ארגון המחשב ומערכות הפעלה

אביב תשפ"ד

תרגול 6 – שפת מכונה Assembly

תזכורת

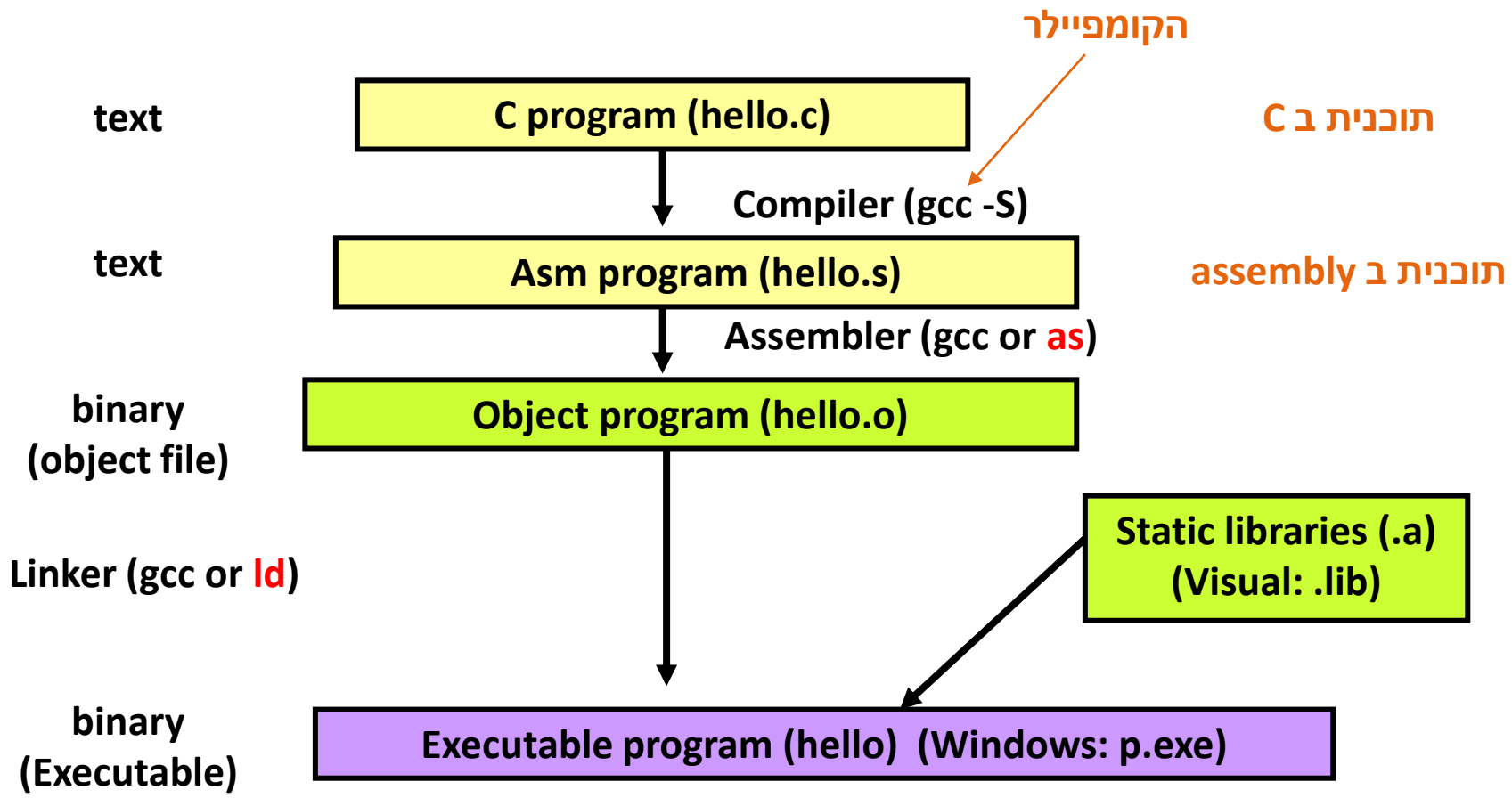
הפונקציה שהמתכנת כותב בשפת c עוברת כמה שלבים עד שהמחשב מסוגל לקרוא את הקלט בצורה שהוא יבין אילו פעולות יש לבצע.

בהתחלה הפכנו את התכנית לקובץ מקור, והמטרה שלנו היא להפוך את התכנית לקובץ של ביטים (0-ים ו-1-ים)

אנחנו ניקח את קובץ המקור ונתרגם אותו לאסמבלי

בעזרת האסמבלי נוכל לתרגם את התכנית לקובץ בינארי ← שהמחשב מבין

המרה מתכנית ב-c לתכנית ברת הרצה



* מקור חיצוני שאינו הקומפיילר שלנו

תכונות האסמבלי

- שפת האסמבלי הנפוצה כיום היא x86-64 (מאפשרת מהירות גבוהה יותר מ-IA32)
- יש פקודות פשוטות, שיש להן תרגום לביטים שהמעבד (CPU) מבין
- אין דבר כזה מערך, וקטור (משתנים מורכבים) ← רק משתנים פשוטים בגדלים קטנים:
  - שלמים בגדלים של 1,2,4 **בתים**
  - floating point בגדלים 4,8
  - פעולות אריתמטיות על *רגיסטרים או ערך בזיכרון
  - פעולות שמירה והעברה לזיכרון
  - פעולות control- ...if, while

* רגיסטר: תא אחסון נתונים, בצורת אוסף סיביות, המשמש כאופרנד (משתנה) ישיר לפעולות המעבד.

