Homework 2

- **Due: Nov. 6th**
- **Please submit your homework to e-learning server with format like 223024****.pdf**

1、Assume we have following address binding table and value of registers

Address	Value	Register	Value
0xbffff0f8	0x00000001	%rax	0xc
0xbffff0fc	0xdeadbeef	%rbx	0xbffff108
0xbffff100	0x10	%rdx	0x4
0xbffff104	0x11	%rbp	0xbffff110
0xbffff108	0x12	%rsp	0xbffff100
0xbffff110	0xbffff138		
0xbffff114	0x8010240		
0xbffff120	0xbffff134		
0xbffff130	0x13		
0xbffff134	0x14		
0xbffff138	unknown		

Addressing

Please fill in the table below

Operand	Value
$0xbffff100	0xbffff100
0xbffff110	0xbffff138
%rbx	0xbffff108
(%rbx)	0x12
(%rbx, %rax)	0x8010240
0x4(%rsp, %rdx)	0x12
-0x10(%rbp, %rdx, 4)	0xbffff138

Instructions

Suppose registers and bound values will be reset as above after each instruction. Please fill in the table below:

Instruction	Destination's Value
movq 0x4(%rbp, %rax), %rbx	%rbx = 0xbffff134
movb %al, %bl	%rbx = 0xbffff10c
movw %bp, %bx	%rbx = 0xbffff110
movsbq %bl, %rsp	%rsp = 0x8
movzbq %bl, %rsp	%rsp = 0x8

pushq %rbp	%rsp = 0xbffff0f8 (%rsp) = 0xbffff110
popq %rax	%rsp = 0xbffff108 %rax= 0x10 (%rsp) = 0x12

2、Assembly

Consider the following bit of C code and its part of disassembled IA64 machine code.

int main() {
1 char a[4] = "f";
2 char b[4];
3 int c = 2;
4 c = someFunc(a, b, &c);
5 return 0;
}
	someFunc:
	
	pushq %rbp
	movq %rsp,%rbp
	movq %rdi,-0x8(%rbp) //char* a
	movq %rsi,-0x10(%rbp) //char* b
	movq %rdx,-0x18(%rbp) //int* c
	movq -0x8(%rbp),%rax //rax = a
	movzbl (%rax),%edx //edx = a[0]
	movq -0x10(%rbp),%rax //rax = b
	movb %dl,(%rax) //b[0] = a[0]
	movq -0x18(%rbp),%rax //rax = c
	movq (%rax),%eax //rax = *c
	leaq 0x1(%rax),%edx //rdx = *c + 1
	movq -0x18(%rbp),%rax //rax = c
	movq %edx,(%rax) //*c = edx = *c + 1
	movq $0x1,%eax //eax = 1
	popq %rbp
	retq

1. Translate the assembly in the right column into C codes.
2. Fill the table below when the C code executed in line 5

Variable	Variable's value
b[0]	'f'
c	1

Your C code:

```
int someFunc(char* a, char* b, int* c)
{
  b[0] = a[0];
  *c = *c + 1;
  return 1;
}
```

3、Consider the following source code, where NR and NC are macro expressions declared with #define that compute the dimensions of array A in terms of parameter n. This code computes the sum of the elements of column j of the array.

```
long sum_col(long n, long A[NR(n)][NC(n)], long j) {
    long i;
    long result = 0;
    for (i = 0; i < NR(n); i++)
        result += A[i][j];
    return result;
}
```

In compiling this program, gcc generates the following assembly code:

```
long sum_col(long n, long A[NR(n)][NC(n)], long j)
n in %rdi, A in %rsi, j in %rdx
sum_col:
  leaq    1(,%rdi,4), %r8
  leaq    (%rdi,%rdi,2), %rax
  movq    %rax, %rdi
  testq   %rax, %rax
  jle     .L4
  salq    $3, %r8
  leaq    (%rsi,%rdx,8), %rcx
  movl    $0, %eax
  movl    $0, %edx
.L3:
  addq    (%rcx), %rax
  addq    $1, %rdx
  addq    %r8, %rcx
  cmpq    %rdi, %rdx
  jne     .L3
  rep; ret
.L4:
  movl    $0, %eax
  ret
```

Use your reverse engineering skills to determine the definitions of NR and NC.

Ans:

```
#define NR (3*(n))
#define NC (4*(n) + 1)
```

Deductions :

```
A in rsi, n in rdi, j in rdx
r8 = 4rdi + 1 = 4n + 1;
rax = 3rdi = 3n;
rdi = rax = 3n;
if(rax <= 0)
{
  rax = 0;
  return;
  // if (n<=0) return n
}
else
{
  r8 *= 8; // r8 = 8(4n+1)
  rcx = rsi + 8rdx = A + 8 * j;
  rax = 0; // result = 0;
  rdx = 0; // i = 0;
```

```
  while(rdx != rdi) // NR(n) = 3n;
  {
    rax += M[rcx]; // result += A[0][j], A[1][j], ……;
    rdx ++; // i ++;
    rcx += r8; //NC(n) = r8 / 8 = 4n+1
  }
}
```

Ans: