[ASM] Systèmes, Microprocesseurs, Architecture (2)

Par Rhaeven, relecture et adaptation par Find3r

Stack

Stack : Gestion automatique de la mémoire

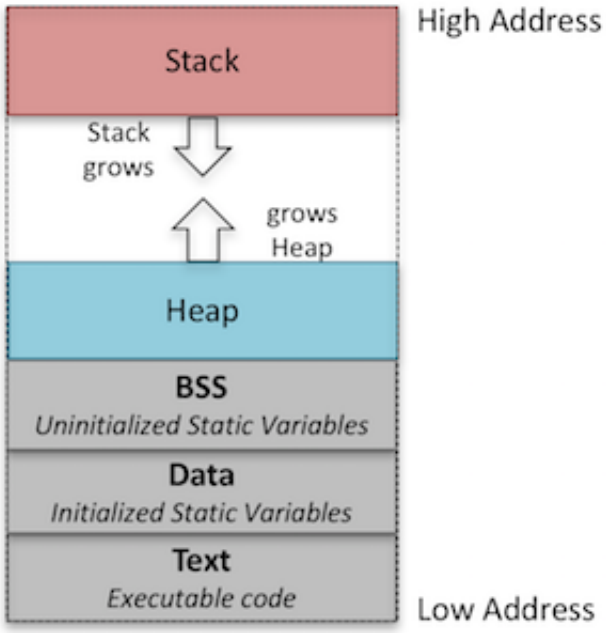
- Stocker les variables locales
- Sauver les adresses de retour des fonctions

Registres :

- **%rbp** :
  - Adresse haute de la pile
  - Base pointer de la stack frame courante
- **%rsp**
  - On accède à la pile grâce à %rsp

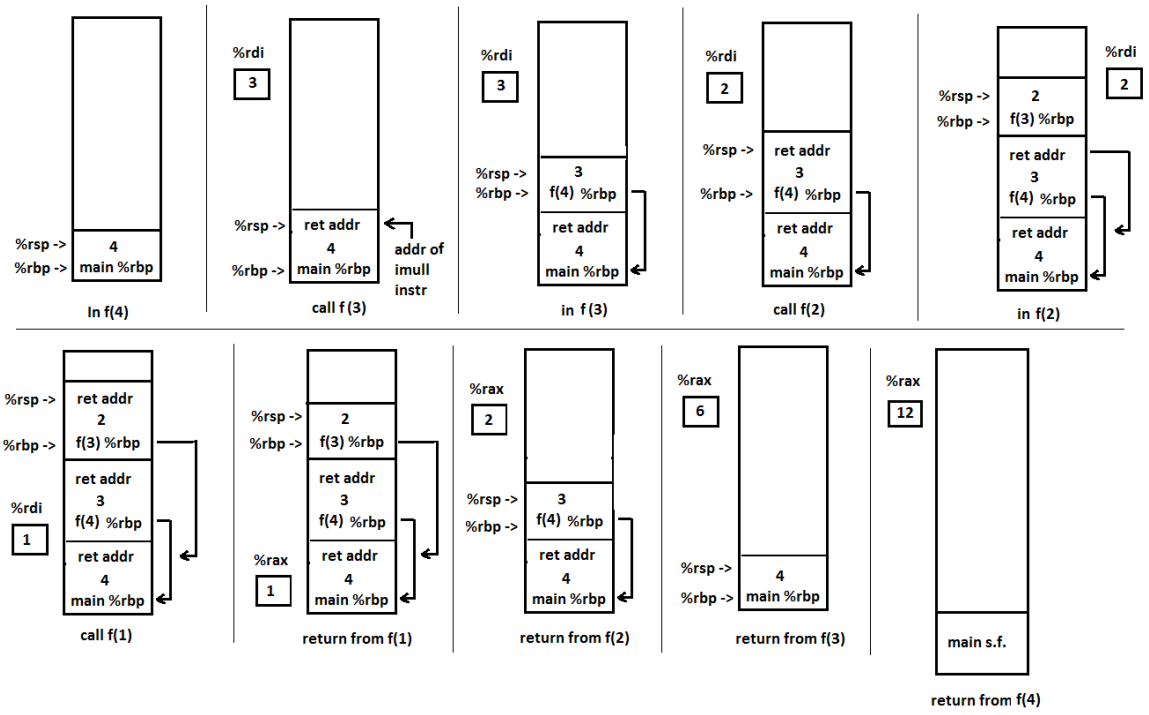
Stack frame : l'espace mémoire sur la pile pour les variables locales