תכונות האסמבלי

כל הפקודות באסמבלי נראות כך

$$instruction\ operand1, operand2\ \ldots\ \ldots$$

לכל פקודה יש 0 או יותר אופרנדים (משתנים)

הפקודה אומרת מה עושים

האופרנד אומר עם מה עושים את הפעולה- יכול להיות רגיסטר, ערך, כתובת

מבחינת דרך כתיבה- בקורס נכתוב בצורת AT&T

$$instruction\ source,destination$$

שיטות מיעון באסמבלי (addressing)

Type	Form	Operand value	Name	Example
constant	$a	a	Immediate	$4, $0x4
Register	R[1-8]	R[n]	Register	R1
Memory	c	M[c]	Absolute	4
Memory	(R)	M[R[n]]	Indirect	(R1)
Memory	c(R)	M[c+R[n]]	Base + displacement	4(R1)
Memory	(R1,R2)	M[R[n1]+R[n2]]	Index	(R1,R2)
Memory	c(R1,R2)	M[c+R[n1]+R[n2]]	Index	4(R1,R2)
Memory	(R1,R2,s)	M[R[n1]+sR[n2]]	Scaled index	(R1,R2,2)
Memory	c(R1,R2,s)	M[c+R[n1]+sR[n2]]	Scaled index	4(R1,R2,2)

האופרנדים יכולים להציג

1. קבועים
2. ערכים של רגיסטרים
3. ערכים מהזכרון

המספרים ב-c,s תמיד יהיו 1,2,4 או 8

שיטות מיעון באסמבלי (addressing)

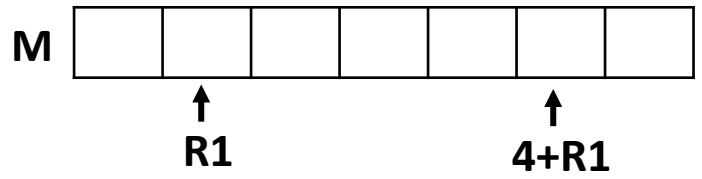
Type	Form	Operand value	Name	Example
constant	$a	a	Immediate	$4, $0x4
Register	R[1-8]	R[n]	Register	R1
Memory	c	M[c]	Absolute	4
Memory	(R)	M[R[n]]	Indirect	(R1)
Memory	c(R)	M[c+R[n]]	Base + displacement	4(R1)
Memory	(R1,R2)	M[R[n1]+R[n2]]	Index	(R1,R2)
Memory	c(R1,R2)	M[c+R[n1]+R[n2]]	Index	4(R1,R2)
Memory	(R1,R2,s)	M[R[n1]+sR[n2]]	Scaled index	(R1,R2,2)
Memory	c(R1,R2,s)	M[c+R[n1]+sR[n2]]	Scaled index	4(R1,R2,2)

הערך של המשתנה

התחלה של אזור בזיכרון

כתובת בזיכרון

הערך שהרגיסטר מצביע עליו



תרגיל 1

זיכרון

כתובת	ערך
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x14

מעבד

רגיסטר	ערך
R1	0x100
R2	0x1
R3	0x3

אופרנד	סוג	ערך
R1	רגיסטר	0x100
0x104		
$0x108		
(R1)		
4(R1)		
9(R1,R3)		
(R1,R3,4)		

תרגיל 1 - פתרון

זיכרון

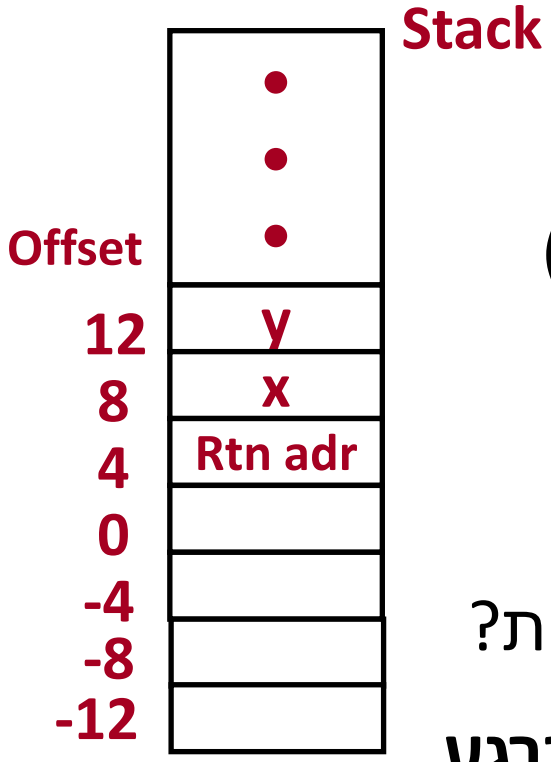
כתובת	ערך
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x14

מעבד

רגיסטר	ערך
R1	0x100
R2	0x1
R3	0x3

אופרנד	סוג	ערך
R1	רגיסטר	0x100
0x104	זיכרון	0xAB
$0x108	קבוע	0x108
(R1)	זיכרון כתובת: 0x100	0xFF
4(R1)	זיכרון כתובת: 0x104	0xAB
9(R1,R3)	זיכרון כתובת: 0x10C	0x14
(R1,R3,4)	זיכרון כתובת: 0x10C	0x14

