כתיבת קוד באסמבלי

1. תרגיל 1:

נתון מספר בכתובת 80H ומספר נוסף בכתובת 84H.

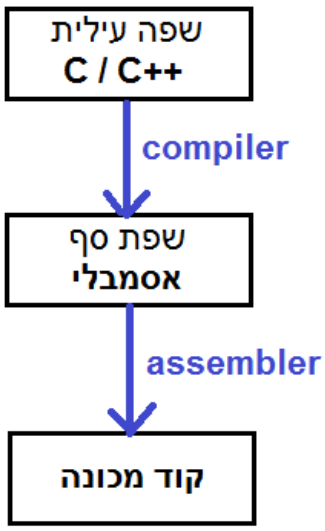
כתוב תוכנית תוך שימוש בפקודות SRC אשר משווה בין שני המספרים ומכניסה את הגדול לכתובת 100H.

פתרון:

כתובת יחסית בזכרון תוכנה	פקודה	הוראה
0	$ld\ r_1, 80H$	$R[r_1] \leftarrow M[80H]$
4	$ld\ r_2, 84H$	$R[r_2] \leftarrow M[84H]$
8	$sub\ r_3, r_2, r_1$	$R[r_3] \leftarrow R[r_2] - R[r_1]$
12	$lar\ r_4, 8$	$R[r_4] \leftarrow PC + 8\ (= 24)$
16	$brpl\ r_4, r_3$	$if\ 0 \leq R[r_3]\ goto\ R[r_4]$
20	$la\ r_2, 0(r_1)$	$R[r_2] \leftarrow R[r_1] + 0$
24	$st\ r_2, 100H$	$M[100H] \leftarrow R[r_2]$
28	$stop$	

- מימוש מנגנון $if - else$ ברמת האסמבלי:

$R[r_3] = R[r_2] - R[r_2];$
$\boldsymbol{if}(R[r_3] \geq 0)\ \ M[100h] = R[r_2];$
$\boldsymbol{else}\ \{$
$\quad R[r_2] = R[r_1];$
$\quad M[100h] = R[r_2];$
$\}$



- פסיאודו קוד שקול בשפה עילית לקוד האסמבלי:

קוד פסיאודו ברמת **הפעולה על משתני התוכנית בלבד** ללא ירידה לרמת ה- CPU (ללא שימוש ברגיסטרים).

$var1 \triangleq M[80h]\ ;$
$var2 \triangleq M[84h]\ ;$
$res \triangleq M[100h];$

$\boldsymbol{if}(var2 - var1 \geq 0)\ \ res = var2;$
$\boldsymbol{else}\ \ \ res = var1;$

2. תרגיל 2:

נתון מספר בזיכרון בכתובת 200. כתוב תכנית בשפת Assembly של מחשב מסוג SRC הבודקת את זוגיות המספר. אם המספר זוגי, התכנית מחלקת אותו ב- 2. אם המספר אי זוגי, התוכנית מחסירה ממנו 1.

יש לשמור את תוצאת התוכנית בכתובת 200.

פתרון

אופן פעולה: נסתכל על ביט ה- LSB של המספר. ניתן לעשות זאת באמצעות "Masking" – סינון סיביות מסוימות ממספר באמצעות פעולת AND, במקרה שלנו, עם 00…01.

לפי תוצאת הפעולה (אם גדולה או שווה ל- 0) נוכל לקבוע אם התוצאה זוגית. אם התוצאה אפס- המספר זוגי, אם אחד- המספר אי זוגי

Address	Assembly	Abstract RTN	
0	$ld\ r_1 200$	$R[r_1] \leftarrow M[200]$	טעינת הערך מהזיכרון
4	$andi\ r_2, r_1, 1$	$R[r_2] \leftarrow r_1 \wedge 1$	Masking
8	$addi\ r_3, r_1, -1$	$R[r_3] \leftarrow R[r_1] - 1$	חיסור 1 מהערך ושמירה
12	$shra\ r_4, r_1, 1$	$R[r_4] \leftarrow R[r_1]/2$	חלוקה ב- 2 (ללא שארית) ע"י הזזה ימינה.
16	$lar\ r_5, 8$	$R[r_5] \leftarrow PC + 8\ (= 28)$	שמירת ערך להסתעפות
20	$brzr\ r_5, r_2$	$(R[r_2] = 0) \rightarrow (PC \leftarrow R[r_5])$	אם המספר זוגי – הסתעף לסוף התוכנית (שמור חלוקה)
24	$la\ r_4, 0(r_3)$	$R[r_4] \leftarrow R[r_3]$	אם לא הייתה הסתעפות – שמור את התוצאה פחות 1
28	$st\ r_4, 200$	$M[200] \leftarrow R[r_4]$	שמירה בזיכרון
32	$stop$		עצירת התוכנית!

