

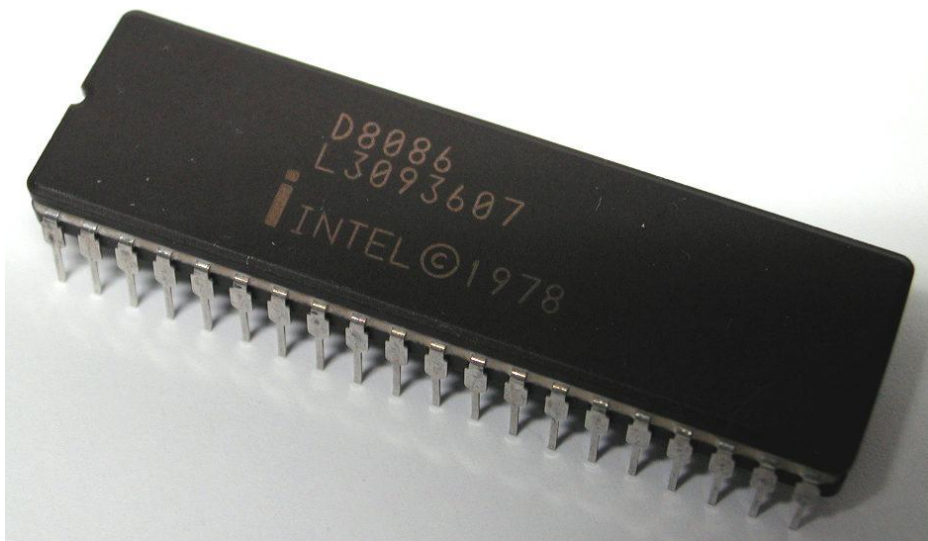


פרק 4 מבנה המחשב

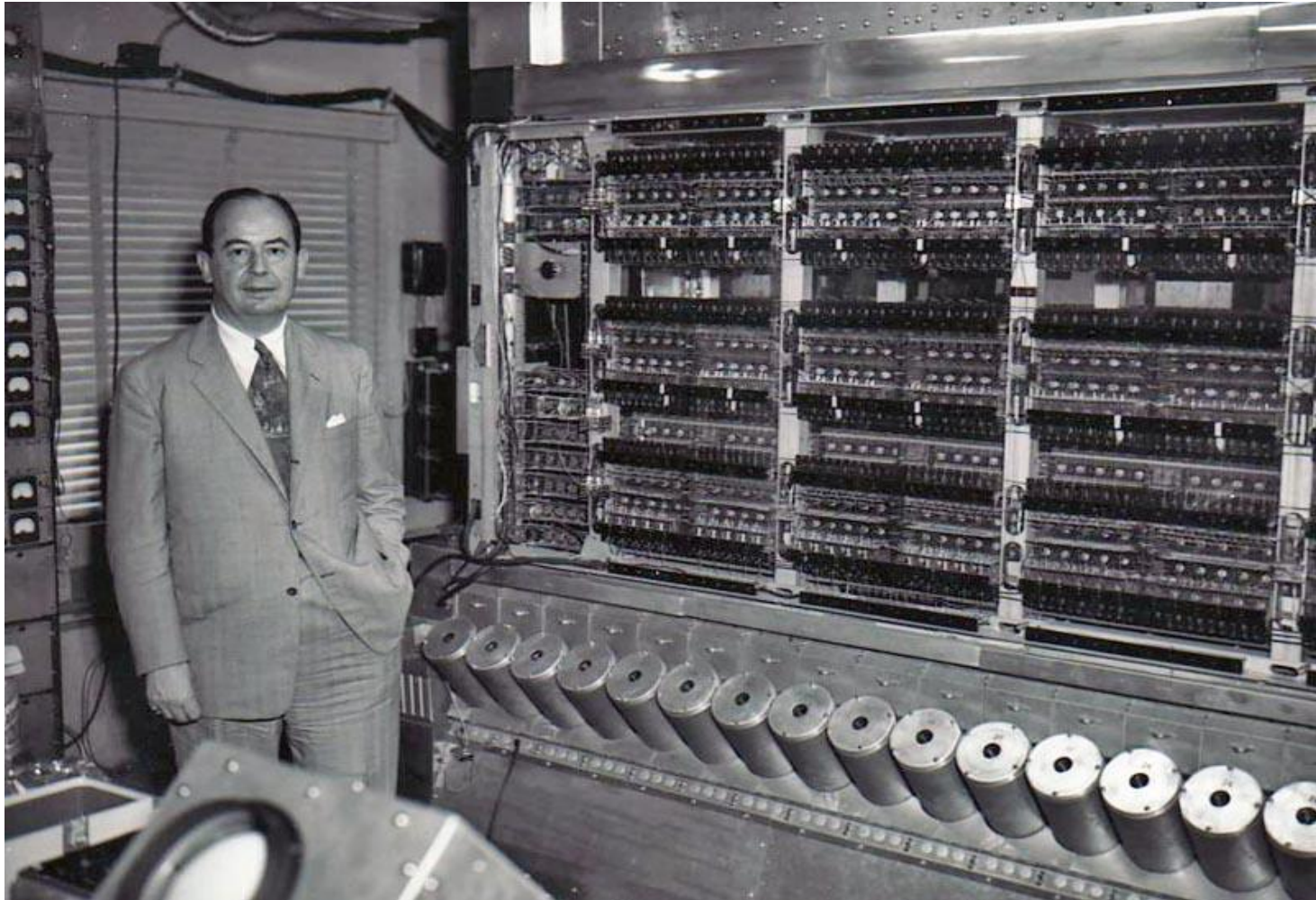
למה ללמוד את מבנה המחשב?

