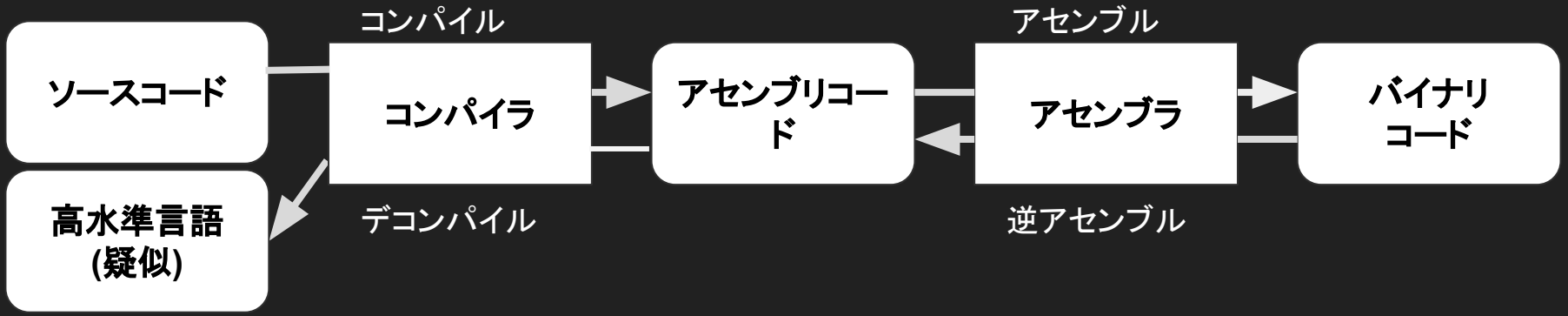
バイナリ前半

BGPを使いながら動きを見てみよう

コンパイルとアセンブルの流れ



ソースコードは.c等がついた高級言語で書かれたプログラム

アセンブリコードは低水準言語、 ADDやSUBなどニーモニックで表現

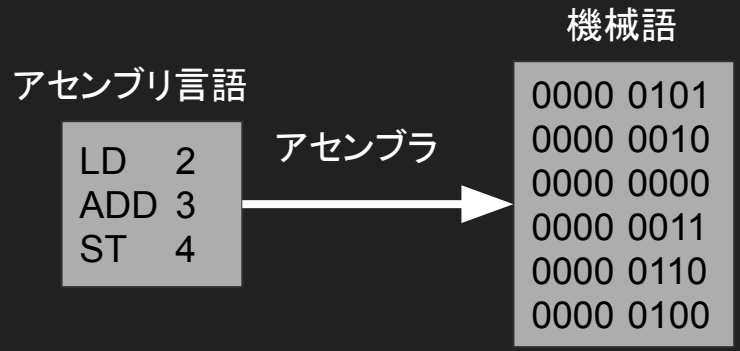
バイナリコードは機械語とも呼ばれる　0,1で表現される16進数で表示

デコンパイラは後半

アセンブラについて

例

命令番号	ニーモニック	意味
000	ADD	加算
001	SUB	減算
010	AND	AND
011	OR	OR



機械語は0と1の羅列で人間向きではない為、上のように命令番号を意味のある単語の頭文字に置き換えてプログラムを書く方法が考えられた

置き換えられた記号を「ニーモニック」という

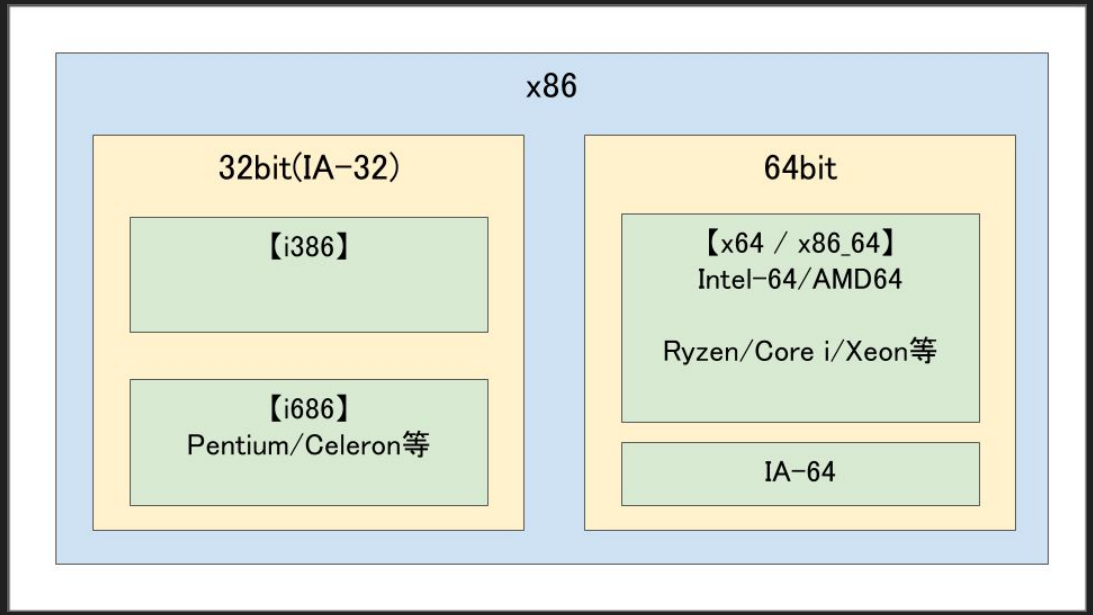
実行ファイル

実行ファイルはアーキテクチャごとに違う（命令セットが違う）

図はIntelや互換性を持つ命令セットの定義である。AMDは互換プロセッサである（命令セットが同じ）

ラズパイ等で使用されるCPUのArmは互換性無し

x86の定義



https://blog.framinal.life/entry/2020/04/22/041548

実行ファイル形式 （PEとELF）

fileコマンドでPEとELFそれぞれ確認

hadson1のなか

これは何か

ファイル形式の種類

WindowsはPE　拡張子：.exeなど

LinuxはELF　拡張子はない　実行可能パーミッションxで判断

MacやiOSはMach-O

https://resources.infosecinstitute.com/author/security-ninja/page/17/

コンパイル言語とスクリプト言語

コンパイル言語：C、Java、Goなど

スクリプト言語：Javascript、PHP、Pythonなど

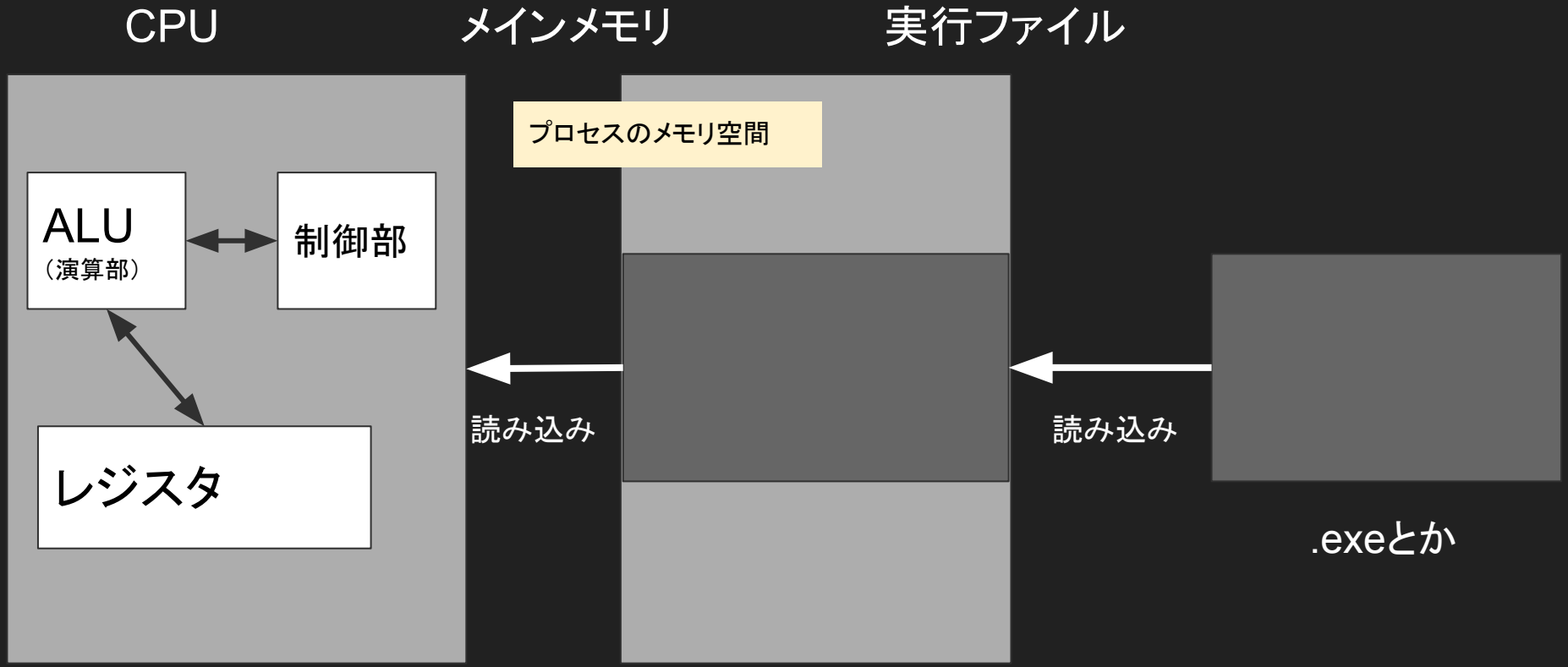
・スクリプト言語は命令を一つずつ機械語にしながら実行するインタプリタ方式

・コンパイルを利用しないか利用するか

・コンパイル言語のほうが早い

・コンパイルするにはアーキテクチャに縛られる

実行ファイルがハードウェアで実行される流れ



レジスタについて

CPU内部の記憶装置

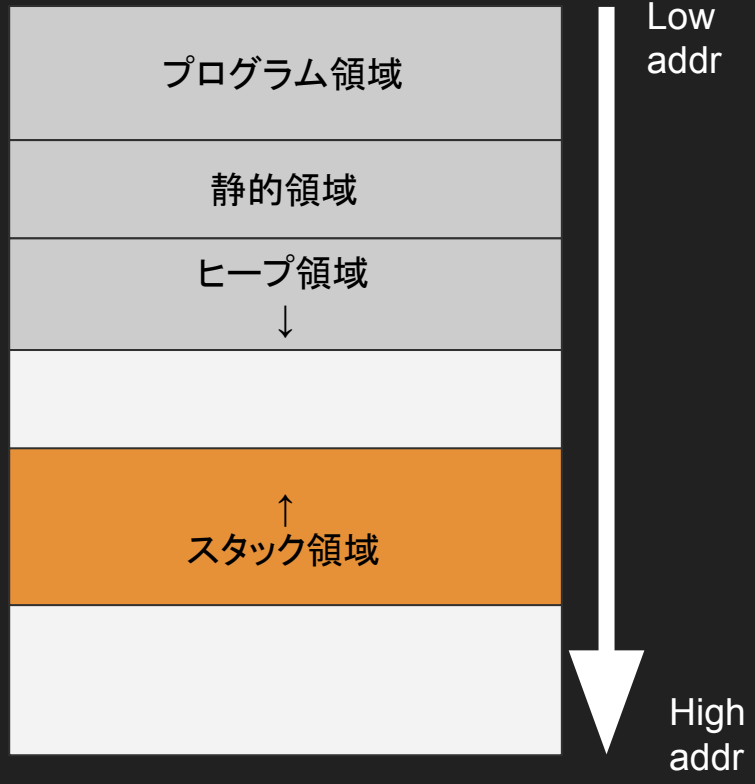
レジスタの記憶容量はメモリより少ないが、高速にアクセス出来る

レジスタの種類

汎用レジスタ、ステータスレジスタ、命令ポインタ、セグメントレジスタ

後に詳しく説明

x86のメモリ領域



プログラム領域:

- 実行コードを格納

静的領域:

- 初期値を持つ変数を格納
- 初期値を持たない変数を格納

ヒープ領域:

