ארגון המחשב ומערכות הפעלה

אביב תשפ"ד

תרגול 7 – המשך Assembly

בתרגול הקודם

- התחלנו להבין קצת מה זה אסמבלי
- אמרנו שמהאסמבלי נוכל להפוך את התכנית שלנו ליותר קומפקטית, ולאחר מכן נוכל להפוך אותו לקובץ בינארי שהמחשב מבין.

פעולות באסמבלי

- למדנו בהרצאה על פעולות אריתמטיות שאפשר לעשות באסמבלי

  למדנו על פעולות עם 2 אופרנדים

  למדנו על פעולות עם אופרנד אחד

שימו לב! בספר הפעולות מיוצגות באופן האמיתי!

פה ובהרצאה נציג באופן יותר קריא ופשוט

פעולות עם 2 אופרנדים

`add`	*Src,Dest*	*Dest* = *Dest* + *Src*	
`subtract`	*Src,Dest*	*Dest* = *Dest* - *Src*	
`multiply`	*Src,Dest*	*Dest* = *Dest* * *Src*	
`shiftal`	*Src,Dest*	*Dest* = *Dest* << *Src*	Also called `shll`
`shiftar`	*Src,Dest*	*Dest* = *Dest* >> *Src*	Arithmetic
`shiftr`	*Src,Dest*	*Dest* = *Dest* >> *Src*	Logical
`xor`	*Src,Dest*	*Dest* = *Dest* ^ *Src*	
`and`	*Src,Dest*	*Dest* = *Dest* & *Src*	
`or`	*Src,Dest*	*Dest* = *Dest* | *Src*	

פעולות עם אופרנד אחד

1. inc counter → counter++
2. dec counter → counter--

פקודת move

למדנו גם על פקודות move - אלו הן פקודות שמזיזות מידע

move source, destination

אופרנדים שונים שעובדים עם move:

1. קבועים: מתחילים ב-$
2. רגיסטרים: R1-R8 (שחלקם עם תפקיד ייעודי)
3. מיקום בזיכרון: סוגריים עגולים

שילובים אפשריים עם פקודת move

תבנית	דוגמה	איך זה נראה ב-C
move *constant register*	move $0x4 , R1	temp1 = 0x4;
move *constant memory*	move $147 , (R1)	*temp1 = 147;
move *register register*	move R1 , R2	temp2 = temp1;
move *register memory*	move R1 , (R2)	*temp2 = temp1;
move *memory register*	move (R1) , R2	temp2 = *temp1;
move *memory memory*		**לא חוקי בפעולה אחת!**

תרגיל 1

1. **move $0x4050, R1**
2. **move R8,R7**
3. **move (R6,R2) ,R1**
4. **move $20,(R7)**
5. **move R6,-12(R8)**

CPU

רגיסטר	ערך
R1	0x100
R2	0x4
R6	0x100
R7	0x10C
R8	0x120

זכרון

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

עבור כל פקודה - מה יהיה הערך לאחר הביצוע?

תרגיל 1 - פתרון

1. **move $0x4050, R1**
2. **move R8,R7**
3. **move (R6,R2) ,R1**
4. **move $20,(R6)**
5. **move R6,-12(R8)**

CPU

רגיסטר	ערך	1	2	3
R1	0x100	**0x4050**	0x4050	**0xAB**
R2	0x4	0x4	0x4	0x4
R6	0x100	0x100	0x100	0x100
R7	0x114	0x114	**0x120**	0x120
R8	0x120	0x120	0x120	0x120

זיכרון

כתובת	ערך	4	5
0x100	0xFF	**0x14**	0x14
0x104	0xAB	0xAB	0xAB
0x108	0x13	0x13	0x13
0x114	0x14	0x14	**0x100**

*(R6+R2) = (0x100 + 0x4) = (0x104) = 0xAB

* בזיכרון המספרים הם בבסיס 16 לכן צריך להמיר את 20

* 0x120 – 12 (חיסור בבסיסים שונים לכן צריך לבצע התאמה)

