Project

- Phase2 -

(Computer Architecture)

Team A

2013311659 곽창근

2013312608 문성암

-Index-

1. Before starting phase #2

We modified a few codes. Because we not accustomed to translate double type data instructions for MIPS. So, we modify double to float. And, We made it identifiable by writing it in bold letters and underlines in the first phase report. In total, eight places were modified in the first phase report.

(This explanation was written in the first phase report too.)

- Phase #2

1. Assembly Code

```
# $s0 = i
# $s1 = j
# $s2 = k
# $s3 = l
# $s4 = m
# $s5 = n
# $s6 = cities' address
# $s7 = distances' address
# $f16 = min
# $f17 = temp
main:
	#store cities' coordinates.
	la $s6, cities		# store cities' address at $s6
	li $t1, -1
	sw $t1, 0($s6)
	sw $t1, 4($s6)		# store {-1,-1}
	li $t1, 0
	sw $t1, 8($s6)
	sw $t1, 12($s6)		# store {0,0}
```

```
	li $t1, 2
	sw $t1, 16($s6)
	li $t1, 6
	sw $t1, 20($s6)		# store {2,6}
	li $t1, 8
	sw $t1, 24($s6)
	li $t1, 4
	sw $t1, 28($s6)		# store {8,4}
	li $t1, 7
	sw $t1, 32($s6)
	li $t1, 2
	sw $t1, 36($s6)		# store {7,2}
	li $t1, 1
	sw $t1, 40($s6)
	li $t1, 6
	sw $t1, 44($s6)		# store {1,6}
	li $t1, 4
	sw $t1, 48($s6)
	li $t1, 9
	sw $t1, 52($s6)		# store {4,9}
	li $t1, 3
	sw $t1, 56($s6)
	li $t1, 2
	sw $t1, 60($s6)		# store {3,2}

# Starting point of 'for' loops for distances[8][8]!
	la $s5, tempmemory
	la $s7, distances
	li $s0, 1	# i
```

```
L1:

    li $s1, 1       # j
L2:
    sll $t1, $s0, 3         # $t1 = i*8
    sll $t2, $s1, 3         # $t2 = j*8
    add $t1, $s6, $t1       # store cities[i]'s address
    add $t2, $s6, $t2       # store cities[j]'s address
    lw $a0, 0($t1)
    lw $a1, 4($t1)
    lw $a2, 0($t2)
    lw $a3, 4($t2)
    # $a0=city1.x, $a1=city1.y, $a2=city2.x, $a3=city2.y
    jal CalDistance         #call CalDistance
    sll $t1, $s0, 5         # $t1 = i*32
    sll $t2, $s1, 2         # $t2 = j*4
    add $t3, $t1, $t2
    add $t3, $t3, $s7       # $t3 stores distances[i][j]'s address
    swc1 $f18, 0($t3)
T2:
    addi $s1, $s1, 1        # j++
    slti $t0, $s1, 8        # if j < 8, set
    bne $t0, $zero, L2      # go back to L2

T1:
    addi $s0, $s0, 1        # i++
    slti $t0, $s0, 8        # if i < 8, set
    bne $t0, $zero, L1      # go back to L1
# Ending point of 'for' loops for distances[8][8]!
```

```
	lwc1 $f16, bigFloat  # min = 999999999.99
	la $t1, resultpath
	li $t0, 1
	sw $t0, 0($t1)		# result_path[0] = 1
	sw $t0, 28($t1) # result_path[7] = 1

# Starting point of the for loops!
	li $s0, 2	# i
loopI:

	li $s1, 2	# j
loopJ:
	beq $s1, $s0, tailJ	# if j==i, go to tailJ
	# calculate temp
	sll $t0, $s0, 2
	addi $t2, $t0, 32
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[1][i]
	add.s $f17, $f0, $f4 # temp = 0 + distances[1][i]