המחסנית Stack



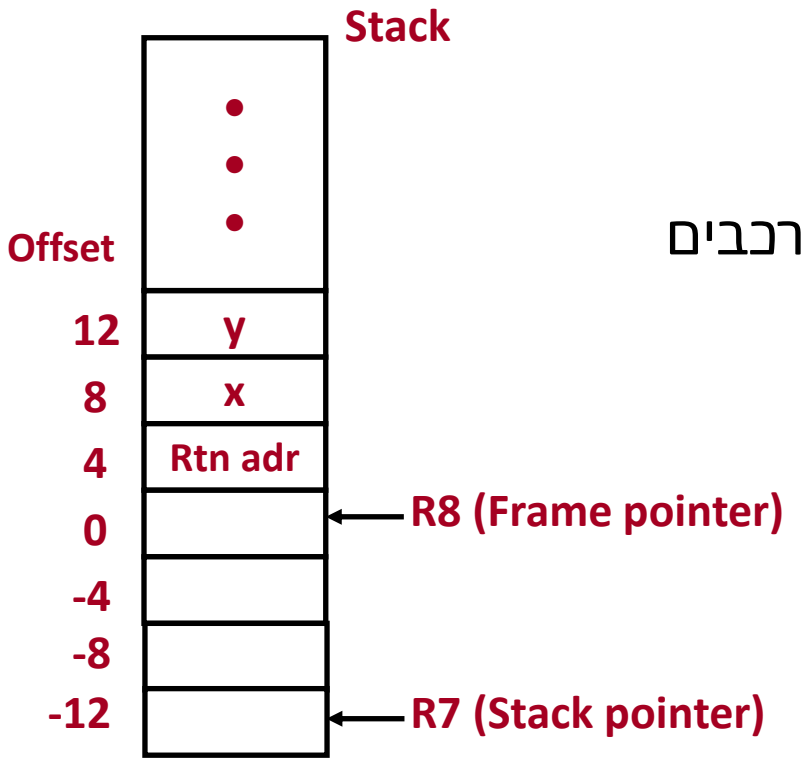
- המחסנית מהווה חלק חשוב בזיכרון (חלק מה-RAM)
- מנהלת את אופן הקריאות לפונקציות
- בעת הפעלת פונקציה- המחסנית משנה את גודלה (מעין אקורדיון)
- לכל פונקציה יש מסגרת משלה (frame)- טווח כתובות משלה
- בשקוראים לפונקציה חדשה- מוקצה עבורה frame משלה

מה הקשר בין מספר הפריימים במחסנית למספר הפונקציות בתוכנית?

<u>תשובה</u>: מספר הפריימים במחסנית = מספר הפונקציות **שפועלות כרגע**.

מה יש בכל מסגרת?

- הכתובת של תחילת המסגרת נשמרת תמיד ברגיסטר R8 ← **frame pointer**
- המקום האחרון שבו כתבנו למחסנית תמיד נשמר ברגיסטר R7 ← **stack pointer**
- הערך שמוחזר מהפונקציה נשמר תמיד ברגיסטר R1



בנוסף לכל אלה, המסגרת מכילה גם:

- משתנים מקומיים: משתנים שאין רגיסטר פנוי עבורם / משתנים מורכבים
- מידע אדמיניסטרטיבי: פרמטרים שהפונקציה מקבלת ועוד
- כתובת לחזרה **Rtn adr**: כתובת בזיכרון הראשי של הפקודה הבאה
- פקודות תחילת / סוף פונקציה
- פקודות move

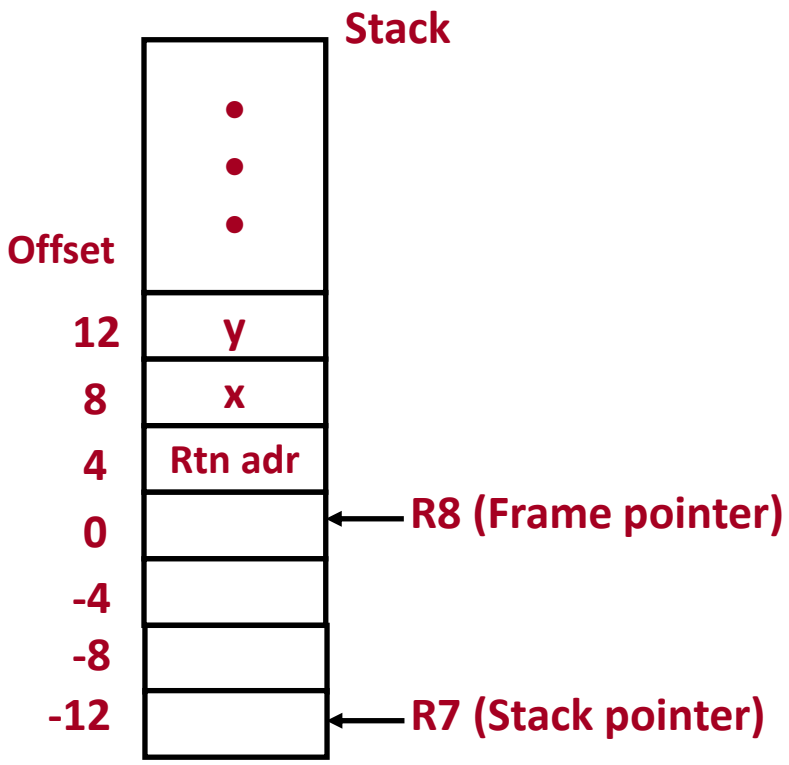
מה יש בכל מסגרת?

