République algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Centre Universitaire Nour Bachir d'El-Bayadh
Faculté de technologie
Département génie électrique

Cours :

Microprocesseurs & Microcontrôleurs

Présenté par :
Dr REGUIEG Salima Khaoula

Cours destiné aux étudiants de Master I
Spécialité : Electrotechnique

Année : 2025

Avant-propos

Les microprocesseurs et les microcontrôleurs dépassent la simple description de composants numériques. En tant que base fondamentale des systèmes embarqués, ils occupent aujourd'hui une place centrale dans des domaines variés tels que l'électrotechnique, l'électronique de puissance, l'automatisation industrielle, la commande des entraînements électriques et l'acquisition de données. Leur présence est devenue incontournable dans les applications modernes : variateurs de vitesse, convertisseurs statiques, supervision, instrumentation, robotique, et systèmes intelligents.

L'objectif de cette polycopie est de fournir une compréhension progressive et approfondie des principes qui régissent le fonctionnement d'un microprocesseur et d'un microcontrôleur, ainsi que des méthodes de programmation associées. À travers l'étude des architectures internes, des mémoires, du jeu d'instructions, des interruptions et des interfaces d'entrées/sorties, l'étudiant pourra relier les concepts théoriques aux contraintes réelles des systèmes embarqués, notamment celles liées au temps réel, à la fiabilité et à l'intégration matérielle.

Ce document constitue une synthèse structurée du module « µ-processeurs et µ-contrôleurs » destiné aux étudiants de première année de Master (Électrotechnique). Des exemples concrets, choisis pour leur intérêt pédagogique, sont utilisés comme point de départ afin de favoriser l'appropriation des notions et de revenir, autant que possible, à l'interprétation fonctionnelle des mécanismes internes. L'étude de cas du microprocesseur **Intel 8086** permet d'illustrer les bases de l'architecture et de la programmation assembleur, tandis que l'exemple d'un microcontrôleur **PIC** met en évidence l'intégration des périphériques et la mise en œuvre d'applications embarquées.

Le polycopié « µ-processeurs et µ-contrôleurs » est divisé en cinq chapitres.

Le premier chapitre introduit les bases nécessaires à la compréhension des systèmes numériques, notamment les systèmes de numération, les codages de l'information et les représentations usuelles. Cette étape est essentielle pour manipuler correctement les données lors de la programmation bas niveau.

Le deuxième chapitre présente l'architecture et le fonctionnement général d'un microprocesseur : structure d'un calculateur, circulation de l'information, rôle des bus, organisation mémoire et principes d'exécution des instructions. Il constitue une passerelle entre la structure matérielle et la logique de programmation.

Le troisième chapitre est consacré à la programmation en assembleur, aux interruptions et aux interfaces d'entrées/sorties. Il met l'accent sur la méthode de programmation, le contrôle du flux d'exécution, ainsi que la prise en charge des événements asynchrones. L'exemple de l'Intel 8086 illustre la manipulation des registres, des modes d'adressage et des mécanismes d'interruption.

Le quatrième chapitre aborde l'architecture et le fonctionnement d'un microcontrôleur. Il présente les principaux blocs fonctionnels (CPU, mémoires, GPIO, timers, ADC, communication) et les étapes de programmation associées. L'exemple d'un microcontrôleur de la famille PIC permet d'illustrer la configuration des périphériques internes et la logique de développement embarqué.

Le cinquième chapitre regroupe des applications typiques des microprocesseurs et microcontrôleurs : interface LCD, clavier matriciel, génération de signaux par ports et PWM, commande pour convertisseurs et moteurs, mesure de fréquence et systèmes d'acquisition de données. Ces applications visent à développer chez l'étudiant une démarche complète allant du besoin fonctionnel à la réalisation matérielle et logicielle.

Nous espérons que cette polycopie aidera les étudiants à acquérir une base solide en architecture et programmation des systèmes embarqués, et à mieux comprendre le rôle des microprocesseurs et microcontrôleurs dans les applications électrotechniques modernes.

Sommaire

Avant-propos **1**
Introduction générale **3**
Chapitre 1 : Notions sur les systèmes de numérisation et les codes **6**
1. Introduction 6
2. Système de numération 6
2.1 Définitions 6
2.2 Système décimal 6
2.3 Système binaire 6
2.4 Système octal 7
2.5 Système hexadécimal 7
3. Changement de système de numération 8
3.1 Conversion binaire, octal et hexadécimal 8
3.2 Conversion décimal en binaire, octal et hexadécimal 8
4. Codage binaire 10
4.1 Code DCB 10
4.2 Codes ASCII 11
4.3 Codage des nombres négatifs 11
4.3.1 Signe et valeur absolue 11
4.3.2 Complément à 1 12
4.3.3 Complément à 2 12
5. Arithmétique binaire 12
5.1 Addition binaire 12
5.2 Soustraction binaire 13
5.3 Multiplication binaire 13
5.4 Division binaire 13
6. Les opérations booléennes 13
6.1 Le complément logique 13
6.2 Le ET (AND) logique 14
6.3 Le OU (OR) logique 14
6.4 Le OU-exclusif (XOR) 14
TD : 01 15
Exercice 1 15
Exercice 2 15
Exercice 4 15
Exercice 5 15

Exercice 6 15
Chapitre 2 : Architecture des systèmes à base de microprocesseur 16
1. Définitions 16
2. Architecture de base d'un ordinateur 16
2.1 Modèle de Von Neumann 16
2.2 L'unité centrale 17
2.3 La mémoire principale 17
2.4 Les interfaces d'entrées/sorties 17
2.5 Les bus 17
3. Le microprocesseur 18
3.1 L'unité de commande 18
3.2 L'unité de traitement 19
4. Cycle d'exécution d'une instruction 21
4.1 Phase 1 : Recherche 21
4.2 Phase 2 : Décodage 21
4.3 Phase 3 : Exécution 21
5. Jeu d'instructions 21
6. Mode d'adressage 22
7. Les mémoires 22
7.2 Caractéristiques d'une mémoire 23
7.3 Classification des mémoires 24
ROM 24
PROM 24
EPROM 24
EEPROM 25
RAM 25
RAM statiques 25
RAM dynamiques 25
8. Notion de hiérarchie mémoire 26
9. Notion d'architecture RISC et CISC 27
9.1 L'architecture CISC 27
9.2 L'architecture RISC 27
Chapitre 3 : Architecture interne du microprocesseur Intel 8086 28
1. Définitions 28
2. Architecture externe du 8086 28
3. Architecture interne du 8086 29
3.1 Fonctions de l'UIB 29
3.2 Fonctions de l'UE 30

3.3 Les registres du 8086 30
3.3.1 Registres de travail 30
3.3.2 Registres de segments 30
3.3.3 Registre d'état FLAG 31
4. Les modes d'adressages 33
Adressage registre 33
Adressage immédiat 34
Adressage direct 34
Adressage indirect par registre 34
Adressage basé 34
Adressage indexé 35
Adressage basé indexé 35
Adressage des entrées/sorties (I/O) 35
Adressage relatif 36
Adressage vectorisé 37
Les instructions de transfert 37
Les instructions arithmétiques 31
Les instructions logiques 39
Les instructions de décalage et de rotation 40
Instructions de branchement 42
Branchements inconditionnels 42
Branchements conditionnels 42
Instructions de boucle 43
Chapitre 4 : Architecture et fonctionnement d'un microcontrôleur 44
1. Introduction : du microprocesseur au microcontrôleur 44
2. Différences microprocesseur/microcontrôleur et architectures internes 44
3. Blocs fonctionnels d'un microcontrôleur 45
3.1 CPU, registres et pipeline 46
3.2 Mémoires 46
3.3 Ports E/S 46
3.4 Timers et PWM 46
3.5 Convertisseur A/N (ADC) 47
3.6 Communication 47
4. Exemple : microcontrôleur PIC 47
4.1 Ressources matérielles 48
4.2 Organisation mémoire et registres SFR 48
4.3 Interruptions 48
4.4 Horloge, watchdog et configuration 49

5. Programmation d’un microcontrôleur PIC 49
5.1 Exemple de configuration 50
Chapitre 5 : Applications des microprocesseurs et microcontrôleurs 51
1. Introduction 51
2. Interface LCD 51
2.1 Séquence d’initialisation et timing 52
2.2 Commandes usuelles 52
2.3 Exemple de driver 52
3. Interface clavier matriciel 52
3.1 Anti-rebond et décodage 53
3.2 Algorithme de scan 53
4. Génération de signaux : timers et PWM 53
4.1 Choix de la fréquence PWM 54
4.2 PWM et temps mort 54
5. Interfaces pour convertisseurs 54
6. Commande de moteurs 55
7. Mesure de fréquence 55
7.1 Exemple de calcul 55
8. Système d’acquisition de données 56
Conclusion générale 57
Références bibliographiques 59

Liste des abréviations

Abréviation	Signification
µP	Microprocesseur
µC	Microcontrôleur
CPU	Central Processing Unit (Unité centrale de traitement)
ALU/UAL	Arithmetic Logic Unit (Unité arithmétique et logique)
PC	Program Counter (Compteur ordinal)
IR	Instruction Register (Registre d'instruction)
RAM	Random Access Memory (Mémoire vive)
ROM	Read-Only Memory (Mémoire morte)
EEPROM	Electrically Erasable Programmable ROM
Flash	Mémoire non volatile reprogrammable
GPIO	General Purpose Input/Output (Entrées/sorties générales)
SFR	Special Function Registers (Registres de fonctions spéciales)
GPR	General Purpose Registers (Registres généraux)
ISR	Interrupt Service Routine (Routine d'interruption)
ADC	Analog-to-Digital Converter (Convertisseur analogique-numérique)
DAC	Digital-to-Analog Converter (Convertisseur numérique-analogique)
PWM	Pulse Width Modulation (Modulation de largeur d'impulsion)
UART/USART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
I²C	Inter-Integrated Circuit
WDT	Watchdog Timer
ICSP	In-Circuit Serial Programming

INTRODUCTION GENERALE

La maîtrise des microprocesseurs et des microcontrôleurs occupe aujourd'hui une place essentielle dans de nombreux domaines scientifiques et technologiques, notamment l'électrotechnique, l'électronique, l'automatisation industrielle, l'instrumentation, la robotique et les systèmes embarqués. Ces composants numériques, devenus omniprésents, assurent des fonctions indispensables dans les systèmes modernes : acquisition de mesures, traitement des signaux, commande d'actionneurs, supervision, communication et gestion temps réel. Leur importance ne se limite donc pas à la simple exécution d'un programme : ils constituent le cœur décisionnel de nombreuses installations industrielles et applications à forte valeur ajoutée.

L'utilisation et la mise en œuvre de ces dispositifs intéressent plusieurs catégories de professionnels, parmi lesquels les ingénieurs en électrotechnique et électronique, les ingénieurs automaticiens, les spécialistes de l'électronique de puissance, les ingénieurs en instrumentation et mesures, ainsi que les ingénieurs de maintenance et d'exploitation. Dans l'industrie, la conception de cartes de commande, la supervision de systèmes, la maintenance préventive et corrective, ou encore l'optimisation énergétique de processus reposent très souvent sur l'analyse et la compréhension du fonctionnement interne d'un microprocesseur ou d'un microcontrôleur.

La conception et la réalisation de systèmes de commande et de contrôle représentent ainsi un objectif central pour l'ingénieur : il ne s'agit pas uniquement de programmer, mais également de comprendre l'architecture matérielle, l'organisation mémoire, le rôle des registres, la circulation de l'information, la gestion des interruptions et les interfaces d'entrées/sorties. De plus, les contraintes liées au temps réel, à la fiabilité et aux perturbations électriques imposent une approche rigoureuse de la programmation et du dimensionnement des interfaces.

Il est fortement recommandé aux étudiants de chercher à comprendre les applications associées à chaque chapitre. L'étude de ce document serait considérablement enrichie si elle était accompagnée d'une activité pratique (TP ou mini-projets), permettant de relier les concepts théoriques à des observations expérimentales : mesure de signaux au moyen d'un oscilloscope, validation des timings, tests d'interruptions, génération PWM, communication série, etc.

Les étudiants engagés dans la lecture de ce cours devraient posséder des connaissances préalables dans plusieurs domaines : notions de base en électronique numérique, architecture des systèmes, programmation, ainsi que des bases en électrotechnique et électronique de puissance. Des connaissances en capteurs et mesures, ainsi que des notions de systèmes embarqués, faciliteront la compréhension des applications proposées, en particulier celles orientées commande et acquisition.

L'objectif de ce document est de fournir une théorie suffisante et progressive pour permettre une compréhension approfondie des concepts fondamentaux, ainsi que de leurs usages dans des applications réelles. Le cours est organisé en cinq chapitres complémentaires :

- **Le premier chapitre** présente les bases nécessaires au traitement numérique, notamment les systèmes de numération, le codage de l'information et la représentation des données. Ces notions sont indispensables pour comprendre la manipulation des registres et des instructions lors de la programmation bas niveau.
- **Le deuxième chapitre** aborde l'architecture et le fonctionnement d'un microprocesseur : structure d'un calculateur, circulation de l'information, rôle des bus, organisation mémoire et principes d'exécution des instructions. Il établit le lien entre la structure matérielle et la logique de fonctionnement.
- **Le troisième chapitre** traite la programmation en assembleur ainsi que la gestion des interruptions et des interfaces d'entrées/sorties. Il met l'accent sur la méthode de programmation, la structure des programmes, l'adressage des ports et les mécanismes permettant de traiter des événements asynchrones, indispensables en temps réel.
- **Le quatrième chapitre** est consacré à l'architecture et au fonctionnement d'un microcontrôleur, en présentant ses blocs internes (CPU, mémoires, ports, timers, ADC, communication) et les étapes essentielles de programmation. L'exemple d'un microcontrôleur de la famille PIC permet d'illustrer les concepts et de relier théorie et pratique.
- **Le cinquième chapitre** regroupe des applications représentatives : interface LCD et clavier, génération de signaux (ports, timers, PWM), commande de convertisseurs et de moteurs, mesure de fréquence et systèmes d'acquisition de

données. Ce chapitre vise à développer chez l'étudiant une démarche complète allant de l'analyse du besoin à la validation expérimentale.

À la fin de ce document, vous trouverez une liste de références bibliographiques pour approfondissement. Dans la mesure du possible, les étudiants devraient les consulter afin de compléter la couverture de ce texte, obtenir des approches alternatives, approfondir certains sujets (assembleur, architectures, systèmes embarqués) et trouver des exercices supplémentaires.

Ce cours est mis à disposition des étudiants via la plateforme d'enseignement à distance, afin de faciliter l'accès aux supports, aux compléments et aux documents de référence.

https://elearn.cu-elbayadh.dz/moodle/course/view.php?id=502

Chapitre 1
Notions sur les systèmes de numérisation et les codes

1. Introduction

Le système de numération binaire et les codes numériques sont essentiels en informatique et en électronique numérique. Le terme numérique vient du principe de fonctionnement des ordinateurs pour exécuter les opérations. Pendant plusieurs années, les applications numériques se limitaient aux systèmes d'ordinateurs. Aujourd'hui, la technologie numérique sert une grande variété de domaines, avec des applications comme la télévision, les systèmes de communication, le radar, les systèmes militaires, l'instrumentation médicale, le contrôle de procédés industriels et l'électronique domestique.

2. Système de numération

2.1 Définitions

On utilise aujourd'hui des systèmes de numération pondérés. La définition d'un système de numération pondéré repose sur trois notions :

- ✓ La base du système : c'est un nombre entier, noté B ;
- ✓ Les digits du système : ce sont des caractères, tous différents, représentant chacun un élément de la base. Il y en a donc B au total ;
- ✓ Le poids de chaque digit selon son rang (sa position). Compté de la droite vers la gauche, ce poids est 0 B (c'est-à-dire1) pour le premier digit, 1 B (c'est-à-dire B) pour le second digit, 2 B pour le troisième digit, etc…

2.2 Système Décimal

Dans ce système, la base est 10. Il y a 10 digits notés : 0, 1, 2, 3, 4, 5, 6,7, 8,9. Exemple : $(195.28)_{10} = 1\times10^{2} + 9\times10^{1} + 5\times10^{0} + 2\times10^{-1} + 8\times10^{-2}$

2.3 Système Binaire

Dans le système binaire, la base est 2 . Il y a 2 digits notés : 0 et 1.