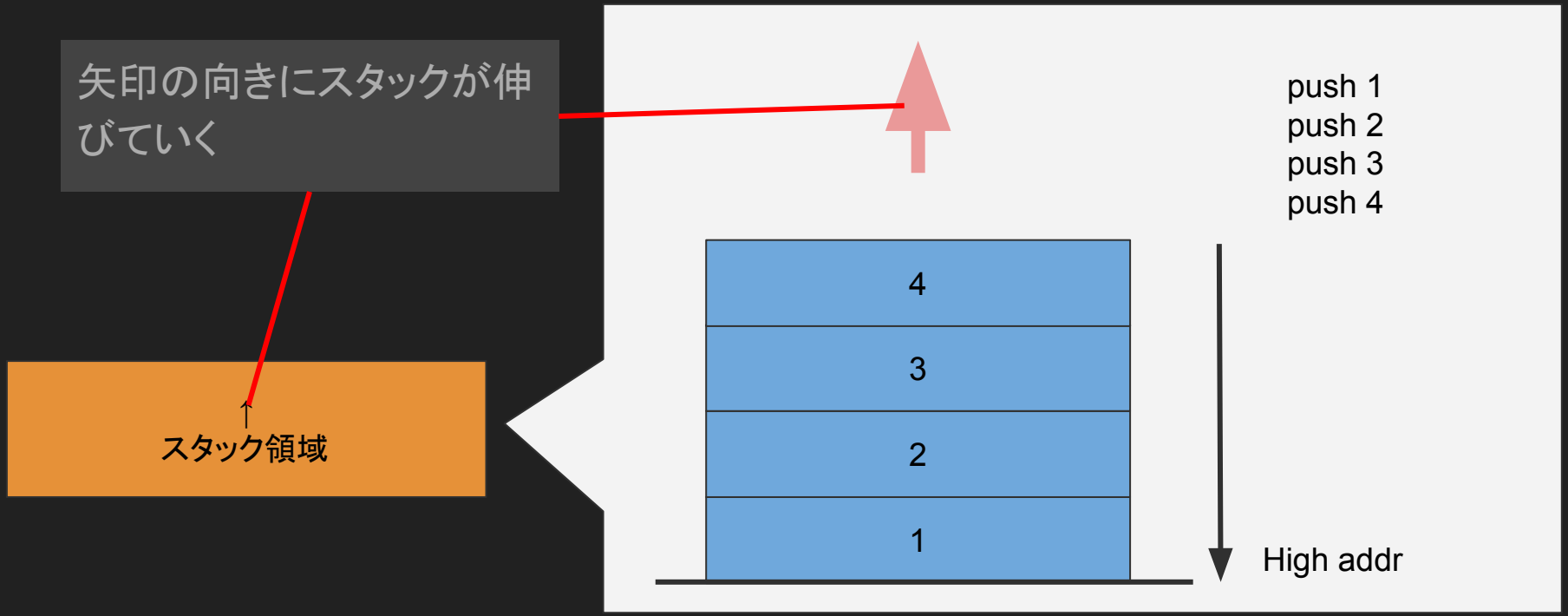
- malloc()等で動的に確保可能なメモリ領域

スタック領域:

