

Architecture des ordinateurs

Bilan des TP

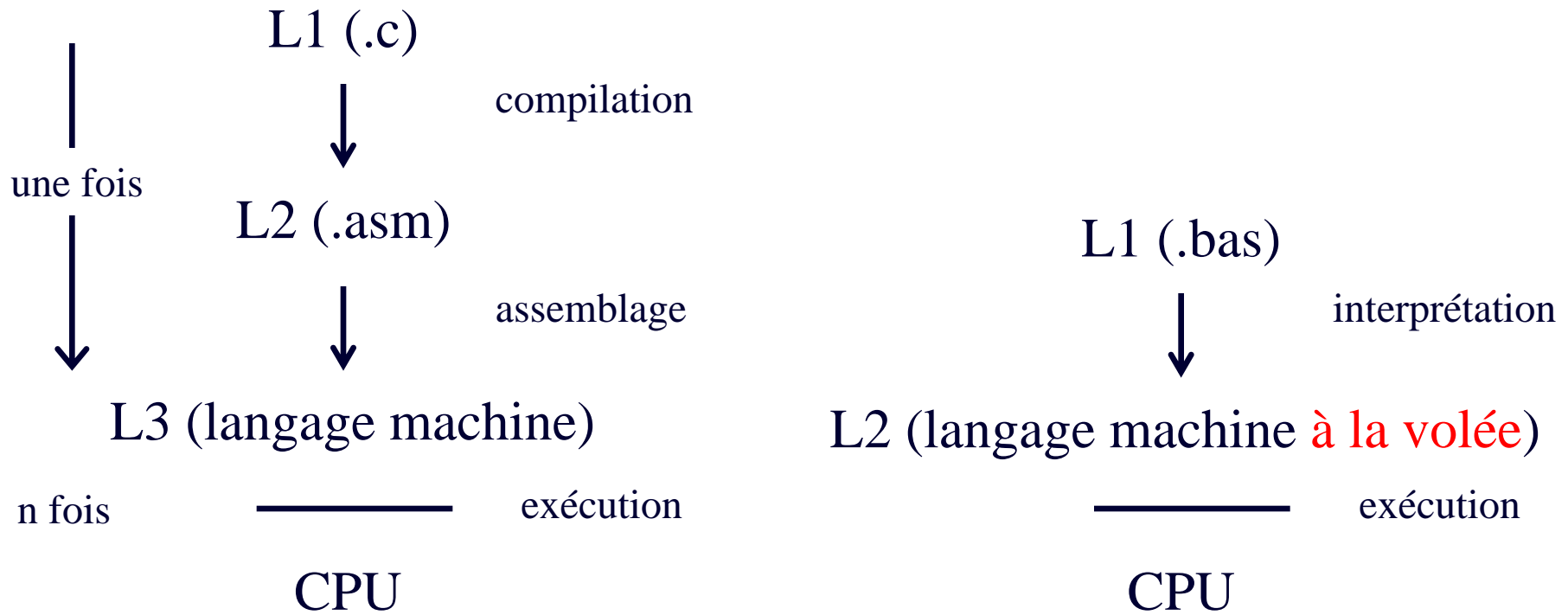
Philippe Darche
IUT Paris Descartes

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

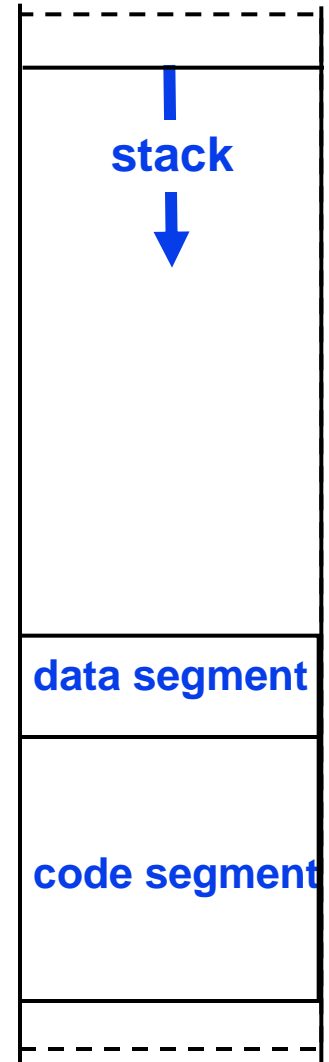
- Chargement du code compilé [code segment]
- Allocation de l'espace mémoire nécessaire pour stocker les variables globales et statiques [data segment]
- Initialisation des variables globales qui le nécessitent
- Appel de la fonction main

