שפת אסמבלי
למחשב האישי

מהדורה שנייה

קבצי המקור נמצאים באתר הוד-עמי

www.hod-ami.co.il

בקטגוריה "ספרי תכנות" בחר בספר
ולחץ על הלינק "קוד מקור". עקוב אחר ההוראות
והקבצים ייפתחו ל- C:\HodAmiBooks\59238

מוקדש
לאשתי חנה ובתי מירי
היקרות והאהובות

ברצוני להודות למורי ועמיתי, אשר הואילו לעבור על כתב היד, העירו והאירו:
רבקה הלפמן, הרצל נח, שמואל כהן.

עורך ראשי: **יצחק עמיהוד**

עריכה לשונית ועיצוב: **קרן לנדאו, שרה עמיהוד**

עיצוב עטיפה: **שרון רז**

פרק **19** נכתב על ידי **אסף שלי**

תודתנו **למר יהודה ויכסלפיש** על הערותיו המועילות

שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. **הוצאת הוד-עמי** עשתה כמיטב יכולתה למסור מידע אודות השמות המסחריים המוזכרים בספר זה ולציין את שמות החברות, המוצרים והשירותים. שמות מסחריים רשומים (registered trademarks) המוזכרים בספר צוינו בהתאמה.

הודעה

ספר זה מיועד לתת מידע אודות מוצרים שונים. נעשו מאמצים רבים לגרום לכך שהספר יהיה שלם ואמין ככל שניתן, אך אין משתמעת מכך אחריות כלשהי.
המידע ניתן "כמות שהוא" ("as is"). הוצאת הוד-עמי אינה אחראית כלפי יחיד או ארגון עבור כל אובדן או נזק אשר ייגרם, אם ייגרם, מהמידע שבספר זה, או מהתקליטור המצורף.

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

❑ **טלפון: 09-9564716**

❑ **פקס: 09-9571582**

❑ **דואר אלקטרוני: Info@hod-ami.co.il**

❑ **אתר באינטרנט: http://www.hod-ami.co.il**

שפת אסמבלי למחשב האישי

מהדורה שנייה

אלי כהן

הוצאת
הוד-עמי
ספרי מחשבים

The PC Assembly Language

2 Edition

By **Eli Cohen**

Editor: **I. Amihud**

(C)

כל הזכויות שמורות

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108 הרצליה 46160
טלפון: 09-9564716 פקס: 09-9571582
info@hod-ami.co.il

אין להעתיק או לשדר בכל אמצעי שהוא ספר זה או קטעים ממנו בשום צורה ובשום אמצעי אלקטרוני או מכני, לרבות צילום והקלטה, אמצעי אחסון והפצת מידע, ללא אישור בכתב מאת ההוצאה, אלא לשם ציטוט קטעים קצרים בציון שם המקור.

הודפס בישראל 1999

עדכון 2000, 2001, 2004

All Rights Reserved
HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL

ISBN 965-361-207-7 מסת"ב

תוכן עניינים מקוצר

הקדמה..15

פרק 1: מושגי יסוד...17

פרק 2: פקודות בסיסיות.......................................27

פרק 3: כתיבת תוכנית שלמה..................................53

פרק 4: תרגול מעשי..57

פרק 5: לולאות..75

פרק 6: פסיקות...105

פרק 7: כתובות זיכרון..131

פרק 8: מושגים...141

פרק 9: אוגר הדגלים...143

פרק 10: עבודה בסיביות.......................................159

פרק 11: משתנים ומערכים.....................................205

פרק 12: המחסנית..227

פרק 13: פרוצדורות..241

פרק 14: קבועים (Constants)...............................257

פרק 15: אסמבלי מותנה (Conditional Assembly) 261

פרק 16: מאקרו (Macro).....................................267

פרק 17: פקודות מחרוזת.......................................277

פרק 18: פקודות נוספות.......................................289

פרק 19: אסמבלי ושפת C.....................................297

אינדקס...307

תוכן העניינים

הקדמה 15

פרק 1: מושגי יסוד 17

המיקרומעבד 17
שפות תכנות 19
יחידות אחסנה 21
בסיסי ספירה 21
ייצוג מספרים........ 21
הזיכרון........ 22
האוגרים 23
אוגרים כלליים (General Purpose registers)........ 23
אוגרי המקטע (Segment registers) 24
אוגרים מצביעים (Pointer registers)........ 24
אוגר הדגלים (Flags register)........ 25

פרק 2: פקודות בסיסיות........ 27

כיצד כותבים פקודות בשפת אסמבלי 27
הפקודה MOV (Move) 28
העתקה ישירה לאוגר........ 28
העתקת נתון מאוגר לאוגר 29
"כלל ברזל" 1: התאמה בגודל האופרנדים 30
העתקת נתון מאוגר לתא זיכרון........ 30
"כלל ברזל" 2: פנייה לזיכרון 31
העתקת נתון מהזיכרון לאוגר 32
דוגמאות לתוכניות ופתרונות 32
תרגילים 35
הפקודה INC (Increment)........ 36
הפקודה DEC (Decrement) 36
הפקודה ADD (Addition) 36
הפקודה SUB (Subtract) 37
הפקודה NOP (No Operation)........ 37
הפקודה CMP (Compare)........ 37
הפקודה JMP (Jump) 38
תווית (label) 38
פקודת קפיצה מותנית (Conditional Jump)........ 39
תוכנית לדוגמה 39

הצבת ערכים הקסדצימליים באוגר 40
תרגילים ופתרונות 40

פרק 3: כתיבת תוכנית שלמה 53
מהי תוכנית שלמה? 53

פרק 4: תרגול מעשי 57
כיצד מריצים תוכנית אסמבלי? 57
תרגיל מעשי: שלבי תכנות, הרצה ובדיקה 58
כתיבת תוכנית דוגמה באמצעות עורך 58
תרגום התוכנית באמצעות אסמבלר MASM 59
קישור באמצעות תוכנת LINK 59
הרצת התוכנית ובדיקתה על ידי תוכנת DEBUG 59
תרגילים מעשיים לדוגמה 61
תרגיל מעשי א' 61
תרגיל מעשי ב' 62
תרגיל מעשי ג' 63
שלבי הרצת הבדיקה באמצעות DEBUG 64
תרגיל מעשי ד' 65
השלבים לכתיבת התוכנית והרצתה 65
שלבי ההרצה והבדיקה 66
טעויות אופייניות בתכנות: הסבר וסיכום 67
שגיאות בהרצת MASM 67
שגיאות בהרצת תוכנת DEBUG 69
תקציר פקודות DEBUG 70
תרגילים 72

פרק 5: לולאות 75
לולאה בתוכנית 75
הפקודה LOOP 78
דוגמאות לביצוע לולאות 78
תרגיל 1: תוכנית שרצה כראוי 78
תרגיל 2: איך בודקים תוכנית שגויה 80
תרגיל 3: הצבת נתונים בבלוק (קטע) זיכרון 82
תרגיל 4: העתקת בלוקים 83
תרגיל 5: ספירת תאי זיכרון המתאימים לקריטריון 84
תרגיל 6: בדיקת ערכי תאים 85
תרגיל 7: בדיקת מיון בלוק נתונים 86
תרגיל 8: סיכום ערכים בזיכרון 88
תרגיל 9: הזזת ערכים בזיכרון 89
תרגיל 10: מציאת הערך הגבוה ביותר 90
תרגיל 11: חיפוש כתובת על פי קריטריון 92
תרגיל 12: בדיקת סכום ערכים בבלוק נתונים 93

תרגיל 13: חיפוש מספר בזיכרון 95
תרגיל 14: חיפוש ערכים על פי קריטריון 96
תרגיל 15: בדיקת תנאים בזיכרון 97
תרגיל 16: בדיקת תנאים בזיכרון 98
תרגיל 17: השוואת בלוקים 99
תרגיל 18: השוואה והצבה 100
תרגיל 19: חיפוש רצף של נתונים בזיכרון 101
תרגיל 20: בדיקת תחום ערכים בזיכרון 102
תרגילים 103

פרק 6: פסיקות 105

מה זו פסיקה וכיצד מפעילים אותה? 105
פסיקה לסיום תוכנית ויציאה למערכת ההפעלה 106
פסיקה להצגת הודעה על המסך 108
פסיקה לקליטת מקש מהמקלדת 110
פסיקה להצגת תו בודד על המסך 113
פסיקה הבודקת אם הוקש מקש 119
קליטת מקשים מיוחדים 120
פסיקה לקביעת מיקום הסמן במסך 122
פסיקה לקליטת מחרוזת מהמקלדת 123
פסיקה לאתחול המחשב (Reset) 125
ניפוי שגיאות בתוכנית הכוללת פסיקות 126
דוגמה 1: הצגת הודעה 126
דוגמה 2: קליטת מקש והצגתו 128
סיכום עיקרי הפסיקות 129
תרגילים 129

פרק 7: כתובות זיכרון 131

שיטת הכתובות במחשב האישי 131
פנייה לכתובת פיסית באמצעות תוכנת ניפוי (כדוגמת DEBUG) 134
תרגילים בכתובות פיסיות (באמצעות DEBUG) 135
פנייה לכתובות פיסיות באמצעות תוכנית 136
הארות והערות 138
תרגילים בכתובות פיסיות 139
כתיבת נתונים בזיכרון 139
תרגילים 140

פרק 8: מושגים 141

פקודות והנחיות 141
סגמנטים - מקטעי זיכרון 141
תרגילים 142

הדגלים לסוגיהם 143
דגל האפס.. 144
תרגילים.. 145
דגל הסימן 145
שיטת המשלים ל-2 146
הסבת מספרים מבסיס הקסדצימלי לבסיס בינארי..................................... 147
תרגילים.. 149
הסבת מספר בינארי לייצוג הקסדצימלי 149
תרגילים.. 150
דגל הזכור (נשא).. 150
דגל הגלישה 152
הצגת הדגלים על ידי DEBUG... 153
תרגילים ופתרונות לדוגמה 154
תרגילים.. 157
סיכום פקודות הקפיצה המותנית הנוספות... 158

פרק 10: עבודה בסיביות... 159

הפקודה MOV עם המאפיינים Byte PTR ו- Word PTR..................................... 159
הפקודה AND... 160
כפל לוגי של סיביות 160
מיסוך באמצעות הפקודה AND 162
תרגול הפקודות 165
תרגילים.. 165
הפקודה TEST.. 166
תוכניות דוגמה והרצתן ב-DEBUG.. 167
תרגילים.. 170
הפקודה OR 170
תרגילים לדוגמה.. 171
תרגילים.. 172
הפקודה NOT... 172
הפקודה NEG... 173
הפקודה XOR... 173
דוגמאות לשימוש בפקודה.. 174
הפקודה SHR 175
חילוק על ידי פעולת SHR 176
דוגמאות לפקודת ההזזה... 178
הרצת התוכנית ב-DEBUG.. 178
פתרון בדרך מהירה יותר... 179
בדיקת התוכנית ב-DEBUG... 180
הפקודה SHL... 182
הפקודה ROR... 183
הפקודה ROL 185

תרגילים ופתרונות לדוגמה 186
בידוד סיביות 186
חיפוש סיבית בתא זיכרון 187
החלפה בין ערכים של שתי סיביות 189
חיפוש נתון בזיכרון 190
הצגת ייצוג בינארי של מקשים 191
קליטת ספרות הקסדצימליות וחישוב הסכום 194
חישוב ממוצע והצגתו 196
מיון של קבוצת מספרים 197
קליטת משפט והצגת המילים 198
תרגילים בנושא "החברים של..." 199
דוגמאות תרגול ל"חברים של..." 201
תרגילים 202
תרגילים כלליים 202
תרגילים בנושא "החברים של AL" 203

פרק 11: משתנים ומערכים 205
מה הם המשתנים? 205
הגדרת המשתנים בתוכנית 206
דוגמאות להגדרת משתנים בתוך מקטע התוכנית 206
דוגמאות להגדרת משתנים במקטע נתונים 208
כללי שימוש במשתנים 210
המשמעות של הגדרת משתנים בתוכנית 211
תרגילים ופתרונות 212
מערכים 217
הגדרת המערך ואיבריו 217
תוכניות דוגמה למערכים 218
תרגילים 223

פרק 12: המחסנית 227
הגדרת המחסנית 228
הפקודה PUSH 228
הפקודה POP 229
תוכניות לדוגמה 229
איך פועלת המחסנית 233
הכנסה והוצאה של נתונים מהמחסנית 235
בדיקת המחסנית באמצעות תוכנית ניפוי 237
תרגילים 238

פרק 13: פרוצדורות 241
הגדרת הפרוצדורה 241
מנגנון ביצוע הפרוצדורה 244
העברת נתונים אל הפרוצדורה ומהפרוצדורה 247
העברת נתונים באמצעות אוגרים 247
העברת נתונים באמצעות משתנים 248
העברת נתונים באמצעות מחסנית 249
פרוצדורה קרובה ופרוצדורה רחוקה 251
מתי נצטרך להגדיר פרוצדורה רחוקה 251
כיצד מגדירים פרוצדורה רחוקה 251
מה ההבדל במנגנון הקריאה והחזרה מהפרוצדורה 252
תיעוד הפרוצדורה 252
תרגילים 255

פרק 14: קבועים (Constants) 257
הקבועים ותפקידם 257

פרק 15: אסמבלי מותנה (Conditional Assembly) 261
מתי משתמשים באסמבלי מותנה 262
הוספת קטעי קוד לניפוי שגיאות 262
התאמות ייחודיות למשתמש 263
הנחיות נוספות של אסמבלי מותנה 263

פרק 16: מאקרו (Macro) 267
מהו מאקרו? 267
מאקרו עם ארגומנט 269
שילוב מאקרו עם אסמבלי מותנה 270
מאקרו לשכפול קוד REPT 271
מאקרו לשכפול קוד על פי רשימה - IRP 273
מאקרו לשכפול קוד על פי רשימה - IRC 274
תרגילים 275

פרק 17: פקודות מחרוזת 277
הפקודה LODSB (LOaD String Byte) 277
הפקודה STOSB (STOre String Byte) 278
הפקודה LODSW (LOaD String Word) 279
הפקודה STOSW (STOre String Word) 279
הפקודה REP (REPeat) 280
הפקודה CMPSB (CoMPare String Byte) 282
הפקודה REPE (REPeat Equal) 282
הפקודה CMPSW (CoMPare String Word) 284
הפקודה SCASB (SCAn String Byte) 284
הפקודה REPNE (REPeatNot Equal) 285

הפקודה SCASW (SCAn String Word)..286
הפקודות STD ו- CLD..286
דוגמה מסכמת לשימוש בפקודות מחרוזת..287
תרגילים..288

פרק 18: פקודות נוספות..289
הפקודה MUL..289
הפקודה DIV..291
כפל וחילוק מספרים מסומנים - IDIV, IMUL..293
הפקודה CBW..293
הפקודה CWD..293
הפקודה DAA..294
הפקודה DAS..294
פקודות לשמירה ואיחזור ערך אוגר הדגלים..294
פקודות קלט-פלט : IN ו-OUT..295
תרגילים..296

פרק 19: אסמבלי ושפת C..297
שפת אסמבלי לעומת שפות עיליות..297
שילוב קוד אסמבלי בפקודות שפת C..298
שילוב אסמבלי עם משתני שפת C - אסמבלי מוכלל..298
קישור אסמבלי לשפת C..301
קישור..302
העברת פרמטרים משפת אסמבלי לשפת C ולהיפך..302
הדגמת הקישור..304
הערות חשובות..306

אינדקס..307

הקדמה

שפת התכנות אסמבלי - Assembly Language - למחשבים אישיים נלמדת במסגרות לימודים רבות, וביניהן: בתי ספר מקצועיים ומכללות שבהם לומדים את מקצועות האלקטרוניקה, מכשור ובקרה (כיתות י״א עד י״ג); מגמות טכ״מ; הפקולטות למתמטיקה ומחשבים באוניברסיטאות; הפקולטות למהנדסי אלקטרוניקה ובקרה; מכללות ללימודי מחשבים ועוד. רבים משתמשים בשפת אסמבלי לכתיבת שגרות ותוכניות כחלק מתוכנות עיבוד גרפיות, מולטימדיה, תקשורת ועוד.

הספר צמח מתוך ניסיון עבודה בהוראת המקצוע לתלמידים ולסטודנטים, ומתוך ניסיון מעשי. הוא מכיל הסברים מפורטים של פקודות שפת אסמבלי, דוגמאות תרגילים עם פתרונות מפורטים, ולעיתים יותר מפתרון אחד לתרגיל, שיטות לניפוי ותיקון שגיאות ועוד. המחבר שם דגש על פישוט הצגת הנושאים ונותן דוגמאות ותרשימים להמחשת ההסבר. התוכניות כתובות לפי כללי תכנות מתקדמים ובכל מקרה שהיה צורך, הודגמו טעויות אופייניות ודרכים למניעתן.

לכל הלומדים והמורים מוקדש ספר זה.

מהדורה זו הנוכחית מתבססת על רב המכר ״שפת אסמבלי למחשב האישי״, שיצא לאור בסוף שנת 1993.

הנחיות לקורא

- הספר נכתב לפי שיטת "קליפות", על פיה החשיפה לחומר הנלמד הינה הדרגתית. בתחילה ניתן הסבר קצר "כיצד לבצע" פעולה כלשהי, ורק מאוחר יותר ניתן פירוט "מדוע לבצע כך".

- במקרים אחרים מובא הסבר חלקי של הנושא, ורק לאחר שתלמד חומר נוסף, אנו חוזרים ומלמדים את הפרטים בהרחבה. שיטה זו נועדה להקל על הבנת החומר.

- קרא בעיון את כל דוגמאות התרגילים ופתרונותיהם. הדבר יסייע לך להבין את החומר התיאורטי ויעזור לך בעת פתרון תרגילים מתקדמים. שים לב שלבעיה כלשהי עשויים להיות מספר פתרונות. בדרך כלל, הוצג פתרון אחד ובמקרים אחרים שניים, או שלושה פתרונות והודגשו הדרכים השונות לפתרון.

- בסוף כל פרק ניתנו תרגילים רבים לצורך חזרה והתנסות בתכנות. מומלץ מאוד לפתור אותם, ואם מתקשים בתרגיל כלשהו, אפשר לפנות לאחת הדוגמאות או להסברים ולחזור לפתרון התרגיל.

  יתכן שבמקרים מסוימים יהיו לך שאלות אודות החומר, או תחושה שטרם הבנת את הדברים בצורה מלאה. רשום שאלה זו בצד, ונסה לענות עליה **בעצמך** מאוחר יותר, כאשר יובהרו נושאים נוספים. זכור שחלק מהלימוד של שפה הינה באמצעות "חקירתה" וניסיון "לפענחה".

- מהדרי אסמבלי שונים עלולים להפיק הודעות שגיאה כאשר תנסה להדר חלק מהתוכניות שבספר זה. במקרה כזה, עליך לנתח את הודעת השגיאה ולתקן בהתאם.

- הספר נכתב בלשון זכר לשם פישוט הכתיבה, אך יש להבין זאת כפנייה גם אליך התלמידה, או הסטודנטית.

אני מאחל לכם הלומדים והמורים שימוש מועיל בספר.

המחבר

1

מושגי יסוד

המיקרומעבד

המיקרומעבד (מיקרופרוססור - **microprocessor**) מהווה את לב המחשב. במחשבים תואמי יבמ (PC), הכוונה היא ל**רכיב מוכלל** (**chip** - "ג'יוקי") המיוצר על ידי חברת Intel ונקרא מעבד פנטיום (80586, פנטיום III וכדומה). רכיבים ישנים יותר נקראים: 8086, 80286, 80386, 80486.

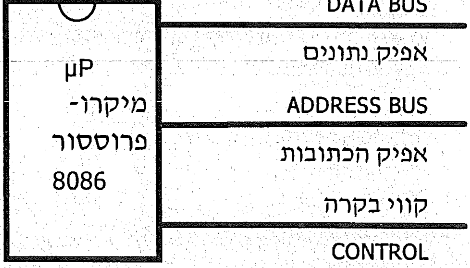
המיקרומעבד, או **מעבד** בקיצור, מפענח ומבצע תוכניות הכתובות בשפת מכונה בלבד (נסביר זאת בהמשך). הפקודות המרכיבות את התוכנית חייבות להיות חלק מאוסף הפקודות של הרכיב. לכל משפחת מעבדים יש אוסף פקודות מיוחד להם. על כן, פקודות המעבד של תואמי יבמ שונות לחלוטין מאלו של מחשבי מקינטוש למשל.

כל רכיב במשפחת 80X86 (ה-X מייצג ספרה המציינת את המשפחה של המעבדים) "מבין" את הפקודות של רכיב מיושן יותר. זוהי **תאימות** (compatibility), בשפת אנשי המקצוע. מפתחי הרכיבים נוהגים כך, כדי לאפשר המשכיות בעבודה על תוכנות ישנות, כאשר עוברים למחשבים מתקדמים יותר.

מתכנת **בשפת אסמבלי** (assembly language) פונה **ישירות** למעבד, ולכן עליו להכיר את רכיביו השונים, את המבנה שלהם, את תפקידיהם ואת אופן פעולתם. עם זאת, הוא אינו חייב להכיר את המבנה הטכנולוגי או את המבנה החשמלי של כל רכיב.

בספר זה נתייחס לאוסף הפקודות שמוכר על ידי כל משפחת המעבדים 80X86. לצורך הפשטות, נבחן את המעבד הבסיסי יותר - 8086.

תיאור סכימטי חיצוני של מעבד



כפי שנראה בתרשים, הרכיב כולל **אפיקים וקווי בקרה**.

אפיק (bus) הוא צירוף של מספר קווים המשמשים לקישור בין רכיבים שונים במערכת. מן המעבד יוצאים שני סוגי אפיק:

- **אפיק הכתובת** (address bus): באמצעותו קובע המעבד לאיזה רכיב או תא זיכרון הוא רוצה לפנות.
- **אפיק הנתונים** (data bus): האפיק הינו דו-כיווני. המעבד שולח בו נתונים (למשל, תו להצגה על גבי המסך), או מקבל באמצעותו נתון (כמו למשל, תו מלוח המקשים). בקווי הנתונים משתמש המעבד גם לקבלת הפקודות בשפת מכונה.

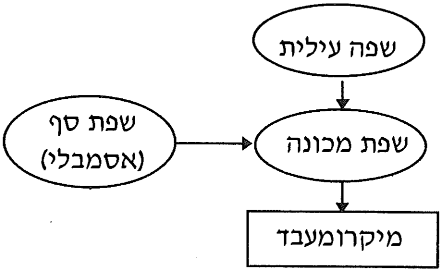
קווי הבקרה (control lines) משמשים לניהול מערכת המחשב. הם כוללים שתי קבוצות. קבוצה אחת של קווי בקרה פועלת על המעבד (כמו למשל, ביצוע reset), וקבוצה שנייה מאפשרת למעבד לפקח על רכיבים אחרים במערכת המחשב (על הזיכרון, למשל).

המעבד מכיל מספר רכיבים:

1. **יחידה אריתמטית-לוגית - ALU** (Arithmetic-Logic Unit), המבצעת את כל הפעולות החשבוניות (חיבור, כפל וכדומה) והלוגיות (and, xor, הזזת סיביות וכדומה).
2. **אוגר הדגלים** (flags register): זהו רכיב השומר נתונים של 9 "דגלים". כל דגל מיוצג על ידי סיבית שערכה "1" או "0", והיא מתארת מאפיין מסוים בפעולה האחרונה שבוצעה על ידי ALU. לדוגמה, במקרה שהתוצאה של פעולת חישוב מסוימת הינה 0, "דגל האפס" יהיה "1" (אחרת ערכו "0").
3. **אוגרים** (registers): במעבד ישנם, פרט לאוגר הדגלים, עוד 13 אוגרים. כל אחד מהם יכול לשמור נתון בגודל 16 סיביות. באוגרים אלה אנו משתמשים כדי לבצע את מערכת ההוראות של המעבד.
4. **יחידות בקרה** (control units): המעבד מכיל יחידות בקרה שונות, ביניהן:
   - **מפענח הפקודות** (command decoder), המתרגם את הפקודה בשפת מכונה לסדרת אותות חשמליים לצורך הביצוע שלה בחומרה (המעגלים האלקטרוניים ברכיבי המחשב).
   - **תור ההוראות** (instruction queue), השומר "בתור" כמה מהפקודות שעומדות להתבצע על ידי המעבד.
   - **חוצץ כתובות** (address buffer), שמחשב ושולח את הכתובת הפיסית המבוקשת, כדי לפנות לרכיב הדרוש.

שפות תכנות

בעולם המחשוב משתמשים בשפות תכנות רבות, אותן ניתן לסווג לשלוש רמות:



1. **שפת מכונה** (machine language): שפת מכונה הינה השפה **היחידה** שהמעבד מכיר. השפה בנויה מרשימת פקודות, המורכבות משילוב של סיביות 1 ו-0.

   שפת מכונה קשה מאוד לתכנות, להבנה ולתחזוקה. אם נתקלת פעם, תוך כדי עבודתך על המחשב, בתווים שונים המוצגים על המסך בליווי צפצופים - ראית את שפת המחשב. אפשר לראות זאת גם בדרכים אחרות, כפי שנלמד בהמשך.

2. **שפת סף** (אסמבלי, assembly): היא השפה הקרובה ביותר לשפת מכונה. כל פקודה בשפת מכונה מיוצגת על ידי פקודה **אחת** בשפת אסמבלי. ההבדל ביניהן הוא בכך, שהפקודות בשפת אסמבלי מובנות ואינן קשות לתכנות בהשוואה לשפת מכונה. עם זאת, התכנות באסמבלי מורכב יותר מתכנות בשפה עילית.

3. **שפה עילית** (high level language): שפה זו קלה ונוחה לתכנות. היא משחררת את המתכנת מעיסוק בנבכי המחשב ומאפשרת לו לבצע פעולות מורכבות באמצעות פקודות בודדות.

   כל פקודה בשפה עילית מתורגמת על ידי **מהדר** (compiler) או **מפרש** (interpreter) לתוכנית ערוכה **בשפת מכונה**, כדי שהמעבד יבין ויבצע אותה. שפות עיליות לדוגמה: פסקל, קובול, C, Visual Basic, Visual C++.

מדוע, אם כן, משתמשים בשפת אסמבלי? הסיבה העיקרית לשימוש בשפה הינה המהירות שבה רצה התוכנית הנכתבת באסמבלי. כך לדוגמה, אילו לא נכתבה התוכנית המטפלת בכספומט בשפת אסמבלי, היה הלקוח ממתין מספר דקות עד לקבלת מבוקשו!

סיבות נוספות להעדפת שפת אסמבלי: באמצעות השפה ניתן לטפל ב"קרביים" של המחשב וניתן ליצור תוכניות מורכבות קצרות יותר מאשר בשפה עילית.

למרות שהשפה נראית למתכנת המתחיל כלא-ברורה ולא-שימושית, ניתן לכתוב באמצעותה משחקי מחשב, לומדות ועוד. תוכנות רבות כתובות באסמבלי, או בעיקר באסמבלי. ניתן גם לשלב קטעי תוכנית אסמבלי בתוך תוכנות הכתובות בשפה עילית. דוגמאות לתוכנות הכתובות באסמבלי: מערכת ההפעלה DOS, תוכנת טורבו-פסקל, חלקים ממעבדי תמלילים ועוד.

נציג לדוגמה תוכנית זהה הכתובה בשפות שונות. כלומר, שלוש התוכניות מבצעות את אותה הפעולה: הן מציגות את המשפט "שלום לכולם" על המסך. התוכנית הראשונה כתובה בשפה עילית (פסקל), השנייה כתובה בשפת אסמבלי, והאחרונה - בשפת מכונה. מכיון שהמחשב מופעל על ידי פקודות מכונה בלבד, "מתורגמות" התוכניות בשפות פסקל ואסמבלי לשפת מכונה. שים לב, שכל אחת מהן יוצרת "תוכנית מכונה" בגודל שונה.

❖ תוכנית בשפת פסקל

```
program dugma;
begin
        write('שלום לכולם');
end.
```

התוכנית תורגמה לקובץ הרצה (COM) שגודלו 11453 בתים.

❖ תוכנית בשפת אסמבלי

```
code segment
        assume cs:code,ds:code
main:   mov  ax,code
        mov  ds,ax
        mov  dx,offset mess
        mov  ah,9
        int  21h
        mov  ax,4c00h
        int  21h
mess  db 'שלום לכולם$'
code ends
end main
```

התוכנית תורגמה לקובץ הרצה (EXE), שגודלו 540 בתים בלבד!

❖ תוכנית בשפת מכונה

בתוכנית זו מייצגת כל שורה פקודת מכונה אחת. במקום הצגה של סיביות בודדות, בחרנו להשתמש בייצוג הקסדצימלי. התוכנית בשפת מכונה צריכה "לחפוף" לתוכנית בשפת אסמבלי. עם זאת, אין זה נכון לגמרי, כפי שנראה בהמשך, ולכן אין שוויון במספר השורות.

גודל הקובץ כולו: 540 בתים. מכיון שגודל הכותר לתוכנית EXE הוא 512 בתים, גודל התוכנית שלפנינו הוא 28 בתים.

```
B81611
8ED8
BA1100
B409
CD21
B8004C
CD21
8D8C858B
8C20
8D858C9924
```

יחידות אחסנה

הנתונים והפקודות מוחזקים במחשב ביחידות אחסנה שונות:

- **סיבית** (Bit) - זוהי יחידת המידע הקטנה ביותר במחשב. היא מייצגת ספרה בינארית שערכה "1" או "0".
- **בית** (Byte) - צירוף של 8 סיביות. לדוגמה: 00001010. הייצוג המקובל של צירופי הסיביות, סה"כ 256 אפשרויות, נקרא **קוד אסקי** (ASCII code). לרוב הצירופים יש ייצוג גרפי שניתן לתצוגה במדפסת ולהדפסה. קיבולת זיכרון המחשב וקיבולת יחידות האחסון נמדדת ביחידות של בתים. אנו נכנה אותם גם בשם תאי זיכרון.
- **מילה** (Word) - צירוף של 16 סיביות, או שני בתים.
- **מילה כפולה** (Dword) - צירוף של 32 סיביות, 4 בתים.

בסיסי ספירה

בשפת אסמבלי נשתמש בשלושה בסיסי ספירה, מתוך האפשרויות הרבות הקיימות.

- **בסיס דצימלי** (עשרוני): בסיס זה הינו היחיד המשמש אותנו בחיי היום-יום. ברם, מכיון שהוא אינו מוכר למחשב כלל (המכיר בערכי 0 ו-1 בלבד), נשתמש בבסיסים אחרים. בכל פעם שנשתמש בבסיס עשרוני, הוא יתורגם במחשב לבסיס בינארי.
- **בסיס בינארי** (בסיס 2): בסיס ספירה זה כולל שתי ספרות בלבד, 0 ו-1.
- **בסיס הקסדצימלי** (בסיס 16): בבסיס ספירה זה משתמשים ב-10 הספרות ובאותיות A עד F:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

השימוש בבסיס הקסדצימלי נוח למדי, מכיון שהוא מתאר בצורה ברורה יותר נתונים הכתובים בבסיס בינארי: **כל** 4 ספרות בינאריות מיוצגות בו על ידי ספרה הקסדצימלית אחת.

ייצוג מספרים

כאשר נכתוב מספר (ערך מספרי) בשפת אסמבלי, נסמן לידו את הבסיס שבו הוא מיוצג. נעשה זאת על ידי כתיבה של אחת מהאותיות הבאות מימין למספר. נוכל לכתוב אות קטנה או גדולה:

- B - מספר בינארי, לדוגמה: 11110010B.
- H - מספר הקסדצימלי, לדוגמה: 45H.
- D - או ללא ציון כלל - מספר דצימלי, לדוגמה: 29D או 29.

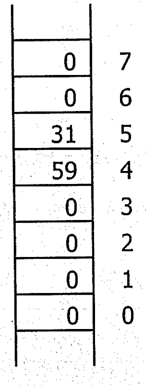
הזיכרון

הזיכרון הינו ההתקן של מערכת המחשב שבו נשמרים פקודות התוכנית המתבצעת במחשב והנתונים השוטפים המשמשים אותה כקלט, או מופקים ומיועדים לאחד מאמצעי הפלט. בזיכרון נמצאים למשל תוכנית המפיקה את רשימת התלמידים הממוינת, נתוני הציונים של 40 תלמידים, תוצאות חישובים שבוצעו על ידי המעבד, חלק מהדוח שטרם הוצג על גבי המסך וכדומה.

כאמור, הזיכרון משמש לאחסון תוכניות. למעשה, כל תוכנית חייבת להיות בזיכרון כדי שהמעבד יבצע אותה. המעבד מקבל את כתובת הזיכרון (המקום) שבו נמצאת התחלת התוכנית ומשם הוא מתחיל לבצע את הפקודות זו אחר זו, עד לסיום התוכנית.

הזיכרון בנוי מתאים, שכל אחד מהם יכול להכיל נתון בגודל **בית אחד בלבד** (8 סיביות). לכל אחד מהתאים יש כתובת: התא הראשון הינו בכתובת 0, לאחריו הכתובת הינה 1, וכך הלאה. למשל מחשב המכיל 64M תאי זיכרון, או 64M בתים, למעשה מכיל יותר מ-64 מיליון תאי זיכרון (K מסמל 1024 או 2^{10} בתים ו-M מסמל 2^{20} בתים).

תיאור הזיכרון



בשרטוט זה הצגנו חלק של זיכרון, שבו כל מלבן מייצג תא זיכרון אחד. לכל תא יש **כתובת** (address) המזהה את המיקום שלו באופן חד-משמעי ויש לו גם **ערך** (value), או **תוכן**. נסה לדמיין לעצמך בתי מגורים שלהם יש גם **כתובת** וגם **תכולה**, שהם שני דברים שונים.

בדוגמה שלפנינו התא בכתובת 4 מכיל את הערך 59. התא בכתובת 5 מכיל את הערך 31. שאר התאים מכילים את הערך 0.

כדי שנוכל לעבד את הנתונים, צריך להיות מנגנון לקריאה וכתיבה שלהם בתאי הזיכרון. **קריאת הנתון** או קליטת נתון אינה הורסת ואינה משנה את ערכו בתא. בכל קריאת נתון מהזיכרון, יש להגדיר את הכתובת שממנה יש לקרוא אותו.

אפשר **לכתוב נתון** חדש לתא, אולם במקרה זה הנתון החדש **ימחק** את הנתון הקודם שנמצא בתא ויחליף אותו. כדי לכתוב נתון לתא, יש לציין את כתובת התא ואת הנתון לכתיבה.

בדוגמה שלעיל, אם נכתוב את הנתון 77 בכתובת 5, נקבל את התוצאה הזו :

0	7
0	6
77	5
59	4
0	3
0	2
0	1
0	0

הנתונים נמצאים בזיכרון המחשב כל עוד המחשב פועל. כאשר מכבים את המחשב, או בעת הפסקת זרם החשמל, אובדים הנתונים. זוכר את הצורך לבצע גיבוי? זוהי פעולת כתיבה של הנתונים מהזיכרון על מדיה מגנטית, דיסק או דיסקט, כדי שיישמרו גם לאחר כיבוי המחשב.

האוגרים

קיימים במעבד **14 אוגרים** (registers), אשר כל אחד מהם הינו בגודל של מילה אחת (16 סיביות). כמו בזיכרון, כך גם האוגר מכיל נתון, אשר אפשר לקרוא אותו, או להחליפו באחר. כיבוי המחשב גורם לאובדן הנתונים שהיו באוגר.

האוגרים משמשים לשמירת נתונים, לביצוע פעולות חשבוניות ופעולות לוגיות, להעברת נתונים אל הזיכרון וממנו, ועוד.

לכל אוגר יש תפקיד, אולם בכמה מהאוגרים משתמשים למטרות דומות. ניתן לחלק את האוגרים ל-4 קבוצות, על פי התפקיד שלהם.

אוגרים כלליים (General Purpose registers)

	High	Low	
Accumulator	AH	AL	AX
Base	BH	BL	BX
count	CH	CL	CX
Data	DH	DL	DX

לרשותנו 4 אוגרים כלליים. נבחין שכל אחד מהאוגרים ניתן לחלוקה לשני חצאי אוגר. לכל מחצית אוגר יש סימון של אות זיהוי של האוגר ולידה האות H או L. החלק השמאלי מסומן באות H, כדי לציין High (החלק "הגבוה" של המספר) והחלק הימני מסומן באות L כדי לציין Low (הערך "הנמוך").

1. **אוגר AX** - אוגר זה מכונה גם צובר או אקומולטור (משתמשים בו לסיכום מספרים). החלק השמאלי נקרא AH (H לציון High) והחלק הימני נקרא AL (L עבור Low).

2. **אוגר BX - אוגר הבסיס.** החלק השמאלי נקרא BH והימני BL.

3. **אוגר CX - אוגר המונה.** החלק השמאלי נקרא CH והימני CL.

4. **אוגר DX - אוגר הנתונים.** החלק השמאלי נקרא DH והימני DL.

למעשה, על ידי חלוקה זו קיבלנו 12 אוגרים שונים: ניתן לפנות למשל לאוגר AX, השלם, או לכל אחד משני חצאיו: אוגר AL, או אוגר AH, וכן הלאה.

אוגרי המקטע (Segment registers)

CS	Code Segment
DS	Data Segment
ES	Extra Segment
SS	Stack Segment

5. **אוגר CS - אוגר מקטע הקוד.** מכיל את כתובת ההתחלה של התוכנית.

6. **אוגר DS - אוגר מקטע הנתונים.** מכיל את כתובת ההתחלה של מקטע הנתונים.

7. **אוגר ES - אוגר הנתונים נוספים.** מכיל את כתובת תחילת הנתונים הנוספים (אם ישנם).

8. **אוגר SS - אוגר המחסנית.** מכיל את כתובת תחילת המחסנית (יוסבר בהמשך).

אוגרים מצביעים (Pointer registers)

SI	Source Index
DI	Destination Index

9. **אוגר SI - אוגר מצביע מקור.** מצביע על כתובת תא זיכרון מבוקש.

10. **אוגר DI - אוגר מצביע יעד.** מצביע על כתובת תא זיכרון מבוקש.

BP	Base Pointer
SP	Stack Pointer

11. **אוגר BP - מצביע הבסיס.** משמש כמצביע על כתובת תאי הזיכרון במחסנית.

12. **אוגר SP - מצביע המחסנית.** מצביע על כתובת תא הזיכרון בקצה המחסנית.

IP	Instruction Pointer

13. **אוגר IP - מצביע פקודה.** זוהי הכתובת של ההוראה **הבאה** לביצוע.

אוגר הדגלים (Flags register)

F	Flags Register

14. **אוגר F** - **אוגר הדגלים**. מכיל 9 דגלים (סיביות) הנמצאים במצב אחד או אפס לוגי. הם מתארים מאפיינים מסוימים בפעולה האחרונה שבוצעה על ידי היחידה אריתמטית-לוגית (ALU) של המעבד.

הצגנו בקצרה את האוגרים ואת תפקידיהם. כמה מהתפקידים בוודאי אינם מובנים בשלב זה. אל דאגה, עוד נחזור ונסביר כל עניין בהרחבה. בשלב זה, יש בידינו די כלים, כדי לצאת לדרך ולכתוב תוכניות פשוטות. בהצלחה!

2

פקודות בסיסיות

כיצד כותבים פקודות בשפת אסמבלי

שפת אסמבלי כוללת עשרות רבות של פקודות. שמות הפקודות מרמזים על מהותן, ולכן ניתן לזכור אותן בנקל. בשל היותה שפה הניתנת לזכירה, שפת אסמבלי מכונה גם בשם mnemonic language.

חלק גדול מפקודות השפה נרשמות בתבנית הבאה:

פקודה	**אופרנד**	,	**אופרנד**	דוגמאות:
MOV	AX	,	56	
ADD	[SI]	,	DL	
CMP	CX	,	BX	

למרות שאינך מבין את הפקודות בשלב זה, תוכל להבחין בתבנית הקבועה שלהן: בשמאל (קרא תמיד משמאל לימין!) נרשמת **פקודה** (command), לאחריה אופרנד אשר יכול להיות אוגר, כתובת זיכרון ועוד. לאחר מכן, יש פסיק (,) המשמש להפרדה בין האופרנדים ואחריו מופיע אופרנד נוסף.

תבנית זו פועלת על פי הכלל הבא: האופרנד השמאלי, הקרוב לפקודה, הוא זה **שעליו** מבוצעת הפקודה. הוא נקרא גם **אופרנד היעד** (destination operand). האופרנד הימני הינו **אופרנד המקור** (source operand).

חלק מהפקודות כולל אופרנד אחד, המשמש כאופרנד היעד. פקודות אחרות אינן כוללות אופרנדים כלל, מכיון שהאופרנדים עבור פקודות אלו הם קבועים ואינם ניתנים לשינוי.

מימין לפקודה ניתן להוסיף **הערות** (remarks). כדי לעשות זאת, יש לרשום את סימן הנקודה-פסיק (;) ולאחריה הערה. דוגמה לפקודה הכוללת הערה:

```
MOV DL,44        ; הצבת המספר 44 באוגר DL
```

משמאל לפקודה ניתן להוסיף **תווית** (label), המאפשרת לקפוץ ממקום כלשהו בתוכנית אל המקום שבו מופיעה התווית. דוגמה לפקודה הכוללת תווית והערה:

```
GOOD: MOV BX,5588        ; מציב 5588 באוגר BX (שם התווית GOOD)
```

הפקודה MOV (Move)

הפקודה MOV גורמת ל**העתקה** (ולא העברה!) של נתון ממקום אחד למקום אחר בתוכנית. לפקודה זו מספר אפשרויות:

1. העתקת ערך ישירות לאוגר. זוהי הצבת נתון באוגר.
2. העתקת נתון מאוגר לאוגר.
3. העתקת נתון מאוגר לתא זיכרון.
4. העתקת נתון מתא זיכרון לאוגר.

העתקה ישירה לאוגר

פעולה זו נקראת גם בשם **הצבה באוגר**.

תבנית הפקודה: צמוד לפקודה MOV נרשם שם של אחד האוגרים ומימינו, לאחר הפסיק, נרשם המספר (הערך) שברצוננו להציב באוגר.

דוגמאות:

1. הצבת הערך 3456 (בבסיס הקסדצימלי) בתוך אוגר DX:

MOV DX,3456H

תמונת האוגר:

34	56	DX
DH	DL	

כל ספרה הקסדצימלית תופסת מקום של 4 סיביות. בכל אחד מחצאי האוגר מוצבות שתי ספרות, המהוות יחד 8 סיביות. הספרות בעלות הערכים הגבוהים יותר, Most Significant Digits - MSD, נשמרות תמיד בחצי האוגר השמאלי, הגבוה (H).

2. הצבת המספר 32 באוגר AL. האות **H** מציינת שהמספר הינו בבסיס הקסדצימלי.

MOV AL,32H

הפקודה אינה משנה את חצי האוגר AH! אם למשל אוגר AX מכיל לפני ביצוע הפקודה את הערך 3728H:

37	28	AX
AH	AL	

לאחר ביצוע הפקודה הזו, נקבל את התמונה הבאה:

37	32	AX
AH	AL	

3. הצבת המספר 234 בתוך אוגר AX.

MOV AX,234H

תמונת האוגר לאחר הפקודה:

AH	AL	
02	34	AX

כלומר, האוגר מכיל 4 ספרות הקסדצימליות, כאשר הספרה 0 ממלאת את הספרה החסרה (המספר 234H המיוצג ב-4 ספרות הוא 0234H).

4. הצבת המספר הבינארי 11110000 באוגר AX.

MOV AX,11110000B

התוצאה:

AH	AL	
00	F0	AX

העתקת נתון מאוגר לאוגר

תבנית הפקודה: צמוד לפקודה MOV נרשם אוגר היעד (שאליו ברצוננו להעתיק את הנתון) ומימינו, לאחר הפסיק, נרשם אוגר המקור שממנו נעתיק.

הפקודה תעתיק את הנתון מאוגר המקור לאוגר היעד. ערך אוגר המקור לא ישתנה. ערכו הקודם של אוגר היעד יימחק, והנתון החדש יתפוס את מקומו.

דוגמאות:

1. העתקה של הנתון שנמצא בחצי האוגר DL **אל** חצי האוגר AH.

MOV AH,DL

נניח למשל, שלפני הפקודה, האוגרים הכילו את הערכים הבאים:

AH	AL		DH	DL	
32	15	AX	64	99	DX

לאחר ביצוע הפקודה, הם יכילו את הערכים האלה:

AH	AL		DH	DL	
99	15	AX	64	99	DX

2. העתקה של הנתון שבאוגר BX **אל** אוגר AX.

MOV AX,BX

אם אוגר BX מכיל את הערך 78H, האוגרים AX ו-BX יכילו לאחר ביצוע הפקודה את הערכים הבאים:

AH	AL		BH	BL	
00	78	AX	00	78	BX

"כלל ברזל" 1: התאמה בגודל האופרנדים

עבור **כל** הפקודות בשפת אסמבלי, יש לשמור על **כלל ההתאמה** בגודל האופרנדים. המשמעות היא: גודלו של אופרנד היעד (זה שמקבל את התוצאה), **חייב** להיות זהה לגודל אופרנד המקור!

כדי להבהיר נקודה זו בפקודה MOV, נראה מספר דוגמאות של **פקודות שגויות**:

1. אסור להכניס לחצי אוגר (שהינו בן 8 סיביות) מספר גדול מ-2 ספרות.

MOV AL,569H

2. אסור להכניס לאוגר מספר גדול מ-4 ספרות.

MOV AX,87321H

3. חייבת להיות התאמה בגודל האוגרים: לא ניתן להעתיק נתון בן 16 סיביות לחצי אוגר בן 8 סיביות.

MOV AL,DX

4. לכאורה, ניתן להעתיק נתון בן 8 סיביות לאוגר של 16 סיביות, אך פעולה זו אסורה.

MOV BX,CL

הערה: מקרים חריגים יתוארו בהמשך.

העתקת נתון מאוגר לתא זיכרון

ניתן לכתוב נתונים לזיכרון, או לקרוא נתונים הכתובים בזיכרון. בכל פעם שרוצים לסמן תא זיכרון, משתמשים בסימן [ ] (סוגריים מרובעים). יתר על כן, כל פעם שיופיע סימן זה, נדע שהכוונה לתא זיכרון. בתוך הסוגריים המרובעים רושמים את הכתובת של תא הזיכרון שאליו רוצים לפנות.

למשל, כדי להכניס את הנתון 88 לתוך תא זיכרון שכתובתו 1000, ניתן היה לכתוב את הפקודה הבאה:

MOV [1000],88

כלומר: העתק את המספר 88 לתוך תא זיכרון (בגלל הסוגריים המרובעים), שכתובתו היא 1000 (לפי הערך בתוך הסוגריים).

כתובת		
1003		
1002		
1001		
1000	<	88

אולם, מפתחי החומרה ומפתחי שפת האסמבלי קבעו מספר מגבלות בקשר לכך. המגבלות של הטיפול בזיכרון ישימות לגבי **כל** פקודות השפה, ועל כן נציג את "כלל הברזל" השני.

"כלל ברזל" 2: פנייה לזיכרון

1. כדי לציין כתובת של תא זיכרון בתוך הסוגריים המרובעים, אסור לכתוב מספר, אלא רק את אחד מהאוגרים DI, SI, BX (במקרים מסוימים מותר השימוש באוגר נוסף).
דוגמאות:
   - אם אוגר BX מכיל את הערך 700H ואנו כותבים [BX], אנו פונים לתא זיכרון בכתובת 700H.
   - אם נתון שאוגר SI מכיל את הערך 1022H ונרשום [SI], אנו פונים לתא זיכרון שכתובתו 1022H.

   קיימות מספר אפשרויות לשימוש באוגרים האלה:
   - שימוש באחד מהאוגרים בלבד, לדוגמה: [BX],[SI] או [DI].
   - שימוש באוגר וערך, לדוגמה: [SI+1],[DI-8], [BX+120H].
   - שימוש באוגר הבסיס BX ואוגר אינדקס (מצביע) SI או DI. האפשרויות הן: [BX+SI], או [BX+DI] בלבד.
   - שימוש בבסיס, אינדקס וערך, לדוגמה: [BX-DI+88H].
2. אין לרשום פקודה הכוללת תא זיכרון ונתון (מספר). כדי לבצע פעולה בין תא זיכרון למספר (למשל הצבת מספר בתא), יש לבצע מספר דברים:
   - להציב את הנתון באוגר בגודל מתאים.
   - לבצע פעולה בין תא הזיכרון לבין האוגר.

   לדוגמה: MOV [BX],8 - שגוי.
3. אסור לבצע פעולות ישירות בין שני תאי זיכרון. כלומר, אסור לכתוב פקודה שבה שני האופרנדים הם תאי זיכרון. זו פעולה שגויה.

   לדוגמה: MOV [SI],[DI] - שגוי.
4. כאשר רוצים לפעול על תא זיכרון אחד, יש להשתמש באוגר בגודל בית, על פי הכלל שצריכה להיות התאמה בגודל האופרנדים. כל תא מכיל נתון בגודל בית.

כאשר מבצעים פעולה על שני תאים צמודים, יש להשתמש באוגר בגודל מילה. דוגמאות לכך ניתן בהמשך.

הערה: בהמשך תוסבר השיטה שלפיה ניתן יהיה לבצע פעולה ישירה בין תאי זיכרון.

כדי להכניס לתא בכתובת 1000 בזיכרון את המספר 88, יש להשתמש בפקודות האלו:

```
MOV BX,1000
MOV AL,88
MOV [BX],AL
```

כדי להמחיש את הטיפול בזיכרון, תמצא להלן מספר דוגמאות חוקיות ולא חוקיות:

לא חוקי:	אין לכתוב מספר בתוך הסוגריים ואסור לכתוב את הנתון ישירות.	MOV [300],6
לא חוקי:	אין להשתמש באוגר AX כמצביע לכתובת.	MOV [AX],DL
חוקי:	״ ״	MOV [BX],CL
חוקי:	״ ״	MOV [SI],AL
לא חוקי:	אין להשתמש באותה פקודה באותו אוגר (BL הינו חצי של האוגר BX).	MOV [BX],BL
חוקי:	״ ״	MOV [DI],CH
לא חוקי:	אסור להעתיק באותה פקודה נתון מתא זיכרון אחד לתא אחר.	MOV [BX],[DI]
חוקי:	״ ״	MOV [SI],DH
חוקי:	הפקודה חוקית, כי ניתן להשתמש באוגר בגודל מילה, רק כאשר רוצים להציב נתון בגודל מילה בשני תאי זיכרון **סמוכים**. הפקודה אינה נכונה, אם המטרה היא להציב נתון בתא זיכרון אחד!	MOV [DI],AX

העתקת נתון מהזיכרון לאוגר

כדי לבצע העתקת ערך של תא זיכרון לאוגר, צריך לכתוב את הפקודה, כך שהאופרנד השמאלי (אופרנד היעד) יהיה האוגר והאופרנד הימני (המקור) יהיה תא הזיכרון הרצוי. לדוגמה, כדי להעתיק את ערך תא זיכרון שכתובתו 819H לאוגר DH, נכתוב:

```
MOV SI,819H
MOV DH,[SI]
```

דוגמאות לתוכניות ופתרונות

1. **אילו פקודות אינן חוקיות, ומדוע?**

`MOV DL,[SI]` — הפקודה **נכונה** מבחינה תחבירית. היא מעתיקה אל אוגר DL את ערך תא הזיכרון שכתובתו מצוינת על ידי אוגר SI.

`MOV [DI],[SI]` — **טעות**: אין להעתיק נתון ישירות מתא זיכרון אחד לתא זיכרון אחר. כדי לבצע העתקה של נתון מתא לתא, יש להשתמש באוגר מתווך בגודל בית אחד.

הנה פתרון לדוגמה:

```
MOV DL,[SI]
MOV [DI],DL
```

`MOV [AX],BL` — **טעות**: אין לציין את כתובת התא על ידי אוגר AX!

`MOV [BX+1],DL` — הפקודה **נכונה**: היא מבצעת העתקת נתון מאוגר אל תא זיכרון שכתובתו גדולה ב-1 מהערך של BX.

הפקודה **נכונה**: היא מעתיקה את ערך האוגר CH אל תא זיכרון שכתובתו הינה (ערך BX + ערך SI).	MOV [BX+SI],CH
טעות: אין להשתמש באוגר AX כמציין כתובת תא זיכרון!	MOV [BX+AX],DL
טעות: אין להשתמש בשני האוגרים SI ו-DI **יחד**, כדי לציין כתובת של תא זיכרון. ניתן להשתמש ב-SI בלבד, או ב-DI בלבד, או בשילוב של אוגר הבסיס BX עם אחד מהם.	MOV [SI+DI],BH
הפקודה **נכונה**. היא מעתיקה את ערך האוגר BL אל תא זיכרון הנמצא בכתובת DI-12H.	MOV [DI-12H],BL
הפקודה **נכונה**, אך שים לב: הפקודה מעתיקה את ערך האוגר CX (16 סיביות) **לשני** תאים עוקבים. מכיון שמשתמשים באוגר בגודל מילה, בעוד שגודל כל תא הינו בית אחד (חצי מילה), הנתון יוכנס ל-2 תאי זיכרון צמודים.	MOV [SI+5],CX

2. כתוב תוכנית שמעתיקה תוכן תא זיכרון שכתובתו 2500H לאוגרים AL ו-BL.

פתרון א'	**פתרון ב'**	**פתרון ג'**
MOV BX,2500H	MOV SI,2500H	MOV DI,2500H
MOV AL,[BX]	MOV BL,[SI]	MOV AL,[DI]
MOV BL,AL	MOV AL,[SI]	MOV BL,AL

פתרונות אלה מדגימים אפשרויות שונות לפתרון. ננתח כעת טעויות שונות שנעשו בכתיבת הפקודות בפתרונות אלה. שים לב והימנע מכך בעת הכתיבה.

טעות א':

MOV AX,2500H	זו עדיין לא טעות, כי מותר להכניס מספר לאוגר AX.
MOV BL,[AX] MOV AL,BL	**טעות!** בתוך הסוגריים המרובעים לציון כתובת של תא זיכרון יכולים להופיע רק אוגרים מסוימים. אם אינך זוכר אילו אוגרים, דפדף אחורה, והשתדל לזכור זאת.

טעות ב':

MOV SI,2500H	בסדר, אבל...
MOV AX,[SI]	כאשר כותבים פקודה כזו כאן, זוהי **טעות**.
MOV BX,[SI]	בפנייה לתא זיכרון אחד צריך להשתמש באוגר בגודל 8 סיביות בלבד. כלומר, AL, DL וכדומה, ולא להשתמש באוגר AX או BX. בהמשך נראה שכאשר מבקשים לפנות ל-2 תאי זיכרון סמוכים, ניתן להשתמש באוגר בן 16 סיביות.

טעות ג':

MOV AL,[2500H] MOV BL,[2500H]	באסמבלי אין לרשום מספר לציון כתובת של תא זיכרון, פרט למקרים מיוחדים בלבד. נלמד זאת בהמשך.

3. כתוב תוכנית שמציבה את הערך ההקסדצימלי 47H בתא זיכרון שכתובתו 2400H:

פתרון א'

```
MOV SI,2400H
MOV DL,47H
MOV [SI],DL
```

פתרון ב'

```
MOV AH,47H
MOV BX,2400H
MOV [BX],AH
```

פתרון ג'

```
MOV CH,47H
MOV DI,2400H
MOV [DI],CH
```

טעות אופיינית:

```
MOV SI,2400H
MOV [SI],47H
```

אסור להכניס נתון ישירות לזיכרון. חובה להשתמש באוגר מתווך בגודל 8 סיביות (בית), למעט שיטת מיעון זיכרון מיוחדת שנלמד בפרק 10.

4. העתק את הנתון שנמצא בתא זיכרון שכתובתו 1000, לתא זיכרון שכתובתו 2000. לביצוע משימה זו יש לכתוב את הפקודות האלו:

הצבת 1000 באוגר SI	MOV SI,1000
הצבת 2000 באוגר DI	MOV DI,2000
העתקת הנתון מתא שכתובתו 2000, לאוגר AL	MOV AL,[SI]
העתקת הנתון מאוגר AL, לתא זיכרון 2000	MOV [DI],AL

זו רק אחת האפשרויות לפתרון. ניתן לפתור זאת גם בדרכים הבאות:

פתרון ב'

```
MOV BX,1000
MOV AL,[BX]
MOV BX,2000
MOV [BX],AL
```

פתרון ג'

```
MOV SI,1000
MOV CH,[SI]
MOV BX,2000
MOV [BX],CH
```

פתרון ד'

```
MOV DI,1000
MOV SI,2000
MOV DH,[DI]
MOV [SI],DH
```

דוגמה לפתרון שגוי:

	MOV BX,1000
	MOV SI,2000
עד לשורה זו (כולל), הפקודות נכונות.	MOV AL,[BX]
פקודה זו מיותרת לחלוטין:	MOV DL,[SI]
כאן מתברר, שאם נכניס את הנתון לאוגר DL, הרי שכאילו הכנסנו את הנתון לתא זיכרון 2000. זוהי טעות אופיינית, ויש להיזהר מכתיבה כזו.	MOV DL,AL

5. כתוב תוכנית שתחליף בין הנתון הנמצא בתא 700 והנתון בתא 800.

בסיום החלפת הנתונים בין שני תאי זיכרון: בתא 700 יהיה הנתון שהיה בתא 800, ובתא 800 יהיה הנתון שהיה בתא 700.

מציב 700 באוגר SI	MOV SI,700
מציב 800 באוגר DI	MOV DI,800
מעתיק לאוגר AL את הנתון מתא 700	MOV AL,[SI]
מעתיק לאוגר AH את הנתון מתא 800	MOV AH,[DI]
מעתיק לתא 700 את הנתון באוגר AH (שהיה בתא 800)	MOV [SI],AH
מעתיק לתא 800 את הנתון באוגר AL (שהיה בתא 700)	MOV [DI],AL

תרגילים

בתרגילים אלה נשתמש בפקודה MOV בלבד.

1. כתוב תוכנית באסמבלי שתציב את המספר 56 בתא זיכרון 499.
2. כתוב תוכנית המציבה את הנתון 66 בתאי זיכרון 400,300, 500.
3. כתוב תוכנית המחליפה בין הנתונים שבתאי זיכרון 300 ו-800.
4. כתוב תוכנית המציבה את הנתון הנמצא בתא שכתובתו 400, בתוך תאי זיכרון שכתובתם 1300,1200.
5. כתוב תוכנית המציבה:

0	בתא זיכרון 1000
1	בתא זיכרון 1001
2	בתא זיכרון 1002

6. כתוב תוכנית המעתיקה את הנתון מתא שכתובתו 800, לאוגרים אלה: DH, DL, CL, CH.
7. הכן טבלת מעקב ובדוק מה יהיה תוכן האוגרים לאחר ביצוע כל אחת מתוכניות אלו:

תוכנית א'	תוכנית ב'	תוכנית ג'
MOV AX,456	MOV AX,7	MOV DX,1122
MOV DH,AL	MOV BX,4321	MOV CX,3456
MOV DL,AH	MOV BL,BH	MOV SI,8765
MOV AH,AL	MOV CX,BX	MOV DI,8976
MOV DL,AH		MOV SI,DX

8. בתאי הזיכרון הבאים נמצאים הנתונים האלה:

בתא 200	-	22
בתא 201	-	33
בתא 202	-	44
בתא 800	-	15
בתא 801	-	16
בתא 802	-	17

מה יכילו תאי זיכרון אלה לאחר ביצוע התוכנית הזו:

```
MOV BX,200
MOV AL,[BX]
MOV SI,201
MOV DI,802
MOV DL,[SI]
MOV CL,[DI]
MOV BX,800
MOV SI,801
```

```
MOV [SI],AL
MOV [DI],DL
MOV [BX],CL
```

בדוק את התוכנית בעזרת טבלת מעקב.

הפקודה INC (Increment)

הפקודה מורה למעבד להגדיל ערך של אוגר, או ערך של משתנה ב-1 (1 בלבד!). כלומר, המעבד יוסיף את הערך 1 לערך שכבר נמצא באוגר או במשתנה לפני תחילת הפעולה.
להלן דוגמאות לפעולה על אוגר:

מוסיף 1 לתוכן אוגר DL:	(DL ← DL+1)	INC DL
מוסיף 1 לתוכן אוגר SI:	(SI ← SI+1)	INC SI
פקודה שאינה חוקית:		INC [SI]

הפקודה DEC (Decrement)

הפקודה מורה למעבד להפחית 1 (1 בלבד!) מהערך שנמצא באוגר, או במשתנה. כלומר, המעבד ייחסר את הערך 1 מהערך שנמצא באוגר לפני תחילת הפעולה. להלן מספר דוגמאות, אשר דומות לאלו של הפקודה INC ומתייחסות לפעולה על אוגר:

מפחית 1 מתוכן אוגר CX:	(CX ← CX -1)	DEC CX
מפחית 1 מתוכן אוגר SI:	(SI ← SI-1)	DEC SI
פקודה שאינה חוקית:		DEC [BX]

הפקודה ADD (Addition)

הפקודה מורה למעבד לבצע פעולת חיבור. הפקודה גם יכולה להורות למעבד לחבר אל תא זיכרון כלשהו ערך שנמצא באוגר אחר. לפניך מספר דוגמאות:

מוסיף 5 לתוכן האוגר CH	(CH ← CH+5)	ADD CH,5
מוסיף 67 לתוכן האוגר DI	(DI ← DI+67)	ADD DI,67
מוסיף את תוכן AX לתוכן BX	(BX ← BX+AX)	ADD BX,AX
מוסיף לאוגר DL את ערכו	(למעשה כופל ב-2)	ADD DL,DL
מוסיף לאוגר AL את ערך תא הזיכרון שכתובתו נתונה ב-BX		ADD AL,[BX]
מוסיף לתוכן תא הזיכרון שכתובתו נתונה ב-DI, את ערך האוגר CL		ADD [DI],CL

דוגמאות לפקודות שאינן חוקיות:

אסור להוסיף נתון ישירות לתא זיכרון (במעבד 8086 בלבד)! ADD [BX],4

אסור להוסיף ערך תא מסוים לערך של תא אחר באותה פקודה! ADD [SI],[DI]

אסור לבצע פעולה בין שני ערכים שאינם שווים בגודלם: ADD CX,DH

אין התאמה בגודל האופרנדים: ADD AL,590H

הפקודה SUB (Subtract)

הפקודה מורה לבצע פעולת חיסור. הנה מספר דוגמאות:

מחסר 7 מתוכן אוגר AL (AL ← AL-7) SUB AL,7

מחסר מערך תא הזיכרון שכתובתו נתונה באוגר BX, את ערך אוגר DL SUB [BX],DL

מחסר מערך אוגר DL את ערך DH SUB DL,DH

הפקודה NOP (No Operation)

פקודה זו אומרת "לא לבצע דבר". הפקודה שימושית כאשר מעוניינים לשתול פקודה כלשהי בין הפקודות מבלי שתעשה דבר (למרימי-הגבה שביניכם יוסבר השימוש בה מאוחר יותר).

הפקודה CMP (Compare)

ביצוע השוואה בין שני ערכים. השוואה נעשית על ידי חיסור של אופרנד המקור מאופרנד היעד, מבלי לשנות את אופרנד היעד. תוצאות ההשוואה יכולות להיות אחת מאלה: שווה, קטן או גדול. תוצאה זו תישמר באוגר הדגלים, כך שמייד לאחר ביצוע הפקודה, אפשר יהיה לבדוק זאת ולהפעיל את אחת מפקודות הקפיצה.

לפניך מספר דוגמאות:

השווה בין הערך שבאוגר AL לבין המספר 7 CMP AL,7

השווה בין הערך שבאוגר BX לבין הערך שבאוגר CX CMP BX,CX

השווה בין הערך שבתא הזיכרון שכתובתו נתונה באוגר SI, לבין הערך שבאוגר CL CMP [SI],CL

האיסורים בפקודה זו דומים לאלה שהכרנו בפקודה MOV:

אסור להשוות ישירות שני תאי זיכרון (במעבד 8086 בלבד)! CMP [SI],[DI]

אסור להשוות בין תא זיכרון לבין מספר: CMP [BX],4

אסור להשוות בין אוגרים שאינם בגודל זהה: CMP AL,BX

כדי להשוות בין נתון שנמצא בתא **יחיד**, יש להשתמש באוגר בגודל בית אחד בלבד כמתווך. לדוגמה, נשווה בין הערך בתא זיכרון שכתובתו 377H, לבין המספר 11H. הפקודות שיש לכתוב:

```
MOV DL,11H
MOV BX,377H
CMP [BX],DL
```

בהמשך נלמד אודות השימושים בפקודה זו.

הפקודה JMP (Jump)

פקודת קפיצה (ללא תנאים) ממקום בתוכנית שבו כתובה הפקודה למקום אחר בתוכנית. פקודה זו דומה לפקודה GOTO בשפת בייסיק או בשפת פסקל. לימין הפקודה יש לרשום תווית שאליה תתבצע הקפיצה. לפניך דוגמה לתוכנית המציגה שימוש בפקודה. אין כל משמעות לתוכנית זו, מטרתה להדגים את הפקודה בלבד:

```
        MOV AL,3
        MOV BL,88
        JMP POPYE      קפוץ לתווית ששמה popye
        MOV DX,22      שורה זו אינה מתבצעת
POPYE:  MOV DX,11      התוכנית קופצת לכאן
```

שים לב, שפקודת הקפיצה גרמה לדילוג "מעל" הפקודה MOV DX,22 אינה מתבצעת.

תווית (label)

התווית משמשת לנו "מראה מקום" של פקודה בתוכנית, כדי שנוכל לפנות אליה בפקודת הקפיצה. על תפקידים נוספים של התווית נלמד בהמשך. לכתיבת תווית יש מספר כללים:

1. ניתן להשתמש בכל אות אנגלית גדולה או קטנה, בספרות ובתווים מיוחדים אלה: $, @, _, ? ונקודה (.).
2. התווים של התווית צריכים להיות סמוכים, ללא רווח ביניהם. לדוגמה, אסור לכתוב למשל GOOD BYE. אם רוצים לכתוב שתי מילים נפרדות, צריך להשתמש במקף תחתון, למשל: GOOD_BYE (או GoodBye).
3. אין לבחור בשם תווית ששמה כשם של פקודה.
4. אין לבחור בשם תווית שמתחיל בספרה, או בנקודה.

פקודת קפיצה מותנית (Conditional Jump)

בשפת אסמבלי יש מספר פקודות קפיצה מותנות, אשר גורמות לקפיצה למקום אחר בתוכנית, רק אם מתקיים תנאי מבוקש כלשהו. נציג כמה מפקודות אלו כאן כדוגמה, אחרות נסביר בהמשך. התבנית של פקודות אלו זהה לזו של הפקודה JMP. כלומר, מימין לפקודה יש לרשום תווית המציינת את המיקום לקפיצה.

קל לזכור את הפקודות הללו, כאשר מבינים שהאותיות הינן קיצור של מילת התנאי:

G = Greater (גדול)	**Z** = Zero (אפס)	**N** = Not (לא)
L = Less (קטן)	**E** = Equal (שווה)	

לפניך רשימה **חלקית** של פקודות הקפיצה המותנות. פקודות נוספות נלמד בהמשך:

- קפוץ אם הערכים שווים: JE
- קפוץ אם הערכים אינם שווים: JNE
- קפוץ אם הערך שרשום משמאל גדול מהערך שרשום מימין: JG
- קפוץ אם הערך משמאל קטן מהערך מימין: JL
- קפוץ אם הערך השמאלי גדול או שווה לימני: JGE
- קפוץ אם התוצאה החשבונית, או הלוגית, האחרונה הינה 0: JZ
- קפוץ אם התוצאה אינה 0: JNZ

שים לב:

1. הפקודה JE זהה לפקודה JZ, וניתן לבחור בכל אחת מהן. המשמעות היא שהתוצאה הלוגית של מצב שוויון נותנת את התוצאה 0.
2. הפקודות JG ו-JL מתייחסות לערכים כאל מספרים מסומנים!

תוכנית לדוגמה

הבעיה: כתוב תוכנית בשפת אסמבלי, שבודקת אם אוגר AL מכיל מספר שערכו גדול מערך המספר שבאוגר DL.

אם "כן" - סיים את התוכנית.
אם "לא" - הצב 0 באוגר AL ובאוגר DL.

השווה בין הערכים שב-AL וב-DL.	CMP AL,DL
אם AL (השמאלי) גדול - קפוץ לתווית BYEBYE	JG BYEBYE
אם לא קפצת (כי AL לא גדול) -	MOV AL,0
אפס את אוגר AL ואת אוגר DL	MOV DL,0
סיים את התוכנית	BYEBYE: NOP

הצבת ערכים הקסדצימליים באוגר

כאשר הספרה הראשונה של המספר הינה אות, יש להוסיף את הספרה 0 (אפס) לפני המספר.

לדוגמה, יש לכתוב:	MOV AL,0FFH
אך לא לכתוב (שגוי):	MOV AL,FFH

תרגילים ופתרונות

הערה כללית: כאשר מצוין "תא 600H", הכוונה היא לתא הזיכרון בכתובת 600H.

תרגיל 1

כתוב תוכנית המעתיקה את הנתון שנמצא בתא 600H, אל תא שכתובתו 605H.

פתרון א׳:

מציב 600 באוגר BX	MOV BX,600H
מעתיק נתון מתא זיכרון שכתובתו נמצאת ב-BX אל אוגר AL	MOV AL,[BX]
מציב 605H באוגר SI	MOV SI,605H
מעתיק את הערך שנמצא באוגר AL לתוך תא זיכרון שכתובתו נמצאת באוגר SI (כלומר כתובת 605H)	MOV [SI],AL

פתרון ב׳:

מציב באוגר SI את הערך 600H	MOV SI,600H
מעתיק לאוגר DL ערך שנמצא בתא שכתובתו רשומה באוגר SI	MOV DL,[SI]
מעתיק את הערך מאוגר DL לתוך תא זיכרון שכתובתו נמצאת באוגר SI ועוד 5 (כלומר לכתובת 605H)	MOV [SI+5],DL

פתרון עם שגיאה אופיינית:

שלוש הפקודות הראשונות הן נכונות:	MOV SI,600H
	MOV DI,605H
	MOV AL,[SI]
כאן הטעות מתחילה. לשם מה להעתיק את הערך מתא 605H אל אוגר AH?	MOV AH,[DI]
המשך הטעות: העתקת הערך מאוגר AL אל אוגר AH אינה מעתיקה את הערך לתוך תא הזיכרון, אף על פי שהמתכנת סבור שכך הדבר.	MOV AH,AL

תרגיל 2

כתוב תוכנית המוסיפה 4 לתוכן של תא 990H.

פתרון א':

מציב 990H באוגר SI	MOV SI,990H
מעתיק תוכן תא 990H לאוגר AL	MOV AL,[SI]
מוסיף 4 לערך שנמצא באוגר AL	ADD AL,4
מעתיק את הערך (המוגדל ב-4) חזרה לתוך תא 990H	MOV [SI],AL

פתרון ב':

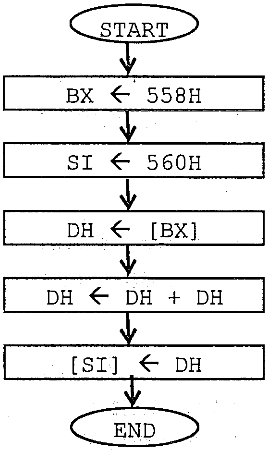
	MOV DI,990H
	MOV CL,4
מוסיף 4 (תוכן של CL) לתוך תא 990H	ADD [DI],CL

תרגיל 3

כתוב תוכנית הכופלת פי 2 את הערך שנמצא בתא 558H (את התוכן של התא), ולאחר מכן מציבה את התוצאה בתא 560H בזיכרון.

מציב 558H באוגר BX	MOV BX,558H
מציב 560H באוגר SI	MOV SI,560H
מעתיק ל-DH את תוכן תא 558H	MOV DH,[BX]
כופל (על ידי חיבור הערך לעצמו)	ADD DH,DH
מעתיק את התוצאה לתא 560H	MOV [SI],DH

תרשים זרימה:



תרגיל 4

כתוב תוכנית המחברת את תוכן תא 1200H עם תוכן תא 1201H, ומציבה את התוצאה בתא 1203H.

פתרון א':

MOV DI,1200H	
MOV SI,1201H	
MOV BX,1203H	
MOV AL,[DI]	מעתיק ל-AL את הערך מתא 1200H
MOV AH,[SI]	מעתיק ל-AH את הערך מתא 1201H
ADD AL,AH	מחבר ל-AL את הערך שב-AH
MOV [BX],AL	מציב את התוצאה בתא 1203H

פתרון ב':

MOV BX,1200H	
MOV DL,[BX]	מעתיק ל-DL את תוכן תא 1200H
ADD DL,[BX+1]	מוסיף ל-DL תוכן תא 1201H
MOV [BX+2],DL	מציב תוצאה בתא 1203H

תרגיל 5

כתוב תוכנית באסמבלי המחליפה בין תוכן תא 700H לבין תוכן תא 800H.

MOV SI,700H	
MOV DI,800H	
MOV AL,[SI]	
MOV AH,[DI]	
MOV [SI],AH	מציב את הערך שהועתק מתא 800H לתוך תא זיכרון 700H
MOV [DI],AL	מציב את הערך שהועתק מתא 700H לתוך תא זיכרון 800H

תרגיל 6

כתוב תוכנית בשפת אסמבלי, שתמצא מהו ההפרש בין תוכן תא 3000H לבין תוכן תא 200H ותציב את התוצאה בתאים 100H ו- 102H

MOV SI,3000H	
MOV DH,[SI]	מעתיק את הנתון מתא 3000H לאוגר DH
MOV SI,200H	שים לב: מותר להשתמש שוב באוגר SI מכיון שהוא סיים תפקידו כמצביע על תא 3000H. כעת הוא יצביע על תא 200H
SUB DH,[SI]	מחסרים את הערך של תא 200H מאוגר DH
MOV SI,100H	
MOV [SI],DH	מציב את התוצאה בתא 100H
MOV [SI+2],DH	מציב את התוצאה בתא 102H

תרגיל 7

כתוב תוכנית הבודקת אם הערך שבתא 777H שווה ל-3.

אם כן - התוכנית מציבה FF בתא 800H, ואם לא - היא מסתיימת.

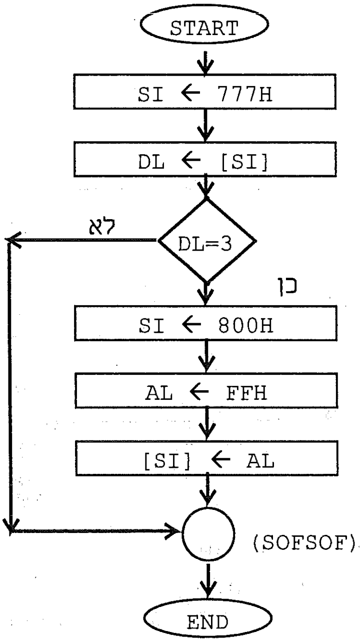
פתרון א':

`MOV SI,777H`	
`MOV DL,[SI]`	
`CMP DL,3`	משווה בין הערך בתא 777H לבין 3
`JNE SOFSOF`	אם אינם שווים - קפוץ ל-sofsof
`MOV SI,800H`	אם התוכנית הגיעה לכאן, המשמעות היא
`MOV AL,0FFH`	שהערכים שווים. לכן הצב את הערך
`MOV [SI],AL`	0FFH בתא שבכתובת 800H
`SOFSOF:NOP`	סיום התוכנית: לא לעשות דבר.

