そのprintf大丈夫ですか

～printfでデータを書き込む～

2024-11-30 Tech Circle Expo#3

TUATMCC SUGAWA

発表者

- 名前
  - SUGAWA
  - @sugawa197203
- 所属
  - 東京農工大学
  - TUATMCC
    （東京農工大学マイクロコンピュータークラブ）
  - B4
- 趣味
  - Unity、ネットワークいじり、マイクラ
  - 競プロ、CTF

MCCの紹介

MCC

競技プログラミング
Web開発
CTF
ゲーム開発
Kaggle
ネットワーク
ハッカソン

C言語使ったことありますか？

Multiple Choice Poll　25 votes　25 participants

ガッツリ使ってる！ - 7 votes

28%

触ったことある！ - 17 votes

68%

使ったことない、、、 - 1 vote

4%

そのprintf大丈夫ですか

～printfでデータを書き込む～

2024-11-30 Tech Circle Expo#3
TUATMCC SUGAWA

はじめに

- C言語とx86-64のお話です。
  - 低レイヤサイコー！

- ポインタとか、レジスタとかアセンブリ言語が出てきます。
  - 嫌いな人は許してください。

- “C言語は悪い”と言っているわけではありません。
  - むしろ良い高級言語です。超速いし

- ※ASLRとかカナリア変数は触れません

printfとは

可変長引数であり、第一引数で標準出力に出力する文字列のフォーマットを書く。

変数の場所は”フォーマット指定子”を使って指定する。
　　int は “%d”, float は “%f”, char[]（string）は “%s”

第二引数以降に表示したい変数を書く。

```c
#include <stdio.h>

int main(void)
{
  int a = 123;
  int b = 456;
  printf("a = %d, b = %d\n", a, b);
  return 0;
}
```

```
$ gcc 1.c
$ ./a.out
a = 123, b = 456
```

フォーマットの個数より引数が多いとき

エラーは出ずに余計な分は無視されます。

printf の内部では、フォーマットの文字列を先頭から見ていき、フォーマット指定子の順番に第二引数以降の変数を順番に見ていきます。

```c
#include <stdio.h>

int main(void)
{
  int a = 123;
  int b = 456;
  printf("a = %d\n", a, b);
  return 0;
}
```

```
$ gcc 1.c
$ ./a.out
a = 123
```

フォーマットの個数が引数より多いとき

エラーは出ずに第三引数が存在するとして処理されます。

printf の内部で、2つ目のフォーマット指定子が現れたら、第三引数があるべきだった場所のメモリを参照します。

```c
#include <stdio.h>

int main(void)
{
  int a = 123;
  int b = 456;
  printf("a = %d, b = %d\n", a);
  return 0;
}
```

```
$ gcc 1.c
$ ./a.out
a = 123, b = -113698104
```

関数を呼び出すときに起きる事

main 関数の2つの変数を
足して返すだけの add 関数

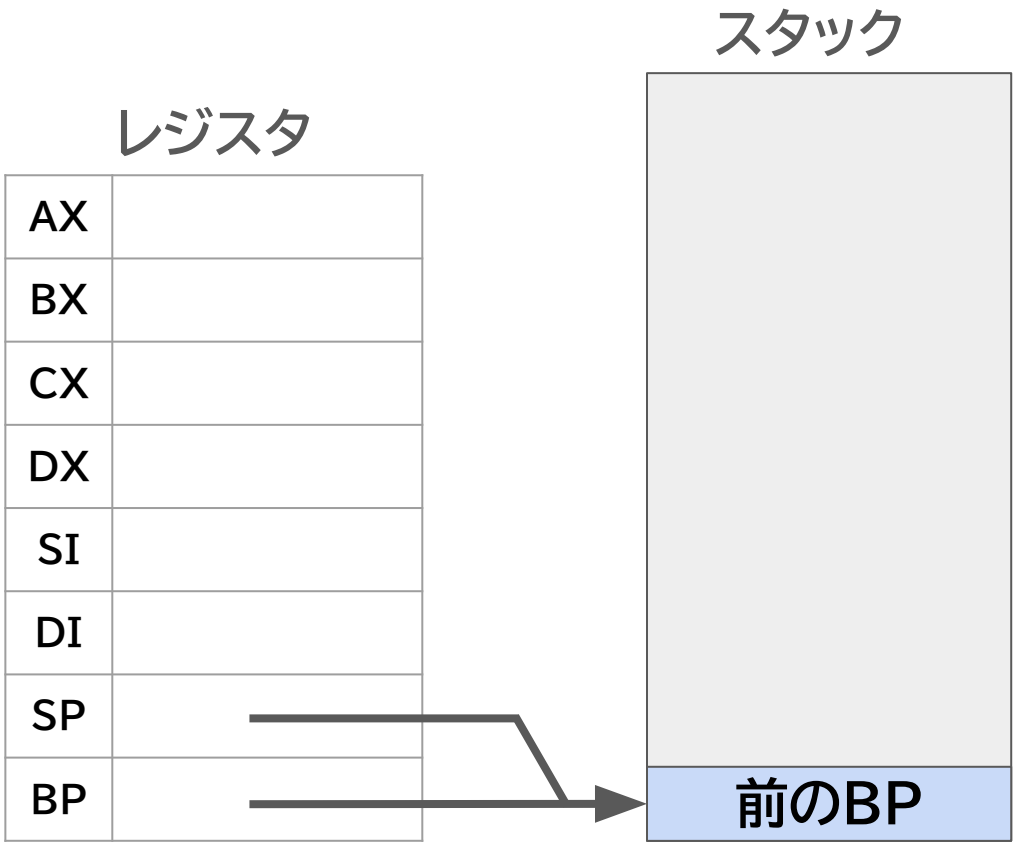
```c
#include <stdio.h>

int add(int a, int b)
{
  return a + b;
}

int main(void)
{
  int a = 1;
  int b = 2;
  int c = add(a, b);
  return 0;
}
```