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'exécution [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)



mémoire principale

adresse physique

adresse logique

A_{max}

haut de pile (TOS)

segment de pile
(taille = 256 octets)

éditeur de liens

SS

@debut
Stack Segment

langage machine

langage source (.asm)

segment
de données

STACK 100h

DATASEG

chaîne DB "Hello World !"

fin_chaine DB "\$"
longueur = \$ - chaîne

DS

@debut
Data Segment

ES

@debut
Extra Segment

CODESEG

debut: mov ax,@data
mov ds,ax
mov es,ax

mov ax,longueur

segment
de code

éditeur de liens

CS

@debut
Code Segment

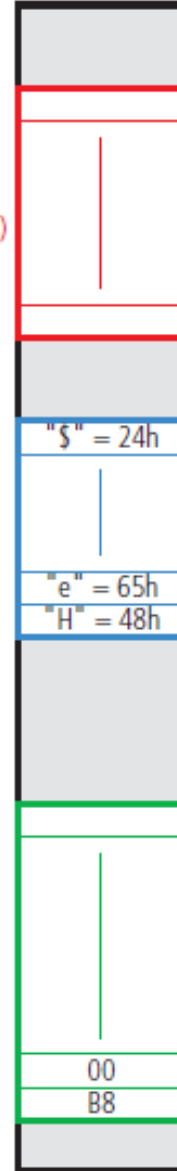
0

adresse
logique

```

0000
0100
0000 4B 65 6C 6C 6F 20 57+
      6F 72 6C 64 20 21
000D 24
      -000E
000E
0000 BB 0000s
0003 8E DB
0005 8E C0
0007 BB 000E

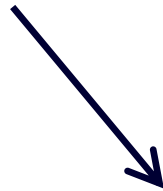
```



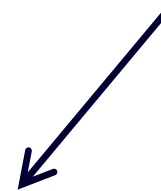
quatre registres de segmentation
du microprocesseur 8086

Qui génère le module objet ?