On trouve la valeur décimale de tout nombre en additionnant le poids de tous les bits dont le chiffre est 1 et en ignorant le poids des bits dont le chiffre est 0.

Exemples :

$(1101)_2 = 1\times2^{3} + 1\times2^{2} + 0\times2^{1} + 1\times2^{0} = 13$

$(101.01)_2 = 1\times2^{2} + 0\times2^{1} + 1\times2^{0} + 0\times2^{-1} + 1\times2^{-2} = 5.25.$

Le nombre décimal maximal pouvant être obtenu en utilisant n bits est égale à 2^n -1. Par exemple, nous pouvons compter de 0 à 255 avec huit bits : 2^8-1=256-1=255.

Vocabulaire

Toute information est stockée, dans un ordinateur, sous forme de **BIT** (**BI**nary digi**T**). Ces informations sont regroupées par blocs de 8 bits nommés **OCTETs** (**BYTE** en anglais) ou par blocs de 16 bits nommés **MOT**s (**WORD** en anglais) ou même par blocs de 32 bits nommés **MOT**s **LONG**s (**LONG WORD**).

Un **bit** est un digit du système binaire.

Un **mot (word)**est un ensemble de bits

Un **quartet** est un mot de 4 bits

Un **octet** (**byte**) est un mot de 8 bits.

1**KO**(kilo octet)=2^{10}octect=1024octect

1**MB** (Méga byetes)=1024KB

1**GO** (Géga octet)=1024MO

1k (kilo) =2^{10}=1024 ; 1M (méga) =2^{20} =1024**k**

1G (giga) = 2^{30} = 1024**M** ; 1T (térra) = 2^{40} =1024**G**

Le bit de poids faible **L.S.B**. (**L**ess **S**ignificant **B**it) est le bit situé le plus à droite.

Le bit de poids fort **M.S.B**. (**M**ost **S**ignificant **B**it) est le bit situé le plus à gauche.

2.4 Système Octal

Dans le système octal, la base est . Il y a digits notés : . 8 8 7,6, 5, 4, 3, 2, 1, 0 Exemples :

$(254)_8 = 2\times8^2+5\times8^1+4\times8^0=128+40+4=172$

2.5 Système Hexadécimal

Le système de numération hexadécimal possède une base de 16, il comprend seize symboles :1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. L'usage du système hexadécimal est très répondu dans la programmation en langage machine d'ordinateurs et de microprocesseurs. Il est beaucoup plus simple de traiter un nombre hexadécimal que son équivalent binaire.

Exemples :

$(1A5F)_{16}=1\times16^3+10\times16^2+5\times16^1+15\times16^0=6751$:

$(5F.2)_{16}= 5\times16^1+15\times16^0+5\times16\text{-}^1=95.125$

3. Changement de système de numération

3.1 Conversion binaire – octal – hexadécimal

Elle est simple du fait que : $8 = 2^3$ → groupe de 3 bits

$16 = 2^4$ → groupe de 4 bits

On groupe les bits par 3 (octal) ou par 4 (hexadécimal) en partant de la droite pour la partie entière et de la gauche pour la partie fractionnaire. Ceci produit une condensation de l'écriture, la représentation physique restant la même.

Exemple :

$$\begin{array}{ll} 1\ \ 5\ \ E & \rightarrow (15E)_h \\ \overline{(101011110)}_2 & \\ \overline{5}\ \ \overline{3}\ \ \overline{6} & \rightarrow (538)_8 \end{array}$$

En octal :

$(1547)_8=(001)(101)(100)(111)=(1101100111)_2$

La conversion inverse se fait de la même manière, en partageant le nombre binaire en ensemble de 3 bits à partir de la droite.

$(110010100)_2=(011)(010)(110)(100)=(3264)_8$

En hexadécimal :

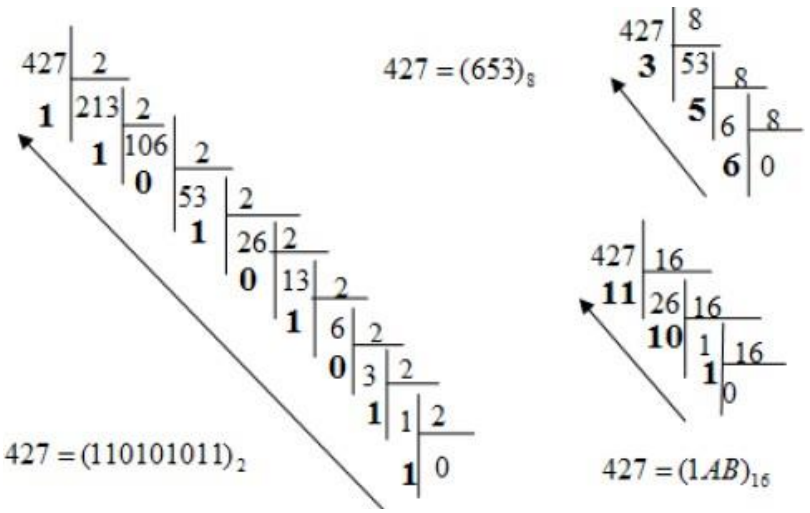
$(6B3)_{16}=(0110)(1011)(0011)=(11010110011)_2$

La conversion inverse se fait de la même manière, en partageant le nombre binaire en ensemble de 4 bits à partir de la droite.

$(1101011101101)_2=(0001)(1010)(1110)(1101)=(1AED)_{16}$

3.2 Conversion Décimal en Binaire, Octal et Hexadécimal

Il suffit d'effectuer des divisions entières successives du nombre par la base, puis du quotient obtenu par la base, puis du nouveau quotient par la base, … jusqu'à ce que le quotient devienne nul. L'expression recherchée est constituée par l'ensemble des restes successifs des divisions, lu à l'envers, comme on le voit ci-dessous :



Les fractions décimales peuvent également être converties en binaire, en le multipliant par 2 de façon répétitive, jusqu'à ce que la partie fractionnaire donne 0. Les chiffres reportés ou retenues, générés par les multiplications, forment le nombre binaire. La première retenue produite devient le **MSB** et la dernière retenue devient le **LSB**.

Exemple : Retenue

↓

0.3125×2=0.625 → 0

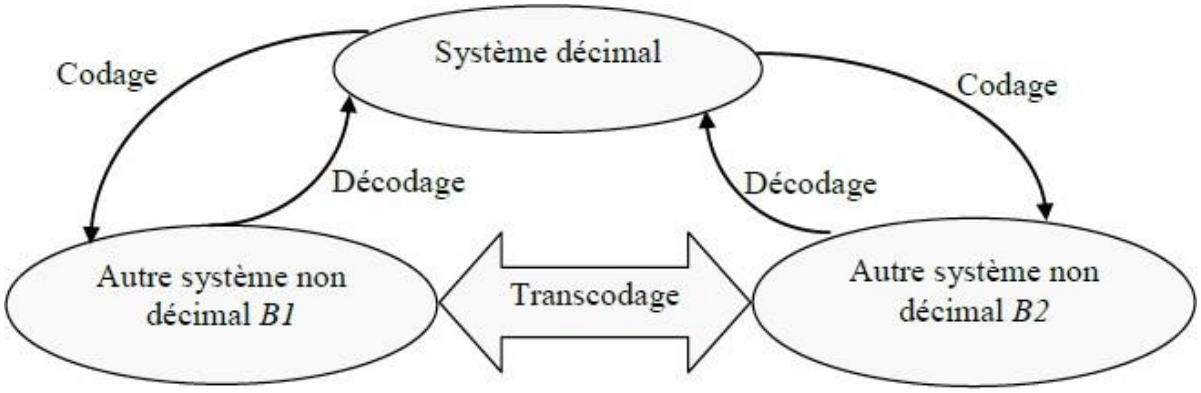
0.625×2=1.25 → 1

0.25×2=0.5 → 0

0.5×2=1.00 → 1

Alors : $0.3125=(0.0101)_2$

Nombres décimaux	Représentation binaire				En Hexa
	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	1	0	1	0	A
11	1	0	1	1	B
12	1	1	0	0	C
13	1	1	0	1	D
14	1	1	1	0	E
15	1	1	1	1	F



4. Codage binaire

✓ **Code DCB (Décimal codé binaire, anglais BCD)**

Chaque chiffre décimal (0, …, 9) est représenté par un groupe de quatre bits en binaire. Un nombre BCD est donc formé de groupes de 4 bits indépendants, qui regroupés, ne forment pas de valeur binaire.

Exp :

$(45)_{10} = (0100\ 0101)_{BCD}$

$(5609)_{10} = (0101\ 0110\ 0000\ 1001)_{BCD}$

✓ Codes ASCII

Le code ASCII (American Standard Code for Information Interchange) est appelé aussi code ISO (International Organization for Standerdization). permet de coder les 26 lettres de l'alphabet (majuscules et minuscules), les 10 chiffres et les signes de ponctuation : il utilise un octet (8 bits). Cet octet donne une certaine souplesse d'utilisation, puisqu'il permet de coder des commandes de contrôle en plus des caractères alphanumériques (bits 1 à7)

Codes *ASCII*
0 à 127

					bit b7	0	0	0	0	1	1	1	1
					bit b6	0	0	1	1	0	0	1	1
					bit b5	0	1	0	1	0	1	0	1
				Décimal		0	16	32	48	64	80	96	112
b4	*b3*	*b2*	*b1*		*Hexa*	0	1	2	3	4	5	6	7
0	0	0	0	0	0	*NUL*	*DLE*	*SP*	0	@	*P*	`	*p*
0	0	0	1	1	1	*SOH*	*DC1*	!	1	*A*	*Q*	*a*	*q*
0	0	1	0	2	2	*STX*	*DC2*	"	2	*B*	*R*	*b*	*r*
0	0	1	1	3	3	*ETX*	*DC3*	#	3	*C*	*S*	*c*	*s*
0	1	0	0	4	4	*EOT*	*DC4*	$	4	*D*	*T*	*d*	*t*
0	1	0	1	5	5	*ENQ*	*NAK*	%	5	*E*	*U*	*e*	*u*
0	1	1	0	6	6	*ACK*	*SYN*	&	6	*F*	*V*	*f*	*v*
0	1	1	1	7	7	*BEL*	*ETB*	'	7	*G*	*W*	*g*	*w*
1	0	0	0	8	8	*BS*	*CAN*	(	8	*H*	*X*	*h*	*x*
1	0	0	1	9	9	*HT*	*EM*	)	9	*I*	*Y*	*i*	*y*
1	0	1	0	10	*A*	*LF*	*SUB*	*	:	*J*	*Z*	*j*	*z*
1	0	1	1	11	*B*	*VT*	*ESC*	+	;	*K*	[	*k*	{
1	1	0	0	12	*C*	*FF*	*FS*	,	⟨	*L*	\	*l*	|
1	1	0	1	13	*D*	*CR*	*GS*	–	=	*M*	]	*m*	}
1	1	1	0	14	*E*	*SO*	*RS*	.	⟩	*N*	^	*n*	~
1	1	1	1	15	*F*	*SI*	*US*	/	?	*O*	_	*o*	*DEL*

Exemple :

$K=(4B)_{16}=75=64+11=(1001011)_2$

$z=(7A)_{16}=122=112+10=(1111010)_2$

✓ Codage des nombres négatifs

Le bit de poids fort (**M**SB : Most **S**ignicant **B**it) est bit de signe. Il indique le signe du nombre : 0 → positif ; 1 →négatif. Il existe trois modes de représentation.

1. Signe et valeur absolue

Le MSB est le bit e signe, les autres bits indiquent la valeur absolue des nombres.

Exp :

(+17)= 0 10001

(-17)= 1 10001 et non $(49)_{10}$

Cette représentation produit une bonne lisibilité, mais nécessite un transcodage pour les calculs arithmétiques.

2. Complément à 1

La notation en complément à 1 représente les nombres positifs de façon identique à la notation **signe/valeur absolue**. Par contre, les nombres négatifs deviennent les compléments à 1 de leurs nombres positifs correspondants. Le complément à 1 d'un nombre binaire s'obtient en changeant chaque 0 par un 1 et chaque 1 par un 0.

Exp :

(+17)= 0 10001

(-17)= 1 01110 et non $(46)_{10}$ exprimé en base 2.

3. Complément à 2

On obtient le complément à 2 d'un nombre binaire en additionnant 1 au bit le moins significatif (LSB) du complément à 1 de ce nombre.

Exp :

(+17)= 0 10001

(-17)= 1 01110 → (complément à 1)

\+ 1

(-17)= 1 01111 → (complément à 2) et non (47) en base 2.

Et sur 8 bites

(+17)= 00010001

(-17)= 10010001 → (signe/valeur absolue)

(-17)= 11101110 → (complément à 1)

\+ 1

(-17)= 11101111 → (complément à 2)

5. Arithmétique binaire

L'arithmétique binaire est essentielle dans tous les ordinateurs. Pour comprendre le fonctionnement de ces systèmes, il fait connaître les notions de base de l'addition, de la soustraction, de multiplication et de la division binaires.

- **Addition binaire**

Les quatre règles de base de l'addition de nombres binaires sont les suivantes{ $0+0=0$, $0+1=1$, $1+0=1$, $1+1=10$

Dans la quatrième règle, on trouve une somme égale à 0 et une retenue égale à 1.

Exemple : Additionner les nombres binaires suivants 11+11 ;111+11 ;110+100.

Solution : 11+11=110 ; 111+11=1010 ; 110+100=1010.

- **Soustraction binaire**

Les quatre règles de base de la soustraction binaire sont les suivantes : { $0-0=0$, $0-1=1$, $1-0=1$, $1-1=0$ avec un emprunt de 1

Exemple : Effectuer les soustractions binaires suivants : 11-01 ;11-10 ; 101-011 ;

Solution : 11-01=10 ;11-10=01 ; 101-011=010.

- **Multiplication binaire**

Les quatre règles de base de multiplication binaires sont les suivantes: { $0\times0=0$, $0\times1=0$, $1\times0=0$, $1+1=1$

On multiplie les nombres binaires de la même façon qu'on multiplie les nombres décimaux.

Exemple : Effectuer les multiplications binaires suivantes : 11×11 ;101×111 ;1101×1010.

Solution : 11×11=1001 ;101×111=100011 ;1101×1010=10000010.

- **Division binaire**

La division de nombres binaires est identique à celle de nombres décimaux.

Exemple : Effectuer les divisions binaires suivantes :110÷11 ;110÷10 ;11000÷100.

Solution : 110÷11=10;110÷10=11 ; 11000÷100=11.

6. Les opérations booliennes :

✓ Le complément logique

C'est la négation ou l'inversion logique, ou le NOT en anglais, mais plus précisément le complément à 1. Elle est souvent notée « ! » devant le nombre à inverser. Son fonctionnement tout simple consiste à inverser tous les bits de l'octet. Des 1 deviendront des 0 et vis versa. A titre d'exemple, le NOT B'10001111' donne 'B01110000 '. Il est à noter que pour simplifier, avant de complémenter à 1 un nombre quelconque, il faut avant tout le coder en binaire. Le complément sert généralement à lire une valeur dont les niveaux actifs ont été inversés dont il faut l'inverser pour avoir la vraie valeur, à réaliser des nombres négatifs …etc.

- ✓ **Le ET (AND) logique**

Un ET logique, AND et souvent noté &, est la multiplication bit à bit. Elle se réalise entre deux mots et elle multiplie chaque bit du premier mot par le bit du même rang du second mot. D'où pour réaliser un opération ET logique, il faut disposer de deux octets. La table de vérité d'une telle opération est présentée dans le tableau ci-après :

Bit1	Bit2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Exemple/ b'11001100' AND b'11110000' donne b'11000000'.

