Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

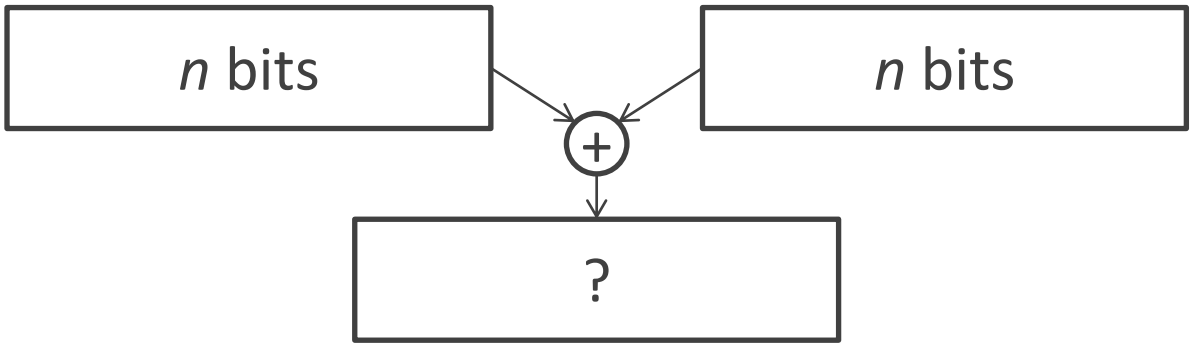
Utilisation de la pile en assembleur

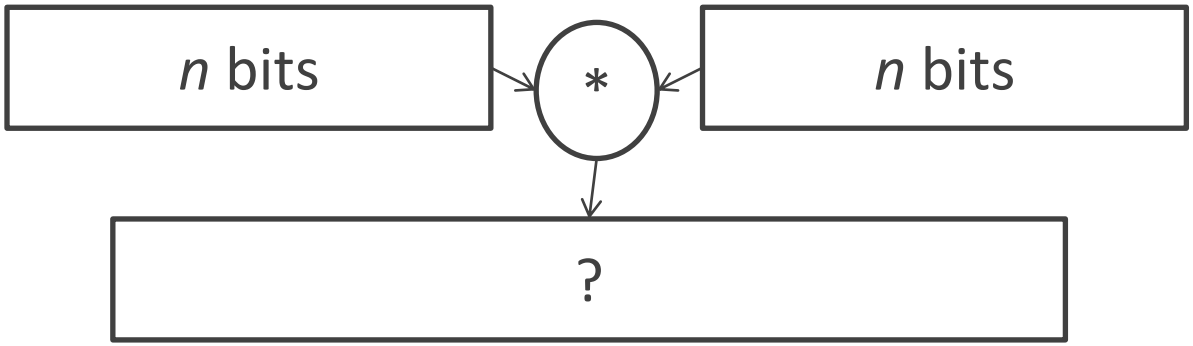
Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Addition et multiplication





Multiplication

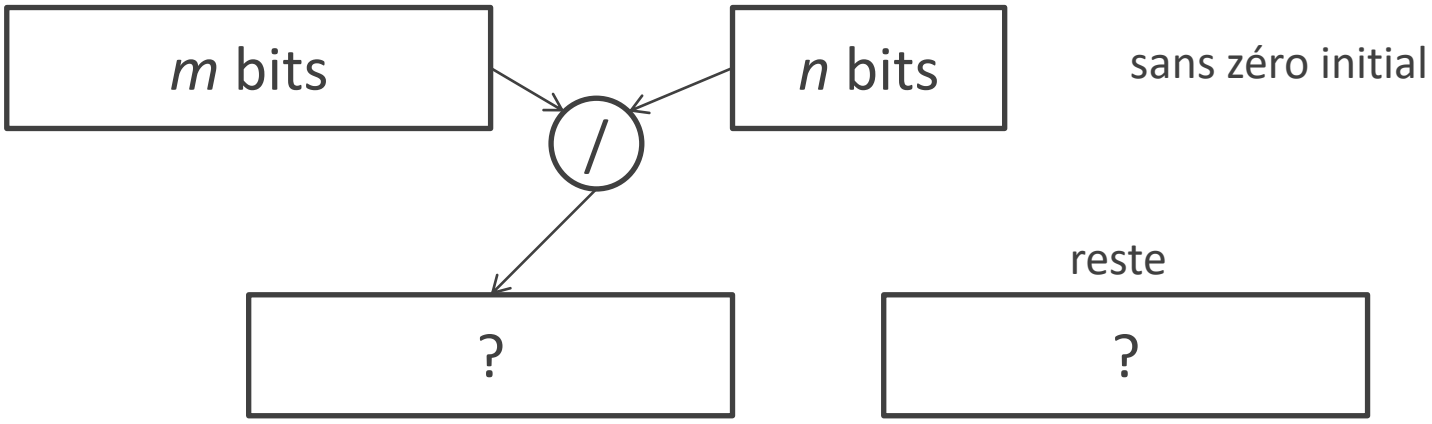
```
_start:
    mov     rax, 90
    mov     rbx, 9
    imul    rax, rbx
    call    print_registers
    mov     rax, 60
    mov     rdi, 0
    syscall
```

`imul rax, rbx` ; rax := rax * rbx

Les opérandes sont interprétés comme entiers signés

Si la valeur absolue du produit ne tient pas sur 8 octets, il est **tronqué** : on perd les 8 octets de poids fort

Division entière



sans zéro initial

Division entière

```
_start:
    mov     rax, 90
    mov     rbx, 9
    idiv    bl      ; divise ax par bl
    call    print_registers
    mov     rax, 60
    mov     rdi, 0
    syscall
```

`idiv x` division entière par `x`

Les opérandes sont interprétés comme entiers signés

Les détails dépendent de la taille de `x`

`x` ne peut pas être une constante

Pour diviser par une constante, la copier dans un registre

Division entière

```
_start:
mov     rdx, 0
mov     rax, 90
mov     rbx, 9
idiv    bx          ; divise ax par bx
call    print_registers
```

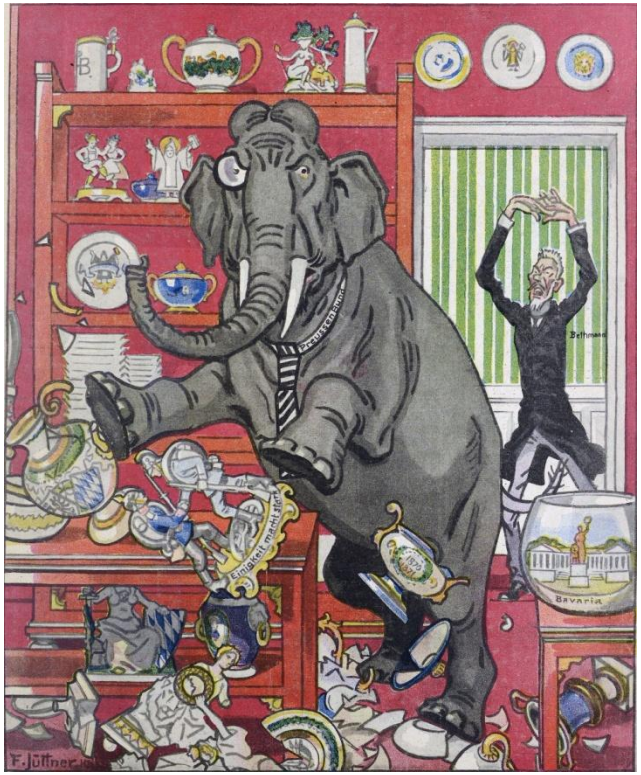
Exemple	taille de x	divise...	par...	met le quotient dans...	et le reste dans...
`idiv bl`	1 octet	`ax`	`x`	`al`	`ah`
`idiv bx`	2 octets	`dx:ax`	`x`	`ax`	`dx`
`idiv ebx`	4 octets	`edx:eax`	`x`	`eax`	`edx`
`idiv rbx`	8 octets	`rdx:rax`	`x`	`rax`	`rdx`

