ARCHITECTURE DES ORDINATEURS

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' *(IV)*

TRANSFERT de CONTROLE

Biblio ::

[1] - « *Assembly Langage for INTEL-based computers*»

[Kip R. IRVINE] – Ed. Prentice Hall, 1999 – ISBN: 0-13-660390-4.

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

Définition:

Toute RUPTURE d'une SEQUENCE d'instructions au profit de (engendrant) l'exécution d'une autre séquence, de manière **réversible** (bidirectionnelle) ou **irréversible** (monodirectionnelle).

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

EXEMPLE / ILLUSTRATION ::

'C':

```
int main()
{
int a1, a2,s;
int somme(int a, int b)
  {
    . . .
  Return a+b;
  }
s=somme(a1,a2);
}
```

'ASM x86':

```
RETOUR: AND AX,1
        SUB AX, 1
        JZ Cas_Impair
        JMP Cas_Pair
        . . .
Cas_Impair: <séq 1>
        . . .
Cas_Pair: <séq 2>
        . . .
        JNZ RETOUR
```

'ASM x86':

```
; exemple / illustration
; = 'Hello World '
SUB AX, 1
. . .
int 21h
. . .
```

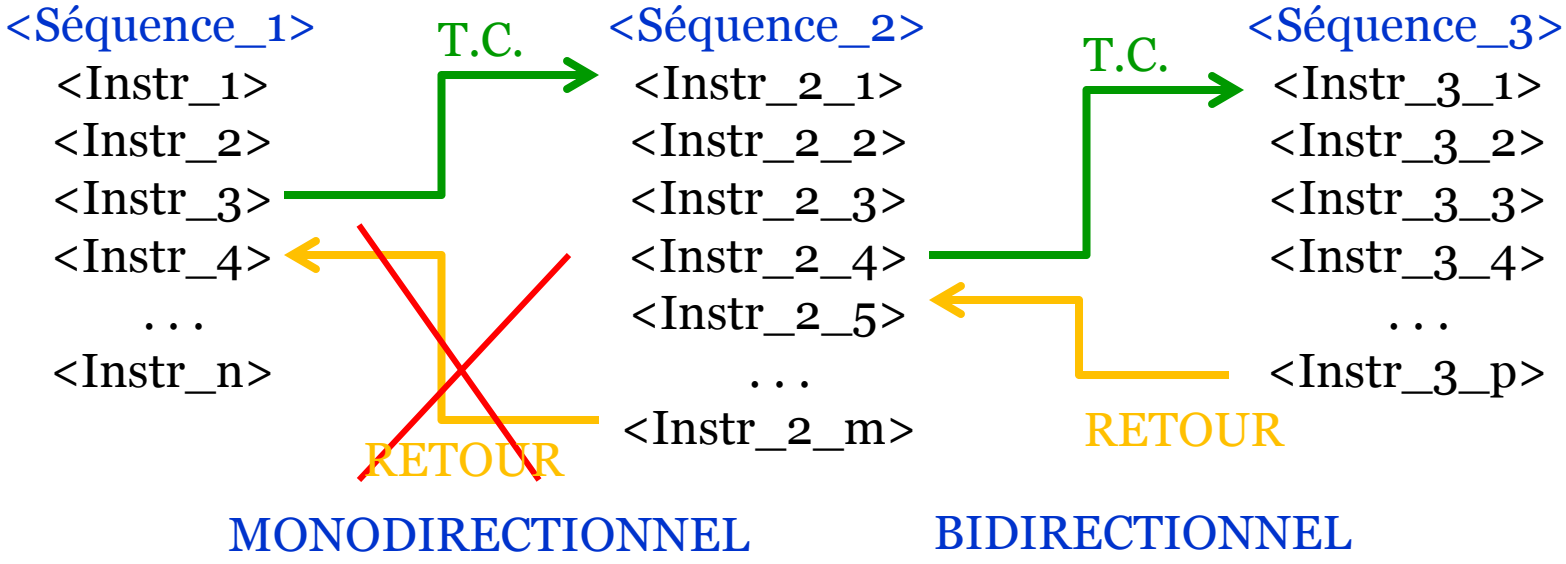
T.C.