הערות:

- לפני המספר FF הוסף 0, מכיון שעבור כל מספר שהספרה הראשונה שלו היא אות, צריך להוסיף לפניו את הספרה 0 (אפס).
- התווית SOFSOF מציינת מיקום בתוכנית. אין חשיבות לשם שנבחר כתווית, אולם רצוי שהוא ירמוז על התפקיד של קטע התוכנית. **אסור** לבחור בשם שהוא פקודה או הוראה באסמבלי (כגון MOV או JE), ואסור שהשם יכיל רווחים (התווים צריכים להיות צמודים).

תרשים זרימה:



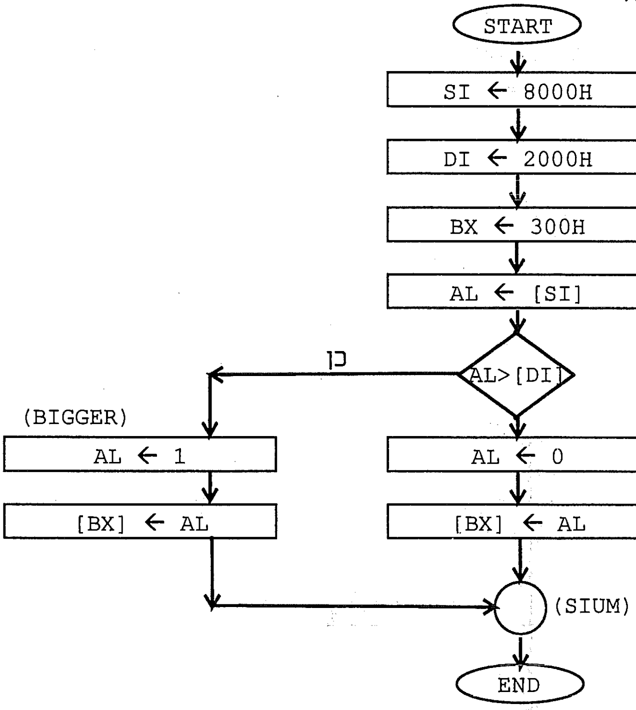
תרגיל 8

כתוב תוכנית שבודקת אם תא זיכרון 800H מכיל ערך גדול מזה של תא 2000H.

אם כן, הצב 1 בתא זיכרון 300H. ואם לא, הצב 0 בתא זיכרון 300H.

	MOV SI,800H	מציב 800H באוגר SI (ישמש כמצביע)
	MOV DI,2000H	מציב 2000H באוגר DI (מצביע)
	MOV BX,300H	מציב 300H ב-BX (אף הוא מצביע)
	MOV AL,[SI]	מעתיק ל-AL את ערך תא שכתובתו 800H
	CMP AL,[DI]	משווה בין ערך זה לערך שבתא 2000H
	JG BIGGER	אם AL גדול יותר (800H) - קופץ
	MOV AL,0	אם לא קפץ - מציב 0 באוגר AL
	MOV [BX],AL	מציב 0 גם בתא שכתובתו 300H
	JMP SIUM	ולבסוף - מדלג לסוף התוכנית
BIGGER:	MOV AL,1	כאן מציבים 1 באוגר AL
	MOV [BX],AL	ואז מציבים ערך זה בתא 300H
SIUM:	NOP	סיום תוכנית - לא לבצע דבר.

תרשים זרימה:



כאן המקום לציין, שניתן לכתוב תוכנית קצרה ופשוטה יותר לביצוע המשימה המפורטת בתרגיל זה. בפתרון זה ניסינו להציג שימושים שונים בפקודות, גם אם אינם יעילים מבחינת ביצוע התוכנית.

בהמשך הדברים נלמד מהי תוכנית הערוכה כראוי, ואת משמעות הדבר מבחינת אחזקה ועדכונים ומבחינת יעילות ביצוע. כל תוכנית ניתן לכתוב בדרכים שונות, בהתאם לנוחות ולגישה של פותר התרגיל.

תרגיל 9

כתוב תוכנית שבודקת אם הסכום של ערכי התאים 400H ו- 380H, גדול מערך תא 500H. אם כן, התוכנית תציב 0 בתאים 600H ו- 601H. במידה שלא, היא תציב 99H בתאים 277H ו- 278H.

מציב 400H באוגר BX	MOV BX,400H
מציב 380H באוגר SI	MOV SI,380H
מציב 500H באוגר DI	MOV DI,500H
מעתיק ל-DL ערך תא 400H	MOV DL,[BX]
מוסיף ערך תא 380H	ADD DL,[SI]
האם הסכום גדול מתא 500H?	CMP DL,[DI]
אם כן - קופץ למקום אחר	JG THE_BIG
אם לא קפץ - מציב 99H	MOV DL,99H
מציב 227H במצביע SI	MOV SI,277H
מציב 99H בתא 277H	MOV [SI],DL
מציב 99H בתא 278H	MOV [SI+1],DL
קופץ לסוף התוכנית	JMP SOF_PROG
אם הגיע לכאן - מציב 0	THE_BIG: MOV DH,0
מציב 600H במצביע SI	MOV SI,600H
מציב 0 בתא 600H	MOV [SI],DH
מציב 0 בתא 601H	MOV [SI+1],DH
סוף התוכנית	SOF_PROG:NOP

תרגיל 10

כתוב תוכנית בשפת אסמבלי הבודקת אם תאי זיכרון בכתובות 990H ו-991H מכילים ערכים זהים. אם כן, יש להציב בתא 1000H את הערך הזהה. אחרת - יש להציב בתא זה את ההפרש ביניהם.

	MOV SI,990H
	MOV AL,[SI]
	MOV AH,[SI+1]
	CMP AL,AH
אם ערכי תאים 990H, 991H שווים - קפוץ	JE SHAVIM
	MOV SI,1000H
	SUB AL,AH

MOV [SI],AL	מציב את ההפרש בין הערכים בתא 1000H
JMP SIYUM	
SHAVIM: MOV SI,1000H	
MOV [SI],AL	מציב את ההפרש בתא 1000H
SIYUM: NOP	

תרגיל 11

כתוב תוכנית הבודקת אם ערכי תאים 200H, 201H ו- 202H מכילים ערכים זהים. אם כן - יש להציב 1 בתא 300H. ואם לא - יש להציב בתא זה את הערך 0H .

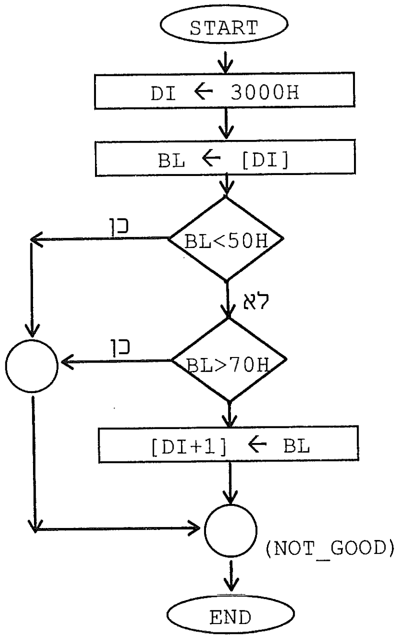
MOV SI,300H	
MOV CH,1	
MOV DH,0	
MOV BX,200H	
MOV AL,[BX]	מעתיק ל-AL את תוכן תא 200H
MOV DL,[BX+1]	מעתיק ל-DL את תוכן תא 201H
MOV CL,[BX+2]	מעתיק ל-CL את תוכן תא 202H
CMP AL,DL	
JNE NOT_EQU	אם אינם שווים - קופץ למקום אחר
CMP AL,CL	אם לא קפץ (הערכים שווים) - משווה
JNE NOT_EQU	אם הערכים אינם שווים - קופץ
MOV [SI],CH	בנקודה זו ידוע שערכי שלושת התאים שווים, ולכן מציבים 1 (ערך CH) בתא 300H (ערך SI) ולאחר מכן
JMP FINISH	קופצים לסוף התוכנית
NOT_EQU:MOV [SI],DH	אם אינם שווים - מציב 1 (הערך שבאוגר DH) בתא 300H
FINISH: NOP	

תרגיל 12

עליך לכתוב תוכנית שבודקת אם ערך תא 3000H הוא בתחום 50H עד 70H. אם כן, יש להציב ערך זה בתא 3001H.

MOV DI,3000H	
MOV BL,[DI]	
CMP BL,50H	
JL NOT_GOOD	אם BL (ערך תא 3000H) קטן מ-50, קפוץ
CMP BL,70H	
JG NOT_GOOD	אם BL גדול מ-70H, קפוץ
MOV [DI+1],BL	אם הגעת לכאן, הצב את הערך כנדרש
NOT_GOOD: NOP	

תרשים זרימה:



תרגיל 13

כתוב תוכנית באסמבלי שמציבה בתא שכתובתו 890H את הערך הגדול מבין הערכים שבתאים בכתובות 665H ו-667H.

	MOV BX,665H	
	MOV SI,667H	
	MOV DI,890H	
	MOV AL,[BX]	
	MOV AH,[SI]	
	CMP AL,AH	
	JG CONT	אם AL (ערך תא 665H) גדול - קפוץ
	MOV [DI],AH	אם לא - AH (ערך 667H) גדול יותר
	JMP ZEHU	לסיים (כדי שלא יעבור לפקודה הבאה)
CONT:	MOV [DI],AL	AL (ערך תא 665H) גדול יותר
ZEHU:	NOP	

תרגיל 14

כתוב תוכנית שתבדוק אם תא 849H מכיל ספרה בודדת. אם לא, היא מאפסת תא זה.

```
        MOV BX,849H
        MOV AL,0FH
        CMP AL,[BX]
        JGE BESEDER        ; אם 0FH גדול או שווה - קפוץ
        MOV AL,0
        MOV [BX],AL
BESEDER: NOP
```

הערה: הפתרון אינו מדויק, בכוונה. לאחר שתקרא את נושא ה"מספרים המכוונים", חזור לתרגיל זה ותקן אותו.

תרגיל 15

כתוב תוכנית שמחליפה בין ערכי התאים 39H ו-40H, אם ערכיהם אינם זהים.

```
        MOV BX,8
        MOV BX,39H
        MOV AL,[BX]
        MOV AH,[BX+1]
        CMP AL,AH
        JE SOF             ; אם שווים - סיים
        MOV [BX],AH
        MOV [BX+1],AL
SOF:NOP
```

תרגיל 16

התבונן ברשימת הפקודות, וכתוב איזו מהן אינה חוקית.

```
MOV AL,123H      ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ אינה חוקית
MOV CX,7788H
MOV [DX],AL      ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ אינה חוקית
MOV AL,BX        ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ אינה חוקית
MOV AX,DL        ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ ◄ אינה חוקית
MOV SI,DI
CMP AL,[DI]
SUB AL,34H
```

INC [SI]	אינה חוקית
ADD [BX],4	אינה חוקית
JOV [SI],[SI+1]	אינה חוקית
ADD SI,SI	

חשוב, מדוע הפקודות שאינן חוקיות, הינן כאלה?

תרגיל 17

רשום מה מבצעת כל אחת מהתוכניות הבאות:

תוכנית א'

```
MOV AL,44H
MOV DL,55H
MOV BX,670H
MOV SI,672H
MOV [BX],AL
MOV [SI],DL
```

תוכנית ב'

```
      MOV AL,8
      MOV AH,71H
      MOV DI,1000H
      CMP [DI],AL
      JE YES
      CMP [DI],AH
      JNE SOF
YES:  MOV AL,0
      MOV [DI],AL
SOF:  NOP
```

תוכנית ג'

```
      MOV BX,666H
      MOV CH,[BX]
      INC BX
      MOV CL,[BX]
      CMP CL,CH
      JNE FIN
      MOV CL,0
      MOV CH,0
FIN:  NOP
```

פתרון:

1. התוכנית מציבה 44H בתא 670H ו-55H בתא 672H.
2. התוכנית בודקת אם בתא 1000H נמצא הערך 8 או הערך 71H. אם כן, היא מציבה 0 בתא 1000H.
3. התוכנית מציבה 0 באוגרים CL, CH, אם ערך תא 666H זהה לערך התא בכתובת 667H.

תרגיל 18

מה מבצעת כל אחת מהתוכניות הבאות (ענה בקצרה)?

תוכנית א'

```
      MOV SI,500H
      MOV DI,550H
      MOV AL,[SI]
      MOV DL,[DI]
      CMP AL,DL
      JZ SOF
      MOV [SI],DL
      MOV [DI],AL
SOF:  NOP
```

תוכנית ב'

```
       MOV DI,880H
       MOV BL,[DI]
       MOV BH,[DI+1]
       MOV CL,[DI+2]
       MOV CH,[DI+3]
       ADD BL,BH
       ADD CL,CH
       CMP BL,CL
       JG FINE
       MOV [DI+20H],CL
       JMP END
FINE:  MOV [DI+20H],BL
END:   NOP
```

תוכנית ג'

```
      MOV BX,1200H
      MOV SI,780H
      MOV DI,235H
      MOV AL,[BX]
      CMP AL,[SI]
      JNE FIN
      CMP AL,[DI]
      JNE FIN
      MOV AL,0
      MOV [BX],AL
      MOV [SI],AL
      MOV [DI],AL
FIN:  NOP
```

פתרון:

1. התוכנית משווה בין תאי זיכרון 500H, 550H. אם הם שונים, מוחלפים הערכים שבתאים הללו.
2. התוכנית משווה בין סכום התאים 880H ו-881H, לבין סכום התאים 882H ו-883H, ומציבה את הגדול מבין סכומים אלה לתוך תא 1000H.
3. תוכנית זו בודקת אם הערכים של תאי זיכרון 1200H, 780H ו-235H שווים זה לזה. אם כן, היא מציבה 0 בתאים אלה. בכל מקרה אחר, התוכנית מסיימת ללא פעולה נוספת.

תרגיל 19

נתון שתוכן תא 3450H מכיל מספר X, תא 3451H מכיל מספר Y ותא 3452H מכיל מספר Z. כתוב שתי תוכניות המציבות בתא 1000H ערכים שונים: תוכנית א': מציבה את הערך X+Y-Z; תוכנית ב': מציבה את הערך Y, אם (Z-X) גדול מ-Y.

תוכנית א'

```
MOV SI,3450H
MOV AL,[SI]
ADD AL,[SI+1]
SUB AL,[SI+2]
MOV SI,1000H
MOV [SI],AL
```

תוכנית ב'

```
       MOV BX,3450H
       MOV DI,1000H
       MOV AL,[BX+2]
       SUB AL,[BX]
       MOV DL,[BX+1]
       CMP AL,DL
       JNG ZEHU
       MOV [DI],DL
ZEHU:  NOP
```

תרגיל 20

כתוב תוכנית באסמבלי, הבודקת את ערך הציון שנמצא בכתובת 4000H. נתון שהציון הינו בתחום 0-100 (איך יכול המספר 100 להיכנס לתא בגודל 8 סיביות? חשוב על כך). התוכנית תציב בתא 4001H מספרים שונים על פי תנאים אלה: את המספר 1 אם הציון בתחום 90-100, את המספר 2 אם הציון בתחום 70-89 ואת המספר 3 אם הציון אחר.

```
        MOV SI,4000H
        MOV DI,4001H
        MOV DH,[SI]
        CMP DH,69
        JG CONT1
        MOV AL,3
        JMP SOFSOF
CONT1:  CMP DH,89
        JG CONT2
        MOV AL,2
        JMP SOFSOF
CONT2:  MOV DL,1
SOFSOF:MOV [DI],AL
```

תרגיל 21

כתוב תוכנית שבודקת אם תא 990H מכיל מספר שונה מתוכן שלושת התאים 678H, 440H ו- 995H. אם כן, היא מציבה בשלושת תאים אלה את הערך 0. אם לא - היא מציבה את הערך FFH.

```
        MOV BX,990H
        MOV DL,[BX]
        MOV BX,440H
        MOV SI,678H
        MOV DI,995H
        CMP DL,[BX]
        JE SORRY
        CMP DL,[SI]
        JE SORRY
        MP DL,[DI]
        JE SORRY
        MOV AL,0
        MOV [BX],AL
        MOV [SI],AL
        MOV [DI],AL
        JMP SIUM
SORRY:  MOV DL,0FFH
        MOV [BX],DL
        MOV [SI],DL
        MOV [DI],DL
SIUM:   NOP
```

3

כתיבת תוכנית שלמה

מהי תוכנית שלמה?

כדי לכתוב תוכנית שלמה בשפת אסמבלי יש להוסיף לתוכניות שכתבנו **פתיחה וסיום**. מבלי להיכנס עדיין להסבר מעמיק על כך, ניתן להגדיר שהפתיחה עבור כל התוכניות שכתבנו עד כה תהיה:

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
```

הסיום של כל תוכנית יהיה:

```
CODE ENDS
END START
```

כלומר, כדי לכתוב תוכנית **שלמה** שמציבה למשל, את המספר 3 בתא 900H, נכתוב:

```
CODE SEGMENT                          פתיחה
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV AL,3                      גוף התוכנית
        MOV BX,900H
        MOV [BX],AL
CODE ENDS                             סיום
END START
```

מעתה, עליך להתרגל לכתוב כל תוכנית בצורה שלמה, מכיון שרק כך צריך לכתוב את התוכניות לשם הרצה במחשב.

חלק מהפתיחה והסיום אינן פקודות אסמבלי, אלא הנחיות (Directives) לתוכנת האסמבלי, המתרגמת את שפת האסמבלי לשפת מכונה.

נסביר את ההוראות שבקטעי הפתיחה והסיום:

- המילה SEGMENT מציינת שזהו **מקטע** של תוכנית אסמבלי, שגודלו המקסימלי 64K (בעניין זה נדון בהמשך).
- המילה CODE נבחרה כדי לייצג את **שם** מקטע התוכנית. במקומה ניתן לבחור כל שם אחר.

בדרך כלל מקובל להשתמש בשם של מקטע, המרמז על מהות הפעולות שמבצע קטע התוכנית. הדבר דרוש לנו, המתכנתים, ולא דרוש למחשב! מכיון שמקטע זה מכיל תוכנית (=קוד), נהוג להשתמש בשם CODE או CSEG (Code SEGment) או גם SEGMENT_CODE. אולם, כאמור אין הכרח לבחור בשם זה או אחר, והדבר נתון לשיקול דעת של המתכנת.

אם בחרת בשם אחר, רשום אותו בכל מקום שבו מופיעה המילה CODE בדוגמאות הבאות.

- המילה ASSUME מרחיבה את ההסבר אודות מקטע תוכנית האסמבלי: אוגר CS, אשר מכיל את כתובת ההתחלה של התוכנית, יתחיל במקטע בשם CODE. כך גם לגבי אוגר DS: כתובת ההתחלה של הנתונים (במידה וישנם) - יתחיל במקטע בשם CODE.
- הפקודה MOV AX,CODE מציבה בתוך אוגר AX את כתובת ההתחלה של המקטע CODE.
- לאחריה, הפקודה MOV DS,AX מציבה באוגר DS את תוכן AX, כלומר את כתובת ההתחלה של המקטע CODE. שים לב, שלא ניתן לכתוב את הפקודה MOV DS,CODE ולכן יש צורך בשתי פקודות לביצוע פעולה זו.
- ההוראה CODE ENDS מציינת שזהו סוף המקטע CODE (ENDS = END Segment).
- ההוראה END START מציינת שני דברים:
  - זהו **סוף** (end) התוכנית.
  - **תחילת** התוכנית נמצאת במקום בו מופיעה התווית START.

דוגמאות נוספות לכתיבת תוכנית שלמה באסמבלי:

1. כתוב תוכנית באסמבלי, המעתיקה את תוכן תא 777H לתא 788H.

```
DUGMA SEGMENT
   ASSUME CS:DUGMA,DS:DUGMA
HERE:   MOV AX,DUGMA
        MOV DS,AX
        MOV SI,777H
        MOV DI,788H
        MOV CL,[SI]
        MOV [DI],CL
DUGMA  ENDS
END     HERE
```

2. כתוב תוכנית המחברת את תוכן תא 1300H אל תוכן תא 1301H, ומציבה את התוצאה בתא 1005H.

```
POPYE SEGMENT
  ASSUME CS:POPYE,DS:POPYE
OLIVE:  MOV AX,POPYE
        MOV DS,AX
        MOV BX,1300H
        MOV DL,[BX]
        ADD DL,[BX+1]
        MOV BX,1005H
        MOV [BX],DL
POPYE ENDS
END OLIVE
```

4

תרגול מעשי

כיצד מריצים תוכנית אסמבלי?

לימוד עקרונות השפה ללא תרגול, דומה ללימוד תיאורטי של משחק כדורסל ללא אימונים במגרש. ברצוני להדגיש, שהניסיון מלמד שכל מי שאינו מתרגל, מבצע טעויות לוגיות ותחביריות רבות. אם כן, הבה ניגש למלאכת התכנות וההרצה.

ה"מאמן" הטוב ביותר לשפת תכנות הינו המחשב האישי (PC). באמצעותו ניתן לכתוב תוכניות באסמבלי, למצוא טעויות תחביריות (לדוגמה, לגלות שכתבנו פקודה שגויה כמו CMP AL,DX), לבדוק אם התוכנית פועלת כנדרש, לראות ולבחון את התוצאות ולמצוא בהן טעויות לוגיות שנעשו.

כדי לכתוב תוכנית בשפת אסמבלי, אנו זקוקים לכלים שיפורטו להלן. בדרך כלל, מומלץ שהתוכנות הללו יהיו בדיסק הקשיח, בספריה אחת או יותר. אם אתה משתמש בדיסקט, רצוי שהן יימצאו בדיסקט אחד.

1. **תוכנת עורך** (editor): באמצעותו כותבים את התוכנית. כל עורך מתאים למטרה זו, ובלבד שיפיק קוד ASCII טהור ללא תווי עריכה המקובלים במעבד תמלילים. העורך EDIT של DOS/Win95/Win98 למשל, מתאים למטרה זו.
2. **תוכנת אסמבלר MASM**: תוכנה זו קולטת את פקודות התוכנית **בשפת מקור** (source code) כפי שהן נכתבות על ידי המתכנת, מוצאת ומודיעה אם קיימות בה שגיאות תחביריות ומתרגמת את התוכנית **לשפת מכונה** (object code); את התוכנה הזו יש לרכוש מאחד מספקי התוכנות.
3. **תוכנת קישור LINK**: מכינה את התוכנית בשפת מכונה שהוכנה על ידי המהדר **לתוכנית בת-ביצוע שניתנת להרצה** (executable) בפיקוח מערכת ההפעלה; תוכנה זו יש לרכוש מאחד מספקי התוכנות.
4. **תוכנת בדיקה DEBUG** (של DOS/Win95/Win98, כגון TD של Borland): מאפשרת הרצת התוכנית, ניפוי שגיאות ובדיקת התוצאות בשלבים שונים של ההרצה.

כדי להבין **כיצד** (ועדיין לא "מדוע") משתמשים בתוכנות אלו, נפתור את התרגילים הבאים, תוך שימוש במחשב. נקליד את התוכניות ונריץ אותן.

שים לב, אלו תוכנות שלמות, כפי שלמדת בפרק הקודם. אם לא תכתוב כך, לא תוכל להריץ אותן.

תרגיל מעשי: שלבי תכנות, הרצה ובדיקה

כדי להסביר בפירוט את שלבי התכנות וההרצה של התוכנית, נשתמש בתוכנית לדוגמה. בתוכניות הדוגמה הבאות נסתמך על הדברים שכבר למדנו ונסקור את שלבי הרצתן בקצרה.

כתוב והרץ תוכנית באסמבלי, המציבה את הערך 66H בתא זיכרון שכתובתו 1000H.

את כל השלבים יש לבצע ב-DOS או במצב DosBos של Win95/Win98.

כתיבת תוכנית דוגמה באמצעות עורך

כאמור, ניתן להשתמש בעורך כלשהו, כגון EDIT, המפיק קוד ASCII ‏(Word אינו מתאים).

1. טעינת העורך וכתיבת **שם קובץ** בעל **סיומת ASM**. סיומת זו מרמזת על השפה שבה נכתבת התוכנית, לדוגמה, נבחר בשם TAR1.ASM.

2. הקלדת התוכנית:

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
 START: MOV AX,CODE
        MOV DS,AX
        MOV AL,66H
        MOV BX,1000H
        MOV [BX],AL
 CODE ENDS
 END START
```

3. שמירת התוכנית שכתבנו, ויציאה מהעורך.

תרגום התוכנית באמצעות אסמבלר MASM

בהנחה ששם התוכנית שלנו הוא TAR1.ASM, נכתוב את הפקודה הזו :

MASM TAR1,,;

כאשר אנו נמצאים מחוץ לתוכנת העורך יש להקליד MASM, כדי להפעיל את האסמבלר לתרגום תוכנית המקור שלנו לשפת מכונה. אחרי המילה MASM נקליד רווח, את שם התוכנית שלנו (ללא סיומת), שני סימני פסיק (,) ולבסוף - נקודה-פסיק (;). כל אלה הם תווי בקרה לתוכנת MASM, את משמעותם נלמד בהמשך.

התוכנה MASM מציגה על המסך הודעות על שגיאות תחביר, אם יש כאלו בתוכנית. אם התגלו שגיאות, **אין להמשיך לשלב הבא**, אלא לחזור לשלב הקודם, לשלב כתיבת התוכנית, ולתקן אותן באמצעות העורך.

קישור באמצעות תוכנת LINK

בהנחה ששם התוכנית שלנו הוא TAR1.ASM, נכתוב את הפקודה הזו :

LINK TAR1,,;

כפי שתוכל לראות, צורת הכתיבה של ההוראה הזו דומה לכתיבת ההוראה להרצת MASM.

אם מתקבלות הודעות שגיאה, יש לחזור לשלב עריכת התוכנית ולתקן אותן. בשלב זה, ניתן להתעלם מהודעה אחת, שהיא למעשה הודעת אזהרה, שפירושה "אין מקטע מחסנית" : WARNING: NO STACK SEGMENT.

הרצת התוכנית ובדיקתה על ידי תוכנת DEBUG

תוכנת DEBUG משמשת לבדיקת תוכניות, אשר הוכנו להרצה (סיומת EXE או COM). כפי שנראה בדוגמאות ההרצה הבאות. תחילה נתרגל מעט, ואחר כך נסקור את אפשרויות התוכנה בפירוט רב יותר.

את האפשרויות השונות של תוכנית DEBUG נפעיל באמצעות הוראות בנות תו אחד, או יותר, אשר נכתוב לאחר שהתוכנה תציג לפנינו את הסימן - (מינוס). נתחיל בבדיקת התרגיל שכתבנו זה עתה, TAR1.

1. נקליד : DEBUG TAR1.EXE

   כלומר, נכתוב את שם התוכנה DEBUG, אחריה רווח, שם קובץ התוכנית שלנו עם סיומת EXE, ולבסוף נקיש Enter.

2. לאחר שמופיע הסימן - (מינוס), נקליד U0 (התו U ולאחריו הספרה אפס - 0) ונקיש Enter.

פקודה זו גורמת להצגת התוכנית שלנו, אשר נמצאת בדיסק בצורה של שפת מכונה, לתוכנית המוצגת בשפת אסמבלי. אנו כותבים את הספרה 0 (אפס) לצד הפקודה U, כדי להציג את התוכנית מראשיתה, מכתובת 0.

בעת עיון בתוכנית המוצגת, ראוי לשים לב לדברים אלה:

- חלק מהפתיחה והסיומת אינם מופיעים!
- לאחר התוכנית שלנו מופיעות מספר פקודות שאינן שייכות לתוכנית. נתעלם מהן בשלב זה.
- תוכל לראות את התוכן של כל האוגרים על ידי כתיבת ההוראה R והקשה על Enter.
- המספר הרשום בעמודה השנייה משמאל, לצד הפקודה הראשונה, הינו 0000. בשורה הבאה רשום המספר 0003, וכך הלאה. מספרים אלה מייצגים כתובות של תאי זיכרון, שבהם מאוחסנות הפקודות בשפת מכונה.
- יש לשים לב מהו המספר הרשום בשורה **שלאחר** התוכנית שלנו. במקרה זה נרשם המספר 000C. ערך זה מייצג את **כתובת תא הזיכרון שאחרי התוכנית** שלנו.
- כל הערכים ב-DEBUG הינם על פי בסיס הקסדצימלי (בסיס 16).

3. כדי להריץ את התוכנית נכתוב את הפקודה: G=0 C

לאחר צירוף התווים **=G** ("Go"), נכתוב את השורה הראשונה של התוכנית (שורה 0). לאחר מכן, נכתוב תו רווח ואחריו את הכתובת של הפקודה שלאחר סוף התוכנית (הכתובת C, ראה סעיף קודם). נסיים בהקשה על Enter.

משמעות הפקודה: נא להריץ את הפקודות הנמצאות בתאי הזיכרון החל מכתובת אפס ועד C, לא כולל.

4. נבדוק את תוצאת ההרצה. נבדוק שהתוכנית הציבה את הערך 66H בתא 1000H.

כדי להפעיל את אפשרות הבדיקה של DEBUG נכתוב את הפקודה: D 1000 1000

הפקודה כוללת את האות D (Dump), רווח, כתובת תא הזיכרון הראשון שרוצים להציג, רווח וכתובת התא האחרון להצגה.

במקרה זה אנו רוצים לראות את תא 1000H בלבד, ולכן זהו התא הראשון וגם האחרון הרשום בפקודה.

אם תוכן התא הוא 66 - התוכנית פועלת כנדרש.

5. יציאה מתוכנת DEBUG: הקשה על Q (Quit) ולאחר מכן הקשה על Enter.

תרגילים מעשיים לדוגמה

בפתרון תרגילים אלה ניישם את אשר למדנו עד כה.

תרגיל מעשי א'

כתוב תוכנית שתציב את הערך 1 בתא 400H ואת הערך 2 בתא 401H.

שלבי הביצוע

א. כתוב את התוכנית באמצעות עורך, שמור וצא מהעורך, כפי שעשית בתרגיל שבסעיף הקודם. השם שנבחר לתוכנית הוא DUG2.ASM:

```
CODE    SEGMENT
ASSUME CS:CODE,DS:CODE
BEGIN: MOV AX,CODE
        MOV DS,AX
        MOV AL,1
        MOV BX,400H
        MOV [BX],AL
        INC AL
        INC BX
        MOV [BX],AL
CODE    ENDS
END     BEGIN
```

ב. כתוב את הפקודה: MASM DUG2,,;

אם אין שגיאות תחביר תוכל להמשיך לשלב הבא.
אם יש שגיאות - עליך לחזור לשלב העריכה ולתקן אותן.

ג. כתוב את הפקודה: LINK DUG2,,;

ד. הפעל את תוכנת DEBUG: DEBUG DUG2.EXE

עכשיו תוכל להתחיל ב**בדיקה עצמה**:

1. כתוב U0 ובדוק את מספר השורה שאחרי התוכנית שלך. זוהי "שורה" 12, או הכתובת 12H.

2. הרץ את התוכנית: G=0 11

3. בדוק את תוכן התאים 400 ו-401: D 400 401

עליך לקבל בתא השמאלי (תא 400) את הערך 1, ובתא הבא (401) - את הערך 2.
אם קיבלת זאת - התוכנית פועלת כנדרש.

רגע אחד! אולי **במקרה** כבר היו הערכים הללו בתאים!

כדי לוודא שהתוכנית אמנם גורמת להצבת הערכים, נבצע את הפעולות הבאות:

1. הקלד את הפקודה: "E 400" (שמשמעותה הכנסת נתון לתא זיכרון) והקש Enter. הערך הנוכחי של תא 400H יוצג לפניך ותוכל לשנות אותו. הקלד את המספר 22 ולאחר מכן הקש על מקש הרווח, כדי לקבל את התא **הבא**.

2. כעת יוצג לפניך ערכו של תא 401H. הקלד ערך חדש לתא זה: הערך 44. כעת, משסיימת להכניס נתונים חדשים לתאים 400 ו-401, לחץ Enter.

3. בדוק כעת את ערך התאים. לשם כך, הקלד את הפקודה "D 400 401" והקש Enter. תראה כעת, שתא 400H מכיל 22H ותא 401H מכיל 44H.

4. הרץ שוב את התוכנית. הקלד: "G=0 11" ואחר כך הקש על Enter.

5. בדוק את ערכי התאים באמצעות הפקודה: "D 400 401" והקש על Enter. אם תא 400H מכיל 1 ותא 401H מכיל 2 תוכל להסיק שהתוכנית פועלת כראוי.

תרגיל מעשי ב'

כתוב תוכנית שמשווה בין הערכים בתאים 2000H ו- 2001H.

אם הערכים שווים, התוכנית מסתיימת. אם הערכים שונים, התוכנית תציב את הערך 55H בשני התאים האלה.

שלבי הביצוע

1. כתוב את התוכנית באמצעות עורך, שמור וצא. שם התוכנית: TARGIL3.ASM.

```
CSEG SEGMENT
ASSUME CS:CSEG,DS:CSEG
START: MOV AX,CSEG
       MOV DS,AX
       MOV BX,2000H
       MOV AL,[BX]
       CMP AL,[BX+1]
       JE SAYEM
       MOV AL,55H
       MOV [BX],AL
       MOV [BX+1],AL
SAYEM: NOP
CSEG ENDS
END START
```

2. הרץ את התוכנית באמצעות האסמבלר: MASM TARGIL3,,;

3. הרץ את התוכנית באמצעות המקשר: LINK TARGIL3,,;

4. הרץ את התוכנית באמצעות DEBUG, כדי לבדוק אותה.

ההרצה לבדיקת התוכנית תיעשה בשלבים

1. הקלד את הפקודה: DEBUG TARGIL3.EXE
2. הקלד את הפקודה: U0

   בדוק את מספר השורה לאחר התוכנית (קיבלנו כתובת 17H).

3. הצג את תאי הזיכרון כדי לבדוק אם הם שווים בערכם: D 2000 2001

   נניח שמצאת שערכם **שונה**.

4. הרץ את התוכנית: G=0 17
5. בדוק שוב את תאי הזיכרון הללו: D 2000 2001

מכיון שתאי זיכרון אלה הכילו ערכים שונים לפני הרצת התוכנית, עלינו לראות כעת, לאחר ההרצה, שתאים אלה השתנו וקיבלו את הערך 55H.

האם בזאת תמה בדיקת התוכנית? **לא!** עלינו לבדוק גם את המצב ההתחלתי האחר, שבו יש בשניהם ערכים שווים. לשם כך, נציב ערכים זהים בשני התאים ונוודא שהפעם תוכן התאים לא השתנה.

6. הצב ערכים זהים בשני התאים: E 2000 (ולחיצה על Enter), הקלד את הספרה 8, הקש על מקש הרווח, הקלד שוב את הספרה 8 (הפעם, עבור תא 2001) וסיים בהקשה על Enter.

   כעת, בדוק שבשני התאים ישנו הערך הזהה 8 שהוזן בהם. עשה זאת באמצעות הפקודה: D 2000 2001

7. הרץ שוב את התוכנית: G=0 17
8. בדוק מהו הערך שנמצא בשני התאים: D 2000 2001

אם נמצא בשני התאים את הערך 8 שהיה בהם מראש, לפני הפעלת התוכנית, תוכל להסיק שהתוכנית לא שינתה אותם, והיא פועלת כנדרש!

תרגיל מעשי ג'

כתוב תוכנית המשווה בין תוכן אוגר BX לבין תוכן אוגר CX, ומעתיקה את הגדול מביניהם אל אוגר DX.

שלבי הביצוע

1. הקלד את התוכנית בעורך, שמור וצא. שם התוכנית: YES.ASM.

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        CMP CX,BX
```

```
        JG GADOL
        MOV DX,BX
        JMP SOF
GADOL:  MOV DX,CX
SOF:    NOP
CODE    ENDS
END     START
```

2. הרץ את האסמבלר: MASM YES,,;

3. הרץ את המקשר: LINK YES,,;

4. הרץ את התוכנית שלך ובדוק אותה באמצעות DEBUG.

שלבי הרצת הבדיקה באמצעות DEBUG

1. הפעל את התוכנה: DEBUG YES.EXE

2. הקלד את הפקודה U0 ובדוק את מספר השורה שאחרי התוכנית. בתוכנית זו קיבלנו מספר כתובת 11H.

שים לב! התוכנה "שותלת" לעיתים פקודת NOP שלא נכתבה במקור, בעיקר לאחר הפקודה JMP. שים לב ש-NOP זו אינה הפקודה NOP שאנו כותבים בסוף התוכנית שלנו.

בוודאי שמת לב שכל התוויות שרשמת בתוכנית, כמו למשל SOF ו-GADOL, אינן מופיעות בתוכנית שהוצגה על ידי DEBUG, ובמקומן מצוינות **כתובות זיכרון**. כך למשל, הפקודה המקורית JG GADOL, שונתה לפקודה JG 000E. ניתן לראות, שהכתובת 000E מכילה את תחילת הפקודה MOV DX,CX. זו גם הפקודה לצד התווית GADOL בתוכנית המקורית.

3. הכנס לאוגר BX את הערך 3 ולאוגר CX הכנס את הערך 5:
   - כתוב את הפקודה R BX (R מציין רגיסטרים, אוגרים) והקש Enter.
   - לפניך יוצג התוכן הנוכחי של אוגר BX. הקש את הערך 3 ו-Enter.
   - הקלד את הפקודה R CX והצב את הערך 5.
   - ודא שערכם של האוגרים שונה עכשיו. עשה זאת על ידי הקשה על R. כל האוגרים יוצגו עכשיו על המסך.

4. הרץ את התוכנית: G=0 11

5. בדוק אם באוגר DX הוצב הערך הגדול יותר (5).

6. הצב כעת לאוגר BX ערך גדול יותר (9). עשה זאת כך: הקלד את הפקודה "R BX", הקש 9 ולחץ על Enter.

7. הרץ שוב את התוכנית באמצעות הפקודה "G=0 11". בדוק שאכן הערך של BX, כלומר 9, הוצב באוגר DX.

בזאת סיימת את בדיקת התוכנית!

שאלה לבדיקה: האם באסמבלי ישנה רגישות לסוג האותיות, למשל האם המילה "CODE" והמילה "Code" נחשבות לזהות? בדוק והשב.

תרגיל מעשי ד'

כתוב תוכנית שבודקת אם בשלושת תאי הזיכרון 410H, 411H ו-412H ישנם ערכים הגדולים מ-2. אם כן - הצב 1 באוגר DX. אם לא (מספיק שאחד מהם אינו גדול מ-2) - הצב ב-DX את הערך 0.

השלבים לכתיבת התוכנית והרצתה

1. כתוב את התוכנית בעורך, שמור וצא. שם התוכנית: CHECK.ASM.

שים לב, שהתוכנית שלפניך אינה הפתרון הטוב ביותר לבעיה, אולם היא תוכל לשמש פתרון מספק בשלב זה.

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
BEGIN: MOV AX,CODE
       MOV DS,AX
       MOV DX,0
       MOV SI,410H
       MOV AL,[SI]
       CMP AL,2
       JNG SOFY
       INC SI
       MOV AL,[SI]
       CMP AL,2
       JNG SOFY
       INC SI
       MOV AL,[SI]
       CMP AL,2
       JNG SOFY
       MOV DX,1
SOFY:  NOP
CODE   ENDS
END    BEGIN
```

2. הפעל את MASM, כדי לבדוק שגיאות תחביר וליצור תוכנית בשפת מכונה: MASM CHECK,,;

3. הפעל את תוכנת LINK כדי ליצור קובץ מוכן להרצה: LINK CHECK,,;

4. הרץ את התוכנית ובדוק אותה באמצעות DEBUG.

שלבי ההרצה והבדיקה

1. כתוב את הפקודה U0 להצגת התוכנית.

להפתעתנו, התוכנית אינה מופיעה בשלמותה! כל שעליך לעשות, הוא לכתוב את הפקודה **U** (ללא 0), ואז יופיע המשך התוכנית.

נסביר זאת: U0 הינה פקודה שמציגה את התוכנית החל מכתובת 0, ואילו U בלבד מציינת שאנו רוצים לראות את **המשך** התוכנית.

שים לב לשורה הראשונה בתוכנית: כתבנו את הפקודה "MOV AX,CODE", ואילו ב-DEBUG נכתב מספר כלשהו, שאינו קבוע. מה פירוש הדבר?

הסבר "על קצה המזלג": המספר מציין את הכתובת ההתחלתית שנבחרה במחשב כדי לאחסן נתונים, ולכן, אם נרצה לראות אילו נתונים מכילים תאי הזיכרון 410H עד 412H, לא נוכל להסתפק בפקודה "D 410 412". לשם כך, נצטרך לבדוק מהו המספר שנרשם בשורה הראשונה (במקום CODE). **נניח** שמספר זה הינו 1127, ואז נרשום את הפקודה "D 1127:410 412". משמעות הפקודה: הצג את תאי הזיכרון בכתובות 410 עד 412 אשר **יחסיים לכתובת** 1127 (על כתובות יחסיות נלמד בהמשך).

2. הצב ערכים גדולים מ-2 (נניח 6) לתאים 410H עד 412H. השתמש בפקודה זו: F 1127:410 412 6 ולחץ על מקש Enter. (זכור ש-"1127" צריך להיות מוחלף בכתובת שמופיעה אצלך במחשב).

הפקודה F (Fill=) מאפשרת למלא נתונים ברצף תאי זיכרון. במקרה זה מציבים את הערך 6 בתאים 410H עד 412H, יחסית לכתובת 1127 (הערך הרשום כאן כ-1127 הינו סתמי ולצורך ההסבר - כאמור, יש להסתכל על המספר הרשום בפקודה הראשונה, **במקום** המילה CODE.

הערה: מספרים גדולים מהערך 7FH (למשל 83H) הינם מספרים שליליים, ולכן הם אינם גדולים מ-2 ואינם מתאימים להצבה בתאי הזיכרון בתוכנית זו.

3. הרץ את התוכנית: G=0 23

4. התאים 410H עד 412H הכילו מספרים גדולים מ-2, ולכן עליך לראות אם הערך הרשום באוגר DX הינו 1.

5. כעת, יש לשנות אחד מהתאים, כך שיכיל מספר שאינו גדול מ-2. שנה, למשל, את ערך תא 411H לערך 1: הקלד E 411 והקש Enter, הקש את הערך 1 ושוב Enter.

6. הרץ שנית את התוכנית: G=0 23

7. כעת תוכל לראות שאוגר DX מכיל 0.

התוכנית פועלת כנדרש!

טעויות אופייניות בתכנות: הסבר וסיכום

לפני שתתחיל לכתוב ולהריץ תוכניות מעשה-ידיך-להתפאר, כדאי שתכיר טוב יותר כל אחד מהשלבים של פיתוח התוכנית. חשוב לדעת אילו שגיאות מתגלות בכל שלב, להבין אותן וללמוד כיצד להימנע מהן ולתקן אותן, אם בכל זאת אירעו.

שגיאות בהרצת MASM

הערה: תוכנות MASM שונות יכולות לתת הודעות שגיאה שונות. להלן יוצגו מספר דוגמאות בלבד.

השגיאות התחביריות המתגלות בתוכנית של המתכנת מוצגות על גבי המסך. פרט לכך יוצרת התוכנה שני קבצים. האחד, שמו כשם התוכנית עם סיומת OBJ. קובץ זה מכיל את התוכנית בשפת מכונה.

האחר, שמו כשם התוכנית ובעל סיומת LST. קובץ זה מכיל את פקודות התוכנית והשגיאות התחביריות שהתגלו בה (את השגיאות הלוגיות נתקן במהלך ההרצה). ניתן להציג את התוכן של קובץ זה על גבי המסך, או להדפיס אותו, באמצעות הפקודה Type של DOS. אם לדוגמה, שם הקובץ הינו YES.LST,

הפקודה להצגה במסך: TYPE YES.LST

הפקודה להדפסה במדפסת: TYPE YES.LST > LPT1

לא תמיד השגיאות המוצגות על גבי המסך הינן ברורות ומובנות, אולם על פי רוב ניתן להבין את הטעויות. נבחן מספר טעויות נפוצות:

1. אם שכחנו לרשום את ההוראה CODE SEGMENT, אנו עלולים לקבל מספר הודעות בו-זמנית. לדוגמה:

```
      ASSUME CS:CODE,DS:CODE
error >> symbol not defined
      BEGIN:MOV AX,CODE
error >> not exists, or not reachable cs
      CODE ENDS
error >> block nesting error
```

כל הטעויות נובעות מהשמטת שתי מילים בלבד! חשוב להבין שכמות גדולה של הודעות שגיאה אינה מרמזת דווקא על טעויות רבות, לכן יש לנסות להבין אותן ולחפש מהי הטעות, או הטעויות.

2. הודעות דומות נקבל אם נשמיט בטעות את השורה:

```
ASSUME  CS:CODE,DS:CODE
```

3. אם לא נכתוב את זוג הפקודות האלו:

```
MOV AX,CODE
MOV DS,AX
```

לא תגלה תוכנת MASM דבר! הסיבה לכך, שפקודות אלו חשובות **למבנה הלוגי** של התוכנית וחסרונן **אינו גורם לטעות תחבירית**.

זכור, תוכנת MASM אינה מהווה מכשיר לבדיקה אם התוכנית תפעל כראוי, אם תבצע את הנדרש, או אם לא יהיו בה שגיאות (באגים).

4. הפקודות הבאות "יזכו" אותנו בהודעת השגיאה הבאה:

symbol not defined (סמל שאינו מוגדר)

הכוונה לכך, שבפקודה כתובים אופרנדים שאינם מוכרים ואינם חוקיים.

- MOV AX,COD במקום: MOV AX,CODE
- MOV DL,0 במקום 0 (אפס) נרשמה האות O (או).
- JNE SOF אין בתוכנית תווית בשם SOF.
- JMP YOFI התווית הכתובה בתוכנית היא YOFY.
- MOV AL,D במקום: MOV AL,DL

5. אם נכתוב את הפקודה INC SI,1, נקבל הודעת שגיאה זו:

extra characters on line (תווים מיותרים בשורה)

כזכור, מבנה הפקודה הוא INC SI.

6. אם נכתוב בטעות את הפקודה MOV SI,AL, נקבל את הודעת השגיאה:

operand types must match

כלומר, אין התאמה בסוגי האופרנדים של הפקודה.

7. אם נשכח לכתוב את ההוראה CODE ENDS, נקבל:

open segments

הכוונה לכך שלא סומן "סוף" התוכנית.

8. הפקודה IN SI הזו למשל, אשר נכתבה במקום INC SI תגרור הודעה:

syntax error

שמשמעותה: טעות תחבירית.

9. אם לא נכתוב את ההוראה END START, תוצג לפנינו ההודעה:

?end of file encountered on input file

כלומר, לא נמצא סוף קובץ.

10. כתיבת פקודה שגויה CMP [SI],[DI] תגרום להצגת ההודעה הבאה:

improper operand type

כתבנו אופרנד שאינו חוקי.

11. הטעות בכתיבת הפקודה CMP [SI],5 תזכה לתגובה:

operand must have size

אין מצב מיידי (חשוב כיצד יש לכתוב את הפקודה!).

12. הפקודה השגויה MOV [SI],15, תגרום להצגת ההודעה:

operand must have size

13. אם לא נכתוב את הסיומת ASM בשלב כתיבת שם הקובץ עבור העורך, נקבל הודעת שגיאה כללית. הודעה כזו תתקבל בכל מקרה שהקובץ אינו נמצא.

שגיאות בהרצת תוכנת DEBUG

בשלב זה נראה שלפנינו תוכנית "בדוקה", אך גם כאן ניתן להיתקל במספר בעיות:

א. יש לשים לב שלא נרשמה ההודעה file not found, שמשמעותה "הקובץ (של התוכנית) לא נמצא".

כאשר מקבלים הודעה זו, יש לצאת מהתוכנה (באמצעות Q) ולבדוק מדוע הקובץ אינו קיים: שגיאה בכתיבת שם הקובץ, הקובץ לא נשמר במקום שאנו מחפשים, שכחנו לבצע LINK, או שנמצאו טעויות בשלב MASM, או LINK.

ב. אם יהיו מקרים בהם המחשב "נתקע" בשלב ההרצה, בדוק אם לא קיימת אחת מסיבות אלו (ואחרות):

1. כתובות הרצה אינן נכונות. כלומר, במקום להקליד למשל G=0 23, הקלדת בטעות G=0 24.

2. כתיבת הפקודה G בלבד, במקום: G=0 23, למשל.

3. טעות בתוכנית, הגורמת לביצוע לולאה אינסופית. לולאה אינסופית הינה חזרה על קטע תוכנית מסוים ללא הפסקה. לדוגמה, תוכנית זו תגרום ל"תקיעת" המחשב בלולאה כזו:

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
AGAIN:  MOV AL,4
        CMP AL,5
        JNE AGAIN
CODE ENDS
END START
```

4. בחלק מגרסאות התוכנה DEBUG יש שגיאה ("באג", תקלה) המופיעה מדי פעם. כדי להימנע ממנה, יש לבדוק את ערכו של אוגר SP לפני הרצת התוכנית. אם ערכו של האוגר הוא 2, יש לשנות את ערכו ל-0 על ידי הפקודה "R SP".

5. קיימת אפשרות שהמקור לבעיה שבתוכנית שלך הינו וירוס מחשב. בדוק זאת באמצעות תוכנה מתאימה.

אל תתפתה לחשוב שכל תוכנית שאינה פועלת כראוי, נפגעה מווירוסים. בדרך כלל הבעיה היא דווקא במתכנת.

ג. אם התוכנית "נעלמה" לפתע מהמסך, בצע פעולות אלו :

1. כתוב U0 ובדוק אם התוכנית הופיעה שוב על המסך. אם לא - עבור לסעיף הבא.

2. צא מהתוכנית באמצעות Q. היכנס שוב ל-DEBUG באמצעות הפקודה DEBUG TARGIL.EXE (בהנחה ששם התוכנית שלך הוא TARGIL). בדוק שלא קיבלת את ההודעה file not found. הקש U0 ובדוק.

תקציר פקודות DEBUG

התוכנה DEBUG כוללת פקודות רבות. כמה מהן כבר הכרנו במהלך הלימוד. כעת, נסקור בהרחבה חלק מהפקודות העיקריות של התוכנה. פקודות נוספות נלמד בהמשך.

לשם תזכורת, את ההוראות כותבים כאשר התוכנה DEBUG מופעלת ובשמאל השורה מופיע הסימן - (מינוס).

ההוראה U: Unassembly - הצגת התוכנית. פקודה זו גורמת לתרגום התוכנית הכתובה בשפת מכונה, לתוכנית בשפת אסמבלי.

דוגמאות:

U0	הצגת התוכנית החל מכתובת 0 (מההתחלה).
U0 18	הצגת התוכנית מכתובות 0 עד 18H (שים לב שאלה הן כתובות יחסיות מתחילת התוכנית ואין אלו כתובות זיכרון).
U	הצגת **המשך** התוכנית, שלאחר הקטע שהוצג.

ההוראה D: Dump - הצגת תוכן תאי זיכרון.

דוגמאות:

D300	הצגת תוכן תאי הזיכרון החל מכתובת 300H.
D300 320	הצגת תאי הזיכרון 300H עד 320H.
D1922:100 120	הצגת תאי הזיכרון בכתובות 100H עד 120H, החל מכתובת 1922H. הכתובת 1922H נקראת **כתובת הבסיס** (base address), הכתובות 100H ואחרות נקראות **היסט** (offset). בנושא הכתובות נחזור ונדון בהמשך.

הוראה E:	Enter - הכנסת נתון לתא בודד, או למספר תאים בודדים רצופים.
דוגמה:	
E700	הצגת הערך של תא 700H, ואפשרות לשנותו על ידי הקלדת הערך הרצוי (לסיום יש ללחוץ Enter). ניתן גם ללחוץ על מקש הרווח, כדי לקבל את התא **הבא** ולשנותו, אם נרצה.

הוראה F:	Fill - הכנסת נתון לקבוצת תאים.
דוגמה:	
F 300 350 24	הכנסת הערך 24H לכל התאים בכתובות 300H עד 350H.

הוראה G:	Go - הרצת התוכנית.
דוגמאות:	
G=0 17	הרצת התוכנית מכתובת 0 ועד כתובת 17H (**לא** כולל).
G 45	הרצת התוכנית מהכתובת **הנוכחית** ועד 45H. הכתובת הנוכחית של התוכנית נמצאת באוגר IP. יש להיזהר בעת כתיבת הפקודה ולא לטעות בכתיבתה, דבר שעלול לגרום ל"תקיעת" המחשב.

הוראה R:	Registers - הצגת תוכן האוגרים.
דוגמאות:	
R	הצגת תוכן כל האוגרים.
R AX	הצגת תוכן אוגר AX, עם אפשרות לשנותו. אם רוצים לשנות רק את אוגר AL, צריך לשנות את AX. כלומר, לרשום מספר אחד הבנוי משתי הספרות השמאליות המקוריות, ושתי הספרות הימניות החדשות.

הוראה T:	Trace - הרצת פקודה בודדת. פעולה זו נקראת גם מצב צעד-יחיד (single step mode).

הוראה זו הינה מבין החשובות של DEBUG, לכן נרחיב עליה מעט. בכל פעם שנכתוב את ההוראה **T**, תבוצע פקודה **אחת** בלבד של התוכנית שלנו.

כך נוכל לעקוב אחרי תוכניות שאינן פועלות כראוי. נריץ את התוכנית פקודה אחר פקודה, ובכל פעם נבדוק כיצד הושפעו האוגרים ומה התבצע בתאי הזיכרון הרלוונטים לתוכנית. בדרך זו, נוכל למצוא היכן הטעות ומהי השגיאה שגרמה לה.

הפקודה שתתבצע בכל פעם, הינה זו **שאוגר IP** מצביע עליה. אוגר זה הינו מצביע ההוראה: מצביע על כתובת ההוראה **הבאה** לביצוע. לכן, אם נרצה להריץ את התוכנית

במצב צעד-יחיד החל מתחילת התוכנית שלנו (כתובת 0), יש לעדכן את אוגר IP, כדי שיצביע על כתובת 0.

להרצה במצב צעד-יחיד נכתוב את הפקודה R IP (ו-Enter) ואחריה נזין לתוכנית את נקודת ההתחלה 0 (כתובת 0 יחסית) ו-Enter. כעת, בכל פעם שנכתוב **T** תבוצע פקודה אחת בלבד בתוכנית שלנו, ואוגר IP יקודם באופן אוטומטי לכתובת של הפקודה הבאה.

דוגמה: נתונה התוכנית הבאה המוצגת ב-DEBUG, אשר נריץ אותה כעת במצב צעד-יחיד. התוכנית מציבה את הערך 56H בתא זיכרון שכתובתו 220H. התוכנית עברה את שלבי העורך, MASM ו-LINK.

```
MOV AX,0F68
MOV DS,AX
MOV AL,56
MOV BX,220
MOV [BX],AL
```

כעת נבצע:

1. נקליד "R IP" (ו-Enter) ונזין את הערך 0 (ו-Enter).
2. נקיש T ונוכל לראות שאוגר AX קיבל את הערך 0F68. כלומר, הפקודה הראשונה פעלה כראוי. ניתן לראות שאוגר IP מצביע כעת על הכתובת של הפקודה הבאה.
3. נקיש שוב T וכעת האוגר DS מקבל את ערך AX.
4. נמשיך להקיש על T עד לפקודה האחרונה, ונבדוק בכל מצב את הביצוע בפועל. כדי לבדוק אם בתא 220H הוצב הערך 56H, נכתוב את הפקודה הזו: D 220 220.

לסיכום: יש להשתמש בפקודה זו בכל פעם שהתוכנית שכתבנו אינה מתבצעת כראוי, ואיננו מוצאים את הסיבה הנראית לעין.

תרגילים

לכל תרגיל כתוב תוכנית שלמה באסמבלי, אשר תבצע את הפעולות כפי שנדרש. הרץ כל תוכנית ובדוק אותה באמצעות המחשב.

תזכורת: כאשר כתוב "תא 600H" הכוונה היא לתא זיכרון אשר בכתובת 600H.

בהצלחה!

1. כתוב תוכנית שתציב באוגר DX את הערך 7.
2. כתוב תוכנית שתציב בתא 600H את הערך 88H.
3. כתוב תוכנית שתעתיק את תוכן תא 900H לתא 901H.
4. כתוב תוכנית שתציב באוגר AL את ערך תא 1250H, ובאוגר BL - תציב את ערך תא 1260H.

5. כתוב תוכנית שתחבר את תוכן תא 666H עם תוכן תא 668H, ותציב את התוצאה בתא 1000H.

6. כתוב תוכנית שתוסיף 1 לתוכן תא 803H, ותחסר 2 מתוכן תא 849H.

7. כתוב תוכנית שתחליף בין תוכן האוגרים CX ו-BX.

8. כתוב תוכנית שתחליף בין תוכן התאים בכתובות 480H ו- 482H.

9. כתוב תוכנית שתשווה בין תוכן אוגר BX לבין תוכן אוגר AX. אם הם שווים בערכם, התוכנית מסתיימת. אם הערכים שונים, היא תציב את הערך 9999H באוגר BP.

10. כתוב תוכנית שתבדוק אם ערך תא 2200H גדול מ-4. אם כן, היא מציבה בתא 2201H את הערך 11H. אחרת - התוכנית תסתיים.

11. כתוב תוכנית שתשווה בין ערכי התאים 570H ו- 680H. אם הערכים שווים, היא תציב 1 באוגר DX. אם הערכים אינם זהים, היא תציב 0 באוגר DX.

12. כתוב תוכנית שתשווה בין תוכן התאים 579H ו- 575H. אם תוכן תא 575H גדול יותר התוכנית תציב את הערך 1 בתא 600H ; אם לא, היא תציב 0 בתא 600H.

13. כתוב תוכנית שתבדוק אם ערך אוגר DX גדול מ-4 וקטן מ-9. אם כן, היא תציב את הערך 33H באוגר BX ; אחרת, תציב 11H באוגר BX.

14. כתוב תוכנית שתבדוק אם שני התאים 638H ו-474H מכילים ערכים קטנים מ-3. אם כן, תציב את הערך 51H בתא 900H, אם לא - תציב את הערך 50H בתא 900H.

15. כתוב תוכנית שתעתיק את הערך הגדול מבין התאים 2439H, 2440H לתא בכתובת 2450H.

16. כתוב תוכנית שתציב 1 באוגר AX, רק אם מתקיימים יחד ("**וגם**") שני התנאים הבאים : בתא זיכרון שבכתובת 1166H יש ערך גדול מ-22H, ובתא 1177H הערך שווה ל-33H.

17. כתוב תוכנית שתחשב את סכום התאים 394H ו- 395H. אם סכום זה גדול מסכום התאים 390H ו-391H, היא תציב באוגר DL את ההפרש בין שני הסכומים.

18. כתוב תוכנית שתציב את הערך FFH בתא 2000H, כאשר בתא 2001H מצוי הערך 1 או 2.

19. כתוב תוכנית שתבדוק את ערך תא 393H, ובהתאם לכך תציב ערכים בתא 430H :
    - אם הערך בתא 393H הוא 1, התוכנית תציב בתא 430H את הערך A (בבסיס 16),
    - אם הערך הוא 2 - היא תציב בתא 430H את הערך B (בבסיס 16),
    - אם הערך הוא 3 - היא תציב את הערך C,
    - אם הערך הוא 4 - התוכנית תציב בתא 430H את הערך D.

20. כתוב תוכנית שתבדוק את ערך אוגר CH. אם הוא מכיל ערך בתחום 16H עד 20H, **או** בתחום 46H עד 50H, התוכנית תציב 0 בתאים 1200H ו- 1201H.

אם תוצאת הבדיקה היא שלילית והערך אינו בתחומים המבוקשים, התוכנית תציב את הערך 1 בתא 1200H, ואת הערך 2 בתא 1201H.

21. כתוב תוכנית שתציב בתא 773H את מספר תאי הזיכרון המכילים ערכים הקטנים מ-55H. הבדיקה תבוצע על תאי הזיכרון בכתובות 990H, 991H ו-992H.

22. כתוב תוכנית שתציב 1 באוגר AH, אם מתקיים אחד, או יותר, מהתנאים הבאים:

- כל התאים 600H עד 602H (כולל) מכילים 0.
- כל התאים 600H עד 602H מכילים ערך קטן מ-44H.
- אף לא אחד מהתאים הללו גדול מ-52H.

אם לא מתקיים אף לא אחד מהתנאים האלה, התוכנית תציב 0 באוגר AX.

23. כתוב תוכנית שתבדוק אם הערכים בתאים 200H עד 202H מכילים ערכים בסדר עולה, או בסדר יורד.

אם הסדר עולה (כלומר, כל ערך גדול מקודמו), התוכנית תציב בתא 205H את ההפרש בין המספר הגדול ביותר לבין המספר הקטן ביותר.

אם הסדר יורד (כל ערך קטן מקודמו), התוכנית תציב בתא 205H את ההפרש בין הערך הגדול לבין הערך הקרוב אליו ביותר.

24. כתוב תוכנית שתבדוק אם תא 1000H מכיל מספר שלילי. אם כן, היא תציב 1 באוגר AX. אם המספר חיובי, היא תציב 0 באוגר AX.

25. כתוב תוכנית שתעתיק מתא 1320H לתא 1321H את המספר **בערך מוחלט**. לשם כך, התוכנית תבדוק אם המספר הינו שלילי. אם כן, היא תהפוך אותו לחיובי על ידי חיסור מ-0. מספר חיובי יועתק ללא שינוי.

5

לולאות

לולאה בתוכנית

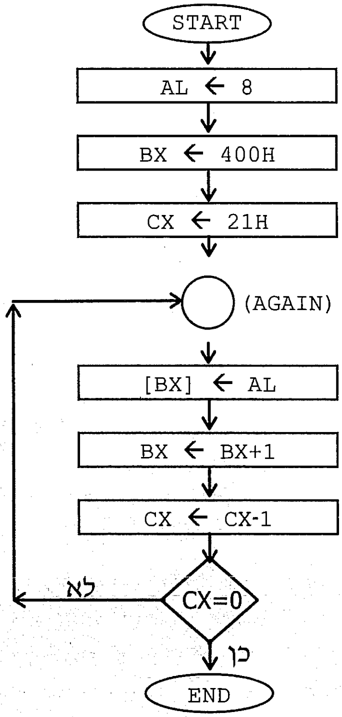
כדי לבצע לולאות באסמבלי, אין צורך בפקודות נוספות מעבר למה שלמדנו עד כה. עיצוב הלולאה הינו פעולה לוגית של המתכנת.

לולאה (loop) הינה פעולה של חזרה מספר פעמים על קטע של תוכנית. יכולה להיות לולאה שבה מספר הפעמים קבוע מראש, כמו לולאת FOR בבייסיק או פסקל, ויכולה להיות לולאה הבודקת תנאי מסוים ופועלת על פי התוצאה של בדיקה זו. כלומר, היא חוזרת על קטע התוכנית על פי קיומו, או אי-קיומו, של התנאי, בדומה ללולאות REPEAT, או WHILE, בשפת פסקל, או לולאות FOR בשפת C.

נראה דוגמה ללולאה המציבה את המספר 8 בתאי זיכרון שבכתובות 400H עד 420H.

```
DUGMA SEGMENT
    ASSUME CS:DUGMA,DS:DUGMA
BEGIN:  MOV AX,DUGMA
        MOV DS,AX
        MOV AL,8            ב-AL מוצב הנתון
        MOV BX,400H         באוגר BX מוצבת הכתובת ההתחלתית
        MOV CX,21H          אוגר CX מכיל את כמות הפעמים לחזרה
AGAIN:  MOV [BX],AL         הצבת הנתון 8 בתא שכתובתו BX
        INC BX              הוספת 1 למצביע הכתובות
        DEC CX              בכל פעם מופחת 1 ממונה הלולאה
        JNZ AGAIN           אם הוא אינו 0 - חוזרים שוב
DUGMA ENDS
END BEGIN
```

תרשים זרימה:



הסבר התוכנית:

באחד האוגרים (בדרך כלל CX, נראה בהמשך מדוע) מציבים את מספר הפעמים שהלולאה תבוצע. בתוכנית זו אנו רוצים לחזור על הצבת ערך ל-21H תאי זיכרון, מתא 400H עד תא 420H, כולל. לאחר ביצוע קטע התוכנית, שבה מוצב הערך 8 באחד התאים, מפחיתים 1 מאוגר CX, כדי לעדכן שכמות הפעמים שנותָרה לביצוע קטנה ב-1.

אם אוגר CX אינו 0, הפעולה לא הסתיימה ועדיין לא הוצב הערך 8 **לכל** התאים. לכן, יש לחזור על התהליך עבור שאר התאים.

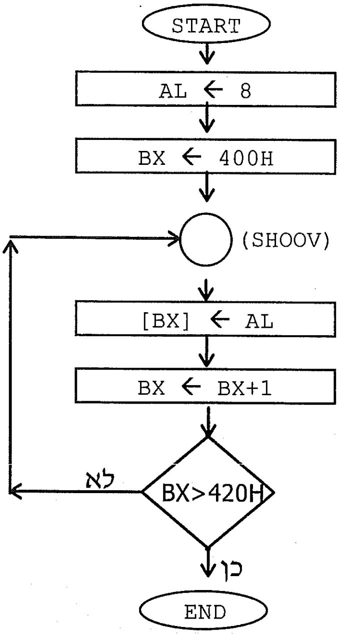
נעיין בטבלת המעקב לתוכנית זו. הטבלה מתארת את השינוי בתוכן האוגרים בכל פעם שמבצעים את הלולאה:

BX	AL	CX	הפעולה המבוצעת
400H	8	21H	אתחול (קביעת ערך התחלתי)
400H	8	21H	הצבת 8 בתא 400H
401H	8	20H	הצבת 8 בתא 401H
402H	8	1FH	הצבת 8 בתא 402H
403H	8	1EH	הצבת 8 בתא 403H
....	.....	......	.....
420H	8	0	הצבת 8 בתא 420H

את התוכנית, המציבה את הערך 8 בתאים 400H עד 420H, ניתן לכתוב בדרך שונה.

כך, לאחר הצבת 8 בתא מסוים והוספת 1 למצביע על כתובת התא, תיעשה בדיקה אם הגענו לכתובת האחרונה 420H. אם לא, נמשיך בביצוע התוכנית. אם כן, התוכנית תסתיים.

תרשים הזרימה:



התוכנית:

```
DUGMA2 SEGMENT
  ASSUME CS:DUGMA2,DS:DUGMA2
START:  MOV AX,DUGMA2
        MOV DS,AX
        MOV AL,8
        MOV BX,400H
SHOOV:  MOV [BX],AL        ; הצבת 8 בתא שכתובתו נתונה ב-BX
        INC BX             ; קידום ב-1 של מצביע הכתובות
        CMP BX,420H        ; האם הגענו לתא האחרון?
        JNG SHOOV          ; אם לא - המשך להציב
DUGMA2 ENDS
END START
```

על פי שיטה זו אין צורך במונה, אלא בודקים כל פעם אם מגיעים לתא האחרון.

אנו משתמשים בפקודה JNG SHOOV, ולא בפקודה JNE SHOOV, מכיון שלאחר שמוסיפים 1 לאוגר BX וערכו מגיע ל-420H, עדיין **לא** הוצב 8 לתא 420H. לכן, יש להמשיך בלולאה כאשר BX שווה ל-420H, ולהפסיק רק כאשר הוא גדול מ-420H, כלומר 421H.

אפשר היה לכתוב זאת גם כך:

```
CMP BX,421H
JNE SHOOV
```

אולם, מבחינת התיעוד, הרישום הקודם טוב יותר, מכיון שקל יותר להבין שבודקים מה קורה בתא 420H.

הפקודה LOOP

פקודה זו מיועדת לסייע למתכנת בכתיבת לולאות. הפקודה מחליפה שתי פקודות:

במקום זוג הפקודות:

```
DEC CX
JNZ AGAIN
```

נכתוב:

```
LOOP AGAIN
```

הפעולה המתבצעת תהיה זהה. הפקודה **LOOP** פועלת על אוגר CX **בלבד**, ולכן לא נוכל להשתמש בה כדי להחליף את צמד הפקודות הבא, למשל:

```
DEC DX
JNZ AGAIN
```

דוגמאות לביצוע לולאות

לאחר שלמדנו את הפקודה LOOP, נתרגל כתיבה והרצה של מספר תוכניות.

תרגיל 1: תוכנית שרצה כראוי

כתוב והרץ תוכנית המציבה את הערך 99H בתאי זיכרון בכתובות 2000H עד 2050H (כולל).

פתרון:

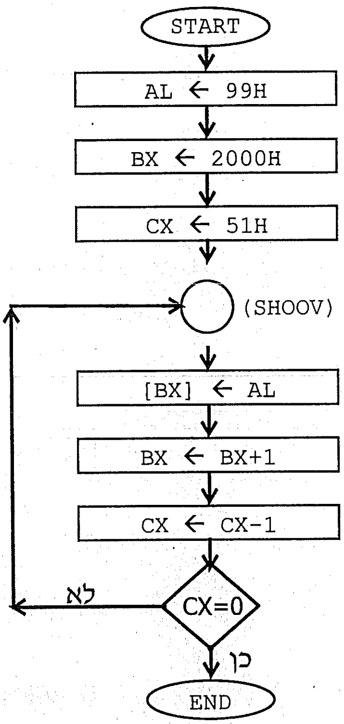
א. נכתוב את התוכנית באמצעות עורך, ונבחר עבורה את השם CODE.ASM. שים לב שאפשר לכתוב בעורך, אם רוצים כמובן, גם את ההערות שאנו כותבים בצד ימין של הפקודות. לשם כך, צריך לכתוב נקודה-פסיק (;) ולאחריו את נוסח ההערה.

התוכנית:

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV AL,99H        ; AL מכיל את הנתון
        MOV BX,2000H      ; BX מצביע על הכתובת ההתחלתית
        MOV CX,51H        ; CX משמש כמונה הלולאה
SHOOV:  MOV [BX],AL       ; מציב את הנתון בתא הזיכרון
        INC BX            ; BX מקודם להצבעה על התא הבא
        LOOP SHOOV        ; מפחית 1 מהמונה CX, אם עדיין
                          ; הוא אינו 0 - קופץ ל-shoov

CODE ENDS
END START
```

תרשים זרימה:



ב. הרצת MASM לצורך בדיקת שגיאות תחביריות ויצירת תוכנית בשפת מכונה:
MASM CODE,,;

ג. הרצת LINK ליצירת קובץ EXE : ;,,LINK CODE

ד. הרצה ובדיקה באמצעות DEBUG :

1. הפעלת DEBUG עבור התוכנית שכתבנו : DEBUG CODE.EXE

2. כתיבת הפקודה U0 ובדיקת הכתובת היחסית של הפקודה שלאחר סוף התוכנית שלנו : הכתובת הינה 12H.

3. הרצת התוכנית : G=0 12

4. בדיקה שהתוכנית ביצעה את משימותיה :

נכתוב את הפקודה : D 2000 2050

נבדוק שכל התאים הללו מכילים את הערך 99H.

5. כדי לבדוק שלא היה במקרה בתאים אלה הערך 99H, יש להציב בהם ערך אחר, למשל 0. נעשה זאת בפקודה : F 2000 2050 0

נריץ את התוכנית שוב : G=0 12. נבדוק אם כעת התאים האלה מכילים את הערך 99H. נפעיל שוב את הפקודה : D 2000 2050

6. יציאה מהתוכנה על ידי Q.

תרגיל 2: איך בודקים תוכנית שגויה

כתוב ובדוק תוכנית אסמבלי שמציבה את המספרים 0 עד 9 בתאי הזיכרון 400H עד 409H בהתאמה. כלומר, בתא 400H יוצב 0, בתא 401H יוצב 1, וכן הלאה.

פתרון:

נניח שפותר שאלה זו כתב בעורך את התוכנית **השגויה** הבאה :

```
CSEG SEGMENT
  ASSUME CS:CSEG,DS:DSEG
FIRST:  MOV AX,CSEG
        MOV DS,AX
        MOV SI,400H
        MOV CX,10
        MOV DL,0
BACK:   MOV [SI],DL
        INC SI
```

כאן נשכחה הפקודה : INC DL -->

```
        LOOP BACK
CSEG ENDS
END FIRST
```

בשל הטעות, או השגיאה, התוכנית תציב את הערך 0 **בכל** התאים 400H עד 450H. אם לא תשים לב לטעות זו ותעבור לשלב הבא:

תריץ את MASM, אשר **לא** תמצא טעות תחבירית כלשהי. לאחר מכן, תריץ את LINK אשר תתריע על כך שאין מחסנית (זו אינה שגיאה). בשלב DEBUG תריץ את התוכנית ותבדוק את תאי הזיכרון, ואז תגלה שהתוכנית לא ביצעה את משימתה כנדרש.

נניח כעת, שלא תצליח להבחין מה אינו כשורה בתוכנית. לפני שתתפוס את ראשך בייאוש מהבעיה, ואולי מהשפה הזו, כדאי שתנסה את ההוראה **T** המאפשרת לבדוק את התקדמות התוכנית, שלב אחר שלב.

נעקוב אחר הפעולות שיש לבצע:

1. עדכון אוגר IP לכתובת 0: "R IP" והזנת הערך 0.
2. כתיבת ההוראה T (אנו נחזור על פעולה זו מספר פעמים): הפקודה הראשונה שתבוצע תהיה הצבת ערך ל-AX. כעת תוצג ההוראה הבאה לביצוע: MOV DS,AX
3. כתיבת ההוראה T: ערך אוגר AX יועתק לאוגר DS.
4. כתיבת T: בתוך אוגר SI מוצב הערך 400H (כל הערכים ב-DEBUG הם בבסיס 16).
5. כתיבת T: הערך 0A (זהו הערך 10 כפי שמיוצג בבסיס 16) מוצב באוגר CX.
6. כתיבת T: באוגר DL יוצב הערך 0 (תוכן חצי-האוגר DH אינו חשוב לנו). כעת תוצג הפקודה הבאה לביצוע: MOV [SI],DL. בנוסף, יוצג ערכו של תא 400H **לפני** ביצוע פקודה זו.
7. כתיבת T: ערך האוגרים לא השתנה. פקודה זו הציבה את תוכן אוגר DL בתא זיכרון שכתובתו נתונה ב-SI. נבדוק זאת על ידי הפקודה "D 400 400". נוכל לראות שכעת ערך תא 400H הוא 0.
8. כתיבת T: אוגר SI הוגדל ב-1, וערכו כעת 401H.
9. כתיבת T: ערכו של CX יורד ל-9, ומתבצעת קפיצה לפקודה "MOV [SI],DL"
10. כתיבת T: כעת הוצב ערך אוגר DL לתוך הזיכרון על פי הכתובת הנתונה ב-SI. נבדוק את ערך התא 401H: D 401 401.

    ערך התא הוא 0 **ולא** 1 כפי שציפינו! כאן התגלתה הטעות. ערך אוגר DL, שאת ערכו אנו מציבים לתאי הזיכרון נשאר 0, ולא השתנה ל-1.

נחזור לעורך, נתקן את התוכנית על ידי הוספת השורה החסרה, נריץ שוב את MASM ואת LINK, ולבסוף נריץ את התוכנית ונבדוק שוב באמצעות DEBUG.

טעות נוספת אפשרית:

נניח שבתוכנית נכתבה השורה השגויה MOV SI,400 במקום השורה: MOV SI,400H. אם הכותב לא הבחין בכך בשלב העריכה, הוא בוודאי יצליח להבחין בכך בשלב DEBUG.

