Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Introduction à un assembleur

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Pourquoi apprendre un assembleur ?

```
    .file   "arith.c"
    .def    ___main; .scl 2;    .type   32; .endef
    .text
    .globl  _main
    .def    _main;  .scl 2;    .type   32; .endef
_main:
LFB7:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $32, %esp
    call    ___main
    movl    $LC0, %eax
    movl    %eax, 28(%esp)
    flds    28(%esp)
    fldl    LC1
    fmulp   %st, %st(1)
    fstps   24(%esp)
    movl    $0, %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
LFE7:
    .section .rdata,"dr"
    .align 4
LC0:
    .long   1087163597
    .align 8
LC1:
    .long   1374389535
    .long   1075388088
    .ident  "GCC: (GNU) 4.9.3"
```

Écrire du code efficace

Comprendre comment fonctionne un processeur

S'amuser

Un programme en GAS

Bibliographie

https://www.felixcloutier.com/x86/index.html

Documentation des instructions

http://esauvage.developpez.com/tutoriels/asm/assembleur-intel-avec-nasm/?page=page_1

Étienne Sauvage. 2011. *Assembleur Intel avec NASM.*

http://www.pravaraengg.org.in/Download/MA/assembly_tutorial.pdf

Assembly Language Tutorial, TutorialsPoint.

http://cs.lmu.edu/~ray/notes/nasmtutorial/

Ray Toal. *NASM Tutorial*, Loyola Marymount University.

http://www.nasm.us/doc/
http://www.nasm.us/doc/nasmdocb.html

The Netwide Assembler: NASM, documentation de NASM.

http://pacman128.github.io/static/pcasm-book.pdf

Paul Carter. 2006. *PC Assembly Language* (Langage d'assemblage pour PC).

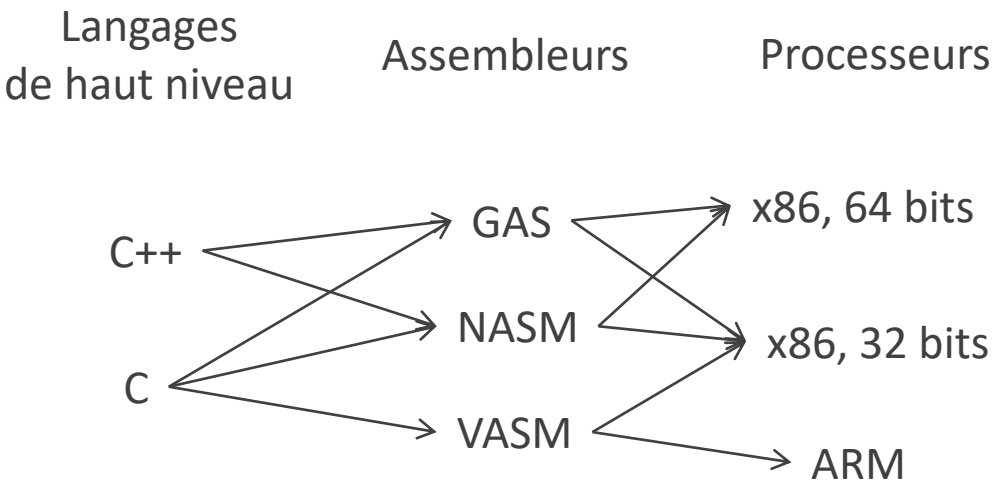
http://asmtutor.com/

Daniel Givney. 2013. *Learn Assembly Language*.

http://www.davidsalomon.name/assem.advertis/asl.pdf

David Salomon. 1993. *Assemblers and Loaders*, Ellis Horwood.

