COMPSYS 304 Assignment 1
840454023, elee353

Q1.

```
#(a)
# assumed signed
sra $14, $12, 7
```

```
#(b)
#initialise 16 to be used
addi $8, $0, 16

#move to FP register
mtc1 $8, $f8

#convert to double
cvt.d.w $f8, $f8

#multiply and put to register
mul.d $f10, $f12, $f8
```

COMPSYS 304 Assignment 1
840454023, elee353

Q2.

```
# load index range
addi $t3, $0, 0
addi $t0, $0, 1000

L1:
# increment index by 4 each time
sll $t4, $t3, 2
# get index value of array A
add $t5, $t4, $11
# get index value of array B
add $t6, $t4, $12

# store the fourth byte of A in the first byte of B
# load i-th element of array A into temporary register
lb $t8, 3($t5)
# store the corresponding element of A into B
sb $t8, 0($t6)

# store the third byte of A in the second byte of B
lb $t8, 2($t5)
sb $t8, 1($6)

# store the second byte of A in the third byte of B
lb $t8, 1($t5)
sb $t8, 2($t6)

# store the first byte of A to the fourth byte of B
lb $t8, 0($t5)
sb $t8, 3($t6)

#increment index range by 1
addi $t3, t3, 1

#decrement loop counter
addi $t0, $t0, -1
#check loop counter
bne $t0, $0, L1
```

COMPSYS 304 Assignment 1
840454023, elee353

Q3.

```
#(a)
funct:
# store first argument to temporary register
add $t0, $a0, $0
#store 253 to temporary register, the max length of
array
addi $t1, $a1, 253
#initialise the result register
addi $v0, $0, 0

# store ASCII ‘i’ to temporary register
addi $t5, $0, 0x69
# store ASCII ‘n’ to temporary register
addi $t6, $0, 0x6E
# store ASCII ‘null’ to temporary register
addi $t7, $0, 0x00

L1:
# from temporary register
lb $t2, 0($t0)
# test if it is ‘i’
bne $t2, $t5, L2

# load next
lb $t3 ,1($t0)

#check t3 is null
beq $t7, $t3, L3

# test if the next character is ‘n’
bne $t3, $t6, L2
# increment result by 1.
addi $v0, $v0, 1

L2:
#increment to point to next char
addi $t0, $t0, 1
#decrement loop counter
addi $t1, $t1, -1

#check loop counter is not zero
bne $t1, $0, L1
#check t2 is not null
bne $t7, $t2, L1

L3:
jr $ra
```

```
#(b)
.data
str1:	.asciiz "Shervin was in the garden in the
morning. "

.text
.globl main

main:
# array base address should be in $a0
la $a0, str1

# array size should be in $a1
ori $a1, $0, 254

jal funct

add $a0, $0, $v0

ori $v0, $0, 1

# print the result
syscall

li $v0, 10
syscall
```

COMPSYS 304 Assignment 1
840454023, elee353

Q4.
(a)

```
funct:
#initialise the result register
addi $v0, $0, 0

# store the arguments into temporary register
# store x
add $t0, $a0, $0

addi $t1, 0

#initialise the registers
addi $t7, $0, 0 # for 3x4
addi $t8, $0, 0 # for 2^n
addi $t9, $0, 0 # for range check

#store some integers to be used
addi $t3, $0, 1
addi $t4, $0, 3

# check x range
#check x < 10
slti $t9, $a0, 10
bne $t9, $t3, Exit
# check if x > 0
blez $a0, Exit

# check if y > 0
blez $a1, Exit

# check n range
# check n < 7
slti $t9, $a2, 7
bne $t9, $t3, Exit
# check n > 0
blez $a2, Exit

#multiply to get 3x and put back to register
mult $a0, $t4
# no overflow so, HI = 0
mflo $t0

#multiply to get (3x)^2
mult $t0, $t0
# no overflow so, HI = 0
mflo $t0

#multiply to get (3x)^4
mult $t0, $t0
# no overflow so, HI = 0
mflo $t0
#shift y right by n to get y/2^n
srav $t1, $a1, $a2

# z = 1
addi $v0, $v0, 1
#z = 1 + (3x)^4
add $v0, $v0, $t0
#z = 1 + (3x)^4+y/2^n
add $v0, $v0, $t1

Exit:
jr $ra
```

COMPSYS 304 Assignment 1
840454023, elee353

Q4. b.

```
.data

.text
.globl main

main:
# load x
li $a0, 4

# load y
li $a1, 4096

# load n
li $a2, 5

jal funct

add $a0, $0, $v0
ori $v0, $0, 1

# print the result
syscall
ori $v0, $0, 10
syscall

#output was 20865
```

COMPSYS 304 Assignment 1
840454023, elee353

Q5.

```
# assume stored in column major format
calculate_element_of_Y:
# get the 5th argumentand put in in $t0. (size of row)
lw $t0, 0($sp)

#initialize FP register pairs $26, $27 with double value 1
addi $t6, $0, 1
mtc1 $t6, $f20

#convert to f26, f27 = 1.0
cvt.d.w $f26, f20

#initialize FP register pairs $28, $29 with double value 8.0
addi $t6, $0, 8
mtc1 $t6, $f20

#convert to f28, f29 = 8.0
cvt.d.w $f28, f20

# calculate the address of X[i][j]
# address of X[i][j] = base address of X + ((j * size of row) + i) * size of data
# (j * size of row)
mult $a3, $t0
# no overflow so, HI = 0
mflo $t1
# ((j * size of row) + i)
add $t1, $t1, $a2
# ((j * size of row) + i) * size of data
# size of data is 8 bytes = 8 bytes = 2 words
sll $t1, $t1, 3
# $t1 = address of X[i][j]
addu $t1, $t1, $a0

# read X[i][j] (the processor is little-endian)
# $f16,$f17 = X[i][j]
lwc1 $f16, 4($t1)
lwc1 $f17, 0($t1)

# $f16, $f17 = X[i][j] / 8
div.d $f16, $f16, $f28

# Y[i][j] = 1-(X[i][j]/8)
sub $f0 , $f26, $16

jr $ra
```