# Abstract

In this phase, we manually compile the C program made in phase1 into MIPS assembly. This report introduces the MIPS assembly code compiled and the estimated execution time in terms of instruction count.

# Assembly Code

# SWE3005 Computer Architectures

# Prof. Hee Yong Youn

# 2019 Spring Course Project

# TSP on MIPS

# Phase 2: Manually compile into MIPS

# Team. 20: Hyungjun Kim, Ju-eun Park

.data

cities:   # struct city\_node num, x, y

  .word 1

  .word 0

  .word 0

  .word 2

  .word 8

  .word 6

  .word 3

  .word 2

  .word 4

  .word 4

  .word 6

  .word 7

  .word 5

  .word 1

  .word 3

  .word 6

  .word 9

  .word 4

  .word 7

  .word 2

  .word 3

ans:

  .double 1000000.0

arr:  # array for distance

  .double 0.000000  # row 1

  .double 10.000000

  .double 4.472136

  .double 9.219544

  .double 3.162278

  .double 9.848858

  .double 3.605551

  .double 10.000000 # row 2

  .double 0.000000

  .double 6.324555

  .double 2.236068

  .double 7.615773

  .double 2.236068

  .double 6.708204

  .double 4.472136  # row 3

  .double 6.324555

  .double 0.000000

  .double 5.000000

  .double 1.414214

  .double 7.000000

  .double 1.000000

  .double 9.219544  # row 4

  .double 2.236068

  .double 5.000000

  .double 0.000000

  .double 6.403124

  .double 4.242641

  .double 5.656854

  .double 3.162278  # row 5

  .double 7.615773

  .double 1.414214

  .double 6.403124

  .double 0.000000

  .double 8.062258

  .double 1.000000

  .double 9.848858  # row 6

  .double 2.236068

  .double 7.000000

  .double 4.242641

  .double 8.062258

  .double 0.000000

  .double 7.071068

  .double 3.605551  # row 7

  .double 6.708204

  .double 1.000000

  .double 5.656854

  .double 1.000000

  .double 7.071068

  .double 0.000000

visit:      .space 28 # int visit[7];

shortest\_path:  .space 28 # int shortest\_path[7];

current\_path: .space 28 # int current\_path[7];

space:      .asciiz " "

newline:    .asciiz "\n"

.text

main:

  li    $t0, 1

  la    $t1, shortest\_path

  sw    $t0, 0($t1)     # shortest\_path[0] = 1

  la    $t2, current\_path

  sw    $t0, 0($t2)     # current\_path[0] = 1

  li    $a0, 0

  li    $a1, 0

  mfc1  $zero, $f14

  jal   dfs   # call dfs

  nop

  ldc1  $f12, ans

  li    $v0, 3

  syscall

  nop

  la    $a0, newline

  li    $v0, 4

  syscall

  nop

  jal   print\_path

  nop

  li    $v0, 10   # terminate program

syscall

print\_path:

  la    $s1, shortest\_path

  li    $t3, 0    # i = 0

  L1:

    beq   $t3, 7, print\_path\_dfs\_end  # if i >= 7 then print\_path\_dfs\_end

    sll   $t4, $t3, 2   # i \* 4 (offset)

    add   $t4, $s1, $t4 # arr[i]

    lw    $a0, 0($t4)   # $a0 = arr[i]

    li    $v0, 1      # print integer

    syscall

    nop

    la    $a0, space    # print space

    li    $v0, 4

    syscall

    nop

    addiu $t3, $t3, 1   # i++

    j   L1        # branch to L1

    nop

  print\_path\_dfs\_end:

    la    $a0, newline  # print newline (not really useful for here)

    li    $v0, 4

    syscall

    nop

    jr    $ra       # jump to $ra

    nop

save\_path:

  li    $t3, 0  # i

  L3:

    bge   $t3, 7, save\_path\_end

    sll   $t4, $t3, 2

    la    $s3, current\_path # $s3 = cur

    add   $s3, $s3, $t4   # current\_path[i]

    lw    $s1, 0($s3)     # $s1 = current\_path[i]

    la    $s3, shortest\_path

    add   $s3, $s3, $t4

    sw    $s1, 0($s3)

    addiu $t3, $t3, 1   # i++

    b   L3        # branch to L3

    nop

  save\_path\_end:

    jr    $ra

dfs: # $a0 - n, $a1 - depth, $f14 - sum, $s4 - i

  beq   $a1, 6, dfs\_end # if depth == 6 then end

  nop

  addi  $sp, $sp, -48

  sw    $ra, 40($sp)

