

פרק 7 פקודות אריתמטיות, לוגיות והזזה

תוכן הפרק

פקודות אריתמטיות ▶

ADD, INC ◦

SUB, DEC ◦

MUL ◦

DIV ◦

פקודות לוגיות ▶

AND ◦

OR ◦

XOR ◦

NOT ◦

פקודות הזזה ▶

SHL ◦

SHR ◦

פקודת ADD

- ▶ מחברת את אופרנד המקור עם אופרנד היעד
- ▶ התוצאה נשמרת באופרנד היעד

תוצאה	דוגמה	הפקודה
$ax = ax + bx$	add ax, bx	add register, register
$ax = ax + var1$	add ax, var1	add register, memory
$ax = ax + 2$	add ax, 2	add register, constant
$var1 = var1 + ax$	add var1, ax	add memory, register
$var1 = var1 + 2$	add var1, 2	add memory, constant

תרגילים - ADD

- ▶ צרו מערך בן 4 בתים. בתחילת התוכנית הזינו לתוכו ארבעה ערכים- הזינו ערכים קטנים (פחות מ-50). חשבו את סכום האיברים במערך לתוך רגיסטר כלשהו.
 - כעת שימרו במערך ערכים שכולם גדולים מ-100. האם התוכנית עובדת? מה צריך לשנות בה?
- ▶ הגדירו משתנים: `var1`, `var2` בגודל בית ומשתנה `sum`. שימרו את סכום `var1`, `var2` בתוך `sum`.

פקודת SUB

- ▶ קיצור של Subtract
- ▶ מחסרת את אופרנד המקור מאופרנד היעד

תוצאה	דוגמה	הפקודה
$ax = ax - bx$	sub ax, bx	sub register, register
$ax = ax - var1$	sub ax, var1	sub register, memory
$ax = ax - 2$	sub ax, 2	sub register, constant
$var1 = var1 - ax$	sub var1, ax	sub memory, register
$var1 = var1 - 2$	sub var1, 2	sub memory, constant

▶ צרו שלושה מערכים בני 4 בתים. אתחלו את שני המערכים הראשונים עם ערכים כלשהם. הכניסו לתוך המערך השלישי את החיסור של שני המערכים הראשונים (לדוגמה המערך הראשון 6,7,8,9 המערך השני 9,8,7,6 המערך השלישי הוא 6,7,8,9. חיסור המערכים הוא $(-3, -1, 1, 3)$)

פקודות INC, DEC

- ▶ בעקבות הנפוצות של פעולות הוספת / חיסור 1, הוגדרו פקודות מיוחדות
- INC - קיצור של Increase
 - DEC - קיצור של Decrease

תוצאה	דוגמה	הפקודה
$ax = ax + 1$	inc ax	inc register
$var1 = var1 + 1$	inc var1	inc memory
$ax = ax - 1$	dec ax	dec register
$var1 = var1 - 1$	dec var1	dec memory

MUL פקודת

▶ MUL - קיצור של Multiply

▶ כאשר מבצעים כפל בין שני אופרנדים, שמירת התוצאה

עשויה לדרוש כמות ביטים גדולה יותר

◦ כפל של אופרנדים בגודל 8 ביט - < 16 ביטים

◦ כפל של אופרנדים בגודל 16 ביט - < 32 ביטים

תוצאה	דוגמה	הפקודה
$ax = al * bl$	mul bl	mul register (8 bit)
$dx:ax = ax * bx$	mul bx	mul register (16 bit)
$ax = al * \text{ByteVar}$	mul ByteVar	mul memory (8 bit)
$dx:ax = ax * \text{WordVar}$	mul WordVar	mul memory (16 bit)

כפל אופרנדים בגודל 8 ביט

▶ האסמבלר ישמור את התוצאה ב-ax

▶ דוגמה:

◦ al=0ABh

◦ bl=10h

◦ ax=al*bl=0AB0h

• ah=0Ah

• al=0B0h

כפל אופרנדים בגודל 16 ביט

האסמבלר ישמור את התוצאה ב- $ax:dx$ ▶

דוגמה: ▶

$ax = 0AB0h$ ◦

$bx = 1010h$ ◦

$dx:ax = ax * bx = 0ABAB00h$ ◦

$dx = 0ABh$ •

$ax = 0AB00h$ •

בעיית חשיבה

▶ כמה זה ב 11111011 כפול ב 00000010 ?