מדוע צריך את ה Frame pointer?

<u>תשובה</u>: כי ממנו והלאה נמצאים הפרמטרים.
לא נוכל להפעיל פונקציה אם לא נדע באיזה מקום בזיכרון היא מאוחסנת.

האם כל הפונקציות בהכרח נשמרות ב stack?

<u>תשובה</u>: לפעמים יש פונקציות שמתבצעות במהירות ואין מה לפתוח עבורן frame. לכן אפשר להמליץ לקומפיילר לעשות inlining (לא לפתוח frame).



תרגיל 2

נתונה הפונקציה F והפונקציות הנקראות על ידה g ו-h.

הציגו את מצב המחסנית לאורך כל זמן ריצת הפונקציה F.

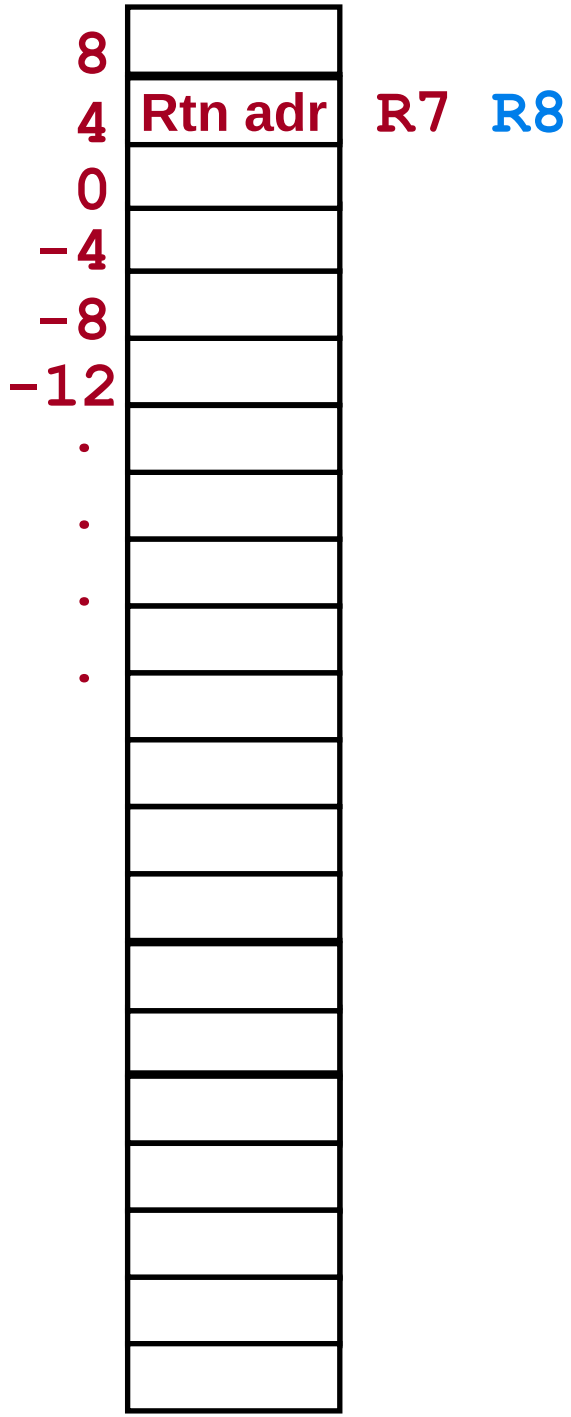
הניחו כי כל המשתנים המקומיים מאוחסנים במחסנית.

```
int F()
{
   int x = 1 , y = 1 ;
   x = g() + x ;
   y = h() + y ;
   return x + y ;
}
```

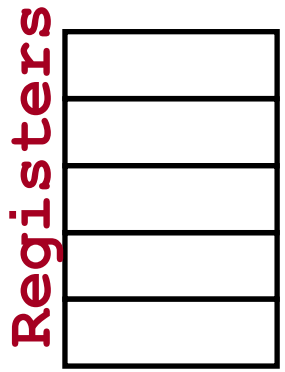
```
int g()
{
   int w = 3 ;
   return w ;
}
```

```
int h()
{
   int z = 5 ;
   return z ;
}
```

