Project

- Phase3 -

(Computer Architecture)

Team A

2013311659 곽창근

2013312608 문성암

-Index-

1. **Predicted execution time**
2. **Simulation output**
3. **Comparing the result**

Phase #3

1. Predicted execution time

We added a hand-calculating part to Phase #3. Because we calculated without hand-calculating using the program to calculate the instruction count.

Let’s see the ‘main’ part.

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}
```

At here, we store cities' coordinates.

The number of instructions is 31.

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]!

At here, we store distances between each set of 2 cities.

In C code,

```
        double distances[8][8]; // Calculate distance between cities and save.
        for (i = 1; i <= 7; i++)
                for (j = 1; j <= 7; j++)
                        distances[i][j] = CalDistance(cities[i], cities[j]);
```

this part.

Inner loop is repeated 7*7=49 times. The MIPS code is also same. L2 and T2 parts are repeated 49 times. So 17*49=833 instructions executed. And L1, T1 parts are repeated 7 times. So 4*7=28 instructions executed.

At L2 part, you can see ‘jal CalDistance’ this sentence. We call ‘CalDistance’ 49 times. The number of instructions in CalDistance is 10. So, 10*49=490 instructions executed.

Total number of executed instructions in this part is 833+28+490=1351.

Next part is below.

```
        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
```

5 instructions.

Next part is 6 times overlapped ‘for’ loop.

In C code, every loop except for the beginning and the end,

```
        if (k == i || k == j )
            continue;
        temptemp = distances[1][i] + distances[i][j] + distances[j][k] + distances[k][1];
        if ( temptemp > min )
            continue;
```

These form of code-line is written. This code-line was inserted to speed execution, but this makes it difficult to accurately predict when we want to measure the instruction count in MIPS version code. So, " Count ++ ; " is inserted between the code-line to measure exactly how many times this part was executed. For example, to measure how many times a overlapped ' for' loop is entered into ' temp' in the last part (The last part of overapped ‘for’ loop is ‘for (n = 2; n <= 7; n++) {…} ‘.)

```
    for (n = 2; n <= 7; n++) {
        if (n == i || n == j || n == k || n == l || n == m)
            continue;
        count++;
        temp = distances[1][i] + distances[i][j] + distances[j][k] + distances[k][l]
            + distances[l][m] + distances[m][n] + distances[n][1];
```

Thus “count ++” was added and executed. Then we checked the count value and measured the number of executed.

Now let's measure the instruction count on MIPS code one by one.

1)The code that enters the temp at loopN was executed 58 times,

the code that enters the temp at loopM was executed 242 times,

the code that enters the temp at loopL was executed 276 times,

the code that enters the temp at loopK was executed 120 times,

and the code that enters the temp at loopJ was executed 30 times.

Therefore, the total instruction count for these parts is 42 * 58 + 36 * 242+30 + 246+120+18 = 22848.

2) If the internal code of ‘ if (j) {…} ’ does not proceed in loopJ, it is the same as in " 1) ".

The same goes for loopK, loopl, loopM, loopN.

Therefore, the total instruction count for these parts is 30*1+120*2+276*3+242*4+58*5 = 2356.

3) If the internal code for ' if (j) {…} ' is processed in loopJ, this is executed 6 times,

internal code for ' if(k==l || k==j){…} ' is processed in loopK, this is executed 60 times,

it is executed 287 times in loopL,

it is executed 484 times in loopM,

and it is executed 290 times in loopN.

It is difficult to measure the exact instruction count for this part, therefore, we calculated the average value. Total instruction count for these parts is 6*1+60 * 1.5 + 287 * 2 + 484 *2.5 290*3=2690.

4) In C code, the part of 'if(min>temp) {…}' is executed 6 times. In other words,

```
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
```

These codes were repeated 6 times, The total instruction count of this part is 6*8=48.

5) "li $s0, 2   # I", the beginning of " loopI : " was executed once.

"li $s1, 2   # I", the beginning of " loopJ : " was executed 6 times.

"li $s2, 2   # I", the beginning of " loopK : " was executed 30 times.

"li $s3, 2   # I", the beginning of " loopL : " was executed 92 times.

"li $s4, 2   # I", the beginning of " loopM : " was executed 121 times.

"li $s5, 2   # I", the beginning of " loopN : " was executed 58 times.

The total instruction count of this part is 1+6+30+92+121+58=308.

6) See how many times the "tailX:" is repeated.

