Embedded Device Exploitation 101: An Introduction to Firmware Hacking

Benjamin Kollenda

embo++ 2023, 23.03.2023

Embedded RE - Recap

Deeply embedded devices

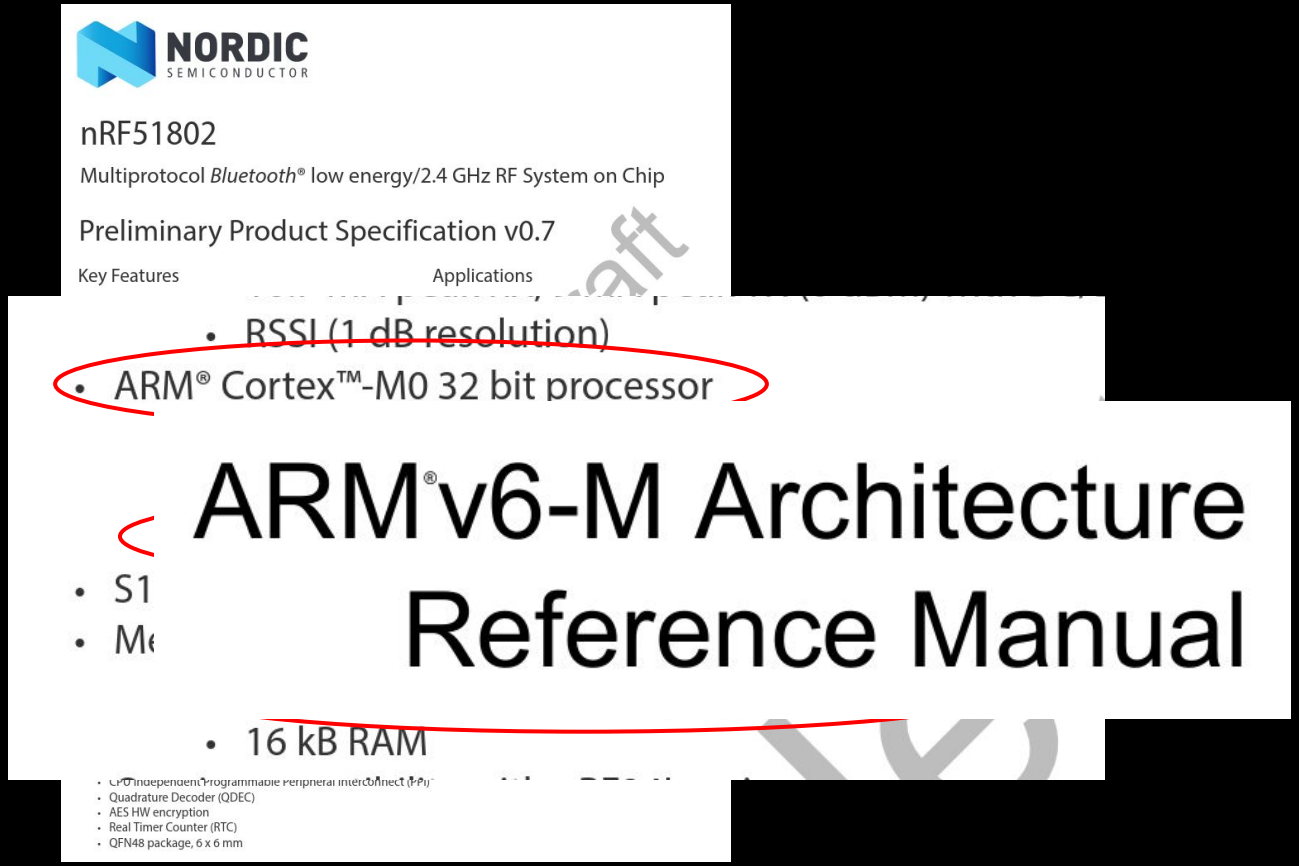
emBO++

How to break things

1. Know it
2. Dump it
3. Analyze it
4. Break it

How to break things

1. Know it
2. Dump it
3. Analyze it
4. Break it



How to break things

1. Know it ✔
2. Dump it
3. Analyze it
4. Break it

How to break things

1. Know it ✔
2. Dump it ✔
3. Analyze it
4. Break it

emBO++

```
0000120 e7f0
0000130 e7e8
0000140 4825
0000150 290d
0000160 2113
0000170 2000
0000180 2000
0000190 2001
00001a0 2000
00001b0 490b
00001c0 d009
00001d0 07c0
00001e0 0500
00001f0 210f
0000200 e7fe
0000210 01a5
0000220 68e3
0000230 f7ff
0000240 1e64
0000250 bc30
0000260 2a00
0000270 2a00
```

```
rld!"

    func2(ptr_Hello);
}
```

How to break things

1. Know it ✔
2. Dump it ✔
3. Analyze it ✔
4. Break it

emBO++

Source to Firmware

C bugs to assembly bugs

A bug's tale

1. Write C bug

```
while (1) {
    // (3) read the plaintext from a prom
    char array[32] = {0};
    if (prompt(UART0, array) != 0) {
        break;
    }
```

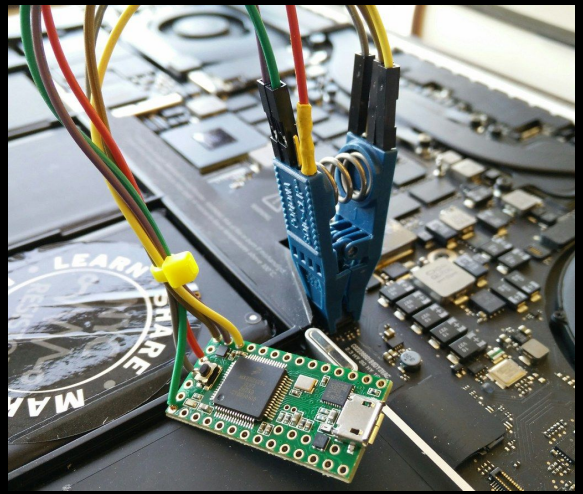
A bug’s tale

1. Write C bug
2. Compile

```
0000120 e7f0 2001 e7ee 2001 e7ec 2000 e7ea 2001
0000130 e7e8 2001 e7e6 2001 e7e4 2001 e7e2 b500
0000140 4825 69c0 b280 381d 0003 f000 f877 1718
0000150 290d 2929 2929 290f 2523 2929 1b19 111f
0000160 2113 1529 2729 291d 2001 bd00 2001 e7fc
0000170 2000 e7fa 2000 e7f8 2001 e7f6 2001 e7f4
0000180 2000 e7f2 2000 e7f0 2000 e7ee 2000 e7ec
0000190 2001 e7ea 2001 e7e8 2001 e7e6 2001 e7e4
00001a0 2000 e7e2 b510 f7ff ffca 2800 d005 480b
00001b0 490b 6048 13c8 490b 6188 f7ff ff8d 2800
00001c0 d009 2001 2105 0709 2207 0212 1889 67c8
00001d0 07c0 4905 60c8 bd10 0040 1000 ffdf c007
00001e0 0500 4000 6c00 4000 0500 5000 4807 6802
00001f0 210f 430a 6002 4806 4780 4806 4700 e7fe
0000200 e7fe e7fe e7fe e7fe e7fe 0000 0524 4000
0000210 01a5 0000 00c1 0000 4c06 2501 4e06 e005
0000220 68e3 cc07 432b 3c0c 4798 3410 42b4 d3f7
0000230 f7ff ff4a 0278 0000 0288 0000 b430 4674
0000240 1e64 7825 1c64 42ab d304 5d63 005b 18e3
0000250 bc30 4718 461d e7f8 e002 c808 1f12 c108
0000260 2a00 d1fa 4770 4770 2000 e001 c101 1f12
0000270 2a00 d1fb 4770 0000 0288 0000 0000 2000
```

A bug’s tale

1. Write C bug
2. Compile
3. Program



A bug’s tale

1. Write C bug
2. Compile
3. Program
4. Ship

A bug’s tale

1. Write C bug
2. Compile
3. Program
4. Ship

1. Dump it
2. Analyze it
3. Find it
4. Exploit it

A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship

1. Dump it
2. Analyze it
3. Find it
4. Exploit it

A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship

1. Dump it ✔
2. Analyze it ✔
3. Find it
4. Exploit it

A bug's tale

1. Write C bug
2. Compile
3. Program
4. Ship

1. Dump it ✔
2. Analyze it ✔
3. Find it
4. Exploit it

emBO++

Classic Vulnerabilities

Classic vulnerabilities

emBO++

- Buffer overflows
  - Stack

```
while (1) {
    // (3) read the plaintext from a prom
    char array[32] = {0};
    if (prompt(UART0, array) != 0) {
        break;
    }
```

Classic vulnerabilities

emBO++

- Buffer overflows
  - Stack
  - Heap

```
void heap_overflow(char* in_buf) {
    char* buf = malloc(32);
    memcpy(buf, in_buf, 64);
    free(buf);
}
```

Classic vulnerabilities

emBO++

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/unde

```
void int_overflow(char* in_buf, size_t length) {
    uint32_t full_size = 64 + length;
    char* buf = malloc(512);
    if (full_size < 512) {
        memcpy(buf, in_buf, length + 64)
    }
    free(buf);
}
```

Classic vulnerabilities

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/und
- Format strings

```
void format_string() {
    char tmp_str[64] = {0};
    char dest_buf[64] = {0};
    uart_read(tmp_str, 64);
    sprintf(dest_buf, tmp_str, strlen(tmp_str));
}
```

Classic vulnerabilities

emBO++

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/und
- Format strings
- Use after free

```
void user_after_free() {
    struct A* a = malloc(sizeof(struct A));
    free(a);
    struct B* b = malloc(sizeof(struct B));
    a->length = 512;
    free(b);
}
```

Classic vulnerabilities

emBO++

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/underflows
- Format strings
- Use after free
- TOCTOU

```
void toctou(char* file_name) {
    if (!is_symlink(file_name)) {
        write_log(file_name);
    }
}
```

Classic vulnerabilities

- Buffer overflows
  - Stack
  - Heap
- Integer overflows/underflows
- Format strings
- Use after free
- TOCTOU

emBO++

… and how to find them

Finding vulnerabilities

emBO++

- Code review
- Fuzzing
- Reverse engineering
- Try things

Finding vulnerabilities

emBO++

- Code review
- Fuzzing
- Reverse engineering
- Try things

Exploiting things

1. Dump it ✔
2. Analyze it ✔
3. Find it

Exploiting things

1. Dump it ✔
2. Analyze it ✔
3. Find it ✔

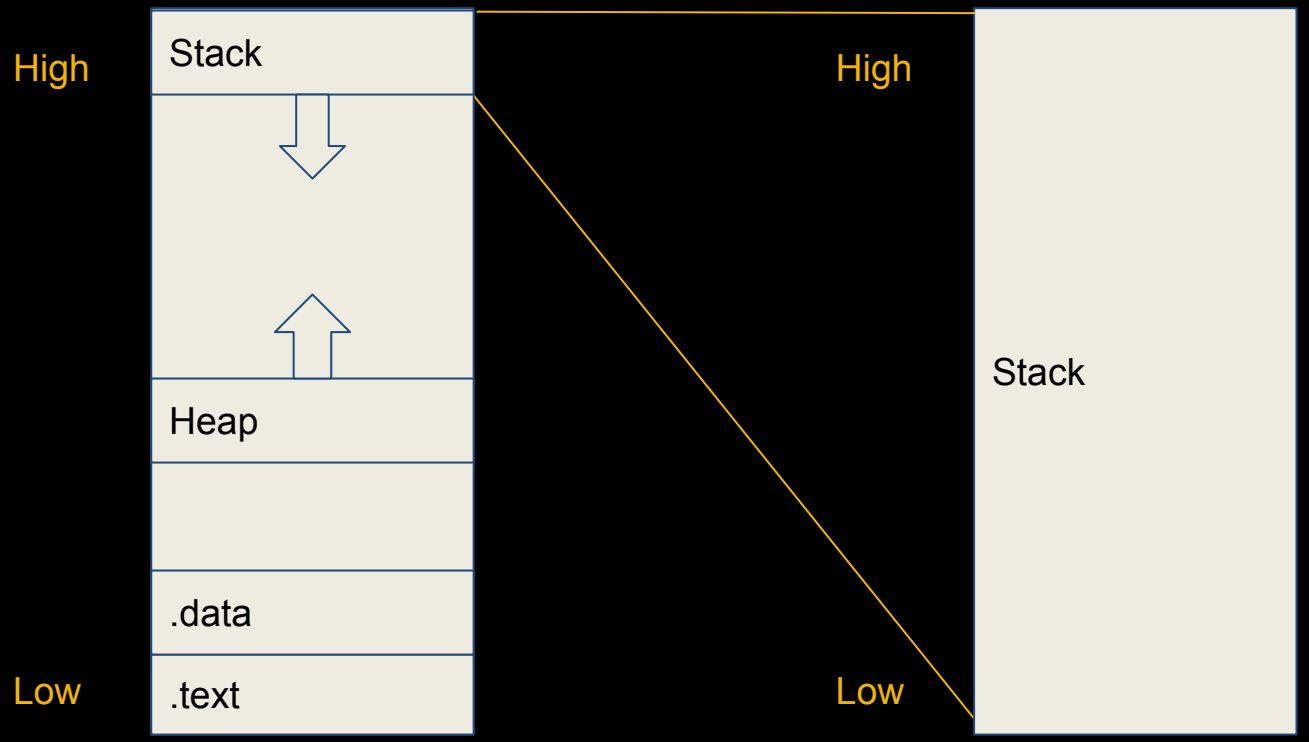
Exploiting things

1. Dump it ✔
2. Analyze it ✔
3. Find it ✔
4. Exploit it

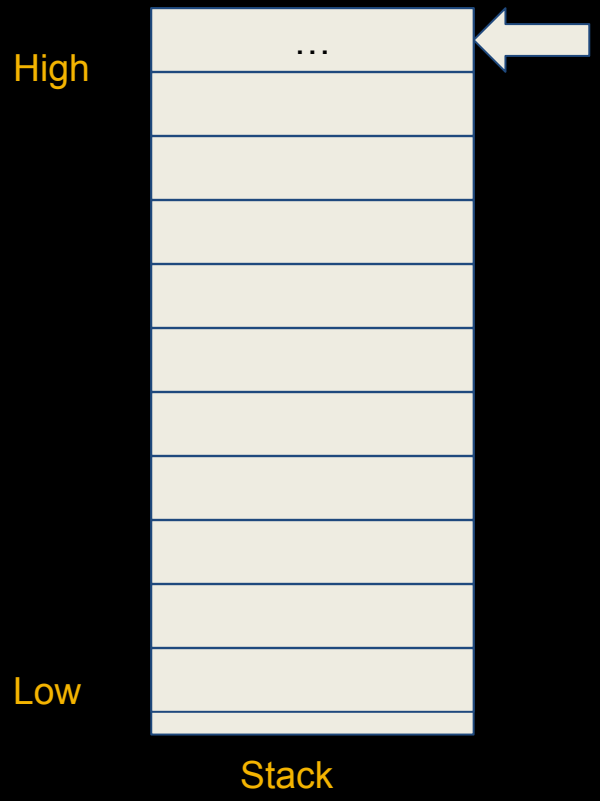
emBO++

Stack buffer overflow I

Stacks 101

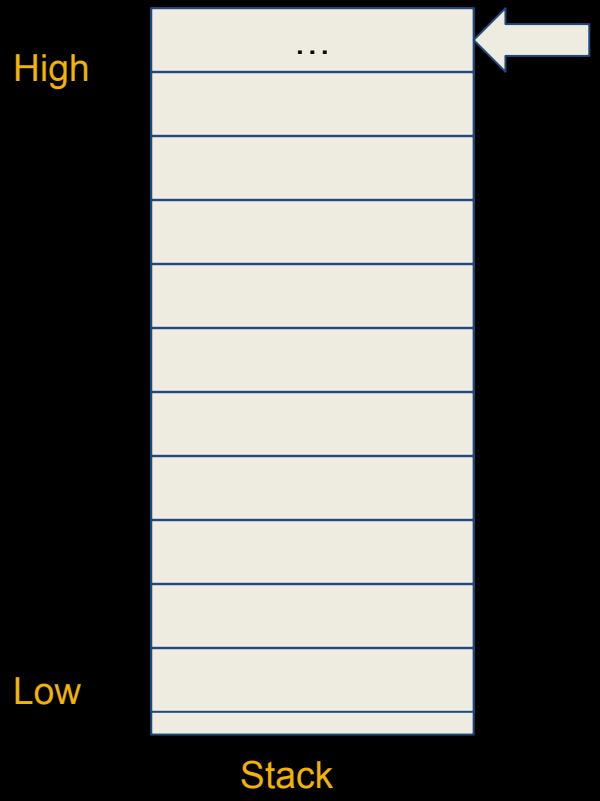


Stacks 101



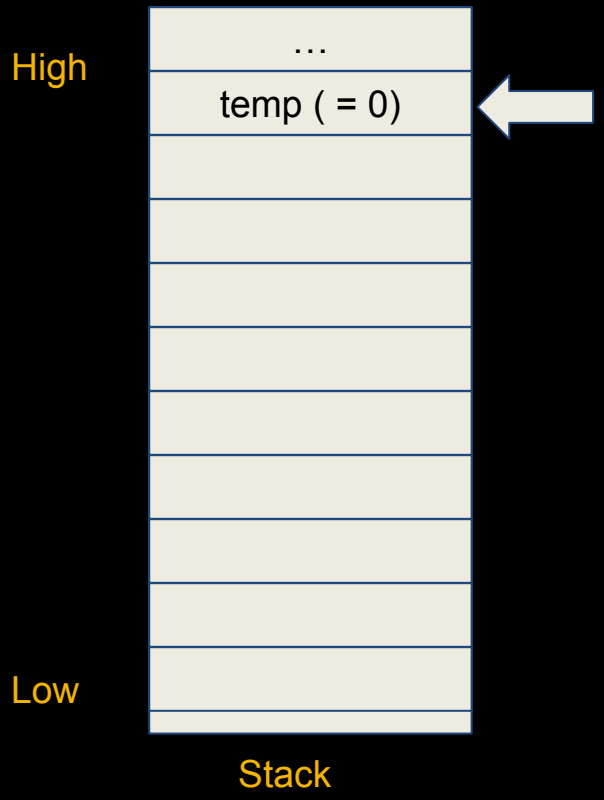

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



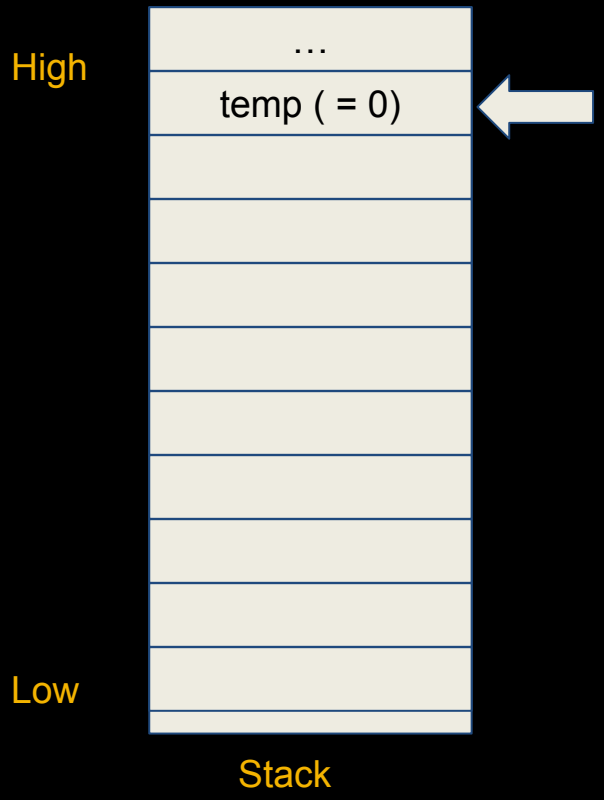

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



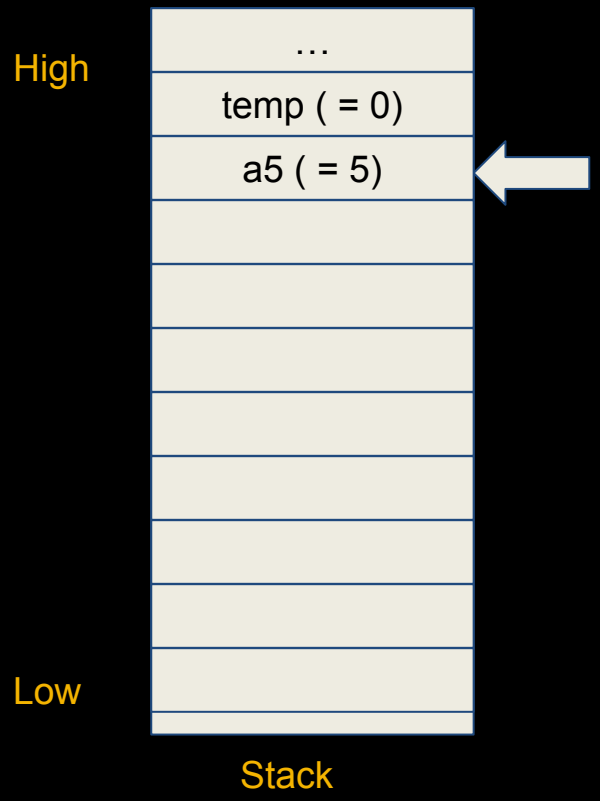

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



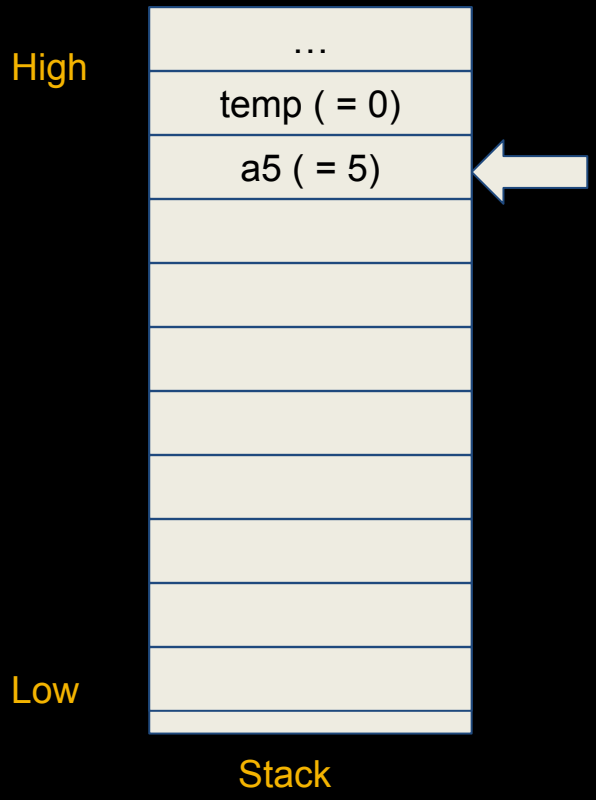

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



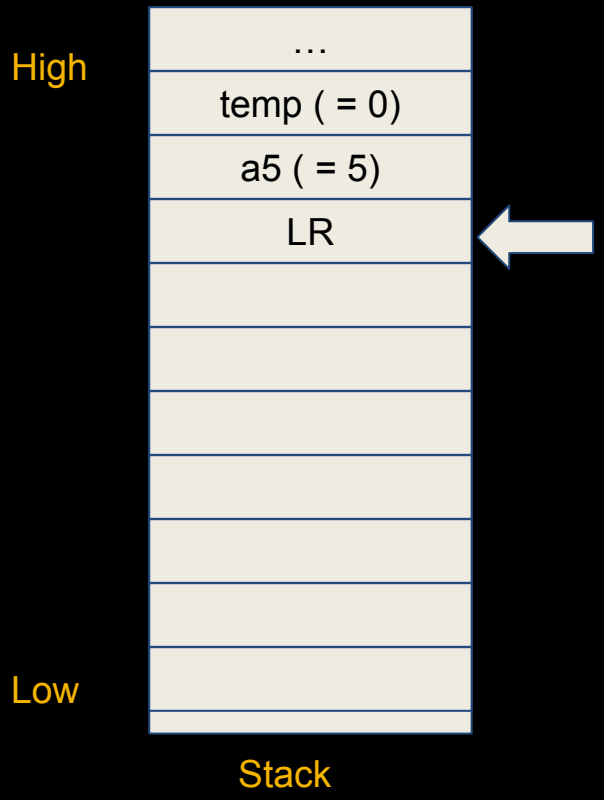

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



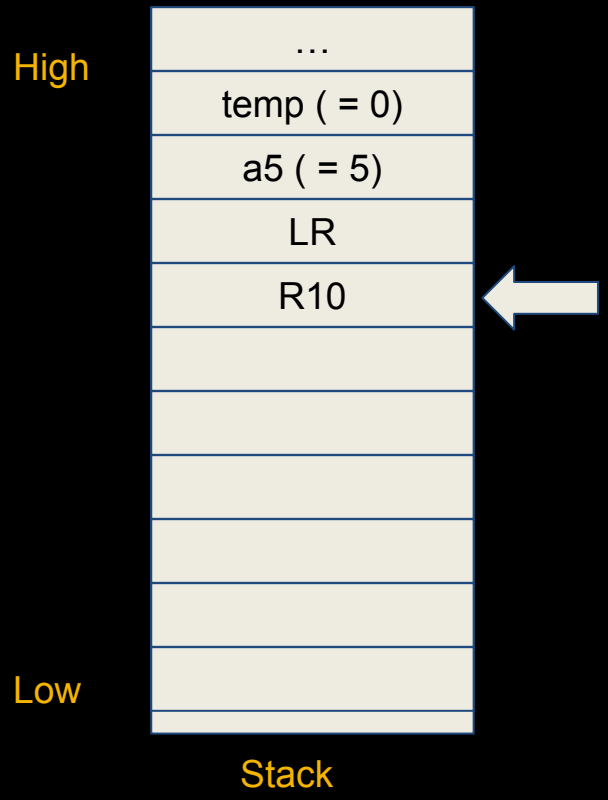

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



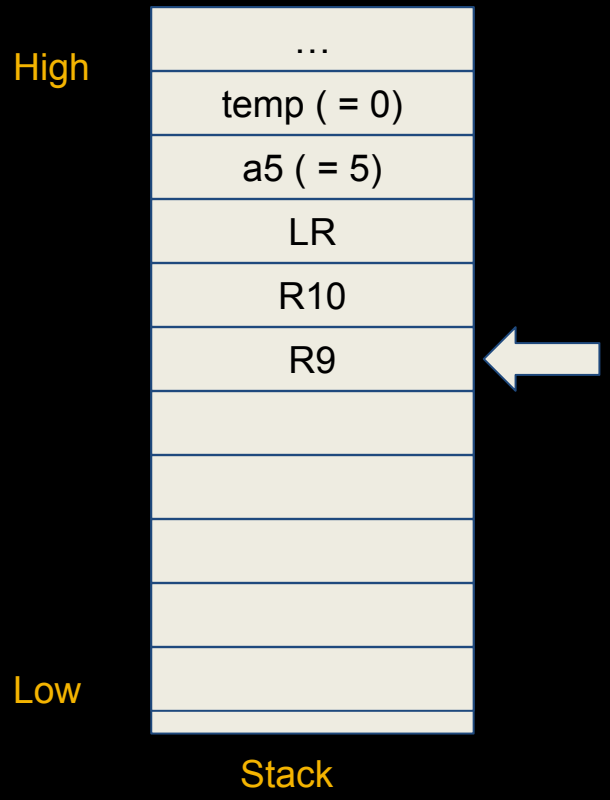

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



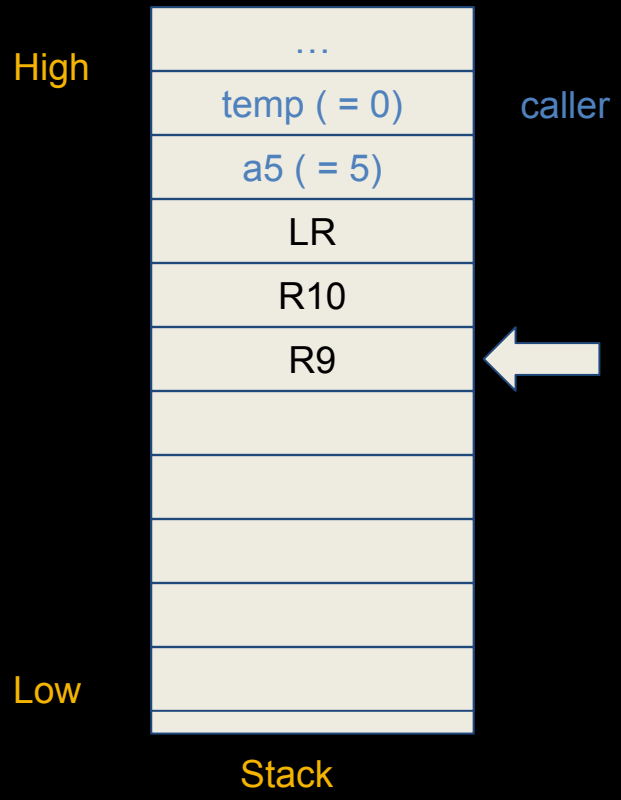

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



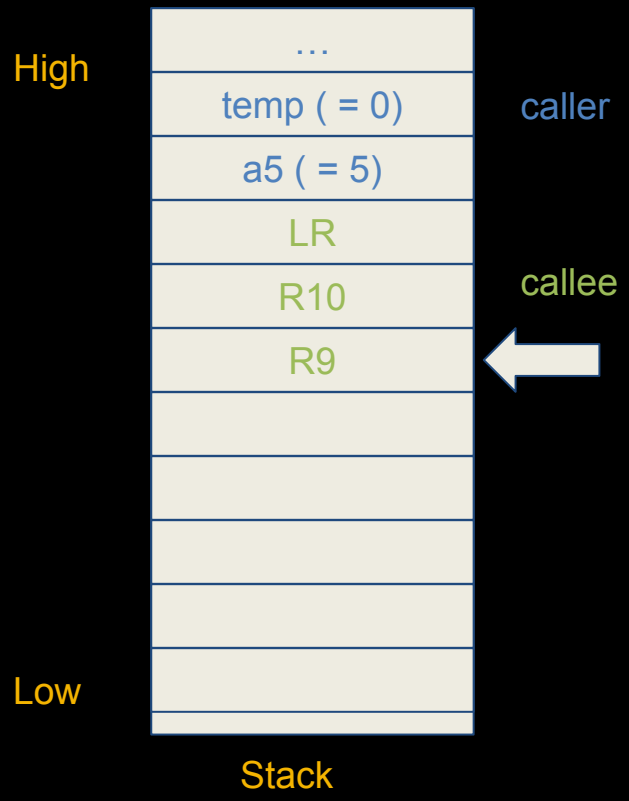

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



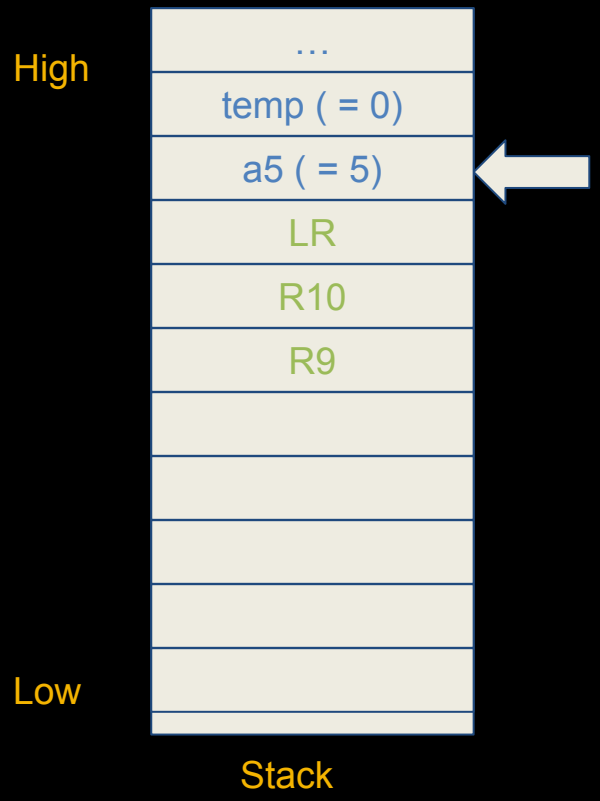

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



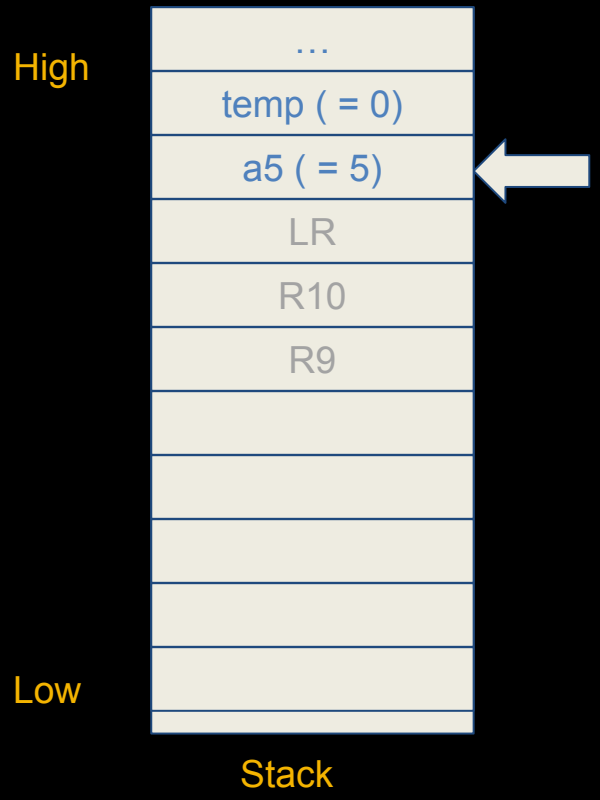

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



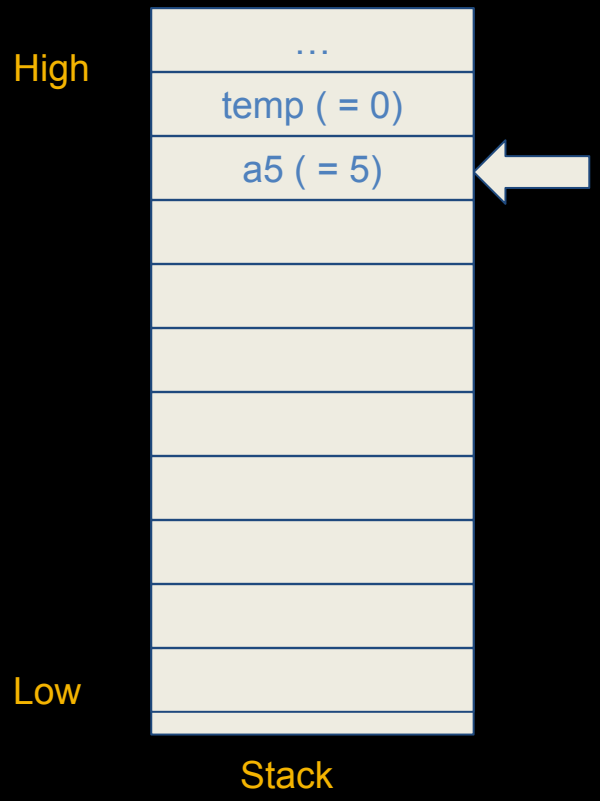

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



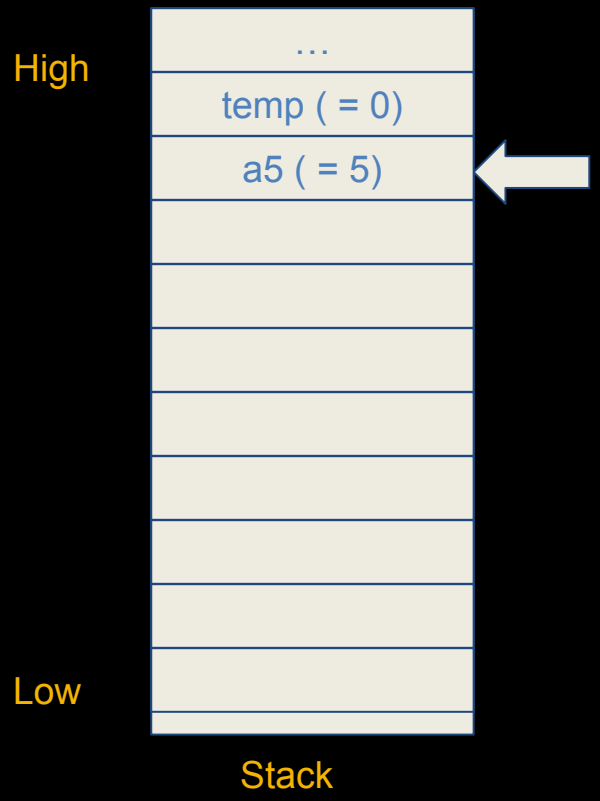

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



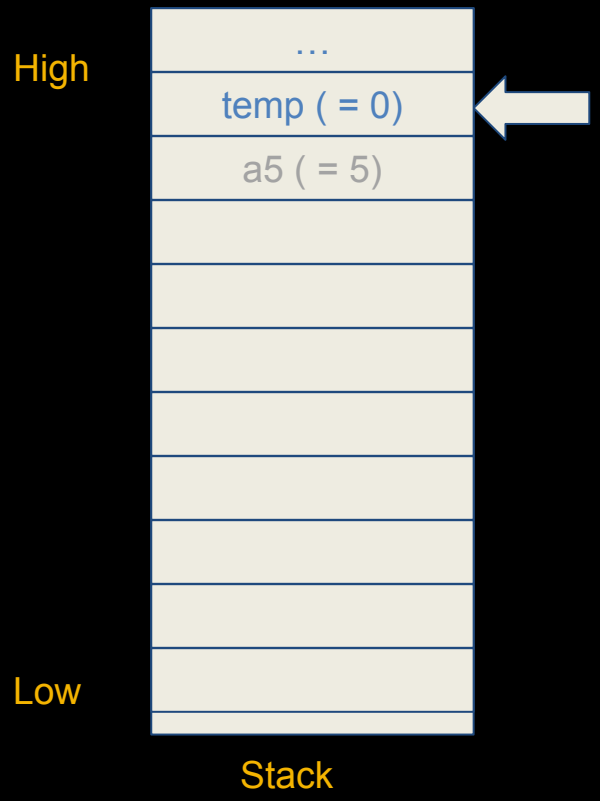

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



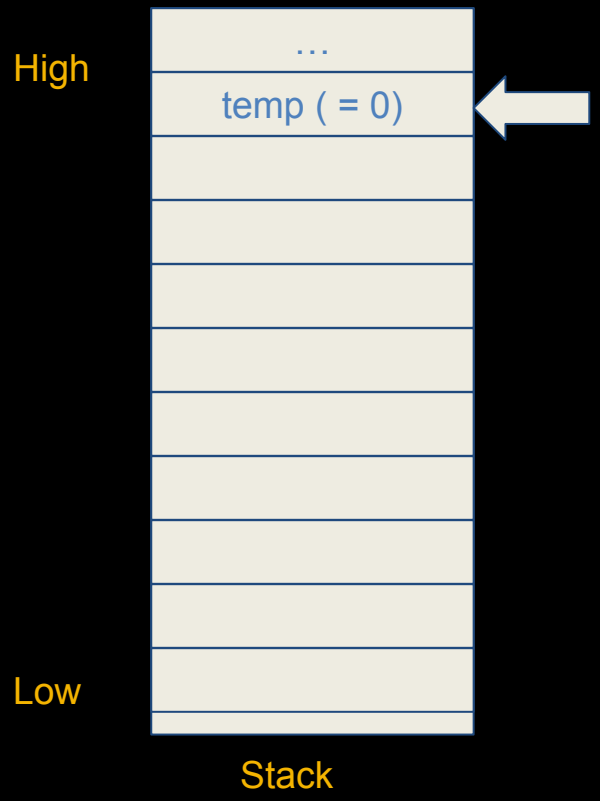

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



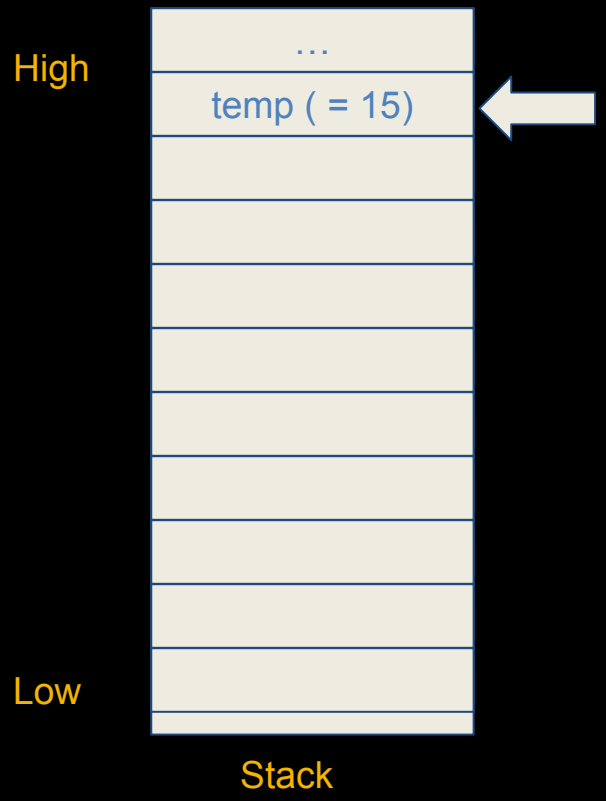

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



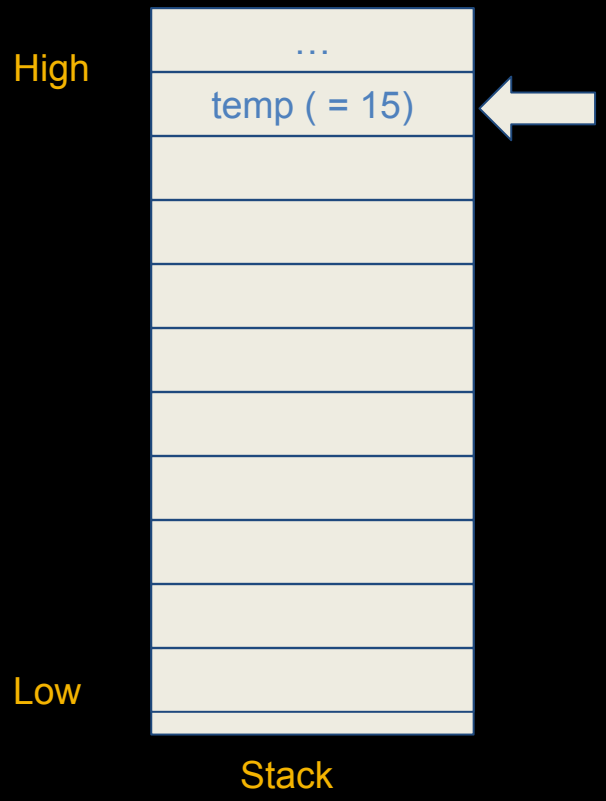

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



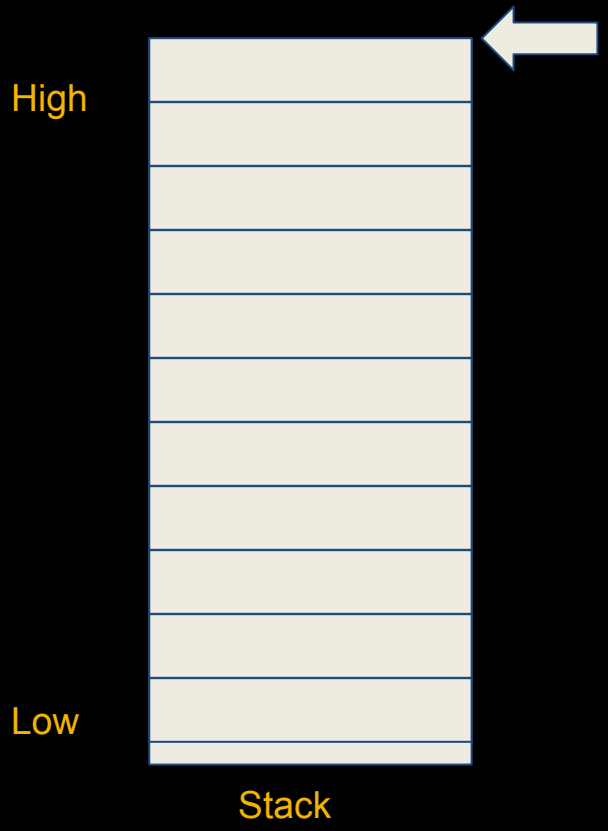

```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101




