HSGS Sec Day 05

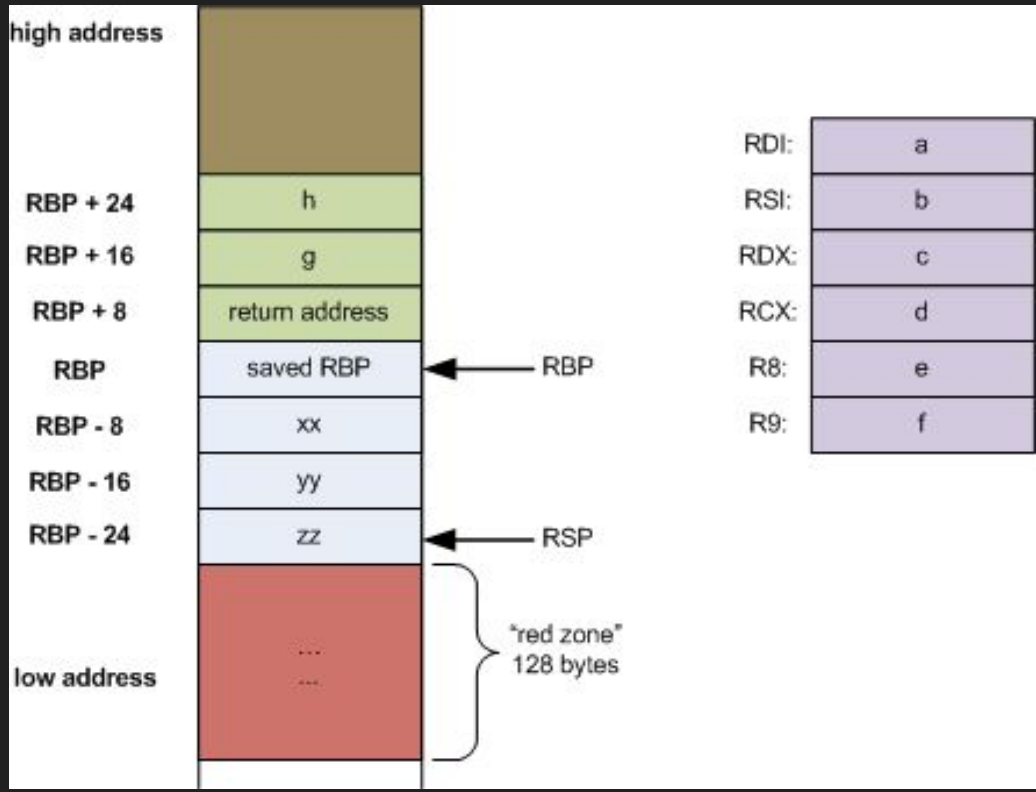
Stack, calling convention, function

Agenda

1. Quiz + Homework
2. Stack
3. Function
4. Calling convention

Stack

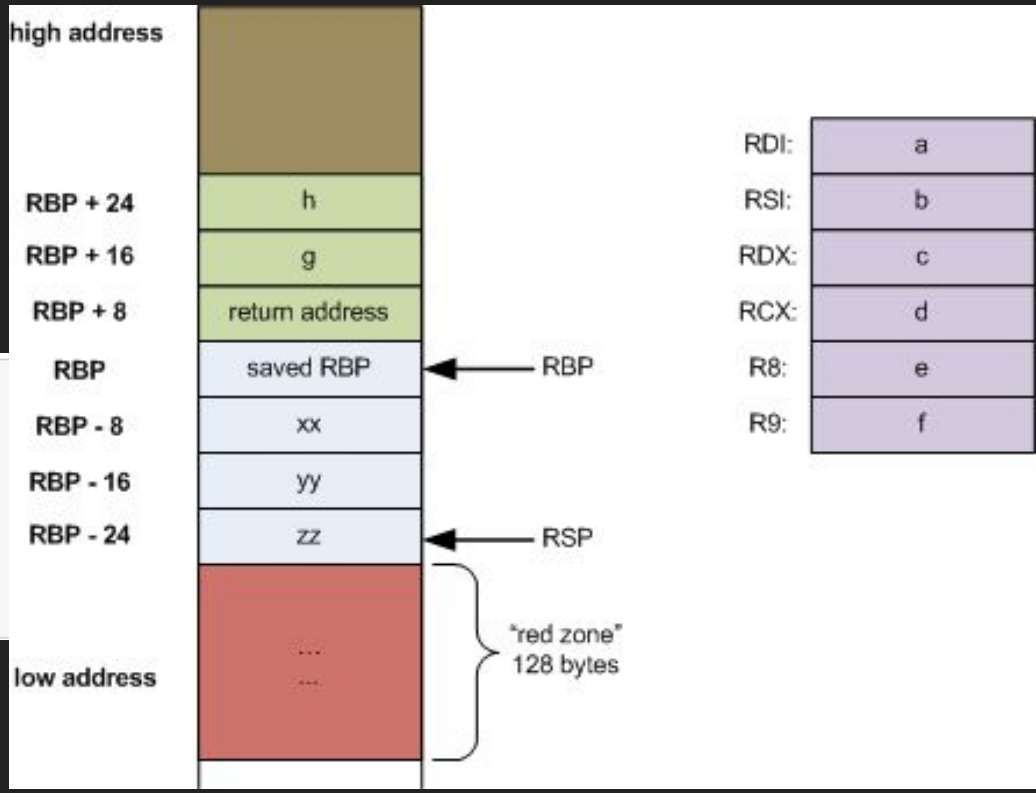
- Về cơ bản là một vùng bộ nhớ được chỉ định sẵn trong RAM
- Giới hạn đầu/cuối tương đối của stack được định nghĩa bởi giá trị của 2 thanh ghi rbp (base pointer) và rsp (stack pointer)
- rbp >= rsp. rbp trỏ tới đáy của stack và rsp trỏ tới đỉnh của stack



Stack

- Biến cục bộ và tham số của hàm được lưu trên stack

```
long myfunc(long a, long b, long c, long d,
            long e, long f, long g, long h)
{
    long xx = a * b * c * d * e * f * g * h;
    long yy = a + b + c + d + e + f + g + h;
    long zz = utilfunc(xx, yy, xx % yy);
    return zz + 20;
}
```



Stack

- Các operation trên stack
  - push: đẩy 1 giá trị lên stack (ví dụ, push rax)
    - rsp -= 8
