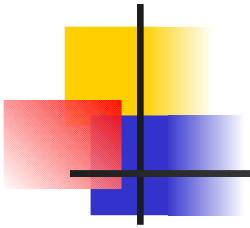


Adressage, directives assembleur

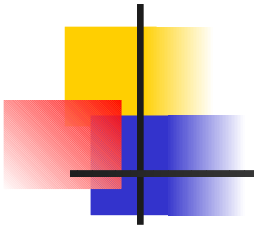
Jacques Chassin de Kergommeaux

(d’après X. Rousset)

Langage d’assemblage

- Rôle d’un langage d’assemblage
  - Fournir une représentation symbolique des instructions et des données
  - Proche du langage d’une machine
  - Permet d’utiliser des étiquettes symboliques pour désigner des emplacements de mémoire

Défauts d'un langage d'assemblage



- Les programmes sont spécifiques à une machine donnée (non portables).
- Les programmeurs sont moins productifs, font plus d'erreurs.
- Les programmes sont moins lisibles.
- Difficile de programmer plus efficacement qu'en langage évolué en raison de :
  - L'efficacité des compilateurs.
  - La complexité des processeurs (effets de cache, de pipe-line, etc.)

Exemples

- **Instructions:**

```
iter:   cmpw $0,%ax       # compare word
        je      fin       # jump if equal
        shrw    $1,%ax    # shift right word
        jnc     suite     # jump if no carry
        add     %dx,%ax   # add
suite:  shlw    $1,%dx    # shift left word
        jmp     iter      # jump inconditionnel
fin:
```

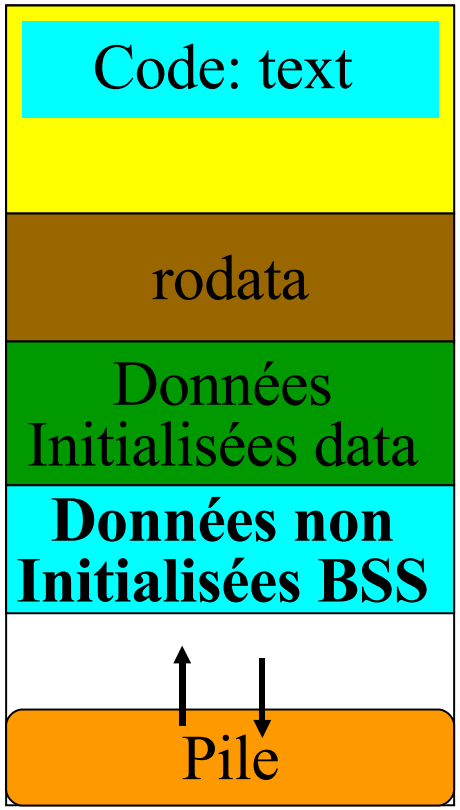
- **Données :**

```
        toto:       .byte 0xff
        lulu:       .int  $5000, suite
```

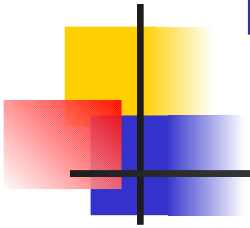
Modèle mémoire (assembleur Gnu)

- Les directives .text, .data, .section

```
            .section .data
un:         .int 1
….          …
            .section .bss
            .lcomm tab,10
tab1:       skip 10
---
            .text
main:       pushl %ebp
```

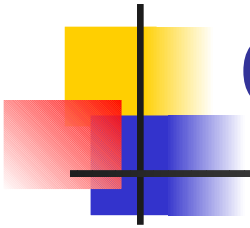


Programme source



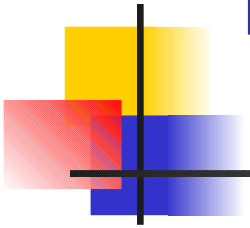
- Ensemble de sections
  .data (.rodata, .bss) pour les données
  .text pour les instructions
- Chaque section est une suite de lignes
- Pour les instructions
  [etiquette:] code_op opérandes
  Pour les données
  [etiquette:] def_de_donnée suite de valeurs
- des commentaires
- des directives d’assemblage