関数を呼び出すときに起きる事

gcc で x86-64 にコンパイルして
objdump で add 関数と main 関数を確認

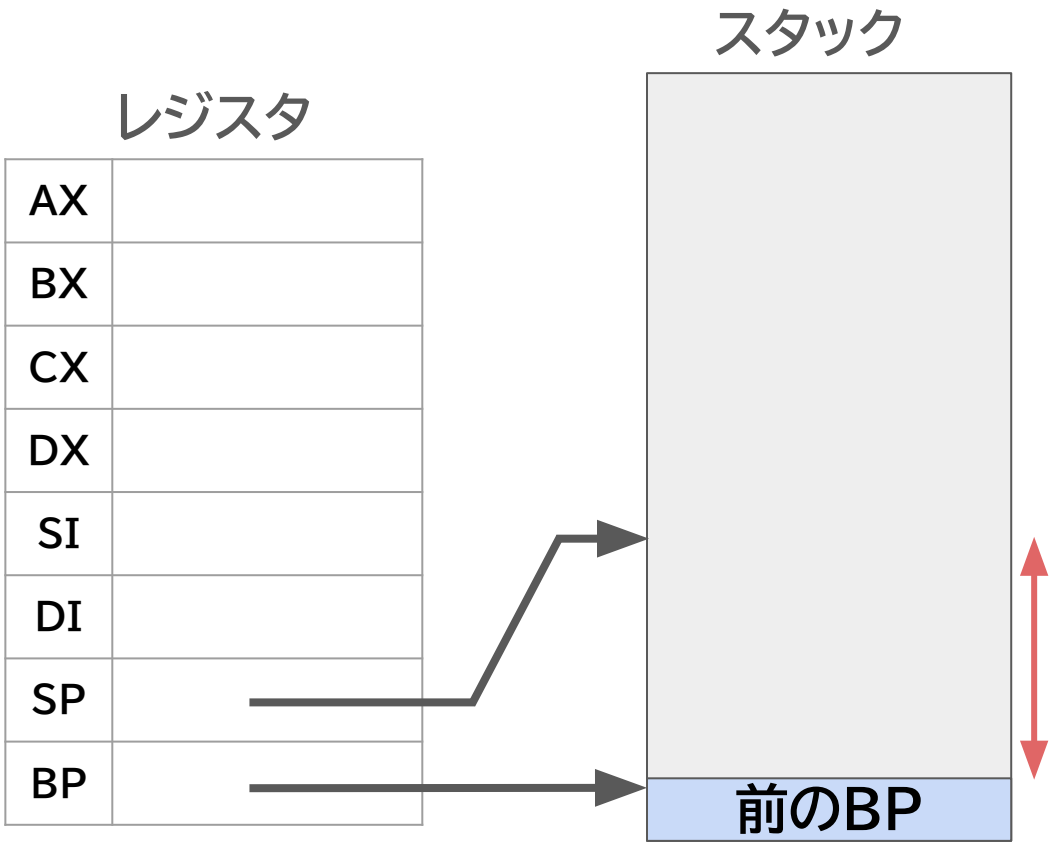
```
0000000000001129 <add>:
    1129:       55                      push   %rbp
    112a:       48 89 e5                mov    %rsp,%rbp
    112d:       89 7d fc                mov    %edi,-0x4(%rbp)
    1130:       89 75 f8                mov    %esi,-0x8(%rbp)
    1133:       8b 55 fc                mov    -0x4(%rbp),%edx
    1136:       8b 45 f8                mov    -0x8(%rbp),%eax
    1139:       01 d0                   add    %edx,%eax
    113b:       5d                      pop    %rbp
    113c:       c3                      ret

000000000000113d <main>:
    113d:       55                      push   %rbp
    113e:       48 89 e5                mov    %rsp,%rbp
    1141:       48 83 ec 10             sub    $0x10,%rsp
    1145:       c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
    114c:       c7 45 f8 02 00 00 00    movl   $0x2,-0x8(%rbp)
    1153:       8b 55 f8                mov    -0x8(%rbp),%edx
    1156:       8b 45 fc                mov    -0x4(%rbp),%eax
    1159:       89 d6                   mov    %edx,%esi
    115b:       89 c7                   mov    %eax,%edi
    115d:       e8 c7 ff ff ff          call   1129 <add>
    1162:       89 45 f4                mov    %eax,-0xc(%rbp)
    1165:       b8 00 00 00 00          mov    $0x0,%eax
    116a:       c9                      leave
    116b:       c3                      ret
```

