משרד החינוך
האגף לתכנון ולפיתוח
תכניות לימודים

המרכז הישראלי לחינוך
מדעי-טכנולוגי
ע"ש עמוס דה-שליט

המרכז לטכנולוגיה
חינוכית (מט"ח)

האוניברסיטה הפתוחה
בית הספר לטכנולוגיה

מבוא למערכות מחשב ואסמבלי

אישור משרד החינוך
אישור מס' נ/4127

כתיבה

שרה פולק

ייעוץ אקדמי

ד״ר דן אהרוני

ייעוץ דידקטי

ד״ר צבי פירסט

קריאה והערות

ד״ר ראובן חוטובלי

דינה קראוס

סופי גילליס

ישראל זילברשטיין

עריכה לשונית

אילנה גולן

ד״ר דן אהרוני (פרקים 1-5)

איורים

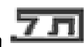
רונית בורלא

הפקה

אביבה אבידן

עיצוב עטיפה

אבי חתם



משרד החינוך
האגף לתכנון ולפיתוח
תכניות לימודים

המרכז הישראלי
לחינוך מדעי-טכנולוגי
ע״ש עמוס דה-שליט

המרכז לטכנולוגיה
חינוכית (מטח)

האוניברסיטה הפתוחה
בית-הספר לטכנולוגיה

מקט 1043311

מהדורת ניסוי

© מהדורת תשס״ז – 2006. כל הזכויות שמורות למשרד החינוך.
בית ההוצאה לאור של המרכז לטכנולוגיה חינוכית, קרית משה רואו, רח׳ קלאוזנר 16, רמת-אביב, ת״ד 39513, תל-אביב 61394.
The Centre For Educational Technology, 16 Klausner St., Ramat-Aviv, P.O.Box 39513, Tel-Aviv, 61394. Printed in Israel.

זכויות הקניין הרוחני, לרבות זכויות היוצרים והזכות המוסרית של היוצר/ים בספר זה מוגנות. אין לשכפל, להעתיק, לסכם, לצלם, להקליט, לתרגם, לאחסן במאגר מידע, לשדר או לקלוט בכל דרך או בכל אמצעי אלקטרוני, אופטי, מכני או אחר, כל חלק שהוא מספר זה. כמו כן, אין לעשות שימוש מסחרי כלשהו בספר זה, בכולו או בחלקים ממנו, אלא אך ורק לאחר קבלת רשות מפורשת בכתב ממטח (המרכז לטכנולוגיה חינוכית).

תוכן העניינים

פרק 1 המחשב הדיגיטלי 7

1.1 מבוא 7

1.2 מודל מופשט של מחשב 11

1.3 מבנה מחשב פשוט ואופן ביצוע ההוראות בשפת מכונה 14

1.4 הקשר בין יחידות המחשב 42

פרק 2 ייצוג מידע במחשב 47

2.1 מבוא 47

2.2 ייצוג מספרים שלמים 48

2.3 ייצוג מספרים ממשיים 64

2.4 ייצוג טקסט 63

2.5 ייצוג תמונה 74

2.6 יחידות זיכרון לאחסון מידע במחשב 77

נספח – ייצוג כללי של מספרים בשיטה מקומית 84

פרק 3 פעולות אריתמטיות על ייצוג בינארי במחשב 85

3.1 חיבור וחיסור מספרים בינאריים בלתי מכוונים 85

3.2 שיטות לייצוג מספרים בינאריים מכוונים 97

3.3 חיבור וחיסור מספרים בינאריים שלמים מכוונים 103

3.4 תחום הייצוג של מספרים בינאריים שלמים מכוונים ובלתי מכוונים 107

פרק 4 שפת אסמבלי והמודל התכנותי של מעבד 8086 109

4.1 מבוא 109

4.2 המודל התכנותי של ה-8086 110

4.3 ארגון הזיכרון במעבד 8086 115

4.4 כתיבת תכנית בשפת אסמבלי 123

4.5 הגדרת משתנים בשפת אסמבלי 131

4.6 הצהרה על קבועים – הנחיית אסמבלר EQU 138

נספח – תיאור תהליך הרצת התכנית 139

פרק 5 תכנות בסיסי בשפת אסמבלי 141

5.1 מבוא 141

5.2 הוראות להעברת נתונים 141

5.3 הוראות אריתמטיות – חיבור וחיסור 144

5.4 הוראות בקרה 163

5.5 הוראות כפל וחילוק 182

5.6 הוראות לוגיות 188

5.7 הוראות הזזה וסיבוב 202

פרק 6 שיטות מיעון, מערכים ורשומות 215

6.1 הצהרה על מערכים ורשומות 215

6.2 שיטות מיעון 221

6.3 מיעון מיידי (Immediate addressing) 224

6.4 מיעון אוגר (Register addressing) 225

6.5 מיעון ישיר (Direct addressing mode) 226

6.6 מיעון עקיף בעזרת אוגר (Indirect Addressing Register) 229

6.7 מיעון אינדקס (Direct Indexed Mode) 235

6.8 מיעון בסיס (Base Relative Addressing) 240

6.9 מיעון אינדקס-בסיס (Based Indexed Addressing Modes) 243

פרק 7 מחסנית, שגרות ומקרו 249

7.1 מבוא 249

7.2 כתיבת פרוצדורה וזימונה 250

7.3 המחסנית ומצביע המחסנית 253

7.4 העברת פרמטרים 264

7.5 מימוש משתנים מקומיים 271

7.6 העברת מערך כפרמטר לפרוצדורה 277

7.7 מימוש פונקציות 281

7.8 פונקציה רקורסיבית 284

7.9 מקרו 302

פרק 8 עיבוד מחרוזות ובלוקים של נתונים 311

8.1 הגדרת מחרוזות בשפת אסמבלי 311

8.2 מבנה של הוראות מחרוזת 313

8.3 העתקת מחרוזות – ההוראה MOVS 313

8.4 חזרה על פעולת ההעתקה 316

8.5 כתיבת תווים במחרוזת – ההוראה STOS (Store a String) 318

8.6 קריאת תו ממחרוזת – ההוראה LODS (Load a String) 319

8.7 השוואת מחרוזות ההוראה CMPS (CoMPare String) 321

8.8 חיפוש תו במחרוזת הוראה SCAS (SCAn String) 325

8.9 טבלאות תרגום וההוראה XLAT 327

פרק 9 פסיקות וקלט-פלט 329

9.1 מבוא 329

9.2 שימוש בשגרת השירות של DOS 329

9.3 מנגנון ביצוע פסיקות במעבד 8086 338

9.4 קריאה ושינוי של פסיקה 346

9.5 הוראות IN ו-OUT 350

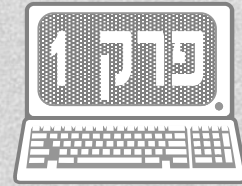
פרק 10 ארכיטקטורה של מעבדים מתקדמים 353

10.1 ההשפעה של ההתפתחות הטכנולוגית על מבנה מעבדים מתקדמים 353

10.2 מבנה האוגרים במעבדים מתקדמים 356

10.3 ארכיטקטורת "צינור הוראות" (pipelining) 358

10.4 ארגון זיכרון 375



המחשב הדיגיטלי

1.1 מבוא

מחשב דיגיטלי הוא מכונה שבאמצעותה ניתן לפתור בעיות שונות. **התוכנה** (Software), היא אוסף של הוראות ונתונים, אשר מגדירים את אופן הפעולה של המחשב. אוסף זה יכול להופיע כתכנית מחשב אחת, או כמספר תוכניות. ניתן לסווג את התכניות לשני סוגים:

1. תכניות, הנקראות "יישומים", שנועדו לפתור בעיות או לבצע משימות ספציפיות עבור משתמש, כגון: מעבד תמלילים, תוכנה לניהול ספריית וידאו, משחק מחשב וכדומה.
2. אוסף תכניות שמספקות פונקציות שירות כלליות למשתמש במחשב, כגון: מערכת הפעלה (Windows, Linux, וכדומה) הכוללת פונקציות להפעלת המחשב, תקשורת, שפות תכנות וכדומה;

המחשב מורכב מרכיבים רבים, החל מההתקנים שהמחשב חייב להכיל – **המעבד הראשי**, שהוא "לב" המחשב, הזיכרון איתו הוא עובד, ועוד, וכלה בהתקנים המכונים "היקפיים" (חיצוניים) כמו: מקלדת, עכבר, דיסק קשיח, צורב דיסקים, ועוד. הרכיבים הללו מורכבים מאלפי מעגלים אלקטרוניים ורכיבים מכאניים, ולכל המכלול הפיזי קוראים **חומרת המחשב** (hardware).

בין החומרה לתוכנה קיימת תלות הדדית: מחד, התוכנה מגדירה מה המחשב צריך לבצע, וכך היא מפעילה את החומרה. מאידך, החומרה מגדירה את סוג ההוראות שהמחשב יכול לבצע ואת אופן הפעולה שלו. ניתן לדמות זאת לשימוש במכונית: באמצעות ההגה, מוט ההילוכים הידני (או האוטומטי) והדוושות אנו קובעים את התנהגות המכונית, כלומר את המהירות, כיוון הנסיעה וכדומה. מצד שני, מבנה המכונית, סוג המנוע ושאר החלקים מגדירים את הפעולות שהנהג יוכל לבצע, כמו: המהירות המרבית שהמכונית תיסע בה, או התאוצה שתפתח.

התכניות מורצות במחשב ב**שפת מכונה** (machine language) ; בשפה זו ההוראות מיוצגות בקודים מספריים המורכבים מהספרות 0 ו-1 בלבד. שפת מכונה (לעיתים מכנים את המחשב בשם "מכונה") היא השפה שבה קל להפעיל את חומרת המחשב. החומרה מורכבת מאוסף של מעגלים אלקטרוניים. מעגלים אלה פועלים על-פי אותות חשמליים, המוגדרים על-ידי הסימנים 0 ו-1. אולם השימוש בשפת המכונה לכתיבת תכניות אינו נוח למתכנתים, והתכניות קשות להבנה. לכן פותחה **שפת אסמבלי** (Assembly language), הנקראת בעברית "שפת-סף". בשפת אסמבלי הוחלפו הקודים המספריים בקודים הנקראים "קודים מנמוניים" (Mnemonic codes)[*]. כל **קוד מנמוני** נכתב באותיות לטיניות, והוא מורכב ממילה או מקיצור של מילה, המייצגים פעולה שיש לבצע. לדוגמה : קוד הפעולה של הוראת חיבור הוא ADD (קיצור של המילה ADDition) וקוד הפעולה של הוראת השוואה הוא CMP (קיצור של המילה CoMPare). כדי להריץ תכנית בשפת אסמבלי, יש לתרגם אותה תחילה לשפת מכונה. התרגום לשפת מכונה מתבצע באמצעות תכנית מחשב הנקראת "**אסמבלר**" (Assembler). האסמבלר מתרגם כל הוראה בשפת אסמבלי להוראה בשפת מכונה. ההוראות בשפת מכונה ובשפת אסמבלי מתייחסות במישרין לחומרת המחשב, לכן נהוג לקרוא להן "**שפות תכנות נמוכות**" (Low level languages).

כיוון שתכניות בשפה נמוכה תלויות בחומרה, לא ניתן להריץ תכנית שנכתבה במחשב מסוים על מחשב אחר, אם החומרה שלו שונה. לדוגמה : תכניות שפותחו עבור מחשבים שיצרה חברת IBM לא יוכלו לרוץ על מחשבים שיצרה חברת אינטל. בעיה זו הייתה אחד הגורמים העיקריים להתפתחות נוספת בשפות התכנות, ולהמצאת השפות העיליות (High Level Languages) כמו פסקל, C, JAVA, C++, Visual Basic ועוד.

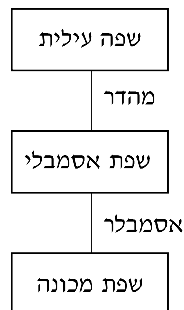
שפות עיליות הן שפות מופשטות, והן כוללות הוראות המתארות את הפעולות שרוצים לבצע, בלי להתייחס לאופן המימוש שלהן בחומרת המחשב. לדוגמה : בהוראת ההשמה A=9, אין אנו רושמים את המיקום הפיזי של המשתנה A בזיכרון המחשב, מפני שקיים מנגנון שידאג לאחסן את המספר 9 בזיכרון, בַּמקום שיוקדש באופן אוטומטי למשתנה אותו אנו מכנים "A".

---

[*] המילה "מנמוני" או "מנמוטכני" היא מילה לטינית, שפירושה : "מסייע לזכירה".

כדי שהמחשב יוכל להבין תכנית שנכתבה בשפה עילית, יש לתרגם אותה לתכנית הכתובה בשפת מכונה. תהליך זה נקרא "**הידור**" (Compilation). ההידור מתבצע על-ידי תכנית הנקראת "**מהדר**" (Compiler). המהדר מתרגם תכנית משפה עילית, ויוצר קובץ המכיל תכנית בשפת מכונה – בהתאם לסוג המחשב עליו מריצים את התכנית.

קיימות שפות עיליות שבהן התרגום לשפת מכונה מתבצע על-ידי תהליך שנקרא "**פירוש**" (Interpretation). בתהליך זה מפוענחת ומבוצעת כל הוראה בנפרד ובמקרה כזה לא נוצר קובץ בשפת מכונה. לביצוע הפירוש אנו משתמשים בתכנית שנקראת "**מפרש**" (Interpreter).



איור 1.1
תהליך תרגום תכנית לשפת מכונה

שאלה 1.1

נכתבו שתי תכניות: האחת בשפת פסקל והשנייה בשפת C. שתי התכניות אמורות לרוץ על מחשב של חברת אפל ועל מחשב של חברת אינטל. מהו מספר המהדרים ומספר שפות המכונה הדרושים כדי שהתכניות תוכלנה לרוץ על שני סוגי המחשבים?

כדוגמה להוראות בשפה עילית והוראות בשפה נמוכה נכתוב הוראות לביצוע הפעולה המתמטית:

result = count1 + count2 + count3

בשפה עילית, פעולה זו נכתבת כהוראה אחת. לדוגמה, בשפת פסקל נרשום את ההוראה

```
result: = count1 + count2 + count3;
```

ובאופן דומה, בשפת C נרשום:

```
result = count1 + count2 + count3;
```

כדי לרשום הוראות מתאימות בשפת אסמבלי, עלינו לדעת מהו המחשב עליו תתבצע התכנית. בספר זה נשתמש בשפת אסמבלי המתאימה למעבדי אינטל המותקנים, בין השאר, גם במחשבים אישיים (מחשב אישי – **P**ersonal **C**omputer – PC). המעבד הוא החלק העיקרי במחשב שבו מורצת התכנית; בהמשך נלמד עליו בהרחבה. נציג לכם כעת סדרת הוראות שאפשר לרשום בשפת אסמבלי למעבד אינטל. הסתכלו בעמודה השמאלית בטבלה 1.1.

טבלה 1.1
קטע תכנית בשפת אסמבלי ובשפת מכונה

הסבר ההוראה	ההוראה בשפת מכונה	הוראה בשפת אסמבלי
שים את count1 ב-AX	101000010000000000000	mov AX, count1
חשב AX+count2, ושים את התוצאה ב-AX	00000011000001100000001000000000	add AX, count2
חשב AX+count3, ושים את התוצאה ב-AX	00000011000001100000010000000000	add AX, count3
שים את AX ב-result	000000110000011000000000	mov result, AX

ההוראה MOV היא הוראת השמה, וההוראה ADD היא הוראת חיבור AX המוזכר בהוראות אלו הוא סוג של רכיב חומרה המכונה "אוגר", עליו נלמד בהמשך, ובו ניתן לאחסן נתונים המשתתפים בפעולות אלו. העמודה השנייה של טבלה 1.1, מציגה את התרגום של כל אחת מן ההוראות בשפת אסמבלי להוראה בשפת מכונה.

במדעי המחשב קיימים כמה תחומים בהם חשובה ההיכרות עם שפה נמוכה. אחד התחומים הוא פיתוח מהדר (קומפיילר) המתרגם תכנית משפה עילית לתכנית בשפת מכונה. מפתח של קומפיילר צריך להכיר בצורה מעמיקה את שפת המכונה ומבנה המחשב, כך

שהתכנית המתורגמת על-ידי הקומפיילר שהוא מפתח, תנצל בצורה יעילה את המשאבים שהמחשב מספק.

גם מפתחים של מערכות הפעלה משתמשים בשפת אסמבלי בחלק מן התכניות. **מערכת הפעלה** היא תוכנה המתווכת בין משתמש מחשב (משתמש ביישום או מתכנת) לבין החומרה, והיא מספקת ממשק בעזרתו יכול המשתמש להפעיל יישומים שונים. מערכת ההפעלה מטפלת בהרצה של כמה תכניות במקביל; היא אחראית לחלוקת הזיכרון בין כל התכניות המורצות במחשב באותו זמן, והיא קובעת מתי ואיזו תכנית תתבצע במעבד בכל רגע ורגע. בנוסף, מערכת ההפעלה מספקת שירותים שונים, בהם יכולים מתכנתים בשפה עילית להשתמש, כדי לטפל בקלט המועבר מן המקלדת או מן העכבר, להציג פלט על הצג וכדומה. דוגמאות למערכות הפעלה נפוצות: Windows, Linux וכדומה.

תחום נוסף הוא **מערכות משובצות מחשב** (Embedded System) בהן משתמשים בבקרים זעירים, המשובצים במכשירים שונים, כמו: כלי-רכב, טלוויזיה, או מיקרו גל. תפקיד הבקר הזעיר הוא לפקח על פעולת המכשיר. הבקר הזעיר ניתן לתכנות והמבנה שלו דומה למבנה של מחשב, אבל הוא לא כולל התקנים היקפיים, כמו: מקלדת, צג, או דיסק קשיח. התקשורת בין הבקר לחלקי המערכת האחרים מתבצעת באמצעות אותות חשמליים, המועברים כקלט מהמכשיר לבקר הזעיר, אשר מוציא אותות חשמליים ששולטים על פעולת המכשיר, בהתאם לתכנית השמורה בו.

כמו-כן, כל מתכנת בשפה עילית צריך להבין איך המחשב פועל ואיך החומרה משפיעה על ההתנהגות של התכנית בשפה העילית בה הוא כותב, וכך הוא יוכל לכתוב תכניות המנצלות את המכונה באופן יעיל יותר. לכן התנסות בכתיבה תכניות בשפת אסמבלי, שבה יש התייחסות לחומרה, מאפשרת להבין את עקרונות הפעולה והמבנה של מחשב.

1.2 מודל מופשט של מחשב

המחשב הוא מערכת מורכבת מאוד המיועדת למשתמשים רבים, שלכל אחד מהם צרכים מגוונים. באיור 1.2 מוצג מודל המתאר רמות שונות של הפשטה של המחשב. בכל רמה מתייחסים לתיאור מופשט (אבסטרקטי) של הפעולות המתבצעות באותה רמה, בלי להיכנס

לפרטים כיצד ימומשו פעולות אלה, מפני שקיימת רמה נמוכה יותר שתפקידה לטפל במימוש הפעולות.

ברמה העליונה, רמה L6, המשתמש ביישומי מחשב (כמו משחקים, מעבד תמלילים, תוכנה לניהול משרד) מתקשר עם המחשב באמצעות ממשק משתמש הכולל בדרך-כלל תפריטים, כפתורים וצלמיות. המשתמש מתעניין בפעולות שהיישום מאפשר לו והוא מניח כי קיימת שכבה נמוכה יותר, רמה L5, שבאמצעותה יוגדרו הפעולות של היישום. להגדרת הפעולות שברמה L5, המתכנת משתמש בהוראות בשפה עילית כלשהי, והוא מניח שקיים, בשכבה נמוכה יותר, מנגנון כלשהו שתפקידו להוציא לפועל את ההוראות ולבצען.

המנגנונים שקיימים ברמה נמוכה יותר, רמה L4, התומכים בתכנית ומאפשרים את הרצתה, הם מערכת ההפעלה ומהדר המתאים לשפה שבה הוא נכתב, שתפקידו לתרגם את התכנית לשפת מכונה. חלקים רבים ממערכת ההפעלה כתובים בשפת עילית, אך בנוסף יש חלקים הכתובים בשפת אסמבלי, שנמצאת ברמה L3. באופן דומה, מתכנת בשפת אסמבלי מניח שברמה נמוכה יותר, רמה L2, קיימת שפת מכונה אשר ההוראות בה מקודדות כמספרים בינאריים.

ההוראות בשפת אסמבלי ובשפת מכונה מתייחסות לארכיטקטורה של המחשב, המוגדרת ברמה L1. הארכיטקטורה מגדירה מודל מופשט של מבנה המחשב ושל הוראות המכונה; הארכיטקטורה ממומשת באמצעות מעגלים ורכיבים אלקטרוניים (מערכת פיזית) הנמצאים ברמה הנמוכה ביותר, רמה L0.

רמה		
L6	יישום	
L5	שפה עילית	
L4	מערכת הפעלה	מהדר
L3	שפת אסמבלי	
L2	שפות מכונה	
L1	ארכיטקטורה של מחשב	
L0	מעגלים ורכיבים אלקטרוניים	

איור 1.2
רמות ההפשטה במחשב

בספר זה נתמקד בשתי רמות: שפת אסמבלי והארכיטקטורה של המחשב. מנקודת מבטו של המתכנת, הארכיטקטורה היא מודל לוגי של המחשב. המודל הלוגי מאפשר לנו להבין את המבנה ואת עקרונות של הפעולה של המחשב, בלי לרדת לפרטים הנוגעים לאופן המימוש. מימוש של ארכיטקטורה דורש גם הבנה בטכנולוגיות קיימות, כמו סוגי זיכרונות ומעגלים אלקטרוניים; בספר זה לא נתייחס לטכנולוגיות אלה.

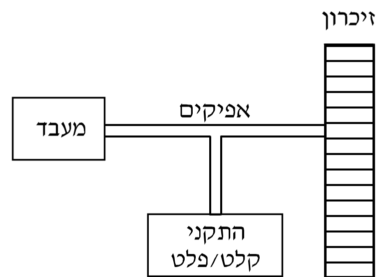
בארכיטקטורה של המחשב אנו נתייחס לשני מרכיבים:

1. **הארכיטקטורה של אוסף הוראות המכונה** – Instruction Set Architecture ובקיצור ISA. מנקודת מבט זו אנו מתעניינים בנתונים האלה:
   - הפעולות שהמחשב מסוגל לבצע (למשל, האם כפל היא אחת מפעולות המחשב, או אולי נצטרך בפעולות חיבור בלבד לביצוע הכפל);
   - במבנה של הוראה;
   - בסוגי נתונים בהם המחשב מטפל, ואופן הגישה אליהם.
2. **הארכיטקטורה של חומרת המחשב** – Hardware System Architecture, ובקיצור HAS, המתארת באופן עקרוני כיצד המחשב פועל כאשר הוא מבצע את הוראות המכונה. מנקודת מבט זו אנו נתייחס לשאלות כגון:
   - באילו רכיבי חומרה אנו משתמשים בהוראה;
   - מהן היחידות העיקריות במחשב ומה הקשר ביניהן;
   - איזה מידע זורם בין היחידות השונות.

תכנון הארכיטקטורה של מחשב מושפע מגורמים רבים, כמו: היישומים שרוצים להריץ במחשב, הטכנולוגיה, ההיסטוריה של התפתחות המחשבים וגם ההתפתחות הצפויה בתחום המִחשוב. למעשה, רוב המחשבים מבוססים כיום על הארכיטקטורה שפותחה לפני כשישים שנה, והיא ידועה בשם "**ארכיטקטורת פון נוימן**". פון נוימן וצוות של מדענים פִּתחו בשנת 1951 מחשב שנקרא EDVAC המבוסס על עקרון "התכנית המאוחסנת", לפיו התכנית מקודדת כמספרים ומאוחסנת באותו זיכרון שבו מאוחסנים הנתונים. לפני כן, המחשבים שנבנו אחסנו את התכניות בנפרד מן הנתונים ובהתאם לכך – המחשב כלל יחידה לטיפול בנתונים ויחידה נפרדת לטיפול בתכנית. בהתאם לארכיטקטורה של פון ניומן מערכת מחשב כוללת ארבעה מרכיבים עיקריים:

- המעבד (Processor) שתפקידו לבצע את התכנית;
- יחידת הזיכרון (Memory) בה מאוחסנים נתונים והוראות;

- ערוצי תקשורת הנקראים "אפיקים" (buses) אשר מקשרים ומעבירים מידע בין המעבד לזיכרון.
- התקני קלט-פלט



איור 1.3
מבנה מחשב בארכיטקטורה של פון ניומן

בסעיף הבא נציג מודל של "מחשב" פשוט, הבנוי לפי העקרונות של פון נוימן. מודל זה יאפשר לנו להדגים את העקרונות הבסיסיים של אופן פעולת המחשב ולעקוב אחר תהליך הביצוע של הוראות השמה והוראות אריתמטיות.

1.3 מבנה מחשב פשוט ואופן ביצוע ההוראות בשפת מכונה

בסעיף זה נגדיר ארכיטקטורה של מחשב פשוט. ארכיטקטורה זו כוללת הגדרה של כמה הוראות מכונה, הגדרת המרכיבים העיקריים של החומרה, והקשרים ביניהם.
המחשב שנגדיר מבצע את הפעולות האלה:

- חיבור
- השמה
- השוואה בין שני מספרים

המודל שנציג מוגבל בסוג הפעולות שהוא מבצע, ולמרות זאת נשמרים בו העקרונות עליהם מבוססים מחשבים אמיתיים. נתחיל את תיאור הרכיבים העיקריים של המחשב הפשוט מתיאור של הפעולות הדרושות להרצת תכנית בשפת מכונה. כדי לפשט את ההסברים,

במודל של המחשב שאנו מציגים לא נשתמש ביחידות קלט/פלט. בהמשך נראה כי תהליך הביצוע של הוראות קלט/פלט זהה לתהליך הביצוע של הוראות השמה והוראות אריתמטיות.

תהליך הביצוע של הוראה בשפת מכונה כולל שני שלבים עיקריים:

- שלב ההבאה (fetch) – קריאת ההוראה ופענוחה
- שלב הביצוע (execute) ההוראה

התהליך של ההבאה והביצוע נקרא גם "**מחזור הבאה-וביצוע**".

1.3.1 מבנה ה"מחשב" הפשוט

בסעיף זה נציג את הרכיבים העיקריים ב"מחשב" הפשוט אליהם נתייחס בתהליך הרצת התכנית.

א. יחידת הזיכרון

ב"מחשב" שלנו יחידת הזיכרון בנויה כמערך חד-ממדי המכיל 100 תאים; בכל תא אפשר לאחסן מספר עשרוני בן 6 ספרות. מספר זה יכול להיות נתון, הוראה או כתובת של תא בזיכרון. המעבד פונה לתא מסוים כדי לבצע אחת משתי הפעולות האלה:

א. קריאה של נתון או הוראה;

ב. כתיבה של נתון.

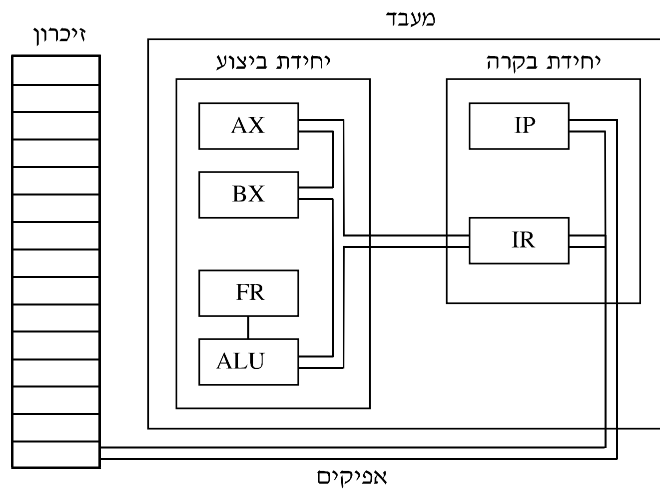
לשם כך יש לכל תא בזיכרון כתובת שהיא מספר בין 0 ל-99.

במחשב שלנו, לפני הרצת התכנית, אנו רושמים בזיכרון את ההוראות בשפת מכונה, וכל הוראה נמצאת בתא נפרד. כזכור, המחשב שלנו בנוי לפי העקרון של פון נוימן, דהיינו: עקרון התכנית המאוחסנת, ולכן הוראות התכנית מאוחסנות באותו זיכרון שבו מאוחסנים גם הנתונים. לכן, כדי להפריד את הנתונים מההוראות, נרשום את הנתונים בכתובות העליונות של הזיכרון (כגון כתובת 98, 99) ואת התכנית בכתובות הנמוכות של הזיכרון, החל מהכתובת 0.

ב. המעבד

המעבד במחשב שלנו כולל שתי יחידות עיקריות, שכל אחת מהן מטפלת בשלב אחר של מחזור ההבאה-ביצוע:

- יחידת הבקרה מטפלת בשלב ההבאה של הוראה מהזיכרון (כולל גם פענוח של הוראה)
- יחידת הביצוע מטפלת בביצוע הוראות אריתמטיות והוראות לוגיות



איור 1.4
מבנה סכמתי של המחשב הפשוט

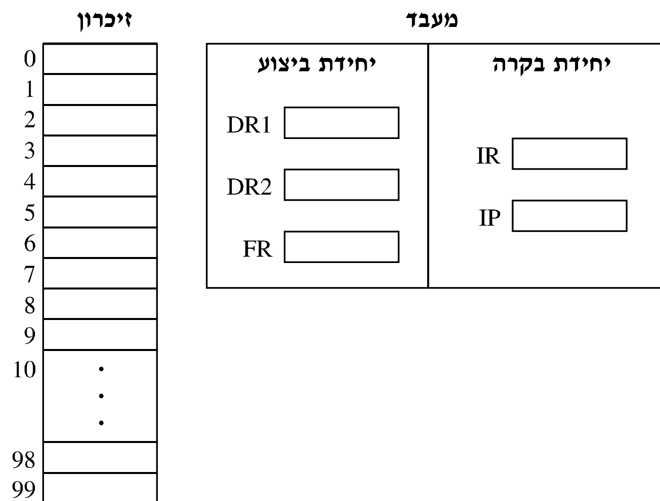
כדי לבצע את שלב ההבאה צריכה יחידת הבקרה לדעת את הכתובת של התא שבו מאוחסנת ההוראה שעליה לבצע, והיא צריכה לשמור את ההוראה שנקראה מהזיכרון כדי שתוכל לפענח אותה. לשם כך כוללת יחידת הבקרה כמה יחידות זיכרון קטנות, שכל אחת מהן היא בגודל תא אחד, היכול להכיל מספר בן 6 ספרות (כמו תא בזיכרון). כל יחידת זיכרון כזו נקראת "**אוגר**" (register); לכל אוגר יש שם שמציין את תפקידו ובו משתמשים כאשר פונים לאוגר. אוגר אחד משמש לשמירת הכתובת של ההוראה שהמעבד צריך להביא מהזיכרון. אוגר זה נקרא "**מצביע להוראה**" (Instruction Pointer ובקיצור IP).

אוגר שני משמש לשמירת ההוראה שהובאה מהזיכרון אל המעבד והוא נקרא "**אוגר הוראות**" (Instruction Register ובקיצור IR). זהו האוגר שבו מאחסנים את ההוראה שצריך לפענח ולעבד.

תפקידה של יחידת הביצוע הוא לבצע את השלב השני בתהליך ביצוע ההוראה; היא מכילה בין השאר כמה רכיבים:

1. היחידה האריתמטית-לוגית (Arithmetic Logic Unit ובקיצור ALU), שתפקידה לבצע את הפעולות המוגדרות בהוראות התכנית, כגון: העתקת נתונים ממקום למקום, פעולות אריתמטיות ופעולות לוגיות.
2. שני אוגרים אותם נכנה AX ו-BX. באוגרים אלה מאוחסנים הנתונים בהם משתמשת היחידה האריתמטית-לוגית לביצוע הפעולות השונות.
3. אוגר נוסף הנקרא "אוגר דגלים" (Flag register ובקיצור FR) שבו נשמר מידע על תוצאת ההשוואה. אוגר זה משמש לטיפול בפעולות לוגיות הקשורות להשוואה של מספרים, כמו: האם X = Y?

במעקב אחר מחזור ההבאה-והביצוע נתייחס למצב האוגרים והזיכרון שבהם נשמרים ההוראות והנתונים כמתואר באיור 1.5.



איור 1.5
תיאור הזיכרון והאוגרים במחשב הפשוט

מובן שהמחשב צריך להכיל רכיבים רבים נוספים כדי לפעול, אבל אנו מעוניינים להציג מודל של מחשב פשוט ולכן נתעלם משאר הרכיבים.

1.3.2 מבנה הוראה בשפת מכונה

הוראה בשפת מכונה, כמו הוראה בשפה עילית, מתארת את הפעולה שיש לבצע ואת הנתונים הדרושים לביצוע הפעולה. הפעולה נקראת **"אופרטור"** (operator); דוגמה לאופרטור בשפת אסמבלר: MOV, כאשר מקבילו בשפת C הוא הסימן =.
הנתונים נקראים **"אופרנדים"** (operand); דוגמה לאופרנדים תוכלו לראות במשפט ההשמה הרשום להלן בשפה עילית:

בשפת C:	A=8;
בשפת פסקל:	A:=8;

משפט ההשמה הזה כולל אופרטור אחד (=) ושני אופרנדים: המשתנה A והנתון 8. אך לא לכל המשפטים בשפה עילית יש מבנה פשוט; לדוגמה נרשום משפט מורכב יותר:

בשפת C:	A=A+B – C
בשפת פסקל:	A:=A+B – C;

משפט זה כולל שלושה אופרטורים: + – = ושלושה אופרנדים: A B C. בשפה עילית יש משפטים יותר מורכבים, כגון משפט for או משפט if הכוללים תנאים לוגיים מורכבים; בהמשך נלמד, שגם בשפת אסמבלי ישנן הוראות הסתעפות, אך הן פשוטות בהרבה מפסוקי הסתעפות בשפות עיליות.

בניגוד למורכבות של ההוראות בשפה עילית, המבנה של הוראת מכונה הוא הרבה יותר פשוט. כל הוראה צריכה לכלול רק אופרטור אחד, ומספר האופרנדים משתנה בין 0 ל-2. מבנה הוראות פשוט מאפשר לבנות חומרה מתאימה לביצוע ההוראה בצורה יעילה. במחשב שלנו הוראה תהיה בנויה במתכונת של אחד משלושת המבנים האלה:

הוראה ללא אופרנדים:	[אופרטור]
הוראה עם אופרנד:	[אופרנד יעד] [אופרטור]
הוראה עם שני אופרנדים:	[אופרנד מקור] [אופרנד יעד] [אופרטור]

שימו לב, הוראה נקראת תמיד משמאל לימין.

ההוראה מתחילה תמיד עם מספר המציין את האופרטור ואחר כך נרשמים מספרים המגדירים את האופרנדים. בשפת מכונה של המחשב שלנו, אופרנד יכול להיות מאחד משלושת הסוגים הבאים:

- **נתון מיידי**, שזהו נתון שהוא עצמו רשום בהוראה
- אחד **האוגרים** (AX, BX)
- תא בזיכרון

כפי שכבר נאמר, הוראה מיוצגת כמספר. כיוון שכל הוראה נשמרת בתא אחד בזיכרון, המספר שמייצג את ההוראה יהיה לכל היותר בן 6 ספרות. כחלק מהגדרת אוסף הוראות המכונה עלינו לקבוע קוד מספרי ייחודי לכל אופרטור ולכל אופרנד, כך שהפענוח של ההוראה יהיה חד משמעי. במחשב שלנו נגדיר כל אופרטור ואופרנד כמספר בין 0 ל-99. כמו-כן נקבע קודים מספריים לאוגרים המשתתפים בביצוע פעולות אריתמטיות-לוגיות, כאשר:

- קוד 01 מציין את האוגר AX
- קוד 02 מציין את האוגר BX

לדוגמה, המבנה של הוראה שיש בה שני אופרנדים יהיה:

[xx] [yy] [zz]

[xx] הוא מספר בן שתי ספרות המציין סוג פעולה (אופרטור);

[yy] ו-[zz] הם מספרים בני שתי ספרות המציינים אופרנדים.

לדוגמה, בהוראה: 12 01 06 הקוד 06 מציין אופרטור, בדוגמה זו הוראת חיבור (אותה נציג בהמשך), והמספרים 12 ו-01 מציינים אופרנדים, בדוגמה זו: 01 מציין אוגר AX, ו-12 הוא נתון מיידי.

1.3.3 כתיבת תכנית המחברת שני משתנים

לאחר שהגדרנו מבנה של הוראת מכונה, נגדיר שלוש הוראות: הוראת העברה (זהו המונח המקובל בשפת אסמבלי להשמה), הוראת חיבור והוראה לסיום ביצוע התכנית. לאחר שנדגים כיצד כותבים תכנית פשוטה עם הוראות כאלה, נציג שלושה סוגים נוספים של הוראות, שיאפשרו לכתוב מבנה בקרה מותנה.

הוראת העברה

ההוראה הראשונה שנגדיר היא הוראת העברה, אשר מעתיקה את התוכן של אופרנד שנקרא "אופרנד המקור" ומשימה אותו באופרנד שני שנקרא "אופרנד היעד". מסמנים זאת כך:
אופרנד מקור ← אופרנד יעד

שימו לב, שזהו סימון בלבד, ולא צורת ההוראה עצמה בתכנית בשפת אסמבלי. ההוראה עצמה בשפת אסמבלי תירשם במבנה הזה:

[xx] [yy] [zz]

סוגי האופרנדים האפשריים בהוראה כזו הם:

- אופרנד היעד חייב להיות רכיב זיכרון שבו ניתן לשמור את הנתון ולכן הוא יכול להיות אחד משני האוגרים AX, BX או תא בזיכרון.
- אופרנד המקור יכול להיות אחד מארבע האפשרויות: נתון מיידי, אוגר AX או אוגר BX או תא בזיכרון.

הגדרת סוגי האופרנדים הללו לא מאפשרת העברה ישירה של הנתונים בין שתי כתובות בזיכרון, והיא קובעת חמישה צירופים של אופרנדים בהם מותר להשתמש בהוראה זו:

נתון מיידי ← אוגר
תוכן אוגר ← אוגר
תוכן כתובת בזיכרון ← אוגר
נתון מיידי ← כתובת בזיכרון
תוכן אוגר ← כתובת בזיכרון

בהתאם, נגדיר חמש הוראות העברה, שכל אחת מהן מגדירה צירוף מסוים של אופרנדים, ונבחר עבורן את הקודים 01 עד 05. הטבלה הבאה מתארת את קוד הפעולה של הוראות ההעברה.

טבלה 1.2
הוראות העברה

תיאור של ההוראה	מספר מציין קוד פעולה
נתון מיידי ← אוגר יעד	01
אוגר מקור ← אוגר יעד	02
תוכן כתובת בזיכרון ← אוגר יעד	03
נתון מיידי ← כתובת בזיכרון	04
אוגר מקור ← כתובת בזיכרון	05

נתאר כמה דוגמאות להוראות השמה:

שים את הנתון 27 באוגר AX	01 01 27
שים את תוכן האוגר BX באוגר AX	02 01 02
שים את הנתון 27 בכתובת 99 בזיכרון	04 99 27

חשבו, כיצד נעתיק תוכן תא שכתובתו 99 לתא שכתובתו 98?

צירופי האופרנדים מוגבלים, לכן לא נוכל לרשום את הצירוף:

שים את תוכן כתובת תא 99 בכתובת תא 98

לכן, עלינו לרשום שתי הוראות:

ההוראה 99 01 03 שפירושה "שים תוכן כתובת תא 99 באוגר AX"

וההוראה 01 98 05 שפירושה שים תוכן AX בתא שכתובתו 98

שימוש בקודים מספריים עשוי לבלבל. למשל המספר 01 יכול להיות אחד מאלה:

- נתון מיידי, כלומר: הנתון עצמו;
- מציין של אוגר AX;
- כתובת תא בזיכרון;
- קוד של אופרטור;

כיצד המעבד יבדיל ביניהם? התשובה לכך פשוטה: המעבד מבדיל בין אופרטור לאופרנד על-פי **מיקום** הקוד בהוראה. במחשב הפשוט ההוראה מתחילה תמיד בקוד של אופרטור.

לדוגמה: הוראה המתחילה ב-01 היא הוראת השמה של נתון באוגר מסוים, והמספר 01 במקרה זה מציין את אופרטור פעולת ההשמה; ואילו 01 המופיע במיקום 3 ו-4 (yy) או במיקום 5 ו-6 (zz) יכול להיות נתון מיידי (המספר 1 עצמו) או האוגר AX או כתובת בזיכרון.

כדי להבטיח פענוח חד משמעי של ההוראה, נגדיר קודי הוראה שונים, בהתאם לסוג הפעולה ולסוג האופרנדים עליהם היא מתבצעת. כך לדוגמה, הגדרנו קוד 02 להעברה של תוכן אוגר אחד לאוגר אחר וקוד נוסף 05 להעברה של תוכן אוגר לתא בזיכרון. כלומר, לכל פעולה נגדיר משפחה של הוראות מכונה, שבה האופרטור מתאר פעולה זהה המתבצעת על צירוף שונה של אופרנדים. לדוגמה את ההוראה 23 01 01 נוכל לפענח בצורה הבאה:

23	01	01
נתון מיידי	אוגר AX	קוד פעולה
משמעות ההוראה:	AX ← 23	

הוראת חיבור

ההוראה השנייה היא הוראת חיבור. הוראה זו כוללת שני אופרנדים והיא ניתנת לתיאור בצורה הבאה:

אופרנד מקור + אופרנד יעד ← אופרנד יעד

סוגי האופרנדים האפשריים בהוראה זו הם:

- אופרנד היעד חייב להיות אחד משני האוגרים AX או BX והוא לא יכול להיות תא בזיכרון;
- אופרנד המקור יכול להיות אחת משתי אפשרויות: נתון מיידי או אוגר.

תוצאת החיבור מושמת תמיד באופרנד היעד.

כיוון שהגבלנו את צירופי האופרנדים האפשריים בהוראת חיבור, נצטרך להגדיר רק שתי הוראות חיבור שונות, והן מתוארות בטבלה 1.3. הקודים שנבחר כקוד פעולה המציינת חיבור הם: 06 ו-07.

טבלה 1.3
הוראות חיבור

תיאור של ההוראה	**מספר מציין קוד פעולה**
נתון + אוגר ← אוגר	06
אוגר מקור + אוגר יעד ← אוגר יעד	07

נרשום כמה הוראות לדוגמה:

חשב AX + 27 ושים את התוצאה ב-AX	06 01 27
חשב AX + BX ושים את התוצאה ב-AX	07 01 02

המגבלה על סוגי האופרנדים, האפשריים בכל פעולה, מגבילה את מספר ההוראות בהן נוכל להשתמש, אולם היא מאפשרת מבנה מעבד פשוט ויחידת פענוח פשוטה.

הוראת סיום

כדי לסיים את ביצוע התכנית, נגדיר הוראת סיום שקוד הפעולה שלה הוא 00;
הוראה זו היא ללא אופרנדים.

דוגמה 1.1 תכנית לחיבור מספרים

נרשום תכנית בשפת המכונה שהגדרנו שתבצע את הפעולות האלה:

שים את הנתון 23 במשתנה A
שים את הנתון 54 במשתנה B
חשב A + B ושים את התוצאה ב-A

בתכנית זו A ו-B מציינים משתנים בזיכרון.

תהליך התכנון

נשתמש בהוראות העברה כדי להשים נתונים למשתנים A ו-B. כיוון שלא קיימת הוראה לחיבור שבה אחד מהאופרנדים הוא תא זיכרון, נעתיק את הנתונים מ-A ו-B אל האוגרים AX ו-BX נחבר אותם ונשמור את התוצאה באוגר AX. לסיום נעתיק את התוצאה מאוגר

AX אל המשתנה A. בנוסף, נגדיר שני תאי זיכרון בכתובות 98 ו-99 בהן נשתמש. בתא שכתובתו 98 נאחסן את A ובתא שכתובתו 99 נאחסן את B.
כעת נרשום תכנית בשפת מכונה של המחשב שלנו :

הסבר	הוראה בשפת מכונה
שים את הנתון 23 בתא שכתובתו 98	04 98 23
שים את הנתון 54 בתא שכתובתו 99	04 99 54
שים את תוכן התא שכתובתו 98 באוגר AX	03 01 98
שים את תוכן התא שכתובתו 99 באוגר BX	03 02 99
חשב AX+BX ושים את התוצאה באוגר AX	07 01 02
שים את תוכן אוגר AX בתא שכתובתו 98	05 98 01
סיום	00

כדי לבצע את ההוראות, יש לרשום בזיכרון אותן ואת הנתונים, כאשר ההוראות מאוחסנות ברצף של תאים בזיכרון המתחיל בתא שכתובתו 0. נציין כי במחשב אמיתי מערכת הפעלה אחראית על פעולה זו. מצב הזיכרון והאוגרים לאחר כתיבת התכנית לזיכרון מוצג באיור 1.6.

מעבד

יחידת בקרה: IR [ ], IP [00]

יחידת ביצוע: DR1 [ ], DR2 [ ], FR [ ]

זיכרון

כתובת	תוכן
00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
. . .	
97	
98	?
99	?

איור 1.6
מצב התחלתי של התכנית בדוגמה 1.1

שימו לב, הערך של מונה התכנית – האוגר IP – הוא 0, כי הוא אמור להצביע על כתובת ההוראה הראשונה לביצוע. ההוראה האחרונה בתכנית מאוחסנת בתא שכתובתו 6, ואילו התאים שכתובתם 98 ו-99, המייצגים את המשתנים A ו-B, מכילים ערך לא ידוע .

מעקב אחר ביצוע התכנית

כעת נעקוב אחר ביצוע התכנית, תוך תיאור מצב הזיכרון ומצב האוגרים בשלבי ההבאה והביצוע של כל הוראה.

1. ההוראה הראשונה - 23 98 04

א. השלב הראשון בביצוע ההוראה הוא שלב ההבאה והוא כולל את הפעולות האלה :

- המעבד קורא מהתא בזיכרון, שכתובתו 0 (עליו IP מצביע) את ההוראה הראשונה ומאחסן אותה באוגר ההוראות IR.
- המעבד מקדם את אוגר IP ב-1, כך שהוא יצביע על ההוראה הבאה (השנייה בתכנית).
- המעבד מפענח את ההוראה כדי לדעת מה עליו לבצע.

מצב האוגרים והזיכרון בסיום שלב ההבאה של ההוראה הראשונה מוצג באיור 1.7א.

מעבד

יחידת בקרה	
IR	04 98 23
IP	01

יחידת ביצוע	
DR1	
DR2	
FR	

זיכרון

כתובת	תוכן
00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
. . .	
97	
98	?
99	?

איור 1.7 א
סיום שלב ההבאה של ההוראה : 23 98 04
ב. השלב השני – ביצוע ההוראה

המעבד כותב את הנתון 23 בתא שכתובתו 98. מצב האוגרים והזיכרון בסיום ההוראה מתואר באיור 1.7ב.

זיכרון

כתובת	תוכן
00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
	. . .
98	23
99	

מעבד

יחידת בקרה

אוגר	תוכן
IR	04 98 23
IP	01

יחידת ביצוע

אוגר	תוכן
DR1	
DR2	
FR	

איור 1.7 ב
סיום שלב הביצוע של ההוראה 23 98 04

2. ההוראה השנייה – 54 99 04

מאחר ששלב הביצוע במחזור ההבאה-ביצוע הסתיים, עוברת יחידת הבקרה אל שלב ההבאה הבא. להלן תיאור ההתרחשויות :

א. שלב ההבאה של ההוראה השנייה דומה לשלב ההבאה של ההוראה הראשונה, כלומר : המעבד קורא מהזיכרון את ההוראה מכתובת 1 (הכתובת ששמורה באוגר IP) ומיד מקדם את אוגר IP ב-1, כך שהוא יצביע על ההוראה הבאה (השלישית בתכנית, שכתובתה 02). לאחר מכן הוא מפענח את ההוראה שהובאה זה עתה, דהיינו : ההוראה השנייה.

בסיום שלב ההבאה של ההוראה השנייה מצב האוגרים והזיכרון הוא :

מעבד

יחידת בקרה		יחידת ביצוע	
IR	04 99 54	DR1	
IP	02	DR2	
		FR	

זיכרון

כתובת	תוכן
00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
	. . .
98	23
99	

איור 1.8 א
סיום שלב ההבאה של ההוראה 54 99 04

ב. בשלב הביצוע כותב המעבד את הנתון 54 בתא שכתובתו 99. מצב האוגרים והזיכרון לאחר ביצוע ההוראה השנייה הוא:

מעבד

יחידת בקרה		יחידת ביצוע	
IR	04 99 54	DR1	
IP	02	DR2	
		FR	

זיכרון

כתובת	תוכן
00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
	. . .
98	23
99	54

איור 1.8 ב
סיום שלב הביצוע של ההוראה 54 99 04

3. ההוראה השלישית – 98 01 03

בשלב ההבאה קורא המעבד מהזיכרון את ההוראה השלישית מכתובת 02 (הכתובת שמורה באוגר IP) ומקדם את אוגר IP ב-1, כך שהוא מצביע על ההוראה הבאה (הרביעית בתכנית). לאחר פענוח ההוראה מתחיל שלב הביצוע: המעבד פונה לזיכרון כדי לקרוא את הנתון מכתובת 98 ואחסונו באוגר AX. לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא:

מעבד

יחידת ביצוע	
DR1	23
DR2	
FR	

יחידת בקרה	
IR	03 01 98
IP	03

זיכרון

00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
	. . .
98	23
99	54

איור 1.9
סיום מחזור הבאה-ביצוע של ההוראה 98 01 03

4. ההוראה הרביעית – 99 02 03

תהליך ביצוע ההוראה הזו וההוראות הבאות הוא זהה, לכן נסתפק בהצגת מצב הזיכרון והאוגרים בסיום ההוראה. בסיום הוראה זו נקרא הנתון 54 מהתא שכתובתו 98 ונשמר באוגר BX. לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא כמתואר באיור 1.10.

5. ההוראה החמישית – 02 01 07

בסיום ביצוע הוראה זו, האוגר AX מכיל את הסכום של 23 + 54. בהתאם לכך מצב האוגרים והזיכרון בסיום ביצוע ההוראה הוא כמתואר באיור 1.11.

זיכרון

00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
	. . .
98	23
99	54

מעבד

יחידת בקרה: IR 03 02 99, IP 04

יחידת ביצוע: DR1 23, DR2 54, FR

איור 1.10
סיום מחזור הבאה-ביצוע של ההוראה 99 02 03

זיכרון

00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
	. . .
98	23
99	54

מעבד

יחידת בקרה: IR 07 01 02, IP 05

יחידת ביצוע: DR1 77, DR2 54, FR

איור 1.11
סיום מחזור הבאה-ביצוע של ההוראה 02 01 07

6. ההוראה השישית – 01 98 05

בעת ביצוע הוראה זו המעבד כותב את התוצאה 77 לתא שכתובתו 98. בסיום מחזור ההבאה-ביצוע של הוראה זו מצב האוגרים והזיכרון הוא כמתואר באיור 1.12.

זיכרון

כתובת	תוכן
00	04 98 23
01	04 99 54
02	03 01 98
03	03 02 99
04	07 01 02
05	05 98 01
06	00
07	
08	
09	
⋮	
98	77
99	54

מעבד

יחידת בקרה		יחידת ביצוע	
IR	05 98 01	DR1	77
IP	06	DR2	54
		FR	

איור 1.12
סיום מחזור ההבאה-ביצוע של ההוראה 01 98 05

7. ההוראה השביעית – 00

לאחר ביצוע ההוראה השישית, האוגר IP יצביע על כתובת התא 06 המכילה הוראה לסיום התכנית. המעבד יפסיק את פעולתו לאחר שיקרא את ההוראה ויפענח אותה.

שאלה 1.2

א. רשמו הוראות מכונה לחיבור תוכן התא שכתובתו 99 עם התא שכתובתו 98 ולשמירת התוצאה בתא שכתובתו 99.

ב. כתבו הוראות מכונה שיגדילו ב-2 את תוכנו של תא שכתובתו 99.

שאלה 1.3

כתבו הוראות בשפת מכונה לביצוע רצף הפעולות האלה:

שים 34 במשתנה A
שים 15 במשתנה B
הגדל את ערכו של A ב-1
חשב A+B ושים את התוצאה ב-B

הניחו כי למשתנה A מוקצה התא שכתובתו 98 ולמשתנה B מוקצה התא שכתובתו 99.

שאלה 1.4

א. כתבו הוראות בשפת מכונה לביצוע רצף הפעולות האלה:

```
שים 15 במשתנה A
שים 32 במשתנה B
חשב A+B+19 ושים את התוצאה ב-A
```

ב. תארו את מצב הזיכרון והאוגרים (באמצעות מפת זיכרון) לאחר ביצוע מחזור ההבאה-ביצוע של כל אחת מהוראות התכנית. הניחו כי הוראות התכנית מאוחסנות בזיכרון ברצף תאים, החל מתא שכתובתו 0 ואילו המשתנים A ו-B מאוחסנים בתאים שכתובתם 98 ו-99.

1.3.4 ביצוע תכנית הכוללת הוראות בקרה

תכניות רבות כוללות לא רק ביצוע הוראות באופן סדרתי, הוראה אחר הוראה, אלא גם ביצוע מותנה (הוראות תנאי) ולולאות. נניח שברצוננו לבצע את הפעולה הזו:

```
אם    A = B   אזי
       A = 2
אחרת
       B = 2
```

כדי לכתוב פעולה זו בשפת המכונה של המחשב שלנו, עלינו להרחיב תחילה את אוצר ההוראות. עד כה הגדרנו שלושה סוגים (משפחות) של הוראות מכונה (שקוד הפעולה שלהן הוגדר מ-00 עד 07), ובאפשרותנו לקבוע עוד 92 הוראות מכונה שונות.

בסעיף זה נציג עוד שלושה סוגים של הוראות:

- הוראת השוואה בין שני נתונים
- הוראת קפיצה (הסתעפות) מותנית
- הוראת קפיצה בלתי מותנית

לסיום נציג תכנית המשתמשת בהוראות אלה למימוש מבנה בקרה מותנה.

הוראת השוואה

הסוג הרביעי של הוראות הוא הוראת השוואה. הוראה זו מבצעת השוואה בין שני אופרנדים. במקרה זה אין אופרנד מקור ואופרנד יעד, מאחר שההוראה משווה בין שני האופרנדים, ולא משנה אף אחד מהם, אלא מעדכנת את תוכנו של אוגר הדגלים FR בהתאם לתוצאת ההשוואה; ניתן לומר, שבמקרה זה היעד הוא אוגר הדגלים. נרשום את האלגוריתם המתאר את אופן ביצוע ההוראה:

אם האופרנד הראשון = אופרנד השני אזי
$FR \leftarrow 1$
אחרת
$FR \leftarrow 0$

סוגי האופרנדים שאפשר לרשום בהוראה זו הם:

- האופרנד הראשון יכול להיות אחד משני האוגרים AX או BX.
- האופרנד השני יכול להיות נתון מיידי או אחד משני האוגרים AX או BX.
- שימו לב, אף אחד מהאופרנדים אינו יכול להיות תא בזיכרון.

בהתאם לצירוף האופרנדים נגדיר שתי הוראות השוואה שונות:

טבלה 1.3
הוראות השוואה

תיאור מילולי של ההוראה	מספר המציין קוד פעולה
האם הנתון המיידי = תוכן האוגר הראשון?	08
האם האוגר השני = תוכן האוגר הראשון?	09

אוגר הדגלים יקבל את הערך 1 אם הנתונים שווים, ו-0 אחרת. ניזכר במטרתנו – כתיבת תכנית לביצוע השוואה בין A ו-B. לאחר ביצוע הוראת ההשוואה, יספק לנו אוגר הדגלים מידע שבו נשתמש כדי להחליט אם עלינו לבצע את הוראת ההשמה A=2 או את הוראת ההשמה B=2. השימוש במידע זה ייעשה בעזרת הוראת קפיצה מותנית.

הוראת קפיצה מותנית

בקבוצת ההוראות החמישית יש הוראה אחת בלבד, והיא הוראת הקפיצה המותנית. הוראה זו קובעת מי היא ההוראה הבאה שתתבצע, בהתאם לתוכן אוגר הדגלים.

אם אוגר הדגלים מכיל את הערך 1 – יועבר הביצוע לכתובת שמציין האופרנד הכלול בהוראה; אחרת – אם אוגר הדגלים יכיל את הערך 0, ביצוע התכנית ימשיך כרגיל, מההוראה העוקבת להוראת הקפיצה המותנית. האלגוריתם הבא מתאר את אופן הביצוע של הוראת הקפיצה המותנית:

אם FR = 1 אזי
מספר המציין כתובת ← IP

בהוראה זו יש רק אופרנד אחד, שהוא מספר (בין 0 ל-99) המציין את הכתובת של ההוראה ממנה יימשך ביצוע התכנית. נזכור, שבשלב ההבאה הבא, יחידת הבקרה קוראת את ההוראה שכתובתה מצוייה ב-IP. לכן, אם ערכו של אוגר הדגלים הוא 1, וכתוצאה מכך IP קיבל ערך חדש, הרי שההוראה הבאה תקרא מהזיכרון מהכתובת החדשה.

בהוראה זו נגדיר את הקוד 10 לציון האופרטור; מבנה ההוראה הוא:
מספר המציין כתובת 10

לדוגמה, משמעות ההוראה 07 10 היא: אם אוגר הדגלים מכיל את הערך 1, טען את IP בערך 07; כתוצאה מכך יעבוֹר המחשב לביצוע ההוראה הנמצאת בתא שכתובתו 07.

הוראת קפיצה בלתי מותנית

ההוראה השישית והאחרונה שנגדיר היא הוראת הקפיצה הבלתי מותנית, אשר מעבירה את הביצוע (ללא תנאי) להוראה כלשהי בתכנית שאינה בהכרח עוקבת להוראה הקודמת. הוראה זו היא בת אופרנד אחד, המציין כתובת של תא בזיכרון שבו נמצאת ההוראה שממנה יימשך ביצוע התכנית. הוראת הקפיצה הבלתי-מותנית מתבצעת בכל מקרה, והיא אינה תלויה בערך של אוגר הדגלים.

בהוראה זו נגדיר את הקוד 11 לציון האופרטור; המבנה של ההוראה הוא:

מספר המציין כתובת 11

לדוגמה, משמעות ההוראה 07 11 היא: טען את IP בערך 07; למעשה, המשמעות היא – עבור לביצוע ההוראה הנמצאת בתא שכתובתו 07.

דוגמה 1.2 מימוש מבנה בקרה

בטבלה 1.5 רשומה קבוצת הוראות שתבצע את הפעולה הזו:

אם A שווה ל-B אזי
שים 2 במשתנה A
אחרת
שים 2 במשתנה B

הוספנו לטבלה עמודה שמתארת את כתובת ההוראה בזיכרון. עמודה זו מסייעת לנו לקבוע את ערך האופרנד הדרוש להוראות קפיצה. כמו כן בחרנו את התא שכתובתו 98 לאחסון A ואת התא שכתובתו 99 לאחסון B.

טבלה 1.5
הוראות התכנית

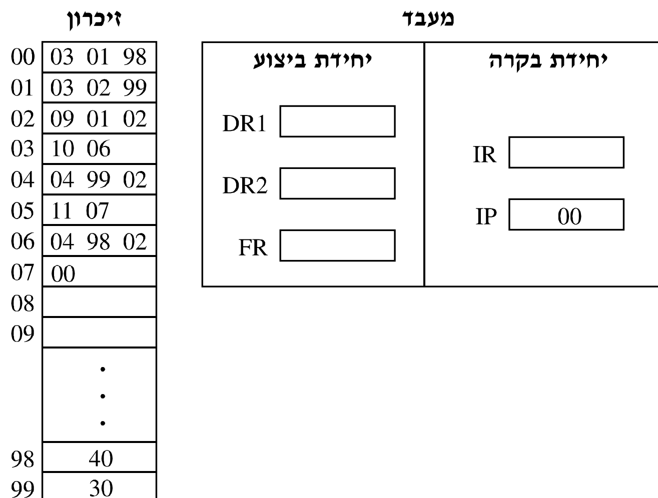
תיאור ההוראה	הוראה בשפת מכונה	מיקום הוראה בזיכרון
שים תוכן כתובת 98 באוגר AX	03 01 98	0
שים תוכן כתובת 99 באוגר BX	03 02 99	1
אם AX=BX שים באוגר הדגלים 1	09 01 02	2
אם תוכן אוגר הדגלים הוא 1, שים באוגר IP את הכתובת 06	10 06	3
שים 2 בתא שכתובתו 99	04 99 02	4
שים באוגר IP את הכתובת 07	11 07	5
שים 2 בתא שכתובתו 98	04 98 02	6
סיום תכנית	00	7

בביצוע תכנית זו יש שני מסלולים אפשריים (המוצגים בטבלה 1.6): מסלול אחד מתבצע כאשר A שונה מ-B ומסלול שני מתבצע כאשר A שווה ל-B.

טבלה 1.6
מסלולי ביצוע של התכנית המממשת מבנה בקרה

מסלול ראשון – מתבצע כאשר A≠B	מסלול שני – מתבצע כאשר A=B
שים תוכן כתובת 98 באוגר AX	שים תוכן כתובת 98 באוגר AX
שים תוכן כתובת 99 באוגר BX	שים תוכן כתובת 99 באוגר BX
אם AX = BX שים באוגר הדגלים 1	אם AX = BX שים באוגר הדגלים 1
תוכן אוגר הדגלים הוא 0, לכן ב-IP מושמת הכתובת 04	תוכן אוגר הדגלים הוא 1, לכן ב-IP מושמת הכתובת 06
שים 2 בתא שכתובתו 99	שים 2 בתא שכתובתו 98
שים באוגר IP את הכתובת 07	סיום תכנית
סיום תכנית	

כדי לתאר את המסלול הראשון נניח כי ערכו ההתחלתי של A הוא 40 וערכו ההתחלתי של B הוא 30. מצב האוגרים והזיכרון לפני תחילת ביצוע ההוראה השלישית בתכנית מתואר באיור 1.13.



איור 1.13
מצב התחלתי של האוגרים והזיכרון כאשר $A \neq B$

תיאור תהליך ביצוע התכנית במקרה ש- $A \neq B$

1. ההוראה הראשונה 98 01 03 וההוראה השנייה 99 02 03

ביצוע שתי ההוראות הראשונות דומה למחזור הבאה-ביצוע של ההוראה הראשונה בדוגמה הקודמת. לאחר ביצוע הוראות אלה מצב האוגרים והזיכרון הוא :

זיכרון

כתובת	תוכן
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	30

מעבד

יחידת ביצוע: DR1 = 40, DR2 = (ריק), FR = (ריק)

יחידת בקרה: IR = 03 02 99, IP = 02

איור 1.14
סיום מחזור ההבאה-ביצוע של שתי ההוראות הראשונות

2. ההוראה השלישית 02 01 09

לאחר שלב ההבאה של ההוראה השלישית, המעבד משווה את תוכן אוגרי הנתונים. כיוון שתוכנם שונה, הוא מציב באוגר הדגלים את הערך 0. לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא :

זיכרון

כתובת	תוכן
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	30

מעבד

יחידת בקרה: IR 09 01 02, IP 03

יחידת ביצוע: DR1 40, DR2 30, FR 0

איור 1.15
סיום מחזור ההבאה-ביצוע של ההוראה 09 01 02

3. הוראה רביעית 10 06

זוהי הוראת קפיצה מותנית. כיוון שערכו של FR=0, לא תתבצע קפיצה והביצוע ימשיך מההוראה הנוכחית (ההוראה הרביעית) להוראה העוקבת (החמישית, שכתובתה 04). לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא:

זיכרון

כתובת	תוכן
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	30

מעבד

יחידת בקרה: IR 10 06, IP 04

יחידת ביצוע: DR1 40, DR2 30, FR 0

איור 1.16
סיום מחזור ההבאה-ביצוע של ההוראה 10 06

5. הוראה חמישית 02 99 04

תהליך ההבאה-ביצוע של הוראה זו דומה לתהליך ההבאה-ביצוע של ההוראה הראשונה בתכנית. לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא:

מעבד

יחידת בקרה	
IR	04 99 02
IP	05

יחידת ביצוע	
DR1	40
DR2	30
FR	0

זיכרון

כתובת	תוכן
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
. . .	
98	40
99	02

איור 1.17
סיום מחזור ההבאה-ביצוע של ההוראה 02 99 04

6. הוראה שישית 07 11

ביצוע הוראה זו גורם לשינוי הערך של אוגר IP (לאחר שלפני כן הוא קודם ב-1). לכן נתאר בפירוט את שלב ההבאה ושלב הביצוע.

שלב ההבאה דומה לשלב ההבאה של כל הוראה; בסיום שלב זה הערך באוגר IP מקודם ב-1, כלומר האוגר יצביע על ההוראה הבאה (הנמצאת בכתובת 06). לאחר שלב ההבאה, מצב האוגרים והזיכרון הוא:

מעבד

יחידת בקרה		יחידת ביצוע	
IR	11 07	DR1	40
IP	06	DR2	50
		FR	0

זיכרון

כתובת	תוכן
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	02

איור 1.18 א
סיום שלב ההבאה של ההוראה 11 07

בשלב הביצוע מעודכן תוכן אוגר IP לערך 07. לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא:

מעבד

יחידת בקרה		יחידת ביצוע	
IR	11 07	DR1	40
IP	07	DR2	30
		FR	0

זיכרון

כתובת	תוכן
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	02

איור 1.18 ב
סיום שלב הביצוע של ההוראה 11 07

7. הוראה שביעית 00

ההוראה בכתובת 07 היא הוראת סיום, ולכן ביצוע התכנית מסתיים.

תיאור תהליך ביצוע התכנית במקרה ש- A = B

כעת נעקוב אחר ביצוע ההוראות כאשר ערכם ההתחלתי של A ו-B הוא 40.

1. ההוראה 98 01 03 וההוראה השנייה 99 02 03

ביצוע שתי ההוראות הראשונות זהה לתהליך הביצוע שתואר במקרה הקודם (בו A ≠ B).
לאחר ביצוע הוראות אלה, מצב האוגרים והזיכרון הוא :

מעבד

יחידת בקרה

IR	03 02 99
IP	02

יחידת ביצוע

DR1	40
DR2	40
FR	

זיכרון

00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	40

איור 1.19
סיום מחזור ההבאה-ביצוע של שתי ההוראות הראשונות

2. הוראה שלישית 02 01 09:

שלב ההבאה-ביצוע של ההוראה השלישית במקרה זה, דומה לשלב ההבאה-ביצוע של ההוראה השלישית שתיארנו במקרה הקודם בו A שונה מ-B, אלא שהפעם תוצאת ההשוואה היא "אמת", ולכן אוגר הדגלים מתעדכן וערכו הוא 1. לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא :

זיכרון | **מעבד**

כתובת	זיכרון
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	40

יחידת ביצוע		יחידת בקרה	
DR1	40	IR	02 01 09
DR2	40	IP	03
FR	1		

איור 1.20
סיום מחזור הבאה-ביצוע של ההוראה 09 01 02

3. הוראה רביעית 06 10

כיוון ש-FR=1, מתבצעת קפיצה, ולכן האוגר IP מתעדכן פעמיים :

– פעם ראשונה בשלב ההבאה, לאחר קידום IP ב-1 (וערכו 04)
– פעם שנייה בשלב הביצוע של הוראת הקפיצה המותנית, וערכו של IP נקבע ל- 06.

לאחר ביצוע הוראה זו מצב האוגרים והזיכרון הוא :

זיכרון | **מעבד**

כתובת	זיכרון
00	03 01 98
01	03 02 99
02	09 01 02
03	10 06
04	04 99 02
05	11 07
06	04 98 02
07	00
08	
09	
	. . .
98	40
99	40

יחידת ביצוע		יחידת בקרה	
DR1	40	IR	10 06
DR2	40	IP	06
FR	1		

איור 1.21
סיום מחזור הבאה-ביצוע של ההוראה 10 06

4. הוראה חמישית 02 98 04

בסיום מחזור ההבאה-ביצוע של הוראה זו, מאוחסן הערך 02 בתא שכתובתו 98.

ההוראה הבאה (00) מסיימת את ביצוע התכנית.

שאלה 1.5

כתבו תכנית, בשפת מכונה, שתבצע את הפעולות האלה:

$B \leftarrow 5$

$A \leftarrow 10$

$B \leftarrow B + 5$

אם $A = B$ אזי

$A \leftarrow A*2$

הדגימו את מהלך הביצוע של התכנית שכתבתם.

הניחו כי הקצו ל-A את התא שכתובתו 98 ול-B הקצו את התא שכתובתו 99.

1.4 הקשר בין יחידות המחשב

בסעיף זה נתאר כיצד זורמים הנתונים וההוראות בין המעבד והזיכרון. הקשר בין המעבד לזיכרון מתבצע באמצעות האפיקים (איו 1.4). בכל מחשב יש שלושה סוגים של אפיקים:

- אפיק הנתונים (Data bus) – בו מועברים נתונים והוראות
- אפיק הכתובות (Address bus) – בו מועברות כתובות של תאי זיכרון
- אפיק הבקרה (Control bus) – בו מועבר סוג הפעולה: קריאה או כתיבה

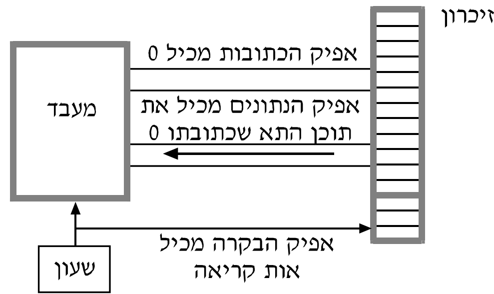
כל אפיק מורכב ממספר תילים ("חוטי חשמל") שבכל אחד מהם מועבר אות חשמלי שהוא חלק מהמידע הזורם באפיקים בין המעבד והזיכרון.

כזכור, המעבד פונה לזיכרון כדי לקרוא הוראה או נתון וכדי לכתוב נתונים. בהתאם לכך ניתן להגדיר שני סוגים של מחזורי אפיק. **מחזור אפיק** היא פעולה אחת של העברת מידע באמצעות האפיקים בין המעבד ליחידת הזיכרון.

א. מחזור קריאה

לדוגמה, כדי לקרוא הוראה, המאוחסנת בתא שכתובתו היא 0, המעבד מבצע את הפעולות האלה:

- המעבד רושם באפיק הכתובות את הערך 0;
- המעבד רושם באפיק הבקרה את סוג הפעולה הדרושה – פעולת קריאה (read);
- יחידת הזיכרון מעתיקה את ההוראה המאוחסנת בתא שכתובתו 0 אל אפיק הנתונים;
  המעבד מעביר את תוכן אפיק הנתונים אל אוגר ההוראות.



איור 1.22
קריאת הוראה מהזיכרון

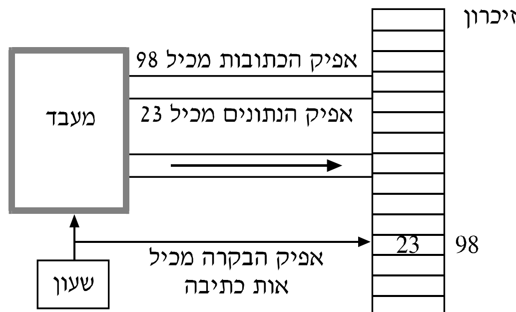
ב. מחזור כתיבה

לדוגמה, כדי לכתוב את הנתון 23 בתא שכתובתו 98 המעבד, מבצע את הפעולות האלה:

המעבד רושם באפיק הכתובות את הערך 98 שבו יאוחסן הנתון;

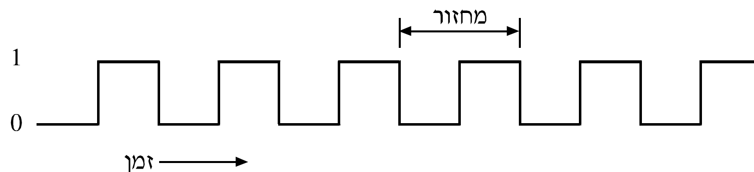
המעבד רושם באפיק הנתונים את הנתון 23;

המעבד רושם באפיק הבקרה כי הפעולה היא כתיבה (write).



איור 1.23
כתיבת נתון בזיכרון

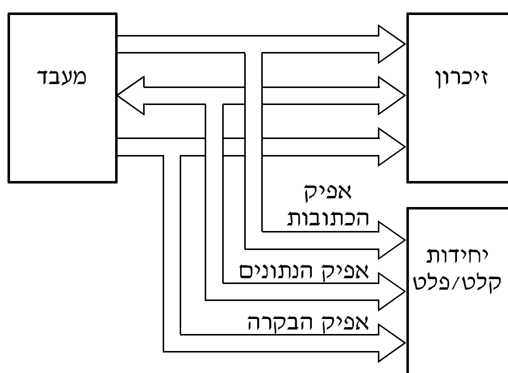
כל הפעולות הללו מתבצעות בזמנים שנקבעים על-ידי אותות שמפיק שעון פנימי שנמצא במעבד. תפקיד השעון הפנימי הוא לסנכרן את פעולתן של היחידות השונות של המחשב; בכך דומה תפקיד השעון לתפקידו של מנצח בתזמורת, הקובע בהינף שרביט מתי להתחיל ומתי להפסיק לנגן. כדי לבצע את הסנכרון, השעון מפיק אותות חשמליים המשתנים בצורה מחזורית בין רמת מתח גבוהה (לדוגמה 3V) לרמת מתח נמוכה של 0V. פעולה יכולה להתחיל, למשל, כאשר המתח עולה מרמת המתח הנמוכה לרמת המתח הגבוהה; שינויים אלה אפשר לתאר כגל ריבועי (ראו איור 1.24). בכל שנייה השעון מפיק מספר מחזורים גדול וככל שמספר מחזורי השעון גדול ביצוע התכנית קצר יותר. לדוגמה שעון המעבד 8086 (אותו נציג בהמשך) מפיק 5 מיליון מחזורים בשנייה ואילו שעון מעבד "פנטיום 4" מפיק 3.6 מיליארד (אלף מיליון) מחזורים בשנייה. אם נניח כי ביצוע הוראת חיבור (ADD) אורך 10 מחזורי שעון, אזי במעבד 8086 ניתן יהיה לבצע 500,000 הוראות חיבור בשנייה ואילו במעבד פנטיום 4 נוכל לבצע 360,000,000 הוראות חיבור בשנייה.



איור 1.24
אותות שעון

מחזור ההבאה-ביצוע, או ביצוע מחזור אפיק, יכול להימשך מספר מחזורי שעון. משך ביצוע ההוראה תלוי בארכיטקטורה של המעבד, וקיימים גורמים רבים שמשפיעים על זמן זה. אחד הגורמים שמשפיע על משך ביצוע ההוראה הוא סוג הפעולה. לדוגמה, זמן הביצוע הדרוש להוראת כפל הוא ארוך מאוד, בעוד שזמן הביצוע הדרוש לחיבור קצר בהרבה. גורם נוסף הוא פנייה לזיכרון, כדי לקרוא תוכן תא בזיכרון עליו מתבצעת הפעולה. לדוגמה: בהוראה כמו 01 02 07 המחברת בין שני אוגרים, אין צורך לפנות לזיכרון. במקרה כזה, זמן ביצוע ההוראה (כלומר, השלמת מחזור ההבאה-ביצוע) קצר יותר מאשר ביצוע של הוראה שבה אחד מהאופרנדים הוא תא בזיכרון. בדרך-כלל, כאשר ביצוע הוראה כרוך בכמה גישות לזיכרון, משך ביצוע ההוראה הוא ממושך. במהלך כתיבת התכנית יכול המתכנת לבחור הוראות שזמן ביצוען קצר יותר, וכך להשפיע על משך הביצוע של התכנית כולה. בנושא זה נרחיב את ההסבר בפרק השישי של הספר.

עד כה תיארנו כיצד מתבצע הקשר בין המעבד לזיכרון, אך כל מחשב מכיל יחידת קלט/פלט המחוברת גם היא למעבד באמצעות שלושת האפיקים. לכל התקן קלט/פלט יש כתובת ייחודית. כאשר כתובת מועברת באפיק הכתובות, היא מועברת באפיק הכתובות לזיכרון וגם להתקני הקלט/פלט. באפיק הבקרה המעבד מציין לא רק את סוג הפעולה אלא גם את סוג היחידה אליה הוא פונה: זיכרון או התקן קלט/פלט. בכל יחידה המחוברת למעבד קיים מעגל המפענח את הכתובת ומזהה אם היא הכתובת ששייכת לו אם לאו. כאשר מופק אות השעון המסנכרן את פעולת היחידות, היחידה שזיהתה שהכתובת באפיק הנתונים שייכת לה, מבצעת את הפעולה הנדרשת, ושאר היחידות אינן מגיבות. הדבר דומה לכיתה שבה יש מורה אחד והרבה תלמידים וכולם מאזינים לדברי המורה; כאשר המורה רוצה להפנות שאלה לתלמיד מסוים בכיתה, הוא מציין במפורש את שמו (כתובתו).



איור 1.25
חיבור יחידות המחשב לאפיקים

סיכום

התפתחות טכנולוגית יחד עם שיפור הארכיטקטורה של המחשב ופיתוח תוכנות, אפשר את השינוי הדרמטי שחל בתחום המחשוב ב-50 השנים האחרונות. אנו רואים כי בטכנולוגיה המגמה היא מצד אחד מזעור ומצד שני מהירויות עבודה גבוהות מאוד ומחיר זול יותר. מאפיינים אלה אפשרו ייצור המוני של מחשבים, ומשום-כך אפשר למצוא אותם כיום בכל בית (כמעט). השיפור בארכיטקטורה ובתוכנה דחף לפיתוח יישומים מורכבים יותר, עשירים בתמונות ובקול (מולטימדיה).

ישנן שאלות רבות המעניינות כיום אנשים שעוסקים בתחום המחשוב. לדוגמה:

- האם יימשך השיפור בטכנולוגיה, או שהגענו לגבול העליון בניצול החומרים מהם הטכנולוגיה בנויה?
- האם ניתן לשפר את הארכיטקטורה של פון-נוימן כדי לנצל את חומרת המחשב טוב יותר, או שהגיע הזמן לחשוב על ארכיטקטורה אחרת?
- בהתבסס על הטכנולוגיות הקיימות, אילו סוגי יישומים נוכל לפתח בעתיד, שלא פותחו עד היום?

בפרק האחרון של ספר זה ננסה לתאר את המעבדים המודרניים ואת המגבלות של ארכיטקטורת פון-נוימן עליה מבוססים מעבדים אלה.

ייצוג מידע במחשב

2.1 מבוא

המחשב מעבד יישומים רבים ומגוונים, לדוגמה: יישומים לעיבוד נתונים מספריים (עיבוד ציונים של תלמידי בית-ספר, חישובים ועיבוד נוסחאות בפיזיקה וכדומה), מעבדי תמלילים המטפלים בטקסטים, משחקים הכוללים הצגת אנימציות וקול, שידורי אינטרנט המעבירים קול ווידאו, ועוד. עלינו לזכור, שכל מגוון הנתונים ביישומים אלו מיוצג בסופו של דבר בייצוג שהמחשב מבין – הוא הייצוג הבינארי. **הייצוג הבינארי** מכיל רק שני סימנים (או שתי ספרות): אפס ואחד; המידע שאנו מייצגים, בין אם זה מספרים, טקסט, תמונה או קול, מומר למספרים המורכבים מרצף של 0 ו-1. לדוגמה: 100101 או 1111. הסיבה לשימוש בשיטת ייצוג זו היא: המחשב בנוי מרכיבים אלקטרוניים העשויים להימצא באחד משני מצבים אפשריים בלבד; כגון: "יש זרם ברכיב" (מצב שנהוג לסמנו כ-1) או "אין זרם ברכיב" (מצב שנהוג לסמנו כ-0).

כדי לבצע המרה משיטת ייצוג אחת לשיטה אחרת, משתמשים ב**כללי המרה**. לכל ערך המיוצג בשיטה הראשונה, מגדירים כללי ההמרה ערך שיתאים לשיטה האחרת. כללי ההמרה הם מגוונים. לעיתים ניתן לנסח אותם בעזרת נוסחאות ולעיתים בעזרת טבלאות תרגום או בשיטות אחרות. לדוגמה: כדי להמיר 1 מטר לסנטימטרים, עלינו להכפילו ב-100 וכדי להמיר 1 סנטימטר למטרים נצטרך לחלק ב-100; דוגמה אחרת: כדי להציג אות אלפבית באמצעות מספר, נשתמש בטבלה (למשל הגימטרייה של האות ריש היא 200).

אחד העקרונות החשובים בהמרה מייצוג מידע בתחום אחד לייצוג בתחום אחר, הוא **שימור המידע** – משמע, שני הייצוגים צריכים לתאר את אותו מידע. ובמילים אחרות, צריכה להיות התאמה חד-חד ערכית בין הייצוגים. כך לדוגמה, ניתן לרשום טמפרטורה בסולם צלסיוס ולהמירה לייצוג בסולם פרנהייט תוך כדי שימור מידע. אולם לא תמיד ההמרה משמרת מידע. לדוגמה, אפשר להציג ציון של תלמיד כמספר בין 0 ל-100 או באמצעות מילים מתוך קבוצה של מילים קבועות (למשל טוב מאוד, טוב וכדומה). אפשר להמיר לציון

מילולי כל ציון המבוטא כמספר, למשל, כל ציון בין 85 ל-95 יומר לציון "טוב מאוד", אבל המרה הפוכה לא תהיה מדויקת. כלומר, אם התלמיד קיבל ציון "טוב מאוד" לא נוכל לדעת אם הציון המספרי שקיבל הוא 89 או 93. לכן, בעת ביצוע המרה משיטת ייצוג אחת לשיטת ייצוג אחרת, עלינו לבדוק אם שימור המידע מתקיים, ואם אינו מתקיים עלינו לבדוק אם איבוד חלק מהמידע מפריע לביצוע המשימה.

בפרק זה נתאר כיצד ניתן להציג מידע בייצוג בינארי. נתאר את הייצוג של מספרים שלמים ומספרים לא-שלמים בשיטות הספירה המקובלות כיום. נתמקד בשיטה העשרונית, בשיטות המקובלות במחשבים, ונציג שיטות להמרה מייצוג לייצוג. כמו-כן נתאר שיטות לייצוג טקסט, תמונות וקול בייצוג בינארי. לסיום נציג את יחידות הזיכרון בהן מאוחסן המידע הבינארי.

2.2 ייצוג מספרים שלמים

בחיי היומיום אנו רגילים להשתמש במספרים המוצגים **בשיטה העשרונית.** ייצוג מספרים בשיטה העשרונית מבוסס על **שיטת הספירה המיקומית,** שפותחה על-ידי הבבלים בתקופה שבין 2000 ל-3000 לפנה"ס*. בשיטת ספירה זו משתמשים באוסף סימנים קבוע, המייצג ספרות, והערך של כל ספרה נקבע על-פי **מיקומה** במספר. במילים אחרות, לאחר שקובעים מספר כלשהו b בתור בסיס הספירה, מגדירים b סימנים בסיסיים עבור המספרים 0, 1, 2, ... b–1, שנקראים גם **ספרות** (digits). בעזרת הספרות הללו ניתן לייצג כל מספר שלם.
ערכו של מספר שבסיס הספירה שלו הוא b, שצורתו הכללית היא:

$$a_0a_{n-1}a_{n-2}\ldots a_0$$

יינתן על-ידי הנוסחה:

$$a_nb^n + a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots + a_0b^0$$

כאשר a_i היא אחת מספרות שיטת הספירה, כלומר $0 \le a_i < b$.

* מערכת המספרים הקדומה ביותר שהתפתחה במצרים העתיקה ב-3500 לפנה"ס הייתה **מערכת מספרים אדיטיבית (חיבורית).** במערכת מספרים אדיטיבית, מספר כלשהו b נבחר כבסיס הספירה ונוצרו סימנים בסיסיים עבור החזקות השונות של הבסיס, b^0, b^1, b^2, b^3 וכו'. לאחר מכן כל מספר נכתב על-ידי צירוף של הסימנים הבסיסיים, כאשר הערך של המספר נקבע לפי הסכום של ערכי הסימנים המרכיבים אותו. מידע נוסף ניתן למצוא באתר מטח: http://lib.cet.ac.il/pages/item.asp?item=7878

לדוגמה, ערכו של המספר העשרוני (בסיס ספירה 10) הבא: 9457362
הוא:

$$9 \times 10^6 + 4 \times 10^5 + 5 \times 10^4 + 7 \times 10^3 + 3 \times 10^2 + 6 \times 10^1 + 2 \times 10^0$$

או, במילים אחרות:

$$9000000 + 400000 + 50000 + 7000 + 300 + 60 + 2$$

שימו לב, שכל אחת מהספרות a_i היא אחת מהספרות העשרוניות, כלומר $0 \le a_i < 9$.

בסעיף זה נתאר את הייצוג של מספרים שלמים בארבע שיטות: בשיטה העשרונית, בשיטה הבינארית, בשיטה ההקסדצימלית (בסיס 16) ובשיטה האוקטלית (בסיס 8).

2.2.1 שיטת הספירה העשרונית

בשיטת הספירה העשרונית (בה אנו רגילים להשתמש), **בסיס הספירה** b הוא **10** וקיימות בה עשר **ספרות**:

0 1 2 3 4 5 6 7 8 9

כאשר כותבים, לדוגמה, את המספר 5827_{10}, הערך של כל אחת מהספרות במספר נקבע לפי המיקום שלה, כפי שמראה טבלה 2.1.

טבלה 2.1
ערך המיקום של הספרות במספר 5827_{10}

ערך המיקום	1	10	100	1000
ספרות המספר	7	2	8	5

הספרה 7 רשומה ראשונה בצד ימין של הטבלה; ערך המיקום שלה הוא 1 ולכן היא מציינת יחידות;
הספרה 2 רשומה משבצת אחת שמאלה; ערך המיקום שלה הוא 10 ולכן היא מציינת עשרות;
הספרה 8 רשומה משבצת נוספת שמאלה; ערך המיקום שלה הוא 100 ולכן היא מציינת מאות;

הספרה האחרונה היא 5; ערך המיקום שלה הוא 1000 ולכן היא מציינת אלפים.

מקובל להגדיר את הספרה בעלת ערך המיקום הגבוה ביותר **כספרה המשמעותית ביותר** במספר, ואת הספרה בעלת ערך המיקום הקטן ביותר **כספרה הפחות משמעותית**. לדוגמה במספר 5287, הספרה המשמעותית ביותר היא 5 (ערך המיקום שלה הוא 1000) והספרה הפחות משמעותית במספר היא 7 (ערך המיקום שלה 1).

נהוג למספר את מיקום הספרות במספר החל מ-0 (הספרה הפחות משמעותית). שיטה זו מאפשרת לרשום בצורה נוחה את ערך המיקום של ספרה כחזקה של 10:

$$5827_{10} = 5 \times 10^3 + 8 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$$

שימו לב, שהמספר 10 הכתוב בכתב קטן מימין למספר 5827, מציין שמדובר במספר בבסיס 10. באופן כללי, ברישום מספרים בשיטת ייצוג עשרונית משמיטים את רישום הבסיס, ולכן במקום לרשום 5827_{10} רושמים 5827. בכל שיטת ספירה אחרת מקובל לרשום את הבסיס ליד המספר, וכך נוכל להבחין בין מספר בשיטת ספירה עשרונית למספר בשיטת ספירה אחרת.

שיטת הספירה המיקומית מאפשרת להציג בדרך דומה גם מספרים לא שלמים, לדוגמה 0.001 או 293.46. כדי להבדיל בין החלק השלם והשבר במספר אנו מוסיפים את הנקודה העשרונית. כמו כן ניתן לרשום מספרים עם סימן על-ידי הוספת הסימן '+' לציין מספר חיובי והסימן '–' לציין מספר שלילי. מספרים עם סימן מכונים "מספרים מכוונים" (מלשון "כיוון").

חשבו: כיצד נקבע ערך המיקום של הספרות בשבר? כדי לענות על השאלה נרשום את המספר כסכום של מכפלת כל ספרה בערך המיקום שלה, לדוגמה:

$$0.461 = 4 \times 0.1 + 6 \times 0.01 + 1 \times 0.001$$

מדוגמה זו ניתן ללמוד כי ערך המיקום של ספרה בשבר היא חזקה שלילית של 10, ובהתאם: ערך המיקום של הספרה 4 הוא $0.1 = 10^{-1}$, ערך המיקום של הספרה 6 הוא $0.01 = 10^{-2}$, וערך המיקום של הספרה 1 הוא $0\ 001 = 10^{-3}$.

לסיכום: בשיטת ספירה מיקומית הבסיס הוא 10, ערך המיקום של מספר מבוטא כחזקה של 10, והמעריך של החזקה נקבע בהתאם למיקומה בספרה. בחלק השלם של המספר המעריך של החזקה 10 הוא חיובי, ובשבר המעריך של החזקה 10 הוא שלילי. במילים אחרות, מספר K בבסיס 10 יירשם בצורה הזו:

$$K_{10} = (a_{n-1}a_{n-2}...a_2a_1a_0a_{-1}a_{-2}...a_{-m})_{10} =$$
$$= a_{n-1}10^{n-1} + a_{n-2}10^{n-2} + ... + a_2 10^2 + a_1 10^1 + a_0 10^0 +$$
$$+ a_{-1}10^{-1} + a_{-2}10^{-2} + ... + a_{-m}10^{-m}$$

a_i מציינת ספרה במספר (האות i היא אינדקס המציין את מיקום הספרה בתוך המספר). בחלק השלם של המספר יש *n* ספרות (המסומנות על-ידי האינדקסים 0 עד n–1) ובשבר יש m ספרות (המסומנות על-ידי האינדקסים 1– עד m–). כדי לציין מספר עם סימן מוסיפים את הסימן '+' או '–'.

שאלה 2.1

מצאו את ערכה המיקומי של הספרה 7 במספרים העשרוניים האלה:

א. 4782 ב. 273589 ג. 408.87 ד. 0.02357

2.2.2 ייצוג בינארי של מספרים שלמים

ייצוג בינארי במחשב מבוסס על שיטת הספירה המיקומית, שבה **בסיס הספירה** הוא b = 2 והיא מכילה **שתי ספרות: 0 ו-1 בלבד**. לדוגמה, המספרים 101_2 ו-1011011000_2 מוצגים בשיטה הבינארית. בייצוג בינארי מכנים ספרה בשם **סיבית** (קיצור של המילים **סִ**פרה **בִּ**ינארי**ת**) ובאנגלית bit (קיצור של המילים **b**inary dig**it**). לדוגמה, במספר 1011_2 יש 4 סיביות (4 ביטים). כפי שציינו, כדי לציין שהמספר הוא מספר בינארי נהוג לרשום את הבסיס (במקרה זה: 2) ליד המספר, לדוגמה 101_2.

במספר בינארי, הערכים המיקומים של הספרות במספר הם חזקות של הבסיס b = 2. לדוגמה טבלה 2.2 מציגה את ערך המיקום של כל ספרה במספר 100101_2.

טבלה 2.2
הערכים המיקומיים של המספר 100101_2

מיקום הספרה במספר	0	1	2	3	4	5
ערך המיקום	2^0	2^1	2^2	2^3	2^4	2^5
ספרות המספר	1	0	1	0	0	1

שאלה 2.2

א. ציינו את הערך המיקומי של הספרות המודגשות:

$11\mathbf{0}10\mathbf{1}10_2$ $\mathbf{1}011\mathbf{011}_2$

ב. רשמו את הספרה המשמעותית ביותר ואת הספרה הפחות משמעותית בכל מספר.

עד כה הראינו כיצד מייצגים מספר בינארי שלם. כאשר רוצים לרשום מספר לא שלם או מספר עם סימן, בשיטת ייצוג מיקומית, צריכים להוסיף נקודה עשרונית או את הסימנים '+' או '–'. אולם, כפי שכבר ציינו, המחשב משתמש רק בשני סימנים, 0 ו-1, כדי לייצג מספר בינארי, והוא לא כולל נקודה עשרונית ולא סימן '–' או '+'. לייצוג מספרים לא שלמים במחשב משתמשים **בשיטת הנקודה הצפה** המתוארת בפרק זה בסעיף 2.3, ולייצוג מספרים עם סימן משתמשים **בשיטת המשלים לשניים** המתוארת בפרק הבא, בסעיף 3.2.

שאלה 2.3

כתבו את המספרים הבאים כסכום של חזקות:

א. 1101011_2 ב. 10001001_2

2.2.3 המרת מספרים שלמים ללא סימן מייצוג עשרוני לייצוג בינארי ולהיפך

בחיי היומיום אנו משתמשים בשיטה העשרונית לייצוג מספרים. לכן, כאשר צריך לחשב את ערכו של מספר המיוצג בשיטה אחרת, נוח לנו להמיר אותו למספר בשיטה העשרונית. לעומת זאת, כדי לייצג את המספר במחשב בשפת אסמבלי עלינו, לדעת לייצג אותו כמספר בינארי או בשיטה ההקסדצימלית (בסיס 16) שהיא, כפי שנראה בהמשך, נוחה יותר לשימוש.

