Langages, Compilation, Automates. Partie 10: Table des symboles, variables et appels de fonction

Florian Bridoux

Polytech Nice Sophia

2022-2023

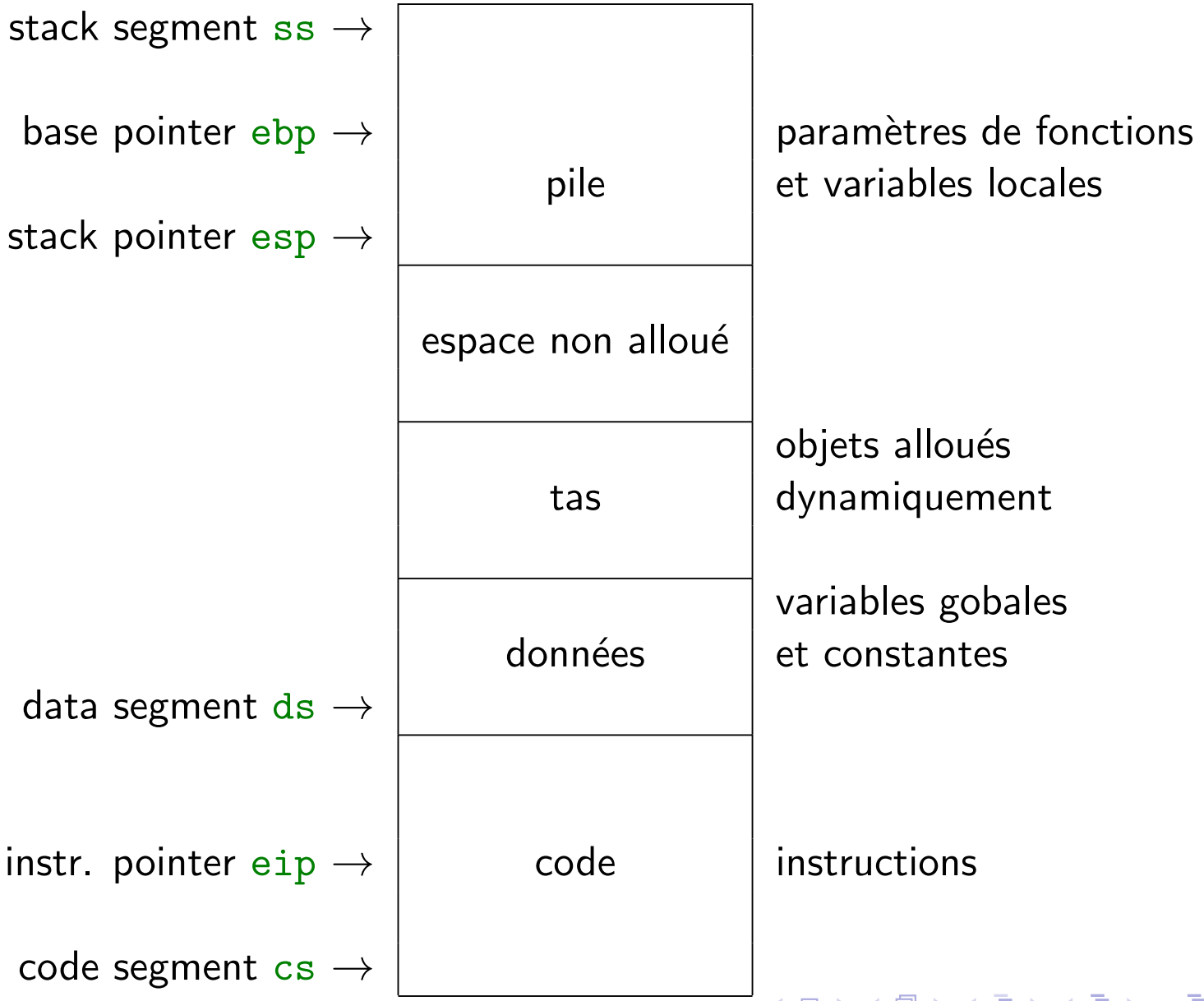
Table des matières

Table des matières

La pile

- Zone de mémoire supposée très grande.
- Le registre utilisé pour le sommet de pile s'appelle `esp`.
- `esp` est l'adresse à laquelle se trouve l'élément en sommet de pile
- Attention, on compte à l'envers en X86 !
  On *augmente* la taille de la pile en *diminuant* `esp`.

Segmentation de la mémoire



La pile: push et pop

Pour empiler les 4 octets contenus dans eax :

```
sub esp, 4
mov [esp], eax
```

ou, plus simplement :

```
push eax
```

Pour dépiler:

```
mov eax, [esp]
add esp, 4
```

ou, plus simplement :

```
pop eax
```

On n'a pas besoin de s'occuper de la valeur initiale de esp, elle est définie comme il faut.