関数を呼び出すときに起きる事



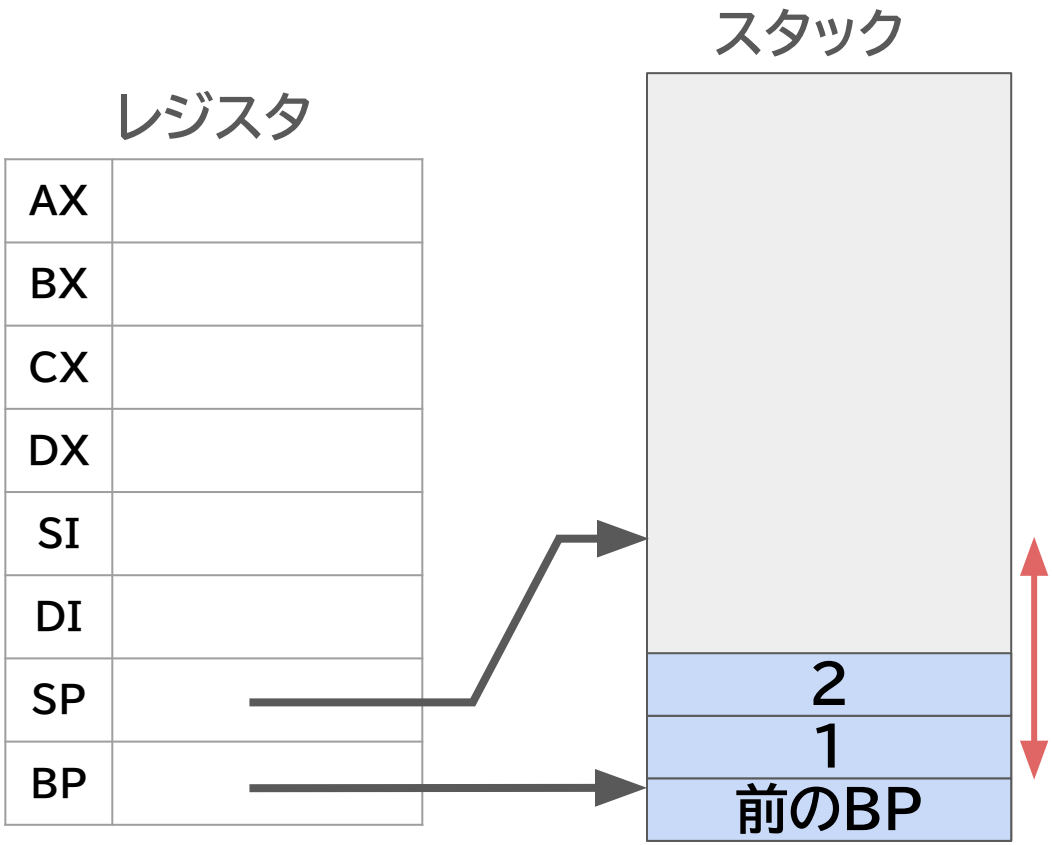

```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

関数を呼び出すときに起きる事



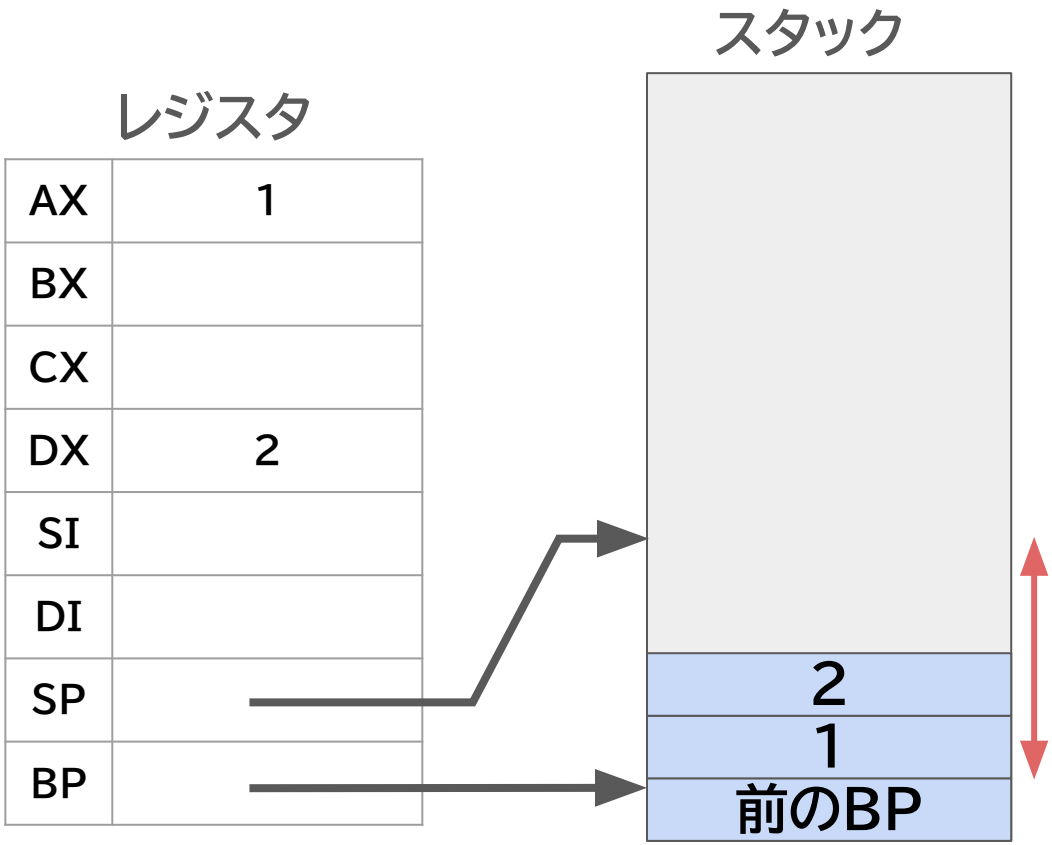