assemblage



édition de liens

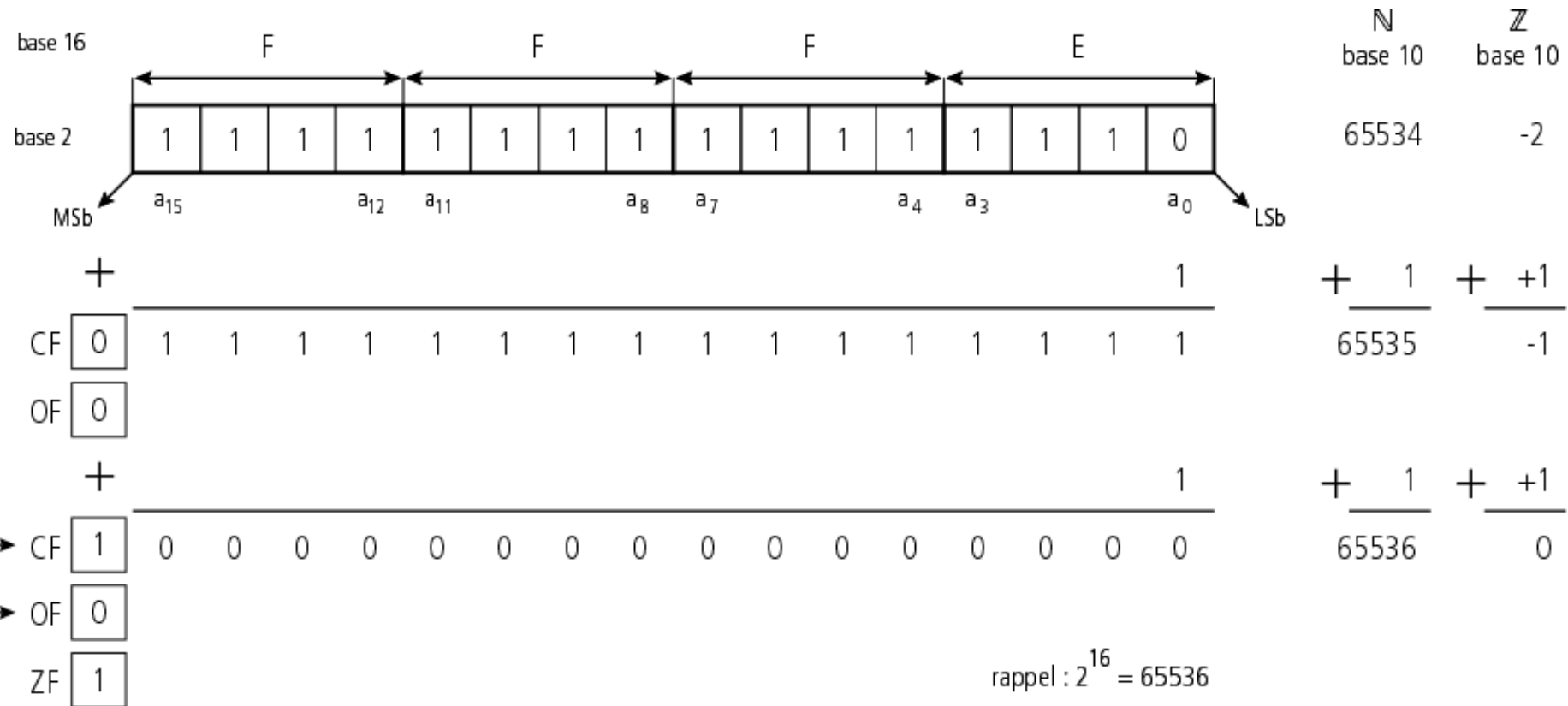


module objet

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

□ 1^{ère} 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]

□ 2^{ème} 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 !

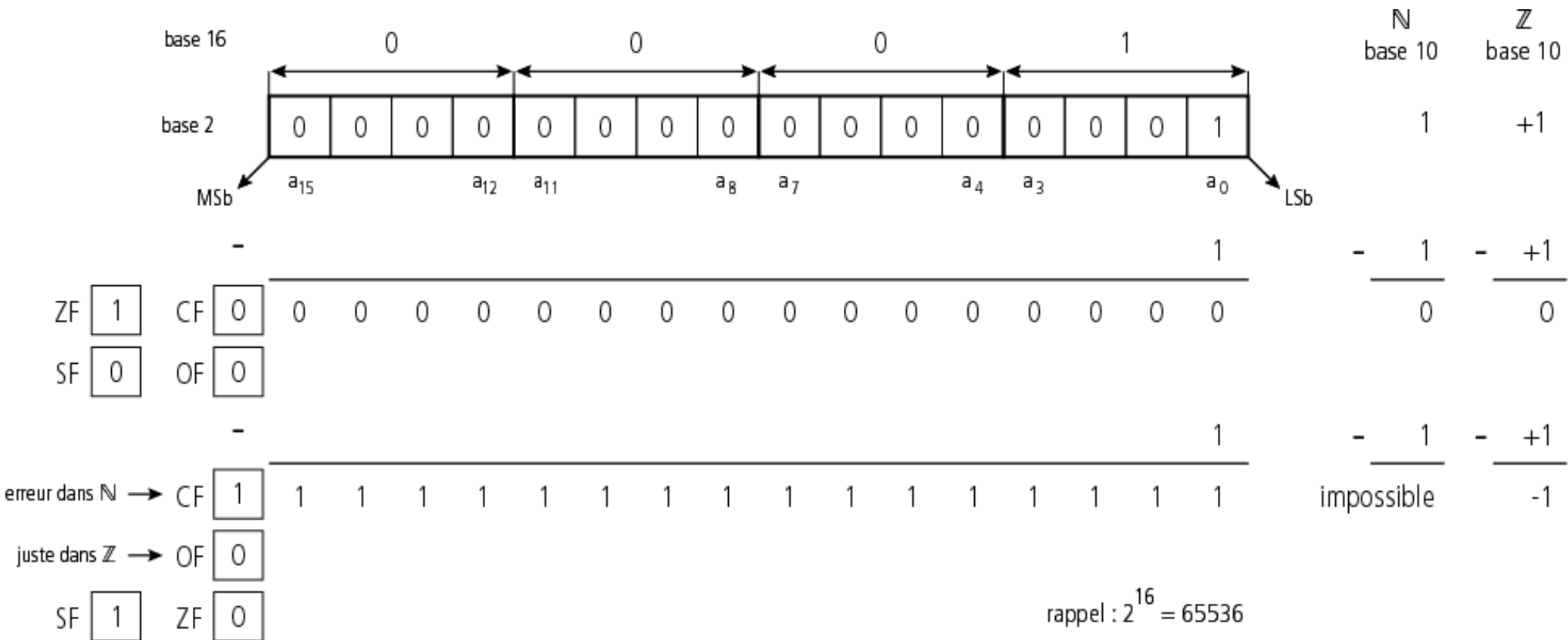
Autres règles (suite et fin)

