**אסמבלר 24.10.18**

**מבנה המחשב**

a

MEMORY

CPu

I/O

Cpu to memory= address bus—חץ כחול

--Cpu to/back memory=data busחץ אדום

Cpu to memory=control bus—חץ ירוק

משמעות של גג היא שהפעולה מתבצעת במצב 0 (פעיל בנמוך '0' לוגי)

|  |  |  |
| --- | --- | --- |
| הפעולה שמבצע ה-CPU | WR | RD |
| מצב אסור | 0 | 0 |
| פעולת קריאה | 1 | 0 |
| פעולת כתיבה | 0 | 1 |
| מצב סרק(לא משתמש בפסים לצורך קריאה וכתיבה) | 1 | 1 |

כל מחשב מכיל שלוש יחידות בסיסיות :

1. CPU-יחידת עיבוד מרכזית, תפקידה לקרוא הוראות מהזיכרון ולבצען. ה-CPU היא היחידה המבצעת, המפקחת ומארגנת את כל הפעולות במערך המחשב. ה-CPU הוא החלק המרכזי החשוב ביותר והוא מהווה בעצם את המוח של כל המערכת.

* ביצוע-כל ההוראות שהמחשב מקבל לביצוע מבוצעות בתוך ה-CPU
* פיקוח-ה-CPU שולט על כל היחידות שמחוברות אליו וקובע את אופן פעולתן
* ארגון-במהירויות שבהן עובד המחשב יש צורך בתזמון מושלם של כל הפעולות המתקיימות בו, לצורך זה ה-CPU קובע את כל התזמונים הנדרשים

1. I/O -קלט פלט- זו היחידה שעוזרת לנו לתקשר עם "העולם החיצון", לדוגמא: מדפסת(פלט), מסך(פלט), מקלדת(קלט)
2. MEMORY-הזיכרון אוגר באופן זמני הוראות ונתונים הדרושים להפעלת המחשב. הזיכרון בנוי מהרבה תאים כאשר כל תא מזוהה על ידי מספר המוגדר ככתובת התא. פתיחת תא בזיכרון מתבצעת על ידי מתן כתובת לזיכרון. הגורם היחיד שיכול לתת את הכתובת לזיכרון הוא ה-CPU. עובר פרק זמן מסויים מהרגע שהמעבד מוסר את הכתובת ועד הרגע שבו התא נפתח (Memory Access Time-זמן גישה לזיכרון), זמן זה בזכרונות מהירים יכול להגיע עד עשרות ננו-שניות.

הוצאת מידע מתוך תא בזיכרון נקראת פעולת קריאה(READ),בפעולה זו מה שהיה קודם באותו התא נשאר בתא. הכנסת מידע לתוך תא בזיכרון נקראת כתיבה(WRITE),בפעולה זו מה שהיה קודם בתא נדרס\נמחק\נעלם.  
הקשר בין חלקי המחשב נעשה בעזרת שלוש קבוצות של מוליכים כאשר בכל מוליך עובר 0 או 1 לוגי.חלקי המחשב קשורים בינהם באמצעות מוליכים חשמליים שרמות המתח בינהם מתאימות לשני מצבים:0 לוגי אין מתח (0V) ו1 לוגי יש מתח (5V).  
אם רוצים להעביר ממקום למקום ביטים(סיביות\מידע), אנו למעשה מעבירים בקבוצות המוליכים אותות חשמליים,מוליכים אלו מחולקים לפי תפקידיהם לשלוש קבוצות כאשר כל קבוצה נקראת BUS (פס):

* Address bus-פס הכתובות-מסומן באותיות A0-An, בעזרת פס זה ה-CPU קובע לאיזו כתובת הוא פונה בזיכרון או ב-I/O.
* DATA BUS-פס הנתונים- מסומן באותיות D0-D7, דרכו ה-CPU מעביר נתון לזיכרון (פעולת כתיבה) או מקבל נתון מהזיכרון או הi/o (פעולת קריאה)
* CONTROL BUS-פס הבקרה- לא מסומן, בעזרת פס זה קובע ה-CPU האם הוא קורא נתונים או כותב נתונים לכתובת שאליה הוא פונה.

**משפחת ה-8086**

CPU

"המבנה של פון ניומן"-

חילוק של הCPU לשניים בכך נוצר

מצב של קיצור זמן העיבוד בחצי

שלבי ביצוע הפקודה:

כאמור ה-CPU קורא הוראות מהזיכרון ומבצע אותן. תהליך של כל ביצוע פקודה כולל שני שלבים:

1. OPCODE FETCH- הבאה הפקודה-
2. לאחר שהפקודה מגיעה מהזיכרון היא מפוענחת ומתבצעת , שלב זה נקרא "שלב ביצוע הפקודה".

המעבדים שנוצרו עד לפיתוחה של משפחת ה-8086 עבדו במבנה שנקרא "המבנה של פון ניומן",בצורה זו ה-CPU ניגש ומבין פקודה מהזיכרון, מבצע אותה ורק לאחר מכן ניגש ומביא פקודה נוספת. כלומר כל תהליך חדש של ביצוע פקודה מתחיל רק לאחר סיום פקודה קודמת.

**המבנה של משפחת ה-8086**

החידוש במשנה של משפחה זו הוא שהמעבד מורכב משתי יחידות עצמאיות:

1. BIU-Bus Interface Unit- יחידת הממשק לפס, תפקידה של יחידה זו הוא להביא הוראות ונתונים מהזיכרון ולשלוח נתונים לזיכרון.
2. ALU-Arithmetic logic unit-יחידה אריתמתית לוגית- תפקידה לבצע פעולות לוגיות כמו XOR NOT OR וכו' ופעולות חשבניות כמו חיבור וחיסור בינארי.

ההוראה הנוכחית שה-CPU מבצע היא זו שקובעת איזו פעולה צריכה יחידת הALU לבצע. פעולת ה-CPU בשיטה זו נקראת שיטת קו הצינור PipeLine. ההבדל הגדול בין המבנה של פון ניומן לבין המבנה של משפחת ה-8086 הוא בעובדה שיחידת הממשק לפס מביאה הוראות מהזיכרון בתהליך הנקרא Prefetch, כלומר מביאים פקודות מהזיכרון ומאחסנים אותן ביחידה הנקראת תור ההוראות. כאשר יחידת הביצוע מסיימת לבצע פקודה, הפקודה הבאה ממתינה לה כבר בתוך הPipeLine וכך נכסך הזמן שהיה דרוש להביא את הפקודה.

\*יש לציין שתור ההוראות מתנהג כמו זיכרון מסוג FIFO-first in first out הפקודה הראשונה שנכנסה תתבצע ראשונה.

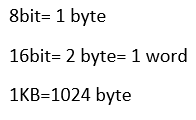
**שפת תכנות-אסמבלי(להוריד תוכנה emu8086)**

השפה הבסיסית שאותה מבין -CPU היא שפת מכונה (שפה של 0 ו 1 לוגיים) שפת סף (אסמבלי) היא שפה שכל פקודה בה מתורגמת לפקודה אחת בשפת מכונה. כאמור תוכנית מחשב היא אוסף של מספרים בינאריים הנמצאים בתאי הזיכרון בזה אחר זה, סדר המספרים של הפקודות קובע את משמעות התוכנית ואת פעולתו של ה-CPU.