- ✓ **Le OU (OR) logique**

Cette opération, notée généralement « | », manipule aussi deux octets et permet de positionner le résultat à 1 si l'un des deux bits de même rang dans les deux octets est à1. Donc le bit du résultat est à 1 si le Bit1 OU le Bit2 est à 1.

Bit1	Bit2	OR
0	0	0
0	1	1
1	0	1
1	1	1

- ✓ **Le OU-EXCUSIF (Exclusif OR) ou XOR**

Le XOR (eXclusif OR) assume la même fonction logique OR avec un détail en plus : Pour obtenir 1, il faut que le Bit1 soit à 1 OU que le Bit2 soit à 1 à l'EXCLUSION des deux bits ensemble. Donc si les 2 bits sont à 1, alors le résultat sera 0 conformément à la table de vérité ci-après. Elle opère sur deux octets.

Bit1	Bit2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Comme exemple b'10001000' XOR b'11000000' donne le résultat b'01001000'.

TD :01

Exercice 1

1. Représentez les nombres 2810, 12910, 14710, 25510 sous leur forme binaire, en déduirez leur représentation hexadécimale.
2. Convertir en décimal les nombres suivants: $(01001011)_2$, $(1245)_8$, $(3C5)_{16}$, $(1001\ 1000)_{BCD}$.

Exercice 2

1- Quel est le plus grand nombre que l'on peut exprimer avec 8 chiffres ? On l'exprimera dans sa base :

a. dans le système décimal.

b. dans le système binaire.

c. dans le système hexadécimal.

2- Quelle est la valeur décimale qui correspond à la valeur BCD : 1000 0110 0001?

Exercice 4

Soit les deux nombres binaires suivants : 11111111_2 et 10110110_2.

1. Donnez leur représentation décimale s'ils sont codés sur 8 bits signés.
2. Donnez leur représentation décimale s'ils sont codés sur 16 bits signés.

Exercice 5

Quelle est le résultat des opérations suivantes :

1. 11010+01011
2. 10110-10001
3. 1010*101
4. 1100/011

Exercice 6

Soit le nombre entier négatif suivant : -80_{10}.

On souhaite le coder sur 8 bits signés. Donnez sa représentation binaire et sa représentation hexadécimale.

.

Chapitre 2 : Architecture des Systèmes à Base de Microprocesseur

Objectifs

À l'issue de ce chapitre, l'étudiant doit être capable de :

- décrire la structure d'un calculateur (CPU, mémoire, bus, E/S) ;
- expliquer le cycle d'exécution d'une instruction (fetch–decode–execute) ;
- distinguer les types de mémoires et leurs rôles (RAM, ROM, Flash) ;
- analyser une architecture type (ex. Intel 8086) à partir de ses blocs fonctionnels.

1. Définitions

Le mot ***informatique*** vient de la contraction des mots information et automatique. On peut définir l'informatique comme étant le traitement automatique de l'information. Nous appelons ***information*** tout ensemble de données. On distingue généralement différents types d'informations : textes, nombres, sons, images, etc., mais aussi les instructions composant un programme. Dans un ordinateur, toute information est manipulée sous forme binaire (ou numérique).

2. Architecture de base d'un ordinateur

2.1 Modèle de Von Neumann

Pour traiter une information, un microprocesseur seul ne suffit pas, il faut l'insérer au sein d'un système minimum de traitement programmé de l'information. John Von Neumann est à l'origine d'un modèle de machine universelle de traitement programmé de l'information (1946). Cette architecture (figure 1) sert de base à la plupart des systèmes à microprocesseur actuel. Elle est composée des éléments suivants :

- ✓ une unité centrale
- ✓ une mémoire principale
- ✓ des interfaces d'entrées/sorties.

Les différents organes du système sont reliés par des voies de communication appelées bus.

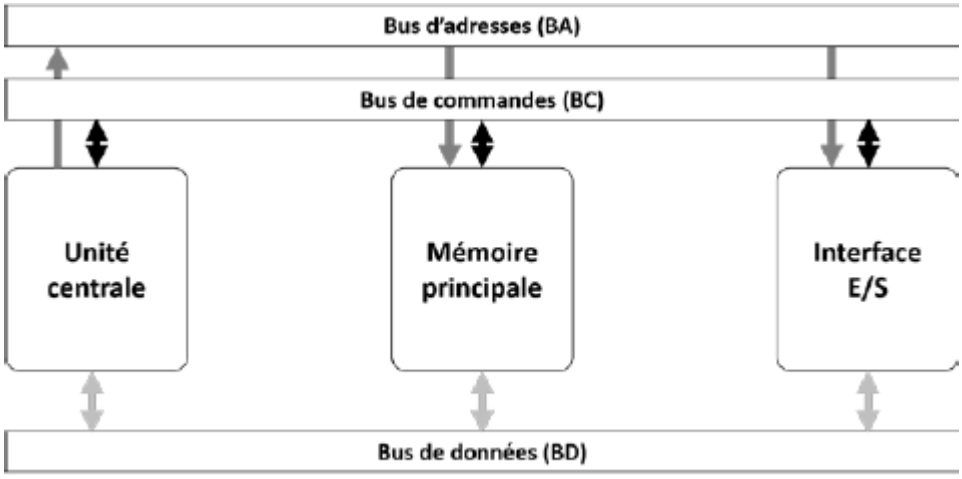


***Figure 1** : Modèle de Von Neumann.*

2.2 L'unité centrale

Elle est composée par le microprocesseur qui est chargé d'interpréter et d'exécuter les instructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire et de communiquer avec les unités d'échange. Toutes les activités du microprocesseur sont cadencées par une horloge. On caractérise le microprocesseur par :

- ✓ La vitesse d'exécution des instructions liée à la fréquence de fonctionnement de l'horloge de cadencement exprimée en MHZ.
- ✓ Le nombre d'octets que le microprocesseur peut adresser (on parle de largeur du bus des adresses).
- ✓ La longueur des mots : données et instructions (on parle de largeur du bus des données).

2.3 La mémoire principale

Elle contient les instructions du ou des programmes en cours d'exécution et les données associées à ce programme. Physiquement, elle se décompose souvent en :

- ✓ **une mémoire morte** (ROM = Read Only Memory) chargée de stocker le programme. C'est une mémoire à lecture seule.
- ✓ **une mémoire vive** (RAM = Random Access Memory) chargée de stocker les données intermédiaires ou les résultats de calculs. On peut lire ou écrire des données dedans, ces données sont perdues à la mise hors tension.

Remarque : Les disques durs, disquettes, CDROM, etc… sont des périphériques de stockage et sont considérés comme des mémoires secondaires.

2.4 Les interfaces d'entrées/sorties

Elles permettent d'assurer la communication entre le microprocesseur et les périphériques. (Capteur, clavier, moniteur ou afficheur, imprimante, modem, etc…).

2.5 Les bus

Un bus est un ensemble de fils qui assure la transmission du même type d'information. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l'information :

1) **un bus de données** : bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement, et inversement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.

2) **un bus d'adresses** : unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire (ou espace adressable) qui peut avoir 2n emplacements, avec n = nombre de lignes du bus d'adresses.
3) **un bus de commande** : constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses.

3. Le microprocesseur (µP)

Un µP est un composant électronique, numérique et programmable.

Electronique : un µP est un circuit intégré constitué par les composants électroniques de bases.

Numérique : les entrées et les sorties aussi que les commandes au niveau du µP sont de nature numérique.

Programmable : c'est cette qualité qui rend le µP intelligent. On peut modifier le comportement d'un système à base de µP en modifiant le programme qu'il exécute.

Un microprocesseur est construit autour de deux éléments principaux :

a) Une unité de commande
b) Une unité de traitement

Ces unités sont associées à des registres chargées de stocker les différentes informations à traiter. Ces éléments sont reliés entre eux par des bus interne permettant les échanges d'informations (figure 2).

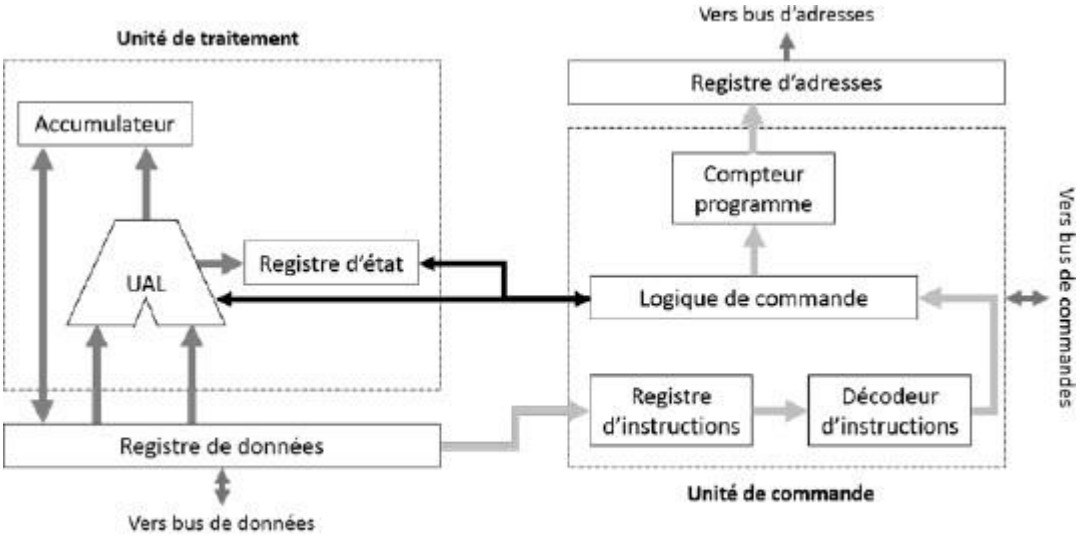


***Figure 2** : Architecture de base d'un µP.*

3.1. L'unité de commande

C'est l'unité de control qui supervise le déroulement de toutes les opérations au sein du processeur. Elle est constituée principalement de :

- ✓ **l'horloge :**

C'est l'horloge qui génère les signaux qui permettent le cadencement et la synchronisation de toutes les opérations. Attention, l'horloge n'est pas une montre au sens commun du terme, c'est juste un signal carré qui a une fréquence fixe, à chaque coup (front) d'horloge, le microprocesseur (qui ne l'oublions pas n'est qu'un circuit électronique) réalise une tache élémentaire. L'exécution d'une instruction nécessite plusieurs coups d'horloges.

- ✓ **Le compteur programme PC :**

Le compteur programme (PC : program counter) est un registre (pointeur) qui contient l'adresse de la case mémoire où est stockée le prochain élément d'instruction qui devra être chargé dans le processeur pour être analysé et exécuté. Au début de l'exécution d'un programme, le PC est initialisé par le système d'exploitation à l'adresse mémoire où est stockée la première instruction du programme. Le compteur programme est incrémenté automatiquement chaque fois qu'un élément d'instruction est chargée dans le processeur.

- ✓ **Le registre d'instruction RI**

C'est là où le CPU stocke l'instruction en cours d'exécution.

- ✓ **Le décodeur**

C'est lui qui va "décoder" l'instruction contenue dans RI et générer les signaux logiques correspondant et les communiquer au séquenceur.

- ✓ **Bloc logique de commande (Le séquenceur)**

Il gère le séquencement des opérations et génère les signaux de commande qui vont activer tous les éléments qui participeront à l'exécution de l'instruction et spécialement l'ALU.

3.2 L'unité de traitement

- ➢ **L'Unité Arithmétique et Logique(UAL)**

C'est un circuit complexe qui assure les fonctions logiques (ET, OU, comparaison …) ou arithmétiques (addition, division, soustraction, …). 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.

Comme indiqué sur (la figure 03), l'entrée de L'UAL est connectée au bus interne par un ensemble de registres "temporaires" et aussi par un registre particulier appelé "accumulateur" ou plus particulièrement le registre de travail (Working Register, le Registre W). Alors que la sortie de l'UAL est connectée uniquement à l'entrée de l'accumulateur. Il est à noter que les deux entrées sont précédées par

une mémoire tampon appelée registres tampons ou verrous. Ces registres, qui ne peuvent être manipulés par le programmeur et le sont totalement transparents, permettent de stocker des octets aux entrées de l'U.A.L. comme l'UAL est constitué d'une logique combinatoire, elle est dépourvue de moyens propres de stockage.

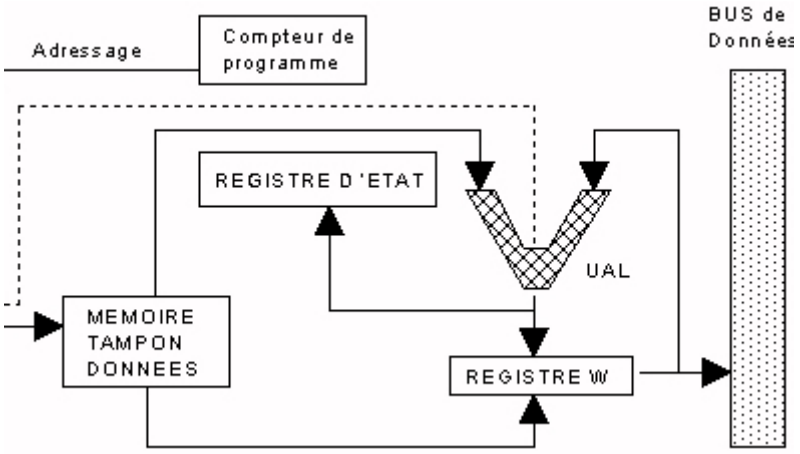


Figure 03 : schéma d'une UAL

- **L'accumulateur ou le registre de travail**

Le processeur utilise toujours des ***registres***, qui sont des petites mémoires internes très rapides d'accès utilisées pour stocker temporairement une donnée, une instruction ou une adresse. Chaque registre stocke 8,

16 ou 32 bits. Le nombre exact de registres dépend du type de processeur et varie typiquement entre une dizaine et une centaine. Parmi les registres, le plus important est le registre ***accumulateur,*** qui est utilisé pour stocker les résultats des opérations arithmétiques et logiques. L'accumulateur intervient dans une proportion importante des instructions.

C'est le registre le plus important du microprocesseur, il sert systématiquement lorsque ce dernier aura besoin de "manipuler" des données. La plupart des opérations logiques et arithmétiques sur les données font appel au couple "UAL - accumulateur".

Ce même registre de travail aussi sert pour les déplacements et les transferts des données d'un endroit à un autre : de mémoire à mémoire ou de mémoire à unités d'entrée-sortie (I/O). Ce type d'actions se fait en deux temps : source vers Accumulateur et ensuite Accumulateur vers destination.

- **Le registre d'état**

Le registre d'état est formé de plusieurs bits appelés drapeaux ou indicateurs (Flags) qui sont positionnés par l'ALU après chaque opération. Par exemple l'indicateur Z indique quand il est positionné

que le résultat de l'opération est égal à Zéro. L'indicateur C indique que l'opération a généré une retenue. Le bit N indique que le résultat est négatif …

4. Cycle d'exécution d'une instruction

Le microprocesseur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire. Le traitement d'une instruction peut être décomposé en trois phases.

4.1 Phase 1 : Recherche de l'instruction à traiter

1. Le PC (compteur programme) contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande qui émet un ordre de lecture.

2. Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire sélectionnée est disponible sur le bus des données.

3. L'instruction est stockée dans le registre instruction du processeur.

4.2 Phase 2 : Décodage de l'instruction et recherche de l'opérande

Le registre d'instruction contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction.

1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.

2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.