- 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

```
cs:0017 B409      ◆ mov ah,9
cs:0019 BA2200    ◆ mov dx,offset chaine
cs:001C CD21      ◆ int 21h
cs:001E B44C      ◆ mov ah,4ch
cs:0020 CD21      ◆ int 21h
```

← passage des paramètres

← appel du système d'exploitation

- 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

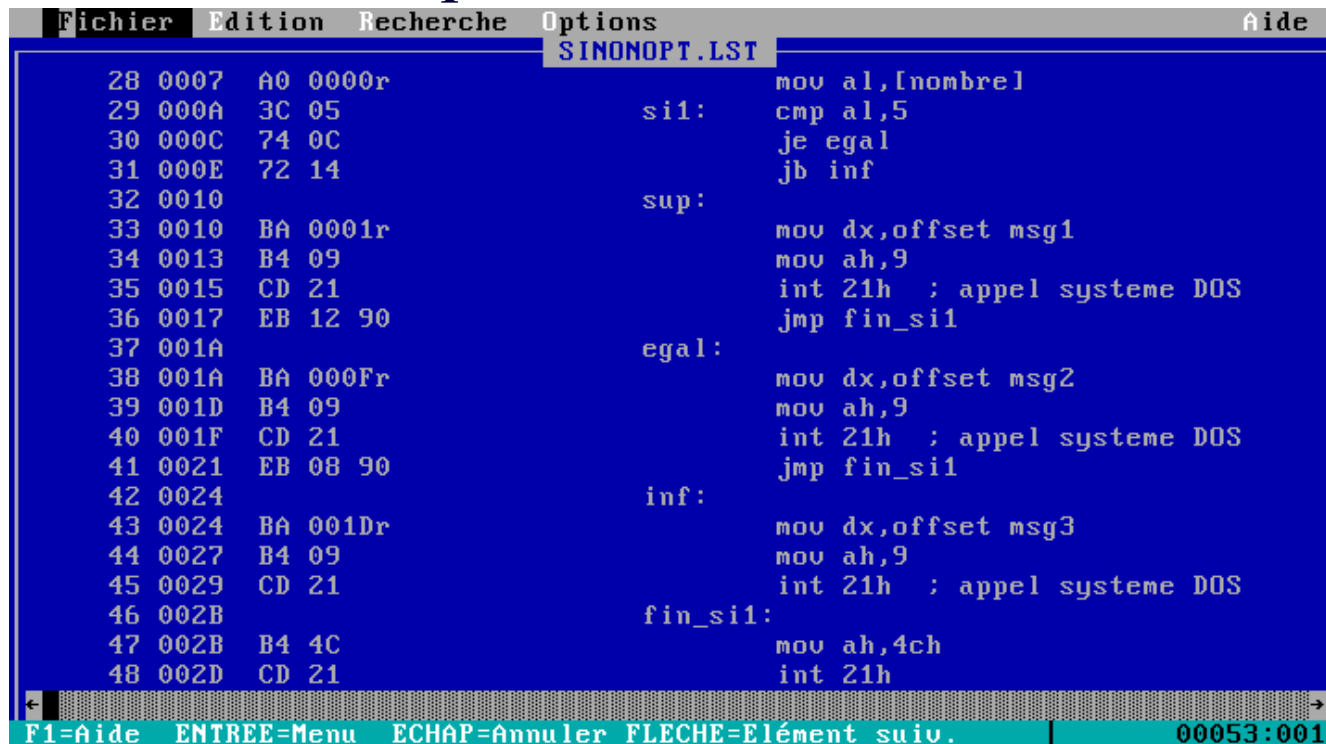
□ Le « si_alors_sinon »

```
Fichier Edition Recherche Options Aide
SI_SINON.LST
28 0007 A0 0000r      mov al,[nombre]
29 000A 3C 05        si1:   cmp al,5
30 000C 74 0C          je sinon_si2
31 000E 72 14          jnb alors1
32 0010
33 0010 BA 0001r      alors1: mov dx,offset msg1
34 0013 B4 09          mov ah,9
35 0015 CD 21          int 21h ; appel systeme DOS
36 0017 EB 12 90          jmp fin_si1
37 001A
38 001A BA 000Fr      sinon_si2: mov dx,offset msg2
39 001D B4 09          mov ah,9
40 001F CD 21          int 21h ; appel systeme DOS
41 0021 EB 08 90          jmp fin_si1
42 0024
43 0024 BA 001Dr      alors2: mov dx,offset msg3
44 0027 B4 09          mov ah,9
45 0029 CD 21          int 21h ; appel systeme DOS
46 002B
47 002B B4 4C          fin_si1: mov ah,4ch
48 002D CD 21          int 21h

F1=Aide  ENTREE=Menu  ECHAP=Annuler  FLECHE=Elément suiv.  00033:001
```

