```
CMP EDI,[BX]
CMP DWORD PTR [EBX],2222222
```

<u>פקודות בקרה</u>

פקודות שמשנות את הפקודה הבאה לביצוע.

CALL - בקרה לרוטינה - קפיצה עם אכסון ה-IP במחסנית (ואפשר גם ה-CS).

RET - הפוך ל-CALL.

INT - הסתעפות לפסיקת תוכנה.

על אלה נרחיב יותר מאוחר.

<u>פקודות קפיצה</u>

JMP - קפיצה בילתי מותנית. ניתו להגדיר SHORT ליעד המרוחק עד 127+ ... 128- בתים. אין טעם לציין SHORT בקפיצות אחורה, כי אם זה אפשרי אז TURBO ASSEMBLER יעשה זאת אוטומטית.

ניתן לבצע קפיצה רחוקה ליעד בטווח 32768- 32767+ בתים - אם התוית נמצאית באותו סגמנט קוד. כמו כן ניתן לכפות קפיצה רחוקה ע"י JMP FAR label גם אם label נמצא קרוב.

ניתן <u>לקפוץ עקיף</u>: JMP AX או JMP [JumpTargetPtr].

<u>פקודת לולאה LOOP</u>: פקודה הגורמת לחיסור CX ולקפיצה לתוית אם CX לא התאפס כתוצאה מההפחתה.

LOOPE/Z - כמו LOOP אלא שסיום הלולאה היא כש- CX שווה 0 או דגל ה-ZF = 0 ברגע ביצוע הפקודה, קודם לכן.

LOOPNE/NZ - כמו LOOPE/Z רק שהתנאי הנוסף הוא ZF == 1.

JCXZ - קפיצה רק אם CX==0.

ב-8086 כל פקודות הקפיצה (למעט JMP) - ליעד בטווח 128- 127+ בתים.
ב-386 ואילך יש גירסאות ליעד בטווח 32768- 32767+ בתים.

<u>פקודות קפיצה מותנות</u>

... JE, JNE, JG, JGE, JC

בדרך כל השימוש שלהם הוא בשילוב עם הפקודה CMP:
לדוגמא

```
CMP AX,BX
JE label
```

משמעותו "קפוץ רק אם AX שווה ל-BX". בהקשר הספציפי <u>הזה</u> של CMP AX,BX המשמעות של הפקודות הבאות הם:

JE - קפוץ במקרה של AX == BX.
JNE - קפוץ במקרה של AX != BX.

JG - קפוץ אם תכני BX < AX כמספרים עם סימן

JGE - קפוץ אם תכני BX ≤ AX כמספרים עם סימן

JLE - קפוץ אם תכני BX ≥ AX כמספרים עם סימן

JL - קפוץ אם תכני BX > AX כמספרים עם סימן

JA - קפוץ אם תכני BX < AX כמספרים חסרי סימן

JAE - קפוץ אם תכני BX ≤ AX כמספרים חסרי סימן

JBE - קפוץ אם תכני BX ≥ AX כמספרים חסרי סימן

JB - קפוץ אם תכני BX > AX כמספרים חסרי סימן

פעולות ביטיות

פקודות הפועלות על כל ביט של האופרנד או אופרנדים לחוד.

פקודות 2 אופרנדים XOR, OR, AND, TEST.
פקודת 1 אופרנד NOT.

כל הפקודות פועלות על אוגרים וזכרון 8, 16 ביט (32 ביט ב-386 ואילך) בדומה לפקודות ADD, SUB וכו'.

לדוגמא, הפקודה

```
AND AL,BL
```

תבצע Bitwise AND על הביטים שך AL ו-BL. אם לדוגמא התכנים של AL ו-BL לפני הפקודה היו:

```
AL =  0 1 0 0 1 1 0 0
BL =  1 1 0 0 0 1 0 1
```

לאחר ביצוע הפקודה התוכן של AL יהיה

```
AL =  0 1 0 0 0 1 0 0
```

שכן רק בביטים בשלישי והשביעי של שני האופרנדים ישנו אחד.

