1. (a)

addi **$**10, **$**0, 128 # $10 = 128

div **$**12, **$**10 # $12 / 128, no shifts due to sign bit

mflo **$**14 # store result (quotient) in $14

(b)

addi **$**10, **$**0, 16 # $10 = 16

mtc1.d **$**10, **$**f4 # $f4-$f5 = 16

mul.d **$**f10, **$**f12, **$**f4 # $f10-$f11 = $f12-$f13 \* 16



# initialisation

add **$**t0, **$**0, **$**11 # $t0 current A element, initially base of A

add **$**t1, **$**0, **$**12 # $t1 current B element, initially base of B

addi **$**t2, **$**0, 0 # initialise loop counter

addi **$**t3, **$**0, 1000 # number of iterations

Loop**:** # copy current A element (little endian)

# to current B element (big endian)

# Endianness only affects the storage of elements (byte order) and not

# the array order. This is where the difference will be taken care of.

lbu **$**t4, 0**($**t1**)** # load first byte from B

sb **$**t4, 3**($**t0**)** # first byte of B = last byte of A

lbu **$**t4, 1**($**t1**)** # load 2nd byte from B

sb **$**t4, 2**($**t0**)** # 2nd byte of B = 3rd byte of A

lbu **$**t4, 2**($**t1**)** # load 3rd byte from B

sb **$**t4, 1**($**t0**)** # 3rd byte of B = 2nd byte of A

lbu **$**t4, 3**($**t1**)** # load 4th (last) byte from B

sb **$**t4, 0**($**t0**)** # last byte of B = first byte of A

# iterate to the next word (array elements) and increment loop counter

addi **$**t0, **$**t0, 4

addi **$**t1, **$**t1, 4

addi **$**t2, **$**t2, 1

# check if finished

slt **$**t6, **$**t2, **$**t3 # if($t2<$t3){$t6 = 1} else {$t6 = 0}

bne **$**t6, **$**0, Loop # if($t6==1){goto Loop} i.e. if($t2<$t3),repeat

1. (a)

countNumIn**:** # a0 base of the array

# initialise

add **$**t0, **$**0, **$**a0 # save $a0 in $t0

addi **$**t1, **$**0, 0 # count of 'in'

addi **$**t2, **$**0, 0 # check of 'i' (i.e. 1 if prev was 'i')

lbu **$**t3, 0**($**t0**)** # load first element of array

addi **$**t4, **$**0, 'i' # to check for 'i'

addi **$**t5 **$**0, 'n' # to check for 'n'

loop**:**

beq **$**t3, **$**0, outLoop # if current element is null, goto outLoop

beq **$**t3, **$**t4, incrementCheck # if curr element is 'i', set check = 1

beq **$**t3, **$**t5, checkPrevWasI # if curr element is 'n', see if check == 1

addi **$**t2, **$**0, 0 # default (curr isn't null, i or n): check = 0

loadNext**:**

addi **$**t0, **$**t0, 1 # iterate array address

lbu **$**t3, 0**($**t0**)** # load next array element

j loop

checkPrevWasI**:**

beq **$**t2, 1, incrementCount # curr is 'n', if prev was 'i' count++

addi **$**t2, **$**0, 0 # check = 0

j loadNext

incrementCheck**:**

addi **$**t2, **$**t2, 1 # check = 1 (i.e. current char is 'i')

j loadNext

incrementCount**:**

addi **$**t1, **$**t1, 1 # count++

addi **$**t2, **$**0, 0 # check = 0

j loadNext

outLoop**:**

add **$**v0, **$**0, **$**t1 # return count, result in $v0

jr **$**ra

(b)

.data

input\_string**:** .asciiz "Shervin was in the garden in the morning.\n"

.text

.globl main

main**:**

la **$**a0, input\_string # load input

j countNumIn

When I had jal countNumIn the program wouldn’t stop because $ra would change value due to nested sub routines I think, not sure how to deal with it (saving the stack pointer maybe). The current config works though, $v0 = 4.