בסעיף זה נתאר תחילה כיצד ניתן להמיר מספרים המיוצגים בשיטה הבינארית למספרים המיוצגים בשיטה העשרונית, ואחר כך נתאר את ההמרה ההפוכה – ממספרים המיוצגים בשיטה העשרונית למספרים המיוצגים בשיטה הבינארית. לביצוע ההמרה נשתמש בכמה כללים מתמטיים פשוטים, המבוססים על תיאור של מספר כסכום של מכפלת הספרות בערך המיקום. המרה זו היא המרה משמרת מידע – לכל ערך בשיטת ייצוג אחת יש ערך חד-חד ערכי בשיטת ייצוג שנייה.

א. המרה של מספר המיוצג בשיטה הבינארית למספר המיוצג בשיטה העשרונית

כדי להמיר מספר שלם K משיטת ייצוג בינארית לשיטה העשרונית נרשום את המספר כסכום מכפלת הספרות בערך המיקום ונחשב את הסכום, בצורה הזו:

$$K = a_{n-1}\cdot 2_{n-1} + a_{n-2}\cdot 2^{n-2} + \ldots + a_i\, 2^i + \ldots + a_1\cdot 2^1 + a_0\cdot 2^0$$

כאשר a_i היא ספרה (0 או 1) ו-i הוא האינדקס המציין את מיקומה במספר.

לדוגמה: נציג את המספר 100101_2 כסכום של חזקות של 2 ונבצע את החישוב בשיטה העשרונית. הסכום המתקבל הוא המספר בשיטת ייצוג עשרונית.

$$100101_2 = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 37$$

כלומר $37 = 100101_2$

שאלה 2.4

המירו את המספרים הבאים למספרים בשיטה העשרונית:

א. 10111_2 ב. 1001_2

שאלה 2.5

בְּדקו אם ביטויי היחס הבאים מתקיימים:

א. $1001_2 = 8$? ב. $110101_2 < 25$? ג. $1111_2 = 15$?

דרך אחרת לבצע את החישוב היא לרשום בטבלה, מעל לכל ספרה, את ערך המיקום שלה ולסכם את ערכי המיקום שבהם הספרה היא 1.

טבלה 2.3 מתארת דוגמה לרישום וחישוב ערך עשרוני של מספר בינארי:

טבלה 2.3
המרה של 100101_2 למספר עשרוני

מיקום הספרה	0	1	2	3	4	5
ערך המיקום	$2^0=1$	$2^1=2$	$2^2=4$	$2^3=8$	$2^4=16$	$2^5=32$
ספרות המספר 100101_2	1	0	1	0	0	1
המספר העשרוני	32+4+1 = 37					

כדי להשתמש בשיטה זו, רצוי לזכור בעל-פה את לוח החזקות של 2 שחלק ממנו מוצג בטבלה 2.4.

טבלה 2.4
ערכי החזקות של 2

החזקה	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
הערך	1	2	4	8	16	32	64	128	256	512	1024

שאלה 2.6

א. השתמשו בטבלה 2.4 כדי להמיר את המספרים הבינאריים הבאים למספרים המיוצגים בשיטה העשרונית:

i. 11111111_2 ii. 1011011_2 iii. 1001_2

ב. השתמשו בשיטה הבאה להמיר מספר בינארי למספר המיוצג בשיטה העשרונית:

$$\mathbf{101101}_2 = (((((\mathbf{1} \times 2 + \mathbf{0}) \times 2) + \mathbf{1}) \times 2 + \mathbf{1}) \times 2 + \mathbf{0}) \times 2 + \mathbf{1}$$

כדי לבצע שוב את החישובים שבצעתם בסעיף א.ii ו-א.iii

ב. המרה של מספר המיוצג בשיטה העשרונית למספר בשיטה הבינארית

נתאר שתי שיטות להמרה של מספר עשרוני למספר בייצוג בינארי.

1. שיטה ראשונה

בשיטה זו תהליך ההמרה מתחיל במציאת הספרה המשמעותית ביותר במספר המתורגם. לשם כך נשתמש בטבלה 2.4 שהובאה לעיל. לפני שנציג אלגוריתם לביצוע ההמרה נביא תחילה דוגמה לאופן החישוב.

כדוגמה, נמיר את המספר 82 למספר בייצוג בינארי.

א. תחילה נחפש את החזקה הגבוהה ביותר של 2 שבה ערך המיקום עדיין קטן מ-82. לשם כך ניעזר בטבלה 2.4 ובה נמצא כי חזקה זו היא $2^6 = 64$ והיא קובעת את ערך המיקום של הספרה המשמעותית ביותר במספר שהוא:

$$1 \times 2^6 = 1000000_2$$

נחסיר את ערך המיקום שמצאנו מהמספר $82 - 64 = 18$, והתוצאה המתקבלת היא מספר עשרוני 18 שיש להמירו למספר בינארי.

ב. נחזור על הפעולה הקודמת ונחפש את החזקה הגבוהה של 2 שבה ערך המיקום עדיין קטן מ-8; מעריך חזקה זו הוא 4, לכן ערך המיקום של ספרה זו יהיה $16 = 2^4$

$$1 \times 2^4 = 10000_2$$

כעת הערך שנותר להציג הוא $18 - 16 = 2$.

ג. נחזור על התהליך; נחפש את החזקה הגבוהה של 2 שבה ערך המיקום קטן מ-2. מעריך חזקה זו הוא 1, לכן ערך המיקום של הספרה הוא 2^1:

$$1 \times 2^1 = 10_2$$

הערך שנותר להציג הוא $2 - 2 = 0$, לכן תהליך החישוב הסתיים.

ד. לסיום, נסכם את כל ערכי המיקום שקיבלנו:

$$\begin{array}{r} 1000000 \\ 10000 \\ +\quad 10 \\ \hline 1010010_2 = 82 \end{array}$$

כדי לבדוק שהחישוב שביצענו הוא נכון, נמיר את המספר הבינארי שקיבלנו בחזרה למספר עשרוני:

טבלה 2.5
המרת המספר 1010010_2 למספר עשרוני

מיקום הספרה	0	1	2	3	4	5	6
ערך המיקום	1	2	4	8	16	32	64
ספרות המספר 100101_2	0	1	0	0	1	0	1
המספר העשרוני	64+16+2= 82						

נתאר אלגוריתם לשימוש בשיטה הזו:

האלגוריתם מקבל מספר עשרוני N שלם, לא שלילי, וממיר אותו למספר בינארי M.

שים ב-M את הערך 0
כל עוד המספר N גדול מ-0 בצע את הפעולות הבאות:
מצא את ערך המיקום הגבוה ביותר של 2 שקטן או שווה ל-N
הוסף את ערך המיקום למספר M
הפחת מ-N את ערך המיקום שמצאת

2. שיטה שנייה

בשיטה זו ההמרה של מספר עשרוני למספר בינארי מבוססת על תהליך איטראטיבי שבו אנו מחלקים את המספר העשרוני ב-2, ובכל שלב אנו רושמים את השארית כספרה במספר הבינארי המבוקש (החל מהספרה הפחות משמעותית); המנה המתקבלת היא הבסיס לחלוקה החוזרת. התהליך מסתיים כאשר המנה היא 0.

לדוגמה, נתאר את תהליך ההמרה של המספר 84 למספר בינארי (בסוגריים אנו מציינים את השארית):

$82:2 = 41\ (0) \qquad a_0 = 0$
$41:2 = 20\ (1) \qquad a_1 = 1$
$20:2 = 10\ (0) \qquad a_2 = 0$
$10:2 = 5\ (0) \qquad a_3 = 0$
$5:2 = 2\ (1) \qquad a_4 = 1$
$2:2 = 2\ (0) \qquad a_5 = 0$
$1:2 = 0\ (1) \qquad a_6 = 1$

↑
התהליך מסתיים

המספר הבינארי שהתקבל הוא: 1010010_2.

שאלה 2.7

המירו את המספרים העשרוניים הבאים למספרים בינאריים; השתמשו בשתי השיטות שהוצגו:

א. 29 ב. 83 ג. 196

לסיום נדגיש כי המרת מספרים שלמים, ללא סימן, מייצוג עשרוני לייצוג בינארי ולהיפך, היא משמרת מידע. כלומר, לכל מספר בשיטה העשרונית יש התאמה חד-חד ערכית לשיטה הבינארית.

2.2.4 ייצוג מספרים שלמים ללא סימן בשיטה האוקטלית ובשיטה ההקסדצימלית

שיטת הספירה הבינארית היא דלה בסימנים, לכן דרושות, לעיתים קרובות, ספרות רבות (שכולן 0 או 1 כמובן) כדי לרשום מספרים בינאריים, גם כאשר ערכם העשרוני של המספרים אינו גדול. לדוגמה: המספר העשרוני 773 (שהוא בן 3 ספרות) מיוצג בשיטה הבינארית כ-1100000101_2. הוא כולל 10 סיביות!

רצף ארוך המורכב מ-0 ו-1 גורם לעיתים קרובות טעויות אנוש ברישום ובקריאה של המספרים הללו. כדי להקטין את אורך המספרים, משתמשים בשיטה ההקסדצימלית (בבסיס 16) או בשיטה האוקטלית (בבסיס 8). הבחירה בשיטות ייצוג אלה אינה מקרית והיא נובעת מכך שהמרה משיטות אלה לייצוג בינארי ובחזרה היא פשוטה ומהירה מאוד. משום כך, בתכניות הכתובות בשפת אסמבלי, נהוג לרשום מספרים בשיטה ההקסדצימלית במקום בשיטה הבינארית. בסעיף זה נציג את שיטות הספירה ההקסדצימלית והאוקטלית, שתיהן שיטות ספירה מיקומיות, ונתאר את אופן ההמרה משיטות אלה לשיטה הבינארית ולשיטה העשרונית.

בשיטה האוקטלית, הבסיס הוא 8, כלומר, יש בה 8 ספרות:

0 1 2 3 4 5 6 7

לדוגמה, הנה מספרים בשיטה זו: 416741_8 , 1002_8

בשיטה ההקסדצימלית הבסיס הוא 16. כדי לרשום מספר בשיטה זו, אנו זקוקים ל-16 ספרות. כיוון שאנו רגילים להשתמש במערכת הכוללת 10 ספרות, נצטרך להוסיף שש ספרות, אותן נהוג לרשום כאותיות של האלפבית האנגלי:

ספרות הקסדצימליות	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	↑	↑	↑	↑	↑	↑										
ערך עשרוני	15	14	13	12	11	10										

לדוגמה, נציג מספרים בבסיס הקסדצימלי:

123_{16} , $A01F_{16}$, FF_{16}

שימו לב, שהייצוג של המספר 16, בשיטה ההקסדצימלית, הוא 10_{16}.

שאלה 2.8

א. מהו ערך המיקום של הספרה 7 במספרים הבאים: 107568, A071B16

ב. חשבו, מה הערך העשרוני של 10_8?

א. המרה מהשיטה האוקטלית וההקסדצימלית לשיטה העשרונית

כמו בכל שיטת ספירה מיקומית, ניתן לרשום מספר בשיטה האוקטלית ובשיטה הקסדצימלית כסכום של מכפלת הספרה בערך המיקום, כאשר ערך המיקום מיוצג כחזקה של הבסיס:

$$K_8 = a_{n-1} \cdot 8^{n-1} + a_{n-2} \cdot 8^{n-2} + \ldots + a_2 \cdot 8^2 + a_1 \cdot 8^1 + a_0 \cdot 8^0$$

ובהתאם, ובשיטה ההקסדצימלית נרשום:

$$K_{16} = a_{n-1} \cdot 16^{n-1} + a_{n-2} \cdot 16^{n-2} + \ldots + a_2 \cdot 16^2 + a_1 \cdot 16^1 + a_0 \cdot 16^0$$

נשתמש ברישום זה כדי להמיר למספר עשרוני את המספרים המיוצגים בשיטה האוקטלית או בשיטה ההקסדצימלית; שימו לב, שהחישוב מימין לסימן השיוויון הראשון הוא כולו בשיטה העשרונית. להלן כמה דוגמאות:

1. $137_8 = 1\times8^2+3\times8^1+7\times8^0 = 1\times64+3\times8+7\times1 = 95$
2. $5067_8 = 5\times8^3+0\times8^2+6\times8^1+7\times8^0 = 5\times512+0+6\times8+7\times1 = 2615$
3. $1AF_{16} = 1\times16^2+A\times16^1+F\times16^0 = 1\times256+10\times16+15\times1+7\times1 = 431$
4. $A6B2_{16} = 10\times16^3+6\times16^2+11\times16^1+2\times16^0 = 10\times4096+6\times256+$
   $+11\times16+2\times1 = 42673$

שימו לב, בהמרה מהשיטה ההקסדצימלית, אנו משתמשים בערך העשרוני של הספרה לביצוע סכום החזקות. כך לדוגמה, במספר $1AF_{16}$ השתמשנו ב-10 במקום הספרה A וב-15 במקום הספרה F.

ב. המרה מהשיטה העשרונית לשיטה האוקטלית ולשיטה ההקסדצימלית

המרת מספר עשרוני למספר בשיטה ההקסדצימלית או למספר בשיטה האוקטלית מתבצעת בתהליך איטראטיבי, הדומה לתהליך ההמרה של מספר בינארי למספר עשרוני, שתיארנו בסעיף הקודם: אנו מחלקים את המספר העשרוני בבסיס של המספר שאנו רוצים להמיר, ובכל שלב אנו רושמים את השארית כספרה במספר המבוקש (החל מהספרה הפחות משמעותית); המנה המתקבלת היא הבסיס לחלוקה החוזרת באיטרציה הבאה. התהליך מסתיים כאשר המנה היא 0.

דוגמה 2.1 המרת 2615 למספר אוקטלי

בתהליך איטראטיבי נחזור ונחלק את המספר ב-8

2615:8 = 326(7) $a_0 = 7$

326:8 = 40(6) $a_1 = 6$

40:8 = 5(0) $a_2 = 0$

5:8 = 0(5) $a_3 = 5$

↑

סיום התהליך

המספר המתקבל הוא 5067_8.

דוגמה 2.2 – המרת 2615 למספר הקסדצימלי

2615:16 = 163 (7) $a_0 = 7$

163:16 = 10 (3) $a_1 = 3$

10:16 = 0 (10) $a_2 = A$

↑

סיום התהליך

המספר שהתקבל הוא $A37_{16}$

שימו לב: לרישום הספרות בשיטה ההקסדצימלית עלינו להשתמש בייצוג A במקום המספר 10.

שאלה 2.9

א. המירו את המספרים הבאים למספרים בשיטה העשרונית: 1. 105328 2. 10AD16

ב. המירו את המספרים 1956 ו-431 למספרים בשיטה האוקטלית ובשיטה ההקסדצימלית.

ג. המרה מהשיטה הבינארית לשיטה האוקטלית

כדי לכתוב מספרים בינאריים בצורה נוחה יותר, ניזכר כיצד מתגברים על קריאת מספרים עשרוניים גדולים, כגון:

$2^{64} = 18556744073709551616$

בשיטה העשרונית אנו מפרידים את הספרות של המספר, מימין לשמאל, לקבוצות של שלוש ספרות וקוראים כל "שְלָשָה" כזו בנפרד:

18,446,744,073,709,551,616

נוכל לנהוג בצורה דומה גם לגבי מספרים בינאריים. נניח שכתבנו מספר בינארי כך:

$M = 010,110,000,101,111,001_2$

הפרדנו את ספרותיו מימין לשמאל לשלשות, ולמען הסדר הטוב השלמנו את השלשה השמאלית על-ידי הוספת אפס בצד שמאל. אם נתבונן בטבלה 2.6, נוכל לראות כי במספר M, הרשום לעיל, כל קבוצה של שלוש ספרות בינאריות מייצגת מספר בין 0 ל-7, ולכן נוכל לכתוב את המספר הבינארי בצורה מקוצרת, בעזרת הספרות 0 – 7. אולם ייצוג מספרים בהם משתמשים רק בספרות 0 עד 7, הוא בבסיס 8 (יצוג אוקטלי). ניתן לראות כי הערך המיקומי של כל ספרה אוקטלית גדול פי 8 מזה של הספרה הנמצאת מימינה; כמו כן, הזזה של ספרה בינארית שלושה מקומות שמאלה, מגדילה את ערכה פי 2^3 שהם 8.

טבלה 2.6
ייצוג בינארי של ערכי הספרות בבסיס 8 ובבסיס 16

בסיס 2	בסיס 8	בסיס 16
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

בסיס 2	בסיס 8	בסיס 16
1000		8
1001		9
1010		A
1011		B
1100		C
1101		D
1110		E
1111		F

כדי להציג את המספר הבינארי בבסיס 8, נמיר כל 3 סיביות (למשל בעזרת חלק מטבלה 2.6) לספרה אוקטאלית. המספר M ייכתב בצורה המקוצרת כך:

$M = 010,110,000,101,111,001_2 = 260571_8$

אורכו של המספר M בבסיס 8 הוא רק 6 ספרות (במקום 17 ספרות במספר הבינארי).

שאלה 2.10

המירו את המספרים הבינאריים הבאים למספרים בשיטה האוקטלית ובשיטה העשרונית:

א. 101101101_2 ב. 10001000110_2

שאלה 2.11

המירו את המספרים האוקטליים הבאים למספרים בשיטה הבינארית ובשיטה העשרונית:

א. 6302_8 ב. 14501_8

ה. המרה מהשיטה הבינארית לשיטה הקסדצימלית

הצגה אוקטלית מקצרת את אורך המספרים הבינאריים פי שלושה בערך. אם נרצה לקצרם עוד יותר, נוכל להשתמש בבסיס הספרה 16, כאשר כל ארבע ספרות בינאריות תיוצגנה על-ידי ספרה אחת בבסיס 16. כדי להמיר 4 ספרות בינאריות לספרה הקסדצימלית נשתמש בטבלה 2.6.

לדוגמה:

המספר הבינארי $0110,1010,0011,1100,1000_2$ ייוצג בשיטה ההקסדצימלית על-ידי המספר $6A3C8_{16}$.

והמספר $AB48_{16}$ יומר למספר הבינארי 1010101101001000_2

לסיכום, ננסח בצורה נוספת את ייחודם של הבסיסים 8 ו-16 ביחס לבסיס הבינארי: תרגום הייצוג האוקטלי וההקסדצימלי לייצוג בינארי, ולהיפך, מצטיין בפשטות ונוחות רבה. התרגום מתבסס על חלוקת מספר בינארי לקבוצות של 3 או 4 ספרות והצבה ישירה של הייצוג האוקטלי או ההקסדצימלי בהתאמה, במקום כל קבוצה של סיביות. שיטות ספירה אחרות, שבסיסן אינו חזקה של 2 (למשל 9 או 13) אינן מצטיינות בתכונה זו, ולכן התרגום לשיטות אלה מסובך יותר.

שאלה 2.12

א. כתבו את המספר 1985 בשיטה הבינארית, על-ידי הצגתו תחילה בשיטה האוקטלית.

ב. כתבו את המספר הבינארי 110101110101_2 בשיטה העשרונית על-ידי כתיבתו תחילה בשיטה האוקטלית. הציגו מספר זה גם בשיטה ההקסדצימלית.

ג. כתבו את המספר ההקסדצימלי $3E1B_{16}$ בשיטה הבינארית.

שאלה 2.13

השלימו את הטבלה הבאה:

בסיס עשרוני	בסיס בינארי	בסיס אוקטלי	בסיס הקסדצימלי
492			
	100101110101		
		721	
			2D4F

שאלה 2.14

א. נתון מספר הכתוב בשיטה האוקטלית. כיצד תדעו אם הוא מתחלק ב-2, ב-4 או ב-8 ללא שארית?

ב. כיצד ניתן לדעת אם מספר בינארי הוא זוגי או אי-זוגי?

שאלה 2.15

כתבו אלגוריתם הממיר מספר עשרוני שלם וחיובי למספר בינארי. ממשו את האלגוריתם בשפה עילית שאתם מכירים.

שאלה 2.16

כתבו אלגוריתם הממיר מספר בשיטת ספירה אחת למספר בשיטת ספירה אחרת על-ידי שימוש בהמרת ביניים לשיטת הספירה העשרונית. ממשו את האלגוריתם בשפה עילית שאתם מכירים.

שאלה 2.17

כתבו אלגוריתם הממיר מספר בינארי למספר בשיטת ייצוג אוקטלית והקסדצימלית. ממשו את האלגוריתם בשפה עילית שאתם מכירים.

2.3 ייצוג מספרים ממשיים

2.3.1 ייצוג מספרים ממשיים בשיטה ספירה מיקומית

ראינו כי בשיטת ספירה מיקומית משתמשים בנקודה כדי לציין מספר ממשי המכיל חלק שלם ושבר. לדוגמה: 1206.9023 או 0.0617. באופן דומה נרשום מספרים ממשיים בשיטה הבינארית: 100101.01101_2 או 1001.1101_2. כדי להמיר מספר ממשי המיוצג בשיטה הבינארית למספר עשרוני, נשתמש בייצוג של מספר כסכום מכפלת הספרות בערכי המיקום. לדוגמה, נמיר מספר ממשי המיוצג בשיטה הבינארית למספר עשרוני:

$$100101.01101_2 =$$
$$1\times2^5 + 0\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} + 0\times2^{-4} + 1\times2^{-5}$$
$$= 32 + 4 + 1 + 0.25 + 0.125 + 0.03125 = 37.40625$$

המרת מספר עשרוני למספר בינארי

כדי להמיר ממספר עשרוני למספר בינארי נטפל בכל חלק בנפרד:

– את החלק השלם נמיר בשיטה שתיארנו (בסעיף 2.2) על-ידי חלוקה ב-2 המבוצעת עד שהמספר מתאפס

– את השבר נכפיל ב-2. אם תוצאת הכפל היא שבר:
  - נרשום את החלק השלם כספרה בחלק השבור של המספר הבינארי אותו אנו מחשבים (החל משמאל לימין);
  - נמשיך להכפיל את השבר.

– תהליך זה יסתיים כאשר תוצאת הכפל היא מספר שלם ללא שבר.

לדוגמה, נמיר את המספר 37.40625 למספר בינארי.

א. כדי להמיר את הערך השלם 37, נרשום את ערכי המיקום:

ערך המיקום	1	2	4	8	16	32
המספר הבינארי	1	0	1	0	0	1

ב. כדי להמיר את השבר 0.40625, נכפיל ב-2:

הספרה	תהליך הכפל
0	$0.40625 \times 2 = 0.8125$
1	$0.8125 \times 2 = 1.625$
1	$0.625 \times 2 = 1.25$
0	$0.25 \times 2 = 0.5$
1	$0.5 \times 2 = 1.0$

↑ סיום התהליך

ג. כעת נרשום את המספר הבינארי: 100101.01101_2

אולם בניגוד למספרים שלמים, ייצוג מספר ממשי לא תמיד משמר מידע. לדוגמה, כדי להמיר את הערך העשרוני 0.2 למספר בינארי, נחזור על תהליך ההכפלה שתיארנו קודם לכן:

הספרה	תהליך החלוקה
0	$0.2 \times 2 = 0.4$
0	$0.4 \times 2 = 0.8$
1	$0.8 \times 2 = 1.6$
0	$0.6 \times 2 = 1.2$
בשלב זה אנו חוזרים לערך העשרוני המקורי, 0	$0.2 \times 2 = 0.4$

ניתן לראות כי התהליך הוא מחזורי ואינסופי. מכאן אנו מסיקים כי לא ניתן לייצג את 0.2 באופן מדויק בשיטה הבינארית. במקרה כזה נחליט על הדיוק הרצוי ונעצור את תהליך ההמרה בשלב זה. הדבר דומה לייצוג של שברים מסוימים או מספרים מעורבים מסוימים, ובכלל זה מספרים לא-רציונאליים, בשיטה העשרונית.

לדוגמה: ייצוג של השבר $\frac{1}{3}$ כשבר עשרוני: $0.333...$ או של המספר $\Pi \approx 3.14159$.

במקרים רבים משתמשים במספרים ממשיים כדי להציג, בצורה נוחה לקריאה, מספרים גדולים מאוד או קטנים מאוד (בערכם המוחלט). לדוגמה: מהירות האור היא בקירוב 300,000,000 מטר לשנייה. רישום מספר זה דורש ספרות רבות. אחת השיטות המקובלות במתמטיקה היא רישום מספר בצורה מעריכית. בדרך זו ניתן לרשום כי מהירות האור שווה

ל-3.0×10^8 מטר לשנייה. כבסיס לחזקה משמש אותו בסיס לפיו מיוצג המספר (בדוגמה זו הבסיס הוא 10). להצגה מעריכית של מספר ממשי יש חשיבות גדולה מאוד במחשבים, מפני שחשוב מאוד שמספר הספרות של המספר המיוצג יהיה קטן ככל האפשר, כדי לחסוך, למשל מקום בזיכרונם של מחשבים.

2.3.2 ייצוג מספרים ממשיים בשיטת הנקודה הקבועה

עדיין לא ענינו על השאלה: כיצד נציג מספרים ממשיים במחשב המשתמש בשני סימנים בלבד? אחת השיטות הפשוטות ביותר לייצוג מספר ממשי במחשב, היא להשתמש בשיטת **הנקודה הקבועה** (fixed point representation). לפי שיטה זו להצגת המספר מוגדרות n סיביות ומיקום הנקודה העשרונית הוא קבוע. לדוגמה: נחליט כי אנו מקצים לשבר שתי ספרות ולכן המיקום של הנקודה העשרונית יהיה בין הספרה השנייה לשלישית, ובהתאם מספר הספרות שמוקצה לחלק השלם יהיה n–2. המיקום של הנקודה העשרונית הוא לוגי, לכן איננו חייבים לרשום אותה. בשיטה זו, המספר 123502, לדוגמה, מייצג את המספר 1235.02, ובאופן דומה, בשיטה הבינארית, המספר 10011_2 מייצג את המספר 100.11_2, ואילו המספר 11100_2 מייצג את המספר 111.00_2.

לגישה זו יש כמה חסרונות. אחד מהם הוא איבוד מידע. נדגים כמה מן החסרונות בעזרת הדוגמה הבאה: בשיטת הנקודה הקבועה, שבה מוקצים שני מקומות לשבר, הן ייצוגו של המספר העשרוני 0.0014 והן ייצוגו של המספר העשרוני 0.006 יהיה 0.01. יתרה מזאת, גם הייצוג של 0.004 ושל 0.007– יהיה זהה ל-0.00, וזאת למרות שהסימנים של שני המספרים מנוגדים. כלומר, השימוש במספר ספרות קבוע, המוקצה לחלק השלם, לא מאפשר לייצג מספרים קטנים מאוד או מספרים גדולים מאוד. לדוגמה, בייצוג בשיטת הנקודה הקבוע בה מספר המקומות המוקצים לחלק השלם הוא 5, לא נוכל לייצג מספרים הגדולים מ-99999.

שאלה 2.18

יש לכתוב תכנית המחשבת סכום מכירות בחנות. מחיר פריט מיוצג בשקלים, לדוגמה: 99.99 או 50. הפריט היקר ביותר מחירו 1000 ₪. רשמו תבנית לייצוג של מחיר בשיטת הנקודה הקבועה. ציינו את מיקום הנקודה העשרונית ותנו שתי דוגמאות של מספרים המיוצגים בתבנית שיצרתם.

2.3.3 ייצוג מספרים ממשיים בשיטת הנקודה הצפה

שיטה אחרת לייצוג מספרים ממשיים, המוכרת לנו ממתמטיקה, היא **שיטת הנקודה הצפה** (floating point representation). בשיטה זו, הנקודה "צפה" או זזה, ומקומה נקבע לפי מעריך החזקה. כל מספר בצורה זו מורכב ממקדם ומחזקה של בסיס הספירה שבו כתוב המספר. מעריך חזקה זו ייקרא להלן **המציין**. למשל מהירות האור, 300,000,000 מטר לשנייה, יכולה להירשם באחת מהדרכים האלה:

3.0×10^8 מטר לשנייה
30.0×10^7 מטר לשנייה
0.3×10^9 מטר לשנייה

מימוש שיטת הנקודה הצפה מיושם במחשב בהתאם לתקנים שנקבעו כדי לאפשר אחידות של ייצוג במחשבים המיוצרים על-ידי חברות שונות. התקן* מגדיר את מספר הספרות המוקצות למעריך ולמקדם ואת מיקומן במספר המיוצג במבנה $\pm N\times2^k$, כאשר N הוא המקדם ולפניו סימן המסמן אם המספר חיובי או שלילי ו-k הוא המעריך. בנוסף, כדי לשמור על אחידות, נקבע בתקן כי N, הנקרא **מנטיסה**, יירשם בצורה מנורמלת, כלומר: ערכו של N תמיד יהיה גדול או שווה ל-1_2 וקטן מ-10_2 (כלומר, בשיטה העשרונית: $1 \le N < 2$).

לדוגמה (שימו לב, שאת החזקה כולה אנחנו רושמים, למען הנוחות, בשיטה העשרונית):

ייצוג של המספר הבינארי 101×2^2 יהיה 1.01×2^4
וייצוג של 0.00101 יהיה 1.01×2^{-3}.

אחד התקנים הסטנדרטיים מגדיר ייצוג של מספרים באמצעות מילה בת 32 סיביות. המספר מורכב משלושה חלקים:

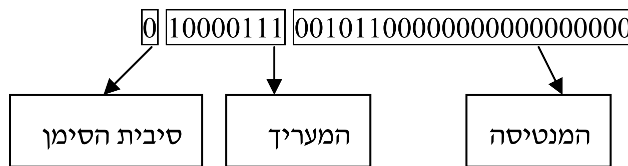
מנטיסה	מעריך	סימן
23 [22–00]	8 [30–23]	1 [31]

* התקן הנפוץ ביותר הוא IEEE Standard 754 floating point המגדיר שני סוגים של מספרים ממשיים בשיטת נקודה צפה: דיוק רגיל (32-bit) ודיוק כפול (64-bit)

כאשר:

- סיבית 31 היא סיבית הסימן של המנטיסה;
- 8 הסיביות הבאות (מסיבית 30 עד סיבית 23) הן המעריך;
- שאר 23 הסיביות (מסיבית 22 עד סיבית 0) הן המנטיסה.

לדוגמה, נתבונן במספר $01000011100101100000000000000000_2$ ונציין את חלקיו:



בתקן זה נקבע כי:

- אם סיבית הסימן (סיבית 31) היא 1 – המספר שלילי ואם סיבית הסימן היא 0 – המספר חיובי.
- ייצוג של מעריך יהיה בן 8 סיביות (הסיביות 23-30) ובתחום שבין 126– ל-127. כך, המספר הכי גדול שנוכל להציג יהיה בעל החזקה 2^{127} והמספר הכי קטן יהיה בעל החזקה 2^{-126}. כדי לאפשר להציג את המעריכים בתחום ערכים זה, משתמשים בשיטת ההטיה (biasing) כאשר ערך ההטיה הוא 127, כלומר, מוסיפים לערך המעריך את המספר 127 ואת התוצאה ממירים לייצוג בינארי.

  הנה דוגמה:

  כדי לייצג את המעריך 5 נחשב: $5 + 127 = 132 = 10000101_2$

  וכדי לייצג את המעריך 5– נחשב: $-5 + 127 = 122 = 01111010_2$

  חשבו, כיצד מיוצג המעריך 126– ? והמעריך 127?

- ייצוג של מנטיסה הוא ייצוג מנורמל שבו מוצג המספר בצורה הזו: $1 + f$, כאשר $0 \le f < 1$. לדוגמה: 1.00101, 1.11101 וכדומה. כלומר, כדי להציג מנטיסה, עלינו לתרגם את המספר למספר בינארי ולנרמל אותו כך שיתאים לייצוג הדרוש.

  לדוגמה:

כדי להציג את המספר 17, נתרגם אותו למספר בינארי 10001_2 ונציגו בצורה מנורמלת, כלומר:

$$10001_2 = 1.0001_2 \times 2^4$$

להלן דוגמאות המציגות מספרים בינאריים בצורה מנורמלת:

המספר הבינארי 1100 יוצג כך 1.1×2^3

המספר הבינארי 10 יוצג כך 1.0×2^1

המספר הבינארי 11100.001 יוצג כך 1.1100001×2^4

למעשה בייצוג מספר בצורה מנורמלת, אנו יכולים להניח כי הספרה הראשונה היא תמיד 1 (חוץ מהמספר אפס אותו נציג בהמשך) ואחריה יוצג שבר, לכן אין צורך לרשום את הספרה הראשונה במנטיסה ולהניח את קיומה. לדוגמה במקום להציג 17 בצורה מנורמלת כ-1.0001_2, נרשום רק $.0001_2$. כך ניתן לחסוך סיבית נוספת בייצוג של המנטיסה.

הנה כמה דוגמאות נוספות לייצוג של מספרים בשיטה זו:

א. המספר העשרוני 85.75 כמספר בשיטת הנקודה הצפה:

תחילה נמיר את הערך 85.75 למספר בינארי ונקבל:

$$85.75 = 1010101.11_2$$

נציג את המספר בצורה מנורמלת: $1010101.11_2 = 1.01010111_2 \times 2^6$. אנו יכולים להסתפק בשבר ולרשום את המנטיסה כך: $.01010111_2$.

כמו כן מצאנו כי המעריך הוא 6, לו נוסיף 127 ונקבל 133 או בייצוג בינארי $.10000101_2$

כיוון שהמספר חיובי, נרשום את סיבית הסימן כ-0.

כעת נוכל לרשום את המספר בייצוג בינארי ובייצוג הקסדצימלי:

$$01000010101010111000000000000000_2 = 42AB8000_{16}$$

ב. נציג את המספר -10000 בשיטת הנקודה הצפה:

נתרגם את המספר 100 לייצוג בינארי $10000 = 1001110001_2$

נציג את המספר בצורה מנורמלת: $1.001110001_2 \times 2^{13}$. אנו יכולים להסתפק בשבר ולרשום את המנטיסה כך: $.01010111_2$.

כמו כן מצאנו כי המעריך הוא 13, לו נוסיף 127 ונקבל 140 או בייצוג בינארי $.10001100_2$

לסימון שהמספר שלילי נרשום בביט הסימן את הערך 1.

לסיכום נרשום את המספר כמספר בינארי וכמספר הקסדצימלי:

$$110001100\ \ 00111000100000000000000_2 = C61C4000_{16}$$

נסכם את תהליך הנרמול של מספר ממשי השונה מאפס:

א. המר את המספר לייצוג בינארי;
ב. נרמל את המספר לצורה 1.f, המנטיסה תהיה הערך f;
ג. חשב את המעריך;
ד. הוסף 127.
ה. רשום את סיבית הסימן בהתאם לערך המספר (חיובי או שלילי).

לסיום, נראה כיצד מוצגים ערכים מיוחדים, כמו אפס ואינסוף:

אֶפֶס מוצג כאשר כל הסיביות של המעריך ושל המנטיסה הן 0; אם סיבית הסימן היא 0 נקבל 0+ וכאשר סיבית הסימן 1 נקבל 0–.

ייצוג הערך ∞ מתבצע כאשר כל סיביות המעריך הן 1 וכל סיביות המנטיסה הן 0. במקרה כזה, אם סיבית הסימן היא 0 נקבל ∞+ ואם סיבית הסימן היא 1 נקבל ∞–.

תחום הערכים שניתן להציג בשיטת הנקודה הצפה הם:

$$(2-2^{-23}) \times 2^{127} \quad - \quad \pm 2^{-126}$$

ובייצוג עשרוני אלה המספרים בתחום:

$$\sim 10^{38.53} \quad - \quad \pm \sim 10^{-44.85}$$

כמו כן צריך לזכור כי לא ניתן להציג מספרים ממשיים בצורה מדויקת, לכן עלינו להחליט בתכנית מהי רמת הדיוק הרצויה, ולקבוע את מספר הספרות אחרי הנקודה.

2.4 ייצוג טקסט

לייצוג טקסט בשפה טבעית, למשל עברית, משתמשים באוסף של סמלים מקובלים (תווים) המכילים אותיות אלפבית, ספרות, סימני פסוק ובנוסף סימנים מיוחדים כמו: +, –, $, %. צירוף של סמלים אלה מאפשר לכתוב טקסטים שונים. כדי לייצג את כל אוסף הסימנים הכלולים בשפה הטבעית, המחשב משתמש בשיטת קידוד המגדירה לכל תו ערך מספרי בינארי. קיימים כיום כמה קודים תקניים (סטנדרטיים) לייצוג תווים, הנמצאים בשימושם של מחשבים רבים, וביניהם נציין את קוד ASCII ואת קוד UNICODE. השימוש בקודים תקניים מאפשר להעביר מידע בין מחשבים שנוצרו על-ידי חברות שונות. לדוגמה: כדי לשלוח קובץ טקסט ממחשב אישי של חברת IBM למחשב מקניטוש, צריכים לשלוח טקסט מקודד באמצעות קוד ASCII. חשבו, מה היה קורה אם כל מחשב היה משתמש בקוד שונה לייצוג טקסט!

2.4.1 קוד ASCII

קוד ASCII (American Standard Code for Information Interchange), חובר בשנת 1968 ונקבע כשיטת קידוד תקנית לכל המחשבים. בקוד זה יש התאמה חד-חד ערכית בין התו והקוד המספרי שנקבע בתקן. במקור קוד זה נקבע כקוד בן 7 סיביות לכל תו, אך כיום משתמשים בקוד ASCII מורחב שיש בו 8 סיביות לתו. קוד ASCII מורחב מאפשר להציג 256 תווים שונים, מתוכם 128 הראשונים כוללים את אותיות ה-ABC האנגלי (אותיות גדולות ואותיות קטנות), ספרות ותווים נוספים, ובשאר 128 התווים משתמשים לצרכים מיוחדים. לדוגמה: בשפה העברית מיוצגות אותיות האלפבית על-ידי הקודים שמספרם מ-224 עד 250 (לפי תקן ISO 8859-8). שימוש בקוד ASCII הוא משמר מידע מפני שיש התאמה חד-חד ערכית בין התו לקוד המספרי המייצג אותו. כך למשל, הערך של התו A הוא 65 ושל התו B הוא 66.

טבלה 2.7 מציגה את קודי ה-ASCII עבור 7 סיביות (כלומר, היא מכילה 128 תווים). שימו לב, ש-31 הקודים הראשונים אינם מייצגים, למעשה, אותיות או סימנים בשפה, אלא משמשים כתווי בקרה למדפסות, צגים, וכדומה.

דוגמה 2.3 תרגום המילה ASSEMBLY למספר בינארי

נחליף כל אות במילה ASSEMBLY לקוד ASCII ונקבל את המספר הבינארי הזה:

01000001 01010011 01010011 01000101 01001101 01000010 01000110

ניעזר בטבלה 2.7 לביצוע המרה מקוד ASCII לטקסט.

טבלה 2.7
קוד ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

שאלה 2.20

הציגו את הטקסטים הבאים כקוד ASCII :

א. Assembly

ב. 6 = 2×3

ג. 'Hello World!'

ד. רשמו את השם הפרטי שלכם (באותיות אנגליות קטנות) בקוד אסקיי.

שאלה 2.21

פענחו (בעזרת טבלה 2.7) את המספר הבינארי הבא ורשמו את המילה המתקבלת :

01000001 01010011 01010011 01000101 01001101 01000010 01001100 01011001

2.4.2 קוד Unicode

עם ההתפתחות הטכנולוגית המהירה ובעיקר עם מהפכת האינטרנט, התברר כי 256 תווים אינם מספיקים לייצוג של תווים בשפות שונות. למשל, קידוד בן 8 סיביות אינו מספיק לייצוג של הסימנים בשפה הסינית, שיש בה אלפי סימנים. בעיה נוספת נבעה מקוד ASCII המורחב, שבו אפשר לייצג בו-זמנית רק שפה אחת שאינה לטינית. לדוגמה, בנוסף לאותיות האנגליות שמיוצגות בקודים הקטנים מ-128, אפשר להציג אותיות בשפה נוספת, למשל עברית או צרפתית, המיוצגים בקודים הגדולים מ-128. השימוש באותם קודים לייצוג תווים בשפות שונות יכול לגרום לסתירות ומקשה על עיבוד תמליל רב-לשוני, כמו למשל עיבוד של מסמך רב-לשוני הכולל אנגלית, עברית וצרפתית.

לכן הוגדר בשנת 1991 תקן חדש ומורחב לייצוג תווים במחשב; התקן החדש נקרא Unicode ("קוד אחיד") ובו כל תו מיוצג באמצעות 16 סיביות (במקום 8 סיביות בקוד ASCII). בקוד המורחב אפשר להציג 65,536 תווים שונים, וכך נמצא בטבלת הקודים ערך מספרי המייצג כל תו אפשרי בכל שפה המוכרת כיום. לדוגמה: ייצוג האות A ב-Unicode הוא U+0041 – האות U מציינת Unicode והמספר 0041 הוא מספר בשיטה ההקסדצימלית המייצג את האות A. באופן דומה U+0042 מציין את התו B וקוד U+0043 מציין את התו C. הקוד עצמו כבר כולל מנגנון המאפשר את הרחבתו הנוספת בעתיד, במקרה שיתעורר צורך.

תקן זה מיושם כיום במערכות מחשב רבות ברחבי העולם, מאחר שמערכת ההפעלה Windows מבוססת עליו, החל ב-Windows NT גרסה 3.1. התקן עדיין אינו מצוי בכל מערכות המחשב, אך השימוש בו צפוי לגדול בצורה משמעותית בשנים הקרובות. התפתחות רשת האינטרנט בפרט ורשתות המידע בכלל, והצורך בהתאמת רשתות אלה לשימוש גלובלי ורב-לשוני, יוצרים קרקע פורייה להתבססות של תקן Unicode כתקן המחייב במערכות המחשבים בכל העולם.

שאלה 2.22

א. האם אפשר להמיר טקסט הרשום בקוד ASCII לטקסט ב-Unicode? אם כן הסבירו כיצד.

ב. האם ההמרה ההפוכה (מטקסט ב-Unicode לטקסט בקוד ASCII) אפשרית באותה דרך?

שאלה 2.23 (רשות)

אתרו את קודי Unicode לאותיות האלפבית בעברית באתר:

http://www.unicode.org/

ורשמו את השם שלכם ב-Unicode.

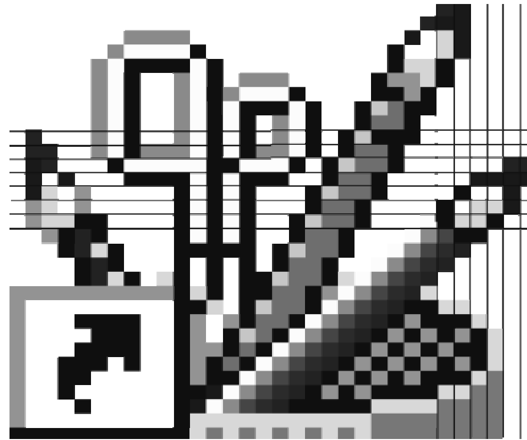
לסיכום, כדי לייצג תווים של טקסט במחשב, משתמשים בטבלאות המכילות קוד מספרי לכל תו. ייצוג זה הוא משמר מידע, כך שלכל טקסט ניתן ליצור המרה חד-חד ערכית של קודים מספריים.

2.5 ייצוג תמונה

תמונה מכילה מידע רב שאפשר להגדירו באמצעות מאפיינים רבים, כגון: בהירות, צבעוניות, ניגודיות, הפרדה וכדומה. יש לייצג במחשב מאפיינים אלה באמצעות הספרות 0 ו-1 בלבד. אחת השיטות המקובלות לייצג תמונה במחשב, היא על-ידי חלוקתה ליחידות ציור קטנות הנקראות פיקסלים (**Pic**ture **El**ement – Pixel). הפיקסל מכיל את המאפיינים של יחידת הציור הקטנה שהוא מייצג יחד עם הפרמטרים של מיקומה בתמונה. כדי לייצג תמונה במחשב מבצעים שתי פעולות:

– תחילה דוגמים את התמונה על-ידי חלוקתה לפיקסלים ;
– בשלב שני מבצעים כימוי שבו מומר המידע של הפיקסל למספר בינארי.

איור 2.1 מתאר דוגמה של חלוקת ציור לפיקסלים.



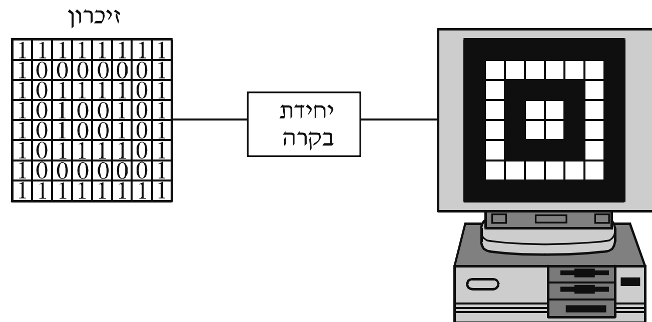
איור 2.1
חלוקה של ציור לפיקסלים

כדי להדגים את אופן השימוש בשיטה זו נתייחס רק למאפייני הצבע ותחילה נתאר כיצד אפשר להציג ציור בשני צבעים בלבד : שחור ולבן. איור 2.2 מתאר ציור המורכב ממשבצות שחורות ולבנות, שבו כל משבצת היא פיקסל, ולידו מטריצה (טבלה דו-ממדית) שבה כל משבצת מייצגת את הצבע של אותו פיקסל בציור המקורי. אפשר להשתמש בסיבית אחת כדי להציג שני צבעים : למשל : 0 מייצג צבע לבן ו-1 מייצג צבע שחור.

איור 2.2 מתאר את טבלת הפיקסלים מצד אחד ומצד שני את התמונה על גבי הצג.
חשבו, כיצד נייצג פיקסל בתמונה שיש בה 4 צבעים, למשל : שחור, אפור בהיר, אפור כהה ולבן? במקרה כזה, נצטרך להקצות לכל פיקסל שתי סיביות, בעזרתן נוכל להציג 4 צבעים שונים. לדוגמה :

00 – לבן
01 – אפור בהיר
10 – אפור כהה
11 – שחור

כלומר, אם התמונה באיור 2.2 כוללת ארבעה צבעים, דרושות 2 סיביות לכל פיקסל, כלומר פי שניים ממספר הסיביות הדרושות לייצוג שני צבעים.



איור 2.2
טבלת הפיקסלים ותמונה על הצג

שאלה 2.23

כמה סיביות נצטרך כדי להציג 16 צבעים לכל פיקסל? 256 צבעים לכל פיקסל?

לסיכום, כמות הסיביות שנצטרך כדי לייצג תמונה תלוי במספר הפיקסלים ובמספר הצבעים המגדירים פיקסל. בדרך כלל ייצוג של תמונה במחשב (גם כשהיא קטנה) דורש שימוש בסיביות רבות מאוד, ובהתאם נדרש הרבה מקום לאחסון.

שאלה 2.24

חשבו כמה סיביות דרושות לתמונה שיש בה 256 צבעים שונים המורכבת מ-1000 פיקסלים.

התמונה מייצגת מידע רציף; תהליך המרת התמונה לאפסים ואחדים על-ידי חלוקתה למשבצות בדידות, גורם לאיבוד מידע. באופן תיאורטי, אנו יכולים לחלק את התמונה לרשת צפופה יותר ויותר של משבצות רבות וקטנות יותר ויותר (וכך להקטין את גודל יחידת הציור שמתאר הפיקסל). אולם, ביישומים שאנו מריצים, מספר הפיקסלים הוא סופי, ואנו מניחים שקיים גבול, שבו חלוקה למספר גדול יותר של פיקסלים לא תשפיע על איכות התמונה המוצגת לנו. כלומר, למרות שייצוג תמונה במחשב אינו משמר מידע, אנו מסתמכים על העובדה שמעבר לגבול זה, העין שלנו לא תוכל להבחין בהבדל בין הצגת

תמונה אחת להצגת אותה תמונה המחולקת למספר גדול יותר של פיקסלים, וכך איבוד המידע לא יפריע לנו להשיג את מטרת היישום.

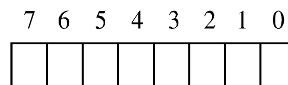
2.6 יחידות זיכרון לאחסון מידע במחשב

גודלם של המספרים בעולם המתמטיקה הוא אינסופי, אבל במחשב, בגלל סיבות טכנולוגיות, יחידות הזיכרון בהן משתמשים לאחסון מידע, הן בעלות גודל קבוע. בתיאור המחשב הפשוט (בפרק הראשון) הִכּרנו שני סוגים של יחידות זיכרון: תאי זיכרון, ואוגרים הנמצאים במעבד. גודל של יחידת אחסון נקבע על-פי מספר הסיביות שניתן לאחסן בה. השימוש בגדלים קבועים מגביל את מספר המספרים שניתן לאחסן. ביחידת אחסון בת n סיביות אפשר לאחסן 2^n מספרים שונים. גודל יחידת האחסון קובע גם את תחום הערכים שניתן לאחסן בה, ולעיתים תוצאת העיבוד שמתקבלת חורגת מהגודל המקסימלי שאפשר לאחסן ביחידת האחסון, ואז מתקבלת תוצאה שגויה. תופעה זו נקראת "**גלישה**" (overflow). בהמשך נציג כמה שיטות לטפל במקרים כאלה. בסעיף זה נציג את הגדלים המקובלים של יחידות אחסון ואת תחום המספרים שניתן לאחסן בכל יחידה.

במחזור אפיק (מחזור קריאה או מחזור כתיבה) יחידת המידע אליה ניתן לגשת בבת אחת היא "**מילה**". המילה מוגדרת בצורה שונה במחשבים שונים והיא תלויה בסוג המעבד. יש מעבדים בהם מילה היא 8 סיביות (מיקרו מעבדים "עתיקים", כמובן), יש של מילים בנות 16 סיביות. כיום נפוצים מעבדים בהם מילה היא בת 32 סיביות (למשל: פנטיום 4), וכבר עובדים עם מעבדים בהם מילים הם 64 סיביות או אפילו 128. במעבד אינטל 8086, איתו נעבוד בהמשך, גודל המילה היא 16 סיביות, ובהתאם גם רוחב פס הנתונים הוא 16 סיביות. אולם בכל בשפת אסמבלי ניתן להגדיר גדלים נוספים של יחידות מידע בהם ניתן להשתמש בהוראות. בסעיף זה נתאר את הגדלים המקובלים במעבד 8086.

א. הבית (byte)

יחידת הזיכרון הקטנה ביותר המקובלת כיום נקראת "בית" (byte) והיא מכילה 8 סיביות.



איור 2.3
בית

בית אחד מאפשר להציג את כל המספרים הבינאריים השלמים (ללא סימן) בתחום שבין 00000000_2 לבין 11111111_2. בגודל זה משתמשים גם כדי לאחסן את טיפוס הנתונים char המכיל תו המיוצג בקוד ASCII. אם נמיר ערכים אלה לשיטה העשרונית, נקבל את כל המספרים מ-0 עד 255, בהתאמה. מספר המספרים שאפשר להציג באמצעות בית אחד הוא:

$$2^8 = 256$$

מבחינה היסטורית, היחידה "בית" נקבעה כ-8 סיביות, מאחר שכפי שכבר אמרנו, כדי לייצג את כל אותיות הא"ב האנגלי וסימנים נוספים נדרשות 7 סיביות, ולכן נוספה עוד סיבית אחת ששימשה לגילוי של שגיאות – נושא שלא נדון בספר זה.

שאלה 2.26

רשמו את תחום הערכים שניתן לייצג בבית אחד בשיטה האוקטלית ובשיטה ההקסדצימלית.

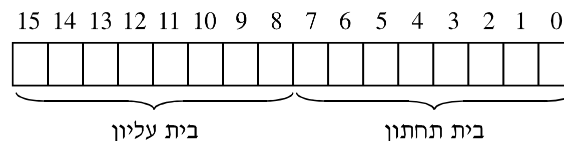
הבית הוא גודל נפוץ מאוד, ולכן נוהגים לציין את גודל הזיכרון באמצעות מספר הבתים שהוא מכיל, וגם את קצב העברת הנתונים ברשת תקשורת מציינים באמצעות מספר הבתים שמועברים בשנייה ממחשב למחשב, אם כי כאשר מתייחסים לקו הפיסי ממש בו מועברים האותות, מציינים את הקצב במספר הסיביות לשנייה, ולא במספר הבתים לשנייה.

ב. המילה (word)

המילה היא יחידת זיכרון המכילה 16 סיביות. מספר המספרים השלמים ללא סימן שניתן לייצג במילה הוא:

$$2^{16} = 65,536$$

הערך העשרוני הנמוך ביותר הוא 0 והערך העשרוני הגבוה ביותר הוא 65,535.



איור 2.4
מילה

אפשר להתייחס למילה כאל יחידת זיכרון אחת המכילה שני בתים. במקרה כזה, הסיביות שמיקומן 0 עד 7 הן הבית **התחתון** (L.O. byte – Low Order byte) של המילה והסיביות שמיקומן מ-8 עד 15 הן הבית **העליון** שלה (H.O. byte – High Order byte).

שאלה 2.27

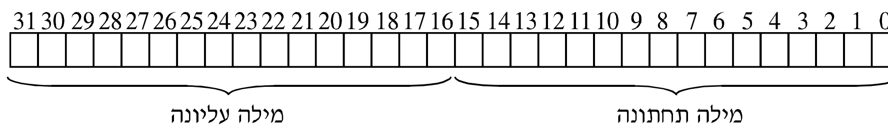
עבור מספרים שלמים, ללא סימן, רשמו את הערך הנמוך ביותר ואת הערך הגבוה ביותר שאפשר לרשום באמצעות מילה בת 16 סיביות – בשיטה הבינארית, בשיטה האוקטלית ובשיטה ההקסדצימלית.

ג. מילה כפולה (Double Words)

מילה כפולה היא יחידת זיכרון המכילה 32 סיביות. מספר המספרים השלמים, ללא סימן, שאפשר לייצג במילה כפולה, הוא:

$$2^{32} = 4,294,967,296$$

כאשר הערך העשרוני הנמוך ביותר הוא 0 והערך העשרוני הגבוה ביותר הוא 4,294,967,295.



איור 2.5
מילה כפולה

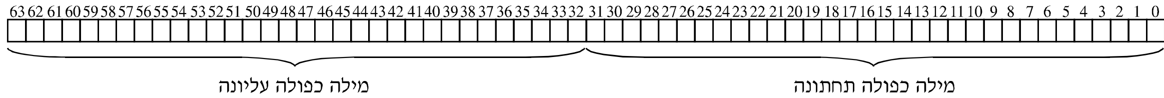
באופן דומה למילה, ניתן להתייחס למילה כפולה כאל יחידת זיכרון המכילה שתי מילים, כאשר הסיביות שמיקומן 0 עד 15 הן המילה הנמוכה במילה הכפולה והסיביות שמיקומן מ-16 עד 31 הן המילה העליונה במילה הכפולה.

שאלה 2.28

עבור מספרים שלמים, ללא סימן – רשמו את הערך הנמוך ביותר ואת הערך הגבוה ביותר שאפשר לרשום באמצעות מילה כפולה, בשיטה הבינארית, בשיטה האוקטלית ובשיטה ההקסדצימלית.

ד. מילה מרובעת (Quadruple Words או Quad Word)

מילה מרובעת מכילה 4 מילים שהם 64 סיביות. ניתן להתייחס למילה מרובעת כאל יחידת זיכרון המכילה שתי מילים כפולות, כאשר הסיביות שמיקומן 0 עד 31 מייצגות את המילה הכפולה הנמוכה והסיביות שמיקומן מ-31 עד 63 מייצגות את המילה הכפולה העליונה. יחידת אחסון כזו קיימת במעבדים מתקדמים מהדורות האחרונים.



איור 2.6
מילה מרובעת

שאלה 2.29

א. חשבו את המספר הבינארי השלם הגדול ביותר שניתן לרשום במילה כפולה.

ב. רשמו ערך זה בשיטה העשרונית ובשיטה הקסדצימלית.

שאלה 2.30

חשבו כמה בתים מכילה מילה כפולה וכמה בתים מכילה מילה מרובעת?

טבלה 2.8 מסכמת את מספר הסיביות והערכים שאפשר לרשום ביחידות אחסון במחשב. בטבלה אפשר לראות כי יש קשר בין הגדלים **בית**, **מילה**, **מילה כפולה ומילה מרובעת**: מספר הסיביות בכל יחידה הוא כפולה ב-2 של הגודל הקטן ממנו.

טבלה 2.8
יחידות מידע במחשב

יחידת מידע	בית	מילה	מילה כפולה	מילה מרובעת
מספר הסיביות	2^3=8	$2^4 = 16$	$2^5 = 32$	$2^6 = 64$
מספר הערכים	256	65536	4294967296	$1.84467*10^{19}$

תופעה זו לא צריכה להפתיע אם ניזכר כי בשיטה העשרונית: כאשר מכפילים 10^n ב-10 מקבלים 10^{n+1}.

כדי לייצג מספרים טבעיים במחשב, נצטרך לקבוע תחילה את גודל היחידה שבה נרצה להשתמש. כדי לבחור את סוג היחידה המתאימה לנתון שאנו רוצים לייצג, עלינו להבטיח ששימור המידע יתקיים, ולכן נצטרך לוודא שסוג היחידה שבחרנו יכולה לייצג את כל המספרים בהם נרצה לטפל ושאותם נרצה לעבד. לדוגמה: ברצוננו לייצג ולעבד ציונים של תלמידים; כיוון שציון של תלמיד נע בין 0 ל-100, יספיק בית אחד לשמירת ציון אחד. אבל כדי להציג מספרים בין 1 ל-1000 נצטרך להשתמש במילה. חשבו, מה יקרה אם נבחר להציג את המספר 320 בבית? ברור לנו שלא יהיה מקום לאחסן את הנתון הדרוש, וחלק מהמידע ילך לאיבוד.

שאלה 2.31

א. באילו יחידות אחסון נשתמש כדי להציג קוד המציין את הגימטרייה של כל אחת מאותיות האלפבית?

ב. באילו יחידות אחסון נשתמש כדי להציג פיקסל בתמונה בה יש 65536 צבעים שונים?

שאלה 2.32

אחד הייצוגים של צבעים הוא ייצוג RGB, שבו הצבע מוצג כערוב של שלושת צבעי היסוד: אדום (R, מהמילה Red), ירוק (G, מהמילה Green) וכחול (B, מהמילה Blue). כל צבע יסוד מייצג 256 גוונים שונים. לדוגמה: לייצוג צבע כחול-ירקרק נשתמש בתערובת 102,205,170.

באילו יחידת אחסון תשתמשו כדי לייצג את המידע על צבע של פיקסל?

סיכום

בפרק זה תיארנו את השיטות לייצוג מידע במחשב. מידע מאוחסן ומעובד במחשב בשיטה הבינארית; לאחסון המידע משמשות יחידות בגדלים אלה: **בית**, **מילה**, **מילה כפולה ומילה מרובעת**. כדי להציג את המספרים הבינאריים בצורה קריאה ונוחה יותר, אנו משתמשים בשיטות ייצוג הקסדצימלית ואוקטלית.

לכל סוג מידע יש שיטה המותאמת להמרתו מייצוג עשרוני, בו אנו נוהגים לעשות שימוש יומיומי, לייצוג בינארי:

מידע מספרי המיוצג במחשב מומר ממספר עשרוני למספר בינארי;
טקסט מומר באמצעות טבלת תרגום לקוד ASCII או ל-UNICODE, שגם הקוד בהם הוא בינארי;

תמונות מחולקות ליחידות מידע קטנות, הנקראות פיקסלים; כל פיקסל מיוצג בצורה בינארית.

המרת מידע לייצוג במחשב אינה תמיד משמר מידע: סיבה אחת נובעת מהגדלים של יחידות המידע בהן משתמש המחשב; סיבה שנייה נובעת מחוסר היכולת להמיר בצורה מדויקת מידע רציף (כמו תמונה) לייצוג בינארי.

בבואנו להמיר מידע לייצוג במחשב עלינו לשקול מהי השיטה המתאימה ומהו הגודל המתאים. אם לא מתקיים שימור מידע, עלינו להחליט אם תופעה זו תפריע לביצוע המשימה. במידה שהיא מפריעה – יש לקחת בחשבון את הטעויות האפשריות.

נספח

ייצוג כללי של מספרים בשיטה מיקומית השונה מ-10 והמרתם לשיטה העשרונית

בנספח זה מוצגת שיטה כללית להמרה משיטת ספירה אחת לשיטת ספירה אחרת.
אפשר לתאר מספרים בשיטה ספירה מיקומית, שהבסיס שלה שונה מ-10. את המספר N_b שהוא מספר בעל n ספרות, המתואר בשיטת ספירה שבסיסה b (שלם, חיובי, גדול מ-1), ניתן לרשום כסכום של מכפלות בצורה הזו:

$$K_b = (a^{n-1}a^{n-2}a^{n-3} \cdots a_2 a_1 a_0)_b$$
$$= a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + a_{n-3}b^{n-3} + \ldots + a_2b^2 + a_1b^1 + a_0b_0$$

כדי לכתוב מספר לפי בסיס b אנו זקוקים ל-b סימנים המתארים ספרות. לדוגמה, כדי לרשום מספר לפי בסיס 6 אנו זקוקים ל-6 ספרות:

0 1 2 3 4 5

חישוב הסכום של מספר המיוצג כסכום מכפלות, מאפשר **להמיר כל מספר בשיטת ספירה מיקומית לשיטה העשרונית**. נציג כמה דוגמאות:

$$421_5 = 4\times5^2 + 2\times5^1 + 1\times5^0 = 111_{10}$$

או 111 (ללא ציון הבסיס 10).

$$32461_8 \quad = 3 \times 8^4 + 2 \times 8^3 + 4 \times 8^2 + 6 \times 8^1 + 1 \times 8^0 = 13617$$
$$101101_2 \quad = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$
$$A1F_{16} \ = A \times 16^2 + 1 \times 16^1 + F \times 16^0 = 2591$$

כדי לחשב את הערך העשרוני של $A1F_{16}$, נמיר כל ספרה הקסדצימלית (המיוצגת כאות) לערך העשרוני שהיא מייצגת, וכך חישבנו:

$$A1F_{16} = A \times 16^2 + 1 \times 16^1 + F \times 16^0 =$$
$$=10 \times 16^2 + 1 \times 16^1 + 15 \times 16^0 = 2591$$

העברת מספרים שלמים מבסיס 10 לבסיס אחר

בתחילת הנספח עסקנו בייצוג מספרים לפי בסיס כלשהו, וראינו כיצד להפוך אותם למספרים עשרוניים. עתה נתאר כיצד לבצע את הפעולה ההפוכה, כלומר: כיצד לייצג מספר עשרוני בבסיס אחר. לשם כך נשתמש בנוסחה שבעזרתה ניתן לבטא מספר בבסיס כלשהו כסכום של מכפלות.

$$N_b = a_{n-1} \cdot b^{n-1} + a_{n-2} \cdot b^{n-2} + \ldots + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0$$

נשנה את צורת הביטוי לעיל בלי לשנות את ערכו; נעשה זאת על-ידי הוצאת הגורם המשותף b אל מחוץ לסוגריים. הנוסחה שנקבל תהיה:

$$K_b = (a_{n-1} \cdot b^{n-2} + a_{n-2} \cdot b^{n-3} + \ldots + a_2 \cdot b^1 + a_1)b + a_0$$

על-ידי חלוקה של המספר K_b ב-b ומציאת השארית, נקבל כמנה את הביטוי שבסוגריים, ואילו השארית תהיה a_0. שארית זו היא הספרה הפחות משמעותית במספר K_b.

אם ניקח את המנה, שהיא התוצאה של החילוק הראשון

$$K_b = a_{n-1} \cdot b^{n-2} + a_{n-2} \cdot b^{n-3} + \ldots + a_2 \cdot b + a_1$$
$$= (a_{n-1}b^{n-3} + a_{n-2}b^{n-4} + \ldots + a_2) \cdot b + a_1$$

ונחלק אותה שוב ב-b, בדרך שתוארה קודם, נקבל כשארית a_1. הפעם מייצגת השארית a_1 את הספרה שערכה המיקומי שווה b_1 (ראה ביטוי מקורי למעלה).

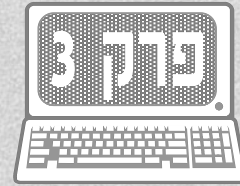
חלוקה נוספת של התוצאה ב-b תיתן כשארית את a_2. זו הספרה שערכה המיקומי b_2. כך נמשיך את התהליך עד שנקבל כתוצאה מהחלוקה ב-b את המנה 0 ושארית a_{n-1}, שהיא הספרה המשמעותית ביותר.

לדוגמה: נמיר את המספר 147 למספר בבסיס 6. לפי התהליך שתיארנו לעיל, נבצע חלוקות חוזרות ונשנות של המספר 147 בבסיס 6. השאריות שיתקבלו יהוו את ספרות הייצוג בבסיס 6, החל בספרה הפחות משמעותית (קראו משמאל לימין):

הערך המיקומי של השארית	ספרת הייצוג לפי בסיס 6	השארית	המנה	
6^0	a_0	3	24	147:6 =
6^1	a_1	0	4	24:6 =
6^2	a_2	4	0	4:6 =

מכאן אפשר לרשום: $147_{10} = 403_6$.

בדיקה: $4 \times 6^2 + 0 \times 6^1 + 3 \times 6^0 = 147_{10}$.



פעולות אריתמטיות על ייצוג בינארי במחשב

בפרק זה נסביר כיצד מייצגים מספרים בינאריים שלמים מכוונים (כלומר: בעלי סימן), ונסביר כיצד מתבצעות ארבע הפעולות האריתמטיות: חיבור, חיסור, כפל וחילוק עם מספרים בינאריים. מידע זה יאפשר לנו לעקוב אחר ביצוע התכנית, לבדוק את התוצאות שמתקבלות מביצוע החישובים, ובמידת הצורך יאפשר לנו גם לאתר שגיאות.

3.1 חיבור וחיסור מספרים בינאריים בלתי מכוונים

חיבור בינארי

תהליך החיבור של מספרים מתבצע בצורה דומה בשיטות ספירה מיקומיות שונות. כדי להציג את התהליך, נדגים את פעולת החיבור של שני מספרים עשרוניים:

$$\begin{array}{r} 359_{10} \\ +\ 245_{10} \\ \hline \end{array}$$

לחיבור שני מספרים אלה מבצעים את הפעולות הבאות:

א. $9 + 5 = 14$

14 גדול מהבסיס (10), לכן מפחיתים את הבסיס ($14 - 10 = 4$), כותבים את התוצאה (4), ומוסיפים יחידה (1) , שהיא ה**נֶשֶׂא** (carry), לסיכום שתי הספרות הבאות.

ב. $5 + 4 + 1 = 10$

מועברת מחיבור שתי הספרות הקודמות ($9 + 5$)

10 שווה לבסיס (10), לכן מפחיתים את הבסיס ($10 - 10 = 0$), כותבים את התוצאה (0), ומוסיפים נשא שהוא 1 לסיכום שתי הספרות הבאות.

ג. $6 = 1 + 2 + 3$

מועברת מחיבור שתי הספרות הקודמות

6 קטן מהבסיס (10), לכן כותבים את התוצאה (6) ולא מעבירים כלום לחיבור שתי הספרות הבאות. זוהי התוצאה הסופית (אין ספרות נוספות לחיבור). כלומר:

$$\begin{array}{r} 1\;1\;\; \\ 3\;5\;9_{10} \\ +\;\; 2\;4\;5_{10} \\ \hline 6\;0\;4_{10} \end{array}$$

מיומנותנו בביצוע פעולות בבסיס עשרוני היא כה גדולה עד כי איננו שמים לב לפרטי הביצוע של התהליך שתואר לעיל; באותה דרך אפשר לבצע פעולות חיבור גם בשיטה הבינארית.

נדגים את תהליך החיבור בשיטה הבינארית, אך תחילה נציג את לוח החיבור של ספרות בינאריות (ראו טבלה 3.1). התוצאה $1+1 = 10$ המופיעה בלוח זה נראית לכאורה מוזרה, אולם היא מובנת אם זוכרים כי 10_2 (עשר בבסיס 2) הופך בשיטה העשרונית למספר 2, ולכן היה צריך לכתוב, למעשה: $1_2+1_2 = 10_2$. מאחר שכל הלוח שלהלן הוא בבסיס בינארי, נשמיט את כתיבת הבסיס.

טבלה 3.1
לוח החיבור הבינארי

הנשא	הסכום	הספרות	
(carry)	a + b	b	a
0	0	0	0
0	1	1	0
0	1	0	1
1	0	1	1

הסיביות שצריכים לחבר רשומות בשני הטורים השמאליים של טבלה 3.1 ; **ספרת הסכום**, כלומר הספרה הפחות משמעותית של הסכום, רשומה בטור השלישי וה**נֶשֶׂא** רשום בטור הימני של טבלה זו.

לפי כללי האריתמטיקה הרגילה, מופיע **הנשֹא** כאשר תוצאת החיבור שווה לבסיס או גדולה ממנו (הבסיס 2 במקרה שלנו) והוא מתחבר לספרות הסמוכות משמאל.

נדגים את תהליך החיבור של שני מספרים, תחילה ללא הסבר מפורט. החיבור מבוסס על לוח החיבור שבטבלה 3.1.

הנשא: 1 1 1   1

$$
\begin{array}{r}
111010_2 \\
+ \\
101011_2 \\
\hline
1100101_2
\end{array}
$$

הסכום: 1100101_2

נבצע את החיבור בהתאם ללוח החיבור: נכתוב את ספרת הסכום מתחת לקו ואת הנשא נכתוב מעל לטור הסמוך, משמאל.

נסו לבצע את החיבור. האם ברור לכם כיצד הגענו לתוצאה?

אם הדרך לא ברורה לכם, קראו את ההסבר לחיבור הבא ואחר חזרו וחשבו את התרגיל הזה.

נחבר את המספרים $10101_2 + 11001_2 =$

נרשום את פעולת החיבור בשיטה הבינארית, ומימין לה נרשום את החיבור של אותם מספרים כשהם מיוצגים בשיטה העשרונית.

חישוב עשרוני	חישוב בינארי	
	1 1	**נשא**
21	10101_2	
+	+	
25	11001_2	
46	101110_2	

נפרט את פעולת החישוב הבינארי:

שתי הספרות הימניות ביותר הן 1 ו-1, סכומן 2, כלומר, 10_2, ולכן אנו רושמים 0 בתוצאה, במקום הימני ביותר, ו"זוכרים" 1 בתור נשא. שתי הספרות הבאות (מקום שני מימין) הן 0 ו-0. נחבר אותן ואת הנשא שנוצר בחיבור הקודם ($0 + 0 + 1$), ונקבל את התוצאה 1; נרשום אותה במקום השני מימין. סכום הספרות התופסות את המקום השלישי הוא 1, וסכום הספרות הנמצאות במקום הרביעי גם הוא 1. סכום הספרות האחרונות הוא 2, לכן יש לרשום במקום החמישי של התוצאה 0, ובמקום השישי רושמים את הנשא 1. בצד החישוב הבינארי מופיע החישוב העשרוני, ועל-ידי בדיקה ניווכח שקיבלנו תוצאות זהות.

נביא עוד כמה דוגמאות:

א.

חישוב עשרוני

$$\begin{array}{r} 11 \\ + \quad 8 \\ \hline 19 \end{array}$$

חישוב בינארי

$$\begin{array}{r} 1011_2 \\ + \quad 1000_2 \\ \hline 10011_2 \end{array}$$

ב.

$$\begin{array}{r} 15 \\ + \quad 15 \\ \hline 30 \end{array}$$

$$\begin{array}{r} 1111_2 \\ + \quad 1111_2 \\ \hline 11110_2 \end{array}$$

בדוגמה האחרונה הנשא הוא 1 וגם שתי הספרות המחוברות הן 1. כיוון שהתוצאה היא 3 ($1 + 1 + 1 = 3$) או 11_2 ברישום בינארי, אנו רושמים 1 כתוצאה ו"זוכרים" 1 בתור נשא. כשצריכים לחבר יותר משני מספרים, לדוגמה $101_2 + 11_2 + 1101_2 =$, אנו יכולים לחבר תחילה שני מספרים ולתוצאה לחבר את המספר השלישי. דרך אחרת היא להשתמש בלוח החיבור (טבלה 3.1) ולסכם תחילה את היחידות, אחר-כך את העשרות וכך הלאה.

	1	1	1		**נשא:**
		1	0	1	**המספר הראשון**
+			1	1	**המספר השני**
	1	1	0	1	**המספר השלישי**
1	0	1	0	1	**התוצאה**

שאלה 3.1

בצעו את פעולות החיבור הבינארי הרשומות להלן, ובדקו את התוצאות בעזרת ייצוג עשרוני.

א. $100011_2 + 101_2$

ב. $1011100_2 + 11011_2$

ג. $1011000101_2 + 11001_2 + 1001_2$

שאלה 3.2

חברו את המספרים המובאים להלן והציגו את התוצאה בבסיס 2, 10 ו-16.

א. $10F_{16} + C10_{16}$

ב. $E0_{16} + FF_{16} + AF_1$

הדרכה: אפשר להמיר את המספרים הרשומים בשיטה ההקסדצימלית למספרים בינאריים, ולאחר ביצוע החישוב להמירם לבסיס הרצוי. דרך אחרת היא לבנות לוח חיבור לשתי ספרות, בבסיס 16, ולהשתמש בו לביצוע החיבור הדרוש.

בעולם המתמטיקה, אין הגבלה על מספר הספרות שיהיו בתוצאת החיבור, אולם, כפי שכבר הסברנו, במחשב יחידות האחסון הן בעלות גדלים קבועים, ולכן תחום המספרים שניתן לאחסן בהם מוגבל. משום-כך, בעת ביצוע פעולות אריתמטיות (כגון חיבור), ייתכן שתוצאת החיבור תגלוש מגודל יחידת האחסון. לדוגמה, נחבר שני מספרים המאוחסנים בתאי זיכרון מטיפוס בית, כלומר הם בני 8 סיביות:

$$\begin{array}{r} 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\ +\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \end{array}$$

ניתן לראות כי תוצאת החיבור שהתקבלה היא בעלת 9 סיביות. במקרה כזה, יאוחסנו בתא הזיכרון (שגודלו בית) רק 8 הסיביות הפחות משמעותיות, כלומר המספר 10011110_2, והסיבית המשמעותית ביותר אבדה. לכן התוצאה שהתקבלה מחישוב זה שגויה. תופעה זו נקראת "**גלישה**" (Overflow). המעבד משתמש באוגר מיוחד (אוגר הדגלים) כדי להתריע על כך שבתוצאת החישוב התרחשה גלישה. בפרק החמישי נתאר תופעה זו בהרחבה ונלמד לטפל בה.

שאלה 3.3

א. לתוצאת החיבור של שני מספרים בינאריים הקצו יחידת אחסון מטיפוס ***מילה***. תנו דוגמה לשל שני מספרים שסכומם גורם לגלישה.

ב. האם בחיבור שני מספרים תיתכן גלישה מעבר לסיבית התשיעית? הסבירו את תשובתכם.

חיסור בינארי

נציג כעת את ההסבר על פעולת חיסור של שני מספרים בינאריים a, b עבור המקרה שבו המחוסר (a) גדול או שווה למחסר (b), כלומר $a \geq b$ (ולכן התוצאה המתקבלת חיובית). ונדחה את הדיון במקרה שבו המחוסר קטן מהמחסר, כלומר $a < b$ (ולכן התוצאה המתקבלת היא שלילית). נדחה את הדיון במקרה הזה לסעיף שבו נסביר כיצד מיוצגים מספרים שליליים במחשב.

תחילה, ננתח דוגמה של פעולת חיסור עשרונית ונשתמש באותו עיקרון לחיסור של מספרים בינאריים:

$$\begin{array}{r} 329 \\ -\underline{245} \end{array}$$

הפעולות שאנו מבצעים הן:

א. בשלב הראשון אנו מפחיתים את ספרת היחידות של המחסר מספרת היחידות של המחוסר:

$$9 - 5 = 4$$

ב. בשלב השני אנו מפחיתים את ספרת העשרות של המחסר מספרת העשרות של המחוסר, כלומר: $2 - 4$. כיוון שפעולה זו אינה אפשרית, אנו לווים 10 מספרת המאות של המחוסר ומבצעים את החישוב $12 - 4 = 8$. לאחר שלווינו 10 מספרת המאות של המחוסר, סִפרת המאות שלו תהיה 2. נוכל לרשום את התרגיל בצורה הזו:

$$\begin{array}{rrr} 2 & 12 & 9 \\ -\underline{2} & \underline{4} & \underline{5} \end{array}$$

ג. בשלב האחרון נפחית את ספרת המאות של המחסר מספרת המאות של המחוסר, כלומר: $2 - 2 = 0$

$$\begin{array}{r} 2\ 12\ 9 \\ -\ 2\ \ 4\ 5 \\ \hline 8\ 4 \end{array}$$

כעת נשתמש בתהליך זה לחיסור מספרים בינאריים, אך תחילה נציג את לוח החיסור הבינארי (טבלה 3.2).

טבלה 3.2
לוח החיסור הבינארי

הספרות		**הפרש**	**לווה**
a	b	a – b	(borrow)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

המספרים הרשומים בעמודה הראשונה משמאל (עמודה a) הם המחוסרים, ואלה הרשומים בעמודה השנייה משמאל (עמודה b) הם המחסרים. העמודה השלישית משמאל מכילה את ההפרש, כלומר – תוצאת החיסור, והעמודה הימנית ביותר מכילה את הלווה. כאשר המחסר גדול מהמחוסר, נעזרים בלווה – כמו בשיטה העשרונית. גם כאן **הלווה** הוא מספר שגודלו שווה לבסיס הספירה, ומוסיפים אותו לסִפרת המחוסר כדי שנוכל לבצע את החיסור. השימוש בלווה מחייב להקטין ב-1 את הספרה הסמוכה משמאל למחוסר, שממנה נלקח הלווה.

שימו לב: תוצאת החיסור $0_2 - 1_2$ אינה חיובית.

כאשר צריכים לחשב את $0_2 - 1_2$ כשלב בחיסור שני מספרים רב-ספרתיים, מבצעים למעשה את הפעולה $10_2 - 1_2$. לדוגמה:

$$\begin{array}{r} 10001101_2 \\ -\ \ 101010_2 \\ \hline 1100011_2 \end{array}$$

בדוגמה זו, בספרה השישית מימין, היינו צריכים לחסר $0_2 - 1_2$, אבל גם הספרה הסמוכה, משמאל ל-0_2 במחוסר, היתה 0_2. במקרה כזה, אפשר ללוות מהספרה השמאלית הבאה במחוסר ולבצע את הפעולה $100_2 - 1_2$:

$$\begin{array}{r} 100_2 \\ - \quad 1_2 \\ \hline 11_2 \end{array}$$

דוגמאות נוספות:

א.

$$\begin{array}{r} 15 \\ - \quad 10 \\ \hline 5 \end{array} \qquad \begin{array}{r} 1111_2 \\ - \quad 1010_2 \\ \hline 101_2 \end{array}$$

ב.

$$\begin{array}{r} 11 \\ - \quad 5 \\ \hline 6 \end{array} \qquad \begin{array}{r} \text{b}^{*} \quad \\ 1011_2 \\ - \quad 101_2 \\ \hline 110_2 \end{array}$$

שאלה 3.4

בצעו את החישובים הבאים, ובדקו את התוצאות על-ידי חיסור עשרוני:

א. $\begin{array}{r} 101011_2 \\ - \quad 100101_2 \end{array}$

ג. $\begin{array}{r} 1101100101_2 \\ - \quad 101111111_2 \end{array}$

ב. $\begin{array}{r} 10000_2 \\ - \quad 1011_2 \end{array}$

ד. $\begin{array}{r} 1011111_2 \\ - \quad 111111_2 \end{array}$

---

* b (קיצור של borrow) מסמן "לווֶה"

כפל בינארי

בטבלה 3.3 מוצג לוח הכפל הבינארי.

טבלה 3.3
לוח הכפל הבינארי

המכפלה	הספרות	
$a \cdot b$	b	a
0	0	0
0	1	0
0	0	1
1	1	1

לדוגמה נבצע את פעולת הכפל הבאה (כל המספרים בינאריים):

$$\begin{array}{r} 110_2 \\ \times\ 101_2 \\ \hline 110 \\ 000 \\ 110 \\ \hline 11110_2 \end{array}$$

הסבר: ביצוע הכפל, לפי בסיס 2, דומה לביצועו בשיטה עשרונית. כופלים את המספרים, תוך שימוש בלוח הכפל לפי בסיס 2, רושמים את התוצאות, תוך הזזות מתאימות, ומחברים את התוצאות החלקיות. כמו בשיטה העשרונית, אפשר לוותר על הכפל בספרה 0, ומספיק להזיז את התוצאה הבאה שני צעדים שמאלה, במקום צעד אחד.

להלן דוגמאות נוספות לכפל בינארי:

א.

$$\begin{array}{r} 10 \\ \times\ 3 \\ \hline 30 \end{array} \qquad \begin{array}{r} 1010_2 \\ \times\ 11_2 \\ \hline 1010 \\ 1010 \\ \hline 11110_2 \end{array}$$

ב.

$$\begin{array}{r} 7 \\ \times\ \underline{7} \\ 49 \end{array} \qquad \begin{array}{r} 111_2 \\ \times\ \underline{111_2} \\ 111 \\ 111 \\ \underline{111} \\ 110001_2 \\ \uparrow\uparrow \end{array}$$

ג. מקרה פרטי של פעולת כפל

כאשר הכופל הוא חזקה של בסיס הספירה, זהו מקרה פרטי של פעולת כפל. לדוגמה, בשיטה העשרונית:

$$1092 \times 100 = 109200$$

בשיטה העשרונית, כפל ב-10 ובחזקותיו אינו אלא הוספת המספר המתאים של אפסים מימין. גם כפל בינארי בחזקות של 2 יתבצע על-ידי הוספת מספר האפסים המתאים מימין, לכן:

$$101011_2 \times 1000_2 = \underbrace{101011}_{\text{הנכפל}}\underbrace{000}_{\text{האפסים שנוספו}}{}_2$$

בדקו את התוצאה על-ידי מעבר לבסיס עשרוני.

למעשה, כאשר המספר מאוחסן בתא בזיכרון או באחד האוגרים, כפל כנ"ל ידרוש, קודם כל, הזזה שמאלה של המספר, ואז מילוי הסיביות שמימין באפסים. לדוגמה:

$$\begin{array}{rccccccccc} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ \times & & & & & & 1 & 0 & 0 & 0 \\ \hline & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

שאלה 3.5

בצעו את הפעולות הבאות בשיטה בינארית, ובדקו את התוצאות על-ידי מעבר לייצוג עשרוני:

א. $\begin{array}{r} 1011_2 \\ \times\ 101_2 \end{array}$ ב. $\begin{array}{r} 1101_2 \\ \times\ 110_2 \end{array}$ ג. $\begin{array}{r} 11010111_2 \\ \times\ 101101_2 \end{array}$

שאלה 3.6

בצעו פעולת כפל בין המספרים הבינאריים: 1111_2 ו-10111_2 ואחסנו את התוצאה בתא זיכרון מטיפוס בית. רשמו מהו המספר שמאוחסן בתא הזיכרון; הסבירו את מדוע התקבל ערך זה.

חילוק בינארי

החילוק (כמו כל שאר הפעולות האריתמטיות), בכל שיטות הספירה, מתבצע בדומה לשיטת החילוק העשרוני, לדוגמה. לוח החילוק הבינארי כולל שתי אפשרויות בלבד: *

$$\frac{0}{1} = 0$$

$$\frac{1}{1} = 1$$

כדי לבצע פעולת חילוק אנו נעזרים בפעולת חיסור, בדומה לשימוש בפעולת החיבור בזמן ביצוע כפל.

דוגמה 3.1

נחשב את המנה של 100001:11.

$$\begin{array}{rrrrrrl}
 & & 1 & 0 & 1 & 1 & \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & \big| \; 1 \; 1 \\
- & 1 & 1 & & & & \\
\hline
 & & 1 & 0 & 0 & & \\
 & - & & 1 & 1 & & \\
\hline
 & & & & 1 & 1 & \\
 & & & - & 1 & 1 & \\
\hline
 & & & & & 0 &
\end{array}$$

השלבים שנבצע בפעולת החילוק הם (בהנחה שהמחולק גדול מהמחלק);

* בלוח החילוק הבינארי לא כללנו את המנות $\frac{1}{0}$; $\frac{0}{0}$ כי החילוק ב-0 אינו מוגדר.

א. נבודד מהמחולק (משמאל לימין) קבוצה קטנה ביותר של ספרות בינאריות המייצגות מספר בינארי גדול או שווה למחלק (במקרה הנ״ל $100 > 11$). נרשום 1 בתוצאה בספרה המשמעותית ביותר (שימו לב, שזה פשוט יותר מחילוק בשיטה העשרונית).

ב. נכפול את המחלק ב-1 (שוב – פשוט יותר מאשר בשיטה העשרונית) ונחסר את התוצאה מהקבוצה שבודדנו (בדוגמה לעיל $100 - 11 = 1$).

ג. לתוצאה המתקבלת מהחיסור נוסיף מימין את הסיבית הסמוכה, מימין לקבוצה שבודדנו (במקרה שלנו, מאחר ש-$10 < 11$, הורדנו שתי סיביות ואז: $100 > 11$ ובתוצאה נרשום 0 בספרה השנייה).

ד. נחלק את המספר המתקבל במחלק, ונמשיך את התהליך עד שהמחולק יתאפס.

הערות

א. גם בחילוק בינארי יכול להיווצר מצב שבו תוצאת החיסור האחרון אינה אפס. במקרה כזה נקבל מנה ושארית, לדוגמה:

```
        1 0 0 1
    ___________
    1 0 1 1 1 0 | 1 0 1
  - 1 0 1
    _____
        0 1 1 0
      -   1 0 1
        _______
              1   שארית
```

מתוצאת החלוקה קיבלנו את המנה 1001_2 ואת השארית 1_2.

ב. אילו היה עלינו לחלק מספר מעורב (שלם ושבר) במספר מעורב, היינו מזיזים את הנקודה גם במחלק וגם במחולק, עד קבלת מספר שלם במחלק, ואז מבצעים את החישוב (פעולת הרחבה דומה לזו שאנו מבצעים בשיטה העשרונית).

ג. גם בחילוק, ניתן להתייחס למקרה שבו המחולק הוא חזקה של הבסיס. במקרה כזה החלוקה תתבצע על-ידי-הזזה ימינה ומיקום הנקודה העשרונית בתוצאת החילוק בהתאם לגודל החזקה. לדוגמה, נחלק מספר ב- $1000 = 10^3$:

$100110100000_2 : 1000_2 = 100110100_2$

$10011010_2 : 1000_2 = 10011.01_2$

שאלה 3.7

בצעו את פעולות החילוק הבאות (המספרים מוצגים בבסיס בינארי):

א. $1001100:100 = ?$ ב. $1000:100 = ?$

שאלה 3.8

בצעו את החישוב הבא, תוך הצגת המספרים לפי בסיס בינארי:

$$(12 + 15) \times (20 - 10)$$

בְּדקו את החישוב על-ידי ביצוע הפעולות לפי בסיס עשרוני.

3.2 שיטות לייצוג מספרים בינאריים מכוונים

כדי להרחיב את פעולות החשבון על תחום המספרים השליליים, עלינו לדון תחילה בשיטות הייצוג של המספרים השליליים בבסיס הבינארי.

3.2.1 שיטת הגודל והסימן

אנו רגילים לסמן מספר עשרוני שלילי באמצעות סימן מינוס משמאל למספר. אולם, לצורך שימוש בבסיס הבינארי במחשב, אנו צריכים למצוא שיטה לייצוג הסימן (חיובי או שלילי) באמצעות הסמלים 0 ו-1. שיטה אחת להוספת הסימן + או – למספר בינארי, היא להוסיף סיבית נוספת משמאל למספר; כאשר מוסיפים את הסיבית 0 משמאל למספר, היא מסמנת מספר חיובי, וכאשר מוסיפים את הסיבית 1 משמאל למספר, היא מסמנת מספר בינארי שלילי. שיטה זו להצגת מספרים מכוונים נקראת שיטת **הגודל והסימן**.

כבר ראינו שהמחשב משתמש ביחידות מידע שהגודל שלהן קבוע. כך לדוגמה, לאחסון של מספר מכוון בבית אחד שיש בו 8 סיביות, משמשת הסיבית המשמעותית ביותר כסיבית הסימן ובשאר 7 הסיביות מאוחסן המספר עצמו. כלומר, אפשר לרשום את כל המספרים החיוביים בבית שיש בו 8 סיביות, בין **0**0000000 לבין **0**1111111, ואת כל המספרים השליליים בין **1**0000000 לבין **1**1111111 (כאן הדגשנו את סיבית הסימן).

שאלה 3.9

רשמו את תחום המספרים החיוביים ואת תחום המספרים השליליים שניתן להציג במילה.

מכאן, שכדי לבנות את הייצוג הבינארי של המספרים השלמים, **החיוביים והשליליים**, שערכם המוחלט בין 0 ל-7, נזדקק ל-4 סיביות: שלוש עבור אחסון ערכם המוחלט של המספרים וסיבית נוספת (הרביעית) שהיא הסיבית השמאלית ביותר – עבור הסימן. כלומר לייצוג תחום המספרים בין 7 ל-7– נזדקק ל-4 סיביות. בטבלה 3.4 רשומים 15 מספרים בינאריים, חיוביים ושליליים, שנבנו בשיטה שתיארנו.

טבלה 3.4
ייצוג מספרים בשיטת הגודל והסימן

מספרים שליליים		מספרים חיוביים	
מספר בינארי	**ערך עשרוני**	**מספר בינארי**	**ערך עשרוני**
1000	0	0000	0
1001	–1	0001	1
1010	–2	0010	2
1011	–3	0011	3
1100	–4	0100	4
1101	–5	0101	5
1110	–6	0110	6
1111	–7	0111	7

בין שני מספרים, הרשומים באותה שורה בטבלה, קיים הבדל רק בסיבית הסימן. שאר הסיביות, המייצגות את הערך המוחלט של המספרים, זהות. כפי שניתן לראות בטבלה, לספרה 0 יש שני ייצוגים: 1000 ו-0000. על בעיה זו ניתן להתגבר בצורה פשוטה (ולהחליט למשל שאפס ייוצג כ-0000), אולם ביצוע פעולות חישוב (כמו חיבור) נעשה מסורבל ולא יעיל בייצוג זה. נדגים טענה זו בעזרת כמה דוגמאות.

לדוגמה, נוסיף 1 לצירוף 1100_2 המסמל את המספר השלילי -4:

$$\begin{array}{r} 1100_2 \\ + \quad 1_2 \\ \hline 1101_2 \end{array}$$

מתקבל הצירוף 1101_2 המסמל את -5

אולם התוצאה הייתה צריכה להיות

$$-4 + 1 = -3$$

נבצע בדיקה נוספת: נחבר למספר 1 (0001_2)

את המספר -1 (1001_2)

התוצאה המתקבלת היא 1010 ולא אפס!

בדיקת מספרים נוספים תביאנו למסקנה, שתוצאות רבות שגויות.

שאלה 3.10

בדקו בטבלה 3.4 ורשמו מה תוצאת החיבור של $a + (-a)$, כאשר a מייצג ספרה בין 0 ל-7.

אם נשתמש בשיטה זו לסימון מספרים, נצטרך לבדוק, לפני כל חישוב, את סימנם של המספרים ובהתאם להחליט אם יש לבצע חיבור או חיסור. בדיקות אלה מאריכות את זמן החישוב, ומאחר שהפעולות חיבור וחיסור הן פעולות בסיסיות במחשב, נרצה להשתמש בפתרון יעיל יותר.

כדי ליעל את ביצוע הפעולות חיבור וחיסור, משתמשים בשני המקרים בפעולת חיבור; אבל כאשר יש לחסר מספר, נשתמש בחיבור של המספר הנגדי לו. אם נרצה, למשל, לחסר מספר חיובי, נשתמש בנגדי לו, שהוא מספר שלילי, בצורה זו (b מייצג כאן מספר חיובי):

$$a - b = a + (-b)$$

כלומר, תחילה מייצגים את המספר הנגדי ל-b, שהוא מספר שלילי, ואז מבצעים פעולת חיבור פשוטה.

שיטת המשלים לשתיים

כאמור, אנו מחפשים ייצוג למספר שלילי שבו תוצאת החיבור של $a + (-a)$ תהיה 0.
לדוגמה אנו מחפשים ייצוג שלילי למספר 3 המיוצג באמצעות 4 סיביות:

$$\begin{array}{r} +\ 0011_2 \\ ????_2 \\ \hline 0000_2 \end{array}$$

בדוגמה, זו אם נייצג (-3) כ-1101_2 ונחבר אותו ל-3

$$\begin{array}{r} +\ \boxed{0011}_2 \\ \boxed{1101}_2 \\ \hline 1\boxed{0000}_2 \end{array}$$

נקבל 5 סיביות, אך מאחר והמספר מיוצג ב-4 סיביות, אנו לא מתייחסים לסיבית החמישית (שהיא 1).