- ▶ אסמבלי עובדת בצורה הדוקה מול החומרה
 - אי אפשר לתכנת בלי להכיר את החומרה
- ▶ הכרת מבנה המחשב מאפשרת:
 - אופטימיזציה של קוד (קיצור זמן ריצה)
 - הקטנת גודל הזיכרון שקוד תופס
 - מציאת באגים בתוכנה
- ▶ בפרק זה נלמד למה מתכוונים כשאומרים:
 - "מחשב 32 ביט / 64 ביט" וכו'

מעבד ה-8086



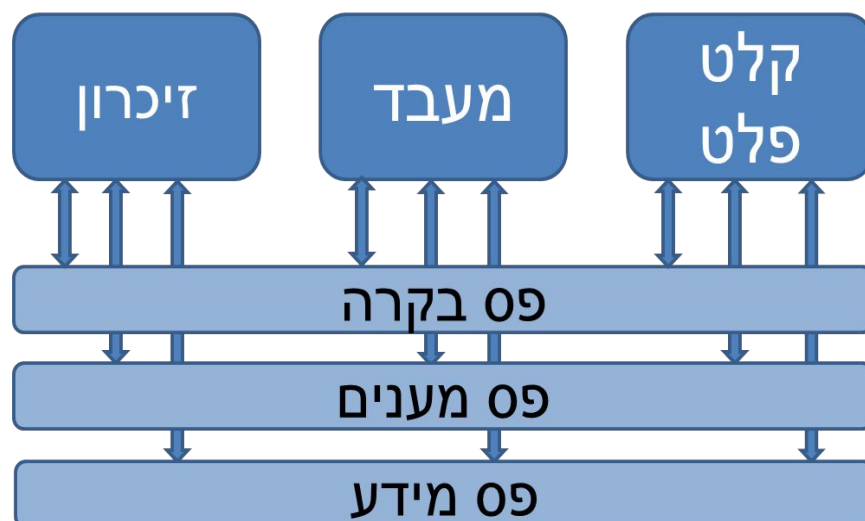
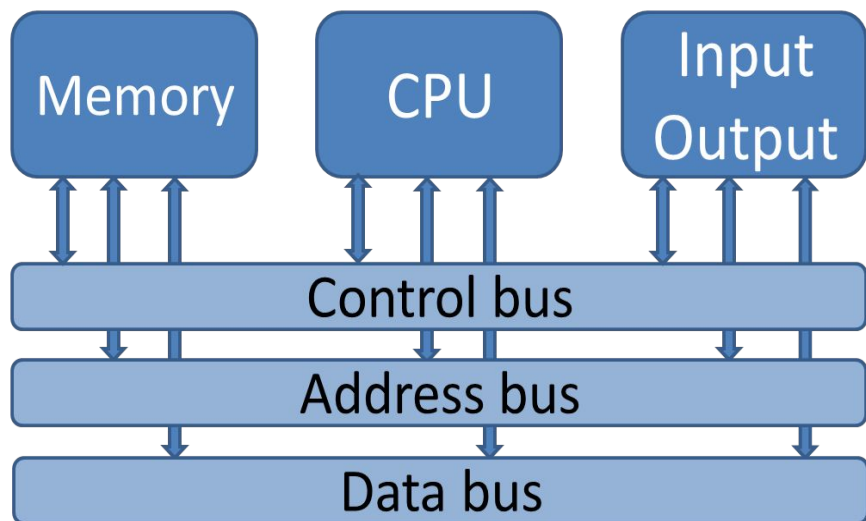
- ▶ תוצרת אינטל
- ▶ שנת 1978
- ▶ הראשון ממשפחת מעבדי ה-80x86
- ▶ בסיס למעבדים הקיימים:
 - תאימות לאחור
 - ארכיטקטורת פון נוימן



