République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Université Abderrahmane Mira (Bejaïa)
Faculté de Technologie
Département de Génie Electrique

Polycopié du Cours

Automates Programmables Industriels et Microprocesseurs (API et µp)

Cours de 2éme année Master- Electromécanique

Réalisé par :
Dr. DJERMOUNI Kamel

2022

BALISES DE PLAN DU COURS

CHAPITRE I : RAPPELS SUR LE GRAFCET

CHAPITRE II : AUTOMATE PROGRAMMABLE INDUSTRIEL –API-

CHAPITRE III : MICROPROCESSEURS, MICROCONTROLEURS ET DSP

BALISES DE PLAN DU COURS

BALISES DE PLAN DU COURS

1- Information sur le cours

- **Faculté** : de Technologie- Département de Génie Electrique
- **Public cible** : 2eme année Master, spécialité Electromécanique
- **Cours** : Microprocesseurs et Automates Programmables Industriels (API)
- **Code du cours** : **UEF 2.1.2**
- **VHS: 45h00 (un cours de 1h30 et un TD de 1h30 par semaine)**
- **Crédits: 4**
- **Coefficient: 2**
- **Horaire du cours** : Selon l'emploi du temps semestriel
- **Salle de cours** : Selon l'emploi du temps semestriel
- **Enseignant** : Djermouni Kamel
- **Coordonnées de l'enseignant** : Mail- djermounikamel@yahoo.fr, Tel : **0660861561**
- Disponibilités de l'enseignant :
  - ***Au bureau (Laboratoire de Maitrise des énergies renouvelables)*** : Le jour et l'heure sont variable selon la disponibilité de l'enseignant, chaque semestre;
  - ***Réponse sur le forum*** : toute question en relation avec le cours postée sur le forum, elle va avoir une réponse au plus tard après 72 heures ;
  - ***Par mail*** : toute question en relation avec le cours envoyé par mail, elle va avoir une réponse au plus tard après 48 heures.
- **Mode d'enseignement** : Hybride (présentiel et en ligne)

2- Présentation du cours

Dans le domaine de l'automatisation des processus industriels, l'évolution des techniques de contrôle/commande s'est traduite par : un développement massif, une approche de plus en plus globale des problèmes et une intégration dès la conception de l'installation.

On est ainsi passé du stade de la machine automatisée à celui du système de production complétement automatisé. L'Automate Programmable Industriel (API) est un appareil électronique programmable, adapté à l'environnement industriel, qui réalise des fonctions d'automatisme pour assurer la commande de pré-actionneurs et d'actionneurs à partir d'informations logique, analogique ou numérique.

Le microprocesseur est l'unité principale qui prend les discisions à exécuter suivant le programme stocké dans la mémoire ROM. La RAM est utilisée pour exécuter le programme.

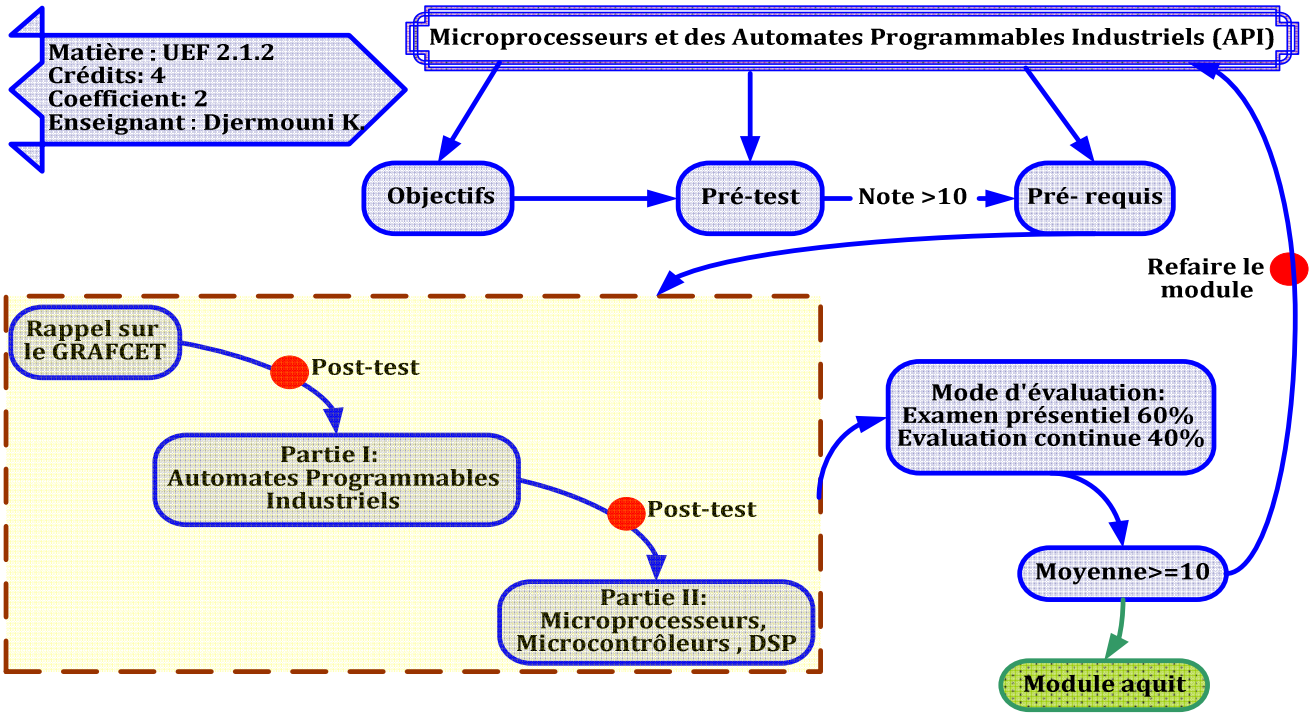
Ce module est divisé en deux grandes parties précédées d'un court rappel, nécessaire et utile, sur le **GRAFCET** afin de remédier au manque et aux difficultés enregistré chez les étudiants qui arrivent chaque année :

a- La première partie permet à l'étudiant d'apprendre : les composants d'un API, la programmation d'un API, comment raccorder un automate, accéder à ses fonctions, identifier des problèmes de fonctionnement d'un automatisme simple commandé par un automate programmable, apporter des modifications au programme et effectuer l'essai de l'automatisme ;

b- La deuxième partie permet à l'étudiant d'avoir une idée générale sur :

- Le premier processeur ou calculateur spécialisé en traitement du signal, le processeur de traitement numérique du signal "DSP" (Digital Signal Processor), en prendront l'exemple du TMS320 C6x ;

- Le fonctionnement et l'utilisation des microcontrôleurs, qui sont devenus aujourd'hui des composants électroniques clé et incontournables pour tous systèmes automatisés, en prendront l'exemple du 68HC11E2 ;
- Le microprocesseur ancêtre et des microprocesseurs actuels tout en essayant : de traiter les architectures et les fonctions de base des microprocesseurs, de connaitre les différents types de mémoires et d'étude la programmation du 8086 d'Intel ainsi que la manière dont ce dernier s'interface avec les modules externes tels que l'interface parallèle 8255 et l'interface série 8250.



3- Contenu du cours

Partie 1. Automates Programmables Industriels (API) (08 semaines)

- ✓ Architecture des API : Organisation, entrée-sortie, mémoire, bus ;
- ✓ Choix et câblages des API : Caractéristiques, environnement, évaluation ;
- ✓ Logiciels de programmation des API : GRAFCE, langages de base, de calcul et séquentiel ;
- ✓ Applications : Automatisations des ascenseurs, des pompes, des systèmes de ventilation, des compresseurs, des mécanismes de transport continu, des machines-outils.

Partie 1. Microprocesseurs (07 semaines)

- ✓ Microprocesseurs à usage général, architecture ;
- ✓ Processeurs de traitement numérique du signal DSP ;
- ✓ Microprocesseurs : Mémoires, dispositifs d'entrées/sorties, modes d'échanges d'informations;
- ✓ Microcontrôleurs : Processeurs de traitement numérique du signal, programmation ;
- ✓ Exemples de processeurs disponibles sur le marché.

4- Prérequis

Les prérequis demandés sont basés essentiellement sur la logiques combinatoire (les systèmes de codage, les tables de Karnaugh,…) et séquentielle (Grafcet) et automatismes.

Un test sur les prérequis cités ci-dessus est disponible sur elearning.univ-Bejaia.dz/ et de choisir la rubrique département génie électrique, le niveau Master 2, semestre 3. L'étudiant à 20 questions chaque question est notée sur un point (01pt ou 5%), sil aura un score supérieur

ou égale à 10 points (50%), il sera jugé capable de suivre ce cours. Dans le cas contraire, l'étudiant sera obligé de se documenter pour enrichir ses connaissances dans les sujets abordés dans le pré test et de me contacter pour une bonne orientation.

Place du cours dans le programme

Ce cours obligatoire et indispensable s'adresse à tous les étudiants du Master II en Electromécanique.

5- Visées d'apprentissages

Globalement l'objectif de ce cours est la connaissance de fonctionnement et la mise en œuvre des Microprocesseurs et des Automates Programmables Industriels (API) en vue de développer des circuits de commande avancés.

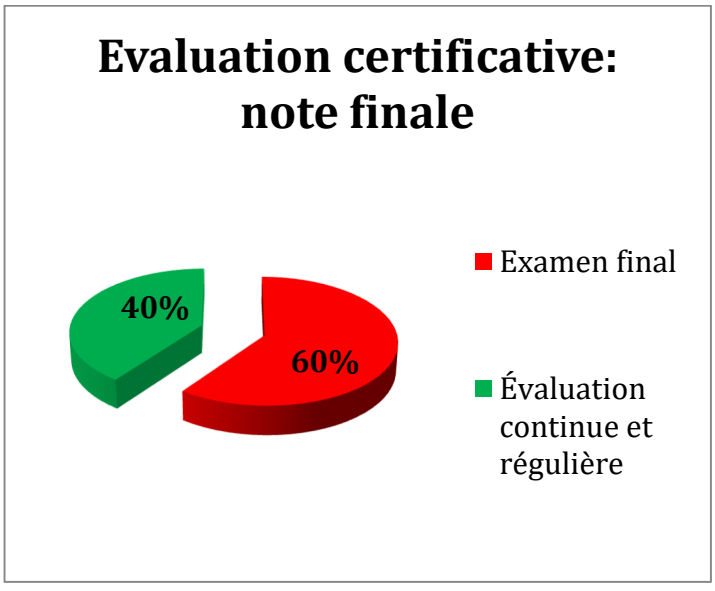
a- En termes de connaissances, comprendre les principes de fonctionnement d'un API et microprocesseur, et identifier les différents composants de chaque système.

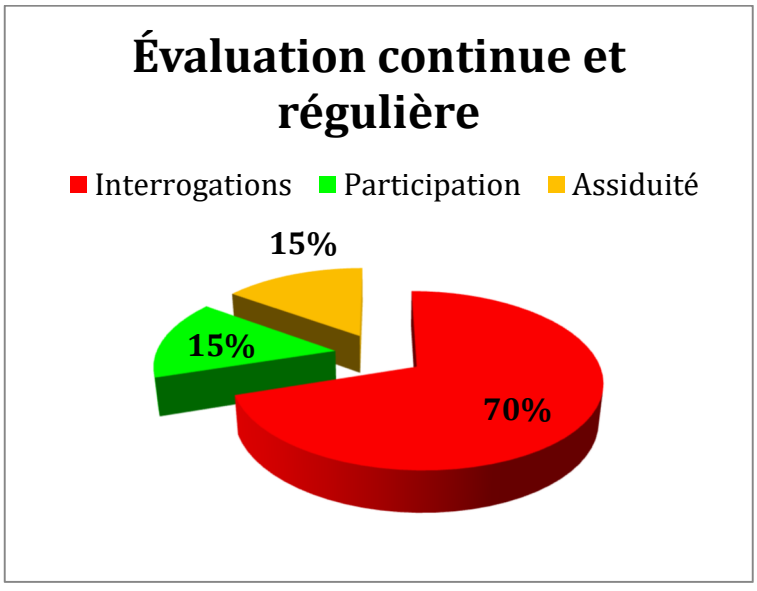
b- En termes de savoir-faire, raccorder un automate, élaborer un programme d'un automatisme commandé par un automate, identifier des problèmes de fonctionnement d'un automatisme simple commandé par un automate, utiliser un logiciel de programmation d'un microprocesseur et effectuer l'essai d'un automatisme simple commandé par un automate.

c- En termes de savoir-être, respect des règles d'application et de sécurité au travail et respect des normes recommandées.

6- Modalités d'évaluation des apprentissages

- Dans le cadre de ce cours, il est prévu que l'étudiant doit se présenter à un examen final sur table qui aura lieu sous surveillance :
  - Type de l'évaluation : Sommative
  - Mode de travail : Individuel
  - Pondération : 60 %
  - Directives de l'évaluation : Questions de cours (20%) et exercices d'analyse et de calcul (80%)
- Évaluation continue et régulière (40%), c'est une évaluation formative et sommative qui se compose de :
  - **a-** Deux interrogations :
    - Mode de travail : Individuel ;
    - Pondération : 70 % de l'évaluation continue.
  - **b-** La participation : 15% de l'évaluation continue ;
  - **c-** L'assiduité : 15% de l'évaluation continue.





N.B : Les note des travaux pratiques programmés (TPs) et indépendante puisque ils sont considérés comme un module indépendant (unité méthodologique).

7- Activités d'enseignement-apprentissage

Le cours : Microprocesseurs et Automates Programmables Industriels (API) est conçu selon une approche pédagogique, soit une partie en présentiel et une partie à distance par Internet. Il y aura une évaluation en salle.
Selon les thématiques abordées dans chacune des séances, les activités d'apprentissage suivantes devront être réalisées.

- Exposés et discussions en salle et sur le forum ;
- Lectures obligatoires ;
- Plusieurs types de documents sont disponibles directement en ligne: textes et articles, extraits d'ouvrages ;
- Les documents audiovisuels viennent compléter les contenus présentés à l'oral et à l'écrit. Il est important de les visionner sur **youtube** ;
- Forum de discussion sur **e. elearning** :
- L'étudiant a un accès à un forum de discussion sur le site du cours. Pour certaines séances, une question sera posée et il sera invité à réagir sur le forum. la participation au forum du cours est importante mais non évaluée. L'étudiant doit le consulter régulièrement et il n'hésite pas à utiliser cet outil lorsqu'il a des questions susceptibles d'intéresser ces collègues.

8- Alignement pédagogique

Introduit pour favoriser la réussite des étudiants, il s'agit, pour l'enseignant «d'aligner» c'est-à-dire de mettre en cohérence : les objectifs d'apprentissage d'un enseignement/d'une formation, les méthodes et les activités pédagogiques déployées pour y parvenir et les méthodes d'évaluation des apprentissages des étudiants. Il est basé sur trois piliers : savoir, savoir-faire et savoir être.

- Le premier pilier le savoir et l'apprentissage des éléments théoriques du cours s'éffectuent en présentiel avec la publication d'un résumé sur le site, ce choix est basé sur plusieurs critères, dont le principale est de faciliter l'apprentissage : des différentes composantes des API et de microprocesseur et des différents développements et démonstrations, là où les étudiants trouve des difficultés d'assimiler en ligne ;
- Pour le deuxième pilier, le savoir-faire soit en présentiel ou en ligne: où les apprenants individuellement et/ou collectivement doivent proposer des solutions pour des situations problématiques proches des situations réelles qui exigent une bonne compréhension suivi d'une analyse pertinente et une décision efficace. À partir des renseignements fournis dans l'étude théorique, les apprenants vont réaliser un diagnostic, formuler des hypothèses de solutions puis choisir la solution la plus adéquate et l'argumenter. Elle permet de découvrir des propriétés, des formules ou des conjectures et de déduire des règles ou des principes applicables à des cas similaires dans l'industrie ;
- Le troisième pilier, le savoir-être : d'une façon collective consiste à faire réfléchir, travailler, échanger des petits groupes qui ne dépasse pas 6 personnes sur un thème ou une problématique donné pendant une quinzaine de jours. À la suite de cet échange, les groupes font une synthèse de leur discussion finalisée par une présentation.

9- Modalités de fonctionnement

- Toutes les activités proposées en amphithéâtre ou bien en ligne sont classées de la plus simple à la plus conséquente en termes d'organisation et de préparation sans devenir complexe. Leurs définitions ont été adaptées à partir de l'inventaire des situations d'apprentissage et des sources citées dans la bibliographie.

- Demander aux apprenants de citer plusieurs applications de ce que l'enseignant vient d'expliquer verbalement. Cette technique peut être utilisée à chaque fois que l'on doit recourir à des exemples. Elle permet de vérifier la capacité de transfert des apprenants.
- L'expérience a montré que les exercices pratiques en TD doivent être en quantité et en fréquence suffisantes pour que les capacités des étudiants aient le temps de se développer. Étant donné les contraintes de temps et de ressources auxquelles nous devons faire face, il est essentiel de se concentrer sur un objectif spécifique (ce que les étudiants doivent réellement apprendre) et de viser un niveau d'attente approprié, relativement au niveau actuel des étudiants.
- Organiser des activités en présentiel ou bien en ligne, où les apprenants échangent leurs points de vue concernant un thème, un problème technique ou un exercice, choisi par l'enseignant dans le but d'arriver à une décision ou une conclusion. Le but est d'amorcer un débat et d'apprendre aux apprenants à défendre leurs positions à l'aide d'arguments et de contre-arguments. Le débat se clôture par une phase de restructuration dirigée par l'enseignant.
- Utiliser des simulations qui consistent à une reproduction d'une situation constituant un modèle simplifié mais issue d'une réalité. Cette technique vise à recréer une situation représentant la réalité de manière objective et à laquelle l'apprenant pourrait être confronté. La simulation permet aux apprenants de mettre en pratique leurs habiletés et leurs apprentissages dans un environnement encadré par des règles.

10- Ressources d'aide

Pour tout complément d'information l'étudiant peut consulter :

- Manuscrit de cours publié en ligne ;
- Progressez avec les microcontrôleurs PIC, Gérard Samblancat, Ed. Dunod, 2006
- Programmation en C des PIC, Christian Tavernier, Ed. Dunod, 2006
- Microcontrôleurs AVR : Description et mise en oeuvre, Christian Tavernier, Ed. Dunod, 2009
- Advanced PIC microcontroller projects in C, Dogan Ibrahim, Ed. Elsevier, 2008
- Microcontrollers in C, T. V. Sickle, Ed. LLH Publishing, 2001
- SIMATIC, « Mise en route STEP7 ». Édition 03 /2006.
- Manuel SIMATIC S7, « langage cont pour SIMATIC S7-300/400 ». Programmation de blocs, C79000-G7077-C504-02
- Document Siemens, « Information et formation, automatisation et entrainements, programmation niveau A ». Edition Siemens AG, 2003.
- Document Siemens, « Automate programmable S7-300, caractéristiques électriques et techniques des CPU SIMATIC». Edition Siemens, 2001.
- A.B. Fontaine, « Le microprocesseur 16 bits 8086/8088 – matériel, logiciel, système d'exploitation ». Masson, Paris, 1988.
- P. Preux, « Assembleur i8086 ». Support de cours, IUT informatique du littoral, France, 95 – 96.

Nota: Ce document contient beaucoup d'informations, de figures et d'images tirés des différents ouvrages où les auteurs sont vivement remerciés, pour des prétextes d'oublier ou de difficulté de trouver la vraie référence peut être certains auteurs ne sont pas cités, pour cela je souhaite qu'ils vont accepter mes allégations d'avance. Dans tous les cas je m'engage, que ce manuscrit ne fera pas l'objet d'activités lucratives, il sera exclusivement réservé à des buts pédagogiques.

Chapitre I :

RAPPELS SUR LE GRAFCET

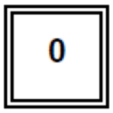
Introduction

Le GRAFCET (**GRA**phe **F**onctionnel de **C**ommande **E**tape **T**ransition) est un **modèle graphique** de description du comportement attendu des systèmes logiques séquentiels. C'est une traduction graphique du cahier des charges. Le Grafcet représente tous les états stables d'une partie ou du système dans lequel on atteint chacun des états à partir des autres en fonction des variations de certaines variables. Le Grafcet est normalisé sous l'indice de classement NF C 03-190. La norme européenne correspondante est EN 60848.

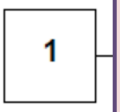
Dans ce chapitre, nous essayerons de donner les bases nécessaires de la modélisation des cahiers de charges et des systèmes automatisés en langage GRAFCET.

I.1- Représentation

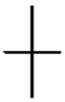
Un GRAFCET est constitué des éléments suivant :



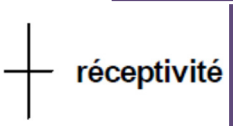
Etape initiale: L'étape initiale caractérise l'état du système au début du fonctionnement.



Etape : Une étape correspond à un comportement stable du système. Les étapes sont numérotées dans l'ordre croissant. A chaque étape on peut associer une ou plusieurs actions.

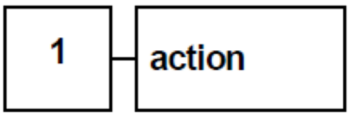


Transition: Les transitions indiquent les possibilités d'évolutions du cycle, à chaque transition est associée à une réceptivité.



Réceptivité : La réceptivité est la condition logique pour l'évolution du grafcet. Si la réceptivité est vrai (=1) le cycle peut évoluer. Les réceptivités proviennent du pupitre de commande, des fins de courses ou d'information provenant de la partie opérative.

Liaisons orientés: Le Grafcet se lit de haut en bas, autrement il est nécessaire d'indiquer son évolution avec des liaisons orientées constituées de flèche indiquant le sens.



Action : L'action est associée à une étape, elle est active lorsque le cycle est arrivé sur l'étape. Il est possible de définir les actions conditionnelles, temporisées (électrovanne, enclenchement d'un contacteur. . .), etc.

Etape active : Le point indique que l'étape est active.

I.2- Règles d'évolution

- **Règle 1 : Situation initiale**

L'étape initiale caractérise le comportement de la partie commande d'un système à l'instant initial (en début de cycle). Elle correspond souvent à la position de référence de la partie opérative (PO). L'étape initiale est activée sans condition en début de cycle (fonctionnement). Il peut y avoir plusieurs étapes initiales dans un même grafcet.

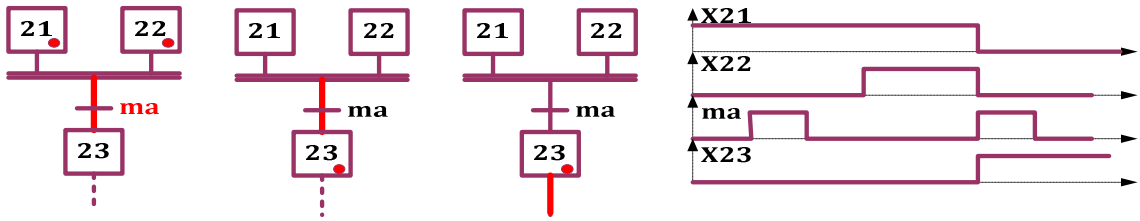
- **Règle 2 : Franchissement d'une transition**

Une transition est validée lorsque toutes les étapes, immédiatement précédentes reliées à cette transition, sont actives. Le franchissement d'une transition se produit :

- Lorsque la transition est validée ;
- Et que la réceptivité associée à cette transition est vraie.

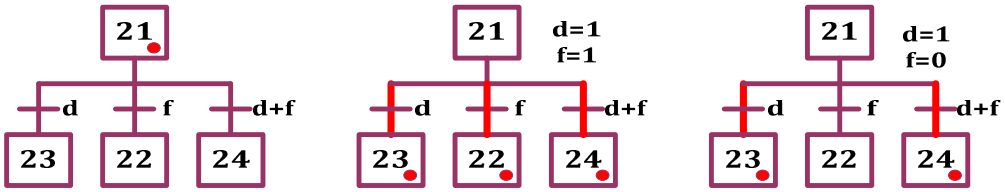
- **Règle 3 : Évolution des étapes actives**

Le franchissement d'une transition entraîne simultanément l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes celles immédiatement précédentes.



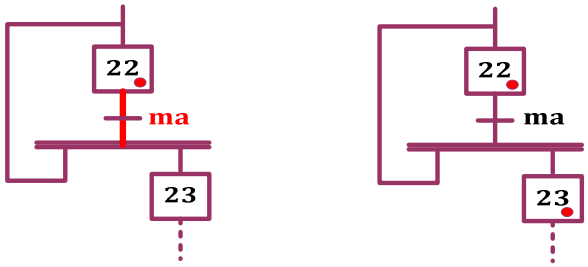
- **Règle 4 : Évolutions simultanées**

Plusieurs transitions simultanément franchissables sont simultanément franchies.



- **Règle 5 : Activation et désactivation simultanée d'une même étape**

Si au cours d'une évolution, une même étape se trouve être à la fois activée et désactivée, elle reste active.

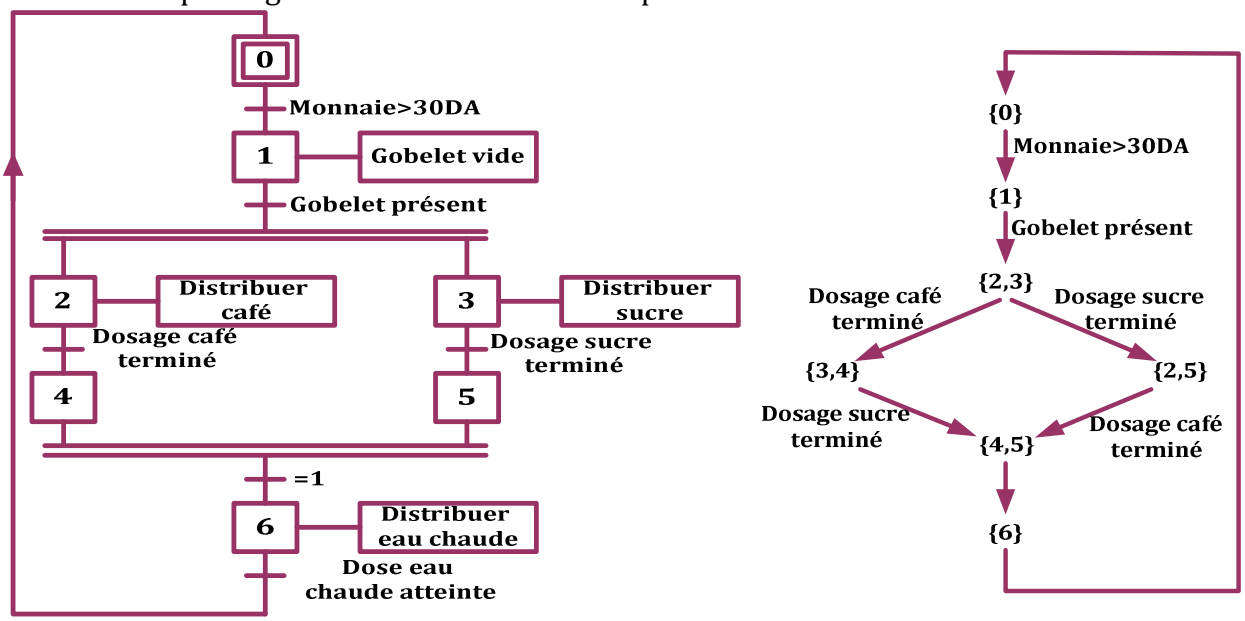


I.3- Règle de syntaxe

L'alternance étape-transition et transition-étape doit toujours être respectée quelle que soit la séquence parcourue. Pour les séquences simultanées, on a une transition unique et deux traits parallèles. Pour les séquences sélectionnées on a : une transition au début de chaque séquence pour la divergence en OU, et une transition à la fin de chaque séquence pour la convergence en OU.

I.4- Notion de situation

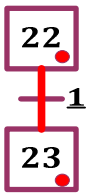
On appelle *situation* l'ensemble des étapes d'un grafcet qui sont actives à un moment donné. Elle est notée {numéros des étapes actives séparés par une virgule}. Elles apparaissent dans le *graphe des situations* qui représente l'enchaînement des différentes situations atteintes par le grafcet en fonction des réceptivités.



I.5- Réceptivités

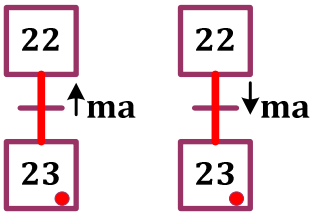
Une proposition logique, appelée réceptivité, qui peut être vraie ou fausse est associée à chaque transition.

a- Réceptivité toujours vraie



La notation 1 (1 souligné) indique que la réceptivité est toujours vraie. Dans ce cas, l'évolution est dite toujours, le franchissement de la transition n'est conditionné que par l'activité de l'étape amont.

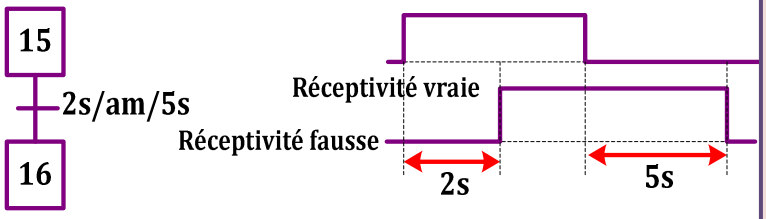
b- Front montant et descendant d'une variable logique



Front montant : La notion ↑indique que la réceptivité n'est vraie que lorsque la variable passe de valeur 0 à la valeur 1.

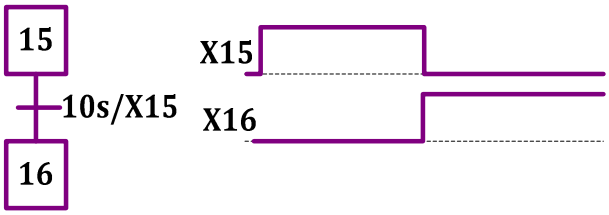
Front descendant : La notion ↓ indique que la réceptivité n'est vraie que lorsque la variable passe de valeur 1 à la valeur 0.

c- Réceptivité dépendante du temps



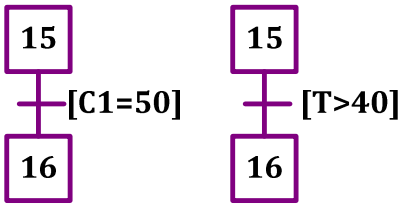
La notation est de la forme « **t1/variable/t2** ». Dans l'exemple ci-dessous, la réceptivité n'est vraie que 3 s après que « a » passe de l'état 0 à l'état 1, elle ne redevient fausse que 7 s après que « a » passe de l'état 1 à l'état 0.

d- Simplification usuelle



L'utilisation la plus courante est la temporisation de la variable d'étape avec un temps t_2 égal à zéro. Dans ce cas la durée d'activité de l'étape 15 est de 10s.

e- Valeur booléenne d'un prédicat

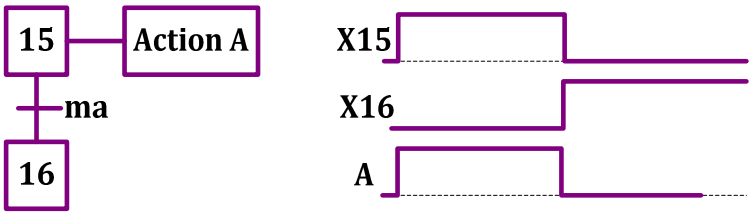


Un prédicat est une expression contenant une ou plusieurs variables et qui est susceptible de devenir une proposition vraie ou fausse.

I.6- Actions associées

Chaque étape peut être associée à une action ou plusieurs (qui s'effectuera quand l'étape sera active), c'est à dire un ordre envoyé vers la partie opérative ou vers d'autres grafcets. L'action est représentée dans un rectangle à droit. Il existe 2 types d'actions : les actions continues et les actions mémorisées.

I.6.a- Action continue

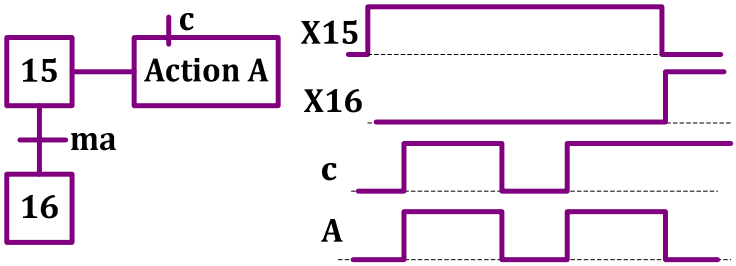


L'exécution de l'action se poursuit tant que l'étape à laquelle elle est associée est active.

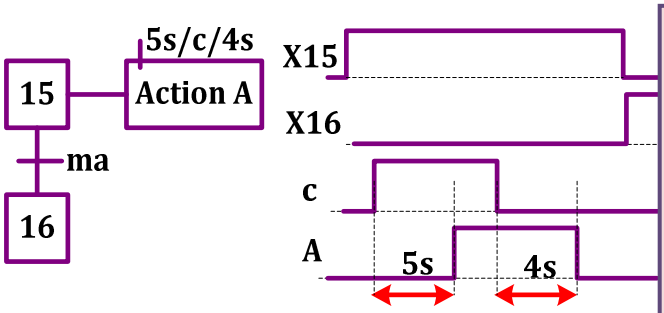
I.6.b- Action avec condition d'assignation (action conditionnelle)

Une action conditionnelle n'est exécutée que si l'étape associée est active et si la condition associée est vraie (La condition est une équation booléenne au même titre qu'une réceptivité.), l'action s'arrête si l'étape est désactivée ou si la condition n'est plus vérifiée. Elle peut être décomposée en 4 cas particuliers.

I.6.b.1- Action conditionnelle simple

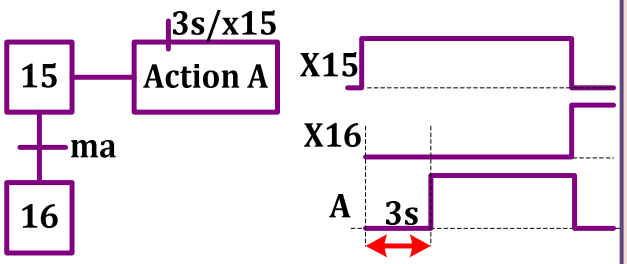


I.6.b.2- Action avec condition d'assignation dépendante du temps



Si l'étape 15 est activée, la condition d'assignation n'est vraie que 5s après que « c » passe de l'état 0 à l'état 1; Elle ne redevient fausse que 4s après que « c » passe de l'état 1 à l'état 0. Si la durée d'activité de l'étape 15 est inférieure à 5s, la sortie A ne sera pas assignée à la valeur vraie.

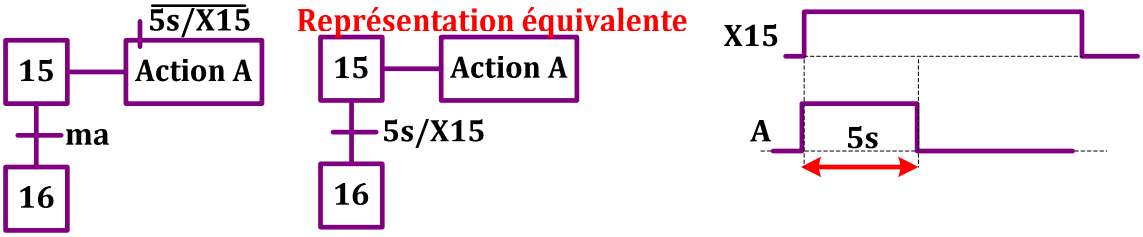
I.6.b.3- Action retardée



L'action retardée est une action continue dont la condition d'assignation n'est vraie qu'après une durée t_1 spécifiée depuis l'activation de l'étape associée. Dans l'exemple ci-dessous, l'action A sera exécutée 3s après l'activation de l'étape 15. Si la durée d'activité de l'étape 15 est inférieure à 3s, la sortie A ne sera pas assignée à la valeur vraie.

I.6.b.4- Action limitée dans le temps

L'action limitée dans le temps est une action continue dont la condition d'assignation n'est vraie que pendant une durée t_1 spécifiée depuis l'activation de l'étape à laquelle elle est associée.

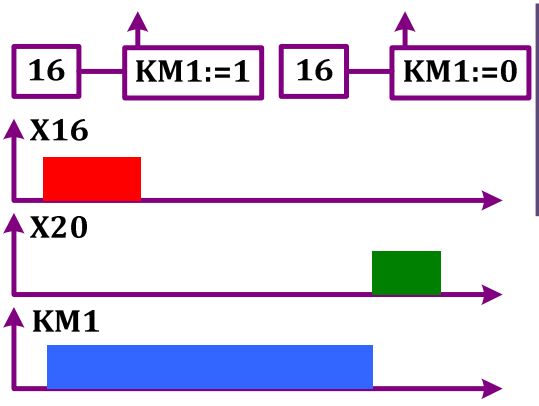


I.6.c - Action maintenue ou mémorisée

Le maintien d'un ordre (Action), sur la durée d'activation de plusieurs étapes consécutives, peut également être obtenu par la mémorisation de l'action. En mode mémorisé c'est l'association d'une action à des événements internes qui permet d'indiquer qu'une variable de sortie prend et garde la valeur imposée si l'un des événements se produit.

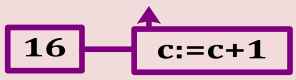
I.6.c.1 - Action à l'activation et à la désactivation

- **Action à l'activation :** Est une action mémorisée lors de l'activation de l'étape liée à cette action.



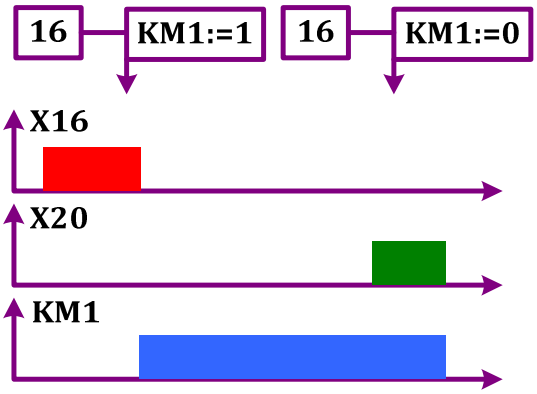
L'action restera (KM1=:1) quel que soit l'évolution du grafcet jusqu'à ce qu'une autre étape stoppe cette action par (KM1=:0). La flèche verticale montante indique que cette mémorisation sera effectuée dès l'activation de l'étape.

- Pour un compteur:



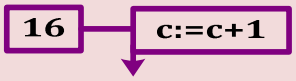
Incrémentation du compteur C à l'activation de l'étape X16

- **Action à la désactivation :** est une action mémorisée lors de la désactivation de l'étape liée à cette action.



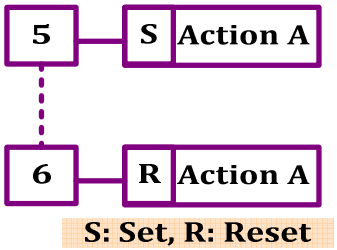
La flèche verticale descendante indique que cette mémorisation sera effectuée dès la désactivation de l'étape. KM1=1 dès la désactivation de l'étape 16 et reste à 1 jusqu'à la désactivation de l'étape 20 (KM1=0).

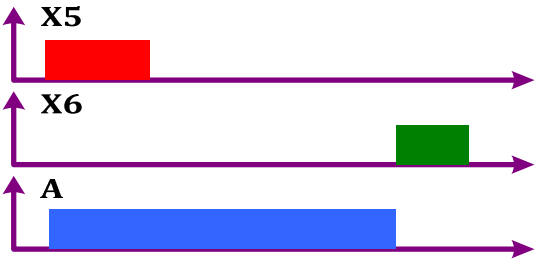
- Pour un compteur:



Incrémentation du compteur C à la désactivation de l'étape X16

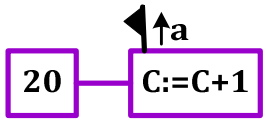
I.6.c.2- Utilisation de Set et Reset : Les actions à l'activation et à la désactivation peuvent être remplacées par les commandes Set et Reset.



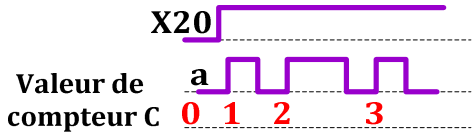


I.6.c.3- Action sur événement

Une action sur évènement est une action mémorisée conditionnée à l'apparition d'un événement, l'étape à laquelle l'action est reliée étant active. Il est impératif que l'expression logique associée à l'évènement comporte un ou plusieurs fronts de variables d'entrées.



Incrémentation du compteur C sur le front montant de «a». L'étape 20 étant active



I.7- Commentaires

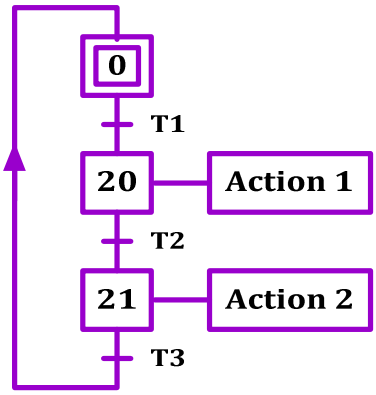
Pour améliorer la lisibilité des grafcets on peut y inclure des commentaires. Ils sont distingués des actions par des guillemets.

" étape d'activation "

I.8- Structures de base

L'automaticien peut sous réserve de l'application stricte de la règle de syntaxe imposant l'alternance étape transition réaliser des GRAFCET utilisant les différentes structures caractéristiques.

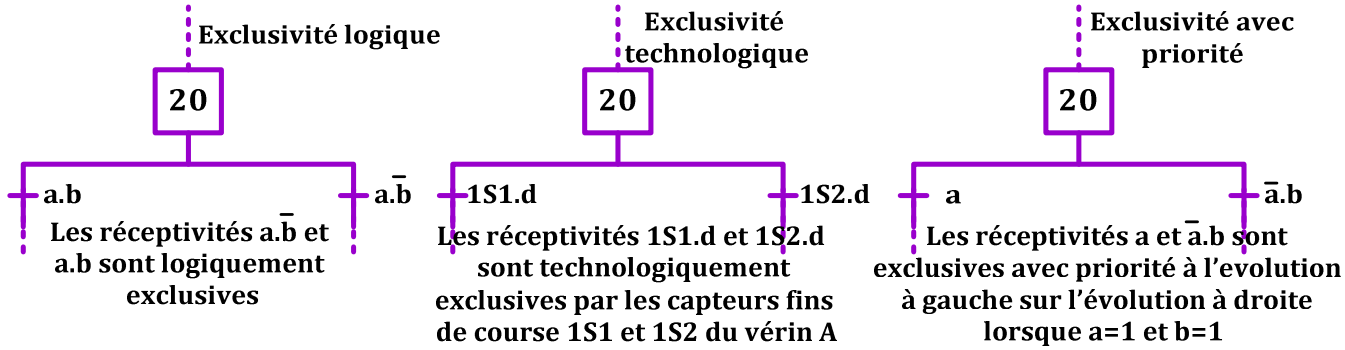
I.8.a- Séquence linéaire



Une séquence linéaire est composée d'une suite d'étapes qui peuvent être activées les unes après les autres. Exemple : pour le grafcet ci-dessous une seule séquence linéaire peut être exécutée : l'activation successivement des étapes 0, 20 et 21.

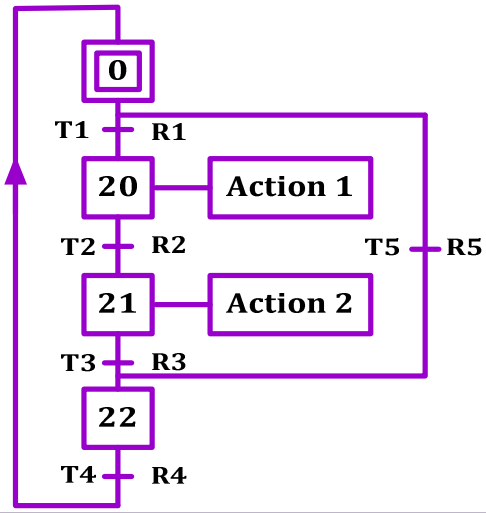
I.8.b- Sélection de séquence

Une sélection de séquence est un choix d'évolution entre plusieurs séquences à partir d'une ou plusieurs étapes. Elle se représente graphiquement par autant de transitions validées en même temps qu'il peut y avoir d'évolution possibles. L'exclusion entre les séquences n'est pas structurelle. Pour l'obtenir, il faut s'assurer soit de l'incompatibilité mécanique ou temporelle des réceptivités, soit de leur exclusion logique.

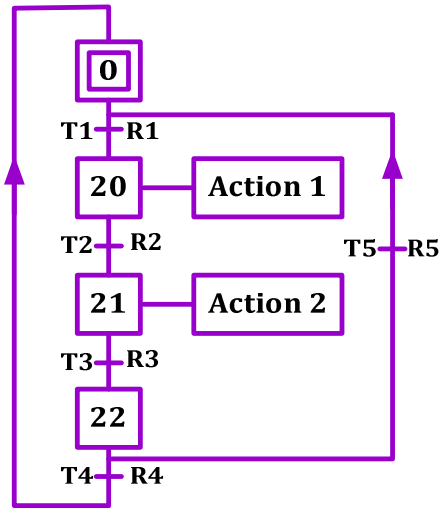


I.8.c- Saut d'étapes et reprise de séquence

- **Saut d'étape :** Il permet de sauter une ou plusieurs étapes lorsque les actions à réaliser deviennent inutiles ;
- **Reprise d'étape (saut en arrière) :** Il permet de reprendre une séquence lorsque les actions à réaliser sont répétitives.

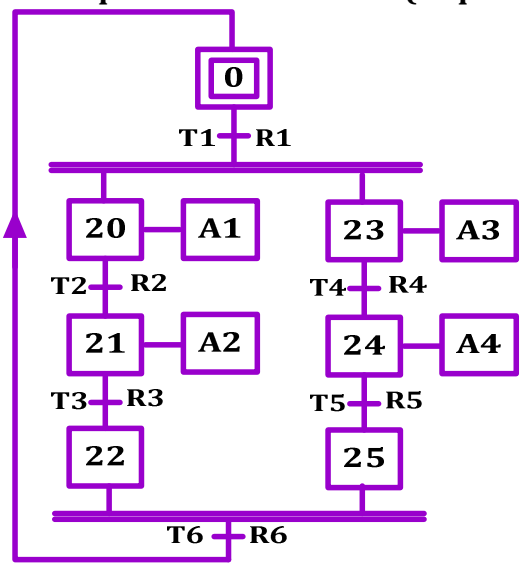


Dans l'exemple ci-contre, le saut des étapes 20 et 21est effectué lorsque l'étape 0 est active et la réceptivité R5 est valide. Dans ce cas, la transition T5 sera franchie ce qui aboutit à l'activation de l'étape 4.



La reprise de séquence permet de recommencer plusieurs fois la même séquence tant qu'une condition n'est pas obtenue. Pour éviter le problème d'indéterminisme, il faut utiliser des réceptivités

I.8.d- Séquences simultanées (séquences parallèles)

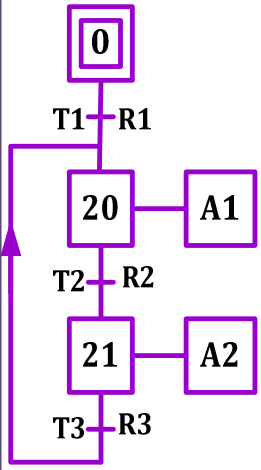


Souvent, dans une machine automatique à postes multiples, plusieurs séquences s'exécutent simultanément, mais les actions des étapes de chaque branche restent indépendantes. Pour représenter ces séquences simultanées, nous utilisons la structure en ET.
- Divergence en ET : lorsque la transition T1 est franchie, les étapes 20 et 23 sont actives.
- Convergence en ET : la transition T6 sera validée lorsque les étapes 22 et 25 seront actives. Si la réceptivité associée à cette transition est vraie, alors celle-ci est franchie.

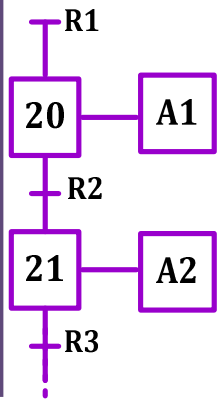
I.8.e- Structures particulières

- **Etape et transition source**

Étape source : Une étape source est une étape qui ne possède aucune transition amont. Pour que cette étape soit active, il faut qu'elle soit :
- Etape initiale ;
-Forcée depuis un grafcet hiérarchiquement supérieur.
Dans l'exemple ci-contre, l'étape source initiale **0** n'est active qu'à l'initialisation (et tant que la réceptivité **R1** n'est pas vraie).

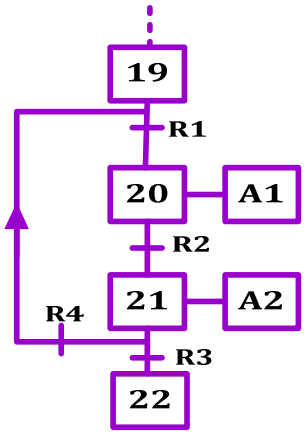


Transition source : Une transition source est une transition qui ne possède aucune étape amont. Par convention, la transition source est toujours validée et est franchie dès que sa réceptivité est vraie. Dans l'exemple ci-contre, l'étape **20** est activée dès que la réceptivité **R1** est vraie.

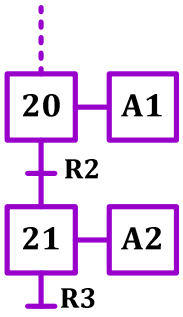


- **Etape et transition puits**

Etape puits : est une étape qui ne possède aucune transition aval ; sa désactivation est possible par un ordre de forçage d'un GRAFCET de niveau supérieur.



Transition puits : est une transition qui ne possède aucune étape aval. Dans l'exemple ci-dessous, lorsque la transition puits est validée et que « **c.f** » est vraie, le franchissement de cette transition a pour unique conséquence de désactiver l'étape 11.



I.9- Structuration par forçage d'un GRAFCET partiel

L'ordre de forçage de situation émis par un GRAFCET hiérarchiquement supérieur permet de modifier la situation courante d'un GRAFCET hiérarchiquement inférieur, sans qu'il y ait franchissement de transition. L'ordre de forçage est un ordre interne prioritaire sur toutes les conditions d'évolution et a pour effet d'activer la ou les étapes correspondant à la situation forcée et de désactiver les autres étapes du GRAFCET forcé. L'ordre de forçage est représenté dans un double rectangle associé à l'étape pour le différencier d'une action.

21 — GPN{50}	Lorsque l'étape 21 est active, le GRAFCET nommé GPN est forcé dans la situation caractérisée par l'activité de l'étape 50 (l'étape 50 est activée et les autres étapes sont désactivées).
21 — GC{50,55}	Lorsque l'étape 21 est active, le GRAFCET nommé GC est forcé dans la situation caractérisée par l'activité des étapes 50 et 55 (les étapes 50 et 55sont activées et les autres étapes sont désactivées).
45 — GPN{*}	Lorsque l'étape 45 est active, GRAFCET nommé GPN est forcé dans la situation où il se trouve à l'instant du forçage. On appelle également cet ordre « **figeage** ».
39 — GPN{ }	Lorsque l'étape 39 est active, le GRAFCET nommé GPN est forcé dans la situation vide. Dans ce cas aucune de ses étapes n'est active.
17 — G5{INIT}	Lorsque l'étape 17 est active, le GRAFCET nommé G5 est forcé dans la situation dans laquelle seules les étapes initiales sont actives.

I.10- Macro-Représentation et Représentation détaillée

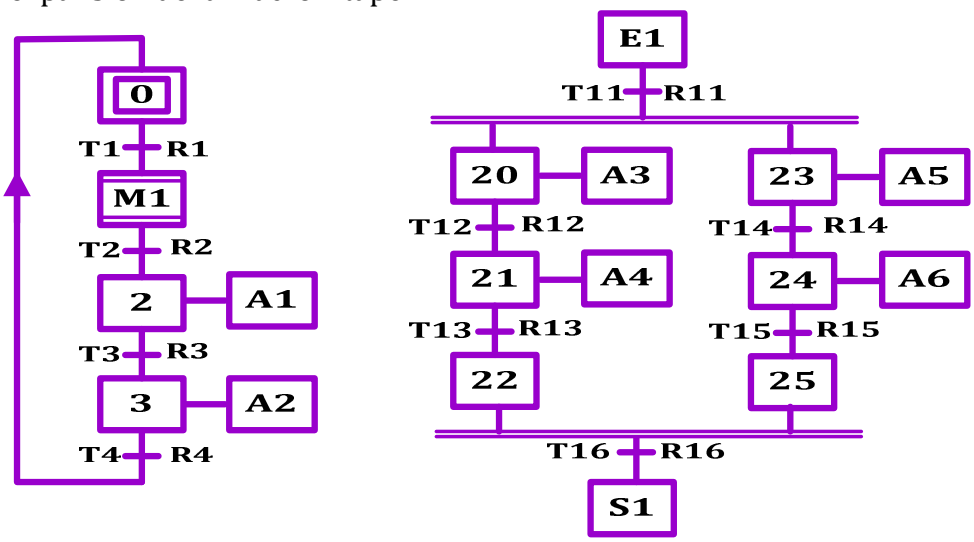
Devant la complexité des systèmes automatisés actuels, il devient indispensable d'utiliser une méthodologie rigoureuse pour définir la commande. Celle-ci est basée sur l'utilisation d'une approche progressive structurée en partant d'un haut niveau de description (macro-représentation) jusqu'au niveau de détail complet ou tous les ordres et informations élémentaires sont pris en compte. Ceci permet d'obtenir une représentation claire et précise dont les avantages sont :

- Une représentation homogène facilement analysable ;
- Sur format réduit : A4, A3 ;
- Approche pédagogique ;
- Facilité de mise à jour, etc.

La Macro-Etape : est la représentation unique d'un ensemble fonctionnel d'étapes et de transitions appelé : expansion de Macro-Etape. Les règles associées à la macro-étape sont :

- L'expansion de Macro- Etape comprend une étape d'entrée et une étape de sortie ;
- L'étape de sortie participe à la validation des transitions avales
- Aucune liaison entre la Macro-Etape et son environnement en dehors de ses points d'accès qui sont l'étape d'entrée et l'étape de sortie.