- מימוש מנגנון $if - else$ ברמת האסמבלי:

```
if(R[r1] and 0x01 == 0) M[200] = R[r4];
else {
        R[r4] = R[r3];
        M[200] = R[r4];
    }
```

- פסיאודו קוד שקול בשפה עילית לקוד האסמבלי:

קוד פסיאודו ברמת **הפעולה על משתני התוכנית בלבד** ללא ירידה לרמת ה- CPU (ללא שימוש ברגיסטרים).

```
var1 ≜ M[200] ;

if(var1 and 0x01 == 0) var1 = var1/2;
else  var1 - -;
```

3. תרגיל 3:

נתונים מספרים חיוביים x_1, x_2 בכתובות 200, 204 בהתאמה.

כתוב תוכנית שמחשבת את החלוקה של x_1 ב- x_2 ומכניסה את המנה לכתובת 208 ואת השארית לכתובת 212.

פתרון

נבצע חלוקה באמצעות חיסור:

נחסר את המספר x_2 מ- x_1 עד שנגיע למספר שלילי, ונספור כמה פעמים חיסרנו.

כמות הפעמים (פחות אחת) היא תוצאת החלוקה.

השארית תחושב ע"י הוספת המספר x_2 למספר השלילי שיתקבל אחרי החיסור.

Address	Assembly	Abstract RTN	
0	$ld\ r_1, 200$	$R[r_1] \leftarrow M[200]$	קריאת x1 מהזיכרון
4	$ld\ r_2, 204$	$R[r_2] \leftarrow M[204]$	קריאת x2 מהזיכרון
8	$la\ r_3, 0$	$R[r_3] \leftarrow 0$	שמור 0 .
12	$lar\ r_4, 0$	$R[r_4] \leftarrow PC\ (= 16)$	שמור נקודה להסתעפות
16	$addi\ r_3, r_3, 1$	$R[r_3] \leftarrow R[r_3] + 1$	ספור חיסור
20	$sub\ r_1, r_1, r_2$	$R[r_1] \leftarrow R[r_1] - R[r_2]$	חסר x2 מ- x1
24	$brpl\ r_4, r_1$	$(R[r_1] \geq 0) \rightarrow (PC \leftarrow R[r_4])$	אם המספר לא שלילי, הסתעף (חזור על חיסור)
28	$add\ r_1, r_1, r_2$	$R[r_1] \leftarrow R[r_1] + R[r_2]$	תיקון שארית
32	$addi\ r_3, r_3, -1$	$R[r_3] \leftarrow R[r_3] - 1$	תיקון מונה
36	$st\ r_1, 212$	$M[212] \leftarrow R[r_1]$	שמור שארית בזיכרון
40	$st\ r_3, 208$	$M[208] \leftarrow R[r_3]$	שמור מונה בזיכרון
44	$stop$		

- מימוש מנגנון $do - while$ ברמת האסמבלי:

$$\boxed{\begin{array}{l} \boldsymbol{do}\{ \\ \quad R[r_3] + +; \\ \quad R[r_1] = R[r_1] - R[r_2]; \\ \}\boldsymbol{while}(R[r_1] \geq 0); \end{array}}$$

- פסיאודו קוד שקול בשפה עילית לקוד האסמבלי:
קוד פסיאודו ברמת **הפעולה על משתני התוכנית בלבד**
ללא ירידה לרמת ה- CPU (ללא שימוש ברגיסטרים).

$$
\begin{aligned}
& x_1 \triangleq M[200]\ , x_2 \triangleq M[204]\ ; \\
& Quotient \triangleq M[208] = 0\ , residue \triangleq M[212] = 0; \\
& \\
& \boldsymbol{do}\{ \\
& \qquad Quotient++; \\
& \qquad x_1 = x_1 - x_2; \\
& \}\boldsymbol{while}(x_1 \geq 0); \\
& Quotient--; \\
& residue = x_1 + x_2;
\end{aligned}
$$

4. תרגיל 4:

נתונים 2 מערכים (מערך מקור ומערך מטרה). מערך המקור מתחיל בכתובת 0 עד הכתובת 399.
מערך המטרה מתחיל בכתובת 400 עד הכתובת 799.

כתוב תוכנית המבצעת בדיקת זוגיות (Parity check) על מערך המקור ושומרת את התוצאה במערך המטרה: אם מספר האחדות הוא זוגי – התוצאה היא 0, אם מספר האחדות אי זוגי – התוצאה היא 1.