	sll $t1, $s0, 5
	sll $t0, $s1, 2
	add $t2, $t1, $t0
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[i][j]
	add.s $f17, $f17, $f4		# temp += distances[i][j]
	sll $t1, $s1, 5
	addi $t2, $t1, 4
	add $t2, $t2, $s7
```

```
    lwc1 $f4, ($t2)         # distances[j][1]
    add.s $f17, $f17, $f4           # temp += distances[j][1]
    #ending point of calculating temp
    c.lt.s $f16, $f17       # if(min<temp)
    bc1t tailJ    # j++

    li $s2, 2     # k
loopK:
    beq $s2, $s0, tailK  # if k==i, go to tailK
    beq $s2, $s1, tailK  # if k==j, go to tailK
    # calculate temp
    sll $t0, $s0, 2
    addi $t2, $t0, 32
    add $t2, $t2, $s7
    lwc1 $f4, ($t2)         # distances[1][i]
    add.s $f17, $f0, $f4 # temp = 0 + distances[1][i]
    sll $t1, $s0, 5
    sll $t0, $s1, 2
    add $t2, $t1, $t0
    add $t2, $t2, $s7
    lwc1 $f4, ($t2)         # distances[i][j]
    add.s $f17, $f17, $f4           # temp += distances[i][j]
    sll $t1, $s1, 5
    sll $t0, $s2, 2
    add $t2, $t1, $t0
    add $t2, $t2, $s7
    lwc1 $f4, ($t2)         # distances[j][k]
    add.s $f17, $f17, $f4           # temp += distances[j][k]
    sll $t1, $s2, 5
```

```
    addi $t2, $t1, 4
    add $t2, $t2, $s7
    lwc1 $f4, ($t2)         # distances[k][1]
    add.s $f17, $f17, $f4           # temp += distances[k][1]
    #ending point of calculating temp
    c.lt.s $f16, $f17       # if(min<temp)
    bc1t tailK    # k++

    li $s3, 2      # l
loopL:
    beq $s3, $s0, tailL
    beq $s3, $s1, tailL
    beq $s3, $s2, tailL
    # calculate temp
    sll $t0, $s0, 2
    addi $t2, $t0, 32
    add $t2, $t2, $s7
    lwc1 $f4, ($t2)         # distances[1][i]
    add.s $f17, $f0, $f4 # temp = 0 + distances[1][i]
    sll $t1, $s0, 5
    sll $t0, $s1, 2
    add $t2, $t1, $t0
    add $t2, $t2, $s7
    lwc1 $f4, ($t2)         # distances[i][j]
    add.s $f17, $f17, $f4           # temp += distances[i][j]
    sll $t1, $s1, 5
    sll $t0, $s2, 2
    add $t2, $t1, $t0
    add $t2, $t2, $s7
```

```
	lwc1 $f4, ($t2)		# distances[j][k]
	add.s $f17, $f17, $f4		# temp += distances[j][k]
	sll $t1, $s2, 5
	sll $t0, $s3, 2
	add $t2, $t1, $t0
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[k][l]
	add.s $f17, $f17, $f4		# temp += distances[k][l]
	sll $t1, $s3, 5
	addi $t2, $t1, 4
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[l][1]
	add.s $f17, $f17, $f4		# temp += distances[l][1]
	#ending point of calculating temp
	c.lt.s $f16, $f17	# if(min<temp)
	bc1t tailL	# l++

	li $s4, 2	# m
loopM:
	beq $s4, $s0, tailM
	beq $s4, $s1, tailM
	beq $s4, $s2, tailM
	beq $s4, $s3, tailM
	# calculate temp
	sll $t0, $s0, 2
	addi $t2, $t0, 32
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[1][i]
	add.s $f17, $f0, $f4 # temp = 0 + distances[1][i]
```

```
sll $t1, $s0, 5