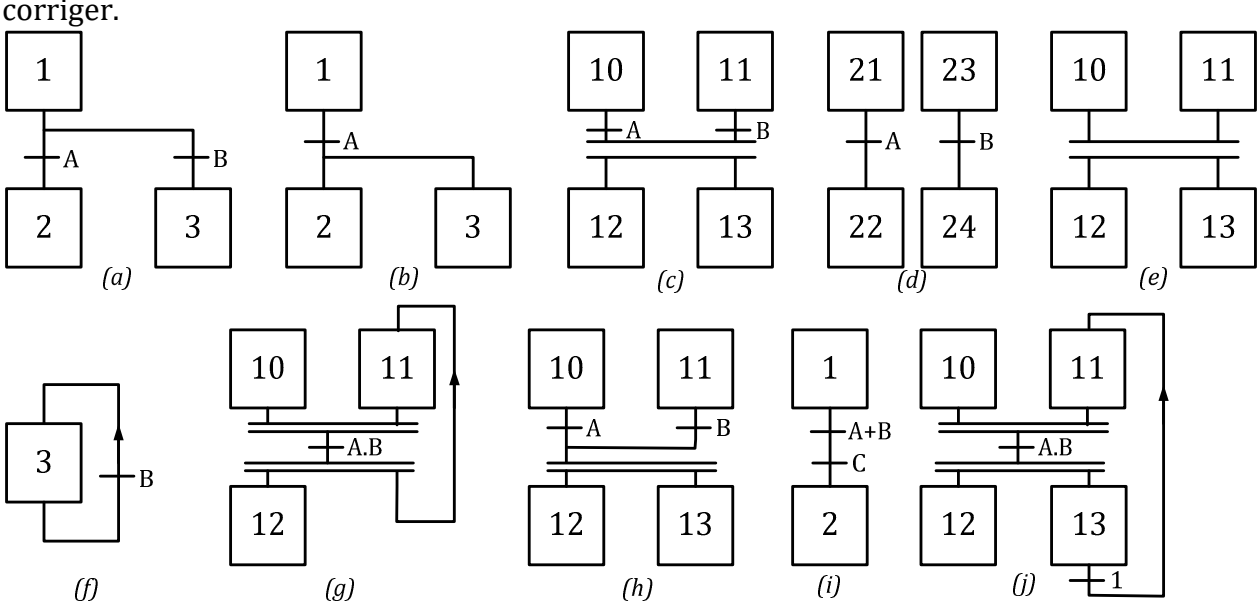
Remarque : L'expansion d'une macro-étape peut comporter d'autres macro-étapes et des étapes initiales. Cependant, il faut éviter que ces étapes initiales ne soient l'étape de sortie ou d'entrée de l'expansion de la Macro-Etape.



Annexe I

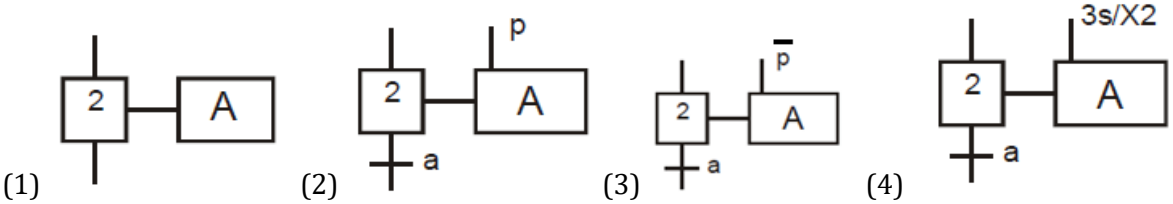
Exercice 1

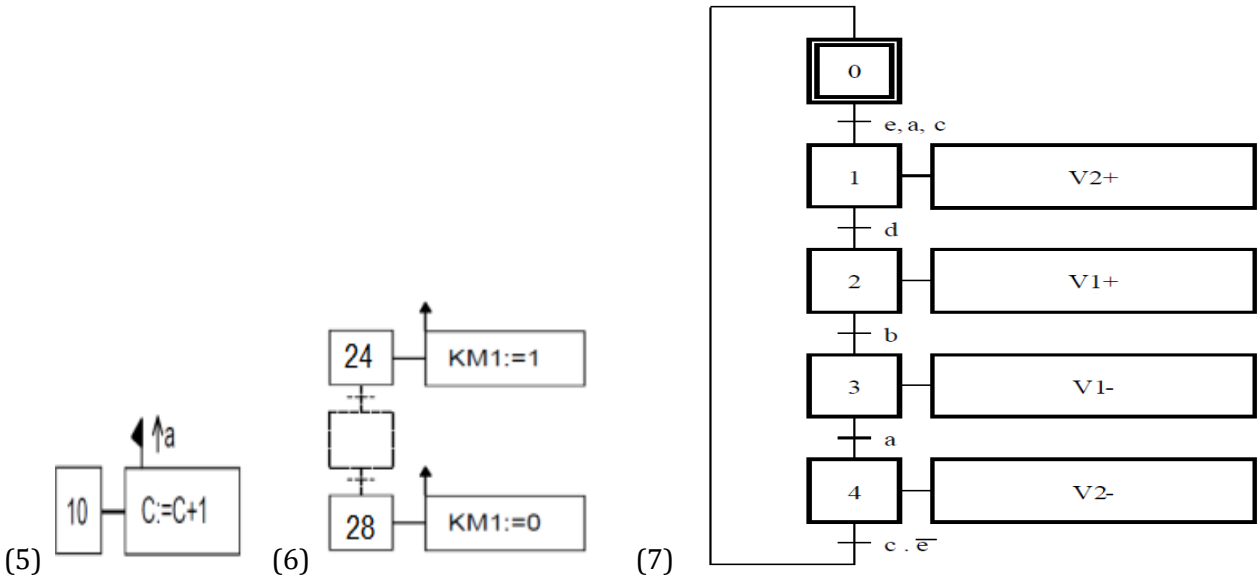
Les Grafcet représentés à la figure ci-dessous comportent des erreurs de construction (non conforme aux règles de construction). Trouver les erreurs et suggérer une façon de les corriger.



Exercice 2

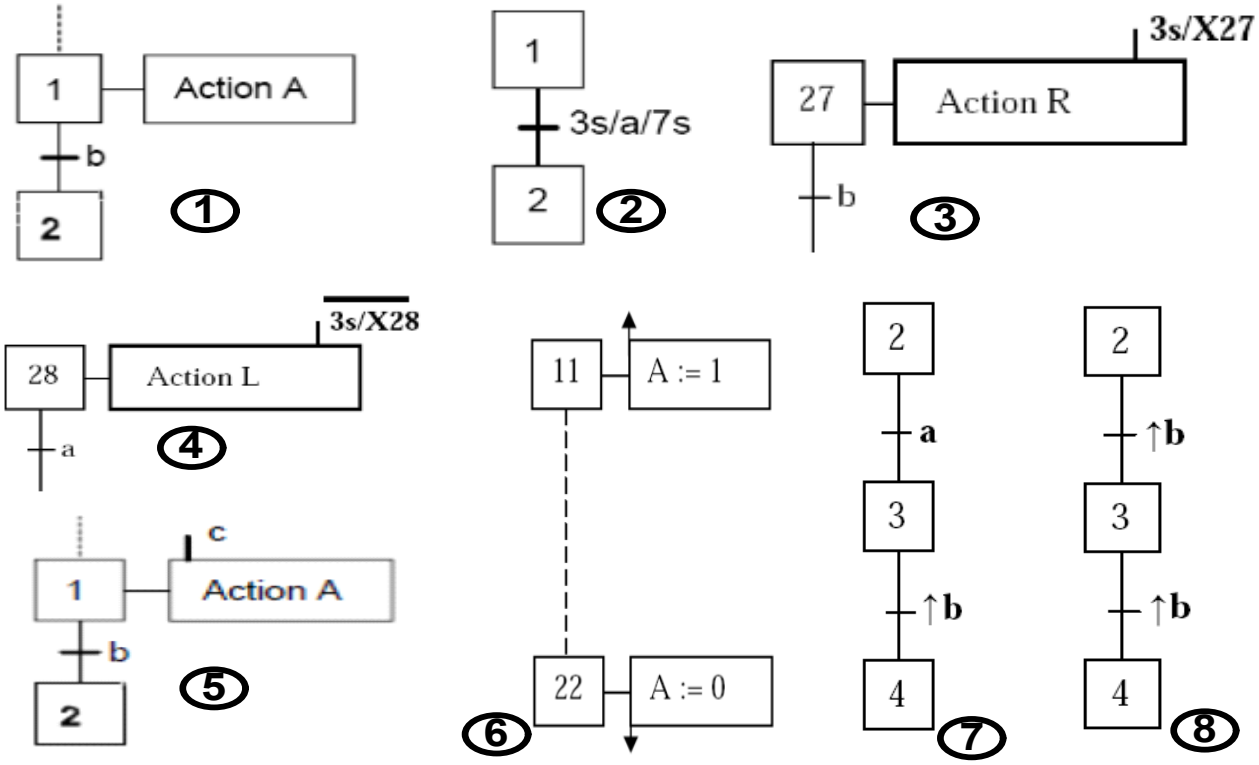
Donner les chronogrammes relatifs aux variables internes et externes liées au temps, à partir de l'analyse des GRAFCET ou des sous- GRAFCET suivants:





Exercice 3

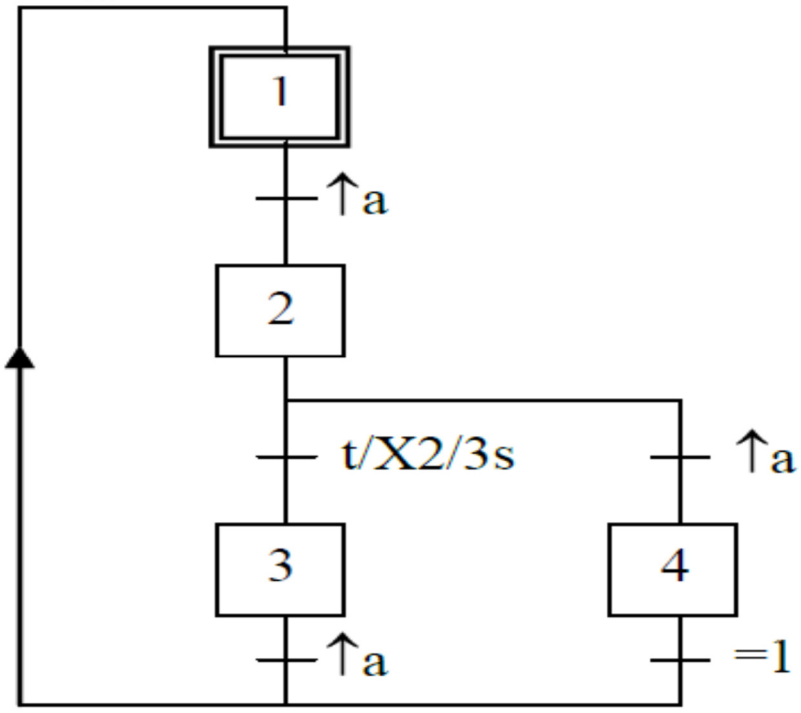
Donner les chronogrammes relatifs aux variables internes et externes liées au temps, à partir de l'analyse des GRAFCET ou des sous- GRAFCET suivants:

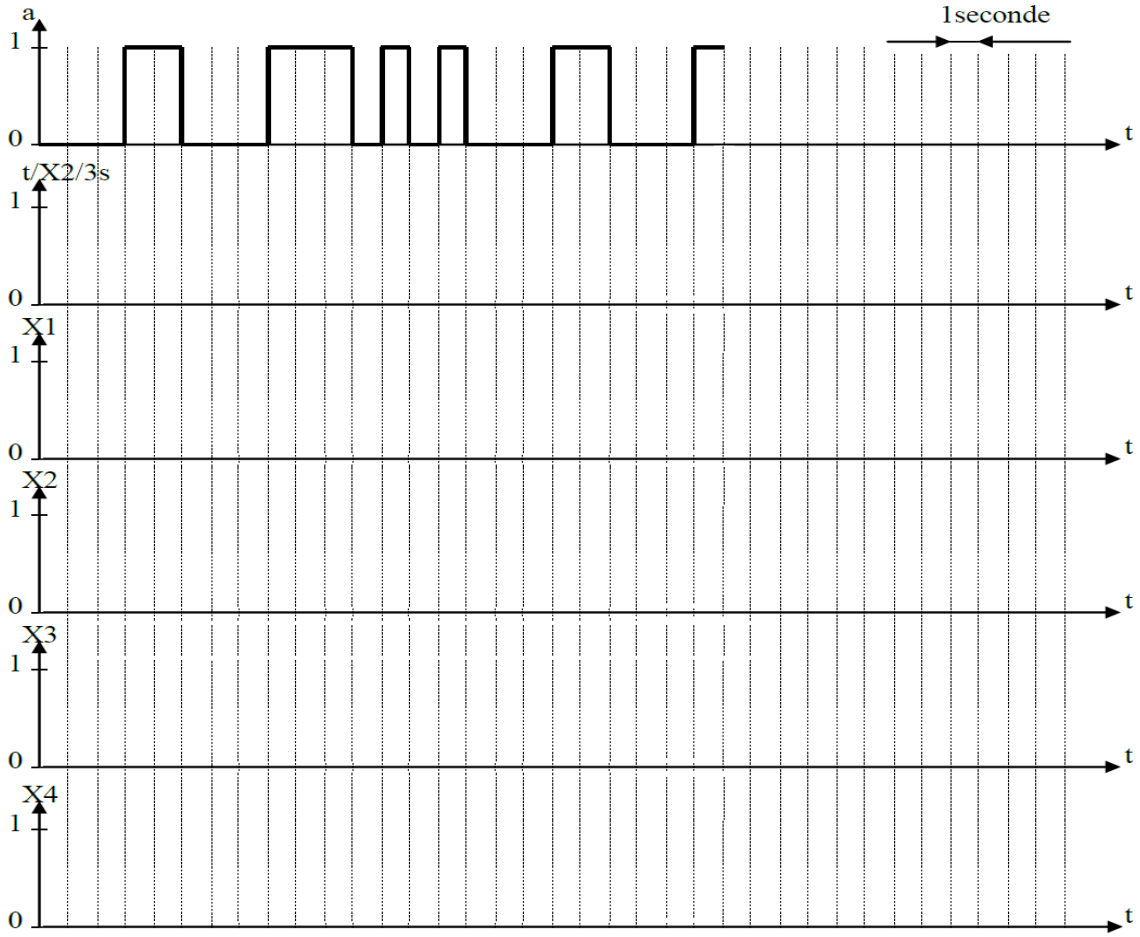


Exercice 4

1- Compléter les chronogrammes (page jointe) relatifs aux variables internes X1, X2, X3 et X4 et à la variable externe liée au temps, à partir de l'analyse du GRAFCET et du chronogramme relatif à l'entrée « a » si, on considère que les durées de franchissement des transitions et d'activation des étapes sont nulles ;

2- Comment en appelant l'évolution associée à l'étape 4? Citer l'une des applications efficaces de ce type d'évolution.

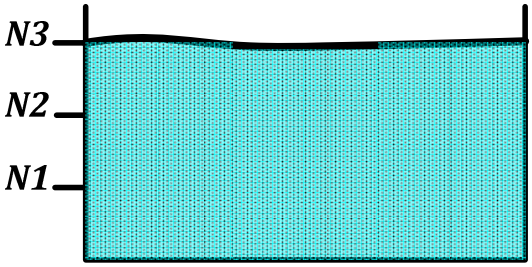




Exercice 5

Le niveau de liquide contenu dans un réservoir est contrôlé par 3 détecteurs N1, N2 et N3. L'alimentation de ce réservoir s'effectue par 3 pompes P1, P2 et P3 (voir figure ci-dessous). Chaque niveau découvert (N) entraine la mise en route d'une pompe. Le nombre de pompes en service sera donc fonction du nombre de niveaux découverts. De plus, afin d'équilibrer l'usage des pompes, celles-ci seront permutées a tour de rôle.

On demande le Grafcet d'une telle commande.



Exercice 6

On considère ci-dessous un système de commande d'une barrière automatique de parking payant. La barrière est composée de deux parties. La partie de gauche peut s'ouvrir seul et laisser passer un véhicule à 2 roues. Les deux parties peuvent s'ouvrir ensemble et laisser passer un véhicule à 4 roues. Sur la gauche une borne de péage avec deux orifices pour les pièces de 50Da et 100Da, et au sol deux plaques A et B pour détecter la présence de véhicules.

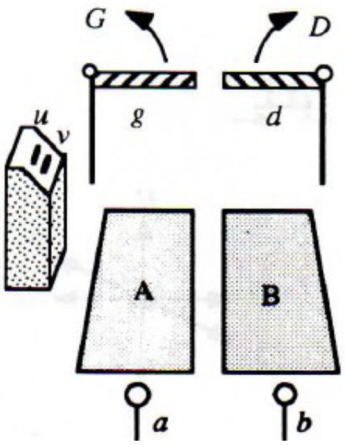
Les entrées et sorties du système à décrire sont présentées sur la figure ci-dessous. Par exemple, on a ***u=1*** pendant un court instant quand on met une pièce de 50Da.

Pour obtenir l'ouverture de la partie gauche, il faut un véhicule sur la plaque A seule, et mettre une pièce de 50Da (au moins). La barrière se referme quand il n'y a plus de véhicule sur la plaque A Pour obtenir l'ouverture des deux parties, il faut un véhicule portant sur les

plaques A et B, et mettre soit une pièce de 100Da, soit deux pièces de 50Da (au moins). La barrière se referme quand il n'y a plus de véhicule sur les plaques A et B.

On admet qu'un véhicule à 4 roues qui appuie d'abord sur la plaque A doit appuyer sur la plaque B dans un délai qui n'excède pas une seconde.

- Décrire le fonctionnement de ce système de commande par un grafcet.



$a = 1$: véhicule sur plaque A
$b = 1$: véhicule sur plaque B
$u = 1$: passage d'une pièce de 50Da
$v = 1$: passage d'une pièce de 100Da
$g = 1$: barrière gauche fermée
$d = 1$: barrière droite fermée

$G = 1$: ouverture (et maintien ouvert) barrière gauche
$D = 1$: ouverture (et maintien ouvert) barrière droite

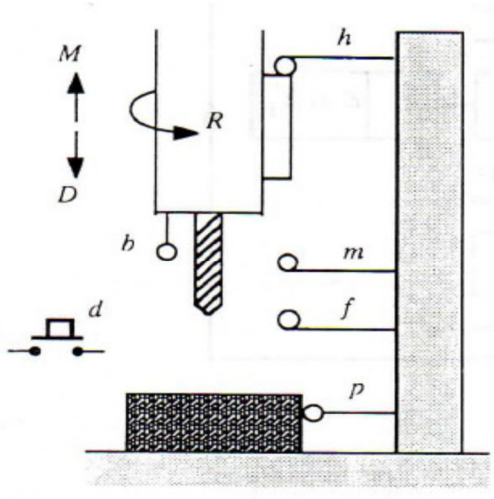
Exercice 7

Sur la figure ci-dessous, une perceuse qui effectue un cycle de perçage, ce cycle est commandé par les variables booléennes ***d, h, m, f, b*** et ***p*** qui sont les entrées de l'automatisme à décrire par grafcet (elles sont à 1 quand il y a contact). Les sorties de cet automatisme sont les variables booléennes ***M, D*** et ***R*** (qui valent ***1*** quand les moteurs qui correspondants sont en marche).

Le cycle commence lorsqu'on appuie sur le bouton poussoir ***d***, s'il y a une pièce présente. Les pièces à percer peuvent être de deux types: pièce basse ou pièce haute. Lorsque la pièce est basse le cycle est le suivant : dès le début du cycle, on a mise en route du moteur de descente et du moteur de rotation de la broche portant le foret. Quand le contact ***f*** est atteint, la broche remonte jusqu'au contact ***h*** et la rotation s'arrête à ce moment-là. Lorsque la pièce est haute (ce qui est repéré par le fait que le contact ***b*** se produit avant le contact à mi-course ***m***), la broche remonte jusqu'au contact ***h*** quand le contact ***m*** est atteint, puis redescend jusqu'au contact ***f*** avant de remonter dans les mêmes conditions que dans le cycle court correspondant à une pièce basse.

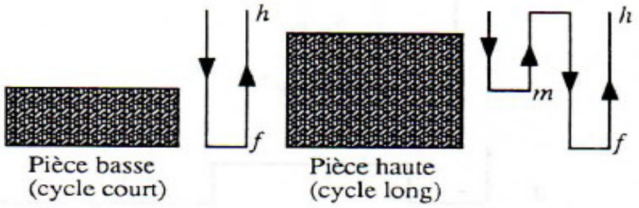
Avant de recommencer un nouveau cycle, il faut que la pièce déjà percée ait été retirée et remplacée.

- Décrire le comportement de l'automatisme par un grafcet.



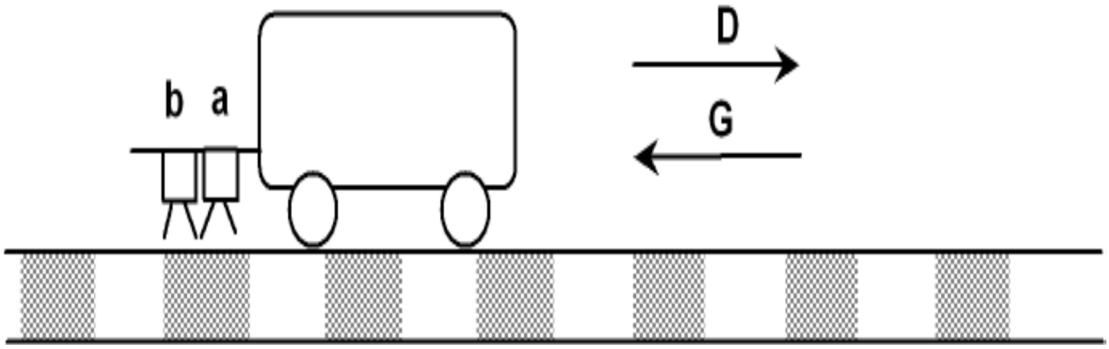
d : bouton-poussoir de départ
h : contact haut de fin de course
m : contact à mi-course
f : contact bas de fin de course
b : contact hauteur de pièce
p : contact présence de pièce

M : moteur de montée
D : moteur de descente
R : moteur de rotation



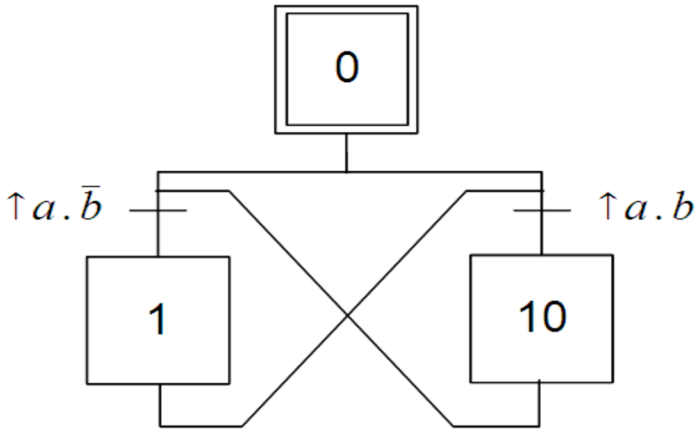
Exercice 8

On désire suivre la position d'un chariot avec un capteur à réflexion double *ab* solidaire du chariot et des marques réfléchissantes au sol . On suppose que la distance entre les capteurs *a* et *b* est inférieure à la largeur des bandes réfléchissantes et aux intervalles les séparant.



a- Donner les chronogrammes des capteurs *a* et *b* en fonction du sens de marche *D* ou *G*.

b- On se propose de déterminer le sens de circulation par le Grafcet suivant:



- Analyser le principe de ce Grafcet
- Ce Grafcet est-il syntaxiquement correct ? Le corriger si nécessaire .
- Les signaux *D1* et *G1* sont-ils utilisables pour réaliser un comptage ?

c- Pour réaliser un comptage des bandes réfléchissantes, on se propose de compléter le Grafcet précédent en détectant les fronts descendants du capteur *a* pour obtenir des étapes *D2* et *G2* .

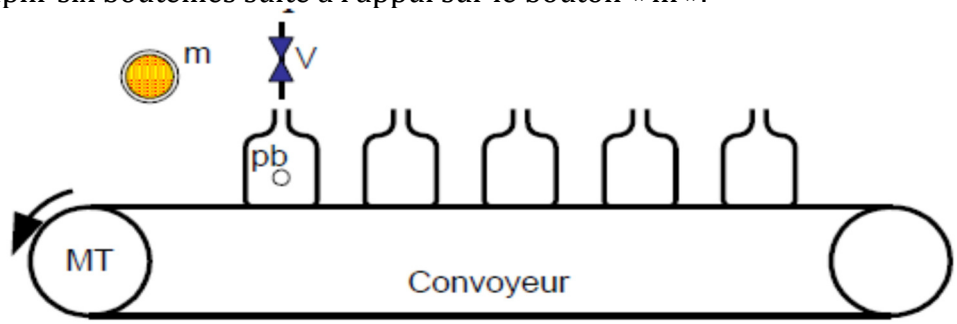
- Donner le Grafcet complet de détection (ne pas oublier que le chariot peut changer de sens à tout instant).
- Représenter les signaux *D1* et *G1* de ce nouveau Grafcet sur un chronogramme .

d- Proposer une solution sans aucune détection de front, sur la base du suivi du couple des signaux *a* et *b* selon le chronogramme. Dans un premier temps, ne considérer qu'un seul sens de circulation (4 étapes en séquence), puis le sens inverse (4 étapes) et enfin le changement de sens à tout instant (8 transitions à établir entre les 2 séquences définies précédemment).

Exercice 8

Ce petit automatisme de remplissage de bouteilles est constitué d'un convoyeur commandé par un moteur « MT », d'une valve de remplissage « V » et de deux capteurs, un capteur de présence de bouteille prête à être remplie « pb » eu un bouton-poussoir « m ». Pour démarrer l'automatisme, il suffit d'appuyer sur le bouton poussoir « m ». Le tapis roulant démarre et une première bouteille se présente au poste de remplissage. Le capteur « pb » détecte la bouteille, puis la valve « V » s'ouvre pendant 12 secondes pour remplir la bouteille.

La bouteille est évacuée et une nouvelle bouteille est présentée pour remplissage. Il faut remplir six bouteilles suite à l'appui sur le bouton « m ».



Chapitre II :

AUTOMATE PROGRAMMABLE INDUSTRIEL –API-

Introduction

L'automate programmable industriel API (ou Programmable Logic Controller PLC) est l'un des appareils de commande des systèmes de production et d'automatisme les plus utilisés dans l'industrie. Son apparition pour la première fois fut aux Etats-Unis dans le secteur de l'industrie automobile.

II.1- Systèmes automatisés

L'automatisation d'un système consiste à transformer l'ensemble des tâches de commande et de surveillance, réalisées par des opérateurs humains, dans un ensemble d'objets techniques appelés partie commande. Cette dernière mémorise le savoir-faire des opérateurs, pour obtenir l'ensemble des actions à effectuer sur la matière d'œuvre, afin d'élaborer le produit final. Parmi les objectifs de l'automatisation on peut citer : réalisation des tâches répétitives, sécurité, économie des matières premières et l'énergie, augmentation de la productivité et plus d'adaptation à des contextes particuliers (flexibilité).

II.2- Structure des systèmes automatisés

Un système automatisé est toujours composé d'une partie commande (PC), une partie opérative (PO) et d'une partie de supervision. Pour faire fonctionner ce système, l'opérateur va donner des consignes à la PC. Celle-ci va traduire ces consignes en ordres qui vont être exécutées par la PO. Une fois les ordres accomplis, la PO va le signaler à la PC, par un retour d'information, qui va à son tour le signaler à l'opérateur, ce dernier pourra donc dire que le travail a bien été réalisé.

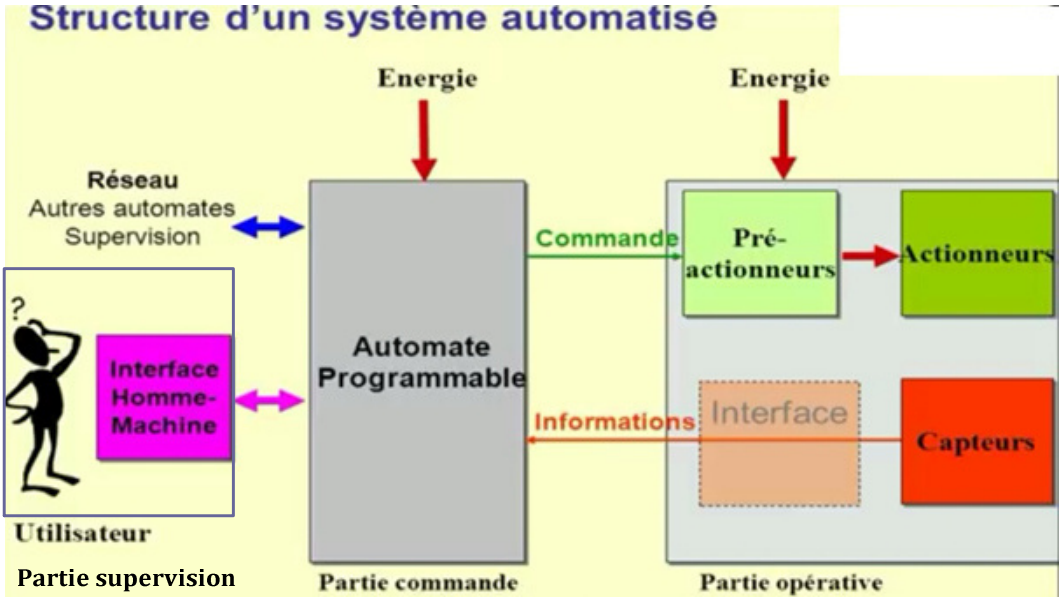


Figure II.1 Structure des systèmes automatisés

a- Partie commande

C'est la partie qui gère le fonctionnement du système automatisé. Elle est, en général, composée d'un ordinateur et/ou d'un automate (API) qui contient dans sa mémoire un programme. Elle transmet les ordres aux actionneurs de la partie opérative à partir :

- Du programme qu'elle contient ;
- Des informations reçues par les capteurs ;
- Des consignes données par l'utilisateur ou par l'opérateur.

b- Partie opérative

Elle consomme de l'énergie électrique, pneumatique ou hydraulique. Elle comporte, en général, un boîtier (appelé bâti) contenant :

- ✓ Des pré-actionneurs et des actionneurs (transforment l'énergie reçue en énergie utile : moteur, vérin, pompe) ;

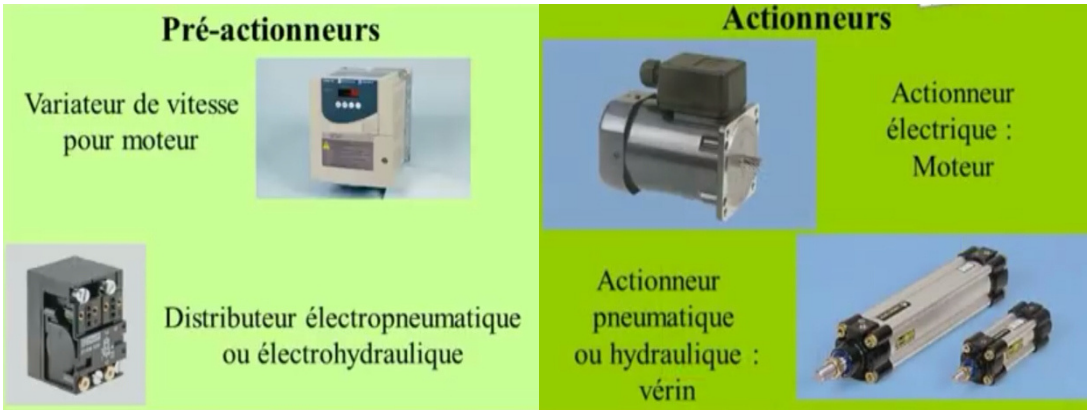


Figure II.2 Exemple des pré-actionneurs et des actionneurs

- ✓ Des capteurs (transforment les variations des grandeurs physiques liées au fonctionnement de l'automatisme en signaux électriques : capteur de position, de température, bouton poussoir).

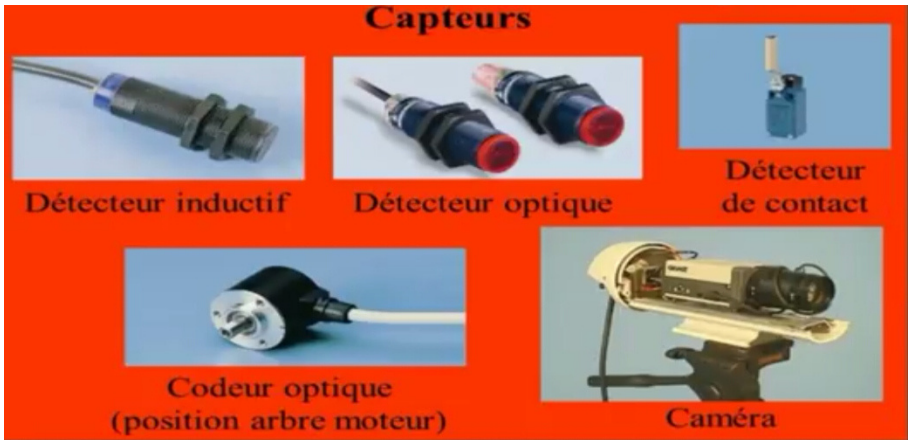


Figure II.3 Exemple des capteurs

c- Interface :

Elle relie la partie opérative (PO) et la partie commande (PC) et/ou, elle relie la partie commande et la partie supervision (PS). C'est un système de traduction d'informations entre la PC et la PO et/ou entre PC et PS par IHM (interface homme machine).

Figure II.4 Exemple des interfaces

II.3- Principe des logiques programmée et câblée

II.3.1- Logique câblée

En logique câblée, la loi de contrôle évoquée ci-dessus est réalisée en interconnectant judicieusement des opérateurs matériels réalisant des fonctions logiques de base. Suivant la technologie adoptée, il peut s'agir d'opérateurs fluidiques (interconnectés par tuyauteries), de relais électromagnétiques ou de relais statiques (interconnectés par fil).

Comme on sait, le nombre de types d'opérateurs nécessaires pour réaliser l'ensemble des fonctions logiques possibles peut être très réduit. Par exemple, les familles d'opérateurs suivantes : [AND, OR, NOT], [porte NAND], [porte NOR], [relais normalement ouvert, relais normalement fermé] permettent, chacune, de réaliser n'importe quelle fonction logique.
Bien entendu, la disposition d'opérateurs supplémentaires (bistables, compteurs, etc.), si elle n'est pas théoriquement nécessaire, est de nature à simplifier considérablement la réalisation d'une fonction logique donnée.

Les problèmes posés par la logique câblée sont :

- Le volume de matériel est directement proportionnel à la complexité de la fonction réalisée ;
- La fonction en question est physiquement inscrite dans le câblage et donc particulièrement difficile à modifier, que ce soit en phase de mise au point ou lors d'extensions ultérieures du processus.

Par contre, la logique câblée présente un certain nombre d'avantages par rapport à la logique programmée qui sont :

- La vitesse de traitement ne dépend pas de la complexité du problème puisque tous les éléments logiques travaillent en parallèle. Pour les relais statiques, cette vitesse peut d'ailleurs être très élevée.

II.3.2- Logique programmée

L'idée de la logique programmée est de n'utiliser qu'un seul jeu d'opérateurs de base (qui portera le nom d'unité logique). Pour réaliser une fonction logique donnée, on emploiera ce jeu unique pour calculer successivement les différents termes de la fonction et les combiner de manière à arriver ainsi, de proche en proche, au résultat cherché. On travaille donc en quelque sorte ici par "balayage". Il est clair que si ce balayage est répété à une cadence suffisamment rapide par rapport à la dynamique des signaux, on aura l'impression d'un fonctionnement parallèle semblable à celui de la logique câblée.
En pratique, on essaye généralement d'avoir des cadences de répétition du même ordre de grandeur que les temps de basculement des relais (de quelques ms à quelques dizaines de ms). La manière dont le balayage en question doit être effectué est décrite par une suite d'instructions stockées dans une mémoire et constituant ce que l'on appelle un programme.

II.4 Généralités sur les API

II.4.1- Définition d'un automate programmable industriel (API)

Un automate programmable est un système électronique fonctionnant de manière numérique, destiné à être utilisé dans un environnement industriel. Il utilise une mémoire programme pour le stockage interne des instructions utilisées aux fins de la mise en œuvre des fonctions spécifiques, telles que : des fonctions logiques, de mise en séquence, de temporisation, de comptage et de calcul arithmétique, pour commander, au moyen des entrées/sorties (de type tout/rien ou analogiques), de divers types de machines ou de processus. L'automate programmable et ses périphériques associés sont conçus pour pouvoir facilement s'intégrer à un système d'automatisme industriel et être facilement utilisé dans toutes leurs fonctions prévues. Un API a trois caractéristiques fondamentales :

- ✓ Il peut être directement connecté aux capteurs et pré-actionneurs grâce à ses E/S industrielles ;
- ✓ Il est conçu pour fonctionner dans des ambiances industrielles sévères ;
- ✓ Enfin, sa programmation à partir de langages spécialement développés pour le traitement de fonctions d'automatisme facilite son exploitation et sa mise en œuvre.

II.4.2- Architecteur des automates

L'architecture du processeur d'un automate programmable est fondamentalement la même que celle d'un ordinateur à usage général. Néanmoins, il existe certaines caractéristiques importantes qui les distinguent. Tout d'abord, contrairement aux ordinateurs, les automates programmables sont conçus pour résister aux conditions difficiles

de l'environnement industriel. Un automate bien conçu peut-être placé dans une zone avec d'importantes quantités : de bruit électrique, des interférences électromagnétiques, des vibrations et d'humidité sans condensation. Une deuxième distinction des automates est que leurs matériels et logiciels sont conçus pour une utilisation facile par les électriciens et les techniciens. Les interfaces matérielles pour la connexion d'appareils de terrain sont en réalité partie de l'automate lui-même et se connectent facilement. Les circuits d'interface modulaires et autodiagnostic sont en mesure d'identifier les dysfonctionnements et, d'ailleurs, ils sont facilement enlevés et remplacés. En outre, la programmation du logiciel utilise des symboles traditionnels relais d'échelle, ou d'autres langues apprises facilement, qui sont familières au personnel de l'usine. Alors que les ordinateurs sont des machines informatiques complexes capables d'exécuter plusieurs programmes ou tâches simultanément et dans n'importe quel ordre. La norme PLC exécute un programme unique dans un mode séquentiel ordonné de la première à la dernière instruction.

II.4.3- Aspect extérieur des API

Les automates peuvent être de type compact ou modulaire.

a- **De type compact :** on distinguera les modules de programmation (LOGO de Siemens, ZELIO de Schneider, MILLENIUM de Crouz,...) des micros automates. Il intègre le processeur, l'alimentation, les entrées et les sorties. Selon les modèles et les fabricants, il pourra réaliser certaines fonctions supplémentaires (comptage rapide, ajout d'entrées/ sorties analogiques ...) et recevoir des extensions en nombre limité. Ces automates, de fonctionnement simple, sont généralement destinés à la commande de petits automatismes.

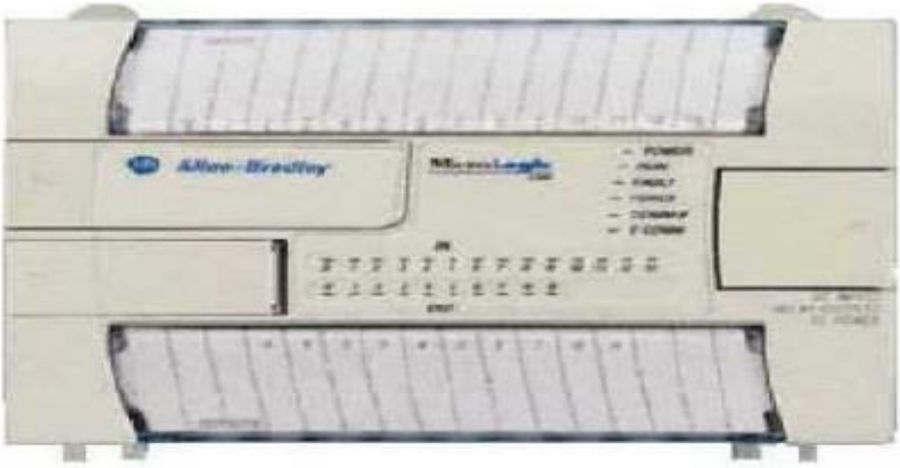


Figure II.5 Exemple d'un automate de type compact

b- **De type modulaire :** le processeur, l'alimentation et les interfaces d'entrées / sorties résident dans des unités séparées (modules) et sont fixées sur un ou plusieurs racks contenant le "Fond de panier" (bus plus connecteurs). Ces automates sont intégrés dans les automatismes complexes où puissants, où la capacité de traitement et flexibilité sont nécessaires.

Figure II.6 Exemple d'un automate de type modulaire

➢ ***Remarque :*** *Les automates modulaires permettent de réaliser de nombreuses autres fonctions grâce à des modules intelligents que l'on dispose sur un ou plusieurs racks. Ces modules ont l'avantage de ne pas surcharger le travail de la CPU car ils disposent bien souvent de leur propre processeur.*

Un automate de type modulaire est constitué essentiellement des éléments suivants :

1. **Module d'alimentation**
2. **Pile de sauvegarde**
3. **Connexion au 24V cc**
4. **Commutateur de mode (à clé)**
5. **LED de signalisation d'état et de défauts**
6. **Carte mémoire**
7. **Interface multipoint (MPI)**
8. **Connecteur frontal**
9. **Volet en face avant**

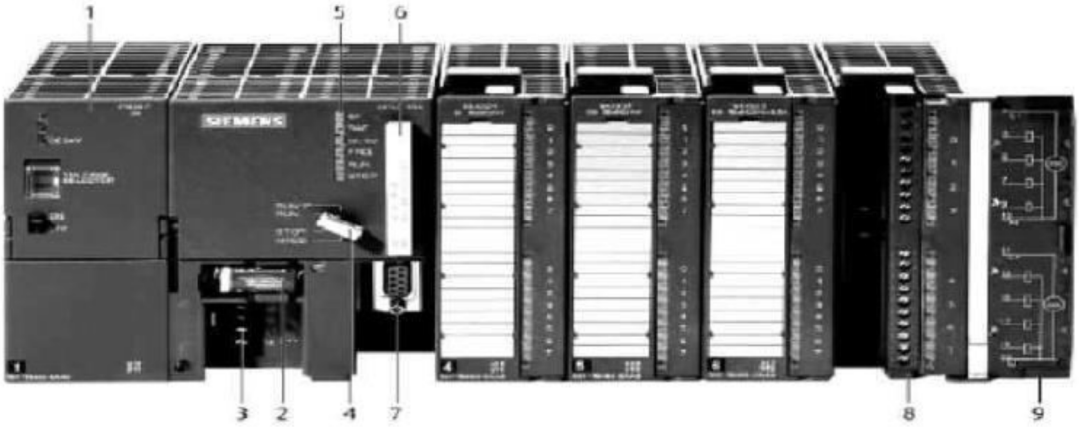


Figure II.7 Exemple des éléments essentiels d'un automate de type modulaire

II.5- Structure interne d'un automate

Les API comportent quatre parties principales : une mémoire, un processeur, des interfaces d'entrées/sorties et d'une alimentation (240Vac, 24Vcc). Ces quatre parties sont reliées entre elles par des bus (ensemble de câbles autorisant le passage de l'information entre ces 4 secteurs de l'API).

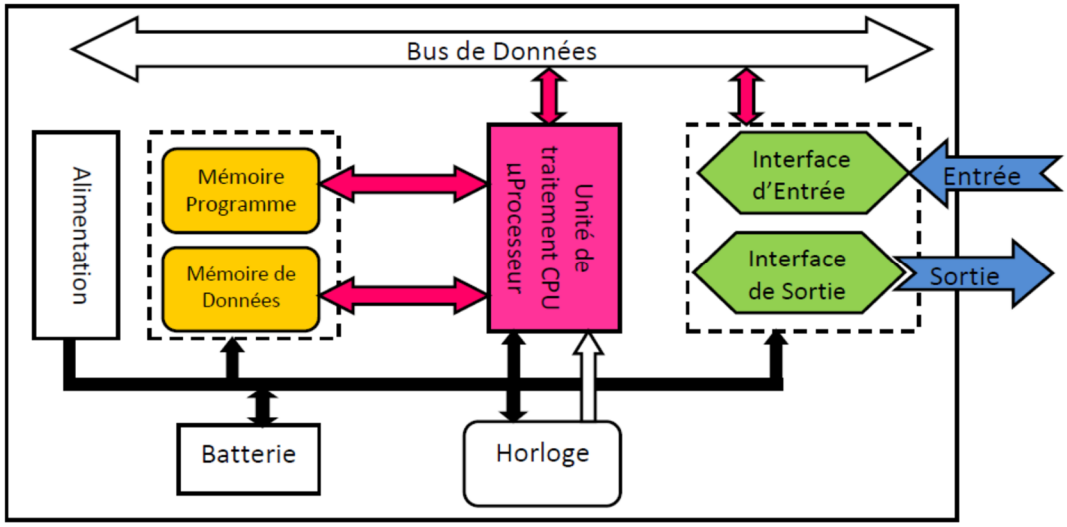


Figure II.8 Structure interne d'un API

II.6- Cycle d'un API

L'API a une caractéristique unique, c'est le fonctionnement cyclique de l'unité centrale. Ce cycle se reproduit indéfiniment, pour chaque cycle tout le programme est exécuté. La durée d'un cycle est de l'ordre de 20ms. Tous les automates fonctionnent selon le même mode opératoire :

- ✓ **Traitement interne :** l'automate effectue des opérations de contrôle et met à jour certains paramètres systèmes (détection des passages en RUN/STOP....) ;
- ✓ **Lecture des entrées :** l'automate lit les entrées (de façon synchrone) et les recopie dans la mémoire image des entrées ;

- ✓ **Exécution du programme :** l'automate exécute le programme instruction par instruction selon la logique programmée stockée dans la mémoire et écrit les sorties dans la mémoire image des sorties ;
- ✓ **Ecriture des sorties :** Lecture des variables de sorties dans la RAM données et transfert vers le module de sorties (de façon synchrone) aux positions définies dans la mémoire image des sorties.

Le temps de scrutation de chaque cycle est vérifié par un temporisateur appelé ***Watchdog*** *(chien de garde)* qui enclenche une procédure d'alarme en cas de dépassement de celui-ci (réglé par l'utilisateur).

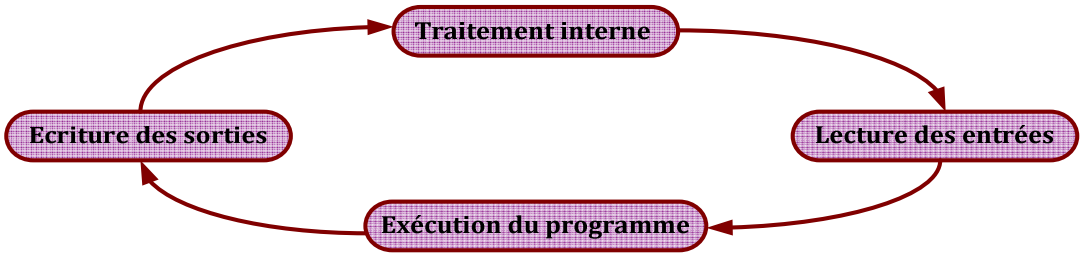


Figure II.9 Cycle d'un API

II.7- Critères de choix d'un API

Le choix d'un automate programmable est, en premier lieu, le choix d'une société ou d'un groupe où les contacts commerciaux et expériences vécues sont déjà un point de départ. Les grandes sociétés privilégieront deux fabricants pour faire jouer la concurrence et pouvoir "se retourner" en cas de "perte de vitesse" de l'une d'entre elles.

Le personnel de maintenance doit toutefois être formé sur ces matériels et une très grande diversité des matériels peut avoir de graves répercussions. Un automate utilisant des langages de programmation de type GRAFCET est également préférable pour assurer les mises au point et dépannages dans les meilleures conditions. La possession d'un logiciel de programmation est aussi source d'économies (achat du logiciel et formation du personnel). Des outils permettant une simulation des programmes sont également souhaitables. Il faut ensuite quantifier les besoins :

- ✓ Nombre d'entrées/sorties : le nombre de cartes peut avoir une incidence sur le nombre de racks dès que le nombre d'entrées / sorties nécessaires devient élevé ;
- ✓ Type de processeur : la taille mémoire, la vitesse de traitement et les fonctions spéciales offertes par le processeur permettront le choix dans la gamme souvent très étendue ;
- ✓ Fonctions ou modules spéciaux ;
- ✓ Fonctions de communication : l'automate doit pouvoir communiquer avec les autres systèmes de commande (API, supervision, ...) et offrir des possibilités de communication avec des standards normalisés (Profibus).

II.8- Nature des informations traitées par l'automate

a- Entrées / sorties TOR

Les API offrent une grande variété d'E/S TOR adaptées au milieu industriel et qui peuvent accepter suivant les cartes, des informations en courant ou en tension, alternatifs ou continus.

- **Entrées TOR** : Les modules d'entrée TOR permettent de recevoir les signaux des différents détecteurs et capteurs logiques (thermostats, fins de course, capteur de proximité, photo-électriques, roues codeuses) ou de simples éléments du pupitre (Boutons poussoir, commutateur ou interrupteur qui peuvent être fermés ou ouverts). Ce qui fait que l'information délivrée par ces capteurs et qui sera traitée par la CPU ne peut prendre que deux valeurs 0 ou 1. Le cheminement d'un signal est le suivant : l'adaptation et la protection, le filtrage pour éviter les parasites et l'isolement électrique de l'unité de commande et la partie opérative pour assurer la fiabilité et la sécurité du signal électrique.

- **Sorties TOR** : Comme les modules d'entrées TOR, les modules de sorties TOR connectent des dispositifs de sorties (actionneurs ou prés-actionneurs tels que : vannes, contacteurs, voyants, électrovannes, relais de puissance, afficheurs, etc.) de champ à l'automate programmable. Ce type de sortie n'ayant que deux états (ON / OFF, OUVRIR / FERMER, VRAI / FAUX, etc.). Les modules comportent généralement 8, 16, 32 sorties logiques via des tensions de commande de 12 VDC, 24 VDC, 48VDC, 120Vac, etc. Les courants variant de quelques mA à quelques A. Le découplage se fait par transformateur d'isolement en alternatif ou par optoélectronique en continu. La puissance de sortie est souvent suffisante pour commander directement des vannes ou des petits moteurs.

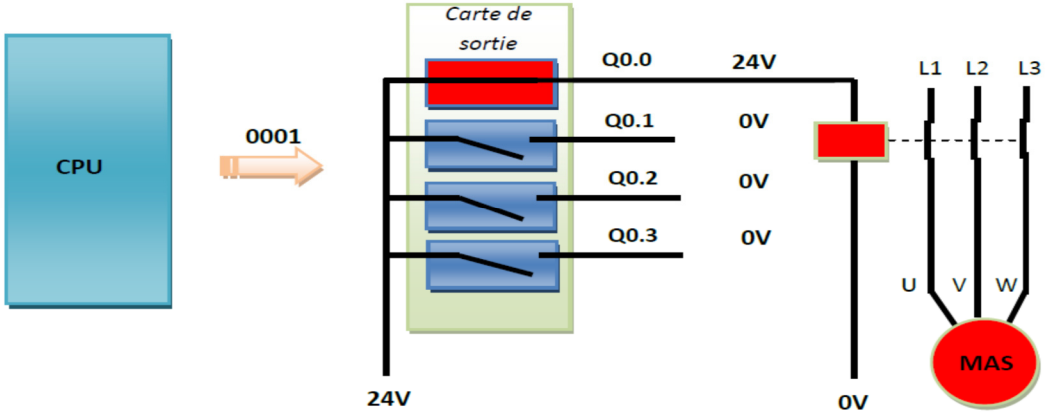


Figure II.10 Principe de commande des sorties état actionnée

Remarque : *Pour assurer la sécurité du procédé, il est parfois indispensable que certaines sorties soient protégées contre les incidents pouvant survenir sur l'API tels que les microcoupures et les coupures de l'alimentation.*

b- Entrées sorties numériques

Utilisés pour les API haute ou moyenne gamme effectuant des traitements numériques. L'information est sous la forme, soit d'un train d'impulsions (signal carré) dont le nombre ou la fréquence est l'image de la grandeur d'entrée, soit d'un code numérique codé sur (n) variables binaires. On trouve parmi les principaux capteurs numériques industriels, les capteurs de positions angulaires incrémentaux, les codeurs absolus, les lecteurs de code à barres et les lecteurs de pistes magnétique.

La longueur d'un module est définit par la taille du mot mémoire de l'API, où il ne peut prendre qu'un nombre limité de valeurs distincte (ex : 16 bits). Les cartes d'E/S numériques se présente donc comme 16 E ou S binaires rassemblées, pour lesquels on utilise les mêmes précautions d'isolement. Souvent, on utilise une seule carte d'E/S numérique et on multiplexe les diverses entrées numériques souhaitées à l'aide de sorties binaires. Cette idée sera utilisée de la même façon pour le multiplexage d'E binaires ou le démultiplexage des sorties. Elle permet artificiellement de multiplier le nombre d'E/S physiques sans augmenter le nombre d'E/S machine.

