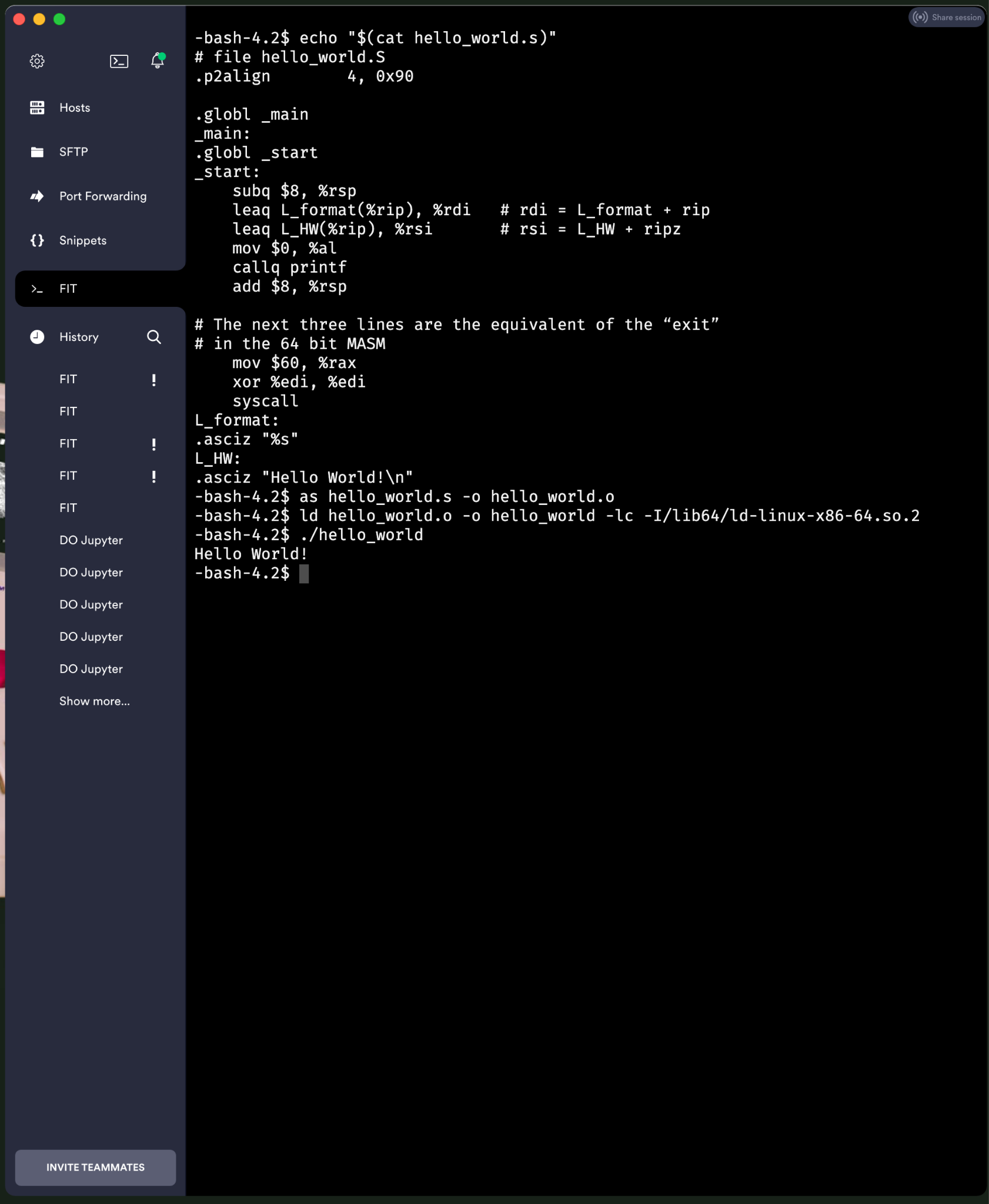
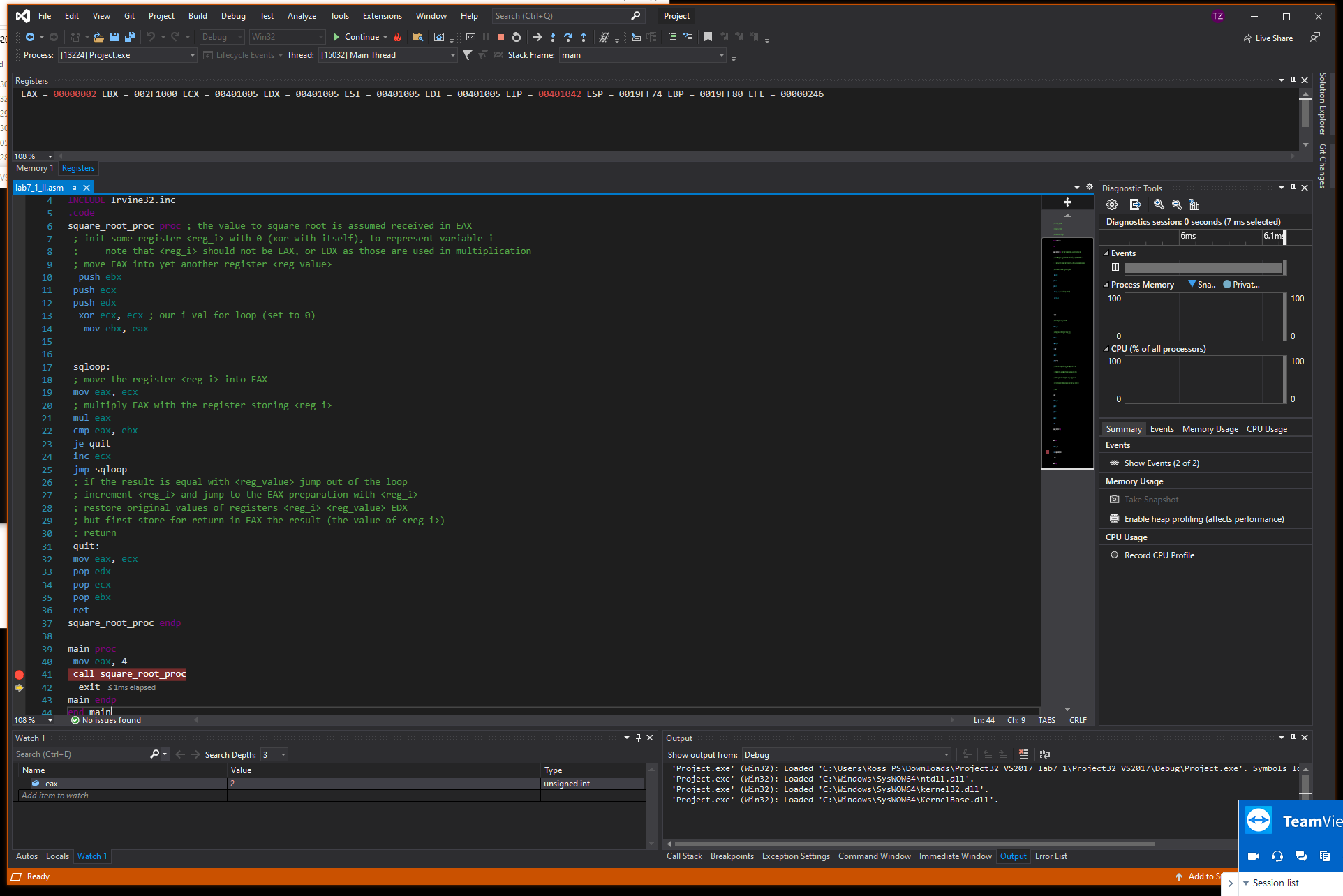
7.1-I)