    - *rsp = rax
    - Giá trị của rax sẽ chiếm 8 ô nhớ (rsp → rsp+7 inclusively)
  - pop: lấy 1 giá trị khỏi stack (ví dụ, pop rax)
    - rax = *rsp
    - rsp += 8
- Trên x64 architecture, các thao tác push/pop thay đổi rsp 8 đơn vị/ô nhớ. Còn trên x86(32 bit) là 4 đơn vị.

Agenda

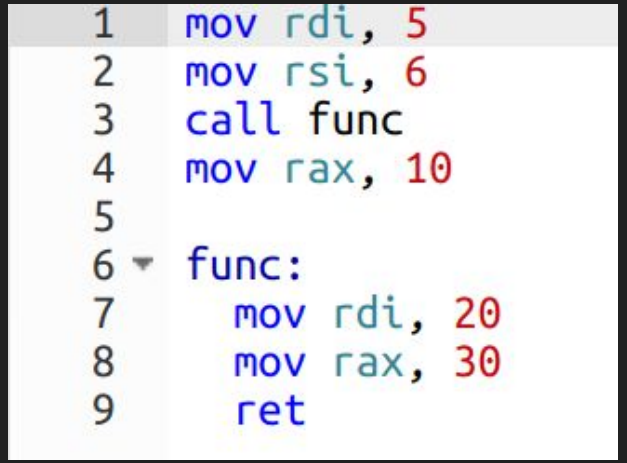
1. Quiz + Homework
2. Stack
3. Function
4. Calling convention

Function

- Để thực hiện các công việc phức tạp hơn, ta cần dùng hàm
- Mỗi hàm có một stack frame để lưu data cho việc thực hiện 1 hàm, stack frame này giới hạn bởi rbp và rsp
- Khi gọi và kết thúc 1 hàm, rbp và rsp được thay đổi tương ứng để cấp cho hàm đó 1 stack frame tương ứng
  - Prologue (bắt đầu hàm): push rbp; mov rbp, rsp; sub rsp, N
  - Epilogue (kết thúc hàm): leave; ret
    - leave == mov rsp, rbp; pop rbp
    - ret: Return về instruction sau khi gọi hàm

Function: Return về đâu?

