INFORMATIQUE INDUSTRIELLE

0. Préambule

INFORMATIQUE INDUSTRIELLE :

INFORMATIQUE AVEC CONTRAINTE DE TEMPS

INFORMATIQUE AVEC PERIPHERIQUES D'E/S

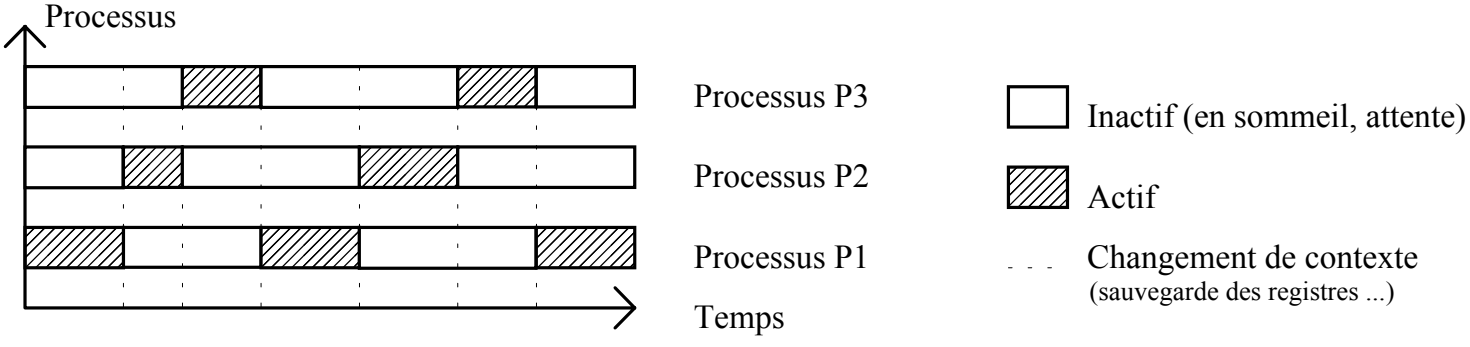
INFORMATIQUE AVEC CONTRAINTE DE TEMPS

Intervention sur le langage de programmation

Le langage assembleur.

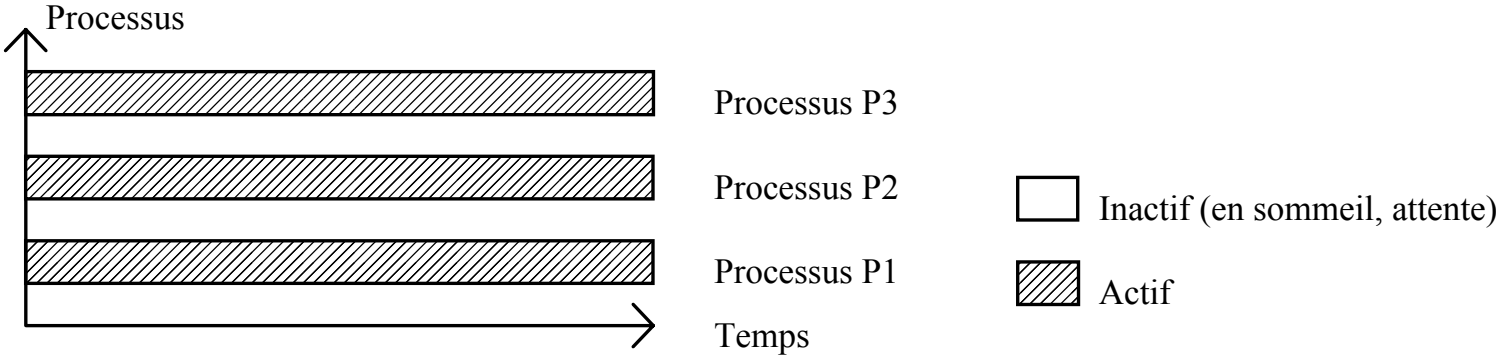
Intervention sur le système d'exploitation

Fonctionnement *multitâche*.