```
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

Stacks 101



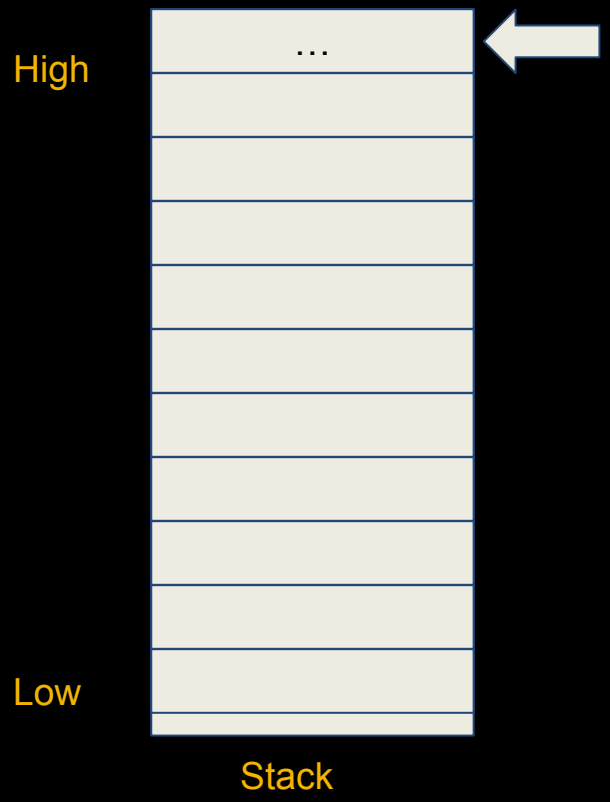

```c
int callee(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}

int caller() {
    int temp = 0;
    temp = callee(1, 2, 3, 4, 5);
    return temp;
}
```

emBO++

Stack overflow

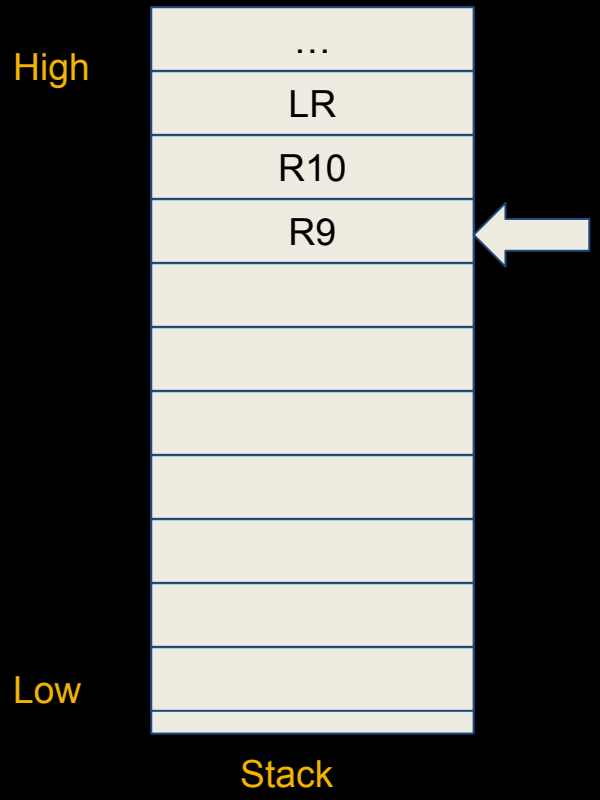

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



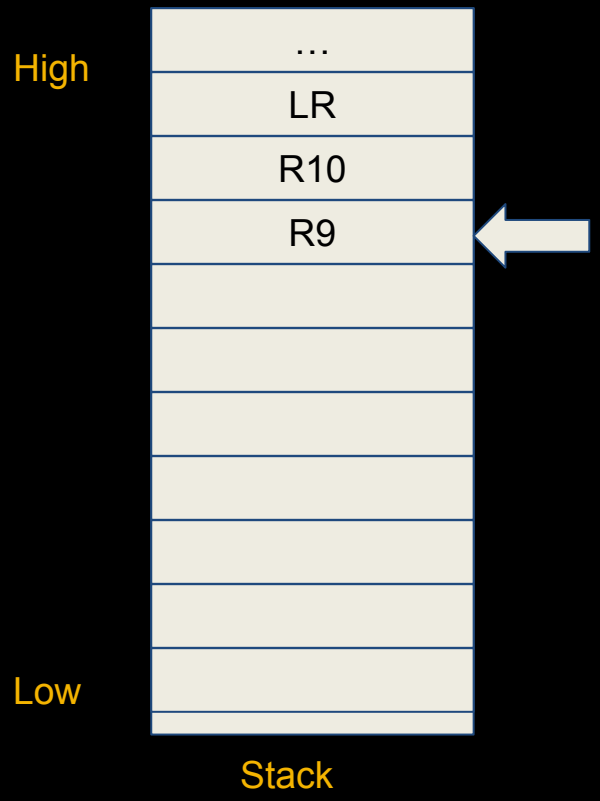

```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



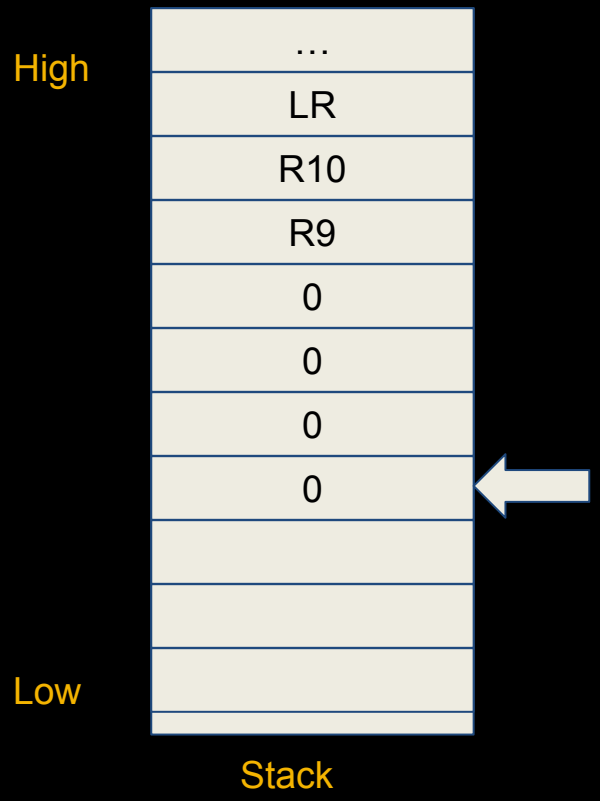

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



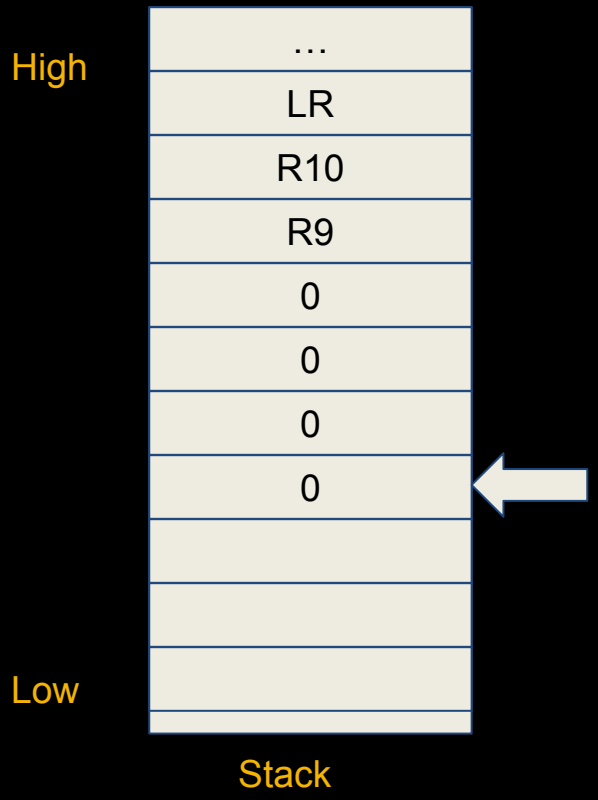

