



DIRO  
IFT 1215

**INTRODUCTION AUX  
SYSTÈMES INFORMATIQUES  
SYSTÈME D'EXPLOITATION  
&  
OUTILS DE PROGRAMMATION**

*Max Mignotte*

Département d'Informatique et de Recherche Opérationnelle  
Http: [//www.iro.umontreal.ca/~mignotte/](http://www.iro.umontreal.ca/~mignotte/)  
*E-mail: [mignotte@iro.umontreal.ca](mailto:mignotte@iro.umontreal.ca)*

## **SYSTÈME D'EXPLOITATION**

### **SOMMAIRE**

Introduction .....	2
Structure d'un OS .....	3
OS mono-tâche .....	5
OS multi-tâche .....	8
Service & Facilités .....	9
Types de Systèmes .....	14

## **OUTILS DE PROGRAMMATION**

### **SOMMAIRE**

Introduction .....	17
Traducteur - Compilateur - Assembleur .....	18

# SYSTÈME D'EXPLOITATION

## INTRODUCTION

### Introduction

[*Operating System*]

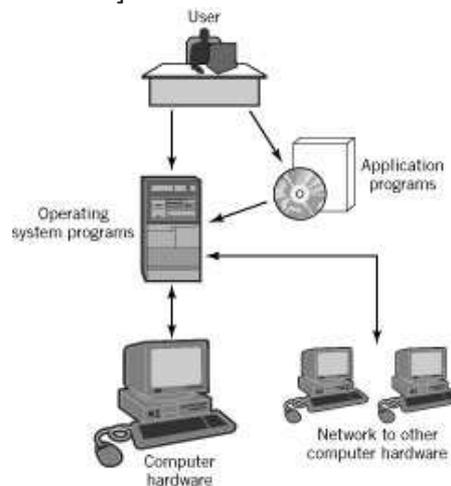
Ensemble de programmes ( $> 10^5$  loc) qui coordonne l'ensemble des tâches essentielles à la bonne marche du complexe matériel et assure la gestion des ressources

Facilite la tâche de l'utilisateur en se chargeant des tâches fastidieuses (chargement/gestion d'un prog., contrôle des périphs., mémoire, interruption, stockage/gestion des fichiers, langage de commande, etc.)

Permet le traitement de multiples programmes et le support pour de multiples utilisateurs, communications entre ordinateurs, etc.

### **Interface entre l'utilisateur et le Système**

Intermédiaire entre l'utilisateur, le programme utilisateur et le [*Hardware*]

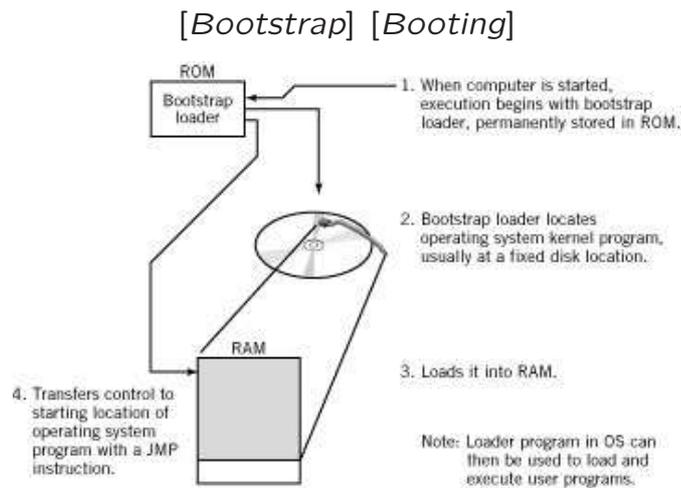


# SYSTÈME D'EXPLOITATION

## STRUCTURE D'UN OS

### Structure d'un OS

Quelques services de l'OS noyau [*Kernel*] (codé en assembleur) sont critiques et indispensables et nécessitent d'être chargés en mémoire dès la mise en marche de l'ordinateur