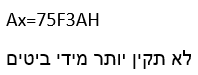
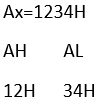
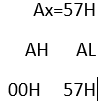
לכל מעבד יש כתובת התחלה שהיא כתובת התא שמכיל את ההוראה הראשונה בתוכנית ( ב8086 הכתובת FFFF0H) כדי לגרום ל-CPU לבצע תוכנית יש להפנות אותו לתא הראשון וזאת על ידי פעולת RESET ומשם הוא ממשיך בתוכנית ב"כוחות עצמו" על ידי ה-IP,

**האוגרים של ה-8086**

אוגר- זהו תא זיכרון בודד המשמש להגירה זמנית של נתונים (\*בשפת אסמבלי אני נכתוב תוכניות ונעזר באוגרים הקיימים ב-8086\*)

אוגרים כלליים ואוגרי נתונים(מכילים עד 16bit אם יש יותר זה שגוי):

|  |  |  |
| --- | --- | --- |
|  |  | שם האוגר |
| AL  8bit | AH  8bit | Ax |
| BL  8bit | BH  8bit | Bx |
| CL  8bit | CH  8bit | Cx |
| DL  8bit | DH  8bit | Dx |



אוגרי הצבעה ואינדקסים(אוגרים של 16BIT לא ניתנים לחלוקה):

* SI- מצביע על כתובת של נתון בזיכרון
* DI-מצביע על כתובת של נתון בזיכרון
* BP-מצביע עזר על המחסנית
* SP-מצביע המחסנית
* IP-מצביע על הכתובת של הפקודה הבאה לביצוע בתוכנית

אוגרי סגמנט(16BIT ללא חלוקה):

* DS-Data segment-אוגר סגמנט הנתונים
* ES-Extra segment-אוגר סגמנט נתונים נוסף
* SS-Stack segment- אוגר סגמנט המחסנית
* CS-Code segment-אוגר סגמנט הקוד\התוכנית

אוגר הדגלים:

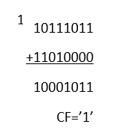
היחידה האריתמתית לוגית (ALU) ,תפקידה לבצע את הפעולות במערך המחשב, בנוסף יחידה זו מפיקה מידע על התוצאה שהתקבלה. מידע זה עובר לאוגר מיוחד שהסיביות שלו נקראות דגלים.

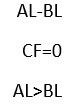
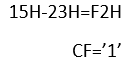
דגל זוהי סיבית שנמצאת בתוך אוגר הדגלים שגודלו 16 סיביות, ורק 9 מהן מנוצלות.

ישנם שני סוגי דגלים:

1. דגלי מצב שתפקידם להראות משהו על התוצאה שהתקבלה ביחידת ה-ALU.

* ZF-Zero flag- דגל זה מקבל 1 לוגי כאשר התוצאה היא 0. ודגל זה מקבל 0 לוגי כאשר התוצאה שונה מ-0. אחד השימושים לדגל זה הוא בדיקת שיוויון או אי שיוויון בין שני ערכים.
* CF-Carry flag-דגל הנשא- דגל זה מקבל 1 לוגי לאחר פעולת חיסור כאשר לווינו מדרגה דימיונית (כאשר מבצעים פעולת חיסור ממספר קטן פחות מספר גדול).  
  לאחר פעולת חיבור דגל זה מקבל 1 לוגי כאשר מתקבלת סיבית 9 או סיבית 17   
  \*אחד השימושים לדגל זה הוא בדיקת היחס קטן-גדול.

\*כאשר מחסרים שתי ספרות בשתי ספרות רושמים רק שתי ספרות מימין לשמאל של התוצאה



**מבנה פקודה בשפת אסבלי**

כל פקודה קובעת פעולה שה-CPU צריך לבצע. פעולה זו יכולה להיות לדוגמא קריאה של נתון מהזיכרון לתוך אחד מאוגרי ה-CPU, חיבור תוכן של שני אוגרים, היפוך לוגי של תוכן אוגר ב-CPU וכו'.

כפי שברור מדוגמאות אלו ההוראה צריכה לבצע ולקבוע גם מה הם הנתונים שעליהם מבוצעת הפעולה.

החלק המגדיר בהוראה את הפעולה שיש לבצע נקרא קוד הפעולה, והנתונים שעליהם מבוצעת הפעולה נקראים אופרנדים.

\*התוצאה תמצא תמיד באופרנד היעד.

מבנה א'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

אופרנד המקור

קוד הפעולה

אופרנד היעד

מבנה ב'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

אופרנד היעד

קוד הפעולה

מבנה ג'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

קוד הפעולה

פעולת MOV מעתיקה את המידע מאופרנד המקור לאופרנד היעד.

\*כל החוקים שיש על פקודת MOV תופסות לכל הפקודות האחרות.

**שיטות מיעון**

לכל פקודה יש מספר שיטות מיעון. שיטת מיעון היא הדרך שבה מייצגים את האופרנדים. אנו נדגים את כל שיטות המיעון על הפקודה MOV שתפקידה להעביר מידע מאופרנד המקור לאופרנד היעד.

פקודה במיעון אוגר

1. Mov AL, BL-חוקי, מעביר מידה מ-BL ל-AL
2. Mov AL, BX-לא חוקי, BX הוא 16 ביטים בזמן ש-AL הוא 8 ביטים
3. Mov BX, AL-לא חוקי, אין התאמה בגודל האופרנדים.

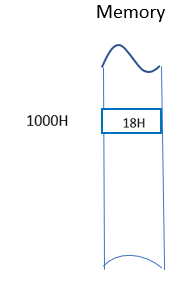
בשיטת מיעון אוגר המידע שעובר בין האופרנדים חייב להיות באותו גודל (8 ביט או 16 ביט)

\*אם לא נרשמת אות עם המספר הוא בבסיס עשרוני, H מסמן בסיס 16,B מסמן בסיס בינארי

\*כל ספרה בהקסה היא 4 ספרות בבינארי

פקודה במיעון מיידי

1. Mov AL,83H-חוקי
2. Mov AL, 182-חוקי, כל עוד מספר בבסיס עשרוני מתחת ל255 ניתן להכניס לאופרנד של 8 ביט.
3. Mov AH, 182H-לא חוקי, לא ניתן להעביר נתון של 16 ביט לאוגר של 8 ביט
4. Mov AX, 182H-חוקי AH=01 AL=82H
5. Mov 12H, AL-לא חוקי, 12H הוא אופרנד היעד אינו בר איחסון.
6. Mov AL,20H-חוקי
7. Mov AL,00100000B-חוקי
8. Mov AL,32 -חוקי

\*מספר בבסיס בינארי חייב להחיל את כל הספרות כולל האפסים בשמאל עם האות B בצד ימין

פקודה במיעון עקיף לזיכרון

[ ]-סוגריים מרובעים מציינים כתובת של נתון בזיכרון.

\*כאשר רואים סוגריים מרובעים ישר מציירים זיכרון

נתונה התוכנית הבאה בשפת אסמבלי עקוב אחר התוכנית על ידי טבלת מעקב ורשום לאחר כל פקודה את התוצאה המתקבלת באופרנד היעד. \*בטבלת מעקב רושמים כל פקודה בשורה חדשה

Mov SI, 1000H

Mov AL, 18H

Mov [SI], AL

|  |  |
| --- | --- |
| AL | SI |
| 18H | 1000H |
|  | 18H |

כתוב תוכנית שמעבירה את הנתון 15H לכתובת 2000H בזיכרון

Mov SI,2000H

Mov AL,15H

Mov [SI], AL

**\*כאשר עובדים עם מספר בבסיס הקסה דצימלי (בסיס 16) שספרת הMSB שלו (השספרה השמאלית) היא מספר בין A-F בבסיס 16 חייב לרשום לפני כן את הערך 0 וזאת כדי להבחין בין שם של משתנה למספר בבסיס 16**

Mov [2000H], 15H

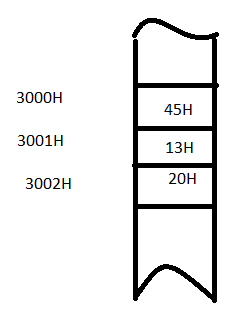
לא חוקי,לא ניתן בשיטת מיעון יקיף לרשום את הכתובת בצורה ישירה

