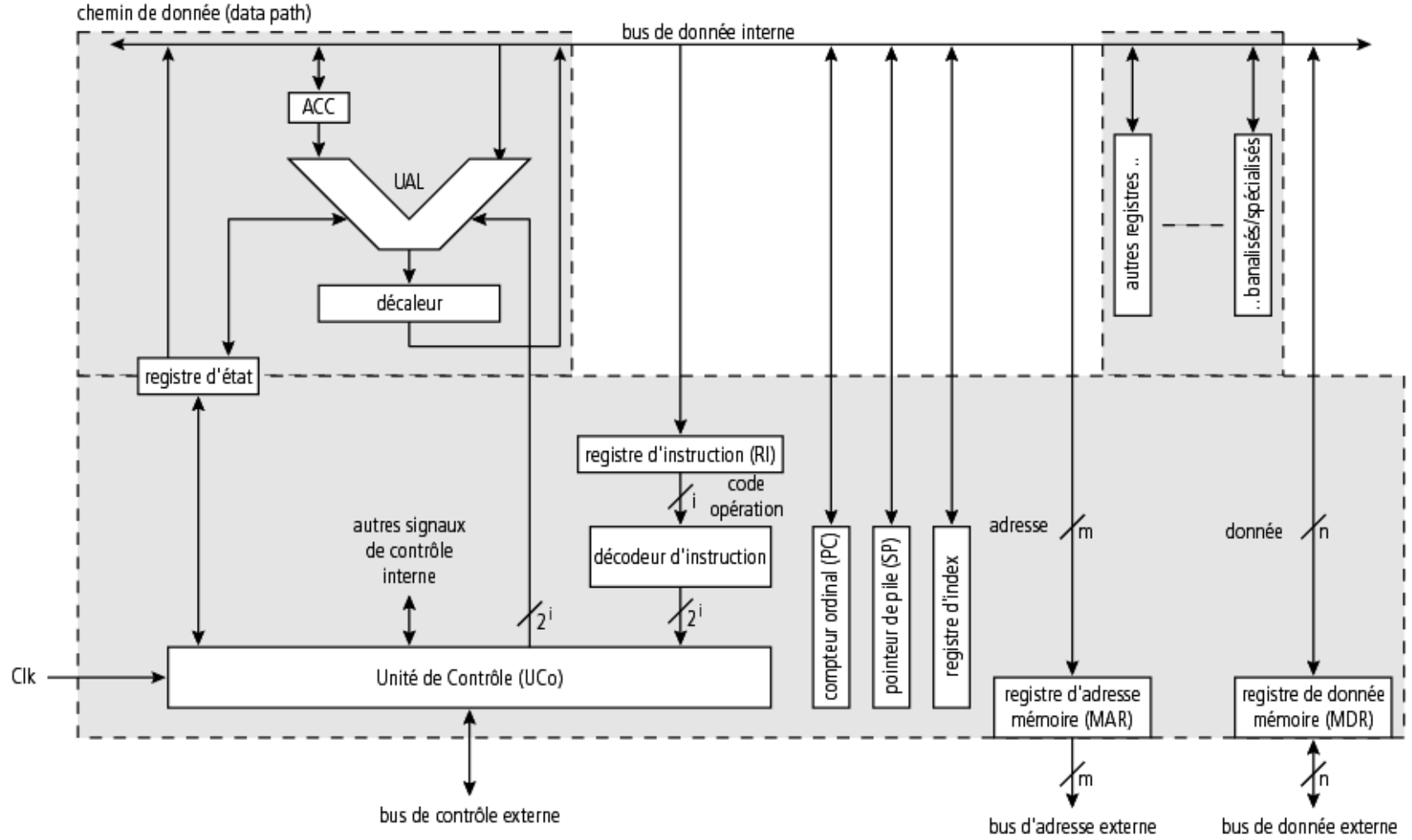
Architecture des ordinateurs

44 - Le langage d'assemblage

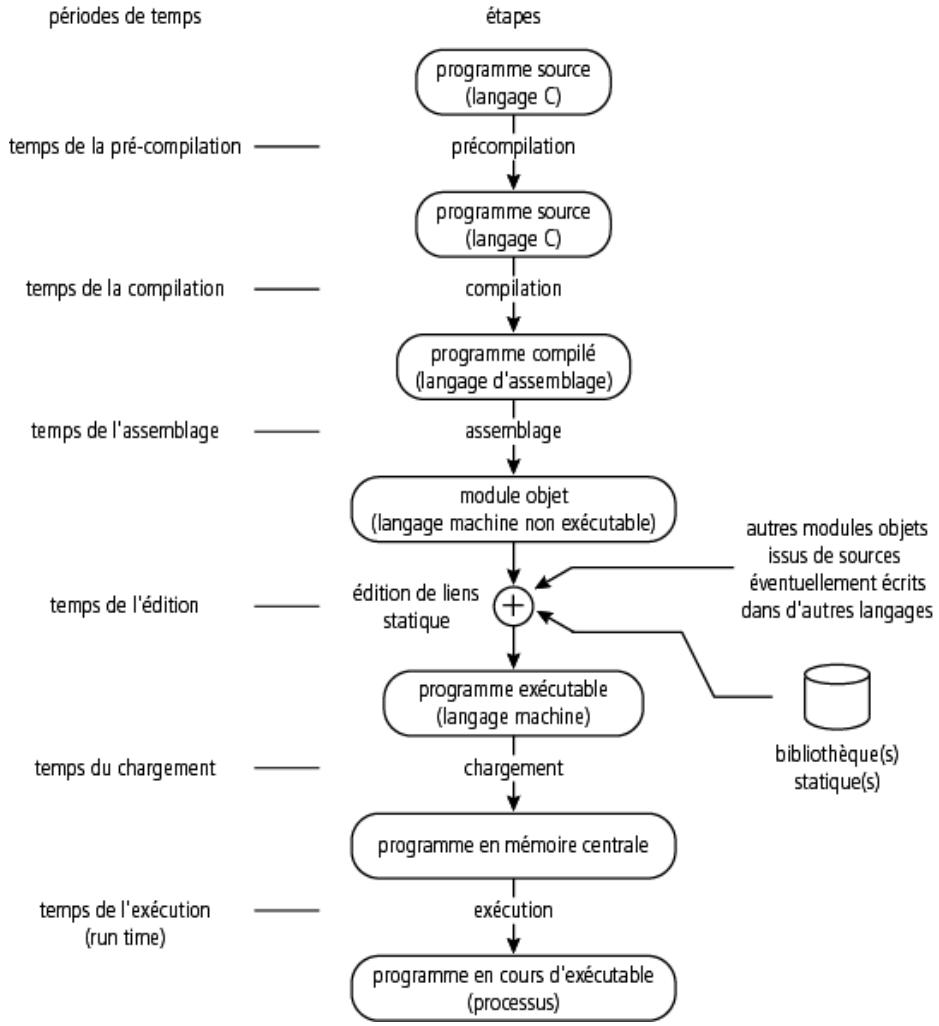
(*some assembly required !*)

Philippe Darche
IUT Paris Descartes

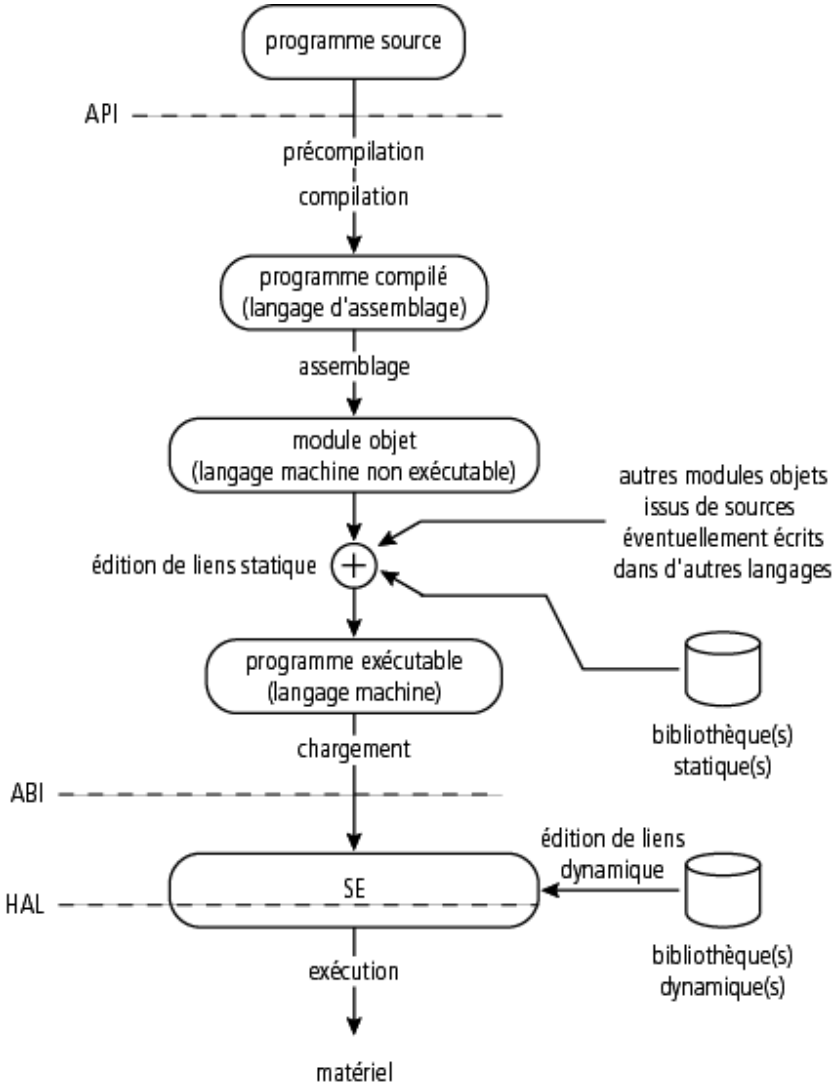
Une micro-architecture simple à bus de donnée unique (rappel)



Chaîne de développement logicielle



Bibliothèques statiques et dynamiques



Définition

- Le Langage d'Assemblage (LA) est le langage de programmation de base d'un microprocesseur
- Langage symbolique
  - comme pour un langage de haut niveau
  - mais attention pas de typage des données
    - sauf exception
- Origine : logiciel pour IBM 650 puis 704 (1956)

Définition

- Assembleur = outil informatique de conversion mnémonique → code machine
  - mnémonique = abrégé, abréviation d'une opération
  - en général destiné à un type de machine
- Exemples

mnémonique (opérande(s))		hexadécimal
mov ds,ax	→	8E D8
int 21h	→	CD 21

Pourquoi l'utiliser

- Gain en vitesse d'exécution :
  - système d'exploitation temps réel pour système embarqué (missile, etc.)
  - traitement du signal : image, son, etc.
- Gain en place mémoire → compacité du code
  - kit d'évaluation de µP → peu de mémoire
- Interfaces spécifiques non offertes par le langage
  - ajout de primitives langage évolué – systèmes d'exploitation
  - interface système d'exploitation - matériel
    - exemple : pilote (*driver*) d'un périphérique
- Intérêt pédagogique
  - connaissance fine des mécanismes de base du CPU et des langages de programmation
    - exemple 1 : passage des paramètres
    - exemple 2 : interruption

Quelques statistiques

Langage	taille en lignes de code	Effort de développement en mois/homme
d'assemblage	10 000	40
de macro-assemblage	6 666	28
C	5 000	22
Cobol	3 333	16
Pascal	2 500	13
Ada	2 222	12
Basic	2 000	10