`dx:ax` est l'entier obtenu en concaténant les 2 octets de **`dx`** avec ceux de **`ax`**
Exemple : **`rax`**=0x6000, **`rdx`**=0x87, **`dx:ax`**=0x876000
Si le dividende est dans **`rax`**, mettre à zéro **`rdx`** avant d'utiliser **`idiv`**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Effets collatéraux



Franz Jüttner

Beaucoup d'instructions lisent ou même écrasent le contenu d'un registre qu'elles ne mentionnent pas

`rflags` est modifié par les instructions arithmétiques, **`cmp`**...

`rdx` est écrasé par les résultats de **`idiv`**

Exercice

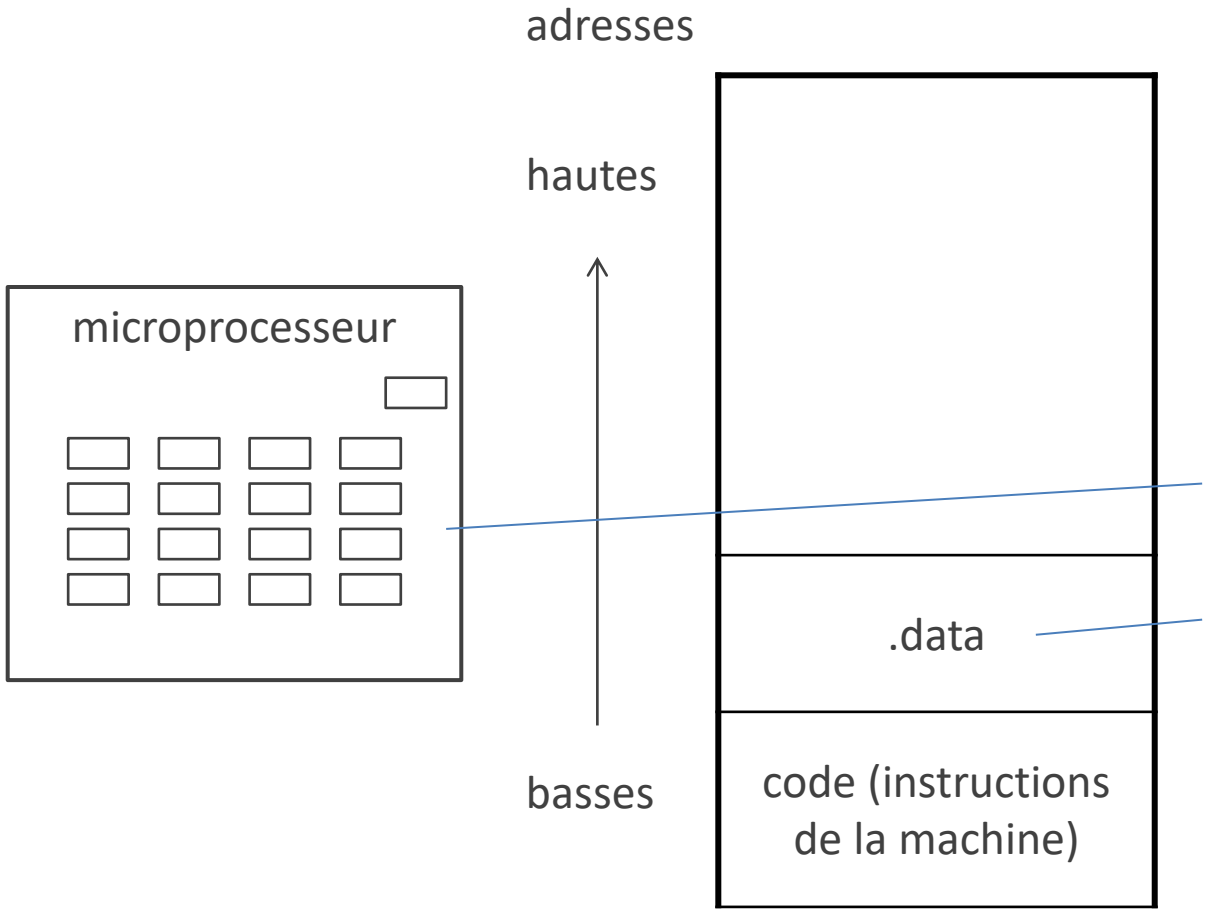
Vérifier si les 6 octets de poids fort de **`rax`** sont à zéro

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Sauvegarder les résultats intermédiaires d'un calcul



Dans des registres ?

À des adresses statiques ?

Calcul dans des registres ou des adresses statiques

```
mov eax, dword [a]
mov ecx, dword [x]
imul eax, ecx
imul eax, ecx
imul ecx, dword [b]
add eax, ecx
add eax, dword [c]
```

```
mov eax, dword [a]
imul eax, dword [x]
imul eax, dword [x]
mov dword [result], eax
mov eax, dword [b]
imul eax, dword [x]
add eax, dword [result]
add eax, dword [c]
```