```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



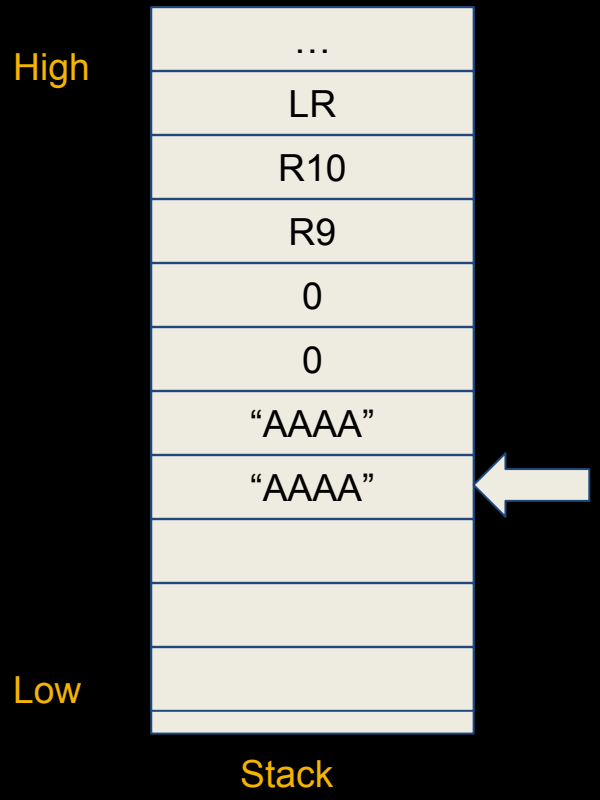

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



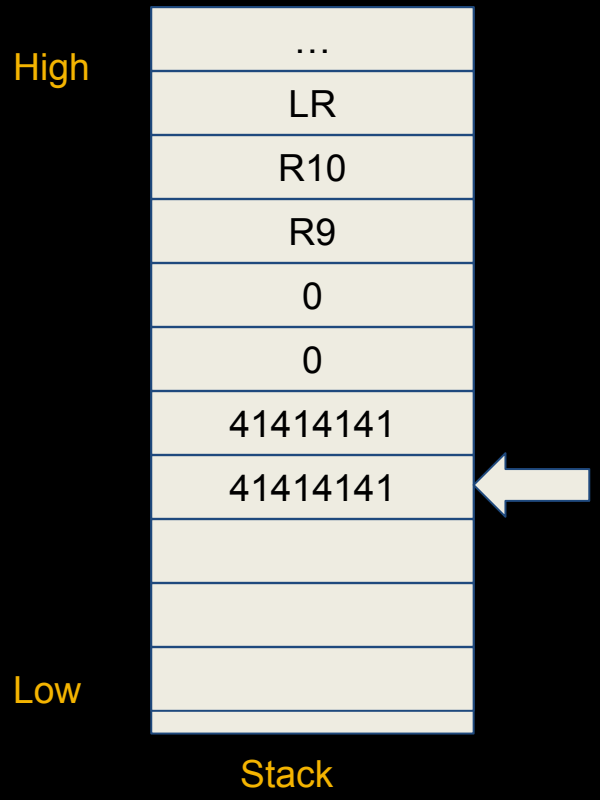

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



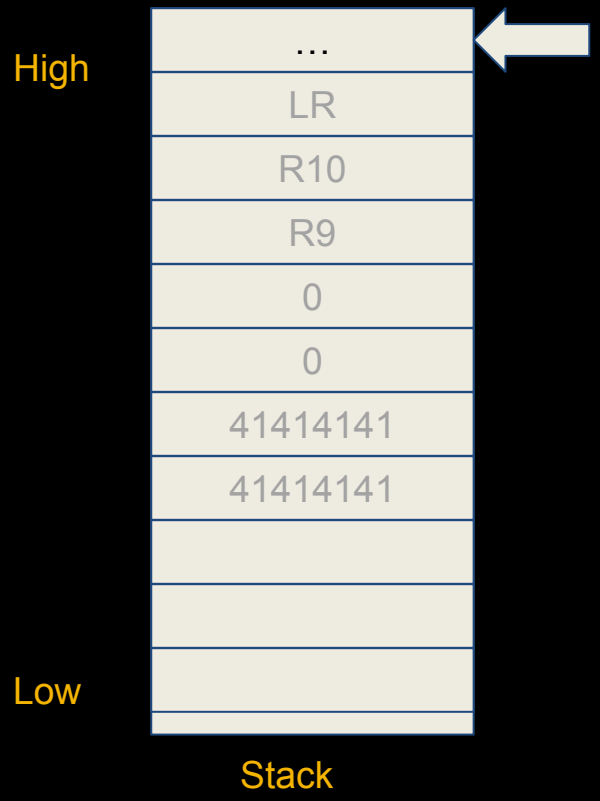

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



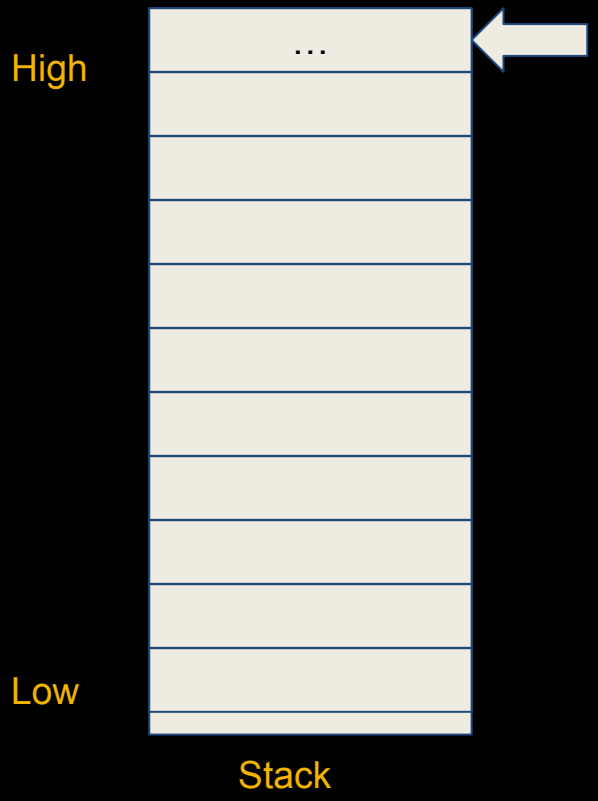

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



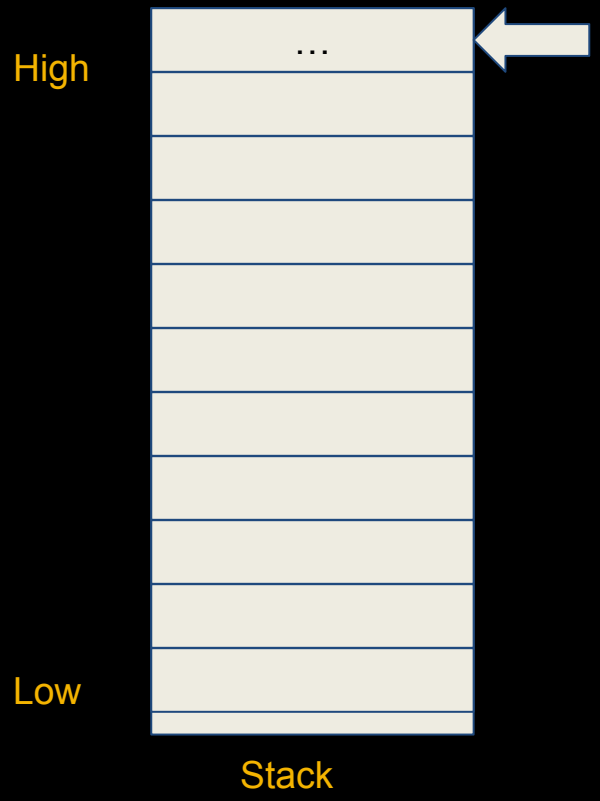

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



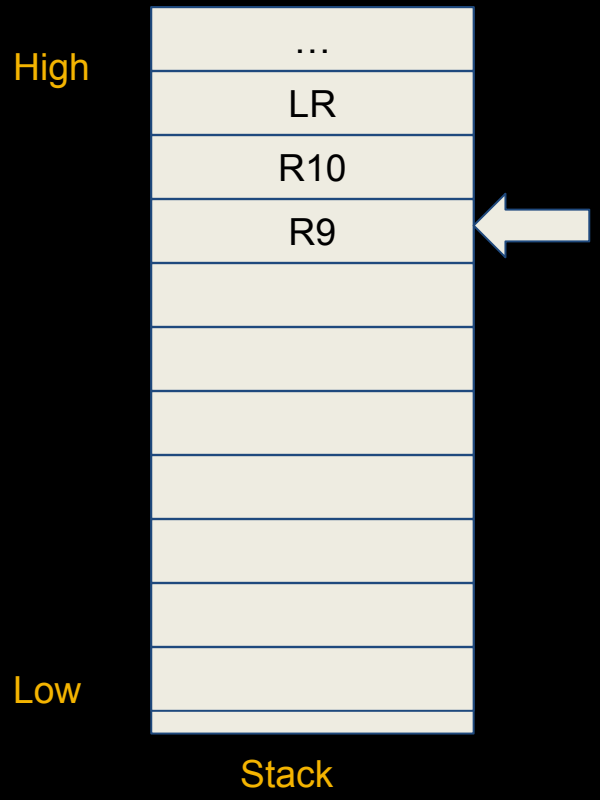

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



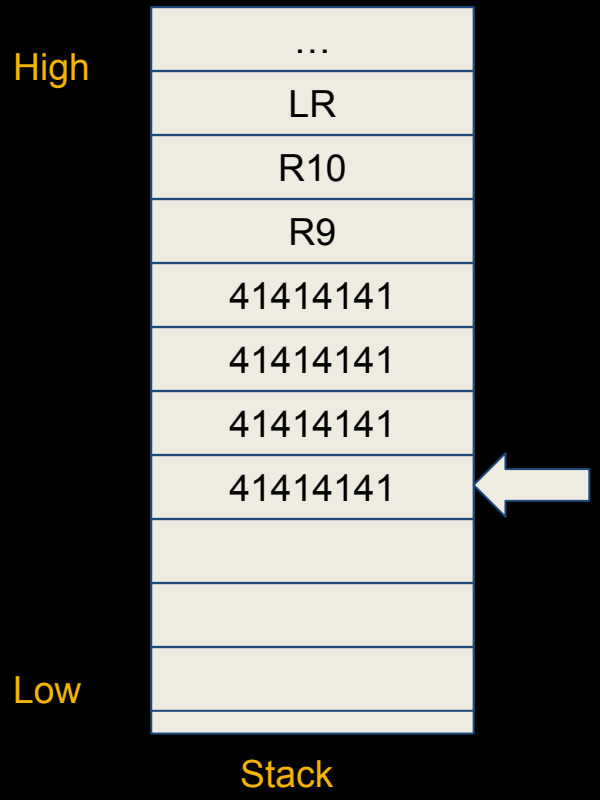

```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



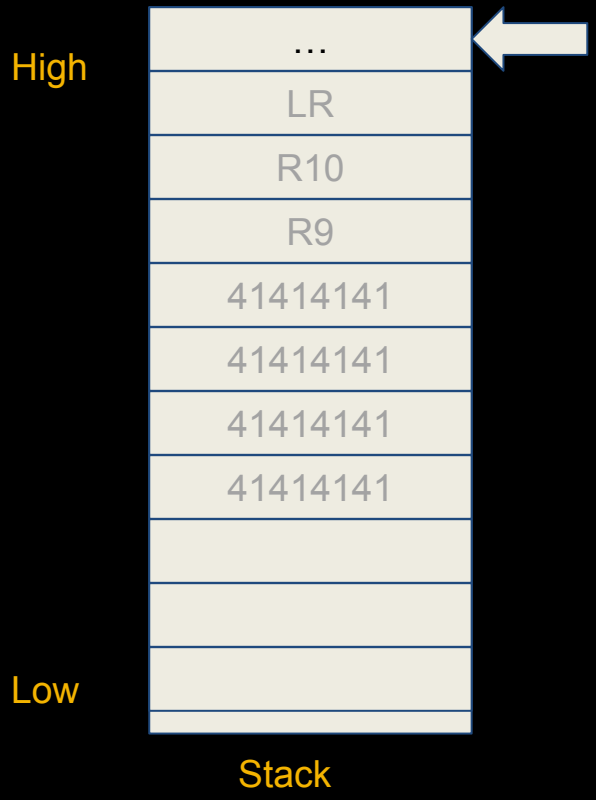

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



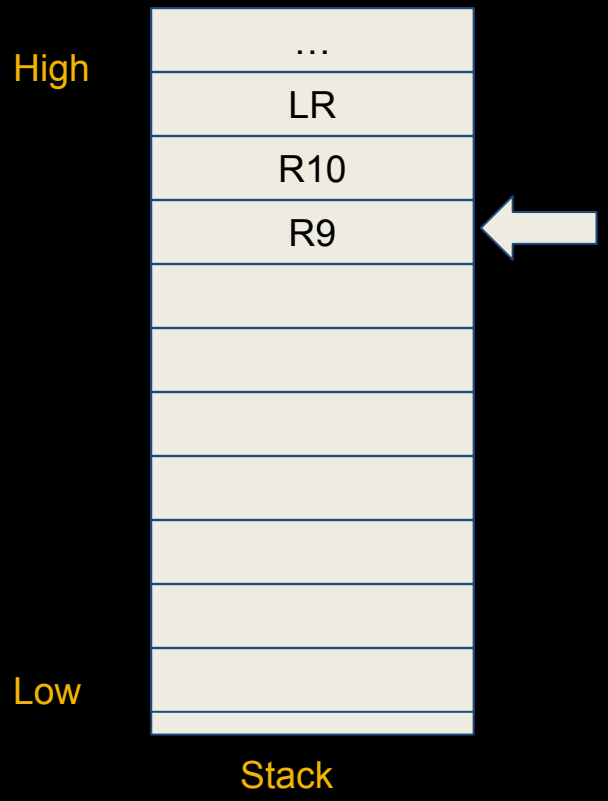

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



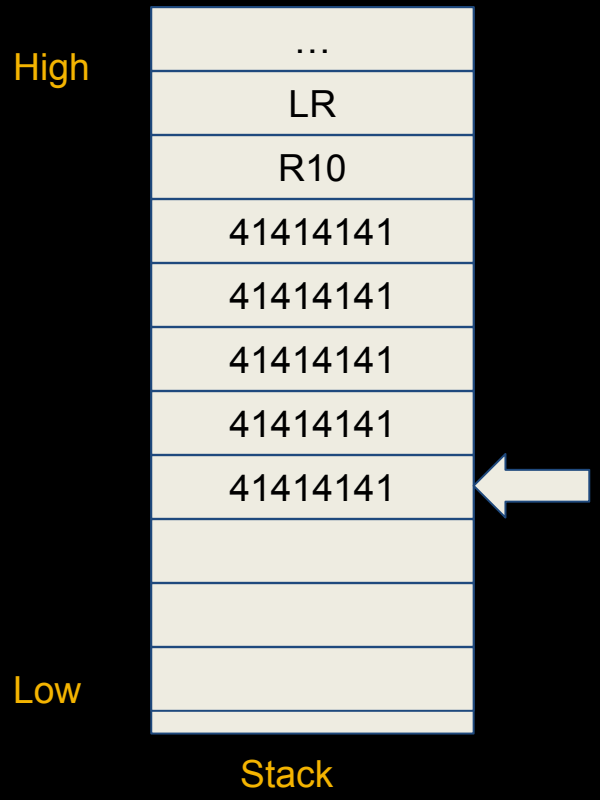

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



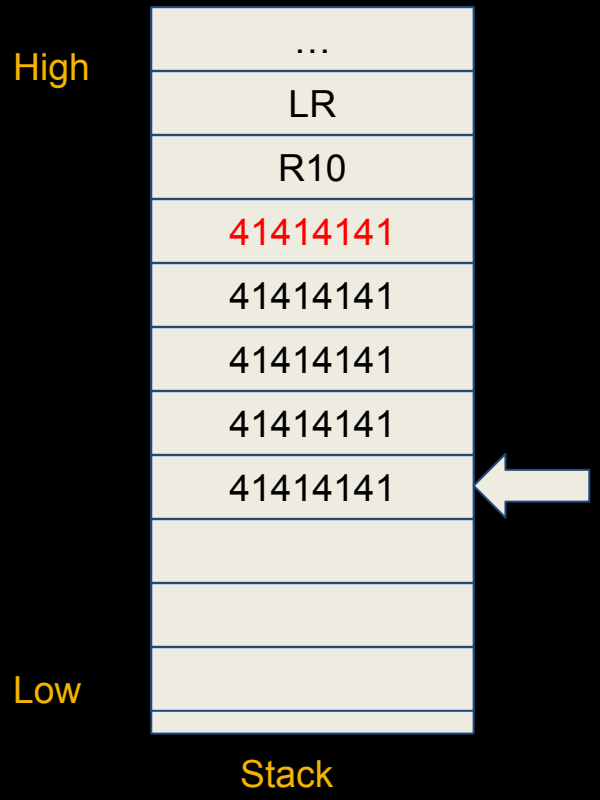

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



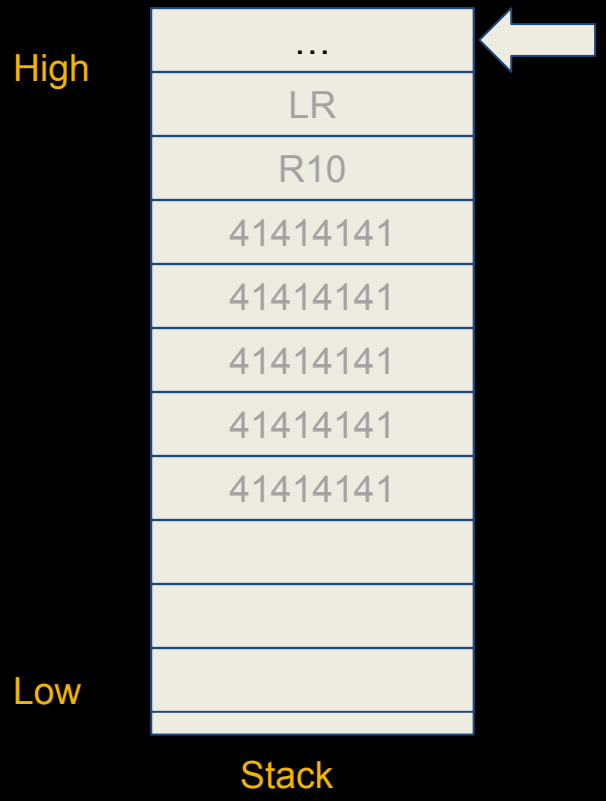

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



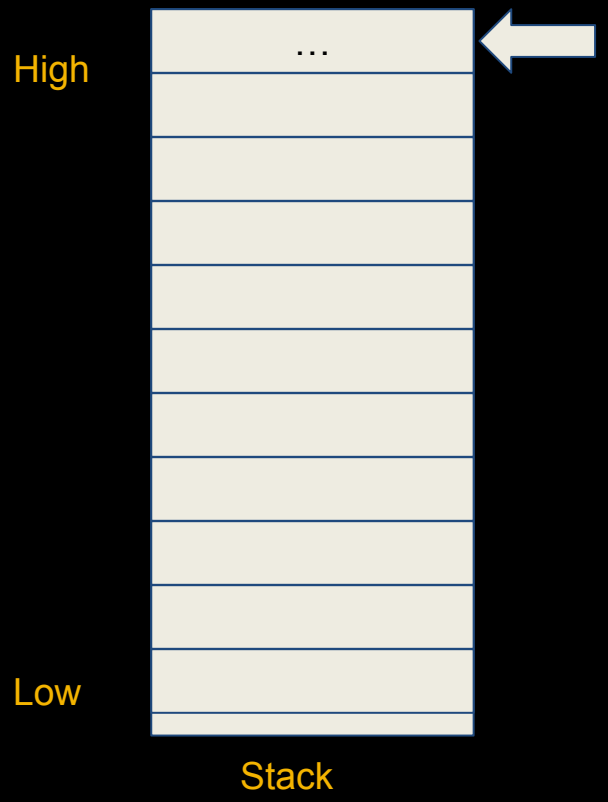

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



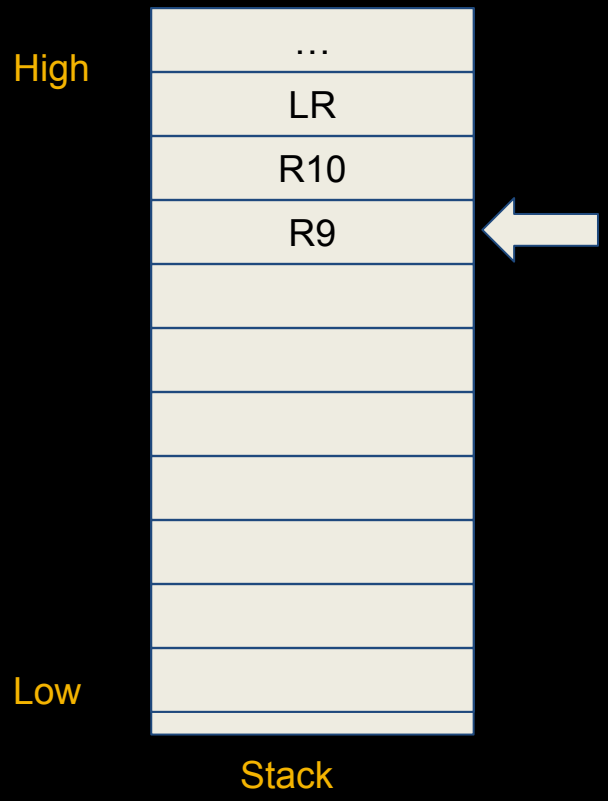

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



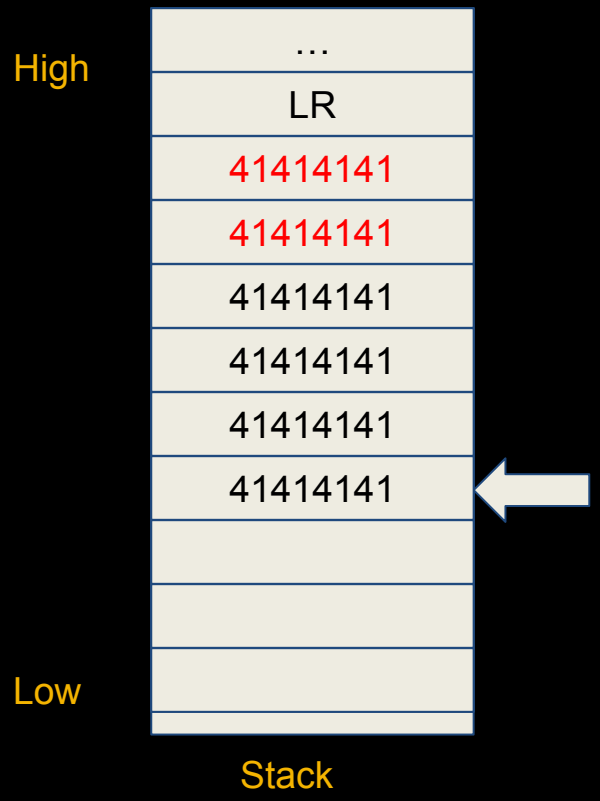

```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



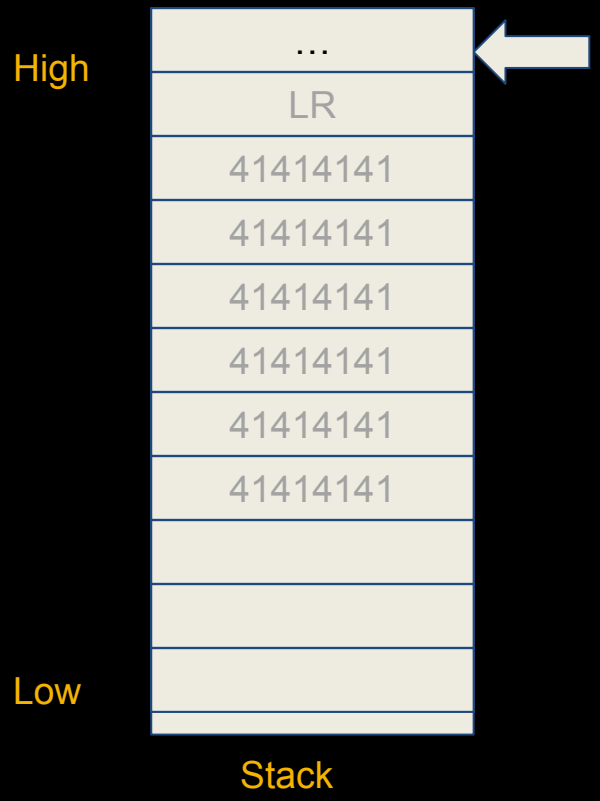

```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



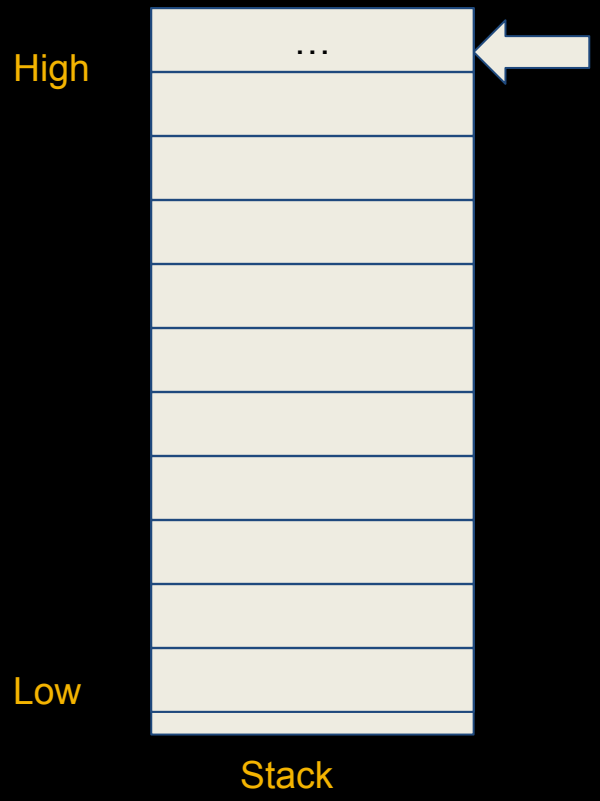

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



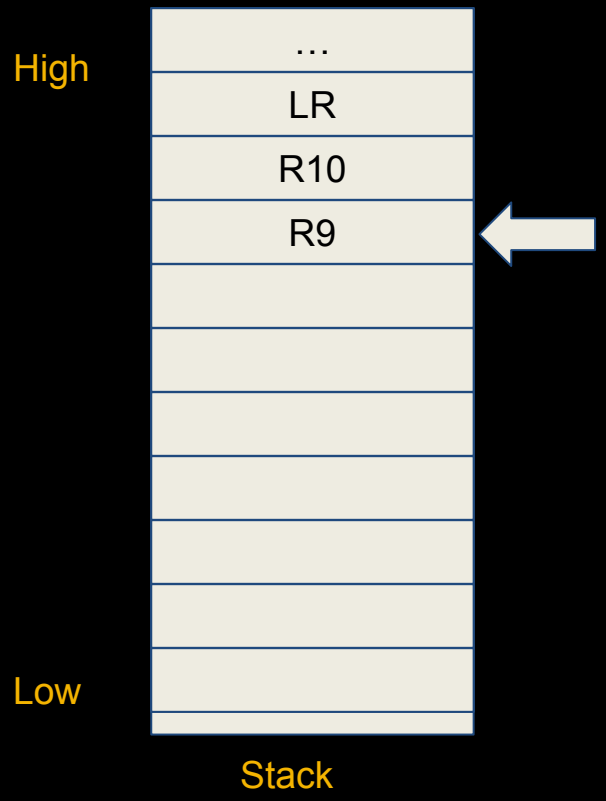

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



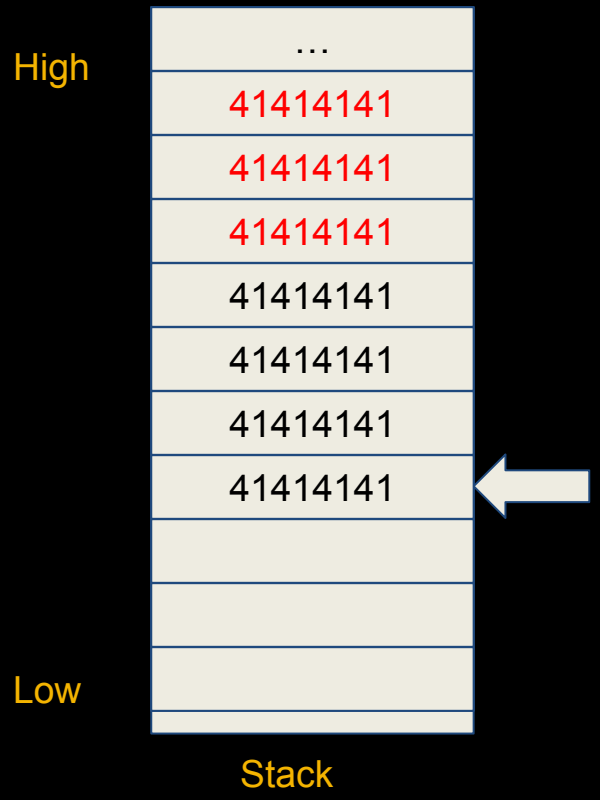

```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



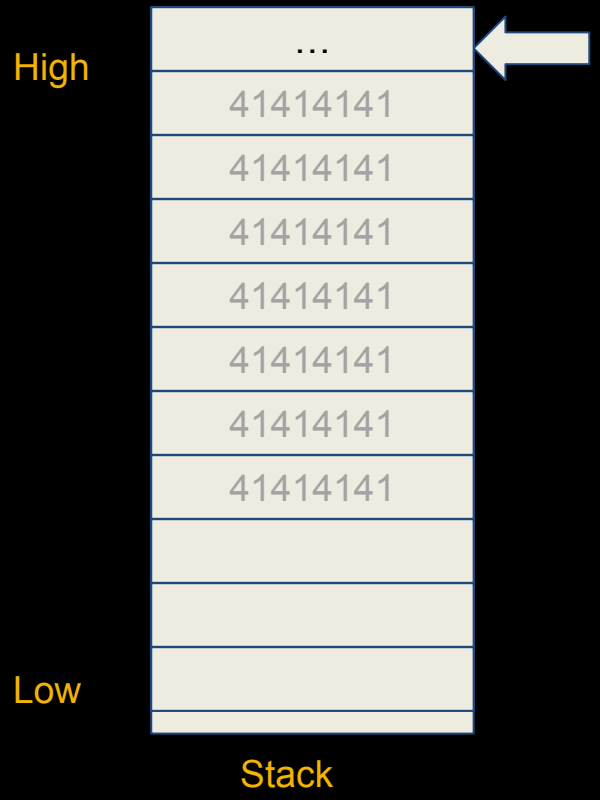

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



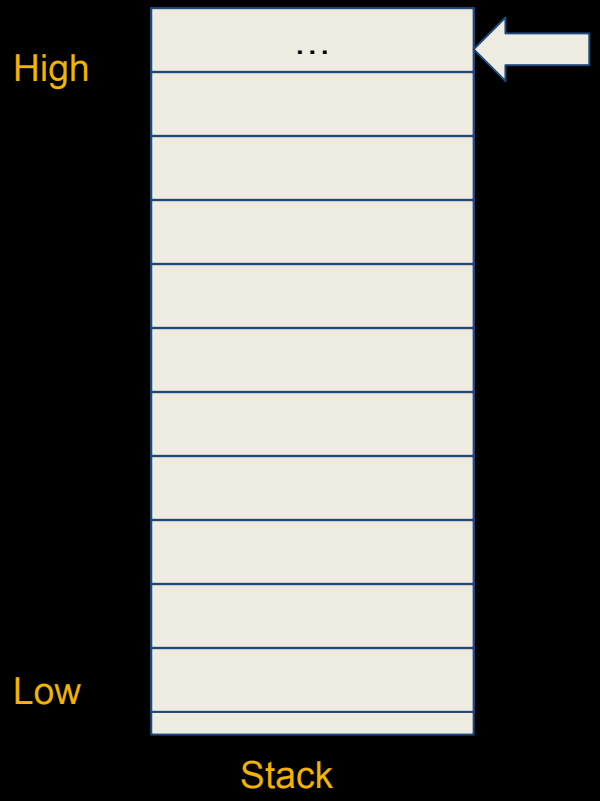

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



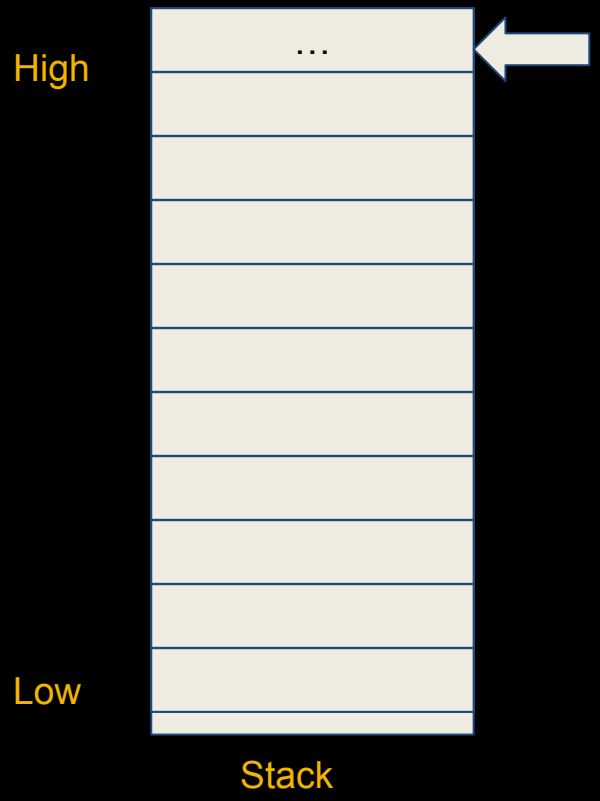

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow



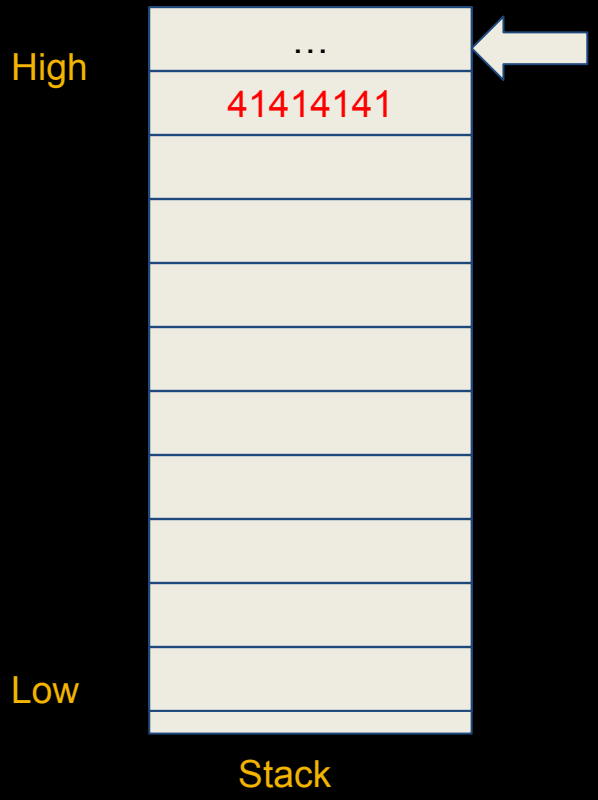

```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow




```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Stack overflow

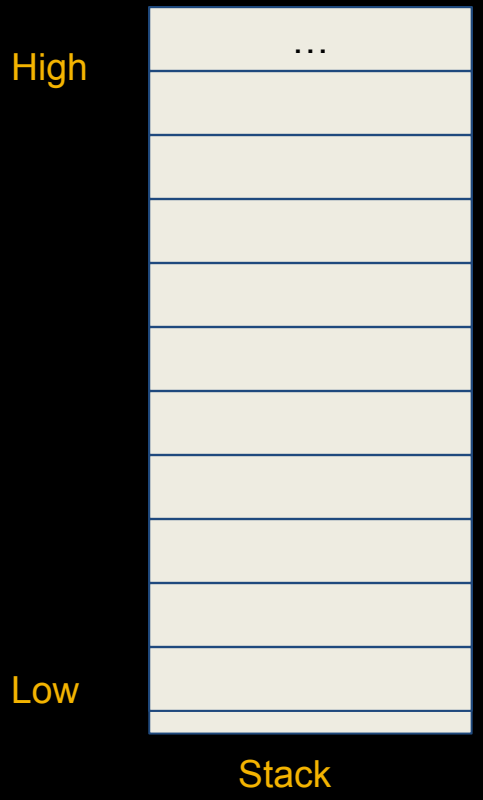



```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

emBO++

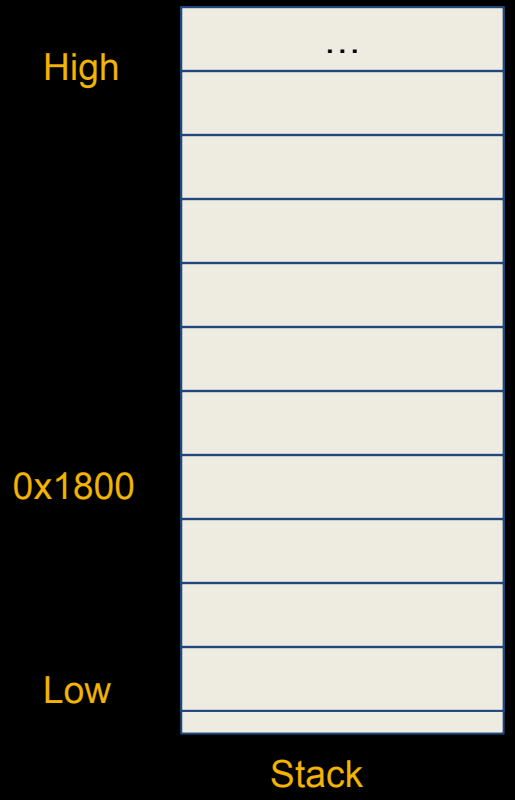
Hmm....

Controlled stack overflow



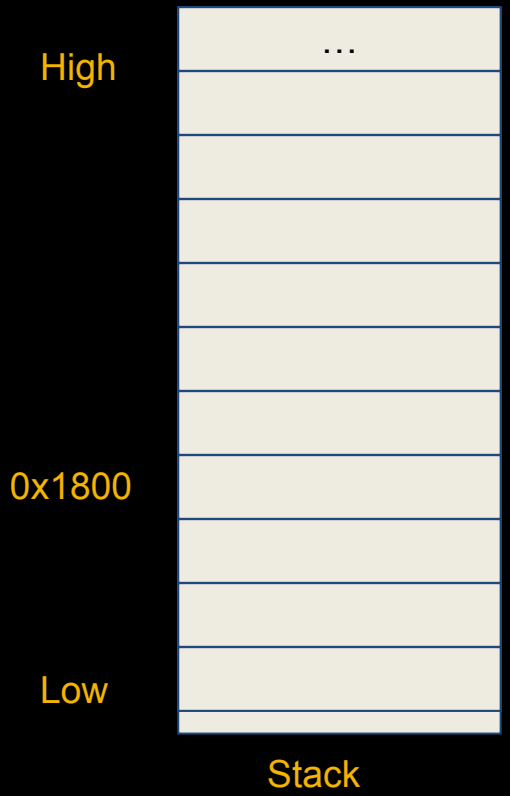

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Controlled stack overflow




```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

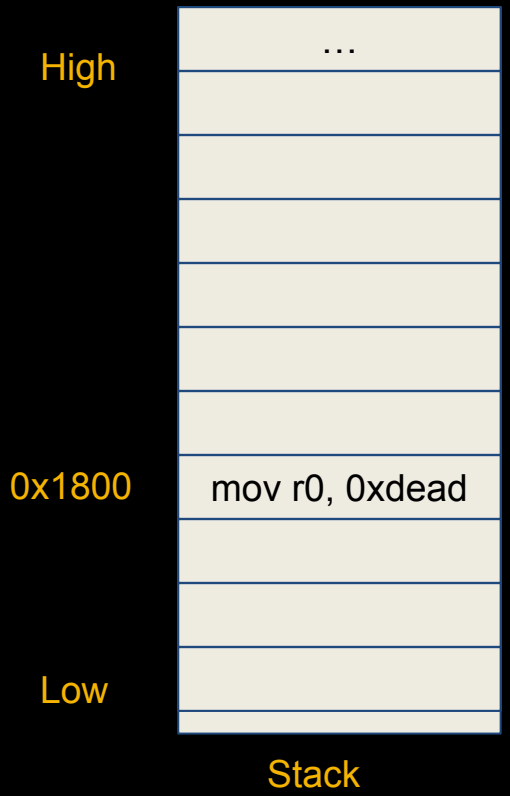
Controlled stack overflow



r0 = 0xdeadbeef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

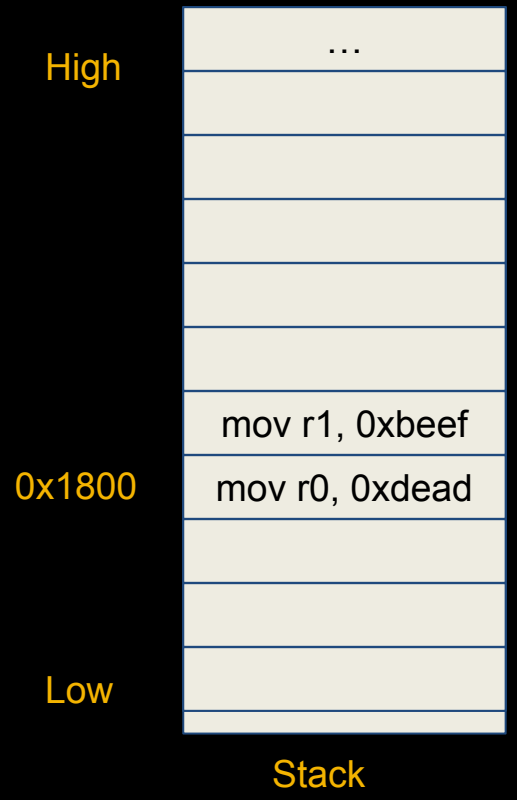
Controlled stack overflow



r0 = 0xdeadbeef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

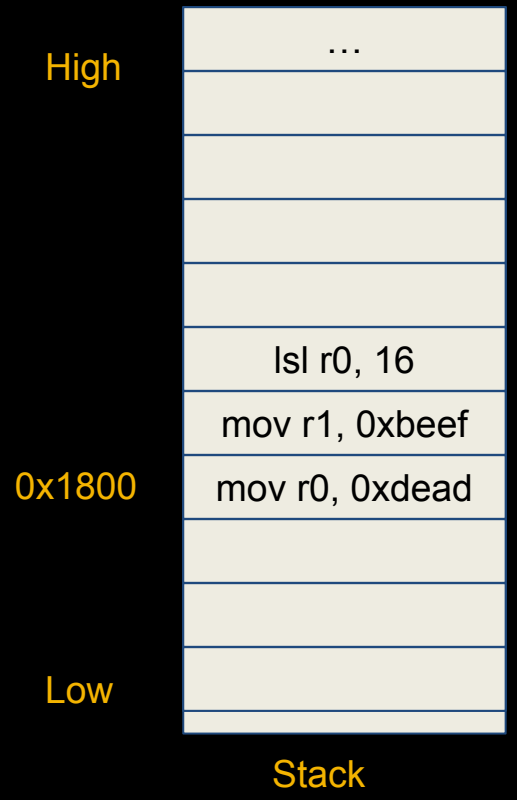
Controlled stack overflow



r0 = 0xdeadbeef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

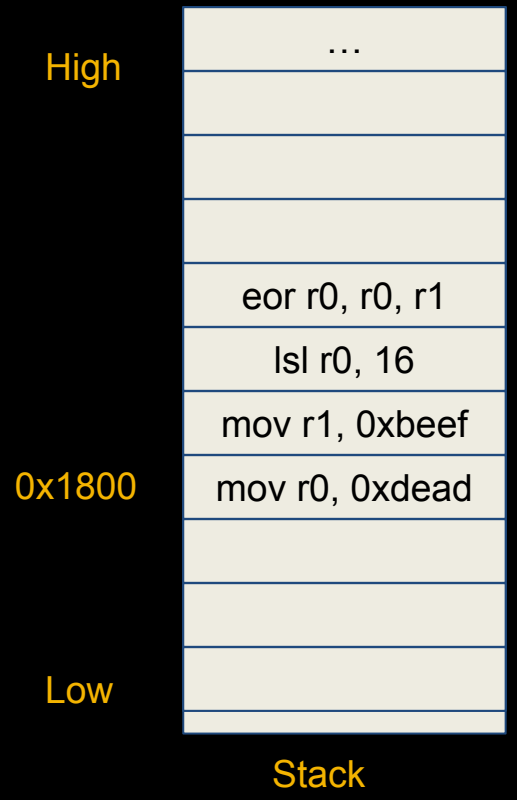
Controlled stack overflow



r0 = 0xdeadbeef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

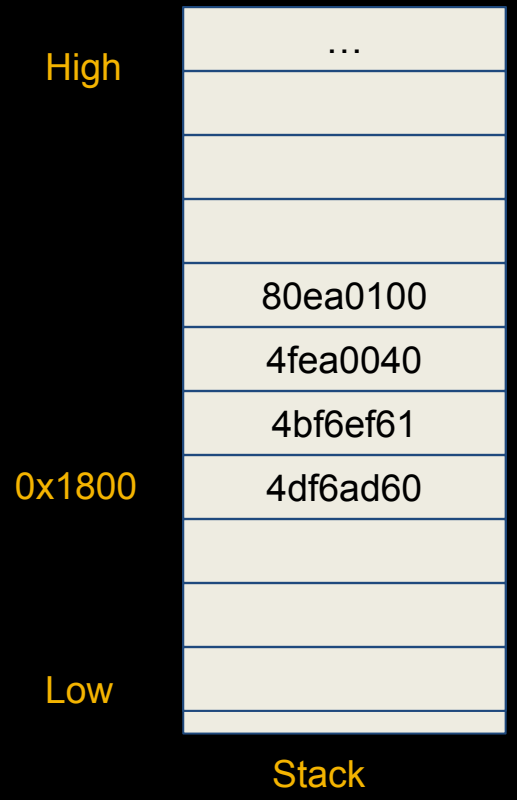
Controlled stack overflow



r0 = 0xdeadbeef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

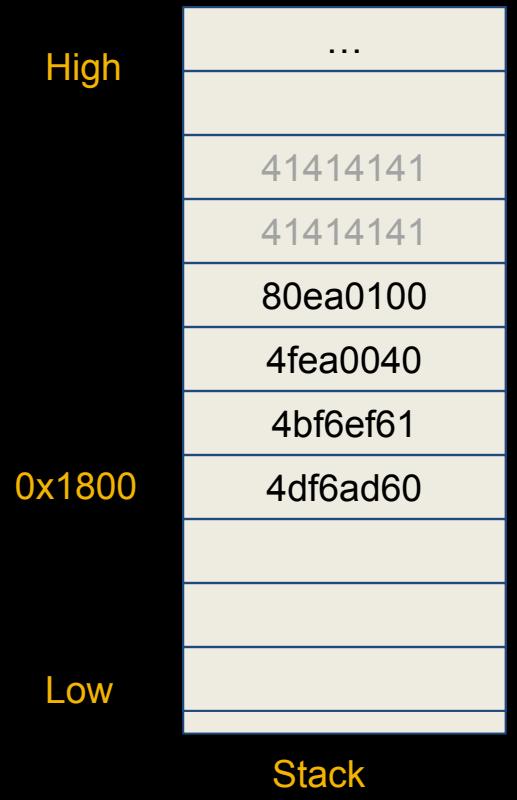
Controlled stack overflow



r0 = 0xdeadbeef

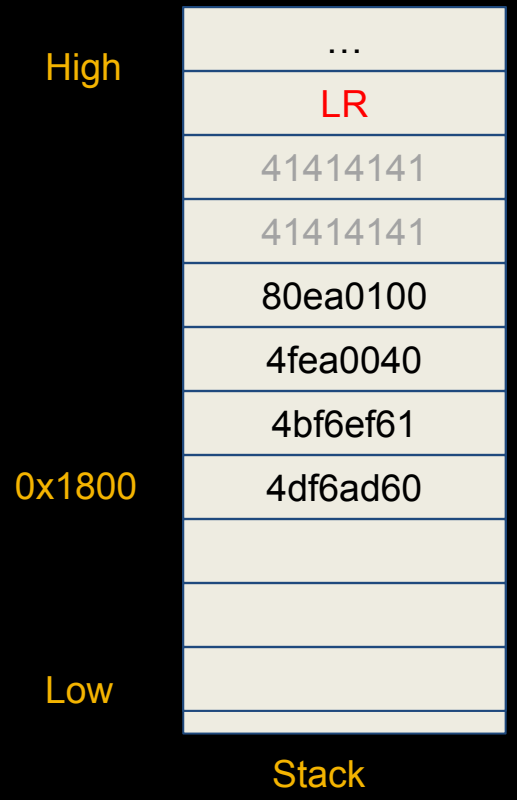
```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Controlled stack overflow




```c
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

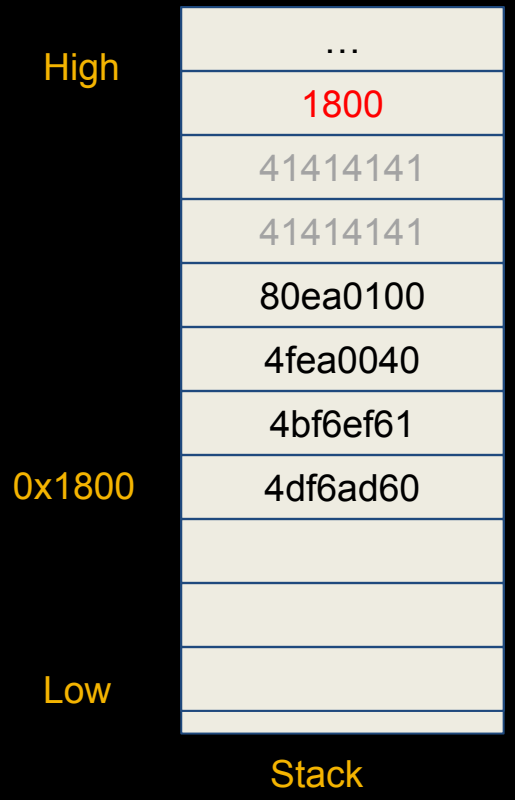
Controlled stack overflow




```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

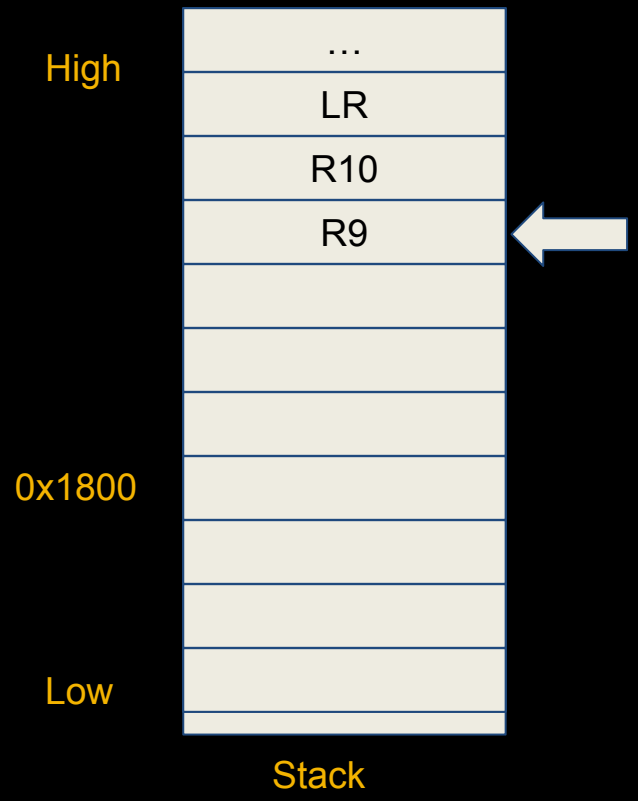
Controlled stack overflow

emBO++



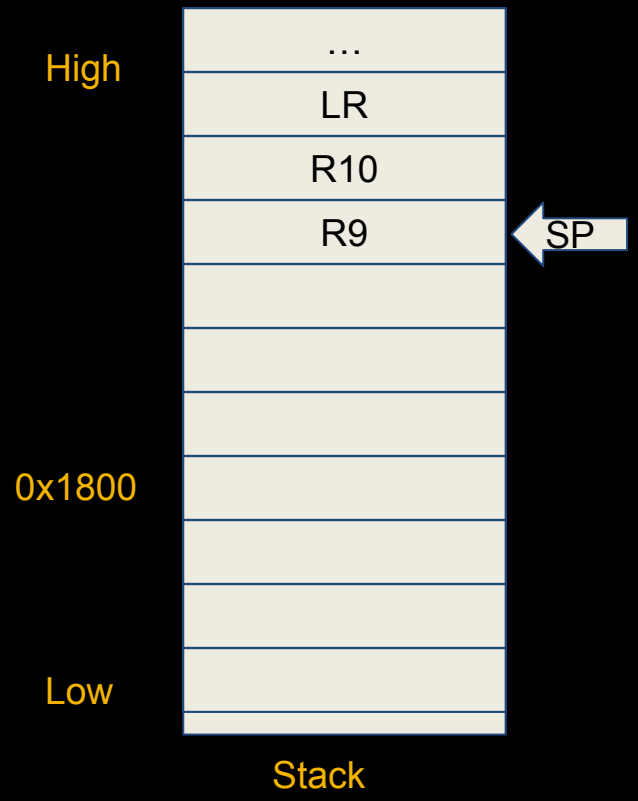

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Controlled stack overflow