Langages et processeurs

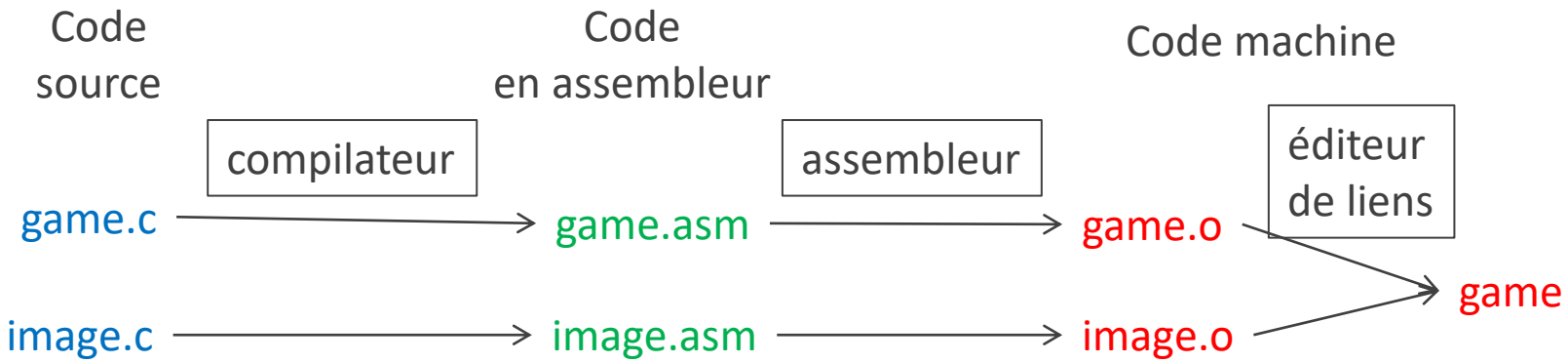


Un langage de haut niveau peut être compilé en plusieurs assembleurs

Un assembleur peut être compilé pour plusieurs processeurs

Chaque assembleur (programme) définit son langage

Chaine de traitement



Les premiers assembleurs faisaient aussi l'édition de liens

Assembleur : ou "langage d'assemblage" (*assembly language*) si on veut éviter la confusion avec l'assembleur (programme)

Netwide Assembler (NASM)

```
section .text
org 0x100
   mov ah, 0x9
   mov dx, hello
   int 0x21
   mov ax, 0x4c00
   int 0x21
section .data
hello: db 'Hello, world!', 13, 10, '$'
```

Un programme en
NASM pour Windows

Un assembleur open-source pour processeurs x86

Formats de fichiers objet

```
nasm -f elf64 my_prog.asm
```

```
global _start
section .text
_start:
  mov eax, 4 ; write
  mov ebx, 1 ; stdout
  mov ecx, msg
  mov edx, msg.len
  int 0x80 ; write
  mov eax, 1 ; exit
  mov ebx, 0
  int 0x80 ; exit(0)
section .data
msg: db "Hello, world!", 10
.len: equ $ - msg
```

Un programme en NASM pour Linux

Dépendent du système d'exploitation

`elf64`	Linux 64 bits
`elf32`	Linux, simuler un processeur 32 bits
`win64`	Windows 64 bits
`win32`	Windows 32 bits
`bin`	MS-DOS

…

Les règles du langage NASM dépendent du format visé

Exemple : la directive `org` (planche précédente) n'est pas compatible avec le format `elf64`

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Options pour l'édition de liens

```
gcc -o my_prog my_prog.o utils.o -nostartfiles -no-pie
```

Avec l'option **`-nostartfiles`**, l'exécution commence à l'étiquette **`_start`**, sinon il y a une initialisation qui appelle **`main()`**

Sans l'option **`-no-pie`** (*position-independent executable*), le programme peut être chargé à n'importe quelle adresse (fonctionnalité non opérationnelle sur les machines de l'Université)

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Commentaires

```
global _start
section .text
_start:
  mov eax, 4 ; write
  mov ebx, 1 ; stdout
  mov ecx, msg
  mov edx, msg.len
  int 0x80 ; write
  mov eax, 1 ; exit
  mov ebx, 0
  int 0x80 ; exit(0)
section .data
msg: db "Hello, world!", 10
.len: equ $ - msg
```

Depuis ";" jusqu'à la fin de la ligne

Ce n'est pas un smiley

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Registres

```
global _start
section .text
_start:
  mov eax , 4 ; write
  mov ebx , 1 ; stdout
  mov ecx , msg
  mov edx , msg.len
  int 0x80 ; write
  mov eax , 1 ; exit
  mov ebx , 0
  int 0x80 ; exit(0)
section .data
msg: db "Hello, world!", 10
.len: equ $ - msg
```

Équivalent des variables en C
mais font partie du processeur et non de la mémoire centrale