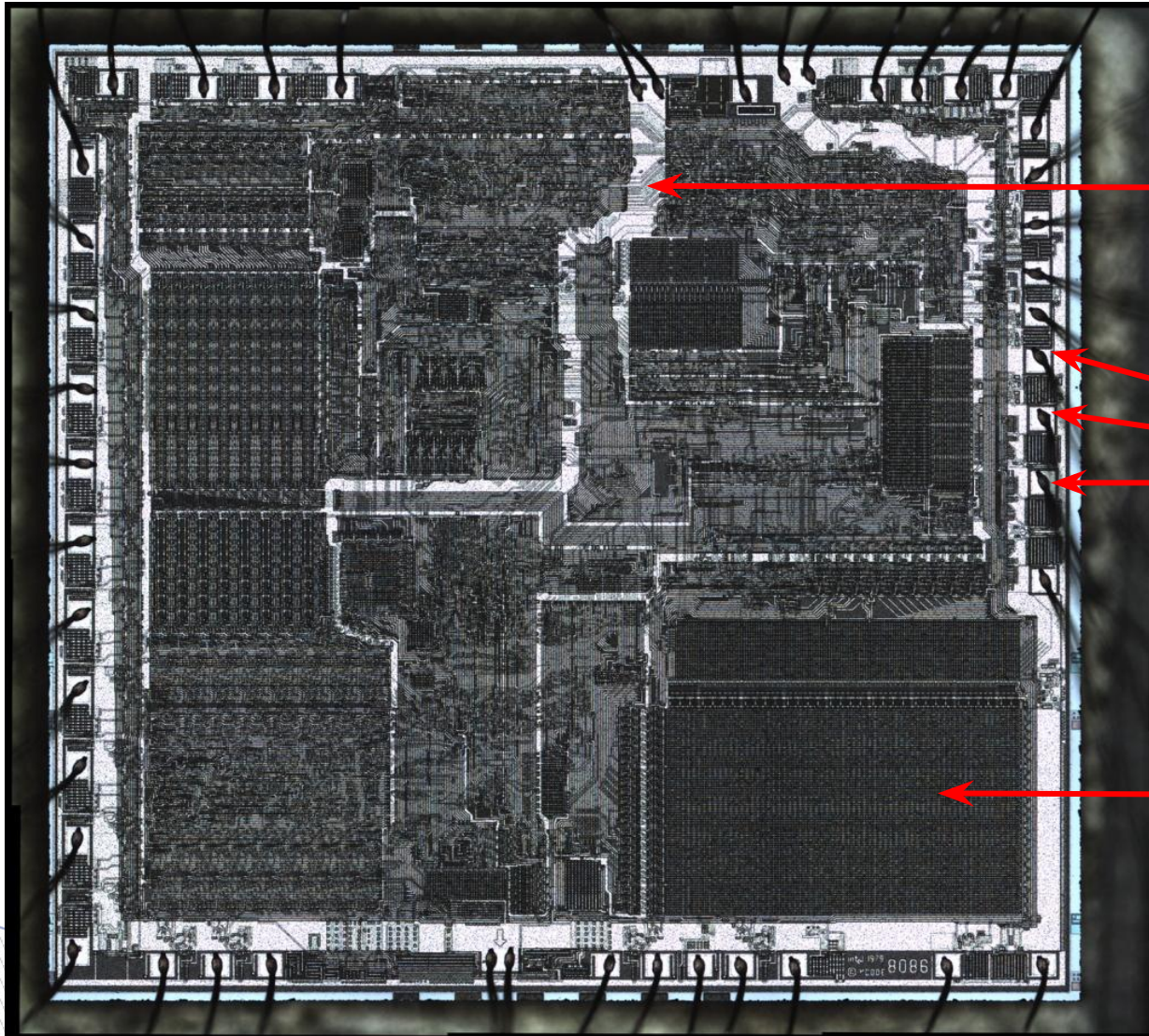
John Von Neumann (1903–1957)

First actual case of bug being found.
1630 antenaut started.
1700 closed down.

ארכיטקטורת פון נוימן



מעבד ה- 8086 (צילום רנטגן)



Bus

I/O

Memory

פסי מערכת System Buses

- ▶ שלושה סוגים של פסי מערכת:
 - פס נתונים Data Bus
 - פס מענים (כתובות) Address Bus
 - פס בקרה Control Bus
- ▶ הפסים מעבירים מתח חשמלי
 - נקבעה רמת מתח שמייצגת '0'
 - נקבעה רמת מתח שמייצגת '1'
- ▶ הקווים מאפשרים לפרש כל ביט
 - המעבד שלח לזיכרון '1000'. למה הכוונה?

פס הנתונים Data Bus

▶ מעתיק נתונים ממקום למקום

▶ גודל הפס משתנה בין מעבדים שונים

▶ פס נתונים של 16 ביט מסוגל להעביר 16 ביטים בהעתקה
יחידה

▶ מעבד עם פס נתונים רחב יוכל לכתוב ולקרוא מהזיכרון מהר
יותר

▶ מי קובע מאיפה לאיפה יועתק המידע?

פס המענים Address Bus

- ▶ לכל בית בזיכרון יש כתובת ייחודית
- ▶ כדי לפנות לזיכרון, פס המענים צריך לקבל את הכתובת המבוקשת
- ▶ כמות הכתובות שניתן לפנות אליהן תלויה ברוחב פס המענים:
 - 2 ביט: הכתובות 00,01,10,11
 - למעבד ה-8086 שני פסי מענים:
 - לזיכרון: פס מענים ברוחב 20 ביט, 1,048,576 כתובות
 - לקלט פלט: פס מענים ברוחב 16 ביט, 65,536 כתובות

