Assembly: x86-64, Back Doors

CS 2130: Computer Systems and Organization 1

October 14, 2022

Announcements

- Homework 5 due Wednesday 10/19 at 11pm
- Quiz 6 out tonight, due Monday at 8am

Functions

```
f(x,y):
    ...
    ...
    return 4
```

f:

retq

```
...
z = f(2,5)
```

callq f

Function Calls

`callq myfun`

- Push return address to stack, then jump to myfun

`retq`

- Pop return address from stack and jump back

This is similar to our Toy ISA's function calls in homework 4

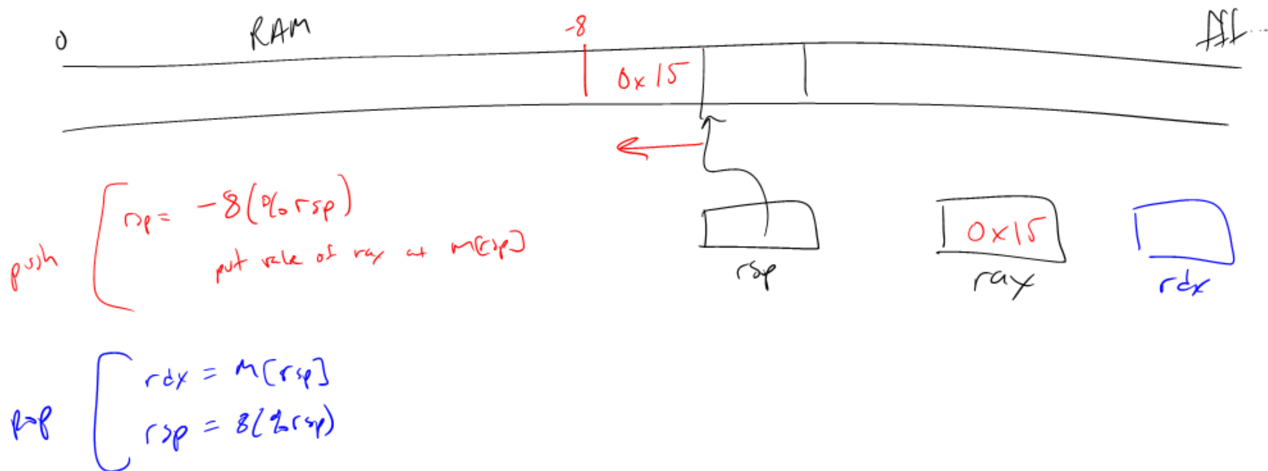
Calling Conventions: Parameters

Calling conventions - recommendations for making function calls

- Where to put arguments/parameters for the function call?
  - First 6 arguments (in order): `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`
  - If more arguments, push onto stack (last to first)
- Where to put return value? in `rax` before calling `retq`
- What happens to values in the registers?
  - Callee-save - The function should ensure the values in these registers are unchanged when the function returns
    - `rbx`, `rsp`, `rbp`, `r12`, `r13`, `r14`, `r15`
  - Caller-save - Before making a function call, save the value, since the function may change it

The Stack

```
pushq %rax
popq  %rdx
```



example.s

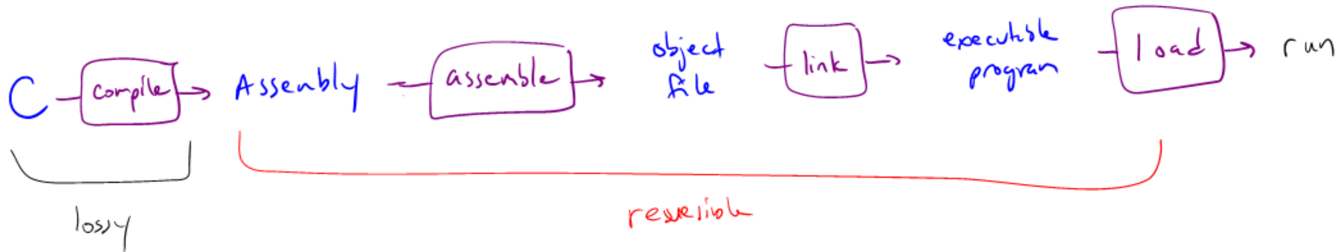
example.s

```
    .globl main
main:
    pushq   %rbp
    movq    $0, %rbp
condition:
    cmpq    $42, %rbp
    jg      after
    movq    %rbp, %rsi
    leaq    fmtstring(%rip), %rdi
    callq   printf
    addq    $1, %rbp
    jmp     condition
after:
    xorl    %eax, %eax
    popq    %rbp
    retq
fmtstring:
    .asciz "i = %ld\n"
```

Compilation Pipeline

Turning our code into something that runs

- **Pipeline** - a sequence of steps in which each builds off the last



Most Common Instructions

- `mov` - =
- `lea` - load effective address
- `call` - push PC and jump to address
- `add` - +=
- `cmp` - set flags as if performing subtract
- `jmp` - unconditional jump
- `test` - set flags as if performing &
- `je` - jump iff flags indicate == 0
- `pop` - pop value from stack
- `push` - push value onto stack
- `ret` - pop PC from the stack