Microprocesseurs (MIC)

Chap. ?: Codage des instructions

Objectifs

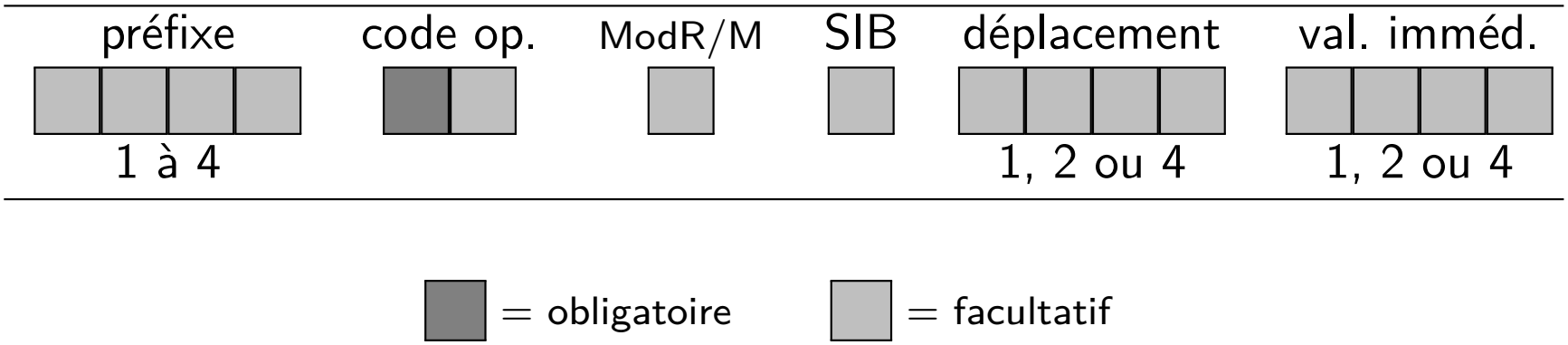
- Comprendre comment sont codées les instructions du `x86`
- Être capable de traduire

$$\text{Assembleur} \Longleftrightarrow \text{Code machine}$$

Plan

- Format général d'une instruction
- Le code opératoire
- Instructions sans opérande
- Instructions avec 1 opérande (cas simples)
- Les bytes ModR/M et SIB
- Modes d'adressages complexes
- Instructions avec 2 opérandes
- Les préfixes

Forme générale d'une instruction



- Longueur variable en fonction de l'"instruction
- Assez complexe $\Rightarrow$ procédons par étapes

Le code opératoire

Identifie l'opération à exécuter

- Souvent sur 1 byte, parfois sur 2
- Parfois, utilise une partie du byte ModR/M (on parle d'extension du code)

Un mnémonique, plusieurs code associés

- En fonction des opérandes
- Ex : **INC** peut se coder 40 à 47, FE ou FF

Un code opératoire, une seule instruction (extension comprise)

- Ex : CD est un **INT**, 01 est un **ADD**

Référence

Où trouver ces codes opératoires et les détails ?

- Dans une référence (disponible sur poÉSI)
- Exemple

DEC

Opcode	Opérandes
FE /1	r/m8
FF /1	r/m16
48+rw	r16
48+rw	r32

- Pas évident à lire ; nous introduirons les notations petit à petit

Instruction sans paramètre

Codée sur 1 byte : le code opératoire de l'instruction

code op.

Exemple : **NOP**

NOP

Opcode	Opérandes
90	

NOP $\Longrightarrow$ 90 (code op.)

Instruction avec une opérande immédiate

Codage : **code op.** | **immédiat sur 1, 2 ou 4 bytes**

INT 0x80

INT

Opcode	Opérandes	Explications
CD ib	imm8	imm8 est le numéro de l'interruption

- **ib** : l'opcode est étendu par une valeur immédiate sur un byte
- **imm8** : l'opérande est une valeur immédiate interprétée sur 1 byte

INT 0x80 $\Longrightarrow$ CD 80

code op. imm8

Coder les registres

Dans les exemples suivants, nous devrons coder des registres. Comment ?