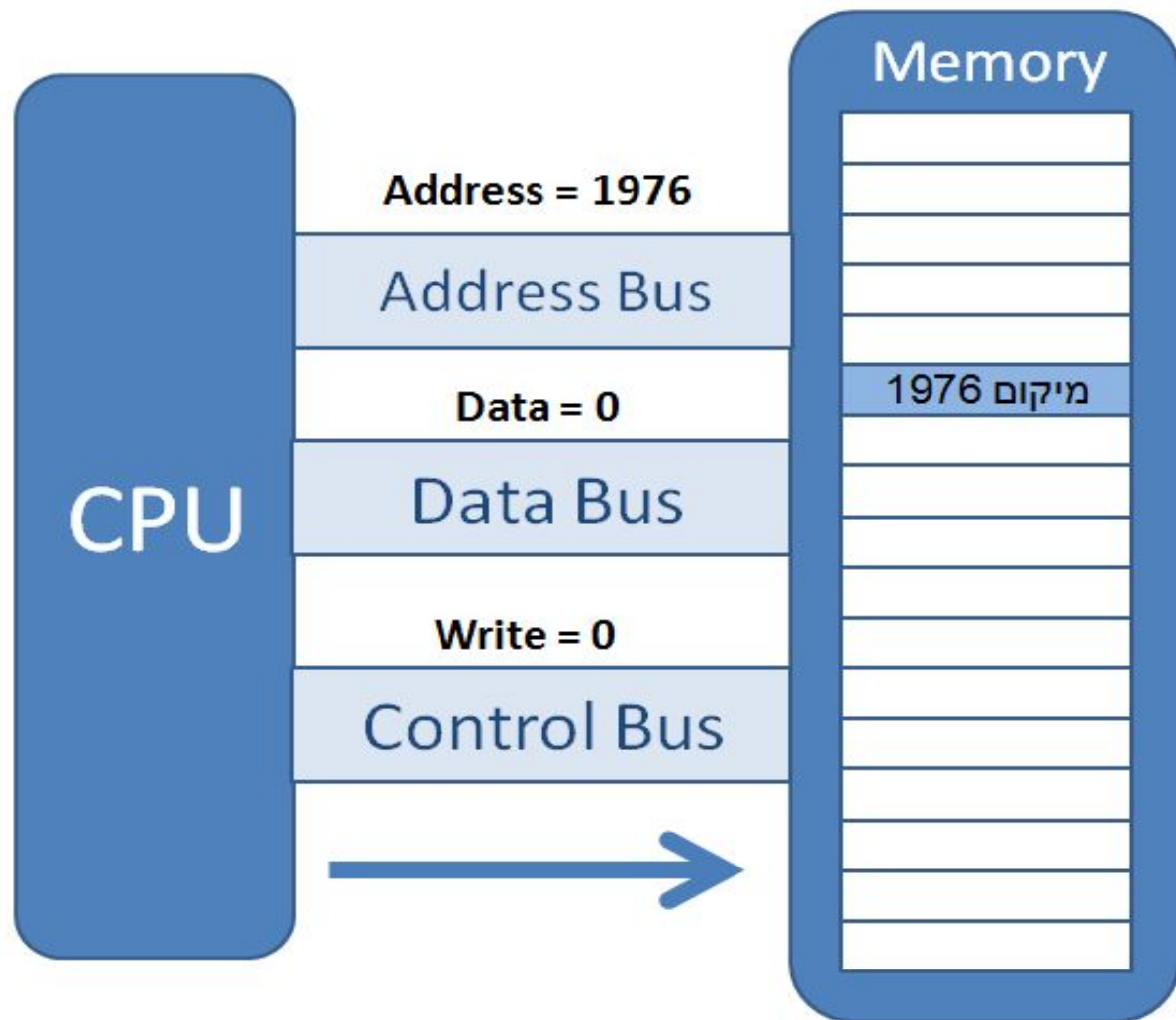
פס הבקרה Control Bus

- ▶ שמנו ערך בפס המענים
 - האם לפנות לזיכרון או ל-I/O?
 - האם לכתוב או לקרוא?
- ▶ פס הבקרה עונה על שאלות אלו
 - לאן לפנות: קו לזיכרון, קו ל-I/O
 - כיוון ההעתקה: קו קריאה read, קו כתיבה write

