מבנה מחשבים - תרגיל 5#

אריאל סטולרמן
קבוצה 02

(1)

הערה:

בקריאה לפרוצדורה *b_sort* לא נעשתה שמירת משתנים במחסנית ושחזורם בתום הפרוצדורה, כיוון שלא היו קריאות לפרוצדורות מתוך *b_sort*, ולכן לא היה צורך בגיבוי *$a0, $a1, $ra*.

להלן הקוד עבור *bubble-sort*:

```
### Ariel Stolerman
### Computer Structure fall 08', ex05q01

      .text
### main

main:
      addi $sp, $sp, -12      # save $ra & pramas to be used on stack
      sw $ra, 8($sp)          # save $ra on stack - since jal calls will occur
      sw $s1, 4($sp)          # save $s1 on stack
      sw $s0, 0($sp)          # save $s0 on stack

      li $v0, 4               # prints msg to user
      la $a0, ask_input_size  # to enter the array size
      syscall                 #

get_size:
      li $v0, 5               # receives the array size
      syscall                 #

      ble $v0, 1, in_size_err # if array size <= 1, goto err msg
      move $s1, $v0           # else set $s1: input array size

      mul $t0, $s1, -4        # allocating memory for the array on stack
      add $sp, $sp, $t0       #
      la $s0, 0($sp)          # set $s0: base address of the input array
      move $t0, $s0           # set start address for receiving input
      move $t1, $zero         # set index

      li $v0, 4               # prints msg to user
      la $a0, ask_input       # to enter the array to sort
      syscall                 # of the size given earlier
      li $v0, 1               #
      move $a0, $s1           #
```

```
      syscall                 #
      li $v0, 4               #
      la $a0, two_dots        #
      syscall                 #

      j for_readarr           # goto read input array from user

in_size_err:
      li $v0, 4               # print err msg to user
      la $a0, input_size_err  # and asks for another
      syscall                 # array input
      j get_size              # get another array size from user


for_readarr:                  # read input from user loop
      li $v0, 5               # read int
      syscall                 #
      sw $v0, 0($t0)          # save input in memory
      addi $t0, $t0, 4        # advance t0 (to the next word)
      addi $t1, $t1, 1        # inc $t1
      beq $t1, $s1, continue  # if ($t1 == array-size) goto continue
      j for_readarr           # else run another iteration

continue:
      move $a0, $s0           # calling b_sort on input array
      move $a1, $s1           # with $a0 as base address of the input
      jal b_sort              # array and $a1 its size

      li $v0, 4               # print the sorted array
      la $a0, output          #
      syscall                 #

      move $t0, $s0           # set start address for printing
      move $t1, $zero         # set index

for_printarr:
      li $v0, 4               # print current element
      la $a0, space           #
      syscall                 #
      li $v0, 1               #
      lw $a0, 0($t0)          #
      syscall                 #
      addi $t0, $t0, 4        # advance $t0 to the next element (word)
      addi $t1, $t1, 1        # inc $t1
```

```
      beq $t1, $s1, exit      # if ($t1 == array-size), goto exit (finish)
      j for_printarr          # else run another iteration

### procedure: b_sort

b_sort:

# a0: array base address, a1: array length
# no procedures calls, and so no need to save on stack
# and restore later any of the registers

      addi $a1, $a1, -1       # set $a1: array max index

do_while_sort:
      move $t2, $zero         # $t2 will be used as an indicator whether
                              # a swap had been made during this while iteration
      addi $a1, $a1, -1       # decreasing upper limit for the following for-loop
      move $t0, $a0           # set $t0 to the beginning of the array
      move $t1, $zero         # set $t1 as index, denote i

for_swapping:
      lw $t3, 0($t0)          # set $t3: array[i]
      lw $t4, 4($t0)          # set $t4: array[i+1]

      ble $t3, $t4, for_cont  # skip the next steps if array[i] <= array[i+1]
      addi $t2, $zero, 1      # set $t2 to 1 - a swap had been made during
                              # this while iteration

      sw $t3, 4($t0)          # swap array[i] with array[i+1]
      sw $t4, 0($t0)          #

for_cont:                     # (jumps here if swap had not been made)
      addi $t0, $t0, 4        # advance $t0 to the next element
      addi $t1, $t1, 1        # inc $t1
      ble $t1, $a1, for_swapping    # if not finished the current for loop,
                                    # run another iteration
      bne $t2, $zero, do_while_sort # if a swap had been made during the last
                                    # for, run another while iteration

      jr $ra                  # else finish - return to calling procedure

### exit program

exit:
```

```
	li $v0, 4		# add "\n" before exiting
	la $a0, newline		#
	syscall			#
	mul $t0, $s1, 4		# restore stack pointer to before
	add $sp, $sp, $t0	# receiving array input
	lw $s0, 0($sp)		# restore $s0 from stack
	lw $s1, 4($sp)		# restore $s1 from stack
	lw $ra, 8($sp)		# restore $ra from stack
	addi $sp, $sp, 12	# restore stack pointer
	jr $ra

	.data

# strings

ask_input_size:
	.asciiz "Please enter an array size bigger than 1:\n"
input_size_err:
	.asciiz "Error! Please enter an array size bigger than 1:\n"
ask_input:
	.asciiz "Please enter an array of size "
two_dots:
	.asciiz ":\n"
output:
	.asciiz "\nThe sorted array is: "
space:
	.asciiz " "
newline:
	.asciiz "\n"
```

(2)

```
### Ariel Stolerman
### Computer Structure fall 08', ex05q02

      .text

# Procedure: convert
# parameters: $a0: some null-terminated string
# returns $v0 s.t.:
# - if $a0 represents an integer between 0 and (2^31-1),
#   $v0 will contain the number as int.
# - if $a0 represents an integer bigger than (2^31-1), $v0 will
#   contain some number.
# - in any other case $v0 will contain -1

convert:
      addi $sp, $sp, -8 # allocate memory in stack for $s0 & $ra
      sw $ra, 4($sp)          # save $ra on stack
      sw $s0, 0($sp)          # save $s0 on stack

      move $v0, $zero         # initialize $v0

while_not_empty:
      lb $s0, 0($a0)          # set $s0: next char in $a0

      beq $s0, $zero, exit    # in case the next char is NULL, goto return $v0

      addi $s0, $s0, -48      # reduce 48 from int stored in $s0
      blt $s0, $zero, not_int # if the (ASCII code - 48) of the next char
      addi $t0, $zero, 9      # is less than 0 or more than 9, then it's not
      blt $t0, $s0, not_int   # a digit, goto return -1

      mul $v0, $v0, 10  # else update $v0
      add $v0, $v0, $s0 #

      addi $a0, $a0, 1  # advance $a0 to the next char (byte)
      j while_not_empty # loop until end of $a0 (null)

not_int:
      move $v0, $zero         # the string contains some char that is
      addi $v0, $v0, -1 # not a digit, therefore return -1

exit:
      lw $s0, 0($sp)          # restore $s0 from stack
```

```
move $t0, $ra           # restore $ra from stack
lw $ra, 4($sp)          #
addi $sp, $sp, 8  # restore stack pointer
jr $t0                  # return to calling procedure
```