תרגיל 2

זיכרון

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

CPU

רגיסטר	ערך
R1	0x100
R2	0x1
R3	0x3

	פקודה	ערך לפני	ערך אחרי
add	addl R2,(R1)		
subtract	subl R3,4(R1)		
multiply	imull $16,(R1,R3,4)		
increment	incl 8(R1)		
decrement	decl R2		
subtract	subl R3,R1		

תרגיל 2 - פתרון

זיכרון

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

CPU

רגיסטר	ערך
R1	0x100
R2	0x1
R3	0x3

	פקודה	ערך לפני	ערך אחרי
add	addl R2,(R1)	0xFF	0x100
subtract	subl R3,4(R1)	0xAB	0xA8
multiply	imull $16,(R1,R3,4)	0x14	0x140
increment	incl 8(R1)	0x13	0x14
decrement	decl R2	0x1	0x0
subtract	subl R3,R1	0x100	0xFD

תרגיל מסכם

אחרי שהבנו אסמבלי והתנסנו בפעולות של השפה

ניקח פונקציה שכתובה ב-c, ונעביר אותה לאסמבלי.

השלבים למעבר:

קוד מקורי ← פישוט קוד ← תרגום ישיר לאסמבלי

הפונקציה הבאה מחפשת את הערך הכי נמוך (המינימום) בתוך מערך

```
int get_min(int * array, unsigned size)
  {
        const int Tmax= -1U >> 1;   (111..1>>1 = 0111…1)
        int min_value = Tmax;
        unsigned i;
        for (i = 0 ; i != size ; i++)
        {
              if (array[i] < min_value)
                    min_value= array[i];
        }
        return min_value;
  }
```

פישוט קוד

אמרנו שאין דבר כזה פעולות מורכבות

- אין for - נשתמש בפקודות JMP, GOTO
  - גם במעבר בין מקטעי קוד נשתמש בפעולות האלה
- אין מערכים - נשתמש במצביעים
- אם יש return לערך - נחזיר את הערך עם הרגיסטר הייעודי R1

פישוט קוד

מפרקים את לולאת ה-for למעין switch עם case-ים (GOTO)

ה-case-ים יהיו:

- i!=size : loop_condition
- i++ : loop_advance
- loop_body: מה שיש בתוך לולאת ה-for

בנוסף, יהיו לנו 2 משתנים: min_value, i

התוכנית המקורית

```
int get_min(int * array, unsigned size)
  {
        const int Tmax= -1U >> 1;
        int min_value = Tmax;
        unsigned i;
        for (i = 0 ; i != size ; i++)
        {
              if (array[i] < min_value)
                    min_value= array[i];
        }
        return min_value;
  }
```

התוכנית לאחר הפישוט

```
int get_min(int * array, unsigned size)
  {
        int min_value= 2147483647;
        unsigned i=0;
        goto loop_condition;

  loop_body:
        if (array[i] >= min_value)  goto loop_advance;
        min_value= array[i];

  loop_advance:
        i++;

  loop_condition:
        if (i != size) goto loop_body;
        return min_value;
  }
```

לאחר העדכון יש ירידה אוטמטית למטה

האם סיימנו?

התוכנית לאחר הפישוט הסופי

```
int get_min(char * array, unsigned size)
{
        int min_value= 2147483647;
        unsigned i=0;
        goto loop_condition;

  loop_body:
        int current_elem= *(int*)(array+i*4);
        if (current_elem >= min_value) goto loop_advance;
        min_value= current_elem;

  loop_advance:
        i++;
  loop_condition:
        if (i != size) goto loop_body;
        return min_value;
  }
```

מצביעים לבית הראשון ובו למקום הראשון

array[i]

התוכנית לאחר הפישוט הראשוני

```
int get_min(int * array, unsigned size)
  {
        int min_value= 2147483647;
        unsigned i=0;
        goto loop_condition;

  loop_body:
        if (array[i] >= min_value)  goto loop_advance;
        min_value= array[i];

  loop_advance:
        i++;

  loop_condition:
        if (i != size) goto loop_body;
        return min_value;
  }
```

