**고급시스템프로그래밍 3주차 실시간수업 과제**

|  |
| --- |
| 제출일: 2021.03.25  이름: 김현용 |

1. 다음 프로그램을 수정하여(“// 수정” 부분만 수정) 모든 단계를 통과하도록 하시오.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdint.h>  int main() {  int x;  float f;  double d;  double d1, d2;  x = 0; // 수정  if ((int)(float)x != x)  printf("#1 Passed! (x:%d)\n", x);  else  printf("#1 Failed! (x:%d)\n", x);  d = 0;  if ((double)(float)d != d)  printf("#1 Passed! (d:%lf)\n", d);  else  printf("#1 Failed! (d:%lf)\n", d);  d1 = 0; // 수정  d2 = 0; // 수정  if ((d1+d2)+d != d1+(d2+d))  printf("#1 Passed! (d1,d2,d:%lf,%lf,%lf)\n", d1, d2, d);  else  printf("#1 Failed! (d1,d2,d:%lf,%lf,%lf)\n", d1, d2, d);  d1 = 0; // 수정  d2 = 0; // 수정  if ((d1+d2)-d1 != d2)  printf("#1 Passed! (d1,d2:%lf,%lf)\n", d1, d2);  else  printf("#1 Failed!! (d1,d2:%lf,%lf)\n", d1, d2);  } |

A1\_1) 소스 코드

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <math.h>

int main() {

int x;

float f;

double d;

double d1, d2;

x = 16777217; // 수정

if ((int)(float)x != x)

printf("#1 Passed! (x:%d)\n", x);

else

printf("#1 Failed! (x:%d)\n", x);

d = 16777217;

if ((double)(float)d != d)

printf("#1 Passed! (d:%lf)\n", d);

else

printf("#1 Failed! (d:%lf)\n", d);

d1 = 0.1; // 수정

d2 = -0.11; // 수정

if ((d1+d2)+d != d1+(d2+d))

printf("#1 Passed! (d1,d2,d:%lf,%lf,%lf)\n", d1, d2, d);

else

printf("#1 Failed! (d1,d2,d:%lf,%lf,%lf)\n", d1, d2, d);

d1 = 1.8\*pow(10,308); // 수정

d2 = 1.8\*pow(10,308); // 수정

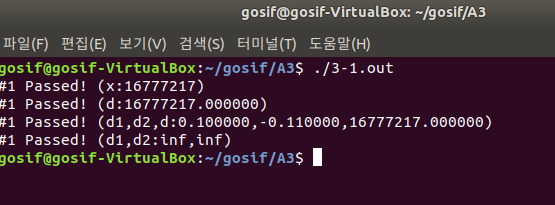
if ((d1+d2)-d1 != d2)

printf("#1 Passed! (d1,d2:%lf,%lf)\n", d1, d2);

else

printf("#1 Failed!! (d1,d2:%lf,%lf)\n", d1, d2);

}

A1\_2) 실행 결과  


A1\_3) 코드 설명  
-첫 번째 수정 : (int)(float)x != x 에서 x는 원래 int형이었기 때문에 float으로 바꾼 후 다시 int로 바꾸면 기존의 x와 같아야 된다고 생각할 수 있지만, float형이 표현가능한 숫자와 int형이 표현가능한 숫자는 차이가 있다. float형은 최대 23bits까지 표현 가능한데 2^24인 16777216보다 클 경우에는 홀수 표현을 못하게 되는 것이다.(최하위 비트는 0으로 인식) 따라서 16777217을 float형으로 바꾸면 16777216이 되고, 다시 int로 바꾸면 16777216이 되어서 원래 x와 같지 않게 된다.  
-두 번째 수정 : 첫 번째와 마찬가지의 이유로 double은 int보다도 더 큰 범위이기 때문에 float의 제한범위 밖의 수를 넣어주면 기존의 수와 같지 않게 된다.

-세 번째 수정 : (d1+d2)+d 와 d1+(d2+d)는 단순히 결합 법칙이 성립하는 지에 관한 것인데, double에서의 소수 표현은 (-1)^S\*M\*2^E 으로 가능한데 여기서 양수에서 음수나 음수에서 양수로 가게 되면 유효 숫자 손실이 발생하게 된다. 따라서 위의 경우에 결합 법칙이 성립되지 않는다.  
-네 번째 수정 : 세번 째와 마찬가지로 소수로 표현하면 통과가 되겠지만, 또 다른 경우를 생각해 보았다. double에서의 무한대를 표현하는 수인 inf에서 inf를 빼면 NAN이 될 것이기 때문에 위의 식이 성립하지 않게 될 것이다.

2. 다음 프로그램을 컴파일한 어셈블리 파일 2.s를 참고하여 물음에 답하시오..

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  long arith  (long x, long y, long z)  {  //???????????????  }  int main() {  printf("%ld\n", arith(1,2,3));  } |

1) 2.s를 컴파일(gcc -o 2 2.s)해 수행해 보고, arith 함수의 동작을 설명하시오.

gcc -o 2. 2.s를 이용해 컴파일하고, 실행파일 2를 실행해 본 결과 45가 출력되는 것을 확인했다. 이 때 2.s에서의 arith 함수의 어셈블리 코드는 다음과 같았다.  
arith:

.LFB38:

.cfi\_startproc

leaq (%rdi,%rsi,4), %rcx

leaq (%rdx,%rdx,2), %rdx

leaq 0(,%rdx,4), %rax

addq %rcx, %rax

ret

.cfi\_endproc

이 때, main에서  
 movl $3, %edx

movl $2, %esi

movl $1, %edi

call arith  
로 edx = 3, esi = 2, edi = 1의 값이 저장되어 있는 것을 확인할 수 있었고, 따라서 arith 함수에서는  
 leaq (%rdi,%rsi,4), %rcx ;rcx = rdi(1) + 4\*rsi(2) = 1+8 = 9

leaq (%rdx,%rdx,2), %rdx ;rdx = rdx(3) + 2\*rdx(3) = 3+6 = 9

leaq 0(,%rdx,4), %rax ;rax = 0 + 4\*rdx(9) = 4\*9 = 36

addq %rcx, %rax ;rax = rax(36) + rcx(9) = 45

라는 것을 알게 되었다.

2) 2.s를 arith 함수가 leaq를 사용하지 않고, addq와 salq, movq를 사용하여 똑같이 동작하도록 프로그램을 수정하시오.(어셈블리 코드와 실행화면을 제시하시오.)

위의 arith 함수의 주석문을 보고 salq로 shift연산을 해 2의 배수 곱셈을 할 수 있고, addq로 덧셈 연산을 할 수 있다. 따라서 수정하면 다음과 같다.  
arith:

.LFB38:

.cfi\_startproc

salq $2, %rsi

addq %rsi, %rdi

movq %rdi, %rcx

movq %rdx, %rdi

salq $1, %rdx

addq %rdi, %rdx

salq $2, %rdx

movq %rdx, %rax

addq %rcx, %rax

ret

.cfi\_endproc  
;나머지 코드 동일

