PROGRAMMATION STRUCTURÉE ET LANGAGES DE PROGRAMMATION

I/ REPRÉSENTATION ET COMPOSITION D'UN PROGRAMME

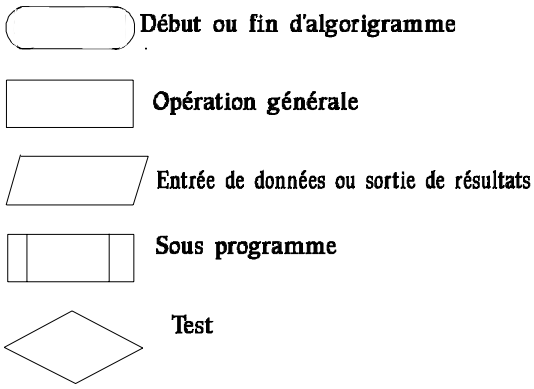
Pour résoudre un problème informatique, il faut trouver une méthode de traduction du problème qui facilite sa compréhension et qui soit proche du langage compréhensible par la machine (*le microprocesseur ou plus généralement l'ordinateur*). Cette méthode est appelée l'***algorithme***.

A l'heure actuelle deux méthodes sont utilisées :

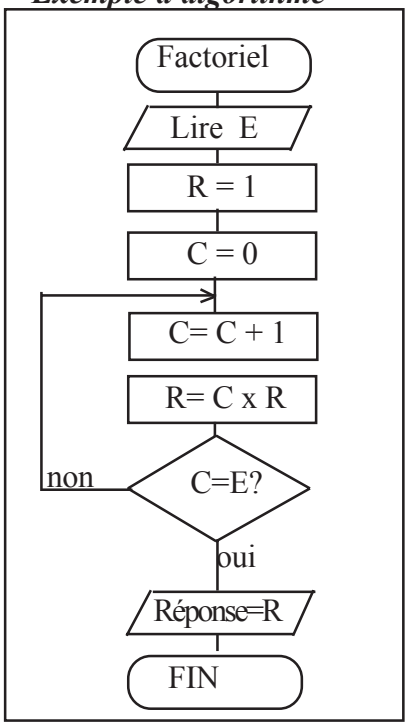
- l'***algorigramme*** en voie de disparition mais encore utilisé en milieu industriel
- l'***algorithme***

I.1/ L'Algorigramme

Il représente graphiquement un problème au moyen de symboles normalisés. Ce graphique se lit du haut vers le bas



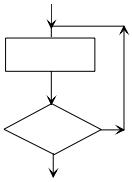
Exemple d'algorithme



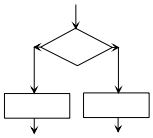
Traitement d'instructions conditionnelles

Remarque préliminaire : pour éviter le blocage (*plantage*) de la machine, il faudra toujours s'assurer que la condition pourra être atteinte.

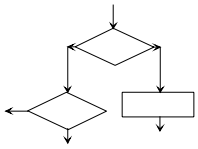
- **La boucle :**
  Elle permet d'effectuer *n* fois la même opération (*ou ensemble d'opérations*) jusqu'à ce que la condition du test soit remplie.



- **Le traitement maillé :**
  Suivant le résultat d'un test on effectuera une succession d'instructions dans une maille ou dans l'autre.



- **Le traitement arborescent :**
  Plusieurs conditions de test se suivent



I.2/ L'algorithme

Il traduit en langage clair les opérations à réaliser. Il est particulièrement adapté à tous les langages de programmation structurée (*Pascal, C, Forth...*).

•Traitement d'instructions conditionnelles

a) SI *Condition*
| **ALORS** *Seq1*
| **SINON** *Seq2*
FINSI

Remarque : Seq2 peut ne pas exister, à ce moment là, le traitement devient:

SI *Condition*
| **ALORS** Seq1
FINSI

Exemple en assembleur 68HC11

	CMPA *#$25*	SI *A=$25*
	BEQ *Seq1*	| ALORS Seq1
Seq2	**CLRA**	| **SINON Seq2**
Seq1	**STAA** *$00*	FINSI

Exemple d'algorithme

```
FACTORIEL
|   Lire la donnée E
|   R=1
|   C=0
|   REPETER
|   |   C=C+1
|   |   R=R x C
|   JUSQU'A C=E
|   Réponse = R
FIN
```

b) TANTQUE *Condition*
| **FAIRE** *Seq*
FINTQ

Exemple :

	LDAA *#$10*	A=$10
DebutBoucle	**BEQ** *FINBoucle*	**TANTQUE** A<>0
	Inst1	**FAIRE**
	Inst2	| Inst1
	DECA	| Inst2
	BRA *DebutBoucle*	| A = A-1
FinBoucle	*Inst3*	**FINTQ**

c) RÉPÉTER
| *seq1*
JUSQU'À *Condition*

Exemple :

	LDAB *#$0F*	B=$0F
LA	*Inst1*	*REPETER*
	Inst2	| *Inst1*
	DECB	| Inst2
	BNE *LA*	| B=B-1
		JUSQU'A B=0