Remarque*: Sur de nombreux automates les E/S binaires ou numériques sont banalisées, pour certains, les E/S numériques sont obtenues par regroupement du nombre de bits nécessaires parmi les E/S binaires. Pour d'autres, les E/S binaires sont en réalité des éléments des E/S numériques. L'interprétation est effectuée par le moniteur selon les directives de l'utilisateur, exemple : U_{12} repère l'entrée binaire n°12 alors que UN_{12} identifie l'entrée numérique n°12.*

c- Entrées sorties analogiques (EA/SA)

Les cartes d'entrées analogiques sont utilisées dans les applications où le signal est continu. Contrairement aux signaux TOR, qui ne possèdent que deux états (ON et OFF), les signaux analogiques ont un nombre infini d'états Les E/S analogiques transforment une grandeur analogique en une valeur numérique par l'intermédiaire d'un convertisseur

analogique-numérique (la conversion en numérique (CAN) est indispensable pour assurer un traitement par le microprocesseur) et vice versa. Le rôle du transducteur dans la figure II.11 est de transformer le signal d'entrée en un signal électrique normalisé que l'entrée analogique peut reconnaître (exemple 0-100 bar en 4-20mA). Notant que le transducteur génère un signal électrique de très faible niveau (courant ou tension), donc on doit amplifier ce signal par l'utilisation des transmetteurs (Tension / intensité) permettent d'adapter les signaux issus pour les rendre compatibles avec l'unité de traitement, ces signaux générés à leurs tours sont envoyés à la carte d'entrée analogique.

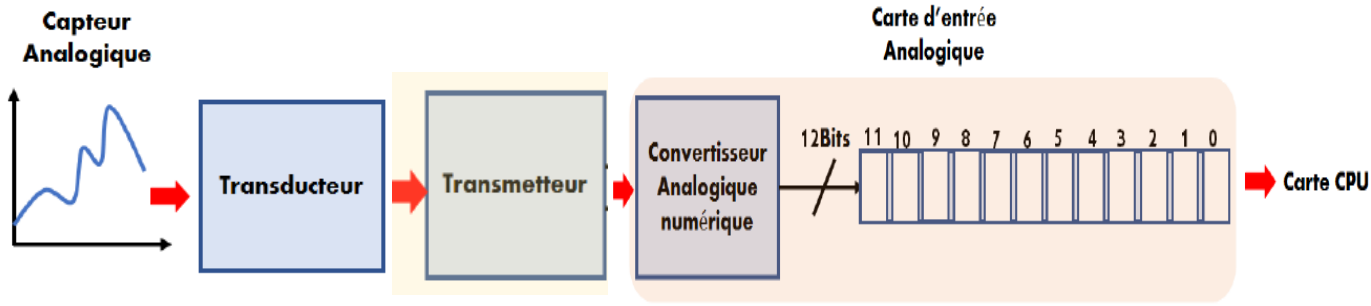


Figure II.11 Principe de fonctionnement d'une carte d'entrée analogique

Technologiquement, les EA/SA sont caractérisées par l'amplitude du signal analogique. En raison des nombreux types de transducteurs disponibles au marché, les cartes d'entrées analogiques ont plusieurs caractéristiques électriques standards (+ 4-20mA, + 0 à +1 V(DC), + 0 à +5 V(DC), + 0 à +10 V(DC), + 1 à +5 V(DC), ±5 V(DC), ±10 V(DC)) mais typiquement 0/10V ou –10/+10V) et par le courant correspondant. Les EA : En générale on opère sur 12 bits (ou 11 bits + 1 bit de signe si nécessaire) = 4096 niveaux quantifiés. Cela signifie que la valeur de tension de 10 V par exemple sera stockée comme information ayant la forme d'une rangée de chiffres binaires.

Comme les entrées analogiques, les cartes de sorties analogiques sont généralement connectées à des dispositifs de contrôle via des transducteurs (voir Figure II.12). Ces transducteurs amplifient, réduisent ou modifient le signal de tension en un signal analogique qui, à son tour de contrôler l'actionneur de sortie.

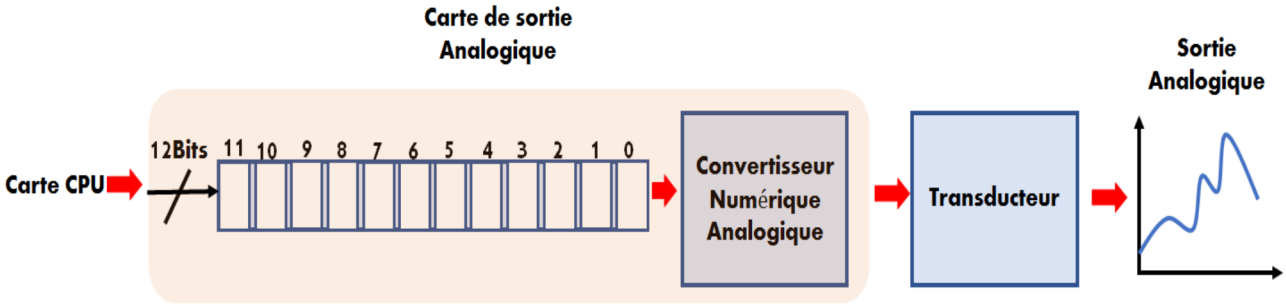


Figure II.12 Sortie analogique

***Remarque** : Le coût du dispositif de conversion impose le multiplexage des voies analogiques : cette contrainte économique et technologique induit un retard de l'ordre de 1µs (temps de multiplexage pour multiplexeur CMOS ! ! !). Des solutions technologiques existent pour accroître les performances des EA : Utiliser des EA intelligent autonomes et effectuant des prétraitements localement. · Les SA : Ces modules assurent la conversion numérique analogique. L'intensité ou la tension de sortie est proportionnelle à la valeur numérique. Les sorties analogiques peuvent posséder un convertisseur par voie. Le nombre des voies sur ces cartes est 2 ou 4.*

d- Entrées sorties spécialisées

Toutes ces cartes dites intelligentes disposent en plus des interfaces d'entrée et de sortie, d'un véritable micro-ordinateur assurant un traitement local plus ou moins sophistiqué. Ceci permet, d'une part d'éviter un développement souvent fastidieux de l'application, d'autre part

de réduire parfois considérablement la place mémoire et le temps d'exécution au niveau de l'UC de l'automate programmable. Parmi les types existant en trouvant :

- Compteurs et temporisateurs : permettent de gagner du temps (comptage électronique) ou encore d'économiser de l'espace mémoire (temporisation).
- Entrées logiques à seuil ajustable : détectent le franchissement d'un seuil par une grandeur physique continue.
- Modules intelligents : conçus souvent à base de microprocesseur, ils assurent de façon autonome certaines fonctions d'automatismes : modules de positionnement / commande d'axes, de régulation numérique, de communication, ...

II.9- Présentation de l'automate S7 – 300

L'automate programmable industriel S7 – 300 fabriqué par SIEMENS (Figure II.13), qui fait partie de la gamme SIMATIC S7 est un automate destiné à des tâches d'automatisation moyennes et hautes gammes. L'automate lui-même est constitué d'une configuration minimale composée d'un module d'alimentation, de la CPU, du coupleur et de modules d'entrées/sorties.

Les profilés supports ou les châssis (rack) constituent des éléments mécaniques de base de la SIMATIC S7 – 300, ils remplissent les fonctions suivantes :

- ✓ La fixation des modules ou l'assemblage mécanique des modules ;
- ✓ La distribution de la tension ;
- ✓ L'acheminement du bus de fond de panier aux différents modules.

Dans le S7 – 300 les modules sont fixés dans l'ordre et leurs nombres sont limités c'est-à-dire que le profilé support dans le S7 – 300 contient au maximum 11 emplacements.

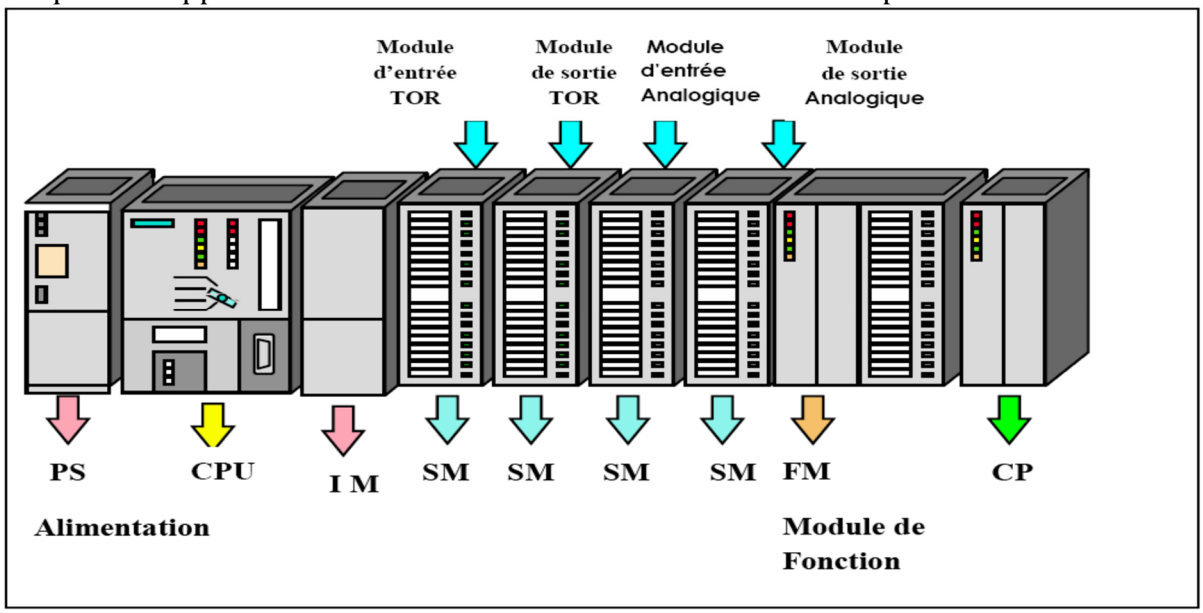


Figure II.13 Schéma représentatif de l'API S7-300

II.9.1- Modules d'alimentations (PS) : Le module d'alimentation assure la conversion de tension du secteur (ou du réseau) en tension de (24V, 48V, 120V ou 230V) pour l'alimentation de l'automate et des capteurs et actionneurs.

- Il remplit aussi des fonctions de surveillance et signalisation à l'aide des LEDS.
- Il permet de sauvegarder le contenu des mémoires RAM au moyen d'une pile de sauvegarde ou d'une alimentation externe.

II.9.2- Unités centrales (CPU) : L'unité centrale commande l'interprétation et l'exécution des instructions programmées. Elle est aussi chargée de détecter les pannes de communication, ainsi que d'autres défaillances qui peuvent survenir pendant le fonctionnement du système. Il doit alerter l'opérateur ou le système en cas de dysfonctionnement. A base de microprocesseur, l'unité centrale réalise toutes les fonctions logiques, arithmétiques et de

traitement numérique (transfert, comptage, temporisation ...). Ce module se compose essentiellement de :

a- ***Microprocesseur :*** Il constitue le cœur de la CPU. Son rôle consiste, d'une part, à organiser les différentes relations entre la zone mémoire et les interfaces d'E/S et, d'autre part, à gérer les instructions du programme.

b- ***Bus :*** C'est un ensemble de pistes conductrices (pistes en cuivre) par lesquelles s'achemine une information binaire (suite de 0 ou 1), c'est-à-dire ensemble de fils autorisant le passage des informations entre les quatre secteurs (l'alimentation, la mémoire, le processeur et l'interface E/S) de l'automate. L'unité centrale dispose de trois bus : bus de données, bus d'adresses et bus de commandes.

c- ***Mémoire :*** Elle est conçue pour recevoir, gérer et stocker des informations issues des différents secteurs du système qui sont :

- Le terminal de programmation ;
- Le processeur, qui lui gère et exécute le programme ;
- Elle reçoit également des informations en provenance des capteurs.

d- ***Interfaces d'entrées / sorties :*** Les interfaces d'entrées / sorties permettent à l'unité centrale de communiquer avec l'environnement ou les périphériques.

La CPU est le cerveau de l'automate elle permet de :

- Lire les états des signaux d'entrées.
- Exécuter le programme utilisateur et commander les sorties.
- Régler le comportement au démarrage et diagnostiquer les défauts par les LEDS.

Le S7 – 300 dispose d'une large gamme de CPU à différents niveaux de performance, on compte les versions suivantes :

- ✓ CPU à utilisateur standard : CPU 313, CPU 314, CPU 315 et CPU 316 ;
- ✓ CPU avec fonctions intégrées : CPU 312 IFM et la CPU 314 IFM ;
- ✓ CPU avec interface PROFILBUS DP (CPU 315–2 DP, CPU 316–2 DP- CPU 318–2 DP).

- **Caractéristiques de la CPU :** La gamme S7-300 offre une grande variété de CPU tels que la CPU312, 314M, 315 ,315-2P, etc. Chacune possède certaines caractéristiques différentes des autres. Par conséquent, le choix de la CPU, pour un problème d'automatisation donné, est conditionné par les caractéristiques offertes par la CPU choisie.

e- Interface (MPI) : Une liaison MPI (Multi Point Interface) est nécessaire pour programmer un SIMATIC S7 300 depuis le PC. Elle est une interface de communication utilisée pour la programmation, le contrôle-commande avec HMI et l'échange de données entre CPU CIMATIC S7 jusqu'à 32 nœuds maximum. Chaque CPU du SIMATIC S7-300 est équipée d'une interface MPI intégré.

f- Signification d'état : La CPU comporte des LED de signalisation suivante :

- ✓ **SF (rouge) :** signalisation groupée de défauts qui s'allume si on a de défauts matériels et en cas d'erreurs de programmation, de paramétrage, de calcul, etc.
- ✓ **BATF (rouge) :** défaut pile qui s'allume si elle est défectueuse, absente ou déchargée.
- ✓ **DC5V (verte) :** alimentation 5Vcc pour la CPU et le bus S7-300 qui s'allume si les 5V sont présente et elle clignote s'il y a surcharge de courant.
- ✓ **FRCE (jaune) :** forçage permanent qui s'allume en cas de forçage permanent.
- ✓ **RUN (verte) :** état de fonctionnement RUN qui clignote en cas de démarrage de la CPU.
- ✓ **STOP (jaune) :** état de fonctionnement STOP qui s'allume si la CPU ne traite aucun programme utilisateur et clignote en cas ou la CPU demande un effacement général.

Les LED de signalisation de défauts **SF-DP** et **BUSF** ne se rencontrent que dans le cas de la CPU314 relative à la configuration maitre-esclave du S7-300.

Figure II.14 Schéma de CPU d'un API S7-300

g- Commutateur de mode : Les différents modes existants sont les suivants

- ✓**RUN-P (mode de fonctionnement RUN programme)**: la CPU traite le programme utilisateur et la clé ne peut être retirée. Il est possible de lire le programme de la CPU avec une PG (CPU vers PG) et de transférer des programmes dans la CPU (PG vers CPU) ;
- ✓**RUN (mode de fonctionnement RUN)** : la CPU traite le programme de l'utilisateur. Dans cette position, la clé peut être retirée pour éviter qu'une personne non habilitée change le mode de fonctionnement.
- ✓**STOP (mode de fonctionnement STOP)** : la CPU ne traite aucun programme utilisateur, la clé peut être retirée pour éviter le changement de mode inattendue mais on peut lire et écrire dans la CPU.
- ✓**MRES** : c'est position instable du commutateur de mode de fonctionnement, en vue de l'effacement générale de la CPU. Le contenu de la mémoire de chargement rémanente intégré reste inchangé après un effacement général.

h- La pile : Elle permet de sauvegarder le contenu de la RAM en cas de coupure du courant.

II.9.3-Coupleur (IM) : Les coupleurs sont des cartes électroniques qui assurent la communication entre les E/S (périphéries ou autre) et l'unité centrale. L'échange de l'information entre la CPU et les modules d'E/S s'effectue par l'intermédiaire d'un bus interne (liaison parallèle codée). Les coupleurs ont pour rôle le raccordement d'un ou plusieurs châssis au châssis de base. Pour l'API S7 – 300, les coupleurs disponibles sont :

- IM 365 : Pour les couplages entre les châssis d'un mètre de distance au max.
- IM 360 et IM 361 : pour les couplages allant jusqu'à 10 mètres de distances.

II.9.4-Module communication (CP) : Les modules de communication sont destinés aux tâches de communication par transmission en série. Ils permettent d'établir également des liaisons point à point avec :

- Des commandes robots.
- Communication avec des pupitres opérateurs.
- Des automates SIMATIC S7, SIMATIC S5 et des automates d'autres constructeurs.

II.9.5-Modules de fonctions (FM) : Ces modules réduisent la charge de traitement de la CPU en assurant des tâches lourdes de calculs. On peut citer les modules suivants :

- FM 354 et FM 357 : Module de commande d'axe pour servomoteur.
- FM 353 : Module de positionnement pour moteur pas à pas.
- FM 355 : Module de régulation.
- FM 350 – 1 et FM 350 – 2 : Module de comptage.

- **Application pratique :** Chaque processus industriel de fabrication ou de transformation se compose d'un ensemble de machines destinées à réaliser la fabrication ou la transformation considérée. Chaque machine ou partie opérative comprend un ensemble de moteurs, vannes et autres dispositifs qui lui permet de fonctionner. Ces actionneurs sont pilotés par un automate (partie commande) à travers les pré-actionneurs.
  - La partie commande reçoit des informations transmises par un opérateur en fonctionnement normal, ou un dépanneur en cas de réglage ou de mauvais fonctionnement de la partie commande ou de la partie opérative ainsi que par les capteurs. Entre la partie commande et l'homme se trouve la partie dialogue qui permet à ce dernier de transmettre au moyen de dispositifs adaptés (bouton poussoirs, commutateurs, etc.). De même, la partie commande renvoie vers l'homme les informations sous des formes compréhensibles (voyant, afficheurs, cadrans, etc.). Ainsi, entre l'homme et la partie opérative, s'instaure un dialogue homme machine dont l'importance naguère sous-estimé et aujourd'hui reconnue, et qui est actuellement l'objet de nombreuses études.

II.9.6-Modules de signaux (SM) : Ils servent d'interface entre le processus et l'automate. Ils existent des modules d'entrées/sorties TOR, ainsi que des modules d'entrées/sorties analogiques.

- ✓ **Modules de simulation (SM 374) :** Le module de simulation SM 374 est un module spécial qui offre à l'utilisateur la possibilité de tester son programme lors de la mise en service en cours de fonctionnement. Dans le S7 – 300, ce module se monte à la place d'un module d'entrée ou de sortie TOR. Il assure plusieurs fonctions telles que :

- La simulation des signaux de capteurs aux moyens d'interrupteurs ;
- La signalisation d'état des signaux de sorties par des LEDS.

II-10 Nature des mémoires

a- Mémoire : Les automates actuels utilisent essentiellement des mémoires à semi-conducteurs. Comme on sait, ce type de mémoire pose le problème de la perte d'information en cas de coupure d'alimentation. Cela a conduit au développement de deux grandes familles de mémoires : les mémoires vives (RAM), permettant la lecture et l'écriture mais "volatiles", et les mémoires mortes (ROM), n'autorisant que la lecture mais non volatiles. Dans les automates, on a des exigences différentes pour la mémoire de données et la mémoire de programme.

b- Mémoire de données : C'est clairement la place ici, de mémoires de type RAM. Même dans le cas de mémoires RAM, il est possible de se prémunir contre des coupures accidentelles d'alimentation en utilisant une technologie spéciale (CMOS) à très faible consommation électrique du moins, à l'état de repos (5000 fois moins qu'une RAM classique). On peut en effet dans ce cas prévoir, au niveau des cartes de mémoire, des micro-batteries, des micro-piles voire des condensateurs, susceptibles d'alimenter les

circuits en cas de coupure de l'alimentation principale. Les temps de sauvegarde garantis vont de quelques jours à quelques mois.

c- **Mémoire de programme :** La mémoire de programme de l'automate doit pouvoir subir sans dommage des coupures plus ou moins longues d'alimentation (entretiens, congés, etc.). C'est la technologie EPROM (Erasable PROM) que l'on rencontre le plus fréquemment ici. Elle présente l'inconvénient que les circuits de mémoire doivent être programmés et effacés (par exposition aux UV dans un dispositif extérieur à l'automate). Il en résulte des manipulations particulièrement gênantes en phase de mise au point de l'équipement. La technologie EEPROM (Electrically Erasable PROM), de développement plus récent, commence à apparaître chez certains constructeurs. L'effacement se fait ici par voie électrique et peut donc être réalisée in situ. Enfin, signalons qu'une certaine confiance s'est établie, à l'usage, dans les mémoires RAM gardées et que beaucoup de constructeurs proposent une option de ce type pour la mémoire programme de leurs automates.

d- **Mémoire système :** Cette mémoire, présente dans le cas d'automates à microprocesseurs, est programmée en usine par le constructeur ; Elle peut donc sans problème être réalisée en technologie EPROM voire PROM (c'est-à-dire programmable une seule fois, sans possibilité d'effacement).

II-11 Mode d'emploi

Avant de créer un projet et de procéder à la programmation, on peut envisager différentes approches. En effet, les différents logiciels en général (STEP 7 par exemple), offrent une liberté, de choix de la procédure à adapter.

Du moment que notre projet contient beaucoup d'entrées et de sorties, il est préférable de commencer par configurer le matériel avant la création du programme. L'application de la configuration matérielle de logiciel présente l'avantage que les adresses y sont sélectionnées pour nous. La configuration matérielle est une étape très importante, elle permet de reproduire à l'identique le système utilisé (alimentation, CPU, module, etc.). Par la configuration, on entend dans ce qui suit la disposition de profilé – support ou châssis, de module, d'appareils de la périphérie décentralisée et de cartouche interface dans une fenêtre de station.

Si nous choisissons la seconde alternative (*voir la figure ci-dessous*), il nous faudra rechercher nous-même les adresses en fonctions des constituants choisis.

La configuration matérielle nous permet non seulement de sélectionner les adresses, mais également de modifier les paramètres et les propriétés des modules.

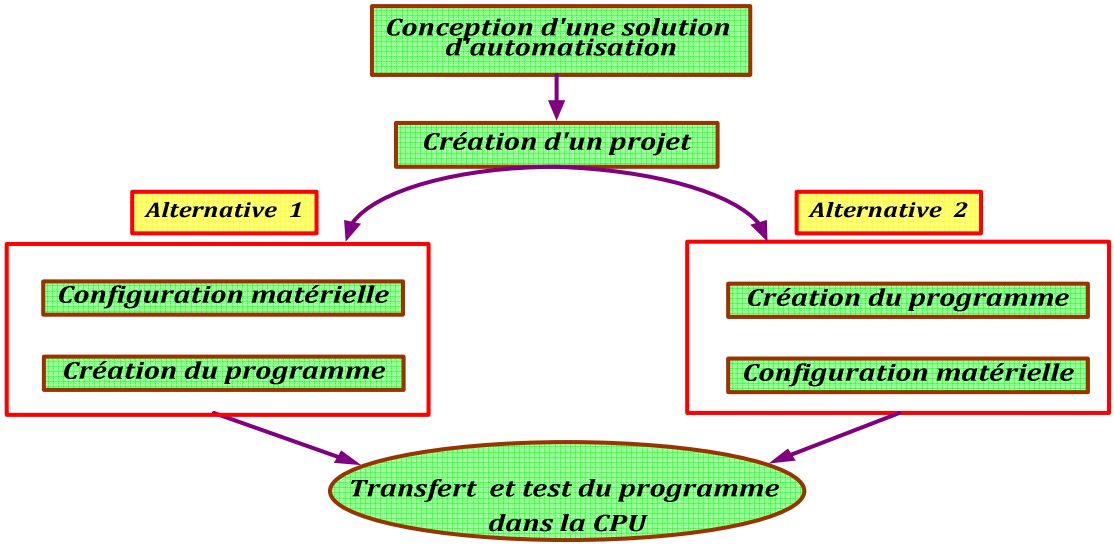


Figure II.15 Différentes étapes de programmation d'un automate

II.12- Mise en équation d'un Grafcet

La plupart des automates ne se programment pas en GRAFCET directement. Mais, généralement ils peuvent être programmés en LADDER. Donc, Il faut transformer le grafcet en des équations afin de traiter les systèmes séquentiels en langage de programmation. Pour la mise en équation d'un Grafcet il faut suivre les étapes suivantes :

a- Calcul des franchissabilités : la franchissabilité d'une transition est une fonction logique F_i (F_i=1 ou F_i =0), d'après la deuxième règle d'évolution du Grafcet, une transition est franchie (F_i =1) si toutes les étapes immédiatement précédentes sont actives (X_i=1) et que la réceptivité associée à cette transition soit vraie (r_i=1). $F_i = X_i \,.\, r_i$

b- Franchissement d'une transition : pour chacune des transitions, on applique la troisième règle d'évolution du Grafcet à travers l'évaluation de sa fonction de franchissabilité (F_i) :

- Si F_i =1, les étapes immédiatement précédentes sont désactivées par contre les étapes immédiatement suivantes sont activées :

$$\overline{X_\iota} = F_i \text{ Et } X_i = F_i \qquad \text{(II.1)}$$

- Si F_i =0, le Grafcet reste dans son état sans évolution.

***NB**. Dans le cas où le Grafcet contient des réceptivités dynamiques, on devrait définir les variables de retard, la façon de définition de ces variables dépend de type d'automate utilisé.*

c- Initialisation et gestion des modes Marche/Arrêt : A l'initialisation du grafcet, toutes les étapes autres que les étapes initiales sont désactivées. Seules les étapes initiales sont activées. Soit la variable logique ***Init*** telle que :

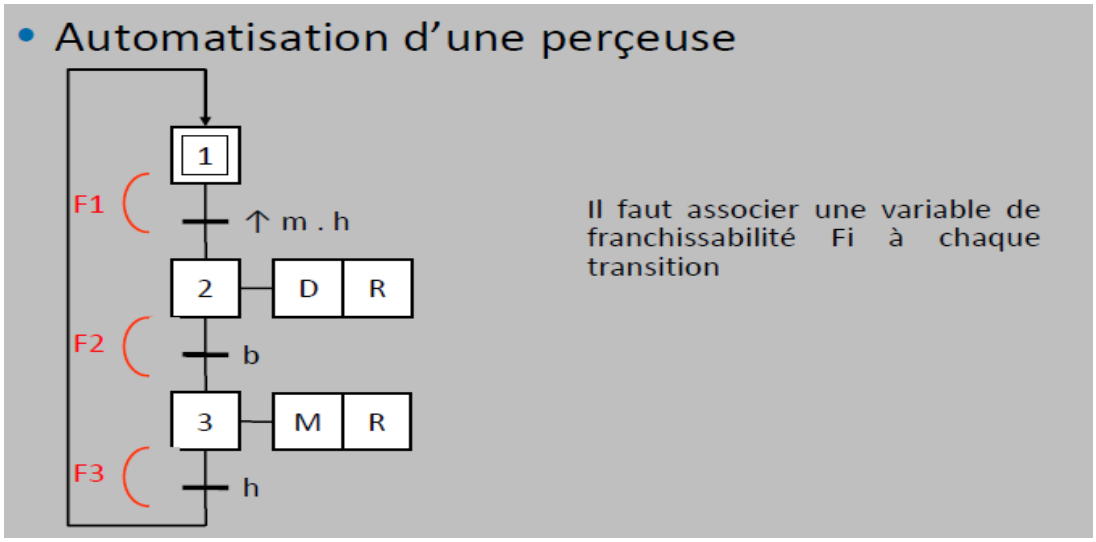
- *Init*= 1 : Initialisation du grafcet : mode Arrêt ;
- *Init*=0 : déroulement du cycle : mode Marche.

- *Etape initiale :* $X_i = Init$;
- *Etape classique :* $\overline{X_\iota} = Init$.

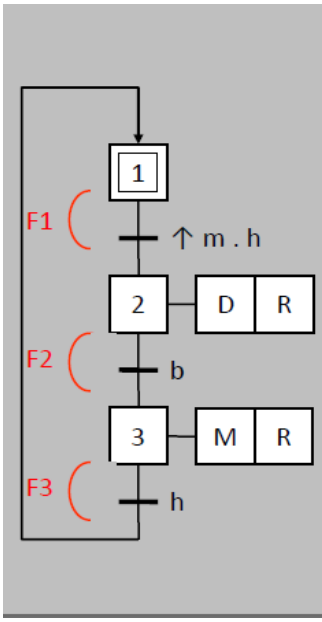
d- Actions : L'activation est forcément à l'activation de l'étape à laquelle est associe.

- **Action simple**
  - Action inconditionnelle : $A = X_i$;
  - Action conditionnelle : $A = X_i \,.\, C$.
- **Action mémorisée**
  - Action inconditionnelle mémorisée à l'appel : $A \approx X_i$;
  - Action conditionnelle mémorisée à l'appel : $A \approx X_i . C$.

Exemple



- Mise en équation de Grafcet généré :



Étiquette	N° de ligne	Instruction
FRA	0	F1=X1.h.↑m
	1	F2=X2.b
	2	F3=X3.h
FTR	3	X1 ≈ F3
	4	$\overline{X1}$ ≈ F1
	5	X2 ≈ F1
	6	$\overline{X2}$ ≈ F2
	7	X3 ≈ F2
	8	$\overline{X3}$ ≈ F3
RET	9	
INI	10	X1 ≈ I
	11	$\overline{X2}$ ≈ I
	12	$\overline{X3}$ ≈ I
ACT	13	R=X2+X3
	14	D=X2
	15	M=X3

II.13- Langages de programmation pour API

Chaque automate possède son propre langage. Mais par contre, les constructeurs proposent tous une interface logicielle répondant à la norme CEI1 1131-3. Cette norme définit cinq langages de programmation utilisables, qui sont :

- ✓ Les langages graphiques :
  - **LD** : **L**adder **D**iagram (Diagrammes échelle) ;
  - **FBD** : **F**unction **B**lock **D**iagram (Logigrammes) ;
  - **SFC** : **S**equential **F**unction **C**hart (Grafcet) .
- ✓ Les langages textuels :
  - **IL** : **I**nstruction **L**ist (Liste d'instructions) ;
  - **ST**: **S**tructured **T**ext (Texte structuré).

II.13.1- GRAFCET ou Sequential function char (SFC)

Ce langage de programmation de haut niveau permet la programmation aisée de tous les procédés séquentiels, par une représentation graphique et de façon structurée. Il est dérivé du grafcet.

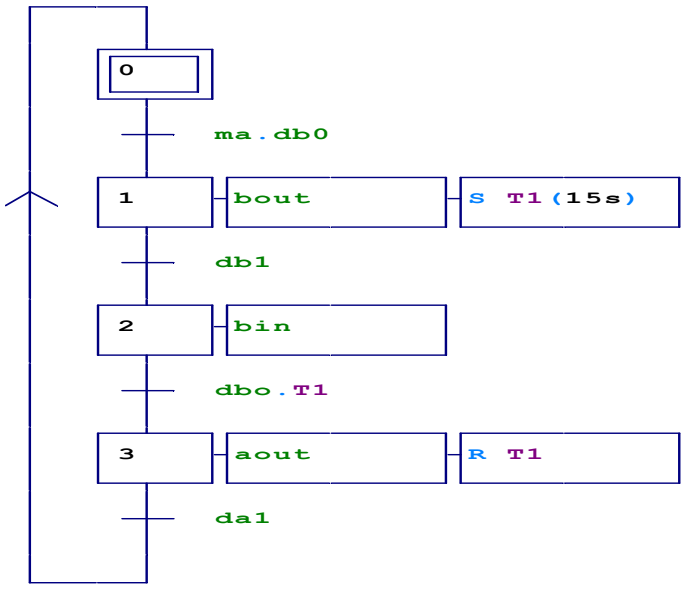
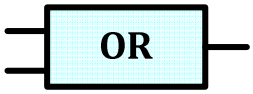


Figure II.16 Exemple de programmation par le langage SFC (logiciel AUTOMGEN)

II.13.2- Schéma par blocs ou boites fonctionnelles (FBD)

C'est un langage graphique permettant d'exprimer le comportement des fonctions, des blocs fonctionnels ou des programmes simples ou très sophistiquées, comme un ensemble de blocs interconnectées (à la manière des portes logiques en électronique) et qui sont programmés (bibliothèque) ou programmables. Ainsi, une seule opération FBD peut représenter la même fonction qu'un ensemble d'instructions ou de contacts (bobines ou boîtes en schéma à contact). Dans ce qui suit en vas représenter quelques blocs fonctionnels, qu'ils sont utilisés pour programmer l'automate programmable industriel.

Cette boîte représente la fonction ET en associant deux bits que l'on peut inverser à l'entrée de la boîte. Il est possible de rajouter jusqu'à 32 entrées.

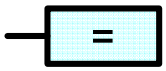


Cette boîte représente la fonction OU en associant deux bits que l'on peut inverser à l'entrée de la boîte. Il est possible de rajouter jusqu'à 32 entrées.

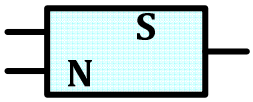
Inverser une entrée

Contact à front montant.

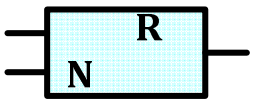
Contact à front descendant.



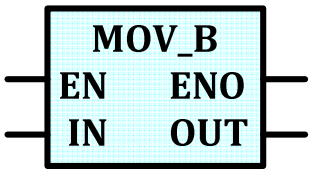
L'opération sortie écrit la nouvelle valeur du bit de sortie dans la mémoire image.



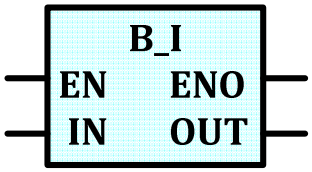
L'opération SET met à 1 un nombre N de sorties à partir de l'adresse bit indiquée.



L'opération RESET met à 0 un nombre N de sorties à partir de l'adresse bit indiquée.



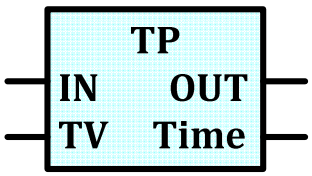
Les opérations de transfert permettent de transférer l'octet d'entrée (IN) dans l'octet de sortie (OUT). Cette fonction est utilisable avec des mots (MOV_W), des doubles mots (MOV_DW), des réels (MOV_R), …



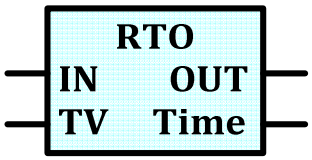
Les blocs de conversion permettent de convertir un nombre d'une base donnée dans une autre base (vu dans le module 2). Il est également possible de tronquer ou arrondir un nombre, de l'encoder ou le décoder.

Si l'entrée IN est activée le temporisateur TON va déclencher la temporisation pendant le temps alloué TV (retard à la montée), après ce temps, le TON va activer la sortie OUT, Time pour l'affichage de TV.

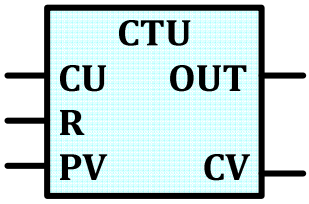
Une fois l'entrée IN est désactivée le temporisateur TOF va déclencher la temporisation pendant le temps alloué TV (retard à la descente), après ce temps, le TOF va désactiver la sortie OUT, Time pour l'affichage de TV.



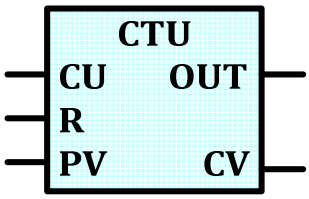
Ce type de temporisateur donnera une sortie de OUT=1, pour une durée fixe (impulsion de temps=TV) quand il y a une entrée à 1, après cette temporisation la sortie devient 0, Time pour l'affichage de TV.



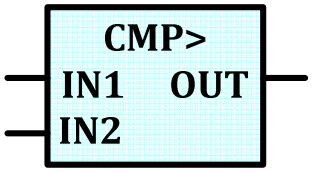
Le type de temporisateur RTO est très similaire au type TON et TOF, à l'exception du fait que la valeur cumulée (TV=TV1+TV2) est conservée même si les conditions des entrées sont à l'état zéro.



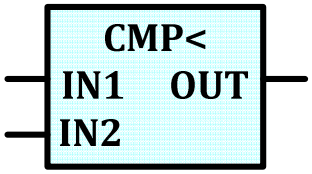
Le fonctionnement de ce type de compteur (CTU) est d'incrémenter avec un une valeur, chaque fois qu'un événement se produit à l'entrée d'incrémentation CU, PV est la valeur prédéfinie et R la remise à zéro, une fois cette valeur incrémentée =PV en aura OUT=1, CV pour l'affichage de la valeur incrémentée.



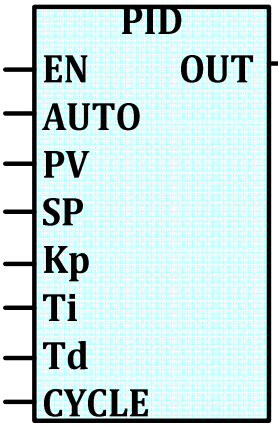
Le fonctionnement de ce type de compteur (CTD) est de décrémenter avec un une valeur, chaque fois qu'un événement se produit à l'entrée de décrémentation CD, PV est la valeur prédéfinie et R la remise à zéro, une fois cette valeur décrémentée =0 en aura OUT=1, CV pour l'affichage de la valeur décrémentée.



Le rôle de ce comparateur est de comparer la valeur mesurée IN1 avec la référence de IN2, si elle est supérieure à cette référence le comparateur va activer la sortie OUT=1.



Le rôle de ce comparateur est de comparer la valeur mesurée IN1 avec la référence de IN2, si elle est inférieure à cette référence le comparateur va activer la sortie OUT=1.



- **EN** : Permet d'activer ou désactiver le régulateur PID;
- **AUTO** : Permet de sélectionner le mode automatique ou le mode manuel du régulateur ;
- **PV** : La mesure du processus ;
- **SP** : La consigne (référence) ;
- **Kp** : L'action proportionnel ;
- **Ti** : L'action intégral ;
- **Kp** : L'action dérivée ;
- **CYCLE** : Permet de définir le fonctionnement cyclique du régulateur par rapport au programme principal ;
- **OUT** : L'action de correction du régulateur PID.

Exemple : Démarré un moteur si le bouton start est pressé et si on n'est pas dans une condition d'alarme.

II.13.3- Texte structure ou ST

Ce langage est un langage textuel de haut niveau de type "informatique" permettant l'écriture structurée (algorithmes) de traitements logiques et numériques. Ce langage est proche d'un langage informatique comme le C et le PASCAL. Ce langage facilite donc la mise en œuvre d'algorithme complexe comportant beaucoup de traitement numérique. Il permet aussi des passerelles vers des blocs extérieurs, ainsi que des traitements particuliers. En contrepartie, il est moins commode pour la mise au point de fonctions booléennes.

Dans le tableau II.1 suivant en essayant de citer quelques opérateurs utilisés dans la programmation des automates industriels avec le langage ST.

Tableau II.1 Quelques opérateurs utilisés dans la programmation avec le langage ST

№	Opération	Symbole
1	Mise entre parenthéses	(Expression)
2	Evaluation de fonction Exemple	Identificateur (liste d'argument) LN(A), MAX(x,y), MIN(a,b), etc.
3	Exponentiation	**
4	Négation	—
5	Complément	NOT
6	Multiplication	*
7	Division	/
8	Modulo	MOD
9	Comparaison	<, >, <=, >=
10	Egalité	=
11	Enégalité	<>
12	AND booléen	&
13	AND booléen	AND
14	OR exclusif booléen	XOR
15	OR exclusif	OR
16	Addition	+
17	sustraction	-

- **Structures des programmes**

Les structures de programmation pour le langage ST, sont les mêmes que celles utilisé dans les langages C et Pascal à l'image des affectations et les tests et les boucles for, while, if, etc. dans ce qui suit quelques exemples de ces structures utilisées dans les programmes ST :

a- Affectation et tests

(* Ceci est un commentaire *)
(* Ceci est une affectation *)
VOYANT :=TRUE;
Compteur := 100;
(* Ceci est un test *)
Température > 50
Position <>0
Fcourse = TRUE

b- Structure IF

IF test THEN
…….;
…….;
ELSE
…….;
…….;
END_IF;

➢ **Example**

IF Capteur3 = TRUE THEN
N: = 3;
END_IF;
IF Compteur > 100 THEN
VOYANT :=TRUE;
ELSE
VOYANT := FALSE;
END_IF;

c- Structure CASE

```
CASE slecteur OF
val1;
 val2;
…….;
val3 ;
…….;
ELSE
…….;
END_CASE;
```

➢ **Exemple** : estimation de la qualité d'une pièce usinée. La pièce doit faire 3 mm d'épaisseur.

```
CASE epaisseur_piece_mm OF
1:
   QUALITE = MAUVAISE;
2 :
   QUALITE = MOYENNE;
3 :
   QUALITE = BONNE;
4 :
   QUALITE = MOYENNE;
5 :
  QUALITE = MAUVAISE;
ELSE :
   QUALITE = TRES_MAUVAISE;
END_CASE;
```

d- Structure REPEAT

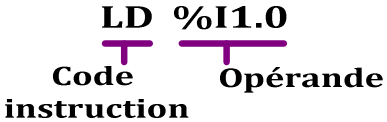
```
  i:=1 ;
REPEAT
  i :=i+2 ;
UNTIL i=200 OR WORDS (i)='KEY'
END_ REPEAT
```

e- Structure FOR

```
B:=19;
FOR i:=1 TO 20 BY  DO
  IF WORD (i) ='KEY' THEN
     B=: I;
     EXIT;
  END_IF
END_FOR
```

II.13.4- Liste d'instructions ou IL

Langage basique des automatismes très proche du langage assembleur, cette programmation s'effectue directement sur le processeur, il représente une liste d'instructions qui permettent de transcrire sous forme d'une liste d'équations logiques : un schéma à contact, un logigramme, équations booléennes, un grafcet. Il réalise aussi des fonctions d'automatisme telles que temporisation, comptage, pas à pas, etc. Chaque instruction est composée d'un CODE INSTRUCTION et d'un OPERANDE.



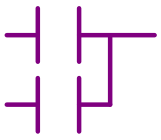
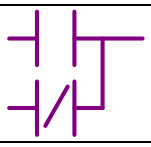
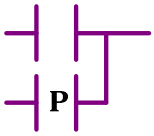
- **Code instruction :** Le code instruction détermine l'opération à exécuter. Il existe 2 types de codes instructions :
  - ✓ **Test**, dans laquelle figurent les conditions nécessaires à une action (ex : LD, AND, OR...) ;
  - ✓ **Action**, qui sanctionne le résultat consécutif à un enchaînement de test. (ex: ST, STN, R, ...).
- **Opérande :** Une instruction agit sur un opérande. Cet opérande peut être :
  - ✓ Une entrée/sortie de l'automate (boutons poussoirs, détecteurs, relais, voyants...) ;
  - ✓ Une fonction d'automatisme (temporisateurs, compteurs...) ;

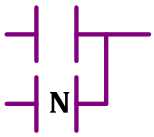
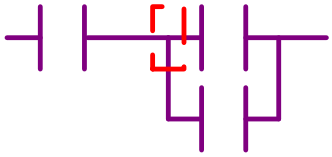
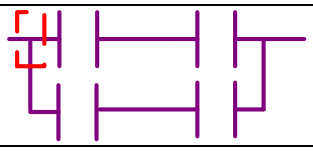
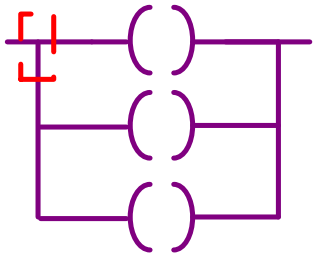
II.13.4.a- Instructions de base

- **Instructions de test**

Les instructions test essentielles utilisées pour la programmation sont illustrées dans le tableau suivant :

Tableau II.2 Instruction de test

Désignation	***Graphisme équivalent***	***Fonctions***
LD		*Le résultat est égal à l'opérande (lire la valeur)*
LDN		*Le résultat est égal à l'inverse de l'opérande*
LDR		*Le résultat booléen passe à 1 à la détection du passage de 0 à 1 de l'opérande (front montant).*
LDF		*Le résultat booléen passe à 1 à la détection du passage de 1 à 0 de l'opérande (front descendant).*
AND		*Le résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente et l'état de l'opérande.*
ANDN		*ET logique entre le résultat précèdent et l'état inverse de l'opérande*
ANDR		*Le résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente, et la détection d'un front montant de l'opérande (1=front montant).*
ANDF		*Le résultat booléen est égal au Et logique entre le résultat booléen de l'instruction précédente, et la détection d'un front descendant de l'opérande (1=front descendant).*
OR		*Le résultat booléen est égal au Ou logique entre le résultat booléen de l'instruction précédente et l'état de l'opérande.*
ORN		*OU logique entre le résultat précèdent et l'état inverse de l'opérande.*
ORR		*Le résultat booléen est égal au Ou logique entre le résultat booléen de l'instruction précédente, et la détection d'un front montant de l'opérande (1=front montant).*

ORF		*Le résultat booléen est égal au OU logique entre le résultat booléen de l'instruction précédente, et la détection d'un front descendant de l'opérande (1=front descendant)*
AND (		*Et logique (8 niveaux de parenthèses)*
OR (		*Ou logique (8 niveaux de parenthèses)*
XOR, XORN, XORR, XORF		*OU exclusif*
MPS MRD MPP		**MPS (Memory PuSh) : a pour effet de stocker le résultat de la dernière instruction de test au sommet de la pile et de décaler les autres valeurs vers le fond de la pile. *MRD (Memory ReaD): lit le sommet de la pile. *MPP (Memory PoP): a pour effet de lire, de déstocker le sommet de la pile et de décaler les autres valeurs vers le sommet de la pile.*

- **Instructions d'action :**

- **ST** : bobine directe. L'objet bit associé prend la valeur du résultat de la zone test.
- **STN** : bobine inversée. L'objet bit associé prend la valeur inversée du résultat de la zonetest.
- **S** : bobine d'enclenchement. L'objet bit associé est mis à 1 lorsque la valeur du résultat de la zone test est à 1.
- **R** : bobine de déclenchement. L'objet bit associé est mis à 0 lorsque la valeur du résultat de la zone test est à 1.

- **Instructions de saut :**

- **JMP** : saut de programme inconditionnel.
- **JMPC** : saut de programme si le résultat de l'instruction test précédente est à 1.
- **JMPCN** : saut de programme si le résultat de l'instruction test précédente est à 0.
- **SRN** : branchement en début de sous-programme.
- **RET** : retour de sous-programme inconditionnel.
- **RETC** : retour de sous-programme si le résultat de l'instruction test précédente est à 1.
- **RETCN** : retour de sous-programme si le résultat de l'instruction test précédente est à 0.

- **Instructions d'arrêt :**

- **END** : fin de programme inconditionnelle.
- **ENDC** : fin de programme si le résultat de l'instruction test précédente est à 1.
- **ENDCN** : fin de programme si le résultat de l'instruction test précédente est à 0.
- **HALT** : Arrêt de l'exécution du programme.

- **Opérations de transfert (SIEMENS) :**

- **MOVB** : transfert d'un octet dans un autre.
- **MOVW** : transfert un mot dans un autre.
- **MOVD** : transfert d'un double mot dans un autre.
- **MOVR** : transfert d'un double mot réel dans un autre.

II.13.4.b- Programmation des blocs fonction

- **Temporisation**: le pilotage est réalisé par des instructions et la sortie peut être transférée dans un bit.

Exemple :

```
LD      % I1.0
IN      % TM1     Le bit d'entrée I1.0 pilote la temporisation TM1
LD      Q         La sortie Q de la temporisation est chargée dans le bit
ST      % Q4.1    de sortie Q4.1
```

- **Compteur/décompteur** : les pilotages sont réalisés par des instructions et la sortie est directement disponible sous forme de bit.

Exemple :

```
LD      % I1.0    Le bit d'entrée I1.0 pilote la fonction remise à zéro R
R       % C8      du compteur C8
LDN     % I1.1    Le bit d'entée inversé I1.1 et le bit de mémoire inversé M0
ANDN    % M0      pilotent la fonction de comptage CU du compteur C8
CU      % C8
LD      % C8.D    Le bit de sortie C8.D est chargé dans le bit de sortie Q2.0
ST      % Q2.0
```

- **Étiquette et commentaire**

➢ L'étiquette permet de repérer une phrase dans une entité de programme (programme principal, sous-programme,...) mais n'est pas obligatoire. Cette étiquette a la syntaxe suivante : %Li avec i compris entre 0 et 999 et se positionne en début d'une phrase.

➢ Le commentaire peut être intégré au début d'une phrase et peut occuper 3 lignes maximum (soit 222 caractères alphanumériques), encadrés de part et d'autre par les caractères **(*** et ***)**. Il facilite l'interprétation de la phrase à laquelle elle est affectée, mais n'est pas obligatoire.

- **Structure d'une phrase**

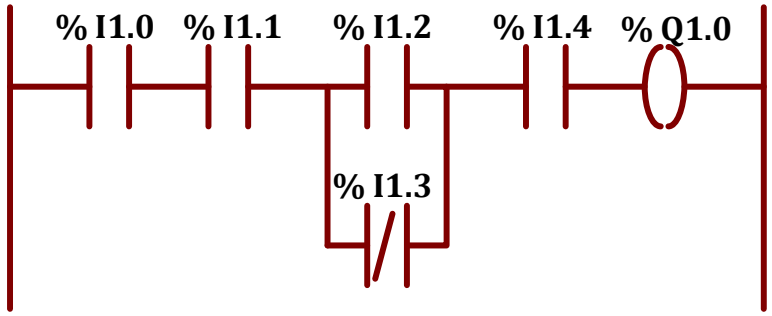
Chaque phrase d'instructions commence par un point d'exclamation généré automatiquement. Comme pour le langage à contacts, elle peut comporter un commentaire et être repérée par une étiquette.

```
! (*Attente de séchage*)          Commentaire entre (* *)
%L2 :                            Etiquette de repérage de la phrase
LD      % I1.0                   Instruction
AND     % M8                     Instruction   } Phrase
ST      % Q2.5                   Instruction
```

- **Utilisation des parenthèses**

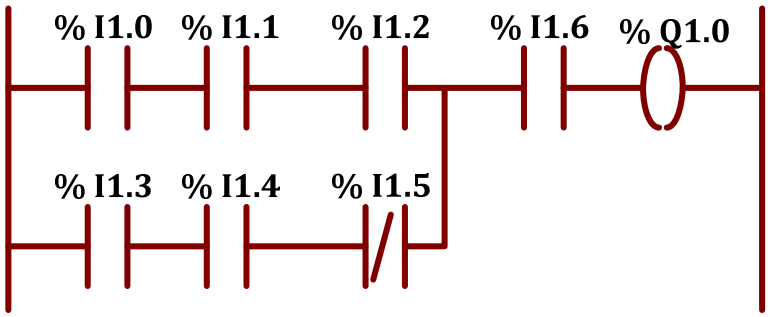
Les instructions AND et OR peuvent utiliser des parenthèses. Ces parenthèses permettent de réaliser des schémas à contacts de façon simple.

➢ **And (**



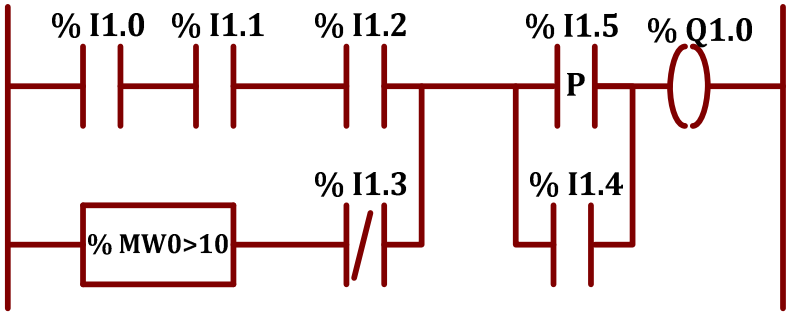

```
LD      % I1.0
AND     % I1.1
AND(    % I1.2
ORN     % I1.3
)
AND     % I1.4
ST      % Q1.0
```

➢ **OR (**



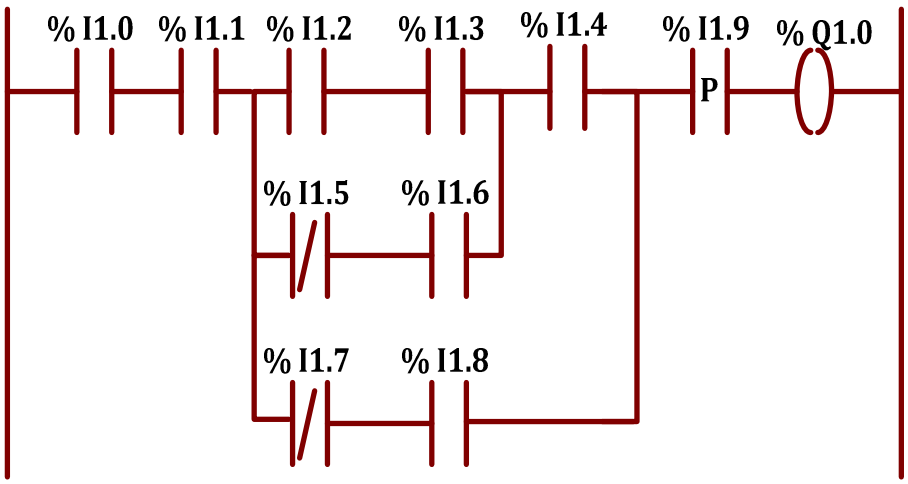

```
LD      % I1.0
AND     % I1.1
AND     % I1.2
OR(     % I1.3
AND     % I1.4
ANDN    % I1.5
   )
AND     % I1.6
ST      % Q1.0
```

- Les parenthèses peuvent associer d'autres codes d'instructions avec AND et OR :
  - N : négation, ex : AND (N ou OR (N ;
  - F : front descendant (**F**alling edge), ex : AND (F ou OR(F ;
  - R : front montant (**R**ising edge), ex : AND(R ou OR(R ;
  - [Comparaison].




```
LD      % I1.0
AND     % I1.1
AND     % I1.2
OR(     [% MW0>10]
ANDN    % I1.3
   )
AND(R   % I1.5
OR      % I1.4
   )
ST      % Q1.0
```

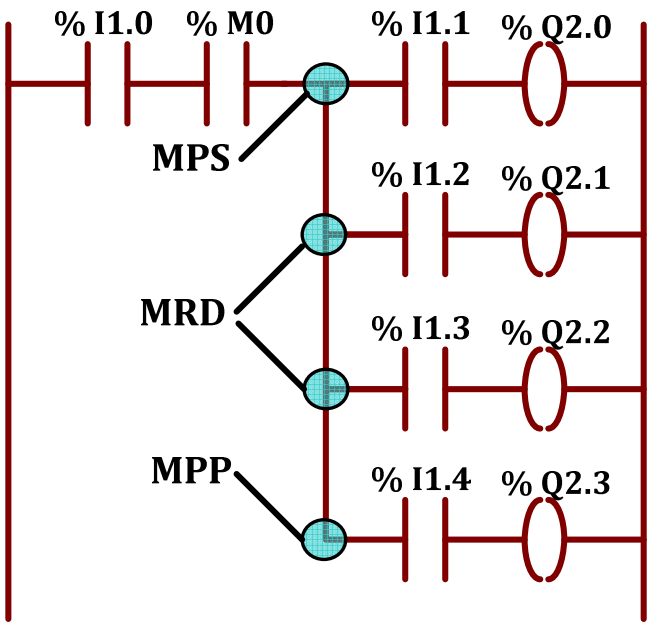
- Imbrication de parenthèses : Il est possible d'imbriquer jusqu'à 8 niveaux de parenthèses.



