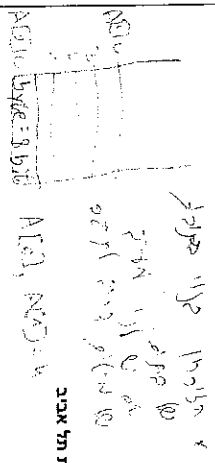


\$s0 \$s1 \$s2  
f - (g + h) - (i + j)

### תרגול 8

add \$t0 \$s2 \$s1  
add \$t1 \$s3 \$s4

## MIPS Assembly Language



אלון שקלר - אוניברסיטת תל אביב

A [E]:  $2^k \cdot (ACS)$

sw \$t0 \$t1  
sw \$t0 4(\$s2)

### פתרון

- נתחיל מהשגרה swap אשר מחליפה תכני שני מקומות בזיכרון.
- שליבי העבודה בשגרות:

- הקצאת אגורים למשתני התכנית
- שמירת תוכן האגורים שעומדים לעבור שינוי - רק את אלה שצריך לשמור אותם
- כתיבת קוד השגרה
- שיחזור ערכי האגורים

## דוגמא לתרגום תכנית למחשב MIPS

Example taken from the book: "Computer Organization and Design: The Hardware/Software Interface" by David A. Patterson, John L. Hennessy and John L. Hennessy. Morgan Kaufmann Publishers Inc. - All rights reserved

• תרגום 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] */
}
```

אלון שקלר - אוניברסיטת תל אביב

### פתרון SWAP

- הקצאת אגורים:
- כמוסכמה, פרמטרים מעברים באגורים \$a0-\$a3, לכן, הפרמטרים ל-swap יהיו באגורים \$a0, \$a1 (א, v בהתאמה).
- עבור temp נקצה את \$t0.
- הקוד:
- כתיבת נתון במחשב MIPS נתונה בבתיים (ולא מילים) לכן את כתיבת [א] נקבל ע"י הכפלת א פי 4.
- # reg \$t1 = k \* 4
- add \$t1, \$a0, \$t1 # reg \$t1 = v + (k\*4)
- # reg \$t1 has the address of [א]

## 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
```

• נעת 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)
```

אלון שקלר – אוניברסיטת תל אביב

5

## 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
```

אלון שקלר – אוניברסיטת תל אביב

5

## פתרון – חזרה אל sort

- הקצאת אוגרים:
  - המשתנים  $v$  ו- $i$  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++
```

אלון שקלר – אוניברסיטת תל אביב

7

## פתרון – הלולאה החיצונית

- הלולאה צריכה להסתתים כאשר לא מתקיים `חא1`.  
נשתמש בפקודה `slt`:
 

```
for1st: slt $t0, $s0, $a1 # reg $t0=0 if $s0>$a1 (>=ח)
        beq $t0, $zero, exit1 # go to exit1 if $s0>=$a1
```
  - הפקודה האחרונה במוף הלולאה מבצעת קפיצה בלתי מותנית לשורת הבדיקה:
 

```
        j for1st # jump to test of outer loop
```
- exit1:

אלון שקלר – אוניברסיטת תל אביב

8

## פתרון – הלולאה החיצונית

- נקבל ששילד הלולאה הראשונה יהיה:

```

move $s0, $zero
for1st: 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    for1st        # jump to test of outer loop
exit1:

```

אלון שקלר – אוניברסיטת תל אביב

9

## פתרון – הלולאה הפנימית

- נעבור ללולאה הפנימית - אתחול:

```

addi    $s1, $s0, -1      # j = i - 1
        ...
        הפחתה מהמונה בסוף גוף הלולאה:
        addi    $s1, $s1, -1      # j --

```

- תגאי הלולאה מורכב משני חלקים. הלולאה תסתיים אם אחד משני החלקים לא יתקיים. נבדוק ראשית את התנאי  $j < n$ :
 

```

for2st: slt $t0, $s1, 0      # reg $t0 = 1 if $s1 < 0 (j < 0)
        bne $t0, $zero, exit2 # go to exit2 if $s1 < 0 (j < 0)
            
```

אלון שקלר – אוניברסיטת תל אביב

10

## פתרון – הלולאה הפנימית

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

```

umul   $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 שהאוגר השני נמצא במילה שאחרי מילת האובר 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]
```

אלון שקלר – אוניברסיטת תל אביב

11

## פתרון – הלולאה הפנימית

- הבדיקה  $v[j] < v[j+1]$ :

```

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

```

- הפקודה האחרונה בגוף הלולאה תקפוע לשרות הבדיקה הראשונה

```

j      for2st          # jump to test if inner loop

```

אלון שקלר – אוניברסיטת תל אביב

12

## פתרון – שלד הלולאות

• נשלב את החלקים ונקבל את שלד הלולאה הפנימית:

```

addi $s1, $s0, -1      # j = i - 1
for2ist: 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)
multi $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]
slti $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--
for2ist: jne $s1, $s1, -1      # jump to test if inner loop
j
    
```

אלון שקלר – אוניברסיטת תל אביב

13

## 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 שנושלחים יהיו באוגרים  
 • \$a0, \$a1 אשר בשימוש sort - בעיה זה ☹ מה עושים ?

אלון שקלר – אוניברסיטת תל אביב

14

## פתרון שלילת הפרמטרים

• **פתרון:**  
 התכנית SWAP לא תשתמש באוגרים \$a0, \$a1, \$a2, \$a3 אלא:  
 • נעתיק את תוכן האוגרים \$a0, \$a1 אל \$s2, \$s3  
**בתחילת התכנית** ונעבוד אותם (מהיר יותר משמירתם במתכנית 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.

אלון שקלר – אוניברסיטת תל אביב

15

## 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.

אלון שקלר – אוניברסיטת תל אביב

16

## התמונה המלאה