שים לב לכך שהפקודה "השתנתה" והיא עכשיו "MOV SI,190", מכיון שהערך העשרוני 400 תורגם לערך ההקסדצימלי 190H.

תוכל לראות באמצעות הפקודה T, שלאחר ביצוע הפקודה הזו לא הוצב בתא 400H הערך 0 כנדרש, וכך לגלות מהי הטעות.

טעות נוספת אפשרית:

במקום לרשום את הפקודה INC SI, נרשמה בטעות הפקודה INC BX.

כתוצאה מכך, תגלה בעת הרצת התוכנית שרק בתא 400H הוצב הערך 0, ואילו שאר התאים לא קיבלו את הערכים הרצויים.

תוכל לגלות זאת על ידי מעקב אחר ביצוע פקודות התוכנית באמצעות הפקודה T.

תרגיל 3: הצבת נתונים בבלוק (קטע) זיכרון

כתוב תוכנית שמציבה את הערכים 0 ו-1 לסירוגין, החל מתא זיכרון בכתובת 700H ועד וכולל תא הזיכרון בכתובת 718H.

פתרון:

1. כתוב את התוכנית בעורך. אחד מהפתרונות האפשריים:

```
CODE_SEG SEGMENT
        ASSUME CS:CODE_SEG,DS:CODE_SEG
BEGINING: MOV AX,CODE_SEG
          MOV DS,AX
          MOV DL,0
          MOV DH,1
          MOV SI,700H
CONTINUE: MOV [SI],DL              מציב 0 בתאים
          INC SI
          CMP SI,718H              האם הגענו לסוף?
          JG END_PROG              אם כן - סיים
          MOV [SI],DH              אם לא - מציב 1
          INC SI
          JMP CONTINUE             המשך בלולאה
END_PROG: NOP
CODE_SEG ENDS
END BEGINING
```

2. הרצה של MASM.
3. הרצה של LINK.

4. הרצה ובדיקה באמצעות DEBUG :

א. הרצת התוכנית : G=0 1B

ב. בדיקת תוכן התאים : D 700 718

ג. שינוי תוכן תאים אלה לערכים אחרים, נניח ל-66 : F 700 718 66

ד. הרצה חוזרת : G=0 1B

ה. בדיקת תוכן התאים : D 700 718. התאים צריכים להכיל ערכי 0 ו-1 לסירוגין. אם התוכנית פעלה כראוי - המשימה הסתיימה. אם לא - צריך לעבור להרצה צעד אחר צעד, למצוא את השגיאה, לתקן, להריץ שוב ולבדוק עד לקבלת התשובה הנכונה.

ו. יציאה מהתוכנה DEBUG באמצעות הפקודה Q.

תרגיל 4: העתקת בלוקים

עליך לכתוב תוכנית שמעתיקה את הערכים מהתאים שבכתובות 700H עד 720H (כולל), לתאים 800H עד 820H בהתאמה. כלומר, תוכן תא 700H יועתק לתא 800H וכך הלאה.

שלבי הפתרון:

1. כתיבת התוכנית בעורך :

```
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
BEGIN: MOV AX,CODE
       MOV DS,AX
       MOV SI,700H        מצביע על התא הראשון
       MOV DI,800H        תא היעד הראשון
       MOV CX,21H
CONT:  MOV BL,[SI]        מעתיק מתא המקור
       MOV [DI],BL        ומעביר לתא היעד
       INC SI             מקדם את המצביעים
       INC DI
       LOOP CONT          חוזר על התהליך
CODE ENDS
END BEGIN
```

2. שלב MASM, בדיקה שאין שגיאות.

3. שלב LINK, בדיקה שאין שגיאות (פרט לאי-קיום מחסנית).

4. שלב DEBUG:

א. הרצת התוכנית: G=0 16

ב. בדיקת התוצאה: D 700 720 ולאחר מכן: D 800 820

לאחר הרצת התוכנית, הערכים של קטע הזיכרון 700H-720H יהיו זהים לערכי התאים 800H-820H בהתאמה.

ג. כדי לוודא שלא במקרה שני קטעי הזיכרון זהים, נבצע:

- שינוי ערכי קטע המקור: F 700 720 66
- בדיקה שקטע המקור וקטע היעד אינם זהים:
  - D 700 720 - כל התאים בעלי ערך 66H.
  - D 800 820 - ערכים שונים.
- הרצת התוכנית שוב: G=0 16
- בדיקה שכעת הקטעים שוב זהים:
  - D 700 720 - כל הערכים מכילים 66H.
  - D 800 820 - כעת גם אלה מכילים 66H.

ד. יציאה מהתוכנה על ידי Q.

תרגיל 5: ספירת תאי זיכרון המתאימים לקריטריון

כתוב תוכנית המציבה באוגר AL את מספר הערכים השווים ל-11H, מתוך ערכי התאים בכתובות 140H-180H.

פתרון:

1. כתיבת התוכנית בעורך (mama.asm):

```
MAMA SEGMENT
  ASSUME CS:MAMA,DS:MAMA
PAPA:   MOV AX,MAMA
        MOV DS,AX
        MOV AL,0          AL משמש כמונה
        MOV BX,140H       מצביע כתובות התאים
        MOV CX,41H        מונה הלולאה
CHECK:  MOV AH,[BX]       מעתיק את תוכן התא
        CMP AH,11H        משווה עם הערך 11H
        JNE CONT          אם לא שווים - קופץ
        INC AL            לא קפץ (ערך התא שווה 11H) - מונה עוד 1 כזה
CONT:   INC BX            מקדם את המצביע
        LOOP CHECK        מפחית 1 מ-CX, ואם אינו 0 - קופץ
MAMA ENDS
END PAPA
```

2. הפעלת MASM.

3. יצירת קובץ מוכן להרצה באמצעות LINK.

4. בדיקה באמצעות DEBUG:

א. הצבת ערכים בתאי הזיכרון 140H-180H.

נציב ערכי 0 בכל התאים הללו, ולאחר מכן נציב רק ב-3 תאים ערך 11H:

❖ נבדוק את המספר הרשום בפקודה הראשונה (במקום המילה MAMA). **נניח** שהמספר הינו 1F68, ועל כן נכתוב: F 1F68:140 180 0 (יגרום להצבת 0).

❖ נציב בשלושה תאים את הערך 11H על ידי: F 1F68:150 152 11.

ב. הרצת התוכנית: G=0 19. בדיקה שאוגר AL מכיל 3.

ג. יציאה מהתוכנה על ידי Q.

תרגיל 6: בדיקת ערכי תאים

כתוב תוכנית שבודקת את הערכים של בלוק (קטע) של תאי זיכרון בכתובות 770H-780H:

❖ אם כל ערכי התאים שונים מ-0, הצב 1 באוגר DX.

❖ אם ערך אחד (או יותר) מהתאים שווה ל-0, הצב 0 באוגר DX.

הפתרון הבא הינו אחד מתוך רבים. שלבי הפתרון המוצע: ערכו של DX יוצב כ-1 בתחילת התוכנית. רק אם יהיה תא אחר שערכו 0, יוחלף ערך DX ל-0.

כתיבת התוכנית (program.asm):

```
PROGRAM SEGMENT
  ASSUME CS:PROGRAM,DS:PROGRAM
BEGIN:  MOV AX,PROGRAM
        MOV DS,AX
        MOV DX,1
        MOV BX,770H
        MOV CX,11H
BDOK:   MOV AH,[BX]
        CMP AH,0
        JE CHANGE          אם נמצא תא שערכו 0 - קפוץ לתווית CHANGE
        INC BX
        LOOP BDOK          חזור על התהליך
        JMP ZEHU           אם הגענו לכאן - כל התאים שונים מ-0, לכן קפוץ לסוף
CHANGE: MOV DX,0           אם נמצא תא שערכו 0 - שנה את ערך DX ל-0
ZEHU:   NOP
PROGRAM ENDS
END BEGIN
```

ב. הרצת MASM ו-LINK.

ג. הרצת התוכנית לבדיקה ב-DEBUG:

- הצב בכל תאי הזיכרון הרלוונטים ערך השונה מ-0: F 770 780 6
- הרץ את התוכנית: G=0 1F. בדוק שאוגר DX מכיל 1.
- הצב 0 באחד מהתאים: E 772 0
- הרץ שוב: G=0 1F. בדוק אם אמנם ערך DX השתנה ל-0.
- סיים בפקודה Q.

תרגיל 7: בדיקת מיון בלוק נתונים

הערה: "בלוק" הוא אזור רציף בזיכרון הכולל מספר תאים.

כתוב את התוכנית הבאה בשפת אסמבלי, הרץ ובדוק.

נתון בלוק זיכרון החל מכתובת 800H ועד 810H. הבלוק כולל 11H ערכים שונים. בדוק אם הערכים ממוינים (מסודרים) בסדר עולה. אם כן - הצב FFH באוגר AX; אם לא - הצב 0 באוגר AX.

פתרון:

נבדוק ערך של כל תא ביחס לערך התא הבא לאחריו. כלומר, נשווה כל שני תאים עוקבים. אם נמצא שהתא הראשון אינו קטן יותר מזה שבא אחריו, נסיק מכך שהנתונים אינם ממוינים כנדרש, ונציב 0 באוגר AX. רק כאשר כל אחד מהתאים קטן מתא שאחריו, נוכל להסיק שהנתונים ממוינים בסדר עולה ונציב FFH באוגר AX.

כתיבת התוכנית (code07.asm):

`CODE SEGMENT`	
`  ASSUME CS:CODE,DS:CODE`	
`START:  MOV AX,CODE`	
`        MOV DS,AX`	
`        MOV SI,800H`	עבור 11H ערכים
`        MOV CX,10H`	דרושות 10H השוואות
`CHECK:  MOV DL,[SI]`	
`        CMP DL,[SI+1]`	השווה בין תא מסוים לתא הבא
`        JNL BAD`	אם אינו קטן - קופץ ל-BAD
`        INC SI`	
`        LOOP CHECK`	חזור על הבדיקה בכל התאים
`        JMP GOOD`	אם הגעת לכאן - הכל תקין
`BAD:    MOV AX,0`	הצב 0 באוגר AX
`        JMP SIUM`	
`GOOD:   MOV AX,0FFH`	הצב FFH באוגר AX
`SIUM:   NOP`	
`CODE ENDS`	
`END START`	

לאחר הרצת MASM ו-LINK, הרץ ובדוק את התוכנית ב-DEBUG:

1. בדוק מהו הערך שכתוב בפקודה הראשונה במקום המילה CODE.

   נניח, לצורך ההסבר שהתוכנית הוטענה לתא זיכרון 1234H ולכן, הערך שמופיע במקום המילה CODE הינו 1234H.

   בדוק מהי הכתובת של הפקודה שלאחר התוכנית. לשם כך, עליך להפעיל את הפקודה U0 ולאחר מכן את הפקודה U, כדי לראות את המשך התוכנית. תוכל להיווכח שהכתובת הינה 22.

2. מלא את הקטע 800H-810H בערכים ממוינים בסדר עולה. בהנחה שערך הבסיס של התוכנית הוא 1234H (ראה סעיף 1), כתוב את הפקודה הבאה, אשר תמלא את הערכים 0 עד 16H החל מתא זיכרון 800H:

   F 1234:800 810 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

3. הרץ את התוכנית על ידי: G=0 22.

4. תוכל לראות שערך אוגר AL (החצי הימני של אוגר AX) הינו FFH.

5. שנה את אחד הנתונים, כך שסדר הנתונים ישתנה. קבע למשל את הערך 4 בתא 805 במקום 6 שהיה בו בתחילה: E 805 4. הדבר יגרום לשיבוש המיון.

6. הרץ שוב: G=0 22. כעת ערכו של AL הינו 0.

7. סיים בפקודה Q.

הערה חשובה!

ייתכן שאתה מבולבל מעט מצורת הכתיבה של הפקודה D. נבהיר זאת.

לעיתים יש צורך לרשום את הפקודה עם הבסיס, כמו למשל: D 1234:800 810

ולפעמים ניתן לרשום רק: D 800 810

מתי רושמים כך ומתי אחרת?

אם רוצים לטפל בזיכרון (להציג, לשנות וכדומה) **לפני** הרצת התוכנית, חובה לבדוק מהו **הבסיס** (הערך הרשום בפקודה הראשונה במקום המילה CODE), ולאחר מכן לכתוב פקודה הכוללת את **הבסיס וההיסט**, כמו למשל:

E 10D6:400 99

אם כבר הרצת את התוכנית, מיותר לכתוב את הבסיס, ואפשר לכתוב את ההיסט בלבד, לדוגמה: E 400 99.

הסבר: לאחר שהרצנו את התוכנית, הבסיס **התעדכן** ולכן אין צורך לרשום אותו בפקודה. בנושא זה נדון בהרחבה בהמשך.

תרגיל 8: סיכום ערכים בזיכרון

כתוב תוכנית המחשבת את סכום הערכים של 5 תאי הזיכרון המתחילים בכתובת 1000H, ומציבה את התוצאה בתא שכתובתו 1200H.

השיטה: נאפס אוגר מסוים ולאחר מכן, נוסיף את ערכו של כל תא לתוך אוגר זה. לבסוף, נעתיק את ערכו של האוגר לתא 1200H.

תהליך זה דומה למקובל בשפות עיליות: למעשה, כתיבת תוכנית באסמבלי דומה מבחינה לוגית לשפה עילית ולעיתים היא אף ברורה יותר!

התוכנית (code08.asm):

```
CODE SEGMENT
  ASSUME CS: CODE,DS: CODE
START:    MOV AX,CODE
          MOV DS,AX
          MOV SI,1000H
          MOV CX,5
          MOV AL,0            באוגר AL נחשב את הסכום
CONT:     ADD AL,[SI]         נחבר ל-AL את ערך התא
          INC SI
          LOOP CONT           נמשיך עבור 5 התאים
          MOV SI,1200H
          MOV [SI],AL         נציב את התוצאה
CODE ENDS
END START
```

לאחר הרצת MASM ו-LINK נריץ ונבדוק באמצעות DEBUG:

1. נכתוב את הפקודה U0 ונראה מהי הכתובת של הפקודה לאחר התוכנית. נקבל את המספר 17. נראה מהי כתובת הבסיס שבמקום המילה CODE בפקודה הראשונה.
2. כדי לעקוב אחרי הביצוע של התוכנית נציב בתאים 1000H-1004H ערכים נוחים עבורנו לחישוב. למשל, נציב את הערך 1. נעשה זאת באמצעות ההוראה F. באמצעות ההוראה D נוודא שהערכים נכונים.
3. נריץ את התוכנית: G=0 17.
4. נבדוק מהי התוצאה בתא 1200H (באמצעות D) התוצאה צריכה להיות 5.
5. ננסה לבדוק חיבור של ערכים נוספים, כדי לוודא פעולה תקינה של התוכנית.

שים לב: אנו מתעלמים מאפשרות שבה התוצאה תהיה גדולה מדי עבור האוגר AL. מצב כזה נקרא גלישה, נעסוק בכך בהמשך.

תרגיל 9: הזזת ערכים בזיכרון

מוטלת עליך משימה לכתוב תוכנית אסמבלי, שתזיז **ימינה** בתא אחד את הערכים של שישה תאים החל מכתובת 200H, ותאפס את התא השמאלי.

לדוגמה, אם ערכי התאים 200H עד 205H (קרא משמאל לימין) הם:

42 54 67 63 57 19

לאחר הרצת התוכנית נקבל:

00 42 54 67 63 57 19

הפתרון:

התחל בתא הימני ביותר, תא 205H. העתק אותו לתא הבא 206H. אחר כך, העתק את תא 204H לתא 205H, וכך הלאה. לבסוף הצב 0 בתא 200H. שים לב, שאם היית מתחיל את התהליך בתא 200H, היית מוחק את תוכן התאים!

התוכנית (code09.asm):

```
 CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
 START:   MOV AX,CODE
          MOV DS,AX
          MOV SI,205H
          MOV CX,6
CONT:     MOV AL,[SI]
          MOV [SI+1],AL
          DEC SI
          LOOP CONT
          MOV SI,200H
          MOV AL,0
          MOV [SI],AL
CODE ENDS
 END START
```

לאחר הרצת MASM ו-LINK, הרץ את התוכנית ובדוק באמצעות DEBUG:

1. בדוק את כתובת סיום התוכנית (תקבל: 1A). בדוק את ערך הבסיס (נניח שקיבלת 0FC9).
2. מלא נתונים בתאים: F 0FC9:200 205 1,2,3,4,5,6
3. הרץ את התוכנית: G=0 1A
4. בדוק את התוצאה: D 200 206
5. בדוק עבור נתונים נוספים.
6. צא מהתוכנית על ידי Q.

תרגיל 10: מציאת הערך הגבוה ביותר

כתוב תוכנית שתמצא את המספר הגדול ביותר בין הערכים של התאים שבקטע הזיכרון 450H-460H, ותציב את התוצאה באוגר DL. התייחס אל המספרים כמסומנים!

האלגוריתם (תהליך הפתרון):

אוגר DL נבחר כדי להכיל את המספר הגדול ביותר. נתחיל בכך שנכניס בו את ערך התא הראשון (450H), אשר כעת הוא הערך הגדול ביותר.

לאחר מכן, נשווה כל אחד מהתאים שבקטע הזיכרון אל אוגר DL, עד שנגיע לתא האחרון:

- ❖ אם מצאנו שערך התא גדול מזה שנמצא ב-DL, נציב אותו ב-DL. כלומר, נעדכן את DL בערך הגדול יותר.
- ❖ אם ערך התא אינו גדול מ-DL (כלומר, קטן או שווה), נמשיך הלאה לתא הבא.

התוכנית שלפניך (code10.asm), הינה אחת מאפשרויות רבות לפתרון הבעיה:

`CODE SEGMENT`	
`  ASSUME CS:CODE ,DS:CODE`	
`START:  MOV AX,CODE`	
`        MOV DS,AX`	
`        MOV BX,450H`	
`        MOV DL,[BX]`	ב-DL מוצב ערך התא הראשון
`CONT:   INC BX`	BX מתקדם ומצביע על התא הבא
`        CMP BX,460H`	
`        JG FINISH`	אם BX הגיע לתא האחרון - סיים
`        MOV AL,[BX]`	ערך התא מוצב ב-AL
`        CMP AL,DL`	
`        JNG CONT`	אם AL אינו גדול - המשך
`        MOV DL,AL`	אם הגעת לכאן - AL מכיל ערך גדול יותר ויש להעביר לאוגר DL את הערך הגדול יותר.
`        JMP CONT`	המשך בתהליך
`FINISH: NOP`	
`CODE ENDS`	
`END START`	

לאחר הרצת MASM ו-LINK, נריץ את התוכנית ונבדוק אותה באמצעות DEBUG:

1. נכתוב U0 כדי לראות את הכתובת של הפקודה שלאחר סיום התוכנית: נקבל 1C. כך גם נראה את כתובת הבסיס. כזכור, זהו המספר הרשום במקום המילה CODE בפקודה הראשונה. נניח, לצורך הדוגמה, שנראה את הערך 1076.
2. נמלא את תאי הזיכרון בכתובות 450H-460H בערך 0: F 1076:450 460 0
3. נציב שני מספרים: 5 בתא 455H ו-9 בתא 456H:

```
E 1076:455 5
E 1076:456 9
```

4. נבדוק מה מכילים התאים בקטע הזיכרון שקבענו: D 1076:450 460

נראה שני תאים המכילים ערכים 5 ו-9, ושאר התאים מכילים את הערך 0.

5. נריץ את התוכנית: G=0 1C

נוכל לראות שאוגר DL (החצי הימני של אוגר DX) מכיל את הערך 9. זהו הערך הגדול ביותר מבין הערכים בתאי הזיכרון שקבענו לחיפוש.

6. נציב ערך שונה ונבדוק שוב. נציב את הערך 77H בתא 459H: E 459 77

תזכורת: לאחר הרצת התוכנית אין צורך לרשום את כתובת הבסיס בפקודה E, כמו למשל: E 1076:459 77

7. נריץ שוב את התוכנית: G=0 1C. נראה עכשיו שאוגר DL מכיל את הערך 77H, והתוכנית פועלת כנדרש.

8. כעת, הצב את הערך 85H באחד התאים שבתחום 450H-460H, ובדוק אם קיבלת באוגר DL את המספר הזה. נסה להסביר את התוצאה שקיבלת!

כרגיל, אם אינך יודע את התשובה, רשום אותה בצד ונסה לענות עליה עם התקדמותך בלימוד השפה.

פתרון נוסף:

פתרון שונה לתרגיל זה נכתב על ידי אחד מתלמידי.

כתוב את התוכנית, הרץ ובדוק. אם התוכנית אינה פועלת לדעתך כנדרש, תקן את השגיאות והרץ שוב.

```
TALMID SEGMENT
ASSUME CS:TALMID,DS:TALMID
BEGIN: MOV AX,TALMID
       MOV DS,AX
       MOV DI,450
       MOV SI,451
BACK:  MOV CL,[SI]
       MOV CH,[DI]
       CMP CL,CH
       JNG SMALL
       MOV CH,CL
SMALL: INC SI
       CMP SI,461
       JNE BACK
       MOV DL,CH
TALMID ENDS
END BEGIN
```

תרגיל 11: חיפוש כתובת על פי קריטריון

כתוב תוכנית המציבה באוגר BX את הכתובת הראשונה שבה נמצא הערך 77H. התוכנית תתחיל את חיפוש הערך 77H בתא 800H ותמשיך עד שתמצא אותו (אם תמצא).

פתרון:

אוגר BX יכיל בהתחלה את כתובת ההתחלה 800H. נבדוק את הערך בכל תא:

- אם הערך אינו 77H, נמשיך לתא הבא.
- אם הערך הינו 77H, נסיים ואז יכיל BX את כתובת התא המכיל 77H.

התוכנית:

```
CSEG SEGMENT
  ASSUME CS:CSEG,DS:CSEG
BEGIN: MOV AX,CSEG
          MOV DS,AX
          MOV BX,800H
          MOV DH,77H
CONT:  CMP [BX],DH
          JE FINISH
          INC BX
          JMP CONT
FINISH: NOP
CSEG ENDS
END BEGIN
```

לאחר כתיבת התוכנית והרצת MASM ו-LINK, נריץ את DEBUG לבדיקה:

1. נכתוב U0 לקבלת התוכנית. נקפיד להבחין בשני דברים:
   - הכתובת של הפקודה לאחר התוכנית (בתוכנית זו: 12H).
   - הכתובת שנקבעה בכתובת הבסיס (הכתובת שיקבל אוגר DS), זהו המספר הרשום בפקודה הראשונה במקום המילה CSEG.
   - נניח, לצורך הסבר תרגיל זה, שהכתובת הינה 0FC9H.
2. נכניס את הערך 77H לאחד התאים, נניח לתא 899H: E 0FC9:899 77

   כלומר, לתא בכתובת 899H **יחסית** לבסיס של כתובות הנתונים, יוצב הערך 77H.
3. נריץ את התוכנית: G=0 12. נוכל לראות שערך אוגר BX הינו 899H. אם ערכו אחר, ייתכן שהערך 77H נמצא בתא בכתובת **קודמת** יותר, ולכן החיפוש ייעצר בכתובת זו. במקרה כזה, בדוק שערך תא זה הוא אכן 77H.

שים לב, במקרה שלא קיים הערך 77H, התוכנית עלולה לגרום לביצוע לולאה אינסופית, ולתקיעת המחשב.

שאלה לדיון: האם התוכנית סורקת את **כל** זיכרון המחשב כדי למצוא את הערך 77H?

תרגיל 12: בדיקת סכום ערכים בבלוק נתונים

כתוב תוכנית באסמבלי, המחשבת את סכום ערכי התאים בכתובות 222H-233H.

- אם התוצאה גדולה מ-29H, התוכנית תסתיים.
- אם לא, התוצאה תועתק לתא בכתובת 244H.

פתרון:

התוכנית (code12.asm):

CODE SEGMENT	
ASSUME CS:CODE,DS:CODE	
START: MOV AX,CODE	
MOV DS,AX	
MOV BX,224H	BX מצביע על הכתובות
MOV DI,244H	DI מצביע על כתובת 244H
MOV CL,0	CL נבחר כצובר את הסכום
CALC: ADD CL,[BX]	מוסיף ל-CL את תוכן התא
INC BX	BX מצביע על התא הבא
CMP BX,233H	אם לא עברנו את התא האחרון
JNG CALC	נחזור לבצע את הצבירה
CMP CL,29H	לכאן הגענו לאחר סיום צבירת הסכום
JG FINISH	אם הסכום גדול מ-29, נסיים
MOV [DI],CL	אם נמצאים כאן, הסכום אינו גדול מ-29H, ולכן נציב את התוצאה בתא 244H
FINISH: NOP	
CODE ENDS	
END START	

לאחר כתיבת התוכנית בעורך, הרצת MASM ו-LINK, נבדוק את התוכנית באמצעות DEBUG:

1. נכתוב U0 ונבדוק את הכתובת של הפקודה שלאחר סיום התוכנית (קיבלנו 1EH), נבדוק את כתובת הבסיס (נניח שהכתובת היא 879H).
2. נציב ערכים בתאים הנבדקים:
   - תחילה נציב 0 בכל התאים: F 879:222 233 0
   - לאחר מכן נציב ערכים שסכומם קטן מ-29H: E 879:225 22 4

   פקודה זו תציב בתא 225H את הערך 22H ובתא 226H את הערך 4. סכום כל התאים הינו 26H.
3. נריץ את התוכנית: G=0 1E
4. נבדוק שהסכום הוצב בתא 244H: D 244 244. עלינו לקבל את הערך 26H.

5. האם בזאת הסתיימה בדיקת התוכנית? מובן שלא. יש לבדוק את תוצאות התוכנית כאשר סכום התאים עולה על 29H. לפי תנאי הבעיה התוכנית צריכה להסתיים, מבלי להציב את הסכום בתא 244H.

כדי לבדוק זאת, נציב ערכים כאלה, שסכומם יהיה גדול מ-29H. למשל, נכתוב את הוראות DEBUG הבאות: F 222 233 0 ולאחר מכן: E 227 28 1 1

נבדוק את הערכים בתאים אלה: D 222 233

עכשיו עלינו **לקבל** את סכום התאים: 2AH (28+1+1=2A).

6. נריץ שוב את התוכנית: G=0 1E.

7. נראה כעת את תוכן תא 244H: D 244 244

נוודא שערך התא **אינו** 2AH.

8. נצא מהתוכנית בפקודה Q.

פתרון נוסף:

כפי שכבר אמרנו בתרגיל קודם, לבעיה אחת יכול להיות יותר מפתרון תכנות אחד.

```
CODE SEGMENT
ASSUME CS:CODE , DS:CODE
BEGIN: MOV AX,CODE
        MOV DS,AX
        MOV BX,222H        מצביע על הכתובת הראשונה
        MOV CX,12H         מונה לולאה (12H תאים)
        MOV DL,0           צובר את סכום ערכי התאים
BACK:   ADD DL,[BX]
        CMP DL,29H
        JG SOFSOF          התוצאה גדולה מ-29H, סיים
        INC BX
        LOOP BACK          חזרה על התהליך 12H פעם
        MOV BX,244H
        MOV [BX],DL
SOFSOF: NOP
CODE ENDS
END BEGIN
```

הערה: גם במקרה זה לא בדקנו האם האוגר DL יכול להכיל את התוצאה של הסכום. למשל, אם ערכו של כל תא הוא 70H, קל להבין ש- 12H * 71H (הכפלה בכמות התאים) תיתן ערך שאינו נכנס בגודל של DL (8 סיביות). מה לדעתך יקרה במקרה כזה?

תרגיל 13: חיפוש מספר בזיכרון

עליך לכתוב תוכנית שתחפש את המספר 70H, ב-100H תאי זיכרון החל מכתובת 800H.

- ❖ אם המספר נמצא לפחות פעם אחת - התוכנית מסתיימת.
- ❖ אם המספר אינו נמצא - היא מציבה את הערך 70H בתא זיכרון שכתובתו 990H.

פתרון:

תחילה נכתוב את התוכנית לפתרון הבעיה (code13.asm):

`CODE SEGMENT`	
`   ASSUME CS:CODE,DS:CODE`	
`START: MOV AX,CODE`	
`       MOV DS,AX`	
`       MOV SI,800H`	מצביע על כתובת ההתחלה
`       MOV CX,100H`	מונה הלולאה
`       MOV DI,990H`	מצביע על תא 990H
`       MOV AH,70H`	הערך שיוצב בתא 990H
`CONT:  CMP AH,[SI]`	משווה בין ערך התא ל-70H
`       JE FINISH`	אם נמצא - סיים
`       INC SI`	
`       LOOP CONT`	חזרה על התהליך 100H פעם
`       MOV [DI],AH`	אם הגענו לכאן - נבדקו כל התאים, לא נמצא הערך 70H ולכן ערך זה מוצב כעת בתא 990H
`FINISH: NOP`	
`CODE ENDS`	
`END START`	

לאחר כתיבת התוכנית, נריץ MASM ו-LINK. נריץ DEBUG ונבצע את הפעולות הבאות:

1. איפוס ערכי תאי הזיכרון 800H-900H.
2. הרצת התוכנית ובדיקה שערך תא 990H מכיל את הערך 70H.
3. כעת, יש לבדוק את המצב השני, כלומר הימצאות הערך 70H באחד מהתאים. נפעל כך:
   - ❖ נציב ערך אחר (למשל 22H) בתא 990H.
   - ❖ נציב באחד מהתאים בכתובות 800H-900H את הערך 70H.
   - ❖ נריץ את התוכנית, ונבדוק שבתא 990H הערך לא שונה.
4. נצא מתוכנת DEBUG.

תרגיל 14: חיפוש ערכים על פי קריטריון

כתוב תוכנית הסופרת כמה תאי זיכרון מכילים ערך הגדול מ-20H.

הבדיקה תיעשה בתאים שכתובתם בתחום 600H עד 700H, והתוצאה (מספר התאים) תירשם בתא 800H.

פתרון:

נבדוק ערך של כל תא: אם הוא גדול מ-20H, נוסיף 1 למונה (צריך לאפס אותו בתחילת העבודה) ואם אינו גדול, עוברים לתא הבא.

לאחר בדיקת כל התאים המבוקשים, יכיל המונה את מספר התאים שערכם גדול מ-20H. ערך זה יוצב בתא 800H.

התוכנית (code14.asm):

PROG SEGMENT	
ASSUME CS:PROG,DS:PROG	
FIRST: MOV AX,PROG	
MOV DS,AX	
MOV SI,600H	SI נבחר כמצביע על הכתובות
MOV CX,101H	מונה הלולאה - 101H תאים
MOV DI,800H	DI מצביע על תא 800H
MOV AH,20H	AH מכיל את הערך 20H
MOV AL,0	AL נבחר לשמש כמונה
AGAIN: CMP [SI],AH	השוואה בין התא לערך 20H
JLE SMALL	אם התא מכיל ערך קטן או שווה - קופץ לתווית בשם SMALL
INC AL	אם לא קפץ והגיע לכאן, התא מכיל ערך גדול יותר מ-20H ולכן מוסיפים 1 למונה
SMALL: INC SI	מצביע על התא הבא
LOOP AGAIN	חוזר על התהליך 101H פעמים
MOV [DI],AL	לאחר סיום בדיקת כל התאים - מציב את ערך המונה בתא 800H
PROG ENDS	
END FIRST	

לבדיקת התוכנית ב-DEBUG נפעל כך:

1. נציב בכל תאי הזיכרון הרלוונטיים, ערך שאינו גדול מ-20H (למשל 20H).
2. נציב במספר תאים (נניח 3) ערך גדול מ-20H.
3. נריץ ונבדוק את ערך תא 800H. ערכו צריך להכיל את כמות התאים שהצבנו בהם ערך גדול מ-20H.
4. ניתן לבדוק את התוכנית בערכים נוספים.

תרגיל 15: בדיקת תנאים בזיכרון

כתוב תוכנית שבודקת אם **כל** תאי הזיכרון בתחום 1200H-1300H מכילים ערך גדול מ-23H.

אם כן - הצב את הערך 15H בתא 1500H, אם לא - הצב את הערך 55H בתא 1500H.

פתרון:

אם יימצא תא כלשהו המכיל ערך שאינו גדול מ-23H, ניתן לדעת שלא כל התאים גדולים מ-23H, ולכן יוצב הערך 55H בתא 1500H. רק אם כל התאים נבדקו, ובכולם ערך גדול מ-23H, יוצב 15H בתא 1500H.

התוכנית (code15.asm):

CODE SEGMENT	
ASSUME CS:CODE,DS:CODE	
START: MOV AX,CODE	
MOV DS,AX	
MOV SI,1200H	מצביע על כתובות התאים
MOV BX,1500H	מצביע על כתובת תא 1500H
MOV CX,101H	מונה הלולאה שתתבצע 101Hפעם
MOV AH,15H	משווה בין ערך הזיכרון ל-23H
MOV AL,55H	
CONT: MOV DL,[SI]	
CMP DL,23H	
JNG SMALL	אם אינו גדול - קופץ לתווית SMALL, שבה התוכנית תציב את הערך 55H בתא 1500H
INC SI	אם לא קפץ - מצביע על התא הבא
LOOP CONT	וחוזר להמשך התהליך
MOV [BX],AH	אם הגיע לכאן ולא דילג לתווית SMALL, הרי שכל התאים מכילים ערכים מעל 23H, לכן מוצב 15H בתא 1500H
JMP ZEHU_ZE	מדלגים על הפקודה שתבוצע במקרה של תא קטן מ-23H
SMALL: MOV [BX],AL	לכאן מגיעים כאשר תא מכיל ערך קטן מ-23H
ZEHU_ZE: NOP	
CODE ENDS	
END START	

באמצעות DEBUG נבדוק את שתי האפשרויות:

1. כאשר כל התאים מכילים ערך גדול מ-23H, ואז יוצב 15H בתא 1500H.
2. כאשר תא אחד, או יותר, אינם גדולים מ-23H, יוצב בתא 1500H הערך 55H.

תרגיל 16: בדיקת תנאים בזיכרון

מוטלת עליך המשימה לכתוב תוכנית בשפת אסמבלי, אשר תבדוק אם כל תאי הזיכרון בכתובות 600H-650H מכילים את הערך 1.

אם כן - התוכנית תציב 0 באוגר AL, אחרת - היא תציב FFH באוגר זה.

פתרון:

אם נמצא תא שערכו אינו 1 - יוצב FFH באוגר AL והתוכנית תסתיים. אם כל התאים מכילים 1 - יוצב 0 באוגר AL והתוכנית תסתיים.

התוכנית שלפניך הינה אחד מהפתרונות האפשריים. כאמור, לכל תוכנית קיימים גם פתרונות טובים נוספים.

התוכנית:

```
CSEG SEGMENT
   ASSUME CS:CSEG,DS:CSEG
BEGINING:  MOV AX,CSEG
           MOV DS,AX
           MOV SI,600H
           MOV CX,51H
AGAIN:     MOV DL,[SI]
           CMP DL,1
           JNE SORRY
           INC SI
           LOOP AGAIN
           MOV AL,0
           JMP FINISH
SORRY:     MOV AL,0FFH
FINISH:    NOP
CSEG ENDS
END BEGINING
```

הפעולות שיבוצעו באמצעות תוכנת DEBUG:

1. נכתוב U0 ונבדוק מהי כתובת הבסיס שתוצב באוגר DS. נניח שהכתובת בתוכנית זו היא 989H.

   נראה מהי כתובת הפקודה שלאחר התוכנית שלנו, ונקבל: 1DH

2. נציב 1 בכל התאים שבתחום הכתובות: 600H-650H   F 989:600 650 1

3. נריץ את התוכנית ונבדוק שאוגר AL מכיל 0:   G=0 1D

4. נשנה רק את אחד מהתאים לערך שונה מ-1:   E 620 2

5. נריץ ונבדוק שכעת ערך אוגר AL הינו FFH: G=0 1D

6. נסיים בפקודה Q.

תרגיל 17: השוואת בלוקים

כתוב תוכנית שתבדוק אם תאי הזיכרון בכתובות 800H-820H מכילים ערכים זהים לערכים שבתאי זיכרון בכתובות 900H-920H **בהתאמה**.

אם הם שווים, הצב FFH בתא 500H. אם לא - הצב 0 בתא זה.

פתרון:

נשווה כל תא לתא המקביל אליו בהתאמה: תא 800H אל תא 900H, וכך הלאה. אם נמצא זוג תאים שערכיהם אינם זהים, נציב 0 בתא 500H ונסיים. רק אם כל זוגות התאים המתאימים יכילו מספרים זהים - נציב את הערך FFH.

התוכנית (code17.asm):

CODE SEGMENT	
ASSUME CS:CODE,DS:CODE	
BEGIN: MOV AX,CODE	
MOV DS,AX	
MOV SI,800H	מצביע על הכתובת הראשונה הנבדקת
MOV DI,900H	מצביע על הכתובת הראשונה
MOV BX,500H	מצביע על כתובת500H
MOV CX,21H	מונה הלולאה: 21H זוגות תאים ייבדקו
AGAIN: MOV DL,[SI]	מעתיק לאוגר DL את ערך התא
CMP DL,[DI]	משווה עם ערך התא שכתובתו נתונה ב-ID
JNE NOT_EQU	אם אינם שווים - קופץ לתווית NOT_EQU
INC SI	
INC DI	
LOOP AGAIN	אם נמצא כאן - ממשיך בבדיקת התאים
MOV AL,0FFH	אם הגיע לכאן ולא דילג לתווית NOT_EQU, הרי
MOV [BX],AL	שכל זוגות התאים מכילים ערכים שווים, ולכן מוצב
JMP END_PRG	הערך FFH בתא 500H, וקופץ לסוף התוכנית
NOT_EQU: MOV AL,0	לכאן התוכנית תגיע רק כאשר אחד מזוגות התאים
MOV [BX],AL	הנבדקים אינו מכיל ערכים זהים, ולכן יוצב 0 בתא 500H
END_PRG: NOP	
CODE ENDS	
END BEGIN	

כדי לבדוק את התוכנית באמצעות DEBUG, נציב ערכים מסוימים בבלוק הזיכרון שבתחום הכתובות 800H-820H, ואת אותם הערכים נציב בבלוק הזיכרון 900H-920H.

נריץ את התוכנית ונבדוק שהוצב FFH בתא שכתובתו 500H. לאחר מכן, נשנה את ערכו של אחד התאים בלבד, כך שלא תהיה זהות מוחלטת בין הערכים בבלוקים הנבדקים.

נריץ שוב, ונבדוק שערך תא 500H הינו כעת 0.

תרגיל 18: השוואה והצבה

עליך לכתוב תוכנית לבדיקת ערכי התאים 1000H ו- 1001H.

- ❖ אם ערך תא 1001H גדול יותר - הצב 1 בכל התאים 900H-950H.
- ❖ אם ערך תא 1000H הינו הגדול - הצב 2 בכל התאים 900H-950H.

לפניך אחת מהתוכניות שניתן לכתוב עבור בעיה זו:

```
CSEG SEGMENT
  ASSUME CS:CSEG,DS:CSEG
START:  MOV AX,CSEG
        MOV DS,AX
        MOV DI,900H
        MOV CX,51H
        MOV BX,1001H
        MOV DL,[BX]
        CMP DL,[BX+1]
        JG ONE
        MOV AL,2
        JMP MAKE
ONE:    MOV AL,1
MAKE:   MOV [DI],AL
        INC DI
        LOOP MAKE
CSEG ENDS
END START
```

לאחר כתיבת התוכנית בעורך, נריץ MASM כדי לבדוק שגיאות תחביריות וליצור קובץ בשפת מכונה (סיומת OBJ) אם אין שגיאות, נריץ LINK ליצירת קובץ מוכן להרצה (EXE).

בשלב האחרון נריץ את התוכנית באמצעות DEBUG ונבדוק אותה:

1. נכתוב U0 ונבדוק מהי הכתובת לאחר סוף התוכנית. בתוכנית זו הכתובת היא 1H.
   נבדוק גם מהי הכתובת שתוצב באוגר DS ותהווה כתובת בסיס (זהו המספר הרשום בפקודה הראשונה במקום המילה CSEG). נניח, לצורך ההסבר בתרגיל זה, שהכתובת הינה 1570H.
2. נציב בתא 1001H ערך גדול מזה של תא 1000H: E 1570:1000 4 9
   הערך 4 הוכנס לתא 1000H והערך 9 - לתא 1001H.
   רצוי לבדוק שהערכים אמנם נמצאים בתאים: D 1570:1000 1001
3. נריץ את התוכנית: G=0 21
4. נבדוק את הערכים ב- 900H-950H: D 900 950
   עלינו לראות שתאים אלה קיבלו את הערך 1.
5. נציב בתא 1000H ערך גדול יותר, נריץ ונבדוק שבתאים 900H-950H הוצב הערך 2.

תרגיל 19: חיפוש רצף של נתונים בזיכרון

הינך מתבקש לכתוב תוכנית באסמבלי, שבודקת אם שני הערכים 30H ו-31H נמצאים בבלוק הנתונים בכתובות 470H עד 490H (כולל), ואם הם רציפים.

אם כן - התוכנית תציב 1 באוגר CX, אחרת - התוכנית תציב 0.

הפתרון:

נבדוק ערך של כל תא. אם נמצא ערך 30H, נבדוק אם בתא שלאחריו נמצא הערך 31H. אם כן, נציב 1 באוגר CX ואם לא - התוכנית תמשיך לחפש תא בעל ערך השווה ל-30H.

התוכנית (code19.asm):

CODE SEGMENT	
ASSUME CS:CODE,DS:CODE	
START: MOV AX,CODE	
MOV DS,AX	
MOV BX,470H	BX משמש כמצביע על תאי הזיכרון
CHECK: MOV DL,[BX]	ערך התא מועתק לאוגרDL
MP DL,30H	
JE YESH	אם הערך הינו 30H, קופץ ל-YESH
INC BX	
CMP BX,490H	
JNG CHECK	אם לא הגיע לתא האחרון - ממשיך
JMP NOT_FOUND	לכאן מגיעים לאחר שנבדקו כל התאים, ואף לא אחד מהם שווה 30H - מתבצעת קפיצה ל- NOT_FOUND
YESH: INC BX	BX מתקדם ומצביע על התא הבא
MOV DL,[BX]	
CMP DL,31H	
JNE CHECK	אם ערך תא זה אינו 31H - חוזר לבדיקת שאר התאים
MOV CX,1	אם הגיע לכאן - ערך התא הינו 31H (ערך התא הקודם הינו 30H) ולכן קיים רצף של הערכים
JMP GAMUR	מתבצעת קפיצה לסוף התוכנית
NOT_FOUND: MOV CX,0	לכאן מגיעים במידה שלא נמצא הרצף המבוקש, ואז מוצב 0 באוגר CX
GAMUR: NOP	
CODE ENDS	
END START	

את התוכנית בודקים פעמיים: פעם אחת כאשר יש רצף כמבוקש (ואז ערך אוגר CX יהיה 1) ובפעם השנייה - כאשר אין רצף (CX יקבל את הערך 0).

תרגיל 20: בדיקת תחום ערכים בזיכרון

כתוב תוכנית המציבה 1 באוגר DX, אם סכום ערכי התאים 800H-830H נע בתחום שבין 40H-60H. אחרת, יוצב 0 באוגר זה.

פתרון:

נחבר תחילה את סכום ערכי התאים, ולאחר מכן נבדוק:

- אם הסכום קטן מ-40H: נציב 0 באוגר DX.
- אם הסכום גדול מ-60H: נציב 0 באוגר DX.

אם לא התקיים אחד משני התנאים האלה - התוצאה הינה בתחום, ולכן יוצב 1 ב-DX.

התוכנית:

```
CODE_SEG SEGMENT
   ASSUME CS:CODE_SEG,DS:CODE_SEG
START:  MOV AX,CODE_SEG
        MOV DS,AX
        MOV BX,800H
        MOV CX,31H
        MOV DL,0
SAKEM: ADD DL,[BX]
        INC BX
        LOOP SAKEM
        CMP DL,40H
        JL BAD
        CMP DL,60H
        JG BAD
        MOV DX,1
        JMP OK
BAD:    MOV DX,0
OK:     NOP
CODE_SEG ENDS
END START
```

נריץ את MASM ואת LINK ואחר כך נריץ את התוכנית שהתקבלה ונבדוק אותה באמצעות DEBUG:

1. נציב 0 בכל התאים שבתחום הכתובות 800H-830H. לאחר מכן, נציב בכמה תאים ערכים כאלה, שסכומם יהיה בתחום 40H-60H.
2. נריץ ונבדוק שאוגר DX קיבל את הערך 1.
3. נשנה את הערכים בתאים 800H-830H, כך שסכומם יהיה קטן מ-40H, נריץ ונבדוק ש-DX שווה 0.
4. נשנה את הערכים כך שסכומם יהיה גדול מ-60H, נריץ ונוודא ש-DX=0.

תרגילים

עליך לכתוב, להריץ ולבדוק במחשב תוכניות שונות. כל שאלה מייצגת תוכנית נפרדת, המבצעת משימה. הקפד לכתוב תוכניות מסודרות ושלמות.

פירוט המשימות בכל תוכנית:

1. הצבת הערך A8H בתאי הזיכרון בכתובות 1900H-1950H.
2. הצבת הערך 1 בתאי זיכרון 800H-810H, ואת הערך 2 בתאי זיכרון 811H-820H.
3. הצבת הערך 6 בתאים זוגיים **בלבד** בתחום הכתובות 1010H-1030H.
4. בדיקת ערכי התאים בכתובות 620H-630H. אם נמצא הערך FFH באחד (או יותר) מתאים אלה, להציב 1 בתא 600H.
5. הצבת מספר התאים שעברנו מתחילת החיפוש ועד למציאת הערך 38H באוגר CX. החיפוש יתחיל בתא שכתובתו 1080H ויסתיים לאחר מציאת הערך.
6. הצבת 44H בתאים זוגיים, 33H בתאים אי-זוגיים, בתחום הכתובות 580H-590H.
7. נתונים ציוני תלמידים בתאים 1500H-1600H.
   - יש למצוא את מספר הנכשלים (ציון נמוך מ-55), ולהציב את התוצאה בתא שכתובתו 1700H.
   - יש למצוא את מספר המצטיינים (ציון 90 ומעלה), ולהציב את התוצאה בתא שכתובתו 1701H.
8. למצוא את המספר הקטן ביותר מבין ערכי התאים 660H-670H. להציב את התוצאה בתא 400H.
9. להציב את ההפרש בין המספר הגדול והמספר הקטן.

   הבדיקה תיערך על ערכי התאים בכתובות 320H-330H. התוצאה תוצב בתא שכתובתו 338H.
10. לבדוק אם קיימים יותר מ-8 ערכים השווים 0 בתאים 700H-800H.

    אם כן, להציב את הערך 1 באוגר AX. אם לא, להציב את הערך 0 באוגר זה.
11. לבדוק אם סכום ערכי התאים הזוגיים שווה לסכום ערכי התאים האי-זוגיים. הבדיקה תיעשה בתחום הכתובות 300H-308H.

    אם הסכומים שווים, יוצב 1 באוגר BX. אחרת - יוצב 0 באוגר זה.
12. **להחליף** בין הנתונים בבלוק 300H-305H לבין הנתונים בבלוק 320H-325H.
13. נתון בלוק נתונים שמתחיל בכתובת 700H ומסתיים במספר FFH. יש למצוא מהי הכתובת, שבה נמצא **לראשונה** זוג תאי זיכרון **צמודים**, המכילים שני מספרים **זהים** זה לזה. את הכתובת יש להעתיק לאוגר AX.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

6

פסיקות

מה זו פסיקה וכיצד מפעילים אותה?

הכרת נושא זה מעלה אותנו בכמה דרגות, מכיון שבאמצעות הפסיקות ניתן לכתוב תוכניות מעניינות ושימושיות יותר. פסיקות הינו נושא נרחב ומעמיק מאוד. בהסברים ובדוגמאות שנציג, נחרוג מעט מהחומר הנדרש בתוכנית הלימודים, כדי לעורר עניין גדול יותר בשפה ולשפר את השליטה בה, אך לא נעסוק בכל ההיבטים של הפסיקות.

כשאנו מבצעים **פסיקה** (Interrupt), אנו מפסיקים למעשה את ביצוע התוכנית שלנו, פונים לתוכנית אחרת הכתובה בזיכרון המחשב, ולאחר מכן חוזרים להמשך התוכנית שהופסקה.

מערכות ההפעלה ויצרני המחשב, מספקים עבור המתכנת רשימה ארוכה של תוכניות כתובות, אשר אנו יכולים להיעזר בהן כדי לבצע פעולות מסוימות, כמו קליטת נתונים מהמקלדת, או הצגת הודעות על גבי המסך.

כדי לפנות לביצוע אחת מהתוכניות הללו (כלומר, לבצע תהליך פסיקה), רושמים את מספר הפסיקה המבוקשת. אם כן, לכל תוכנית פסיקה יש מספר מזהה, קוד פסיקה (interrupt code), שיש להכניס אותו לאוגר. כך, כאשר נבצע פסיקת DOS, תהיה פנייה לתוכנית המבוקשת, או השירות של הפסיקה, והיא תורץ עבורנו.

כך למשל, אם נרשום "פסיקת DOS מספר 1" באמצעות פקודות מתאימות, כמובן, נפעיל תוכנית הקולטת תו בודד מהמקלדת. אם נרשום "פסיקת DOS מספר 9", נגרום להרצת תוכנית המציגה הודעות על גבי המסך, וכן הלאה.

כל תוכנית המשתמשת בפסיקה, **חייבת** להכיל **מקטע קוד** (code segment), וגם **מקטע מחסנית** (stack segment).

בנושא המחסנית נדון בהרחבה בהמשך. כאן נאמר שזהו אזור בזיכרון המיועד לשמירת נתונים זמניים. כדי שהתוכנית תכיל מקטע מחסנית, עלינו לכתוב את הפקודות האלו:

```
STA SEGMENT STACK
        DB 100H DUP (0)
STA ENDS
```

הערה: באסמבלרים שונים, ייתכן שההגדרה שלעיל תהיה שונה. עליך לבחון זאת בתיעוד של האסמבלר, במקרה שההגדרה לעיל אינה פועלת כנדרש.

הסבר:

1. המילה **STA** נבחרה כמייצגת את שם המקטע (במקומה ניתן כמובן לבחור בכל שם חוקי אחר).
2. המילה **STACK** היא מילה שמורה וחובה לרשום אותה. היא מציינת שמתבצעת הגדרה של מחסנית.
3. הפקודה **DB** (Define Byte) מציינת שאנו מגדירים נתונים בגודל בית (8 סיביות, גודלו של תא זיכרון). ההוראה **DUP** (Duplicate) מציינת שאנו רוצים שהמחסנית תכלול מספר נתונים כאלה ובדוגמה - 100H פעמים. הספרה (0) מציינת שכל אחד מהנתונים יהיה בעל ערך 0. לסיכום, השורה: "DB 100H DUP (0)" מציינת שבתוך המחסנית מוגדרים 100H תאי זיכרון שכל אחד מהם מקבל את הערך 0.
4. **STA ENDS** - סיום הוראות המקטע בעל השם STA.

בנוסף לכך, יש להוסיף לשורה ASSUME CS:CODE,DS:CODE התייחסות לכך שהמקטע בשם STA הינו התחלת המחסנית. נעשה זאת על ידי שינוי הפקודה:

ASSUME CS:CODE,DS:CODE,SS:STA

כלומר: אוגר SS מכיל את כתובת ההתחלה של המחסנית. בתוכנית זו נציין שהתחלת המחסנית בתוכנית היא במקטע STA.

נציג מספר פסיקות ונדגים באמצעות תוכניות את השימוש בהן.

פסיקה לסיום תוכנית ויציאה למערכת ההפעלה

פסיקת DOS מספר 21H; קוד הפסיקה 4C00H, לאוגר AX.

כאשר הרצנו תוכניות באמצעות DEBUG (באמצעות ההוראה =G), לא נדרש מאיתנו לחזור למערכת ההפעלה, למצב שבו מופיע הסימן המנחה (prompt) <A או <C.

לעומת זאת, אם נרצה להריץ תוכנית באופן ישיר ממערכת ההפעלה, כפי שאנו מריצים משחק או מעבד תמלילים, עלינו לרשום **בסוף** התוכנית שלנו הוראת פסיקה שתגרום להעברת השליטה למערכת ההפעלה לאחר סיום ביצוע התוכנית.

מתי נרצה להריץ תוכנית באסמבלי ישירות ממערכת ההפעלה?

נרצה בזאת, אם התוכנית מבצעת פעולה של קלט או פלט, כמו למשל קליטת נתונים מהמקלדת, הצגת הודעות על המסך, הפקת צלילים וכדומה. בכל המקרים האלה נעדיף להריץ את התוכנית ישירות ממערכת ההפעלה.

תוכניות כגון HEBREW, FORMAT, DISKCOPY ואחרות, הן תוכניות של DOS שנכתבו בשפת אסמבלי ומורצות ישירות ממערכת ההפעלה, ולא באמצעות DEBUG, כפי שעשינו עד כה.

כתיבת הוראת פסיקה בסיום התוכנית הינה דבר פשוט לביצוע: בסוף כל תוכנית שנרצה להריץ ישירות ממערכת ההפעלה, נכתוב את צמד הפקודות האלו:

```
MOV AX,4C00H
INT 21H
```

הסבר:

הפקודה INT (קיצור של interrupt) גורמת לכך שתתבצע פסיקה. המספר **21H** הנמצא מימין לפקודה מציין שזוהי פסיקה של DOS, כלומר תוכנית שירות מיוחדת, שהינה חלק ממערכת ההפעלה.

הערך **4C00H** המוכנס לאוגר AX הוא **קוד הפסיקה**, או זיהוי של **תוכנית הפסיקה** שאנו רוצים להפעיל. במקרה זה, התוכנית תגרום ליציאה למערכת ההפעלה. בעניין זה נשוב ונדון כשנעסוק בנושא "וקטור הפסיקות".

דוגמה:

כדי להמחיש את ההסבר, נכתוב תוכנית המשתמשת בפסיקה זו. התוכנית אינה מבצעת דבר, אך נועדה להבהרת הנושא. נכתוב את התוכנית בעורך ונקבע עבורה את השם 001.ASM.

```
STA SEGMENT STACK                    ; הגדרת המחסנית
        DB 100H DUP (0)
STA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE,SS:STA
START: MOV AX,CODE
        MOV DS,AX
        MOV AX,4C00H
        INT 21H                      ; סיום ויציאה ל-DOS
CODE ENDS
END START
```

לאחר מכן, נריץ כרגיל MASM ו-LINK. הפעם, לא נקבל את ההודעה של LINK: "WARNING: NO STACK SEGMENT", שמשמעותה: "אזהרה: אין מחסנית". הסיבה מובנת: התוכנית שלנו כוללת עכשיו מקטע מחסנית.

כעת לא ניכנס ל-DEBUG, אלא נריץ ישירות ממערכת ההפעלה: נקליד את שם התוכנית ונקיש Enter. בדוגמה זו נקליד 001 ונקיש Enter.

התוכנית לא תבצע דבר ותצא למערכת ההפעלה. ללא הוראת הפסיקה, המחשב היה נתקע!

פסיקה להצגת הודעה על המסך

פסיקת DOS מספר 21H ; קוד הפסיקה 9, לאוגר AH.

כדי להציג הודעה על המסך יש לבצע את הפעולות הבאות :

1. לכתוב את ההודעה שברצוננו להציג. ההודעה נרשמת בסוף התוכנית בין גרשיים, ובסיומה נרשם הסימן $ (לסימון סוף ההודעה). ההודעה תתחיל בשם כלשהו, לאחריו ההוראה DB, לאחריו ההודעה. עיין גם בנושא "משתנים".

2. להציב באוגר DX את כתובת ההתחלה של ההודעה.

הדבר נעשה באמצעות הפקודה **offset** המחשבת את ההיסט (ההפרש) של תחילת ההודעה מתחילת הסגמנט (המקטע). עיין גם בנושא "משתנים".

3. לרשום את צמד הפקודות :

```
MOV AH,9
INT 21H
```

משמעות הפקודות : לבצע פסיקת DOS מספר **9** : הצגת הודעה.

שים לב: פסיקה **9**, וגם פסיקה **2** גורמות לשינוי של תוכן האוגר AL.

דוגמה 1

נכתוב תוכנית המציגה את ההודעה "Good morning" (002.asm) :

```
STA SEGMENT STACK
        DB 100H DUP (0)
STA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:CODE,SS:STA
START:MOV AX,CODE
        MOV DS,AX
        MOV DX,OFFSET BOKER
        MOV AH,9
        INT 21H
        MOV AX,4C00H
        INT 21H
        BOKER DB 'Good morning $'
CODE ENDS
END START
```

הסבר התוכנית:

נתחיל דווקא בסוף, מההודעה שנרשמה בסוף התוכנית, לאחר הפסיקה ליציאה ל-DOS. כפי שניתן לראות, אנו מכנים את ההודעה בשם כלשהו (כאן נבחר השם BOKER), ההודעה נרשמת בין גרשיים ובסיומה נרשם סימן הדולר, המסמן את סיום ההודעה. אל תשמיט אותו בטעות! (שים לב שההודעה בגרשים בודדים).

הפקודה הבאה גורמת להציב באוגר DX את הכתובת ההתחלתית של ההודעה בשם BOKER :

```
MOV DX,OFFSET BOKER
```

הפקודות הבאות גורמות לביצוע פסיקת DOS להצגת הודעות במסך :

```
MOV AH,9
INT 21H
```

צמד הפקודות הבאות גורמות לפסיקת DOS לסיום התוכנית וליציאה למערכת ההפעלה :

```
MOV AX,4C00H
INT 21H
```

לאחר הרצת MASM ו-LINK נריץ את התוכנית ישירות ממערכת ההפעלה. על המסך נקבל את ההודעה : "Good morning".

דוגמה 2

נרשום על המסך את ההודעה : "Hi", ובשורה מתחת : "Bye!!!".

התוכנית (003.asm) :

```
SSEG SEGMENT STACK
      DB 100H DUP (0)
SSEG ENDS
CODE SEGMENT
      ASSUME CS:CODE,DS:CODE,SS:SSEG
BEGIN:  MOV AX,CODE
        MOV DS,AX
         MOV DX,OFFSET MES1
         MOV AH,9
         INT 21H
         MOV AX,4C00H
         INT 21H
         MES1 DB   ' Hi'
                DB      10,13
                DB     'Bye!!! $'
CODE ENDS
END START
```

הסבר: המשתנה MES1 מכיל את ההודעה "Hi", את תווי הבקרה 10 ו-13 (הגורמים להעברת הסמן במסך לתחילת השורה הבאה) ואת ההודעה "Bye!!!".

שים לב: הסימן $ נרשם פעם אחת בלבד, בסיום ההודעה!

פסיקה לקליטת מקש מהמקלדת

פסיקת DOS מספר 21H; קוד הפסיקה 1, לאוגר AH.

פסיקה זו גורמת לכך שהמחשב יעצור וימתין ללחיצה על מקש. כאשר יילחץ מקש, **הקוד** שלו ייכנס לאוגר AL, וכך התוכנית תוכל לדעת איזה מקש נלחץ.

לכל מקש יש קוד מזהה משלו (הנקרא **קוד סריקה**, Scan Code). דוגמאות :

- מקש Enter : הקוד 13, או 0DH בבסיס 16.
- מקש Space Bar (מקש הרווח) : הקוד 32, או 20H.
- מקשי הספרות 0 עד 9 : הקוד 48 (30H) עד 57 (39H) בהתאמה.

כדי לבצע פסיקה זו יש לרשום את צמד הפקודות :

```
MOV AH,1
INT 21H
```

דוגמה 1

נכתוב תוכנית הקולטת מקשים שוב ושוב, עד שמקש הספרה 8 נלחץ ויגרום לסיום התוכנית.

התוכנית (004.asm) :

```
STAM SEGMENT STACK                     הגדרת המחסנית
  DB 100H DUP (0)
STAM ENDS
PROG SEGMENT
  ASSUME CS:PROG,DS:PROG,SS:STAM
FIRST:  MOV AX,PROG
        MOV DS,AX
BACK:   MOV AH,1                       פסיקה לקליטת מקש
        INT 21H
        CMP AL,38H                     האם המקש שנלחץ הינו מקש הספרה 8?
        JNE BACK                       אם לא - חזור לתווית BACK לקלוט מקש נוסף
        MOV AX,4C00H                   פסיקה לסיום ויציאה ל-DOS
        INT 21H
STAM ENDS
 END FIRST
```

במקום הפקודה "CMP AL,38H" ניתן גם לכתוב CMP AL,'8'.

בשני המקרים מבצעים השוואה בין אוגר AL לבין **הקוד** הידוע של מקש 8. גם תוכנית זו תורץ ישירות ממערכת ההפעלה.

דוגמה 2

נכתוב תוכנית שתציג הודעה: "Very Good", אם נלחץ מקש הספרה 4. לאחר מכן, התוכנית תמשיך לקלוט מקש כלשהו. התוכנית תסתיים רק כאשר נלחץ על מקש הרווח. כל מקש אחר לא ישפיע ולא יגרום להצגת הודעה.

האלגוריתם לפתרון: נבצע פסיקה לקליטת מקש.

- אם המקש הוא מקש הרווח (קוד 20H), נקפוץ לסוף התוכנית ונסיים.
- אם המקש הוא 4 (קוד 34H), נבצע פסיקה המציגה את ההודעה המבוקשת.
- אם המקש אינו אחד מאלה - נחזור לביצוע פסיקה לקליטת מקש נוסף.

התוכנית (005.asm):

`STA SEGMENT STACK`	הגדרת המחסנית
`  DB 100H DUP (0)`	
`STA ENDS`	
`CSEG SEGMENT`	
`  ASSUME CS: CSEG,DS: CSEG,SS:STA`	
`BEGIN : MOV AX, CSEG`	
`        MOV DS,AX`	
`KLOT:   MOV AH,1`	קליטת מקש
`        INT 21H`	
`        CMP AL,20H`	
`        JE FINI`	אם נקלט מקש רווח - סיים
`        CMP AL,34H`	
`        JNE KLOT`	אם לא נקלט מקש הספרה 4 חזור לתווית KLOT
`        MOV DX,OFFSET YES`	אם הגענו לכאן - נלחץ המקש 4, ולכן נבצע
`        MOV AH,9`	פסיקה להצגת ההודעה הרשומה ב-YES
`        INT 21H`	
`        JMP KLOT`	לאחר מכן נחזור להמשך קליטה
`FINI:   MOV AX,4C00H`	סיום ויציאה ל-DOS
`        INT 21H`	
`           YES DB ' Very Good $'`	
`CSEG ENDS`	
`END BEGIN`	

דוגמה 3

נכתוב תוכנית שתקלוט מספר דו-ספרתי ותציג הודעה "Yes" אם הוא גדול מ-86.

האלגוריתם לפתרון הבעיה: מספר דו-ספרתי מורכב משתי ספרות ולכן צריך לקלוט שני מקשים. המקש הראשון שנקלוט יהיה ספרת העשרות והמקש השני - ספרת האחדות. הקליטה תתבצע באמצעות שתי פסיקות לקליטת מקש.

לאחר הקליטה נבדוק:

- אם ספרת המאות הנקלטת גדולה מ-8: המספר הנקלט גדול יותר.
- אם ספרת המאות הנקלטת קטנה מ-8: המספר הנקלט אינו גדול מ-86.
- אם ספרת המאות הנקלטת שווה ל-8, יש לבדוק את ספרת האחדות:

1. אם ספרת האחדות של המספר הנקלט גדולה מ-6: המספר הנקלט גדול מ-86.
2. אם ספרת האחדות קטנה או שווה ל-6: המספר הנקלט אינו גדול מ-86.

אם לאחר הבדיקה נמצא שהמספר הנקלט גדול מ-86, נבצע פסיקה להצגת הודעה. אם הבדיקה מציינת שהמספר אינו גדול, נסיים על ידי פסיקה ליציאה למערכת ההפעלה.

התוכנית (006.asm):

```
SSEG SEGMENT STACK                      הגדרת המחסנית
  DB 100H DUP (0)
SSEG ENDS
CSEG SEGMENT
  ASSUME CS: CSEG,DS: CSEG,SS:SSEG
START : MOV AX, CSEG
        MOV DS,AX
AGAIN:  MOV AH,1
        INT 21H                         קליטת ספרה ראשונה (ספרת עשרות)
        MOV DL,AL                       העתקת הספרה ל-DL לשם שמירתה
        MOV AH,1
        INT 21H                         קליטת ספרה שנייה (ספרת אחדות)
        MOV DH,AL                       העתקת הספרה ל-DH לשם שמירתה
        CMP DL,38H
        JG BIGGER                       אם הספרה הנקלטת גדולה מ-8
        JE CHECK                        אם הספרה שווה ל-8
        JL FIN                          אם הספרה קטנה מ-8
CHECK:  CMP DH,36H
        JLE FIN                         אם הספרה השנייה קטנה מ-6 - סיים
```

```
BIGGER: MOV DX,OFFSET YOFI
        MOV AH,9
        INT 21H                    הצגת ההודעה על המסך
FIN:    MOV AX,4C00H
        INT 21H                    סיום ויציאה למערכת ההפעלה
          YOFI DB ' Yes $ '
CSEG ENDS
END START
```

פסיקה להצגת תו בודד על המסך

פסיקת DOS מספר 21H.

קוד הפסיקה: 1, לאוגר AH. לקליטת מקש והצגתו.

קוד הפסיקה: 7, לאוגר AH. לקליטת מקש ללא תצוגה.

כדי להציג תו בודד (אות, ספרה וכדומה) על המסך, נציב באוגר DL את הקוד של התו שברצוננו להציג, ולאחר מכן נרשום את צמד הפקודות:

```
MOV AH,2
INT 21H
```

לדוגמה, כדי להציג את הספרה 9 על גבי המסך, נרשום:

```
MOV DL,39H        קוד ASCII של הספרה 9 הוא 39H
MOV AH,2          הפסיקה
INT 21H
```

ניתן לעשות זאת בצורה שונה במקצת:

```
MOV DL,'9'
MOV AH,2
INT 21H
```

לפי שיטה זו, אנו רושמים את התו שברצוננו להציג בין גרשיים. כך אנו חוסכים מעצמנו את הצורך לדעת מהו **הקוד** של התו. כאמור, בכל מקרה לאוגר DL יוצב הערך 39h.

דוגמה 1

כתוב תוכנית (007.asm) המשתמשת בשיטת הפסיקה שהצגנו, כדי להציג 100 פעם את התו ■ (ריבוע קטן), שקוד ASCII שלו הוא 254. תו זה מתקבל על ידי לחיצה קבועה על מקש Alt ולחיצה על מקשי הספרות **שמימין** המקלדת: 2, 5 ו-4, או בקיצור: Alt+254.

```
STACK_SEG SEGMENT STACK
  DB 100H DUP (0)
STACK_SEG ENDS
CODE_SEG SEGMENT
   ASSUME CS:CODE_SEG,DS:CODE_SEG,SS:STACK_SEG
BEGINING: MOV AX,CODE_SEG
          MOV DS,AX
          MOV CX,100
LOOPY:    MOV DL,'■'
          MOV AH,2
          INT 21H
          LOOP LOOPY
          MOV AX,4C00H
          INT 21H
CODE_SEG ENDS
END BEGINING
```

דוגמה 2

נכתוב תוכנית (008.asm) המציגה את הספרות 0 עד 9, זו אחר זו.

השיטה: נבצע לולאה שתציג בהתחלה את התו בעל הקוד 30H (כלומר הספרה 0), נוסיף 1 לקוד ונשלח לתצוגה (ואז תוצג הספרה 1), וכך הלאה עד הקוד 39H (המייצג את הספרה 9).

```
SSEG SEGMENT STACK
  DB 100H DUP (?)
SSEG ENDS
CSEG SEGMENT
  ASSUME CS:CSEG,DS:CSEG,SS:SSEG
START: MOV AX,CSEG
       MOV DS,AX
       MOV CX,10
       MOV DL,30H
AGAIN: MOV AH,2
       INT 21H
       INC DL
       LOOP AGAIN
       MOV AX,4C00H
       INT 21H
CSEG ENDS
END START
```

דוגמה 3

נכתוב תוכנית (009.asm) המציגה על המסך את כל התווים.

השיטה: נשלח לתצוגה את התו בעל הקוד 0 (התו הראשון), לאחריו את התו בעל הקוד 1, וכך הלאה, סה"כ 255 תווים.

```
SSEG SEGMENT STACK
    DB 100H DUP (0)
SSEG ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE,SS:SSEG
BEGIN:  MOV AX,CODE
        MOV DS,AX
        MOV CX,255
        MOV DL,0
KTOV:   MOV AH,2
        INT 21H
        INC DL
        LOOP KTOV
        MOV AX,4C00H
        INT 21H
CODE ENDS
END BEGIN
```

דוגמה 4

נכתוב תוכנית שתקלוט שתי ספרות, שהראשונה בהן גדולה מהשנייה. התוכנית תציג על המסך את ההפרש ביניהן.

האלגוריתם לפתרון הבעיה:

1. פסיקה לקליטת מקש של הספרה הראשונה. נשמור את הנתון שהתקבל באוגר DL.
2. פסיקה לקליטת הספרה השנייה. נשמור את הנתון באוגר DH.
3. נחסר את הספרה השנייה (DH) מהראשונה (DL).
4. לתוצאה זו נוסיף 30H.

   הסיבה: כדי להציג את הספרה באמצעות הפסיקה, יש להציב באוגר DL את **הקוד** של התו. הקוד של כל ספרה שווה לערך הספרה ועוד 30H.
5. נשלח את הספרה לתצוגה באמצעות פסיקה.
6. נסיים את התוכנית באמצעות פסיקה.

הערה: בתוכנית זו איננו בודקים אם הוקשו ספרות. כמובן שבפרויקט מעשי, **חובה** לבדוק זאת, כדי להבטיח את אמינות הפעולות.

התוכנית (010.asm):

```
MACH SEGMENT STACK
     DB 100H DUP (0)
MACH ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE,SS:MACH
BEG:    MOV AX,CODE
        MOV DS,AX
INS:    MOV AH,1
        INT 21H
        MOV AL,DL
        MOV AH,1
        INT 21H
        MOV DH,AL
        SUB DL,DH
        ADD DL,30H
        MOV AH,2
        INT 21H
        MOV AX,4C00H
        INT 21H
CODE ENDS
END BEG
```

נשפר את התצוגה על המסך:

- נציג את הסימן מינוס (-) לאחר קליטת הספרה הראשונה.
- נציג את הסימן שווה (=) לאחר קליטת הספרה השנייה.

התוכנית המשופרת (011.asm):

```
MACH SEGMENT STACK
    DB 100H DUP (0)
MACH ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE,SS:MACH
BEG: MOV AX,CODE
     MOV DS,AX
INS:  MOV AH,1
     INT 21H
     MOV CL,AL
     MOV DL,'-'
     MOV AH,2
     INT 21H
     MOV AH,1
     INT 21H
     MOV CH,AL
```

```
    MOV DL,'='
    MOV AH,2
    INT 21H
    SUB CL,CH
    ADD CL,30H
    MOV DL,CL
    MOV AH,2
    INT 21H
    MOV AX,4C00H
    INT 21H
CODE ENDS
END BEG
```

דוגמה 5

נכתוב תוכנית לקליטה של ספרה (0-9), והצגה על המסך של כל הספרות מ-0 ועד אליה. למשל, אם הקשת 3, יוצגו על המסך הספרות 1, 2, 3.

בתוכנית זו נשתמש באפשרות נוספת לקליטת מקש, באמצעות קוד פסיקה **7**:

```
MOV AH,7
INT 21H
```

פקודות אלו יגרמו להמתנה לקליטת מקש (על פי הספרה **7** באוגר AH), בדומה לפסיקה שלמדנו קודם, אך ההבדל הוא שלאחר לחיצה על מקש כלשהו, הוא **לא יוצג** על המסך.

פסיקה זו שימושית כאשר מסיבה כלשהי (סודיות, נוחות, אסתטיות) לא מעוניינים שיופיע על המסך התו של המקש הנלחץ.

אזהרה: כאשר מבצעים פסיקה, עלול אוגר AH להשתנות! על כן, אסור לשמור נתונים חשובים באוגר AH כאשר עוברים לשלב פסיקה, כי הם עלולים להשתבש בו.

האלגוריתם לפתרון:

1. נבצע פסיקה לקליטת מקש (לצורך הדגמה נבחרה הפסיקה החדשה).
2. נפחית 30H מאוגר AL, כדי לקבל את הערך של הספרה שהוקשה.
3. ערך זה ישמש כמונה לולאה (מספר הספרות שנציג על המסך), לכן נעתיק את הערך לאוגר CL.
4. נציב באוגר DL את הערך 30H (מייצג את הספרה 0).
5. נבצע פסיקה שתציג את הספרה שהקוד שלה ב-DL.
6. נוסיף 1 לאוגר DL.
7. נחזור על סעיפים 5-6 מספר פעמים, השווה לערך CL.
8. נסיים על ידי פסיקה ליציאה ל-DOS.

התוכנית (012.asm):

הפעם, נלווה את התוכנית במספר הערות הסבר:

הגדרת מקטע המחסנית (אזור בזיכרון המיועד לשמירת נתונים זמניים). ניתן לבחור כל שם למקטע (רצף תווים ללא רווחים), אך חובה לכתוב את המילה STACK בסוף ההגדרה, כדי לציין שהגדרת המקטע מתייחסת למחסנית.

STA SEGMENT STACK	
DB 100H DUP (0)	"מילוי" המחסנית ב-100H ערכי 0
STA ENDS	הגדרת המקטע המכיל את הקוד (התוכנית)
CODE SEGMENT	השם שבחרנו הוא CODE
ASSUME CS:CODE,DS:CODE,SS:STA	
PA: MOV AX,CODE	
MOV DS,AX	הצבת כתובת ההתחלה של הנתונים ב-DS
MOV AH,7	
INT 21H	פסיקה הממתינה לקליטת מקש ומציבה באוגר AL את קוד ASCII של המקש
SUB AL,30H	הפחתת 30H גורמת לקבלת ערך הספרה **שים לב!** מדובר על מקשי הספרות, ואינו ישים עבור מקש אחר!
MOV CL,AL	פעולה זו הינה הכרחית! ערך אוגר AL עלול להשתנות לאחר הפסיקה, ולכן חובה להעתיקו לאוגר אחר לשמירה
MOV DL,30H	קוד ASCII של הספרה 0
AG: MOV AH,2	
INT 21H	פסיקה להצגת התו, שהקוד שלו נתון באוגר DL, על המסך
INC DL	מוסיף 1 לקוד הנתון ב-DL
DEC CL	
JNZ AG	חזרה על התהליך עד ש-CL ישווה ל-0
MOV AX,4C00H	
INT 21H	פסיקה הגורמת לסיום ולהעברת השליטה למערכת ההפעלה (ללא פקודה זו המחשב עלול להיתקע!)
CODE ENDS	
END PA	

פסיקה הבודקת אם הוקש מקש

פסיקת DOS מספר 21H; קוד הפסיקה 0BH, לאוגר AH.

כדי לבדוק אם הוקש **מקש כלשהו** במקלדת, צריך לכתוב את שתי הפקודות האלו:

```
MOV AH,0BH
INT 21H
```

אם נקיש על מקש כלשהו, יוצב באוגר AL הערך FFH, ולא - יוצב הערך 0.

דוגמה:

נכתוב תוכנית המציגה את ההודעה "Waiting" שוב ושוב, עד אשר יוקש מקש כלשהו.

בתוכנית זו עלינו ליצור **מצב של השהייה** בין הצגת הודעה אחת לאחרת, כדי שההודעות לא "ירוצו" על המסך במהירות רבה מדי.