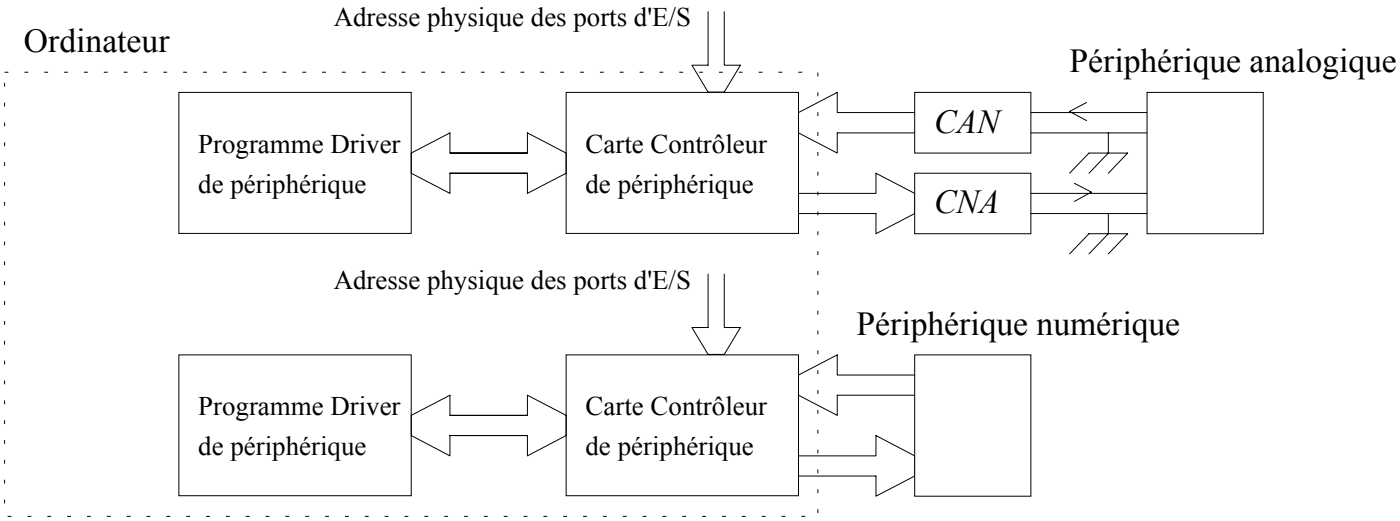


Intervention sur l'architecture de la machine

Machine parallèle.



INFORMATIQUE AVEC PERIPHERIQUES D'E/S



Plan du Cours

INFORMATIQUE INDUSTRIELLE (II) + REGULATION (RI)

COURS + TP

ASSEMBLEUR 80x86-Pentium

1. ARCHITECTURE DU 80x86-Pentium
2. SOUS-PROGRAMMES ASSEMBLEUR

SYSTEMES TEMPS REEL

3. SYSTEMES TEMPS RÉEL (TR)

PERIPHERIQUES : INTERFACES d'E/S / FILTRAGE

4. TUTORIAL LabWindows/CVI
5. LE PIO 8255 (Interface parallèle)
6. L'UART 8250 (Interface série)
7. FILTRAGE NUMERIQUE TEMPS DIFFERE

REGULATION

1. SIMULATION
2. CONTRÔLE d'un processus électrique
3. CONTRÔLE d'un processus électromécanique
4Annexe.CONTRÔLE d'un processus thermique

SYSTEMES A DSP - ROBOTIQUE (sous réserve)

. SYSTEMES à DSP
. ROBOTIQUE

PROJET

. PROJET

Contrôle des connaissances

Projet (II ou RI) + TP.

Projet

TP

1 si travail satisfaisant,
-1 sinon
0 si travail moyen.

Note globale TP = +2 si tous TPs OK.

Note d'Informatique Industrielle

Note Projet + Note globale de TP.