3. L'opérande est stocké dans un registre.

4.3 Phase 3 : Exécution de l'instruction

1. Le microprogramme réalisant l'instruction est exécuté.

2. Les drapeaux sont positionnés (registre d'état).

3. L'unité de commande positionne le PC pour l'instruction suivante.

5. Jeu d'instructions

La première étape de la conception d'un microprocesseur est la définition de son jeu d'instructions. Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le microprocesseur pourra exécuter. Il va donc en partie déterminer l'architecture du microprocesseur à réaliser et notamment celle du séquenceur. A un même jeu d'instructions peut correspondre un grand nombre d'implémentations différentes du microprocesseur.

Les instructions que l'on retrouve dans chaque microprocesseur peuvent être classées en 4 groupes :

- **Transfert de données :** pour charger ou sauver en mémoire, effectuer des transferts de registre à registre, etc…

- **Opérations arithmétiques :** addition, soustraction, division, multiplication

- **Opérations logiques :** ET, OU, NON, NAND, comparaison, test, etc…

- **Contrôle de séquence :** branchement, test, etc…

6. Mode d'adressage

Un mode d'adressage définit la manière dont le microprocesseur va accéder à l'opérande.

Les différents modes d'adressage dépendent des microprocesseurs mais on retrouve en général :

- l'adressage de registre où l'on traite la donnée contenue dans un registre.
- l'adressage immédiat où l'on définit la valeur de la donnée dans l'instruction.
- l'adressage direct (ou indirect) où l'on traite une données en mémoire.

Selon le mode d'adressage de la donnée, une instruction sera codée par 1 ou plusieurs octets.

7. Les mémoires

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. La capacité de mémorisation détermine la polyvalence des systèmes numériques et leur adaptabilité a de nombreuse situations. De nos jours, il existe des mémoires à lecture seule appelées ROM (Read Only Memery) et d'autres à lecture/écriture improprement appelées RAM (Random Access Memory). Le contenu d'une ROM, fixé lors de la fabrication, reste dans la mémoire même en absence d'alimentation. Au contraire une RAM perd son contenu une fois mise hors secteur et du coup se sont des mémoires volatiles.

Actuellement, Il existe deux types de mémoires qui se distinguent par leur technique de fabrication : les mémoires dynamiques et les mémoires statiques. Il s'agit dans les deux cas de mémoires volatiles qui nécessitent une alimentation pour conserver leur contenu. La mémoire dynamique est appelée DRAM (Dymanic RAM) par opposition à la mémoire statique appelée SRAM (Static RAM). Vue leur moindre coût et densité supérieure d'intégration, les DRAMs sont plus utilisées comme des mémoires principales dans les ordinateurs par contre les SRAMs sont utilisées pour les caches en raison de leurs plus grande vitesse.

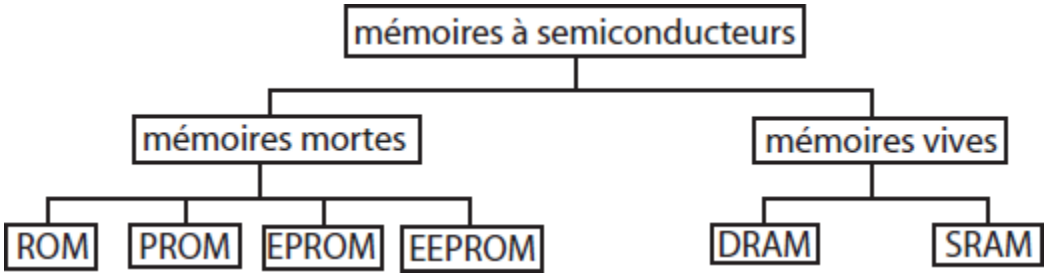
Mémoire dynamique	Mémoire statique
Grande densité d'intégration	Petite densité d'intégration
Bon marché	Chère
Lente	Rapide
Mécanisme de rafraîchissent	

7.2 Caractéristiques d'une mémoire

Les caractéristiques principales et communes à toutes les mémoires sont :

- ✓ **La capacité** : exprimée souvent en octet, c'est le nombre total de bits que contient la mémoire.
- ✓ **Le format des données** : c'est le nombre de bits que l'on peut mémoriser par case mémoire ou tout simplement c'est la largeur du mot mémorisable.
- ✓ **Le temps d'accès** : c'est le temps qui s'écoule entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.
- ✓ **Le temps de cycle** : il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture.
- ✓ **Le débit** : c'est le nombre maximum d'informations lues ou écrites par seconde.
- ✓ **Volatilité** : elle caractérise la permanence des informations dans la mémoire. L'information stockée est volatile si elle risque d'être altérée par un défaut d'alimentation électrique et non volatile dans le cas contraire.

7.3. Classification des mémoires



1. La ROM

Elle est programmée par le fabricant et son contenu ne peut plus être ni modifiée, ni effacée par l'utilisateur.

Avantages :

- Non volatile
- Mémoire rapide

Inconvénients :

- Écriture impossible
- Modification impossible (toute erreur est fatale)
- Délai de fabrication (3 à 6 semaines)
- Obligation de grandes quantités en raison du coût élevé qu'entraîne la production du masque et le processus de fabrication.

- **La PROM**

C'est une ROM qui peut être programmée une seule fois par l'utilisateur (*Programmable ROM*). La programmation est réalisée à partir d'un programmateur spécifique.

Avantages :

- Claquage en quelques minutes
- Coût relativement faible

Inconvénients :

- Modification impossible (toute erreur est fatale)

- **L'EPROM ou l'UV-EPROM**

Pour faciliter la mise au point d'un programme ou tout simplement permettre une erreur de programmation, il est intéressant de pouvoir reprogrammer une PROM. La technique de claquage utilisée dans celles-ci ne le permet évidemment pas. L'EPROM (*Erasable Programmable ROM*) est une PROM qui peut être effacée.

Avantages :

- Reprogrammable et non Volatile

Inconvénients :

- Impossible de sélectionner une seule cellule à effacer
- Impossible d'effacer la mémoire in-situ
- l'écriture est beaucoup plus lente que sur une RAM (environ 1000x).

➢ **L'EEPROM**

L'EEPROM (*Electically EPROM*) est une mémoire programmable et effaçable électriquement. Elle répond ainsi à l'inconvénient principal de l'EPROM et peut être programmée in situ.

Avantages :

- Comportement d'une RAM non Volatile.
- Programmation et effacement mot par mot possible.

Inconvénients :

- Très lente pour une utilisation en RAM.
- Coût de réalisation.

2. Les mémoires vives (RAM)

Une mémoire vive sert au stockage temporaire de données. Elle doit avoir un temps de cycle très court pour ne pas ralentir le microprocesseur. Les mémoires vives sont en général volatiles : elles perdent leurs informations en cas de coupure d'alimentation. Certaines d'entre elles, ayant une faible consommation, peuvent être rendues non volatiles par l'adjonction d'une batterie. Il existe deux grandes familles de mémoires RAM (*Random Acces Memory : mémoire à accès aléatoire*) :

- Les RAM statiques
- Les RAM dynamiques

✓ **Les RAM statiques**

Le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule. Chaque bascule contient entre 4 et 6 transistors.

✓ **Les RAM dynamiques**

Dans les RAM dynamiques (DRAM), l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur. Cette technique permet une plus grande densité d'intégration, car un point mémoire nécessite environ quatre fois moins de transistors que dans une mémoire statique. Sa consommation s'en retrouve donc aussi très réduite. La présence de courants de fuite dans le

condensateur contribue à sa décharge. Ainsi, l'information est perdue si on ne la régénère pas périodiquement (charge du condensateur). Les RAM dynamiques doivent donc être rafraîchies régulièrement pour entretenir la mémorisation : il s'agit de lire l'information et de la recharger.

8. Notion de hiérarchie mémoire

Une mémoire idéale serait une mémoire de grande capacité, capable de stocker un maximum d'informations et possédant un temps d'accès très faible afin de pouvoir travailler rapidement sur ces informations.

Mais il se trouve que les mémoires de grande capacité sont souvent très lente et que les mémoires rapides sont très chères. Et pourtant, la vitesse d'accès à la mémoire conditionne dans une large mesure les performances d'un système.

Afin d'obtenir le meilleur compromis coût-performance, on définit donc une hiérarchie mémoire. On utilise des mémoires de faible capacité mais très rapide pour stocker les informations dont le microprocesseur se sert le plus et on utilise des mémoires de capacité importante mais beaucoup plus lente pour stocker les informations dont le microprocesseur se sert le moins. Ainsi, plus on s'éloigne du microprocesseur et plus la capacité et le temps d'accès des mémoires vont augmenter.

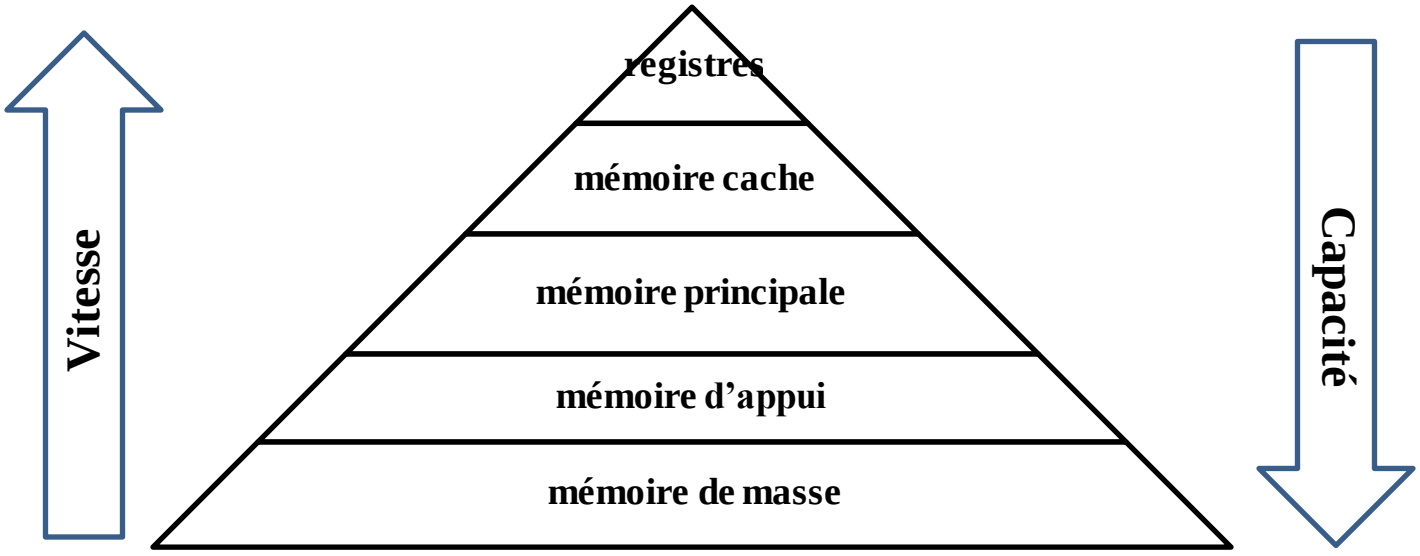


Figure5 : Hiérarchie mémoire.

Les registres : sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.

La mémoire cache : est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.

La mémoire principale : est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.

La mémoire d'appui : sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.

La mémoire de masse : est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnétiques (disque dur) ou optiques (CDROM, DVDROM).

9. Notion d'architecture RISC et CISC

Actuellement l'architecture des microprocesseurs se composent de deux grandes familles :

- L'architecture CISC (Complex Instruction Set Computer).
- L'architecture RISC (Reduced Instruction Set Computer).

9.1. L'architecture CISC

C'est une architecture avec un grand nombre d'instructions où le microprocesseur doit exécuter des tâches complexes par instruction unique. Pour une tâche donnée, une machine

CISC exécute ainsi un petit nombre d'instructions mais chacune nécessite un plus grand nombre de cycles d'horloge. Le code machine de ces instructions varie d'une instruction à l'autre et nécessite donc un décodeur complexe (microcode).

9.2 L'architecture RISC

C'est une architecture dans laquelle les instructions sont en nombre réduit (chargement, branchement, appel sous-programme). Les architectures RISC peuvent donc être réalisées à partir de séquenceur câblé. Leur réalisation libère de la surface permettant d'augmenter le nombre de registres ou d'unités de traitement par exemple. Chacune de ces instructions s'exécutent ainsi en un cycle d'horloge. Bien souvent, ces instructions ne disposent que d'un seul mode d'adressage. Les accès à la mémoire s'effectuent seulement à partir de deux instructions (*Load et Store*). Par contre, les instructions complexes doivent être réalisées à partir de séquences basées sur les instructions élémentaires, ce qui nécessite un compilateur très évolué dans le cas de programmation en langage de haut niveau.

Chapitre 3 : Architecture Interne du Microprocesseur Intel 8086

Objectifs

À l'issue de ce chapitre, l'étudiant doit être capable de :

- expliquer le rôle du jeu d'instructions et des modes d'adressage ;
- écrire des séquences simples (transferts, opérations arithmétiques/logiques, sauts) ;
- structurer un programme assembleur (données, code, procédures) ;
- déboguer des erreurs courantes (drapeaux, registres, débordements).

1. Définitions

L'Intel 8086 (également appelé iAPX 86) est un microprocesseur CISC 16 bits fabriqué par Intel à partir de 1978. C'est le premier processeur de la famille 80x86 (8086, 80186, 80286, 80386, 80486, Pentium, ...)., qui est devenue l'architecture de processeur la plus répandue dans le monde des ordinateurs personnels, stations de travail et serveurs informatiques.

Il est basé sur des registres 16 bits, et dispose d'un bus externe de données de 16 bits (*D0 – D15*) et d'un bus d'adresse de 20 bits (*A0 – A19*), qui lui permet d'adresser 1 Mo. Il contient 29 000 transistors gravés en 3 µm. Sa puissance de calcul varie de 0,33 h (lorsqu'il est cadencé à 4,77 MHz comme dans l'IBM PC) jusqu'à 0,75 MIPS pour la version 10 MHz.

2. Architecture externe du 8086

Il se présente sous la forme d'un boîtier DIP (Dual In-line Package) à 40 broches (figure 1).

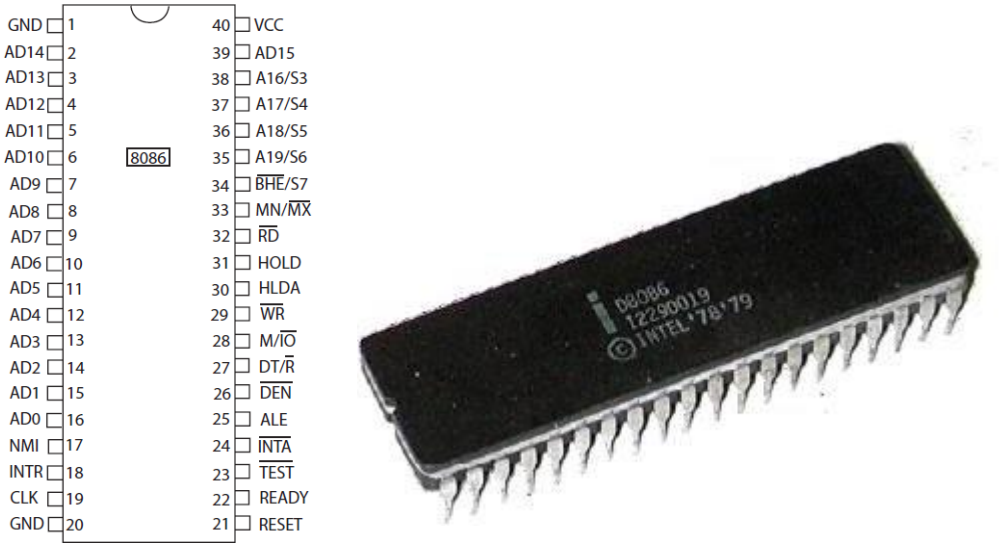


Figure 1 : Le 8086

Caractéristiques du 8086

Le microprocesseur i8086 se caractérise par :

Structure 16 bits.