- ローカル変数や関数の戻り先など格納

スタックについて

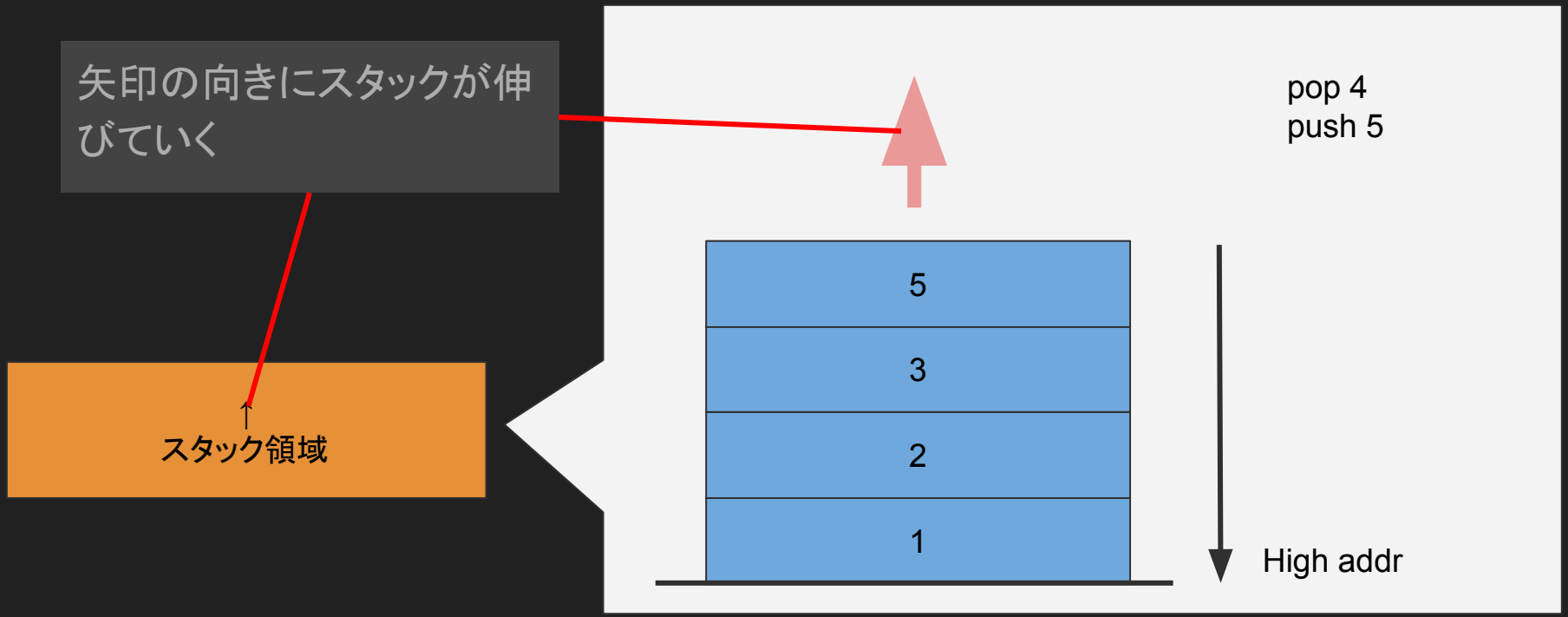
矢印の向きにスタックが伸びていく



スタックについて

メモリとスタックのアドレスの動きが逆なので混乱しないように注意！

矢印の向きにスタックが伸びていく



GDBでスタックの動きを確認

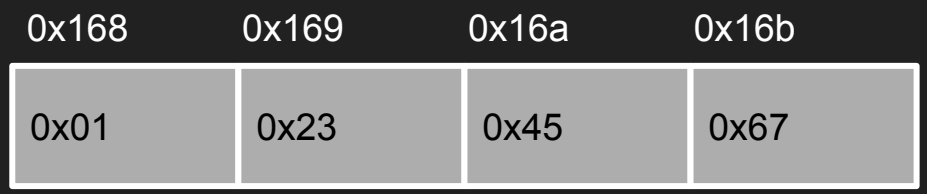
stack_4-1をgdbで見る

コマンド
gdb stack.out
b main
run
n

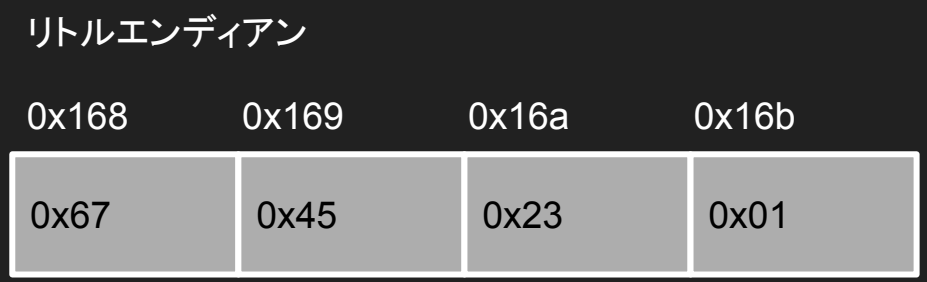
リトルエンディアンとビッグエンディアンについて

0x01234567がメモリに書き込まれるときそれぞれどうなるか

ビッグエンディアン



リトルエンディアン



バイトオーダーも意識しよう
32bitは4バイトこの4バイト単位でメモリとCPUがやり取りする

0x01234567は16進数で8桁つまり2ケタが1バイトなので32ビット

メモリのアドレスは適当

BGPでエンディアンを確認

stack.outをgdbで見る

コマンド
gdb endian
b main
run
n
x/s RAX等で表示されているアドレス
x/c ↑と同じアドレス
enterを押して文字を一つずつ表示

