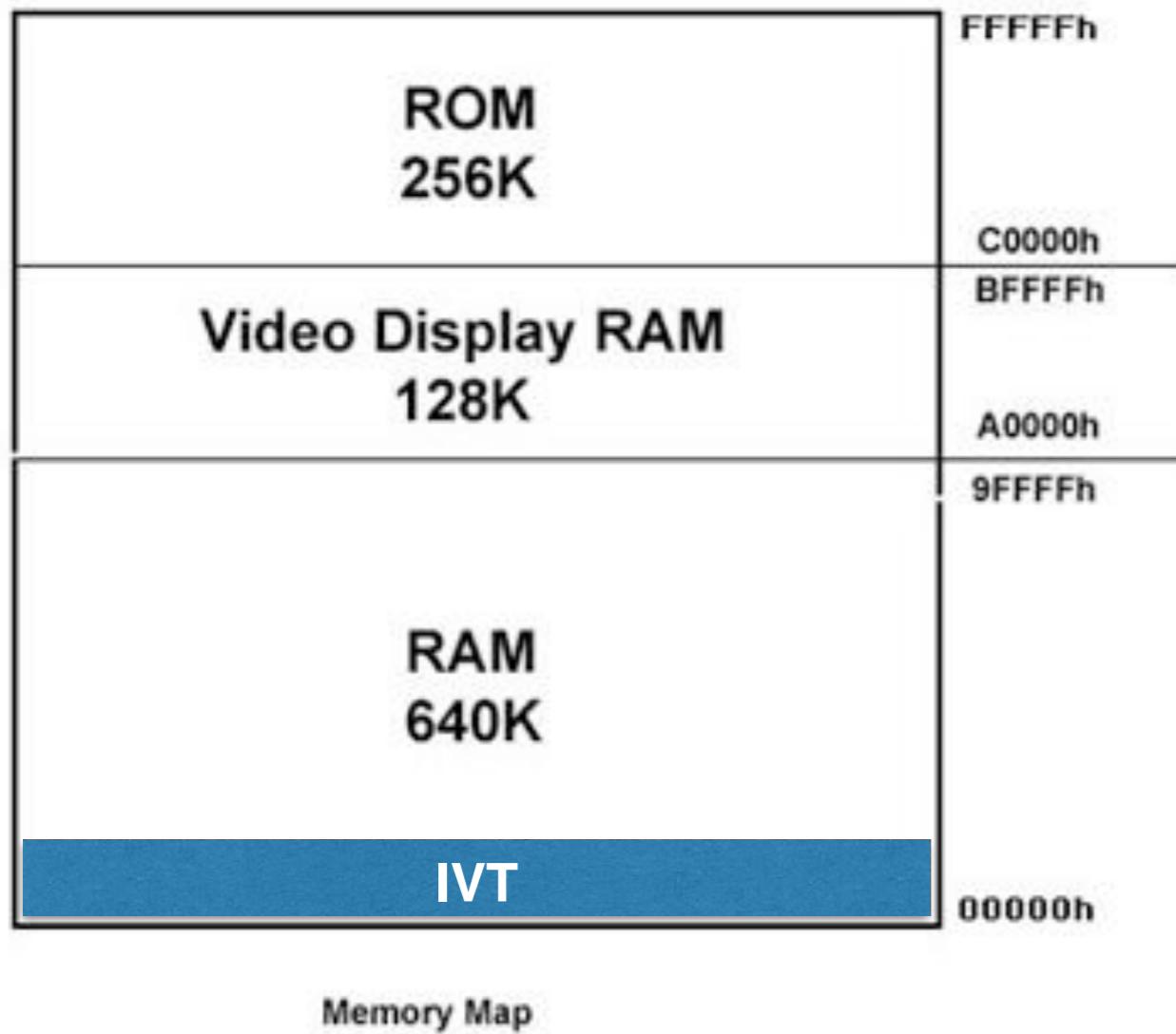
# הבדיקה ואיתחול

- תהליך התעוררות / איתחול:
- "איפוס" הדגמים
- איפוס אוגרי הסגמנט ES,DS,SS
- **CS:IP ← FFFF:0000**
- ממשיך ריצה רגילה

# ”ארגוו הזיכרון“ במחשב ביתי

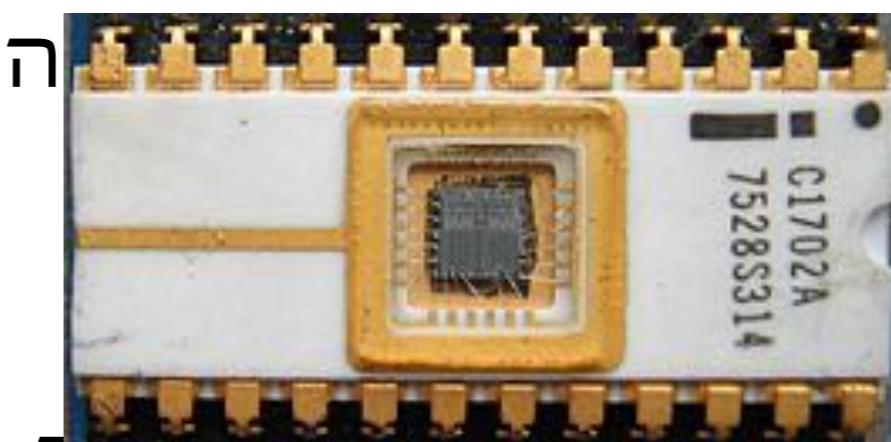
- כתובת **FFFF:0000** נמצאת 16 בתים לפני סוף מרחב הזיכרון
- לרוב נמצא שם פקודת קפיצה (JMP) לכתובות אחרת המכילה את תוכנה ה-BIOS של המחשב

# “ארגוו הזיכרון” במחשב ביתי



- רכיב EPROM - זיכרון לא מחיק מהוויט לאיזור כתובות גבוהות

- מכיל שגרות BIOS אשר נכתבו עבור לוח-האם הגרפי בו משתמשים



- אזור לכריום לאחסן נתונים נעלים (זיהום נתפס ע"י מערכת-הפעלה)



# קצת פרופורציות . . .

Pranay Pathole  
@PPathole

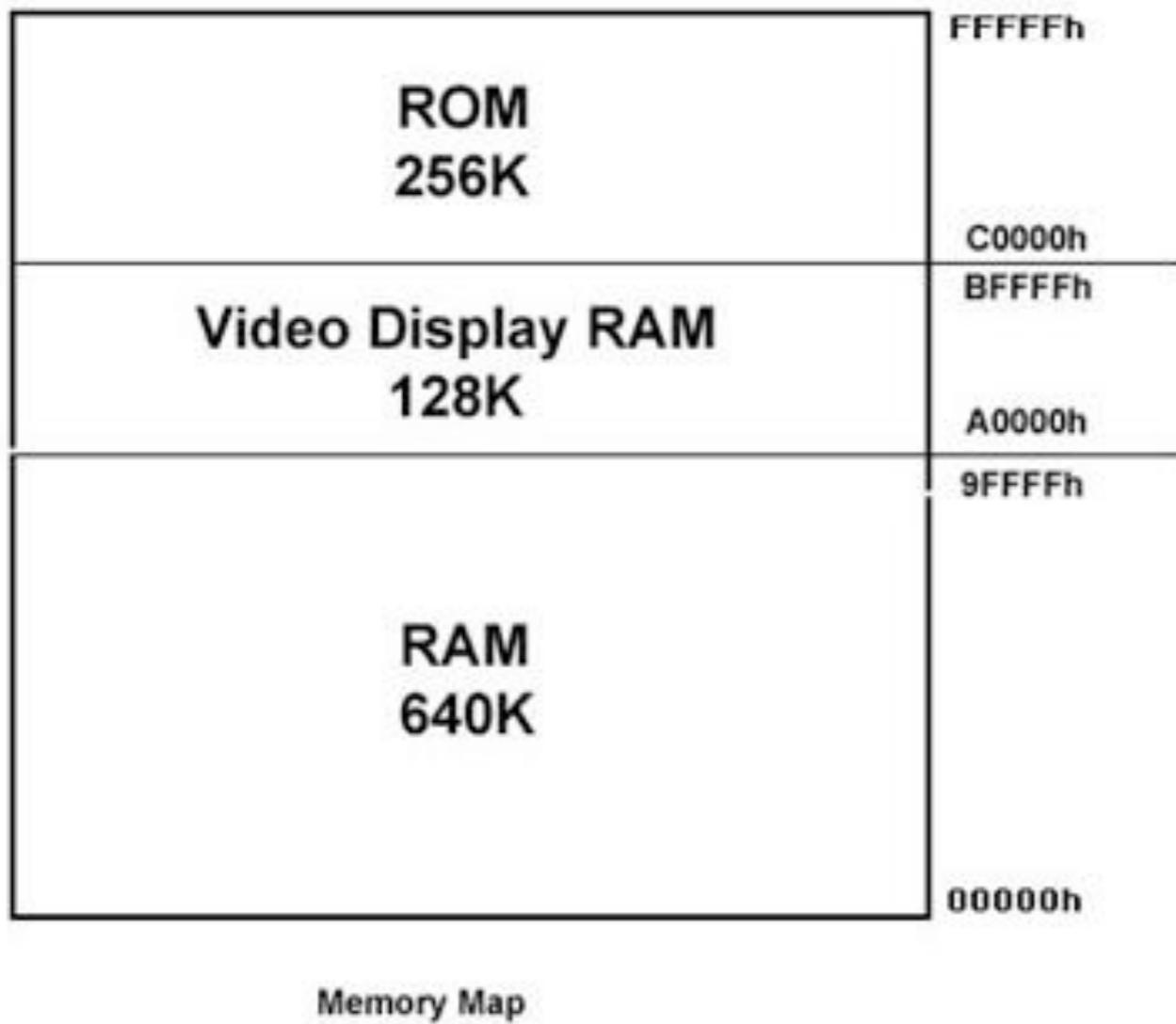
1973:

- What are you doing with that 4KB of RAM?
- Sending people to the moon.

