מודל מעבד-זיכרון

SFS: משתנה המציין את הפעולה שרוצים לבצע.
ערכים:

- **00** – כתיבה (store)
- **01** – קריאה (fetch)
- **10** – הפקודה בוצעה
- **11** – תקלה

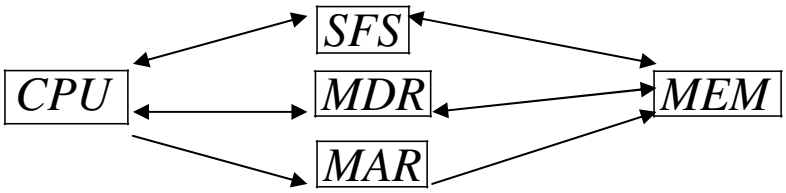
MDR: זהו המקום שהמעבד רושם בו את **התוכן** שהוא רוצה להכניס לזיכרון (במקרה store),
וזהו המקום שהזיכרון רושם בו **תוכן כתובת** כלשהי (במקרה שהמעבד ביקש fetch).

- MDR מחזיק את תוכן הכתובת היושבת ב- MAR

MAR: זהו המקום שהמעבד רושם בו את **הכתובת אליה** הוא רוצה **לכתוב** (במקרה store),
וזהו המקום שהמעבד רושם בו את **הכתובת ממנה** הוא רוצה **לקרוא** (במקרה fetch).

שם המשתנה	רוחב הקו בביטים בין המשתנה למעבד ב- PDP-11
SFS	2
MDR	16
MAR	16

כיווני העברת המידע במודל מעבד-זיכרון



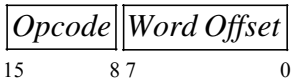
$$CPU \leftrightarrow SFS \leftrightarrow MEM$$
$$CPU \leftrightarrow MDR \leftrightarrow MEM$$
$$CPU \rightarrow MAR \rightarrow MEM$$

Mode	Name	Syntax	EA	Comments
0	Register	**Rn**	-	האופרנד נמצא ברגיסטר
1	Register Deferred	**(Rn),@Rn**	Rn	כתובת האופרנד נמצאת ברגיסטר
2	Auto Increment	**(Rn)+**	Rn	- כתובת האופרנד נמצאת ברגיסטר - בתום החישוב, הוסף 2 ל- Rn
3	Auto Increment Deferred	**@(Rn)+**	(Rn)	- הכתובת של כתובת האופרנד נמצאת ברגיסטר. - בתום החישוב, הוסף 2 ל- Rn - **מקודם ב-2 גם עבור movb**
4	Auto Decrement	**-(Rn)**	Rn-2	- לפני החישוב, הורד 2 מ- Rn - כתובת האופרנד נמצאת ברגיסטר
5	Auto Decrement Deferred	**@-(Rn)**	(Rn-2)	- לפני החישוב, הורד 2 מ- Rn - הכתובת של כתובת האופרנד נמצאת ברגיסטר. - **מקודם ב-2 גם עבור movb**
6	Index (אבסולוטי)	**X(Rn)**	X+ Rn	- האופרנד בכתובת X+Rn *instruction* *X* - לאחר שליפת X, PC מקודם ב-2.
7	Index Deferred (אבסולוטי)	**@X(Rn)**	(X+Rn)	- כתובת האופרנד בכתובת X+Rn *instruction* *X* - לאחר שליפת X, PC מקודם ב-2.
pc MODES				
27	Immediate (אבסולוטי)	**#K**	PC (לאחר קריאת הפקודה)	- K נמצא בכתובת המוצבעת ע"י PC *instruction* *K* - אם K הינה תווית, לך אל כתובתה
37	Absolute (אבסולוטי)	**@#K**	K	- האופרנד נמצא בכתובת K *instruction* *K*
67	Relative	**A**	A	- האופרנד נמצא בכתובת A l : *instruction* $A-m-2$ - m היא כתובת האופרנד בתוך ההוראה $l+2 \le m \le l+4$ - $A-m-2$ הוא המרחק בין EA ובין כתובת המלה הבאה לביצוע.
77	Relative Deferred	**@A**	(A)	- כתובת האופרנד נמצאת בכתובת A l : *instruction* $A-m-2$

<u>הערה:</u> השיטות המוקפות בעיגול מצריכות מלת זיכרון נוספת.

פקודות branch

קידוד:



WO- מספר המלים שיש לקפוץ מערכו הנוכחי של ה- PC, **לאחר קריאת הפקודה.**

$$\boxed{WO=\frac{destination-PC}{2}}$$

<u>הערה:</u> מאחר שמוקצים רק 8 ביטים לקידוד ה- Offset, ניתן לקפוץ רק $2^8-1=256$ מילים, לכל היותר (עבור 11111111). בעצם, רק חצי מכך, אם רוצים לכלול גם קפיצה לאחור.

<u>חישוב ה- Opcode הסופי:</u>
נעשה ע"י חיבור ה- Opcode של פקודת ההסתעפות המתאימה עם W0, כאשר האחרון מיוצג באמצעות 8 ביטים.

<u>דוגמא:</u>

1000 $br \quad here$

$\vdots$

1020 $here: mov\#3, r0$

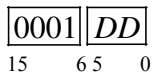
$$W0=\frac{1020-1002}{2}=\frac{16}{2}=asr(16)=7=00000111$$

$$\Rightarrow Opcode = \begin{array}{l} 0000000100000000 \\ + \\ \quad 0000000000000111 \\ \hline 0000000100000111 \end{array} = (000407)_8$$

פקודת jmp

$jmp \ A$

קידוד:



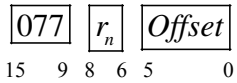
כאשר DD נקבע עפ"י שיטת המיעון של A.

- הקפיצה הינה אל הכתובת האפקטיבית !
- מרחק הקפיצה אינו מוגבל

פקודת sob

$$A:\ sob\ rn,l$$

קידוד:



Offset- מספר חיובי, בטווח 0-63, המציין את מספר המלים שיש לקפוץ אחורה

$$\boxed{Offset = \frac{A-l}{2}+1}$$

דוגמא:

```
1000    here: mov #3,r0
                 ⋮
1020          sob r1,here
```

$$Offset=\frac{1020-1000}{2}+1=11 \Rightarrow Opcode=\boxed{077|1|11}$$

פקודת jsr ופקודת rts

$jsr\ r_n,subr$

$\boxed{004|r_n|DD}$

1). $mov\ r_n,-(sp)$

2). $mov\ pc,r_n$

3). $mov\ \#subr,pc$

$rts\ r_n$

$\boxed{00020|r_n}$

1). $mov\ r_n,pc$

2). $mov\ (sp)+,r_n$

ה- PSW

$$\boxed{|||||||priority|T|N|Z|V|C}$$

5-7 4 3 2 1 0

חוקי המכונה:

- דגל C נדלק אמ"מ **יש נשא בחיבור** מה-MSB
- דגל C נדלק אמ"מ **אין נשא בחיסור** מה- MSB
- דגל V נדלק אמ"מ שני המחוברים/מחוסרים בעלי MSB זהה, והתוצאה בעלת MSB הפוך.