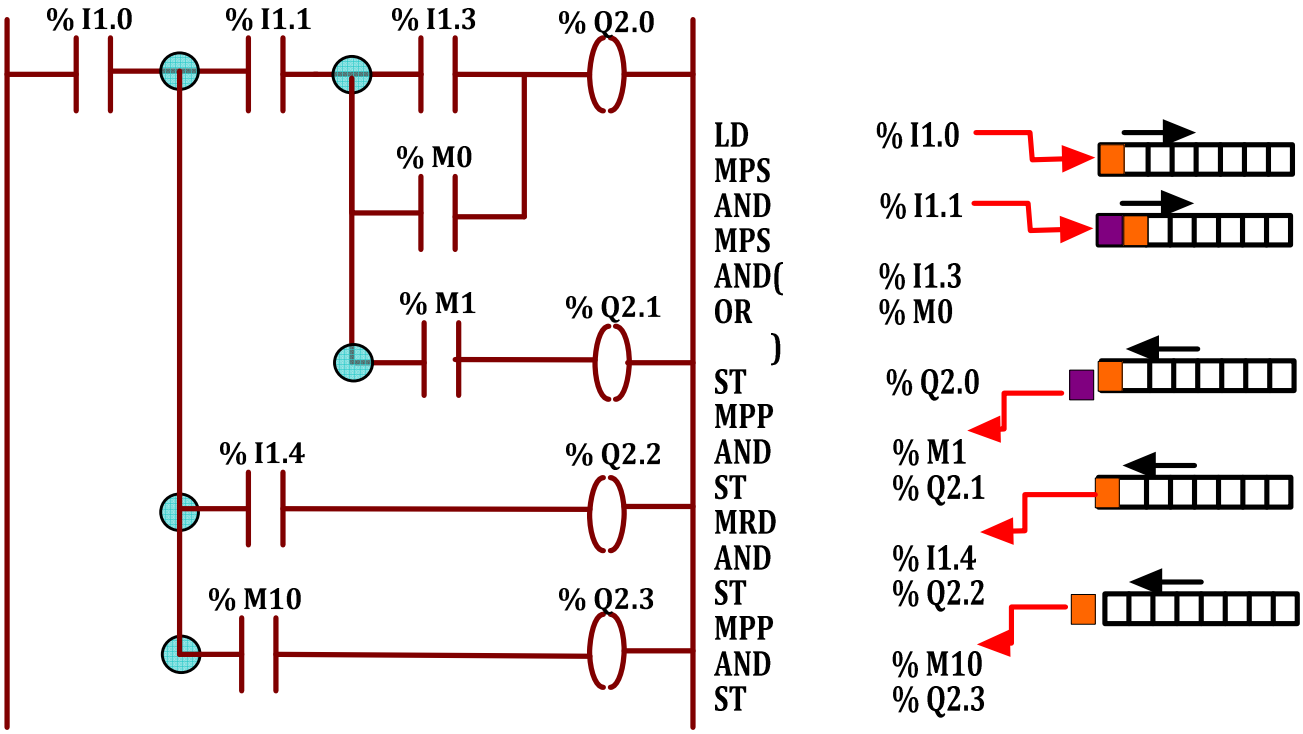

```
LD      % I1.0
AND     % I1.1
AND(    % I1.2
AND     % I1.3
 OR(N   % I1.5
AND     % I1.6
   )
AND     % I1.4
OR(N    % I1.7
AND     % I1.8
   )
   )
ANDR    % I1.9
ST      % Q1.0
```

- **Instructions MPS, MRD, MPP**

Pour savoir l'intérêt de ces trois instructions et la manière de leur utilisation dans la programmation ave le langage IL en prenant les deux exemples suivants (par la traduction de deux programmes de langage contact en langage IL) :




```
LD      % I1.0
AND     % M0
MPS
AND     % I1.1
ST      % Q2.0
MRD
AND     % I1.2
ST      % Q2.1
MRD
AND     % I1.3
ST      % Q2.2
MPP
AND     % I1.4
ST      % Q2.3
```



- **Application:** Démarrer un moteur si le bouton «start» est pressé et si on n'est pas dans un condition d'alarme.

```
VAR
        start       : BOOL   AT   \% IX0.1;
        alarm       : BOOL   AT   \% MX1.5;
        power_on    : BOOL   AT   \% OX3.2;
END_VAR

LD      start
ANDN    alarm
ST      power_on

VAR:        I/M/O     ⇒input/internal/output
            X/B/W/L   ⇒bit, byte, word, double word
```

II.13.5- Schéma à relais ou Ladder (LD)

Un langage graphique, très utilisé en milieu industriel, et qui a été développé pour imiter la logique des relais câblés. Le langage à contacts (LD: Ladder Diagram) est composé de réseaux lus les uns à la suite des autres de gauche à droit par l'automate, chaque réseau contient deux lignes verticales qui présentent les barres d'alimentation. Les contacts (entrées, mémoires, opérations ainsi que des bits de systèmes internes de l'automate) et les blocs fonctionnels (temporisations, compteurs, communication…) et les bobines (sorties et mémoires) sont représentés sous forme des lignes horizontales (réseaux) entre ces lignes verticales.

II.13.5.1- Règles d'évolution d'un réseau de contacts

La lecture d'un réseau se fait réseau connexe par réseau connexe (de haut en bas), puis de gauche à droite à l'intérieur d'un réseau connexe. Un réseau connexe est constitué d'éléments graphiques tous reliés entre eux, mais indépendants des autres éléments graphiques du réseau. Si l'on rencontre une liaison verticale de convergence, on évalue d'abord le sous-réseau qui lui est associé (toujours dans la même logique) avant de continuer l'évaluation du sous réseau qui l'englobe.

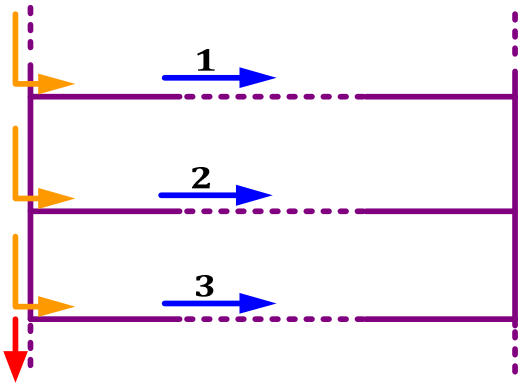
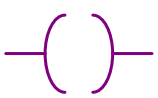
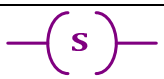
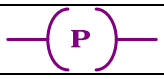


Figure II.17 Evolution d'un réseau Ladder

Dans le tableau II.3 ci-dessous, en trouvant quelques exemples des symboles de programmation en langage Ladder utilisés par le langage **SIMATIC Step 7**.

Tableau II.3 Quelques exemples des symboles de programmation en langage Ladder

Désignation	Symbole	Fonctions
Contact normalement ouvert		Le contact est fermé (passant) si l'état logique du bit qui le pilote est à état 1
Contact normalement fermé		Le contact est fermé (passant) si l'état logique du bit qui le pilote est à état 0
Inverser RLG		Cette opération inverse le bit de résultat logique (RLG)
Contact fermé au front montant		Le contact est fermé (passant) si le bit qui le pilote détecte un passage de l'état logique de 0 à 1
Contact fermé au front descendant		Le contact est fermé (passant) si le bit qui le pilote détecte un passage de l'état logique de 1 à 0
Bobine directe		Cette opération fonctionne comme une bobine dans un schéma à relais. L'objet bit associé prend la valeur du résultat logique de la zone test
Bobine inverse		L'objet bit associé prend la valeur inverse du résultat logique de la zone test
Bobine d'enclenchement		L'objet bit associé est mis à 1 lorsque le résultat de la zone test est à 1 (maintenu à 1 une fois actionné)

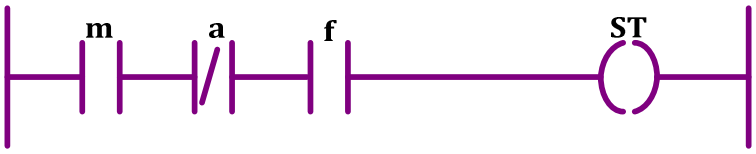
Bobine de déclenchement		L'objet bit associé est mis à 0 lorsque le résultat de la zone test est à 1 (maintenu à 0 une fois actionné)
Bobine front montant		Bobine active (sensible) au front montant de son entrée
Bobine front descendant		Bobine active (sensible) au front descendant de son entrée
Bobine dièse		Proposée en langage Grafcet, utilisée dièse lors de la programmation des réceptivités associées aux transitions provoque le passage à l'étape suivante

Exemples :

a-

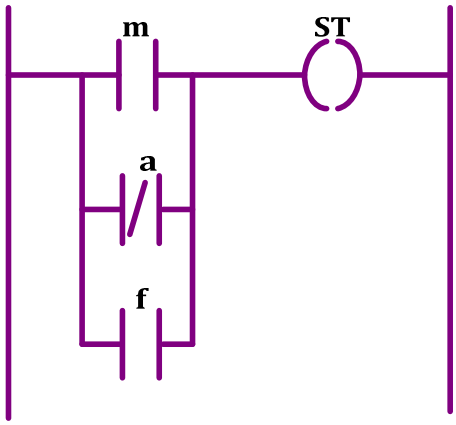
Démarrage de moteur si : marche=1 et alarme=0.

b-



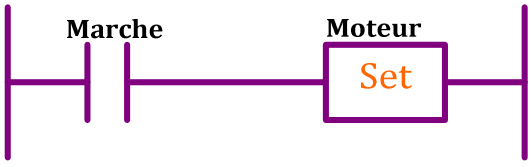
ST= m. not (a) .b

c-

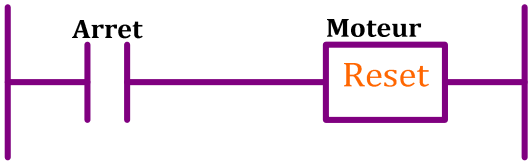


ST= m+ not (a) +b

d- Fonctions mémoire



Traduction de la mise à 1 est maintenue



Traduction de la mise à 0 est maintenue

II.13.5.2- Présentation de logiciel de programmation STEP7

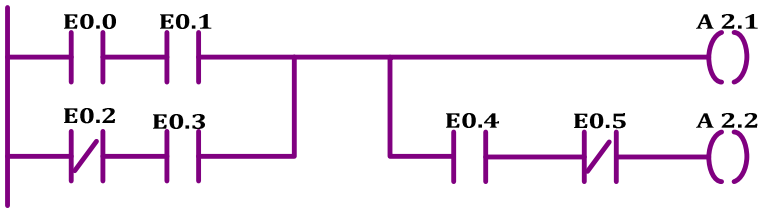
Pour la programmation en langage Ladder, il existe plusieurs logiciels, mais le logiciel le plus utilisé c'est bien que le STEP7 qui fait partie de l'industrie logicielle SIMATIC (SIEMENS). Le logiciel STEP 7 c'est le logiciel de base qui permet de concevoir, configurer, programmer, tester, mettre en service et maintenir les systèmes SIMATIC.

Pour programmer sous STEP7, on ne dispose pas seulement de langage Ladder mais en a aussi deux autres langages (IL et FBD), ces trois langages (LD, IL et FBD) peuvent être combinés dans le même programme. Selon la spécialité du programmeur, il choisira le modèle qui lui convient.

Il s'exécute sous les systèmes d'exploitation de Microsoft à partir de la version Windows 95, à travers une console de programmation ou d'un PC. Il existe plusieurs versions : STEP micro/DOS et STEP micro/ Win pour les applications S7-300 et S7-400.

Dans ce cours en vas essayer de citer quelques fonctions et blocs inespérés des guides fournis par la société SIEMENS. Pour une bonne maîtrise, les étudiants sont obligés de consulter les différents guides disponibles sur le net gratuitement.

Exemple :



- L'état de signal logique de la sortie A 2.1 est 1 si :
  - L'état de signal est 1 aux entrées E 0.0 ET E 0.1 **OU** l'état de signal est 0 à l'entrée E 0.2 et 1 à l'entrée E 0.3.
- L'état de signal logique de la sortie A 2.2 est 1 si :
  - L'état de signal est 1 aux entrées E 0.0 ET E0.1 **ET** l'état de signal est 1 à l'entrée E 0.4 et 0 à l'entrée E 0.5 **OU** l'état de signal est 0 à l'entrée E 0.2 et 1 à l'entrée E 0.3 ET l'état de signal est 1 à l'entrée E 0.4 et 0 à l'entrée E 0.5.

II.13.5.3-Règles fondamentales de saisie des éléments CONT

Un réseau CONT peut être composé par plusieurs éléments dans plusieurs branches. Tous les éléments et les branches doivent être reliés entre eux, la barre d'alimentation gauche n'étant pas considérée comme connexion. La programmation en CONT comme tous les logiciels observe quelques règles nécessaires pour son bon fonctionnement, dans le cas contraire le système assiste l'utilisateur par l'affichage, des messages d'erreur.

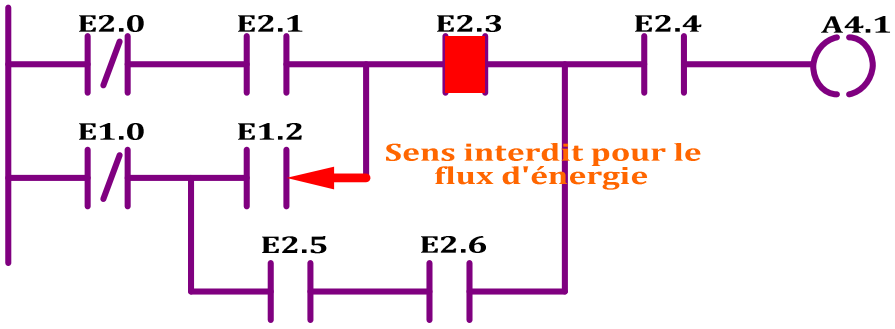
- **Règle 1 : Fermeture d'un réseau CONT**

Tout réseau CONT doit posséder une terminaison sous forme de bobine ou de boîte (de pavé). En ne peut cependant pas utiliser les éléments CONT suivants comme terminaison de réseau :

- Pavés de comparaison
- Bobines pour connecteurs _(#)_
- Bobines pour l'évaluation des fronts montants _(P)_ ou descendants _(N)_

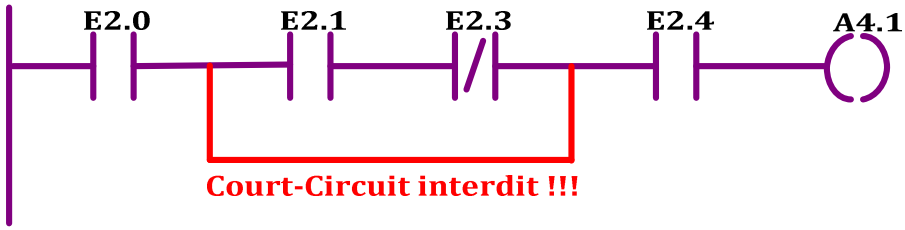
- **Règle 2 : Flux d'énergie**

Il ne faut jamais utiliser des branches qui peuvent provoqueraient une circulation inverse du flux d'énergie. La figure suivante en montre un exemple. Si E 2.3 à l'état de signal "0", E 1.2 entraînerait une circulation de flux d'énergie de la droite vers la gauche. Ceci est interdit.



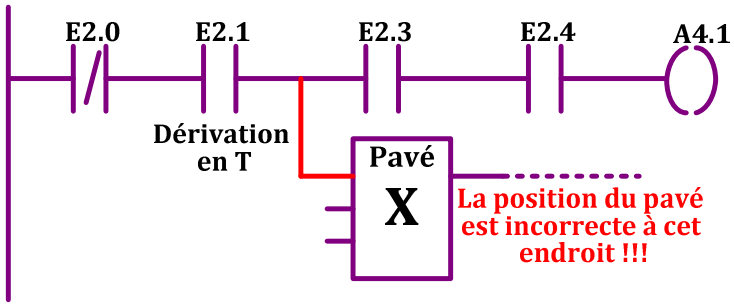
- **Règle 3 : Court-circuit**

En pouvant jamais utiliser un branchement qui provoque un court-circuit. La figure ci-dessous en montre un exemple :



- **Règle 3 : Disposition des pavés**

Le point de départ d'une branche contenant un pavé doit toujours être la barre d'alimentation gauche. La branche qui vient après un pavé peut toutefois contenir des opérations logiques ou d'autres pavés. Il ne faut jamais placez un pavé dans une dérivation en T, à l'exception les pavés de comparaison. La figure suivante nous montre un exemple :



- **Règle 4 : Disposition des bobines**

Les bobines sont automatiquement placées à l'extrémité droite du réseau, où elles constituent la fermeture d'une branche. Exceptions : les bobines pour connecteurs —(#)— et les bobines pour l'évaluation des fronts montants —(P)— ou descendants —(N)— ne peuvent être placées ni complétement à gauche ni complétement à droite de la branche. Elles ne sont pas admises non plus dans les branchements en parallèle.

II.13.5.4- Opérations combinatoires sur bits

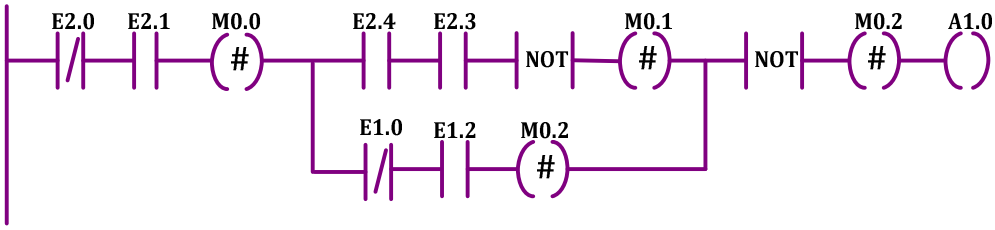
Les opérations combinatoires sur bits utilisent deux chiffres (deux états) : 1 et 0. Ces deux chiffres sont à la base du système de numération binaire et sont appelés chiffres binaires ou bits. Pour les contacts et les bobines, 1 signifie activé ou excité et 0 signifie désactivé ou désexcité. Les opérations de combinaison sur bits évaluent les états de signal 1 et 0 et les combinent selon la logique booléenne. Le résultat de ces combinaisons est égal à 1 ou 0. Il s'agit du résultat logique (RLG). Dans cette partie en vas citer quelques opérations seulement, pour plus d'informations les étudiants sont conseiller de consulter les guides offrissent gratuitement par SIEMENS dans ce sens.

a- Sauvegarder RLG dans RB

L'opérateur « SAVE » permet de sauvegarder le RLG d'un réseau dans le bit RB du mot d'état.

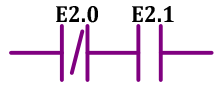
b- Connecteur

L'opération « Connecteur » est un élément d'affectation intermédiaire qui mémorise le RLG. Cet élément sauvegarde la combinaison sur bits de la dernière branche ouverte jusqu'à ce que l'élément d'affectation soit atteint. En série avec d'autres contacts, « Connecteur » fonctionne comme un contact normal.



Le RLG de chaque branche sont comme suit:

M0.0 a le RLG de

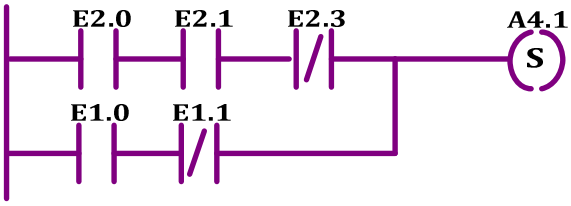


M0.0 a le RLG de

Le RLG de M0.2 est celui de la combinaison sur bits complète

c- Mettre à 1

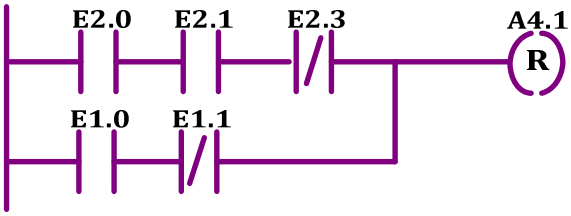
L'opération de la «Mise à 1» ne s'exécute que si le RLG de la combinaison précédente égale à 1.



- L'état de signal de la sortie A 4.1 est mis à 1 si :
  - L'état de signal est 1 aux entrées E 2.0 ET E 2.1 ET le signal est 0 à l'entrée E 2.3 ;
  - OU l'état de signal est 1 à l'entrée E 1.1 ET le signal est 0 à l'entrée E 1.1.
- Si le RLG de la branche est égal à 0, l'état de signal de la sortie A 4.1 reste inchangé.

d- Mettre à 0

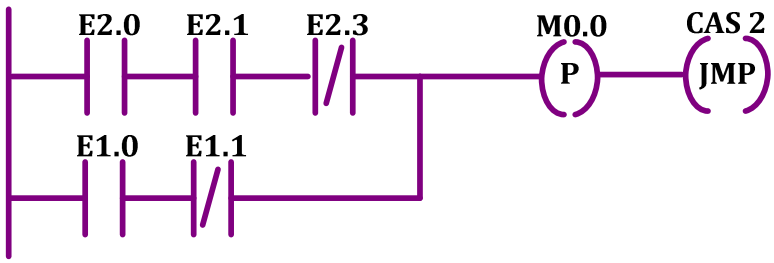
L'opération de la «Mise à 0» ne s'exécute que si le RLG de la combinaison précédente égale à 1.



- L'état de signal de la sortie A 4.1 est mis à 0 si :
  - L'état de signal est 1 aux entrées E 2.0 ET E 2.1 ET le signal est 0 à l'entrée E 2.3 ;
  - OU l'état de signal est 1 à l'entrée E 1.1 ET le signal est 0 à l'entrée E 1.1.
- Si le RLG de la branche est égal à 0, l'état de signal de la sortie A 4.1 reste inchangé.

e- Détecter front montant du RLG

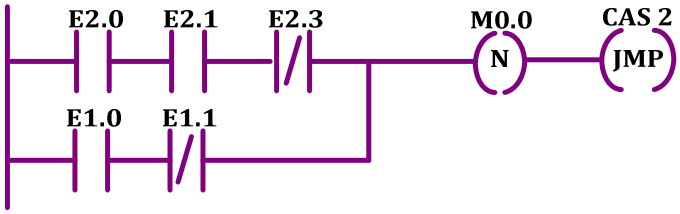
L'opération « Détecter front montant du RLG » détecte le passage de 0 à 1 de l'état de signal du RLG si l'état du signal précédant du RLG mémorisé par le mémento de front est 0 en comparaison avec celui de nouveau RLG qui passe à 1 (impulsion). Les deux conditions précédentes sont les seules conditions pour lesquelles le détecteur de front montant réagisse.



Le mémento de front M0.0 mémorise l'état de signal du RLG précédant, en cas où le RLG avec son nouvel état de signal passe de 0 à 1, le programme effectue un saut au repère CAS2.

f- Détecter front descendant du RLG

L'opération « Détecter front descendant du RLG » détecte le passage de 1 à 0 de l'état de signal du RLG, si l'état du signal précédant du RLG mémorisé par le mémento de front est 1 en comparaison avec l'état du signal l'actuel de RLG qui passe à 0. Les deux conditions précédentes sont les seules conditions pour lesquelles le détecteur de front montant réagisse.



Le mémento de front M0.0 mémorise l'état de signal du RLG précédant, en cas où le RLG avec son nouvel état de signal passe de 1 à 0, le programme effectue un saut au repère CAS2.

g- Initialiser un compteur

Pour initialiser un compteur par l'affectation d'une valeur initiale, l'opérateur « Initialiser compteur SZ» permet d'affecter une valeur initiale au compteur qu'en veux défini, si le signal d'état de RLG présente un front montant (il passe de 0 à 1).

Si l'état de signal de l'entrée E 1.2 passe de 0 à 1 (front montant du RLG), le compteur Z4 est initialisé avec la valeur 20. C# indique que vous entrez une valeur en format DCB. En l'absence de front montant, la valeur de Z5 reste inchangée.

h- Incrémenter un compteur

L'opérateur (ZV) incrémente par 1 la valeur du compteur précisé si le signal d'état du RLG présente un front montant (passe de 0 à 1) et si la valeur du compteur est inférieure à 999. En l'absence de front montant au RLG ou si le compteur est déjà égal à 999, la valeur du compteur reste inchangée.

Si l'état de signal de l'entrée E 1.2 passe de 0 à 1 (front montant du RLG), la valeur du compteur Z4 est incrémentée de 1, à moins qu'elle ne soit déjà égale à 999. En l'absence de front montant, la valeur de Z4 reste inchangée.

i- Décrémenter un compteur

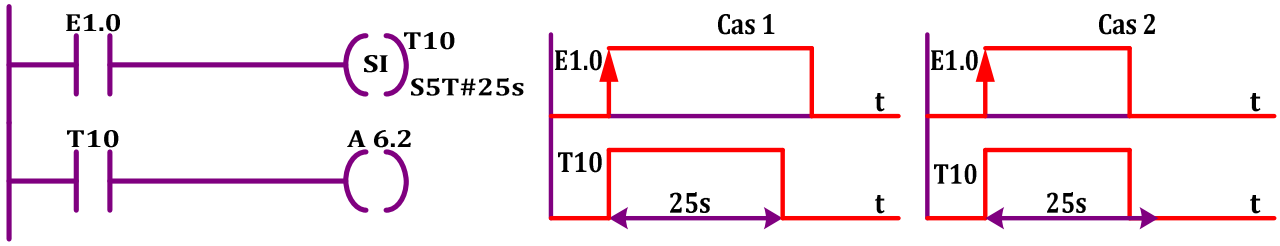
L'opérateur (ZR) décrémente par 1 la valeur du compteur précisé si le signal d'état du RLG présente un front montant (passe de 0 à 1) et si la valeur du compteur est supérieure à 0. En l'absence de front montant au RLG ou si le compteur est déjà égal à 0, la valeur du compteur reste inchangée.

Si l'état de signal de l'entrée E1.2 passe de 0 à 1 (front montant du RLG), la valeur du compteur Z4 est décrémentée de 1, à moins qu'elle ne soit déjà égale à 0. En l'absence de front montant, la valeur de Z4 reste inchangée.

j- Temporisation sous forme d'impulsion

L'opération « Temporisation sous forme d'impulsion » démarre la temporisation indiquée avec une valeur de temps donnée si le RLG présente un front montant (c'est-à-dire si le RLG passe de 0 à 1). La temporisation continue à s'exécuter tant que le RLG est positif.

L'interrogation à 1 de l'état du signal de la temporisation fournit un résultat égal à 1 tant que la temporisation s'exécute. Si le RLG passe de 1 à 0 avant que le temps indiqué ne soit écoulé, la temporisation s'arrête. Les unités de temps sont d (jours), h (heures), m (minutes), s (secondes) et ms (millisecondes).



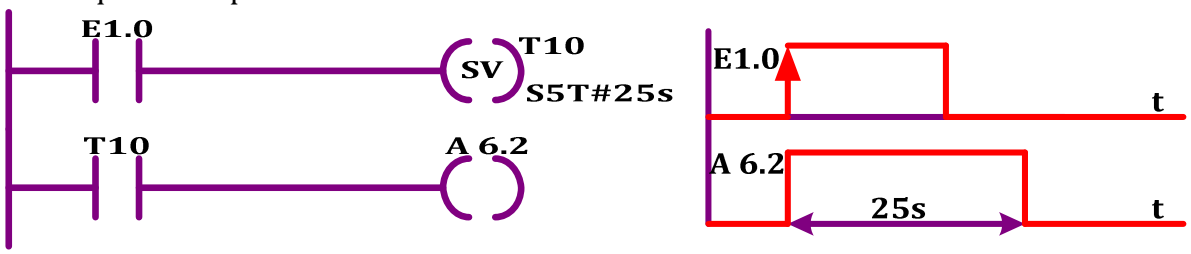
- **Cas1** : Si l'état de signal de l'entrée E1.0 passe de 0 à 1 (front montant du RLG), la temporisation T10 est démarrée. La temporisation continue à s'exécuter avec la valeur de temps précisée de 25 secondes tant que l'état de signal de l'entrée E 1.0 est égal à 1.
- **Cas 2** : Si l'état de signal de l'entrée E 1.0 passe de 1 à 0 avant expiration du temps précisé, la temporisation s'arrête.

L'état de signal à la sortie A 6.2 est 1 tant que la temporisation s'exécute.

Exemples de valeurs de temps :
S5T#12s = 12 secondes
S5T#22m_36s = 22 minutes et 36 secondes

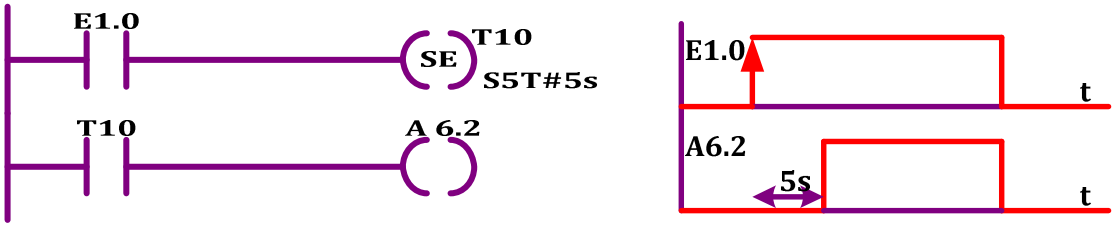
k- Temporisation sous forme d'impulsion prolongée

Si l'état de signal de l'entrée E1.0 passe de 0 à 1 (front montant du RLG), la temporisation T10 est démarrée. La temporisation continue à s'exécuter même en présence d'un front descendant du RLG. Si l'état de signal de l'entrée E1.0 passe de 0 à 1 avant expiration du temps précisé, la temporisation est réenclenchée. L'état de signal à la sortie A4.0 est 1 tant que la temporisation s'exécute.



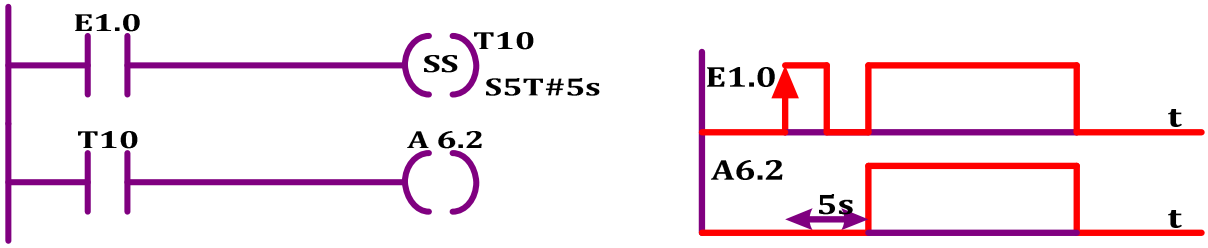
l- Temporisation sous forme de retard à la montée

Si l'état de signal de l'entrée E 1.0 passe de 0 à 1 (front montant du RLG), la temporisation T10 est démarrée. Si le temps expire et que l'état de signal de l'entrée E1.0 est toujours 1, la sortie A6.2 est mise à 1. Si l'état de signal de l'entrée E1.0 passe de 1 à 0, la temporisation est arrêtée et la sortie A6.2 est à 0.



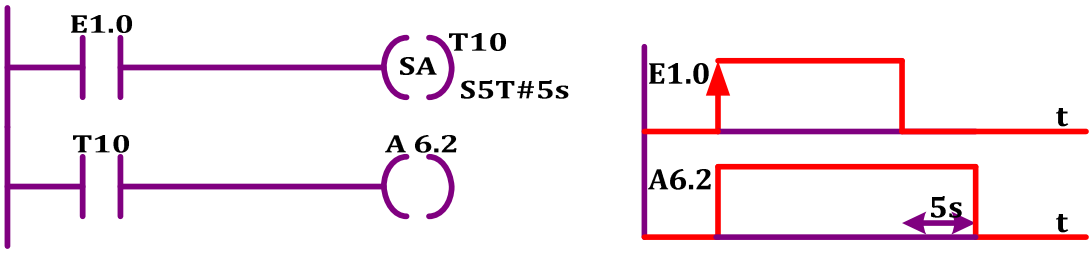
m- Temporisation sous forme de retard à la montée mémorisé

Si l'état de signal de l'entrée E1.0 passe de 0 à 1 (front montant du RLG), la temporisation T10 est démarrée. La temporisation continue à s'exécuter même si l'état de signal passe de 1 à 0 à l'entrée E1.0. Si l'état de signal de l'entrée E1.0 passe de 0 à 1 avant que le temps n'ait expiré, la temporisation est réenclenchée. L'état de signal de la sortie A6.2 est à 1 si le temps a expiré.



n- Temporisation sous forme de retard à la retombée

Si l'état de signal de l'entrée E1.0 passe de 1 à 0, la temporisation T10 est démarrée. Si l'état de signal de l'entrée E1.0 passe de 0 à 1, la temporisation est mise à 0. L'état de signal de la sortie A6.2 est à 1 lorsque l'état de signal de l'entrée E1.0 est 1 ou que la temporisation s'exécute.



o- Adresse d'une temporisation en mémoire et composants d'une temporisation

Lorsqu'une temporisation est démarrée, le contenu de la cellule de temporisation est utilisé comme valeur de temps. Les bits 0 à 11 de la cellule de temporisation contiennent la valeur de temps en format décimal codé binaire (format DCB : chaque groupe de quatre bits contient le code binaire d'une valeur décimale). Les bits 12 et 13 contiennent la base de temps en code binaire. Le tableau II.4 montre les résolutions possibles avec les plages correspondantes.

Tableau II.4 Code binaire et la base de temps avec son résolution plage

Base de temps	Code binaire de la base de temps	Résolution Plage
10 ms	00	10MS à 9S_990MS
100 ms	01	100MS à 1M_39S_900MS
1 s	10	1S à 16M_39S
10 s	11	10S à 2HR_46M_30S

La **valeur de temps maximale** que en pouvant indiquer est égale à 9 990 secondes c-à-dire S5T#2h_46m_30s (2 heures, 46 minutes et 30 secondes)

La figure II.18 suivante montre le contenu de la cellule de temporisation dans laquelle vous avez chargé la valeur de temps 127 et une base de temps de 1 seconde.

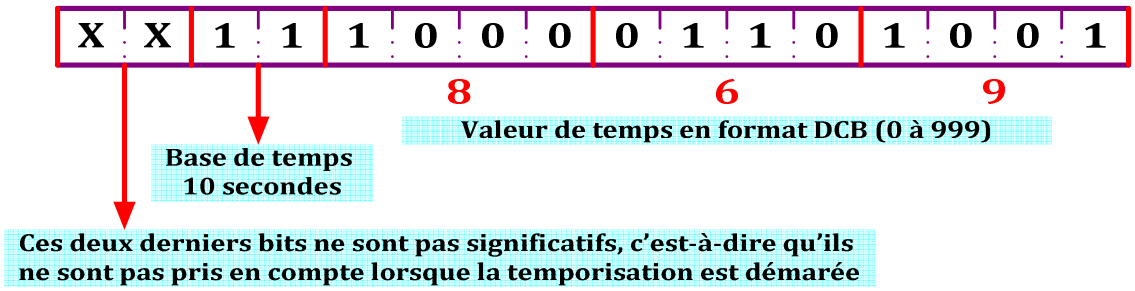


Figure II.18 Contenu de la cellule de temporisation

II.13.5.5- Choix de la temporisation correcte

Dans de nombreuses applications de contrôle à base d'automates programmables, il est nécessaire de contrôler le temps. Les automates programmables ont donc des minuteries en tant que périphériques intégrés. Dans la figure ci-dessous on va montrer les diagrammes d'échelle pour les différents types des minuteries (cinq types de temporisations) les plus utilisés.

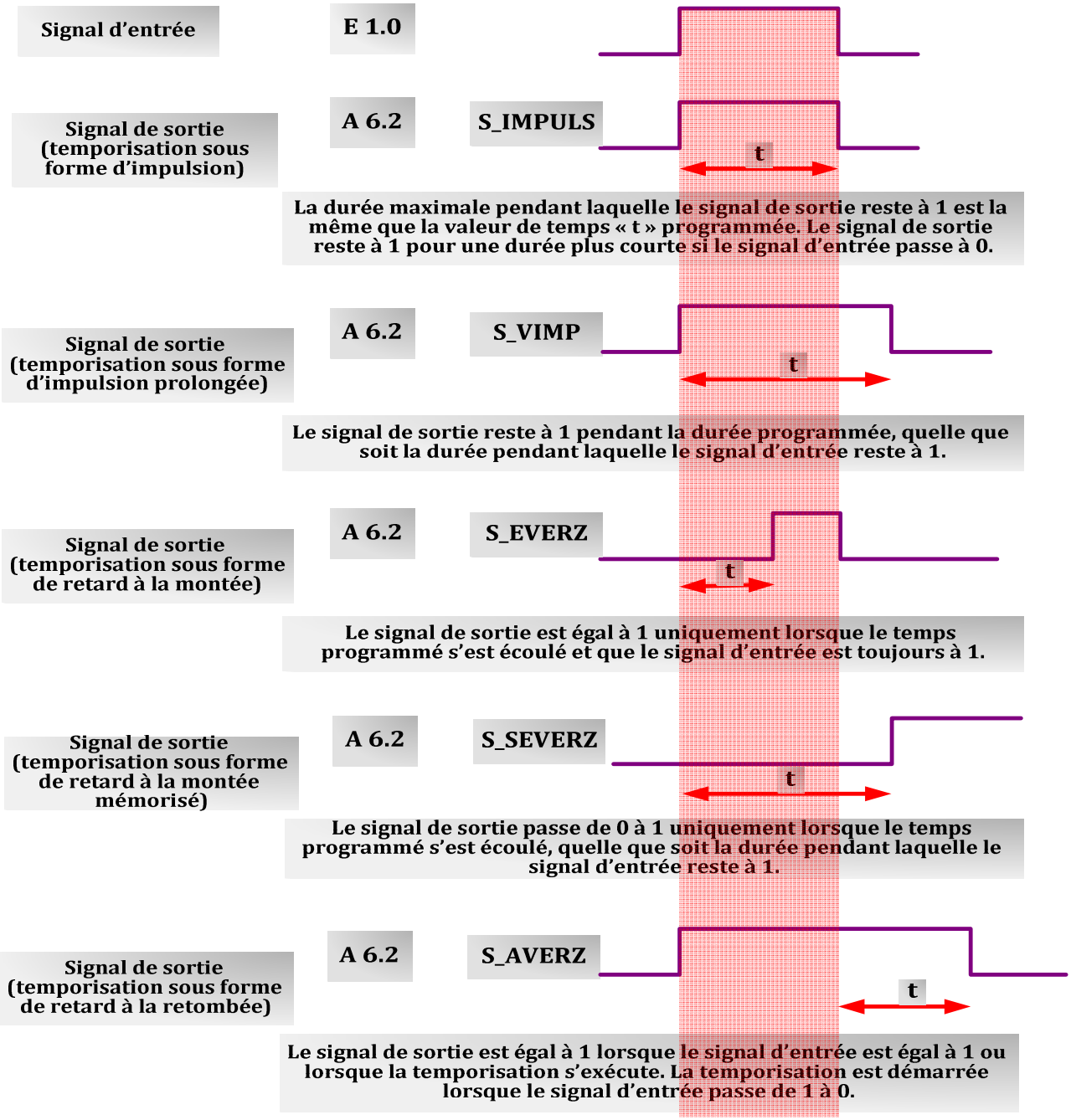
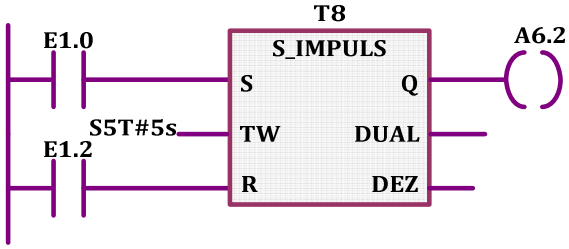


Figure II.19 Différents types de temporisations

a- Temporisation sous forme d'impulsion

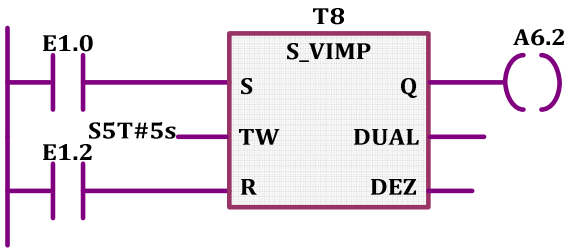
Ce type de temporisation est utilisé pour produire une sortie de 1 de durée fixe prédéterminée de temps quand il y a une entrée à 1. Dans l'exemple ci-dessous la temporisation T 5 est démarrée si l'état de signal passe de 0 à 1 à l'entrée E1.0 (front montant du RLG), et elle va durer (5s), tant que E1.0 est à 1. Si l'état de signal de l'entrée E1.0 passe de 1 à 0 avant que le temps n'ait expiré, la temporisation s'arrête. Si l'état de signal de l'entrée E1.2 passe de 0 à 1 alors que la temporisation s'exécute, la temporisation est remise à zéro. L'état de signal à la sortie A6.2 est 1 tant que la temporisation s'exécute.



S : Entrée de démarrage ;
TW : Valeur de temps prédéfinie (plage : 0 à 9999) ;
R : Entrée de remise à zéro ;
Q : Etat de la temporisation ;
DUAL : Valeur de temps restante (format binaire) ;
DEZ : Valeur de temps restante (format DCB).

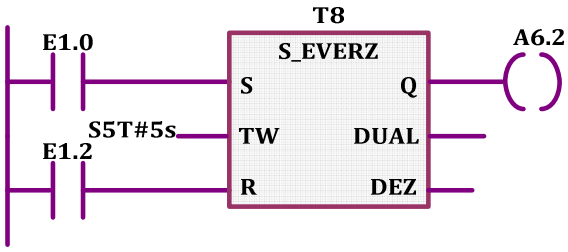
b- Temporisation sous forme d'impulsion prolongée

La temporisation T 8 est démarrée si l'état de signal passe de 0 à 1 à l'entrée E1.0 (front montant du RLG). Le temps de temporisation (5s) indiqué continue à s'écouler même en cas de front descendant à l'entrée S. Si l'état de signal de l'entrée E1.0 passe de 0 à 1 avant que le temps n'ait expiré, la temporisation est redémarrée. Si l'état de signal de l'entrée E1.2 passe de 0 à 1 alors que la temporisation s'exécute, la temporisation est remise à 0. L'état de signal à la sortie A6.2 est 1 tant que la temporisation s'exécute.



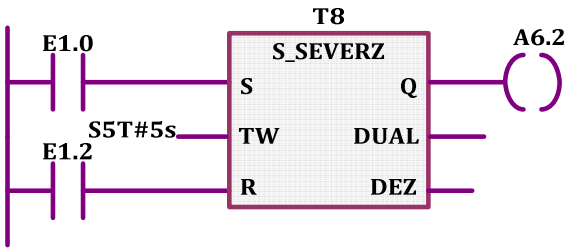
c-Temporisation sous forme de retard à la montée

Cette opération sert à retarder l'activation d'une sortie (le bit du temporisateur) pour un intervalle de temps donné après que l'entrée a été activée. Dans l'exemple ci-après, si on appuyer sur le bouton start dont l'adresse est E1.0 le moteur(A6.2) ne démarre qu'après 5 seconde. C.à.d. la temporisation T 5 est démarrée si l'état de signal passe de 0 à 1 à l'entrée E1.0 (front montant du RLG). Si la temporisation (5s) indiquée expire et que l'état de signal à l'entrée E1.0 égale toujours 1, l'état de signal à la sortie A 6.2 est 1. Si l'état de signal de l'entrée E1.0 passe de 1 à 0, la temporisation s'arrête et A 46.2 est à 0. Si l'état de signal de l'entrée E1.0 passe de 0 à 1 alors que la temporisation s'exécute, la temporisation est redémarrée.



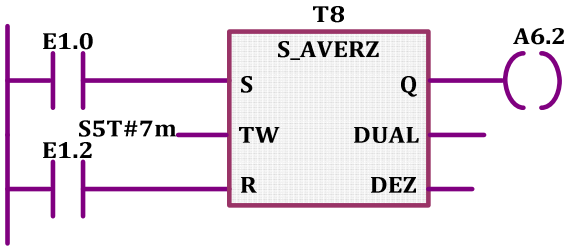
d- Temporisation sous forme de retard à la montée mémorisé

L'opération « Temporisation sous forme de retard à la montée mémorisé » démarre la temporisation précisée en cas de front montant à l'entrée de démarrage S. La valeur de temps indiquée à l'entrée TW continue à s'écouler même si l'état de signal à l'entrée S passe à 0 avant que la temporisation n'ait expiré. L'interrogation à 1 de l'état de signal à la sortie Q donne 1 comme résultat lorsque le temps a expiré, quel que soit l'état de signal à l'entrée S, et lorsque l'entrée de remise à zéro (R) reste à 0. Si l'état de signal à l'entrée S passe de 0 à 1 alors que la temporisation s'exécute, cette dernière est redémarrée avec la valeur de temps indiquée. En cas de passage de 0 à 1 à l'entrée de remise à zéro (R), la temporisation est remise à zéro quel que soit le RLG à l'entrée S. La temporisation T8 est démarrée si l'état de signal passe de 0 à 1 à l'entrée E1.0. La temporisation continue à s'exécuter même si l'état de signal de l'entrée E1.0 passe de 1 à 0. Si l'état de signal de l'entrée E1.0 passe de 0 à 1 avant que le temps n'ait expiré, la temporisation est redémarrée. L'état de signal à la sortie A6.2 est 1 si le temps a expiré et que E1.2 reste à 0.



e- Temporisation sous forme de retard à la retombée

La temporisation sous forme de retard à la retombée sert à retarder la désactivation d'une sortie pour un intervalle de temps donné après que l'entrée a été désactivée. L'opération « Temporisation sous forme de retard à la retombée » démarre la temporisation précisée en cas de front descendant à l'entrée de démarrage S. L'interrogation à 1 de l'état de signal à la sortie Q donne 1 comme résultat lorsque l'état de signal à l'entrée S est 1 ou lorsque la temporisation s'exécute. La temporisation est remise à zéro lorsque l'état de signal à l'entrée S passe de 0 à 1 alors que la temporisation s'exécute. La temporisation n'est redémarrée que lorsque l'état de signal à l'entrée S repasse de 1 à 0. En cas de passage de 0 à 1 à l'entrée de remise à zéro (R) pendant que la temporisation s'exécute, cette dernière est remise à zéro. Dans l'exemple ci-après, on veut assurer le refroidissement d'un moteur pendant son état de marche, et même après son arrêt, dans ce cas-là, le processus de refroidissement va continuer à marcher après l'arrêt du moteur pendant 7 minutes. Dans le programme de fonctionnement, la temporisation T8 est démarrée si l'état de signal passe de 1 à 0 à l'entrée E1.0. L'état de signal à la sortie A6.2 est 1 lorsque l'état de signal de l'entrée E1.0 est 1 ou que la temporisation s'exécute. Si l'état de signal de l'entrée E1.2 passe de 0 à 1 alors que la temporisation s'exécute, la temporisation est remise à zéro.



II.13.5.6- Adresse d'un compteur en mémoire et composants d'un compteur

Les compteurs sont fournis en tant qu'éléments intégrés dans les automates et permettent de compter le nombre d'occurrences de signaux d'entrées. Les bits 0 à 11 du compteur contiennent la valeur de comptage en format DCB. La valeur de comptage est contenue dans les bits 0 à 9 du mot de comptage en format binaire. La plage de la valeur de comptage est comprise entre 0 et 999. Généralement dans les API on trouve deux types de fonctions de comptage, le compteur et le décompteur. Ces deux fonctions peuvent être séparées ou réunies dans un seul bloc selon les marques.

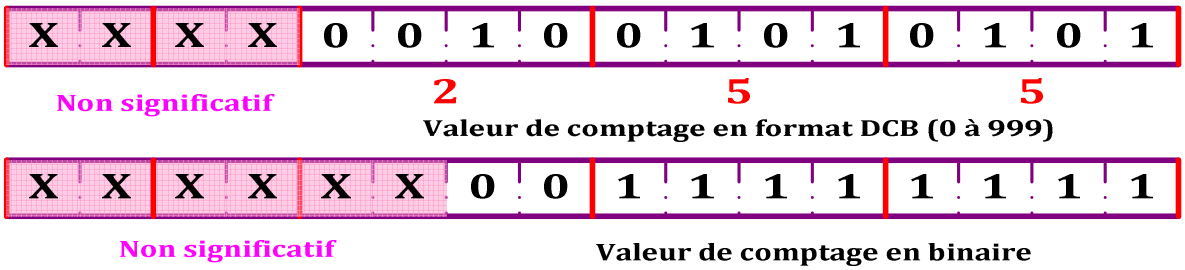
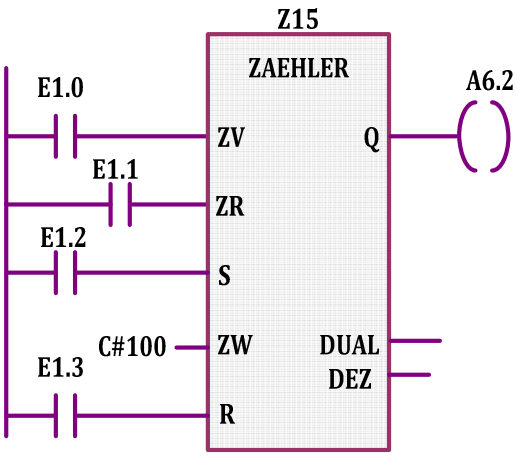


Figure II.20 Adressage d'un compteur

a- Compteur incrémental/décrémental

Le compteur incrémental/décrémental incrémente en partant de la Valeur d'une unité en cours à chaque front montant de l'entrée d'incrémentation ZV (si la valeur du compteur est inférieure à 999) et décrémente d'une unité à chaque front montant de l'entrée de décrémentation ZR (si la valeur du compteur est supérieure à 0). En cas de front montant aux deux entrées de comptage, les deux fonctions sont exécutées et le compteur reste inchangé.

Un front montant à l'entrée S initialise le compteur à la valeur figurant dans l'entrée ZW. En cas de front montant à l'entrée R, le compteur est remis à 0 et la valeur d'initialisation est mise à 0. L'interrogation à 1 de l'état de signal de la sortie Q donne 1 comme résultat lorsque le compteur est supérieur à 0 ; cette interrogation donne 0 comme résultat lorsque le compteur est égal à 0.

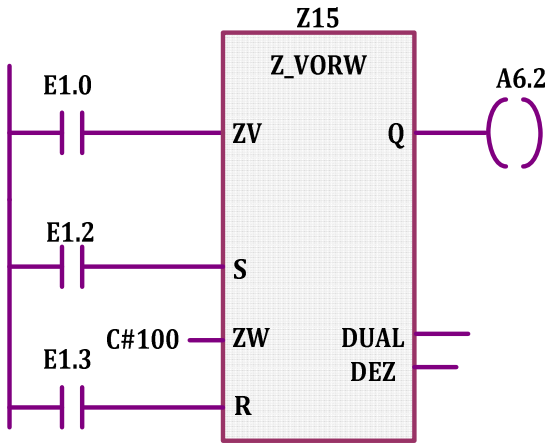


Le compteur Z15 est initialisé à la valeur 100 en format DCB si l'état de signal passe de 0 à 1 à l'entrée E1.2. Si l'état de signal de l'entrée E1.0 passe de 0 à 1, la valeur du compteur Z15 est incrémentée d'un à moins qu'elle ne soit déjà égale à 999. Si l'état de signal de l'entrée E1.1 passe de 0 à 1, la valeur du compteur Z15 est décrémentée d'un à moins qu'elle ne soit déjà égale à 0. Si l'état de signal de l'entrée E1.3 passe de 0 à 1, la valeur du compteur Z15 est mise à 0. L'état de signal de la sortie A6.2 est 1 tant que Z15 est différent de zéro.

b- Compteur incrémental

Le compteur incrémental incrémente d'une unité en partant de la valeur en cours à chaque front montant de l'entrée d'incrémentation ZV et que la valeur du compteur soit inférieure à 999. Un front montant à l'entrée S initialise le compteur à la valeur figurant dans l'entrée ZW. En cas de front montant à l'entrée R, le compteur est remis à 0 et la valeur d'initialisation est mise à 0. L'interrogation à 1 de l'état de signal de la sortie Q donne 1 comme résultat lorsque le compteur est supérieur à 0 ; cette interrogation donne 0 comme résultat lorsque le compteur est égal à 0.