- Sau call func, chương trình sẽ chạy dòng code 5
- Luồng thực thi (execution flow) được điều khiển bằng 1 register đặc biệt: rip (instruction pointer) hoặc pc (program counter)
- Giá trị của rip là địa chỉ của dòng code sẽ được thực thi. Ví dụ, ngay trước khi thực hiện gọi hàm func, rip = 3. Sau khi gọi hàm func và trước khi thực hiện dòng 4, rip = 4



Function: Return về đâu?

- Vì chỉ có 1 rip, khi gọi hàm func, rip sẽ được thay đổi như sau
  - rip = 3 ; Trước khi gọi hàm func
  - rip = 7; rip = 8; rip = 9; Trong khi thực hiện hàm func
  - rip = 4; Sau khi gọi hàm func
- **Lưu ý đây chỉ là minh họa, không phải ví dụ đầy đủ về cách code function trong assembly**

```
1  mov rdi, 5
2  mov rsi, 6
3  call func
4  mov rax, 10
5
6  func:
7    mov rdi, 20
8    mov rax, 30
9    ret
```

Function: Return về đâu?

- Để có thể chuyển đổi trước/sau khi gọi hàm, trước khi gọi hàm func, địa chỉ trở về sau khi gọi hàm sẽ được đẩy lên stack. Địa chỉ này gọi là **RETURN ADDRESS**
  - Trước khi gọi hàm func, 4 sẽ được đẩy lên stack
  - instruction ret trong hàm func sẽ dựa vào giá trị được đẩy lên stack để biết nơi tiếp theo cần chạy code.
  - 4 là return address khi gọi hàm func tại dòng 3

```
1  mov rdi, 5
2  mov rsi, 6
3  call func
4  mov rax, 10
5
6  func:
7    mov rdi, 20
8    mov rax, 30
9    ret
```

Agenda

1. Quiz + Homework
2. Stack
3. Function
4. Calling convention

Calling convention

- Calling convention có thể coi như phương thức gọi hàm. Cần làm một số thao tác để có thể gọi hàm thành công, bảo toàn các dữ liệu cần thiết trước và sau khi gọi hàm
- Trước khi gọi hàm
  - 0. register không đảm bảo sẽ được giữ nguyên sau khi gọi hàm. Vì vậy, nếu muốn bảo toàn giá trị, cần dùng local variable hoặc lưu giá trị của register vào local variable trước khi gọi hàm
  - 1. Tham số sẽ được chuyển tới nơi được quy định. Điều này tương đương với việc thực hiện syscall, nhưng vị trí các argument sẽ khác. Khi gọi hàm, các tham số thứ 1 → 6 sẽ lần lượt được lưu vào các register theo thứ tự sau: rdi, rsi, rdx, rcx, r8, r9. Các tham số từ 7 trở đi sẽ được push lên stack của hàm hiện tại.
  - 2. Địa chỉ trở về sau khi gọi hàm (return address) sẽ được đẩy lên stack