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

Définition:

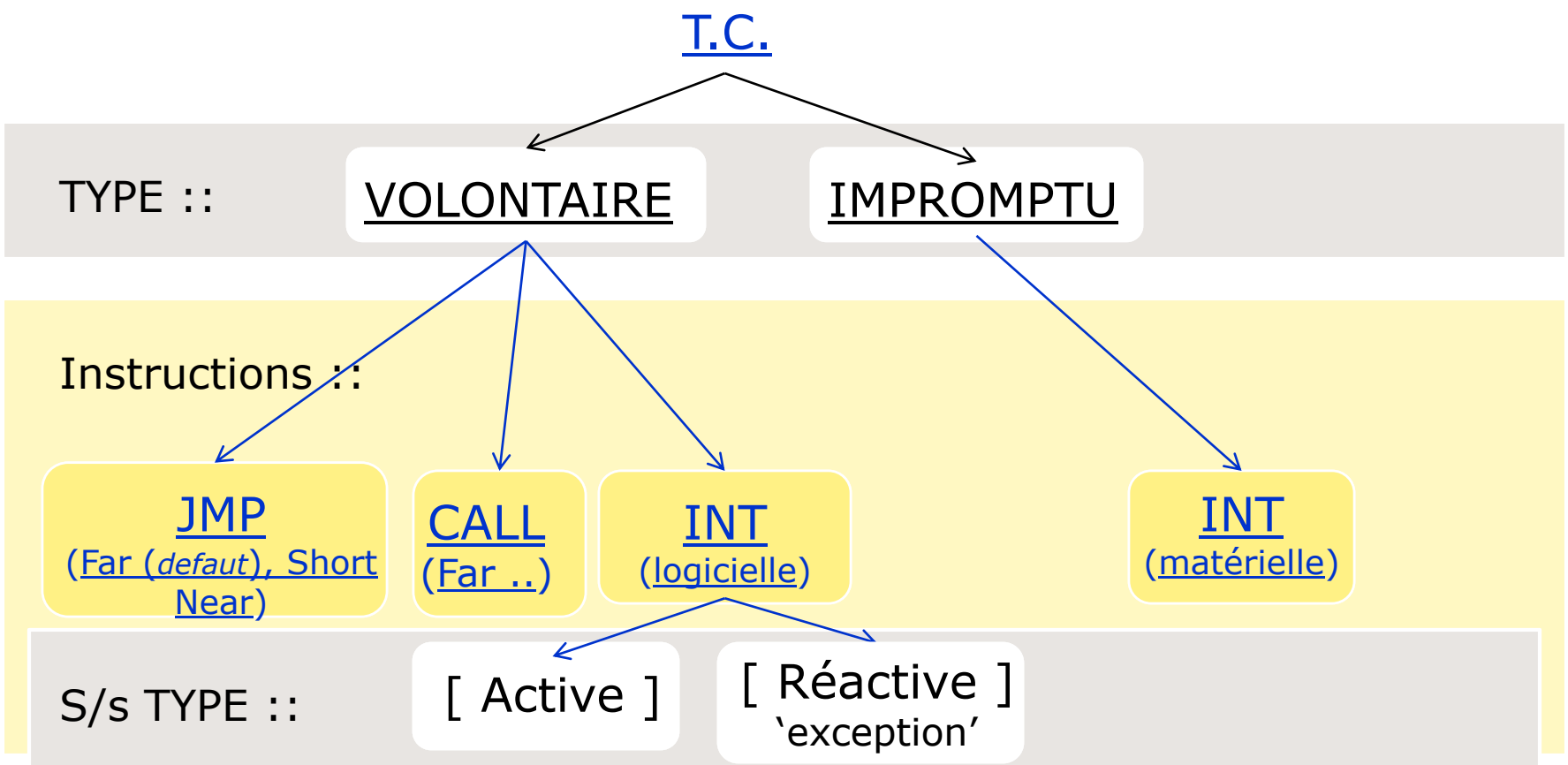
Toute RUPTURE d'une SEQUENCE d'instructions au profit de (engendrant) l'exécution d'une autre séquence, de manière **réversible** (bidirectionnelle) ou **irréversible** (monodirectionnelle).