Représentation symbolique des instructions



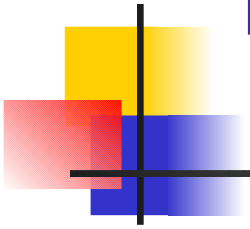
- Code de l'opération
  - La dernière lettre correspond à la longueur des opérandes
  - Exemple : shr, subl, movb
- Représentation symbolique des opérandes
  - Registre. Ex: %eax
  - Adresse en mémoire, dénotée par un mode d'adressage Ex: 4(%ecx)
  - Valeur immédiate Ex: $0x45ab
- Attention : les types d'opérandes valides dépendent des instructions

Les commentaires



- Définition : il s'agit de textes non interprétés par l'assembleur et qui sont fournis par le programmeur pour augmenter la lisibilité de son programme.
- Deux possibilités
  - soit sur une ligne tout ce qui suit le caractère # jusqu'à la fin de ligne
  - soit comme en C ce qui est entre les deux couples de caractères /* et */

Les étiquettes



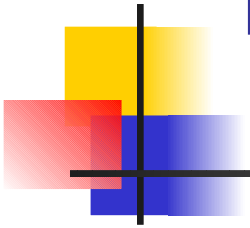
- Une étiquette (identificateur suivi de « : ») sert à désigner l'adresse d'un emplacement de mémoire
- On peut l'utiliser dans un champ opérande

```
toto:    movw %eax,lulu
```

Directives d’assemblage

- Directives d'assemblage : commandes fournies par l’assembleurs qui ne correspondent à aucune instruction du processeur.
- Elles permettent entre autres :
  - La définition de données.
  - La définition de constantes ou de symboles.
  - Les sections (.text et .data)

Définition de données



- Définition de données initialisées
  - [étiquette] .<DNAME> val1,val2,..., valn

    <DNAME> = byte | hword | int | quad | string

    xi: .int 0xaabbccdd, xi, -4500

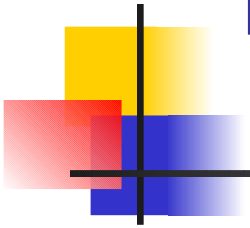
    xb: .byte 0x3f, 35, ‘c’

    message: .string "Hello World"
- Définition de données « non initialisées »
  - .lcomm nom, taille
  - [étiquette] .skip taille (,valeur)

Directives assembleur de définition de données

```
1                                       .section .data
2 0000 DDCCBBAA  xi:                    .int 0xaabbccdd, xi, -4500
2      00000000
2      6CEEFFFF
3 000c 3F2363          xb:              .byte 0x3f, 35, 'c'
4 000f  48656C6C       message:         .string "Hello World"
4      6F20576F
4      726C6400
5                                       .lcomm tab,10
6                                       .text
7                                       .global main
8 0000 55              main:            pushl %ebp
```

Définition de constantes



- Ressemblent au #define du langage C
  - .equ symbole, expression

    associe de façon définitive la valeur d'*expression* au symbole défini par le champ *symbole*.

    .equ x, 1024

    movl $x, %ax

    . tableau: .skip 400

    .equ last_elem, .- 4 adresse du dernier élément de tableau
  - .set symbole, expression

    associe de façon temporaire la valeur d'*expression* au symbole défini par le champ *symbole*.

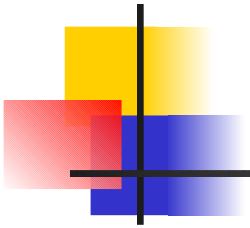
    .set n, 1

    .set n, n+1

    ATTENTION : une constante n'est **PAS** une variable !

Exportation de symbole

- Motivation : pouvoir définir (resp. utiliser) dans un module d'assemblage du code et des données utilisables (resp. définis) dans un autre module d'assemblage produit par un compilateur ou par un programmeur.
- Directive .global
  - .global <étiquette>,... les étiquettes du champ opérande définies dans le module courant sont rendues visibles à l'extérieur de ce module.
  - Toute étiquette référencée dans le module courant sans y être définie est considérée comme externe, donc définie dans un autre module d'assemblage.
  - Exemple: **.global main** # chaque programme comporte un
    # « main » appelé par le système



