# 5. Couche conventionnelle. Introduction à la programmation en assembleur et en langage machine. Instructions - types, formats, champs; types d'adressage et utilisation des registres. Branchements, sauts et interruptions. Structuration du logiciel et algorithmes. Exemples de programmation en assembleur.

## *5.1. Couche conventionnelle*

La couche conventionnelle permet de décoder et d'exécuter les instructions de la couche microprogrammée. L'ensemble des instructions possibles est appelé le *jeu d'instructions*, qui peut varier d'une architecture à l'autre.

Chaque instruction (un mot, soit un champ de bits) est séparée en plusieurs parties:

- l'instruction ;
- le premier opérande ;
- le deuxième opérande.

## *5.2. Introduction à la programmation en assembleur et en langage machine*

On distingue cinq générations de langages actuels à programmation.

Les langages machine forment la **première** génération. Le langage machine est élémentaire à programmer, constitué de longues colonnes avec zéros et uns (0,1), qui puis être accomplit du processeur central sans aucunes transformations après. Dans les langages machine les instructions (ou commandes) sont présentées en code binaire et sont accomplies directement dans le matériel (hardware) du processeur central. On ne les utilise plus dès le début des $80^{th}$ années.

La **seconde** génération est composée des *langages de symbole à programmer* qui sont nommés *langages d'assemblage ou assembleurs* (les premiers apparus en 1940 année). On doit marquer tout de suite que ces langages sont toujours utilisés, mais rarement pour des programmes complets. On les utilise avec les langages des générations suivantes, pour des parties de programmes – sous-programmes – quand on cherche une plus haute vitesse de remplissement de ce sous-programme. Leur défaut général est qu'ils dépendent du matériel de base de l'ordinateur - c'est-à-dire qu'assembleur concret puis être appliqué sur un seul type du matériel de base de l'ordinateur. D'ici les programmes (logiciels), composés en assembleur, ne puissent être transportés directement sur un autre type de matériel de l'ordinateur. Ou autrement dit, ces langages sont nommés aussi *les langages objectifs* pour programmer. D'ici il provient, qu'on exige du programmeur d'avoir des notions fondamentales solides sur l'ordinateur d'exécution (d'objet) pour lequel on va constituer des programmes en assembleur. D'ici la conclusion que l'assembleur est et reste d'être un langage pour les professionnels.

Pendant les $50^{th}$ années du siècle passé commencent à naitre ainsi nommés *les langages à programmer du haut niveau* ou autrement appelés *langage à orientation de problème pour programmer*. Ces langages sont distribués de la **troisième** à la **cinquième** génération et ils sont indépendants du matériel de base de l'ordinateur.

Les langages à programmer de **la troisième** génération sont nommés encore *les langages de procédure* : **C, Pascal, FORTRAN, BASIC, COBOL, Java**.

Les langages à programmer de **la quatrième** génération sont nommés encore *les langages de non-procédure* : **Excel, Lotus 1-2-3, Oracle, Symphony**.

Les langages à programmer de **la cinquième** génération sont nommés encore *les langages d'intellect artificiel et langages orientés à objets* : **Lisp, Prolog, SmalTalk**.

## *5.3. Instructions - types, formats, champs; types d'adressage.*

### *5.3.1. Présentation des nombres*

L'assembleur prévoit la possibilité de présenter les nombres dans les systèmes : binaire, hexadécimal ou décimal. Dans l'assembleur tout nombre est présenté par son correspondent ASCII symbole. Ou autrement dit les nombres dans l'assembleur se présentent avec leur ASCII code.

### *5.3.2. Mnémonique*

Le langage d'assemblage est un langage, orienté à machine. Chez lui les colonnes longues des chiffres - 0 et 1 laissent la place des abréviations de symbole. Ces abréviations sont nommées ***mnémonique*** – un mot, créé artificiellement des mots anglais : ***memory*** – mémoire et ***name*** – nom. Dans la base de la mnémonique ou son principe est que la fonction d'un symbole donné est mémorisée dans la mémoire de l'homme plus facilement par un nom de symbole.

### *5.3.3. Instruction – Commande dans l'assembleur*

Toute commande – ou instruction dans l'assembleur *répond juste* à une commande à machine du processeur donné. Comme résultat de cela, le principe de fonction de l'assembleur est une transformation absolue des symboles du langage en instructions – commandes du processeur central. Donc l'assembleur, comme on ne pratique plus d'un demi-siècle le langage à machine, représente un outil élémentaire et universel pour développer le logiciel. Son avantage (sa priorité) en comparaison des autres langages orientés en problèmes est la construction de sous-programmes très compacts, exécutés bien vite, grâce à la réaction directe du matériel de l'ordinateur concret. C'est-à-dire que l'assembleur apparait irremplaçable pour des sous-programmes, critiques à vitesse, qu'on rencontre dans le logiciel systématique. Ce sont à fond des problèmes dans le développement des systèmes opérationnels ou de contrôle, ces derniers pour différentes unités externes de l'ordinateur.