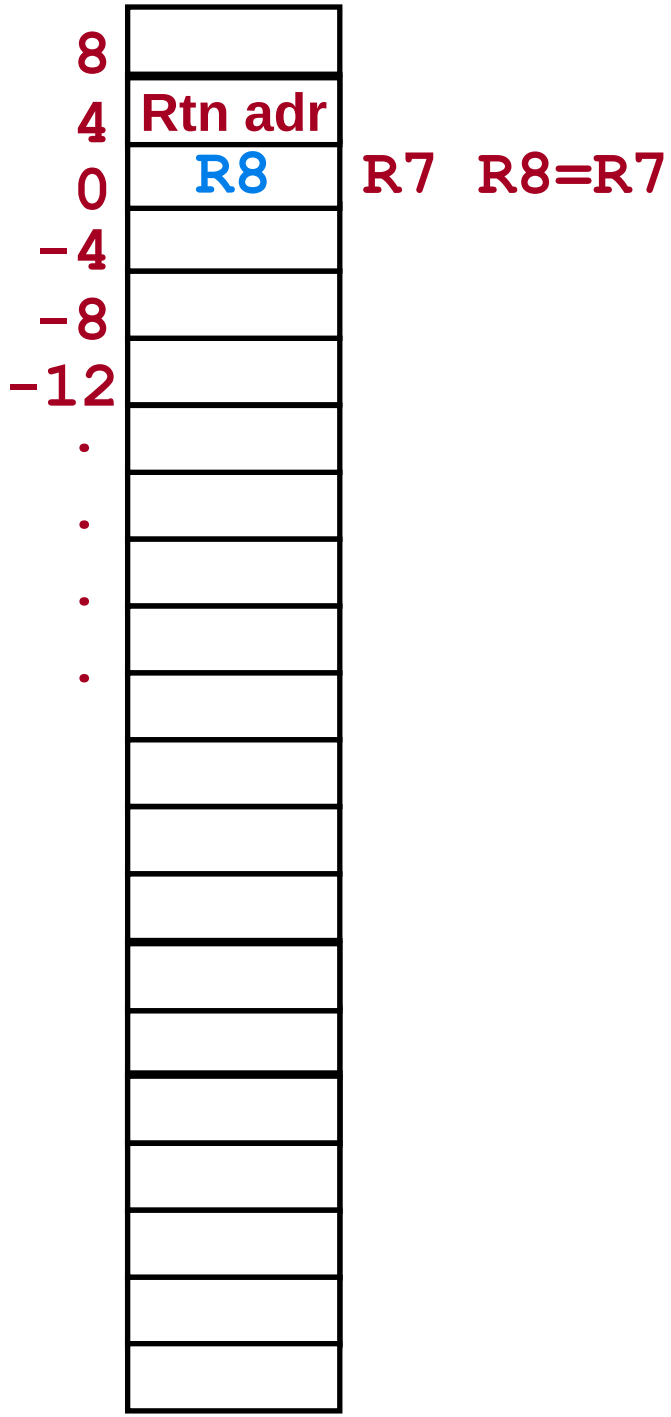
תרגיל 2 - פתרון




```
int F( ) {
```

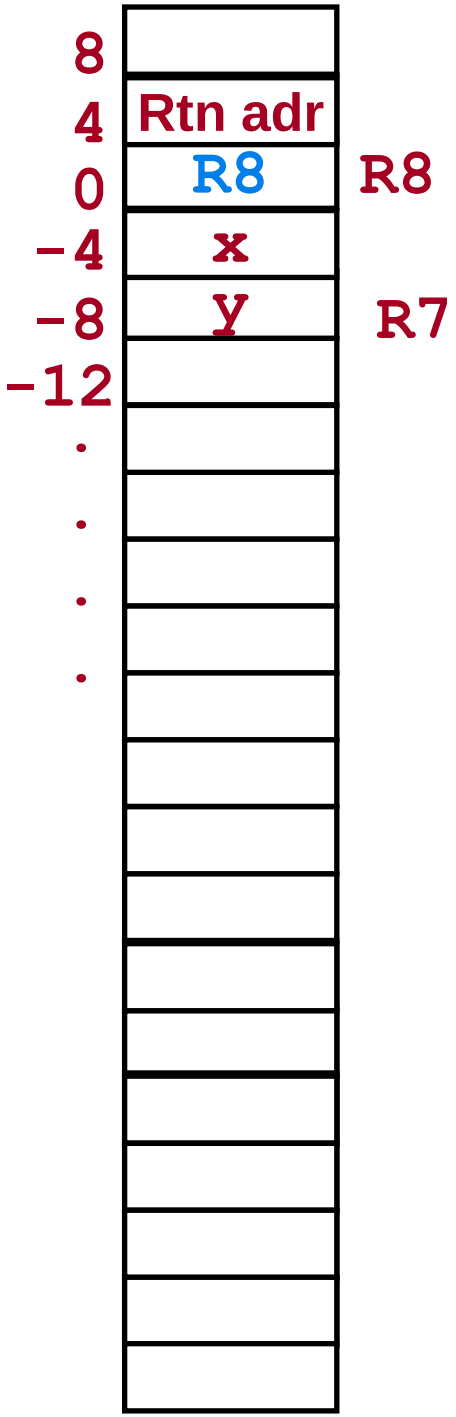


תרגיל 2 - פתרון




```
int F( ) {
```

תרגיל 2 - פתרון

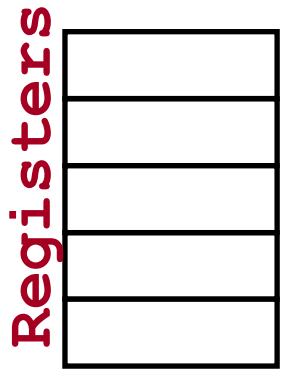



```
int F( ) {
        int x=1, y=1;
```