2019:

- What are you doing with that 16GB of RAM and 102% CPU?
- Excel has a dialogue box open somewhere.

# שבוע הבא... . . .



- איפה מומוקמת התוכנית שאנו כותבים ומריצים?
- מיקום **לשחו** באזור ה RAM
- התוכנה שלנו חייבת להיות מותאמת להרצה בכל מקום אפשרי.
- האוגרים (למשל, DS) חייבים להתאים למקום הנכון!!
- **מי דואג לבך?**

# אורים: מקורות

- B. Brey, The Intel Microprocessor, 2009
- Intel 8086 spec, INTEL, 1990
- Randall Hyde, Art of Assembly, 1996
- I/O interfacing circuits –Hand shaking, serial and parallel  
<https://www.slideshare.net/jineshkj/8255-37761336>
- ארגון המחשב ושפת ספ, ברק גונן, גבהים, מרכז סייבר הצהלי 2015
- List of MS-DOS interrupts: <http://spike.scu.edu.au/~barry/interrupts.html>

# הידור, קישור והרצאה

מיקромעבדים ושפת אסמלר 83-255-83



# תכנות מודולרי

# עקרונות תכנות מודולרי

- תוכנה גדולה מורכבת מרכיבי תוכנה (module) קטנים
- רכיבים שונים נכתבים ע"י צוותים שונים
- כל רכיב הוא קוד "עצמאי"  
על-פי רב כל קובץ קוד יכול יכילה מודול יחיד (יכונה באותו השם)
- מודולים "סגורים" מופצים למשתמש כקובץ ספרייה  
(Library)

# עקרונות תכנות מודולרי

- תוכנה מודרנית (שנות 90+) נכתבת לרוב בשפה עילית (כגון C/C++ או JAVA)
- ניתן לשלב קטעי קודasm בתוך פונקציות שפה עילית למשל, ע"מ לבצע אופטימיזציה
- פעולות מסוימות עדין דורשות כתיבת באסמלר (למשל: גישה ישירה לחומרה ולבكري קלט/פלט)

# תכנות מודולרי: שיתוף תוויות ומשתנים

- נניח מודול א' (קובץ א') מכיל אזכור נתוניים משותף,  
ומודול ב' רוצה להשתמש בו

Module A:

GRADES DB 50 DUP(?)

...

Module B:

MOV AX, GRADES

- מה יקרה ?

# תכנות מודולרי: שיתוף תוויות ומשתנים

- בעט ביצוע קומפילציה, מודול ב', לא מכיר את התווית GRADES, ויעזר עמו שגיאה
- פתרון:
- מודול א', מגדר את grades בתור משתנה **פומבי** ע"י הפקודה PUBLIC
- מודול ב', מצהיר כי grades **מוגדר במודול אחר** ע"י הפקודה EXTERN
- התוכנה תשמור מקום בזיכרון לשנתה grades ותאפשר לכל המודולים השונים לגשת אל אוצר זיכרון זה

# תכנות מודולרי: שיתוף תוויות ומשתנים

Module A:

```
.data
PUBLIC GRADES DB 50 DUP(?)
```

```
.code
```

```
....
```

Module B:

```
.data
EXTERN GRADES:BYTE
```

```
.code
```

```
MOV AL, GRADES
```

# תכנות מודולרי: שיתוף תוויות ומשתנים

- באופנו דומה ניתן לשתף גם שגרות

Module A:

```
.code
PUBLIC upGRD
upGRD PROC FAR
.... ; update grades
RET
upGRD ENDP
```

Module B:

```
.code
EXTERN upGRD:FAR
...
CALL upGRD
```

# תכנות מודולרי: יתרונות וחסרונות

- **יתרונות:**
- כל רכיב הוא עצמאי,  **נכתב ונבדק** באופן בלתי תלוי
- רכיב לרוב מכיל פעילות ייחודית ומוגדרת היטב - קטן (שורות קוד) וקל יותר לכתיבת בדיקה
- שינויים משפיעים רק על הרכיב, ולא מצריכים שינויים בכלל התוכנה (כל עוד שומרים על משק)
- שימוש חוזר של קוד (re-use) - ספריות שגורות

# תכונות מודולרי: יתרונות וחסרונות

- **חסרונות:**
- איחוד רכיבים שונים הופך פעולה קשה - בדיקת כל מודול בנפרד אינה מספקת
- בדיקות אינטגרציה ארוכות וקשות (האשומות הדדיות - أيיזה מודול/צווות גרם לשגיאה)
- מצרייך תיעוד מפורט

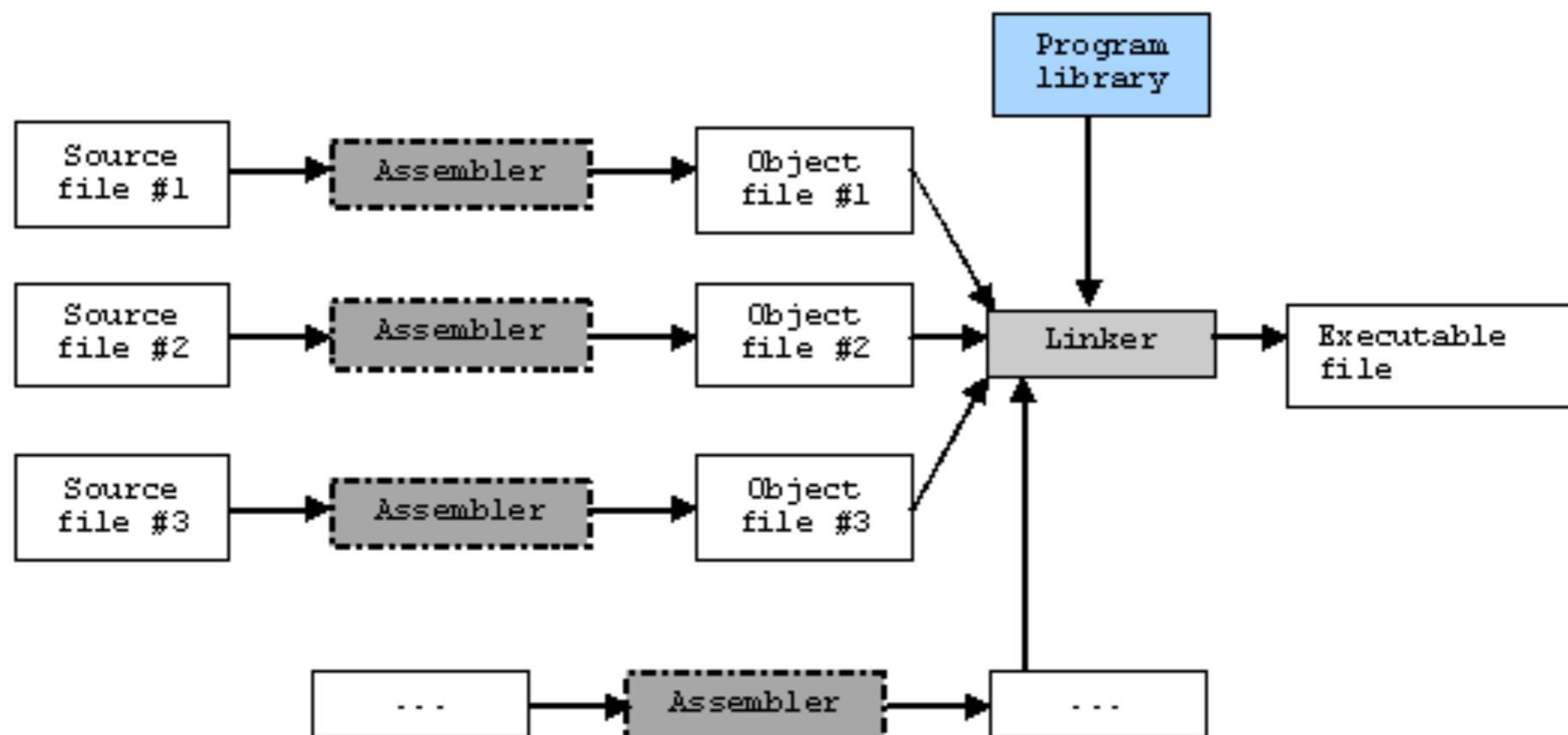
# תהליכי ההידור והקישור

# הידור, קישור וטעינה

- מהדר (assembler) או מאסף (compiler)
- מתרגם את קוד האסמבילר לרצף בתים המייצגים את הנתונים (data) ואת פקודות המכונה (instructions) המתאימות
- מייצר קובץ אובייקט file.obj
- מקשר (linker)
- מקשר בין קבצי אובייקט Shonis, ומיציר מהם קובץ הריצה file.exe ייחיד
- טוען (Loader)
- טוען את קובץ הריצה ל זיכרון, ומתחילה את הרצתו

# הידור, קישור וטיעינה

- התהlixir ממיר את קוד שפת האסמלבלר לקובץ הריצה  
שמוכרך ע"י מערכת הפעלה



# דוגמא

```
.MODEL small

.data
x_pos dw 0
y_pos dw 0

.code

START: MOV AX, x_pos
 MOV BL, 8
 MUL BL
 MOV y_pos, AX

END START
```