La pile: `ret` et `call`

Pour appeler une procédure:

```
call nom_etiquette_procedure
```

`call` empile la valeur de `eip`, et saute à l'adresse du code indiquée par l'étiquette. C'est l'équivalent de ces **fausses instructions**:

```
push eip - 4
jmp nom_etiquette
```

Pour quitter une procédure:

```
ret
```

`ret` dépile une valeur (a priori, empilé par `call`), et saute à l'adresse représentée par cette valeur. Il faut donc avoir dépilé tout ce qu'on a empilé depuis le `call` correspondant. C'est l'équivalent de cette **fausse instruction**:

```
pop eip
```

Remarque: Le registre `eip` n'est pas un registre général, on ne peut pas y accéder directement avec `mv, pop, push,...`, seulement indirectement avec `call, ret, jmp,...`

Table des matières

Appels de fonction

Les fonctions (et procédures) en assembleur sont simplement des adresses dans le code.

- le passage des arguments se fait à la main,
- la récupération du résultat aussi,
- il n'y a pas de "variables locales".

On utilise la pile pour stocker

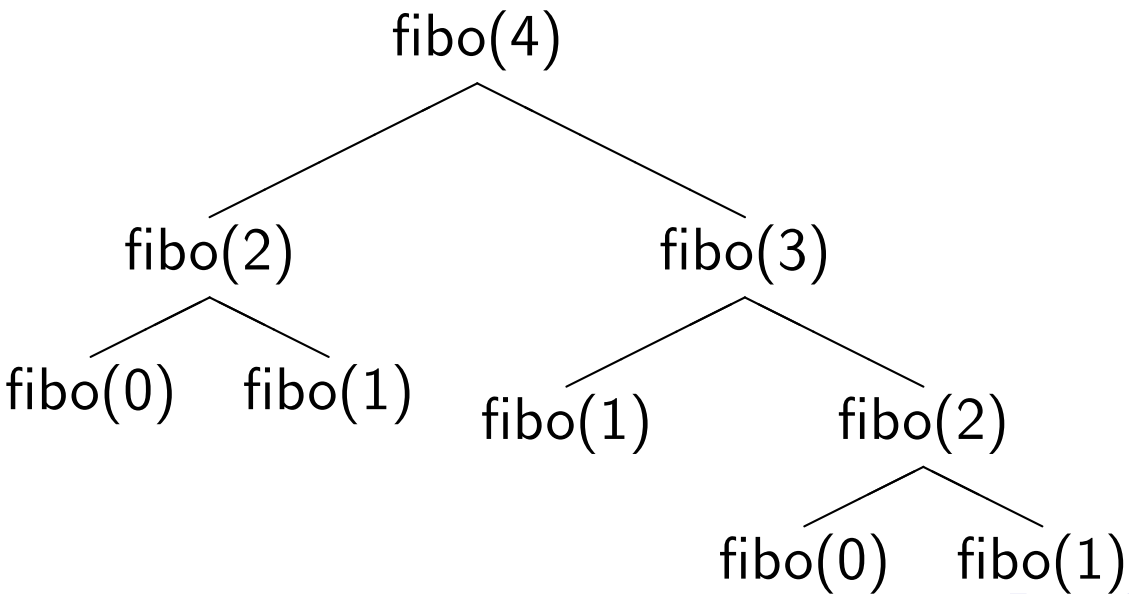
- le résultat,
- les arguments,
- les variables locales.

Arbre d'activation

- À n'importe quel moment, une seule fonction est *active*
- Elle est représentée par une *trame de pile* (aussi appelée *trame d'activation*)
- Les activations successives forment un arbre d'activation
- Une trame de pile est créée lors d'un appel à fonction
- La trame est écrasée quand la fonction finit son activation