### Remarque

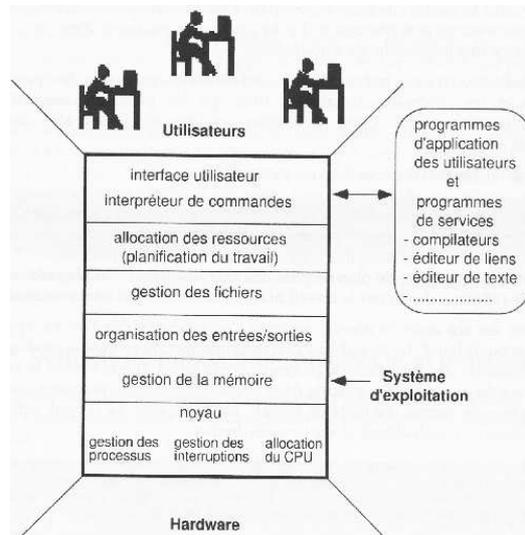
Possible qu'un ordinateur rattaché à un réseau obtienne les progs de l'OS d'un autre ordinateur sur le réseau [*Thin Client*]

### Exemple de services $\in$ et $\notin$ kernel

Langage de commande vs formatage d'un disque

# SYSTÈME D'EXPLOITATION STRUCTURE D'UN OS

## Couche fonctionnelle de l'OS



### Exemple

Un système de gestion de fichier fait appel aux services du système de gestion I/O qui utilise le module de traitement des interruptions, etc.

### Virtualisation de la machine

Une tâche essentielle de l'OS est de présenter une machine virtuelle plus facile à programmer que la machine réelle

Le **Langage de commande** permet à l'utilisateur de formuler ses requêtes au système ou **Interface Graphique [GUI]**

# SYSTÈME D'EXPLOITATION

## OS MONO-TACHE

### OS mono-tache

---

[*Single-job processing OS*]

MS/DOS (ou notre modèle LMC) est un exemple d'un tel OS dans lequel seulement un programme est chargé en mémoire et exécuté

3 majeurs composants résidant en mémoire

1. Langage de commande [*command interface shell*]
2. Programmes gérant les I/O (résidant en ROM connue sous le nom de BIOS)
3. Système de gestion de fichier [*File Management System*]

- En démarrant le MS-DOS, l'OS attend une commande de l'utilisateur en affichant sur l'écran:

*C* : | >

- Une entrée au clavier sera suivi d'un JUMP à la routine système appropriée

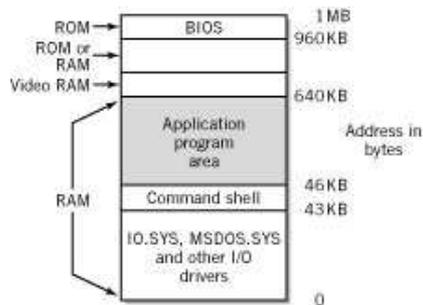
- L'utilisateur peut interrompre un prog. *via* une interruption déclenché par le clavier ou attendre que le prog. se termine et redonne la main à l'interpréteur de commande

# SYSTÈME D'EXPLOITATION

## OS MONO-TACHE

### Désavantages

- Manque de sécurité
- Les programmes peuvent écrire dans la partie réservée à l'OS



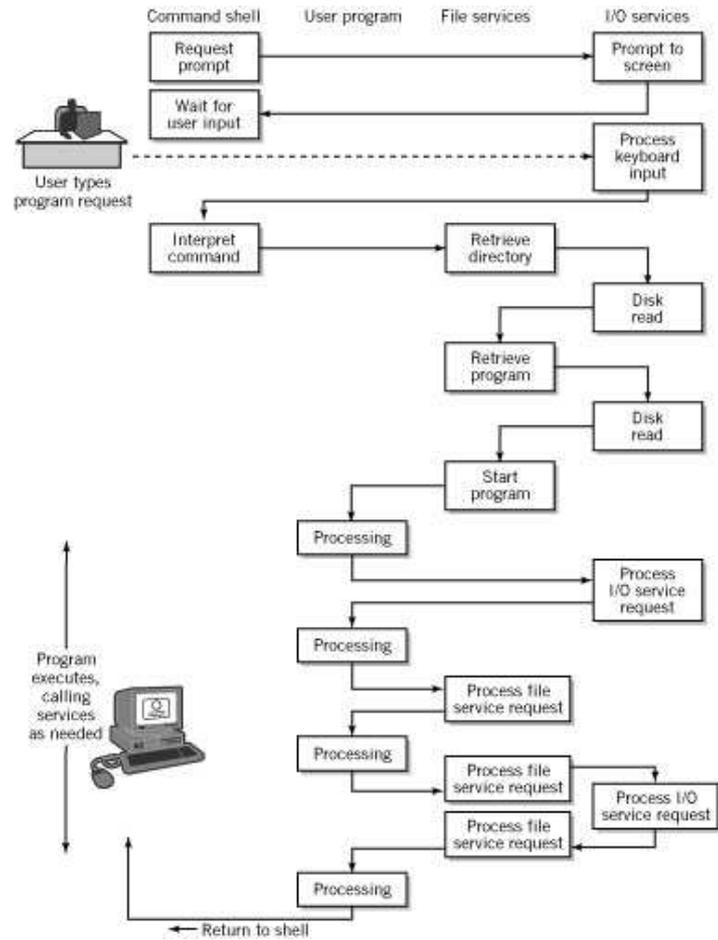
- Les programmes peuvent écrire directement au I/O

