ARCHITECTURE DES ORDINATEURS

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

' TRAITEMENT CONDITIONNEL '

Ou quels sont les 'outils architecturaux du CPU:x86' pour implémenter des ' TRAITEMENTS CONDITIONNELS ' ?

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

RAPPEL:

Tout TRAITEMENT (ou '*ensemble d'instructions*') assujetti / asservi à l'état VRAI (ou FAUX) d'une PROPOSITION dite 'CONDITION'

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Corollaire:
Toute CONDITION peut toujours être ramenée à l'expression d'une égalité / inégalité.

ALGO:

```
IF (op1=op2)  THEN
        <Inst1>
        <Inst2>
        . . .
ELSE
ENDIF
```

'C':

```
IF (op1==op2)
        {
        <Inst1>
        <Inst2>
        . . .
        }
ELSE {. . .}
```

'ASM x86':

```
        CMP  op1,op2
        JE Egal
        JMP Autre
Egal:   Inst1
        Inst2
        . . .
Autre : <Suite>
```

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

L'expression de l'EGALITE est assimilée à une COMPARAISON

ALGO:

```
IF (op1=op2)  THEN
        <Inst1>
        <Inst2>
        . . .
ELSE
ENDIF
```

'C':

```
IF (op1==op2)
        {
        <Inst1>
        <Inst2>
        . . .
        }
ELSE {. . .}
```

'ASM x86':

```
        CMP  op1,op2
        JE Egal
        JMP Autre
Egal:   Inst1
        Inst2
        . . .
Autre:  <Suite>
```

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

JE: **J**ump if **E**qual
(choix d'un traitment [conditionné par le résultat de CMP] débutant à l'étiquette 'EGAL')

ALGO:

```
IF (op1=op2)  THEN
        <Inst1>
        <Inst2>
        . . .
ELSE
ENDIF
```

'C':

```
IF (op1==op2)
        {
        <Inst1>
        <Inst2>
        . . .
        }
ELSE {. . .}
```

'ASM x86':

```
        CMP  op1,op2
        JE Egal
        JMP Autre
Egal:   Inst1
        Inst2
        . . .
Autre:  <Suite>
```

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

JE: **J**ump if **E**qual
(choix d'un traitment [conditionné par le résultat de CMP] débutant à l'étiquette 'EGAL')

L'état '**E**QUAL' est décisif pour l'exécution de la séquence <Inst1, …>: il dépend lui même de **l'état** généré par '**CMP**', **état** inscrit dans le **registre (IF)** (à venir)

ALGO:

```
IF (op1=op2)  THEN
        <Inst1>
        <Inst2>
        . . .
ELSE
ENDIF
```

'C':

```
IF (op1==op2)
        {
        <Inst1>
        <Inst2>
        . . .
        }
ELSE {. . .}
```

'ASM x86':

```
        CMP  op1,op2
        JE Egal
        JMP Autre
Egal:   Inst1
        Inst2
        . . .
Autre:  <Suite>
```

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Corollaire:
Toute INSTRUCTION génère un 'résultat principal' (AIM), indexé par d'éventuels informations (Etats/ States) secondaires.

Exemple 1:
Si AX= 00FFh

ADD AX, 1
=>
AX = 0100h
& déb (LSB -> MSB)

Exemple 2:
Si AX= FFFFh

ADD AX, 1
=>
AX = 0100h
& déb (LSB -> MSB)
& déb (MSB -> 'retenue')

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Corollaire:
Toute INSTRUCTION génère un 'résultat principal' (AIM), indexé par d'éventuels informations (Etats/ States) secondaires.

Exemple 1:
Si AX= 00FFh

ADD AX, 1
=>
AX = 0100h
& déb (LSB -> MSB)