או לחילופין:

- דגל C נדלק אם יש טעות בחישוב, כאשר האופרנדים ללא סימן.
- דגל V נדלק אם יש טעות בחישוב, כאשר האופרנדים עם סימן.

פקודות ההסתעפות

Type	Opcode	Mnemonic	Name	Comments
Signed	002400	**blt**	branch if less than	$(N \wedge (\sim V)) \vee (\sim N \wedge V) \Leftrightarrow N \oplus V = 1$
	003400	**ble**	branch if less than or same	$(N \oplus V) \vee Z$
	003000	**bgt**	branch if greater than	$\sim ((N \oplus V) \vee Z)$
	002000	**bge**	branch if greater than or same	$\sim (N \oplus V) \Leftrightarrow N = V$
Unsigned	101000	**bhi**	branch if higher	~ (CvZ)
	103000	**bhis**	branch if higher or	~C
	103400	**blo**	same branch if lower	C
	101400	**blos**	branch if lower or same	(CvZ)
Z&V branches	001400	**beq**	branch if equal	Z
	001000	**bne**	branch if not equal	~ Z
	100400	**bmi**	branch if minus	N
	100000	**bpl**	branch if plus	~ N
C&V branches	103400	**bcs**	branch if carry set	C
	103000	**bcc**	branch if carry clear	~ C
	102400	**bvs**	branch if overflow set	V
	102000	**bvc**	branch if overflow clear	~ V
unconditional	000400	**br**	branch	

OpCodes של פקודות נפוצות

Mnemonic	OpCode
add	06
asl	0063
asr	0062
bic	04
bit	03
bis	05
clr	0050
cmp	02
dec	0053
div	071RSS
inc	0052
mov	01
mul	070RSS
neg	0054
rti	000002
rtt	000006
sub	16
sxt	0067
tst	0057

טבלת עדיפויות

עדיפות	ערך אוקטלי
1	40
2	100
3	140
4	200
5	240
6	300
7	340

פסיקות

פסיקה תתבצע כאשר יתקיימו כל התנאים הבאים:

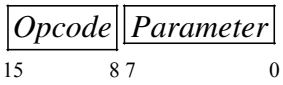
- ההתקן פעיל (read enable = 1)
- ההתקן ביקש פסיקה (ready/done = 1)
- ההתקן רשאי לפסוק (interrupt enable = 1)
- עדיפות החומרה של ההתקן הפוסק גדולה ממש מעדיפות התוכנה ברגע קבלת הפסיקה

התקן	כתובת אוגר הבקרה	כתובת אוגר הנתונים	עדיפות חומרה	וקטורי פסיקה	ביטים "מעניינים" באוגר הבקרה
מקלדת	TKS=177560	TKB=177562	4	60/62	11 busy; 7 done; 6 interrupt enable; 0 read enable
מסך	TPS=177564	TPB=177566	4	64/66	7 ready; 6 interrupt enable; 0 read enable
שעון	CLS=177546	-	6	100/102	6 interrupt enable

וקטורי פסיקה	Error	הערות
4/6	Odd Address	
4/6	Stack Violation	כתיבה לכתובות נמוכות מ-376
4/6	Stack Warning	כתיבה לכתובות נמוכות מ-420
10/12	Illegal Opcode	
24/26	Power Failure	

פסיקה	וקטורי פסיקה	OpCode
bpt	14/16	000003
iot	20/22	000004
emt n	30/32	104000-104377
trap n	34/36	104400-104777

מבנה הפקודה של emt/trap:



קפיצה לשגרת טיפול בפסיקה

1). $mov\ psw, -(sp)$
2). $mov\ pc, -(sp)$
3). $mov\ @\#k, pc$
4). $mov\ @\#k+2, psw$

חזרה משגרת טיפול בפסיקה

1). $mov\ (sp)+, pc$
2). $mov\ (sp)+, psw$

כאשר k הינה הכתובת הראשונה בוקטור הפסיקה.

האסמבלר

תוכנה המתרגמת תוכנית בשפת אסמבלי לשפת מכונה.
LC - משתנה פנימי של האסמבלר, המצביע בכל רגע נתון על הפקודה שהאסמבלר מטפל בה.
זוהי הכתובת שהחל ממנה תיטען הפקודה שהאסמבלר עומד לייצר. **קיים רק בזמן תרגום!**
PC – מצביע על הפקודה הבאה לביצוע. **קיים רק בזמן ריצה!**

תהליך התרגום בשני מעברים:

מעבר ראשון:
- חישוב מספר המלים שצורכת כל פקודה
- בדיקת הסינטקס
- יצירת טבלת הסמלים

בסוף מעבר ראשון נבדקת עקביות טבלת הסמלים. כלומר, האם כל הסמלים שמשתמשים בהם בתוכנית אכן מוגדרים, ואין סמלים כפולים. בתום מעבר ראשון, אם לא התגלתה טעות, טבלת הסמלים שלמה.

מעבר שני:
- חישוב כל האופרנדים: המידיים (כמו #2000) והמסתמכים על תוויות (כמו #A)
- יצירת קוד המכונה.

הכתובות וערכי התוויות לחישוב האופרנדים נלקחים מטבלת הסמלים.

