פקודות מכונה מסוימות

פרק זה הוא מבוא לקבוצה של פקודות מכונה בסיסיות יחסית, שהכרתן חיונית לכתיבת תוכניות ראשוניות. פקודות שאינן מופיעות כאן נוטות להיות יחודיות יותר, וההכרה והשימוש בהן הוא נושא יותר מתקדם. בחלק מהפקודות נרחיב יותר מאוחר.

מושגים נוספים שנפרט מאוחר יותר הוא מושג **אוגר הדגלים** והביטים הפעילים שלו, הנקראים **דגלים**. בשלב זה נחוץ רק להבין שמדובר באחד מאוגרי ה-CPU וחלק מהביטים שלו.

עובדה נוספת שיש צורך לדעת הוא שהאוגרים 16 ביט AX, BX, CX ו-DX מתחלקים לתתי אוגרים 8 ביט הנקראים AL,AH,BL,BH,CL,CH,DL,DH בהתאמה. גם על כך נרחיב בשלב מאוחר יותר.

פקודות העברת נתונים

3 סוגים

1. העברה מאוגר לאוגר או בין זכרון לאוגר.
2. העברה מ/אל המחסנית.
3. העברת גושי בתים ממקום אחד בזכרון למשנהו - סוג זה ידון בפעולות מחרוזות.

על האפשריות השונות של התיחסות לזכרון נרחיב בשלב מאוחר יותר. נציין כאן רק שכאשר אופרנד מוקף סוגריים מרובעות [] או הינו שם משתנה מדובר בהתיחסות לזכרון.

העברה מאוגר לאוגר או בין זכרון לאוגר

הפקודה MOV

מבנה כללי (אופיני לכמעט כל הפקודות של שני אופרנדים):

```
MOV Target_operand, Source_operand
```

כאשר Target_operand יכול להיות אוגר או זכרון ו- Source_operand יכול להיות אוגר, זכרון או קבוע בהגבלה אחת: אין תמיכה בהעברה של זכרון לזכרון (כלומר אי אפשר שגם Target_operand ו- Source_operand הם יעדים בזכרון. שני האופרנדים צריכים להיות באותו גודל (8 ביט, 16 ביט ו386 ואילך גם 32 ביט).

דוגמאות:

```
MOV AX,0        קבועים - אוגר
MOV BX,9
MOV AX,BX       אוגר - אוגר
```

```
MOV AL,1

MOV BX,[DI]          זכרון - אוגר

MOV [BX],AX          אוגר - זכרון

MOV [TestChar],'A'
                     או
MOV TestChar,'A'
```

כאשר מוגדר
בשטח המידע TestChar DB ?

```
MOV   WORD PTR [BX],1

MOV   BYTE PTR [BX],1
     (MOV   [BX],1 כי מהמשפט
              (לא ניתן לקבוע גודל היעד
```

386 ואילך

אופרנדי היעד יכולים להיות אוגרים או זכרון 32 ביט (... EAX, EBX) ואופרנדי המקור יכולים להיות אוגרים או זכרון או קבועים 32 ביט (בהגבלה הקודמת ששני האופרנדים אינם יכולים להיות בו זמנית התיחסיות לזכרון).

דוגמאות:

```
MOV ESI,EDX
MOV DWORD PTR [ESI],99000
MOV EAX,[BX]
```

אוגרי הסגמנט כאופרנד

אוגרי הסגמנט (CS,DS,SS,ES) אינם יכולים להיות אופרנדים ברוב פקודות המכונה (כמו ADD או INC ..). אחת הפקודות המעטות שהם כן יכולים להופיע הינה פקודת MOV. כל אוגרי סגמנט יכולים להיות אופרנד מקור וכולם חוץ מ-CS יכולים להיות אופרנדי יעד אולם אין תמיכה בהעברת מידע מאוגר סגמנט אחד לשני. האופרנד השני שאיננו אוגר סגמנט יכול להיות אוגר כללי 16 ביט או זכרון. אין תמיכה בהצבת קבוע לתוך אוגר סגמנט.

דוגמאות:

```
MOV AX,CS
MOV DS,[BX]
MOV [SI],ES
MOV DS,SI
```

386 ואילך

ב-386 ואילך יש עוד שני אוגרי סגמנט אפשריים FS ו-GS. האופרנד הזכרון יכול להיות היסט 32 ביט (על כך נרחיב בהמשך). מעבר לכך הכללים זהים ל-8086.