Exemple 2:
Si AX= FFFFh

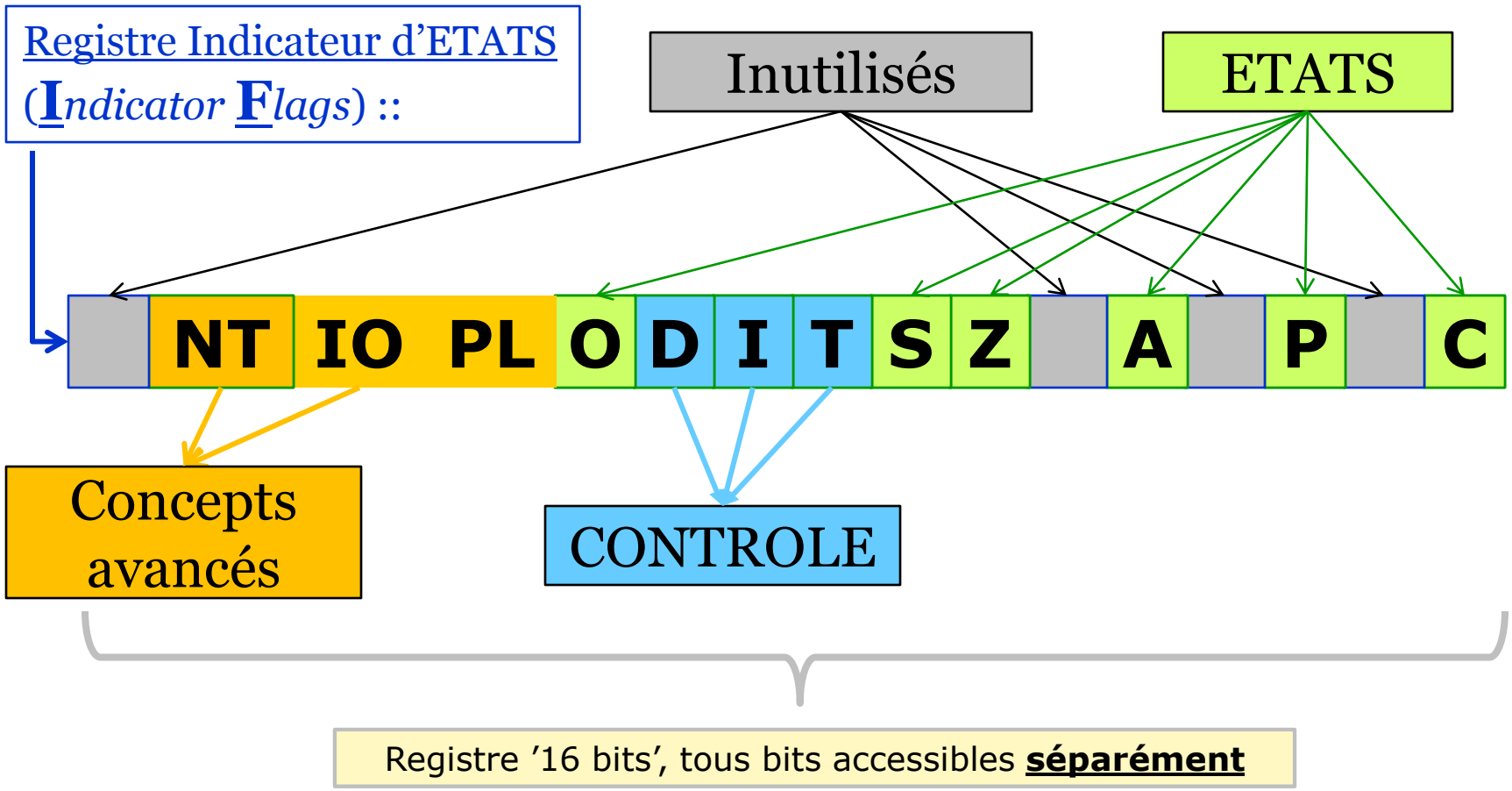
ADD AX, 1
=>
AX = 0100h
& déb (LSB -> MSB)
& déb (MSB -> 'retenue')

Ces (Etats/ States) secondaires sont '**immédiatement**' induits

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL :

Registre Indicateur d'ETATS



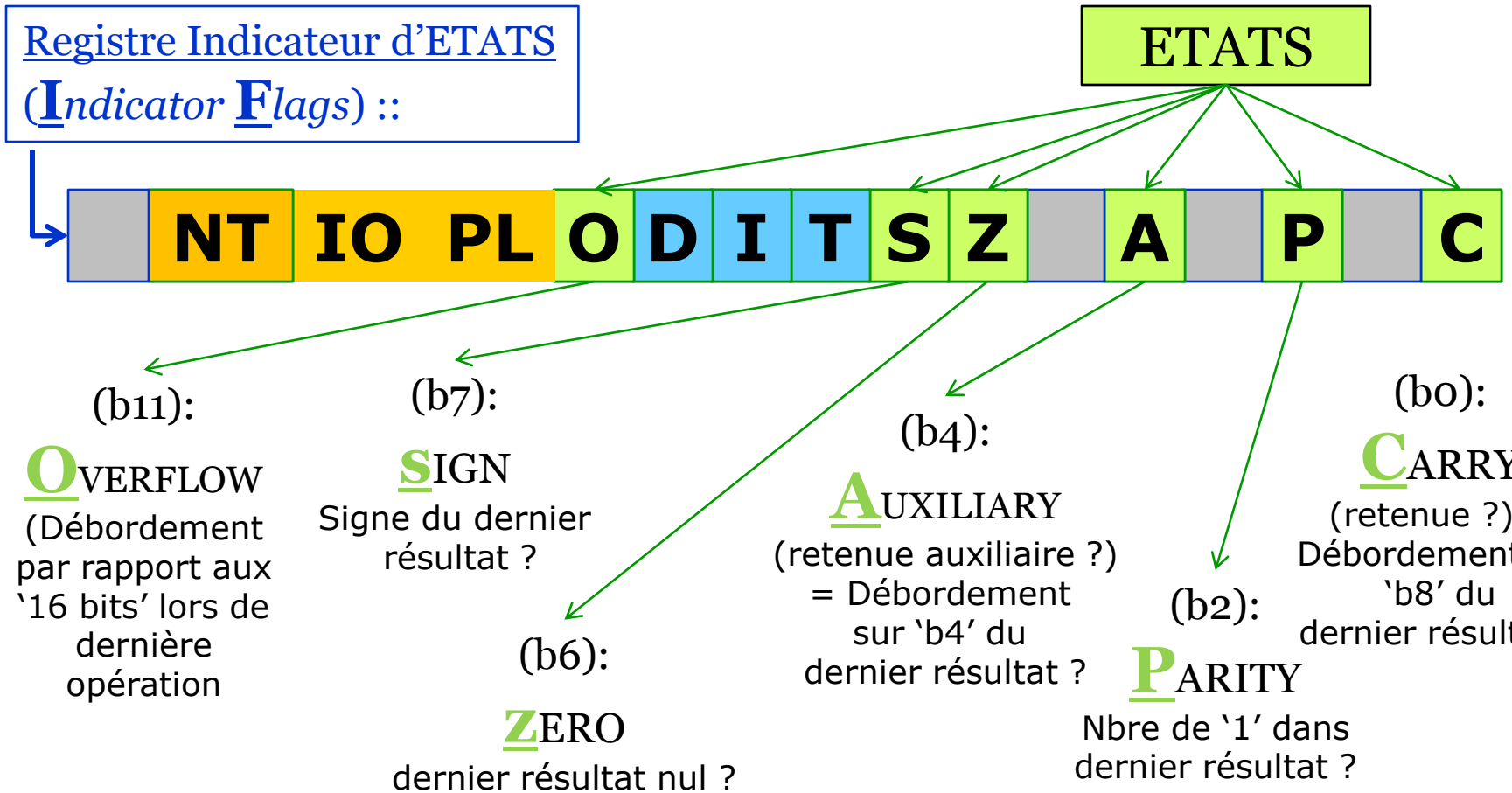
Registre '16 bits', tous bits accessibles **séparément**

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL :

Registre Indicateur d'ETATS

Registre Indicateur d'ETATS (*Indicator Flags*) ::



(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL :

Registre Indicateur d'ETATS

Registre Indicateur d'ETATS (*Indicator Flags*) ::

CONTROLE

(b10):

DIRECTION (Up / Down) :

Choix du Sens de comptage croissant (0)/ décroissant (1)

(b9):

INTERRUPT

Choix de l'autorisation (1) ou interdiction (0) des interruptions masquables

(b8):

TRAP

Choix de l'autorisation (1) ou interdiction (0) des interruptions générant le mode 'PAS à PAS'

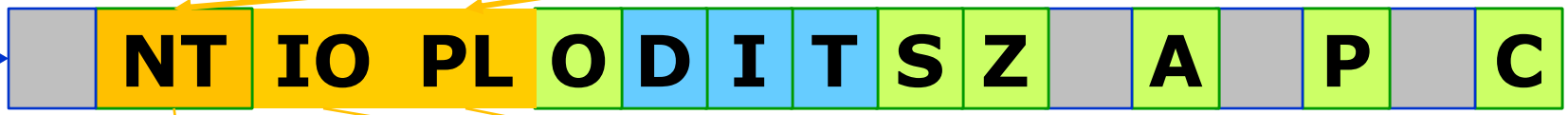
(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL :

Registre Indicateur d'ETATS

Registre Indicateur d'ETATS
(*Indicator Flags*) ::

Concepts avancés



(b14):

NESTED **T**ASK

Indique (NT=1) qu'une tache système a invoqué une autre *via* l'instruction 'CALL'

(b13-b12):

I/O PRIVILEGE **L**EVEL

Niveau de PL recquis pour une procédure faisant appel aux instruction E / S :: IN, OUT, INS …

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Registre Indicateur d'ETATS :: (*Indicator Flags*)

Exemple d'application

```
                        ; Analyse (Paire/Impaire) de la donnée logée à l'@ '1000h'
                        MOV AX, [1000H]
                        MOV DX, AX
                        AND DX, 1           ; (3)
                        SUB DX, 1           ; (4)
                        [JZ TRAITE_IMPAIR]
                        JMP TRAITE_PAIR
TRAITE_IMPAIR:          <Seq_Instr1>        Traitement Nbres. IMPAIRS
TRAITE_PAIR:            < Seq_Instr2>       Traitement Nbres. PAIRS_NUL
```