דרך פשוטה יותר לבצע את החישוב היא לבצע את החישוב $-a = 0 - a$, כאשר 0 מיוצג כ-n הסיביות הנמוכות של 2^n. בדוגמה שלנו אנו משתמשים בארבע הסיביות הנמוכות של $1\boxed{0000}$ לייצוג 0 וכדי לחשב את (-3) חשבנו את הייצוג הבינארי של $2^4 - 3$ שהוא 1101_2.
נבדוק לדוגמה כיצד לייצג את המספר -3 ונחשב $0 - 3$. התוצאה שקבלנו היא המספר 1101_2 והלווה הוא 1:

1 (לווה)

$$\begin{array}{r} -\ 0000_2 \\ 0011_2 \\ \hline 1101_2 \end{array}$$

בתוצאה זו הסיבית החמישית המשמעותית ביותר היא הלווה, שממנו אנו מתעלמים.

שיטת ייצוג זו נקראת שיטת **המשלים לשתיים** (two's complement). טבלה 3.5 מציגה את המספרים בשיטת המשלים ל-2 עבור 4 סיביות.

טבלה 3.5
ייצוג מספרים בשיטת המשלים ל-2

ייצוג	ערך בשיטת המשלים לשתיים	ערך כמספר בלתי מכוון	ייצוג	ערך בשיטת המשלים לשתיים	ערך כמספר מכוון
0000	0	0	1000	–8	8
0001	1	1	1001	–7	9
0010	2	2	1010	–6	10
0011	3	3	1011	–5	11
0100	4	4	1100	–4	12
0101	5	5	1101	–3	13
0110	6	6	1110	–2	14
0111	7	7	1111	–1	15

כדי למצוא את הערך הנגדי למספר בן n סיביות ניתן להשתמש בתהליך האלגוריתמי הזה:

1. נהפוך את הערך של כל סיבית במספר, כלומר, סיבית שערכה 1 נהפוך ל-0 וסיבית שערכה 0 נהפוך ל-1.
2. נוסיף 1 למספר שנוצר.

לדוגמה נחשב את הייצוג של המספר 5– בשיטת המשלים ל-2:

המספר 5	0101_2
הפיכת הספרות	1010_2
הוספת של 1	$+ \quad 1_2$
המספר 5–	1011_2

נציג דוגמה נוספת לייצוג של 25–

חישוב הערך הבינארי של 25:	0 0 0 1 1 0 0 1
הפיכת הסיביות:	1 1 1 0 0 1 1 0
הוספת 1:	+ 1
תוצאה: הערך הבינארי של 25– בשיטת המשלים ל-	1 1 1 0 0 1 1 1

בדיקה:

$$
\begin{array}{r}
00011001_2 \\
+\ \ 11100111_2 \\
\hline
1\boxed{00000000}_2
\end{array}
$$

לסיום נציג שיטה נוספת לביצוע החישוב, אך תחילה נביא דוגמה שבה נרשום מספר בשיטת המשלים לשתיים. בשיטה זו אנו מחפשים, החל מהסיבית הפחות משמעותית, את הסיבית הראשונה שהיא 1, והופכים כל סיבית, החל מהסיבית שעוקבת לה, עד הסיבית המשמעותית ביותר, בהתאם. לדוגמה נציג את המספר $\mathbf{01}010000_2$. בדוגמה זו נהפוך את הסיביות, החל מהסיבית השישית (אלה הסיביות המודגשות) ונקבל 10110000_2, שהוא המשלים לשתיים.

נרשום כעת אלגוריתם המתאים להיפוך סימן בשיטת המשלים לשתיים:

בהינתן מספר בן n סיביות בצע את הפעולות האלה:
1. עבור על כל הסיביות של המספר החל מהספרה הפחות משמעותית, עד שתגיע לסיבית הראשונה שהיא 1
2. הפוך את ערכן של כל הסיביות משמאלה של הסיבית שנמצאה בשלב 1.

נציג כמה דוגמאות נוספות המשתמשות באלגוריתם זה למציאת מספר בשיטת המשלים לשתיים:

א. נמצא את המספר הנגדי ל-25.

ראשית, $00011001_2 = 25$

הסיבית הימנית ביותר היא 1, לכן נשנה ערכי כל הסיביות, החל מהסיבית השנייה מימין, ונקבל 11100111_2, שהוא הייצוג של -25 בשיטת המשלים לשתיים.

ב. נמצא את המספר הנגדי ל-64.

ראשית, $01000000_2 = 64$

הסיבית ה-7 היא הסיבית הראשונה שערכה 1, ולכן נשנה את ערכה של הסיבית השמינית בלבד ונקבל 11000000_2 שהוא הייצוג של -64 בשיטת המשלים לשתיים.

נשתמש באותה שיטה גם כדי לפענח מספר שלילי המיוצג בשיטת המשלים לשתיים. לדוגמה: המשלים לשתיים של המספר $a=11110101_2$ הוא 00001011_2, כלומר $a = -11$.

שאלה 3.11

בשאלה זו, הניחו כי אורכו של כל מספר בינארי נתון, מוגבל ל-8 סיביות בלבד.

א. חשבו את המשלים לשתיים של המספרים הבאים:

(1) 10111001_2 (2) 01011011_2 (3) 110110101_2

ומצאו את הייצוג העשרוני של כל מספר.

ב. מהו המשלים לשתיים של המספר 00000000_2?

ג. חשבו, איזה מספר עשרוני מכוון מייצג המספר הבינארי 11111111_2? ואיזה מספר עשרוני בלתי מכוון הוא מייצג?

3.3 חיבור וחיסור מספרים בינאריים שלמים מכוונים

בסעיף הזה נציג פעולות חיבור וחיסור של מספרים שלמים מכוונים.

3.3.1 חיבור מספרים מכוונים

נדגים חיבור של מספרים מכוונים ונבחן שני מקרים: מקרה ראשון – כאשר תוצאת החיבור היא חיובית; מקרה שני – כאשר התוצאה שלילית. בכל הדוגמאות שנביא נניח כי המספרים הבינאריים הם בני 8 סיביות (בית).

בכל הדוגמאות שלהלן, A ו-B מציינים את הערכים המוחלטים של המספרים, ולכן $-B$, למשל, מציין מספר שלילי.

דוגמה 3.2 A + (–B) כאשר התוצאה חיובית

נחבר $23 + (-13)$

פתרון

תחילה נמיר את הערך המוחלט של כל מספר לייצוג בינארי ונרשום אותו באמצעות 8 סיביות:

$$23 = 00010111_2$$

$$13 = 00001101_2$$

המספר 13 הוא שלילי, לכן נשתמש באלגוריתם להיפוך הסימן בשיטת המשלים לשתיים:

$$-13 = 11110011_2$$

ביצוע החיבור:

$$\begin{array}{r} 00010111_2 \\ +\ \underline{11110011_2} \\ 1\boxed{00001010}_2 \end{array}$$

נתעלם מהסיבית התשיעית שהתקבלה. סיבית זו נקראת **נשא סופי** (end-carry) והיא שקולה להוספת 100000000 לתוצאה, או, במילים אחרות, הוספת המספר אפס לתוצאה. סיבית הסימן היא הסיבית השמינית; כיוון שערכה 0 היא מציינת שהמספר חיובי, כלומר התוצאה היא:

$$1010_2 = 10$$

דוגמה 3.3 A + (–B) כאשר התוצאה היא שלילית

נחבר $18 + (-45)$

פתרון

תחילה נמיר את הערך המוחלט של כל מספר לייצוג בינארי ונרשום אותו באמצעות 8 סיביות:

$$18 = 00010010_2$$

$$45 = 00101101_2$$

המספר השני הוא שלילי, ולכן נשתמש באלגוריתם להיפוך הסימן בשיטת המשלים לשתיים:

$$-45 = 11010011_2$$

ביצוע החיבור:

$$\begin{array}{r} 00010010_2 \\ +\ \underline{11010011_2} \\ 11100101_2 \end{array}$$

הפעם לא התקבל נשא סופי והסיבית השמינית היא 1, כלומר היא מציינת שהתוצאה שלילית (זכרו, שאנו עובדים בשיטת המשלים לשתיים). כדי למצוא את המספר נשתמש שוב באלגוריתם להיפוך הסימן בשיטת המשלים לשתיים, ונקבל:

$$00011011_2 = -27$$

גם בחיבור מספרים מכוונים תיתכן גלישה, כלומר, תוצאת החיבור אינה ניתנת לייצוג ב-8 סיביות. נציג שני מקרים בהם מתרחשת גלישה.

- כאשר סכום שני מספרים חיוביים התקבל כשלילי.
  לדוגמה:

$$53 + 120 = 173$$

אבל, בחישוב בינארי, כאשר המחוברים מיוצגים בשיטת המשלים לשתיים, חיברנו שני מספרים שסיבית הסימן שלהם היא 0, וקיבלנו מספר שסיבית הסימן שלו היא 1:

הייצוג של 53 הוא	$\mathbf{0}0110101_2$
הייצוג של 120 הוא	$\underline{\mathbf{0}1111000_2}$
	$\mathbf{1}0101101_2$

התוצאה שהתקבלה היא שגויה; בייצוג עשרוני המספר הוא 83–.

- כאשר הסכום של שני מספרים שליליים מתקבל כחיובי.
  לדוגמה:

$$(-64) + (-80) = (-144)$$

ייצוג בשיטת המשלים לשתיים של המספר 80– הוא 10110000_2

ייצוג בשיטת המשלים לשתיים של המספר 64– הוא 11000000_2

תוצאת החיבור היא: $1\boxed{\mathbf{0}1110000}_2$

סיבית הסימן 0 (הסיבית השמינית) מלמדת שהתוצאה שגויה, כי היא חיובית.

3.3.2 פעולת חיסור של מספרים מכוונים

מבחינה מתמטית, ניתן לרשום פעולת חיסור גם בצורה הזו:

$$b = a + (-b) - a$$

לכן, כדי לבצע פעולת חיסור במספרים בינאריים נשתמש בייצוג בשיטת המשלים לשתיים כדי למצוא את הנגדי למחסר, ונבצע חיבור רגיל.

דוגמה 3.4

נחסר $(-23) - 13-$.

פתרון

$$(-13) - (-23) = (-13) + 23$$

כדי לבצע פעולת חיסור, עלינו לחבר למחוסר את המספר הנגדי של המחסר.

$$\begin{array}{r} -13 \\ +\ \ 23 \\ \hline +10 \end{array} \qquad\qquad \begin{array}{rr} -13 = & \mathbf{1}1110011_2 \\ +\ \ 23 = & \mathbf{0}0010111_2 \\ \hline 10 = & 1\boxed{00001010}_2 \end{array}$$

שאלה 3.12

נתונים: $U = 10110111_2$; $V = 01100011_2$; $W = 10011110_2$

(U, V, W הם מספרים מכוונים בשיטת המשלים ל-2). חשבו את ערכם של הביטויים שלפניכם, בשיטה העשרונית ובשיטה הבינארית:

א. $Y = U + V + W$

ב. $Z = U + V - W$

שאלה 3.13

נתונים: $B = -105_{10}$, $A = 223_8$

חשבו את ערכים של הביטויים שלהלן בשיטה הבינארית:

א. $S = A + B$

ב. $X = A - B$

3.4 תחום הייצוג של מספרים בינאריים שלמים מכוונים ובלתי מכוונים

בכתיבת תכניות חשוב לדעת את תחום המספרים שיחידת אחסון יכולה להכיל, ולבחור יחידת אחסון שתתאים לנתונים שיעובדו בתכנית.

באופן כללי, אלה סוגי המספרים שאפשר להציג ביחידת מידע שיש בה n סיביות:

- כל המספרים הבלתי מכוונים מ-0 עד $2^n - 1$
- כל המספרים המכוונים בין $(2^{n-1} - 1)+$ ל-(-2^{n-1}).

לדוגמה נתייחס ליחידת אחסון בגודל של בית אחד:

– תחום המספרים **הבלתי מכוונים** שניתן לאחסן בבית הוא בין 00000000_2 ל- 11111111_2 והערך העשרוני הוא בין 0 לבין 255. אפשר לחשב את המספר העשרוני הגדול ביותר שניתן לאחסן בבית, בצורה הזו: $2^8 - 1 = 256 - 1$.

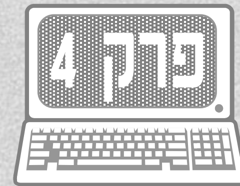
– תחום המספרים **המכוונים** שניתן לאחסן בבית בשיטת הגודל והסימן הוא בין 01111111_2 (שהוא המספר החיובי הגדול ביותר) ל-11111111_2 (שהוא המספר השלילי הקטן ביותר) והערך עשרוני הוא בין 127+ לבין (128–). אפשר לחשב את המספרים העשרוניים בצורה הזו:

המספר הגדול ביותר הוא $127 = 128 - 1 = 2^7 - 1$

המספר הקטן ביותר הוא $-128 = -2^7$

שאלה 3.14

חשבו את תחום המספרים הבלתי מכוונים ואת תחום המספרים המכוונים, שניתן לאחסן במילה שגודלה 16 סיביות. הציגו את המספרים בשיטה הבינארית, בשיטה ההקסדצימלית ובשיטה העשרונית.



שפת אסמבלי והמודל התכנותי של מעבד 8086

4.1 מבוא

ההוראות בשפה נמוכה כמו שפת אסמבלי, כוללות פנייה מפורשת לרכיבים במחשב, כגון אוגרים ותאי זיכרון. לכן עלינו להכיר את הארכיטקטורה של המעבד שבו נשתמש, לפני שניגש לכתיבת תכנית עבורו. בפרק זה נציג את המודל התִּכנותי של מעבד 8086, מנקודת המבט של מתכנת בשפת אסמבלי, ונתייחס למרכיביו השונים: נתאר את האוגרים אליהם מתייחסים בהוראות, את ארגון הזיכרון וכיצד מגדירים כתובת של תא בזיכרון. לסיום נתאר את המבנה של תכנית בשפת אסמבלי ונסביר כיצד מגדירים משתנים.

המעבד שנציג בספר זה הוא מעבד 8086 שייצרה חברת אינטל בשנת 1978. חברת אינטל החלה בפיתוח מעבדים למחשבים אישיים בשנת 1970, ומאז היא הוציאה לשוק גרסאות משופרות של המעבדים. כל גרסה כזו נחשבת ל-"דור". שמות המעבדים בדורות הראשונים* אופיינו על-ידי מספרים, המתחילים בספרות 80 לדוגמה: 8086, 80186, 80286, 80386, 80486. החל מן המעבד 80487 התחילה חברת אינטל לכנות את המעבדים בשם "פנטיום" (פנטה = 5 ביוונית). משפחת המעבדים של אינטל מכונה בקיצור **משפחת x86** וכך נכנה אותה בספר זה.

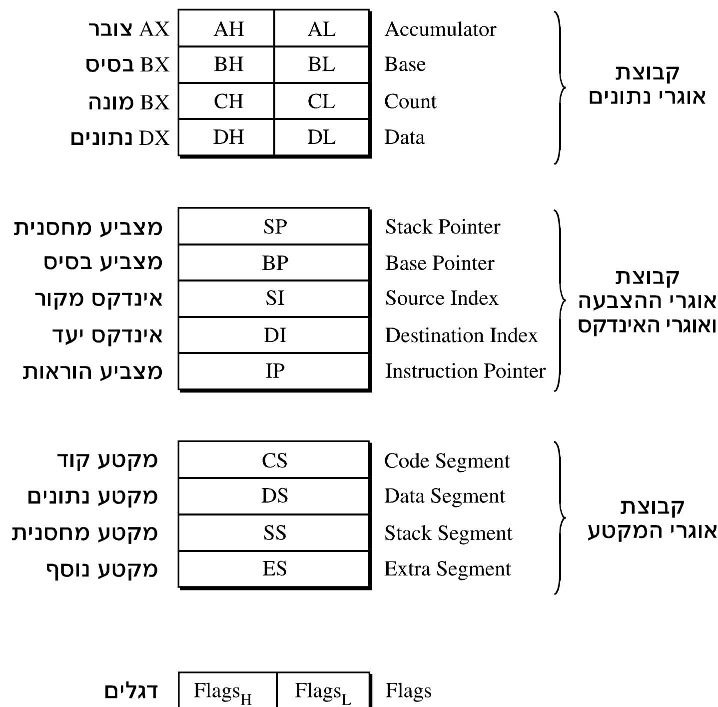
כל דור חדש של מעבדים במשפחה זו בא לשפר את כושר העיבוד של המעבדים מן הדור הקודם ויחד עם זאת נשמר עקרון התאימות (Compatibility). לפי **עקרון התאימות**, כל מחשב המבוסס על מעבד מדור חדש, מסוגל להריץ גם תכניות הכתובות למעבדים מדורות קודמים. כך הבטיחה חברת אינטל שמשתמשים המחליפים את המחשב שלהם, לא יצטרכו לשכתב את כל התכניות, כמובן בתנאי שהמחשב מתאים לארכיטקטורה של מעבדי אינטל. כדי לשמור על תאימות, קבוצת ההוראות של המעבדים החדשים כוללת גם את קבוצת ההוראות של 8086, וניתן להגדיר צורת עבודה המתאימה למעבד 8086.

---

* משפחת המעבדים הראשונה שפיתחה חברת אינטל נקראה משפחת 4000. כיום מעבדים אלו אינם נמצאים בשימוש.

4.2 המודל התכנותי של ה-8086

המודל התִּכנותי המוצג באיור 4.1, מתאר את האוגרים הפנימיים אליהם מתייחסים בשפת אסמבלי. במעבד 8086 יש 14 אוגרים שמסוגלים לאחסן מילה בת 16 סיביות (כרוחב אפיק הנתונים). מחלקים את האוגרים לארבע קבוצות, בהתאם לתפקידים השונים שהם ממלאים בביצוע תכנית. בסעיף זה נאפיין בקצרה כל קבוצה של אוגרים לפי גודל האוגר, סוג הנתונים שניתן לאחסן בו והתפקידים שהוא ממלא בביצוע התכנית.



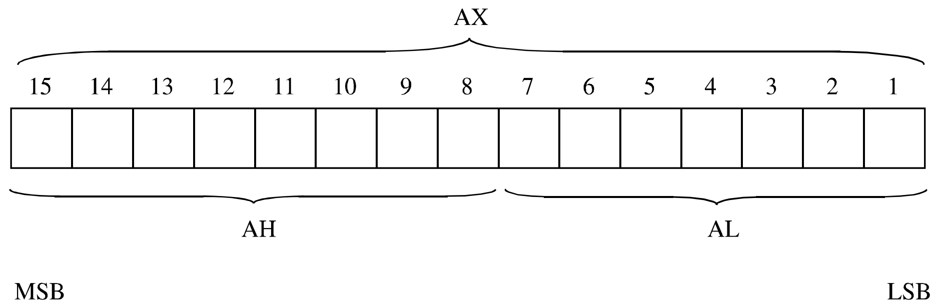
איור 4.1
האוגרים במעבד 8086

קבוצת אוגרי הנתונים

אוגרי הנתונים הם אוגרים למטרות כלליות, המשמשים כאופרנדים בהוראה; מאחסנים בהם תוצאות ונתוני ביניים או כתובות של תאי זיכרון (והתקני קלט/פלט). בקבוצה זו יש ארבעה אוגרים: AX, BX, CX ו-DX.

חלק מן ההוראות בשפת אסמבלי יכולות להשתמש בכל אחד מהאוגרים לביצוע הפעולה. לדוגמה: בהוראת חיבור האופרנדים אפשר לאחסן את המחובר ו/או את המחבר, בכל אחד מהאוגרים של קבוצה זו, ולרשום את ההוראה ADD AX, BX או ADD CX, DX. אך יש הוראות בשבילן דרוש אוגר מסוים לביצוע ההוראה, ולכן לאוגרים אלה מוצמד שם נוסף, המעיד על תפקידו המיוחד בהוראות אלה. האוגר AX, נקרא גם צובר (Accumulator), ואוגר DX נקרא גם אוגר נתונים (Data register); כאשר מבצעים הוראות קלט/פלט, שומרים באוגר AX את הנתון שנקלט או את הנתון שמוצג כפלט, ובאוגר DX שומרים את כתובת התקן הקלט/פלט. כאשר מבצעים הוראות הזזה או לולאה, משתמשים באוגר CX הנקרא מונה (Counter register). האוגר BX שנקרא אוגר הבסיס (Base register), משמש גם כמצביע בהוראות הכוללות פנייה לקבוצה של תאים בזיכרון (מערך).

כפי שציינו, כל האוגרים בקבוצה זו הם בגודל מילה (16 סיביות). אולם כפי שמראה איור 4.2, אפשר להתייחס אל אוגר הנתונים כאל אוגר המכיל בתוכו שני אוגרים נפרדים, שכל אחד מהם בגודל של בית (8 סיביות). חלקו התחתון של האוגר (סיביות 0 עד 7) משמש כאוגר אחד, המצוין באמצעות האות L (קיצור של Low); חלקו העליון של האוגר (סיביות 8 עד 15) משמש כאוגר שני, המצוין באמצעות האות H (קיצור של High). האותיות L ו-H מחליפות את האות X באוגר השלם. כך אפשר, למשל, לאחסן באוגר AX מילה בת 16 סיביות ואחר כך להעתיק את הבית העליון של המילה לאוגר אחר.



איור 4.2
מבנה האוגר AX

לדוגמה, נתאר את השימוש באוגר AX באמצעות ההוראה MOV (עליה נרחיב בהמשך) המעתיקה את הנתון שבאופרנד המקור אל אופרנד היעד:

העתקת המילה FF00h לאוגר AX　　mov ax, 0FF00h

העתקת הבית התחתון AL (של האוגר AX) לאוגר BL　　mov bl, al

בסיום ערכם של האוגרים BL ו-AL זהה, כלומר: BL = AL = 00h.

שימו לב, אי-אפשר להשתמש באותה הוראה באופרנד אחד שהוא אוגר בגודל בית ובאופרנד אחר שהוא אוגר בגודל מילה. למשל ההוראה:

```
mov bl, ax
```

היא שגויה.

קבוצת אוגרי ההצבעה ואוגרי האינדקס

אוגרי ההצבעה ואוגרי האינדקס הם בגודל של מילה ולא ניתן לגשת בנפרד לבית העליון או לבית התחתון שלהם. האוגרים האלה מכילים כתובות והם משמשים כמצביעים לתא בזיכרון. בקבוצה זו יש חמישה אוגרים: SP, BP, SI, DI ו-IP.

נפרט בקצרה את תפקידי האוגרים האלה:

- **מצביע ההוראות** הוא האוגר Instruction Pointer) IP) – הוא מצביע על כתובתו של תא בזיכרון ממנו תובא ההוראה הבאה בתכנית. בניגוד לשאר אוגרי ההצבעה, הוא אינו בשליטת המתכנת, כלומר, אי אפשר לכתוב הוראה שבה IP הוא אופרנד.
- **מצביע המחסנית** הוא האוגר Stack Pointer) SP) – הוא מצביע על כתובתו של תא שנמצא באיזור מיוחד של הזיכרון הנקרא מחסנית (stack). על מחסנית נרחיב בפרק 7.
- האוגרים BP, SI, DI – מכילים כתובות של תאי זיכרון ומשמשים, בין השאר, כמצביעים לקבוצה של תאי זיכרון (מערך). על תפקידם של אוגרים אלה נרחיב בפרק 6.

אוגר הדגלים

אוגר הדגלים גם הוא בגודל של מילה, אולם להבדיל משאר האוגרים בהם מתייחסים לתוכן האוגר כמקשה אחת, באוגר הדגלים מתייחסים בנפרד לתוכן של כל סיבית יחידה. כל סיבית באוגר הדגלים נקראת "דגל", ויש לה גם שם שבאמצעותו אפשר לפנות אליה בנפרד. במעבד 8086 מנוצלות רק תשע סיביות (מתוך 16); מחלקים אותן לשני סוגים:

- **דגלי המצב** (status flags) שתפקידם לאחסן תכונות של תוצאות החישוב האריתמטי או החישוב הלוגי הנוגע להרצת התכנית בהמשך.
- **דגלי בקרה** (control flags) שתפקידם לבקר את אופן הפעולה של המעבד.

שמות הדגלים והתכונות שהם מייצגים מתוארים בטבלה 4.1. בפרקים הבאים נסביר כיצד משתמשים בדגלים. שימו לב, שהמעבד קובע את ערך דגלי המצב בהתאם לתוצאת הפעולה האחרונה שהתבצעה.

קבוצת אוגרי המקטע

בקבוצה זו ארבעה אוגרי מקטע: CS, DS, SS ו-ES, שגודל כל אחד מהם הוא מילה. אוגרי המקטע מכילים כתובות; המעבד משתמש בהם לחישוב הכתובת של תא בזיכרון, אליו הוא צריך לפנות כדי לקרוא או לכתוב. בסעיף הבא נרחיב את ההסבר על תפקיד אוגרי המקטע.

טבלה 4.1
תיאור הדגלים

תיאור	שם הדגל	מיקום הסיבית באוגר הדגלים	סוג הדגל
דגל הנשא (Carry Flag): מקבל '1' כאשר יש נשא או לווה מהסיבית העליונה – אחרת הדגל יתאפס	CF	0	מצב
דגל הזוגיות (Parity Flag): מקבל '1' כאשר 8 הסיביות התחתונות בתוצאה מכילות מספר זוגי של סיביות '1' – אחרת הדגל יתאפס	PF	2	מצב
דגל נשא-העזר (Auxiliary Flag): מקבל '1' כאשר יש לווה או נשא מ-4 הסיביות התחתונות של AL – אחרת הדגל יתאפס	AF	4	מצב
דגל האפס (Zero Flag): מקבל '1' כשהתוצאה היא אפס – אחרת הדגל יתאפס	ZF	6	מצב
דגל הסימן (Sign Flag): מקבל את ערך הסיבית העליונה של התוצאה ('0' כשהתוצאה חיובית; '1' כשהיא שלילית)	SF	7	מצב
דגל המלכודת (Trap Flag): (מכונה גם 'דגל צעד יחיד') כאשר הוא נקבע כ-'1', מתבצעת פסיקת צעד-יחיד לאחר ביצוע ההוראה הבאה. פסיקה זו תאפס את TF.	TF	8	בקרה
דגל אפשור הפסיקות (Interrupt-enable Flag): כאשר הוא נקבע כ-'1', הפסיקות הניתנות למיסוך יגרמו למיקרו-מעבד לבצע הוראות, החל מהמקום המצוין על-ידי וקטור הפסיקות	IF	9	בקרה
דגל הכיוון (Direction Flag): כאשר הוא נקבע כ-'1', בהוראות מחרוזת תתבצע הפחתה אוטומטית של אוגר האינדקס המתאים. כאשר הוא נקבע כ-'0' – יתבצע קידום אוטומטי.	DF	10	בקרה
דגל הגלישה (Overflow Flag): מקבל '1' כאשר משתנה היעד אינו מספיק רחב כדי להכיל את התוצאה – אחרת הדגל יתאפס.	OF	11	מצב

4.3 ארגון הזיכרון במעבד 8086

4.3.1 הזיכרון הראשי

יחידת הזיכרון של המחשב משמשת לאחסון נתונים ותכניות. חלק מן הנתונים מאוחסנים בה לפני ביצוע התכנית, חלקם – במהלכה, וחלקם נשמרים בזיכרון גם לאחר שביצוע התכנית הסתיים. המאפיינים העיקריים של תא בזיכרון, אליהם אנו מתייחסים בכתיבת תכנית בשפת אסמבלי, הם: גודלו של תא זיכרון וכתובתו.

1. **הגודל של תא זיכרון** מגדיר את מספר הסיביות שמכיל תא אחד. במעבד 8086 גודלו של תא זיכרון הוא בית (byte), כלומר 8 סיביות, ובו ניתן לאחסן בו $2^8 = 256$ ערכים שונים. כדי לאחסן מספר גדול יותר של ערכים, ניתן להשתמש בכמה תאי זיכרון עוקבים על-פי הצורך. לדוגמה, כדי לאחסן 1000 ערכים שונים נשתמש בשני תאי זיכרון.
2. **הכתובת של תא בזיכרון** משמשת לזיהוי של התא. כדי לזהות תא, צריך להגדיר כתובת ייחודית, הנקראת ***כתובת פיזית*** או ***כתובת מוחלטת***. מספר הכתובות הפיזיות השונות שבהן ניתן להשתמש תלוי ברוחב אפיק הכתובות. במעבד 8086 רוחב אפיק הכתובות הוא 20 סיביות, לכן אפשר לרשום 2^{20} כתובות מוחלטות שונות שבעזרתן ניתן לפנות ל-1,048,576 תאים שונים, שגודל כל אחד מהם הוא בית.

הכתובת הראשונה בזיכרון היא 00000000000000000000_2 והכתובת האחרונה בו היא 11111111111111111111_2. כיוון שרישום כתובת בת 20 סיביות אינו נוח, משתמשים בדרך כלל בייצוג הקסדיצמאלי. בייצוג זה, הכתובת הפיזית הראשונה במפת הזיכרון היא 00000h והכתובת האחרונה היא FFFFFh.

שאלה 4.1

א. חשבו כמה כתובות שונות אפשר לרשום במעבד פנטיום שרוחב אפיק הכתובות שלו הוא 32 סיביות.

ב. רשמו, בשיטה ההקסדצימאלית, את הכתובת הפיזית של התא הראשון ואת הכתובת הפיזית של התא האחרון במעבד פנטיום שרוחב אפיק הכתובות שלו הוא 32 סיביות.

כיום הזיכרונות מכילים מאות מיליארדי תאים ואף יותר; כדי לציין את גודלם משתמשים במספר גדולים סטנדרטיים להגדרת גודל הזיכרון . גדלים אלה מתוארים בטבלה 4.2.

טבלה 4.2
גדלים סטנדרטיים של זיכרון

יחידת מדידה	מספר תאי הזיכרון
Kilo	2^{10} = 1,024 = 1 K
Mega	2^{20} = 1,024 K = 1 M
Giga	2^{30} = 1,024 M = 1 G
Tera	2^{40} = 1,024 G = 1 T
Peta	2^{50} = 1,024 T = 1 P
Exae	2^{60} = 1,024 PB = 1 E

גדלים אלה מאפשרים לציין את גודל הזיכרון בצורה נוחה יותר. לדוגמה: במקום לציין שבמעבד 8086 יש 1,048,576 תאים, נרשום כי גודל הזיכרון הוא 1M. לעיתים מוסיפים לגדלים אלה את הסימול B המסמל בית (Byte). לדוגמה 1MB (קיצור של 1Mega Byte), וכך מגדירים, בנוסף למספר התאים בזיכרון, גם את הגודל של תא יחיד.

שאלה 4.2

חשבו כמה תאי זיכרון יש במעבד פנטיום-פרו שבו רוחב אפיק הכתובות הוא 36 סיביות; ציינו את התוצאה ביחידות של Giga.

הזיכרון של המחשב נקרא גם ***זיכרון ראשי*** כדי להבדילו מזיכרונות חיצוניים כמו דיסק קשיח או כונן תקליטורים (הנקראים זיכרונות משניים). פרק זה מתייחס לזיכרון הראשי בלבד.

הזיכרון הראשי כולל שני סוגים של רכיבים:

א. זיכרון לקריאה בלבד, הנקרא ***זיכרון* ROM** (Read-Only Memory). הנתונים בזיכרון זה אינם הולכים לאיבוד בעת כיבוי המחשב.

ב. זיכרון לקריאה וכתיבה, הנקרא ***זיכרון* RAM** (Random Access Memory – "זיכרון גישה אקראית"). תוכנו של זיכרון זה נמחק בעת כיבוי המחשב.

את מרחב הכתובות בזיכרון של מעבד 8086 מחלקים לשלושה אזורים:

1. האזור הראשון נמצא בתחום הכתובות 0 עד 640K. הוא נקרא גם "זיכרון קונווציונלי". אזור זה הוא זיכרון RAM, לתוכו טוענים את התכנית בשפת מכונה שרוצים להריץ.
2. האזור השני נמצא בתחום הכתובות שבין 640K ל-896K. תחום כתובות זה שייך לרכיבי קלט/פלט שונים כמו כרטיס המסך.
3. האזור השלישי נמצא בתחום הכתובות שבין 896K ל-1M. גם הוא זיכרון לקריאה בלבד (זיכרון ROM) והוא נקרא ROM BIOS (קיצור של Basic Input/Output System). באזור זה, יצרן המעבד צרב תוכנה (שאינה נמחקת) והיא מספקת שירותים לבדיקת החומרה (עם אתחול המחשב) ושירותים נוספים של מערכת ההפעלה (שחלק מהם נתאר בהמשך).

ההוראות בשפת אסמבלי מאפשרות לפנות לכל תא בכל שלושת האזורים ולקרוא ממנו נתון. אולם כתיבה של נתון אפשרית רק לתא זיכרון שנמצא באזור הראשון או השני.

שאלה 4.3

הסבירו מדוע חייבים לטעון דווקא לאזור הראשון את התכנית בשפת המכונה שרוצים להריץ.

שאלה 4.4

רשמו בשיטה ההקסדצימלית את הכתובת הפיזית של תאים אלה:

א. תא האחרון באזור הראשון;

ב. התא הראשון באזור השלישי;

ג. התא האחרון באזור השלישי.

4.3.2 חלוקת הזיכרון למקטעים (סגמנטציה)

במעבדי אינטל, הפנייה לתא בזיכרון נעשית בשיטה הנקראת **מיעון מקטעים** (segmented addressing). בשיטה זו, מרחב הזיכרון הליניארי מחולק בצורה לוגית למקטעים (segments), שגודלם המרבי הוא 64KB. הסיבה היא, שה-8086 הוא "מעבד 16 סיביות", דהיינו: אפיק הנתונים מכיל 16 סיביות, והאוגרים השונים הם בני 16 סיביות. מאחר שהאוגרים מכילים גם כתובות בזיכרון, הרי שהם יכולים למען ("מעון" – ציון המען, או

הכתובת) רק $64K = 65536 = 2^{16}$ בתים. כדי להתגבר על בעיה זו ולאפשר מיעון של כל ה-1MB בזיכרון, מחלקים את מרחב הזיכרון של המעבד 8086 לכמה מקטעים. בהמשך נראה כיצד ממענים כתובת כלשהי בזיכרון.

כל מקטע צריך להתחיל בכתובת פיזית שהיא כפולה של 16 בתים; כתובת כזו נקראת **פיסקה** (paragraph), כלומר, אפשר להתחיל מקטע מן הכתובת 16×0 שהיא הכתובת 00000h, או מן הכתובת 16×1 שהיא הכתובת 00010h, או מן הכתובת 16×2 שהיא הכתובת 00020h, וכך הלאה. שימו לב, שבכתובות אלה הספרה הפחות משמעותית היא תמיד 0. לכתובת של תחילת מקטע קוראים גם **כתובת הבסיס**.

שאלה 4.5

בזיכרון בגודל 1MB

א. רשמו את הכתובת הפיזית של מקטע המתחיל בכתובת 16×10.

ב. הפסקה האחרונה בזיכרון יכולה להיות בכתובת $1MB - 16$. רשמו כתובת זו בשיטה ההקסדצימלית.

נחשב את מספר הסיביות הדרושות כדי לציין כתובת של תא בתוך מקטע בגודל מרבי של 64K. ניעזר בטבלה 4.2 שם נוכל לראות כי $K = 2^{10}$ לכן:

$$64K = 64 \times 2^{10} = 2^{6} \times 2^{10} = 2^{6+10} = 2^{16}$$

כלומר, כדי לציין כתובת של תא במקטע מסוים, מספיקות 16 סיביות. לכתובת זו קוראים **כתובת יחסית** (Relative address), מפני שהיא יחסית לכתובת ההתחלה של המקטע והיא מציינת את המרחק ממנה (ולכן היא תמיד חיובית). לכתובת הזו יש שמות נוספים: **כתובת אפקטיבית** (Effective addres), **כתובת לוגית** (Logical address) והיסט (Offset). השם האחרון מצביע על כך שהכתובת מבטאת מרחק בבתים מתחילת המקטע.

שאלה 4.6

רשמו, בשיטה הקסדצימלית, את הכתובת היחסית הראשונה ואת הכתובת היחסית האחרונה במקטע שגודלו 64K.

כדי לחשב כתובת פיזית, המעבד משתמש בכתובת היחסית של התא ובכתובת הבסיס של המקטע אליו משתייכת הכתובת היחסית שציינו. שיטה זו דומה לארגון מספרי טלפון המחולקים לאזורי חיוג. בכל אזור חיוג יש מספרי טלפון רבים, וכל אחד מהם הוא בן 7

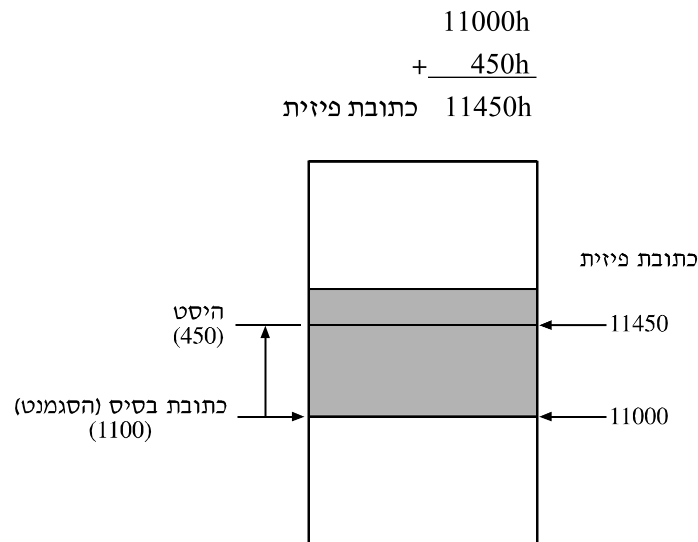
ספרות. זיהוי חד-משמעי (בדומה לכתובת פיזית) של מספר טלפון כולל ציון הקידומת של אזור החיוג (כתובת הבסיס) ומספר הטלפון (כתובת יחסית), לדוגמה 03-6666457. לעיתים אנו משתמשים גם בציון מספר טלפון בלבד, ללא קידומת. במקרה כזה אנו משתמשים בברירת מחדל, לפיה מספר הטלפון אותו חייגנו שייך לאזור החיוג ממנו אנו מחייגים.

במעבד 8086 כתובת יחסית וכתובת בסיס הן בנות 16 סיביות. אולם שמירה על עקרון זה גורמת לבעיה: כיצד ניתן לתרגם כתובות בנות 16 סיביות לכתובת פיזית של תא שהיא בת 20 סיביות? כדי לפתור בעיה זו בחישוב הכתובת הפיזית, המעבד מבצע שתי פעולות:

א. תחילה הוא מוסיף לכתובת הבסיס 4 סיביות: 0000 (או בהקסדצימאלי את הספרה 0h) שנרשמות כסיביות הפחות משמעותיות. תוספת זו שקולה להכפלת כתובת הבסיס ב-16.

ב. לאחר-מכן הוא מוסיף לתוצאה שהתקבלה את הכתובת היחסית.

לדוגמה, נתבונן באיור 4.3, בו כתובת הבסיס היא 1100h והכתובת היחסית היא 450h. המעבד מפרש את כתובת הבסיס כ-11000h והוא מוסיף לה את הכתובת היחסית 450h ובהתאם:

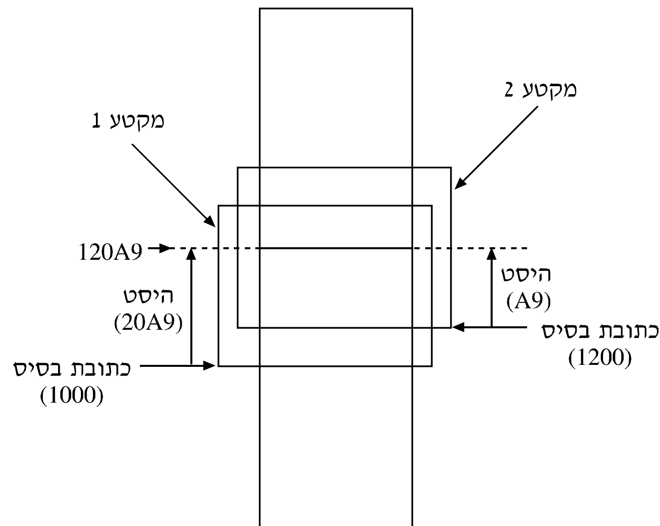


איור 4.3
הקשר בין הכתובת הפיזית 11450h והכתובת היחסית 450h

שאלה 4.7

חשבו את הכתובת הפיזית של תא שההיסט שלו הוא 100h וכתובת הבסיס שלו היא 2000h.

עד כה תיארנו מקטעים שמוגדרים בקטעים נפרדים בזיכרון; אבל ניתן להגדיר גם מקטעים שהם חופפים, כך שחלק מהתאים (או כולם) שייכים לשני מקטעים. לדוגמה, איור 4.4 מתאר שני מקטעים שהגודל של כל אחד מהם הוא 64KB.



איור 4.4
חפיפת מקטעים

כתובת הבסיס של המקטע הראשון היא 1000h וכתובת הבסיס של המקטע השני היא 1200h. ומהי הכתובת הפיזית של תא שההיסט שלו הוא 20A9h מכתובת הבסיס של המקטע הראשון? כתובתו הפיזית היא 120A9h. זו גם כתובתו הפיזית של תא שההיסט שלו הוא A9h מכתובת הבסיס של המקטע השני. כלומר, תא זה משותף לשני המקטעים, והוא יכול לשמש כנתון משותף לשתי תכניות שונות.

שאלה 4.8

א. חשבו את הכתובת הפיזית של תא שכתובתו היחסית היא A9h וכתובת הבסיס שלו היא 1000h.

ב. חשבו את הכתובת הפיזית של תא שכתובתו היחסית היא A9h וכתובת הבסיס שלו היא 1200h.

ג. האם שתי הכתובות היחסיות מצביעות על אותו תא בזיכרון?

ד. מתי אותה כתובת יחסית, השייכת לשני מקטעים שונים, תצביע על אותה כתובת פיזית בזיכרון?

עד כה תיארנו באופן כללי את השימוש במקטעים ואת אופן חישוב הכתובות; כעת נציג כיצד עיקרון המקטעים ממומש במעבד 8086. המעבד 8086 מקצה לתכנית ארבעה מקטעים (בגודל עד 64KB); לכל אחד מהמקטעים יש שם וייעוד שונה:

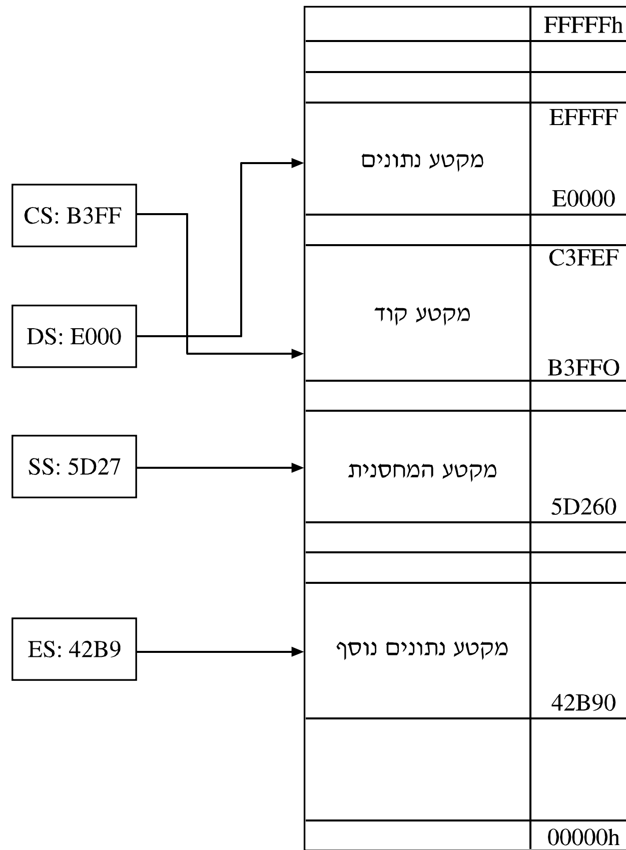
מקטע הקוד (Code Segment) – בו נשמרת התכנית שיש להריץ. בדרך-כלל זהו אזור רציף של כתובות המכילות את הוראות התכנית בשפת מכונה; לאזור הזה המעבד פונה בשלב ההבאה של מחזור ההבאה-ביצוע, כדי לקרוא את ההוראה שצריך לבצע.

מקטע הנתונים (Data Segment) – באזור הזה מאוחסנים המשתנים. המעבד פונה לאזור הזה בשלב הביצוע של מחזור ההבאה-ביצוע לצורך קריאה וכתיבה של נתונים שיש לעבד.

מקטע המחסנית (Stack Segment) משמש לניהול פרוצדורות (שגרות) וכאמצעי זיכרון לאחסון נתונים. על השימוש במקטע המחסנית נרחיב בפרק 7.

המקטע הנוסף (External Segment) הוא מקטע נתונים נוסף, המשמש בדרך-כלל לאחסון של נתונים המשותפים לכמה משימות.

כתובת הבסיס של כל מקטע, מאוחסנת באחד מארבעת אוגרי המקטעים: CS, DS, SS ו-ES. איור 4.5 מציג דוגמה של חלוקת הזיכרון למקטעים ואת הקשר בינם ובין אוגרי המקטע.



איור 4.5
הקצאת מקטעים לתכנית

כאשר אנו רוצים להריץ את התכנית, מערכת ההפעלה מקצה את המקטעים הדרושים ובהתאם מאותחלים גם אוגרי המקטע בכתובות הבסיס המתאימות. בדרך-כלל תוכן אוגרי המקטע אינו משתנה במהלך ביצוע התכנית, ולכן אנו מציינים בהוראות התכנית רק את הכתובת היחסית של תא בזיכרון. אפשר לרשום כתובת יחסית בצורה מפורשת או להשתמש באחד מאוגרי ההצבעה ואוגרי האינדקס. לדוגמה, התבוננו בהוראה:

```
mov  al, [si]
```

בהוראה זו מועתק לאוגר AL, התוכן של תא בזיכרון, עליו מצביע האוגר SI.

כדי לתרגם את הכתובת היחסית לכתובת פיזית, קיימת ברירת מחדל לכל הוראה הניגשת לזיכרון. ברירת המחדל מגדירה את המקטע ואת אוגר המקטע (המכיל את כתובת הבסיס

שאליו פונים. ברירת המחדל של האוגר IP היא מקטע הקוד, לכן כתובת הבסיס נמצאת באוגר הקוד CS. באופן דומה, כאשר משתמשים באוגר BP כמצביע לזיכרון, ברירת המחדל היא אוגר מקטע המחסנית SS, וכאשר משתמשים באוגרים BX, SI ו-DI כאוגרים המצביעים על היסט, ברירת המחדל היא אוגר מקטע הנתונים DS. בהמשך, כחלק מתיאור של הוראה, נפרט גם את מקטע ברירת המחדל המתאים. אולם, אם במהלך התכנית נצטרך לפנות לתא שכתובתו נמצאת מחוץ לאחד המקטעים, נצטרך לכתוב הוראה שתשנה את כתובת אוגר המקטע בהתאמה.

השימוש במקטעים מאפשר גם תאימות למעבדים ישנים. למשל, במעבד 8085, שקדם למעבד 8086, מרחב הזיכרון היה רק 64KB; עיקרון התאימות היה צריך להבטיח כי תכניות שהורצו על 8085 יוכלו לרוץ גם במעבד 8086. וכך במעבד 8086 בו פנייה לזיכרון נעשית בשיטת מיעון מקטעים שגודלם 64KB ניתן להריץ תכנית שנכתבה למעבד 8085. אך אולם מעבר לתאימות למעבדים ישנים, יש לשימוש במקטעים יתרונות נוספים:

- בשימוש בזיכרון ליניארי, שבו מרחב הזיכרון כולו זמין לתכנית (כפי שתיארנו בפרק הראשון), יש לדאוג שהוראות לא ידרסו נתונים (או להיפך). דריסה מתרחשת כאשר הוראה ונתון נכתבים לאותו תא זיכרון.
- בגלל ההפרדה בין הוראות לנתונים, אפשר להריץ את אותה תכנית, ולשנות בכל פעם רק את הנתונים.
- שימוש בקיטוע (סגמנטציה) מנוצל על-ידי מערכת ההפעלה כדי להריץ כמה תכניות במקביל, כאשר לכל תכנית מוקצים מקטעים נפרדים.
- כמו כן, ניתן לשתף בקלות נתונים בין תכניות שונות על-ידי פניה למקטע משותף.

4.4 כתיבת תכנית בשפת אסמבלי

4.4.1 מבנה הוראות בשפת אסמבלי

מבנה ההוראות בשפת אסמבלי דומה למבנה ההוראות בשפת מכונה (שתיארנו בפרק הראשון) והן מורכבות מאופרטור ואופרנדים. כדי לתאר את ההבדלים בין שפת מכונה לשפת אסמבלי נשתמש בדוגמה המציגה הוראות אסמבלי לביצוע הפעולות הבאות:

שים 23 במשתנה a
שים 54 במשתנה b
חבר a + b ושים את התוצאה במשתנה a

להלן תרגום אפשרי של הוראות אלה להוראות בשפת אסמבלי:

תיאור ההוראה	ההוראה בשפת אסמבלי
שים 23 במשתנה a	mov a, 23d
שים 54 במשתנה b	mov b, 54d
שים תוכן משתנה b באוגר ax	mov ax, b
חבר a + ax ושים את התוצאה באוגר ax	add a, ax

בקטע התכנית אנו משתמשים בשתי הוראות אסמבלי: להעתקת נתונים אנו משתמשים בהוראה MOV (קיצור של המילה move) ולחיבור שני נתונים אנו משתמשים בהוראה ADD. כפי שניתן לראות, האופרטור רשום בשפת אסמבלי בקוד מנמוני (אותיות אנגליות מהוות קיצור של הפעולה) ולא בקוד מספרי.

בשפת מכונה, קוד הפעולה של האופרטור הוא מספר התלוי בסוג הפעולה ובצירוף של האופרנדים. כך לדוגמה הגדרנו, במעבד שהוצג בפרק הראשון, חמש הוראות MOV שונות. בשפת אסמבלי, לעומת זאת קיימת הוראת MOV אחת בלבד (מה שבהחלט מקל על כתיבת התכנית וקריאתה). בצורה כללית אפשר לרשום את ההוראה MOV כך (קראו משמאל לימין):

mov אופרנד יעד, אופרנד מקור

משמעות ההוראה: העתק את התוכן של אופרנד המקור לאופרנד יעד; אופרנד יכול להיות: אוגר, תא בזיכרון או קבוע. למעשה ההוראה mov מגדירה בשפת אסמבלי משפחה של הוראות MOV בשפת מכונה, אבל התרגום להוראת מכונה ספציפית אינו מוטל על המתכנת אלא על תכנית שנוצרה כדי לתרגם. לתכנית זו קוראים "אסמבלר" (Assembler). היא קוראת את הוראות התכנית כמחרוזת של תווים, ומפרקת אותן לאופרטורים ואופרנדים, ובהתאם לכך מתרגמת כל אופרטור לקוד מסוים בשפת המכונה.

הנה כמה דוגמאות לתרגום במעבד 8086 :

ההוראה mov ax, a

מתורגמת להוראה בשפת מכונה שבה קוד הפעולה הוא A1h

וההוראה mov ax, 0FFh

מתורגמת להוראה בשפת מכונה שבה קוד הפעולה הוא B8h

באופן דומה, גם הוראת החיבור ADD וכל ההוראות בשפת אסמבלי שנציג בהמשך, מתארות משפחה של הוראות בשפת מכונה.

למעבד 8086 יש אוצר פקודות ומבנה עשיר יותר מאשר למעבד הפשוט שהיצגנו ; לכל הוראה בשפה מוגדר תחביר (syntax) המתאר את צירוף האופרנדים המותר. לדוגמה : ניתן לחבר תא בזיכרון עם אוגר, אולם לא ניתן לבצע פעולה בין שני אופרנדים שהם תאים בזיכרון.

בשפת המכונה כל הנתונים מוצגים בשיטה הבינארית. בשפת אסמבלי, לעומת זאת, אפשר להציג נתונים בשיטה הבינארית, העשרונית או ההקסאדצימלית. נתונים אלה יתורגמו על-ידי האסמבלר לנתונים בינאריים, כלומר למספרים המיוצגים באמצעות הספרות 0 ו-1 בלבד.

להוראות בשפת אסמבלי יש מבנה אחיד :

{Label:} Mnemonic {Operands} {;Comment}

הפרמטרים בסוגריים מסולסלים הם אופציונאליים, דהיינו : לא תמיד הם מופיעים. כפי שיוסבר בהמשך, האופרטור (Mnemonic) מופיע תמיד. המשמעות היא, שההוראה צריכה להכיל לפחות שדה אחד, אך לא יותר מארבעה שדות. בסעיף זה נתאר את השדות ותפקידיהם :

1. השדה הראשון הוא **תווית** שאפשר להוסיף לחלק משורות התכנית. התווית מציינת את הכתובת הסמלית (בזיכרון) של ההוראה אליה מוצמדת התווית. לאחר התווית ובצמוד אליה, יש להוסיף את התו [:] . שימוש נפוץ בתווית, כפי שהדגמנו בפרק הראשון, הוא בהוראות קפיצה, ומימוש הוראות תנאי ולולאות.

לדוגמה, נוסיף תווית doOper להוראה mov :

```
doOper:   mov ax, bx
```

בהמשך נרחיב על השימוש בתווית לביצוע מבני בקרה.

2. השדה השני הוא קוד פעולה של אופרטור הרשום כשם מנמוני (mnemonic) המתאר את הפעולה, כגון MOV, ADD, CMP. זהו השדה היחיד, מבין ארבעת השדות, שחייב להופיע בכל הוראה בשפת אסמבלי.

3. השדה השלישי הוא שדה האופרנדים (operands) שמכיל את הארגומנטים עליהם מתבצעת ההוראה; אופרנד יכול להיות: נתון, אוגר או משתנה בזיכרון. במעבד 8086 הוראה יכולה לכלול אופרנד אחד או שניים או לא לכלול שום אופרנד.

4. השדה הרביעי מכיל הערות שאינן מיועדות לביצוע; האסמבלר מתעלם מהן בזמן תרגום ההוראה לשפת מכונה. התו הראשון בשדה ההערה חייב להיות ; לאחר תו זה יכולה ההערה להימשך עד סוף השורה. התו ; יכול להופיע בכל מקום בשורה ואפילו בעמודה הראשונה (במקרה כזה יתייחס האסמבלר לכל השורה כאל הערה).

4.4.2 מבנה תכנית והנחיות בשפת אסמבלי

כדי להריץ את התכנית עלינו לרשום הוראות מיוחדות, בנוסף להוראות בשפת אסמבלי; ההוראות המיוחדות יתארו, בין היתר, את המקטעים ואת המשתנים בהם נשתמש בתכנית וייסעו למערכת ההפעלה בארגון התכנית לריצה. ההוראות המיוחדות נקראות **"הנחיות אסמבלר"** (Assembler directives); הן אינן מתורגמות לשפת מכונה ולכן אינן משתתפות בתהליך ההבאה-ביצוע, אך בלעדיהן לא ניתן להריץ את התכנית. בסעיף זה נתאר, באמצעות תכנית לדוגמה, את המבנה של תכנית בשפת אסמבלי הכוללת את ההנחיות המינימאליות הדרושות לביצוע תכנית המחברת בין שני משתנים. בתכנית לדוגמה ההוראות ממוספרות; הדבר נעשה כדי שנוכל להתייחס אליהן בהסברים; שימו לב, שכאשר כותבים תכנית בשפת אסמבלי, לא ממספרים את ההוראות.

```
.MODEL SMALL
.STACK 100H
.DATA
```

```
a      DW   2
b      DW   5
sum  DW   ?
.CODE
start:
1.                                              ; אתחול אוגר מקטע הנתונים
2.         mov  ax,   @DATA
3.         mov  ds,    ax
4.                                              ; חיבור המספרים
5.         mov  ax,    a                        ; ax = a
6.         add   ax,    b                       ; ax = a + b
7.         mov  sum, ax                         ; sum = ax
8.                                              ; סיום התכנית והחזרת הבקרה למערכת ההפעלה
9.         mov  ax,   4C00h
10.        int    21h
END start
```

כאמור, כל תכנית בשפת אסמבלי מכילה שני סוגים של הוראות:

- הוראות ביצוע השייכות לאוצר הפקודות של המעבד שבו אנו משתמשים. הוראות אלה מתורגמות על-ידי האסמבלר לשפת מכונה, והן מתבצעות בתהליך ההבאה-ביצוע שתואר בפרק הראשון.
- הנחיות אסמבלר המכילות הגדרות הקובעות את ארגון הזיכרון, הגדרת המשתנים ונתונים נוספים שמגדירים את הסביבה. הנחיות אלה אינן מתורגמות לשפת מכונה.

החלק הראשון של התכנית כולל כמה הנחיות אסמבלר המתחילות בנקודה ואחריהן רשומה ההנחיה עצמה. הנחיות אלה מגדירות את ארגון הזיכרון:

הנחיה מס. 1

```
.MODEL SMALL
```

ההנחיה MODEL מתייחסת להגדרת המקטעים בזיכרון, והערך SMALL מגדיר תכנית שבה יש מקטע קוד אחד ומקטע נתונים אחד.

שלוש ההנחיות הבאות מגדירות את המקטעים:

הנחיה מס. 2 קובעת את תחילת מקטע המחסנית ואת גודל המחסנית:

```
.STACK 100h
```

בתכנית זו לא נשתמש במחסנית, למרות שהנחיה זו כלולה בתכניות שנכתוב; בהמשך נסביר מהו תפקיד המחסנית, כיצד משתמשים בה וכיצד קובעים את גודלה.

הנחיה מס. 3 מגדירה את תחילת מקטע הנתונים:

```
.DATA
```

ההנחיות מס. 4 עד מס. 6 מגדירות שלושה נתונים שיאוחסנו במקטע הנתונים:

א. משתנה בשם a המוגדר על-ידי DW (קיצור של Define Words) הוא מטיפוס **מילה** ומותחל לערך 2.

```
a      DW   2
```

ב. משתנה בשם b המוגדר על-ידי DW הוא מטיפוס מילה ומותחל לערך 5.

```
b      DW   5
```

ג. משתנה בשם sum המוגדר על-ידי DW הוא מטיפוס מילה ללא ערך תחילי ומצוין על-ידי התו '?'.

```
sum   DW    ?
```

הנחיה מס. 7 מציינת את תחילתו של מקטע ההוראות ("הקוד"), ולמעשה כך נקבע גם סיום מקטע הנתונים:

```
CODE
```

שורה מס. 8 מתחילה בתווית המסתיימת בנקודתיים:

```
start:
```

התווית start מציינת את תחילת קטע הוראות התכנית. תכנית האמסבלר מתרגמת תווית start לכתובת המציינת היכן מתחיל ביצוע הוראות התכנית. למעשה כל תווית בתכנית מתורגמת על-ידי האסמבלר לכתובת בזיכרון בה נמצאת הוראה אליה מוצמדת התווית.

שורה מס. 9 מתחילה בנקודה פסיק והיא שורה המכילה הערה. כמקובל גם בשפות עיליות, האסמבלר מתעלם מהערות והן נועדו לשפר את קריאות התכנית. שימו לב: הערה מתחילה בתו ';'

; אתחול מקטע הנתונים

שתי הוראות הביצוע הראשונות רשומות בשורות 10 ו-11; תפקידן לאתחל את אוגר המקטע בכתובת הבסיס של מקטע הנתונים:

הוראה מס. 10 מעבירה לאוגר AX את מען תחילת מקטע הנתונים.
הסימול @DATA מסמן לאסמבלר את המען של מקטע הנתונים כפי שנקבע בזמן הרצת התכנית. במהלך פעולת האסמבלר המען של מקטע הנתונים נקבע לאפס ורק בזמן הטעינה לזיכרון הוא מקבל את הכתובת האמיתית של מקטע הנתונים.

הוראה מס. 11 היא הוראת השמה המעתיקה כתובת זו מאוגר AX אל אוגר מקטע הנתונים DS.

```
mov  ax,  @DATA        ; ax ← כתובת בסיס של מקטע הנתונים
mov  ds,  ax           ; ds ← ax
```

שתי ההוראות הללו כוללות הערות המופיעות אחרי התו ';' האמסבלר מתעלם מהן אולם הן תורמות להבנת התכנית ע"י אדם הקורא אותה.

שורה מס. 12 היא הערה; בשלוש השורות הבאות (מס. 13 עד 15), רשומות הוראות המשמשות לחיבור a + b ולהשמת התוצאה בזיכרון, בתא שכתובתו היא sum.

```
mov  ax,  a            ; ax ← a
add  ax,  b            ; ax ← a + b
mov  sum, ax           ; sum ← ax
```

שורה מס. 16 היא שוב הערה, ושתי ההוראות האחרונות, הרשומות בשורות מס 17 ו-18, הן הוראות שחייבים לרשום בכל תכנית, כי הן מאפשרות יציאה מסודרת מהרצת התכנית והחזרת הבקרה למערכת ההפעלה. על נושא זה נרחיב בפרק 9.

```
mov  ax,   4C00h
int    21h
```

שורה מס 19 היא ההנחיה האחרונה בתכנית, והיא מציינת לאסמבלר את סוף התכנית; המהדר לא יתייחס לכל ההוראות שיופיעו אחרי הנחיה זו.

```
 END   start
```

לסיכום נציג שלד של תכנית, שישמש אותנו לכתיבת תכניות:

```
.MODEL SMALL
.STACK 100h
.DATA
                        הגדרת משתנים
CODE
start:
;אתחול אוגר מקטע הנתונים
        mov  ax,  @DATA
        mov  ds,  ax
                        הוראות התכנית
;הוראות ליציאה מהתכנית והעברת הבקרה למערכת ההפעלה
        mov  ax,  4C00h
        int    21h
END start
```

הערות: האסמבלר אינו מבחין בין אותיות קטנות לגדולות. לצורך נוחות הקריאה נרשום באותיות גדולות את הנחיות האסמבלר ואת ההוראות בשפת אסמבלי. שמות האוגרים, התוויות ושמות המשתנים יופיעו באותיות קטנות.

בתכניות הכתובות בשפת אסמבלי ההערות הן חלק חשוב ביותר בתכנית, בעיקר משום שההוראות עצמן נרשמות בצורה סימבולית. שיטה נוספת לשיפור הקריאות היא הוספת שורות רווח בין קטעים שונים של התכנית.

שאלה 4.9

כתבו תכנית אסמבלי מלאה (כולל הנחיות אסמבלר) לביצוע שתי הפעולות האלה:

```
x = 10
y = x+1
```

4.5 הגדרת משתנים בשפת אסמבלי

לסיום נסביר כיצד מגדירים משתנים בשפת אסמבלי. **שם המשתנה** מייצג את הכתובת בזיכרון שם מאוחסן הנתון. הגדרת משתנים בשפה עילית כוללת שני מרכיבים חשובים:

1. שם המשתנה, בו משתמשים לייצוג הנתון;
2. טיפוס נתונים, המגדיר את תחום הערכים ואת הפעולות שניתן לבצע על הנתונים. הוא גם מגדיר את צורת ייצוג הנתון בזיכרון.

לדוגמה, כדי להצהיר על הטיפוס "מספר שלם" בשפת פסקל, נרשום:

```
var i: integer;
```

הצהרה דומה בשפת C תירשם כך:

```
int i;
```

הצהרה זו מגדירה משתנה i מטיפוס "שלם" שבו ניתן לאחסן מספרים שלמים בלבד (שברים אסורים). הפעולות האריתמטיות שניתן לבצע על משתנה מטיפוס שלם, הן: חיבור, חיסור, כפל והפעולות MOD ו-DIV.

טיפוס נתונים בשפה עילית מגדיר גם את גודל הזיכרון המוקצה למשתנה. כך לדוגמה, בשפת פסקל גודל הזיכרון המוקצה למשתנה מטיפוס שלם, הוא יחידת אחסון בגודל מילה (שני בתים).

בשפת אסמבלי, טיפוס של משתנה מוגדר על-ידי גודל הזיכרון, כלומר מספר הבתים המוקצים למשתנה.

הגדרה של משתנה כוללת שלושה שדות:

```
var-name    data-type    init-value
```

- השדה הראשון הוא שם המשתנה – אורכו עד 31 תווים, הוא מתחיל באות ויכול להכיל אותיות, מספרים ותווים מיוחדים '$', '-', '@', '?', '.'.
- השדה השני מציין את טיפוס הנתונים – הוא מתחיל תמיד באות D (המציינת קיצור של המילה Define), ואחריו רשומה אות נוספת, המציינת את הטיפוס, מתוך חמישה טיפוסי נתונים אפשריים:

תיאור	מספר הבתים	הנחייה לציון טיפוס משתנה
טיפוס בית – Byte	1	DB
טיפוס מילה – Word	2	DW
טיפוס מילה כפולה – Double Word	4	DD
טיפוס מילה מרובעת – Quad Word	8	DQ
טיפוס Tera (ten) Byte	10	DT

- השדה השלישי מציין ערך תחילי, המושם במקום האחסון עם ההקצאה של הזיכרון.

ניתן להגדיר משתנה ללא ערך תחילי על-ידי הסימול '?'.

לדוגמה נרשום שלוש הגדרות של משתנים:

א. משתנה בשם stored, מטיפוס BYTE, שערכו התחילי הוא קוד ASCII של התו y

```
sorted      DB 'y' ;
```

ב. משתנה בשם response, מטיפוס BYTE, שהוא ללא ערך תחילי

```
response    DB ?
```

ג. משתנה בשם value, מטיפוס WORD, שערכו התחילי הוא המספר 25159

```
value       DW 25159
```

הנתונים שאפשר לאחסן בתא הם:

- מספר המייצג מספר שלם ללא סימן או מספר שלם עם סימן
- תו המיוצג בקוד ASCII

אפשר לרשום מספרים בשיטה העשרונית, הבינארית או ההקסאדצימלית. כדי להבדיל בין הייצוגים השונים, מוסיפים סימול מתאים בסוף המספר:

- הסימול d בסיום מספר מציין מספר עשרוני; לדוגמה: 23d .
- הסימול h בסיום מספר מציין מספר בשיטה הקסאדצימלית; לדוגמה: 23h.
- הסימול b בסיום מספר מציין מספר בינארי; לדוגמה: 10001011b.

שימו לב: במספרים המיוצגים בשיטה ההקסדיצמלית ומתחילים באחת מן האותיות A – F, יש להוסיף את הספרה 0 בתחילת המספר. כל עוד לא ציינו הנחיה מפורשת לשיטת הספירה, ברירת המחדל בתכניות היא שמספר ללא סימן מייצג מספר עשרוני.

אפשר לרשום מספר עשרוני שלם שלילי, והוא ייוצג בשיטת המשלים לשתיים. לדוגמה, כדי להגדיר משתנה מטיפוס בית, בשם a, המאותחל למספר 10–, נרשום:

```
a    DB        –10d
```

הערך שיאוחסן בתא הזיכרון שיוקצה למשתנה, יהיה F6h שהוא המשלים לשתיים של הערך 10–.

המעבד 8086 אינו כולל הוראות בשפת אסמבלי לעיבוד מספרים שאינם שלמים, אולם במעבדים מתקדמים ממשפחה זו נכללות הוראות אסמבלי נוספות, המאפשרות לבצע חישובים עם מספרים שאינם שלמים. הדיון במספרים שאינם שלמים ועיבודם חורג מגבולות ספר זה.

שאלה למחשבה: כיצד ניתן לבצע פעולות עם מספרים שאינם שלמים אם שפת המכונה לא כוללת הוראות לעיבוד מספרים שאינם שלמים?

כדי לאתחל משתנה לתו, נרשום את התו בין גרשים יחידים או כפולים. לדוגמה, נגדיר משתנה בשם char המאותחל לתו a:

```
char DB "a"
```

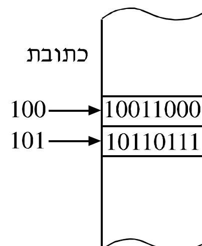
המשתנה char הוא מטיפוס בית, ובו יאוחסן קוד ASCII של התו a.

הגדרות של משתנים נרשמות בתכנית אסמבלי, לאחר ההנחיה המגדירה את תחילת מקטע הנתונים:

.DATA

לפני שאנו מריצים תכנית, אנו מתרגמים אותה לשפת מכונה (באמצעות תכנית אסמבלר), ולאחר מכן אנו מפעילים תהליך שנקרא "קישור" (link). בתהליך הקישור נקבעים מיקומי המקטעים השונים בזיכרון וביניהם גם מקטע הנתונים, הכולל הקצאה של בתים בהתאם למשתנים שהגדרנו. למשתנה הראשון, שהוגדר במקטע הנתונים עבור הנתון הראשון, מוקצים התאים הראשונים במקטע, והוא מתחיל בהיסט 0. המשתנה השני עוקב לו והוא מתחיל בתא העוקב לתא האחרון שהוקצה למשתנה שלפניו, וכך הלאה, לפי סדר ההגדרה של המשתנים במקטע הנתונים.

מאחר שגודל תא בזיכרון הוא בית אחד, נשאלת השאלה: כיצד מאוחסנים בזיכרון משתנים מטיפוס מילה או מילה כפולה? במעבדים של אינטל, מוקצים כמה בתים עוקבים למשתנים מטיפוס הגדול מטיפוס בית. כך מוקצים שני בתים עוקבים למשתנה מטיפוס מילה, ולטיפוס מילה כפולה מוקצים ארבעה בתים עוקבים. אחסון הנתון בשטח שהוקצה למשתנה מתבצע בשיטה הנקראת little-endian. לפי שיטה זו, הבית הפחות משמעותי מאוחסן בכתובת הנמוכה, והבית המשמעותי של המילה מאוחסן בכתובת העוקבת (ומכאן השם little-endian, דהיינו: מי שמופיע ראשון הוא "הקצה הקטן"; השם נבחר בהומור, שכן הוא מזכיר "אינדיאני קטן" – little Indian). לדוגמה, המילה 1011011110011000_2 תאוחסן בשני תאי זיכרון: בתא שכתובתו היחסית היא 101h יאוחסן הבית המשמעותי, כלומר 10110111_2, ובכתובת היחסית 100h יאוחסן הבית הפחות משמעותי, כלומר 10011000_2.



איור 4.6
אחסון מילה בזיכרון בשיטה little-endian

בציון כתובת של משתנה, אנו רושמים רק את כתובת הבית התחתון; בדוגמה הקודמת הכתובת היא 100h. כאשר אנו קוראים משתנה זה, המעבד מתרגם את הכתובת היחסית שלו לכתובת פיזית (תוך שימוש באוגר הבסיס ובאוגר המצביע), ובהתאם לטיפוס המשתנה הוא קורא את כל הבתים שהוקצו לו.

לדוגמה, נציג את צורת האחסון של נתונים המוגדרים במקטע הנתונים הבא:

```
.DATA
var1   DB    10h
var2   DB    00h
var3   DW    1234h
sum    DB    28h
var4   DW    51ABh
var5   DB    ?
char1  DB    'e'
char2  DB    'H'
var6   DD    12345678h
```

אם נניח כי מקטע הנתונים מתחיל בכתובת DS = 01000h, נוכל לצייר מפת זיכרון (איור 4.7) שתתאר את המיקום של כל אחד מהמשתנים שהגדרנו. נציין מספר הערות בקשר למפת הזיכרון המוצגת באיור 4.7:

- מקומות בזיכרון המוקצים למשתנים נמצאים ברצף תאים החל מתחילת מקטע הנתונים.
- ההיסט של המשתנה הראשון מתחילת מקטע הנתונים הוא 0.
- המשתנה השני מתחיל בתא העוקב לתא שבו הסתיים המשתנה הקודם. ההיסט של המשתנה השני הוא הסכום המתקבל ע"י חיבור ההיסט של המשתנה שקדם לו ומספר התאים שתפס המשתנה שקדם לו.
- משתנה בגודל מילה תופס שני תאים – בתא שכתובתו נמוכה נמצאות שתי הספרות הפחות משמעותיות ובתא שכתובתו עוקבת נמצאות שתי הספרות משמעותיות.
- משתנה בגודל מילה כפולה מאוחסן על פי אותו רעיון: הספרות הפחות משמעותיות נמצאות בכתובות הנמוכות.

- כאשר אנו מציינים כתובת של משתנה שגודלו גדול מבית אחד, כלומר, מילה, מילה כפולה וכדומה, אנו מציינים את כתובת הבית הנמוך ביותר. לדוגמה: כתובת משתנה var3 היא 1002h (למרות שהוא מאוחסן בתאים 1002h ו-1003h).
- הכתובות 1009h ו-1008h הוקצו למשתנים char1 ו-char2. שימו לב, בזיכרון אוחסן קוד ASCII 65h המתאים לתו H וקוד ASCII 48h של התו e.

	כתובת (הקסדימאלית)	תוכן התא	שם המשתנה
	0100F		
	100E		
	100D	12h	
	100C	34h	
	100B	56h	
	100A	78h	var6
	1009	48h	char2
	1008	65h	char1
	1007		var5
	1006	51h	
	1005	0ABh	var4
	1004	28h	sum
	1003	12h	
	1002	34h	var3
	1001	00h	var2
DS →	1000	10h	var1

איור 4.7
הקצאת משתנים במקטע הנתונים DS

כדי לפנות לנתון אנו יכולים לציין את שם המשתנה; האסמבלר דואג להחליף את השם הסימבולי בכתובת המתאימה. לדוגמה:

- הוראה המעתיקה את תוכן המשתנה var1 שהוא בגודל בית לאוגר al שגם הוא בגודל בית:

```
mov al,   var1
```

הוראה זו זהה להוראה הבאה:

```
mov al,   1000h
```

אך התוכנית בצורה כזו הרבה פחות קריאה.

- הוראה המעתיקה תוכן המשתנה var3 לאוגר BX. שימו לב, המשתנה var3 הוא בגודל מילה וכך גם אוגר BX:

```
mov bx,   var3
```

הערך שנטען לאוגר הוא: BX =1234h

עד כה ראינו כיצד מייצגים מספרים שלמים ותווים, דבר המסביר כיצד מגדירים טיפוסים כמו integer או char. כדי לייצג משתנה בוליאני מספיקה סיבית אחד לציון 0 או 1, בהתאמה לאמת ושקר, אולם יחידת המידע הקטנה ביותר, המועברת בין הזיכרון למעבד, היא בית אחד. לכן, ברוב השפות העיליות מוגדר משתנה בוליאני כטיפוס BYTE, לדוגמה:

```
boolean   DB 0
```

אפשר לקבוע כי ערך הבית יוגדר כאמת (true) כאשר ערכו הוא 0 וכל ערך אחר (גדול מאפס) יוגדר כשקר (false). כך זה, למשל, בשפת C; בשפת פסקל ההגדרה שונה. במעבד 8086, משתנה מטיפוס ממשי מטופל באמצעות תוכנה, כלומר, ניתן להגדיר מספר בן שני חלקים: שלם ושבר ולכתוב תכנית מתאימה שתטפל בערכים אלה.

שאלה 4.10

בתכנית הוגדרו המשתנים האלה:

```
.DATA
   a   DB    10h
   b   DD    1234h
   c   DW    100h
   d   DB    'H'
```

ציירו את מפת הזיכרון של מקטע הנתונים וחשבו את ההיסט של כל משתנה.

4.6 הצהרה על קבועים – הנחיית אסמבלר EQU (EQUal או EQUate).

כדי להגדיר קבוע נרשום את ההנחיה הזו:

שם קבוע EQU ערך

לדוגמה:

```
MAX EQU 5        ; הצהרה על קבוע בשם MAX שערכו 5
TEMP EQU 2*3     ; הצהרה על קבוע בשם TEMP שערכו 6
```

בדומה למשתנה, קבוע הוא שם לוגי של נתון כלשהו, אך לקבוע לא מוקצה מקום בזיכרון. בזמן תרגום התכנית לשפת מכונה, הקבוע מוחלף בנתון מספרי, אותו הוא מייצג (בדומה למשתנה המוחלף בכתובת שהוקצתה לנתון שהמשתנה מייצג). נהוג לרשום את הגדרות הקבועים לפני קטע הנתונים, לדוגמה:

```
MODEL SMALL
.STACK 100h
   MAX EQU 5          ; הגדרה של קבוע
DATA
   a DB 1,2,3,4, 5    ; אתחול מערך שבו 5 איברים בגודל בית
   sum db 0           ; סכום איברי המערך
```

שימוש בקבוע משפר גם את קריאות התכנית; כך נוכל לתת לנתון מסוים שם לוגי שיתאר את משמעותו בתכנית. לאחר שהצהרנו על קבוע MAX, נוכל להשתמש בו כאופרנד בהוראות שונות; לדוגמה:

```
mov al, MAX
add ah, MAX
```

נספח – תיאור תהליך הרצת התכנית

לאחר התרגום של תכנית האסמבלי והקישור, אם אין שגיאות תחביר, מופק קובץ הרצה (במקרה של מערכת Windows או DOS, למשל, זהו קובץ שהסיומת שלו היא exe). כאשר מריצים קובץ זה, מערכת ההפעלה טוענת את התכנית לזיכרון ומעבירה את הבקרה לתחילת התכנית לשם הרצתה.

מערכת ההפעלה מקצה את המקטעים הדרושים לתכנית וטוענת לכל מקטע את קטע התכנית המתאים: למקטע הקוד את ההוראות, למקטע הנתונים את המשתנים וכך הלאה.

תחילה מקצה מערכת ההפעלה קבוצת תאי זיכרון נוספת הנקראת **כותרת**. הכותרת מכילה מאפיינים שונים של התכנית. מערכת ההפעלה משתמשת בקבוצת התאים הזו (הנקראת PSP – קיצור של Program Segment Prefix) כדי לנהל את המשימות המורצות במחשב. אחרי הכותרת מוקצים המקטעים לפי הסדר הבא: מקטע הקוד, מקטע הנתונים ומקטע המחסנית. לאחר הקצאת המקטעים, מאותחל כל אחד מאוגרי הסגמנט לכתובת של תחילת המקטע המתאים; אוגרי סגמנט הנתונים DS ו-ES מאותחלים לכתובת של תחילת ה-PSP. כדי לאתחל את אוגר מקטע הנתונים DS לכתובת הבסיס של מקטע הנתונים, אנו רושמים את שתי ההוראות האלה:

```
mov   ax,   @DATA
mov   ds,    ax
```

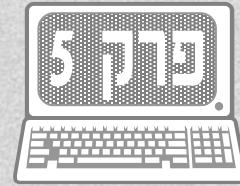
הוראות אלה מעתיקות את כתובת הבסיס של מקטע הנתונים ומאתחלות את האוגר DS. בגלל המגבלות של מבנה המחשב, ובעקבות זאת — מגבלות של שפת המכונה (בפרק השישי נרחיב את הדיון בהן), איננו יכולים לשים את כתובת הבסיס של מקטע הנתונים ישירות באוגר DS, ולכן אנו משתמשים באוגר AX. הסימול @ הוא הנחיה של האסמבלר והוא מציין כתובת.

כאשר אנו טוענים תכנית לזיכרון, שמות המשתנים והתוויות מוחלפים בכתובות אפקטיביות מתאימות. לשם כך מנהל האסמבלר טבלה הנקראת "טבלת הסמלים" (symbol table). בטבלה זו מאוחסנים כל הסמלים (שמות משתנים, תוויות, וכדומה)

המופיעים בתכנית, יחד עם התכונות הרלוונטיות שלהם. לסיום, נציג קטע מטבלת הסמלים של התכנית לדוגמה שהוצגה בסעיף 4.3.

טבלה 4.3
קטע מטבלת סמלים של תכנית

סמל	מקטע	כתובת היסט	טיפוס
a	DATA	0	WORD
b	DATA	2	WORD
sum	DATA	4	WORD
start	CODE	0	



תכנות בסיסי בשפת אסמבלי

5.1 מבוא

לאחר שהכרנו את המודל התכנותי של המעבד 8086 נציג בפרק זה חלק מאוסף הוראות האסמבלי של מעבד זה. את אוסף הוראות האסמבלי ניתן לחלק לקבוצות על-פי סוג הפעולה שהן מבצעות. בפרק זה נתמקד בהוראות מקבוצות ההוראות הבאות: הוראות להעברת נתונים, הוראות אריתמטיות, הוראות לוגיות והוראות להעברת בקרה. לכל הוראה נציג את התחביר שלה ונתייחס לסוגי האופרנדים בהם ניתן להשתמש בה ולהשפעה של ביצוע ההוראה (במידה שיש) על אוגר הדגלים. בהוראות אלה נשתמש כדי להציג כיצד כותבים תכניות בשפת אסמבלי וכיצד מממשים תבניות תכנות שונות המוכרות לנו משפות עיליות, למשל: הוראות תנאי ולולאות.

5.2 הוראות להעברת נתונים

קבוצה זו כוללת הוראות להעברת נתונים בין שני אופרנדים אותם נכנה בשמות הבאים:

- אופרנד מקור – המכיל את הנתון שיש להעביר.
- אופרנד יעד – המציין את המקום בו יאוחסן הנתון שהועבר.

בסעיף זה נציג רק שתי הוראות מקבוצה זו: הוראת MOV שהכרנו ואת ההוראה XCHG המחליפה בין ערכי שני אופרנדים. הוראות נוספות השייכות לקבוצה נציג בהמשך.

5.2.1 ההוראה MOV (MOVe)

ההוראה MOV מעתיקה את תוכן המקור ליעד והיא נרשמת במבנה הבא:

MOV אופרנד יעד, אופרנד מקור

ההוראה מבצעת את פעולת ההשמה הכתובה להלן בשפה עילית:

אופרנד יעד ← אופרנד מקור

ההעתקה היא חוקית רק בין שני אופרנדים (יעד ומקור) שהם מאותו טיפוס (בית, מילה וכדומה), ולכן ההוראה הבאה: mov ax, al היא שגויה. באופן דומה, העתקה מתא זיכרון מטיפוס בית לאוגר מטיפוס מילה, אסורה.

צירופי האופרנדים המותרים לשימוש בהוראה זו ודוגמאות מוצגים בטבלה 5.1.

טבלה 5.1

צירופי אופרנדים אפשריים בהוראת MOV

אופרנד המקור	אופרנד היעד	דוגמה
אוגר	אוגר	mov ax, bx
נתון מיידי	אוגר	mov al, 10h
תא בזיכרון	אוגר	mov ax, var1
נתון מיידי	תא בזיכרון	mov var2, 10h
אוגר	תא בזיכרון	mov var2, al

שימו לב: כאשר אופרנד המקור הוא תא בזיכרון ניתן להעתיק אליו רק נתון מיידי או תוכן אוגר. כדי לבצע השמה של נתון מתא זיכרון אחד לתא זיכרון שני

var2 ←var1

נרשום את שתי ההוראות ההבאות:

```
mov  ax, var1        ; ax ← var1
mov  var2, ax        ; var2 ← ax
```

ככלל, אין הוראה בשפת אסמבלי בה קיים הצירוף בו שני האופרנדים, אופרנד המקור ואופרנד היעד, הם תאים בזיכרון.

5.2.2 ההוראה XCHG (eXCHanGe)

הוראה נוספת השייכת לקבוצה זו היא ההוראה להחלפת תכנם של אופרנד המקור ואופרנד היעד:

XCHG אופרנד יעד, אופרנד מקור

הוראה זו מבצעת את החלפה בין תוכן אופרנד המקור לבין תוכן אופרנד היעד.

לדוגמה:

```
mov  al, 10h          ; al ← 10h
mov  bl, 0F0h         ; bl ← 0F0h
xchg  al, bl          ; al ← 0F0h, bl = 10h
```

הצירופים של אופרנד המקור ואופרנד היעד המותרים בהוראה זו מוצגים בטבלה 5.2.

טבלה 5.2
צירופי אופרנדים אפשריים בהוראת **XCHG**

אופרנד היעד	אופרנד המקור	דוגמה
אוגר	אוגר	xchg ax, bx
תא בזיכרון	אוגר	xchg al, var
אוגר	תא בזיכרון	xchg var, AL

שאלה 5.1

ציינו אילו מן ההוראות שלהלן תקינות ואילו אינן תקינות.

א. mov bh, al

ב. mov dh, cx

ג. mov bh, bh

ד. mov cl, 4F2h

ה. mov cx, 002dh

ו. xchg var1, var2

שאלה 5.2

מטרת התכנית הבאה היא לבצע החלפה סיבובית בין שלושה משתנים המוגדרים במקטע הנתונים בצורה הבאה:

```
.DATA
  a  db  12h
  b  db  34h
  c  db  78h
```

כאשר ערכו של כל משתנה מוחלף עם ערך המשתנה הסמוך לו: כלומר a מוחלף עם b, b מוחלף עם c, ו- c מוחלף עם a. בסיום התכנית ערכי המשתנים צריכים להיות:

a = 78h

b = 12h

c = 34h

האם ההוראות הבאות מבצעות את הנדרש? אם לא, תקנו את התכנית כך שתשיג את מטרתה.

```
mov   al, a
xchg  al, b
xchg  al, c
```

שאלה 5.3

רשמו הוראות להחלפה בין אוגר AX ואוגר BX ללא שימוש בהוראת XCHG.

5.3 הוראות אריתמטיות – חיבור וחיסור

בסעיף זה נתאר את מבנה ההוראות לחיבור וחיסור של מספרים שלמים בלתי מכוונים ועם מספרים שלמים מכוונים, נבחן את מגבלות הייצוג במחשב ונתאר את ההשפעה של ביצוע ההוראה על אוגר הדגלים. לקבוצת ההוראות האריתמטיות משתייכות גם ההוראות לכפל וחילוק, אותן נציג בסעיף 5.5. במעבד 8086 קיימות מספר הוראות לביצוע פעולות חיבור וחיסור ואנו נתחיל בהצגת שתי הוראות לחיבור וחיסור שני אופרנדים.

5.3.1 ההוראות ADD ו-SUB

א. ההוראה ADD

ההוראה ADD מחברת שני נתונים והיא נרשמת במבנה הבא:

ADD אופרנד יעד , אופרנד מקור

ומשמעותה

אופרנד יעד ← אופרנד יעד + אופרנד מקור

לדוגמה,

```
add  ax, 1234        ; ax ← ax + 1234
add  ax, bx          ; ax ← ax + bx
```

ב. ההוראה SUB (SUBtract)

ההוראה SUB מחסרת שני נתונים והיא נרשמת במבנה הבא

SUB אופרנד יעד, אופרנד מקור

ומשמעותה

אופרנד יעד ← אופרנד יעד – אופרנד מקור

לדוגמה,

```
sub  bx, 1234        ; bx ← bx – 1234
sub  cx,  dx         ; cx ← cx – dx
```

צירופי סוגי האופרנדים האפשריים בהוראות ADD ו-SUB מוצגים בטבלה 5.3.

טבלה 5.3
צירופי אופרנדים אפשריים בהוראות **ADD** ו-**SUB**

אופרנד המקור	אופרנד היעד	דוגמאות	
אוגר	אוגר	sub al, ah	add al, ah
תא בזיכרון	אוגר	sub ax, var	add ax, var
קבוע	אוגר	sub ax, 10h	add ax, 10h
אוגר	תא בזיכרון	sub var, ax	add var, ax
קבוע	תא בזיכרון	sub var, 10h	add var, 10h

נתאר את השימוש בהוראות חיבור וחיסור לכתיבת תכנית.

דוגמה 5.1

נכתוב תכנית בשפת אסמבלי שתבצע את הפעולות הבאות:

$a \leftarrow a + b - c$
$b \leftarrow c - b + 20$
$c \leftarrow 2c$

ולסיום נבדוק את התכנית עבור הערכים הבאים: $a = 38,\ \ b = 23,\ \ c = 36$.

תכנון הפתרון:

תחילה נגדיר היכן יאוחסנו הנתונים שמוגדרים בבעיה. בשפת אסמבלי אנו יכולים לאחסן נתונים בזיכרון (במקטע הנתונים) או באוגרים. לבחירה זו יש השפעה על זמן ביצוע התכנית והיא תלויה בגורמים רבים שלחלקם נתייחס בפרק 6. בשלב זה מטרתנו להדגים את השימוש במשתנים ובהתאם לכך נבחר לאחסן את הנתונים בזיכרון. כדי להגדיר משתנה יש לציין את שם המשתנה ואת טיפוס הנתונים שאנו בוחרים והוא צריך להתאים לבעיה הנתונה. בבעיה זו המספרים אותם נעבד הם קטנים ולכן נגדיר כל משתנה מטיפוס בית. כעת ניתן לרשום את הנחיות האסבלר להגדרת המשתנים ואתחול תאי הזיכרון לערכים שבחרנו כדי לבדוק את התכנית.

```
.DATA
      a    DB   38                    ; a =26h
      b    DB   23                    ; b = 17h
      c    DB   36                    ; c = 24h
```

בהערה רשמנו את ערכו של הנתון גם בשיטה ההקסדצימאלית. כך נוכל להמירו במהירות למספר בינארי כדי לעקוב אחר ביצוע ההוראות.

כדי לכתוב את ההוראות הדרושות בשפת אסמבלי, נטפל בכל פעולה בנפרד:

א. הפעולה הראשונה: $a \leftarrow a + b - c$

הפעולה הראשונה מורכבת משתי פעולות אריתמטיות: "+" ו- "-" וכדי לתרגם אותה לשפת אסמבלי יש לפרק אותה לשתי פעולות, שכל אחת מהן מבצעת פעולת אריתמטית אחת.

כלומר עלינו לחשב:

$a \leftarrow a + b$
$a \leftarrow a - c$

מאחר שאין הוראת חיבור וחיסור שבה שני האופרנדים הם תאי זיכרון, עלינו להשתמש באוגר עזר בו נאחסן את אחד הנתונים ואת הפעולה הדרושה נבצע בין תא בזיכרון ואוגר העזר. לביצוע הוראות אריתמטיות אנו יכולים לבחור בכל אוגר מקבוצת האוגרים למטרות כלליות, אך נעדיף לבחור את האוגר AX הנקרא גם "צובר". נזכור כי טיפוס האוגר צריך להתאים לטיפוס המשתנה ולכן לא נשתמש בכל 16 הסיביות של הצובר אלא נבחר באוגר AL מטיפוס בית. כעת אנו יכולים לרשום אלגוריתם מתאים לביצוע הפעולה $a \leftarrow a+b-c$:

$al \leftarrow a$
$al \leftarrow al + b$
$al \leftarrow al - c$
$a \leftarrow al$

ובהתאם נרשום את הוראות האסמבלי הדרושות

```
mov  al, a      ; al ← 26h
add  al, b      ; al ← al + b = 26h + 17h = 3Dh
sub  al, c      ; al ← al – c = 3Dh – 24h = 19h
mov  a, al      ; a  ← 19h
```

ב. הפעולה השנייה: $b \leftarrow c - b + 20$

גם פעולה זו עלינו לפרק לשתי פעולות שכל אחת מבצעת פעולה אריתמטית אחת:

$b \leftarrow c - b$
$b \leftarrow b + 20$

כדי לבצע חיסור בין שני משתני זיכרון, נבחר שוב באוגר AL כאוגר עזר בו נאחסן תחילה את c ואחר נחסר ממנו את b. אנו יכולים להשתמש באוגר AL שוב מבלי לחשוש שנאבד את תוצאת החישוב של הפעולה הראשונה משום שבסיומה העתקנו אותה למשתנה a.

האלגוריתם המתאים לביצוע הפעולה $b \leftarrow c - b + 20$ הוא:

$al \leftarrow c$
$al \leftarrow al - b$
$al \leftarrow al + 20$
$b \leftarrow al$

ובהתאם נרשום את הוראות האסמבלי הדרושות

```
mov  al, c        ; al ← 24h
sub  al, b        ; al ← al – b = 24h – 17h = 0Dh
add  al, 20       ; al ← al + 14h = 0Dh + 14h = 21h
mov  b, al        ; b ← 21h
```

ג. הפעולה השלישית: $c \leftarrow 2c$

בשפת אסמבלי קיימות מספר אפשרויות לביצוע כפל, אבל בדוגמה זו נשתמש בהוראת חיבור ונבצע את הפעולה:

$$c \leftarrow c + c$$

גם כאן נשתמש באוגר AL כאוגר עזר לביצוע האלגוריתם הבא:

$al \leftarrow c$
$c \leftarrow c + al$

ובהתאם נרשום את הוראות האסמבלי המתאימות:

```
mov  al, c        ; al ← 24h
add  c, al        ; c ← c + al = 24h + 24h = 48h
```

לסיכום נכתוב את תכנית האסמבלי המלאה:

```
.MODEL SMALL
.STACK  100h
.DATA
        a  DB   38                              ; a = 26h
        b  DB   23                              ; b = 17h
        c  DB   36                              ; c = 24h
.CODE
start:
                                                ; אתחול אוגר סגמנט הנתונים
        mov  ax, @DATA
        mov  ds, ax
                                                ; a ←a + b – c
        mov  al, a                              ; al ← 26h
        add   al, b                             ; al ← al + b = 26h + 17h = 3Dh
        sub   al, c                             ; al ← al – c = 3Dh – 24h = 19h
        mov  a, al                              ; a ← 19h
                                                ; a ← c – b + 20
        mov  al, c                              ; al ← 24h
        sub   al, b                             ; al ← al  – b = 24h – 17h = 0D
        add   al, 20                            ; al ← al + 14h = 0Dh + 14h = 21h
        mov  b, al                              ; b ← 21h
                                                ; c ← 2c
        mov  al, c                              ; al ← 24h
        add   c, al                             ; c ←  c + al = 24h + 24h = 48h
                                                ;הוראות ליציאה מהתכנית
        mov ax, 4C00h
        int 21 h

 END start
```

שאלה 5.4

רשמו הוראות לביצוע הפעולה הבאה: $a \leftarrow 2b - (c+12)$. השתמשו בנתונים שהוגדרו בבעיה לדוגמה.