- Via 3 bits

AL, AX, EAX	000	AH, SP, ESP	100
CL, CX, ECX	001	CH, BP, EBP	101
DL, DX, EDX	010	DH, SI, ESI	110
BL, BX, EBX	011	BH, DI, EDI	111

- À quel registre correspond un code ?
  Dépend du contexte.

Instruction avec une opérande registre

Codage : **code op. + n° registre**

INC EBX

INC (extrait)

Opcode	Opérandes	Explications
40 + rd	r32	incrémente le registre désigné

- **+rd** : le numéro du registre 32 bits est ajouté à l'opcode
- **r32** : l'opérande est un registre 32 bits

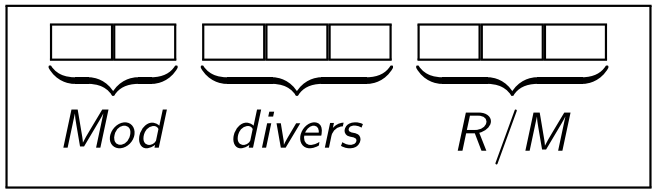
INC EBX $\Longrightarrow$ $\underbrace{\boxed{43}}$

40 (code op.) + 3 (EBX)

Le byte ModR/M

Utilisé pour les modes d'adressages complexes

ModR/M



- **Mod** : mode d'adressage pour la partie R/M
- **R/M** : opérande
  (peut nécessiter d'autres bytes)
- **divers** : dépend du nombre d'opérandes
  - 1 opérande : extension du code opératoire
  - 2 opérandes : deuxième opérande (un registre)

Mode d'adressage plus complexe

Analysons la référence de **INC**

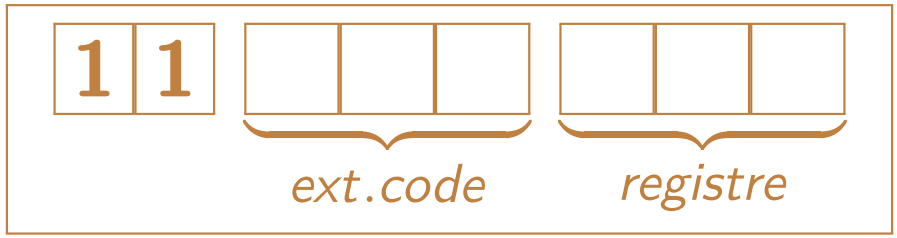
INC (extrait)

Opcode	Opérandes
FF /0	r/m32

- **r/m32** : l'opérande est un registre 32 bits ou une mémoire 32 bits
  - utilisation du byte ModR/M
  - permet les différents adressages complexes
  - nous allons les voir un à un
- **/0** : la partie *divers* de ModR/M vaut 0
  - on parle d'extension de code

Mode d'adressage par registre

Codage : **code op.**

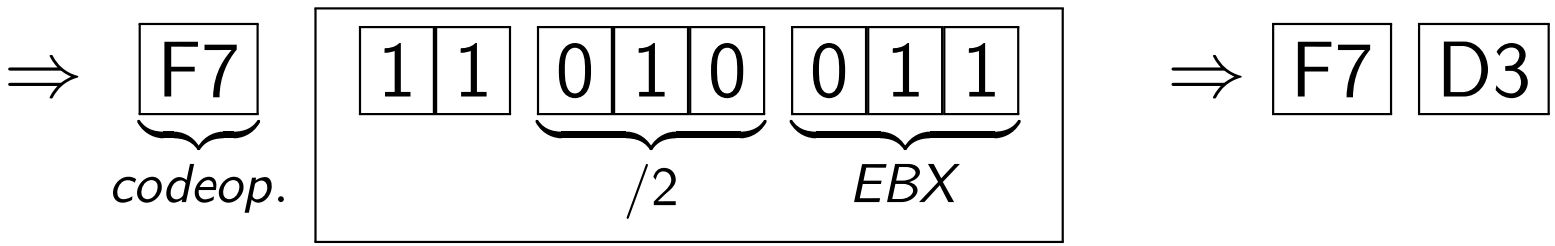


Inutile pour **INC** qui propose une autre forme mais ce n'est pas toujours le cas