XCHG - החלפת ערכי זוג אופרנדים.

פקודה 2 - אופרנדים שבו ערכי שני אופרנדים מתחלפים. פקודה זו יוצאת דופן כפקודת 2 אופרנדים במובן הזה ששני האופרנדים מחליפים ערכים (בדרך כלל רק האופרנד השמאלי מחליף ערך). נוסף לכך האופרנד הימני לא יכול להיות קבוע. כרגיל, אין תמיכה במצב ששני האופנדים הם התיחסיות לזכרון.

דוגמאות:

```
XCHG AX,BX

XCHG DX,[BX]
```

386 ואילך

האופרנדים יכולים להיות גם אוגרים או זכרון 32 ביט (אבל כרגיל לא שניהם זכרון). דוגמאות:

```
XCHG EAX,EBX

XCHG EDX,[BX]
```

העברת אינפורמציה למחסנית

על הפקודות האלו נרחיב בהמשך. בשלב זה נציין רק מבנה הפקודות הבסיסיות:

הפקודה

```
PUSH Op16
```

דחיפת אופרנד 16 ביט למחסנית. Op16 יכול להיות כל אחד מאוגרי ה-CPU (AX, BX, ...) למעט IP ו-FLAGS בכלל זה אוגרי הסגמנטים (CS,DS,ES,SS). Op16 יכול להיות גם אופרנד זכרון וברמת שפת האסמבלי גם קבוע אם כי למעשה ב-8086 אין פקודת מכונה כזו אלא שהאסמבלר ממש אותה בעקיפין.

דוגמאות:

```
PUSH AX
PUSH DS
PUSH Var1
PUSH WORD PTR [BX]
PUSH 12000
```

הפקודה

```
POP Op16
```

פקודה השולפת אופרנד מהמחסנית ומציבה אותו ל-Op16. כאן Op16 יכול להיות אחד מאוגרי ה-CPU למעט IP, FLAGS ו-CS. הוא יכול להיות גם התיחסות לזכרון 16 ביט אבל כמובן שאינו יכול להות קבוע.

דוגמאות:

```
POP BX
POP ES
POP Var1
POP WORD PTR [SI]
```

הפקודה

```
PUSHF
```

פקודה ללא אופרנדים המבצע דחיפה של אוגר הדגלים למחסנית.

הפקודה

```
POPF
```

פקודה ללא אופרנדים המבצעת שליפה של 16 ביט מהמחסנית והצבתם לתוך אוגר הדגלים.

<u>386 ואילך</u>

כאן יש תמיכה גם בביצוע PUSH ו-POP לאוגרים הכלליים והתיחסיות לזכרון 32 ביט, פקודות PUSH ו-POP לאוגרי הסגמנטים הנוספים FS ו-GS, ופקודות PUSHDF ו-POPDF לאוגר הדגלים המורחב EFLAGS. כמו כן ביצוע PUSH לקבוע נתמך כאן ברמת פקודת מכונה (ולא מדומה ע"י האסמבלר כמו ב-8086) בכלל זה קבעים 32 ביט.

דוגמאות:

```
PUSH EAX
PUSH DWORD PTR [BX]
PUSHDF
POPDF
PUSH FS
PUSH 1000000
```

<u>פעולות אריטמטיות -</u> לא כולל פקודות מעבד מתמטי

1. חיבור
2. חיסור
3. כפל
4. חילוק
5. השוואה

פקודות חיבור (ADD, ADC) וחיסור (SUB, SBB) הם פקודות 2 אופרנדים דומות ל-MOV בכל הקשור למספר האופרנדים שהם יכולים לקבל. INC ו-DEC הם פקודות של 1 אופרנד.

<u>חיבור</u>

חיבור	ADD
חיבור עם carry	ADC
קידום באחד	INC

(INC BX תופס בית אחד, לעומת 3 שתופס ADD BX,1).