sll $t0, $s1, 2
add $t2, $t1, $t0
add $t2, $t2, $s7
lwc1 $f4, ($t2)        # distances[i][j]
add.s $f17, $f17, $f4          # temp += distances[i][j]
sll $t1, $s1, 5
sll $t0, $s2, 2
add $t2, $t1, $t0
add $t2, $t2, $s7
lwc1 $f4, ($t2)        # distances[j][k]
add.s $f17, $f17, $f4          # temp += distances[j][k]
sll $t1, $s2, 5
sll $t0, $s3, 2
add $t2, $t1, $t0
add $t2, $t2, $s7
lwc1 $f4, ($t2)        # distances[k][l]
add.s $f17, $f17, $f4          # temp += distances[k][l]
sll $t1, $s3, 5
sll $t0, $s4, 2
add $t2, $t1, $t0
add $t2, $t2, $s7
lwc1 $f4, ($t2)        # distances[l][m]
add.s $f17, $f17, $f4          # temp += distances[l][m]
sll $t1, $s4, 5
addi $t2, $t1, 4
add $t2, $t2, $s7
lwc1 $f4, ($t2)        # distances[m][1]
add.s $f17, $f17, $f4          # temp += distances[m][1]
```

```
	#ending point of calculating temp
	c.lt.s $f16, $f17	# if(min<temp)
	bc1t tailM	# m++

	li $s5, 2	# n
loopN:
	# first if statement
	beq $s5, $s0, tailN
	beq $s5, $s1, tailN
	beq $s5, $s2, tailN
	beq $s5, $s3, tailN
	beq $s5, $s4, tailN
	# calculate temp
	sll $t0, $s0, 2
	addi $t2, $t0, 32
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[1][i]
	add.s $f17, $f0, $f4	# temp = 0 + distances[1][i]
	sll $t1, $s0, 5
	sll $t0, $s1, 2
	add $t2, $t1, $t0
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[i][j]
	add.s $f17, $f17, $f4		# temp += distances[i][j]
	sll $t1, $s1, 5
	sll $t0, $s2, 2
	add $t2, $t1, $t0
	add $t2, $t2, $s7
	lwc1 $f4, ($t2)		# distances[j][k]
```

```
add.s $f17, $f17, $f4		# temp += distances[j][k]
sll $t1, $s2, 5
sll $t0, $s3, 2
add $t2, $t1, $t0
add $t2, $t2, $s7
lwc1 $f4, ($t2)		# distances[k][l]
add.s $f17, $f17, $f4		# temp += distances[k][l]
sll $t1, $s3, 5
sll $t0, $s4, 2
add $t2, $t1, $t0
add $t2, $t2, $s7
lwc1 $f4, ($t2)		# distances[l][m]
add.s $f17, $f17, $f4		# temp += distances[l][m]
sll $t1, $s4, 5
sll $t0, $s5, 2
add $t2, $t1, $t0
add $t2, $t2, $s7
lwc1 $f4, ($t2)		# distances[m][n]
add.s $f17, $f17, $f4		# temp += distances[m][n]
sll $t1, $s5, 5
addi $t2, $t1, 4
add $t2, $t2, $s7
lwc1 $f4, ($t2)		# distances[n][1]
add.s $f17, $f17, $f4		# temp += distances[n][1]
#ending point of calculating temp
c.lt.s $f17, $f16		# if(temp<min)
bc1f tailN		# When false, go to outN
mov.s $f16, $f17		# inner of 'if' statement. min = temp;
la $t0, resultpath
```

```
    sw $s0, 4($t0)       # result_path[1] = i
    sw $s1, 8($t0)       # result_path[2] = j
    sw $s2, 12($t0)      # result_path[3] = k
    sw $s3, 16($t0)      # result_path[4] = l
    sw $s4, 20($t0)      # result_path[5] = m
    sw $s5, 24($t0)      # result_path[6] = n
tailN:
    addi $s5, $s5, 1     # n++
    slti $t0, $s5, 8     # if n < 8, set
    bne $t0, $zero, loopN        # go back to loopN

tailM:
    addi $s4, $s4, 1     # m++