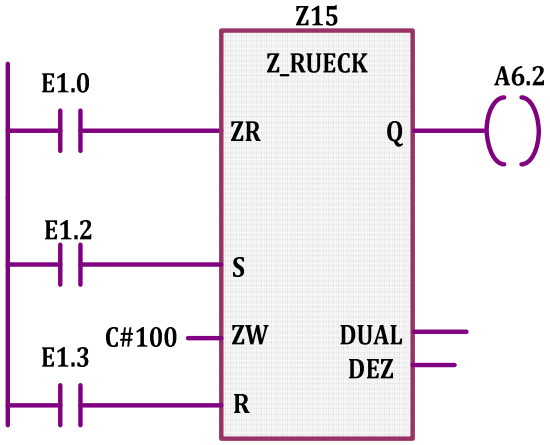
Le compteur Z15 est initialisé à la valeur 100 en format DCB si l'état de signal passe de 0 à 1 à l'entrée E1.2. Si l'état de signal en E1.0 passe de 0 à 1, la valeur du compteur Z15 est incrémentée d'un à moins qu'elle ne soit déjà égale à 999. Si l'état de signal en E1.3 passe de 0 à 1, la valeur du compteur Z15 est mise à 0. L'état de signal de la sortie A6.2 est 1 si Z15 est différent de zéro.



Le compteur Z15 est initialisé à la valeur 100 en format DCB si l'état de signal passe de 0 à 1 à l'entrée E1.2. Si l'état de signal en E1.0 passe de 0 à 1, la valeur du compteur Z15 est incrémentée d'un à moins qu'elle ne soit déjà égale à 999. Si l'état de signal en E1.3 passe de 0 à 1, la valeur du compteur Z15 est mise à 0. L'état de signal de la sortie A6.2 est 1 si Z15 est différent de zéro. Ce programme de comptage peut être utilisé pour compter le nombre des pièces produites par une chaine de production.

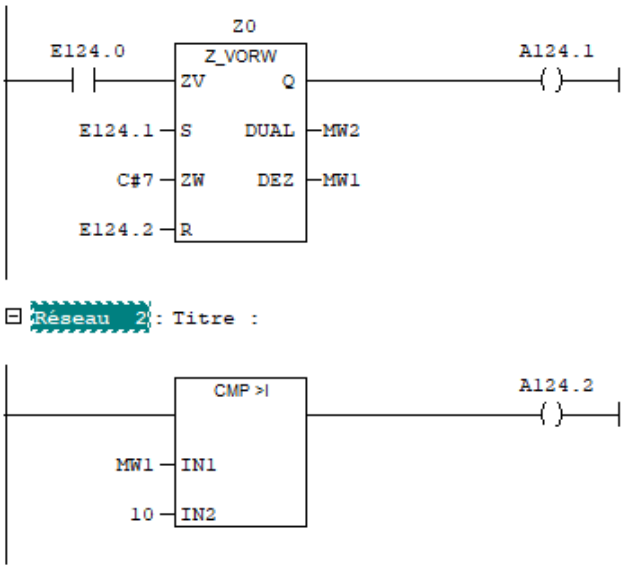
c- Compteur décrémental

Dans certaines applications on a besoin de répéter une séquence plusieurs fois, par la suite il continu la séquence, dans ce cas-là on exploite le fonctionnement du décompteur Z_RUECK. Le compteur décrémental décrémente d'une unité en partant de la valeur en cours à chaque front montant de l'entrée de décrémentation ZR si la valeur du compteur est supérieure à 0, un front montant à l'entrée S initialise le compteur à la valeur figurant dans l'entrée ZW. En cas de front montant à l'entrée R, le compteur est remis à 0, et la valeur d'initialisation est mise à 0. L'interrogation à 1 de l'état de signal de la sortie Q donne 1 comme résultat lorsque le compteur est supérieur à 0 ; cette interrogation donne 0 comme résultat lorsque le compteur est égal à 0.



L'exemple ci-contre montre un tel exemple où la séquence est répétée 100 fois, le compteur Z15 est initialisé à la valeur 100 en format DCB si l'état de signal passe de 0 à 1 à l'entrée E1.2. Si l'état de signal de l'entrée E1.0 passe de 0 à 1, la valeur du compteur Z15 est décrémentée d'un à moins qu'elle ne soit déjà égale à 0. Si l'état de signal de l'entrée E1.3 passe de 0 à 1, la valeur du compteur Z15 est mise à zéro.

NB. Généralement les opérations des fonctions du compteur sont associées à la fonction de comparaison pour lier le compteur au reste du programme comme le montre l'exemple suivant :



Dans l'exemple ci-contre le système termine le remplissage d'une caisse des bouteilles pleines d'une matière conditionnée. Le nombre initial (ZW) de bouteilles dans la caisse est 7, une fois le nombre atteint 10 et après la comparaison entre cette valeur mémorisée dans MW1 et la limite mémorisé dans IN1 le compteur donne l'ordre pour emballer ces 10 bouteilles (A124.2)

La bibliothèque de langage CONT pour SIMATIC S7-300/400 contient plusieurs blocs et opérations qu'en peuvent citer dans ce chapitre comme :

- **Opérations combinatoires sur bits :** Mettre à 1 et à 0, bascule mise à 1et mise à 0, détecteurs des fronts montrant et descendant du RLG , etc.
- **Opérations arithmétiques sur nombres entiers :** Addition, soustraction, multiplication et division des entiers de 16 bits et 32 bits, reste de division (32 bits), etc.
- **Opérations arithmétiques sur nombres réels :** Addition, soustraction, multiplication et division des réels, valeur absolue d'un nombre réel, logarithme naturel d'un nombre réel, fonctions trigonométriques d'angles sous forme de nombres réels, etc.
- **Opérations de comparaison :** Comparaison des entiers de 16 bits et 32 bits, et comparer nombres réels ;
- **Opérations de transfert et de conversion :** Affecter valeur, Convertir de nombre en format DCB en entier de 16 bits ou 32 bits et vice-versa, complément à un d'entier de 16 bits ou 32 bits, convertir nombre réel en entier supérieur le plus proche, etc.
- **Opérations combinatoires sur mots:** ET mot ou double mot, OU mot, OU exclusif mot, etc.
- **Opérations de décalage et de rotation ;**
- **Opérations sur blocs de données, opérations de saut, etc.**

Les étudiants sont obligés de consulter les différents guides du SIMATIC S7-300/400 disponibles sur le net gratuitement !!!

Annexe II

Question de cours

1- Comment allez-vous procéder pour écrire la syntaxe d' :
   - Une entrée d'un automate ;
   - Une sortie d'un automate ;
   - Un bit interne ;
   - Un bloc de fonction temporisateur;
   - Un bloc fonction compteur.

2- Quel est l'état du bouton RUN, situé sur la face avant de l'automate, quand ce dernier est en mode RUN (Exécution du programme).

3- C'est quoi le rôle d'un rack dans un API ?

4- Quel est le rôle de l'IHM ?

5- La CPU comporte des LED de signalisation, donner la signification des LED suivant: SF (rouge), BATF (rouge), DC5V (verte) et FRCE (jaune) ;

6- Pour la mémoire de programme de l'automate quelle est la technologie fréquemment rencontrée ?

7- En mode RUN-P : est-ce-que le superviseur peut introduire une modification dans le programme de L'API ? pourquoi ?

8- Les entrées TOR : Elles permettent de raccorder à l'automate les différents capteurs logiques, citer trois exemples ;

9- Les sorties TOR : Elles permettent de raccorder à l'automate les différents pré-actionneurs à commande simple, citer trois exemples ;

10- Proposer un programme sur le logiciel Step 7 langage Ladder, pour un démarrage direct d'un moteur dans deux sens.

11- On peut considérer qu'un API comme un :
   - Microprocesseur.
   - Microcontrôleur.
   - Micro-ordinateur doté d'une interface d'E/S.
   - Carte de développement à base d'un microprocesseur.

12- Le rôle d'interface de communication dans le module CPU est de :
   - Communiquer avec les cartes d'E/S.
   - Communiquer avec la console de programmation.
   - Communiquer avec d'autres systèmes.
   - Communiquer avec les cartes d'E/S distantes.

13- Le fonctionnement cyclique d'un API est de :
   - Lire les entrées, lire les sorties et exécution du programme.
   - Lire les entrées TOR, les entrées analogiques et la mise à jours des sorties.
   - Lire les entrées, exécuter le programme et la mise à jours des sorties.
   - Lire les entrées, vérification et exécution du programme et la mise à jours des sorties.

14- La distance, entre un API local et ses interfaces d'E/S distants :
   - Est illimitée.
   - Dépend du protocole de communication.
   - Dépend du protocole de communication et la technologie des câbles.
   - Dépend de la technologie des câbles.

15- L'échange de données, entre un API local et ses interfaces d'E/S distants, doit être effectué:
   - Au début de chaque cycle.
   - A la fin de chaque cycle.

- Au période d'exécution du programme de chaque cycle.
- Au période de mise à jour des sorties de chaque cycle.

16- Le rôle du module d'alimentation d'un API est :
- D'alimenter le module CPU.
- D'alimenter les modules d'interfaces d'E/S.
- D'alimenter les modules d'interfaces d'E/S et le CPU.
- D'alimenter les modules d'interfaces d'E/S local et distants.

17- Le signal TOR signifie qu'il y a :
- Un nombre infini d'états ;
- Un nombre fini d'états ;
- Deux états ;
- Deux états numériques.

18- L'interrupteur à molette est considéré comme une :
- Sortie TOR ;
- Sortie analogique ;
- Entrée TOR ;
- Sortie analogique.

19- Pour quoi on met un circuit de détection de seuil dans les cartes d'entrée :
- Pour la connexion entre le capteur et la carte ;
- Pour l'isolation ;
- Pour détecter le niveau de tension nécessaire à l'entrée ;
- Pour détecter le niveau de courant nécessaire à l'entrée

20- La différence entre une connexion source et réceptrice d'un actionneur est dans le :
- Type de signal DC ou AC ;
- Type de capteur;
- Type de la carte ;
- Type d'automate.

21- L'avantage d'une carte d'entrée de type BCD réside dans :
- La rapidité ;
- Le type de connexion électrique ;
- Le parallélisme ;
- La liaison série.

22- Un signal analogique il n'a :
- Qu'un seul état ;
- Que deux états ;
- Que trois états ;
- Qu'une infinité des états.

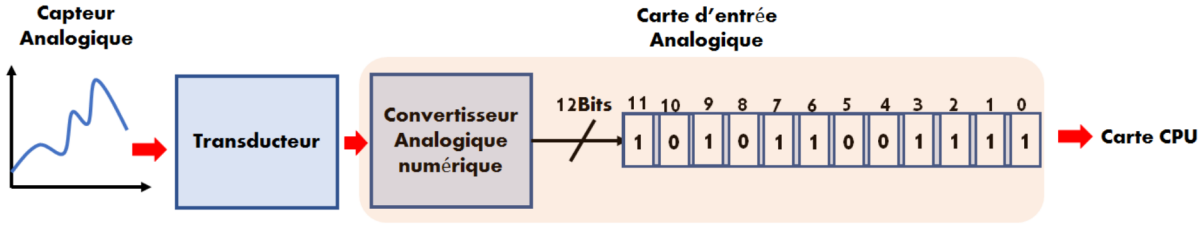
23- Généralement la plupart des APIs utilisent la communication :
- Série ;
- Parallèle ;
- Mixte ;
- Bus de données.

24- Le rôle d'un transducteur est :
- D'amplifier le signal ;
- De normaliser le signal ;
- De convertir le signal ;
- De normaliser et convertir le signal.

25- Le rôle d'un transmetteur est :

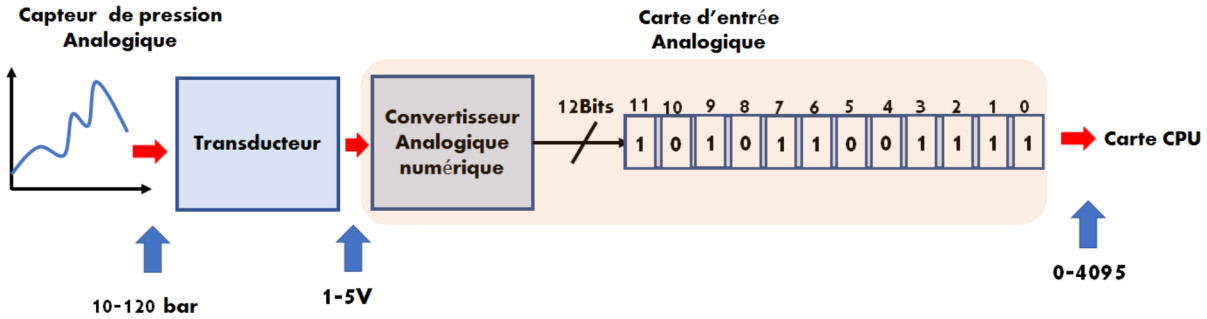
- D'amplifier le signal ;
- De normaliser le signal ;
- De convertir le signal ;
- De normaliser et convertir le signal.

26- Quel sera le contenu du registre de donnée après l'exécution de l'instruction multibit illustrée à la figure suivante :



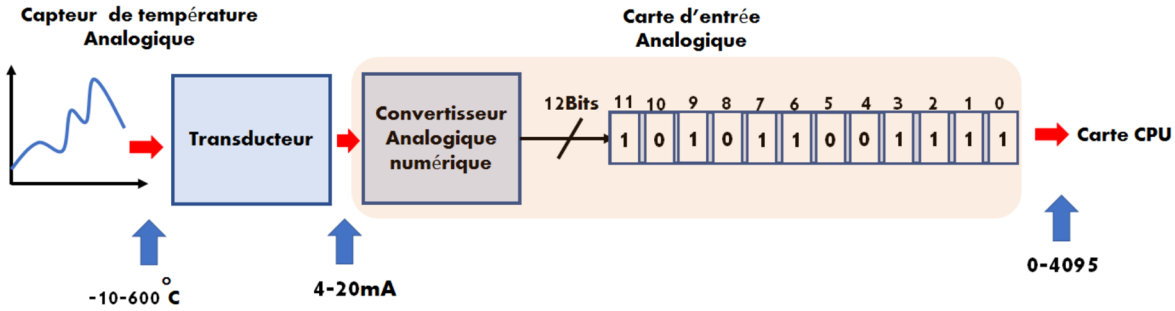
- Notez que la valeur numérisée correspondant à la transformation analogique est codée sur 12 bits.

27- Une carte d'entrée, connectée à un capteur de pression, a une résolution de 12 Bits. Lorsque le capteur de pression reçoit un signal valide du processus (10 à 120 bar), il fournit, via un transmetteur, un signal de +1 à +5 V(DC) compatible avec le module d'entrée analogique.



- Trouvez le changement de tension équivalent pour chaque changement de pression, le changement de tension par un bar, en supposant que la carte d'entrée transforme les données en un nombre linéaire de 0 à 4095.
- Recherchez les mêmes valeurs pour une carte d'entrée avec une résolution de 10 bits.

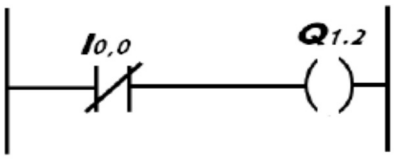
28- Une carte d'entrée, connectée à un capteur de température, a une résolution de 12 Bits (voir Figure 3.29). Lorsque le capteur de pression reçoit un signal valide du processus (-10 à 600_C), il fournit, via un transmetteur, un signal de 4-20mA compatible avec la carte d'entrée analogique.



- Trouvez le changement de tension équivalent pour chaque changement de pression, le changement de tension par un bar, en supposant que le module d'entrée transforme les données en un nombre linéaire de 0 à 4095.
- Recherchez les mêmes valeurs pour une carte d'entrée avec une résolution de 10 bits.

29- Constatez si chacune de ces instructions est Vrai (V) ou fausse (F). La figure suivante montre un diagramme Ladder pour lequel :

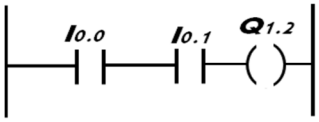
- (i)Le contact d'entrée est normalement ouvert.
- (ii)Il y a une sortie lorsqu'il y a une entrée.

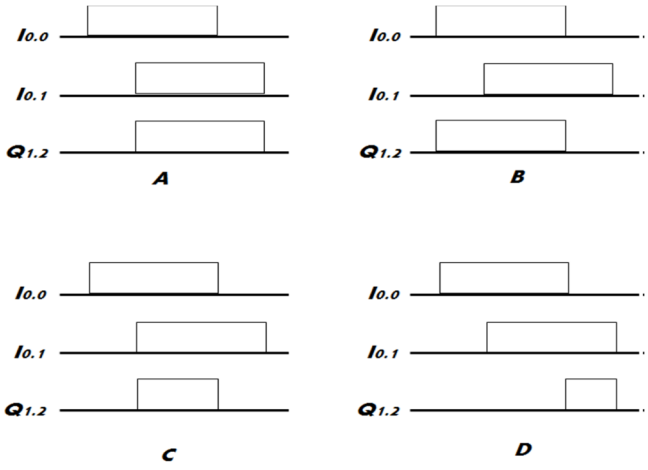


30- Pour chaque cas, tracer le langage Ladder correspondant :

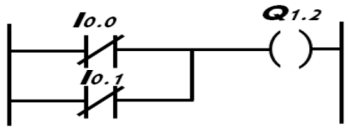
- Deux interrupteurs sont normalement ouverts et les deux doivent être fermés pour qu'un moteur fonctionne.
- L'un ou l'autre des deux interrupteurs normalement ouverts doit être fermé pour qu'une bobine soit alimentée et actionné un actionneur.
- Un moteur est allumé en appuyant sur un bouton-poussoir de démarrage et le moteur reste allumé jusqu'à ce qu'un autre bouton-poussoir d'arrêt soit actionné.
- Un signal lumineux doit être allumé s'il y a une entrée du capteur A ou du capteur B.
- Un voyant s'allume s'il n'y a pas de signal sur un capteur.
- Une électrovanne doit être activée si le capteur A est actionné.
- Pour des raisons de sécurité, les machines sont souvent établies pour s'assurer qu'une machine ne peut être exploitée que par l'opérateur en appuyant simultanément sur deux interrupteurs, l'un par la main droite et l'autre par la main gauche. Il s'agit de veiller à ce que les deux mains de l'opérateur se trouvent sur les interrupteurs et ne peut pas être dans la machine quand il fonctionne. Donner en langage Ladder une telle exigence.

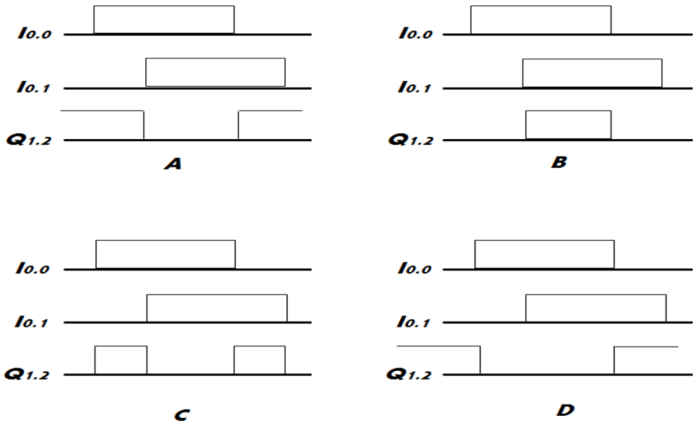
31- La figure suivante montre un programme Ladder, le lequel des diagrammes montrant les entrées et les signaux de sortie se produirait avec ce programme ?



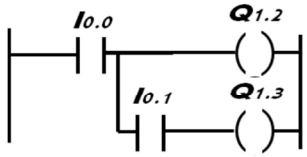


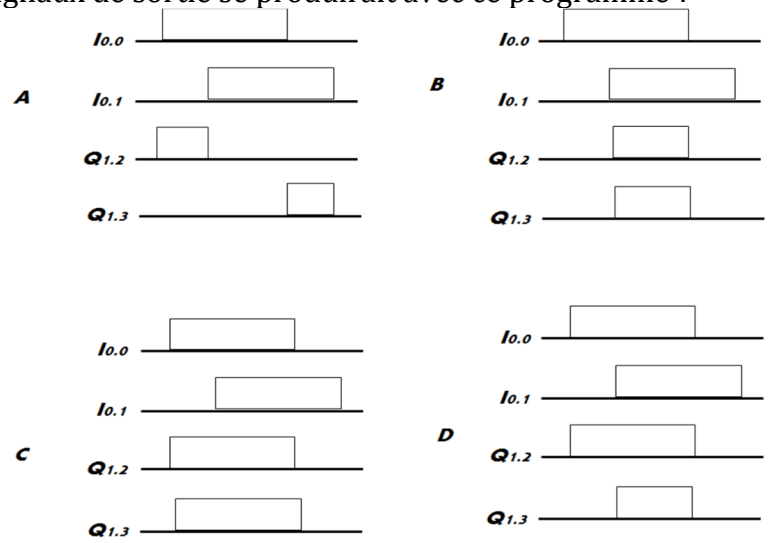
32- La figure suivante montre un programme Ladder, le lequel des diagrammes montrant les entrées et les signaux de sortie se produirait avec ce programme ?





33- La figure suivante montre un programme Ladder, dans le lequel des diagrammes montrant les entrées et les signaux de sortie se produirait avec ce programme ?





34- Définir en langage FBD les fonctions suivantes :

- Il doit y avoir un démarrage du moteur lorsque l'interrupteur A ou B est activé ;
- Un moteur doit démarrer lorsque deux boutons poussoirs (NO) sont activés et restent en marche, même si le premier des deux BP s'éteint, mais pas si le deuxième BP s'éteint.
- Une pompe doit être allumée si l'interrupteur de démarrage de la pompe est allumé ou si un interrupteur d'essai est actionné.

35- Constatez si chacune de ces instructions est Vrai (V) ou fausse (F). La figure suivante montre un programme en langage LIST pour lequel il y a une sortie quand :

```
AN  I0.1
AN  I0.2
=   Q2.1
```

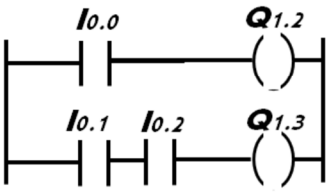
- L'entrée I0.1 est activée mais I0.2 est désactivée.
- L'entrée I0.2 et I0.1 sont activées.

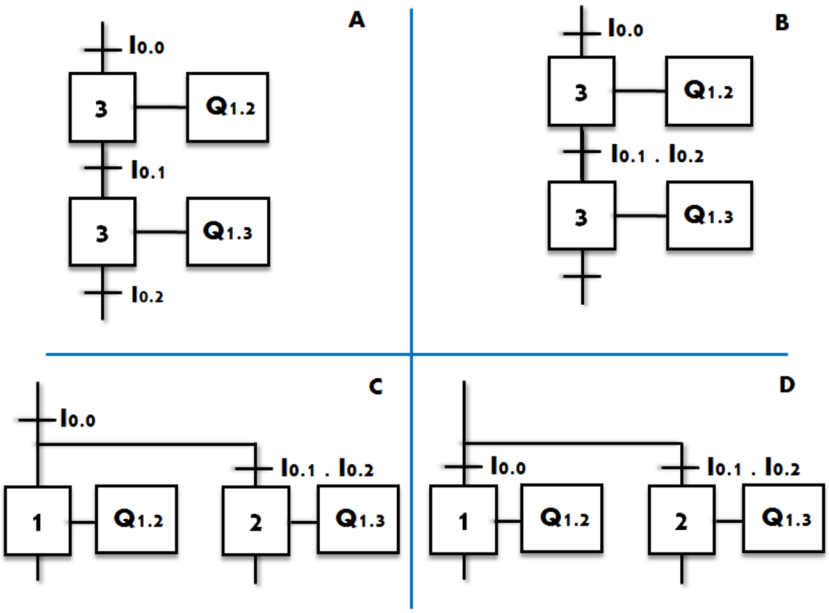
36- Constatez si chacune de ces instructions est Vrai (V) ou fausse (F). La figure suivante montre un programme en langage LIST pour lequel il y a une sortie quand :

```
A   I0.1
O   Q2.0
AN  I0.2
=   Q2.1
```

- La sortie est activée lorsque l'entrée I0.1 est momentanément activée.
- Pas de sortie quand l'entrée I0.2 est activée.

37- Pour le programme Ladder décrit à la figure ci-dessous, lequel des programmes SFC représentera-t-il ?





Exercice 01

Programmer, en langage Ladder, les fonctions suivantes :

$$Y_1 = I_0(\overline{I_1.I_2 + I_3.I_4})$$

$$Y_2 = I_0.(I_1 + I_2).(I_3 + I_4)$$

$$Y_3 = \overline{I_0}.(\overline{I_1} + I_2).(I_3.I_4)$$

$$Y_4 = \overline{I_0}.(\overline{I_1} + I_2).(I_3.I_4).I_0.(I_1 + I_2).(I_3 + I_4)$$

$$Y_5 = I_0.(I_1 + I_2).(I_3 + I_4).I_0(\overline{I_1.I_2 + I_3.I_4})$$

$$Y_6 = \overline{I_0}.(\overline{I_1} + I_2).(I_3.I_4).I_0.(I_1 + I_2).(I_3 + I_4).I_0(\overline{I_1.I_2 + I_3.I_4})$$

Reprogrammer, en langage FBD et IL les fonctions Y_1, Y_3, Y_4

Exercice 02

Un chariot initialement à gauche effectue le déplacement suivant :

- Déplacement à droite jusqu'à fin course droite ;
- Retour à sa position initiale.

Entrées :

- Fin course droite ;
- Fin course gauche ;
- Bouton poussoir marche ;
- Bouton arrêt.

Sorties :

- Bobine gauche ;
- Bobine droite.

- Faites une programmation en LADDEER de cahier des charges citez ci-dessous.
- Refaire le programme en langages FBD et IL

Exercice 03

Etablir un programme, en langage Ladder, qui décrit la commande d'une pompe (P) à l'aide de deux boutons poussoirs (Marche (M)-Arrêt(A)):

a- En appuyant sur M (marche) :

- Si la pompe (P) est arrêtée, elle démarre et continue à tourner lorsqu'on lâche le bouton M;
- Si la pompe fonctionne, elle continue à fonctionner.

b- En appuyant sur A :

- Si la pompe fonctionne, elle s'arrête et reste arrêtée lorsqu'on lâche le bouton A ;
- Si la pompe est arrêtée, elle demeure arrêtée.

Exercice 04

Monte-charge

Pour déplacer une marchandise d'un niveau à un autre, après que l'opération de chargement sera terminée. La cabine (chargée) est dans la position initiale (niveau 1, 2 ou 3) avec des portes ouvertes, l'enfoncement du bouton poussoir d'étage désiré provoque, la fermeture des portes de la cabine (après 5s de la demande si aucun opérateur ni détecté à la porte par le capteur photo-électrique **co** et une signalisation permanente de l'alarme 1 durant la fermeture) suivie d'une action de montée ou de descente pour atteindre le niveau voulu et enfin ses porte s'ouvrent pour déchargé la marchandise.

Une opération de vérification et d'entretien des systèmes mécaniques s'effectue à la fin de 100 cycles complets après le déclenchement d'une alarme (le redémarrage s'effectue par un bouton DCY).

Remarque :

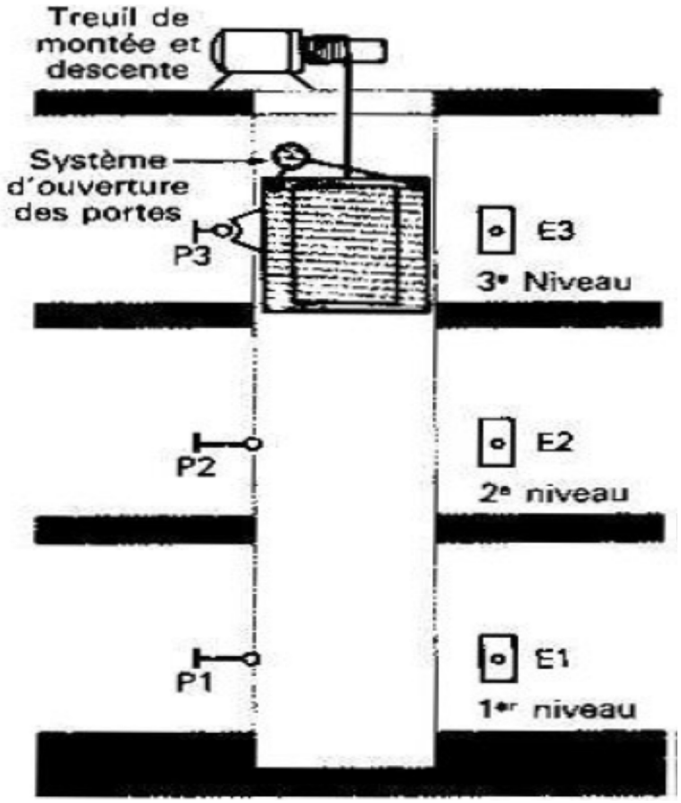
On suppose que les boutons poussoirs ne sont jamais appuyés simultanément.

Ordres :

- Mo : Montée de la cabine ;
- De : Descente de la cabine ;
- Ou : Ouverture des portes ;
- Fe : Fermeture des portes ;
- E1, E2 et E3 : Bouton poussoir d'étages.

Capteurs :

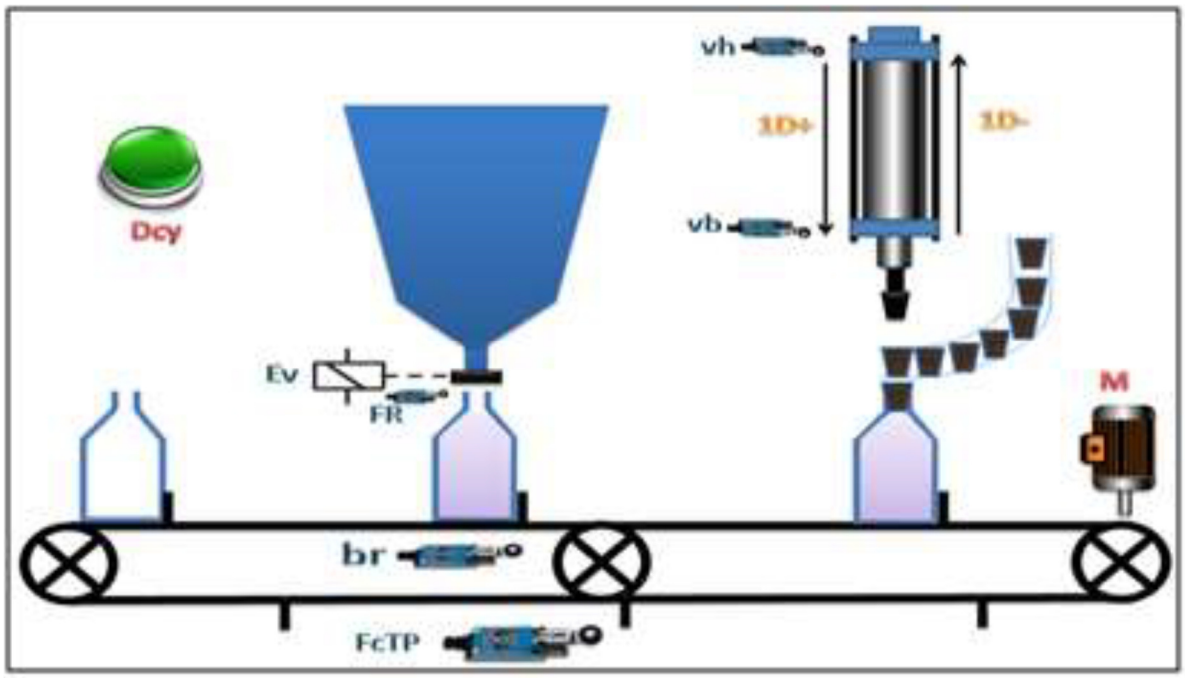
- a : Porte ouverte ;
- b : Porte fermée ;
- co : Absence d'opérateur
- P1, P2 et P3 : Positions de la cabine.



- Etablir le GRAFCET de système automatisé d'un point de vue partie opérative, avec un niveau de spécifications fonctionnelles, avec une mise en équations et la représentation sous le langage LADDER.

Exercice 05

En étudiant le fonctionnement d'un système de production des boissons liquides. La partie étudiée décrit le processus qui assure les fonctions de remplissage et de bouchage des bouteilles.



Le système est composé de :

- ✓ Un tapis roulant permettant le déplacement des bouteilles.
- ✓ Un poste de remplissage **P1** commandé par l'électrovanne **EV**.
- ✓ Un poste de bouchage **P2** commandé par un vérin presseur **1D** à double effet.
- ✓ Le déclenchement de la chaîne d'embouteillage se fait par action sur l'interrupteur **Dcy**.
- ✓ Le moteur "Avance Tapis : **M**" tourne d'un pas jusqu'à l'action du capteur "Tapis en position : **FcTP**". Une bouteille est alors présente à chacun des postes **P1** (détecter par **pbv**) et **P2** (détecter par **pbp**).

Les opérations de remplissage et de bouchage s'effectueront simultanément sur les deux bouteilles :

- **Le remplissage est réalisé en deux étapes :**
  - ✓ Ouverture de l'électrovanne **EV** ;
  - ✓ Fermeture de l'**EV** après le remplissage de la bouteille. Le capteur "Bouteille remplie: **br**" permettra de contrôler le niveau de remplissage des bouteilles.
- **Le bouchage se fera en deux étapes :**
  - ✓ Descente du vérin presseur **D+** ;
  - ✓ Remonte du vérin **D-** après l'enfoncement du bouchon.

Remarque: Le cycle ne recommencera que si les deux opérations (remplissage et de bouchage) sont achevées.

- Tracez le grafcet qui décrit le fonctionnement du système avec les boutons : marche, arrêt d'urgence, stop, arrêt momentané. Traduire ce grafcet en langage Ladder.

Exercice 06

Construire le GRAFCET d'après le combinatoire des étapes et les équations des sorties ci-dessous.

- **Combinatoire des étapes :**

X0 = X1 . X3
X1 = X0 . Sdcy . Sc0
X2 = X0 . Sdcy . Sc0
X3 = X2 . Sc1 . t/X2/2s

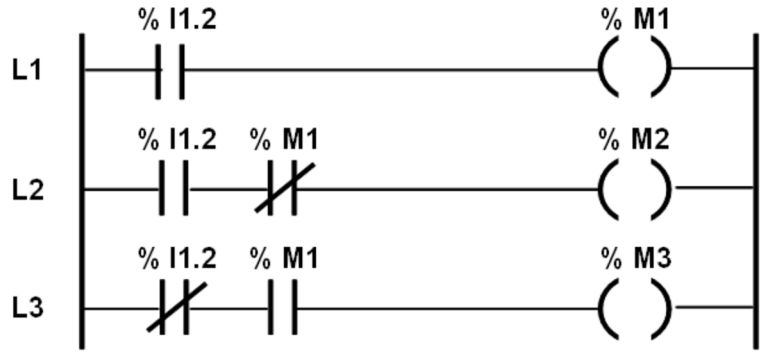
- **Équations des sorties :**

YC = X2
KMP = X1

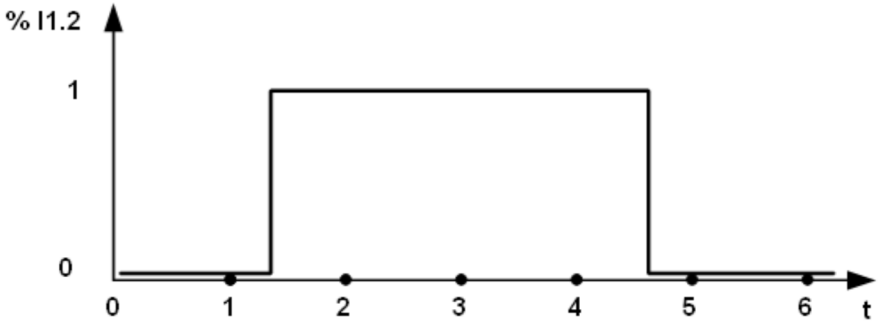
- Traduire ces équations à tous les langages possibles.

Exercice 07

Soit le schéma Ladder suivant :



- Préciser quelles sont les variables d'entrée, interne et de sortie.
- Rappeler les étapes du cycle fonctionnement d'un automate. A quelle étape du cycle les variables sont-elles *effectivement* modifiées ?
- Donner le chronogramme (graphe temporel) des variables %M1, %M2 et %M3 pour un signal %I1.2 de la forme suivante (l'échelle de temps est le cycle automate) :



- Quelles sont les fonctions [usuelles] réalisées par les lignes L2 et L3 ? L'ordre des lignes peut-il être modifié sans conséquence ? Peut-on réutiliser ces variables pour d'autres fonctions ?
- Convertir le programme Ladder en programme ST.

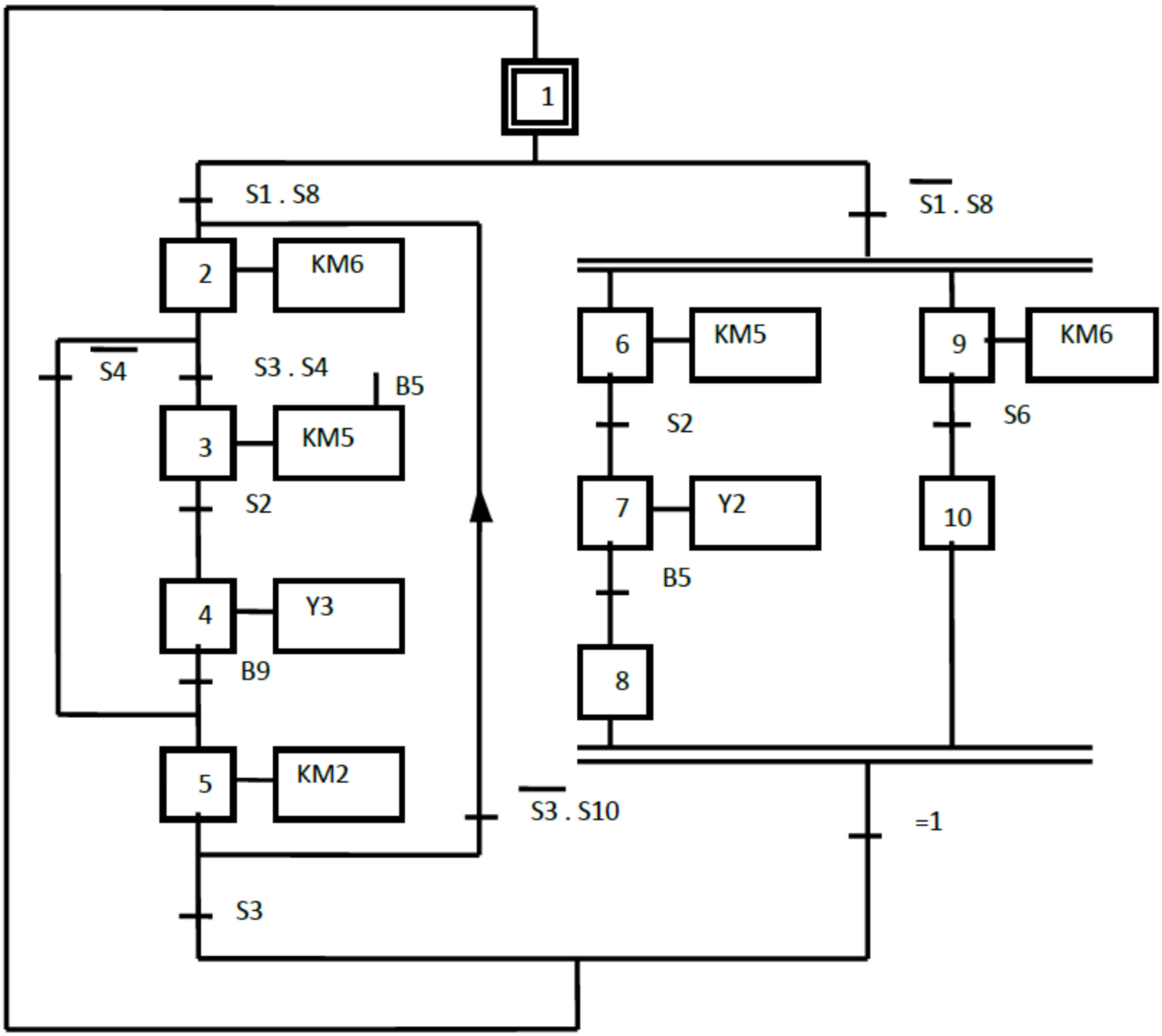
Exercice 08

Soit le programme suivant exprimé en langage ST:

```
%TM1.P := %KW0
IF %I2.0 THEN
START %TM1
END_IF
IF (%I2.1 AND %TM1.Q ) OR not( %I2.2) THEN
SET %Q3.1
END_IF
IF (%TM1.V < 3)
%Q3.2 := TRUE ;
END_IF
```

- Commenter ce programme : quelle est la fonction spécifique mise en œuvre ? quels sont les paramètres de la fonction.
- Convertir ce programme en code Ladder.

Exercice 09



- Donner les équations d'activation et de désactivation des étapes, ainsi que les équations des sorties ;
- Traduire ce grafcet en langage Ladder ;
- Refaire le programme en langages FBD et IL.

Chapitre III :

MICROPROCESSEURS, MICROCONTROLEURS ET DSP

Introduction

Le développement des systèmes électroniques et les exigences technologiques amènent de plus en plus les promoteurs des systèmes de commande et de supervision à remplacer l'électronique câblée à base de nombreux circuits intégrés par un circuit programmable moins volumineux et qui remplit à lui seul toutes les fonctions avec une grand souplesse. Cette évolution est due au développement des circuits intégrés par l'augmentation perpétuelle du taux et de la capacité d'intégration des transistors dans une puce, à partir de cette révolution apparu l'un des composants fondamentaux incontournable de tous les systèmes technologiques actuels, c'est le microprocesseur. Depuis son invention en 1971 par INTEL et avec le succès qu'a connu et devant le besoin manifeste des utilisateurs, ce composant n'a cessé de se perfectionner pour être utilisé aujourd'hui dans des domaines dépassant les prévisions de départ.

Les microprocesseurs ont connu un grand développement mais dans trois directions différentes, ce qui a abouti à la génération de :

- Microprocesseurs plus puissants: les microprocesseurs à usage général (Ordinateurs) ;
- Microprocesseurs plus complets: les microcontrôleurs ;
- Microprocesseurs plus spécialisés: les DSP (Digital Signal Processor), les GPU (Graphics Processing Unit), les ASIP (Application Specific Instruction Set Processor), ASIC (Application Specific Integral Circuit).

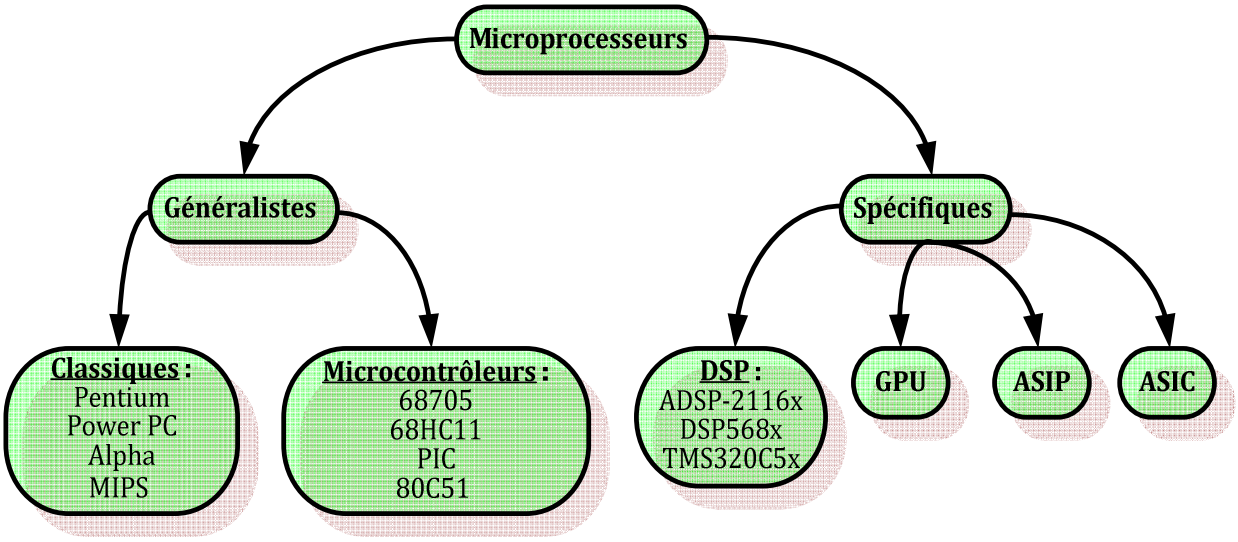


Figure III.1 Familles de microprocesseurs

Le but de cette partie de cours est d'essayer de familiariser nos étudiants avec le fonctionnement et l'utilisation des microprocesseurs, microcontrôleurs et les DSP, qui sont devenus aujourd'hui des composants électroniques clé pour tous systèmes automatisés. Bien que les privilèges de la programmation assembleur soient mis en évidence dans ce manuel, nous voulons également faire apprendre aux étudiants les limites de cette dernière et expliquer les raisons pour lesquelles la programmation de haut niveau prime.

Cet enseignement est composé de cours théoriques, de séances de travaux dirigés et de travaux pratiques.

III.1- Microcontrôleurs

L'usage de microcontrôleurs est actuellement en plein développement dans toute l'informatique industrielle, et à tous les degrés de complexité.

Ils jouent le rôle d'un ordinateur monté dans un circuit intégré, grâce à l'avancement technologique en matière d'intégration, qui a permis d'implanter sur une puce de silicium de quelques millimètres carrés la totalité des composants qui forment la structure de base d'un

ordinateur. Les microcontrôleurs améliorent l'intégration et le coût (lié à la conception et à la réalisation) d'un système à base de microprocesseur en rassemblant les éléments essentiels d'un tel système dans un seul circuit intégré. On parle alors de "système sur une puce" (en anglais : "System On chip"). Généralement un microcontrôleur est constitué de :

- Mémoires : soit morte ou ROM qui contient le programme que va exécuter l'unité centrale, soit vive ou RAM (Random Access Memory) qui est utilisée dans les phases de calcul du programme, pour stocker des résultats intermédiaires et variables d'une application.
- L'unité centrale ou CPU (Central Processing Unit) est le cœur du système puisqu'il est chargé d'interpréter les instructions du programme en cours d'exécution et de réaliser les opérations qu'elles contiennent. avec un microprocesseur d'une puissance généralement moindre, l'unité arithmétique et logique interprète, traduit et exécute les instructions de calcul.
- Les périphériques ont pour tâche de connecter le processeur avec le monde extérieur dans les deux sens. Des interfaces parallèles pour la connexion des entrées / sorties, des interfaces séries (synchrone ou asynchrone) pour le dialogue avec d'autres unités. Donc ces périphériques soit fournissent des informations vers l'extérieur (périphérique de sortie), soit ils reçoivent (périphérique d'entrée).

Tous ces éléments sont reliés entre eux par ce que l'on appelle un bus, c'est-à-dire un ensemble de liaisons transportant des adresses, des données et des signaux de contrôle.

La majorité des grands fabricants de circuits intégrés dispose aujourd'hui de plusieurs gammes de microcontrôleurs qui sont performantes (Motorola 68HC11, PIC (Programmable Interface Controller) de Microchip, PICBASIC de Comfile Technology, C167 de Siemens/Infineon, l'Intel 8085 à l'origine conçu pour être un microprocesseur, etc.)

III.1.1- Utilisation des Microcontrôleurs

Les systèmes intelligents automatiques, intégrant une technologie à base de microcontrôleur, sont divers et dans tous les domaines :

- Aéronautique, militaire et spatial : sonde, lanceurs de fusées, missile, robots ;
- Contrôle des processus industriels (régulation, pilotage, supervision) ;
- Télécommunication : carte FAX/MODEM, minitel, téléphones portables, interfaces homme machines, gestion d'écran graphique ;
- Appareil de mesure (affichage, calcul statistique, mémorisation) ;
- Automobile (ABS, régulateur de vitesse, calculateur d'injection, GPS, airbag) ;
- Multimédia (carte audio, carte vidéo, décodeur MP3, magnétoscope) ;
- Electroménager (lave-vaisselle, lave-linge, four micro-onde).

III.1.2- Avantages de microcontrôleurs

Leurs large utilisation est due aux :

- Utilisation quasi immédiate, sans problème de conception, de réalisation et de mise au point ;
- Son faible encombrement et l'optimisation de l'espace dû à sa taille ;
- Protection contre les pannes par la notion de chien de garde (Watch dog) ;
- Protection contre le piratage des programmes pour ne pas être utilisés pare d'autre personne ;
- Son prix modéré, contribue à réduire les coûts à plusieurs niveaux: moins cher que les composants qu'il remplace et diminution des coûts de main d'œuvre (conception et montage) ;
- Grande fiabilité du fait que les lignes de contrôle sont toutes internes.

III.1.3- Défauts des microcontrôleurs

Malgré leurs performances les microcontrôleurs présentent quelques inconvénients, en peuvent citer :

- Le microcontrôleur est souvent surdimensionné devant les besoins de l'application ;
- Investissement dans les outils de développement ;
- Écrire les programmes, les tester et tester leur mise en place sur le matériel qui entoure le microcontrôleur ;
- Incompatibilité possible des outils de développement pour des microcontrôleurs de même marque ;
- Les microcontrôleurs les plus intégrés et les moins coûteux sont ceux disposant de ROM programmables par masque.

III.1.4- Entrées et sorties d'un microcontrôleur

Plusieurs types de microcontrôleurs existent, pour notre cours en vas essayer de donner quelques indications pour l'exemple de microcontrôleur Motorola 68HC11, qui peut fonctionner avec des horloges allant jusqu'à 12MHz. Tous les registres étant statiques, une coupure d'horloge n'entraine pas de perte de donnée. Il existe diverses versions de 68HC11 (A1, A8, F1, E2, etc.) qui disposent de leur propre plan mémoire. Cette famille des microcontrôleurs est référenciée de la manière suivante :

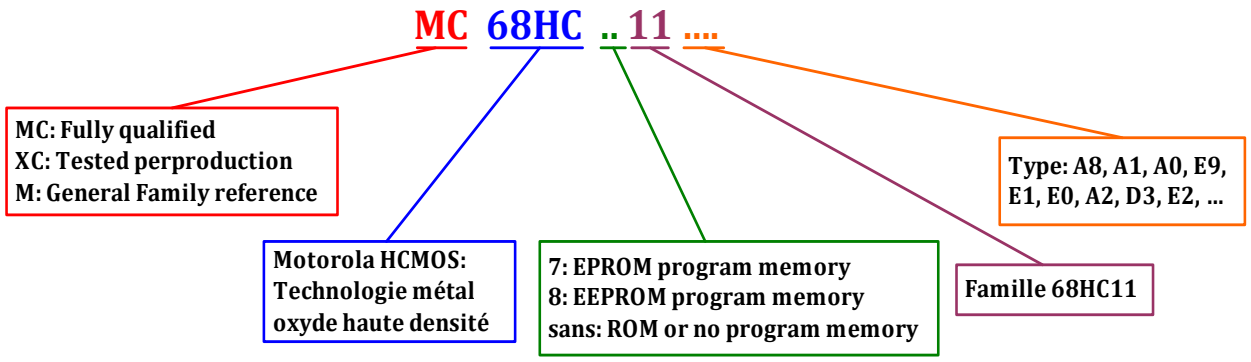


Figure III.2 Manière de référencier les microcontrôleurs de la famille 68HC11

La figure III.3 montre le diagramme bloc d'un microcontrôleur 68 HC811 E2.

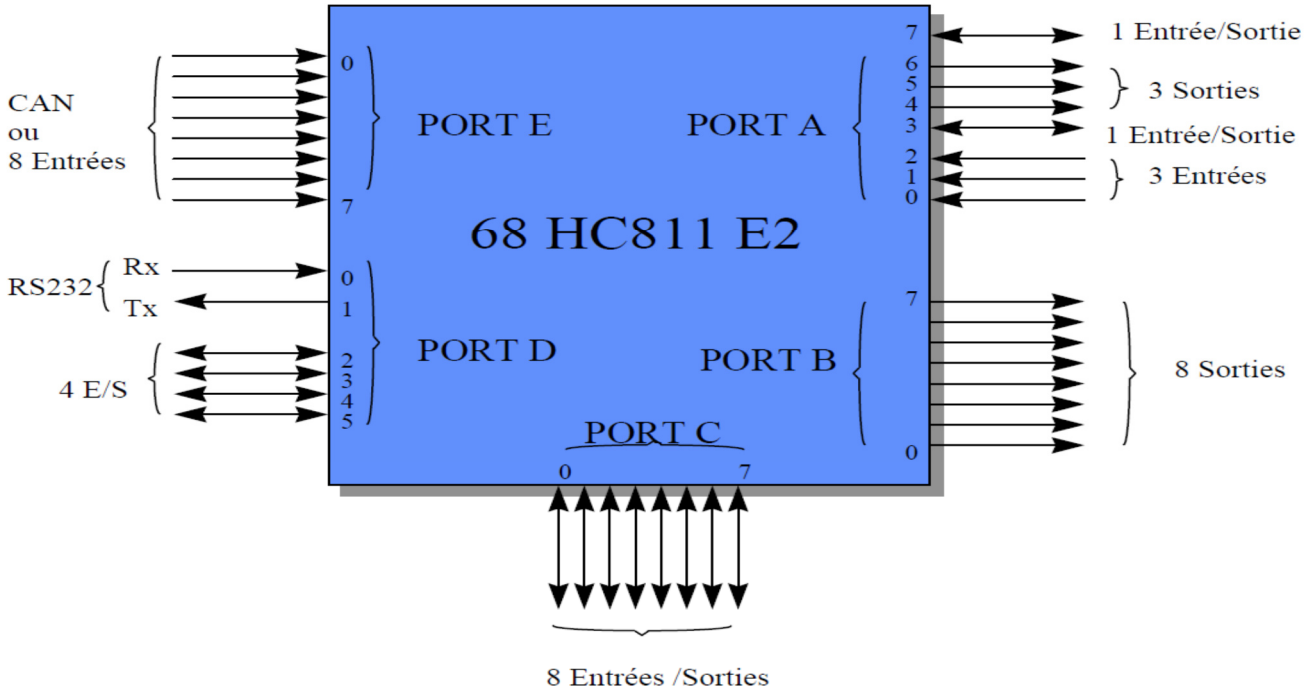


Figure III.3 Schéma d'un microcontrôleur 68HC11E2

Le microcontrôleur 68 HC811 E2 possède : 12 Entrées, 12 Sorties et de 14 Entrées / Sorties, répartis sur 5 ports de la manière suivante :

- **PORT A** : 2 Entrées/Sorties, 3 Entrées, 4 Sorties. Partagé avec les E/S du TIMER ;
- **PORT B** : 8 Sorties. Partagé avec MSB du bus adresse en mode étendu ;
- **PORT C**: 8 Entrées/Sortie. Partagé avec Bus Data et LSB adresse en mode étendu ;
- **PORT D**: 6 Entrées/Sorties. Partagé avec PORT SYN (I2C) ou ASYN (RS232) ;
- **PORT E**: 8 Entrées. Partagé avec les entrées du Convertisseur Analogique Numérique.