```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

Controlled stack overflow

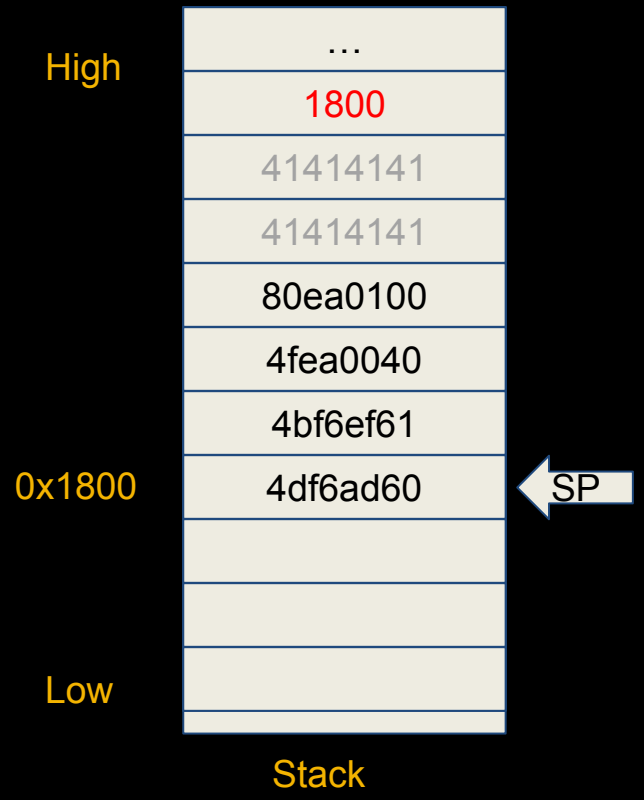



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

PC

LR

Controlled stack overflow

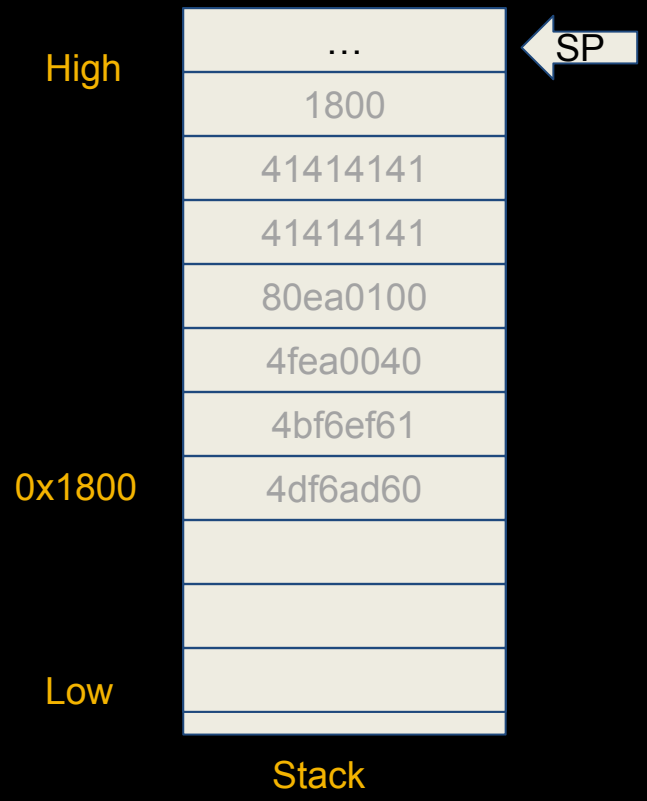



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

PC

LR

Controlled stack overflow

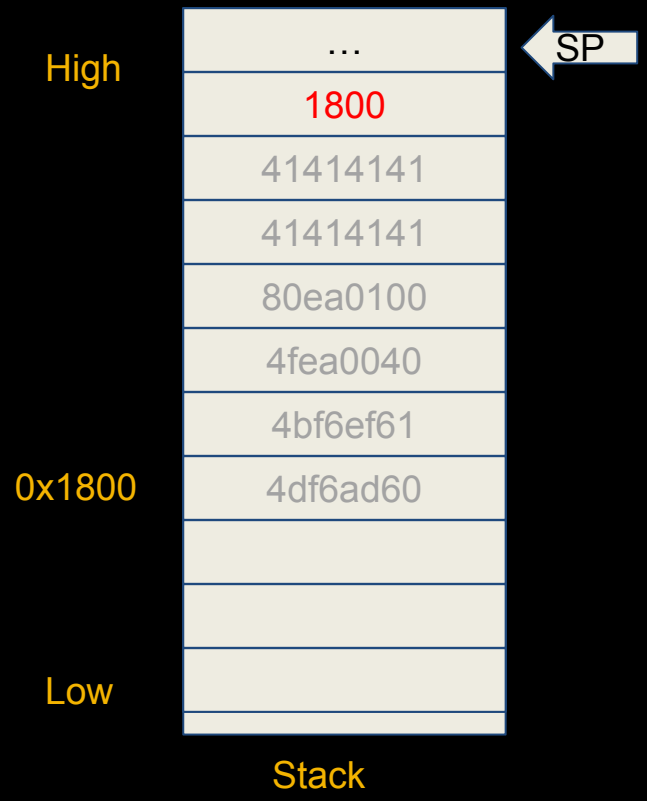



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

PC

LR

Controlled stack overflow

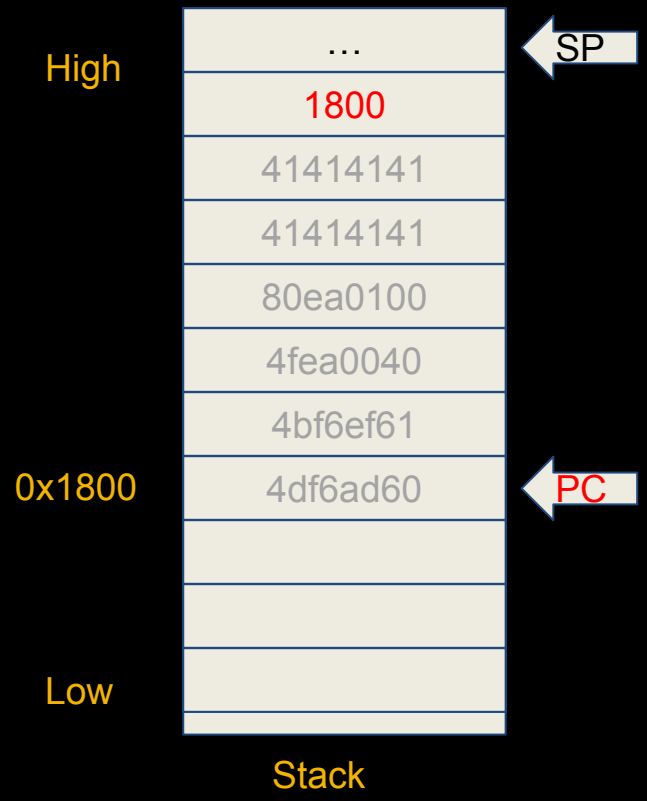



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

PC

LR

Controlled stack overflow

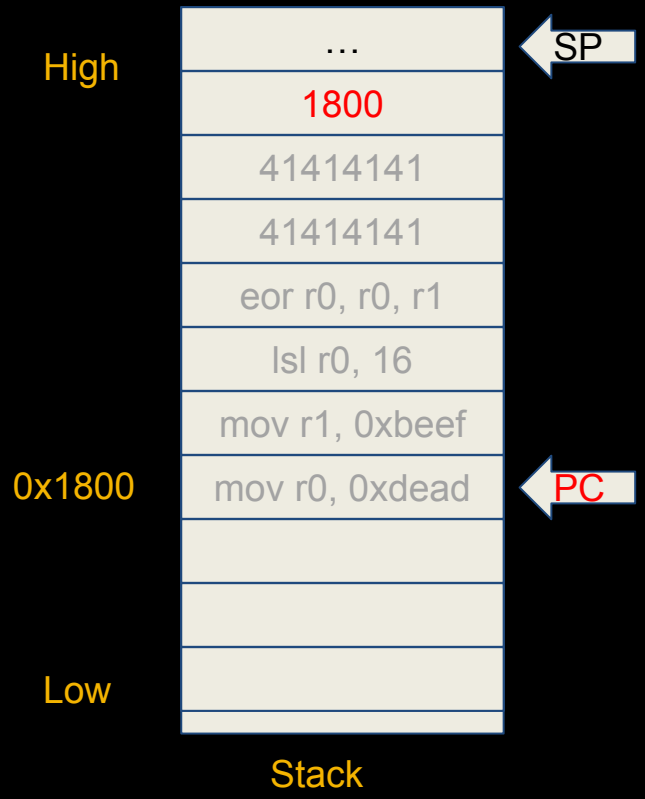



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Controlled stack overflow

emBO++

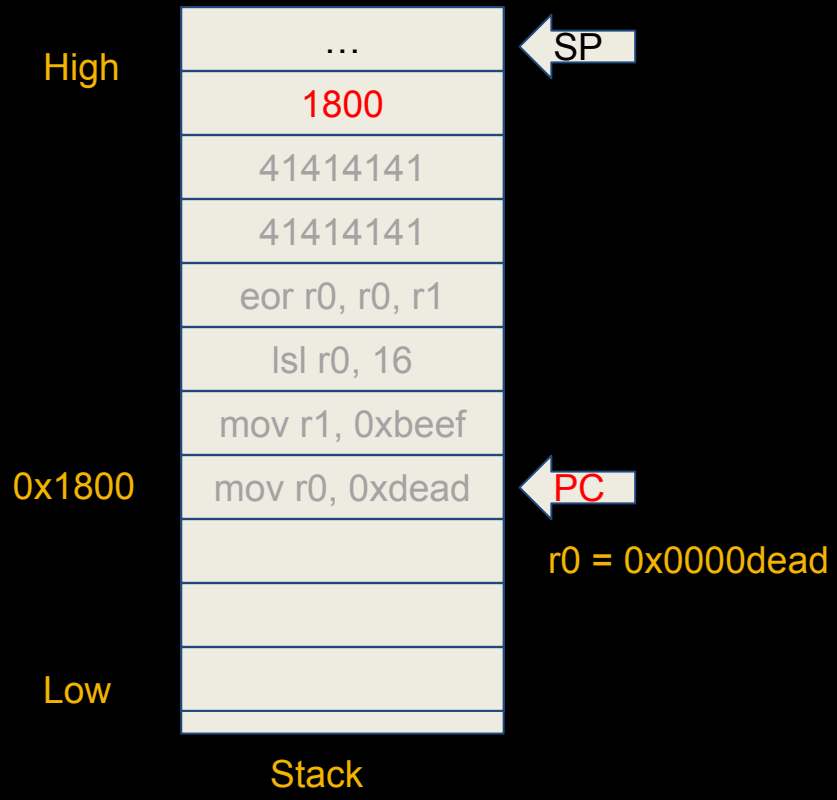



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Controlled stack overflow

emBO++

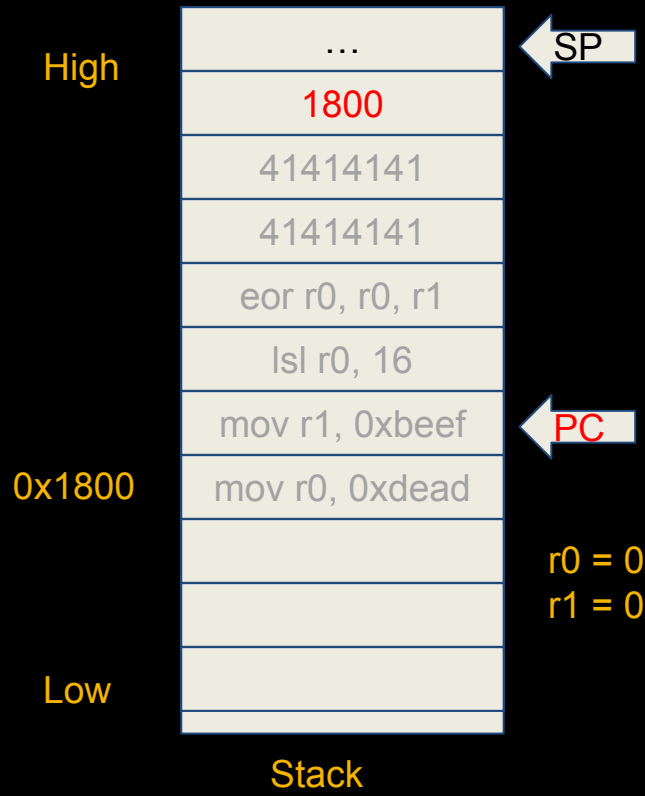



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Controlled stack overflow

emBO++



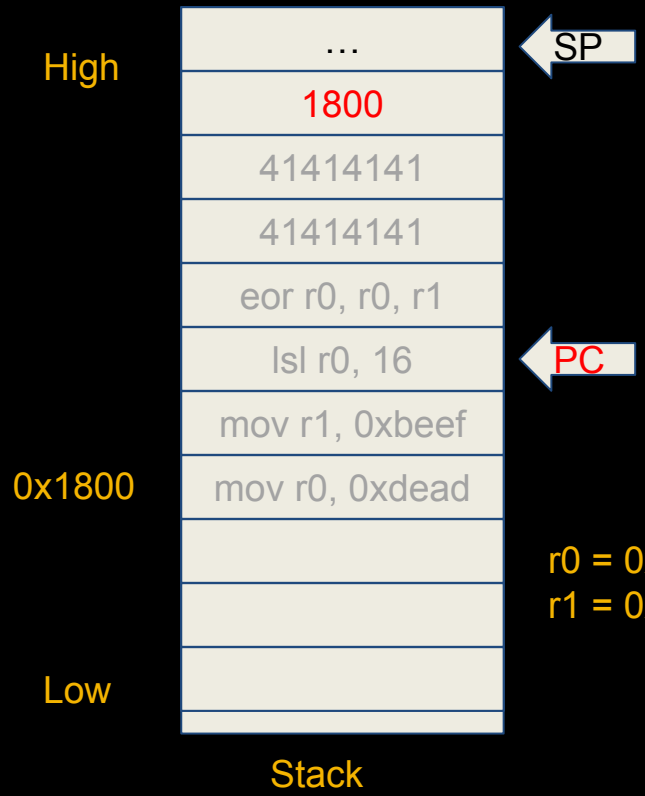
r0 = 0x0000dead
r1 = 0x0000beef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Controlled stack overflow

emBO++



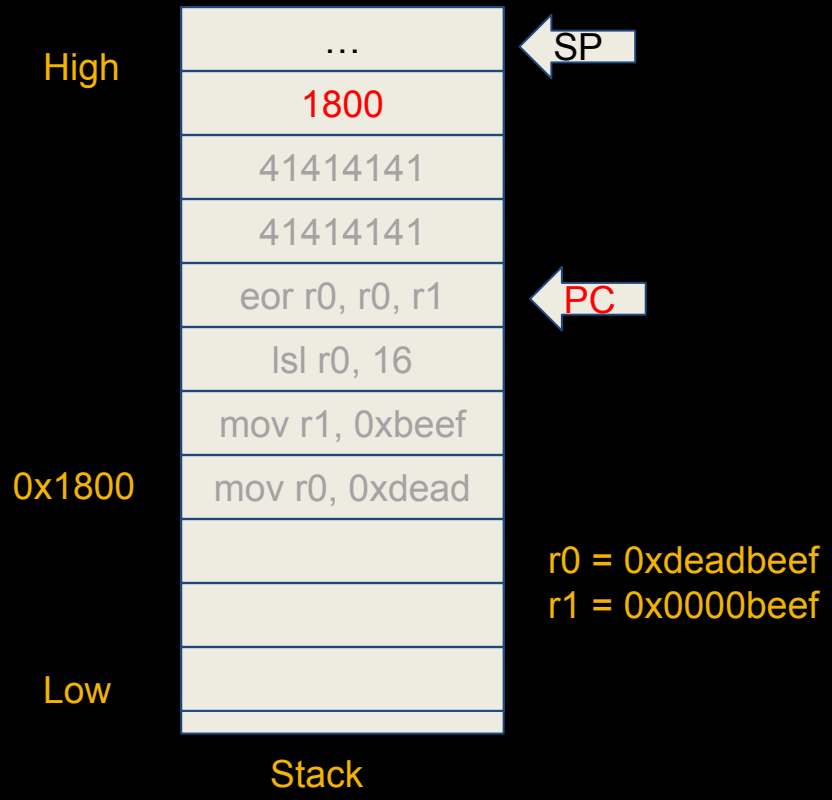
r0 = 0xdead0000
r1 = 0x0000beef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Controlled stack overflow

emBO++



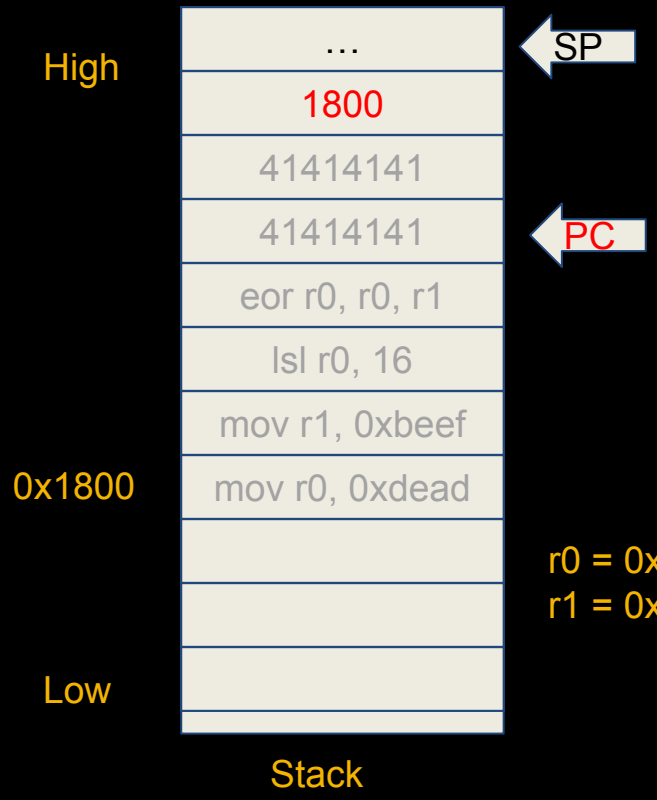
r0 = 0xdeadbeef
r1 = 0x0000beef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Controlled stack overflow

emBO++



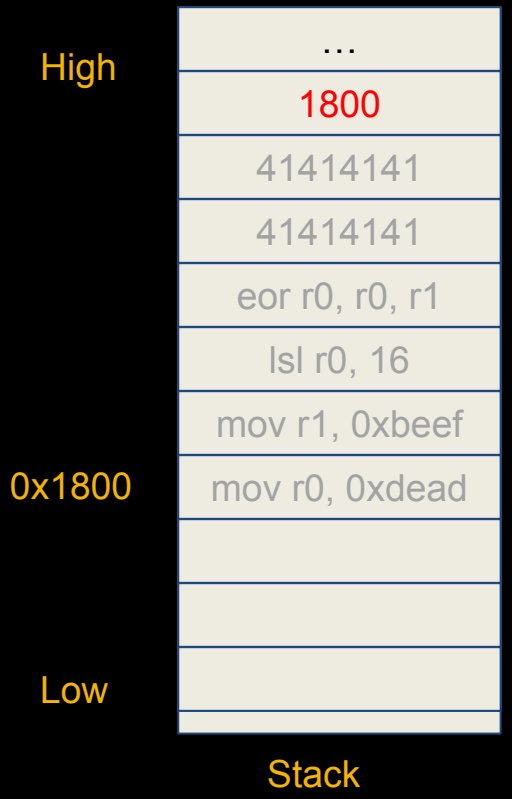
r0 = 0xdeadbeef
r1 = 0x0000beef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Controlled stack overflow

emBO++




```
char* buf, size_t len) {
buf[16] = {0};
y_buf, buf, len);

char* aaa_buf) {
aa_buf, 8);
aa_buf, 16);
aa_buf, 20);
aa_buf, 24);
callee(aaa_buf, 28);
}
```

LR

r0 = 0xdeadbeef
r1 = 0x0000beef

Exploiting things

1. Dump it ✔
2. Analyze it ✔
3. Find it ✔
4. Exploit it ✔

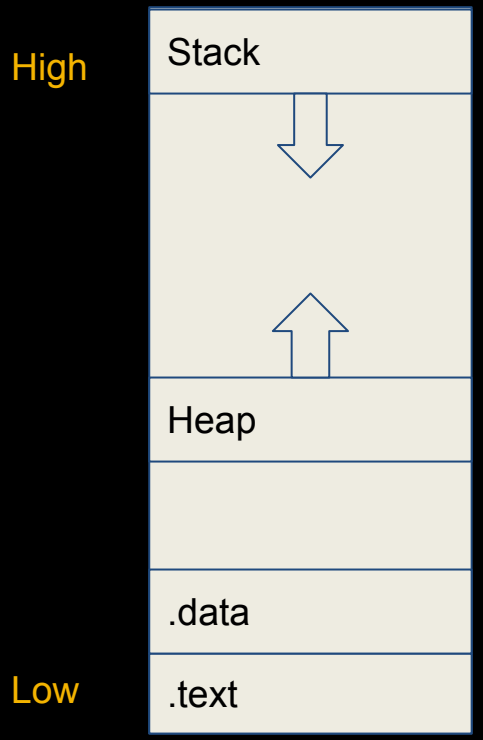
Breaking Exploiting things

emBO++

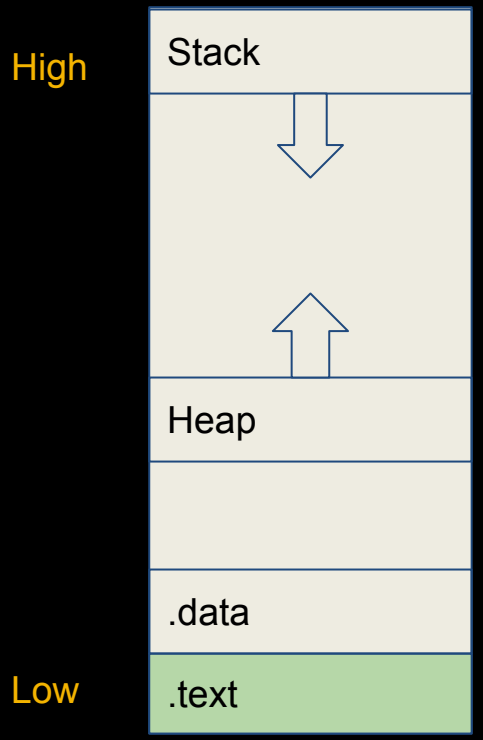
1. Dump it ✔
2. Analyze it ✔
3. Find it ✔
4. Exploit it ✔

