NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MID-TERM TEST
AY2018/19 Semester 2

CS2100 — COMPUTER ORGANISATION

13 March 2019 Time Allowed: **1 hour 45 minutes**

INSTRUCTIONS

1. This question paper contains **THIRTEEN (13)** questions and comprises **EIGHT (8)** printed pages.
2. Page 7 contains the **MIPS Reference Data** sheet.
3. Page 8 contains reference tables for **ASCII** and **Powers of Two**.
4. An **Answer Sheet**, comprising **TWO (2)** printed pages, is provided for you.
5. Write your **Student Number** and **Tutorial Group Number** on the Answer Sheet with a **PEN**.
6. Answer **ALL** questions within the space provided on the Answer Sheet.
7. You may write your answers in pencil (at least 2B).
8. You must write legibly or marks may be deducted.
9. Submit only the Answer Sheet at the end of the test. You may keep the question paper.
10. This is a **CLOSED BOOK** test. However, an A4 single-sheet double-sided reference sheet is allowed.
11. Maximum score of this test is **40 marks**.
12. Calculators and computing devices such as laptops and PDAs are <u>not allowed</u>.

——— END OF INSTRUCTIONS ———

Questions 1 – 6: Each multiple-choice-question has only one correct answer. Write your answers in the boxes on the **Answer Sheet**. Two (2) marks are awarded for each correct answer and no penalty for wrong answer. **[Total: 12 marks]** 8/12

1. You are told that $(\mathbf{32})_b$ is a product of two prime numbers (*i.e.*, $(32)_b = P_1 \times P_2$ *where* P_1 *and* P_2 *are primes*). What is/are the value of base $\boldsymbol{b}$?

   (i) 8 (ii) 9 (iii) 11

   A. (i) only
   B. (i) and (ii) only
   C. (i) and (iii) only
   D. (i), (ii), and (iii)
   E. None of the above

2. What is the content of **$t2** after executing the following MIPS code?

```
lui   $t0,   0xAAAA
srl   $t0,   $t0   ,  16
lui   $t0,   0xA0A0
ori   $t1,   $zero, 0x5555
and   $t2,   $t1   ,  $t0
```

   A. 0x00000000
   B. ~~0x0000FFFF~~
   C. 0xA0A00000
   D. 0xA0A05555
   E. None of the above.

For questions 3 – 4:

You are designing a machine with 6 registers and 64 addresses. You are in the process of creating two (2) classes of 16-bit instructions. The first is instruction class A that has 3 registers. The second is instruction class B that has 1 address and 2 registers. Both instructions exist and the encoding space is completely utilised.

2^3 2^6

A: 3 3 3 7 | r | r | r | op |
B: 3 3 6 #4 | r | r | a | 0000 |

3. What is the ***maximum*** total number of instructions?

   A. 119
   B. 120
   C. 121
   D. 122
   E. None of the above.

$1 + (2^4 - 1)(2^3) = 121$

4. What is the ***minimum*** total number of instructions?

   A. 22
   B. 21
   C. 20
   D. 19
   E. None of the above.

$2^3 + 2^4 - 1 = 23$

$(2^4 - 1) + 1 \times 2^3$

For questions 5 – 6:
Study the following C programs.

```
#include <stdio.h>

typedef struct {
  int numer[1]; // numerator
  int *denom;   // denominator
} rational;

void multiply(rational, rational*);

int main(void) {
  int val1 = 2, val2 = 5;
  rational num1 = {{1}, &val1}, // 1/2
           num2 = {{2}, &val2}; // 2/5
  multiply(num1, &num2);
  printf(“%d %d\n”, num1.numer[0], *(num1.denom)); // Question 3
  printf(“%d %d\n”, num2.numer[0], *(num2.denom)); // Question 4
}

void multiply(rational x, rational *y) {
  int x_num = *(x.numer), y_num = *(y->numer),
      x_den = *(x.denom), y_den = *(y->denom);
  *(x.numer)  = x_num * y_num;
  *(x.denom)  = x_den * y_den;
  *(y->numer) = x_num * y_num;
  *(y->denom) = x_den * y_den;
}
```

5. What is the output of the ***first*** print?

   A. 1 5
   B. 1 10
   C. 2 5
   D. 2 10
   E. None of the above

6. What is the output of the ***second*** print?

   A. 1 5
   B. 1 10
   C. 2 5
   D. 2 10
   E. None of the above

***Questions 7 – 10:* C & MIPS (Tracing, Compiling, Encoding)** **[Total: 14 marks]**

For the next *four* (4) questions, refer to the code below. The code has been partially filled in for you. One of the blanks has been filled for you.