ההשהייה תבוצע על ידי פקודות אלו:

```
      MOV CX,0FFFFH
HERE: LOOP HERE
```

הסבר:

בתחילה נציב באוגר CX הערך FFFFH. זהו הערך הגדול ביותר שניתן להכניס לאוגר בגודל 16 סיביות. בשורה הבאה נחסיר 1 מערכו של אוגר CX. אם ערכו אינו שווה ל-0, נקפוץ לתווית HERE, שהינה השורה שבה מתבצע החיסור של 1 מאוגר CX, וכך הלאה.

כלומר, התוכנית תבצע FFFFH פעמים את הפקודה "HERE: LOOP HERE".

הזמן הדרוש לביצוע הפקודה פעמים רבות כל כך, גורם להשהייה. **משך ההשהייה** נקבע, אם כן, על פי הערך ההתחלתי באוגר CX.

אם נרצה להגדיל את זמן ההשהייה לא נוכל לעשות זאת בפשטות, כי FFFFH הוא הערך המקסימלי שאפשר להכניס לאוגר CX. במקרה זה עלינו לבצע לולאה בתוך לולאה, כמו בדוגמה שלפניך:

```
       MOV DX,5
CONT:  MOV CX,0FFFFH
       HERE: LOOP HERE
       DEC DX
       JNZ CONT
```

הלולאה החיצונית, שערכה 5, גורמת לכך שזמן ההשהייה הקודם יוכפל ב-5. כדי לקבל השהייה ארוכה יותר, פשוט נציב ערך גדול יותר מ- 5 באוגר DX.

נחזור אל התוכנית שעלינו לכתוב (013.asm) :

`SSEG SEGMENT STACK`	
`   DB 100H DUP (0)`	
`SSEG ENDS`	
`CSEG SEGMENT`	
`   ASSUME CS:CSEG,DS:CSEG,SS:SSEG`	
`START: MOV AX,CSEG`	
`       MOV DS,AX`	
`SHOOV: MOV AH,0BH`	בדיקה אם נלחץ מקש כלשהו
`       INT 21H`	
`       CMP AL,0FFH`	
`       JE FINISH`	אם נלחץ מקש - סיים
`       MOV DX,OFFSET STAM`	אם לא נלחץ מקש - מגיעים לכאן - ואז
`       MOV AH,9`	מתבצעת פסיקה לשליחת הודעה למסך
`       INT 21H`	
`       MOV CX,0FFFFH`	
`DELAY: LOOP DELAY`	ביצוע השהייה לזמן קצר
`       JMP SHOOV`	חזרה להתחלה
`FINISH: MOV AX,4C00H`	
`       INT 21H`	סיום תוכנית ויציאה ל-DOS
`          STAM DB ' Waiting $ '`	
`CSEG ENDS`	
`END START`	

קליטת מקשים מיוחדים

פסיקת DOS מספר 21H ; קודי פסיקה 2 או 7, לאוגר AH.

חלק מהמקשים מאופיינים על ידי שני קודי ASCII, במקום קוד אחד, המאפיין כל מקש. במקשים אלה, הקוד הראשון הינו 0 והקוד השני משתנה בהתאם למקש. בקבוצת מקשים זו נמצאים : מקשי F1 עד F9, מקשי החצים, מקשי HOME, DEL ועוד. המקשים המיוחדים האלה נקראים לעיתים גם **מקשים כפולים**.

כדי לזהות בתוכנית אם נלחץ מקש מיוחד, יש לבדוק אם קוד ASCII שהתקבל שווה ל-0. אם כן, יש לבצע פסיקה נוספת, כדי לקלוט את הקוד **השני** השייך למקש.

בתוכנית שנציג כעת, נדגים זיהוי הקשה על **מקש חץ-ימינה**. מקש זה נותן קוד ראשון 0, וקוד שני 77. התוכנית תצייר את הסימן Σ (Sigma) בכל פעם שנקיש על מקש חץ-ימינה. כאשר נקיש על מקש Enter, התוכנית תסתיים.

לפניך התוכנית בשלמותה (014.asm) :

STA SEGMENT STACK	
DB 100H DUP (0)	
STA ENDS	
CODE SEGMENT	
ASSUME CS:CODE,DS:CODE,SS:STA	
START: MOV AX,CODE	
MOV DS,AX	
SHOOV: MOV AH,0BH	בדיקה אם נלחץ מקש כלשהו
INT 21H	
CMP AL,0FFH	
JE FINISH	אם נלחץ מקש - סיים
MOV DX,OFFSET STAM	אם לא נלחץ מקש - מגיעים
MOV AH,9	לכאן - ואז מתבצעת פסיקה
INT 21H	לשליחת הודעה למסך
MOV CX,0FFFFH	קוד ASCII של התו Σ הוא 228
MOV DL,'Σ'	תו זה יצויר קודם על המסך
MOV AH,2	התו מתקבל על ידי לחיצה קבועה על מקש
INT 21H	Alt ועל הספרות שמימין המקלדת : 2, 2 ו-8.
AGA: MOV AH,7	
INT 21H	קליטת מקש לתוך אוגר AL
CMP AL,13	
JE FINISH	אם נלחץ Enter - סיים
CMP AL,0	האם זה מקש מיוחד?
JNE AGA	אם לא - המשך לקלוט מקש
MOV AH,7	אם הגענו לכאן - זהו מקש
INT 21H	מיוחד - ויש לקלוט את הקוד השני
CMP AL,77	האם הקוד השני הינו 77 (הקוד של מקש חץ-ימינה)
JNE AGA	אם לא - המשך לקלוט
MOV DL,'Σ'	
MOV AH,2	אם הגענו לכאן - נלחץ מקש
INT 21H	חץ-ימינה, ולכן יוצג התו Σ
JMP AGA	המשך לקלוט מקש
FINISH: MOV AX,4C00H	סיום ויציאה למערכת ההפעלה
INT 21H	
CODE ENDS	
END START	

תרגילים ודוגמאות נוספות אודות המקשים המיוחדים נביא בהמשך.

פסיקה לקביעת מיקום הסמן במסך

פסיקת BIOS מספר 10H; קוד הפסיקה 2, לאוגר AH.

כדי לשנות את מיקום הסמן במסך יש לבצע את הפעולות הבאות:

1. להציב באוגר BH את מספר המסך המוצג (מסך ראשון הינו 0).
2. להציב באוגר DH את מספר השורה (0...24) ובאוגר DL - את מספר הטור (0...79).
3. לרשום את הפקודות:

```
MOV AH,2
INT 10H
```

שים לב, שמספר הפסיקה הינו **10H**, ולא 21H, כפי שראינו עד כה. זוהי פסיקה שונה. **כל הפסיקות בעלות המספר 21H הן פסיקות DOS**, כלומר, אלו הן תוכניות של מערכת ההפעלה. פסיקות בעלות מספר אחר הן **פסיקות BIOS**, תוכנית שמסופקת על ידי יצרן המחשב ונמצאות בזיכרון הקרוי ROM-BIOS.

נראה דוגמה לתוכנית המציגה על ידי שימוש בפסיקה 10H, את הספרה 1 בפינה העליונה השמאלית של המסך, ואת הספרה 2 בפינה התחתונה הימנית (015.asm):

```
STAC SEGMENT STACK
   DB 100H DUP (0)
STAC ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE,SS:STAC
 START: MOV AX,CODE
        MOV DS,AX
        MOV BH,0          קביעת מסך ראשון (מסך 0)
        MOV DH,5          שורה 5
        MOV DL,5          עמודה 5
        MOV AH,2
        INT 10H           פסיקת BIOS למיקום הסמן
        MOV DL,'1'
        MOV AH,2
        INT 21H           הצגת הספרה 1 במיקום הסמן
        MOV DH,20         שורה 20
        MOV DL,75         עמודה 75
        MOV AH,2
        INT 10H           פסיקת BIOS למיקום הסמן
        MOV DL,'2'
        MOV AH,2
        INT 21H           הצגת הספרה 2 במיקום הסמן
```

```
    MOV AX,4C00H
    INT 21H                     סיום ויציאה למערכת ההפעלה
CODE ENDS
END START
```

פסיקה לקליטת מחרוזת מהמקלדת

פסיקת DOS מספר 21H; קוד הפסיקה 0AH, לאוגר AH.

עד כה למדנו, שכדי לקלוט מספר תווים מהמקלדת, צריך תווים בודדים, בכל פעם מקש אחד. אפשר לעשות זאת בדרך שונה: לקלוט ברצף את כל המקשים שהוקשו **עד** להקשת מקש Enter, ואז לפענח אותם זה אחר זה. כיצד נעשה זאת? כך:

1. נגדיר משתנה כלשהו (בתוכנית שבדוגמה, שמו MAKOM), שיכיל:

- במיקום הראשון שלו יירשם מספר המציין את גודל המחרוזת (מספר התווים המקסימלי שאנו רוצים לקלוט; כאשר נחרוג ממספר זה המחשב יצפצף). בתוכנית שתוצג נקבע את גודל המחרוזת 30.
- נגדיר שטח של מספר בתים ריקים, אשר לשם תוכנס המחרוזת מהמקלדת. צריך להקצות מספר בתים גדול יותר מגודל המחרוזת, אך ניתן להסתפק במספר בתים השווה לגודל המחרוזת ועוד 2.

בתוכנית שתוצג, הוגדר אורך המחרוזת בהוראה: DB 30, ומספר הבתים עבור המחרוזת הנקלטת הוגדרה כ-32, בהוראה: DB 32 DUP (?). משמעות הפקודה היא, שיש להקצות 32 בתים ללא ערך התחלתי. סימן השאלה שבתוך הסוגריים מציין נתון ללא ערך התחלתי מוגדר.

2. נציב באוגר DX את כתובת ההתחלה של המשתנה. בתוכנית נרשמה לצורך כך הפקודה: MOV DX,OFFSET MAKOM.

3. נרשום את הפקודות:

```
MOV AH,0AH
INT 21H
```

כתוצאה מכך נקבל:

- בתא **השני** של המשתנה יירשם מספר התווים שנקלטו, לא כולל את המקש Enter המסמן את סוף הקלט מהמקלדת. בתוכנית השתמשנו בתא זה, כדי לדעת כמה מקשים הוקשו. נכתוב פקודות אלו:

```
MOV SI,DX
MOV CL,[SI+1]
```

הפקודות מציבות בתוך אוגר CL את מספר המקשים שהוקשו.

תזכורת: התא הראשון - כתובתו אפס, ולכן כתובתו של התא השני היא 1.

- קודי ASCII של המקשים שהוקשו, יוכנסו החל מהתא השלישי של המשתנה.

הערה: נושא זה מורכב יותר מקודמיו. תוכל לחזור וללמוד אותו לאחר שנעסוק בנושאים מתקדמים ויהיה לך רקע רחב יותר להבינו.

דוגמה

התוכנית הבאה (016.asm) תקלוט מחרוזת מהמקלדת, עד להקשה על Enter, ולאחר מכן תציג את המחרוזת על המסך.

השיטה: קליטת המחרוזת מהמקלדת תיעשה על ידי שימוש בפסיקה לקליטת מחרוזת. הצגת המחרוזת במסך תיעשה על ידי הצגת תו אחר תו, באמצעות פסיקה להצגת תווים בודדים.

```
SSEG SEGMENT STACK
  DB 100H DUP (?)
SSEG ENDS
CSEG SEGMENT
  ASSUME CS:CSEG,DS:CSEG,SS:SSEG
START: MOV AX,CSEG
        MOV DS,AX
        MOV DX,OFFSET MAKOM
        MOV AH,0AH
        INT 21H                 קליטת מחרוזת
        MOV SI,DX
        MOV CL,[SI+1]           אוגר CL יכיל את מספר המקשים שהוקשו
        MOV DL,10
        MOV AH,2
        INT 21H                 מעבר לשורה הבאה
        ADD SI,2                מצביע על תחילתהנתונים של המקשים שנקלטו
BACK:   MOV DL,[SI]             מעתיק את קוד המקש שנקלט לתוך DL
        MOV AH,2
        INT 21H                 מציג את התו
        INC SI                  מצביע על הנתון הבא (המקש הבא שנקלט)
        DEC CL                  מפחית 1 מכמות המקשים שיש להציג
        JNZ BACK
        MOV AX,4C00H
        INT 21H                 סיום ויציאה ל-DOS
        MAKOM DB 30
              DB 32 DUP (?)
CSEG ENDS
END START
```

פסיקה לאתחול המחשב (Reset)

פסיקת DOS מספר 19H; ללא קוד כלשהו.

אחת הדרכים לאתחל את המחשב הינה לחיצה על Reset. אפשר, כמובן, לכבות ולהדליק את המחשב או לאתחל על ידי לחיצה בו-זמנית על Alt+Ctrl+Del. אנו נלמד לעשות זאת בתוכנית שלנו, באמצעות פסיקה **19H**.

התוכנית שלפנינו (017.asm) תציג הודעה על המסך "Reset now?". אם תוקש האות 'y' (אות קטנה) כתשובה, יתבצע האתחול. כל מקש אחר יסיים את התוכנית.

```
STA SEGMENT STACK
DB 100H DUP (0)
STA ENDS
CODE SEGMENT
 ASSUME CS:CODE,DS:CODE,SS:STA
 START: MOV AX,CODE
        MOV DS,AX
        MOV DX,OFFSET MES
        MOV AH,9
        INT 21H            הצגת הודעה על המסך
        MOV AH,7
        INT 21H            קליטת מקש (ללא הצגתו)
        CMP AL, 'y'        האם המקש הנלחץ הינו y
                           שים לב שהאות חייבת להיות קטנה
                           האסמבלר מבדיל בין אות קטנה לגדולה!
        JNE SOFY           אם לא הוקש y - סיים
        INT 19H            פסיקה לאתחול המחשב
SOFY:   MOV AX,4C00H
        INT 21H
           MES DB ' Reset now $ '
CODE ENDS
END START
```

ניפוי שגיאות בתוכנית הכוללת פסיקות

כאשר משתמשים בתוכנת ניפוי, כמו DEBUG למשל, כדי לבדוק תוכניות הכוללות פסיקות, יש לנהוג לפי כללי עבודה מסוימים.

דוגמה 1: הצגת הודעה

ניקח לדוגמה את התוכנית הבאה (018.asm), אשר מציגה הודעה "HELLO" על המסך ומסיימת את פעולתה.

```
SSEG SEGMENT STACK
    DB 100H DP (0)
SSEG ENDS
CSEG SEGMENT
    ASSUME CS:CSEG,DS:CSEG,SS:SSEG
START: MOV AX,CSEG
        MOV DS,AX
        MOV DX,OFFSET HOD
        MOV AH,9
        INT 21H
        MOV AX,4C00H
        INT 21H
            HOD DB ' HELLO $'
CSEG ENDS
END START
```

לאחר שלב MASM ו-LINK, נעבור לתוכנת הניפוי (DEBUG, או אחרת).

התוכנית תוצג ב-DEBUG בצורה זו:

```
MOV AX,1168
MOV DS,AX
MOV DX,0011
MOV AH,9
INT 21
MOV AX,4C00
INT 21
.....
```

הערות:

1. במקום המספר 1168 תקבל ודאי מספר שונה. מספר זה מציין את כתובת הבסיס (הכתובת ההתחלתית) של התוכנית בזיכרון ה-RAM.

2. המספר 0011 שמוצב באוגר DX, מציין את הכתובת ההתחלתית של ההודעה HOD. בתוכנית המקורית נרשמה הפקודה MOV DX,OFFSET HOD

   פירוש הפקודה: העבר לאוגר DX את הכתובת ההתחלתית של המשתנה HOD. הכתובת ההתחלתית הינה, אם כן, 11.

3. הנקודות בסוף התוכנית מציינות שיש המשך לפקודות, אולם הן לא נרשמו מאחר שאינן שייכות לתוכנית.

נניח שברצוננו לבדוק את התוכנית בהרצת **צעד-יחיד**, פקודה אחר פקודה. פעולה זו תיעשה בשני שלבים:

1. בתחילה, נשנה ל-0 את ערכו של אוגר IP, אשר מצביע על כתובת ההוראה הבאה לביצוע. נעשה זאת על ידי הפקודה RIP והערך 0.

2. בשלב השני, נשתמש בהוראה T של DEBUG. בכל פעם שנקיש T תבוצע פקודה אחת בלבד.

נעקוב אחר שלבי ההרצה הזו:

לאחר שנקיש T בפעם הראשונה תבוצע הפקודה הראשונה, ותוצג הפקודה הבאה, שעדיין **לא** בוצעה: MOV DS,AX.

שוב נקיש T ונבצע את הפקודה: MOV DS,AX. אבל כעת... ראה זה פלא! הפקודה שרשומה כפקודה הבאה לביצוע, הינה: MOV AH,9. כלומר, המחשב "דילג" על הפקודה MOV DX,0011. מבט נוסף יגלה שהפקודה כבר **בוצעה**, אך לא הוצגה (זכור שאנו מפעילים את תוכנת DEBUG, ייתכן שבתוכנית ניפוי אחרת תופעה זו אינה קיימת).

כשתקיש שוב על T תבוצע הפקודה: MOV AH,9
הפקודה שתוצג כממתינה היא: INT 21

כעת, עליך לזכור **שאין** להקיש T.

הסיבה: אם תעשה זאת (אתה מוזמן לנסות), תגיע לתוכנית **אחרת**, תוכנית הפסיקה, וכל הקשה על T תבצע כעת הרצת צעד-יחיד של תוכנית הפסיקה (ייתכן שבתוכנית הבדיקה שלך תופעה זו אינה מתרחשת). אם מסיבה כלשהי הגעת לתוכנית הפסיקה, הדרך הנוחה לחזור לתוכנית שלך היא להתחיל מההתחלה, כלומר:

- לכתוב Q ולצאת מתוכנת הניפוי.
- להיכנס שוב לתוכנת הניפוי.

מכאן למדנו - **אין לבצע הרצת צעד-יחיד כאשר עומדים לפני ביצוע פסיקה.**

ניתן להריץ את החלק המבצע את הפסיקה בתוכנית על ידי הפקודה G. אם נכתוב: "G=0 C" התוכנית תורץ עד לביצוע הפסיקה הראשונה, כי הפקודה הזו מריצה את התוכנית מכתובת 0, כתובת התחלת התוכנית ועד לכתובת C, לא כולל.

נוכל גם לכתוב את הפקודה "G C" שתגרום להרצת התוכנית מהכתובת שאוגר מצביע ההוראות IP, מצביע עליו כעת, ועד לכתובת C (לא כולל). הרצה כזאת הינה הרצה עם **נקודות עצירה** (break points).

דוגמה 2: קליטת מקש והצגתו

נציג תוכנית נוספת, הקולטת מקש ומציגה אותו על המסך. לפניך הפקודות הראשונות של התוכנית, כפי שהיא מוצגת ב-DEBUG:

```
MOV AX,10F4  < כאן יירשם מספר אחר
MOV DS,AX
MOV AH,1
INT 21H
MOV DL,AL
MOV AH,2
INT 21H
MOV AX,4C00
INT 21H
....
```

לתוכנית זו מספר שלבים:

1. קליטת מקש (באמצעות פסיקה).
2. הצגת המקש הנקלט (באמצעות פסיקה אחרת).
3. סיום ויציאה ל-DOS (באמצעות פסיקה שלישית).

נניח שהרצנו את כל התוכנית (G=0 14), והיא אינה פועלת מסיבה לא ידועה. כדי לאתר את מקור הבעיה, נבדוק כיצד היא פועלת בכל שלב (פקודה, או מספר פקודות) בתוכנית שלנו, ורק לאחר שנמצא כי השלב מבוצע כראוי, נעבור לבדיקת השלב הבא:

1. נכתוב "G=0 9", כדי להריץ את התוכנית עד לכתובת 9 (לא כולל). כלומר מבצעים את שלב קליטת המקש.

   המחשב ייעצר, והדבר עלול בטעות להתפרש כמחשב "תקוע", וימתין ללחיצה על מקש כלשהו. לאחר שנקיש על מקש כלשהו, נוכל לראות שקוד ASCII שלו הועבר לאוגר AL (חצי ימני של אוגר AX).

   אם נקיש למשל על הספרה 8, יועבר קוד הקסדצימלי 38. מקש רווח יגרום לקוד 20, מקש Enter לקוד 0D, וכן הלאה. שים לב, שזוהי דרך טובה למצוא את קוד ASCII של המקשים!

   אם שלב זה אינו מבוצע כמצופה, יש לבדוק מהי הבעיה בחלק זה של התוכנית. אם השלב עבר בהצלחה, יש לעבור לבדיקת השלב הבא.

2. נכתוב את הפקודה "C F" שתגרום להרצת התוכנית מהכתובת הנוכחית ועד לנקודת-העצירה, הכתובת F.

   כתוצאה מכך, עלינו לקבל על המסך את הצגת המקש שקלטנו. אם שלב זה לא מתבצע כנדרש, יש לבדוק את הפקודות המטפלות בחלק זה של התוכנית.

לסיכום: כדי להריץ תוכניות המכילות פסיקות, נשתמש בהרצה עם נקודות-עצירה ולא בהרצת צעד-יחיד.

סיכום עיקרי הפסיקות

1. פסיקה לסיום תוכנית ויציאה למערכת ההפעלה:

```
MOV AX,4C00H
INT 21H
```

2. פסיקה להצגת הודעה על המסך:

א. יש לרשום משתנה המכיל הודעה. ההודעה תהיה בין גרשיים ובסופה, **בין** הגרשיים, יירשם סימן דולר. לדוגמה: MAMA DB 'POPYE LOVES OLIVE $'.

ב. כדי להציג את ההודעה הקודמת על המסך, יש לכתוב את הפקודות האלו:

```
MOV DX,OFFSET MAMA
MOV AH,9
INT 21H
```

3. פסיקה לקליטת תו מהמקלדת:

א. ללא הד (echo, ללא הצגת התו הנקלט על המסך):

```
MOV AH,7
INT 21H
```

ב. עם הד (התו יוצג במסך):

```
MOV AH,1
INT 21H
```

קוד ASCII של המקש הנקלט יירשם באוגר AL.

4. פסיקה להצגת תו בודד:

א. מציבים באוגר DL את קוד ASCII של התו. ניתן לעשות זאת בשתי דרכים. ניקח לדוגמה את ההצגה של הספרה 9:

דרך א': MOV DL,39H

דרך ב': MOV DL,'9'

ב. נכתוב את הפקודות:

```
MOV AH,2
INT 21H
```

תרגילים

כתוב תוכנית לכל אחד מהתרגילים הבאים והרץ אותה לבדיקה. **בהצלחה!**

1. מציגה על המסך את שמך.
2. קולטת תו בודד מהמקלדת ומציגה אותו חמש פעמים על המסך.
3. מציגה על המסך את הספרות 3 עד 7 בשורות נפרדות.
4. קולטת חמש ספרות ומציגה את הספרה הגדולה ביותר.

5. קולטת מקש בודד ומציגה הודעה "זוהי ספרה", אם המקש הנלחץ הוא ספרה (0-9).

6. קולטת רצף של שלושה מקשים המהווים "קוד סודי". אם הקוד הנקלט אינו 4E7, יבוצע Reset באמצעות פסיקה. אם הקוד נכון תירשם הודעה "הקוד נכון, המשך בבקשה".

7. קולטת שתי ספרות. אם ההפרש בין הספרה השנייה לראשונה גדול מ-3, תירשם הודעה "Big". במקרה אחר, תירשם הודעה "Small".

8. קולטת מקשים שוב ושוב, עד להקשה על מקש Enter. התוכנית תציג הודעה "True", אם מספר המקשים שהוקשו (לא כולל Enter) גדול מ-6.

9. מציגה על המסך את התו ! מספר פעמים ללא הפסקה, עד להקשה על מקש כלשהו (רצוי להשתמש בלולאות השהייה לצורך השהיית התצוגה).

10. מציגה על המסך קו בעמודה המרכזית לאורך המסך, מהשורה הראשונה ועד לשורה התחתונה.

11. קולטת תו מהמקלדת, לאחריו קולטת ספרה, ולבסוף מציגה את התו הנקלט מספר פעמים אשר שווה לערך הספרה שנקלטה.

12. קולטת ספרות שוב ושוב, עד לקליטת מקש שאינו ספרה.

התוכנית תציג את הספרה הגדולה ביותר שנקלטה.

13. קולטת מספר דו-ספרתי המציין שנה נוכחית, לאחר מכן קולטת מספר דו-ספרתי המציין שנת לידה, ומציגה על המסך את הגיל המחושב.

14. מציירת את התו בעל קוד ASCII 254. על גבי המסך, במקום אחד ימינה עבור כל לחיצה על מקש R, שמאלה עבור כל לחיצה על מקש L, למעלה עבור לחיצה על U ולמטה עבור לחיצה על D.

לחיצה על מקש Q תגרום לסיום התוכנית. בתחילת העבודה הסמן יהיה בפינה השמאלית העליונה של המסך.

15. יוצרת תנועה של התו (בעל קוד ASCII 206) לאורך המסך (אנימציה).

לפניך הנחיות לפתרון התרגיל. התנועה נוצרת על ידי סדרת פעולות:

- הצגת התו במקום מסוים על המסך.
- ביצוע השהייה קצרה.
- הצגת תו רווח **באותו** מקום, אשר יגרום למחיקת התו.
- שינוי מיקום הסמן אל שורה אחת נמוך יותר באותה עמודה.
- חזרה על הפעולות עד לשורה תחתונה.

הערה: בשני התרגילים האחרונים תוכל לבחור תווים אחרים כרצונך.

7

כתובות זיכרון

שיטת הכתובות במחשב האישי

התוכניות שלנו והנתונים שהן פועלות עליהם נמצאים בזיכרון המחשב הקרוי **RAM** (Random Access Memory), אשר ניתן לכתיבה וקריאה. הזיכרון הבסיסי של המחשב מכיל תאי זיכרון בגודל בית. **בית** (byte) הוא יחידת המידע הקטנה ביותר שאנו יכולים לפנות אליה בזיכרון המחשב והוא מכיל 8 סיביות. אפשר לפנות גם לסיביות בודדות כאשר הנתון נמצא באוגר, אך על כך נלמד בהמשך.

לכל תא זיכרון יש **כתובת** (address), אשר מציינת את מיקומו בזיכרון. על שיטת הכתובות במחשב ופנייה לנתונים, נלמד בפרק זה.

כדי להעתיק לאוגר DL את תוכן תא 2000H, אנו כותבים:

```
MOV BX,2000H
MOV DL,[BX]
```

אילו פקודות נכתוב, כדי להעתיק ל-DL את תוכן תא 10000H?

ניסיון לבצע זאת על ידי הפקודה MOV BX,10000H יגרור הודעת שגיאה, שהרי המספר 10000H גדול מדי עבור אוגר BX (או עבור כל אוגר אחר).

מפתחי המיקרומעבד פתרו את הבעיה על ידי הגדרה האומרת ש**כל כתובת** של תא זיכרון תהיה מורכבת משני חלקים:

1. **כתובת בסיס** (base address).
2. **היסט** (offset).

הבסיס הוא ערך בגודל מילה (16 סיביות) שיסמן **כתובת התחלה**. כתובת התחלה דרושה למספר תפקידים, שלכל אחד מהם הוקדש אוגר נפרד:

1. אוגר CS Code Segment כתובת ההתחלה של הקוד (התוכנית).
2. אוגר DS Data Segment כתובת ההתחלה של הנתונים.
3. אוגר ES Extra Segment כתובת ההתחלה של נתונים נוספים.
4. אוגר SS Stack Segment כתובת התחלה של המחסנית.

ההיסט הוא המרחק בבתים (תאי זיכרון) שבין כתובת הבסיס, ועד לתא הרצוי.

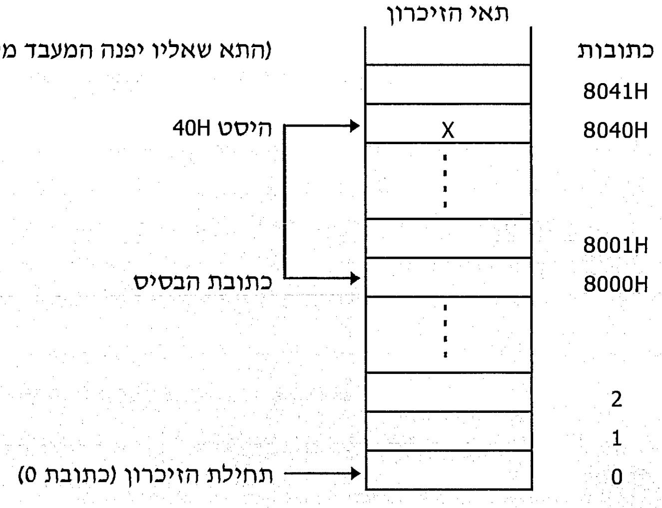
כתובת הזיכרון שאליה פונה המיקרומעבד מחושבת על פי הנוסחה הבאה:

היסט + 10H * כתובת בסיס

דוגמאות:

1. כתובת הבסיס הינה 800H וההיסט הינו 40H. הכתובת של תא הזיכרון שאליו יפנה המיקרומעבד:

800H*10H + 40H = 8000H + 40H = 8040H



2. כתובת הבסיס 1030H, היסט 80H. הכתובת בפועל:

1030H*10H + 80H = 80H + 10300 = 10380

3. כתובת הבסיס 40H, ההיסט 8H. הכתובת בפועל:

40H*10H + 8H = 400H + 8H = 408H

4. כתובת הבסיס 20H, היסט 208H. הכתובת בפועל:

20H*10H + 208H = 200H + 208H = 408H

שתי הדוגמאות האחרונות מיועדות להדגים, שכדי לפנות לתא מסוים אפשר לנצל מיגוון כתובות בסיס והיסט:

כתובת בסיס 40H והיסט 8H (דוגמה 3) גורמות למיקרומעבד לפנות לאותה כתובת, כמו במקרה של בסיס 20H והיסט 208H (דוגמה 4).

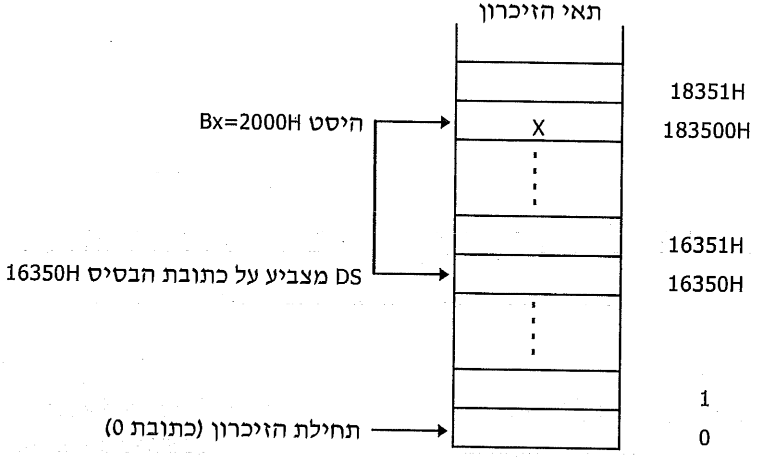
לאיזה תא פונה המיקרומעבד, כאשר רושמים בתוכנית את הפקודות הבאות:

```
MOV BX,2000H
MOV DL,[BX]
```

ההיסט הינו הערך 2000H (כל ערך של אחד מהאוגרים המצביעים, הינו **היסט**). הבסיס, במקרה זה, הוא ערך אוגר DS (בהמשך נסביר מדוע).

נניח למשל, שבתוכנית שלנו ערך אוגר DS הינו 1635H. כתובת התא שאליו יפנה המיקרומעבד היא:

1635H*10H + 2000H = 16350H + 2000H = 18350H



(התא שאליו יפנה המעבד מסומן ב-X)

רגע אחד! כל התוכניות שכתבנו עד כה אינן נכונות?! האם בכל פעם שכתבנו בתוכנית תא 2000H, פנינו למעשה לתא 18350H?

התשובה היא, שאין כל פגם בתוכניות.

כאן המקום להבהיר את העניין, אך לפני כן נכיר שני מושגים נוספים:

1. **כתובת פיסית** (physical address).
2. **כתובת לוגית** (logical address).

כתובת פיסית הינה הכתובת שאליה יפנה המיקרו בפועל. כתובת לוגית הינה ההיסט.

מכאן שניתן לרשום: **כתובת פיסית = כתובת לוגית + בסיס * 10H**

בכל התוכניות שכתבנו עד עתה, התייחסנו לכך שכל הכתובות הינן כתובות לוגיות (היסט בלבד), ולא כתובות פיסיות (הכתובות בפועל). נשנן את הכלל הבא:

בכל תוכנית, אם לא נדרש אחרת, כל התייחסות לכתובת זיכרון הינה כאל כתובת **לוגית**. לכן, כל התוכניות שפתרנו עד כה הינן כשורה. בכולן לא נדרשה התייחסות לכתובות פיסיות, ולכן הכתובות הינן לוגיות.

באילו מקרים נרצה לכתוב תוכניות המתייחסות לכתובות פיסיות דווקא?

בכל מחשב PC קיימים מספר תאי זיכרון בכתובות מסוימות, המכילים מידע חשוב. חלק מתאים אלה מכיל מידע על פסיקות וכמה מהם קשורים למסך.

לשם הדגמה נציג מספר תאים כאלה:

- התא בכתובת פיסית 449H מכיל מידע על סוג המסך הנמצא בשימוש.
- התאים בכתובת פיסיות 408H ו-409H מכילים מידע על כרטיס המדפסת.
- תאי הזיכרון בכתובות פיסיות B8000H ואילך, מטפלים בהצגת תווים וצבעים ברוב המסכים.

כיצד מטפלים בכתובות פיסיות? ניתן לפנות לכתובת פיסית באמצעות DEBUG בלבד, או באמצעות תוכנית. נלמד לעשות זאת.

פנייה לכתובת פיסית באמצעות תוכנת ניפוי (כדוגמת DEBUG)

כדי לקבל את ערך התא בכתובת **הלוגית** 1000H נכתוב כך: D 1000 1000

כאשר אנו רוצים בתא שכתובתו **הפיסית** היא 1000H, נוכל לרשום: D 0:1000 1000

כלומר, קיבלנו את הכתובת על ידי חיבור של בסיס 0 והיסט 1000H. על פי נוסחה זו, הכתובת הפיסית היא 1000H.

ניתן, כמובן, לרשום צירוף אחר של בסיס והיסט, שייתנו כתובת פיסית זהה. למשל:

D 100:0 0

הסבר: בסיס 100H והיסט 0 מייצגים גם כן את הכתובת הפיסית 1000H.

לסיכום, צורת הרישום ב-DEBUG מאפשרת לפנות לכל כתובת פיסית רצויה.

תבנית הכתיבה: **היסט : בסיס**

נראה מספר דוגמאות לטיפול בכתובות פיסיות באמצעות DEBUG:

1. נציב את הערך 55H בתא שכתובתו הפיסית B8000H.

   פתרון אפשרי א': נרשום את ההוראה E B800:0 55

   פתרון ב': E B700:1000 55

   פתרון ג': F B800:0 0 55

2. נציב את הערך 33H ב-101H תאי זיכרון, החל מכתובת פיסית B800H.

   פתרון: F B800:0 100 33

3. נציב את הערך FFH בכתובת בעלת בסיס DS והיסט 100H.

   פתרון: E DS:100 FF

4. נציב את הערך 11H בתאי הזיכרון החל מכתובת בעלת בסיס SS והיסט 0, ועד היסט 100H.

פתרון: F SS:0 100 11

תרגילים בכתובות פיסיות (באמצעות DEBUG)

1. בדוק ורשום מהו ערך התא שכתובתו הפיסית 0.
2. מה מכיל התא שכתובתו הפיסית 408H, מה מכיל 409H?
3. הצב את הערך DDH ב- 300H תאים, החל מכתובת פיסית B8000H. מה קיבלת?
4. בדוק אם בכתובת פיסית F0000H מצוי זיכרון ROM (זיכרון לקריאה בלבד). רמז: הבדיקה תיעשה על ידי הצבת ערך בתא זה, ולאחר מכן בדיקה אם הוא אמנם נרשם בתא. אם הוא לא נרשם, זהו זיכרון ROM.
5. ערך תא בכתובת פיסית 410H מכיל מידע על הציוד ההיקפי במחשב. מה ערך סיבית D0?
6. מהו ערך התא שכתובתו הפיסית 449H?
7. מה מכיל תא בכתובת פיסית 44AH?
   תא זה מכיל מידע על גודל המסך. מה, להערכתך, הוא מציין?
8. שני התאים בכתובות פיסיות 413H ו- 414H מכילים מידע של גודל הזיכרון. מה הם מכילים?
9. מהו ערך התא בכתובת פיסית 46FH?
   בדוק מספר פעמים נוספות. האם ערך התא השתנה בכל פעם?
10. בדוק מהו ערך התא שכתובתו הפיסית 417H, ובדוק האם וכיצד משתנה ערך תא זה כאשר:
    - מקישים על מקש Ins.
    - מקישים על מקש Caps Lock.
    - מקישים על מקש Shift בצד ימין של המקלדת.
    - מקישים על מקש Shift בצד שמאל.

פנייה לכתובות פיסיות באמצעות תוכנית

בכל פעם שאנו משתמשים באוגרים BX, SI ו-DI כמצביעים על כתובות תאי זיכרון, הערכים של אוגרים אלה הינם **היסט**, ואילו **הבסיס** הינו תוכן אוגר DS כברירת מחדל (אלא אם נשנה זאת).

לכן, אם נרצה, לדוגמה, להציב את הערך 3 בתא שכתובתו הפיסית 850H, עלינו לבחור בצירוף כזה של בסיס (האוגר DS) והיסט (אחד האוגרים BX, SI או DI), כך שנקבל לפי הנוסחה (היסט + בסיס * 10H) את הכתובת הפיסית 850H.

אם ערך DS יהיה 80H וערך BX יהיה 50H, נקבל:

80H*10H + 50H = 800H + 50H = 850H

בתוכנית נכתוב זאת בשני שלבים:

- הצבת הערך 80H לאוגר DS:

```
MOV AX,80H
MOV DS,AX
```

זכור: אסור להציב ישירות את המספר לאוגר DS, אלא בתיווך של אוגר AX.

- פנייה לכתובת לוגית 50H:

```
MOV DL,3
MOV BX,50H
MOV [BX],DL
```

דוגמה 1

את הבעיה שהצגנו לעיל נפתור עתה בתוכנית אותה נכתוב בשלמותה (001.asm):

```
CODE SEGMENT
   ASSUME CS:CODE
START:  MOV AX,80H
        MOV DS,AX
        MOV DL,3
        MOV BX,50H
        MOV [BX],DL
CODE ENDS
END START
```

הסבר:

ערך אוגר DS הוא 80H. לכן, מיותר לרשום כהרגלנו: ASSUME CS:CODE,DS:CODE והדבר החשוב ביותר, אין צורך לכתוב את צמד הפקודות:

```
MOV AX,CODE
MOV DS,AX
```

שים לב! הפקודות האלו מציבות באוגר DS את כתובת ההתחלה של התוכנית המתחילה בשם CODE. כאשר אנו כותבים פקודות אלו, **אין לנו שליטה** על הערך שייקבע לאוגר DS! ערכו נקבע בידי מערכת ההפעלה, על פי המקום הפנוי בזיכרון. זאת הסיבה שבגללה מופיע ערך אחר במקום המילה CODE, בכל פעם שאנו מריצים תוכניות ב-DEBUG.

דוגמה 2

נכתוב תוכנית שמחליפה בין ערך התא שכתובתו הפיסית 10020H לבין זה שכתובתו 10022H. התוכנית (002.asm):

```
CODE SEGMENT
  ASSUME CS:CODE
START:  MOV AX,1000H
        MOV DS,AX
        MOV SI,20H
        MOV AL,[SI]
        MOV AH,[SI+2]
        MOV [SI],AH
        MOV [SI+2],AL
CODE ENDS
END START
```

הסבר:

אוגר DS מקבל את הערך 1000H ואוגר SI מקבל 20H. על כן, הכתובת הפיסית שהמחשב יפנה אליה תהיה:

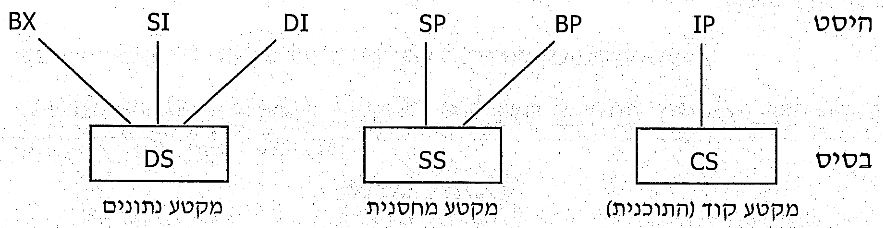
1000H*10H + 20H = 10020H

כאמור, כאשר משתמשים באוגרי ההצבעה SI, BX או DI, הם מהווים **היסט**, ואילו **הבסיס** נקבע על פי הערך של אוגר DS.

אם נשתמש באוגר SP כדי להצביע על תאי זיכרון (השימוש העיקרי שלו הינו במחסנית), הבסיס יהיה הערך של אוגר SS. עבור אוגר IP (המצביע על כתובת ההוראה הבאה לביצוע), הבסיס הוא הערך של אוגר CS.

נסכם את הדברים בטבלה:

אוגרים מצביעים	הבסיס
DI , SI , BX	DS
BP , SP	SS
IP	CS



קיים מספר קטן של פקודות מיוחדות, שעבורן הטבלה אינה מדויקת. פקודות אלו נקראות **פקודות מחרוזת**.

הבסיס שנקבע הינו **ברירת המחדל**. כלומר, ניתן לשנות את הבסיס. אם למשל, נרצה שאוגר BX יהיה היסט והבסיס יהיה CS (במקום DS), נכתוב להם **קידומת** שתציג זאת!

לדוגמה, כדי להעביר לאוגר DL את תוכן התא שנמצא בהיסט 54H החל מבסיס CS, נכתוב כך את הפקודות:

```
MOV BX,54H
MOV DL,CS:[BX]
```

דוגמאות נוספות

1. נציב את הערך 77H בתא שנמצא בהיסט 660H, החל מבסיס SS:

```
MOV AL,77H
MOV DI,660H
MOV SS:[SI],DL
```

2. נציב את ערך תא שנמצא בהיסט 21H מבסיס ES, אל תוך אוגר CH.

```
MOV BX,21H
MOV CH,ES:[BX]
```

3. נציב את הערך 0 בתא שכתובתו הפיסית 30000H.

```
MOV AX,3000H
MOV ES,AX
MOV SI,0
MOV AL,0
MOV ES:[SI],AL
```

הארות והערות

1. אין לשנות את ערך אוגר CS! ההוראה MOV CS,AX תציג שגיאת הידור, ולכן אין לשנות את CS בדרך זו. כמו כן, מומלץ לא לשנות את ערך אוגר SS.

2. רצוי להשתמש באוגר ES, כדי להגיע לכתובת פיסית. לדוגמה, כדי להעתיק את תוכן התא שכתובתו הפיסית 267H, נכתוב:

```
MOV AX,0
MOV ES,AX
MOV DI,267H
MOV AL,ES:[DI]
```

3. השיטה של שינוי הבסיס על ידי קידומת נקראת גם **אילוץ מקטע**.

4. כתובות פיסיות ב-DEBUG. למעשה, בכל פעם שרשמנו הוראות בפורמט של D 1234:0, התייחסנו לתבנית:

היסט : בסיס

הערך משמאל הוא כתובת הבסיס של הפקודות, או הנתונים, ואילו המספר מימין, אשר מתחיל ב-0 מהווים את ההיסט.

תרגילים בכתובות פיסיות

1. כתוב תוכנית המציבה את תוכן התא שכתובתו הפיסית 408H בתוך אוגר DL, ואת תוכן תא 409H באוגר DH.
2. כתוב תוכנית המציבה את הערך העשרוני 219 ב-300H בתים, החל מכתובת פיסית B8000H.
3. כתוב תוכנית המעתיקה את ערך התא שנמצא בהיסט 5CH לפי בסיס CS, אל תא שנמצא בהיסט 6CH לפי בסיס DS.

כתיבת נתונים בזיכרון

כאשר מציבים מספר בגודל בית (8 סיביות) לתוך תא זיכרון, המספר ייכנס בשלמותו לתוך התא. לדוגמה, נכתוב את הפקודות הבאות:

```
MOV BX,1000H
MOV AL,22H
MOV [BX],AL
```

פקודות אלו יגרמו להצבת המספר 22H בתא זיכרון שכתובתו 1000H (מיותר להזכיר שזו כתובת לוגית).

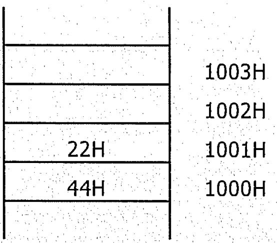
אולם, מה יקרה כאשר ננסה להכניס מספר בגודל מילה (16 סיביות) לתוך תא זיכרון? כאשר ננסה להכניס מספר כזה, הוא יוצב באופן אוטומטי בשני תאי זיכרון **צמודים**: התא שאליו פנינו, והתא הבא אחריו (שכתובתו גבוהה יותר). כלומר, אם נכתוב:

```
MOV BX,1000H
MOV AX,2244H
MOV [BX],AX
```

המספר 2244H יוצב בתא 1000H **וגם** בתא 1001H העוקב לו.

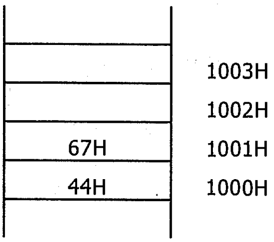
המספר תמיד יירשם על פי הכלל הבא: **הספרות הגבוהות יותר יירשמו בתא הגבוה יותר**. במילים אחרות: הספרות המשמעותיות יותר של המספר תיכתבנה בתא שכתובתו גדולה יותר. בדוגמה זו נקבל:

כתובת התא	הערך
1000H	44H
1001H	22H



אם היינו משתמשים בקטעי התוכניות הקודמות, כדי להציב את הערך 6789H באוגר AX, התוצאה היתה:

- ❖ בתא 1000H יוצב הערך 89H
- ❖ בתא 1001H יוצב הערך 67H



כלל זה נכון גם לנתונים שערכם גדול יותר (כגון מילה-כפולה).

תרגילים

1. בדוק והשב: מהו הנתון **בגודל מילה** הנמצא בתאים שבכתובת פיסית 0 ו-1?
2. כתוב תוכנית שתציב באוגר DX את הנתון בגודל מילה, שנמצא בתאים שבכתובות פיסיות 408H ו-409H. לאחר ההרצה, רשום מה הוצב באוגר DX.
3. בתאים בכתובות פיסיות 413H ו- 414H תמצא את גודל הזיכרון של המחשב. מהו?
4. כתוב תוכנית להעתקת **המילה** הרשומה החל מכתובת 700H אל כתובת 800H.

8

מושגים

פקודות והנחיות

ייתכן וההבחנה בין "פקודה" לבין "הנחיה" אינה נעשית תמיד, אולם מומלץ להכיר את ההבדלים בין שתי קבוצות אלה. כאשר כותבים תוכנית בעורך, אנו כותבים הן פקודות והן הוראות.

- **פקודות** (commands) מתורגמות לשפת מכונה ומבוצעות בשלב הרצת התוכנית.
- **הנחיה** (directives) מיועדות ל"הנחות" את תוכניות MASM ו-LINK כיצד לתרגם את התוכנית לשפת מכונה.

זו הסיבה, שבשלב DEBUG איננו רואים זכר לחלק מהתוכנית שלנו שנכתבה בעורך. למשל, השורות הבאות אינן מופיעות:

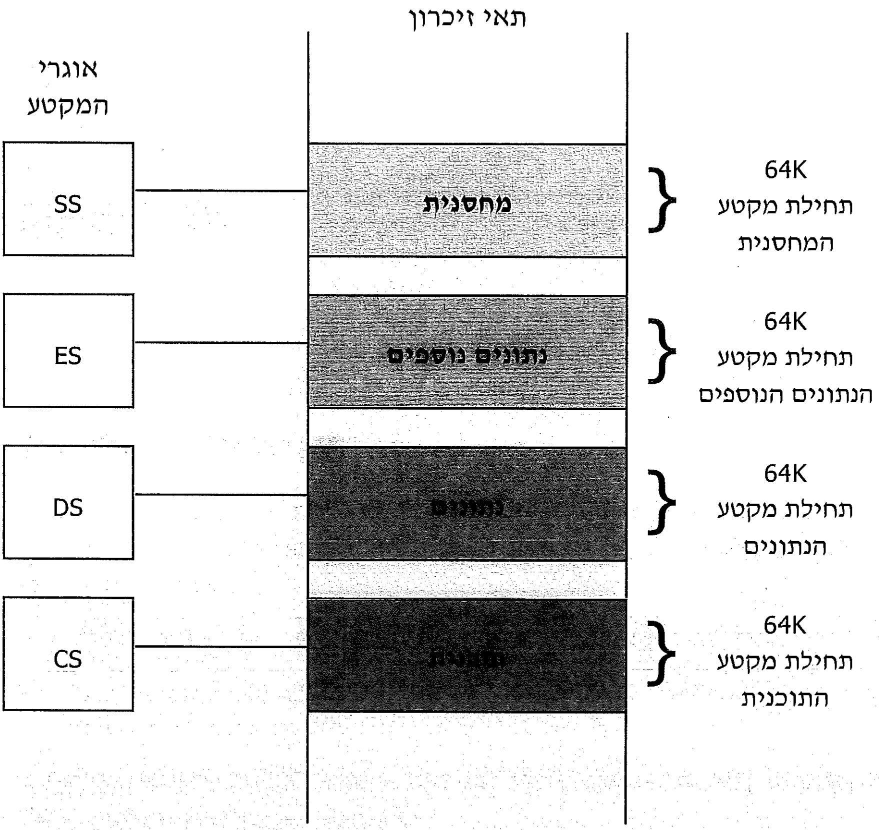
- ASSUME CS:CODE,DS:CODE,SS:STA
- CODE SEGMENT
- END START
- DB 100H DUP (0)

כל אלו הן **הנחיות** ולא פקודות.

סגמנטים - מקטעי זיכרון

סגמנט (segment) הינו **מקטע זיכרון** בגודל 64KB, שהם 65536 בתים (תאי זיכרון). המיקרומעבד מסוגל לעבוד בכל תוכנית עם ארבעה מקטעים:

- **מקטע תוכנית**, מתחיל בבסיס שכתובתו באוגר CS.
- **מקטע נתונים**, מתחיל בבסיס שכתובתו באוגר DS.
- **מקטע נתונים נוספים**, מתחיל בבסיס שכתובתו ב-ES.
- **מקטע מחסנית**, מתחיל בבסיס שכתובתו ב-SS.



העובדה שקיימים ארבעה מקטעים, אינה מעידה על כך שהם תופסים גודל של 4 כפול 64K תאי זיכרון: כל מקטע מתחיל סמוך מאוד לתחילת מקטע אחר. לעיתים, כל המקטעים מתחילים באותה כתובת.

הסדר שבו נמצאים המקטעים בזיכרון, מי הראשון ומי אחריו, תלוי בסדר כתיבתם בתוכנית ובהוראות מיוחדות שהמתכנת יכול לכתוב.

בנושאים אלה נעסוק בהמשך.

תרגילים

1. בדוק וענה: האם מקטע הנתונים ומקטע התוכנית מתחילים באותה כתובת בסיס? ומה לגבי המחסנית? במילים אחרות: האם ערכי CS, DS ו-SS זהים?
2. האם הערכים משתנים כתלות **באורך** התוכנית?
3. האם ניתן לשנות את כתובת הבסיס של המקטע?
4. כתוב תוכנית שבה המקטעים יתחילו **באותה** כתובת זיכרון.

9

אוגר הדגלים

הדגלים ותפקידיהם

אוגר הדגלים, שגודלו 16 סיביות, מכיל 9 סיביות המשמשות **כדגלים** (flags). כל אחת מהסיביות יכולה להיות במצב 0 לוגי ("0") או במצב 1 לוגי ("1"). הדגלים מושפעים מפקודות אריתמטיות ולוגיות בלבד! חלק מהפקודות אינן משפיעות על הדגלים. יש לזכור: מצב הדגלים יישאר באוגר עד אשר אחת (או יותר) מהפקודות יגרמו לשינוי שלהם! את הערכים של חלק מהדגלים אפשר **לקבוע** על ידי פקודות מיוחדות.

הדגלים לסוגיהם

1. **דגל אפס** (zero flag): דגל זה יהיה במצב "1" כאשר התוצאה של הפעולה האריתמטית או הלוגית **האחרונה** היא 0.
2. **דגל סימן** (sign): ערכו כערכה של הסיבית השמאלית (msb) של התוצאה בפעולות חיבור/חיסור ובפעולות לוגיות. כאשר סיבית זו היא "1", התוצאה שלילית, אך ורק אם אנו מתייחסים לתוצאה כאל מספר מסומן.
3. **דגל גלישה** (overflow): מצבו "1" כאשר התוצאה של פעולות חיבור/חיסור היא מחוץ לתחום 128- עד 127 כשמדובר בבית, ומחוץ לתחום 32768- עד 32767 כשמדובר במילה. בפעולות כפל, סיבוב והזזה דגל זה ממלא תפקיד שונה. .
4. **דגל זכור/נשא** (carry): מצבו "1" כאשר יש זכור בפעולה האריתמטית. סיבית שערכה "1" יוצאת מחוץ לאוגר משמאל, או מימין.
5. **דגל זכור-עזר** (auxiliary carry): מצבו "1" כאשר יש העברה של "1" מה-nibble התחתון ל-nibble העליון (nibble = קבוצת 4 סיביות, חצי אוגר).
6. **דגל פסיקה** (interrupt): כאשר דגל זה במצב "0", לא ניתן לבצע פסיקות חומרה.
7. **דגל זוגיות** (parity): בפעולות חיבור/חיסור ופעולות לוגיות הוא במצב "1", אם יש מספר זוגי של סיביות "1" בבית (byte) התחתון (הפחות משמעותי) של התוצאה.
8. **דגל כיוון** (direction): דגל זה קובע את הכיוון עבור פקודות מחרוזת. אלו פקודות מיוחדות המטפלות במחרוזות בזיכרון, נדון בהן בהמשך.

9. **דגל צעד-יחיד** (trap): כאשר הדגל במצב "1" מתבצעת הרצת צעד-יחיד. מכנים זאת גם "מלכודת", מכיון שלוכדים כל פקודה בנפרד כאשר היא מתבצעת.

מיקום הדגלים קבוע בתוך האוגר. הסיבית הימנית ביותר הינה סיבית 0:

סיבית	הדגל
0	זכור (נשא)
2	זוגיות
4	זכור-העזר
6	אפס
7	סימן
8	צעד-יחיד (מלכודת)
9	פסיקה
10	כיוון
11	גלישה

דגל האפס

דגל זה נקרא באנגלית **Zero Flag**, או בקצרה: Z.F. כאשר מתבצע **חישוב** והתוצאה הינה 0, דגל זה עובר למצב "1". לפניך מספר דוגמאות לפעולות שלאחריהן דגל האפס יקבל את הערך "1":

1. לאחר ביצוע הפקודות:

```
MOV AL,4
SUB AL,4
```

2. לאחר הפקודות:

```
MOV CL,0FFH
ADD CL,1
```

הסבר: בתחילה מוצב הערך FFH באוגר CL. ערך זה הוא למעשה 1-. לאחר הוספת 1, התוצאה היא 0, ולכן דגל האפס הוא "1".

3. לאחר הפקודות:

```
MOV CX,1
DEC CX
```

דגל האפס **לא** יהיה "1" כאשר לא מתבצעת פעולה חשבונית. הפעולה הבאה, למשל, **לא** תגרום לכך שדגל האפס יהיה "1", מכיון שזו אינה פעולה אריתמטית!

```
MOV BX,0
```

מקרה נוסף שיגרום ל-"1" בדגל האפס הוא כאשר פעולת **השוואה** מניבה תוצאת **שוויון**. לאחר ביצוע הפקודות:

```
MOV DH,5
CMP DH,5
```

דגל האפס יעלה ל-"1", מכיון שקיים שוויון.

כאשר אנו כותבים את הפקודות:

```
DEC CX
JZ FINISH
```

למעשה, אנו נעזרים בדגל האפס: הפקודה JZ בודקת את מצב הדגל, ועל פי מצבו מחליטה אם לקפוץ לתווית FINISH.

כאשר נכתוב את הפקודות הבאות:

```
CMP DL,4
JE GOOD
```

נבחין שתוכנת DEBUG **שינתה** את הפקודה JE לפקודה JZ. שתי הפקודות הללו **זהות!** כלומר, ניתן לכתוב כל אחת מהן, כי המשמעות זהה.

כאשר מבצעים פעולת השוואה CMP, מתבצעת "הפחתה מדומה". בדוגמה שלפנינו מופחת 4 מערך אוגר DL, ללא שינוי הערך של DL; האוגר היחיד המושפע מביצוע הפקודה CMP הינו אוגר הדגלים. אם DL היה שווה ל-4, תוצאת "ההפחתה המדומה" הינה 0, ולכן דגל האפס יעלה ל-"1".

תרגילים

באילו מהמקרים הבאים יעלה דגל האפס ל-"1"?

א. MOV DL,0

ב. MOV AL,5
SUB AL,5

ג. MOV BL,0FFH
ADD BL,1

ד. MOV CX,36H
CMP CX,'6'

ה. MOV AH,-2
CMP AH,0FEH

דגל הסימן

דגל זה נקרא **Sign Flag**, או בקיצור: S.F. כאשר הערך המתקבל כתוצאה של פעולה חשבונית, או לוגית, הוא ערך שלילי, דגל זה עולה ל-"1". אם הערך חיובי, הדגל יהיה "0".

מספרים שליליים נרשמים במחשב בשיטת **המשלים ל-2**. על עקרונות הייצוג של מספרים שליליים נעמוד להלן.

עיקרי השיטה: הסיבית השמאלית ביותר (MSB - Most Significant Bit - הסיבית הבכירה) מציינת את הסימן: "0" מציין מספר חיובי, "1" מציין מספר שלילי.

דוגמאות

1. הערך 6 הינו חיובי.
2. הערך 78H הינה חיובי.
3. הערך 83H הינו **שלילי** (!).

נסביר זאת: נהפוך את הערך 83H לבסיס 2, כדי לבדוק את הערך של הסיבית החשובה ביותר (משמאל). נקבל את המספר הבינארי 1000 0011.

אנו רואים שהסיבית השמאלית ביותר הינה "1" ועל כן, זהו מספר שלילי!

4. הערך 99H הוא שלילי. נסביר זאת: הערך 99H בייצוג בינארי הינו 1001 1001. הסיבית השמאלית ביותר הינה "1" ועל כן, לפנינו מספר שלילי.

שיטת המשלים ל-2

לצורך הדיון נסביר בקצרה את שיטת המשלים ל-2 (2's complement) לייצוג מספרים שליליים. באמצעות שיטה זו ניתן להפוך ייצוג של מספר חיובי למספר שלילי, ולהיפך. כדי להסביר השיטה נשתמש בדוגמה: נרשום את המספר השלילי 26H-.

1. נכתוב את המספר **החיובי** 26H במספר בינארי (בסיס 2).

   נקבל את הערך הבינארי: 0010 0110.

2. נהפוך, באמצעות שיטת המשלים ל-2, את המספר החיובי הזה למספר שלילי. כדי לעשות זאת, נפעל בסדר הבא:

   - נעתיק החל מצד ימין את כל הסיביות, עד שתופיע לראשונה הסיבית "1", וגם אותה נעתיק.

     בדוגמה שלנו, נעתיק מתוך המספר: 0010 0110

     את הסיביות: 10. . . . . .

     (הנקודות מציינות את הסיביות שלא העתקנו).

   - את שאר הסיביות נהפוך: סיבית "0" תועתק כ-"1" ולהיפך, סיבית "1" תועתק כ-"0":

     נקבל: 1101 1010

3. נייצג את המספר הבינארי שהתקבל, על פי בסיס 16.

   נקבל את הערך **DAH**, אשר מייצג במחשב את הערך השלילי **26H-**.

דוגמאות

1. א. נרשום במחשב את הערך 2-. פתרון:

- ❖ הערך 2 בייצוג בינארי הינו: 0000 0010
- ❖ נכתוב אותו כמספר שלילי בשיטת המשלים ל-2:

  א. נעתיק עד להופעת "1" ראשון: 10

  ב. נהפוך את שאר הסיביות: 1111 1110

- ❖ נתרגם לבסיס 16, והערך שנקבל הוא FEH.

2. מהו הערך המוצב לאוגר AL בפקודה MOV AL,0FFH?

- ❖ כאן אנו מתבקשים לבצע פעולה הפוכה: לרשום מהו הערך של המספר הרשום במחשב.

  נכתוב את הייצוג הבינארי של המספר FFH: 1111 1111

  ניתן לראות שזהו מספר שלילי, כי הסיבית השמאלית היא "1".

- ❖ נשתמש בשיטת המשלים ל-2, כדי להפוך את הערך השלילי לערך חיובי. כאמור, אפשר להפוך מספר שלילי לחיובי באמצעות שיטה זו.

  א. נעתיק עד להופעת-אורח של "1": 1

  ב. נהפוך את שאר הסיביות: 0000 0001

- ❖ נכתוב את התוצאה בייצוג הקסדצימלי, ונקבל את המספר "1". כלומר, הערך 0FFH הינו 1-, מכיון שהתוצאה ההפוכה היתה 1.

בדוק את פתרונך לתרגילים שנית בנושא דגל האפס!

הסבת מספרים מבסיס הקסדצימלי לבסיס בינארי

ראינו שכדי למצוא ערכים שליליים או חיוביים, דרוש ידע בביצוע הסבות מבסיס הקסדצימלי לבסיס בינארי, ולהיפך. בנוסף, לצורך טיפול בפקודות רבות נוספות שנלמד בהמשך, יש לשלוט היטב בפעולות ההסבה האלו.

כדי להסב מספר המיוצג בבסיס הקסדצימלי למספר המיוצג על פי בסיס בינארי, יש להפוך **כל ספרה** הקסדצימלית ל-4 ספרות בינאריות.

לדוגמה, מספר הקסדצימלי 56 יוצג כך בבסיס 2:

- ❖ את הספרה 6 נהפוך ל-4 ספרות בינאריות: 0110
- ❖ את הספרה 5 נהפוך ל-4 ספרות בינאריות: 0101

נצרף את שתי קבוצות המספרים ונקבל את המספר הבינארי: 0101 0110 (הרווח שבין שתי הקבוצות במספר הוא לצורך בהירות הכתיבה בלבד!).

כדי להימנע מהצורך לזכור את הסבת המספרים בשני הכיוונים, תוכל להיעזר בטבלה שלהלן:

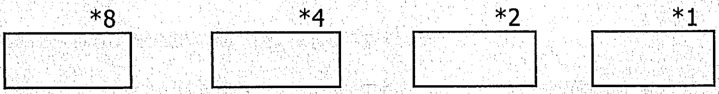
דצימלי בסיס 10	**הקסדצימלי בסיס 16**	**בינארי בסיס 2**
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

אינך צריך לזכור את הטבלה, אולם עליך לדעת לבצע בעצמך את התרגום של כל ספרה הקסדצימלית ל-4 ספרות בינאריות.

השיטה לביצוע פעולה זו: לכל ספרה **בינארית** יש משקל, או ערך מיקומי (כמו לספרות בבסיס עשרוני). במילים אחרות, הספרה הימנית ביותר תוכפל ב-1 (2^0), הספרה השנייה מימין תוכפל ב-2 (2^1), הספרה הבאה תוכפל ב-4 (2^2) והספרה השמאלית ביותר מבין 4 הספרות תוכפל ב-8 (2^3). **סכום** המכפלות של 4 הספרות שווה לערך הספרה ההקסדצימלית.

נבהיר זאת באמצעות דוגמה: כיצד נרשום את המספר הבינארי המייצג את הערך ההקסדצימלי 9?

תשובה: עלינו לחפש מספר בין 4 סיביות (ספרות בינאריות), אשר סכום המכפלות שלו יתן את הערך 9.



נראה שאם נציב 1 ב-"תא 8*" ו-1 ב-"תא 1*", נקבל:

$$1*8 + 1*1 = 9$$

באותו אופן נוכל להסב כל ספרה הקסדצימלית ל-4 ספרות בינאריות. למשל, כדי להסב את הספרה F בבסיס 16, נבצע:

- הספרה ההקסדצימלית F שווה בערכה ל-15. נמלא את הריבועים כך שייתנו סכום מכפלות השווה ל-15.
- נרשום 1 ב"תא 8*". הסכום כעת הינו 8, חסר עוד 7 כדי להגיע ל-15.
  נרשום 1 ב-"תא 4*". הסכום גדל כעת ב-4, ולכן נותר להוסיף 3, כדי להגיע ל-15.
  נרשום 1 ב-"תא 2*" והסכום גדל ב-2, ונותר 1.
  נרשום 1 ב-"תא 1*". כעת הסכום הגיע ל-15.

המספר שקיבלנו: 1111. זהו הייצוג הבינארי של F.

זכור, כל ספרה הקסדצימלית תהפוך בנפרד ל-4 ספרות בינאריות!

תרגילים

1. תרגם את המספרים הבאים הנתונים בבסיס 16, לבסיס בינארי:

א.	2	ו.	A7
ב.	D	ז.	E0
ג.	93	ח.	14C
ד.	25	ט.	D6
ה.	81	י.	125

2. אילו מהמספרים הבאים, שניתנים בייצוג הקסדצימלי (בסיס 16) הם שליליים?

א.	29	ד.	A2
ב.	78	ה.	99
ג.	90	ו.	0F

הסבת מספר בינארי לייצוג הקסדצימלי

כדי להפוך מספר בינארי למספר בייצוג הקסדצימלי, יש להפוך **כל ארבע** סיביות, החל מימין, לספרה הקסדצימלית **בודדת**.

דוגמאות

1. נתון המספר הבינארי 0111 0100 (הרווח הוא לשם בהירות בלבד):
   - 4 הספרות הימניות "0100" מייצגות את הספרה 4 בבסיס 16.
   - 4 הספרות הבאות "0111" מייצגות את הספרה 7.

   לכן, המספר הוא "74" בבסיס 16.

2. נתון המספר הבא בבסיס 2 : 10101010110. מהו הייצוג שלו בבסיס 16?

- רביעיית הספרות הראשונה מימין : 0110 (הספרה 6).
- הרביעיה הבאה : 0101 (הספרה 5).
- הרביעיה הבאה כוללת רק 3 ספרות : 101.

כאשר חסרות ספרות לרביעיה, נשלים אותה ב"אפסים" **משמאל** ונקבל: 0101 (הספרה 5).

המסקנה היא שהמספר הוא 556 בייצוג הקסדצימלי (בסיס 16).

3. מהו הייצוג בבסיס 16 של המספר הבינארי 11010111?

- הרביעיה הראשונה מימין (0111), היא הספרה 7.
- הרביעיה הבאה (1101), היא הספרה D.

הייצוג ההקסדצימלי (בסיס 16) של המספר הוא : D7.

תרגילים

שים לב: כל המספרים מיוצגים בבסיס 16, אלא אם יצוין אחרת.

1. איזה מספר גדול יותר : 78 או 88?
2. איזה מספר נקבל (בבסיס 16), אם נחסר 2 מ-0?
3. כיצד יוצג המספר השלילי 45H-?
4. מה התוצאה של החישוב : FE+5?
5. מה קורה כאשר מוסיפים למספר 7F את המספר 2?
6. כיצד ייתכן שהמספר הבינארי 1100 הינו חיובי, למרות שהסיבית השמאלית ביותר שלו הינה "1"?
7. תן דוגמאות לפקודות לוגיות, הגורמות לדגל הסימן לעלות ל-"1".

דגל הזכור (נשא)

דגל הזכור (נשא) נקרא גם **Carry Flag**, או בקיצור : C.F. הדגל עולה ל-"1", כאשר בפעולה חשבונית או פעולה לוגית קיים **זכור** (carry), או **לווה** (borrow). כלומר, נוצר מצב שבו סיבית "1" "יוצאת" מחוץ למסגרת המקום המוקצב למספר. נראה זאת בדוגמאות שבהמשך.

דוגמאות

א. הפקודות הבאות יגרמו לדגל הזכור לעלות ל-"1":

```
MOV DL,0FFH
ADD DL,1
```

כדי להבין מדוע יש זכור, יש להפוך את המספרים לבסיס 2:

- המספר FF בייצוג בינארי: 1111 1111
- המספר 1 בייצוג בינארי: 0001 0000

נחבר שני מספרים אלה. נזכיר שבפעולת חיבור מתקיימים הכללים הבאים:

חיבור הערכים	תוצאה	זכור
0 + 0	0	0
0 + 1	1	0
1 + 1	0	1
0 + 1 + 1	0	1
1 + 1 + 1	1	1

נחזור לחיבור שבדוגמה זו:

דגל זכור ← (1) 1111 111 ← שורת זכור

1111 1111

\+

0000 0001

0000 0000 ← תוצאת החיבור

הסבר: מתחילים לחבר מימין, כמקובל. חיבור 1 עם 1 בעמודה הימנית, נותן תוצאה 0 (הנרשמת למטה) וזכור "1", הנרשם למעלה, בעמודה **הבאה**.

בעמודה הבאה מחברים 3 סיביות: את הזכור מהעמודה הקודמת (1), את הסיבית 1 והסיבית 0. התוצאה בעמודה זו: 0. הזכור המועבר לטור הבא "1".

בעמודה האחרונה משמאל, תוצאת החיבור הינה 0, אך לזכור "1" אין מקום! לפיכך, הזכור **נשמר** באוגר הדגלים, בסיבית השמורה **לדגל הזכור**.

ב. יש פקודות המזיזות את תוכן האוגר ימינה או שמאלה (על פעולת ההזזה, shift, נלמד בהמשך). פקודות אלו משפיעות על דגל הזכור.

לדוגמה, אוגר CL מכיל את המספר הבינארי: 1101 1001

פקודת הזזה שמאלה תגרום להזזת כל סיבית שמאלה: 101 0011 1

כך נראה שהסיבית השמאלית ביותר (שערכה 1), **הוצאה** אל מחוץ לאוגר **והועתקה** לדגל הזכור. במקרה זה, דגל הזכור יהיה "1".

אם נבצע הזזה ימינה של המספר הזה: 1110 1000, נקבל: 0 0111 100.

במקרה זה יקבל דגל הזכור ערך "0", כי הסיבית שהוצאה היא 0.

דגל הגלישה

דגל הגלישה נקרא גם **Overflow Flag**, או בקיצור: O.F. כאשר אופרנד היעד, שהוא האוגר או התא המקבלים את התוצאה, אינו גדול דיו כדי להכיל את התוצאה, עולה דגל הגלישה ל-"1".

עניין זה פשוט למדי. נסביר זאת בעזרת מספר דוגמאות.

דוגמה 1

נבחן את הפקודות האלו:

```
MOV CL,7FH
ADD CL,5
```

הפקודות מציבות באוגר CL את הערך 84H, שהוא תוצאת החיבור 7F+5.

האם דגל הגלישה יעלה ל-"1"?

התשובה היא כן! למרות שאוגר CL יכול להכיל את הערך 84H התבצעה גלישה, מכיון שהוספנו למספר החיובי 7FH את הערך 5. במקום לקבל תוצאה של מספר חיובי גדול יותר, קיבלנו מספר שלילי! (הערך 84H מייצג מספר שלילי, כי הסיבית השמאלית ביותר שלו הינה 1). כלומר, אוגר CL אינו גדול דיו, כדי להכיל את התוצאה החיובית האמורה להתקבל, ולכן דגל הגלישה עולה ל-"1".

כאן המקום להסביר עניין חשוב:

- הפקודה MOV AL,84H - מציבה באוגר AL מספר בינארי, ואם נתייחס אליו כאל מספר מסומן, אזי הוא מייצג מספר שלילי.
- הפקודה MOV AX,84H - מציבה באוגר AX מספר חיובי.

הסבר: כאשר מוצב המספר 84H באוגר AL (מחצית אוגר), הוא מכיל את הסיביות: 1000 0100. הסיבית השמאלית ביותר היא 1, ולכן המספר הינו שלילי.

כאשר מוצב המספר 84H באוגר AX (אוגר שלם), נקבל: 0000 0000 1000 0100

ניתן לראות, שהסיבית השמאלית ביותר כעת היא 0, ולכן המספר מוצג כמספר חיובי!

כעת נכתוב כך את הפקודות שבדוגמה:

```
MOV AX,7FH
ADD AX,5
```

פקודות אלו יציבו את הערך **החיובי** 0084H באוגר AX, ועל כן דגל הגלישה במקרה זה יהיה "0".

דוגמה 2

נריץ את הפקודות הבאות:

```
MOV AH,0FFH
ADD AL,3
```

פקודות אלו יגרמו להצבת תוצאת החיבור FFH+3 באוגר AH, שהוא מחצית האוגר בלבד. איזה ערך יכיל אוגר AH? FFH שווה בערכו למספר 1- ולכן, כאשר מוסיפים לערך זה 3, מקבלים את הערך 2. והתשובה היא, שהאוגר AH יכיל את הערך 2.

האם תהיה גלישה? ובכן, הערך 2 יכול להיות מיוצג באוגר AH, כך שלא תהיה גלישה ודגל הגלישה יהיה "0".

דוגמה 3

צמד הפקודות הבא גורם להצבת תוצאת החיבור 81H+3 באוגר DL:

```
MOV DL,81H
SUB DL,3
```

הערך 81H הינו מספר שלילי, מכיון שהסיבית השמאלית ביותר של המספר הינה "1". כדי לדעת מהו ערך שלילי זה, נשתמש בשיטת המשלים ל-2.

המספר 81H בבסיס 2: 1000 0001

תרגום באמצעות משלים ל-2: 0111 1111

כלומר, הערך הינו 7FH-. אם נפחית 3 מערך זה נקבל את הערך השלילי 82H-.

לא ניתן להציג ערך כזה באמצעות אוגר DL, מכיון שאין מקום במחצית האוגר לייצוג המספר והסימן, ולכן תהיה גלישה והדגל יהיה "1".

הצגת הדגלים על ידי DEBUG

ניתן לראות את הדגלים על ידי ביצוע ההוראה **R** (Register) של התוכנה DEBUG: הדגלים מופיעים בשמותיהם הקצרים, כפי שנסביר להלן.

כדי לשנות את מצבו של דגל אחד או יותר נשתמש בהוראה **RF** (Register Flags). כל זוג אותיות לועזיות מציינות את מצב אחד הדגלים. בדרך כלל, האותיות מרמזות על שם הדגל, למשל:

סימון	משמעות הסימון	מצב הדגל
CY	Carry Yes	דגל הזכור במצב "1"
NC	No Carry	דגל הזכור במצב "0"
ZR	Zero	דגל האפס במצב "1"
NZ	No Zero	דגל האפס במצב "0"

לפניך טבלת סימוני הדגלים בתוכנת DEBUG:

	אפס	סימן	זכור	גלישה	כיוון	פסיקות	זכור-עזר	זוגיות
במצב "1"	ZR	NG	CY	OV	DN	EI	AC	PE
במצב "0"	NZ	PL	NC	NV	UP	DI	NA	PO

שים לב, שדגל המלכודת (דגל צעד-יחיד) אינו מוצג על ידי DEBUG.