טעויות תרגום:

הטעות	מתגלה במעבר	הערות ודוגמאות
תווית מוגדרת פעמיים	ראשון	
פקודה לא חוקית	ראשון	למשל, פקודה שאינה קיימת
מספר אופרנדים שגוי	ראשון	add #2
שימוש בתווית לא מוגדרת	שני	על אף שטבלת הסמלים שלמה בסוף מעבר ראשון
קפיצה רחוקה מדי	שני	תקף לפקודות הסתעפות
ביטוי שגוי	שני	add #88,r0
ביצוע sob קדימה	שני	חישוב האופרנדים נעשה במעבר שני

המידע הנשלח אל הקשר-טען ע"י האסמבלר:

- אורך של כל מודול.
- רשימת כל הסמלים שיש לבצע עליהם relocation (לא רלבנטי לקוד PIC).
- רשימת ה- entry של כל מודול- רשימת כל הסמלים המוגדרים במודול, ושניתן להשתמש בהם במודולים אחרים.
- רשימת ה- extrn של כל מודול – רשימת כל הסמלים המוגדרים במודולים אחרים, ושנעשה בהם שימוש במודול הנוכחי.
- המודול עצמו- קוד המכונה.

קישור וטעינה

הנחות אסמבלר:

- כתובת תחילת הטעינה של כל מודול הינה 0.
- ערכם של הסמלים החיצוניים במודול הינו 0.

טבלת ESD:

טבלת מידע על הסמלים המיוחדים במודול - כאלה שהוגדרו ע"י ההנחיות .extrn , .entry , .csesct

Relative Location	Type	Symbol
הכתובת היחסית של כל סמל – מרחקו מתחילת ה- csect הנוכחי. סמלים חיצוניים לא ידועים בזמן אסמבלי, לכן אלה לא מקבלים כתובת יחסית.	**SD** - עבור שם המודול שהוגדר ע"י csect. **LD** – עבור הסמלים המקומיים שהוגדרו ע"י entry. **ER** – עבור הסמלים החיצוניים שהוגדרו ע"י extrn.	שם הסמל שהוגדר כמיוחד בראש המודול

טבלת RLD:

טבלת מידע על הכתובות במודול, שאותן יש לתקן/לשנות בזמן טעינה.

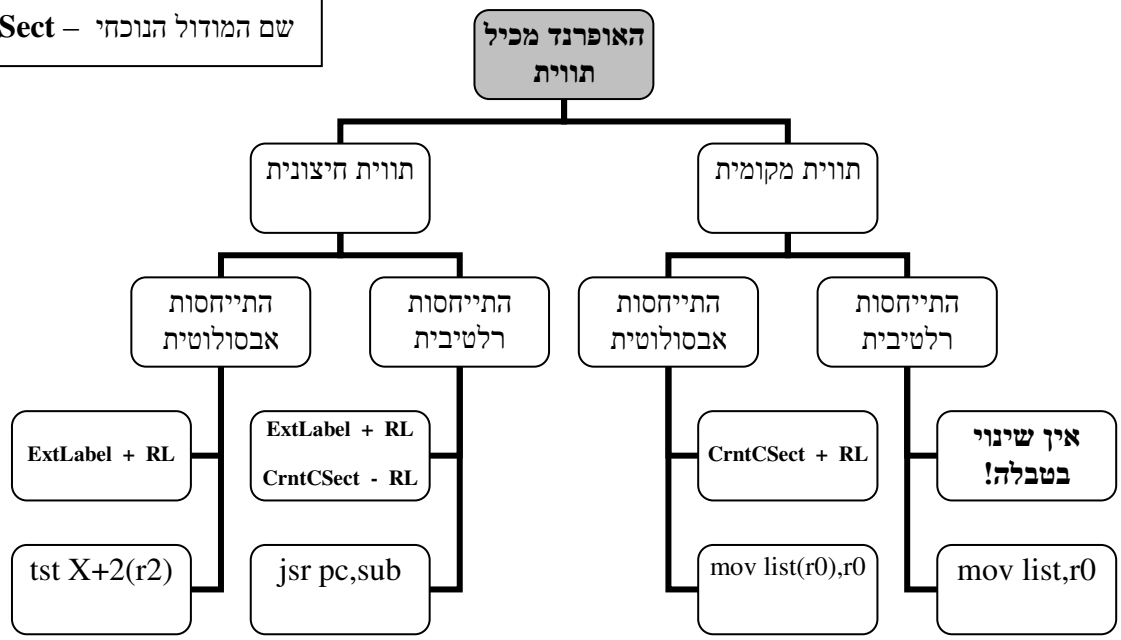
Relative Location	flag	Symbol
הכתובת היחסית של הסמל מתחילת ה- csect הנוכחי.	+ - תוספת - - הפחתה	הסמל שאת ערכו יש להוסיף/להפחית מהכתובת היחסית.

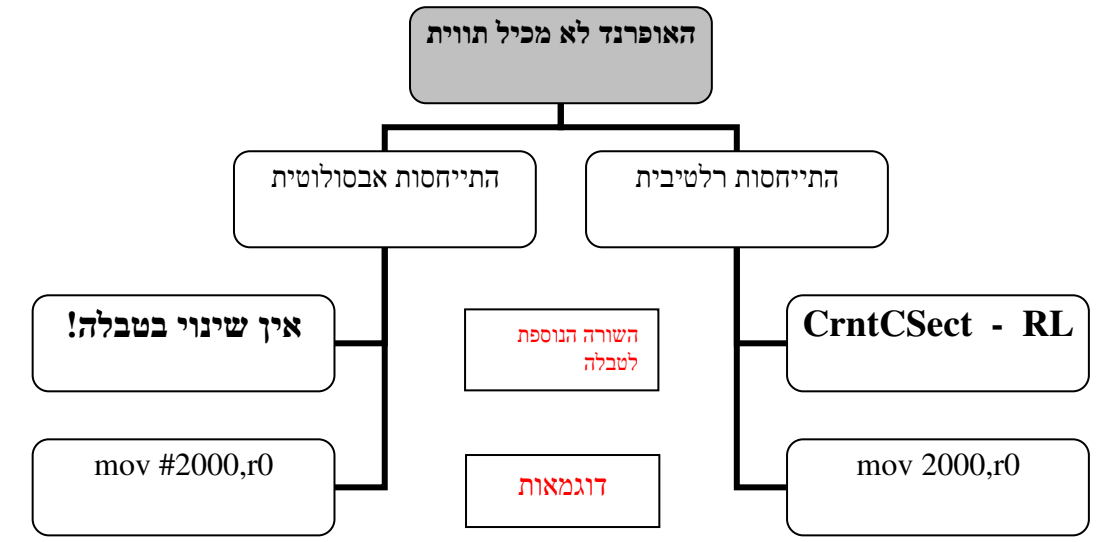
הערה חשובה: טבלת RLD נקראת מימין לשמאל!