Optimisation possible pour les « experts »

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



```
Fichier Edition Recherche Options Aide
SINONOPT.LST
28 0007 A0 0000r      mov al,[nombre]
29 000A 3C 05          si1:  cmp al,5
30 000C 74 0C          je egal
31 000E 72 14          jb inf
32 0010              sup:
33 0010 BA 0001r      mov dx,offset msg1
34 0013 B4 09          mov ah,9
35 0015 CD 21          int 21h ; appel systeme DOS
36 0017 EB 12 90      jmp fin_si1
37 001A              egal:
38 001A BA 000Fr      mov dx,offset msg2
39 001D B4 09          mov ah,9
40 001F CD 21          int 21h ; appel systeme DOS
41 0021 EB 08 90      jmp fin_si1
42 0024              inf:
43 0024 BA 001Dr      mov dx,offset msg3
44 0027 B4 09          mov ah,9
45 0029 CD 21          int 21h ; appel systeme DOS
46 002B              fin_si1:
47 002B B4 4C          mov ah,4ch
48 002D CD 21          int 21h
F1=Aide ENTREE=Menu ECHAP=Annuler FLECHE=Elément suiv. 00053:001
```

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) $\in [-129, +127]$ octets

Intel

THE 8086/8088 ARCHITECTURE AND INSTRUCTIONS

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

```
Fichier  Edition  Recherche  Options  Aide
SI_SINON.LST
28 0007  A0 0000r      mov al,[nombre]
29 000A  3C 05          si1:  cmp al,5
30 000C  74 0C          je sinon_si2
31 000E  72 14          jb alors2
32 0010  +
33 0010  BA 0001r      alors1: mov dx,offset msg1
34 0013  B4 09          mov ah,9
35 0015  CD 21          int 21h ; appel systeme DOS
36 0017  EB 12 90      jmp fin_si1
37 001A
38 001A  BA 000Fr      sinon_si2: mov dx,offset msg2
39 001D  B4 09          mov ah,9
40 001F  CD 21          int 21h ; appel systeme DOS
41 0021  EB 08 90      jmp fin_si1
42 0024
= 43 0024  BA 001Dr      alors2: mov dx,offset msg3
44 0027  B4 09          mov ah,9
45 0029  CD 21          int 21h ; appel systeme DOS
46 002B
47 002B  B4 4C          fin_si1: mov ah,4ch
48 002D  CD 21          int 21h

F1=Aide  ENTREE=Menu  ECHAP=Annuler  FLECHE=Elément suiv.  00033:001
```