La stack progresse vers le bas



Instructions : push, pop

push %rax
pop %rbx



Functions

Instructions:

- **call** : Ajoute la prochaine instruction (son adresse) sur la stack et déplace l'exécution sur "fonction"
- **ret** : Prend le haut de la pile et retourne à l'adresse

```
call toto
mov %rax, %rbx

toto: // long toto() { long a = 12; return a; }
prologue:
        push %rbp // on le save pour le restorer a la fin
        mov %rsp, %rbp // "nouvelle pile"
end_of_prologue:
        sub $16, %rsp // on prend de la placee pour les variables
        ...
epilogue:
        mov %rbp, %rsp // Flush la zone pour la fonction
        pop %rbp // Restore l'ancien haut de la pile
end_of_epilogue:
        ret
```

Calling conventions

API (Application Programming Interface): int foo(int)
ABI (Application Binary Interface)

Appeler une fonction

- Return value des fonctions (scalaire) : %rax

Paramètres :

- 6 premiers paramètres scalaires : %rdi; %rsi, %edx, %rcx, %r8, %r9
- les suivants sont sur la stack, empilés en ordre inverse
  - obj: printf ou autres pouvant lire les arguments

```
printf(........., a, b, c);

push c
push b
push a
```

Memory operations

Deréférencement mémoire simple :

```
mov (%rax), %rdx
```

Format complet :

```
offset(base, index, scale)
(base + index * scale + offset)
```

- **base** : obligatoire, registre
- **index** : facultatif, registre, default = 0
- **scale** : facultatif, immediat, default = 1
- **offset** : facultatif, immédiat, default = 0

Exemple :

```
mov 8(%rdi, %rcx, 4), %rax
```

```
long *a = malloc(16);
a[0] = 7; // %rax

mov $16, %rdi
call malloc // %rax
mov $7, (%rax)

a[1] = 7; // mov $7, 8(%rax)

for (size_t i = 0; i < 12; i++) { // %rcx
    a[i] = 7; // mov \$7, (%rqx, %rcx, 8)
}

struct A {
    long *s; // %rsi
    long field1; // %rbx
    long field2; // %rcx
};

struct A *d; // %rdx
long *s; // %rsi
long b; // %rbx
long i; // %rcx

b = d->field2; // mov 8(%rdx), %rbx

b = s[0]; // mov (%rsi), %rbx

i = 12; // %rcx
b = s[i]; // mov (%rsi, %rcx, 8), %rbx

b = d[i].field2 // mov 8(%rdx, %rcx, 16), %rbx // recuperation stack
variable
```

Register aliases

```
\%rax     8 bytes
%eax     4 bytes
%ax      2 bytes
%ah, %al: 1 byte  %ah = hi(%ax)
                  %al = lo(%ax)
```

$\Longrightarrow$ Valide pour tous les registres (presque)

```
%rbx, %ebx, %bx, %bh, %bl

%rdi, %edi, %dx, %dil    |
%rsi, %esi, %sx, %sil    | pas de %xxh
```

Instructions size

```
q, quad                  : 8 bytes
l, double ou long        : 4 bytes
w, wide                  : 2 bytes
b, byte                  : 1 byte
```

Équivalences :

```
movb $1, %al  // char
movl $1, %eax // int
movq $1, %rax // long

mov $1, %al // -> movb



struct B {
    long field;
    int int_field;
}

struct B $d; // %rdx
d->int_field = 12;

movl $12, 8(%rdx) // mov = erreur, il ne peut pas deviner la taille du

mov 8(%rdx), %ebx // pas besoin de spe-écifier taille, %ebx la specifie
```