Homework06

Problem 1: Procedures(x86-64)

Consider the following assembly code segment compiled on x86-64 machine and answer the questions.

```
 1  fun:
 2  .LFB11:
 3      .cfi_startproc
 4      leaq     (%rdi,%rdi), %r10
 5      leaq     (%r10,%rdi,8), %rax
 6      addq     %rax, %rsi
 7      addq     %rsi, %r10
 8      movq     %rdx, %rax
 9      cqto
10      idivq    %rcx
11      imulq    %rax, %r8
12      addq     %r8, %r10
13      movq     8(%rsp), %rax
14      cqto
15      idivq    16(%rsp)
16      subq     %rax, %r9
17      leaq     (%r10,%r9), %rax
18      ret
19  .cfi_endproc
```

(1) Assume that all reference to stack frame only use stack pointer *%rsp*, how many arguments are supposed to pass in order to call *fun*?

Eight Arguments. (Six of them are stored in registers and two of them are stored in stack)

(2) Explain the assembly codes between line 8 and line 10. Use R*[%reg]* to represent the value of register *reg*.

Eight Arguments. (Six of them are stored in registers and two of them are stored on stack)
Codes between line 8 and line 10 computes R[*%rdx*]/R[*%rdx*]. Line 8 move quad word from register *%rdx* to register *%rax*. Line 9 convert R[*%rax*] to oct word and store the result in R[*%rdx*]:R[*%rax*]. Line 10 computer signed division and store the quotient in R[*%rax*].(According to line 11, the valid value is quotient in R[*%rax*], rather than remainder in R[*%rdx*]).

(3) Translate the assembly codes of fun to C codes. Donate the return value as *long* type, the parameters as *long* type and name *p1,p2,p3*…from left to right in the parameter lists.

<u>The source code is:</u>

```
long fun(long p1, long p2, long p3, long p4, long p5, long p6,
long p7, long p8){
    long v1 = p1 * 2;
    long v2 = p2 + v1 * 5;
    long v3 = p3 / p4 * p5;
    long v4 = p6 - p7 / p8;
    return v1 + v2 + v3 + v4;
}
```

<u>Or the answer can also be simply written as:</u>

```
long fun(long p1, long p2, long p3, long p4, long p5, long p6,
long p7, long p8){
    return p1 * 12 + p2 + p3 / p4 * p5 + p6 - p7 / p8;
}
```

Problem 2: Arrays

Consider the C code segment compiled on IA32 machine where the value of M and N are hidden deliberately:

```
#include <stdio.h>

#define M ?
#define N ?

int a[M][N];
int b[N][M];

in tele_mul(int i, int j){
    return a[i][j] * b[j][i];
}
```

(1) If the corresponding assembly code with –O0 optimization using GCC is as follows:

```
ele_mul
.LFB0
    .cfi_startproc
    pushl  %ebp
    .cfi_def_cfa_offset  8
    .cfi_offset   5, -8
    movl   %esp, %ebp
    .cfi_def_cfa_register 5
    movl   8(%ebp), %eax
    leal   (%eax,%eax), %edx
```

```
11      movl   12(%ebp), %eax
12      addl   %edx, %eax
13      movl   a(,%eax,4), %ecx
14      movl   12(%ebp), %edx
15      movl   %edx, %eax
16      sall   $2, %eax
17      addl   %edx, %eax
18      movl   8(%ebp), %edx
19      addl   %edx, %eax
20      movl   b(,%eax,4), %eax
21      imull  %ecx, %eax
22      popl   %ebp
23      .cfi_restore 5
24      .cfi_def_cfa 4,4
25      ret
26      .cfi_endproc
```

Inter the value of constants M and N and give your reason.

M = 5 and N = 2. From line 9 to line 12 we can get R[%eax] = 2i + j. That is, each row a[i] contains j elements and a[i][j] is accessed via address (base + (2 * i + j) * 4). There fore, M = 2. Similarly, from line 14 to line 19 we can get R[%eax] = 5j+i.Therefore, N = 5.

(2) If the corresponding assembly code with –O3 optimization using GCC is as follows:

```
1  ele_mul:
2  .LFB11:
3      .cfi_startproc
4      movl   4(%esp), %edx
5      movl   8(%esp), %eax
6      leal   (%eax,%edx,4), %ecx
7      leal   (%eax,%eax,3), %eax
8      leal   (%edx,%eax,2), %edx
9      movl   a(,%ecx,4), %eax
10     imull  b(,%edx,4), %eax
11     ret
12     .cfi_endproc
```

Inter the value of constants M and N and give your reason.

M = 8 and N = 4. Note that %esp is no longer used to refer to stack frame since –O3 optimization is aggressive. Thus, %ebp does not need to be saved, and 4(%esp) saves the first parameter i and 8(%esp) saves the second parameter j. From line 6 we can get R[%ecx] = 4i + j and from line 7 to line 8 we can get R[%edx] = 8j + i. Similar to question(1), clearly M = 8 and N = 4.