ניתן לשנות מצב של דגל אחד או יותר. נניח שדגל האפס הינו במצב "1" וברצוננו לשנותו ל-"0". הפעולות שנבצע:

- נכתוב את ההוראה **RF**, שמשמעותה: הצג ואפשר שינוי של אוגר הדגלים. המצב הנוכחי של הדגלים (הסיביות) באוגר הדגלים יוצג ברשימה. כל דגל מתואר על ידי שתי אותיות, לפי הטבלה שלעיל.
- נכתוב את אותיות ההוראה NZ, כלומר את דגל האפס ב-"0", ונקיש Enter.
- נכתוב שוב **RF** ונראה שהדגל השתנה. נסיים בהקשת Enter.

בשיטה זו אפשר לשנות מספר דגלים בפעולה אחת. לדוגמה, כדי לאפס את דגלי הסימן, הפסיקות והכיוון, נכתוב:

- ההוראה: RF.
- סימון מצב הדגלים: DI, PL, UP ו-Enter.

אין חשיבות לסדר הכתיבה של הדגלים שעומדים לשנות!

תרגילים ופתרונות לדוגמה

נציג עתה מספר תרגילים ופתרונות הקשורים לנושא הדגלים. בכמה מהם יוצגו גם פקודות חדשות.

תרגיל 1

המטלה: כתיבת תוכנית לבדיקת תוכן תא הזיכרון שכתובתו 270H.

אם ערכו שלילי, יש להציב 1 באוגר AX.
אם ערכו חיובי או 0, יש להציב 0 באוגר AX.

פתרון א': בתחילה נקבע את אוגר AX כ-0. לאחר מכן, תבוצע השוואה בין ערך התא ל-0. אם הערך אינו קטן מ-0, התוכנית תסתיים. אחרת, יוצב 1 באוגר AX. בתוכנית זו נשתמש **בפקודה JL** (Jump Less: קפוץ אם קטן) ונוסיף תנאי שלילה, Not: **JNL**.

התוכנית (001.asm):

```
CSEG SEGMENT
   ASSUME CS:CSEG,DS:DSEG
MAIN:   MOV AX,CSEG
        MOV DS,AX
        MOV AX,0
        MOV BX,270H
        MOV DL,[BX]
        CMP DL,0
        JNL POS                    קפוץ אם לא קטן
        MOV AX,1
POS:    NOP
CSEG ENDS
END MAIN
```

פתרון ב': למדנו שהמספר 7F (בבסיס 16, כמובן) הינו המספר החיובי הגדול ביותר. לכן, אם נשווה את ערך התא ל-80H ונמצא מספר נמוך יותר, זהו מספר חיובי. כעת נשתמש בפקודות אלו:

```
CMP DL,80H
JL POS
```

נמצא, שעבור הערך 71H למשל, לא תתבצע קפיצה, כי 71H אינו קטן מ-80H! לכן הפקודה **JL** אינה מתאימה. במקומה ניתן לרשום את הפקודה **JB** (Jump Below: קפוץ אם נמוך מ-). פקודה זו מתייחסת למיקום. כך למשל, המספר 74H נקטן מהמספר 99H; והאות "K" נמוכה בערכה מהאות "Z".

נבחן את התוכנית לפתרון זה (002.asm):

```
CSEG SEGMENT
   ASSUME CS:CSEG,DS:CSEG
FIRST:  MOV AX,CSEG
        MOV DS,AX
        MOV AX,0
        MOV BX,270H
        MOV DL,[BX]
        CMP DL,80H
        JB POSIT                   jump if below 80H
        MOV AX,1
POSIT:  NOP
CSEG ENDS
END FIRST
```

פתרון ג׳: נבדוק את דגל הסימן: אם הוא "1", לפנינו מספר שלילי. כדי להשפיע על מצב הדגל, עלינו לבצע פעולה חשבונית. אחת האפשרויות היא להוסיף 1 למספר, ולאחר מכן לחסר ממנו 1. כך יישאר המספר בערכו המקורי ודגל הסימן יכיל "1" עבור ערך שלילי או "0" עבור ערך חיובי.

הפקודה JS (Jump Sign: קפוץ אם דגל הסימן הוא "1") שבוחנת אם התוצאה שלילית, תשמש אותנו למטרה זו. בתוכנית זו נשתמש בתנאי השלילה: **JNS** - אם המספר אינו (Not) שלילי. להלן התוכנית (003.asm):

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
BEGIN:  MOV AX,CODE
        MOV DS,AX
        MOV AX,0
        MOV BX,270H
        MOV DL,[BX]
        ADD DL,1        מוסיפים 1 למספר
        SUB DL,1        מפחיתים 1
        JNS POSIT       אם המספר אינו שלילי
        MOV AX,1
POSIT:  NOP
CODE ENDS
END BEGIN
```

תרגיל 2

כתוב תוכנית המחברת את הערכים של האוגרים AL ו-BL ובודקת אם קיים זכור. אם יש זכור, הצב 1 באוגר CX; אחרת, אפס את האוגר CX.

בתוכנית זו נשתמש **בפקודה JC** (Jump Carry: קפוץ אם קיים זכור). גם כאן, כמו בדוגמה הקודמת, נשתמש בתנאי השלילה Not. התוכנית בשלמותה (004.asm):

```
CODE SEGMENT
  ASSUME CS:CODE
COOL:   MOV CX,0
        ADD AL,BL
        JNC FIN         קפוץ אם אין זכור
        MOV CX,1
FIN:    NOP
CODE ENDS
END COOL
```

הערה לתוכניות הדוגמה:

בוודאי שמת לב, שבכל התוכניות עד כה הוספנו בפקודה ASSUME גם את אוגר DS. הפקודות הראשונות גם אתחלו את אוגר DS:

```
MOV CODE
MOV DS,AX
```

בדוגמה 2 לא עשינו זאת, מכיון שכאשר בתוכנית **אין פנייה לזיכרון**, **אין צורך** לאתחל את אוגר DS. אין זו טעות להוסיף את הפקודות המתייחסות לאוגר זה כפי שעשינו בתוכניות שונות עד כה, אולם ניתן היה להשמיט אותן ללא פגיעה בתוכנית.

תרגילים

1. כתוב תוכנית שתספור כמה מספרים שליליים נמצאים בתאי זיכרון שבכתובות 600H עד 610H. התוצאה תוצב בתא שכתובתו 900H.

2. ציין עבור כל אחת מהפקודות, או מקבוצת הפקודות שלפניך כיצד מושפע אחד מהדגלים:

א.
```
MOV AL,82H
```

ב.
```
CMP AL VOM
SUB DL,36H
```

ג.
```
ADD AL,' '
CMP DL,20H
```

ד.
```
MOV AL,6
ADD AL,31H
CMP AL,'7'
```

ה.
```
MOV AL,0F0H
ADD AL,20H
```

ו.
```
MOV CH,66H
ADD CH,66H
```

סיכום פקודות הקפיצה המותנית הנוספות

- JB (Jump if Below) - קפוץ אם נמוך (75H נמוך מ- 85H).
- JA (Jump if Above) - קפוץ אם מעל (88H גבוה מ-60H לעומת JG : 88H קטן מ-60H).
- JNB (Jump if not Below) - קפוץ אם לא נמוך.
- JNA (Jump if not Above) - קפוץ אם לא מעל.
- JC (Jump if Carry) - קפוץ אם יש זכור (נשא).
- JNC (Jump if not Carry) - קפוץ אם אין זכור (נשא).
- JS (Jump if Sign) - קפוץ אם יש סימן (כלומר, התוצאה שלילית).
- JNS (Jump if not Carry) - קפוץ אם אין סימן (כלומר, התוצאה חיובית).

10

עבודה בסיביות

בפרק זה נציג מספר פקודות מתקדמות, אשר תהיינה דרושות לנו לפתרון בעיות מורכבות יותר. נעסוק בעיקר בפקודות לבדיקה של סיביות באוגר ובפקודות הזזה של סיביות באוגר.

הפקודה MOV עם המאפיינים Byte PTR ו- Word PTR

כדי להציב בתא שכתובתו 1000H את הערך 17H, כתבנו עד כה:

```
MOV DL,17H
MOV BX,1000H
MOV [BX],DL
```

ניתן לבצע את ההצבה בצורה מקוצרת יותר:

```
MOV BX,1000H
MOV BYTE PTR [BX],17H
```

הסבר: כבר למדנו שהפקודה MOV[BX],17H אינה חוקית. אולם, על ידי הוספת **המילים BYTE PTR** ("מצביע על בית"), ניתן להציב ישירות ערך בתא זיכרון ללא צורך בתיווך של אוגר.

דוגמה נוספת - נהגנו לכתוב את הפקודות האלו:

```
MOV CL,12H
MOV SI,370H
MOV [SI],CL
```

ניתן לכתוב במקומן:

```
MOV SI,370H
MOV BYTE PTR [SI],12H
```

אף ניתן לקצר ולכתוב פקודה אחת בלבד:

```
MOV BYTE PTR DS:[370H],12H
```

הקידומת :DS מציינת שהבסיס של הכתובת הינו האוגר DS. במקרה כזה ניתן לכתוב את כתובת התא בתוך הסוגריים המרובעים, מבלי להשתמש באוגר מצביע.

לדוגמה, כדי להציב את המספר 5678H בתאי זיכרון עוקבים 1000H ו-1001H, נהגנו לכתוב:

```
MOV DI,1000H
MOV AX,5678H
MOV [DI],AX
```

הסבר: הפקודה האחרונה מציבה את תוכן AX בזיכרון. מכיון שגודל כל תא זיכרון הינו בית אחד (8 סיביות), הנתון שנמצא באוגר של חצי מילה (שני בתים) מוצב בשני תאים צמודים: הנתון מ-AL (78H) מוצב בתא 1000H והנתון שב-AH (56H) מוצב בתא 1001H. ניתן לכתוב את הפעולה הזו באופן מקוצר:

```
MOV DI,1000H
MOV WORD PTR [DI],5678H
```

הסבר: המילים **WORD PTR** ("מצביע על מילה") מציינות שהפנייה היא למילה המורכבת משני תאים עוקבים.

נוכל לבצע את הפעולה הזו באמצעות פקודה אחת:

```
MOV WORD PTR DS:[1000H],5678H
```

השימוש ב- BYTE PTR וב- WORD PTR אינו מוגבל לפקודות MOV בלבד. כך למשל, כדי להוסיף 1 לתוכן תא 620H ניתן לכתוב:

```
INC BYTE PTR DS:[620H]
```

לדוגמה, אם נרצה להפחית 8 מתוכן תא 460H נוכל לכתוב בצורה מקוצרת:

```
SUB BYTE PTR DS:[460H],8
```

הפקודה AND

כפל לוגי של סיביות

הפקודה AND מבצעת כפל סיביות (כפל לוגי), סיבית אחר סיבית. היא אינה "רואה" את המספר בשלמותו ועל כן, אינה מיועדת לבצע כפל חשבוני!

מהו **כפל סיביות**? לפניך טבלה שמסבירה זאת:

$0 * 0 = 0$
$0 * 1 = 0$
$1 * 0 = 0$
$1 * 1 = 1$

על פי הטבלה ניתן לראות שמכפלת סיביות דומה לכפל של ספרות בודדות. מכפלה של שתי סיביות, שלפחות אחת מהן היא אפס (0), נותנת את התוצאה 0.

מקובל לכנות את הסיביות לפי מיקומן היחסי. על כן, הסיבית הימנית ביותר נקראת **סיבית 0**, או **D0**, לשמאלה נמצאת סיבית **D1** ולאחריה **D2** וכן הלאה.

דוגמה 1

נניח שאוגר AL מכיל את המספר 16H, ואנו כותבים את הפקודה:

```
AND AL,38H
```

פקודה זו יוצרת כפל סיביות בין ערך אוגר AL (שערכו 16H) לבין המספר 38H, ושומרת את התוצאה באוגר AL. כדי לבדוק מה התוצאה, יש להסב את כל המספרים לבסיס 2:

מספר עמודה	7654 3210
ערך התחלתי של אוגר AL (16H)	0001 0110
הערך 38H	0011 1000
תוצאה	0001 0000

הסבר: בעמודה 0 מוכפלת סיבית 0 בסיבית 0, ולכן התוצאה היא 0. בעמודה 1 מוכפלת סיבית 1 בסיבית 0, והתוצאה שווה ל-0. בעמודה 3 התוצאה של כפל 0 ב-1 היא 0. בעמודה 4 מוכפלת סיבית 1 בסיבית 1 והתוצאה במקרה זה היא 1. בשאר העמודות התוצאה היא 0.

בסיום ביצוע הפקודה יכיל האוגר AL את הערך הבינארי: 0001 0000. אם נסב זאת לבסיס 16 (כל 4 סיביות מהוות ספרה הקסדצימלית אחת), נקבל את הערך 10H.

דוגמה 2

לפניך קטע תוכנית:

```
MOV DL,1
MOV AL,35H
AND DL,AL
```

הפקודה האחרונה מבצעת כפל לוגי בין ערכי האוגרים AL ו-DL, ושומרת את התוצאה באוגר DL. כרגיל, נשמרת התוצאה באוגר, או בתא, הרשום משמאל. מה יכיל אוגר DL לאחר ביצוע פקודות אלו?

מספר עמודה	7654 3210
ערך אוגר DL (=1)	0000 0001
ערך אוגר AL (=35H)	0011 0101
תוצאה	0000 0001

הסבר: בעמודה 0 מוכפלת סיבית 1 בסיבית 1, והתוצאה המתקבלת היא 1. בשאר העמודות התוצאה היא 0. מכאן שאוגר DL יכיל את הערך 1.

מיסוך באמצעות הפקודה AND

השימוש העיקרי בפקודה **AND** הוא לצורך בידוד סיבית אחת או יותר. כדי לבודד סיבית אחת בלבד, נאפס את כל הסיביות האחרות, פרט לסיבית המבוקשת. פעולת האיפוס של חלק מהסיביות נקראת **מיסוך** (masking).

דוגמה 1

נכתוב תוכנית המאפסת את כל הסיביות של הנתון בתא 500H, למעט סיבית D7 (הקיצונית משמאל). לפניך התוכנית (001.asm) :

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BX,500H
        MOV DL,[BX]
        AND DL,80H
        MOV [BX],DL
CODE ENDS
END START
```

בתוכנית זו מבוצע כפל סיביות בין תוכן התא לבין 80H. הערך 80H בייצוג בינארי הוא :

סיבית:	D0	D1	D2	D3	D4	D5	D6	D7
ערך:	0	0	0	0	0	0	0	1

כלומר, כל הסיביות D0 עד D6 יתאפסו, ואילו הסיבית D7 תשמור על ערכה. לצורך ההסבר, נניח שתא 500H מכיל את הערך F1H.

ערך התחלתי של התא	1111 0001
כפל לוגי עם 80H	1000 0000
תוצאה	1000 0000

התוצאה המתקבלת היא 80H. למעשה, התוצאה שתתקבל עבור כל ערך של תא 500H תהיה אחת משתיים : 80H או 0 (הסבר מדוע!).

נעיין שוב בפתרון שלנו, בתוכנית :

1. במקום הפקודה : "AND DL,80H" ניתן לכתוב, אם רוצים, את הפקודה :

```
AND DL,10000000B
```

האות B באופרנד השני מציינת שהמספר מוצג בבסיס בינארי.

2. ניתן לקצר את התוכנית לפקודה אחת על ידי שימוש ב-BYTE PTR :

```
AND BYTE PTR DS:[500H],80H
```

דוגמה 2

כתוב תוכנית הבודקת אם הספרה הימנית של תוכן תא הזיכרון בכתובת 210H הינה 5. אם כן, יש להציב 1 באוגר CX ובמקרה אחר, יש לאפס את CX.

הפתרון: כל תא זיכרון מכיל נתון בן 8 סיביות, כלומר: 2 ספרות הקסדצימליות. כדי לבדוק מהו ערך הספרה הימנית, יש לבודד את הספרה מכל המספר. אם נאפס באמצעות הפקודה AND את כל 4 הסיביות של הספרה השמאלית, נקבל מספר שיכיל ספרה שמאלית 0 וספרה ימנית מקורית.

לשם הבהרה, נניח שתא זיכרון מכיל את הנתון 49H.

סיבית	D7	D6	D5	D4	D3	D2	D1	D0
תוכן התא (49H)	0	1	0	0	1	0	0	1
איפוס ספרה שמאלית	0	0	0	0	1	1	1	1
תוצאת כפל לוגי	0	0	0	0	1	0	0	1

הסבר: כדי לאפס את הספרה השמאלית, מציבים 4 אפסים במקום השמאלי ביותר (כל ספרה "תופסת" 4 סיביות). כדי לשמור על ערך הספרה הימנית מציבים 4 ספרות "1" ב-4 הסיביות הימניות. התוצאה, כפי שניתן לראות, היא 09 (או בקיצור: 9). כלומר, בודדנו את הספרה הימנית. המיסוך, או הבידוד, של הספרה נעשה על ידי כפל לוגי עם המספר 0000 1111, שהינו 0FH בייצוג הקסדצימלי. לפניך התוכנית (002.asm):

```
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV CX,0
        MOV DI,210H
        MOV AL,[DI]
        AND AL,0FH
        CMP AL,5
        JNE NO
        MOV CX,1
NO:     NOP
CODE ENDS
END START
```

דוגמה 3

נכתוב תוכנית המחברת את הספרה הימנית של תא 300H עם הספרה הימנית של תא 301H, ומציבה את התוצאה בתא 302H.

לפניך התוכנית (003.asm):

```
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BX,300H
        MOV DL,[BX]
        AND DL,0FH
        MOV DH,[BX+1]
        AND DH,0FH
        ADD DL,DH
        MOV [BX+2],DL
CODE ENDS
END START
```

דוגמה 4

נכתוב תוכנית המאפסת את סיבית הסימן של כל תאי הזיכרון בתחום הכתובות 300H-330H.

פתרון: נאפס את הסיבית השמאלית ביותר (D7) באמצעות הפקודה AND.

לפניך התוכנית (004.asm):

```
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
BEGIN:  MOV AX,CODE
        MOV DS,AX
        MOV SI,300H
        MOV CX,31H
BACK:   AND BYTE PTR [SI],01111111B
        INC SI
        LOOP BACK
CODE ENDS
END BEGIN
```

תרגול הפקודות

אילו מהפקודות שגויות?

1. AND AL,BX < לא חוקי!
2. AND DL,-7 < הדבר אפשרי: הערך 7- יוסב ל-1111 1001.
3. AND SI,44H < חוקי.
4. AND BYTE PTR [BX],445H < לא חוקי!
5. AND CL,CL < הדבר מותר. אין שינוי של ערך אוגר CL, אך הפקודה משפיעה על הדגלים.

תרגילים

1. מה מבצעת הפקודה AND AL,AL? האם לדעתך יש בה שימוש כלשהו?
2. אוגר DH מכיל את הערך 83H. כיצד ישתנה ערכו לאחר ביצוע הפקודה AND DH,16H?
3. האם ניתן לרשום את הפקודה AND, כדי לאפס את כל הסיביות, למעט סיבית D0?
4. מה יכיל אוגר CX לאחר ביצוע התוכנית הבאה:

```
       MOV DX,2
       AND DX,1
       MOV BX,2
       MOV CX,2
       JZ GOOD
       MOV DX,4
GOOD:  NOP
```

5. כתוב תוכנית שסופרת את תאי הזיכרון שבהם ערך סיבית D6 הינו 0. תאי הזיכרון הנבדקים נמצאים בכתובות 608H עד 624H.

   את התוצאה יש להציב בתא שכתובתו 630H.
6. כתוב תוכנית שבודקת אם בכל תאי הזיכרון בתחום הכתובות 1040H עד 1050H הספרה הימנית גדולה מ-9.

   אם כן, התוכנית תסתיים; אם לא, הצב באוגר AH את מספר התאים שבהם הספרה הימנית אינה גדולה מ-9.
7. כתוב תוכנית שמציגה על המסך את הספרה הימנית של תא זיכרון שכתובתו 90H.
8. סיביות D7 ו-D6 של תא הזיכרון בכתובת **פיסית** 410H מכילות מידע על מספר הכוננים במחשב (00 = כונן אחד, 01 = 2 כוננים וכדומה). כתוב תוכנית שבודקת את מצב הסיביות ומציגה על המסך את מספר הכוננים. אל תשכח לתת גם הודעת הסבר למספר שתציג.

הפקודה TEST

הפקודה TEST מבצעת פעולה דומה לפקודה AND, אולם היא **אינה משנה** את ערך אופרנד היעד. לכן, משתמשים בפקודה זו כאשר רוצים לבדוק מצב של סיבית אחת או יותר, מבלי לשנות את הערך הנבדק. את התוצאה מכנים בשם "**תוצאה מדומה**". כיצד נשתמש בפקודה?

למשל, אם נרצה לבדוק את סיבית D2 (השלישית מימין) באוגר DH, נכתוב:

```
TEST DH,00000100B
```

ניתן, כמובן, לייצג את המספר כהקסדצימלי ולכתוב:

```
TEST DH,4
```

לאחר פקודה זו נוכל להשתמש בפקודה **JZ**, כדי לבדוק את מצב דגל האפס: אם הוא במצב "1" (תוצאה 0), הדבר מצביע על כך שהסיבית D2 באוגר DH הינה "0".

נציג **דוגמה** פשוטה לשימוש בפקודה:

```
MOV AL,86H
TEST AL,1
```

נבחן את ביצוע הפקודות האלו:

ערך AL (86H)	1000 0110
כפל לוגי עם 1	0000 0001
תוצאה	0000 0000

ערך התוצאה שהתקבלה הוא 0. תוצאה זו **לא** תוצב באוגר AL, אולם היא תגרום לדגל האפס להיות במצב "1". מכיון שערך האוגר לא השתנה, מכנים את התוצאה בשם "תוצאה מדומה".

דוגמה נוספת:

```
MOV CL,45H
TEST CL,12H
```

ערך CL (45H)	0100 0101
כפל לוגי עם 12H	0001 0010
תוצאה	0000 0000

גם כאן התוצאה אינה משפיעה על תוכן האוגר CL. תוכנו יישאר 45H, אולם דגל האפס יהיה "1" בסיום הפעולה.

תוכניות דוגמה והרצתן ב-DEBUG

דוגמה 1

כתוב תוכנית שבודקת אם סיבית D0 בתא זיכרון שבכתובת 404H הינה "1". אם כן, הצב 55H בתא שכתובתו 405H ואם לא, התוכנית תסתיים.

הפתרון:

כדי לבדוק את מצב סיבית D0, עלינו לבצע TEST עם המספר הבינארי: 0001 0000. מספר זה יגרום לכך שכל הסיביות, למעט הסיבית הראשונה מימין, יהיו במצב "0", ולכן לא ישפיעו על התוצאה.

דגל האפס, שייבדק לאחר מכן, יושפע אך ורק ממצב סיבית D0: אם הסיבית הינה "1", דגל האפס יהיה במצב "0"; אם הסיבית הינה "0", דגל האפס יהיה במצב "1".

נניח שערך תא 404H הינו 27H. נבדוק כיצד ישפיע הדבר על ביצוע TEST:

ערך התא (27H)	0010 1111
TEST עם 0000 0001	0000 0001
תוצאה מדומה	0000 0001

מצב דגל האפס: 0 (התוצאה אינה 0).

נניח כעת, שערך התא הוא 16H:

ערך התא (16H)	0001 0110
TEST עם 0000 0001	0000 0001
תוצאה מדומה	0000 0000

מצב דגל האפס: 1 (התוצאה הינה 0).

ניתן לראות, אם כן, שעל ידי פעולת TEST עם המספר 0000 0001 (הערך 1) ובדיקת מצב דגל האפס לאחר מכן, ניתן לדעת מהו ערך הסיבית D0 .

להלן התוכנית (005.asm):

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        TEST BYTE PTR DS:[404H],1
        JZ ZEHU
        MOV BYTE PTR DS:[405H],55H
ZEHU:   NOP
CODE ENDS
END START
```

הסבר התוכנית:

בפתרון שהוצג השתמשנו בפקודות מקוצרות באמצעות BYTE PTR. למעשה, כאשר משתמשים בפקודה TEST נוח מאוד לעשות כך, מכיון שהפקודה אינה משנה את ערך אופרנד היעד (בניגוד ל-AND, למשל).

הפקודה TEST BYTE PTR DS:[404H],1 מבצעת את פעולת הבדיקה בין ערך תא הזיכרון בכתובת 404H (הבסיס הינו DS וההיסט הוא 404H) לבין המספר 1.

ערך תא הזיכרון לא ישתנה, אך דגל האפס יעודכן על פי התוצאה. באמצעות הפקודה "JZ ZEHU" תתבצע קפיצה לסוף התוכנית, כאשר דגל האפס במצב "1". כלומר, הקפיצה תתקיים כאשר התוצאה הינה 0, משמע שסיבית D0 היא "0". כאשר לא מתבצעת קפיצה, הדבר מעיד שהתוצאה אינה 0, והסיבית D0 היא "1". במקרה זה מוצב הערך 55H בתא שכתובתו הלוגית 405H.

בדיקת התוכנית ב-DEBUG:

א. נכתוב U0 להצגת התוכנית. שים לב שהקידומת :DS אינה מופיעה כעת. הסיבה לכך היא, שזוהי ברירת המחדל ולכן הקידומת "מושמטת" על ידי MASM במהלך התרגום לשפת מכונה. כאשר הקידומת שונה מ-DS (למשל, ES), היא תוצג על ידי DEBUG (נראה זאת בתוכניות אחרות שבהמשך).

ב. אם תנסה להריץ את התוכנית בצעד-יחיד (על ידי הפקודה T), שים לב, שחלק מהפקודות של התוכנית מבוצעות מבלי שיוצגו.

ג. עתה נבדוק את ביצוע התוכנית:

1. בסיבית D0 של תא 404H נציב ערך שאינו מכיל "1".

   למשל, ניתן להציב כל אחד מהערכים הבאים: 36H, F8H, 2AH.

2. נציב ערך שונה מ-55H בתא 405H (מדוע חשוב לעשות זאת?)

3. נריץ את התוכנית.

4. נבדוק שערך תא 405H לא השתנה (ואינו מכיל 55H).

5. בסיבית D0 של תא 404H נציב ערך המכיל "1". נציב למשל: 71H או 93H.

6. נריץ שוב את התוכנית.

7. נבדוק את הערך בתא 405H. עכשיו נקבל את המספר 55H.

דוגמה 2

כתוב תוכנית לבדיקת ערך תא הזיכרון בכתובת **פיסית** 4F0H. אם המספר שלילי (סיבית D7 היא "1") יוצב 1 באוגר CX; אם המספר חיובי, יוצב 0 באוגר CX.

פתרון:

כדי לבדוק את הסיבית D7 (הסיבית השמאלית ביותר) נבצע TEST עם הערך הבינארי 1000 0000 (ערך הקסדצימלי 80H). ערך זה יאפשר למסך את כל הסיביות, פרט ל-D7. לאחר מכן, נבדוק את דגל האפס: אם הדגל הוא "1" (תוצאה 0), הרי ש-D7 הוא "0" (המספר חיובי) ואם הדגל הוא "0" (תוצאה 1), הרי ש-D7 הוא "1" (המספר שלילי).

כדי להגיע לכתובת פיסית 4F0H, נציב 0 באוגר ES ונשתמש בו כבסיס. כאמור, כתובת פיסית מתקבלת על ידי כפל של ערך הבסיס ב-10H והוספת ההיסט.

הנה התוכנית (006.asm):

```
 CODE SEGMENT
   ASSUME CS:CODE
```

מדוע אין חובה לאתחל את DS?
אם אינך זוכר, פנה לסעיף "כתובות זיכרון"

```
 START: MOV AX,0
        MOV ES,AX
        MOV CX,0
        TEST BYTE PTR ES:[4F0H],80H
        JZ NUKVAR
        MOV CX,1
NUKVAR: NOP
CODE ENDS
END START
```

בדיקת התוכנית ב-DEBUG:

1. נציג את התוכנית: תוכל לראות שהפקודה TEST בתוכנית התפצלה לשתי שורות. בתחילה נרשם :ES, המציין שהפקודה **הבאה** מתייחסת לכתובת שהבסיס שלה הינו ES. בשורה שאחריה נרשמת הפקודה ללא ES.

   בכל פעם שבתוכנית יש קידומת שונה מברירת המחדל, יופיע ב-DEBUG הכיתוב שהוסבר לעיל. כלומר, בתחילה יופיע האוגר המציין את כתובת תחילת המקטע (אחד מהאוגרים CS, DS, ES, SS) ולאחר מכן מוצגת הפקודה.

2. נציב ערך חיובי בתא. למשל, המספר 66H: E 0:4F0 66
3. נריץ את התוכנית ונבדוק שאוגר CX מכיל 0.
4. נציב ערך שלילי, למשל: 90H, 82H, DAH.
5. נריץ את התוכנית ונבדוק שאוגר CX השתנה ל-1.

תרגילים

1. מה מבצעת כל אחת מהפקודות האלו:

```
a. TEST BYTE PTR DS:[1803H],1
b. TEST BYTE PTR DS:[640H],7
c. TEST WORD PTR DS:[709H],8000H
d. TEST WORD PTR ES:[5],5555H
```

2. כתוב תוכנית שבודקת אם כל תאי הזיכרון בתחום הכתובות 350H-360H מכילים מספרים זוגיים. רמז: עשה זאת באמצעות בדיקת סיבית D0.
3. כתובת פיסית 417H מכילה מידע על מצב מקשים מסוימים. סיבית D6 מציינת את מצב מקש CapsLock: "1" מציין שהמקש פעיל, "0" מציין שהמקש אינו פעיל.
4. כתוב תוכנית שתציג על המסך הודעה על מצבו של המקש, על פי מצב הסיבית.

הפקודה OR

הפקודה מבצעת חיבור לוגי, סיבית אחר סיבית, מבלי "לראות" את המספר כולו. אין זה חיבור חשבוני! לפניך החיבור הלוגי:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

ניתן לראות, שכאשר ערך של סיבית אחת לפחות הוא "1", התוצאה המתקבלת תהיה "1". לפניך **דוגמה** לשימוש בפקודה OR:

```
MOV BL,69H
OR BL,84H
```

נבחן את התוצאה:

מספר סיבית	D0	D1	D2	D3	D4	D5	D6	D7
ערך BL (69H)	1	0	0	1	0	1	1	0
המספר 84H	0	0	1	0	0	0	0	1
תוצאה	1	0	1	1	0	1	1	1

הסבר: בעמודה D0, החיבור הלוגי של "1" ו-"0" גרם לתוצאה "1". בעמודה D1 החיבור הלוגי של "0" ו-"0" נתן תוצאה "0". כך מבצעים חיבור של כל עמודה **בנפרד**. התוצאה המתקבלת: 1110 1101 (EDH).

דוגמה נוספת:

```
MOV CL,55H
OR CH,0AAH
```

שים לב שהספרה 0 לפני המספר AA אינה מהווה חלק מהמספר.

מספר סיבית	D0	D1	D2	D3	D4	D5	D6	D7
ערך CH (55H)	1	0	1	0	1	0	1	0
המספר AAH	0	1	0	1	0	1	0	1
תוצאה	1	1	1	1	1	1	1	1

בכל אחת מהעמודות האלו, תוצאת החיבור הלוגי הינה "1". על כן, התוצאה המתקבלת היא: 1111 1111 (FFH).

בפקודה OR משתמשים כאשר רוצים "להדליק" (או להעלות ל-"1") סיבית אחת או יותר, למשל: OR AL,1.

פקודה זו גורמת לכך, שסיבית D0 (הימנית ביותר) תהיה במצב "1" בכל מקרה. שאר הסיביות ישארו ללא שינוי. הפקודה OR AL,7, למשל, תגרום לכך ששלוש הסיביות D0, D1 ו-D2 יהיו במצב "1" בסיום הפעולה, כי הערך 7 מייצג את המספר הבינארי "111".

תרגילים לדוגמה

דוגמה 1

כתוב תוכנית להפיכת כל המספרים בבלוק (קטע) הזיכרון שבכתובות 412H-41FH לערכים שליליים כלשהם.

פתרון: נהפוך מספר כלשהו למספר שלילי על ידי העלאת סיבית D7 (סיבית הסימן) שלו למצב "1" (007.asm).

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
FIRST:  MOV AX,CODE
        MOV DS,AX
        MOV SI,412H
ODD:    OR BYTE PTR [SI],80H
        INC SI
        CMP SI,41FH
        JNG ODD
CODE ENDS
END FIRST
```

דוגמה 2

כתוב תוכנית שתשנה את ערכי התאים 790H עד 797H לערכים כלשהם, ובלבד שיהיו שווים או גדולים מ-40H.

פתרון: כדי שכל המספרים יהיו שווים או גדולים מ-40H, נפעל כך:

1. נאפס את סיבית הסימן D7 (כדי שהמספר לא יהיה שלילי).

2. נעלה את סיבית D6 למצב "1". על ידי פעולה זו נגרום לכך שערכו של התא לא יהיה קטן מ-40H (בדוק מדוע!). 008.asm:

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV DI,790H
        MOV CX,8
YALA:   AND BYTE PTR [DI],7FH
        OR BYTE PTR [DI],40H
        INC DI
        LOOP YALA
CODE ENDS
END START
```

תרגילים

1. כתוב תוכנית להפיכת ערכי התאים בכתובות 99H עד 100H לערכים אי-זוגיים כלשהם.
2. כתוב תוכנית להחלפה של ערך סיבית D2 שבתא בכתובת 188H, עם ערך סיבית D2 בתא שבכתובת 199H.
3. כתוב תוכנית לקליטת שתי ספרות מהמקלדת, ביצוע פעולת OR ביניהן והצגת התוצאה על המסך.

הפקודה NOT

הפקודה NOT הופכת כל סיבית "0" לסיבית "1", ולהיפך. לדוגמה:

```
MOV DL,17H
NOT DL
```

ערך DL (17H):	0001 0111
לאחר ביצוע NOT:	1110 1000

דוגמה נוספת: הפוך את הסיביות של הנתון שבתא 460H. נציג שני פתרונות לבעיה זו:

פתרון א'

```
MOV SI,460H
MOV CL,[SI]
NOT CL
MOV [SI],CL
```

פתרון ב'

```
NOT BYTE PTR DS:[460H]
```

הפקודה NEG

הפקודה NEG מבצעת פעולת משלים ל-2. כלומר, פקודה זו הופכת מספר חיובי לשלילי, ולהיפך. לדוגמה:

```
MOV DX,-2
NEG DX
```

לאחר ביצוע שתי פקודות אלו, DL יכיל את הערך 2 (חיובי).

דוגמה נוספת:

```
MOV CL,0FFH
NEG CL
```

בסיום, אוגר CL יכיל את הערך 1 (חיובי), מכיון ש-CL הכיל ערך שלילי (FF).

הפקודה XOR

הפקודה מבצעת פעולת XOR על סיביות בודדות. פעולת XOR של סיבית מסוימת, X עם "1", תגרום **להפיכת** סיבית X. כלומר, אם הסיבית X היתה "1" היא תהיה "0", ולהיפך. כך ניתן להפוך את מצב הסיבית מבלי שנצטרך לדעת מה ערכה!

את תוצאות פקודה זו ניתן להציג במספר דרכים.

א. על פי **נוסחה**: $Y = \bar{A} * B + A * \bar{B}$

כאשר:

Y = התוצאה

A = סיבית ראשונה

B = סיבית שנייה

* = פעולת AND

\+ = פעולת OR

\- = פעולת NOT (הסימן שמעל שם המשתנה)

ב. על פי **טבלת אמת**. טבלת אמת מציגה תוצאה אפשרית עבור כל אפשרות.

סיבית א'	סיבית ב'	תוצאה
0	0	0
1	0	1
0	1	1
1	1	0

ניתן לזכור את הטבלה על פי התכונה הבאה: כאשר הסיביות **שונות** התוצאה הינה "1". ואכן, אחד מהמקרים בהם משתמשים בפקודת XOR, הוא לצורך השוואה בין סיביות.

דוגמאות לשימוש בפקודה

דוגמה 1

כתוב תוכנית שבודקת אם סיבית D1 של אוגר AH זהה לסיבית D1 של AL. אם כן, הצב 1 באוגר DX ואם לא, הצב 0.

פתרון: תחילה נבודד את סיבית D1 של כל אחד מהאוגרים (נעשה זאת באמצעות הפקודה AND: ההנחה היא שאין צורך **לשמור** את ערך המקורי של האוגרים!). לאחר מכן, נשווה בין שתי הסיביות (על ידי XOR); אם דגל האפס יהיה "1" ניתן להסיק שהסיביות זהות.

התוכנית (009.asm):

```
CSEG SEGMENT
   ASSUME CS:CODE

HERE:   MOV DX,0
        AND AL,1
        AND AH,1
        XOR AH,AL
        JNZ NOT_EQU
        MOV DX,1
NOT_EQU:
        NOP
CSEG ENDS
END HERE
```

היי! מדוע DS אינו מופיע?

דוגמה 2

כתוב תוכנית שבודקת באמצעות פקודת XOR, אם ערך אוגר DH זהה לערך של אוגר DL. אם כן, התוכנית תסתיים. אם הערכים אינם שווים, DL יאופס.

התוכנית (010.asm):

```
CODE SEGMENT
   ASSUME CS:CODE
MAIN:   TEST DL,DH
        JZ BESEDER
        MOV DL,0
BESEDER: NOP
CODE ENDS
END MAIN
```

דוגמה 3

כתוב פקודה שגורמת להפיכת סיבית הסימן של אוגר AL.

פתרון: XOR AL,80H

אפשר לכתוב זאת גם בייצוג בינארי: XOR AL,10000000B

הסבר: הסיבית השמאלית ביותר הינה במצב "1", וגורמת להפיכת סיבית D7. שאר הסיביות אינן משתנות.

דוגמה 4

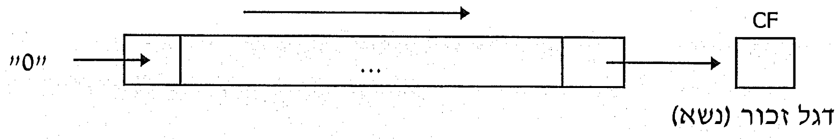
כתוב פקודה שתהפוך את ערך הסיביות D3 ו-D4 של הנתון בתא שכתובתו 646H.

פתרון: XOR BYTE PTR DS:[646H],0011000B

הסבר: רק במקום של הסיביות D3 ו-D4 הוצב "1", כדי שיגרום להפיכת הערך שלהן.

הפקודה SHR

הפקודה SHR **מזיזה ימינה** (Shift Right) את כל הסיביות שבאוגר מקום אחד או יותר, ומוסיפה סיביות "0" מצד שמאל.



נראה עתה מהי פעולת ההזזה ומה מקבלים כתוצאה. נכתוב את הפקודות האלו, למשל:

```
MOV AL,48H
SHR AL,1
```

נתאר עתה את משמעות פעולת ההזזה:

מספר סיבית	D0	D1	D2	D3	D4	D5	D6	D7
ערך AL (48H)	0	0	0	1	0	0	1	0
הזזה ימינה במקום אחד והוספת "0" משמאל	0	0	1	0	0	1	0	0

הוספת "0"

הסבר: סיבית D7 הוזזה ימינה והפכה להיות D6. במקומה נוספה ספרה "0". סיבית D6 שבמקור, הוזזה ימינה גם היא והפכה ל-D5, וכך אירע לכל שאר הסיביות. הסיבית המקורית D0 הוזזה ימינה ונשרה, אך ערכה נשמר בדגל הזכור. זכור זאת!

דוגמה נוספת:

```
MOV AX,1234H
SHR AX,1
```

ערך AX (1234H)   0010 0001 0011 0100

לאחר ביצוע SHR   0000 1010 0001 1010 ⟵ הוספת "0"

כאשר רוצים להזיז את הסיביות ביותר ממקום אחד, יש להשתמש באוגר CL בלבד! באוגר זה אנו מכניסים מראש את מספר המקומות שיש להזיז את הסיביות, כפי שנראה בדוגמאות הבאות.

לדוגמה, כדי להזיז ימינה את הסיביות של אוגר DL ב-3 מקומות, נכתוב כך:

```
MOV CL,3
SHR DL,CL
```

כלומר, תחילה טוענים את המספר 3 באוגר CL ואחר כך מבצעים את ההזזה. במעבד 8088 המקורי אין אפשרות לכתוב את הפקודה SHR DL,3 (בגרסאות המתקדמות של המעבד ניתן לעשות זאת).

דוגמה נוספת: כדי להזיז ב-8 מקומות ימינה את הסיביות בתא שבכתובת 52H, נכתוב:

```
MOV CL,8
SHR BYTE PTR DS:[52H],CL
```

מה יהיה ערך התא לאחר ביצוע פקודות אלו? הזזה של 8 מקומות פירושה איפוס האוגר, כיון שנכנסו 8 סיביות "0" משמאל.

חילוק על ידי פעולת SHR

כפי שנוכל לראות מהדוגמאות שהוצגו, תוצאת ההזזה SHR הינה מספר בינארי שערכו קטן יותר. כמה יותר קטן? המספר המתקבל לאחר הזזה של מקום אחד ימינה גרמה לכך שקיבלנו מספר קטן פי 2, או חלוקה של המספר המקורי ב-2. הזזה של שני מקומות הינה חלוקה ב-4, הזזה של שלושה מקומות הינה חלוקה ב-8, וכן הלאה.

אפשר לסכם כך את פעולת ההזזה ימינה: החלוקה של המספר המקורי תהיה במספר 2^n, כאשר n מייצג את מספר המקומות שהוזזו:

הזזה של מקום אחד: 2^1 או 2

הזזה של שני מקומות: 2^2 או 4

הזזה של שני מקומות: 2^3 או 8

הזזה של שני מקומות: 2^4 או 16

וכן הלאה.

החלוקה בשיטה זו תקפה עבור **מספרים חיוביים בלבד**! כלומר, לא ניתן להשתמש בפעולה זו אם אין יודעים בוודאות שהערך חיובי.

כפי שנראה בדוגמאות הבאות, אנו עוסקים ב**מספרים לא-מסומנים** (unsigned numbers). אלה הם מספרים ללא סימן, שנחשבים לחיוביים. בנושא זה נדון בהמשך.

להבהרת פעולת ההזזה וחלוקת המספר נבחן מספר **דוגמאות**:

- אוגר BL מכיל 40H. נזיז את הסיביות ימינה במקום אחד:

ערך מקורי (40H)	0100 0000
לאחר הזזה ימינה	0010 0000

התוצאה שהתקבלה: 20H. כלומר, מחצית מהערך המקורי של המספר.

- אוגר BL מכיל 26H. נזיז את הסיביות ימינה במקום אחד:

ערך ראשוני (26H)	0010 0110
לאחר הזזה ימינה	0001 0011

התוצאה שהתקבלה: 13H. כלומר, מחצית מהערך 26H.

- אוגר BL מכיל 69H. נזיז את הסיביות ימינה במקום אחד:

ערך ראשוני (69H)	0110 1001
לאחר הזזה ימינה	0011 0100

התוצאה המתקבלת היא 34H. זהו הערך השלם לאחר חלוקה ב-2.

- אוגר BL מכיל 44H. נזיז ימינה ב**שני** מקומות:

ערך התחלתי (44H)	0100 0100
הזזה בשני מקומות	0001 0001

נכנסו שתי ספרות "0"

התוצאה היא 11H. המשמעות: המספר המקורי חולק ב-4!

- אוגר BL מכיל 88H. נזיז ימינה מקום אחד:

ערך התחלתי (88H)	1000 1000
לאחר הזזה ימינה	0001 0001

התוצאה היא 11H. לכאורה, זוהי חלוקה ב-8, אולם המספר 88H הוא שלילי, ואילו 11H הוא חיובי. לא ייתכן שחלוקה כזו תהפוך מספר שלילי לחיובי!

כפועל יוצא, המספר 88H הוא מספר חיובי לפי הגדרת מספרים לא מסומנים, אשר בהם כל המספרים נחשבים חיוביים, ועל כן התוצאה נכונה.

דוגמאות לפקודת ההזזה

בפקודה SHR יש שימוש במקרים רבים. בדוגמאות הבאות יוצגו כמה משימושים אלה.

דוגמה 1

בתאים שכתובתם 600H ו-601H נתונים שני ציוני תלמיד. יש לחשב את הממוצע ולהציבו בתא 602H.

פתרון: את הנתונים נכתוב בייצוג הקסדצימלי ולכן הציון הגבוה ביותר, 100, יוצג כ-64H, שהינו מספר חיובי. כך נוכל להשתמש בפקודה SHR לחלוקת ערכי הציון ב-2. את ממוצע הציונים נחשב בתרגיל זה כסכום של ציון א' מחולק ב-2 ועוד ציון ב' מחולק ב-2. הנה התוכנית (011.asm):

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BX,600H
        MOV DL,[BX]          הציון הראשון
        MOV DH,[BX+1]        הציון השני
        SHR DL,1             חלוקה ב-2 של כל אחד מהציונים
        SHR DH,1
        ADD DL,DH            הסכום = הממוצע
        MOV [BX+2],DL
CODE ENDS
END START
```

הרצת התוכנית ב-DEBUG

1. הצגת התוכנית (על ידי U0), בדיקת כתובת הסיום שלה (זו כתובת הפקודה שאחרי התוכנית) ובדיקת הכתובת שנקבעה ל-DS (זהו המספר הרשום בפקודה הראשונה במקום המילה CODE). לצורך דוגמה זו, נניח שכתוב 1000.
2. הצבת ציונים בתאי הזיכרון 600H ו-601H.

נציב, למשל, את הציון 96 (זהו 60 בבסיס הקסדצימלי) בתא 600H וציון 64 (40 בבסיס הקסדצימלי) בתא 601H:

```
E 600 60
E 601 40
```

3. נריץ ונבדוק את הממוצע בתא 602H: D602 602

שים לב, שגם התוצאה (50H) מתקבלת בבסיס הקסדצימלי.

דוגמה 2

נכתוב תוכנית כדי למצוא את מספר הסיביות שערכן "1", בנתון שבתא 983H. התוצאה תוצב באוגר BH.

פתרון: כאשר מזיזים את הנתון לימין באמצעות SHR, נושרת בכל פעם סיבית אחת מימין. ערך הסיבית, "1" או "0", נשמר בדגל הזכור. כאשר נבדוק את דגל הזכור, נוכל לדעת אחרי כל הזזה, מה היה ערך הסיבית שנשרה. אם נבצע 8 הזזות ימינה, נוכל לבדוק כל אחת מהסיביות של הנתון.

נשתמש **בפקודה JC** (Jump Carry) - קפוץ אם דגל הזכור הוא "1" (012.asm).

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BX, 0                      מונה מספר "1"
        MOV CX,8                       מונה לולאה
BDOK:   SHR BYTE PTR DS:[983],1        הזזה ימינה.
        JNC CONT                       אין זכור = הסיבית היתה "0", אין לספור
        INC BH                         יש זכור = הסיבית הינה "1"
CONT:   LOOP BDOK                      חזרה על התהליך
CODE ENDS
END START
```

הערה: בתוכנית זו הנחנו שניתן לשנות את הנתון בתא הזיכרון. זכור, כי ביצוע הפקודה SHR על הנתון שבתא **משנה** את ערכו של התא. אם אנו צריכים להשתמש שוב בנתון המקורי, עלינו **לשמור** אותו בתא זיכרון אחר לפני שמתחילים בפעולת ההזזה. אפשר גם להעתיק את הנתון לתא זיכרון אחר ובו לבצע את פעולת ההזזה.

בכל מקרה אנו בודקים את כל 8 הסיביות של הנתון. האם צריך לעשות זאת? התשובה היא: **לא**. אם גורם הזמן חשוב לנו, יש לבחור בדרך מהירה יותר, כדי למצוא את מספר הסיביות השוות "1".

פתרון בדרך מהירה יותר

נציג שיטה לחסוך בזמן ביצוע הפעולה הדרושה. לאחר כל הזזה ימינה ובדיקת הזכור, נבדוק אם ערך התא שווה ל-0. אם כן, אין צורך להמשיך בתהליך, כי אין בו סיביות נוספות אשר שוות ל-"1"! זכור שהערך 0 מתקבל כאשר כל הסיביות הן "0", כך: 0000 0000.

הערה חשובה: הדיון על חיסכון בזמן עשוי להיות תמוה. מה החשיבות של מספר מיליוניות שנייה? אולם, עלינו לזכור שלפנינו תרגיל בלבד. בעבודה מעשית, אנו עשויים לחזור על פעולות מסוימות פעמים רבות, ואז בוודאי שיש חשיבות לחיסכון בגלל כפולתו במספר הפעמים.

כדי לבדוק אם ערך התא הוא 0, ניתן להשתמש בפקודה CMP. ניתן גם לעשות זאת בדרך שלפניך, בהנחה שהנתון נמצא באוגר AL:

```
AND AL,AL
JZ FINISH
```

הסבר: ביצוע AND של נתון מסוים עם עצמו אינו משנה את ערכו, אך משפיע על הדגלים. אם הנתון הינו 0, יעלה דגל האפס ל-"1", ותתבצע קפיצה ל-FINISH.

בתוכנית הבאה (013.asm) מיושמת השיטה הזו. נניח שבבעיה זו צריך לשמור את הערך המקורי של תא 983H.

```
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BH,0
        MOV SI,983H
        MOV AL,[SI]
BDOK:   SHR AL,1
        JNC CONT
        INC BH
CONT:   AND AL,AL
        JNZ BDOK
CODE ENDS
END START
```

בדיקת התוכנית ב-DEBUG

1. הצגת התוכנית באמצעות ההוראה U0. יש לשים לב לכתובת סיום התוכנית (16H) ולכתובת הבסיס. לצורך הדוגמה נניח שכתובת הבסיס הינה 1000H.
2. הצבת ערך בתא 983H. נציב למשל את הערך 5, המכיל 2 סיביות "1":

```
E 1000:983 5
```

3. הרצת התוכנית: G=0 16.
4. בדיקת ערך אוגר BH. אלו הן שתי הספרות השמאליות של אוגר BX. תמצא בו את הערך 2.
5. הצבת ערך אחר בתא, למשל FFH. כלומר, כל 8 הסיביות הן "1":

```
E 983 FF
```

זכור, כי לאחר הרצת התוכנית, אין צורך להוסיף את כתובת הבסיס (שערכה 1000 בדוגמה זו).

6. הרצת התוכנית: G=0 16. בדיקה ש-BH מכיל את הערך 8.
7. ניתן לעקוב אחר ביצוע התוכנית באמצעות ההוראה T, ולבחון את יעילותה.

דוגמה 3

יש לכתוב תוכנית שתפריד בין שתי הספרות של הנתון בתא שבכתובת 800H. בתא שכתובתו 802H תוצב הספרה הימנית, ובתא 804H תוצב הספרה השמאלית.

פתרון: נבודד את הספרה הימנית על ידי הפקודה AND. את הספרה השמאלית ניתן לבודד באמצעות הפקודה SHR בלבד.

נסביר תחילה את משמעות המשימה. נניח שתא 400H מכיל את המספר 38H. התוכנית צריכה להציב את הספרה 8 בתא 402H ואת הספרה 3 - בתא 404H. ראוי לשים לב לכך, שכל תא מכיל מקום לשתי ספרות הקסדצימליות. לכן, כאשר אנו מציבים נתון לתא זיכרון, הוא חייב להכיל שתי ספרות! כלומר, גם כאשר רוצים להציב את הספרה 8 בלבד, יש לכתוב ולהציב את הערך 08.

עד כה, לא היינו צריכים לעסוק בכך, מכיון שה"מנגנון" היה אוטומטי: כאשר רשמנו את הפקודות הבאות, הוצב ב-AL הערך 08 (0000 1000 בבסיס בינארי), ותא הזיכרון קיבל את הערך 08:

```
MOV AL,8
MOV [SI],AL
```

נמשיך בפתרון הבעיה: נבודד את הספרה הימנית באמצעות הפקודה AND. נניח לצורך הדוגמה שהנתון נמצא באוגר AL.

```
AND AL,0FH
```

למעשה, מאפסים את 4 הסיביות השמאליות (את הספרה השמאלית), ונותרת הספרה הימנית בלבד. אם AL הכיל את המספר 86H למשל, לאחר הפקודה יהיה ערכו 06H. את הספרה השמאלית נבודד על ידי הזזה ימינה ב-4 מקומות. כך תעבור הספרה השמאלית לצד ימין ואפסים יתווספו משמאל.

לדוגמה: אוגר AH מכיל את המספר 27H, ואנו רוצים שיכיל את הספרה השמאלית בלבד (כלומר 02). הפקודות הבאות יבצעו זאת:

```
MOV CL,4
SHR AH,CL
```

נבדוק את הפעולה:

ערך מקורי של AH (27H)	0010 0111	
הזזה ימינה 4 מקומות והוספת 4 אפסים משמאל	0000 0010	← 4 אפסים

קיבלנו את הערך 02H, המייצג את הספרה השמאלית!

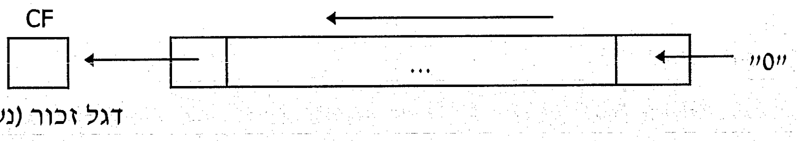
התוכנית (014.asm):

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BX,802H
```

```
        MOV AL,[BX-2]
        MOV AH,[BX-2]
        AND AL,0FH          בידוד ספרה ימנית
        MOV CL,4
        SHR AH,CL           בידוד ספרה שמאלית
        MOV [BX],AL
        MOV [BX+2],AH
CODE ENDS
END START
```

הפקודה SHL

הפקודה SHL **מזיזה שמאלה** (shift left) את כל הסיביות שבאוגר מקום אחד או יותר, ומוסיפה סיביות "0" מצד ימין. נראה עתה מהי פעולת ההזזה ומה מקבלים כתוצאה.



השימוש בפקודה זו זהה לשימוש בפקודה SHR (הזזה לימין). ניתן להשתמש בפקודה זו לצורך כפל ב-2 (או כפולותיו). כך למשל, הזזה של מקום אחד שמאלה הינו כפל ב-2, הזזה של שני מקומות - כפל ב-4, הזזה של 3 מקומות - כפל ב-8 וכן הלאה. למעשה, ההזזה שמאלה הינה כפל ב-2^n, כאשר n הוא מספר המקומות שאנו מזיזים את המספר שבאוגר.

הפקודה SHL, לעומת SHR, מטפלת **במספרים מסומנים** (signed numbers), הן חיוביים והן שליליים. הכוונה לכך שהיא **שומרת על כיוון המספר**.

לדוגמה: נניח שרשמנו את הפקודה MOV BL,-7

הערך 7 בבסיס בינארי	0000 0111
הפיכה ל: 7- (משלים ל-2)	1111 1001

אם נרשום את הפקודה SHL BL,1, נקבל:

ערך התחלתי של BL	1111 1001
הזזה שמאלה במקום אחד והוספת "0" מימין	1111 0010

קיבלנו את המספר השלילי (לפי "1" בסיבית D7): 1111 0010.

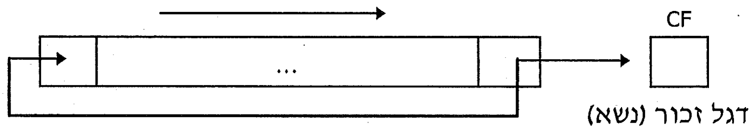
כדי לדעת מה ערכו, נהפוך אותו לחיובי באמצעות משלים ל-2:

המספר השלילי שהתקבל	1111 0010
הפיכתו לחיובי (משלים ל-2)	0000 1110

כלומר: המספר השלילי הינו EH- (ערך 14- בבסיס 10). בוצע כפל ב-2!

הפקודה ROR

הפקודה ROR **מסובבת בכיוון ימין** (rotate right) את כל הסיביות, במקום אחד או יותר. כללי השימוש בפקודה דומים לאלה של SHR, ובכלל זה השימוש ב-CL בלבד, כדי לסובב ביותר מאשר סיבית אחת.



פעולת הסיבוב מתבטאת בכך שהסיבית שיצאה מימין **חוזרת** לסיבית השמאלית ביותר. אין כניסה של "0" מצד שמאל, כמו שראינו בפקודה SHR. הסיבית שיוצאת מימין וחוזרת משמאל נשמרת **גם** בדגל הזכור. הפקודה הזו אינה מבצעת חילוק.

נראה דוגמה לשימוש בפקודה:

```
MOV AL,61H
ROR AL,1
```

ערך התחלתי של AL (61H)	0110 0001
סיבוב לימין במיקום אחד	1011 0000

הסבר: הסיבית הימנית D0 יצאה מימין והפכה להיות הסיבית השמאלית ביותר, D7. כל שאר הסיביות זזו מקום אחד ימינה. לתוצאה שהתקבלה אין משמעות חשבונית.

נראה שתי דוגמאות לשימוש בפקודה.

דוגמה 1

נכתוב תוכנית שתציג על המסך באופן תמידי, בלולאה אינסופית, את הערכים הנוצרים על ידי העלאת סיבית "1" במקום אחר בכל פעם. שאר הסיביות הן "0".

כלומר, בהתחלה יוצג:	1000 0000	(בערכים הקסדצימליים)
לאחר מכן:	0100 0000	
אחר כך:	0010 0000	
....		
... עד להצגת המספר:	0000 0001	
וחוזר חלילה (מהתחלה)	1000 0000	
....		

פתרון: נציב באוגר AL את הערך 80H (1000 0000 בבסיס בינארי), ובאמצעות ROR נזיז את הסיביות ימינה, זו אחר זו.

כדי להציג את ערך אוגר AL (הכולל שתי ספרות), נשתמש בשיטות שלמדנו: בידוד ספרה ימנית על ידי הפקודה AND, ובידוד הספרה השמאלית על ידי הפקודה SHR.

לפניך התוכנית (015.asm):

CODE_SEG SEGMENT	
ASSUME CS:CODE_SEG	
MAIN_PROG:	
MOV DH,80H	המספר הראשון שיוצג. זכור שאין להשתמש באוגר AL לשמירת נתון, כי ערכו משתנה בפסיקות!
HERE: MOV BL,DH	שמירת ערך DH באוגרBL
MOV BH,DH	שמירת ערך DH ב-BH .
AND BL,0FH	בידוד ספרה ימנית ב-BL
MOV CL,4	
SHR BH,CL	בידוד ספרה שמאלית ב-BH
MOV DL,BH	
OR DL,30H	הוספת 30H לספרה: במקרה זה משמשת הפקודה לחיבור (במקומה ניתן להשתמש בפקודה ADD). ההוספה נועדה
MOV AH,2	ליצור קוד ASCII של הספרה (הספרה + 30H).
INT 21H	הצגת הספרה השמאלית
MOV DL,BL	
OR DL,30H	הפיכה לקוד ASCII
MOV AH,2	
INT 21H	הצגת הספרה הימנית
MOV DL,' '	
MOV AH,2	
INT 21H	הצגת רווח בין המספרים
ROR DH,1	סיבוב לימין של המספר
MOV CX,0FFFFH	
DELAY: LOOP DELAY	השהיה בין הצגת המספרים
MOV AH,0BH	
INT 21H	האם נלחץ מקש כלשהו?
CMP AL,0	
JE HERE	אם לא (AL=0), חוזר
MOV AX,4C00H	
INT 21H	אם כן, מסיים
CODE_SEG ENDS	
END MAIN_PROG	

לאחר הרצת MASM ו-LINK, תוכל להריץ את התוכנית ישירות ממערכת ההפעלה.

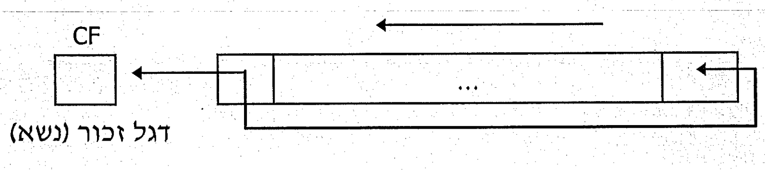
דוגמה 2

התוכנית הבאה מאפשרת להפוך את סדר הספרות בכל תאי הזיכרון שבתחום הכתובות 1500H-1530H. כלומר, הספרה הימנית תעבור לשמאל והשמאלית תעבור לימין. למשל, בתחילה נתון המספר 3AH ולאחר הפעולה נקבל A3H. להלן התוכנית (016.asm) :

```
CSEG SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX          אנו צריכים לטפל ב-31H תאי זיכרון
        MOV DX,31H         הצבת מונה הלולאה DX (ב-CL נשתמש לצורך הסיבוב,
        MOV SI,1500H       ולכן לא יתן להשתמש ב-CX כמונה לולאה)
        MOV CL,4
CONT:   MOV AL,[SI]
        ROR AL,CL          סיבוב 4 מקומות לימין
        MOV [SI],AL        הצבת הנתון החדש בזיכרון
        INC SI
        DEC DX
        JNZ CONT
CSEG ENDS
END START
```

הפקודה ROL

הפקודה ROL **מסובבת את כל הסיביות שמאלה** (rotate left) במקום אחד, או יותר.



להזזה של יותר מסיבית אחת צריך להיעזר באוגר CL. נסתכל בשתי הפקודות האלו :

```
MOV CL,2
ROL BYTE PTR [SI],CL
```

הפקודה ROL תסובב בשני מקומות שמאלה את כל הסיביות של התא שכתובתו מוצבעת על ידי SI. לדוגמה, אם התא הכיל את הערך 43H, הפקודות האלו יגרמו לשינוי הזה :

ערך התחלתי של התא (43H)	0100 0011
סיבוב שמאלה בשני מקומות	0000 1101

הסבר: שתי הסיביות הימניות בתוצאה, D0 ו-D1 הן שתי הסיביות השמאליות של המקור, D6 ו-D7, שנשרו משמאל וחזרו מימין.

תרגילים ופתרונות לדוגמה

בידוד סיביות

מטלה: כתוב תוכנית שתבודד את שלוש הסיביות הראשונות של תוכן תא 1000H. את סיבית D2 יש להציב בתא 1001H את סיבית D1 בתא 1002H ואת סיבית D0 בתא 1003H.

פתרון: נבודד כל סיבית בנפרד בפעולת AND, נזיז ימינה את הסיביות ונציב בתא. מדוע עלינו להזיז ימינה את הסיביות? לצורך ההסבר, נציג דוגמה:

תא 1000Hמכיל את הערך: 0100 1101

בידוד סיבית D2 יגרום לתוצאה: 0000 0100

ניתן לראות שלאחר בידוד סיבית D2, התוצאה היא: 0100 0000 וערכה הוא 4.

בתרגיל זה עלינו להציב בתא את ערך הסיבית, כלומר: 1 או 0 בלבד. לכן, אם נזיז את הערך 0100 0000 ימינה בשני מקומות נקבל את הערך 1, שהינו ערך הסיבית D2. אם ערך הסיבית היה 0, לאחר ההזזה היינו מקבלים את המספר 0, שאף הוא שווה לערך הסיבית.

התוכנית (017.asm):

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BX,1000H
        MOV AL,[BX]
        AND AL,4                         בידוד סיבית D3
        MOV CL,2
        SHR AL,CL                        הזזה לימין
        MOV [BX+1],AL
        MOV AL,[BX]
        AND AL,2                         בידוד סיבית D2
        SHR AL,1                         הזזה לימין
        MOV [BX+2],AL
        MOV AL,[BX]
        AND AL,1                         בידוד סיבית D1
        MOV [BX+3],AL
CODE ENDS
END START
```

הצע פתרון נוסף לתרגיל זה.

חיפוש סיבית בתא זיכרון

מטלה: תא הזיכרון בכתובת 345H מכיל נתון. כתוב תוכנית שתבדוק את מיקום הסיבית הראשונה שערכה "0". החיפוש יחל מימין, ממיקום 0. התוצאה תוצב באוגר DH. נציג לבעיה זו מספר פתרונות.

פתרון א'

נזיז ימינה את הנתון ונבדוק את דגל הזכור: אם הוא יכיל "0", הדבר מצביע שהסיבית שנשרה מימין היא "0". כדי לדעת מהו המקום, נציב בתחילה באוגר DH את הערך 0 ונוסיף לו 1 עבור כל הזזה ימינה. נראה את התוכנית (018.asm):

```
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
STAR:   MOV AX,CODE
        MOV DS,AX
        MOV SI,345H
        MOV DH,0
        MOV AL,[SI]
CHECK:  SHR AL,1
        JNC FOUND
        INC DH
        CMP DH,8
        JNE CHECK
FOUND: NOP
CODE ENDS
END STAR
```

בדוק: איזה ערך יכיל DH כאשר הנתון בתא 345H הוא FFH?

פתרון ב'

כדי לזהות את המקום הראשון של סיבית "0", נבצע:

- פעולת OR בין ערך התא לבין הערך 1.
- השוואה בין הערך המקורי של התא, לבין הערך החדש. אם הערכים **שונים**, הרי שהסיבית D0 היא "0"!
- נמשיך את התהליך באופן דומה עבור שאר הסיביות, עד למציאת "0".

נסביר את השיטה באמצעות דוגמה.

נניח שתא 345H מכיל את הערך 18H:	1000 0001
נבצע פעולת OR עם 1:	0000 0001
התוצאה:	0001 1001

ניתן לראות שהתוצאה המתקבלת **שונה** מהערך המקורי. נסביר זאת.

פעולת OR של "1" עם "1" אינה משנה את התוצאה. אולם כאשר הסיבית היא "0" ומבצעים פעולת OR עם "1", הסיבית משתנה ל-"1". כדי לבדוק את סיבית D0, נעשה OR עם הערך 1, כפי שהודגם. כדי לבדוק את הסיבית D1, נבצע OR עם 2 (ערך בינארי 0000 0010), וכך הלאה.

כך נכתוב את התוכנית בשלמותה (019.asm):

```
CSEG SEGMENT
  ASSUME CS:CSEG,DS:CSEG
OK:     MOV AX,CSEG
        MOV DS,AX
        MOV BX,345H
        MOV DH,0          המקום הראשון הינו 0
        MOV AL,[BX]
        MOV DL,1          בתחילה נבדקת סיבית D0
CHECK:  OR AL,DL          פעולת OR בין הערך המקורי של התא, לבין DL
        CMP AL,[BX]
        JNE FOUND         אם אין שוויון בין ערך התא לבין הערך לאחר
                          OR, הסיבית הנבדקת היא 0
        SHL DL,1          נזיז שמאלה את הסיבית "1", כך שתיבדק
                          הסיבית הבאה
        INC DH            אם לא התבצעה קפיצה לתווית FOUND, יש
                          לעדכן את המיקום
        CMP DH,8          אם המיקום אינו 8, נסיים (המיקום 8 מציין
        JNE CHECK         שאף לא אחת מהסיביות הינה "0")
FOUND: NOP
CSEG ENDS
END OK
```

פתרון ג'

נבצע פעולת AND עם סיבית אחרת בכל פעם. אם התוצאה הינה 0, המסקנה היא שהסיבית היא "0". כדי לא לשנות את ערך תא הזיכרון, נשתמש ב-TEST במקום ב-AND. גוף התוכנית (020.asm):

```
        MOV AL,DS:[345H]
        MOV DH,0
        MOV AH,1
CHECK:  TEST AL,AH
        JZ FOUND
        SHL AH,1
        INC DH
        CMP DH,8
        JNE CHECK
FOUND:NOP
```

החלפה בין ערכים של שתי סיביות

יש לכתוב תוכנית שהופכת בין ערכי הסיביות D3 ו-D5 של תא בכתובת 1800H, כלומר: סיבית D3 תקבל את ערך סיבית D5, וסיבית D5 תקבל את ערך סיבית D3.

אלגוריתם א'

- בידוד כל סיבית בנפרד והצבת שתי התוצאות בשני אוגרים: D3 יוצב באוגר הראשון (1), D5 יוצב באוגר השני (2).
- הזזת D3 למיקום D5 באוגר הראשון, והזזת D5 למיקום D3 באוגר השני.
- חיבור לוגי של אוגר 1 ואוגר 2, כך שאוגר אחד יכיל את D5 (לשעבר D3) ואת D3 (לשעבר D5).
- איפוס שתי הסיביות D3 ו-D5 בתא הזיכרון.
- חיבור לוגי של הסיביות החדשות.

לשם המחשת ההסבר, נראה את הדוגמה הבאה:

סיבית ←	7654 3210
תא הזיכרון מכיל לדוגמה, את הערך 89H:	1000 1001
נבודד את D3 (על ידי AND) ונציב באוגר 1:	0000 1000
נבודד את D5 (על ידי AND) ונציב באוגר 2:	0000 0000
נזיז את D3 של אוגר 1, למיקום D5:	0001 0000
נזיז את D5 של אוגר 2, למיקום D3:	0000 0000
חיבור לוגי באמצעות OR, של 2 האוגרים:	0010 0000
איפוס 2 הסיביות המקוריות במספר 89H:	0000 0001
חיבור לוגי של הסיביות החדשות:	0001 0001

והתוצאה: הסיביות הוחלפו כנדרש.

גוף התוכנית (021.asm):

```
MOV SI,1800H
MOV BL,[SI]               שמירת הנתון באוגר 1
MOV BH,[SI]               שמירת הנתון באוגר 2
AND BL,8                  בידוד סיבית D3 באוגר 1
AND BH,20H                בידוד D5 באוגר 2
MOV CL,2                  הזזה שמאלה למיקום D5
SHL BL,CL
SHR BH,CL                 הזזה ימינה למיקום D3
OR BL,BH                  חיבור לוגי
AND BYTE PTR [SI],0D7H    איפוס D3 ו-D5
OR [SI],BL                חיבור לוגי של הסיביות
```

אלגוריתם ב'

- אם שתי הסיביות זהות ("0" או "1"), אין צורך לבצע החלפה, והתוכנית תסתיים.
- אחרת, התוכנית תהפוך את הערך של כל סיבית (באמצעות XOR)! (הסיבית שהיתה "0" תהיה "1", ולהיפך).

גוף התוכנית (022.asm):

```
        MOV SI,1800H
        MOV AL,[SI]
        AND AL,28H
        JZ FINI
        CMP AL,28H
        JE FINI
        XOR BYTE PTR [SI],28H
FINI:   NOP
```

הסבר: הערך 28H הוא 0010 1000 בייצוג בינארי. הוא מתקבל על ידי הצבת "1" במיקום של D3 ו-D5. כאשר מבצעים פעולת AND בין הנתון לבין 28H, ניתן לקבל מספר תוצאות:

- אם התוצאה היא 0, הרי ששתי הסיביות D3 ו-D5 הן "0", כלומר זהות.
- אם התוצאה היא 28H, הדבר מעיד שערך הסיביות "1", והן זהות.
- כל תוצאה אחרת תסמן אי-זהות של הסיביות.

חשוב וענה: אילו שתי תוצאות נוספות אפשריות (בערכים הקסדצימליים)?

בתוכנית נבדקת התוצאה לאחר פעולת הכפל הלוגי. אם התוצאה היא 0 או 28H התוכנית מסתיימת. במקרה אחר, הופכים את הסיביות באמצעות XOR.

חיפוש נתון בזיכרון

כתוב תוכנית שמחפשת נתון בתוך בלוק בזיכרון. הכתובת להתחלת החיפוש נמצאת בתאים 1000H ו-1001H. גודל הבלוק (מספר התאים לחיפוש) נמצא בתא 1002H והנתון לחיפוש נמצא בתא 1003H. אם הנתון נמצא, יש להציב בתאים 1004H ו-1005H את הכתובת הראשונה שבה הוא מופיע.

כזכור, כל תא זיכרון מכיל נתון בגודל בית בלבד. על כן, כאשר רוצים לשמור נתון בגודל מילה, הוא יוצב בשני תאים צמודים. כתובת זיכרון היא בגודל מילה, ולכן יש לשמור אותה בשני תאי זיכרון.

בתוכנית המבוקשת, התאים 1000H ו-1001H מכילים את הכתובת שאליה יש להגיע, כדי להתחיל בחיפוש. נרשום את הפקודות האלו:

```
MOV DI,1000H
MOV BX,[DI]
```

כתוצאה, נקבל באוגר BX נתון המורכב מתוכן התאים 1000H ו-1001H. נתון זה הינו הכתובת ההתחלתית לחיפוש.

פתרון לדוגמה (גוף התוכנית בלבד. 023.asm):

	MOV DI,1000H	
	MOV BX,[DI]	כתובת התחלת חיפוש
	MOV DI,1002H	
	MOV DX,[DI]	DH מכיל את הנתון לחיפוש
	MOV CH,0	DL מכיל את מספר התאים לחיפוש
	MOV CL,DL	CX מכיל את מספר התאים
	MOV AL,DH	AL מכיל את הנתון לחיפוש
	MOV SI,1004H	
LOOK:	MOV AH,[BX]	
	CMP AH,AL	האם זהו הנתון המבוקש?
	JE YESH	
	INC BX	
	LOOP LOOK	אם לא - המשך חיפוש
	JMP DAY	
YESH:	MOV [SI],BX	הצבת הכתובת שבה נמצא הנתון, בתאים 1004H, 1005H
DAY:	NOP	

משימה לתרגול: שנה את התוכנית, כך שבתאים 1004H ו-1005H תוצב הכתובת **האחרונה** שבה נמצא הנתון המבוקש.

הצגת ייצוג בינארי של מקשים

מטלה: התוכנית הבאה תקלוט מהמקלדת ספרה הקסדצימלית (0 עד F), ותציג על המסך את הייצוג הבינארי שלה. למשל, אם נלחץ על המקש 7 נקבל על המסך 0111 ואם נלחץ A נקבל 1010.

רמזים לפתרון:

1. מכיון שאנו עוסקים בפסיקות, נגדיר מחסנית.
2. לפני קליטת המקש נציג הודעה קצרה, באמצעות פסיקת DOS מסי 9.
3. נקלוט את המקש באמצעות פסיקת DOS מסי 7.
4. נזכור שאין לשמור את הנתון הנקלט בתוך אוגר AL, כי הוא עלול להשתנות בזמן ביצוע פסיקות DOS!
5. הפסיקה לקליטת מקש תגרום להצבת קוד ASCII שלו באוגר AL. אנו נצטרך לפענח על פי הקוד **מהי הספרה** שהוקשה.

   נסביר בהרחבה את קליטת המקשים והצגתם על המסך בקוד בינארי.

כאשר נקיש על הספרות 0 עד 9, נקבל את הקודים 30H עד 39H בהתאמה. ולכן, כדי לדעת את הספרה של אחד מהמקשים האלה, **נחסר** 30H מערך המקש שנקלט. אולם, כאשר נקיש על האותיות a עד f (אותיות קטנות, כאשר Caps Lock כבוי), נקבל את הערכים 61H עד 66H בהתאמה. כדי להפוך את הקודים המתקבלים לערכים שהם מייצגים, נשתמש בטבלה הבאה:

המקש הנלחץ	**קוד ASCII**	**הערך ההקסדצימלי**	**ההפרש בין קוד ASCII לבין הערך**
a	61H	A	57
b	62H	B	57
c	63H	C	57
d	64H	D	57
e	65H	E	57
f	66H	F	57

ניתן לראות על פי הטבלה, שכדי לקבל את הערך של המקש הנקלט (כאשר המקש הוא אות), יש להפחית 57H מקוד ASCII שהתקבל.

נסכם את הפעולות שיש לבצע. לאחר הפסיקה לקליטת מקש, יש לבדוק לאיזו קבוצה הוא שייך ולפעול בדרך זו:

- אם הקוד אינו גדול מ-39H, זוהי כנראה אחת מהספרות 0-9, ולכן יופחת 30H מקוד ASCII שהתקבל.
- אם הקוד גדול מ-39H, יש להניח שזוהי אחת מהספרות ההקסדצימליות a עד f, ולכן יופחת 57H מקוד ASCII שיתקבל.
- אם לאחר ההפחתה הערך גדול מ-F, המקש שהוקש אינו אחד מהספרות 0 עד F, והתוכנית תחזור לקליטת מקש אחר.

6. הצגת כל סיבית על המסך תתבצע באמצעות פסיקת DOS מסי 2.
7. בדיקת הסיביות תיעשה על ידי הפקודה TEST, שתפעל בכל פעם על סיבית אחרת; אם התוצאה היא 0, לפנינו סיבית "0". אחרת, לפנינו סיבית "1".

התוכנית (024.asm):

```
MA SEGMENT STACK
   DB 100H DUP (0)
MA ENDS
CODE SEGMENT
    ASSUME
CS:CODE,DS:CODE,SS:MA
MAIN:   MOV AX,CODE
        MOV DS,AX
FIRST:  MOV DX,OFFSET MY_MES
        MOV AH,9
        INT 21H              הצגת ההודעה
        MOV AH,7
        INT 21H              קליטת מקש
        MOV BL,AL            שמירת הקוד ב-BL
        CMP BL,39H
        JG OTT               אם הקוד גדול מ-39H, כנראה שזוהי אות.
        SUB BL,30H           אם הקוד אינו גדול, זוהי ספרה,
        JMP CONT             ולכן מופחת 30H
OTT:    SUB BL,57H           במקרה של אות מופחת 57H
CONT:   CMP BL,0FH           אם לאחר ההפחתה BL מכיל
        JA FIRST             ערך גבוה מ-0FH, יש לחזור לקליטת מקש נוסף
                             ** חשוב וענה : מדוע השתמשנו בפקודה JA ולא JG?
        MOV CX,4
        MOV DH,8             בתחילה נבדקת הסיבית השמאלית ביותר של הספרה
SIBIT:  TEST BL,DH           בדיקת מצב הסיביות
        JZ ZERO
        MOV DL,'1'
        JMP YALA
ZERO:   MOV DL,'0'
YALA:   MOV AH,2
        INT 21H
        SHR DH,1
        LOOP SIBIT           חזרה עבור 4 הסיביות
        MOV AX,4C00H
        INT 21H
        MY_MES DB 'Press one digit... ',10,13,'$'
CODE ENDS
END MAIN
```

קליטת ספרות הקסדצימליות וחישוב הסכום

התוכנית הבאה תקלוט ספרות הקסדצימליות (0 עד F) מהמקלדת, עד להקשה על Enter. היא תציג במסך את סכום הספרות שנקלטו.

נקודות התייחסות לפתרון הבעיה:

1. את המקשים נקלוט בדומה לתרגיל הקודם. דהיינו, לאחר קליטת מקש, נבדוק אם הוא נמצא בקבוצת הספרות או האותיות: עבור ספרות נפחית 30H מקוד ASCII של המקש שהתקבל, ועבור אותיות נפחית 57H.
2. פתרון התרגיל נעשה תוך הנחה, שהתוצאה לא תחרוג מגודל בית.
3. כדי להציג את התוצאה על המסך, יש לבצע את הפעולות הבאות:
   - לבודד את הספרה השמאלית על ידי הזזה ימינה ב-4 מקומות (SHR).
   - לבודד את הספרה הימנית באמצעות AND.
   - לבדוק את הספרה השמאלית: אם לפנינו ספרה (0-9), יש להוסיף 30H כדי להפוך את הספרה לקוד ASCII לתצוגה; אם לפנינו אות (A-F), יש להוסיף 57H, מאותה סיבה.
   - להציג את הספרה השמאלית באמצעות פסיקת DOS מס' 2.
   - לבדוק את הספרה הימנית ולקבוע אם לפנינו ספרה או אות, ואחר כך להציג את הספרה על המסך.

התוכנית (025.asm):

```
GOOD SEGMENT STACK
  DB 100H DUP (0)
GOOD ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE,SS:GOOD
MAIN:   MOV AX,CODE
        MOV DS,AX
        MOV BH,0                       BH ישמש כסכום, בו תישמר התוצאה
FIRST:  MOV DX,OFFSET MY_MES
        MOV AH,9
        INT 21H                        הודעה לפני קליטת כל מקש
        MOV AH,7
        INT 21H                        פסיקה לקליטת מקש (הקוד נכנס ל-AL)
        CMP AL,13
        JE GAMAR                       אם נלחץ מקש Enter, יש לסיים
        MOV BL,AL                      שמירת הקוד של המקש הנלחץ ב-BL
        CMP BL,39H
```

	JG OTT	אם הקוד גדול מ-39H, זוהי אות
	SUB BL,30H	
	JMP CONT	
OTT:	SUB BL,57H	עבור אות יש להפחית 57H
CONT:	CMP BL,0FH	
	JA FIRST	
	ADD BH,BL	
	JMP FIRST	חיבור הספרה החדשה ל-BH
GAMAR:	MOV BL,BH	יש להציג את התוצאה: תחילה מעתיקים את התוצאה ל-BL
	AND BL,0FH	בידוד הספרה הימנית של התוצאה
	MOV CL,4	
	SHR BH,CL	בידוד הספרה השמאלית של התוצאה
	MOV DL,BH	
	CMP DL,9	
	JG CHAR	אם הספרה השמאלית הינה אות - קפוץ
	ADD DL,30H	
	JMP PRINT	
CHAR:	ADD DL,57H	במקרה של אות מוסיפים 57H
PRINT:	MOV AH,2	
	INT 21H	פסיקה להצגת הספרה השמאלית
	MOV DL,BL	
	CMP DL,9	
	JG CHAR2	
	ADD DL,30H	
	JMP PRIN2	
CHAR2:	ADD DL,57H	
PRIN2:	MOV AH,2	
	INT 21H	פסיקה להצגת הספרה הימנית
	MOV AX,4C00H	
	INT 21H	סיום ויציאה ל-DOS

```
MY_MES DB 'Press one digit...',10,13,'$'
CODE ENDS
END MAIN
```

חישוב ממוצע והצגתו

מטלה: בתאי זיכרון שבכתובות 705H-714H נתונים ציוני 16 תלמידים (הציונים מיוצגים בבסיס 16). כתוב תוכנית שתחשב ותציג על המסך את הממוצע של ציונים אלה.

פתרון: את הממוצע נחשב על ידי מציאת סכום הציונים וחלוקתו ב-16. את סכום הציונים חובה להציב באוגר בגודל מילה! הסיבה לכך היא, שהסכום של 16 הציונים יחרוג מגודל של בית אחד.

בחן את התוכנית הבאה (026.asm), וראה כיצד עושים זאת.

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV BX,0
        MOV SI,705H
CALC:   MOV AL,[SI]
        MOV AH,0
        ADD BX,AX
        INC SI
        CMP SI,714H
        JNG CALC
        MOV CL,4
        SHR BX,CL
        MOV BH,BL
        SHR BH,CL
        AND BL,0FH
        MOV DL,BH
        ADD DL,30H
        MOV AH,2
        INT 21H
        MOV DL,BL
        CMP DL,9
        JG CHAR
        ADD DL,30H
        JMP PRINT
CHAR:   ADD DL,57H
PRINT:  MOV AH,2
        INT 21H
        MOV AX,4C00H
        INT 21H
CODE ENDS
END START
```

מיון של קבוצת מספרים

מטלה: בלוק תאי הזיכרון בתחום הכתובות 1000H-1024H מכיל מספרים שונים, אשר יש לסדר אותם בצורה הבאה: תחילה יופיעו המספרים שבהם הסיבית הגבוהה ביותר שערכה "1" הינה ב-D7; לאחריהם יופיעו המספרים בעלי הסיבית הגבוהה ביותר שערכה "1" ב-D6, וכך הלאה.