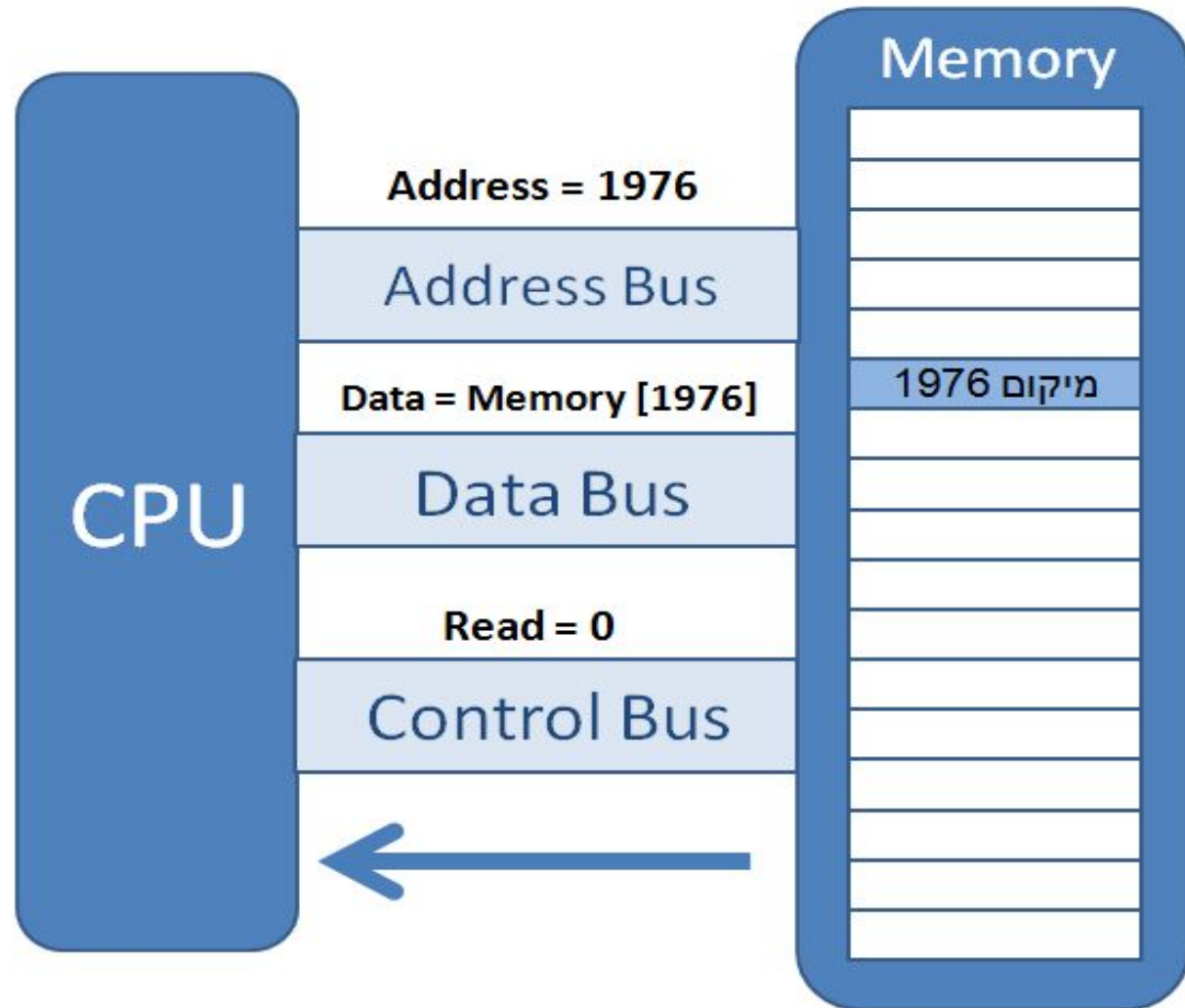
הזיכרון Memory

- ▶ כמות הכתובות בזיכרון מכונה "גודל הזיכרון"
- ▶ כל כתובת = בית אחד
- ▶ פס כתובות בגודל n ביטים מאפשר לפנות לזיכרון בגודל 2^n בתים
- ▶ למעבד ה-8086 זיכרון בגודל 1,048,576 בתים
 - ניתן לחשוב על הזיכרון כמערך $[1,048,575..0]$