▽  
Très sensible aux Bugs

- Le système fournit un minimum de gestion mémoire
- Le CPU passe la majeure partie de son temps à attendre le transfert des données entre CPU et périphériques

# SYSTÈME D'EXPLOITATION OS MONO-TACHE

## Chargement/Exécution d'un programme



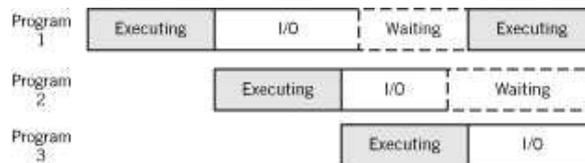
# SYSTÈME D'EXPLOITATION OS MULTI-TACHE

## OS multi-tache

Les OS modernes fournissent les moyens de gérer plusieurs programmes sur un seul CPU, on dit que la machine est multitache ([*multitask*]) ou multiprogrammée [*multiprogrammed*]

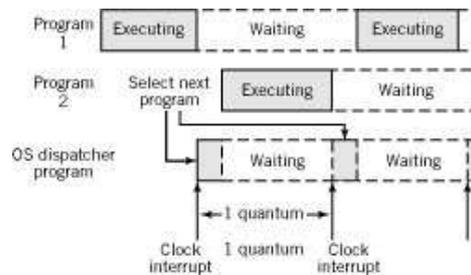
Le multitache peut être appliqué à un simple utilisateur ou peut permettre à de multiples utilisateurs de partager les ressources de l'ordinateur

- Les tps d'attentes du CPU durant les opérations I/O est utilisé pour l'exécution d'un autre prog.



- Le CPU exécute quelques instructions d'un programme puis est *switché* à un autre prog. pour une tranche de tps [*slice of time*]

▽  
[*Dispatching*]



# SYSTÈME D'EXPLOITATION

## SERVICES & FACILITÉS

### Services & Facilités

---

#### 1 • Interface Utilisateur

Type d'interfaces utilisateur

- Interface de Ligne de Commande  
**CLI** [*Command Line Interface*]
- Interface Graphique  
**GUI** [*Graphical User Interface*]

Les ordinateurs modernes ont la possibilité de regrouper ces commandes en *miniprogrammes* (instruction interprétées par l'OS  $\neq$  Langage Ht Niveau)

1. IBM Mainframes - **Job Control Language**
2. MS Windows - Fichier **BAT**
3. UNIX Script Shell (cf. Chapitre Shell Script)

#### 2 • Gestion de Fichier

- Structures en répertoires [*directories*] et sous-répertoires
- Information sur chaque fichier et outils pour copier, effacer, déplacer accéder à un fichier
- Mécanisme de protection
- Qqfois, [*Back-up*], compression, etc.

# SYSTÈME D'EXPLOITATION

## SERVICES & FACILITÉS

### 3 • Entrées/Sorties (I/O)

- logiciels de contrôleur de périphérique (*[I/O device driver]*) (certains sont en ROM)

### 4 • Gestion du contrôle des *process*

Un "Process" est un prog. (App. ou Syst.) en exécution

Un "Thread" est une partie individ. executable d'un process

- État de chaque process (*[running]*, *[ready to run]*, *[waiting]*)
- Gestion des message/connexions entre process (*[pipe]*)

### 5 • Gestion Mémoire

- Allocation d'un espace mémoire & des ressources pour chaque process (empêche l'écriture/lecture dans une zone mémoire non permise)
- Gère la queue des process en attente d'exécution
- Desallocation mémoire lorsque le programme est terminée
- Gère la queue des process en attente d'exécution

# SYSTÈME D'EXPLOITATION

## SERVICES & FACILITÉS

### 6 • Planificateur ([*scheduler*])

Pour les OS multi-tache

– [*High Level Scheduling*] Gestion d'une file d'attente dans laquelle les processus prêt à utiliser sont classés par ordre de priorité (attribué par le planificateur selon l'urgence du traitement et les ressources requises)

– Le [*dispatcher*] alloue le CPU au processus qui se trouve en tête de la queue au moment où il se trouve disponible

Appel au Dispatcher ?