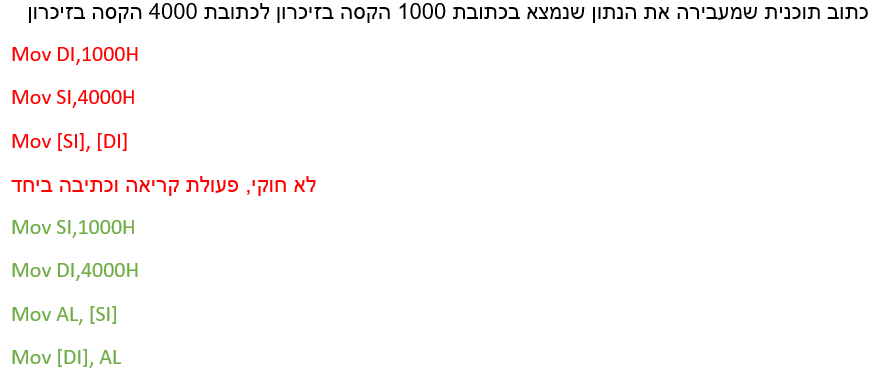
נתונה התוכנית הבאה, יש לעקוב אחר התוכנית ע"י טבלת מעקב ולרשום את התוצאות שמתקבלות באופרנד היעד לאחר כל פקודה:

Mov DI,3000H

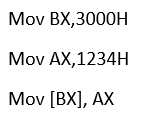
Mov AH, [DI]

|  |  |
| --- | --- |
| DI | AH |
| 3000H |  |
|  | 45H |







\*הפקודה mov [BX],AX מעבירה את הנתון שנמצא באוגר AX לכתובת שהאוגר BX מצביע עליה בזיכרון. מכיוון שהאוגר AX הוא בגודל 16 ביט הCPU טוען את הנתון לשני בתים עוקבים בזיכרון

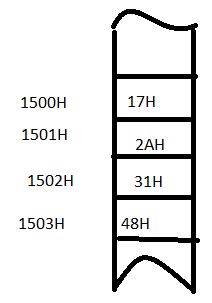
האוגרים שיכולים לשמש כמצביעים על כתובת של נתון בזיכרון הם:

* BX
* SI
* DI

כל דבר אחר שירשם בתוך סוגריים מרובעים הוא לא חוקי

**הרצת תוכנית במחשב**

1. לוחצים NEW ובוחרים באפשרות הראשונה
2. נכתוב את התוכנית
3. לוחצים COMPILE
4. מבטלים את השמירה
5. לוחצים CLOSE
6. לוחצים EMULATOR
7. בוחרים את האופציה השניה
8. לוחצים על VIEW ובוחרים MEMORY
9. לוחצים על SINGLE STEP כדי לבצע את הפעולות אחת אחת



Mov DI, 1500H

Mov AX, [DI]

|  |  |  |
| --- | --- | --- |
| DI | AH | AL |
| 1500H |  |  |
|  | 2AH | 17H |
|  |  |  |
|  |  |  |

\*הפקודה mov AX,[DI] מעבירה את הנתון מהכתובת שהאוגר DI מצביע עליו בזיכרון לאוגר AX. גם כאן מכיוון שאופרנד היעד הוא בגודל 16 ביט ה-CPU לוקח נתון משתי כתובות עוקבות בזיכרון.

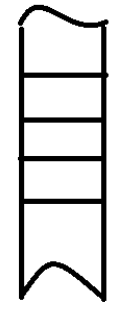
\*\*בשיטת מיון עקיף לזיכרון האוגרים שיכולים לשמש כמצביע על כתובת של נתון בזיכרון ולהיות בתוך סוגריים מרובעים הם BX,SI,DI בלבד

פקודה במיעון אינדקס-בסיס

בשיטת מיעון זו האוגר BX משמש כאוגר בסיס ואילו האוגרים SI ו- DI משמשים כמצביעים. בשיטת מיעון זו הכתובת שאליה פונים חייבת להיות מורכבת מ-BX ו- SI או BX ו- DI.

לדוגמא: נתונה התוכנית הבאה יש לעקוב בעזרת טבלת מעקב ולרשום את התוצאות לאחר כל פקודה

Mov BX,2000H

Mov DI,1000H

Mov AL, [BX+DI]

1000H

2000H

3000H

|  |  |  |
| --- | --- | --- |
| BX | DI | AL |
| 2000H |  |  |
|  | 1000H |  |
|  |  | הערך בכתובת 3000 |

שני הצדדים בכל אחת מהשורות הבאות שקולים אחד לשני\*

Mov AH, [BX][SI] == mov AH, [BX+SI]

Mov AL,4[BX][DI] == mov AL, [BX+DI+4]

\*סדר הכתיבה חשוב בצד ימין הספרה חייבת להיות בימין ובשמאל מצד שמאל

Mov AL, [SI][DI]

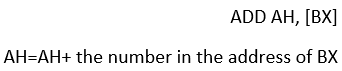
לא חוקי חסר אוגר בסיס

לאחר כל פקודה בשפת אסמבלי הדגלים מתעדכנים בהתאם לתוצאה שהתקבלה,למעט לאחר הפקודה mov שבה הדגלים לא מתעדכנים, אלא נשארים כפי שהיו לפני הפקודה. כמו כן לאחר הפקודות

INC ו- DEC קארי פלאג (carry flag) לא מתעדכן.

פקודות בסיסיות בשפת אסמבלי

1)ADD-הוספה/חיבור האופרנדים או תכנים בכתובות.

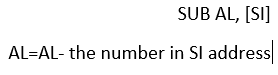
ADD AL, BL

AL= AL+BL

ADD [BX+SI], AL

תוכן התא בזיכרון שהוא חיבור בין הכתובות BX ו-SI, תוכן התא יחובר עם AL

2)SUB-חיסור בין האופרנדים או תכנים בכתובות.

SUB AH, AL

AH=AH-AL

3)INC,DEC -קידום של הערך באחד, חיסור של הערך באחד

INC AH-לקדם ב-1

AH=AH+1

DEC AH לחסר ב-1

AH=AH-1

**פקודות קפיצה**

1. פקודת קפיצה "ללא תנאי"- בסוג זה של פקודת קפיצה מקבל ה-CPU פקודה לקפוץ למקום כלשהו בזיכרון התוכנית ללא שום תנאי מקביל,

לדוגמא:JMP XYZ   
JMP-פקודת קפיצה ללא תנאי  
XYZ תווית שם כלשהו

1. פקודת קפיצה "עם תנאי"- בסוג זה של פקודת הקפיצה ה-CPU מקבל פקודה לקפוץ למקום כלשהו בתוכנית בהתאם לדגל על פי הפקודה הבאה

|  |  |  |
| --- | --- | --- |
| משמעות | קפיצה | סוג דגל |
| קפוץ לתווית אם ZF =1  קפוץ לתווית אם ZF = 0 | תווית JZ  תווית JNZ | ZF |
| קפוץ לתווית אם CF = 1  קפוץ לתווית אם CF = 0 | תווית JC  תווית JNC | CF |

כתוב תוכנית שתבדוק אם הנתון שנמצא באוגר AL שווה לנתון שנמצא באוגר AH

אם כן, יש להעביר לאוגר CL את הערך 40H

אחרת, יש להעביר לאוגר CL את הערך 50H

SUB AL, AH בדיקה האם שני הערכים שווים אחד לשני

JZ XYZ קפיצה לתווית בהנחה ושני הערכים שווים

mov CL,50H במידה ודגל האפס הוא 0 זה אומר שהערכים שונים ולכן נעביר 50

JMP sof קפיצה ללא תנאי כדי לדלג על החלק שלא נרצה לבצע במידה והם שונים

XYZ: mov CL, 40Hהתווית אליה נקפוץ אם דגל האפס שווה 0 ולכן נעביר 40

Sof: השורה אליה נקפוץ מהקפיצה ללא תנאי

פקודת compare

פקודת COMPARE תירשם על ידי האותיות CMP

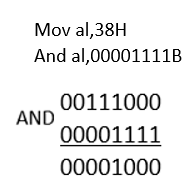
הפקודה בודקת שיוויון בין שני אופרנדים בלי לשנות את הערכים באופרנד, פקודת CMP תיתן ZF=1 אם הערכים שווים אחד לשני

במקרים רבים הכנסת תוצאה לאופרנד היעד כתוצאה מפעולת החיסור מפריע לנו מכיוון שאנו מעוניינים רק לבדוק תנאי במלי לשנות את אופרנד היעד, ולשם כך קיימת הפקודה CMP שמבצעת חיסור אבל לא מעדכנת את הערך של אופרנד היעד.