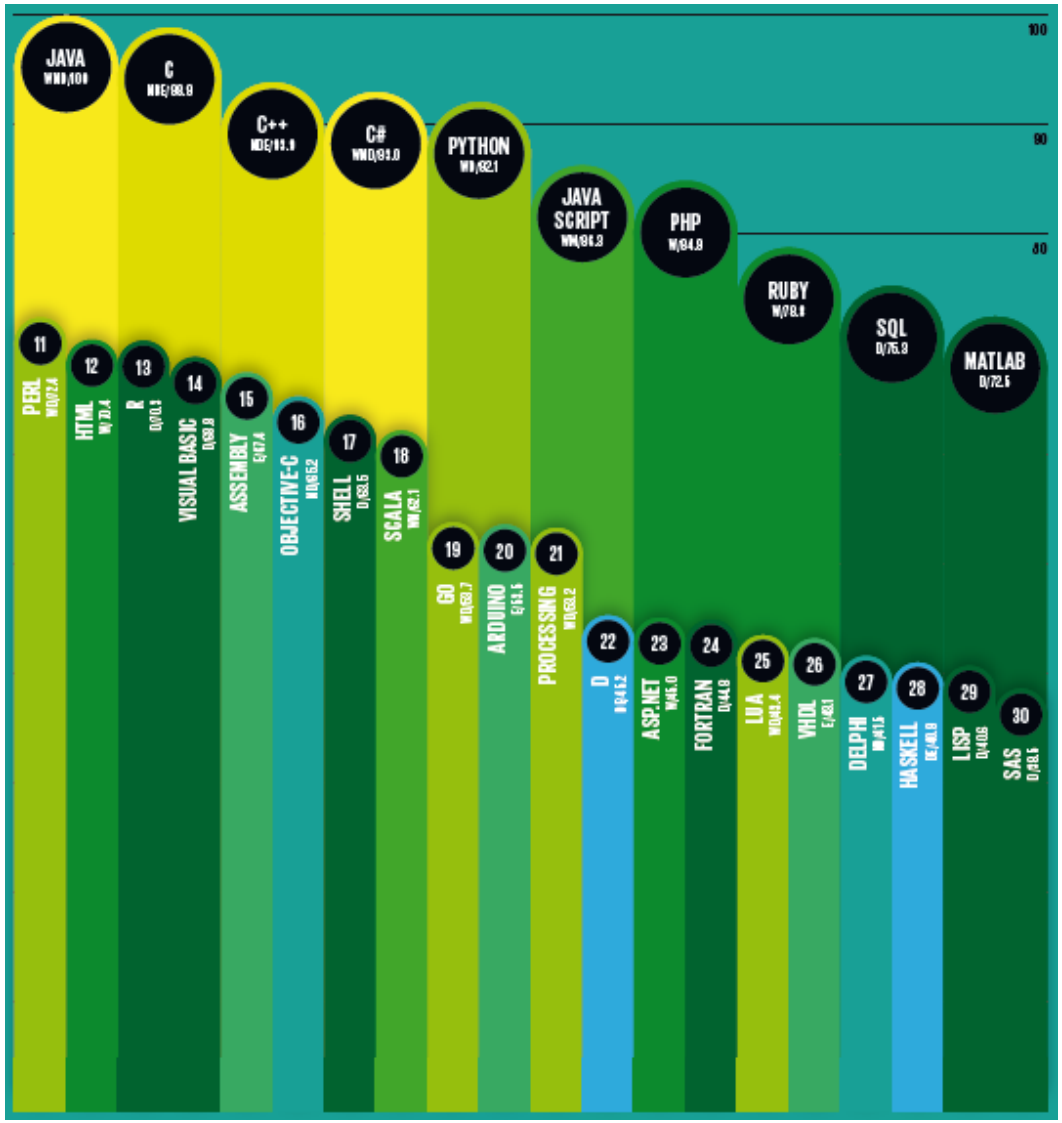
d'après Capers Jones (Programming Productivity 1986)

Raisons contraires

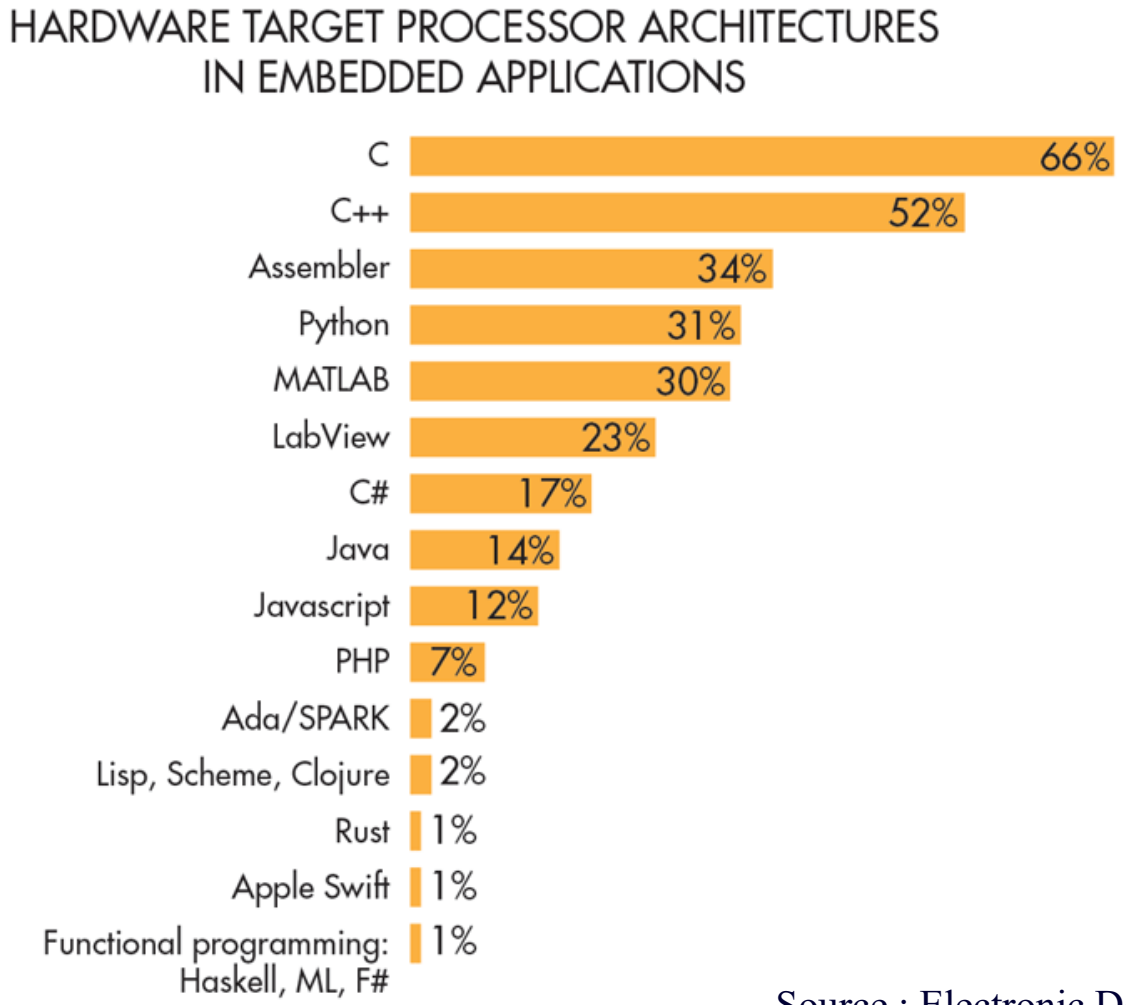
- Mais attention !
  - la mémoire coûte de moins en moins cher
  - les compilateurs pour langages évolués (C, Ada, etc.) génèrent du code de plus en plus optimisé :
    - plusieurs critères :
      - vitesse d'exécution,
      - encombrement mémoire,
      - etc.
  - surtout vrai pour les micro-contrôleurs

Statistiques 2014

Source :
IEEE Spectrum July 2014



Langages pour systèmes embarqués (2017)