1. Lorsque le processus exécutant déclenche une opération I/O
2. Lorsque la tranche de tps est écoulé
3. Interruption externe (ex: traitement d'une erreur)

### 7 • Gestion Mémoire secondaire

[*Secondary Storage Management*]

– Fournit les [*drivers*] qui contrôleront le transfert entre mémoire secondaire et mémoire principale

– Optimise l'ordre des requêtes I/O (ex: pour un meilleur taux de transfert du disque dur en minimisant son [*seek time*])

## SYSTÈME D'EXPLOITATION SERVICES & FACILITÉS

### 8 • Sécurité & service de protection

[*Security & Protection Service*]

- Protège l'OS des progs d'autres utilisateurs
- Protège un utilisateur d'un autre utilisateur
- Empêche les entrées non-autorisé dans le système (service de login, [*password*])

### 9 • Services Réseau et communication

[*Network and Communications Support Services*]

- Fonctions nécessaire à l'interconnexion de l'ordinateur dans un réseau (protocole TCP-IP)
- Progs d'application et extension (ex: E-Mail, login, etc.)
- [*drivers*] pour modems, communication sans fil, etc.

### 10 • Support pour l'administration du système

L'administrateur système [*sysadmin*] : responsable de l'administration du système

- Configuration du système
- Ajoute ou enlève des utilisateurs/privilèges

## SYSTÈME D'EXPLOITATION SERVICES & FACILITÉS

- S'occupe de la sécurité du système
- Gère le système de fichier
- S'occupe des sauvegardes [*backups*], installation, mise à jour des logicielles et de l'OS
- Optimise le système

L'administrateur système se "logue" en tant que *superuser*, avec des privilèges qui sont au dessus de toute les restrictions et les sécurité défini pour les simples utilisateurs

# SYSTÈME D'EXPLOITATION

## TYPE DE SYSTÈMES

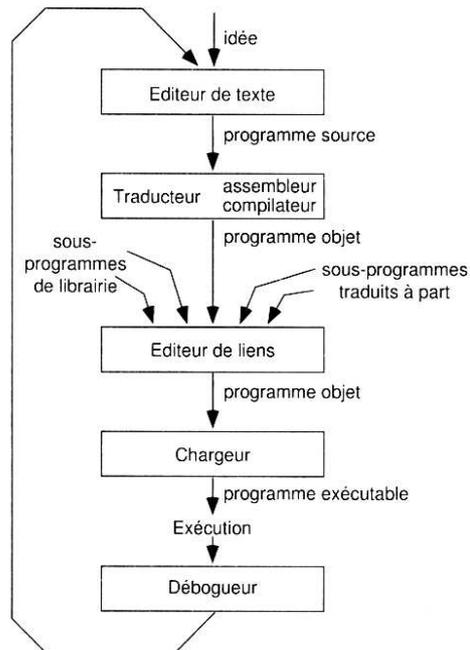
1. **Simple utilisateur, monotache** [*Single user, Single Tasking*]
2. **Simple utilisateur, multitache** [*Single user, multi-tasking*]
3. **Multi utilisateurs, multitache**
4. **Serveur de réseau** [*Network Server*]
  - Utilisé pour gérer le réseau
  - Gère les fichiers, base de données, qqfois support pour démarrage du système (*Thin Client*)
  - Progs exécutes sur les clients
5. **Systèmes temps réel** ([*Real Time System*])  
(réponse impérative dans un délai de tps restreint)
6. **Systèmes embarqués** ([*Embedded Control System*]) système tps réel dédié pour une application particulière (injection automatique pour une auto, programmation micro-onde)  
Logiciel en ROM, caractéristiques d'un système multi-taches (ex: automobile)
7. **Systèmes distribués** ([*Distributed System*])  
La puissance du processeur est distribuée parmi les ordinateurs regroupés en *cluster* ou réseau. Internet peut être utilisé comme système distribué