C Code	MIPS Code

`int main(void) {`
`  i = 0;`
`  A = 'A';`
`  a = 'a';`
`  Z = 'Z';`
	`[ addi $s1, $zero, 0 ]`
	`[ addi $s3, $zero, 65 ]`
	`[ addi $s4, $zero, 97 ]`
	`[ addi $s5, $zero, 90 ]`

`  do {`	***loop:***

`    if( str[i] >= 'A'`
`        && str[i] <= 'Z' ) {`
	`add  $s7, $s0 , $s1`
	`lb   $t2, 0($s7)`
	`slt  $t1, $t2 , $s3`
	`bne  $t1, $zero, else`
	`[ slt $t1, $s5, $t2 ]`
	`[ bne $t1, $zero, else ]`

`      func(str + i);`	`j    func` ***ret:***
`    }`	***else:***
`[ i = i + 1; ];`	`addi $s1, $s1, 1`
`  } while(str[i-1] != 0);`	`[ bne $t2, $zero, loop ]`
`  return 0;` `}`	***quit:*** `j    exit`
// Code omitted	***# Code omitted***
`void func(char* str) {`	***func:***

`[ *str = *str + 'a' - 'A' ];`
`  return;`
	`lb   $t8, 0($s7)`
	`addi $t8, $t8,  97`
	`addi $t8, $t8, -65`
	`sb   $t8, 0($s7)`
	`j    ret`

`}`	
// Code omitted	***# Code omitted***
// End of Program	***exit:***

We will also use the following variable-to-register mapping within the ***main*** function and not the ***func*** function.

base addr of **str**	**$s0**	**A**	**$s3**
i	**$s1**	**a**	**$s4**
len	**$s2**	**Z**	**$s5**

Note that in the section marked with "**Code omitted**", there can be *any number* of instructions. The ASCII table, if required, is available at the end of the question paper.

7. Fill in the blanks with appropriate C code and MIPS code. Note that you must fill in ***exactly*** the given number of instructions. You CANNOT use *pseudo-instructions* for the MIPS instructions. [8 marks]

8. Consider running the program with the initial value of **str** as **"CS2100 is Easy!"**. What will be the final value of **str**? "cs2100 is easy!" [2 marks]

9. What is the encoding in ***hexadecimal*** for the 8th MIPS instruction "**bne $t1, $zero, else**"? [2 marks]
   0001 0101 0010 0000 0000 0000 0000 0011 = 0x15200003

10. [CHALLENGING] What is the ***maximum*** possible number of MIPS instructions that can be inserted into the regions marked with "**Code omitted**"? Note that the code can be inserted into any one of the two regions. We are only interested in the total. For simplicity, write your answer in terms of $2^x \pm y$ [2 marks]
   2^26 - 9

Question 11: **Number Systems** **[Total: 6 marks]**

For the next question, recall the IEEE 754 single-precision floating-point number representation.

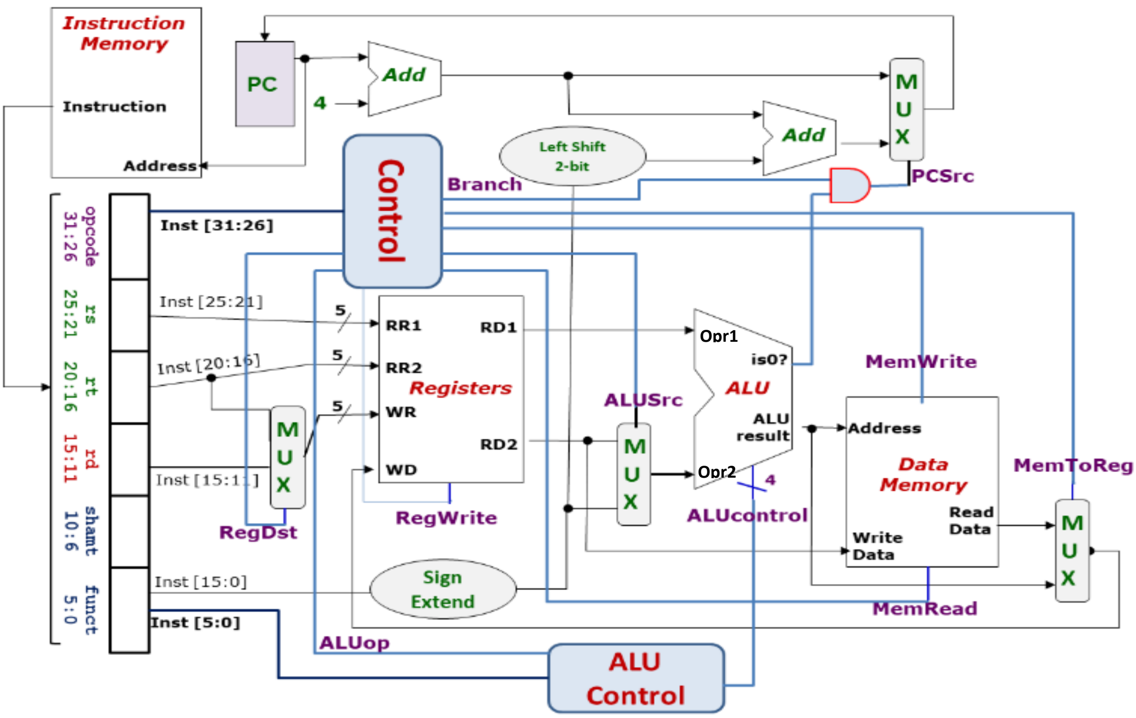
11. Given the following hexadecimal value in the IEEE 754 single-precision floating-point number representation: 1 100 00001 011 1011 0000 0000 0000 0000 = -101.1 1011 = -5.84375