5.3.2 השפעת טיפוס האופרנד על ביצוע הוראות ADD ו-SUB

בתכנית שהצגנו בדוגמה 5.1 טפלנו במספרים קטנים והתוצאות של החישוב היו בתחום הערכים שבית יכול להכיל (בין 0 ל-255). אולם מה יקרה אם נשנה את הנתונים לערכים הבאים:

a = 172d

b = 201d

c = 78d

כשנריץ שוב את התכנית, נקבל כי הערך של a לאחר ביצוע הפעולה הראשונה:

$a \leftarrow a + b - c = 39$

אולם מחישוב מתמטי של הביטוי מתקבל:

$172 + 201 - 78 = 295$

חשבו: מה השתבש? מאין נובעת הטעות בחישוב?

הטעות בחישוב נובעת מכך, שלא התחשבנו בתחום הערכים שניתן לאחסן בבית אחד שהוא בין 0 ל-255 עשרוני. כדי להבין כיצד פועל המחשב במקרה זה נעקוב אחר ביצוע ההוראות הבאות:

```
mov al, a        ; al ← 0Ach
add  al, b       ; al ← al + b = ?
```

ונתאר את פעולת החיבור באיור 5.1:

חישוב בשיטה עשרונית	חישוב בשיטה הבינארית
172	10101100
+ 201	+ 11001001
373	1 01110101

איור 5.1
חיבור מספרים מטיפוס בית

מאיור 5.1 אנו רואים כי תוצאת החיבור היא בת 9 סיביות. אולם כפי שציינו בפרק השני, בבית אחד יש רק 8 סיביות, ולכן הסיבית התשיעית "נפלה" והערך שאוחסן באוגר AL בסופו של דבר הוא:

AL ← 011101012 = 75h = 117

בהתאם לכך, התוצאה לאחר בצוע ההוראה השלישית היא:

```
sub  al, c                    ; AL ← 75h – 4Eh = 27h = 39
```

וכך קיבלנו את הטעות בתוצאה.

בביצוע חישובים על מספרים שלמים בלתי מכוונים יתכן ונקבל תוצאות שלא ניתן לאחסן באופרנד היעד. זכרו כי אופרנד היעד, שיכול להיות אוגר או תא בזיכרון, הוא מוגבל בתחום המספרים שהוא יכול להכיל. אם תוצאת החישוב גדולה מהמספר הגדול ביותר שניתן לאחסן ביחידת אחסון, מתקבלת **גלישה** (overflow). המחשב מטפל בגלישה על-ידי ביצוע חישוב במודולו 2^n, כאשר n הן מספר הסיביות, ורק השארית שמתקבלת מהחישוב במודולו נשמרת ביחידת האחסון. כלומר, אם תוצאת החישוב אותה נכנה x, גדולה מדי, המחשב מבצע את החישוב הבא:

x = x mod 2n

לדוגמה, בחישוב שתיארנו באיור 5.1, יחידת האחסון היא מטיפוס בית, בה יש 8 סיביות ולכן יתבצע החישוב במודולו 2^8 כלומר:

x = (201+172) mod 28 = 373 mod 256 = 117

כדי לפתור את בעיית הגלישה שנוצרה בדוגמה זו, אנו יכולים להגדיר משתנה מטיפוס מילה או מילה כפולה וכך להגדיל את תחום המספרים בהם ניתן לטפל. שיטה אחרת אותה נציג בסעיף 5.4 היא לבדוק אם אכן התרחשה גלישה ובהתאם להפיק הודעה למשתמש או לבצע פעולה שנקבעה מראש במקרה זה.

שאלה 5.5

מה יהיה המספר שיאוחסן באוגר AX לאחר ביצוע ההוראות הבאות:

```
mov  ax,  0F023h
add   ax,  067FAh
```

הסבירו את תשובתכם על-ידי חישוב במודולו מתאים.

שאלה 5.6

א. שנו את הגדרת המשתנים שבתכנית שהוצגה בדוגמה 5.1 והגדירו אותם כמשתנים מטיפוס מילה עם הערכים התחיליים הבאים:

a = 172d

b = 201d

c = 78d

שנו את ההוראות המתאימות, הריצו את התכנית, ובדקו אם תוצאת החישוב תקינה.

ב. מבלי לשנות את התכנית שבסעיף א', אתחלו את המשתנים לנתונים חדשים כך שחישובם יגרום לגלישה. רשמו מה הנתונים שבחרתם ורשמו את החישובים שהובילו לתוצאה שהתקבלה.

5.3.3 השפעת ביצוע הוראות ADD ו-SUB על אוגר הדגלים

כאשר המעבד מבצע הוראות חיבור וחיסור הוא לא רק מבצע את החישוב עצמו אלא גם מעדכן את דגלי המצב שבאוגר הדגלים בהתאם לתוצאה שהתקבלה. באמצעות אוגר הדגלים אנו יכולים למשל להבחין כי תוצאת חישוב אריתמטית "גלשה" מהתחום של טיפוס אופרנד היעד, או שתוצאת החישוב היא אפס.

נזכור כי אוגר הדגלים הוא אוגר בן 16 סיביות, והמעבד 8086 מגדיר 9 סיביות מתוכן כשכל סיבית משמשת בתפקיד של דגל המציג מצב מסוים שהתקבל כתוצאה מביצוע הוראה. תוכן אוגר הדגלים יכול להשתנות בשני מקרים:

- הוא משתנה על-ידי המעבד, כפי שנתאר בסעיף זה, בהתאם לתוצאה שהתקבלה מביצוע הוראה.
- כאשר המתכנת משתמש בהוראות מיוחדות, אותן נתאר בהמשך, ותפקידן לקבוע את הדגלים לערך מסוים.

לא כל ביצוע הוראה בשפת אסמבלי משפיע על הדגלים, ולכל הוראה יכולה להיות השפעה שונה. כך לדוגמה, ביצוע הוראות חיבור וחיסור משפיע על מצבם של חלק מהדגלים, אך ביצוע הוראות העברה כמו MOV ו-XCHG אינו משנה את מצב הדגלים. יצרן של מעבד מגדיר לכל הוראה והוראה את הדגלים שיושפעו מביצועה ואת התנאים שבהם מתרחש השינוי. בסעיף זה נתמקד בדגלים המושפעים מהוראות חיבור וחיסור ADD ו-SUB, ובהמשך נוסיף לכל הוראה שנתאר את הדגלים המושפעים מביצועה.

כפי שציינו, מביצוע הוראות ADD ו-SUB מושפעת קבוצת הדגלים שנקראת "דגלי המצב". תחילה נציג בקצרה את הדגלים והתנאים בהם מתרחש שינוי:

א. **דגל האפס ZF** נקבע לערך 1 אם תוצאת החישוב היא 0.
ב. **דגל הסימן SF** נקבע לערך 1 אם בתוצאת החישוב הסיבית המשמעותית ביותר היא 1.
ג. **דגל הנשא – CF** נקבע לערך 1 כאשר תוצאת החישוב במספרים שלמים בלתי מכוונים גולשת מהתחום של אופרנד היעד.
ד. **דגל הגלישה – OF**, נקבע לערך 1 כאשר תוצאת החישוב של מספרים שלמים מכוונים גולשת מהתחום של אופרנד היעד.
ה. **דגל נשא למחצה – AF**, נקבע לערך 1 כאשר קיים נשא בין סיבית 3 לסיבית 4 והוא משמש בחישוב מספרים המיוצגים בקוד BCD בו בכל בית מאוחסנות שתי ספרות עשרוניות.
ו. **דגל זוגיות – PF**, נקבע לערך 1 כאשר בבית התחתון מספר הסיביות שהן 1 הוא זוגי. הוא משתמש בתקשורת לבדיקה האם בית ששודר ממחשב למחשב התקבל בצורה תקינה.

בספר זה נשתמש רק בארבעת הדגלים הראשונים ואת מנגנון פעולתם נתאר כאן בפירוט.

א. דגל האפס ZERO FLAG (ZF)

משמש לזיהוי תוצאת חישוב שהיא 0. בביצוע הוראה ערכו של דגל האפס מתעדכן על פי הכללים הבאים:

```
אם תוצאת החישוב היא 0 אזי
        ZF = 1
אחרת
        ZF = 0
```

לדוגמה,

```
sub  al, al          ; al = 0 ,  ZF = 1
```

כפי שנראה בהמשך דגל האפס שימושי מאד, למשל כדי לבדוק אם שני נתונים הם שווים או האם ערך של מונה הלולאה התאפס.

שאלה 5.7

האם דגל האפס מורם רק בביצוע הוראת חיסור? נמקו את תשובתכם; אם תשובתכם חיובית – הסבירו מדוע, ואם לא – תנו דוגמה לביצוע הוראת add בה ZF יהיה 1.

ב. דגל הנשא – CF

תפקיד דגל הנשא להצביע על גלישה שהתרחשה כתוצאה מפעולה עם מספרים בלתי מכוונים. בביצוע פעולה דגל הנשא מתעדכן לפי הכלל הבא:

> אם (תוצאת החישוב > מהמספר הבלתי מכוון הגדול שניתן לאחסן באופרנד היעד) או (תוצאת החישוב < 0) אזי
>
> CF = 1
>
> אחרת
>
> CF = 0

בהתאם לאלגוריתם זה נוכל לרשום את שני הכללים הבאים:

א. אם אופרנד היעד הוא מטיפוס בית, אזי גלישה תתרחש אם התוצאה גדולה מ-0FFh או שהתוצאה קטנה מאפס.

ב. אם אופרנד היעד הוא מטיפוס מילה אזי גלישה תתרחש אם התוצאה גדולה מ-0FFFFh או שהתוצאה קטנה מאפס.

נתאר שלוש דוגמאות בהן נעקוב אחר ביצוע הוראות עם אופרנדים מטיפוס בית.

דוגמה 5.2

א. בדוגמה הראשונה יש גלישה כי התוצאה גדולה מהערך המרבי בבית:

```
mov  al, 2          ;al = 2,    CF אין שינוי במצב
add  al, 0FFh       ;al = 1,    CF = 1
```

בביצוע הוראת החיבור ערך דגל הנשא נקבע ל-1 כי תוצאת החיבור גלשה, כלומר
2+0FFh = 101h > 0FFh

ב. בדוגמה השנייה נוצרת גלישה כי התוצאה היא מספר שלילי:

```
mov ah, 09h         ; ah = 09h,   CF אין שינוי במצב
sub  ah,  0FAh      ; ah = 0Fh,   CF = 1
```

ביצוע הוראת החיבור קובעת את ערך דגל הנשא ל-1 כי

09h – 0FAh= –240h < 0

זכרו כי 240h–, מיוצג בשיטת המשלים לשתיים כ-DC0h.

ג. בדוגמה השלישית אין גלישה וערך דגל הנשא הוא 0 :

אם לאחר ההוראה האחרונה בדוגמה ב' נרשום את ההוראה

```
sub  ah, ah          ; ah = 0h,   CF = 0,    ZF = 0
```

התוצאה התאפסה ולכן היא בתחום ודגל הנשא הוא 0, אבל דגל האפס נקבע ל-1.

בביצוע הוראה כלשהי המעבד מעדכן **את כל אחד** מהדגלים שמושפעים מביצוע הוראה זו. כל אחד מהדגלים המתאימים נבדק ובהתאם לאלגוריתם הפעולה שלו וסוג ההוראה מתעדכן מצבו, ולכן יתכן שבעקבות הוראה מסוימת ישתנה מצבם של מספר דגלים.

שאלה 5.8

רשמו בכל הוראה את מצב דגל הנשא ודגל הסימן :

א.
```
mov cl, 0FEH
add cl, 1
```

ב.
```
mov bl, 0FEh
add bl, 2
```

ג.
```
mov bl, 0FFh
inc bl
```

ד.
```
sub dl, dl
add dl, 0FFh
```

ה.
```
mov 10h
mov bl, 10
sub bl, al
add bl, al
```

ג. דגל הסימן (SF)

דגל הסימן מסמן אם המספר המכוון הוא חיובי או שלילי וערכו נקבע על פי האלגוריתם הבא:

אם הסיבית המשמעותית ביותר בתוצאה היא 1 אזי
SF = 1
אחרת
SF = 0

דגל הסימן שימושי כאשר אנו מבצעים פעולה אריתמטית עם מספרים מכוונים. במקרה כזה, אם ערכו של דגל הסימן הוא 1, הדבר מצביע על כך שהתוצאה שלילית ואם ערכו הוא 0 אז התוצאה חיובית. נדגים את השימוש במספרים מכוונים לכתיבת תכנית לחישוב מפלס מי הכנרת.

דוגמה 5.3

יש לכתוב קטע של תכנית המחשב את גובה מפלס מי הכנרת כיום. ידוע כי הקו האדום העליון של מפלס הכנרת נקבע ל-208– מטר. בינואר 2003 נמדד גובה המפלס והוא עמד על 214– מטר (שהוא נמוך מהקו האדום העליון), ומאז ועד היום עלה המפלס ב-5 מ'. התכנית תחשב:

א. את גובה מפלס מי הכנרת היום.
ב. כמה מטרים חסרים מהמפלס היום כדי שהמפלס יעמוד על הקו האדום?

פתרון

תחילה נגדיר את המשתנים הדרושים לאחסון גובה המפלס הנוכחי והקו האדום. נזכור כי תחום המספרים המכוונים שניתן לאחסן בבית הוא בין 127+ לבין 128-, ולכן כדי לאחסן את הנתונים על מפלס מי הכנרת נגדיר משתנים מטיפוס מילה:

```
.DATA
    waterLevel   DW   –214        ; waterLevel = 0FF2Ah
    redLine      DW   –208        ; redLine = 0FF30h
```

האסמבלר ממיר את המספרים השליליים ומייצג אותם בשיטת המשלים לשתיים.

כעת נרשום בטבלה 5.4 את ההוראות לביצוע החישובים ונתאר את מצב הדגלים לאחר ביצוע החישוב:

טבלה 5.4

מצב הדגלים	תיאור של ביצוע ההוראה	ההוראה
הדגלים לא מושפעים מפעולת העברה	; ax ← 0FF2Ah	mov ax, waterLevel
CF=0 SF=1	; ax ← ax + 5 = 0FF2Fh	add ax, 5
CF=0 SF=0	;redLine ← redLine – ax = 0FF30h – 0FF2Fh = 01h	sub redLine, ax

לאחר ביצוע ההוראה השנייה, גובה מפלס מי הכנרת הוא שלילי ולכן ערכו של דגל הסימן הוא 1. אולם לאחר ביצוע ההוראה השלישית, תוצאת החישוב היא חיובית ולכן בסיום ערך דגל הסימן הוא 0.

שאלה 5.9

נשנה את ההוראה האחרונה בדוגמה ונרשום: SUB ax, redline. חשבו את ערכו הסופי של האוגר ax ואת ערכם של דגל הנשא ודגל הסימן.

ד. דגל הגלישה במספרים מכוונים OF

תפקיד דגל הגלישה הוא להצביע על גלישה שהתרחשה כתוצאה מפעולה על מספרים מכוונים. דגל הנשא מתעדכן לפי האלגוריתם הבא:

אם (תוצאת החישוב > מהמספר המכוון הגדול ביותר שניתן לאחסן באופרנד היעד) או
(תוצאת החישוב < מהמספר המכוון הקטן ביותר שניתן לאחסן באופרנד היעד) אזי
OF = 1
אחרת
OF = 0

בהתאם לאלגוריתם זה נוכל לרשום את שני הכללים הבאים:

א. אם אופרנד היעד הוא מטיפוס בית, אזי גלישה תתרחש אם התוצאה גדולה מ-127+ או שהתוצאה קטנה מ-128–.

ב. אם אופרנד היעד הוא מטיפוס מילה אזי גלישה תתרחש אם התוצאה גדולה מ-32767+ או שהתוצאה קטנה מ-32768–.

נתאר דוגמה שבה החישוב משפיע על דגל הגלישה.

דוגמה 5.4

במדידות טמפרטורה שהתבצעו ברוסיה, נמדדה בקיץ טמפרטורת שיא של 65 מעלות ואילו בחורף הטמפרטורה הנמוכה ביותר שנמדדה הייתה 75– מעלות. יש לכתוב קטע של תכנית שיחשב את הפרש הטמפרטורות בין הקיץ לחורף.

פתרון

הגדרת נתונים:

```
summer  DB 65d        ; summer = 41h
winter  DB – 75d      ; winter = 0B5h
```

הוראות לביצוע החישוב:

```
mov al,  summer       ; al ← 41h
sub al,  winter       ; al ← 41h – B5h = 8Ch,   OF =1, SF = 1
```

דגל הסימן SF הוא 1 משום שהסיבית המשמעותית ביותר היא 1 ואילו דגל הגלישה OF מסמן שהתוצאה חרגה מתחום הערכים המכוונים שניתן לאחסן בבית אחד, ואכן:

$$65 – (75) = 140 > 127$$

נרשום גרסה מתוקנת לקטע התכנית ובה נשנה את טיפוס המשתנים למילה.

הגדרת נתונים:

```
summer  DD 65d        ; summer  = 0041h
winter  DD – 75d      ; winter = 0FFB5h
```

הוראות לביצוע החישוב:

```
mov ax,   summer      ; ax ← 0041h
sub ax,   winter      ; ax ← 0041h – 0FFB5h = 008Ch,   OF =0, SF = 0
```

כעת, ערך התוצאה הוא 140 > 32767 ולכן דגל הגלישה OF "מורד", דהיינו: שווה ל-0.

האם להשתמש בדגל הנשא או בדגל הגלישה?

בכל ביצוע הוראה המשפיעה על הדגלים, המעבד משנה באופן אוטומטי את מצבם בהתאם לתוצאה שהתקבלה, אבל את המשמעות להם מעניק המתכנת בשפת אסמבלי והוא מחליט כיצד להשתמש במידע זה. בדוגמה 5.4, המחשבת את הפרש הטמפרטורות, לא התייחסנו לדגל הנשא CF, אך אם נבחן את ערכו נבחין כי בביצוע שתי פעולות החיסור שהצגנו, ערכו הוא CF = 1. כלומר התוצאה גלשה מגודל של בית במקרה הראשון ומגודל של מילה במקרה השני. אולם המספרים בהם השתמשנו היו קטנים מ-255; כיצד, אם כך, התרחשה גלישה?

אם נתייחס למספרים כאל מספרים בלתי מכוונים תוצאת החישוב היא: 41h – B5h < 0 ולפי הכללים לעדכון דגל הנשא, כאשר התוצאה קטנה מ-0 דגל הנשא הוא 1. אבל מאחר שבדוגמה זו ייצגנו את הטמפרטורות כמספרים מכוונים, לא התענינו במצב דגל הנשא. לעומת זאת התייחסנו למצב דגל הגלישה משום שהוא מצביע על חריגה מתחום המותר למספרים מכוונים בהם אנו משתמשים.

ככלל **במספרים בלתי מכוונים אנו מתייחסים למצב דגל הנשא (CF) ובמספרים מכוונים אנו מתייחסים למצב דגל הגלישה (OF).**

שאלה 5.10

רשמו את מצב דגל הנשא ודגל הגלישה לאחר ביצוע כל הוראה מההוראות הבאות:

```
mov  al, 0A9h
add  al, 010h
sub  al, 0D0h
add  al, 0FEh
```

5.3.4 הוראות חיבור וחיסור נוספות

בנוסף להוראות ADD ו-SUB קיימות הוראות נוספות ששייכות לקבוצת ההוראות האריתמטיות. נציג אותן בסעיף זה.

א. חיבור וחיסור עם נשא

במעבד 8086 יש הוראות נוספות לחיבור וחיסור **עם נשא**. הוראות אלו מאפשרות חישוב עם מספרים ארוכים. לדוגמה, כדי לחבר שני מספרים מטיפוס מילה כפולה, אנו יכולים להשתמש באוגרים מטיפוס מילה ולבצע את החיבור בשני שלבים, כאשר בכל פעם מחברים זוג מספרים מטיפוס מילה, תוך התחשבות בנשא שמועבר מחיבור זוג המילים הראשון לחיבור זוג המילים השני. לדוגמה,

נשא 1

+ 1234	+ 5678
C9BA	D876
DBEF	2EEE

לביצוע פעולות מסוג זה קימות במעבד 8086 שתי הוראות:

חיבור עם נשא ADC (ADd with Carry)

ADC אופרנד יעד, אופרנד מקור

שמשמעותה: CF + אופרנד המקור + אופרנד היעד ← אופרנד היעד

הוראת חיסור עם לווה – SBB (SuBtract with Borrow):

SBB אופרנד יעד, אופרנד מקור

שמשמעותה: CF – אופרנד המקור – אופרנד היעד ← אופרנד היעד

בהוראות אלו צירופי האופרנדים האפשריים הם בדומה לצירופים האפשריים שהוצגו בטבלה 5.3, כלומר: אופרנד המקור יכול להיות אוגר, משתנה בזיכרון או נתון מיידי, ואופרנד היעד יכול להיות אוגר או תא בזיכרון.

ההוראות ADC ו-SBB משפיעות על כל דגלי המצב: ZF, SF, CF, OF, PF ו-AF

נדגים את השימוש בהוראה לחיבור מספרים ארוכים מטיפוס מילה כפולה.

דוגמה 5.5 חיבור שני מספרים מטיפוס מילה כפולה

נכתוב הוראות לחיבור שני המספרים:

12345678h
+ C9BAB876h

פתרון

את המספר הראשון 12345678h נאחסן באוגרים AX ו-BX ואת המספר השני C9BAB876h נאחסן באוגרים CX ו-DX ואחר נחבר כל זוג מספרים:

```
mov  ax, 5678h                                ; ax ←    5678h
mov  bx, 1234h                                ; bx ←    1234h
mov  cx,  0B876h                              ; cx ←  0B876h
mov  dx,  0C9BAh                              ; dx ← 0C9BAh
add  ax, cx        ; ax ← 5678h + 0B876h = 2EEEh          CF = 1
adc  bx, dx        ; bx ← 1234h + 0C9BAh + 1 = 0DBEFh      CF = 0
```

שאלה 5.11

א. האם בדוגמה 5.5 ניתן להחליף את ההוראה לפני האחרונה

```
add ax, cx
```

בהוראה:

```
adc ax, cx
```

נמקו את תשובתכם.

ב. כתבו הוראות לחיסור שני המספרים מטיפוס מילה כפולה המוצגים בדוגמה 5.5.

ב. ההוראות INC ו-DEC

תכניות רבות כוללות פעולות שמעדכנות ערכי מונים על-ידי הוספת 1 למונה או הפחתה של 1 ממונה, ולכן בכל שפת אסמבלי קיימות שתי הוראות למימוש פעולות אלו.

ההוראה INC (INCrement)

מגדילה ערך אופרנד היעד באחד

inc אופרנד יעד

ומשמעותה: אופרנד יעד ← אופרנד יעד + 1

ההוראה DEC (DECrement)

מקטינה את ערך אופרנד היעד באחד

dec אופרנד יעד

ומשמעותה: 1 – אופרנד יעד ← אופרנד יעד

אופרנד היעד בהוראות אלו יכול להיות אוגר או תא בזיכרון כמוצג בטבלה 5.5.

טבלה 5.5
סוגי אופרנדים אפשרי בהוראות **INC** ו-**DEC**

אופרנד היעד	דוגמאות	
אוגר	dec al	inc al
תא בזיכרון	dec var	inc var

ההוראות INC ו-DEC משפיעות על דגלי המצב: ZF, SF, OF, PF ו-AF אך לא על דגל הנשא CF.

הוראות אלו נכללות באוסף ההוראות של מעבד 8086 למרות שניתן לממשן באמצעות הוראות רגילות, למשל:

```
add al, 1
sub al, 1
```

הסיבה לכך היא שזמן הביצוע של ההוראות INC ו-DEC הוא קצר יותר מביצוע אותן הפעולות באמצעות ADD ו-SUB, ולכך יש חשיבות רבה בתוכניות שבהן עדכון מונה היא פעולה שכיחה ביותר בתכנות. בפרק הבא נתייחס ביתר פירוט לזמן ביצוע של הוראה, שהוא פרמטר חשוב ביותר המשפיע למשל על בחירת הוראות לכתיבת תכנית בשפת אסמבלי, או בפיתוח קומפיילר בו יש לתרגם הוראות בשפה עילית לשפת מכונה.

ג. הוראה לחישוב המשלים לשתיים

בשפת אסמבלי קיימת הוראה לחישוב המשלים לשתיים של מספר.

ההוראה neg (NEGate)

NEG אופרנד יעד

ההוראה מבצעת את הפעולה הבאה:

אופרנד יעד – 0 ← אופרנד יעד

אופרנד היעד יכול להיות אוגר או תא בזיכרון כמתואר בטבלה 5.5.

לדוגמה, אנו יכולים לאחסן את הערך 208 במשתנה redLine:

```
redLine  DW 208d
```

ולהשתמש בהוראה neg כדי להציג את הערך השלילי 208– בשיטת המשלים ל-2:

```
neg  redLine                                  ; redLine ← 0FF30h
```

הוראה זו משפיעה על דגל הסימן ועל דגל הנשא.

שאלה 5.12

א. כתבו הוראות בשפת אסמבלי למימוש ההוראה neg (ללא שימוש בהוראה זו)

ב. רשמו את מצב הדגלים בעקבות ביצוע ההוראות הבאות:

```
mov  al,  0Ah
neg   al
neg    al
```

5.4 הוראות בקרה

ההוראות שדנו בהן עד כה מתבצעות באופן סדרתי, כאשר בשלב ההבאה של מחזור ההבאה-ביצוע של ההוראה, האוגר IP מתעדכן ומצביע תמיד על ההוראה העוקבת לה בזיכרון. ואולם, פעמים רבות יש לשנות את הביצוע הסדרתי של התכנית כדי לבצע הוראות המותנות בנתונים מסוימים או כדי לחזור ולבצע הוראות מסוימות מספר פעמים. במקרים אלו, ההוראה הבאה לביצוע אינה בהכרח ההוראה העוקבת להוראה הנוכחית, ולכן בתכניות כאלה נצטרך לרשום הוראות שמשנות את תוכן אוגר IP בצורה יזומה. קבוצת ההוראות שביצוען משפיע על תוכן אוגר IP נקראת "הוראות להעברת בקרה". בסעיף זה נציג את הוראות הקפיצה, שהן חלק מקבוצת ההוראות להעברת בקרה, ונדגים את השימוש בכתיבת הוראות תנאי ולולאות. הוראות הקפיצה נחלקות לשתי קבוצות:

א. הוראות בהן בקרת התכנית עוברת לכתובת המטרה ללא תלות בקיום תנאים כלשהם.

ב. הוראות בהן בקרת התכנית עוברת לכתובת המטרה רק אם התקיימו תנאים מסוימים.

5.4.1 הוראת קפיצה שאינה מותנית – JMP (JuMP)

הוראת הקפיצה שאינה מותנית מתבצעת ללא התניה:

JMP operand

האופרנד בהוראה זו היא כתובת בזיכרון. במקום כתובת זו ניתן להשתמש בשם הנקרא "**תווית**" (label) או "כתובת סמלית". התווית מציינת את כתובת ההוראה במקטע הקוד. ניתן להתייחס לתווית כאל כתובת סמלית המציינת הוראה מסוימת אליה היא מוצמדת, באופן דומה לשם משתנה שהוא כתובת סמלית המציינת כתובת של תא בזיכרון (שנמצא במקטע הנתונים). לדוגמה ההוראה

JMP label

משמעותה: קפוץ להוראה שאליה מוצמדת התווית.

כאשר תכנית נטענת לזיכרון לשם הרצה, מוחלפת התווית בכתובת האקטואלית שבה מאוחסנת ההוראה אליה היא צמודה, ובשלב ביצוע ההוראה, מוחלף תוכנו של IP בכתובת עליה מצביעה התווית כלומר

IP ← label

את התווית להוראה מגדירים על-ידי שם ולאחריו שימוש בנקודתיים הנרשמים משמאל להוראה. לדוגמה נצמיד את התווית nextSection להוראת MOV

```
nextSection:     mov  cx, 0Fh
```

ואז, לדוגמה, ההוראה

```
jmp nextSection
```

משמעותה היא: קפוץ להוראה שאליה מוצמדת התווית, כלומר להוראה mov cx, 0Fh.
הוראה זו אינה משנה את מצב הדגלים.

אנו נקפיד לבחור לתווית שם משמעותי אותו נרשום באותיות קטנות. אם התווית מכילה מספר מילים נתחיל כל מילה (החל מהמילה השנייה) באות גדולה. כמובן שתווית לא יכולה להיות מילה שמורה כמו הוראה או הנחיית אסמבלר.

נשתמש בדוגמה 5.6 כדי לעקוב אחר מהלך ביצוע הוראת jmp.

דוגמה 5.6

נתון קטע של תכנית בו ארבע הוראות ועלינו לחשב מה יהיה ערכו של al בסיומו.

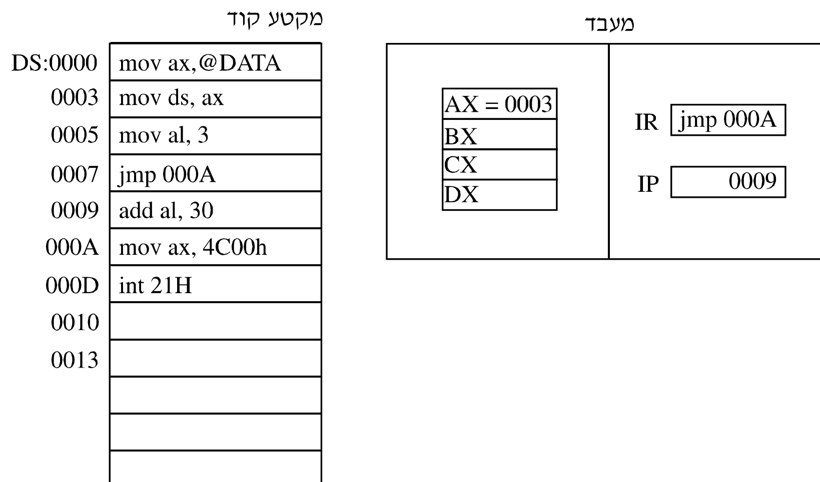
```
      mov  al,  3h
      jmp   next
      add   al,  12h
next:  add   al,  30h
```

פתרון

בקטע זה רשמנו הוראת קפיצה בלתי מותנית בה התווית בשם next המוצמדת להוראה הרביעית. כלומר לאחר שתתבצע ההוראה השניה (היא הוראת הקפיצה הבלתי מותנית), ההוראה הבאה לביצוע תהיה ההוראה הרביעית ולא ההוראה השלישית. לכן בסיום הקטע ערכו של אוגר al יהיה:

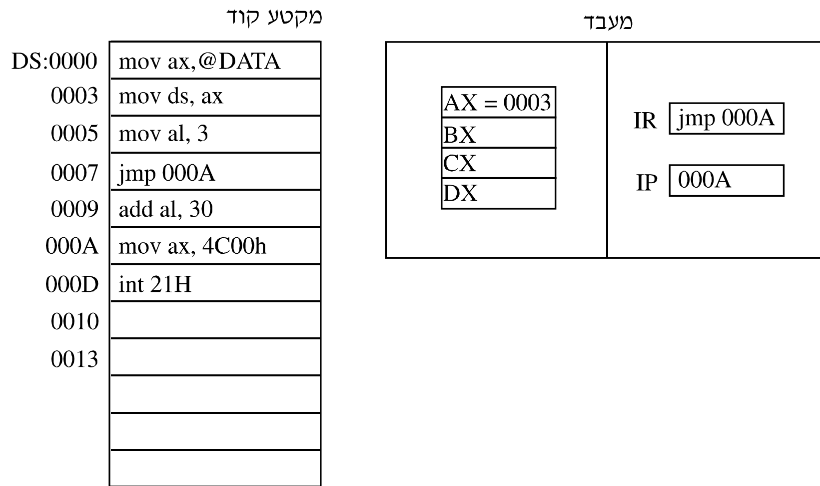
al = 3h + 30h = 33h

נעקוב אחר מחזור ההבאה-ביצוע של הוראת הקפיצה הבלתי מותנית בעזרת איור המתאר את מצב האוגרים והזיכרון. בסיום שלב ההבאה של ההוראה jmp next המוצג באיור 5.2 א' אוגר IP מכיל את כתובת ההוראה החמישית, כלומר את הכתובת 0009.



איור 5.2א
מצב הזיכרון והאוגרים לאחר סיום שלב ההבאה של ההוראה **jmp next**

לאחר שלב הביצוע ערכו של IP משתנה שוב והפעם מושמת בו הכתובת של ההוראה אליה הוצמדה התווית next, כלומר ההוראה שבכתובת 000Ah



איור 5.2 ב
מצב הזיכרון והאוגרים לאחר סיום ביצוע ההוראה **jmp next**

שאלה למחשבה: מתי מתבצעת ההוראה add al, 12h ?

5.4.2 הוראות קפיצה מותנית

בקבוצה זו מספר הוראות בהן בקרת התכנית עוברת לכתובת המטרה רק אם התקיימו תנאים מסוימים המוגדרים בעזרת אוגר הדגלים. לדוגמה ההוראה JZ (Jump if Zero, כלומר "קפוץ אם אפס") בודקת אם דגל האפס היא 1 ואילו ההוראה JA (Jump if Above, כלומר "קפוץ אם גדול מ-") בודקת אם דגל הנשא וגם דגל האפס הם אפס. אם ערכם של הדגלים הנבדקים מקיים את התנאים הדרושים, התכנית ממשיכה את הביצוע מההוראה אליה מוצמדת תווית, אחרת הביצוע ממשיך בהוראה העוקבת להוראת הקפיצה. נתאר באלגוריתם את אופן הביצוע של הוראת קפיצה מותנית:

אם בדיקת הדגלים מצליחה אזי
    IP ← label הכתובת שמייצגת
אחרת
    IP← כתובת ההוראה העוקבת להוראת הקפיצה המותנית

המבנה של כל הוראות הקפיצה דומה, והוא מכיל אופרנד אחד המציין תווית של ההוראה שיש לדלג אליה.

Jxxx תווית

השמות המנמוניים של ההוראות האלה מתחילים באות J; המשך השם xxx הוא רק "מחזיק מקום" בהוראה, ואין לרשום שם ממש שלוש פעמים האות x, אלא אות או צירוף אותיות המציין את התנאי לקפיצה. בדוגמאות למעלה, למשל, במקום x רשמנו Z או A. בדומה להוראת הקפיצה הבלתי מותנית, בזמן טעינת התכנית לזיכרון, מתורגמת התווית לכתובת של ההוראה אליה מוצמדת התווית.

במעבד 8086 יש אילוץ על המרחק אליה ניתן לקפוץ בקפיצות מותנות: כל הקפיצות המותנות חייבות להתבצע למען שאת מרחקו בינו לבין מען פקודת הקפיצה המותנית עצמה ניתן לרשום כמספר מכוון מטיפוס בית. כלומר, הקפיצה יכולה להיות עד 127 בתים קדימה, או עד 128 בתים אחורה. קפיצה למרחק גדול יותר תוגדר על-ידי האסמבלר כשגיאה. במידה שאנו רוצים לבצע קפיצה למרחק גדול יותר, אנו יכולים להשתמש בשתי הוראות: תחילה לבצע קפיצה מותנית למען חוקי ובו לרשום הוראה לקפיצה בלתי מותנית לכל יעד במקטע הקוד.

את הוראות הקפיצה המותנות ניתן לחלק לשלושה סוגים:

א. הוראות קפיצה מותנות הבודקות ערך של דגל מסוים.

ב. הוראות קפיצה מותנות עבור מספרים בלתי מכוונים.

ג. הוראות קפיצה מותנות עבור מספרים מכוונים.

נתאר כל קבוצה ונדגים את השימוש בהן.

הוראות הקפיצה המותנות הבודקות ערך של דגל מסוים מרוכזות בטבלה 5.6.

טבלה 5.6
הוראות קפיצה מותנות המותנות במצב דגל

התנאי הנבדק	תיאור	הוראה
CF = 1	קפוץ אם דגל הנשא דלוק	JC (Jump if Carry)
CF = 0	קפוץ אם דגל הנשא כבוי	JNC (Jump if Not Carry)
ZF = 1	קפוץ אם דגל האפס דלוק	JZ (Jump if Zero)
ZF = 0	קפוץ אם דגל האפס כבוי	JNZ (Jump if Not Zero)
SF = 1	קפוץ אם דגל הסימן דלוק	JS (Jump if Sign)
SF = 0	קפוץ אם דגל הסימן כבוי	JNS (Jump if Not Sign)
OF = 1	קפוץ אם דגל הגלישה דלוק	JO (Jump if Overflow)
OF = 0	קפוץ אם דגל הגלישה כבוי	JNO (Jump if Not Overflow)
PF = 1	קפוץ אם דגל הזוגיות דלוק	JP (Jump if Parity)
PF = 0	קפוץ אם דגל הזוגיות כבוי	JPO (Jump if Not Parity)

כדי להשתמש בהוראת קפיצה מותנית עלינו תחילה לבצע הוראה שתשפיע על הדגל המתאים. נדגים זאת על-ידי חישוב ערך מוחלט של מספר מכוון.

דוגמה 5.7 חישוב ערך מוחלט

נרשום תחילה אלגוריתם מתאים לחישוב ערך מוחלט של מספר מכוון אותו אנו מסמנים ב-x:

אם $x < 0$ אזי
$x \leftarrow (-x)$
סיום-אם

כדי לבדוק אם המספר שלילי אנו יכולים להשתמש בדגל הסימן SF המושפע מערך הסיבית המשמעותית ביותר במספר, היא סיבית הסימן. כדי שדגל הסימן יעודכן בהתאם לערך סיבית הסימן של המשתנה x, עלינו תחילה לבצע פעולה אריתמטית כלשהי שתשנה את דגל הסימן בהתאם לערך המספר x. הפעולה המתאימה בבעיה זו היא חיסור 0 מ-x, כלומר: $x - 0$. לאחר חישוב זה, דגל הסימן ישתנה בצורה הבאה:

- אם x הוא חיובי או אפס, אז תוצאת החישוב היא $(x - 0) \geq 0$ ולכן דגל הסימן הוא 0
- אם x הוא שלילי אז תוצאת החישוב קטנה מאפס $(x - 0) < 0$ ולכן דגל הסימן הוא 1

לאחר ביצוע פעולת החיסור נבדוק את מצב דגל הסימן באמצעות הוראת קפיצה מותנית ונפעל בהתאם.

קיימות שתי הוראות קפיצה המותנות בדגל הסימן: JS בה הקפיצה להוראה אחרת מתבצעת כאשר דגל הסימן הוא 1, והוראה JNS בה הקפיצה להוראה אחרת מתבצעת כאשר דגל הסימן הוא 0. נכתוב שני קטעי הקוד מתאימים לחישוב הערך המוחלט של מספר מכוון. בקטע קוד מס. 1 נשתמש בהוראה JS ובקטע קוד השני נשתמש בהוראה JNS.

קטע קוד מס. 1: שימוש בהוראה JS	קטע קוד מס. 2: שימוש בהוראה JNS
sub x, 0 js doAbs ; x הוא שלילי jmp doNothing doAbs: neg x doNothing: ; המשך התכנית	sub x, 0 jnz doNothing ; x הוא חיובי neg al doNothing: ; המשך תכנית

ניתן לראות כי קטע קוד מס. 2 קצר יותר מקטע קוד מס. 1. בקטע קוד מס. 2 המימוש של מבנה התנאי "אם... אזי..." המתואר באלגוריתם הבא, פשוט יותר ולכן הוא כולל רק הוראת קפיצה מותנית אחת:

1. אם $x > 0$ אזי קפוץ להוראה מס. 3
2. שנה את הסימן של x
3. המשך בתכנית

לעומת זאת, קטע קוד 1 מממש את התנאי "אם... אזי.. אחרת..." בצורה מסורבלת יותר, כמתואר באלגוריתם הבא, ולכן הוא כולל שתי הוראות קפיצה:

1. אם $x < 0$ אזי קפוץ להוראה מס. 3
2. (אחרת) קפוץ להוראה מס. 4
3. שנה את הסימן של x
4. המשך בתכנית

ביצוע תכנית בה שתי הוראות קפיצה יארך זמן רב יותר מאשר תכנית בה הוראת קפיצה אחת, ולכן נעדיף בבעיות דומות להשתמש במבנה המתואר בקטע קוד מס. 2.

הביצוע של הוראת קפיצה מותנית דומה לאופן הביצוע של הוראת קפיצה בלתי מותנית, אך ביצוע הוראת קפיצה מותנית עשוי לשנות את ערכו של אוגר IP בשלב הביצוע או לא, בהתאם לתנאי, בעוד שביצוע הוראת קפיצה מותנית משנה בהכרח את ערכו של אוגר IP בשלב הביצוע (אלא אם כן הכתובת המצוינת בהוראת הקפיצה הבלתי מותנית זהה לכתובת ההוראה הבאה אחריה).

שאלה 5.13

ציירו את מצב אוגרי המעבד בשלב הבאה ובשלב הביצוע של הוראת JNS doNothing בקטע קוד מס. 1. הניחו כי ערכו של x הוא 10.

שאלה 5.14

האם ניתן בקטע קוד מס. 2 להוריד את ההוראה sub ולרשום את ההוראות הבאות:

```
        jnz  doNothing
        neg x
doNothing:                    ; המשך התכנית
```

אם לא הסבירו מדוע.

הוראות קפיצה מותנות נוספות, השימושיות בעיבוד של מספרים מכוונים, והוראות קפיצה מותנות, השימושיות בעיבוד של מספרים בלתי מכוונים מתוארות בטבלה 5.7. בהוראות המעבד בודק דגלים שבאמצעותם ניתן לממש את ששת היחסים הלוגיים: גדול, קטן, שווה, גדול או שווה, קטן או שווה ושונה.

טבלה 5.7
הוראות קפיצה מותנות למספרים מכוונים ולמספרים בלתי מכוונים

תיאור תנאי	הוראות קפיצה למספרים מכוונים	הוראות קפיצה למספרים בלתי מכוונים
קפוץ אם גדול ממש	JG (JNLE) (Jump if Greater או Jump if not less or Equal)	JA (JNBE) (Jump if Above או Jump if Not Below or Equal)
קפוץ אם קטן ממש	JL (JNGE) (Jump if Less או Jump if not Greater or Equal)	JB (JNAE) (Jump if Below או Jump if Not Above or Equal)
קפוץ אם גדול או שווה	JGE (JNL) (Jump if Greater or Equal או Jump if not less)	JAE (JNB) (Jump if Above or Equal או Jump if Not Below)
קפוץ אם קטן או שווה	JLE (JNG) (Jump if Less or Equal או Jump if not Greater)	JBE (JNA) (Jump if Below or Equal או Jump if Not Above)
קפוץ אם שווה	JE (Jump if Equal)	JE (Jump if Equal)
קפוץ אם שונה	JNE (Jump if Not Equal)	JNE (Jump if Not Equal)

כפי שניתן לראות מן הטבלה, ישנו מספר תנאים, שלכל אחד מהם יש מספר הוראות בהן ניתן להשתמש.לדוגמה, כדי לבדוק אם X **גדול ממש** מ-Y, כאשר X Y הם מספרים בלתי מכוונים, ניתן להשתמש:

- בהוראה JA (קיצור של Jump Above) , ובה התנאי הנבדק הוא האם $X > Y$?
- או בהוראה JNBE (קיצור של Jump Not Below), שבה התנאי הנבדק הוא האם NOT $(X \leq Y)$.

שאלה 5.15

שנו את קטע הקוד שתואר בדוגמה 5.7 והשתמשו בהוראה קפיצה מותנית למספרים מכוונים כדי לחשב את הערך המוחלט של המספר x.

לכאורה יש כפילות בהוראות קפיצה מותנות עבור מספרים בלתי מכוונים ועבור מספרים מכוונים. כדי להבין מדוע אנו זקוקים לאוסף הוראות נפרד לכל מקרה, נתבונן בדוגמה הבאה.

דוגמה 5.8

קטע התכנית הבא מבצע את האלגוריתם הבא:

```
אם  al < ah אזי
      inc ah
סיום-אם
      inc ah
```

כלומר אם AL < AH אזי בסיום קטע התכנית ערכו של AH יגדל ב-2, אחרת ערכו יגדל ב-1.

נרשום הוראות אסמבלי עם דוגמת ערכים, למימוש האלגוריתם:

```
mov  al, 11111110b
mov  ah, 00000010b
sub   al, ah                    ; CF=0,  SF=1, OF=1  :מצב הדגלים
                                ; al < ah ?
         jb next                ; next-אם מתקיים  al < ah  אזי המשך מ
                                ; al >= ah     אחרת מתקיים
         inc ah
next:    inc ah
```

כעת נבחן שני מקרים:

א. מקרה ראשון, כאשר אנו מתייחסים לנתונים כאל מספרים בלתי מכוונים ולכן

$11111110b - 00000010b > 0$

במקרה כזה אנו לאחר ביצוע הוראת הקפיצה JB next, המשך ביצוע התכנית הוא מההוראה העוקבת ובסיום וערכו של AH יגדל ב-2.

ב. מקרה שני: נניח כי המספרים הם מכוונים ולכן

$$11111110b - 00000010b < 0$$

אם נשנה את הוראת הקפיצה ובמקום JB next ונרשום הוראת קפיצה מותנית למספרים מכוונים, כלומר:

```
JL next
```

ביצוע התכנית יימשך מההוראה אליה מוצמדת התווית next. במקרה כזה ערכו של AH יגדל ב-1.

כמובן שאת אותו אלגוריתם ניתן לממש גם באמצעות הוראות הקפיצה המותנות שתיארנו בטבלה 5.6. למשל נשתמש בדגל הסימן ובעזרתו נבדוק האם $AL - AH < 0$. אבל כדי לשפר את קריאות התכנית אנו נעדיף להשתמש בהוראות שבהן השם המנמוני מצביע גם סוג הפעולה כמוצג בטבלה 5.7.

הוראה להשוואה CMP (compare)

לעתים קרובות, יש צורך להשוות את תוכנם של משתנים בתכנית. אפשר לעשות השוואה כזו על-ידי הפחתה של משתנה אחד ממשתנה שני, באמצעות ההוראה SUB. ואולם הוראה זו משנה את תוכנו של אופרנד היעד, ולא תמיד שינוי כזה רצוי, משום שבדרך-כלל, בהשוואה, תוצאת החיסור אינה נדרשת. ההוראה CMP של ה-8086 נועדה לערוך השוואה בין אופרנדים מבלי לשנות את תוכנם. כתוצאה מביצוע הוראה זו יעודכנו דגלי המצב, כפי שמתרחש בעקבות ההוראה SUB.

מבנה הוראת CMP הוא:

CMP אופרנד יעד, אופרנד מקור

שמשמעותה הלוגית היא: באיזה תחום נמצא ההפרש בין אופרנד המקור לאופרנד היעד?

כדי לבצע פעולה זו, המעבד מבצע את פעולת החיסור, כלומר אופרנד המקור פחות אופרנד היעד, ומעדכן את דגלי המצב: CF, SF, OF, AF, PF בהתאם לתוצאה שהתקבלה. ביצוע הוראה זו אינו משנה את ערכו של אופרנד היעד. בדרך-כלל, לאחר ההוראה CMP, מופיעה בתכנית הוראת קפיצה המותנית במצב הדגלים.

צירופי אופרנדים המותרים בהוראה זו מתוארים בטבלה 5.3.

דוגמה 5.9

לדוגמה, נכתוב תכנית הממיינת שני ערכים x, y, כך ש-x יכיל את הערך הגדול מבין השנים, ו-y את הערך הקטן.

תחילה נרשום אלגוריתם למיון שני ערכים:

אם $x - y < 0$ אזי
 החלף את x ו-y ביניהם
סיום-אם

נניח כי x ו-y מוגדרים במקטע הזיכרון כמספרים בלתי מכוונים מטיפוס בית ונרשום את קטע הקוד המתאים:

```
    mov  al, x          ; al ← x
    cmp  al, y          ; x – y?
    jae end_if          ; אם x >= y קפוץ להוראה אליה צמודה ההתווית end_if
    xchg al, y          ; החלף al עם y
    mov  x, al          ; x ← al
end_if:
```

שאלה 5.16

כתבו קטע תכנית הבודקת אם ערכו של AX קטן מ-100 ואם כן מכפילה את ערכו. לדוגמה אם AX = 67, ערכו יוכפל ובסיום ערכו של AX הוא 134.

דוגמה 5.10

הדוגמה הבאה מתארת שימוש בהוראות הקפיצה המותנות למימוש מבנה מותנה "אם...אזי...":

נניח כי a, b ו-m הם משתנים בלתי מכוונים מטיפוס בית. ברצוננו לכתוב בשפת אסמבלי את התנאי המתואר בפסאודו-קוד הבא:

אם a = b אזי
m ← m + 1
אחרת
m ← m – 1
סיום אם

כעת נרשום את ההוראות המתאימות בשפת אסבמלי:

```
        mov al, a                ; al ← a
        cmp al, b                ; al – b

        jne  doAction1           ; if a<>b then jump doAction1
        inc m                    ; m ← m + 1
        jmp continue             ; continue קפוץ לסיום התנאי לתווית
doAction1:  dec m                ; m ← m – 1
continue:                        ;סיום התנאי והמשך התכנית
```

שאלה 5.17

א. האם ניתן להחליף את הוראת הקפיצה המותנית JNE, בהוראה JNZ?

ב. כתבו את קטע התכנית מחדש, אך במקום ההוראה JNE השתמשו בהוראה JE.

ג. האם לדעתכם עדיף להשתמש בהוראה JE ולא בהוראה JNE למימוש קטע קוד זה? הסבירו את תשובתכם.

ד. האם קטע התכנית הבא מבצע את אותה הפעולה שמתבצעת בדוגמה 5.9?

```
        inc m
        mov al, a
        cmp al, b
        je  continue
        dec m
        dec m
continue:
```

שאלה 5.18

כתבו קטע תכנית שממיין שלושה ערכים x, y, z, בסדר עולה כך ש: x> y > z.

שאלה 5.19

במפעל לייצור פלסטיק קיימת מערכת אלקטרונית שתפקידה לבקר את פעולת התנורים שבהם מתיכים את הפלסטיק. לכל תנור מחוברים שני מדי טמפרטורה. מד טמפרטורה א' מודד טמפרטורה חיצונית (במעטפת התנור) ומד טמפרטורה ב' מודד טמפרטורה פנימית (בגוף התנור). מערכת הבקרה נועדה לפיקוח על הפרשי הטמפרטורה בין גוף התנור לבין המעטפת שלו, כך שהטמפרטורה הנמדדת בגוף התנור תהיה תמיד גבוהה לפחות ב-50 מעלות צלזיוס מן הטמפרטורה במעטפת התנור. אם הפרש הטמפרטורה בין גוף התנור לבין המעטפת שלו הוא 50 מעלות צלזיוס או פחות, יש להוציא אות להפעלת החימום ואות להפסקת הקירור, ואם ההפרש ביניהן גדול מ-50 מעלות צלזיוס, יש להוציא אות המפסיק את פעולת החימום ואות המפעיל את פעולת הקירור. הניחו כי הטמפרטורה שמודד מד הטמפרטורה ב' תמיד גבוהה מהטמפרטורה שמודד מד הטמפרטורה א', וכי הטמפרטורות הנמדדות הן בטווח שבין 0 מעלות צלזיוס ל-250 מעלות צלזיוס.

כתבו קטע של תכנית המדמה את פעולת מערכת הבקרה. הניחו כי הטמפרטורה שנמדדת על-ידי מד הטמפרטורה א' נשמרת בזיכרון בתא בשם Temp_Out, והטמפרטורה שנמדדת על-ידי מד הטמפרטורה ב' נשמרת בזיכרון בתא בשם Temp_In. כמו כן, הניחו כי הערך הקבוע 00 מציין כי יש להפעיל את החימום ולהפסיק את את הקירור, ואילו הערך הקבוע FF מציין שיש להפסיק את החימום ולהפעיל את הקירור.

דוגמה 5.11

בדוגמה זו נכתוב תכנית בה נשתמש בהוראות הקפיצה המותנות למימוש מבנה מותנה רב-ברירה (case-of). התכנית מממשת את הפונקציה המתוארת בטבלה הבאה :

טבלה 5.8
תיאור פונקציה

F(x)	Y
$x < 10$	1
$10 \le x < 20$	2
$20 \le x < 40$	3
$x \ge 40$	4

נשתמש בתבנית של מבנה case לכתוב תכנית מתאימה :

```
.model small
.stack 100h
.data
                                        ; הצהרה על משתנים
   x  DB 10
   y  DB ?

.CODE
start:
                                        ; אתחול מקטע הנתונים
   mov  ax,  @DATA
   mov  ds,  ax
                              ; action1 קפוץ לתווית x <10  אם
   cmp  x,    10
   jb  action1
```

```
; if 10 <= x < 20 goto action2
    cmp     x, 20
    jb      action2
; if 20 <= x < 40 goto action3
    cmp     x, 40
    jb      action3
; x >= 40 goto action4
    jmp     action4
action1:  mov y,1
    jmp     endcase
action2:  mov y,2
    jmp     endcase
action3:  mov y,3
    jmp     endcase
action4:  mov y,4
    jmp     endcase
endcase:
    mov     ax,  400h        ; סיום תכנית
    int     21h
END start
```

ביצוע חוזר (לולאות)

הוראות לביצוע חוזר מאפשרות לבצע קטע של התכנית מספר פעמים עד שמתקיים תנאי לסיום הלולאה. בסעיף זה נתאר מימוש בשפת אסמבלי של שני מבנים לביצוע חוזר:

- ביצוע חוזר באורך מחושב מראש
- ביצוע חוזר בתנאי

דוגמה 5.11

כדי להדגים ביצוע חוזר בתנאי נכתוב קטע תוכנית לדוגמה בו נחבר את סדרת המספרים הטבעיים החל מ-1. החיבור יתבצע כל עוד סכום המספרים קטן מ-100.

תחילה, נרשום שני אלגוריתמים המתארים שני פתרונות:

אלגוריתם מס 1:	אלגוריתם מס 2:
$x \leftarrow 1$ $sum \leftarrow 0$ כל עוד $sum < 100$ בצע: $sum \leftarrow sum + x$ $x \leftarrow x + 1$	$x \leftarrow 1$ $sum \leftarrow 0$ בצע: $sum \leftarrow sum + x$ $x \leftarrow x + 1$ עד ש- $sum \geq 100$

כדי לכתוב תכנית בשפת אסמבלי, נגדיר תחילה את המשתנים:

- אוגר AL יכיל את המספרים בסדרה
- אוגר BL יכיל את סכום המספרים

שימו לב: מאחר שסכום המספרים שאנו מחברים קטן מ-100, אנו מסתפקים באוגרים מטיפוס בית.

כעת נכתוב שני קטעי תכנית מתאימים לאלגוריתמים שתיארנו קודם לכן:

מימוש אלגוריתם מס 1

```
;אתחול הערכים
          mov al, 1
          mov bl, 0
; BL < 100 כל עוד
doAgain:  cmp bl, 100
          jae endLoop
;גוף הלולאה
          Add bl, al
          inc al
          jmp doAgin
endLoop:
```

מימוש אלגוריתם מס 2

```
;אתחול הערכים
          mov al, 1
          mov bl, 0
doAgain:
          add bl, al
          in al
; BL >= 100 בצע כל עוד
          cmp bl, 100
          jb doAgain
```

שימו לב כי בקטע מס 2 התנאי נבדק בסיום הלולאה, ולכן גוף הלולאה מתבצע לפחות פעם אחת.

שאלה 5.20

א. כתבו תכנית המסכמת את כל המספרים האי-זוגיים החל מ-1 כל עוד הסכום קטן מ-100.

ב. כתבו תכנית המסכמת את איברי הסדרה הבאה: 1, 4, 7, 10, 13. האיבר האחרון אותו נחבר יהיה לא גדול מ-50.

ג. כתבו תכנית הסופרת את מספר האיברים בסדרת פיבונצי עד האיבר האחרון שאינו גדול מ-100.

ביצוע חוזר באורך מחושב מראש

במעבד 8086 ישנן מספר הוראות לביצוע חוזר באורך מחושב מראש אותן נציג בסעיף זה.

ההוראה LOOP

ההוראה הפשוטה ביותר היא הוראת **LOOP** שהמבנה שלה הוא:

LOOP label

בביצוע הוראה זו, המעבד משתמש באוגר CX כמונה המאותחל לערך כלשהו וכל עוד ערכו אינו 0 הוא חוזר על ביצוע גוף הלולאה. אם נניח כי האוגר CX מאותחל לערך כלשהו, נוכל לתאר את הפעולות שהמעבד מבצע באלגוריתם הבא:

$CX \leftarrow CX - 1$

אם $CX \neq 0$ קפוץ לביצוע ההוראה שהתווית label מוצמדת אליה

דוגמה 5.12

לדוגמה, נכתוב קטע התכנית המחבר את 10 המספרים הראשונים בסדרה: 1,2,3,...

```
                  ; אתחול משתנים
    mov  cx, 10   ; CX ← 10  אתחול מונה הלולאה
    mov  bx, 0    ; BX ← 0  אתחול המסכם
doLoop:
    add  bx, cx   ; bx ← bx + cx
                  ; כל עוד ביצוע לולאה לא הסתיים חזור ל- doLoop
  LOOP doLoop
```

כאשר משתמשים בהוראה LOOP צריך קודם כל לאתחל את ערכו של CX למספר הפעמים הרצוי, ולכן לפני ביצוע הלולאה אתחלנו אותו ל-10. שימו לב: תחילה מופחת הערך 1 מהאוגר CX ולאחר מכן ערכו מושווה לאפס.

מאחר שלביצוע ההוראה LOOP המעבד משתמש באוגר CX, כדי לא לשבש את ביצוע הלולאה עלינו להימנע מלשנות את ערכו של אוגר CX בגוף הלולאה. לדוגמה, נוסיף בקטע התכנית המסכם את 10 המספרים החל מ-1, הוראה המשימה את ערכו של BX ב-CX:

```
doLoop:
        add   bx, cx
        mov  cx, bx
  LOOP doLoop
```

במקרה כזה, ערכו של CX יגדל בגוף הלולאה (הוא יכיל סכום של CX + BX) ובהוראת LOOP יקטן ב-1. כדי לעקוב אחר ביצוע התכנית נרשום טבלת מעקב המתארת את מצב CX - BX:

טבלה 5.10
טבלת מעקב אחר ביצוע הלולאה

ערכו של BX (הקסדצימאלי)	ערכו של CX (הקסדצימאלי)	ההוראה
0	0Ah	; אתחול הערכים
0Ah		add bx, cx
	0Ah	mov cx, bx
	9	loop do_loop
13h		add bx, cx
	13h	mov cx, bx
	12h	loop do_loop
25h		add bx, cx
	25h	mov cx, bx
	24h	loop do_loop

מהטבלה ניתן לראות כי ערכו של CX גדל בכל מחזור של הלולאה, ולכן ניתן להניח כי ביצוע הלולאה לא יסתיים. למעשה, מאחר שתחום הערכים ש-BX יכול לקבל הוא מוגבל, כאשר ערכו יהיה גדול מהמספר המקסימלי האפשרי, תהיה גלישה וערכו יכול לקטון שוב.

במעבד 8086 קיימות שתי הוראות לולאה נוספות, בהן סיום הלולאה אינו מותנה רק בכך שערכו של CX הוא אפס, אלא גם במצבו של דגל האפס. הוראות אלה נציג בהמשך.

שאלה 5.21

א. הריצו את התכנית ובדקו האם ביצוע הלולאה מסתיים ואם כן כיצד.

ב. כמה פעמים תתבצע הלולאה הבאה:

```
        mov  cx, 0
        mov  bx, 0
doloop:
        inc  bx
      loop  doloop
```

5.5 הוראות כפל וחילוק

מבנה של הוראות כפל וחילוק בשפת אסמבלי הוא מורכב יותר מאשר חיבור וחיסור משום שבפעולות כפל וחילוק יש צורך בשני אופרנדים המשמשים כאופרנד יעד. לביצוע פעולת חילוק מוקצים שני אופרנדי יעד: אופרנד אחד מכיל את המנה והשני את השארית. באופן דומה, בפעולת כפל אופרנד היעד תמיד כפול מגודל אופרנד המקור וזאת משום שרבים הסיכויים שהתוצאה שתתקבל בביצוע כפל תהיה גדולה מגודל טיפוס אופרנד המקור. לדוגמה: אם נכפיל מספר עשרוני בן 2 ספרות במספר עשרוני בן שתי ספרות, התוצאה יכולה להיות בת 4 ספרות (99×99 = 9801). באופן דומה, בשיטה הבינארית תוצאת ההכפלה של שני מספרים מטיפוס בית, יכולה להיות בגודל מילה (0FFx0FF = 0FE01) ולכן אופרנד היעד בו מאוחסנת התוצאה תמיד כפול בגודלו מגודל טיפוס אופרנד המקור בו מאוחסן הכופל.

במעבד 8086 נקבע כברירת מחדל, כי אופרנדי היעד בפעולות כפל וחילוק יהיו AX (או AH ו-AL) ו-DX (בהתאם לטיפוס האופרנדים שאנו מכפילים), ולכן בהוראות אלו אנו מציינים את אופרנד המקור בלבד, ואילו אופרנד היעד הוא "אופרנד מרומז" או "אופרנד במשתמע" (implicit operand). כמובן שאם השתמשנו קודם לכן באוגרים אלו בהוראות אחרות, תכנם הקודם ימחק ועלינו לדאוג לשמור מידע זה לפני ביצוע הוראות כפל וחילוק.

5.5.1 הוראות כפל

בניגוד לפעולות חיבור וחיסור add, sub הפעולות על מספרים מכוונים ומספרים בלתי מכוונים, בהוראות כפל (וכן בחילוק) ישנה הפרדה, ולכן קיימות שתי הוראות כפל:

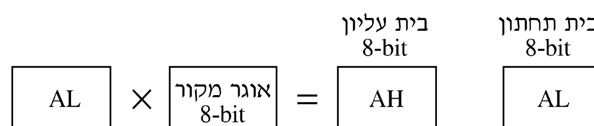
א. כפל של מספרים בלתי מכוונים **MUL** (multiplication):

MUL אופרנד מקור

ב. כפל של מספרים מכוונים **IMUL** (Integer MULtiplication)

IMUL אופרנד מקור

אופרנד המקור מכיל את הכופל והוא יכול להיות מטיפוס בית ובמקרה כזה הנכפל הוא תמיד אוגר AL ויש לאתחלו לערך המתאים לפני ביצוע הוראת הכפל. התוצאה של פעולת הכפל נשמרת באוגר AX (שהוא מטיפוס מילה), כאשר באוגר AL נשמר הבית התחתון של התוצאה ובאוגר AH נשמר הבית העליון של התוצאה, כמתואר באיור 5.3:

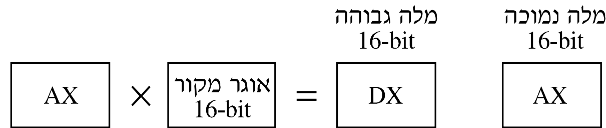


איור 5.3
כפל מספרים מטיפוס בית

לדוגמה:

```
mov  al,  12h          ; al ← 12h
mov  cl,  0A1h         ; cl ← 0A1h
mul   cl               ; ax ← al · cl = 0B52h
```

כאשר מכפילים נתונים מטיפוס מילה, התוצאה המתקבלת היא מטיפוס מילה כפולה והיא נשמרת בשני אוגרים: באוגר DX נשמרת המילה העליונה של התוצאה ובאוגר AX נשמרת המילה התחתונה של התוצאה, כמתואר באיור 5.4:



איור 5.4
כפל מספרים מטיפוס מילה

לדוגמה:

; ax ← 0AFh	mov ax, 0AFh
; cx ← 0BF9h	mov cx, 0FAh
; dx: ax ← ax · cx	mul cx

תוצאת החישוב היא 82F37h כאשר:

; AX ← 2F37h   DX ← 0008h

הוראת IMUL למספרים מכוונים פועלת באופן דומה, כאשר אופרנד המקור מוגבל לתחום מ-128– עד 127+ עבור כפל בתים, ומ-32768– עד 32767+ עבור כפל מילים.

הוראות הכפל משפיעות על הדגלים הבאים:
דגל הגלישה מורם כאשר התוצאה חורגת מהטיפוס המתאים, כלומר אם לא ניתן לאחסן את התוצאה במילה אחת (בכפל בתים) או במילה כפולה (בכפל מילים), הדגלים CF ו-OF ייקבעו לערך 1; אחרת הם יאופסו. כל שאר הדגלים אינם מוגדרים.

בהוראה IMUL הפועלת על מספרים מכוונים הכלל לגבי ערכם של CF ו-OF דומה. במקרה שתוצאת הכפל היא שלילית, הסימן מורחב לאוגר המשמעותי יותר. לדוגמה:

; al ← 12h	mov al, 12h
;cl ← 0A1h	mov cl, 0A1h
; ax ← al · cl = 0F952h	imul cl

5.5.2 הוראות חילוק

באופן דומה להוראות כפל, קיימות שתי הוראות חילוק: חילוק עם מספרים בלתי מכוונים div וחילוק עם מספרים מכוונים idiv.

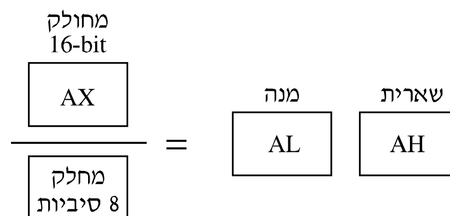
א. חילוק של מספרים בלתי מכוונים – DIV (DIVision)

DIV אופרנד מקור

ב. חילוק של מספרים מכוונים – IDIV (Integer DIVison)

IDIV אופרנד מקור

אופרנד המקור מכיל את המחלק והוא יכול להיות מטיפוס בית ובמקרה כזה המחולק הוא אוגר AX. התוצאה של החישוב נשמרת באוגר AX – מטיפוס מילה, כאשר אוגר AL מכיל את המנה ואוגר AH מכיל את השארית:

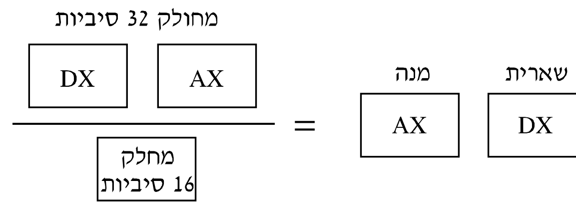


איור 5.5
חילוק מספרים מטיפוס בית

לדוגמה:

```
mov ax, 0FBh        ; ax ← 00FBh ← 251D
mov cl, 0Ch         ; cl ← 0Ch
div   cl            ; ax ← ax/cl ,  al = 20d מנה ,  ah = 11d שארית
```

כאשר אופרנד המקור – המחלק – הוא מטיפוס מילה, המחולק גם כן מטיפוס מילה והוא מאוחסן באוגר AX. התוצאה נשמרת בשני אוגרים: אוגר AX מכיל את המנה ואוגר DX מכיל את השארית:



איור 5.6
חילוק מספרים מטיפוס מילה

לדוגמה:

```
mov  dx,  0                 ; dx ← 0
mov  ax,  141Bh             ; ax ← 5147fd
mov  cx,  012Ch             ; cx ← 300d
div   cx                    ; dx: ax ← ax/cx , ax = 17d מנה , dx = 47d שארית
```

הוראה IDIV פועלת באופן דומה.

בהוראות חילוק אם המנה גולשת מיכולת הייצוג של אוגר היעד (AL או AX) התכנית מופסקת באמצעות פעולה הנקראת "פסיקה", עליה נרחיב בפרק הבא. כל הדגלים אינם מוגדרים בהוראות החילוק.

5.5.3 הוראות להרחבת הסימן

כאשר מחלקים שני מספרים מכוונים עלולה להתעורר בעיה כיוון שההוראה idiv דורשת שחלק מהמספר המוכפל יהיה באוגר AX שגודלו 16 סיביות. במקרה כזה, נשתמש בהוראה CBW הממירה בית למילה.

ההוראה CBW (Convert Byte to Word)

CBW

היא ללא אופרנדים והיא פועלת תמיד על האוגר AL אותו היא מרחיבה למילה שמאוחסנת באוגר AX. כתוצאה מהרחבת המילה, הסימן של הבית - AL מועתק לאוגר AH.

לדוגמה:

```
mov al, 0FBh                ; al ← 0FBh = –5h
cbw                         ; ah = 0FFh,  al = 0FBh
```

וכך התקבל AX = FFFBh. כאשר המספר שמאוחסן באוגר AL הוא חיובי, הסימן יורחב בהתאם לאוגר AH, לדוגמה:

```
mov al, 0FBh        ; al ← 0FBh
cbw                 ; ah = 0h,  al = 07Bh
```

ההוראה CWD (Convert Word to Double Word)

כאשר יש צורך לחלק שני מספרים מכוונים בגודל מילה, נשתמש בפעולה דומה, שמבצעת ההוראה CWD, המרחיבה מילה למילה כפולה. לאחר ביצוע CWD, המילה עם הסימן שבאוגר AX תורחב למילה כפולה מכוונת באוגרים DX:AX. מבנה ההוראה:

CWD

דוגמה 5.13

א. חילוק מספרים מכוונים מטיפוס בית

```
mov al,  0A1H       ; al ← –95D
cbw                 ;ah ← 0FFH
mov cl,  0CH        ; cl ← 12D
idiv  cl            ; al = –7d מנה , alh = –11d שארית
```

ב. חילוק מספרים מכוונים מטיפוס מילה

```
mov ax,  0EBE5      ; ax ←  –5147D
cwd                 ; dx ← 0FFFFH
mov cx,  012CH      ; cx ← 300D
idiv  cx            ; ax = –17d מנה , dx = –47d שארית
```

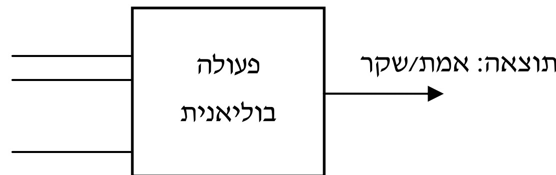
שאלה 5.22

כתבו תכנית המחשבת את העצרת של מספר n המאוחסן בתא זיכרון מטיפוס בית. התכנית בודקת אם החישוב יצר גלישה ואם כן משימה במשתנה a את הערך 1 (אחרת ערכו 0). לדוגמה חישוב של !5:

$$5! = 1\cdot 2\cdot 3\cdot 4\cdot 5$$

5.6 הוראות לוגיות

על מספרים המיוצגים בשיטה בינארית ניתן להפעיל פונקציות (פעולות) לוגיות המבוססות על כללי האלגברה הבוליאנית. אלגברה בוליאנית מגדירה כללי חישוב על משתנים שיכולים לקבל אחד משני הערכים, הנקראים "ערכי אמת", והם: "אמת" ו-"שקר". פונקציה בוליאנית מקבלת כקלט קבוצה של משתנים בוליאניים ומוציאה תמיד פלט אחד ("ערך הפונקציה") שהוא גם כן משתנה בוליאני. אפשר להציג פונקציה לוגית בטבלה הנקראת "טבלת אמת", על שום כך שהיא מציגה את ערך הפונקציה כתוצאה מכל צרוף אפשרי של ערכי הקלט שלה, וכל הערכים הללו הם ערכי אמת (כלומר: "אמת" או "שקר").



איור 5.7
פונקציה לוגית

בניגוד לפעולות אריתמטיות בהן אנו מתייחסים לערך המיוצג בשיטה הבינארית כמספר, בפונקציות (פעולות) לוגיות מספיק, לעיתים, לעבד רק חלק מהסיביות. לדוגמה, כדי לכתוב יישום המטפל בהגרלה בה צריך לנחש 4 מספרים מתוך 16 המספרים מ-0 עד 15, ניתן להשתמש בארבעה משתנים מטיפוס בית שכל אחד מהם מייצג מספר אחד מתוך ארבעת המספרים. אולם דרך יעילה יותר היא להשתמש במשתנה אחד מטיפוס מילה ובה הסיבית שמקומה i מציינת בחירה או אי בחירה של המספר i ($0 \le i \le 15$): כדי לסמן שמספר מסוים נבחר נסמן את הסיבית המתאימה כ-1 ואת שאר הסיביות שמייצגות את המספרים שלא נבחרו נקבע כ-0. לדוגמה איור 5.14 מתאר טופס שבו אדם בחר את המספרים: 5, 8, 10, 14.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0

איור 5.8
ייצוג מספרים שנבחרו בהגרלה במילה

מבחינת זיכרון ייצוג זה הוא חסכוני יותר משום שאנו "אורזים" את כל המידע במילה אחת. כמו כן על-ידי השוואה פשוטה ניתן לבדוק אם אדם ניחש את ארבעת המספרים ואם לאו. שימו לב, בבעיות מסוג זה, אנו מעוניינים במידע על מיקום סיבית, ולערך של המספר עצמו (בדוגמה הוא 2290h) אין משמעות.

בשפת אסמבלי קיימות ארבע הוראות לוגיות: AND, OR, XOR, NOT. לכל ההוראות הלוגיות, מלבד הוראה NOT, יש שני אופרנדים: אופרנד מקור ואופרנד יעד, שיכולים להיות אחד מהצירופים המתוארים בטבלה 5.3.

הפונקציה הלוגית מתבצעת בין כל אחת מהסיביות של אופרנד המקור לבין הסיביות התואמות להן באופרנד היעד. שני האופרנדים בהוראה יכולים להיות ברוחב בית או ברוחב מילה, ובלבד שיהיו בעלי רוחב זהה.

הדגלים המושפעים מהוראה זו הם: SF, ZF, PF.
מאחר שגודל הנתון לא משתנה, הדגלים CF ו-OF נקבעים לערך 0. דגל AF אינו מושפע מביצוע ההוראה.

5.6.1 הפונקציה "וגם" – AND

הפונקציה הבוליאנית "וגם", או באנגלית AND, מקבלת כקלט שתי סיביות x, y והפלט הוא הערך "אמת" אם ערכי שני האופרנדים הם "אמת". לעיתים נהוג לסמן פעולה זו באמצעות הסימן "x" (בדומה לפעולת כפל). נרשום טבלת אמת של פעולת x AND y; נשתמש בערכים 1 ו-0 במקום "אמת" ו-"שקר" בהתאמה, כדי להתאים את הערכים לערכים בהם נשתמש בהמשך בתכנות בשפת אסמבלי:

x AND y	x	y
0	0	0
0	0	1
0	1	0
1	1	1

איור 5.9
טבלת האמת של פעולת **AND**

בשפת אסמבלי ההוראה AND מבצעת את הפונקציה הלוגית AND בין סיביות של שני אופרנדים:

אופרנד מקור, אופרנד יעד AND

ומשמעותה היא:

אופרנד מקור AND אופרנד יעד ← אופרנד יעד

נדגים את פעולת AND בקטע התכנית שלהלן:

```
mov cl, 7Ah
mov bl, 5Bh
and bl, cl
```

כדי לחשב את ערכו הסופי של BL, נציג, זו מול זו, את תבנית הסיביות של האוגרים CL ו-BL, המשתתפים בהוראה AND.

מספר הסיבית:	7	6	5	4	3	2	1	0
תוכן האוגר CL	0	1	1	1	1	0	1	0
תוכן האוגר BL	0	1	0	1	1	0	1	1
התוצאה של ההוראה AND BL, CL	0	1	0	1	1	0	1	0

5.6.2 מיסוך

אחד השימושים החשובים בהוראות לוגיות הנכתבות בשפת אסבמלי הוא ביצוע **מיסוך**, כלומר – התייחסות רק לחלק מהסיביות של האופרנד. לביצוע פעולת המיסוך עלינו להגדיר באופרנד המקור ערך הנקרא "מסיכה" (Mask). ביצוע פעולת מיסוך בעזרת פעולת AND, קובעת חלק מהסיביות של אופרנד היעד לערך 0 ואינה משנה את שאר הסיביות. לדוגמה נבצע פעולת מיסוך על אוגר AL בה נאפס את מחצית הבית העליון של האוגר. כדי לבצע פעולה זו נבחר מסיכה שערכה 0Fh ונרשום את ההוראה:

```
and al, 0Fh
```

האיור הבא מדגים את פעולת המיסוך בין המסיכה ואוגר AL שמכיל ערך כלשהו ולכן כל סיביותיו מסומנות כ-x (כלומר, הן יכולות 0 או 1).

	7	6	5	4	3	2	1	0
מספר סיבית:								
תוכן האוגר AL	x	x	x	x	x	x	x	x
מסיכה	0	0	0	0	1	1	1	1
התוצאה של ההוראה and al, 0Fh	0	0	0	0	x	x	x	x

איור 5.10
מיסוך אוגר AL עם המסיכה 0Fh

שאלה 5.23

בדקו את תוצאת המיסוך באיור 5.5 כאשר ערכו של AL הוא

א. 0AFh

ב. 96h

נדגים את השימוש בפעולת מיסוך עם הוראת AND כדי לפתור את בעיית השידוכים.

דוגמה 5.14

כל לקוח חדש במשרד שידוכים ממלא טופס בקשה בו הוא מסמן את התכונות המבוקשות בבן הזוג מתוך רשימה של 8 תכונות אפשריות. לכל תכונה מסמן הלקוח 1 אם חשוב שלבן הזוג שלו תהיה תכונה זו; אחרת הוא מסמן 0. במשרד בודקים את טופס הבקשה שלו מול הטפסים האישיים שמלאו לקוחות אחרים הרשומים במאגר של המשרד. התאמה מלאה קיימת אם כל התכונות שסימן הלקוח החדש ב-1 וכל התכונות שסימן 0 בטופס הבקשה, קיימות גם בטופס הפרטים האישיים של בן זוג פוטנציאלי. התאמה חלקית מתקיימת אם כל התכונות שסימן הלקוח כ-1 קיימות בבן הזוג, ושאר התכונות שהלקוח סימן ב-0 יכולות להתקיים או לא להתקיים בטופס הפרטים האישיים של בן הזוג הפוטנציאלי. כפלט יש לשים באוגר BL את הערך 1 אם יש התאמה מלאה, 2 אם יש התאמה חלקית ובכל מקרה אחר את הערך 0.

לדוגמה, אם בטופס הבקשה רשם הלקוח שתכונות מס: 1, 4, 6 ו-7 חשובות (מסומנות כ-1), ובטופס בן הזוג הפוטנציאלי שנבדק רשום שהוא בעל התכונות 1, 2, 4, 6 ו-7, אזי יש התאמה חלקית.

פתרון

נייצג טופס בקשה וטופס פרטים אישיים במשתנים מטיפוס בית, כאשר כל סיבית מייצגת תכונה מסוימת. הסיביות שערכן 1 מסמלות לנבדק יש תכונה זו והסיביות שערכן 0 מסמלות שתכונה זו אינה קיימת בנבדק.

כדי לבדוק התאמה בין טופס בקשה וטופס פרטים אישיים יש לבצע את הבדיקות הבאות:

- כדי לבדוק אם יש התאמה מלאה, נשווה את טופס הבקשה של לקוח מול טופס פרטים אישיים של בן זוג פוטנציאלי.
- כדי לבדוק אם יש התאמה חלקית, נמסך בעזרת פעולת AND את טופס הבקשה עם טופס התכונות של בן הזוג הפוטנציאלי . תוצאת המיסוך תאפס את כל הסיביות שלקוח סימן 0 ואילו הסיביות שהלקוח סימן 1 לא ישתנו.

לדוגמה:

טופס לקוח	10010110
טופס בן זוג	11011111
לאחר פעולת AND	10010110

נתבונן בדוגמה נוספת בה אין התאמה חלקית. טופס הפרטים האישיים של בן הזוג מכיל תכונות אחרות מהתכונות שצוינו בטופס הלקוח:

טופס הלקוח – מסיכה	10010110
פרטים אישיים של בן זוג פוטנציאלי	01011101
לאחר פעולת AND	00010100

הסיביות המוקפות במשבצת מדגישות את הסיביות שסומנו כ-1 בטופס הבקשה של הלקוח וב-0 בטופס של בן הזוג הפוטנציאלי. ניתן לראות כי תוצאת פעולת המיסוך יוצרת מספר שונה מהמספר שקיים בטופס הלקוח.

תחילה נגדיר את המשתנים והאוגרים בהם נשתמש ונרשום אלגוריתם מתאים.

המשתנים והאוגרים:

client	יכיל את טופס הבקשה שסימן הלקוח
partner	מכיל טופס פרטים אישיים של בן הזוג הנבדק
BL	בו נאחסן את תוצאת הבדיקה
AL	אוגר עזר

אלגוריתם מתאים:

```
BL ← 0
AL ← client
אם AL = partner אזי
    BL← 1
אחרת
    חשב AL ← AL AND partner
    אם AL = partner אזי
        BL← 2
סיום-אם
```

ערכו של BL מאותחל לאפס שהוא המצב המציין שאין התאמה מלאה ולא התאמה חלקית. ערך זה ישתנה במקרה שנמצא התאמה מלאה או התאמה חלקית.

```
.model small
.stack
.data
    client db 01101010b                  ; תכונות שמסומנות בטופס הבקשה:
    partner db 01111010b                 ; תכונות בן הזוג
.code
start:
                                         ; אתחול מקטע נתונים
    mov ax, DATA
    mov ds, ax
                                         ; אתחול ערכים
    mov bl, 0
    mov al, client
                                         ; בדיקה אם יש התאמה מלאה
    cmp al, partner
                                         ; האם client = partner ?
    jne partial                          ; if client <> partner then jump partial
```

```
; client = partner טיפול במקרה בו
    mov  bl, 1                  ; קיימת התאמה מלאה
    jmp  finish                 ; קפוץ לסוף התכנית
partial:
                                ; בדיקה אם יש התאמה חלקית
    and  al,  partner
    cmp  al,  x
                                ;? client = (partner AND x) האם
    jne  finish                 ; אין התאמה חלקית קפוץ לסוף התכנית
                                ; התאמה חלקית
    mov  bl,  2
finish:
    mov  ax,  4ch               ; סיום התכנית
    int    21h
end start
```

5.6.3 הפונקציה "או" – OR

הפונקציה הבוליאנית "או" (OR) בשתי סיביות מוציאה כפלט את הערך "אמת" אם אחת מהסיביות ערכה "אמת". לעיתים נהוג לסמן פעולה זו באמצעות הסימן "+" . נרשום טבלת אמת של פעולת x OR y של שתי סיביות:

x OR y	x	Y
0	0	0
1	0	1
1	1	0
1	1	1

איור 5.11
טבלת האמת של הפונקציה **OR**

הוראת אסמבלי המיישמת את הפונקציה OR נרשמת במבנה הבא:

אופרנד מקור, אופרנד יעד OR

שמשמעותה:

אופרנד המקור OR אופרנד היעד ← אופרנד היעד

פעולת מיסוך באמצעות ההוראה OR, קובעת את הערך "1" לחלק מהסיביות, ומשאירה את שאר הסיביות ללא שינוי. לדוגמה, נמסך את ערך הסיביות שהאינדקסים שלהם הם 6 ו-7 (שהן הסיביות השביעית והשמינית בבית) של אופרנד היעד ל-1 ושאר הסיביות תשארנה ללא שינוי. לשם כך נבחר במסיכה 11000000 ונרשום את ההוראה:

```
or cl,  11000000b
```

פעולת המיסוך של ההוראה OR מתוארת באיור 5.12, אם נניח כי לפני ביצוע ההוראה ערכו של CL היה 0B2h.

ערך cl	0	1	0	0	1	1	0	1
ערך מיסוך	0	0	0	0	0	0	1	1
cl or 11000000	0	1	0	0	1	1	1	1

איור 5.12
פעולת המיסוך של ההוראה OR

נדגים את השימוש במיסוך עם פעולת OR בבעיה הבאה.

דוגמה 5.15

במשרד הגדירו משימה מסוימת, המורכבת ממספר פעילויות מתוך סדרה של 8 פעילויות סטנדרטיות. את המשימה צריכים לבצע שני אנשים, כך שאת כל אחת מהפעילויות הדרושות יכול לבצע לפחות אחד מהם. לדוגמה משימה מורכבת מפעילויות מס. 1,3,4,7 יכולה להתבצע על-ידי עובד שההכשרה שלו כוללת את הפעולות 1 ו-7 ועובד שני שהכשרתו כוללת את הפעולות 3 ו-4. יש לכתוב קטע תכנית הבודק אם קיימים שני עובדים שיחד יכולים לבצע את המשימה הדרושה.

פתרון

נציג את הפעילויות הדרושות לביצוע המשימה במשתנה operation מטיפוס בית ונסמן את הסיביות שמיקומן 0, 2, 3, 6 ב-1. סיביות אלו מייצגות את הפעילויות 1,3,4,7 (זכרו כי מיקום הסיבית הראשונה בבית היא 0).

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

את ההכשרות של שני העובדים אותם בודקים נייצג במשתנים worker1, worker2.

איור 5.13 מתאר שימוש בפעולת OR כדי להציג הפעולות ששני העובדים יודעים לבצע:

7	6	5	4	3	2	1	0	
0	1	0	0	1	0	0	0	worker1
0	0	0	0	0	1	0	1	worker2
0	1	0	0	1	1	0	1	תוצאת OR

איור 5.13
פונקצית OR לאיחוד הפעולות ששני העובדים יודעים לבצע

כעת ניתן להשוות את תוצאת פעולת ה-OR לפרטי המשימה הדרושה. נרשום קטע התכנית המבצע את הפעולות הדרושות:

```
mov  al, worker1      ; פעולות הדרושות למשימה
or   al, worker2      ; הפעולות המשותפות לשני העובדים
                      ; האם פעילויות המשימה = להכשרות המשותפות לשני העובדים?
cmp  al, operation
```

חשבו, מה קורה אם העובדים יכולים לבצע משימות נוספות? לדוגמה, עובד א' יודע לבצע פעולות: 1,2,3,6,7 ועובד ב' מבצע את הפעולות 3, 4?

אם נבדוק את תוצאת פעולת OR בין הפעולות שעובד א' יודע לבצע עם הפעולות שעובד ב' יודע לבצע נקבל ששניהם יחד יודעים לבצע את הפעולות: 1,2,3,4,6 ו-7.

	7	6	5	4	3	2	1	0
עובד א	1	1	0	0	1	0	1	1
עובד ב	0	0	0	0	0	1	0	1
תוצאת OR	1	1	0	0	1	1	1	1

איור 5.14
דוגמה לאיחוד הפעולות ששני העובדים יודעים לבצע

במקרה כזה ההשוואה לפעולות הדרושות במשימה תכשל. ניתן לפתור את הבעיה אם נוסיף מיסוך עם פעולת AND המאפסת את הפעילויות המיותרות ונרשום את ההוראות הבאות :

```
mov  al,  worker1      ; פעולות הדרושות למשימה
or   al,  worker2      ; הפעולות המשותפות לשני העובדים
and  al,  operation    ; איפוס פעולות מיותרות
                       ; האם פעולות המשימה = לפעולות המשותפות לשני העובדים?
cmp  al,  operation
```

לבדיקה, עקבו אחר ביצוע הפעולות הלוגיות .

שאלה 5.24

לביצוע משימה מסוימת דרושים שני אנשים שתכונותיהם משלימות זו את זו. לכל אדם בודקים 8 תכונות, כאשר כל תכונה שמתקיימת מסמנים ב-1 וכל תכונה שאינה מתקיימת מסמנים ב-0. אם לדוגמה, תכונות של אדם א׳ הן : 11001000 ותכונות של אדם ב׳ הן : 00110111, אזי התכונות של שניהם משלימות זו את זו. הניחו כי חפיפה בתכונות של שני האנשים אפשרית.

רשמו קטע של תכנית הבודק אם התכונות של שני אנשים משלימות זו את זו ואם כן – שם 1 באוגר AL, אחרת ערכו יהיה 0.

5.6.4 הפונקציה הלוגית "או-מוציא" XOR

הפונקציה הלוגית ״או מוציא״ (XOR, קיצור של eXclusive OR) מסומנת על-ידי האופרטור ⊕, פועלת על שתי סיביות ומוציאה כתוצאה 1 רק כאשר ערך אחד משתי הסיביות הוא 1 ; בכל מקרה אחר התוצאה היא 0.

x XOR y	x	y
0	0	0
1	0	1
1	1	0
0	1	1

איור 5.15
טבלת האמת של פינקצית XOR

הפונקציה הלוגית XOR (או-מוציא) מיושמת על-ידי ההוראה

XOR אופרנד יעד, אופרנד מקור

שמשמעותה:

אופרנד יעד ← אופרנד ידע XOR אופרנד מקור

נדגים את ביצוע הפונקציה XOR על הערכים 35h ו-26h:

35h = 00110101
XOR
26h = 00100110
――――――――
התוצאה: 13h = 00010011

אחד השימושים הנפוצים בהוראת XOR הוא לאיפוס ערך של אוגרים. במקום לרשום הוראת העברה:

```
mov ax, 0
```

יעיל יותר לרשום:

```
xor ax, ax
```

ביצוע הוראות לוגיות ובכללן XOR מהיר יותר משום שהרכיבים האלקטרוניים מהם בנוי המחשב הם רכיבים שמבצעים פעולות לוגיות; הפעולות האריתמטיות מבוצעות על-ידי מעגלים שהם עצמם מורכבים מרכיבים לוגיים, ולכן ביצוע פעולה לוגית מהיר יותר מביצוע פעולה אריתמטית או הוראת העברה.

5.6.5 הפונקציה הלוגית "לא" NOT

פונקצית השלילה "לא" הנקראת גם NOT הופכת ערכה של סיבית כמוצג בטבלת האמת הבאה:

X	NOT X
1	0
0	1

איור 5.16
טבלת האמת של פעולת NOT

פונקציה זו ממומשת על-ידי ההוראה NOT בה יש ש אופרנד אחד המשמש מקור ויעד כאחד.

NOT אופרנד

כאשר האופרנד יכול להיות אוגר או תא בזיכרון, אך לא קבוע.

הוראה זו אינה משפיעה על אוגר הדגלים.

לדוגמה, התוצאה של ביצוע ההוראה

not al

כאשר ערכו של AL הוא 01001101 מוצגת באיור הבא:

	0	1	2	3	4	5	6	7
AL	1	0	1	1	0	0	1	0
NOT AL	0	1	0	0	1	1	0	1

איור 5.17
דוגמה לפעולת NOT

שאלה 5.25

כתבו הוראות המציגות מספר בשיטת המשלים ל-2, ללא שימוש בהוראת NEG.

ההוראה TEST

בדומה להוראה CMP שמשווה ערכים של שני משתנים, ההוראה TEST נועדה לעריכת השוואה לוגית בין ערכים של שני משתנים באמצעות הפונקציה AND. בהוראה זו , בניגוד להוראה AND, תוצאת הפעולה אינה נשמרת **ולכן ערכו של אופרנד היעד אינו משתנה**.

אופרנד מקור, אופרנד יעד TEST

שמשמעותה :

אופרנד מקור AND אופרנד יעד

צירופי האופרנדים האפשריים בהוראה זו מוצגים בטבלה 5.3.

הדגלים המושפעים מהוראה זו הם : SF, ZF, PF.

מאחר שגודל הנתון לא משתנה, הדגלים CF ו-OF נקבעים לערך 0. דגל AF אינו מושפע מביצוע ההוראה.

דוגמה 5.16

א. מה יהיה ערכו של דגל האפס לאחר ביצוע סדרת ההוראות שלהלן :

```
mov  al,  00000101b
test  al,  0Fh
```

ב. מהו הערך שיש לרשום באופרנד המקור כדי שביצוע ההוראה TEST על AL יציג רמה לוגית 1 בדגל האפס?

פתרון

התוצאה של ההוראה TEST :

00000101 AND 00001111

היא : 00000101. הערך הזה שונה מאפס ולכן ערכו של דגל האפס הוא 0.

ערך אופרנד המקור יכול להיות : 0F0h או 1Fh. באופן כללי כל ערך שבו ארבע הסיביות התחתונות הן 0 יגרום לדגל האפס לקבל 1.

שאלה 5.26

עקבו אחר ההוראות הבאות, וציינו עבור כל אחת מהן אילו סיביות באוגר AX יושפעו מביצוע פעולת המיסוך ומה יהיה ערכו של האוגר AX לאחר ביצוע כל הוראה:

```
mov cx,  0C123h
or    ax,  08h
and  ax,  0FFDFh
xor  ax,  8000h
or    ax,  0F00h
and ax,  0FFF0h
xor ax,  0F00Fh
xor ax,  0FFFFh
```

שאלה 5.27

ערכו סקר בין צופי הטלוויזיה ובדקו את שעות הצפייה המועדפות שלהם. בכל טופס סימן צופה את המידע הבא:

מין	**1 – נקבה / 0 – זכר**
בין 7:00 ל-10:00	1 – צופה 0 – לא צופה
בין 10:00 ל- 12:00	
בין 12:00 ל- 16:00	
בין 16:00 ל- 20:00	
בין 20:00 ל- 24:00	
בין 00:00 ל- 7:00	

כתבו תכנית המגדירה 4 משתנים שמכילים את המידע של ארבעה צופי טלוויזיה (שתי נשים ושני גברים) והשוו בין הרגלי הצפייה של נשים לגברים. תהליך ההשוואה יתבצע כדלקמן: תחילה מצאו את שעות הצפייה המשותפות לכל הנשים ואחר את שעות הצפייה המשותפות לכל הגברים. לסיום השוו את שעות הצפייה של הנשים והגברים. אם גברים ונשים מעדיפים אותן שעות צפייה השימו במשתנה חמישי את הערך 1 אחרת ערכו יהיה 0.

