

## Pratique du C Pile d'exécution

Licence Informatique — Université Lille 1  
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 4 — 2015-2016

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdf

Si  $x$  est un élément de  $E$ , les relations satisfaites par une pile  $P$  et ces opérations sont :

1.  $\text{estVide}(P_0) = \text{vrai}$
2.  $\text{supprimer}(\text{empiler}(x, P)) = P$
3.  $\text{estVide}(\text{empiler}(x, P)) = \text{faux}$
4.  $\text{depiler}(\text{empiler}(x, P)) = x$

Cette dernière règle caractérise les piles.

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdf

## Notion de pile en assembleur : instructions assembleurs associées (Intel 32)

Les modification de la structure de la pile se font par les instructions :

- **push** reg : (empiler depuis le registre reg). Lorsque l'on empile un élément sur la pile, l'adresse contenue dans %ESP est décrémentée de 4 octets (car un emplacement de la pile est un mot machine de 32 shannons). En effet, lorsque l'on parcourt la pile de la base vers le sommet, les adresses décroissent.
- **pop** reg : (dépiler vers le registre reg). Cette instruction incrémente de 4 octets la valeur de %ESP. Attention, lorsque la pile est vide %ESP pointe sous la pile (l'emplacement mémoire en-dessous de la base de la pile) et un nouveau pop provoquera une erreur.

Il est aussi possible — pour lire ou modifier des valeurs dans la pile — d'utiliser les références mémoire :

`movl %SS :4(%ESP),%EAX`

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdf

Schématiquement, une pile est une structure de données linéaire pour laquelle les insertions et les suppressions d'éléments se font toutes *du même côté*. On parle de structure LIFO : Last In First Out.

Plus formellement, on peut considérer un ensemble d'éléments  $E$  et noter  $\text{Pil}(E)$  l'ensemble de toutes les piles sur  $E$ . Par exemple, les entiers peuvent constituer l'ensemble  $E$  ; la pile vide  $P_0$  est dans  $\text{Pil}(E)$ . Les opérations usuelles sur une pile sont :

- $\text{estVide}$  est une application de  $\text{Pil}(E)$  dans {vrai, faux},  $\text{estVide}(P)$  est vrai si, et seulement si,  $P$  est la pile  $P_0$ .
- $\text{empiler}$  est une application de  $E \times \text{Pil}(E)$  dans  $\text{Pil}(E)$ .
- $\text{depiler}$  est une application de  $\text{Pil}(E) \setminus P_0$  dans  $E$ .
- $\text{supprimer}$  est une application de  $\text{Pil}(E) \setminus P_0$  dans  $\text{Pil}(E)$ .

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdf

## Implantation d'une pile (architecture Intel 32)

Un segment de la mémoire est dévolu à la pile.

Les registres %SS et %ESP sont deux registres servant à gérer la pile :

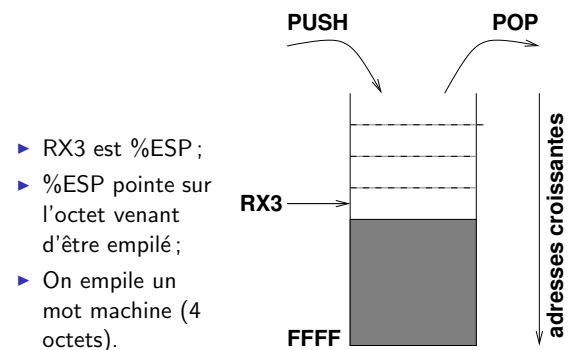
**%SS** (Stack Segment i.e. segment de pile) est un registre 16 bits contenant l'adresse du segment de pile courant ;

L'assembleur vous fera manipuler une pile qui est stockée "en fond de panier", c.à.d dans les adresses les plus hautes de la mémoire. Ainsi, la base de la pile se trouve à l'adresse maximale, et elle s'accroît vers les adresses basses.

**%ESP** (Stack Pointer i.e. pointeur de pile) est le déplacement pour atteindre le sommet de la pile.

Ainsi, %ESP pointe sur le dernier mot machine occupé de la pile en mémoire.

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdf



- RX3 est %ESP ;
- %ESP pointe sur l'octet venant d'être empilé ;
- On empile un mot machine (4 octets).

www.fil.univ-lille1.fr/~sedoglav/C/Cours09.pdf

Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

Les variables *automatiques* (locales à une fonction) sont stockées dans la pile.

Le registre %EBP (Base Pointer) contient un déplacement correspondant à une position dans la pile.

Il sert à pointer sur une donnée dans la pile.

Représentation d'une variable automatique dans la pile :

```
int globale = 7 ;
globale: .long 7
        .text
        .globl main .type main,@function
main:
(void)
{
    int locale ;
    locale = 1 ;
    return 0 ;
}
```

## Représentation de variables de types différents et manipulation

Peu importe le type des variables automatiques que l'on veut représenter, la méthode est la même :

```
int main (void)
{
    struct Gauss{
        int re ;
        int im ;
    } var = {
        .re = 1 ,
        .im = 1 ,
    } ;

    char tab[2] = {'a','b'} ;
    tab[0] += tab[1] ;
    return 0 ;
}
        .text
        .globl main
        .type main, @function
main:
    movl %esp, %ebp
    subl $24, %esp
    ....
    movl $1, -8(%ebp)
    movl $1, -4(%ebp)
    movb $97, -10(%ebp)
    movb $98, -9(%ebp)
    movb -9(%ebp), %dl
    leal -10(%ebp), %eax
    addb %dl, (%eax)
    ....
```

## Les instructions assembleur call et ret

L'appel d'une *routine* se fait par `call label_routine`. Soit un code implanté à l'adresse 1000 et une routine à l'adresse 1100.

```
1000 mov $1,%eax
1002 mov $3,%ebx
1004 CALL label
1007 mov $2,%eax
1009 int 0x80
+----> label: 1100 shl $1,%eax
| 1102 add %ebx,%eax
| 1104 and $7,%eax
| 1106 add '0',%eax
|____ 1108 RET
```

Le sous-programme doit contenir l'instruction `RET` qui permet de revenir au programme appelant.