למשל חישוב AX = AX + BX

```
ADD AX,BX
```

ננית

```
.DATA

Var1 DW 100
Var2 DW 130
Var3 DW  ?
```

למשל חישוב Var3 = Var1 + Var2

```
MOV AX,Var1
ADD AX,Var2
MOV Var3,AX
```

ננית מצב

```
Var4 DD 71234
Var5 DD 45678
```

ננית שאנחנו רוצים לחשב Var4 = Var4 + Var5 רק באוגרים של ה-8086 (מבלי להשתמש באוגרים 32 ביט של ה-386) אזי הדרך לעשות זאת תהיה

```
MOV AX,WORD PTR Var5    ;          טען החלק המשמעותי פחות של Var4
ADD WORD PTR Var4,AX    ; Var5 ו- Var4 סכם החלקים המעותיים פחות של
MOV AX,WORD PTR Var5+2  ;          טען את החלק המשמעותי יותר של Var4
ADC WORD PTR Var4+2,AX  ; סכם את החלקים המשמעותים יותר, תוך התחשבות
                        ;                                  ב- carry
```

הערות:

- AX יכול היה להיות מוחלף בכל אוגר כללי 16 ביט אחר (BX, CX, DX, SI ...)

חישוב Var5 = Var4 + Var5 תוך שימוש באוגרי ה-32 ביט ב-386:

```
MOV EAX,Var5
ADD Var4,EAX
```

הפקודה INC מקדמת אופרנד יחד ב-1. האופרנד יכול להיות אוגר או זכרון, לדוגמא

```
INC AL
INC BX
INC Var1
INC BYTE PTR [SI]
INC WORD PTR [BX]
```

386 ואילך

```
INC ECX
INC Var5
INC  DWORD PTR [BX]
```

חיסור

מתנהל באופן עקרוני כמו חיבור.

SUB	חיסור
SBB	חיסור עם carry (borrow)
DEC	הפחתה באחד

למשל חישוב Var3 = Var1 - Var2

```
MOV AX,Var1
SUB AX,Var2
MOV Var3,AX
```

חישוב Var4 = Var4 - Var5 רק באוגרים של ה-8086 (מבלי להשתמש באוגרים 32 ביט של ה-386) אזי הדרך לעשות זאת תהיה

```
MOV AX,WORD PTR Var5    ;              טען החלק המשמעותי פחות של Var4
SUB WORD PTR Var4,AX    ; חשב הפרש החלקים המשמותיים פחות של Var4 ו-Var5
MOV AX,BYTE PTR Var5+2  ;              טען את החלק המשמעותי יותר של Var4
SBB WORD PTR Var4+2,AX  ;    חשב את ההפרש את החלקים המשמעותים יותר, תוך
                        ;                          התחשבות ב-Borrow
```

חישוב Var5 = Var4 - Var5 תוך שימוש באוגרי ה-32 ביט ב-386:

```
MOV EAX,Var4
SUB Var5,EAX
```

הפקודה DEC מקדמת אופרנד יחד ב-1. האופרנד יכול להיות אוגר או זכרון, לדוגמא

```
DEC AL
DEC BX
DEC Var1
DEC BYTE PTR [SI]
DEC WORD PTR [BX]
```

386 ואילך

```
DEC ECX
DEC Var5
DEC  DWORD PTR [BX]
```

היפוך סימן NEG

לדוגמא,

```
NEG Var1
```

אם ב-Var1 היה 100 אזי עכשיו יהיה בו 100-.

```
NEG BX
```

כפל מספרים חסרי סימן MUL
כפל מתחשב בסימן IMUL

כפל וחילוק נעשים תמיד מערב את AX, או את DX,AX או (386 ואילך) EDX,EAX.

באופן עקרוני, כאשר מכפילים 2 אופרנדים בגודל n בתים, תוצאת הכפל היא היא בגודל כפול (2n בתים) כאשר n = 1,2,4 בתים.

ב-8086 גורם אחד של הכפל הוא AL או AX, ואילו הגורם השני הוא אופרנד יחיד של פקודת הכפל, שיכול להיות אוגר (בית או 2 בתים) או זכרון (בית או 2 בתים). הגודל של האופרנד קובע את אופי הפעולה שבה מדובר. לדוגמא,

```
MUL CL  או  IMUL CX
                    פירושם
AX = AL * CL

MUL BX   או  IMUL BX
                    פירושם
DX:AX = AX * CX
```

דוגמאות לכפל עם זכרון

```
N1 DB ?
N2 DW ?

MUL N1 או IMUL N1
                    פירושם
AX = AL * N1

MUL BYTE PTR [BX]   או  IMUL BYTE PTR [BX]
                    פירושם
AX = AL * [BX]   בית

MUL WORD PTR [BX]   או  IMUL WORD PTR [BX]
                    פירושם
DX:AX = AX * [BX]  מילה
```