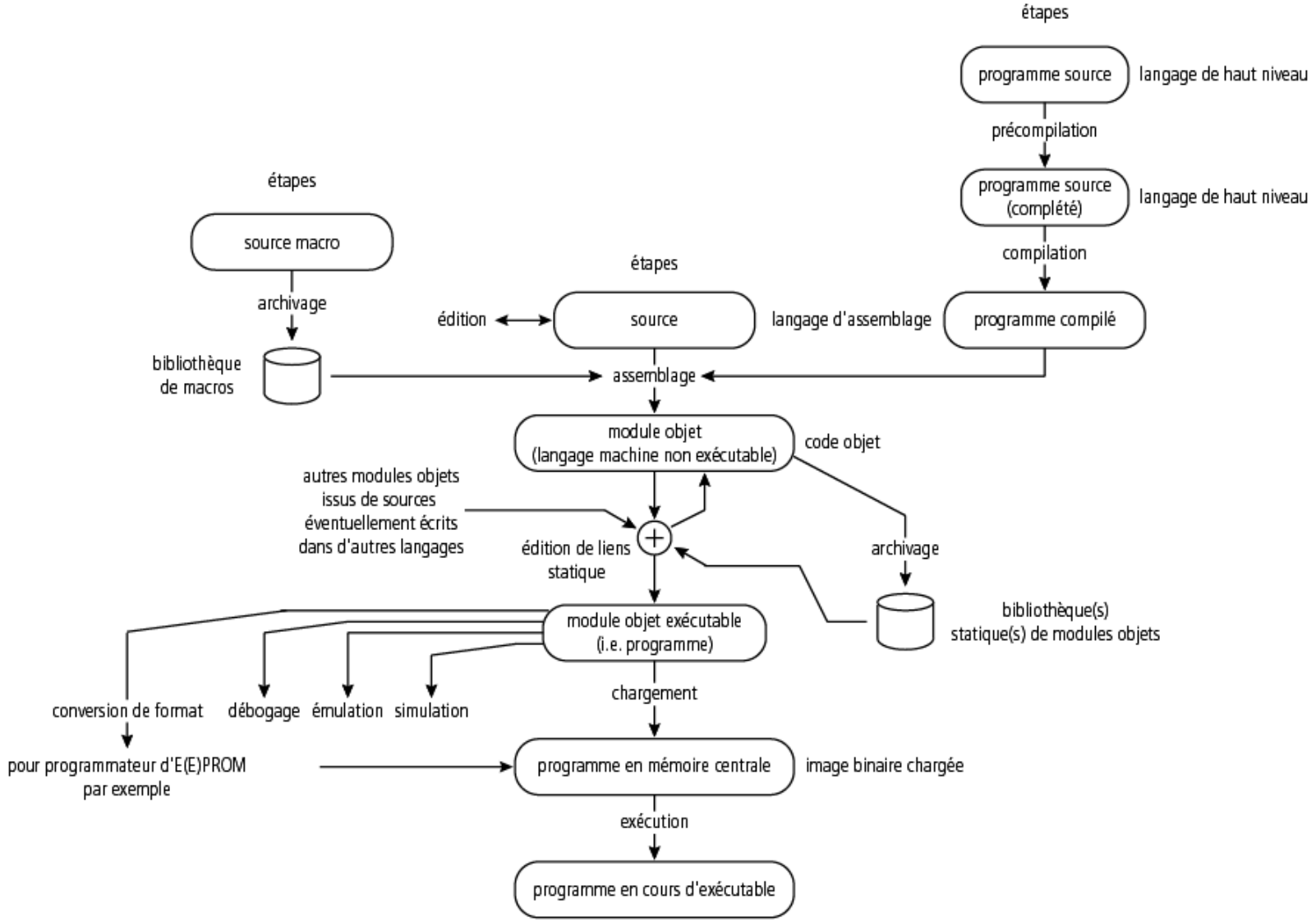


Source : Electronic Design

Macro-assembleur

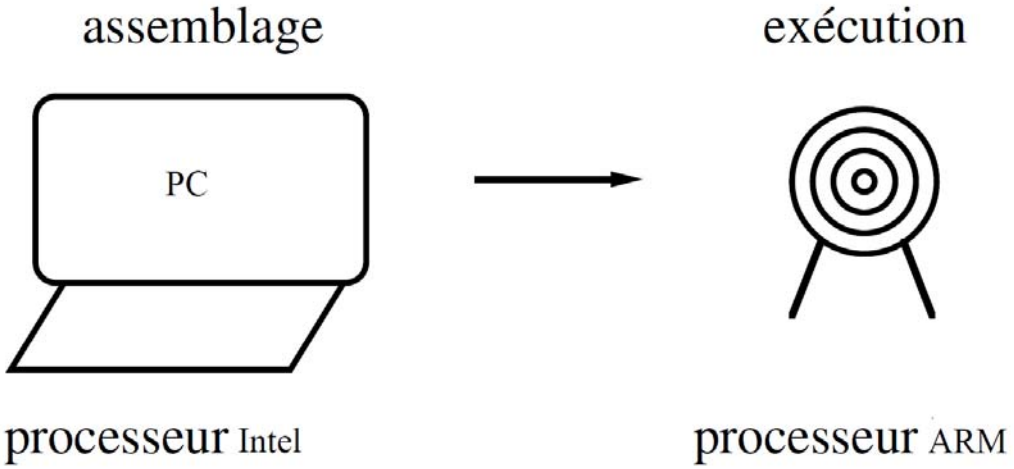
- Utilisation de macro-instructions
  - macro-instruction :
    - nom symbolique substituable par une suite de textes (directives, mnémoniques, données)
    - possibilité de paramétrage
    - équivalent au « #define » du langage C

Gestion des macros



Cross-assembleur

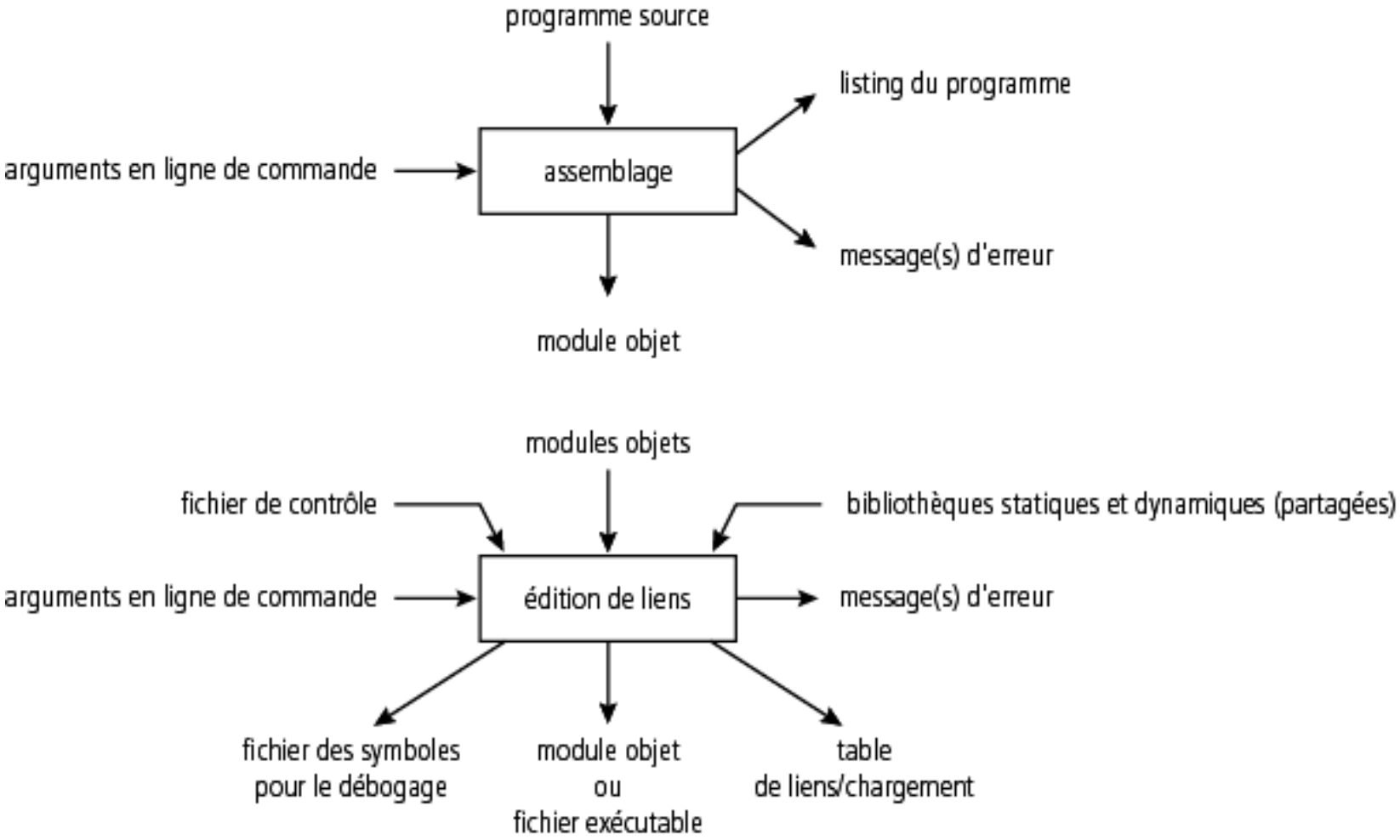
- Assembleur s'exécutant sur une machine d'architecture différente de celle qui doit exécuter le code
- Exemple : cross-assembleur MIPS générant du code destiné à s'exécuter sur un micro-ordinateur de type PC (µP Intel X86)



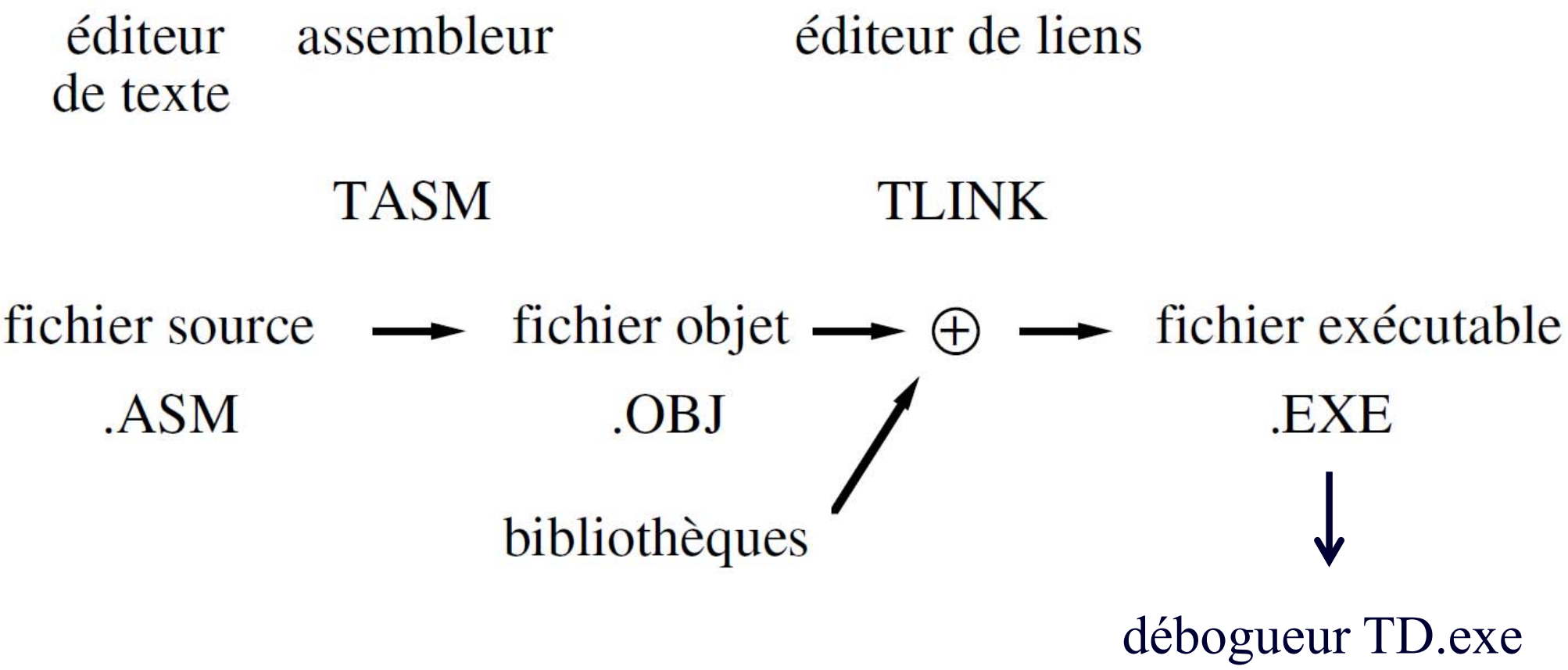
Autres assembleurs

- Assembleur en ligne du compilateur (*inline assembler*)
- Assembleur de correctifs (*patch assembler*)
- Assembleur multi-processeur
- Assembleur de haut niveau
  - permet le typage
- Méta-assembleur (curiosité scientifique)

Assembleur et éditeur de liens



Chaîne de développement logiciel



Syntaxe des commandes

- TASM <options> <nom_fichier>
  - /l fichier listing (.LST) généré
  - /z affichage des lignes de source à côté des messages d'erreur
  - /zi génération d'informations de déboggage dans le fichier objet
- TLINK <options> <nom_fichier>
  - /v ajout d'informations de déboggage au fichier exécutable
- Exemples :
  - tasm /l /z /zi essai.asm
  - tlink /v essai.obj

Structure d'une ligne instruction

- Programme en langage d'assemblage
  = {directives, données, instructions, commentaires (50 % !)}
- Structure d'une ligne instruction

  [label] [préfixe] [mnémonique] [opérande] [,opérande] [;commentaire]
- Convention :

  instruction opérande destination,opérande source

Structure d’un programme

<directives d'assemblage>

<directives d'édition de lien>

DATASEG

<déclaration des variables>

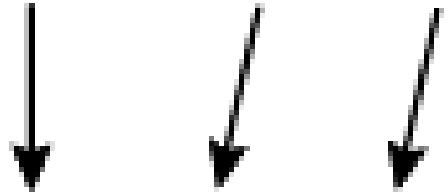
CODESEG

<déclaration du code>

END.