0x C 0 B B 0 0 0 0

What decimal value does it represent? [6 marks]

Questions 12 – 13: **Datapath & Control** **[Total: 8 marks]**

For the next *two* (2) questions, refer to the diagram of the Datapath and Control below.



However, let us consider a *different* ***Control*** unit from the one we have in the lecture notes. Assume that our modified control unit produces the following control signals:

Instr. Type	RegDst	ALUsrc	MemToReg	RegWrite	MemRead	MemWrite	Branch
R-type	1	0	0	1	0	0	0
lw	0	1	1	1	1	0	0
sw	1	1	1	0	0	1	**1 (wrong)**
beq	1	0	1	0	1	0	1

However, **ALUop** is still the same as before. We will also be using the following notation:
- The value of constants will be given as constants (e.g., **-5**)
- The value of the register number 8 will be represented as **$8**
- The value stored in the register number 8 will be represented as **R[$8]**
- The value of an arithmetic operation (op) between two values A and B will be represented as A op B. For example, A + B in the case of addition.
- The value stored in the memory location L will be represented as **M[L]**

12. You are given an instruction below, fill in the table in the answer sheet. Assume that the label **L1** is converted to the immediate value **-2**.
    beq $8, $0, L1 [6 marks]
    $8, $zero, $zero OR $31, M[R[$8]-R[$0]], R[$8], R[$0], R[$8]-R[$0], R[$0]

13. **[CHALLENGING]** The instruction **sw** should not produce the value 1 for Branch control signal (as highlighted above). Give one MIPS instruction using **sw** that is *guaranteed* to cause the processor to perform a branch. [2 marks]

    sw $zero, 0($zero)

MIPS Reference Data

1

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate (15–0)		
J	opcode	address (25–0)				

ARITHMETIC CORE INSTRUCTION SET

2

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)

FLOATING-POINT INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
FR	opcode	fmt	ft	fs	fd	funct
FI	opcode	fmt	ft	immediate (15–0)		

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Positive Power of 2

Exp	Val	Exp	Val	Exp	Val	Exp	Val
2^{0}	1	2^{8}	256	2^{16}	65,536	2^{24}	16,777,216
2^{1}	2	2^{9}	512	2^{17}	131,072	2^{25}	33,554,432
2^{2}	4	2^{10}	1,024	2^{18}	262,144	2^{26}	67,108,864
2^{3}	8	2^{11}	2,048	2^{19}	524,288	2^{27}	134,217,728
2^{4}	16	2^{12}	4,096	2^{20}	1,048,576	2^{28}	268,435,456
2^{5}	32	2^{13}	8,192	2^{21}	2,097,152	2^{29}	536,870,912
2^{6}	64	2^{14}	16,384	2^{22}	4,194,304	2^{30}	1,073,741,824
2^{7}	128	2^{15}	32,768	2^{23}	8,388,608	2^{31}	2,147,483,648

Negative Power of 2

Exp	Val	Exp	Val
2^{-1}	0.5	2^{-9}	0.001953125
2^{-2}	0.25	2^{-10}	0.0009765625
2^{-3}	0.125	2^{-11}	0.00048828125
2^{-4}	0.0625	2^{-12}	0.000244140625
2^{-5}	0.03125	2^{-13}	0.0001220703125
2^{-6}	0.015625	2^{-14}	0.00006103515625
2^{-7}	0.0078125	2^{-15}	0.000030517578125
2^{-8}	0.00390625	2^{-16}	0.0000152587890625