▶ בשיטת המשלים ל-2:

◦ 11111011 שווה 251 או -5

◦ 00000010 שווה 2

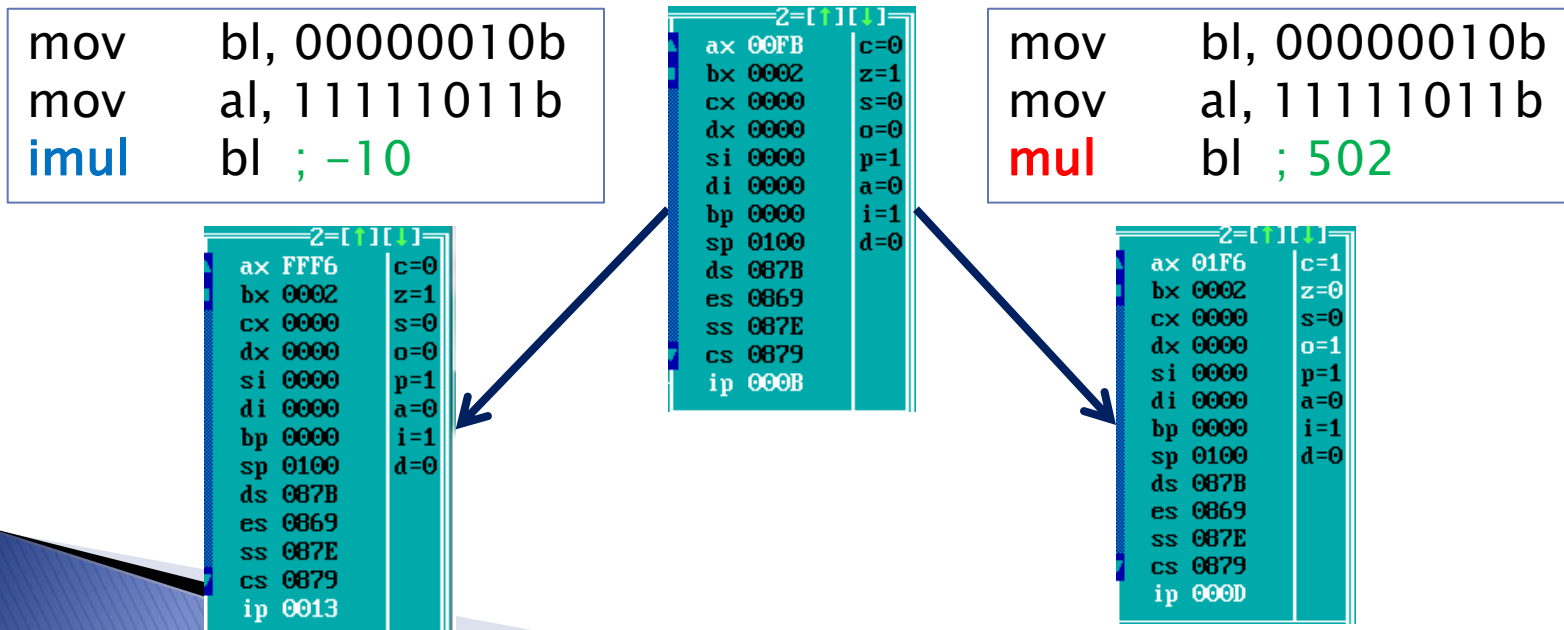
◦ 00000010 אינו שווה -254

• מחוץ לתחום המוגדר ע"י 8 ביט signed

▶ אז מה התוצאה?