Syntaxes autorisées

add r/m8,imm8



add al,255

ou

add [memo],0FFh

memo db 4

Syntaxes interdites

add r/m8,imm16

incohérence de format

add al,65535

add imm8,imm8

cas impossible

add 20,020h

Exemples de codage d'une instruction

- 72 XX jb boucle
- 49 dec cx
- C3 ret

- X : chiffre hexadécimal

Description constructeur d'une instruction

ADD — Add

Opcode	Instruction	Clocks	Description
04 ib	ADD AL,imm8	2	Add immediate byte to AL
05 iw	ADD AX,imm16	2	Add immediate word to AX
05 id	ADD EAX,imm32	2	Add immediate dword to EAX
80 /0 ib	ADD r/m8,imm8	2/7	Add immediate byte to r/m byte
81 /0 iw	ADD r/m16,imm16	2/7	Add immediate word to r/m word
81 /0 id	ADD r/m32,imm32	2/7	Add immediate dword to r/m dword
83 /0 ib	ADD r/m16,imm8	2/7	Add sign-extended immediate byte to r/m word
83 /0 ib	ADD r/m32,imm8	2/7	Add sign-extended immediate byte to r/m dword
00 /r	ADD r/m8,r8	2/7	Add byte register to r/m byte
01 /r	ADD r/m16,r16	2/7	Add word register to r/m word
01 /r	ADD r/m32,r32	2/7	Add dword register to r/m dword
02 /r	ADD r8,r/m8	2/6	Add r/m byte to byte register
03 /r	ADD r16,r/m16	2/6	Add r/m word to word register
03 /r	ADD r32,r/m32	2/6	Add r/m dword to dword register

Operation

```
DEST ← DEST + SRC;
```

Description

ADD performs an integer addition of the two operands (DEST and SRC). The result of the addition is assigned to the first operand (DEST), and the flags are set accordingly.

When an immediate byte is added to a word or doubleword operand, the immediate value is sign-extended to the size of the word or doubleword operand.

Flags Affected

OF, SF, ZF, AF, CF, and PF as described in Appendix C

Déclaration des données

- [nom_variable] format [valeur initiale][,valeur initiale]

  ⇒ pas de typage des données comme dans un langage évolué

DB	1	octet
DW	2	mot 16 bits
DD	4	double mots 16 bits = 32 bits
DF,DP	6	
DQ	8	
DT	10	

Exemples de déclaration de données

- memo1 DW 256
  - variable memo1 d'une taille de 16 bits initialisée à la valeur décimale 256
- memo2 DB 12,13,14,15
  - tableau de 4 octets initialisés aux valeurs décimales 12, 13, etc.
- DD 0FF00FF12h
  - variable sans nom initialisée à une valeur hexadécimale

Exemples de déclaration de données

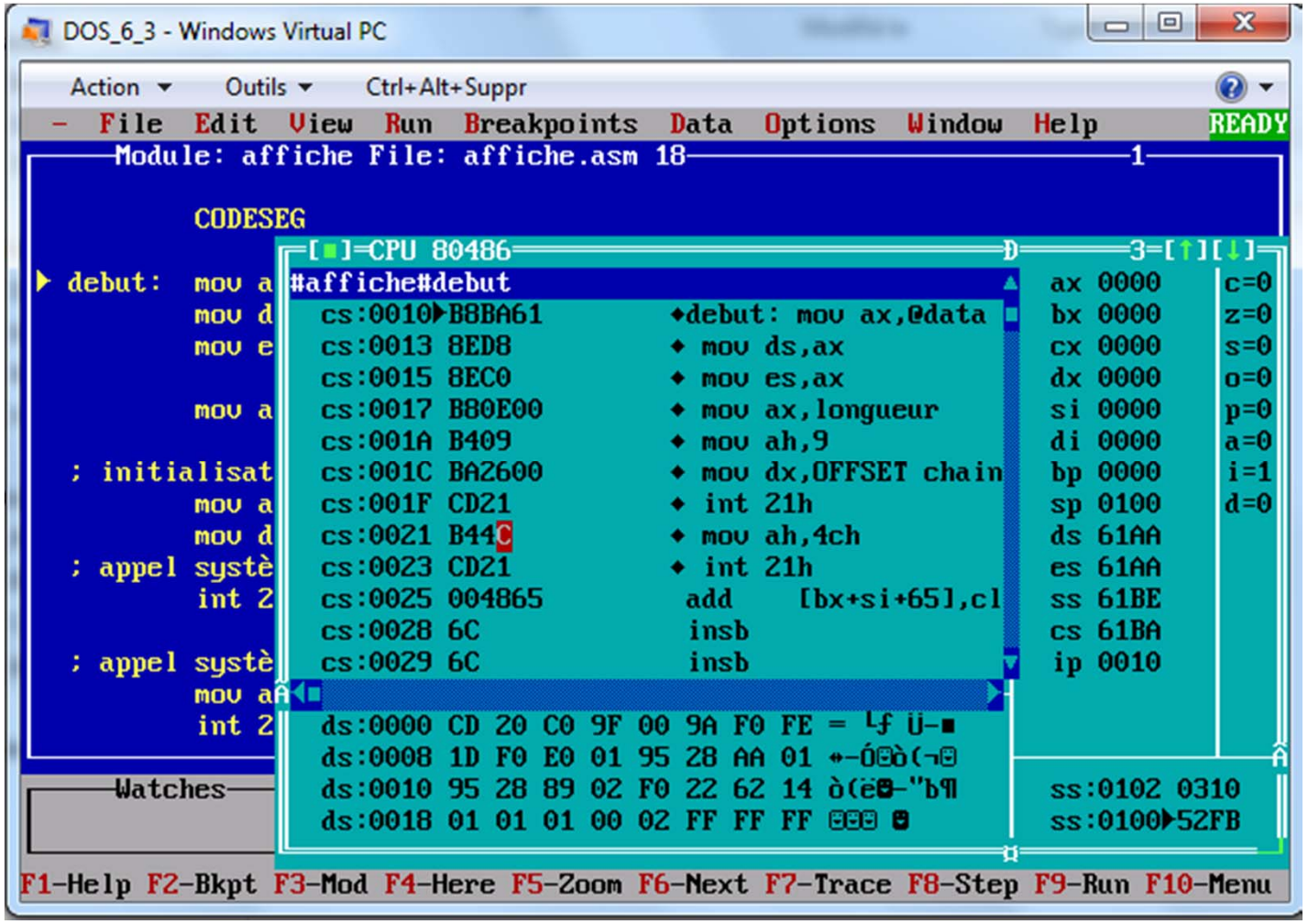
- chaine1 DB "string",0Dh,0Ah
  - variable au format chaîne de caractères initialisée à la valeur "string" (codes alphanumériques) avec, en fin, un caractère de contrôle (13_{10}), le retour-chariot et un saut de ligne (Line Feed, 10_{10}),
- memo2 DB 256 DUP (0)
  - zone memo2 de 256 octets initialisés à la valeur 0

Déclaration d'un label

- Définition
  - un label est une déclaration de substitution
- Exemple

```
retour_chariot EQU 13
chaine DB "bonjour",retour_chariot
```

Les registres du 8086 dans Turbo Debugger



Les registres internes

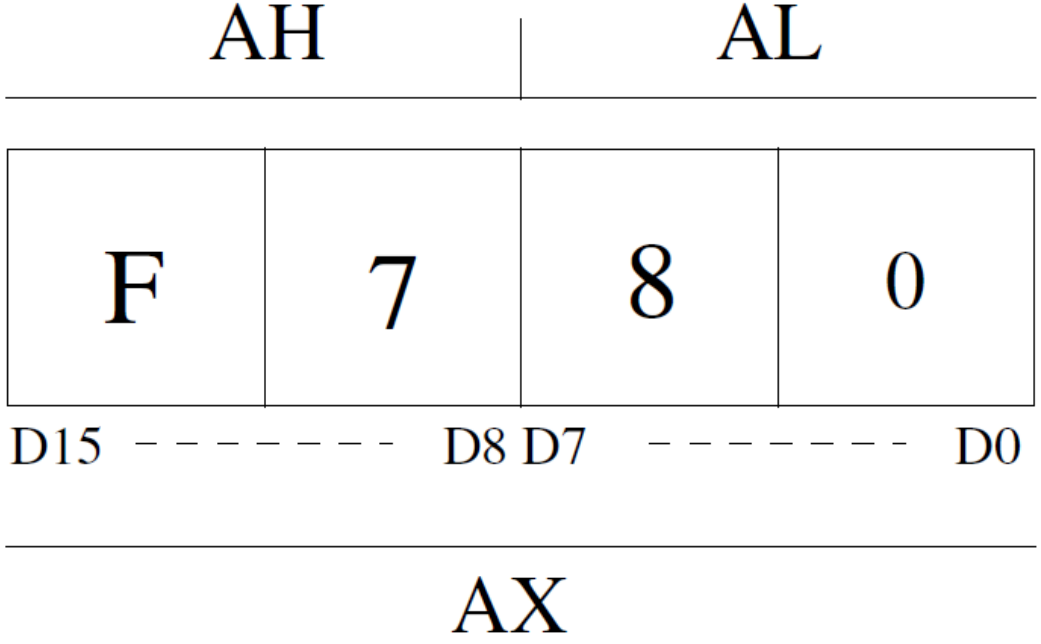
- 14 registres de 16 bits répartis en 3 groupes A, B et C :
  - certains accessibles au programmeur
  - d'autres non accessibles sauf mécanismes spéciaux
- Groupe A des registres généraux
  - sous-groupe A1 des registres de données (registres de travail)
  - sous-groupe A2 des registres d'index (accès à des tableaux)
  - sous-groupe A3 des registres pointeurs (indirection)
- Groupe B des registres de contrôle
- Groupe C des registres de segmentation (gestion mémoire)

Le sous-groupe A1

- Les registres de donnée
  - AX : registre implicite
  - BX : registre de base
  - CX : registre de comptage
  - DX : registre banalisé
- Chaque registre décomposable selon la syntaxe en deux registres 8 bits

Décomposition d'un registre de donnée

- AH représente l'octet le plus significatif ou *Most Significant Byte* (MSB) de AX
- AL représente l'octet le moins significatif ou *Least Significant Byte* (LSB) de AX
- Exemple :
  - AH = F7h
  - AL = 80h
- et donc :
  - AX = F780h
  - AX = ((AH) × 16^2) + (AL)
- Idem pour :
  - BX
  - CX
  - DX



Le sous-groupe A2

- Les registres d'index
  - SI : Source Index
  - DI : Destination Index
- Utilisés principalement pour les opérations s'appliquant aux caractères

Le sous-groupe A3

- Les registres des pointeurs
  - BP : pointeur de base (*Base Pointer*)
  - SP : pointeur de pile (*Stack Pointer*)
- Utilisés principalement pour la sauvegarde de contexte et le passage de paramètres

Le groupe B

- Les registres de contrôle
  - IP : Pointeur d'instruction (*Instruction Pointer*)
    - autre appellation : PC ou *Program Counter*
    - à chaque exécution, IP contient l'adresse de la prochaine instruction à aller chercher, sauf exception
  - Registre d'état : registre des indicateurs binaires
    - autre appellation : registre des drapeaux (*Flags Register*)
    - renseigne le programmeur sur la validité et la nature du résultat