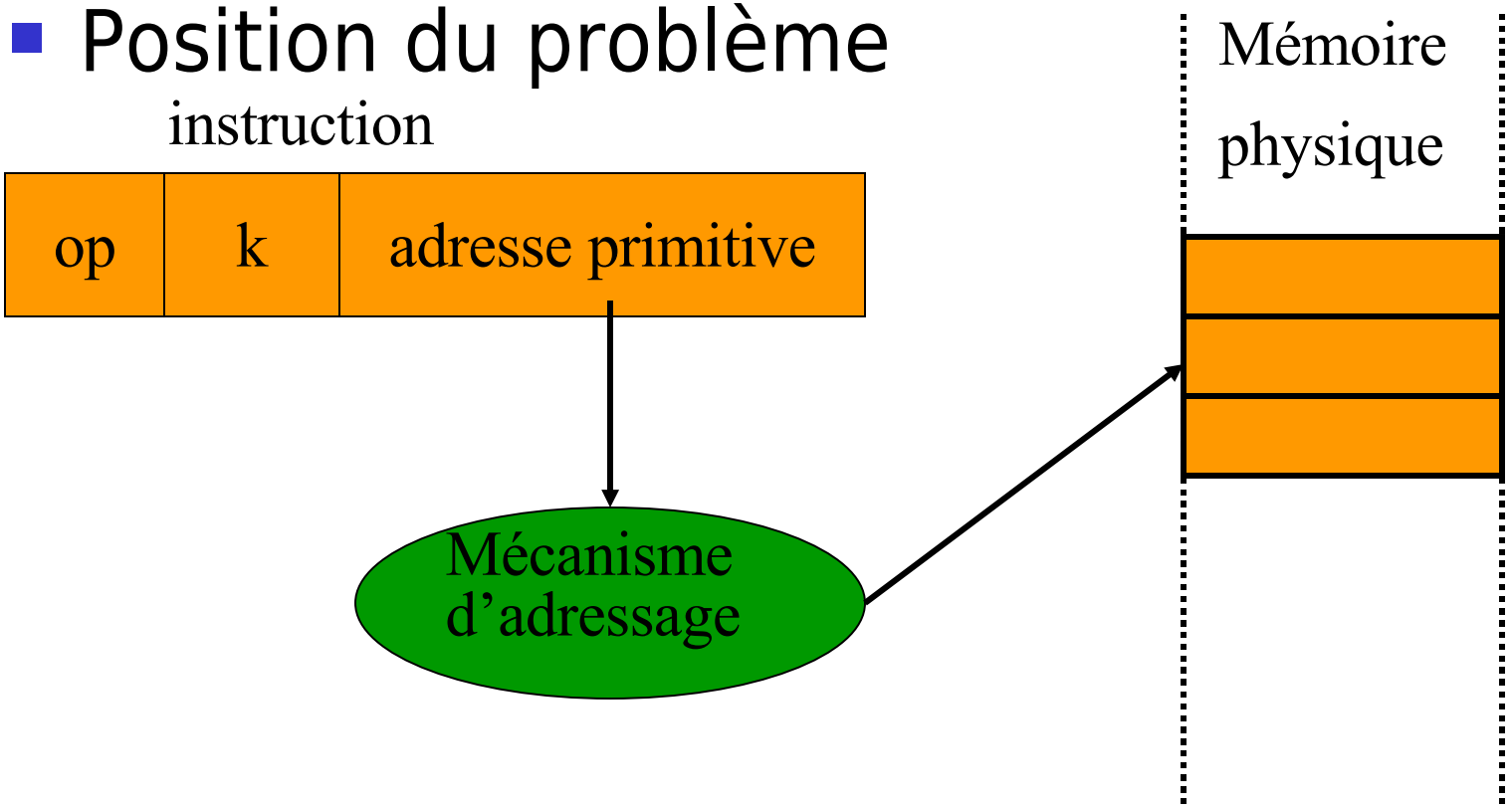
Mécanismes d’adressage

Introduction (1)

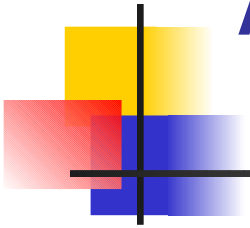
- Pourquoi des mécanismes d'interprétation d'adresses et pas directement les adresses ?
  - Réduire la taille des champs d'adresse dans les instructions
  - Faciliter l'accès à des données structurées (tableaux ou structures)
  - Faciliter le partage d'une même machine entre plusieurs utilisateurs.
  - On peut, cependant, avoir directement l'adresse !