# דוגמא

Microsoft (R) Macro Assembler Version 6.11

03/16/17 18:52:46

Page 1 - 1

לומד

לובת"

"תוכן"

.MODEL small

0000 .data  
0000 0000 x\_pos dw 0  
0002 0000 y\_pos dw 0

0000 .code  
0000 A1 0000 R START: MOV AX, x\_pos  
0003 B3 08 MOV BL, 8  
0005 F6 E3 MUL BL  
0007 A3 0002 R MOV y\_pos, AX

התחלת סגמנט

END START

# דוגמא

Microsoft (R) Macro Assembler Version 6.11  
demo.asm Page 1 - 1

03/16/17 18:52:46

```
.MODEL small

0000 .data
0000 0000 x_pos dw 0
0002 0000 y_pos dw 0

0000 .code
0000 A1 0000 R START: MOV AX, x_pos
0003 B3 08 MOV BL, 8
0005 F6 E3 MUL BL
0007 A3 0002 R MOV y_pos, AX

END START
```

הכתובות יחסיות  
**לתחילת הסגמנט**

# דוגמא (המשר)

Microsoft (R) Macro Assembler Version 6.11  
demo.asm Symbols 2 - 1

03/16/17 18:58:02

Segments and Groups:

	Name	Size	Length	Align
Combine Class				
DGROUP	. . . . . . . . . . . . . . . . . . GROUP			
_DATA	. . . . . . . . . . . . . . . . . . 16 Bit	0004	Word	Public 'DATA'
_TEXT	. . . . . . . . . . . . . . . . . . 16 Bit	000A	Word	Public 'CODE'

# דוגמא (המשר)

Symbols:

	Name	Type	Value	Attr
@CodeSize	.....	Number	0000h	
@DataSize	.....	Number	0000h	
@Interface	.....	Number	0000h	
@Model	.....	Number	0002h	
@code	.....	Text	_TEXT	
@data	.....	Text	DGROUP	
@fardata?	.....	Text	FAR_BSS	
@fardata	.....	Text	FAR_DATA	
@stack	.....	Text	DGROUP	
START	L Near	0000	_TEXT	
x_pos	Word	0000	_DATA	
y_pos	Word	0002	_DATA	

0 Warnings  
0 Errors

L=Label  
P=Proc

# מאספ / מהדר

- מהדר (compiler) או מאספ (assembler) או מאספ (compiler) או מאספ (assembler) או מאספ (assembler)
- מתרגם את קוד האסמבילר לפקודות המכונה המתאימות (macros והנחיות אסמבילר directives ומחליף אותם בקוד מתאים)
- שומר מקום בזיכרון עבור משתנים ונתונים שהוגדרו
- "מניח" כתובות מסויימות עבור משתנים/שגרות בהם נעשה שימוש (גם אם אינם מוגדרים בקובץ) - **מסמן שכתובות אלו עלולות להשנות**
- פלט: קובץ אובייקט (Object File, file.obj) (j)
- מכיל את רשימת פקודות המכונה (instructions) ותוכן זיכרון הנתונים
- מכיל את רשימת כל ה"סימבולים" שקיים בקובץ (משתנים/תיוות/סגמנטים/שגרות)
- סימון רשימת משתנים / שגרות ש"הנחנו" את מיקומם

# דוגמא

0000 .code

0000 33 DB

0002 B8 0001

0004 8B D8

0006 EA 0002

000B 33 DB

R

XOR BX,BX

START: ADD AX,1

MOV BX,AX

JMP FAR PTR START

XOR BX, BX

הקומpileר שומר  
מקום כנדראש

מדוע המשך קידוד הפקודה חסר?

מצין שזו כתובות relocatable.  
ה-linker אולי יתקן את פקודת  
המכונה (היסט) לאור הזיה  
עתידית של הסגמנטים

# הידור, קישור וטעינה

- **קשר (linker):**
- **קשר בין קבצי אובייקט שונים:**
- **אחד סגמנטים שונים לתוכנית רציפה, משנה **סדר** סגמנטים אם נדרש**
- **כל השימושים במשתנה/שגרה "מוזיים"** ייחסית אל הכתובת של המשתנה היחיד שהוגדר, בהתאם לסדר הסגמנטים החדש (relocation)
- מודא שכל משתנה/שגרה שנעשה בו שימוש (בקובץ אחד או יותר) **מוגדר** במקום כלשהו
- (אם FAR:) כתובות זו **עלולה** להשתנות בעתיד!  
בעת הקישור לא ידוע איפה המשתנה הנ"ל יישב בזיכרון.
- **פלט: תוכנית הריצה (file.exe)**

# דוגמא 1: סדר סגמנטים

Microsoft (R) Macro Assembler Version 6.11  
demo.asm

03/16/17 18:52:46

Page 1 - 1

```
.MODEL small
0000 .data
0000 0000 x_pos dw 0
0002 0000 y_pos dw 0
0000 .code
0000
000 A1 0000 R START: MOV AX, x_pos
0003 B3 08 MOV BL, 8
0005 F6 E3 MUL BL
0007 A3 0002 R MOV y_pos, AX
END START
```

בעת הקישור, linker עשוי לשים את ה data מייד אחרי הקוד.

במקרה זה אין דרך לקבוע את DS כך ש x\_pos יהיה ב[0000:!!DS]

פקודות מכונה אלו אינן סופיות!  
ה-linker יתכו לפי מיקום היחסי ל-DS

# דוגמה :2 extern

הצהרה על label

שינגדר בקובץ אחר

EXTRN UP:FAR

0000 33 DB  
0002 B8 0001  
0005 E9 02F9

XOR BX,BX  
START: ADD AX,1  
JMP NEXT

<skipped memory locations>

0300 8B D8  
0302 EA 0002 — R  
0307 EA 0000 — E

NEXT: MOV BX,AX  
JMP FAR PTR START  
JMP UP

הכתובת תושם (חלקית)  
בעת ביצוע link

# הידור, קישור וטעינה

- טוען (loader):
- מייצר נתוני תוכנית PSP (מגיל, למשל, אורך התוכנית, מצביע לתוכנית הקודמת...).
- טוען את התוכנה + PSP לאיזור פנוי בזיכרון (מערכת הפעלה דואגת לשמר רשימה של איזה איזור פנוי ואיזה לא)
- מעדכן את DS לתחילת הסגמנט אליו נטענה התוכנה;  
معدכו את SP:SS לאיזור זיכרו עבור המחסנית
- משנה כתובות (אבסולוטיות) של משתנים/שגרות בהתאם למקום אליו נטענה התוכנה

# דוגמה

נניח שהקוד נטען לכתובת 0700:0008

CS = 0700h

AFTER  
LOAD

AFTER  
Compiler

EXTRN UP:FAR

0700:0008

0000 33 DB

XOR BX,BX

0700:000A

0002 B8 0001

START: ADD AX,1

0700:000D

0005 E9 02F9 (0300 R)

JMP NEXT

<skipped memory locations>

0700:0308

0300 8B D8

NEXT: MOV BX,AX

0700:030A

0302 EA 0002 — R

—> 000A 0700

FAR PTR START

0700:030F

0307 EA 0000 — E

—> ??

JMP UP

# Program Segment Prefix (PSP)



- איזור בגודל **100 בתים** שמשמש את מערכת הפעלה לשמר נתונים על התוכניות המופעלות בה
- לכל תוכנית - PSP משל עצמה
- כאשר נבקש ממ"ה לטעון תוכנית לזיכרון
  - ה-loader מגדר PSP לאוთה תוכנית
  - ה-loader שומר את ה-PSP בזיכרון מיד לפני התוכנית
- אוגר DS יציביע לתחילה איזור ה-PSP

# Program Segment Prefix (PSP)

<b>Offset</b>	<b>Length</b>	<b>Description</b>
0	2	exit (INT 20h) instruction is stored here
2	2	Program ending address
4	1	Unused, reserved by DOS
5	5	Call to DOS function dispatcher
0Ah	4	Address of program termination code
0Eh	4	Address of break handler routine
12h	4	Address of critical error handler routine
16h	22	Reserved for use by DOS (parent PSP segment)
2Ch	2	Segment address of environment area
2Eh	34	Reserved by DOS (dos version, internal pointers)
50h	3	INT 21h, RETF instructions
53h	9	Reserved by DOS
5Ch	16	Default FCB #1
6Ch	20	Default FCB #2
80h	1	Length of command line string
81h	127	Command line string

“שורת הפקודה”

# .model ההנחיה |

- ההנחיה מושםת את המהדר/קשר בקביעת פרישת הזיכרון והסגמנטציה, בהתאם לגודל התוכנית
- 

**Table 2.1 Attributes of Memory Models**

Memory Model	Default Code	Default Data	Operating System	Data and Code Combined
Tiny	Near	Near	MS-DOS	Yes
Small	Near	Near	MS-DOS, Windows	No
Medium	Far	Near	MS-DOS, Windows	No
Compact	Near	Far	MS-DOS, Windows	No
Large	Far	Far	MS-DOS, Windows	No
Huge	Far	Far	MS-DOS, Windows	No