x86の汎用レジスタの意味と主な用途

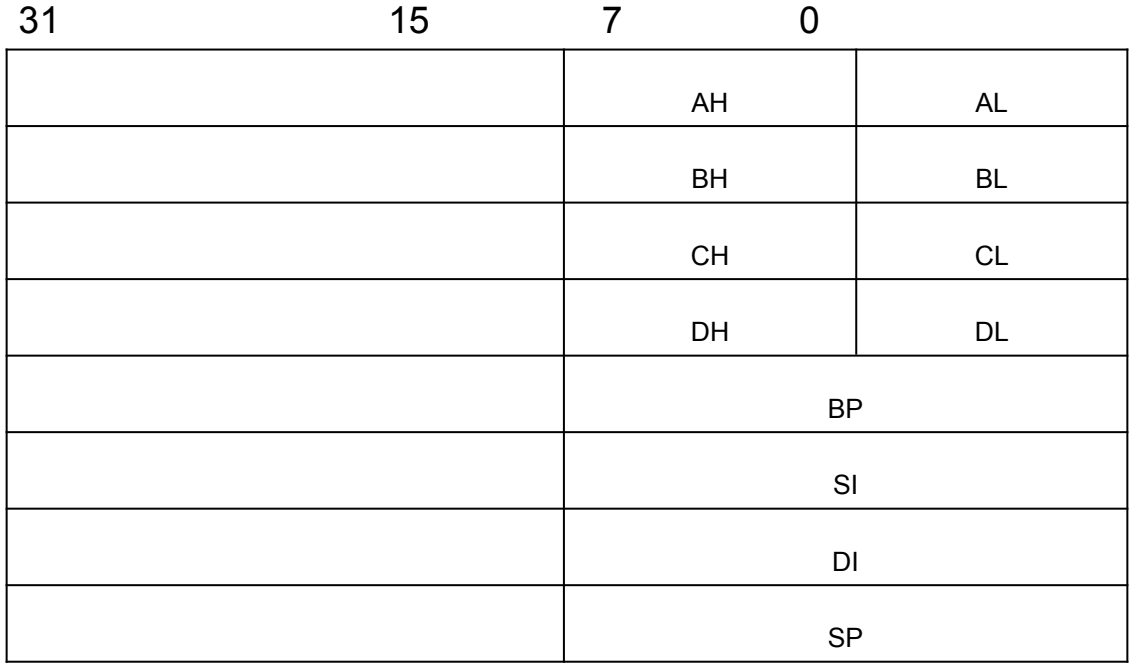
レジスタ名	名前の意味	主な用途
EAX	アキュームレジスタ	演算、関数の戻り値
EBX	ベースレジスタ	データに対するポインタ
ECX	カウントレジスタ	ループ処理時のカウンタ
EDX	データレジスタ	演算
ESI	ソースインデックス	一部のデータ転送命令において、データの転送元を格納する
EDI	ディスティネーションインデックス	一部のデータ転送命令において、データの転送先を格納する

特殊レジスタ

ESP	スタックポインタ	スタックのトップアドレスへのポインタ
EBP	ベースポインタ	スタックのベースアドレスへのポインタ
EIP	インストラクションポインタ （命令ポインタ）	次に実行するアセンブリ命令のアドレスを保持する

X86汎用レジスタ構造

AX,BX,CX,DX



EAX,EBX,ECX,EDX,EBP,ESI,EDI,ESP

ステータスレジスタ

X86ではEFLAGレジスタと呼ばれる。
これらのフラグを使用して、演算結果の状態を示す

名称	概要
ZF	演算結果が0のときセットされる
SF	演算結果が負のときセットされる
CF	符号なし演算結果がオーバーフローしたときセットされる
0F	符号付き演算でオーバーフローしたときセットされる

x64汎用レジスタ

レジスタ名	名前の意味	主な用途
RAX	アキュームレジスタ	演算、関数の戻り値
RBX	ベースレジスタ	データに対するポインタ
RCX	カウントレジスタ	ループ処理時のカウンタ
RDX	データレジスタ	演算
RSP	スタックポインタ	スタックのトップへのポインタ
RBP	ベースポインタ	BPとしても使われるが汎用レジスタとして扱えるようになった
RSI	ソースインデックス	一部のデータ転送命令において、データの転送元を格納する
RDI	ディスティネーションインデックス	一部のデータ転送命令において、データの転送先を格納する
r8~r15	―	引数などに使用