    slti $t0, $s4, 8     # if m < 8, set
    bne $t0, $zero, loopM        # go back to loopM

tailL:
    addi $s3, $s3, 1     # l++
    slti $t0, $s3, 8     # if l < 8, set
    bne $t0, $zero, loopL        # go back to loopL

tailK:
    addi $s2, $s2, 1     # k++
    slti $t0, $s2, 8     # if k < 8, set
    bne $t0, $zero, loopK        # go back to loopK

tailJ:
    addi $s1, $s1, 1     # j++
    slti $t0, $s1, 8     # if j < 8, set
```

```
	bne $t0, $zero, loopJ		# go back to loopJ

tailI:
	addi $s0, $s0, 1	# i++
	slti $t0, $s0, 8	# if i < 8, set
	bne $t0, $zero, loopI		# go back to loopI
# Ending point of the for loops!

	# Print results part
	la $t0, resultpath
	li $v0, 4	# syscall 4 = print String
	la $a0, Path
	syscall		# Print 'Path'
	li $v0, 1	# sysccall 1 = print integer
	lw $a0, 0($t0)
	syscall
	li $v0, 4	# syscall 4 = print String
	la $a0, Space
	syscall
	li $v0, 1	# sysccall 1 = print integer
	lw $a0, 4($t0)
	syscall
	li $v0, 4	# syscall 4 = print String
	la $a0, Space
	syscall
	li $v0, 1	# sysccall 1 = print integer
	lw $a0, 8($t0)
	syscall
	li $v0, 4	# syscall 4 = print String
```

```
la $a0, Space
syscall
li $v0, 1      # sysccall 1 = print integer
lw $a0, 12($t0)
syscall
li $v0, 4      # syscall 4 = print String
la $a0, Space
syscall
li $v0, 1      # sysccall 1 = print integer
lw $a0, 16($t0)
syscall
li $v0, 4      # syscall 4 = print String
la $a0, Space
syscall
li $v0, 1      # sysccall 1 = print integer
lw $a0, 20($t0)
syscall
li $v0, 4      # syscall 4 = print String
la $a0, Space
syscall
li $v0, 1      # sysccall 1 = print integer
lw $a0, 24($t0)
syscall
li $v0, 4      # syscall 4 = print String
la $a0, Space
syscall
li $v0, 1      # sysccall 1 = print integer
lw $a0, 28($t0)
syscall
```

```
	li $v0, 4	# syscall 4 = print String
	la $a0, Enter
	syscall

	la $a0, Distance
	syscall		# Print 'Distance'
	li $v0, 2	# syscall 2 = print float
	add.s $f12, $f16, $f0
	syscall

	li $v0 10	# syscall 10 (finish the program)
	syscall		# Finish the program.

CalDistance:	# $a0=city1.x, $a1=city1.y, $a2=city2.x, $a3=city2.y
	sub $t0, $a0, $a2	# dx = city1.x - city2.x
	sub $t1, $a1, $a3	# dy = city1.y - city2.y
	mul $t0, $t0, $t0	# $t0 = dx * dx
	mul $t1, $t1, $t1	# $t1 = dy * dy
	add $t2, $t0, $t1	# $t2 = dx*dx + dy*dy
	sw $t2, 0($s5)		# at tempmemory, save $t2
	lwc1 $f4, 0($s5)	# load tempmemory's value to $f4
	cvt.s.w $f8, $f4	# convert int to float
	sqrt.s $f18, $f8	# $f18 = $f8's sqrt
	jr $ra

.data
bigFloat:
	.float 999999999.99
Path:
```

```
    .asciiz "Path : "
Distance:
    .asciiz "Distance : "
Space :
    .asciiz " "
Enter :
    .asciiz "\n"

tempmemory :
    .space 4
distances :
    .space 256
resultpath :        # result_path[8]
    .space 32
cities :   # cities[8]
    .space 64
```