Bibliographie

[5] **H. Nussbaumer** « Informatique Industrielle II » *PPR*

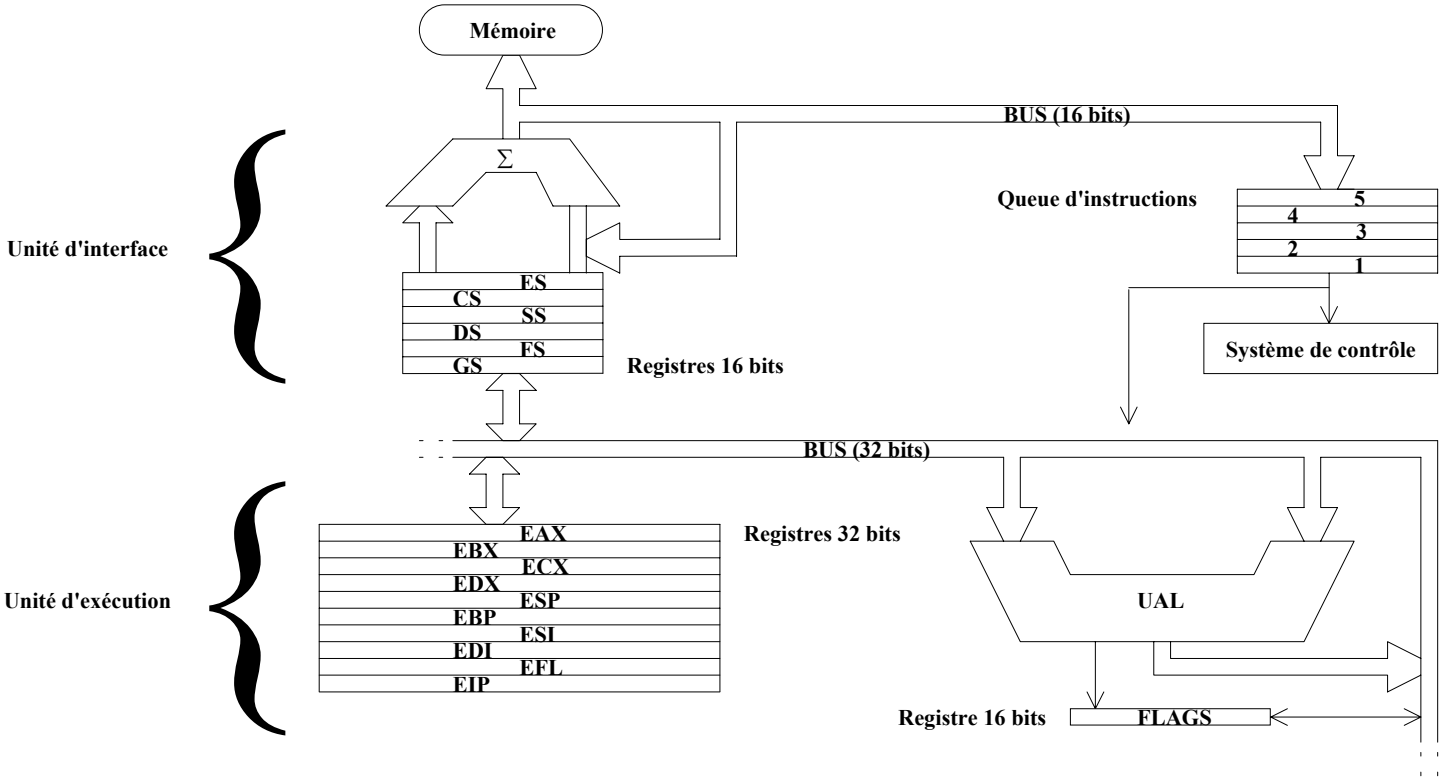
[8] **A. Schiper** « Programmation concurrente » *PPR*