# ההנחיה | .model

## • MODEL TINY •

- מייצר תוכנית מסוג ".COM". מ
  - מivilah אר וرك את ה-instructions (נטענת so-as לזכור)
- כל התוכנה בסגמנט **יחיד** - גודל התוכנית מוגבל ל-64ק"ב
- הטוען יאותחל SS=DS=CS להציביע על תחילת ה-PSP.
- הטוען יאותחל את IP לפקודה הראשונה לאחר ה-PSP. כולם הפקודה שבמיקום 100. תוכן קובץ ה-.COM ייטען לכתובת זו.
- המחסנית ממוקמת **בסוף** המקטע גודלה בגודל הזיכרון הלא-מנוצל שנותר בסגמנט שהוקצה

# דוגמא לתוכנית TINY

```
.MODEL tiny
.code

org 100h ; MUST: put first instruction @100h from CS

COM: jmp start ; jump over data declaration

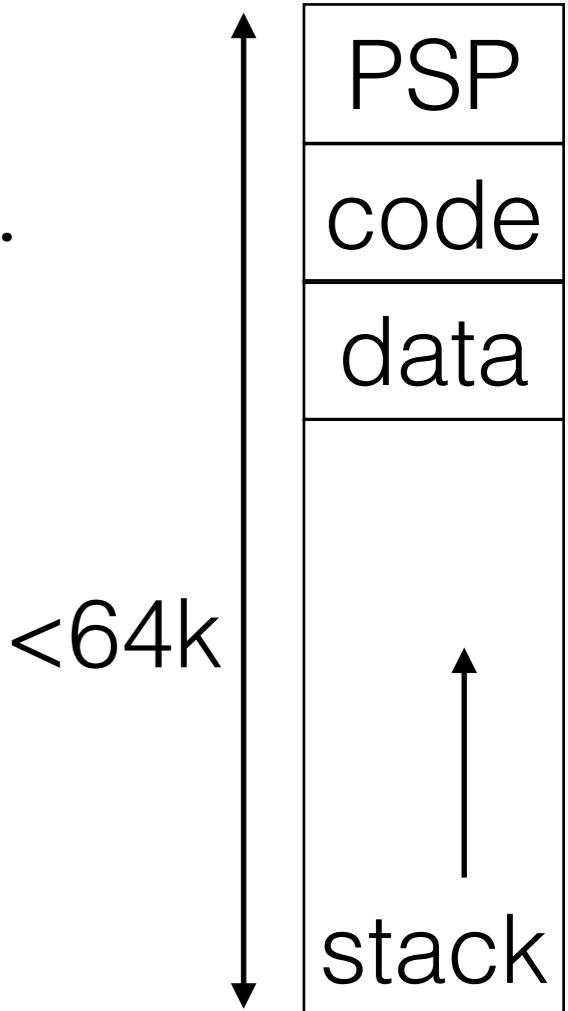
msg db "Hello, World!", 0Dh, 0Ah, 24h

start: LEA dx, msg ; load offset of msg (in DS) into dx.
 mov ah, 09h ; print function is 9.
 int 21h ; do it!

 mov ah, 0
 int 16h ; wait for any key.....

.exit ; return to MS-DOS

END COM
```



# ההנחיה model.

- שאר המודלים (small,large,...) מייצרים תוכנית EXE.
- קובץ exe מכיל את הפקודות ומידע נוסף:
  - מידע relocation
- מידע איתחול סגמנט מחסנית ואוגר SP
- נקודת ההתחלה של התוכנה:
  - איתחול IP:CS
- לאחר הטעינה: DS מציביע **תמיד** לתחילת ה-PSP.  
תוכנת האסמבלר צריכה לעדכו את DS לאוגר הנתונים  
בעצמה, למשל, ע"י
  - MOV AX, SEG DSEG
  - MOV DS, AX

# ההנחיה |.model

## .MODEL SMALL •

- מניח ש-DS/CS לא משתנים, גודל תוכנית ונתונים כל-אחד קטנים מ-64ק"ב

## .MODEL MEDIUM •

- מניח שקוד התוכנית גדול, ו-CS עלול לשנות במהלכה (= מספר סגמנטי קוד שונים).
- כבירית מחדל יaddir שגרות כ-FAR ויבצע קפיצות FAR

# .model להנחיה

## .MODEL LARGE •

- כתת מניה שגם מקטע הנתונים גדול מסגמנט יחיד ו-DS משתנה במהלך הריצה

- גישה למשתנים צריכה להתחשב בסגמנט

MOV AX, msg

- למשל, במקום

نبצע:

MOV AX, SEG msg

MOV DS, AX

MOV AX, DS:msg

# TSR

# Terminate & Stay Resident

# TSR

- מרבית התוכנות נטעןות לזכרון, ובסיוםו "יוצאות" חזרה אל מערכת הפעלה (MOV AH, 4Ch INT 21h)
- בסיום הריצה, מערכת הפעלה מקaza את הזיכרון מחדש לתוכניות הבאות
- תוכנת **TSR** אינה משחררת את הזיכרון לאחר סיוםה ולמעשה **משיכה לפעול / לשכון בזכרון** עד איתחול המחשב
- שימוש: החלפת שגרות פסיקה, ניהול זיכרון, וירוסים..

# TSR

- **מימוש TSR ע"י פסיקת INT 27h , MS-DOS**
- קלט:
  - **DX** מכיל את כמות הבטים שיש לשמר בזיכרון (יחסית לתחילת ה-PSP)
  - **CS** מצביע ל-PSP (לכו, עדיף [עדיף tiny !!.model](#))
- פלט: אין
- פעולה: מסיימת את פעולת התוכנית (וחזרת למ"ה), אבל משאיר בזיכרון את כמות הבטים הרשומה ב-**DX**
- הערות: ניתן להשאיר בזיכרון לכל היותר סגמנט יחיד (64k).  
INT 21h, AH=31h, תחליף: בפסקה המקורית יש באג (אם  $32k > DX$ )

# מבנה תוכנת TSR

- לרוב תוכנת TSR מורכבת משני חלקים:
  1. קוד איתחול
  2. קוד שנשאר בזיכרון
- **האיתחול** - קוד שפועל פעם אחת ולא נדרש להשאר בזיכרון.  
למשל: הקוד שמעדכן את וקטור הפסיקה לבתובת החדשה
- **הקוד שנשאר** - מכיל את פעלת ה- TSR ונדרש להיות בזיכרון לאורך כל פעולה ה- TSR.  
למשל: קוד שגרת הפסיקה ...

# אורים: מקורות

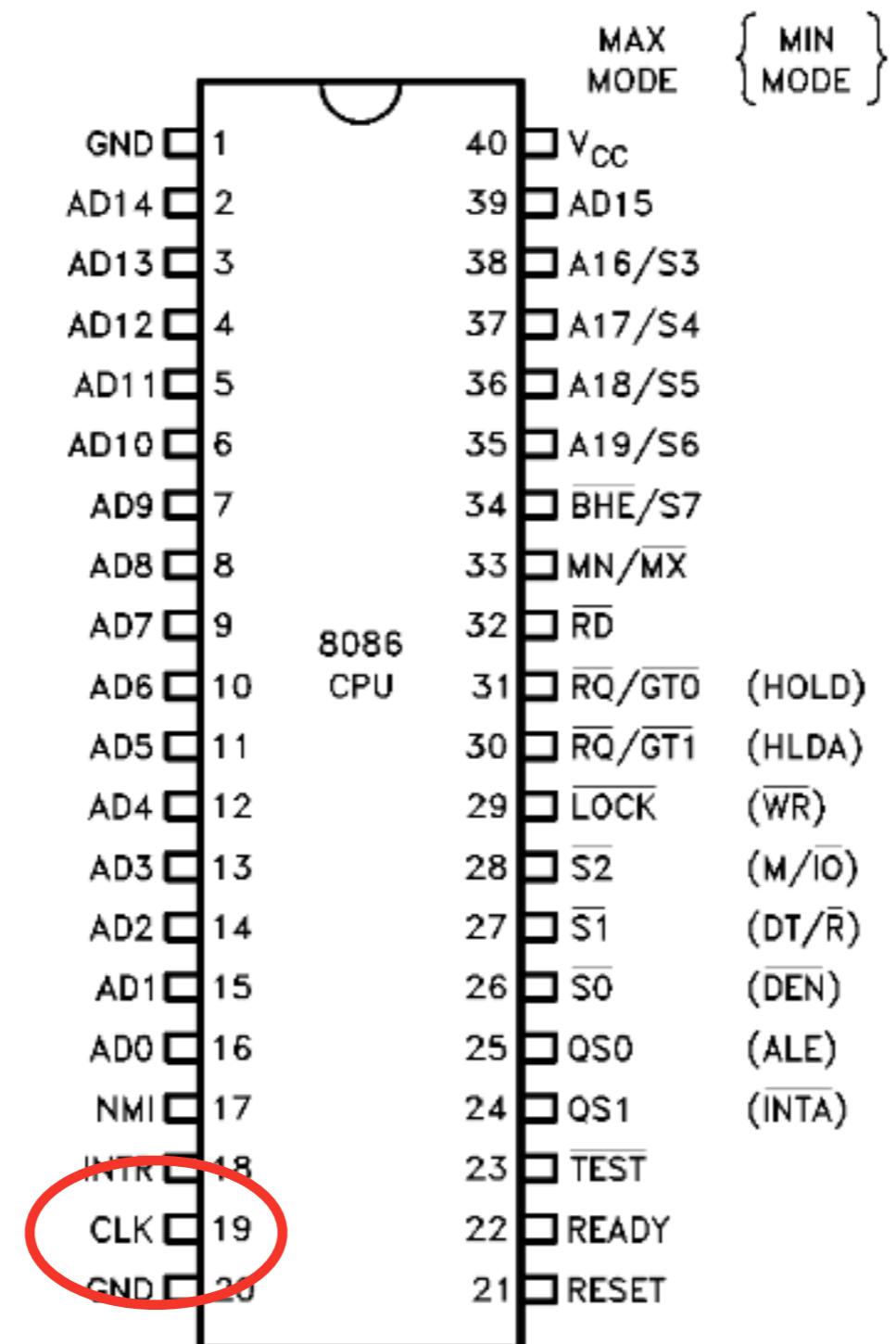
- <http://www.tenouk.com/ModuleW.html>
- Further Reading About TSR:
  - <https://www.scribd.com/doc/104797784/TSR-Terminate-and-Stay-Resident>
  - <https://codesandhacks.wordpress.com/2015/06/27/write-a-tsr-program-in-8086-alp-to-implement-real-time-clock rtc-read-the-real-time-from-cmos-chip-by-suitable-int-and-function-and-display-the-rtc-at-the-bottom-right-corner-on-the-screen-access/>