\*מספר מסומן-הסיבית השמאלית היא סיבית סימן, אם היא 0 המספר חיובי, אם 1 המספר שלילי

\*במספר שלילי יש לבצע שיטת המשלים ל-2, הופכים את כל הספרות להופכי(0 הופך ל1 ו-1 הופך ל0), ומוסיפים 1. התוצאה תהיה המספר החיובי.

פקודות לוגיות

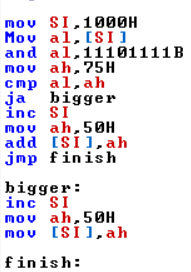
AND לוגי- אחד השימושים של הפקודה AND לוגי הוא לאפס

סיבית או סיביות מתוך נתון כלשהו ואת השאר להעביר ללא שינוי

|  |
| --- |
| al |
| 38H |
| 08H |

**תרגיל:**

כתוב תוכנית שמאפסת את הסיבית D4 לנתון בכתובת 1000Hבזיכרון, אם לאחר פעולה זו התוצאה שהתקבלה גדולה מ75H יש להעביר לכתובת 1001H את הערך 50H אחרת יש להוסיף 50H לנתון בכתובת 1001H.

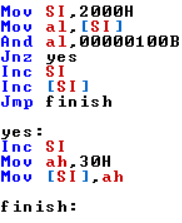


השימוש השני לפקודה AND לוגי הוא בדיקת ערך של סיבית בודדת או סיביות מתוך נתון כלשהו. זאת אומרת באותה סיבית יש לשים 1 לוגי שאר הסיביות יקבלו 0 ונבצע פעולת AND לוגי.

לאחר פעולה זו נבדוק את ה-ZF, אם ה-ZF שווה 1 משמע שהתוצאה שהתקבלה היא 0 ולכן הסיבית הנבדקת הייתה 0.

אם ה-ZF שווה 0 לוגי, זאת אומרת שהתוצאה שהתקבלה שונה מ-0, משמע הסיבית הנבדקת שווה ל-1.

**תרגיל:**

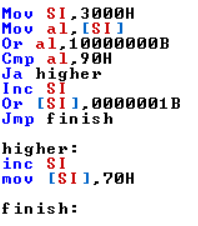
כתובת תוכנית שבודקת את ערך סיבית D2 לנתון בכתובת 2000H בזיכרון, אם ערך הסיבית שווה ל-1 לוגי יש להעביר לתא 2001H את הערך 30H, אחרת יש לקדם באחד את הנתון בכתובת 2001H בזיכרון

OR לוגי:

אחד השימושים של הפקודה OR לוגי הוא קביעת 1 לוגי בסיבית או סיביות מתוך נתון כלשהו ואת שאר הסיביות להשאיר ללא שינוי

\*סיבית ימנית במספר בינארי קובע אם המספר הוא זוגי או אי-זוגי,0-זוגי,1-אי זוגי

**תרגיל**

כתוב תוכנית שקובעת 1 לוגי בסיבית D7 לנתון בכתובת 3000H בזיכרון, אם לאחר פעולה זו התוצאה שהתקבלה גדולה מ90H יש להעביר לתא 3001H את הערך 70H,אחרת יש לקבוע 1 לוגי בסיבית D0 של הנתון בתא 3001H בזיכרון

NOT לוגי:

\*בפקודת NOT לא מעדכנים את הדגלים

הפקודה NOT מבצעת היפוך לוגי למידע הקיים באופרנד היעד. משמע 0 לוגי הופך ל-1 לוגי, ולהפך.

\*אין אפשרות ע"י פקודה זו להפוך ערך של סיבית בודדת או סיביות מתוך נתון כלשהו.

XOR לוגי:

|  |  |  |
| --- | --- | --- |
| F | B | A |
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |

x XOR 0= x

x XOR 1= ~x

אחד השימושים לפקודה XOR לוגי הוא להפוך ערך של סיבית בודדת או סיביות מתוך נתון כלשהו ואת שאר הסיביות להעביר ללא שינוי. בניגוד לפקודה NOT שהופכת את כל המידע שבאופרנד.

פקודת TEST:

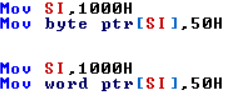
הפקודה TEST מבצעת פעולת AND לוגי אך לא מעדכנת את אופרנד היעד, אלא את הדגלים בלבד (דגל האפס ) וכך ניתן בעזרת פקודה זו לבדוק ערך של סיבית או סיביות מתוך נתון כלשהו מבלי לשנות את אופרנד היעד.

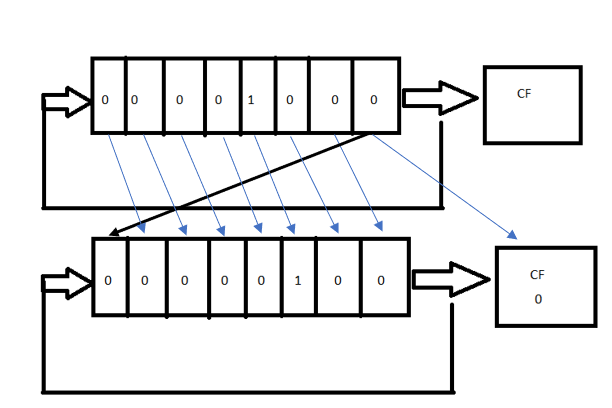
כתוב תוכנית הבודקת את ערך סיבית D4 לנתון הקיים בכתובת 400H בזיכרון. אם ערכה של סיבית זו שווה ל-0 לוגי יש להוסיף 50H לנתון בכתובת 401H, אחרת יש לקדם ב-1 את הנתון בכתובת 401H.

האופרטור PTR:

האופרטור PTR מאפשר לציין במדוייק מהו סוג הנתון איתו אנו משתמשים במקרים בהם הדבר לא ברור שבהם הדבר לא ברור. כדי להחליט האם הפעולה שהתבצעה היא על נתון בגודל BYTE או על נתון בגודל WORD, יש להשתמש באופרטור PTR

\*אנו נשתמש באופרטור PTR אך ורק כאשר נרצה להעביר נתון בצורה ישירה לתוך **תא בזיכרון**



פקודות סיבוב:

1. ROR al,N

הפקודה הנ"ל תפקידה לסובב את

הנתון שנמצא באופרנד היעד פעם

אחת ימינה, זאת אומרת סיבית הLSB

מגיעה לכיוון סיבית ה-MSB וכל שאר הסיביות

מתקדמות פעם אחת ימינה, כמו כן סיבית

ה-LSB מעדכנת את הCF.

\*אם רוצים לסובב יותר מפעם אחת

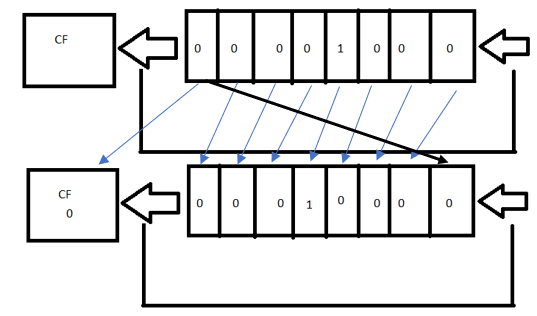
חייבים לעבור דרך אוגר CL בלבד.