כפל signed, unsigned

- אנו חייבים להודיע לאסמבלר האם להתייחס לערכים כ-
- signed או unsigned
 - פקודת unsigned-MUL
 - פקודת signed-IMUL



▶ הגדירו שני מערכים, בכל מערך 4 ערכים מסוג בית, signed. שימו בהם ערכים התחלתיים. בצעו כפל של המערכים והכניסו את התוצאה לתוך sum.
◦ לטובת הפשטות, הניחו שהתוצאה נכנסת ב-word

◦ הדרכה: אם שמות המערכים הם a ו-b, אז

$$\text{sum} = a[0]*b[0]+a[1]*b[1]+...$$

פקודת DIV

▶ קיצור של Divide

▶ כאשר מבצעים חילוק באופרנדים, יש צורך לשמור הן את המנה והן את השארית

- אופרנדים של 8 ביט: המנה ב-`al`, השארית ב-`ah`
- אופרנדים של 16 ביט: המנה ב-`ax`, השארית ב-`dx`

תוצאה	דוגמה	הפקודה
$al = ax \text{ div } bl$ $ah = ax \text{ mod } bl$	div bl	div register (8 bit)
$ax = dx:ax \text{ div } bx$ $dx = dx:ax \text{ mod } bx$	div bx	div register (16 bit)
$al = ax \text{ div ByteVar}$ $ah = ax \text{ mod ByteVar}$	div ByteVar	div memory (8 bit)
$ax = dx:ax \text{ div WordVar}$ $dx = dx:ax \text{ mod WordVar}$	div WordVar	div memory (16 bit)

פקודת DIV - שאלה למחשבה

▶ נניח $ax=10h$, $bx=2h$.

▶ מה תהיה תוצאת הפקודה הבאה?

`div bx`

▶ התשובה תלויה בערך של dx !

◦ לפני חלוקה של 16 ביט יש לוודא שב- dx אין ערך אקראי

◦ לפני חלוקה של 8 ביט יש לוודא שב- ah אין ערך אקראי

```

IDEAL
MODEL small
STACK 100h
DATASEG
CODESEG
start:
    mov     ax, @data
    mov     ds, ax
    mov     al, 7
    mov     bl, 2
    mov     ah, 0
    div     bl
    mov     ax, 7
    mov     dx, 0
    mov     bx, 2
    div     bx
quit:
    mov     ax, 4c00h
    int     21h
END start
  
```

▶ מה יהיו ערכי
 הרגיסטרים בזמן ריצת
 התוכנית, לאחר הרצת
 הפקודות המודגשות?

חילוק signed / unsigned

- ▶ פקודת div משמשת לחילוק מספרים unsigned
- ▶ פקודת idiv משמשת לחילוק מספרים signed

פקודת NEG

- ▶ קיצור של Negative
- ▶ מציאת המשלים ל-2 של האופרנד

תוצאה	דוגמה	הפקודה
al = 0 - al	neg al	neg register (8 bit)
ax = 0 - ax	neg ax	neg register (16 bit)
ByteVar = 0 - ByteVar	neg ByteVar	neg memory (8 bit)
WordVar = 0 - WordVar	neg WordVar	neg memory (16 bit)

פקודות לוגיות

- ▶ שימושיות כשרוצים לשנות ביטים בודדים
- ▶ מתי רוצים לעבוד עם ביטים בודדים?
 - "דחיסה" של מידע
 - הצפנה

00000000 00000001 00000001 00000001 00000000 00000000 00000001 00000000



01110010

מידע דחוס:



00100111

מידע מוצפן- כל ביט
שני הפוך:

פקודות לוגיות

and / or/ xor	register, register
and / or/ xor	memory, register
and / or/ xor	register, memory
and / or/ xor	register, constant
and / or/ xor	memory, constant
not	register
not	memory

סט הפקודות: ▶

AND ◦

OR ◦

XOR ◦

NOT ◦

להלן צורות ▶

הכתיבה החוקיות.

פקודת AND

- ▶ מקבלת כקלט שני אופרנדים
- ▶ מבצעת את הפעולה על כל ביט בנפרד לפי טבלת האמת

AND	1	0
1	1	0
0	0	0

```

0000 0111 and
1001 0110
-----
0000 0110
    
```

פקודת AND - חידות

▶ איך אפשר לבדוק באמצעות פקודת AND:

- אם מספר (8 ביט) הוא זוגי?
- אם מספר (8 ביט) מתחלק ב-4?
- אם מספר signed (8 ביט) הוא שלילי?