Calcul de l'adresse du saut inconditionnel

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

Intel THE 8086/8088 ARCHITECTURE AND INSTRUCTIONS

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



```
Fichier  Edition  Recherche  Options  Aide
SI_SINON.LST
28 0007  A0 0000r  mov al,[nombre]
29 000A  3C 05      si1:  cmp al,5
30 000C  74 0C      je sinon_si2
31 000E  72 14      jb alors2
32 0010
33 0010  BA 0001r  alors1:  mov dx,offset msg1
34 0013  B4 09      mov ah,9
35 0015  CD 21      int 21h ; appel systeme DOS
36 0017  EB 12 90  jmp fin_si1
37 001A
38 001A  - 1 octet  sinon_si2:  mov dx,offset msg2
39 001D  B4 09      mov ah,9
40 001F  CD 21      int 21h ; appel systeme DOS
41 0021  EB 08 90  jmp fin_si1
42 0024
43 0024  BA 001Dr  mov ah,9
44 0027  B4 09      int 21h ; appel systeme DOS
45 0029  CD 21
46 002B
47 002B  B4 4C      fin_si1:  mov ah,4ch
48 002D  CD 21      int 21h

F1=Aide  ENTREE=Menu  ECHAP=Annuler  FLECHE=Elément suiv.  00033:001
```

Annotations:

- Red circle around `002B` in the instruction `jmp fin_si1`.
- Light blue box: `- 1 octet`
- Red arrow pointing to the instruction with text: `nop !`
- Light blue box: `pour un problème d'alignement (adresse paire)`

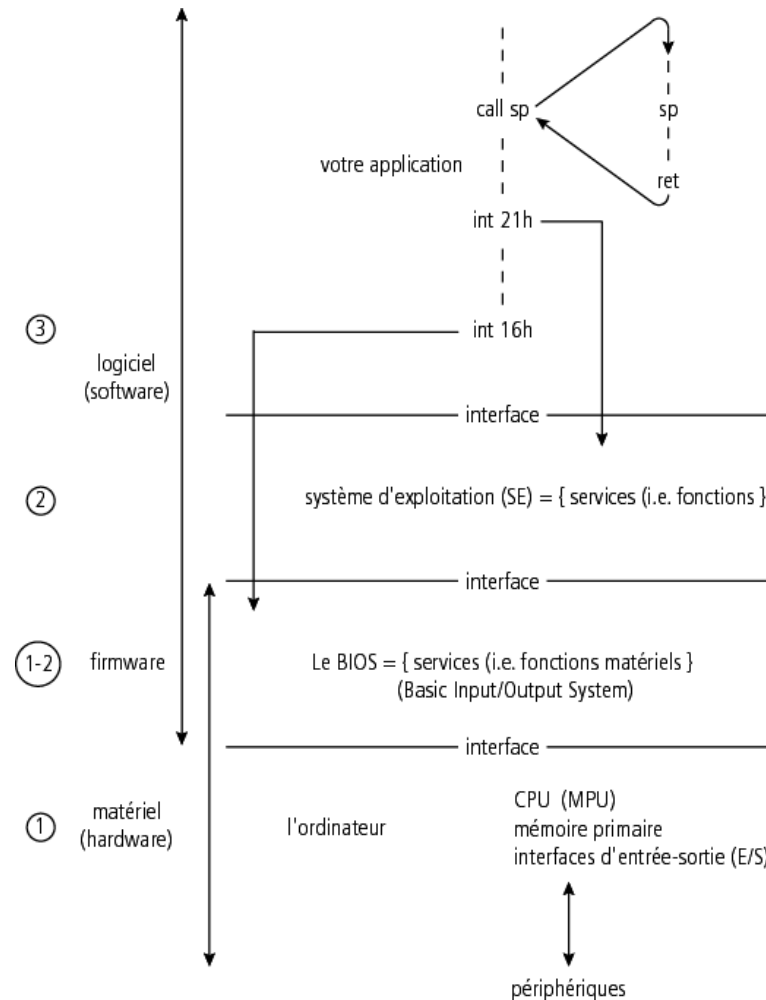
Pour se préparer au DST

- 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
ja suite

sub al,2
jnc suite

erreur:

code pour dépassement

suite:

Instructions conditionnelles

□ Parfois des simplifications

```
dec cx  
cmp cx,0  
jne repeter
```




```
dec cx  
jnz repeter
```



```
dec cx  
instructions modifiant les drapeaux  
jcz repeter
```

Structures de contrôle

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

	si var >		si1: cmp ax,5
alors			jbe sinon
	bloc 1		alors1: bloc 1
sinon			jmp fin_si1
	bloc 2		sinon1: bloc 2
fin_si			fin_si1:

Autre exemple

faire pour i = 1 à 5

bloc

fin_faire



```
faire:  mov cx,5  
        jcxz fin_faire
```

bloc

```
        dec cx  
        jmp faire
```

```
fin_faire:
```