הפתרון: כדי לחפש את המספרים בעלי סיבית "1" ב-D7 נשתמש בפקודה TEST בין ערך התא לבין הערך 80H. אם התוצאה אינה 0, הדבר מעיד שערך סיבית D7 הוא "1". לאחר מכן, נחפש את התאים בעלי סיבית "1" ב-D6 על ידי הפעלת TEST עם הערך 40H. את 40H נקבל מהזזת הערך 80H מקום אחד ימינה. נמשיך כך עד סיום הבדיקה.

כדי לסדר את הנתונים נפעל על פי האלגוריתם הבא:

1. אוגר SI ישמש כמצביע על תא הזיכרון שאליו יוכנס הנתון הממוין, אשר ערכו בתחילה יהיה 1000H.
2. אוגר BX ישמש כמצביע על תאי הזיכרון שבהם יש לחפש. ערכו ההתחלתי של BX יהיה תמיד כערכו של SI, כדי שלא נחפש בנתונים שכבר ממוינים. אוגר BX תמיד יסרוק עד התא האחרון (1024H).
3. נחפש ערכים שערך סיבית D7 שלהם הוא "1". אם נמצא, נבצע:
   - נעתיק את ערך התא שמצאנו (שעליו מצביע BX) למקום המכיל את הנתונים הממוינים (שעליו מצביע SI).
   - נוסיף 1 ל-SI, כדי שיצביע על התא הבא כתא **פנוי** עבור הנתון הממוין הבא.
4. נמשיך בחיפוש ערך "1" בסיבית D6 באותו אופן. כשנסיים, נמשיך ונחפש ערך "1" בשאר הסיביות, עד וכולל D0.

גוף התוכנית (027.asm):

`        MOV AH,80H`	
`        MOV SI,1000H`	
`AGAIN:  MOV BX,SI`	BX מתחיל את החיפוש מ- SI
`BDOK:   MOV AL,[BX]`	
`        TEST AH,AL`	
`        JZ CONTIN`	אם התוצאה 0, הסיבית אינה 1
`        MOV DL,[SI]`	לכאן התוכנית מגיעה כאשר
`        MOV [SI],AL`	נמצאת סיבית "1" כמבוקש
`        MOV [BX],DL`	ההחלפה מבוצעת בתיווך DL
`        INC SI`	עדכון כתובת הנתון הממוין
`CONTIN: INC BX`	המשך הסריקה לכתובת הבאה
`        CMP BX,1024H`	
`        JNG BDOK`	אם לא הגענו לסוף, ממשיכים
`        SHR AH,1`	מחפשים סיבית נמוכה יותר
`        JNZ AGAIN`	אם עדיין לא בדקנו את כל הסיביות - יש להמשיך

קליטת משפט והצגת המילים

מטלה: כתוב תוכנית המבצעת את הפעולות הבאות:

- תקלוט מהמקלדת משפט בן מספר מילים, אשר יסתיים ב-Enter, ותציב אותו בזיכרון החל מהתא בכתובת 1020H.
- תציג כל מילה בשורה נפרדת. את המילה אפשר להציב במקום כלשהו בשורה.

שלבי הפתרון:

1. כל אות הנקלטת מהמקלדת תוצב בתא נפרד, ובכלל זה הקוד של מקש Enter. כאשר ייקלט מקש רווח (קוד 20H ASCII) יוחלף ערך התא בערך 0AH. קוד זה מציין מעבר לשורה חדשה עבור הדפסה על המסך.
2. באמצעות פסיקת DOS מס' 2 נציג על המסך תו אחרי תו (עד למקש Enter), כאשר התו 0AH יגרום למעבר לשורה חדשה, באותם מקומות שבהם היה רווח.

התוכנית (028.asm):

```
STORE SEGMENT STACK
   DB 100H DUP (0)
STORE ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE,SS:STORE
MAIN:   MOV AX,CODE
        MOV DS,AX
        MOV BX,1020H
KLOT:   MOV AH,1
        INT 21H
        CMP AL,20H
        JNE SHMOR
        MOV AL,10
SHMOR:  MOV [BX],AL
        INC BX
        CMP AL,13
        JNE KLOT
        MOV BX,101FH
        MOV BYTE PTR [BX],10       גורם למעבר לשורה חדשה לפני הצגת המילים
PRINT:  MOV DL,[BX]
        MOV AH,2
        INT 21H
        INC BX
        CMP DL,13
        JNE PRINT
        MOV AX,4C00H
        INT 21H
CODE ENDS
END MAIN
```

תרגילים בנושא "החברים של..."

"החברים של..." הינו שם שניתן לתרגילים שיובאו בהמשך, כדי שנוכל להתייחס אליהן ביתר פשטות.

נתחיל בתרגילים המתייחסים ל"**חברים של AL**".

אוגר AL מכיל 8 סיביות, שכל אחת מהן מציינת "חבר" אחר, על פי מיקומה באוגר:

בסיבית D0 נמצא החבר ששמו: **א**.

בסיבית D1 נמצא החבר **ב**.

בסיבית D2 נמצא החבר **ג**.

. . . . .

בסיבית D7 נמצא החבר **ח**.

נוכל לומר שהחבר נמצא ב-AL אם יש "1" במקום שלו, ונאמר שהחבר אינו נמצא באוגר כאשר נמצא "0" במקום שלו. נניח למשל, שאוגר AL מכיל את הערך הבינארי 1000 0110:

החבר ←	א	ב	ג	ד	ה	ו	ז	ח
אוגר AL ←	0	1	1	0	0	0	0	1

בדוגמה זו אנו רואים שיש "1" במקומות של **ב**, **ג**, **ח** ולכן נאמר ש-AL מכיל את החברים: **ב**, **ג** ו-**ח** בלבד.

דוגמה נוספת: נתון שאוגר AL מכיל את הערך הבינארי 0110 0000.

החבר ←	א	ב	ג	ד	ה	ו	ז	ח
אוגר AL ←	0	0	0	0	0	1	1	0

בדוגמה זו האוגר AL מכיל את החברים **ו** ו-**ז** בלבד.

התרגילים והפתרונות הבאים וחלק מהתרגילים לחזרה עוסקים בנושא זה.

התרגילים הבאים וחלק מתרגילי הבית מתייחסים ל"חברים של AL", ל"חברים של AH", ל"חברים של BL" ואחרים. גם באוגרים אלה יכולים להופיע החברים **א** עד **ח** (זכור שהערך "1" מציין שהחבר נמצא, בהתאמה). אלה הם אותם החברים שהצגנו בקשר לאוגר AL. חבר מסוים יכול להיות שייך לקבוצת "החברים של AL", וגם לקבוצת ה"חברים של BL", למשל.

הנה דוגמה לאפשרות שחבר מסוים ישתייך לשתי קבוצות שונות:

החבר ←	א	ב	ג	ד	ה	ו	ז	ח
החברים של AL	0	0	0	1	0	1	0	0
החברים של AH	0	1	0	0	0	1	0	0
החברים של BL	0	0	0	1	0	0	0	1

מדוגמה זו נוכל לראות שהחבר **ו** שייך לקבוצת "החברים של AL" וגם לקבוצת "החברים של AH". החבר **ד** שייך לקבוצת "החברים של AL" וגם לקבוצת "החברים של BL".

בדיקה אם "חבר" כלשהו נמצא באוגר

מטלה: נתון שאוגר BL מכיל את אחת האותיות: **א** עד **ח**, המציינת את שם החבר. על התוכנית לבדוק אם החבר נמצא ב-AL (כלומר, יש "1" במקום שלו). אם כן, יוצב 1 ב-BH; אם לא נמצא, יוצב 0 ב-BH.

שלבי הפתרון: תחילה עלינו למצוא דרך כדי להפוך את האות המוצבת ב-BL ומציינת את **שם** החבר, לספרה המציינת את **המקום** של החבר. כלומר, עבור החבר **א** נרצה לקבל את הערך 0 (זהו מיקומו), עבור **ב** נרצה לקבל 1, וכך הלאה.

אם נבדוק את טבלת ASCII המהווה את רשימת הקודים של התווים, נראה שהקוד של האות **א** הוא 128, הקוד של **ב** הוא 129, וכן הלאה. כלומר, אם נחסר 128 מהקוד הרשום באוגר BL, נקבל את המיקום של החבר באוגר!

לאחר שיש בידינו המיקום, ניתן להציב באוגר כלשהו את הערך "1" במקום זה; ואחר כך נוכל לבדוק באמצעות פעולת AND אם ב-AL יש גם כן "1" באותו מיקום. אם כן, המסקנה היא שהחבר נמצא ב-AL.

בפתרון המובא להלן, נבצע את האלגוריתם הבא (זכור תמיד, כי אין זה פתרון יחיד):

1. באוגר BH נציב את הערך 1.
2. נפחית 128 מאוגר BL, כדי להפוך את שם החבר למיקום שלו באוגר.
3. נזיז שמאלה את הסיביות באוגר BH עד למיקום של החבר שמחפשים. כך נקבל "1" במיקום של החבר המבוקש.
4. נבצע פעולת כפל לוגי (AND) בין AL לבין BH: אם החבר אינו נמצא ב-AL (יש "0" במיקום זה), התוצאה תהיה 0.
5. נזיז ימינה את הסיביות באוגר BH עד שנחזור למיקום המקורי. אם תוצאת הכפל הלוגי הקודמת היתה 0, התוצאה תישאר 0. אך אילו הסיבית במיקום החבר היתה "1", לאחר ההזזה ימינה נקבל את התוצאה 1.

נסביר את דרך הפתרון באמצעות דוגמה:

נתון: אוגר BL מכיל את האות (=החבר) **ג** (קוד ASCII 130).
אוגר AL מכיל את הערך הבינארי: 0110 1110 (והמסקנה היא שחבר **ג** נמצא).

1. נפחית 128 מאוגר BL, והוא יכיל את הערך 2.
2. אוגר BH הכיל 1, ולאחר הזזה שמאלה בשני מקומות (על פי ערך BL) נקבל ב-BH את הערך הבינארי: 0000 0100. שים לב שה-"1" נמצא במיקום של החבר המבוקש.
3. נראה את תוצאת הכפל הלוגי בין AL לבין BH:

אוגר AL מכיל	0110 1110
אוגר BH מכיל	0000 0100
התוצאה ב-BH ←	0000 0100

4. לאחר הזזה ימינה של אוגר BH בשני מקומות, נקבל:

BH לפני הזזה ⟵ 0000 0100

המצב אחרי הזזה ⟵ 0000 0001

אוגר BH מכיל 1, והמשמעות היא שהחבר **נמצא!**

התוכנית (029.asm):

```
CODE SEGMENT
   ASSUME CS:CODE
START:  MOV BH,1
        SUB BL,128
        MOV CL,BL
        SHL BH,CL
        AND BH,AL
        SHR BH,CL
CODE ENDS
END START
```

דוגמאות תרגול ל"חברים של..."

דוגמה 1

נכתוב פקודה, או פקודות, להוסיף את החבר **ז** אל אוגר AL, אם הוא אינו נמצא בו.

הפתרון: לא נבדוק אם החבר **ז** נמצא באוגר, אלא נוסיף אותו בכל מקרה באמצעות פעולת OR. אם הוא כבר נמצא, הפעולה לא תשנה דבר, כי פעולת OR של "1" עם "1" משאירה "1". אם הוא לא נמצא (במקום זה יש "0"), פעולת OR תגרום לכך שבמקום זה יהיה כעת "1", שפירושו במקרה זה - החבר נוסף.

הפקודה המבצעת זאת: OR AL,40H

דוגמה 2

נכתוב פקודה, או פקודות, להציב באוגר DL את החברים המשתייכים ל-AL ול-AH. פעולת "הצבה" בהקשר זה משמעותה לגרום שיהיה ערך "1" במיקום של החברים המתאימים. הפתרון:

```
MOV DL,AL
AND DL,AH
```

דוגמה 3

יש להוסיף ל"חברים של BL" את "החברים של AL" ואת "החברים של AL". הפתרון:

```
OR BL,AL
OR BL,AH
```

דוגמה 4

התוכנית הבאה (030.asm) מונה את מספר החברים שמשתייכים הן ל-AL והן ל-AH, ומציבה את המספר הזה באוגר DL.

```
CODE SEGMENT
   ASSUME CS:CODE
START:  MOV DH,AL
        AND DH,AH
        MOV DL,0
        MOV AL,80H
CHECK:  TEST DH,AL
        JZ ZUZU
        INC DL
ZUZU:   SHR AL,1
        JNZ CHECK
CODE ENDS
END START
```

תרגילים

התרגילים הבאים עוסקים בעיקר בנושאים האחרונים שנלמדו ומסודרים לפי נושאים.

תרגילים כלליים

1. כתוב תוכנית שתאפס את דגל הזכור.
2. כתוב תוכנית שתשנה את דגל הסימן למצב "1", או שתשאיר אותו במצב זה.
3. כתוב תוכנית שתעתיק לכתובת פיסית B8008H את הנתון מתא בכתובת פיסית 4H.
4. כתוב תוכנית שתהפוך את מצב סיבית D6 של התא בכתובת 440H.
5. כתוב תוכנית שתמצא את כתובת התא המכיל את הספרה השמאלית הגדולה ביותר. התוכנית תבדוק את תחום הכתובות 730H-788H. הכתובת המבוקשת תוצב בתאים 800H ו-801H.
6. כתוב תוכנית שתבדוק כמה סיביות "1" וכמה סיביות "0" יש בנתון שבתא 703H. המטרה היא שיהיה מספר שווה של סיביות "1" וסיביות "0" בתא זה, ולכן התוכנית תאפס, או תעלה סיביות למצב "1", כדי להשוות את מספרן.
7. כתוב תוכנית שתחליף בין ערך של התא שכתובתו הפיסית 4F0H, לבין הערך של התא בכתובת פיסית 4F2H.
8. כתוב תוכנית שתספור את תאי הזיכרון שבהם הספרה הימנית זהה לספרה השמאלית. הבדיקה תיעשה בתאים 2170H-2280H. התוצאה, בגודל מילה, תירשם בתאים 1004H ו-1005H.

9. מה יכיל כל אחד מהאוגרים הרלוונטיים לאחר ביצוע כל אחת מהתוכניות האלו?

תוכנית א'

```
MOV DH,63H
MOV CL,2
MOV DL,DH
ADD DL,DH
OR AL,6
SHL DL,CL
SHR DH,1
```

תוכנית ב'

```
MOV AX,4678H
AND AL,34H
OR AL,35H
XOR AH,9
```

תוכנית ג'

```
MOV AL,52H
XOR AL,25H
SHL AL,1
AND AL,7
XOR AL,99H
```

10. כתוב תוכנית שתבדוק אם סכום ספרות העשרות של ערכי התאים בתחום הכתובות 500H-520H, שווה לסכום ספרות האחדות של תאים אלה. אם כן, תוצג הודעה על המסך "Equal".

11. כתוב תוכנית שתקלוט מספר דו-ספרתי מהמקלדת ותציג על המסך הודעה "Negative" או "Positive", בהתאם לעניין.

תרגילים בנושא "החברים של AL"

12. כתוב תוכנית שמקבלת באוגר BL שם של חבר (האותיות **א** עד **ח**) וגורעת אותו מה"חברים של AL". הכוונה לכך שהיא מוציאה אותו על ידי הצבת "0" במיקום שלו באוגר.

13. כתוב תוכנית שמציבה באוגר DL את המספר הגדול ביותר של רצף חברים ב-AL, החל מימין. כלומר, זהו המספר הגדול ביותר של רצף המופעים של הסיבית "1".

14. כתוב תוכנית שמציגה על המסך את שמו (**א** עד **ח**) של החבר השמאלי ביותר בקבוצה.

15. כתוב תוכנית שגורעת (מוציאה) את שני החברים הקיצוניים, מימין ומשמאל, בקבוצת "החברים של AL".

16. כתוב תוכנית שמציבה באוגר DL את מספר החברים שאינם משתייכים ל-AL וגם לא ל-AH.

17. כתוב תוכנית שמציבה ב"חברים של BL" רק את החברים הנמצאים ב-AL, אך אינם שייכים ל-BL.

18. כתוב תוכנית שמציבה באוגר DL את מספר החברים הנמצאים רק באחת, אך לא יותר, מהקבוצות AL, AH, BH.

19. כתוב תוכנית שמוסיפה ל"חברים של BL" את כל החברים שאינם מופיעים ב-AL וגם לא מופיעים ב-AH.

20. כתוב תוכנית שמציבה באוגר DL את מספר החברים העונים לדרישה: נמצא ב-AL, לא נמצא ב-BL, נמצא ב-AH.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

11

משתנים ומערכים

מה הם המשתנים?

לשמירת נתונים השתמשנו עד כה באוגרים, או בתאי זיכרון. שפת אסמבלי, בדומה לשפות אחרות, מאפשרת שימוש ב**משתנים** (variables), שבהם ניתן להציב נתונים ולבצע פעולות חשבוניות ולוגיות.

כדי להשתמש במשתנה, עלינו להגדירו. ההגדרה כוללת את **שם המשתנה**, את **הגודל** ואת **הערך ההתחלתי** שלו. לדוגמה: RESULT DB 0.

שם המשתנה בדוגמה זו הוא RESULT, גודלו בית אחד, על פי ההוראה Define Byte) DB - הגדר גודל של בית) והערך ההתחלתי שלו הינו 0.

שם המשתנה צריך להכיל תווים צמודים, אשר כוללים את האותיות a-z (באותיות קטנות, או גדולות, כרצונך). אין להשתמש בתו רווח, אך אפשר להשתמש בקו-תחתון (_), כמו למשל A_B. אין להשתמש בספרה בתחילת שם משתנה, אך מותר להשתמש בה בגוף השם, למשל MONE1, או Test3. אין לתת שם משתנה זהה ל**מילים שמורות** (reserved words) עבור שמות של פקודות, או של הוראות.

דוגמאות לשמות חוקיים ולא-חוקיים:

חוקי	-	NewLine
חוקי	-	THIS_IS_MY_TEST
חוקי	-	_SHALOM2
לא חוקי: השם מכיל רווח	-	MISPAAR 1
חוקי	-	beseder
חוקי	-	MONE1
לא חוקי: שם מתחיל בספרה	-	7total
לא חוקי: שם שמור (פקודה)	-	CMP
חוקי	-	A
לא חוקי: שם שמור (הנחיה)	-	ASSIGN

את גודל המשתנה, כמה בתים יתפוס בזיכרון, קובעים בהתאם לשימוש בו. בדרך כלל, גודל המשתנה ייקבע כאחת מהאפשרויות הבאות:

- **DB** Define Byte בית (8 סיביות)
- **DW** Define Word מילה (16 סיביות)
- **DD** Define Double מילה כפולה (32 סיביות)

למשתנה ניתן לקבוע ערך התחלתי, למשל: SIMAN DB 37H.

גם ניתן לקבוע משתנה שאינו-מאותחל (שלא נקבע לו ערך התחלתי): SIMAN DB ?.

סימן השאלה (?) מציין שלא נקבע ערך התחלתי למשתנה, והערך שלו אינו ידוע!

דוגמאות להגדרת משתנים:

POPYE DW 1427H

המשתנה נקרא POPYE, גודלו מילה אחת וערכו ההתחלתי 1427H.

OLIVE DB ?

המשתנה OLIVE הינו בגודל מילה ואינו מאותחל.

SHALOM_LEKULAM DB 181H

זו שגיאה בהגדרת המשתנה: המשתנה מוגדר כ-DB (בגודל בית) ולכן אין להציב בו ערך התחלתי החורג מגודלו!

KEY_PRESSED DW 8

המשתנה מוגדר בשם KEY_PRESSED, גודלו מילה והוא מכיל ערך התחלתי 8.

הגדרת המשתנים בתוכנית

ניתן להגדיר משתנים בתוכנית בשני אופנים:

1. בתוך **מקטע התוכנית** - **CS** (Code Segment).
2. **במקטע נתונים** נפרד - **DS** (Data Segment).

דוגמאות להגדרת משתנים בתוך מקטע התוכנית

לרוב אנו מגדירים את המשתנים בתחילת המקטע לפני קוד התוכנית, או בסוף המקטע לאחר התוכנית. כלל חשוב שמומלץ להקפיד עליו:

שמור על **הפרדה** בין פקודות התוכנית לבין הגדרת המשתנים!

דוגמה 1

התוכנית מציבה באוגר DX את ערך המשתנה YOFI, שערכו ההתחלתי 1234H (001.asm).

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  MOV AX,CODE
        MOV DS,AX
        MOV DX,YOFI          ערך המשתנה מועתק ל-DX
        MOV AX,4C00H
        INT 21H              סיום התוכנית ויציאה ל-DOS
YOFI    DW 1234H
CODE ENDS
END START
```

המשתנה הוגדר **לאחר** הפקודות לסיום התוכנית ולכן הוא אינו חלק מהפקודות. לולא נרשמו הפקודות האלו,

```
MOV AX,4C00H
INT 21H
```

היה המיקרומעבד ממשיך לשורה הבאה ומנסה לפענח כפקודה את המשפט הבא:

```
YOFI  DW 1234H
```

דבר זה עלול לגרום להודעת שגיאה, או לתוצאות לא רצויות.

אם מריצים את התוכנית מתוך DEBUG ולא ישירות ממערכת ההפעלה, ניתן להחליף את צמד הפקודות שלעיל באחת מהפקודות הבאות:

- INT 3 - פסיקה שגורמת לעצירת התוכנית.
- HLT - פקודה לעצירת התוכנית.

דוגמה 2

התוכנית מציבה את ערך המשתנה STAM, שערכו ההתחלתי 88H בתא זיכרון שבכתובת 712H (002.asm).

```
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE
START:  JMP BEGIN
STAM    DB 88H
BEGIN:  MOV AX,CODE
        MOV DS,AX
        MOV AL,STAM          נעתיק ל-AL את ערך המשתנה
        MOV BX,712H
        MOV [BX],AL
CODE ENDS
END START
```

הפקודה הראשונה גורמת לדילוג מעל שורת הגדרת המשתנה. כך מתבצעת הפרדה בין הפקודות לבין הגדרת המשתנה. בדרך כלל, שיטה זו אינה מקובלת.

דוגמאות להגדרת משתנים במקטע נתונים

לפי שיטה זו, הגדרת משתנים במקטע נתונים (שהינה המקובלת והנוחה יותר), מגדירים מקטע חדש המכיל את המשתנים בלבד. בדוגמה הבאה, שם המקטע החדש הוא DATA ובו מוגדרים שני משתנים שאינם מאותחלים : PAPA ו-MAMA.

```
DATA SEGMENT
  MAMA DB ?
  PAPA DB ?
DATA ENDS
```

מתחת למקטע זה נכתוב את מקטע התוכנית CS (לרוב כינינו אותו בשם CODE), הכולל את השינויים הבאים :

1. בפקודה ASSUME נכתוב את שם מקטע המשתנים.

   במקום לרשום : ASSUME CS:CODE,DS:CODE

   יש לרשום : ASSUME CS:CODE,DS:DATA

   בהוראה ASSUME אנו מודיעים לאסמבלר MASM את הכתובת ההתחלתית של התוכנית ושל הנתונים. כאשר לא השתמשנו במקטע מיוחד לנתונים, כתבנו בשורה ASSUME את שם המקטע CODE עבור מקטע התוכנית (קוד) CS ועבור מקטע הנתונים DS. כלומר, כתובת הבסיס של התוכנית זהה לכתובת הבסיס של הנתונים. כאמור, המושג **נתונים** כולל ערכים של תאי זיכרון או של משתנים.

   במקרה שיש מקטע נפרד לנתונים ושמו למשל DATA, יש להודיע ש-DS יכיל את כתובת ההתחלה של מקטע הנתונים DATA. הדבר נעשה על ידי כתיבת ההוראה DS:DATA במקום DS:CODE, כפי שעשינו עד כה.

2. בתחילת התוכנית, כתבנו בדרך כלל את הפקודות האלו :

```
MOV AX,CODE
MOV DS,AX
```

   נכתוב במקומן, בהתאמה :

```
MOV AX,DATA
MOV DS,AX
```

   שתי הפקודות האלו מציבות באוגר DS את כתובת ההתחלה של הנתונים. יש לזכור, שההוראה ASSUME רק "מודיעה" לאסמבלר איך עומדים לנהוג, אך אינה מציבה באוגר את הכתובת של CODE, או את הכתובת של DATA. לכן, כאשר הנתונים נמצאים במקטע DATA ולא במקטע CODE, יש לרשום DATA במקום CODE.

נבחן מספר דוגמאות של ההגדרה והשימוש במשתנים.

דוגמה 1

התוכנית מעתיקה את ערך המשתנה NO1 אל האוגר DL, ואת ערך המשתנה NO2 - אל האוגר DH. משתנה NO1 מכיל את הערך ההתחלתי 4 ו-NO2 את הערך 5 (003.asm).

```
DATA SEGMENT
        NO1 DB 4
        NO2 DB 5
DATA ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV DL,NO1          מעתיק את ערך NO1 לאוגר DL
        MOV DH,NO2          מעתיק את ערך NO2 לאוגר DH
CODE ENDS
END START
```

במקטע הנתונים הוגדרו שני המשתנים המבוקשים. גודלם נבחר כבית אחד (DB), כי חייבת להיות התאמה לגודל חצאי-האוגר DX. השם שנבחר למקטע הנתונים הוא DATA, ולכן הותאמו ההגדרה וההצבה באוגר DS לפי שם זה.

דוגמה 2

התוכנית מעתיקה לתוך משתנה SIMAN שאינו מאותחל, את ערך תא הזיכרון בכתובת 1000H (004.asm).

```
NATUN SEGMENT
  SIMAN DB ?
NATUN ENDS
PROGRAM SEGMENT
  ASSUME CS:PROGRAM,DS:NATUN
START:  MOV AX,NATUN
        MOV DS,AX
        MOV BX,1000H
        MOV AL,[BX]         מעתיק ל-AL את ערך התא
        MOV SIMAN,AL        מעתיק למשתנה את ערך AL
PROGRAM ENDS
END START
```

השם שנבחר למקטע הנתונים הוא NATUN, והשם שנבחר למקטע הקוד (התוכנית) הוא PROGRAM. המשתנה SIMAN הוגדר כלא-מאותחל (על ידי סימן השאלה). העתקת ערך התא נעשתה בתיווך של אוגר בגודל תואם. על כך עוד נלמד בהמשך.

כללי שימוש במשתנים

הדוגמאות וההסברים בהמשך מבוססים על גרסת האסמבלר MASM שבה השתמשנו. ייתכנו שינויים קלים בגרסאות שונות של תוכנת MASM, או תוכנות דומות שתשתמש בהם למטרה דומה. עם זאת, חשוב לציין שהכללים שנציג להלן טובים לכל הגרסאות שבדקנו.

1. ניתן להציב נתון (ערך) בתוך משתנה, ללא תיווך אוגר. למשל,

```
MOV NO1,83H
MOV POPYE,102H
```

2. אין להעתיק נתון באופן ישיר ממשתנה אחד לאחר. לדוגמה:

הפקודה אינה חוקית ⟵ MOV NO1,NO2

כדי לבצע פעולה זו, חובה להשתמש באוגר מתווך:

```
MOV DL,NO1
MOV NO2,DL
```

3. יש להקפיד על שימוש נכון בגודל המשתנים, כפי שהוגדרו. לדוגמה, אם המשתנה MORE הוגדר כבית (DB), הפקודה "MOV MORE,168H" אינה חוקית.

דוגמה נוספת: אם המשתנה YES מוגדר כבית, הפקודה "MOV AX,YES" תגרום להודעת שגיאה. זכור שחייבת להיות התאמה בגודל האופרנדים.

4. אין לבצע העתקת נתונים באופן ישיר בין משתנה לבין תא זיכרון, ולהיפך. למשל, פקודה זו אינה חוקית: "MOV [BX],GOOD". כדי לבצע את הפעולה יש להשתמש באוגר מתווך בגודל תואם. דוגמה לפתרון נכון:

```
MOV CL,GOOD
MOV [BX],CL
```

5. ניתן לרשום את שם המשתנה בין סוגריים מרובעים.

לדוגמה, במקום: MOV MONE,0 אפשר לכתוב: MOV [MONE],0 (המשמעות זהה)

ובמקום: MOV CX,KAMUT אפשר לכתוב: MOV CX,[KAMUT] (המשמעות זהה)

עד כה, השתמשנו בסימון של סוגריים מרובעים, כדי לציין תא זיכרון. האם גם המשתנים הם תאי זיכרון? ובכן, התשובה היא **כן**! את ההסבר נספק בהמשך.

6. אפשר לבצע פעולות חשבוניות ולוגיות על המשתנים. הנה מספר דוגמאות:

- SHR RESULT,1
- CMP MISPAR,32
- TEST POPYE,80H
- SUB MISHKAL,20

המשמעות של הגדרת משתנים בתוכנית

נכתוב הגדרה של משתנה כמו זו, למשל: NATUN DB 8.

תוכנת אסמבלר מקצה תא זיכרון אחד בגודל של בית אחד ומציבה בו את הערך 8. בכל פעם שאנו משתמשים בתוכנית במשתנה NATUN, אנו פונים לתא הזיכרון שהוקצה לו. למשל, אם נרשום את הפקודה "ADD NATUN,2" אנו גורמים למעשה להוספת הערך 2 אל הערך שנמצא **בתא הזיכרון** המייצג את המשתנה.

ומה עושה ההוראה הזו: SFARIM DW 1082H ?

היא גורמת להקצאה של שני תאי זיכרון עוקבים, ומציבה בהם את הערך 1082H. זכור שהערך 10H מוצב בתא הגבוה יותר.

כתיבת ההוראה הזו: BIG DD 12345678H

תגרום להקצאת 4 תאי זיכרון עוקבים ותציב בהם ערכים: הערך 78H יוצב בתא הראשון (שמספרו הנמוך ביותר), 56H יוצב בתא הבא וכן הלאה.

נכתוב את ההוראה הזו: HOD DB 'THIS IS AN EXAMPLE'

ההוראה מציינת הגדרת משתנה המכיל מחרוזת תווים. מדוע, אם כן, כתבנו DB? התשובה היא שכל אחד מהתווים, כולל תו רווח, מוצב בתא אחד שגודלו בית, כמובן. בתא הראשון מוצבת האות T על פי ערך קוד ASCII שלה, בתא הבא מוצבת האות H וכדומה. בדוגמה זו, מוקצים 18 תאים עוקבים המכילים את כל התווים שבמחרוזת, לרבות הרווחים.

נניח שהוגדר מקטע הנתונים הבא:

```
DSEG SEGMENT
  NO1 DB 3
  NO2 DB 2
  NO3 DW ?
DSEG ENDS
```

למשתנה NO1 יוקצה תא אחד ויוצב בו הערך 3, ל-NO2 יוקצה תא אחד ויוצב בו 2 ול-NO3 יוקצו שני תאים עוקבים ללא ערך התחלתי, ועל כן יישאר בהם ערכם המקורי. יתר על כן, התא של NO2 יהיה התא בכתובת הבאה, אחרי התא של NO1 ושני התאים של NO3 יבואו מייד לאחר התא של NO2.

תיאור סכימטי של מיקום המשתנים בזיכרון ניתן בתרשים הבא, שבו החץ מסמן את כיוון הכתובות הגבוהות בזיכרון וכל ריבוע מתאר תא זיכרון אחד:

המשתנה הראשון שמוגדר במקטע הנתונים יהיה תמיד בכתובת 0, מלבד מקרים מסוימים שידונו בהמשך.

כתובת 0 היא, כמובן, **היסט** בלבד; **הבסיס** הוא ערך אוגר DS.

בדוגמה הקודמת ניתן לומר ש-NO1 נמצא בכתובת 0, NO2 - בכתובת 1 ו-NO3 בכתובות 2 ו- 3. משום כך, נוכל לראות ב-DEBUG, שהפקודה: MOV NO3,1600H

הוחלפה בפקודה: MOV WORD PTR [0002],1600

הסיבה לכך היא שהמשתנה NO3 נמצא החל בכתובת 2. הערך 1600H יוצב בתא שבכתובת 2 ובתא שבכתובת 3.

דוגמה: הפקודה שבתוכנית ADD NO1,7

כתובת המשתנה NO1 הינה 0, לכן בתוכנית הניפוי תופיע כך: ADD BYTE PTR [0000],7

תרגילים ופתרונות

דוגמה 1

התוכנית הבאה כוללת מקטע נתונים ובו משתנה בשם POS, שערכו ההתחלתי 0. התוכנית תציב 1 במשתנה זה (005.asm).

```
DATA SEGMENT
   POS DB 0
DATA ENDS
CSEG SEGMENT
   ASSUME CS:CSEG,DS:DATA
BEGIN:  MOV AX,DATA
        MOV DS,AX
        MOV POS,1
CSEG ENDS
END BEGIN
```

שלבי בדיקת התוכנית באמצעות DEBUG:

1. נכתוב U0 להצגת התוכנית. שים לב, שמקטע הנתונים אינו מופיע כלל. הדבר נובע מכך שהפקודה U0 הופכת את התוכנית הכתובה בשפת מכונה (בעלת הסיומת EXE) לפקודות אסמבלי, ולכן מופיעות פקודות בלבד.

2. בדיקת כתובת סיום התוכנית (במקרה זה קיבלנו 000A). נבדוק גם את הערך שנקבע לאוגר DS. זהו המספר שרשום בשורה הראשונה במקום המילה DATA. ערך זה הוא כתובת הבסיס שנקבעה לצורך שמירת המשתנים והנתונים המוגדרים במקטע הנתונים.

   כתובת הבסיס נקבעת על ידי מערכת ההפעלה על פי המקום הפנוי בזיכרון, ולכן היא עשויה להשתנות בהפעלות שונות ובמחשבים שונים. לצורך הסבר הדוגמה, נניח שכתובת הבסיס שנקבעה היא 1200H.

3. הצגת ערך המשתנה POS לפני הרצת התוכנית: D 1200:0 0

המשתנה POS הינו תא זיכרון ולכן משתמשים בהוראה D. כתובת הבסיס היא הערך שנקבע לאוגר DS, שבדוגמה זו הנחנו שערכו 1200H. המשתנה POS הינו הראשון שהוגדר במקטע הנתונים, ולכן כתובתו 0.

במילים אחרות: כדי להציג את ערך המשתנה POS, יש להציג את תא הזיכרון שכתובת הבסיס שלו שווה לכתובת תחילת מקטע הנתונים, וההיסט הוא 0. כתוצאה נקבל את הערך 0, שהוא הערך שנקבע כערך התחלתי של המשתנה.

4. הרצת התוכנית: G=0 A

5. בדיקת ערך המשתנה POS, אחרי הרצת התוכנית. נשתמש בפקודה: D 0 0

לאחר הרצת התוכנית, הערך החדש של כתובת תחילת הנתונים יוצב באוגר DS, ולכן אין צורך להוסיף את הבסיס. כלומר, מיותר לרשום את ההוראה "D 1200:0 0", כפי שנהגנו בתוכניות קודמות.

ערכו של המשתנה בסיום הפעולה הוא 1.

דוגמה 2

נכתוב תוכנית הכוללת מקטע נתונים המכיל שלושה משתנים, שכל אחד מהם בגודל בית אחד:

משתנה A, שערכו ההתחלתי 12H,

משתנה B, המכיל את הערך 13H,

משתנה C, המכיל את הערך 0.

התוכנית תציב את סכום ערכי המשתנים A ו-B לתוך משתנה C (006.asm).

(הנחה: התוצאה לא תגלוש מעבר לגודל של בית)

```
DATA SEGMENT
   A DB 12H
   B DB 13H
   C DB 0
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV AH,A
        ADD AH,B
        MOV C,AH
CODE ENDS
END START
```

שלבי בדיקת התוכנית ב-DEBUG:

1. הצגת התוכנית, בדיקת כתובת סיום התוכנית (בדוגמה זו קיבלנו 11H) וכתובת תחילת מקטע הנתונים (הכתובת שתוצב באוגר DS). לצורך הדוגמה, נניח שהכתובת לתחילת מקטע הנתונים היא 1700H.

2. בדיקת ערך המשתנים: D 1700:0 2

כתובת המשתנה הראשון היא 0, כתובת השני 1 וכתובת השלישי היא 2. כתובות אלו הן **היסט** מתחילת מקטע הנתונים, ולכן דרושה הגדרה של כתובת הבסיס (כתובת ההתחלה) של מקטע הנתונים.

על המסך יוצגו ערכי המשתנים: 0 13 12

הערך השמאלי ביותר הוא תא הזיכרון בכתובת 0, הוא המשתנה הראשון (A). לאחריו, בתא 1, מופיע המשתנה B והערך האחרון (0) שייך למשתנה C.

3. הרצת התוכנית: G=0 11

4. בדיקה חוזרת של ערכי המשתנים: D 0 2.

נקבל על המסך את הערכים: 25 13 12.

המשתנה C (השמאלי), אכן קיבל את סכום שני המשתנים A ו-B, כפי שציפינו.

דוגמה 3

התוכנית הבאה מציגה על המסך את המחרוזת הרשומה במשתנה COME:

"THIS IS MY COMPUTER"

תרגיל בנוסח זה פתרנו כבר. הפעם המשתנה יוגדר בתוך מקטע הנתונים (007.asm).

```
DATA SEGMENT
   COME DB 'THIS IS MY COMPUTER $'
DATA ENDS
M  SEGMENT STACK
    DB 100H DUP (0)
M  ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA,SS:M
MAIN:    MOV AX,DATA
         MOV DS,AX
         MOV DX,OFFSET COME
         MOV AH,9
         INT 21H
         MOV AX,4C00H
         INT 21H
CODE ENDS
END MAIN
```

בתוכנית הוגדר מקטע מחסנית STACK. כאשר תוכנית כוללת פסיקות יש להגדיר בה מחסנית (מושג המחסנית יוסבר בהמשך).

המחרוזת שהוגדרה עבור המשתנה COME גורמת להצבת קוד ASCII של התו T בתא שבכתובת 0, לאחריו יוצב הקוד של התו H וכדומה (תוכל לבדוק בעזרת DEBUG שזה אכן קורה).

הפקודה MOV DX,OFFSET COME מציבה ב-DX את ההיסט (offset) של המשתנה COME מתחילת המקטע שבו הוא נמצא. בתוכנית זו ההיסט הוא 0, מכיון שזהו המשתנה הראשון המוגדר. תוכל לבדוק בעזרת DEBUG שהערך 0 מוצב ב-DX.

דוגמה 4

במקטע הנתונים מוגדרים שלושה משתנים: NAME1 מכיל את המחרוזת "COM", NAME2 מכיל את המחרוזת "PUTER" ו- THE_NAME מכיל 8 תאים שאינם מאותחלים ואת התו "$" בתא התשיעי.

צריך לכתוב תוכנית שתעתיק למשתנה THE_NAME את המחרוזת של משתנה NAME1 ומייד לאחריה את המחרוזת של NAME2. כך תיווצר המחרוזת "COMPUTER". לבסוף, תוצג על המסך המחרוזת הרשומה במשתנה THE_NAME.

התוכנית לפתרון הבעיה נתונה להלן (008.asm). בחן את הפתרון על פי הכללים שלמדת עד כה. הסבר כיצד בעזרת לולאה אחת ניתן להעתיק את שתי המחרוזות.

```
DATA SEGMENT
  NAME1 DB 'COM'
  NAME2 DB 'PUTER'
  THE_NAME DB 8 DUP (?)
              DB '$'
DATA ENDS
M SEGMENT STACK
   DB 100H DUP (0)
M ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA,SS:M
MAIN:   MOV AX,DATA
        MOV DS,AX
        MOV SI,OFFSET NAME1
        MOV DI,OFFSET THE_NAME
        MOV CX,8
PASS:   MOV AL,[SI]
        MOV [DI],AL
        INC SI
        INC DI
        LOOP PASS
        MOV DX,OFFSET THE_NAME
        MOV AH,9
        INT 21H
        MOV AX,4C00H
        INT 21H
CODE ENDS
END MAIN
```

דוגמה 5

במקטע נתונים מוגדרים שלושה משתנים בגודל בית: משתנה ONE מכיל את הערך 38H; המשתנה TWO מכיל את הערך 46H; הערך ONE_TWO אינו מאותחל.

התוכנית תציב במשתנה ONE_TWO מספר שספרתו השמאלית זהה לספרה הימנית של ONE, ושספרתו הימנית זהה לספרתו שהמאלית של TWO. אין צורך לשמור את התוכן ההתחלתי של המשתנים ONE ו-TWO, אפשר לשנות אותם במהלך העבודה (009.asm).

```
DATA SEGMENT
  ONE DB 38H
  TWO DB 46H
  ONE_TWO DB ?
DATA ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:DATA
MAIN:   MOV AX,DATA
        MOV DS,AX
        MOV CL,4
        SHL ONE,CL
        SHR TWO,CL
        MOV DL,ONE
        OR DL,TWO
        MOV ONE_TWO,DL
CODE ENDS
END MAIN
```

דוגמה 6

במקטע הנתונים מוגדר משתנה GADOL המכיל את הערך 1993H. המשתנים YAMIN ו-SMOL מוגדרים בגודל של בית וללא ערך התחלתי. התוכנית תציב ב-SMOL את שתי הספרות השמאליות של GADOL ותציב ב-YAMIN את שתי הספרות הימניות של GADOL.

פתרון א' (010.asm):

```
DATA SEGMENT
  GADOL DW 1993H
  SMOL DB ?
  YAMIN DB ?
DATA ENDS
CSEG SEGMENT
  ASSUME CS:CSEG,DS:DSEG
START:  MOV AX,DSEG
        MOV DS,AX
        MOV BX,OFFSET GADOL
        MOV AL,[BX]
        MOV YAMIN,AL
        MOV AL,[BX+1]
        MOV SMOL,AL
CSEG ENDS
END START
```

פתרון ב' (גוף התוכנית בלבד):

```
MOV AL,BYTE PTR GADOL
MOV YAMIN,AL
MOV AL,BYTE PTR GADOL+1
MOV SMOL,AL
```

ניתן לפנות לחלק מהמשתנה (בגודל בית) באמצעות BYTE PTR:

- כאשר נרשמת הפקודה "MOV AL,BYTE PTR GADOL" מקבל AL את ערך הבית הראשון של המשתנה. למעשה, זהו תא הזיכרון הראשון המייצג את המשתנה.
- הפקודה "MOV AL,BYTE PTR GADOL+1" מציבה ב-AL את הבית השני של המשתנה. התא השני הוא תא הזיכרון בהיסט 1 מתחילת המשתנה.

מערכים

הגדרת המערך ואיבריו

עד כה הגדרנו נתון אחד עבור כל משתנה. לדוגמה: MISPAR DB 8

ניתן להגדיר מספר נתונים עבור משתנה אחד: MESS DB 8,9,4

הגדרנו משתנה בשם MESS המכיל שלושה נתונים, שכל אחד מופרד מחברו בפסיק. כך אנו יוצרים **מערך** (Array), שמגדיר קבוצה הומוגנית של משתנים בודדים המתייחסים לעניין אחד. כל אחד מהמשתנים הבודדים של המערך קרוי **איבר**. מושג המערך בשפת אסמבלי, זהה למושג המערך בשפת פסקל ובשפות אחרות.

כיצד ניתן לפנות לכל אחד מהאיברים של MESS?

יש לציין במפורש **לאיזה** מהאיברים מתוך קבוצת האיברים של המערך MESS אנו רוצים לפנות. כאן נאמר, שהאיבר הראשון הוא תמיד איבר 0, השני הוא איבר 1, אחריו איבר 2 וכן הלאה (בדומה לשפת C).

כך, כדי לפנות לאיבר הראשון של מערך MESS נכתוב MESS[0]. לאיבר השני נפנה על ידי MESS[1] וכן הלאה. כלומר, כדי לפנות לאיבר מסוים של המערך, אנו כותבים את שם המערך ומימין לו, בסוגריים מרובעים, את המספר הסידורי של האיבר במערך. את ספירת האיברים במערך אנו מתחילים ב-0.

אם ברצוננו להציב באוגר AL את האיבר הראשון של MESS וב-AH את האיבר השני שלו, נרשום את הפקודות האלו:

```
MOV AL,MESS[0]
MOV AH,MESS[1]
```

נכתוב מספר תוכניות דוגמה, המטפלות במערכים.

תוכניות דוגמה למערכים

דוגמה 1

התוכנית כוללת מערך בשם A בן שני מספרים בגודל בית כל אחד. היא מציבה באוגר DL את סכום שני המספרים של המערך A.

להלן התוכנית (011.asm) :

```
VAR SEGMENT
  A DB 4,5
VAR ENDS
PROGRAM SEGMENT
  ASSUME CS:PROGRAM,DS:VAR
START:  MOV AX,VAR
        MOV DS,AX
        MOV DL,A[0]
        ADD DL,A[1]
PROGRAM ENDS
END START
```

התוכנית מציבה ב-DL את האיבר הראשון של המערך A ומוסיפה את הנתון השני. אנו ממליצים שתריץ את התוכנית ותבדוק אותה באמצעות DEBUG.

דוגמה 2

תוכנית זו כוללת שני מערכים : מערך A מכיל שני איברים שמכילים מספרים, ומערך B הכולל שני איברים שאינם מאותחלים. התוכנית תעתיק את האיבר הראשון של מערך A למיקום השני של מערך B, ואת האיבר השני של A אל המיקום הראשון של B (012.asm).

```
DATA SEGMENT
  A DB 52H,46H
  B DB 2 DUP (?)
DATA ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV AL,A[0]
       MOV B[1],AL
       MOV AL,A[1]
       MOV B[0],AL
CODE ENDS
END START
```

דוגמה 3

מערך X כולל שמונה איברים שבהם מספרים. יש לכתוב תוכנית שתסכם את המספרים האלה ותציב את התוצאה באוגר AL.

פתרון: עלינו לפנות ל-8 איברים של המערך X, ולכן מובן שהפקודות הבאות מהוות פתרון **שאינו** מתאים:

```
MOV AL,X[0]
ADD AL,X[1]
ADD AL,X[2]
:
:
ADD AL,X[7]
```

בדומה לשפת פסקל למשל, נוכל להשתמש במונה שערכו ההתחלתי 0, אשר נקדם אותו ב-1 בכל פעם, וכך נוכל לכתוב פקודה אחת לטיפול בכל האיברים של המערך. נשתמש לצורך כך באוגר SI:

```
        MOV SI,0
        MOV AL,0
AGAIN:  ADD AL,X[SI]
        INC SI
        CMP SI,7
        JNG AGAIN
```

נניח שהמערך X מוגדר כך: X DB 2,1,3,2,5,3,4,5

נבחן את הפתרון באמצעות טבלת מעקב:

הפעולה	SI	X[SI]	AL (עשרוני)
אתחול	0		0
מחזור ראשון	0	2	2
מחזור שני	1	1	3
מחזור שלישי	2	3	6
מחזור רביעי	3	2	8
מחזור חמישי	4	5	13
מחזור שישי	5	3	16
מחזור שביעי	6	4	20
מחזור שמיני	7	5	25
סיום	8		25

הערה: בתוך הסוגריים המרובעים, המציינים את האיבר הסידורי במערך, מותר להשתמש באוגרים BX, SI ו-DI בלבד!

הסבר: כזכור, המשתנים של המערך הם למעשה תאי זיכרון. כאשר אנו כותבים את הפקודה MOV AH,X[SI] למשל, אנו מציבים ב-AH את ערך **תא הזיכרון** שהכתובת שלו נמצאת בהיסט SI, החל מהכתובת ההתחלתית **של המערך**. נחזור לעניין זה בהמשך.

שים לב לפקודות הבאות:

```
MOV SI,OFFSET X
ADD SI,2
MOV AL,[SI]
```

הן מבצעות פעולה זהה לפקודות אלו:

```
MOV SI,2
MOV AL,X[SI]
```

בשני המקרים פונים לתא הזיכרון המכיל את הנתון המבוקש, שכאן הוא האיבר השלישי במערך X.

נוכל לראות שכתובת התא מתקבלת על ידי חיבור **ההיסט בתוך המערך** עם **ההיסט של המערך** מתחילת המקטע.

זכור:

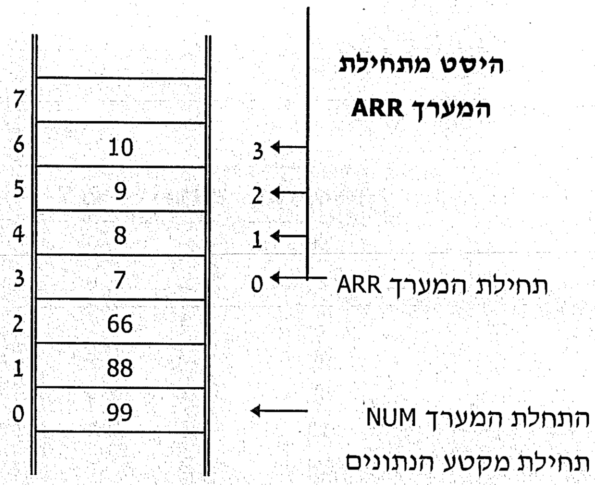
- **היסט בתוך המערך** הוא המרחק בבתים של האיבר מתחילת המערך.
- **היסט של המערך** הוא המרחק בתווים בין תחילת המקטע לבין תחילת המערך.

נבחן זאת באמצעות דוגמה (013.asm).

התוכנית

```
DATA SEGMENT
  NUM DB 99H,88H,66H
  ARR DB 7,8,9,10H
DATA ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:DATA
BEGIN:  MOV AX,DATA
        MOV DS,AX
        MOV SI,3
        MOV AL,ARR[SI]
CODE ENDS
END BEGIN
```

תמונת הזיכרון



הסבר: התוכנית כוללת שני מערכים. המערך הראשון NUM מכיל שלושה איברים בגודל בית, והמערך השני ARR מכיל ארבעה איברים בגודל בית. התוכנית מציבה באוגר AL את ערך האיבר הרביעי של מערך ARR.

במבנה הטבלה ניתן לראות את "תמונת מקטע הזיכרון" הכולל את המערכים. רישום המערכים בזיכרון נעשה לפי הגדרתם במקטע הנתונים. שים לב, שכל אחד מהאיברים נכנס לתא אחד, שגודלו בית אחד.

התא המכיל את הערך 99H הוא התא הראשון השייך למקטע הנתונים. בתא זה מוצב האיבר הראשון של מערך MUN. המערך ARR מתחיל מייד לאחר האיבר האחרון של MUN. ניתן לראות, שהמערך ARR נמצא בהיסט (מרחק) של שלושה תאי זיכרון מתחילת מקטע הנתונים.

האיבר המכיל את הנתון 10H נמצא בהיסט של שלושה תאים מתחילת המערך ARR. ההיסט של איבר זה **מתחילת המקטע** הינו **6**. כלומר, זהו התא ה-**7** מתחילת מקטע הנתונים.

דוגמה 4

במקטע הנתונים מוגדר המשתנה MISPAR1 שמכיל חמישה איברים בגודל **מילה** כל אחד; ומוגדר גם מערך MISPAR2 המכיל חמישה איברים שערכם 0. התוכנית צריכה להעתיק את חמשת האיברים של מערך MISPAR1 אל מערך MISPAR2 (014.asm).

```
VAR SEGMENT
  MISPAR1 DW 12H,9892H,2244H,0,8FFFH
  MISPAR2 DW 5 DUP (0)
VAR ENDS
TAR SEGMENT
  ASSUME CS:TAR,DS:VAR
BEG:    MOV AX,VAR
        MOV DS,AX
        MOV CX,5
        MOV DI,0
CONT:   MOV AX,MISPAR1[DI]
        MOV MISPAR2[DI],AX
        ADD DI,2
        LOOP CONT
TAR ENDS
END BEG
```

שים לב, שהעתקת נתון ממערך אחד אל מערך אחר, מחייבת שימוש באוגר מתווך מתאים. כמו כן, המצביע DI מתקדם ב-2 בכל פעם, כדי להגיע לאיבר הבא (בניגוד לשפות עיליות, בהן יש לקדם את המצביע ב- 1 בכל מקרה).

דוגמה 5

כתוב תוכנית שבמקטע הנתונים שלה יוגדרו המספרים הבאים (קרא משמאל לימין):

9, 5, 0, 2, 8, 7, 5, 1, 3

התוכנית תחבר את המספרים ותציב את התוצאה במשתנה בשם TOTAL. נניח שהתוצאה היא בגודל בית אחד בלבד.

פתרון א':

לפתרון בעיה זו נגדיר את המספרים האלה במערך (015.asm).

```
DATA SEGMENT
  MIS DB 9,5,0,2,8,7,5,1,3
  TOTAL DB 0
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV CX,9
        MOV BX,0
SHOOV:  MOV DH,MIS[BX]
        ADD TOTAL,DH
        INC BX
        LOOP SHOOV
CODE ENDS
END START
```

פתרון ב':

אין חובה להגדיר את הנתונים כמערך. הנתון הראשון שיוגדר במקטע הנתונים יהיה בכתובת 0, הנתון הבא בכתובת 1 וכדומה (016.asm).

```
DATA SEGMENT
  DB 9,5,0,2,8,7,5,1,3
  TOTAL DB 0
DATA ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV BX,0          הנתון הראשון נמצא בכתובת 0
        MOV CX,9
AGAIN:  MOV DL,[BX]
        ADD TOTAL,DL
        INC BX
        LOOP AGAIN
CODE ENDS
END START
```

דוגמה זו הובאה גם כדי להבהיר שניתן להגדיר במקטע נתונים ללא משתנים. שים לב, שלו היינו מגדירים את המשתנה TOTAL לפני הנתונים, היתה שגיאה בתוכנית (חשוב מדוע!).

דוגמה 6

בתוך מקטע נתונים מוגדרים שלושה משתנים: משתנה NATUN1 בגודל בית, משתנה NATUN2 בגודל בית ו-KTOVET בגודל מילה. נתון שהמשתנה NATUN1 מכיל ערך קטן יותר מזה של משתנה NATUN2.

נכתוב את גוף התוכנית המציבה את הערכים בתחום NATUN1 עד NATUN2, החל מתא זיכרון שכתובתו נתונה ב-KTOVET. כלומר, תא הזיכרון הראשון יכיל את הערך הרשום ב-NATUN1, התא הבא יכיל ערך הגדול ממנו ב-1 וכך הלאה, עד לערך הרשום ב-NATUN2.

התוכנית:

```
        MOV BX,KTOVET
        MOV AL,NATUN1
AGA:    MOV [BX],AL
        INC BX
        INC AL
        CMP AL,NATUN2
        JNG AGA
```

תרגילים

1. כתוב תוכנית, שבמקטע הנתונים שלה מוגדרים שלושה משתנים: NO1 שערכו 23H, NO2 שערכו 78H ו-RES שאינו מאותחל.

   התוכנית צריכה לחבר את NO1 עם NO2 ולהציב את התוצאה ב-RES.

2. כתוב תוכנית הכוללת מקטע נתונים ומקטע קוד. במקטע הנתונים יוגדר משתנה בגודל **מילה** בשם KLITA ללא אתחול. במקטע הקוד יירשמו הפקודות אשר יגרמו להצבת ערך התא בכתובת 480H בבית התחתון של המשתנה KLITA, ולהצבת ערך תא 612H בבית העליון של המשתנה.

   זכור, שביקשנו שהמשתנה KLITA יהיה בגודל מילה, ועל כן הוא יכול להכיל את שני תאי הזיכרון, שגודל כל אחד מהם הוא בית אחד.

3. נתונה התוכנית הבאה (017.asm). מה יהיה תוכן אוגר BX, ומה יכיל DH לאחר שנריץ אותה?

```
TALMID SEGMENT
   BOY DB ?
  GIRL DB ?
   YES DB 1
    NO DB 2
TALMID ENDS
CODE SEGMENT
   ASSUME CS:CODE,DS:TALMID
```

```
HERE:   MOV AX,TALMID
        MOV DS,AX
        MOV BX,OFFSET GIRL
        MOV DH,[BX+1]
CODE ENDS
END HERE
```

4. נתון מקטע הנתונים הבא:

```
DAT SEGMENT
     DW 4
      DB 5
 BEST DB 2
DAT ENDS
```

- מהו ההיסט של המשתנה BEST מתחילת המקטע?
- רשום את הפקודה שתציב באוגר CL את הערך השני ממקטע הנתונים (זהו הערך 5).

5. במקטע הנתונים מוגדרים שלושה מערכים: ARRAY1 המכיל שישה איברים בגודל מילה כל אחד, ARRAY2 המכיל שישה איברים בגודל מילה, ARR המכיל שישה איברים ריקים.

   כתוב את גוף התוכנית המציבה במערך ARR את הערכים האלה:

   - באיבר הראשון היא תציב את סכום האיברים הראשונים של ARRAY1 ו-ARRAY2.
   - באיבר השני - את סכום האיברים השניים של ARRAY1 ושל ARRAY2, וכדומה.

6. כתוב תוכנית, שבמקטע הנתונים שלה מוגדר המשתנה MISHKAL המכיל עשרה נתונים בגודל בית. התוכנית תמצא את הערך הגדול ביותר ותציב אותו במשתנה BIG.

7. במקטע נתונים רשומים המשתנים האלה: ADDR המכיל ערך בן 16 סיביות, NATUN המכיל ערך בן 8 סיביות.

   התוכנית תציב את ערך המשתנה NATUN ב- 120H תאי זיכרון, החל בכתובת המוגדרת ב-ADDR.

8. כתוב תוכנית שתקלוט 15 הקשות מקש. התוכנית תציב במשתנה CHAR את מספר ההקשות של מקשי אותיות (a עד z), במשתנה NUM היא תציב את מספר ההקשות של מקשי ספרות (0 עד 9) ובמשתנה ELSE - את מספר ההקשות של מקשים אחרים.

9. במערך RESULTS נתונים 12 איברים בגודל בית. כתוב תוכנית המעתיקה למערך SIGN שבו 12 איברים שאינם מאותחלים, את ערך סיבית הסימן של כל איבר של

RESULT. כלומר, האיבר הראשון של SIGN יכיל "0" או "1" בהתאם לסיבית הסימן של האיבר הראשון של RESULT וכן לגבי האיבר השני, עד האיבר ה-12.

10. מערך CHAR מכיל 20 איברים בגודל בית. מערך ATT מכיל אף הוא 20 איברים בגודל בית. כתוב תוכנית שתציב את האיברים של CHAR בתאים הזוגיים, החל מהתא בכתובת **פיסית** B8000H, ואת האיברים של ATT היא תציב בתאים האי-זוגיים החל מכתובת **פיסית** B8001H.

כתוב את התוכנית כך שניתן יהיה להריצה ישירות ממערכת ההפעלה, ובדוק את אופן הביצוע.

11. הסבר בקצרה מה מבצעת כל אחת מהתוכניות האלו:

תוכנית א'

```
D SEGMENT
  SAY DB 61H
  TALK DB ?
D ENDS
C SEGMENT
  ASSUME CS:C,DS:D
HER: MOV AX,D
     MOV DS,AX
     MOV BH,SAY
     MOV CL,4
     SHR BH,CL
     AND SAY,0FH
     ADD BH,SA
     MOV TALK,BH
C ENDS
END HER
```

תוכנית ב'

```
D SEGMENT
  4 DUP (66H)
  STAR DW 100H
D ENDS
C SEGMENT
  ASSUME CS:C,DS:D
ONE:MOV AX,D
     MOV DS,AX
     MOV SI,STAR
     MOV BX,0
     MOV CX,4
YES: MOV AL,[BX]
     MOV [SI],AL
     INC BX
     INC SI
     LOOP YES
C ENDS
END ONE
```

תוכנית ג'

```
DA SEGMENT
  GOOD DB 78,95,82,97
  BAD DB 33,48,53,0
DA ENDS
CO SEGMENT
 ASSUME CS:CO,DS:DA
FIR: MOV AX,DA
     MOV DS,AX
     MOV AL,0
     MOV CX,4
     MOV SI,0
KEN:MOV AH,GOOD[SI]
     SUB AH,BAD[SI]
     ADD AL,AH
     INC SI
     LOOP KEN
CO ENDS
END FIR
```

12

המחסנית

מחסנית (Stack) מהווה חלק מזיכרון המחשב, המוקצה לשמירת נתונים זמניים. שני גורמים זקוקים למחסנית:

1. **המיקרומעבד:** בעת ביצוע פקודות מסוימות צריך המיקרומעבד לשמור באופן זמני מספר נתונים. לדוגמה, כאשר הוא מבצע פסיקה (הפקודה INT) הוא יוצא מהתוכנית וניגש לכתובת חדשה במערכת DOS או BIOS, לביצוע תוכנית הפסיקה.

   כיצד יודע המעבד, אשר מסיים את תוכנית הפסיקה, לאיזו כתובת לחזור להמשך ביצוע התוכנית שלנו? המיקרומעבד שומר במחסנית, באופן אוטומטי, את כתובת החזרה לתוכנית, **לפני** שהוא יוצא מהתוכנית הראשית. לאחר סיום תוכנית הפסיקה, הוא "שולף" מהמחסנית את הכתובת של התוכנית הראשית וחוזר אליה להמשך ביצועה (נחזור לדון בעניין זה בנושא וקטור הפסיקות).

2. **המתכנת:** קיימות מספר פקודות העומדות לרשות המתכנת המאפשרות לו להשתמש במחסנית. המתכנת באסמבלי עשוי להשתמש במחסנית במקרים כאלה:
   - לשמירת נתונים באופן זמני ואחזורם מאוחר יותר.
   - להעברת נתונים בין שגרות וקטעי תוכניות.
   - להעברת נתונים בין תוכניות שונות, גם כאלה הכתובות בשפות שונות, כמו למשל אסמבלי, פסקל, שפת C וכדומה.

אופן שמירת הנתונים במחסנית ואחזורם שונה ומיוחד: הנתון **האחרון** שנכנס, הוא **הראשון** שיוצא. זהו **עיקרון LIFO** (Last In First Out), ומכאן גם שם המחסנית. הדבר דומה למחסנית כדורים ברובה: הכדור הראשון שיוצא הוא האחרון שהוכנס למחסנית.

לדוגמה, למחסנית הוכנסו הערכים 60H, 70H ו- 80H, לפי סדר זה. כעת מוציאים נתון מהמחסנית. הנתון הראשון שנוציא יהיה 80H.

הגדרת המחסנית

הגדרת מחסנית אינה הכרחית בכל תוכנית. אך, **חובה** להגדיר מחסנית במקרים האלה:

- ❖ התוכנית כוללת פסיקות: המעבד שומר במחסנית את כתובת החזרה לתוכנית הראשית ואת אוגר הדגלים.
- ❖ התוכנית כוללת שגרות: המעבד שומר את כתובת החזרה **לפני** היציאה לשגרה.
- ❖ המתכנת משתמש בפקודות המטפלות במחסנית.

כדי להגדיר מחסנית בתוכנית, דרושות שלוש פעולות אלו:

1. להגדיר מקטע חדש בשם כלשהו ולהוסיף את המילה השמורה STACK.
2. להגדיר את גודל המחסנית באמצעות הקצאת מספר רצוי של תאי זיכרון.
3. להכריז על אוגר SS בהוראה ASSUME (תזכורת, שאינה קשורה להגדרה זו).

לפניך דוגמת תכנות להגדרת מחסנית, שבה נגדיר מחסנית בשם GOOD, בגודל 170H בתים (תאי זיכרון):

```
GOOD SEGMENT STACK
DB 170H DUP (?)
GOOD ENDS
```

המילה STACK נרשמת לאחר המילה SEGMENT ועל ידי כך אנו מודיעים שמקטע זה הוא מסוג מחסנית.

ההוראה "DB 170H DUP (?)" גורמת להקצאה של 170H תאים שאינם מאותחלים, וכך היא קובעת את גודל המחסנית.

בהוראה ASSUME נהגנו לכתוב: ASSUME CS:CODE,DS:DATA

עתה נוסיף גם את ההכרזה על המחסנית: ASSUME CS:CODE,DS:DATA,SS:GOOD

כלומר, אנו מגדירים שאוגר SS יכיל את כתובת תחילת המחסנית, שהינה תחילת המקטע GOOD. כתובת זו היא גם "שם המחסנית" שצריך להתאים לכללי הכתיבה של שמות משתנים ושל שמות תוויות.

הפקודה PUSH

הפקודה PUSH (דחיפה) מעתיקה לתוך המחסנית נתון בגודל **מילה שלמה בלבד**. נציג מספר דוגמאות:

PUSH SI	מעתיקה למחסנית את ערך אוגר SI.
PUSH CX	מעתיקה למחסנית את ערך אוגר CX.
PUSH [BX]	מעתיקה למחסנית את ערך שני התאים הצמודים: זה שבכתובת ש-BX מצביע עליה וזה שבכתובת BX+1.
PUSH DL	פקודה זו **אינה** חוקית (גודל המשתנה הוא חצי מילה)!

הפקודה POP

הפקודה POP שולפת מתוך המחסנית את הנתון האחרון שהוכנס אליה, בגודל **מילה שלמה**. דוגמאות:

POP DX	מעתיקה אל DX את הנתון האחרון שהוכנס למחסנית.
POP [DI]	מעתיקה את הנתון האחרון מהמחסנית אל צמד התאים בכתובות DI ו- DI+1.
POP BH	שגיאה (חצי מילה)!
POP MY_DATA	מעתיקה את הנתון האחרון מהמחסנית, לתוך משתנה שגודלו מילה.

תוכניות לדוגמה

דוגמה 1

מה מבצעת התוכנית הזו?

```
MOV AX,12H
MOV BX,8844H
PUSH BX
POP AX
```

תשובה: התוכנית מציבה במחסנית את ערך אוגר BX (שערכו 8844H), לאחר מכן שולפת את הנתון האחרון שהוכנס (שערכו 8844H) לתוך אוגר AX. כלומר, התוכנית מציבה באוגר AX את הערך של אוגר BX.

דוגמה 2

מה מבצעת התוכנית הזו?