עם אותם תכנים הפקודה

```
OR AL,BL
```

תבצע Bitwise OR על שני האופרנדים כלומר תציב ל-AL את הערך

```
AL =  1 1 0 0 1 1 0 1
```

הפקודה

```
XOR AL,BL
```

תבצע Bitwise XOR על שני האופרנדים כלומר תציב ל-AL את הערך

```
AL =  1 0 0 0 1 0 0 1
```

NOT - תבצע הפוך סיביות לדוגמא

```
NOT AL
```

תציב לערך הקודם של

```
AL = 0 1 0 0 1 1 0 0
```

את הערך

```
AL =  1 0 1 1 0 0 1 1
```

TEST - מבצע AND מבלי לשנות תוכן האופרנדים - רק הדגלים מושפעים.

הזזה וסיבוב SHIFT, ROTATE

פקודות המזיזות את הביטים ימינה או שמאלה בתוך אופרנד שיכול להיות אוגר או זכרון 8 או 16 ביט (32 ביט ב-386 ואילך).

SHL dest,source - הזזה לוגית

הזז את תוכן dest שמאלה במספר הסיביות המצוין ב-source.
הביט המשמעותי ביותר נכנס ל-carry ומשם "הולך לאבוד".
לתוך הביטים הנמוכים נכנס בכל מקרה 0.

ניתן להזיז שמאלה במספר קבוע כמו:
SHL DL,1 - הזזה רק באחד.

CF	DL
1	1 0 0 0 0 0 0 1

עובר ל-

CF	DL
1	0 0 0 0 0 0 1 0

ניתן להזיז לפי ערך טעון ב-CL למשל:

```
MOV CX,4
SHL DX,CL
```

- SHR dest,source

כמו SHL אבל הזזה ימינה במקום שמאלה.
הביטים הנמוכים נאבדים, העליונים מתאפסים.

SHR DL,1 - הזזה רק באחד.

CF	DL
1	1 0 0 0 0 0 0 1

עובר ל-

CF	DL
1	0 1 0 0 0 0 0 0

- SAL dest,source
זהה ל- SHL.

הזזה <u>אריטמטית</u>

- SAR dest,source

דומה ל- SHR אך <u>משמשר סימן</u>. הביט המשמעותי ביותר הוא הערך המוזז לתוך הביטים העליונים, במקום פשוט 0 ב-SHL.

SAR DL,1 - הזזה רק באחד.

CF	DL
0	1 0 0 0 0 0 0 1

עובר ל-

אילו בביט הימני היה 0 - רק ה-CF היה משתנה ל-0.

CF	DL
1	1 1 0 0 0 0 0 0

4 פעולות סיבוביות.

ROR - כמו SHR אלא שהביט הפחות משמעותי עובר לתוך הביט המשמעותי ביותר וגם לתוך ה-CF.

ROR DL,1 - הזזה רק באחד.

CF	DL
0	1 0 0 0 0 0 0 1

עובר ל-

CF	DL
1	1 1 0 0 0 0 0 0

ROL - כמו ROR אלא שהכוון הוא שמאלה.

RCR - כמו ROR אלא שה-CF הוא חלק מהאופרנד. הביט הפחות משמעותי עובר לתוך ה-CF וערך ה-CF עובר לתוך הביט המשמעותי ביותר.

RCR DL,1 - הזזה רק באחד.

CF	DL
0	1 0 0 0 0 0 0 1

עובר ל-

CF	DL
1	0 1 0 0 0 0 0 0

RCL - כמו RCR אלא שהכוון הוא שמאלה.

הערה:

תחת ה-8086 אם נכתבה פקודה נוסך

```
SHL AX,3
```

האסמבלר יפרוש את הפקודה SHL AX,1 שלוש פעמים. כנ"ל לפקודות הסיבוביות האחרות. כלומר יש כביכול פקודה שבו אופרנד המקור הוא קבוע אבל למעשה זו לא פקודת מכונה.