קרי: הכפלה של אופרנד של בית אחד פירושו ביצוע של כפל של AL עם האופרנד ושמירת התוצאה בתוך AX.
הכפלה של אופרנד של שני בתים פירושו ביצוע של כפל של AX עם האופרנד ושמירת התוצאה בתוך שני האוגרים AX ו-DX, כאשר AX מקבל את החלק המשמעותי פחות ו-DX מקבל את המשמעותי יותר.
העובדה שהתוצאה מאוכסנת בגודל כפול מהאופרנדים אינו צריך להפתיע לפחות במובן הזה שתוצאת כפל של שני מספרים בגודל מסוים אכן יכולה להיות בגודל שהוא פי שניים מגודל האופרנדים. למשל עבור אופרנדים בני בית אחד, הגודל המירבי של שני האופרנדים (בהנחה שהם מתפרשים כחסרי סימן) הוא 255. תוצאת הכפל המירבית היא איפוא 65025 = 255*255, המחייבת יצוג של 16 ביט אבל אין צורך ביתר מכך (במספרים חסרי סימן). במידה ותוצאת הכפל חורג מגודל האופרנדים הדבר משתקף בדגלים CF ו-OF.

386 ואילך

כאן יש תמיכה נוספת להכפלה של מספרים 32 באופן דומה לקודמים, למשל

```
MUL EDX או IMUL EDX
            פירושם
```

EDX:EAX = EAX * EDX

MUL DWORD PTR [SI] או IMUL DWORD PTR [SI]
פירושם
EDX:EAX = EAX * [SI] מילה כפולה

הפקודה IMUL

ב-386 ואילך מבין פקודות הכפל/חילוק יש יוצא דופן, לפקודה IMUL יש גירסת 2 אופרנדים, אבל רק אוגרים 32 ביט (אי אפשר אופרנדי זכרון). לדוגמא

IMUL ESI,EDI
פירושו
ESI = ESI*EDI

בנגוד לגרסאות הקודמות של IMUL, הגרסה הזאת של IMUL לא מאפשרת להתמודד עם תוצאת כפלים החורגים מגודל האופרנדים המקוריים.

<u>חילוק</u>

חילוק בלי בסימן DIV
חילוק מתחשב בסימן IDIV

<u>התמרה מ-byte ל-word</u>

CBW המשך את ביט הסימן ב-AL לאורך AH[0] - AH[7]

CWD המשך את ביט הסימן ב-AX לאורך האוגר DX

במחשב הזה חלוקה של מספרים שלמים במעבד הרגיל נעשה תמיד תוך מערבות של האקומולטור (AX, EAX) ובדרך כלל גם אוגר ה-DATA (DX, EDX) הקובעים את המונה. פקודות החלוקה הן פקודות אופרנד אחד, היכול להיות אוגר או כתובת בזכרון, והאופרנד הזה הוא תמיד המכנה. המונה תמיד גדול פי שניים מהמכנה, והגדלים נקבעים לפי גודל האופרנד. לפיכך אם הפקודה היא

```
DIV Op8
```

כאשר Op8 הוא אפרנד 8 ביט, למשל

```
DIV  BL
```

או

```
DIV  BYTE PTR [BX]
```

אזי משמעות הפקודה היא ביצוע AX / Op8 כאשר המנה ללא שארית תוצב ב-AL ושארית החלוקה ב-AH. בדוגמא

```
DIV BL
```

משמעות הדבר הוא שמתבצע AX / BL ותוצאת החלוקה תוצב ב-AL והשארית תוצב ב-AH.

למשל בקטע הקוד:

```
MOV AX,1003
MOV BL,4
DIV BL
```

יוצב ל-AL הערך 250 ול-AH יוצב 3.

הגירסאות של חלוקת מספרים גדולים יותר:

```
DIV Op16
```

כאשר Op16 הוא אוגר או זכרון 16 ביט פירושו AX = DX:EX / Op16 ושארית החלוקה מוצבת לתוך DX.