# התקנים חיצוניים

מיקרומעבדים וشفת אסמלר 83-255-83



# שעון המערכת



231455-2

40 Lead

# שעון המערכת

- המעבד הוא מנגנון לוגי סידרתי סינכרוני המזמין ע"י שעון חיצוני
- כל פעולה פנימית במעבד מתבצעת בעקבות מאורע של עליית שעון או של ירידת שעון
- גישה לזכרו (וכן ביצוע "פקודות" כגון ADD, MOV וכו') מבוצעות לאורך מספר מחזורי שעון.
- משך ביצוע פקודה משתנה לפי מורכבותה:  
למשל **מייעון אוגר** - 2 מחזורי שעון;  
**מייעון עקיף** - מעל 10 מחזורי שעון (פירוט בהמשך)

# משמעות פקודה (1)

Table 2-21. Instruction Set Reference Data (Cont'd.)

MOV	MOV destination,source Move				Flags	O	D	I	S	Z	A	P	C
	Operands	Clocks	Transfers*	Bytes		Coding Example							
memory, accumulator		10	1	3		MOV ARRAY [SI], AL							
accumulator, memory		10	1	3		MOV AX, TEMP_RESULT							
register, register		2	—	2		MOV AX,CX							
register, memory		8+EA	1	2-4		MOV BP, STACK_TOP							
memory, register		9+EA	1	2-4		MOV COUNT [DI], CX							
register, immediate		4	—	2-3		MOV CL, 2							
memory, immediate		10+EA	1	3-6		MOV MASK [BX] [SI], 2CH							
seg-reg, reg16		2	—	2		MOV ES, CX							
seg-reg, mem16		8+EA	1	2-4		MOV DS, SEGMENT_BASE							
reg16, seg-reg		2	—	2		MOV BP, SS							
memory, seg-reg		9+EA	1	2-4		MOV [BX].SEG_SAVE, CS							

- סוג המיעון מופיע על משך הפקודה
- EA - משך חישוב הכתובת האבסולוטית

# משר פקודה (2)

**Table 2-20. Effective Address Calculation Time**

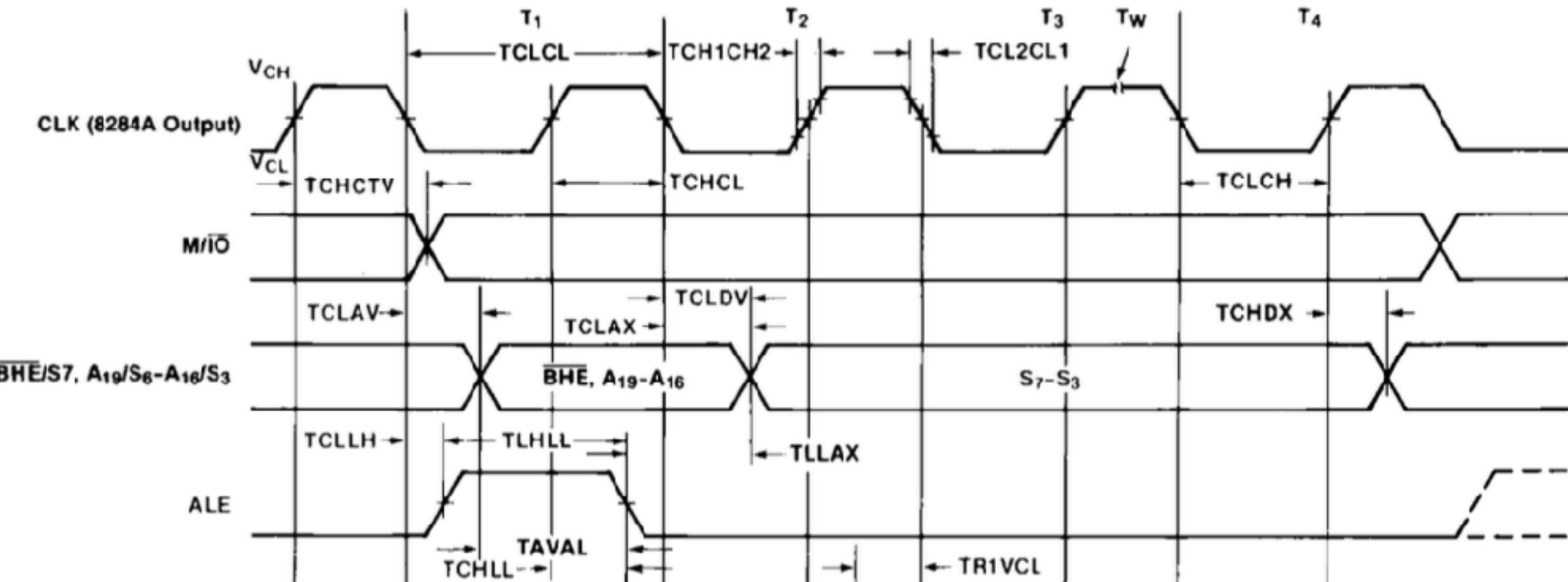
EA COMPONENTS	CLOCKS*
Displacement Only	6
Base or Index Only (BX,BP,SI,DI)	5
Displacement + Base or Index (BX,BP,SI,DI)	9
Base BP + DI, BX + SI + Index BP + SI, BX + DI	7
Displacement BP + DI + DISP + BX + SI + DISP Base + BP + SI + DISP Index BX + DI + DISP	11
	12