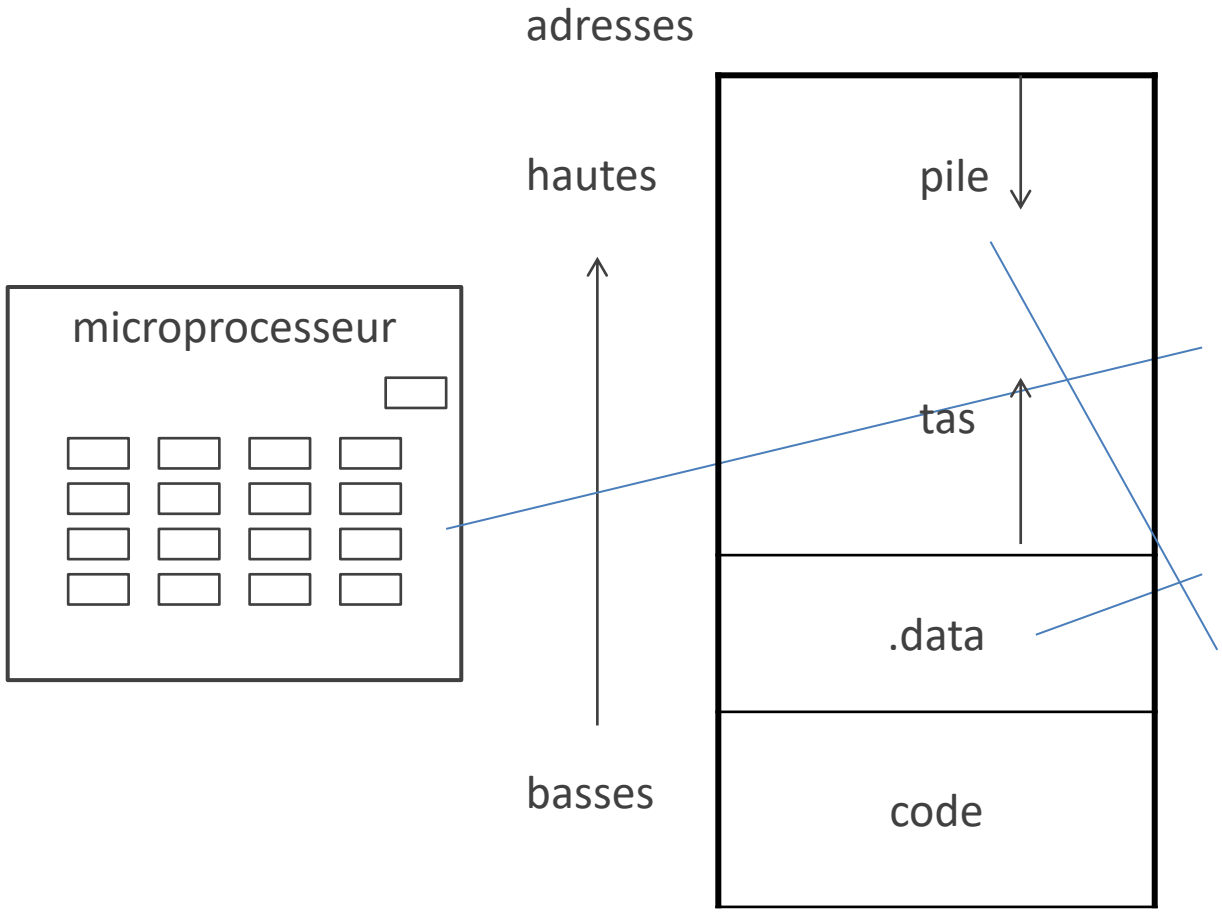
Calculer ax^2+bx+c

- avec 2 registres

- avec 1 registre et 1 adresse statique

```
section .data
a: dd 3
b: dd -12
c: dd 8
x: dd -154
result: dd 0
```

Sauvegarder les résultats intermédiaires d'un calcul



Dans des registres

Rapide mais le nombre de registres est fini

À des adresses statiques

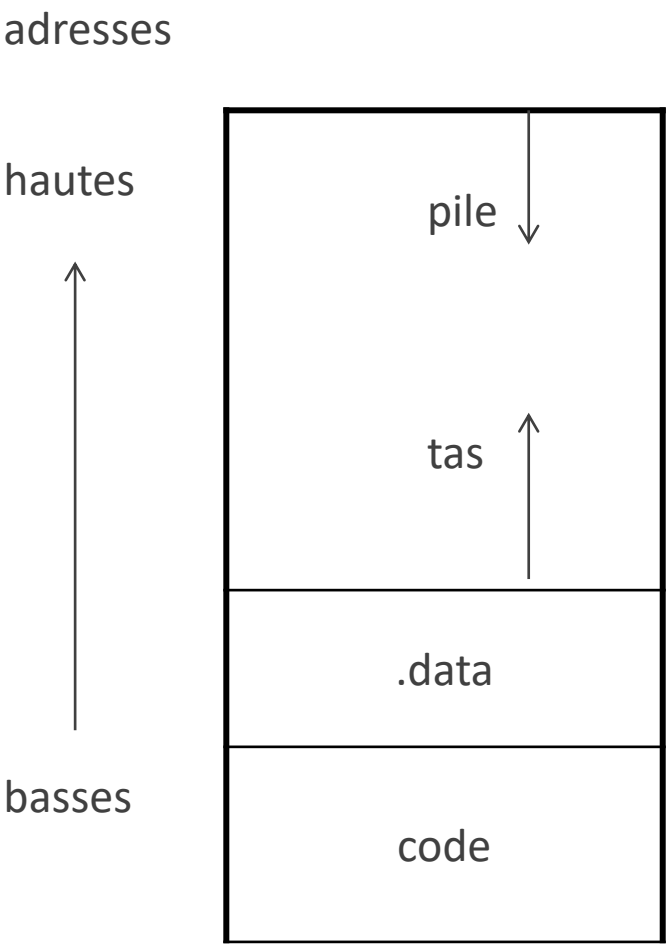
Durée : tout le programme

Dans la pile

Durée limitée

Récursivité

La pile



"Fond" de pile dans les adresses hautes
"Sommet" de pile dans les adresses basses

`push x` empiler **`x`** (**`x`** peut être une constante)

`pop x` dépiler et copier dans **`x`** (**`x`** ne peut pas être une constante)

En mode 64 bits, **`push x`** empile 8 octets

Si **`x`** est un registre ou une zone de mémoire, il doit occuper 8 octets

Si **`x`** est une constante occupant moins de 8 octets, extension de signe

Calcul avec la pile

```
mov eax, dword [a]
push rax
mov ecx, dword [x]
push rcx
pop rdx         ; x
pop rax         ; a
imul eax, edx
push rax        ; ax
push rcx        ; x
pop rdx         ; x
pop rax         ; ax
imul eax, edx
push rax        ; ax2
mov eax, dword [b]
push rax        ; b
push ecx        ; x
pop rdx         ; x
pop rax         ; b
imul eax, edx
push rax        ; bx
```

```
pop rdx         ; bx
pop rax         ; ax2
add eax, edx
push rax        ; ax2+bx
mov eax, dword [c]
push rax
pop rdx         ; c
pop rax         ; ax2+bx
add eax, edx
```

