Reference Parameters

Reference parameters are passed to a procedure by address.

This means that the procedure receives a pointer to the actual variable, instead of a copy of the variable's value.

This allows the procedure to modify the value of the variable in the caller's scope.

To access a reference parameter, the procedure can use base-offset addressing with the EBP register.

The EBP register points to the base of the current stack frame.

Each reference parameter is pushed onto the stack in reverse order, so the offset of the first reference parameter is 12 bytes from EBP.

For example, the following code loads the pointer to the array passed to the ArrayFill procedure into the ESI register:

```
mov esi, [ebp+12]
```

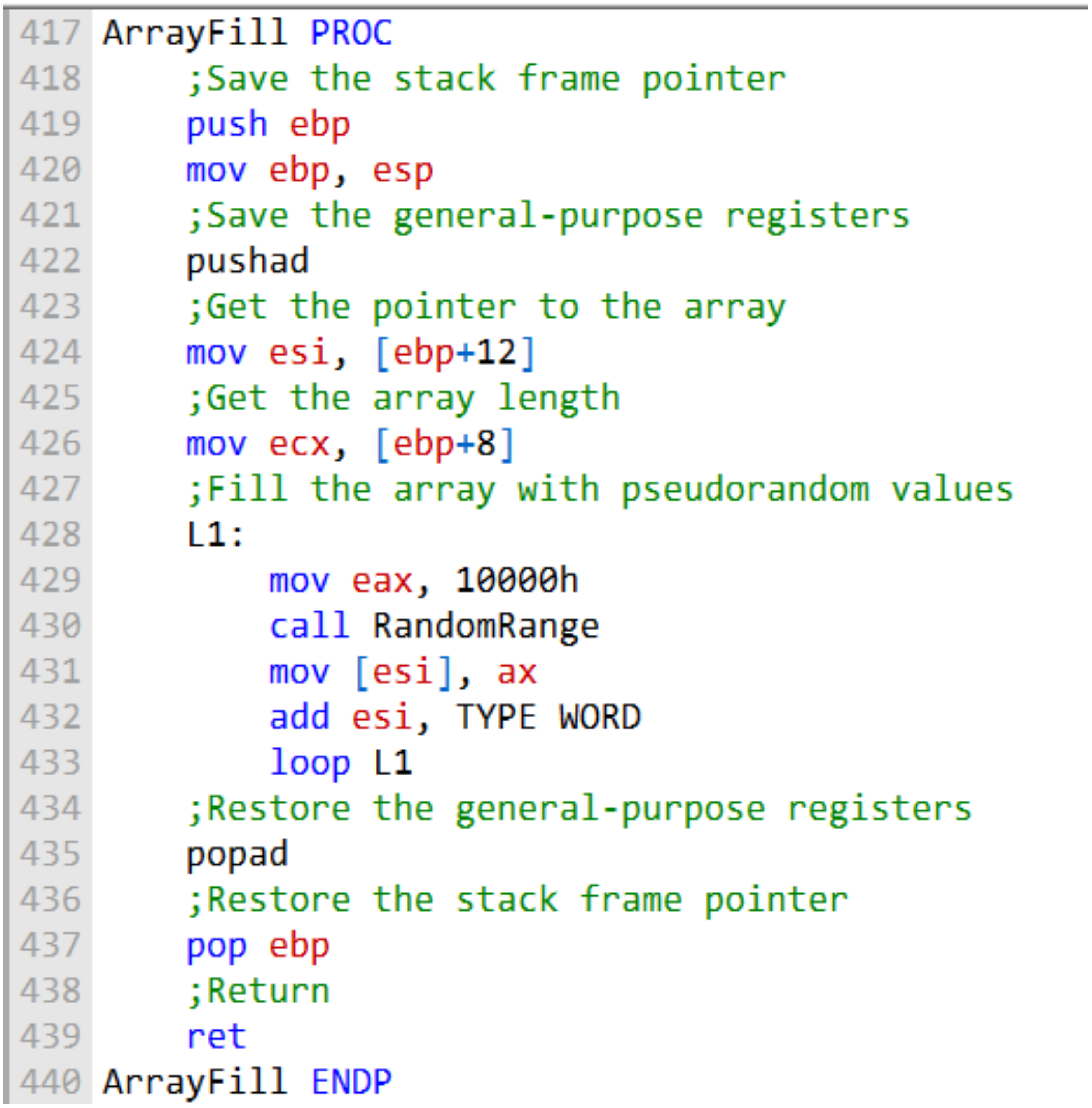
The ESI register can then be used to access the elements of the array.