NOT EBX

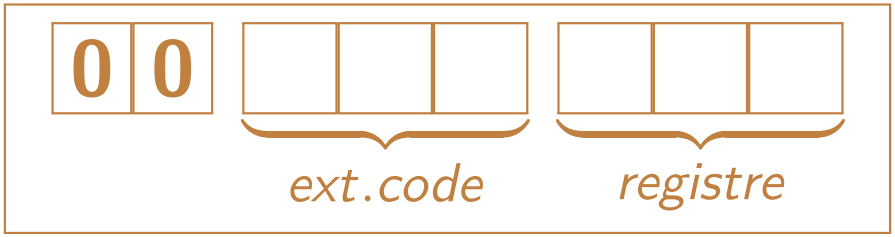
NOT (extrait)

Opcode	Opérandes
F7 /2	r/m32



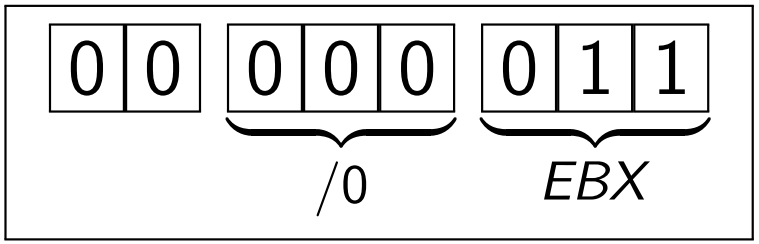
Mode d'adressage indirect

Codage : **code op.**



INC dword [EBX]

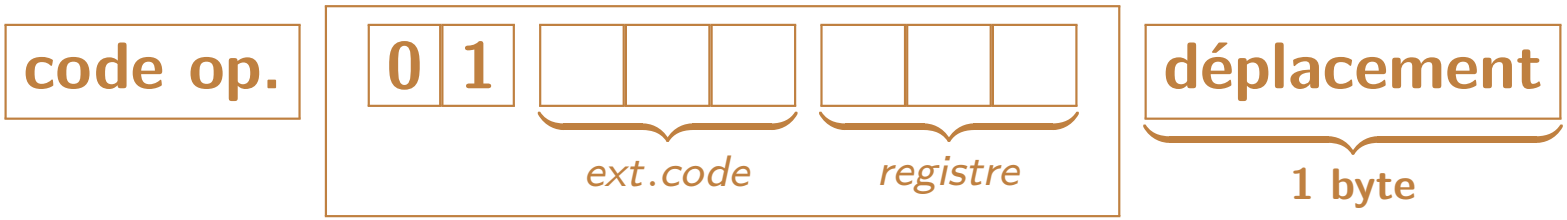
$\Rightarrow$ FF (*codeop.*)