```
PUSH AX
PUSH BX
POP AX
POP BX
```

תשובה: התוכנית מחליפה בין ערכי האוגרים BX ו-AX.

למחסנית מוכנסים ערכי האוגרים AX ו-BX, לפי סדר זה. לאחר מכן, שולפים מהמחסנית את הנתון האחרון שהוכנס (ערך BX) ומציבים אותו באוגר AX (הפקודה POP AX). לבסוף, שולפים את הנתון הקודם שהוכנס (ערך AX) ומציבים אותו באוגר BX (הפקודה POP BX).

דוגמה 3

יש לכתוב פקודה, או פקודות, להעתיק למשתנה בגודל מילה STORE את הנתון שהוכנס למחסנית **לפני** הנתון האחרון.

נשלוף את הנתון האחרון ונציב ב-AX כדי להוציאו מהמחסנית POP AX
נשלוף את הנתון שלפני האחרון ונציב במשתנה POP STORE

הסבר: הפקודה הראשונה POP AX, מוציאה את הנתון האחרון שהוכנס למחסנית, כך שבפעולת POP הבאה יישלף הנתון המבוקש.

דוגמה 4

התוכנית הבאה מבצעת את הפעולות הבאות:

- קולטת ספרה מהמקלדת.
- שומרת את ערכה במחסנית.
- קולטת ספרה נוספת ומציבה אותה במשתנה SIFRA.
- שולפת מהמחסנית את הספרה הראשונה ומחברת אותה למשתנה SIFRA.

התוכנית (001.asm):

```
DATA SEGMENT
  SIFRA DB 0
DATA ENDS
M SEGMENT STACK
  DB 5 DUP (?)
M ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:DATA,SS:M
MAIN:   MOV AX,DATA
        MOV DS,AX
        MOV AH,1
        INT 21H
        AND AL,0FH
        PUSH AX
        MOV AH,1
        INT 21H
        AND AL,0FH
        MOV SIFRA,AL
        POP DX
        ADD SIFRA,DL
CODE ENDS
END MAIN
```

במקטע הנתונים הוגדר המשתנה בגודל בית. במקטע המחסנית M הוגדרו חמישה תאים שאינם מאותחלים. נוכל להגדיר מספר שונה של תאים, ובלבד שיספיקו להכיל את הנתונים המוכנסים למחסנית.

לאחר קליטת המקש, מבוצע מיסוך על הספרה השמאלית (הפקודה: AND AL,0FH), כדי שתיוותר הספרה בלבד. לא ניתן לשמור את הספרה בלבד, על ידי הפקודה PUSH AL, ולכן שומרים את AX בשלמותו (PUSH AX). הדבר אינו יוצר כל בעיה, מכיון שאחר כך נשלוף את הנתון חזרה לאוגר ונתייחס לחציו הימני בלבד.

בהמשך התוכנית קולטים ספרה נוספת וערכה מוצב במשתנה SIFRA (לאחר מיסוך). את הנתון מהמחסנית שולפים לאוגר DX. אנו יכולים להשתמש בכל אוגר בגודל מילה. בחרנו באוגר DX באופן שרירותי. מתוך הנתון המתקבל באוגר DX, אנו מעוניינים בחלק הימני בלבד (אוגר DL), ולכן מוסיפים אותו למשתנה.

דוגמה 5

התוכנית הבאה מכילה משתנה בגודל מילה כפולה בשם MISH. ערכו של המשתנה יועתק למחסנית.

פתרון א' (002.asm):

```
D SEGMENT
   MISH DD 10203045H
D ENDS
M SEGMENT STACK
   DW 2 DUP (0)
M ENDS
C SEGMENT
  ASSUME CS:C,DS:D,SS:M
START:  MOV AX,D
        MOV DS,AX
        PUSH WORD PTR MISH
        PUSH WORD PTR MISH+2
C ENDS
END START
```

גודל המחסנית נקבע לשתי מילים (4 בתים), כך שניתן להכניס בה את הנתון בגודל מילה כפולה. נוכל גם להגדיר מחסנית גדולה יותר מהנדרש.

מכיון שהפקודה PUSH מאפשרת להכניס נתון בגודל מילה בלבד, משתמשים ב-WORD PTR (גודל מילה). בתחילה מבצעים PUSH למילה הנמוכה של המשתנה (16 הסיביות הנמוכות, המספר 3045H) ולאחר מכן, מוצבת המילה הגבוהה של המשתנה. הרישום MISH+2 מתייחס לבית בהיסט 2 מתחילת המשתנה (זהו הבית השלישי). הפקודה "PUSH WORD PTR+2" מציבה במחסנית את הבית השלישי ואת הבית הרביעי של המשתנה, המייצגים את המילה השנייה (16 הסיביות הגבוהות, המספר 1020H).

פתרון ב' (גוף התוכנית בלבד):

```
MOV SI,OFFSET MISH
PUSH [SI]
PUSH [SI+2]
```

באוגר SI מוצבת כתובת ההתחלה של המשתנה. זהו ההיסט מתחילת המקטע שבו הוא מוגדר.

הפקודה [PUSH [SI מכניסה למחסנית את המילה הנמוכה של המשתנה MISH.

הפקודה [PUSH [SI+2 מציבה את המילה הגבוהה של המשתנה.

דוגמה 6

התוכנית בדוגמה זו, מעתיקה לתוך המחסנית 10 תאי זיכרון החל מכתובת 300H. לאחר מכן, היא שולפת את הנתונים מהמחסנית אל התאים 400H-409H, באופן כזה שהם יכילו ערכים זהים בהתאמה לערכי התאים המקוריים 300H-309H.

התוכנית (003.asm):

```
MA SEGMENT STACK
  DB 9 DUP (0)
MA ENDS
CO SEGMENT
  ASSUME CS:CO,DS:CO,SS:MA
BEGIN:  MOV AX,CO
        MOV DS,AX
        MOV BX,300H
        MOV CX,5
TAKE:   MOV DX,[BX]
        PUSH DX
        ADD BX,2
        LOOP TAKE
        MOV BX,408H
        MOV CX,5
GIVE:   POP DX
        MOV [BX],DX
        SUB BX,2
        LOOP GIVE
CO ENDS
END BEGIN
```

שים לב למספר פעולות חשובות בתוכנית זו:

- מונה הלולאה CX מקבל את הערך 5 (מדוע?).
- כדי להעתיק את הנתונים לתאי הזיכרון שבכתובות 400H-409H, מתחילים בכתובת 408H על ידי הפקודה "MOV BX,408" (מדוע?).

איך פועלת המחסנית

נניח שכתבנו את התוכנית הבאה (004.asm):

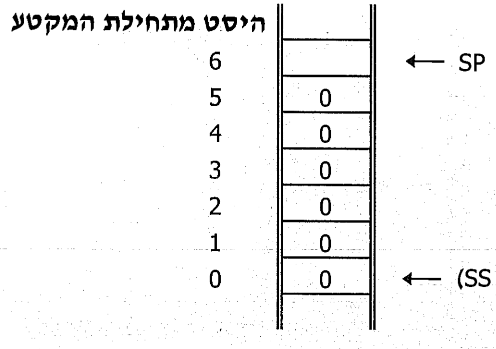
```
MA SEGMENT STACK                          שלב א
  DB 6 DUP (0)
MA ENDS
CODE SEGMENT
  ASSUME CS:CODE,DS:CODE,SS:MA
FIRST:  MOV AX,1234H
        PUSH AX                           שלב ב
        MOV BX,7890H
        PUSH BX                           שלב ג
        POP CX                            שלב ד
CODE ENDS
END FIRST
```

כיצד תיראה המחסנית MA בכל שלב של התוכנית?

כדי להשיב על כך, נשרטט את מצב המחסנית בשלבי פעולה שונים:

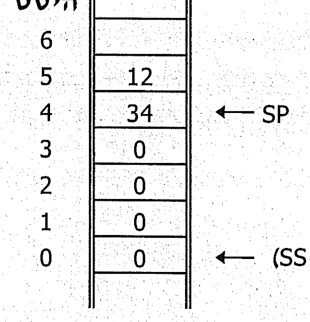
- **שלב א** - כתוצאה מהגדרת מקטע המחסנית, נקבל את המבנה הבא:

אוגר SS מכיל את כתובת הבסיס של המחסנית. אוגר SP מקבל את הערך 6 ומצביע על הכתובת שמתחתיה יוכנסו נתונים. התאים במחסנית מקבלים ערך 0.



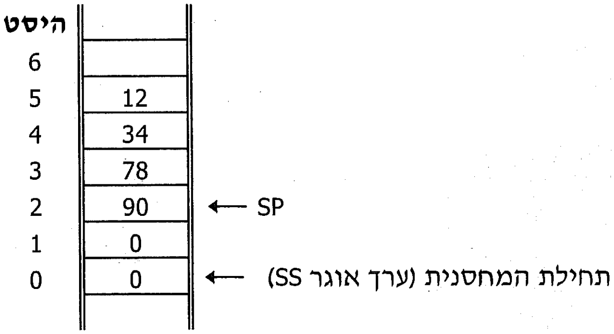
- **שלב ב** - לאחר ביצוע הפקודה PUSH AX:

הנתונים מוצבים בתאים 5 ו-4. כרגיל, הבית הגבוה מוצב בתא הגבוה יותר. פעולת PUSH מציבה את הערך בשני תאי זיכרון שמתחת ל-SP. ערך SP יורד ב-2. גם כעת מצביע SP על הכתובת, שמתחתיה יוכנסו נתונים למחסנית.



❖ **שלב ג - לאחר ביצוע הפקודה PUSH BX:**

ערך BX מוצב במחסנית. ערך SP יורד ב-2.

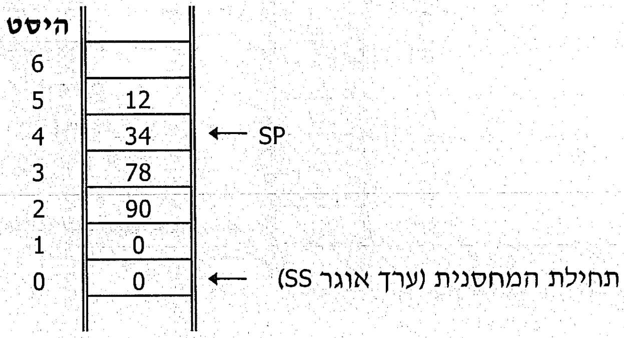


❖ **שלב ד - לאחר ביצוע הפקודה POP CX:**

CX מקבל את ערך הנתון שנמצא בכתובת ש-SP מצביע עליה (2), וערך התא שמעליה (3). הפקודה POP שלפה נתונים מתא שכתובתו SP ומתא SP+1. אוגר SP עולה ב-2, ונמצא בכתובת 4.

אם בתוכנית היתה מתבצעת הפקודה PUSH, הנתון היה נכנס לתאים 3 ו-2, ומוחק את הנתונים SS שנמצאים בהם.

אילו הינו מבצעים POP, הינו מקבלים את ערכי התאים 4 ו-5.



לסיכום, **דחיפה** (PUSH) של נתונים למחסנית גורמת להצבת הנתון בשני תאי זיכרון שבכתובות SP-1, SP-2. **שליפה** (POP) של נתונים מהמחסנית גורמת להעתקת הנתון משני תאי זיכרון בכתובות SP, SP+1.

הכנסה והוצאה של נתונים מהמחסנית

המחסנית מאפשרת הוצאה והכנסה של נתונים לפי **עיקרון LIFO**. ברם, לעיתים אנו צריכים להעתיק נתון "פנימי" יותר. כדי לבצע זאת, מבלי להוציא את כל הנתונים שלפניו (כפי שעשינו באחד התרגילים הקודמים), משתמשים באוגר BP.

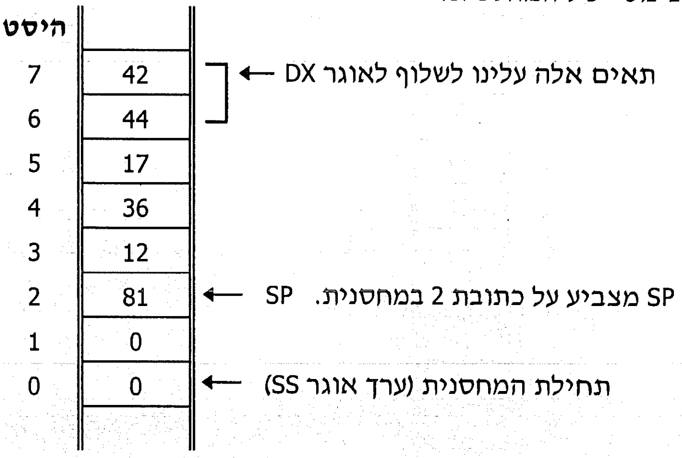
אוגר BP מאפשר להצביע על תאי זיכרון הנמצאים **בתוך** המחסנית. נראה למשל, את הפקודות האלו:

```
MOV BP,6
MOV AL,[BP]
```

פקודות אלו מעתיקות את ערך תא הזיכרון שבכתובת 6 בתוך המחסנית אל אוגר AL. יש לזכור, שהכתובת של תא הזיכרון הינה **היסט** מתחילת המקטע. במקרה זה, זהו תא הזיכרון בכתובת 6 החל מתחילת המחסנית ולמעשה, התא השביעי במחסנית. נראה דוגמה:

המטלה: נתונה מחסנית. עלינו להציב באוגר DX את ערך התאים 6 ו-7 שבמחסנית.

תיאור סכימטי של המחסנית:



כדי לבצע את הפעולה בשיטה הרגילה, נכתוב את הפקודות האלו:

```
     MOV CX,3
GET: POP DX
     LOOP GET
```

בפעם הראשונה יישלפו ערכי התאים 2 ו-3 ויוצבו באוגר DX. ואז SP יעלה ב-2 ויצביע על כתובת 4. במחזור השני של הלולאה יוצבו ב-DX ערכי התאים 4 ו-5 (SP יעלה ל-6). במחזור השלישי והאחרון של הלולאה, יוצבו באוגר DX הערכים המבוקשים.

שים לב, שבדרך זו אנו משנים את מיקום מצביע המחסנית (אוגר SP)! דבר זה עלול לגרום לבעיה, כי אם יבוצע PUSH מספר פעמים, הנתונים החדשים ייכנסו למחסנית **במקום** הנתונים הקיימים.

כדי לפתור בעיה זו, נשמור את ערך SP **לפני** שליפת הנתונים, ונחזירו לערכו המקורי לאחר השליפה:

```
         MOV AX,SP
         MOV CX,5
GET:     OP DX
         LOOP GET
         MOV SP,AX
```

באמצעות אוגר BP ניתן לבצע את שליפת הנתונים בתאים 6 ו-7 על ידי שתי פקודות אלו בלבד:

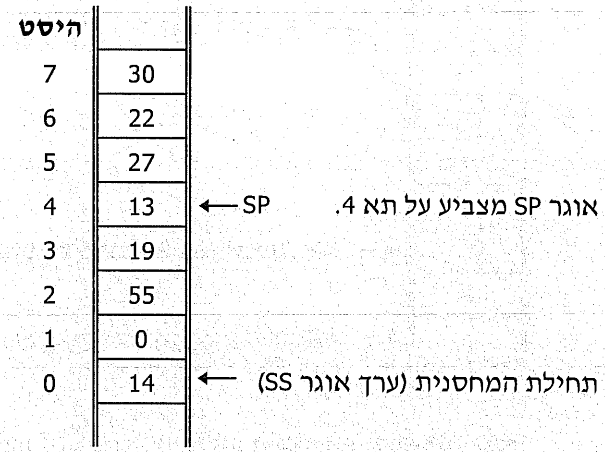
```
MOV BP,6
MOV DX,[BP]
```

פתרון זה פשוט ואלגנטי יותר.

קיימים מצבים שבהם קשה להשתמש בפקודה POP כדי לשלוף נתון מהמחסנית. ניתן לראות זאת בדוגמה הבאה.

המטלה: לשלוף את הנתון שמאוחסן בתאים 2 ו-3.

נוכל לראות שלא נוכל לעשות זאת באופן פשוט באמצעות הפקודה POP.



הפתרון הנכון הוא באמצעות BP:

```
MOV BP,2
MOV DX,[BP]
```

ניתן לגשת לתאי המחסנית גם באמצעות אוגרים מצביעים אחרים, באמצעות **אילוץ מקטע** (prefix):

```
MOV SI,2
MOV DX,SS:[SI]
```

הסבר: באופן רגיל, SI משמש כמצביע על היסט מהבסיס DS. כאשר אנו כותבים SS:[SI], אנו **מאלצים** את SI להצביע על היסט החל מהבסיס SS. כלומר, מתחילת המחסנית.

קיימים מקרים נוספים שבהם משתמשים באוגר BP, בעיקר כאשר משתמשים במחסנית לצורך העברת נתונים בין תוכניות הכתובה בשפה אחת (פסקל, C וכדומה) לבין פרוצדורה הכתובה באסמבלי.

במקרים אלה, מעוניינים להעתיק מהמחסנית נתון שנמצא בהיסט מסוים מהמיקום של SP. לדוגמה, אם מעוניינים להעתיק את הנתון שנמצא בהיסט של 4 תאים מ-SP, נכתוב:

```
MOV BP,SP
MOV AL,[BP+4]
```

בדיקת המחסנית באמצעות תוכנית ניפוי

כדי לבדוק את תוכן המחסנית, יש לבדוק את ערך האוגר SP. ערכו נקבע בהתאם להגדרת גודל המחסנית בתוכנית. אם כתבנו במקטע המחסנית את הפקודה:

```
DB 15H DUP (?)
```

ערך אוגר SP יהיה 15H. המחסנית תתחיל בהיסט 0 מהמקום המוצבע על ידי אוגר SS ותסתיים בתא שבהיסט 14H. זכור, המחסנית מסתיימת תמיד בערך הקטן ב-1 מערך SP.

אם רשמנו במקטע המחסנית: DW 12H DUP (?)

נקבל באוגר SP את הערך 24H (DW מגדיר גודל מילה, או שני תאים).

המחסנית תסתיים בכתובת 23H.

באמצעות ההוראה D של DEBUG, ניתן לראות את המחסנית.

בדוגמה הראשונה: DB 15H DUP (?) נכתוב: D SS:0 14

בדוגמה השנייה: DW 12H DUP (?) נכתוב: D SS:0 23

הסבר: כל הכתובות יהיו החל מתחילת המחסנית, כלומר החל מערך SS.

- שים לב שהכתובות הגבוהות יכילו את הנתונים הראשונים שייכנסו.
- בחן את השינוי של אוגר SP בכל שליפה, או הכנסת נתונים למחסנית.
- זכור שהנתונים נכנסים בשני תאים עוקבים, כך שהבית הגבוה יותר של הנתון מוצב בכתובת הגבוהה יותר.

תרגילים

1. שרטט כיצד תיראה המחסנית (כולל מיקום SP) **לאחר** ביצוע כל אחת מהתוכניות הבאות. הסבר מה עושה כל תוכנית:

תוכנית ב' (006.asm)	**תוכנית א'** (005.asm)
S SEGMENT STACK	S SEGMENT STACK
DW 4 DUP (11H)	DB 5 DUP (0)
S ENDS	S ENDS
C SEGMENT	C SEGMENT
ASSUME CS:C,SS:S	ASSUME CS:C,SS:S
BEG: MOV AX,0FFH	STA: MOV AX,481H
MOV CX,3	PUSH AX
YES: INC AX	SUB AX,2
PUSH AX	PUSH AX
LOOP YES	C ENDS
MOV BP,2	END STA
MOV AL,[BP]	
C ENDS	
END BEG	

2. לפניך תרגיל מחשבה. כתוב את התוכנית הבאה (007.asm):

```
STORE SEGMENT STACK
  DB 20H DUP (0)
STORE ENDS
CODE SEGMENT
  ASSUME CS:CODE,SS:STORE
STAR:   MOV DL,'1'
        MOV AH,2
        INT 21H
        NOP
CODE ENDS
END STAR
```

הרץ את MASM ו-LINK. טען את התוכנית באמצעות DEBUG ובצע את הפעולות הבאות:

א. בדוק אילו נתונים מכילה המחסנית (השתמש בפקודה: D SS:0 1F).

ב. בצע T פעם אחת ובדוק שוב את המחסנית. האם יש שינוי?

ג. צא מ-DEBUG וטען שוב את התוכנית. הרץ כעת את התוכנית בשלמותה באמצעות G. בדוק את המחסנית, והסבר:

- ❖ מדוע מופיעים נתונים במחסנית, למרות שלא ביצענו PUSH?
- ❖ מה מייצגים הערכים שקיבלת?

3. כתוב תוכנית שמעתיקה למשתנה DEMY בגודל בית, את ערך התא בכתובת 3 שבמחסנית.

4. בדוק והשב: מה קורה כאשר מכניסים נתונים למחסנית **מעבר** לקיבולת שלה כפי שהוגדרה (גודל)?

5. כתוב תוכנית המגדירה 8 תאים במחסנית ללא אתחול. התוכנית תספור כמה תאי מחסנית מכילים ערך שלילי, ותציב את התוצאה במשתנה NRGATIVE.

6. לפניך קטע של תוכנית. מאיזה סוג מקטע מועתק הנתון לאוגר BL: מקטע קוד, מקטע מחסנית, או מקטע נתונים?

```
MOV SI,1
PUSH CS
POP DS
MOV BL,[SI]
```

ומה תהיה התשובה במקרה הבא?

```
MOV BP,1
MOV BL,DS:[BP]
```

7. כתוב תוכנית שבה מוגדרת מחסנית בגודל 10 תאים, ללא אתחול. התוכנית תציג על המסך את הערכים של עשרת התאים.

8. כתוב תוכנית, שבה:
   - מקטע מחסנית בגודל 12H תאים שאינם מאותחלים.
   - מקטע נתונים הכולל מערך בגודל 12H בתים המכילים ערך 0.
   - מקטע קוד הכולל פקודות המעתיקות למערך את 12H תאי המחסנית.

9. כתוב תוכנית המציבה את הערכים 0 עד 7 ב-8 תאי המחסנית.

10. כתוב תוכנית הקולטת שני מספרים מהמקלדת. המספר הראשון מכיל שתי ספרות והמספר השני מכיל ספרה בודדת.

    התוכנית תציב את המספר הדו-ספרתי בתוך המחסנית, בכתובת המצוינת על ידי הספרה הבודדת.

11. כתוב תוכנית המציגה על המסך את מספר התאים של המחסנית (מספר דו-ספרתי).

[illegible]

[illegible]

[illegible]

[illegible]

13

פרוצדורות

בהנחה שכתבת תוכנית בשפה עילית כלשהי, מושג הפרוצדורה ודאי נהיר עבורך. נזכיר שהשימוש בפרוצדורות מביא למספר רב של יתרונות, וביניהם:

- גודל התוכנית המוכנה להרצה קטן (executable). משמע שהתוכנית זקוקה לפחות מקום בזיכרון כדי לפעול.
- התוכנית הופכת לברורה יותר.
- תחזוקת התוכנית קלה יותר, מכיון שאם יש צורך לבצע תיקונים במנגנון הפרוצדורה, די לעשות זאת בקטע הקוד של הפרוצדורה עצמה - במקום לבצע תיקונים רבים במקומות שונים (כמו במקרה שלא נעשה שימוש בפרוצדורה).

הבנה טובה של מנגנון הפרוצדורה בשפת אסמבלי, תסייע לך בהבנת מנגנון הפרוצדורות/הפונקציות בשפות עיליות. כך תוכל להגיע לתכנות יעיל יותר גם בשפה עילית.

הגדרת הפרוצדורה

בתוך סגמנט התוכנית אנו מגדירים וכותבים את הפרוצדורה. בצורה הפשוטה ביותר, פרוצדורה מתחילה בתווית ומסתיימת בפקודה RET (RETurn - חזרה).

הקריאה לפרוצדורה מתוך התוכנית נעשית על ידי הפקודה CALL, מלווה בשם הפרוצדורה, שהיא למעשה שם התווית אליה יש לקפוץ. למשל, אם נכתוב CALL GOOD תבוצע קפיצה אל קוד הפרוצדורה שמתחילה בתווית GOOD, ומסתיימת בפקודה RET. הפקודה RET תגרום לחזרה להמשך התוכנית הקוראת לפרוצדורה.

הדוגמאות הבאות מציגות את השימוש בפרוצדורות.

```
;-----------------------------------------------------------------------------------------
; Name:  proc1.asm
; Task:  The program displays a message using a procedure
;-----------------------------------------------------------------------------------------
Data segment
  Message db 'Wellcome to procedures...',10,13,'$'
Data ends
Code segment
  assume cs:Code,ds:Data
Main:     mov ax,Data
          mov ds,ax
          call PrintMessage   ; Call the procedure
          mov ax,4c00h
               int 21h             ; Terminate
;---------------------------Here the procedure starts--------------------------------------
PrintMessage:
          mov dx,offset Message
          mov ah,9
          int 21h
          ret               ; Return from the procedure
;-------------------------------End of the procedure----------------------------------------
Code ends
    end Main
```

```
;-----------------------------------------------------------------------------------------
; Name: proc2.asm
; Task: This program uses a procedure to check if a number is positive
;-----------------------------------------------------------------------------------------
Data segment
   Number           db  83h
   NegativeMessage   db  'Number is negative...',10,13,'$'
   PositiveMessage   db  'Number is positive...',10,13,'$'
Data ends

Code segment
  assume cs:Code,ds:Data
Main:   mov ax,Data
      mov ds,ax
      call CheckNumber              ; Call the procedure CheckNumber
      call DisplayMessageForNumber  ; Call DisplayMessageForNumber
      mov ax,4c00h
      int 21h            ; Terminate
;-----------------------------------------------------------------------------------------
; Name:     CheckNumber
; Task:     Checks if Number value is positive or negative
; Remarks: 0 is considered as a positive value
; Input:    Number (as a global variable)
; Output:   in AX:  1 if Number is positive, 0 if it is negative
;-----------------------------------------------------------------------------------------
```

```
CheckNumber:
        mov     ax,0
        cmp     Number,0
        jl      CheckNumberDone
        mov     ax,1
CheckNumberDone:
        ret                 ; Return from the procedure
;--------------------------------------------------------------------------------------------
; Name:   DisplayMessageForNumber
; Task:   Displays a meesage according to Number value
; Input:  AX (1 if Number is positive, 0 if it is negative)
; Output: None
;--------------------------------------------------------------------------------------------
DisplayMessageForNumber:
        mov     dx,offset NegativeMessage
        cmp     ax,0
        je      Print
        mov     dx,offset PositiveMessage
Print:
        mov     ah,9
        int     21h
        ret                 ; Return from the procedure
;--------------------------------------------------------------------------------------------
Code ends
    end Main
```

שים לב, שבין התוכנית לבין הפרוצדורה יש הפרדה מוחלטת, כדי לוודא שקוד הפרוצדורה יתבצע רק כאשר יש קריאה באמצעות הפקודה CALL.

הנה דוגמה נוספת לשימוש בפרוצדורה:

```
;--------------------------------------------------------------------------------------------
; Name: proc3.asm
; Task:  The program compares the two numbers NUM1 and NUM2, and
;        copies the bigger one to RESULT
;--------------------------------------------------------------------------------------------

;--------------------------------------------------------------------------------------------
;                              GLOBAL VARIABLES
;--------------------------------------------------------------------------------------------
VARIABLES segment
      NUM1    db  4
      NUM2    db  22
      RESULT  db  0
VARIABLES ends
;--------------------------------------------------------------------------------------------
;                              MAIN  PROGRAM
;--------------------------------------------------------------------------------------------
PROGRAM segment
  assume cs:PROGRAM,ds:VARIABLES
```

```
;---------------------------------------------------------------------------------------
 BIGGER   proc
       mov     al,NUM1
       cmp     al,NUM2
       jg      UpdateBigger
       mov     al,NUM2
UpdateBigger:
       mov RESULT,al
       ret
BIGGER   endp
;---------------------------------------------------------------------------------------
MAIN:
       mov     ax,VARIABLES
       mov     ds,ax
       call    BIGGER
       mov     ax,4c00h
       int     21h
PROGRAM ends
       end MAIN
```

מנגנון ביצוע הפרוצדורה

המעבד (CPU) מבצע תמיד פקודה אחר פקודה. כאשר המעבד פוגש בפקודה CALL (בשפת מכונה כמובן), הוא מבצע את הפעולות הבאות:

1. שומר בתוך המחסנית (PUSH) את הכתובת של הפקודה הבאה לביצוע (זו שבשורה שלאחר פקודת CALL). כתוצאה מדחיפת הנתון למחסנית, ערך SP ירד ב-2.
2. משנה את ערכו של האוגר IP, כך שיצביע על הכתובת של הפרוצדורה, ועל ידי כך למעשה ממשיך לבצע את הפקודות הרשומות בפרוצדורה.

בסיום הפרוצדורה, כאשר המעבד פוגש בפקודה RET, הוא מבצע את הפעולות האלו:

1. שולף מתוך המחסנית (POP) את הכתובת שרשומה (זוהי הכתובת שהוכנסה בשלב הקריאה לפרוצדורה). כתוצאה מכך, ערכו של SP עולה ב-2.
2. הכתובת הנשלפת מהמחסנית מועתקת לאוגר IP, וכך למעשה הוא קופץ להמשך ביצוע התוכנית הראשית (לפקודה הרשומה מייד לאחר שורת ה-CALL).

נראה את הדוגמה הבאה. קוד התוכנית:

```
;---------------------------------------------------------------------------------------
; Name:  proc4.asm
; Task:   Demonstration:
;         The program calls a procedure X
;         Procedure X calls procedure Y
;---------------------------------------------------------------------------------------
CODE segment
  assume cs:CODE, ds:CODE
```

```
X   proc
        mov     ax,3
        call    Y        ; Call procedure Y
        ret
X   endp
Y   proc
        add     ax,ax   ; Multipy ax value
        ret
Y   endp
START_PROGRAM:
        mov     ax,CODE
        mov     ds,ax
        call    X        ; Call procedure X
        mov     ax,4c00h
        int     21h
CODE ends
        end START_PROGRAM
```

כאשר התוכנית נמצאת בזיכרון, היא נראית כך:

```
-----------------------------------------------------------------------------------------------
 Using DEBUG to check the program proc4.asm
-----------------------------------------------------------------------------------------------
-n proc4.exe
-l
-u0
110C:0000 B80300        MOV     AX,0003
110C:0003 E80100        CALL    0007
110C:0006 C3            RET                 ; End of procedure X

110C:0007 03C0          ADD     AX,AX
110C:0009 C3            RET                 ; End of procedure Y

110C:000A B80C11        MOV     AX,110C       ; Begining of the program
110C:000D 8ED8          MOV     DS,AX
110C:000F E8EEFF        CALL    0000          ; Call X
110C:0012 B8004C        MOV     AX,4C00
110C:0015 CD21          INT     21          ; End of this program
-t
AX=110C  BX=0000  CX=0137  DX=0000  SP=0120  BP=0000  SI=0000  DI=0000
DS=10EA  ES=10EA  SS=10FA  CS=110C  IP=000D   NV UP EI PL NZ NA PO NC
110C:000D 8ED8          MOV     DS,AX
-t
AX=110C  BX=0000  CX=0137  DX=0000  SP=0120  BP=0000  SI=0000  DI=0000
DS=110C  ES=10EA  SS=10FA  CS=110C  IP=000F   NV UP EI PL NZ NA PO NC
110C:000F E8EEFF        CALL    0000
-t
-----------------------------------------------------------------------------------------------
 After executing CALL X, SP=SP-2 , and the CPU jumps to address CS:0000
-----------------------------------------------------------------------------------------------
```

```
AX=110C  BX=0000  CX=0137  DX=0000  SP=011E  BP=0000  SI=0000  DI=0000
DS=110C  ES=10EA  SS=10FA  CS=110C  IP=0000   NV UP EI PL NZ NA PO NC
110C:0000 B80300       MOV    AX,0003
```

We can see the address which was pushed to the stack: 0012H, which is the address of the instruction MOV AX,4C00 in the main program.

```
-d ss:011e l 2
10FA:0110                        12 00            ..
-t
AX=0003  BX=0000  CX=0137  DX=0000  SP=011E  BP=0000  SI=0000  DI=0000
DS=110C  ES=10EA  SS=10FA  CS=110C  IP=0003   NV UP EI PL NZ NA PO NC
110C:0003 E80100       CALL    0007
-t
```

Calling to 0007H, where procedure Y starts. Note that SP=SP-2.

```
AX=0003  BX=0000  CX=0137  DX=0000  SP=011C  BP=0000  SI=0000  DI=0000
DS=110C  ES=10EA  SS=10FA  CS=110C  IP=0007   NV UP EI PL NZ NA PO NC
110C:0007 03C0        ADD    AX,AX
```

The address which was pushed to the stack is 0006H, which is the address of the instruction RET, right after CALL 0007

```
-d ss:011c l 2
10FA:0110                     06 00            ..
-t
AX=0006  BX=0000  CX=0137  DX=0000  SP=011C  BP=0000  SI=0000  DI=0000
DS=110C  ES=10EA  SS=10FA  CS=110C  IP=0009   NV UP EI PL NZ NA PE NC
110C:0009 C3          RET
-t
```

Return from Y procedure: pop the address (0006) which is pointed by SP, and jumps to that address (CS:0006)

```
AX=0006  BX=0000  CX=0137  DX=0000  SP=011E  BP=0000  SI=0000  DI=0000
DS=110C  ES=10EA  SS=10FA  CS=110C  IP=0006   NV UP EI PL NZ NA PE NC
110C:0006 C3          RET
-t
```

Return from X procedure: pop the address (0012) which is pointed by SP, and jumps to that address (CS:0012)

```
AX=0006  BX=0000  CX=0137  DX=0000  SP=0120  BP=0000  SI=0000  DI=0000
DS=110C  ES=10EA  SS=10FA  CS=110C  IP=0012   NV UP EI PL NZ NA PE NC
110C:0012 B8004C       MOV    AX,4C00
-q
```

העברת נתונים אל הפרוצדורה ומהפרוצדורה

יש מקרים שבהם אין צורך להעביר נתונים כלשהם אל הפרוצדורה. למשל, כאשר קוראים לפרוצדורה שמציגה הודעה קבועה על המסך. אך, כאשר אנו מעוניינים לבנות פרוצדורה שתציג לנו בכל פעם הודעה אחרת, הרי שעלינו לבנות פרוצדורה שמקבלת כקלט (INPUT) את הכתובת שבה נמצאת ההודעה שאותה אנו מעוניינים להציג.

העברת נתונים באמצעות אוגרים

נראה לדוגמה את התוכנית הבאה, המציגה הודעות שונות בכל פעם על ידי העברת כתובת ההודעה שרוצים להציג באמצעות האוגר DX.

```
;------------------------------------------------------------------------------------------------
; Name:  proc6.asm
; Task:  The program transfers data to procedure using the DX register
;------------------------------------------------------------------------------------------------
Data segment
   Message1 db 'Good morning...',10,13,'$'
   Message2 db 'See you later...',10,13,'$'
   Message3 db 'Bye...',10,13,'$'
Data ends
Code segment
   assume cs:Code,ds:Data
Main: mov ax,Data
        mov ds,ax
        mov  dx,offset Message1
        call PrintMessage   ; Call the procedure for Message1
        mov  dx,offset Message2
        call PrintMessage   ; Call the procedure for Message2
        mov  dx,offset Message3
        call PrintMessage   ; Call the procedure for Message3
        mov ax,4c00h
        int 21h             ; Terminate
;------------------------------------------------------------------------------------------------
; Name:   PrintMessage
; Task:   Displays message
; Input:  DX: points to the string to be displayed
; Output: None
;------------------------------------------------------------------------------------------------
PrintMessage:
        mov ah,9
        int 21h
        ret                 ; Return
Code ends
    end Main
```

העברת נתונים באמצעות משתנים

אפשר להעביר נתונים באמצעות משתנים. התוכנית הבאה מעבירה דרך המשתנה NUMBER את הנתון שרוצים לכפול ב-2, ומקבלת מהפרוצדורה את התוצאה דרך המשתנה RESULT.

```
;-------------------------------------------------------------------------------
; Name:  proc7.asm
; Task:  Transfering data to procedure via variables
;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------
;              GLOBAL VARIABLES
;-------------------------------------------------------------------------------
DATA segment
      NUMBER   db  ?
      RESULT   dw  ?
DATA ends
;-------------------------------------------------------------------------------
;              MAIN  PROGRAM
;-------------------------------------------------------------------------------
PROGRAM segment
  assume cs:PROGRAM, ds:DATA
;-------------------------------------------------------------------------------
; Name:  Multiply
; Task:  Copies to RESULT the value of NUMBER * 2
; Input:  NUMBER  (byte)
; Output: RESULT  (word)
;-------------------------------------------------------------------------------
Multiply  proc
      mov    al,NUMBER
      cbw                    ; Convert byte to word (in AX)
      add    ax,ax           ; Multiply by 2
      mov    RESULT,ax       ; Save the result
      ret                    ; Return
Multiply  endp
;-------------------------------------------------------------------------------
MAIN:
      mov    ax,DATA
      mov    ds,ax
      mov    NUMBER,43h
      call   Multiply     ; Result will be in RESULT
      mov    ax,4c00h
      int    21h
PROGRAM ends
      end MAIN
```

המשתנים המוגדרים בתוכנית, נקראים **גלובליים**, כי הם נגישים הן לתוכנית והן לפרוצדורה.

העברת נתונים באמצעות מחסנית

שיטה נוספת להעברת נתונים בין התוכנית הראשית לפרוצדורה, היא באמצעות המחסנית. למעשה, בשפות עיליות רוב העברות הנתונים אל הפרוצדורות ומהן, נעשית - מבלי שהמתכנת ידאג לכך - דרך המחסנית. למשל, בשפת C, כאשר כותבים:

PRINTF ("%S" , MESSAGE);

גורמים להעברת כתובת המחרוזת MESSAGE אל המחסנית ולקריאה לפונקציה PRINTF לצורך הצגת ההודעה על המסך.

בדוגמה הבאה, נראה כיצד מעבירים אל פרוצדורה נתון דרך המחסנית. הנתון הוא הכתובת של ההודעה שרוצים להציג:

```
;------------------------------------------------------------------------
; Name:  proc8.asm
; Task:  Transfering data to the procedure via the stack
;------------------------------------------------------------------------
Data segment
  Notify   db 'The address of this message has been transfered via the
           stack...',10,13,'$'
Data ends
S segment stack 'stack'
 dw 100h dup (0)
S ends
Code segment
  assume cs:Code,ds:Data, ss:S
Main:  mov ax,Data
       mov ds,ax
       mov  cx,offset Notify    ; Get the address of the string to display
       push cx                  ; Transfer the address via the stack
       call PrintMessage        ; Call the procedure to display message
       mov ax,4c00h
       int 21h                  ; Terminate
;------------------------------------------------------------------------
; Name:   PrintMessage
; Task:   Displays a message
; Input:  Via the stack: Address of the string to display
; Output: None
;------------------------------------------------------------------------
PrintMessage:
       pop bx              ; Save the address to return
       pop dx              ; Get the address of the message
       mov ah,9
       int 21h             ; Print the message in DS:DX
       push bx             ; Push the address of the calling program
       ret                 ; Return (pop the address and then jump)
Code ends
    end Main
```

שים לב, שכאשר בתוך הפרוצדורה המעבד יגיע לפקודה RET, עלינו לדאוג שהנתון שיישלף מהמחסנית הוא כתובת החזרה, להמשך ביצוע התוכנית הראשית.

דוגמה נוספת: יש לכתוב פרוצדורה שמקבלת כקלט שני מספרים בגודל מילה כל אחד, ומחזירה את תוצאת החיבור ביניהם. הקלט והפלט (INPUT ו-OUTPUT) לפרוצדורה נעשים באמצעות המחסנית. התוכנית הראשית תקרא לפרוצדורה, כדי לבצע חיבור של שני המספרים 83H ו- 45H, ולאחר מכן תציב את התוצאה של החיבור באוגר AX:

```
;--------------------------------------------------------------------------------------
; Name:  proc9.asm
; Task:  The program adds two numbers.
;        A demonstration of using the stack for transfering
;        data to procedures
;--------------------------------------------------------------------------------------
StackSeg        segment stack 'stack'
        dw 100h dup (?)
StackSeg        ends
;--------------------------------------------------------------------------------------
;               MAIN  PROGRAM
;--------------------------------------------------------------------------------------
Program segment
  assume cs:Program,ss:StackSeg
;--------------------------------------------------------------------------------------
; Name:   AddTwoNumbers
; Task:   Adds two numbers
; Input:  Via the stack: two word-sized numbers
; Output: Result in AX
;--------------------------------------------------------------------------------------
AddTwoNumbers   proc
        mov     bp,sp           ; Save current stack pointer
        mov     ax,[bp+2]       ; Get one of the two numbers
        add     ax,[bp+4]       ; Get the second one
        ret                     ; Return
AddTwoNumbers   endp
;--------------------------------------------------------------------------------------
;               main  program
;--------------------------------------------------------------------------------------
MAIN:
        mov     ax,Program
        mov     ds,ax
        mov     cx,83h
        push    cx                      ; Transfer first number
        mov     cx,45h
        push    cx                      ; Transfer second number
        call    AddTwoNumbers           ; Call the procedure
        mov     ax,4c00h
        int     21h
Program ends
        end MAIN
```

שים לב לשימוש שנעשה באוגר BP. אוגר זה משמש כאוגר מצביע בתוך מקטע המחסנית (כברירת מחדל).

הערה חשובה שישימה גם לשפות עיליות: כדי לא לגרום להעמסת המחסנית (שגודלה מוגבל), על המתכנת להימנע מהעברת נתונים רבים מדי אל הפרוצדורות באמצעות המחסנית. למשל, במקום להעביר דרך המחסנית מחרוזת תווים שלמה שרוצים להציג, נעביר רק את הכתובת שבה נמצאת ההודעה (גודל הכתובת היא מילה או מילה כפולה).

פרוצדורה קרובה ופרוצדורה רחוקה

פרוצדורה קרובה (NEAR) נמצאת בזיכרון בגבולות הסגמנט של התוכנית.

פרוצדורה רחוקה (FAR), יכולה להיות בכל מקום בזיכרון.

עד כה, כל פרוצדורה שהגדרנו היתה פרוצדורה קרובה.

מתי נצטרך להגדיר פרוצדורה רחוקה

1. כאשר נרצה שהפרוצדורה תוכל להיות נגישה עבור כל תוכנית שנמצאת בזיכרון, גם אם היא אינה נמצאת באותו סגמנט שבה נמצאת הפרוצדורה.
2. כאשר גודל התוכנית הראשית הוא מעל 64K (הגודל המקסימלי של סגמנט).

ניתן להגדיר כל פרוצדורה כרחוקה, גם אם אין בטוחים שיש בכך צורך.

כיצד מגדירים פרוצדורה רחוקה

במקום להשתמש בתווית לסימון תחילת הפרוצדורה, נשתמש בהנחיות (directives) של האסמבלר, בצורה זו:

```
ProcedureName PROC FAR
--
ret
ProcedureName ENDP
```

גם פרוצדורה קרובה אנו יכולים להגדיר בצורה דומה:

```
ProcedureName PROC NEAR
--
ret
ProcedureName ENDP
```

כלומר, ההנחיה NEAR מציינת פרוצדורה קרובה, וההנחיה FAR מציינת פרוצדורה רחוקה. בפרוצדורה קרובה אפשר להשמיט את המילה NEAR, מכיון שהיא ברירת המחדל.

מה ההבדל במנגנון הקריאה והחזרה מהפרוצדורה

כאשר המעבד מבצע את הפקודה CALL, הוא שומר במחסנית את כתובת הפקודה הבאה לביצוע. **בפרוצדורה קרובה**, הוא שומר במחסנית ערך האוגר IP בלבד. אך, **בפרוצדורה רחוקה** הוא שומר את הערכים של IP וגם של CS, שהם כתובת הבסיס של הסגמנט וערך ההיסט.

כאשר המעבד מבצע את הפקודה RET עבור פרוצדורה קרובה, הוא שולף רק נתון אחד (ערך IP), ועבור פרוצדורה רחוקה הוא שולף שני נתונים (IP ו-CS).

תיעוד הפרוצדורה

התיעוד הוא מרכיב חיוני בבנייה ובתחזוקה של קוד. תיעוד הפרוצדורה חשוב לא פחות מתיעוד התוכנית, מכיון שהפרוצדורה משרתת לעיתים תוכניות **שונות** שנכתבות על ידי מתכנתים שונים.

תיעוד הפרוצדורה אמור לתת את המידע הנדרש כדי לעשות שימוש בה, ולכן רצוי לרשום לפני כל פרוצדורה בגוף התוכנית, את הנתונים הבאים:

1. **NAME** - שם הפרוצדורה.
2. **TASK** - מה היא מבצעת.
3. **INPUT** - מה הפרוצדורה מקבלת כקלט.
4. **OUTPUT** - איזה נתונים הפרוצדורה מספקת כפלט.
5. **REMARKS** - הערות חשובות לגבי הפרוצדורה (במידה ויש).
6. **DESTROYS** - איזה אוגרים "נהרסים" (שתוכנם משתנה) בעת הכניסה לפרוצדורה, או בעת הפעולה בה. רצוי שערכי כל האוגרים יישמרו על ידי דחיפה למחסנית (PUSH) בעת כניסה לפרוצדורה, והם יישלפו ממנה (POP) לפני היציאה מהפרוצדורה.

דוגמאות נוספות לפרוצדורות:

```
; Name:  proc10.asm
;---------------------------------------------------------------------------------
;                CONSTANTS
;---------------------------------------------------------------------------------
SIZE_OF_STACK = 100h
END_OF_STRING = 0
;---------------------------------------------------------------------------------
;                GLOBAL VARIABLES
;---------------------------------------------------------------------------------
DATA segment
        String        db  'Testing this program...'
        StringLength  dw  0
DATA ends
```

```
;--------------------------------------------------------------------------------------
;               STACK  SEGMENT
;--------------------------------------------------------------------------------------
S Segment stack 'stack'
      dw SIZE_OF_STACK dup (?)
S ends
;--------------------------------------------------------------------------------------
;               MAIN  PROGRAM
;--------------------------------------------------------------------------------------
MAIN segment
  assume cs:MAIN, ds:DATA, ss: S
START:
      mov     ax,DATA
      mov     ds,ax              ; Initialize
      mov     bx,offset String
      push    bx                 ; Address of String (input for STRLEN)
      call    STRLEN
      pop     StringLength       ; Result (output from STRLEN)
      mov     ax,4c00h
      int     21h                ; Terminate
;--------------------------------------------------------------------------------------
STRLEN   proc
      mov     bp,sp              ; Current stack pointer
      mov     si,[bp+2]          ; Get the input
      mov     ax,0               ; Counter of bytes
CheckMore:
      ; Is it end of string?
      cmp     byte ptr [si], END_OF_STRING
      je      GiveOutput
      inc     si                 ; Point to the next byte
      inc     ax                 ; Increment # of bytes
      jmp     CheckMore          ; Continue
GiveOutput:
      mov     [bp+2],ax          ; Update output
      ret
STRLEN   endp
;--------------------------------------------------------------------------------------
MAIN ends
      end START
```

```
; Name: proc11.asm
Data  segment
  FIRST   dw  1111h        ; First number
  SECOND  dw  2345h        ; Second number
  SUM     dw  ?            ; Result
Data  ends
Sta segment stack 'stack'
   dw 100h dup (?)
Sta ends
Code  segment
  assume cs:Code, ds:Data, ss: Sta
;-----------------------------------------------------------------------------------------------
;                        Procedures
;-----------------------------------------------------------------------------------------------
AddTwoNumbers  proc near
       mov     bp, sp       ; Save SP value before changing it
       ;-------------Save registers befor using----------------
       push    ax
       push    bp
       mov     ax, [bp+4]  ; FIRST value
       add     ax, [bp+2]  ; Add SECOND value
       mov     [bp+4], ax  ; Save the result
       ;----------Restore register original values-------------
       pop     bp
       pop     ax
       ret     2           ; Return and add 2 to SP register
                           ; This is done in order to "clean" the stack
                           ; Now SP will point the result
AddTwoNumbers  endp
;-----------------------------------------------------------------------------------------------
;                        Main program
;-----------------------------------------------------------------------------------------------
Begin:
       mov     ax, Data
       mov     ds, ax
       push    FIRST
       push    SECOND
       call    AddTwoNumbers   ; Add the two pushed numbers
       pop     SUM             ; Pop the result into SUM
       mov     ax, 4c00h
       int     21h             ; Terminate
Code  ends
    end Begin
```

תרגילים

1. בנה פרוצדורה המקבלת שני נתונים בגודל בית כל אחד, דרך המשתנים NUM2, NUM1.

הפרוצדורה תחזיר את הערכים הבאים:

0 אם num1 = num2

1 אם num1 > num2

2 אם num2 > num1

התוכנית תקרא לפרוצדורה זו, כדי לבדוק את הערכים של שני תאי הזיכרון בכתובות 300H ו-400H (היסטים ממקטע הקוד). אם הערכים זהים, תוצג הודעה "equal"; אחרת תוצג הודעה "not equal".

2. בנה פרוצדורה המקבלת דרך המחסנית נתון בגודל מילה, ומחזירה דרך המחסנית ערך השווה למחצית הנתון המתקבל.

3. רשום פרוצדורה שמקבלת שני נתונים בגודל בית דרך המחסנית, ובודקת האם כמות הסיביות בערך "1" הינו זהה ב-2 הנתונים. אם הכמות זהה, יוחזר דרך המחסנית הערך 0. אם הכמות שונה, יוחזר הנתון שלו כמות הסיביות הגדול יותר.

4. תכנן פרוצדורה שמקבלת דרך שלושה משתנים שלושה נתונים:

- כתובת ההתחלה של מערך הכולל נתונים בגודל בית כל אחד.
- כתובת הסיום של המערך.
- ערך X בגודל בית.

הפרוצדורה תחזיר דרך המחסנית את כמות הפעמים שנמצא הערך X בתוך המערך.

5. בנה פרוצדורה בשם scanf המקבלת דרך המחסנית כתובת של מערך. הפרוצדורה תקלוט מלוח-המקשים מחרוזת תווים, עד להקשה על Enter, ותציב מחרוזת זו בכתובת הקלט.

[illegible]

[illegible]

[illegible]

14

קבועים (Constants)

הקבועים ותפקידם

בדומה לשפות עיליות, לקבועים יש שימוש במקרים אלה:

1. כאשר ערך קבוע חוזר על עצמו מספר פעמים במהלך התוכנית, ובמקום לרשום מספר פעמים את אותו הערך, רושמים את הקבוע.
2. כאשר הערך יכול להשתנות בעתיד, ולכן במקום לבצע שינויים בתוך התוכנית בכל אותם מקומות רלוונטיים, נוכל לבצע את השינוי במקום אחד בלבד.
3. לצורך תיעוד טוב יותר. למשל, השם Value ברור יותר מאשר "סתם" 4.
4. כאשר משתמשים בקבועים שכבר הוגדרו על ידי תוכניות אחרות (למשל כאשר משתמשים בתכנות בפרוצדורות של WINDOWS API).

נהוג להגדיר את הקבועים בראש התוכנית, אך למעשה ניתן להגדירם בכל מקום.

קיימים שני סוגי קבועים. האחד מוגדר על ידי הסימן "שווה" (=) והשני - על ידי ההנחיה EQU (שווה, EQUal). לדוגמה:

```
; const1.asm
NumberOfUnits = 44          ; Number of units
UnitCost = 30             ; Cost of one unit
Code    segment
   assume cs:Code,ds:Code
OK:
        mov     ax,Code
        mov     ds,ax
        ;---------------------Calculate  AX = NumberOfUnits * UnitCost-------------------
        mov     cx, NumberOfUnits
        mov     ax, 0
AddMore:
        add     ax, UnitCost
        dec     cx
        jnz     AddMore
Code    ends
        end     OK
```

שים לב להבדלים בין = לבין EQU:

1. בעזרת EQU בלבד, ניתן להגדיר לא רק ערכים מספריים, אלא גם מחרוזות:

```
X  EQU 'This is an example'
```

2. בעזרת = בלבד ניתן לבצע הגדרה מחדש של הקבוע. למשל:

```
R  = 4
R = R + 1
R = 25
```

דוגמאות נוספות לקבועים:

```
; Name:   const2.asm
; Task:   Demonstration of constants
;---------------------------------------------------------------------------------------
;                       CONSTANTS (apply to tasm)
;---------------------------------------------------------------------------------------
NEW_LINE       equ  10, 13, '$'   ; For new line. Compile warning will appear in
                                   ; masm 4.0 and masm 5.10
INT21          equ  int 21h       ; int 21h. Compile error will appear in
                                   ; masm 4.0 and masm 5.10
ENTER_CODE     =    13            ; Enter code
;---------------------------------------------------------------------------------------
;                      MAIN   PROGRAM
;---------------------------------------------------------------------------------------
Program  segment
   assume cs:Program, ds:Program
Begin:
      mov     ax,Program
      mov     ds,ax                  ; Initialize
      mov     dx,offset FirstMessage
      mov     ah,9
      INT21                          ; Display first message
GetMoreKeys:
      mov     ah,1
      INT21                          ; Read one Key
      cmp     al,ENTER_CODE          ; Is it Enter?
      jne     GetMoreKeys            ; If not, continue
      mov     ax,4c00h               ; If Enter -
      INT21                          ; Terminate
;------------------------------- Message to display ------------------------------------
FirstMessage db 'Ready to go...', NEW_LINE
Program    ends
      end    Begin
```

```
; const3.asm
NUMBER = 3
_main    segment
   assume cs:_main,ds:_main
main1:
        mov     ax,_main
        mov     ds,ax
        mov     cx,3                ; Counter
        mov     al, NUMBER          ; Here NUMBER = 3
More:
NUMBER = NUMBER + 1
        add     al, NUMBER          ; Here NUMBER = 4  Always!!!
        loop    More                ; Do it 3 times
_main    ends
        end     main1
```

נראה כיצד נראית תוכנית const3.asm לאחר האסמבלר:

```
-n const3.exe
-l
-u0 0e
10FA:0000 B8FA10        MOV     AX,10FA
10FA:0003 8ED8          MOV     DS,AX
10FA:0005 B90300        MOV     CX,0003
10FA:0008 B003          MOV     AL,03
10FA:000A 0404          ADD     AL,04
10FA:000C E2FC          LOOP    000A
10FA:000E 216344        AND     [BP+DI+44],SP
-q
```

שים לב, שערכו של NUMBER בתוך הלולאה יהיה תמיד 4, מכיון שבזמן התרגום משפת אסמבלי לשפת מכונה, הקבוע NUMBER מתורגם ל-4. הלולאה מתבצעת רק בזמן הרצת התוכנית, שבה הקבוע הסמלי NUMBER אינו קיים כלל ובמקומו יש 4.

15

אסמבלי מותנה (Conditional Assembly)

אסמבלי מותנה הוא קטע קוד בשפת אסמבלי, שבאמצעות תנאי מסוים שכלול בו, אפשר להחליט אם קטע הקוד הזה יעבור אסמבלר ויהפוך לשפת-מכונה, או לאו.

נציג דוגמה:

```
;------------------------------------------------------------------------
; Name: Mutne1.asm
;------------------------------------------------------------------------
Moshe = 0              ; Constant
Code    segment
Start:
        mov     ax,Code
        mov     ds,ax
IF Moshe               ; If constant Moshe isn't 0
        mov     ax,5
ELSE                   ; Else (if constant Moshe is 0)
        mov     ax,6
ENDIF
        add     ax,2
        add     ax,ax
        int     3               ; Stop
Code    ends
end     Start
```

בדוגמה זו אנו משתמשים בהנחיה IF. שים לב, שזו אינה פקודת אסמבלי (instruction), אלא הנחיה (directive) לאסמבלר, המנחה אותו כיצד להתייחס לקטע התוכנית. ההבדל בין הנחיה לבין פקודה הוא בכך, שהפקודות מתורגמות לשפת מכונה על ידי תוכנית האסמבלר, ואילו ההנחיות המיועדות לאסמבלר עצמו אינן הופכות לפקודות מכונה כלשהן.

ההנחיה IF תגרום להכללת קטע הקוד (התחום בין IF ל-ENDIF) בזמן האסמבלר, אם ערכו של הקבוע שונה מאפס, ולהיפך: אם ערכו של הקבוע הינו אפס, אזי האסמבלר יתעלם מקטע הקוד, ולא יהפוך אותו לשפת-מכונה.

בדוגמה זו, כאשר ערכו של הקבוע היה 1, לאחר האסמבלר התוכנית נראית כך:

```
-n mutne1.exe
-l
-u0 0d
10FA:0000 B8FA10      MOV     AX,10FA
10FA:0003 8ED8        MOV     DS,AX
10FA:0005 B80500      MOV     AX,0005
10FA:0008 050200      ADD     AX,0002
10FA:000B 03C0        ADD     AX,AX
10FA:000D CC          INT     3
-q
```

כאשר ערכו של הקבוע הוא 0, התוכנית לאחר האסמבלר נראית כך:

```
-n mutne1.exe
-l
-u0 0d
10FA:0000 B8FA10      MOV     AX,10FA
10FA:0003 8ED8        MOV     DS,AX
10FA:0005 B80600      MOV     AX,0006
10FA:0008 050200      ADD     AX,0002
10FA:000B 03C0        ADD     AX,AX
10FA:000D CC          INT     3
-q
```

מתי משתמשים באסמבלי מותנה

הוספת קטעי קוד לניפוי שגיאות

כשבונים תוכנית, שותלים בה במקומות שונים קטעי קוד שמציגים על המסך ערכים של משתנים, או הודעות שונות לסיוע בעת הניפוי. כלומר, לצורך מעקב אחר ביצוע הפעולות השונות. עושים זאת מתוך כוונה לסייע בזיהוי או בפתרון שגיאות.

הלקוח הסופי של התוכנית אינו צריך לקבל את קטעי הקוד האלה, אך אנו רוצים להשאיר אותם עבור בעיות, או בדיקות עתידיות.

הפתרון: קטעי הקוד הללו אכן יהיו ויישארו בתוך קוד המקור, אולם כאשר התוכנית תימסר ללקוח, נדאג שקטעי קוד אלה לא יעברו את תהליך האסמבלר ולא ייכנסו לתוכנית המיועדת לביצוע.

התאמות ייחודיות למשתמש

לעיתים, לקוחות שונים דורשים התאמות ייחודיות עבורם. אם יש שינויים מהותיים בין הדרישות של משתמשים שונים, נאלץ לעיתים לכתוב חלקי תוכניות שונים עבור כל לקוח, ואף תוכנית מלאה המותאמת לכל אחד מהם. יחד עם זאת, הגישה הנכונה היא לנסות עד כמה שאפשר ליצור תוכנית אחת, שנותנת מענה לכל הלקוחות. תחזוקה של תוכנית אחת, ולא של מספר תוכניות, טובה וזולה יותר.

כאן בא לעזרתנו האסמבלי המותנה: התוכנית תכלול את **כל** הדרישות הרלוונטיות לכל לקוח, אבל כאשר נבצע אסמבלר ללקוח א׳, נפעיל בעזרת מנגנון האסמבלי המותנה את תוכנית המקור עבור הקטעים הרלוונטיים ללקוח א׳ בלבד. בהמשך נציג מספר דוגמאות למקרים אלה.

הנחיות נוספות של אסמבלי מותנה

- IFDEF (IF DEFINED): קטע הקוד יעבור אסמבלר, אם הוגדרה תווית.

צורת הרישום:

```
IFDEF תווית
ENDIF
```

דוגמאות:

```
;-----------------------------------------------------------------------------------------
; Name:    mutne6.asm
;
; Task:    Caculates  Sum = 1 + 2 + ... + Value
;          For example, if Value=5, then Sum = 1+2+3+4+5
;
; Remark:  Define DEBUG for debug-version of this program
;          You may define DEBUG while executing Tasm/Masm
;-----------------------------------------------------------------------------------------
Data    segment
    Value    db  5      ; Value to calculate
    Sum      dw  0      ; Sum of addition
;--------------------------------For Debug tests--------------------------------------------
    NL          db  10 , 13, '$' ; New Line
    ShiftRight  db  ?            ; # of bits to shift right
    NumOfDigits db  ?            ; # of digits to display
;-----------------------------------------------------------------------------------------
Data    ends
S       segment stack 'stack'
    dw 100b dup (?)
S ends
Code    segment
   assume cs:Code, ds:Data, ss:S
Start:
     mov    ax,Data
```

```
        mov     ds,ax
        mov     Sum,0           ; Initialize Sum
        cmp     Value,0         ; Check if Value = 0
        je      ReadyToGo       ; No need to continue
        mov     al,0            ; Value to add
MoreToAdd:
        inc     al
        cbw                     ; AX <-- AL  (convert Byte to Word)
        add     Sum,ax          ; Add to Sum (add word to word)
;------------------------------For Debug tests---------------------------------------------
IFDEF DEBUG
        call    PrintSumValue   ; Print Sum value
ENDIF
;-----------------------------------------------------------------------------------------
        dec     Value
        jnz     MoreToAdd       ; Continue for all
ReadyToGo:
        mov     ax,4c00h
        int     21h             ; terminate program
;------------------------------For Debug tests---------------------------------------------
IFDEF DEBUG
;-----------------------------------------------------------------------------------------
; Name:    PrintSumValue
; Task:    Prints the global word-sized variable Sum
; Input:   Sum is always used as input
; Output:  None
; Remarks: Procedure PrintOneDigit is used
;-----------------------------------------------------------------------------------------
PrintSumValue  proc near
        push    ax
        push    bx
        push    cx
        push    dx
        mov     NumOfDigits,4   ; 4 Digits to display
        mov     ShiftRight, 16  ; # of bits to shift right
MoreToDisplay:
        mov     bx,Sum          ; The value to print
        sub     ShiftRight, 4   ; bits to shift right in order
                                ; to get the left digit
        mov     cl,ShiftRight
        shr     bx,cl           ; Shift right
        and     bx,000fh        ; Mask to get only the right hex-digit
        call    PrintOneDigit   ; Print that digit
        dec     NumOfDigits
        jnz     MoreToDisplay   ; Continue for all 4 digits
        mov     dx,offset NL
        mov     ah,9
        int     21h             ; Print New-Line
```

```
        pop     dx
        pop     bx
        pop     cx
        pop     ax
        ret
PrintSumValue  endp
;------------------------------------------------------------------------------------------
; Name:   PrintOneDigit
; Task:   Prints only one digit
; Input:  BL as the digit
; Output: None
;------------------------------------------------------------------------------------------
PrintOneDigit  proc
        push    ax
        push    bx
        push    cx
        push    dx
        mov     dl,bl           ; Copy the digit
        add     dl,30h          ; Turn to ascii code
        cmp     dl,39h          ; For digit greater then 9 (a,b,c,d,e,f)
        jbe     PrintOK         ; we have to add more 7 in order to
        add     dl,7            ; turn into ascii
PrintOK:
        mov     ah,2
        int     21h             ; Print one character
        pop     dx
        pop     bx
        pop     cx
        pop     ax
        ret
PrintOneDigit  endp
ENDIF
;------------------------------------------------------------------------------------------
Code    ends
end     Start
```

```
; Name: mutne5.asm : One source for Popye, Olive and Bluto

Popye EQU 'yes'             ; Popye is defined now
; Olive EQU 'yes'           ; <--- Will be choosed for Olive
; Bluto EQU 'yes'           ; <--- Will be choosed for Olive
Code    segment
Start:
        mov     ax,Code
        mov     ds,ax
        mov     dx,offset Message
```

```
        mov     ah,9
        int     21h         ; Display Message
        mov     ax,4c00h
        int     21h             ; terminate program
IFDEF Popye                 ; If Popye is defined (it is defined now)
        Message db 'Popye is the best!', 10, 13, '$'
ENDIF
IFDEF Olive                 ; If Olive is defined
        Message db 'Olive is lovely!', 10, 13, '$'
ENDIF
IFDEF Bluto                 ; If Bluto is defined
        Message db 'Bluto likes Olive', 10, 13, '$'
ENDIF
Code    ends
end     Start
```

- IFDIF (IF DIFFERENT): הקטע יעבור אסמבלר, אם יש הבדל בין שני הארגומנטים:

```
IFDIF <ארגומנט 1> <ארגומנט 2>
ENDIF
```

- IFE (IF EQUAL): אם הביטוי שווה לאפס, קטע הקוד יעבור אסמבלי:

```
IFE ביטוי
ENDIF
```

- IFIDN (IF IDENTICAL): אם שני הארגומנטים זהים, קטע הקוד יעבור אסמבלי:

```
IFUDN <ארגומנט 1> <ארגומנט 2>
ENDIF
```

- IFNDEF (IF NOT DEFINED): אם לא הוגדרה תווית, קטע הקוד יעבור אסמבלי.

```
IFNDEF תווית
ENDIF
```

- IF1 (IF PASS 1): כדי לבצע את התרגום משפת אסמבלי לשפת מכונה, האסמבלר מבצע באופן רגיל שני שלבים (או שני מעברים). במעבר הראשון הוא "עובר" על התוכנית ובוחן את הקבועים, המשתנים והתוויות שהוגדרו, ויוצר "טבלת סמלים" המגדירה לכל סמל את שמו ואת הכתובת שלו. במעבר השני, האסמבלר שותל בכל מקום שבו מופיע סמל, את הכתובת שלו.

ב-IF1, קטע הקוד שבתוכו יבוצע רק במעבר הראשון, ואילו ב-IF2 יבוצע קטע הקוד רק במעבר השני של האסמבלר. בדרך כלל, משתמשים ב-IF1 וב-IF2 לצורך הודעות המופיעות במהלך האסמבלי.

- IFB (IF BLANK): קטע הקוד יעבור אסמבלר אם הסמל הוא ריק מתוכן (BLANK).

IFNB (IF NOT BLANK): קטע הקוד יעבור אסמבלר אם ישנו ערך מסוים בסמל.

16

מאקרו (Macro)

מהו מאקרו?

המאקרו הוא הנחיה עבור האסמבלר, המאפשרת לו לבנות קטע קוד בשם מסוים. לאחר מכן, ניתן לרשום בתוכנית רק את השם הזה וכתוצאה מכך - האסמבלר יחליף את השם, בקטע הקוד. למשל, נכתוב את המאקרו הבא:

```
MACRO GOOD
MOV 0, AX
MOV 0, BX
ENDM
```

שם המאקרו הוא GOOD, והוא מסתיים בהנחיה ENDM (END Macro). כעת, בכל מקום בתוכנית שנכתוב GOOD, כאילו כתבנו את שתי הפקודות:

```
MOV  AX,0
MOV  BX,0
```

מדוע יש בכך צורך? הסיבה העיקרית היא נוחות התכנות. נראה את הדוגמה הבאה (Macro1.asm), שבה הגדרנו מאקרו שבעזרתו אנו דוחפים אוגרים לתוך המחסנית, ומאקרו נוסף שבעזרתו אנו שולפים את האוגרים מהמחסנית.

```
;------------------------------------------------------------------------------------------
; Macro which pushes some registers
;------------------------------------------------------------------------------------------
PushRegisters  macro
        push   ax
        push   bx
        push   cx
        push   dx
        push   ds
        push   es
        push   di
        push   si
endm
```

```
;------------------------------------------------------------------------------------------------
; Macro which pops the previously pushed registers
;------------------------------------------------------------------------------------------------
PopRegisters  macro
        pop     si
        pop     di
        pop     es
        pop     ds
        pop     dx
        pop     cx
        pop     bx
        pop     ax
endm
Program segment
  assume cs:Program, ds:Program
Begin:
        mov     ax,Program
        mov     ds,ax
        PushRegisters           ; Push registers
        inc     ax
        PopRegisters            ; Pop registers
        mov     ax,4c00h
        int     21h
Program ends
        end     Begin
```

לאחר האסמבלר, התוכנית תיראה כך:

```
-debug
   ^ Error
-n macro1.exe
-l
-u0 19
10AC:0000 B8AC10    MOV     AX,10AC
10AC:0003 8ED8      MOV     DS,AX
10AC:0005 50        PUSH    AX  ; First macro
10AC:0006 53        PUSH    BX
10AC:0007 51        PUSH    CX
10AC:0008 52        PUSH    DX
10AC:0009 1E        PUSH    DS
10AC:000A 06        PUSH    ES
10AC:000B 57        PUSH    DI
10AC:000C 56        PUSH    SI
10AC:000D 40        INC     AX
10AC:000E 5E        POP     SI  ; Second macro
10AC:000F 5F        POP     DI
10AC:0010 07        POP     ES
10AC:0011 1F        POP     DS
10AC:0012 5A        POP     DX
10AC:0013 59        POP     CX
```

```
10AC:0014 5B        POP     BX
10AC:0015 58        POP     AX
10AC:0016 B8004C    MOV     AX,4C00
10AC:0019 CD21      INT     21
-q
```

מהו ההבדל העיקרי בין פרוצדורה לבין מאקרו?

- פרוצדורה מתבצעת בזמן הרצת התוכנית.
- מאקרו אינו מתבצע. הוא הופך בזמן האסמבלר, לפקודות מתאימות.

מאקרו עם ארגומנט

נראה כעת דוגמה נוספת ובה נפעיל את המאקרו עם ארגומנט:

```
; macro3.asm

;----------------------------------Macro with 2 arguments----------------------------------
AddValue  macro    A, B
  add A, B
endm
;----------------------------------Program--------------------------------------------------
Code    segment
  assume cs:Code, ds:Code
Start:
        mov     ax,Code
        mov     ds,ax
        mov     ax,5
        mov     bx,2
        AddValue        ax,4            ; add 4 to AX
        AddValue        bx,2            ; add 2 to BX
        nop                             ; No OPeration - do nothing
        nop                             ; Just to mark the end of the program
Code    ends
        end     Start
```

לאחר האסמבלר, התוכנית תיראה כך:

```
-n macro3.exe
-l
-u0 12
10FA:0000 B8FA10      MOV    AX,10FA
10FA:0003 8ED8        MOV    DS,AX
10FA:0005 B80500      MOV    AX,0005
10FA:0008 BB0200      MOV    BX,0002
10FA:000B 050400      ADD    AX,0004      ; Was: AdValue AX,4
10FA:000E 83C302      ADD    BX,0002      ; Was: AdValue BX,2
10FA:0011 90          NOP
10FA:0012 90          NOP
-q
```

שילוב מאקרו עם אסמבלי מותנה

שילוב המאקרו עם אסמבלי מותנה, פותח אפשרויות שימוש נוספות במאקרו.
לדוגמה:

```
; macro5.asm

;-----------------------------------------------------------------------------------------
; Name:   StackHandle
; Type:   Macro
; Task:   Push or Pop AX,BX,CX,DX to/from the stack
;
; Usage:  StackHandle  A
;          * If A is 1, registers will be pushed
;          * If A is 2, registers will be poped
;-----------------------------------------------------------------------------------------
StackHandle   macro   A
  ifidn <A>,<1>    ; If A and 1 are identical
    push  ax
    push  bx
    push  cx
    push  dx
  endif
  ifidn <A>,<2>    ; If A and 2 are identical
    pop  dx
    pop  cx
    pop  bx
    pop  ax
  endif
endm
;-----------------------------------------------------------------------------------------
;            MAIN    PROGRAM
;-----------------------------------------------------------------------------------------
Code    segment
 assume cs:Code, ds:Code
Start:
        mov     ax,Code
        mov     ds,ax
        StackHandle   1           ; Push
        nop                       ; No OPeration, just to mark
        StackHandle   2           ; Pop
        mov     ax,4c00h
        int    21h
Code    ends
        end     Start
```

לאחר האסמבלר נקבל:

```
-n macro5.exe
-l
-u0 11
10FA:0000 B8FA10     MOV    AX,10FA
10FA:0003 8ED8       MOV    DS,AX
10FA:0005 50         PUSH   AX       ; The macro when A = 1
10FA:0006 53         PUSH   BX
10FA:0007 51         PUSH   CX
10FA:0008 52         PUSH   DX
10FA:0009 90         NOP
10FA:000A 5A         POP    DX       ; The macro when A = 2
10FA:000B 59         POP    CX
10FA:000C 5B         POP    BX
10FA:000D 58         POP    AX
10FA:000E B8004C     MOV    AX,4C00
10FA:0011 CD21       INT    21
-q
```

מאקרו לשכפול קוד REPT

REPT משכפל את הקוד מספר פעמים. לדוגמה:

```
; macro7.asm
;------------------------------------------------------------------------------------------
;           MAIN    PROGRAM
;------------------------------------------------------------------------------------------
Code    segment
  assume cs:Code, ds:Code
Num   dw  0                  ; Num variable
Start:
        mov     ax,Code
        mov     ds,ax
        mov     Num, 3
    rept    7
        inc     Num             ; Dublicates this line 7 times
    endm
        mov     ax,4c00h
        int     21h
Code    ends
        end     Start
```

לאחר האסמבלר, התוכנית תיראה כך:

```
-n macro7.exe
-l
-u0 2c
10FA:0000 0000          ADD   [BX+SI],AL              ; This is Num
10FA:0002 B8FA10        MOV   AX,10FA
10FA:0005 8ED8          MOV   DS,AX
10FA:0007 C70600000300  MOV   WORD PTR [0000],0003   ; Mov Num,3
10FA:000D FF060000      INC   WORD PTR [0000]         ; (Dublicated 7 times)
10FA:0011 FF060000      INC   WORD PTR [0000]
10FA:0015 FF060000      INC   WORD PTR [0000]
10FA:0019 FF060000      INC   WORD PTR [0000]
10FA:001D FF060000      INC   WORD PTR [0000]
10FA:0021 FF060000      INC   WORD PTR [0000]
10FA:0025 FF060000      INC   WORD PTR [0000]
10FA:0029 B8004C        MOV   AX,4C00
10FA:002C CD21          INT   21
-q
```

כפי שניתן לראות, מאקרו זה נכתב בתוך מקטע התוכנית. אפשר גם לכתוב מאקרו לשכפול קוד בתוך מקטע נתונים, כפי שנראה בדוגמה זו:

```
; macro9.asm

Data   segment
  X = 1
  rept  8
  dw  X
  X = X + 1
  endm
Data   ends
Code   segment
  assume cs:Code,ds:Data
Main:
      mov    ax,Data
      mov    ds,ax
      mov    ax,4c00h
      int    21h
Code   ends
      end  Main
```

לאחר האסמבלר נראה במקטע הנתונים:

```
10AC:0000  01 00 02 00 03 00 04 00-05 00 06 00 07 00 08 00
```

התוכנית macro9.asm עשתה שימוש במאקרו לשכפול קוד ובקבועים - לצורך נוחות. במקום לכתוב:

```
DW    1,2,3,4,5,6,7,8
```

קיבלנו את אותה תוצאה באמצעות המאקרו.

מאקרו לשכפול קוד על פי רשימה - IRP

מאקרו זה משכפל את הקוד הכתוב בתוכו, ובכל פעם הוא מציב בתוך הקוד נתון אחר מהרשימה.

הפורמט של הנחיה זו היא:

```
IRP פרמטר , < 1 נתון , 2 נתון ,...>
-
-
ENDM
```

נראה דוגמה:

```
; macro11.asm

Main    segment
  assume cs:Main,ds:Main
ProgramStart:
        mov     ax,Main
        mov     ds,ax
        mov     ax,0
    irp  A, <1, 2, 3, 4, 5, 6>  ; Repeate for every argument in the list
        add     ax,A
    endm
        nop
        nop
        nop             ; No OPeration, does nothing
Main    ends
        end     ProgramStart
```

לאחר האסמבלר, נראה:

```
-n macro11.exe
-l
-u 0 1c
10FA:0000 B8FA10        MOV     AX,10FA
10FA:0003 8ED8          MOV     DS,AX
10FA:0005 B80000        MOV     AX,0000
10FA:0008 050100        ADD     AX,0001
10FA:000B 050200        ADD     AX,0002
10FA:000E 050300        ADD     AX,0003
10FA:0011 050400        ADD     AX,0004
10FA:0014 050500        ADD     AX,0005
10FA:0017 050600        ADD     AX,0006
10FA:001A 90            NOP
10FA:001B 90            NOP
10FA:001C 90            NOP
-q
```

מאקרו לשכפול קוד על פי רשימה - IRPC

כאן מבוצע שכפול עבור כל תו שברשימה. הנה דוגמה:

```
; macro13.asm

Code    segment
  assume cs:Code,ds:Code
Start:
        mov     ax,Code
        mov     ds,ax
        mov     bx,0
  irpc    N, 6789
        add     bx, N
  endm
        mov   ax,4c00h
        int     21h
Code    ends
        end  Start
```

לאחר האסמבלר נקבל:

```
-n macro13.exe
-l
-u 0 17
10FA:0000 B8FA10      MOV     AX,10FA
10FA:0003 8ED8        MOV     DS,AX
10FA:0005 BB0000      MOV     BX,0000
10FA:0008 83C306      ADD     BX,+06
10FA:000B 83C307      ADD     BX,+07
10FA:000E 83C308      ADD     BX,+08
10FA:0011 83C309      ADD     BX,+09
10FA:0014 B8004C      MOV     AX,4C00
10FA:0017 CD21        INT     21
-q
```

תרגילים

1. בנה מאקרו בשם ARRAY, שכאשר נרשום אותו בתוך מקטע הנתונים, הוא יביא לתוצאה דומה כאילו כתבנו:

```
A DW  5
B DW ,2 ,4 ,6 ,8 ,10 ,12 ,14 ,16
C DW -1
```

2. בנה מאקרו בשם INSERT, המקבל ארגומנט יחיד. למשל: INSERT 55. המאקרו יגרום להצבת הארגומנט באוגר AX, רק אם הוא אינו מכיל ערך כזה.

3. כתוב מאקרו בשם PRINT, המקבל ארגומנט יחיד שהינו מחרוזת תווים (למשל: THIS IS AN EXAMPLE). המאקרו יגרום להדפסת מחרוזת זו, תוך שימוש בפסיקה להצגת תו בודד.

4. מה מבצעת התוכנית הבאה?