Mitigation I: NX

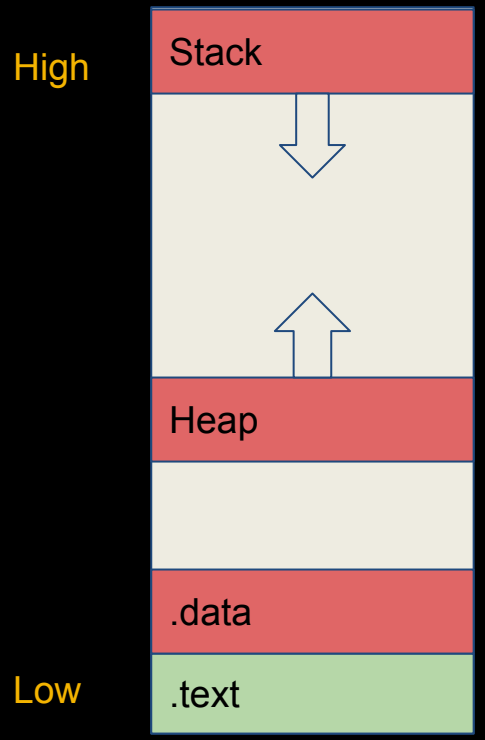
No eXecute



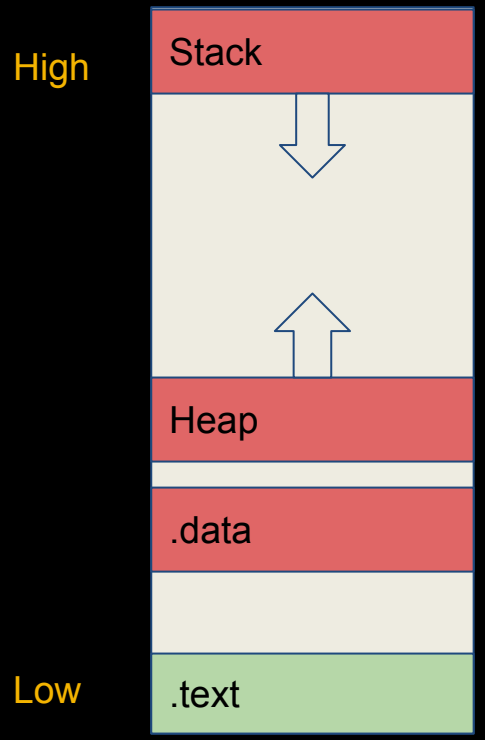
No eXecute



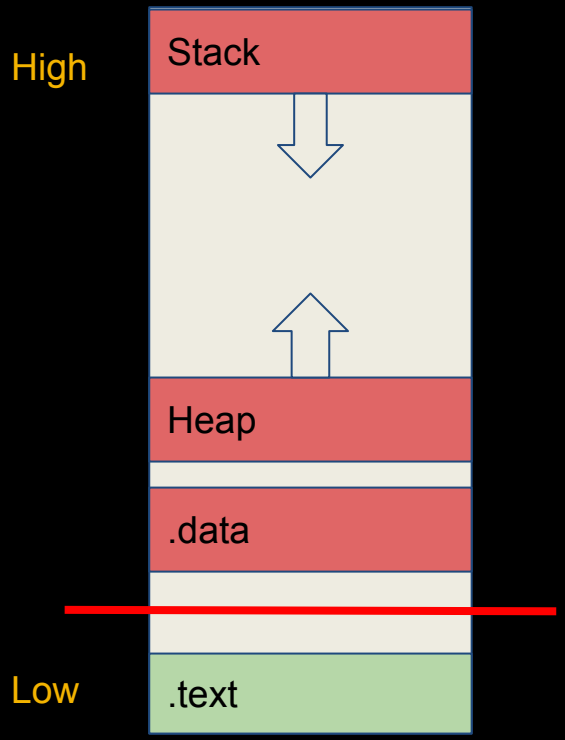
No eXecute



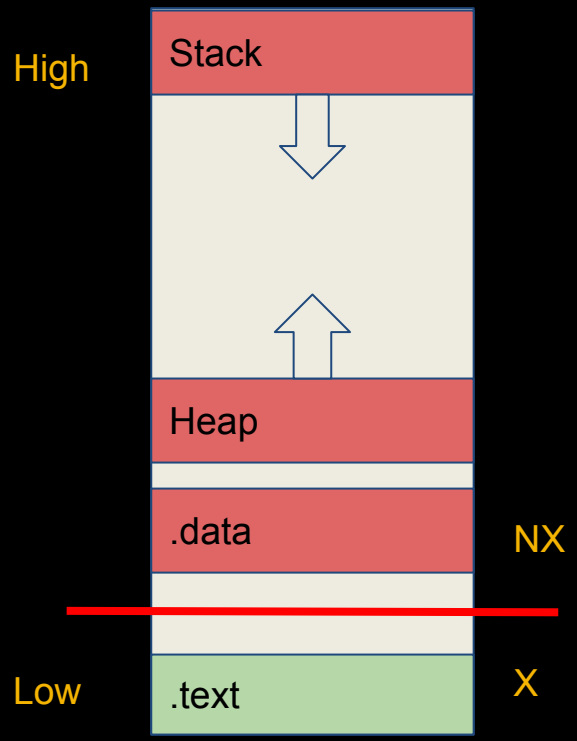
No eXecute



No eXecute

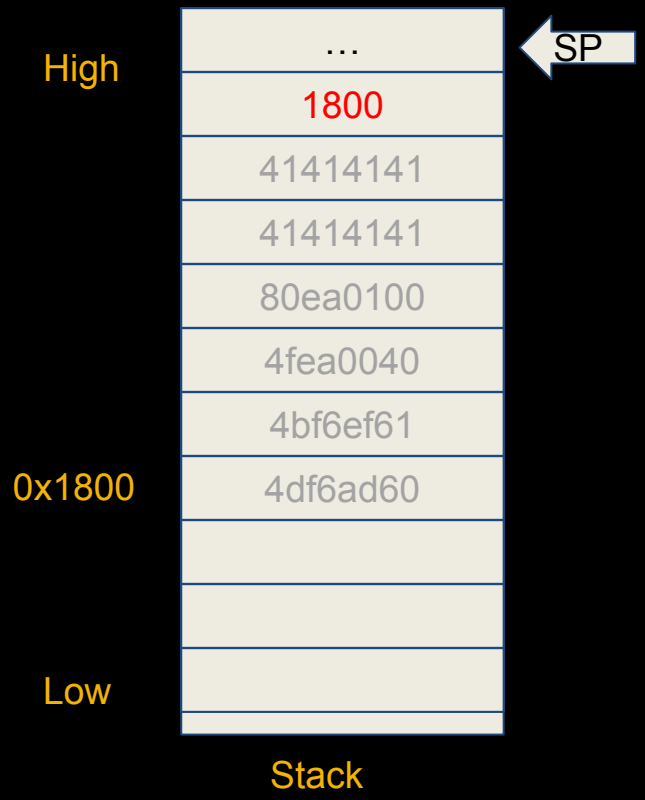


No eXecute



No eXecute

emBO++

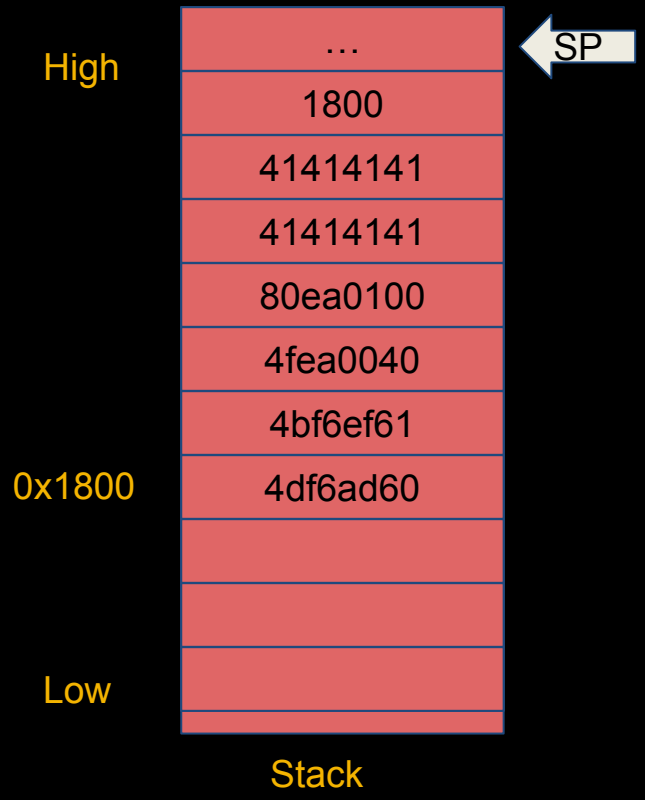



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

PC

LR

No eXecute

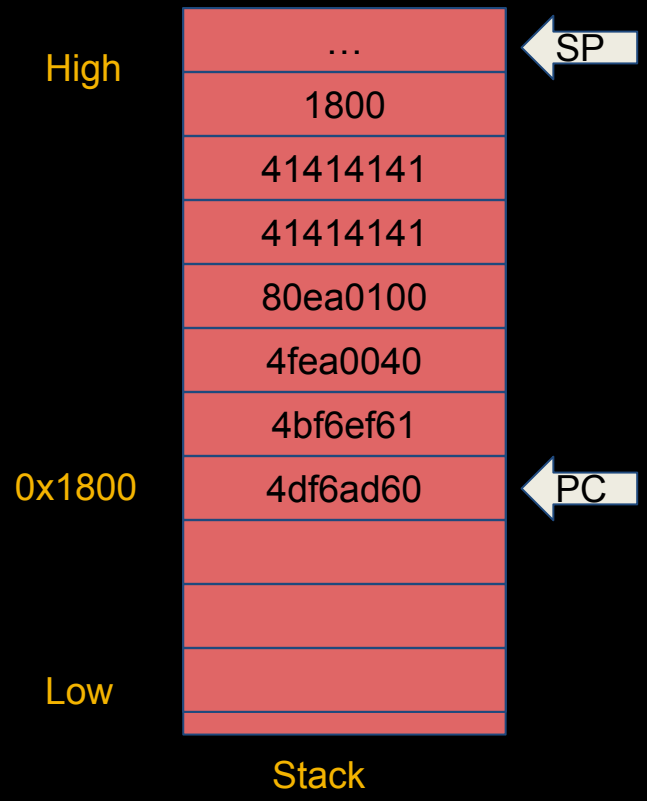



```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

PC

LR

No eXecute




```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

No eXecute


```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 8);
    callee(aaa_buf, 16);
    callee(aaa_buf, 20);
    callee(aaa_buf, 24);
    callee(aaa_buf, 28);
}
```

LR

Breaking Exploiting things

emBO++

1. Dump it ✔
2. Analyze it ✔
3. Find it ✔
4. Exploit it ✔

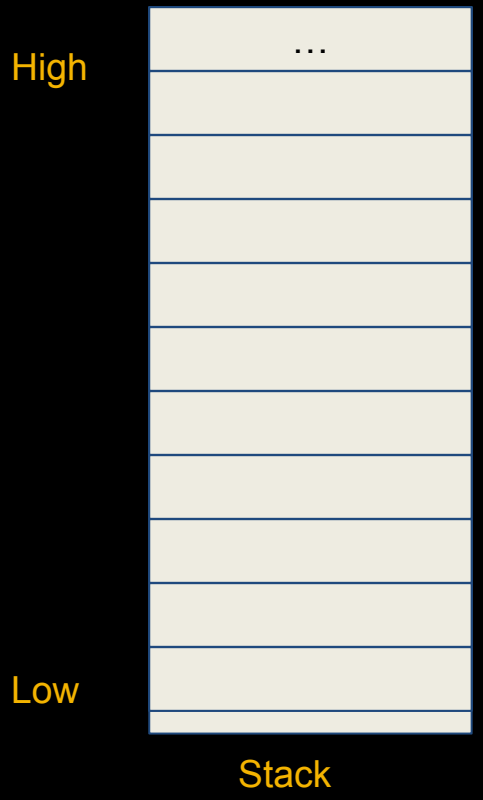
Breaking Breakin gs

1. Dump it ✔
2. Analyze it ✔
3. Find it ✔
4. Exploit it ✔

emBO++

Stack buffer overflow II

Return of the Operation



r0 = 0xdeadbeef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

Return of the Operation

r0 = 0xdeadbeef

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

Return of the Operation

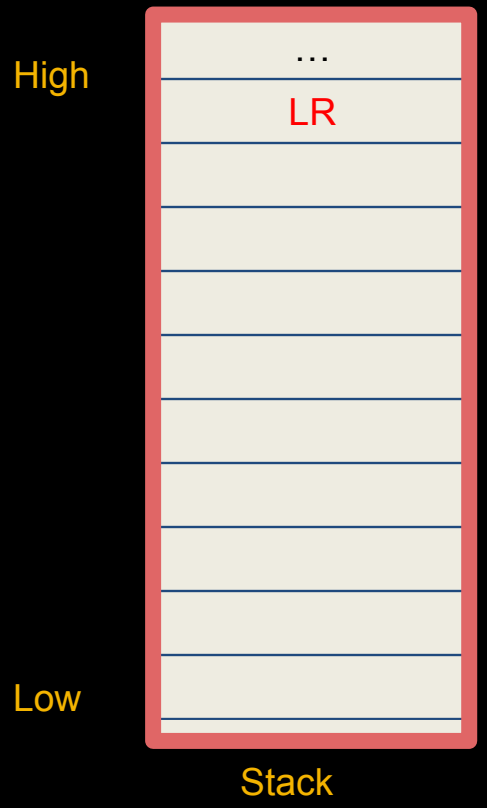
r0 = 0xdeadbeef

0x800

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

r0 = 0xdeadbeef

Return of the Operation



0x800

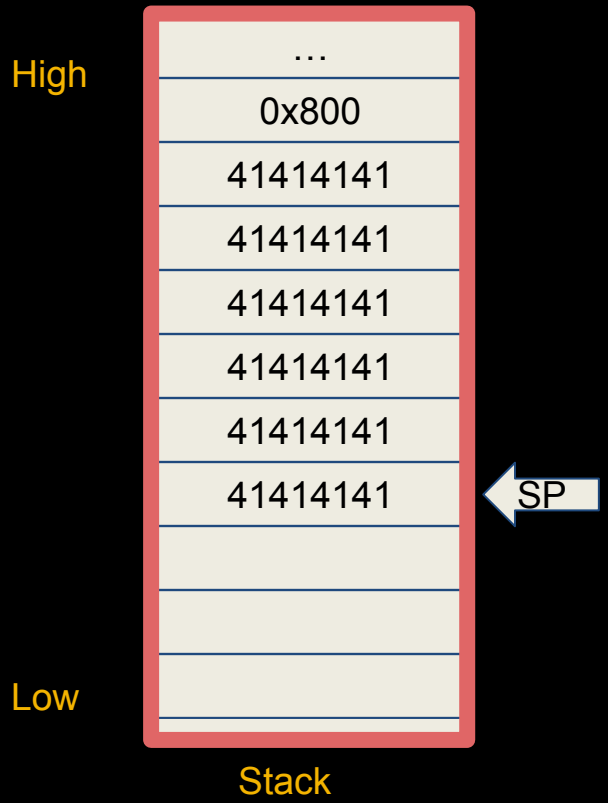
```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

Return of the Operation

0x800

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

Return of the Operation



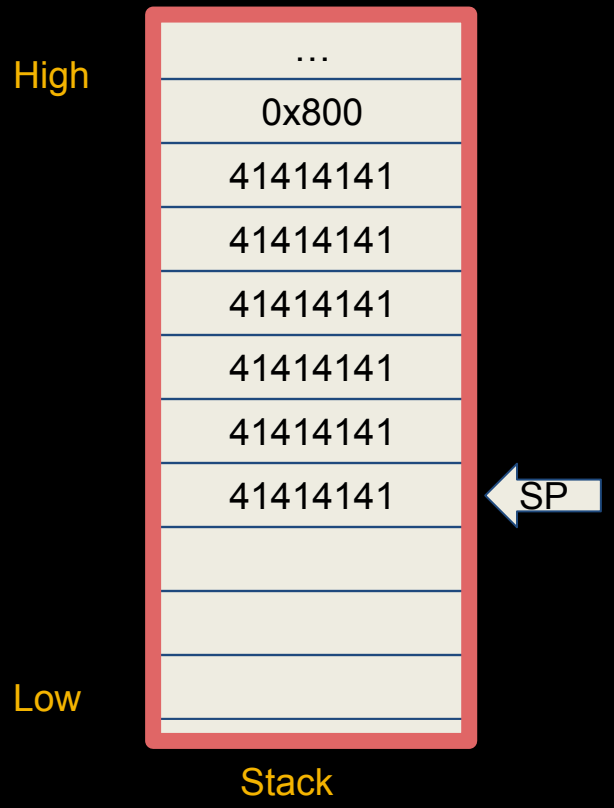
0x800

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

PC

LR

Return of the Operation



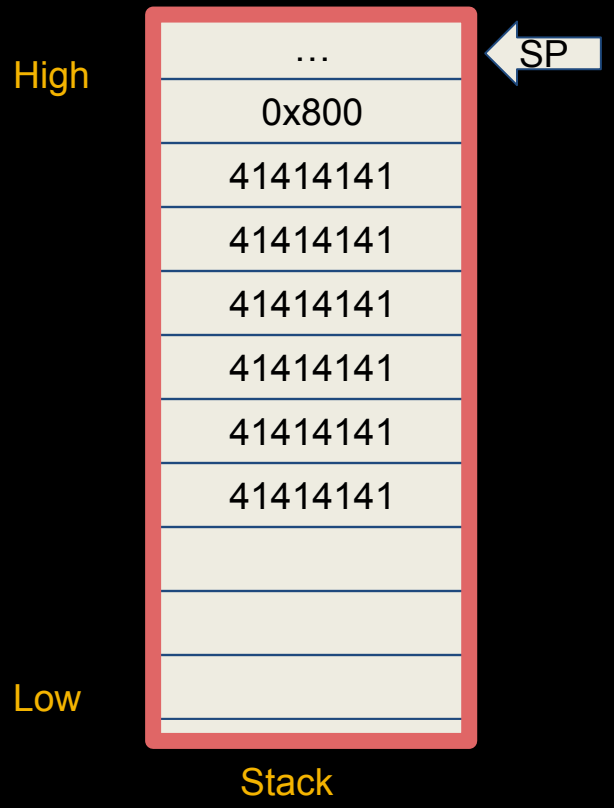
0x800

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

PC

LR

Return of the Operation



0x800

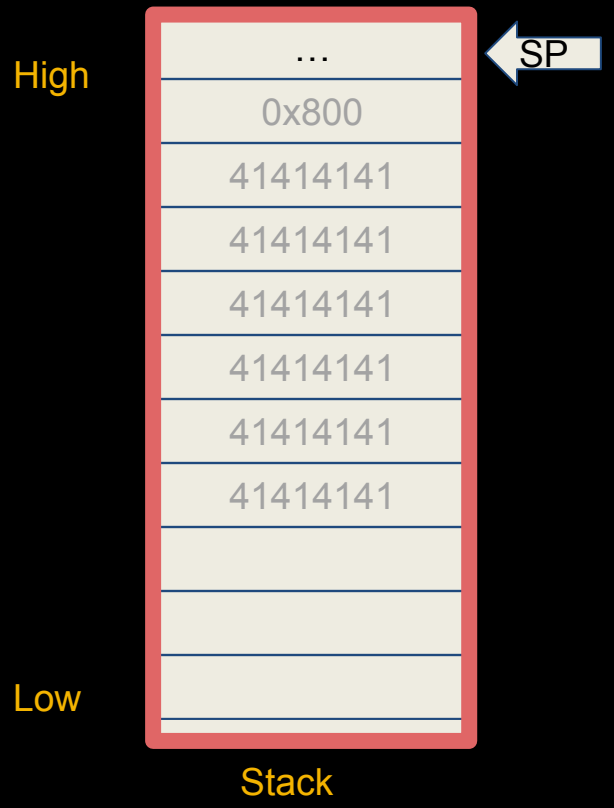
```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