Le registre d'état

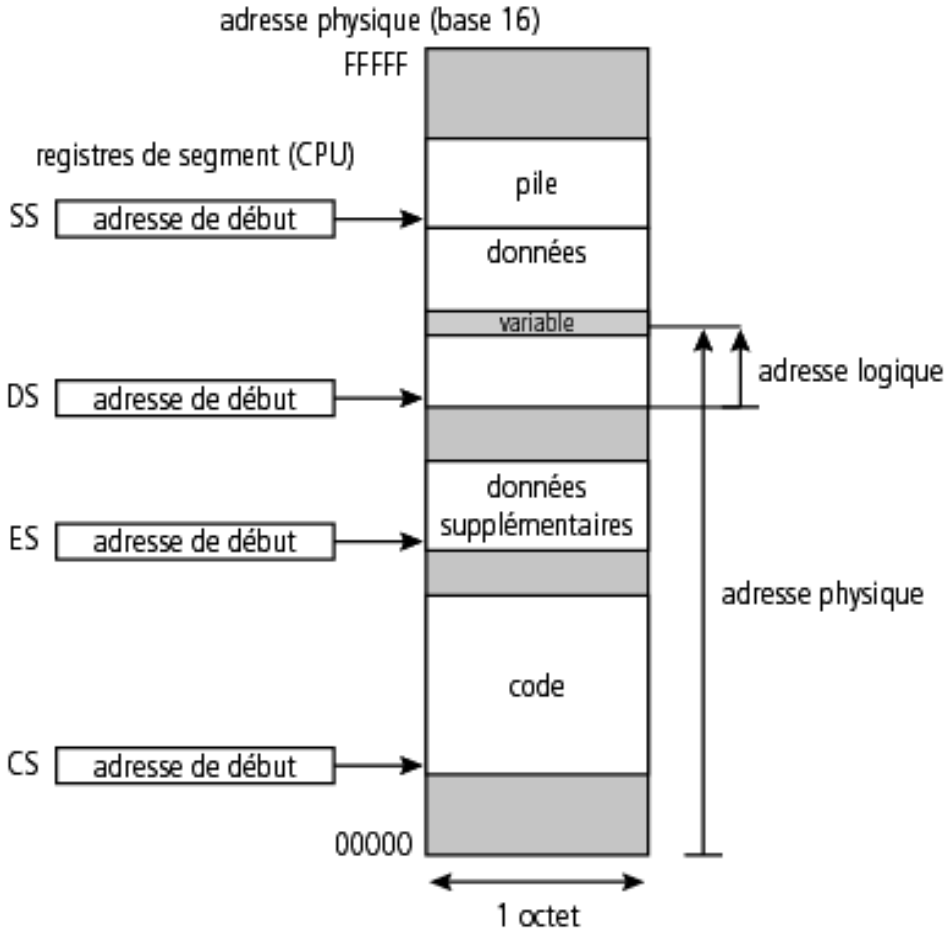
- Caractéristiques d'un registre d'état :
  - état temporaire ➡ jusqu'à la prochaine instruction risquant de modifier l'indicateur concerné
  - état permanent pour contrôler le fonctionnement du processeur
- Utilité des drapeaux :
  - renseignement sur la nature du résultat :
    - résultat nul, polarité, parité logique, etc.
  - renseignement sur la validité du résultat :
    - retenues, dépassement de capacité, etc.
  - modification du comportement de certaines instructions
    - Sens de transfert lors de la manipulation des caractères
  - modification du comportement du processeur
    - mode pas à pas
    - masquage des interruptions

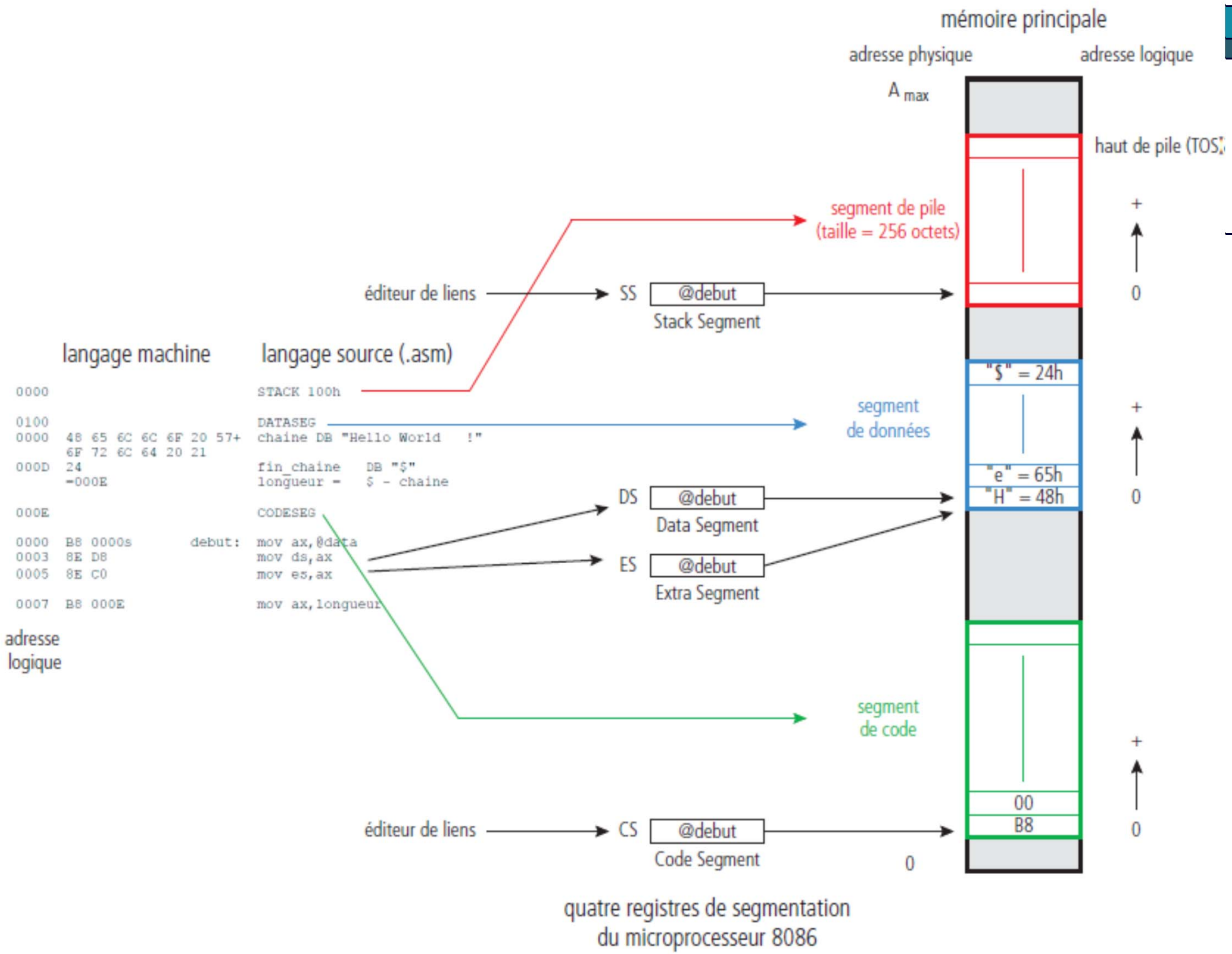
Le groupe C

- CS : *Code Segment*
  - registre de segment de code
- SS : *Stack Segment*
  - registre de segment de pile
- DS : *Data Segment*
  - registre de segment de donnée
- ES : *Extra Segment*
  - registre d'extra-segment

Les segments du 8086

- Un segment = bloc
  de mots mémoires contigus
  de taille variable
  et à contenu spécialisé
- 4 registres de segment
  - contiennent l'adresse
    de début du segment en MC





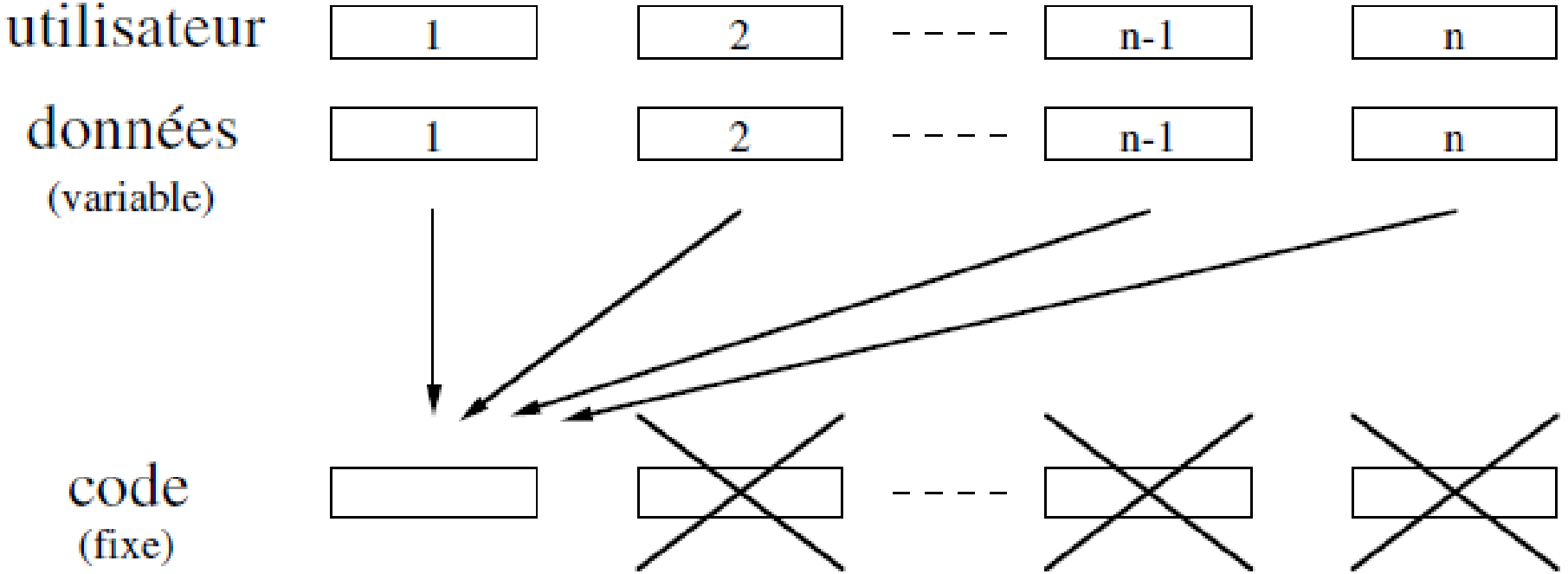
quatre registres de segmentation
du microprocesseur 8086

Intérêt de la spécialisation du contenu

- Données / code séparés

réentrance possible ☞ un seul programme utilisé par plusieurs utilisateurs

☞ un segment de données par utilisateur

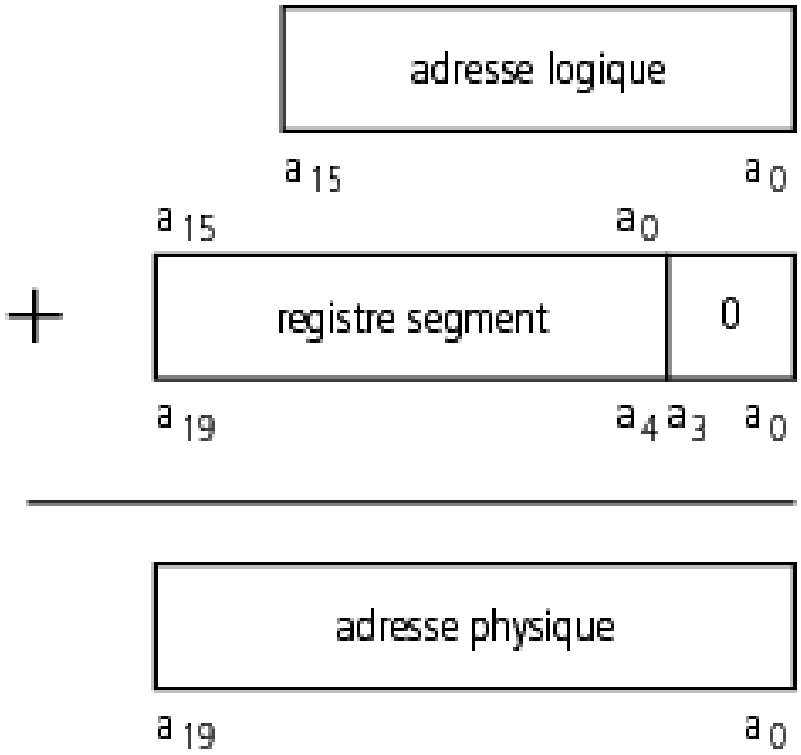


Adresses

- Logique
  - relative au début d'un segment
  - générée par l'assembleur
- Physique
  - relative au début de la mémoire principale
  - générée par le microprocesseur sur le bus d'adresse pour la mémoire

