Architecture des ordinateurs

Bilan des TP

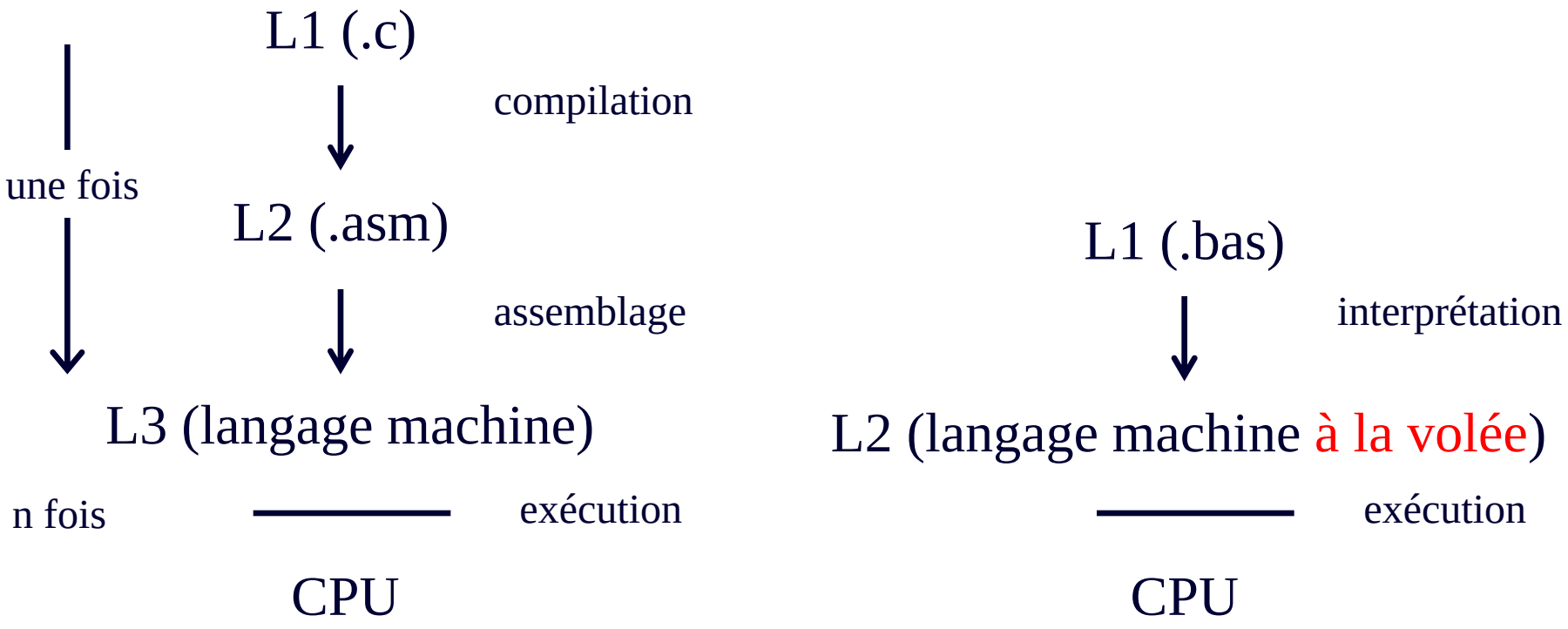
Philippe Darche
IUT Paris Descartes

Plan du TP1

- Chaîne de développement logiciel générique d'un programme écrit en langage compilé
- Découverte d'un source en langage d'assemblage
- Assemblage/édition de liens/exécution/débogage

Interprétation/exécution (avec exemples)

- Passage d'un langage à un autre → traduction



Exécution de programme et gestion mémoire

Au lancement d'un programme

- Chargement du code compilé [code segment]
- Allocation de l'espace mémoire nécessaire pour stocker les variables globales et statiques [data segment]
- Initialisation des variables globales qui le nécessitent
- Appel de la fonction main