- Capacité d'adressage de 1 Mo.
- 14 registres internes de 16 bits.
- 7 modes d'adressage.
- Opérations sur des bits, des octets, des mots et des chaînes de caractères.
- Arithmétique signée ou non signée.
- Arithmétique binaire ou BCD, sur 8 ou 16 bits.

3. Architecture interne du 8086

Le 8086 se compose de deux unités de traitement séparées : l'unité d'interface de bus (***UIB***) et l'unité d'exécution (***UE***). Les deux unités fonctionnent simultanément, d'où une accélération du processus d'exécution d'un programme

L'unité d'exécution comporte l'**UAL**, les registres généraux (**AX, BX, CX, DX**), les registres d'adressage (**SP, BP, SI, DI**), le registre d'état (**Flags**) et le décodeur d'instructions.

L'unité d'interface de bus (**UIB**) comporte une file d'attente d'instructions gérée en **FIFO** (First In First Out), les registres de segments (**CS, DS, SS, ES**) et le pointeur d'instruction (**IP**).

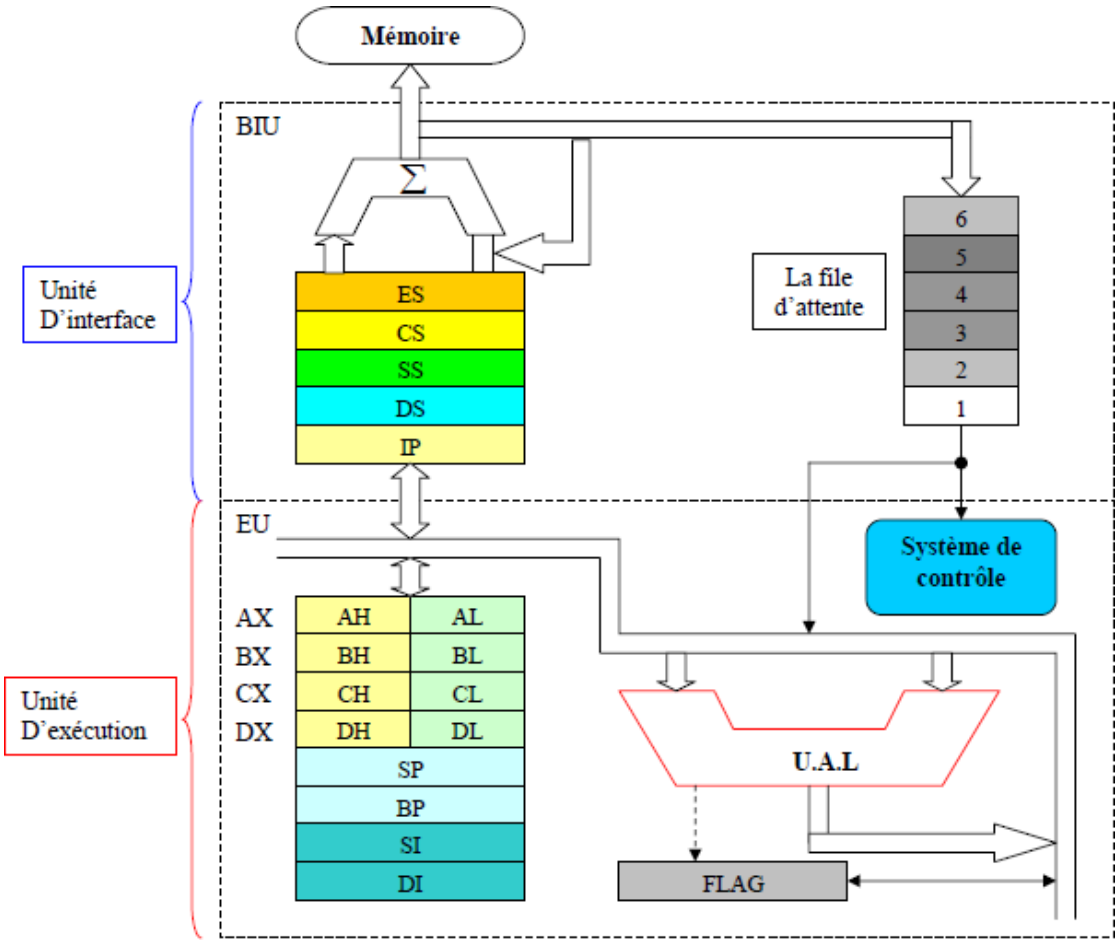


Figure2 : architecteur interne du 8086

3.1 Fonctions de l'UIB (Unité d'Interface de Bus)

- ✓ Elle cherche les instructions à exécuter dans la mémoire et les stocke dans la file d'attente *FIFO*.
- ✓ Elle calcule les adresses physiques sur 20 bits.
- ✓ Elle réalise le transfert des données avec la mémoire.

3.2 Fonctions de l'UE (Unité d'Exécution)

- Elle extrait les codes des instructions à partir de la file d'attente et les exécute.

- Elle fournit les adresses des opérandes à l'UIB en nommant le segment concerné et en fournissant le déplacement dans ce segment.

3.3 Les registres du 8086

Le jeu de registres contient l'ensemble des registres du microprocesseur. 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.

3.3.1. Registres de travails :

Le CPU 8086 comporte 8 registres à usage général, chaque registre est identifié par son nom :

AX	Registre accumulateur 16 bits (composé de deux registres 8 bits, **AH** et **AL**).
BX	Registre d'adresses de base 16 bits (composé de deux registres 8 bits, **BH** et **BL**).
CX	Registre compteur 16 bits (composé de deux registres 8 bits, **CH** et **CL**).
DX	Registre de données 16 bits (composé de deux registres 8 bits, **DH** et **DL**).
SP	Registre pointeur de pile 16 bits.
BP	Registre pointeur de base 16 bits.
SI	Registre d'index source 16 bits.
DI	Registre d'index destination 16 bits.

3.3.2. Registres de segments :

CS	Segment de code contenant le programme en cours.
DS	Segment de données contenant l'adresse du segment de données dans lequel le programme déposera ses éléments de travail (variables, champs, tableaux de branchement, etc.).
ES	Extra segment appartient au programmeur pour définir son utilisation
SS	Segment de pile contenant l'adresse de la pile.

Plus 2 registres spécieux :

- ✓ Le registre IP : pointeur d'instruction.
- ✓ Le registre FLAG : indique l'état en courant de µp.

3.3.3. Le registre d'état(FLAG):

La plupart des instructions arithmétiques et logiques affectent le registre d'état du processeur (ou registre de FLAG).

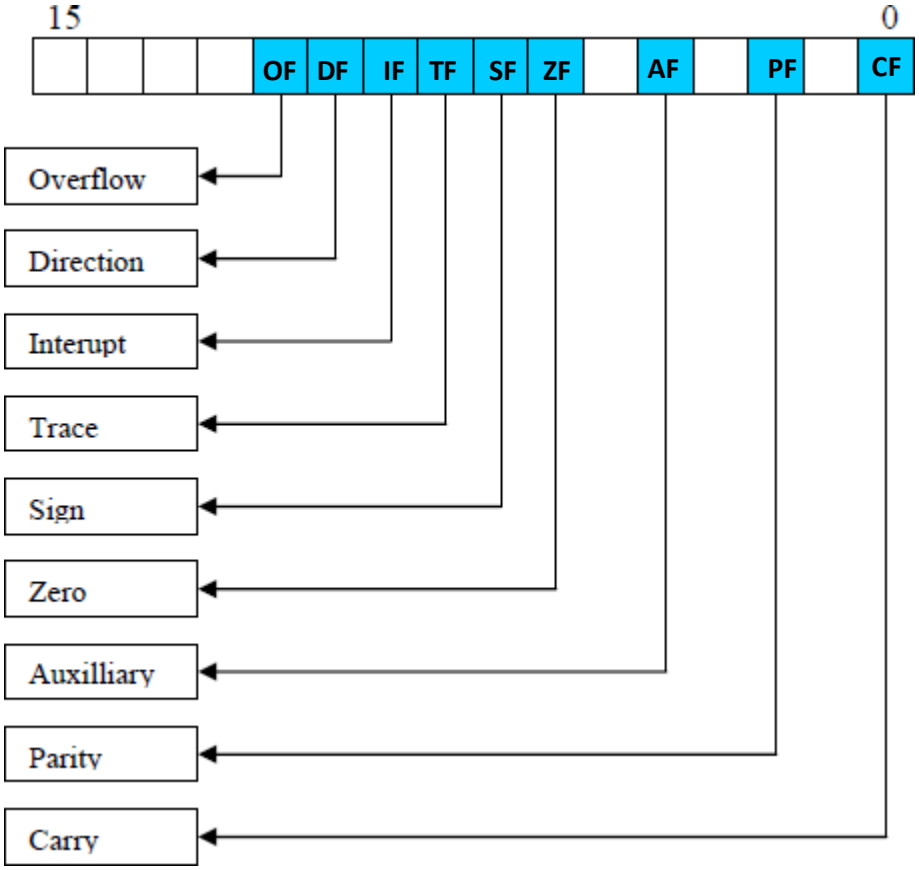


Figure3: Le register d'état (Program Status Word).

CF (Carry Flag) Retenue	cet indicateur est positionné à ***1*** lorsqu'il y a une retenue du résultat à 8 ou 16 bits. Il intervient dans les opérations d'additions (*retenue*) et de soustractions (*emprunt*) sur des entiers naturels. Il est positionné en particulier par les instructions *ADD, SUB et CMP* (*comparaison entre deux valeurs*). ***CF = 1*** s'il y a une retenue après l'addition ou la soustraction du bit de poids fort des opérandes.
PF (Parity Flag) Parité	si les 8 bits du poids faible du résultat de l'opération contiennent un nombre pair de 1, cet indicateur est positionné à ***1***, sinon ***zéro***.
AF (Auxiliary Carry : Demie retenue)	ce bit est égal à ***1*** si on a une retenue du quartet (*4 bits*) de poids faible dans le quartet de poids plus fort.
ZF (Zero Flag : Zéro)	Cet indicateur est positionné à ***1*** quand le résultat d'une opération est égal à zéro. Lorsque l'on vient d'effectuer une soustraction (*ou une comparaison*), ***ZF = 1*** indique que les deux opérandes étaient égaux. Sinon, ***ZF*** est mis à ***0***.
SF (Sign Flag : signe)	***SF*** est positionné à ***1*** si le bit de poids fort du résultat d'une addition ou soustraction est ***1*** ; sinon ***SF = 0***. ***SF*** est utile lorsque l'on manipule des entiers signés, car le bit de poids fort donne alors le signe du résultat.
OF (Overflow Flag : Débordement)	si on a un débordement arithmétique ce bit est positionné à ***1***, c'est à dire le résultat d'une opération excède la capacité de l'opérande (*registre ou case mémoire*), sinon il est à ***0***.
DF (Direction Flag : Auto Incrémentation/Décrémentation)	utilisé par les instructions de traitement des chaînes de caractères pour auto incrémenter ou auto décrémenter le ***SI*** et le ***DI***.
IF (Interrupt Flag : Masque d'interruption)	pour masquer les interruptions venant de l'extérieur ce bit est mis à ***0***, dans le cas contraire (*IF = 1*) le microprocesseur reconnaît l'interruption de l'extérieur.
TF (Trap Flag : Piège)	pour que le microprocesseur exécute le programme pas à pas.

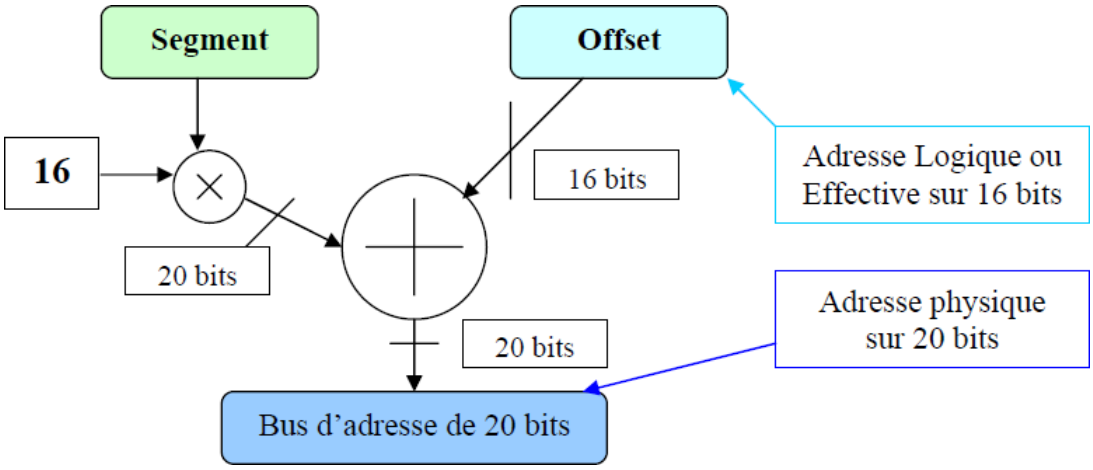
4. Les modes d'adressages :

Introduction :

Avant de commencer, il faut résoudre le problème de la taille de bus d'adresse de 20 bits et la taille des registres internes du µp (16 bit), la question qui il faut poser est « Quelle est la solution proposer par INTEL pour fabriquer une adresse de 20 bits à partir des registres de 16 bits ? ».

La solution est la suivante :

Chaque registre de segment sera multiplié par 16 plus un offset sur 16 bits comme l'indique le schéma au-dessous :



1) Adressage registre :

Ce mode utilise les registres internes de µp, dans ce mode il faut respecté la taille de différents registres.

Exemple :

Dans le plus part du cours, on va utiliser l'instruction MOV (instruction de transfert) :

Syntaxe :

MOV Destination, Source ; Destination → Source.

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.

2) Adressage immédiat :

C'est un mode de manipulation de valeurs latérales avec un registre interne ou une case mémoire.

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.

3) Adressage direct :

Comme l'indique son nom, c'est la réalisation des opérations entre les registres internes de µp et la mémoire.

Exemple :

MOV AL, Bêta ; transférer le contenu de la case mémoire Bêta vers AL.

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

4) 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] ; transfert de contenu de la case mémoire 16 bits pointer par BX ; par rapport au DATA segment vers AX.

MOV [SI], DL ; transfert de contenu de DL vers la case mémoire 8 bits pointer ; par SI par rapport au DATA segment.

5) Adressage basé :

Tous simplement, il utilise la 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.

6) 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].

Remarque : les cases mémoires consultées ou modifiées sont par rapport au DATA segment

7) Adressage basé indexé :

C'est un mode combine entre de modes d'adressage comme l'indique son nom.

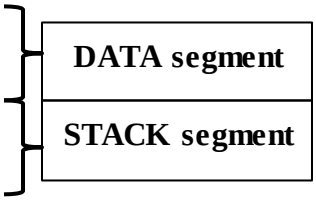
Syntaxe :

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

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

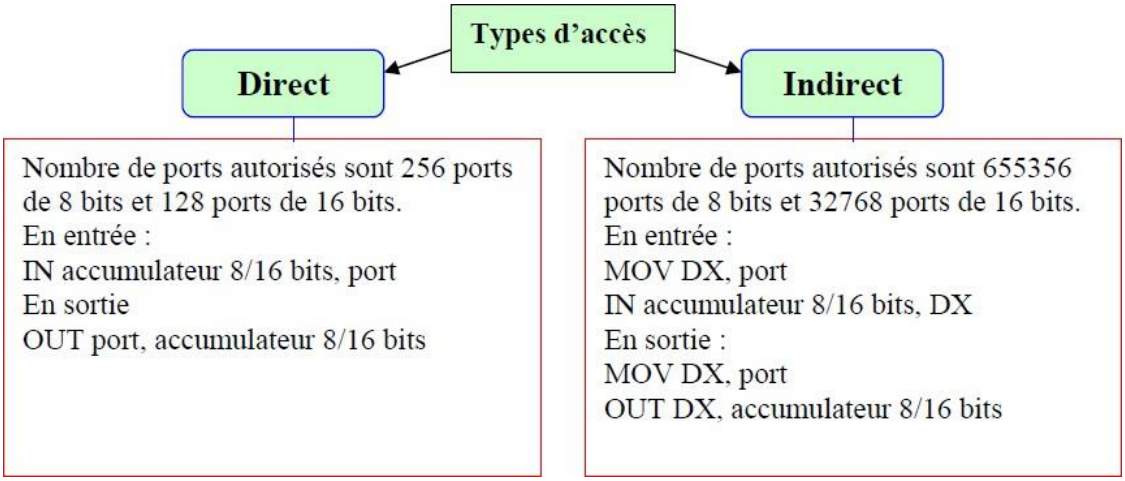
MOV [BP+ SI/DI + déplacement 8 bits ou 16 bits], source.

