Architecture des Ordinateurs et Systèmes d'Exploitation

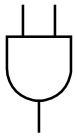
Cours n°4

Le langage assembleur : Introduction et Présentation de l'assembleur du 8086

Ph. Leray

3ème année

Niveaux de programmation (rappel)



MUX ... 0/1

Unité de Traitement
(UAL, Chemin de données)

Unité de Commande

μ-instructions
= suite de 0/1

μ-programme
= suite de μ-instructions

Codop

101010	000010100

Langage machine = suite de 0/1

ADD A,20
JZ 13

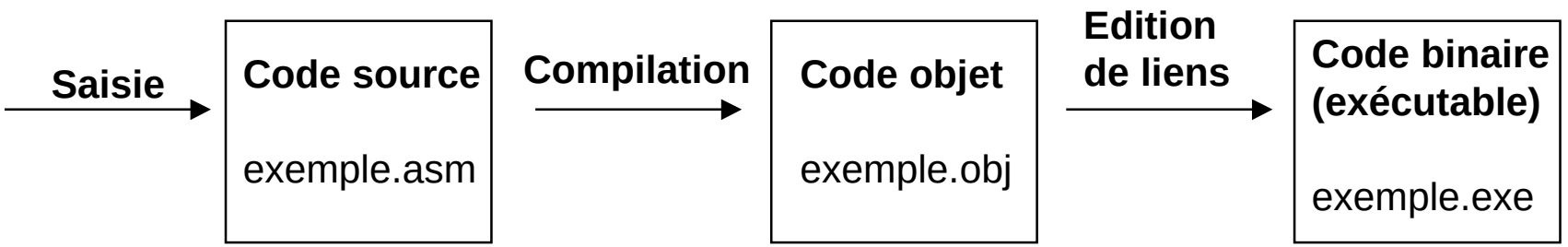
Assembleur = remplacer les codop par
des codes mnémotechniques

Quelques définitions

- **Langage Machine = ordres (en binaire) compréhensibles par un processeur donné**
- **Exécutable = suite d'instructions en langage machine**
- **Langage Assembleur = mnémoniques associées au langage machine (JUMP, ADD, MOV…)**
- **Assemblage = utilisation d'un logiciel spécifique (logiciel d'assemblage ou *assembleur*) pour transformer une suite d'instructions écrites en langage assembleur en un exécutable**

Processus d'assemblage

- **3 phases :**



1. **Saisie du code source avec un éditeur de texte**
2. **Compilation du code source**
3. **Edition des liens**
   - » **permet de lier plusieurs codes objets en un seul exécutable**
   - » **permet d 'inclure des fonctions prédéfinies dans des bibliothèques**

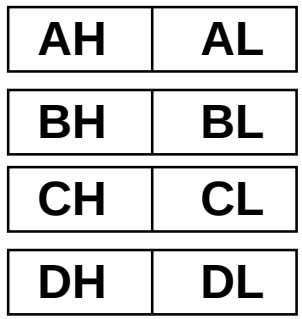
Les microprocesseurs

- **Deux grandes familles :**
  - **Les microprocesseurs CISC (Complex Instruction Set Computer)**
    - » **Jeu d'instruction étendu ⇒ taille « moyenne »**
    - » **2 grands constructeurs :**
      - **INTEL (8088, 8086, 80286… Pentium…)**
      - **MOTOROLA (68000… 68040)**
  - **Les microprocesseurs RISC (Reduced Instruction Set Computer)**
    - » **Jeu d'instruction réduit ⇒ taille « faible »**
    - » **cela permet de rajouter plus de registres, de mémoire cache, … sur le processeur**
      - **IBM/MOTOROLA (PowerPC)**
      - **SUN (Supersparc)**
      - **DIGITAL (Alpha)**

Le microprocesseur 8086

- **Microprocesseur des premiers PC et compatibles**
- **Compatibilité ascendante : un programme écrit pour le 8086 marche pour les processeurs suivants**
- **Caractéristiques :**
  - **Bus de données : 16 bits**
  - **Bus d'adresse : 20 bits**
  - **Registres : 16 bits**
    - » **Accumulator (AX)**
    - » **Base (BX)**
    - » **Counter (CX)**
    - » **Data (DX)**
    - » **Pointeur d'instruction (IP)**
    - » **Registres segments code (CS), data (DS), extra (ES), stack (SS)**
    - » **Pointeur de pile (stack) (SP), de base (BP)**
    - » **Index source (SI), Index destination (DI)**