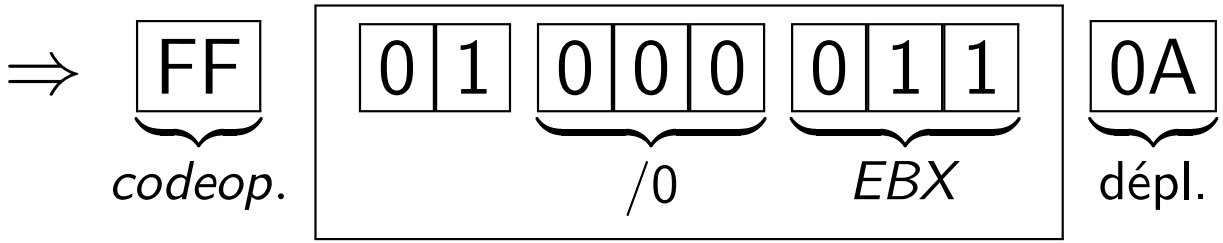
$\Rightarrow$ FF 03

Remarque : le registre ne peut être ni **EBP** ni **ESP**

Adressage indirect avec déplacement court



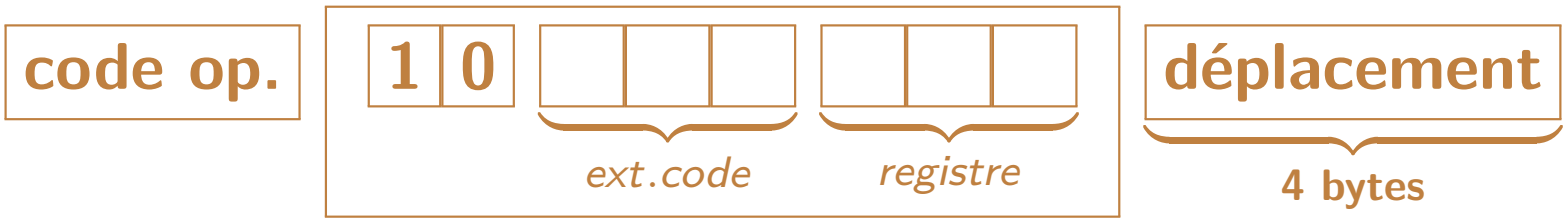
INC dword [EBX + 10]



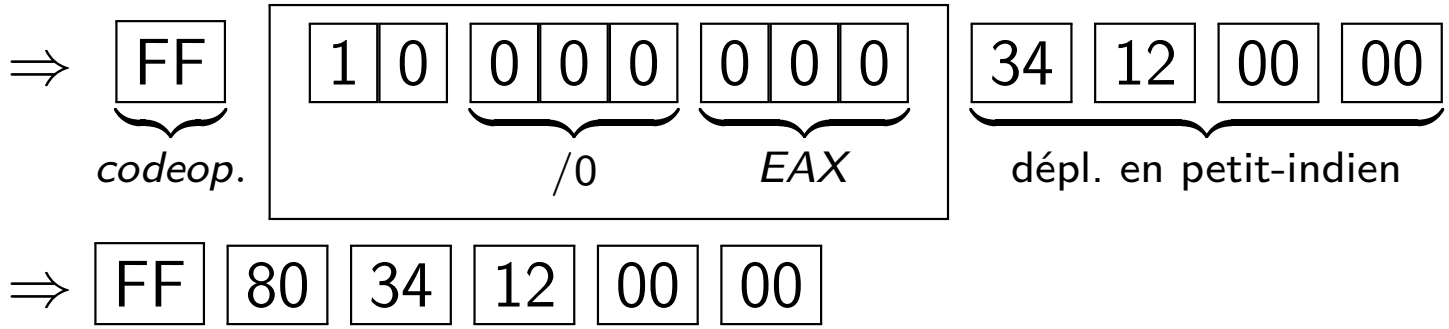
⇒ FF 43 0A

Remarque : le registre ne peut pas être **ESP**

Adressage indirect avec déplacement long

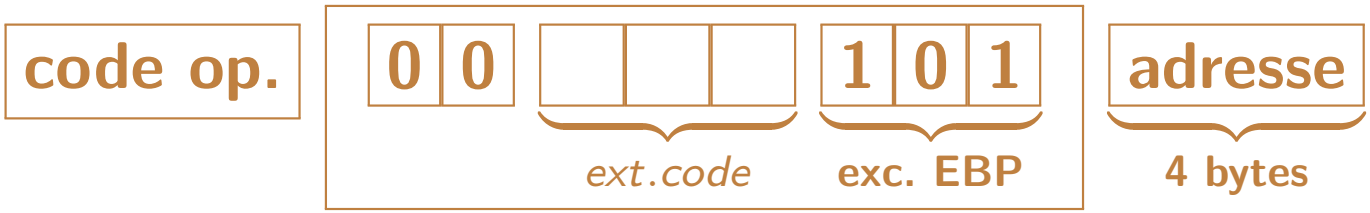


INC dword [EAX+0x1234]