# OUTILS DE PROGRAMMATION

## INTRODUCTION

### Introduction

Les outils classiques utilisés dans le développement d'un programme sont l'**éditeur de texte**, **traducteur** (compilateur & assembleur), **éditeur de liens**, **débogueur**



### Compilateur [*compiler*]

Le rôle du compilateur est de traduire un fichier programme source en langage de haut niveau (avec lequel le programmeur s'est libéré des détails des instructions machines) en instructions/langage machine

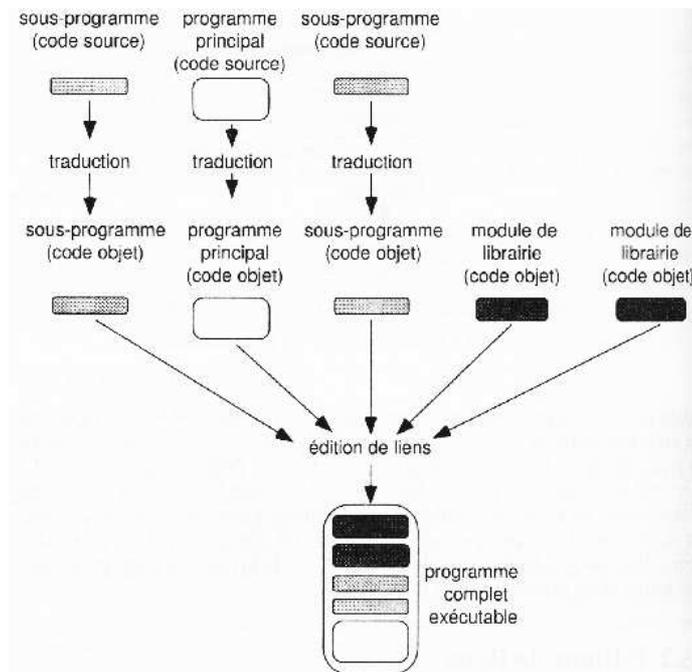
# OUTILS DE PROGRAMMATION

## INTRODUCTION

### Éditeur de Lien [*linker*]

Un éditeur de liens [*linker*], [*linkage editor*] est un logiciel qui permet de combiner plusieurs programmes objet en un seul

Pour pouvoir développer de gros progs, on structure ceux-ci en modules que l'on traduit indépendamment



Des directives particulières permettent d'annoncer à l'assembleur qu'une certaine adresse symbolique est -1- extérieure au module, -2- susceptible d'être appelé depuis l'extérieur (le prog. objet crée 2 tables associés à ces deux types d'adresses)

# OUTILS DE PROGRAMMATION

## INTRODUCTION

### Chargeur [*loader*]

Le programme objet, après édition de lien, doit être chargé en mémoire centrale pour être exécuté

Dans le cas d'un OS mono-tâche, cette partie est triviale (chargeur absolu). Dans le cas d'OS multi-tâche ▷ chargeur relogeable

**Exemple:** *Si l'on veut charger le prog. à partir de l'adresse 2000 ▷ première instruction en 2000 et on ajoute 2000 à tt les adresses*

**Problème ! :** *Comment le loader sait que l'opérande est une adresse ?*

**Réponse :** *L'assembleur va indiquer si le contenu de l'instruction doit être relogé*

Adresse de l'instruction	Contenu	Indic. de Relocation
00043	17160012	1

### Débogueur [*debugger*]

Le débogueur est un logiciel qui facilite la mise au point détection des erreurs dans un prog. Il permet de suivre pas à pas l'exécution d'un prog. en examinant le contenu de la mémoire et des registres

# OUTILS DE PROGRAMMATION

## TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

### Traduction vs Interprétation

*L'exécution d'un Programme s'effectue soit par traduction ou interprétation*

**Interprétation:** Les instructions sont lues les unes après les autres et sont converties en langage machine puis exécuté

**Traduction:** Consiste à générer un programme équivalent au programme source mais codé dans le langage binaire de l'ordinateur (*[machine code]*)

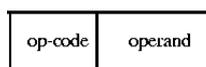
*Traducteur de deux types*

**Compilation:** Traducteur dont le langage source est un langage haut niveau et le code objet est de plus bas niveau tel que le langage machine

**Assembleur:** Traducteur dont le langage source n'est qu'une variante symbolique (le langage d'assemblage) du langage machine et le code objet du code machine