Université Gustave Eiffel

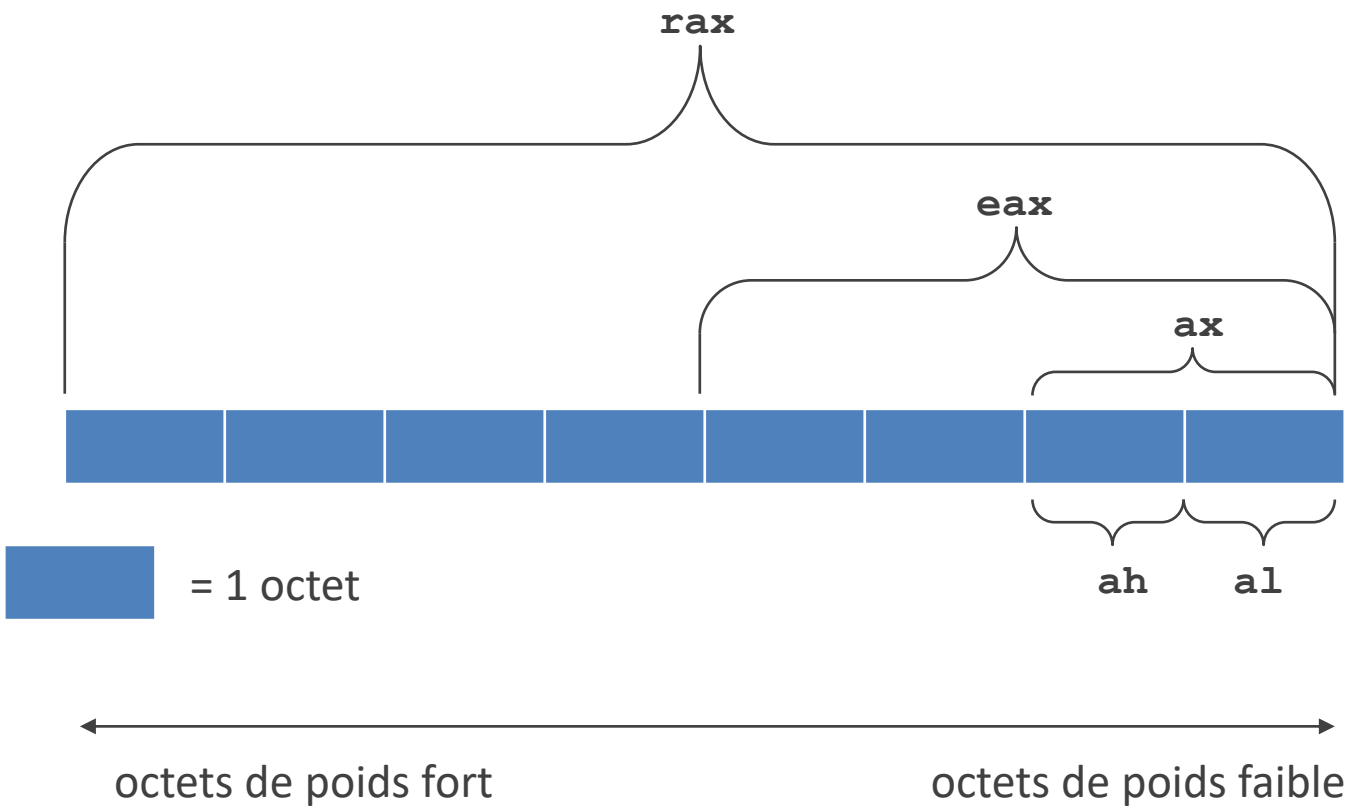
INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Taille des principaux registres

8 octets	4 octets	2 octets	1 octet	1 octet
rax	eax	ax	ah	al
rbx	ebx	bx	bh	bl
rcx	ecx	cx	ch	cl
rdx	edx	dx	dh	dl
r12	r12d	r12w		r12b
r13	r13d	r13w		r13b
r14	r14d	r14w		r14b
r15	r15d	r15w		r15b
rsi	esi	si		sil
rdi	edi	di		dil

`r` : register
`e` : extended
`h` : high
`l` : low
`d` : double word
`w` : word
`b` : byte
`si` : source index
`di` : destination index