פתרון

בדיקת הזוגיות מתבצעת על כל תא בזיכרון, כאשר 0 מסמל מספר זוגי של אחדות בסיביות, ו- 1- (המחושב בשיטת המשלים ל-2, ע"י NOT 0) מסמל מספר אי זוגי של אחדות.

בפועל, אנו צריכים לעבור על כל הסיביות בכל תא בזיכרון.

- שימו לב שכל תא בזיכרון הוא 4 בתים!

במקום למנות את כמות האחדות, נעבוד עם רגיסטר שיתהפך בכל פעם שנתקל ב- 1.
היפוך זה יעשה באמצעות פקודות NOT.

1. עלינו לעבור על כל תא בזיכרון, עד שהגענו לכתובת ה- 400H (לולאה ראשית)
2. עלינו לעבור על כל הסיביות (לולאה משנית)

ביצוע בדיקת הסיביות תתבצע על סיבית ה- LSB באמצעות Masking, והזזה ימינה בכל שלב בלולאה המשנית.

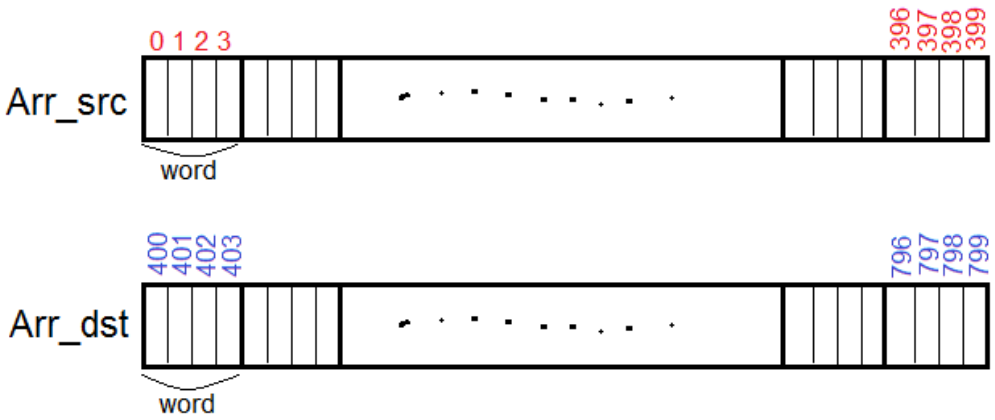
נקצה את הרגיסטרים הבאים:

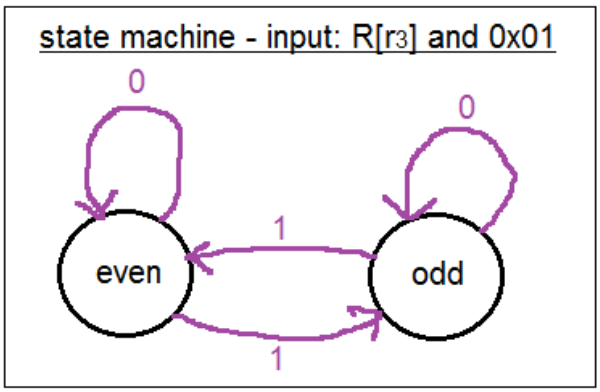
רגיסטר	מטרה
r_1	כתובת התא הבא במערך שיש לבדוק.
r_3	קריאה מהזיכרון.
r_5	משמש להזזות של המספר הנקרא מהזיכרון.
r_6	כתובת להסתעפות אם אין צורך בשינוי תוצאה.
r_7	תוצאת הבדיקה בכל שלב.
r_8	כתובת להסתעפות לקריאת סיביות.
r_9	תנאי להסתעפות – סוף מערך.
r_{10}	כתובת להסתעפות – איבר הבא במערך.

Address	Assembly	
0	$la\ r_1, 0$	הכנס כתובת 0 למצביע על המערך.
READ: 4	$ld\ r_3, 0(r_1)$	קרא מהזיכרון.
8	$la\ r_7, 0$	אפס את תוצאת הבדיקה.
MASK: 12	$andi\ r_5, r_3, 1$	בדד את סיבית ה LSB
16	$lar\ r_6, SHIFT$	
20	$brzr\ r_6, r_5$	אם ה- LSB היא 0 קפוץ שורה (ל- SHIFT).
24	$not\ r_7, r_7$	אם לא קפצת, אז הפוך את תוצאת הבדיקה.
SHIFT: 28	$shr\ r_3, r_3, 1$	הזז את הערך הנבדק ימינה ב-1. הערה: את ה- MSB מחליפה הספרה 0.
32	$lar\ r_8, MASK$	
36	$brnz\ r_8, r_3$	אם המספר הוא לא 0, קפוץ ל- MASK.
40	$st\ r_7, 400(r_1)$	שמור את התוצאה במקום הנוכחי + 400 (מערך מטרה).
44	$addi\ r_1, r_1, 4$	עבור למקום הבא במערך.
48	$addi\ r_9, r_1, -400$	חסר 400 מהמקום הבא.
52	$lar\ r_{10} READ$	
56	$brnz\ r_{10}, r_9$	אם לא עברת את המקום ה- 400, חזור ל- READ.
60	$stop$	