Le 68HC11E2 est inclus dans un boitier à 52 broches. En plus des 38 E/S des ports, il possède 14 broches:

- **VSS** et **VDD**: Alimentation VDD=5V, VSS=0 ;
- **XTAL** et **EXTAL**: Connexion d'un quartz ou d'une horloge externe ;
- **E** et **4XOUT**: Sorties d'horloge ;
- $\overline{\textbf{IRQ}}$ et $\overline{\textbf{XIRQ}}$: Entrées d'interruptions externes ;
- **VRH** et **VRL**: Tension de référence pour la conversion A/N. VRH<VDD ;
- **MODA** et **MODB**: Mode de fonctionnement ;
- $\overline{\textbf{RESET}}$: Signal d'initialisation ;
- **STRA** et **STRB**: En mode normal (mono-puce), signaux de communication avec une interface parallèle.

III.1.5- Registres du microcontrôleur

Sont des mémoires de petite taille (quelques octets), suffisamment rapides pour que l'UAL (l'unité arithmétique et logique) puisse manipuler leur contenu à chaque cycle de l'horloge. Un microcontrôleur utilise en permanence un certain nombre de registres pour contrôler et traiter son information. On peut considérer ces registres comme des mémoires internes au CPU plus ou moins spécialisées par le jeu d'instructions qui leur sont attachées. Le nombre, les noms et les rôles des registres sont spécifiques à un microcontrôleur donné. Toutefois, on retrouve toujours, sous des noms différents et avec des modes d'utilisation variables, les mêmes types de registres dans tous les microcontrôleurs courants. Dans le cas du HCS12, les registres sont les suivants :

- **Accumulateurs A, B et D :** Les registres A et B sont des accumulateurs 8 bits indépendants utilisés pour toutes opérations arithmétiques et logiques. A et B peuvent être concaténés et former un accumulateur D de 16 bits. Attention cet accumulateur n'est pas indépendant de A et B.
- **Registres d'index X et Y :** Les registres d'index X et Y, indépendants, possèdent 16 bits chacun, puisque la capacité d'adressage du 68HC11 est de 16 bits d'adresses. Leur rôle premier est d'être utilisé pour l'adressage indexé, mais ils peuvent être utilisés pour le stockage temporaire de données, ou pour quelques opérations arithmétiques élémentaires.
- **Pointeur de pile S :** Le pointeur de pile S est un registre 16 bits. Ce registre pointe sur la première adresse libre de la zone mémoire définie pour ranger différents paramètres, tel registre PC, registre CCR, etc., suivant la demande (interruption, appel à un sous-programme). Il est indispensable de l'initialiser à une adresse RAM, sinon à la première interruption ou appel à un sous-programme, le programme se plantera (bug).
- **Compteur programme ou PC :** Le compteur programme, PC, indique l'adresse du prochain code binaire à 8 bits (instruction ou valeur de travail) à traiter. La taille du registre PC est donc de 16 bits.
- **Registre d'état ou CCR :** Le registre d'état ou CCR (Condition Code Register) est un registre sur 8 bits. Chaque bit à une signification particulière.

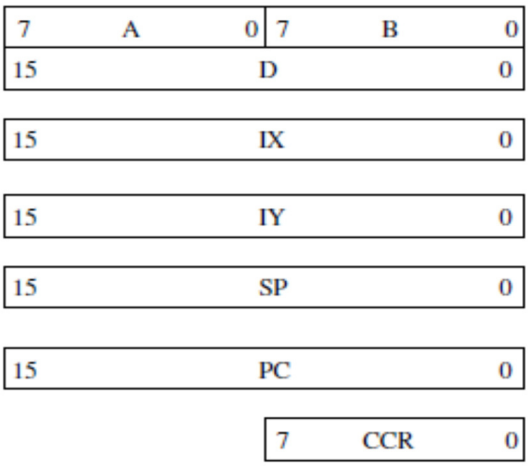


Figure III.4 Registres de travail du 68HC11

III.1.6- Programmation des microcontrôleurs

Pour programmer un microcontrôleur, il est nécessaire de connaître sa structure interne : registres, mémoires, ports d'entrées sorties, et toutes leurs possibilités. Le langage utilisé pour la réalisation du « programme produira une suite d'instructions plus ou moins «évoluées» (riches). Tous les microcontrôleurs se disent supporter les langages de haut niveau (HLL) mais il s'agit souvent plus d'un argument commercial qu'une réalité. Anciennement le langage assembleur (sera développé dans la partie microprocesseur) représentait la référence en la matière, ce langage est directement compris, après transcription en fichier binaire. Fortement bas niveau (et donc, intrinsèquement optimisé), l'assembleur posa et pose toujours d'énormes problèmes d'élaboration et de mise à jour des programmes. Pour contourner cette difficulté, les langages dits évolués sont développés : Basic, Pascal, C, C++, C#, java…etc. Un logiciel se charge alors de traduire les instructions en code machine : c'est le compilateur. Si en prend l'exemple de 68HC12, a été conçu dans cette optique, ce qui n'est pas le cas des autres microcontrôleurs bas de gamme comme le 68HC11. Avec les microcontrôleurs non adaptés aux HLL, le compilateur "se débrouille" mais au prix d'une "source-assembleur" extrêmement lourde puisqu'il faut 5 à 10 instructions là où le 68HC12, 6809 ou 68000 n'en nécessite qu'une seule. La principale différence entre le 68HC11 et le 68HC12 est la présence d'une mémoire FLASH dans ce dernier microcontrôleur. Cette mémoire est importante et permet de programmer directement en langage C ou C++.

III.2- Processeurs DSP

Le premier processeur ou calculateur spécialisé (apparue vers 1982) en traitement du signal (TS ou SP pour Signal Processing), le "DSP" (Digital Signal Processor), était né et depuis, cette famille n'a cessé de s'agrandir. Parallèlement aux microprocesseurs et aux microcontrôleurs, les processeurs de traitement numérique du signal, ou DSP (Digital Signal Processor), ont bénéficié des énormes progrès en rapidité (grâce au faible temps de commutation) et en puissance de calculs (grâce au nombre de bits des bus internes) des composants logiques intégrés programmables. Par exemple, l'une des opérations les plus courantes en traitement du signal est la somme de produits qu'on retrouve dans les problèmes de filtrage numérique :

$$y(n) = \sum_{i=0}^{n_a} a_i . x(n-i) - \sum_{i=0}^{n_b} b_i . y(n-j) \qquad \text{(III. 1)}$$

Les différentes étapes d'insertion d'une DSP dans une chaine de traitement du signal sont illustrées dans la figure III.4. Cette chaine comporte généralement 03 étapes :

- Etape de conversion de signal analogique vers le numérique : le traitement numérique du signal (DSP) implique la manipulation de signaux numériques afin d'en extraire des

informations utiles. Bien qu'une quantité croissante de traitement du signal soit effectuée dans le domaine numérique, il reste le besoin de s'interfacer avec le monde analogique dans lequel nous vivons.

- Etape de traitement du signal par le DSP : dans cette étape s'effectue le traitement désiré comme le filtrage, détection d'un signal, estimation, régulation, modulation, etc.
- Etape de conversion numérique analogique : le signal traité est alors de nouveau converti (si nécessaire) en signal analogique pour l'utiliser dans l'application à laquelle est destiné.

A noté que les étapes d'entrée et de sortie peuvent elles-mêmes être décomposées en plusieurs étapes.

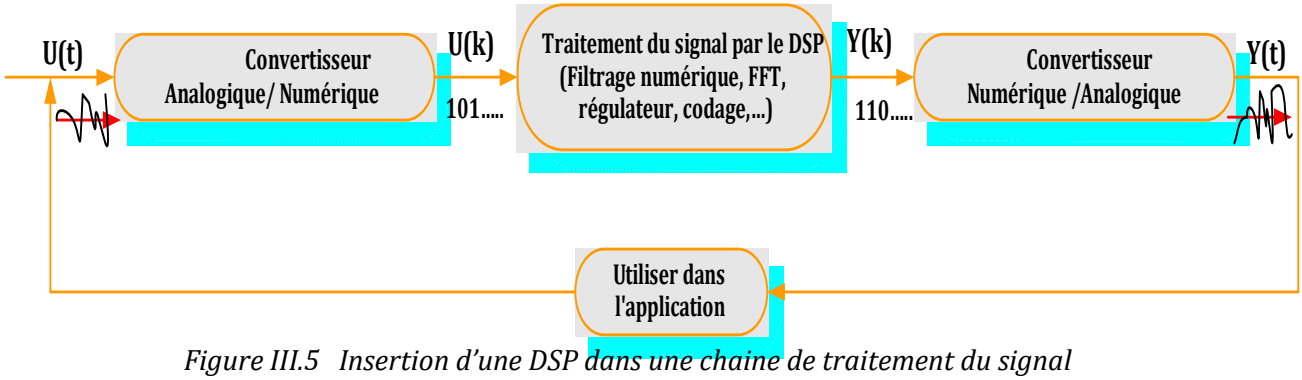


Figure III.5 Insertion d'une DSP dans une chaine de traitement du signal

Il existe de nombreuses raisons pour lesquelles on voudrait traiter un signal analogique de manière numérique en le convertissant en un signal numérique. La raison principale est que le traitement numérique permet la programmabilité. Le même DSP peut être utilisé pour de nombreuses applications différentes en changeant simplement le code résidant en mémoire. Une autre raison est que les circuits numériques fournissent une sortie plus stable et tolérante que les circuits analogique, par exemple lorsqu'ils sont soumis à des changements de température.

III.2.1- Classification des DSP

Les DSP sont classés en fonction de l'amplitude et le type de données qui sont capables de traiter en deux grandes catégories:

a- DSP en virgule fixe : le codage en virgule fixe, représente un nombre réel en partant d'un nombre entier signé en complément à 2, on rajoute simplement dans la représentation une virgule **virtuelle** permettant de séparer la partie entière et la partie décimale. On utilise N bits ($m+k$) pour représenter les nombres binaires non signés ou signés (complément à 2), m bits pour la partie entière et k bits pour la partie fractionnaire. Les données sont codées de 16 à 32 bits à l'exemple de : DSP56000 (freescale), Shark (Analog Device), Blackfin (Analog Device), TMS 320C25, etc. Ce genre de DSP sont utilisés pour le conditionnement de signaux vocaux et de trouver leur principal domaine d'application téléphonie (Fixe et mobile).

$$Nombre = -b_{m-1}.2^{m-1} + \sum_{i=-k}^{m-2} b_i . 2^i \qquad \text{(III. 2)}$$

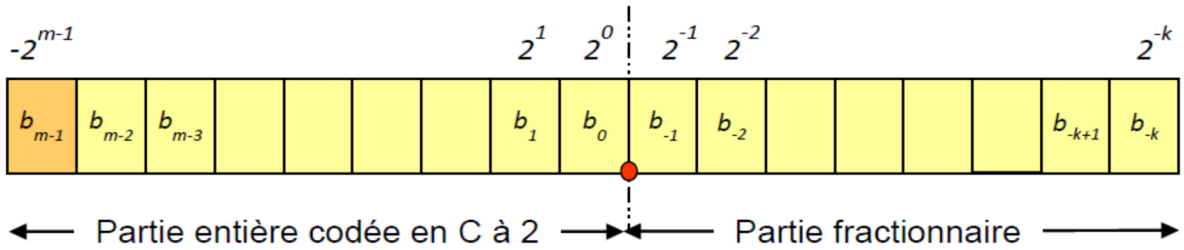


Figure III.6 Représentation d'un nombre réel en virgule fixe

b- DSP en virgule flottante : la notation en virgule flottante est très similaire à la notation scientifique en base 10. En base 10, nous fixons un nombre de bits significatif, puis nous représentons le nombre sous la forme :

$$Nombre = mantisse \cdot 10^{exposant}$$

Dans la notation scientifique en base 10 : 1≤| mantisse |<10

En binaire, la représentation est similaire mais en base 2 :

$$Nombre = mantisse \cdot 2^{exposant}$$

Dans la notation scientifique en base 2 : 0 ≤| mantisse |<2

- La mantisse est un nombre réel codé sur M bits en virgule fixe
- L'exposant est un nombre entier signé codé sur E bits en complément à 2.

DSP virgule flottante de 32 bits (à l'exemple de TMS 320C25) ayant une dynamique beaucoup plus élevé, sont principalement utilisés dans le traitement de l'image et en graphique tridimensionnel.

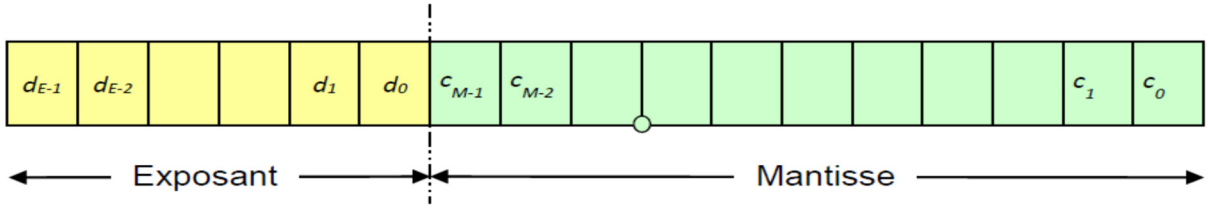


Figure III.7 Représentation binaire d'un nombre en virgule flottante

III.2.2- Domaines d'applications des DSP

Les principales applications se trouvent dans :

- Domaine d'ordre général : filtrage numérique, convolution, filtrage adaptatif, générateur de signal etc. ;
- Télécommunications : téléphone filaire et cellulaire, modem, fax, transcodeurs, interpolateurs, répondeurs, routeurs, codec etc. ;
- Traitement de la parole et de voix: identification de voix, reconnaissance de la parole, courrier électronique vocal, compression, synthèse de la parole, décodeur MP3, etc. ;
- Commande de procédés : des moteurs et des robots, asservissement, contrôle flou, diagnostique automatique, automobile (ABS), commande des imprimantes lasers, etc. ;
- Domaine militaire : sécurisation des communications, radar, guidage des missiles, etc. ;
- Instrumentation : analyse spectrale, oscilloscope, traitement de données sismiques etc. ;
- Traitement d'image et graphisme: transmission et compression d'image, image par résonance magnétique, image ultrasonore, rotation 3D, reconnaissance, compression, transmission, animation, etc.

III.2.3- Architecture des DSP (Exemple de TMS320 C6x)

La famille de processeurs TMS320C6x, fabriqués par Texas Instruments, est conçue pour offrir de la vitesse. Ils sont conçus pour des millions d'instructions par seconde (MIPS), telles que l'imagerie sans fil et numérique de troisième et quatrième génération (3 et 4G). Il existe de nombreuses versions de processeurs appartenant à cette famille, différents par le temps de cycle d'instruction, la vitesse, la consommation d'énergie, la mémoire, les périphériques, l'emballage et le coût. Par exemple, la version C6416-600 à virgule fixe fonctionne à 600 MHz (temps de cycle de 1,67 ns), offrant une performance de pointe de 4800 MIPS. La version à virgule flottante C6713-225 fonctionne à 225 MHz (temps de cycle de 4,4 ns), offrant une performance de pointe de 1350 MIPS.

La figure III.7, montre un schéma de principe de l'architecture générique C6x. L'unité centrale de traitement (CPU) du 6x se compose de huit unités fonctionnelles divisées en deux côtés: A et B. Chaque côté a une unité .M (utilisée pour l'opération de multiplication), une unité .L (utilisée pour les opérations logiques et arithmétiques), une unité .S (utilisée pour le

branchement, la manipulation des bits et les opérations arithmétiques) et une unité .D (utilisée pour le chargement, le stockage et les opérations arithmétiques). Certaines instructions, telles que ADD, peuvent être effectuées par plusieurs unités. Il y a seize registres de 32 bits associés à chaque côté. L'interaction avec le CPU doit se faire via ces registres.

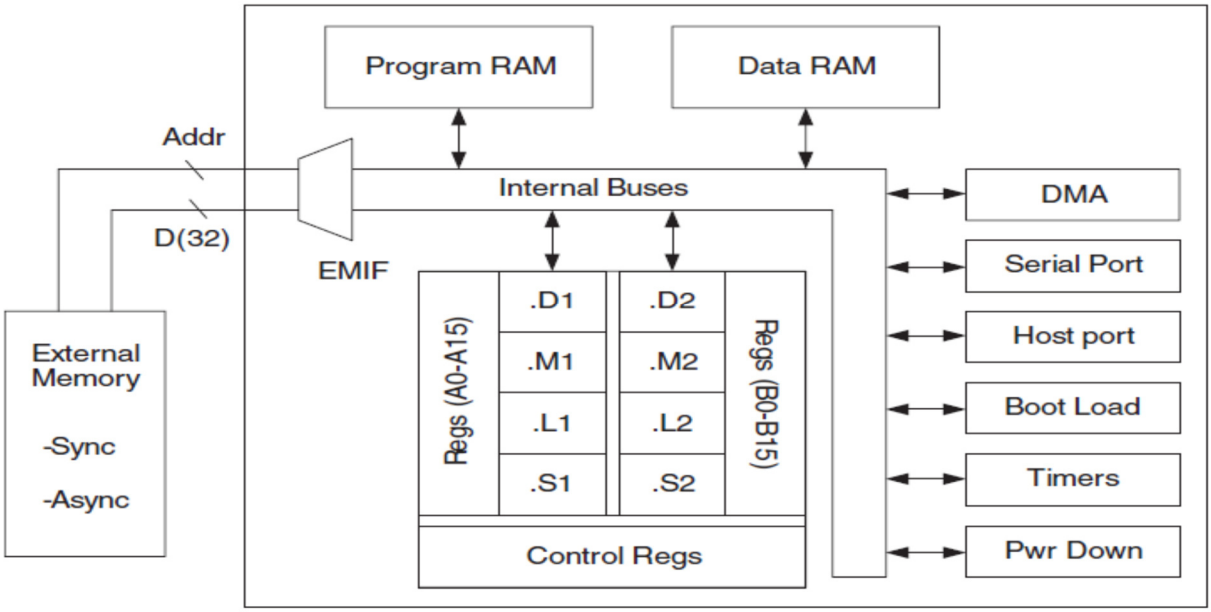


Figure 3.8 Architecture générique du C6x

III.2.4- Pipeline des instructions

Dans tout processeur, le traitement d'une instruction de programme se déroule sur quatre étapes :

1. Aller chercher l'instruction en mémoire programme (Fetch) ;
2. Réaliser le décodage de l'instruction, et des adresses des opérandes (Decode) ;
3. Lire les opérandes en mémoire de données (Read) ;
4. Exécuter l'opération et écrire le résultat (Execute).

Sans le mécanisme de pipeline ces quatre étapes doivent être entièrement achevées pour une instruction, afin que le traitement de l'instruction suivante dans le programme puisse s'entamer. Pour améliorer les performances de l'unité de traitement, les DSP les plus récents utilisent la méthode du pipeline. Elle consiste à imposer un ordre et un rythme dans le déroulement des instructions de manière à optimiser en rapidité leur exécution Le principe de pipeline consiste à découper le travail en tâches élémentaires de même durée pour permettre leur réalisation en parallèle. Il faut prévoir des registres tampon entre chaque opération élémentaire, ce qui montre le Tableau III.1.

Tableau III.1 Principe de pipeline

Instruction	t	T+1.t_{clk}	T+2.t_{clk}	T+3.t_{clk}	T+4.t_{clk}	T+5.t_{clk}	T+6.t_{clk}
n	Fetch	Decode	Read	Execute			
n+1		Fetch	Decode	Read	Execute		
n+2			Fetch	Decode	Read	Execute	
n+3				Fetch	Decode	Read	Execute

III.2.5- Programmation

Les DSP sont des composants comme les microcontrôleurs et les microprocesseurs qui exécutent un programme qui constitué de suites d'instructions. Le problème qui se pose c'est le choix du langage de programmation : langage de bas niveau (assembleur) ou langage de haut niveau (C, Basic, Pascal,…). A l'exception des cas où une rapidité d'exécution ou compacité du code est recherchée, les développements se font autant que possible en langage de haut niveau (C et C^{++} principalement, éventuellement JAVA avec l'apparition récente de

quelques compilateurs). Pratiquement on effectue un mixage des deux méthodes: modules assembleurs pour les parties requérant de la vitesse d'exécution et langage de haut niveau pour le reste du programme.

NB. *Il est à noter toutefois que pour avoir un bon niveau d'expertise des systèmes au temps, réel la connaissance de l'assembleur peut s'avérer incontournable.*

III.3- Microprocesseur

Un microprocesseur est un circuit intégré complexe caractérisé par une très grande intégration et doté des facultés d'interprétation et d'exécution des instructions d'un programme. Il est chargé d'organiser les tâches précisées par le programme et d'assurer leur traitement. C'est le cerveau du système. Il est parfois appelé CPU (Central Processing Unit).

Le concept de microprocesseur a été créé par la société INTEL. Cette Société, créée en 1968, était spécialisée dans la conception et la fabrication de puces mémoire. À la demande de deux de ses clients — fabricants de calculatrices et de terminaux — Intel étudia une unité de calcul implémentée sur une seule puce. En 1971, c'est la date de premier microprocesseur, le 4004, qui était une unité de calcul à 4 bits fonctionnant à 108 kHz. Il résultait de l'intégration d'environ 2300 transistors.

Un microprocesseur est caractérisé par sa fréquence d'horloge (en MHz), sa largeur des bus de données et d'adresse, sa mémoire adressable, le nombre de transistors et la taille de la gravure (en microns). Ses performances sont caractérisées par le CPI (Cycle Par Instruction) et le MIPS (Millions d'Instructions Par Secondes) qui dépendent directement de la fréquence d'horloge et son jeu d'instructions. Parmi les microprocesseurs fabriqués en trouvant:

- Microprocesseurs à 8 bits : 8080 et 8085 d'Intel, 6800 de Motorola, Z80 de Zilog, début des années 1980 ;
- Microprocesseurs 16 bits : 8086/8088 d'Intel, 68000 de Motorola ;
- Microprocesseurs 32 bits en 1986 : 80386 d'Intel et 68020 de Motorola.

III.3.1- Caractéristiques générales du 8086

Le processeur 8086 d'Intel est à la base des processeurs Pentium actuels, est un microprocesseur 16 bits, apparu en 1978.

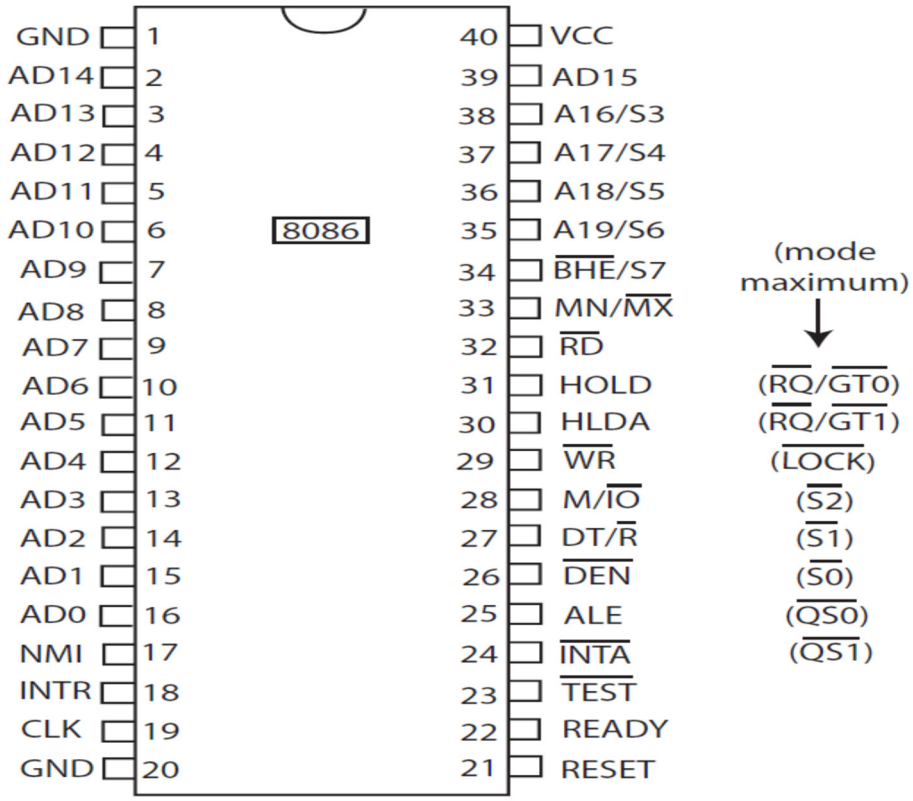


Figure III.9 Architecture extérieure du processeur 8086 d'Intel

C'est le premier microprocesseur de la famille Intel 80x86 (8086, 80186, 80286, 80386, 80486, Pentium, ...). C'est un circuit intégré sous la forme d'un boıtier DIP (Dual In-line Package) à 40 broches :

- **CLK** : Entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur. Ce signal provient d'un **générateur d'horloge** : le 8284 ;

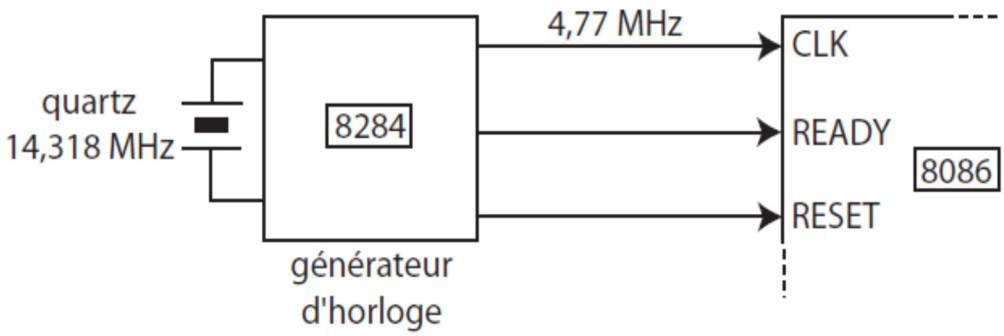


Figure III.10 Générateur d'horloge

- **RESET** : Entrée de remise à zéro du microprocesseur ;
- **READY** : Entrée de synchronisation avec la mémoire. Ce signal provient également du générateur d'horloge ;
- $\overline{\textbf{TEST}}$: Entrée de mise en attente du microprocesseur d'un évènement extérieur ;
- **MN/$\overline{\textbf{MX}}$:** Entrée de choix du mode de fonctionnement du microprocesseur :
  - Mode minimum (**MN/$\overline{\textbf{MX}}$** = 1) : le 8086 fonctionne de manière autonome, il génère lui-même le bus de commande ($\overline{\textbf{RD}}$, $\overline{\textbf{WR}}$) ;
  - Mode maximum (**MN/$\overline{\textbf{MX}}$**= 0) : ces signaux de commande sont produits par un **contrôleur de bus**, le 8288. Ce mode permet de réaliser des systèmes multiprocesseurs.
- **NMI** et **INTR** : Entrées de demande d'interruption. INTR : interruption normale, NMI (Non Maskable Interrupt) : interruption prioritaire.
- $\overline{\textbf{INTA}}$: Interrupt Acknowledge, indique que le microprocesseur accepte l'interruption.
- **HOLD** et **HLDA** : signaux de demande d'accord d'accès direct à la mémoire (DMA).
- **S0** à **S7** : Signaux d'état indiquant le type d'opération en cours sur le bus.
- **A16/S3** à **A19/S6** : 4 bits de poids fort du bus d'adresses, **multiplexés** avec 4 bits d´état;
- **AD0** à **AD15** : 16 bits de poids faible du bus d'adresses, **multiplexés** avec 16 bits de données. Le bus A/D est multiplexé (multiplexage temporel) d'où la nécessite d'un **démultiplexage** pour obtenir séparément les bus d'adresses et de données :
  - 16 bits de données (microprocesseur 16 bits) ;
  - 20 bits d'adresses, d'où 2^{20} = 1 Mo d'espace mémoire adressable par le 8086.

III.3.2- Architecture interne d'un Microprocesseur 8086

Le 8086 est constitué essentiellement de deux unités internes distinctes (Figure III.11) fonctionnant en parallèle :

a- Unité d'interface de bus (BIU : Bus Interface Unit) : qui fournit l'interface physique entre le microprocesseur et le monde extérieur, génère les adresses grâce à un additionneur (∑) et lit les instructions qu'il range dans une file d'attente (Queue), de 6 octets pour le 8086 et de 4 pour le 8088.

b- Unité d'exécution (EU : Execution Unit) : qui contient essentiellement l'unité arithmétique et logique (UAL) de 16 bits qui manipule les registres généraux de 16 bits aussi, son rôle est d'exécuter les instructions.

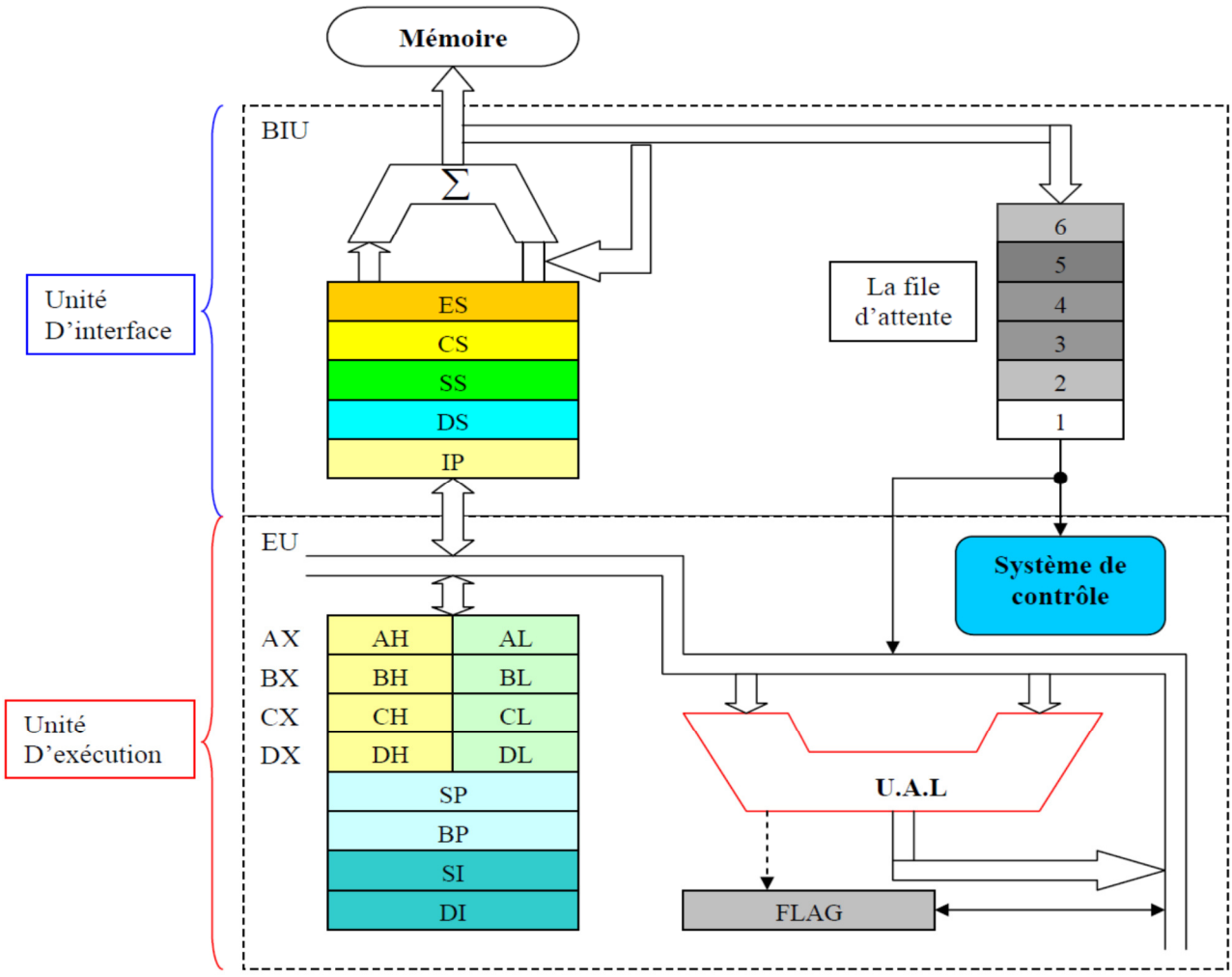


Figure III.11 Schéma de la structure interne de µP 8086

c- Unité Arithmétique et Logique : C'est un circuit complexe et comme son nom l'indique, cette unité peut exécuter deux types d'opérations :

- **Opérations arithmétiques** : Elles incluent l'addition et la soustraction qui sont des opérations de base (une soustraction est une addition avec le complément à deux). Les données traitées sont considérées dans des représentations entières ;
- **Opérations logiques** : Ces opérations sont effectuées bit à bit sur les bits de même poids de deux mots, tel que And, Or, Not, Ou exclusif, de même les opérations de rotation et de décalage (arithmétique et logique).

Elle reçoit ses opérandes (les octets qu'elle manipule) du bus de données. Celles-ci peuvent provenir des registres, de la mémoire et aussi par un registre particulier appelé "accumulateur" ou plus particulièrement le registre de travail (Working Register, le registre W). À la fin d'une opération, UAL peut aller modifier certains bits du registre d'état.

Il permet donc de traiter et tester les données. Comme l'objectif est de développer un programme qui gère une application voulue, toute instruction qui modifie une donnée fait toujours appel à l'UAL.

d- Registres : Un registre est une petite partie de mémoire intégrée au microprocesseur, dans le but de recevoir des informations spécifiques, notamment des adresses et des données stockées durant l'exécution d'un programme. Il existe plusieurs types de registres. Certains d'entre eux sont affectés à des opérations d'ordre général et sont accessibles au programmeur à tout moment. Nous disons alors qu'il s'agit de registres généraux. D'autres registres ont des rôles bien plus spécifiques et ne peuvent pas

servir à un usage non spécialisé. Les registres du 8086 se décomposent en 4 grandes familles :

✓ **Registres de données (04)** : Se décomposant chacun en deux parties : une partie « haute » et une partie « basse » de 8 bits chacune, ce qui permet au microprocesseur de manipuler des données sur 8 ou 16 bits :

- AX (décomposable en AH et AL) sert d'accumulateur et est principalement utilisé lors d'opérations arithmétiques et logiques. Les multiplications, les divisions, les traitements de chaînes, les transferts entrées-sorties, la conversion en BCD du résultat d'une opération arithmétique se font tous par son intermédiaire ;
- BX registre de base, composé de BH et de BL, est la plupart du temps utilisé comme opérande dans les calculs ;
- CX composé de CH et de CL, est utilisé en compteur lors des instructions de boucles, dont les traitements de chaînes. Le registre CL est utilisé en compteur lors des instructions de décalages et de rotations multiples (sur plusieurs bits) ;
- DX composé de DH et de DL, tout comme AX, est utilisé pour les calculs arithmétiques et notamment dans la division et la multiplication (extension 32 bits pour les multiplications et les divisions de mots). Il intervient également dans les opérations d'entrées/sorties.

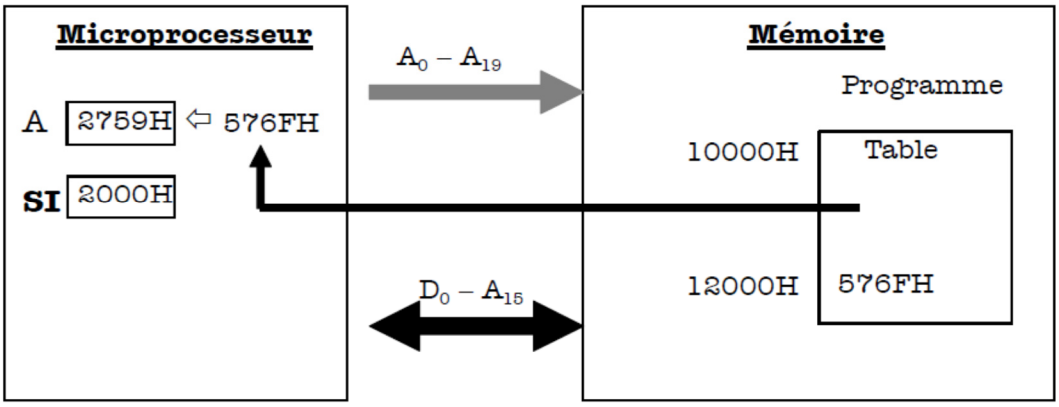
✓ **Registres de segmentation :** L'espace mémoire de 1 Mo est découpé en tranches de 64 Ko maximum, appelées « segments ». Le CPU possède un accès direct aux quatre segments. Leurs adresses de base se trouvent dans les registres segments.

- CS (segment de code) permet de déterminer les adresses sur 20 bits, définit le début de la zone mémoire programme. Les adresses des différentes instructions sont représentées par un offset relatif au registre CS.
- DS (segment de données), définit la zone mémoire réservée aux données traitées par le programme dans lequel déposeront ses éléments de travail (variables, champs, tableaux de branchement, etc.).
- SS (segment de pile), le SS pointe sur la pile : la pile est une zone mémoire de sauvegarde des registres ou les adresses ou les données pour les récupérer après l'exécution d'un sous-programme ou l'exécution d'un programme d'interruption, en général il est conseillée de ne pas changer le contenu de ce registre car on risque de perdre des informations très importantes (exemple les passages d'arguments entre le programme principal et le sous-programme).
- ES (segment supplémentaire), le registre de données supplémentaires ES est utilisé par le microprocesseur lorsque l'accès aux autres registres est devenu difficile ou impossible pour modifier des données, de même ce segment est utilisé pour le stockage des chaînes de caractères.

✓ **Registres pointeurs ou d'index :** Ces registres sont plus spécialement adaptés au traitement des éléments dans la mémoire. Ils sont en général munis de propriétés d'incrémentation et de décrémentation.

- SP (Stack Pointer : SP ou pointeur de pile) pointe sur le sommet de la pile de données, pour stocker des données ou des adresses selon le principe du "Dernier Entré Premier Sorti" ou "LIFO" ;
- BP (**B**ase **P**ointer ou pointeur de base) pointe sur la base de la pile de données, utilisé pour adresser des données sur la pile.
- SI (index de source) : Il permet de pointer la mémoire il forme en général un décalage (un offset) par rapport à une base fixe (le registre DS), il sert aussi pour les instructions de chaîne de caractères, en effet il pointe sur le caractère source.

Exemple : MOV A, [SI+10000H] place le contenu de la mémoire d'adresse 10000H+le contenu de SI, dans le registre A.



- DI (index de destination), il permet aussi de pointer la mémoire, il présente un décalage par rapport à une base fixe (DS ou ES), il sert aussi pour les instructions de chaîne de caractères, il pointe alors sur la destination. Exemple : mov [di], bx : charge les cases mémoire d'offset DI et DI + 1 avec le contenu du registre BX.

✓ **Registre IP (Instruction Pointer)** : Est un registre du processeur qui désigne la prochaine instruction à exécuter par le microprocesseur.

✓ **Registre spécial,** contenant 9 indicateurs binaires nommés « flags » :

15															0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

- **CF** (indicateur de retenue) est mis à 1 lorsqu'il y a eu une retenue lors d'un calcul, cet indicateur et mis à 1 lorsque il y a une retenue du résultat à 8 ou 16 bits. il intervient dans les opérations d'additions (retenue) et de soustractions (borrow) sur des entiers naturels. Il est positionné en particulier par les instructions ADD, SUB et CMP.
  Exemples (sur 8 bits pour simplifier) :

```
    10010110          11011001
+   01010100      +   01010010
CF=0 11101010     CF=1 00101011
```

- **PF** (indicateur de parité) est mis à 1 lorsque le résultat d'une opération contient un nombre pair de 1 ;
- **AF** (indicateur de retenue auxiliaire), la retenue intermédiaire est la retenue qui se propage du quartet (4 bits) de poids faible vers le quartet du poids fort.
- **ZF** (indicateur de zéro) est mis à 1 lorsque le résultat d'une opération vaut 0 ;
- **SF** (indicateur de signe) représente le signe du résultat d'une opération, le signe est le bit de poids fort, huitième ou seizième bit, d'un nombre en arithmétique signé. Comme les nombres binaires négatifs sont représentés en complément à deux, **SF** représente le signe du résultat : SF =0, le nombre et positif, SF = 1, le nombre est négatif.

```
    0 1 0 0          1 1 0 0          1 1 1 1
+   0 1 1 0      +   0 1 1 0      +   0 0 0 1
-------------    -------------    -------------
SF=1 1 0 1 0     SF=0 0 0 1 0     SF=0 0 0 0 0
```

- **OF** (indicateur de débordement) est mis à 1 lorsqu'un débordement arithmétique a eu lieu (lorsque le résultat d'une opération ne peut tenir sur 16 bits). Ceci signifie que le format du résultat a dépassé les possibilités de stockage de la destination est qu'un bit significatif a été perdu.

- **DF** (indicateur de direction), à chaque exécution d'une instruction de traitement de chaîne, les registres d'indexes sont incrémentés ou décrémentés selon le flag DF de registre d'état. Si DF=0 alors, on va incrémenter l'index SI sinon on décrémente.
- **IF** (indicateur d'autorisation d'interruption), pour masquer les interruptions venant de l'extérieur ce bit est mis à 0, dans le cas contraire le microprocesseur reconnaît l'interruption de l'extérieur.
- **TF** (indicateur d'interruption pas à pas), ce mode permet la mise au point d'un programme pas à pas, sans gestion externe.

e- Bus

Un bus est un ensemble de fils qui assure la transmission d'informations entre le microprocesseur, la mémoire et l'unité d'E/S, sous forme de mots binaires. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l'information :

- Bus d'adresse : c'est un bus unidirectionnel qui permet la sélection des informations à traiter dans un *espace mémoire* à lire ou à écrire (Dans le cas du 8086 est de taille 20 bits);
- Bus de données : c'est un bus bidirectionnel qui assure la transmission et la réception des données en parallèle entre le microprocesseur et la mémoire ou les E/S. (Pour le 8086 est de taille 16 bits) ;
- Bus de commande: constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses.

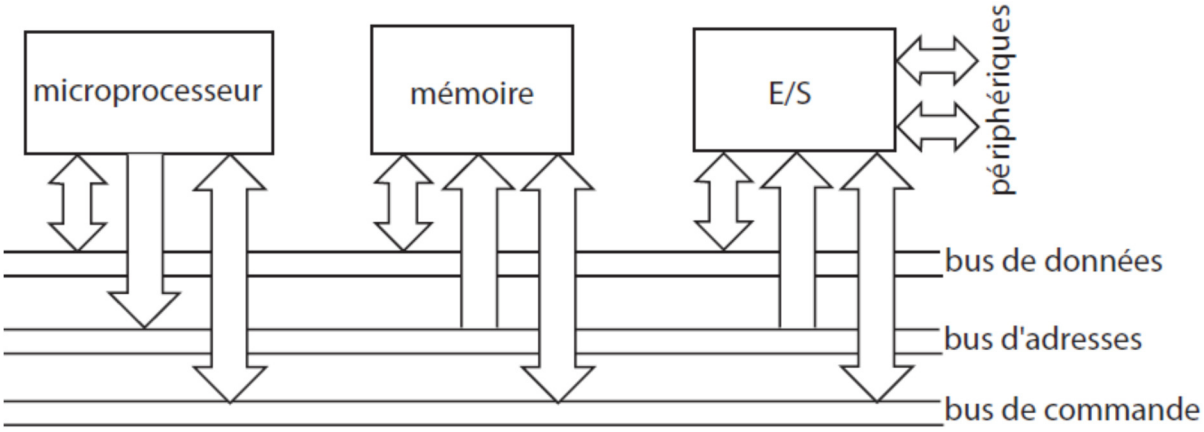


Figure III.12 Circulation de l'information dans un système à microprocesseur

III.3.3- Gestion de la mémoire par le 8086

Une mémoire est un composant électronique à base de semi-conducteurs permettant de stocker (enregistrer, conserver et restituer) des informations (instructions ou variables) sous forme binaire. Ces informations peuvent alors être écrites ou lues. Le 8086 possède 20 bits d'adresse, donc l'espace mémoire adressable par le 8086 est de 20^2 = 1 048 576 octets = 1 Mo. Cet espace est segmenté en 16 blocs de 64 Ko. Un segment est une zone mémoire de 64 Ko (65 536 octets) définie par son adresse de départ qui doit être un multiple de 16. L'adresse de la première case mémoire est 0000 0000 0000 0000 0000 et celle de la dernière casse est 1111 1111 1111 1111 1111. C'est pour ça la représentation des adresse en binaire est toujours à éviter et en essayant d'utiliser à partir de cet instant une représentation en hexadécimal, et notre 8086 peut donc adresser 1 Mo allant de 00000 à FFFFF.

Le problème qui se pose est comment représenter ces adresses au sein du microprocesseur puisque les registres ne font que 16 bits soit 4 digits au maximum en hexadécimal. La solution adoptée par Intel a été la suivante : La mémoire est segmentée en 16 blocs de 64 Ko et une adresse sur 20 bits est obtenue en combinant deux parties :

a- Le registre CS qu'on appelle registre segment, permet de stocker les 4 bits de poids fort donnant le numéro de segment de mémoire ;

b- Le registre IP fournit les 16 bits de poids faible donnant l'adresse à l'intérieur du segment de mémoire spécifié par CS (un déplacement ou offset (appelé aussi adresse effective) dans ce segment).

Une adresse se présente toujours sous la forme **segment: offset**

Tableau III.2 Représentation des adresses pour un microprocesseur 8086

Segment	Adresse de début	Adresse de la fin	Pointeur de segment
Segment 0	00000	0FFFF	00000
Segment 1	10000	1FFFF	10000
Segment 2	20000	2FFFF	20000
……	……	……	……
Segment 14	E0000	EFFFF	E0000
Segment 15	F0000	FFFFF	F0000

L'adresse mémoire est retrouvée selon la formule :

$$Adresse = (16 \times CS) + IP \qquad \text{(III.3)}$$

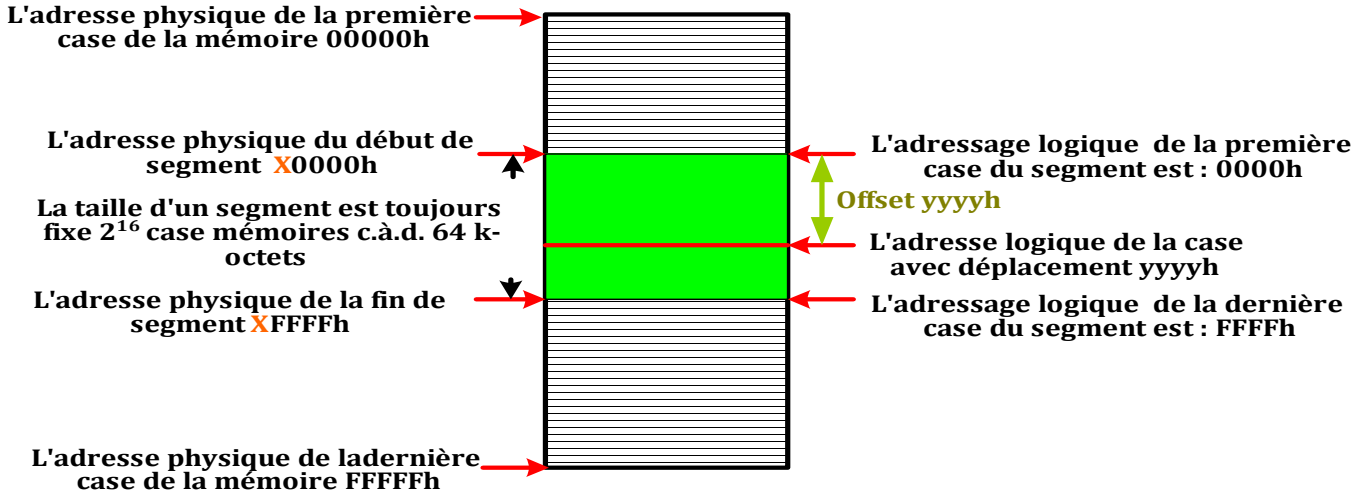


Figure III.13 Définition d'un segment de mémoire

L'adresse réelle d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits en hexadécimal) est appelée **adresse physique**. Cette adresse physique est générée et calculé par l'unité d'adressage (Figure III.14) de la manière illustrée par la figure suivante:

Bits	20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
	Base	0 0 0 0
+	0 0 0 0	Offset
=	Adresse physique	

Figure III.14 Relation entre les adresses logique et physique

Application :

Donner les adresses physiques de la première et la dernière case mémoire d'un segment définit par l'adresse 42A1h.

Réponse :

Le segment est définit par 42A1h (c'est l'adresse segment) :

c- L'adresse physique du début de segment :
= segment ×10h=42Ah×10h=42A10h

❖ L'adresse physique de la 1 ére case mémoire:
=Adresse début du segment +0000h

=42A10h+0000h=42A10h

- L'adresse physique de la dernière case mémoire:
  =Adresse début du segment +FFFFh
  =42A10h+FFFFh=52A0Fh

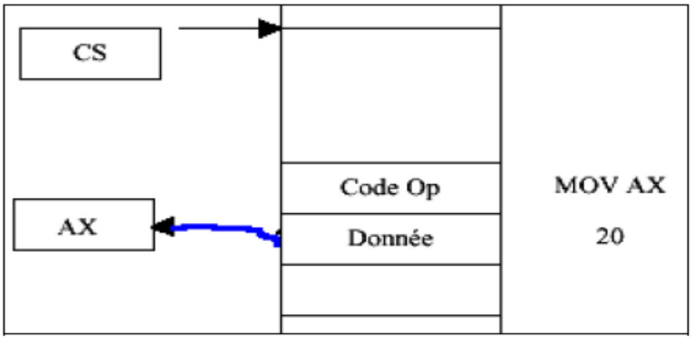
Donc les adresses physiques sont: **42A10h** et **52A0Fh**

III.3.4- Mode d'adressage

Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale. La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d'opérande. Pour les transferts de/vers la mémoire, il existe différentes façons de spécifier l'adresse d'une case mémoire dans une instruction: ce sont les modes d'adressage. Le mode d'adressage est la façon d'accéder à une donnée qui sera utile dans le programme. Il existe sept (07) modes d'adressage répartis en quatre grands types d'accès à une donnée :

III.3.4.a- Adressage immédiat

La donnée est fournie immédiatement avec l'instruction, elle est donc située dans le segment de code. MOV AX, 20



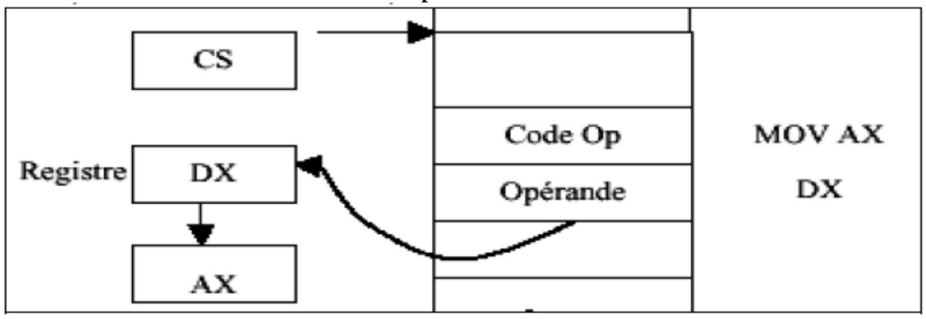
Exemple :

- MOV Alpha, 40h : Alpha est une case mémoire de 8 bits.
- MOV AL, 102h : Déclaration d'une erreur, AL registre de 8 bits, mais 102h est une valeur immédiate sur 16 bits.
- MOV AX, 40h : Cette instruction est juste. Par ce que, on peut écrire 40h ⇒ 0040h valeur sur 16 bits.

III.3.4.b- Adressage Registre

Ce mode d'adressage concerne tout transfert ou toute opération, entre deux registres de même taille. Dans ce mode l'opérande sera stocké dans un registre interne au microprocesseur.

MOV AX, BX : Cela signifie que l'opérande stocké dans le registre BX sera transféré vers le registre AX. Quand on utilise l'adressage registre, le microprocesseur effectue toutes les opérations d'une façon interne. Donc dans ce mode il n'y a pas d'échange avec la mémoire, ce qui augmente la vitesse de traitement de l'opérande.



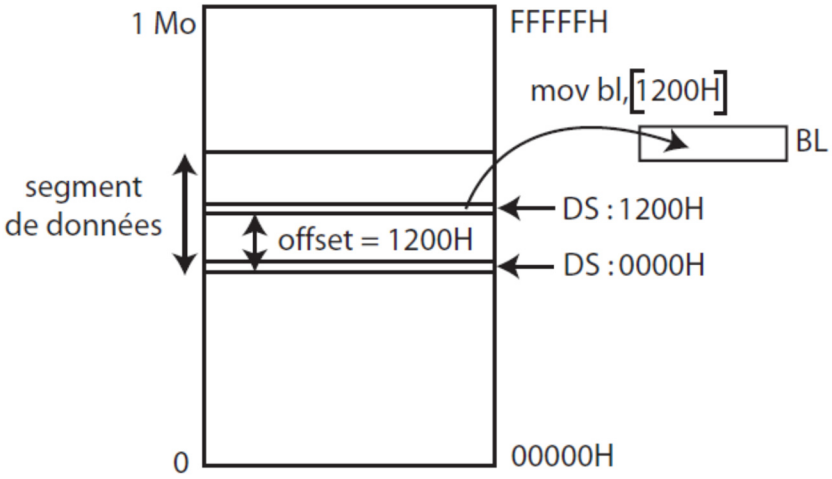
MOV BX, AX : Transfert du contenu de AX vers BX.
MOV BX, AL : Déclaration d'erreur par ce que BX est registre 16 bits et AL est registre 8 bits.