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



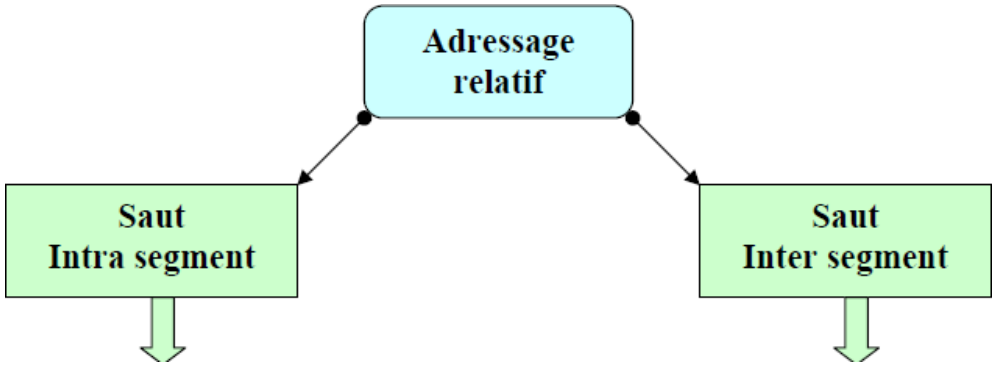
8) Adressage des entrées sorties (I/O) :

Dans ce mode, le bus d'adresse devient de taille 16 bits □ l'espace adressable par I/O est 64 KO. Il utilise comme instructions d'accès (IN, OUT).



9) adressage relatif :

Ce type d'adressage est utilisé par les instructions de saut.



Ce type de saut change seulement le pointeur d'instructions (IP), et le CS ne change pas.

1) saut conditionnel :

le saut n'est réalisé que si une condition est vérifiée. L'adresse est calculée en ajoutant au contenue de (IP), avec cette valeur est un déplacement signé sur 8bits.

Exemple :

DEC CX
JNZ ETIQUETTE
CALL SEND_BYTE

2) saut inconditionnel :

le saut est réalisé quelque soit l'état de µp, 2 types d'adressage sont utilisés :

a) ADR direct relatif : l'adresse est calculée par l'ajout d'un déplacement signé à (IP).

Long : (+32767 à –32768 bytes)

Exemple : JMP LONG ETIQ1

Court : (+127 à –128)

Exemple : JMP SHORT ETIQ2

b) ADR indirect : ici, la nouvelle valeur du contenu de (IP) est le contenu d'une case mémoire ou d'un registre.

Exemple :

JMP AX ; IP=AX (par registre)
JMP WORD PTR [BX] ; IP=case de 16 bits pointé par BX par rapport à DS (par mémoire).

Dans ce mode de saut le changement se fait sur 32 bits [CS:IP].

1) **ADR direct :** la valeur de saut est précisée le programme après le code de l'instruction de saut

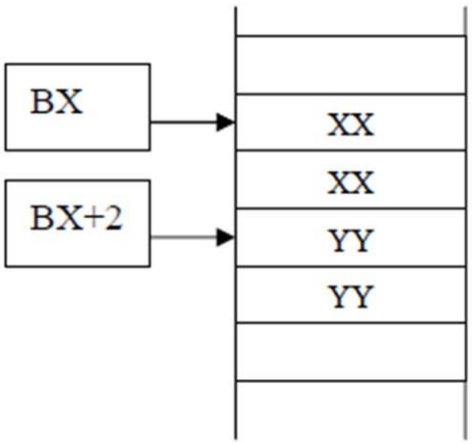
Exemple :

JMP 6060h:4000h ; CS□6060h IP□4000h.

2) **ADR indirect :** la valeur de saut (32bits) se trouve dans la mémoire, dans ce mode on utilise le mode d'adressage indirect.

Exemple :

JMP DWORD PTR [BX]



IP ←BX].
CS ← [BX+2].
IP ←XXXX.
CS ←YYYY.

10) Adressage vectorisé :

Ce mode d'adressage est utilisé seulement pour le traitement des d'interruption. Le mot **VECTEUR** est l'adresse de l'ISR. (INTERUPT SERVICE ROUTINE) dans le premier Kilo bytes du bloc mémoire (00000h : 003FFh). Cet escape mémoire est segmenté en 256 vecteurs □ 4 bytes par vecteur.

Pour quoi 4 bytes ?

Par ce que le vecteur composé par la valeur de (CS) et la valeur de (IP) de l'ISR.

Les instructions de transfert

MOV Destination, Source

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

```
MOV AX, 77H ; adressage immédiat
MOV AX, [1064H] ; adressage direct
MOV BX, OFFSET TAB1 ; charger dans BX l'adresse de TAB1
MOV AX, SEG_DATA ; Il ne faut jamais charger un registre segment par
MOV DS, AX ; une valeur; CS n'est jamais un registre cible.
```

Xchange XCHG

Cette instruction permet d'échanger le contenu de deux registres ou un registre et une case mémoire. Exemple :

```
XCHG AX, CX ; pas pour les registres segments
XCHG BX, CASE_MEM
```

PUSH & POP

PUSH : Sauvegarde une donnée dans la pile,

POP : Restauration d'un mot depuis la pile,

Le mot peut être un Registre 16 bits, Registre SEGMENT ou Case Mémoire. Si le mot est de 8 bits il sera étendu à 16 bits signés.

Exp :

```
PUSH AX ; Sauvegarde AX
POP AX ; Restaure AX
PUSHF ; Sauvegarde les Flags
POPF ; Restaure F
PUSH Var1 ; Sauvegarde une variable
```

POP Var1 ; Restaure Var1

Instructions de transfert d'adresse

LEA : Cette instruction charge l'adresse d'un opérande (offset) dans un registre.

Syntaxe : **LEA** Reg16, Mem16

LEA BX, Table ; charge dans BX l'offset de Table

LEA BX, [Table + SI] ; charge dans BX L'adresse effective Table + SI

LDS : chargement simultané d'un registre et du segment DS

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

Instructions d'Entrées/Sorties

IN & OUT : Transfert des données pour les périphériques,

IN Accumulateur, Port ; port compris entre 0 et 255

OUT Port, Accumulateur

Il est plus commode de charger l'adresse du port dans DX

IN AL, DX ; charger un octet dans le registre AL

OUT DX, AX ; le contenu de AX est sorti par le port d'adresse DX

4.2. Les instructions Arithmétiques

Instructions d'addition

ADD destination , Source : Addition sans retenu de destination+source

ADC destination , source : Addition avec retenu de destination+source

Exp : ADD AX,BX ; AX □AX+BX

AAA Ajustement ASCII pour Addition en ASCII

DAA Ajustement Décimal pour Addition en BCD

Exp : ADD AL, 37H

AAA

Instructions de soustraction

SBB Soustraction avec retenue

SUB destination , source : destination □destination - source

Exp : SBB AX,BX ; AX □AX-BX –CF

SUB AX,BX ; AX □AX-BX

AAS Ajustement ASCII pour division

Instructions de multiplication

MUL : multiplication non signe du contenu de l'accumulateur par le continu d'un registre ou d'une case mémoire.

Exp : MUL BX ; DX&AX □AX * BX

IMUL : multiplication signée

AAM : correction ASCII de multiplication

Instructions de division

DIV : Réalise la division non signée d'un nombre de 16 bits, contenu dans

AX (ou 32bits contenu dans DX et AX) par le nombre de 8 ou 16 bits. Le quotient est stocké dans AL (ou AX) et le reste dans AH (ou DX).

DIV BX ; AL□AX/BX et AH□reste (AX/BX)

IDIV : division signe

AAD : Ajustement ASCII pour division

NEG Destination

Cette instruction va tout simplement inverser le contenu d'un registre ou d'une adresse mémoire. Pour cela, elle inverse d'abord tous les bits, et ajoute 1.

Flags modifiés : AF - OF - PF - SF – ZF

Exemple : NEG AX

INC Destination :

Cette instruction incrémente de 1 un registre ou un emplacement mémoire.

Exemple : INC AX

INC TAB[DI]

DEC Destination

Décrémente d'une unité un registre ou une case mémoire.

Exemple : DEC AL

DEC TABLE

CMP Opérande1, Opérande2

Soustrait les deux opérandes. Le résultat de la soustraction n'est pas sauvé, seuls les indicateurs sont modifiés.

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

Exemple : CMP AX, 7

4.3. Les instructions logiques

AND Opérande1, Opérande2

Cette instruction est l'équivalant d'une porte ET logique, elle réalise un ET entre 2 opérandes. Le résultat est placé dans le premier opérande.

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

Exemple : AND AH, BH

OR Opérande1, Opérande2

Encore une autre porte logique, cette porte effectue un OU entre deux opérandes.

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

Exemple : OR AH, BH

XOR Opérande1, Opérande2

Cette instruction effectue un OU exclusif.

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

Exemple : XOR AH, BH

NOT Destination

C'est l'équivalent de la porte logique NON, le principe de cette porte est simple, inverser tous les bits.

Flags modifiés : Aucun

Exemple : NOT AX

TEST Opérande1, Opérande2

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

Exemple : TEST AL, 01010110 B

4.4. Les instructions de décalage et de rotation

En mode immédiat, seuls les décalages et rotations de 1 bit peuvent se réaliser. Pour réaliser des décalages ou rotations de plusieurs bits, il faut explicitement utiliser le registre CL.

SHR (Shift logical Right)

Décale à droite de x positions, x 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.

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

Exemple: SHR AH, CL

SHL / SAL (Shift logical Left / Shift Arithmetic Left)

Effectue un décalage des bits du premier opérande, les bits sortant sont insérés dans CF, les bits entrant sont mis à 0.

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

Exemple : SHL AH, 1

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 bit 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.

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

Exemple : SAR AH, CL

ROR (ROtate Right)

A peu près le même style que les instructions précédentes, les bits sont décalés vers la droite, mais, cette fois, le bit sortant est injecté à dans CF et dans le bit de poids fort.

Flags modifiés : CF - OF

Exemple : ROR AH, 1

ROL (ROtate Left)

Identique à ROR, mais décale sur la gauche.

Flags modifiés : CF - OF

Exemple : ROL AL, CL

RCR (Rotate trought Cary Right)

Cette instruction agit sur les bits de l'opérande, elle les décale de x vers la droite. Le bit contenu dans CF prend la position du bit le plus fort, puis CF prend la valeur du bit sortant. X est la valeur contenue dans le deuxième opérande.

Flags modifiés : CF - OF

Exemple : RCR AH, CL

RCL (Rotate trought Cary Left)

Identique à RCR, mais déplace les bits vers la gauche.

Flags modifiés : CF - OF

Exemple : RCL AH, 1

4.5. Instructions de branchement

4.5.1. Les branchements inconditionnels

JMP : (Jump)

Cette instruction effectue un saut sans condition. Le saut peut être sur un label, une adresse mémoire, ou un registre.

Flags modifiés : Aucun

Exemple : JMP AX ; IP □ AX

JMP Boucle

INT (Interrupt)

INT active la procédure d'interruption spécifiée par l'opérande de type de l'interruption. L'adresse de pointeur d'interruption est calculée en multipliant le type d'interruption par 4. On verra plus loin l'interruption INT 21H du DOS. Cette instruction sauvegarde dans la pile les contenus des registres (Flags, CS et IP) et remet à zéro les flags TF et IF. La routine d'interruption se termine toujours par l'instruction IRET (Interrupt Return)

CALL

Cette instruction permet l'exécution d'une procédure (sous-programme) située en dehors du programme principal.

- un CALL intra-segment sauvegarde dans la pile le contenu de IP.
- un CALL inter-segment sauvegarde dans la pile le contenu de CS et IP.

Une procédure se termine toujours par l'instruction RET (Return)

4.5.2. Les branchements conditionnels

Ces instructions permettent de sauter "une partie d'un programme" si une condition est vérifiée. Les tableaux suivants donnent la liste des instructions de saut sur test des nombres signés et non signés

Test des nombres non signés

Instruction	Condition	Indicateurs	Définition
JA	A > B	CF = 0 et ZF = 0	Jump if Above
JNBE			Jump if Not Below or Equal
JAE	A >= B	CF = 0	Jump if Above or Equal
JNB			Jump if Not Below
JNC			Jump if Not Carry
JBE	A <= B	(CF = 1 et ZF = 1) ou (CF <> ZF)	Jump if Below or Equal
JNA			Jump if Not Above
JB	A < B	CF = 1	Jump if Below
JC			Jump if Carry
JNAE			Jump if Not Above or Equal
JE	A = B	ZF = 1	Jump if Equal
JZ			Jump if Zero
JNE	A <> B	ZF = 0	Jump if Not Equal
JNZ			Jump if Not Zero

Test des nombres signés

Instruction	Condition	Indicateurs	Définition
JG	A > B	ZF = 0 et OF = SF	Jump if Greater
JNLE			Jump if Not Lower or Equal
JGE	A >= B	SF = OF	Jump if Greater or Equal
JNL			Jump if Not Lower
JNG	A <= B	(ZF = 1) ou (SF <> OF)	Jump if Not Greater
JLE			Jump if Lower or Equal
JNGE	A < B	SF <> OF	Jump if Not Greater or Equal
JL			Jump if Lower
JE	A = B	ZF = 1	Jump if Equal
JZ			Jump if Zero
JNE	A <> B	ZF = 0	Jump if Not Equal
JNZ			Jump if Not Zero

4.5.3. Les instructions de boucle

LOOP

Cette instruction effectue une répétition tant que CX n'est pas égal à zéro; celle-ci décrémente CX de 1 à chaque saut.

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).

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).

Chapitre 4 Architecture et fonctionnement d'un microcontrôleur

Objectifs

À l'issue de ce chapitre, l'étudiant doit être capable de :

- comparer µP et µC (niveau d'intégration, coût, consommation) ;
- identifier les blocs d'un µC (GPIO, timers, PWM, ADC, UART/SPI/I²C) ;
- configurer les registres essentiels (TRIS/PORT, timers, ADC, interruptions) ;
- proposer une architecture logicielle simple (init + boucle + ISR).

1. Introduction : du microprocesseur au microcontrôleur

Le microprocesseur est principalement une unité de calcul (CPU) qui s'appuie sur des mémoires et des interfaces externes. Cette organisation, très flexible, est adaptée aux ordinateurs et aux systèmes complexes. Pour les applications embarquées (commande, automatisme, acquisition), elle devient toutefois coûteuse (nombre de composants, bus, consommation) et parfois surdimensionnée.

Un microcontrôleur (µC) intègre sur une seule puce le CPU, des mémoires (programme et données) et des périphériques :
entrées/sorties (GPIO), timers, ADC, interfaces série, etc. Le µC constitue ainsi un « système sur puce » (SoC) dédié au contrôle-commande : lecture de capteurs, calcul, commande d'actionneurs, communication.

2. Différences µP / µC et architectures internes

Le microcontrôleur se distingue du microprocesseur par le niveau d'intégration : là où le µP nécessite des composants externes (mémoire, contrôleurs d'E/S), le µC est autonome. Cette autonomie facilite l'embarqué : carte plus compacte, coût réduit, consommation plus faible et démarrage rapide.