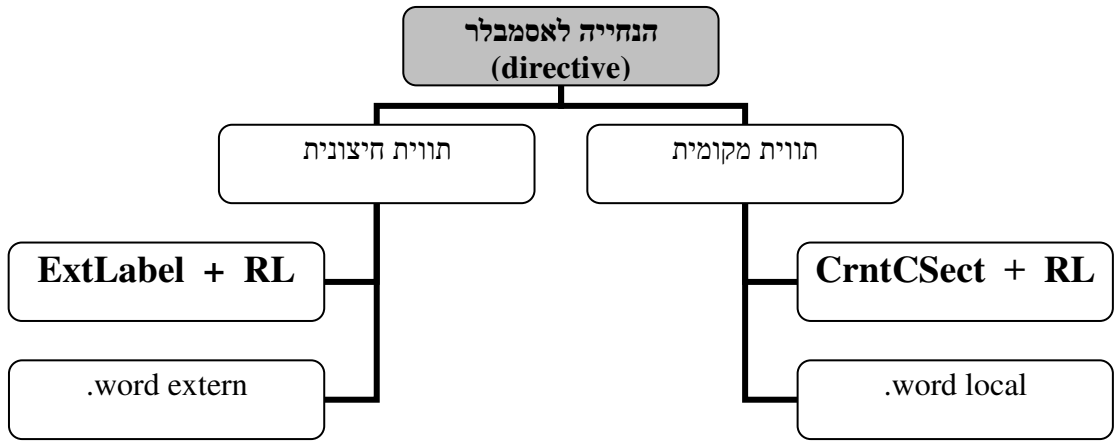
לדוגמא, עבור שורה בטבלה: sym + 02, יש לקרוא:
" לכתובת 02 יש להוסיף את ערך התווית sym".

בניית טבלת RLD

מקרא:
ExtLabel - שם התווית החיצונית
CrntCSect – שם המודול הנוכחי







קישור

קלט:
- המודולים
- טבלת ESD וטבלת RLD של כל אחד מהמודולים
- סדר הקישור

פלט:
- load module מושלם, עד כדי כתובת תחילת הטעינה
- **טבלת טעינה** – טבלת RLD מאוחדת ומצומצמת, המכילה את הכתובות שלהן יש להוסיף/להחסיר את כתובת תחילת הטעינה.

פעולות הקשר:

1). **בניית מפת טעינה** – כל המודולים מאוחדים למודול אחד (עפ"י סדר הקישור), שתחילתו בכתובת 0, וכל "מודול מתאחד" מתחיל בכתובת הזוגית הראשונה שאינה נמצאת בשימוש המודול הקודם לו, ושאליו הוא מתאחד.
למשל, עבור מודולים A,B, אם A מסתיימת בכתובת 30, אזי המודול B, המתאחד אליו, יתחיל בכתובת 32, וכל הכתובות יוזזו במספר זה.

2). **איחוד טבלות ESD:**
- בדיקה שלכל ER יש LD מתאים (אחרת, שגיאה בזמן קישור)
- בדיקה ש- LD הוא יחיד עבור ER מסוים (אחרת, שגיאה בזמן קישור)
- אם הבדיקות תקינות – מחיקת שורות ER מהטבלה
- איחוד הטבלות עפ"י העיקרון המתואר בסעיף 1). , ולפי סדר הקישור

3). **איחוד טבלות RLD:**
- איחוד הטבלות עפ"י העיקרון המתואר בסעיף 1). , ולפי סדר הקישור
- איתור הכתובות בטבלה, שעבורן יש תיקון (+) **וגם** תיקון (-)
- עבור כתובת כזו, יש לבצע את התיקונים הרשומים בטבלה בקוד הטעינה, כאשר ערכי הסמלים נלקחים מטבלת ה- ESD המאוחדת.
- מחיקת השורות הנ"ל.
- ביצוע התיקונים בקוד הטעינה עבור השורות הנותרות בטבלת ה-RLD.
- השמטת עמודת ה- Symbol מהטבלה המתקבלת (זוהי טבלת הטעינה)

בנקודה זו, סוף תהליך הקישור, ה- load module מושלם, עד כדי הזזה של הכתובות המופיעות בטבלת הטעינה בכתובת תחילת הטעינה.

טעינה

פעולות הטען:

- השג את כתובת תחילת הטעינה.
- לכל שורה בטבלת הטעינה, הוסף/החסר את כתובת תחילת הטעינה אל/מן הכתובת היחסית המתאימה כאשר ערכי הסמלים נלקחים מתוך טבלת ה-ESD המאוחדת.

דוגמא מסכמת: (מתוך מועד א' אביב תשס"ב)

נתונים שלושת המודולים הבאים:

```
000000                                  1                .csect  m1
000000                                  2                .entry  data,pd
000000                                  3                .extrn  input,print
000000                                  4
000000                                  5                TKS  =  177560
000000                                  6                TKB  =  177562
                                        7
000000  012737  000000  000060        8        main:   mov     #input,  @#60
000006  012737  000101  177560        9                mov     #101,  @#TKS
000014  005767  000010                10       loop:   tst     data
000020  001775                        11               beq     loop
000022  004067  177752                12               jsr     r0,  print
000026  000000                        13               halt
                                        14
000030  000000                        15       data:   .blkw   1
000032  000030                        16       pd:     .word   data
```

```
000000                                  1                .csect  m2
000000                                  2                .entry  input
000000                                  3                .extrn  data
                                        4
000000                                  5                MASK  =  177600
000000                                  6                TKS  =  177560
000000                                  7                TKB  =  177562
000000                                  8
000000  013701  177562                  9        input:  mov     @#TKB,  r1
000004  042701  177600                 10               bic     #MASK,  r1
000010  010137  000000                 11               mov     r1,  @#data
000014  005237  177560                 12               inc     @#TKS
000020  000002                        13               rti
```

```
000000                                  1                .csect  m3
000000                                  2                .entry  print
000000                                  3                .extrn  pd
                                        4
000000                                  5                TPS  =  177564
000000                                  6                TPB  =  177566
000000                                  7
000000  105737  177564                  8        print:  tstb    @#TPS
000004  100375                         9                bpl     print
000006  017737  177766  177566        10               mov     @pd,  @#TPB
000014  000200                        11               rts     r0
```