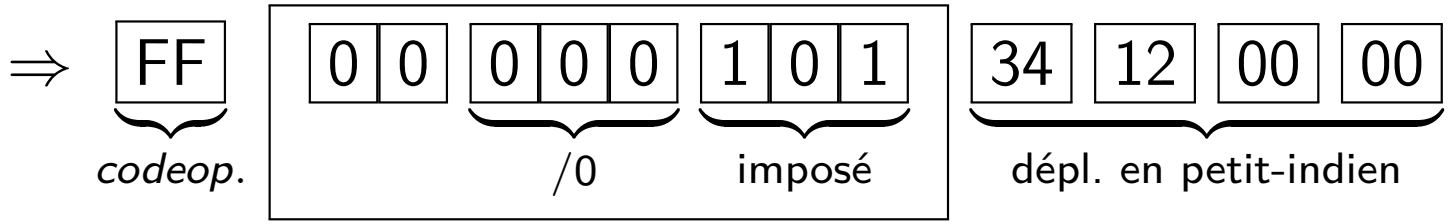


Remarque : le registre ne peut pas être **ESP**

Adressage direct



INC dword [0x1234]

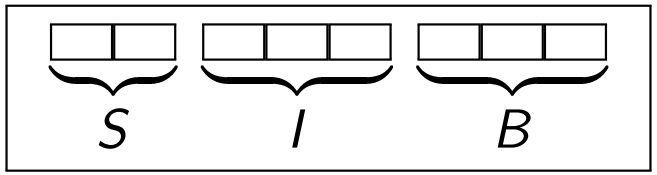


⇒ FF 05 34 12 00 00

Le byte SIB

- Utilisé en complément à ModR/M
- Pour les modes d'adressages indexés $(B + S \times I)$

SIB



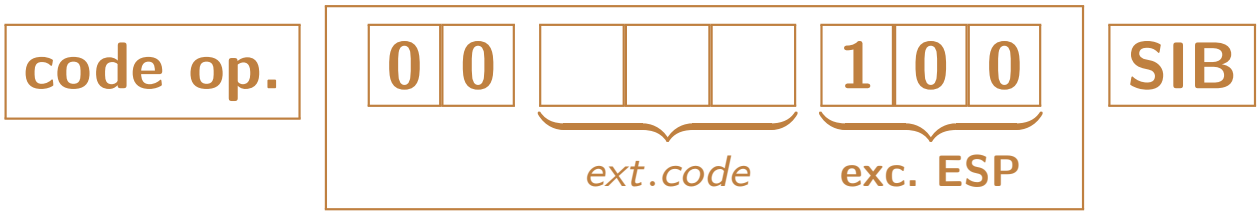
S facteur multiplicatif (scale)