Exemple : arbre d'activation de la fonction de Fibonacci

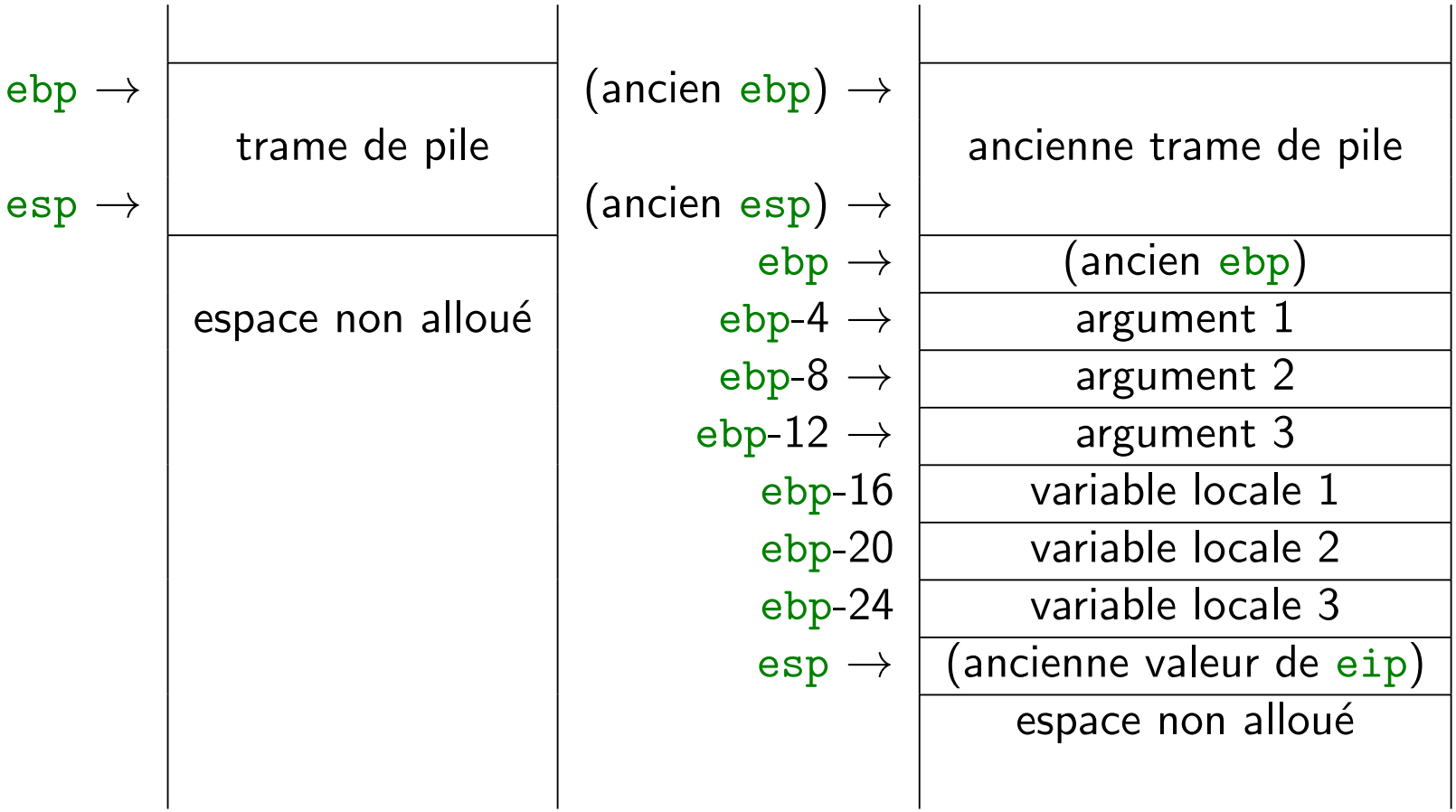
```
entier fibo(entier n){
 si ( n <= 1){
  retourner 1;
 }
 retourner fibo(n-2) + fibo(n-1);
}
ecrire(fibo(4));
```



Trame de pile

- Bloc mémoire sur la pile, contenant toutes les informations sur une fonction en cours d'exécution
  - l'adresse à laquelle poursuivre l'exécution après l'appel
  - la base de la précédente trame de pile.
  - ses paramètres
  - ses variables locales
- Le registre `ebp` pointe sur la base de la trame de pile active,
- Lors de l'appel à une fonction, il faut
  1. sauvegarder l'ancienne valeur de `ebp` dans la pile
  2. donner une nouvelle valeur à `ebp`
- à la sortie de la fonction, il faut
  1. restaurer l'ancienne valeur de `ebp`

Trame de pile



Appel de fonction — côté appelant

Lors d'un appel de fonction, il faut :

1. empiler la valeur de `ebp` avant de la changer (ancien `esp`-4).
2. empiler les arguments,
3. réserver de la mémoire pour les variables locales
4. empiler le pointeur de programme `eip`
   (opération effectuée par `call`),
5. aller à l'adresse de la fonction
   (opération effectuée par `call`),

A l'issue de l'appel, il faut :

1. désallouer la mémoire de la fonction (arguments + variables locales) (=augmenter la valeur de `esp`) ,
2. rétablir la valeur de `ebp`.
3. empiler le retour de la fonction,

Appel de fonction— côté appelée

Quand on sort de la fonction, il faut :

1. stocker le résultat (dans le registre `eax` par exemple),
2. dépiler le pointeur de programme `eip`
   (opération effectuée par `ret`),
3. sauter vers l'adresse de retour
   (opération effectuée par `ret`).