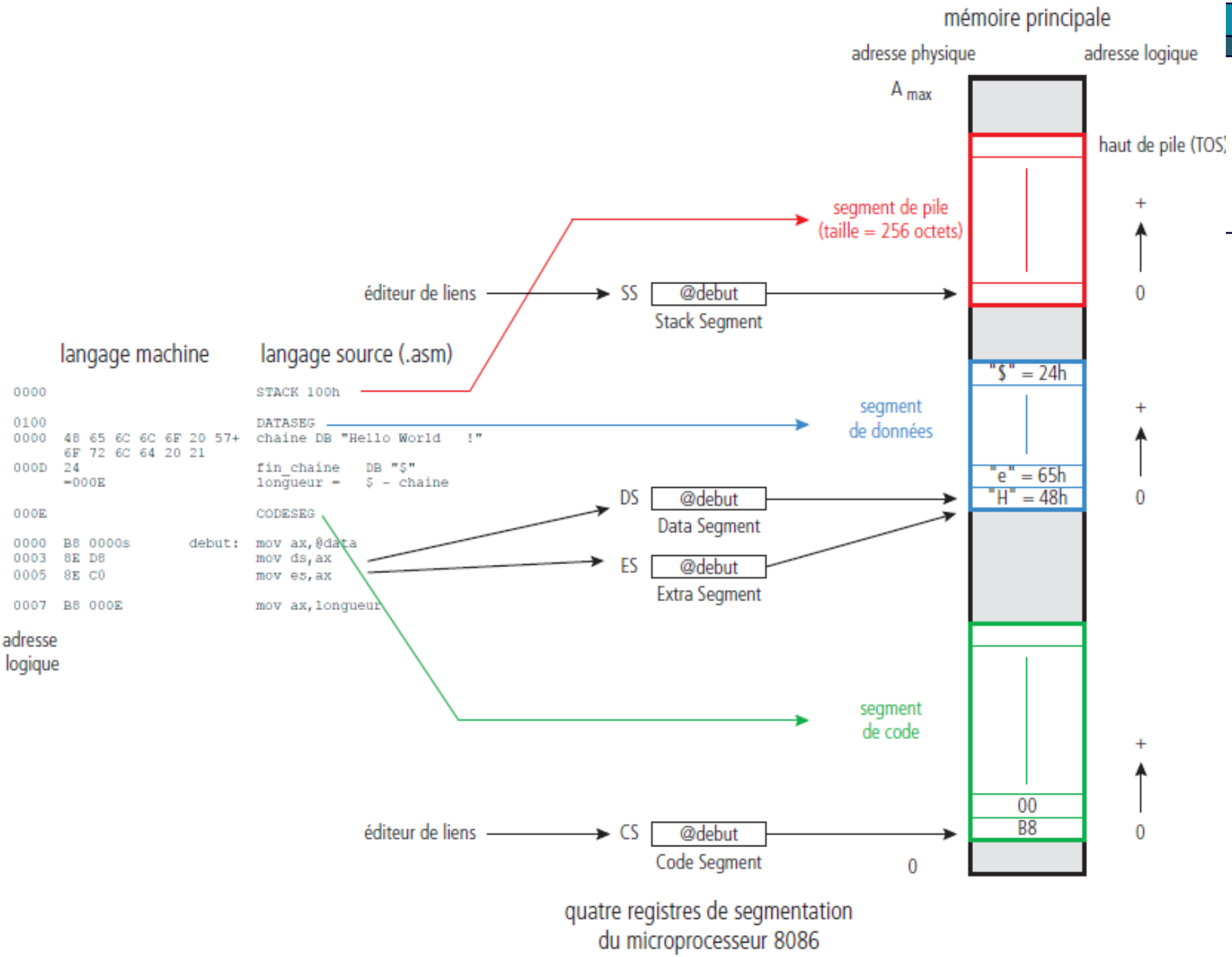
A chaque appel de fonction

- Interruption d'exécution du programme appelant
- Mise en place du contexte d'exécution propre à la fonction dans la pile d'éxécution [stack]

  Remarque : plusieurs contextes d'exécution peuvent être empilés dans la pile d'exécution (ajout d'un contexte d'exécution dès qu'une fonction en appelle une autre)
- Exécution de la fonction dans son contexte d'exécution

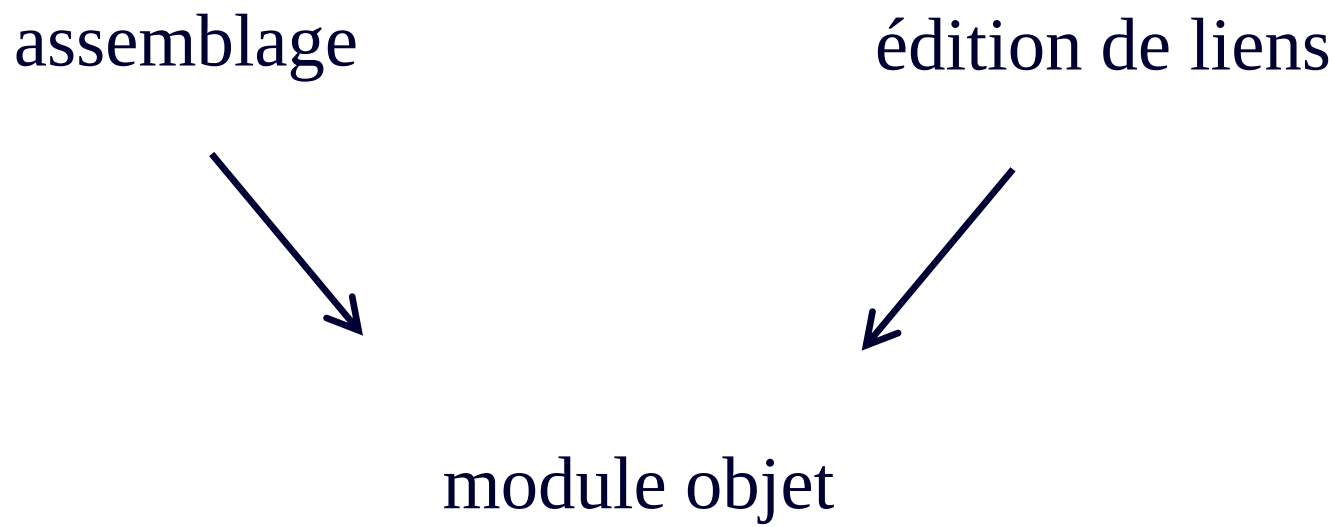
Qui gère les segments ?