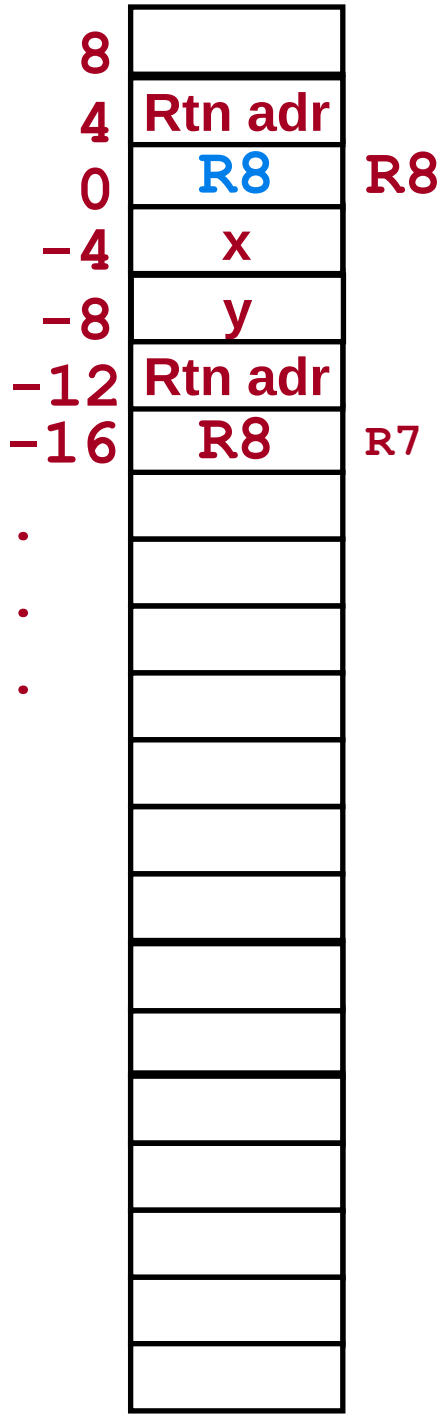
אם היה רשום:

```
int x;
int y;
```

זה לא היה נכנס למחסנית!
רק בעת השמת ערך
המשתנה נכנס.



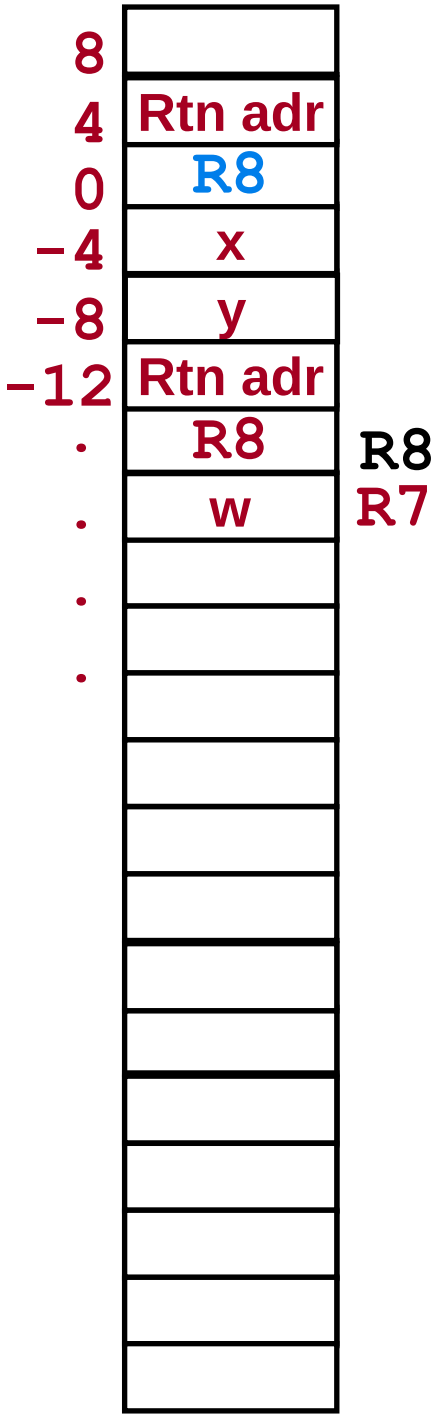
תרגיל 2 - פתרון




```
int g( ) {
```

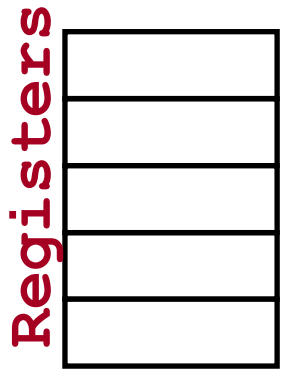
```
int F( ) {
        int x=1, y=1;
        x=g( )+x;
```

תרגיל 2 - פתרון

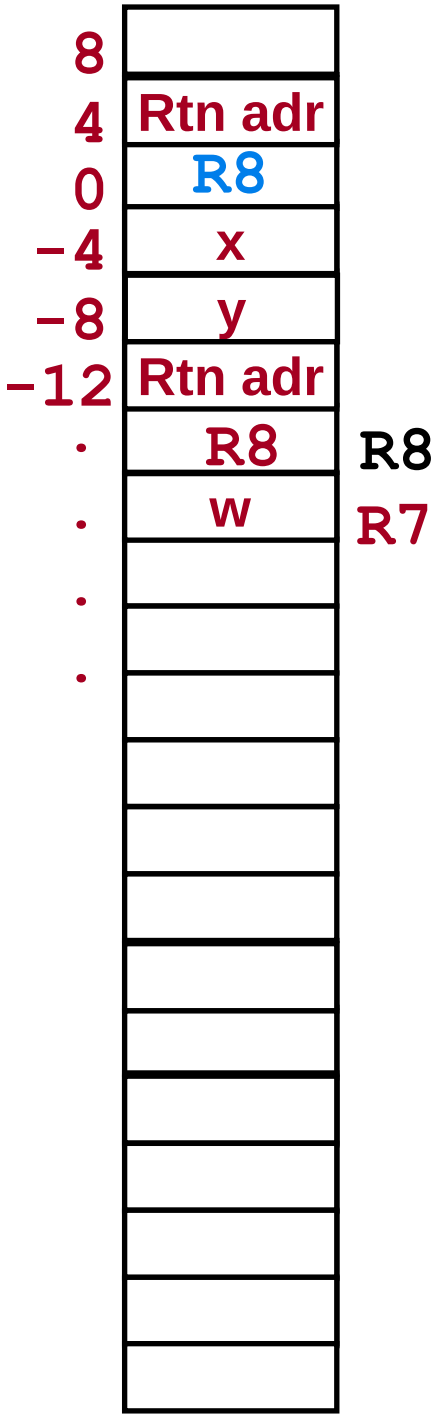



```
int g( ) {
        int w=3;
}
```

```
int F( ) {
        int x=1, y=1;
        x=g( )+x;
}
```