En interne, de nombreux microcontrôleurs adoptent une architecture Harvard, où la mémoire programme et les mémoires données sont séparées. Ceci autorise l'accès simultané aux instructions et aux données, et facilite une exécution en pipeline (préchargement de l'instruction suivante).
L'architecture Von Neumann, au contraire, utilise un seul espace mémoire et un bus unique, ce qui peut introduire un « goulot d'étranglement ».

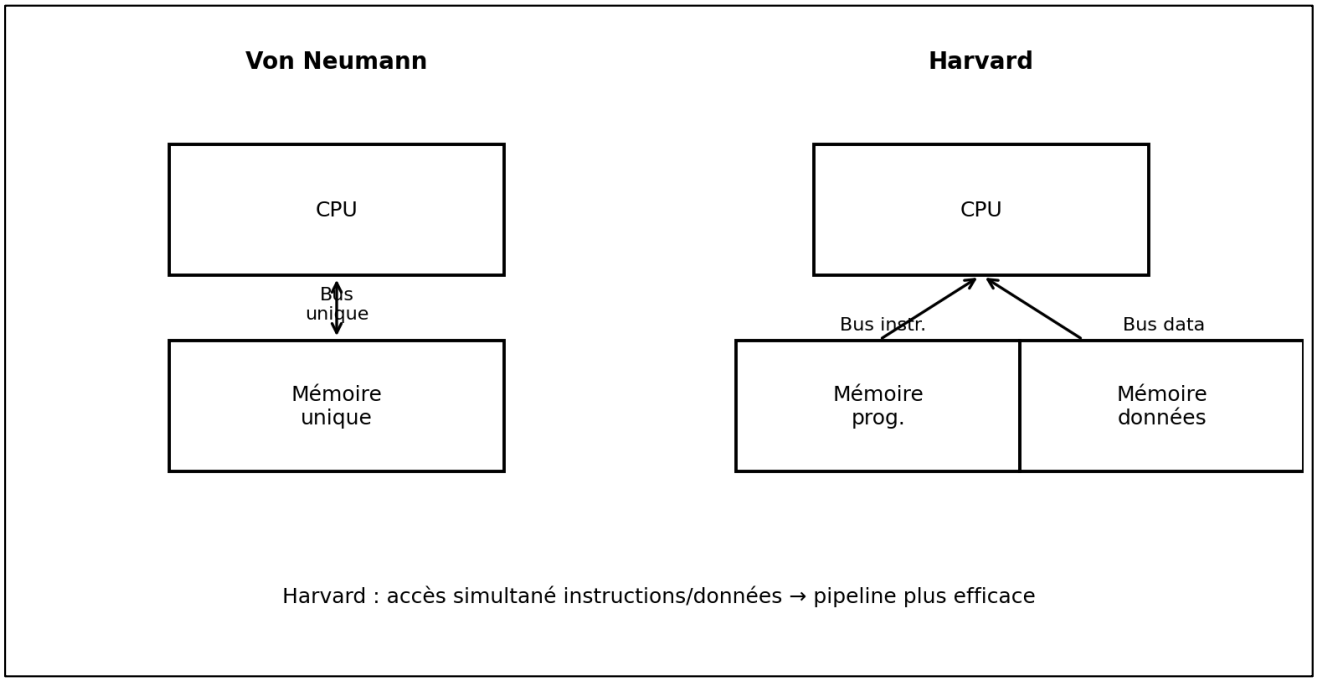


Figure 1 : Architectures Von Neumann vs Harvard (comparaison).

3. Blocs fonctionnels d'un microcontrôleur

La Figure 2 présente les blocs principaux d'un microcontrôleur. On y retrouve : (i) le cœur CPU (UC + UAL + registres), (ii) les mémoires (Flash, RAM, EEPROM), (iii) les ports d'E/S et (iv) les périphériques internes (timers, PWM, ADC, communication).

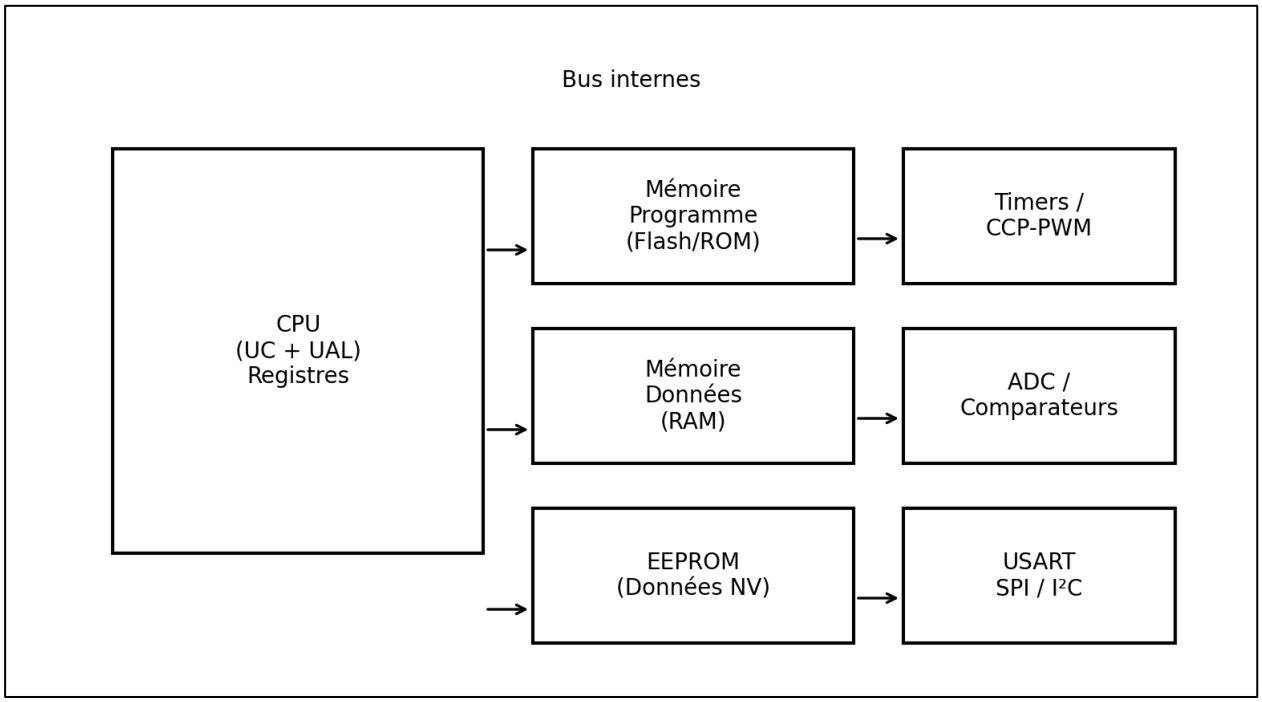


Figure 2 : Blocs principaux d'un microcontrôleur.

3.1. CPU, registres et pipeline

Le CPU exécute un cycle « fetch – decode – execute ». Le compteur programme (PC) pointe sur l'instruction à exécuter. Le décodage sélectionne l'opération (arithmétique, logique, transfert, saut). Dans les architectures RISC des PIC16, les instructions sont généralement simples, ce qui permet un pipeline efficace : la plupart des instructions s'exécutent en un cycle.

Les registres (accumulateur W, STATUS, FSR…) contiennent l'état du CPU et des indicateurs (zéro, retenue, etc.). La programmation bas niveau consiste souvent à manipuler directement ces registres via des instructions de transfert.

3.2. Mémoires : Flash, RAM, EEPROM

* Flash/ROM : stocke le programme (firmware). La Flash est reprogrammable et permet le développement itératif.
* RAM : stocke les variables, la pile logicielle (selon famille) et une partie des registres
* EEPROM : mémorise des paramètres non volatils (calibration, consignes) avec un nombre limité de cycles d'écriture.

Dans un projet de commande électrotechnique, il est courant de stocker en EEPROM des constantes de calibration (capteurs), des paramètres de régulation (gains PI) ou des seuils de protection.

3.3. Ports E/S (GPIO) et fonctions alternatives

Les ports d'E/S (PORTx) sont configurés via les registres TRISx (direction entrée/sortie). Les broches peuvent également être multiplexées avec des fonctions périphériques : entrée ADC, UART, SPI, PWM, etc. Le concepteur doit donc planifier l'affectation des broches en fonction des besoins.

Bonnes pratiques : (i) initialiser TRIS avant l'activation des périphériques, (ii) éviter les conflits de multiplexage, (iii) utiliser des résistances de tirage adaptées et (iv) protéger les entrées (limitation, filtrage, ESD) si nécessaire.

3.4. Timers, capture/compare et PWM

Les timers permettent la génération d'interruptions périodiques (base de temps), la mesure d'événements et la génération PWM. En commande de convertisseurs, la PWM fixe la tension moyenne appliquée (buck/boost) ou la modulation d'un onduleur. Dans la commande de moteurs DC, la PWM contrôle la vitesse par variation de la tension moyenne.

3.5. ADC : acquisition analogique

L'ADC mesure des tensions analogiques. Pour mesurer courant/tension de puissance, il faut généralement un conditionnement : shunt + amplificateur, diviseur résistif, filtrage anti-repliement. La précision globale dépend de la résolution ADC, de la référence de tension et du bruit.

Une démarche classique : choix du capteur → conditionnement → calibration → acquisition → filtrage numérique → exploitation (RMS, moyenne, seuils).

3.6. Communication : UART, SPI, I²C

* UART/USART : communication série asynchrone, simple, utilisée vers PC, modules GSM/BT.
* SPI : bus synchrone rapide (maître/esclave), utilisé pour mémoires, convertisseurs, afficheurs.
* I²C : bus synchrone multi-maître avec adressage, pratique pour capteurs et circuits périphériques. Le choix dépend du débit, du câblage, du nombre de nœuds et de la robustesse vis-à-vis des perturbations.

4. Exemple : microcontrôleur PIC (famille PIC16)

Les microcontrôleurs PIC (Microchip) sont très répandus. La famille PIC16 est de type RISC 8 bits, souvent utilisée en enseignement : architecture claire, registres SFR accessibles, nombreux périphériques.

Nous prendrons le PIC16F877A comme exemple pédagogique.

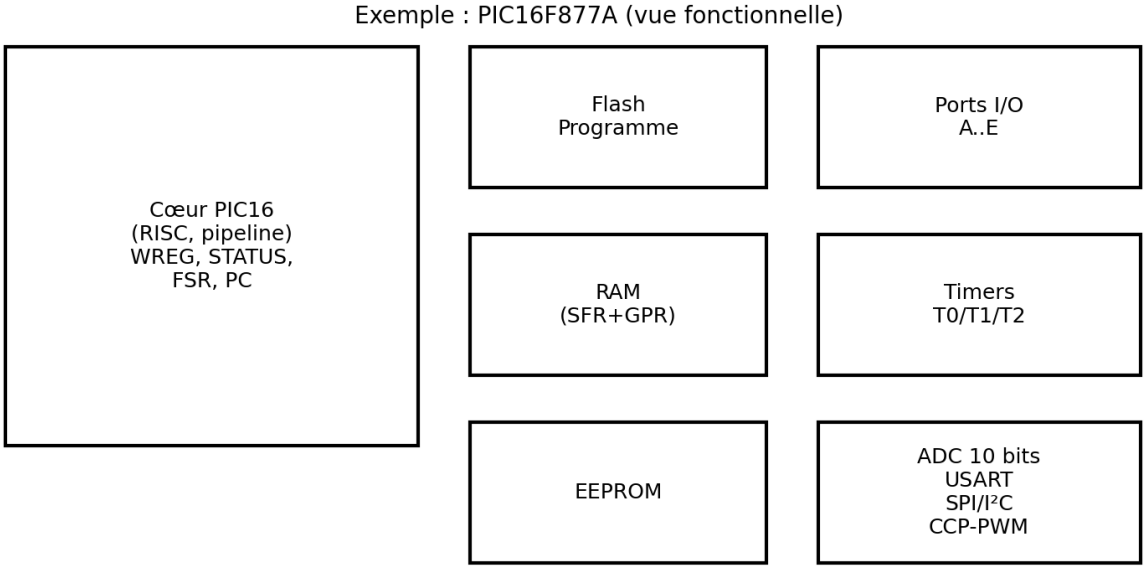


Figure 3 : Vue fonctionnelle simplifiée d'un PIC16 (ex. PIC16F877A).

4.1. Ressources matérielles (synthèse)

Ressource	Rôle	Exemple (PIC16F877A)
Flash programme	Stockage firmware	kilo-mots (ordre de grandeur)
RAM	Variables, registres	SFR + GPR (banques)
EEPROM	Paramètres NV	octets à centaines d'octets
Ports I/O	Interface capteurs/actionneurs	Ports A…E
Timers	Base de temps / capture	TMR0, TMR1, TMR2
CCP/PWM	Modulation/commande	PWM + capture/compare
ADC	Mesure analogique	10 bits, multi-voies
USART	Communication	Tx/Rx asynchrone
SPI/I²C (MSSP)	Bus synchrone	SPI + I²C

Tableau 1 : Ressources PIC16 (exemple de synthèse).

4.2. Organisation mémoire et registres SFR

La mémoire de données est organisée en banques. Les registres SFR (Special Function Registers) pilotent le matériel : TRISx/PORTx (ports), TMRx (timers), ADCON/ADRES (ADC), TXSTA/RCSTA/SPBRG (UART), etc. La RAM GPR contient les variables utilisateur.

Une méthode de programmation robuste consiste à : (i) définir des macros/constantes pour les bits importants, (ii) commenter chaque configuration, et (iii) centraliser l'initialisation dans des fonctions dédiées.

4.3. Interruptions : mécanisme et bonnes pratiques

Les interruptions traitent des événements asynchrones (overflow timer, réception UART, fin conversion ADC). Le CPU interrompt temporairement le programme principal, sauvegarde un contexte minimal, exécute l'ISR puis retourne à l'exécution normale.

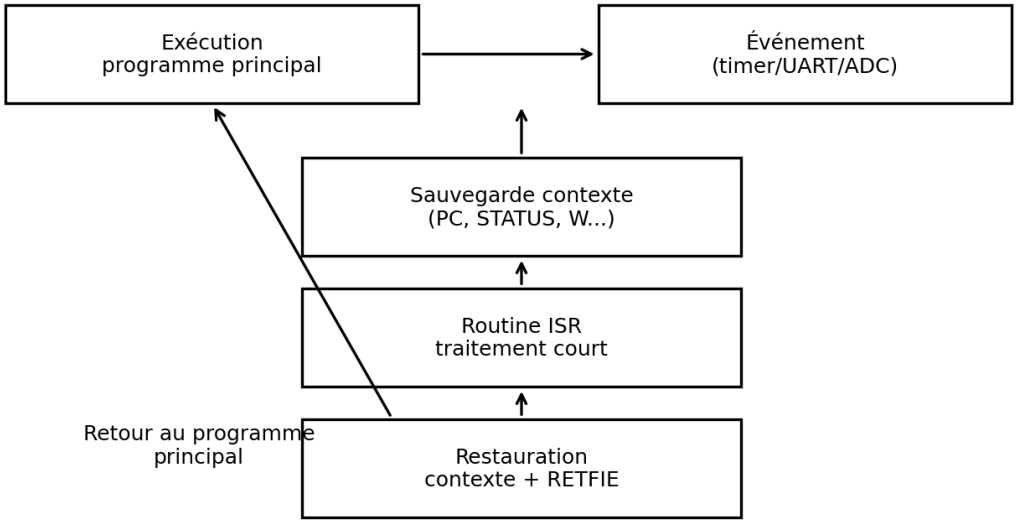


Figure 4 : Logigramme simplifié de prise en charge d'une interruption (ISR).

- Limiter le temps d'exécution de l'ISR (éviter boucles longues).
- Utiliser des drapeaux (flags) pour signaler au programme principal qu'un traitement est à faire.
- Protéger les accès concurrents (variables partagées) : sections critiques ou variables volatiles.
- Prioriser les interruptions critiques (si la famille le permet).