- Le programmeur en langage d’assemblage (nous !)
- Le compilateur (de manière automatique)



quatre registres de segmentation
du microprocesseur 8086

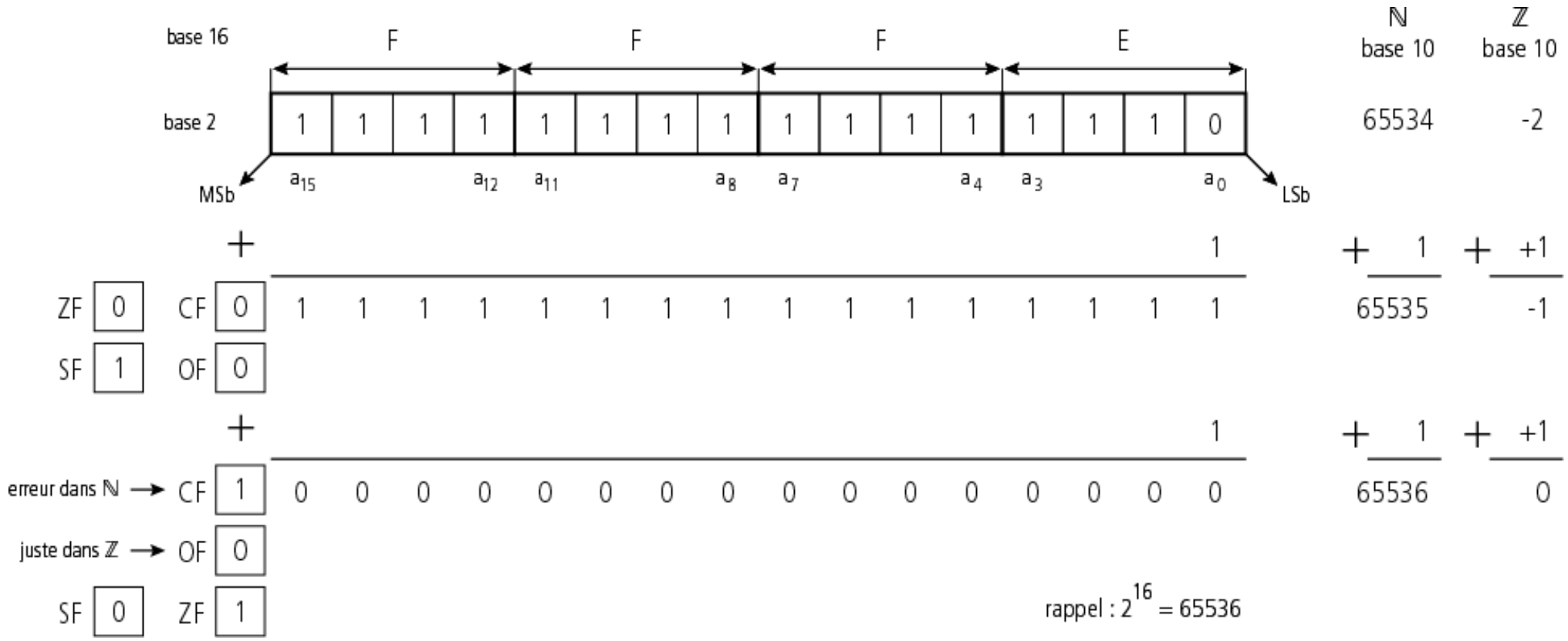
Qui génère le module objet ?



Plan du TP2

- Le fichier listing
- Addition en binaire naturel au format n = 16 bits
- Appellation des registres de données
- Syntaxes d'une instruction
- Règles du programmeur en langage d'assemblage liées à la micro-architecture du 8086

Interprétation du résultat d'une addition



Syntaxe d'une instruction du 8086

- Zéro opérande
  - nop
- Un opérande
  - not al ; complément à 1 du registre al
- Deux opérandes (au maximum !)
  - <instruction> opérande_destination,opérande_source
  - <instruction> opérande_gauche,opérande_droite
    - mov [M1],dx
    - add cx,[M1]

Les règles d'or

- 1ère règle d'or
  - une instruction ne peut accéder à au plus un opérande en mémoire centrale.

  ⇒ utilisation nécessaire des registres internes du MPU

  mais il y a des exceptions !

  exemple : dec [var]
- 2ème règle d'or
  - deux opérandes au maximum par instruction

Sémantique/syntaxe autorisées

Lignes d'instructions	Nombre d'accès à la mémoire pour les opérandes	Nombre d'accès à la mémoire pour ranger le résultat	Remarques
mov ax,[M1]	1	0	
mov [M1],dx	0	1	
mov [M2],[M1]	1	1	interdit non-respect de la règle d'or
add dx,[M1]	1	0	
add [M1], dx	1	1	contre-exemple
add [M1], [M1]	~~2~~	1	interdit
add [RM],[M1],[M2]	~~2~~	1	interdit trois opérandes

Autres règles

- Règles d'argent
  - il est préférable d'utiliser un registre plutôt qu'un emplacement mémoire sauf si la valeur n'est utilisée qu'une fois (à apprécier au coup par coup) car un accès mémoire est moins rapide qu'un accès à un registre.
  - les opérandes doivent avoir le même format
    - par exemple mov al,dx interdit !