(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE (en ASM)

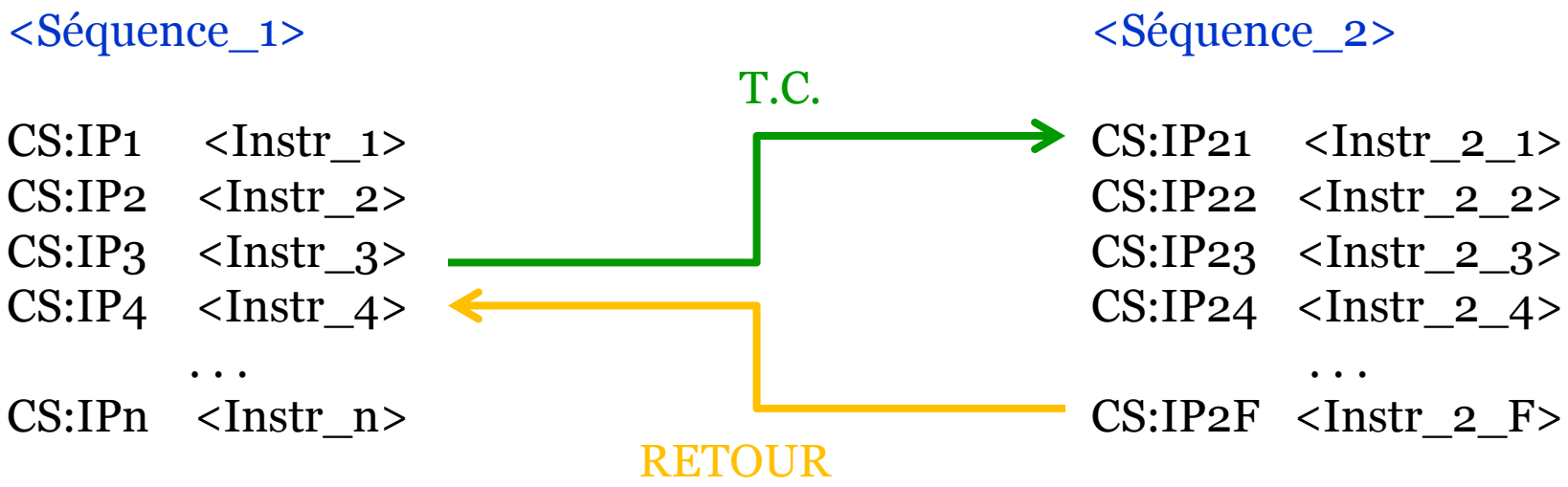
TYPOLOGIE & CLASSIFICATION ::