x86と変わっている所をさす

x64からの変更

すべてのアドレス・ポインタが64ビットに

レジスタが64ビットになって数が増加

- 浮動小数点演算用SSE2レジスタなどが追加

関数の呼び出し規約がx64呼び出し規約に変更

- 引数の渡され方など変更

静的解析と動的解析について

静的解析:逆アセンブルデコンパイルしたコードを読み解析

ソースコードを読む、linter、逆アセンブルで使用（objdump）

動的解析：実際にバイナリを実行し、実行中のメモリやレジスタの状態を確認しながら解析

デバッガ—（windbg、GDB）

GhidraやIDAはディスアセンブルだけでなくデコンパイル、デバッガがある**強い!**

関数とスタック

・関数ごとにスタックの固有の領域を持つ

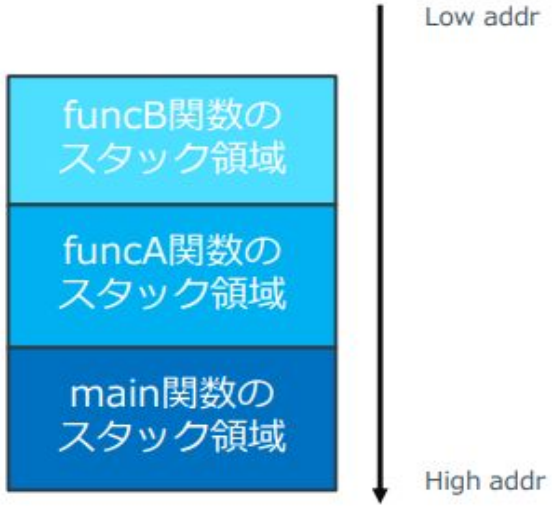
・main()→funcA()→funcB()のようにスタックが積みあがっていく

- 処理が終われはpopされスタック上から消される

```
void funcB(int c)
{
        ...
}

void funcA(int a, int b)
{
        ...
        funcB(a);
        ...
}

int main(void)
{
        ...
        funcA(a, b);
        ...
        return 0;
}
```

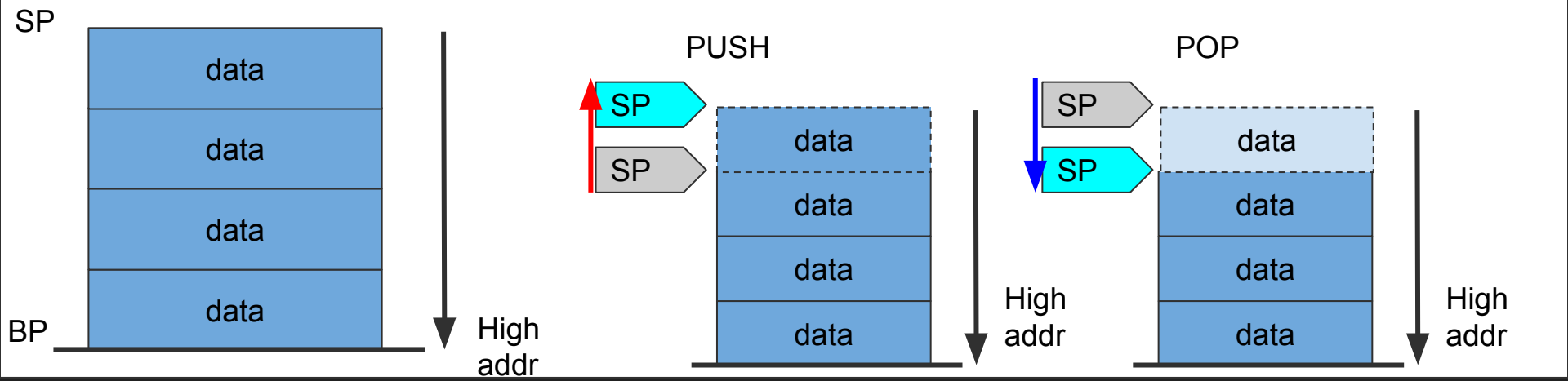


ベースポインタとスタックポインタ

スタックの一番上をスタックポインタ(SPと表記している)

スタックの一番下をベースポインタ(BPと表記している)

