CS 335: Notes on Runtime Environments

Swarnendu Biswas

- Assembly listings use ATnT syntax
- `q` is for 64 bits, `l` is for 32 bits, `w` is 16 bits, `b` is 8 bits. `S` is for 32-bit floating point and `l` is for 64-bit floating point.
- `%rbp` is the frame pointer on `x86_64`

Understanding Stack Manipulation during Function Calls

Source Code

```
void proc(long a1, long* a1p, int a2, int* a2p, short a3, short* a3p, char a4, char* a4p) {
  *a1p += a1;
  *a2p += a2;
  *a3p += a3;
  *a4p += a4;
}
```

Compilation Command and Assembly

```
gcc -S -m64 -fno-asynchronous-unwind-tables -fno-exceptions proc-call.c
```

```
proc:
    endbr64 ; ignore, related to CET
    ; start of prologue
    pushq %rbp ; save caller's %rbp in callee stack, callee-saved
    movq  %rsp, %rbp ; %rbp is now the base of the new stack frame
    ; %rsp will change, so we can use constant offsets from %rbp while generating code for
    ; local variables and function parameters
    ; end of prologue
    movq  %rdi, -8(%rbp) ; save a1 (8 bytes) on the stack
    movq   %rsi, -16(%rbp) ; save a1p (8 bytes) on the stack
    movl   %edx, -20(%rbp) ; save a2 (4 bytes) on the stack
    movq   %rcx, -32(%rbp) ; save a2p (8 bytes) on the stack
    ; stack pointer is aligned to 16 bytes before making a call
    movl   %r8d, %eax ; move a3 (lower 32-bits of %r8) to %eax
    movq   %r9, -40(%rbp) ; save a3p (8 bytes) on the stack
    movl   16(%rbp), %edx ; move a4 (4 bytes) to %edx
    movw   %ax, -24(%rbp) ; move a3 (lower 2 bytes of %eax) to stack
    movl   %edx, %eax ; move a4 to %eax
    movb   %al, -44(%rbp) ; save a4 (4 bytes) on the stack
```

```
    ; *a1p += a1;
    movq  -16(%rbp), %rax ; move a1p to %rax
    movq   (%rax), %rdx ; move *a1p to %rdx
    movq   -8(%rbp), %rax ; %rax contains a1
    addq   %rax, %rdx ; add a1 to *a1p, store in %rdx
    movq   -16(%rbp), %rax ; move a1p to %rax
    movq   %rdx, (%rax) ; store a1+*a1p to the location a1p
    ;  *a2p += a2;
    movq   -32(%rbp), %rax ; move a2p to %rax
    movl   (%rax), %edx ; move *a2p to %edx
    movl   -20(%rbp), %eax ; move a2 to %eax
    addl   %eax, %edx ; add a2 and *a2p, store in %edx
    movq   -32(%rbp), %rax ; move a2p to %rax
    movl   %edx, (%rax) ; store a2+*a2p to the location a2p
    ;  *a3p += a3;
    movq  -40(%rbp), %rax ; move a3p to %rax
    movzwl   (%rax), %eax ; move *a3p (low order 2 bytes) to %eax
    movl   %eax, %edx ; move *a3p to %edx
    movzwl   -24(%rbp), %eax ; move a3 to %eax
    addl   %edx, %eax ; add a3 and *a3p, store in %eax
    movl   %eax, %edx ; move a3+*a3p to %edx
    movq   -40(%rbp), %rax ; move a4p to %rax
    movw   %dx, (%rax) ; store a3+*a3p (2 bytes) to the location a3p
    ; *a4p += a4;
    movq   24(%rbp), %rax ; move a4p to %rax
    movzbl   (%rax), %eax ; move *a4p (low order 1 byte) to %eax
    movl   %eax, %edx ; move *a4p to %edx
    movzbl   -44(%rbp), %eax ; move a4 to %eax
    addl   %edx, %eax ; add *a4p and a4, store in %eax
    movl   %eax, %edx ; move a4+*a4p to %edx
    movq   24(%rbp), %rax ; move a4p to %rax
    movb   %dl, (%rax) ; store a4+*a4p (1 byte) to the location a4p
    ; start of epilogue
    nop
    popq   %rbp ; restore the callee-saved register
    ; end of epilogue
    ret
```

