

Lab-6

1. `cmpl %esi, %edi`
`setl %al-----> unsigned int/int , COMP <`
`cmpw %si, %di`
`setl %al----->unsigned short/short, COMP <`
`cmpb %sil, %dil`
`setbe %al-----> unsigned char/char , COMP <=`
`cmpq %rsi, %rdi`
`setne %a-----> unsigned long/long , COMP !=`
2. a. `4003fa: 74 02 je 0x4003fe`
`4003fc: ff d0 calq *%rax`
b. `40042f: 74 f4 je 0x400425`
`400431: 5d pop %rbp`
c. `400543: 77 02 ja 0x400547`
`400545: 5d pop %rbp`
d. `4005e8: e9 73 ff ff ff jmpq 0x400560`
`4005ed: 90 nop`
3. `movb $0xF, (%ebx)`
`movq %rax, (%rsp) # өгөдлийн урт, регистрэйн нэр таарахгүй`
`movw (%rax), %ax #`
`movw %ax, 4(%rsp) # memory direct move`
`movb %al, %sil # %sl -> %sil`
`movq $0x123, %rax # $0x123 dst байж болохгүй`
`movl %eax, %edx # өгөдлийн хэмжээ таарахгүй`
`movw %si, 8(%rbp) # бит өргөн таарахгүй`
4. `void cond(long a, long *p)`

```
{
    if (!p) goto end;
    if (a <= *p) goto end;
    *p = a;
end:
    return;
}
```
5. `long test(long x, long y, long z) {`
 `long val = x + y + z;`
 `if (x < -3) {`
 `if (y < z)`
 `val = x * y;`

```

        else
            val = y * z;
    } else if (x > 2)
        val = x * z;
    return val;
}
6. long loop_while(long a, long b) {
    long result = 1;
    while (a < b) {
        result = result * (a + b);
        a = a + 1;
    }
    return result;
}

```

Lab-5

1. Операнд Утга

%rax	0x100
0x104	0xAB
\$0x108	0x13?
(%rax)	0xFF
4(%rax)	0xAB
9(%rax, %rdx)	0x11
0xFC(%rcx,4)	0x100
(%rax,%rdx, 4)	0x10C
2. movl %eax, (%rsp)
 movw (%rax), %dx
 movb \$0xFF, %b1
 movl (%rsp,%rdx,4), %d1
 movq (%rdx), %rax
 mov_w%dx, (%rax)
3. movb \$0xF, (%ebx)->Error: invalid register `'%ebx' for 64-bit mode
 movl %rax, (%rsp)->Error: invalid register `'%rax' for 32-bit instruction
 movw (%rax), 4(%rsp)-> Error: invalid combination of opcode and operands
 movb %al, %sl->Error: invalid register name `'%sl'
 movq %rax, \$0x123 ->Error: immediate operand required in the source
 movl %eax, %rdx -> Error: invalid register `'%rdx' for 32-bit instruction
 movl %si, 8(%rbp) -> Error: invalid register `'%si' for 32-bit instruction
4. 000000100003ee4 <_fact>:

100003ee4: d10083ff	sub sp, sp, #0x20
100003ee8: a9017bfd	stp x29, x30, [sp, #0x10]
100003eec: 910043fd	add x29, sp, #0x10

100003ef0: b9000be0	str w0, [sp, #0x8]
100003ef4: b9400be8	ldr w8, [sp, #0x8]
100003ef8: 71000108	subs w8, w8, #0x0
100003efc: 1a9f07e8	cset w8, ne
100003f00: 370000a8	tbnz w8, #0x0, 0x100003f14 <_fact+0x30>
100003f04: 14000001	b 0x100003f08 <_fact+0x24>
100003f08: 52800028	mov w8, #0x1 ;=1
100003f0c: b81fc3a8	stur w8, [x29, #-0x4]
100003f10: 1400000a	b 0x100003f38 <_fact+0x54>
100003f14: b9400be8	ldr w8, [sp, #0x8]
100003f18: 71000500	subs w0, w8, #0x1
100003f1c: 97fffff2 bl	0x100003ee4 <_fact>
100003f20: b90007e0	str w0, [sp, #0x4]
100003f24: b94007e8	ldr w8, [sp, #0x4]
100003f28: b9400be9	ldr w9, [sp, #0x8]
100003f2c: 1b097d08	mul w8, w8, w9
100003f30: b81fc3a8	stur w8, [x29, #-0x4]
100003f34: 14000001	b 0x100003f38 <_fact+0x54>
100003f38: b85fc3a0	ldur w0, [x29, #-0x4]
100003f3c: a9417bfd	ldp x29, x30, [sp, #0x10]
100003f40: 910083ff	add sp, sp, #0x20
100003f44: d65f03c0	ret