MOV BL, BH : Transfert du contenu de BH vers BL.

III.3.4.c- Adressage direct

Dans ce mode on spécifie directement l'adresse de l'opérande dans l'instruction. L'instruction comporte l'adresse de la case mémoire ou se trouve la donnée. L'adresse effective représente l'offset de la case mémoire dans le segment de données (segment dont l'adresse est contenu dans le registre DS).

MOV bl, [1200H] : Transféré le contenu de la case memoire d'adresse effective (offset) 1200H vers le registre BL.



MOV AL, Bêta : Transférer le contenu de la case mémoire Bêta vers AL.

MOV (2000h), BL : Transférer le contenu de BL vers la case mémoire de l'adresse 2000h par rapport au DATA segment.

III.3.4.d- Adressage indirect

On distingue quatre modes :

i. Adressage indirect par registre

C'est le mode d'accès entre la mémoire et les registres internes de µp par un intermédiaire, ces sont les registres pointeurs : (SI, DI, BX).

Exemple : MOV AX, [BX] : Charger AX par le contenu de la case mémoire pointée par BX.
MOV AX, [SI] : Charger AX par le contenu de la case mémoire pointée par SI.
MOV [SI], DL : Transfert de contenu de DL vers la case mémoire 8 bits pointer par SI par rapport au DATA segment.

ii. Adressage basé : tous simplement, il utilise les registres de base comme pointeur entre la mémoire et les registres internes de µp ; les registres de base sont (BX, BP).

Syntaxe :

- MOV [BX/BP + déplacement 8 bits ou 16 bits], source.
- MOV destination, [BX/BP + déplacement 8 bits ou 16 bits].

Exemple :

- MOV [BX+12], AL : Transfert de contenu de AL vers la case mémoire pointer par (BX+12) par rapport au DATA segment.
- MOV SI, [BP+400] : Transfert de contenu de la case mémoire 16 bits pointer par (BP+400) par rapport au STACK segment vers SI.

iii. Adressage indexé : le même mode que le basé, mais il utilise les registres d'indexes comme pointeur ; les registres d'indexes sont (SI, DI).

Syntaxe :

- MOV [SI/DI + déplacement 8 bits ou 16 bits], source.
- MOV destination, [SI/DI + déplacement 8 bits ou 16 bits].

- MOV al, [si] : Charge le registre AL avec le contenu de la case memoire dont l'offset est contenu dans SI ;
- MOV [di], bx : Charge les cases memoire d'offset DI et DI + 1 avec le contenu du registre BX.

Remarque : Une valeur constante peut éventuellement être ajoutée aux registres de base ou d'index pour obtenir l'offset.

Exemple :

MOV [si+100H], AX

Qui peut aussi s'écrire

MOV [si] [100H], AX

Ou encore

MOV 100H[si], AX

iv. Adressage indirect base indexée

Ce mode d'adressage utilise la somme du contenu de deux registres pour déterminer l'adresse effective.

Syntaxe :

- MOV [BX+ SI/DI + déplacement 8 bits ou 16 bits], source.
- MOV destination, [BX + SI/DI + déplacement 8 bits ou 16 bits].
- MOV [BP+ SI/DI + déplacement 8 bits ou 16 bits], source.
- MOV destination, [BP + SI/DI + déplacement 8 bits ou 16 bits].

Exemple :

MOV BX, 10
MOV SI, 15
MOV byte ptr matrice [BX] [SI] ,12H. Dans cet exemple, BX et SI jouent respectivement le rôle d'indices de ligne et de colonne dans la matrice.

III.3.5- Programmation en assembleur du microprocesseur 8086

La programmation en langage machine nécessite la connaissance au préalable des registres internes du processeur utilisé, ce type de langage n'est pas compréhensible par un humain, ou tout du moins pas directement, ce qui nécessite l'intervention d'un compilateur, chargé de traduire un langage de programmation "humain" (Basic, Java, C++, etc.) en langage machine composé d'arides 0 et 1. Il existe toutefois un langage intermédiaire classé dans la catégorie des langages "de bas niveaux" renommé par « Langage Assembleur », il est intimement lié au microprocesseur et aux circuits d'un ordinateur. En effet, en langage assembleur chaque instruction est définie par son code opératoire (valeur numérique binaire) et fait référence à une action élémentaire effectuée par le processeur. On utilise donc une notation symbolique pour représenter les instructions et les mnémoniques.

III.3.5.1- Instructions

Les instructions peuvent être classées en groupes :

- ✓ Instructions de transfert de données ;
- ✓ Instructions arithmétiques : addition (ADD), soustraction (SUB), multiplication (MUL), division (DIV), incrémentation (INC), décrémentation (DEC) et échange (XCHG) ;
- ✓ Instructions logiques et booléennes : et (AND), ou (OR) et non (NOT) ;
- ✓ Instruction de comparaison (CMP) : met à jour les flags pour permettre l'utilisation des instructions de saut ;
- ✓ Instructions de saut : saut si égal (JE), saut si différent (JNE), saut si inférieur (JL) ;
- ✓ Instructions de gestion de la pile : empilement (PUSH) et dépilement (POP)
- ✓ Instruction d'appel (CALL) et de retour (RET) ;
- ✓ Instructions de chaîne de caractères ;
- ✓ Instructions de contrôle de processus ;

- ✓ Instructions d'interruptions ;
- ✓ De nombreuses autres instructions que nous ne détaillerons pas ici.

Le modèle générale de la description d'une instruction est le suivant :

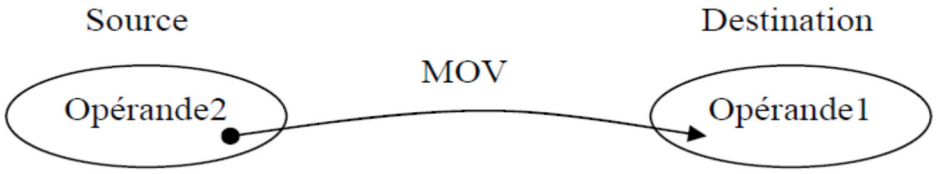
```
XXX   Opérande 1, Opérande 2, … ;   Commentaire
. . .
. . .
```

III.3.5.1.a- Instructions de transfert de données

Ces sont les instructions suivantes : MOV, PUSH, PUSHF, POP, POPF, XCHG, XLAT, LEA, LDS, LES, LAHF, SAHF, IN et OUT

- **Instruction MOV**

C'est l'instruction la plus courante, elle permet le transfert d'une donnée ou d'un mot entre un registre et une case mémoire ou entre deux registres, la source pouvant être une valeur immédiate.



La source peut être : Registre, mémoire, valeur immédiate (latérale) ;
La destination peut être : Registre, mémoire.

Exemples :

```
MOV destination, source
MOV AX, BX ; Transfert d'un registre de 16 bits vers un registre de 16 Bits
MOV AH, CL ; Transfert d'un registre de 8 bits vers un registre de 8 bits
MOV AX, Val1 ; Transfert du contenu d'une case mémoire 16 bits vers AX
MOV Val2, AL ; Transfert du contenu du AL vers une case mémoire d'adresse Val2
MOV AX, 77H ; adressage immédiat
MOV AX, [1064H] ; adressage direct
MOV BX, OFFSET TAB1 ; charger dans BX l'adresse de TAB1
```

Remarques

- Il est strictement interdit de transférer le contenu d'une case mémoire vers une autre, il faut passer par un registre interne de même taille.

  ```
  MOV Val1, Val2 ; c'est une instruction fausse.
  ```

  L'expression juste est la suivante :

  ```
  MOV AL, Val2
  MOV Val1, AL
  ```

- Il est strictement interdit de copier la valeur d'un registre de segment vers un autre registre de segment, il faut passer par une case mémoire ou par un registre interne de 16 bits ;

  ```
  MOV DS, ES; c'est une instruction fausse.
  ```

  L'expression juste est la suivante :

  ```
  MOV AX, ES
  MOV DS, AX
  ```

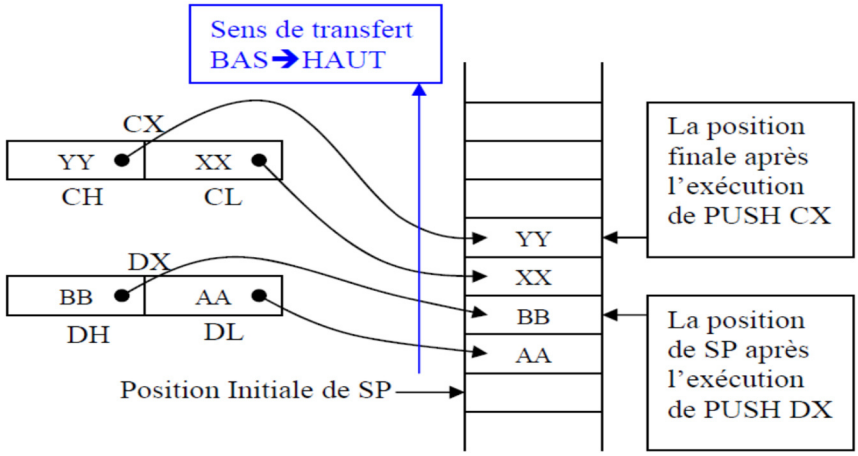
- Le CS n'est jamais utilisé comme registre de destination ;
- Interdit de lire ou écrire dans le registre (IP) ;
- Interdit de copier une valeur immédiate vers un registre de segment, il faut passer par un registre interne ou une case mémoire de 16 bits.

- **Instructions PUSH et POP**

L'instruction PUSH place la valeur spécifiée d'un opérande sur la pile (place la valeur du sa valeur au sommet de la pile.). L'instruction POP retire le contenu du sommet de la pile et le place dans l'opérande du POP.

Exemple 1 :

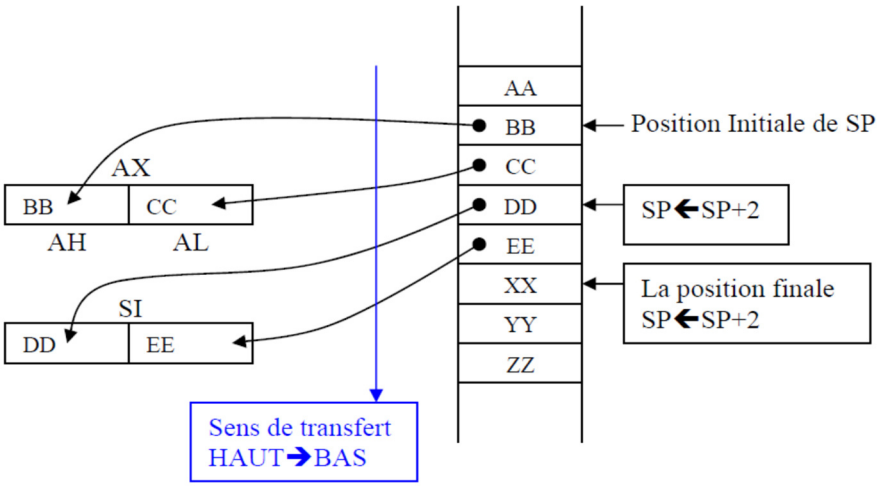
```
PUSH DX
PUSH CX
```



Après chaque exécution de l'instruction PUSH la valeur de SP décrémente automatiquement par deux (2) **SP←SP-2**.

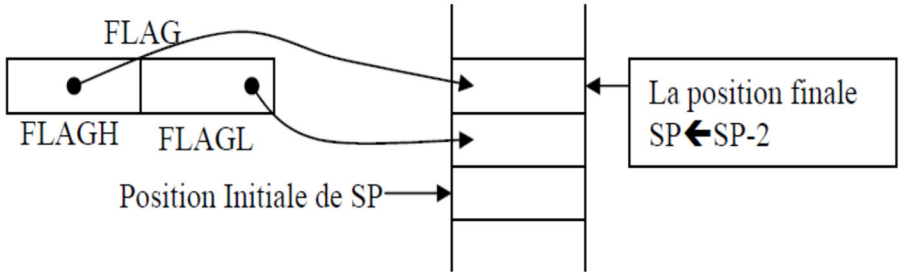
Exemple 2

```
POP AX
POP SI
```

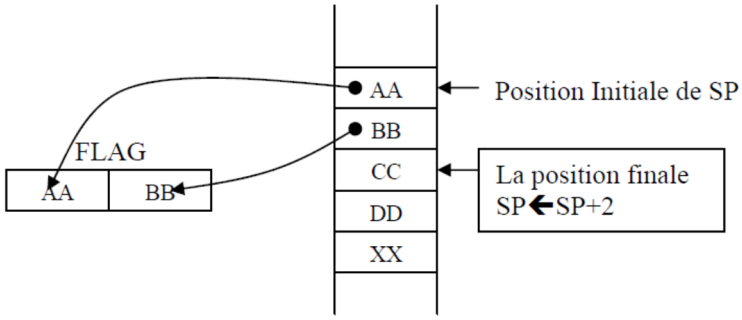


- **Instructions PUSHF et POPF**

PUSHF : Sauvegarde des flags sur la pile
POPF : Restauration des flags à partir de la pile
PUSHF et POPF, des instructions sans opérande.
PUSHF : Le destinataire est une case mémoire de 16 bits pointer par SS : [SP].



POPF : La source est une case mémoire de 16 bits pointer par SS : [SP].



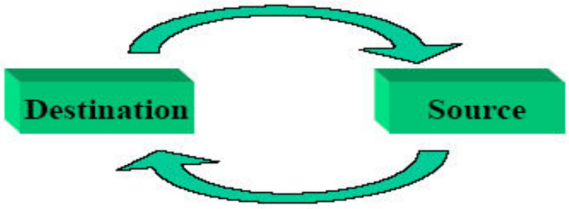
- **Instructions PUSHA et POPA**
  - PUSHA: Cette instruction permet d'empiler la totalité des registres internes du microprocesseur sur la pile ;
  - POPA : Cette instruction permet de dépiler la totalité des registres internes du microprocesseur sur la pile.

- **Xchange (XCHG)**

Cette instruction permet d'échanger le contenu de deux registres ou un registre et une case mémoire, mais pas pour les registres segments.

Exemple :

```
XCHG AX, BX ;
XCHG BX, CASE_MEM ;
```



XCHG AX,BX ; elle permute entre AX et BX
Exemple AX=0123 et BX = 5678
Après l'exécution de l'instruction on aura :
AX=5678 et BX = 0123

- **LEA**

Cette instruction charge l'adresse d'un opérande (offset) dans un registre, (en général, l'opérande est une donnée déclarée dans le segment de données).

Syntaxe : LEA Reg16, Mem16

Exemple :

`Alpha DB 40h` ; Déclaration d'une case mémoire dans le `DATA SEGMENT`, initialisé par `40h`

`LEA SI, Alpha ;` (1)

`MOV AL, Alpha ;` (2)

Dans (1) : SI contient l'adresse effective de la case Alpha.
Dans (2) : AL contient le continu de la case Alpha (AL←40h).

- **XLAT**

Cette instruction est utilisée pour convertir des données d'un code à un autre, en effet elle permet de placer dans l'accumulateur AL le contenu de la case mémoire adressée en adressage base +décalage (8 bits), la base étant le registre BX et le décalage étant AL lui-même dans le segment DS

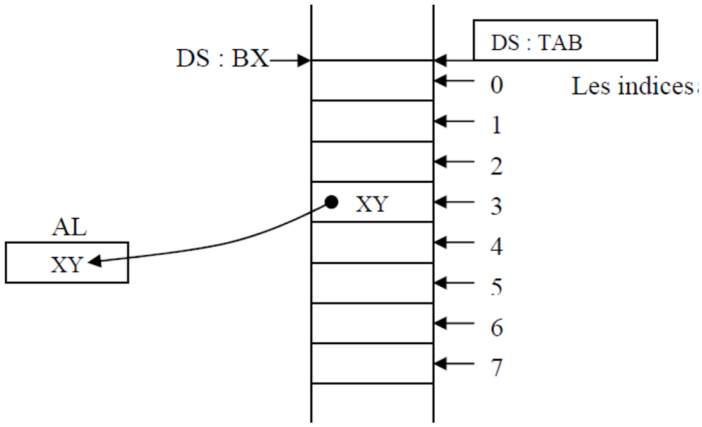
Syntaxe d'utilisation : On place l'adresse de la table dans le registre BX

MOV BX, OFFSET TABLE
On place l'indice de l'octet à consulter dans AL

MOV AL, indice ; 0 ≤ indice ≤ 255
XLAT; AL ← DS: [BX+AL].

Example 1:

```
MOV BX, OFFSET TAB
MOV AL, 03h
XLAT
```



Après l'exécution de l'instruction XLAT la valeur dans AL est écrasé par le contenu de la case mémoire pointer par DS : [BX+AL].

Exemple 2 :

Conversion du code binaire 4 bits en un digit hexa codé en ASCII

```
Tab db '0123456789ABCDEF'
MOV AL,1110B ; chargement de la valeur à convertir (14) MOV BX, OFFSET TAB
             ; pointé sur le tableau
XLAT ; Al est chargé par le code ASCII de 'E'
```

- **LDS et LES**
  - **LDS** : Chargement simultané du segment DS et l'offset d'une adresse

```
LDS BX, Table ; Charge dans BX offset Table dans le registre BX
              ; et le segment dans le registre DS
```

  - **LES** : Chargement simultané du segment ES et l'offset d'une adresse

```
LDS BX, Table ; Charge dans BX offset Table dans le registre BX
              ; et le segment dans le registre ES
```

- **IN et OUT**

Elle permet le transfert des données avec les périphériques d'E/S, c.à.d. de récupérer des données d'un port d'entrée (donc de la périphérie) ou restituer des données à un port de sortie, dans les deux cas s'il s'agit d'envoyer ou de recevoir un octet on utilise l'accumulateur AL, s'il s'agit d'envoyer ou de recevoir un mot on utilise l'accumulateur AX.

- Lecture d'un port d'E/S :

a- Si l'adresse du port d'E/S est sur un octet :

```
IN AL, adresse : lecture d'un port sur 8 bits ;
IN AX, adresse : lecture d'un port sur 16 bits.
```

b- Si l'adresse du port d'E/S est sur deux octets :

On va mettre l'adresse du port dans le registre DX.

```
MOV DX, adresse
IN  AL, DX : lecture d'un port sur 8 bits ;
IN  AX, DX : lecture d'un port sur 16 bits.
```

Où le registre DX contient l'adresse du port d'E/S à lire.

- Ecriture d'un port d'E/S

a- Si l'adresse du port d'E/S est sur un octet :

```
OUT adresse, AL : écriture d'un port sur 8 bits ;
OUT adresse, AX : écriture d'un port sur 16 bits.
```

b- Si l'adresse du port d'E/S est sur deux octets :

On va mettre l'adresse du port dans le registre DX.

- Ecriture d'un port sur 8 bits ;

```
MOV DX, adresse
MOV AL, data8
OUT DX, AL
```

- Ecriture d'un port sur 16 bits ;

```
MOV DX, adresse
MOV AX, data16
OUT DX, AX
```

Où le registre DX contient l'adresse du port d'E/S à écrire.

Exemples :

• Lecture d'un port d'E/S sur 8 bits à l'adresse 300H :

```
MOV DX, 300H
IN   AL, DX
```

• Ecriture de la valeur 1234H dans le port d'E/S sur 16 bits à l'adresse 49H :

```
MOV AX, 1234H
OUT 49H, AX
```

III.3.5.1.b- Instructions arithmétiques

Comme dans de nombreux processeurs, le 8086 possède les instructions de base sont l'addition, la soustraction, la multiplication et la division qui incluent diverses variantes. Plusieurs modes d'adressage sont possibles. Pour faire des opérations sur des données plus complexes (des nombres flottants par exemple), on devra les programmer. Les instructions arithmétiques peuvent manipuler quatre types de nombres :

a- Les nombres binaires non signés ;
b- Les nombres binaires signés ;
c- Les nombres décimaux codés binaires (DCB), non signés ;
d- Les nombres DCB non condensés, non signés.

1- Instructions d'addition

- **ADD** effectue l'addition du contenu du registre source au registre destination, sans report de retenue, soit ;

  ADD opérande1, opérande 2

  opérande1 ← opérande1 + opérande2.

Les FLAG affectés :

CF	OF	AF	SF	ZF	PF

Exemples:

```
MOV BL, a9h
MOV AL, 72h
ADD AL, BL
```

Alors le registre AL contient la valeur 1bh, le bit CF est positionne à la valeur 1, la retenue auxiliaire AF est mise à 0.

Si nous considérons les deux instructions suivantes ;

```
MOV AL, 09h
ADD AL, 3ah
```

Alors le registre AL contient la valeur 43h, le bit CF est mis à 0, la retenue auxiliaire AF est mise à 1.

- **ADC** effectue l'addition du contenu du registre source au registre destination avec report de retenue, soit :

  ADC operande1, opérande 2

  opérande1 ← opérande1 + opérande2 + retenue.

Exemples :

```
ADC AX, BX  ; AX = AX + BX + CF (addition sur 16 bits)
ADC AL, BH  ; AL = AL + BH + CF (addition sur 8 bits)
ADC AL, [SI] ; AL = AL + le contenu de la case mémoire pointé par SI + CF
ADC [DI],AL ; le contenu de la case mémoire pointé par DI
            ; est additionné avec AL + CF, le résultat est
            ; mis dans la case mémoire pointé par DI
```

Remarque :

- Pour les instructions ADD et ADC on a presque les mêmes restrictions de l'instruction MOV c.à.d. on n'a pas le droit d'additionner deux cases mémoires sans utiliser un registre de données.

2- Instructions de soustraction (SUB, SBB)

L'instruction **SUB** effectue la soustraction de l'opérande source de l'opérande destination et place le résultat dans l'opérande destination. Si un emprunt est nécessaire, le bit CF est mis à 1. L'instruction **SBB** tient compte de cet emprunt dans une soustraction de précision. Les FLAG sont affectés.

SUB destination, source : destination ← destination – source

```
SUB AX, BX ; AX ←AX-BX
```

Exemple:

```
MOV BL, 18h
MOV AL, 39h
SUB AL, BL
```

Le registre AX contient ensuite la valeur 21h. Les bits ZF, SF et CF du registre FLAGS sont mis à 0 car le résultat n'est pas nul, son signe est positif et aucune retenue n'est générée.

```
MOV AL, 26h
SUB AL, 59h
```

Le registre AL contient ensuite la valeur cdh. Le bit ZF est mis à zéro et les bits CF, AF et SF sont mises à 1.

SBB destination, source : destination ←destination – source - retenue

```
SBB AX, BX ; AX ←AX-BX –CF
```

Remarque :

- On a les mêmes restrictions de l'instruction ADD.

3- Instructions de multiplication

Il y a deux instructions pour la multiplication: **MUL**, qui effectue la multiplication d'opérandes non signes (toujours positives) et **IMUL**, qui IMUL effectue la multiplication d'opérandes signes et considère les opérandes comme des quantités signées en complément à 2 en binaire et en complément à 15 en hexadécimal, plus un (+1) pour les deux codage .

Les indicateurs de retenue (CF) et de débordement (OF) sont mis à un si le résultat ne peut pas être stockée dans l'opérande destination.

Ces instructions effectuent la multiplication du contenu de AL par un opérande sur 1 octet ou du contenu de AX par un opérande sur 2 octets. Le résultat est place dans AX si les données à multiplier sont sur 1 octet (résultat sur 16 bits), dans (DX, AX) si elles sont sur 2 octets (résultat sur 32 bits).

Examples

```
MOV AL, 51
MOV BL, 32
MUL BL
→ AX = 51 × 32

MOV AX, 4253
MOV BX, 1689
MUL BX
→ (DX, AX) = 4253 × 1689
```

Pour bien comprendre la différence entre les instructions **IMUL** et **MUL**, regardons les exemples suivants :

```
MOV BX, 435
MOV AX, 2372
IMUL BX
```

A l'issue de l'exécution de ces 3 instructions, AX contient la valeur BE8C et DX la valeur F, soit la valeur hexadécimale FBE8C, c'est à dire 1031820, le produit de 435 par 2372. Les deux données étant positives, le résultat est le même que l'on utilise l'instruction IMUL ou l'instruction MUL. Considérons maintenant la séquence d'instructions :

```
MOV BX, - 435
MOV AX, 2372
IMUL BX
```

A l'issue de leur exécution, AX contient la valeur 4174 et DX contient la valeur FFF0, soit la valeur -1031820.

Si l'on remplace IMUL par MUL, le résultat n'a pas de sens.

4- Instructions de division

Comme pour la multiplication, il y a deux instructions pour la division, **DIV** pour la division non signée, et **IDIV** pour la division signée. Ces instructions effectuent la division du contenu de AX par un opérande sur 1 octet ou le contenu de (DX, AX) par un opérande sur 2 octets. **Résultat** : si l'opérande est sur 1 octet, alors AL = quotient et AH = reste ; si l'opérande est sur 2 octets, alors AX = quotient et DX = reste.

Exemples

```
MOV AX, 55
MOV BL, 10
DIV BL
```

→ AL = 5 (quotient) et AH = 5 (reste)

```
MOV DX, 0
MOV AX, 4325
MOV BX, 10
DIV BX
```

→ AX = 432 (quotient) et DX = 5 (reste)

```
MOV DX, 0000h
MOV AX, 4000h
MOV CX, 0002h
DIV CX      ; AX ← (DX: AX) / CX
            ; AX ← (00004000h)/2=2000h.
            ; DX←0000h.
```

```
MOV DX, 0000
MOV BX, -435
MOV AX, 2372
IDIV BX
```

On aura ensuite AX qui contient la valeur -5 (soit FFFB en hexadécimal) et DX la valeur 197. Si l'on remplace IDIV par DIV, le résultat n'a pas de sens.

5. Instructions d'incrémentation et de décrémentation

Ces deux instructions sont fréquemment utilisées et représentent des cas particuliers des instructions d'addition et de soustraction, en ajoutant (INC) 1 ou retirant (DEC) 1 pour un registre ou un emplacement. La destination peut être : registre, mémoire.

Exemples

```
MOV AX, 2372
INC AX ; AX = AX + 1 ⇒ AX=2373 (incrémentation sur 16 bits).
MOV AX, 3fh
INC AL ; AL = AL +1 ⇒ AL=40h (incrémentation sur 8 bits).
```

```
INC [SI] ; [SI] = [SI] + 1 le contenu de la case mémoire pointé
         ; par SI sera incrémenté
MOV Alpha, 0Fh
INC Alpha ; Alpha ← Alpha+1 ⇒ Alpha=10h.
MOV AX, 2372
DEC AX ; AX = AX – 1 ⇒ AL=2371 (décrémentation sur 16 bits).
MOV AX 3Fh
DEC AL ; AL = AL -1 ⇒ AL=3Eh (décrémentation sur 8 bits).
DEC [SI] ; [SI] = [SI] - 1 le contenu de la case mémoire
         ; pointé par SI sera décrémenter
```

6. Instruction de négation par complément à 2

NEG transforme la valeur d'un registre ou d'un opérande memoire en son complément à deux, pour cela, elle inverse d'abord tous les bits, et ajoute 1. D'une autre façon c'est de soustraire l'opérande destination (octet ou mot) de 0 le résultat est stocker dans la destination.

Les indicateurs de FLAG affectés par cette opération sont : AF, CF, OF, PF, SF, ZF.

Exemple :

```
MOV AL, 05h
NEG AL; AL ← AL*(-1)
```

0	0	0	0	0	1	0	1

Inversion bit à bit

1	1	1	1	1	0	1	0

Addition d'une unité

1	1	1	1	1	0	1	1

AL = FBh la représentation de (-5) en complément à 2.

```
MOV AL,25
MOV BL,12
NEG BL
ADD AL,BL ; → AL = 25 + (-12) = 13
```

7. Instruction de comparaison (CMP)

La comparaison entre deux opérandes, soustrait les deux opérandes. Le résultat de la soustraction n'est pas sauvé, seuls les indicateurs sont modifiés.

CMP destination, source ; Résultat ← **destination – source.**

CMP Op1, Op2 ; Résultat ← **Op1 – Op2.**

Résultat est un registre virtuel, mais le plus important, c'est la mise à jour des FLAG.

	Opérande non signé				Opérande signé			
	OF	SF	ZF	CF	OF	SF	ZF	CF
Source < destination	-	-	0	0	0/1	0	0	-
Source = destination	-	-	1	0	0	0	1	-
Source > destination	-	-	0	1	0/1	1	0	-

Exemple :

```
MOV AL, 61h
MOV AH, 47h
CMP AL, AH
```

La mise à jour des FLAG : CF←0 ; ZF←0. Donc, on peut dire que AL plus grand que AH.

III.3.5.1.c- Instructions logiques

Ce sont des instructions qui permettent de manipuler des données au niveau des bits, et non sur des valeurs numériques comme les instructions vues jusqu'à présent).

1. AND (et logique)

Cette instruction est l'équivalant d'une porte ET logique, elle réalise un ET logique bit à bit entre l'opérande source et l'opérande destination. Le résultat est placé dans le premier opérande. Application : masquage de bits pour mettre à zéro certains bits dans un mot.

AND Opérande1, Opérande2 ; operande1 ← operande1 ET operande2

Les FLAGs affectés :

ZF	SF	PF

Exemples :

```
MOV AL, 10110011B
MOV BL, 11101101B
AND AL, BL
```

```
MOV AL, 36H
AND AL, 5CH
```

36h = 0011 0110
5Ch = 0101 1100
AL= 0001 0100

```
AND AX, BX ; AX = AX AND BX (Et logique entre AX et BX)
AND AL, BH ; AL = AL AND BH (ET logique sur 8 bits)
AND AL, [SI] ; AL = AL AND le contenu de la case mémoire
             ; pointé par SI
AND [DI], AL ; ET logique entre la case mémoire pointé par DI et AL,
             ; le résultat est mis dans la case mémoire pointé par DI
```

2. OR (OU logique)

OR réalise un ou-logique bit à bit entre l'opérande source et l'opérande destination. Le résultat est rangé dans l'opérande destination. La destination peut être : registre, mémoire. La source peut être : registre, mémoire, immédiate. Application : mise `à 1 d'un ou plusieurs bits dans un mot.

OR operande1, operande2. L'opération effectuée est :

opérande1 ← opérande1 OU opérande2.

Exemples :

```
MOV AL, 00110001B
MOV BL, 11011100B
OR  AL, BL
```

AL = 0011 0001
Bl = 1101 1100
AL= 1111 1101

```
MOV AL, 36h
OR  AL, 5Ch
```

36h = 0011 0110
5Ch = 0101 1100
AL = 0111 1110

```
OR AX, BX ; AX = AX OR BX (OU logique entre AX et BX)
```

```
OR AL, BH ; AL = AL + BH (OU logique sur 8 bits)
OR AL, [SI] ; AL = AL OU le contenu de la case mémoire pointé par SI
OR [DI], AL ; OR logique entre la case mémoire pointé par DI et AL,
            ; le résultat est mis dans la case mémoire pointé par DI
```

3. OU exclusif (XOR)

XOR réalise un ou-exclusif bit `a bit entre l'opérande source et l'opérande destination.

XOR Destination, Source ; Destination ← destination (ou exclusif) source.

Flags modifiés: CF - OF - PF - SF - ZF

Exemples :

```
MOV AL, 36h
XOR AL, 5Ch
```

36h = 0011 0110
5Ch = 0101 1100
AL = 0110 1010

```
XOR AX, BX ; AX = AX ⊕ BX (OU exclusif entre AX et BX)
XOR AL, BH ; AL = AL ⊕ BH (OU exclusif sur 8 bits)
XOR AL, [SI]; AL = AL OU exclusif le contenu de la case Mémoire
           ; pointé par SI
XOR [DI], AL ; XOR logique entre la case mémoire pointé par DI et AL,
             ; le résultat est mis dans la case mémoire pointé par DI
```

4. Instruction NOT (Négation) :

NOT transforme la valeur d'un registre ou d'un opérande mémoire en son complément logique bit à bit.

NOT Destination ; Destination ← destination.

Exemples :

```
MOV AX, 500; AX = 0000 0101 0000 0000
NOT AX; AX = 1111 1010 1111 1111
MOV AL, 36h; 36h 0011 0110
NOT AL       ; C9h 1100 1001
MOV AL, 45h
NOT AL; AL←0BAh.
```

5. Instruction TEST

Cette instruction va effectuer un ET logique, mais le résultat ne sera pas gardé, seul les indicateurs seront modifiés.

Flags modifiés : CF - OF - PF - SF - ZF

TEST Opérande1, Opérande2

Exemple :

```
MOV AL, 0F2h
MOV BL, 3Dh
TEST AL, BL
```

F2h 1111 0010
(and)
3Dh 0011 1101
= 30h 0011 0000

ZF=0, SF=0, PF=1, CF=0, OF=0, AF=X.

III.3.5.1.d- Instructions de décalages et de rotations

Ces instructions permettent le déplacement des bits d'un mot ou d'un octet d'une position vers la gauche ou vers la droite.

Dans les décalages, les bits qui sont déplacés sont remplacés par des zéros, soit pour le décalage logique ou le décalage arithmétique à l'exception le décalage arithmétique vers la

droite, qui permet de déplacer tous les bits de 1 bit vers la droite, et la position laissée libre à gauche est remplacée par le bit le plus significatif de l'opérande, le bit de signe. Ce décalage maintient donc le signe de l'opérande.

1. **Décalage logique/ arithmitique vers la gauche : SHL** (Shift Left)/ **SAL** (Shift Arithmetic Left):

Effectue un décalage de n bits vers la gauche, les bits sortant sont insérés dans CF, les bits entrant sont mis à 0.

MOV Destination, opérande1
MOV CL, N
SHL Destination, CL
MOV Destination, opérande2
MOV CL, N
SAL Destination, CL

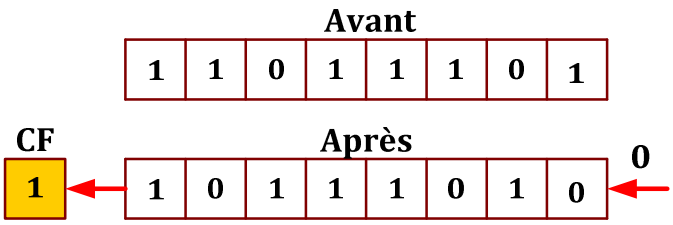
Remarque : Le nombre de bits à décaler (n) doit être place dans le registre CL ou CX, s'il est supérieur à 1 (n >1).

Exemples :

```
MOV CL, 1
MOV AL, 11011101B
SHL AL, CL
```

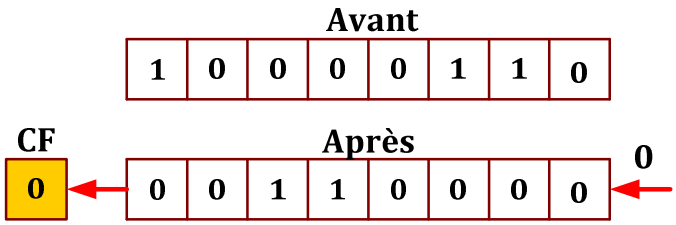
Dans ce cas n=1, donc on peut utiliser expression suivante:

```
MOV AL, 11011101B
SHL AL, 1; AL ← 10111010
```



Astuce :
Cette instruction réalise l'opération suivante : Destination ← destination * 2^N.

```
MOV AL, 86h
MOV CL, 3
SAL AL, CL; AL ← 30h
```



Au début, 86h s'écrit 10000 0110 en binaire. Si on décale cette valeur de trois positions vers la gauche, on obtient 0011 0000, soit 30h. Le bit CF est mis à 0. Le registre AL contient ensuite la valeur 30h.

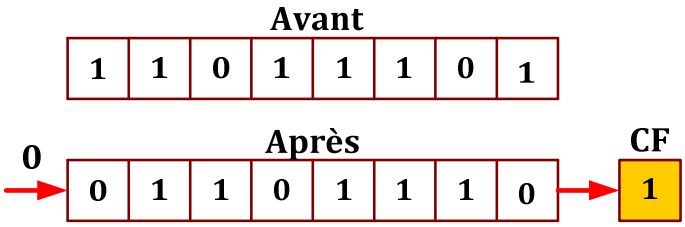
2. **Décalage logique vers la droite** SHR (Shift Right) :

Cette instruction décale à droite de n positions, n contenu dans le deuxième opérande, les bits entrant sont mis à 0, les bits sortant sont placés successivement dans CF, sans conservation, celui- ci prend donc la valeur du dernier bit sorti.

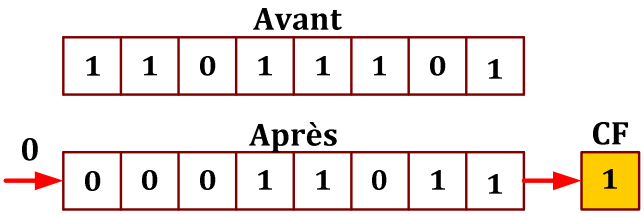
Remarque : Le nombre de bits à décaler (n) doit être place dans le registre CL ou CX, s'il est supérieur à 1 (n >1).

Exemple :

```
MOV AL, 11011101B
SHR AL, 1; AL ← 10111010
```




```
MOV CL, 3
MOV AL, 11011101B
SHR AL, CL
```

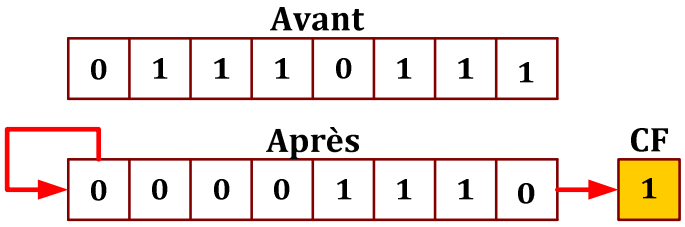


3. Décalage arithmétique vers la droite SAR (Shift Arithmetic Right)

Cette instruction effectue un décalage vers la droite, le nombre de positions à décaler est inscrit dans le second opérande. Le bit sortant est mis dans CF, et le bit rentrant varie en fonction du signe du nombre au départ; si au départ le nombre était positif, tous les bits entrant seront de 0, sinon, l'inverse. Comme le carry (CF) prend successivement les valeurs des bits sortant, après cette instruction CF est égal au dernier bit sorti.

Exemple :

```
MOV CL, 3
MOV AL, 77h
SAR AL, CL ; AL ← 0Eh et CF ← 1
```



Cette opération réalise aussi la fonction suivante : Destination ← destination / 2^N (signé)

Les instructions de rotation **ROL**, **RCL**, **ROR** et **RCR**, se comportent comme des décalages, mais où le bit sortant, en plus d'être envoyé dans CF est rappliqué à l'autre bout du registre. Elles considèrent un opérande (octet ou mot) comme un tore dont elles décalent les bits à gauche ou à droite, selon le sens de la rotation.

4. Instructions ROL (Rotate Left) et ROR (Rotate Right):

ROL : Les bits sont décalés vers la gauche, 1 ou CL fois, le bit de poids fort sortant est injecté dans CF et dans le bit de poids faible.

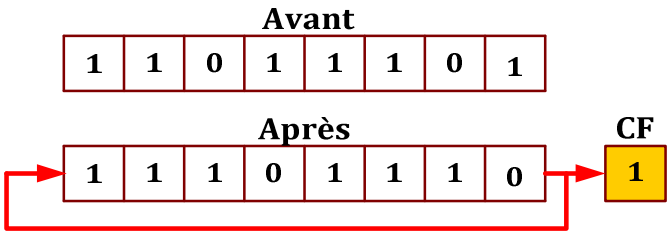
Exemple :

```
MOV CL, 2
MOV AL, 10000110B
ROL AL, CL ; AL ← 00011010B et CF ← 0
```

ROR : Les bits sont décalés vers la droite, 1 ou CL fois, le bit de poids faible sortant est injecté dans CF et dans le bit de poids fort.

Exemple :

```
MOV AL, 11011101B
ROR AL, 1; AL ← 11101110B et CF ← 1
```

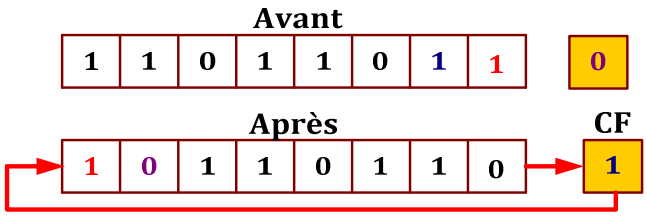


5. **Instructions RCR** (Rotate trought Cary Right) **RCL** (Rotate trought Cary Left)

RCR : Cette instruction décale les bits de l'opérande de n positions vers la droite. Le bit contenu de CF est met dans le bit le plus fort et le CF prend la valeur du bit le plus faible sortant.

Exemple

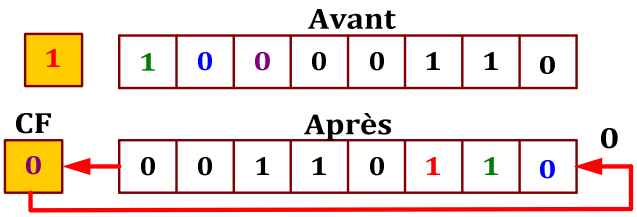
```
MOV AL, DBh
MOV CL, 2
RCR AL, CL; Al ← B6h et CF ← 1
```



RCL : Cette instruction décale les bits de l'opérande de n positions vers la gauche. Le bit contenu de CF est met dans le bit le plus faible et le CF prend la valeur du bit le plus fort sortant.

Exemple :

```
MOV AL, 86h
MOV CL, 3
RCL AL, CL; Al ← 36h et CF ← 0
```



III.3.5.1.e- Instructions de saut et de branchement

Il s'agit d'instructions qui altèrent le déroulement normal du programme. On distingue les sauts et les branchements.

Les sauts provoquent un branchement conditionnel ou inconditionnel du programme vers une adresse mémoire qui n'est pas contiguë à l'endroit où l'on se trouve. L'exécution du programme continuera à l'adresse du saut.

Les branchements provoquent un saut vers un sous-programme. La grande différence par rapport au saut, c'est qu'au moment du branchement il faut mémoriser l'adresse d'où l'on vient, afin de pouvoir y revenir une fois le sous-programme terminé. La mémorisation de l'adresse de départ se fait par l'intermédiaire d'un registre interne du processeur appelé souvent stack pointé ou pile.

1. **Instruction de saut inconditionnel** : JMP (JMP cible)

Cette instruction effectue un saut (**jump**) sans condition. Le saut peut être sur un label (représentation symbolique d'une instruction en memoire), une adresse mémoire, ou un registre. Flags modifiés : Aucun

```
                ⋮   ← instructions précédant le saut
          jmp suite
                ⋮   ← instructions suivant le saut (jamais exécutées)
suite :   ...       ← instruction exécutée après le saut
```

Exemple : JMP AX ; IP ← AX
JMP Boucle

2. **Instructions de sauts conditionnels**

Ces instructions permettent de sauter "une partie d'un programme" si une condition est vérifiée. Sinon l'exécution se poursuit séquentiellement à l'instruction suivante. La condition du saut dépend de l'état de l'un (ou plusieurs) des indicateurs d'état (flags) du microprocesseur. Les tableaux suivants donnent la liste des instructions de saut sur test des nombres signés et non signés.

Figure III.3 Instructions de sauts conditionnels

Instructions	Conditions de saut
JC	Saut en cas de retenue (Si CF=1 alors IP = IP + déplacement)
JE/JZ	Si Saut si égal / si nul (Si ZF=1 alors IP = IP + déplacement)
JNC	Si pas de retenue (Si CF=0 alors IP = IP + déplacement)
JNE/JNZ	Saut si non égal / si non nul (si (ZF = 0 alors IP = IP + déplacement).
JNO	Saut si pas de débordement, (Si OF=0 alors IP = IP + déplacement.
JNP/JPO	Si pas de parité/ s'il y'a de parité impair, (Si PF=0 alors IP = IP + déplacement)
JNS	Saut si le signe est positif (Si SF=0 alors IP = IP + déplacement)
JO	Saut s'il y'a de débordement, (Si OF=0 alors IP = IP + déplacement)
JP/JPE	Saut s'il y'a de parité/ s'il y'a de parité paire (Si PF=1 alors IP = IP + déplacement)
JS	Saut si le signe est négatif (Si SF=1 alors IP = IP + déplacement)

NB: les indicateurs sont positionnés en fonction du résultat de la dernière opération.

Exemple :

```
                ⋮   ← instructions précédant le saut conditionnel
          jnz suite
                ⋮   ← instructions exécutées si la condition ZF = 0 est vérifiée
suite :   ...       ← instruction exécutée à la suite du saut
```

Remarque : il existe un autre type de saut conditionnel, les sauts arithmétiques. Ils suivent en général une comparions de deux données par l'instruction CMP : CMP operande1, operande2. Ce sont des tests d'ordre (inferieur, supérieur, ...), comme les montre le tableau III.4 suivant :

Figure III.4 Instructions de sauts conditionnels qui suivent instruction CMP

Condition	Nombres signes	Nombres non signes
=	JEQ label	JEQ label
>	JG label	JA label
<	JL label	JB label
≠	JNE label	JNE label

Exemple :

```
                CMP AX, BX
                JG superieur
                JL inferieur
superieur:      ..........

inferieur :     ..........
```

- **Exemple d'application des instructions de sauts conditionnels** : on veut additionner deux nombres signes N1 et N2 se trouvant respectivement aux offsets 1100H et 1101H. Le résultat est rangé à l'offset 1102H s'il est positif, à l'offset 1103H s'il est négatif et à l'offset 1104H s'il est nul :
  - **Organigramme**

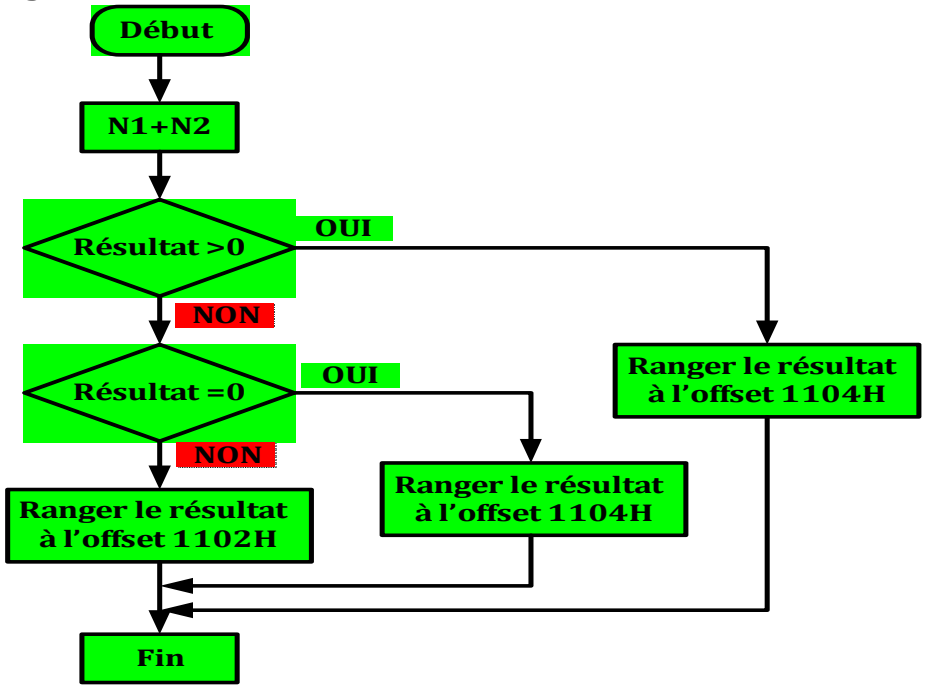


Figure 15 Organigramme d'addition et de stockage de deux nombre

  - **Programme**

```
                mov     al,[1100H]
                add     al,[1101H]
                js      negatif
                jz      nul
                mov     [1102H],al
                jmp     fin
negatif :       mov     [1103H],al
                jmp     fin
nul :           mov     [1104H],al
fin :           hlt
```

III.3.5.1.f- Instructions d'appelle sous-programme

La notion de sous-programme ou de procédure en assembleur correspond à celle de fonction en langage C par exemple, leurs rôles se sont identiques, ils consistent à éviter la répétition d'une même séquence d'instructions plusieurs fois dans un programme. Il suffit alors de rédiger la séquence une seule fois en lui attribuant un nom (au choix) et on l'appelle lorsqu'on en a besoin. Pour cela on utilise l'instruction CALL pour appeler le sous-programme et l'instruction RET pour effectuer le retour à l'appelant.

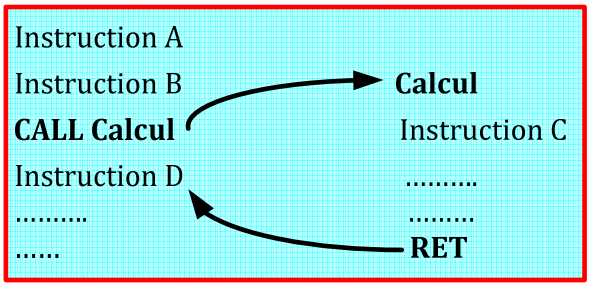
- **CALL**

Cette instruction permet l'appel d'un sous-programme (procédure) que celui-ci dépende du même segment ou non.

- Un CALL intra-segment (CALL NEAR) sauvegarde dans la pile le contenu de IP (IP : changé, CS : inchangé) ;
- Un CALL inter-segment (CALL FAR) sauvegarde dans la pile le contenu de CS et IP (IP : changé, CS : changé).

Une procédure se termine toujours par l'instruction RET (Return) qui rend le contrôle au sous-programme appelant. Pour cela, le sommet de pile est dépile dans le registre IP. L'adresse de retour doit donc nécessairement se trouver en sommet de pile.

Dans l'exemple ci-dessous la procédure est nommée calcul. Après l'instruction B, le processeur passe à l'instruction C de la procédure, puis continue jusqu'à rencontrer RET et revient à l'instruction D.



Pour mieux comprendre le principe de fonctionnement des instructions CALL et RET, on prend l'exemple ci-dessous :

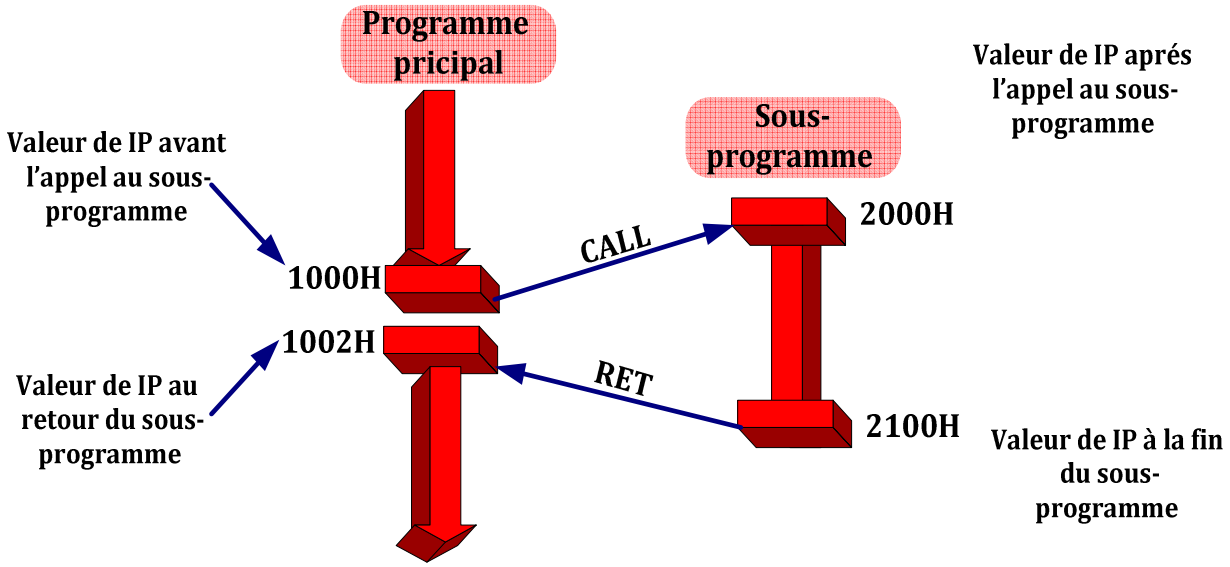
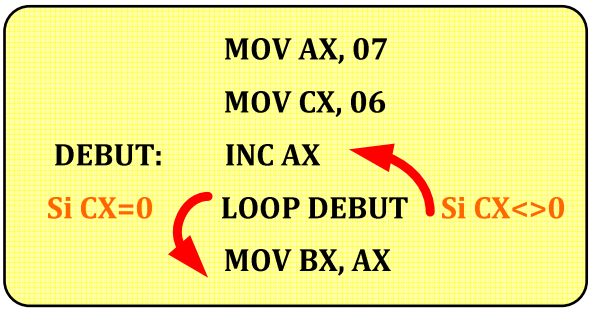


Figure III.16 Principe de fonctionnement des instructions CALL et RET

III.3.5.1.g- Instructions de boucle LOOP

Cette instruction fonctionne avec le registre CX qui joue le rôle de compteur de boucles, une répétition est effectuée tant que CX n'est pas égal à zéro. Elle décrémente le compteur sans modifier aucun des indicateurs.

Exemple :

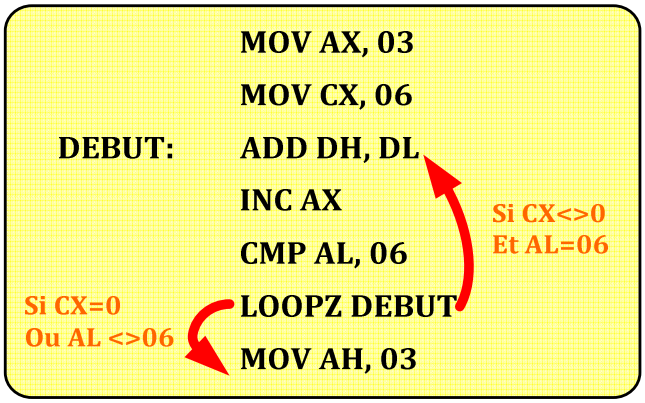


L'exécution de l'instruction MOV BX, AX sera faite après l'exécution de la boucle 5 fois.