Lors du `CALL`, %EIP reçoit la valeur 1100, adresse de la prochaine instruction à exécuter, tandis que l'adresse de retour 1007 est empilée sur la pile.

Sur le `RET`, le sommet de pile de valeur 1007 est dépilé, et son contenu est rangé dans %EIP.

Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

## Représentation des variables automatiques

movl %esp, %ebp		subl \$8, %esp	
		%ESP	vide
			vide
%ESP		%EBP	data
			4 octets
			:
FFFF		FFFF	bas de pile
%EBP mis en place		réservé de l'espace	
		movl \$1, -4(%ebp)	
		%ESP	vide
			1
		%EBP	data
			4 octets
			:
		FFFF	bas de pile

## Rappel sur les registres associés à l'exécution du code

Le code exécutable d'un programme se présente sous la forme d'une suite finie contigüe d'octets stockée dans un segment de la mémoire.

**Le registre %CS (Code Segment).** Ce registre 16 bits contient le numéro du segment mémoire dans lequel sont stocké les instructions assembleur du code à exécuter. On ne peut pas accéder directement à ce registre.

**Le registre %EIP (Instruction Pointer).** Le registre %EIP contient l'offset de la prochaine instruction à exécuter. Il est modifié automatique à chaque exécution et peut être manipulé par des instruction du type `jmp`, `call`, `ret`, etc. On ne peut pas accéder directement à ce registre.

## Appel de fonction en C (sans paramètre)

```
int UN (void)
{
    return 1 ;
}
        .text
        .globl UN
UN:
    ...
    movl $1, %eax
    ...
    ret

        .globl main
main:
    ...
    movl %esp, %ebp
    subl $8, %esp
    ...
    call UN
    movl %eax, -4(%ebp)
    movl $0, %eax
    ...
    ret
/* Remarquez que la valeur de retour transite par le registre %eax */
```

Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

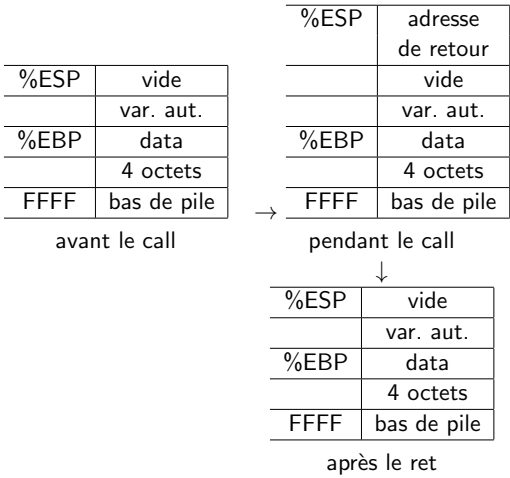
Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

Ce qui se passe sur la pile



Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

```
int PlusUn(int par)
{
    return par+1 ;
}

main:
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    movl 8(%ebp), %eax
    incl %eax
    leave ret

main:
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    movl $7, -4(%ebp)
    subl $12, %esp
    pushl -4(%ebp)
    call PlusUn
    addl $16, %esp
    leave
    ret
```

Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

Comme toutes variables automatiques, les paramètres sont stockés dans la pile. Dans une fonction C les variables et les paramètres sont gérés en suivant les étapes :

1. Sauver le pointeur de base de pile courant sur la pile ;
2. Se donner un nouveau pointeur de base de pile ;
3. Déclarer les variables automatiques sur la pile ;  
En cas d'appel de fonction avec passage de paramètres :
4. Empiler les paramètres sur la pile ;
5. Effectuer un call (qui empile automatiquement l'adresse de retour sur la pile) ;  
Dans la fonction appelée, on peut utiliser l'espace de pile associée aux paramètres ; Cette fonction se termine par un ret (qui dépile automatiquement l'adresse de retour) ;
6. Supprimer l'espace de pile — maintenant inutile — associé aux paramètres.

Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

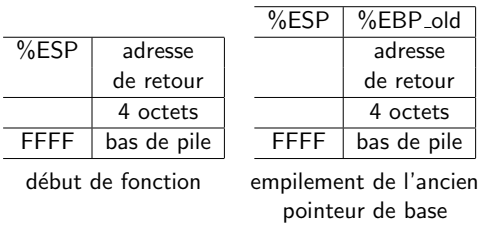
Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

Ce qui se passe sur la pile



Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

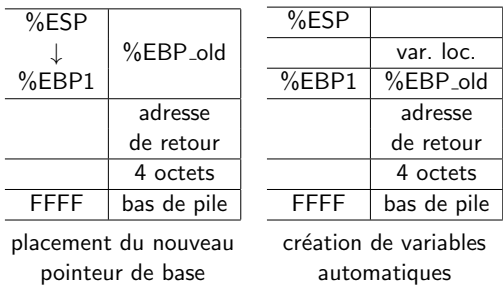
Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

Ce qui se passe sur la pile



Pratique du C  
Pile d'exécution

Notion de pile en assembleur : définitions théoriques et pratiques

Représentation des variables automatiques

Appel de fonction en C

Passage de paramètres par la pile

Passage de paramètre : une copie est faite sur la pile

V91 (04-03-2014)