[10] **D. Tschirhart** « Commande en Temps Réel » *Dunod*

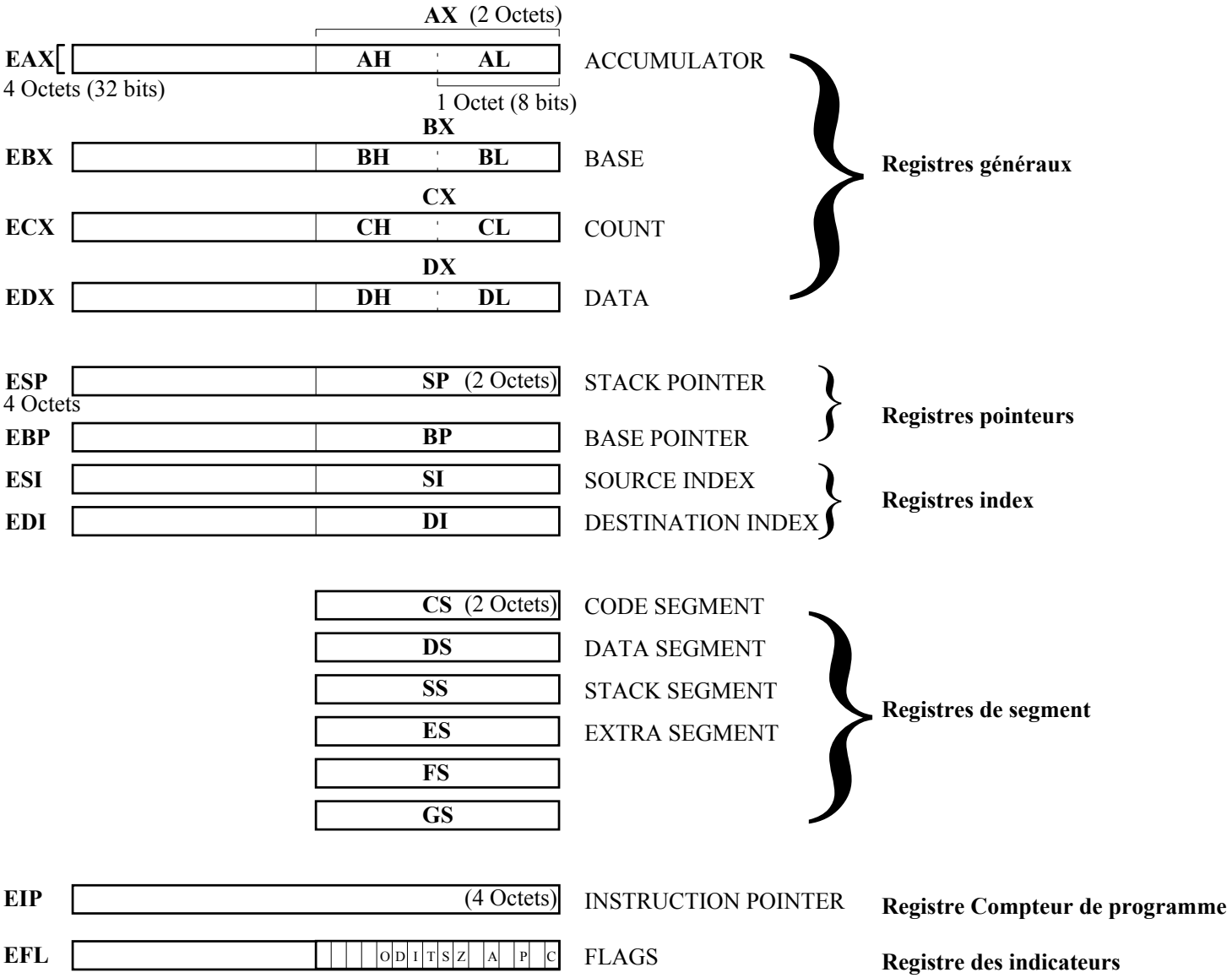
[11] **P. Wratil/R. Schmidt** « Contrôle mesure régulation sur PC » *Radio*