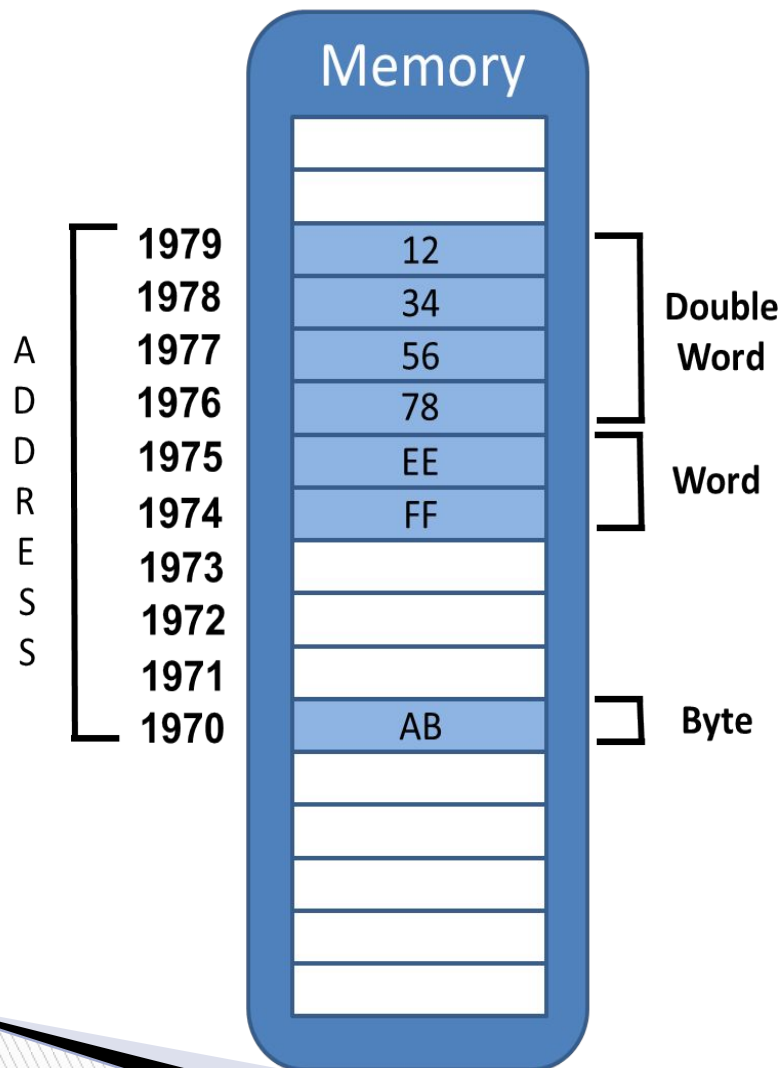
דוגמה – העתקת הערך '0' לזיכרון



דוגמה – העתקה מהזיכרון

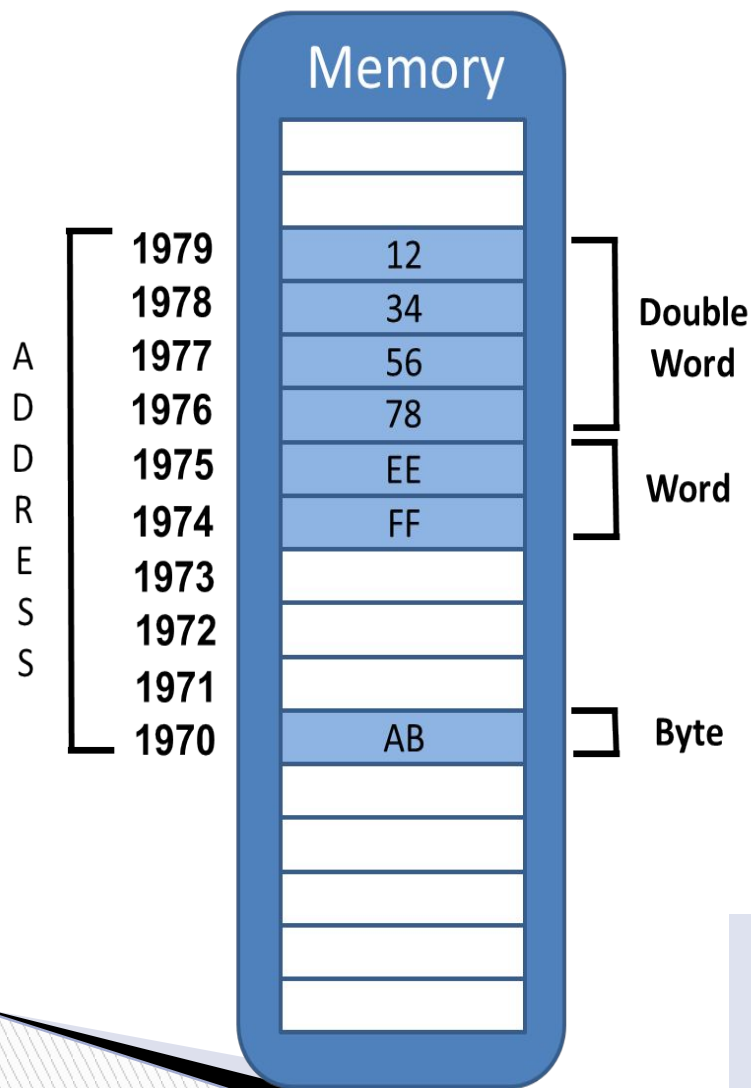


דוגמה – העתקת ערכים לזיכרון



- ▶ לתא שבכתובת 1970 העתקנו את הערך 0ABh
- ▶ לתא שבכתובת 1974 העתקנו את הערך 0EEFFh
- ▶ לתא שבכתובת 1976 העתקנו את הערך 12345678h

דוגמה - קריאה מהזיכרון



קריאת byte מכתובת 1974:

0FFh

קריאת word מכתובת 1975:

78EEh

אפשרי לקרוא word מכתובת 1970?

כן! יתקבל 0ABh ועוד בית עם "זבל"
שנמצא בכתובת 1971

מקטעים והיסטים Segment & Offset

- ▶ למעבד ה-8086 מרחב כתובות בגודל 20 ביטים
- ▶ כדי לגשת לזיכרון, המעבד משתמש ב"מתווך" בגודל 16 ביטים (נלמד בהמשך)
- ▶ צריך שיטה שתאפשר לפנות לכתובת של 20 ביטים ע"י משאבים של 16 ביטים
 - סגמנט: כתובת בגודל 16 ביט
 - אופסט: כתובת בגודל 16 ביט

Segment:Offset

סגמנט ואופסט – המשך

צורת הרישום היא ▶

Segment:Offset

לדוגמה:

0A00:0010h

כדי להגיע לכתובת של 20 ביט, מבצעים את הפעולה הבאה: ▶

$\text{Address} = \text{Segment} \times 16 + \text{Offset}$

0A32:001Bh =

$0A32 \times 16 + 001B =$

$0A320 + 1B = 0A33Bh$

▶ שתלנו רצף מספרים החל מהכתובת 0627:0000h



► קידמנו את האופסט ב-1

DOSBox 0.74, Cpu speed: max 100% cycles, Frameski...

File Edit Search Run Data Options Calls Windows Help

■=[5]===== memory1 b 0x0627:0x0001 =====↑↓

0627:0001	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
0627:000E	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C

► התוצאה- אנחנו מסתכלים על הבית הבא בזיכרון

► החזרנו את האופסט, קידמנו את הסגמנט ב-1

The screenshot shows the DOSBox 0.74 interface with the CPU speed set to max 100% cycles. The menu bar includes File, Edit, Search, Run, Data, Options, Calls, Windows, and Help. The command line shows '[5] memory1 b 0x0628:0x0000'. Below this, a memory editor window is open, displaying two rows of memory addresses and their corresponding hexadecimal and ASCII values. The first row shows addresses 0628:0000 to 0628:000C, and the second row shows 0628:000D to 0628:0009. The address 0628:0000 is highlighted with a red box.

Address	Hex	ASCII
0628:0000	10 11 12 13 14 15 16 17 18 19 1A 1B 1C	▶◀↑!!¶§_±↑↓→←
0628:000D	1D 1E 1F 20 21 22 23 24 25 26 27 28 29	⚡▲▼ !"#\$%&'()

► התוצאה: קפצנו 16 בתים קדימה בזיכרון

הזיכרון – סיכום

- ▶ לכל בית יש כתובת
- ▶ ל-8086 מרחב כתובות של 2^{20} כתובות
- ▶ פונים לכתובת בזיכרון ע"י צירוף של סגמנט ואופסט
- ▶ זמן לתרגול 😊

המעבד Central Processing Unit

▶ רגיסטרים Registers

- רגיסטרים כלליים

- רגיסטרי סגמנט

- רגיסטרים ייעודיים

▶ יחידה אריתמטית לוגית Arithmetic Logic Unit

▶ יחידת בקרה Control Unit

▶ יחידת קלט/פלט I/O Ports – לימוד עצמי

- פרויקט סיום

▶ שערון Timer – לימוד עצמי

- פרויקט סיום

אוגרים Registers

- ▶ רכיבי חומרה שצמודים למעבד
- ▶ משמשים למגוון פעולות
- ▶ המעבד יכול לפנות אליהם בלי להמתין
- ▶ משאב מוגבל
- ▶ כמות וגודל משתנים בין דורות של מעבדים
- ▶ נלמד רגיסטרים של 16 ביט ב-8086



רגיסטרים כלליים General Purpose registers

הרגיסטר	שם לועזי	שם עברי	תיאור ושימוש עיקרי
AX	Accumulator register	צובר	משמש לרוב הפעולות האריתמטיות והלוגיות. למרות שניתן לבצע פעולות חישוב גם בעזרת רגיסטרים אחרים, השימוש ב-AX הוא בדרך כלל יעיל יותר.
BX	Base address register	בסיס	בעל חשיבות מיוחדת בגישה לזיכרון. בדרך כלל משמש לשמירת כתובות בזיכרון.
CX	Count register	מונה	מונה דברים. בדרך כלל נשתמש בו לספירת כמות הפעמים שהרצנו לולאה, לכמות התווים בקובץ או במחרוזת.
DX	Data register	מידע	משמש לשתי פעולות מיוחדות: ראשית, ישנן פעולות אריתמטיות שדורשות מיקום נוסף לשמירת התוצאה. שנית, כשפונים להתקני I/O, רגיסטר DX שומר את הכתובת אליה צריך לפנות.