Certains registres 16 bits peuvent être utilisés comme deux registres 8 bits



L'adressage par segment

- **Largeur du bus d 'adresse = 20 bits**

⇒ **Adressage mémoire = 2^{20} = 1 Mo**

- **Le pointeur d 'instruction fait 16 bits**

⇒ **Adressage = 2^{16} = 64 Ko** cela ne couvre qu 'une partie de la mémoire

= 1 segment de mémoire

- **Il faut 2 registres pour indiquer une adresse au processeur**

Ex: CS:IP = (4055:3192) = 40550 + 3192 = 436E2

Adresse de début de segment — Offset — Adresse pour le bus de donnée

CS:IP, (index) DS:SI, DS:DI, ES:SI, ES:DI, (pile) SS:SP, SS:BP

Types d'adressage 8086

- **Adressage par registre (ou implicite)**
  Add AX, BX
  Inc AX **(pas d 'accès mémoire pour les opérandes)**

- **Adressage immédiat**
  Add AX, valeur **(1 accès mémoire pour lire la valeur)**

- **Adressage direct**
  Add AX, [adresse] **(2 accès mémoire : adresse puis valeur [adresse])**

- **Adressage indexé (ou relatif)**
  Add AX, [adresse+index] **(2 accès mémoire)**

Jeu d'instructions 8086 (1/2)

- **Instruction d'affectation** **MOV**

- **Instructions arithmétiques**
  INC **(incrémentation)**
  DEC **(décrementation)**
  ADD **(addition)**
  SUB **(soustraction)**
  CMP **(soustraction sans sauvegarde)**
  NEG

- **Instructions logiques**
  NOT, OR, XOR
  AND, TEST (= AND sans sauvegarde)
  SHL (SHR), SAL (SAR)
  ROL (ROR), RCL (RCR)

Jeu d'instructions 8086 (2/2)

- **Branchement :** **JMP**

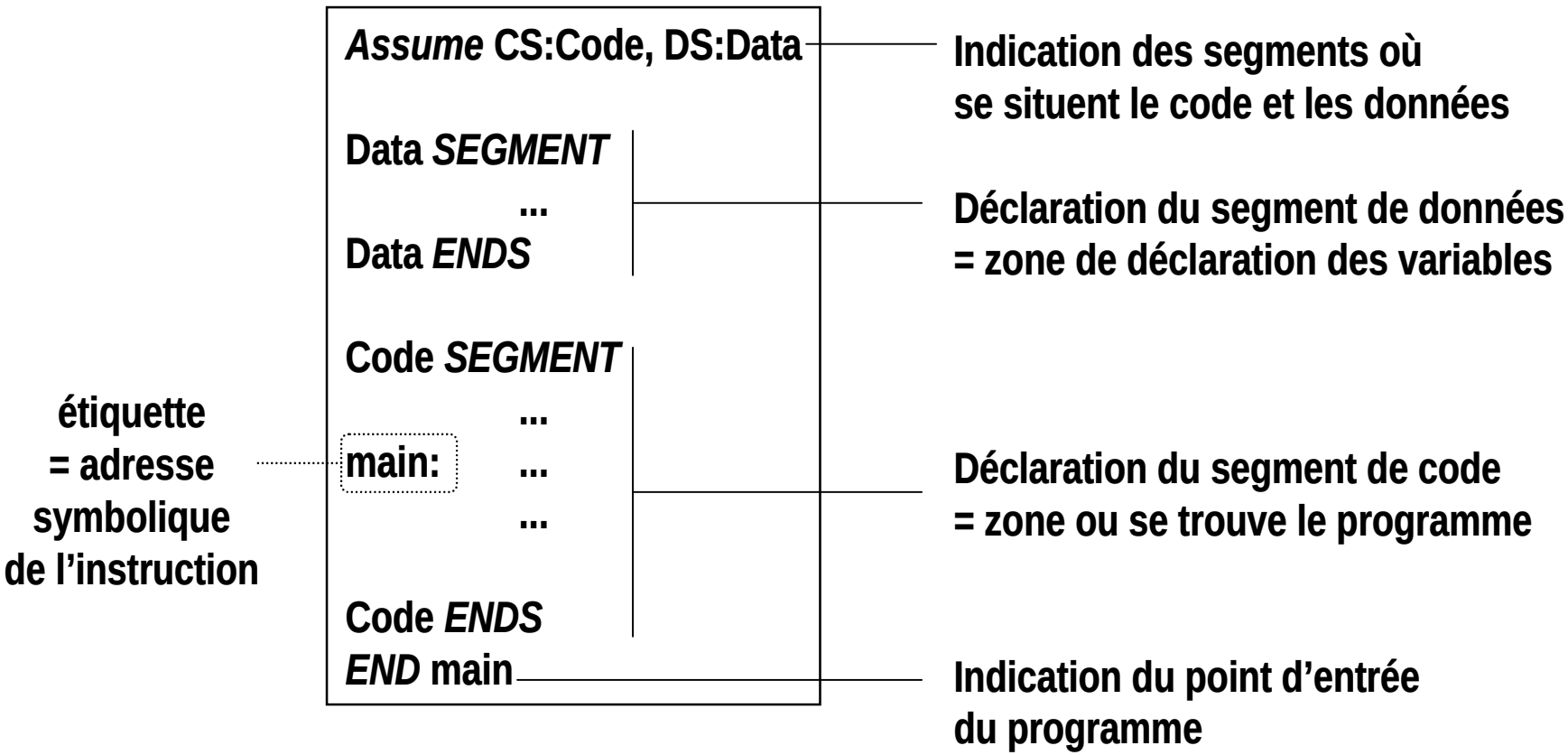
- **Branchements conditionnels :** **JE/JZ (JNE/JNZ) : Jump if zero**
  JO (JNO) : Jump if overflow
  JS (JNS) : Jump if sign