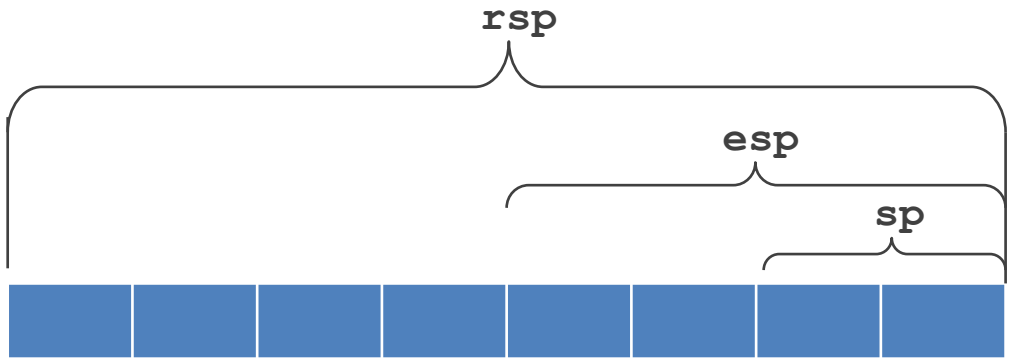
Calculer ax^2+bx+c dans la pile

Méthode

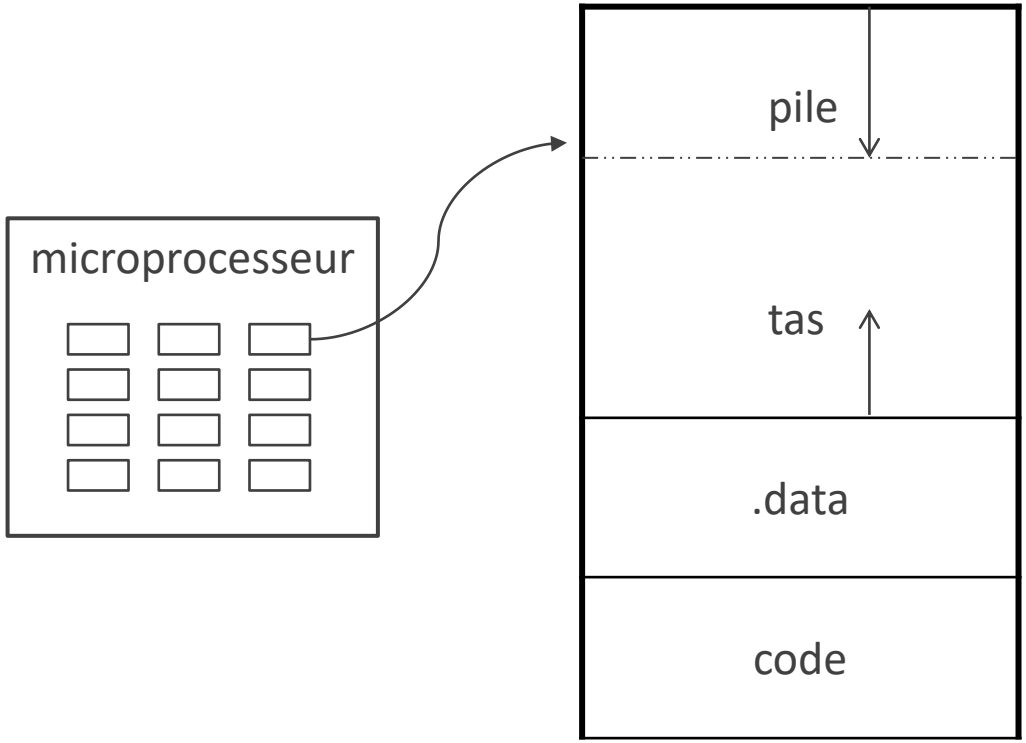
Empiler 2 opérandes

Dépiler 2 opérandes

Calculer le résultat



Le registre `rsp`

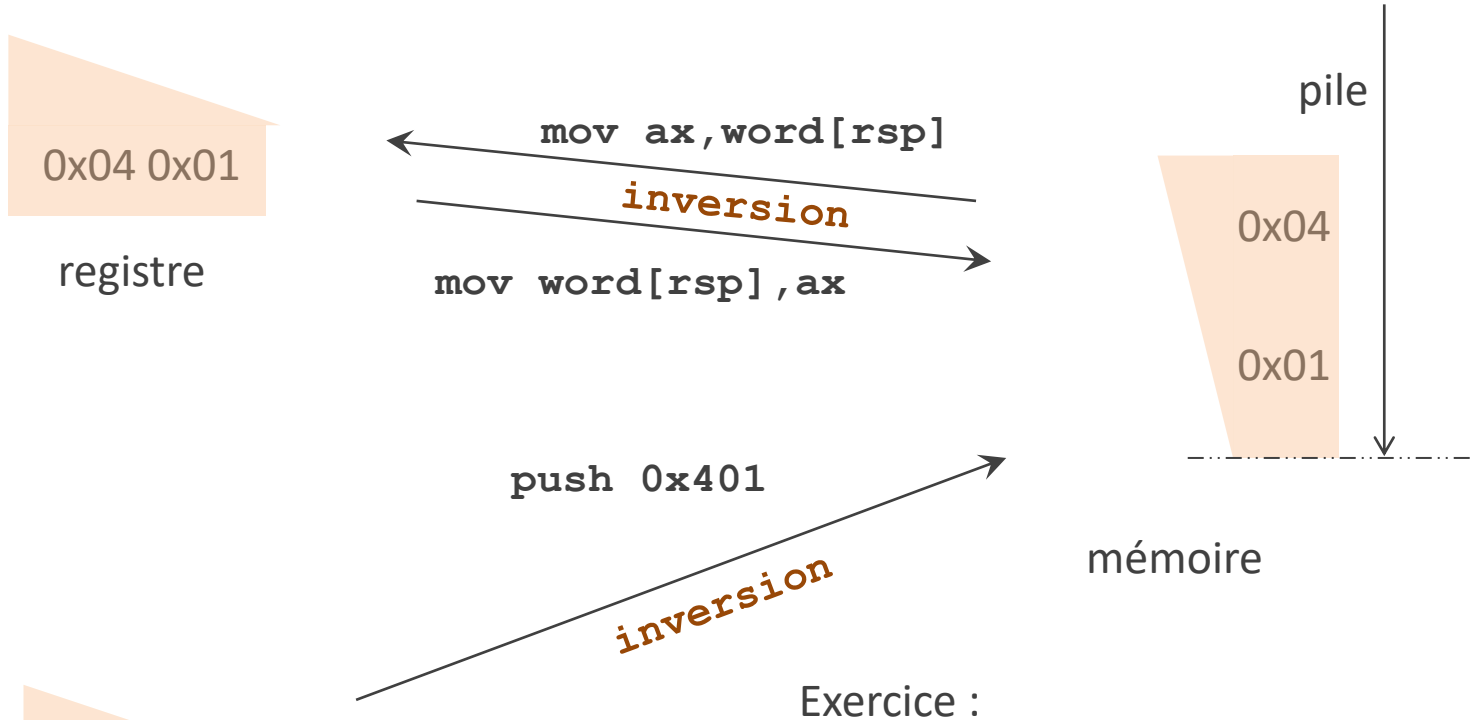


sp : stack pointer

`rsp` pointe sur l'octet d'adresse la plus basse dans la pile (la dernière donnée empilée)

```
push 0x401
push 'math'
mov rcx, qword [rsp]
mov rax, qword [rsp+8]
```