0000000100003f48 <_main>:

100003f48: d100c3ff	sub sp, sp, #0x30
100003f4c: a9027bfd	stp x29, x30, [sp, #0x20]
100003f50: 910083fd	add x29, sp, #0x20
100003f54: 52800008	mov w8, #0x0 ;=0
100003f58: b81f43a8	stur w8, [x29, #-0xc]
100003f5c: b81fc3bf	stur wzr, [x29, #-0x4]
100003f60: 528000a0	mov w0, #0x5 ;=5
100003f64: 97ffffe0	bl 0x100003ee4 <_fact>
100003f68: b81f83a0	stur w0, [x29, #-0x8]
100003f6c: b85f83a9	ldur w9, [x29, #-0x8]
100003f70: aa0903e8	mov x8, x9
100003f74: 910003e9	mov x9, sp
100003f78: f9000128	str x8, [x9]
100003f7c: 90000000	adrp x0, 0x100003000 <_printf+0x100003000>
100003f80: 913e9000	add x0, x0, #0xfa4
100003f84: 94000005	bl 0x100003f98 <_printf+0x100003f98>
100003f88: b85f43a0	ldur w0, [x29, #-0xc]

```

100003f8c: a9427bfd      ldp    x29, x30, [sp, #0x20]
100003f90: 9100c3ff      add    sp, sp, #0x30
100003f94: d65f03c0      ret

```

Disassembly of section __TEXT,__stubs:

0000000100003f98 <__stubs>:

```

100003f98: b0000010      adrp   x16, 0x100004000 <_printf+0x100004000>
100003f9c: f9400210      ldr    x16, [x16]
100003fa0: d61f0200      br    x16

```

```

5. void decode1(long *xp, long *yp, long *zp) {
    long x = *xp;
    long y = *yp;
    long z = *zp;
    *yp = x;
    *zp = y;
    *xp = z;
}

```

Lab-4

1.	Бутархай тоо	Хоёртын тоололд	Аравтын тоололд
	1/8	0,001	0,125
	3/4	0,11	0,75
	5/16	0,0101	0,3125
	10 11/16	1010,1011	10,6875
	1 1/8	1,001	1,125
	5 7/8	101,111	5,875
	3 3/16	11,001	3,1875
2.	e E 2E f M 2E*M V	Аравт	
	0 00 01 0 0 1	1/4 1/4 1/4	0.25
	0 00 10 0 0 1	1/2 1/2 1/2	0.5
	0 00 11 0 0 1	3/4 3/4 3/4	0.75
	0 01 00 1 0 1	0 1 1	1
	0 01 01 1 0 1	1/4 1.25 1.25	1.25
	0 01 10 1 0 1	1/2 1.5 1.5	1.5
	0 01 11 1 0 1	3/4 1.75 1.75	1.75
	0 10 00 2 1 2	0 1 2	2
	0 10 01 2 1 2	1/4 1.25 2.5	2.5
	0 10 10 2 1 2	1/2 1.5 3	3
	0 10 11 2 1 2	3/4 1.75 3.5	3.5

0 11 00	3	2	4	0	1	4	4
0 11 01	3	2	4	1/4	1.25	5	5
0 11 10	3	2	4	1/2	1.5	6	6
0 11 11	3	2	4	3/4	1.75	7	7

3. Float нь тоог экспонент ба мантисад задлан кодолдог тул Int-ийн битийн хээгээс өөр гардаг.
4. hamgiin_baga_ilerhiilehgui_buhel=pow(2, n+1)+1
5. `typedef unsigned float_bits;`

```

float_bits float_twice(float_bits f) {
    unsigned sign = f >> 31;
    unsigned exp = (f >> 23) & 0xFF;
    unsigned frac = f & 0x7FFFFFF;

    if (exp == 0xFF) {
        return f;
    } else if (exp == 0) {
        frac <<= 1;
        if (frac & 0x800000) {
            exp = 1;
            frac &= 0x7FFFFFF;
        }
    } else {
        exp++;
        if (exp == 0xFF) {
            frac = 0;
        }
    }

    return (sign << 31) | (exp << 23) | frac;
}