1. ASM 80x86-Pentium : Architecture

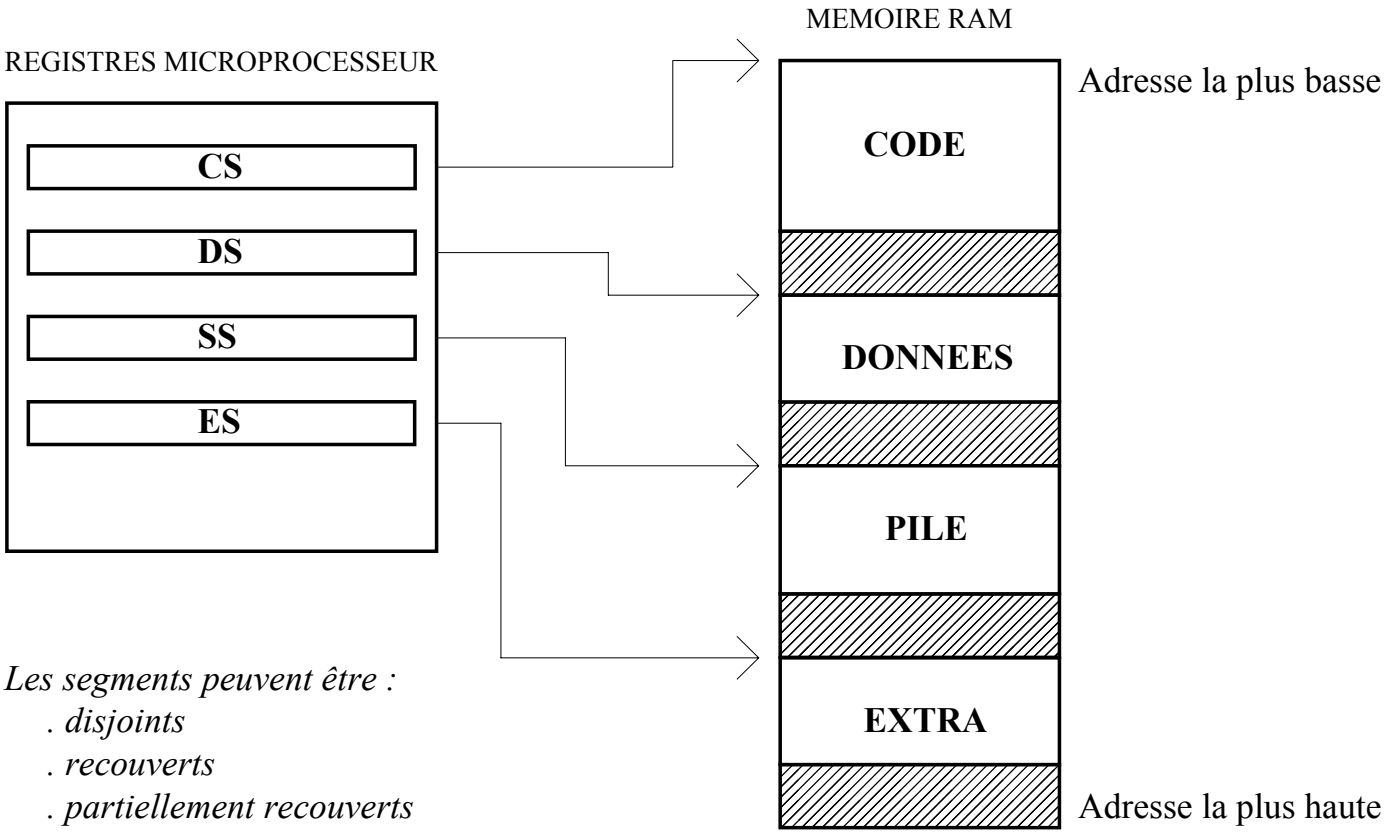
1. Architecture du 80x86-Pentium



2. Registres du 80x86-Pentium



3. La segmentation



Segmentation de la mémoire centrale (≡ RAM)

Adressage physique

Adresse physique sur bus 32 bits : 2 parties :

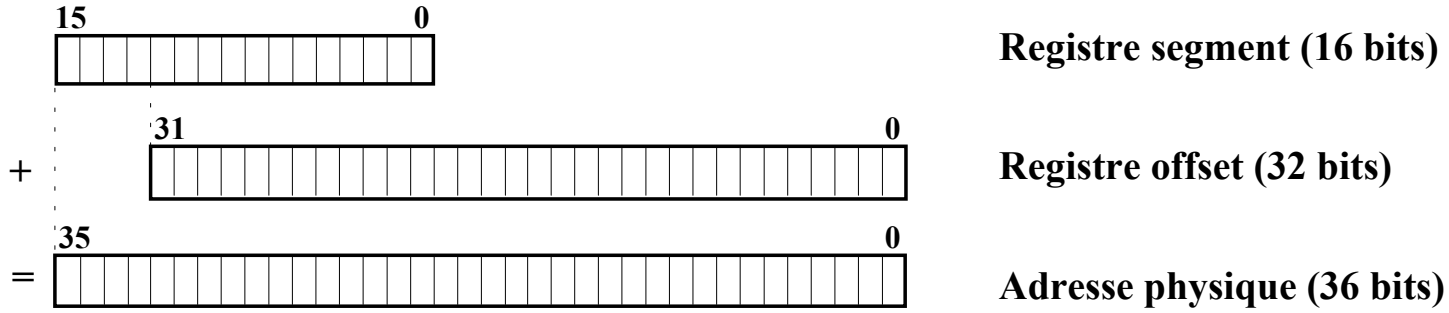
. une partie segment, notée SEGMENT sur 16 bits

. une partie offset par rapport au segment SEGMENT, notée OFFSET sur 32 bits.