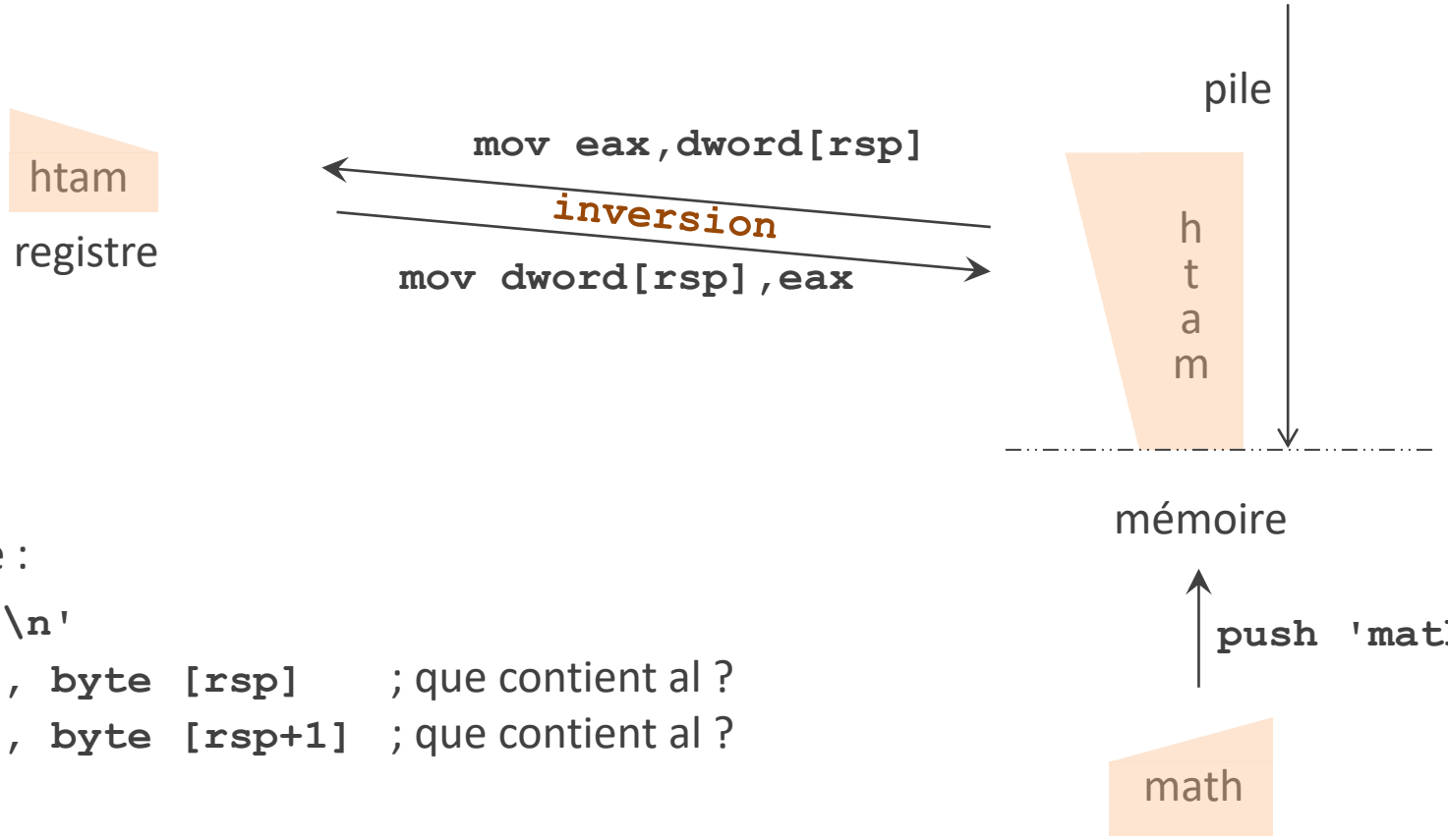
Orientation little-endian



Exercice :

```
push 0x401
mov ax,word[rsp]    ; que contient ax ?
mov ax,word[rsp+1]  ; que contient ax ?
```

Orientation little-endian



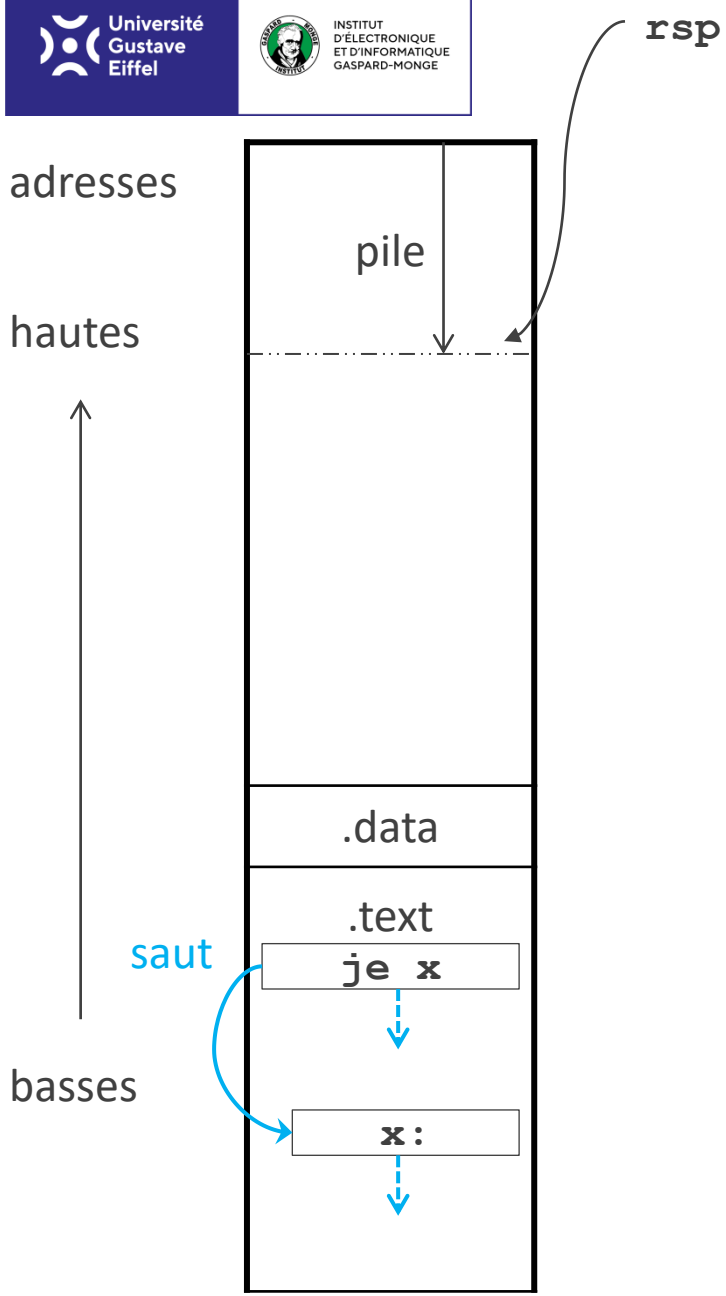
Exercice :

```
push '\n'
mov al, byte [rsp]     ; que contient al ?
mov al, byte [rsp+1]   ; que contient al ?
```