$$2^i \Rightarrow 00 = 1\times, 01 = 2\times, 10 = 4\times, 11 = 8\times$$

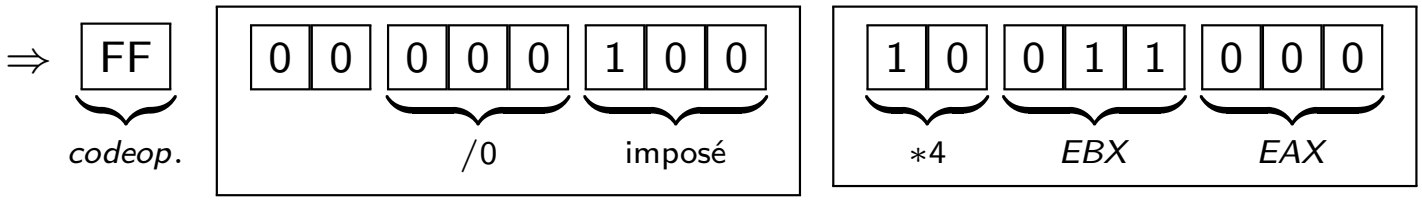
I registre d'index

B registre de base

Indirect indexé

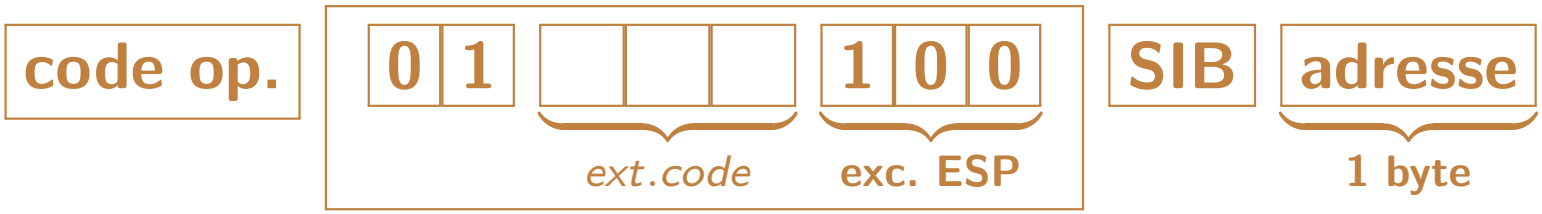


INC dword [EAX+4∗EBX]

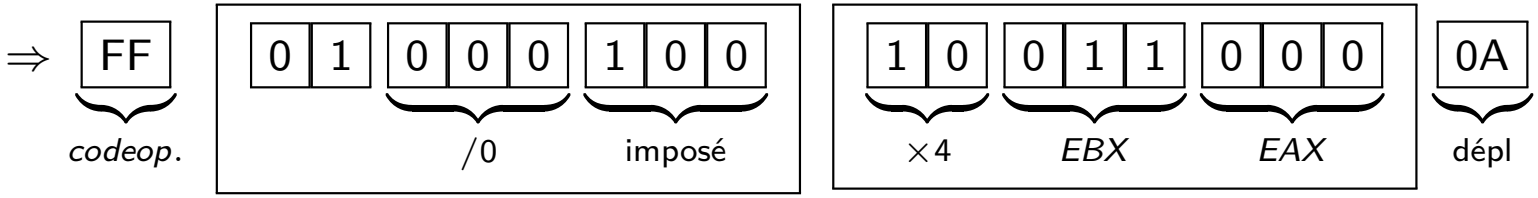


⇒ FF 04 98

Indirect indexé avec déplacement court



INC dword [EAX+4∗EBX+10]



⇒ FF 44 98 0A

ModR/M - récapitulatif

Adressage	Exemple	ModR/M
registre	EAX	1 1 . . . . . .
indirect	[EAX]	0 0 . . . . . .
indirect + court	[EAX+10]	0 1 . . . . . .
indirect + long	[EAX+800]	1 0 . . . . . .
direct	[adresse]	0 0 . . . 1 0 1
indirect indexé	[EAX+4*EBX]	0 0 . . . 1 0 0
indexé + court	[EAX+4*EBX+10]	0 1 . . . 1 0 0
indexé + long	[EAX+4*EBX+800]	1 0 . . . 1 0 0

Instruction avec deux opérandes

Remarques générales

- Une des deux opérandes sera toujours un registre (mode registre)
- Pour l'autre on aura toutes les possibilités