5.7 הוראות הזזה וסיבוב

קבוצת ההוראות הבאה מטפלת בהזזת סיביות ממקומן, כאשר כל סיבית מועתקת ימינה או שמאלה בהתאם להוראה. להוראות אלו שני שימושים עיקריים: ביצוע פעולות כפל וחילוק ובדיקת הסיביות של הנתון.

פעולות הזזה מאפשרות לבצע פעולות כפל וחילוק בצורה יעילה יותר מאשר בהוראות אריתמטיות. כזכור, מספר בינארי מוצג בשיטת ספירה מיקומית, כאשר מיקום הסיבית קובע את ערכה במספר, ולכן הזזה של סיביות משנה את הערך של המספר.

לדוגמה: המספר: 00001100_2 מוצג בשיטת ספירה מיקומית:

מיקום ספרה במספר	0	1	2	3	4	5	6	7	
ערך המיקום	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	
ספרות המספר	0	0	1	1	0	0	0	0	$= 12_{10}$

איור 5.18
הצגת מספר 00001100_2 בשיטת מיקומית

קיימות יש שתי אפשרויות להזזה של סיביות:
בהזזה שמאלה – כל סיבית מועתקת מיקום אחד שמאלה ובמקום הסיבית הכי פחות משמעותית (מיקומה 0 במספר), נוסיף 0.

מיקום ספרה במספר	0	1	2	3	4	5	6	7	
ערך המיקום	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	
ספרות המספר	0	0	0	1	1	0	0	0	$= 24_{10}$

איור 5.19
הכפלת מספר

במקרה כזה, ערך המספר הוכפל פי 2 (בדומה לכפל מספר עשרוני ב-10).

בהזזה ימינה – כל סיבית מועתקת מיקום אחד ימינה, כאשר במקום הסיבית המשמעותית ביותר (הסיבית שמיקומה 7) נוסיף 0.

מיקום ספרה במספר	0	1	2	3	4	5	6	7	
ערך המיקום	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	
ספרות המספר	0	1	1	0	0	0	0	0	6_{10} =

איור 5.20
חלוקת מספר

במקרה כזה, בצענו חלוקה וערך המספר קטן פי 2 (חלוקה בשתיים) (בדומה לחלוקת מספר עשרוני ב-10). לדוגמה,

$$1011000 : 100 = 10110\cancel{00} = 10110$$

במקרה כזה המספר 1011000 הוזז שתי מקומות ימינה והתוצאה היא 10110. אולם אם נחלק 10110 ב-100 נקבל תוצאה שגויה (פעולה אינה משמרת מידע) משום שגם כן מוזזות שתי סיביות ימינה והן "הולכות לאיבוד"

$$10110 : 100 = 101\cancel{10} = 101$$

בנוסף, הוראות אלו שימושיות כאשר יש להתייחס לסיביות בודדות שהן חלק מבית או מילה. לדוגמה, במשרד שידוכים ממלאים לקוחות טפסים אודות בן זוג מבוקש. כדי לדעת כמה תכונות סימן לקוח בטופס התכונות הרצויות של בן הזוג, אנו יכולים להזיז את סיביות המשתנה שמייצג את הטופס שסימן הלקוח ולספור את מספר הסיביות שערכן הוא 1. בהמשך נציג בעיות מסוג זה.

הוראות הזזה וסיבוב יכולות לפעול על אוגר או תא בזיכרון בגודל בית או מילה. ההבדל בין הזזה לסיבוב הוא, שבהוראות הזזה הסיביות המוזזות "נופלות" בקצה האחד של האוגר, בעוד שבהוראות הסיבוב, נכנסות הסיביות בחזרה, בצידו האחר של האוגר. לדוגמה,

נתון	10110011
הזזה 2 מקומות ימינה	00101100
סיבוב 2 מקומות ימינה	11101100

במעבד 8086 לכל הוראה בקבוצה זו 2 צורות:

א. הזזה/סיבוב פעם אחת

1, אופרנד יעד פעולה (הזזה/סיבוב)

ב. הזזה/סיבוב n פעמים, כאשר הערך n מאוחסן באוגר CL

CL, אופרנד יעד פעולה (הזזה/סיבוב)

כמו כן ההוראות בקבוצה זו משפיעות על חלק מדגלי המצב. נתאר זאת בהמשך.

5.7.1. הוראות הזזה

בקבוצה זו יש שתי תת-קבוצות: קבוצה אחת מטפלת בהכפלה וחילוק ב-2 של מספרים בינאריים בלתי מכוונים והקבוצה השנייה מטפלת בהכפלה וחילוק ב-2 של מספרים בינאריים מכוונים. אנו נתייחס תחילה להוראות הזזה שמאלה ואחר כך להוראות הזזה ימינה.

הזזה שמאלה – SHL, SAL

קבוצה זו כוללת שתי הוראות:

- הוראת ההזזה האריתמטית שמאלה – **SAL** (Shift Arithmetic Left) מטפלת במספרים מכוונים.
- ההוראה להזזה לוגית שמאלה – **SHL** (Shift Left) מטפלת במספרים בלתי מכוונים.

כאשר לכל הוראה יש שתי צורות:

1, אופרנד יעד SHL

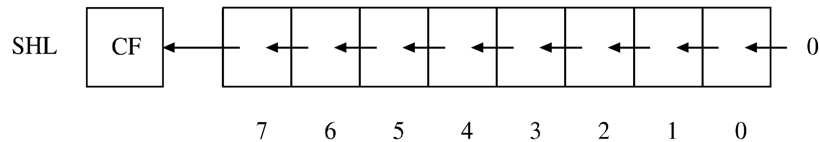
שמשמעותה: $2 \cdot$ אופרנד היעד ← אופרנד היעד

CL, אופרנד יעד SHL

שמשמעותה: $2^{CL} \cdot$ אופרנד היעד ← אופרנד היעד

כאשר אופרנד היעד – הוא אוגר או תא בזיכרון.

הוראה SHL מבצעת הכפלה ב-2 של מספר בלתי מכוון, על-ידי הזזת כל סיביות המספר מקום אחד שמאלה, והוספת 0 במקום הימני ביותר. הסיבית המשמעותית ביותר "נופלת" ומועברת לדגל הנשא. איור 5.21 מתאר הזזה לוגית שמאלה של בית.

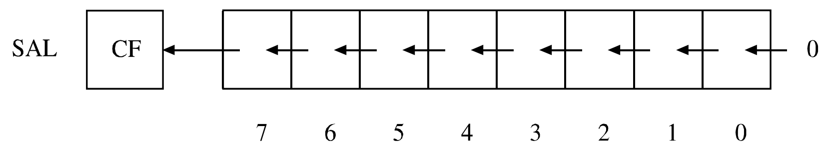


איור 5.21
הזזה לוגית שמאלה של בית

הזזה אריתמטית שמאלה מבצעת את אותה הפעולה וגם כאן יש שתי הוראות:

SAL אופרנד יעד, 1
SAL אופרנד יעד, CL

וביצוע הוראה זו זהה לביצוע הזזה לוגית שמאלה:



איור 5.22
הזזה אריתמטית שמאלה של בית

כלומר, כפל מספר מכוון ב-2 נותן תוצאה כמו כפל מספר בלתי מכוון ב-2. ואכן בעת תרגום התכנית נראה כי האסמבלר מתרגם את הוראת SAL להוראה SHL. מדוע, אם כן, יצרו שתי הוראות שתפקידן זהה? מאחר שיש הבדל בין הוראות ההזזה ימינה SAL ו-SHL, רצו ליצור גם הוראות תואמות להזזה שמאלה, כדי לשפר את הקריאות של התכנית. משתמשים בהוראה המתאימה לפי אופי הנתונים, הן להזזה ימינה, שם ההוראות מבצעות דברים שונים, והן להזזה שמאלה, שם שתי ההוראות מבצעות אותו הדבר.

להוראות אלו יש השפעה על חלק מדגלי המצב ובעיקר על CF , OF, SF, ZF

לדוגמה:

```
mov al, 28h        ; al ← 00101000
shl  al, 1         ; al ← 01010000
sal  al, 1         ; al ← 10100000
```

טבלה 5.11

הסבר למצב הדגלים	ZF	SF	OF	CF	הערה	ההוראה
	?	?	?	?	; AL = 00101000	mov al, 28h
	0	0	0	0	; AL = 01010000	shl al, 1
דגל הסימן השתנה (מ-0 ל-1) מכאן ניתן להסיק כי בפעולה עם מספר מסומן, הייתה גלישה ולכן OF הונף.	0	1	1	0	; AL = 10100000	sal al, 1
דגל הסימן השתנה (מ-1 ל-0) מה שמלמד שעבור מספר מכוון הייתה גלישה ולכן OF הונף. כמו כן הסיבית השמאלית היתה 1 והיא גלשה; לכן היה נשא עבור מספר בלתי מכוון ולכן דגל CF הונף.	0	0	1	1	; AL = 01000000	sal al, 1

אנו יכולים לבצע אותו הדבר תוך שימוש בהוראת הזזה מרובה, לדוגמה:

```
mov  al, 28h
mov  cl, 3
shl   al, cl
```

אלא שהפעם, דגל הנשא יונף רק אם הסיבית האחרונה שגלשה היא 1, ודגל הגלישה OF לא מושפע.

לסיכום, הדגלים המושפעים בהוראות הזזה:

CF – נקבע לערך הסיבית השמאלית ביותר שנשמטה

OF יהיה 1 ויציין גלישה אם התוצאה שינתה את הסימן בהזזה אחרונה בלבד

הדגלים SF, PF, ZF משתנים בהתאם לתוצאה

הדגל AF לא מוגדר

ראינו הכפלה בחזקות של 2, אך ניתן גם לבצע הכפלה ב-N שאינו חזקה של 2. לדוגמה, כדי להכפיל את אוגר AX ב-10, נוכל לרשום את הפעולה:

$$AX \cdot 10 = AX \cdot 8 + AX \cdot 2 = AX \cdot (2^3) + AX \cdot (2^1)$$

ובהתאם, נרשום את ההוראות הבאות:

```
shl  ax, 1        ; AX ← AX · 2
mov bx, ax        ; BX ← 2AX
shl  ax, 2        ; AX ← 2AX · 4 = 8 · AX
add  ax, bx       ; AX← AX + BX = 8AX + 2AX = 10AX
```

נציין כי ניתן כמובן להשתמש בהוראה MUL לביצוע כפל זה. אך כפי שנראה בפרק הבא, מספר מחזורי הוראה לביצוע הוראות לוגיות והוראות הזזה קטן ולכן למרות שכתבנו מספר הוראות זמן ביצוע התכנית יהיה קצר יותר לעומת תכנית הכוללת הוראת MUL אחת משום שמספר מחזורי ההוראה לביצוע MUL גדול מאוד.

שאלה 5.29

השתמשו בהוראות הזזה ורשמו הוראות בשפת אסמבלי להכפלת תוכן האוגר AX ב-18.

5.7.2 הוראות הזזה ימינה SHR ו-SAR

קבוצה זו כוללת שתי הוראות:

- הוראת הזזה אריתמטית ימינה – **SAR** (Shift ArithmeticRight), המטפלת במספרים מכוונים.
- ההוראה להזזה לוגית ימינה – **SHR** (Shift Right), המטפלת במספרים בלתי מכוונים.

כאשר לכל הוראה יש שתי צורות:

SHR אופרנד יעד, 1

SAR אופרנד יעד, 1

שמשמעותה: אופרנד יעד ← אופרנד יעד / 2

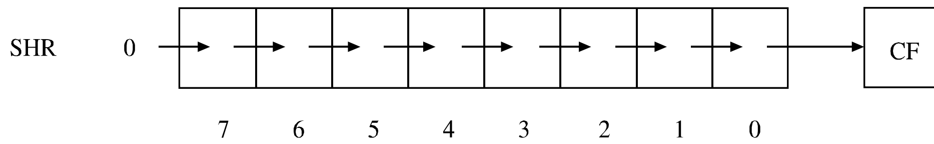
SHR אופרנד יעד, CL

SAR אופרנד יעד, CL

שמשמעותה: אופרנד יעד ← אופרנד יעד / 2^{CL}

כאשר אופרנד היעד הוא אוגר או תא בזיכרון.

ההוראה SHR מזיזה את הסיביות ימינה ומשלימה את הסיבית משמאל (המשמעותית ביותר) באפס. הסיבית האחרונה שהוצאה מוכנסת לדגל CF:

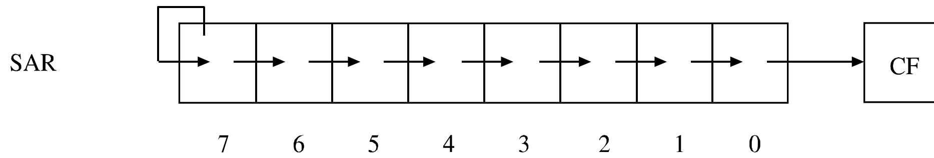


איור 5.23
הזזה לוגית ימינה של בית

לדוגמה:

```
mov al, 01100101        ; al ← 01100101
mov cl, 2
shr   al,  cl           ; al ← 00011011  התוצאה היא:
```

ההוראה SAR מזיזה את הסיביות ימינה ומשלימה את הסיבית משמאל (המשמעותית ביותר) בערך של סיבית הסימן לפני ההזזה. הסיבית האחרונה שהוצאה מוכנסת לדגל CF:



איור 5.24
הזזה אריתמטית ימינה של בית

לפקודה זו יש משמעות כאשר מתייחסים לערך של אופרנד היעד כאל מספר מכוון. במקרה כזה הוראה זו מבצעת חילוק ב-2N

לדוגמה:

```
mov al, 11100101          ; al ← 11100101
mov cl, 2
sar   al, cl              ; al ← 11111001  התוצאה היא:
```

הדגלים המושפעים מהוראות הזזה ימינה הם:

CF – מכיל את הסיבית האחרונה שגלשה
OF – הוא 1 אם משתנה סימן האופרנד
SF, ZF משקפים אם התוצאה היא אפס או שלילית
AF אינו מושפע

שאלה 5.30

א. כתבו קטע של תכנית הבודק אם מספר נתון הוא זוגי.
ב. הסבירו מה קטע התכנית הבא מבצע:

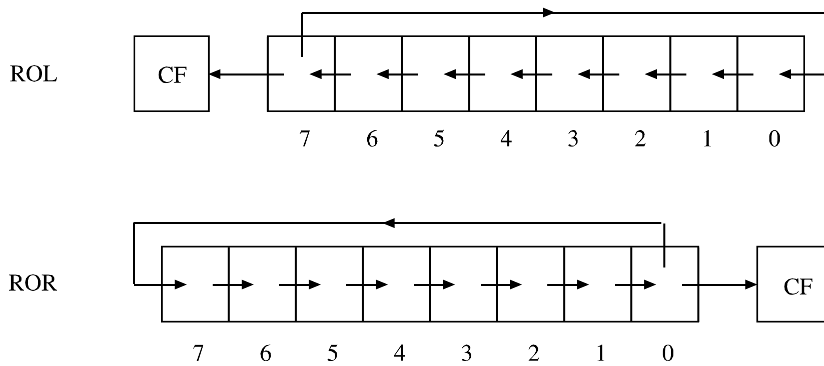
```
mov ah,  al
shl   al,   4
shr   ah,  4
or     al,  ah
```

5.7.3 הוראות סיבוב

בהוראות סיבוב, הסיבית שגולשת מוחזרת מהצד השני של האופרנד, כאשר יש שתי אפשרויות:

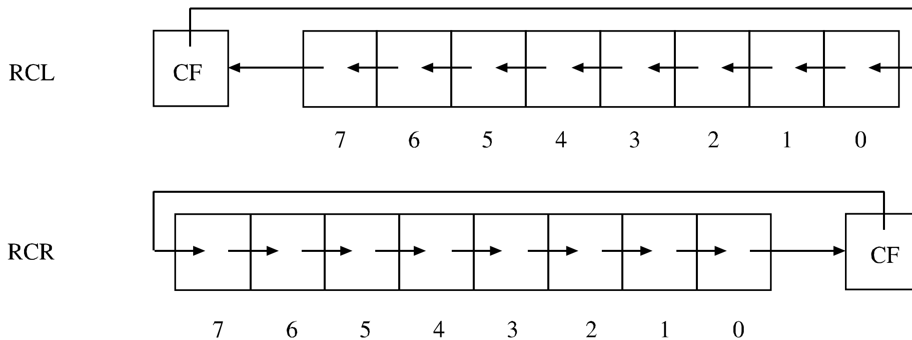
- הזזה מעגלית ימינה ושמאלה
- הזזה מעגלית ימינה ושמאלה כאשר הסיבית בדגל הנשא CF היא חלק מהאופרנד.

א. הזזה ללא שימוש בדגל הנשא – ROL (Rotate Left) ו-ROR (Rotate Right)



איור 5.25
הזזה מעגלית ללא נשא

ב. **הזזה עם שימוש בדגל הנשא RCL** (Rotate Left through Carry) ו-**RCR** (Rotate Right through Carry) – הזזה מעגלית ימינה ושמאלה כאשר הסיבית בדגל הנשא CF היא חלק מהאופרנד:



איור 5.26
הזזה מעגלית עם נשא

הדגלים המושפעים:

רק הדגלים CP ו-OF מתעדכנים.

בהוראה בה אנו מסובבים פעם אחת, דגל OF יונף אם משתנה סימן האופרנד (בהוראה בה מספר הסיבובים גדול מ-1, דגל זה לא מושפע).

דוגמה 5.17

בסקר צפייה בטלוויזיה התבקש כל משתתף לסמן את כל התכניות בהן הוא צופה מבין 10 תכניות נתונות. את תשובות הצופה שומרים במשתנה response בו הסיביות 0 עד 9 מייצגות את התכניות וכל סיבית שהיא 1 במיקום I מציינת שהמשתתף צופה בתוכנית I. כתבו תכנית המונה את מספר התוכניות בהן צופה המשתתף.

כדי לפתור בעיה זו נסובב ימינה את הערך שבמשתנה response ונשתמש בביט שהועבר ל-CF כדי לקדם את מונה התכניות; נחזור על פעולה זו 10 פעמים.

תחילה נרשום אלגוריתם:

מונה הלולאה CL←0Ah
מונה מספר התכניות שנצפו BL ← 0
AX ← response
בצע
סובב ימינה את AX
אם CF ← 1 אזי
BL ← BL + 1
CL ← CL – 1
עד ש- CL ← 0

התכנית

```
.MODEL SMALL
.STACK 100h
.DATA
    response DW  02A3h
.CODE
```

```
start:
; אתחול סגמנט נתונים
    MOV  AX,  @DATA
    MOV  DS,  AX
; אתחול משתנים
    MOV  CL, 0Ah            ; מונה לולאה CF ← 0Ah
    MOV  BL, 0              ; מונה מספר התכניות שנצפו BL ← 0
    MOV  AX, response       ; טופס של צופה AX ← 02A3h
; repeat
again:   ROL  AX, 1         ; סובב ימינה את AX
    JNC  next               ; אם CF ← 0 דלג ל-next
                            ; תכנית נצפית על-ידי הצופה CF←1
    INC  BL                 ; BL ← BL + 1
                            ; until CL ← 0
next:
    DEC  CL                 ; עדכון מונה לולאה CL ← CL – 1
    JNZ    again            ; אם CL > 0 חזור

; סיום התכנית
    MOV  AX,  4ch
    INT    21h
END start
```

שאלה 5.30

ניתן לשפר את התכנית המוצגת בדוגמה 5.17 אם נשנה את הוראת הסיבוב ונשתמש בהוראת הזזה ובהתאם נשנה את תנאי היציאה מהלולאה. להלן קטע קוד לאחר ביצוע השינויים.

א. רשמו מהו התנאי לסיום הלולאה והסבירו את הקשר של התנאי להוראת SHL בה השתמשנו (במקום ROR).

ב. תנו דוגמה לערך של response עבורו תכנית זו יעילה יותר, כלומר הלולאה תתבצע מספר קטן של פעמים מאשר בתכנית הקודמת.

ג. תנו דוגמה לערך של response עבורו השינוי בתכנית לא שיפר את היעילות, כלומר מספר הפעמים שלולאה תתבצע יהיה זהה בשני המקרים.

```
start:
    mov  ax,  @DATA
    mov  ds,  ax
    mov  bl,   0
    mov  ax,   x
again:  shl  ax,  1
    jnc  next
    inc  BL
next:
    cmp  ax,  0
    jne  again

    mov  ax,  4ch                                        ; סיום התכנית
    int    21h
END  start
```

שאלה 5.31

יש להרחיב את התכנית הבודקת את טופס הצפייה response של צופה טלוויזיה כך שתבדוק גם את התנאים הבאים:

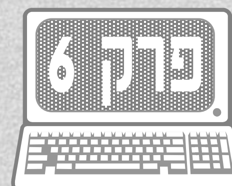
א. האם הוא צופה בתכנית 2 ו-5

ב. האם הוא צופה בתכנית 2 או 5

ג. האם הוא צופה ב-3 תכניות בדיוק

ד. האם הוא צופה בין 2 ל-4 תכניות מתוך ה-10

את תוצאת כל תנאי שמרו במשתנה זיכרון מתאים.



שיטות מיעון, מערכים ורשומות

6.1 הצהרה על מערכים ורשומות

התכניות שהצגנו עד כה התייחסו לעיבוד של מספר משתנים קטן, אותם אחסנו באוגרים או בזיכרון. אך כל שפת אסמבלי מספקת שיטות שמאפשרות גם עיבוד של מבנים המורכבים מבלוק של תאים בזיכרון כגון מערכים ורשומות. כדי לעבד מבנים אלו נתאר תחילה כיצד מייצגים מערכים ורשומות בשפת אסמבלי ואחר נתאר שיטות שונות לגישה לנתונים אלו .

א. הגדרת מערך חד-ממדי

איברי מערך חד-ממדי נשמרים במקטע הנתונים ברצף של תאים. כדי להצהיר על מערך חד-ממדי רושמים שם מערך (שם משתנה), אחריו את טיפוס האיבר במערך ואחריו את הערכים שיושמו במערך לאחר אתחולו. לדוגמה:

W D a 100 ,7 ,0 ,3 ,12   אתחול מערך שיש בו 5 איברים מטיפוס מילה

הנחיית אסמבלר זו שקולה להצהרה על מערך ששמו a ואתחול מערך זה, שמצייני איבריו הם 0 עד 4 במתואר באיור 6.1 א' (שימו לב: בשפת אסמבלי, המציין הראשון הוא 0!). איברים אלו נשמרים במקטע הזיכרון ברצף של תאים שעל הראשון מצביע שם המשתנה a במתואר באיור 6.1 ב'.

מציין איבר	4	3	2	1	0
תוכן האיבר	12	3	0	7	100

איור 6.1 א'
מערך חד-ממדי (תוכן האיברים מצויין בשיטה העשרונית)

	תוכן	היסט
a →	64h	0000
	7h	0001
	0h	0002
	3h	0003
	0Ch	0004
		0005
		0006

איור 6.1ב'
אחסון מערך חד-ממדי בזיכרון (תוכן האיברים מצויין במספרים הקסדצימליים)

אפשר להצהיר על מערך חד-ממדי ולציין את מספר האיברים שיהיו בו; לשם כך משתמשים בהנחיית האסמבלר DUP. לדוגמה:

```
a   DB    8 DUP (?)
```

בהצהרה זו הגדרנו מערך חד-ממדי ששמו a ובו 8 איברים (שאינם מאותחלים לערך כלשהו) שמצייניהם הם מ-0 עד 7. ובאופן דומה, ההצהרה הזו:

```
a   DW    100 DUP (10)
```

מגדירה מערך בן 100 איברים מטיפוס מילה, המאותחלים כולם לערך 10 (בשיטה העשרונית) ומצייני איבריהם הם מ-0 עד 99.

כדי לפנות לאיבר ה-i במערך חד-ממדי יש לחשב את ההיסט של האיבר מתחילת המערך ככפל של מיקום האיבר בגודל טיפוס האיבר. לדוגמה:

א. ההיסט של האיבר השלישי, כלומר i=2, מתחילת המערך מטיפוס בית הוא: 2×1.

ב. ההיסט של האיבר השלישי, כלומר i=2, מתחילת המערך מטיפוס מילה הוא: 2×2.

שאלה 6.1

רשמו הצהרות מתאימות למערכים הבאים:

א. מערך חד-ממדי בו 100 איברים מטיפוס בית שכל איבריו מאותחלים ל-0.

ב. מערך חד-ממדי בן 25 איברים מטיפוס מילה כפולה.

ג. האם לשני המערכים שהגדרתם בסעיפים א' ו-ב' מוקצה אותו שטח זיכרון? נמקו את תשובתכם.

ד. הגדירו קבוע SIZE והשתמשו בו להגדרת מערך חד-ממדי בן 10 איברים.

ה. חשבו את ההיסט של האיבר ה-7 במעד חד-ממדי שהגדרתם בסעיף א' ובמערך הדו-ממדי שהגדרתם בסעיף ב.

ב. הגדרת מערך דו-ממדי

איברי מערך דו-ממדי מאוחסנים במקטע הנתונים ברצף של תאים בדומה למערך חד-ממדי. ראו איור 6.2.

	0	1	2	3	4
0					
1					
2					
3					

	0
	1
	2
	3
	4
	5
	6
	7
	8
	9
	10
	11
	12
	13
	14
	15
	16
	17
	18
	19

איור 6.2
מיפוי מערך דו-ממדי לזיכרון ליניארי

קיימות שתי אפשרויות למפות מערך דו-ממדי כמערך חד ממדי:

- מיפוי לפי שורות: 4 התאים הראשונים הם איברי השורה הראשונה במערך, 4 התאים הבאים הם איברי השורה השנייה וכך הלאה
- מיפוי לפי עמודות: 4 תאים ראשונים הם איברי העמודה הראשונה במערך, 4 התאים הבאים הם איברי העמודה השנייה וכך הלאה

מתכנת יכול לבחור באחת משתי האפשרויות אלה כדי למפות מערך דו-ממדי ולאחסן אותו. בספר זה נשתמש במיפוי שורה, שהיא שיטת המיפוי המקובלת בשפות עיליות, כמו פסקל ושפת C. האיור הבא מתאר מיפוי לפי שורות של מערך דו-ממדי בזיכרון לינארי:

	0	1	2	3	4
0	2	7	3	4	32
1	5	9	0	1	15
2	34	6	28	19	23
3	22	35	78	12	32

איור 6.3 א
מערך דו-ממדי

שורה 1					שורה 2					שורה 3						שורה 4			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	7	3	4	32	5	9	0	1	15	34	6	28	19	23	22	35	78	12	32

איור 6.3 ב
מיפוי של מערך דו-ממדי לזיכרון לינארי

למערך דו-ממדי, כמו למערך חד-ממדי, מוקצה בלוק של תאי זיכרון בהתאם לטיפוס האיברים. גודל הזיכרון (מספר הבתים) הדרוש למערך שיש בו n שורות ו-m עמודות, כאשר כל איבר הוא מטיפוס שמספר הסיביות בו הוא p בתים, מחושב על-ידי $n \cdot m \cdot p$ בתים. לדוגמה: למערך דו-ממדי בן 5 שורות ו-4 עמודות, מטיפוס בית, יוקצו $4 \cdot 5 \cdot 1 = 20$ בתים. אבל למערך מטיפוס מילה, באותו גודל, נצטרך להקצות $5 \cdot 4 \cdot 2 = 40$ בתים בזיכרון.

לדוגמה ההצהרה על מערך דו-ממדי מטיפוס בית, שיש בו 4 שורות ו-5 עמודות, נרשמת כך:

```
a   DB    4·5 DUP(?)
```

הצהרה זו שקולה להצהרות הבאות:

```
a   DB    4 dup (5 DUP (?) )
a   DB    20 DUP(?)
```

כדי לאתחל את אברי המערך נרשום את ההצהרה הזו:

```
a    DB   2, 7, 3, 4, 32
```

```
DB 5, 9, 0, 1, 15
DB 34, 6, 28, 19, 23
DB 22, 35, 78, 12, 32
```

דרך אחרת היא לרשום את כל הערכים באותה שורה:

```
a DB 2,7,3,4,32,5,9,0,1,15,34,6,28,19,23,22,35,78,12,32
```

בשני המקרים יוקצו 20 תאים רצופים ובכל אחד מהם יואחסן ערך אחד.

שימו לב, מבחינת המעבד אין הבדל בין הקצאה של תאים למערך חד-ממדי לעומת מערך דו-ממדי.

כדי לִפְנות לאיבר מסוים במערך הדו-ממדי, עלינו לחשב את ההיסט של האיבר מתחילת המערך, בהתאם למיפוי שורה. באופן כללי, מיקומו של איבר [i,j] יחושב בצורה הזו:

היסט של איבר במערך דו-ממדי = (i · COLUMNS + j) · ELEMENT_SIZE

כאשר

COLUMNS מציין את מספר העמודות במערך;

ELEMENT_SIZE מציין את גודלו של האיבר.

שימו לב, מצייני שורה ראשונה ועמודה ראשונה הם 0 ולא 1.

לדוגמה, כדי לחשב את ההיסט של האיבר שנמצא בשורה הרביעית ובעמודה השלישית, כלומר את האיבר a[3,2], במערך שיש בו 4 שורות ו-5 עמודות, ואיבריו הם מטיפוס בית, נציב את הערכים:

i=3,

j=2,

מספר העמודות COLUMNS = 5

וגודל האיבר ELEMENT_SIZE = 1

כעת נחשב את ההיסט באמצעות הנוסחה שרשמנו קודם לכן:

$$(3 \cdot 5 + 2) \cdot 1 = 3 \cdot 5 + 2 = 17$$

ואכן אם נתבונן באיור 6.3ב', נראה כי ההיסט של האיבר [3,2] מתחילת המערך הוא 17 תאים.

אם המערך הוא מטיפוס מילה יש להכפיל את החישוב הקודם ב-2:

$$(3 \cdot 5 + 2) \cdot 2 = 17 \cdot 2 = 34$$

שאלה 6.2

א. הגדירו מערך דו-ממדי בשם a, בגודל 5 שורות ו-10 עמודות, שאיבריו יהיו מטיפוס מילה.

ב. חשבו את ההיסט של האיברים האלה מתחילת המערך:

- a[3, 9]
- a[5,10]
- a[0,0]

ג. ייצוג רשומה בשפת אסמבלי

רשומה (record) היא קבוצה של פריטים, הנקראים שדות; לכל שדה יש שם מזהה והוא יכול להיות מטיפוס שונה. בהמשך סעיף זה נציג כיצד מגדירים ומעבדים רשומות בשפת אסמבלי.

כדי להקצות מקום בזיכרון לרשומה, עלינו להגדיר תחילה את המבנה של הרשומה ולאחר מכן לאתחל את ערכי השדות בערכים.

א. **ההצהרה של רשומה** נרשמת לפני מקטע הנתונים. מבנה ההצהרה הוא:

```
<שם רשומה> STRUC
            פירוט השדות
<שם רשומה> ENDS
```

לדוגמה, נגדיר מבנה של רשומה בשם item. רשומה זו מכילה את השדות האלה:

- מספר הפריט — שדה מטיפוס מילה;
- כמות במלאי — שדה מטיפוס בית;
- מספר ספק המספק את הפריט — שדה מטיפוס מילה.

הצהרה על רשומה item:

```
item STRUC
   itemnum    DW ?
   quantity   DB ?
   supplier   DW ?
item ENDS
```

ב. כעת נאתחל במקטע הנתונים שתי רשומות של פריט:

```
.DATA
p1  item < 1111, 100, 923 >    ; מספר הפריט 1111; הכמות במלאי היא 100 והוא מסופק
                               ; על-ידי ספק 923
p2  item < 2222, 50, 120 >     ; מספר הפריט 2222, הכמות במלאי היא 50 והוא מסופק
                               ; על-ידי ספק 120
```

שאלה 6.3

א. הגדירו את רשומה בשם: book שמכילה את השדות הבאים:

מספר ספר בן 5 ספרות;

שפה בה כתוב הספר מטיפוס תו, כאשר: H מסמל עברית, E מסמל אנגלית, A מסמל ערבית;

מספר עמודים בן 3 ספרות;

ב. אתחלו 3 רשומות בנתוני ספר.

ג. חשבו את ההיסט של השדה הראשון וההיסט של השדה השלישי מתחילת הרשומה.

6.2 שיטות מיעון

הוראה בשפת אסמבלי מגדירה קבוצה של הוראות בשפת מכונה, שבהן הפעולה שהמחשב מבצע היא זהה אך היא מתבצעת על סוגים שונים של אופרנדים. האופרנדים מציינים את אופן הגישה לנתונים עליהם מתבצעת הפעולה והם יכולים להיות מאוחסנים בהוראה עצמה, באוגר או בזיכרון. קיימות שיטות שונות לציון האופרנדים; שיטות אלה נקראות **שיטות מיעון** (Addressing modes), כאשר מקור המילה "מיעון" הוא במילה "מען", דהיינו: "כתובת". למעשה בתכניות שהצגנו בפרק הקודם השתמשנו בשלוש שיטות מיעון שונות:

מיעון מיידי שבו אופרנד המקור הוא נתון, לדוגמה:

```
mov al, 10h
```

מיעון אוגר שבו האופרנדים הם אוגרים, לדוגמה:

```
mov al, ah
```

מיעון ישיר שבו אחד מהאופרנדים הוא משתנה בזיכרון, לדוגמה:

mov al, var1

בחירת שיטת המיעון משפיעה לא רק על אופן הגישה לנתון, אלא גם על אורך הוראת המכונה ועל משך הזמן הדרוש לביצועה. בסעיף זה נציג בצורה מופשטת את המכניזם שגורם לכך.

אורך הוראת המכונה נקבע על-ידי האסמבלר, בתהליך התרגום של כל הוראה בשפת אסמבלי להוראה בשפת מכונה. לדוגמה: ההוראה mov al, ah מתורגמת להוראת המכונה 8AC4h, ואילו ההוראה mov al, x, כאשר ההיסט של משתנה x הוא 0, מתורגמת להוראת מכונה A00000h. במעבד 8086 האורך של הוראת מכונה נע בין בית אחד לשישה בתים והוא תלוי בסוג ההוראה ובשיטת המיעון שבה אנו משתמשים. בדרך-כלל, אם ההוראות כוללות אופרנדים המציינים גישה לזיכרון, הן ארוכות יותר מאשר הוראות שמבצעות את אותה פעולה ובהן האופרנדים הם אוגרים.

לדוגמה: אורך ההוראה mov ax, 1000h הוא שלושה בתים, ואילו אורך ההוראה mov x, 1000h, כאשר x הוא משתנה מטיפוס מילה, הוא 6 בתים.

זמן ביצוע ההוראה מושפע מגורמים רבים, ביניהם אורך ההוראה וסוג הפעולה. כדי להבין עיקרון זה נחזור ונתאר את המחזור **הבאה-וביצוע** של הוראה, הכולל את הפעולות האלה:

1. קריאת ההוראה לביצוע וקידום מצביע ההוראות (IP);
2. פענוח ההוראה;
3. קריאת אופרנדים המאוחסנים בזיכרון (במידת הצורך);
4. ביצוע ההוראה;
5. אחסון התוצאה.

הפעולות 1-3 מתארות את תהליך ההבאה, והפעולות 4-5 מתארות את תהליך הביצוע.

זמן הקריאה של ההוראה (פעולה 1) תלוי באורכה, כי בכל גישה לזיכרון, מעבד 8086 יכול לקרוא רק מילה אחת. כאשר אורך ההוראה הוא 2 בתים, תידרש גישה אחת לזיכרון (כפי שתיארנו בפרק הראשון), וכאשר אורך ההוראה הוא 6 בתים, יידרשו שלוש גישות כדי לקרוא את ההוראה מהזיכרון אל המעבד. בגלל מגבלות חומרה, זמן התגובה של יחידת

הזיכרון ויחידות הקלט/פלט הוא איטי ביחס לזמן הפעולה של המעבד עצמו, לכן זמן ביצוע ההוראה מושפע במידה ניכרת ממספר הגישות לזיכרון (או להתקני הקלט/פלט). פעולות בהן המעבד צריך לקרוא או לכתוב נתון ליחידות זיכרון וקלט/פלט הן איטיות יותר מפעולות המתבצעות בין הרכיבים הפנימיים של המעבד (כמו העתקת נתון מאוגר לאוגר, חיבור נתון לאוגר וכדומה).

אחרי שמתבצעת קריאת ההוראה, מתעדכן מצביע ההוראות (האוגר IP) ומצביע על כתובת ההוראה הבאה לביצוע. שימו לב, בניגוד למחשב הפשוט שתיארנו בפרק הראשון (שבו כל הוראה אוחסנה בתא אחד בזיכרון והמונה גדל ב-1 בכל פעם), במעבד 8086 האוגר IP גדל בכל פעם במספר שונה של בתים, בהתאם לאורך ההוראה שנקראה. לדוגמה: אם ההוראה mov al, ah אוחסנה בכתובת 0005 ואורכה שני בתים, ההוראה הבאה תימצא בכתובת 0007 ולכן יש להוסיף 2 לאוגר IP.

גם הפעולה השלישית בתהליך ההבאה תלויה בשיטת המיעון: אם ההוראה היא במיעון מיידי ובמיעון אוגר, פעולה זו אינה נחוצה, כי הנתונים נקראו בזמן ביצוע הפעולה הראשונה (קריאת ההוראה). אולם כאשר משתמשים במיעון ישיר, המעבד פונה שוב לזיכרון כדי לקרוא את הנתון הדרוש לביצוע ההוראה ואז זמן הביצוע ארוך יותר.

בביצוע הפעולה החמישית תהיה גישה לזיכרון אם אופרנד היעד הוא תא בזיכרון. במקרה כזה המעבד יבצע גישה נוספת לזיכרון, כדי לכתוב (לאחסן) בו את התוצאה.

שאלה 6.4

האם קיימת הוראה בשפת מכונה שבה יש גישה לזיכרון בתהליך הבאה-וביצוע בפעולה השלישית וגם בפעולה החמישית? הסבירו את תשובתכם ותנו דוגמה.

בפרק זה נקבע שמספר הגישות לזיכרון במהלך ביצוע הוראה, ישמש כמדד לזמן הביצוע. חישוב מדויק של זמן ביצוע הוראה הוא מורכב מאוד והוא תלוי בסוג ההוראה, במבנה המעבד, במהירות יחידת הזיכרון, ובגורמים רבים נוספים, שלא נתייחס אליהם כאן. מנקודת מבטו של המתכנת, אפשר לשפר את זמן הביצוע של תכנית על-ידי בחירה בשיטות מיעון שמפחיתות את מספר הגישות לזיכרון. בנוסף לכך, כפי שנראה בפרק האחרון, מפתחי חומרה של מעבדים מתקדמים מנסים לצמצם את זמן ההמתנה של מעבד בזמן גישה ליחידות זיכרון והתקני קלט/פלט וכך לשפר את זמן הביצוע של תכניות.

נרחיב כעת את ההסבר על שיטות המיעון האפשריות. למעשה, אפשר לחלק את שיטות המיעון לשלוש קבוצות עיקריות:

א. מיעון מיידי, שכבר הוסבר.
ב. מיעון אוגר, שגם הוא הוסבר.
ג. מיעון זיכרון. עד כה דיברנו רק על "מיעון ישיר" , אך למעשה "מיעון זיכרון" מגדיר קבוצה של שיטות מיעון בהן אחד מהאופרנדים בהוראה מציין כתובת בזיכרון.

את מיעון הזיכרון ניתן לחלק לשתי קבוצות עיקריות:

- מיעון ישיר – בו מצוינת, בצורה מפורשת, הכתובת של הנתון הדרוש הנמצא בזיכרון;
- מיעון עקיף – בו מחושבת, בצורה עקיפה, הכתובת של הנתון הדרוש שנמצא בזיכרון; הכתובת מחושבת בעזרת נתונים וכללים שמגדירה שיטת המיעון. גם קבוצה זו נחלקת למספר שיטות מיעון אותן נפרט בהמשך.

נציג כאת הסבר מלא על כל השיטות, כולל סיכום על אלה עליהן כבר דיברנו.

6.3 מיעון מיידי (Immediate addressing)

בשיטת המיעון המיידי, הנתון עצמו הוא חלק מהגדרת ההוראה, לדוגמה:

```
mov  al,  10h
mov  cx, 0FFh
```

בשיטת מיעון זו, הנתון יכול להיות אופרנד מקור בלבד. זמן ביצוע הוראה זו הוא קצר בדרך כלל, משום שהנתון נקרא בפעולה הראשונה של שלב ההבאה כחלק מהוראה עצמה, ולכן תהיה פניה לזיכרון רק במהלך הביצוע של פעולה זו. משתמשים בגישה זו כאשר יודעים את הנתון עצמו בעת כתיבת התכנית, למשל: כדי לאתחל משתנים, לאתחל מונים של לולאה וכדומה.

6.4 מיעון אוגר (Register addressing)

בשיטת מיעון זו, האופרנדים הם אוגרים. הוראה יכולה להכיל אופרנד אחד שהוא אוגר, לדוגמה:

```
inc ax
```

או שני אופרנדים שהם אוגרים:

```
add al, ah
```

בדומה לשיטת המיעון המיידי, גם הביצוע של הוראות בשיטת מיעון זו הוא מהיר יחסית, מפני שהפנייה לזיכרון נעשית רק בשלב ההבאה, שבו נקראת ההוראה עצמה. כדי לקצר את זמן הביצוע של התכנית, נעדיף בדרך כלל, להשתמש במיעון אוגרים ולבצע את הפעולות הדרושות על האוגרים. אולם מספר האוגרים למטרות כלליות הוא קטן, ולכן לא תמיד ניתן לאחסן בהם את כל משתני התכנית. במקרים אלה נעדיף להקצות אוגרים למשתנים שפונים אליהם פעמים רבות במהלך ביצוע התכנית, למשל כאשר משתמשים בלולאות, ואילו ליתר המשתנים, אליהם פונים פחות פעמים, נקצה מקומות בזיכרון.

דוגמה 6.1

נתרגם את האלגוריתם הבא להוראות בשפת אסמבלי ונשתמש במיעון אוגר בהוראות המרכיבות את גוף הלולאה:

a = 4
b = 3
לכל I מ-1 עד 100 בצע
$a \leftarrow a + 2$
$b \leftarrow b + 1$

במהלך ביצוע הלולאה נגשים הרבה פעמים (100 פעם) למשתנים a, b ו-i, לכן נעדיף לאחסן משתנים אלה באוגרים ונשתמש בהם לביצוע ההוראות בגוף הלולאה. בסיום הלולאה נאחסן את התוצאות בזיכרון. בהתאם לכך נתרגם את האלגוריתם ונרשום את ההוראות המתאימות בשפת סף:

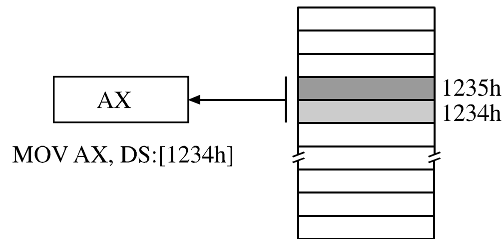
```
        mov   al,  a          ; העתקת נתונים מזיכרון לאוגרים
        mov   bl,  b          ; al = 4, bl = 3
        mov   cx, 100         ; אתחול מונה לולאה
                              ; גוף הלולאה
again:  add  al, 2            ; al = al + 2
        inc   bl              ; bl = bl + 1
        loop again            ; endloop
                              ; העתקת תוצאות לזיכרון
        mov   a, al           ; a = al
        mov   b, bl           ; b = bl
```

בתכנית זו, ההוראות בגוף הלולאה קובעות את משך הביצוע של התכנית משום שהן מתבצעות 100 פעם (בעוד שהוראות שלפני ואחרי הלולאה מתבצעות רק פעם אחת). אם רוצים להעריך את משך הביצוע של התכנית כולה, יש לבחון את ההוראות הרשומות בגוף הלולאה ולהעריך כמה פעמים המעבד פונה לזיכרון בתהליך הבאה-וביצוע. גוף הלולאה בתכנית כולל שתי הוראות: חיבור הנתון 2 לאוגר AL והוספת 1 לערכו של BL. בביצוע המחזור הבאה-וביצוע של כל אחת מההוראות הללו, מתבצעת פנייה לזיכרון רק כדי לקרוא את ההוראה (פעולה מס. 1 במחזור הבאה-ביצוע). גם בביצוע ההוראה LOOP הפנייה לזיכרון נעשית רק בשלב קריאת ההוראה עצמה.

6.5 מיעון ישיר (Direct addressing mode)

בשיטת המיעון הישיר אחד מהאופרנדים בהוראה מכיל את הכתובת של ***תא בזיכרון*** שמכיל את הנתון עליו תתבצע ההוראה. כתובת של תא בזיכרון יכולה להיות נתונה על-ידי ציון של שם משתנה או בצורה מפורשת. לדוגמה: נניח כי ההיסט של המשתנה a במקטע הנתונים הוא 1234h; שתי ההוראות הבאות מעתיקות את תוכן אותו תא ב\יכרון לאוגר AX והן תתורגמנה לאותה כתובת בזיכרון:

```
mov   ax, a
mov   ax, ds:[1234h]
```



איור 6.4
שיטת מיעון ישיר

בהוראה הראשונה, תכנית הקישור דואגת להגדיר את הכתובת האפקטיבית של המשתנה. ההוראה השנייה, לעומת זאת, מכילה רישום מפורש של כתובת המשתנה a ומשמעותה: העתק את הנתון, מכתובת הנמצאת בהיסט של 1234h בתים מכתובת תחילת סגמנט DS, אל אוגר AX. בדרך כלל, שימוש בצורה המפורשת DS:[offset] אינו מומלץ כי בזמן כתיבת התכנית איננו יודעים את הכתובת שבה יאוחסן הנתון, ולכן יהיה עלינו לחשב באופן ידני את ההיסט של הנתון. בנוסף לכך, שימוש בשם משתנה מסייע להבין גם את תפקידו בתכנית ולכן משפר את הקריאות שלה.

בדרך-כלל, זמן הביצוע של הוראות בשיטת מיעון מיידי ובשיטת מיעון אוגר, קצר יותר מאשר זמן הביצוע של אותן הוראות בשיטת מיעון ישיר. זמן ביצוע הוראות בשיטת מיעון ישיר ארוך יותר, מפני שבמהלך ביצוע מחזור ההבאה-והביצוע של הוראות אלו יש מספר פניות: פנייה אחת מתבצעת בפעולה הראשונה לקריאת ההוראה עצמה, פנייה שנייה מתבצעת בפעולה השלישית, כדי לקרוא מן הזיכרון את הנתון עצמו (במידה והאופרנד הוא אופרנד מקור) ופנייה שלישית מתבצעת בפעולה החמישית, כדי לכתוב את הנתון בזיכרון (במידה והאופרנד הוא אופרנד היעד).

דוגמה 6.2

נשנה את קטע התכנית שבדוגמה 6.1 ונשתמש בשיטת מיעון ישיר ונחשב את זמן הביצוע המשוער של התכנית:

```
        mov  cx, 100          ; אתחול מונה הלולאה
                              ; גוף הלולאה
again:  add  a, 2             ; a ← a + 2
```

```
inc  b                ; b ← b + 1
loop  again           ; endloop
```

שימו לב: a ו-b הם שמות תאים בזיכרון, ולא שמות אוגרים!
כמו כן שימו לב, שתכנית זו קצרה יותר מהתכנית שרשומה בדוגמה 6.1, מפני שלא צריך להעתיק נתונים ממשתנים לאוגרים ולהיפך. נבחן כעת את מספר הפניות לזיכרון בביצוע ההוראות בגוף הלולאה

```
add a, 2
inc  b
```

המחזור הבאה-וביצוע של כל אחת מההוראות האלה, כולל פנייה לזיכרון בפעולות אלה:

- בפעולה הראשונה שבה נקראת ההוראה מהזיכרון;
- בפעולה השלישית, כדי לקרוא את ערך האופרנד המשתתף בחישוב;
- בפעולה החמישית, כדי לכתוב את תוצאת החישוב בזיכרון.

אפשר לראות כי למרות שקטע תכנית זה קצר יותר, ביצועו יארך זמן רב יותר, וזאת משום שבמהלך ביצוע כל אחת משתי ההוראות הראשונות בגוף הלולאה, יש 3 פניות לזיכרון (במקום פנייה אחת במיעון אוגר).

שאלה 6.5

רשמו את שיטת המיעון המתאימה לכל הוראת ADD, MOV ו-INC בקטע התכנית שהובא בדוגמה 6.1.

שאלה 6.6

בזיכרון הוגדרו שני משתנים:

```
a   DB 20h
b   DB 51h
```

א. לכל אחת מההוראות הבאות, רשמו את הפעולות במחזור

הבאה-וביצוע בהן יש פנייה לזיכרון:

```
mov al, a
add  bl, b
add  bl, a
mov b, bl
```

ב. חשבו מה יהיה ערכו המספרי של b בסיום.

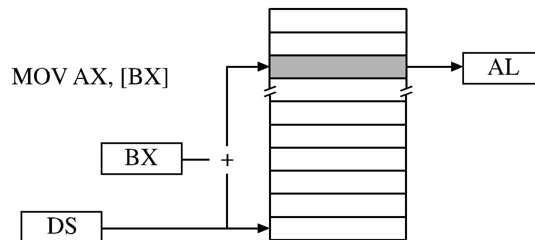
6.6 מיעון עקיף בעזרת אוגר (Register Indirect Addressing)

בניגוד למיעון ישיר, בו מציינים בצורה מפורשת את כתובת המשתנה (כלומר את כתובת הנתון בזיכרון), במיעון עקיף כתובת המשתנה נרשמת בצורה עקיפה. התייחסות לכתובת בצורה עקיפה מאפשרת לבצע חישובים על כתובות וכך, כפי שנדגים בהמשך, מתאפשרת הגישה לקבוצה של תאי זיכרון בה מאוחסנים למשל איברי מערך חד-ממדי.

בשיטת מיעון עקיף בעזרת אוגר, כתובתו של האופרנד מאוחסנת באחד מהאוגרים האלה: BX, BP, SI, DI. לדוגמה, המשמעות של ההוראה

```
mov al, [bx]
```

היא: העתק לאוגר AL את הנתון המאוחסן בכתובת עליה מצביע אוגר BX. האוגר שבסוגריים אינו מכיל את הנתון המיידי שהאמור להכנס לאוגר AL, אלא את כתובתו של נתון זה. הסימול [ ] סביב האוגר BX מציין שהאוגר מצביע על כתובת של נתון המאוחסן בזיכרון. איור 6.5 מציג את אופן החישוב של כתובת תא בזיכרון במיעון עקיף.



איור 6.5
שיטת מיעון עקיף בעזרת אוגר

לדוגמה, נניח כי ערכו של האוגר BX הוא 100h. לאחר ביצוע ההוראה, מועתק לאוגר AL הנתון השמור בכתובת DS:[0100h].

שימו לב: בשיטת מיעון זו אפשר להשתמש רק באוגרים BX, SI, DI, BP ולכן ההוראה:

```
add ax, [cx]
```

אינה חוקית.

ארבעת האוגרים שמצביעים על כתובת תא בזיכרון, מציינים היסט ביחס לאוגר מקטע מסוים, כאשר:

- האוגרים BX, SI, DI מציינים את ההיסט יחסית לאוגר סגמנט הנתונים DS;
- אוגר BP מכיל היסט יחסית לסגמנט המחסנית SS (עליו נרחיב את ההסבר בפרק הבא).

6.6.1 אתחול אוגר מצביע

לפני שמשתמשים בשיטת מיעון עקיף בעזרת אוגר, יש לדאוג שהאוגר המצביע המתאים, יכיל את הכתובת של המשתנה אליו רוצים לפנות. נתאר שתי שיטות לאתחול אוגר מצביע בכתובת:

א. בשיטה הראשונה נשתמש בהוראת מיעון הנקראת LEA.

מבנה ההוראה **LEA** (Load Effective Address) הוא:

LEA אוגר, כתובת

ומשמעותה: טען את הכתובת של תא בזיכרון לתוך אוגר מצביע שחייב להיות בן 16 סיביות. הוראה זו שייכת לקבוצת הוראות העברה, ובדומה לשאר ההוראות בקבוצה זו אין לה השפעה על אוגר הדגלים.

לדוגמה: ההוראה

```
lea bx, a
```

מעתיקה לאוגר BX את הכתובת של משתנה a.

שימו לב, יש הבדל בין שתי ההוראות האלה:

```
mov bx, a        ; הנתון השמור במשתנה a מועתק לאוגר BX
lea bx, a        ; הכתובת של משתנה a מועתקת לאוגר BX
```

ב. בשיטה השנייה נשתמש ב**אופרטור offset** ו**במצביע PTR**.

באופרטור offset משתמשים לפני שם המשתנה :

```
mov bx, offset a
```

בהוראה זו תוצאת האופרטור offset היא ההיסט של המשתנה a, ולכן הוראת MOV זו מעבירה לאוגר BX את ההיסט, כלומר את מספר הבתים מתחילת המקטע הנתונים.

אחת הבעיות שמתעוררת כאשר משתמשים באופרטור offset היא שלעיתים ההוראה עצמה לא מספקת מידע חד-משמעי על טיפוס תא זיכרון עליו מצביע האוגר, ובמקרה כזה נקבל שגיאת הידור. לדוגמה, כאשר רושמים את ההוראה :

```
mov [bx], al
```

אופרנד המקור הוא מטיפוס בית, ובהתאם, המהדר מפענח את ההצבעה [bx] לנתון מטיפוס בית. אולם כאשר רושמים את ההוראה

```
mov [bx], 0FFh
```

הכתובת [bx] לא מלמדת על הטיפוס של הנתון ולכן המהדר לא יכול לדעת אם יש לאחסן את הנתון 0FFh בתא זיכרון מטיפוס בית או בתא זיכרון מטיפוס מילה. כדי לאכוף טיפוס נתונים מסוים, משתמשים באופרטור PTR שיכול להיות אחד משני הסוגים :

byte ptr	לציון בית
word ptr	לציון מילה

לדוגמה :

```
mov byte  ptr [bx], 0FFh        ; מתייחס לטיפוס נתון עליו מצביע bx כאל בית
mov word ptr [bx], 0FFh         ; מתייחס לטיפוס נתון עליו מצביע bx כאל מילה
```

אנו יכולים להשתמש באופרטור PTR כדי לאכוף פנייה לבית אחד מתוך מילה; לדוגמה, נגדיר משתנה מטיפוס מילה :

```
b  DW 1234
```

ונרשום הוראה שמעתיקה לאוגר AH רק את הבית העליון של משתנה b :

```
mov ah, byte ptr b+1                 ;ah = 12
```

שאלה 6.7

הסבירו מה מבצעת התכנית הרשומה להלן; רשמו לכל הוראה הערה שתתאר את הפעולה ואת הערך המושם באופרנד היעד.

```
.MODEL SMALL
.STACK 100h
.DATA
    x  DD 12345678h
    y  DD  11A2092Ah
    z  DD   ?
    t  DD   ?
.CODE
start:
    mov  ax,             @DATA
    mov  ds,             ax
    mov  ax,             word ptr x
    add   ax,            word ptr y
    mov  word ptr z,     ax
    mov  ax,             word ptr x+2
    adc    ax,           word ptr y+2
    mov  word ptr z+2,   ax
    mov  ax,             004ch
    int    21h
END start
```

6.6.2 חישוב כתובות

בשיטת מיעון זו, בדומה לשאר שיטות המיעון העקיפות (שנציג בהמשך), אנו מתייחסים לכתובת כאל מספר המאוחסן באוגר שאפשר לבצע עליו פעולות חשבון, כמו חיבור וחיסור. תוצאת החישוב היא מספר המבטא כתובת אחרת בזיכרון, שאפשר לפנות אליה בהמשך. כך למשל ניתן לחשב את הכתובת של משתנה מסוים, כאשר ידוע מיקומו ביחס לכתובת נתונה של משתנה אחר.

דוגמה 6.3

נציג תכנית שמשווה בין שני נתונים מטיפוס בית, המאוחסנים בזיכרון בתאים עוקבים ואם הנתונים אינם שווים, נגדיל את ערכו של הנתון השני (b) ב-1. בתכנית זו נחשב את כתובת אחד המשתנים, בעזרת הכתובת של המשתנה השני.

```
.MODEL SMALL
.STACK 100h
.DATA
  a DB 01
  b DB 02
.CODE
start:
                                         ; אתחול מקטע הנתונים
   mov  ax, @DATA
   mov  ds,  ax
   lea    bx, a                          ; bx ← a כתובת של משתנה
   mov  al,  [bx]                        ; al ← a
      inc    bx                          ; b חישוב כתובת משתנה
      cmp  al, [bx]                      ; a=b האם
      je     end_if
      inc    byte ptr [bx]               ; b ← b + 1
end_if:
                                         ; סיום התכנית
      mov  ax, 004ch
       int    21h
end start
```

בתכנית זו השתמשנו בהוראה LEA כדי להעתיק את כתובת המשתנה a לאוגר BX ובאמצעותו חישבנו את כתובתו של המשתנה b כסכום של כתובתו של המשתנה a ומספר הבתים שתופס משתנה a.

שאלה 6.8

אנו רוצים להחליף בין ערכי המשתנים b ו-c, המוגדרים במקטע הנתונים בצורה זו :

```
b   DW  129h
c   DW  0AFh
```

לשם כך רשמנו את שתי ההוראות האלה :

```
lea     bx,  c
xchg [bx], b
```

א. האם ההוראות תקינות?

ב. אם לא, הסבירו מדוע ורשמו הוראות שיבצעו את ההחלפה בצורה תקינה.

6.6.3 יישום משתנה מטיפוס מצביע

שיטת מיעון זו מאפשרת לממש טיפוס נתונים שנקרא **מצביע** (pointer). מצביע הוא משתנה המכיל מען (כתובת) של משתנה אחר הנמצא בזיכרון. משתנה המגדיר מצביע חייב להיות מטיפוס מילה, משום שכתובת אפקטיבית בזיכרון היא בת 16 סיביות.

דוגמה 6.4

בדוגמה זו נציג שימוש במצביעים. תחילה נגדיר שני מצביעים : מצביע p המצביע למשתנה a (מכיל את הכתובת של משתנה a) ומצביע q שאינו מאותחל.

```
a  db   172
b  dw  120
p  dw   a          ; p מצביע ל-a
q  dw   ?          ; q מצביע לא מאותחל
```

לכאורה ההצהרה על מצביע p נראית מוזרה, אולם אם נזכור כי שם משתנה הוא שם לוגי של כתובת ובזמן תרגום ההוראה לשפת מכונה, מוחלף שם המשתנה בהיסט (כתובת אפקטיבית) של המשתנה במקטע הנתונים, אזי ברישום ההצהרה

```
p  dw  a
```

מוקצה מקום בזיכרון למשתנה p שהוא מטיפוס מילה והוא מאותחל לכתובת אפקטיבית של משתנה a. בדוגמה שתיארנו נניח כי הכתובת האפקטיבית של a היא 0000, ולכן p מאותחל לערך זה.

נציג כעת סדרת הוראות בהן נשתמש במצביע לביצוע מספר פעולות:

א. כדי להשתמש במצביע, עלינו להעתיק את המצביע לאוגר מצביע. נרשום את ההוראה:

```
mov  bx, p                                        ; bx = p
```

ב. כעת נשתמש בהצבעה כדי להעתיק את תוכן המשתנה a למשתנה x.:

```
mov  al,  [bx]                                    ; al = 0ACh
mov  x,  al                                       ; x = a
```

ג. כדי שמצביע q יצביע על אותו משתנה עליו מצביע p, נרשום את ההוראה:

```
mov  q,  bx                                       ; q = p = 0001
```

שאלה 6.9

א. האם ביצוע ההוראה הבאה תקין:

```
mov  x,  [p]
```

אם לא, הסבירו מדוע?

ב. כתבו הוראות כך שמשתנה q יצביע על כתובת תא בזיכרון שעוקבת לכתובת עליה מצביע p.

6.7 מיעון אינדקס ישיר (Direct Indexed Addressing)

6.7.1 רישום אופרנד שיטת מיעון אינדקס ישיר

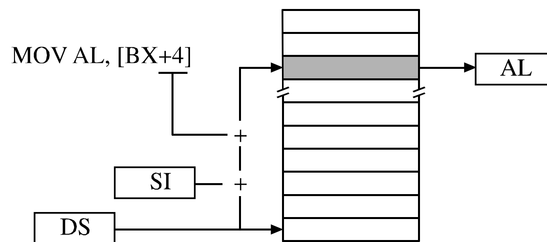
בשיטת מיעון אינדקס ישיר, הכתובת האפקטיבית של נתון מתקבלת על-ידי חיבור של ההעתק הרשום בהוראה לערכו של אוגר מצביע:

העתק + אוגר מצביע = כתובת תא בזיכרון

נתאר תחילה את התחביר של שיטה זו: אוגר המצביע יכול להיות אחד מאוגרי האינדקס – SI ו-DI – והכתובת היא היסט, יחסית לאוגר מקטע הנתונים DS. ההעתק הוא מספר עם סימן בגודל בית או מילה. אפשר לרשום הוראות בשיטה מיעון זו בשתי צורות:

```
mov  al,  [si+4]
mov  al,  4[si]
```

שתי הוראות אלה הן זהות, ומשמעותן: העתק את תוכן הבית שכתובתו היחסית היא SI+4 לאוגר AL. איור 6.6 מציג את אופן החישוב של הכתובת האפקטיבית של תא בזיכרון בשיטת מיעון זו. הנתון שנמצא בכתובת DS:[SI+4] מועתק בהוראה זו לאוגר AL; disp הוא קיצור של displacement, דהיינו: "הֶעתק".



איור 6.6
מיעון אינדקס ישיר

לדוגמה, אם נניח כי האוגר SI שבאיור 6.6 מכיל את הכתובת 100h, הנתון שבכתובת 104h יועתק לאוגר AL. נציג כמה דוגמאות נוספות לרישום הוראות בשיטת מיעון זו:

add ax, [si+4]	; חבר את תוכן אוגר AX עם תוכן המשתנה DS:[SI+4]
mov [di-6], cx	; העתק את תוכן האוגר CX למשתנה מטיפוס מילה שכתובתו היא DS:[SI–6] ;
sub word ptr [si-192], 100h	; חסר 100h מהמשתנה מטיפוס מילה שכתובתו היא DS:[SI–192]

שימו לב, תוכן האוגרים SI ו-DI אינו משתנה בעקבות הוראה זו.

6.7.2 גישה לאיברי מערך חד-ממדי

נציג כעת כיצד משתמשים בשיטת מיעון זו כדי לטפל במערך חד-ממדי. כדי לגשת לאיבר במערך, עלינו לרשום את הנתונים אלה:

א. ההעתק מציין את כתובת תחילת המערך;
ב. אוגר האינדקס, המציין את מרחק האיבר מתחילת המערך, מחושב כך:

מספר איבר · גודל טיפוס איבר

כאשר המציין לאיבר הראשון במערך הוא 0.

נרשום כמה דוגמאות:

א. כדי להעתיק את האיבר הרביעי במערך a, שאיבריו הם מטיפוס בית, לאוגר AL, נרשום:

```
mov si, 3
mov al, a[si]
```

ב. באופן דומה, כדי להעתיק לאוגר AX את האיבר הרביעי במערך a, האיבר a[3], שאיבריו הם מטיפוס מילה, נרשום:

```
mov si,  2·3
mov ax, a[si]
```

ג. נעתיק את האיבר השני ממערך a, שאיבריו הם מטיפוס בית, ונשים אותו באיבר השלישי, כלומר, נבצע את ההשמה הזו: a[3] ← a[2]

כדי לבצע זאת נרשום את ההוראות האלה:

```
mov si, 1          ; SI מצביע על האיבר השני
mov al, a[si]      ; העתק את האיבר השני לאוגר AL
inc   si           ; חשב את הכתובת של האיבר השלישי
mov a[si], al      ; העתק את אוגר AL לאיבר השלישי
```

שאלה 6.10

נתונים שני מערכים המוגדרים בצורה הזו :

```
q  DW  1,2,3,4,5
p  DW  5 dup(?)
```

רשמו הוראות מתאימות לביצוע ההשמות האלה :

- $p[2]\leftarrow q[2]$
- $p[4] \leftarrow q[3]$
- $p[1] \leftarrow q[4]$

זכרו שמציין תא מתחיל ב-0.

שיטת מיעון זו מאפשרת גישה נוחה לכל האיברים של מערך חד-ממדי (או לחלק מהם). שימו לב, אם הערך המושם באוגר SI חורג מגבולות המערך, לא נקבל הודעת שגיאה (אלא אם נפנה לאזור בזיכרון שאליו אנו לא מורשים לפנות; הדיון בנושא זה יובא בפרק האחרון) וביצוע התכנית יימשך כרגיל. סביר להניח שבמקרה כזה נקבל תוצאות שגויות שאינן צפויות. בשפת אסמבלי, האחריות לחישוב כתובות תקינות מוטלת על המתכנת, בניגוד לשפות עיליות (כדוגמת שפת פסקל, JAVA), שבהן חריגה כזו תגרור שגיאת הידור.

דוגמה 6.5

נציג תכנית המסכמת איברי מערך a, המכיל 5 איברים מטיפוס בית. האלגוריתם שנממש הוא :

השם sum = 0
לכל i = 0 עד i = 4 בצע
$sum \leftarrow sum + a[i]$

פתרון

לחישוב כתובת של איבר במערך : ההעתק יצביע על הכתובת של האיבר הראשון במערך, ובאוגר SI נצביע על איבר i במערך. המערך הוא מטיפוס בית, עלינו להוסיף 1 לאוגר SI, כדי להצביע על איבר העוקב לאיבר ה- i .

```
.MODEL SMALL
.STACK 100h
.DATA
    a DB 10, 20, 30, 40, 50          ; אתחול מערך שבו 5 איברים בגודל בית
    sum db 0                          ;סכום איברי המערך
.CODE
start:
;אתחול מקטע הנתונים
mov  ax, @DATA
mov  ds, ax
;אתחול משתנים
    mov  si,  0                       ; מצביע על איבר המערך
    mov  al,  0                       ; סכום איברי מערך
again:                                ;גוף הלולאה
    add   al,  a[si]                  ; al ← al + a[si]
    inc   si                          ;si ← si + 1  קדם מציין לאיבר במערך
    cmp  si,   5                      ; השווה מציין מערך למספר איברי המערך
    jne   again                       ; מציין שונה מאיבר אחרון במערך

    mov  sum, bl                      ; sum  העתק סכום איברים למשתנה
;יציאה מתכנית
    mov  ax, 004ch
    int    21h
end start
```

כדי לכתוב תכנית כללית יותר, נשתדל להימנע מלרשום בהוראות את מספר האיברים בצורה מפורשת; במקום זאת נשתמש בקבוע MAX שמציין את מספר האיברים במערך. לשימוש בקבוע יש יתרון, כי הדבר היחיד שנצטרך לעשות כאשר נרצה לשנות את גודל המערך, יהיה לשנות את ערכו של הקבוע .

שאלה 6.11

א. חזרו לתכנית שהוצגה בפתרון לדוגמה 6.5 והכניסו בה שינוי: השתמשו בהוראה LOOP כדי לסכם מערך שאיבריו מטיפוס מילה וגודלו מוגדר כקבוע MAX.

ב. כתבו תכנית שתסכם איברי מערך חד-ממדי שגודלו אינו נתון אך האיבר האחרון בו הוא 1–. הניחו כי גודל המערך קטן מ-100.

לסיכום, נציין כי אפשר לכתוב תכנית דומה ולהשתמש במיעון אוגר עקיף כדי לגשת לאיבר במערך. לדוגמה, נרשום את ההוראה:

```
add al, [si]
```

עֶרכו התחילי של SI מכיל את הכתובת האפקטיבית של האיבר הראשון במערך.

החיסרון של שיטה זו הוא: הכתובת של תחילת המערך אינה נשמרת ואם נצטרך לבצע כמה פעולות על המערך, נצטרך לאתחל, בכל פעולה, את ערכו של SI לכתובת האפקטיבית של האיבר הראשון.

שאלה 6.12

כתבו תכנית שתגדיר ואתתחל מערך מטיפוס בית, בן 10 איברים, המכילים מספרים עם סימן. התכנית תמצא את המספר הקטן ביותר ואת המספר הגדול ביותר במערך. הניחו כי איברי המערך שונים זה מזה.

שאלה 6.13

כתבו תכנית שתגדיר מערך מטיפוס מילה, בן 20 איברים, ותאתחל אותו לסדרת המספרים: 1, 2, 3,....

6.8 מיעון בסיס (Base Relative Addressing)

שיטת מיעון זו דומה לשיטת מיעון אינדקס ישיר, אלא שבה משתמשים להצבעה באחד מאוגרי הבסיס BX ו-BP. אוגר BX מתאר היסט של כתובת יחסית לאוגר מקטע הנתונים (DS) ואוגר BP מתאר היסט של כתובת יחסית לאוגר מקטע מחסנית (SS).

הכתובת במיעון בסיס מחושבת כסכום של:

ההעתק + אוגר הבסיס = כתובת תא בזיכרון

לדוגמה, נציג כמה הוראות בשיטת מיעון זו:

```
mov  ax, [bx – 3]        ; AX לאוגר  BX – 3  העתק איבר שכתובתו האפקטיבית היא
add  ax,  a[bx]          ; AX לאוגר BX  הוא a חבר את האיבר שההיסט שלו מתחילת מערך
```

מנקודת מבטו של מתכנת, אין הבדל מהותי בין מיעון אינדקס ישיר לבין מיעון בסיס ובמקרים מסוימים מתייחסים לשתי השיטות הללו בשם מיעון אינדקס ישיר.

גישה לשדות רשומה במיעון בסיס

שיטה מיעון זו נוחה כדי לגשת לשדה ברשומה (או למבנה). במערך בו כל האיברים הם מאותו טיפוס, ולכן נוח להחזיק את המציין לאיבר באוגר מצביע ולחשב את האיבר הבא על-ידי תוספת (או חיסור) של גודל האיבר במערך. ברשומה, לעומת זאת, כל שדה יכול להיות מטיפוס שונה, לכן ביצוע חישוב על אוגר אינדקס לא יעיל. הפעולות שנצטרך לבצע בשיטת מיעון בסיס, כדי להצביע על שדה ברשומה, הן:

א. אוגר הבסיס יצביע על הכתובת של השדה הראשון ברשומה;

ב. ההעתק יציין את ההיסט של שדה מסוים מתחילת הרשומה.

לדוגמה, נשתמש ברשומת פריט שהגדרנו בסעיף 6.1, וכדי להעתק את כמות הפריט שמכילה הרשומה הראשונה לאוגר AL, נרשום את ההוראות האלה:

```
lea    bx, p1
mov  al,   [bx+2]
```

דוגמה 6.6

נכתוב תכנית מלאה, שתסכם את כמות הפריטים שיש בשתי רשומות הפריט:

```
.MODEL SMALL
.STACK 100h
```

; הגדרה על מבנה רשומת פריט

```
item STRUC
    itemnum   DW ?
    quantity   DB  ?
    supplier   DW ?
item ENDS
.DATA
                                        ; אתחול הערכים של שתי רשומות פריט
    p1item <1111,100, 923>
    p2item <2222,50,120>
.CODE
start:
                                        ;אתחול מקטע הנתונים
    mov  ax, @DATA
    mov  ds,  ax
    lea    bx,  p1                      ; BX מצביע על תחילת הרשומה הראשונה
    mov  al,  [bx+2]                    ; AL = תוכן השדה כמות של הרשומה הראשונה
    lea    bx,  p2                      ; BX מצביע על תחילת הרשומה השנייה
    add   al,  [bx+2]                   ; AL ← AL + תוכן השדה כמות של הרשומה השנייה
                                        ; יציאה מתכנית
    mov  ax, 004ch
    int     21h
end start
```

שאלה 6.14

א. הגדירו רשומה שתתאר את הפריט **תלמיד** ותכלול את השדות האלה:

- מספר תלמיד מטיפוס מילה;
- מספר הכיתה שבה לומד התלמיד מטיפוס בית;
- ציון במקצוע מדעי המחשב מטיפוס בית;

ב. אתחלו בערכים עשר רשומות של תלמידים וכתבו תכנית שתבדוק אם הציונים של כל תלמידים בכיתה מסוימת הם חיוביים (גדולים מ-60). אם ציוני כל התלמידים בכיתה חיוביים, התכנית תשים את הערך 1 במשתנה a ; אחרת יושם בו הערך 0.

שאלה 6.15

א. השתמשו בהגדרת רשומת הפריט שהובאה בדוגמה 6.6 והגדירו חמישה פריטים שונים.

ב. כתבו תכנית שתכיל משתנה שבו רשום מספר פריט ומערך מכירות; כל איבר במערך המכירות יכיל את מספר הפריטים שנמכרו מאותו פריט. התכנית תעדכן את תוכן השדה **כמות** ברשומת אותו פריט, לאחר שיופחתו כל הכמויות שנמכרו. בחישובכם הניחו כי סך-כל המכירות קטן מהכמות במלאי.

6.9 מיעון אינדקס-בסיס (Based Indexed Addressing Modes)

שיטת מיעון זו משלבת את שתי שיטות המיעון הקודמות: שיטת מיעון אינדקס ישיר עם שיטת מיעון בסיס, ובהתאם, חישוב כתובת הנתון אליו פונים בגישה זו נעשה במבנה זה:

[העתק] + אוגר הבסיס + אוגר אינדקס = כתובת תא בזיכרון

כאשר ההעתק הוא אופציונלי.

ברישום כתובת בשיטה זו קיימים צירופי אוגרים מותרים ואפשר לרשום את הכתובת בשתי צורות:

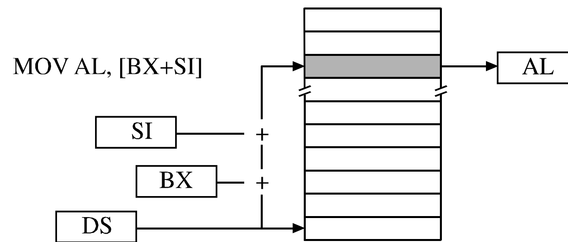
[bx][si] או [bx + si]
[bx][di] או [bx + di]
[bp][si] או [bp + si]
[bp][di] או [bp + di]

נזכיר שוב, כי ברישום כתובת שבה אוגר הבסיס הוא BX, האוגר מתייחס להיסט במקטע נתונים, ואילו ברישום כתובת שבה אוגר הבסיס הוא BP, האוגר מתייחס להיסט במקטע המחסנית.

לדוגמה, לפנינו ההוראה

```
mov  al,  [bx][si]
```

נניח כי BX = 1000h והאוגר SI = 880h, ההוראה הזו תעתיק את הנתון הרשום בכתובת DS:1880 אל אוגר AL, כפי שאפשר לראות באיור 6.7.



איור 6.7
מיעון אינדקס-בסיס

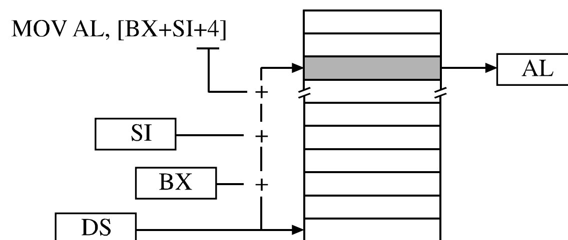
באופן דומה אפשר לרשום כתובת במיעון אינדקס-בסיס עם העתק

[BX+SI+disp]	או	disp[BX] [si]
[BX+DI+disp]	או	disp[BX][DI]
[BP+SI+disp]	או	disp[BP][SI]
[BP+DI+disp]	או	disp[BP][DI]

לדוגמה: באיור 6.7 הוצג חישוב של הכתובת במיעון אינדקס-בסיס עם העתק, אם נניח כי BX = 1000h, האוגר SI = 880h וההעתק הוא 4, ההוראה

```
mov  al,  4[bx][si]
```

תעתיק את הנתון בכתובת DS:1884 אל האוגר AL, כמתואר באיור 6.8.



איור 6.8
מיעון אינדקס-בסיס עם העתק

6.9.2 עיבוד מערך דו-ממדי

שיטת מיעון אינדקס-בסיס מאפשרת גמישות רבה יותר מאשר שתי שיטות המיעון הקודמות, כי אפשר לשנות בה את ערך אוגר הבסיס או את ערך אוגר האינדקס או את שניהם גם יחד. שיטה זו מאפשרת לעבד מבני נתונים שיש להם שני מציינים כמו מערך דו-ממדי או רשומה שמכילה שדות שהם מערכים. בסעיף נציג כמה תכניות לדוגמה, ונראה איך הן מעבדות מערך דו-ממדי.

דוגמה 6.7

נכתוב תכנית שתגדיר מערך דו-ממדי שיש בו 5 שורות ו-10 עמודות, ותאתחל את איברי המערך לסדרת המספרים הזוגיים, החל מ-0, 2, 4, 6, ... וכך הלאה.

```
.MODEL SMALL
.STACK 100h
                                   הגדרת קבועים;
ROW EQU 5
COLUMN EQU 10
.DATA
    a DB ROW×COLUMN dup(?)          ; 5×10 הגדרת מערך דו-ממדי
.CODE
start:
    mov ax, @DATA
    mov ds, ax
                                   אתחול אוגרי עזר ;
    mov al, 0                       ; איבר בסדרה
    mov si, 0                       ; מצביע לאיבר
again:
    ; לכל איבר במערך עליו מצביע BX החל מ-0 ועד ROW×COLUMN בצע
    mov a[si], al
    add al, 2
```

```
    inc   si
    cmp si, ROW×COLUMN        ; האם עברנו על כל איברי המערך?
    jb again
; סיום התכנית
    mov ax,   4c00h
    int    21h
END start
```

שימו לב, המערך הדו-ממדי ממפה בצורה לינארית, לכן מספיקה לולאה אחת לאתחול כל איבריו.

דוגמה 6.8

נכתוב תכנית שתאחל מערך דו-ממדי שיש בו 3 שורות ו-4 עמודות. לאתחול נשתמש בסדרת המספרים ...,10,20,30 התכנית תחשב את הסכום של כל שורה בנפרד, ותאחסן את הסכום במערך חד-ממדי sum שיש בו 3 איברים (כל איבר במערך חד-ממדי מאחסן סכום של שורה אחת במערך הדו-ממדי).

פתרון

נגדיר שני קבועים: האחד, בשם ROW, יציין את מספר השורות, והשני בשם COLUMN יציין את מספר העמודות.

כמו כן נגדיר שני מערכים:

- a מערך דו-ממדי;
- sum מערך חד-ממדי שבו יאוחסנו סכומי השורות.

נגדיר את שיטות המיעון בהן נשתמש:

- בהצבעה לאיבר במערך הדו-ממדי a נשתמש במיעון אינדקס-בסיס עם העתק ובהתאם נשתמש באוגרים BX ו-SI.
- בהצבעה לאיבר במערך החד-ממדי נשתמש במיעון אינדקס ובהתאם נשתמש באוגר DI.

להלן התכנית המתאימה:

```
.MODEL SMALL
.STACK 100h
; הגדרת קבועים
ROW        EQU  3
COLUMN  EQU  4
.DATA
; אתחול מערך דו-ממדי
    a DB 10,20,30,40,50,60,70,80,90,100,110,120
    sum DB ROW dup(0)                           ; אתחול מערך חד-ממדי
.CODE
start:
    mov  ax,  @DATA
    mov  ds,  ax
; אתחול אוגרים
    xor    bx,  bx                               ; מציין שורה במערך a
    xor    di,  di                                ; מציין איבר במערך sum
; סיכום שורות: לכל BX מ-0 עד ROW×COLUMN בצע
  loop_row:
; אתחול אוגרים
    xor    si,  si                                 ; מציין עמודה במערך a
    xor    cl,  cl                                 ; מסכם של איברי השורה
; לולאה פנימית  לסיכום שורה אחת
; לכל SI מ-0 עד COLUMN-1 בצע
    loop_column:
        add   cl,  a[bx][si]
        inc   si
        cmp  si, COLUMN
        jb loop_column
; סוף לולאה פנימית
```

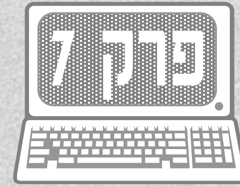
```
    mov  sum[di],  cl                    ; sum  אחסון תוצאה במערך
    inc   di
    add  bx,   COLUMN
    cmp  bx,  ROW×COLUMN
    jb loop_row
                                         ; סיום לולאה חיצונית ויציאה מהתכנית
    mov  ax,  4ch
    int    21h
END start
```

שאלה 6.16

כתבו תכנית שתאחסן את סדרת המספרים 1, 4, 7, 10, 13,... במערך דו-ממדי בגודל 4×3.

שאלה 6.17

כתבו תכנית שתמצא את האיבר המקסימלי בכל שורה במערך דו-ממדי בגודל 4 עמודות ו-6 שורות, ותאחסן את הסכומים במערך חד-ממדי.



מחסנית, שגרות ומקרו

7.1 מבוא

שגרה (routine) היא קבוצת הוראות בתכנית מחשב, לביצוע משימה מסוימת. בתהליך הפיתוח והכתיבה של תכנית נרחבת, ההתמודדות נעשית קלה יותר כאשר מפרקים את הבעיה למשימות קטנות שכל אחת מהן מוגדרת כשגרה. גישה זו מאפשרת להתמקד כל פעם בכתיבה ובדיקה של שגרה אחת בנפרד; בסיום מרכיבים את התכנית כולה מן השגרות השונות שנכתבו. לאחר ששגרה נבדקה, נוכל להשתמש בה שוב ושוב באותה תכנית או בתכניות אחרות וכך לחסוך חלק מן הזמן הדרוש לכתיבת התכנית.

בשפות עיליות ממומשים שני סוגים של שגרות: פרוצדורות (Procedures או הליך) שאינן מחזירות שום ערך ופונקציות (Functions) שמחזירות ערך יחיד או יותר. בשפת אסמבלי קיים מנגנון למימוש פרוצדורות בלבד ובעזרתו ניתן, כפי שנראה בהמשך, להגדיר פונקציות.

לפרוצדורה יש מחזור חיים המתחיל ברגע שתכנית קוראת לפרוצדורה (Call); אחרי שהפרוצדורה מסתיימת, התכנית מתחדשת, לפי ההוראות הרשומות אחרי הפרוצדורה. בזמן הקריאה לפרוצדורה אפשר להעביר לה פרמטרים ובסיום הפרוצדורה אפשר להשתמש במנגנונים שונים כדי להחזיר (אם צריך) לתכנית הקוראת את תוצאות העיבוד. בנוסף, פרוצדורה יכולה להצהיר ולהשתמש במשתנים מקומיים שמשך החיים שלהם חופף למחזור החיים של הפרוצדורה. כלומר, בזמן ביצוע הפרוצדורה מוקצה זיכרון למשתנים המקומיים, וברגע שהפרוצדורה מסתיימת והביצוע חוזר לתכנית שקראה לה, ההקצאה למשתנים המקומיים מתבטלת והם אינם קיימים יותר. מנגנון הקצאת מקום בזיכרון למשתנים מקומיים שונה ממנגנון הקצאת הזיכרון למשתני התכנית (המוגדרים במקטע הנתונים) שמשך החיים שלהם הוא כמחזור חיי התכנית עצמה.

בפרק זה נתאר כיצד כותבים ומשתמשים בפרוצדורות ופונקציות בשפת אסמבלי. כמו כן, נסביר איך משתמשים במחסנית לביצוע פרוצדורה, ונציג שיטות להעברת פרמטרים לפרוצדורה והקצאת משתנים מקומיים וכיצד מתבצעת פונקציה רקורסיבית. לסיום נציג מנגנון אחר לביצוע קטע קוד שחוזר על עצמו הנקרא מקרו (macro) ונשווה בינו לבין שגרה.

7.2 כתיבת פרוצדורה וזימונה

בשפת אסמבלי פרוצדורה היא קטע קוד הנכתב כחלק מהתכנית (במקטע הקוד). לציון התחלת הפרוצדורה וסיומה משתמשים בהנחיות מיוחדות: ההנחיה PROC מגדירה את תחילת הפרוצדורה, וההנחיה ENDP מגדירה את סיומה:

```
<שם פרוצדורה> PROC
        גוף הפרוצדורה

    RET
<שם הפרוצדורה> ENDP
```

דוגמה 7.1

נכתוב תכנית שתשתמש בפרוצדורה שתפקידה לאתחל את אוגר מקטע הנתונים.

```
.MODEL SMALL
.STACK 100h
.DATA
    a   DB  ?
    x   DW  ?
.CODE
; תכנית ראשית
start:
    call  initData        ; זימון פרוצדורה
; סיום התכנית
    mov  ax,  4c00h
    int   21h
```

```
; הגדרת הפרוצדורה initData
initData PROC
    mov ax, @DATA
    mov ds, ax
    ret                         ; חזרה מפרוצדורה
initData ENDP                   ; סוף הגדרת הפרוצדורה
END start
```

בתכנית זו, הפרוצדורה נרשמה בסיום התכנית, לאחר ההוראות שסיימו את ביצועה והחזירו את הבקרה למערכת ההפעלה. קוד הפרוצדורה תחום בין שתי הנחיות אסמבלר: initData PROC ו-initData ENDP. הפרוצדורה היא חלק מהתכנית כולה ובהתאם מאוחסנת גם היא במקטע הקוד. כאשר מריצים את התכנית שבדוגמה 7.1, ההוראה הראשונה שמתבצעת היא ההוראה שמזמנת את הפרוצדורה, כלומר ההוראה: call initData. לאחר ביצוע הוראה זו, מתבצעות ההוראות של גוף הפרוצדורה כלומר ההוראות:

```
    mov ax, @DATA
    mov ds, ax
    ret                         ; חזרה מפרוצדורה
```

לאחר ביצוע ההוראה השלישית, ההוראה ret, הביצוע ימשיך מההוראה העוקבת להוראה שזימנה את הפרוצדורה, כלומר מההוראה:

```
    mov ax, 4c00h
```

תכנית זו כוללת שתי הוראות חדשות:

א. ההוראה **CALL**

זימון הפרוצדורה מתבצע על-ידי הוראה מיוחדת

CALL אופרנד

שמשמעותה: עבור לביצוע הפרוצדורה עליה מצביע האופרנד, תוך שמירת כתובת ההוראה אליה יש לחזור לאחר סיום הפרוצדורה. כפי שנראה בהמשך (וזה מה שמבדיל הוראה זו מהוראת JMP), האופרנד מכיל את שם הפרוצדורה שהיא תווית המוצמדת להוראה

הראשונה בפרוצדורה. לאחר הידור וקישור, שם הפרוצדורה מוחלף בכתובת של ההוראה הראשונה בה.

ב. ההוראה **RET** (RETurn)

RET n

להוראה זו אין אופרנדים אבל היא יכולה לכלול מספר n (אופציונלי) שאליו נתייחס בהמשך. משמעות ההוראה היא: חזור להוראה העוקבת להוראה שקראה לפרוצדורה, והמשך משם את ביצוע התכנית. כדי לבצע פעולה זו, המעבד צריך לזכור את כתובת הוראת החזרה; לשם כך הוא משתמש במקטע המחסנית (בהמשך נדון בנושא זה).

אפשרות נוספת היא לרשום את הפרוצדורה בתחילת התכנית, לפני ההוראה הראשונה. לדוגמה:

```
.MODEL SMALL
.STACK 100h
.DATA
    a   DB  ?
    x   DW ?
.CODE
initData PROC                       ; הגדרת הפרוצדורה
    mov ax,   @DATA
    mov ds,   ax
    ret                             ; חזרה מפרוצדורה
initData ENDP
                                    ; תכנית ראשית
start:
    call  initData                  ; זימון פרוצדורה
                                    ; סיום התכנית
    mov ax,   4c00h
    int    21h
END start
```

בספר זה נעדיף לרשום את הפרוצדורות בסיום קטע הקוד של התכנית הראשית (המזמנת את הפרוצדורה). שימו לב, המעבד לא מבדיל בין קטע קוד של הפרוצדורה לבין קטע הקוד של התכנית הראשית; המתכנת הוא זה שצריך לדאוג להפרדה בין קטעים האלה. אם נכתוב את הפרוצדורה לאחר התווית שמציינת את תחילת התכנית (בדוגמה שלעיל זו התווית start), המעבד יתחיל לבצע את הפרוצדורה (החל מן ההוראה הראשונה) אבל כאשר יסיים את הפרוצדורה, ויגיע להוראה RET, הוא לא ימצא כתובת חזרה (משום שהפרוצדורה לא נקראה כלל), ולכן המעבד יקפוץ לכתובת אקראית במקטע הקוד.

שאלה 7.1

כתבו תכנית המשתמשת בפרוצדורה בשם terminateProg ותפקידה לסיים את התכנית ולהחזיר את הבקרה למערכת ההפעלה.

7.3 המחסנית ומצביע המחסנית

7.3.1 מבנה המחסנית

לניהול מנגנון החזרה מפרוצדורה, המעבד משתמש באזור מיוחד בזיכרון הראשי, שנקרא **מחסנית** (**Stack**). המחסנית ממוקמת במקטע מיוחד שהוקצה לה, מקטע זה נקרא "מקטע המחסנית" (Stack Segment או בקיצור SS). המחסנית מתוכננת כך שתוכל לאחסן לזמן קצר נתונים וכתובות. כדי להקצות מקום אחסון למחסנית אנו משתמשים בהנחיה:

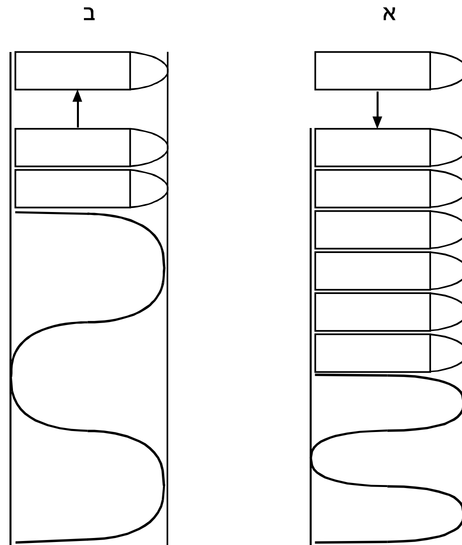
.STACK מספר בתים

לדוגמה, נגדיר מחסנית בגודל 256 בתים:

.STACK 100h

המחסנית מנוהלת בדרך הדומה מאוד לדרך שבה מנוהלת מחסנית תחמושת של רובה אוטומטי. הכדורים נטענים במחסנית תחמושת בזה אחר זה, ודוחפים זה את זה דרך הפתח הנמצא בראש המחסנית. פריקת הכדורים מן המחסנית נעשית גם היא דרך הפתח, אך סדר הוצאת הכדורים הפוך מסדר הכנסת הכדורים: זה שנכנס-אחרון-יוצא-ראשון (LIFO – Last-In-First-Out).

לאיור 7.1 שני חלקים המתארים דחיפה של כדור למחסנית ופריקת כדור ממנה.



א. דחיפת כדור למחסנית; ב. הוצאת כדור מן המחסנית

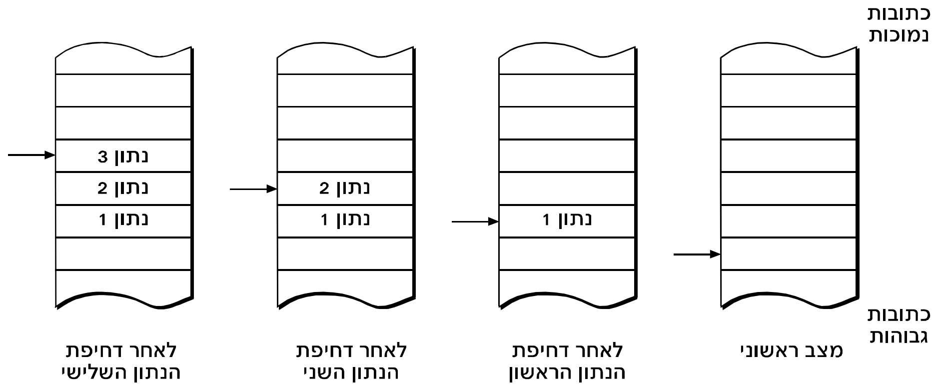
איור 7.1

כל דחיפה של כדור למחסנית התחמושת מזיזה את כל הכדורים שכבר נמצאים בה, ומפנה מקום לכדור החדש. פתח מחסנית התחמושת (ראש המחסנית) נשאר, כמובן, קבוע במקומו.

בדומה למחסנית התחמושת, גם המחסנית שבזיכרון המחשב מנוהלת בדרך המאפשרת להכניס ולהוציא נתונים בשיטת נכנס-אחרון-יוצא-ראשון, אולם הכנסת נתון למחסנית שבזיכרון **אינה** גורמת להזזת הנתונים הקודמים. בְּמקום זה, **ראש המחסנית** זז, כלומר המקום שאליו נכנס הנתון הבא משתנה, כדי שהנתון החדש שייכנס לא ייכתב על הנתון הקודם ויהרוס אותו. הכנסת נתון למחסנית נקראת **דחיפה** (Pushing), והוצאת נתון מן המחסנית נקראת **שליפה** (Popping). המעקב אחר מיקומו של ראש המחסנית, המשתנה עם כל דחיפה ושליפה, נעשה על-ידי האוגר **SP**, שנקרא **מצביע המחסנית**, ותפקידו להצביע בכל רגע נתון על הכתובת שאליה נדחף הנתון האחרון. SP מכיל כתובת יחסית לאוגר SS.