▶ מסכה (MASK) אוסף של ביטים שמאפשר לנו לבודד

ולפעול על ביטים מסויימים

- מסכה לבדיקה אם מספר הוא זוגי 0000 0001
- פעולת AND של מספר זוגי עם המסכה תיתן תמיד 0
- פעולת AND של מספר אי זוגי עם המסכה תיתן תמיד 1

► מקבלת שני אופרנדים ופועלת על כל ביט לפי טבלת האמת

OR	1	0
1	1	1
0	1	0

► שימושית כשרוצים "להדליק" ביט

```

0100 0111 or
0001 0000
-----
0101 0111
    
```

פקודת XOR

▶ מקבלת שני אופרנדים ופועלת על כל ביט לפי טבלת האמת

XOR	1	0
1	0	1
0	1	0

▶ שקולה לחיבור מודולו 2

▶ XOR של רצף ביטים עם רצף ביטים זהה תמיד שווה אפס

◦ תכונה שימושית להצפנה

◦ `xor ax, ax ; mov ax, 0`

פקודת XOR- ביצוע הצפנה

מסר: 1001 0011

מפתח: 0101 0100

1001 0011 xor
0101 0100

מסר מוצפן: 1100 0111

1100 0111 xor
0101 0100

מסר מפוענח: 1001 0011

- ▶ ברשותנו מסר שאנו רוצים להצפין
- ▶ נגדיר מפתח הצפנה
- ▶ המסר המוצפן הוא XOR בין המסר לבין מפתח ההצפנה
- ▶ פענוח המסר בעזרת XOR נוסף עם מפתח ההצפנה
 - ללא המפתח אי אפשר לפענח את המסר

תרגיל XOR

```
; print msg
```

```
mov dx, offset msg
```

```
mov ah, 9h
```

```
int 21h
```

```
mov ah, 2
```

```
; new line
```

```
mov dl, 10
```

```
int 21h
```

```
mov dl, 13
```

```
int 21h
```

```
C:\TASM\BIN\CH6>xor 1
?U:?=3U7?/?3:4:/
I LIKE ASSEMBLY
C:\TASM\BIN\CH6>
```

- ▶ הגדירו מערך בשם msg שהוא אוסף של תווי ASCII. לדוגמה 'I LIKE ASSEMBLY\$'.
 - תו ה-\$ משמש להדפסת המחרוזת
- ▶ הגדירו מפתח הצפנה בן 8 ביטים, לבחירתכם.
- ▶ הצפינו את המסר בעזרת מפתח ההצפנה
- ▶ פענחו את המסר בעזרת מפתח ההצפנה.
- ▶ בידקו ב-DATASEG את המסר לאחר ההצפנה ולאחר הפענוח.
- ▶ למעוניינים להדפיס את המסר ואת המסר המוצפן, ניתן להשתמש בקוד הבא.

