課題 3

1)

$ gcc -O -c quiz1.c

これで quiz1.o(quiz1-2018-macos.o)が生成される。(アセンブル)

これを逆アセンブルする。

$ objdump -d quiz1-2018-macos.o

出力結果

```
kamiyakntanoMBP:プログラミングB kamiyakenta$ objdump -d quiz1-2018-macos.o

quiz1-2018-macos.o:  file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:
_arith:
       0:  8d 14 7f   leal  (%rdi,%rdi,2), %edx          %rdi + 2*%rdi を%edxに格納
       3:  8d 86 ff 03 00 00 leal  1023(%rsi), %eax     %rsi + 1023 を %eaxに格納
       9:  85 f6    testl %esi, %esi                    %esiの正負判定
       b:  0f 49 c6   cmovnsl  %esi, %eax               cmovsは一つ前のtestで負だった場合のみ%esiから%eaxへの転送を行う
       e:  c1 f8 0a   sarl  $10, %eax                   算術右シフトで%eax/2^10 (10右へシフト)
      11:  8d 04 d0  leal  (%rax,%rdx,8), %eax          %rax + 8*%rdx を%eaxに格納
      14:  c3    retq                                  %eaxが返り値で呼び出し元に戻る
```

アセンブリ言語からどのような挙動かをコメントしてある。

これを元に挙動を c で軽く書くと以下のように書ける。(右にあるコメントは変数の説明)

```
temp1 =  x + 2x;                %edx => temp1, %rdi => x
temp2 = y + 1023;               %eax => temp2, %rsi => y
if (y>0){                       %esi => y
  temp2 = y;
}
temp2 >> 10;
temp2 = temp2 + 8*temp1;        %rax => temp2, %rdx => temp1
return temp2;
```

二行目で 1023 足しているのは y<0 の場合を考慮していると考えられる。

また、temp2>>10 は 2^10 を割っている。

これを元にスライドのコードに合わせて書くと以下のように書けた。

```
//答え----------------------------------------------------------------------------------------------------------

long arith (long x, long y){
long result;
result = x*24 + y/1024 ;
return result;
}
```

2)

1)と同様に逆アセンブリした結果以下のようなコードが得られた。

```
kamiyakntanoMBP:プログラミングB kamiyakenta$ objdump -d quiz2-2018-macos.o

quiz2-2018-macos.o:  file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:
_arith:
       0:  89 f8    movl %edi, %eax              %ediの値を%eaxに格納
       2:  c1 e0 08  shll  $8, %eax              算術左シフトで%eax * 2^8(8左へシフト)
       5:  29 f8    subl  %edi, %eax             %eax - %ediを%eaxに格納
       7:  8d 56 1f   leal  31(%rsi), %edx       %rsi + 31を%edxに格納
       a:  85 f6    testl %esi, %esi             %esiの正負判定
       c:  0f 49 d6   cmovnsl   %esi, %edx       %esiが正だった場合のみ%esiから%edxへの転送を行う
       f:  c1 fa 05   sarl  $5, %edx             算術右シフトで%edx / 2^5(5右へシフト)
      12: 29 d0   subl  %edx, %eax               %eax - %edxを%eaxに格納
      14: c3    retq                            返り値
```

これを読解しイメージしたコードが以下のコードになる。(問題 1 と手順は何も変わらない)

```
temp2 =  x;                     %eax => temp2, %edi => x
temp2 << 8;
temp2 = temp2 - x;
temp1 = y + 31;                 %edx => temp1, %rsi => y
if (y>0){                       %esi => y
  temp1 = y;
}
temp1 >> 5;                     2^5で割ってる
temp2 = temp2 - temp1;
return temp2;
```

今回(2^8-1)*temp2 と temp1/2^5 の引き算になっており、y に 31 足したのは問題 1 と同じで y<0 の時を考慮しているからだと考えられる。

以下が C のコードを予測した回答

```
//答え-------------------------------------------------------------------------------------

long arith (long x, long y){
long result;
result = x*255 - y/32 ;
return result;
}
```

3)

1),2)と同様に逆アセンブリした結果を以下に示す。

```
kamiyakntanoMBP:プログラミングB kamiyakenta$ objdump -d quiz3-2018-macos.o

quiz3-2018-macos.o:  file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:
_arith:
       0:  8d 0c ff          leal    (%rdi,%rdi,8), %ecx       %rdi + 8*%rdi を%ecxに格納
       3:  ba 39 8e e3 38    movl    $954437177, %edx          954437177を%edxに格納
       8:  89 f0             movl    %esi, %eax                %esiの値を%eaxに格納
       a:  f7 ea             imull   %edx                      %eax * %edx を%edxに格納
       c:  d1 fa             sarl    %edx                      算術右シフトで%edx/2(1右シフト)
       e:  c1 fe 1f          sarl    $31, %esi                 算術右シフトで%esi/2^31(31右シフト)
      11: 29 f2              subl    %esi, %edx                %edx - %esiを%edxに格納
      13: 8d 04 ca           leal    (%rdx,%rcx,8), %eax       %rdx + 8*%rcx を%eaxに格納
      16: c3                 retq                              返り値
```

これを読解し、イメージしたコードが以下のコードになる。

```
temp3 =  x + 8*x;           %ecx => temp3, %rdi => x
temp2 = 954437177;          %edx => temp2              954437177(10) = 11100011100011100011100011001(2)
temp1 = y;                  %eax => temp1, %esi => y   (仮にy=-1だとすると11111111111111111111111111111111(2))
temp2 = temp2 * temp1;      %rsi => y                  y<0の時掛け算するとものすごい大きい二進数になる
temp2 >> 1;                 2で割ってる                (y>=0の時 temp2 = 477218588y I y<0の時 temp2はものすごい大きいまま )
y >> 31;                    y>=0の時0, y<0の時1 要するに符号
temp2 = temp2 - y;          (y:正   477218588y 、y:負   ものすごい大きい値 - 1)
temp1 = temp2 + 8*temp3     temp1 = 72x + ?y
return temp1;
```

このレジスタの過程を元にコードを推測する。

```
//答え----------------------------------------------------------------------------------------------

long arith (long x, long y){
long result;
result = x*72 + ?y ;
return result;
}
```

4)全体の考察

今回のアセンブリ言語を読むにあたって重要な点があると考える。それは y が負だった時の場合を考慮して一度数字を足すことだ。これを行うことにより y>0 の時は if 文を通り temp=y で普通に計算が行われ、y<0 の時は if 文を通らないが数字を足しているので正常な計算が可能となる。また、整数レジスタの対応をしっかりと確認することが今回は重要だと考える。

5)感想

意外と時間がかかってしまった。また、最後の問題の N がわからないので解説をしっかり聞きたいと思う。