Chevauchement (*overlapping*) entre registres

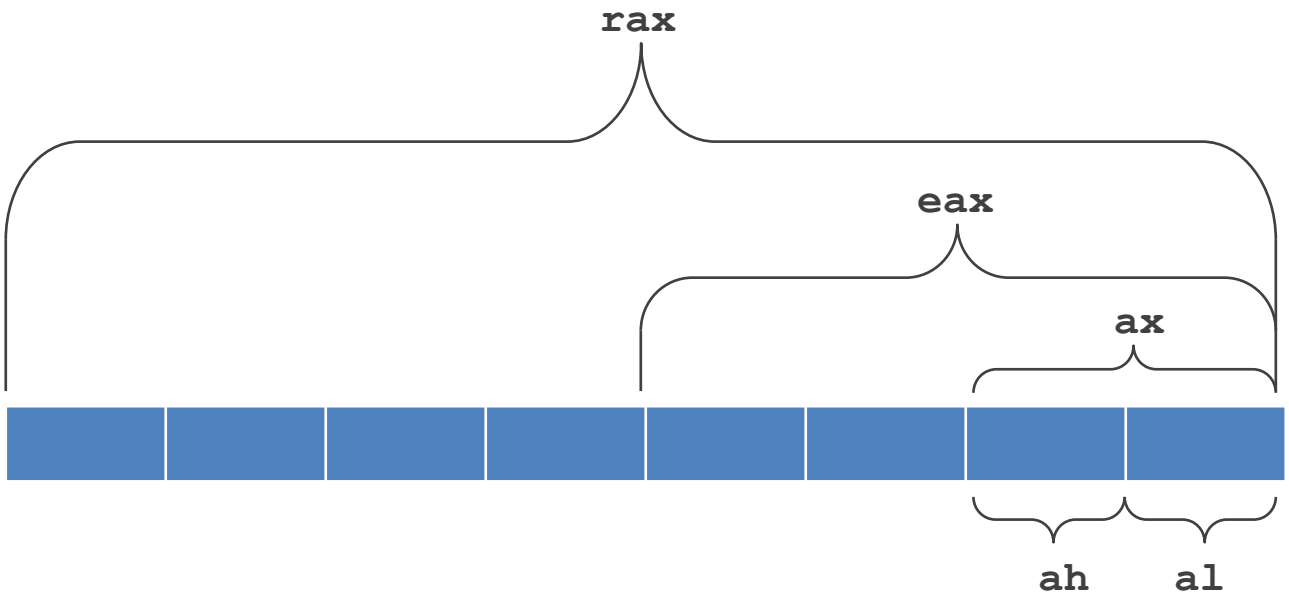


Exercice : étant donné la valeur d'un registre, déduire celle d'un sous-registre

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Chevauchement (*overlapping*)



Si on modifie **`eax`**, ça modifie **`rax`**

Si on modifie **`rax`**, ça peut modifier **`eax`**

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Quelques instructions de base

Nom symbolique (*mnemonics*) de l'instruction			
`mov`	move	`mov x, y`	copier `y` dans `x` (écrase `x`)
`add`	add	`add x, y`	`x` := `x`+`y`
`sub`	subtract		
`inc`	increment		
`dec`	decrement		
`call`	function call	`call x`	appeler une fonction `x`
`syscall`	system call	`syscall`	appel système

Opérandes de `mov`

Exercice : changer de base

Convertir de l'hexadécimal en binaire

0x11 0xba 0xcf3b 0x4e43 0x1234 0xff 0xf34e

Registres

`mov rax, rbx` copier `rbx` dans `rax`

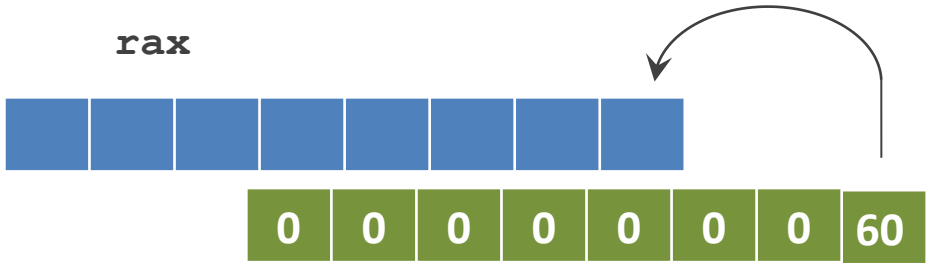
Constantes ("données immédiates")

`mov rax, 60`	copier 60 dans `rax`
`60`	décimal
`3ch`	hexadécimal
`0x3c`	hexadécimal
`0h3c`	hexadécimal
`00111100b`	binaire
`0b0011_1100`	binaire

Une constante en hexadécimal doit commencer par un chiffre

`aah`	identificateur
`0aah`	constante numérique

Taille des opérandes




```
mov rax, 60

mov rax, rbx
mov eax, ebx
mov ax, bx
mov al, bh
```

L'instruction `mov` écrase **tout l'opérande destination**, même si la constante tient sur une taille mémoire plus petite

Dans un premier temps, faire seulement des instructions avec des opérandes **de même taille**

La taille de la constante est fixée par la taille du registre

Opérations sur registres de 4 octets

```
mov eax, ebx ; écrase tout rax
mov eax, 60  ; écrase tout rax

mov ax, bx ; écrase 2 octets
mov al, bh ; écrase 1 octet
```

Si le premier opérande est un registre sur **4 octets**,
- `mov` met à zéro les 4 octets de poids fort

Compatibilité descendante avec le mode 32 bits

Pas de mise à zéro pour les opérations sur les registres de 1 ou 2 octets

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Instructions arithmétiques

`add x, y`	`x` := `x`+`y`
`sub x, y`	`x` := `x`-`y`

Utiliser des opérandes de même taille

`inc x`	`x` := `x`+1
`dec x`	`x` := `x`-1

Appels système avec `syscall`

Exercice

Écrire du code pour terminer le programme et renvoyer 0

L'instruction **`syscall`** peut servir pour plusieurs appels système différents

La valeur dans **`rax`** détermine quel appel système est réalisé

La valeur dans **`rdi`** sert d'opérande à l'appel

`rax`	**appel réalisé**	**`rdi`**	**`rsi`**	**`rdx`**
60	exit	valeur de retour		
0	read	fichier	adr. destination	taille
1	write	fichier	adr. source	taille

Si on ne met pas d'instruction de terminaison, le système passera à l'instruction suivante, même si c'est la dernière dans le fichier .asm