### Langage du Modèle LMC

A mi-chemin entre le le langage machine et le langage d'assemblage



- Mnémonic pour les opcodes
- Pas de mnémonic pour les opérandes

## OUTILS DE PROGRAMMATION TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

Programme du LMC permettant de trouver la  
différence absolue de deux nombres

<pre> (00) START:    INP (01)           STA FIRST (02)           INP (03)           SUB FIRST (04)           BRP GOON (05)           BR DOOUT (06) GOON:     LDA FIRST (07)           SUB SECOND (08) DOOUT:    OUT (09)           COB (10) FIRST:    (used for data) (11) SECOND:   (used for data)           </pre>	<pre> 00 INPUT      00 INPUT 01 STA 10     01 STA 11 02 INPUT      02 INPUT 03 SUB 10     03 STA 12 04 BRP 06     04 SUB 11 05 BR 08      05 BRP 07 06 LDA 09     06 BR 09 07 SUB 10     07 LDA 10 08 OUT        08 SUB 11 09 COB        09 OUT 10 (used for data) 10 COB 11 (used for data) 11 (used for data) 12 (used for data)           </pre>
(a)	(b)

Langage assemblage : on utilise des mnémonics (i.e., noms) pour représenter des noms et des adresses importantes dans le programme (ex: adresse de branchement, donnés, etc.)

### Opération de l'assembleur

On utilise une table de correspondance pour convertir les mnémonics en code machine

Pour le modèle LMC, la règle de traduction est:

code machine = 100 × opcode + opérande (si nécess.)

## OUTILS DE PROGRAMMATION TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

code machine = 100 × opcode + opérande (si nécess.)

		Mnemonic	OP code	Number of operands
(00)	START: INP	ADD	1	1
(01)	STA FIRST	SUB	2	1
(02)	INP	STA	3	1
(03)	SUB FIRST	LDA	5	1
(04)	BRP GOON	BR	6	1
(05)	BR DOOUT	BRZ	7	1
(06)	GOON: LDA FIRST	BRP	8	1
(07)	SUB SECOND	INP	901	0
(08)	DOOUT: OUT	OUT	902	0
(09)	COB	COB	0	0
(10)	FIRST: (used for data)			
(11)	SECOND: (used for data)			

### Traduction en deux passes

- 1
  - Première ligne et vérifie que l'instruction a le format correct (aucune opérande) et START ▷ adresse mémoire 0 (Table des Symboles)
  - Deuxième ligne FIRST inconnue
  - i<sup>ème</sup> ligne ...
- 2
  - Traduction des instruction non traduite dans la première étape

START	00
GOON	06
DOOUT	08
FIRST	10
SECOND	11

## OUTILS DE PROGRAMMATION TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

Lors de la seconde passe, si une adresse référencé n'est pas trouvé dans la table des symbole, alors la traduction indique qu'il y a une erreur dans le code source assembleur

- La première étape de la traduction s'appelle [*scanning*]
- La seconde étape s'appelle le [*parsing*]

Des traductions en une seule passe sont possible mais nécessite de revenir en arrière dans l'étape de [*scanning*]

### Remarque

Jeux d'instructions  $\neq$  d'une machine à une autre  $\Leftrightarrow$  langage d'assembleur  $\neq$

Néanmoins Grande Similarités entre  $\neq$  langage

```
/
/   Program : Add01.pal
/   Date   : May 30, 1997
/
/   Desc : This program computes C = A + B
/
/-----
/
/   Code Section
/
*0200           / code starts at address 0200
Main,  cla cll   / clear AC and Link
        tad A    / load A
        tad B    / add B
        dca C    / store sum at C
        hlt     / halt program
        jmp Main / to continue - goto Main
/
/   Data Section
/
*0300           / data starts at address 0300
A,      2       / A equals 2
B,      3       / B equals 3
C,      0       / C declared
$Main           / End of Program; Main is entry point
```

# OUTILS DE PROGRAMMATION

## TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

### Extensions

Quelques instructions ne font pas partie du langage mais considéré comme des directives [*Pseudo-op codes*]

---

START	Indique à l'assembleur ou commencer l'assemblage
END	Indique ou stopper
ORG <AD>	Indique à l'assembleur que le code suivant cette directive doit être assemblé à partir de l'adresse <AD>
EQV	Ex: TROIS : EQU 03
EXTRN	Indique que cette adresse réside dans un autre module, assemblé séparément
	...