Exemples

- **ADD EAX**, brol (registre, immédiat) $\Rightarrow$ OK
- **ADD [EAX], EBX** (indirect, registre) $\Rightarrow$ OK
- **ADD EAX, EBX** (registre, registre) $\Rightarrow$ OK
- **ADD [EAX], [EBX]** (indirect, indirect) $\Rightarrow$ interdit

Instruction avec deux opérandes

Le code opératoire va dépendre de l'ordre des opérandes

ADD (extrait)

Opcode	Opérandes
01 /r	r32/mem32, r32
03 /r	r32, r32/mem32

Exemples

- **ADD EAX**, brol utilise le code 03
- **ADD [EAX]**, **EBX** utilise le code 01
- **ADD EAX**, **EBX** utilise le code 01 ou 03

Exemple

ADD ECX, [EBX]

ADD (extrait)

Opcode	Opérandes
01 /r	r32/mem32, r32
03 /r	r32, r32/mem32

$\Rightarrow$ 03 (*codeop.*) | 0 0 (*indirect*) | 0 0 1 (*ECX*) | 0 1 1 (*EBX*)

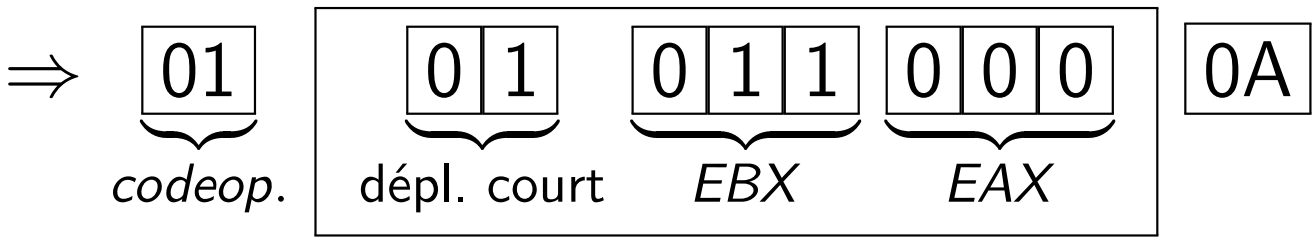
$\Rightarrow$ 03 0B

Exemple

ADD [EAX+10], EBX

ADD (extrait)