(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

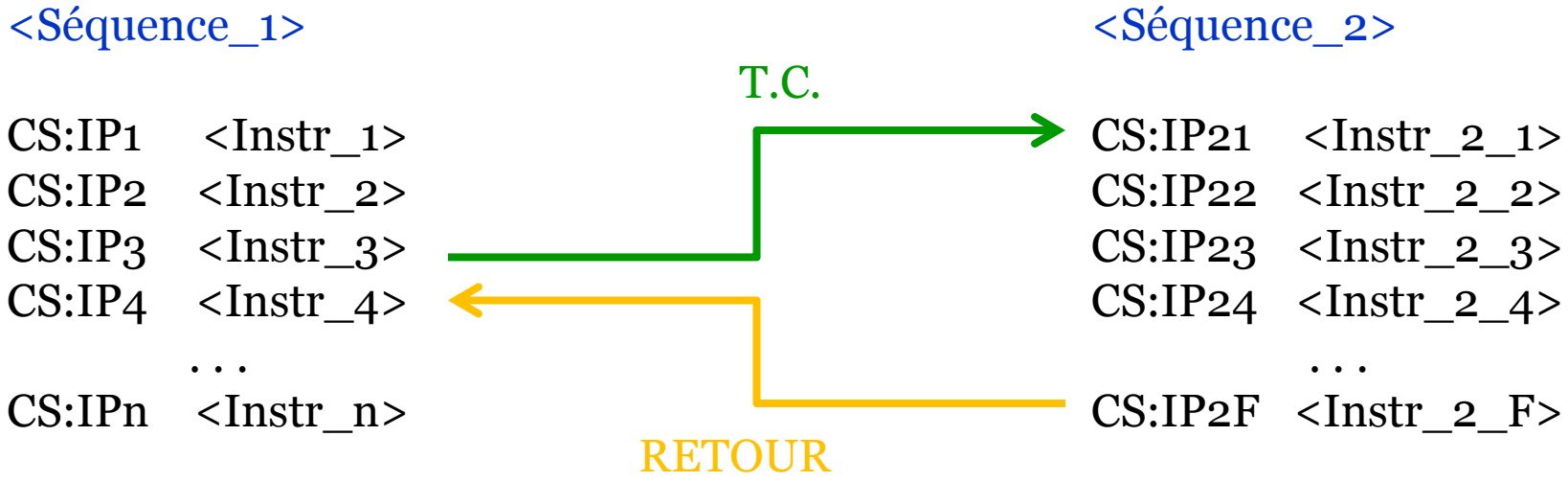
MECANISME DE MISE EN OEUVRE



(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

MECANISME DE MISE EN OEUVRE



1. Incrémentation (CO) : IP3 → IP4
2. SAVE: IP4 → PILE (CS ??)
3. Chargement: IP21 → Reg (IP)
   (exécution <séq 2>)
4. Après 'fin_séquence2':Dépilement (IP4): PILE → Reg (IP)
   (exéc. <séq 1 // Instr_4, Instr_5, .. suite>

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

ILLUSTRATION :: **JMP** & **JMP SHORT**

'JMP' **avec** et **sans** condition

; application d'un masque (valeur =1) pour identification du bit (b_0), suivi de
; saut conditionnel puis inconditionnel.

```
                        mov ax, [1000 h]
                        and ax, 1
                        sub ax,1
                        JZ TRAITE_IMPAIR          ; conditionnel
                        JMP TRAITE_PAIR           ; inconditionnel
                                ; peut être remplacée indifféremment par . . .
                                ; JMP SHORT  TRAITE_PAIR
TRAITE_IMPAIR:          <seq_instr_1>
                        . . .
TRAITE_PAIR:            <seq_Instr_2>
                        . . .
```

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

ILLUSTRATION :: **JMP NEAR** & **JMP FAR**

; branchement 'JMP' NEAR ou FAR similaire à 'JMP' SHORT
; aux différences suivantes près:

- « JMP Label1 » ou « JMP **short** ptr Label1 » :: branchement à Label1 qui se trouve à une adresse située entre **-128** et **+127** octets par rapport à l'adresse de l'instruction d'appel 'JMP Label1' => (seul) IP est incrémentée ou décrémentée en conséquence, d'une valeur « 8 bits signée »;
- « JMP **near** Label2 » :: branchement à Label2 qui se trouve à une adresse située au-delà de **-128** et **+128** octets (et inférieure à 2^{16} octets) par rapport à l'adresse de l'instruction d'appel 'JMP short Label2', mais en restant dans le même segment => (seul) IP est incrémentée ou décrémentée en conséquence , d'une valeur « 16 bits signée »;
- « JMP **FAR** ptr Label3 » :: branchement à Label3 qui se trouve à une adresse située au-delà de 2^{16} octets par rapport à l'adresse de l'instruction d'appel 'JMP FAR Label3', donc en se branchant à un autre segment => CS et IP de l'instruction en Label3 est chargée dans CS:IP du CPU.

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

ILLUSTRATION :: **CALL**

'CALL' **procédure** (ou ROUTINE)

```
        ; procédure 'SUM' avec appel depuis le 'main', suivi de plusieurs appels
        ; récursifs depuis la procédure elle même.
                main proc                       ; début 'main'
                        mov cx, 5
                        mov ax, 0
                        call sum                ; appel à 'procédure = sum'
Etiq_L1:                mov ax, 4C00h
                        int 21h
                main endp               ; fin 'main'
                sum proc                        ; début 'procédure = sum'
                        or cx, cx
                        jz Etiq_L2
                        add ax, cx
                        dec cx
                        call sum                ; appel à 'procédure = sum'
Etiq_L2:                ret
                sum endp                        ; fin 'procédure = sum'
```

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

```
; exemple / illustration = 'Hello World '
title hello World Program              (hello.asm)

.model small
.stack 100h
.data
message db "hello, world !",0dh,0ah, '$'

.code
main proc
        mov ax, @data
        mov ds, ax

        mov ah, 9
        mov dx, offset message
        int 21h

        mov ax, 4C00h
        int 21h
main endp
end main
```

NB: **[1] '29 Ko'** en **ASM** *vs* **'562 octets'** en **C** !!

(9)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(IV)***

TRANSFERT de CONTRÔLE

ILLUSTRATION :: **INT**

'INT' {ROUTINE}

À suivre .. cours 11 . . .