CS224

Section No.: 04

Fall 2019

Lab No. 3

Okan Şen/21202377

1) -77.125

a)

<u>Single precision:</u>

77 = 38 . 2 + 1	0.125 . 2 = 0.25
38 = 19 . 2 + 0	0.25 . 2 = 0.5
19 = 9 . 2 + 1	0.5 . 2 = 1.0
9 = 4 . 2 + 1	
4 = 2 . 2 + 0	.125 = 001
2 = 1 . 2 + 0	
1 = 0 . 2 + 1	

77 = 1001101 So, 77.125 = 1001101.001 in binary form

We convert the number into scientific notation.

1.001101001 x 2^6
Mantissa = 001101001

Sign is 1 since it is negative. Sign = 1

Exponential is 6. But this is biased.

127 + 6 = 133
In binary form 133 = 1000 0101

In single precision format the number consists of;

sign(1), exponential(8), mantissa(23)

So -77.125 is;
1100 0010 1001 1010 0100 0000 0000 0000 = $(C2\ 9A\ 40\ 00)_{16}$

<u>Double precision:</u>
We get the scientific notation of the binary number using the same method as before.
So 77.125 = 1.001101001 x 2^6

The exponent bias is 1023 in double precision conversion and the exponent is 6.
Alas, we know that 1029 = 1000 0000 101 in binary. Sign is 1 since the value is negative.

Combining these we get;

1 1000 0000 101 0011 0100 1000 0000 0…..

The hexadecimal value would be = $(C0\ 53\ 48\ 00\ 00\ 00\ 00\ 00)_{16}$

b)

Single Precision:

If the bias for single precision was 120 then only the exponential part would change so the result would be

-77.125 = 1011 1111 0001 1010 0100 0000 0000 0000 = $(DF\ 1A\ 40\ 00)_{16}$

Double Precision:

With a bias of 1020 the binary would be as follows:
1 1000 0000 010 0011 0100 1000 0000 0000 0… = $(C0\ 23\ 48\ 00\ 00\ 00\ 00\ 00)_{16}$

c) C1 A0 00 00 = 1100 0001 1010 0000 0000 0000 0000

We know that the sign consists of 1 bit and exponential part is 8 bits and the rest (23 bits) is mantissa.

So;

Sign = 1 = negative
Exponential = 1000 0011
Mantissa = 0100 0000 0000 000….0

We have to get the bias removed from the exponent part which means we must convert the binary into decimal. 1000 0011 = 131, 131 - 127 = 4.

This means that, to get the scientific notation we jumped 4 digits to the left.

1.0100000… x 2^4 = 10100.0000…

In this case, we can clearly see that we don't have a fractional part and 10100 = 20

C1 10 00 00 = -20

2)

```
# -------------------------
# Okan Sen
# Start of recursiveSummation
#
# -------------------------
la $a0, number

move $t0, $a0



jal recursiveSummation
# result comes in $v0
	move $a1, $a0
	move $t0, $v0

	la $a0, msg1
	li $v0, 4
	syscall

	move $a0, $t0
	li $v0, 1
	syscall



exit:
	li $v0, 10			# system call to exit
	syscall				# bye bye
# stop execution here by syscall

recursiveSummation:
	# Basically, stores all chars in stacks,
	# when the last string char is reached, pop all the stacks
	# convert each char into integers and add them in another stack
	# a0 = string
	# if (string.length() == 1) return Value
	#return RS(string-1)

	# save in stack
	addi $sp, $sp, -12
```

```
            sw $ra, 0($sp)
            sw $s0, 4($sp)
            sw $s1, 8($sp)

            move $s0, $a0           # point to string

            lb $t5, 0($a0)  # get byte length
            beq $t5, $zero, zeroVal          # if equals one length get the value

            addi $a0, $s0, 1

            jal recursiveSummation

             add $s1, $zero, $v0        # $s1 = SR(str - 1)

             jal returnVal

             add $v0, $v0, $s1

exitRS:
            lw   $ra, 0($sp)        # read registers from stack
        lw   $s0, 4($sp)
        lw   $s1, 8($sp)
        addi $sp, $sp, 12       # bring back stack pointer
        jr $ra

returnVal:
            lbu $t1, ($s0)
            addi $t1, $t1, -48
            add $v0, $v0, $t1
            j exitRS

zeroVal:
            j exitRS




.data
number:                 .asciiz "1204"

msg1:                   .asciiz "Result is: "
```

```
endl:                          .asciiz "\n"

# -------------------------
#
# End of recursiveSummation
#
# -------------------------
```

3)

I have included which parts should be added to test the code, without errors.

