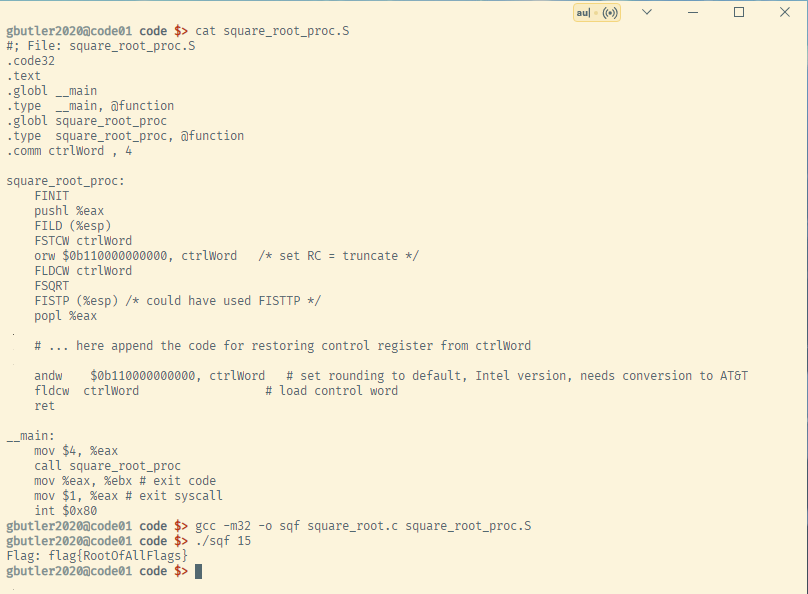
##### task-1:

| #; File: square\_root\_proc.S .code32 .text .globl \_\_main .type \_\_main, @function .globl square\_root\_proc .type square\_root\_proc, @function .comm ctrlWord , 4  square\_root\_proc:  FINIT  pushl %eax  FILD (%esp)  FSTCW ctrlWord  orw $0b110000000000, ctrlWord /\* set RC = truncate \*/  FLDCW ctrlWord  FSQRT  FISTP (%esp) /\* could have used FISTTP \*/  popl %eax   # ... here append the code for restoring control register from ctrlWord   andw $0b110000000000, ctrlWord # set rounding to default, Intel version, needs conversion to AT&T  fldcw ctrlWord # load control word  ret  \_\_main:  mov $4, %eax  call square\_root\_proc  mov %eax, %ebx # exit code  mov $1, %eax # exit syscall  int $0x80 |
| --- |