שימושים אפשריים לפקודה זו:

* 1. בדיקת סיבית LSB וזאת על ידי סיבוב פעם אחת ימינה ובדיקת ה-CF
  2. חילוק ללא שארית ב 2 בחזקת N, כאשר N זה מספר סיבובים

1. ROL al,N

הפקודה הנ"ל תפקידה לסובב את הנתון שנמצא באופרנד היעד פעם אחת שמאלה, זאת אומרת סיבית ה-LSB באה לכיוון סיבית ה-MSB וכל שאר הסיביות זזות שמאלה פעם אחת כמו כן סיבית ה-MSB מעדכנת את ה-CF, גם כאן אם רוצים לסובב יותר מפעם אחת נאלץ להשתמש באוגר CL בלבד.

שימושים אפשריים:

* 1. בדיקת סיבית MSB על ידי

סיבוב פעם אחת שמאלה

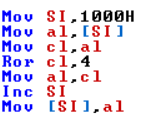
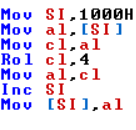
ובדיקת ה-CF

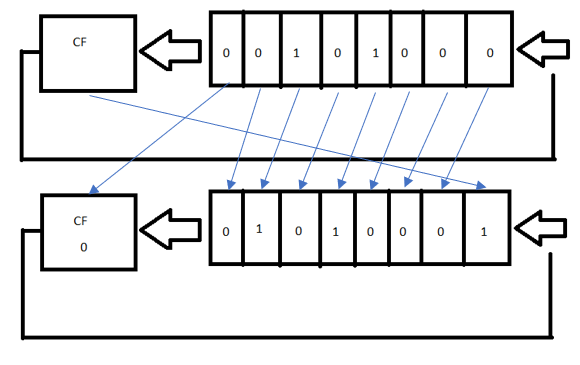
* 1. כפל של האופרנד ב

בחזקת N כאשר N מציין

את מספר הסיבובים

כתוב תוכנית שלוקחת את הנתון מכתובת 1000H בזיכרון , מסובבת אותו 4 פעמים ואת התוצא יש להציב בכתובת 1001H

\\\

1. RCL al,N

הפקודה הנ"ל תפקידה

לסובב את הנתון שנמצא

באופרנד היעד פעם אחת

שמאלה דרך ה-CF, זאת

אומרת ערכה של סיבית

ה-CF מגיע לסיבית ה-LSB

וכל שאר הסיביות זזות פעם

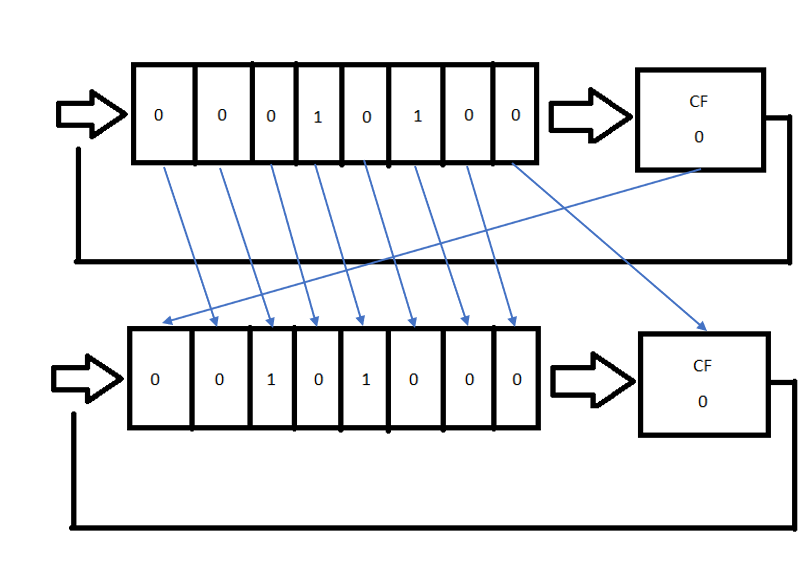
אחת שמאלה, כמו כן סיבית

ה-MSB מעדכנת את ה-CF

גם כאן אם רוצים לסובב יותר

מפעם אחת יש להשתמש

באוגרcl בלבד

אחד השימושים לפקודה זו הוא בדיקת סיבית MSB

1. RCR al,N

הפקודה הנ"ל תפקידה

לסובב את הנתון שנמצא

באופרנד היעד פעם אחת

ימינה דרך ה-CF, זאת

אומרת ערכה של סיבית

ה-CF מגיע לסיבית ה-MSB

וכל שאר הסיביות זזות פעם

אחת ימינה, כמו כן סיבית

ה-LSB מעדכנת את ה-CF

גם כאן אם רוצים לסובב יותר

מפעם אחת יש להשתמש

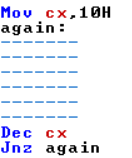
באוגרcl בלבד

אחד השימושים לפקודה זו הוא בדיקת סיבית LSB

לולאות

ישנן שתי סוגים של לולאות:

1. לולאה אינסופית המתבצעת אינסוף פעמים לדוגמא
2. לולאה סופית

יש לרשום את מספר הפעמים שרוצים להריץ את הלולאה

מימוש לולאה סופית בשפת אסמבלי מתבצע בצורה נוחה

וזאת על ידי נתינת ערך למונה הלולאה מחוץ לגוף הלולאה.

מבצעים את גוף הלולאה כשבסיומה יש להוריד 1 ממונה הלולאה

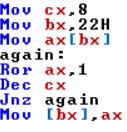
ובודקים:

אם מונה הלולאה שונה מ-0 (zf=0) קופצים לתווית שבסמוך לפקודה

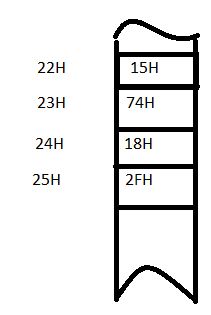
ומבצעים את הלולאה פעם נוספת. אם מונה הלולאה שווה 0 (zf=1)

התנאי לא מתקיים ואז יוצאים מגוף הלולאה וממשיכים בתוכנתית

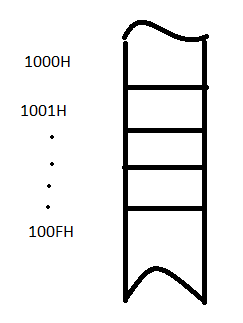
נתונה התוכנית הבאה בשפת אסמבלי עבור המעבד 8086:

1. יש לבצע מעקב על ידי טבלת מעקב עבור האטרציה הראשונה והשנייה של הלולאה.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CF | ZF | bx | cx | ax |
|  |  | 22H | 8 | 01110100000010101 |
| 1 | 0 |  | 7 | 00111010000001010 |
| 0 | 0 |  | 6 | 10011101000000101 |
| 1 | 0 |  | 5 | 01001110100000010 |
| 0 | 0 |  | 4 | 10100111010000001 |
| 1 | 0 |  | 3 | 01010011101000000 |
| 0 | 0 |  | 2 | 10101001110100000 |
| 0 | 0 |  | 1 | 01010100111010000 |
| 0 | 1 |  | 0 | 00101010011101000 |

1. סרטט את מצב הזיכרון בסוף התוכנית.
2. רשום במילים מה מבצעת התוכנית.

תוכניות המטפלות בבלוקים/מערכים של נתונים

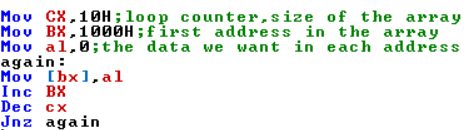
בלוק של נתונים (מערך של נתונים) מוגדר כאוסף של נתונים הנמצאים בזיכרון ברצף. כאשר נכתוב תוכניות שמטפלות בבלוקים של נתונים אנו נשתמש בלולאות ובעוד אלמנט נוסף הנקרא מצביע הבלוק. הנתונים הנמצאים בזיכרון בכל תא הם בגודל 8bit, אנו נכתוב תוכניות שמטפלות בנתונים בגודל byte או בגודל word.