מתחילים את האסמבלי - הנחות

1. הערך יוחזר מהרגיסטר R1- לכן min_value יישמר בתוך R1
2. נשמור את הערך current_elem ברגיסטר R3
3. המצביע למערך array יישמר ברגיסטר R2
4. המשתנה i יישמר ברגיסטר R4

```
R1= 2147483647;
R2= array;
R4=0;    i
goto loop_condition;
loop_body:              array + i*4
Current elem  R3= *(int*)(R2+R4*4);
              if (R3 >= R1) goto loop_advance;
              R1= R3;
loop_advance:
              R4++;
loop_condition:
              if (R4 != size) goto loop_body;
              return;
```

ארגון המחסנית

אחרי שהגענו לפישוט מלא נתחיל את כתיבת האסמבלי

קודם כל- מתחילים מסגרת חדשה

אח"כ- מכניסים רגיסטרים למחסנית

זוכרים? הולכים מלמעלה למטה- הכתובת של ראש המחסנית יותר גדולה מהכתובת של תחתית המחסנית!

```
get_min:
        push   R8
        move  R7, R8
```

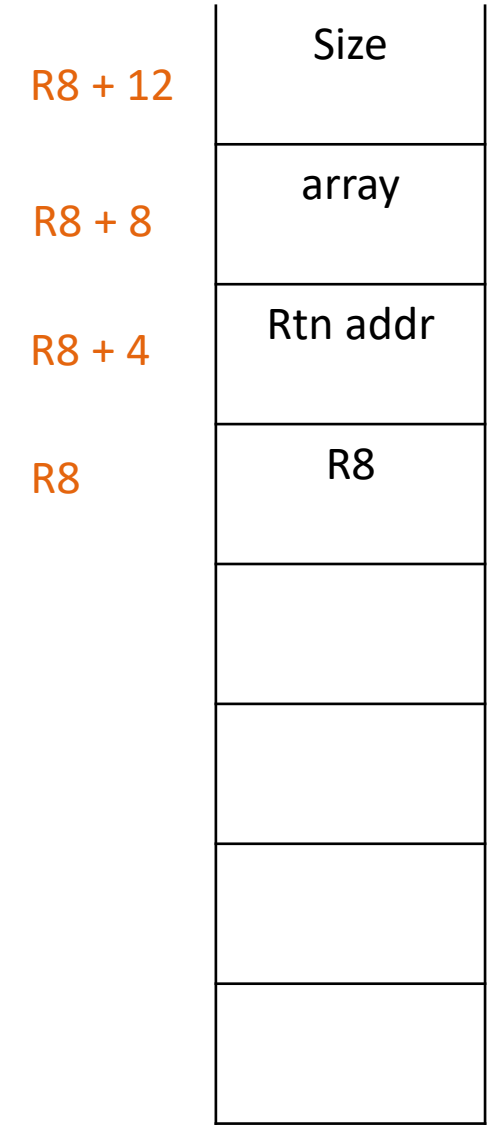
```
R1= 2147483647;
R2= read (R8+8);
R4=0;
goto loop_condition;

loop_body:
        R3= read (R2+R4*4);
        if (R3>=R1) goto loop_advance;
        R1= R3;

loop_advance:
        R4++;

loop_condition:
        if (R4 != read (R8+12)) goto loop_body
    return;
```

read
היא לא פונקציה שקיימת,
זו דרך להציג עבורכם
בצורה פשוטה יותר את המעבר



קלטים לפונקציה יישמרו מעל לתחילת המסגרת שלה

תוצאה סופית באסמבלי

```
R1 = 2147483647;
R2 = read(R8+8);
R4 = 0;
goto loop_condition
loop_body:
   R3= read (R2+R4*4);
   if (R3>=R1)
                    goto loop_advance;
   R1= R3;
loop_advance:
   R4++;
loop_condition:
   if (R4 != read(R8+12))
                    goto loop_body;
```

```
move $2147483647, R1
move  8(R8), R2
xor    R4, R4
jmp   loop_condition
loop_body:
        move   (R2,R4,4),R3
        compare  R3, R1
        jle       loop_advance
        move   R3, R1
loop_advance:
        increment          R4
loop_condition:
        compare  12(R8), R4
       jne       loop_body
```