Branchement à 'TRAITE_IMPAIR' si (dernière/précédente) instruction 'SUB DX,1' a généré un résultat nul (équivaut à b0(DX)=1)

Les instructions 3 & 4 engendrent une équivalence entre la valeur du bit b0(DX) (désignant la parité de DX) et le bit 'Z' du registre (IF): on tire profit de cette équivalence pour notre traitement conditionnel

NB: avec 'JZ', la condition est **EXPLICITEMENT** rapportée au bit 'Z' du registre 'IF'

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Instructions de 'SAUT CONDITIONNEL' :: **Classement par TYPE**

GENERAL

- **Z** → Jump if **Z**ero / Jump if **N**ot **Z**ero (JZ / JNZ)
- **C** → Jump if **C**arry / Jump if **N**ot **C**arry (JC / JNC)
- **P** → Jump if **P**arity / Jump if **N**ot **P**arity (JP / JNP)
- □ → Jump if **E**qual / Jump if **N**ot **E**qual (JE / JNE)
- □ → Jump if **CX=Z**ero (JCXZ)

Nbres SIGNéS

- □ → Jump if **G**reater / Jump if **N**ot **G**reater (JG / JNG)
- □ → Jump if **G**reater or **E**qual / Jump if **N**ot **G**reater or **E**qual (JGE / JNGE)
- **S** → Jump if **S**ign / Jump if **N**ot **S**ign (JS / JNS)
- **O** → Jump if **O**verflow / Jump if **N**ot **O**verflow (JO / JNO)

Usage **EXPLICITE** du registre 'IF'

Usage **IMPLICITE** du registre 'IF'

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Instructions de 'SAUT CONDITIONNEL' :: **Classement par TYPE** (SUITE)

Nbres NON-SIGNéS

- **J**ump if **A**bove / **J**ump if **N**ot **A**bove (JA / JNA)
- **J**ump if **A**bove or **E**qual/ **J**ump if **N**ot **A**bove or **E**qual (JAE / JNAE)
- **J**ump if **B**elow / **J**ump if **N**ot **B**elow (JB / JNB)

Usage **IMPLICITE** du registre 'IF'

Attention: liste des 'jump' conditionnels non complète

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Instructions de 'SAUT CONDITIONNEL' :: Exemples d'application

Exemple 1:

```
                    ; Recherche du plus petit des nombres NON-SIGNéS
                                        ; small est un identifiant qcq

                    MOV small, AL       ; on suppose AL : le + petit
                    CMP small, BL       ; .. On compare avec BL
                    JBE L1              ; saut à 'L1' si AL = small 'antérieur' BL
                    MOV small, BL       ; sinon, BL devient small
L1 :                CMP small, CL       ; . . . puis compare avec CL
                    JBE L2              ; . . . etc
                    MOV small, CL
L2 :                <suite>             ;  suite avec CL plus petit des 3 nbres
                                        ; . . . non-signés
```

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Instructions de 'SAUT CONDITIONNEL' :: Exemples d'application

Exemple 2:

```
                    ; Recherche du plus petit des nombres SIGNéS
                                        ; small est un identifiant qcq

                    MOV small, AL       ; on suppose AL : le + petit
                    CMP small, BL       ; .. On compare avec BL
                    JNG L1              ; saut à 'L1' si AL = small '+ petit' que BL
                    MOV small, BL       ; sinon, BL devient small
L1 :                CMP small, CL       ; . . . puis compare avec CL
                    JNG L2              ; . . . etc
                    MOV small, CL
L2 :                <suite>             ;  suite avec CL plus petit des 3 nbres
                                        ; . . . signés
```

(8)- INTRODUCTION à l' ASSEMBLEUR 'x86' ***(III)***

TRAITEMENT CONDITIONNEL

Instructions de manipulation des bits de CONTROLE

BITS de CONTRÔLE

D → **S**e**T** **D**irection Flag / **CL**ear **D**irection Flag (**STD** / **CLD**)

I → **S**e**T** **I**nterrupt Flag / **CL**ear **I**nterrupr Flag (**STI** / **CLI**)

BITS d'ETATS

C →
- **S**e**T** **C**arry Flag / **CL**ear **C**arry Flag (**STC** / **CLC**)
- **Co**M**p**lement **C**arry Flag (**CMC**)

REGISTRE d'ETATS

- **L**oad **AH** with **F**lag / **S**tore **AH** to **F**lag (**LAHF** / **SAHF**)
- **PUSH F**lag / **POP F**lag (**PUSHF** / **POPF**)