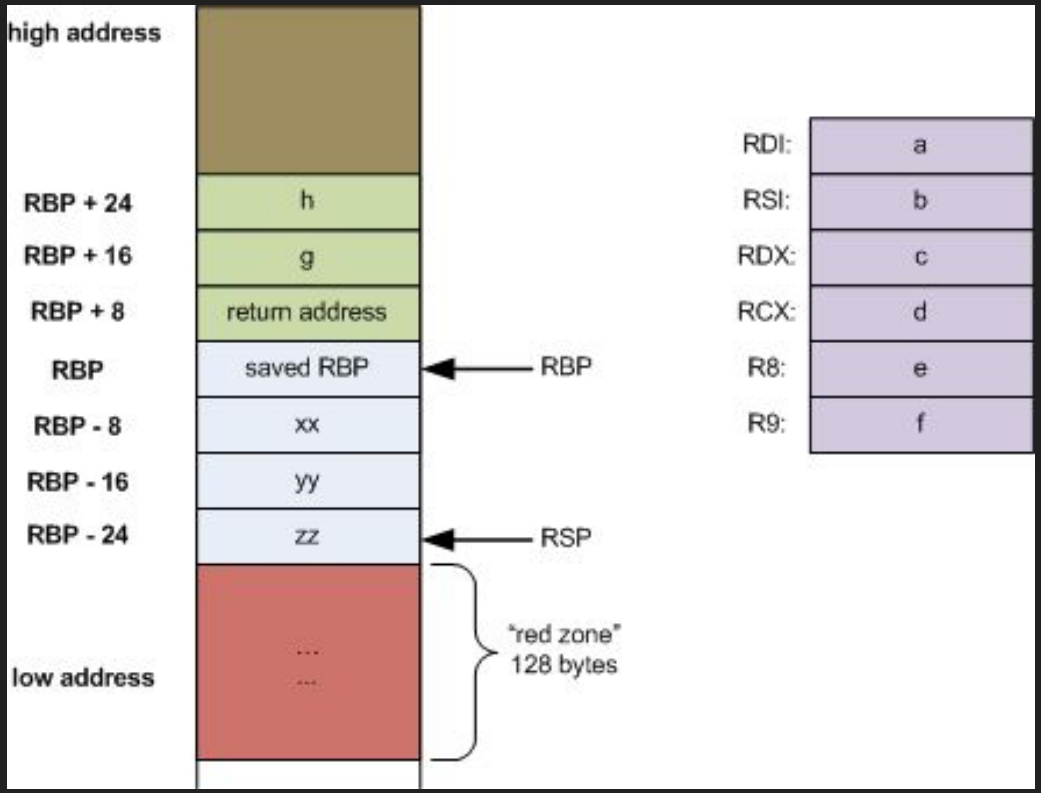
Calling convention

Register	Usage	callee saved
%rax	temporary register; with variable arguments passes information about the number of vector registers used; 1st return register	No
%rbx	callee-saved register	Yes
%rcx	used to pass 4th integer argument to functions	No
%rdx	used to pass 3rd argument to functions; 2nd return register	No
%rsp	stack pointer	Yes
%rbp	callee-saved register; optionally used as frame pointer	Yes
%rsi	used to pass 2nd argument to functions	No
%rdi	used to pass 1st argument to functions	No
%r8	used to pass 5th argument to functions	No
%r9	used to pass 6th argument to functions	No
%r10	temporary register, used for passing a function's static chain pointer	No
%r11	temporary register	No
%r12–r14	callee-saved registers	Yes
%r15	callee-saved register; optionally used as GOT base pointer	Yes

Calling convention

- Trong khi gọi hàm
  - Bắt đầu gọi hàm: Thiết lập stackframe cho bản thân bằng cách thay đổi rbp và rsp
    - 1. Bảo toàn rbp cho hàm gọi mình (caller) bằng cách **push rbp**. Vì rsp thay đổi liên tục nên không cần bảo toàn
    - 2. Gán giá trị mới cho rbp. **mov rbp, rsp**. Điều này thiết lập 1 boundary cho stackframe của hàm này. Lúc này độ rộng stackframe = 0 vì rbp == rsp
    - 3. Nới rộng stackframe của hàm bằng cách giảm rsp. **sub rsp, N**
  - Kết thúc gọi hàm
    - Trả lại rsp về vị trí lúc bắt đầu gọi hàm: leave == **mov rsp, rbp; pop rbp. rbp ở thời điểm này mang giá trị tại bước (2) phía trên.**
    - Nhảy về return address: **ret**

Calling convention



Calling convention

- Truy cập dữ liệu trong hàm như nào?
  - Address trên stackframe của 1 hàm có thể tính bằng (rbp - x) hoặc (rsp + x)
  - x >= 0; rbp >= rsp
  - Điều này đảm bảo địa chỉ được refer nằm trong stackframe đó.
- Sau khi gọi hàm, dữ liệu trong stackframe của hàm đó sẽ bị thay đổi tùy ý bởi các lệnh gọi hàm khác. Vì vậy, biến cục bộ không được vượt quá phạm vi stackframe 1 hàm.

Resources

[1] Stack frame layout on x86-64

[2] Functions and Stack in NASM 32-bit | Tachyon