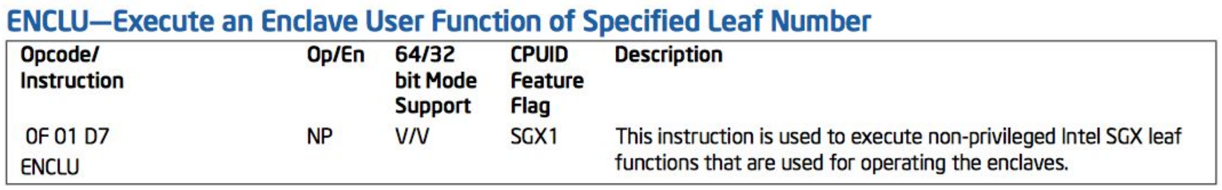
1 Lab overview

Lab 3 is a follow-up lab for Lab 2. In Lab 2, students have learnt how to enter a enclave program. In this lab, students can further learn how to pass arguments to enclave programs and how to exit from enclave programs.

Having a basic understanding of SGX and the format of Enclave instructions is still necessary to complete the lab. The SGX manual can be found in here

https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

However, reading through the entire document would be burdensome and time consuming. Here we provide several key information for you to complete the task. Specifically, in Section 5.2, you can find the opcode for the ENCLU instruction. Also, in table 5.2, you can find the register usage for each Enclave leaf functions.



ENCLU—Execute an Enclave User Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 01 D7 ENCLU	NP	V/V	SGX1	This instruction is used to execute non-privileged Intel SGX leaf functions that are used for operating the enclaves.

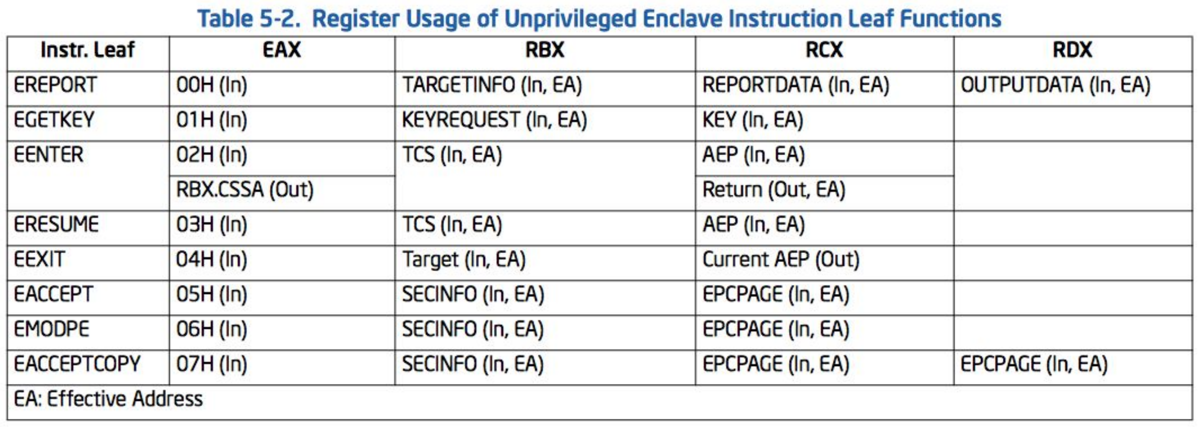


Table 5-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
EREPORT	00H (In)	TARGETINFO (In, EA)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)
EGETKEY	01H (In)	KEYREQUEST (In, EA)	KEY (In, EA)	
EENTER	02H (In)	TCS (In, EA)	AEP (In, EA)	
	RBX.CSSA (Out)		Return (Out, EA)	
ERESUME	03H (In)	TCS (In, EA)	AEP (In, EA)	
EEXIT	04H (In)	Target (In, EA)	Current AEP (Out)	
EACCEPT	05H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODPE	06H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EACCEPTCOPY	07H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	EPCPAGE (In, EA)
EA: Effective Address				

In lab 2, we have mentioned that in order to issue an EENTER instruction, you need to first load EAX register with leaf number (0x02) and load RBX register with the address of TCS data structure.

In this lab, we will continue to discuss argument passing and returning from enclave.

2 Lab Tasks

For each of the lab task, please capture the screen-shot of the key step and include it in your lab report.

Before you run the below programs, use the following command to load SGX emulator in your system, you could find it under sgx_emuluator directory. (See Lab 0 for details)

```
sudo insmod sgx.ko
```

Task 2.1 Pass one argument to enclave

We use register %RDX (or %EDX in a 32-bit setting) to pass the argument to enclave programs.

Program list 1 shows an incomplete C program. This program demonstrates how to issue a CPU instruction called EENTER to enter enclave program, and pass one argument at the same time. Replace the question mark with variable names so that the program can be compiled and work.