שלב 1 – השלמת השורות החסרות ב- listing

שם המודול	מס' שורה	ההוראה	מלה ראשונה להוספה	מלה שנייה להוספה	הסבר
m1	8	mov #input,@#60	000000	000060	**מילה ראשונה**: input הוא סמל חיצוני, לכן מקבל את הערך 0, עפ"י הנחות אסמבלר (עמ' 8) **מילה שנייה**: עפ"י שיטת מיעון 37, תוכן המלה השנייה הינו הערך הרשום באופרנד. כלומר, 60.
	9	mov #101,@#TKS	000101	-	עפ"י שיטת מיעון 27 (עמ' 2), תוכן המלה השנייה הינו הערך הרשום באופרנד: 101.
	16	pd: .word data	000030	-	data הינו סמל מקומי, לכן יש להוסיף את כתובתו בזיכרון. כלומר, 30.
m2	10	bic #MASK,r1	042701	177600	**מילה ראשונה**: ה-Opcode של bic הינו 04 (עמ' 5). אופרנד ראשון ממוען בשיטה 27, ואופרנד שני בשיטה 0 עם רגיסטר 1, לכן 04|27|01. **מילה שנייה**: MASK הינו קבוע, לכן תוכן המילה הוא ערכו. כלומר, 177600
	11	mov r1,@#data	000000	-	data סמל חיצוני, לכן מקבל 0 עפ"י הנחות אסמבלר.
m3	8	tstb @#TPS	177564	-	TPS הינו קבוע, לכן תוכן המילה הינו ערך הקבוע, עפ"י שיטת מיעון 37.
	9	bpl print	100375	-	ה-Opcode של bpl הינו 100000. print הוא סמל מקומי שכתובתו 0. ה- pc לאחר קריאת הפקודה הינו 6, לכן ה-offset הינו (עמ' 3): (0-6)/2 = -3 = 11111101 = 375 ולכן, ה-Opcode הסופי הינו הסכום – 100375.
	10	mov @pd,@#TPB	017737	-	ה-Opcode של mov הינו 01, אופרנד ראשון ממוען בשיטה 77, ואופרנד שני בשיטה 37, לכן מקבלים 01|77|37

- ההשלמות מודגשות באדום במודולים שבעמוד קודם.

שלב 2 – בניית טבלאות

טבלות ESD:

1). עפ"י הוראות בניית טבלת ESD (עמ' 8), כל שעלינו לעשות הוא לאסוף את הסמלים "המיוחדים" מכל מודול (בשלוש שורות ראשונות), ולרשום אותם בעמודה ראשונה של הטבלה.

2). לכל סמל לרשום את סוגו בעמודה שנייה בטבלה –
SD עבור סמל שהוגדר תחת ההנחיה csect. (שם המודול)
LD עבור הסמלים שהוגדרו תחת ההנחיה entry. (סמלים מקומיים)
ER עבור הסמלים שהוגדרו תחת ההנחיה extrn. (סמלים חיצוניים)

3). בעמודה שלישית יש להוסיף את הכתובת היחסית של הסמל ביחס לכתובת תחילת המודול.
מאחר שכתובת תחילת המודול הוגדרה להיות 0 (הנחות אסמבלר), הכתובת היחסית הינה הכתובת שבה מוגדר הסמל.
כמובן שאין אזכור בעמודה זו לסמלים חיצוניים, שהרי אינם ידועים במודול הנוכחי.

נקבל את הטבלות הבאות:

ESD		
Symbol	Type	Location
m1	SD	00
data	LD	30
pd	LD	32
input	ER	-
print	ER	-

ESD		
Symbol	Type	Location
m2	SD	0
input	LD	0
data	ER	-

ESD		
Symbol	Type	Location
m3	SD	0
print	LD	0
pd	ER	-

טבלות RLD:

לשם בניית טבלות אלה, ניצמד לתרשימי הבנייה המוגדרים בעמ' 9, עפ"י אופי האופרנדים.
כדי לבנות את הטבלה, עלינו לעבור על כל שורות הקוד במודול ולבחון את האופרנדים,
ולהחליט אם יידרשו בהם תיקונים מאוחר יותר.

מודול m1:

```
000000      1              .csect  m1
000000      2              .entry  data,  pd
000000      3              .extrn  print,  input
000000      4
000000      5              TKS  =  177560
000000      6              TKB  =  177562
            7
000000      8      main:   mov    #input,  @#60
000006      9              mov    #101,  @#TKS
000014      10     loop:   tst    data
000020      11             beq    loop
000022      12             jsr    r0,  print
000026      13             halt
            14
000030      15     data:   .blkw  1
000032      16     pd:     .word  data
```

שורה 8:
האופרנד הראשון מכיל תווית input, לכן נביט על התרשים הראשון. התווית input הינה תווית חיצונית למודול m1, וממוענת בשיטה אבסולוטית (27), לכן יש להוסיף לטבלה את השורה:
input + 02, כאשר 02 היא הכתובת של המילה ביחס לתחילת ה-csect.
האופרנד השני אינו מכיל תווית, לכן נביט על תרשים שני. שיטת המיעון של האופרנד הינה אבסולוטית (37), לכן אין שינוי בטבלה. כלומר, אין צורך בתיקון לאופרנד השני.

שורה 9:
האופרנד הראשון אינו מכיל תווית, והוא ממוען בשיטה אבסולוטית, לכן אין שינוי בטבלה.
האופרנד השני גם אינו מכיל תווית (זהו קבוע) וממוען אבסולוטית, לכן אין שינוי בטבלה.

שורה 10:
האופרנד מכיל תווית data, והיא תווית מקומית הממוענת בשיטה יחסית (67), לכן אין שינוי בטבלה.

שורה 11:
האופרנד מכיל תווית loop, והיא תווית מקומית הממוענת בשיטה יחסית, לכן אין שינוי בטבלה.

שורה 12:
האופרנד מכיל תווית חיצונית print, הממוענת בשיטה יחסית (67), לכן עפ"י תרשים ראשון, יש להוסיף שתי שורות לטבלה: **print + 24** ואת השורה **m1 – 24**.

שורה 13-15:
לא מצריך התלבטות.

שורה 16:
זוהי הנחייה לאסמבלר, לכן נביט על תרשים שלישי. בהנחיה מופיעה התווית data שהיא מקומית, לכן יש להוסיף את השורה **m1 + 32**.