\*בדיקת מספר הערכים במערך

(100FH-1000H)+1 =10H

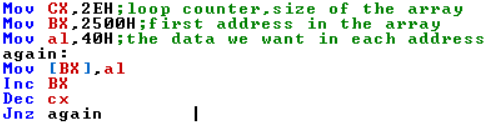
לדוגמא

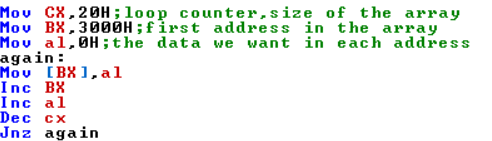
כתוב תוכנית לאיפוס בלוק של 10H נתונים הנמצאים בזיכרון בכתובת 1000H ועד כתובת 100FH



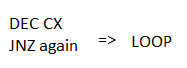
כתוב תוכנית שמעבירה את הערך 40H לתאים בזיכרון מכתובת 2500H עד 252DH

252DH-2500H+1=2EH



כתוב תוכנית שמעבירה וקטור של מספרים בסדר עולה החל מהערך 0 בתא 3000H בזיכרון עבור 20H נתונים.

הפקודה LOOP

הפקודה LOOP נועדה להקל עלינו בכתיבת תוכניות המשתמשות בלולאות.

פקודה זו מבצעת שתי פעולות:

1. מורידה 1 ממונה הלולאה (CX)
2. בודקת האם מונה הלולאה (CX) שונה מ-0, אם כן קופצים לתווית שבסמיכות לפקודה ומבצעים את הלולאה פעם נוספת. אחרת ממשיכים לפקודה הבאה בתוכנית

\*פקודה זו פועלת על האוגר CX בלבד.

\*פקודת LOOP לא מעדכנת את הדגלים.



תרגיל:

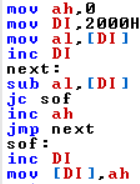
נתונה התוכנית הבאה בשפת אסמבלי עבור המעבד 8086.

1. יש לבצע מעקב עבור האיטרציה הראשונה והשניה של הלולאה.
2. סרטט את מצב הזיכרון בסוף התוכנית.
3. רשום במילים מה התוכנית מבצעת בלי להזכיר אוגרים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cf | zf | al | cl | SI | CH |
|  |  |  |  |  | 0 |
|  |  |  |  | 2000H |  |
|  |  |  | 4H |  |  |
|  |  | 0 |  |  |  |
| 0 | 0 |  |  | 2001H |  |
| 0 | 0 | 2 |  |  |  |
|  |  |  | 3H |  |  |
| 0 | 0 | 4 |  |  |  |
|  |  |  | 2H |  |  |
| 0 | 0 | 6 |  |  |  |

|  |  |
| --- | --- |
| ערך | תא |
| 4 | 2000H |
| 2 | 2001H |
| 8 | 2002H |

התוכנית מבצעת פעולת הכפלה בין הערכים בכתובת 2000H ו-2001H ומכניסה את התוצאה בגודל byte לכתובת 2002H

תרגיל:

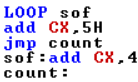
נתונה התוכנית הבאה בשפת אסמבלי עבור המעבד 8086.

1. יש לבצע מעקב עבור האיטרציה הראשונה והשניה של הלולאה.
2. סרטט את מצב הזיכרון בסוף התוכנית.
3. רשום במילים מה התוכנית מבצעת בלי להזכיר אוגרים

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CF | ZF | DI | al | ah |
|  |  |  |  | 0 |
|  |  | 2000H |  |  |
|  |  |  | 11H |  |
| 0 | 0 | 2001H |  |  |
| 0 | 0 |  | EH |  |
| 0 | 0 |  |  | 1 |
| 0 | 0 |  | BH |  |
|  |  |  |  | 2 |
| 0 | 0 |  | 8H |  |
|  |  |  |  | 3 |
| 0 | 0 |  | 5H |  |
|  |  |  |  | 4 |
| 0 | 0 |  | 2H |  |
|  |  |  |  | 5 |
| 1 | 0 |  | FH |  |
|  |  | 2002H |  |  |

|  |  |
| --- | --- |
| ערך | כתובת |
| 11H | 2000H |
| 3H | 2001H |
| 5 | 2002H |

התוכנית מבצעת פעולת חילוק בין הערך בכתובת 2000H והערך בכתובת 2001H ומכניסה את התוצאה ללא שארית בגודל byte לכתובת 2002H

נתון קטע הקוד הבא, רשום את ערכו של CX בסוף התוכנית במקרים הבאים:

|  |  |
| --- | --- |
| CX | CX |
| 0 | 1 |
| FFFFH | 0 |
| 3 | 5 |

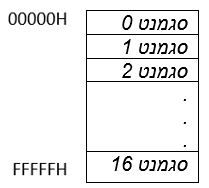
1. ערכו ההתחלתי של CX הוא 1
2. ערכו ההתחלתי של CX הוא 0
3. Cx=5
4. Cx=3

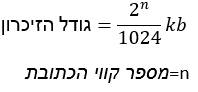
סיגמנטציה

בתוך הפקודה mov al,[BX] אנו רואים שניתן לפנות לזיכרון בתחום כתובות של 0000H-FFFFH שהם ה-64Kb. מצד שני מתוך דפי היצרן של המעבד ממשפחת ה-8086 , ניתן לראות שאפשר לפנות לתחום כתובות 00000H-FFFFFH ( A0-A19) שמתאים לחיבור זיכרון של 1Mb.

אנו רואים שכביכול אנו לא מסוגלים לפנות לכל מרחב הזיכרון הנ"ל מכיוון שגודל האוגרים המצביעים על כתובת בזיכרון הוא 16bit.

התשובה לסטירה זו טמונה באוגרי הסגמנט ובמושג הסגמנטציה:

1. מחלקים את הזיכרון למקטעים הנקראים סגמנטים כאשר כל סגמנט תופס מקסימום מרחב זיכרון של 64Kb בצורה הבאה:



1. בעזרת אוגרי הסיגמנט הקיימים במעבד ניתן לפנות לכתובת פיזית אמיתית ולחשב אותה:
   1. בעזרת האוגר ds (data segment-אוגר סגמנט הנתונים) ניתן לפנות לכתובת הנמצאת בסגמנט הנתונים בזיכרון.
   2. בעזרת האוגר cs (code segment-אוגר סגמנט הקוד\התוכנית) ניתן לפנות לכתובת של פקודה הנמצאת בסגמנט הקוד בזיכרון.
   3. בעזרת האוגר ss (stack segment-אוגר סגמנט המחסנית) ניתן לפנות לכתובת של נתון הנמצא במחסנית.
   4. בעזרת האוגר es (extra segment-אוגר סגמנט הנתונים הנוסף) ניתן לפנות לכתובת של נתון הנמצא בסגמנט הנתונים הנוסף.

\*בדרך כלל מי שייתן את הערך לאוגרי הסגמנט יהיה מערכת ההפעלה

חשב את בכתובת הפיזית האמיתית שאליה פונה ה-CPU בקטע הקוד הבא:

אפס קבוע

בנוסחה

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 |

אוגר סגמנט  
 +

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 2 |

כתובת יחסית

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 2 | 1 |

כתובת פיזית אמיתית

חשב את בכתובת הפיזית האמיתית שאליה פונה ה-CPU בקטע הקוד הבא:

אפס קבוע

בנוסחה

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | F | 3 | C |

אוגר סגמנט  
 +

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 | 3 | 5 |

כתובת יחסית

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 2 | 2 | 9 | C |

כתובת פיזית אמיתית

במרחב הזיכרון של מערכת מיקרו מחשב יש 1MB. הגישה לזיכרון נעשית בשיטת המקטעים.