- **Comparaison de valeurs**
  CMP AX, BX suivi d 'un test :

	(entiers naturels)	(complément à 2)
AX > BX ?	JA (≥ JAE)	JG (JGE)
AX < BX ?	JB (JBE)	JL (JLE)

Programmation 8086 (1/2)

- **Structure du programme**



Programmation 8086 (2/2)

- **Déclaration des variables (segment de données)**

```
Assume CS:Code, DS:Data

Data SEGMENT
  N1        dw      25h
  N2        dw      4
  Prod      dw      ?
            ...
  NP        dw      2,3,5,7,11,13
  Table     dw      10,20,30,40,50


TabVide     dw      100 dup (?)
  Tab1      dw      50 dup (1b)

Chaine      db      'TEXTE',0Dh,0Ah

Data ENDS
```

db = Define Byte (Byte = Octet = 8 bits)
dw = Define Word (Word = Mot = 16 bits)
dd = Define Double (Double = 32 bits)
+ valeur initiale (ou liste de valeurs) ou de ?

N1 est une variable de type WORD (dw = Define Word) initialisée à 25 (en héxa)

Table est un tableau (une suite) de WORD initialisé à 10…50 (en décimal)

TabVide est un tableau de 100 WORD non initialisé

Tab1 est un tableau de 50 WORD initialisés à 1 (en binaire)

Les appels système DOS

Pour réaliser les opérations standards (affichage, saisie, …) le système d'exploitation (ici DOS) fournit des fonctions pré-écrites :

– Affichage d'un caractère	Mov DL,"A"	; caractère
	Mov AH, 2	; fonction n°2
	Int 21h	; appel système
– Saisie d'un caractère (avec écho)	Mov AH,1	
	Int 21h	; (résultat dans AL)
– Saisie d 'un caractère (sans écho)	Mov AH,7	
	Int 21h	; (résultat dans AL)
– arrêt du programme	Mov AH, 4Ch	; A mettre a la fin de
	Int 21h	; chaque programme

Mon premier programme

```
Assume CS:Code, DS:Data
Data SEGMENT
   N1          dw          25h
   N2          dw          4
   Prod        dw          ?
Data ENDS
Code SEGMENT
  main :       Mov AX, Data
               Mov DS,AX
               Mov AX, N1
               Mov CX, N2
               Mov DX, 0
  boucle :     Add DX, AX
               Loop boucle
               Mov Prod, DX
               Mov AH, 4Ch
               Int 21h
Code ENDS
END main
```

Indication du segment de données à utiliser dans le programme

Loop = instruction de boucle qui équivaut à
Dec CX
JNZ boucle
(CX sert de compteur de boucle)

A quoi sert ce programme ???