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

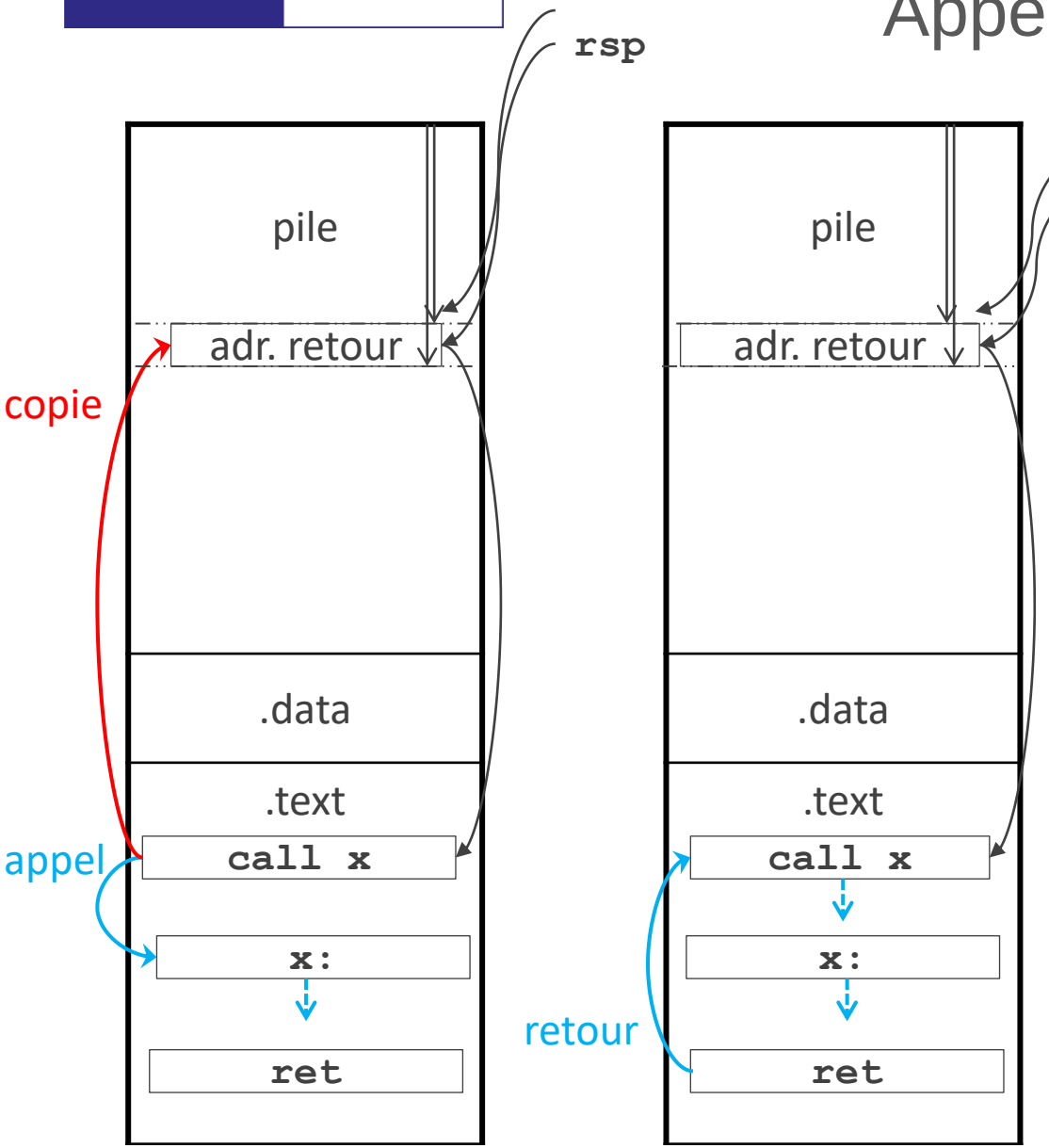


Sauts

`jmp x`	saut (sans retour)
`cmp x, y`	avant saut conditionnel
`je x`	saut conditionnel (sans retour)
`jne x`	
`jg x`	
`jng x`	

La seule possibilité de retour est de faire un autre saut

Appel de fonction et retour



`call x` appeler une étiquette `x`
`ret` retour à l'appel

L'instruction `call` empile l'**adresse de retour**

L'instruction `ret` dépile l'adresse de retour et l'utilise

Il faut donc dépiler tout ce qu'on a empilé depuis l'étiquette cible de l'appel

Dépiler avant le retour

```
push    'the end'
mov     rax, rsp
call    sprint
pop     rax
```

```
sprintLF:
call    sprint
push    '\n'
mov     rax, rsp
call    sprint
pop     rax
ret
```

Exemple

`sprint` affiche la chaine qui commence à l'adresse `rax` et se termine par 0

Exercice

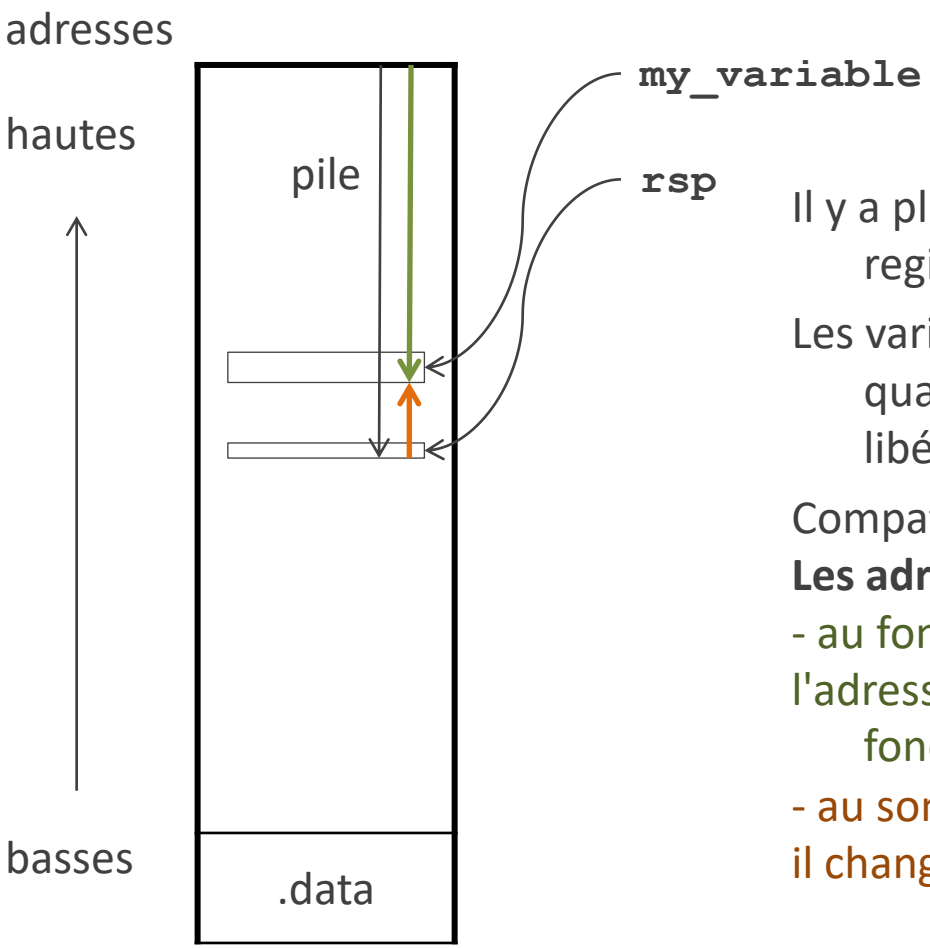
Compléter `sprintLF` pour qu'il affiche un caractère de fin de ligne après la chaine

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Sauvegarder des variables dans la pile



Il y a plus d'espace disponible que dans les registres

Les variables peuvent être locales à une fonction : quand la fonction se termine, la mémoire est libérée

Compatible avec les fonctions récursives

Les adresses sont relatives à quoi ?