\*Add 2 clocks for segment override

- **EA** - כל שחייב הכתוב מרכיב יותר, הפעולה נמשכת יותר מחזורי שעון



# שעון המערכת: מגבלות

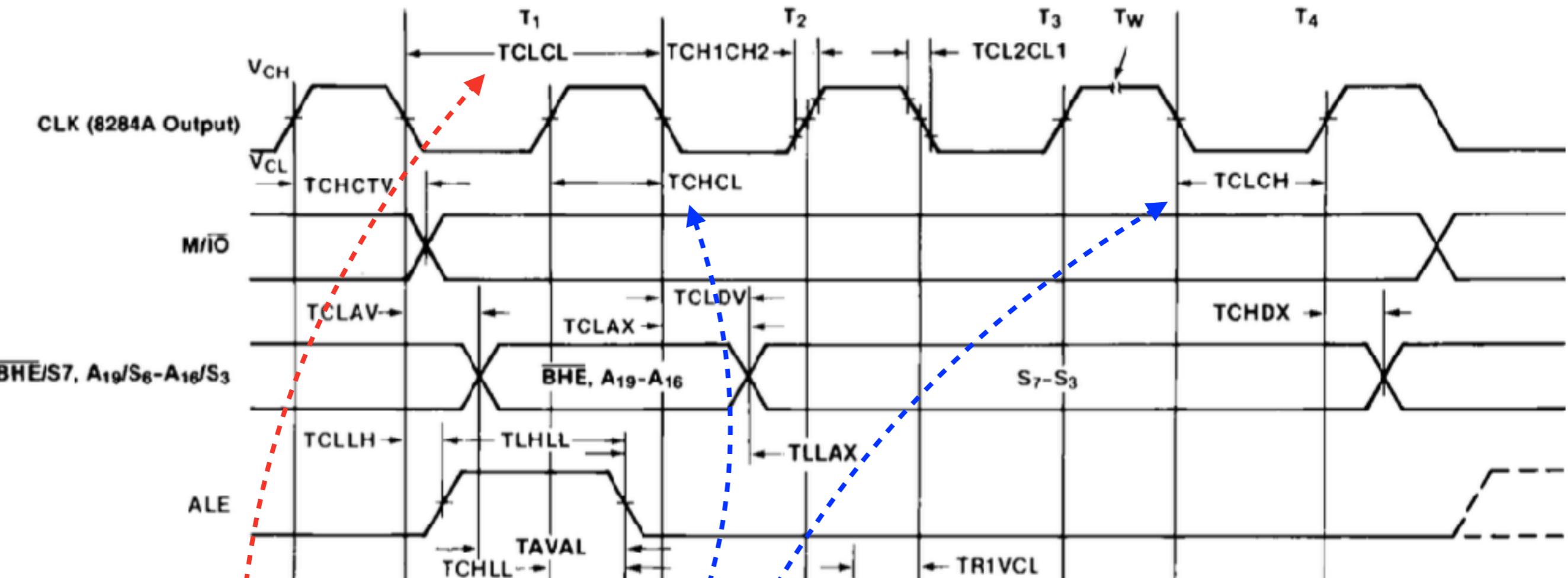


MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	



# שעון המערכת: מגבלות



MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

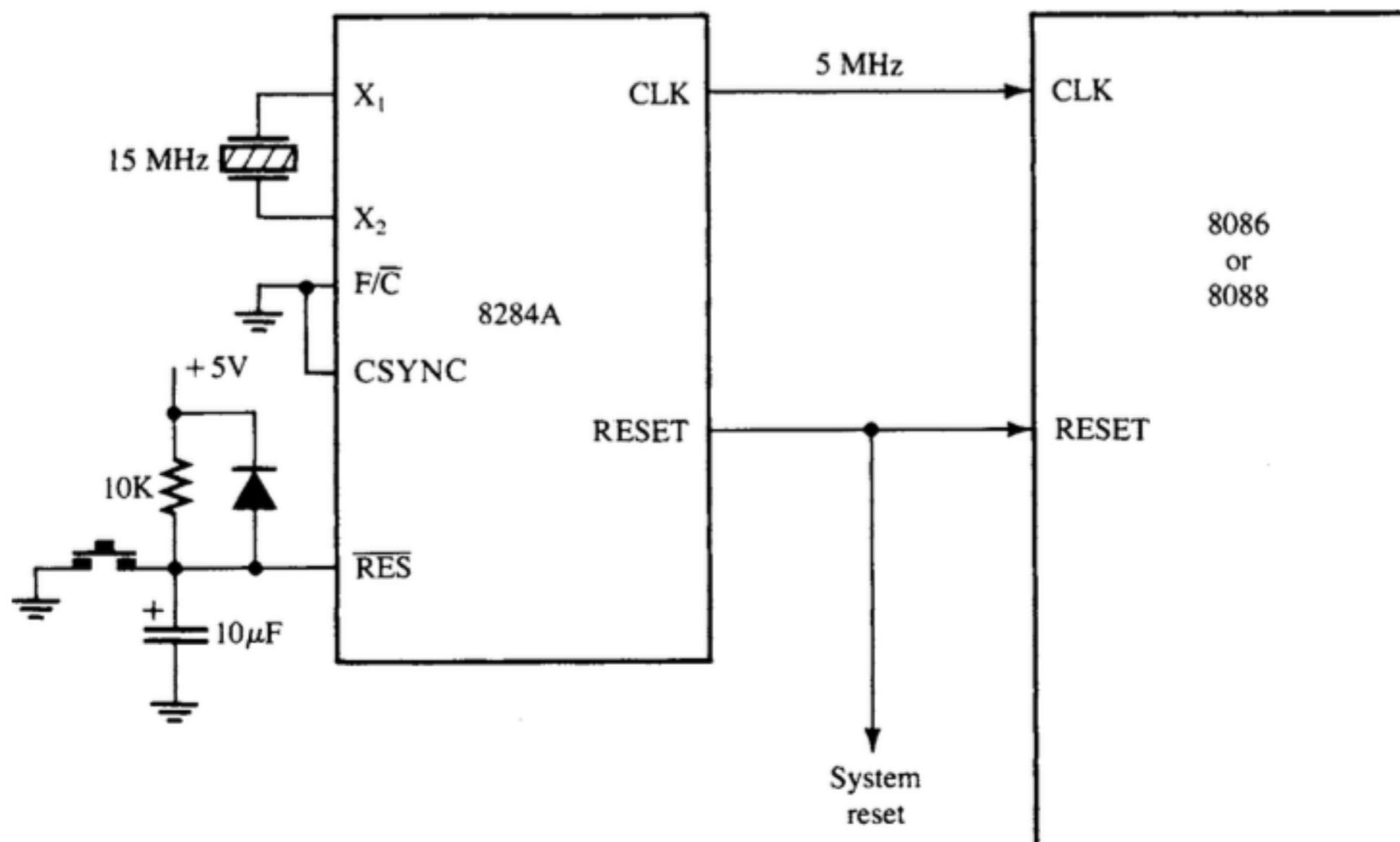
Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time			10		10		ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time			10		10		ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	

# מעגל ייצור שעון

- אותן השעות מיוצר מגביש (מתנד)
- הגביש מגיב למתח חשמלי ומיציר אותן מחזורי בתדר של  $C\text{-}fz$ , duty cycle 15Mhz, בערך 50%.
- אותן הגביש מוכנס לרכיב **A8284**
- תפקידו “לנקות” וליציב את אותן המחזורי
- גוזר גם בקווים RESET (נדרש גבואה ל-4 מחזוריים לפחות) ומסנכרנו גם את קו READY



# מעגל ייצור שעון



# שליטה חיצונית

פקודת WAIT, LOCK, ושליטה חיצונית על הפס

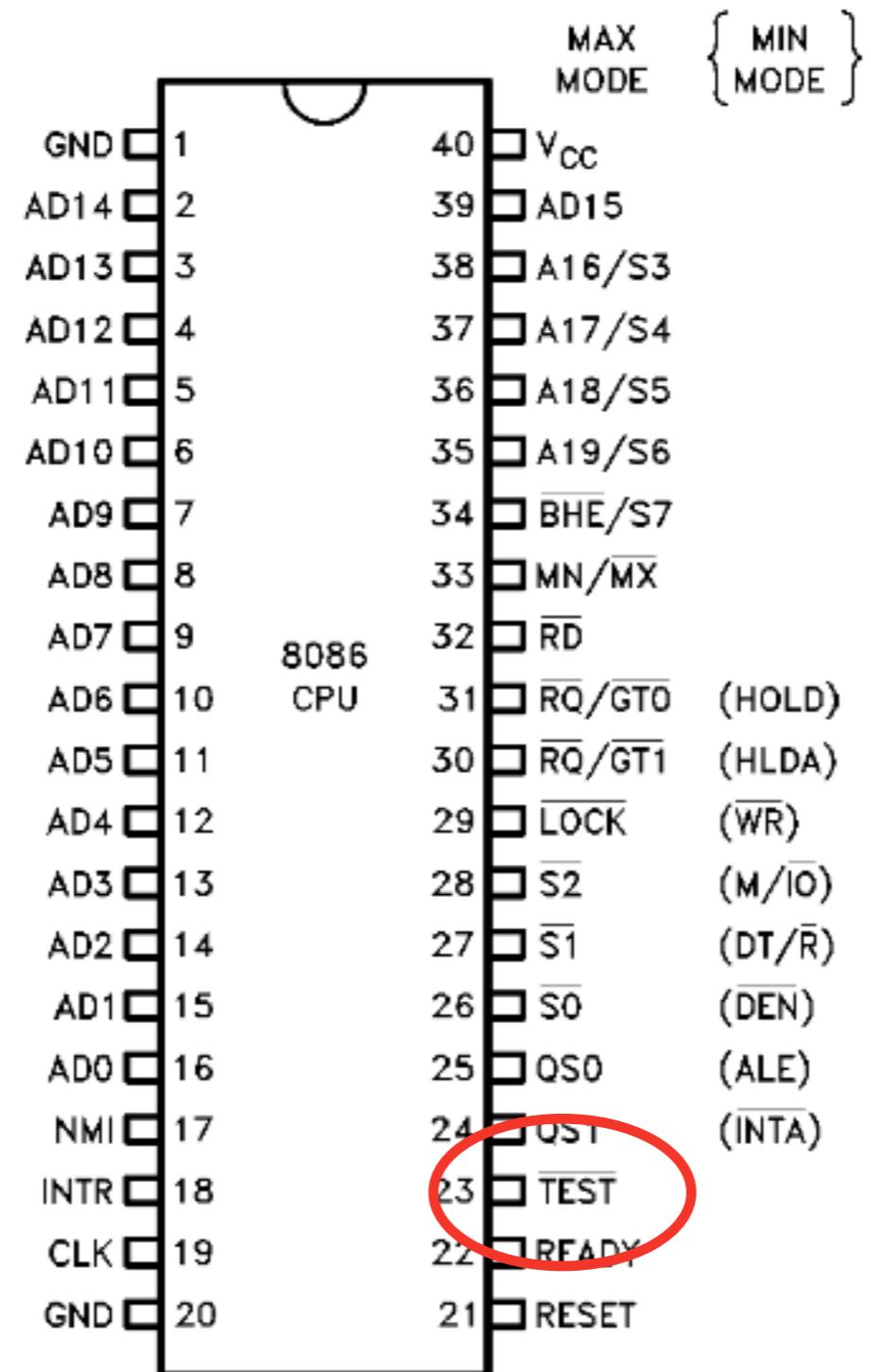
# פקודה WAIT

- הפקודה WAIT מאפשרת "לעצור" את המעבד עד שתרחש אירוע חיצוני
- הפקודה אינה מקבלת אופרנדים

WAIT

- המשך הפעולות כאשר קו TEST יורד ל-0
- משמש לרוב להמתנה למעבד-עזר, דוגמאות 7087 (מעבד מתמטי)

# פקודת WAIT



231455-2

40 Lead

# העברה שליטה להתקן חיצוני

- לעתים נדרש כי התקן חיצוני יישלוט בפס
- למשל: ייקרא נתונים מהזיכרון.
- אם שני רכיבים יכתבו לפס תיווצר **התנגשות שימושית**
- בדרך כלל, המעבד מחזיק את השליטה בפס, אך הוא יכול להעביר את השליטה להתקן אחר, בתחילת שנקרא "לחיצת-יד" (handshake)