Introduction (2)

- Position du problème

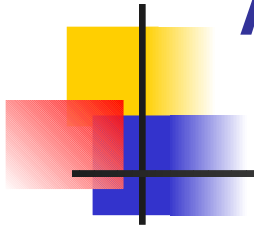


Adressage registre direct



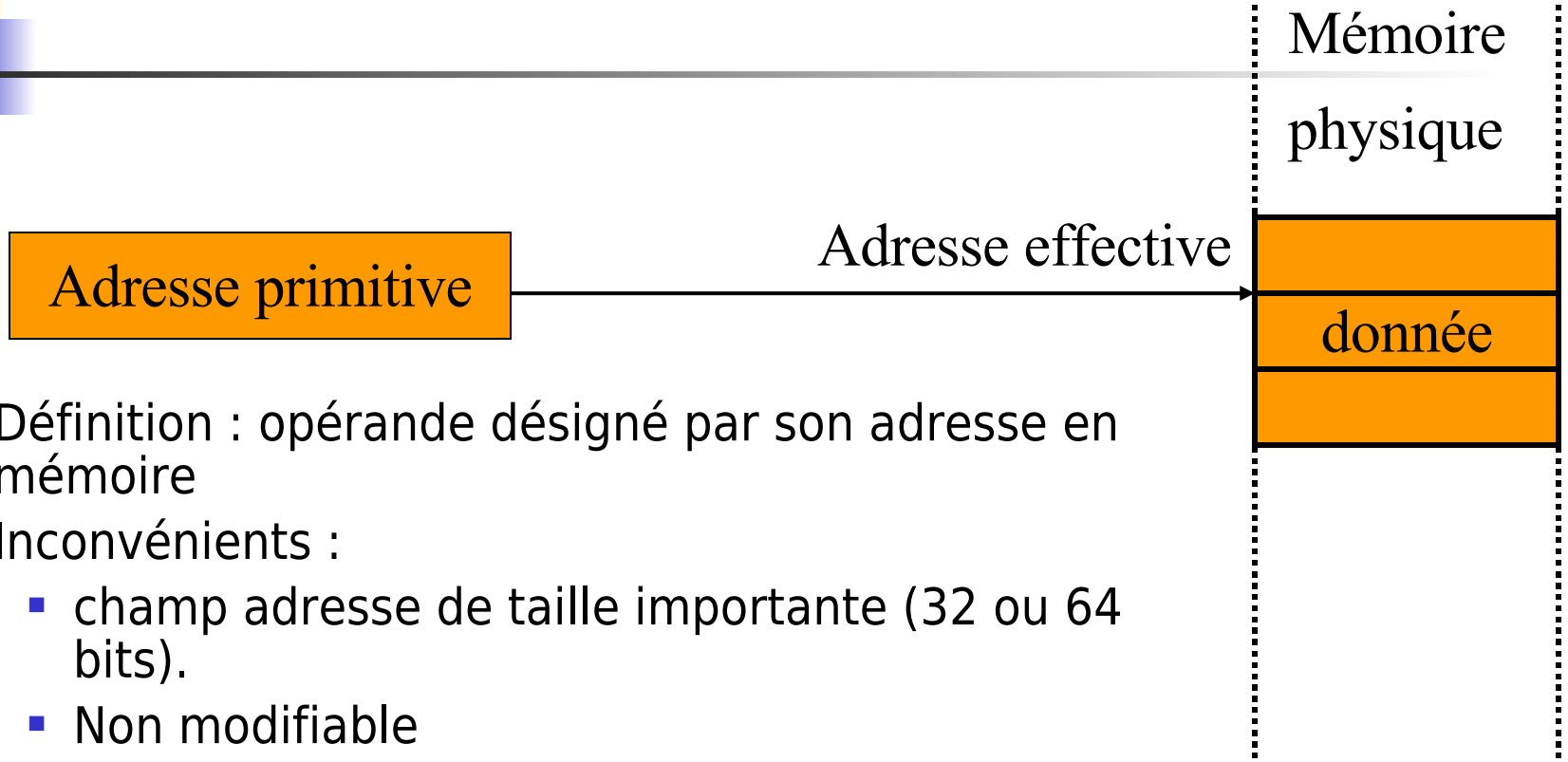
- L’opération peut porter sur 8, 16 ou 32 bits.
- Les opérandes doivent avoir la même taille que celle spécifiée dans l ’instruction :
  - Exemple : syntaxe assembleur GNU (OP source destination) :
    - movl %esp, %ebp
    - movw %ax, %bx
    - movb %al, %ah