---

### Relocabilité

Pour les ordinateur multi-taches, nécessaire de savoir si l'opérande est une adresse

Adresse de l'instruction	Contenu	Indic. de Relocation
00043	17160012	1

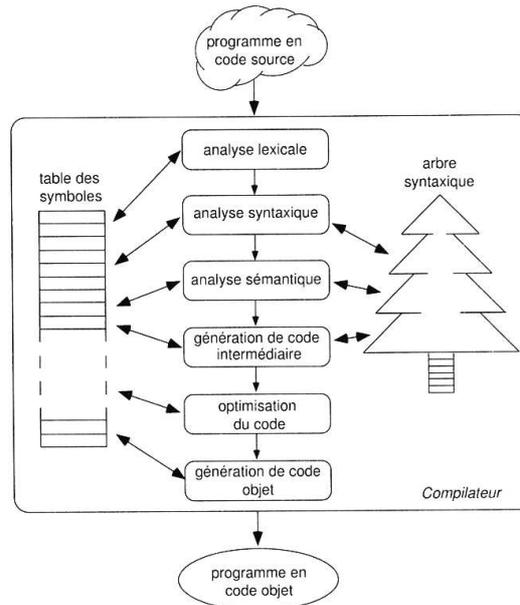
# OUTILS DE PROGRAMMATION

## TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

### Compilation

Toutes les étapes nécessaires pour la traduction d'un programme assembleur en code machine restent valables pour la compilation mais en plus complexe

Le travail du compilateur se divise en plusieurs phases



### Analyse Lexicale

Consiste à lire le programme source et à produire une séquence d'éléments syntaxique (nombres, variables, identificateurs, booléens, opérateurs, affectation, etc.)

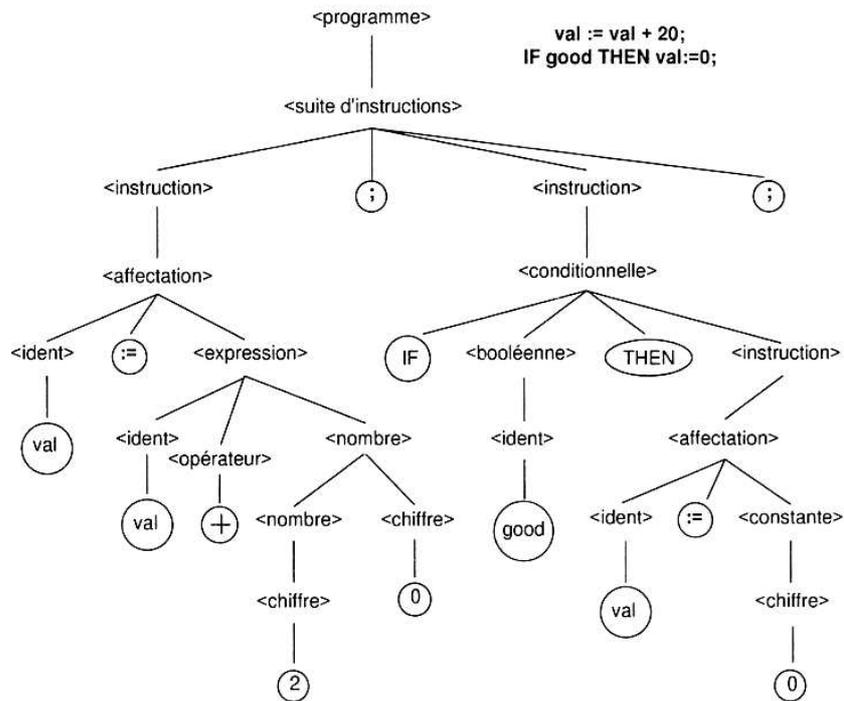
Permet d'identifier quelques erreurs (éléments  $\notin$  langage, nombres illégaux, etc.)