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
struct tcs {
        unsigned long enclave_rsp_heap;
        unsigned long p_saved_regs_e;
        bool saved;
        unsigned long res1;
        unsigned long flags;
        unsigned long ossa;
        unsigned int cssa;
        unsigned int nssa;
        unsigned long oentry; // entry point
        unsigned long res2;
        unsigned long ofsbasgx;
        unsigned long ogsbasgx;
        unsigned int fslimit;
        unsigned int gslimit;
        char res3[4024];
};

struct tcs* p_tcs;

void enclave_program() {
        int arg;
        asm volatile (
                        "mov %%edx, %0\n\t"
                        : "=r" ( arg) ::
                   );
        printf("We are now in the enclave and argument is %d,
cheers!\n",arg);
        exit(0);
}

int main() {
        p_tcs = (struct tcs *)malloc(sizeof(struct tcs));
        p_tcs->oentry = (unsigned long)enclave_program;
        int arg = 661;

        asm volatile (
                        "mov %0, %%rbx\n\t"
                        "movl %1, %%edx\n\t"
                        "movl $0x???, %%eax\n\t" // EENTER, instruction
type
                        ".byte 0x0f,0x01,0xd7\n\t" // call opcode 01d7
                        :
                        : "r" (???),"r"(???)
                        : "eax", "rbx" ,"rdx"
                   );
        return 0;
}
```

Program List 1

Task 2.2 Pass multiple arguments to Enclave

If %RDX(EDX) can only take one argument, then how can we pass multiple arguments to enclave programs, one trick here is to pack several arguments in a data structure, and pass the pointer to that data structure. Complete the below program by replacing question marks.

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
struct tcs {
        unsigned long enclave_rsp_heap;
        unsigned long p_saved_regs_e;
        bool saved;
        unsigned long res1;
        unsigned long flags;
        unsigned long ossa;
        unsigned int cssa;
        unsigned int nssa;
        unsigned long oentry; // entry point
        unsigned long res2;
        unsigned long ofsbasgx;
        unsigned long ogsbasgx;
        unsigned int fslimit;
        unsigned int gslimit;
        char res3[4024];
};

struct tcs* p_tcs;
struct enclave_args {
        int val;
        char* msg;
};


void enclave_program() {
        struct enclave_args *my_arg;
        asm volatile (
                        "mov %%rdx, %0\n\t"
                        : "=r" (my_arg) ::
                     );
        printf("We are now in the enclave and val is %d,
msg=%s,cheers!\n",my_arg->val,my_arg->msg);
      exit(0);
}

int main() {
        p_tcs = (struct tcs *)malloc(sizeof(struct tcs));
        p_tcs->oentry = (unsigned long)enclave_program;
        struct enclave_args args;
        args.val = 661;
        args.msg = "Computer Architecture!";

        asm volatile (
                        "mov %0, %%rbx\n\t"
                        "mov %1, %%rdx\n\t"
                        "movl $0x???, %%eax\n\t" // EENTER, instruction type
                        ".byte 0x0f,0x01,0xd7\n\t" // call opcode 01d7
                        :
                        : "r" (???),"r"(???)
                        : "eax", "rbx" ,"rdx"
                     );
        return 0;
}
```

Task 2.3 Returning from enclave

In order to return from Enclave, you first need to figure out the return address. In SGX, the return address can be get from %RCX register.

The next step is to issue an EEXIT instruction to exit enclave. Which is done by loading %EAX register with a constant 4 (instruction type for EEXIT) and loading %RBX with return address.

Use the above information to complete the below program by replacing question marks.

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
struct tcs {
        unsigned long enclave_rsp_heap;
        unsigned long p_saved_regs_e;
        bool saved;
        unsigned long res1;
        unsigned long flags;
        unsigned long ossa;
        unsigned int cssa;
        unsigned int nssa;
        unsigned long oentry; // entry point
        unsigned long res2;
        unsigned long ofsbasgx;
        unsigned long ogsbasgx;
        unsigned int fslimit;
        unsigned int gslimit;
        char res3[4024];
};

struct tcs* p_tcs;

void enclave_program() {
        void *return_address;
        asm volatile (
                        "mov %%???, %0\n\t"
                        : "=r" (return_address) ::
                     );
        printf("We are now in the enclave, cheers!\n");
        asm volatile ("mov %0, %%rbx \n\t"
                        "movl $0x??? , %%eax\n\t" // EEXIT
                        ".byte 0x0f,0x01,0xd7"
                        :
                        : "r" (???));
}

int main() {
        p_tcs = (struct tcs *)malloc(sizeof(struct tcs));
        p_tcs->oentry = (unsigned long)enclave_program;
        asm volatile (
                        "mov %0, %%rbx\n\t"
                        "movl $0x???, %%eax\n\t" // EENTER, instruction type
                        ".byte 0x0f,0x01,0xd7\n\t" // call opcode 01d7
                        :
                        : "r" (???)
                        : "eax", "rbx"
                  );

        printf("return from Enclave program!\n");
        return 0;
}
```