Adressage immédiat



- Définition : le champ adresse est utilisé pour représenter la valeur de l'opérande.
  - Avantage
    - Pas de gaspillage d'emplacement mémoire
    - Economie d'un accès à la mémoire (accès immédiat à la valeur)
  - Inconvénient
    - L'opérande représente forcément une constante
    - La taille des constantes représentables est limitée
- Exemples : movb $0xff, %al
  movl $toto, %eax # $toto: valeur du symbole toto,
  # const. ou ad. donnée ou code
- ATTENTION au $!

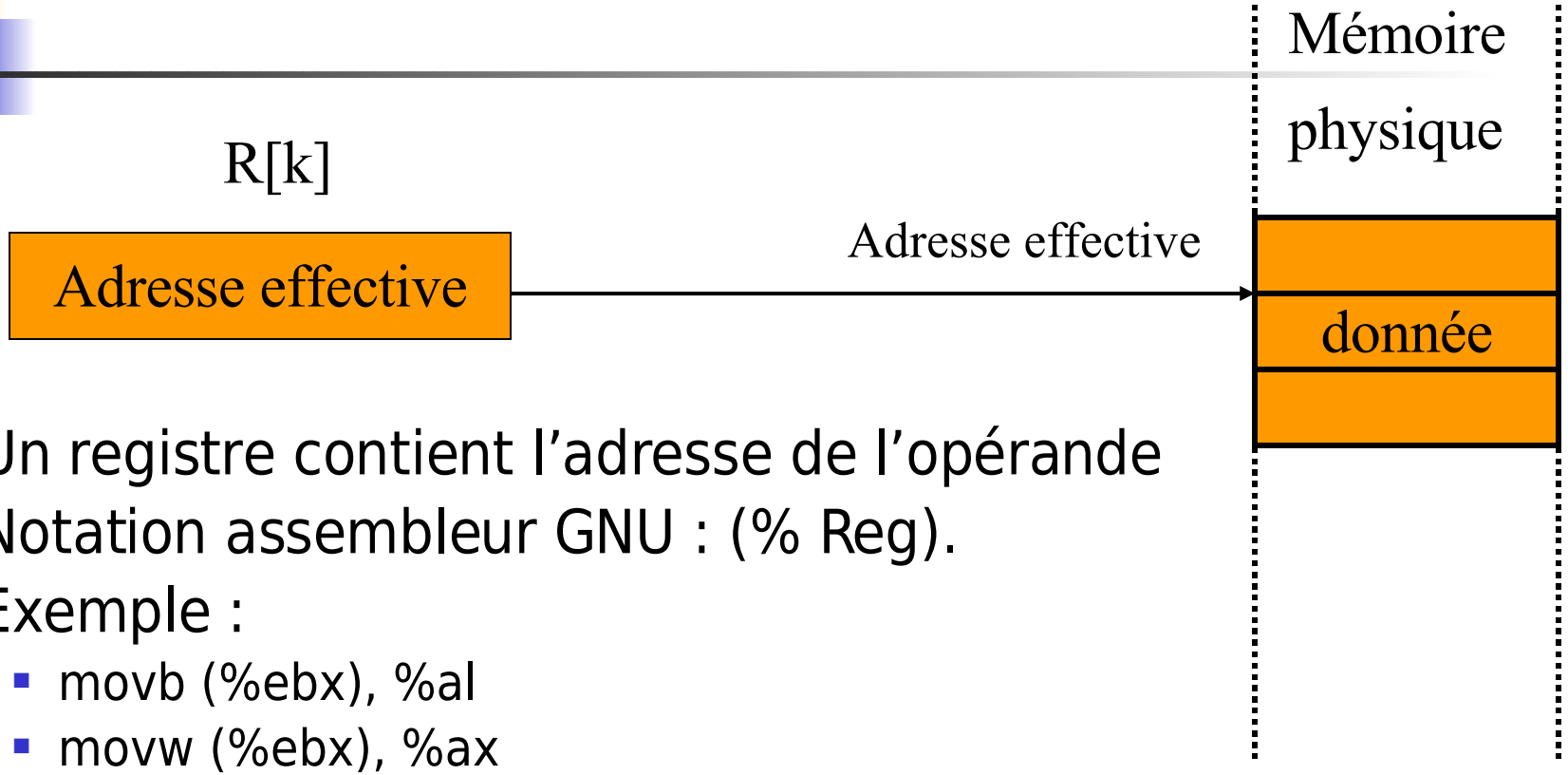
Adressage direct



- Définition : opérande désigné par son adresse en mémoire
- Inconvénients :
  - champ adresse de taille importante (32 ou 64 bits).
  - Non modifiable
- Exemples :

```
movb 0x804943C, %al
movl toto, %eax
```

Adressage Indirect Registre