  sw    $a0, 36($sp)

  sw    $a1, 24($sp)

  s.d   $f14, 16($sp) # 8 byte double

  sw    $s7, 8($sp)   # &visit[i] of caller

  sw    $s4, 4($sp)   # i index

  li    $s4, 0  # $s4 is i index

  L2: # for loop

    addi  $s4, $s4, 1     # ++i

    bgt   $s4, 6, dfs\_end   # if i > 6 then end recursive call

    nop

    sll   $t3, $s4, 2     # index processing

    la    $s1, visit      # load visit address

    add   $s7, $t3, $s1   # $s7 = address of visit[i]

    lw    $t5, 0($s7)     # $t5 = visit[i]

    beq   $t5, 1, L2      # if visit[i] == 1 continue;

    nop

    la    $t0, arr      # $t0 = &arr

    mul   $t1, $a0, 7     # col processing; $t1 = n \* 7

    add   $t1, $t1, $s4   # row proceesing; $ti = n \* 7 + i

    mul   $t1, $t1, 8     # address processing; size of double

    add   $t0, $t0, $t1   # $t0 = &arr[n][i]

    l.d   $f4, 0($t0)     # $f4 = arr[n][i]

    add.d $f0, $f4, $f14    # $f0 = sum + arr[n][i]

    ldc1  $f2, ans      # $f2 = ans

    c.lt.d  $f2, $f0      # if sum + arr[n][i] > ans then L2 (inverse condition)

    bc1t  L2

    nop

    li    $t5, 1        # $t5 = 1

    sw    $t5, 0($s7)     # visit[i] = 1

    la    $s1, cities     # $s1 = &cities[0]

    mul   $s3, $s4, 12    # i \* 12 (size of struct)

    add   $s1, $s1, $s3   # $s1 = &cities[i].num

    lw    $t6, 0($s1)     # $t6 = cities[i].num

    # addi  $t6, $s4, 1     # city num (same thing as upper)

    addi  $t7, $a1, 1     # depth+1

    mul   $t8, $t7, 4     # [depth+1]

    la    $s2, current\_path

    add   $s2, $t8, $s2   # $s2 = current\_path[depth+1]

    sw    $t6, 0($s2)     # current\_path[i] = cities[i].num

    # save next argument

    move  $a0, $s4      # n = i

    move  $a1, $t7      # depth = depth+1

    mfc1 $zero, $f6      # $f6 = 0.0

    add.d $f14, $f0, $f6    # move $f0(sum+arr[n][i]) to $f14

    jal   dfs   # recursive call

    nop

    sw    $zero, 0($s7)   # visit[i] = 0

    j   L2          # jump to L2

  dfs\_end:

    la    $t0, arr

    mul   $t1, $a0, 7

    mul   $t1, $t1, 8

    add   $t0, $t0, $t1   # $t0 = &arr[n][0]

    ldc1  $f4, 0($t0)     # $f4 = arr[n][0]

    add.d $f14, $f14, $f4   # sum += arr[n][0]

    la    $t9, ans      # $t9 = &ans

    l.d   $f6, 0($t9)     # $f6 = ans

    c.lt.d  $f14, $f6     # if sum < ans

    bc1t  save        # then goto save

    nop

    lw    $s4, 4($sp)

    lw    $s7, 8($sp)

    l.d   $f14, 16($sp)

    lw    $a1, 24($sp)

    lw    $a0, 36($sp)

    lw    $ra, 40($sp)

    addi  $sp, $sp, 48

    jr    $ra

  save:

    s.d   $f14, 0($t9)    # ans = sum

    jal   save\_path

    nop

    lw    $s4, 4($sp)

    lw    $s7, 8($sp)

    l.d   $f14, 16($sp)

    lw    $a1, 24($sp)

    lw    $a0, 36($sp)

    lw    $ra, 40($sp)

    addi  $sp, $sp, 48

    jr    $ra

# Execution Time Estimate at Instruction Count

We assumeed the worst case (O(n^2)) and calculated the instruction count.

At main: we have 18 instructions except nop instruction. We always didn’t count nop instruction.

At print\_path: we have 2 + 11 \* 7 + 4= 83 instructions.

At save\_path: we have 1 + 10 \* 7 + 1 = 72 instructions.

At dfs : we have 9 + 35 \* 7 + 28 + save\_path(72) = 354 instructions.

Total : 455 instructions (save\_path is included in dfs).