נאסוף את התוצאות לטבלה, ונקבל:

RLD m1		
Symbol	**Flag**	**Location**
input	+	02
print	+	24
m1	-	24
m1	+	32

מודול m2:

```
                    1           .csect  m2
                    2           .entry  input
                    3           .extrn  data
                    4
000000              5           MASK = 177600
000000              6           TKS = 177560
000000              7           TKB = 177562
000000              8
000000              9   input:  mov   @#TKB, r1
000004             10           bic   #MASK, r1
000010             11           mov   r1, @#data
000014             12           inc   @#TKS
000020             13           rti
```

שורה 9:
אופרנד ראשון לא מכיל תווית, וממוען בשיטה אבסולוטית, לכן אין שינוי בטבלה.
שורה 10:
אופרנד ראשון לא מכיל תווית, וממוען בשיטה אבסולוטית, לכן אין שינוי בטבלה.
שורה 11:
אופרנד שני מכיל תווית חיצונית data, וממוען בשיטה אבסולוטית,
לכן יש להוסיף את השורה **data + 12**
שורה 12:
האופרנד לא מכיל תווית, וממוען בשיטה אבסולוטית, לכן אין שינוי בטבלה.
שורה 13:
לא מכילה אופרנדים

נאסוף את התוצאות לטבלה, ונקבל:

RLD m2		
Symbol	**Flag**	**Location**
data	**+**	**12**

מודול m3:

```
                1               .csect  m3
                2               .entry  print
                3               .extrn  pd
                4
000000          5               TPS     =   177564
000000          6               TPB     =   177566
000000          7
000000          8       print:  tstb    @#TPS
000004          9               bpl     print
000006          10              mov     @pd,    @#TPB
000014          11              rts     r0
```

שורה 8:
האופרנד לא מכיל תווית, וממוען בשיטה אבסולוטית, לכן אין שינוי בטבלה.
שורה 9:
האופרנד מכיל תווית מקומית print, הממוענת בשיטה יחסית, לכן אין תיקונים בטבלה.
שורה 10:
האופרנד הראשון מכיל תווית חיצונית pd, וממוען בשיטה יחסית, לכן בכתובת 10 יש לבצע שני תיקונים: **pd + 10** ותיקון **m3 – 10**
האופרנד השני לא מכיל תווית, וממוען בשיטה אבסולוטית, לכן אין שינוי בטבלה.
שורה 11:
האופרנד הוא רגיסטר, אין שינוי בטבלה.

נאסוף את התוצאות לטבלה, ונקבל:

RLD		
Symbol	**Flag**	**Location**
pd	**+**	**10**
m3	**-**	**10**

לפני שנעבור לתהליך הקישור, כדאי יהיה לשמור רישום על אורכו של כל אחד מהמודולים.
האורך הינו כמובן אורך יחסי, והוא פשוט מציין את מספר המלים בזיכרון שצורך כל מודול.
אם אורך של מודול מסוים הינו X, אזי המודול המתאחד אליו יתחיל בכתובת X+2, אם X הינו זוגי, או בכתובת X+3, אם אי זוגי.
נוסיף גם עמודה, המציינת את הכתובת החדשה של תחילת המודולים המתאחדים למודולים אחרים:

טבלת אורכים:

שם המודול	אורך המודול	כתובת תחילת המודול לאחר איחוד
m1	32	00
m2	20	34 (0+32+2)
m3	14	56 (34+20+2)

שלב 3 – קישור

השלבים שיתבצעו פה זהים לחלוטין למתואר בעמ' 10.

בניית מפת טעינה:

בשלב זה נאחד את כל המודולים למודול אחד.
המודול הראשון, m1, יתחיל בכתובת 0. המודול m2, המתאחד אליו, יתחיל בכתובת 34 (עפ"י טבלת האורכים), וכל שורה במודול תוזז במספר הזה.
מודול m3 יתחיל בכתובת 56, וכל שורה בו תוזז במספר זה.
כמובן שאין צורך יותר בסמלים "המיוחדים", לכן אלה יוסרו מהתוכנית:

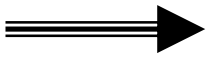
```
000000                          1               TKS  = 177560
000000                          2               TKB  = 177562
000000                          3               TPS  = 177564
000000                          4               TPB  = 177566
000000                          5               MASK = 177600
000000 012737 000000 000060     6       main:   mov  #input, @#60
000006 012737 000101 177560     7               mov  #101,   @#TKS
000014 005767 000010            8       loop:   tst  data
000020 001775                   9               beq  loop
000022 004067 177752            10              jsr  r0, print
000026 000000                   11              halt
000030 000000                   12      data:   .blkw 1
000032 000030                   13      pd:     .word data
000000                          14
000034 013701 177562            15      input:  mov  @#TKB, r1
000040 042701 177600            16              bic  #MASK, r1
000044 010137 000000            17              mov  r1, @#data
000050 005237 177560            18              inc  @#TKS
000054 000002                   19              rti
000056 105737 177564            20      print:  tstb @#TPS
000062 100375                   21              bpl  print
000064 017737 177766 177566     22              mov  @pd, @#TPB
000072 000200                   23              rts  r0
```

איחוד טבלות ESD:

- לאחר שוידאנו כי כל סמל המוגדר ER בטבלות מוגדר גם LD , נסיר את שורות ER מהטבלות.
- כעת נאחד את הטבלות לטבלה אחת, כאשר זוכרים לבצע הזזה של 34 בכתובות היחסיות של טבלת m2, והזזה של 56 בכתובות היחסיות של טבלה m3 (עפ"י טבלת האורכים).

נקבל את טבלת ESD המאוחדת:

Joined ESD		
Symbol	Type	Location
m1	SD	00
data	LD	30
pd	LD	32
~~input~~	~~ER~~	-
~~print~~	~~ER~~	-
m2	SD	~~0~~ 34
input	LD	~~0~~ 34
~~data~~	~~ER~~	-
m3	SD	~~0~~ 56
print	LD	~~0~~ 56
~~pd~~	~~ER~~	-



Joined ESD		
Symbol	Type	Location
m1	SD	00
data	LD	30
pd	LD	32
m2	SD	34
input	LD	34
m3	SD	56
print	LD	56

בשלב זה, יש בידינו את טבלת הסמלים המאוחדת ומיקומם בזיכרון ביחס לתחילת הכתובת במפת הטעינה. כלומר, ביחס לכתובת 0.