- Un registre contient l'adresse de l'opérande
- Notation assembleur GNU : (% Reg).
- Exemple :
  - movb (%ebx), %al
  - movw (%ebx), %ax
  - movl (%ebx), %eax
- Attention : (%ebx) désigne un **emplacement mémoire !**

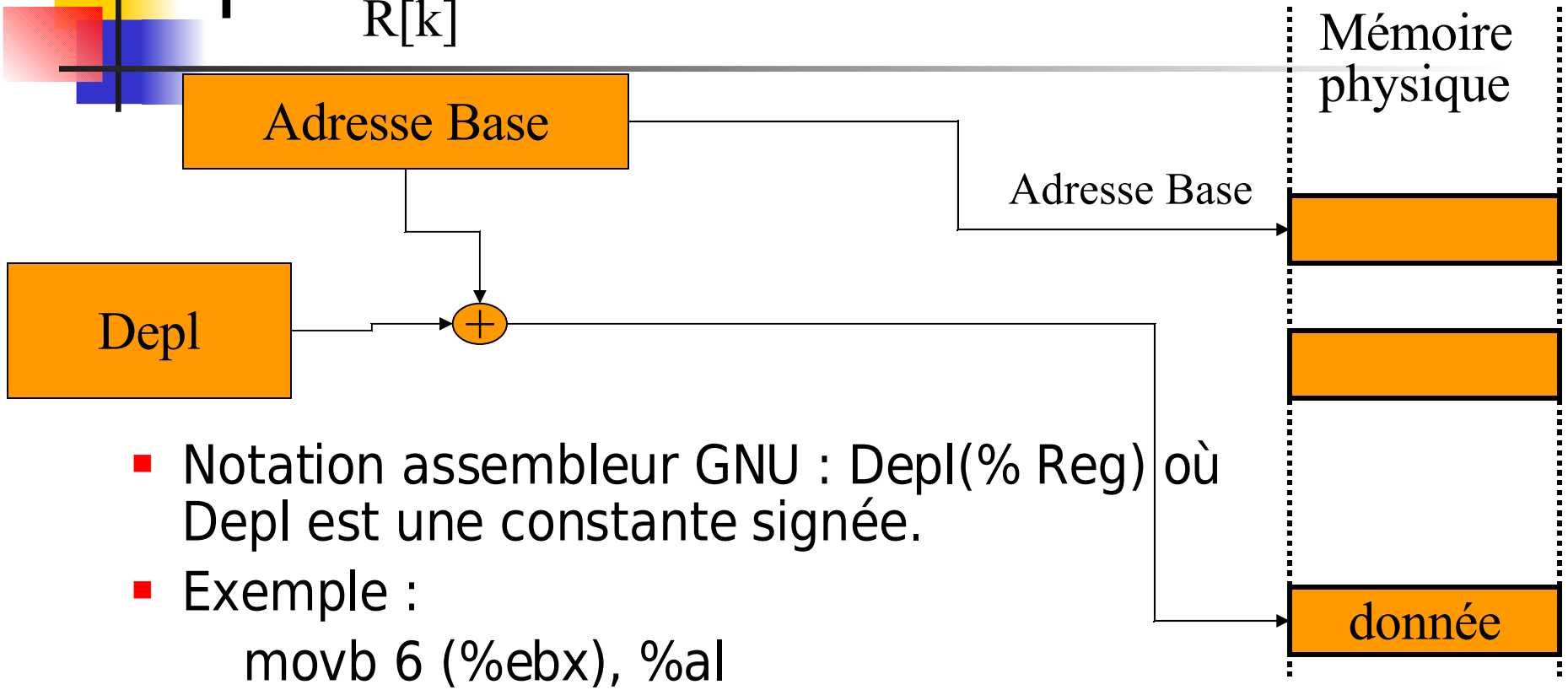
Exemples

```
 4                                  .equ cte,3
 6                                  .data
 7 0000 05000000                    i:   .int 5
 8 0004 03000000                    j:   .int 3
 9 0008 02000000                    k:  .int 2
17 0003 B803000000                  movl $cte, %eax
18 0008 A108000000                  movl k, %eax
19 000d 89C1                        movl %eax, %ecx
20 000f  B808000000                 movl $k, %eax
21 0014 8B08                        movl (%eax), %ecx
```