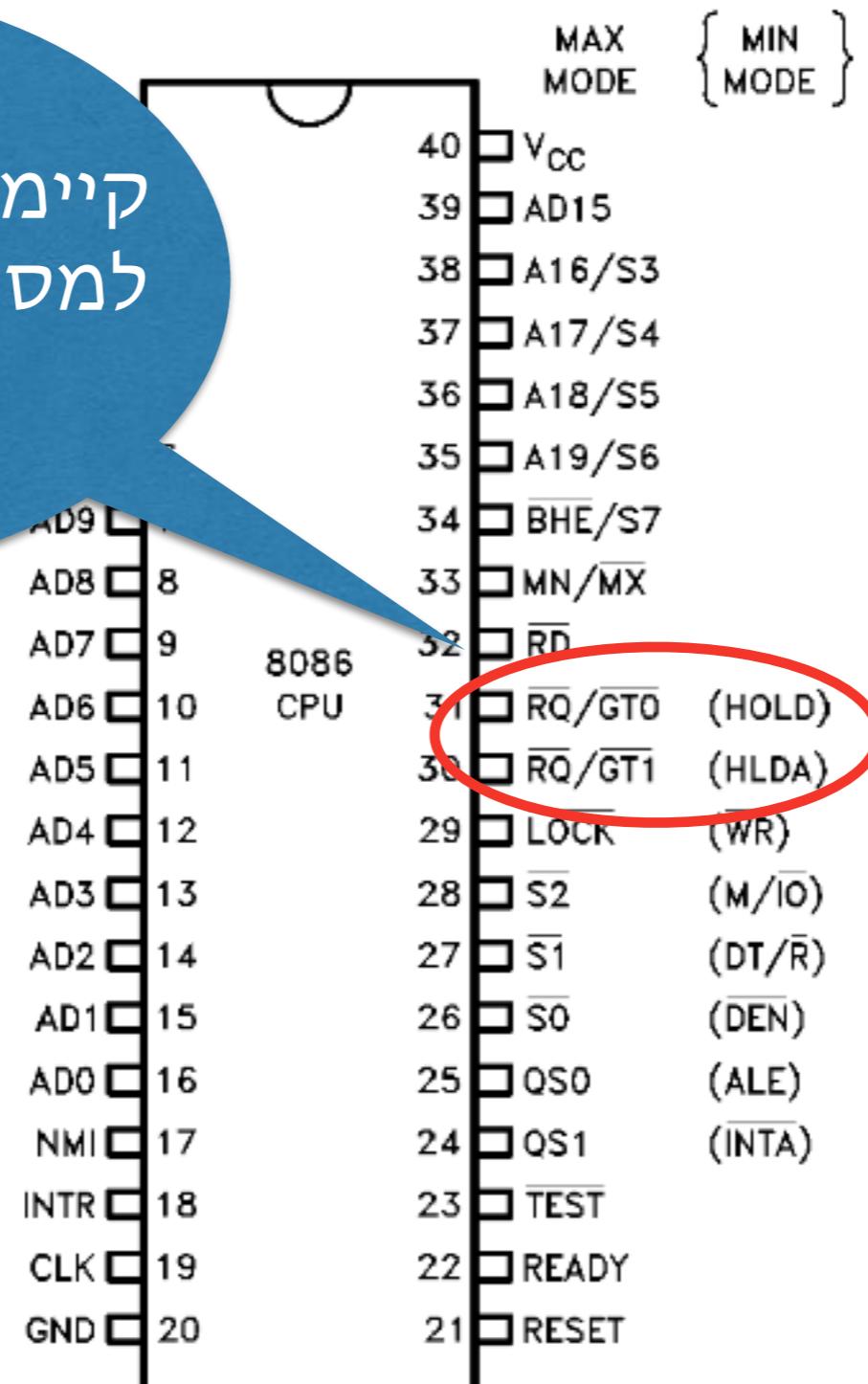
# העברה שליטה להתקן חיצוני

- **הערה:**  
קיימות **שתי** שיטות להעברת שליטה  
(בהתלות בكونFIGורציית המעבד).
- אנחנו נתמקד רק באחת מהן -  
(רגל 33 מוארקט לאדמה)  
Maximum Configuration -
- התקן מבקש (ומקבל) גישה בעזרת קו סטוס יחיד

$$\overline{RQ}/\overline{GT}$$

# העברת שליטה להתקו חיצוני

קיים שני קווי בקשה,  
למספר 0 יש עדיפות על  
מספר 1.



231455-2

# העברה שליטה להתקן

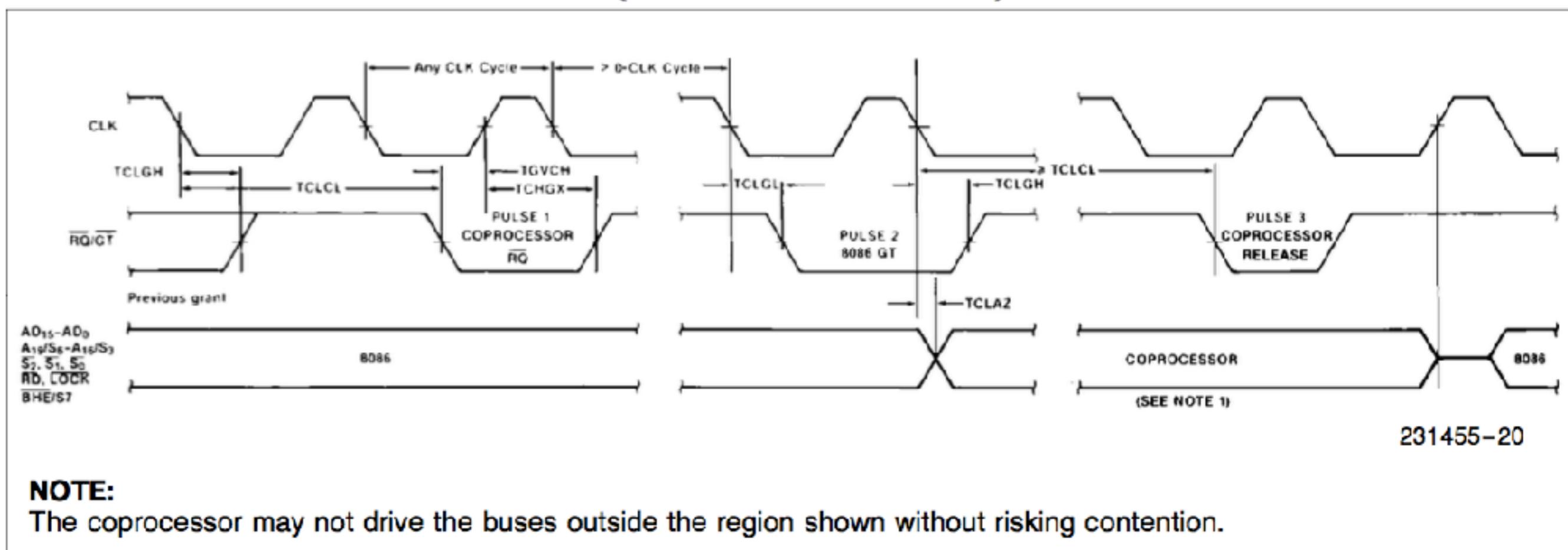
## חיצוני

- תהליך העברת שליטה תלת-שלבי:
  1. **התקן** מבקש שליטה ע"י הורדת קו  $\overline{RQ}/\overline{GT}$  ל-0 למשר פולס יחיד. **לאחר מכן התקן מפסיק לכתובuko**
  2. כאשר **המעבד** מוכן להעברת השליטה, הוא כותבuko הנ"ל 0 למשר פולס אחד.  
 כתיבה זו מסמנת **שהמעבד אינו** שולט על הפס יותר  $\overline{RQ}/\overline{GT}$ .
  3. כאשר התקן מסיים את השימוש בפס, הוא כותב 0 ל-  
 המעבד **ישתלו** על הפס החל מחזור השעון לאחר מכן.

קו מוחזק גבוה ע"י pull-ups, אלא אם התקן מאלץ אותו ל-0

# העברה שליטה להתקן חיצוני

## REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



# העברה שליטה להתקו חיצוני

- מרגע שהשליטה עברה להתקו חיצוני, החזורת השליטה תלוייה בהתקו החיצוני ולא במעבד.
- מה קורה בעת קבלת פסיקה, או זמנים?
- לקו  $\overline{RQ}/\overline{GT}$  עדיפות על פני פסיקה
- הטיפול בפסקה יבוצע מיד כאשר השליטה בפס תחזור אל המעבד
  - (המעבד זוקק לפס כדי לטפל בפסקה, למשל לברר את מספר הפסקה...)