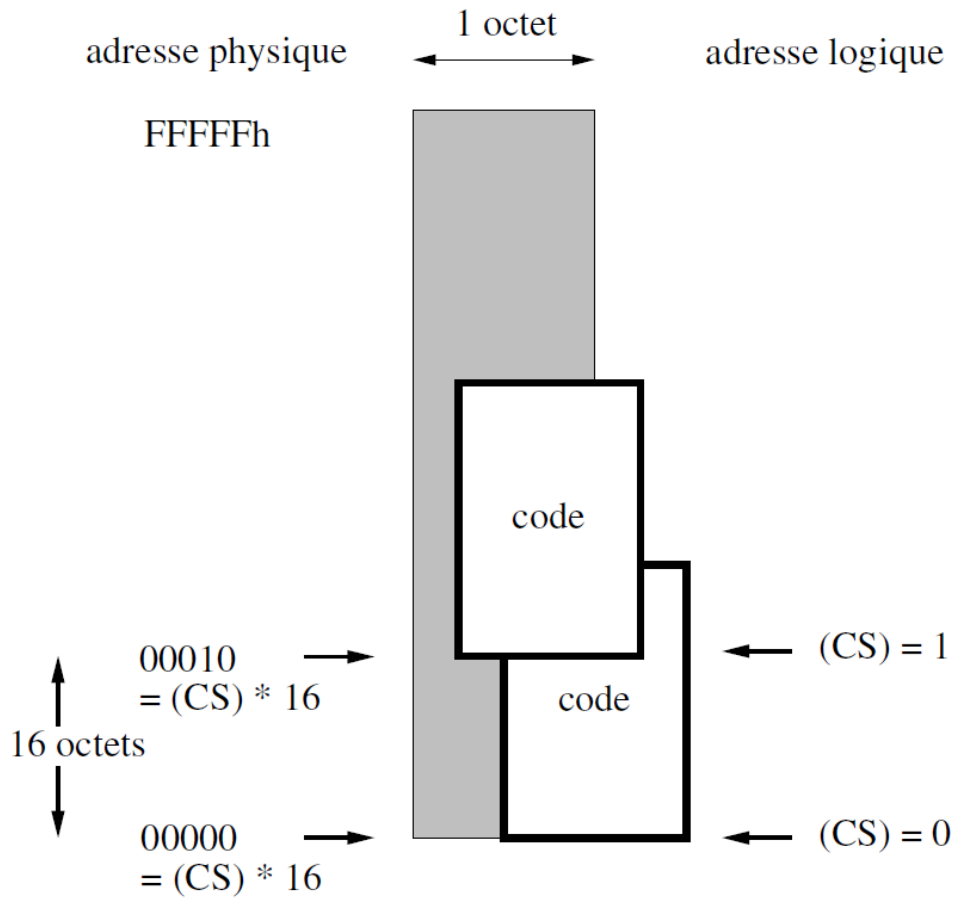
Calcul de l'adresse physique

- *i.e.* l'adresse d'un identificateur (fonction ou donnée) ou d'une instruction



La segmentation

- Pour le x86, un segment est positionné en mémoire centrale tous les 16 octets
- Exemple avec le segment de code



Les familles principales d'instructions

- Instructions de transfert de données
- Instructions de traitement
  - arithmétiques
  - logiques
- Instructions de contrôle
  - sauts
  - appels de sous-programme
  - interruptions logicielles

Transfert de données

- L'instruction MOV
  - syntaxe : mov opérande destination, opérande source
    - ⇒ deux opérandes
  - opération :
    - copie du contenu de l'opérande source dans l'opérande destination
  - pseudo-code :
    [opérande destination] (opérande source)
  - aucun indicateur positionné
  - exemples
    mov ax,bx; ax ← (bx)
    mov bx,[memo]; bx ← ([memo])
    mov cl,255; cl ← FFh

Les transferts possibles

- Registre vers registre
  - mov ax,bx
- Registre vers mémoire
  - mov [memo],ax
  - mov [bx],bx
- Mémoire vers registre
  - mov dl,[memo1]
- Valeur immédiate vers registre
  - mov cx,4
- Valeur immédiate vers mémoire
  - mov [memo],0FFFFh

Les transferts possibles (suite)

- Cas particuliers :
  - mémoire vers mémoire
    - instructions de manipulation de caractères :
      - movsb
    - instructions de manipulation de la pile :
      - push [memo]

Les quatre opérations arithmétiques de base

- Addition : ADD
- Soustraction : SUB
- Multiplication : MUL
- Division : DIV

L’addition

- Syntaxe

  add opérande destination, opérande source

  ⇒ deux opérandes

- Opération :
  - addition du contenu de l'opérande source avec l'opérande destination et résultat dans l'opérande destination
- Pseudo-code :

  [opérande destination] ← (opérande source) + (opérande destination)

Exemples d'addition

- add ax,bx; ax ← (ax) + (bx)
- add bx,[memo]; bx ← (bx) + ([memo])
- add cl,255; cl ← (cl) + FFh

L'incrémentation

- Syntaxe :

  inc opérande

  ⇒ un opérande
- Opération :
  - incrémentation du contenu de l'opérande
- Pseudo-code :
  - opérande ← (opérande) + 1
- Ajustements AAA et DAA opérationnels

Exemples d'incrémentation

- inc ax; ax ← (ax) + 1
- inc cl; cl ← (cl) + 1
- inc [memo]; [memo] ← ([memo]) + 1

La soustraction

- Fonctionne comme l'addition
- Voir le TP correspondant

La décrémentation

- Syntaxe :

  dec opérande

  ⇒ un opérande
- Opération :
  - décrémentation du contenu de l'opérande
- Pseudo-code :
  - opérande ← (opérande) - 1
- Ajustements AAA et DAA opérationnels

Exemples de décrémentation

- dec ax; ax ← (ax) - 1
- dec cl; cl ← (cl) - 1
- dec [memo]; [memo] ← ([memo]) – 1

La multiplication

- Syntaxe :

  mul opérande

  ⇒ un opérande explicite

  - deux registres implicites AX et DX

- Opération :

  - multiplication non signée de l'opérande avec le registre implicite AX et résultat dans le(s) registre(s) implicite(s)

La multiplication (suite)

- Pseudo-code :

  si format (opérande) = octet

  alors AX ← (AL) × (opérande)

  sinon DX:AX ← (AX) × (opérande)

  fin si

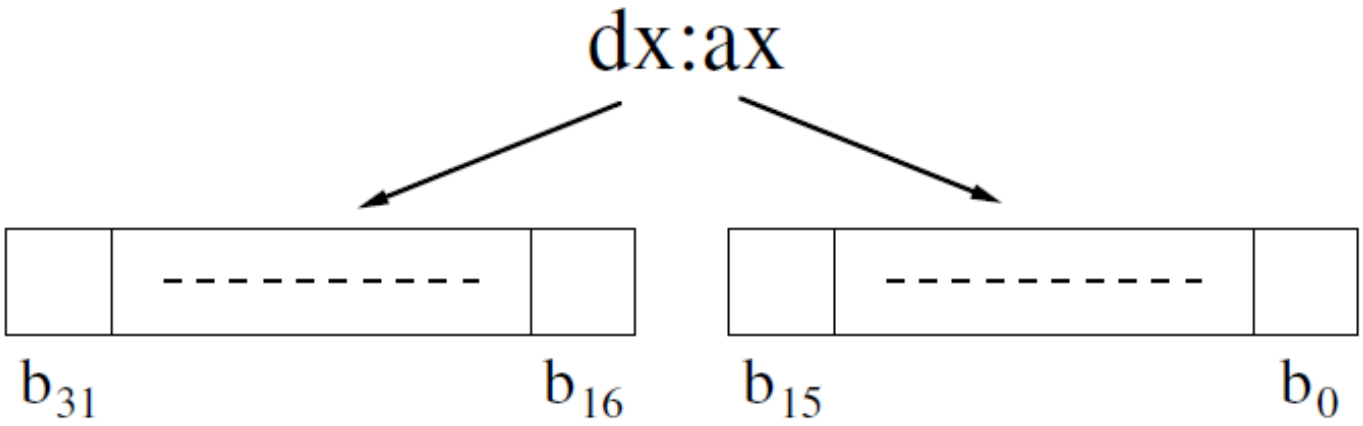
Exemples d'utilisation

- Multiplication au format octet
  - mov al,2; al ← 2
  - mov bl,3; bl ← 3
  - mul bl; ax ← (al) × (bl)
- Multiplication au format 16 bits
  - mov ax,2; ax ← 2
  - mov bx,3; bx ← 3
  - mul bx; dx:ax ← (bx) × (ax)

La multiplication au format 16 bits

- Résultat sur deux registres

MUL 16 X 16 bits = résultat sur 32 bits



Les opérateurs logiques

- NOT : complément
- AND : ET
- OR : OU (inclusif)
- XOR : OU exclusif
- TEST : ET logique sans résultat (voir cours ASR2 sur les débranchements)
- SHL/SAL : *SHift logical Left / Shift Arithmetic Left*
- SHR : *SHift logical Right*
- SAR : *Shift Arithmetic Right*
- ROL : *ROtate Left*
- ROR : *ROtate Right*
- RCL : *Rotate through Carry Left*
- RCR : *Rotate through Carry Right*