| # file hello\_world.S .p2align 4, 0x90 # 16 byte alignment, filled nops  .globl \_start \_start:  subq $8, %rsp  leaq L\_format(%rip), %rdi # rdi = L\_format + rip  leaq L\_HW(%rip), %rsi # rsi = L\_HW + ripz  mov $0, %al  callq printf  add $8, %rsp  # The next three lines are the equivalent of the "exit" # in the 64 bit MASM  mov $60, %rax  xor %edi, %edi  syscall  L\_format:  .asciz "%s" L\_HW:  .asciz "Hello World!\n" |
| --- |



7.1-II)

| INCLUDE Irvine32.inc .code square\_root\_proc proc ; the value to square root is assumed received in EAX  ; init some register <reg\_i> with 0 (xor with itself), to represent variable i  ; note that <reg\_i> should not be EAX, or EDX as those are used in multiplication  ; move EAX into yet another register <reg\_value>  push ebx  push ecx  push edx  xor ecx, ecx ; our i val for loop (set to 0)  mov ebx, eax   sqloop:  ; move the register <reg\_i> into EAX  mov eax, ecx  ; multiply EAX with the register storing <reg\_i>  mul eax  cmp eax, ebx  je quit  inc ecx  jmp sqloop  ; if the result is equal with <reg\_value> jump out of the loop  ; increment <reg\_i> and jump to the EAX preparation with <reg\_i>  ; restore original values of registers <reg\_i> <reg\_value> EDX  ; but first store for return in EAX the result (the value of <reg\_i>)  ; return  quit:  mov eax, ecx  pop edx  pop ecx  pop ebx  ret square\_root\_proc endp  main proc  mov eax, 4  call square\_root\_proc  exit main endp end main |
| --- |



7.1-III)

| .code32 .text .globl main .type main, @function .globl square\_root\_proc .type square\_root\_proc, @function square\_root\_proc:  push %ebx  push %ecx  push %edx  xor %ecx, %ecx  mov %eax, %ebx   sqloop:  mov %ecx, %eax  mul %eax  cmp %ebx, %eax  je quit  inc %ecx  jmp sqloop  quit:  mov %ecx, %eax  pop %edx  pop %ecx  pop %ebx  ret   main:  mov $4, %eax  call square\_root\_proc  mov %eax, %ebx # exit code  mov $1, %eax # exit syscall  int $0x80 |
| --- |