להגדרת כתובת ראשונה בכל מקטע משתמשים באוגר מקטע המכיל 4 ספרות HEX המשמעותיות של כתובת זו (הספרה HEX החמישית נקבעת כ-0) , את הכתובת הפיסית האמיתית בתוך המקטע קובעים במערכת זו ע"י צירוף ערכו של אוגר ההיסט בין 4 ספרות HEX לכתובת תחילת המקטע.

1.מהם הכתובות הפיסיות ב-HEX בתחילת הזיכרון וסופו?

00000H-FFFFFH

2.מהן מספרי הבתים ב-KB שמכיל כל מקטע?

64KB

3.מהו המספר המרבי של מקטעים שונים בזיכרון מבלי שתהיה חפיפת כתובות בינהם?

16bit

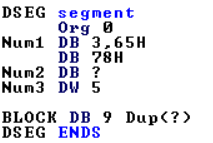
4.השלם את הטבלה הבאה:

|  |  |  |
| --- | --- | --- |
| הכתובת הפיזית של בזיכרון | תוכן אוגר ההיסט | תוכן אוגר מקטע |
| 31730H | A300H | 2743H |
| 25964H | B214H | 1A75H |
| C79A2H | AB02H | BCEAH |

כתוב פקודה אחת בלבד שמעבירה את הנתון 34H לכתובת 2500H בDS בזיכרון



הגדרת הנתונים בסגמנט הנתונים בזכרון

Data segment בזיכרון

|  |  |
| --- | --- |
| 3 | 0000H |
| 65H | 0001H |
| 78H | 0002H |
|  | 0003H |
| 05H | 0004H |
| 00H | 0005H |
|  | 0006H |
|  | 0007H |
|  | 0008H |
|  | 0009H |
|  | AH |
|  | BH |
|  | CH |
|  | DH |
|  | EH |

NUM1

NUM2

NUM3

BLOCK

ORG 0- קביעת מיקום בזיכרון זאת אומרת מאיפה בתוכנית הנ"ל אני מתחיל להציב נתונים בסגמנט הנתונים (כתובת יחסית).

DB-define byte- הגדר נתון בגודל 8bit-byte

DW-define word- הגדר נתון בגודל 16bit-word

בעזרת הגדרה זו אנו משכפלים את הערך שנמצא בתוך הסוגריים n פעמים. בדוגמא שלנו אנו נשכפל את הסימן שאלה (מקום ריק) תשעה תאים בזיכרון החל מהכתובת שהמשתנה block מצביע עליה.

הפקודה LEA-load effective address-טען כתובת אפקטיבית\יחסית

Lea BX,BLOCK

הפקודה הזו מקבילה גם לפקודה

Mov BX,offset BLOCK

הגדרת קבועים בתוכנית

בדרך כלל משתמשים בהגדרה של קבועים בתוכנית לנוחות המשתמש על מנת לקבוע ערך מסויים בתור קבוע. אם בהמשך רוצים לשנות את ערך זה, משנים אותו בהגדרתו ואז אוטומטית הוא משתנה בכל התוכנית.

\*קבועים לא תופסים מקום בזיכרון

MOD=modolo\*

100h mod 17🡪256/17=15.056🡪15\*17=255🡪256-255=1

100h mod 17=1

פקודת הזזה

פקודת הזזה לוגית

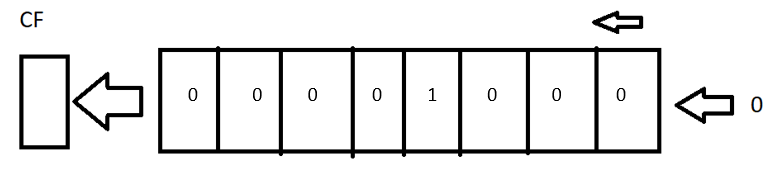
SHL AL,1

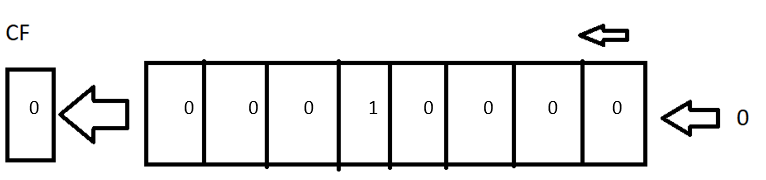
הפקודה shl al,1 תפקידה להזיז את הסיביות שנמצאות באופרנד היעד פעם אחת שמאלה. זאת אומרת 0 לוגי נכנס מצד ימין כל הסיביות זזות פעם אחת שמאלה. כמו כן ערך סיבית ה-MSB מעדכנת את ה-CF

אם רוצים להזיז יותר מפעם אחת נעביר ל-CL את הערך שרוצים ואז נרשום SHL AL,CL (נעבור דרך CL בלבד.

שימושים אפשריים לפקודה זו הם:

1. כפל של האופרנד ב
2. בדיקת סיבית ה-MSB כתוצאה מבדיקת ה-CF





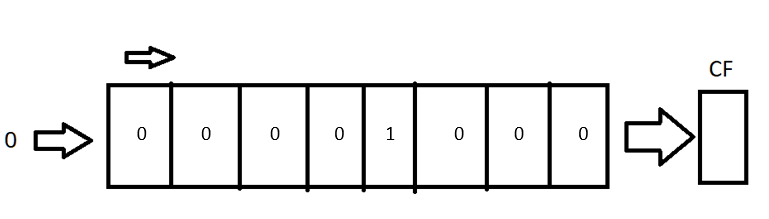
2)

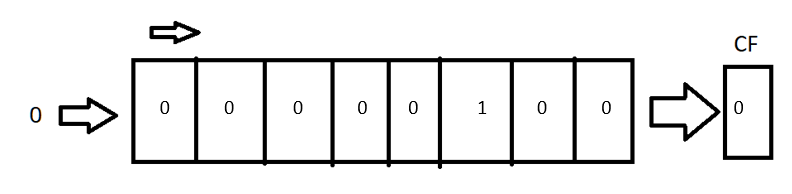
Shr al,1

הפקודה shr al,1 תפקידה להזיז את האופרנד פעם אחת ימינה. זאת אומרת 0 לוגי נכנס מצד שמאל וכל הסיביות זזות ימינה פעם אחת. כמו כן ערך סיבית ה-LSB נופלת ל-CF. גם כאן אם רוצים להזיז יותר מפעם אחת יש להשתמש באוגר CL בלבד.

שימושים אפשריים לפקודה זו:

1. חילוק של האופרנד ב
2. בדיקת סיבית ה-LSB





פקודות הזזה אריטמטיות

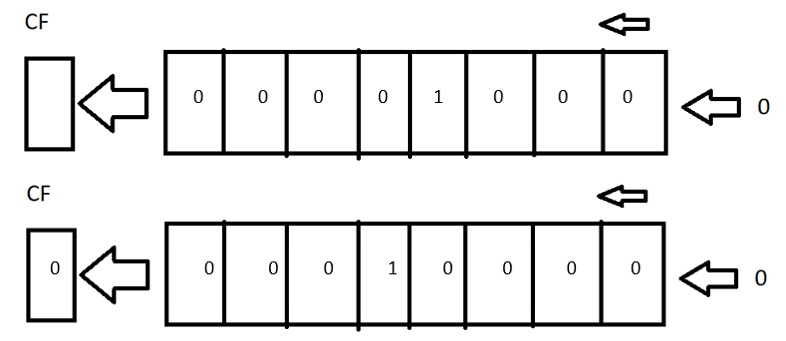
הזזה שמאלה:

Sal al,1

הפקודה sal al,1 תפקידה להזיז את הסיביות שנמצאות באופרנד היעד פעם אחת שמאלה. זאת אומרת 0 לוגי נכנס מצד ימין כל הסיביות זזות פעם אחת שמאלה. כמו כן ערך סיבית ה-MSB מעדכנת את ה-CF

אם רוצים להזיז יותר מפעם אחת נעביר ל-CL את הערך שרוצים ואז נרשום sal AL,CL (נעבור דרך CL בלבד.