L'opérateur complément

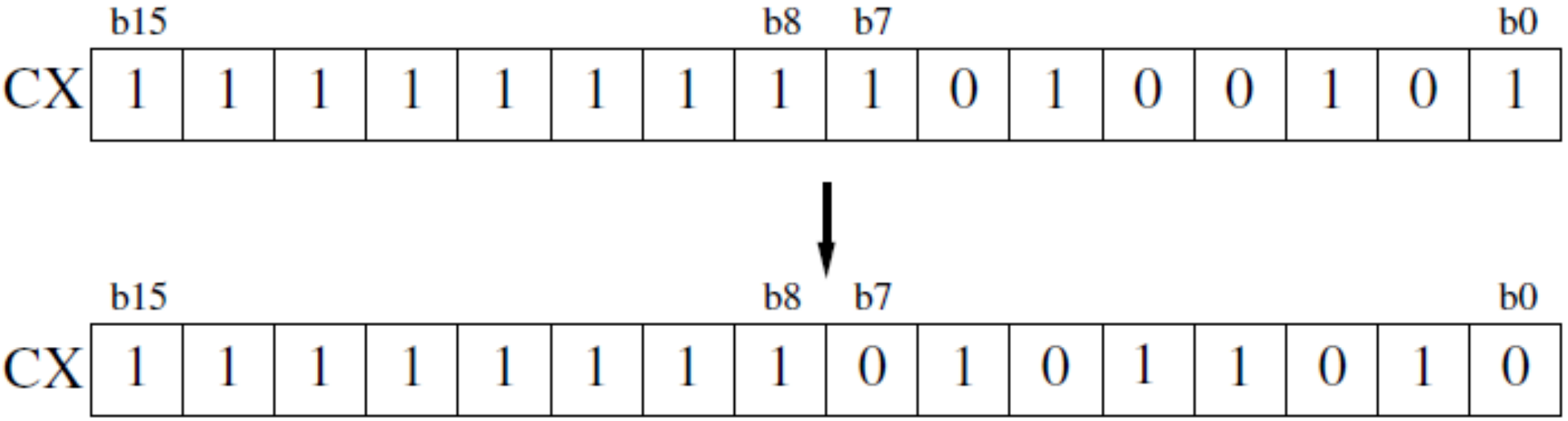
- Syntaxe :

  not op_dest
- Un opérande
- Opération :
  - négation de l'opérande destination = non logique
- Pseudo-code :
  - [op_dest] ← /(op_dest)

Exemples

- not [memo]; [memo] ← /([memo])

- mov cx,0FFA5h
- not cl; cl ← /(cl); (cx) = FF5Ah



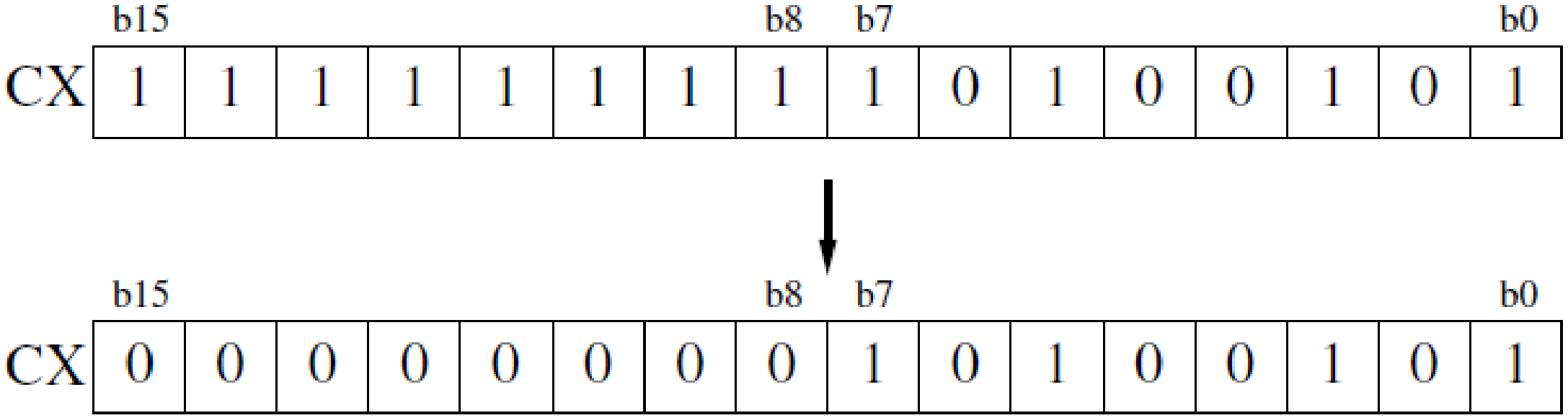
Le ET logique

- Syntaxe :

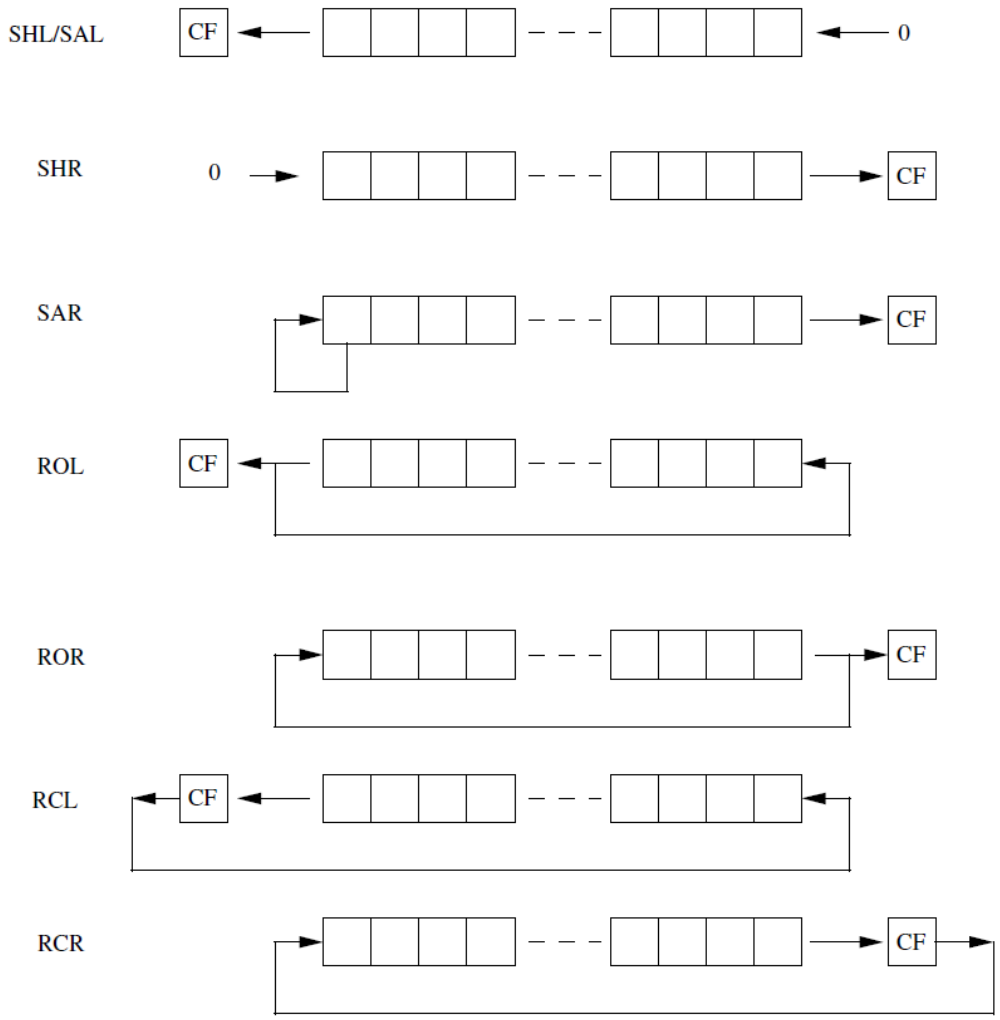
  and op_dest,op_source
- Deux opérandes
- Opération :
  - ET logique du contenu de l'opérande destination avec le contenu de l'opérande source
- Pseudo-code :
  - [op_dest] ← (op_dest) • (op_source)
- Utilité : masquage de bits

Exemples

- and [memo],256; [memo] ← ([memo]) • 0100h
- mov cx,0FFA5h
- and cx,0000000011111111b; cx ← (cx) • FFh



Décalages et rotations



Décalages à gauche

- Syntaxes :

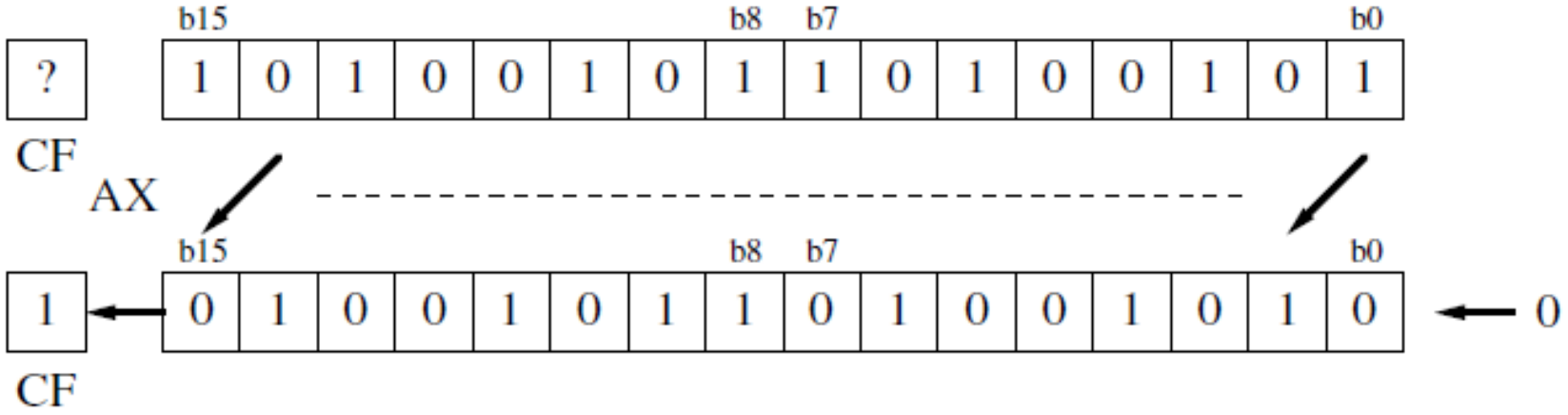
  sal op_dest,op_source

  shl op_dest,op_source
- Deux opérandes
- Opération :
  - décalage logique du contenu de l'opérande destination,
    le contenu de l'opérande source indique le nombre de décalages
- Utilité : multiplication par 2

Exemples

- sal [memo],2

- mov ax,0A5A5h
- mov bl,1
- shl ax,bl; décalage d'un bit à gauche



Décalage logique à droite

- Syntaxe :

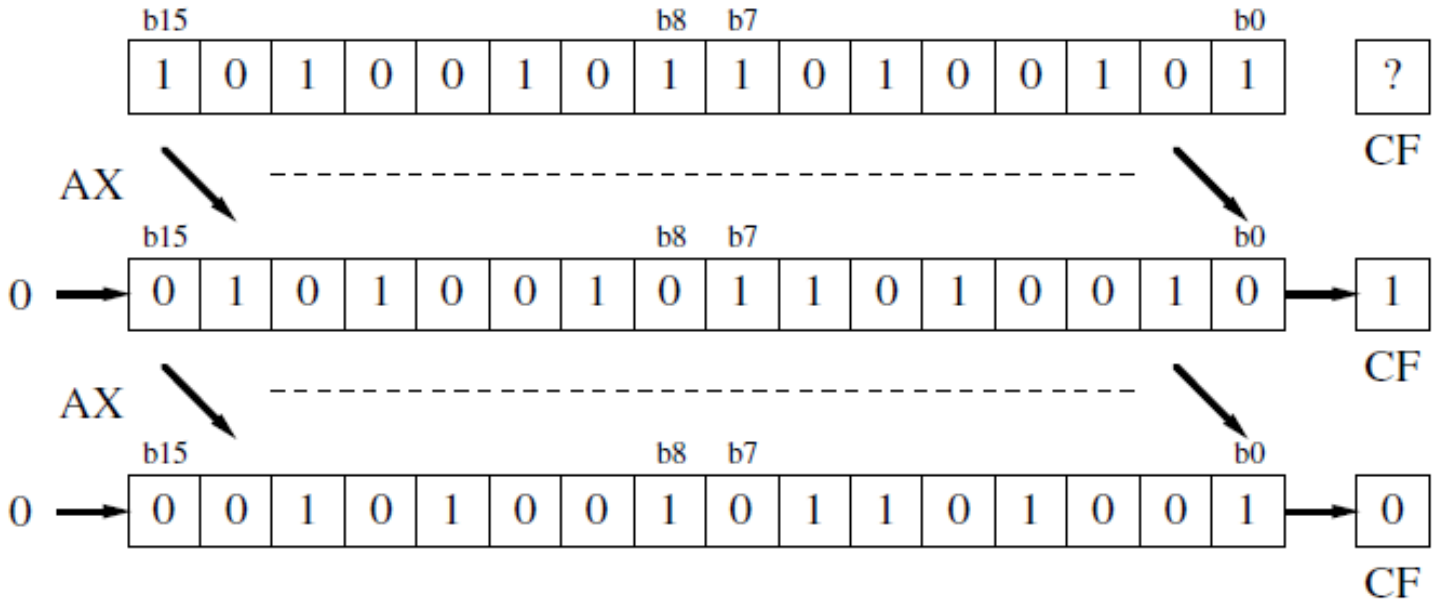
  shr op_dest,op_source

  ⇒ deux opérandes

- Opération :
  - décalage logique à droite du contenu de l'opérande destination, le contenu de l'opérande source indique le nombre de décalages
- Utilité : division non signée par 2