<u>איחוד טבלות RLD:</u>

בדיוק באותו אופן, נקבל את טבלת ה- RLD המאוחדת:

Joined RLD		
Symbol	**Flag**	**Location**
input	+	02
print	+	24
m1	-	24
m1	+	32
data	+	46
pd	+	66
m3	-	66

כתובת עם תיקון (+) וגם תיקון (-)

- כעת, עפ"י המתואר בעמ' 10, נאתר את הכתובות שעבורן יש תיקון (+) וגם (-). קל לראות שאלו הן הכתובות 24 ו- 66.
- כעת, עלינו לתקן את השורות האלה במפת הטעינה:

<u>כתובת 24:</u>

```
000022  004067  177752      jsr   r0,  print
```

עלינו להוסיף לכתובת 24 (המכילה כעת את הערך 177752) את הערך print=56 ולהוריד ממנה את הערך m1=0, כאשר ערכי הסמלים נלקחו מתוך טבלת ה- ESD המאוחדת. נבצע את החישוב ונקבל את הערך 000030 (שווה לבדוק).
השורה המתוקנת הינה כעת:

```
000022  004067  000030      jsr   r0,  print
```

<u>כתובת 66:</u>

```
000064  017737  177766  177566      mov @pd,  @#TPB
```

עלינו להוסיף לכתובת 66 (המכילה כעת את הערך 177766) את הערך pd=32 ולהוריד ממנה את הערך m3=56, כאשר ערכי הסמלים נלקחו מתוך טבלת ה- ESD המאוחדת. נבצע את החישוב ונקבל את הערך 177742.
השורה המתוקנת הינה כעת:

```
000064  017737  177742  177566      mov @pd,  @#TPB
```

- לאחר שבוצעו כל התיקונים במפת הטעינה, (הוספת input=34 לכתובת 02, הוספת m1=0 לכתובת 32 והוספת data=30 לכתובת 46) נוריד את השורות "המיוחדות" מהטבלה, ונשמיט ממנה את עמודת שמות הסמלים (השמאלית ביותר). נקבל את טבלת ה- RLD המצומצמת הנקראת – טבלת הטעינה.

טבלת הטעינה:

Flag	Location
+	02
+	32
+	46

ה- load module:

```
000000                                  1               TKS  =  177560
000000                                  2               TKB  =  177562
000000                                  3               TPS  =  177564
000000                                  4               TPB  =  177566
000000                                  5               MASK =  177600
000000  012737  000034  000060          6       main:   mov     #input, @#60
000006  012737  000101  177560          7               mov     #101,   @#TKS
000014  005767  000010                  8       loop:   tst     data
000020  001775                          9               beq     loop
000022  004067  000030                  10              jsr     r0,  print
000026  000000                          11              halt
000030  000000                          12      data:   .blkw   1
000032  000030                          13      pd:     .word   data
000000                                  14
000034  013701  177562                  15      input:  mov     @#TKB,  r1
000040  042701  177600                  16              bic     #MASK,  r1
000044  010137  000030                  17              mov     r1,  @#data
000050  005237  177560                  18              inc     @#TKS
000054  000002                          19              rti
000056  105737  177564                  20      print:  tstb    @#TPS
000062  100375                          21              bpl     print
000064  017737  177742  177566          22              mov     @pd,  @#TPB
000072  000200                          23              rts     r0
```

תם תהליך הקישור. בנקודה זו, מודול הטעינה מושלם לגמרי, עד כדי הזזה בכתובת תחילת הטעינה.

שלב 4 - טעינה

נניח שהתוכנית נטענת לכתובת 1000, אזי כתובות מודול הטעינה יוזזו בערך זה,
ובנוסף, יתווסף ערך זה לכל **הערכים** בכתובות המופיעות בטבלת הטעינה
(כתובות 1002, 1032, 1046)

התוכנית הסופית:

```
000000                                  1               TKS  =  177560
000000                                  2               TKB  =  177562
000000                                  3               TPS  =  177564
000000                                  4               TPB  =  177566
000000                                  5               MASK =  177600
001000  012737  001034  000060          6       main:   mov     #input, @#60
001006  012737  000101  177560          7               mov     #101,   @#TKS
001014  005767  000010                  8       loop:   tst     data
001020  001775                          9               beq     loop
001022  004067  000030                  10              jsr     r0,     print
001026  000000                          11              halt
001030  000000                          12      data:   .blkw   1
001032  001030                          13      pd:     .word   data
000000                                  14
001034  013701  177562                  15      input:  mov     @#TKB,  r1
001040  042701  177600                  16              bic     #MASK,  r1
001044  010137  001030                  17              mov     r1,     @#data
001050  005237  177560                  18              inc     @#TKS
001054  000002                          19              rti
001056  105737  177564                  20      print:  tstb    @#TPS
001062  100375                          21              bpl     print
001064  017737  177742  177566          22              mov     @pd,    @#TPB
001072  000200                          23              rts     r0
```

נספח א':

טבלת המרות (נלקח ממקור אחר)

ערך משלים ל-2 באוקטל	ערך אוקטלי	ערך דצימלי	ערך משלים ל-2 באוקטלי	ערך אוקטלי	ערך דצימלי
177724	**-54**	-44	177777	**-1**	-1
177723	**-55**	-45	177776	**-2**	-2
177722	**-56**	-46	177775	**-3**	-3
177721	**-57**	-47	177774	**-4**	-4
177720	**-60**	-48	177773	**-5**	-5
177717	**-61**	-49	177772	**-6**	-6
177716	**-62**	-50	177771	**-7**	-7
177715	**-63**	-51	177770	**-10**	-8
177714	**-64**	-52	177767	**-11**	-9
177713	**-65**	-53	177766	**-12**	-10
177712	**-66**	-54	177765	**-13**	-11
177711	**-67**	-55	177764	**-14**	-12
177710	**-70**	-56	177763	**-15**	-13
177707	**-71**	-57	177762	**-16**	-14
177706	**-72**	-58	177761	**-17**	-15
177705	**-73**	-59	177760	**-20**	-16
177704	**-74**	-60	177757	**-21**	-17
177703	**-75**	-61	177756	**-22**	-18
177702	**-76**	-62	177755	**-23**	-19
177701	**-77**	-63	177754	**-24**	-20
177700	**-100**	-64	177753	**-25**	-21
177677	**-101**	-65	177752	**-26**	-22
177676	**-102**	-66	177751	**-27**	-23
177675	**-103**	-67	177750	**-30**	-24
177674	**-104**	-68	177747	**-31**	-25
177673	**-105**	-69	177746	**-32**	-26
177672	**-106**	-70	177745	**-33**	-27
177671	**-107**	-71	177744	**-34**	-28
177670	**-110**	-72	177743	**-35**	-29
177667	**-111**	-73	177742	**-36**	-30
177666	**-112**	-74	177741	**-37**	-31
177665	**-113**	-75	177740	**-40**	-32
177664	**-114**	-76	177737	**-41**	-33
177663	**-115**	-77	177736	**-42**	-34
177662	**-116**	-78	177735	**-43**	-35
177661	**-117**	-79	177734	**-44**	-36
177660	**-120**	-80	177733	**-45**	-37
177657	**-121**	-81	177732	**-46**	-38
177656	**-122**	-82	177731	**-47**	-39
177655	**-123**	-83	177730	**-50**	-40
177654	**-124**	-84	177727	**-51**	-41
177653	**-125**	-85	177726	**-52**	-42
177652	**-126**	-86	177725	**-53**	-43