PC

LR

Return of the Operation

emBO++



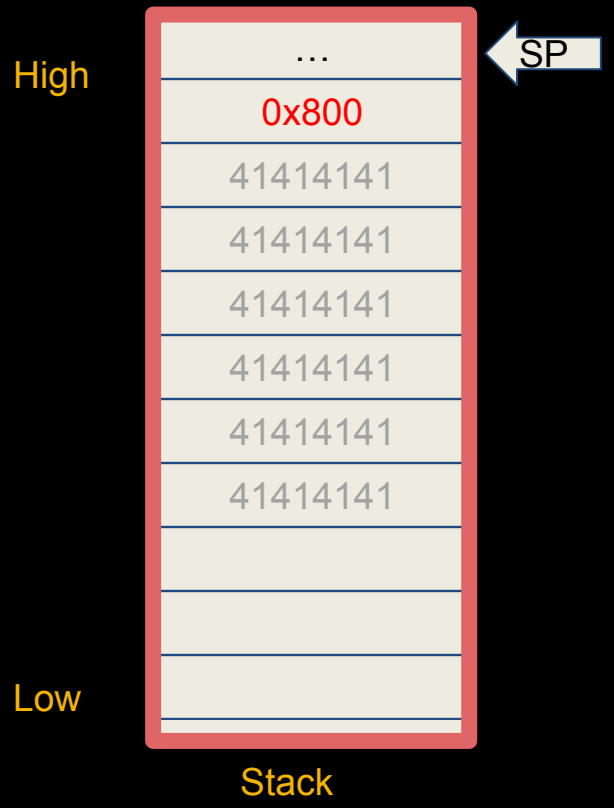
0x800

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

PC

LR

Return of the Operation



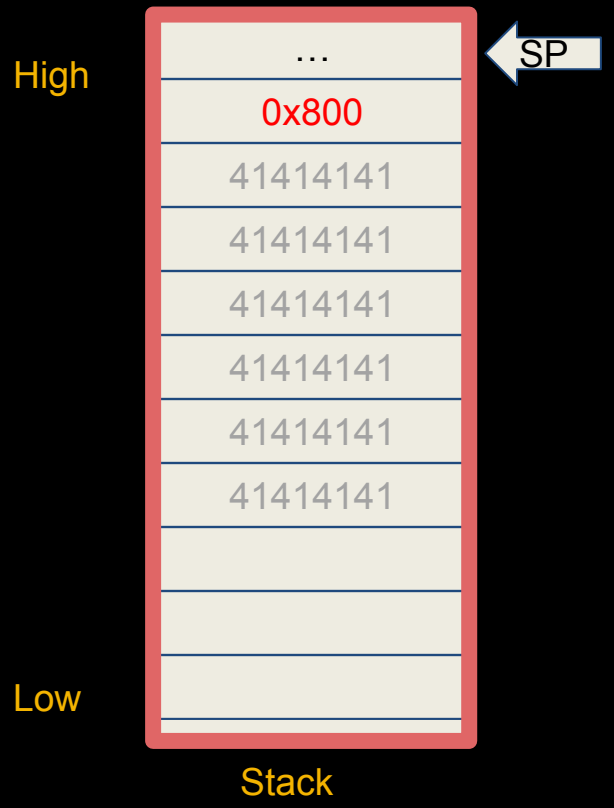
0x800

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

PC

LR

Return of the Operation



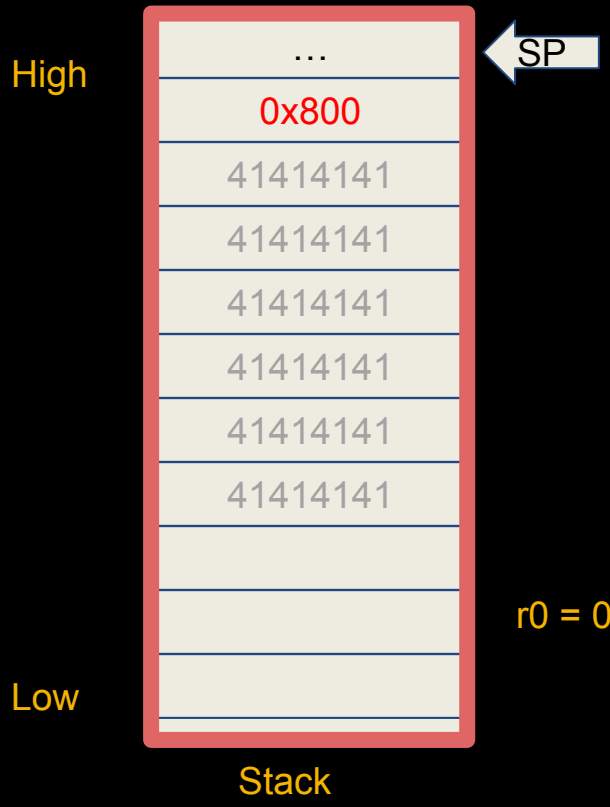
0x800

```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

LR

PC

Return of the Operation



0x800

r0 = 0xdeadbeef

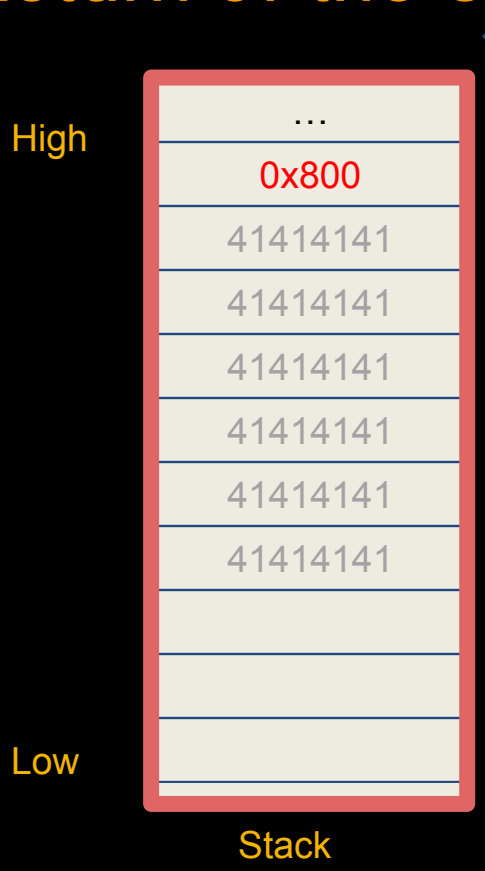
```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

LR

PC

Return of the Operation

emBO++




```
void callee(char* buf, size_t len) {
    char my_buf[16] = {0};
    memcpy(my_buf, buf, len);
}

void caller(char* aaa_buf) {
    callee(aaa_buf, 28);
}

void return_deadbeef() {
    return 0xdeadbeef;
}
```

LR

0x800

r0 = 0xdeadbeef

PC

Breaking Breakin gs

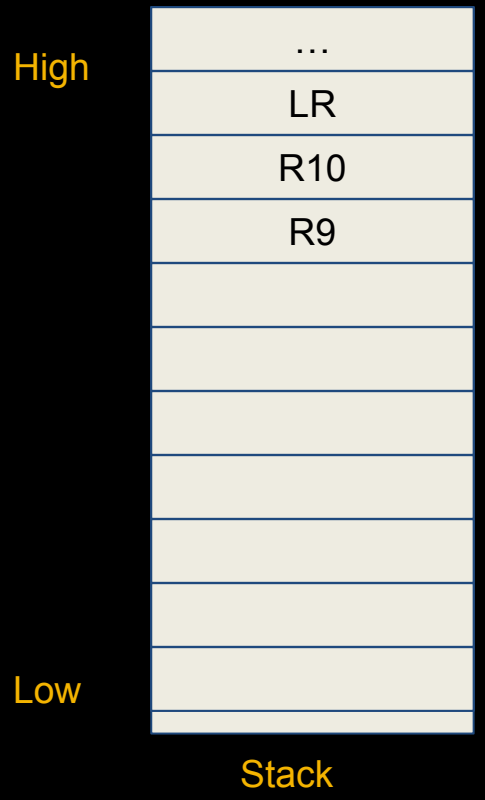
1. Dump it ✔
2. Analyze it ✔
3. Find it ✔
4. Exploit it ✔

Breaking Breakin[illegible]king Breaking
Exploiting things

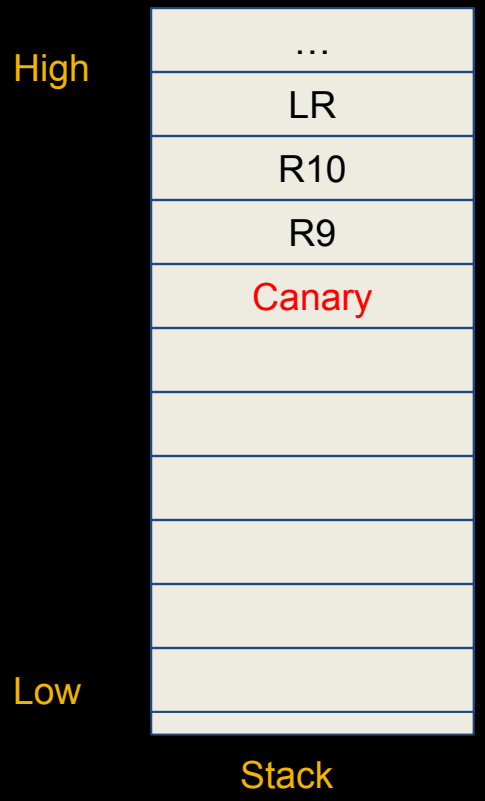
1. Dump it ✔
2. Analyze it ✔
3. Find it ✔
4. Exploit it ✔✔✔

Mitigations II

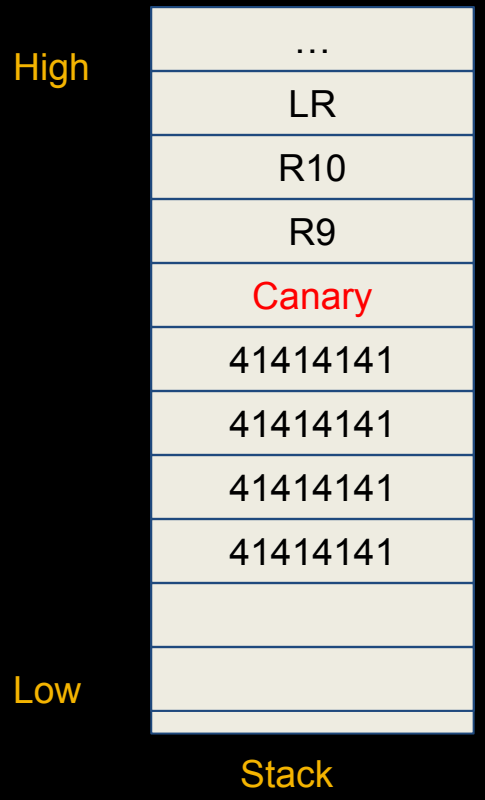
Stack canaries



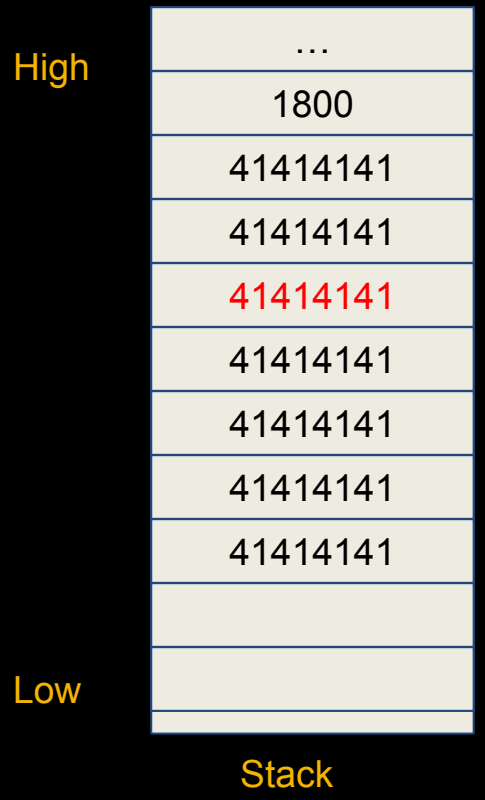
Stack canaries



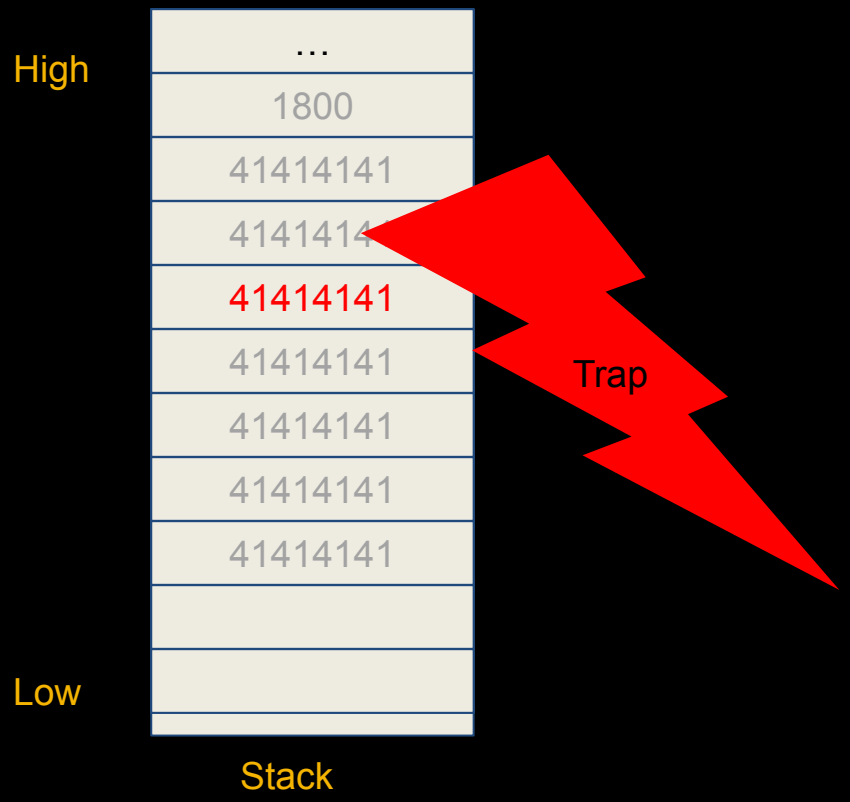
Stack canaries



Stack canaries

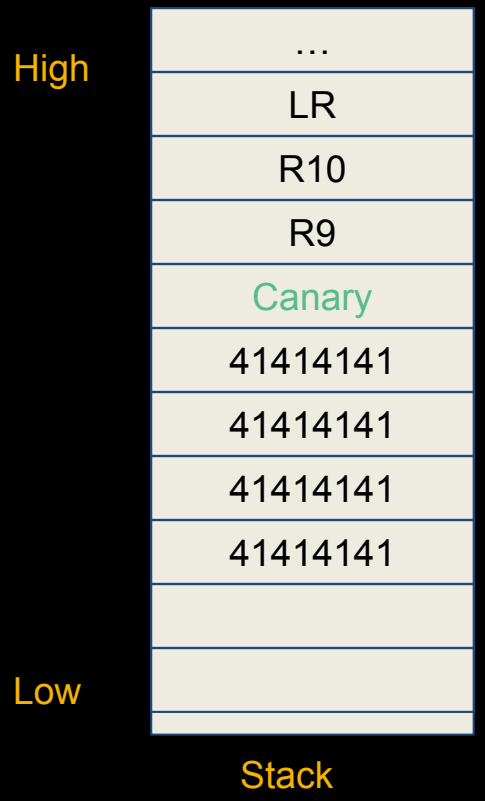


Stack canaries

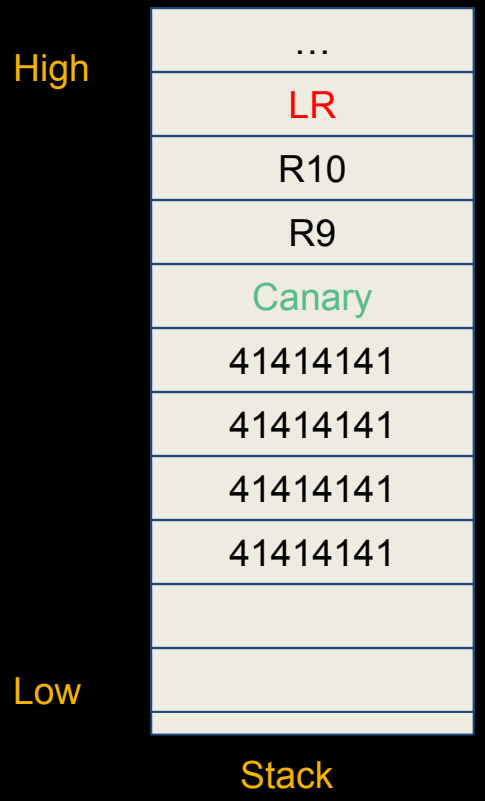


Control Flow Integrity - CFI

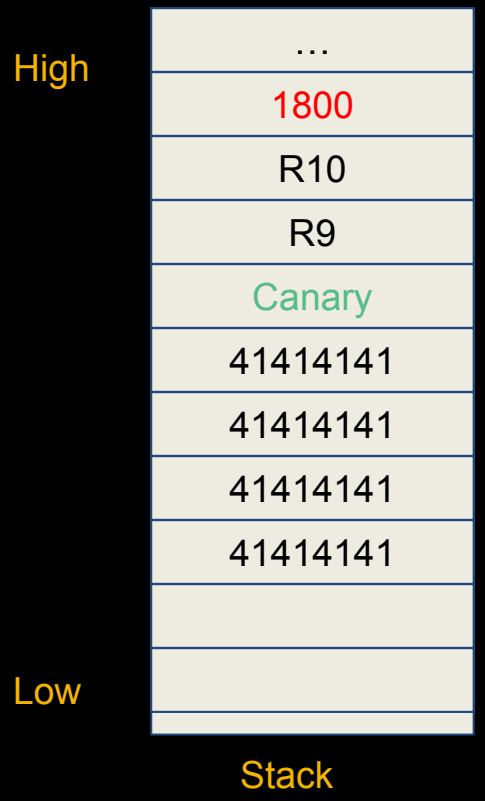
Control Flow Integrity - CFI



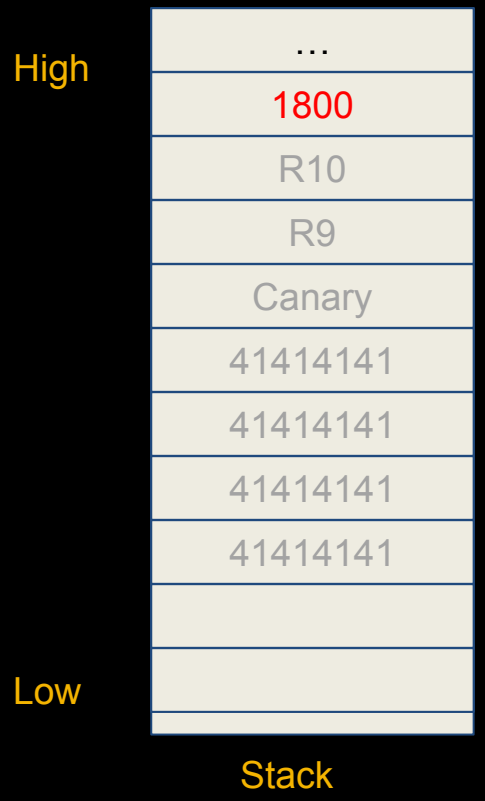
Control Flow Integrity - CFI



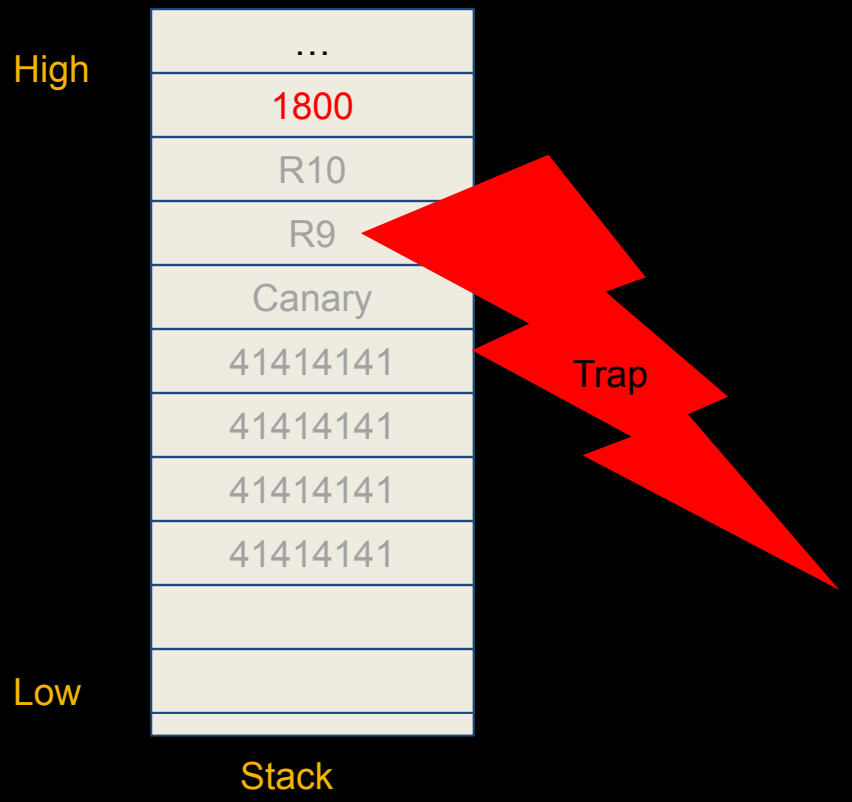
Control Flow Integrity - CFI



Control Flow Integrity - CFI



Control Flow Integrity - CFI



Mitigations

emBO++

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

Mitigations

emBO++

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

Mitigations

emBO++

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

Mitigations

emBO++

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

Mitigations

- NX
- Stack canaries
- CFI
- ASLR
- Binary diversity
- Anti-RE/Anti-Dump

Conclusion

- Many attacks, but also many defenses
- Try not to create bugs ;)
- … but always consider what happens if you do

Conclusion

emBO++

- Many attacks, but also many defenses
- Try not to create bugs ;)
- … but always consider what happens if you do
- Have fun :)

bkollenda@emproof.com
emproof.com/blogs/