Appels système avec `int 80h`

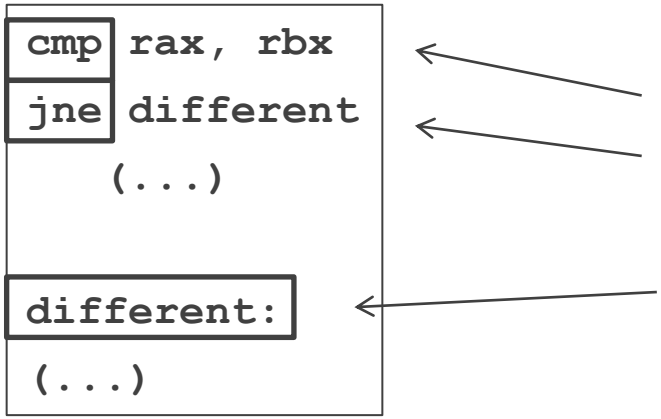
L'instruction **`int`** permet de faire des interruptions
`int 80h` fait un appel système
La valeur dans **`eax`** détermine quel appel système est réalisé
Les bits de poids fort de **`rax`** sont ignorés
La valeur dans **`ebx`** est un opérande

`eax`	**appel réalisé**	**`ebx`**	**`ecx`**	**`edx`**
1	exit	valeur de retour		
3	read	fichier	adr. destination	taille
4	write	fichier	adr. source	taille

Exercice
Écrire du code pour terminer le programme et renvoyer 0

Instructions de branchement

Exercice
Écrire du code pour vérifier si les 4 octets de poids fort de **`rax`** sont à zéro sans écraser **`rax`**



Un branchement typique a 3 étapes :

- une instruction qui affecte le registre **`rflags`**
- une instruction de saut conditionnel qui accède à **`rflags`**
- une étiquette cible

Quelques instructions de saut conditionnel

`je x`	equal
`jne x`	not equal
`jg x`	(strictly) greater
`jng x`	not greater (less than or equal)

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Le registre `rflags`

8 octets	4 octets	2 octets
`rflags`	`eflags`	`flags`

Certaines instructions modifient `rflags` :

- résultat nul
- retenue non résolue
- résultat positif

et autres situations qui peuvent servir de conditions pour des branchements

Étiquettes (*labels*)

```
global _start
section .text
_start:
   mov eax, 4 ; write
   mov ebx, 1 ; stdout
   mov ecx, msg
   mov edx, msg.len
   int 0x80 ; write
   mov eax, 1 ; exit
   mov ebx, 0
   int 0x80 ; exit(0)
section .data
msg: db "Hello, world!", 10
.len: equ $ - msg
```

Forme générale

<étiquette> ":"

Le ":" n'est pas obligatoire

Déclarations

global <étiquette>	exporter l'étiquette
extern <étiquette>	importer l'étiquette

Étiquettes locales

```
global _start
section .text
_start:
  mov eax, 4 ; write
  mov ebx, 1 ; stdout
  mov ecx, msg
  mov edx, msg.len
  int 0x80 ; write
  mov eax, 1 ; exit
  mov ebx, 0
  int 0x80 ; exit(0)
section .data
msg: db "Hello, world!", 10
.len: equ $ - msg
```

- Rattachées à la dernière étiquette non locale
- Commencent par un "."
- On peut y faire référence en combinant les deux étiquettes

Branchement inconditionnel

Exercice
Ajouter un compteur

```
loop:
cmp rax, rbx
je same
   inc rax
   jmp loop
same:
(...)
```

`jmp x` aller à l'étiquette x

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Segments

```
global _start
section .text
_start:
  mov eax, 4 ; write
  mov ebx, 1 ; stdout
  mov ecx, msg
  mov edx, msg.len
  int 0x80 ; write
  mov eax, 1 ; exit
  mov ebx, 0
  int 0x80 ; exit(0)
section .data
msg: db "Hello, world!", 10
.len: equ $ - msg
```

Mémoire

= Mémoire centrale et non registres

Directive `segment` ou `section`

Segment `.text`

Les instructions du programme

Segment `.data`

Réserver de la mémoire et initialiser les valeurs

Réservation de données en mémoire

```
section .data
length: dw 512
(...)
section .text
mov rsi, length
```

Forme générale d'une réservation

<étiquette> ":" <code> <valeur initiale>

<étiquette> ":" <code> <valeur1>, <valeur2>

<étiquette> ":" <code> <val1>, <val2>, <val3>

Le ":" n'est pas obligatoire

Le code fixe la quantité de mémoire réservée (ici 2 octets)

L'étiquette devient une **constante qui vaut l'adresse** de la mémoire réservée (mais ne donne pas la taille réservée)

Codes pour déclarer de la mémoire

`db`	define byte	1 octet
`dw`	define word	2 octets (vestige de la version 16 bits)
`dd`	define double word	4 octets
`dq`	define quadruple word	8 octets
`dt`	define ten bytes	10 octets

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Initialisation avec des caractères

```
letter: db "y"
letters: db "y","n","?"
message1: db "file"
message2: dd "file"
```