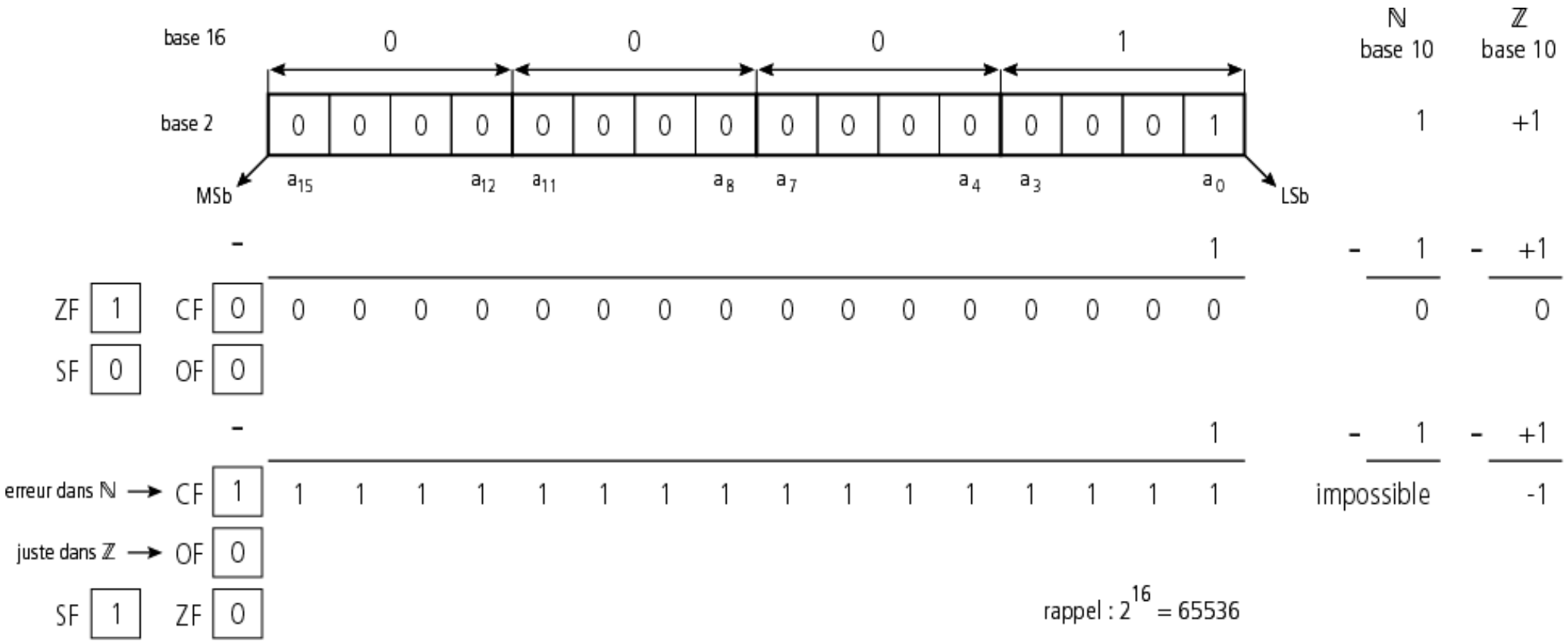
Autres règles (suite et fin)

- Règle de bronze (règle de syntaxe)
  - toute variable mémoire utilisée dans CODESEG doit être encadré (*i.e.* [ ]).
  - seuls des noms de registres d'indirection peuvent être encadrés → mode pointeur
    - ex. : [bx], [si] ou [di]

Plan du TP3

- Soustraction en binaire naturel au format n = 16 bits
- Implémentation d'une structure de contrôle
- Affichage d'une chaîne de caractères

Interprétation du résultat d'une soustraction



Implémentation d'une structure de contrôle

- Faire la correspondance mots réservés – étiquette
- Condition complémentaire implémentée
- Seule une étiquette est utilisée (*i.e.* fin_si1)
  - les deux autres structurent le code (lisibilité)

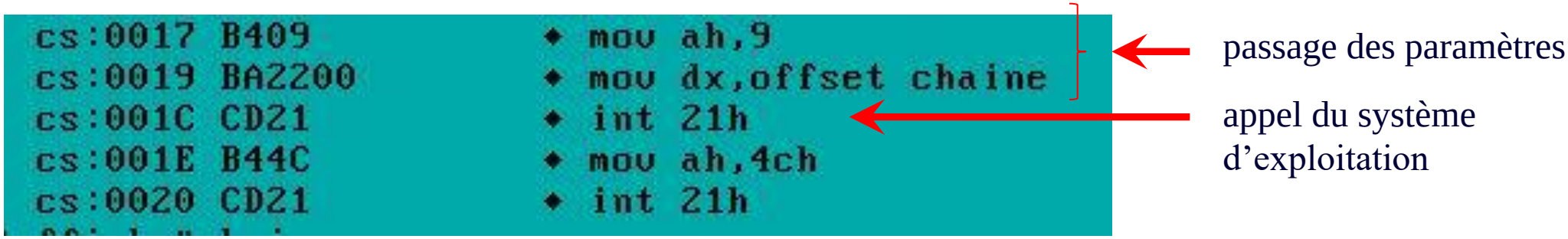
```
si nombre = 0
alors
        code du alors
fin_si
```

```
si1:    cmp [nombre],0
        jne fin_si1
alors1:
        code du alors
fin_si1:
```

Appel à une fonction

- Utilisation d'une instruction de saut
  - exemple : *jmp* nom_fct
  - mais comment faire pour calculer l'adresse de retour ?
- Instructions spécialisées
  - *call* nom_fct et *ret* pour le 8086
- Appel système par interruption
  - *int* 21 h (DOS) ou *int* 80h (linux ou FreeBSD)
- Passage des paramètres avant l'appel effectif
  - contrairement aux langages de haut niveau

