Liya Xu

lx2hy

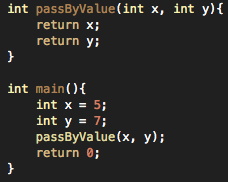
03/26/2014

Section 103

**Parameter Passing**

* Int pass by value:

I wrote a small C++ program that contained a function taking integer value x and y and then return them.



Then I generated the assembly code, and found out that the caller code is:

*push EBP*

*mov EBP, ESP*

*sub ESP, 24*

*mov DWORD PTR [EBP - 4], 0*

*mov DWORD PTR [EBP - 8], 5*

*mov DWORD PTR [EBP - 12], 7*

*mov EAX, DWORD PTR [EBP - 8]*

*mov ECX, DWORD PTR [EBP - 12]*

*mov DWORD PTR [ESP], EAX*

*mov DWORD PTR [ESP + 4], ECX*

*call \_Z11passByValueii*

*mov ECX, 0*

*mov DWORD PTR [EBP - 16], EAX # 4-byte Spill*

*mov EAX, ECX*

*add ESP, 24*

*pop EBP*

*ret*

it directly load the variables to the registers.

The callee code is:

*sub ESP, 8*

*mov EAX, DWORD PTR [ESP + 16]*

*mov ECX, DWORD PTR [ESP + 12]*

*mov DWORD PTR [ESP + 4], ECX*

*mov DWORD PTR [ESP], EAX*

*mov EAX, DWORD PTR [ESP + 4]*

*add ESP, 8*

*ret*

* Int pass by reference: The codes are all the same except that there are two more lines of code in the forth and fifth line:

*lea EAX, DWORD PTR [EBP - 8]*

*lea ECX, DWORD PTR [EBP - 12]*

Since the user is passing the memory location, these two lines serve to load the addresses of the value.

* Char pass by value: I changed the C++ function return type to char, and set x and y to chars. Then I generated the assembly code.

Caller code:

*push EBP*

*mov EBP, ESP*

*sub ESP, 24*

*mov DWORD PTR [EBP - 4], 0*

*mov BYTE PTR [EBP - 5], 97*

*mov BYTE PTR [EBP - 6], 98*

*mov AL, BYTE PTR [EBP - 5]*

*movsx ECX, AL*

*mov DWORD PTR [ESP], ECX*

*movsx ECX, BYTE PTR [EBP - 6]*

*mov DWORD PTR [ESP + 4], ECX*

*call \_Z11passByValuecc*

*mov ECX, 0*

*mov BYTE PTR [EBP - 7], AL # 1-byte Spill*

*mov EAX, ECX*

*add ESP, 24*

*pop EBP*

*ret*

In the underlined part, 97 and 98 are ascii values of char a and b which I set for x and y. It also uses BYTE instead of DWORD, since a char take up a byte.

Callee code:

*sub ESP, 2*

*mov AL, BYTE PTR [ESP + 10]*

*mov CL, BYTE PTR [ESP + 6]*

*mov BYTE PTR [ESP + 1], CL*

*mov BYTE PTR [ESP], AL*

*movsx EAX, BYTE PTR [ESP + 1]*

*add ESP, 2*

*ret*

* Char pass by reference: There are two more lines of code in the fourth and fifth line

*lea EAX, DWORD PTR [EBP - 5]*

*lea ECX, DWORD PTR [EBP - 6]*

It works the same way as it is in int pass by reference – load the effective addresses of the char value.

* Pointer by value: I changed the function to take in pointers and return them. Caller code:

*push EBP*

*mov EBP, ESP*

*sub ESP, 40*

*lea EAX, DWORD PTR [EBP - 12]*

*lea ECX, DWORD PTR [EBP - 8]*

*mov DWORD PTR [EBP - 4], 0*

*mov DWORD PTR [EBP - 8], 5*

*mov DWORD PTR [EBP - 12], 7*

*mov DWORD PTR [EBP - 16], ECX*

*mov DWORD PTR [EBP - 20], EAX*

*mov EAX, DWORD PTR [EBP - 16]*

*mov ECX, DWORD PTR [EBP - 20]*

*mov DWORD PTR [ESP], EAX*

*mov DWORD PTR [ESP + 4], ECX*

*call \_Z11passByValuePiS\_*

*mov ECX, 0*

*mov BYTE PTR [EBP - 21], AL # 1-byte Spill*

*mov EAX, ECX*

*add ESP, 40*

*pop EBP*

*ret*

In the underlined code, it handled the addresses differently than the caller code of int pass by reference, since it is passed by the pointer.