```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

関数を呼び出すときに起きる事



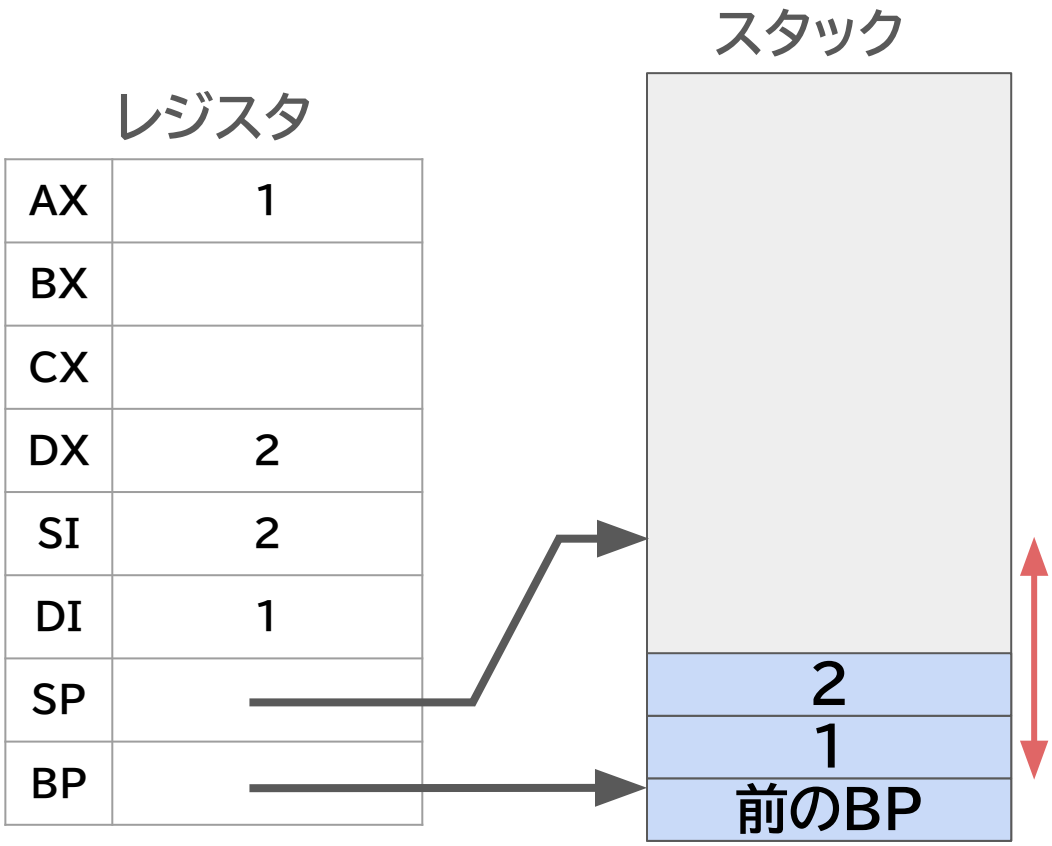

```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

関数を呼び出すときに起きる事




```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

関数を呼び出すときに起きる事



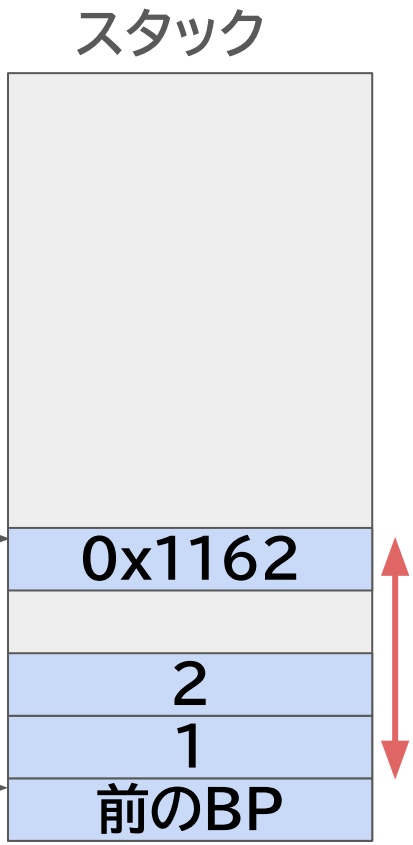

```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

DI: 第一引数
SI: 第二引数

関数を呼び出すときに起きる事

レジスタ

AX	1
BX	
CX	
DX	2
SI	2
DI	1
SP	
BP	



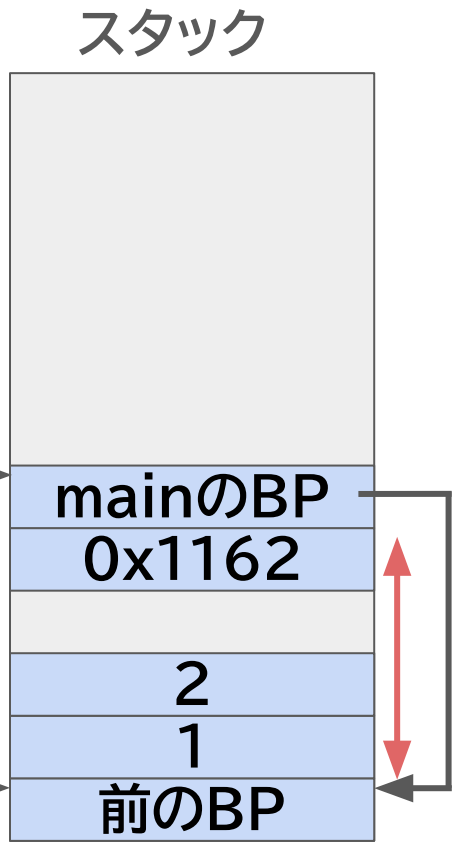

