Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Fonctions en assembleur

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Passer les paramètres et la valeur de retour

```
print_registers:
push rbp
mov rbp, rsp
mov r8, r14
mov rcx, r13
mov rdx, r12
mov rsi, rbx
mov rdi, format_registers
mov rax, 0
call printf
pop rbp
ret
```

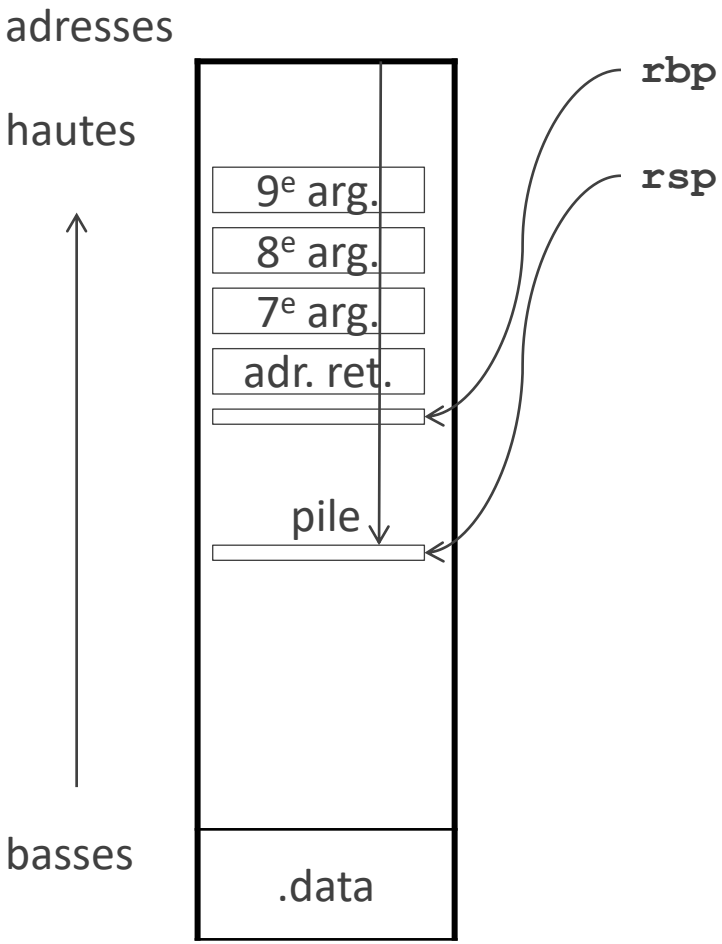
Fonctions en nasm

Instructions `call` et `ret`

Aucune notion de paramètres ni de valeur de retour

On utilise les registres et la pile

Respecter les conventions d'appel AMD 64



Paramètres entiers

Les 6 premiers

Dans les registres `rdi`, `rsi`, `rdx`, `rcx`, `r8` et `r9`

Tous volatils

Si ces registres sont déjà utilisés, les sauvegarder sur la pile avant l'appel et les restaurer après

S'il y a plus de 6 paramètres entiers

Sur 8 octets chacun

Empilés avant l'appel

En commençant par le dernier (pour que l'adresse relative par rapport à `rbp` ne dépende pas du nombre d'arguments)

Paramètres entiers

Le DJ signe à la radio récemment pour des hits neufs

Dix sirènes radieuses récoltent des huitres neuves

Dis-moi si la radio raconte la révolution

Pour se souvenir de la liste des registres des 6 premiers arguments entiers

Laquelle des trois phrases marche le mieux ?
Votez sur https://xoyondo.com/ap/j7nELTvPGQW7jn2

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Valeur de retour entière

Dans **`rax`** (volatil)

S'il est utilisé au moment de l'appel, le sauvegarder sur la pile avant l'appel et le restaurer après

Alignement de la pile

```
printf("...",x,y,z,t);
```

```
print_registers:
push rbp
mov rbp, rsp
mov r8, r14
mov rcx, r13
mov rdx, r12
mov rsi, rbx
mov rdi, format_registers
mov rax, 0
call printf
pop rbp
ret
```

Si la pile était alignée juste avant l'appel à **`print_registers`**, elle est à nouveau alignée ici

Dans certaines configurations, la taille de la pile doit être un multiple de 16 octets juste avant un appel

C'est pour faciliter l'emploi de certaines instructions qui présupposent que l'adresse d'un opérande est un multiple de 16

Alignement de la pile

```
my_function:
push rbp
(... conditional jumps...)
mov r14, rsp
(... how to align stack?)
call printf
mov rsp, r14
pop rbp
ret
```

Cas où on ne peut pas prévoir la taille de la pile

La taille de la pile doit être un multiple de 16 octets juste avant un appel

Comment placer dans **rsp** le plus grand multiple de 16 inférieur ou égal à **rsp** ?