On peut remplacer les guillemets (") par des apostrophes (')

```
symbols: db ".",":","'",'"'
```

Accès à la mémoire

```
section .data
my_data: dw 512
(...)
section .text
mov esi, dword [ my_data ]
```

Pour identifier un espace mémoire il faut 2 données :

- son adresse
- sa taille

Spécificateur de taille

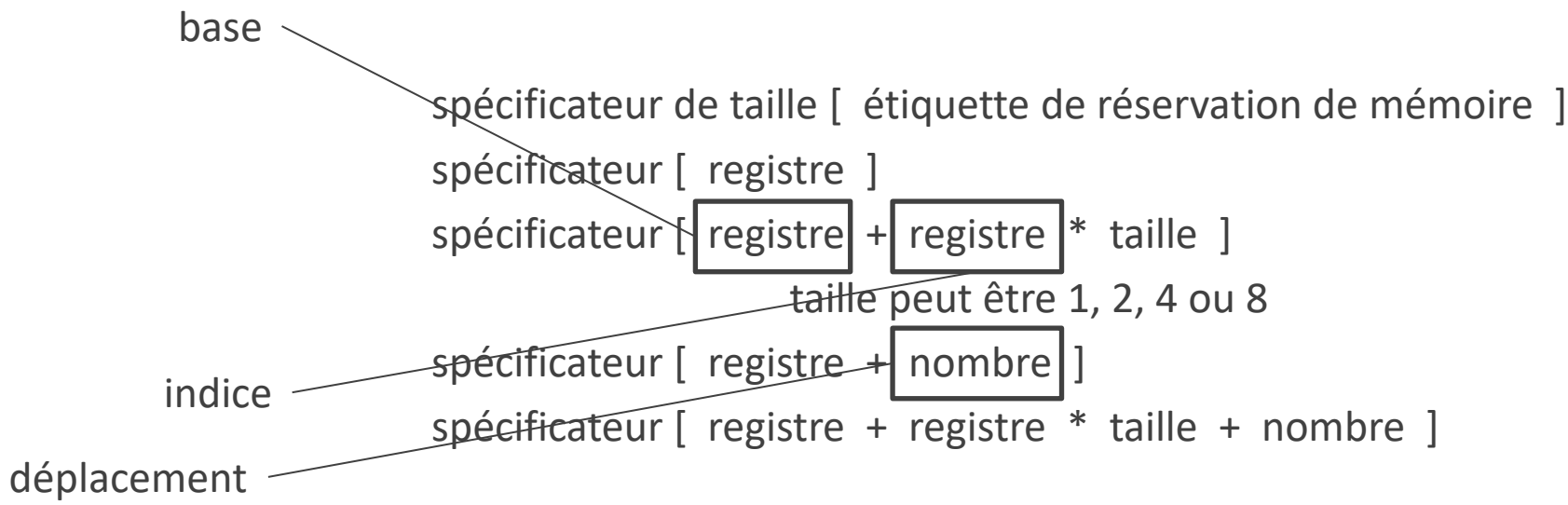
Spécificateurs de taille

```
section .data
my_data: dw 512
(...)
section .text
mov esi, dword [my_data]
```

Réserver dans .data	Spécificateur de taille	Nombre d'octets
db	byte	1
dw	word	2
dd	dword	4
dq	qword	8
dt	tword	10

Syntaxe pour l'accès à la mémoire

```
mov esi, dword [my_data]
```



Ce sont les seuls cas où une instruction nasm contient une expression arithmétique complexe

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Au plus un opérande en mémoire

Avec la plupart des instructions, on a le droit à au plus 1 opérande en mémoire

L'autre doit être un registre ou une constante

```
mov rdi, qword [rsi]    ; OK
mov byte [rdi], 60      ; OK
mov byte [rdi], byte [rsi]     ; non
```

Exercice : comment faire ?

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Taille des opérandes en mémoire

Il suffit qu'un des deux opérandes donne sa taille

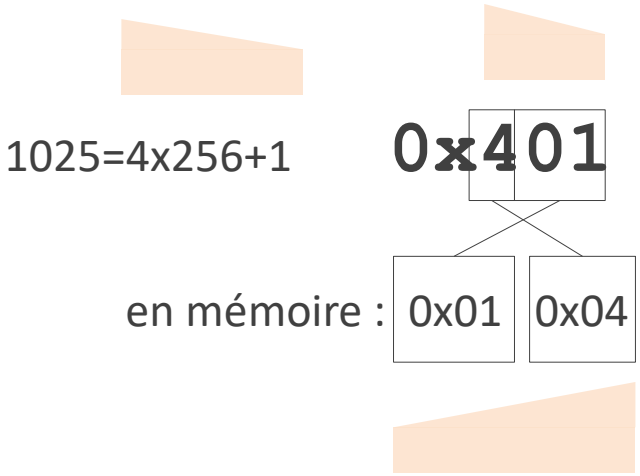
```
mov rsi, qword [length]   ; copie 8 octets
mov rsi,       [length]   ; 8 octets (taille de rsi)
mov qword [rax], rsi      ; 8 octets
mov       [rax], rsi      ; 8 octets (taille de rsi)
mov word [length],      12  ; 2 octets
mov      [length], word 12  ; 2 octets
mov      [length],      12  ; erreur (quelle taille ?)
mov      [rsi],         12  ; erreur (quelle taille ?)
```