386 ואילך

כאן יש תמיכה של פקודות סיבוביות עם אופרנד מקור קבוע ברמה של פקודת מכונה. כמו כן הפקודות הסיבוביות יכולות להיות על אוגרים וזכרון 32 ביט.

פקודות או פעולות מחרוזת הם למעשה מימוש מהיר של פעולות קריאה, השמה והשוואה למעשה על מערכים של מספרים בגודל 1,2 בתים (4 בתים ב-386 ואילך).

המאפין של כל הפקודות הללו היא שצריך לאתחל את הפיונטרים DS:SI או ES:DI או שניהם לכתובת של מערך (תחילתו או סופו). כל ביצוע של כל אחד מהפקודות הללו, לצד פקודת ההשמה/השוואה, מקדם או מפחית באופן אוטומטי את אוגר ההיסט (SI או DI). ההפחתה / קידום הוא ביחידות 1 או 2 (4 ב-386 אילך).

החיבור/חיסור נקבע ע"י דגל ה-DF שערכו ניתן לקבוע ע"י פקודות STD, CLD.

CLD מציב 0 ל-DF.

STD מציב 1 ל-DF.

במידה ו-DF שווה ל-0 מתבצע קידום אוטומטי.
במידה ו-DF שווה ל-1 מתבצע הפחתה אוטומטית.

LODS B/W - טעינת בית או מילה מהזכרון לצובר AX/AL.

LODSB - טעינת בית ממוען ע"י DS:SI ל-AL וקידום או הפחתה ב-1 של SI.

LODSW - טעינת מילה ממוענת ע"י DS:SI ל-AX וקידום או הפחתה ב-2 של SI.
החיבור/חיסור נקבע ע"י ה- DF שערכו ניתן ע"י פקודות STD, CLD.

לדוגמא

```
Word_Arr DW 2,4,6
         ....

CLD
MOV SI,OFFSET Word_Arr+2
LODSW
```

יטען את המילה השניה במערך Word_Arr לתוך AX (AX = 4).

כמו כן SI = SI - 2.

על פי רוב זה יהיה בתוך חוג המצע פעולה בתוך מערך דרך AX.

STOS B/W - טעינת בית או מילה מצובר AX/AL לזכרון.

STOSB - אחסון בית ב-AL לכתובת ממוענת ע"י ES:DI וקידום או הפחתה ב-1

של DI.

STOSW - אחסון מילה ב-AX לכתובת ממוענת ע"י ES:DI וקידום או הפחתה ב-2 של DI.

לדוגמא

```
Word_Arr DW 2,4,6
         ....

CLD
MOV AX,8
MOV DI,OFFSET Word_Arr+2
STOSW
```

ישנה את המילה השניה ב-Word_Arr ל-8.

אופציה REP

REP - גורם לפקודת מחרוזת לחזור על עצמה כמספר הפעמים שהוא הערך של CX.
CX מופחת ב-1 בכל ביציע - כמו ב-LOOP.

שימוש ברישא REP לפקודת STOSB או STOSW יגרום לחזרה על הפקודה CX פעמים (שבסופה CX יכיל את המספר אפס). זו הדוגמה הראשונה לאופצית REP שלמעשה מהווה מימוש חסכוני של לולאה פשוטה.

לדוגמא, נניח שיש לנו בשטח המידע הגדרה

```
W_array DW 1,2,3,4,5,6,7,8
```

ניתן להציב את הערך 1000 לכל איברי המערך בדרך הבאה:

```
MOV DI,OFFSET W_array
MOV BX,SEG W_array
MOV CX,8
MOV AX,1000
MOV ES,BX
CLD
REP STOSW
```

דוגמא: פרוצדורת העתקת מחרוזות נוסח שפת C.