שימו לב: התוכנית לא יעילה. לדוגמה, נוכל להזיז את ההשמות של המצביעים מחוץ ללולאות, וכך לחסוך זמן ריצה של התוכנית. נראה זאת בדוגמה הבאה.

כל איבר בכל אחד מהמערכים הוא בגודל מילה (32 bit = 4 Byte) .





- מימוש מנגנון לולאה מקוננת $for\{do-while\{\ldots\}\}$ ברמת האסמבלי:

```
for(R[r1] = 0 ; R[r1] − 400 ≠ 0 ; R[r1] = R[r1] + 4){
        R[r3] = M[R[r1]];
        R[r7] = 0;
        do{
                if(R[r3] and 0x01 == 0) R[r3] = shr1−pos( R[r3]);
                else {
                        R[r7] = not (R[r7]);
                        R[r3] = shr1−pos( R[r3]);
                }
        }while(R[r3] ≠ 0);
        M[R[r1] + 400] = R[r7];
}
```

- פסיאודו קוד שקול בשפה עילית לקוד האסמבלי:
קוד פסיאודו ברמת **הפעולה על משתני התוכנית בלבד**
ללא ירידה לרמת ה- CPU (ללא שימוש ברגיסטרים).

```
Arr_src[100] ≜ M[0] − M[396] ;
Arr_dst[100] ≜ M[400] − M[796] ;
state;

for(i = 0 ; i < 100 ; i++){
        state = 0;
        do{
                if(Arr_src[i] and 0x01 == 0) Arr_src[i] = shr_{1−pos}( Arr_src[i]);
                else {
                        state = not (state);
                        Arr_src[i] = shr_{1−pos}( Arr_src[i]);
                    }
        }while(Arr_src[i] ≠ 0);
        Arr_src[i + 100] = state;
}
```

5. תרגיל 5:

נתון מערך מקור בו אגורים 1024 ערכים שלמים וחיוביים בני 32 סיביות כל אחד, המאוחסנים בזיכרון החל מהכתובת 1000. בנה מערך מטרה בו יהיו אגורים 1024 ערכים בני 32 סיביות כל אחד המאוחסנים בזיכרון החל מהמקום 10,000. תוכן התאים מוגדר באופן הבא:

$$M[4i + 10{,}000] = \lfloor M[4i + 1000]2^{-i \bmod 32} \rfloor, \qquad 0 \le i \le 1023$$

כאשר $\lfloor \quad \rfloor$ הינו הערך השלם התחתון של התוצאה.

פתרון

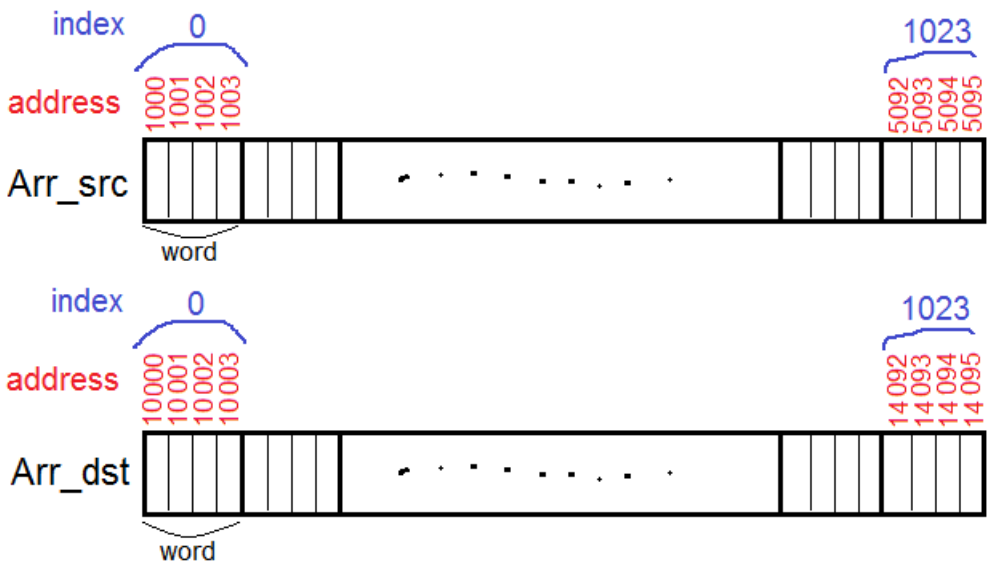
בבסיס בינארי, חילוק בחזקות של 2 משמעו הזזת הנקודה שמאלה. במקרה שלנו, $2^5 = 32$ ולכן עלינו לבודד את 5 סיביות ה- LSB ואלו הן השארית חלוקה ב- 2.