2. Details of Assembly Code

Let’s see the ‘data’ section.

```
.data
bigFloat:
	.float 999999999.99
Path:
	.asciiz "Path : "
Distance:
	.asciiz "Distance : "
Space :
	.asciiz " "
Enter :
	.asciiz "\n"

tempmemory :
	.space 4
distances :
	.space 256
resultpath :	# result_path[8]
	.space 32
cities :   # cities[8]
	.space 64
```

bigFloat is for initializing ‘min’ variable.

Path, Distance, Space, Enter are for printing some strings.

Tempmemory is for storing temp value (at CalDistance function).

Distances is for the 2d array, distances[8][8].

Resultpath is for the array, result_path[8].

Cities is for the array, cities[8].

.

Next, let's see the CalDistance function.

```
CalDistance:    # $a0=city1.x, $a1=city1.y, $a2=city2.x, $a3=city2.y
    sub $t0, $a0, $a2    # dx = city1.x - city2.x
    sub $t1, $a1, $a3    # dy = city1.y - city2.y
    mul $t0, $t0, $t0    # $t0 = dx * dx
    mul $t1, $t1, $t1    # $t1 = dy * dy
    add $t2, $t0, $t1    # $t2 = dx*dx + dy*dy
    sw $t2, 0($s5)       # at tempmemory, save $t2
    lwc1 $f4, 0($s5)     # load tempmemory's value to $f4
    cvt.s.w $f8, $f4     # convert int to float
    sqrt.s $f18, $f8     # $f18 = $f8's sqrt
    jr $ra
```

First two sub instructions are same with

float dx = city1.x – city2.x;

float dy = city1.y – city2.y;

And next 8 lines are same with

Return sqrt(dx*dx + dy*dy);

2 mul instructions calculate dx and dy's square values. And add them. Sw, lwc1, cvt.s.w instructions are for converting from integer to FP. sqrt.s instruction calculates the number's root value.

Let's see the 'main' part.

i in $s0, j in $s1, k in $s2, l in $s3, m in $s4, n in $s5, cities' address in $s6, distances' address in $s7, min in $f16, temp in $f17. We use temp and temptemp as same($f17).

At below code lines,

#store cities' coordinates.

```
la $s6, cities          # store cities' address at $s6
li $t1, -1
sw $t1, 0($s6)
sw $t1, 4($s6)          # store {-1,-1}
li $t1, 0
sw $t1, 8($s6)
sw $t1, 12($s6)         # store {0,0}
li $t1, 2
sw $t1, 16($s6)
li $t1, 6
sw $t1, 20($s6)         # store {2,6}
li $t1, 8
sw $t1, 24($s6)
li $t1, 4
sw $t1, 28($s6)         # store {8,4}
li $t1, 7
sw $t1, 32($s6)
li $t1, 2
sw $t1, 36($s6)         # store {7,2}
li $t1, 1
sw $t1, 40($s6)
li $t1, 6
sw $t1, 44($s6)         # store {1,6}
li $t1, 4
sw $t1, 48($s6)
li $t1, 9
sw $t1, 52($s6)         # store {4,9}
```

```
    li $t1, 3
    sw $t1, 56($s6)
    li $t1, 2
    sw $t1, 60($s6)        # store {3,2}
```

We store cities’ coordinates.

And at below code lines,

```
# Starting point of 'for' loops for distances[8][8]!
    la $s7, distances
    li $s0, 1       # i
L1:

    li $s1, 1       # j
L2:
    sll $t1, $s0, 3         # $t1 = i*8
    sll $t2, $s1, 3         # $t2 = j*8
    add $t1, $s6, $t1       # store cities[i]'s address
    add $t2, $s6, $t2       # store cities[j]'s address
    lw $a0, 0($t1)
    lw $a1, 4($t1)
    lw $a2, 0($t2)
    lw $a3, 4($t2)
    # $a0=city1.x, $a1=city1.y, $a2=city2.x, $a3=city2.y
    jal CalDistance         #call CalDistance
    sll $t1, $s0, 5         # $t1 = i*32
    sll $t2, $s1, 2         # $t2 = j*4
    add $t3, $t1, $t2
    add $t3, $t3, $s7       # $t3 stores distances[i][j]'s address
    swc1 $f18, 0($t3)
```

```
T2:
    addi $s1, $s1, 1       # j++
    slti $t0, $s1, 8       # if j < 8, set
    bne $t0, $zero, L2     # go back to L2

T1:
    addi $s0, $s0, 1       # i++
    slti $t0, $s0, 8       # if i < 8, set
    bne $t0, $zero, L1     # go back to L1
# Ending point of 'for' loops for distances[8][8]!
```

We store distances between each set of 2 cities.

Calculate distances by calling CalDistance function.

The calculated value by CalDistance is stored at $f18.

```
        sll $t1, $s0, 5        # $t1 = i*32
        sll $t2, $s1, 2        # $t2 = j*4
        add $t3, $t1, $t2
        add $t3, $t3, $s7      # $t3 stores distances[i][j]'s address
        swc1 $f18, 0($t3)
```

When storing values at certain address at distances, we stored the certain address at $t3.

```
    lwc1 $f16, bigFloat  # min = 999999999.99
    la $t1, resultpath
    li $t0, 1
    sw $t0, 0($t1)       # result_path[0] = 1
    sw $t0, 28($t1) # result_path[7] = 1
```

This part is same with

```
    min = 1.7e+307;
    result_path[0] = result_path[7] = 1;
```

Let’s see the next part. Next part is started at # Starting point of the for loops!

and ended at # Ending point of the for loops!

