Francesco Zhang

301133976

CMPT 295 Assignment 5 (2%)

1. [7 marks] Floating-Point Integers Let S be the set of all 32-bit IEEE floating-point values that are integers. Answer the following questions about S.

(a) [1 mark] How many elements are in S?

Answer: There are 3 elements in S.

(b) [1 mark] What's the largest odd value of S?

Answer: It's 16,777,215.

(c) [1 mark] What are the largest two values of S? Write your answers as exact expressions.

Answer: They're $(2-2^{-23})*2^{127}$ and $(2-2^{-22})*2^{127}$.

(d) [1 mark] How many consecutive integers are in S? You shouldn't count both 0 and −0 in your total.

Answer: It's $2^{25}+1$.

(e) [1 mark] The number 264 is in S. What are its nearest neighbours? In other words, what two values in S are closest to 264?

Answer: They're 263 and 265.

(f) [2 marks] Determine the number of values in S which are representable using 32-bit unsigned.

Answer: It's $1+2*2^{24}*(127-24+1)$.

2. [2 marks] Floating-Point Addition Add the following pairs of 32-bit IEEE floating-point numbers. Show all your work.

• 0x43938000 + 0xc2280000.

Answer:

$X=0x43938000=0\ 10000111\ 00100111000000000000000_2$

$Y=0xc2280000=1\ 10000100\ 01010000000000000000000_2$

$X_e=10000111_2-127_{10}=135-127=8$

$Y_e=10000100_2-127_{10}=132-127=5$

- ⇨ $X=1.00100111_2*2^8{}_{10}$
- ⇨ $Y=-1.0101_2*2^5=-0.0010101_2*2^8{}_{10}$
- ⇨ $X+Y=2^8{}_{10}*(1.00100111_2-0.0010101_2)=2^8{}_{10}*0.11111101_2=2^7{}_{10}*1.1111101_2$
- ⇨ $=0\ 10000110\ 11111010000000000000000_2=0x437d0000$

• 0x3f2aaaaa + 0x3eaaaaab.

Answer:

$X=0x3f2aaaaa=0\ 01111110\ 01010101010101010101010_2$

$Y=0x3eaaaaab=0\ 01111101\ 01010101010101010101011_2$

$X_e=01111110_2-127_{10}=126-127=-1$

$Y_e=01111101_2-127_{10}=125-127=-2$

- ⇨ $X=1.01010101010101010101010_2*2^{-1}{}_{10}$
- ⇨ $Y=1.01010101010101010101011_2*2^{-2}=0.101010101010101010101011_2*2^{-1}{}_{10}$
- ⇨ $X+Y=2^{-1}{}_{10}*(1.01010101010101010101010_2-0.101010101010101010101011_2)$
- ⇨ $=2^{-1}{}_{10}*0.101010101010101010101001_2=2^{-2}{}_{10}*1.01010101010101010101001_2$
- ⇨ $=0\ 01111101\ 01010101010101010101001=0x3eaaaaa9$

3. [11 marks] Adding Positive Floating-Point Numbers

(c) [7 marks]

sum_float.s:

```
	.globl sum_float
sum_float:
	push	%rbp
	leaq	(%rdi, %rsi, 4), %rbp
	# endptr <- F + n
	xorps	%xmm1, %xmm1
	xorps	%xmm2, %xmm2
	#clear xmm1 and xmm2. xmm1 is x. xmm2 is y.
	leaq	-8(%rsp), %rdx
	#create a queue and let %rdx be the tail pointer of the queue
	movq	%rdx, %rbx
	#let %rbx be the head pointer of the queue
	#now, %rdi and %rbp is the head and tail pointers of queue F to sum
	#%rbx and %rdx is the head and tail pointers of queue Q to store sums
	jmp		loop
	#start the loop
loop:
	cmpq	%rdi, %rbp
	jle		el
	#when F has all been dequeued, go to el for advanced check
	cmpq	%rbx, %rdx
	#if F is not empty, check whether Q is empty
	jne		compare1
	#when two queues are not empty, go to compare the head of two queues
	jmp		compare1l
	#if only Q is empty, go to dequeue F
el:
	movq	%rbx, %rcx
	subq	$8, %rcx
	cmpq	%rcx, %rdx
	je		endloop
	#if F is empty and Q has only 1 element, go to end the loop
	jmp		compare1g
	#if Q has more than 1 element, dequeue Q
compare1:
	xorps	%xmm0, %xmm0
	xorps	%xmm3, %xmm3
	movss	(%rdi), %xmm0
	movss	-8(%rbx), %xmm3
	ucomiss		%xmm0, %xmm3
	#use xmm0 and xmm3 to temparily store the value of heads of two queues for comparison
	jae		compare1l
	jmp		compare1g
	#dequeue the smaller one
compare1g:
	subq	$8, %rbx
	movss	(%rbx), %xmm1
	jmp second
	#Assign head of Q to x and dequeue Q and go to do the second dequeue
compare1l:
	addss	(%rdi), %xmm1
	add		$4, %rdi
	jmp		second
	#Assign head of F to x and dequeue F and go to do the second dequeue
```

second:

```
        cmpq    %rdi, %rbp
        jle             compare2g
        #if F is empty, dequeue Q
        cmpq    %rbx, %rdx
        jne             compare2
        #when two queues are not empty, go to compare the head of two queues
        jmp             compare2l
        #if only Q is empty, go to dequeue F
compare2:
        xorps   %xmm0, %xmm0
        xorps   %xmm3, %xmm3
        movss   (%rdi), %xmm0
        movss   -8(%rbx), %xmm3
        #use xmm0 and xmm3 to temparily store the value of heads of two queues for comparison
        ucomiss         %xmm0, %xmm3
        jae             compare2l
        jmp             compare2g
        #dequeue the smaller one
compare2g:
        subq    $8, %rbx
        movss   (%rbx), %xmm2
        jmp adding
        #Assign head of Q to y and dequeue Q and go to sum x and y
compare2l:
        addss   (%rdi), %xmm2
        add             $4, %rdi
        jmp             adding
        #Assign head of F to y and dequeue F and go to sum x and y
adding:
        addss   %xmm1, %xmm2
        subq    $8, %rdx
        movss   %xmm2, (%rdx)
        #adding x and y and enqueue to Q
        xorps   %xmm1, %xmm1
        xorps   %xmm2, %xmm2
        jmp     loop
        #clear x and y and go to loop
endloop:
        movss   -8(%rbx), %xmm0
        #let %xmm0 to store the final result at the head of Q
        pop     %rbp
        ret
        #return the answer and finish the function
```

(d) [2 BONUS marks] Hardcopy: Produce a [mathematical] justification/proof as to why the elements of Q must certainly be in increasing order.

Answer:

In floating point addition like in question 2, we find out that we need to convert the two values to add in the same exponent base then shift the fraction elements. The fraction part has 23 bits in IEEE 32-bit floating point system which means the smallest fraction is 2^{-23}. Therefore, if the exponent bases of two values are the same, the fraction part after shifting and addition less than 2^{-23} will be cut off. If a number is less than $1/2^{-23}$ of the number to be added, addition has no effects on the number to be added even if we take infinity times of this addition. However, if we take adding in the increasing order, we can sum the little numbers together first to let their result be greater than $1/2^{-23}$ of the number to be added. Then the addition of this result and the big number will have effect on the final result.