Opcode	Opérandes
01 /r	r32/mem32, r32
03 /r	r32, r32/mem32



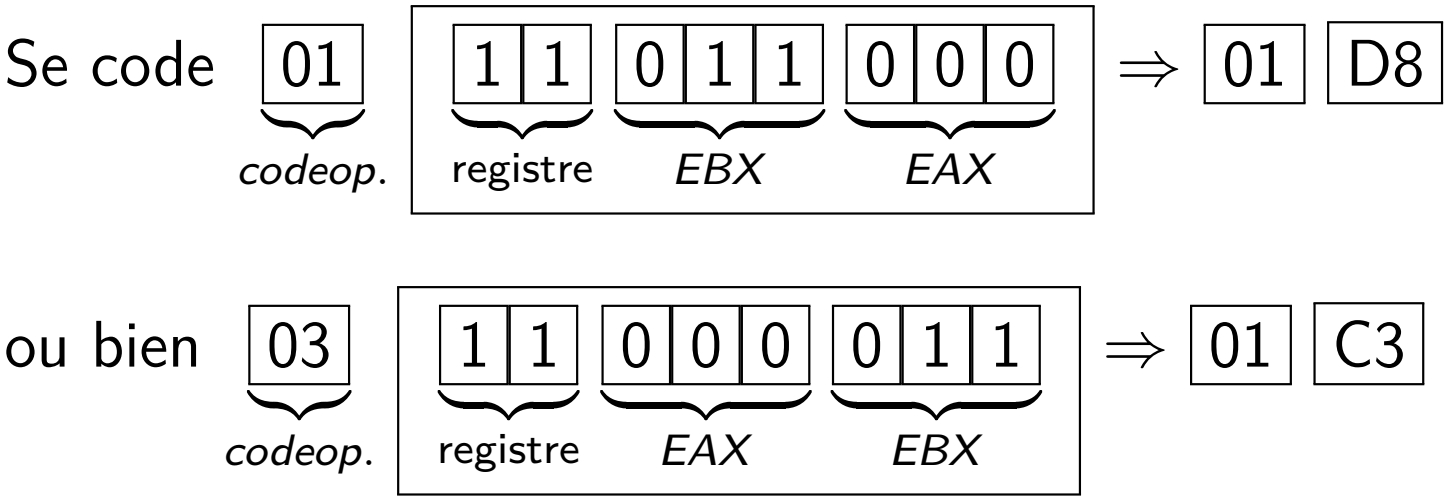
⇒ 01 58 0A

Exemple

ADD EAX, EBX

ADD (extrait)

Opcode	Opérandes
01 /r	r32/mem32, r32
03 /r	r32, r32/mem32



Les préfixes

Les préfixes modifient le comportement de l'instruction :

- **Répétition** : répétition automatique de l'instruction
- **Taille adresse** : modifie la taille par défaut des adresses (16/32)
- **Taille registre** : modifie la taille par défaut des registres (16/32)
- **Segment** : modifie le segment par défaut

À placer dans cet ordre

Modification du segment

Rappel : une adresse est sous la forme **base : offset**

- **base** : est l'adresse de début du segment
- Une table reprend les descripteurs de segments
- Des registres spécialisés `CS`, `DS`, `ES`, `FS`, `GS` et `SS` donnent l'index dans la table des segments
- **offset** : est le déplacement dans le segment (spécifié par le programmeur)

Modification du segment

Le segment utilisé dans une instruction est implicite

- **INC [AX]** utilisera le segment des données (DS)
- **JMP** brol utilisera le segment du code (CS)

Peut être modifié (ex : **ADD [ES:EAX], EBX**)

Codé dans le préfixe

2E = CS	3E = DS	26 = ES
64 = FS	65 = GS	36 = SS

ADD [ES:EAX], EBX $\Rightarrow$ 26 01 18

Préfixe de répétition

Permet de répéter `ECX` fois une instruction

- En assembleur, on préfixe par **REP**
- Codé `F3`

Exemple : Répétition de l'instruction **MOVSB**

- **MOVSB** copie un byte de `[ESI]` vers `[EDI]` puis incrémente `ESI` et `EDI`.
- Donc, pour copier les 10 premiers bytes de chaine1 dans chaine2

```
MOV ESI, chaine1
MOV EDI, chaine2
MOV ECX, 10
REP MOVSB          ; codé F3 A4
```

Taille des des données et des adresses

Extrait de la référence de ADD

ADD (extrait)

Opcode	Opérandes
00 /r	r8/m8, r8
01 /r	r16/m16, r16
01 /r	r32/m32, r32

Mais alors, **ADD EAX, EBX** et **ADD AX, BX** se codent de la même façon ?

Taille des des données et des adresses

Un bit `D` dans le descripteur de segment indique si on utilise des registres/adresses 16 bits (`D=0`) ou 32 bits (`D=1`)

Spécifié en assembleur via les macros [BITS 16] et [BITS 32]

Pour une instruction donnée, peut être modifié pour via les préfixes

- `0x66` : pour la taille d'une donnée
- `0x67` : pour la taille d'une adresse

Taille des des données et des adresses

Exemples

```
[BITS 16]
INC AX              ; pas de préfixe
INC EAX             ; 0x66 : donnée sur 32 bits
INC word [EAX]      ; 0x67 : adresse 32 bits
INC dword [EAX]     ; 0x66 0x67 : donnée et adresse 32 bits
ADD AX, BX          ; pas de préfixe
ADD EAX, EBX        ; 0x66 : données sur 32 bits
ADD EAX, [EBX]      ; 0x66 0x67 : donnée et adresse 32 bits
```