תרגול 8

MIPS Assembly Language

דוגמא לתרגום תכנית למחשב MIPS

- תרגם את התכנית הבאה לשפת אסמבלי:

```
sort(int v[ ], int n)
{
   int i,j;
   for (i=0; i<n; i++)
        for (j=i-1; j>=0 && v[j]>v[j+1]; j--)
             swap(v, j);   /*  swap v[j] & v[j+1]  */
}
```

פתרון

- נתחיל מהשגרה swap אשר מחליפה תכני שני מקומות בזיכרון.
- שלבי העבודה בשגרות:
  - הקצאת אוגרים למשתני התכנית
  - שמירת תוכן האוגרים שעומדים לעבור שינוי – רק את אלה שצריך לשמור אותם
  - כתיבת קוד השגרה
  - שיחזור ערכי האוגרים

פתרון SWAP

- הקצאת אוגרים:
  - כמוסכמה, פרמטרים מועברים באוגרים $a0-$a3. לכן, הפרמטרים ל-swap יהיו באוגרים $a0,$a1 (v,k בהתאמה).
  - עבור temp נקצה את $t0.
- הקוד:
  - כתובת נתון במחשב MIPS נתונה בבתים (ולא מילים) לכן את כתובת v[k] נקבל ע"י הכפלת k פי 4.

```
muli   $t1, $a1, 4          # reg $t1 = k * 4
add    $t1, $a0, $t1        # reg $t1 = v + (k*4)
                            # reg $t1 has the address of v[k]
```

פתרון SWAP

- נטען את v[k] בעזרת $t1 ואת v[k+1] ע"י הוספת 4 ל-$t1:

```
lw    $t0, 0($t1)        # reg $t0 (temp) = v[k]
lw    $t2, 4($t1)        # reg $t2 = v[k+1]
                         # refers to next element of v
```

- כעת נשמור את תכני $t0 ו-$t2 בכתובות המוחלפות

```
sw    $t2, 0($t1)        # v[k] = reg $t2
sw    $t0, 4($t1)        # v[k+1] = reg $t0 (temp)
```

הפתרון המלא של swap

```
swap: muli   $t1, $a1, 4        # reg $t1 = k * 4
      add    $t1, $a0, $t1      # reg $t1 = v + (k*4)
                                # reg $t1 has the addr of v[k]
      lw     $t0, 0($t1)        # reg $t0 (temp) = v[k]
      lw     $t2, 4($t1)        # reg $t2 = v[k+1]
                                # refers to next element of v
      sw     $t2, 0($t1)        # v[k] = reg $t2
      sw     $t0, 4($t1)        # v[k+1] = reg $t0 (temp)
      jr     $ra                # return to calling procedure
```

פתרון – חזרה אל sort

- הקצאת אוגרים:
  - המשתנים v ו- n נמצאים באוגרים $a0 ו- $a1.
  - המשתנים i ו- j יהיו באוגרים $s0 ו- $s1.
- ראשית נתרגם את פקודת ה-for החיצונית

```
move    $s0, $zero      # i = 0
```

- החלק האחרון בפקודת ה-for יהיה קידום i

```
addi    $s0,$s0,1       # i++
```

פתרון – הלולאה החיצונית

- הלולאה צריכה להסתיים כאשר **לא** מתקיים i<n.
  נשתמש בפקודה slt:

```
for1tst:  slt   $t0,$s0,$a1          # reg $t0=0 if $s0>$a1 (i>=n)
          beq $t0, $zero, exit1      # go to exit1 if $s0>=$a1
```

- הפקודה האחרונה בגוף הלולאה מבצעת קפיצה בלתי מותנית לשורת הבדיקה:

```
          j     for1tst              # jump to test of outer loop
exit1:
```

פתרון – הלולאה החיצונית

- נקבל ששלד הלולאה הראשונה יהיה:

```
        move  $s0, $zero
for1tst: slt   $t0,$s0,$a1        # reg $t0=0 if $s0>$a1
                                     i.e. (i>=n)
         beq   $t0, $zero, exit1  # go to exit1 if $s0>=$a1
               …
               (body of outer loop)
               …
         addi  $s0,$s0,1          # i++
         j     for1tst            # jump to test of outer loop
exit1:
```

פתרון – הלולאה הפנימית

- נעבור ללולאה הפנימית - אתחול:

```
addi    $s1, $s0, -1        # j = i – 1
```

- הפחתה מהמונה בסוף גוף הלולאה:

```
addi    $s1, $s1, -1        # j - -
```

- תנאי הלולאה מורכב משני חלקים. הלולאה תסתיים אם אחד משני החלקים לא יתקיים. נבדוק ראשית את התנאי j<0:

```
for2tst:  slti  $t0, $s1, 0         # reg $t0 = 1 if $s1 < 0 (j<0)
          bne $t0, $zero, exit2     # go to exit2 if $s1<0 (j<0)
```