Affichage d'une chaîne de caractères



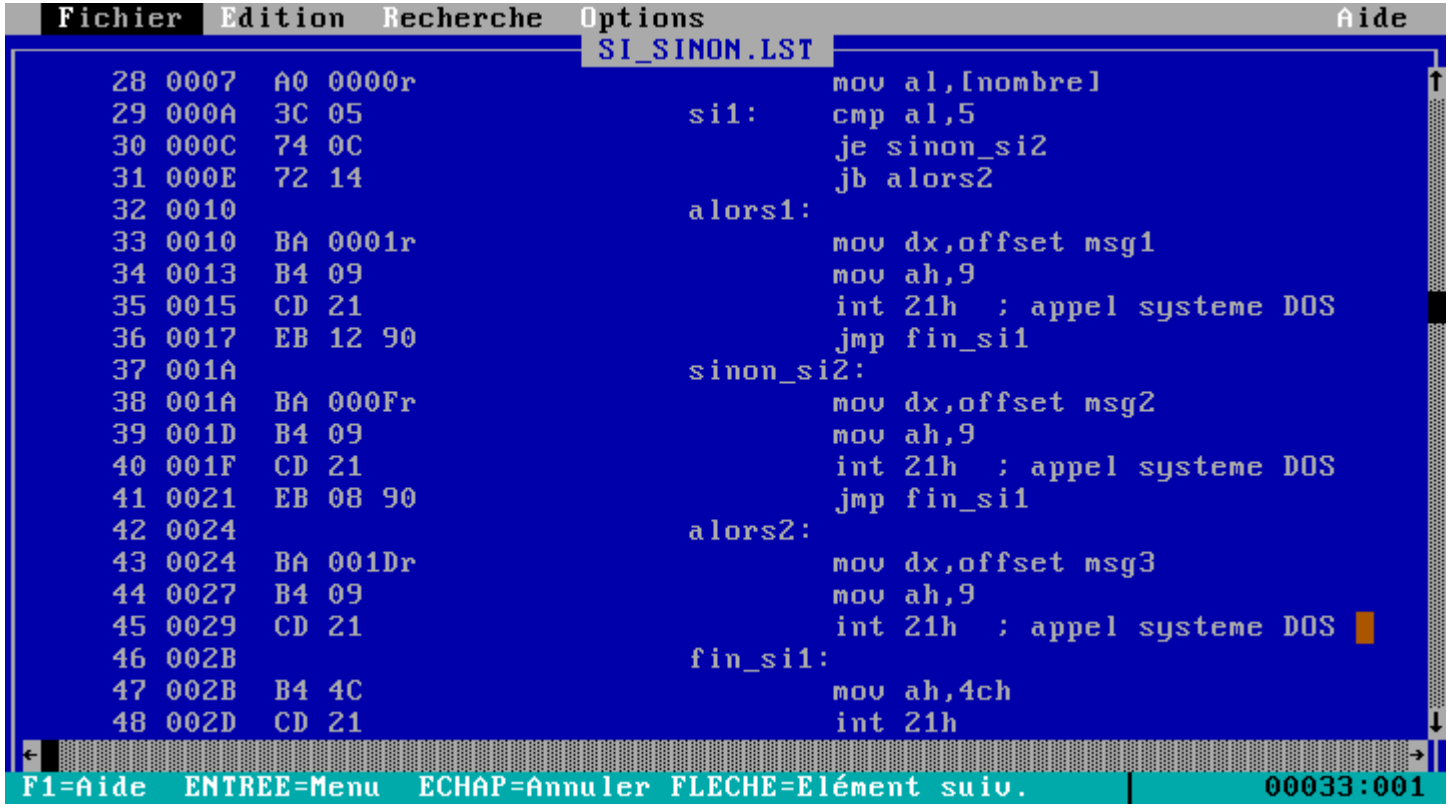
- Passage des paramètres
  - pour le système (*i.e.* N° de fonction)
  - Pour la fonction (*i.e.* adresse de début de chaîne)

Plan du TP4

- Implémentation d'une structure de contrôle complexe
- Calcul des adresses de saut
- Préparation du DST