### *5.3.4. Syntaxe des instructions (commandes)*

La programmation en assembleur signifie que toute commande-instruction doit répondre à un mot à machine du processeur central. La commande (l'instruction) s'inscrit avec des mots courts et compréhensibles ou abréviations de ces mots en anglais. Ces mots ou abréviations sont nommés *symboles*. Ainsi grâce à ce syntaxe des commandes le langage d'assemblage est nommée la langage de symbole. Et pour rappeler encore une fois on doit connaitre en détails l'architecture de l'ordinateur. Pour utiliser complètement ce langage de symbole, quelques-uns des langages de niveau haut prévoient une possibilité de branchement des fragments, écrits en assembleur.

Les commandes dans le langage d'assemblage sont écrites en *lettres majuscules*, en utilisant le langage à symboles. Ainsi les commandes les plus utilisées sont :

***MOV***-du mot anglais ***Move***, pour transférer ou copier ;

***ADD***- du mot anglais ***Addition***, pour l'opération arithmétique addition ;

***ADC***-des mots anglais ***Addition + Carry***, pour l'opération arithmétique addition avec retenue (report) ;

***JMP***- du mot anglais ***Jump***, pour la commande de transition inconditionnelle ;

***JZ***- de la phrase en anglais ***Jump if Zero***, pour la commande de transition conditionnelle.

## *5.4. Branchements, sauts et interruptions*

Un programme, écrit à l'Assembleur, est composé des commandes (instructions), écrites à l'aide du code mnémonique et à l'aide des ***directives***. Les directives présentent les indices (signes) de l'Assembleur. Les instructions sont traduites en code objectif et les directives commandent le processus de la traduction. Les directives traduisent les instructions spécifiques pour le consommateur - nommées macro-instructions en une série d'instructions à machine.

Les directives le plus souvent rencontrées pour l'Assembleur pour les processeurs **Intel** sont écrits en majuscules et sont les suivantes :

***SEGMENT –*** Directive pour le début de tour segment du programme ;

***ENDS –*** Directive pour la fin du segment ;

***END –*** Directive pour la fin du programme ;

***ASSUME –*** Directive pour l'adresse de base de l'étiquette dans la langue d'assemblage.

Ici on doit définir la notion ***étiquette*** dans la lange d'assemblage :

En utilisant l'Assembleur il n'est pas nécessaire de signaler des adresses dans la mémoire opérationnelle de l'ordinateur. On peut utiliser des adresses symboliques, qui sont nommées encore ***étiquettes*** – le mot anglait de cette notion est ***labels***. La notation de l'étiquette est la suivante : ***M1 :*** les deux points marquent, que M1 est une étiquette.

## *5.5. Structuration du logiciel et utilisation des registres. Algorithmes.*

### *5.5.1. Structuration du logiciel et utilisation des registres internes du processeur central :*

La mémoire opérationnelle de l'ordinateur pour tout type de processeur central est divisées en plusieurs parties, nommées segments (ou pages). On montre, par l'exemple suivant – Figure 16, la création de tout programme, étroitement lié avec la structure matérielle (hardware) du processeur central. On prévoit les segments suivants, montrés sur cette figure. Ainsi tout programme créé est composé de plusieurs segments. Tout segment est bordé de deux directives : il commence avec la

directive ***SEGMENT*** et finit avec la directive ***ENDS***. Tout segment est nommé par le programmeur, indépendamment de son type, que ce nom n'est pas transmis sur le programme objectif. Q la fin du programme doit apparaitre la directive ***END***. Cette directive montre à l'Assembleur, que le programme a fini - c'est-à-dire qu'à cette place l'Assembleur puit finir la traduction des instructions. Après le mot ***END*** l'adresse de début (START) puit être introduit. Lorsque le programme d'exécution est chargé, à cette place elle est startée automatiquement.

| |
|---|
| **Segment de code**<br>**CS** |
| |
| **Segment à données**<br>**DS** |
| |
| **Segment auxiliaire pour le moment**<br>**ES** |
| |
| **Segment à pile (stack)**<br>**SS** |
| |
| **Segments auxiliaires**<br>**(pour un concret type de processeur)**<br>**FS, GS, etc.** |

**Figure 16. Modèle en segment du processeur central**

Donc le programme en Assembleur a la structuration suivante :

***Nom du segment SEGMENT***
---
---
***Nom du segment ENDS***

***END***

### *5.5.2. Algorithmes :*

L'algorithme d'un programme concret représente un bloc-schéma de ce programme. Un exemple d'un algorithme du programme concret est composé de plusieurs blocs obligatoires :

- ***DEBUT – START*** du programme ;
- ***TRANSITION INCONDITIONNELLE*** dans le programme ;
- ***TRANSITION CONDITIONNELLE*** dans le programme ;
- ***FIN –STOP*** du programme.

