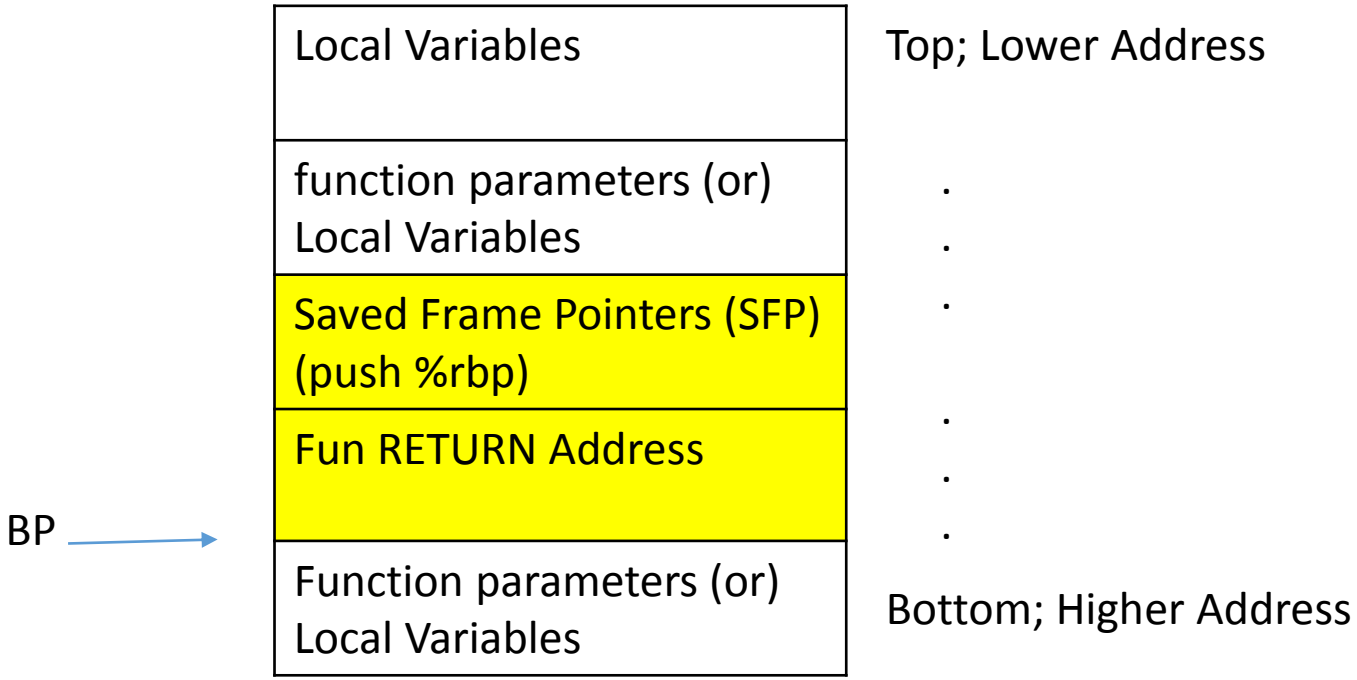
X86 Stack Frame

Lab Assignments: X86 Stack Frame

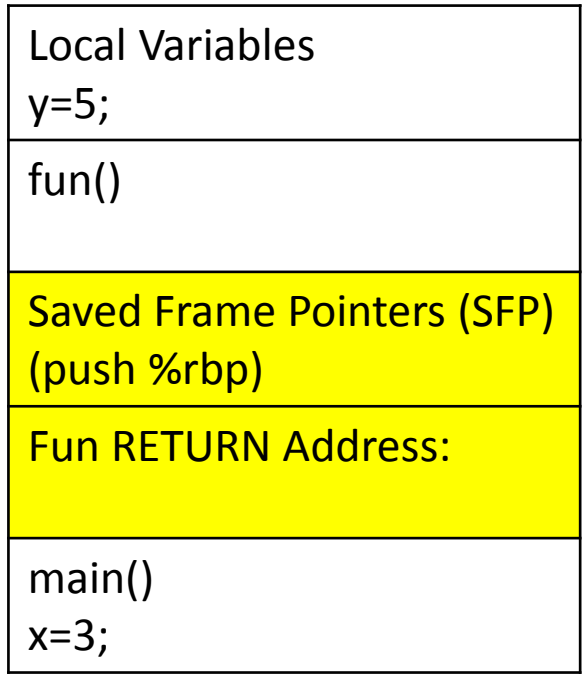
1. How call instruction works in 8085 microprocessor explain with example?
2. Draw x86 stack frame and explain?

X86 Stack Frame Layout



Example1:

Stack frame of stack1.c



Stack1.c

```
main ()
{
int x=3;
fun ();
}
fun()
{
int y=5;
}
```

Assignment 3: Prepare a stack frame output from the below program using gdb.

```
main ()
{
int x=3;
fun ();
}
fun()
{
int y=5;
}
```

Step1: Assign a breakpoint at fun()

Step2: execute run command in gdb prompt.

Step3: Print local variables y & x.

What is the output? Explain

Step4: run the below command in gdb prompt and prepare a stack frame format.