This part is same with C version file’s 6 times overlapped for loop part.

Each for loop is composed with 3 part, set initial value, ‘loop’ part, and ‘tail’ part. Initial value is for setting initial value. ‘loop’ part is the inner part of for loop. ‘tail’ part is for increasing and checking the index value.

If you want the more detail about the overlapped for loops, see the comments. We wrote many comments very carefully.

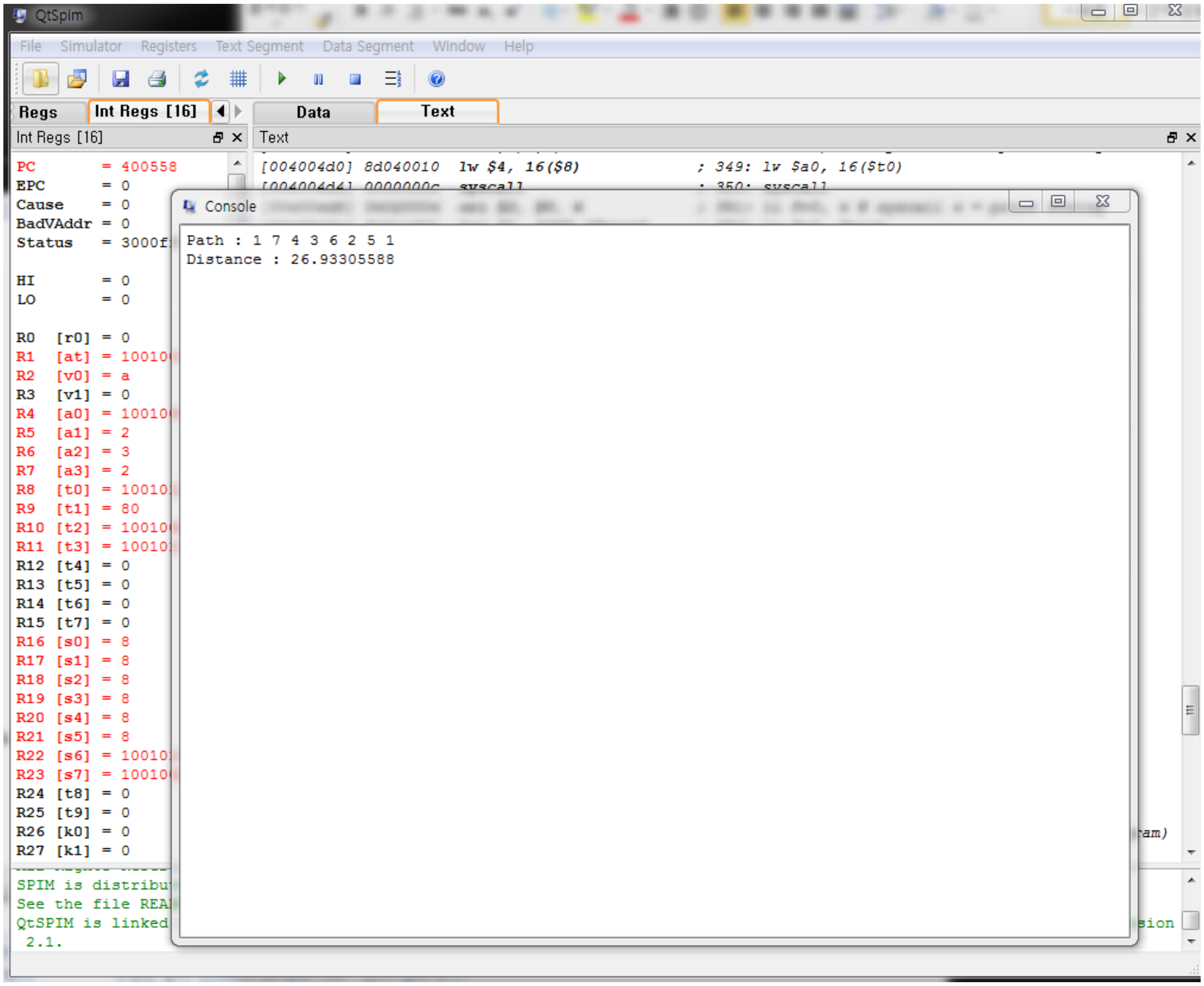
And finally, print the result.

