Pat Wongwiset
nw9ca
10/31/17
inlab08.pdf

Parameter Passing

Passing by Value

Passing by integer

```
        .globl  __Z7foo_vali
        .align  4, 0x90
__Z7foo_vali:                           ## @_Z7foo_vali
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp2:
        .cfi_def_cfa_register rbp
        mov     eax, 1
        mov     dword ptr [rbp - 4], edi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by Char

```
        .globl  __Z8foo_charc
        .align  4, 0x90
__Z8foo_charc:                          ## @_Z8foo_charc
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp6:
        .cfi_def_cfa_offset 16
Ltmp7:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp8:
        .cfi_def_cfa_register rbp
        mov     al, dil
        mov     edi, 2
        mov     byte ptr [rbp - 1], al
        mov     eax, edi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by float

```
        .globl  __Z9foo_floatf
        .align  4, 0x90
__Z9foo_floatf:                         ## @_Z9foo_floatf
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp12:
        .cfi_def_cfa_offset 16
Ltmp13:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp14:
        .cfi_def_cfa_register rbp
        mov     eax, 4
        movss   dword ptr [rbp - 4], xmm0
        pop     rbp
        ret
        .cfi_endproc
```

Passing by Pointer

```
        .globl  __Z7foo_ptrPi
        .align  4, 0x90
__Z7foo_ptrPi:                          ## @_Z7foo_ptrPi
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp9:
        .cfi_def_cfa_offset 16
Ltmp10:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp11:
        .cfi_def_cfa_register rbp
        xor     eax, eax
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by user defined class

```
        .globl  __Z11foo_intChar7intChar
        .align  4, 0x90
__Z11foo_intChar7intChar:               ## @_Z11foo_intChar7intChar
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp27:
        .cfi_def_cfa_offset 16
Ltmp28:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp29:
        .cfi_def_cfa_register rbp
        xor     eax, eax
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by Reference

Passing by &integer

```
        .globl  __Z10foo_refIntRi
        .align  4, 0x90
__Z10foo_refIntRi:                      ## @_Z10foo_refIntRi
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp30:
        .cfi_def_cfa_offset 16
Ltmp31:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp32:
        .cfi_def_cfa_register rbp
        xor     eax, eax
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by &Char

```
        .globl  __Z11foo_refCharRc
        .align  4, 0x90
__Z11foo_refCharRc:                     ## @_Z11foo_refCharRc
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp30:
        .cfi_def_cfa_offset 16
Ltmp31:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp32:
        .cfi_def_cfa_register rbp
        xor     eax, eax
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by &float

```
        .globl  __Z12foo_refFloatPf
        .align  4, 0x90
__Z12foo_refFloatPf:                    ## @_Z12foo_refFloatPf
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp36:
        .cfi_def_cfa_offset 16
Ltmp37:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp38:
        .cfi_def_cfa_register rbp
        xor     eax, eax
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by &Pointer

```
        .globl  __Z10foo_refPtrRPi
        .align  4, 0x90
__Z10foo_refPtrRPi:                     ## @_Z10foo_refPtrRPi
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp33:
        .cfi_def_cfa_offset 16
Ltmp34:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp35:
        .cfi_def_cfa_register rbp
        xor     eax, eax
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
```

Passing by &(user defined class)

```
        .globl  __Z14foo_refIntCharP7IntChar
        .align  4, 0x90
__Z14foo_refIntCharP7IntChar:           ## @_Z14foo_refIntCharP7IntChar
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp39:
        .cfi_def_cfa_offset 16
Ltmp40:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp41:
        .cfi_def_cfa_register rbp
        xor     eax, eax
        mov     qword ptr [rbp - 8], rdi
        pop     rbp
        ret
        .cfi_endproc
```

Program has a similar pattern when passing by value. The program pushes rbp to continue storing value in rsp. Then, the program would copy (mov) value in first parameter into [rbp - x] in order to plant the integer in subroutine; x = size in bytes of passing parameter. The register in assembly code changes in pattern (rax, eax, al) corresponding to the desired memory it needs to allocate a parameter. Therefore, the [rbp - x] or local variable means the value is located right below rbp which stores rsp.

User Defined Class: The assembly code will write to allocate the passing parameter in the same way as the pointer regardless of the size or types of parameter in the class.

Passing by Reference: All of the snippet code in different types look very similar such that it locates as a local parameter at right below rsp. (-8 in rap -8 means that the memory used to store parameter is 8 bytes. The number of bytes (8) is equal to the memory used to store pointer).

Float: movss means moving a scalar single-precision floating-point value from the source operand (second operand) to the destination operand (first operand). The source and destination operands of the floating number can be XMM registers or 32-bit memory locations.

(http://x86.renejeschke.de/)

Array

C++ Code **Assembly Code**