Utiliser la division entière de **rsp** par 16

```
mov rax, rsp
mov rbx, 16
idiv bl   ; divides ax by bl, remainder in ah
sub rsp, ah
```

Comment restaurer **rsp** juste après l'appel ?

Il faut l'avoir sauvegardé avant d'aligner

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Appeler du C depuis nasm

```
extern printf
section .data
    display_format db "rbx:%ld r12:%ld r13:%ld r14:%ld\n", 0
section .text
display_registers:
push rbp
mov rbp, rsp
mov r8, r14
mov rcx, r13
mov rdx, r12
mov rsi, rbx
mov rdi, display_format
mov rax, 0
call printf WRT ..plt
pop rbp
ret
```

Déclarer le nom de la fonction en **`extern`**

Utiliser l'instruction **`call`** avec le nom de la fonction et éventuellement **`WRT ..plt`**

Respecter les conventions d'appel :

- placer les arguments dans les registres avant l'appel
- récupérer la valeur de retour après l'appel

Définir une fonction `main`

```
global  main
extern  puts
section .text
main:
push rdi    ; sauvegarde avant utilisation comme argument de puts
push rsi    ; registre volatil
sub  rsp,8    ; aligne la pile
mov  rdi,[rsi] ; argument
call puts WRT ..plt
add  rsp,8      ; restaure rsp
pop  rsi
pop  rdi
add  rsi,8 ; prochaine chaine
dec  rdi        ; compte
jnz main        ; boucle
ret
```

Définir une **`fonction`** d'étiquette **`main`**

La fonction sera appelée avec les arguments **`argc`** et **`argv`** comme en C et C++

Pour l'édition de liens avec **`gcc`**, ne pas utiliser l'option **`-nostartfiles`**

`dec` modifie le flag de zéro dans **`rflags`**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Appeler du nasm depuis C

```c
#include <stdio.h>
#include <inttypes.h>
int64_t maxof3(int64_t, int64_t, int64_t);
int main() {
    printf("%ld\n", maxof3(2, 3, 1));
    printf("%ld\n", maxof3(2, -6, 5));
    return 0;
}
```

```
    global maxof3
section .text
maxof3:
mov rax, rdi
cmp rax, rsi
jge yless
    mov rax, rsi
yless:
cmp rax, rdx
jge zless
    mov rax, rdx
zless:
ret
```

Déclarer le prototype de la fonction
On est sur que `int64_t` fait 8 octets
Utiliser le nom de la fonction comme étiquette
Respecter les conventions d'appel :
- lire les arguments dans les registres
- placer la valeur de retour dans **`rax`**

Pour l'édition de liens avec **`gcc`**, ne pas utiliser l'option **`-nostartfiles`**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Programme "pilote" en C pour appeler une fonction en nasm

```
int nasm_function(int argc, char *argv[]);
int main(int argc, char *argv[]) {
  int return_code ;
  return_code = nasm_function(argc, argv);
  return return_code ;
}
```

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Taille
des opérandes en mémoire

```
cmp edx, byte [ebx] ; mismatch in operand sizes
```

```
mov rax, dword 32000
```

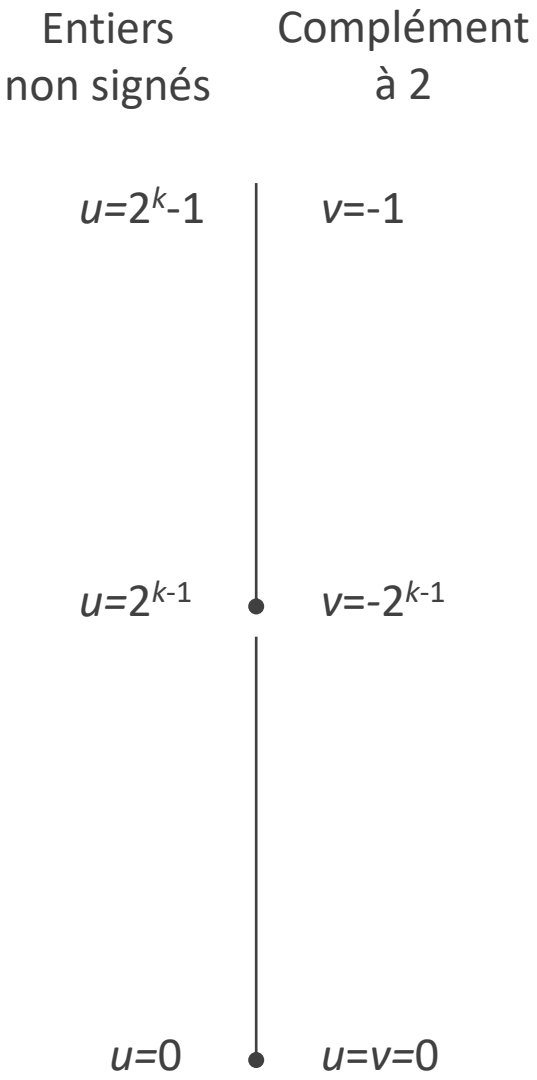
Certaines instructions exigent 2 opérandes de même taille

