Riya Baphna
Rb4nk
3/28/19
postlab.pdf

Parameter Passing

1. How are variables (ints, chars, pointers, floats, etc.) passed by value? How are they passed by reference? Create several functions and examine the parameter registers to help you answer this question.

   Variables passed by value are actually placed on the stack and mainly rely on the mov instruction. However, when passed by reference, the instruction lea was used, which was copying the variables addresses or arguments instead of the actual variable.

```
// Type your code here, or load an example.
int square(int num) {
    num++;
}
int square2(int &num) {
    num++;
}
```

```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        add     DWORD PTR [rbp-4], 1
        nop
        pop     rbp
        ret
square2(int&):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rax]
        lea     edx, [rax+1]
        mov     rax, QWORD PTR [rbp-8]
        mov     DWORD PTR [rax], edx
        nop
        pop     rbp
        ret
```

2. Create a simple function that takes in an object. How are objects passed by value? How are they passed by reference? Specifically, what is contained in the parameter registers in each case?

```
#include <stack>
#include <cstdlib>
#include <iostream>

using namespace std;

void pop(stack<int>s) {
    s.pop();

}
void square2(stack<int> &s) {
    s.pop();
}
int main(){
    stack<int> s;
    s.push(2);
    return 0;

}
```

```
        pop     rbp
        ret
pop(std::stack<int, std::deque<int, std::allocator<int> > >):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    std::stack<int, std::deque<int, std::allocator<int>
        nop
        leave
        ret
square2(std::stack<int, std::deque<int, std::allocator<int> > >&):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    std::stack<int, std::deque<int, std::allocator<int>
        nop
        leave
        ret
std::stack<int, std::deque<int, std::allocator<int> > >::~stack() [
        push    rbp
        mov     rbp, rsp
```

They seem to be treated by the assembly code similarly, showing that they are passed in the same way. Each parameter register consists of push, move sub and mov and uses the same registers to do so.

3. Create an array in your main method, and write a function that takes it in as a parameter. How are arrays passed into functions? How does the callee access the parameters? Where are the data values placed? Hint: you will need to determine at least a register-relative address

```
#include <array>
#include <cstdlib>
#include <iostream>

using namespace std;

void copy(int a[]) {
    int copy[5];
    copy[2]=a[2];


}
int main(){
    int a[5] = {1,2,3,4,5};
    copy(a);
    return 0;


}
```

```
copy(int*):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-40], rdi
        mov     rax, QWORD PTR [rbp-40]
        mov     eax, DWORD PTR [rax+8]
        mov     DWORD PTR [rbp-24], eax
        nop
        pop     rbp
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     DWORD PTR [rbp-32], 1
        mov     DWORD PTR [rbp-28], 2
        mov     DWORD PTR [rbp-24], 3
        mov     DWORD PTR [rbp-20], 4
        mov     DWORD PTR [rbp-16], 5
        lea     rax, [rbp-32]
        mov     rdi, rax
        call    copy(int*)
        mov     eax, 0
        leave
        ret
__static_initialization_and_destruction_0(int, int):
        push    rbp
        mov     rbp, rsp
```

They are passed in as pointers. The callee accesses them as a pointer and the data values are placed using the mov register. The mov register is relative to the rbp

register, and it makes space in the pointer (s shown by DMORD PTR) and then moving each value into the pointer.

Is passing values by reference different than passing by pointer? If they are the same, what exactly is passed in the parameter register? If they are different, how so?

```
#include <array>
#include <cstdlib>
#include <iostream>

using namespace std;

int sq(int *num) {
    num++;



}
int square2(int &num) {
   num++;
}

int main(){
    int a =5;
    int *num;
    num = &a;
    sq(num);
    square2(a);
}
```

```
sq(int*):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        add     QWORD PTR [rbp-8], 4
        nop
        pop     rbp
        ret
square2(int&):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rax]
        lea     edx, [rax+1]
        mov     rax, QWORD PTR [rbp-8]
        mov     DWORD PTR [rax], edx
        nop
        pop     rbp
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-12], 5
        lea     rax, [rbp-12]
        mov     QWORD PTR [rbp-8], rax
```

Output (0/8) x86-64 gcc 8.3 - 1057ms (1171398)

Pass by value seems to be the same as passing by pointer. The parameter register consists of instructions push, mov and mov using the registers rbp and rsp. Therefore, most callee and caller instructions were similar as the caller pushes the value onto the stack, and the callee pops it off the stuck for manipulation.

Objects

```
#include <stack>
#include <cstdlib>
#include <iostream>

using namespace std;
class Square{
    public:
    int x;
    int y;

    int getArea(){
        return x*y;
    }

    Square(int l, int w){
        l = x;
        w = y;


    }

    int com(){
    Square S = Square(5,4);
    return S.getArea();
    return getArea();
}



};
```

```
__static_initialization_and_destruction_0(int, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        mov     DWORD PTR [rbp-8], esi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L3
        cmp     DWORD PTR [rbp-8], 65535
        jne     .L3
        mov     edi, OFFSET FLAT:_ZStL8__ioinit
        call    std::ios_base::Init::Init() [complete object constr
        mov     edx, OFFSET FLAT:__dso_handle
        mov     esi, OFFSET FLAT:_ZStL8__ioinit
        mov     edi, OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
        call    __cxa_atexit
.L3:
        nop
        leave
        ret
_GLOBAL__sub_I_example.cpp:
        push    rbp
        mov     rbp, rsp
        mov     esi, 65535
        mov     edi, 1
        call    __static_initialization_and_destruction_0(int, int)
        pop     rbp
        ret
```

I created the square class then created an instance of it to use. The assembly code looks at my program sequentially, going through it line by line. It is then invoked when a function is called. Thus the object data is looked at sequentially and considering the fact that assembly uses the stack to store data in memory, it is able to keep the fields of an object together.

To look at how data members are actually accessed in assembly, the function comp calls instances of getArea using the this keyword and also the object. None of the assembly code changes, as also emphasised when working with stacks. This answers the question that assembly code is accesses the members the same whether from inside or outside the member function, as the code is simply retrieved from the stack.

The pointer, as seen above in one of the screenshots is accessed and stored through the rsp register. It is passed to all the member functions by this register, rsp which sends the data of the pointer to the other functions to use. Although the pointer is also implemented through the stack, assembly is not as detailed with the updating of the pointer.

Resources:

- http://www.drdobbs.com/embedded-systems/object-oriented-programming-in-assembly/184408319
- https://en.wikibooks.org/wiki/X86_Disassembly/Objects_and_Classes
-