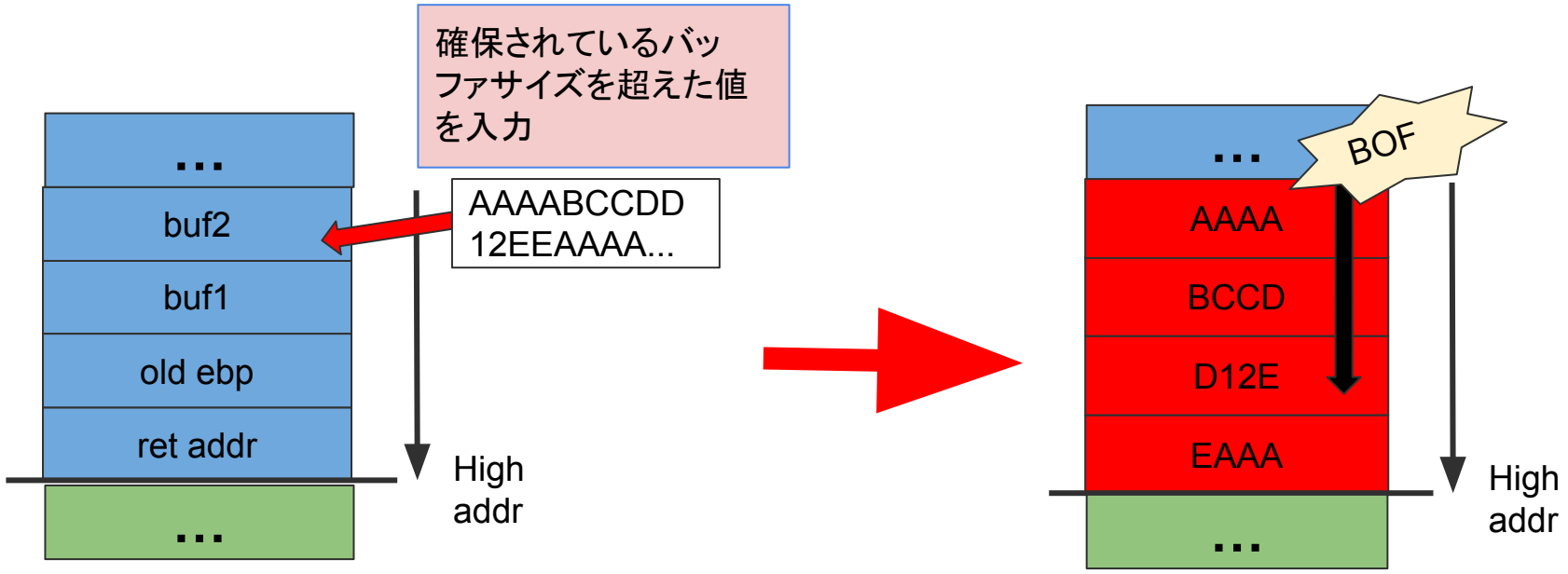
SPはPUSHされると－、POPされると＋と変化する。32bitは4バイト、64bitは8バイトずつ



バイナリとセキュリティ
stackに関して

バッファオーバーフローについて

スタック領域にて確保されたバッファを越えた書き込みが行われあふれること



＊これにより他の変数を書き換える事が可能になってしまう

簡単なバッファオーバーフローを見る

bof1を実行
bof2を実行

対策方法について

バッファ以上の入力を受け付けないようにする必要がある

fgetsとgets

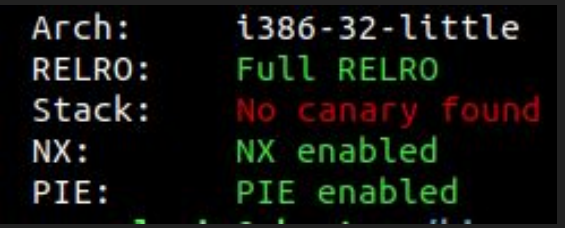
C言語ではgetsではなくfgets関数を利用することで読み込むサイズを指定できるため正しいサイズを指定することで安全

防御機能

今回の実行ファイルはわざとセキュリティを外してコンパイルしている

防御機能について

ターミナルでchecksecを実行してみる



RELRO: RELocation Read-Only
Stack: Stack Smash Protection カナリアを挿入
NX: NX Bit
PIE: Position Independent Executable

他
ASLR: Address Space Layout Randomization

カナリアについて

Stack Smash Protection(SSP)

コンパイラが付与するセキュリティ機構の一つ

カナリアと呼ばれる値を挿入

呼び出し先の関数の実行が終了した際にカナリアの値が書き換えられているかどうかを確認し、意図しないローカル変数の書き換えの発生を検出する

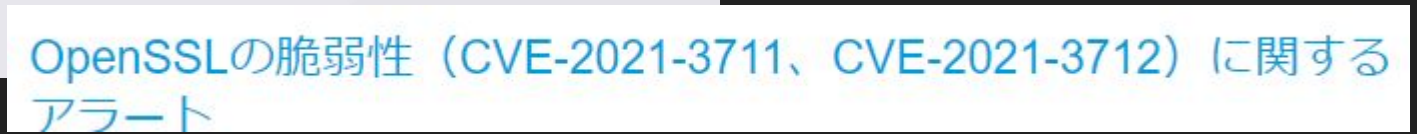
これを突破する手法も登場している

バッファオーバーフローって今時あるの？

https://www.mandiant.com/resources/critical-buffer-overflow-vulnerability-in-solaris-can-allow-remote-takeover

https://www.jpcert.or.jp/english/at/2021/at210036.html

結構見つかる



発展

Pwn

- Stack buffer overflow
- Format string bug
- Heap buffer overflow
- Use after free
- Race Contidion

興味がある人はやってみてください

https://pwn.college/
Pwnを学べるサイト