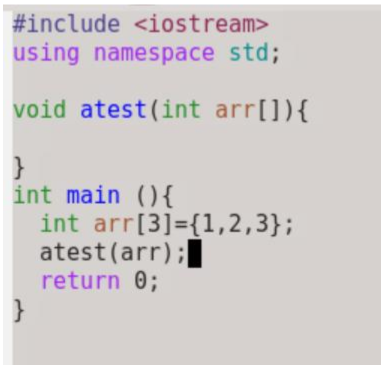
Phillip Seaton (pbs5kx)

Inlab8.pdf 3/30/16

Lab 106

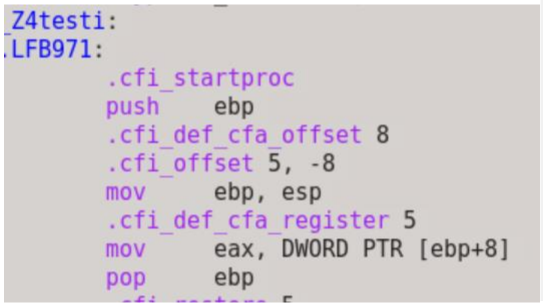
For all the different programs with different data types I made a simple program that declared variables and used them as parameters for a function the sample structure is below. The main is the caller which calls the function with its declared variable as parameters. The function is the callee which is called upon by the caller or main.



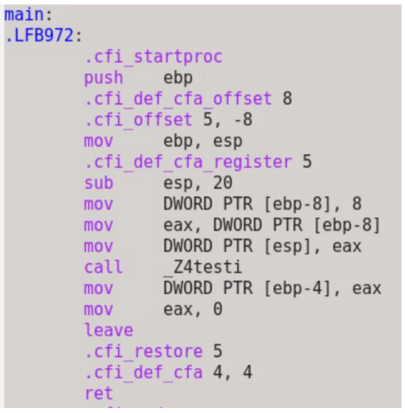
```
#include <iostream>
using namespace std;

void atest(int arr[]){

}
int main (){
  int arr[3]={1,2,3};
  atest(arr);
  return 0;
}
```

Int:

For my generic int program, the function or callee has a typical prologue and then moves the int into eax and then pops the base pointer. This is the code for pass by value. In pass by reference, the line of code is used (move   eax, DWORD PTR [eax]). Since it is pass by reference it has to load the value from memory into eax using the address that was originally stored in eax. The size is 4 bytes from DWORD.



```
_Z4testi:
.LFB971:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        mov     eax, DWORD PTR [ebp+8]
        pop     ebp
```



```
main:
.LFB972:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 20
        mov     DWORD PTR [ebp-8], 8
        mov     eax, DWORD PTR [ebp-8]
        mov     DWORD PTR [esp], eax
        call    _Z4testi
        mov     DWORD PTR [ebp-4], eax
        mov     eax, 0
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
```

There was no difference in the main or caller between pass by value and reference for int. It just declared variable and moved it to the stack and then called the function.

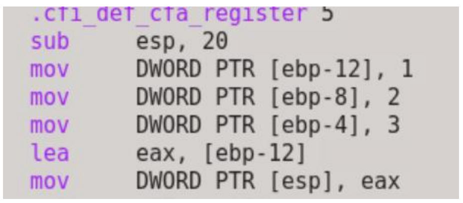
Char: The char was similar to the int. The two main/caller pieces of the program were no different between pass by reference and pass by value. The size in memory is only one byte compared to the 4 bytes from int. Similar to int, it has (move   eax, DWORD PTR [eax]). Using the address that was originally stored in eax It loads the value from memory into eax using the address that was originally stored in eax. This is in the callee.

Float: The float was very similar to int. It also had the line (move   eax, DWORD PTR [eax]). This was again to access the value that is stored in memory instead of just accessing the value in pass by value. The float is also 4 bytes because DWORD is used in front of PTR. There was only one line of difference in the callee. It also has .LC1 after DWORD PTR.

Pointer: For a pointer, pass by value I used   "void test(int * x)" for the signature of the function. For pass by reference I used "void test(int **x)" and had "test(&x)" when calling it in the main. These both game just about the same assembly. This is to show that there are barely any differences between pointers and references other than the 3 restrictions of references. Assembly does not need much more code to do a reference of a pointer.

Object: For the object callee in pass by value, the caller will pass the actual object with the fields taking up more memory. While in pass by reference, the callee is given the base address and the memory next to the base address is the fields. This is the reason that objects should be passed by reference because it saves a great deal of space for high memory object that need to be used multiple times. The size of a parameter that is passed by value will generally be much larger than the size of the address which is pass by reference.

2. Array: This is the assembly of an array being created in the main. The array is {1,2,3}.



```
	.cfi_def_cfa_register 5
	sub	esp, 20
	mov	DWORD PTR [ebp-12], 1
	mov	DWORD PTR [ebp-8], 2
	mov	DWORD PTR [ebp-4], 3
	lea	eax, [ebp-12]
	mov	DWORD PTR [esp], eax
```

The base address is passed as the parameter for an array. For my case this is [ebp-12]. This is the first element of the array. There are two more elements and the last element is at [ebp-4]. The function is able to access the rest of the array by [base+4i] where I is the index. They are the bytes after the base address. The function will have the base address relative to the ebp. All that is passed to the function is the base address.

3. Passing by value in assembly works by sending the base address of the object or primitive type. For an array or object, the elements or fields will be after the base address. For an array, the first element is the base address. The address is 4 bytes, which is little memory compared to passing a large object by value. Passing by reference is just sending an address that assigned to a spot in memory. Then the function is able to access that by [base address]. A passing by reference and passing by pointer are essentially the same thing. The pass by reference uses a

lea instruction, which loads the effective address where the pointer already has the address stored. In assembly they are very similar. The main differences arise in the compiler.