1. (a)

compute**:** # a0: x, a1: y, a2: n

# load args

add **$**t0, **$**0, **$**a0 # $t0 -> x

add **$**t1, **$**0, **$**a1 # $t1 -> y

add **$**t2, **$**0, **$**a2 # $t2 -> n

# check x,y are in valid range: 0<x<10, 0<n<7

addi **$**t3, **$**0, 9

addi **$**t4, **$**0, 6

addi **$**t5, **$**0, 1

# if (x < 10 && x > 0 && n < 7 && n > 0), we can continue

# using OR (demorgan): (x >= 10 || x <= 0 || n >= 7 || n <= 0)

# this way $t6 can be checked to see if any condition failed

slt **$**t6, **$**t3, **$**t0 # x >= 10 -> 10 <= x -> 9 < x

bne **$**t6, **$**0, invalidArg

slt **$**t6 **$**t0, **$**t5 # x <= 0 -> x < 1

bne **$**t6, **$**0, invalidArg

slt **$**t6, **$**t4, **$**t2 # n >= 7 -> 7 <= n -> 6 < n

bne **$**t6, **$**0, invalidArg

slt **$**t6 **$**t2, **$**t5 # n <= 0 -> n < 1

bne **$**t6, **$**0, invalidArg

# args are valid, compute z = 1 + pow(3\*x, 4) + (y / pow(2, n))

addi **$**t3, **$**0, 1 # $t3 = z = 1 (for now)

addi **$**t4, **$**0, 3 # for 3\*x

addi **$**t5, **$**0, 3 # exponent in (3x)^4, -1 for loop counter

addi **$**t6, **$**0, 2 # for 2^n

# compute (3x)^4

mult **$**t0, **$**t4 # $t0: 3\*x,

mflo **$**t0 # max (27) fits in LSB

add **$**t7, **$**0, **$**t0 # save 3\*x

pow**:**

addi **$**t5, **$**t5, **-**1

mult **$**t0, **$**t7

mflo **$**t0 # max (531441) fits in LSB

bne **$**t5, **$**0, pow

# compute y / 2^n

addi **$**t2, **$**t2, **-**1 # so we can use shift left (e.g. don't shift if n=1)

sll **$**t2, **$**t6, **$**t2 # $t2: 2^n

div **$**t1, **$**t2 # $t1: y/2^n

mflo **$**t1

# add results to z

add **$**t3, **$**t3, **$**t0 # + (3x)^4

add **$**t3, **$**t3, **$**t1 # + (y/2^n)

# return z

add **$**v0, **$**0, **$**t3

jr **$**ra

invalidArg**:**

addi **$**v0, **$**0, 0

jr **$**ra

4. (b)

.text

.globl main

main**:**

li **$**a0, 4

li **$**a1, 4096

li **$**a2, 5

j compute

Result: $v0 = 20865, same as my c code.

5.

func**:** # $a0: base of X, $a1: base of Y

# $a2: i, $a3: j, $a4: num\_rows (cannot access with $a4!)

# first: get X[i][j]

# assuming row major, 0 based and X,Y same dimensions:

# offset = (i\*num\_cols + j)\*8 (bytes), need num\_cols

# num\_cols = total\_size / num\_rows

# compute size of the arrays (treat as 1D)

add **$**t0, **$**0, **$**a0 # $t0: pointer to X

addi **$**t1, **$**0, 0 # counter for total\_size

countArray**:**

lwc1 **$**f4, 0**($**t0**)** # load current X element (a double)

cvt.s.d **$**f4, **$**f2 # convert to single (for int cvt)

mfc1 **$**t0, **$**f2 # convert to int (for check)

bne **$**t0, **$**0, endCount # check if at end of array

addi **$**t1, **$**t1, 1 # total\_size++

addi **$**t0, **$**t0, 8 # iterate to next double

j countArray

endCount**:**

# now compute num\_cols

lw **$**t0, 40**($**sp**)** # $t0: num\_rows (5th arg) (no longer points to X)

div **$**t1, **$**t0 # total\_size/num\_rows

mflo **$**t0 # $t0: num\_cols

# compute offset

mult **$**t0, **$**a2 # $t0: \*= i

mflo **$**t0 # overflow? i can't be big, memory would exceed

add **$**t0, **$**t0, **$**a3 # $t0: += j

sll **$**t0, **$**t0, 3 # $t0: \*= 8, now the offset

# access X[i][j], offset += base

add **$**t1, **$**t0, **$**a0 # $t1 points to X[i][j]

lwc1 **$**f2, 0**($**t0**)** # load, $f2 = X[i][j]

# compute: 1 - X[i][j]/8

addi **$**t1, **$**0, 8 # for ../8

mtc1.d **$**t1, **$**f4

div.d **$**f2, **$**f2, **$**f4 # $f2 = X[i][j]/8

addi **$**t1, **$**0, 1 # for 1 - ..

mtc1.d **$**t1, **$**f4

sub.d **$**f2, **$**f4, **$**f2 # $f2 = 1 - (X[i][j]/8)

# store result in Y[i][j]

add **$**t2, **$**t0, **$**a1 # $t2 points to Y[i][j]

swc1 **$**f2, 0**($**t2**)** # Y[i][j] = $f2 = 1 - (X[i][j]/8)

jr **$**ra

I didn’t test this and think I’m missing something (didn’t consider little-endian for example).