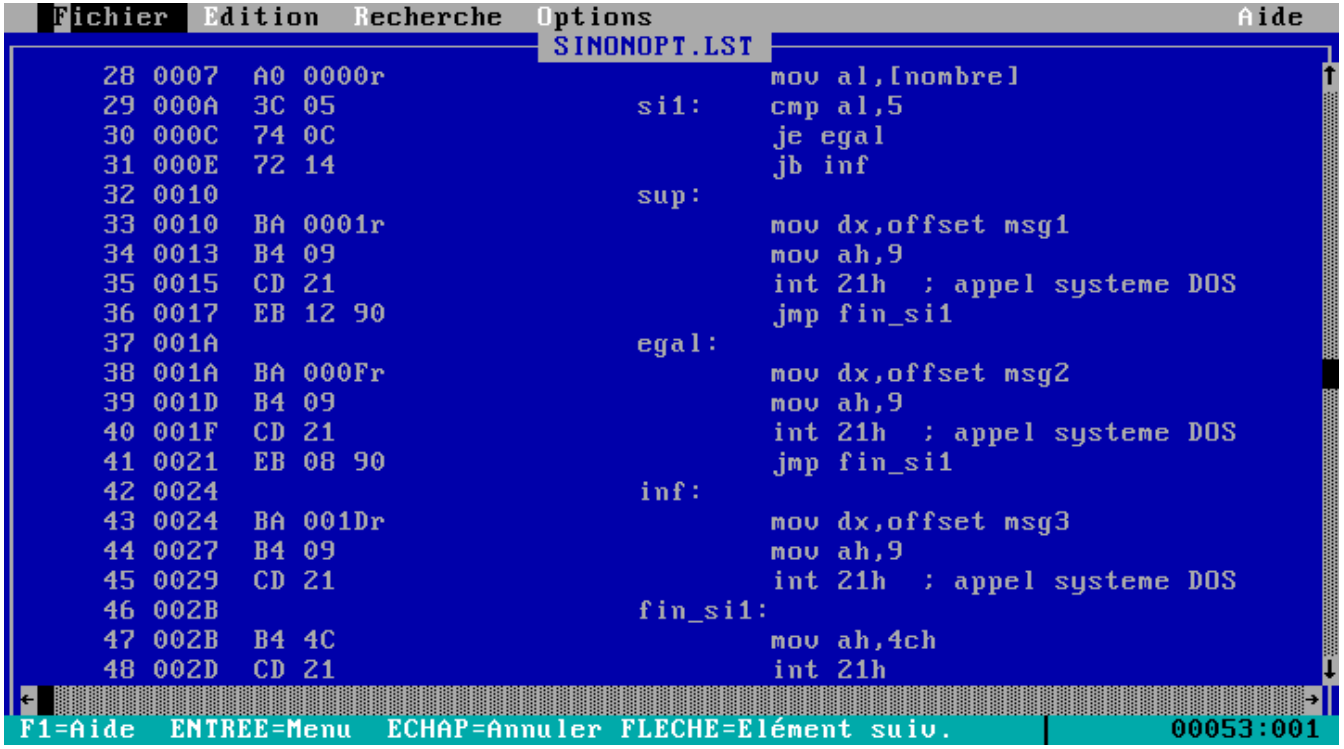
Structure de contrôle complexe

- Le « si_alors_sinon »



Optimisation possible pour les « experts »

- Diminution du nombre des étiquettes
- Meilleur nom d'étiquette



Calcul de l'adresse du saut conditionnel

- Adressage relatif (à la position de l'instruction qui suit celle du saut) sur un octet
  - offset (*i.e.* déplacement) ∈ [-129,+127] octets

JB — JUMP ON BELOW
JNAE — JUMP ON NOT ABOVE OR EQUAL

Operation if (CF) = 1 then
(IP)←(IP) + disp (sign-extended to 16-bits)

Flags Affected None

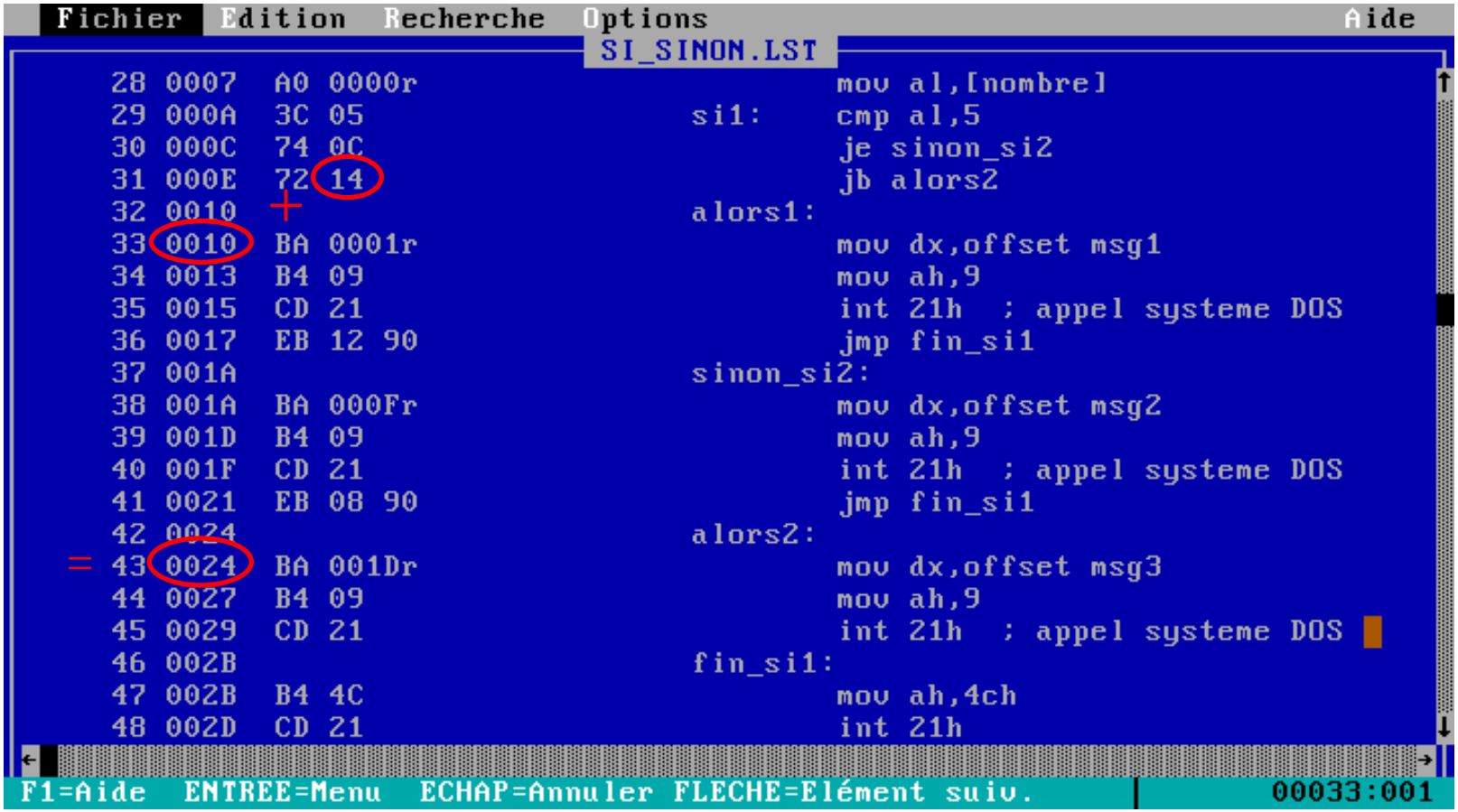
Description JB (Jump on Below)/JNAE (Jump on Not Above or Equal) transfers control to the target operand (IP + displacement) if CF = 1.