```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

DI: 第一引数
SI: 第二引数

関数を呼び出すときに起きる事

レジスタ

AX	1
BX	
CX	
DX	2
SI	2
DI	1
SP	
BP	

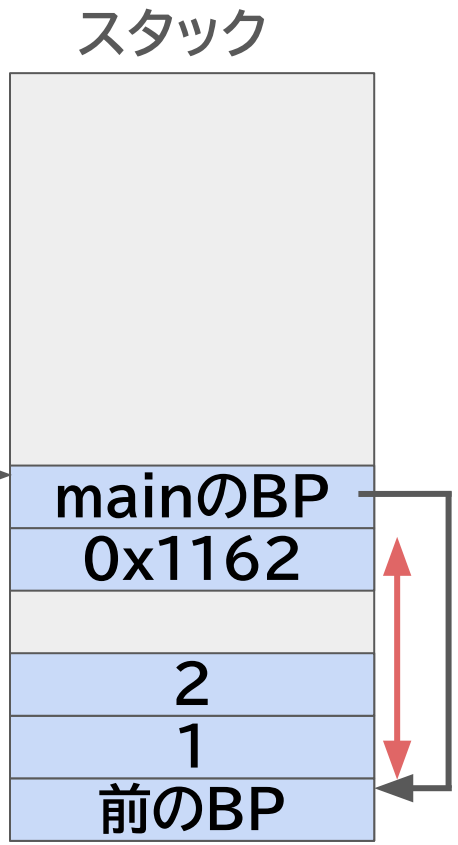



```
0000000000001129 <add>:
    1129: push   %rbp
    112a: mov    %rsp,%rbp
    112d: mov    %edi,-0x4(%rbp)
    1130: mov    %esi,-0x8(%rbp)
    1133: mov    -0x4(%rbp),%edx
    1136: mov    -0x8(%rbp),%eax
    1139: add    %edx,%eax
    113b: pop    %rbp
    113c: ret
```

関数を呼び出すときに起きる事

レジスタ

AX	1
BX	
CX	
DX	2
SI	2
DI	1
SP	
BP	

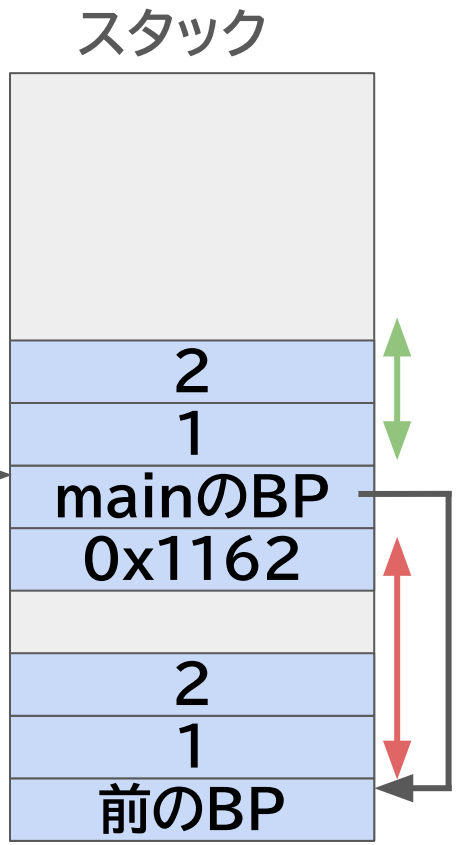



```
0000000000001129 <add>:
    1129: push   %rbp
    112a: mov    %rsp,%rbp
    112d: mov    %edi,-0x4(%rbp)
    1130: mov    %esi,-0x8(%rbp)
    1133: mov    -0x4(%rbp),%edx
    1136: mov    -0x8(%rbp),%eax
    1139: add    %edx,%eax
    113b: pop    %rbp
    113c: ret
```

関数を呼び出すときに起きる事

レジスタ

AX	1
BX	
CX	
DX	2
SI	2
DI	1
SP	
BP	

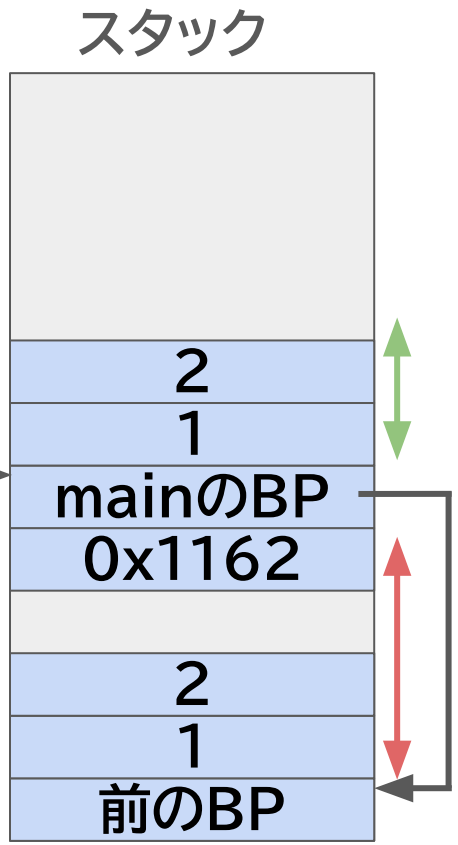



```
0000000000001129 <add>:
    1129: push   %rbp
    112a: mov    %rsp,%rbp
    112d: mov    %edi,-0x4(%rbp)
    1130: mov    %esi,-0x8(%rbp)
    1133: mov    -0x4(%rbp),%edx
    1136: mov    -0x8(%rbp),%eax
    1139: add    %edx,%eax
    113b: pop    %rbp
    113c: ret
```