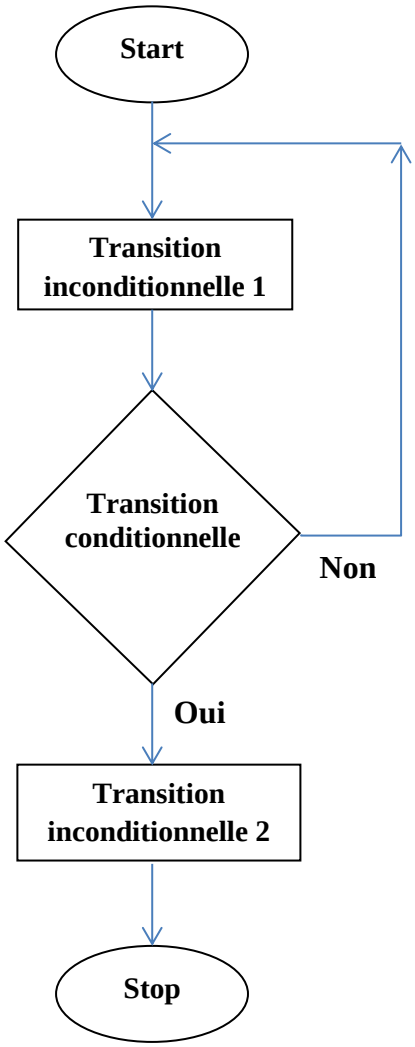


**Figure 17. Algorithme du logiciel – modèle d'organigramme (de schéma fonctionnelle)**

## *5.6. Exemples de programmation en assembleur*

L'assembleur, comme un langage à machine, opère en régime interactif d'introduction de commandes (instructions). Toute commande introduite se transforme en code à machine et est inscrite dans la mémoire opérationnelle à une adresse concrète. Tout code valide de l'opération – COP, est transformé en mnémonique de la commande et un opérande. Tous les COP non-valides apparaissent sur l'écran comme ILLOP (ILLégale Opération).

Tout programme de commande (programme de contrôle) pour fonctionner avec l'assembleur est écrit (mémorisé) sur la mémoire constante (type ROM) de l'ordinateur. Il est composé de commande (instructions) pour fonctionner avec le programme en Assembleur pour le concret processeur central.

La création des programmes en Assembleur exige l'exploitation des moyens suivants :

### *5.6.1. Editeur :*

C’est un programme, largement utilisé pour traiter le texte. En programmation sur l’Assembleur il est nécessaire que le fichier de sortie doive être textuel - c’est-à-dire que les symboles doivent être présentés en code ASCII.

### *5.5.2. Assembleur :*

C’est un programme, qui accepte le programme, créé à l’aide de l’Editeur et le transforme en programme objectif, qui directement puisse être exécuté. Lorsqu’on opère avec l’Assembleur de Microsoft, nommé MASM, on doit savoir que cet Assembleur donne au programmeur de définir ces propres instructions, nommées macro-instructions. D’ici le nom de MASM : Macro-Assembleur.

### *5.5.3. Link :*

Ce programme transforme le programme objectif en programme déplaçable. Cela permet au système opérationnel (qui commande les ressources de l’ordinateur) de positionner ce programme déplaçable sur une place convenable de la mémoire opérationnelle. Ainsi l’opérateur se libère d’une fonction, quelquefois bien difficile.

### *5.5.4. Debug :*

A l’aide de ce programme est exécuté l’accordage des nouvelles réalisations programmées. Souvent il est utilisé pour découvrir les fautes des nouveaux programmes réalisés. Le programme représente un programme instrumental, branché dans le complet de programmes du système opérationnel (DOS). Il est utilisé pour vérification et changement :

- ✓ Si une faute est découverte,
- ✓ Des fichiers binaires de programme.

C’est-à-dire que ce sont des fichiers exécutables, contenant des instructions à machine. Pour ce but ce programme se retourne sur des cellules de la mémoire opérationnelle par leurs adresses, données en deux nombres, chacun de deux bytes. Le premier nombre de deux bytes est l’adresse du segment et le second nombre de deux bytes, nommé adresse relative, présente l’adresse de la cellule dans le segment déjà choisi par le premier nombre. Les deux adresses sont écrites , divisées par deux points.