פתרון – הלולאה הפנימית

- התנאי השני יגרום ליציאה מהלולאה אם לא מתקיים [v[j] > v[j+1].
  ראשית נחשב את הכתובת ע"י הכפלת j ב-4 היות וכתובת נתונים היא תמיד בבתים

```
muli    $t1, $s1, 4          # reg $t1 = j * 4
add     $t2, $a0, $t1        # reg $t2 = v + (j*4)
```

- נטען את v[j] לאוגר $t3:

```
lw      $t3, 0($t2)          # reg $t3 = v[j]
```

- משום שהאיבר השני נמצא במילה שאחרי מילת האיבר הראשון,
  נטען את v[j+1] אל האוגר $t4 ע"י הוספת 4 לכתובת שב-$t2

```
lw      $t4, 4($t2)          # reg $t4 = v[j+1]
```

פתרון – הלולאה הפנימית

- הבדיקה v[j+1] < v[j] :

```
slt     $t0, $t4, $t3          # reg $t0 = 0 if $t4 >= $t3
beq     $t0, $zero, exit2      # go to exit2  if $t4 >= $t3
```

- הפקודה האחרונה בגוף הלולאה תקפוץ לשורת הבדיקה הראשונה

```
j       for2tst                # jump to test if inner loop
```

פתרון – שלד הלולאות

- נשלב את החלקים ונקבל את שלד הלולאה הפנימית:

```
        addi  $s1, $s0, -1        # j = i – 1
for2tst: slti  $t0, $s1, 0        # reg $t0 = 1  if  $s1 < 0 i.e. (j<0)
        bne   $t0, $zero, exit2   # go to exit2  if  $s1 < 0 i.e. (j<0)
        muli  $t1, $s1, 4         # reg $t1 = j * 4
        add   $t2, $a0, $t1       # reg $t2 = v + (j*4)
        lw    $t3, 0($t2)         # reg $t3 = v[j]
        lw    $t4, 4($t2)         # reg $t4 = v[j+1]
        slt   $t0, $t4, $t3       # reg $t0 = 0 if $t4 >= $t3
        beq   $t0, $zero, exit2   # go to exit2  if $t4 >= $t3
              …
              (body of inner loop)
              …
addi    $s1, $s1, -1              # j - -
j       for2tst                   # jump to test if inner loop
```

פתרון

- גוף הלולאה הפנימית מורכב מקריאה לפרוצדורה swap אשר יש להעביר לה פרמטרים. העברת הפרמטרים ל-swap תיעשה ע"י הפקודות:

```
move $a0, $s2        #first swap param is v
move $a1, $s1        #second swap param is j
```

- והקריאה לפרוצדורה ע"י הפקודה

```
jal     swap
```

- **אבל** swap מצפה שהפרמטרים שנשלחים יהיו באוגרים $a0,$a1 **אשר בשימוש sort -> בעיה זה** ☹
- **מה עושים ?**

פתרון שליחת הפרמטרים

- **פתרון**:
התכנית SWAP לא תשתמש באוגרים $a0,$a1 אלא:
- נעתיק את תוכן האוגרים $a0,$a1 אל $s2,$s3 **בתחילת התכנית** ונעבוד איתם (מהיר יותר משמירתם במחסנית זמן הריצה).

```
move    $s2, $a0    # copy param $a0 into $s2
move    $s3, $a1    # copy param $a1 into $s3
```

- כלומר, יש לשנות את התכנית שראינו ולבצע Find&Replace אשר ישנה את כל ה-$a0 ל-$s2 ואת כל ה-$a1 ל-$s3.

פתרון

- אחרון אחרון חביב – הקצאת האוגרים ושחרורם:
  התכנית עושה שימוש באוגרים $s0,$s1,$s2,$s3 אשר בתחילת התכנית יש לשמור את תוכנם בנוסף לשמירת כתובת החזרה:

```
addi   $sp,$sp,-20        # make room on stack for 5 regs
sw     $ra,16($sp)        # save $ra on stack
sw     $s3,12($sp)        # save $s3 on stack
sw     $s2,8($sp)         # save $s2 on stack
sw     $s1,4($sp)         # save $s1 on stack
sw     $s0,0($sp)         # save $s0 on stack
```

- הערה: יש לשמור את תוכן $ra בגלל קריאות ה-jal ל-swap אשר משנות אותו לכתובת החזרה ב-sort מ-swap.