Exemples

- shr [memo],2



- mov ax,0A5A5h
- mov bl,2
- shr ax,bl; décalage de deux bits à droite

Le décalage arithmétique à droite

- Syntaxe :

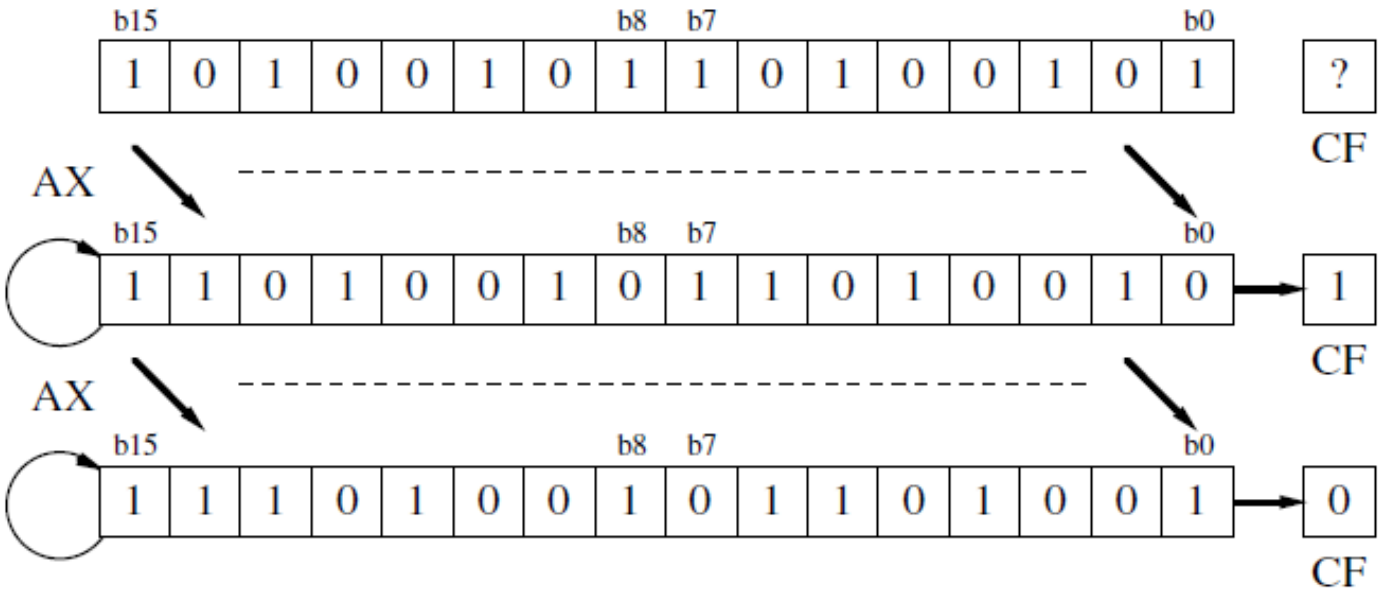
  sar op_dest,op_source

  ⇒ deux opérandes

- Opération :
  - décalage arithmétique à droite du contenu de l'opérande destination, le contenu de l'opérande source indique le nombre de décalages
- Remarque : comportement différent par rapport à la division signée (IDIV) par 2

Exemples

- sar [memo],2



- mov ax,0A5A5h
- mov bl,2
- sar ax,bl; décalage de deux bits à droite

Rotation à gauche

- Syntaxe :

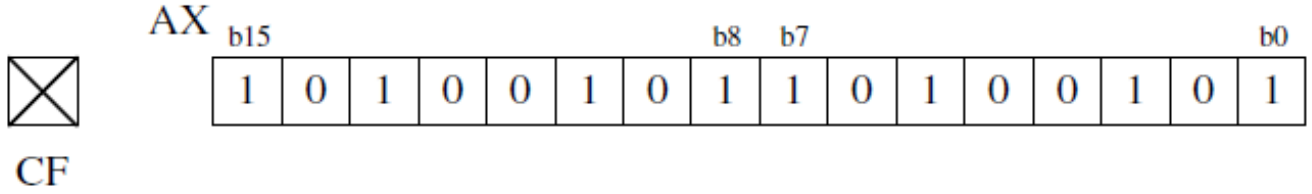
  rol op_dest,op_source

  ⇒ deux opérandes

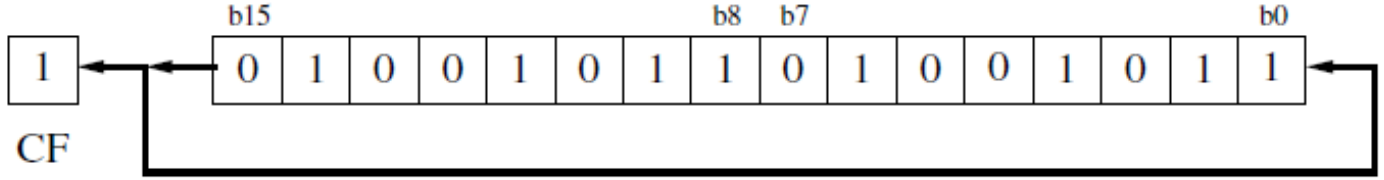
- Opération :
  - rotation à gauche du contenu de l'opérande destination et remplissage du CF avec le MSB, le contenu de l'opérande source indique le nombre de décalages

Exemples

- rol [memo],2



- mov ax,0A5A5h
- mov bl,1
- rol ax,bl; décalage d'un bit à gauche



Rotation à gauche avec retenue

- Syntaxe :

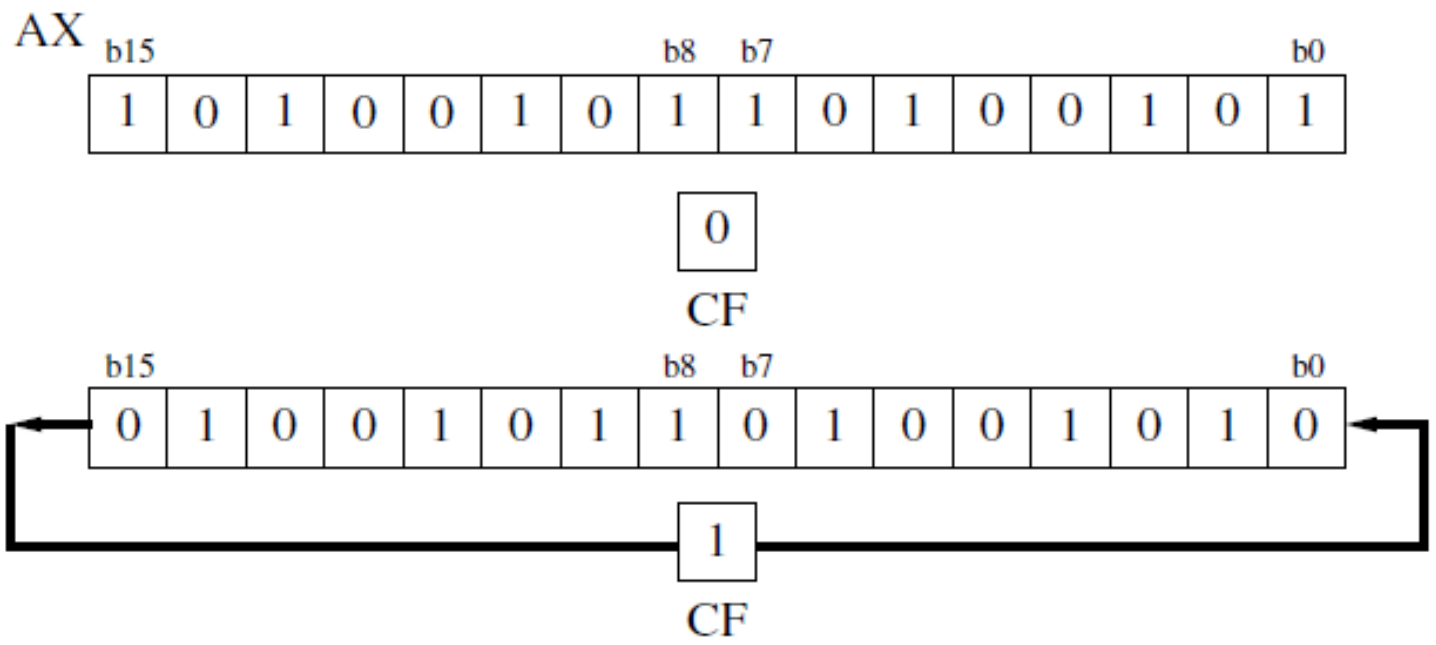
  rcl op_dest,op_source

  ⇒ deux opérandes

- Opération :
  - rotation à gauche du contenu de l'opérande destination, le contenu de l'opérande source indique le nombre de décalages, CF fait partie de l'opérande destination

Exemples

- rcl [memo],2



- clc
- mov ax,0A5A5h
- mov bl,1
- rcl ax,bl; décalage d'un bit à gauche

Les instructions de manipulation de caractères

- LODS chargement de caractère(s)
  - LODSB format octet = un caractère
  - LODSW format mot = deux caractères
- STOS déchargement de caractère(s)
  - STOSB format octet = un caractère
  - STOSW format mot = deux caractères
- MOVS transfert de caractère(s)
  - MOVSB format octet = un caractère
  - MOVSW format mot = deux caractères
- CMPS comparaison de caractère(s)
  - CMPSB format octet = un caractère
  - CMPSW format mot = deux caractères
- SCAS recherche de caractère(s)
  - SCASB format octet = un caractère
  - SCASW format mot = deux caractères

Les instructions de manipulation de caractères

- Répétition d'une instruction de manipulation
  - REP
  - REPNE ou REPNZ
  - REPE ou REPZ
- Manipulation de l'indicateur de direction (DF)
  - CLD
  - STD

Conclusion sur le langage d'assemblage

- Langage symbolique de l'UC (*i.e.* CPU)
  - ⇒ rapidité d'exécution, compacité du code
- Utilisée dans des niches technologiques
  - SE, compilation-édition de liens, interface langages et systèmes embarqués