חשוב לזכור שפקודות החלוקה תמיד מחלקים מספר בגודל כפול מהמחלק ותמיד יש להכין את המספר הזה. לדוגמא הפקודה

```
DIV BX
```

מחלק את DX:AX כמספר 32 ביט ב-BX ולא רק את AX כפי שרבים טועים. אם אנחנו רוצים לחלק שני מספרים חסרי סימן 16 ביט, נניח משתנים Var1 ו-Var2 אנחנו צריכים לעשות זאת בצורה הבאה:

```
   .DATA
Var1 DW 5000
Var2 DW 1300

   .....

   .CODE

   MOV AX,Var1
   MOV DX,0
   DIV Var2
```

לתוך AX יכנס 3 ולתוך DX יכנס 1100. שים לב לאיפוס של DX. אם האיפוס הזה לא נעשה ו-DX מכיל משהו הערך שלו יובא בחשבון בחלוקה כחצי משמעותי של המספר המחולק גם אם כותב התוכנית לא מעונין בזאת. לפיכך בכל מקרה של תכנות חלוקה בין שני מספרים באותו גודל יש להתמיר את המונה למספר בגודל כפול. במקרה של חלוקה של מספרים חסרי סימן ב-DIV פירוש הדבר איפוס החלק העליון של ההמספר המחולק: AH במקרה של חלוקה ב-8 ביט ו-DX במקרה של חלוקה באופרנד 16 ביט. במקרה של מספרים עם סימן פירוש הדבר התמרה של המספר תוך התחשבות בסימן. יש פקודות מיוחדות לכך:

```
CBW  -      AX -ל  AL  התמרה של
CWD  -   DX:AX -ל  AX  התמרה של
```

אם התוצאה אינה יכולה להכנס כולה ל-AL או AX בשל גודלה, מתרחשת חריגה

(מעין תקלה שמהותה תוסבר בפרק על פסיקות) הנקראת Divide Overflow הגורמת לעצירת התוכנית (זהו תגובה זהה אם מתבצע חלוקה של מספר כלשהוא באפס). למשל בקוד הבא

```
MOV AX,60000
MOV BL,100
DIV BL
```

יגרום ל- Divide Overflow משום שתוצאת החלוקה 600 לא יכולה להיות מוצבת ב-AL שהערך המירבי שהוא יכול להכיל הוא 255.

386 ואילך

הפקודות DIV ו-IDIV מוכללות בצורה דומה ל-MUL כלומר

```
DIV Op32

IDIV Op32
```

יחשבו את EDX:EAX / Op32 כמספרים חסרי סימן ועם סימן בהתאמה כאשר Op32 יכול להיות אוגר 32 ביט או אופרנד זכרון 32 ביט. אחרי חלוקה תקינה תוצב המנה ללא שארית ב-EAX ושארית החלוקה ב-EDX. אם לא ניתן לאכסן את התוצאה ב-EAX מבחינת הגודל או שמתבצע ש-Op32 מכיל אפס יתרחש Divide Overflow. בכדי לחלק 2 מספרים 32 ביט יש צורך לאפס את EDX במקרה של DIV ובמקרה של IDIV ישנה הפקודה

```
CDQ  -   EDX:EAX -ל  EAX  התמרה של
```

ישנה גם הפקודה

```
CWDE  -   EAX -ל  AX  התמרה של
```

השוואה

```
CMP
```

בצע חיסור - אבל תוצאת החיסור אינה נשמרת.

הפקודה משפיעה רק הדגלים הדגלים הארתמטיים, מושג שנרחיב עליו בשלב יותר מאוחר. בשלב זה נומר שהמשמעות של ההשוואה (חיסור) נשמרת אבל האופרנדים שומרים על ערכם, רק התשובה לשאלות אם הם שוים ואם לא מי גדול ממי נשמרים (בין אם מתיחסים למספרים כחסרי סימן או עם סימן) נשמרת. CMP היא פקודה של 2 אופרנדים כמו ADD או SUB. כלומר 2 האופרנדים יכולים להיות צירוף של אוגר/זכרון/קבוע, אוגר/זכרון לפי החוקים הרגילים. למשל:

```
CMP AX,BX
CMP AX,7
CMP AL,[BX]
CMP [SI],BX
CMP BYTE PTR [DI],19
```

386 ואילך

CMP פועל על האוגרים 32 ביט וזכרון או קבועים 32 ביט כמו SUB למשל

```
CMP EDX,ESI
```