```

Lab-3

1. x reversed

H D	H D
2->2	E-> 14
3->3	7-> 7
9->9	D-> 13

B-> 11 5-> 5
C-> 12 4-> 4

2. Mode x y x·y (Binary) Truncated
Unsigned [100] (4) [101] (5) [10100] (20) [100] (4)
Two's complement [100] (-4) [101] (-3) [01100] (12) [100] (-4)
Unsigned [010] (2) [111] (7) [01110] (14) [110] (6)
Two's complement [010] (2) [111] (-1) [11110] (-2) [110] (-2)
Unsigned [110] (6) [110] (6) [100100] (36) [100] (4)
Two's complement [110] (-2) [110] (-2) [00100] (4) [100] (-4)
3. #include <limits.h>

```
int tmult_ok_complex(int x, int y) {  
    if (x > 0 && y > 0) {  
        if (x > INT_MAX / y) return 0;  
    } else if (x < 0 && y < 0) {  
        if (x < INT_MAX / y) return 0;  
    } else if (x > 0 && y < 0) {  
        if (y < INT_MIN / x) return 0;  
    } else if (x < 0 && y > 0) {  
        if (x < INT_MIN / y) return 0;  
    }  
    return 1;  
}
```

4. #include <stdlib.h>
#include <string.h>
#include <limits.h>

```
void *copy_elements_safe(void *ele_src[], int ele_cnt, size_t ele_size) {  
    if (ele_cnt < 0) {  
        return NULL;  
    }  
    size_t num_elements = (size_t)ele_cnt;  
  
    if (ele_size > 0 && num_elements > SIZE_MAX / ele_size) {  
        return NULL;  
    }  
  
    size_t total_size = num_elements * ele_size;  
    void *result = malloc(total_size);  
  
    if (result == NULL) {
```

```

        return NULL;
    }

void *next = result;
int i;

for (i = 0; i < ele_cnt; i++) {
    // Copy object i to the destination.
    memcpy(next, ele_src[i], ele_size);

    // Move the pointer to the next memory region.
    next = (char *)next + ele_size;
}
return result;
}

```

5. K shifts add/subs x*K

7	1	1	(x << 3) - x
30	4	3	(x << 4) + (x << 3) + (x << 2) + (x << 1)
28	2	1	((x << 3) - x) << 2
55	2	2	(x << 6) - ((x << 3) + x)
6. x=(x*32)-x=31x
M=31;
if (y < 0) y += 7;
N=8;
7. (a) (x > 0) || (x-1 < 0) //Hudal, hasah too
(b) (x & 7) == 7 || (x << 29) < 0 // Hudal x=8 uyd
(c) (x * x) >= 0 // Hudal x=46341
(d) x < 0 || -x <= 0 //True
(e) x > 0 || -x >= 0 //True
(g) x * ~y + uy * ux == -x //True

Lab-2

1. 1111 -8+4+2+1
1011 -8+0+2+1
1010 -8+0+2+0
1100 -8+4+0+0
0001 0+0+0+1
1000 8+0+0+0
2. fun1(0x00000076)->118, fun2(0x00000076)-> 118
fun1(0x87654321)->33, fun2(0x87654321)-> 33

```
fun1(0x000000C9)->201, fun2(0x000000C9)-> -55  
fun1(0xEDCBA987)->135, fun2(0xEDCBA987)-> -121
```

3. #include <stdio.h>

```
int uadd_ok(unsigned x, unsigned y) {  
    unsigned sum = x + y;  
    return (sum >= x) ? 1 : 0;  
}
```

```
int main() {  
    unsigned a, b;  
    scanf("%u %u", &a, &b);  
    printf("%d\n", uadd_ok(a, b));  
    return 0;  
}
```

4. #include <stdio.h>

```
int tadd_ok(int x, int y) {  
    int sum = x + y;  
    if (x > 0 && y > 0 && sum < 0) return 0;  
    if (x < 0 && y < 0 && sum >= 0) return 0;  
    return 1;  
}
```

```
int main() {  
    int a, b;  
    scanf("%d %d", &a, &b);  
    printf("%d\n", tadd_ok(a, b));  
    return 0;  
}
```

5. #include <stdio.h>

```
int tsub_ok(int x, int y){  
    int sub = x + y;  
    if (x > 0 && y > 0 && sub < 0) return 0;  
    if (x < 0 && y < 0 && sub >= 0) return 0;  
    return 1;  
}  
int main() {  
    int a, b;  
    scanf("%d %d", &a, &b);  
    printf("%d\n", tsub_ok(a, b));
```

```
    return 0;
}
6. x <=>= 5 // (Left shift) 2iin 5 zereg buyu 32, 32x
  x -= t // 32x - x = 31x.
  y >=>= 3 // (Right shift) y/8.
```

M = 31

N = 8