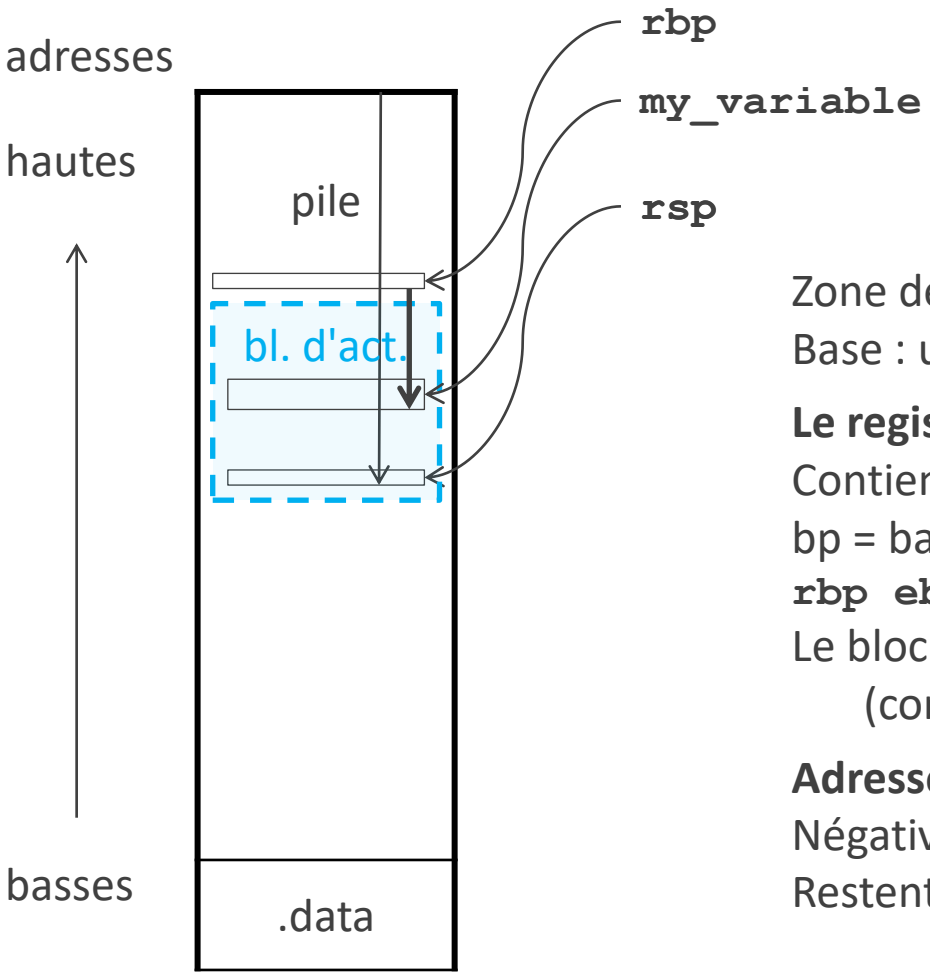
- au fond de la pile ?

l'adresse relative changerait avec chaque appel de fonction

- au sommet de pile ?

il change tout le temps avec les `push` et les `pop`

Bloc d'activation (*stack frame*)

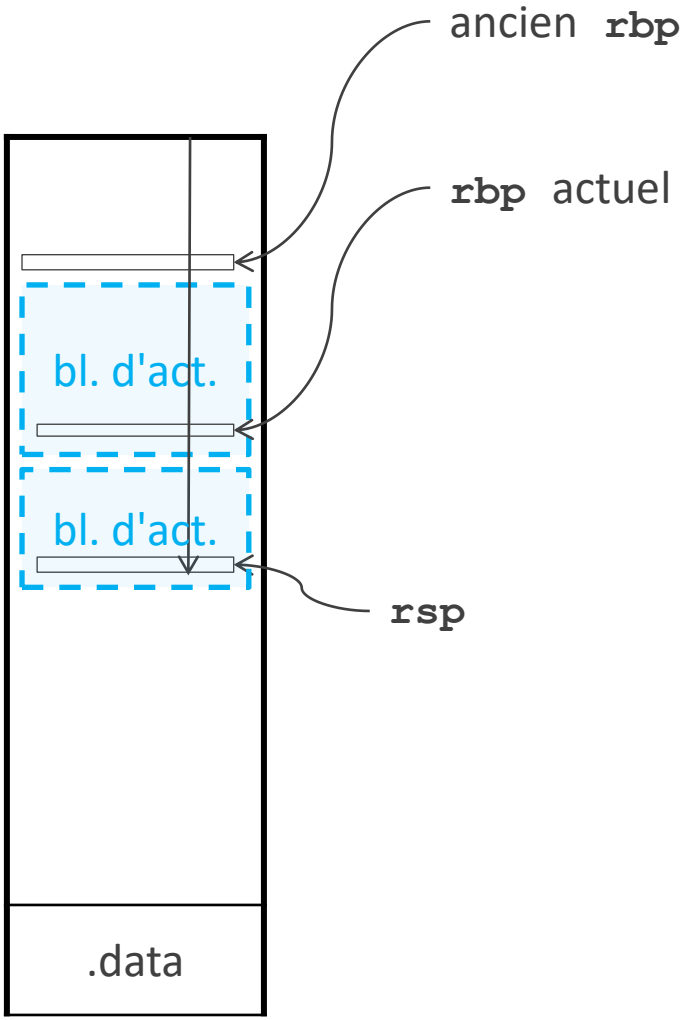


Zone de mémoire sous le sommet de pile
Base : un point plus stable que le sommet de pile

Le registre `rbp`
Contient l'adresse de la base du bloc d'activation
bp = base pointer
`rbp ebp bp`
Le bloc est entre **`rbp`** (non compris) et **`rsp`** (compris)

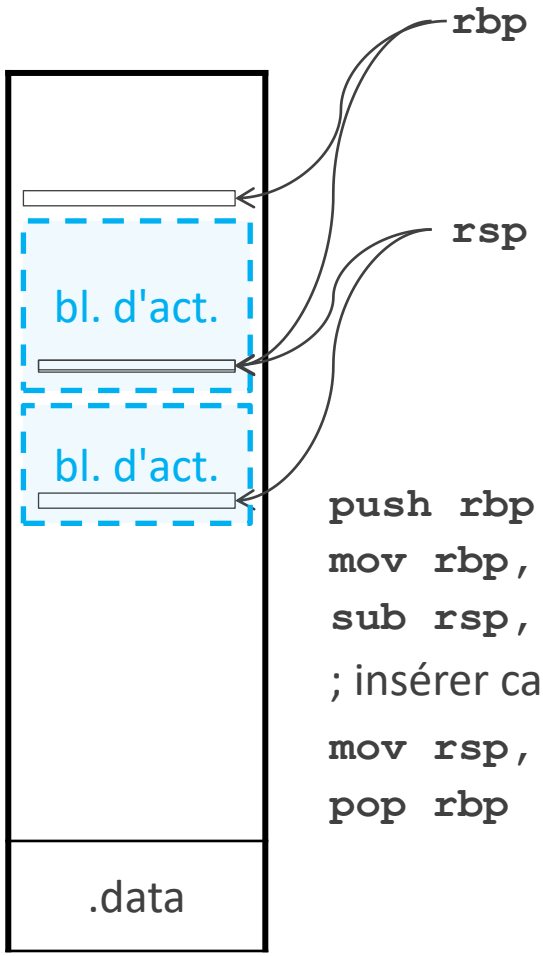
Adresses relatives dans le bloc d'activation
Négatives par rapport à **`rbp`**
Restent valides malgré les **`push`** et les **`pop`**

Pile de blocs d'activation



Réserver un nouveau bloc d'activation
On n'a plus directement accès aux variables du bloc précédent
Libérer le bloc d'activation courant
Restaurer l'ancienne valeur de `rbp`

Pile de blocs d'activation




```
push rbp
mov rbp, rsp
sub rsp, 24
; insérer calculs ici
mov rsp, rbp
pop rbp
```