D'autres acceptent que le 2e opérande soit plus petit et font une extension

Deux types d'extension

- l'une est conçue plutôt pour les entiers signés
- l'autre plutôt pour les entiers non signés

Entiers non signés



La même séquence de bits peut être interprétée soit comme entier non signé, soit en complément à 2

u : l'entier non signé

v : l'entier signé

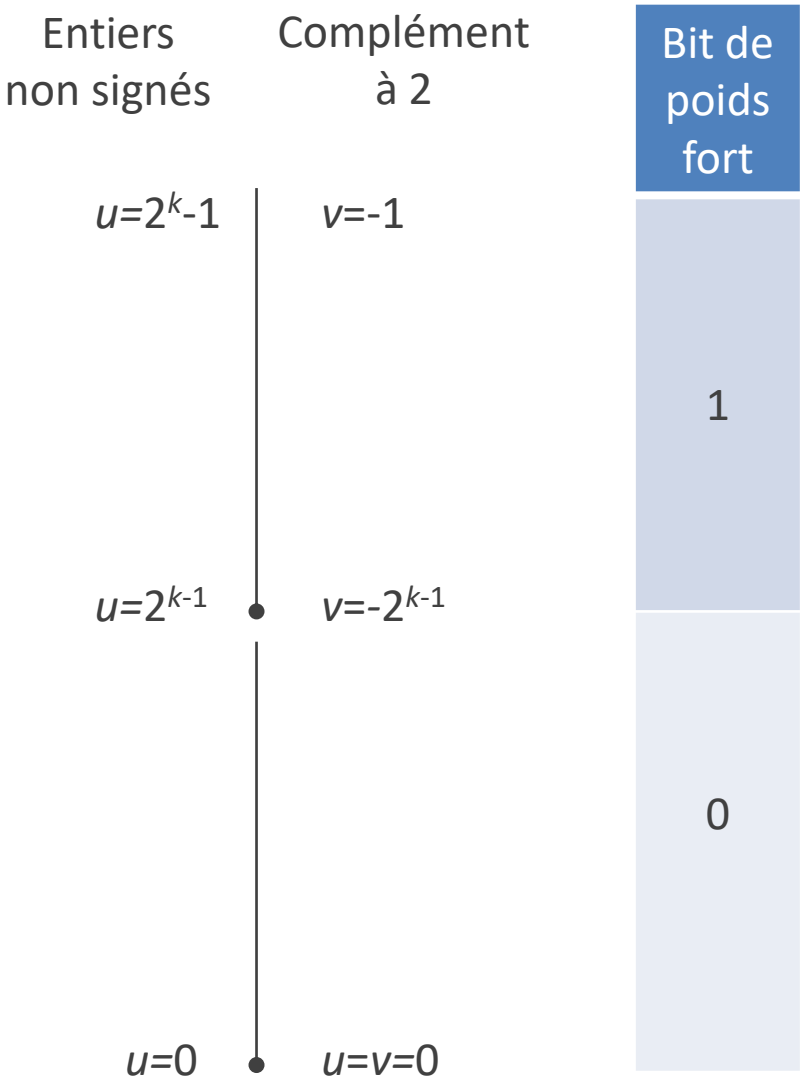
Entiers non signés

Les entiers de 2^{k-1} à 2^k-1 sont interprétés directement

Entiers signés en complément à 2

Les entiers u de 2^{k-1} à 2^k-1 sont interprétés comme $v=u-2^k$, donc entre -2^{k-1} et -1

Entiers non signés



Le bit de poids fort s'appelle aussi bit de signe à cause des entiers signés

Opérations communes aux entiers signés et non signés

Addition, soustraction

L'algorithme de `add` est le même si le développeur veut utiliser l'entier u pour représenter u ou $v=u-2^k$

Copie d'une constante dans un registre sur 8 octets

```
mov reg64, imm
```

```
mov rax, qword 1
```

L'algorithme de `mov` est le même si le développeur veut utiliser l'entier u pour représenter u ou $v=u-2^k$