רגיסטרים כלליים General Purpose registers

הרגיסטר	שם לועזי	שם עברי	תיאור ושימוש עיקרי
SI	Source Index	מצביע מקור	ניתן להשתמש בהם בתור מצביעים כדי לפנות לזיכרון (כמו שהראינו שניתן לעשות עם BX). כמו כן הם שימושיים בטיפול במחרוזות.
DI	Destination Index	מצביע יעד	
BP	Base Pointer	מצביע בסיס	משמש לגישה לזיכרון בסגמנט שקרוי "מחסנית" Stack.
SP	Stack Pointer	מצביע מחסנית	בעל תפקיד מיוחד מאד. SP שומר את מיקום המחסנית. באופן נורמלי, לעולם לא ניגע ב-SP ולא נעשה בו שימוש לטובת ביצוע פעולות מתמטיות. התפקוד התקין של התכנית שלנו תלוי בכך שערכו של SP תמיד יצביע למיקום הנכון בתוך המחסנית.

חלוקה לרגיסטרים בני 8 ביט

- ▶ הרגיסטרים AX, BX, CX, DX הם בגודל 16 ביט
- ▶ כדי לאפשר גמישות לטיפול בבתים בודדים, יש חלוקה משנית לרגיסטרים בגודל 8 ביט

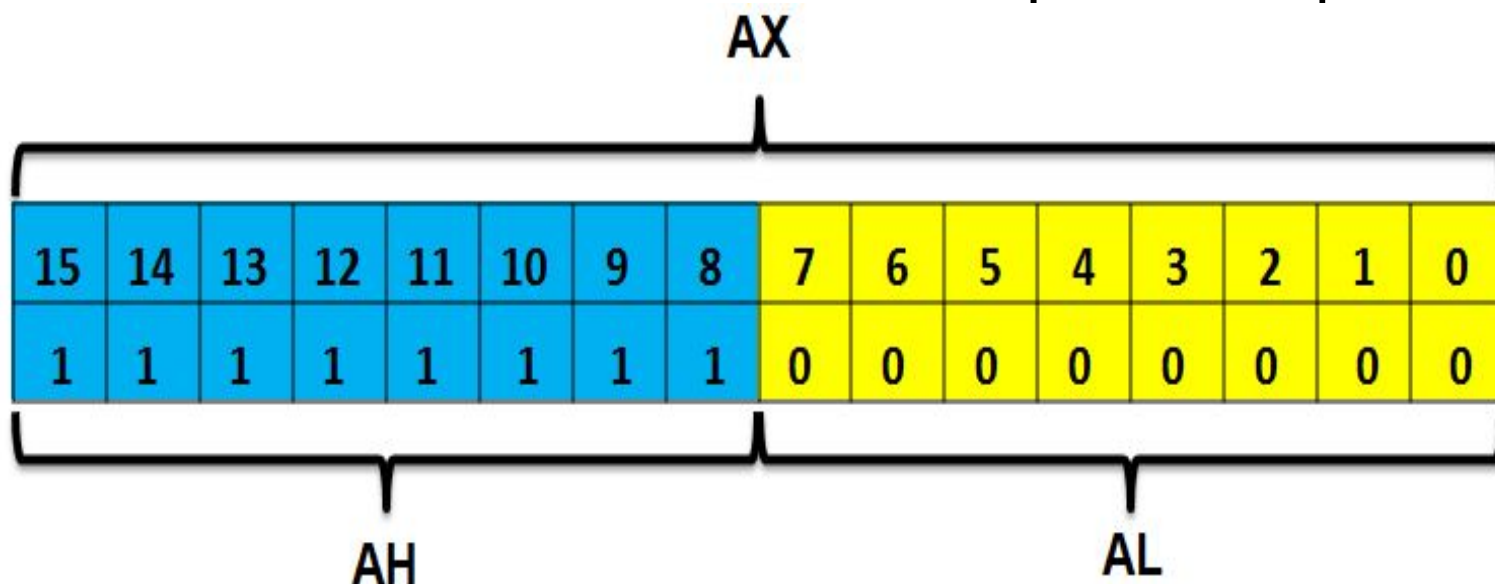
bit 16	bit 8	bit 8
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

דוגמה – רגיסטרים

▶ נניח שנעתיק לתוך AX את הערך 0FF00h:

◦ AH יקבל את הערך 0FFh

◦ AL יקבל את הערך 00h



דוגמה - רגיסטרים

The screenshot shows the DOSBox 0.74 interface. The main window displays assembly code from a file named 'empty.asm'. The code is as follows:

```

12:
13: Main      proc
14:           mov     ax, seg dseg
15:           mov     ds, ax
16:
17:           mov     ax, 1234h
18:           mov     bx, 0
19:           mov     bl, 34h
20:           mov     cx, 0
21:           mov     ch, 12h
22:

```

On the right side, the register window shows the following values:

Register	Value
AX	1234
BX	0034
CX	1200
DX	0000
SP	2000
BP	0000
SI	0000
DI	0000
DS	07E7
ES	05D7
SS	05E7
CS	07E7
IP	0012
FL	0202

At the bottom, the command window shows a warning: 'Warning: TOOLS.INI not found'.