Encoding

0 1 1 1 0 0 1 0	disp

JB/JNAE Operands	Clocks	Transfers	Bytes	JB Coding Example
short-label	16 or 4	–	2	JB BELOW

Calcul de l'adresse du saut conditionnel



Calcul de l'adresse du saut inconditionnel

- Adressage relatif
  si saut intra-segment
  - court pour notre exemple
    - déplacement sur un octet

JMP — JUMP UNCONDITIONALLY

Operation if Inter-Segment then (CS)←SEG
(IP)←DEST

Flags Affected None

Description **JMP** *target*

JMP unconditionally transfers control to the target location. Unlike a CALL instruction, JMP does not save any information on the stack; no return to the instruction following the JMP is expected. Like CALL, the address of the target operand may be obtained from the instruction itself (direct JMP), or from memory or a register referenced by the instruction (indirect JMP).

An intrasegment direct JMP changes the instruction pointer by adding the relative displacement of the target from the JMP instruction. If the assembler can determine that the target is within 127 bytes of the JMP, it automatically generates a two-byte instruction form called a SHORT JMP; otherwise, it generates a NEAR JMP that can address a target within ±32K. Intrasegment direct JMPS are self-relative and appropriate in position-independent (dynamically relocatable) routines in which the JMP and its target are moved together in the same segment.

An intrasegment indirect JMP may be made either through memory or a 16-bit general register. In the first case, the word content referenced by the instruction replaces the instruction pointer. In the second case, the new IP value is taken from the register named in the instruction.

An intersegment direct JMP replaces IP and CS with values contained in the instruction.

An intersegment indirect JMP may be made only through memory. The first word of the doubleword pointer referenced by the instruction replaces IP and the second word replaces CS.