נספח ב':

הערות וחידות

- **פקודות הסתעפות המכילות בתוכן הנחיות לאסמבלר מקודדות באותו אופן.**
דוגמא: נניח שהפקודה br +6. נמצאת בכתובת 1054, אזי הפקודה מעבירה אותנו אל הכתובת 1062 = 6 + 1054. כעת, נחשב את ה-Offset:
2 = 2/(1062 – 1056), לכן ה- Opcode הינו 000402.

- **יהי t הביט הדולק במספר X, אזי הפקודה div #X,r0 שקולה לפקודה t asr r0 פעמים, והפקודה mul #X,r1 שקולה לפקודה t asl r1 פעמים. (עבור t=0 אין הזזה)**
למשל, נניח שברגיסטר r1 נמצא הערך 2, אזי הפקודה mul #40,r1 תניב את התוצאה 100.
הערך הבינארי של 40 הינו 0000000000100000. ביט מספר 6 דולק, לכן עלינו לזוז 5 צעדים ימינה. כלומר, לערך 100=0000000001000000.
<u>הערה:</u> אם יש שני ביטים דולקים, או יותר, רק הביט הראשון (הנמוך ביותר) יוזז על פי חוק זה. שאר הביטים יועברו על פי החוק הבא:
אם ביט u דולק, אזי ביט u+1 ידלוק ביעד.
למשל, עבור mul #45,r1, נקבל את התוצאה 112. הערך הבינארי של 45 הינו 100101.
הביט 0 דולק, לכן אין הזזה במספר 2 = 010. עוד דולקים, הביטים 2 ו-5, לכן נדליק במספר 2 את הביטים 3 ו-6, בהתאמה. נקבל 1001010 = 112 אוקטאלי.

- **כדי לייצג מספר שלילי בן m ביטים באמצעות n ביטים (n>m), עלינו להוסיף משמאל n-m ביטים, כאשר כולם דולקים.**
למשל, אם נרצה לייצג את 10 באמצעות 5 ביטים, נוסיף משמאל 3 ביטים דולקים: 11110

- **נניח כי x-y=z, אזי האופרנד** $x-y(r_n)$ **יחושב כמו** $z(r_n)$.
למשל, נניח כי בכתובת 2002 נמצא הערך 5, וכן $r_0 = 2000$
אזי הפקודה $mov\ \ 4-2(r_0), r1$ תכניס את הערך 5 ל-$r1$.

- **מה עושה הפקודה . br?** גורמת ללולאה אינסופית!
הפקודה הנ"ל שקולה לפקודה br .0+. נניח שהפקודה יושבת בכתובת 2000,
אזי ה-Offset הינו 1- =2/[2002-(2000+0)]. כלומר, זהו מספר המלים המילים שעלינו לקפוץ מערכו של ה-pc, לאחר קריאת הפקודה. כלומר, מהערך 2002. כלומר, עלינו לקפוץ מילה אחת אחורה, אל הכתובת 2000, וחוזר חלילה.

- **למה תגרום הפקודה br .1000+?** לשגיאה על קפיצה רחוקה מדי!
כפי שנאמר (עמ' 3), נוכל לקפוץ לכל היותר 2^7 מילים. נניח כעת שהפקודה נמצאת בכתובת X,
אזי ה- Offset יהיה: 776/2 = 2/[(X+1000)-(X+2)], וכמובן שלא ניתן לייצג את 776 באמצעות 8 ביטים.
השגיאה תתגלה בזמן תרגום, במעבר שני, שהרי שם מתבצע חישוב האופרנדים.

- **למה תגרום הפקודה jmp pc ומתי תתגלה?** לשגיאת "שיטה לא חוקית" ותתגלה בזמן ריצה.
כזכור, הקפיצה נעשית לכתובת האפקטיבית בלבד, ולשיטת מיעון 0 אין כתובת אפקטיבית.

- **למה תגרום הפקודה jmp #147?** ללולאה אינסופית!
נניח שהפקודה נמצאת בכתובת X, אז הקפיצה תהיה לכתובת X+2, עפ"י שיטת מיעון 27.
בכתובת זו נמצא הערך 000147. אם נבחן ערך זה, נגלה שזו בעצם הפקודה jmp –(pc),
שהרי ה-Opcode של jmp הינו 0001.
כלומר, תתבצע קפיצה אל הכתובת האפקטיבית pc-2. כלומר, אל X+2 שוב, וחוזר חלילה.

- **r0 מכיל את הערך 100000. מה יהיה ערכו לאחר הפקודה mul #10,r0?** 177774.
מאחר שהרגיסטר זוגי, תוצאת המכפלה תשב ברגיסטרים r0,r1 כאשר r1 יהווה את המילה הנמוכה.
כפי שהוער למעלה, הפקודה mul יכולה להתבצע באמצעות פקודת asl, לכן הפקודה הנ"ל שקולה לפקודה asl r0-r1 3 פעמים (10_8=1000). המספר 100000 המיוצג ב-16 ביטים הוא המספר 37777700000 המיוצג באמצעות 32 ביט. (1111111111111111 1000000000000000).
כעת נבצע את הפקודה על r0-r1 3 פעמים.
אחרי הזזה ראשונה נקבל: 11111111111111110000000000000000
אחרי הזזה שנייה נקבל: 11111111111111100000000000000000
אחרי הזזה שלישית נקבל: 11111111111111000000000000000000
ניקח את 16 הביטים הראשונים (משמאל, של r0): 1111111111111100
וזהו בדיוק הערך 177774 באוקטלי.

- **למה שקולה הפקודה sob r0 , . ?** לפקודה clr r0.
הפקודה הנ"ל שקולה לפקודה x: sob r0,x, לכן השורה מתבצעת כל עוד r0 שונה מאפס.