Callee:

*sub ESP, 8*

*mov EAX, DWORD PTR [ESP + 16]*

*mov ECX, DWORD PTR [ESP + 12]*

*mov DWORD PTR [ESP + 4], ECX*

*mov DWORD PTR [ESP], EAX*

*mov EAX, DWORD PTR [ESP + 4]*

*mov EAX, DWORD PTR [EAX]*

*mov DL, AL*

*movsx EAX, DL*

*add ESP, 8*

*ret*

* Float pass by value:

Caller code:

*push EBP*

*mov EBP, ESP*

*sub ESP, 40*

*movss XMM0, DWORD PTR [.LCPI3\_0]*

*movss XMM1, DWORD PTR [.LCPI3\_1]*

*mov DWORD PTR [EBP - 4], 0*

*movss DWORD PTR [EBP - 8], XMM1*

*movss DWORD PTR [EBP - 12], XMM0*

*movss XMM0, DWORD PTR [EBP - 8]*

*movss XMM1, DWORD PTR [EBP - 12]*

*movss DWORD PTR [ESP], XMM0*

*movss DWORD PTR [ESP + 4], XMM1*

*call \_Z11passByValueff*

*fstp DWORD PTR [EBP - 16]*

*movss XMM0, DWORD PTR [EBP - 16]*

*mov EAX, 0*

*movss DWORD PTR [EBP - 20], XMM0 # 4-byte Spill*

*add ESP, 40*

*pop EBP*

*ret*

Callee code:

*sub ESP, 12*

*movss XMM0, DWORD PTR [ESP + 20]*

*movss XMM1, DWORD PTR [ESP + 16]*

*movss DWORD PTR [ESP + 8], XMM1*

*movss DWORD PTR [ESP + 4], XMM0*

*movss XMM0, DWORD PTR [ESP + 8]*

*movss DWORD PTR [ESP], XMM0*

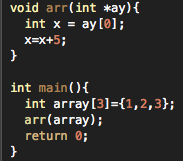
*fld DWORD PTR [ESP]*

*add ESP, 12*

*ret*

It changes the “mov” command to “movss”, which is a move command for floating point numbers.

* Float pass by reference: it is similar to int by reference except the movss command.
* Object pass: the structure will change accordingly to the type of object users pass in. If you pass objects that hold chars, it will have similar codes as the char passing.
* Passing Arrays

I wrote a simple function to show how arrays are passed. 

Accordingly, the assembly codes are:

Caller code:

*push EBP*

*mov EBP, ESP*

*sub ESP, 24*

*mov DWORD PTR [EBP - 4], 0*

*mov EAX, .L\_ZZ4mainE5array*

*mov DWORD PTR [EBP - 16], EAX*

*mov EAX, .L\_ZZ4mainE5array+4*

*mov DWORD PTR [EBP - 12], EAX*

*mov EAX, .L\_ZZ4mainE5array+8*

*mov DWORD PTR [EBP - 8], EAX*

*lea EAX, DWORD PTR [EBP - 16]*

*mov DWORD PTR [ESP], EAX*

*call \_Z3arrPi*

*mov EAX, 0*

*add ESP, 24*

*pop EBP*

*ret*

Callee code:

*sub ESP, 8*

*mov EAX, DWORD PTR [ESP + 12]*

*mov DWORD PTR [ESP + 4], EAX*

*mov EAX, DWORD PTR [ESP + 4]*

*mov EAX, DWORD PTR [EAX]*

*mov DWORD PTR [ESP], EAX*

*mov EAX, DWORD PTR [ESP]*

*add EAX, 5*

*mov DWORD PTR [ESP], EAX*

*add ESP, 8*

*ret*