התמונה המלאה

```
sort:     addi    $sp,$sp,-20          # make room on stack for 5 regs
          sw      $ra,16($sp)          # save $ra on stack
          sw      $s3,12($sp)          # save $s3 on stack
          sw      $s2,8($sp)           # save $s2 on stack
          sw      $s1,4($sp)           # save $s1 on stack
          sw      $s0,0($sp)           # save $s0 on stack

          move    $s2, $a0             # copy param $a0 into $s2
          move    $s3, $a1             # copy param $a1 into $s3

          move    $s0, $zero
for1tst:  slt     $t0,$s0,$s3          # reg $t0=0 if $s0>=$s3(i≥n)
          beq     $t0, $zero, exit1    # go to exit1 if $s0>=$s3(i≥n)
          addi    $s1, $s0, -1         # j = i – 1
For2tst:  slti    $t0, $s1, 0          # reg $t0 = 1 if $s1 < 0 (j<0)
          bne     $t0, $zero, exit2    # go to exit2 if $s1<0    (j<0)
          muli    $t1, $s1, 4          # reg $t1 = j * 4
          add     $t2, $s2, $t1        # reg $t2 = v + (j*4)
```

התמונה המלאה - המשך

```
        lw      $t3, 0($t2)          # reg $t3 = v[j]
        lw      $t4, 4($t2)          # reg $t4 = v[j+1]
        slt     $t0, $t4, $t3        # reg $t0 = 0 if $t4 >= $t3
        beq     $t0, $zero, exit2    # go to exit2  if $t4 >= $t3

        move    $a0, $s2             # first swap param is v
        move    $a1, $s1             # second swap param is j
        jal     swap                 # see swap code earlier

        addi    $s1, $s1, -1         # j - -
        j       for2tst              # jump to test of inner loop
exit2:  addi    $s0, $s0, 1          # i++
        j        for1tst             # jump to test of outer loop

exit1:  lw      $s0, 0($sp)          # restore $s0 from stack
        lw      $s1, 4($sp)          # restore $s1 from stack
        lw      $s2, 8($sp)          # restore $s2 from stack
        lw      $s3, 12($sp)         # restore $s3 from stack
        lw      $ra, 16($sp)         # restore $ra from stack
        addi    $sp, $sp, 20         # restore stack pointer
        jr      $ra
```

דוגמא לתרגום תכנית למחשב MIPS

- תרגם את פונקצית strcpy של שפת C לשפת אסמבלי:

```
void strcpy(char x[ ], char y[ ])
{
   int i;

   i=0;
   while ((x[i]=y[i]) != 0)
        i++;
}
```

פתרון

- הקצאת אוגרים:
  - כתובות תחילת המערכים x ו-y נתונות ב:$a0 ו-$a1 בהתאמה.
  - נקצה את $s0 עבור i.
- שלב 1: strcpy מעדכנת את sp ומקצה לו מקום במחסנית בגלל השימוש ב- $s0 :

```
addi   $sp,$sp,-4          # adjust stack for 1 more item
sw     $s0, 0($sp)         # save $s0
```

פתרון

- <u>שלב 2:</u> אתחול l

```
        move $s0, $zero         # i = 0
```

- <u>שלב 3:</u> תחילת הלולאה – כתובת [i]y מחושבת תחילה ע"י הוספת i ל []y

```
l1:     add $t1, $a1, $s0       # y[i]’s addr is in $t1
```

- *נשים לב* שאין צורך להכפיל את i ב-4 משום שזהו מערך תווים/בתים.

פתרון

- <u>שלב 4:</u> על מנת לטעון את התו שב-y[i] נשתמש ב-lb שישים את התו ב-$t2:

```
lb      $t2, 0($t1)          # $t2 = y[i]
```

- <u>שלב 5:</u> באופן דומה נחשב את הכתובת של x[i] ונציב אותה ב-$t3.את התו שנמצא בכתובת זו נטען אל $t2

```
add     $t3, $a0, $s0        # address of x[i] into $t3
sb      $t2, 0($t3)          # x[i] = y[i]
```

- <u>שלב 6:</u> קידום i והמשך הלולאה אם התו אינו 0 (טרם הגענו לסוף המחרוזת)

```
addi    $s0, $s0, 1          # i++
bne     $t2, $zero, l1       # if y[i] != 0 then go to l1
```

פתרון

- שלב 7: אם לא ממשיכים בלולאה, נתקלנו בתו האחרון במחרוזת; נשחזר את $s0 ואת sp ונחזור מהפונקציה:

```
lw      $s0, 0($sp)         # y[i]==0 -> end of string
                            # restore old s0
addi    $sp, $sp, 4         # pop 1 word off stack
jr      $ra                 # return
```

התמונה המלאה

```
strcpy: addi   $sp,$sp,-4          # adjust stack for 1 more item
        sw     $s0, 0($sp)         # save $s0

        move   $s0, $zero          # i = 0
l1:     add    $t1, $a1, $s0       # y[i]'s addr is in $t1
        lb     $t2, 0($t1)         # $t2 = y[i]
        add    $t3, $a0, $s0       # address of x[i] into $t3
        sb     $t2, 0($t3)         # x[i] = y[i]
        add    $s0, $s0, 1         # i++
        bne    $t2, $zero, l1      # if y[i] != 0 then go to l1

        lw     $s0, 0($sp)         # y[i]==0 -> end of string
                                   # restore old s0
        addi   $sp, $sp, 4         # pop 1 word off stack
        jr     $ra                 # return
```