The ArrayFill procedure fills an array with a pseudorandom sequence of 16-bit integers.

It receives two arguments: a pointer to the array and the array length.

The first argument is passed by reference and the second argument is passed by value.

Here is a simplified version of the ArrayFill procedure:



```
ArrayFill PROC
    ;Save the stack frame pointer
    push ebp
    mov ebp, esp
    ;Save the general-purpose registers
    pushad
    ;Get the pointer to the array
    mov esi, [ebp+12]
    ;Get the array length
    mov ecx, [ebp+8]
    ;Fill the array with pseudorandom values
    L1:
        mov eax, 10000h
        call RandomRange
        mov [esi], ax
        add esi, TYPE WORD
        loop L1
    ;Restore the general-purpose registers
    popad
    ;Restore the stack frame pointer
    pop ebp
    ;Return
    ret
ArrayFill ENDP
```

The first few lines of the procedure save the stack frame pointer and the general-purpose registers. Then, the procedure gets the pointer to the array and the array length from the stack.

The procedure then loops through the array and fills each element with a pseudorandom value using the RandomRange function. The RandomRange function is a library function that generates a random number between 0 and FFFFh.

After the loop, the procedure restores the general-purpose registers and the stack frame pointer, and then returns.

The ArrayFill procedure is written in assembly language and serves the purpose of filling an array with pseudorandom values. Here's a detailed breakdown of how it works:

Procedure Declaration:
The ArrayFill PROC statement indicates the beginning of the procedure.

Prologue - Saving the Stack Frame:
push ebp: This instruction pushes the current stack frame pointer onto the stack. mov ebp, esp: Here, the current stack pointer (esp) is copied into the base pointer (ebp). This step is essential for setting up a new stack frame for the function.

Saving General-Purpose Registers:
pushad: The pushad instruction is used to save the values of all general-purpose registers (EAX, ECX, EDX, EBX, ESI, EDI, and EBP) on the stack. This is done to preserve the state of these registers during the execution of the procedure.

Getting Array Pointer and Length:
mov esi, [ebp+12]: This line loads the address of the array into the esi register. mov ecx, [ebp+8]: The value of the array length is loaded into the ecx register. These values are passed as parameters to the function, with [ebp+12] representing the array pointer and [ebp+8] representing the array length. Array Filling Loop: The labeled loop, L1, is the core of the procedure.

In each iteration:
mov eax, 10000h: The eax register is loaded with the value 10000h (40960 in decimal). call RandomRange: This likely calls a function named RandomRange to generate pseudorandom values. mov [esi], ax: The result of the RandomRange call is stored in the memory location pointed to by esi. add esi, TYPE WORD: The esi register is incremented by the size of a word, effectively pointing to the next element in the array.

Loop L1:
This checks if the loop counter (ecx) is not zero and decrements it. If it's not zero, the code jumps back to L1, continuing the array filling process.

Epilogue - Restoring Registers and Exiting:

popad: This instruction restores the values of the general-purpose registers to their original state. pop ebp: It pops the stack frame pointer (ebp) to restore the previous stack frame.

ret: Finally, the ret instruction is used to return from the procedure, effectively exiting it. In summary, the ArrayFill procedure follows a standard structure: it saves the current state, sets up a loop to fill the array with pseudorandom values, and then restores the saved state before exiting. It's a crucial part of the code for filling an array with random data.

Conclusion

Reference parameters are a powerful feature of assembly language that allow procedures to modify the values of variables in the caller's scope. By understanding how to use reference parameters, you can write more efficient and reusable assembly code.