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Orientation little-endian



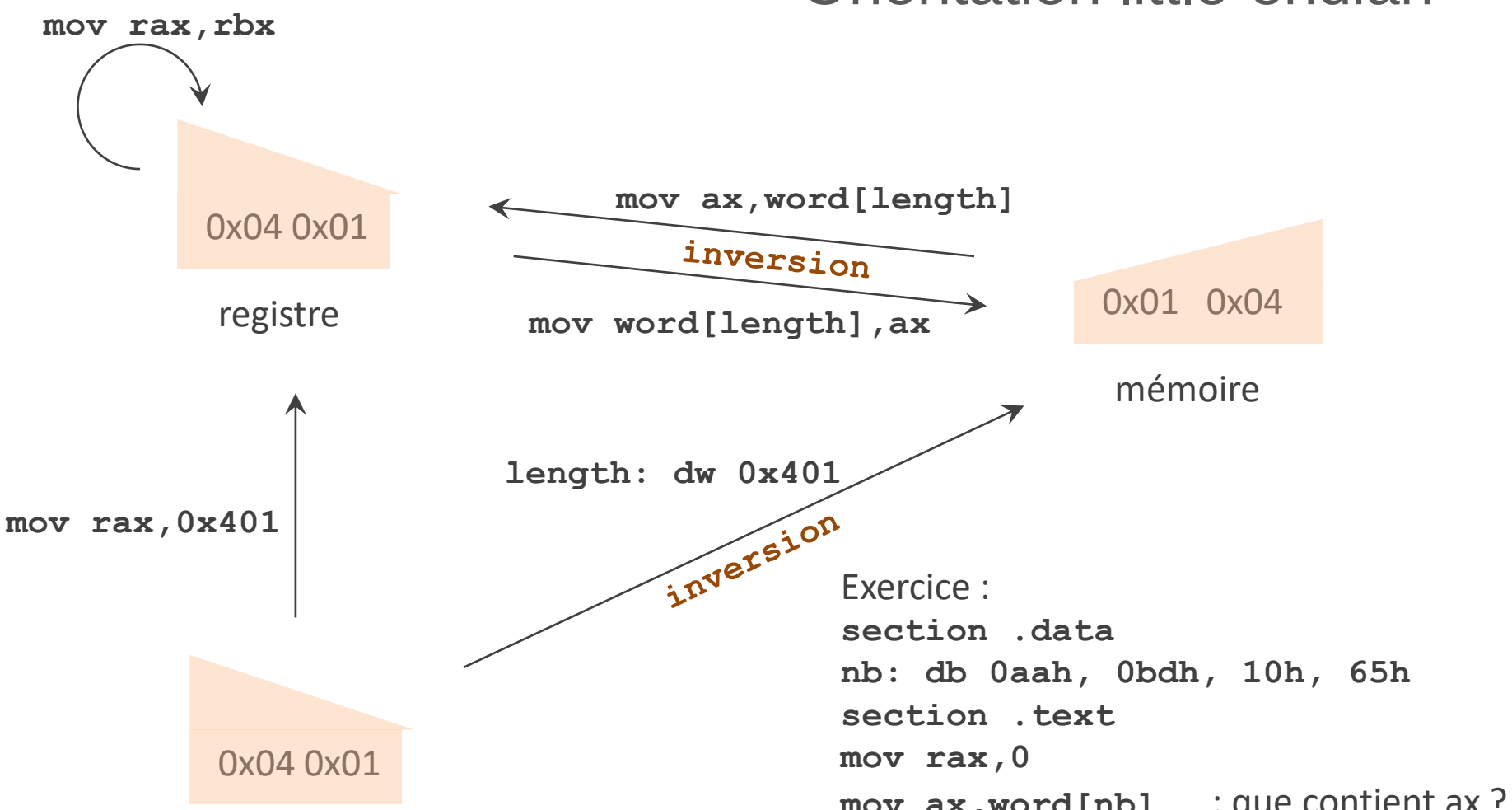
Pour les processeurs x86, la **mémoire** est orientée little-endian