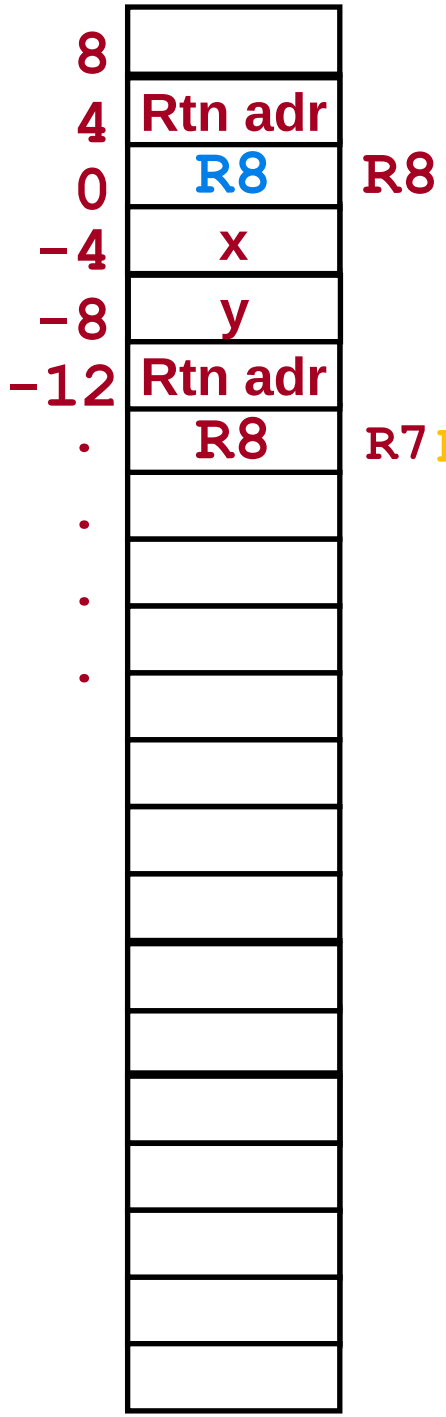
תרגיל 2 - פתרון




```
int g( ) {
        int w=3;
        return w;
}
```

```
int F( ) {
        int x=1, y=1;
        x=g( )+x;
```

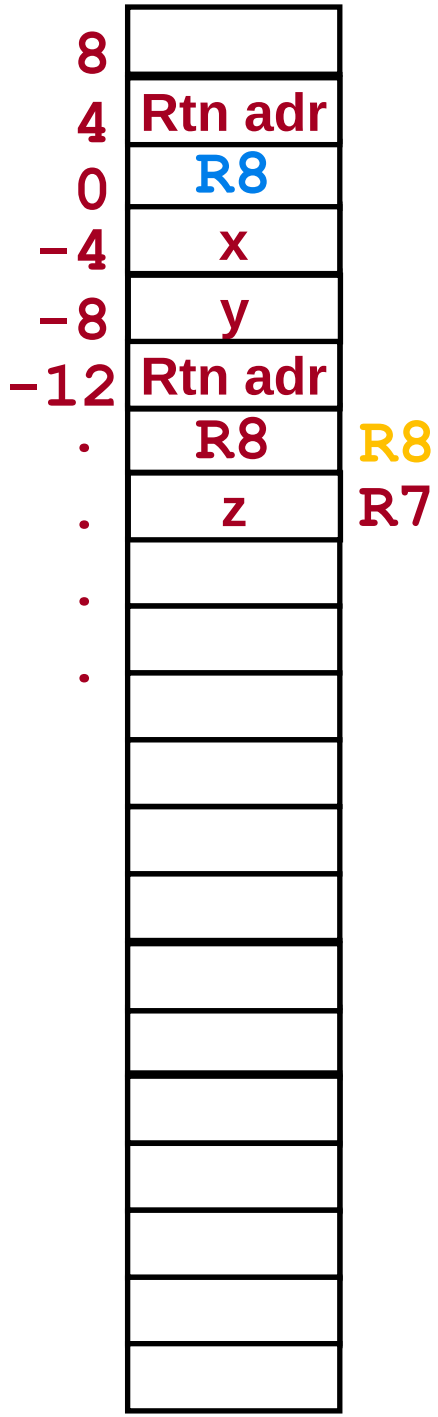
תרגיל 2 - פתרון




```
int h( ) {
}
```

```
int F( ) {
        int x=1, y=1;
        x=g( )+x;
        y=h()+y;
}
```