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Opérations différentes pour les entiers signés et non signés Sauts conditionnels

`jg`	jump if greater
`jl`	jump if less
`ja`	jump if above
`jb`	jump if below
`jge`	jump if greater or equal
`jle`	jump if less or equal
`jae`	jump if above or equal
`jbe`	jump if below or equal

Pour les entiers signés

`jg`, `jl`, `jge`, `jle`

Négations : `jng`, `jnl`, `jnge`, `jnle`

Pour les entiers non signés

`ja`, `jb`, `jae`, `jbe`

Négations : `jna`, `jnb`, `jnae`, `jnbe`

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sauts conditionnels pour entiers signés

```
mov rax, 250   ; = 0x100 - 0x6
mov rbx, 10
mov rcx, 1
cmp al, bl     ; 0xfa, 0x0a
jg greater
   mov rcx, 0
greater:       ; rcx contient 0
```

On peut initialiser un entier comme positif et l'utiliser comme négatif

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Extension d'entiers

`mov eax, ebx`

met à **zéro** les 32 bits de poids fort de `rax`

`mov rax, dword -2`

met à **1** les 63 bits de poids fort de `rax`

Mettre un entier sur une zone mémoire plus étendue

Extension par zéro

Conçue plutôt pour les entiers non signés

Exemple : adresses relatives

Extension de signe

Conçue plutôt pour les entiers signés

Copie du bit de plus haut rang ou bit de signe

Exemple : constantes numériques

Extension d'entiers

```
movzx eax, byte [ebx]
add edx, eax
```

Extension par zéro

`movzx x,y`

Extension de signe

`movsx x,y`

```
cmp edx, byte [ebx] ; mismatch in operand sizes
```

On doit faire l'extension explicitement

On peut choisir entre **`movzx`** et **`movsx`**

```
movzx eax, byte [ebx]
cmp edx, eax
```

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Opérandes de tailles différentes

```
add rax, -1          ; même taille
add rax, byte -1
add rax, dword -1    ; error: mismatch in operand sizes
mov rax, -1          ; même taille
mov rax, byte -1     ; error: mismatch in operand sizes
mov rax, dword -1
```

Peu de combinaisons permises
Extension de signe

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Constante sur 4 octets avec registre sur 8 octets

```
add rax,          0xfd0a_74c3 ; OK
add rax,        0x3_fd0a_74c3 ; tronqué puis extension de signe
add rax, qword  0x3_fd0a_74c3 ; tronqué puis extension de signe
mov rax,          0xfd0a_74c3 ; OK
mov rax,        0x3_fd0a_74c3 ; OK
mov rax, qword  0x3_fd0a_74c3 ; OK
```

Les instructions autres que **`mov`** refusent que le 2e opérande soit une constante sur plus de 4 octets, même si le 1er opérande est sur 8 octets

Si le 1er opérande est sur 8 octets (sauf pour **`mov r64, imm64`**)
les constantes sont tronquées à 4 octets puis étendues par extension **de signe**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Nouvelles instructions

`and`	bitwise and
`or`	bitwise or
`xor`	bitwise exclusive or

`xor rax, rax` ; est plus rapide que `mov rax, 0`

`not`	bitwise negation

Mnemonic : nom d'une instruction
Code opération : par ex. REX.W + 23 /*r*

Exercice
Écrire du code pour vérifier si
les bits de rang 6 et 8 de `rflags`
sont tous les deux à 0

Application : alignement de la pile

```
my_function:
push rbp
(... conditional jumps...)
mov r14, rsp
(... how to align stack?)
call printf
mov rsp, r14
pop rbp
ret
```

Cas où on ne peut pas prévoir la taille de la pile

La taille de la pile doit être un multiple de 16 octets juste avant l'appel

Comment placer dans **`rsp`** le plus grand multiple de 16 inférieur ou égal à **`rsp`** ?

Mettre à 0 les 4 bits de poids faible de **`rsp`**

`and rsp, -16`

-16 étendu à 0ffff_ffff_ffff_fff0h

(si on écrit directement **`0ffff_ffff_ffff_fff0h`** on a un avertissement parce que la constante est tronquée à 4 octets)

Plusieurs opérations arithmétiques à la fois

`lea` load effective address

`nasm` peut faire plusieurs opérations arithmétiques en une seule instruction pour calculer une adresse :

[registre + registre * taille]

taille peut être 1, 2, 4 ou 8

[registre + nombre]

[registre + registre * taille + nombre]

`lea x,y` permet de récupérer dans **`x`** le résultat du calcul

... même si ce n'est pas une adresse !

```
lea esi, [ebx + 8*eax + 4]

lea edi, [eax + 17]
```