```
; macro20.asm

Great  macro  RegistersToHandle
   irp A, <RegistersToHandle>
      mov    A,0
   endm
endm
Code    segment
  assume cs:Code,ds:Code
Start:  mov    ax,Code
      mov    ds,ax
      Great   <AX, BX, CX, DX>
      mov    ax,4c00h
      int    21h
Code    ends
      end    Start
```

17

פקודות מחרוזת

פקודות מחרוזת מאפשרות טיפול נוח ומהיר במחרוזות בזיכרון. במינוח "מחרוזת" (string) הכוונה לאזור רציף של הבתים בזיכרון. כל פקודות המחרוזת מתייחסות לאזור אחד, או לשני אזורים, בזיכרון:

1. אזור הנקרא מחרוזת המקור או בלוק המקור (source), המתחיל בכתובת DS:[SI].
2. אזור הנקרא מחרוזת (או בלוק) היעד (destination), המתחיל בכתובת ES:[DI].

תזכורת: הרישום ES:[DI] מציין שכתובת הבסיס של המקטע היא ES, וההיסט (המרחק מתחילת המקטע) הוא DI.

נציג את פקודות המחרוזת בליווי דוגמאות.

הפקודה LODSB (LOaD String Byte)

פקודה זו גורמת להעתקת ערך תא הזיכרון בכתובת DS:[SI] (מחרוזת המקור), אל אוגר AL (קבוע, בלתי ניתן לשינוי), ולאחר מכן ערכו של אוגר SI מועלה ב-1 (פרט למקרים שנראה בהמשך).

את הפקודה רושמים כך: LODSB. למעשה, פקודה זו מחליפה את שתי הפקודות האלו:

```
MOV AL,DS:[SI]
INC SI
```

דוגמה לשימוש בפקודה:

```
; string1.asm

Code    segment
  assume cs:Code,ds:Code
Start:
        mov     ax,Code
        mov     ds,ax
        mov     si, 0
        lodsb                ; al = ds:[0]
        nop                  ; No operation
Code    ends
        end     Start
```

לאחר הרצת התוכנית נקבל :

```
-n string1.exe
-l
-t
AX=10FA  BX=0000  CX=000A  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10EA  ES=10EA  SS=10FA  CS=10FA  IP=0003   NV UP EI PL NZ NA PO NC
10FA:0003 8ED8        MOV     DS,AX
-t
AX=10FA  BX=0000  CX=000A  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10FA  ES=10EA  SS=10FA  CS=10FA  IP=0005   NV UP EI PL NZ NA PO NC
10FA:0005 BE0000      MOV     SI,0000
-t
//////////////////////////////////////////////////////////////////////////////////////////
 The value of the byte at address 0 is B8h
//////////////////////////////////////////////////////////////////////////////////////////
-d 0 0
10FA:0000  B8
AX=10FA  BX=0000  CX=000A  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10FA  ES=10EA  SS=10FA  CS=10FA  IP=0008   NV UP EI PL NZ NA PO NC
10FA:0008 AC          LODSB
-t
//////////////////////////////////////////////////////////////////////////////////////////
  B8h is copied to AL
  SI is increased to 1
//////////////////////////////////////////////////////////////////////////////////////////
AX=10B8  BX=0000  CX=000A  DX=0000  SP=0000  BP=0000  SI=0001  DI=0000
DS=10FA  ES=10EA  SS=10FA  CS=10FA  IP=0009   NV UP EI PL NZ NA PO NC
10FA:0009 90          NOP
-q
```

הפקודה STOSB (STOre String Byte)

הפקודה מעתיקה את ערכו של האוגר AL, אל תא הזיכרון במחרוזת היעד ES:[DI], ולאחר מכן ערכו של DI עולה ב-1 (פרט למקרים שנראה בהמשך).

מבנה הפקודה : STOSB

הפקודה מחליפה את צמד הפקודות האלו :

```
MOV ES:[DI],AL
INC DI
```

הפקודה LODSW (LOaD String Word)

הפקודה זהה ל-LODSB, אלא שהיא מעתיקה נתון בגודל מילה ממחרוזת המקור, אל אוגר AX, ולאחר מכן מעלה את ערכו של האוגר SI ב-2 (פרט למקרים שנראה בהמשך).

הפקודה מחליפה את הפקודות האלו:

```
MOV AX,DS:[SI]
ADD SI,2
```

הפקודה STOSW (STOre String Word)

הפקודה זהה ל-STOSB, אלא שהיא מטפלת במילה שלמה ולא בתו בודד.

הפקודה מחליפה את צמד הפקודות:

```
MOV ES:[DI],AX
ADD DI,2
```

דוגמה לשימוש בפקודה:

```
; string2.asm

Data    segment
      Value  dw  0
Data    ends
Code    segment
  assume cs:Code,ds:Data
Start:
        mov     ax,Data
        mov     ds,ax
        mov     es,ax           ; ES = DS (segment of the Value)
        mov     di,offset Value ; now ES:[DI] points to Value
        mov     ax,1234h
        stosw                   ; Value = AX
        nop                     ; No Operation
Code    ends
        end     Start
```

בתוכנית זו השתמשנו בפקודה stosw, כדי להציב במשתנה Value את הערך 1234H.

הפקודה REP (REPeat)

כאשר פקודה זו נרשמת לפני פקודת מחרוזת אחרת, היא גורמת לביצוע כפול של פקודת המחרוזת CX. למשל, אם CX=3, וכתבנו את הפקודה:

```
REP STOSB
```

נגרום לביצוע הפקודה STOSB שלוש פעמים,כביכול כתבנו את כל הפקודות האלו:

```
MOV ES:[DI],AL
INC DI
MOV ES:[DI],AL
INC DI
MOV ES:[DI],AL
INC DI
```

נראה דוגמה לשימוש בפקודה REP SOSB.

התוכנית תגרום להצבת התו "A" בכל תאי הזיכרון בתחום הכתובות 700H עד 720H.

```
; string3.asm

Code    segment
  assume cs:Code,ds:Code
Start:
        mov    ax,Code
        mov    ds,ax
        mov    es,ax        ; ES = CS
        mov    di,700h      ; Start address
        mov    al,'A'       ; The byte to copy to ES:[DI]
        mov    cx,21h       ; Number of bytes (betwween 700-720h)
        rep   stosb         ; Do it now
        nop                 ; No Operation
Code    ends
        end    Start
```

לאחר הרצה נקבל:

```
-n string3.exe
-l
-t27
AX=10FA  BX=0000  CX=0012  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10EA  ES=10EA  SS=10FA  CS=10FA  IP=0003   NV UP EI PL NZ NA PO NC
10FA:0003 8ED8          MOV     DS,AX

AX=10FA  BX=0000  CX=0012  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10FA  ES=10EA  SS=10FA  CS=10FA  IP=0005   NV UP EI PL NZ NA PO NC
10FA:0005 8EC0          MOV     ES,AX

AX=10FA  BX=0000  CX=0012  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10FA  ES=10FA  SS=10FA  CS=10FA  IP=0007   NV UP EI PL NZ NA PO NC
10FA:0007 BF0007        MOV     DI,0700
```

```
AX=10FA  BX=0000  CX=0012  DX=0000  SP=0000  BP=0000  SI=0000  DI=0700
DS=10FA  ES=10FA  SS=10FA  CS=10FA  IP=000A   NV UP EI PL NZ NA PO NC
10FA:000A B041          MOV     AL,41

AX=1041  BX=0000  CX=0012  DX=0000  SP=0000  BP=0000  SI=0000  DI=0700
DS=10FA  ES=10FA  SS=10FA  CS=10FA  IP=000C   NV UP EI PL NZ NA PO NC
10FA:000C B92100        MOV     CX,0021

/////////////////////////////////////////////////////////////////////////////////////////////
 The instruction REP STOSB becomes REPZ, but it is the same
/////////////////////////////////////////////////////////////////////////////////////////////
AX=1041  BX=0000  CX=0021  DX=0000  SP=0000  BP=0000  SI=0000  DI=0700
DS=10FA  ES=10FA  SS=10FA  CS=10FA  IP=000F   NV UP EI PL NZ NA PO NC
10FA:000F F3            REPZ
10FA:0010 AA            STOSB

/////////////////////////////////////////////////////////////////////////////////////////////
 The instruction REP STOSB repeats itself 21h times
/////////////////////////////////////////////////////////////////////////////////////////////
AX=1041  BX=0000  CX=0020  DX=0000  SP=0000  BP=0000  SI=0000  DI=0701
DS=10FA  ES=10FA  SS=10FA  CS=10FA  IP=000F   NV UP EI PL NZ NA PO NC
10FA:000F F3            REPZ
10FA:0010 AA            STOSB
    ---
    ---
    ---
/////////////////////////////////////////////////////////////////////////////////////////////
 This is the last REP STOSB.
 After this instruction CX becomes 0
/////////////////////////////////////////////////////////////////////////////////////////////
AX=1041  BX=0000  CX=0001  DX=0000  SP=0000  BP=0000  SI=0000  DI=0720
DS=10FA  ES=10FA  SS=10FA  CS=10FA  IP=000F   NV UP EI PL NZ NA PO NC
10FA:000F F3            REPZ
10FA:0010 AA            STOSB

AX=1041  BX=0000  CX=0000  DX=0000  SP=0000  BP=0000  SI=0000  DI=0721
DS=10FA  ES=10FA  SS=10FA  CS=10FA  IP=0011   NV UP EI PL NZ NA PO NC
10FA:0011 90            NOP

/////////////////////////////////////////////////////////////////////////////////////////////
 Now all the bytes in addresses 700h - 720h contains 'A'
/////////////////////////////////////////////////////////////////////////////////////////////
-d 700 720
10FA:0700  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
10FA:0710  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41
AAAAAAAAAAAAAAAA
10FA:0720  41                                        A
-q
```

הפקודה CMPSB (CoMPare String Byte)

הפקודה משווה בין ערכו של בית מבלוק המקור,לבין ערכו של בית-היעד. ההשוואה מעדכנת את אוגר הדגלים לאחר סיום ההשוואה, ערכם של SI ו-DI מקודמים ב-1 (פרט למקרים שנראה בהמשך).

אילו היינו נדרשים לכתוב מספר פקודות המבצעות פעולה זהה לפקודות CMPSB, היה עלינו לכתוב כך :

```
push ax  ; Save AX Value
mov al,ds:[si]
cmp al,es:[di]; update flags
ind si
inc di
pop ax  ; Restore AX value
```

הפקודה REPE (REPeat Equal)

כאשר כותבים פקודה זו לפני פקודת מחרוזת אחרת, היא גורמת לביצוע פקודת המחרוזת כל עוד יש שוויון (equal), וגם כל עוד CX אינו שווה ל-0. בכל פעם ש-REPE מתבצעת, ערכו של CX יורד ב-1.

אם נכתוב : REPE CMPSB

נוכל לבדוק האם המחרוזות בבלוק המקור ובבלוק היעד, הן זהות.

בדוגמה הבאה אנו בודקים אם מחרוזת string1 זהה למחרוזת string2 :

```
; string4.asm
Data    segment
   String1      db 'This is a string'      ; The first string to compare
;-------------------------------------------------------------------------------------------
 ; The following causes gives the number of bytes of String1.
 ; The $ means "The current address", and $-String1 means: The
 ; current address - The start address of String1, which actually
 ; gives the number of bytes of String1
;-------------------------------------------------------------------------------------------
 NUMBER_OF_CHARS equ $ - String1
   String2      db 'This is my string'      ; The second string to compare
   EqualMessage db 'The strings are equal...',10,13,'$'
   NotEqMessage db 'The strings are NOT equal!!!',10,13,'$'
Data    ends
Code    segment
  assume cs:Code,ds:Data
Start:
      mov    ax,Data
      mov    ds,ax
      mov    si,offset String1        ; Now DS:[SI] points to String1
      mov    es,ax
```

```
      mov    di,offset String2              ; Now ES:[DI] points to String2
      mov    cx, NUMBER_OF_CHARS           ; Maximum Number of chars to compare
      repe   cmpsb                          ; Continue while equal
;-----------------------------------------------------------------------------------------
;If REPE CMPSB is done, this can be caused by one of the 2 options:
;a) All the cx times are compared and all are equal
;b) One of the comparison isn't equal
;By checking "JNE" we can tell if the LAST comparison is equal. If the
;last one is equal, then ALL are equal.
;-----------------------------------------------------------------------------------------
      jne    NotEq
      mov    dx,offset EqualMessage
      jmp    Finish
NotEq:  mov    dx,offset NotEqMessage
Finish: mov    ah,9
      int    21h                    ; Dispay the message
      mov    ax,4c00h
      int    21h                    ; Terminate
Code    ends
      end    Start
```

לאחר ההרצה נקבל:

```
-n string4.exe
-l
-t10

AX=10FA  BX=0000  CX=0086  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10EA  ES=10EA  SS=10FA  CS=1100  IP=0003   NV UP EI PL NZ NA PO NC
1100:0003 8ED8          MOV     DS,AX

AX=10FA  BX=0000  CX=0086  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10FA  ES=10EA  SS=10FA  CS=1100  IP=0005   NV UP EI PL NZ NA PO NC
1100:0005 BE0000        MOV     SI,0000

AX=10FA  BX=0000  CX=0086  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10FA  ES=10EA  SS=10FA  CS=1100  IP=0008   NV UP EI PL NZ NA PO NC
1100:0008 8EC0          MOV     ES,AX

AX=10FA  BX=0000  CX=0086  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=10FA  ES=10FA  SS=10FA  CS=1100  IP=000A   NV UP EI PL NZ NA PO NC
1100:000A BF1000        MOV     DI,0010

AX=10FA  BX=0000  CX=0086  DX=0000  SP=0000  BP=0000  SI=0000  DI=0010
DS=10FA  ES=10FA  SS=10FA  CS=1100  IP=000D   NV UP EI PL NZ NA PO NC
1100:000D B91000        MOV     CX,0010

/////////////////////////////////////////////////////////////////////////////////////////
 REPE becomes REPZ which is the same
/////////////////////////////////////////////////////////////////////////////////////////
```

```
AX=10FA  BX=0000  CX=0010  DX=0000  SP=0000  BP=0000  SI=0000  DI=0010
DS=10FA  ES=10FA  SS=10FA  CS=1100  IP=0010   NV UP EI PL NZ NA PO NC
1100:0010 F3          REPZ
1100:0011 A6          CMPSB
///////////////////////////////////////////////////////////////////////////////
 After each REPE CMPSB:
   DI=DI+1  and  SI=SI+1
   CX=CX-1
///////////////////////////////////////////////////////////////////////////////
AX=10FA  BX=0000  CX=000F  DX=0000  SP=0000  BP=0000  SI=0001  DI=0011
DS=10FA  ES=10FA  SS=10FA  CS=1100  IP=0010   NV UP EI PL ZR NA PE NC
1100:0010 F3          REPZ
1100:0011 A6          CMPSB
  ---
  ---
  ---
AX=10FA  BX=0000  CX=0008  DX=0000  SP=0000  BP=0000  SI=0008  DI=0018
DS=10FA  ES=10FA  SS=10FA  CS=1100  IP=0010   NV UP EI PL ZR NA PE NC
1100:0010 F3          REPZ
1100:0011 A6          CMPSB
///////////////////////////////////////////////////////////////////////////////
 After the last REPZ CMPSB instruction, Zero-Flag becomes "NZ" (Not Zero)
 which means "Not Equal", thus it jumps to 001A.
///////////////////////////////////////////////////////////////////////////////
AX=10FA  BX=0000  CX=0007  DX=0000  SP=0000  BP=0000  SI=0009  DI=0019
DS=10FA  ES=10FA  SS=10FA  CS=1100  IP=0012   NV UP EI NG NZ AC PO CY
1100:0012 7506          JNZ    001A

AX=10FA  BX=0000  CX=0007  DX=0000  SP=0000  BP=0000  SI=0009  DI=0019
DS=10FA  ES=10FA  SS=10FA  CS=1100  IP=001A   NV UP EI NG NZ AC PO CY
1100:001A BA3C00        MOV    DX,003C
-q
```

הפקודה CMPSW (CoMPare String Word)

הפקודה זהה לפקודה CMPSB, אלא שכאן מתבצעת השוואה בין נתון בגודל מילה בבלוק המקור לבין נתון בגודל מילה בבלוק היעד (פרט למקרים שנראה בהמשך).

הפקודה SCASB (SCAn String Byte)

הפקודה משווה בין ערך האוגר AL, לבין נתון בגודל בית בבלוק היעד. אוגר הדגלים מעודכן בתוצאת ההשוואה. בנוסף, ערכו של DI מקודם ב-1 (פרט למקרים שנראה בהמשך).

בפקודה זו משתמשים כדי לבצע חיפוש של נתון בתוך מחרוזת (דוגמה תובא בהמשך).

הפקודה REPNE (REPeatNot Equal)

פקודה זו דומה ל- REPE, אלא שפקודת המחרוזת ממשיכה להתבצע כל עוד אין שוויון. משום כך משתמשים ב-repne כדי למצוא בית מסוים.

בתוכנית הבאה נשתמש בצמד פקודות כדי לבדוק אם המחרוזת address מכילה את הספרה 7:

repne SCASB

בפתרון התוכנית נעשו ההנחות הבאות:

- המחרוזת מסתיימת בספרה 0
- אורך המחרוזת אינו עולה על 512 בתים.

```
;string5.asm

Data    segment
;    In this string the 0 marks the End of string
   Address    db 'Kop street 375, Dar-tap',0
   AddressSize    dw  ?     ; The size (in bytes) of Address
   MessageFound    db '7 is found in Address...',10,13,'$'
   MessageNotFound  db '7 is NOT found in Address...',10,13,   '$'
Data    ends
Code    segment
  assume cs:Code,ds:Data
Start:
      mov    ax,Data
      mov    ds,ax
;-----------------------------------------------------------------------------------------
; First check the size of the string Address (scan for 0)
;-----------------------------------------------------------------------------------------
      mov    es,ax
      mov    di,offset Address        ; Now ES:[DI] points to Address
      mov    cx,512                   ; Max size of Address
      mov    al,0                     ; The byte to scan (0)
      repne  scasb                    ; Scan for the end of Address
;       AddressSize is equal to   DI - (offset Address)
      mov    AddressSize,di
      sub    AddressSize,offset Address
;-----------------------------------------------------------------------------------------
; Now scan for '7' inside Address
;-----------------------------------------------------------------------------------------
      mov    di,offset Address        ; Now ES:[DI] points AGAIN to Address
      mov    cx,AddressSize           ; The size of Address
      mov    al,'7'                   ; The byte to scan ('7(' 
      repne  scasb                    ; Scan for the byte '7' in Address
```

```
        je      Found               ; If found
        mov     dx,offset MessageNotFound
        jmp     Finish
Found:  mov     dx,offset MessageFound
Finish: mov     ah,9
        int     21h                 ; Dispay the message
        mov     ax,4c00h
        int     21h                 ; Terminate
Code    ends
        end     Start
```

הפקודה SCASW (SCAn String Word)

הפקודה זהה ל-SCASB, אלא שמבוצעת השוואה בין ערך האוגר ax לבין נתון בגודל מילה בבלוק היעד.

הפקודות STD ו-CLD

פקודות אלו נקראות **פקודות כיוון**, מכיון שהן קובעות את כיוון תנועת הסמן במחרוזת לאחר ביצוע הפקודה: קדימה או אחורה.

הפקודה **CLD** (clear direction) גורמת לכך, שלאחר ביצוע כל פקודת מחרוזת, האוגר SI או האוגר DI יקודם ב-1, או ב-2, בהתאם לפקודה.

הפקודה **STD** (set direction) גורמת לכך, שלאחר ביצוע כל פקודת מחרוזת, האוגר SI או האוגר DI יקטן ב-1, או ב-2, בהתאם לפקודה.

שתי פקודות אלו גורמות לשינוי ערכו של דגל הכוון (direction) שבאוגר הדגלים. מכיון שערך דגל הכיוון אינו ידוע מראש, יש לרשום את הפקודה CLD, או STD, לפני שימוש בפקודות מחרוזת. בכל התוכניות שהצגנו עד כה, היה צריך לרשום CLD בתחילת התוכנית, לפני השימוש בפקודות מחרוזת.

דוגמה מסכמת לשימוש בפקודות מחרוזת

בתוכנית הבאה אנו בודקים אם המחרוזת הקטנה SmallString נמצאת בתוך המחרוזת הגדולה BigString, ובהתאם לכך מוצגת הודעה על המסך. שים לב לשימוש בקבוע:

SMALL_SIZE equ $ - SmallString

פעולה זו גורמת לשמירת כמות הבתים של SmallString בתוך הקבוע SMALL_SIZE:

```
; string6.asm

Data segment
  SmallString     db 'name'
 SMALL_SIZE equ $ - SmallString        ; Size (in bytes) of SmallString
  BigString       db 'My name is Ron'
 BIG_SIZE  equ $ - BigString           ; Size (in bytes) of BigString
  MessageFound    db 'SmallString is found in BigString!!!',10,13,'$'
  MessageNotFound db 'SmallString is NOT found in BigString...',10,13,'$'
Data ends
Code segment
 assume cs:Code,ds:Data
Start:
     mov   ax,Data
     mov   ds,ax
     mov   es,ax
     ;-----------------------------------------------------------------------------
     ; Cleare Direction Flag (direction is up, thus SI and DI will be
     ; incremented after every string-instruction
     ;-----------------------------------------------------------------------------
     cld
     mov   cx,BIG_SIZE - SMALL_SIZE; Amount of bytes of to compare
     mov   di,offset BigString    ; ES:[DI] points to BigString
CompareSmall:
     push  di                     ; Save original address of DI
     push  cx                     ; Save the original value of CX
     mov   si,offset SmallString  ; DS:[SI] points to SmallString
     mov   cx,SMALL_SIZE          ; Amount of bytes of the SmallString
     repe  cmpsb                  ; Continue while equal
     pop   cx                     ; Restore original value of CX
     pop   di                     ; Restore original value of DI
     je    Found                  ; If equal - finish
     ;-----------------------------------------------------------------------------
     ; Jump if CX is Zero (If all the bytes of BigString were tested)
     ;-----------------------------------------------------------------------------
     jcxz  NotFound
     inc   di                     ; Next byte of BigString
     dec   cx                     ; One less byte to check
     jmp   CompareSmall
```

```
NotFound:
        mov     dx,offset MessageNotFound
        jmp     Finish
Found:  mov     dx,offset MessageFound
Finish: mov     ah,9
        int     21h             ; Dispay the message
        mov     ax,4c00h
        int     21h             ; Terminate
Code    ends
        end     Start
```

תרגילים

1. עליך לכתוב תוכנית שבודקת אם כל התווים של מחרוזת בשם Srting1, המסתיימת בנקודה, זהים זה לזה.

   לדוגמה, אם String1 מכיל "AAAAAAAA." כל התווים זהים, ואז יש להציג את ההודעה "eqaul" על המסך.

2. רשום תוכנית הבודקת אם צמד התווים "MZ" מופיעים בתוך מקטע הנתונים בתחום הכתובות 30h-100h. אם כן, יש להציב במשתנה Place את הכתובת בה נמצא צמד התווים; אחרת, יש להציב אפס במשתנה זה.

3. כתוב תוכנית שקולטת מלוח המקשים מחרוזת, עד להקשה על Enter. המחרוזת תישמר בתוך מערך בשם input בתוך מקטע הנתונים. תוכל להניח שגודל המחרוזת הנקלטת אינו עולה על 50 תווים. הנח שהמחרוזת הנקלטת כוללת שתי מילים, שביניהן תו רווח אחד, למשל: "beautiful computer".

   התוכנית תציג על המסך את המילים בסדר הפוך: "computer beautiful".

4. כתוב תוכנית שמציבה את הערך 33H בכל תאי הזיכרון בתחום הכתובות B800:OH - B800:300H.

תזכורת: מדובר בכתובת פיסית, שבה B800 הינו הבסיס, ו- 300H הוא ההיסט. תוכל להשתמש בפקודת מחרוזת rep stosb לצורך זה. אם התוכנית פועלת כראוי, תוכל לראות על המסך את הספרה 3 מוצגת בכמה שורות בצבע. הסיבה לכך: אזור הזיכרון הזה, שנקרא "זיכרון מסך", גורם להצגת התווים במסך.

18

פקודות נוספות

הפקודה MUL

פקודה זו משמשת לביצוע כפל חשבוני. ניתן להשתמש בה באחת משתי צורות:

1. כפל של אופרנד בגודל בית. מבנה הפקודה:

MUL <אוגר/משתנה בגודל בית>

הפקודה תגרום להכפלת ערך האוגר/המשתנה, בערכו של האוגר AL (האוגר קבוע ואינו ניתן לשינוי) ותציב את התוצאה בגודל מילה, באוגר AX (באופן קבוע).
לדוגמה: MUL BL

פקודה זו תגרום לכפל בין ערך האוגר BL לבין ערך האוגר AL, והתוצאה תוצב באוגר AX.

2. כפל של אופרנד, בגודל מילה. מבנה הפקודה זהה:

MUL <אוגר/משתנה בגודל מילה>

הפקודה תגרום לכפל ערך האוגר/המשתנה, בערכו של האוגר AX (קבוע) ותציב את התוצאה בגודל מילה-כפולה באוגרים DX (הספרות הבכירות) ו-AX (הספרות הזוטרות).

לדוגמה: MUL CX

הפקודה תגרום לביצוע כפל בין ערך האוגר CX לבין ערך האוגר AX, והתוצאה תוצב בצמד האוגרים DX ו- AX. דוגמה לשימוש בפקודה:

```
; mul1.asm

Code segment
  assume cs: Code, ds:Code
Starting:
        mov    ax,Code
        mov    ds,ax
        mov    cl,5
        mov    al,4
        mul    cl       ; AX  =  5  * 4
```

```
        nop
        nop
        nop
Code ends
    end Starting
```

לאחר האסמבלר והקישור, נקבל:

```
-n mul1.exe
-l
-t5

AX=13A6  BX=0000  CX=000E  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1396  ES=1396  SS=13A6  CS=13A6  IP=0003   NV UP EI PL NZ NA PO NC
13A6:0003 8ED8          MOV      DS,AX

AX=13A6  BX=0000  CX=000E  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A6  CS=13A6  IP=0005   NV UP EI PL NZ NA PO NC
13A6:0005 B105          MOV      CL,05

AX=13A6  BX=0000  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A6  CS=13A6  IP=0007   NV UP EI PL NZ NA PO NC
13A6:0007 B004          MOV      AL,04

AX=1304  BX=0000  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A6  CS=13A6  IP=0009   NV UP EI PL NZ NA PO NC
13A6:0009 F6E1          MUL      CL

//////////////////////////////////////////////////////////////////////////////////////////
  result in AX = 14h = 20
//////////////////////////////////////////////////////////////////////////////////////////
AX=0014  BX=0000  CX=0005  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A6  CS=13A6  IP=000B   NV UP EI PL NZ NA PO NC
13A6:000B 90            NOP
-q
```

תרגיל: רשום תוכנית שמציבה במשתנה C (בגודל מילה), את תוצאת הפעולה החשבונית הבאה (גודלו של כל משתנה שלהלן הינו בית):

A = C * (B - 2) + 5

הפקודה DIV

הפקודה DIV מבצעת פעולת חילוק. בפקודה זו ניתן להשתמש באחת משתי צורות:

1. חילוק בין ערך האוגר AX (קבוע) לבין משתנה או אוגר בגודל בית. מבנה הפקודה:

DIV <אוגר/משתנה בגודל בית>

למשל, אם נרשום: DIV BL, יחולק ערך האוגר AX בערכו של האוגר BL. התוצאה השלמה תוצב באוגר AL (קבוע, בלתי ניתן לשינוי) והשארית תוצב באוגר AH (באופן קבוע).

2. חילוק בין המילה הכפולה DX AX (DX מכיל את הספרות הבכירות ו-AX מכיל את הספרות הזוטרות), לבין משתנה בגודל מילה, או אוגר בגודל מילה. מבנה הפקודה:

DIV <אוגר/משתנה בגודל מילה>

למשל, אם נרשום: DIV BX, יחולק ערך המילה הכפולה DX AX, באוגר BX. התוצאה תוצב באוגר AX, ואילו השארית תוצב ב-DX (קבוע, בלתי ניתן לשינוי).

דוגמה לשימוש ב- DIV:

```
; div1.asm

D  segment
      Number   dw 13 ; Number to devide
      Divided  db 0  ; The result after divided by 4
      Reminder db 0  ; The reminder
D  ends
S segment stack 'stack'
      dw 100h dup (?)
S ends
C segment
  assume cs:C, ds:D, ss:S
Begin:  mov    ax, D
      mov    ds, ax
      mov    ax, Number
      mov    dl, 3
      div    dl          ; AL = Number/3
                         ; (AH is the reminder)
      mov    Divided,  al
      mov    Reminder, ah
      nop
      nop
C ends
  end Begin
```

לאחר הרצת התוכנית נקבל:

```
-n div1.exe
-l
-t7

AX=13A6  BX=0000  CX=0025  DX=0000  SP=0200  BP=0000  SI=0000  DI=0000
DS=1396  ES=1396  SS=13A9  CS=13A7  IP=0003   NV UP EI PL NZ NA PO NC
13A7:0003 8ED8          MOV DS,AX

AX=13A6  BX=0000  CX=0025  DX=0000  SP=0200  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A9  CS=13A7  IP=0005   NV UP EI PL NZ NA PO NC
13A7:0005 A10000        MOV AX,[0000] DS:0000=000D

AX=000D  BX=0000  CX=0025  DX=0000  SP=0200  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A9  CS=13A7  IP=0008   NV UP EI PL NZ NA PO NC
13A7:0008 B203          MOV       DL,03

AX=000D  BX=0000  CX=0025  DX=0003  SP=0200  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A9  CS=13A7  IP=000A   NV UP EI PL NZ NA PO NC
13A7:000A F6F2          DIV       DL

///////////////////////////////////////////////////////////////////////////////////
 AL = 4 (result), AH = 1 (reminder)
///////////////////////////////////////////////////////////////////////////////////
AX=0104  BX=0000  CX=0025  DX=0003  SP=0200  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A9  CS=13A7  IP=000C   NV UP EI NG NZ NA PO NC
13A7:000C A20200        MOV [0002],AL  DS:0002=00

AX=0104  BX=0000  CX=0025  DX=0003  SP=0200  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A9  CS=13A7  IP=000F   NV UP EI NG NZ NA PO NC
13A7:000F 88260300      MOV [0003],AH  DS:0003=00

AX=0104  BX=0000  CX=0025  DX=0003  SP=0200  BP=0000  SI=0000  DI=0000
DS=13A6  ES=1396  SS=13A9  CS=13A7  IP=0013   NV UP EI NG NZ NA PO NC
13A7:0013 90            NOP
-q
```

תרגיל: כתוב פרוצדורה שמקבלת דרך משתנה A בגודל בית את מספר החודשים, מחשבת ומציבה במשתנה B את מספר השנים ובמשתנה C - את יתרת החודשים. לדוגמה, אם ב-A יש 15, אזי ב-B צריך להיות 1 (שנה אחת) וב-C יהיה 3 (שארית החילוק).

כפל וחילוק מספרים מסומנים - IDIV, IMUL

הפקודות DIV ו-MUL מתייחסות לכל המספרים כאל חיוביים. למשל, המספר 85H שגודלו בית ייחשב כמספר חיובי. אם המתכנת רוצה לפעול עם מספרים מסומנים, כלומר בעלי ערך חיובי או שלילי, עליו להשתמש בפקודות אחרות:

```
IDIV
IMUL
```

במקרה זה, המספר 85H נחשב מספר שלילי, כי יש לו ספרה 1 משמאל.

השימוש בפקודות אלו זהה לזה שהוצג עבור DIV ו-MUL.

הפקודה CBW

הפקודה CBW היא קיצור של Convert Byte to Word. CBW מרחיבה את הנתון גודל בית שבאוגר AL (קבוע, בלתי-ניתן לשינוי), לנתון בגודל מילה ומציבה אותו באוגר AX. למשל:

```
MOV AL,93H
CBW
```

AX יכיל כעת את הערך FF93H.

המספר 93H הוא מספר שלילי (תזכורת: כאשר הסיבית השמאלית ביותר היא "1" אזי המספר הוא שלילי). לכן, כאשר מרחיבים את הנתון מגודל בית לגודל מילה, מוסיפים סיביות "1" לכל הספרות המשמעותיות.

הפקודה CWD

בדומה ל-CBW, גם הפקודה CWD מרחיבה נתון בגודל מילה שנמצא באוגר AX, לצמד האוגרים DX AX (כאשר DX מכילה את 4 הספרות הבכירות, המשמעותיות יותר). לדוגמה:

```
MOV AX,9831H
CWD
```

כעת AX יישאר בערכו הנוכחי ו-DX יקבל FFFF, כך שבסופו של דבר מקבלים נתון בגודל מילה כפולה:

```
FFFF9831H
```

תרגיל: נניח שלא היתה קיימת הפקודה CWD. כתוב פרוצדורה המבצעת פעולה זהה ל-CWD.

הפקודה DAA

שם הפקודה נגזר מ- Decimal Adjustment for Adition. הפקודה מבצעת המרה עשרונית לאחר פעולת **חיבור**. אפשר להשתמש בה רק לאחר פעולת חיבור, ורק על האוגר AL. נראה דוגמאות לשימוש בפקודה:

MOV AL,8	
ADD AL,4	כעת ערכו של AL הוא 0CH
DAA	כעת ערכו של AL הוא 12H

כך, כביכול, ביצענו חיבור עשרוני של: 12 = 4 + 8.

תרגיל: כתוב תוכנית שמציגה על המסך בשורות נפרדות את המספרים העשרוניים מ- 1 עד 20 והשתמש בפקודה זו.

הפקודה DAS

שם הפקודה DAS נגזר מ- Decimal Adujstment for Subtruct.

הפקודה DAS דומה ל-DAA, אך פועלת עבור **חיסור**.

פקודות לשמירה ואיחזור ערך אוגר הדגלים

הפקודה PUSHF שומרת את ערך אוגר הדגלים בתוך המחסנית, והפקודה POPF שולפת את הנתון מהמחסנית ומציבה אותו באוגר הדגלים.

לדוגמה, הפרוצדורה הבאה שומרת את ערך אוגר הדגלים לפני שהיא משנה אותו (תזכורת: הפקודה CMP גורמת לשינוי באוגר הדגלים). לפני היציאה מהפרוצדורה היא משחזרת את ערכו המקורי.

```
      PUSHF
      CMP AX,8
      JE YES
      MOV AX,2
YES:  POPF
      RET
```

לאותה מטרה אפשר להשתמש גם בפקודות האלו:

- Load AH with Flags - LAHF - פקודה זו מעתיקה את הבית הפחות משמעותי (LSB) של אוגר הדגלים אל אוגר AH.
- Save AH into Flags - SAHF - פקודה זו מעתיקה את הערך מאוגר AH אל אוגר הדגלים (פעולה הפוכה).

פקודות קלט-פלט: IN ו-OUT

המעבד (CPU) יכול לשלוח ולקבל נתונים מרכיבי-חומרה אחרים. כדי לעשות זאת, המתכנת צריך לדעת מהי כתובת רכיב החומרה שאליו רוצים לפנות. מתכנני החומרה הם אלה שקבעו את כתובות הרכיבים שהמעבד מתקשר איתם. כתובת של התקן-חומרה נקרא **פורט** (PORT).

כדי לקלוט נתון מרכיב חומרה, משתמשים בפקודה IN, באחת משתי דרכים:

- IN AL, כתובת - (קליטת נתון בגודל בית)
- IN AX, כתובת - (קליטת נתון בגודל מילה)

אם הכתובת בתחום 0-FFH, ניתן לרשום אותה בפקודה. אולם, כאשר הכתובת גדולה יותר, יש להיעזר באוגר DX. למשל:

```
IN AL, 43H
```

אבל:

```
MOV DX, 158H
IN AL,DX
```

כדי לשלוח נתון לרכיב חומרה, משתמשים בפקודה OUT:

```
OUT כתובת , AL
OUT כתובת , AX
```

גם כאן יש להסתייע באוגר DX עבור כתובות הגדולות יותר מ-FFH.

לדוגמה, רמקול המחשב מופעל על ידי כתובת 61H. סיבית D1 (הסיבית השנייה מימין) בכתובת זו גורמת ליצירת הצליל ברמקול. כאשר ניתן לסיבית זו את הערכים "1" ו-"0" לסירוגין, יווצר צליל. ככל שניתן את הסיביות "1" ו-"0" בהפרשי-זמן קצרים יותר, כך הצליל יהיה גבוה יותר (ולהיפך).

כתובת 61H משמשת גם עבור התקני-חומרה אחרים במחשב ולכן, כדי ליצור צליל ברמקול, נדאג לשנות את סיבית D1 בלבד, מבלי לשנות את ערכי הסיביות האחרות.

התוכנית הבאה גורמת ליצירת צליל ברמקול. אם נשנה את ערכו של המשתנה DELAY, נוכל לקבל צליל גבוה יותר, או נמוך יותר.

```
; inout1.asm

PORT  =  61h                    ; Constant used to define the port used
Data    segment
  Delay         dw   ?          ; Defines the sound level - high/low
  SoundLength   dw   ?          ; Defines the sound length
Data    ends
S       segment stack 'stack'
  dw 100h dup (?)
S       ends
Code    segment
  assume cs:Code, ds:Data, ss:S
```

```
Start:  mov     ax,Data
        mov     ds,ax               ; Initialize
        mov     SoundLength, 8000h ; Defines the length of the sound
        in      al,PORT             ; Read the byte from the port
ContinueSound:
        xor     al,00000010B    ; Change bit D1 only (second bit from the right)
        out     PORT,al         ; Send to the port
        mov     Delay,1000h     ; Delay between "1" and "0" (sound level)
MoreDelay:
        dec     Delay
        jnz     MoreDelay       ; Delay
        dec     SoundLength
        jnz     ContinueSound   ; Sound length
        and     al,11111101B    ; Reset bit D1 in order to stop the sound
        out     PORT,al         ; No more sound
        mov     ax,4c00h
        int     21h             ; Terminate to Operating system
Code    ends
        end Start
```

תרגילים

1. על בסיס התוכנית שהוצגה, כתוב תוכנית שמשנה את הצליל עבור כל אחד מהמקשים 0 עד 9.
2. כתוב תוכנית שגורמת לצליל אזעקה עולה ויורד.

19

אסמבלי ושפת C

(פרק זה נכתב על ידי **אסף שלי**. לפניות : ASM@Shelly.co.il)

שפת אסמבלי לעומת שפות עיליות

את שפות התכנות נהוג לסווג לשני סוגים :

1. **שפה עילית**, כמו למשל פסקל, C, ויזואל בייסיק ועוד. בדרך כלל, בשפה עילית כל פקודה מתורגמת למספר רב של פקודות בשפת מכונה.
2. **שפת סף**, כמו שפת אסמבלי. שפת אסמבלי מאופיינת בכך, שכל פקודה שלה מתורגמת לפקודה בודדת בשפת מכונה.

בדרך כלל, קל יותר לכתוב יישומים בשפה עילית. השימוש בשפת אסמבלי נובע מכך שיש לה מספר יתרונות חשובים על פני שפות עיליות :

- התוכנית המוכנה להרצה (Executable) קטנה יותר.
- זמן ביצוע התוכנית קטן יותר, לפעמים בכמה עשרות מונים.
- השפה מאפשרת גישה נוחה לרזי-המחשב, לרבות המחסנית.

לעיתים, רוצים לשלב את יתרונות שפת האסמבלי עם פשטות השפה העילית. למשל, כאשר הממשק (User Interface) נכתב בשפה Visual C, וחלק מהפרוצדורות שבהן מהירות ביצוע הינה גורם קריטי - מפתחים את התוכנית, או את קטע הקוד, בשפת אסמבלי.

אחת השפות בעלות העוצמה היא שפת C, ולכן הדוגמאות בפרק זה יתבססו על שימוש בשפה זו.

שילוב קוד אסמבלי בפקודות שפת C

כדי לשלב פקודות אסמבלי בתוכנית בשפת C, יש להשתמש במילה השמורה "_ _asm" (או "_asm", או "asm", על פי סוג המהדר). הנה כך:

```
_ _asm  mov ax,3
```

המילה **_ _asm** (שני קווים תחתיים משמאל) פירושה: "לפניך פקודת אסמבלי בהמשך השורה".

המהדר Borland C 5.02 תומך בשלושה אופני כתיבה: _ _asm, _asm, asm; מהדר Borland C 3 תומך בצורה asm; המהדר MS Visual C++ תומך בצורה _ _asm בלבד. כל הדוגמאות כאן ישתמשו ב- _ _asm.

הפקודה תסתיים במעבר לשורה חדשה כפי שנהוג באסמבלי, או באמצעות נקודה-פסיק (;), כנהוג בשפת C:

```
_ _asm pop ax
_ _asm pop bx; _ _asm pop cx
_ _asm pop dx
```

מותר לכתוב מספר תווי ';' רצופים, אבל משמעות התו אינה **הערה**, אלא **פקודה חדשה**. כדי לציין הערה, יש להשתמש בשיטה שמוכרת למהדר C – שני תווי לוכסן (//) צמודים:

```
_ _asm xor ax,ax          ;// This is a note: Clear AX
```

שימוש כזה בפקודות אסמבלי בתוכנית C נקרא Inline Assembly – **אסמבלי מוכלל**.

כדי לכתוב מספר פקודות אסמבלי, נשתמש בסימון בלוק של שפת C:

```
_ _ asm {
    pop ax
    pop bx
  }
```

שים לב, שהסוגריים המסולסלים לפתיחת בלוק חייבים להימצא בשורה של המילה ASM. אם הם יהיו בשורה אחרת, הם ייחשבו כפתיחת בלוק של פקודות C:

```
_ _asm
 {
   int a=5;         ;// C and not Assembly
  a++;              ;// C and not Assembly
 }
```

אם היינו רושמים פקודות אסמבלי בבלוק זה, היינו מקבלים הודעת שגיאה.

שילוב אסמבלי עם משתני שפת C - אסמבלי מוכלל

בפקודות אסמבלי מוכלל מותר להשתמש במשתנים של התוכנית הראשית הכתובה בשפת C. לדוגמה, נצהיר על משתנה בגודל מילה בשם A. בתחילה ערכו הוא 5, ואחר כך אנו משתמשים בפקודה בשפת אסמבלי, כדי להגדיל את ערכו ב-7:

```
int a=5;                  ;// C declaration
_ _asm add a,7;           ;// Using Assembly
```

באופן דומה, נוכל להכניס מספר פקודות אל תוכנית או פרוצדורה בשפת C:

```
int proced  (int a,int b)
{
  _ _asm {
        add a,7;
        dec b;
   }
  return(a);
 } // End of C function
```

מהדרים מגרסאות קודמות אינם תומכים באסמבלי מוכלל באופן מלא, ובמקום זאת הם שולחים את הקטע באסמבלי כפי שהוא – להידור. דוגמה למהדר מסוג זה הוא Turbo C 3. עבור מהדר זה נכתוב את הדוגמה הקודמת בדרך זו:

```
asm {
       Mov ax,a
       Add ax,7
       Mov a,ax
      }
```

פירוש הדבר הוא, שפקודות אסמבלי חייבות להיכתב על פי חוקי האסמבלר הרגיל. גם לא נוכל להשתמש בתוויות (labels) במהדרים אלה, ולכן לא ניתן להשתמש במשפטי בקרה jmp הפונים אל קטעי קטע בתוכנית האסמבלי. כדי להתגבר על קושי זה עלינו להשתמש במשפט if מחוץ לבלוק האסמבלי. להלן דוגמה:

```
asm{
        Mov  cx,3
        Add a,3
        Loop x
       }
```

תמיכה לא מלאה באסמבלי מוכלל, יכולה להיראות כך:

```
asm{
        Mov cx,3
       }
x:
asm {
        mov ax, a
        add ax,3
        mov a,ax
        loop x   //  _x למעשה יש לרשום
       }
```

בדוגמאות שנציג בהמשך נשתמש ב**אסמבלי מוכלל בתמיכה מלאה**.

כעת, ננצל את היתרון של שפת אסמבלי ונבצע פעולה ששפת C אינה יכולה לעשות, מכיון שהיא שפה עילית. שפת C אינה יכולה לבדוק גלישה של משתנים.

לפניך דוגמה לפרוצדורה שמבצעת חיבור של שני משתנים ומשנה את ערכו של משתנה שמוגדר בתחילת התוכנית (גלובלי) לפי מצב הגלישה: היתה גלישה=1; לא היתה גלישה=0. המשתנה הוא OVRFLW.

```
int operator_plus (int a, int b)
 {
  _ _asm {
        mov ovrflw,0;    ;// איפוס המשתנה שמציין גלישה
        add a,b;         ;// חיבור שני משתנים
        jno xx01;        ;// אם לא היתה גלישה לך לסוף
        mov ovrflw,1;    ;// אם הגענו לכאן היתה גלישה
   xx01:                 ;// תווית לציון סוף הבלוק
  }
  return (a);
 }
```

בדוגמה זו כתובה הפקודה "ADD A,B", שבה שני משתנים. בתוכנית רגילה בשפת אסמבלי הדבר אסור, אך באסמבלי המוכלל בשפת C מותר. המהדר ממיר את הפקודה הזו במספר פקודות של שפת אסמבלי. ראוי לשים לב לנקודות אלו:

1. אם משנים את ערכי האוגרים, יש להחזירם לערכם המקורי. האוגרים הכלליים ax, bx, cx ו-dx מיועדים לשימוש כללי ואין לשנותם, בתנאי שאין קוד C נוסף לאחר שינויים (סיום פרוצדורה).
2. אסור לשנות נתונים במחסנית. מצב המחסנית ביציאה מבלוק (קטע קוד התחום בסוגריים מסולסלים) חייב להיות בדיוק כפי שהיה בכניסה אליה.

לדוגמה, נשתמש באסמבלי מוכלל, כדי לכתוב פקודה בשפת ++C שגורמת לעכבר להופיע במצב DOS:

```
void Mouse_Show()
 {
  _ _asm {            ;// prepare mouse for work.
   mov ah,0
   int 33h
   }
  _ _asm {            ;// show mouse
   mov ah,1
   int 33h
   }
 }
```

קישור אסמבלי לשפת C

מהדר שפת C יוצר קובץ יעד (Object) שהסיומת שלו היא OBJ. המקשר (Linker) מסוגל לקשר מספר קבצי OBJ יחד, כך שתיווצר תוכנית שלמה מוכנה להרצה. היתרון הוא, שניתן לשמור ספריות של פקודות מוכנות מראש, ולהשתמש בהן בכל תוכנית. יתרון עצום שיש לקישור הוא, שניתן לקשר שני חלקי תוכנית שנכתבו **בשתי שפות תכנות שונות**. למעשה, פקודות שפת C הן קבצי יעד עם סיומת מיוחדת (lib) והמהדר מוסיף את הפקודות הנכונות מתוך קבצים אלה.

נשתמש בתוכנית הקישור, כדי לקשר קובץ OBJ של C עם קובץ OBJ של אסמבלי ונדגים על ידי תוכנית פשוטה, כיצד לבצע זאת. הקובץ הראשון הוא קובץ אסמבלי המכיל פרוצדורה הקוראת לפסיקה 33H, כדי לגרום לעכבר להופיע.

```
CODE SEGMENT:
 M_show  PROC  NEAR          ;// הצהרה
   mov ax,0001h              ;// הכנת מספר פעולה של הפסיקה
   int 33h                   ;// קריאה לפסיקה
   ret                       ;// חזרה
 M_show  ENDP
```

כעת, עלינו להגדיר את הפרוצדורה בשפת C, כך שהמהדר של השפה יידע כיצד לפנות אליה.

```
void M_show (void);
```

בכך מסתיימת ההגדרה בשפת C. המהדר יודע כעת, שקריאה לפרוצדורה זו אינה דורשת פרמטרים ואינה מחזירה דבר. השימוש בפקודה בתוך שפת C פשוט למדי:

```
main()
 {
  M_show();
  exit(0);
 }
```

התוכנית בשפת C קוראת לפקודה של תוכנית הכתובה באסמבלי. המהדר שותל פקודת קריאה, אך הפרוצדורה עצמה עדיין אינה קיימת בתוכנית. תפקיד המקשר למצוא את הקריאה שמקורה בשפת C ולקשר אותה אל הפקודה שבאסמבלי.

כל מהדר יוצר קובץ יעד שיש בו פקודות למעבד (בשפת מכונה). עם זאת, במקום כתובות יש בו את שמות המשתנים והפרוצדורות. תפקיד המקשר הוא לתת לכל שם כתובת, ולכתוב אותה במקום שם המשתנה שמופיע בפקודות השונות. כך, כאשר המקשר מבצע את הקישור, הוא מחליף את השמות בכתובות ויוצר תוכנית שבה פקודות בשפת מכונה שהמעבד יכול לבצע.

קישור

לאחר הידור קובץ אסמבלי בשם STDIO.ASM למשל, וקובץ בשפת C בשם PROG.C, קיבלנו שני קבצי יעד (OBJ). כדי לקשר אותם בעזרת מקשר ששמו LINK, לדוגמה, נכתוב כך:

```
link  prog.obj + stdio.obj
```

התוצאה תהיה תוכנית יעד בשם PROG.EXE.

אין חובה לרשום למקשר את הסיומת של הקובץ אם היא OBJ. חובה להגדיר לשפת C את הפרוצדורות החדשות. ללא הגדרה, המהדר לא יידע כיצד להשתמש בהן.

העברת פרמטרים משפת אסמבלי לשפת C ולהיפך

שפת C מאפשרת אותיות גדולות, אותיות קטנות, קו תחתון וספרות (אם אינן תו ראשון). המהדר של שפת C מוסיף קו תחתון ('_') כתו ראשון של כל שם בשפה. כדי לפנות לשם, משתנה או פונקציה, יש לזכור ששמו באסמבלי הוא בדיוק שמו בשפת C, אלא שנוסף לו '_' מקדים. כך, הפרוצדורה 'putc' בשפת C נקראת בשם '_putc' באסמבלי.

משתנה בשם 'X' בשפת C יהיה 'X_' באסמבלי. משתנה 'X_' בשפת C יהפוך ל- 'X_ _' (שני קווים תחתונים) באסמבלי. למעשה, הדוגמה בסעיף הקודם, היתה אמורה להכיל פרוצדורה באסמבלי בשם "M_show_".

פרוצדורה בשפת C יכולה לקבל פרמטרים. בקובץ המקור בשפת C יש לכתוב את הערכים בתוך סוגריים, לאחר שם הפרוצדורה, למשל:

```
putchar(var_01);
```

הפקודה קוראת לפרוצדורה בשם "putchar" ומעבירה לה את ערכו של המשתנה "var_01". בפועל העניין מורכב מעט יותר. העברת משתנים בשפת C כרוכה בהכנסתם למחסנית. עבור הפרוצדורה שבדוגמה, הנוהל הוא כזה:

- הכנסת ערכו של var_01 למחסנית (Push).
- ביצוע קפיצה בעזרת CALL (דחיפה למחסנית של הכתובת לחזרה).
- הפרוצדורה משתמשת בנתון, אך **משאירה** אותו במחסנית (קריאה בעזרת BP).
- ביצוע RET (שליפת הכתובת לחזרה מהמחסנית).
- ניקוי המחסנית מהנתונים שהועברו, על ידי מי שהכניס אותם.

להלן קטע אסמבלי שקורא לפרוצדורה _putchar :

```
CODE SEGMENT:
_callProc PROC NEAR
  mov ax,var_01
  push ax            ; //   הכנסת הפרמטר למחסנית
  call _putchar      //     הכנסת כתובת חזרה למחסנית
                                                   ;
  pop ax             ; //   ניקוי הפרמטרים מהמחסנית
  ret
_callProc ENDP
```

כשנסתכל על המחסנית, נראה כעת שהיא מכילה כתובת חזרה, ומייד לפניה את הנתון שהועבר, ערכו של var_01.

היישום של הפרוצדורה שמקבלת את הפרמטרים דרך המחסנית ייראה כך :

```
_putchar PROC NEAR        ;//   הגדרת הפרוצדורה
  push bp                 ;//   שמירת ערכו של bp, כדי שנוכל לשחזר אותו בסיום
  mov bp,sp               ;//   קבלת הכתובת של הפרמטר האחרון (התחתון) במחסנית
  mov dl,byte ptr[bp+4]   ;//   קבלת הפרמטר שנדחף לפני bp ולפני כתובת החזרה
  mov ah,02               ;//   פונקציית הדפסה של DOS
  int 21h                 ;//   בקשה פסיקה של DOS
  pop bp                  ;//   שחזור אוגר bp מהמחסנית
  cld                     ;//   עבור שפת C חובה לוודא כי DF מאופס
  ret                     ;//   הוצאת כתובת חזרה מהמחסנית
_putchar ENDP
```

לאחר פקודת חזרה (RET), הוּצָאָה כתובת החזרה מהמחסנית על ידי הפקודה RET עצמה. כדי להמשיך באופן תקין, יש לשחרר את ערכו של val_01 מהמחסנית, לאחר שהוכנס בה בזמן הקריאה, כפרמטר pop cx.

בניגוד לשפת פסקל, שפת C מחייבת את **מי שקרא** לפרוצדורה לשחרר את הנתונים מהמחסנית. הנוהל בשפת פסקל הוא כזה, שהפרוצדורה עצמה מנקה את המחסנית. לכן, הפרוצדורה צריכה לבצע את הפקודה POP בעצמה, לפני פקודת החזרה RET. כדי לבצע זאת ניתן גם לכתוב :

```
ret 2
```

המעבד "יודע" שהכוונה היא לניקוי המחסנית לפני החזרה, ובפועל הוא מעלה את ערכו של האוגר SP ב-2.

ערך מוחזר: כדי להחזיר בית (char בשפת C), נשתמש בחצי אוגר AL ; כדי להחזיר מילה (int בשפת C) נשתמש באוגר AX ; כדי להחזיר מילה כפולה (long בשפת C) נשתמש בצמד של אוגר DX כמילה עליונה, ובאוגר AX כמילה תחתונה.

הדגמת הקישור

כעת, נראה כיצד תיראה תוכנית שמצליחה לנצל את הפוטנציאל שקיים בפעולת הקישור. נסביר את התוכנית ואת שלבי התכנון השונים.

התוכנית עוסקת ב**הפעלה בסיסית של העכבר.**

בתחילה עלינו להכין את העכבר לפעולה. נבצע זאת בעזרת פסיקה 33H, פונקציה 0000H. שם הפקודה יהיה "M_ini", קיצור של "mouse initialization" (אתחול עכבר). כדי לדעת מה הפקודה מחזירה, עלינו לבדוק תחילה מה מחזירה הפסיקה:

- פסיקה: 33h.
- פונקציה: ax=0000h.
- פעולה: אתחול מנהל התקן (דרייבר) העכבר.
- ערכים נוספים מועברים באוגרים: אין.
- ערכים מוחזרים:
  1. AX: 0000h אם לא ניתן לזהות את ההתקן, ffffh אם הכל תקין.
  2. BX: 0002h או ffffh - שני לחצנים, 0003h - שלושה לחצנים, 0000h - מספר אחר של לחצנים.

כעת, נוכל ליצור פרוצדורה פשוטה שתחזיר '1' אם הצליחה, או תחזיר '0' אם נכשלה (זהו נוהל רגיל בשפת C). נראה כעת כיצד לבצע זאת.

נגדיר פרוצדורה בשפת C, שתוכנס לקובץ MOUSE.H:

```
int M_ini (void);
```

פרוצדורה זו אינה מקבלת פרמטרים כלשהם (void), אך מחזירה משתנה בגודל מילה (INT). כעת, נגדיר את הפרוצדורה בשפת אסמבלי, שתוכנס לקובץ MOUSE.ASM:

```
_TEXT  SEGMENT
  _M_ini  PROC  NEAR
    mov ax,0000h        ;// הפונקציה של הפסיקה
    int 33h             ;// קריאה לפסיקה
    and ax,0001h        ;// אם ערכו FFFFh שיהיה 1
    cld                 ;// איפוס דגל כיוון (דגשים בהמשך)
    ret                 ;// חזרה
  _M_ini ENDP
_TEXT ENDS
```

מקטע הפקודות נקרא "TEXT_", כיון שכך מכנה אותו שפת C. כל הפקודות צריכות להיות באותו המקטע. כעת נוכל להשתמש בפקודה בתוך התוכנית בשפת C:

```
#include <stdio.h>
#include "c:\my_lib\mouse.h"
main()
 {
  if (!M_ini()) puts("NO MOUSE FOUND");
   else puts("MOUSE O.K.");
 }
```

התוכנית בשפת C תקרא את הערך שחוזר מהפרוצדורה. אם הערך הוא '0' יירשם על המסך שהעכבר לא נמצא; ואם הערך הוא '1' יירשם כי העכבר בסדר.

שם הקובץ בשפת C יהיה SETMOUSE.C בדוגמה זו. בתוכו קיימת פקודה להוספת קובץ ההגדרות MOUSE.H, שבו מוגדרת הפרוצדורה עבור שפת C.

כאשר נהדר את התוכנית נקבל קובץ בשם SETMOUSE.OBJ, אשר מיועד למקשר. אם ננסה לקשר אותו, נקבל הודעת שגיאה שמשמעותה: "פנייה לכתובת לא ידועה M_ini_ במודול SETMOUSE.OBJ". כלומר, יש קריאה לפרוצדורה שגוף ההגדרה שלה חסר, מכיון שהוא נמצא בקובץ אחר.

לאחר הידור הקובץ MOUSE.ASM נקבל קובץ בשם MOUSE.OBJ. כדי לקשר את שניהם יחד באמצעות המקשר Link.exe, נכתוב:

```
LINK SETMOUSE.OBJ+MOUSE.OBJ
```

פעולה זו תיצור קובץ הרצה ניתן לביצוע, ושמו כשם הקובץ הראשון שנמסר למקשר (SETMOUSE בדוגמה), והסיומת שלו היא EXE, שמציינת שהוא קובץ בר-הרצה.

כדי להפעיל את התוכנית שיצרנו, נכתוב בשורת הפקודה:

```
SETMOUSE
```

התוכנית תזהה אם מותקן במחשב עכבר, או לא.

הערה: המהדר של מיקרוסופט משתמש תוכנית קישור (Linker) בשם Blink.exe, ומהדר בורלנד משתמש בתוכנית מקבילה בשם Tlink.exe. ניתן להשתמש בפקודה Dir שבספרייה \Bin של המהדר כדי למצוא את תוכנית הקישור שיש להשתמש בה.

הערות חשובות

להלן מספר הערות, לתשומת לבך, בעת קישור של תוכנית אסמבלי לתוכנית בשפת C :

- בהעברת מספר פרמטרים לפרוצדורה, הם נדחפים למחסנית בסדר הפוך מסדר הצגתם בשורה. קריאת הפרמטרים האלה בעזרת פקודות POP תשחזר אותם בסדר הנכון, כפי שנכתבו בשפת C בהגדרת הפרוצדורה.
- מייד לפני הקריאה נדחפים הפרמטרים למחסנית; מייד לאחר הקריאה הפרמטרים משוחררים מהמחסנית. אם הפרוצדורה קוראת את הנתונים בעזרת POP, עליה להחזיר את המחסנית למצבה הקודם על ידי שימוש בפקודה PUSH.
- מומלץ לדחוף את אוגר BP למחסנית ואז להשתמש בו לקריאת הנתונים. בסיום יש לשחזר אותו.
- אם התבצעה קריאה לפסיקה, יש לכתוב את הפקודה CLD מייד לפני החזרה.
- הפרוצדורה צריכה להיות מסוג NEAR או FAR בשתי השפות, כדי למנוע שגיאות. קריאה לפרוצדורה שמוגדרת כ-NEAR מרחיקה את הפרמטרים בשני בתים (פקודת POP אחת). קריאה לפרוצדורה שמוגדרת כ-FAR מרחיקה את הפרמטרים בארבעה בתים (שתי פקודות POP).
- אם מעבירים מצביעים בין שתי השפות, יש לציין באופן מפורש את סוג המצביע כדי שנהיה בטוחים בגודלו על ידי ציון FAR או NEAR בשפת C. כך :

```
INT FAR *A;
```

- העברה על פי ייחוס (By Reference) מוסרת בפועל כתובת. כדי להשתמש בשיטה זו באופן תקין, מומלץ לציין את אופן ההצבעה : "INT FAR A;".
- תוכנית עד גודל של 64K (65,535) תהיה פשוטה ויעילה יותר, אם נשתמש במצביע מסוג NEAR. תוכנית גדולה יותר, או שבעתיד תהיה גדולה יותר, חייבת להיות מוגדרת כ-FAR.
- מהדר של שפת C מגדיר קריאה לפונקציה בשם "_main" באופן אוטומטי. פונקציה זו היא הראשונה להתבצע. מומלץ להשתמש בה כדי לקרוא להמשך התוכנית, אם משתמשים בפקודות שפת C בתוכנית אסמבלי.
- האסמבלר ו-DOS (שמות קבצים ושמות פקודות) אינם רגישים לגודל אות, כלומר – אינם מבדילים בין אות קטנה לגדולה, והמתכנת יכול לכתוב כרצונו). על כן כתבנו, למשל MOUSE.EXE וגם mouse.exe. כדי לעבוד באופן תקין עם מהדר של שפת C, **חובה** להגדיר 'רגישות לגודל אות' ("Case Sensitive") ויש להקפיד על כתיבה נכונה. ניתן לבצע זאת, אם נכתוב בשורת הפקודה את שם מהדר האסמבלר ואחריו הפרמטר הנכון. ניתן לברר מהו הפרמטר על ידי הקלדת שם מהדר האסמבלר ואחריו "/?" או "?" או "-?", על פי סוג המהדר.

אינדקס

א

אוגר 18 register
דגלים 18, 25, 143 flags register
אפס 143, 144 zero
גלישה 143, 152 overflow
הצגת הדגלים על ידי debug 153
זוגיות 143 parity
זכור/נשא 143, 150 carry
זכור-עזר 143 auxiliary carry
כיוון 143 direction
סוגים 143
סימן 143, 145 sign
פסיקה 143 interrupt
צעד יחיד/מלכודת 143 trap
שמירה ואחזור ערך 294
העברת נתונים בפרוצדורה 247
העתקה - MOV 28, 29
הצבה 28
ערך הקסדצימלי 40
כללי 23 general purpose
מצביע 24 pointer
מקטע 24 segment
אופרנד 27 operand
גודל 30
היעד 27 destination
המקור 27 source
אחסנה 21
אסמבלי מותנה 261 conditional assembly
הוספת קטעי קוד לניפוי שגיאות 262
הנחיות 263
התאמות ייחודיות למשתמש 263
שילוב מאקרו 270
שימוש 262
אפיק 18 bus
כתובת 18 address
נתונים 18 data

ב

בדיקה DEBUG (ניפוי), תוכנה 57, 59, 63
תוכנית שגויה 80
בית 21, 131 byte
בלוק נתונים 86
בסיס 87 base
בינארי 21, 147, 191 binary
דצימלי 21, 148 decimal
הקסדצימלי 21, 147, 194 hexadecimal

ד

דגלים, אוגר 18, 25, 143 flags register (ראה גם אוגר)
הצגת הדגלים על ידי debug 153
סוגים 143

ה

הזזה 175
היסט 87, 131 offset
הנחיה 141 directive
ASSUME 53
CODE 53
END 53
IF 261
IF1 266
IFB 266
IFDEF 263
IFDIF 266
IFE 266
IFIDN 266
IFNDEF 266
R - Register 153
RF - Register Flags 153
SEGMENT 53
START 53
הערות 27 remarks
העתקה - MOV 28, 29
העתקת נתונים 30, 83
הרצת תוכנית 57
השהייה 119 pause/delay

ז

זיכרון 22, 131 memory (ראה גם כתובת)
RAM 131
בדיקת ערכי תאים 85
הזזת ערכים 89
העתקת נתונים 30, 83
הצבת נתונים 82
חיפוש מספר 95
חיפוש נתון 190
חיפוש סיבית בתא 187
כתובת 22, 131
כתיבת נתונים 139
סגמנט 141 segment
סיכום ערכים 88
ספירת תאים 84
תנאים 97

ח

חוצץ כתובות 18 address buffer
חילוק 176
חישוב ממוצע 196

ט

טעויות בתכנות 67, 80

י

יחידה אריתמטית-לוגית 18 Arithmetic-Logic Unit - ALU
יחידת בקרה 18 control unit
ייצוג מספרים 21

כ

כתובת 22, 131 address (ראה גם זיכרון)
בסיס 131 base
התחלה 131
זיכרון 22, 131 memory
חיפוש 92
לוגית 133 logical
פיסית 133, 134 physical
פנייה באמצעות תוכנית 136

ל

לולאה 75 loop
פקודה loop 78

מ

מאפיין
159 Byte PTR
159 Word PTR
מאקרו 267 macro
אסמבלי מותנה 270
ארגומנט 269
שכפול קוד REPT 271
שכפול קוד על פי רשימה
274 IRC
273 IRP
מהדר 19 compiler
מחסנית 227 stack
בדיקה באמצעות תוכנת debug 237
דרך פעולה 233
הגדרה 228
העברת נתונים 249
מעבד 227
מתכנת 227
נתונים, הכנסה והוצאה 235
פקודה POP 229
פקודה PUSH 228
שמירת נתונים 227
מחרוזת 277 string (ראה פקודה)
מיון מספרים 197
מילה 21 word
מילה כפולה 21 Dword
מילה שמורה reserved word
106 STACK
מיסוך 162 masking
מעבד, מיקרומעבד, מיקרופרוססור 17 microprocessor
17 80x86
מערך 205, 217 array
איברים 217
הגדרה 217
היסט 220
מפענח פקודות 18 command decoder
מפרש 19 interpreter
מקטע 24 segment registers
106 STA
אילוץ מקטע prefix

מחסנית 105 stack
נתונים 208 data
קוד/תוכנית 105, 206 code
מקשים, ייצוג בינארי 191
משתנה 205 variable
גלובלי 248
הגדרה 206, 211
במקטע נתונים 208
במקטע תוכנית 206
העברת נתונים 248
כללי שימוש 210

נ

ניפוי שגיאות debug (ראה גם תוכנת debug)
בתוכנית שכוללת פסיקות 126
תוכנת ניפוי 134, 167

ס

סגמנט 141 segment
סיבית 21, 159 bit
MSB 145
בידוד 186
הזזה 175
החלפה בין ערכים 189
חיפוש בתא 187
כפל 160
סיבוב 183, 185
סיומת
asm 58
com 59
exe 59

ע

עורך, תוכנית 57 editor
עיקרון LIFO 227
ערך 22 value

פ

פסיקה 105, 129 interrupt
BIOS 122
DOS 106
int 107
אתחול המחשב 125 reset

בדיקה האם הוקש מקש 119
הצגת הודעה על המסך 108
הצגת תו בודד על המסך 113
ניפוי שגיאות בתוכנית שכוללת פסיקות 126
סיום תוכנית 106
קביעת מיקום הסמן במסך 122
קוד פסיקה/תוכנית פסיקה 107 interrupt code
קליטת מחרוזת מהמקלדת 123
קליטת מקש מהמקלדת 110
קליטת מקשים מיוחדים 120
פקודה 27, 141 command
ADD - Addition 36
AND 160, 162
CALL 243
CBW 293
cld - clear direction 286
CMP - Compare 37
CMPSB 282
CMPSW 284
CWD 293
DAA 294
DAS 294
debug 70
DEC - Decrement 36
DIV 291
IDIV 293
IMUL 293
IN 295
INC - Increment 36
Int - Interrupt 107
JB - Jump Below 155
JC - Jump Carry 156
JL - Jump Less 154
JMP - Jump 38
JS - Jump Sign 156
LODSB 277
LODSW 279
loop 78
MOV - העתקה 28, 159
MUL 289
NEG 173

37 No Operation - NOP
172 NOT
108 offset
170 OR
295 OUT
229 POP
228 PUSH
280 REP
282 REPE
285 REPNE
241 Return - RET
185 ROL
183 ROR
284 SCASB
286 SCASW
182 SHL
175 SHR
286 set direction - std
278 STOSB
279 STOSW
37 Substract - SUB
166 TEST
173 XOR
מחרוזת 277 string
יעד 277 destination
מקור 277 source
קלט/פלט 295
קפיצה מותנית 39, 158 conditional jump
תווית 27, 38 label
פרוצדורה 241
דרך פעולה 244
הגדרה 241
העברת נתונים 247
באמצעות אוגרים 247
באמצעות מחסנית 249
באמצעות משתנים 248
פקודה CALL 243
פקודה Return - RET 241
קרובה 251 near
רחוקה 251 far
תיעוד 252

ק

קבוע 257 constant
= 258
EQU 258
תפקיד 257
קו בקרה 18 control line
קישור LINK, תוכנה 57, 59

ר

רכיב מוכלל, שבב 17 chip

ש

שגיאות 67, 80
שיטת המשלים ל-2 145, 146 2's complement
שפה language
C 297
העברת פרמטרים לשפת אסמבלי 301
קישור אסמבלי 300
שילוב אסמבלי 298
אסמבלי/סף 19, 297 assembly
מכונה 19 machine
עילית 19, 297 high level

ת

תווית 27, 38 label
תוכנית
בדיקה 63
גוף 53
הרצה 57
סיום 53
פתיחה 53
תוכנית מהדר MASM 57, 59, 141
שגיאות בהרצה 67
תוכנית עורך 57 editor
תוכנת בדיקה DEBUG (ניפוי) 57, 59, 63, 126, 134, 167
בדיקת מחסנית 237
שגיאות הרצה 69
תקציר פקודות 70
תוכנת קישור LINK 57, 59, 141
תוצאה מדומה 166
תור ההוראות 18 instruction queue

ת.ד. 6108 הרצליה 46160 **הזמנות באתר**
info@hod-ami.co.il www.**hod-ami**.co.il

קטלוג אוקטובר 2013 ***הנחות ומבצעים באתר***

	כולל	ת. הוצאה	עמ'	מחיר*
אינטרנט - מפתחי אתרים/גרפיקה				
אמא, אבא - בניתי אתר באינטרנט		9/2003	256	**29**
הגדל את הכנסות העסק שלך באמצעות פרסום בגוגל Google AdWords		6/2010	300	249
מבוא לתכנות בסביבת אינטרנט – הכל כלול בספר אחד HTML + JS + ASP		3/2006	192	**9**
מבוא לתכנות בסביבת אינטרנט - מבוא ו- **HTML** - חלק 1 מתוך 3 - מהד' 3		6/2003	320	59
מבוא לתכנות בסביבת אינטרנט - **JavaScript** - חלק 2 מתוך 3		5/2002	240	49
מבוא לתכנות בסביבת אינטרנט - **ASP** - חלק 3 מתוך 3 - מהד' 3		3/2006	192	49
HTML5 המדריך לבניית אתרים, הדור הבא חדש		10/2012	352	149
HTML & CSS למפתחי אתרים באינטרנט - מהד' 5		4/2005	592	**79**
The **Java** Tutorial סדנת לימוד	CD	7/2001	768	179
JavaScript סדנת לימוד		2001	586	159
מדריך **ASP.NET MVC 4** חדש		7/2013	514	199
ASP.NET 3.5 סדנת לימוד בשפות C# ו-VB		10/2009	824	**99**
תכנות				
Code Complete – מדריך מעשי לפיתוח תוכנה (ברכישה ישירה) חדש		10/2013	288	139
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3 (ברכישה ישירה)		7/2011	350	169
Visual C# 3.0 סדנת לימוד		9/2008	656	**99**
ללמוד **C** - מהד' 3		2/2001	480	139
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 3 (ברכישה ישירה)		2012	152	95
PC - חומרה, תוכנה ורשתות				
מדריך Hacking ואבטחת מידע, מהד' 2		9/2011	428	169
מדריך **חומרה ותוכנה** לטכנאי PC - מהד' 5 כולל עדכון 2011	CD	2/2011	752	189
Windows 7/8 נושאים מתקדמים (ברכישה ישירה. המחיר כולל משלוח)		חדש	140	49
מדריך **רשתות** לטכנאי PC ולמנהלי רשת - מהד' 4		4/2007	608	219
Windows				
Windows 8 מדריך למשתמש חדש		7/2013	438	129
Windows 7 צעד-אחר-צעד		1/2010	272	**39**
Windows XP והכרת המחשב האישי למתחילים		3/2003	208	**19**
גרפיקה				
Flash – ספר הדרכה ותרגילים (ברכישה ישירה)		2012	122	64
Illusatrator – ספר הדרכה ותרגילים (ברכישה ישירה)		2012	120	64
Photoshop צעד אחר צעד (צבע מלא, למתחילים), מהד' 3		3/2012	200	159
מדריך לתוכנת העיצוב והאנימציה **3ds max** (2 כרכים) (ברכישה ישירה)	CD	מהד' 5	1400	419

* מחיר מומלץ לצרכן כולל מע"מ (המחירים המודגשים במחיר מיוחד בהזמנה ישירה).

יש להיכנס לאתר כדי לבדוק מבצעים ומחירים מיוחדים

היכנס לאתר להתעדכן בספרים החדשים ומבצעים www.**hod-ami**.co.il

	כולל	הוצאה	עמ'	מחיר*
OFFICE 2010				
טבלאות ציר – ניתוח נתונים חכם (ברכישה ישירה)		2012	116	87
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 3 (ברכישה ישירה)		2012	152	95
Word 2010 צעד אחר צעד		11/2010	160	69
Excel 2010 סדנת לימוד		11/2010	344	129
PowerPoint 2010 צעד אחר צעד		11/2010	202	69
Outlook 2010 צעד אחר צעד		11/2010	190	69
Access 2010 סדנת לימוד		11/2010	360	179
OFFICE 2007				
PowerPoint 2007 למתחילים		8/2007	240	99
Outlook 2007 למתחילים		1/2008	224	**19**
OFFICE 2003				
Word 2003 צעד אחר צעד		3/2004	208	**9**
Excel 2003 הסדרה הידידותית למתחילים		2/2004	160	49
PowerPoint 2003 הסדרה הידידותית למתחילים		2/2004	96	**9**
Outlook 2003 הסדרה הידידותית למתחילים		2/2004	96	**9**
Access 2003 הסדרה הידידותית למתחילים		2/2004	144	**9**
OFFICE XP				
Office XP צעד אחר צעד	CD	7/2001	576	**19**
Word XP הסדרה הידידותית למתחילים		7/2001	144	**9**
ניהול, כלכלה ושונות				
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3 (ברכישה ישירה)		7/2011	350	169
חקר ביצועים כשירות ללקוח		2/2010	236	92
ניהול ממוקד לעשות יותר עם מה שיש (כריכה קשה) - מהד' 4		9/2012	358	133
לי זה עולה יותר (תמחיר) (כריכה קשה) - מהד' 3		10/2006	368	129
מערכות מידע				
Oracle SQL יכולות מתקדמות (ברכישה ישירה)		7/2011	626	249
SAS (**Statistical Analysis System**) – ספר לימוד (ברכישה ישירה)		11/2010	256	169
בסיסי נתונים ושפת **SQL** – עקרונות ועיצוב (ברכישה ישירה)			648	149
ניתוח מערכות מידע כולל מתודולוגיית ה- **UML** (ברכישה ישירה)		11/2004	818	229
המדריך העברי השלם **UML** (ברכישה ישירה)		1/2009	346	329

* מחיר מומלץ לצרכן כולל מע"מ. קטלוג 10/2013 (המחירים המודגשים במחיר מיוחד ברכישה ישירה).

יש להיכנס לאתר כדי לבדוק מבצעים ומחירים מיוחדים

היכנס לאתר להתעדכן בספרים החדשים ומבצעים

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il