Réserver un bloc d'activation

Sauvegarder le `rbp` précédent sur la pile

Écraser `rbp` avec sa nouvelle valeur

Déplacer `rsp` de la taille du nouveau bloc

Libérer un bloc d'activation

Restaurer le `rsp` précédent

Restaurer le `rbp` précédent

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Utiliser le bloc d'activation

```
push rbp
mov rbp, rsp
sub rsp, 8
mov qword [rbp-8], 0 ; sum = 0
  begin_loop:
  cmp rbx, 0
  jle end_loop
add [rbp-8], rbx      ; sum += rbx
dec rbx
jmp begin_loop
  end_loop:
  mov rax, [rbp-8]    ; rax = sum
  mov rsp, rbp
  pop rbp
```

On utilise l'adresse par rapport à **`rbp`** et non par rapport à **`rsp`**

Pendant la durée de vie d'un bloc d'activation,

- **`rbp`** ne change jamais
- **`rsp`** change dès qu'on utilise la pile pour faire un calcul ou sauvegarder une donnée

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Conventions d'appel

```
final_msg:
push    'the end'
mov     rax, rsp
call    sprint
mov     rbx, [rax]
mov     qword [msg], rbx
pop     rax
ret


sprintLF:
call    sprint
push    '\n'
mov     rax, rsp
call    sprint
pop     rax
ret
```

Le même registre sert à plusieurs choses

Les registres sont en nombre fini

Chaque registre est disponible depuis tout le code

Exemple

`sprint` affiche la chaine qui commence à l'adresse `rax` et se termine par 0

`sprintLF` aussi, en ajoutant \n

`sprintLF` écrase le `rax` d'entrée après usage

`final_msg` suppose que `sprint` n'écrase pas `rax` après usage...

Conventions d'appel

Discipline pour assurer entre autres la conservation des valeurs dans les registres

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Conventions d'appel AMD 64

- **cdecl** avec processeurs IA-32
- **vectorcall** sous Windows
- **register** pour Delphi avec processeurs IA-32

Conventions sur l'utilisation des registres et de la pile lors de l'appel et du retour des fonctions

Valables quand 3 conditions sont réunies :

- processeurs X86-64
- Linux
- C ou C++

D'autres conventions sont valables dans d'autres contextes

C'est de la responsabilité des développeurs de respecter ces conventions

Un même registre utilisé dans plusieurs fonctions

2 solutions :

La fonction 2 sauvegarde la valeur juste avant d'utiliser le registre
Elle assure la conservation de la valeur pour la fonction 1

La fonction 1 sauvegarde la valeur juste avant d'appeler la fonction 2
Elle assure la conservation de la valeur pour elle-même

Les conventions AMD64 imposent une solution ou l'autre, suivant les registres

Registres non volatils (*callee-save registers, preserved registers*)

By Ed Yourdon [CC BY-SA 2.0 (http://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons

```
my_function:
push rbp
mov rbp, rsp
sub rsp, 24
; insérer les calculs ici
mov rsp, rbp
pop rbp
ret
```

Le fait d'appeler une fonction ne les écrasera pas

Les fonctions qui les écrasent doivent restaurer au retour la même valeur qu'à l'appel

`rbx`, `rbp`, `r12`, `r13`, `r14`, `r15`

Idéal dans du code

- qui ne peut pas être appelé (pas d'instruction `ret`)
- même s'il appelle des fonctions

Limites

Seulement 6 registres non volatils

Registres volatils (*caller-save registers*, *scratch registers*)

By PiccoloNamek (English wikipedia) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons

```
mov   qword [table], 7
mov   rax, rdi
add   rax, [table]
push  rax
push  rdi
call  print_stack
pop   rdi
pop   rax
```

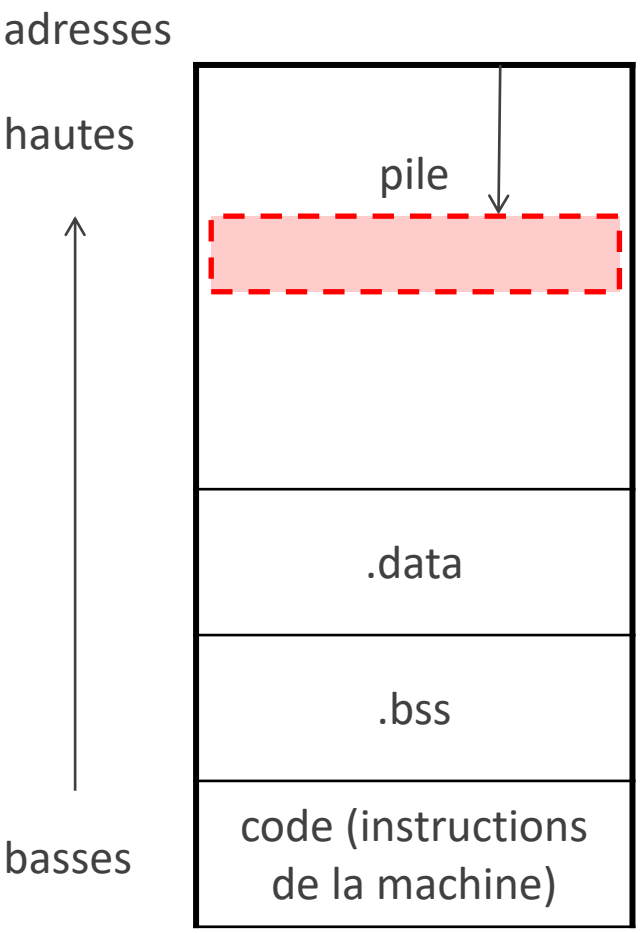
- Le fait d'appeler une fonction peut les écraser
- Si on veut appeler une fonction alors qu'on les utilise, on doit les sauvegarder avant l'appel et les restaurer après
- Tous les registres sauf **rbx**, **rbp** et **r12-15**
- Déjà connus : **rax**, **rcx**, **rdx**, **rdi**, **rsi**, **rsp**
- Une nouvelle sélection de registres de 64 bits : **r8**, **r9**, **r10**, **r11**
- Idéal dans une fonction qui n'appelle aucune fonction (fonction-feuille)

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Zone rouge



Conventions d'appel AMD 64 :
Zone de 128 octets juste au-dessous de la pile
Protégée des interruptions
Adresses relatives négatives

Zone rouge

```
mov [rsp-8], rax
sub [rsp-8], rcx
mov [rsp-16], rcx
sub [rsp-16], rdx
mov [rsp-24], rdx
sub [rsp-24], rax
mov rdi, [rsp-8]
cmp rdi, [rsp-16]
je  xeqy
   cmp rdi, [rsp-24]
   je  xeqz
mov rdi, [rsp-16]
cmp rdi, [rsp-24]
je  yeqz
   jmp notequal
```

Limites

Les instructions `call` et `push` écrasent la zone rouge

Les instructions `push` et `pop` changent les adresses relatives par rapport à `rsp`

Technique réservée aux fonctions qui ne contiennent aucun appel

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

```
%define SYS_EXIT 60
%define SYS_READ  0
%define SYS_WRITE 1
%define x rsp-8
%define y rsp-16
%define z rsp-24
section .text
mov [x], rax
sub [x], rcx
mov [y], rcx
sub [y], rdx
mov [z], rdx
sub [z], rax
mov rdi, [x]
cmp rdi, [y]
je  xeqy
   cmp rdi, [z]
   je  xeqz
mov rdi, [y]
cmp rdi, [z]
je  yeqz
   jmp notequal
```

Macros

%define permet de définir des macros comme en C

%define MACRO_NAME expression