関数を呼び出すときに起きる事

レジスタ

AX	2
BX	
CX	
DX	1
SI	2
DI	1
SP	
BP	

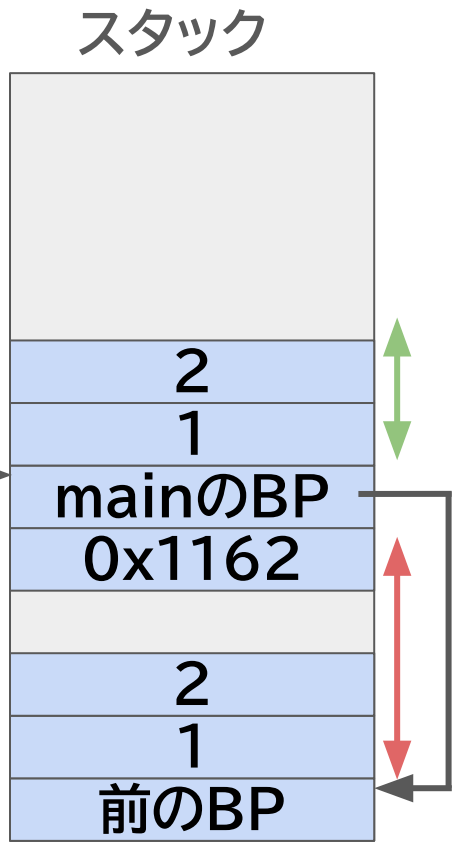



```
0000000000001129 <add>:
    1129: push   %rbp
    112a: mov    %rsp,%rbp
    112d: mov    %edi,-0x4(%rbp)
    1130: mov    %esi,-0x8(%rbp)
    1133: mov    -0x4(%rbp),%edx
    1136: mov    -0x8(%rbp),%eax
    1139: add    %edx,%eax
    113b: pop    %rbp
    113c: ret
```

関数を呼び出すときに起きる事

レジスタ

AX	3
BX	
CX	
DX	1
SI	2
DI	1
SP	
BP	

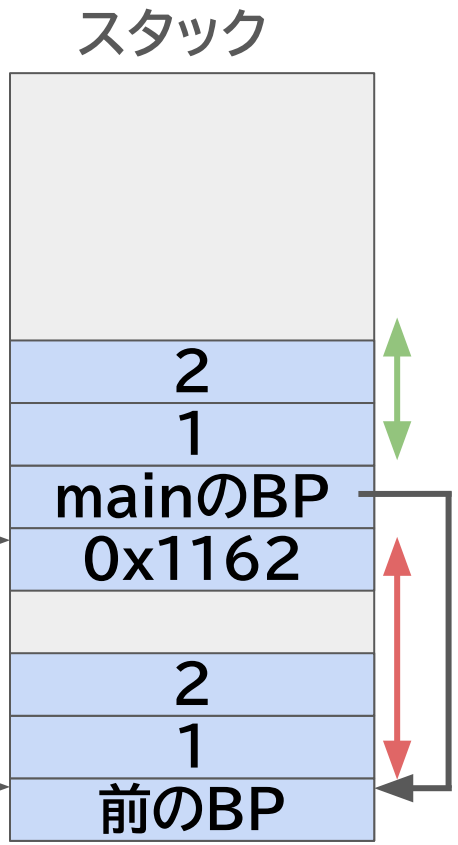



```
0000000000001129 <add>:
    1129: push   %rbp
    112a: mov    %rsp,%rbp
    112d: mov    %edi,-0x4(%rbp)
    1130: mov    %esi,-0x8(%rbp)
    1133: mov    -0x4(%rbp),%edx
    1136: mov    -0x8(%rbp),%eax
    1139: add    %edx,%eax
    113b: pop    %rbp
    113c: ret
```

関数を呼び出すときに起きる事

レジスタ

AX	3
BX	
CX	
DX	1
SI	2
DI	1
SP	
BP	

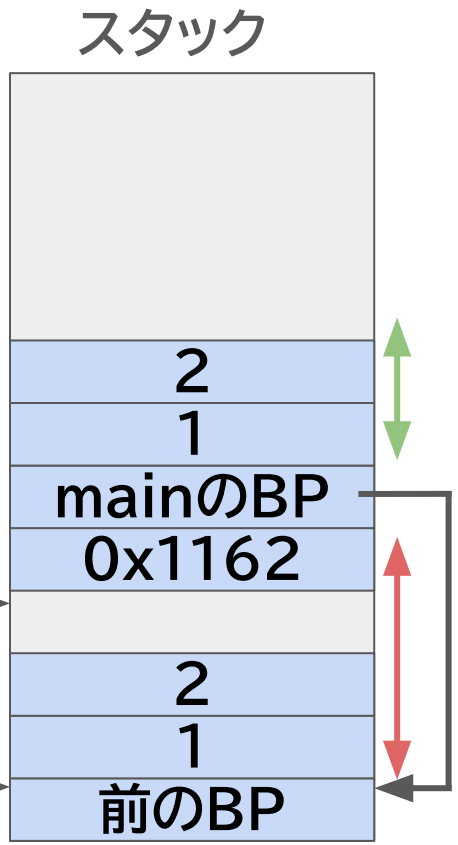



```
0000000000001129 <add>:
    1129: push   %rbp
    112a: mov    %rsp,%rbp
    112d: mov    %edi,-0x4(%rbp)
    1130: mov    %esi,-0x8(%rbp)
    1133: mov    -0x4(%rbp),%edx
    1136: mov    -0x8(%rbp),%eax
    1139: add    %edx,%eax
    113b: pop    %rbp
    113c: ret
```