לכן, עלינו לחשב לפי הנוסחה הבאה:

1. מצא את הערך i.
2. שמור שארית חלוקה ב- 32 (5 סיביות LSB) ב- r_{31}.
3. שלוף ערך השמור במערך מקור ושמור ב- r_1.
4. קח את הערך השלם של חלוקת r_1 ב $2^{r_{31}}$ – ע"י הזזה ימינה r_3 פעמים.
5. שמור את התוצאה במערך המטרה.

- כדי לבודד את 5 סיביות ה- LSB עלינו לעשות Masking עם המספר 31.
- נעבור על המערך מהסוף להתחלה, כדי שנשתמש ישירות ב- $brpl$ על הרגיסטר של הלולאה. באופן כזה נחסוך חישוב.

Address	Assembly	Abstract RTN	
0	$la\ r_{30}, 4092$	$R[r_{30}] \leftarrow 4092$	מצביע לסוף מערך מקור.
4	$lar\ r_{29}, READ$	$R[r_{29}] \leftarrow PC + 0(= 8)$	מצביע ל- READ
READ : 8	$ld\ r_1, 1000(r_{30})$	$R[r_1] \leftarrow M[R[r_{30}] + 1000]$	קריאה מהזיכרון.
12	$shr\ r_{31}, r_{30}, 2$	$R[r_{31}] \leftarrow R[r_{30}]/\ 4$	חלוקה ב- 4 (i).
16	$andi\ r_{31}, r_{31}, 31$	$R[r_{31}] \leftarrow R[r_{31}] mod\ 32$	5 סיביות LSB.
20	$shr\ r_1, r_1, r_{31}$	$R[r_1] \leftarrow R[r_1]/R[r_{31}]$	החישוב הנדרש.
24	$st\ r_1, 10000(r_{30})$	$M[R[r_{30}] + 10000] \leftarrow R[r_1]$	שמירה בזיכרון.
28	$addi\ r_{30}, r_{30}, -4$	$R[r_{30}] \leftarrow R[r_{30}] - 4$	חזור איבר במערך מקור.
32	$bprl\ r_{29}, r_{30}$	$(R[r_{30}] \geq 0) \rightarrow (PC \leftarrow\ R[r_{29}])$	אם לא עברת את ההתחלה, חזור ל- READ
36	$stop$		



- מימוש מנגנון לולאת $for\{\}$ ברמת האסמבלי:

$$
\begin{aligned}
&\boldsymbol{for}(R[r_{30}] = 4092\ ;\ R[r_{30}] \geq 0\ ;\ R[r_{30}] = R[r_{30}] - 4)\{\\
&\qquad R[r_1] = M[R[r_{30}] + 1{,}000];\\
&\qquad R[r_{31}] = R[r_{30}]/4;\\
&\qquad R[r_{31}] = R[r_{31}]\ \boldsymbol{and}\ 31;\\
&\qquad R[r_1] = \boldsymbol{shr}_{R[r_{31}]-pos}(\ R[r_1]);\\
&\qquad M[R[r_{30}] + 10{,}000] = R[r_1];\\
&\}
\end{aligned}
$$

- פסיאודו קוד שקול בשפה עילית לקוד האסמבלי:
קוד פסיאודו ברמת **הפעולה על משתני התוכנית בלבד**
ללא ירידה לרמת ה- CPU (ללא שימוש ברגיסטרים).

$$
\begin{array}{l}
Arr\_src[1024] \triangleq M[1{,}000] - M[5{,}092]\ ; \\
Arr\_dst[1024] \triangleq M[10{,}000] - M[14{,}092]\ ; \\
count; \\
\boldsymbol{for}(i = 0\ ;\ i < 1024\ ;\ i++)\{ \\
\qquad count = i\ \ \boldsymbol{and}\ \ 0x01F; \\
\qquad Arr\_dst[i] = \boldsymbol{shr}_{\boldsymbol{count-pos}}(\ Arr\_src[i]); \\
\}
\end{array}
$$