Table des matières

La table des symboles

- Elle rassemble toutes les informations utiles concernant les variables et les fonctions du programme.
- Pour toute variable, elle garde l'information de :
  - son nom
  - son type
  - sa "portée"
  - son adresse en mémoire
- Pour toute fonction, elle garde l'information de :
  - son nom
  - le nombre et le type de ses arguments
  - le type du résultat qu'elle fournit
  - la mémoire utilisée par les variables locales (éventuellement + les paramètres)
- Éventuellement, des informations utiles comme le nombre maximum de variables utilisées simultanément dans une fonction ou le nom de la fonction courante.
- La table des symboles est construite lors du parcours de l'arbre abstrait.

Remplissage de la table

```
entier a = lire();
si ( a > 2){
  entier b = 3;
  booleen c = Vrai;
}
entier d = 4;
si ( a > d){
  entier e = 5;
}
```

nom	type	adresse	imbrication

Remplissage de la table

```
entier a = lire();
si ( a > 2){
  entier b = 3;
  booleen c = Vrai;
}
entier d = 4;
si ( a > d){
  entier e = 5;
}
```

nom	type	adresse	imbrication
a	entier	0	0

Remplissage de la table

```
entier a = lire();
si ( a > 2){
  entier b = 3;
  booleen c = Vrai;
}
entier d = 4;
si ( a > d){
  entier e = 5;
}
```

nom	type	adresse	imbrication
a	entier	0	0
b	entier	4	1

Remplissage de la table

```
entier a = lire();
si ( a > 2){
  entier b = 3;
  booleen c = Vrai;
}
entier d = 4;
si ( a > d){
  entier e = 5;
}
```

nom	type	adresse	imbrication
a	entier	0	0
b	entier	4	1
c	booleen	8	1

Remplissage de la table

```
entier a = lire();
si ( a > 2){
  entier b = 3;
  booleen c = Vrai;
}
entier d = 4;
si ( a > d){
  entier e = 5;
}
```

nom	type	adresse	imbrication
a	entier	0	0
d	entier	4	0

Remplissage de la table

```
entier a = lire();
si ( a > 2){
  entier b = 3;
  booleen c = Vrai;
}
entier d = 4;
si ( a > d){
  entier e = 5;
}
```

nom	type	adresse	imbrication
a	entier	0	0
d	entier	4	0
e	entier	8	1

La table des symboles

Quelques remarques:

- En `FLO`, une variable déclarée dans une boucle (ou dans une fonction, ou dans un si) n'est pas accessible en dehors de celle-ci.
- On peut donc représenter la portée d'une variable en fonction de son imbrication dans un bloc d'instruction: quand on sort du bloc d'instruction dans lequel elle a été déclarée, elle disparaît de la table des symboles.
- Plusieurs variables peuvent donc occuper le même espace mémoire si elles sont déclarées dans différents blocs. Ça optimise la place utilisée par une fonction.

La table des symboles

Quelques remarque:

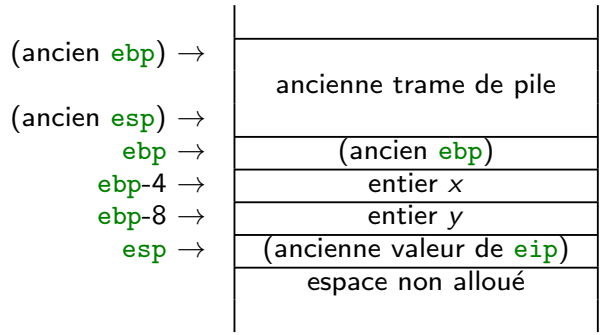
- Le langage `FLO` autorise les fonctions d'appeler des fonctions définies plus bas et les variables peuvent être définie à tout moment du programme.
- Cette souplesse empêche (ou rend très difficile) le fait de ne parcourir qu'une seule fois l'arbre abstrait.
- En effet, il faut faire un premier parcours pour trouver le nom et les caractéristiques des fonctions avant de pouvoir générer les appels de fonction (pour vérifier que les fonctions existent par exemple).
- En particulier, calculer l'espace nécessaire à chaque fonction implique de parcourir sa liste d'instructions pour savoir combien de variables vont être utilisées simultanément.

Table des matières

Exemple

```
entier f(entier x){
        entier y = 8;
        retourner x+y;
}

ecrire(f(5));
```




```
_f:
        ;entier y = 8;
        push    8
        pop     eax
        mov     [ebp - 8],      eax
         ;retourner x+y
        mov     eax,    [ebp - 4]
        push    eax
        mov     eax,    [ebp - 8]
        push    eax
        pop     ebx
        pop     eax
        add     eax,    ebx
        push    eax
        pop     eax
        ret
_main:
        ;ecrire(f(5));
        push    ebp
        mov     esi,    esp
        push    5
        mov     ebp,    esi
        sub     esp,    4
        call    _f
        add     esp,    8
        pop     ebp
        push    eax
        pop     eax
        call    iprintLF
```