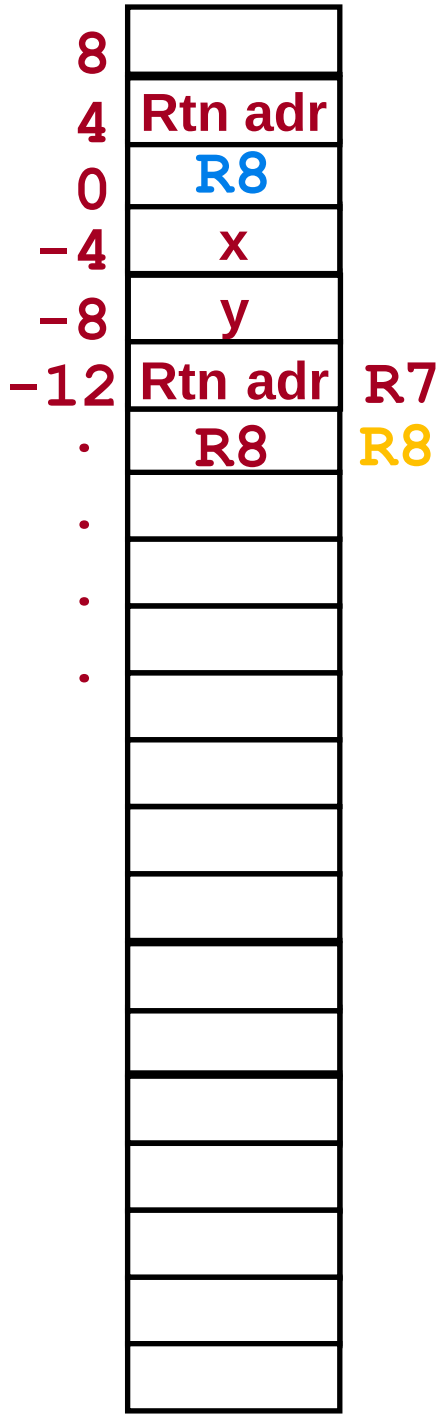
תרגיל 2 - פתרון




```
int h( ) {
        int z=5;
```

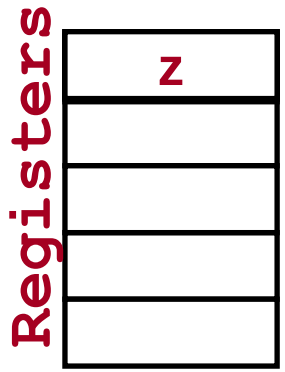
```
int F( ) {
        int x=1, y=1;
        x=g( )+x;
        y=h( )+y;
```

תרגיל 2 - פתרון

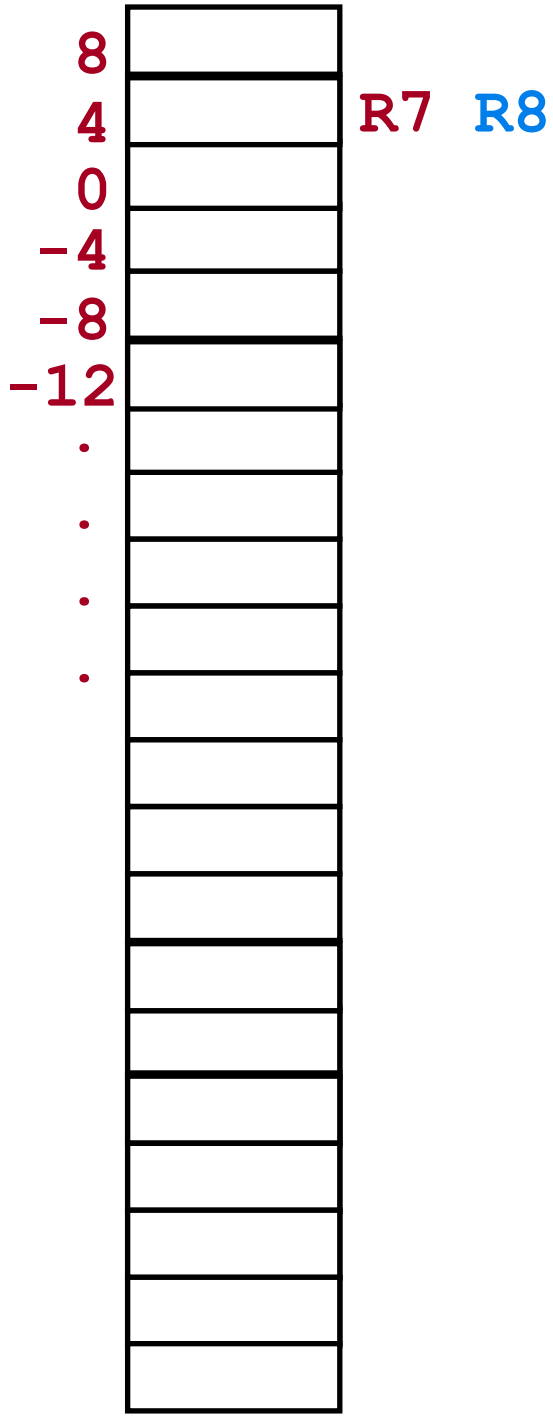



```
int h( ) {
        int z=5;
        return z;
}
```

```
int F( ) {
        int x=1, y=1;
        x=g( )+x;
        y=h( )+y;
        return x+y; }
```



תרגיל 2 - פתרון




```
int F( ) {
        int x=1, y=1;
        x=g( )+x;
        y=h( )+y;
        return x+y; }
```