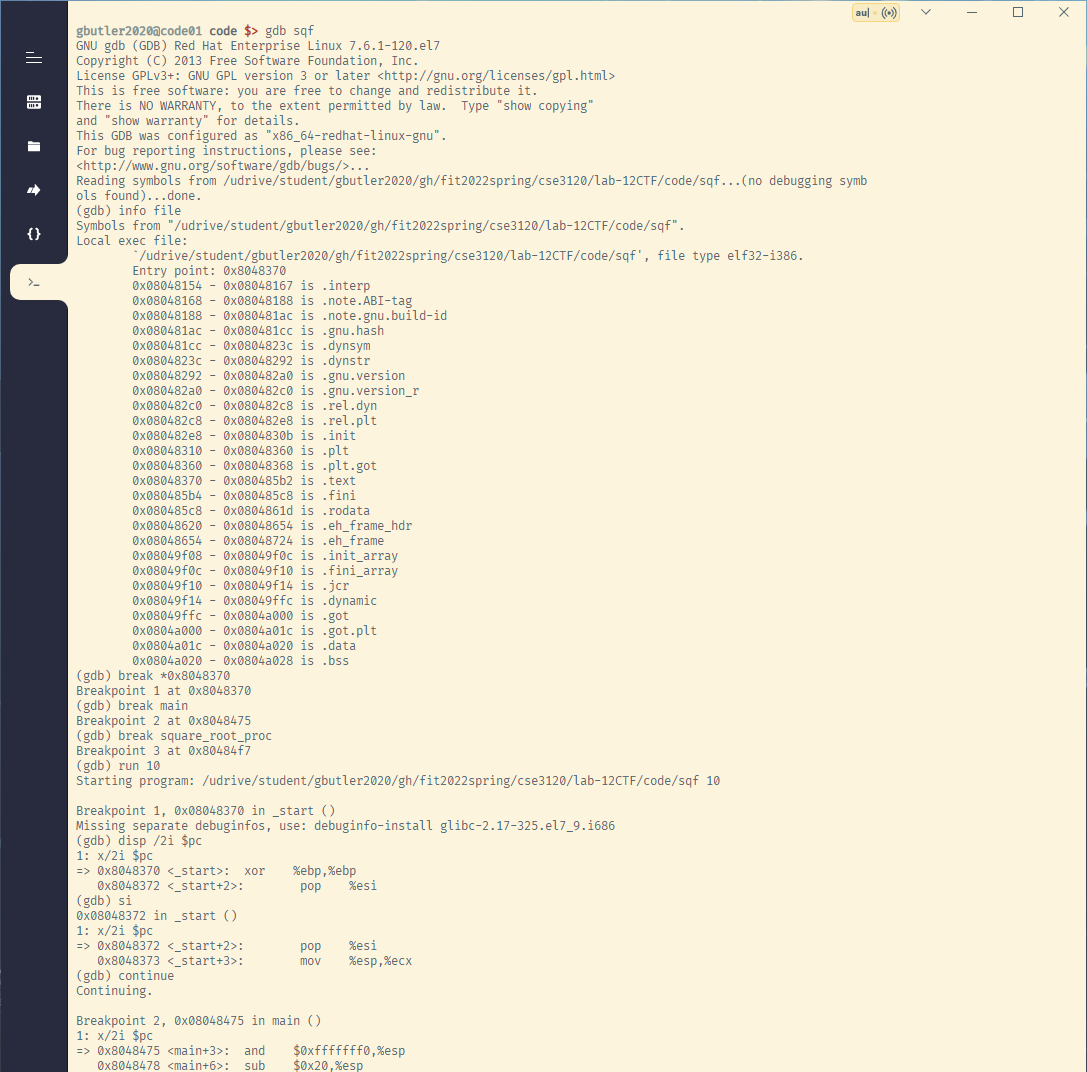
| /\* File: square\_proc.c \*/ #include <stdlib.h> #include <stdio.h> #ifdef \_\_cplusplus extern "C" #endif int square\_root\_proc(int); int square\_root\_proc\_mangled(int x){} int main(int argc, char\*\*argv) {  if (argc <= 1) {printf("Parameter absent\n"); return 0;}  int in = atoi(argv[1]);  int a = square\_root\_proc(in\*in);  if (in != a) printf("Wrong val=%d rather than %d\n", a, in);  else printf ("Flag: flag{RootOfAllFlags}\n");  return a; } |
| --- |

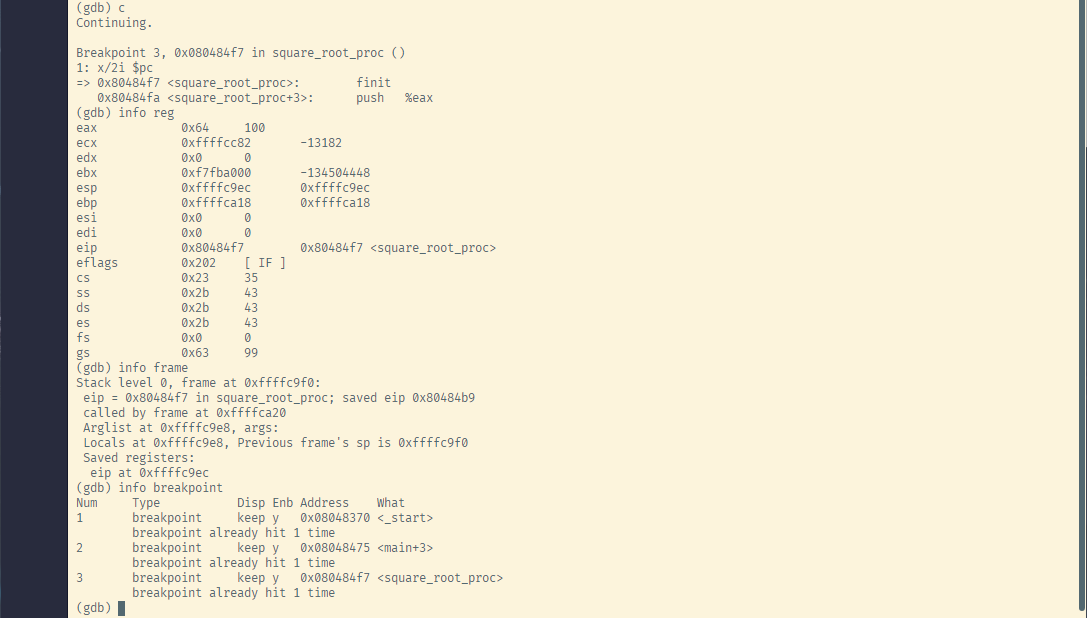
###### code01:



##### task-2:

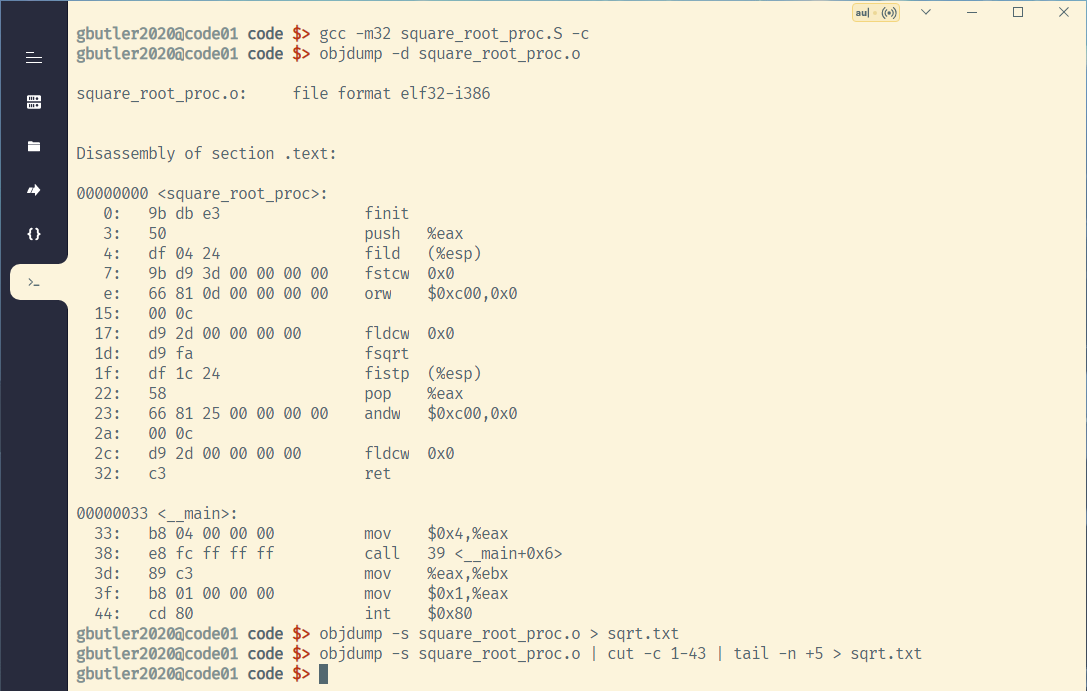
###### code01:

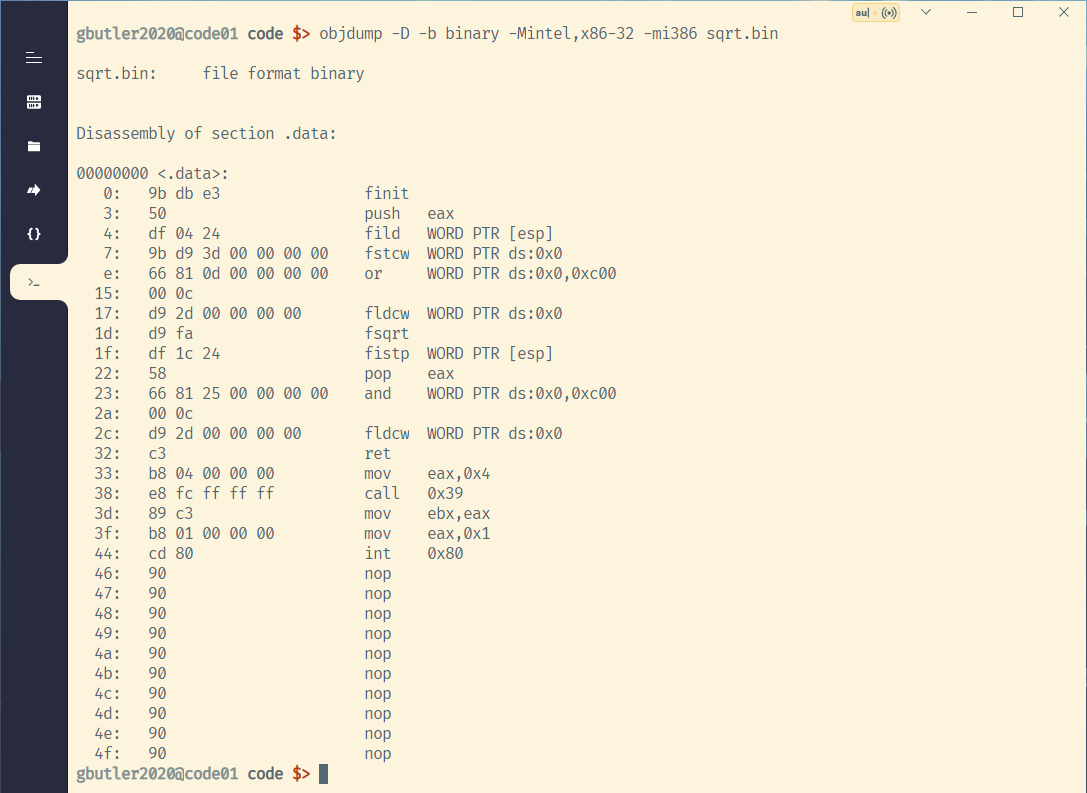


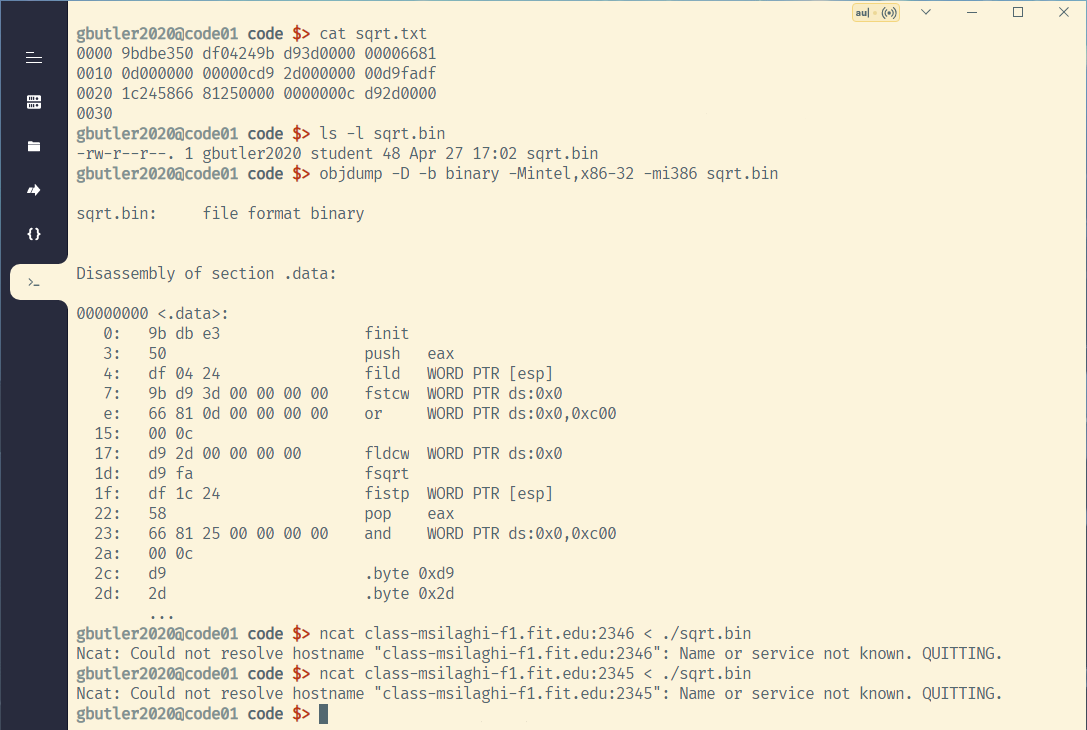


##### task-3:

###### code01:







Why the implementation at Lab12 FPU would have failed to capture the flag, while the one in step 1 can succeed?

The FPU would have failed because we couldn’t overload the FPU, but we can without FPU.