# דוגמה:

## מעבד-עזר מתמטי 8087



- **פועלות מתמטיות "כבדות" נעות ע"י מעבד-עזר מתמטי 8087 Intel 8087**
- יודע לבצע פועלות על מספרים ממשיים ברמת דיוק של 64 ביט (לעתים 80 ביט): שורש, חזקה, 4-פועלות חשבון, וכו'
- מעבד העזר מחובר במקביל על כל הפסים ומקשיב להם. חלק מהאופקودים שאינם בשימוש ה-8086 משמשים את ה-8087

# דוגמה:

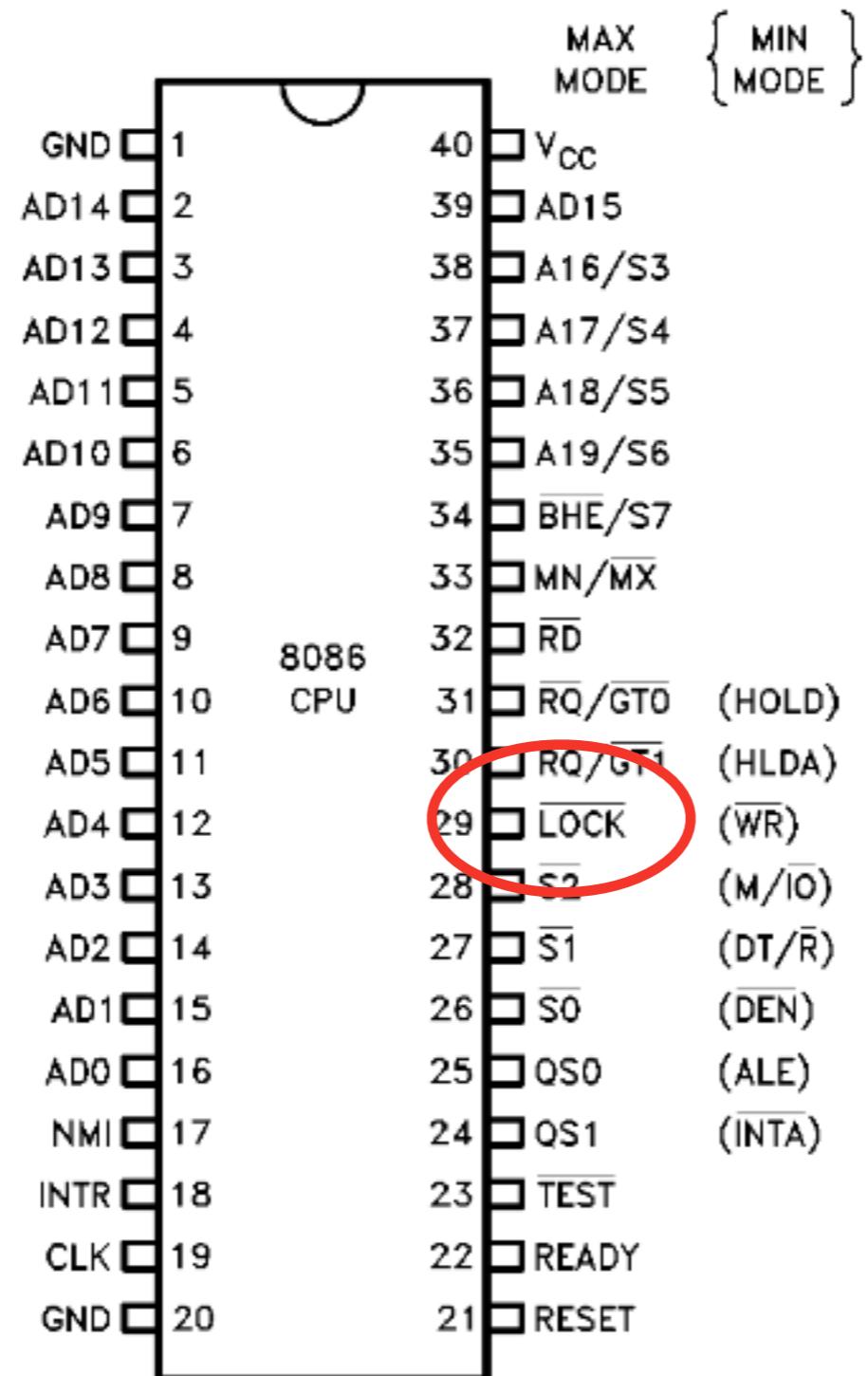
## מעבד-עזר מתמטי 8087

- פקודות ל-8087 נקראות ע"י ה-8086 בהליך הרגיל (ה-87 מאזין לפס ופועל לפי הפקודות שה-86 קורא)
- כדי לקרוא/לכתוב נתונים ל זיכרונו, ה-8087 מבקש גישה לפס ע"י  $\overline{RQ}/\overline{GT}$
- חישוב פעולה מתמטית נמשכת מספר רב של מהזורי שעון (לדוגמה: כ-250 מהזורים עבור חילוק) ה-8087 מחזק את קו  $\overline{\text{TEST}}$  גבוה למשך הפעולה. ה-8086 ממשיך פעולה תקינה במקביל.
- בסיום החישוב המתמטי, ה-8087 מוריד את קו  $\overline{\text{TEST}}$ . ה-8086 יכול לזהות סיום פעולה ע"י פקודת WAIT

# פקודת LOCK

- הפקודה LOCK מונעת מרכיב חיצוני לפנות אל זיכרון משותף למשר ביצוע הפקודה של אחרת
- LOCK ADD AX,5
- ניתן להשתמש רק לפני הפקודות ADD, ADC, AND, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR, and XCHG
- אחרת, תיגרם פסיקת חריגה של פקודה לא נcona

# פקודת LOCK



231455-2

40 Lead

# פקודת LOCK

- הפקודה **LOCK** גורמתuko **LOCK** לרדת לערך-0 ומסמנת לרכיבים חיצוניים כי הם לא יכולים להשתלט על הפס או לגשת לזיכרון משותף
- לאחר ביצוע הפקודה ה"**"נעולה"**uko **LOCK** חוזר ל-1 והמעבד מטפל בבקשתות RQ/GT שנתקבלו
- שימוש: הפיכת פעולה **לאוטומטית** למשל, ביצוע פקודות HCGH ( מבוצע ע"י שתי גישות נפרדות לזיכרון ) או במערכת מרובת מעבדים ( כתיבת סמפור בזיכרון משותף )

# רכיב DMA

גישה ישירה לזיכרון - Direct Memory Access

# התקן DMA

- התקני I/O שונים צריכים לעיתים לקרוא או לכתוב לזיכרון  
כמוניות מידע גדולות
  - המעבד נדרש לבצע את המעבר בין הזיכרון לתקן
- ← פעולה איטית וציבורנית

```
DO: IN AX, DX ; read from I/O
 MOV [BX], AX ; write to mem
 ADD BX, 2
 LOOP DO
```

# התקו DMA

- התקו DMA מאפשר העברת מידע בין הזיכרון לתקו ללא מוערבות של המעבד
- השליטה בפס עוברת ל-DMA שמנהLAT את העתקת המידע
- המעבד יכול המשיך בפעולות אחרות (למעט גישה לפס)
- ניתן גם לבצע ע"י DMA העברת מידע זיכרון-זיכרון

# התקן DMA

- יתרונות:
  - קלות הפעלה והעברת מידע: המעבד צריך להגדיר ל-DMA כתובת מקור / יעד וגודל, ולהפעיל את ההעברה.
  - קצב העברת המידע נקבע רק לפי הגבלת התקן/זיכרון ויכולת DMA ולא לפי קצב המעבד (כזכור, חישוב EA יכול להגיע ל-10 מהזרוי שעון) **(כיוון פחות משמעותי, כי קצב המעבד גבוה בהרבה מקצב זיכרון)**
- חסרונות:
  - המעבד למשה "מושבת" ללא גישה לפס במרקם מסוימים מאט את פעילות המערכת בזמן העברת הנתונים (עקב חסימת הפס)
  - עלול ליצור בעיות אם למעבד יש זיכרון פנימי (cache)

# תכנות ה-DMA

- בקר ה-DMA מחווט למעבד כרכיב קלט/פלט וניתן לתוכנות על פקודות IN/OUT
- המעבד מגדר:
- כתובת התחלת (בזיכרונו) או כתובת מקור קלט/פלט
- כמות מילים להעברה
- אופן העברה
- בסיום פעולה ההעתקה, ה-DMA יודיע למעבד ע"י פסיקה

# תכנות ה-DMA

- **אופן העברה:**
- העברת רציפה (Burst): ה-DMA מבצע את כל ההעברה בחת אחת ללא שיחזור הפס
- DMACycle Stealing/Interleaving: לאחר העתקת מילה אחת, הDMA מוחזיר את השליטה לمعالג ומבקש שנית גישה (מתאים למערכות בהן לא ניתן להשבית את המمعالג לזמן ארוך..)
- העברת שקופה (Transparant): ה-DMA מבצע את העתקה רק במחזורי שעון בהם המمعالג אינו משתמש בפס (מצרי DMA חכם שמנטור ומזזה פעילות מעבד)
- עבר O/I איטי, ה-DMA ימתין עד שהמילה הבאה מוכנה ורק אז ישתלט על הפס. מאפשר ל-CPU המשיך לפעול **בלי להמתין ל-O/I**

# בקר DMA 8237

- בקר DMA מסוג 8237 מתאים לחברו למעבד 8086

-IOR	1	40	A7
-IOW	2	39	A6
-MEMR	3	38	A5
-MEMW	4	37	A4
(High or VCC)	5	36	-EOP
READY	6	35	A3
HLDA	7	34	A2
ADSTB	8	33	A1
AEN	9	32	A0
HRQ	10	Intel 8237	VCC (+5V)
-CS	11	30	DB0
CLK	12	29	DB1
RESET	13	28	DB2
DACK2	14	27	DB3
DACK3	15	26	DB4
DREQ3	16	25	DACK0
DREQ2	17	24	DACK1
DREQ1	18	23	DB5
DREQ0	19	22	DB6
(GND) VSS	20	21	DB7

- מאפשר העברת מידע של עד 1.6MB/sec
- עד 64kB בהפעלת העברה ייחודית
- משתמש בקוו HOLD/HODA ולא בT/RQ/GT (קונFIGורציה minimum)

# אורים: מקורות

- Intel: 8086 Complete Technical Specification / User Manual, 1979
- B. Brey, The Intel Microprocessor, 2009
- [https://en.wikipedia.org/wiki/Intel\\_8237](https://en.wikipedia.org/wiki/Intel_8237)