LEA Instruction

The OFFSET directive in assembly language allows you to get the address of a variable or label at compile time. However, **it does not work with stack parameters** because the addresses of stack parameters are not known until runtime.

The following statement would not assemble:

```
mov esi, OFFSET [ebp-30]
```

This is because the **compiler does not know the value of ebp at compile time.** ebp is the base pointer register, and it points to the top of the stack frame.

The offset of the local variable myString from the base pointer is -30, but the value of the base pointer is not known until runtime.

```
404 void makeArray( )
405 {
406     char myString[30];
407     for( int i = 0; i < 30; i++ )
408     myString[i] = '*';
409 }
```

The code then enters a for loop that iterates from i = 0 to i = 29. In each iteration, it assigns the character '*' to the i-th element of the myString array.

Effectively, this code initializes all 30 elements of the myString array to the character '*'. After the function is called, the myString array will contain 30 asterisk characters, like this:

The **LEA instruction, on the other hand, can be used to calculate the address of a stack parameter at runtime.** The LEA instruction takes a memory operand as its operand and loads the effective address of the operand into the destination register.

The following assembly language code is equivalent to the C++ code in

the example:

```
386 makeArray PROC
387     push ebp
388     mov ebp, esp
389     sub esp, 32 ; myString is at EBP-30
390     lea esi, [ebp-30] ; load address of myString
391     mov ecx, 30 ; loop counter
392     L1:
393     mov BYTE PTR [esi], '*' ; fill one position
394     inc esi ; move to next
395     loop L1 ; continue until ECX = 0
396     add esp, 32 ; remove the array (restore ESP)
397     pop ebp
398     ret
399 makeArray ENDP
```

The LEA instruction calculates the effective address of the operand [ebp-30] and loads it into the register esi. The operand [ebp-30] references the local variable myString because myString is located 30 bytes below the base pointer register.

Once you have loaded the address of the stack parameter into a register, you can use the register to access the stack parameter. For example, the following assembly language code shows how to use the register esi to access the local variable myString:

```
mov BYTE PTR [esi], '*' ; fill one position
```

This code stores the character '*' in the first byte of the local variable myString.

The LEA instruction is a powerful tool that can be used to calculate the addresses of memory locations at runtime. It is especially useful for working with stack parameters and dynamic data structures.