```
int foo_arr(int x[4]){ // pass by array
  int i = 0;
  int j = 0;
  while(i < 4){
    j += x[i];
    i++;
      }

  return j;
}
```

```
Dump of assembler code for function foo_arr(int*):
=> 0x000000100000c20 <+0>:     push   rbp
   0x000000100000c21 <+1>:     mov    rbp,rsp
   0x000000100000c24 <+4>:     mov    QWORD PTR [rbp-0x8],rdi
   0x000000100000c28 <+8>:     mov    DWORD PTR [rbp-0xc],0x0
   0x000000100000c2f <+15>:    mov    DWORD PTR [rbp-0x10],0x0
   0x000000100000c36 <+22>:    cmp    DWORD PTR [rbp-0xc],0x4
   0x000000100000c3d <+29>:    jge    0x100000c65 <foo_arr(int*)+69>
   0x000000100000c43 <+35>:    movsxd rax,DWORD PTR [rbp-0xc]
   0x000000100000c47 <+39>:    mov    rcx,QWORD PTR [rbp-0x8]
   0x000000100000c4b <+43>:    mov    edx,DWORD PTR [rcx+rax*4]
   0x000000100000c4e <+46>:    add    edx,DWORD PTR [rbp-0x10]
   0x000000100000c51 <+49>:    mov    DWORD PTR [rbp-0x10],edx
   0x000000100000c54 <+52>:    mov    edx,DWORD PTR [rbp-0xc]
   0x000000100000c57 <+55>:    add    edx,0x1
   0x000000100000c5d <+61>:    mov    DWORD PTR [rbp-0xc],edx
   0x000000100000c60 <+64>:    jmp    0x100000c36 <foo_arr(int*)+22>
   0x000000100000c65 <+69>:    mov    eax,DWORD PTR [rbp-0x10]
   0x000000100000c68 <+72>:    pop    rbp
   0x000000100000c69 <+73>:    ret
End of assembler dump.
```

The base address of the array will be located at rcx, and has rdi as a pointer to rcx. When the C++ code increments in the loop, the loop in assembly code will increment the index of the array by +rax*4 as shown on the line which the red arrow points to. rax which stores incrementing number, is multiplied by 4 because the value in the array is 'int'. It would define the next number which is in the next 4 bytes when incrementing i (value of rax). Since the address of each value in array is coded in form of a pointer, the code has to dereference in order to access and operate it.

The loop intends to show how callee accesses parameters and how parameters are passed in the function.

Passing by Pointer VS Passing by Reference

```
        .globl  __Z7foo_ptrPi
        .align  4, 0x90
__Z7foo_ptrPi:                          ## @_Z7foo_ptrPi
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp9:
        .cfi_def_cfa_offset 16
Ltmp10:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp11:
        .cfi_def_cfa_register rbp
        mov     qword ptr [rbp - 8], rdi
        mov     rdi, qword ptr [rbp - 8]
        mov     eax, dword ptr [rdi]
        pop     rbp
        ret
        .cfi_endproc
```

```
        .globl  __Z7foo_refRi
        .align  4, 0x90
__Z7foo_refRi:                          ## @_Z7foo_refRi
        .cfi_startproc
## BB#0:
        push    rbp
Ltmp15:
        .cfi_def_cfa_offset 16
Ltmp16:
        .cfi_offset rbp, -16
        mov     rbp, rsp
Ltmp17:
        .cfi_def_cfa_register rbp
        mov     qword ptr [rbp - 8], rdi
        mov     rdi, qword ptr [rbp - 8]
        mov     eax, dword ptr [rdi]
        pop     rbp
        ret
        .cfi_endproc
```

The assembly code looks exactly similar (i.e. parameter passing, dereference to return) when passing by pointer and reference, so there is no difference between passing by pointer and passing by reference in assembly code. When passing by reference, the parameter which stores value will be allocated the same way as the pointer. The memory address of parameter will be stored as a local variable in the stack at [rbp -8] which is the position right after rsp.