- **LOOPE / LOOPZ**

Cette instruction décrémente le contenu de CX, et provoque un saut si le flag ZF vaut 1 ET le contenu de CX n'est pas nul (on sort de la boucle si ZF = 0 ou CX =0).

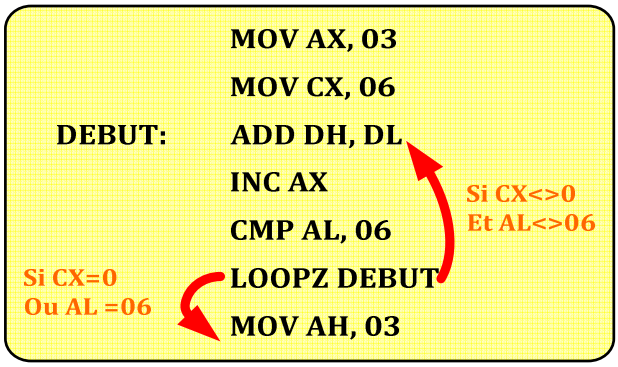
Exemple :



- **LOOPNE / LOOPNZ**

Cette instruction décrémente le contenu de CX, et provoque un saut si le flag ZF est nul ET le contenu de CX n'est pas nul (on sort de la boucle si ZF = 1 ou CX =0).

Exemple :



III.3.5.2- Interruptions

Un système à base de microprocesseur peut comporter plusieurs périphériques tels que: un Ecran, une souris, une imprimante, un disque dure, un CNA (convertisseur numérique analogique), un CAN etc. Il y a trois méthodes possibles pour échanger les données entre le microprocesseur et les périphériques : **scrutation périodique** (ou **polling**), **communication directe** entre deux périphériques (DMA : Direct memory acces) et **interruption** (ce que nous intéresse dans notre cours).

Comme son nom l'indique, une interruption provoque l'arrêt du programme principal pour aller exécuter une procédure d'interruption (sous-programme). A la fin de cette procédure, le microprocesseur reprend le programme à l'endroit où il s'était arrêté et ce par enregistrement de l'adresse de retour dans la pile du système par le CP.

On peut distinguer deux sortes d'interruptions. Les **interruptions matérielles** et les **interruptions logicielles**. Les deux sont traitées de la même façon. La différence tient à ce que les interruptions matérielles peuvent se produire n'importe quand, tandis que les interruptions logicielles se produisent à un endroit précis du code où se trouve une des instructions **int** ou **into**. De plus, les interruptions matérielles peuvent être masquées (interdites ou autorisées) à l'aide de l'indicateur IF.

III.3.5.2.a- Interruptions logicielles

Les interruptions logicielles, sont aussi appelées trappes ou déroutements. Elles incluent aussi les fautes et les arrêts. Une faute se produit quand le processeur détecte une erreur durant le traitement d'une instruction. Par exemple, division par 0, ou code invalide, etc. Quand une erreur est grave au point qu'une partie du contexte d'exécution est perdue, le résultat est un arrêt.

Les instructions des interruptions logicielles sont:

- Instruction INT: Appel un sous-programme qui s'appelle ISR (INTERRUPT SERVICE ROUTINE) ;
- Instruction INTO: Déclenche l'interruption de vecteur numéro « 4 » si OF=1 ;
- Instruction IRET: la routine d'interruption se termine toujours par l'instruction IRET (Interrupt Return).

La syntaxe de l'appel d'une interruption logicielle (Instruction INT) en langage symbolique pour le microprocesseur 8086/8088 est tout simplement :

```
INT constante
```

Où **constante** est un entier compris entre **00h** et **FFh**. Il y a donc 256 interruptions possibles pour le microprocesseur 8086.

Les cinq premières interruptions montrées dans le tableau III.5 suivant, sont définies par Intel. Les autres interruptions sont définies par le DOS et le BIOS.

Tableau III.5 Les cinq premières interruptions définies par Intel

Interruption	Adresses	Fonction
INT 00h	00 à 03h	Division par zéro. Le gestionnaire est appelé lorsqu'un essai de division par zéro est tente
INT 01h	04 à 08h	Etape par étape. Cette interruption est utilisée par debug et d'autres d'débogueurs pour exécuter un programme ligne par ligne
INT 02h	09 à 0Bh	Interruption non masquable (NMI)
INT 03h	0Ch à 0Fh	INT 03h : Point d'arrêt. Cette interruption est utilisée par debug et d'autres débogueurs pour arrêter l'exécution d'un programme
INT 04h	10h à 13h	INT 04h : Dépassement de capacité

Exemple

1- En citant quelques interruptions dont la fonctionnalité est définie par le BIOS:

- INT 05h : Impression de l'écran ;
- INT 08h : Temporisateur du système ;
- INT 09h : Interruption du clavier avec de nombreuses fonctions ;
- INT 10h : Interruption avec de nombreuses fonctions concernant le moniteur ;
- INT 11h : identification de la configuration du PC, utilisée en particulier lors du démarrage de l'ordinateur ;
- INT 17h : sortie sur l'imprimante ;
- INT 1Ah : lit le temps (date et heure) ou le met à jour ;
- INT 1Bh : prend le contrôle lors d'une interruption (CTRL-break) effectuée au clavier.

2- Pour les interruptions du DOS, l'appel se fait via l'instruction int 21h. Le registre AH contient un numéro qui référence la fonctionnalité que l'on veut utiliser (9 pour afficher une chaine de caractères, 1 pour saisir la frappe d'un caractère au clavier, ...)

- 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)
```

Renvoie dans le registre AL le code du caractère lu au clavier.

- Saisie d'un caractère (Sans écho)
```
MOV AH, 8
INT 21H ; (résultat dans AL)
```

Lit un caractère au clavier et le renvoie dans le registre AL. Ce caractère n'est pas affiche à l'écran.

- Arrêt du programme
```
MOV AH, 4CH ; A mettre à la fin de chaque programme
INT 21H;
```

III.3.5.2.b- Interruptions matérielles

Sont générées par les périphériques : souris, clavier, disque, horloge temps réel, etc. Le microprocesseur 8086 possède trois lignes principales d'interruption matérielles:

- NMI (No masquable interrupt) : Une interruption est dite non masquable signifie qu'elle doit toujours être reconnue par le microprocesseur dès que le signal électrique a été déclenché ;
- INTR (INTERUPT REQUASTE) : C'est une interruption autorisée si IF=1 sinon elle est masquée ;
- Reset remise à zéro de l'exécution du programme et réinitialisation du µp.

Les interruptions matérielles sont produites par l'activation de l'une de ces trois lignes du microprocesseur. À la différence des interruptions logicielles, **INTR** peut être autorisée ou interdits au moyen de l'indicateur IF du registre EFLAGS.

Le microprocesseur 8086 ne dispose réellement que de deux lignes de demandes d'interruptions matérielles (NMI et INTR), puisque RESET sont rôle est limitée dans la réinitialisation du µp. Pour pouvoir connecter plusieurs périphériques utilisant des interruptions, on peut utiliser le contrôleur programmable d'interruptions 8259.

Il peut arriver que l'on ait besoin, lors de l'exécution d'une routine, qu'aucune interruption (nécessairement matérielle) ne soit prise en compte, pour ne pas gêner le bon déroulement du programme. Ceci est possible pour les interruptions externes masquables, c'est-à-dire celles déclenchées par la broche INTR.

III.3.5.3- Boucles en assembleur

Comme tous les autres langages de haut ou de bas niveau, en assembleur en utilisant sauvant des boucles dans les algorithmes. Mais, il n'existe pas de structure générale en pour coder une boucle. Cependant, à l'aide des instructions vues précédemment pour réaliser des tests, on peut coder n'importe quel type de boucle.

III.3.5.3.a- Boucles tant-que

Le squelette d'une boucle tant-que dans le langage assembleur peut être représenté de la manière suivante :

```
BTQ: Test de la condition
     J_SC  FBTQ
     Exécution des actions
     .....
     .....
     JMP BTQ
FBTQ:
     .....
```

- J_SC : dénote l'une des instructions de saut conditionnel vues déjà ;
- BTQ : une étiquette indique le début de la boucle ;
- FBTQ : une étiquette indiquant la fin de la boucle ;
- En fonction de la valeur des indicateurs (CF=1 par exemple pour instruction JC), un saut conditionnel est effectué en fin de boucle (J_SC FBTQ), pour quitter la boucle le moment venu ;
- On trouve ensuite le corps de la boucle, terminé par une instruction de saut inconditionnel vers le début de la boucle (JMP BTQ) qui permettra de réévaluer la condition d'arrêt après chaque itération

III.3.5.3.b- Boucles répéter

Le squelette de la boucle REPETER dans le langage assembleur est le suivant :

```
B_REPETER:
        Exécution des actions
        .......
        .......
        Test de la condition
        J_SC   B_REPETER
```

Notons les points suivants :

- Le début de la boucle est repéré par une étiquette B_REPETER;
- On trouve ensuite le corps de la boucle qui constitué d'un ensemble d'actions ;
- A la fin du corps de la boucle, le test de la condition d'arrêt qui positionne les indicateurs du registre FLAGS ;
- La fin de la boucle est repérée par une instruction de saut conditionnel effectue un branchement pour continuer les itérations si la condition d'arrêt n'est pas vérifiée.

III.3.5.3.c- Boucles pour

Cette boucle est codée généralement à l'aide d'une instruction de la famille LOOP, cette dernière d´décrémente le compteur de boucles (registre CX).

Le squelette de cette boucle en assembleur est donné de la manière suivante :

```
        MOV  CX, b
B_POUR: ....
        Exécution des actions
        ....
        ....
        LOOP B_POUR
```

III.3.5.4- Directives de base

Pour programmer en assembleur, on doit utiliser, en plus des instructions assembleur, des directives ou pseudo-instructions, par exemple pour créer de l'espace mémoire pour des variables, pour définir des constantes, etc. Une directive est une information que le programmeur fournit au compilateur. Elle n'est pas transformée en une instruction en langage machine. Elle n'ajoute donc aucun octet au programme compilé. Donc les directives sont des déclarations qui vont guider l'assembleur.

III.3.5.4.a- Constantes numériques

Une constante peut être définie par un nom en utilisant la directive **EQU.** La directive EQU est utilisée pour remplacer les adresses physiques des ports par des noms symboliques, ceci va faciliter la lecture et la compréhension du programme.

Syntaxe : Nom constante EQU valeur ; commentaire

Exemple :

```
Val1   equ 1BH ; définir la valeur 1
Quatre EQU  4
Duree  EQU 12
Part2  EQU 01001010B
```

III.3.5.4.b- Caractères

Les chaînes de caractères sont définies entre apostrophes, chaque caractère sera traduit en code ASCII (un octet par caractère).

Exemple :

```
MOV AL, 'X';Chargement de AL par le caractère «X»
```

Si on veut mettre un caractère dans une constante, il suffit de la doubler les apostrophes, exemple: 'l''arbre'.

III.3.5.4.c- Déclarations de variables

Comme dans tout langage, il faut préalablement déclarer les variables à l'aide de directives bien précises, on repère les variables grâce à leur nom et en assembleur chaque variable a une adresse. Pour stocker des variables sous les différents types, une réservation de cases mémoires (un emplacement en mémoire) doit être toujours effectuée. Les noms des variables sont composés d'une suite de 31 caractères au maximum, commençant obligatoirement par une lettre. Le nom peut comporter des majuscules, des minuscules, des chiffres, plus les caractères @, et _. Lors de la déclaration d'une variable, on peut lui affecter une valeur initiale. LENGTH : nombre de termes de la variable (octets, mots, double mots), SIZE : nombre d'octets (SIZE = LENGTH * TYPE).

Les directives utilisées sont les suivantes :

❖ **DB** : **Define Byte**, réservation d'un octet pour la variable

Exemples :

```
nbre1 DB 19 ; une variable (un octet) initialisée par la valeur 19
Table1 DB 21H, 07H, 18H ; Table1 référence un tableau de 3 octets,
                        ; réinitialisé par  21 pour le premier octet, 07
                        ; pour le second octet et 18 pour le troisième
Message1 DB 'bonjour tous' ; référence un tableau définis par le code
                           ;  Ascii de chaque lettre de la  chaine de
                           ;  caractères composée des caractères 'b', 'o',
                           ;  'n', 'j', 'o', 'u ', 'r', 't', 'o', 'u', 's'
Var2 DB ? ; définit une variable 8 bits de valeur initiale quelconque
CHIFFRES DB 0,1,2,3,4,5,6,7,8,9; a une longueur 10 et une dimension 10
```

❖ **DW** : **Define word**, la variable est constituée d'un mot (réservation de 2 octets)

Exemples :

```
Val2  DW 0559 ; en décimal réserve deux cases mémoire (un mot) à partir
              ; de l'adresse Val2.
Val1  DW 1B2AH ; en hexa, réservation de 2 octets (mot)
Calcul2  DW ((16 * 36) / 96 + 86) ; référence une donnée codée sur
                                  ; un mot et qui est initialisée avec
                                  ; la valeur 92
ANNEES DW 1986, 1996, 2012, 2020  ; a une longueur 4 et une dimension 8
Varb1  DW ?    ; réserve un mot dans l'adresse mémoire m et m+1 de
               ; valeur initial quelconque.
```

❖ **DD : Define double**, la variable est constituée de 2 mots soit un espace mémoire de quatre (04) octets.

Exemples :

```
Val4 DD 26500010H ; en hexadécimal réserve quatre cases mémoire (deux mots)
```

❖ **DQ** : des mots quadruples, réservation d'un espace mémoire de 8 octets (64 bits)

Exemples :

```
Val4 DQ 2650001026500010H
```

❖ **Tableaux :** Il est aussi possible de déclarer des tableaux, c'est à dire des suites d'octets ou de mots consécutifs. Pour cela, il faut utiliser plusieurs valeurs initiales :

Exemples :

```
mach2 DB 10, 0FH ; réserve un tableau de 2 cases (2 octets)
Table4 DB 11H, 17H, 28H ; Table4 référence un tableau de 3 octets,
                        ; réinitialisé par  11H pour le premier octet, 17H
                        ; pour le second octet et 28H pour le troisième
TAB1 DW 14H, 18H, 24H, 32H ; réserve un tableau de 08 cases chaque valeur
                           ; sera mise sur deux cases (une longueur 4 et
                           ; une dimension 4×2=8)
```

❖ **Directive dup**

Lorsque l'on veut déclarer un tableau de n cases, toutes initialisées à la même valeur, on utilise la directive dup. Permet de dupliquer une valeur lors de la définition de variables au moyen de DB, DW, DD ou DQ.

Exemples :

```
  Table8 DB 100 dup (15); 100 octets initialisés par 15
  AYZ DW 10 dup (?); 10 mots de 16 bits non initialises
  TABLEAU DB 40 dup (0); TABLEAU référence un tableau de 40 octets
                       ;  Dont la valeur initiale des éléments est 0
  MOTS DW 50 DUP (0); le symbole TMOTS  qui référence un tableau de 50
                    ; mots initialises avec la valeur 0
```

❖ **Directive PTR**

Permet de préciser la taille du la donnée concernée lors de traitement des cases mémoires avec un adressage généralement indirect. L'assembleur ne sait pas si l'instruction concerne 1, 2, 4 ou 8 octets consécutifs. Afin de lever l'ambiguïté, on doit utiliser une directive spécifiant la taille de la donnée à transférer.

Exemples :

```
MOV BYTE PTR [1010H] ,45H ; transfère la valeur 45H (sur 1 octet) dans la
                          ; case memoire d'offset 1010H
MOV WORD PTR [1110H] ,85H ; transfère la valeur 0085H (sur 2 octets) dans
                          les cases memoire d'offset 1110H et 1111H
ADD BYTE PTR [1201H] ,08H; ajoute la valeur 08H au contenu de la case
                          memoire d'offset 1201H (adressage immédiat)
JMP WORD PTR [BX] ; s'il s'agit d'un saut intra segment
JMP DWORD PTR [BX] ; s'il s'agit d'un saut inter segment
```

III.3.5.4.d- Directives de segment

Cette directive précise dans quel segment, les instructions ou les données qui la suivent seront assemblées. La définition minimale du segment comprend :

- Sa borne supérieure où se trouve son nom, un identificateur suivi de SEGMENT;
- Sa borne inférieure donnée par ENDS précédée du nom du segment.

❖ **SEGMENT .data :** Les données (initialisée) déclarée après cette directive sont placée dans le segment de données (Data segment) ;

```
Data SEGMENT
Truc DW 0F0AH          ; en hexa
 Masque DB 01110000b    ; en binaire
Entree DW 15           ; 2 octets initialises a 15
Naga   DB -1           ; 1 octet initialise a -1
Data ENDS
```

- **SEGMENT. text :** Les instructions ou les données initialisée (par db, dw, etc.) qui suivent cette 'déclaration' seront placé dans le segment programme (Code segment).
- **SEGMENT .bss :** Les données déclarées après cette directive seront placées dans le segment de données mais cette partie est réservée à la déclaration des variables non initialisée.

* La directive ASSUME est toujours présente et sera expliquée section suivante.

Exemples :

```
Data SEGMENT ; data est le nom du segment de données
             ; Directives de déclaration de données
Data ENDS    ; fin du segment de données

       ASSUME DS: data, CS: code

Code SEGMENT ; code est le nom du segment d'instructions
Debut:       ; 1ere instruction, avec l'étiquette début
             ; Suite d'instructions

Code         ENDS
             END Debut ; fin du programme, avec l'étiquette
                       ; de la première instruction.
```

III.3.5.4.e- Directive ASSUME

Les directives ***SEGMENT*** et ***ENDS*** permettent de définir les segments de codes et de données. Pour une génération correcte des adresses, il faut bien préciser quel est le registre de segment de données et celui de codes. Cette opération est dévolue à la directive **ASSUME** que l'on écrit généralement en première ligne dans le segment du code. Le programme doit commencer, avant toute référence au segment de données, par initialiser le registre segment DS, de la façon suivante :

```
MOV AX, Data
MOV DS, AX
```

III.3.5.4.f- Anatomie d'un programme en assembleur

Un programme écrit en assembleur suit un enchainement d'étapes très important pour une réalisation correcte et efficace, sachant qu'il comprend des définitions, des opérandes, des données et des instructions, qui s'écrivent chacune sur une ligne de texte. La meilleures façon qu'en trouvant efficace pour répondre aux différentes exigences est la suivante :

a- Définir le problème à résoudre et comment le résoudre: que faut-il faire exactement ?

b- Enchainer les différents éléments de la solution dans des algorithmes ou des organigrammes en répondant à ces deux questions : Comment faire ? Par quoi commencer, puis poursuivre ?

c- Rédiger le programme (**code source**) : En utilisant les différente directives citées en haut, les jeux d'instructions (mnémoniques) et des textes explicatifs sous forme des commentaires, tout en respectant sa forme bien particulière. Chaque ligne d'une source assembleur comporte une instruction. Chaque ligne est composée de champs. De gauche à droite, on a :
- Le champ étiquette (Une étiquette est un identificateur composé de lettres, chiffres et de caractères $, %, _ et ?), qui peut être vide ;
- Le champ mnémonique (le nom de l'instruction) ;
- Le champ opérande (les arguments de l'instruction), qui peut être vide ;
- Le champ commentaire, qui peut être vide.

d- Implanter le programme sur l'environnement (logiciel choisir), et le tester en réel ;

e- Corriger les erreurs éventuelles puis refaire des tests jusqu'à obtention d'un programme fonctionnant de manière satisfaisante.

III.3.5.5- Interface d'entrées/sorties

Une interface d'entrées/sorties est un circuit intégré spécialisé , est donc été conçu pour faciliter la réalisation d'une interface de périphériques permettant au microprocesseur de communiquer et d'échanger l'information (peut tester, lire, écrire ou contrôler) avec l'environnement extérieur (périphériques reliées à ce microprocesseur) : clavier, écran, imprimante, modem, disques, processus industriel, etc. Un autre but est surtout de décharger le microprocesseur pour améliorer les performances du système. Les interfaces d'E/S sont connectées au microprocesseur à travers les bus d'adresses, de données et de commandes. Il existe plusieurs types d'interface : parallèle, série et même série/parallèle.

III.3.5.5.a- Interface parallèle

Le rôle d'une interface parallèle est de transférer des données du microprocesseur vers des périphériques et inversement, tous les bits de données étant envoyés ou reçus simultanément. Par exemple, Intel a conçu le 8255A (qui peut être configurée en entrée et/ou en sortie) adapté au microprocesseur 8086 (et donc au 8088).
Le PPI (Programmable Peripheral Interface) 8255A est un interface programmable travaillant en mode parallèle, il possède :

- Un bus de données de 8 bits pour la communication avec le microprocesseur ;
- 24 lignes programmables en entrées ou en sorties, réparties en 2 port de 8 bits (A et B) et deux demi-ports (C) de 4 lignes ;
- Tension d'alimentation 5V ±5%.

III.3.5.5.b- Interface série

Une interface série permet d'échanger des données à faible débit entre le microprocesseur et un périphérique **bit par bit**.

Il existe deux types de transmissions séries :

- **Asynchrone** : chaque octet peut être émis ou reçu sans durée déterminée entre un octet et le suivant ;
- **Synchrone** : les octets successifs sont transmis par blocs séparés par des octets de synchronisation.

Par exemple, le composant électronique chargé de la gestion des transmissions séries asynchrones dans les PC est appelé UART (*Universal Asynchronous Receiver Transmitter*), comme le Intel 8250.

L'interface 8250 possède plus de registres de 8 bits permettant de gérer la communication. Ces registres sont lus et modifiés par le processeur par le biais des instructions IN et OUT vues plus haut.

Annexe III

Questions

Au sein du processeur 8086 :

1. Quel est le rôle du registre IP ?
2. Quel sont les composants du registre d'état ?
3. CF est égal à 1, expliquez ?
4. ZF est égal à 1, expliquez ?

Réponses

1. Le registre IP (Instruction Pointer) est un registre du processeur qui désigne la prochaine instruction à exécuter. Ce n'est qu'un simple compteur d'instructions la raison pour laquelle il est dit *Compteur Ordinal*.
2. Le registre d'état est constitué des bits suivants :

15															0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

3. CF= 1 car l'instruction précédente a généré une retenue.
4. ZF=1 car l'instruction précédente a généré un résultat nul.

Exercice 1

a- Pour chacune des lignes de la routine assembleur, identifiez les modes d'adressage utilisés :

Numéro de ligne	
1	`STRLEN: MOV SI, [SP+4]`
2	`MOV AX, 0`
3	`@@: CMP BYTE PTR [SI+AX*1], 0`
4	`JE @F`
5	`INC AX`
6	`JMP @B`
7	`@@: RET`

b- Pour chaque instruction du programme suivant, définir le type d'adressage :

Numéro de ligne	
1	`MOV CX, 10`
2	`MOV DI, 0`
3	`MOV AX, 0`
4	`DEC CX`
5	`JZ Fin`
6	`ADD AX, [DI+0200h]`
7	`ADD AX, [BX+0100h]`
8	`Fin : MOV [0300h], AX`

c- Pour chaque instruction, définir le type d'adressage :

Numéro de ligne	
1	`MOV AL, 0DFh`
2	`AND AL, 11011111b`
3	`INC AL`
4	`MOV AH, 0CCh`
5	`OR AH, 20h`
6	`DEC AH`

- Que contiennent les registres AL et AH après l'exécution de cette séquence ?

Réponse :

a-

1. Registre, Indirect avec déplacement
2. Registre, Immédiate
3. Indirect avec index, Immédiate
4. Immédiate, Implicite (le registre de drapeaux)
5. Registre
6. Immédiate
7. Implicite (le pointeur de pile et la pile)

b-

1. Immédiat
2. Immédiat
3. Immédiat
4. Implicite
5. Relatif
6. Indexé avec déplacement
7. Basé avec déplacement
8. Direct

c-

1. Immédiat
2. Immédiat
3. Implicite
4. Immédiat
5. Immédiat
6. Implicite

Après l'exécution de cette séquence : AL = E0 h = 11100000 b ; AH = EB h = 11101011 b

Exercice 2

Voici un code un peu tordu. Avant le début de l'exécution, les drapeaux « CF », « ZF », « SF » et « OF » sont tous à zéro

```
MOV AX, 0A9876543H
ADD AX, 065432100H
MOV AX, 1
DEC AX
```

1) Quel est l'état des drapeaux « CF », « ZF», « SF », « OF » après l'exécution de l'instruction `add` ?
2) Quel est l'état de ces mêmes drapeaux après l'exécution de l'instruction `dec` ?

Notes

- Le segment de code courant travaille avec des opérandes de 32 bits.
- Le segment de pile courant utilise des adresses de 32 bits.

Réponse :

1) Carry = 1, Zero = 0, Sign = 0 et Overflow = 0
2) Carry = 1, Zero = 1, Sign = 0 et Overflow = 0

Exercice 3

On veut additionner deux nombres non signés **52h** et **29h**. Le résultat est rangé à mémoire SUM s'il est positif.

Réponse :

```
            .MODEL SMALL
            .STACK 64
            .DATA
DATA 1      DB          52h
DATA 2      DB          29h
SUM         DB          ?
            .CODE
MAIN        PROC         FAR

DEBUT:      MOV     AX, @DATA
            MOV     DS, AX
            MOV     AL, DATA1
            MOV     BL, DATA2
            ADD     AL, BL
            JZ      DEBUT
            MOV     SUM, AL
            MOV     AH, 4Ch
            INT     21h
MAIN        ENDP
            END     MAIN
```

Exercice 4

On veut additionner cinq nombres (**5 octets)** non signés **(25h, 12h, 15h, 1Fh, et 2Bh)**. Le résultat est rangé à la mémoire SUM.

Réponse :

```
TITLE      prog2_1.asm: Addition de 5 octets (25h, 12h, 15h, 1Fh, et 2Bh) et stocker le
           résultat
           .MODEL SMALL                    ; définit le modèle de mémoire comme petit
           .STACK 64                       ; marque le début de SS, et lui réserve 64Ko
           .DATA                           ; marque le début de DS
DATA_IN    DB      25h, 12h, 15h, 1Fh, 2Bh
SUM        DB      ?
           .CODE                           ; marque le début de CS
MAIN       PROC    FAR                     ; Entrée du progam. avec PROCedure étiquetée
                                           ; MAIN d'option FAR
           MOV     AX, @DATA               ; charger l'adresse du segment de données
           MOV     DS, AX                  ; assigner une valeur à DS
           MOV     CX, 05                  ; Initialiser le compteur boucle a la valeur 5
           MOV     BX, OFFSET DATA_IN      ; Assigner au pointeur de données BX l'adresse
                                           ; offset de DATA_IN.
                                           ; Note: La directive OFFSET permet d'accéder à
                                           ; une adresse offset assignée a une variable ou
                                           ; un nom
             MOV   AL, 0                   ; Initialiser AL
ENCORE:      ADD   AL, [BX]                ; Ajouter la prochaine donnée au registre
                                           ; Accum. A
             INC   BX
             DEC   CX                      ; Décrémenter la boucle du compteur
             JNZ   ENCORE                  ; Sauter si le compteur n'est pas zéro
             MOV   SUM, AL                 ; charger le résultat dans la location mémoire
                                           ; SUM
             MOV   AH, 4Ch
            INT    21h                     ; retourne le control au DOS
MAIN  ENDP                                           ; fin de la procédure MAIN
           END     MAIN
```

Autre méthode

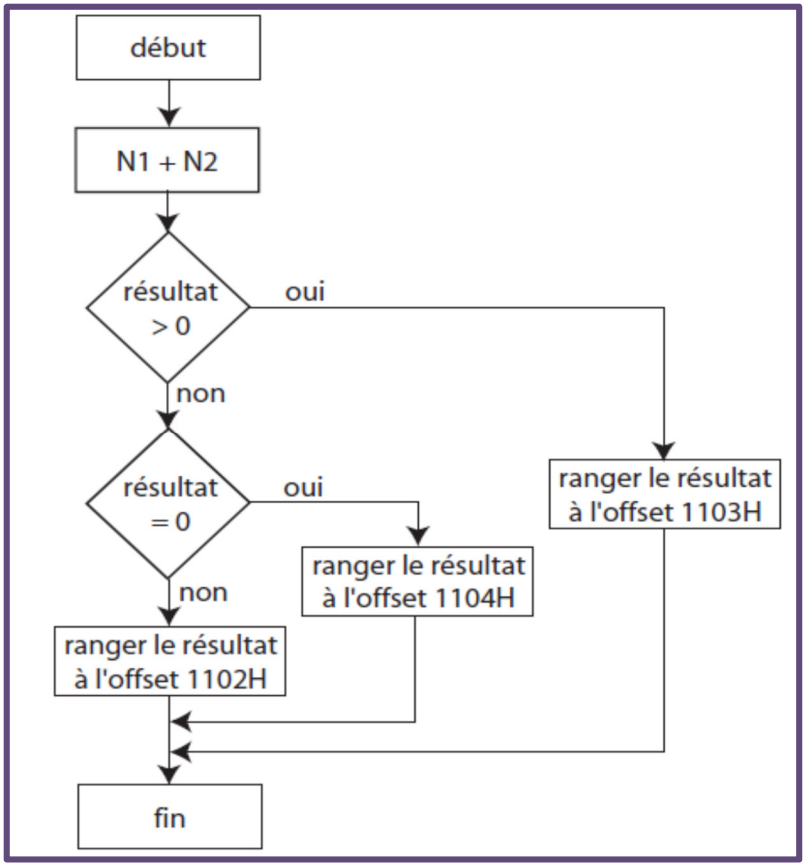
```
TITLE   prog2_1b.asm: Addition de 5 octets (25h, 12h, 15h, 1Fh, et 2Bh)
        .MODEL SMALL
        .STACK 64
        .DATA
DATA1   DB      25h     ; Le DS définit une donnée: DATA1
DATA2   DB      12h     ; Le DS définit une donnée: DATA2
DATA3   DB      15h     ; Le DS définit une donnée: DATA3
DATA4   DB      1Fh     ; Le DS définit une donnée: DATA4
DATA5   DB      2Bh     ; Le DS définit une donnée: DATA5
SUM     DB      ?       ; Le DS définit une donnée résultat: SUM
        .CODE
MAIN    PROC    FAR
        MOV     AX, @DATA   ; charger l'adresse du segment de données
        MOV     DS, AX      ; assigner une valeur a DS
        MOV     AL, DATA1   ; Déplacer DATA1 vers AL
        ADD     AL, DATA2   ; Ajouter la donnée DATA2 au registre AL
        ADD     AL, DATA3   ; Ajouter la donnee DATA3 au registre AL
        ADD     AL, DATA4   ; Ajouter la donnee DATA4 au registre AL
        ADD     AL, DATA5   ; Ajouter la donnee DATA5 au registre AL
        MOV     SUM, AL     ; Charger le résultat dans la mémoire SUM
        MOV     AH, 4Ch
        INT     21h         ; Retourne le control au DOS
MAIN    ENDP                ; Fin de la procédure MAIN
        END     MAIN
```

Exercice 5

On veut additionner deux nombres signés N1 et N2 se trouvant respectivement aux offsets 1100H et 1101H. Le résultat est rangé à l'offset 1102H s'il est positif, à l'offset 1103H s'il est négatif et à l'offset 1104H s'il est nul :

Réponse :

Organigramme :



Programme

```
                MOV AL,[1100H]
                ADD AL,[1101H]
                JS  NEGATIF
                JZ  NUL
                MOV [1102H],AL
                JMP FIN
NEGATIF :       MOV [1103H],AL
                JMP FIN
NUL :           MOV [1104H],AL
FIN :           HLT
```

Exercice 6

Ecrire une fonction à deux arguments a et b et qui retourne leur moyenne.

Réponse :

```
; Convention:
; ax contient le premier entier
; bx contient le deuxième entier
; le résultat sera dans ax
moyenne:
                add ax, bx
                shr ax, 1
                ret

                mov ax, 12
                mov bx, 6
call moyenne
; ici ax contient 9
```

Exercice 7

Nous recherchons dans un tableau d'entiers l''élément d'indice le plus faible qui soit multiple de 7.

Réponse :

```
          .DATA
TABLEAU   DB 24, 75, 51, 36, 63, 13, 8, 9, 47, 70, 83, 95
          .CODE
          MOV AX, @DATA
          MOV DS, AX
          MOV CX, 12        ; NOMBRE D'ELEMENTS DANS LE TABLEAU
          MOV DL, 7         ; VALEUR DU DIVISEUR
          MOV BX, 0         ; INDICE DE L'ELEMENT CONSIDERE
BOUCLE1:
          MOV AH, 0
          MOV AL, BYTE PTR TABLEAU+BX
          DIV DL            ; DIVISION DE L'ELEMENT COURANT PAR 7
          INC BX            ; MISE A JOUR DE L'INDICE
          CMP AH, 0         ; AH CONTIENT LE RESTE DE LA DIVISION
          LOOPNE BOUCLE1    ; SAUT SI NON MULTIPLE ET NON TERMINE
          INT 21H
CODE      ENDS
          END BOUCLE1
```

Exercice 8

Ecrire en assembleur un programme qui permet d'afficher en binaire la valeur d'un nombre.

Réponse :

Principe

Afficher l'un apr`es l'autre les bits du nombre en effectuant une boucle du genre :

Pour b := 1 a 16 faire

Si (valeur.bit[b] = 0) alors

Afficher ('0')

Sinon afficher ('1')

Fait

Programme

```
          .MODEL SMALL
          .STACK 100H
          .DATA
VALEUR    DW  52612
          .CODE
          MOV AX, @DATA
          MOV DS, AX
          MOV BX, [VALEUR]
          MOV CX, 16
BOUCLE:   ROL BX, 1
          JC  UN
          MOV DL, '0'
          JMP PRINT
UN:       MOV DL, '1'
PRINT:    MOV AH, 2
          INT 21H
```

```
        LOOP BOUCLE
FIN:    MOV AH,4CH
        INT 21H
        END
```

Affiche 1100110110000100 qui est bien la représentation binaire de 52612.

Exercice 9

On suppose les données suivantes : (IP) = 2B1Ah, (BX) = 637Eh, (SI) = 2A9Bh et déplacement = 0F7h.

A partir de ces données, déterminer l'adresse effective (si elle existe) pour chacun des modes d'adressages :

- Adressage basé par BX ; Adressage basé par BX avec déplacement ; Adressage immédiat
- Adressage basé indexé avec déplacement ; Adressage relatif

Réponse :

On suppose les données suivantes : (IP) = 2B1Ah, (BX) = 637Eh, (SI) = 2A9Bh et déplacement = 0F7h.

Mode d'adressage	Calcul de l'adresse	Adresse Effective
Adressage basé par BX	[BX]	637E h
Adressage basé par BX avec déplacement	[BX+Dep.]	637E h - 9 h = 6375 h (dep. 8bits) 637E h + 00F7 h = 6475h (dep. 16 bits)
Adressage immédiat	Aucune Adresse	Aucune Adresse
Adressage basé indexé avec déplacement	[BX+SI+Dep.]	637E h +2A9B h-9 h = 8E10 h (dep. 8bits) 637E h +2A9B h +00F7 h = 8F10 h (dep. 16bits)
Adressage relatif	[IP + Dep.]	2B1A h + 0F7 h = 2B1A h - 9 h = 2B11 h

Exercice 10

Donner l'équivalent des instructions du langage C suivantes en assembleur

- If (ax==1) bx = 10; else { bx = 0; cx = 10; }
- For (cx=0; cx<=10; cx++) { bx = bx + cx; ax=ax+5; }
- while (bx >0) { bx = bx -1; }
- do { cx=cx+1 ; bx=bl*2} while (cx<=15)
- if ((ax>bx) && (cx <= dx)) { ax=ax-bx ; cx=cx+dx ; } -- Pour des nombres signés.
- Switch (ax) {case 1: bx=3 ; break ; case 2: bx=5 ; break ; default : bx=1 ; }
- if ((ax>bx) || (cx <= dx)) { bx=bx+cx ; cx=cl*dl } -- Pour des nombres non-signés.

Réponse :

- If (ax==1) bx = 10; else { bx = 0; cx = 10; }

```
      CMP AX, 1
      JE  egal
      MOV BX, 0
      MOV CX, 0
      JMP fin
egal: MOV BX, 10
fin:  HLT
```

- For (cx=0; cx<=10; cx++) { bx = bx + cx; ax=ax+5; }

1. Sans l'utilisation de l'instruction LOOP

```
 MOV    CX, 0
Boucle: ADD BX, CX
        ADD AX, 5
        INC CX
        CMP CX, 11
        JNE Boucle
        HLT
```

2. Avec l'instruction LOOP

```
        MOV CX, 11
        MOV DX,0
Boucle: ADD BX, DX
        ADD AX, 5
        INC DX
        LOOP Boucle
        HLT
```

• while (bx >0) { bx = bx -1; }

```
Boucle: CMP BX, 0
        JLE Fin
        DEC BX
        JMP Boucle
Fin:    HLT
```

• do { cx=cx+1 ; bx=bl*2} while (cx<=15)

```
Boucle: INC CX
        MOV AL, 2
        MUL BL
        MOV BX, AX
        CMP CX, 16
        JNE Boucle
        HLT
```

• if ((ax>bx) && (cx <= dx)) { ax=ax-bx ; cx=cx+dx ; } -- Pour des nombres signés.

```
     CMP AX, BX
     JLE Fin
     CMP CX, DX
     JG Fin
     SUB AX, BX
     ADD CX, DX
Fin: HLT
```

• Switch (ax) {case 1: bx=3 ; break ; case 2: bx=5 ; break ; default : bx=1 ; }

```
         CMP AX, 1
         JE Case1
         CMP AX, 2
         JE Case2
Default: MOV BX, 1 ; Correspond à Default
         JMP Break
Case1:   MOV BX, 3
         JMP Break
Case2:   MOV BX, 5
Break:   HLT
```

• if ((ax>bx) || (cx <= dx)) { bx=bx+cx ; cx=cl*dl } -- Pour des nombres non-signés.

```
        CMP AX, BX
        JA Valide ; Correspond à une condition vérifiée
        CMP CX, DX
        JBE Valide ; Correspond à la deuxième condition vérifiée
        JMP Fin ; Correspond à aucune condition vérifiée
Valide: ADD BX, CX
        MOV Al, Dl
        MUL Cl
        MOV CX, AX
Fin :   HLT
```

Exercice 11

- Ecrire en assembleur les instructions qui déterminent la valeur du bit N° 3 du registre AX ; (LSB=bit N°0 ; MSB=bit N°7) ;
- Donner les instructions assembleur qui permettent de mettre le bit N°1 du registre DH à 1, le bit N° 3 à 0 sans changer les autres bits du registre ;
- Donner la séquence d'instructions permettant de vérifier si les bits du registre AX sont: A0=1, A3=0, A5=0, A8=1 et A14=1.

Réponse :

1- Déterminer la valeur du bit N° 3 du registre AX

En utilisant l'opération de masquage par une opération AND

AND AX, 0008 h ; (0000 0000 0000 1000 b) Masquage de tous les bits sauf le bit N° 3
CMP AX, 0008 h ; (0000 0000 0000 1000 b) Test si le bit N° 3 est égal à 1
JE Etiq

L'opération de masquage peut être également effectuée par une opération OU

OR AX, 0FFF7h ; (1111 1111 1111 0111 b) Masquage de tous les bits sauf le bit N° 3
CMP AX, 0FFF7h ; (1111 1111 1111 0111 b) Test si le bit N° 3 est égal à 0
JNE Etiq

En utilisant 4 décalages successifs à droite :

SHR AX, 4
JC Etiq

2- Mettre le bit N°1 du registre DH à 1, le bit N° 3 à 0 sans changer les autres bits du registre.

OR DH, 02 h ; (0000 0010 b) Forcer le bit N°1 à 1
AND DH, F7 h ; (1111 0111 b) Forcer le bit N°3 à 0

3- Vérifier si les bits du registre AX sont : A0=1, A3=0, A5=0, A8=1 et A14=1.

AND AX, 4129h ; (0100 0001 0010 1001 b) Masquer tous les bits sauf les bits 0, 3, 5, 8 et
CMP AX, 4101h ; (0100 0001 0000 0001 b) Comparer par rapport à la bonne combinaison

Ou bien en utilisant un masquage par une opération OR :

OR AX, 0BED6h ; (1011 1110 1101 0110 b) Masquer tous les bits sauf les bits 0, 3, 5, 8 et
CMP AX, 0FFD7h ; (1111 1111 1101 0111 b) Comparer par rapport à la bonne combinaison

Exercice 12

Réaliser un programme en assembleur 8086, qui permet d'additionner deux nombres signés N1 et N2 se trouvant respectivement aux offsets 1100H et 1101H du segment de données. Le résultat est rangé dans le segment de données supplémentaires, à l'offset 1102H s'il est positif, et à l'offset 1103H s'il est négatif et à l'offset 1104H s'il est nul.

Réponse :

```
            MOV AL, [1100h]
            ADD AL, [1101h]
            JS  Negatif
            JZ  Nul
MOV ES:     [1102 h], AL
            JMP fin
Negatif:    MOV ES:[1103 h], AL
            JMP Fin
Nul :       MOV ES:[1104 h], AL
Fin :       HLT
```

Exercice 13

Ecrire un programme assembleur 8086 qui permet d'annuler la diagonale d'une matrice carrée 10x10. Les autres valeurs devront rester inchangées. On suppose que la matrice est sauvegardée à l'adresse 1000h du segment 0200h, et que les éléments sont sur 8 bits.

Réponse

$$\begin{pmatrix} 0 & X & X & X & X & X & X & X & X & X \\ X & 0 & X & X & X & X & X & X & X & X \\ X & X & 0 & X & X & X & X & X & X & X \\ X & X & X & 0 & X & X & X & X & X & X \\ X & X & X & X & 0 & X & X & X & X & X \\ X & X & X & X & X & 0 & X & X & X & X \\ X & X & X & X & X & X & 0 & X & X & X \\ X & X & X & X & X & X & X & 0 & X & X \\ X & X & X & X & X & X & X & X & 0 & X \\ X & X & X & X & X & X & X & X & X & 0 \end{pmatrix}$$

```
      MOV  AX, 0200h
      MOV  DS, AX
      MOV  CL, 10
      MOV  SI, 0
      MOV  BX, 0
Etiq: MOV  BX+SI+1000h],0
      ADD  BX, 10
      INC  SI
      LOOP Etiq
      HLT
```

Exercice 14

Transfert 6 octets de données des adresses avec offset 0010h vers des adresses avec offset 0028h.

Réponse :

```
TITLE       prog3_3.asm: Transfert 6 octets de données
            .MODEL SMALL                          ; définit le modèle de mémoire
                                                  ;comme petit
            .STACK 64                             ; marque le début de SS, et lui réserve 64Ko
            .DATA                                 ; marque le début de DS
            ORG       15h
DATA_IN     DB        2Eh, 23h, 0D5h, 6Ah, 89h, 11h  ; Le DS définit 6 données de type DB →
                                                  ; notez que 0D5h et non D5h
            ORG       32h
COPIE       DB        6 DUP(?)                    ; pour dupliquer un certain nombre de
                                                  ; caracteres: reserver 6 octets
            .CODE                                 ; marque le début de CS
MAIN        PROC      FAR
            MOV       AX, @DATA                   ; charger l'adresse du segment de
                                                  ; données
            MOV       DS, AX                      ; assigner une valeur a DS
            MOV       SI, OFFSET DATA_IN          ; SI pointe sur la donnée a copier.
            MOV       DI, OFFSET COPIE            ; DI pointe sur la COPIE de la données.
            MOV       CX, 06h                     ; Compteur de boucle = 06
MOV_LP:     MOV       AL, [SI]                    ; Déplacer la prochaine donnée de
                                                  ; l'espace DATA_IN vers AL
            MOV       [DI], AL                    ; Déplacer la prochaine donnée vers
                                                  ; l'espace COPIE
            INC       SI                          ; Incrémenter SI
            INC       DI                          ; Incrémenter DI
            DEC       CX                          ; Décrémenter la boucle du compteur
            JNZ       MOV_LP                      ; Sauter a l'instruction MOV_LP si le
                                                  ; compteur est non zero
            MOV       AH, 4Ch                     ; retourne le control au DOS
            INT       21h                         ; retourne le control au DOS
MAIN        ENDP                                  ; fin de la procédure MAIN
            END       MAIN                        ; point de sortie du programme
```

CONCLUSION

Conclusion

Ce support de cours est destiné à être utilisé comme un manuel par les étudiants de deuxième année Master Electromécanique - Sciences et Technologie du département Génie Electrique (Faculté de Technologie)- dans le domaine de la programmation des automates industriels et les microprocesseurs en général. Il a été rédigé dans le but d'ouvrir aux étudiants une fenêtre sur l'automatisation des systèmes industriels et la programmation en assembleur et de leurs permettre aussi d'avoir un outil de travail et de référence recouvrant les connaissances qui leurs sont demandés. Le manuscrit est constitué de cours et de quelques applications sous forme d'exercices et de problèmes avec solution détaillée et sans solution pour pousser les étudiants à faire des efforts individuels, il est conforme aux programmes agrée par le ministère. Sa présentation didactique est le fruit de quelques années d'expérience pédagogique par l'auteur. Son contenu résulte de la lecture de nombreux ouvrages et documents dont les plus importants sont cités dans les références bibliographiques. Ce travail a été effectué à base de la collecte de données précises et ciblés afin de faciliter la compréhension aux étudiants.

La résolution de la complexité de certaines taches industrielles ou les applications dites critiques sont parmi les plus grands challenges. L'automatisation conduit à une très grande rapidité, une meilleure régularité des résultats et évite à l'homme des tâches pénibles et répétitives, elle est considéré comme l'élément porteur de civilisation actuellement. La programmation doit être effectuée par le langage de programmation qui sera traduit en un langage compréhensible directement par le microprocesseur. Ce langage est propre à chaque constructeur, il est lié au matériel mis en œuvre. A cet effet, nous avons jugé utile d'ajouter au programme officiel quelques notions de fait de l'interactivité existante pendant l'exécution d'un programme, c'est pour ça il est nécessaire de comprendre les effets de chaque instruction non seulement sur l'ordre d'exécution de celles-ci, mais aussi sur les données manipulées (mémoire, pile, registres et flags). L'apprentissage de la programmation en assembleur étant une tâche ardue pour un programmeur débutant, nous avons tenté, au travers de ce manuscrit, de la lui faciliter pour ambition d'aider les étudiants à mieux appréhender le fonctionnement de l'assembleur et sa programmation.

D'autre part, en plus de ce polycopie, l'étudiant doit être accompagné par un complément des travaux pratiques implémentés sous l'environnement SIMATIC Manager du constructeur SIEMENS, pour une meilleure maitrise de logiciel de programmation des APIs, Step 7, et de logiciel de conception des interfaces homme machine WINCC Flexible.

Bien que l'élaboration de ce manuscrit ait été faite avec le plus grand soin, le contrôle que nous avons pu faire de notre travail n'est pas absolu, et il serait étonnant qu'il ne subsiste pas d'erreurs.

Bibliographie

BIBLIOGRAPHIE

[1] M.pinot et al, « Du Grafcet aux automates programmables ». Collection L.P édit Foucher. Paris 1986.

[2] R. Azizi et R. Chemali, «Commande et supervision de l'unité de conditionnement d'huile - Cevital». Mémoire de Master 2, université de Béjaia, promotion 2010.

[3] Document Siemens, « Information et formation, automatisation et entrainements, programmation niveau A, ». Edition Siemens AG, 2003.

[4] Document Siemens, « Automate programmable S7-300, caractéristiques électriques et techniques des CPU SIMATIC». Edition Siemens, 2001.

[5] A. HADJAISSA, « Automates programmables industriels Description et programmation ». Support de cours, université Amar Telidji de Laghouat, 2019

[6] G. MICHEL, « Les API, architecture et application des automates programmables. Industriels ». Dunod, Paris, 1987.

[7] SIMATIC, « Mise en route STEP7 ». Édition 03 /2006.

[8] Manuel SIMATIC S7, « langage cont pour SIMATIC S7-300/400 ». Programmation de blocs, C79000-G7077-C504-02

[9] A. GIUA, « Automates programmables industriels ». 3A - Génie Industriel et Informatique, Polytech' Marseille. France, 2013

[9] S. Doudou, « Automates programmables industriels ». Support de cours, université Mohammed Seddik Benyahia -Jijel.

[10] E. MAALEM, I. TAOUADJI, «Les langages de programmation de l'automate programmable industriel ». Mémoire master 2, université d'Adrar, 2017

[11] N. Elkorno, « Utilisation de l'automate programmable ». Résumé théorique et guide de travaux pratiques, OFPPT, Maroc, 2007

[12] E. George, E. Bryla, «Software architecture and Framework for programmable logics controllers: A case study and suggestions for Research '', in, '' machine». 2016, 4, 13.

[13] N. Vandenbroucke, « Automatisation de processus industriels- l'automate programmable industriel (API) ». Ecole d'ingénieurs du Littoral, Côte d'Opale.

[14] L. Bergougnoux, « A.P.I. automates programmables industriels ». Polythèque Marseille, 2004/2005.

[15] A. Anissia et B. Salah-Eddine, « Réalisation et gestion d'un prototype de station de pompage à base d'automates programmables industriels SIEMENS ». PFE, école nationale polytechnique, Alger 2006/2007.

[16] M. E. M. Ben Gaid, « Modélisation et vérification des aspects temporisés des langages pour automates programmables industriels ». Rapport de stage, laboratoire spécification et vérification, université de Paris Sud – XI, 2003

[17] F. Benragouba, S. Allou, « Automatisation et supervision d'une station d'ensachage de la cimenterie de Sour el Ghozlane par API s7-300 ». Mémoire master 2, Université M'hamed Bouguerra Boumerdes, 2006.

[18] Document Siemens, « Information et formation, automatisation et entrainements, programation niveau A ». Edition Siemens AG, 2003.

[19] Document Siemens, « Automate programmable S7-300, caractéristiques électriques et techniques des CPU SIMATIC». Edition Siemens, 2001.

[20] A. Simon, « Automates programmables industriels Niveau 1». Edition l'Elan-Liège, 1991.

[21] K. Jlassi, « Les microcontrôleurs », Université virtuelle de Tunis.

[22] G. Comte, J-L. Calmon, « Le microcontrôleur 68HC11 (Motorola) ». IUFM de Toulouse, 1998.

[24] M. EBEL, « Informatique industrielle microcontrôleur 68HC811 ». Iut mesure physique, Valenciennes- France.

[25] « Le microcontrôleur 68HC9S12E128 ». Cours, école nationale d'ingénieurs de Tarbes- France.

[26] C. Durand, «Microcontrôleurs basé sur l'utilisation du HCS12 (Motorola) ». Cours de l'université Joseph Fourier Polytech' Grenoble, France 2009-2010.

[27] Caractéristiques des DSP http://www.technologuepro.com

[28] B. Sahli, « Processeurs de signaux numériques (DSP) ». Polycopie de cours, Université Ibn Khaldoun de Tiaret.

[29] Sen M Kuo, Bob H Lee, «Real-time digital signal processing». 2001 John Wiley. ISBN 0-470-84534-1.

[30] B. A. Shenoi, «Introduction to digital signal processing». 2006 by John Wiley & Sons, Inc. ISBN-13 978-0-471-46482.

[31] J. Proakis and D. Manolakis, «Digital signal processing: principles, algorithms, and applications». Prentice-Hall, 1996.

[32] A. Fruleux (EASYTEC), «Cours unité optionnelle DSP : processeur de traitement de signal ».

[33] S. Montagny, « Cours DSP : digital signal processors ». Université Savoie Mont Blanc, sylvain.montagny@univ-smb.fr

[34] H. ATOUI, « Programmation et interfaçage de microprocesseur 8086 ». Support de cours, Centre de formation *HBM Corporation*, 2006-2007,

[35] E. Magarotto, « Cours d'informatique industrielle ». Université de Caen, France, 2005-2006

[36] B. Benadda B. Beldjilali, « Travaux pratique en programmation assembleur des systèmes à base d'un processeur 8086 avec rappels de cours, corrigés et programmes types ». Université Abou Bekr Belkaid Tlemcen, 2016

[37] J.Y. Haggege, « Microprocesseur ». Support de cours, ISET de Radès, Tunisie, 2003

[38] A.B. Fontaine, « Le microprocesseur 16 bits 8086/8088 – matériel, logiciel, système d'exploitation ». Masson, Paris, 1988.

[39] B. Geoffrion, « 8086 - 8088 – Programmation en langage assembleur ». Editions Radio, Paris, 1986.

[40] A. Hmidene, « Microprocesseur 8086 ». Support de cours, institut supérieur des études technologiques de Sousse, 2009-2010.

[41] P. Preux, « Assembleur i8086 ». Support de cours, IUT informatique du littoral, France, 95 – 96.

[42] http://www.courstechinfo.be/Programmation/IntroASM.pdf

[43] E. Viennet, « Architecture des ordinateurs ». IUT de Villetaneuse, France, 1999-2000, viennet@lipn.univ-paris13.fr

[44] Brown, Ralf & Kyle, Jim, « PC Interrupts : A programmer's reference to BIOS, DOS, and Third-Party Calls, Addison-Wesley». Second edition 1994.

[45] Bradley, J. David, « Assembly Language Programming for the IBM Personal Computer, Prentice-Hall». Traduction française Assembleur sur IBM PC, Masson, 1986.

[46] C. Cauchois, « Assembleur d'INTEL ». Support de cours, Institut universitaire de technologie d'Amiens, France, 1999/2000

[47] R. Tourki, « L'ordinateur PC – architecture et programmation ». Cours et exercices, centre de publication universitaire, Tunis, 2002.

[48] R. Zaks et A. Wolfe, « Du composant au système – introduction aux microprocesseurs ». Sybex, Paris, 1988.