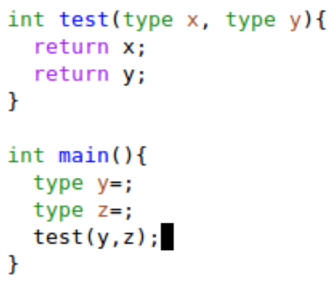
//Robert Alexander, RMA3MZ

In-Lab 8

Parameter Passing



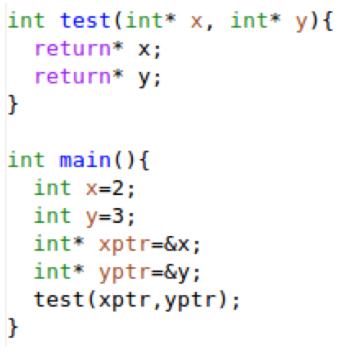
```
int test(type x, type y){
  return x;
  return y;
}

int main(){
  type y=;
  type z=;
  test(y,z);
}
```

1. Int – By value: I just created a simple c++ function that set variables x and y and called a function that then returned them (y=5, z=6). I observed the following from this, the caller sets up variables (loads into the registers):
   - Mov DWORD PTR [EBP -4], 0
   - Mov DWORD PTR [EBP -8], 5
   - Mov DWORD PTR [EBP-12], 6
     i. It then moves them into the correct registers to call the callee…

   The Callee performs functions using these set variables (int test):
   - Mov EAX, DWORD PTR [ESP +16]
   - Mov ECX, DWORD PTR [ESP +12]
   - Mov DWORD PTR [ESP+4], ECX
   - Mov EAX, DWORD PTR [ESP +4]
     i. It then continues moving things into eax and returns the result
2. Int – Pass by reference: Caller is different since you are now passing the memory location, not the actual value, callee stays the same because the caller will still pass the correct information to the callee. New Caller:
   - Lea EAX, DWORD PTR [EBP -8]
   - Lea ECX, DWORD PTR [EBP -12]
     i. These lines load the effective addresses.
     ii. The rest is the same
3. Char- By value: I used the same format as obove but replaced int with chars (y='y', z='z') caller:
   - Mov DWORD PTR [EBP -4], 0
   - Mov BYTE PTR [EBP -5], 121
   - Mov BYTE PTR [EBP-6], 122
     i. This makes sense, since an ascii char takes up a single byte, so it would use BYTE instead of the DWORD, also it only needs one byte between the local variables. I am assuming the 121 and 122 are the ascii values.

   Callee:
   - Mov AL, BYTE PTR [ESP +10] //only uses part of the register
   - Mov CL, BYTE PTR [ESP +6]
   - Then moves cl into esp +1 and al into esp, and moves the correct information into eax and then returns the values.
4. Char- By reference:
   - Changing char to reference causes the same changes that occurred when using an int, the addition of lea to make sense of the addresses that are being passed.

//Robert Alexander, RMA3MZ



```
int test(int* x, int* y){
  return* x;
  return* y;
}

int main(){
  int x=2;
  int y=3;
  int* xptr=&x;
  int* yptr=&y;
  test(xptr,yptr);
}
```

5. Pointer by value: I changed my code to pass pointers into the test function. The results in the caller were:
    - lea EAX, DWORD PTR [EBP -12]
    - lea ECX, DWORD PTR [EBP -8]
        i. These were the only two additional lines that differed from the caller when using ints, this is expected since, it just needs to handle the memory addresses that the pointer passes.

   Callee:
    - Nothing changed here
6. Floats by value: Changing the varialbes to floats (x=10.5 and y=11.7) really started to alter the look of the assembly. Notes on both caller and callee:
    - It now used the movss command as opposed to the mov command used for all of the others, which I discovered is a special move command for floating point numbers. It still allocated 4 bytes for each number, and beyond the change in the mov command, it was fairly similar.
7. Floats by reference: Passing a float by reference seems to be fairly similar to an int by reference. The major difference is the addition of the lea in the caller and the additional move commands that are required to move the values into the correct locations for the callee.
8. Object: When passing an object, the implementation of the object alters the resulting structure of the assembly code. For example, it will perform similarly if you pass an object that holds two ints.