Registres

Pas d'orientation : pas d'adresses basses ni hautes à l'intérieur d'un registre

Pour raisonner, on les oriente big-endian comme les entiers, par habitude

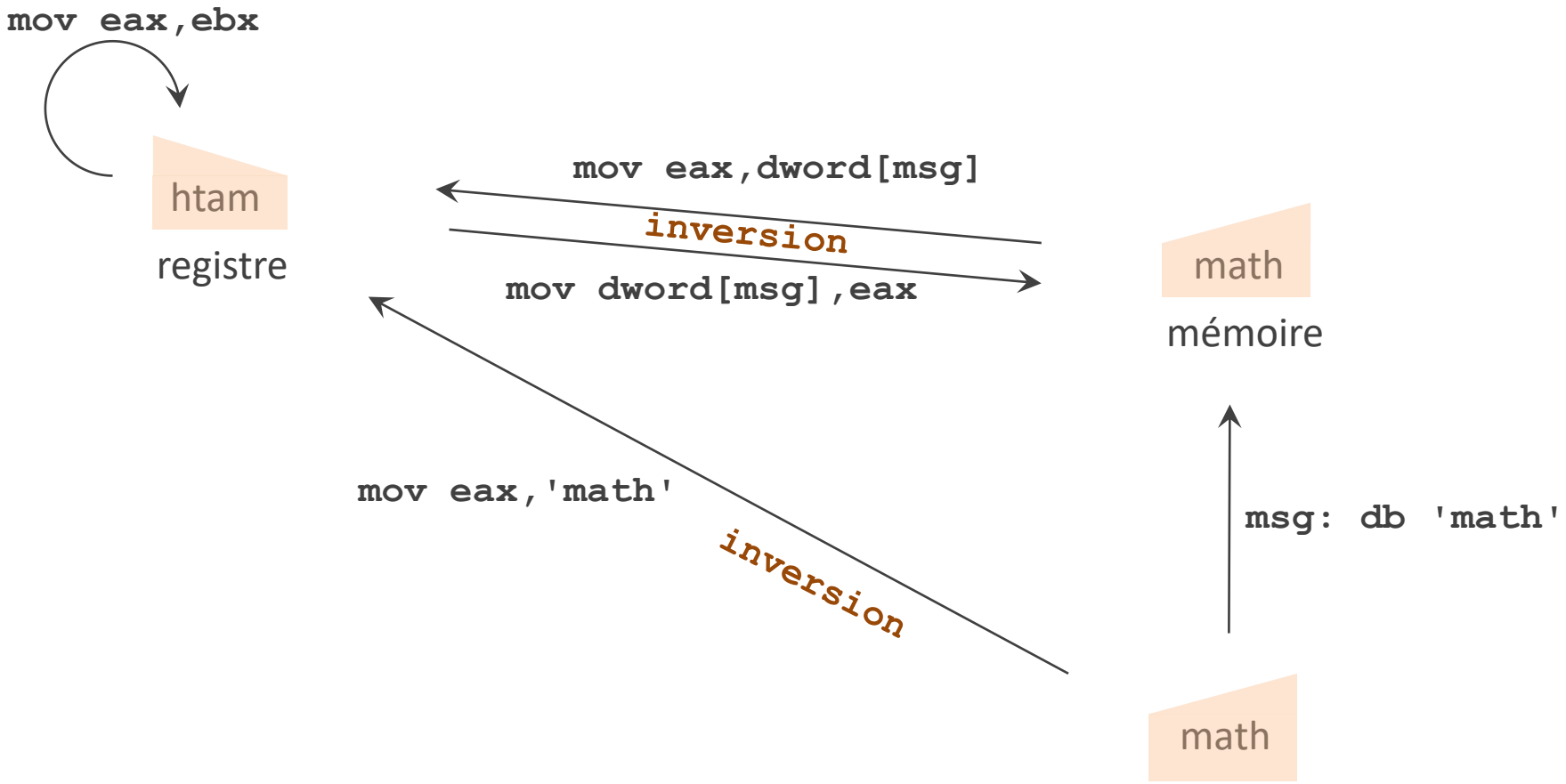
Orientation little-endian



Exercice :

```
section .data
nb: db 0aah, 0bdh, 10h, 65h
section .text
mov rax,0
mov ax,word[nb]     ; que contient ax ?
mov ax,word[nb+1]   ; que contient ax ?
```

Orientation little-endian



Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Orientation little-endian

Exercice :

```
length: db 0x1, 0x4
```

est-il équivalent à

```
length: dw 0x401
```

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Arithmétique binaire

On utilise l'hexadécimal pour représenter de façon simplifiée le binaire

Valeurs		Taille
0 à 255	0h à 0ffh	1 octet
0 à 65 535	0h à 0ffffh	2 octets
0 à 2^{8n}-1	0 à 256^{n}-1	*n* octets

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Arithmétique binaire
Exercices

Convertir du binaire en décimal

10001001 11111101 11111 10000011

Convertir du décimal en binaire

13 347 1023 752 12 21

Convertir de l'hexadécimal en décimal

0ff 0fe 0f3e 0a2b 221 34e 8000 44f

Additionner en hexadécimal

0ffe+0fef 0fa+0a2 0a3b4+0f654 8987+0a567 1+12