# OUTILS DE PROGRAMMATION

## TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

### Analyse Syntaxique

L'analyseur syntaxique reçoit cette liste d'éléments syntaxique, et génère l'arbre syntaxique du programme



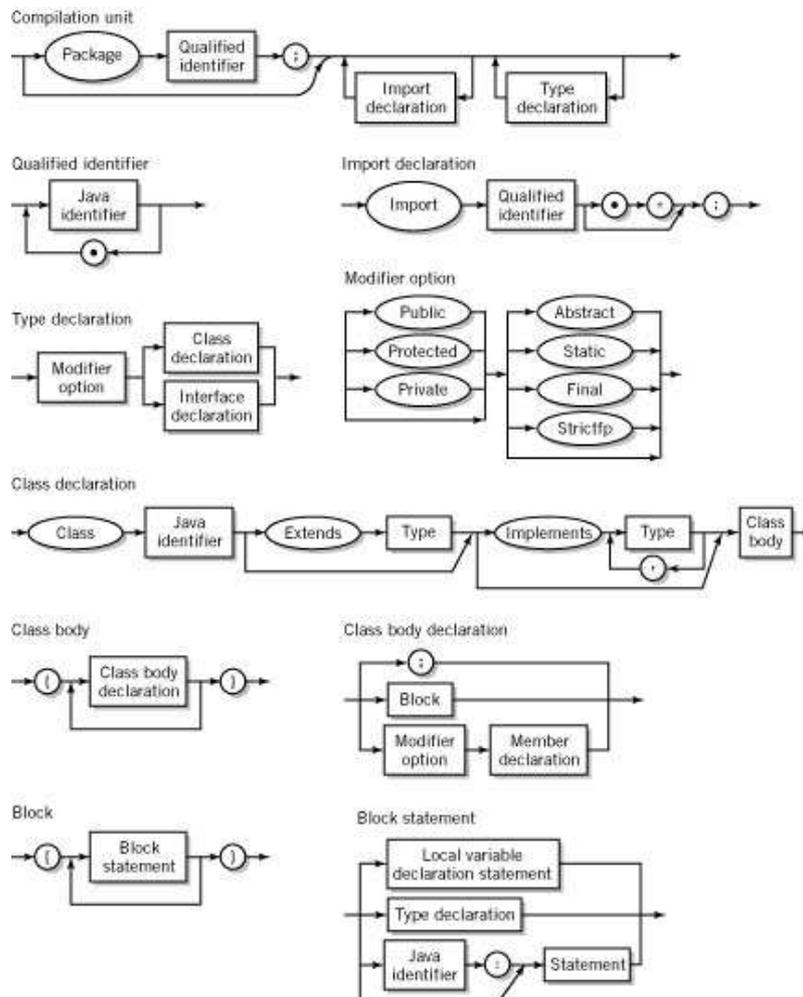
**Approche descendante** : à partir de la racine, appliquer les règles qui permettent de remonter jusqu'à la phrase désirée

**Approche ascendante** : à partir des élément syntax., trouver toute les règles qui permettent de remonter à la racine

# OUTILS DE PROGRAMMATION

## TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

[Railroad Diagrams]



## OUTILS DE PROGRAMMATION

### TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

#### Analyse Sémantique

Consiste à vérifier la concordance des types, ce qui revient à vérifier que chaque opérateur travaille sur des opérandes qui sont autorisés par le langage (utilise la table des symboles)

Ex.:

val := val + 20  
val doit être un entier (cf. Table des Symboles)

if *Booléen*

THEN val := 0

la suite de THEN peut être une affectation,  
ou une/une série d'instruction(s)

...

#### Génération de Code Intermédiaire

Le code intermédiaire doit

1. Être facile à produire à partir de l'arbre syntaxique
2. Être facile à traduire en code objet

Le code intermédiaire est un flux d'instructions simples (macros) qui ne font pas de référence aux registres de la machine cible

## OUTILS DE PROGRAMMATION

### TRADUCTEUR - COMPILATEUR - ASSEMBLEUR

#### Optimisation de Code

Consiste à améliorer le code généré pour le rendre plus rapide à l'exécution et moins encombrant en mémoire

- Éliminer les redondances pour réduire le nb final d'instruction

```
tmp1 := 3,1415926 × 100,0
tmp2 := RealToInteger (tmp1)
tmp3 := id1 + tmp2           id : identificateur
tmp4 := tmp3 × id2
id3  := tmp4
```

#### Code optimisé

```
tmp1 := id1 + 314
id3  := tmp1 × id2
```

- Phase délicate particulièrement déterminante pour les machines RISC (compilateur tirant profit d'un grand nb de registres)

#### Génération du code objet

Génération de code objet (propre à la machine) *relocable*, i.e., relatif à l'origine 0