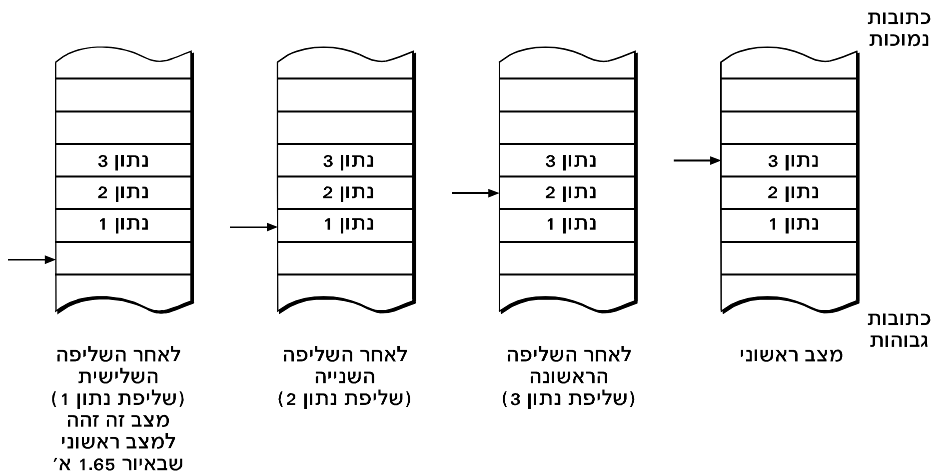
באיור 7.2 מתוארת דחיפה של שלושה נתונים בזה אחר זה למחסנית. החץ באיור מציין את הכתובת שעליה מצביע האוגר SP – כתובת ראש המחסנית. יש לשים לב לכך שערכו של

האוגר SP קטֵן עם כל דחיפה של נתון, והמחסנית 'צומחת' בכיוון הכתובות הנמוכות. במחסנית שבאיור האוגר SP מצביע על הכתובת שאליה נדחף הנתון האחרון.



איור 7.2
דחיפת שלושה נתונים למחסנית בזה אחר זה

באיור 7.3 מוצג תהליך הפוך: שליפת שלושה נתונים מן המחסנית בזה אחר זה. השליפה אינה הוצאה פיזית או מחיקה של הנתון מן המחסנית, אלא רק קריאה של נתון מן הכתובת שעליה מצביע האוגר SP. לאחר השליפה, האוגר SP יקודם ויצביע על כתובת גבוהה יותר, שממנה תבוצע השליפה הבאה. סדר השליפות הפוך מסדר הדחיפות, וכך נתון 3 יישלף בשליפה הראשונה, ונתון 1 יישלף בשליפה השלישית. לאחר השליפה השלישית, נמצא האוגר SP בדיוק באותו מצב שבו הוא היה לפני שדחפו נתונים למחסנית (ראו איור 7.2).



איור 7.3
שליפת שלושה נתונים מן המחסנית בזה אחר זה

שימו לב, כאשר משתמשים במחסנית, יש להימנע משליפת נתונים מכתובות שלא הוכנסו אליהן נתונים. כך, למשל, אם לאחר השליפה השלישית תבוצע שליפה נוספת, יישלף נתון נוסף שערכו אקראי, כיוון שלכתובת הזו במחסנית לא הוכנס כל נתון במהלך העבודה עם המחסנית. המצב הזה מכונה **גלישת מחסנית** (Stack overflow), שכן ראש המחסנית גולש בשליפה הזו מעבר למיקומו הראשוני, לאזור בזיכרון שאינו שייך למחסנית. בהגדרת המחסנית עלינו להקפיד להקצות מספיק מקום לסגמנט המחסנית כדי שיהיה אפשר לנהל את כל הנתונים הדרושים.

לסיכום, המעקב אחר מיקומו של ראש המחסנית נעשה באוגר SP: כל פעולת דחיפה מקטינה באופן אוטומטי את ערכו של האוגר SP, וכל פעולת שליפה מגדילה את הערך הזה. מבחינת ארגון הזיכרון של ה-8086, המחסנית נמצאת במקטע המחסנית, והאוגר SS מצביע על תחילת המקטע. האוגר SP מצביע על כתובת יחסית הנתונה ביחס לתחילת מקטע המחסנית (SS:SP).

שאלה 7.2

האם אפשר להגדיר מחסנית בגודל 1MB? נמקו את תשובתכם.

7.3.2 הוראות דחיפה של נתונים למחסנית ושליפתם ממנה

המעבד משתמש במחסנית לניהול מנגנון ביצוע הפרוצדורות; בנוסף אפשר להשתמש במחסנית כאמצעי אחסון זמני לנתונים שהגישה אליהם צריכה להיות מהירה יחסית. כך ניתן, במהלך הרצה של תכנית, להקצות מקום בזיכרון לנתונים נוספים, שלא הוגדרו במקטע הנתונים. בסעיף זה נציג שתי הוראות בסיסיות המאפשרות דחיפה של נתונים למחסנית ושליפתם ממנה, ואחר כך נסביר כיצד להשתמש בהוראות האלה בתכנית. אך תחילה נדגיש כי כל פעולות הדחיפה והשליפה במעבד 8086 נעשות על אופרנדים של 16 סיביות. אי-אפשר לדחוף או לשלוף בית אחד (8 סיביות).

א. ההוראה PUSH

ההוראה לדחיפת נתונים למחסנית היא

PUSH אופרנד

משמעותה: דחוף למחסנית את האופרנד ועדכן את תוכנו של SP (ביצוע ההוראה הזו מקטין ב-2 את תוכנו של SP). האופרנד יכול להיות הוא נתון מיידי, אוגר או משתנה, מטיפוס מילה.
להוראה זו אין השפעה על דגלים.

לדוגמה:

```
push  ax          ; ax  דחוף למחסנית את תוכן  אוגר
push  10          ; 10 דחוף למחסנית את הערך
push  var         ; var  דחוף למחסנית את תוכן המשתנה
push  al          ; הוראה שגויה!!! האופרנד אינו מטיפוס מילה
```

פעולת הדחיפה מבטיחה שהשיטה לאחסון מילים במחסנית תהיה עקבית ביחס לשיטה שבה מאוחסן נתון במקטע הנתונים. לכן, בפעולת הדחיפה, החלק התחתון של האופרנד מאוחסן בכתובת נמוכה יותר, וחלקו העליון – בכתובת גבוהה יותר.

דוגמה 7.2

הדוגמה שלפניכם ממחישה בעזרת ערכים מספריים את צורת האחסון של האופרנד במחסנית. הניחו כי:

AX = 3F4Ch

SP = 100h

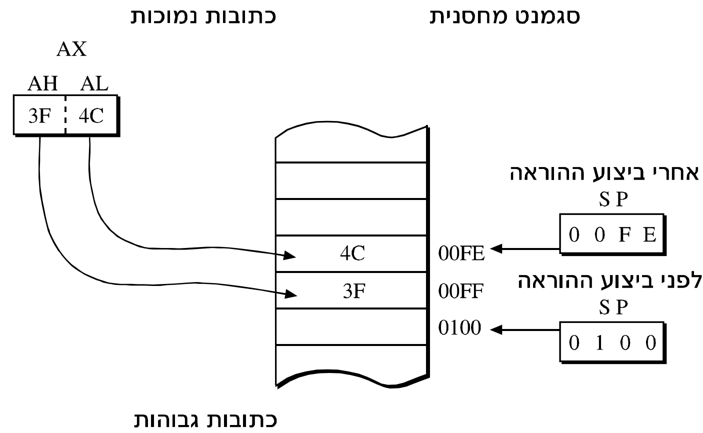
וההוראה הבאה שתבוצע היא:

push ax

בדקו כיצד האוגר SP והאוגר AX יושפעו מביצוע ההוראה.

פתרון

ביצוע ההוראה יגרום לדחיפת עותק של האוגר AX למחסנית, כפי שאפשר לראות באיור 7.4.



איור 7.4
דחיפת האוגר למחסנית

נעקוב אחר ביצוע ההוראה לפי השלבים 1-4, שיתוארו להלן:

1. הפחתת 1 מ-SP: $SP = 100h - 1 = FFh$
2. דחיפת AH = 3Fh לתוך הכתובת היחסית FFh במקטע המחסנית. חישוב הכתובת האפקטיבית של התא במחסנית אליו נדחף ערך זה, מתבצע על-ידי חיבור ערכו של SS עם ערכו של SP.
3. הפחתת 1 מ-SP: $SP = FFh - 1 = FEh$
4. דחיפת AL = 4Ch לתוך הכתובת היחסית FEh במקטע המחסנית. באופן דומה למתואר בשלב 2, חישוב הכתובת האפקטיבית של התא אליו נדחף ערך זה, מתבצע על-ידי חיבור ערכו של SS עם ערכו של SP.

ב. ההוראה POP

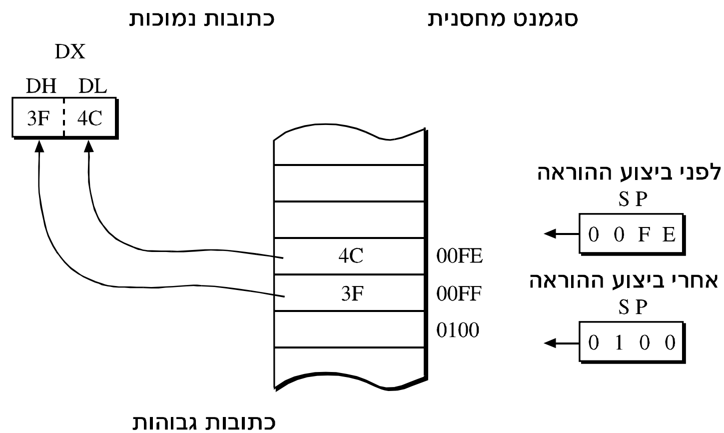
הפעולה ההפוכה לפעולת הדחיפה היא פעולת השליפה

POP אופרנד

משמעותה: שלוף מן המחסנית מילה, והעבר אותה לתוך האופרנד. ביצוע ההוראה הזו מגדיל ב-2 את תוכנו של SP (הכתובת היחסית). האופרנד יכול להיות אוגר או משתנה מטיפוס מילה.

להוראה זו אין השפעה על הדגלים.

באיור 7.5 מודגם התהליך של שליפת איבר ממחסנית.



איור 7.5
ביצוע ההוראה pop dx

נעקוב אחר שלבי הביצוע של הוראת השליפה (אחרי שדחפנו את הערך 3F4Ch):

1. שליפת הבית מן הכתובת היחסית FEh, והעברתו לתוך הבית התחתון של DX:
   DL = 4Ch
2. הוספת 1 ל-SP: SP = FEh + FFh
3. שליפת הבית מן הכתובת היחסית FFh, והעברתו לתוך הבית העליון של DX:
   DH = 3Fh
4. הוספת 1 ל-SP: SP = FFh + 1 = 100h

שימו לב, ההוראה אינה משנה את התוכן של כתובת כלשהי בזיכרון, ואינה מוחקת את הנתון שנשלף מן המחסנית. הנתון הזה יוחלף רק כאשר נתון חדש יידחף למחסנית באותה כתובת. בהמשך פרק זה נמשיך להתייחס לכתובת היחסית במחסנית עליה מצביע האוגר SP, אך נזכור שלחישוב הכתובת האפקטיבית של איבר במחסנית יש לחבר את ערכו של SP עם ערכו של SS.

שאלה 7.3

א. עקבו אחר ביצוע התכנית הרשומה להלן. רשמו את תוכן המחסנית ותוכן אוגר SP בכל הוראה ואת ערכם של a ו-b בסיום התכנית.

```
.MODEL SMALL
.STACK  100h
.DATA
    a DW 1234h
    b DW 5678h
.CODE
start:
        mov  ax,   @DATA
        mov  ds,   ax
        push  a
        push  b
        lea     bx, a
        pop   [bx]
        pop   [bx+2]
                                                  ; סיום תכנית
        mov  ax,   4c00h
        int     21h
END start
```

ב. השתמשו בעקרון המודגם בסעיף א וכתבו תכנית שתהפוך את המחרוזת str1 בת 4 תווים מטיפוס מילה.

שאלה 7.4

כתבו קטע של תכנית שיחליף בין ערכיהם של האוגרים AX, BX, CX, DX כך שערכו של AX יוחלף עם BX, ערכו של CX יוחלף עם DX. השתמשו במחסנית לביצוע הפעולות.

7.3.3 שימוש במחסנית למנגנון ביצוע פרוצדורות

כדי לחזור בצורה תקינה מן הפרוצדורה לתכנית שקראה לה, חייב המעבד לשמור את תוכן האוגר IP לפני הקריאה, ולשחזר אותו בעת החזרה מהפרוצדורה. שמירה זו של אוגר IP ושחזורו נעשים באמצעות דחיפת נתון למחסנית (בזמן קריאה לפרוצדורה) ושליפתו ממנה (בזמן ביצוע הוראת החזרה). הכתובת הנשמרת במחסנית נקראת "**כתובת החזרה**" (Return address). ב-8086 קיימים שני סוגים של קריאות לפרוצדורה: קריאה לפרוצדורה קרובה (Near routine) וקריאה לפרוצדורה רחוקה (Far routine).

"**פרוצדורה קרובה**" היא פרוצדורה המצויה במקטע הקוד הנוכחי. בתוך המקטע הזה אין הגבלה על מיקומה של הפרוצדורה או על מרחק ההתחלה שלה מן ה-CS:IP, לכן, בקפיצה לפרוצדורה, צריך ה-8086 לשנות רק את ערכו של האוגר IP, ולקבוע בו את הכתובת היחסית של הפרוצדורה. כדי שיוכל לחזור אל התכנית בסיום ביצוע הפרוצדורה, יש לשמור את כתובת החזרה, כלומר את הערך של האוגר IP.

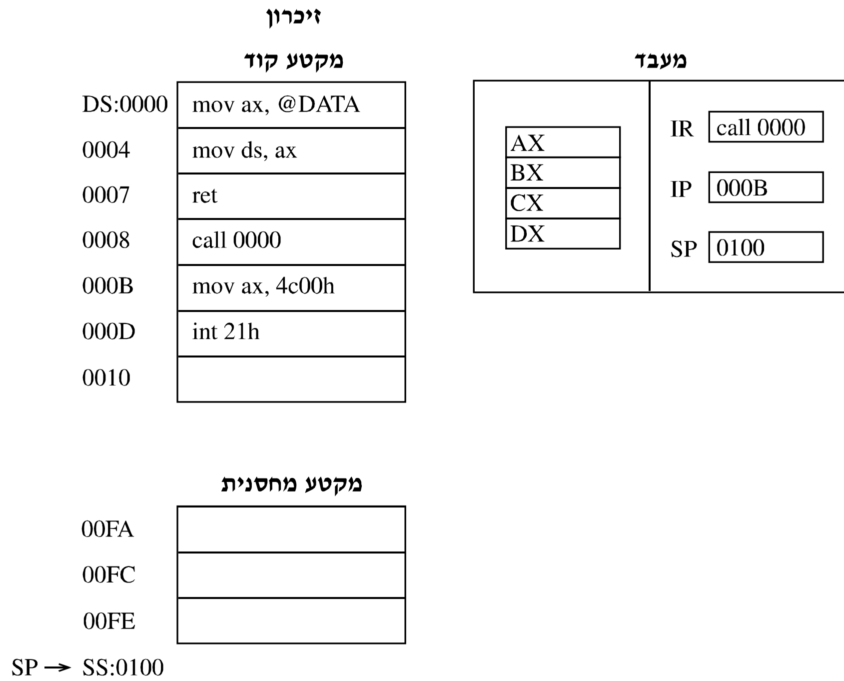
"**פרוצדורה רחוקה**" היא פרוצדורה הנמצאת במקטע קוד אחר, לכן, כאשר קוראים לפרוצדורה רחוקה, יש לשמור במחסנית שתי כתובות: את תוכן אוגר מקטע הקוד CS (שמכיל את כתובת מקטע הקוד) ואת תוכן אוגר IP (הכתובת אליה יש לחזור, יחסית לתחילת מקטע הקוד), ולאחר מכן צריך לקבוע את הכתובות החדשות שבאוגר מקטע הקוד ואוגר IP בהתאם לכתובת מקטע הקוד החדש ומיקום הפרוצדורה בו. בנוסף, יש לעדן את כתובת אוגר CS ואוגר IP בזמן החזרה מהפרוצדורה, ולהתאימן לכתובת של ההוראה הבאה בתכנית, אחרי ההוראה שזימנה את הפרוצדורה. בספר זה נציג רק את השימוש בפרוצדורה קרובה.

דוגמה 7.3

נדגים כיצד אפשר לבצע את התכנית שהוצגה בדוגמה 7.1 תוך כדי שימוש במחסנית לניהול מנגנון הזימון והחזרה לפרוצדורה וממנה; לשם כך נשתמש במודל לוגי שבו נציג את מצב הזיכרון ומצב האוגרים המושפעים מביצוע התכנית. באמצעות מודל זה נציג שלושה שלבים לפיהם תבוצע התכנית: לאחר טעינת התכנית, לאחר הקריאה לפרוצדורה ולאחר סיום הפרוצדורה.

שלב א   לאחר טעינת התכנית לזיכרון

באיור 7.6 אפשר לראות את מצב הזיכרון לאחר טעינת התכנית לזיכרון. במצב זה אוגר IP מכיל את כתובת ההוראה הראשונה, שהיא ההוראה המזמנת את הפרוצדורה.



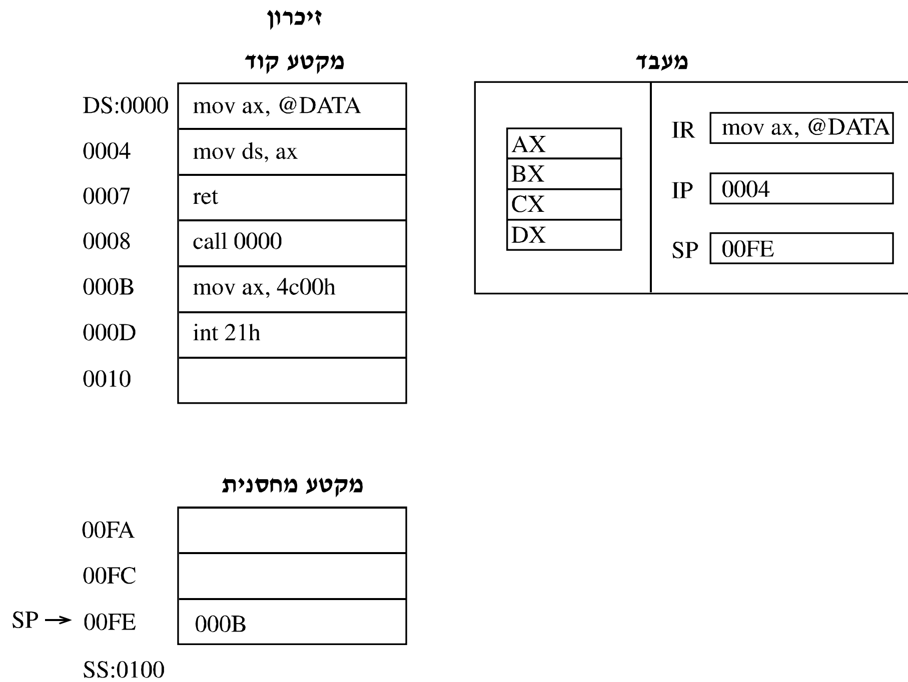
איור 7.6
מצב הזיכרון לאחר טעינת התכנית

שלב ב   לאחר הקריאה לפרוצדורה

לאחר קריאה לפרוצדורה זו מתבצעות הפעולות האלה :

- ערכו של אוגר SP קטן ב-2.
- כתובת החזרה (שהיא כתובת ההוראה הבאה לביצוע השמורה ב-IP) מוכנסת למחסנית בכתובת SS:SP.
- כתובת IP מתעדכנת ומצביעה על ההוראה הראשונה לביצוע בפרוצדורה.

התוצאה של ביצוע הזימון לפרוצדורה מוצגת באיור 7.7.



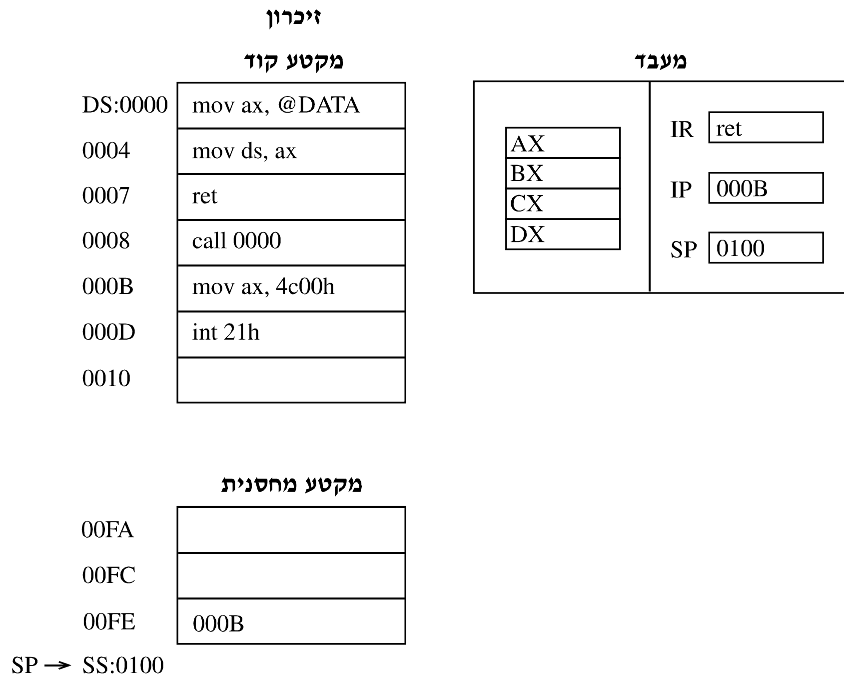
איור 7.7
מצב הזיכרון לאחר ביצוע הקריאה לפרוצדורה: call initData

שלב ג חזרה מפרוצדורה

לאחר ביצוע ההוראות שמכיל גוף הפרוצדורה, נקראת לביצוע ההוראה RET הרשומה בסופה של הפרוצדורה. בעת ביצוע הוראה זו מתבצעות הפעולות האלה:

- הכתובת השמורה במחסנית בכתובת SS:SP מועתקת לאוגר IP.
- ערכו של אוגר SP גדל ב-2.

ובהתאם, מצב הזיכרון ומצב האוגרים, לאחר ביצוע הוראת החזרה, מוצגים באיור 7.8.



איור 7.8
מצב הזיכרון לאחר ההוראה ret (הוראת החזרה מהפרוצדורה)

שאלה 7.5

השתמשו באיור המציג את מצב הזיכרון והאוגרים והסבירו בעזרתו מה היה קורה אילו הפרוצדורה initData לא היתה כוללת את הוראת החזרה RET?

7.4 העברת פרמטרים

בשפת אסמבלי אפשר לפנות אל משתני התכנית גם בהוראות הפרוצדורה וכך להתייחס אליהם כאל משתנים גלובליים. קיימות שתי שיטות למימוש מנגנון העברת הפרמטרים בהן משתמשים בשפת אסמבלי: שימוש באוגרים כלליים ושימוש במחסנית. באופן דומה למשתנים גלובליים, הגישה לאוגרים אפשרית מכל הוראה בתכנית, בין אם היא הוראה בפרוצדורה או הוראה בתכנית הראשית. אולם השימוש באוגרים מוגבל משום שמספרם קטן וגם התכנית הראשית משתמשת בהם. לעומת זאת, השימוש במחסנית מאפשר לנו

לממש מנגנון להעברת פרמטרים אל הפרוצדורה שמחזור החיים שלהם הוא כמשך מחזור חיי הפרוצדורה (או הפונקציה). לפרמטרים אלה מוקצה מקום בזיכרון כאשר הפרוצדורה מזומנת על-ידי התכנית הראשית; שטח זיכרון זה משוחרר כאשר ביצוע הפרוצדורה מסתיים, וניתן להשתמש בו שוב. לעומת זאת, משך החיים של משתני התכנית הוא כמשך התכנית כולה, לכן שטח הזיכרון מוקצה להם במקטע הנתונים בזמן טעינת התכנית לזיכרון. בסעיף זה נתאר כיצד משתמשים במחסנית כדי לממש את המנגנון להעברת פרמטרים משני סוגים (באותן שיטות משתמשים גם בשפות עיליות):

- **פרמטר לפי ערך** (by value) – פרמטר שאת ערכו הפרוצדורה אינה משנה;
- **פרמטר לפי אזכור** (by reference) – פרמטר שהפרוצדורה יכולה לשנות את ערכו.

כאשר משתמשים במחסנית להעברת פרמטרים, דוחפים אליה את הפרמטרים לפני הזימון לפרוצדורה, ובסיומה דואגים לשחרר את המקום שהוקצה לפרמטרים. בדרך זו "מחזור החיים" של הפרמטרים חופף ל"מחזור החיים" של הפרוצדורה. בשיטה זו, כדי לממש פרמטר לפי ערך, נדחוף למחסנית את הנתונים שיש להעביר לפרוצדורה, וכדי לממש פרמטר לפי אזכור, נדחוף למחסנית את הכתובות של משתנים בהם תשתמש הפרוצדורה. נסביר כיצד משתמשים בכל אחת מהשיטות כאשר כותבים תכנית שמשתמשת בפרוצדורה swap, להחלפת תוכנם של משתנים.

7.4.1 שיטה למימוש פרמטר לפי ערך (by value)

כפי שציינו, לפני הזימון לפרוצדורה דוחפים בשיטה זו למחסנית את הערכים של הפרמטרים ולאחר שביצוע הפרוצדורה מסתיים "מנקים" את הזיכרון של המחסנית ומחזירים את ערכו של אוגר SP למצבו ההתחלתי.

דוגמה 7.4

בדוגמה זו נציג נסיון שגוי לשימוש בפרוצדורה *שמטרתה* להחליף בין שני ערכי פרמטרים המועברים אליה לפי ערך. בדוגמה זו נתאר את תהליך הביצוע של הפרוצדורה ונבין את מהות השגיאה. לשם כך, נכתוב תכנית שתזמן את הפרוצדורה swap(a,b) המחליפה בין ערכי שני פרמטרים ומממשת העברת פרמטרים לפי ערך. לפני זימון הפרוצדורה עלינו לדחוף למחסנית את הנתונים שרשומים במשתנים a ו-b. להלן קטע התכנית הראשית המזמנת את הפרוצדורה:

```
.MODEL SMALL
.STACK   100h
.DATA
    a  DW   12h
    b  DW  A9h
.CODE
 start:
      mov  ax,  @DATA
      mov  ds,  ax
      push  a                              ; a דוחפים למחסנית את ערך של
      push  b                              ; b דוחפים למחסנית את ערך של
      call   swap                          ; swap(a,b) זימון הפרוצדורה
      mov  ax,   4c00h                     ; סיום התכנית
      int    21h

      swap PROC                            ; הגדרת הפרוצדורה
                                           ; גוף הפרוצדורה (יפורט בהמשך)
      swap ENDP
END start
```

איור 7.9 מציג את מקטע הנתונים וקטע ממקטע המחסנית, לאחר זימון הפרוצדורה (ביצוע ההוראה call swap), אך לפני ביצוע הפרוצדורה עצמה:

מקטע נתונים

DS:0000	0012
0002	00A9

מקטע מחסנית

SP → 00FA	כתובת החזרה
00FC	00A9
00FE	0012
SS:0000	

איור 7.9

מצב מקטע הנתונים והמחסנית לאחר זימון הפרוצדורה swap, אך לפני תחילת ביצועה

באיור 7.9 אפשר לראות כי לאחר זימון הפרוצדורה, יצביע האוגר SP על ראש המחסנית (התא שמכיל את כתובת החזרה), וכי במחסנית שמור עותק של הנתונים שנשלחו לפרוצדורה. כיצד אם כך ניתן לפנות לפרמטרים שנשלחו לפרוצדורה? מקובל להשתמש באוגר BP (שהוא אוגר בן 16 סיביות שמצביע על היסט במקטע המחסנית), כדי לגשת לפרמטרים השמורים במחסנית. אך תחילה עלינו לאתחל את ערכו של BP לערכו של SP, ולכן בתחילת הפרוצדורה נרשום את ההשמה:

```
MOV   BP, SP
```

כעת נשתמש בשיטת מיעון בסיס כדי לחשב את ההעתק של הפרמטר מראש המחסנית; כתוצאה, יחולו השינויים האלה, לפי סדר הכנסתם של הפרמטרים למחסנית:

- תוכנו של המשתנה a הועבר לכתובת BP+4
- תוכנו של המשתנה b הועבר לכתובת BP+2

בהתאם, נכתוב את הפרוצדורה swap:

```
swap PROC
    mov bp, sp              ; BP = SP
    mov ax, [bp+2]          ; AX = 00A9h
    xchg ax, [bp+4]         ; AX = 0012h, [BP+2] = 00A9h
    mov [bp+2], ax          ; [BP+2] = 0012h
    ret  4
swap ENDP
```

בסיום הפרוצדורה, עלינו להחזיר את המחסנית למצבה המקורי, כלומר לאחר שתישלף כתובת החזרה, עדיין יהיו במחסנית שני הפרמטרים שהכנסנו לתוכה לפני הזימון לפרוצדורה. כדי להוציא את הפרמטרים מהמחסנית נשתמש בהוראת החזרה

```
RET 4
```

הוראה זו מקדמת את ערכו של SP ב-4 וכך נשלפת תחילה כתובת החזרה (עליה מצביע SP) ובהתאם ערכו של SP גדל ב-2. לאחר מכן גדל ערכו של SP ב-4 וכך הוא חוזר למצב ההתחלתי, שבו הוא הצביע על הכתובת 0100h, והקצאת הזיכרון עבור הפרמטרים של הפרוצדורה מבוטלת.

שאלה 7.6

א. כאשר מחשבים את הכתובת של פרמטר a (לדוגמה), הסבירו מדוע מחברים bp+2 ולא מחסרים bp-2?

ב. בעקבות ביצוע ההוראה RET 4, האם הערכים שהיו במחסנית נמחקו? מתי ערכים אלה נמחקים?

ג. הסבירו מה יקרה אם נשנה את הוראת החזרה מהפרוצדורה ונרשום את ההוראה RET.

לסיכום, בתכנית זו העברנו לפרוצדורה פרמטרים לפי ערך, כלומר העתקנו למחסנית את הנתונים ששמורים במשתנים a ו-b וביצענו את ההחלפה בין הנתונים במחסנית ולא בין תאי הזיכרון שהוקצו למשתנים אלה. בסיום הפרוצדורה אנו "משחררים" במחסנית את הזיכרון שהוקצה לפרמטרים שהועברו, ומרגע זה לא ניתן "לגשת" אליהם, לעומת זאת, ערכם של המשתנים a ו-b לא השתנה כלל. כדי לבצע משימה בהצלחה נממש העברת פרמטרים לפי אזכור.

7.4.2 שימוש במחסנית להעברת פרמטרים לפי אזכור (by reference)

כדי לממש העברת פרמטרים לפי אזכור, יש לשמור במחסנית את כתובות המשתנים ולא את הערכים עצמם.

דוגמה 7.5

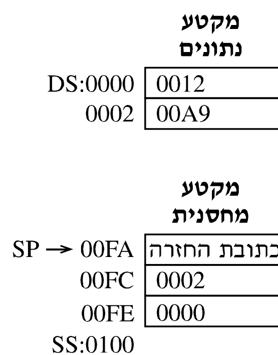
עלינו לכתוב תכנית שתזמן את הפרוצדורה swapByRef (a,b) המחליפה בין ערכי שני פרמטרים ומממשת העברת פרמטרים לפי אזכור. בשיטה זו, לפני הזימון של הפרוצדורה עלינו לדחוף למחסנית את הכתובות של המשתנים a ו-b. להלן קטע התכנית הראשית המזמנת את הפרוצדורה:

```
.MODEL SMALL
.STACK  100h
.DATA
   a DW  12h
   b DW  A9h
.CODE
```

```
start:
    mov  ax,  @DATA
    mov  ds,  ax
    lea   bx,  a
    push  bx                          ; a דוחפים למחסנית את כתובת המשתנה
    lea   bx,  b
    push  bx                          ; b דוחפים למחסנית את כתובת המשתנה
    call swapByRef                    ; swapByRef (a,b) זימון הפרוצדורה
                                      ; סיום התכנית
    mov  ax,  4c00h
    int   21h
                                      ; הגדרת הפרוצדורה
    swapByRef PROC
                                      ; גוף הפרוצדורה (יפורט בהמשך)

    swapByRef ENDP
END start
```

באיור 7.10 מוצג מצבו של קטע מהמחסנית, לאחר זימון הפרוצדורה:



איור 7.10
מצב מקטע הנתונים והמחסנית לאחר זימון פרוצדורה swapByRef

כדי להתייחס לפרמטרים ולפנות לכתובות המשתנים שנשמרו במחסנית, נשתמש בשיטת מיעון בסיס; באוגר BP נשתמש להצבעה על מקטע הנתונים, ובאוגרים SP ו-BP נשתמש כדי להצביע על מקטע המחסנית. נרשום את ההוראות בגוף הפרוצדורה swapByRef:

```
swapByRef PROC
    mov  bp,  sp            ; BP מצביע על ראש המחסנית
    mov  bx,  [bp+2]        ; BX מצביע על כתובתו של b
    mov  ax,  [bx]          ; ax = 00A9h
    mov  si,  [bp+4]        ; SI מצביע על כתובתו של a
    xchg ax,  [si]          ; AX = 0012h,  a = 00A9h
    mov  [bx], ax           ; b = 0012h
    ret 4
swapByRef ENDP
```

המחסנית מכילה כתובות של המשתנים a ו-b, ואנו משתמשים בשיטת מיעון בסיס כדי לשנות את הנתונים במקטע הנתונים. כדי לעקוב אחר מהלך ביצוע הפרוצדורה, נשתמש בטבלת המעקב הזו:

טבלה 7.1

טבלת מעקב אחר ביצוע הפרוצדורה swapByRef

	ההוראה	מצב האוגרים					מקטע הנתונים	
מס.	**שמתבצעת**	sp	bp	bx	si	ax	DS:[0000]	DS:[0002]
1	mov bp, sp	00FA	00FA				0012	00A9
2	mov bx, [bp+2]			0002				
3	mov ax, [bx]					00A9		
4	mov si, [bp+4]				0000			
5	xchg ax, [si]					0012	00A9	
6	mov [bx], ax							0012
7	ret 4							

התבוננות בטבלה מלמדת כי בהוראה 2, הכתובת 0002 מועתקת אל האוגר bx והוא מצביע על המשתנה b. בהוראה 3 מועתק תוכן המשתנה b, עליו מצביע bx, אל האוגר ax. בהוראה 4 משתמשים באוגר SI כדי להצביע על המשתנה a, ובעזרתו מחליפים את תוכן האוגר ax עם תוכן המשתנה a. ההוראה השישית מעדכנת את תוכנו של המשתנה b, וכך בסיום הפרוצדורה מוחלפים תוכניהם של המשתנים a ו-b. איור 7.11 מציג את המצב של מקטע הנתונים ושל המחסנית לאחר סיום הפרוצדורה.

	מקטע נתונים
DS:0000	00A9
0002	0012

	מקטע מחסנית
SP → 00FA	כתובת החזרה
00FC	0002
00FE	0000
SS:0100	

איור 7.11
מצב מקטע הנתונים והמחסנית לאחר סיום הפרוצדורה swapByRef

שאלה 7.6

א. נניח שהמשתנים a ו-b, שכתבנו בדוגמה 6.20, הם מטיפוס בית, האם צריך להכניס שינויים בתכנית? אם לא – הסבירו מדוע, ואם כן – בצעו את השינויים הדרושים.

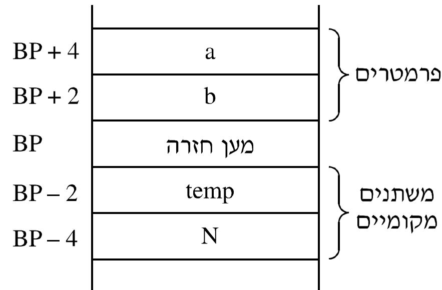
ב. השתמשו בפרוצדורה swapByRef כדי לכתוב תכנית שתמיין שלושה משתנים a, b, c מטיפוס מילה, בסדר עולה.

7.5 מימוש משתנים מקומיים

משתנים מקומיים הם משתנים שהפרוצדורה מגדירה, לכן הם אינם מוכרים מחוץ לה. משך "מחזור החיים" של משתנים מקומיים הוא כמשך מחזור חיי הפרוצדורה עצמה. אך בניגוד לפרמטרים שמועברים לפרוצדורה, הקצאת זיכרון למשתנים מקומיים ושחרורו מתבצעת בפרוצדורה עצמה.

כדי לממש מנגנון זה אנו משתמשים בהוראות PUSH ו-POP הנרשמות בגוף הפרוצדורה. בהתאם לכך, מיקומם היחסי של משתנים מקומיים במחסנית הוא מתחת לכתובת החזרה

(כלומר בכתובות נמוכות יותר). נקבע כמוסכמה כי BP (וגם SP) תמיד מצביע על כתובת החזרה. לכן תוקצה למשתנה המקומי הראשון (שהוא מטיפוס מילה) הכתובת SS:[BP-2], למשתנה המקומי השני תוקצה הכתובת SS:[BP-4], וכך הלאה.



איור 7.12
מצב מחסנית הכולל פרמטרים ומשתנים מקומיים

באופן כללי: המיקום של פרמטרים מקומיים הוא בכתובות [bp-n] ואילו מיקום הפרמטרים המועברים אל הפרוצדורה הוא בכתובות [bp+n] (כאשר n=0,2,4,6,...).

דוגמה 7.6

נכתוב תכנית שתסכם את n האיברים הראשונים בסדרת המספרים 1,3,5,7... . התכנית תשתמש בפרוצדורה sumNum המסכמת n איברים בסדרה. הפרוצדורה תקבל את הפרמטרים האלה: n כפרמטר לפי ערך ו-sum כפרמטר לפי כתובת; הפרוצדורה תשתמש בפרמטר מקומי בעזרתו מחושב הסכום, ובסיום יועתק ערך זה אל המשתנה sum.

בתכנית מוגדרים שני משתנים:

- משתנה n מטיפוס מילה המכיל את מספר האיברים בסדרה
- משתנה sum מטיפוס מילה שבו יישמר סכום הסדרה.

תחילה נכתוב אלגוריתם המתאר את הפעולות העיקריות בתכנית:

דחוף למחסנית את ערכו של n
דחוף למחסנית את כתובתו של sum
קרא לפרוצדורה sumNum
סיים תכנית.

הפרוצדורה sumNum

{טענת כניסה: הפרוצדורה מקבלת את מספר האיברים בסדרה n }

{טענת יציאה: הפרוצדורה מחזירה את הסכום של איברי הסדרה: ...,1,3,5,7 }

הפרוצדורה משתמשת באוגר AX לחישוב איבר ובאוגר CX כמונה לולאה. כמו-כן מגדירה הפרוצדורה משתנה מקומי temp המשמש לחישוב סכום האיברים. להלן האלגוריתם של הפרוצדורה:

bp = sp
אַפס אוגר cx = 0
הקצה מקום למשתנה מקומי temp: דחוף cx למחסנית
cx = n
ax = 1
כל עוד cx≠0 בצע:
    temp = temp + ax
    ax = ax + 2
    cx = cx – 1
שלוף את סכום הסדרה ax = [bp–2]
אחסן תוצאה במשתנה sum: [bp+2] = ax

הערה, באלגוריתם זה ניתן לסכם את הסדרה ישירות למשתנה sum

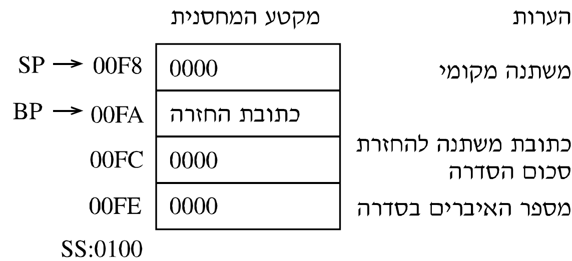
להלן התכנית המלאה:

```
.MODEL SMALL
.STACK 100h
.DATA
   n      DW 10
   sum    DW 0
.CODE
; תכנית ראשית
start:
 mov  ax,   @DATA
```

```
  mov  ds,   ax
                                                ; דחיפת פרמטרים למחסנית
  push n                                        ; העברת מספר איברים כפרמטר לפי ערך
  lea   bx,  sum
  push bx                                       ; העברת פרמטר לפי אזכור שבו יישמר סכום הסדרה
  call  sumNum                                  ; זימון הפרוצדורה
  mov  ax,  4c00h                               ; סיום התכנית
  int    21h
                                                ; הגדרת הפרוצדורה
sumNum PROC
   mov  bp,  sp                                 ; bp = sp
   xor    cx,  cx                               ; cx = 0
   push  cx                                     ; [BP–2] הגדרת משתנה עזר temp שכתובתו
   mov  cx,  [bp+4]                             ; cx = n  אתחול מונה לולאה
   mov  ax,  1                                  ; ax = 1  אתחול איבר ראשון בסדרה
again:
   add   [bp–2],  ax                            ; temp = temp + ax
   inc    ax
   inc    ax                                    ;ax= ax+2 חישוב האיבר הבא
   loop  again                                  ; cx>0  חזור על ביצוע לולאה כל עוד
                                                ; סיום הלולאה לחיבור
   pop   ax                                     ;   ax = temp
   mov  [bp+2],  ax                             ; sum = ax  אחסון סכום סדרה בזיכרון
   ret    4
ENDP sumNum

END start
```

איור 7.13 מתאר את מצב המחסנית לאחר זימון הפרוצדורה sumNum ולאחר הקצאת מקום למשתנה המקומי (לאחר ההוראה push cx). מהאיור אנו למדים כי SP מצביע על ראש המחסנית, בעוד ש-BP מצביע על כתובת החזרה.



איור 7.13
מצב מחסנית לאחר הקצאת מקום למשתנה מקומי

במהלך ביצוע ההוראות של הפרוצדורה, כל פנייה לכתובת [bp-2] מתייחסת למשתנה העזר.

בסיום הפרוצדורה עלינו לדאוג ש-SP יצביע שוב על תחילת המחסנית, כלומר על הכתובת 100h. לכן עלינו לשלוף תחילה את תוכן משתנה העזר, באמצעות ההוראה POP ax, ולאחסן אותו בכתובת עליה מצביע המצביע [bp+2]. כתוצאה גדל ערכו של מצביע המחסנית ב-2 והוא חזר למצב שבו sp=bp. כעת נפעל לפי הנוהל לפיו פעלנו קודם – נשתמש בהוראת החזרה ret 4, שתגדיל את ערכו של SP לערך הכתובת המצביעה על ראש המחסנית.

שאלה 7.7

כתבו תכנית שתסכם את n האיברים הראשונים בסדרת פיבונצי. בסדרת פיבונצי ערכם של שני האיברים הראשונים הוא 1 וכל איבר אחר הוא הסכום של שני האיברים שקדמו לו, .

שימו לב, אנו משתמשים באופרנדים בהוראות התכנית הראשית וגם בהוראות הפרוצדורה. כדי למנוע מצב שבו ערכי האוגרים עלולים להשתנות בעקבות ביצוע הפרוצדורה, נהוג לשמרם במחסנית לפני העברת הפרמטרים ולפני הזימון לפרוצדורה. אחרי ביצוע הפרוצדורה, אנו "משחזרים" את ערכי האוגרים ושולפים אותם מהמחסנית.

דוגמה 7.7

לדוגמה, נשנה את קטע התכנית שבדוגמה 7.6 ונרשום הוראות השומרות את ערכי האוגרים ax ו-cx (ערכים בהם אנו משתמשים בפרוצדורה) לפני העברת הפרמטרים. השינויים שיש להכניס בתכנית הם:

```
start:
    mov  ax,  @DATA
    mov  ds,  ax
                                          ; שמירת תוכנם של האוגרים ax ו-cx
    push  ax
    push  cx
                                          ; דחיפת פרמטרים למחסנית
    push  n                               ; העברת מספר איברים כפרמטר לפי ערך
    lea   bx,  sum
    push  bx                              ; העברת פרמטר לפי אזכור שבו יישמר סכום הסדרה
    call  sumNum                          ; זימון הפרוצדורה
                                          ; שחזור תוכנם של האוגרים ax ו-cx
    pop   cx
    pop   ax
                                          ; סיום התכנית
```

שאלה 7.8

הסבירו מה יקרה אם נשנה את סדר ההוראות באמצעותן משחזרים את תוכן האוגרים ונרשום:

```
    pop   ax
    pop   cx
```

7.6 העברת מערך כפרמטר לפרוצדורה

העברה של מערכים לפרוצדורה יכולה להתבצע לפי אזכור או לפי ערך; לפי אזכור דוחפים למחסנית את כתובת תחילת המערך ואת מספר האיברים (או סמן אחר המציין את סוף המערך); לפי ערך דוחפים למחסנית את כל איברי המערך לפני זימון הפרוצדורה. ההעברה של מערך כפרמטר לפי ערך, יוצרת עותק נוסף של המערך במחסנית, לכן עלינו לדאוג שהמחסנית תהיה בגודל מתאים. מבחינת הזיכרון ניתן לומר שהכפילות יוצרת "בזבוז" של מקום בזיכרון, אולם יש לה יתרון ברור כאשר רוצים להגן על תוכן המערך מפני כתיבה. השיטה להעברת מערך (או מחרוזת) כפרמטר לפי אזכור, לא רק חוסכת זיכרון, אלא מבטיחה גם שכל שינוי איבר מתבצע במערך המקורי וכשהפרוצדורה מסתיימת השינויים נשמרים.

דוגמה 7.8

התכנית שנביא להלן מדגימה כיצד מועבר מערך כפרמטר לפי אזכור, והיא מתאימה גם להעברת מחרוזת או רשומה כפרמטר לפי אזכור.

נכתוב תכנית שתזמן פרוצדורה המסכמת איברים במערך, החל מהאיבר ה-i עד האיבר האחרון (n). הפרוצדורה מחזירת את התוצאה במשתנה sumarr.

בתכנית מוגדרים המשתנים האלה:

- מערך a מטיפוס מילה
- משתנה n מטיפוס מילה, המכיל את מספר האיברים במערך
- משתנה i מטיפוס מילה, שהוא אינדקס של איבר במערך
- משתנה sumarr מטיפוס מילה, שיכיל את סכום האיברים

התכנית תשתמש בפרוצדורה sum המסכמת ומחזירה את סכום האיברים במערך a, החל מהאיבר ה-i ועד סוף המערך. הפרוצדורה תקבל כפרמטרים את הנתונים האלה:

- n ו-i כפרמטר לפי ערך
- כתובת המערך a וכתובת משתנה sumarr כפרמטר לפי אזכור
- הפרוצדורה תחזיר את סכום הסדרה שחושבה.

תחילה נכתוב אלגוריתם שיתאר את התכנית הראשית:

דחוף למחסנית את כתובת האיבר הראשון במערך a
דחוף למחסנית את ערכו של n
דחוף למחסנית את ערכו של i
דחוף למחסנית את הכתובת של sumarr (סכום האיברים)
קרא לפרוצדורה sum
סיים תכנית

הפרוצדורה sum

במימוש הפרוצדורה עלינו לזכור כי מיקום האיבר הראשון במערך הוא 0, לכן המיקום הפיזי של האיבר ה-i במערך הוא i–1. כדי לפנות לאיבר במערך אנו משתמשים בשיטת מיעון אינדקס-בסיס, כאשר BX מכיל את כתובת התחלת המערך ו-SI מצביע לאיבר.

{טענת כניסה: הפרוצדורה מקבלת מצביע למערך a, n הוא מספר אברי המערך, ו-i הוא המיקום של איבר במערך}

{טענת יציאה: הפרוצדורה מחזירה את סכום איברי המערך מהאיבר ה-i ועד סוף המערך}

BP= SP
העתק את n לאוגר CX
העתק לאוגר BX את כתובת התחלת המערך a
העתק את i–1 לאוגר SI
אפס את אוגר AX
כל עוד CX שונה מ-0 בצע:
  חבר את a[SI] לאוגר AX
  קדם את SI ב-1
  אחסן את AX במשתנה sumarr

להלן התכנית:

```
.MODEL SMALL
.STACK 100h

MAX EQU 9
```

```
.DATA
    a  DB   9, 0Ah, 4, 5, 6, 8, 1, 4, 12
    n  DW MAX
    i   DW 04h
    sumarr DW ?
.CODE

start:
    mov  ax,  @DATA
    mov  ds,  ax
                                        ; העברת פרמטרים למחסנית
    lea    bx,  a
    push  bx                            ; כתובת תחילת המערך, לפי אזכור
    push  n                             ; מספר האיברים במערך, לפי ערך
    push  i                             ; מספרו הסידורי של האיבר הראשון שיש לסכם  לפי ערך
    lea     bx,  sumarr                 ; סכום חלק מאיברי מערך, לפי אזכור
    push  bx
    call    sum                         ; זימון הפרוצדורה
    mov  ax,                            ; סיום התכנית
    int     21h
                                        ; הגדרת הפרוצדורה
sum PROC
      mov  bp,  sp
      mov  cx,  [bp+6]                  ; מספר איברי המערך
      mov  si,   [bp+4]                 ; מספרו הסידורי של האיבר הראשון לחיבור
      sub   cx,  si                     ; חישוב מספר האיברים שיש לחבר
      inc    cx
      dec    si
    mov  bx,  [bp+8]                    ; כתובת תחילת המערך
      xor    ax,  ax                    ; אתחול מסכם לערך 0
```

```
again: add ax, [bx+si]
       inc si
       loop again
   mov  bx,  [bp+2]          ; sumarr אחסון סכום האיברים במשתנה
   mov  [bx],  ax
   ret 8
sum ENDP

END start
```

שאלה 7.9

כתבו תכנית שתבדוק אם מערך של איברים מטיפוס מילה ממוין בסדר עולה תוך כדי שימוש במחסנית.

שאלה 7.10

כתבו תכנית שתעביר לפרוצדורה מערך כפרמטר לפי אזכור. הפרוצדורה תמצא את הערך המקסימלי של המערך.

שאלה 7.11

נתונה תכנית שבה פרוצדורה מזמנת פרוצודרה אחרת; עקבו אחר ביצוע התכנית ותארו את תפקיד המחסנית ואת תוכנה בשלב הקריאה לפרוצדורות השונות ובשלב החזרה מהן.

```
.MODEL SMALL
.STACK  100h
.DATA
.CODE
start:
                                        ;אתחול מקטע הנתונים
```

```
    mov  ax,  @DATA
    mov  ds,  ax
    call   outproc
    mov  ax,  4c00h                                    ; יציאה מתכנית
    int     21h

outproc PROC
     jmp  endouts
     inproc PROC
     mov  ah,  0Fh
     ret
     inproc ENDP
endouts:
     call inproc
     inc  ah
     mov  bh,  0FFh
     ret
outproc ENDP
END start
```

7.7 מימוש פונקציות

פונקציה היא פרוצדורה שמחזירה ערך. מקובל להחזיר ערכים באמצעות האוגרים – ערך מטיפוס בית יוחזר באוגר AL וערך מטיפוס מילה יוחזר באוגר AX. אם נרצה להחזיר ערך מטיפוס מילה כפולה, נשתמש באוגרים AX ו-DX כאשר, המילה העליונה תאוחסן ב-DX והמילה התחתונה תאוחסן ב-AX. כאשר כותבים פונקציה יש לשים לב שלא יימחקו הערכים הקודמים של האוגרים בהם משתמשים להחזרת ערך, ובהתאם להכניס את הערכים המקוריים של האוגרים הללו למחסנית לפני הזימון שלה.

דוגמה 7.9

נכתוב פונקציה greater(a,b) שתקבל שני מספרים ללא סימן (לפי ערך) ותחזיר את הגדול בין השניים.

```
greater PROC
    mov  bp,  sp
    mov  ax,  [bp+2]          ;העתק לאוגר ax את b
    cmp  ax,  [bp+4]          ;השווה את a ל-b
    ja next                   ;אם b גדול מ-a קפוץ להוראה אליה צמודה התווית next
    mov  ax,  [bp+4]          ;העתק לאוגר ax את המספר הגדול
    next: ret   4
greater ENDP
```

בתכנית שמזמנת את הפרוצדורה, נאחסן תחילה את תוכנו של AX במחסנית, ולאחר מכן נעביר את הפרמטרים לפונקציה. אחרי שיסתיים ביצוע הפונקציה, נעתיק את הערך שהוחזר מהפרוצדורה לאוגר BX, ונשחזר את ערכו המקורי של ax:

```
    push  ax                  ;שמירת ערכו של ax
    push  a                   ;העברת פרמטר (לפי ערך) a
    push  b                   ;העברת פרמטר (לפי ערך) b
    call greater
    mov  bx, ax               ;המספר הגדול בין השניים
    pop   ax                  ;שחזור ערכו המקורי של ax
```

שאלה 7.12

נשנה את התכנית שהוצגה בדוגמה 7.9. הסבירו מה יקרה אם תוכן אוגר AX יישמר במחסנית במהלך ביצוע הפונקציה ולא בתכנית הראשית, כפי שנעשה בדוגמה?

```
.MODEL SMALL
.STACK  100h
.DATA
    a  dw 34
    b  dw 87
 .CODE
start:
                                        ; אתחול מקטע הנתונים
  mov  ax,  @DATA
  mov  ds,  ax
                                        ; העברת פרמטרים
  push  a
  push  b
  call greater
  mov  bx,  ax                          ; המספר הגדול בין השניים
  mov  ax,  4c00h                       ; סיום התכנית
  int     21h

greater PROC                            ;הגדרת הפרוצדורה
    mov  bp,  sp
    push  ax                            ; ax שמירת ערכו המקורי של
    mov  ax,  [bp+2]                    ; ax = b
    cmp  ax,  [bp+4]                    ; if b > a ?
    ja next                             ; then (b > a ) jump next
    mov  ax,  [bp+4]                    ; else ( a ≥ b) ax =a
next:
    pop  ax                             ; ax שחזור ערכו המקורי של
    ret    4
greater ENDP
END start
```

7.8 פונקציה רקורסיבית

פונקציה רקורסיבית היא פונקציה שבמהלכה הפונקציה מזמנת את עצמה. הרקורסיה היא כלי נוח ומתוחכם לפיתוח אלגוריתמים לפתרון בעיות, ובמקרים מסוימים היא מפשטת את תהליך הפתרון. בסעיף זה נסביר תחילה מהו חישוב רקורסיבי ונציג את המבנה של האלגוריתם הרקורסיבי; בהמשך נתאר כיצד המעבד משתמש במחסנית לביצוע תכנית שמכילה פונקציה רקורסיבית.

דוגמה 7.10

נתאר חישוב רקורסיבי של פונקציה המחשבת עצרת של מספר חיובי ושלם. עצרת של המספר n הוא מכפלת כל המספרים השלמים מ-1 עד n, כאשר 0 עצרת מוגדר כ-1. משתמשים בסימן ! כדי לציין עצרת, לדוגמה, שתיים עצרת נכתב !2, ובאופן כללי, עבור n רושמים !n .

נראה שני חישובים לדוגמה:

- 3 עצרת מחושב כ- $3! = 3\cdot2\cdot1$
- 7 עצרת מחושב כ- $7! = 7\cdot6\cdot5\cdot4\cdot3\cdot2\cdot1$

וכדומה.

אלגוריתם איטראטיבי הוא אלגוריתם המבצע חזרה מפורשת של קטע מסוים בו, כמה פעמים. האלגוריתם לחישוב n עצרת עבור $n>0$ הוא פשוט ומוכר (המילה "עצרת" באנגלית היא "factorial"):

```
factorial =1
לכל x מ-n עד 1 בצע:
    factorial = factorial·x
    x = x – 1
```

אבל אפשר לחשב עצרת של מספר גם בעזרת תהליך רקורסיבי. נבחן כמה מקרים פרטיים וננסה למצוא כלל לחישוב !n. נתחיל ברישום מפורש של נוסחת החישוב עבור n-ים שונים; במהרה נגלה כי יש קשר בין חישוב !n לחישוב מקרה פשוט יותר שהוא !(n-1):

$0! = 1$

$1! = 1 \cdot 0!$

$2! = 2 \cdot 1 = 2 \cdot 1!$

$3! = 3 \cdot 2 \cdot 1 = 3 \cdot 2!$

$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 4 \cdot 3!$

ועבור $n<0$ אפשר לרשום $n! = n \cdot (n-1)!$

כדי להבין כיצד מתבצע החישוב הרקורסיבי, נעקוב אחר החישוב של $4!$:

כדי לחשב $4!$ עלינו לחשב	$4! = 4 \cdot (4-1)! = 4 \cdot 3!$
כדי לחשב $3!$ עלינו לחשב	$3! = 3 \cdot (3-1)! = 3 \cdot 2!$
כדי לחשב $2!$ עלינו לחשב	$2! = 2 \cdot (2-1)! = 2 \cdot 1!$
כדי לחשב $1!$ עלינו לחשב	$1! = 1 \cdot (1-1)! = 1 \cdot 0!$
אולם, כפי שאמרנו, $0!$ מוגדר:	$0! = 1$
ולכן כעת נוכל לחשב את $1!$:	$1! = 1 \cdot 0! = 1 \cdot 1 = 1$
ובהתאם נוכל לחשב את $2!$:	$2! = 2 \cdot 1! = 2 \cdot 1 = 2$
ובהתאם נוכל לחשב את $3!$:	$3! = 3 \cdot 2! = 3 \cdot 2 = 6$
ולסיום נוכל לחשב את $4!$:	$4! = 4 \cdot 3! = 4 \cdot 6 = 24$

חישוב מפורש של המקרים הפרטיים, מאפשר להבחין כי בחישוב עצרת ישנם שני מקרים:

א. מקרה פשוט, שבו החישוב אינו תלוי בחישוב עצרת של מקרה אחר:

$0! = 1$

מקרה זה נקרא "תנאי העצירה של החישוב הרקורסיבי".

ב. המקרה המורכב יותר, שבו חישוב העצרת תלוי בחישוב העצרת של ערך אחר, לדוגמה:

$4! = 4 \cdot 3!$

ולכן יש לחשב קודם את $(n-1)!$ ואחר כך את $n!$ עצמו.

דוגמה 7.11

כדוגמה נוספת, נתאר ביצוע כפל של שני מספרים טבעיים $a\cdot b$ בצורה רקורסיבית. אפשר להציג את המכפלה של $a\cdot b$, כאשר a ו-b הם מספרים שלמים וחיוביים, כחיבור b פעמים של a, כלומר:

$$a\cdot b = \underbrace{a+a+\ldots +a}_{\text{פעמים } b}$$

אם נבחן כמה מקרים פרטיים, נראה כי ניתן לרשום את פעולת הכפל תוך שימוש בהגדרה רקורסיבית, כאשר תנאי העצירה של החישוב הרקורסיבי הוא $x\cdot 0=0$.

לדוגמה:

חישוב $4\cdot 3$

$a=4, b=3 \rightarrow 4\cdot 3 = 4\cdot 2 + 4$

$a=4, b=2 \rightarrow 4\cdot 2 = 4\cdot 1 + 4$

$a=4, b=1 \rightarrow 4\cdot 1 = 4\cdot 0 + 4$

$a=4, b=0 \rightarrow 4\cdot 0 = 0$ (על-פי תנאי העצירה)

כעת נוכל לחשב

$4\cdot 1 = 0 + 4$

$4\cdot 2 = 4\cdot 1 + 4 = 8$

$4\cdot 3 = 4\cdot 2 + 4 = 12$

או חישוב $9\cdot 5$

$a=9, b=5 \rightarrow 9\cdot 5 = 9\cdot 4 + 9$

$a=9, b=4 \rightarrow 9\cdot 4 = 9\cdot 3 + 9$

$a=9, b=3 \rightarrow 9\cdot 3 = 9\cdot 2 + 9$

$a=9, b=2 \rightarrow 9\cdot 2 = 9\cdot 1 + 9$

$a=9, b=1 \rightarrow 9\cdot 1 = 9\cdot 0 + 9$

$a=9, b=0 \rightarrow 9\cdot 0 = 0$ (על-פי תנאי העצירה)

כעת נוכל לחשב

$9\cdot 1 = 9\cdot 0 + 9 = 9$

$9\cdot 2 = 9\cdot 1 + 9 = 18$

$9\cdot 3 = 9\cdot 2 + 9 = 27$

$9\cdot 4 = 9\cdot 3 + 9 = 36$

$9\cdot 3 = 9\cdot 4 + 9 = 45$

כעת נרשום את ההגדרה הרקורסיבית הבאה כדי לחשב כפל שני מספרים טבעיים:

- תנאי העצירה יהיה עבור המקרה הפשוט, שבו $b=0$ ההגדרה תהיה $a \cdot b = 0$
- עבור המקרה המורכב, שבו $b>1$ ההגדרה תהיה $a \cdot b = a + a \cdot (b-1)$

באופן כללי, הגדרה של תהליך רקורסיבי צריכה לכלול שלושה תנאים:

א. תנאי עצירה, דהיינו: מקרה "פשוט" או מקרים "פשוטים", בהם החישוב ידוע ואינו תלוי בחישוב של ביטוי רקורסיבי אחר, למשל $0!=1$

ב. מקרה מורכב, המוגדר באמצעות חישוב של מקרים "פשוטים" יותר, למשל $n! = n \cdot (n-1)!$

ג. "פישוט" חוזר ונשנה, באמצעות הפעולות שהוגדרו במקרה "המורכב", חייב להוביל, בסופו של דבר, לתנאי העצירה שאותו הגדרנו מפורשות בתנאי א.

כתיבת פונקציה רקורסיבית בשפת אסמבלי

בסעיף זה נתאר מימוש של פונקציות רקורסיביות בשפת אסמבלי בהן הערך המוחזר מחישוב רקורסיבי מושם באוגר AL (או AX).

דוגמה 7.12

תחילה נציג פונקציה רקורסיבית פשוטה בשם example המקבלת כפרמטר מספר שלם חיובי באוגר AL ומזמנת את עצמה כל עוד AL גדול מ-0. האלגוריתם הבא מתאר את אופן פעולה הפונקציה example:

אם $AL > 0$ אזי
  חשב $AL = AL - 1$
  קריאה לפונקציה: CALL example
אחרת $AL = 0$

נשים לב כי הפונקציה מכילה את כל התנאים הדרושים לביצוע תהליך חישוב רקורסיבי: המקרה המורכב מוגדר כ- $AL=AL-1$ ובאמצעותו נקבל, לאחר ביצוע חישובים חוזרים ונשנים, את הערך הפשוט ביותר $AL=0$.

נרשום תכנית הכוללת את הפונקציה example :

```
.MODEL  SMALL
.STACK 100h
.DATA
.CODE
 start:
        mov  ax, @DATA          ; אתחול מקטע נתונים
        mov  ds,  ax
        mov  al,  2             ;העברת פרמטר לפי ערך באוגר al
        call  example           ; זימון הפונקציה הרקורסיבית
        mov  ax,  4c00h         ; יציאה מתכנית
        int    21h
                                ;הגדרת הפונקציה הרקורסיבית
   example PROC
        cmp   al, 0
        je  done                ; אם al = 0 סיים
        dec    al               ; al= al –1
        call  example           ; קריאה רקורסיבית
        done:  ret
   ENDP example
 END start
```

כדי לעקוב אחר ביצוע הפונקציה example נתאר את קטע ממקטע הקוד שמכיל את התכנית, קטע המחסנית ואת האוגרים בהם נשתמש לביצוע התכנית. נניח כי לאחר תרגום וטעינת התכנית זיכרון, הפונקציה example מאוחסנת בכתובות 000Fh-0018h.

שלב ראשון: תכנית ראשית מזמנת את הפונקציה example

בהתחלה, מצביע תכנית IP מכיל את הכתובת של ההוראה הראשונה בתכנית והיא 0000.
הוראות התכנית מתבצעות בזו אחר זו, עד להוראה שמזמנת את הפונקציה :

CALL example

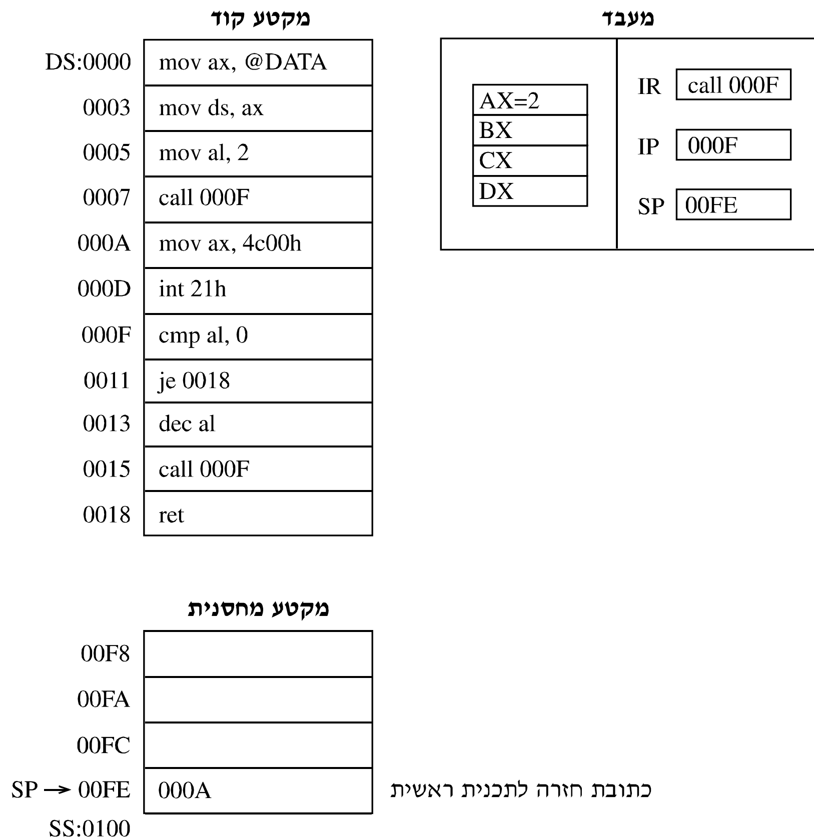
כתוצאה מזימון הפונקציה, מתבצעות הפעולות הבאות :

א. SP קטן ב-2 וערכו הוא 00FE

ב. כתובת החזרה, היא הכתובת 000A, נדחפת למחסנית לכתובת עליה מצביע SP

ג. ערכו של IP משתנה והוא מצביע על הכתובת הראשונה בפונקציה היא הכתובת 000F.

האיור הבא מתאר את מצב הזיכרון והמעבד לאחר ביצוע הקריאה לפונקציה:



איור 7.14
מצב הזיכרון והמעבד לאחר קריאה לפונקציה **example**

שלב שני: פונקציה רקורסיבית מזומנת בפעם הראשונה

ההוראה הבאה שמתבצעת היא ההוראה עליה מצביע IP והיא נמצאת בכתובת 000F. בהוראה זו ערכו של AL מושווה לערך 0, ומאחר וערכו הוא 2, הדבר מתבטא באוגר הדגלים. ההוראה הבאה je 0018 בודקת האם הערכים שהושוו מקודם זהים, ומאחר שעל-פי אוגר הדגלים התנאי נכשל – הקפיצה אינה מתבצעת, והביצוע עובר להוראה הבאה

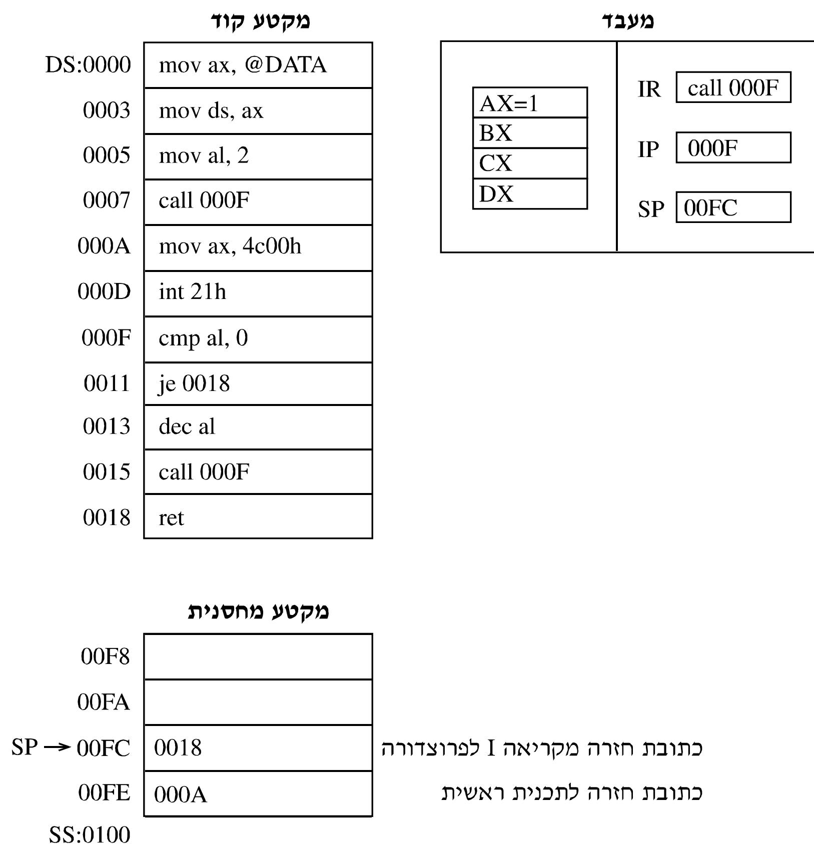
שנמצאת בכתובת 0013. בהוראה זו מופחת 1 מערכו של AL ולאחר מכאן מתבצעת ההוראה call 000F שנמצאת בכתובת 0015. בעת הקריאה לפונקציה מתבצעות הפעולות הבאות:

א. ערכו של SP קטן ב-2 וערכו הוא 00FC

ב. כתובת החזרה, היא הכתובת 0018, נדחפת למחסנית לכתובת עליה מצביע SP

ג. ערכו של IP משתנה והוא מצביע על הכתובת הראשונה בפונקציה היא הכתובת 000F.

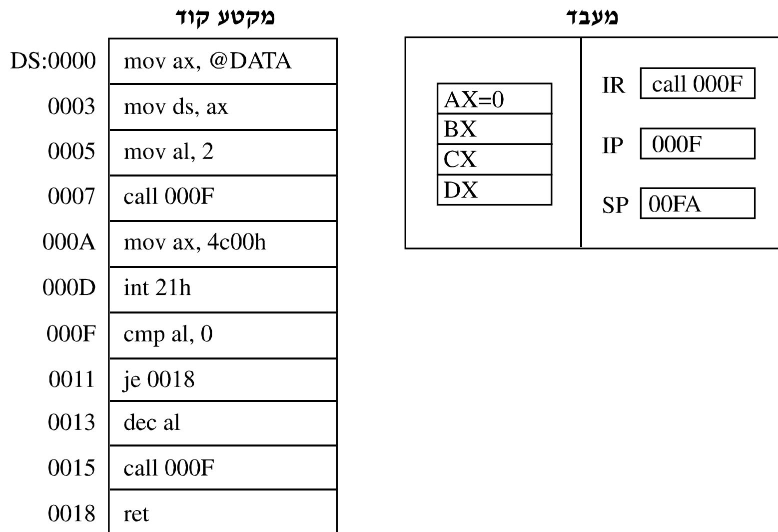
כדי לפשט את המעקב נתאר את מצב המעבד והזיכרון רק לאחר ביצוע הוראת הקריאה לפונקציה וביצוע הוראת חזרה מפונקציה. איור 7.15 מציג את המצב של הזיכרון לאחר ביצוע הזימון לפונקציה:

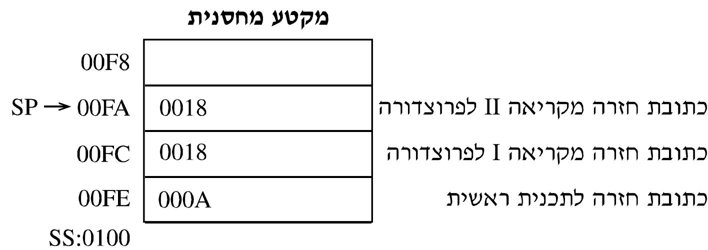


איור 7.15
מצב הזיכרון והמעבד לאחר הזימון הרקורסיבי הראשון

שלב שלישי: פונקציה רקורסיבית מזומנת בפעם השניה

מאחר וערכו של AL אינו 0, התהליך שתואר בשלב השני מתבצע שוב: ערכו של AL מופחת ב-1 ומתבצעת הוראת הקריאה לפונקציה. בהתאם איור 7.16 מציג את מצב המעבד והזיכרון לאחר הזימון הרקורסיבי הראשון.





איור 7.16
מצב הזיכרון והמעבד לאחר הזימון הרקורסיבי השני

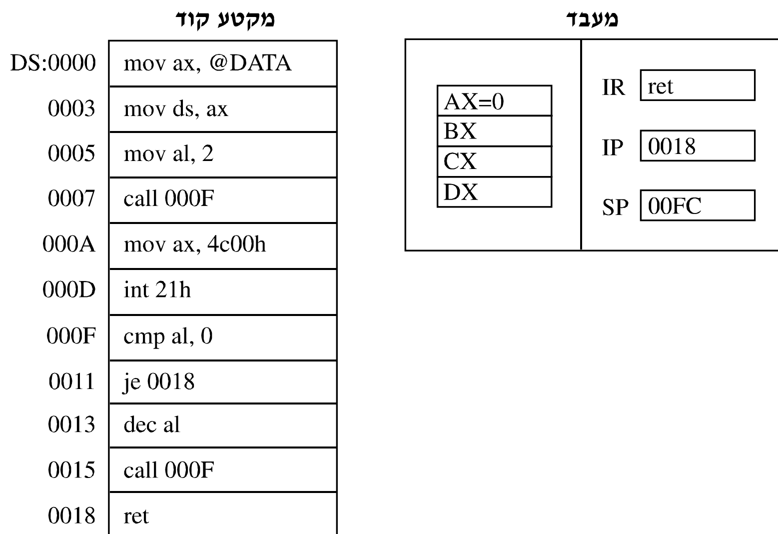
שלב רביעי: חזרה מביצוע של הפונקציה הרקורסיבית (זימון שני)

כעת ערכו של AL הוא 0 ולכן תהליך החישוב משתנה. כתוצאה מפעולת ההשוואה, הוראת הקפיצה המותנת מעבירה את המשך הביצוע להוראת החזרה שבכתובת 0018. בביצוע הוראת RET מתבצעות הפעולות הבאות:

א. כתובת החזרה 0018 נשלפת מכתובת במחסנית עליה מצביע SP והיא מאוחסנת באוגר IP

ב. ערכו של SP גדל ב-2 והוא כעת 00FC

בהתאם איור 7.17 מציג את מצב המעבד והזיכרון לאחר ביצוע הוראת החזרה:



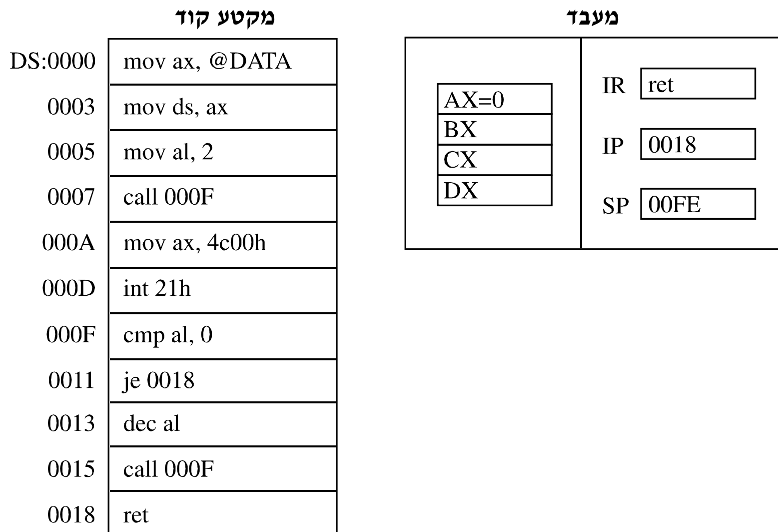
איור 7.17
מצב הזיכרון והמעבד לאחר חזרה מהזימון הרקורסיבי השני

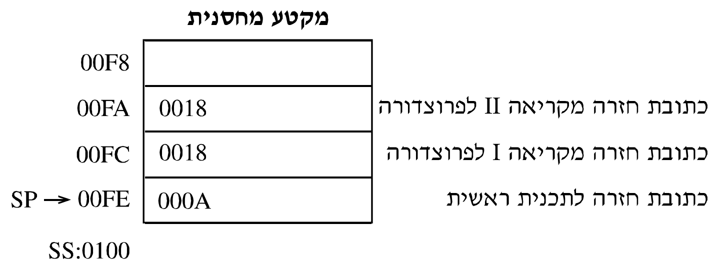
שלב חמישי: חזרה מביצוע של הפונקציה הרקורסיבית (זימון ראשון)

בכתובת החזרה נמצאת ההוראה RET ובהתאם מתבצעות הפעולות הבאות:

א. כתובת החזרה 0018 נשלפת מהכתובת במחסנית עליה מצביע SP והיא מאוחסנת באוגר IP

ב. ערכו של SP גדל ב-2 והוא כעת 00FE





איור 7.18
מצב הזיכרון והמעבד לאחר הזימון הרקורסיבי הראשון

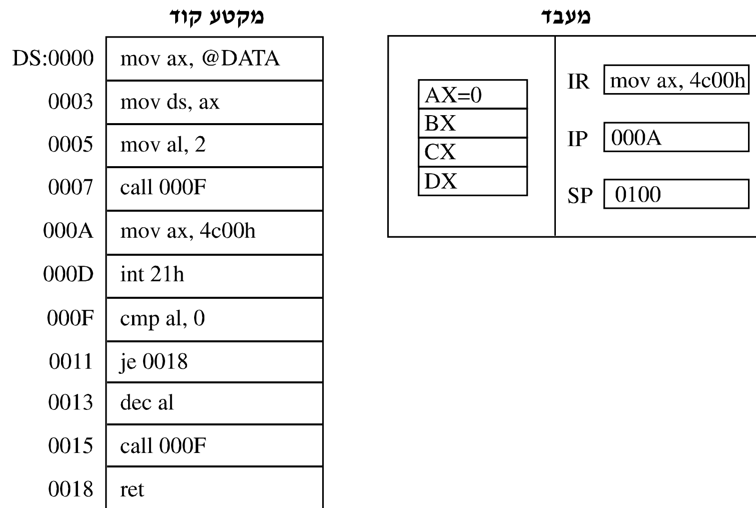
שלב שישי: חזרה מביצוע של הפונקציה רקורסיבית והמשך התכנית הראשית

בשלב זה מתבצעות הפעולות הבאות:

א. כתובת החזרה 000A נשלפת מהכתובת במחסנית עליה מצביע SP והיא מאוחסנת באוגר IP

ב. ערכו של SP גדל ב-2 והוא כעת 0100

כעת התכנית ממשיכה את הביצוע מהוראה שעוקבת להוראה שזימנה את הפונקציה, כלומר מההוראה mov ax, 4C00h.



איור 7.19
מצב הזיכרון והמעבד לאחר חזרה לתכנית הראשית

שימו לב כי לאחר שהפונקציה הסתיימה, ערכו של SP הוא שוב 0100h.
שאלה למחשבה – מדוע בסיום ביצוע הפונקציה example ערכו של אוגר AL הוא 0?

דוגמה 7.13

הפונקציה הרקורסיבית שתיארנו אינה מעבירה פרמטרים. אך פונקציות רבות, למשל, פונקציה המחשבת את n! , צריכות להעביר פרמטרים. במימוש של מגנון לחישוב רקורסיבי משתמשים במחסנית לא רק כדי לשמור כתובות חזרה, אלא גם לשמור את הערכים שחושבו בכל קריאה של הרקורסיה.

נכתוב תכנית לחישוב n!, אך תחילה נציג אלגוריתם מתאים:

הפונקציה מקבל מקבלת מספר טבעי n, מחשבת ומחזירה את n!; שימו לב, ש-n! מוגדרת גם עבור 0 (0!=1), אך כאן אנחנו מתייחסים למספרים טבעיים בלבד:

הפונקציה factorial(n) :

אם $n = 1$ אזי
החזר $factorial = 1$
אחרת
החזר $factorial = n \cdot factorial(n-1)$

נשתמש באוגר BX כפרמטר עבור n ובאוגר AX להחזרת ערך. במקרה הכללי בו $n>0$ יש לזמן שוב את הפונקציה עבור $n-1$. זימון הפונקציה נעשה מתוך הפונקציה, אך לפני כן עלינו לשמור את ערכו של n ולשם כך נשתמש במחסנית.

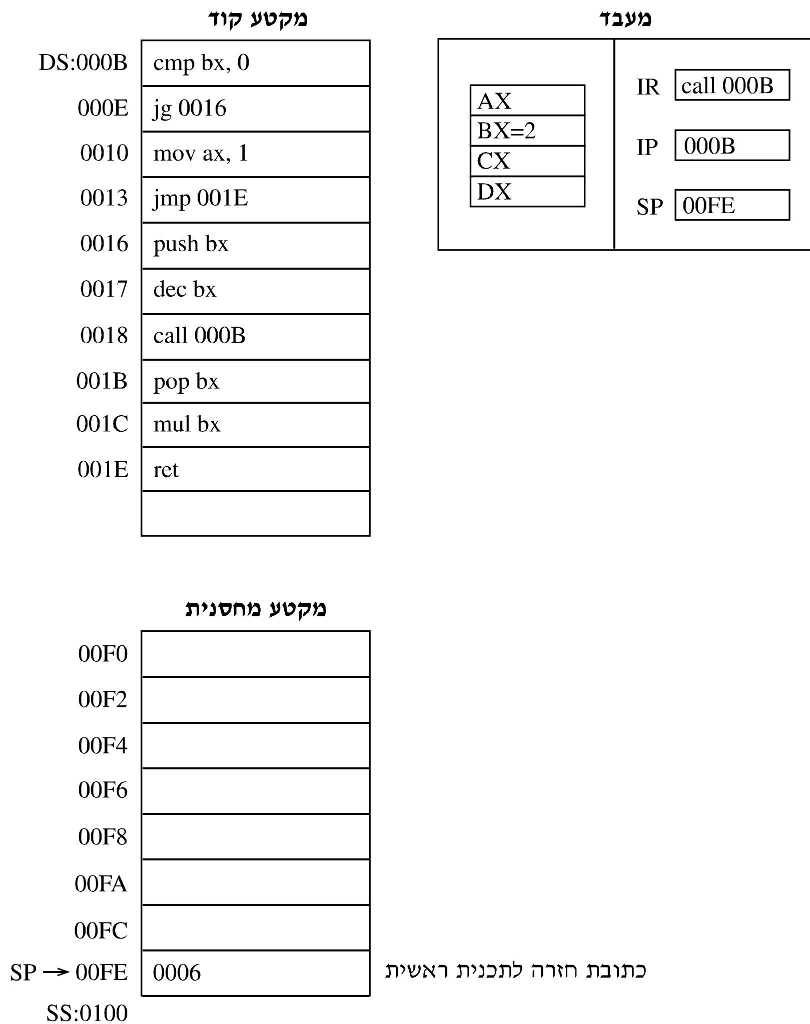
להלן קטע תכנית המכילה פונקציה רקורסיבית לחישוב n! :

```
    mov  bx, 2                    ; bx=2
    call  factorial               ; factorial(2) זימון של הפונקציה הרקורסיבית
finish:                           ; סיום התכנית
    mov  ax,  4c00h
    int    21h
factorial PROC
                                  ;n=1 טיפול במקרה הפשוט
    cmp  bx,  0                   ;השווה את ערכו של bx לאפס
    jg call_fact                  ; אם BX >1 קפוץ להמשך ביצוע רקורסיה
    mov  ax,  1                   ; AX =1 תנאי עצירה המקרה בו
    jmp done
                                  ; n>11 טיפול במקרה הכללי
    call_fact:
         push  bx                 ; bx דחוף למחסנית את
         dec    bx                ; bx הפחת 1 מאוגר
         call  factorial          ; facotiral(bx–1) זימון לקריאה הרקורסיבית
    pop  bx                       ; bx בסיום הרקורסיה: שלוף ערך
    mul  bx                       ; ax הכפל את ערכו של bx ב-ax ושים את התוצאה באוגר
done:     ret
ENDP factorial
END start
```

לאחר קריאה לפונקציה, ערכו של BX הוא 2 ולמחסנית נדחפת כתובת הוראת החזרה 001F ומצביע מחסנית SP=00FEh. כדי לפשט את המעקב אחר החישוב הרקורסיבי נתאר את הזיכרון ומצב המעבד המכיל את הפונקציה הרקורסיבית בלבד.

שלב ראשון: קריאה לפונקציה factorial(2)

איור 7.20 מציג את מצב המעבד והזיכרון לאחר שהתכנית הראשית קראה לפונקציה הרקורסיבית.



איור 7.20
מצב המעבד והזיכרון לאחר שהתכנית הראשית קראה ל-factorial(2)

שלב שני: קריאה לפונקציה factorial(1)

ההוראה עליה מצביע IP היא ההוראה הראשונה בפונקציה. מאחר וערכו של BX שונה מ-0 התכנית ממשיכה את הביצוע החל מהוראה 0016. לפני הזימון הרקורסיבי הבא, ערכו של BX נדחף למחסנית, ערכו של BX קטן ב-1 ומתבצעת קריאה לפונקציה factorial לחישוב factorial(1).

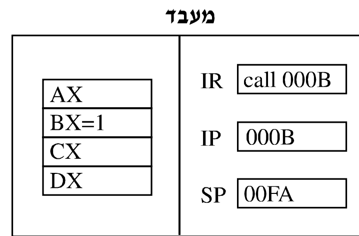
איור 7.21 מתאר את מצב הזיכרון והאוגרים לאחר ביצוע שלוש ההוראות:

```
push bx
dec bx
call factorial                                  ; call factorial(1)
```

מקטע קוד

DS:000B	cmp bx, 0
000E	jg 0016
0010	mov ax, 1
0013	jmp 001E
0016	push bx
0017	dec bx
0018	call 000B
001B	pop bx
001C	mul bx
001E	ret

מעבד



מקטע מחסנית

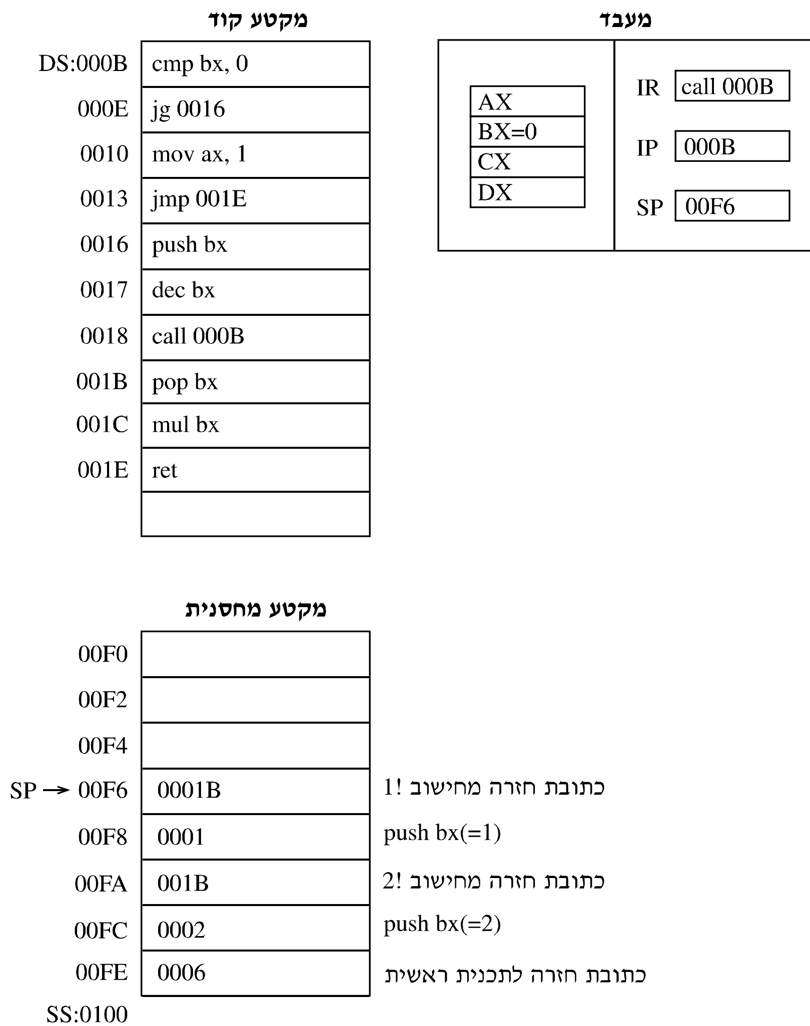
00F0		
00F2		
00F4		
00F6		
00F8		
SP → 00FA	001B	כתובת חזרה מחישוב 2!
00FC	0002	push bx(=2)
00FE	0006	כתובת חזרה לתכנית ראשית
SS:0100		

איור 7.21
מצב המעבד והזיכרון לאחר שהתכנית הראשית קראה ל-factorial(1)

שלב שלישי: קריאה לפונקציה factorial(0)

חישוב זה דומה לחישוב הקודם, ההוראה עליה מצביע IP היא ההוראה הראשונה בפונקציה. ערכו של BX שונה מ-0 לכן מדלגים להמשך ביצוע החל מהוראה 0016. לפני הזימון הרקורסיבי הבא, ערכו של BX נדחף למחסנית, ערכו של BX קטן ב-1 ומתבצעת קריאה לפונקציה factorial לחישוב factorial(0).

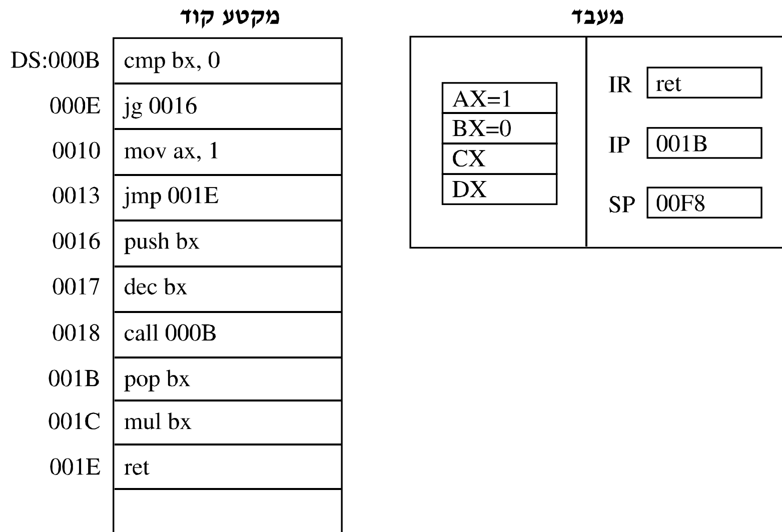
איור 7.22 מתאר את מצב הזיכרון והאוגרים לאחר הקריאה הרקורסיבית.

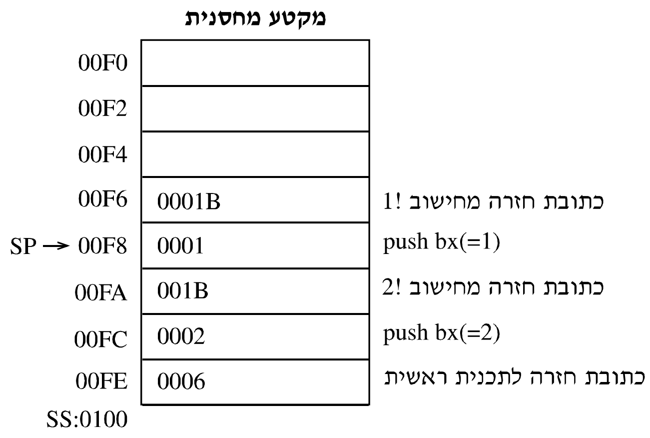


איור 7.22
מצב המעבד והזיכרון לאחר שהתכנית הראשית קראה ל-factorial(0)

שלב רביעי: חזרה מחישוב factorial(0)

הפעם תהליך החישוב משתנה, מאחר וערכו של BX הוא 0, הביצוע עובר להוראה שכתובתה 0010 ובהתאם מתבצעות הפעולות הבאות: מושם הערך 1 באוגר AX והביצוע ממשיך מהוראה שכתובתה 001E, היא הוראת החזרה RET. איור 7.23 מציג את מצב הזיכרון והאוגרים לאחר ביצוע הוראת החזרה.