תרגילים – רגיסטרים

DX		CX		BX		AX		הפעולה	
DH	DL	CH	CL	BH	BL	AH	AL	לרגיסטר	העתק את
								AX	הערך 1234h
								AL	הערך ABh
								BX	הערך ABCDh
								CH	הערך EEh
								DL	הערך BBh
								DH	הרגיסטר CH
								AH	הרגיסטר DL
								CX	הרגיסטר BX

תרגיל רגיסטרים – first.asm

- ▶ תרגמו את הפקודות שבתרגיל לפקודות אסמבלי
- ▶ שלבו את הפקודות בקובץ base.asm. שימרו את הקובץ החדש בשם first.asm.
- ▶ הריצו את התוכנית ובידקו את השינויים ברגיסטרים תוך כדי ריצה.
- ▶ עזרה: פקודת ההעתקה נקראת mov. לידה נרשום שני רגיסטרים – את הימני מעתיקים לשמאלי. לדוגמה העתקה של dx לתוך ax:

```
mov ax, dx
```

אוגרי מקטע – Segment Registers

▶ ארבעה רגיסטרים שמשמשים לשמירת כתובות סגמנטים:

- ▶ CS – Code Segment
- ▶ DS – Data Segment
- ▶ SS – Stack Segment
- ▶ ES – Extra Segment

▶ כזכור כל סגמנט יכול להיות בגודל עד 64KByte

▶ ניתן להגדיר מספר סגמנטי קוד ונתונים ולשנות את CS ו-DS כך שיצביעו בכל פעם על סגמנט אחר

אוגרים ייעודיים – Special Purpose Registers

- ▶ IP– Instruction Pointer
- ▶ FLAGS– דגלים

▶ הכרות עם IP חשובה ל:

- פרוצדורות

- פסיקות

▶ הכרות עם FLAGS חשובה ל:

- תנאים לוגיים

- לולאות

Arithmetic & Logical Unit

▶ ה-ALU הוא המקום בו מתבצעות פעולות החישוב

▶ לדוגמה `add ax, 3`:

- המעבד יעתיק את הערך שב-AX לתוך ה-ALU
- המעבד ישלח ל-ALU את הערך 3
- המעבד ייתן ל-ALU הוראה לחבר את שני הערכים
- המעבד יחזיר את תוצאת החישוב מה-ALU ל-AX

יחידת הבקרה – Control Unit

- ▶ איך המעבד יודע אילו פקודות לבצע?
- ▶ יחידת הבקרה מחזיקה את הרגיסטר IP
 - IP מצביע על הפקודה הבאה אותה יש לבצע
- ▶ יחידת הבקרה קוראת את הפקודות מהזיכרון
 - Opcodes – Operational Codes
- ▶ יחידת הבקרה מעתיקה את ה-Opcode אל רגיסטר פענוח
- ▶ יחידת הבקרה מקדמת את IP אל הפקודה הבאה

שלבי ביצוע התכנית : הבאה, פענוח וביצוע

הוראה

זיכרון

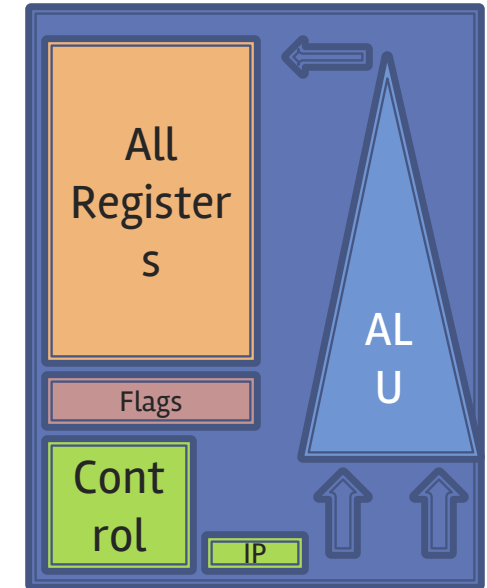
xor bx, bx
mov ax, 0D903h
inc ah
mov [bx], ax

33
DB
B8
03
D9
FE
C4
89
07
03
DA
...

Code segment
9 bytes



Data segment
לאחר ביצוע ההוראה רביעית



לאחר ביצוע כל פקודה, IP מתקדם אל הפקודה הבאה

- 1) xor bx, bx 33DB (IP+2)
- 2) mov ax, 0D903h B803D9 (IP+3)
- 3) inc ah FEC4 (IP+2)
- 4) mov [bx], ax 8907 (IP+2)

- ▶ מיצאו את ה-OpCode של הפקודה `mov ax, dx`
 - בלי לבצע אסמבלי לפקודה הנ"ל! 😊
 - הדרכה: התבוננו בפקודות אחרות, דומות, וחיקרו את הדרך שבה הן מיתרגמות לפקודות מכונה. הוסיפו פקודות נוספות לפי הצורך!
 - טיפ: מיצאו איך כל רגיסטר מיתרגם לרצף ביטים

```
mov    ax, 1
mov    cx, bx
mov    dx, [100]
```

▶ מבנה המחשב – ארכיטקטורת פון נוימן

▶ פסים:

◦ נתונים

◦ מענים (כתובות)

◦ בקרה

▶ ארגון הזיכרון – סגמנט ואופסט

▶ רגיסטרים