פקודת NOT

▶ פקודת NOT הופכת את כל הביטים באופרנד

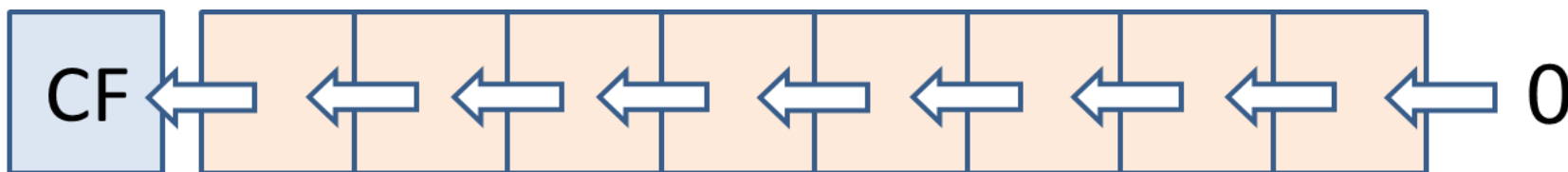
NOT	
1	0
0	1

פקודות הזזה

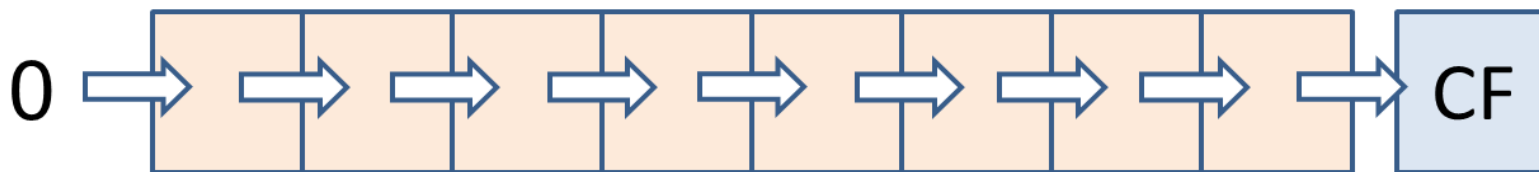
- ▶ מקבלות אופרנד ו"מזיזות" את הביטים שלו
 - SHR– Shift Right
 - SHL– Shift Left
- ▶ שימושים (נפרט בהמשך):
 - כפל וחילוק
 - תיקון שגיאות והצפנה
 - דחיסה ופריסה של מידע
 - כתיבה לזיכרון הוידאו
- ▶ להלן צורות כתיבה חוקיות.

shl / shr	register, const
shl / shr	register, cl
shl / shr	memory, const
shl / shr	memory, cl

- ▶ מזיזה מקום אחד שמאלה את כל הביטים
- ▶ דוחפת 0 לביט הימני
- ▶ מעתיקה לדגל הנשא את הביט הימני
- ▶ מבצעת את השלבים הנ"ל כמספר הפעמים שהוגדר בפקודה



- ▶ מזיזה מקום אחד ימינה את כל הביטים
- ▶ דוחפת 0 לביט השמאלי
- ▶ מעתיקה לדגל הנשא את הביט הימני
- ▶ מבצעת את השלבים הנ"ל כמספר הפעמים שהוגדר בפקודה



שימוש בפקודות הזזה לכפל וחילוק

- ▶ ביצוע בקלות של פעולות כפל וחילוק בחזקות של 2
 - דוגמה: חלוקה ב-2 של המשתנה counter (unsigned) בגודל 8 ביטים)

```
shr    [counter], 1
```

פקודת shr

```
mov    al, [counter]
xor    ah, ah
mov    [divider], 2
idiv   [divider]
mov    [counter], al
```

פקודת idiv

תרגילים - פקודות הזזה

- ▶ הכניסו ל-al את הערך 3. בעזרת פקודות הזזה, כיפלו את al ב-4.
- ▶ הכניסו ל-al את הערך 120 (דצימלי). בעזרת פקודות הזזה, חלקו את al ב-8.
- ▶ הכניסו ל-al את הערך 10 (דצימלי). בעזרת פקודות הזזה וחיבור, כיפלו את al ב-20.
- הדרכה: התייחסו ל-20 בתור סכום של 16 ו-4. השתמשו ברגיסטרים נוספים כדי לשמור חישובי ביניים.

שימוש בפקודות הזזה בתיקון שגיאות ובהצפנה

▶ נושאים ללימוד עצמי:

◦ הצפנת LFSR

• http://en.wikipedia.org/wiki/Linear_feedback_shift_register

◦ קוד קונבולוציה לתיקון שגיאות

• http://en.wikipedia.org/wiki/Convolutional_code

- ▶ בפרק זה למדנו מגוון פקודות:
 - פקודות אריתמטיות
 - פקודות לוגיות
 - פקודות הזזה
- ▶ סקרנו שימושים שונים של הפקודות
- ▶ עדיין חסרה לנו יכולת לכתוב אלגוריתמים ותנאים
 - בפרק הבא