```

sort:  addi    $ra,$ssp,-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        # reg $t0=0 if $s0>=$s3(!zn)
for1st:  slt    $t0,$s0,$s3     # go to exit1 if $s0>=$s3(!zn)
      beq    $t0,$zero,exit1  # j = i - 1
      addi   $t1,$s0,-1       # reg $t0 = 1 if $s1 < 0 (j<0)
for2st:  slti   $t0,$s1,0     # go to exit2 if $s1<0 (j<0)
      bne    $t0,$zero,exit2  # reg $t1 = j * 4
      multi  $t1,$s1,4        # reg $t2 = v + (j*4)
      add    $t2,$s2,$t1
  
```

אלון שקלר – אוניברסיטת תל אביב 17

## דוגמא לתרגום תכנית למחשב MIPS

Example taken from the book: "Computer Organization and Design: The Hardware/Software Interface" by David A. Patterson, John L. Hennessy and John L. Hennessy. Morgan Kaufmann Publishers Inc. – All rights reserved

```

:תרגם את פונקציית strcpy של שפת C לשפת אסמבל:
void strcpy(char x[], char y[])
{
  int i;
  i=0;
  while ((x[i]=y[i]) != 0)
    i++;
}
  
```

אלון שקלר – אוניברסיטת תל אביב 19

## התמונה המלאה - המשך

```

      lw     $t3,0($t2)        # reg $t3 = v[i]
      lw     $t4,4($t2)        # reg $t4 = v[i+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   $t1,$s1,-1       # j --
      j      $t1,$zero,for2st # jump to test of inner loop
      addi   $t0,$s0,1        # i++
      j      $t0,$zero,for1st # 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
  
```

אלון שקלר – אוניברסיטת תל אביב 18

## פתרון

- הקצאת אוגרים:
  - כמותות תחילת המערכים א ו-י נחנות ב: \$a0 ו-1 \$a1 בהתאמה.
  - נקצה את \$s0 עבור j.
- 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

אלון שקלר – אוניברסיטת תל אביב 20

## פתרון

- שלב 2: אתחול |  
move \$s0, \$zero # i = 0
  - שלב 3: תחילת הלולאה – כתובת [i] מוחשבת תחילה ע"י הוספת i ל [i]
  - 11: add \$t1, \$a1, \$s0 # [i]'s addr is in \$t1
- נשים לב שאין צורך להכפיל את i ב-4 משום שזוהי מערך תווים/בתים.

אליו שקלר – אוניברסיטת תל אביב  
ד"ר משה גורן (תואר ראשון)

## 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

אליו שקלר – אוניברסיטת תל אביב

23

## פתרון

- שלב 4: על מנת לטעון את התו שב-[i] נשתמש ב-\$s0 ושישים את התו ב-\$t2:  
lb \$t2, 0(\$t1) # \$t2 = y[i]
- שלב 5: באופן דומה נושב את הכתובת של [i+1] ונוציב 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]
- שלב 6: קידום ו הומשך הלולאה אם התו אינו 0 (טרם הגענו לסוף המחרוזת)  
addi \$s0, \$s0, 1 # i++  
bne \$t2, \$zero, 11 # if y[i] != 0 then go to 11

אליו שקלר – אוניברסיטת תל אביב

22

## התמונה המלאה

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

11:     move $s0, $zero # i = 0
        add  $t1, $a1, $s0 # [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, 11 # if y[i] != 0 then go to 11

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
```

אליו שקלר – אוניברסיטת תל אביב

24

# D1D104

## MIPS operands

| Name                         | Example                                                                          | Comments                                                                                                                                                                                                             |
|------------------------------|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 32 registers                 | \$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at | Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register \$zero always equals 0. Register \$at is reserved for the assembler to handle large constants.                      |
| 2 <sup>30</sup> memory words | Memory[0], Memory[4], ..., Memory[4294967292]                                    | Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls. |

## MIPS assembly language

| Instruction        | Example                                                                                                                                                                                                                                                                                                                              | Comments                                                                                                                                                                                                                                                                                                                   |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Arithmetic         | add \$s1, \$s2, \$s3<br>subtract \$s1, \$s2, \$s3                                                                                                                                                                                                                                                                                    | Three operands; data in registers                                                                                                                                                                                                                                                                                          |
| Data transfer      | addi \$s1, \$s2, 100<br>load word \$s1, 100(\$s2)<br>store word \$s1, 100(\$s2)<br>load byte \$s1, 100(\$s2)<br>store byte \$s1, 100(\$s2)<br>load upper immediate \$s1, 100<br>branch on equal \$s1, \$s2, 25<br>branch on not equal \$s1, \$s2, 25<br>set on less than \$s1, \$s2, \$s3<br>set less than immediate \$s1, \$s2, 100 | Used to add constants<br>Word from memory to register<br>Word from register to memory<br>Byte from memory to register<br>Byte from register to memory<br>Loads constant in upper 16 bits<br>Equal test; PC-relative branch<br>Not equal test; PC-relative<br>Compare less than; for beq, bne<br>Compare less than constant |
| Control            | beq \$s1, \$s2, 25<br>bne \$s1, \$s2, 25<br>slt \$s1, \$s2, \$s3<br>slti \$s1, \$s2, 100<br>j 2500<br>jr \$ra<br>jal 2500                                                                                                                                                                                                            | Equal test; PC-relative branch<br>Not equal test; PC-relative<br>Compare less than; for beq, bne<br>Compare less than constant<br>Jump to target address<br>For switch, procedure return<br>For procedure call                                                                                                             |
| Unconditional jump | jal 2500                                                                                                                                                                                                                                                                                                                             | For procedure call                                                                                                                                                                                                                                                                                                         |