Like the comments,

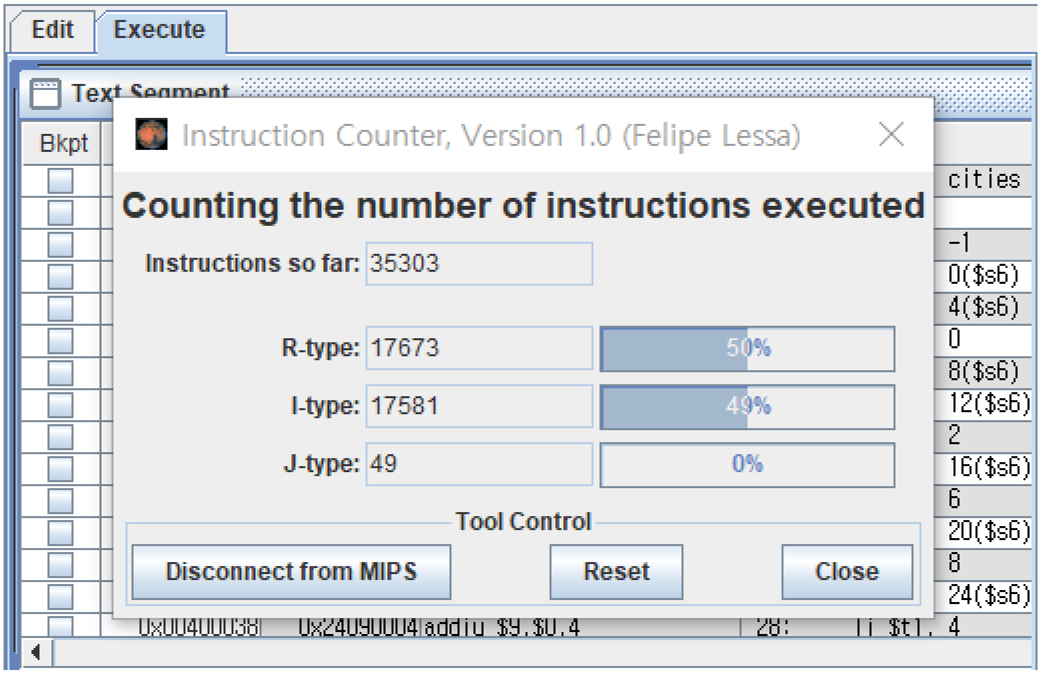
```
li $v0, 4     # syscall 4 = print String
li $v0, 1     # syscall 4 = print integer
li $v0, 2     # syscall 2 = print float
li $v0, 10    # syscall 10 (finish the program).
```

By these, we print the result.

3. Output


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Regs
Int Regs [16]
Data
Text
[004004d0] 8d040010 lw $4, 16($8) ; 349: lw $a0, 16($t0)
Console
Path : 1 7 4 3 6 2 5 1
Distance : 26.93305588
PC = 400558
EPC = 0
Cause = 0
BadVAddr = 0
HI = 0
LO = 0
R0 [r0] = 0
R2 [v0] = a
R3 [v1] = 0
R5 [a1] = 2
R6 [a2] = 3
R7 [a3] = 2
R9 [t1] = 80
R16 [s0] = 8
SPIM is distribu
See the file REA
QtSPIM is linked

4. The Estimated Execution Time



Device name DESKTOP-B7TLKUA

Processor Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz

-InstructionCount : 35,303

-CPI : 1

-ClockRate : 2.5GHz

Execution time = InstructionCount X Cycles per Instruction

= InstructionCount X CPI / ClockRate

So, Excution time = 35,303 X 1 / (2.5 X 10^9) = 1.41212 X 10^(-5) sec