<u>Add to data part</u>

```
msg1171: .asciiz "7 - delete the nodes between a given value x \n"

msg7.1:         .asciiz "Enter the value: "
msg7.2:         .asciiz "Number of deleted nodes is: "
endl:           .asciiz "\n"
```

<u>Change if wanted(just copy paste on top of it, 7 is now 8):</u>

```
msg118:  .asciiz "8 - exit this program \n"
```

<u>COPY PASTE STARTING FROM T7</u>

```
T7:     bne $s1,7, T8  # if s1 = 7, do these things. Else go to T8


        la $a0,msg7.1 # put msg address into a0
        li $v0,4            # system call to print
        syscall


        li $v0,5# system call to read
        syscall             # in the integer

        move $a1, $v0            # put x value into a1 before the call
        move $a0, $s0            # put pointer to linked list in a0 before the call
```

```
        li $v0, 0

        jal deleteAfter_x

        move $t0, $v0

        la $a0, msg7.2
        li $v0, 4
        syscall

        move $a0, $t0           # then put its value in a0 to print it out
        li $v0,1
        syscall

        la $a0, endl
        li $v0, 4
        syscall

        la $a0, endl
        li $v0, 4
        syscall

T8:     bne $s1,8, T8no         # if s1 = 8, do these things. Else go to T8no


        la $a0,msg127           # put msg127 address into a0


        li $v0,4# system call to print


        syscall         #   out the thank you string




        li $v0,10
        # the exit syscall is 10

        syscall         # goodbye...


T8no:
```

la $a0,msg128 # put msg128 address into a0

li $v0,4# system call to print

syscall # out the msg128 string

j EnterChoice # go to the place to enter the choice

<u>COPY PASTE INTO METHODS PART</u>

```
deleteAfter_x:
	li $v0, 0
	move $t0, $a0
	lw $t1, 4($t0)

	DAnext:		beq $t1, $a1, DAfirst1		# if the input value is seen get to the first label

		lw $t0, 0($t0)	# goes to next node
		beq $t0, $zero, DAdone		# we might reach the end of the list

		lw $t1, 4($t0)	# loads the value in t1

		j DAnext

	DAfirst1:
		lw $t2, 0($t0)	# get next in t2
		beq $t2, $zero, DAdone		# we might reach the end of the list
		lw $t1, 4($t2)	# get current value
	DAfirst2:


		beq $t1, $a1, DAokay# if we see the second occurence of the input value, we can proceed with the process

		lw $t2, 0($t2)		# get to next
```

```
		beq $t2, $zero, DAdone		# there might not be a second occurence and if we reach the end of the list finish method

		lw $t1, 4($t2)		# get current value
		j DAfirst2



	DAokay:
		lw $t2, 0($t0)	# get next in t2
		lw $t1, 4($t2)	# get next value

		beq $t1, $a1, DALdone		# if we still don't see the second occurence of X, we will continue deleting nodes until we see it

		lw $t3, 0($t2)	# next's next is t3
		sw $t3, 0($t0)	# set current's next t3

		li $t2, 0
		addi $v0, $v0, 1

		j DAokay


	DALdone:
		move $t0, $t2	# t0 now points to next
		lw $t0, 0($t0)	# and now t0 points to next's next for another run in case we have more occurences of X
		beq $t0, $zero, DAdone		# we might reach the end of the list
		lw $t1, 4($t0)

		j DAnext

	DAdone:
		li $t0, 0
		li $t1, 0
		li $t2, 0
		li $t3, 0

		jr $ra
```

I was not able to get the deleted node back to the heap. In other words, I can only make the nodes to point to necessary nodes after deletion but I can never delete a node. They actually stay in the heap in random memories, still having unnecessary data. In the long run, the program would have moderate memory leaks. The problem here is; there is no proper way to delete the node from the heap. Otherwise, the program works perfectly as intended. Deleting all the nodes between given X value, and with multiple occurences; such as 2, 7, 4, 8, 4, 9, 4, 10, 15, 4, 7 will return when X is 4; 2, 7, 4, 4, 9, 4, 4, 7.