שימושים אפשריים לפקודה זו הם:

1. כפל של האופרנד ב
2. בדיקת סיבית ה-MSB כתוצאה מבדיקת ה-CF

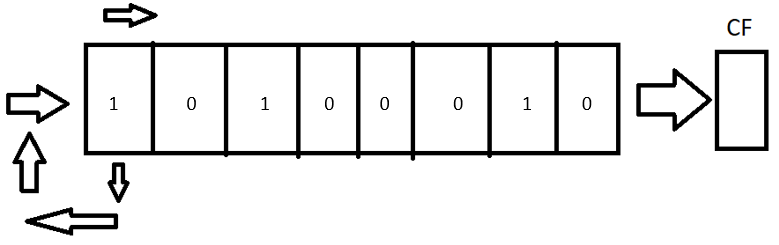
הזזה ימינה:

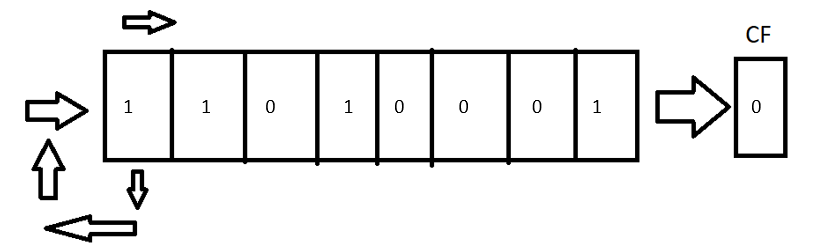
Sar al,1

הזזה אריתמטית ימינה אנו משכפלים את סיבית ה-MSB ובכך שומרים על ערכה של סיבית הסימן. אם עובדים **עם** מספרים מסומנים ורוצים לבצע הזזה ימינה, סיבית ה-MSB מסמנת האם המספר הוא חיובי או שלילי ולכן כדי לא לאבד את ערכה אנו משכפלים את סיבית ה-MSB ובכך מקבלים במידת הצורך חלוקה של האופרנד ב על פי הדוגמא הבאה:

עליון שווה -97

תחתון שווה -47





\*במספר שלילי (מספר משמאל הוא 1) יש לבצע שיטת המשלים ל-2, הופכים את כל הספרות להופכי(0 הופך ל1 ו-1 הופך ל0), ומוסיפים 1. התוצאה תהיה המספר החיובי.

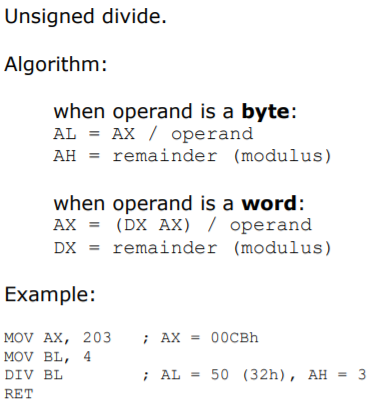
פקודות כפל וחילוק

Mul-פקודת כפל עבור מספרים לא מסומנים

Imul-פקודת כפל עבור מספרים מסומנים

Div-פקודת חילוק למספרים לא מסומנים

Idiv-פקודת חילוק למספרים מסומנים



פקודת חילוק למספרים בגודל BYTE

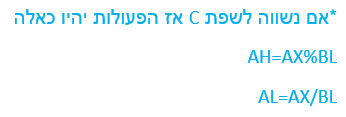


הפעולה div תמיד תעשה חילוק בעזרת AX

התוצאה תיכנס ל AL והשארית תיכנס לAH

מה תחזיר הפקודה הבאה?





נוסחת חישוב מודולו במחשבון

שארית=Ax-(num\*ah)

מחולק-AX

מחלק – num

תוצאה-ah

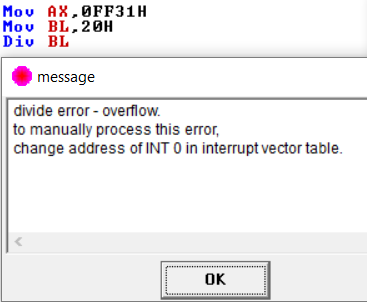
פקודת חילוק למספרים בגודל WORD

Div cx= ( dx & ax ) / cx



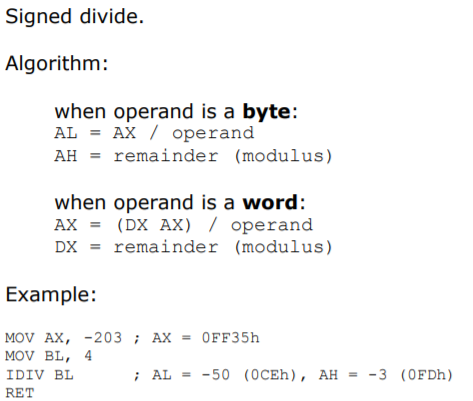
משרשרים אותם אחד לשני במקרה הזה נבצע את הפעולה הבאה

00300040H / 15F

נתון קטע הקוד הבא רשום את ערכו של AX בסוף התוכנית

AL=7F9

**\*\*\*\*\*במקרה זה תוצאת החילוק 7F9H שאמורה להיכנס לאוגר AL גדולה מ-8 ביט ולכן בעת הרצת התוכנית נקבל הודעה על חילוק שגוי כפי שמוצג בתמונה למעלה\*\*\*\*\***

פעולת חילוק למספרים מסומנים

נמיר את המספרים לבינארי, במידה ואחד המספרים שלילי

נעשה לו המשלים ל2 כדי למצוא איזה מספר זה בדצימלי,

לאחר מכן נבצע פעולת חילוק על המספרים הדצימליים

נכניס את התוצאה לAL ואת השארית ל AH ונהפוך אותם

להקסה דצימלי



1111111100110001=0FF31H

1111111100110001 0000000011001110 0000000011001111=207

1+

0000000011001111 1111111100110001=-207

20H=32

נבצע חילוק בין -207 ל 32 את התוצאה נמיר להקסה דצימלי ונכניס לAL

-207/32=-6 AL=-6 AL=FAH

כעת נבצע את פעולת המודולו את התוצאה נמיר להקסה דצימלי ונכניס לAH

-207-(32\*-6)=-15 AH=-15 AH=F1H

פקודת כפל עבור מספרים בגודל BYTE



קבוע יהיה AL

AL\*CH

תוצאה תיכנס לAX

דוגמא:



7A\*E1=6B3AH

AX=6B3AH



AX=0006H

פקודת כפל עבור מספרים בגודל WORD



קבוע יהיה AX

AX\*BX

תוצאה-> (AX ו DX)

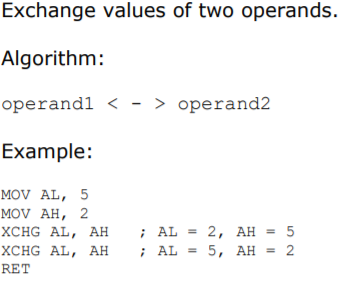
DX=MSB

AX=LSB



Ax=9c5fH

DX=0053H

הפקודה XCHG

XCHG AL,BL

מחליפה בין אופרנד המקור לאופרנד היעד

כתוב תוכנית להגדרת ארבעה משתנים בגודל בייט בסגמנט הנתונים בזיכרון num,ahadot,asarot,meot.

התוכנית תיקח את הנתון מהכתובת שהמשתנה NUM מצביע עליה בזיכרון ותפרק אותו למאות עשרות ואחדות ואת התוצאה יש להציב בהתאמה במשתנים.