(gdb) x/16gx $rbp-32

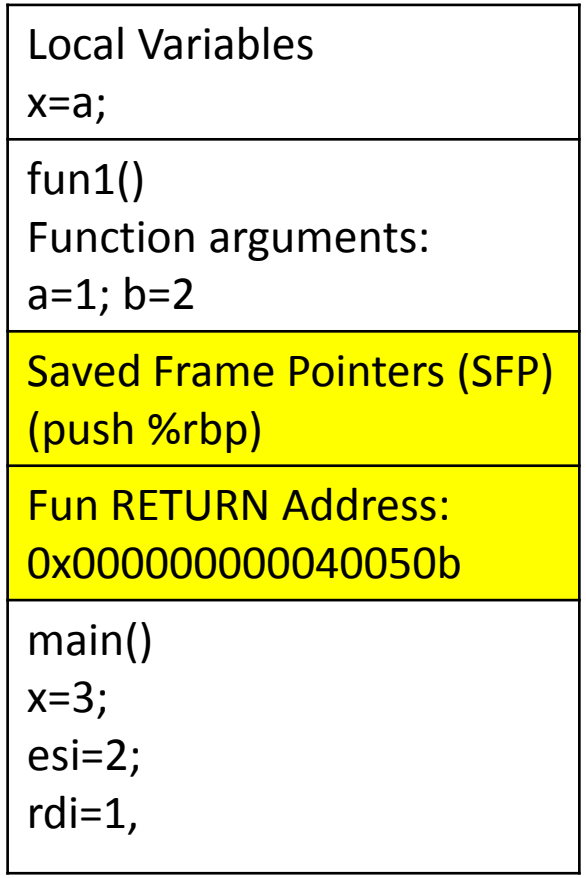
Assignment 4: WAP swap two numbers using call by value and call by reference and prepare a stack frame using gdb?

X86 Stack Frame

1. Parameters are pushed onto the stack in reverse order (because of the LIFO mechanism)
2. When the assembly language "call" instruction is issued, to change the execution context to the called function, the return address is pushed onto the stack. This will be the address of the instruction following the current EIP.
3. (Procedure Prolog): Current value of EBP (the frame pointer) is pushed onto the stack. This value is called the Saved Frame Pointer (SFP) and is later used to restore EBP back to it's original state.
4. The current value of ESP is then copied into EBP to set the new frame pointer.
5. Memory is allocated onto the stack for local (automatic) variables by subtracting from ESP. The memory allocated for these variables isn't pushed onto the stack, so the variables are in expected (same) order (as their being declared).

Example2: stack2.c

Stack2.c Stack frame



Top; Lower Address

.
.
.

.
.
.

Bottom; Higher Address

stack2.c

```
main ()
{
int x=3;
fun1 (1,2);
}
fun(int a, int b)
{
int x;
x=a;
}
```

```c
#include <stdio.h>
 void fun1 (int , int );
 main ()
 {
 int i=3;
 fun1(1, 2);
 }
void fun1 (int a, int b)
{
      int x;
      x = a;
}
```

```
(gdb) x/20gx $rbp-0x30
0x7fffffffe9d0: 0x00007ffff7ffe1c8      0x0000000000000000
0x7fffffffe9e0: 0x0000000000000001      0x0000000100000002
0x7fffffffe9f0: 0x00007fffffffea20      0x0000000100000000
0x7fffffffea00: 0x00007fffffffea20      0x000000000040050b
0x7fffffffea10: 0x00007fffffffeb00      0x0000000300000000
0x7fffffffea20: 0x0000000000000000      0x00007ffff7a36ec5
```

Lower Address

	Local Variables	fun(1,2)
	Local Variables	X=a;
	SFP	Save Main() rbp
	Return Address	Main() return
	Main() Local	i=3

Higher Address

```
(gdb) disass
Dump of assembler code for function main:
   0x00000000004004ed <+0>:     push   %rbp
   0x00000000004004ee <+1>:     mov    %rsp,%rbp
   0x00000000004004f1 <+4>:     sub    $0x10,%rsp
=> 0x00000000004004f5 <+8>:     movl   $0x3,-0x4(%rbp)
   0x00000000004004fc <+15>:    mov    $0x2,%esi
   0x0000000000400501 <+20>:    mov    $0x1,%edi
   0x0000000000400506 <+25>:    callq  0x40050d <fun1>
   0x000000000040050b <+30>:    leaveq
   0x000000000040050c <+31>:    retq
End of assembler dump.
(gdb) p $rbp
0x7fffffffea20
(gdb) p $rsp
0x7fffffffea10
```

```
(gdb) disass
Dump of assembler code for function fun1:
   0x000000000040050d <+0>:     push   %rbp
   0x000000000040050e <+1>:     mov    %rsp,%rbp
   0x0000000000400511 <+4>:     mov    %edi,-0x14(%rbp)
   0x0000000000400514 <+7>:     mov    %esi,-0x18(%rbp)
   0x0000000000400517 <+10>:    mov    -0x14(%rbp),%eax
   0x000000000040051a <+13>:    mov    %eax,-0x4(%rbp)
=> 0x000000000040051d <+16>:    pop    %rbp
   0x000000000040051e <+17>:    retq
End of assembler dump.
(gdb) p $rbp
0x7fffffffea00
(gdb) p $rsp
0x7fffffffea00
```