Exemples (suite)

```
(gdb) x /3w &i
0x804952c <i>:  0x00000005     0x00000003     0x00000002
(gdb) x /12b &i
0x804952c <i>:  0x05    0x00    0x00    0x00    0x03   0x00   0x00    0x00
0x8049534 <k>:  0x02    0x00    0x00    0x00
(gdb) x /50xb main
0x8048334 <main>:    0x55  0x89   0xe5  0xb8   0x03  0x00  0x00  0x00
0x804833c <main+8>: 0xa1  0x34   0x95  0x04  0x08   0x89  0xc1  0xb8
0x8048344 <main+16>:0x34 0x95   0x04  0x08  0x8b   0x08  0xc9  0xc3
```

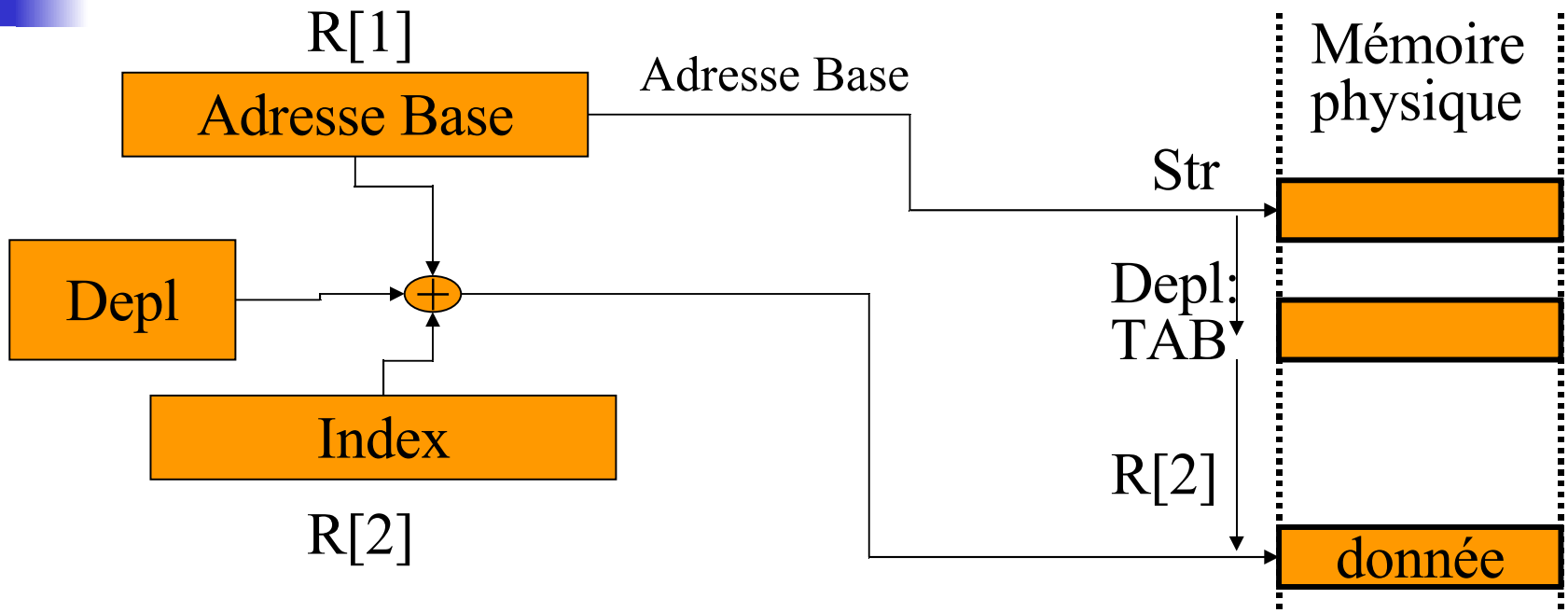
Indirect avec Base Déplacement



- Notation assembleur GNU : Depl(% Reg) où Depl est une constante signée.
- Exemple :

  movb 6 (%ebx), %al
- Attention : 6(%ebx) désigne un emplacement mémoire!

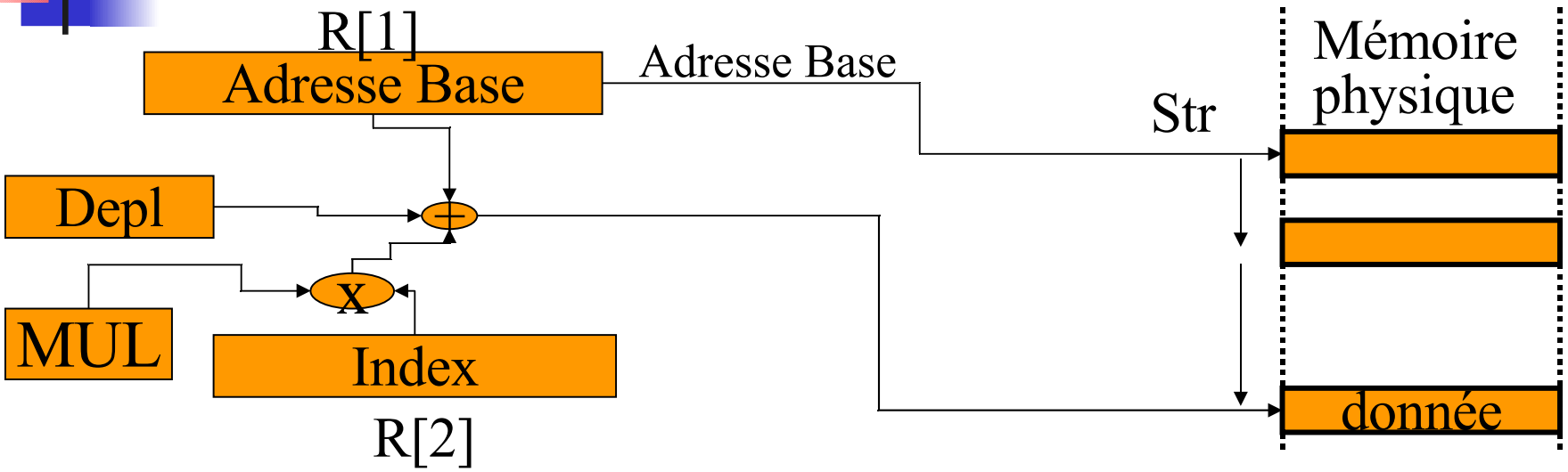
Indirect avec Base, Déplacement, Index



- Notation ass. GNU : Depl(%R[1], %R[2]).
- Déplacement **facultatif**!
- Exemple :

  movl 4(%ebx, %ecx), %eax

Indirect avec Base, Déplacement, Index typé



- Notation ass. GNU : Depl(%R[1], %R[2], MUL).
- Exemple :

  movl 4(%eax, %ecx, 4), %ebx

  movl TAB (%eax, %ecx, 4), %ebx
- Depl facultatif

Adressage mémoire programme

- Utilisation dans les instructions de branchement *jmp*
- Direct : *jmp 0x080483cc* $\Rightarrow$ EIP = *0x080483cc*
- Relatif au registre EIP : *jmp etiq* (le plus courant dans les programmes simples).
  - Exemple: jmp iter # on ajoute un nombre d'octets à
    # EIP
- Indirect :
  - Relais = registre : *% Reg
  - Relais = mémoire donnée : cf. adressage opérandes
    (syntaxe: * devant opérande).
  - * AdRelais : où AdRelais contient l'ad. de l'inst.