Au total :

Adresse = SEGMENT:OFFSET

Calcul de l'adrese physique :



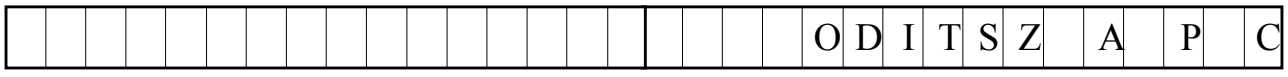
*Adresse = SEGMENT * 16 + OFFSET*

Modes d'adressage

Adresse effective	*Par rapport au segment*
MEMOIRE	DS
(EBP)	Interdit
(EBX)	DS
(ESI)	DS
(EDI)	DS
(EBP) + MEMOIRE	SS
(EBX) + MEMOIRE	DS
(ESI) + MEMOIRE	DS
(EDI) + MEMOIRE	DS
(EBP) + (ESI)	SS
(EBP) + (EDI)	SS
(EBX) + (ESI)	DS
(EBX) + (EDI)	DS
(EBP) + (ESI) + MEMOIRE	SS
(EBP) + (EDI) + MEMOIRE	SS
(EBX) + (ESI) + MEMOIRE	DS
(EBX) + (EDI) + MEMOIRE	DS
(EIP)	CS
(ESP)	SS
(EDI)	ES

4. Le registre des indicateurs (FLAGS)

(4 Octets)



FLAGS

5. Rangement mémoire

La mémoire est indexée *octet par octet*.

Ex. : int TOTO = 256 (Décimal) = **00 00 01 00** (Hexadécimal)

Adresse mémoire (offset) (indexée octet par octet) :
Donnée

00000000 :	00
00000001 :	01
00000002 :	00
00000003 :	00

6. Implantation de la pile en RAM. Squelette d'un programme assembleur sous Visual C++

. Pile (= LIFO)

.Ordres de manipulation de pile : PUSH (empilement)
et POP (dépilement).

.Le pointeur de pile (*Stack Pointer*) SP :
repère la position courante du sommet de la pile.

4 champs constituent une instruction assembleur :

1. Champ étiquette
2. Champ instruction
3. Champ opérande
4. Champ commentaire

Exemple :

Empilement de(AX), puis dépilement dans le registre BX.

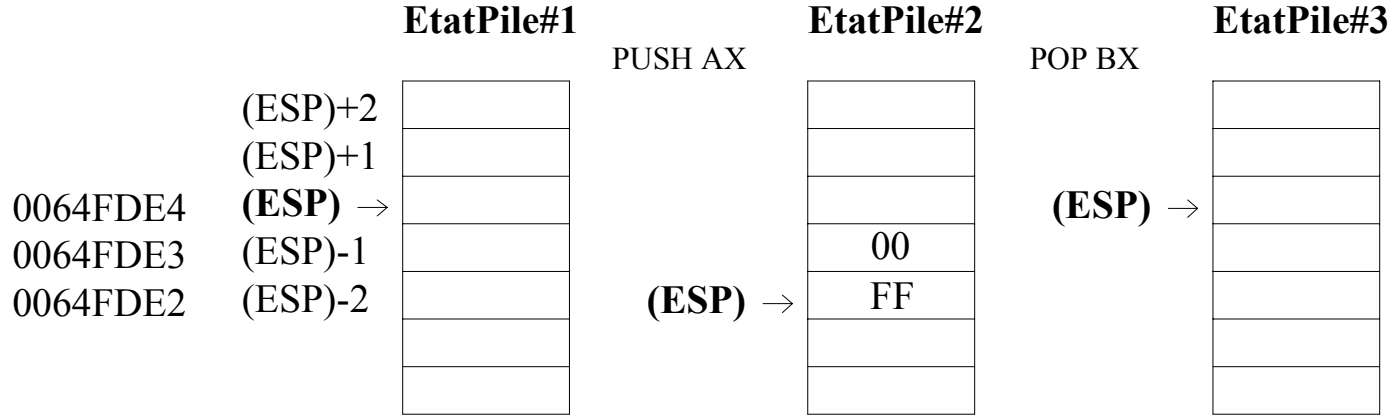
Supposons : (ESP) = 0064FDE4 avant manipulation de la pile

```
// Squelette d'un pgm assembleur pour Visual C++
//        Exemple de manipulation de la pile
#include <stdio.h>
void main()
{
// Partie C
```