Encoding

Intra-Segment Direct

1 1 1 0 1 0 0 1	disp-low	disp-high

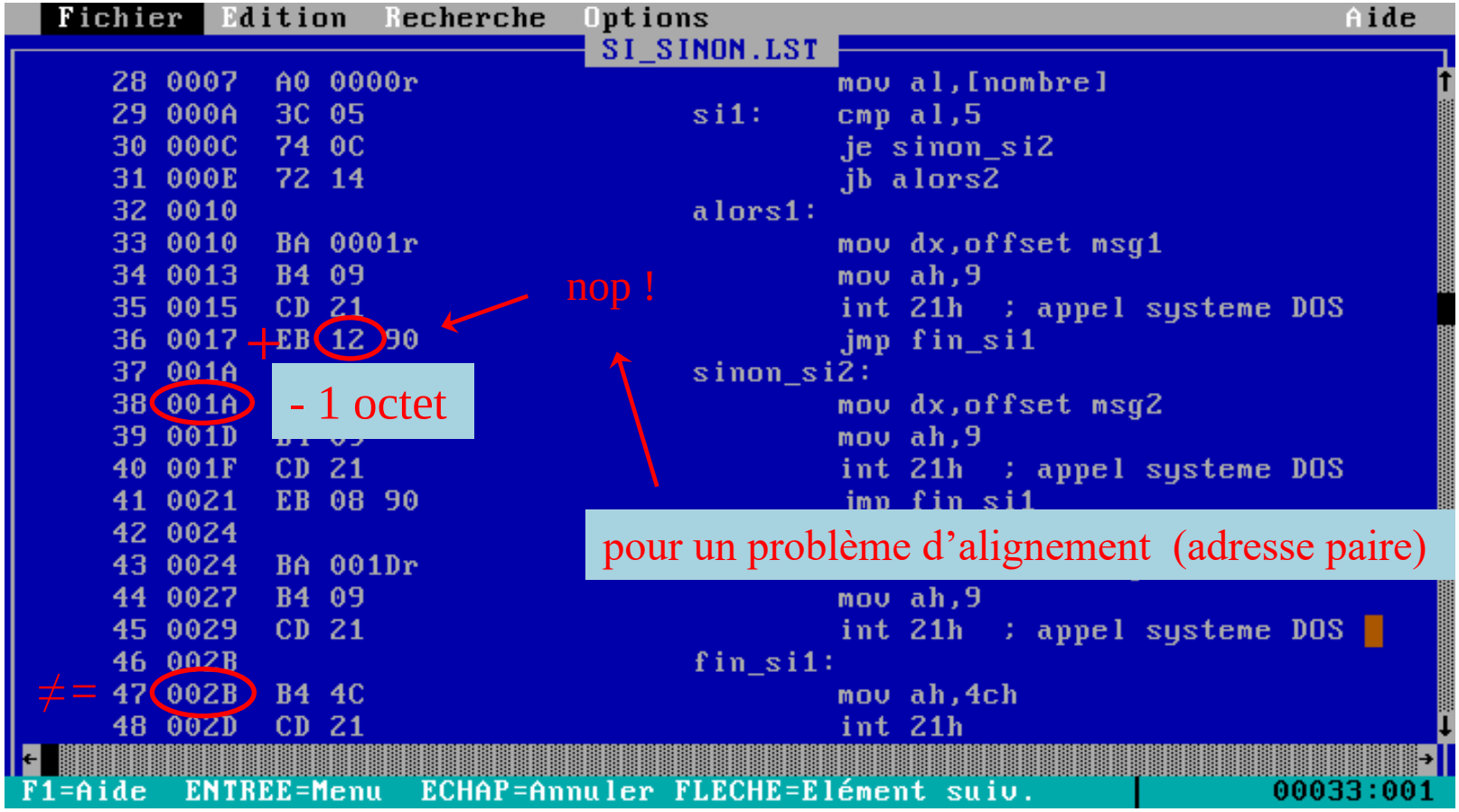
DEST = (IP) + disp

Intra-Segment Direct Short

1 1 1 0 1 0 1 1	disp

DEST = (IP) + disp sign extended to 16-bits

Calcul de l'adresse du saut inconditionnel



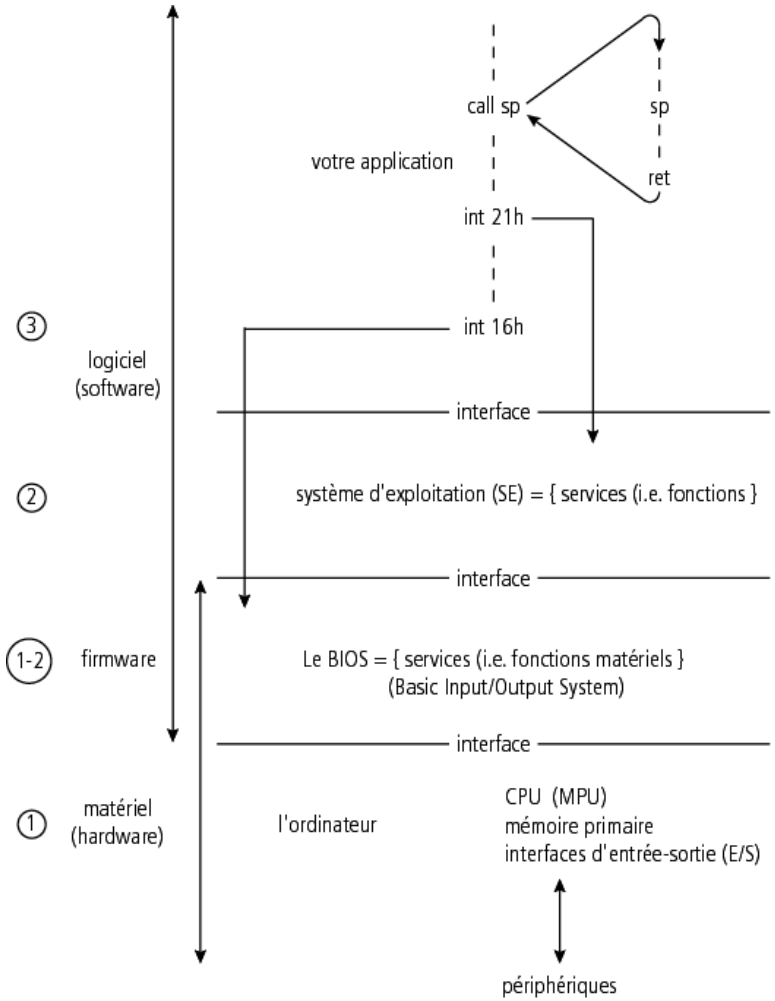
Pour se préparer au DST

- Voir les sujets de DST des années précédentes
- TP 9 exercice 1
- Vous devez savoir utiliser les instructions vues en TP
  - add, sub, mul, jmp, jxx, loop, etc.

Plan du TP5

- Appel d'une fonction (le retour !)
  - saut inter-segment (complément du cours)
  - formes d'appel
    - appel classique (call/ret)
    - appel par interruption
      - appel (au) système (d'exploitation)
      - appel au BIOS
- DOSBox
  - émulateur DOS pour vos TP

Formes d'appel d'une fonction



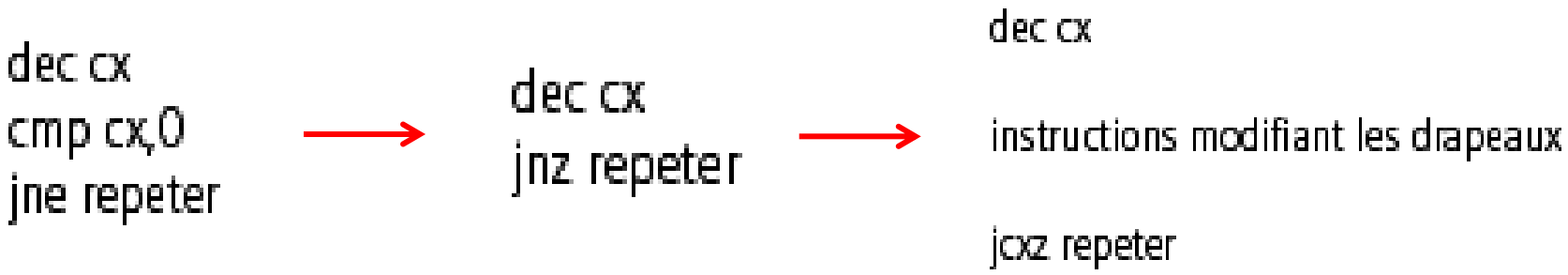
Compléments : instructions de saut conditionnel

- Schéma de base
  - instruction qui positionne les drapeaux
  - instruction qui les lit et qui prend une décision

```
cmp ax,4                    sub al,2
ja suite                    jnc suite
                        erreur:
                            code pour dépassement
                        suite:
```

Instructions conditionnelles

- Parfois des simplifications



Structures de contrôle

- Pensez à prendre la condition complémentaire
  - $>$ → $\leq$, $<$ → $\geq$ et $=$ → $\neq$

```
                si var  >
alors
                bloc 1
sinon
                bloc 2
fin_si
```

```
si1:        cmp ax,5
            jbe sinon
alors1:
            bloc 1
            jmp fin_si1
sinon1:
            bloc 2
fin_si1:
```

Autre exemple

```
faire pour i = 1 à 5

        bloc

fin_faire
```

```
                mov cx,5
faire:          jcxz fin_faire

                bloc

                dec cx
                jmp faire
fin_faire:
```