Compilation Command and Assembly

`gcc -O2 -S -m64 -fno-asynchronous-unwind-tables -fno-exceptions proc-call.c`

`-O2` has the effect of including `-fomit-frame-pointer`.

```
proc:
    endbr64 ; ignore, related to CET
    movq   16(%rsp), %rax ; save *a4p to %rax
    addq   %rdi, (%rsi) ; *a1p+=a1;
    addl   %edx, (%rcx) ; *a2p += a2;
    movl   8(%rsp), %edx ; move a4 to %dl (1 byte)
    addw   %r8w, (%r9) ; *a3p += a3; (2 bytes)
```

```
    addb   %dl, (%rax) ; *a4p += a4; (1 byte)
    ret
```

Use of Callee-Saved Registers

Source Code

```
long proc2(long);

long proc1(long x, long y) {
  long u = proc2(y);
  long v = proc2(x);
  return u + v;
}
```

Compilation Command and Assembly

```
gcc -S -m64 -fno-asynchronous-unwind-tables -fno-exceptions callee-saved-regs.c
```

```
proc1:
    ; x is in %rdi, y is in %rsi
    ; start of prologue
    pushq   %rbp ; save caller's %rbp in callee stack, callee-saved register
    movq   %rsp, %rbp ; %rbp is now the base of the new stack frame
    ; %rsp will change, so we can use constant offsets from %rbp while generating code for
    ; local variables and function parameters
    ; end of prologue
    subq   $32, %rsp ; allocate space on stack
    movq   %rdi, -24(%rbp) ; save x on stack
    movq   %rsi, -32(%rbp) ; save y on stack
    movq   -32(%rbp), %rax ; move y to %rax
    movq   %rax, %rdi ; move y to %rdi, prepare first parameter
    call   proc2@PLT
    movq   %rax, -16(%rbp) ; save return value u to stack
    movq   -24(%rbp), %rax ; move x to %rax
    movq   %rax, %rdi ; move x to %rdi, prepare first parameter
    call   proc2@PLT
    movq   %rax, -8(%rbp) ; save return value v to stack
    movq   -16(%rbp), %rdx ; pop u
    movq   -8(%rbp), %rax ; pop v
    addq   %rdx, %rax ; store u+v in %rax
    leave ; epilogue
    ret
```

Compilation Command and Assembly

```
gcc -O2 -S -m64 -fno-asynchronous-unwind-tables -fno-exceptions callee-saved-regs.c
```

```
proc1:
    pushq   %rbp ; save caller's %rbp in callee stack, callee-saved register
    movq   %rdi, %rbp ; save x on stack
    movq   %rsi, %rdi ; move y to %rdi
    pushq   %rbx ; save caller's %rbp in callee stack, callee-saved register
```

```
; cannot use 128-byte red zone below stack pointer because proc1 is not a leaf function,
; so proc2 can clobber the red zone
subq   $8, %rsp ; reserve space on stack for locals
call   proc2@PLT
movq   %rbp, %rdi ; move x to %rdi
movq   %rax, %rbx ; move u to %rbx
call   proc2@PLT
addq   $8, %rsp ; deallocate space from stack
addq   %rbx, %rax ; store u+v in %rax
popq   %rbx ; restore the callee-saved register
popq   %rbp ; restore the callee-saved register
ret
```

References

- Randal E. Bryant and David R. O'Hallaron. Computer Systems: A Programmer's Perspective, 3rd edition.
- Where the top of the stack is on x86
- Stack frame layout on x86-64
- Position Independent Code (PIC) in shared libraries
- C++ Internals Memory layout
- x86 Disassembly/Calling Conventions
- Calling conventions for different C++ compilers and operating systems