Les déclarations des données de l'asm se font dans la partie C

```
short int Toto = 255;     // Toto = 255 Décimal = 00 FF Hexa
unsigned char Titi;
printf("\n Toto = % d", (int) Toto);
// Partie assembleur  Minuscules et majuscules indifférentes
__asm
 {
mov   ax, Toto    // Toto -> AX :   (AX) = 255 = 00 FF      EtatPile#1
push  ax          // Empilement de (AX)  :  (AX) ≡ (AX)     EtatPile#2
label: pop   bx   // Depilement dans BX :  (BX) = 00 FF     EtatPile#3
mov   Titi, bl    // (BL) -> Titi          :  Titi = 255 = FF
 }
// Partie C
 printf("\n Titi = % d\n", (int) Titi);
}
```

Etat de la pile :



Résultat de l'exécution Toto = 255 Titi = 255

7. Jeu d'instructions (instructions principales)

INSTRUCTIONS DE TRANSFERT

MOV
PUSH
POP
XCHG
XLAT
LEA
LDS
LES
IN
OUT
LAHF
SAHF
PUSHF
POPF

INSTRUCTIONS LOGIQUES

SHL
SAL
SHR
SAR
ROL
RCL
ROR
RCR
NOT
TEST
AND
OR
XOR
CMP
CBW
CWD

INSTRUCTIONS DE TEST et BRANCHEMENT

CALL
RET
JMP
J??
JN?
LOOP
LOOPE
LOOPNE
JCXZ
REP
REPE
REPNE

INSTRUCTIONS SYSTEME

INT
INTO
IRET
HLT
WAIT
ESC
LOCK
STC
STD
STI
CLC
CLD
CLI
CMC
NOP

INSTRUCTIONS ARITHMETIQUES

ADD
ADC
INC
AAA
DAA
SUB
SBB
DEC
AAS
DAS
DIV
IDIV
AAD
MUL
IMUL
AAM
NEG

SUFFIXES

A, G
E
B, L
C
Z
O
P
S

INSTRUCTIONS DE CHAÎNES

CMPS
SCAS
LODS
STOS
MOVS

9. Exemple de pgm : calcul du maximum d'un vecteur

```
#include <stdio.h>
char VECT[5];
char RES;
char Chaine;
int MaxElemVect;
void main (void)
{  int NbElem = 5;
VECT[0] = 2;
VECT[1] = 3;
VECT[2] = 1;
VECT[3] = 7;
VECT[4] = 0;
// Debut de la partie assembleur
__asm
{
        MOV   ECX, NbElem      ; (ECX)=NbElem
        DEC   ECX
        MOV   ESI, 0           ; i := 0
        MOV   AL, VECT[ESI]    ; (AL) = MAX = VECT[0]
        INC   ESI              ; i := 1
NEXT:   MOV   AH, VECT[ESI]    ; (AH) = VECT[i]
        CMP   AL, AH
        JGE   SUP              ; MAX >= VECT[i] ?
        MOV    AL, AH          ; MAX < VECT[i]     (AL) = VECT[i]
SUP:    INC   ESI              ; MAX >= VECT[i]      i = i + 1
        LOOP  NEXT             ; (ECX) = (ECX) - 1
        MOV    RES, AL         ; (AL) → RES
}  // Fin de la partie assembleur
MaxElemVect = RES;
printf ("\nElement maximum du vecteur = %d\n\n", MaxElemVect);
        }
```