```
for (i = 2; i <= 7; count++, i++)
```

As shown in the above code, 'count ++' was added within the third parameter of for statement, and we checked the instruction count with this.

And the number of times that have been executed is as follows :

tailI was executed 6 times.

tailJ was executed 36 times.

taiK was executed 180 times.

tailL was executed 552 times.

tailM was executed 726 times.

tailN was executed 348 times.

Because each tailx has 3 instructions, the total instruction count of this part is (6+36+180+552+726+348)*3=5544.

We calculated 6 times overlapped ‘for’ loop part as above divided into 6 parts.

So, the total instruction count of 6 times overlapped ‘for’ loop part is 22848+2356+2690+48+308+5544=33794.

Last part is,

```
# 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)
```

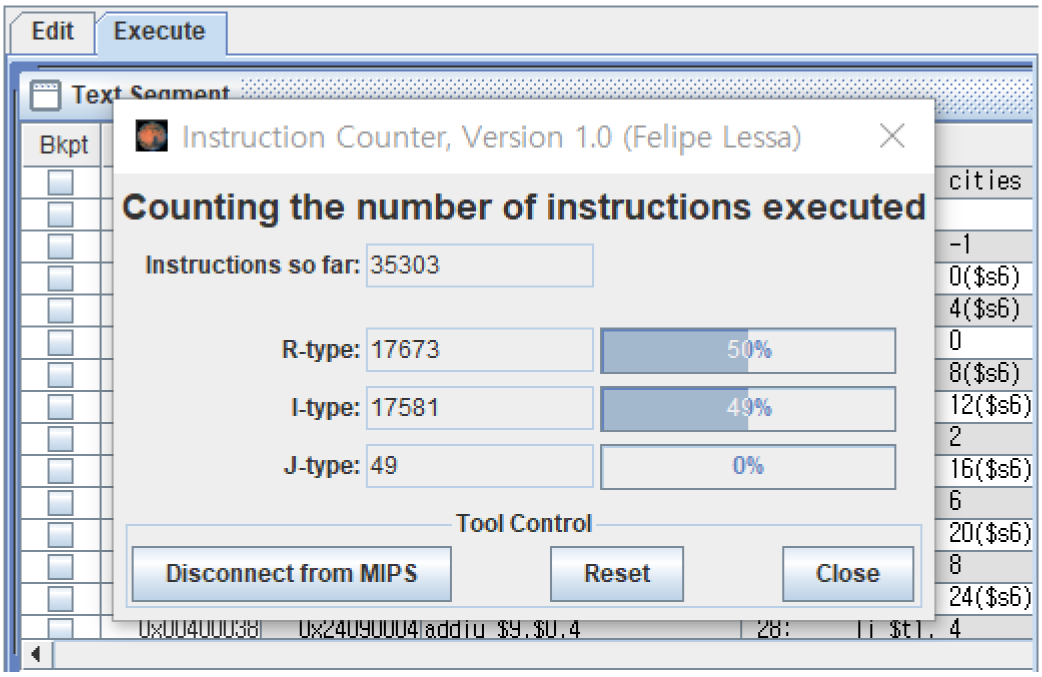
```
    syscall
    li $v0, 4       # syscall 4 = print String
    la $a0, Enter
    syscall

    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.
```

the same as the above.

So,the instruction count of last part is 59 instructions.

Finally, the total instruction count is 31+1351+5+33794+59 = 35240.

2. Simulation output



Instruction Counter, Version 1.0 (Felipe Lessa)

Counting the number of instructions executed

Instructions so far: 35303

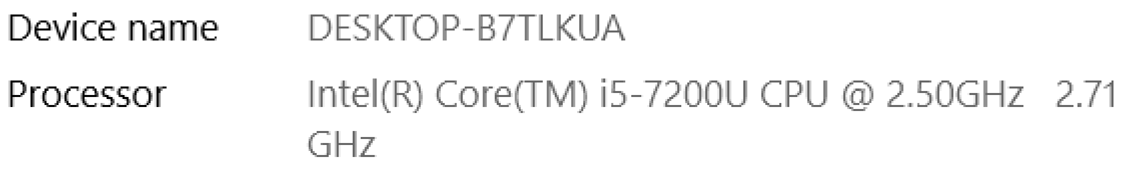
R-type: 17673 50%

I-type: 17581 49%

J-type: 49 0%

Tool Control

Disconnect from MIPS | Reset | Close



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

3. Comparing the result

We predicted the total instruction count at 35240.

CPI=1, ClockRate=2.5GHz

So, we predicted Execution time=35,240 X 1 / (2.5 X 10^9) = 1.4096 Xt 10^(-5) sec.

Erorr rate = (1.41212 X 10^(-5) - 1.4096 Xt 10^(-5)) / 1.41212 X 10^(-5)

= 0.00178455088802651332747925105515 ≒ 0.178455%

The main reason for this error is that it is calculated as the average value from the third part among 6 times overlapped ‘for’ loop part.