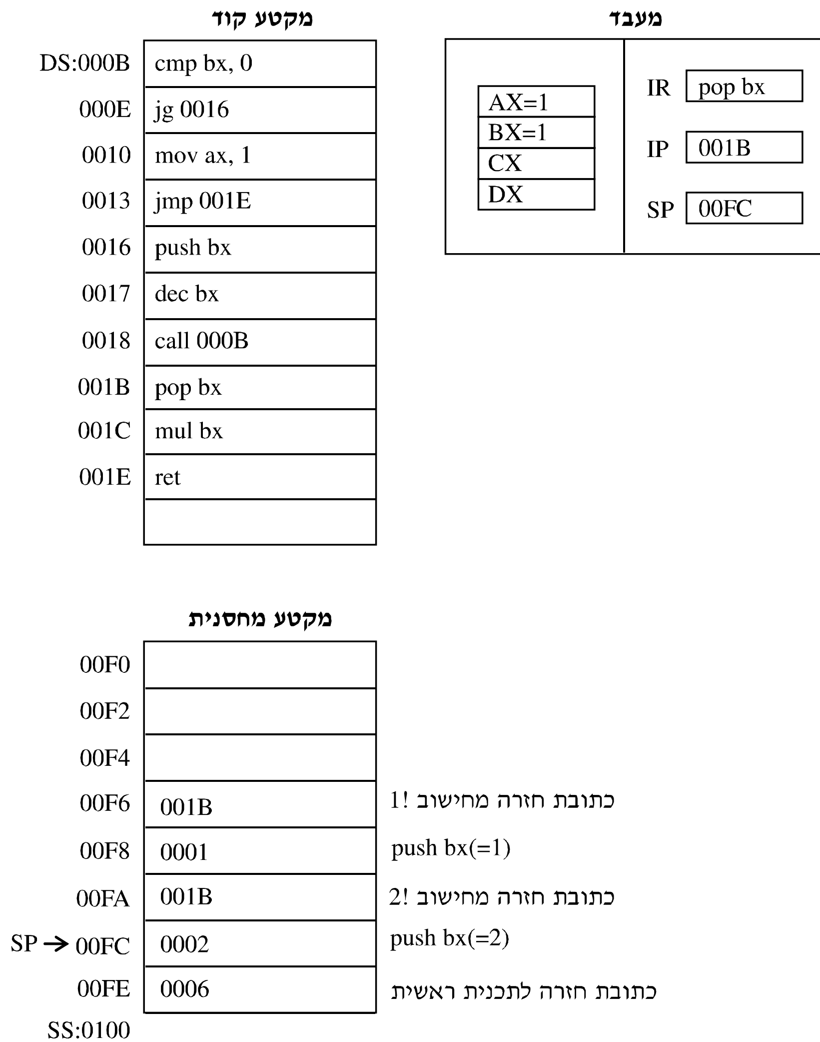




איור 7.23
מצב המעבד והזיכרון לאחר חזרה מזימון של factorial(0)

שלב חמישי: חזרה מחישוב factorial(1)

שוב מתבצעות ההוראות שמתחילות בכתובת 001B ובהתאם נשלף ערך מהמחסנית והוא מושם ב-bx=1 ולאחר מכן מתבצע החישוב $AX \cdot BX = 1 \cdot 1 = 1$ ולסיום מתבצעת הוראת חזרה RET הנמצאת בכתובת 001E. איור 7.24 מציג מצב הזיכרון והאוגרים לאחר ביצוע הוראת החזרה מ-factorial(1).

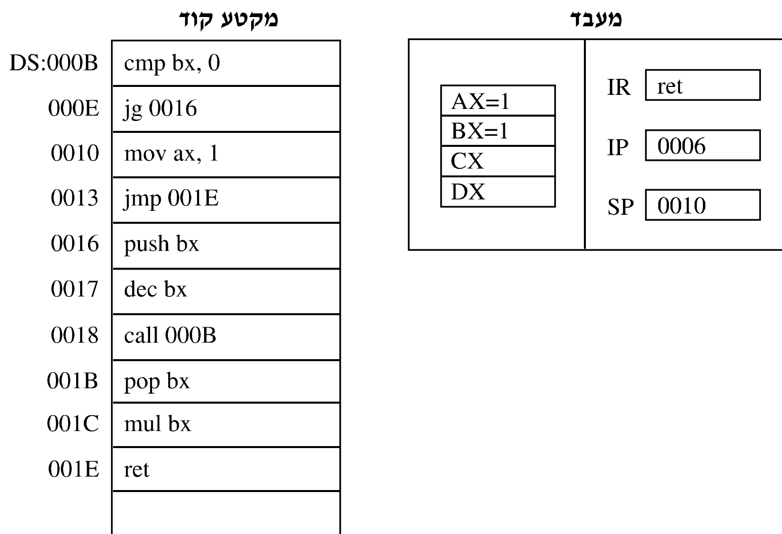


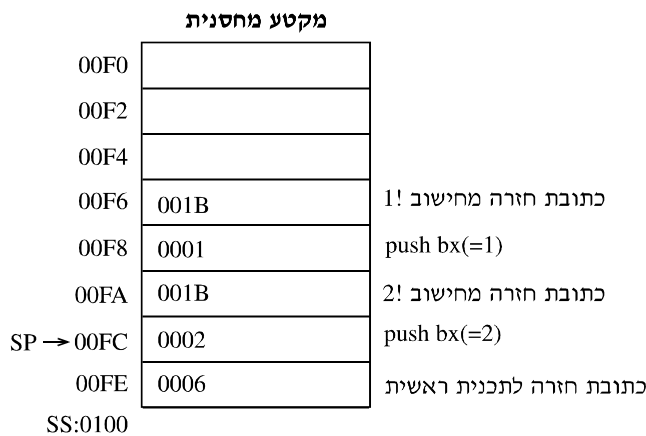
איור 7.24
מצב המעבד והזיכרון לאחר חזרה מזימון של factorial(1)

שלב שישי: חזרה לתכנית הראשית

שוב מתבצעות ההוראות שמתחילות בכתובת 001B ובהתאם נשלף ערך מהמחסנית והוא מושם ב-bx=2 ולאחר מכן מתבצע החישוב $AX \cdot BX = 1 \cdot 2 = 2$ ולסיום מתבצעת הוראת חזרה RET.

הפעם כתובת החזרה שנשלפת היא כתובת החזרה בתכנית הראשית. איור 7.25 מציג מצב הזיכרון והאוגרים לאחר ביצוע הוראת החזרה לתכנית הראשית





איור 7.25
מצב המעבד והזיכרון לאחר חזרה לתכנית הראשית

7.9 מקרו

מקרו (macro) הוא כלי שמספק האסמבלר – הוא מאפשר לייצג בתכנית קטע טקסט (שיכול להכיל הוראות או נתונים) באמצעות שם לוגי; בזמן ההידור אפשר להחליף את השם הלוגי בקטע הטקסט עצמו. כלי זה מסופק על-ידי מהדרים רבים, בשפות שונות, לאו דווקא בשפת אסמבלי. כאשר יש בתכנית קטעי קוד שחוזרים על עצמם, המקרו מקל את תהליך הכתיבה. מגדירים את המקרו פעם אחת בתכנית, ובכל מקום שקטע קוד זה דרוש, מחליפים את השם בקוד עצמו. נדגיש כי במקרו החלפת הקוד מתבצעת בשלב האסמבלר כך שביצוע התכנית הוא לפי סדר ההוראות הרשומות בתכנית, ואילו לביצוע הפרוצדורה יש לקפוץ למיקום אחר במקטע הקוד בו נמצאות ההוראות של הפרוצדורה (ערכו של IP מתעדכן לכתובת ההוראה הראשונה של הפרוצדורה).

7.9.1 מבנה של מקרו

כתיבת מקרו מתחילה בכותרת המכילה את שם המקרו, אחריו נכתב גוף המקרו המכיל הוראות והנחיות לביצוע, ולסיום נכתבת הנחיה המציינת את סיום המקרו:

```
MACRO    שם        [paramet1, paramet2,…]
    גוף המקרו

ENDM
```

אפשר להגדיר את המקרו בכל מקום בקובץ התכנית. כאשר המהדר נתקל בקריאה למקרו, הוא משבץ את גוף המקרו במקום שבו הופיעה הקריאה. אפשר להעביר למקרו פרמטרים, אותם יש לציין בכותרת של המקרו. פרמטרים אלה נקראים פרמטרים פורמליים. בהמשך נתייחס אליהם.

דוגמה 7.14

לדוגמה, נכתוב מקרו המבצע השמה של תחילת מקטע הנתונים לאוגר סגמנט הנתונים DS:

```
MACRO setDS
   mov  ax,  @DATA
   mov  ds,  ax
ENDM
```

המילים MACRO ו-ENDM הן הנחיות אסמבלר ולא פקודות אסמבלי.

הזימון של מקרו, נעשה על-ידי כתיבת שם המקרו בתכנית. פעולה זו גורמת להכנסת הקוד שלו לתכנית. תכנית האסמבלר שמתרגם את התכנית מטפל במקרו בצורה הזו: בתחילת ההידור, מסמן האסמבלר את כל ההתייחסויות למקרו; אחרי-כן הוא עובר שוב על התכנית, ובכל מקום שסומנה התייחסות למקרו, הוא מחליף את שם המקרו בגוף המקרו.

לדוגמה, נרשום מקרו setDS לאתחול אוגר DS, ומקרו endprogram ליציאה מהתכנית, ואז נזמן את המקרו בתכנית הזו:

```
.MODEL  SMALL
setDS MACRO                          ; מקרו לאתחול אוגר מקטע הנתונים
   MOV  AX,  @DATA
   MOV  DS,  AX
ENDM
endprogram MACRO
   MOV  AX,  4c00h                   ; מקרו ליציאה מהתכנית
   INT     21h
ENDM
.STACK 100h
.DATA
.CODE
start:
                                     ; אתחול מקטע הנתונים
   setDS
                                     ; יציאה מתכנית
   endprogram
END start
```

אם נהדר את התכנית וניצור קובץ LST, נוכל לראות כי בקובץ שהתקבל, נרשמו הוראות גוף המקרו בכל מקום שזימנו את המקרו.

7.9.2 העברת פרמטרים למקרו

ניתן להגדיר פרמטרים למקרו; אל הפרמטרים המועברים מתייחסים כמחרוזות, ובזמן שהאסמבלר מחליף את שם המקרו בגוף המקרו, הוא גם מחליף את שמות הפרמטרים המצויים בגוף המקרו ("הפרמטרים הפורמאליים") במחרוזות שהועברו ("הפרמטרים האקטואליים"). שימו לב, שהפרמטרים הפורמאליים מציינים עבור האסמבלר היכן להכניס את הפרמטרים האקטואליים המצורפים להוראה הקוראת למקרו; הם אינם מציינים כתובות של משתנים בזיכרון.

לדוגמה, נכתוב מקרו בשם writeChar המקבל כפרמטר תו ומציגו על הצג.

```
writeChar MACRO  char
    mov  dl,  char
    mov  ah,  2
    int  21h
ENDM
```

בתכנית נזמן את המקרו עם פרמטר אקטואלי המכיל את התו שאנו רוצים להציג:

```
writeChar 'A'
```

הקריאה למקרו עם פרמטרים אקטואליים נעשית על-ידי רישום שם המקרו ואחריו רשימה של פרמטרים אקטואליים המופרדים על-ידי פסיקים, רווחים או תווי TAB.

ניתן לזמן את המקרו כמה פעמים בתכנית; במקרה כזה כל זימון יוחלף בקוד המקרו, כך שהתכנית שתתקבל עשויה להיות ארוכה מאוד.

דוגמה 7.15

נכתוב תכנית למיון שלושה מספרים; הם יוכנסו לתוך המשתנים a, b , c כך ש-a>b>c.
התכנית תיעזר במקרו swap שתפקידו להחליף בין שני ערכים ולממש את האלגוריתם הזה:

אם a<b החלף את תוכן a ו-b ביניהם
אם b<c החלף את תוכן בין b ל-c ביניהם
אם a<b החלף את תוכן בין a ל-b ביניהם

נתבונן תחילה במקרו המתאים:

שם המקרו הוא swap והוא משתמש בשני פרמטרים first ו-second. קטע המקרו כולל הוראות בשפת-סף והנחיות מקרו. ההנחיה LOCAL מגדירה כי התווית in_order היא תווית מקומית. אנו חייבים להוסיף הנחיה זו, כדי למנוע מצב שבו המקרו יפרוש את קטע קוד פעמיים באותה תכנית ואז יוגדרו שתי תווית עם אותו שם. ההנחיה LOCAL מגדירה תווית לשימוש מקומי בתוך המקרו, וכך ניתן להשתמש באותה תווית בכל הקריאות למקרו ובזמן ההידור מחליף המהדר כל תווית בכתובת של הוראה.

```
.MODEL  SMALL
 swap   MACRO first, second
    LOCAL in_order                          ; הגדרת תווית
    mov  al,  first                         ; AL נתון ראשון
    cmp  al,  second                        ; השוואת שני הנתונים
    jnb in_order
    xchg  al,  second                       ; החלפת נתונים במידת הצורך
    mov  first, AL
in_order:
ENDM

.STACK 100h
.DATA
      a  DB  29
      b  DB  15
      c  DB  23
.CODE
start:
                                            ;אתחול מקטע הנתונים
  mov  ax,  @DATA
  mov  ds,   ax
                                            ;מיון שלושה איברים
  swap  a,  b
  swap  b,  c
```

```
  swap  a,  b
; יציאה מהתכנית
  mov   ax,  4c00h
  int    21h
END start
```

כדי להיווכח ששם המקרו אכן הוחלף בגוף המקרו, הדרו את התכנית וצרו גם קובץ lst. אפשר להתבונן בתכנית גם באמצעות ה-debugger.

שאלה 7.15

1. כתבו מקרו להחלפת תוכן שני משתנים בזיכרון;
2. כתבו מקרו לחיבור 3 משתנים;
3. כתבו מקרו בשם pushreg שידחוף למחסנית את ערכי האוגרים ומקרו בשם popreg שישלוף את ערכי האוגרים.

7.9.3 מנגנון ביצוע מקרו והעברת פרמטרים

למרות הדמיון בין מקרו לפונקציה, יש הבדל משמעותי בין דרכי המימוש שלהם. בעוד שגוף פונקציה מופיע בתכנית פעם אחת, גוף המקרו משוכפל בכל מקום אליו הוא זומן. הבדל זה משפיע על אופן ביצוע המקרו לעומת הביצוע של פונקציה. כאשר מזמנים פונקציה, מצביע ההוראה IP מוחלף בכתובת ההוראה הראשונה של הפונקציה; בסיום הפונקציה הוא מוחלף בכתובת של ההוראה הבאה לביצוע. כאשר משתמשים במקרו, קטע קוד שמתאר המקרו מוכנס בתכנית במקום שם המקרו על-ידי תכנית האסמבלר וכך מתקבלת התכנית שמורצת.

השימוש במקרו "מנפח" את קוד התכנית ומאריך אותה בצורה משמעותית, לכן בוחרים להשתמש במקרו למימוש פעולות בסיסיות, המופיעות בתכנית כמה פעמים, במיוחד כאשר הקוד הדרוש עבורן קצר יחסית. השימוש בפונקציה, לעומת זאת, אינו מאריך את התכנית, אלא את הזמן הדרוש לביצועה, בגלל מנגנון הקפיצה והחזרה מהתכנית, שמאריך את שלב ההבאה במחזור הבאה-וביצוע של הוראה.

יש להבדיל הבדל מוחלט בין העברת פרמטרים למקרו לבין העברת פרמטרים לפונקציה: בעוד שפרמטרים לפונקציה מייצגים ערכים או כתובות של משתנים, פרמטרים למקרו מייצגים קטעי טקסט. למעשה, מנגנון העברת פרמטרים לפונקציה פועל בזמן ריצת התוכנית, ואילו מנגנון העברת פרמטרים למקרו מופעל בזמן ביצוע האסמבלי, והוא מטפל בהחלפת קטעי טקסט: בזמן שכפול המקרו, הטקסט של הפרמטר האקטואלי מחליף את הפרמטר הפורמלי המתאים.

לדוגמה, בתכנית שבה זומן המקרו:

```
swap  a,  b
```

לאחר ההידור והקישור של התכנית יוחלפו הפרמטרים בכתובות של a , b, בקובץ הריצה ובהוראה המתייחסת לפרמטר first תוצב הכתובת [0000] ובהוראה המתייחסת לפרמטר second תוצב הכתובת [0001]. באופן דומה, בתכנית שבה זומן המקרו writeChar 'A' המציג תו על הצג, מוחלף הפרמטר char בתו 'A'. מובן שעלינו להקפיד להעביר למקרו מחרוזת תקינה, אם איננו רוצים לקבל שגיאה. לדוגמה, הזימון של

```
writeChar add
```

יגרום לשגיאה, משום שהמהדר יחליף את הפרמטר char במילה add ונקבל את ההוראה השגויה:

```
MOV DL, add
```

פרמטרים פורמליים משמשים לייצוג ערכים מספריים או מחרוזות, אבל הם יכולים לשמש גם לייצוג שמות אוגרים, ואפילו לייצוג קוד פעולה והוראות.

לדוגמה, נוסיף למקרו swap את שם האוגר בו נשתמש:

```
swap  MACRO tempreg, first, second
   LOCAL in_order                          ; הגדרת תווית
   mov tempreg, first
   cmp tempreg, second
   jnb  in_order
```

```
    xchg tempreg, second
    mov  first, tempreg
in_order:
ENDM
```

כעת נוכל לכתוב, למשל, קריאה למקרו שבו נשתמש באוגר BL או CL :

```
swap  BL,  a,  b
swap  CL,  b,  c
```

מספר הפרמטרים האקטואליים לא חייב להיות שווה למספר הפרמטרים הפורמליים. אם מספר הפרמטרים האקטואליים גדול ממספר הפרמטרים הפורמליים שמופיע בכותרת של המקרו, המהדר מתעלם מהפרמטרים המיותרים. כאשר המצב הפוך, ומספר הפרמטרים האקטואליים קטן ממספר הפרמטרים הפורמליים, מוכנס ערך ריק לכל פרמטר פורמלי עבור כל פרמטר אקטואלי שחסר.

7.9.4 מקרו להכפלת קוד והגדרת נתונים

בשפות עיליות, כאשר מעוניינים לחזור ולבצע סדרת הוראות בתכנית, משתמשים בלולאה. למעשה לולאה חוסכת את הצורך לכתוב את אותן ההוראות כמה פעמים.
אסמבלר מספק הוראות מקרו מיוחדות המאפשרות לקבל תוצאה דומה לזו המתקבלת בלולאה, ובאמצעותן מוכפלות ההוראות שבגוף הלולאה כמה פעמים.
המקרו-משכפל REPT (REPeaT) :

```
REPT ביטוי
סדרת ההוראות שיש לחזור עליהן
ENDM
```

הביטוי הוא ביטוי מתמטי המחזיר ערך מספרי, אשר מציין את מספר הפעמים שיש לשכפל את ההוראות. לדוגמה המקרו הזה :

```
REPT  3
   inc  ax
ENDM
```

ייפרש על-ידי האסמבלר לסדרת ההוראות:

```
inc ax
inc ax
inc ax
```

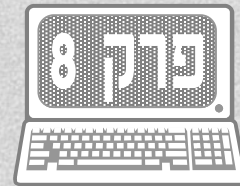
פרישת ההוראות תתבצע בזמן התרגום, לכן ביצוע לולאה בדרך זו מהיר יותר מאשר כתיבת הוראות למימוש לולאה וביצוען בזמן ריצה. בשיטה כזו, אנו חוסכים את זמן העברת הבקרה ובדיקת התנאים ליציאה מלולאה. עלינו להיזהר ולהימנע משימוש במקרו משכפל כאשר מספר החזרות גדול מאוד. בדרך כלל משתמשים במקרו משכפל לא כדי לשכפל קוד אלא כאשר מעוניינים להגדיר ערכים בזיכרון.

דוגמה 7.16

להלן קטע תכנית שמשתמשת בהנחיה '=' . במקטע הנתונים נרשום:

```
.DATA
   x  DB  0
   y  DB  0FFh
asciicode = 'A'
        rept 10
              DB asciicode
              asciicode = asciicode + 1
ENDM
```

כדי לראות את תוצאת ביצוע התכנית שמכילה מקרו זה, התבוננו במפת הזיכרון לאחר אתחול מקטע הנתונים.



עיבוד מחרוזות ובלוקים של נתונים

בשפת אסמבלי יש הוראות מיוחדות המטפלות במחרוזות באמצעותן יישומים מעבדים טקסט. מחרוזת היא רצף של תווים, שבה כל תו מיוצג בקוד ASCII (ראו פרק 2). הפעולות הנפוצות על מחרוזות הן: העתקת מחרוזת ממקום למקום, חיפוש תו במחרוזת, עדכון מחרוזת, השוואה של שתי מחרוזות וכדומה. כמו-כן, מאחר ותו מיוצג כמספר בקוד ASCII, ומאחר שתווים עוקבים (למשל: a, b, c, וכו') מיוצגים ע"י מספרים עוקבים אפשר לבצע חישובים אריתמטיים וכך למצוא את התו העוקב לתו מסוים או את התו הקודם לו, ואפשר לחשב ערך של ספרה בין 0 ל-9 על-ידי חיסור קוד ASCII של הספרה 0 מקוד ASCII של הספרה הרצויה וכדומה.

אוסף הוראות המחרוזת של המעבד 8086 כולל קבוצה של שבע הוראות מיוחדות, שתורמות לפישוט תהליך פיתוח התכנית (אם כי ניתן לממשן גם באמצעות הוראות אחרות). ההוראות לטיפול במחרוזת יכולות לשמש גם לעיבוד בלוקים של נתונים שאינם מכילים מחרוזות אלא מספרים בינאריים מטיפוס בית או מטיפוס מילה. לפני שנציג את ההוראות לטיפול במחרוזת נתאר כיצד מצהירים בשפת אסמבלי על מחרוזת וכיצד מאחסנים אותה.

8.1 הגדרת מחרוזות בשפת אסמבלי

מחרוזת בשפת אסמבלי מיוצגת כמערך של תווים מטיפוס בית או מטיפוס מילה. לדוגמה, שלוש ההצהרות הבאות הן שקולות, ובכולן אנו מגדירים מחרוזת באורך 5 תווים, המאותחלת לערך "abcde":

```
str1  DB "abcde"
str2  DB 'abcde'
str3  DB "a", "b", "c", "d", "e"
```

מחרוזת נרשמת בין מירכאות או בין גרשים יחידים. לכל מחרוזת שהגדרנו לעיל יוקצו 5 בתים, ובכל בית יאוחסן קוד ASCII של התו. הנה דוגמה נוספת, המגדירה מחרוזת מטיפוס מילה:

```
str4  DW "a", "b", "c", "d", "e"
```

לפי ההוראה הזו לכל תו תוקצה מילה שבה הבית התחתון מכיל את קוד ASCII של התו עצמו והבית העליון מכיל 0. איור 8.1 מתאר מפת זיכרון שמכילה מחרוזת בת 5 תווים מטיפוס בית, ומחרוזת בת 5 תווים מטיפוס מילה.

```
ds: 0000  61  62  63  64  65  61  00  62  abcdea  b
ds: 0008  00  63  00  64  00  65  00  4C    c  d  e  L
ds: 0010  CD  21  00  00  00  00  00  00  = ?
ds: 0018  00  00  00  00  00  00  00  00
```

איור 8.1
מפת זיכרון המכיל מחרוזות

אחת התכונות החשובות של מחרוזת היא אורכה. כדי לחשב את האורך של מחרוזת נוסיף למחרוזת תו שיציין סוף מחרוזת. במקרה כזה נוכל לחשב את אורך המחרוזת כהפרש בין הכתובת של התו המסיים במחרוזת לכתובת התו הראשון בה. לדוגמה,

```
string1  DB 'abcde$'
strLen   EQU $ – string1
```

המשתנה string מכיל את כתובת תחילת המחרוזת, וכך ערכו של הקבוע strLen מחושב כהפרש בין כתובת התו $ לכתובת תחילת המחרוזת string1.

דרך אחרת היא להגדיר גודל קבוע למחרוזת, ואם היא קצרה מהאורך שהוקצה לה – לרפד את סוף המחרוזת בתווי רווח. לדוגמה:

```
string2  DB 80  DUP(?)
```

שאלה 8.1

הגדירו את המחרוזות הבאות:

א. מחרוזת בגודל 10 תווים מטיפוס בית שמאותחלים לתו רווח

ב. מחרוזת בגודל lenStr (קבוע המגדיר 100 איברים) מטיפוס מילה

ג. מחרוזת מטיפוס בית שמאותחלת למחרוזת "Hello!"

8.2 מבנה של הוראות מחרוזות

הוראות המחרוזת הן ללא אופרנדים והן פועלות בהתאם לכמה פרמטרים שצריך להגדיר לפני ביצוע הוראת המחרוזת עצמה.

הפרמטרים בהם משתמשים בהוראות מחרוזת הם אוגרי אינדקס המשמשים כמצייני כתובת:

א. האוגר SI מגדיר היסט (כתובת אפקטיבית) של מחרוזת המקור ביחס לכתובת מקטע הנתונים DS;

ב. האוגר DI מגדיר היסט (כתובת אפקטיבית) של מחרוזת היעד ביחס למקטע הנתונים ES.

ג. דגל הכיוון DF באוגר הדגלים, המציין את כיוון ההתקדמות, כלומר קובע אם הפעולה על מחרוזת תתבצע החל מהתו הראשון לתו האחרון או להיפך – מהתו האחרון לתו הראשון.

ד. אפשרות לשימוש באופרטור חזרה, באמצעותו נממש מבנה הדומה ללולאה.

בדוגמאות שנביא בספר זה, נשתמש במקטע נתונים אחד שיכיל את מחרוזת המקור וגם את מחרוזת היעד, לכן נאתחל את שני אוגרי המקטעים, DS ו-ES לאותה כתובת:

```
mov ax, @DATA
mov ds, ax
mov es, ax
```

בהמשך הפרק נסביר כיצד משתמשים בפרמטרים אלה על-ידי הצגת כל הוראה והוראה מהוראות המחרוזות. נתחיל בהוראות להעתקת מחרוזות.

8.3 העתקת מחרוזות – ההוראה MOVS

ההוראה MOVS מעתיקה בית מהתא, שכתובתו נתונה באוגר SI, לתא שכתובתו נתונה באוגר DI. קיימות שתי הוראות להעתקת תו:

- העתקת תו מטיפוס בית:

```
MOVSB
```

- העתקת תו מטיפוס מילה:

MOVSW

ההוראות האלה מבצעות את הפעולה:

ES:[DI] ←DS:[SI]

כלומר, העתקה של תו (מטיפוס בית או מילה), עליו מצביע האוגר SI, למיקום חדש עליו מצביע האוגר DI.
ביצוע שתי ההוראות האלה אינו משפיע על אוגר הדגלים.

כאמור, למרות שאין להוראות אלה אופרנדים, הן משתמשות באוגרי האינדקס SI ו-DI כמצביעים על איבר במחרוזת המקור ואיבר במחרוזת היעד. לאחר שמתבצעת ההעתקה של האיבר המתאים, מעודכנים גם אוגרי האינדקס, כך שיצביעו על האיבר הבא וזאת בהתאם למצבו של דגל הכיוון (Direction Flag) שהוא אחד מתשעת הדגלים של אוגר הדגלים במעבד 8086). האלגוריתם הבא מתאר את הפעולה MOVS:

```
העתק בית או מילה  מ-DS[SI] ל-ES[DI].
אם DF=0 (כיוון קדימה) אזי
    SI = SI+1
    DI = DI+1
אחרת (כיוון אחורה)
    SI = SI–1
    DI = DI–1
סיום-אם
```

ניתן לקבוע לדגל הכיוון את הערך 1 או 0, באמצעות ההוראות CLD ו-STD.

ההוראה CLD (Clear Direction flag)

להוראה CLD אין אופרנדים; היא משימה את הערך 0 בדגל הכיוון DF (Direction Flag), וכך היא קובעת שההעתקה תתבצע בסדר עולה, כלומר, תחילה יועתק האיבר הראשון, אחריו יועתק האיבר השני וכך הלאה.

ההוראה STD (SeT Direction flag)

להוראה STD אין אופרנדים; היא משימה את הערך 1 בדגל הכיוון DF וקובעת בכך כי במקרה כזה יועתק תחילה האיבר האחרון, אחריו יועתק האיבר שלפני האחרון וכך הלאה.

שתי ההוראות האלה שייכות לקבוצה של הוראות המאתחלות דגל לערך מסוים ומשתמשים בהן לביצוע פעולות הנקראות פסיקות, אותן נציג בפרק הבא.

כדי לבצע את ההוראה MOVS עלינו לרשום תחילה כמה הוראות המאתחלות את אוגרי האינדקס ואת דגל הכיוון. לדוגמה, בקטע התכנית הבא מועתק האיבר שעליו מצביע האוגר SI למיקום עליו מצביע האוגר DI:

```
mov  si,   OFFSET a        ; SI מצביע על איבר ראשון במחרוזת a
mov  di,  OFFSET b         ; DI מצביע על איבר ראשון במחרוזת b
cld                        ; דגל כיוון מוגדר כ-"קדימה"
movsb                      ; ביצוע  העתקה
```

כדי לקבוע באוגרי האינדקס את כתובות ההתחלה של המחרוזות אנו משתמשים באופרטור offset (שהוצגה בפרק 6) וכך ערכו של האוגר SI מכיל את הכתובת שבה מתחילה המחרוזת a וערכו של האוגר DI מכיל את הכתובת שבה מתחילה המחרוזת b. כתובות אלה מוגדרות כהיסט שלהן מכתובת הבסיס של מקטע הנתונים DS.

שאלה 8.2

רשמו הוראות אחרות שיעבירו לאוגרי האינדקס SI ו-DI את כתובות ההתחלה של המחרוזות.

שאלה 8.3

בהנחה שערכו של דגל הכיוון הוא 1, מה יהיה ערכם של SI ו-DI לאחר ביצוע ההוראה MOVSB ומה יהיה ערכם לאחר ביצוע ההוראה MOVSW?

8.4 חזרה על פעולת ההעתקה

כדי להעתיק קבוצה של איברים מוסיפים קידומת מיוחדת, בשם **REP**, הנרשמת לפני הוראת ההעתקה, לדוגמה

REP MOVSB

המילה REP (קיצור המילה REPeat) חוסכת את הצורך לכתוב לולאה להעתקת n תווים ממחרוזת למחרוזת. כדי להשתמש בקידומת REP צריך לאתחל קודם-כל את ערכו של אוגר CX כך שיכיל את מספר הפעמים שהוראת MOVS צריכה להתבצע. האלגוריתם הבא מתאר את הפעולות המתבצעות בהוראה REP MOVSB :

כל עוד $CX \neq 0$ בצע
  בצע MOVSB
  חשב CX = CX – 1

דוגמה 8.1

נכתוב תכנית שתעתיק את מחרוזת a למחרוזת b. שתי המחרוזות הן מטיפוס בית ובאורך 7.

```
.MODELL SMALL
.STACK 100h
MAX EQU 7
.DATA
   a  DB "abcdefg"
   b  DB  MAX dup(?)
.CODE
start:
                                          ; אתחול מקטעי נתונים
   mov  ax,  @DATA
   mov  ds,  ax
   mov  es,  ax
   mov  si,   OFFSET a                    ; SI מצביע על איבר ראשון במחרוזת a
```

```
    mov  di,  OFFSET b          ; DI מצביע על איבר ראשון במחרוזת b
    cld                         ; דגל כיוון מוגדר קדימה
    mov  cx,  MAX               ; מונה CX מותחל למספר האיברים במחרוזת
    rep    movsb                ; ביצוע חוזר של העתקה
  mov  ax,  4c00h               ; סיום התכנית
  int    21h
END start
```

נסביר את התכנית:

א. תחילה מאתחלים את האוגרים של מקטעי הנתונים DS ו-ES לאותו מקטע נתונים, כך שאוגרי האינדקס SI ו-DI יצביעו על היסט בתוך מקטע DS.

ב. לאחר מכן מאתחלים את אוגרי האינדקס SI ו-DI.

ג. כעת קובעים את כיוון ההעתקה של התווים, על-ידי ההוראה CLD. בתכנית זו אין חשיבות לסדר העתקה, לכן אפשר לרשום STD במקום CLD, אולם לא תמיד זה כך.

ד. כדי לקבוע כמה פעמים תתבצע ההוראה MOVSB, נשים באוגר CX את מספר התווים שיש להעתיק (מספר התווים במחרוזת a).

ה. ההוראה האחרונה שמבצעת את ההעתקה

REP MOVSB

תבצע את פעולת ההעתקה, תו אחר תו, ובכל העתקה כזו תפחית ערכו של CX באחד; כל זה יתבצע כל עוד ערכו של אוגר CX שונה מאפס.

שאלה 8.4

א. שנו את התכנית שהוצגה ברוגמה 8.1 כך שהיא תעתיק מחרוזת של מילים בגודל 10 בכיוון יורד.

ב. הסבירו מה יקרה אם נעתיק ל-b מחרוזת a שאורכה קצר מ-MAX? מה יקרה אם אורך מחרוזת a יהיה יותר מ-MAX?

ג. האם אפשר להשתמש בהוראה זו כדי להעתיק מַעֲרך שאיבריו הם מטיפוס בית ומכילים מספרים מכוונים, למערך אחר?

ד. כתבו תכנית המשרשרת מחרוזת באורך 10 תווים לעצמה ויוצרת מחרוזת חדשה.

8.5 כתיבת תווים במחרוזת – ההוראה STOS (Store a String)

ההוראה STOS מעתיקה את התוכן של האוגר AL למיקום מסוים בזיכרון, עליו מצביע האוגר DI, ומאפשרת להשתמש בה, בצירוף של אופרטור חזרה, כדי למלא קטע של זיכרון בנתון מטיפוס בית או מילה. משמעות ההוראה

STOSB

היא: ES:[DI] ← AL, כאשר DI מצביע על המיקום בו יאוחסן הנתון.

קיימת הוראה נוספת, המעתיקה תוכן אוגר AX למיקום מסוים מטיפוס מילה, זוהי ההוראה

STOSW

שמשמעותה: ES:[DI] ← AX.

לשתי ההוראות האלה אין אופרנדים ואין להן השפעה על אוגר הדגלים.

האלגוריתם הבא מתאר ביצוע של ההוראה STOSB:

```
בצע MOV byte ptr ES:[DI], AL
אם DF=0 אזי
        DI = DI + 1
אחרת
        DI = DI – 1
```

שאלה 8.6

רשמו אלגוריתם שיתאר את הביצוע של ההוראה STOSW

דוגמה 8.2

נכתוב תכנית המאחסנת בזיכרון מחרוזת המורכבת מ-10 פעמים התו c.

```
. model  small
.STACK 100h
.data
   chr DB 10 dup(?)
.code
```

```
start:
                                              ; אתחול מקטעי נתונים
    mov  ax,  @DATA
    mov  ds,  ax
    mov  es,  ax
    mov  di,  offset chr                      ; DI אתחול אוגר
    mov  cx,  10                              ; קביעת מספר התווים
    cld                                       ; קביעת כיוון ההתקדמות קדימה
    mov  al,  'c'                             ; al ← 'c' התו
    rep  stosb                                ; ביצוע חוזר של פעולה לאחסון תו במחרוזת
  mov  ax,  4c00h                             ; סיום התכנית
  int   21h
END start
```

8.6 קריאת תו ממחרוזות – ההוראה LODS (Load a String)

שתי הוראות נוספות מעתיקה את תוכנו של איבר ממחרוזת עליו מצביע האוגר SI לאוגר AL:

הוראה המעתיקה איבר ממחרוזת שהיא מטיפוס בית

LODSB

שמשמעותה AL ← byte ptr DS:[SI]

ובאופן דומה איבר במחרוזת יכול להיות גם מטיפוס מילה ובמקרה כזה תוכנו מועתק לאוגר AX. משמעות ההוראה

LODSW

היא: AL ← word ptr DS:[SI]

הביצוע של שתי ההוראות האלה אינו משפיע על אוגר הדגלים.

האלגוריתם הבא המתאר ביצוע LODSB :

השם MOV AL , ES:[SI]
אם DF=0 אזי
SI = SI + 1
אחרת
SI = SI – 1

בדרך-כלל **לא** משתמשים בקידומת REP לפני הוראות LODS. הסיבה לכך היא שהשילוב של הקידומת REP עם ההוראה LODSB (או LODSW) גורם לביצוע סדרת הפעולות הבאות : תחילה יועתק הבית הראשון מבלוק הנתונים לאוגר AL, אחר יועתק הבית השני וערכו של AL ישתנה, וכך הלאה. אחרי שהפעולה תסתיים, נקבל באוגר AL רק את ערך התא האחרון שהועתק וכל שאר הערכים יימחקו. לעומת זאת, ביצוע חוזר של שתי ההוראות, LODS ו-STOS, בזו אחר זו, מאפשר להעתיק נתונים מבלוק אחד ולעדכן אותם לפני שהם מאוחסנים בבלוק אחר.

דוגמה 8.3

נכתוב תכנית שתעתיק את תווי המחרוזת a למחרוזת b כאשר היא משנה כל תו רווח למקף (לתו '-').

```
.model  small
.stack  100h
   MAX EQU 7                              ; מספר איברי מחרוזת
.data
   a  DB  "12  45  7"                     ; מקור מחרוזת
   b  DB  MAX dup(?)                      ; מחרוזת יעד
.code
start:
                                          ; אתחול מקטע נתונים
         mov  ax,  @DATA
         mov  ds,  ax
         mov  es,  ax
```

```
; אתחול פרמטרים: אוגרי אינדקס, דגל הכיוון ומונה לולאה
        mov  si,  offset a
        mov  di,  offset b
        cmp  cx,  0
        cld
; ביצוע העתקה ועדכון
again:
 lodsb
 cmp  al, ' '                      ; al=' '  השווה
 jne   cont                        ; cont  עבור לביצוע ההוראה עם התווית  al<> ' '  אם
 mov  al, '-'                      ; ערכו של al הוא ' ' ולכן יש לשנות ל '-'
cont:   stosb                      ; b[di] = al
   inc  cx                         ; cx ← cx + 1
   cmp  cx,  MAX                   ; האם סיימנו לבדוק את כל איברי המחרוזת?
   jb  again                       ; אם לא עבור להוראה אליה מוצמדת התווית again
 mov   ax,  4c00h                  ; סיום התכנית
 int    21h
end start
```

8.7 השוואת מחרוזות – ההוראה CMPS (CoMPare String)

כדי להשוות שתי מחרוזות משתמשים בהוראה CMPS. הוראה זו משווה בין שני תווים, בהתאם לערכי קוד ASCII של התווים. קיימות שתי הוראות להשוואה: האחת מטפלת באיברים מטיפוס בית והשנייה באיברים מטיפוס מילה:

```
CMPSB   ; השוואה בין שני איברים מטיפוס בית
CMPSW   ; השוואה בין שני איברים מטיפוס מילה
```

משמעות ההוראות היא: האם התו שאוגר SI מצביע עליו שווה לתו שאוגר DI מצביע עליו? כלומר, האם ES:[DI] = DS:[SI] ?

בדומה להוראה CMP, גם הוראות אלו משפיעות על כל דגלי המצב: ZF, CF, OF, SF, AF, PF.

האלגוריתם הבא מתאר את פעולת ההוראה CMPSB:

בצע CMP byte ptr ES:[DI],DS:[SI]
אם DF=0
SI = SI + 1
DI = DI + 1
אחרת
SI = SI – 1
DI = DI – 1
סיום-אם

כדי לבדוק את תוצאת ההשוואה, נבדוק את מצב הדגלים ובהתאם לערכם נחליט איזו פעולה יש לבצע.

דוגמה 8.4

בקטע התכנית הבא משווים את התו הראשון של מחרוזת a לתו הראשון של מחרוזת b, ובהתאם לתוצאת הבדיקה משימים 0 (אם התווים שונים) או 1 (אם התווים שווים) באוגר AX:

```
    mov  si,  offset a
    mov  di,  offset b
    mov  ax,  1
    cld
    cmpsb                ; האם התווים שווים?
    jz  equal            ; אם התווים שווים עבור להוראה שכתובת equal
                         ; אם התווים לא שווים
    mov  ax,  0
    jmp  endcmp
equal:
```

כדי להרחיב את הבדיקה ולהשוות שתי מחרוזות, יש לבדוק שני תנאים:

- המחרוזות שוות בגודלן
- כל תו במיקום i במחרוזת הראשונה שווה לתו במיקום i במחרוזת השנייה.

לשם כך נשתמש בהוראה CMPSB, עם קידומת המאפשרת חזרה, אך לא בהוראה REP. אם נשתמש בקידומת REP יתבצע האלגוריתם הזה:

```
לכל איבר מ-i עד n בצע:
    השווה DS:[SI] = ES:[DI] ועדכן את הדגלים בהתאם
```

לפי אלגוריתם זה, מצב הדגלים בסיום הלולאה מכיל מידע רק על תוצאת ההשוואה של הזוג האחרון. לכן בהוראה CMPS נשתמש בקידומת חזרה מותנית REPZ (REPeat while Zero) או REPE (REPeat while Equal), שמשמעותן היא: בצע פעולת CMPSB כל עוד נותר זוג תווים, וזוג התווים שנבדק הוא שווה.

REPE CMPSB
REPZ CMPSB

האלגוריתם הבא מציג את פעולת REPZ:

```
כל עוד CX ≠ 0 וגם ZF≠ 0 בצע:
    בצע CMP ES:[DI],DS:[ SI]
    אם DF=0
        SI  = SI + 1
        DI = DI + 1
    אחרת
        SI = SI – 1
        DI = DI – 1
    סיום-אם
```

כמו כן קיימות הקידומות

REPNZ (REPeat while Not Zero)

או REPENE (REPeat while Not Equal)

בהן התנאי ליציאה מהלולאה הוא:

כל עוד $CX \neq 0$ וגם $ZF \neq 1$ בצע

כאשר משתמשים באחד מסוגי הקידומות הללו, יש לרשום אחריהן הוראה שתבדוק את מצב הדגלים, כדי לדעת אם המחרוזות שוות.

דוגמה 8.5

נכתוב תכנית שתבדוק אם שתי המחרוזות שוות, כאשר ההנחה היא שהן שוות באורכן. אם המחרוזות שוות נשים FFh באוגר al, אחרת נקבע את ערכו ל-0.

```
.MODEL SMALL
.STACK 100h
.DATA
    a DB "1234567 ″
    b DB "1234567 ″
.CODE
start:
; אתחול מקטע הנתונים
    mov  ax,  @DATA
    mov  ds,  ax
    mov  es,  ax
; אתחול פרמטרים הדרושים לביצוע ההשוואה
  mov  si,  OFFSET a        ; אתחול מצביע למחרוזת המקור
  mov  di,  PFFSET b        ; אתחול מצביע למחרוזת היעד
  mov  cx, 7                ; אתחול מונה מספר ההשוואות
  cld                       ; קביעת כיוון ההשוואה
```

```
  repz  cmpsb                    ; השוואת שתי המחרוזות
                                 ;בדיקת תוצאת ההשוואה
jne  else_if
    mov  al, 0FFh`               ; המחרוזות שוות
    jmp  end_if
else_if:                         ; המחרוזות שונות
    mov al, 0
end_if:
  mov  ax,  4c00h                ; סיום התכנית
  int    21h
end start
```

שאלה 8.7

כתבו תכנית שתבדוק אם שתי מחרוזות מטיפוס בית הן באותו אורך. התכנית תספור את מספר המקומות בהן המחרוזות שונות זו מזו. לדוגמה: במחרוזות abcdef ו- abedcx יש 2 מקומות כאלה (המקומות 2, 4).

8.8 חיפוש תו במחרוזת הוראה SCAS (SCAn String)

ההוראה **SCAS** (SCAn String) סורקת מחרוזת בחיפוש אחר בית (או מילה) מסוימת המאוחסנת באוגר AL (או באוגר AX). להוראה זו שתי גרסאות:

SCASB

SCASW

בהוראה SCASB תוכן הבית עליו מצביע האוגר DI מושווה לתוכן אוגר AL;
בהוראה SCASW תוכן המילה עליה מצביע האוגר DI מושווה לתוכן אוגר AX.

ביצוע הוראות אלה משפיע על דגלי המצב: CF, OF, SF, ZF, AF, PF.

להלן האלגוריתם שמתאר את ההוראה SCASB :

בצע השוואה AL, ES:[DI] CMP
אם DF=0 אזי
DI := DI + 1
אחרת
DI := DI – 1
סיום-אם

דוגמה 8.6

נכתוב קטע תכנית שיבדוק האם התו '5' נמצא במחרוזת נתונה, שאורכה 7. אם התו '5' נמצא במחרוזת נשים FFh באוגר al, אחרת נקבע את ערכו ל-0.

```
mov  di,  offset a
mov  al,  '5'                    ; תו שאותו מחפשים
mov  cx,  7
cld
repnz  scasb
jne  else_if
    mov  al, 0FFH
    jmp  end_if
else_if:  mov al, 0
end_if:
```

שאלה 8.8

כתבו תכנית שתבדוק אם מחרוזת נתונה היא פלינדרום.

שאלה 8.9

א. כתבו תכנית שתמחק את כל תווי הרווח ממחרוזת נתונה. מחיקת הרווחים תתבצע על-ידי העתקת התווים הרשומים בה (מלבד הרווחים) למחרוזת חדשה.

ב. חזרו וכתבו תכנית למחיקת רווחים ממחרוזת, אך הפעם עדכנו את אותה מחרוזת. רמז : הזיזו תווים במחרוזת וכתבו אותם במקום תווי הרווח.

8.9 טבלאות תרגום וההוראה XLAT (״קיצור״ של translate)

אחת הפעולות השכיחות בעיבוד תווים, היא החלפת תו בקוד מסוים הנתון כטבלת תרגום. טבלת ASCII היא דוגמה לטבלת תרגום הממירה כל תו במספר. אחד השימושים בטבלת תרגום הוא בתחום הצפנת מידע. השימוש בהוראה XLAT מייעל את ביצוע המשימות המחייבות גישה חוזרת לטבלת תרגום המאוחסנת בזיכרון.

להוראה **XLAT** אין פרמטרים, והיא שולפת מטבלה איבר שמספרו נתון באוגר AL; כדי להשתמש בהוראה זו יש לאתחל את הפרמטרים האלה:

- אוגר BX מכיל את הכתובת של התחלת טבלת התרגום;
- אוגר AL מכיל מציין מקום של איבר.

בעת ביצוע ההוראה XLAT נקרא מטבלת התרגום הנתון שכתובתו האפקטיבית מחושבת כסכום של BX + AL והוא מאוחסן באוגר AL (כך שערכו של האינדקס אובד). בהוראה זו גודל טבלת התרגום יכול להיות לכל היותר 256 ערכים (הערך המכסימלי שניתן לאחסן באוגר AL).

דוגמה 8.7

נתונה טבלת הצפנה לספרות מ-0 עד 9:

נתון	0	1	2	3	4	5	6	7	8	9
קוד להמרה	4	6	9	5	0	3	1	8	7	2

נגדיר במקטע נתונים טבלת תרגום

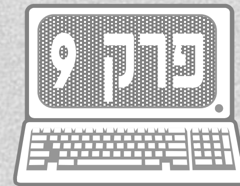
```
.DATA
table  DB  '4695031872'
```

כעת נכתוב קטע קוד שיתרגם את הספרה 3 לקוד המתאים:

```
mov  bx, offset table
mov  al,  3                ; AL = 3
xlat                       ; AL = 5
```

שאלה 8.10

כתבו תכנית שתצפין הודעה (מחרוזת) בצורה מעגלית: כל אות של ההודעה תוחלף באות אלפבית הנמצאת 5 מקומות ממנה באלפבית; דוגמה: לאחר הצפנת המחרוזת "adrtx" נקבל את המחרוזת "fiwyc".



פסיקות וקלט-פלט

9.1 הוראות קלט-פלט

עד כה כתבנו תכניות בהן הנתונים אוחסנו בזיכרון. כדי לראות את תוצאות ההרצה השתמשנו בתכנה לניפוי שגיאות (debugger) המתאר את תוכן הזיכרון ואת תוכן האוגרים בזמן ריצה. סביבה זו מתאימה לפיתוח תכנית, אך אינה מתאימה כאשר צריך להריץ יישומי משתמש, כלומר, לקלוט נתונים מהמשתמש ובהתאם להציג לו את המידע המתקבל מהתכנית. התקני הקלט הנפוצים במחשבי PC, ביישומי משתמש, הם מקלדת, עכבר, דיסק, תקליטור וכדומה; התקני הפלט הנפוצים הם צג, מדפסת, דיסק, תקליטור, רמקול, וכדומה. כדי לקלוט ולהציג מידע בהתקני קלט/פלט עלינו לדעת לא רק את הכתובות של ההתקנים, אלא גם כיצד לתאם בין קצב העבודה המהיר של המעבד לעומת קצב העבודה האיטי של התקן קלט/פלט, לתאם רמות זרם ומתח בהם פועלים ההתקנים השונים וכדומה. שפות עיליות מכילות הוראות קלט והוראות פלט שמשחררות את המתכנת מהצורך לדעת פרטים רבים על המבנה של ההתקנים ההיקפיים. בזמן הרצה מזומנות שגרות שירות של מערכת ההפעלה שדואגת שהפעולות יתבצעו. בשפת אסמבלי קיים מנגנון דומה: מערכת ההפעלה מספקת שגרות שירות המתווכות בין התכנית בשפת-סף לבין החומרה. המתכנת רושם הוראות מיוחדות הנקראות פסיקות, אותן נתאר בהמשך, אשר מזמנות את שגרות השירות ומבצעות את הפעולות הדרושות להפעלת החומרה ולהעברת מידע מהתקני הקלט אל התכנית ומהתכנית להתקני הפלט.

9.2 שימוש בשגרת השירות של DOS

בסעיף זה נדון בחלק קטן מהשירותים של מערכת ההפעלה DOS לכתיבת יישומים הכוללים קלט-פלט. DOS קיצור של **D**isk **O**perating **S**ystem, הינו מערכת הפעלה למחשב המבוססת על ממשק משתמש טקסטואלי, שפותחה בשנות ה-80 למחשבים אישיים והייתה בשימוש נרחב במחשבים אישיים עד אמצע שנות ה-90. כיום מערכות ההפעלה שבשימוש הן מערכות הפעלה המבוססות על ממשק גרפי, לדוגמה מערכת הפעלה "חלונות".

לשגרות השירות אין שם, במקום זאת הן ממוספרות. כדי לקרוא לשגרת שירות נבצע את הפעולות האלה:

1. נשים באוגר AH את המספר של שגרת השירות;
2. נכניס את הערכים המתאימים הדרושים לשגרת השירות;
3. נזמן את הפסיקה INT 21h.

ההוראה INT 21h (בדומה להוראה CALL) היא קריאה לשגרת השירות שהקוד שלה הוכנס לאוגר AH, אך כפי שנתאר בהמשך, מנגנון המימוש של שגרת השירות הוא שונה מימוש שגרת משתמשת שתוארה בפרק הקודם. תחילה נציג כמה שגרות שירות שימושיות למימוש הוראות פלט והוראות קלט.

9.2.1 שירות מס׳ 4Ch: לסיום תכנית

שגרת שירות מס׳ 4Ch לסיום התכנית היא שגרה שהשתמשנו בה בעבר. כפי שציינו במודל SMALL, תכנית המסיימת את פעולתה צריכה ׳להחזיר את השרביט׳ למערכת ההפעלה. תיארנו שם את השיטה שבה התכנית מחזירה את הבקרה ל-DOS. לא ציינו שם בפירוש כי התיאור שהובא הציג את השימוש בשגרה 4Ch של DOS. כעת נרחיב קמעה את ההסבר על השגרה.

השגרה 4Ch של DOS מאפשרת להעביר, בסיום התכנית, מידע המציין אם התכנית הסתיימה בהצלחה. מידע זה מועבר דרך משתנה של מערכת הפעלה בשם ERRORLEVEL. מקובל כי הערך ERRORLEVEL = 0 מציין סיום מוצלח של תכנית, וערכים השונים מ-0 מציינים קוד שגיאה (ומכאן שם המשתנה). כדי להעביר את ערכו של קוד השגיאה, יש לשים ערך זה באוגר AL, לפני הקריאה לשגרה 4Ch. יש לציין כי העברת קוד שגיאה אינה חובה.

להלן קטע תכנית המציג את נוהל הקריאה לשגרה לסיום תכנית ולהעברת קוד שגיאה.

```
mov  al,  0          ;העבר את קוד השגיאה
mov ah,  4ch         ;קבע את מספר שגרת השירות
int    21h           ;זמן את שגרת השירות
```

אפשר כמובן לאחד את שתי ההוראות MOV להוראה אחת: mov ax, 4C00h.

9.2.2 שגרת השירות מס' 2: הצגת תו על הצג

שגרת שירות זו מציגה תו אחד על צג המחשב, במקום שבו נמצא הסמן. בעת הקריאה לשגרה, צריך האוגר DL להכיל את קוד ASCII של התו שיוצג. נוהל הקריאה לשגרה זו מוצג בקטע התכנית הבא:

```
mov  dl,  תו      ; שים את התו שיש להציג באוגר DL
mov  ah, 2        ; קבע את מספר שגרת השירות
int 21h           ; זמן את שגרת השירות
```

דוגמה 9.1

נדגים שימוש בשגרת שירות זו; נכתוב תכנית שתציג על הצג תווים של מחרוזת נתונה. המחרוזת כוללת את אורך המחרוזת n ואחריה סדרה של n אותיות אנגליות גדולות. כל תו יוצג כאות גדולה וכאות קטנה. לדוגמה: אם המחרוזת מכילה את התווים "AFCTDR", הפלט המתקבל הוא: AaFfCcTtDdRr.

פתרון

התכנית שאנו מציגים משתמשת בהוראות מחרוזת ומגדירה שגרה printChar המקבלת כפרמטר את התו שיש להציג והיא מציגה את התו על הצג. אם נתבונן בטבלת קודי אסקי נראה כי לצורך ההמרה מאות אנגלית גדולה לאות אנגלית קטנה, יש להוסיף לקוד ASCII 20h.

```
.MODEL  SMALL
.STACK 100h
.DATA
    str1 db  6,"AFCTDR"        ; מחרוזת להצגת תווים 6
.CODE
start:
                               ; אתחול מקטע הנתונים
```

```
 mov ax, @DATA
 mov ds, ax
                                           ; אתחול משתנים
   xor cx, cx                              ;מונה מספר תווים מאותחל ל-0
   mov si, offset str1                     ; SI מצביע לתחילת המחרוזת
   lodsb                                   ; AL = n,   SI = SI+1
   mov cl, al                              ; CX = n
   cld                                     ; קביעת כיוון ההתקדמות של הסריקה במחרוזת
doloop:
   lodsb                                   ;AL = str[SI]  קריאת תו מהמחרוזת
   cbw                                     ; תרגום בית למילה
   push ax                                 ; דחיפת תו (אות גדולה) להצגה למחסנית
   call printChar                          ; קריאה לשגרה המציגה תו
   add ax, 20h                             ; המרת אות אנגלית גדולה לאות אנגלית קטנה
   push ax                                 ; דחיפת אות קטנה להצגה למחסנית
   call printChar                          ; קריאה לשגרה המציגה תו
   pop ax                                  ; הוצאת איבר ממחסנית
   loop doloop
                                           ; יציאה מהתכנית
 mov ax, 4c00h
 int  21h
                                           ; שגרה המקבלת תו ומציגה אותו על הצג
printChar PROC
   mov bp, sp
   mov dl, [bp+2]                          ; התו להצגה
   mov ah, 2                               ; קביעת שגרת שירות 2
   int  21h                                ; קריאה לשגרת שירות של DOS
   ret 2
printChar ENDP
END start
```

9.2.3 שגרת השירות מס׳ 9: הצגת מחרוזת תווים על הצג

שגרת השירות מס׳ 9 מציגה על הצג מחרוזת של תווים ובכך היא חוסכת את הצורך להציג תווים יחידים בזה אחר זה. שגרה זו מקבלת מחרוזת של תווים המאוחסנת במקטע הנתונים, עליו מצביע האוגר DX. אורך המחרוזת אינו מוגבל; התו '$' מסמן את סוף המחרוזת. למשל: כדי להציג את המחרוזת 'GOOD-BYE', מאחסנים בסגמנט הנתונים את המחרוזת 'GOOD-BYE$'; מובן שהתו '$' לא יוצג.

התבוננו כעת בקטע תכנית המציג נוהל קריאה לשגרה, המציגה על הצג את תוכנה של המחרוזת str:

```
mov  dx,  offset str        ; אוגר DX מצביע למחרוזת שיש להציג
mov  ah,  9                 ; קבע את מספר שגרת השירות
int    21h                  ; זמן את שגרת השירות
```

שאלה 9.1

כתבו תכנית שתציג על הצג את התוכן של מחרוזת שמכילה את שמכם. הגדירו משתנה שיכיל את שמכם ובסיום רשמו "$" ושגרה בשם printString, שתקבל כפרמטר מצביע לתחילת מחרוזת ותציג את המחרוזת על הצג.

9.2.4 שגרת השירות מס׳ 1: קלט של תו מלוח המקשים עם הדהוד

שגרת שירות זו ממתינה להקשה של תו על לוח המקשים. כשהיא מסיימת את פעולתה, יכיל האוגר AL את קוד ASCII של המקש שהוקש. במהלך פעולתה מציגה השגרה על צג המחשב את התו המתאים למקש. פעולה זו נקראת 'הדהוד' (echo). לחיצה ctrl+break מסיימת את הביצוע ומחזירה את הבקרה למערכת ההפעלה DOS. נוהל הקריאה לשגרה מוצג בקטע התכנית הזה:

```
mov  ah,  1                 ; קבע את מספר שגרת השירות
int    21h                  ; זמן את שגרת השירות
```

כאשר שגרת הטיפול בפסיקה תסתיים, AL יכיל את קוד ASCII של המקש.

שאלה 9.2

כתבו תכנית שתקלוט מהמשתמש תו ותציג אותו על הצג. התכנית תשתמש בשגרה getChar הקולטת ומחזירה תו ובשגרה printChar שהוצגה קודם לכן להצגת התו.

9.2.5 שגרת השירות מס' 0Ah: קלט של מחרוזת מלוח המקשים עם הדהוד

שגרה זו ממתינה לקבלת מחרוזת תווים שהמשתמש מקליד. השגרה תחזיר את הבקרה לתכנית שהפעילה אותה, אחרי שהמשתמש יסיים את הקלדת המחרוזת בהקשה על <ENTER>. במהלך קליטת המחרוזת מלוח המקשים, השגרה תציג כל תו שהוקלד על-ידי המשתמש. אם נפלו שגיאות במהלך ההקלדה, המשתמש יכול לתקן אותן לפני ההקשה על <ENTER>. מספר התווים המרבי שהשגרה תקלוט, נקבע על-ידי התכנית שהפעילה את השגרה. במקרה שהמשתמש ינסה להקליד מחרוזת שאורכה עולה על האורך המרבי שנקבע בתכנית, השגרה תתריע, בהשמעת צפצוף, ואף תדחה כל ניסיון כזה. שגרת שירות מס' 0Ah תעביר את המחרוזת לתכנית שהפעילה אותה, בתוספת הבית 0Dh (קוד ASCII של התו CR – Carriage Return) וגם מספר התווים שהמשתמש הקליד לפני ההקשה על <ENTER>.

תכנית המפעילה את השגרה לקליטת מחרוזת בת n תווים, צריכה להקצות בסגמנט הנתונים שלה אזור זיכרון, בגודל n+3 בתים כמפורט בטבלה 9.1.

טבלה 9.1

מבנה זיכרון הדרוש למחרוזת

גודל (בבתים)	מטרה
1	מספר התווים המרבי במחרוזת + 1. מספר זה נקבע על-ידי התכנית המפעילה.
1	מספר התווים שהשגרה קלטה במחרוזת. המספר נקבע על-ידי השגרה.
n+1	מקום עבור המחרוזת (+ הבית 0Dh) שתיקלט על-ידי השגרה.

בעת הקריאה לשגרה, האוגר DX צריך להצביע על תחילת אזור הזיכרון בו הוגדרה המחרוזת. נוהל הקריאה לשגרה מוצג בקטע התכנית הבא, המדגים קליטה של מחרוזת שיש בה 10 תווים לכל היותר.

א. הגדרת משתנה מטיפוס מחרוזת

```
inpBuffer  DB 10            ; אורך המחרוזת 10 תווים
           DB 0             ; מספר התווים שנקלטו בפועל
           DB 11 dup(0)     ; מקום עבור המחרוזת עצמה
```

בתו האחרון של המחרוזת נשמר קוד ASCII (D) של המקש return.

ב. נוהל לקליטת מחרוזת

```
mov dx, inpBuffer   ; אוגר dx מצביע לחוצץ הקלט
mov ah, 1           ; קבע את מספר שגרת השירות
int 21h             ; זמן את שגרת השירות
```

דוגמה 9.2

כדוגמה מסכמת, נציג תכנית הקולטת מחרוזת מהמשתמש ומציגה אותה על הצג. התכנית מגדירה ומשתמשת בשגרות האלה:

השגרה printString – מקבלת פרמטר מטיפוס מחרוזת ומציגה אותו על הצג;

השגרה getString – קולטת ומחזירה מחרוזת;

השגרה printChar – מקבלת כפרמטר תו ומציגה אותו על הצג.

בתכנית הגדרנו שני משתנים:

א. משתני פלט מטיפוס מחרוזת בשם outMessage1 ו-outMessage2; המחרוזות מכילות הודעות להצגה למשתמש.

ב. משתנה פלט מטיפוס מחרוזת בשם inpBuffer; במשתנה נשמרת המחרוזת שנקלטה.

בסוף כל מחרוזת פלט יש לרשום את התווים 0Dh, 0Ah הגורמים למעבר לשורה חדשה. ללא התווים הללו, הפלט יירשם תמיד בתחילת אותה שורה, כלומר, הפלט הקודם יימחק ובמקומו יוצג הפלט החדש. התו "$" מסמן סוף מחרוזת.

```
.MODEL  SMALL
.STACK  100h
.DATA
    outMessage1  DB " input string: ",  0Dh, 0Ah, "$"          ; הודעות פלט
    outMessage2  DB " the string is:",  0Dh, 0Ah, "$"
    inpBuffer     DB  10                                         ; מחרוזת קלט
                  DB  0
                  DB  11 dup(0)
.CODE
start:
                                                                 ; אתחול מקטע הנתונים
    mov  ax,  @DATA
    mov  ds,  ax
                                                                 ; הצגת הודעת פלט ראשונה
    mov  bx,  offset outMessage1
    push  bx
    call    printString
    pop    bx
                                                                 ; קליטת המחרוזת
    mov  bx ,  offset inpBuffer
    push  bx
    call    getString
    pop   bx
                                                                 ; הצגת הודעת פלט שנייה
    mov  bx,  offset outMessage2
    push  bx
    call    printString
    pop  bx
                                                                 ; הצגת המחרוזת שנקלטה
    xor   cx,  cx                                                ; אתחול מספר התווים
    mov  si,  offset inpBuffer+1
```

```
    lodsb
    mov  cl,  al
    cld                                   ; קביעת כיוון קריאת המחרוזת
                                          ; לולאה להצגת תווים
do_loop:
    lodsb                                 ; התו שיש להציג
    cbw
    push  ax
    call  printChar                       ; קריאה לשגרה המציגה תו
    pop  ax
    loop  do_loop
                                          ; יציאה מהתכנית
    mov  ax,  4c00h
    int     21h
                                          ; שגרה המקבלת כפרמטר תו ומציגה אותו על הצג
printChar PROC
    mov  bp,  sp
    mov  dl,   [bp+2]                     ; התו להצגה
    mov  ah,  2
    int     21h
    ret 2
printChar ENDP

                                          ; שגרה המחזירה מחרוזת שהיא קולטת מהמשתמש
getString PROC
    mov  bp,  sp
    mov  dx,  [bp+2]                      ; כתובת המשתנה שבו תאוחסן המחרוזת שתיקלט
    mov  ah,  0Ah
    int     21h
    ret 2
getString ENDP
```

```
; שגרה המציגה מחרוזת
printString PROC
    mov  bp,  sp
    mov  dx,  [bp+2]          ; כתובת המחרוזת להצגה
    mov  ah,  9
    int   21h
    ret 2
printString ENDP

END start
```

שאלה 9.3

כתבו תכנית שתקלוט שתי מחרוזות המכילות ספרות בלבד (בין 0 ל-9), תמיר מחרוזות אלה למספרים ותציג את סכום המספרים.

9.3 מנגנון ביצוע פסיקות במעבד 8086

בתכנית ללא הוראות קלט-פלט, המעבד מבצע ברצף, לכל אחת מהוראות התכנית, את המחזור הבאה-וביצוע. הטיפול בפעולות קלט-פלט של שירותי MS-DOS הוא שונה. כאשר אנו מזמנים שגרת השירות של DOS, ביצוע התכנית מופסק והמשך הביצוע עובר למערכת ההפעלה; שגרת השירות מבוצעת ובסיומה, מערכת ההפעלה מחזירה את הבקרה לתכנית להמשך ביצוע שאר ההוראות. מנגנון זה נקרא **פסיקה** (Interrupt).

פסיקה היא הפסקת הביצוע השוטף של תוכנית , כתוצאה מאירוע פנימי (בתוך ה-CPU) או חיצוני (בקשה מהתקן). כתגובה לאירוע, מתבצע מעבר לקטע קוד שנכתב מראש לטיפול באירוע, הקרוי שגרת שרות הפסיקה – interrupt service routine (ISR). עם סיום הטפול באירוע, מתבצעת חזרה לתוכנית שהופסקה, באופן שזו לא תחוש כלל בהפרעה שחלה בשטף ביצועה. ניתן לחלק את הפסיקות לשני סוגים. הסוג הראשון הוא פסיקה חיצונית המוזעקת על-ידי רכיב חיצוני שאינו חלק מהמעבד, ובסוג זה של פסיקות לא נעסוק בספר זה. לדוגמה פסיקה חיצונית מתרחשת כאשר אנו מכניסים רכיב USB (Universal Serial Bus) ליציאת

מחשב וכתגובה אנו מקבלים הודעה כי חובר רכיב חדש למחשב. הסוג השני הן פסיקות פנימיות היזומות על-ידי המעבד. את הפסיקות הפנימיות ניתן לחלק לשלושה סוגים:

- פסיקות תכנה הן פסיקות המוזעקות באמצעות פקודת אסמבלי int. דוגמה לפסיקת תכנה הן שימוש בשגרות DOS שתיארנו בסעיף הקודם.
- חריגה (exception) היא שגיאה שהתעוררה תוך כדי בצוע פקודה בתוכנית, כגון חלוקה ב-0, או כתיבה לאזור בזיכרון שאליו התכנית אינה מורשה לכתוב.
- מלכודת (trap), היא פסיקה המוזעקת בעקבות הדלקת דגל. לדוגמה, במעבד 8086 קיים דגל בשם TF (Trap Flag) שכאשר הוא 1, מוזעקת פסיקה מספר 1 בסיום ביצוע כל פקודה. פסיקה זו משמשת, למשל, בתוכנות לניפוי שגיאות, למשל כדי להריץ תכנית צעד-צעד (single step). המעבד בודק בסיום מחזור הבאה-וביצוע של כל הוראה אם דגל TF הוא 1, ואם כן הוא עובר למחזור פסיקה ומזעיק את פסקה מס. 1.

מעבד 8086 מקבל בקשות פסיקה ממקורות שונים, וכל מקור דורש טיפול שונה. לכן, מקור הפסיקה מודיע ל-8086 מהו סוג הפסיקה (Type) שהוא מבקש. ה-8086 יכול לטפל ב-256 סוגי פסיקה; כל סוג מצוין באמצעות מספר שערכו בין 0 ל-255.

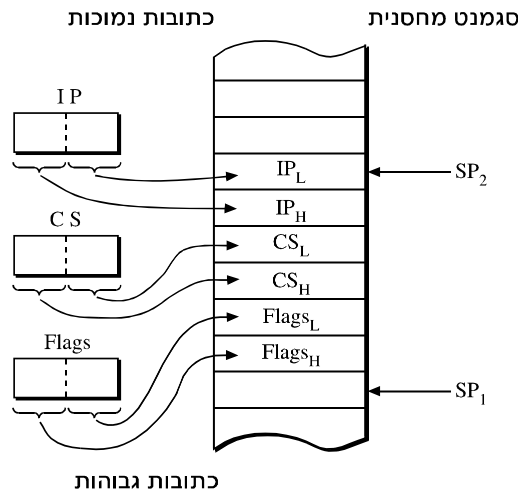
כדי לדעת מהו הטיפול הנדרש בהתאם לסוג הפסיקה, נעזרים בטבלה מיוחדת, הנקראת "טבלת וקטור הפסיקות" (IVT–Interrupt Vector Table) והיא מאוחסנת בזיכרון. הטבלה הזאת כוללת את הכתובות של שגרות שרות הפסיקות, שכל אחת מהן מתאימה לסוג פסיקה אחד. בהתאם לסוג הפסיקה (המצוין על-ידי מספר) המעבד פונה לשורה המתאימה בווקטור הפסיקות (מספר הפסיקה הוא מספר השורה), שולף ממנה את כתובת שגרת השרות המתאימה, וניגש לבצעה.

תהליך ביצוע הפסיקה

כאשר מתקבלת פסיקת חומרה מהתקן חיצוני, או כשקורית חריגה במעבד, או כשמתבצעת פסיקת תכנה יזומה על-ידי הפקודה int, פועל המעבד באופן הבא:

אם הפסיקה היא פסיקת חומרה, המעבד מסיים קודם כול את ביצוע ההוראה שבה הוא עוסק, ורק אחר-כך הוא נענה לבקשת הפסיקה. לאחר מכן הוא דוחף למחסנית את אוגר הדגלים, את האוגר CS ואת האוגר IP. דחיפת הערכים האלה למחסנית מאפשרת ל-8086 לחזור לתכנית שהופסקה, אחרי שהוא מסיים את הטיפול בפסיקה.

התהליך הזה מעורר את השאלה: מדוע ה-8086 שומר במחסנית גם את האוגר CS וגם את תוכן אוגר הדגלים, מדוע אינו מסתפק באוגר IP, כאשר הוא נענה לבקשת פסיקה? שמירת האוגר CS מאפשרת לשגרת הטיפול בפסיקה להימצא במקטע קוד השונה מן המקטע הנוכחי. הפעלת שגרת הטיפול בפסיקה דומה לקריאת שגרה רחוקה. בשני המקרים יש לחזור לתכנית הנמצאת במקטע הקוד המקורי, ולהמשיך את ביצוע התכנית מן המקום שבו היא הופסקה. מכאן: שמירת האוגר CS במחסנית מאפשרת לחזור ולבצע את התכנית. אוגר הדגלים נשמר במחסנית בעת ההיענות לבקשת פסיקה, מפני שלא תמיד בקשות הפסיקה מתוזמנות עם ביצוע התכנית שהופסקה (פסיקת חמרה לדוגמה יכולה להתבצע בכל רגע גם זמן שתכנית כלשהי נמצאת בתהליך הרצה). כאמור, בקשת פסיקה יכולה להופיע מיד אחרי שהמעבד ביצע חישוב כלשהו, למשל: בדיוק כאשר הוא עומד לבדוק את הדגל ZF (אפס) ולהחליט, על-פי ערכו, איזה קטע מתוך התכנית עליו לבצע. באיור 9.1 שלפניכם מוצג מצב הזיכרון לאחר היענות לבקשת פסיקה ב-8086, לפני ביצוע שגרת הטיפול עצמה.



איור 9.1
מצב הזיכרון לאחר ההענות לבקשת פסיקה ב-**8086** (לפני ביצוע שגרת הטיפול עצמה)

שמירת הדגלים במחסנית בעת ההיענות לפסיקה, מאפשרת ל-8086 לשחזר את המצב שבו התכנית הופסקה, ולבדוק מה היה ערכם של הדגלים, גם אם שגרת הטיפול בפסיקה שינתה בינתיים את ערכם.

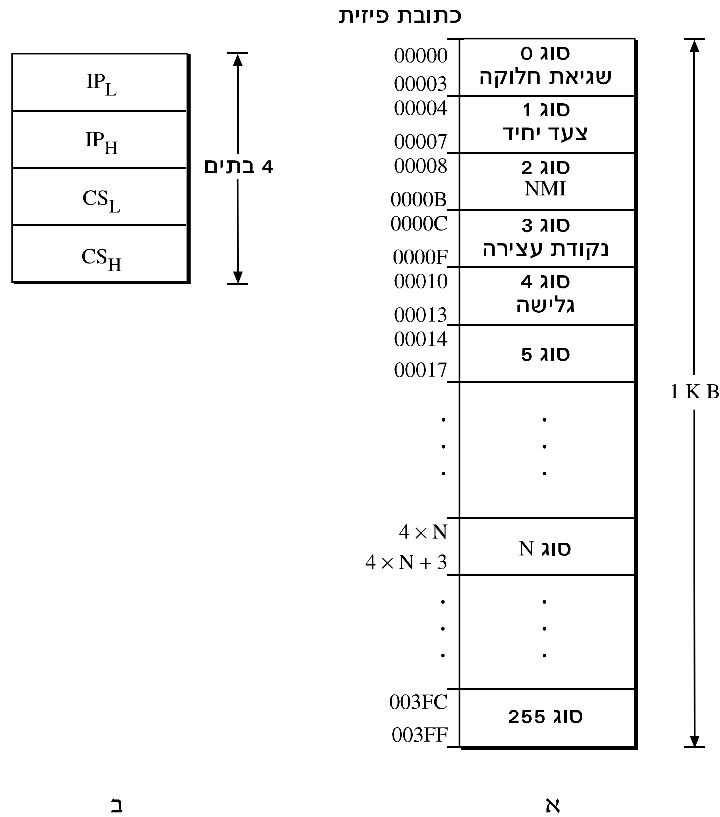
אחרי שערכי ה-CS, ה-IP והדגלים נדחפו למחסנית, המעבד מכבה את הדגלים IF ו-TF. דגל אפשור הפסיקה IF (Interrupt Flag) קובע אם המעבד יענה לבקשת פסיקת חמרה (במקרה ש-IF=1) או יתעלם ממנה (במקרה ש-IF=0). הימנעות מקבלת הפסיקות נקראת **מיסוך פסיקות**. שימו לב, שבעת ביצועה של שגרת שרות הפסיקה עלולה להגיע עוד פסיקה, מצב בו שגרת השרות עצמה תופסק ותקרא שגרת שרות נוספת; תופעה זו עלולה לחזור שוב ושוב. לכן, כדי לטפל בפסיקה ללא הפרעה, דגלים אלו מורדים באופן אוטומטי ללא צורך בפקודה מפורשת ובסיום ביצוע הפסיקה, המעבד משחזר את ערכם המקורי של הדגלים הללו.

כעת המעבד צריך למצוא את שגרת הטיפול בפסיקה (ISR – Interrupt Service Routine). כדי לברר מהי הכתובת של שגרת הטיפול בפסיקה (ISR), ה-8086 פונה לאזור הכתובות הפיזיות 00000h – 003FFh. האזור הזה נקרא **טבלת וקטור הפסיקות**, ובו נמצאות הכתובות של שגרות הטיפול בפסיקות השונות. הכתובות האלה מסודרות בטבלה לפי מספר הסוג שלהן, כמו שמוצג בחלק א של איור 9.2 שלהלן: איבר 0 בטבלה מכיל את הכתובת של שגרת הטיפול בפסיקה מסוג 0, איבר 1 בטבלה מכיל את הכתובת של שגרת הטיפול בפסיקה מסוג 1, וכן הלאה עד האיבר האחרון בטבלה, המכיל את הכתובת של שגרת הטיפול בפסיקה מסוג 255.

לכל איבר בטבלה יש 4 בתים, המכילים את כתובת המקטע ואת הכתובת היחסית של שגרת הטיפול בפסיקה, כפי שמתואר בחלק ב של האיור. טבלת וקטור הפסיקות, שגודלה $1024 = 4 \times 256$ בתים, ממוקמת בחלק התחתון של מרחב הכתובות הפיזיות של ה-8086.

כאשר ה-8086 פונה לטבלת וקטור הפסיקות, הוא מחשב את הכתובת של האיבר הדרוש לו לפי מספר הסוג של הפסיקה. אם נסמן את סוג הפסיקה ב-N, יחשב ה-8086 את ערכו של $4 \times N$, והתוצאה תהיה הכתובת שבה מתחיל האיבר השייך לפסיקה. שימו לב, כדי להכפיל מספר בינארי ב-4, די להזיזו פעמיים שמאלה. העובדה הזאת מקלה על ה-8086 לאתר את האיבר בוקטור הפסיקה.

דוגמה 9.4 מתארת את אופן תהליך חישוב הכתובת של שגרת פסיקה .



איור 9.2
א. טבלת וקטור הפסיקות של ה-8086 ; ב. איבר בטבלת וקטור הפסיקות

דוגמה 9.3

נניח כי תכנית כלשהי נמצאת במקטע קוד שכתובתו 00BAh. בזמן ביצוע הוראת התכנית, הנמצאת בכתובות היחסיות 0622h - 0623h, מתקבלת בקשת פסיקה מסוג 5. הניחו כי:

– SP = 004Fh
– כל הדגלים מאופסים, פרט לדגלים CF ו-IF.

בטבלה שלפניכם מפורט תוכנו של וקטור הפסיקה מסוג 5.

טבלה 9.2
וקטור הפסיקה מסוג 5

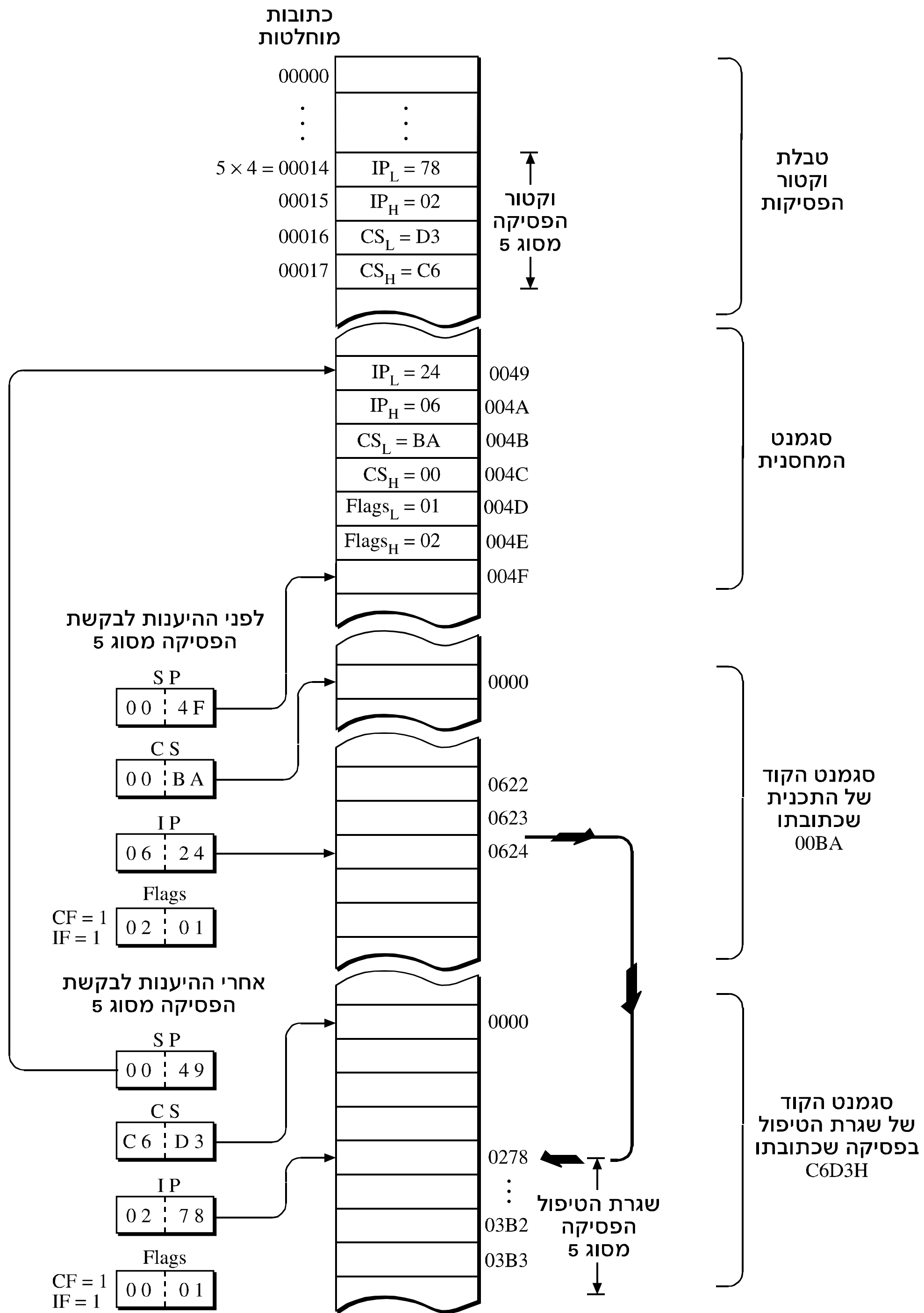
כתובת מוחלטת	תוכן
00014h	78h
00015h	02h
00016h	D3h
00017h	C6h

תארו מה צריך לעשות כדי להיענות לבקשת הפסיקה.

פתרון

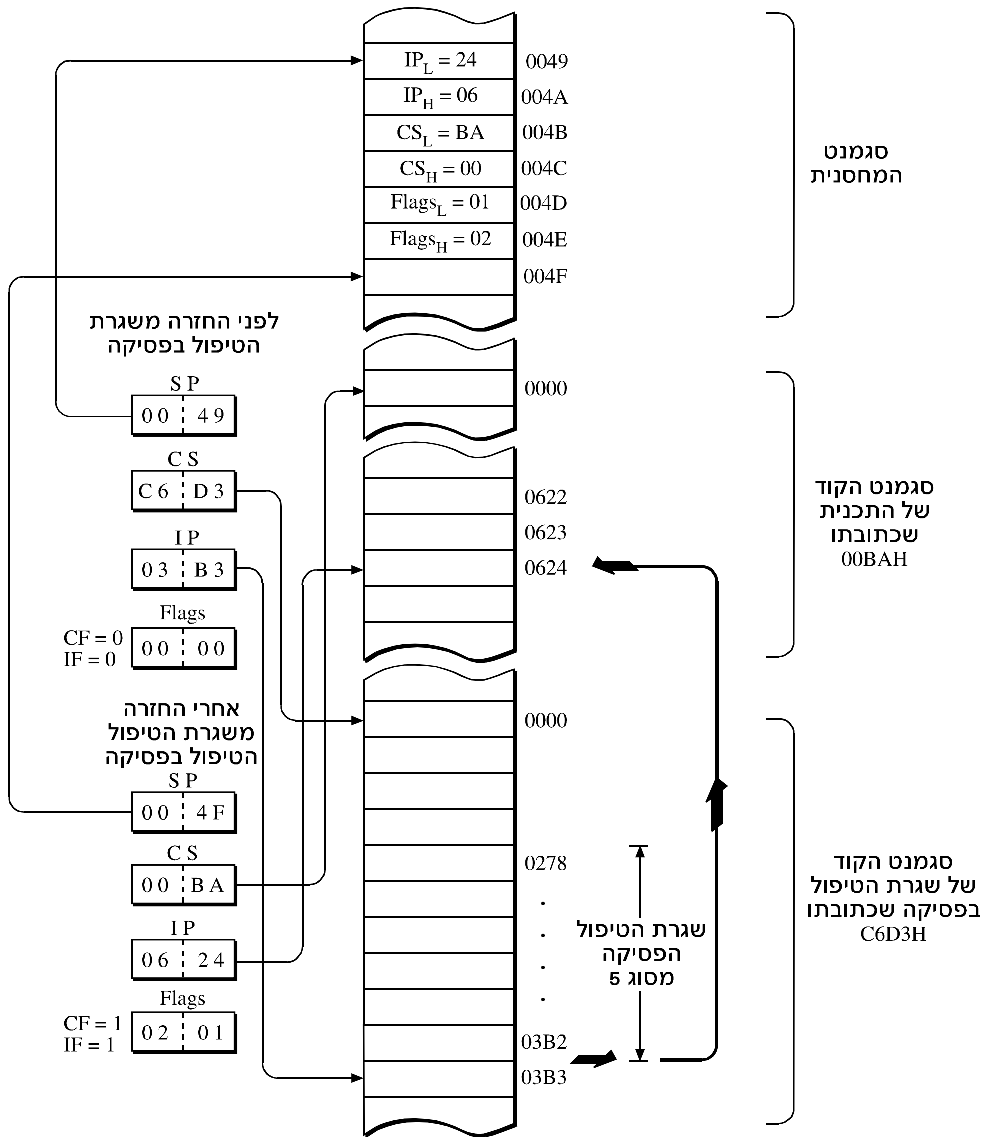
התבוננו באיור 9.3 המתאר מה צריך לעשות כדי להיענות לפסיקה.

לפני ההיענות לבקשת הפסיקה, ההוראה שצריכה להתבצע היא ההוראה שהכתובת שלה CS:IP = 00BAh:0624h, אוגר הדגלים מכיל 0201h, ו-SP = 004Fh. כאשר ה-8086 נענה לבקשת הפסיקה, הוא דוחף למחסנית את אוגר הדגלים, את האוגר CS ואת האוגר IP (כל אחד – מילה אחת, כלומר שני בתים). לאחר הדחיפה, ערכו של האוגר SP קטֵן ב-6 מערכו הקודם, ועתה הוא 0049h. בשלב הזה ה-8086 פונה לאיבר 5 בטבלת וקטור הפסיקות. האיבר הזה נמצא בין הכתובות המוחלטות 5×4 = 00014h ו-00017h = 5×4+3. ה-8086 טוען לתוך האוגר IP את תוכנם של שני הבתים הראשונים של הווקטור, ולתוך האוגר CS הוא טוען את שני הבתים האחרים. ההוראה הבאה שהוא יבצע היא ההוראה הראשונה של שגרת הטיפול בפסיקה, המתחילה בכתובת C6D3h:0278h.



איור 9.3
היענות לבקשת פסיקה מסוג 5

באיור 9.4 מוצג גם השינוי שחל בדגל IF, שהתאפס במהלך ההיענות לפסיקה. שימו לב, כל הכתובות המציינות כתובת בזיכרון באיור 9.3 (ובהמשך גם באיור 9.4) הן כתובות יחסיות.



איור 9.4
ביצוע חזרה משגרת טיפול בפסיקות

דוגמה 9.4

הניחו שהכתובת האחרונה של שגרת הטיפול בפסיקה (ראו דוגמה קודמת) היא C6D3h:03B2h, ושכל הדגלים של ה-8086 מאופסים. תארו את החזרה משגרת הטיפול בפסיקה לתכנית הראשית.

פתרון

ביצוע החזרה משגרת הטיפול בפסיקה מתואר באיור 9.4 שבעמוד הקודם.

לפני ביצוע החזרה: CS:IP = C6D3h:03B3h (הכתובת שלאחר הוראת החזרה מן השגרה), SP = 0049h, ואוגר הדגלים מאופס. בעת ביצוע החזרה, יוחלף התוכן של האוגר CS ושל האוגר IP בערכים שנשמרו במחסנית, והם יכילו את כתובת ההוראה הבאה בתכנית הראשית: CS:IP = 00BAh:0624h, בדומה לחזרה משגרה רחוקה. נוסף על כך, ישוחזר מצבו של אוגר הדגלים, פרט לאוגרים CF ו-IF. ערך האוגר SP יגדל ב-6 ויחזור לערכו המקורי, כמו שהיה לפני ההיענות לבקשת הפסיקה SP = 004Fh.

9.4 קריאה ושינוי של פסיקה

כדי לקרוא או לשנות תוכן של תא מטבלת הפסיקות, נשתמש בשגרות שירות של DOS

א. שגרות שירות DOS מספר 35h – לקריאת תוכן תא בטבלת הפסיקות

כדי לקרוא ערך של תא בטבלת הפסיקות נשתמש בשגרת שירות DOS מספר 35h.
שגרת שירות זו מקבלת באוגר AL מספר התא בטבלת הפסיקות ומחזירה באוגרים ES (אוגר מקטע) ו- BX את מען השגרה שמספרה הוכנס באוגר AL. האוגר ES יכיל את מען מקטע הקוד של השגרה והאוגר BX יכיל את ההיסט של השגרה המבוקשת. נסמן כתובת המוכלת בזוג אוגרים אלו בסימון ES:DX.

לדוגמה, נכתוב קטע תכנית המחזיר באוגרים ES:BX את מען שגרת הפסיקה מס. 5:

```
mov  ah,  35h
mov  al,  5
int    21h
```

ב. שגרות שירות DOS מספר 25h – לשינוי תוכן תא בטבלת הפסיקות

כדי לשנות ערך של תא בטבלת הפסיקות נשתמש בשגרת השירות 25h. שגרת שירות זו מקבלת ערכים באוגרים הבאים:

- AL מכיל את מספר התא בטבלת הפסיקות שאת ערכו ברצוננו לשנות
- DS מכיל את מען המקטע שברצוננו להכניס לתא בטבלת הפסיקות
- DX מכיל את ההיסט שברצוננו להכניס לטבלת הפסיקות

לדוגמה, נכתוב קטע קוד המכניס לתא מספר 5 בטבלת הפסיקות את מען השגרה myProc (בהנחה ששגרה זו מוגדרת בתכנית).

```
mov  ah,  25h                 ; קביעת סוג שגרת השירות
mov  al,   5                  ; קביעת מספר תא בטבלת הפסיקות
push seg myProc               ; דחיפת מען מקטע הקוד של השגרה למחסנית
pop   ds                      ; DS מכיל את כתובת מקטע של השגרה
mov  dx,  offset myProc       ; DS מכיל את ההיסט של השגרה
int     21h                   ; קריאה לשגרת השירות
```

ניתן להשתמש באופרטור offset לקבוע באוגר לא רק את ההיסט של משתנה אלא גם את ההיסט של מען של שגרה. בהוראה mov dx, offset myProc תוצאת האופרטור offset הוא ההיסט של מען השגרה myProc המושם באוגר dx.

באופן דומה תוצאת האופרטור seg בהוראהpush seg myProc היא מען מקטע הקוד בה נמצאת השגרה myProc.

ג. כתיבת שגרת פסיקה

שגרת פסיקה היא שגרה רחוקה FAR ובה אנו משתמשים בהוראת חזרה IRET. הוראת חזרה שולפת מהמחסנית לא רק את כתובת החזרה אלא גם את תוכן אוגר הדגלים.
מבנה של שגרת פסיקה הוא:

```
name  PROC  FAR

   iret
name  ENDP
```

דוגמה 9.5

לסיכום, נכתוב תכנית שמשנה את שגרת הפסיקה מס. 0 המוזעקת כאשר מתבצעת חלוקה באפס. אנו נשנה שגרה זו ונציג הודעה כי התבצעה חלוקה באפס. תחילה בדקו כיצד מתבצעת תכנית הכולל את שתי ההוראות הבאות:

```
mov cl, 0
div   cl
```

כפי שניתן לראות, ביצוע התכנית מופסק ומתקבלת הודעה: divide by zero.

שגרת פסיקה מס. 0 מוגדרת כך שבזמן שהיא מוזעקת, תחילה המעבד דוחף למחסנית את מען הוראת החילוק עצמה ולא את מען ההוראה הבאה. כך ניתן לדעת מהי הוראת החילוק שבוצעה וגרמה לשגיאה. כפי שראינו בפרק 6, תרגום הוראת חילוק לשפת מכונה יכול ליצר הוראות מכונה באורך שונה והדבר תלוי בשיטת המיעון ובסוג האופרנדים. אם למחסנית תוכנס כתובת של ההוראה העוקבת להוראת החילוק, קשה יהיה לחשב את כתובת הוראת החילוק שגרמה לשגיאה.

כדי לסיים את התכנית ולהפיק את ההודעה המתאימה, נכתוב שגרת פסיקה שתחילה מקדמת את כתובת ההוראה הבאה לביצוע ואחר מציגה הודעת שגיאה.