(כלומר תו אחרון אפס)

בהנחה ש-DS:SI מועברים כפרמטרים כמחרוזת מקור,
ו-ES:DI מועברים כפרמטרים כמחרוזת יעד.

```
CopyString          PROC  FAR
                    CLD
CopyStringLoop:
                    LODSB                   קרא מהראשון
                    STOSB                   כתוב לשני
                    CMP AL,0
                    JNZ CopyStringLoop          בדוק אפס
                    RET
CopyString          ENDP
```

MOVS - מעין שילוב של LODS ו-STOS (אבל בלי לערב את AL/AX). קריאת בית/מילה מ-DS:SI והעתקתו ל-ES:DI.
מקדמת/מפחיתה את SI ו-DI בהתאם לדגל הכיוון והוריאנת של הפקודה (B/W). אורך הפקודה בית אחד.

MOVS B/W - טעינת בית או מילה מצובר AX/AL לזכרון.

MOVSB - העבר בית במהכתובת ממוענת ע"י DS:SI לכתובת הממוענת ע"י ES:DI וקידום או הפחתה ב-1 של DI ו-SI.

MOVSW - העבר בית במהכתובת ממוענת ע"י DS:SI לכתובת הממוענת ע"י ES:DI וקידום או הפחתה ב-2 של DI ו-SI.

אופצית REP

בהנחה ש-DS:SI מצביע למערך בתים בגודל 10 שאותו רוצים להעתיק למערך שכתובתו ב-ES:DI, ניתן לבצע:

```
          MOV CX,10
          CLD
CopyLoop:
          MOVSB
          LOOP CopyLoop
```

אפשרות פשוטה יותר - בעזרת ה-prefix REP ואז מקבלים אותו אפקט ע"י הפקודה

```
MOV CX,10
CLD
REP MOVSB
```

פקודות SCAS, CMPS

B
/
SCAS - בדיקת התאמה/אי התאמה של בית/מילה.
\
W

השוואת בית/מילה בצובר AX/AL לתוכן כתובת זכרון ES:DI וקידום/הפחתה של DI.

קיימת אפשרות קידום ע"י REPE או REPNE.

REPE או REPZ - הקידום הוא כל עוד יש שויון.

REPNE או REPNZ - הקידום הוא כל עוד יש אי שויון.

מאפשר לממש רעיונות כמו חיפוש תו מסוים.

B
/
CMPS - בדיקת התאמה/אי התאמה של שתי מחרוזות בזכרון.
\
W

שתי המחרוזות נמצאות בכתובות DS:SI ו-ES:DI.
יתבצע קידום/הפחתה של SI ו-DI.

קיימת אפשרות קידום ע"י REPE או REPNE.

מאפשר לממש רעיונות כמו השוואת מחרוזות או מציאת תו לא שווה ראשון.

386 ואילך

ישנם פקודות

LODSD

STOSD

MOVSD

SCASD

CMPSD

עם מבנה לוגי זהה לפקודות הקודמות של ה-8086 אך הפעולה היא בנפח 4 בתים ומוצבעת ע"י האוגרים ESI ו-EDI לפי המקרה. גירסאות ה-REP שך הפקודות הללו חוזרות על עצמן לפי התוכן של ECX.

לפיכך:

LODSD - טעינה מילה כפולה ב-EAX מלכתובת ממוענת ע"י DS:ESI וקידום או הפחתה ב-4 של ESI.

STOSD - אחסון מילה כפולה מ-EAX לכתובת ממוענת ע"י ES:EDI וקידום או הפחתה ב-4 של EDI.

MOVSD - העבר מילה כפולה במהכתובת ממוענת ע"י DS:ESI לכתובת הממוענת ע"י ES:EDI וקידום או הפחתה ב-4 של EDI ו-ESI.

SCASD - השוואת מילה כפולה בצובר EAX לתוכן כתובת זכרון ES:EDI וקידום/הפחתה של EDI ב-4.

CMPSD - בדיקת התאמה/אי התאמה של שתי מילים כפולות בזכרון ביעדים DS:[ESI] ו-ES:[EDI] וקידום אוטמטי של ESI ו-EDI ב-4.

גורסאות ה-REP של הפקודות LODSD, STOSD, MOVSD ו-REPE, REPNE עבור SCASD ו-CMPSD - משמעותם חזור על הפקודה תוכן ECX פעמים.