関数を呼び出すときに起きる事

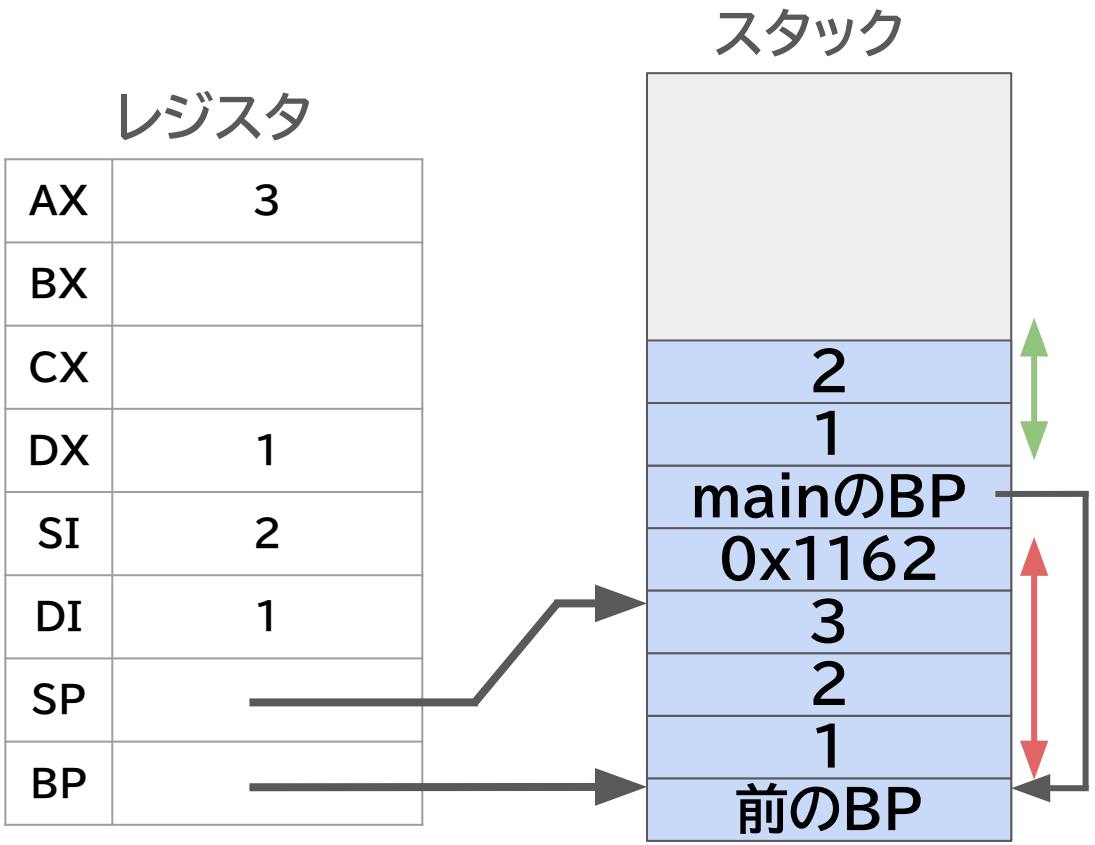
レジスタ

AX	3
BX	
CX	
DX	1
SI	2
DI	1
SP	
BP	




```
0000000000001129 <add>:
    1129: push   %rbp
    112a: mov    %rsp,%rbp
    112d: mov    %edi,-0x4(%rbp)
    1130: mov    %esi,-0x8(%rbp)
    1133: mov    -0x4(%rbp),%edx
    1136: mov    -0x8(%rbp),%eax
    1139: add    %edx,%eax
    113b: pop    %rbp
    113c: ret
```

関数を呼び出すときに起きる事




```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

関数を呼び出すときに起きる事

引数はレジスタに入れられて関数が呼ばれる。

第一引数 DI
第二引数 SI
第三引数 DX
第四引数 CX
第五引数 R8
第六引数 R9
第七引数以降 スタック

printf を呼び出すときに起きる事

DI にはフォーマットの文字列のポインタが入る。第二引数以降は SI などに入る。
フォーマット指定子を増やせば引数が本来ある場所のスタック見る。

```c
#include <stdio.h>

int main(void)
{
  printf("%p,%p,%p,%p,%p,%p,%p,%p,%p,%p\n");
  return 0;
}
```

プログラム

```
(gdb) x/16xg $rbp
0x7fffffffd580: 0x0000000000000001      0x00007ffff7de2d68
0x7fffffffd590: 0x00007fffffffd680      0x0000555555555139
0x7fffffffd5a0: 0x0000000155554040      0x00007fffffffd698
0x7fffffffd5b0: 0x00007fffffffd698      0x7efa7396507c0ef4
0x7fffffffd5c0: 0x0000000000000000      0x00007fffffffd6a8
0x7fffffffd5d0: 0x00007ffff7ffd000      0x0000555555557dd8
0x7fffffffd5e0: 0x81058c69fb5e0ef4      0x81059c2a0a3e0ef4
0x7fffffffd5f0: 0x0000000000000000      0x0000000000000000
```

printf実行前のスタック