התכנית: מחליפה את כתובת תא של פסיקה מס. 0 בכתובת שגרה myProc שכתבנו, אך תחילה שומרת את כתובת שגרת הפסיקה המקורית. לאחר הזעקת שגרת הפסיקה מס. 0, נשחזר את טבלת הפסיקות ונחזיר את הכתובת של שגרת הפסיקה המקורית.

```
.MODEL SMALL
.STACK 100h

.DATA
outMessage db 0Dh, 0Ah, ' divide by zero ', 0Dh, 0Ah, '$'          ; הודעת שגיאה
.CODE
start:
   mov  ax,  @DATA
   mov  ds,  ax
                                        ; ES:BX שמירת כתובת שגרת פסקה מס 0 באוגרים
   mov  ah,  35h
   mov  al,  0
   int    21h
```

```
; myProc החלפת כתובת שגרת פסיקה מס 0 בכתובת השגרה
    mov  ah,  25h
    mov  al,   0
    mov  dx,  offset myProc
    push  seg myProc
    pop  ds
    int    21h
; myProc ביצוע חילוק ב-0  המזעיק את שגרת הפסיקה
    mov  cl,  0
    div    cl
; שחזור מען שגרת השגרה מס. 0 המקורית
    mov  ah,  25h
    mov  al,   0
    mov  dx,  bx
    push  es
    pop  es
    int    21h
exit :
    mov  ax,  4c00h
    int     21h
; שגרת פסיקה
myProc  PROC
    mov  bp, sp
    add   word ptr [ bp], 2
    lea    dx,  byte ptr  cs:outMessage          ; כתובת המחרוזת להצגה
    mov  ah ,  9
    int     21h
    iret
myProc ENDP
END start
```

9.5 הוראות IN ו-OUT

בדומה לזיכרון הראשי, גם התקני הקלט/פלט מחוברים אל המעבד באמצעות הפסים, כאשר לכל התקן מוקצה כתובת ייחודית. כפי שתיארנו בפרק 4, גודלו של מרחב המענים של הזיכרון הראשי במעבד 8086 הוא 1M וגודל מרחב המענים של הקלט/פלט במעבד 8086 הוא 64K. למען במרחב המענים של הקלט/פלט קוראים בדרך כלל **כניסה** (port). מרחב המענים של הקלט/פלט אינו חופף למרחב המענים של הזיכרון הראשי, וכך לדוגמה המען 100h בזיכרון הראשי הוא מקום פיזי שונה מהמען 100h במרחב המענים של הקלט פלט. במחזור אפיק, הגישה לשני מרחבי המענים מתבצעת באמצעות קווי בקרה. קו הבקרה $M/\overline{IO}$ משמש כדי להבדיל בין פניה לזיכרון הראשי לפניה להתקן קלט/פלט. כאשר רמת המתח גבוהה בקו זה, יש פניה למרחב המענים של הזיכרון הראשי וכאשר רמת המתח נמוכה בקו זה יש פניה למרחב המענים של הקלט/פלט. בנוסף לכך, כאשר יש פניה לזיכרון ראשי משתמשים בקו בקרה $Read/\overline{Write}$ כדי להבדיל בין קריאה מהזיכרון (רמת מתח גבוהה) או לכתיבה בזיכרון (רמת מתח נמוכה).

ההפרדה בין מרחבי המענים באה לידי ביטוי גם בהוראות אסמבלי שונות עבור כל מרחב מענים. למענים במרחב הזיכרון ניגשים באמצעות פקודות כמו mov ax, var או add var, 18. הגישה למענים של הקלט פלט מתבצעת באמצעות הפקודות IN ו-OUT בלבד.

א. הוראת הקלט IN

כדי להעביר מידע מהתקן הקלט למעבד נשתמש בהוראה:

IN אופרנד יעד , אופרנד מקור

קוד הפעולה של ההוראה הוא IN (קיצור של המילה INput), והוא מציין העברת מידע מהתקן הקלט למיקרו-מעבד. יש לציין בהוראה שני אופרנדים:

- אופרנד מקור מכיל את כתובתו של כניסת התקן הקלט.
- אופרנד יעד הוא אוגר AL או AX שמאחסן את הנתון שנקלט מהתקן הקלט. בסעיף הזה נאחסן את הנתון שנקלט מן ההתקן הקלט באוגר AL.

ב. הוראת הפלט OUT

כדי להעביר מידע מהמעבד אל התקן הקלט נשתמש בהוראה:

OUT אופרנד יעד , אופרנד מקור

קוד הפעולה של ההוראה הוא 'OUT' (קיצור של המילה OUTput) והוא מציין העברת מידע מן המיקרו-מעבד אל התקן הפלט. משמעות ההוראה: העבר את המידע שמאוחסן באוגר למִפתח הפלט שכתובתו נתונה בהוראה. כלומר, יש לציין בהוראה שני אופרנדים:

- אופרנד היעד הוא כתובת כניסת התקן הפלט
- אופרנד המקור הוא האוגר AL או AX שמכיל את המידע שיש להציג בהתקן הפלט.

בהוראות אלו ניתן להשתמש בשתי שיטות מיעון:

א. במיעון ישיר כאשר כתובת ההתקן היא חלק מההוראה ובמקרה כזה היא מוגבלת לתחום שבין 0 ל-255 יכולה (בית).

ב. מיעון אוגר עקיף אם כתובתו של כניסת התקן הקלט או הפלט גדולה מ-FFh, יש לאחסן אותה באוגר DX ולציין אותו כאופרנד שמכיל את כתובת הקלט. לדוגמה: כדי לקלוט ערך כלשהו מהתקן הקלט שמחובר לכתובת 0379h, נאחסן תחילה את הכתובת באוגר DX:

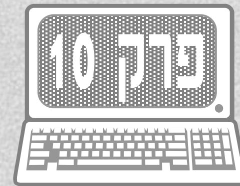
```
MOV  DX,  379h
IN  AL,  DX
```

משמעות ההוראה השנייה: קרא ממפתח הקלט שכתובתו רשומה באוגר DX את הנתון שבהתקן הקלט, ושמור אותו באוגר AL.

דוגמה 9.6

נכתוב קטע תכנית הקולטת נתון מטיפוס בית מהתקן קלט המחובר לכתובת 0378h. התכנית מציגה בהתקן פלט המחובר לכתובת 037h את ההיפוך הלוגי של נתון הקלט.

```
mov dx, 0378h
in al, dx
neg al
out  037h, al
```



ארכיטקטורה של מעבדים מתקדמים

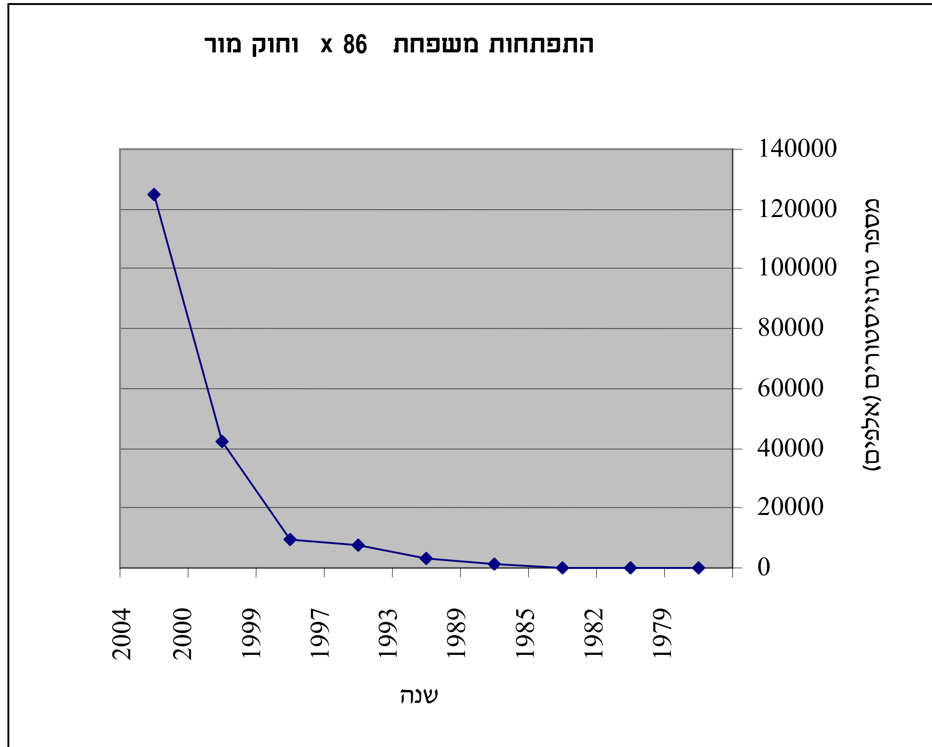
10.1 ההשפעה של ההתפתחות הטכנולוגית על מבנה מעבדים מתקדמים

ההתפתחות בתחום המחשוב בשנים האחרונות היא דרמטית, ומובילה כיום להפצת השימוש במחשבים בכל מקום: בתעשייה, בארגונים עסקיים וציבוריים, בבתי ספר, בתי חולים ובבתים פרטיים. גם היקף היישומים שפותחו ונמצאים בשימוש, הוא גדול מאוד – הם מכילים מאות אלפי שורות קוד ומורכבים מעשרות תכניות, והם מעבדים כמויות עצומות של נתונים מסוגים שונים, החל מממספרים וטקסט וכלה בתמונות, קול, סרטי וידאו וכדומה.

בפרק זה נתאר בצורה מופשטת את ההשפעה של ההתפתחות הטכנולוגית על מבנה המעבדים ועל ביצועיהם, את השינויים בארכיטקטורה של מעבדים מתקדמים, התורמים לקיצור זמן הביצוע של מחזורי הבאה-וביצוע בתכנית. כמו-כן נתאר את השיטות לארגון הזיכרון במעבדים מתקדמים, המאפשרים שמירה ועיבוד של כמויות גדולות של נתונים. לסיום נסביר כיצד נשמר עקרון ה-"תאימות אחורה" ונציג את אופני העבודה שקיימים במעבדי אינטל מתקדמים, המאפשרים להפעיל יישומים שנכתבו עבור מעבדים ישנים יותר.

בשנת 1965 העלה אחד ממייסדי חברת אינטל (גורדון מור) השערה הידועה כיום בשם **חוק מור** ("Moore's Law"). ההשערה, או התחזית שלו, הייתה כי מספר הטרנזיסטורים לאינץ' מרובע, המשובצים במעגלים מוכללים, יכפיל את עצמו בכל שנה. הערכה זו עודכנה בשנים הבאות, וכיום כמות הטרנזיסטורים לאינץ' מרובע מוכפלת בכל 18 חודש.

איור 10.1 מראה את התפתחות המעגלים המוכללים, לפי הדורות של מעבדי חברת אינטל, ואת ההתאמה של התפתחות זו לחוק מור.



איור 10.1
התפתחות מעבדי אינטל וחוק מור (נלקח מאתר של אינטל)

קצב ההתפתחות הטכנולוגית (שנקבע בחוק מור) השפיע בצורה דרמטית על התפתחות המחשבים, ואחת לשנה-שנתיים אנו מתבשרים על הדור הבא של מעבדים. המזעור של הרכיבים והמעגלים האלקטרוניים, המרכיבים את המחשב בכלל ואת המעבד בפרט, הקטין את הגודל הפיזי של המעבדים. בנוסף הגדיל המזעור את מורכבותם של המעגלים האלקטרוניים, ואפשר בכך בניית מעבד משוכלל יותר וגם שיפור של מהירות העיבוד. המזעור תרם גם להגדלת קיבולת הזיכרון וכך מאפשר עיבוד של תוכנות עתירות נתונים.

מגמה זו השפיעה גם על ההתפתחות של מעבדי משפחת 86× של חברת אינטל. התבוננו בטבלה 10.1. בעמודה **גודל רכיב** בטבלה רשום הגודל הפיזי של פיסת הסיליקון עליה מורכב המעבד; הגודל הזה נתון ביחידת מידה הנקראת מיקרון (1 מיקרון = אלפית המילימטר). אפשר לראות בטבלה כי גודל הרכיב הולך וקטן: בדור הראשון גודלו היה 0.003mm והוא הכיל "רק" 29,000 טרנזיסטורים; כיום גודלו 0.0018mm והוא מכיל 25,000,000

טרנזיסטורים. יחד עם זאת, מהירות השעון של המעבד גדלה פי 200 ויותר (מקצב של 4.77MHz במעבד מהדור הראשון לקצב של 1000MHz במעבד בדור השמיני).

יחידת המדידה 1MHz (1 מגה הרץ) מציינת מיליון תנודות בשנייה. כל תנודה מספקת אות חשמלי שמאפשר למעבד לבצע פעולה הנקראת "מחזור שעון". ככל שקצב השעון גדול יותר, המעבד מהיר יותר.

כדי לקבל הערכה גסה של השיפור במהירות פעולתם של המחשבים, נחשב לדוגמה את מספר הוראות ADD שמעבד יכול לבצע בשנייה. ביצוע ההוראה ADD נמשך במעבד 8086 בין 3 ל-17 מחזורי שעון (הדבר תלוי בסוג האופרנדים ובשיטת המיעון). מעבד שפועל בקצב של 1MHz יכול לבצע בין 58,000 ל-330,000 הוראות כאלה בשנייה, ומעבד שפועל בקצב של 1000MHz יכול לבצע בין 58,000,000 ל-330,000,000 הוראות ADD בשנייה.

כמו כן ניתן ללמוד מהטבלה כי רוחב האוגרים, רוחב הפסים וגודל הזיכרון גדלו גם הם. ההרחבה של פס הנתונים במעבדים מגדילה את קצב העברת הנתונים בין המעבד לזיכרון ובין המעבד ליחידות הקלט/פלט, וכתוצאה ניתן להעביר כמות גדולה יותר של נתונים בכל פנייה לזיכרון. לדוגמה, אם נרצה להעביר מילה מרובעת (מטיפוס DQ), במעבד שרוחב פס הנתונים שלו הוא 16 סיביות, נצטרך לפנות לזיכרון ארבע פעמים, ואילו במעבד P5 (פנטיום) שבו רוחב פס הנתונים הוא 32 סיביות, יידרשו רק שתי פניות לזיכרון.

הרחבת פס הכתובות משפיעה על גודל הזיכרון משום שהיא מאפשרת מרחב כתובות פיזיות גדול יותר. לדוגמה, בפרק 4 ראינו כי במעבד 8086 רוחב פס הכתובות הוא בן 20 סיביות והוא מאפשר מרחב זיכרון של 1Mbyte. החל ממעבד פנטיום P5 רוחב פס הכתובות הוא בן 64 סיביות והוא מאפשר מרחב זיכרון של 64GB $=2^{64}$ (ראו טבלה 10.1). מרחב זיכרון ראשי גדול יותר, מאפשר עיבוד של יישומים גדולים יותר, המשתמשים בכמות גדולה של נתונים, כמו עיבודי תמונה וקול.

אולם לא רק הטכנולוגיה השתפרה, דגש ניכר הושם על שינוי הארכיטקטורה של המעבדים כדי לשפר את כושר העיבוד של המחשב. בסעיף הבא נדון בשיפור שחל במודל התכנותי, ובמחזור ההבאה-וביצוע של הוראות באמצעות ארכיטקטורה של "צינור הוראות" (pipelining).

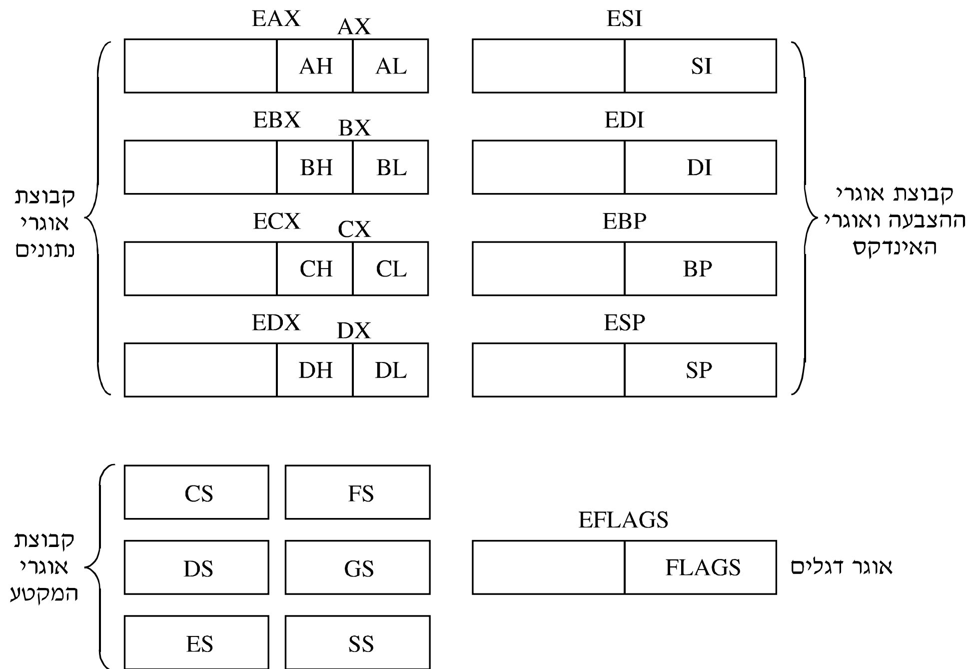
טבלה 10.1
התפתחות מעבדים ממשפחת 80x86

שם מעבד	תאריך ייצור	גודל רכיב (מיקרון)	מספר טרנזיסטורים (אלפים)	רוחב פס נתונים	מהירות שעון MHz
8088	1979	3	29,000	16	5
80286	1982	1.5	134,000	16	6
80386	1985	1.5	275,000	32	16
80486	1989	1	1,200,000	32	25
Pentium	1993	0.8	3,100,000	32	60
Pentium II	1997	0.35	7,500,000	32	233
Pentium III	1999	0.25	9,500,000	32	450
Pentium 4	2000	0.18	42,000,000	32	1500
Pentium 4 "Prescott"	2004	0.09	125,000,000	32	3600

10.2 מבנה האוגרים במעבדים מתקדמים

כפי שראינו בפרק הקודם, אחת הדרכים לשפר את מהירות הביצוע של תכניות היא להשתמש בשיטת מיעון אוגר. אולם למרות ההתפתחות הטכנולוגית בתחום המעגלים המוכללים, השינוי העיקרי במבנה האוגרים הוא בהרחבתם ולא במספרם. רוחב האוגרים במעבדים מתקדמים גדל מ-16 סיביות ל-32 סיביות, וכיום מוצגים מעבדים שרוחב האוגרים שלהם הוא 64 סיביות. מנקודת מבט של המתכנת, נוספו שני אוגרים חדשים לקבוצת אוגרי המקטע של מעבד פנטיום: אוגר מקטע F ואוגר מקטע G, וכך ניתן להגדיר שישה מקטעים (סגמנטים) בזיכרון של מעבד פנטיום. הגדלת מספר המקטעים מאפשרת הרצה במקביל של כמה תכניות ועיבוד כמות גדולה יותר של נתונים.

כדי לשמור על עקרון התאימות, לשמות האוגרים במעבדים החדשים נוספה בהתחלה האות E (Extended). קבוצת האוגרים לשימוש כללי בפנטיום מוצגת באיור 10.2; היא כוללת את אוגרי הנתונים, אוגרי ההצבה ואוגרי האינדקס (חוץ מהאוגר EIP – מצביע הוראות). מעבד פנטיום מוגדר כמעבד של 32 סיביות, בגלל גודל האוגרים לשימוש כללי וגם גודל מילת הזיכרון, שהוא 32 סיביות. תכנית המורצת במעבד פנטיום יכולה לפנות לאוגרים בני 32 סיביות, אך גם לאוגרים של 8 או 16 סיביות.



איור 10.2
האוגרים לשימוש כללי בפנטיום

לדוגמה נציג קטע קוד שמתאים למעבד פנטיום:

```
      neg   eax
      je    L3
L1:   neg   eax
      xchg  eax,  edx
L2:   sub   eax,  edx
      jg    L2
      jne   L1
L3:   add   eax,  edx
```

אפשר לראות כי משתמשים בקוד זה בהוראות דומות, אך האופרנדים הם בני 32 סיביות. מובן שאוצר הפקודות של המעבדים הורחב. נוספו הוראות המטפלות באוגרים החדשים, למשל, הוראות לטעינת כתובת אוגרי המקטע, כמו LFS, LGS או הוראות המטפלות בהרחבת האוגרים, כמו CDQ המרחיבה מילה כפולה (בת 32 סיביות) למילה מרובעת (בת 64 סיביות). ההוראה CDQ (Convert Doubleword to Quadword), קיימת החל

ממעבד 80386 והיא, בדומה להוראה CBW שהצגנו בפרק 5, מרחיבה את הסימן של האוגר EAX ושמה אותו באוגר EDX. לדוגמה

```
mov   eax, 0FFFFFFFBh        ; eax ← –5 (FFFFFFFBh)
cdq                          ; edx ← FFFFFFFFh, eax ← FFFFFFFBh
```

כמו כן נוספו הוראות חדשות לחלוטין, כמו CMPXCHG המשווה ומחליפה אופרנדים או INS ו-OUTS המשמשות לקלט-פלט של מחרוזות. לסיום נציין כי להרצה של תכניות אלה, אנו זקוקים לאסמבלר עבור מעבד 32 סיביות.

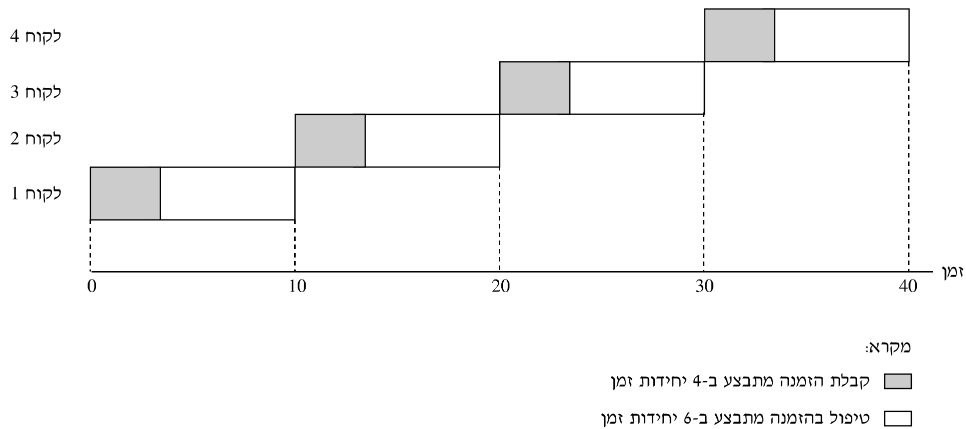
אוגרים נוספים שקיימים במעבדי אינטל הם אוגרים ייעודיים שהמעבד משתמש בהם לניהול הזיכרון ולקביעת אופן העבודה ואוגרים המסייעים בתהליך ניפוי שגיאות (debuging).

10.3 ארכיטקטורת "צינור הוראות" (pipelining)

הארכיטקטורה של רוב המעבדים עדיין מבוססת כיום על הארכיטקטורה של פון נוימן, כלומר, על העיקרון של אחסון התכנית והנתונים באותו הזיכרון. בפרק הראשון תיארנו כיצד מתבצעת תכנית שבה לכל הוראה מתבצע מחזור הבאה-וביצוע, המתחיל אחרי שהסתיים ביצוע המחזור של ההוראה הקודמת. אחת השיטות להגביר את קצב העיבוד (בנוסף לשיפורים הטכנולוגיים כגון השעון שהצגנו בסעיף הקודם) היא לבצע כמה הוראות בו-זמנית. ארכיטקטורת מעבד המממשת שיטה זו נקראת **ארכיטקטורת צינור הוראות** (pipelining).

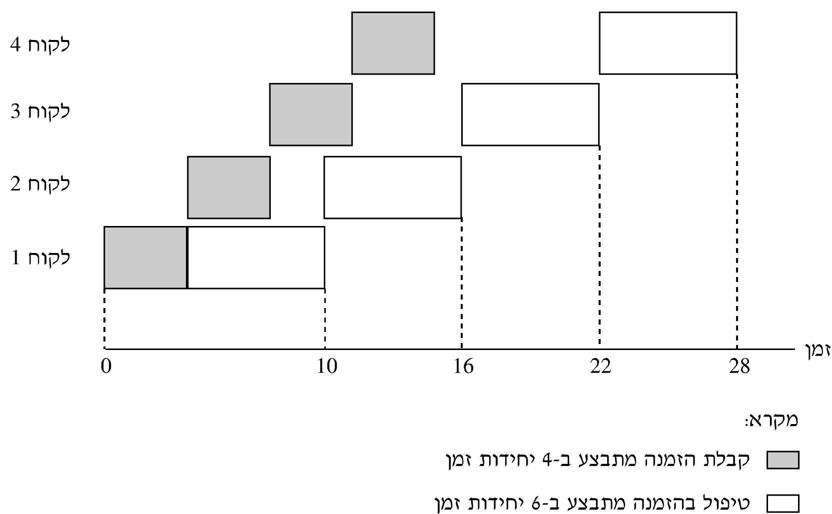
כדי להסביר את העיקרון של שימוש בצינור הוראות, נניח לדוגמה מזנון מהיר שבו מועסק עובד שתפקידו לקבל הזמנה מלקוח ולהכין עבורו את המזון. כיוון שרק עובד אחד מועסק בתפקיד זה, הוא יכול לטפל בכל פעם רק בלקוח אחד. איור 10.3 מתאר כיצד העובד מטפל בארבעה לקוחות, בזה אחר זה, כלומר בצורה סדרתית.

אם נניח כי משך הזמן הדרוש לקבלת ההזמנה הוא 4 יחידות וביצוע ההזמנה נמשך 6 יחידות, אזי זמן הטיפול בלקוח אחד הוא 10 יחידות, ומשך הטיפול בארבעה לקוחות יהיה $4\cdot(4 + 6) = 40$ יחידות זמן.



איור 10.3
עובד אחד מטפל בארבעה לקוחות

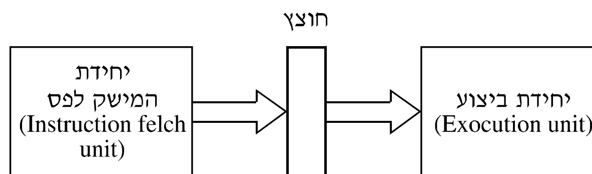
כדי לקצר את הזמן שנמשך הטיפול בלקוחות, החליטו בחברה להעסיק שני עובדים: עובד א יקבל את ההזמנה ועובד ב יבצע אותה. כיוון שלכל עובד הוקצה תפקיד מסוים, הם יכולים לעבוד במקביל ובזמן שהאחד מקבל את ההזמנה של לקוח מסוים, העובד השני יוכל להכין את ההזמנה של הלקוח שקדם לו בתור. באיור 10.4 אפשר לראות את השיפור המושג בזמן הטיפול בארבעה לקוחות. השיפור – צמצום הזמן – מושג הודות לחפיפה בין הפעולות שמבצעים העובדים המטפלים בלקוחות.



איור 10.4
שני עובדים מטפלים בארבע לקוחות

באיור 10.4 אנו רואים כי זמן הטיפול בלקוח אחד לא השתנה, אולם זמן ההמתנה של הלקוחות (חוץ מהראשון בתור) מתקצר, ולכן הזמן הדרוש לטפל בכל הלקוחות מתקצר אף הוא. זמן הטיפול בכל ארבעת הלקוחות הוא כעת רק 28 יחידות זמן. כמו-כן אפשר לראות כי הפעולה הארוכה ביותר היא הגורם העיקרי שמשפיע על משך הביצוע הכללי של הטיפול בכל הלקוחות. כיוון שזמן קבלת ההזמנה קצר מהזמן הדרוש לביצועה, נאלץ העובד הראשון, שמקבל את ההזמנות, להמתין עם ההזמנה שקיבל מן הלקוח החדש, עד שהעובד השני, שמבצע את ההזמנה יסיים את הטיפול בלקוח הקודם. כדי למנוע את התלות בין הפעולות של שני העובדים, נניח כי העובד הראשון אינו צריך להמתין לעובד השני (שמכין את ההזמנה), הוא יכול לטפל בקבלת ההזמנות ברצף, ואת ההזמנות שקיבל הוא תולה על לוח הודעות. העובד השני ניגש ללוח ההודעות כדי להוריד ממנו, בכל פעם, הזמנה אחת ולטפל בה.

באופן דומה, המבנה של מעבד 8086 מממש את העיקרון של צינור הוראות, ובהתאם הוא מחולק לשתי יחידות עיקריות: יחידת מישק לפס (BIU – Bus Interface Unit) המטפלת בשלב ההבאה ויחידת ביצוע (EU – Execution Unit) המטפלת בשלב הביצוע, כלומר, מבצעת את ההוראה.



איור 10.5
מבנה סכמתי של מרכיבי מעבד 8086

במעבד 8086, הקשר בין שתי היחידות מתבצע באמצעות יחידה הנקראת **תור הוראות**. זוהי יחידת זיכרון שיכולה להכיל עד שש הוראות; כאשר יחידת המישק לפס מביאה הוראות מן הזיכרון, היא מכניסה אותן לתור ההוראות, ממנו שולפת יחידת הביצוע את ההוראות שעליה לבצע; ההוראה שהוכנסה ראשונה, על-ידי יחידת המישק לפס, תישלף ותבוצע ראשונה על-ידי יחידת הביצוע.

תור ההוראות משמש כחוצץ בין יחידת המישק לפס לבין יחידת הביצוע, בכך הוא מאפשר ליחידת הביצוע לפעול בלי להיות תלויה ביחידת המישק לפס. מיד לאחר שיחידת הביצוע סיימה לבצע הוראה, היא שולפת מהתור את ההוראה הבאה ומתחילה בביצועה. המתכננים

הגיעו למסקנה שתור הוראות גדול יותר, יוכל לשפר את זמן הביצוע של התכנית, ואכן במעבד 386 תור ההוראות הוא בעומק (בגודל) של 16 הוראות. אולם, כפי שנראה בהמשך, יש מגבלה על גודלו של תור ההוראות, ושימוש בתור הוראות גדול מאוד, במצבים מסוימים, יכול להאריך את זמן הביצוע של התכנית ולא לקצר אותו.

יחידת המישק לפס מתוכננת להביא הוראה חדשה – כל עוד לא הסתיימה התכנית ויש מקום בתור. היתרון בשיטה זו הוא שיחידת הביצוע יכולה לבצע הוראות באופן רציף, בלי להמתין להבאת ההוראה. יש שלושה מקרים בהם יחידת הביצוע תמתין להבאת הוראה חדשה:

מקרה ראשון: כאשר הוראה צריכה לגשת למקום בזיכרון שאינו בתור ההוראות.
במקרה כזה יחידת המישק לפס משהה את תהליך ההבאה השוטף, ופועלת להבאת ההוראה הדרושה; לאחר מכן היא תחזור לתהליך הבאת ההוראות (עליהן מצביע באופן סדרתי האוגר IP). לאחר שההוראה הדרושה הובאה, יחידת הביצוע מחדשת את פעולתה.

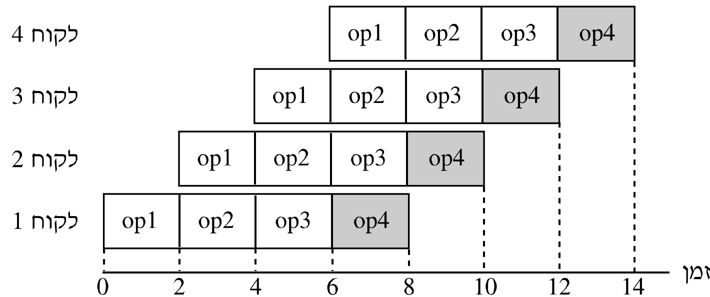
מקרה שני: כאשר מתבצעת הוראה מקבוצת ההוראות להעברת בקרה (ואז ההוראה הבאה לביצוע אינה ההוראה הבאה בזיכרון אחרי ההוראה הנוכחית).
במקרה כזה יחידת הביצוע תמתין עד הבאת ההוראה הדרושה.

מקרה שלישי: כאשר תור ההוראות מלא.
מצב זה מתרחש כאשר מתבצעת הוראה שזמן ביצועה הוא ארוך (לדוגמה MUL) ואז יחידת המישק לפס ממתינה שיחידת הביצוע תשלוף הוראה מהתור, כדי שהיא תוכל להמשיך בתהליך הבאת ההוראות.

נחזור לדוגמה של המזנון המהיר: ראינו כי באמצעות שני עובדים השגנו שיפור משמעותי בזמן הדרוש לביצוע תכנית; כעת עולה השאלה: נניח שנחליט להוסיף עובדים, האם נשיג שיפור נוסף? נבדוק לדוגמה מצב שבו ארבעה עובדים במזנון המהיר מטפלים בארבעה לקוחות. נחלק בין ארבעתם את הפעולות השונות הכרוכות בטיפול בלקוח:

- עובד א מקבל הזמנה; נסמן פעולה זו כ-op1
- עובד ב מטפל בטיגון המבורגר וצ׳יפס; נסמן פעולה זו כ-op2
- עובד ג מכין את הקינוחים והתוספות; נסמן פעולה זו כ-op3
- עובד ד אורז את ההזמנה; נסמן פעולה זו כ-op4

כדי לפשט את החישוב, נניח כי זמן הביצוע של כל פעולה הוא זהה ושווה ל-2 יחידות זמן. איור 10.6 מתאר את מהלך הטיפול של ארבעה עובדים בארבעה לקוחות, ואת משך זמן הטיפול.



איור 10.6
ארבעה עובדים מטפלים בארבעה לקוחות

התבוננות באיור מלמדת כי זמן הטיפול בארבעת הלקוחות התקצר, והוא נמשך כעת רק 14 יחידות זמן – מחצית מזמן הטיפול שנדרש לשני עובדים שטיפלו בארבעה לקוחות. המסקנה היא שהגדלת מספר העובדים, שכל אחד מהם מבצע פעולה קצרה יחסית, משפרת את זמן הטיפול בכל הלקוחות. ואכן, הגדלה נוספת של מספר העובדים, שכל אחד מהם יבצע פעולה קצרה, יקצר עוד יותר את זמן הטיפול בלקוחות, אבל צריך לזכור כי יש לכך מחיר: ככל שמספר העובדים גדל, גם ההוצאות גדלות: יש לשלם משכורת לכל עובד, כמו-כן יש להשקיע כסף בארגון המטבח – אם לא יהיה לכל עובד מרחב משלו, העובדים יפריעו אחד לשני בעבודתם, התקשורת בין העובדים דורשת זמן ומשאבים אחרים..

נחזור למחשבים ונראה כי גם כאן מספר הפעולות המבוצעות בתהליך הבאה-וביצוע גדל עם התקדמות המחשבים, ובהתאם זמן ביצוע התכניות התקצר. לדוגמה, במעבד 80486, צינור ההוראות מכיל 5 פעולות ובמעבד פנטיום 4 הוא מכיל כבר 20 פעולות. אולם הגדלת מספר הפעולות מגדילה את מספר היחידות מהן מורכב המעבד ובהתאם מסבכת את המבנה שלו ומייקרת את החומרה. בנוסף לכך, השימוש בחוצץ בין יחידה ליחידה, שבאמצעותו מעבירים את הנתונים מיחידה ליחידה, מוסיף גם הוא לזמן הביצוע של ההוראה (התקורה גדלה). בנוסף לבעיות בחומרה יש לדאוג שלא תהיה התנגשות בין משאבים הדרושים לביצוע הפעולות השונות.

כדי להמחיש לכם כיצד מעובדות תכניות במעבדים המשתמשים בארכיטקטורת "צינור הוראות" נתאר מחזור הבאה-וביצוע המכיל ארבעה שלבים :

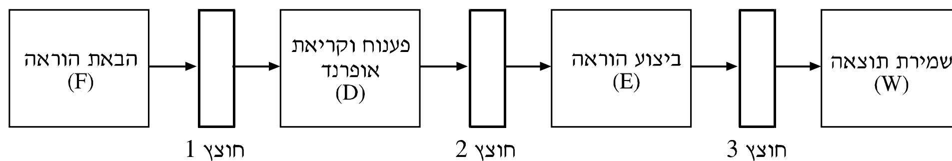
F (FETCH) – שלב הוצאת ההוראה מהזיכרון ;

D (DECODE) – שלב פענוח ההוראה והבאת אופרנדים ;

E (EXECUTE) – שלב ביצוע הפעולה ;

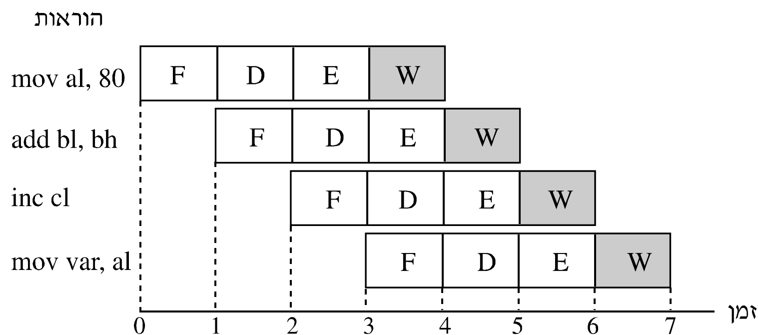
W (WRITE) – שלב כתיבת התוצאה.

בנוסף לכך, כדי לפשט את התיאור נניח כי זמן הביצוע של כל שלב ושלב, שווה ליחידת זמן אחת.



איור 10.7
מבנה מעבד עם צינור הוראות בן 4 פעולות

לדוגמה, נתאר את העיבוד של קטע התכנית הזה :



איור 10.8
תהליך ביצוע הוראות בצינור הוראות בן 4 שלבים

באיור 10.8 אפשר לראות כי ביצוע ארבע ההוראות יארך 7 יחידות זמן.

השימוש בטכניקה של צינור הוראות יעיל כל עוד כל יחידה מבצעת את הפעולה המוטלת עליה ברציפות, ללא הפרעות. אולם במציאות המצב הוא לא תמיד כך ; לדוגמה, במזנון

המהיר ייתכן מצב כזה: מלאי הטוגנים אזל והעובד השני צריך להמתין לחידוש המלאי; בעקבות עיכוב זה ימתינו גם שאר העובדים. הפרעות יכולות להיות גם במחשב, בביצוע השוטף של ההוראות. מצבים אלה נקראים מצבי "**סיכון**" (hazard). הם מסמנים מצב שבו הוראה מסוימת אינה מתבצעת בזמן שהיא הייתה אמורה להתבצע, ובעקבות כך מואט קצב הביצוע של התכנית כולה. קיימים שלושה סוגים של מצבים בהם מופרע ביצוע זרם ההוראות שנמצאות בצינור ההוראות:

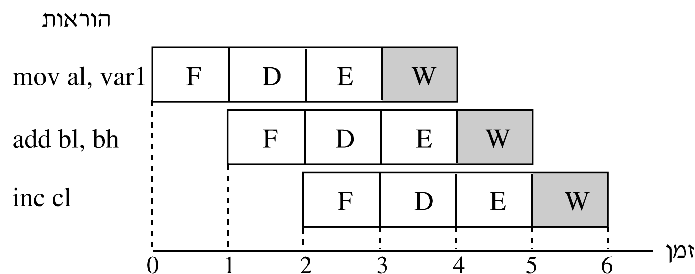
- Structural Hazards – מתרחש כאשר יש התנגשות בין משאבי החומרה, למשל: שתי פעולות, בשתי הוראות שונות, צריכות לקרוא מהזיכרון באותו זמן ושתיהן משתמשות באותם הפסים ובמקרה כזה המידע משתבש.
- Data Hazards – מתרחש כאשר ביצוע הוראה מסוימת תלוי בתוצאה של הוראה אחרת, שקדמה לה, אך ביצועה לא הסתיים עדיין.
- Control Hazards – מתרחש כאשר יש הוראות להעברת בקרה, אך ההוראה אליה יש לפנות אינה נמצאת עדיין בתור.

טיפול במצבי סיכון (hazard)

כדי לפתור בעיה הנוצרת ממצב "סיכון" המעבד צריך לזהות תחילה את המצב ואחר כך לטפל בו. בסעיף זה נתאר כמה שיטות לטיפול במצבים אלה.

א. טיפול ב-Structural Hazards

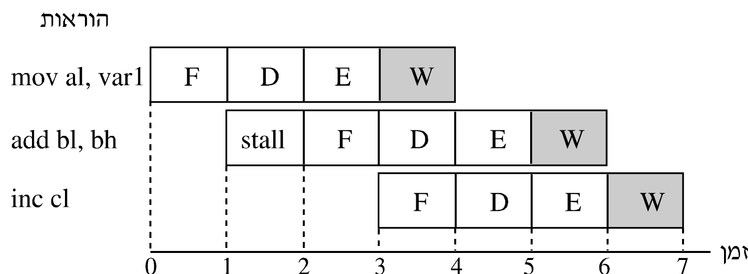
הפתרון הפשוט ביותר למצבים אלה הוא להשהות (to stall) את הפעולה הבאה. לדוגמה, נתאר מצב של Structural Hazards בקטע תכנית המכיל שלוש הוראות:



איור 10.9 א
מצב של structural hazard

בשלב D (בדוגמה המוצגת באיור 10.9) בהוראה הראשונה (MOV AL, var1), המעבד צריך לפנות לזיכרון כדי לקרוא אופרנד השמור בזיכרון, ובו-בזמן מתבצע שלב F של ההוראה השנייה, שבה המעבד צריך לפנות לזיכרון כדי לקרוא את ההוראה. במקרה כזה מתרחשת התנגשות כי למרות ששני השלבים פונים לתאי זיכרון שונים, הם משתמשים באותם פסי נתונים ובאותם פסי כתובות, וכך משתבש המידע שמועבר בהם.

אחת השיטות לפתור את הבעיה היא,כאמור, להשהות ביחידת זמן אחת את ביצוע פעולת ההבאה של ההוראה השנייה. בחלק ב של איור 10.9 מובא ביצוע תכנית הכולל השהיה באורך יחידת זמן אחת, כתוצאה מכך הביצוע של קטע תכנית זה נמשך 8 יחידות זמן במקום 7.



איור 10.9 ב
שימוש בהשהיה כדי למנוע מצב structural hazard

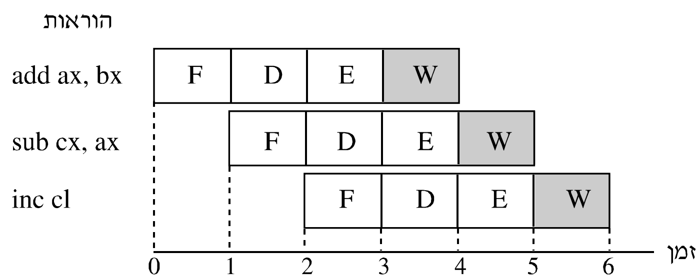
שיטה נוספת לפתור בעיה זו היא להכפיל את מספר המשאבים. לדוגמה: כדי למנוע את ההתנגשות הנובעת מביצוע שתי פעולות אריתמטיות, מעבד פנטיום כולל שתי יחידות ביצוע (ALU), וכדי למנוע התנגשות במהלך הגישה לזיכרון, מעבד פנטיום מכיל שתי יחידות זיכרון הנקראות **זיכרון מטמון לנתונים** ו **זיכרון מטמון להוראות**. בסעיף הבא נעסוק בארגון הזיכרון במעבדים מתקדמים ונדון בו בתפקידו של זיכרון המטמון .

ב. טיפול ב-Data Hazard

כפי שכבר אמרנו, ביצוע התכנית מואט כאשר הביצוע של הוראה אחת תלוי בתוצאת החישוב של הוראה אחרת שעדיין לא התבצעה; מצב זה נקרא Data Hazards. נניח לדוגמה שתכנית כוללת ההוראות האלה:

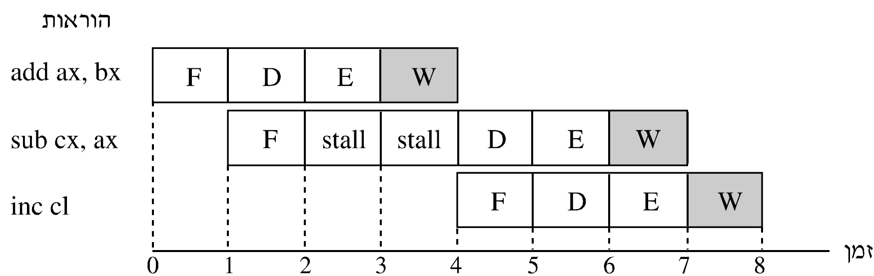
```
add  ax,  bx
sub  cx,  ax
inc  dx
```

פעולת החיסור תלויה בערכו של AX, לכן היא צריכה להתבצע רק אחרי שהסתיים הביצוע של הוראת החיבור שקדמה לה, והתוצאה נכתבה באוגר AX (אוגר היעד). אבל כפי שאפשר לראות באיור 10.10, כאשר שתי הוראות אלה רשומות בזו אחר זו בתור ההוראות, ייתכן שייווצר מצב שבו שלב D של ההוראה SUB יתבצע לפני ששלב W של הוראה ADD התבצע; במקרה כזה הערך שייקרא יהיה הערך הישן שהיה רשום ב-AX. זיהוי של מצב כזה הוא פשוט יחסית, כי אפשר לראות שאופרנד היעד הרשום בהוראה אחת הוא אותו אופרנד שבו משתמשת אחת מההוראות העוקבות.



איור 10.10 א
מצב של data hazard

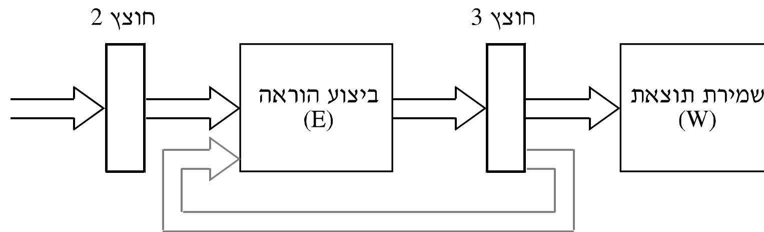
גם כאן הפתרון הפשוט יהיה השהיית שלב D של הוראת החיסור, עד שהערך החדש של אוגר AX יתעדכן.



איור 10.10 ב
שימוש בהשהיה כדי לפתור מצב של data hazard

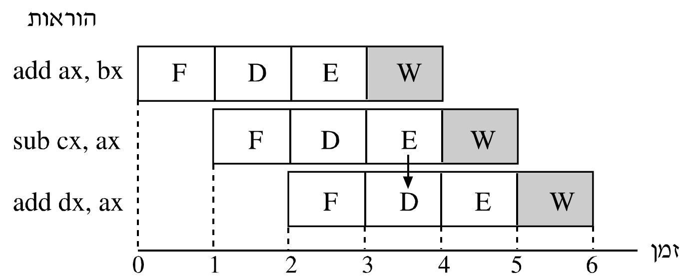
במקרה של השהיה, כל ההוראות הרשומות לאחר ההוראה שבה הושהה שלב מסוים, מושהות גם כן. ההוראות שנמצאות לפניה בתור, יכולות להמשיך את הביצוע כרגיל. בדוגמה שתיארנו, ביצוע קטע התכנית התארך ב-2 יחידות זמן.

פתרון אחר לבעיה זו הוא תכנון החומרה כך שלאחר חישוב הסכום, הערך יוחזר מיד לביצוע ביחידה האריתמטית-לוגית (בנוסף לכתיבתו בפעולה WB באוגר AX). במקרה כזה, כאשר תבוצע הוראת החיסור, הפעולה D2 לא תצטרך להמתין לקריאת הערך מהאוגר AX . לשיטה זו קוראים **קידום נתונים** (data forwarding). איור 10.11 מתאר בצורה מופשטת את השינוי הדרוש בחומרה.



איור 10.11
מבנה מעבד הכולל קידום נתונים (data forwarding)

אם נשתמש בשיטה זו, ביצוע התכנית יתקצר, כי השלב E מעביר את הנתונים ישירות לשלב הביצוע של הוראת החיסור, משום כך היא אינה צריכה להמתין לשלב W של הוראת החיבור (שבו הנתון נכתב באוגר AX).



איור 10.12
שימוש בקידום נתונים

הצגנו שני פתרונות המיושמים בחומרה (השהיית מעבד וקידום נתונים), אבל בחלק מהמעבדים פתרון בעיה זו מוטל על המהדר, ובמקרה כזה הפתרון הוא בתוכנה. נתאר שתי שיטת בהן מהדרים מתגברים על בעיית data hazard.

אפשרות ראשונה

המהדר מנסה לשנות את סדר ביצוע הפעולות, ולהכניס בין הוראת החיבור והוראת החיסור הוראה אחרת, שאינה תלויה באופרנדים המחושבים בהוראת החיבור. הנה לדוגמה סדרת ההוראות:

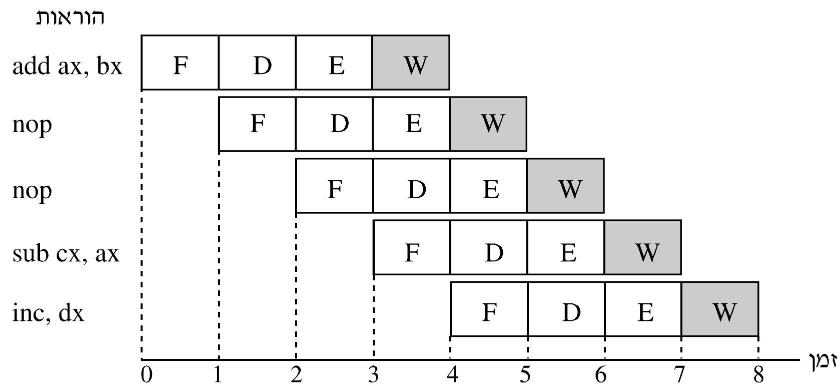
```
add  ax,  bx
sub  cx,  ax
inc  dx
```

בסדרת ההוראות הזו, ביצוע ההוראה INC DX אינו תלוי בביצוע שתי ההוראות האחרות, בעוד שביצוע ההוראה SUB CX, AX תלוי בביצוע ההוראה ADD AX, BX. נוכל לשנות את הסדר לפיו מתבצעות ההוראות, ולקבוע שהן יתבצעו בסדר הזה:

```
add  ax,  bx
inc  dx
sub  cx,  ax
```

אפשרות שנייה

אפשר להכניס הוראות דמה, NOP, שאינן מבצעות שום פעולה, אך משהות את זמן הביצוע של הוראת החיסור, עד שיסתיים הביצוע של הוראת החיבור.



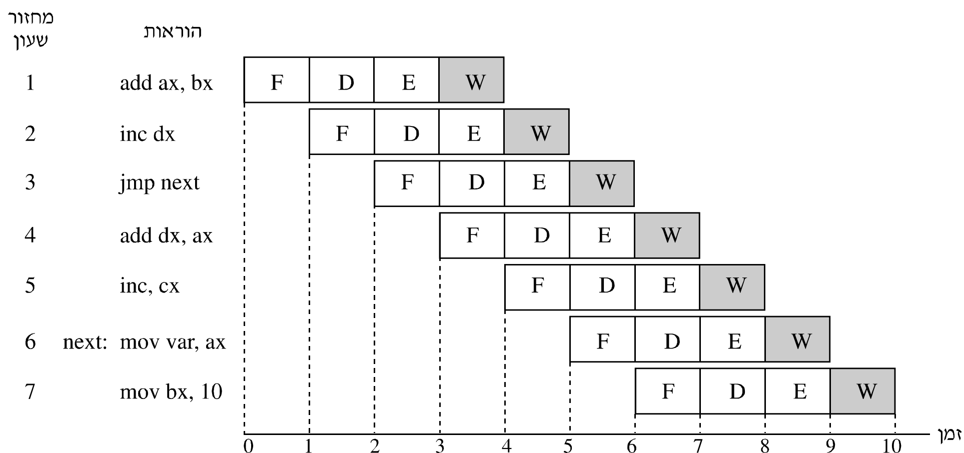
איור 10.13
הכנסת הוראות NOP כדי לפתור את בעיית data hazard

השיטה שבה משתמשים בהוראה NOP (שאין בה אופרנדים ולכן אינה מבצעת שום פעולה), ממששת בתוכנה את השימוש בהשהיה stall המבוצעת בחומרה.

ג. טיפול ב-Control Hazard

הפרעות לביצוע שוטף של הוראות בצינור הוראות מהסוג אחרון, נקראות Control Hazard. הפרעות מסוג זה מתרחשות כאשר קיימות הוראות בקרה שמעבירות את ביצוע ההוראה להוראה שאינה בהכרח ההוראה העוקבת. נזכור כי השימוש בתור ההוראות מבוסס על העיקרון שהוראות התכנית מתבצעות בצורה סדרתית, ולכן יחידת המישק לפס יכולה למלא את התור כשהיא מביאה בכל פעם את ההוראה הבאה, שכתובתה שמורה באוגר IP. אבל תכניות מכילות בדרך כלל גם הוראות בקרה, שמשנות את הביצוע הסדרתי של התכנית. נזכיר כמה מהוראות אלה: הוראות קפיצה מותנית, הוראות קפיצה בלתי-מותנית, לולאות, קריאה לפרוצדורה או חזרה ממנה.

נתאר כעת מה קורה כאשר תכנית מכילה הוראת קפיצה בלתי-מותנית. כפי שאפשר לראות באיור 10.14, לאחר ביצוע ההוראה השלישית, ההוראה JMP, התכנית צריכה להמשיך את הביצוע מההוראה שמוצמדת אליה התווית next (בדוגמה שלנו ההוראה השישית – הוראת MOV).

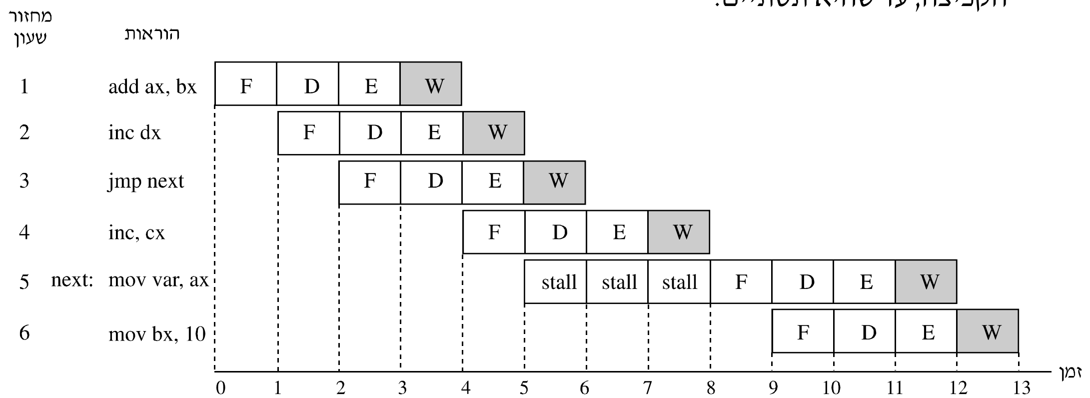


איור 10.14 א

אבל אפשר לראות כי במחזור השעון השלישי הובאה הוראת הקפיצה, ובמחזור השעון הרביעי היא נמצאת בשלב הפענוח D. באותו זמן הובאה ההוראה ADD DX, AX (והיא נמצאת בשלב F). במחזור השעון החמישי נמצאת הוראת הקפיצה בשלב הביצוע (E), הוראת ADD עוברת לשלב הפענוח D, וגם ההוראה INC מובאת (נמצאת בשלב F). אולם בביצוע תקין של התכנית הוראה זו כלל לא תתבצע כי לאחר עיבוד הוראת הקפיצה, נבצע את ההוראה MOV, שמוצמדת אליה התווית next. במקרה כזה, כאשר עיבוד הוראת

הקפיצה יסתיים, וכתובת הקפיצה תחושב ותעודכן באוגר IP, המעבד יצטרך לבטל את כל ההוראות שהוכנסו לתור אחרי הוראת הקפיצה, ובמקומן להכניס לתור את כל ההוראות הרשומות בתכנית אחרי ההוראה MOV. ככל שתור ההוראות עמוק יותר, זמן ה"התאוששות" הדרוש למעבד כדי להוציא מהתור את ההוראות שאינן נחוצות ולמלא את התור בהוראות שצריכות להתבצע, יהיה ארוך יותר. זו אחת הסיבות לכך שהוצבה מגבלה על עומק תור ההוראות.

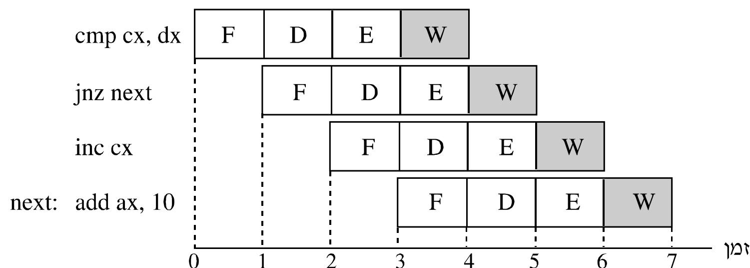
לפתרון בעיה זו משתמשים בהשהיה – משהים את ביצוע ההוראה העוקבת להוראת הקפיצה, עד שהיא תסתיים.



איור 10.14 ב

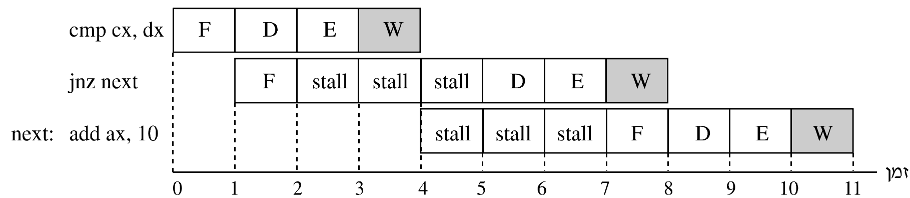
ביצוע ההוראה הרשומה לאחר הוראת הקפיצה, צריך להתעכב עד שיסתיים הביצוע של ההוראה JMP. במקרה הזה יתוספו לפחות 3 מחזורי שעון לזמן הביצוע של התכנית.

ביצוע הוראת קפיצה מותנית מסבך את העניינים עוד יותר, מפני שכעת הוראת הקפיצה צריכה להמתין עד שההוראה CMP תסתיים – דבר המשלב גם מצב של Data hazard. לדוגמה:



איור 10.15 א

בדוגמה שראינו בחלק א של איור 10.15, ההוראה JNZ צריכה להתבצע אחרי שמסתיימת ההשוואה של CX ל-DX. כמו-כן צריכה להתקבל החלטה מה תהיה הפעולה העוקבת להוראת הקפיצה המותנית, והחלטה זו יכולה להתקבל רק אחרי שתסתיים הוראת הקפיצה המותנית. כדי לפתור את הבעיה, מוסיפים מחזורי השהיה, כמתואר באיור 10.15 ב. בזמן ביצוע ההוראה JNZ הוספנו 2 מחזורי השהיה, כדי שחישוב התנאי יבוצע לאחר שביצוע ההשוואה הסתיים. כמן-כן הוספנו שלושה מחזורי השהיה במהלך ביצוע ההוראה ADD, כדי שההוראה הבאה לאחר ביצוע ההוראה JNZ תתבצע כאשר יעד הקפיצה כבר ידוע. הוספה של מחזורי השהיה מאריכה את ביצוע התכנית ב-5 יחידות זמן. הביצוע של 3 הוראות, ללא השהיה, אורך 6 יחידות זמן, ואילו ביצוע הכולל 5 מחזורי השהיה, נמשך 11 יחידות זמן – כמעט פי 2.



איור 10.15 ב

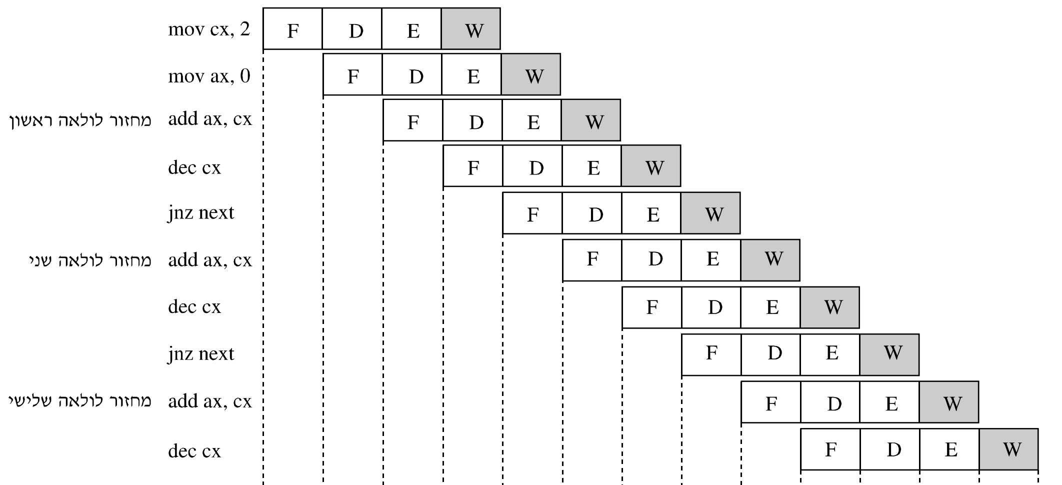
במעבדים מודרניים משתדלים למנוע איבוד זמן, ומשתמשים בשיטות מגוונות כדי לפתור את הבעיה של control hazard. בין שאר השיטות נוקטים בשיטת **ניבוי הסתעפות** (branch prediction). לפי שיטה זו, המעבד מנסה לנבא מה תהיה התוצאה של בדיקת התנאי, ובהתאם הוא מנסה לחזות מהי ההוראה אליה יצטרך "לקפוץ". ניבוי זה נעשה על סמך קריטריונים רבים ומגוונים.

בפנטיום, למשל, כאשר יש צורך לבחור בין הוראות, הקפיצה תתבצע להוראה הנמצאת בכתובת היותר נמוכה בתור ההוראות, או להוראה שבוצעה בעבר. כאשר מסתיים הביצוע של ההוראה שקובעת לאן צריך המעבד לקפוץ (שלב W), המעבד בודק את נכונות הנבואה כדי לבטל או לאשר את הקפיצה. לדוגמה, נתבונן בקטע התכנית הבא:

```
      mov  cx,  100
      mov  ax,  0
next: add  ax,  cx
      dec  cx
      jnz  next
      inc  bx
```

לפני חישוב ערכו של האוגר CX ועדכון הדגל ZF, המעבד מנבא כי יש לקפוץ לתווית NEXT (להוראה שבוצעה בעבר), לכן, כל עוד תנאי הלולאה מתקיים, הוא מביא לתור ההוראות את ההוראות העוקבות להוראה אליה מוצמדת התווית next. כאשר תנאי הלולאה לא יתקיים, המעבד יוציא מהתור את ההוראות שאינן נחוצות ויכניס במקומן את ההוראות העוקבות להוראת JNZ. בתכנית שלנו, תנאי זה מתקיים 99 פעם (מתוך 100), בצורות דומות הארכיטקטורה של ניבוי ההוראה הבאה מאפשרת שימוש יעיל בצינור ההוראות במקרים רבים.

איור 10.16 מדגים את ביצוע הלולאה בהנחה ש-CX = 2. אפשר לראות כי המחזור השלישי של הלולאה מתחיל לפני שהוראת JNZ הסתיימה והמעבד עדיין לא יודע שהלולאה הסתיימה. בשלב E של הוראת JNZ, במחזור הלולאה השני, כתובת הקפיצה חושבה, לכן המעבד יבטל את שתי ההוראות האחרונות בתור, ויביא את ההוראות הרשומות אחרי הוראת הקפיצה ; בדוגמה שלנו זו ההוראה INC BX.



איור 10.16

בשיטה זו, המעבד מצליח לנחש נכון את ההוראה הבאה לביצוע, כל עוד הלולאה מתבצעת, והוא טועה רק בביצוע המחזור האחרון של ההוראה. אם הלולאה צריכה להתבצע 100 פעם, המעבד יבצע 99 מחזורים בהצלחה ורק במחזור ה-100 הניבוי נכשל והמעבד יצטרך להוציא את ההוראות המיותרות מהתור ולהכניס את ההוראות הנכונות.

ביצוע במקביל – ארכיטקטורת SUPERSCALAR

בפנטיום קיימים כמה צינורות (Pipelines) המאפשרים לבצע במקביל כמה הוראות (שאינן תלויות זו בזו). למיקרו-מעבד במבנה מעין זה קוראים Superscalar. לפנטיום I יש שני צינורות (U Pipeline ו-V Pipeline) כפי שמראה איור 10.17. פירוש הדבר, שפנטיום I מסוגל לבצע שתי הוראות במקביל. לפנטיום III יש שלושה צינורות.

	Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6	Clock 7	Clock 8
PF u	I1	I3	I5	I7				
PF v	I2	I4	I6	I8				
D1 u		I1	I3	I5	I7			
D1 u		I2	I4	I6	I8			
D2 u			I1	I3	I5	I7		
D2 v			I2	I4	I6	I8		
EX u				I1	I3	I5	I7	
EX v				I2	I4	I6	I8	
WB u					I1	I3	I5	I7
WB v					I2	I4	I6	I8

איור 10.17
שתי יחידות ביצוע מקבילות

במקרה זה, אם שתי ההוראות העוקבות אינן תלויות האחת בתוצאת השנייה, ניתן לבצע אותן בו-בזמן בשתי יחידות הביצוע הנפרדות, וכך להחיש את קצב ביצוע התכנית פי 2. כל אחת משתי היחידות, v-pipeline ו-u-pipeline, מסוגלת לבצע הוראה אחת במחזור שעון יחיד, לכן מסוגל הפנטיום לבצע שתי הוראות במחזור שעון יחיד. הקצב שלו בביצוע הוראות הוא כפול מהקצב של המעבד 486. אך השיפור הוא חלקי, משום שבעוד שצינור הוראות U מסוגל לטפל בכל הוראה, צינור הוראות V מטפל רק בחלק מההוראות. לכל אחת משתי יחידות הצינור בפנטיום יכולת דומה לזו של המעבד 486, והוא כולל חמישה שלבים.

איור 10.17 מדגים ביצוע 8 הוראות המסומנות באותיות I1-I8 בצינור הוראות כפול.

- השלב הראשון, המסומן באיור 10.17 באותיות PF (PreFetch), אחראי להבאת שתי ההוראות, כשבמרבית המקרים ההוראות הבאות (עבור שתי יחידות הביצוע) נמצאת כבר בתור ההוראות. תור ההוראות במעבד הוא תור כפול, כדי ליעל את ביצוע הוראת הקפיצה. תור אחד מכיל את ההוראות לביצוע במקרה שהוראת הקפיצה לא תבוצע, והתור השני כולל את ההוראות במקרה של קפיצה. לאחר שיתברר אם הוראת הקפיצה בוצעה או לא, אחד התורים יהפוך לתקף והשני לא.

- השלב השני, המסומן באיור 10.17 ב-D1, הוא שלב פענוח ההוראות. בשלב זה מתבצעת ההחלטה אם ניתן לבצע את שתי ההוראות בו-זמנית. ביצוע הוראות בו-זמנית יימשך רק בתנאים מסוימים: ההוראות חייבות להיות פשוטות וחייבת להתקיים אי תלות בין הנתונים של שתי ההוראות.

- החל מהשלב השלישי, המסומן באיור 10.17 ב-D2, צינור ההוראות של שתי יחידות הביצוע נפרדות לחלוטין. בשלב השלישי כל צינור הוראות מבצע חישוב של כתובות הנתונים בזיכרון (אם יש כתובות כאלה), וקורא אותן.

- בשלב הרביעי, המסומן באיור 10.17 ב-EX, מבוצעות ההוראות ב-ALU. שתי ההוראות מגיעות בו-זמנית לשלב הביצוע. הזמן הדרוש לביצוע כל אחת משתי ההוראות יכול להיות שונה. למשל, אם ביצוע ההוראה מהצינור U (u-pipeline) הסתיים לפני ביצוע ההוראה מהצינור V (v-pipeline), המעבד לא ירשה להוראה חדשה להיכנס לשלב הביצוע בצינור U (u-pipeline) עד שיושלם ביצוע ההוראה בצינור V (v-pipeline).

- בשלב האחרון, המסומן באיור 10.17 ב-WB, בביצוע ההוראות נשמרת התוצאה. אמרנו כי הפנטיום יכול לבצע שתי הוראות במספרים שלמים, במחזור שעון יחיד. אולם להוראות בנקודה צפה דרושים שלושה מחזורי שעון, ובמרבית המקרים יכול המעבד לבצע בו-זמנית רק הוראה אחת בנקודה צפה.

10.4 ארגון זיכרון

10.4.1 אופני עבודה

במעבדים ממשפחת 86× יש כמה אופני עבודה, המאפשרים לתכניות שנכתבו עבור מעבדים מהדורות הקודמים, להתבצע על מעבדים חדישים יותר. אופני העבודה הם:

אופן עבודה "אמיתי" (Real mode)

באופן פעולה זה פועל המעבד כמו 8086, הוא מאפשר גישה לזיכרון של 1MB בלבד. כדי לממש אופן עבודה זה, המעבד משתמש רק ב-20 קווי כתובת נמוכים בפס הכתובות. אופן עבודה זה הוא אופן העבודה הבסיסי שמתקבל עם הפעלת המחשב.

אופן עבודה מוגן (Protected mode)

אופן עבודה זה תומך במנגנוני המקטעים ובדפדוף (אותם נתאר בהמשך); הוא מאפשר הרצה בו-זמנית של כמה תכניות וגם הרצה של תכניות ארוכות, שאורכן (יחד או כל אחת לחוד) עולה על מרחב הזיכרון הראשי (למשל במעבד 80386 להריץ תכניות שאורכן גדול מ-4GB).

אופן עבודה "וירטואלי-אמיתי" (Virtual real mode)

באופן עבודה זה תכניות של מעבד 8086 יכולות לפעול בשני אופנים: באופן עבודה "אמיתי" ובאופן עבודה "וירטואלי" המאפשר למספר תכניות של מעבד 8086 להתבצע בו זמנית כאשר לכל תכנית יש מרחב זיכרון של 1MB. מרחב זיכרון זה יכול להיות מורחב למרחב זיכרון פיזי (למשל ב-40385 ל-4GB) באמצעות מנגנון הנקרא "דפדוף".

טבלה 10.2 מפרטת את אוגרי המקטע ואת אוגרי ההיסט, הן בעבודה במצב אמיתי והן בעבודה במצב מוגן.

טבלה 10.2
אוגרי המקטע ואוגרי ההיסט בעבודה במצב אמיתי ובעבודה במצב מוגן

אוגר המקטע	הכתובת היחסית (היסט)	סוג הגישה לזיכרון	MODE
CS	IP	הבאת הוראות	REAL MODE (16 סיביות)
SS	SP, BP	פעולה מחסנית	
DS	BX, DI, SI, מספר בן 8 או 16 סיביות	נתונים כלליים	
ES	DI	פעולה עם מחרוזות	
CS	EIP	הבאת הוראות	PROTECTED MODE (32 סיביות)
SS	ESP , EBP	פעולה מחסנית	
DS	EBX, EAX, ECX, EDX, ESI, EDI, מספר בן 8 או 32 סיביות	נתונים כלליים	
ES	EDI	פעולה עם מחרוזות	
FS	אין ברירת מחדל	נתונים כלליים	
GS	אין ברירת מחדל	נתונים כלליים	

10.4.2 סוגי הזיכרון

במחשב קיימים כמה סוגים של זכרונות:

- האוגרים הם זיכרון טיוטה בהם משתמש המעבד בזמן ביצוע המחזור הבאה-וביצוע כדי לאחסן את קוד הפעולה ואת האופרנדים של כל הוראה בתכנית. העברת מידע בין האוגרים לבין עצמם, או בין האוגרים ליחידת ה-ALU, מתבצעת באמצעות פסים פנימיים, במהירות גדולה מאוד, הנמדדת בננו-שניות (1 ננו שנייה היא 10^{-9} שנייה).
- בזיכרון הראשי מאוחסנות תכניות (המכילות הוראות ונתונים) המורצות במחשב. במהלך המחזור הבאה-וביצוע של הוראות התכנית, הן נקראות בזו אחר זו ומועברות אל המעבד באמצעות פסי הנתונים, פסי הכתובות ופסי הבקרה. זמן העברת המידע בין המעבד לזיכרון הראשי הוא איטי יחסית, הוא נמשך כ-100 ננו שניות (כלומר פי 100 בערך מזמן העברת המידע בתוך המעבד). קצב העברה זה מאריך את זמן הביצוע של התכנית, כי המעבד, שפונה לזיכרון כדי לקרוא את ההוראה הבאה לביצוע, צריך להמתין עד שההוראה תועבר בפס הנתונים. נזכיר שני מאפיינים חשובים נוספים של

הזיכרון הראשי: א. הוא זמני – תוכנו נמחק כאשר מכבים את המחשב; ב. גודלו מוגבל על-ידי רוחב פס הכתובות.

- הזיכרון המשני משמש לשמירת התכניות לאורך זמן. זכרונות משניים, כמו דיסק קשיח ותקליטור, הם זכרונות קבועים – תוכנם אינו נמחק כאשר מכבים את המחשב. כמות המידע שאפשר לשמור בזיכרון משני היא גדולה מאוד. לדוגמה: הגודל הסטנדרטי של דיסק קשיח, שבו משתמשים כיום במחשבים ביתיים, הוא 80GB. בנוסף אפשר לחבר למחשב כמה יחידות חיצוניות, וכך להגדיל את כמות הזיכרון. כאשר רוצים להריץ תכנית, יש לטעון אותה תחילה לזיכרון הראשי. זמן העברת הנתונים בין זיכרון משני לבין הזיכרון הראשי, נמדד במילי שניות (1 מילי שנייה היא אלפית השנייה). זמן זה גדול פי 10000 בקירוב מזמן העברת נתונים בין הזיכרון הראשי לבין המעבד.

שאלה 10.1

חשבו פי כמה גדול הזכרון שיש בדיסק קשיח, בגודל 80GB, מגודל הזיכרון של מעבד שבו רוחב פס הכתובות הוא בן 64 סיביות.

כדי לשפר את ביצועי המחשב, מאמצים רבים מושקעים בשיפור הארכיטקטורה של הזיכרון, כדי לקצר, מצד אחד, את זמן העברת הנתונים בין המעבד לזיכרון הראשי, ומצד שני להגדיל את הקיבולת של הזיכרון הראשי; שיפורים אלה מאפשרים להריץ תכניות גדולות מאוד ולבצע כמה תכניות במקביל. בסעיף זה נציג את השימוש בזכרונות מטמון (cache memory) לשיפור זמן הגישה, ואת השימוש בזיכרון וירטואלי (virtual memory) כדי להגדיל לכאורה את קיבולת הזיכרון הראשי.

10.4.3 זיכרון מטמון (Cache Memory)

כדי לקצר את זמן העברת המידע בין המעבד והזיכרון הראשי הוסיפו יחידת זיכרון מהיר כחוצץ בין הזיכרון הראשי למעבד, הנקראת זיכרון מטמון. גודלו של זיכרון מטמון נע בין מאות KB לבין מספר MB אחדים; זמן העברת הנתונים בינו לבין המעבד הוא כ-10 ננו-שניות.

בזמן הרצת תכנית, יחידת המישק לפס קוראת חלק מההוראות של התכנית ומכניסה אותן לזיכרון המטמון; ההוראות שצריכות להתבצע מועתקות מזיכרון המטמון אל תור ההוראות.

איור 10.18
הירכיית זכרונות

ביצוע תכנית דורש אופטימיזציה של המשאבים העומדים לרשות המחשב, כדי שמספר הפעמים שהמעבד יצטרך לגשת לזיכרון הראשי (ולא למטמון) יהיה מינימלי. כיוון שגודלו של זיכרון המטמון מוגבל, נוקטים בשיטות שונות כדי להבטיח שהמעבד ימצא שם, ברוב הזמן, את ההוראות והנתונים הדרושים לו. כאשר תכנית צריכה לקרוא הוראה או נתון מסוים, המעבד מחפש קודם-כל בזיכרון המטמון. אם המידע הדרוש אינו נמצא בזיכרון המטמון, המעבד פונה אל הזיכרון הראשי וקורא ממנו את המידע הדרוש.

אסטרטגיה יעילה לניהול זיכרון צריכה להחליף את תוכן זיכרון המטמון כך שמידע שאינו דרוש יפונה ממנו, ובמקומו יאוחסן מידע חדש, שהתכנית תזדקק לו בהמשך, ובכך לצמצם את מספר הפעמים שהמעבד צריך לגשת לזיכרון הראשי.

אחת השיטות מבוססת על ההבחנה, שבעת ביצוע תכנית יש נטיה ברורה של כל הפניות לזיכרון להתייחס לתחום קטן בזיכרון במשך פרקי זמן קצרים. תכונה זו נקראת **תכונת המקומיות** (locality of reference); עקב תכונה זו, ניתן לפנות מזיכרון המטמון את ההוראות שבוצעו ולקרוא אליו את ההוראות הסמוכות לקטע הקודם.

המדד לאיכות ביצועי זיכרון המטמון מבוסס על אלגוריתם חישובי; הוא נקרא אחוז הפגיעה (Hit ratio). תפקידו למדוד כמה גישות לזיכרון המטמון אכן מצאו בתוכו את המידע המבוקש (ולכן לא נדרשה פנייה לזיכרון הראשי), לעומת מספר הפניות הכולל. אחוז הפגיעה המקובל בזיכרון מטמון של 8KB (גודלו במעבד 80386) הוא 70-80%. הרחבה של זיכרון המטמון ל-32KB תעלה את אחוז הפגיעה ל-85-90%. הרחבת זיכרון המטמון מעבר ל-32KB, אינה מבטיחה שביצוע המערכת ישתפר תמיד, אלא רק במקרים שבהם יש לתכנית "התנהגות" הפונה באקראי למקומות שונים בזיכרון, ולא ברצף, כמקובל ברוב התכניות. לזיכרון מטמון גדול יש יתרון ניכר במערכות מרובות-משימות, לעומת מערכות המסוגלות לפעול רק על משימה אחת, כמו מערכת ההפעלה DOS.

טכנולוגיה של מעבדים מתקדמים מאפשרת לארגן כמה רמות של זיכרון מטמון: חלק מהן נמצאות על פיסת הסיליקון של המעבד והאחרות הן יחידות חיצוניות. כדי להבדיל ביניהן נהוג להגדיר רמות של זיכרון מטמון: **רמה 1** של זיכרון מטמון הוא יחידת זיכרון הנמצאת במעבד עצמו (כחלק מהמעבד); **רמה 2** של זיכרון מטמון היא חיצונית למעבד (רכיב נפרד). הרמות נקבעו בהתאם לזמן הגישה הדרוש להן. זמן הגישה המהיר ביותר הוא בין זיכרון מטמון ברמה 1 לבין המעבד. כאשר מתרחקים מהמעבד, גודל הזיכרון הולך וגדל וגם זמן הגישה לרכיב מתארך, בהתאמה.

השימוש בזיכרון מטמון להוראות, משפר גם את זמן הביצוע של תכנית הכוללת הוראות קפיצה (control hazard). בסעיף הקודם תיארנו מצב שבו הביצוע של הוראת קפיצה גורם לכך שהמעבד צריך לנקות את תור ההוראות ולהביא את ההוראות שיש לבצע. כאשר משתמשים בזיכרון מטמון, המעבד קורא את ההוראות הדרושות במהירות גדולה יותר, וכך מתקצר זמן ההמתנה של יחידת הביצוע, ובהתאם ביצוע התכנית כולה.

השימוש בזיכרון מטמון פותר חלק מן הבעיות הנובעות מתחרות על משאבים (structural hazard) – בעיות שנוצרו כתוצאה משימוש בצינור הוראות. בסעיף הקודם הצגנו מצב שבו שתי פעולות שונות השתמשו בפסים באותו זמן: בתור ההוראות, פעולה אחת נמצאה בשלב קריאת ההוראה (שלב F) ופעולה אחרת קראה אופנרד מהזיכרון (שלב D); בשתי הפעולות האלה המעבד השתמש בפסים כדי לקרוא מהזיכרון. הסברנו שמעבדים מתקדמים מכילים שני סוגים של זכרונות מטמון, כדי לפתור בעיה זו:

- זיכרון מטמון להוראות, שבו נשמר חלק מהוראות התכנית;
- זיכרון מטמון לנתונים שבו נשמר חלק מנתוני התכנית.

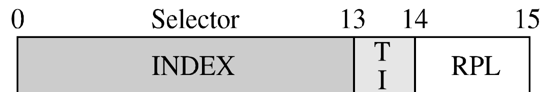
שימוש בשני סוגים של זכרונות מטמון מונע את ההתנגשות שנובעת מתחרות על הפסים ובדוגמה שתיארנו תהיה פנייה לזיכרון מטמון להוראות בפעולת ההבאה F, ובפעולה D תהיה פנייה לזיכרון מטמון נתונים.

10.4.4 קיטוע (סגמנטציה) והרצה של כמה תכניות במקביל

סגמנטציה היא שיטה המאפשרת לממש אופן עבודה מוגן, שבו מורצות כמה תכניות במקביל. באופן עבודה זה, מוקצים לכל תכנית כמה מקטעים (כמו מקטע קוד, מקטע מחסנית) שגודלם משתנה (בניגוד למעבד 8086 שבו הגודל של מקטע מוגבל ל-64KB).

בנוסף קיים מנגנון הגנה, המונע מתכנית אחת אפשרות לגשת למקטעי הזיכרון של תכנית אחרת. כדי לממש דרישות אלה, המעבד משתמש בכמה טבלאות המכילות פרטים על מיקום המקטעים שהוקצו לתכניות ועל מיקום המקטעים הפנויים בזיכרון. טבלאות אלה נקראות Descriptor Tables והן שוכנות בזיכרון הראשי. כל שורה (כניסה) בטבלה היא בת 8 בתים שמכילים את המידע הזה: כתובת בסיס של המקטע (32 סיביות), גודל המקטע (20 סיביות), רמות הרשאה ומידע נוסף על המקטע.
הסבר על רמות הרשאה

כדי לתרגם כתובת לוגית לכתובת פיזית, המעבד משתמש באוגר מקטע ובהיסט הרשום בהוראה עצמה. תפקיד אוגרי המקטע באופן העבודה המוגן הוא שונה – הם אינם מצביעים על כתובת הבסיס של המקטע, אלא על כניסה בטבלת המקטעים. אוגרי המקטע מכילים מידע נוסף, המציין לאיזו טבלת מקטעים יש לפנות ומהן רמות הגישה למקטע. איור 10.19 מתאר את המבנה של אוגר מקטע; באופן עבודה מוגן מבנה זה נקרא גם **אוגר בורר** או Selector.



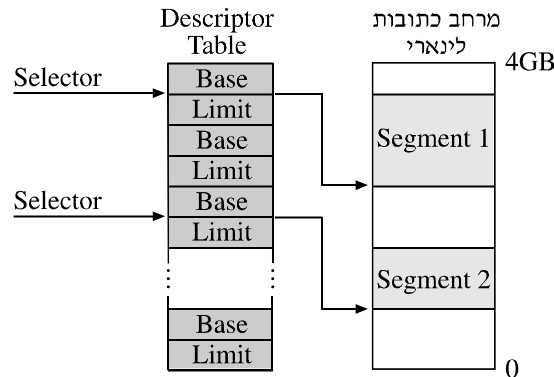
איור 10.19
מבנה אוגר מקטע (selector)

אוגר המקטע מכיל שלושה סוגים של מידע (שהשימוש בהם יוסבר בהמשך):

א. 13 סיביות עליונות המסומנות כ-Index מכילות מספר מקטע.
ב. סיבית אחת המסומנת כ-TI (קיצור של Table Indicator) מציינת את סוג טבלת המקטעים שבשימוש (כללית או מקומית).
ג. 2 סיביות תחתונות המסומנת כ-RPL (קיצור של Request Privilege Level) מציינות את רמת ההרשאה.

תהליך התרגום של כתובת לוגית לכתובת פיזית כולל שני שלבים (ראו איור 10.20).

א. בעזרת 13 הסיביות העליונות באוגר המקטע, המעבד שולף את כתובת הבסיס של המקטע מטבלת המקטעים;
ב. הוא מוסיף את ההיסט לכתובת הבסיס של המקטע.



איור 10.20
תרגום כתובת לוגית לכתובת פיזית

הסימון Base באיור 10.20 מציין כתובת של תחילת המקטע בזיכרון לינארי (שגודלו 4GB) והסימון Limit קובע את גודלו של המקטע.

במחשב מנוהלים שני סוגים של טבלאות; מערכת ההפעלה משתמשת בטבלאות האלה לניהול הזיכרון של המחשב:

- **טבלת מקטעים כללית** (Global Descriptor Table) **GDT**
  בטבלה זו מוחזק מידע על קטעי הזיכרון ששייך למערכת ההפעלה ומידע על על קטעי זיכרון פנויים שאפשר להקצות אותם לתכנית.
- **טבלת מקטעים מקומית** (Local Descriptor Table) **LDT**
  טבלה זו מוגדרת לכל תכנית מורצת, ובה נשמר מידע על המקטעים שהוקצו לה.

בזמן אתחול המחשב, מערכת ההפעלה יוצרת טבלת GDT שבה נשמר מידע על הזיכרון כולו. כאשר תכנית נכנסת למצב ריצה, מערכת ההפעלה בונה טבלת LDT שבה נשמר מידע על קטעי הזיכרון שהוקצו לה. קטעי הזיכרון המוקצים לתכנית הם קטעי זיכרון שנמצאים בטבלת המקטעים הכללית ומוגדרים כקטעי זיכרון פנויים, שהרשאת הגישה אליהם מתאימה לתכנית. מנגנון תרגום בעזרת הטבלאות GDT ו-LDT זהה, כאשר ביט TI ב-Selector בורר לאיזו טבלה נגשים, כאשר ערכו הוא '1', המעבד ניגש לטבלה המקומית (LDT) וכאשר ערכו הוא '0' אוגר הבורר פונה לטבלה הכללית (GDT).

כדי לממש את מנגנון ההגנה, משתמש המעבד בשתי הסיביות האחרונות של אוגר המקטע המציינות את רמת ההרשאה של התכנית. שתי סיביות אלה מאפשרות להגדיר 4 רמות הגנה, כאשר הרמה הגבוהה ביותר מוענקת למערכת ההפעלה והרמה הנמוכה ביותר לתכנית.

כאשר הוראה בתכנית ניגשת לזיכרון, נבדקת רמת ההרשאה של התכנית לפי ערכם של אוגרי המקטע המוגדרים לה. אם הכתובת המבוקשת נמצאת בתחום ההרשאות המתאים, המעבד מאפשר לתכנית לגשת לכתובת המבוקשת, ואם הכתובת אינה בתחום המותר – יאסור המעבד גישה לכתובת זו, ומערכת ההפעלה תודיע על שגיאה. זו ההודעה שמופיעה לעיתים במערכת "חלונות" (Windows) ומודיעה שתכנית כלשהי ביצעה גישה לא חוקית לזיכרון ולכן היא תיסגר.

כדי לדעת את מיקום הטבלאות בזיכרון, המעבד משתמש בשני אוגרים:

- GDTR מצביע לטבלה GDT, הוא מאותחל כאשר המחשב מופעל ומערכת ההפעלה יוצרת את טבלת GDT;
- LDTR מצביע לטבלה LDT של התכנית והוא מאותחל כאשר התכנית מתבצעת.

לסיום נציין כי אי-אפשר לבצע תכניות במקביל באופן אמיתי, כלומר בו-זמנית, במערכת שיש בה מעבד יחיד. אמרנו שמעבד יחיד מבצע פקודות בזו אחר זו, אך עם זאת, הוא מאפשר להפסיק פעולה של תכנית (להפעיל פסיקה), לעבור לתכנית אחרת לפרק זמן מסוים, ולחזור לתכנית הקודמת. המעבר חייב להיעשות בצורה תקינה, כך שהמשתמש לא ירגיש בו. המעבד מספק כלים למעבר מביצוע תכנית אחת לשנייה על-ידי שמירה מלאה של התכנית המופסקת והצבה של מנגנוני הגנה המאפשרים לה להתבצע ללא הפרעות של תכניות אחרות. המעבר בין התכניות נעשה על-ידי תוכנה השולטת על המערכת, דהיינו, מערכת ההפעלה, כשפרק הזמן הנדרש לביצוע מעברים אלה מתארך ככל שיש לבצע יותר תכניות. מספר התכניות שניתן לבצע במקביל תלוי באותו חלק של מערכת ההפעלה השולט על ריבוי המשימות.

קיימות מערכות מחשב הכוללות כמה מעבדים; מבנה כזה מאפשר לכל תכנית להתבצע על מעבד אחר, או לבצע קטעי קוד של תכנית אחת על מעבדים שונים. כיום מקובלים

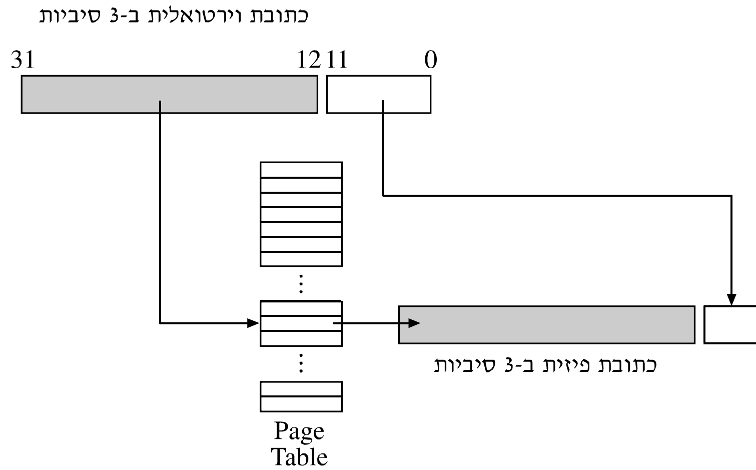
במחשבים ביתיים מעבדים כפולי-ליבה (Dual-Core processors), כאשר כל יחידה כזו מכילה, בעצם, שני מעבדים

10.4.5 זיכרון מדומה (Virtual memeory)

למרות השיפור המשמעותי בגודל הזיכרון הראשי המשמש להרצת תכניות, קיבולת זו אינה מספיקה ליישומים המכילים מידע רב (לדוגמה יישומי מולטימדיה) ויישומים גדולים מאוד המכילים מאות אלפי הוראות. כדי להתגבר על בעיות אלה משתמשים בזיכרון מדומה (virtual memory). **זיכרון מדומה** מאפשר לכל תכנית שמורצת במחשב להשתמש בזיכרון גדול יותר מהזיכרון הפיסי, כאשר חלק מהתכנית שאינה מעובדת באותו רגע יושבת בזיכרון משני, ואילו לזיכרון הראשי טוענים רק קטע מכל תכנית. תהליכי הטעינה של קטעים מהזיכרון המשני אל הזיכרון הפיסי, והשמירה של קטעים מהזיכרון הפיסי אל הזיכרון המשני, הם שקופים לתכנית המורצת, ומבחינתה הוקצה לה שטח זיכרון גדול כרצונה. כאשר יש צורך לגשת לנתון או הוראה שאינם נמצאים בזיכרון, קיים מנגנון שתפקידו להוציא מהתכנית קטע שאינו בשימוש באותו רגע ולהכניס במקומו את הקטע המבוקש. מערכת ההפעלה מופקדת על פעולות אלה, ומבחינת המתכנת היא שקופה.

כדי להשתמש בזיכרון מדומה יש צורך במנגנון שיתרגם את הכתובת הוירטואלית (היא הכתובת אליה מתייחס המתכנת בתכנית) לכתובת פיזית (שהיא כתובת הנתון או ההוראה בזיכרון הראשי). בנוסף לכך, מערכת ההפעלה צריכה לדאוג שקטע התכנית והנתונים, הדרושים באותו רגע לביצוע התכנית, יימצאו בזיכרון הראשי.

בניהול זיכרון מדומה המעבד משתמש בשיטה הנקראת דפדוף (paging), לפיה הזיכרון הראשי מחולק ל-"מסגרות דף" (page frames) בגודל קבוע, ובהתאמה לכך הזיכרון המשני מחולק לדפים (pages) בגודל קבוע. במעבדים של אינטל ממשפחת 86× גודל דף הוא 4Kbyte. לכל דף מוגדרת כתובת פיזית של תחילת הדף, השמורה בטבלת דפים (הנקראת TLB). משתמשים בטבלה זו לתרגום של כתובות מדומות לכתובות פיזיות. כתובת מדומה מחולקת לשני שדות: הסיביות העליונות מגדירות את מספר מסגרת הדף, שהוא גם מספר הכניסה בטבלת הדפים והסיביות הנמוכות מגדירות את המיקום במסגרת דף. לדוגמה: במעבד שבו הכתובת היא 32 סיביות: 20 הסיביות העליונות מגדירות את מספר הכניסה ו-12 הסיביות הנמוכות משמשות כאינדקס למיקום בתוך מסגרת הדף.



איור 10.21

כדי שחיפוש יהיה מהיר, טבלת הדפים צריכה להימצא בזיכרון הראשי. אולם שטח האחסון הדרוש לטבלה שיש בה 2^{20} כניסות (שהם 1Mbyte) הוא גדול מאוד. כדי לחסוך בזיכרון משתמשים בכמה רמות של טבלת דפים. למשל, נחלק את הכתובת ל-3 שדות.

10 bit	10 bit	12 bit
רמה ראשונה	רמה שנייה	היסט

כעת גודל הטבלה הראשונה הוא 2^{10} שהם 1 Kbyte. הטבלה ברמה ראשונה נשמרת בזיכרון ראשי וטבלאות ברמה השנייה נשמרות בזיכרון המשני. כדי ליעל את החיפוש משתמשים בזיכרון מטמון שמאפשר לשמור את הטבלה של הרמה הראשונה. כאשר מנהלים טבלאות דפים בשתי רמות, ומחפשים דף מסוים, יש לגשת תחילה לטבלת הדפים ברמה הראשונה, לחפש שם את כתובת טבלת הדפים של הרמה השנייה ולאחר מכן למצוא את כתובת הדף בטבלת הדפים ברמה השנייה. כלומר, אנו משלמים בזמן גישה (שתי גישות לזיכרון) תמורת החיסכון בזיכרון.

בדומה לניהול של זיכרון מטמון, מערכת ההפעלה צריכה לדאוג לפַנות מהזיכרון הראשי דפים שבאותו רגע אינם בשימוש התכנית המתבצעת, ולהכניס במקומם דפים חדשים להם התכנית זקוקה. כדי לדאוג שהמעבד יבצע הוראות באופן רציף, קיימים אלגוריתמים שונים בהם משתמשת מערכת ההפעלה, כדי להחליף את קטעי הזיכרון ולדאוג שברוב הזמן, קטע הזיכרון הרלוונטי יימצא בזיכרון הראשי.