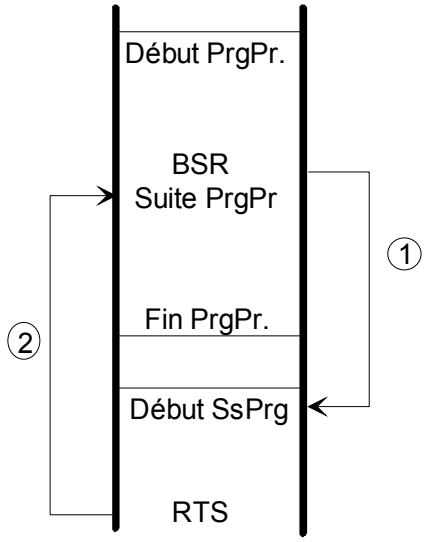
II/ SOUS-PROGRAMME

Au cours d'un programme, on a souvent besoin d'effectuer à des instants différents une même suite d'opérations. Cette suite, qui forme un ensemble indissociable est appelé ***sous-***

programme.

Plutôt que de réécrire une suite d'instructions plusieurs fois dans un programme, il est plus commode de les ranger une seule fois dans un sous programme en mémoire et de l'appeler à chaque besoin par des instructions spécifiques (ex ***BSR*** *en assembleur 68HC11*)

La gestion de ces sous programmes est telle qu'à la fin de leur exécution le programme principal se retrouve à l'instruction suivant l'appel du sous-programme.



Remarques :

1/ Les sous programmes existent également dans les langages évolués comme le Pascal (***FUNCTION*** ou ***PROCEDURE***), le C(***VOID***) ou d'autres.

2/ Un sous-programme peut appeler un autre sous-programme. Dans ce cas le retour au programme principal se fera lorsque tous les sous-programmes auront été terminés

III/ LANGAGES DE PROGRAMMATION

Pour écrire les programmes, le programmeur dispose de plusieurs codes ou langage de programmation.

On distingue :

- Les langages élémentaires : binaire, assembleur
- les langages évolués : C, Pascal, Basic, Algol, Fortran, GRAFCET...

Le micro-ordinateur ne peut traiter que des mots sous forme binaire. Ce programme écrit sous forme binaire est appelé le ***langage machine***. Il n'est plus utilisé par les programmeurs étant donné que des logiciels "*d'assemblage*" le génèrent à partir du langage assembleur ou évolué.

Le programme écrit en langage non compréhensible directement par la machine (*ex : langage évolué*) est appelé ***programme source***.

Ce programme source traduit en langage machine deviendra le ***programme objet*** implanté en mémoire et directement exécutable par la machine.

III.1/ Langage assembleur

Dans ce langage, les instructions sont représentées par des abréviations alphabétiques de terme anglais appelés ***mnémoniques.*** Ils sont propres à chaque microprocesseur.

La présentation du programme est divisée en différentes colonnes appelées champs.

Exemple :

Adresses	Etiquettes	Code Op	Opérandes	Commentaires
		ORG	*$00F0*	origine du programme en
$00F0	*Début*	**LDAA**	*#$4E*	chargement de A par $4E
$00F2	*Alp*	**DECB**		B=B-1
$00F3		**BNE**	*Alpha*	Si B différent de 0 aller à Alp
$00F5		**END**		

On distingue :

♦**le champ adresse :** les adresses successives des instructions y sont portées. Il représente la localisatrion de la ligne d'instruction dans l'espace mémoire

♦**Le champ étiquette** : l'étiquette ou label est une abréviation qui désigne symboliquement une adresse ou encore un sous-programme, une boucle de programme.

♦**le champ code opération :** il contient le code opératoire des instructions.

♦**le champ opérande :** il comporte l'opérande de l'instruction ou son adresse

Remarques :

Le programme source écrit en assembleur est traduit en programme objet (langage machine) par un programme lui aussi appelé "***ASSEMBLEUR***" (ou ***compilateur***). Cette opération est l'***assemblage.***

Directives d'assemblage (ou pseudo-instructions)

•**ORG :** La première ligne du programme précédent n'est pas une instruction exécutable par la machine : c'est une directive d'assemblage qui fournit à l'assembleur l'adresse de début du programme ($00F0 dans l'exemple) lui permettant ainsi de calculer les adresses des instructions du programme.

•**EQU :** Permet d'attribuer à un mot personnalisé, une valeur particulière. Dans la suite du programme , le mot sera automatiquement remplacé par l'assembleur par sa valeur.

•**END :** spécifie la fin du programme. Tout code suivant cette commande ne sera plus examiné par l'assembleur.

III.2/ Langages évolués

Les langages évolués possèdent des instructions similaires aux expressions du langage courant et mathématique habituels.

Chaque instruction de langage évolué est équivalente à plusieurs instructions machines.

Les plus courants sont : C, Pascal, Basic, Fortran, COBOL.... chacun possède des spécificités propres.