```
(gdb) info regi
rax            0x0
rbx            0x7fffffffd698
rcx            0x555555557dd8
rdx            0x7fffffffd6a8
rsi            0x7fffffffd698
rdi            0x555555556008
rbp            0x7fffffffd580
rsp            0x7fffffffd578
r8             0x0
r9             0x7ffff7fcbf40
r10            0x3
r11            0x7ffff7e121c0
r12            0x0
r13            0x7fffffffd6a8
r14            0x7ffff7ffd000
r15            0x555555557dd8
rip            0x7ffff7e121c0 <__printf>
eflags         0x206 [ PF IF ]
cs             0x33
ss             0x2b
ds             0x0
es             0x0
fs             0x0
gs             0x0
fs_base        0x7ffff7db6740
gs_base        0x0
```

printf実行前のレジスタ

```
0x7fffffffd698,0x7fffffffd6a8,0x555555557dd8,(nil),0x7ffff7fcbf40,0x1,0x7ffff7de2d68,0x7fffffffd680,0x555555555139,0x155554040
```

実行結果

printf のフォーマット指定子

- printf の %c でスペースの後に文字を出力する機能

```
printf("%4c\n", 'A');
```

```
   A
```

printf のフォーマット指定子

- printf の引数をスキップする機能

```c
int a = 1, b = 2, c = 3;
printf("%3$d\n", a, b, c);
```

```
3
```

printf のフォーマット指定子

- printf の %n で出力した文字をカウントする機能

```c
int n;
printf("abcdef%n\n", &n);
printf("%d\n", n);
```

```
abcdef
6
```

printfでデータを書き込む

- printf の %c でスペースの後に文字を出力する機能
  - 特定の数字を作れる
- printf の引数をスキップする機能
  - スタックを参照できる
- printf の %n で出力した文字をカウントする機能
  - スタックで指定したアドレスに特定の数字を書き込める

printfでデータを書き込む

```c
#include <stdio.h>

int main(void)
{
  int a = 1;
  char str[64];

  printf("%p\n", &a);
  scanf("%s", str);
  printf(str);
  printf("a=%d\n", a);

  return 0;
}
```

```
0x7ffc96e4bdcc
```

最初の printf

```
b'%10c%8$n,1234567\xcc\xbd\xe4\x96\xfc\x7f\x00\x00'
```

入力

```
a=10
```

最後の printf

printfでデータを書き込む

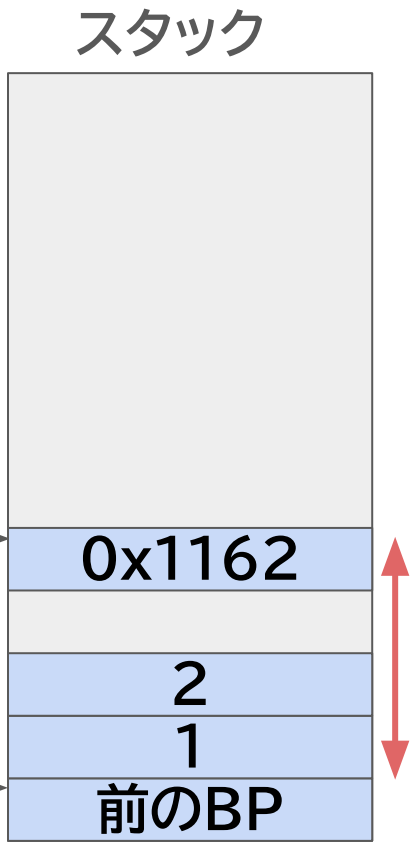
- printf でスタック上にある call したときにスタックに入れた printf の戻りアドレスも書き換えられる
  - 任意コードが実行可能

- Format String Bug

関数を呼び出すときに起きる事

レジスタ

AX	1
BX	
CX	
DX	2
SI	2
DI	1
SP	
BP	




```
000000000000113d <main>:
    113d: push   %rbp
    113e: mov    %rsp,%rbp
    1141: sub    $0x10,%rsp
    1145: movl   $0x1,-0x4(%rbp)
    114c: movl   $0x2,-0x8(%rbp)
    1153: mov    -0x8(%rbp),%edx
    1156: mov    -0x4(%rbp),%eax
    1159: mov    %edx,%esi
    115b: mov    %eax,%edi
    115d: call   1129 <add>
    1162: mov    %eax,-0xc(%rbp)
    1165: mov    $0x0,%eax
    116a: leave
    116b: ret
```

DI: 第一引数
SI: 第二引数

printfでデータを書き込む

- printf でスタック上にある call したときにスタックに入れた printf の戻りアドレスも書き換えられる
  - 任意コードが実行可能
- system関数
  - 引数の文字列をコマンドとして実行

戻りアドレスに system 関数のアドレスを書き込み、実行したいコマンドを引数として書き込めば任意コマンドが実行可能

まとめ

- プログラムで関数呼び出し時のスタックの動きを見た
- printf のフォーマットを不正に使うと、
  スタックのメモリリーク、メモリの書き換えが可能
  - 任意コード実行可能

```c
scanf("%s", str);
printf(str);
```

危ないコード

```c
scanf("%s", str);
printf("%s", str);
```

まぁ大丈夫なコード

ご清聴ありがとうございました！