4.4. Horloge, watchdog et configuration

Le fonctionnement dépend de l'horloge (oscillateur interne/externe). La fréquence fixe le temps de cycle machine et donc la précision des timers/PWM. Le watchdog (WDT) est un temporisateur de sécurité : si le logiciel se bloque, le WDT provoque un reset pour restaurer un état sûr. Les bits de configuration (fuses) définissent ces choix (type d'oscillateur, WDT, protection code, etc.).

5. Programmation d'un microcontrôleur PIC (méthodologie)

- Le flux de développement est généralement :

  1) écrire le programme (C ou assembleur),

  2) compiler/assembler et lier pour obtenir un fichier .hex,

  3) programmer la Flash via ICSP,

  4) tester et déboguer (simulateur, ICD).

En C embarqué, une architecture simple et efficace consiste à séparer :

* l'initialisation (horloge, ports, périphériques),
* la boucle principale (supervision),
* et les ISR (événements temps réel).

5.1. Exemple de configuration ports + timer (pseudo-code)

```
// Exemple (pseudo-code) : initialisation GPIO et Timer
init() {
 TRISB = 0b00000001;  // RB0 en entrée (bouton), le reste en sortie
 PORTB = 0;           // état initial
 OPTION_REG = ...;    // préscaler et configuration Timer0
 TMR0 = 0;
 INTCONbits.T0IE = 1; // autoriser interruption Timer0
 INTCONbits.GIE  = 1; // interruption globale
}
ISR() {
 if (INTCONbits.T0IF) { INTCONbits.T0IF = 0; /* traitement court */ }
}
```

Chapitre 5 Applications des microprocesseurs et microcontrôleurs (interfaces, commande, acquisition)

Objectifs

À l'issue de ce chapitre, l'étudiant doit être capable de :

- réaliser des interfaces utilisateur (LCD, clavier) ;
- générer des signaux stables via timers/PWM ;
- proposer une architecture de commande pour convertisseurs et moteurs (mesure + PWM + protections) ;
- mettre en place une acquisition de données (échantillonnage, filtrage, transmission).

1. Introduction

Ce chapitre rassemble des applications typiques réalisées avec un microprocesseur ou un microcontrôleur : affichage, interface clavier, génération de signaux (ports, timers, PWM), commande de convertisseurs et de moteurs, mesure de fréquence et acquisition de données. L'objectif est de passer d'un cahier des charges (signaux, timing, précision) à une solution matérielle et logicielle validée.

La démarche recommandée :

1. Analyser le besoin et les contraintes (électriques, temps réel, coût),
2. Concevoir l'interface matérielle (adaptation, isolation, filtrage),
3. Développer les drivers logiciels (LCD, clavier, PWM, acquisition),
4. Valider expérimentalement (oscilloscope, analyseur logique).

2. Interface LCD (HD44780) : mode 4 bits

Les afficheurs LCD alphanumériques 16×2 ou 20×4 (contrôleur HD44780) sont utilisés pour afficher des mesures (tension, courant), des états (alarme, mode) ou des menus. En mode 4 bits, on économise des broches : chaque octet est transmis en deux demi-octets.

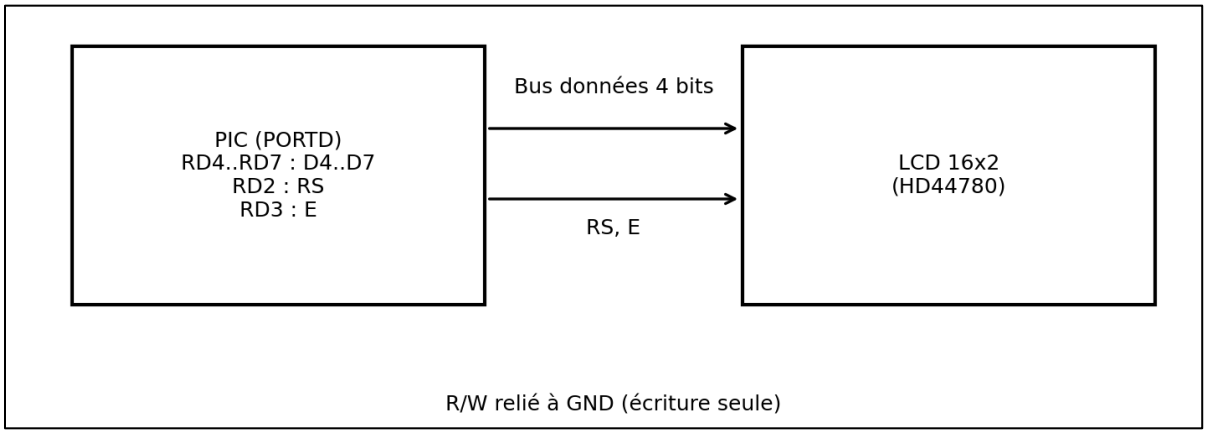


Figure 1 : Exemple de câblage LCD en mode 4 bits.

2.1. Séquence d'initialisation et timing

Le LCD impose une séquence d'initialisation au démarrage. Les temporisations (ms) doivent être respectées, surtout après la mise sous tension. En pratique, un driver logiciel envoie : Function Set, Display Control, Clear Display, Entry Mode Set, puis écrit les caractères.

2.2. Commandes usuelles

Commande	Hex	Rôle
Clear display	0x01	Efface l'écran et repositionne le curseur
Return home	0x02	Curseur au début (sans effacer)
Entry mode set	0x04–0x07	Direction du curseur, shift
Display ON/OFF	0x08–0x0F	Affichage, curseur, clignotement
Function set	0x20–0x3F	Mode 4/8 bits, lignes, police
Set DDRAM addr	0x80+addr	Position du curseur

Tableau 1 : Commandes LCD (HD44780).

2.3. Exemple de driver (pseudo-code)

```
lcd_write_nibble(n) { D4..D7 = n; pulse(E); }
lcd_write_byte(rs, b) { RS=rs; lcd_write_nibble(b>>4); lcd_write_nibble(b&0x0F); delay_us(50); }
lcd_cmd(c) { lcd_write_byte(0, c); }
lcd_data(d) { lcd_write_byte(1, d); }
```

3. Interface clavier matriciel

Un clavier matriciel (4×3, 4×4) relie les touches entre des lignes et des colonnes. On configure les lignes en sorties et les colonnes en entrées (avec pull-up). Le balayage consiste à activer une ligne à la fois et lire l'état des colonnes.

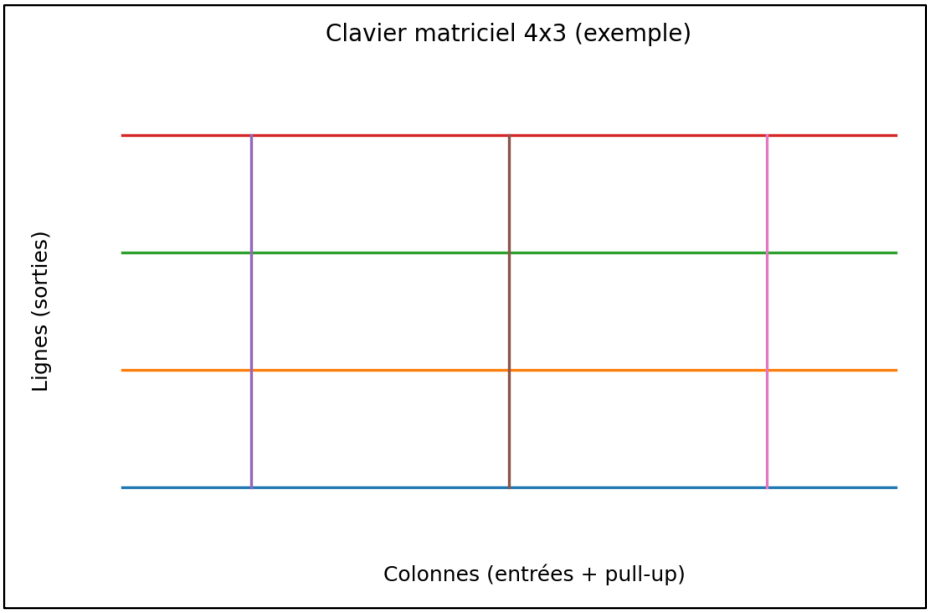


Figure 2 : Principe d'un clavier matriciel 4×3.

3.1. Anti-rebond et décodage

Les contacts mécaniques provoquent des rebonds (oscillations rapides). La solution : relire après 10–20 ms, ou utiliser un filtre logiciel (compteur). Le décodage se fait via une table de correspondance (ligne, colonne) → touche.

3.2. Algorithme de scan (pseudo-code)

```
for each row i:
drive row i = 0 (others = 1)
read columns
if any column == 0:
  wait 15 ms and re-read (debounce)
  decode key
  wait key release (optionnel)
```

4. Génération de signaux : timers et PWM

La génération de signaux par « bit-banging » est possible mais imprécise et dépendante des interruptions. Les timers et PWM matériels offrent une fréquence stable, un faible jitter et libèrent le

CPU.

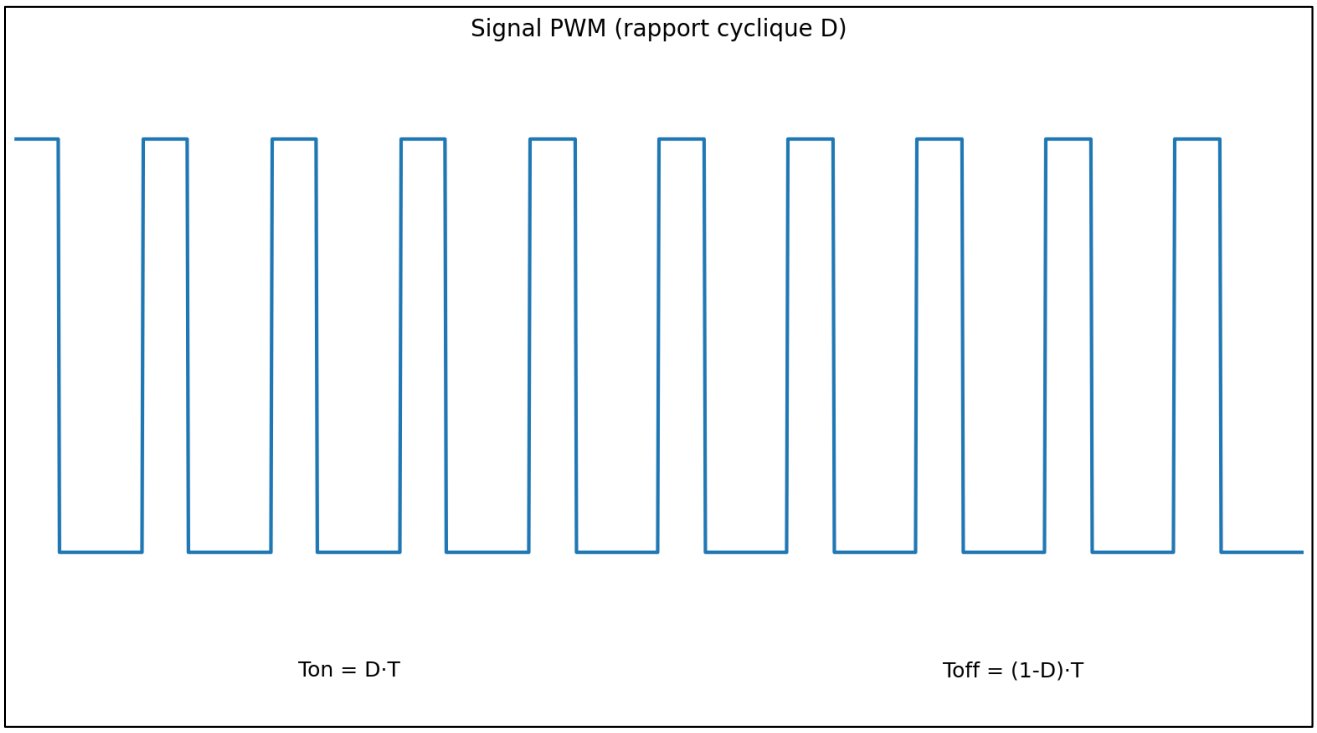


Figure 3 : Forme d'onde PWM et rapport cyclique D.

4.1. Choix de la fréquence PWM

Dans les convertisseurs DC/DC, on choisit typiquement une fréquence entre quelques kHz et plusieurs centaines de kHz selon les composants et la topologie. En commande moteur DC, des fréquences > 16 kHz évitent le bruit audible. En onduleurs, la fréquence de découpage dépend du compromis pertes/filtrage.

4.2. PWM et temps mort (dead-time)

En commande de demi-ponts (MOSFET/IGBT), il faut insérer un temps mort entre l'extinction d'un interrupteur et l'allumage de l'autre afin d'éviter le court-circuit du bus DC. Ce temps mort peut être géré par le périphérique PWM (selon famille) ou par une logique externe.

5. Interfaces pour convertisseurs (DC/DC, onduleurs)

Un microcontrôleur pilote généralement un driver de grille (high-side/low-side) qui adapte les niveaux logiques et fournit le courant de commande.

Les exigences principales :

* adaptation 3.3/5 V → driver,

* isolation galvanique si nécessaire,
* protections : surintensité (OC), surtension (OV), surtempérature (OT),
* mesures : tension/courant via ADC pour boucle de régulation.

Bloc	Rôle	Exemple
PWM	Commande interrupteurs	Buck/Boost, onduleur
ADC	Mesure U/I	Shunt + ampli, diviseur
Comparateur/OC	Protection rapide	Arrêt PWM
Driver	Adaptation + courant grille	IR2110, drivers isolés

Tableau 2 : Chaîne de commande d'un convertisseur (synthèse).

6. Commande de moteurs (DC, BLDC, AC)

Moteur DC : pont en H + PWM. La vitesse est proportionnelle à la tension moyenne, le couple au courant. BLDC : commutation synchronisée avec la position rotor (capteurs Hall ou observateur). On génère des PWM sur les phases. Moteur AC (asynchrone) : variateur à onduleur triphasé, commande scalaire V/f ou commande vectorielle (selon niveau).

Dans les applications électrotechniques, on ajoute souvent une mesure de courant pour limiter le couple et protéger le convertisseur.

7. Mesure de fréquence

Deux méthodes :

* Comptage d'impulsions sur une fenêtre Δt ($f \approx N/\Delta t$) : simple, moins précis à basse fréquence
* Mesure de période par capture ($f = 1/T$) : plus précis à basse fréquence.
* Les modules Capture/Compare figent la valeur du timer sur un front montant/descendant.

7.1. Exemple de calcul

Si un timer s'incrémente à 1 MHz et que deux captures successives donnent un écart de 20 000 ticks, alors $T = 20$ ms et $f = 50$ Hz.

8. Système d'acquisition de données

Une acquisition de données comprend : capteurs → conditionnement → ADC → traitement → stockage/transmission.

Le choix de la fréquence d'échantillonnage dépend de la dynamique : mesure réseau 50 Hz (échantillonnage ≥ 1 kHz pour analyse), température (faible). Pour éviter l'aliasing, on ajoute un filtre passe-bas analogique avant l'ADC.

* Conditionnement : adaptation d'échelle, isolation si besoin, filtrage analogique.
* Traitement : moyenne glissante, RMS, détection de pics, seuils d'alarme.
* Communication : UART vers PC, SPI vers carte SD, I^2C vers capteurs.

Conclusion générale

L'étude des microprocesseurs et des microcontrôleurs constitue une base incontournable pour la conception et l'exploitation des systèmes numériques modernes. Au cours de ce document, nous avons mis en évidence que ces composants ne se limitent pas à exécuter un programme : ils assurent des fonctions essentielles de commande, de supervision, de mesure et de communication, tout en respectant des contraintes fortes telles que le temps réel, la fiabilité, la consommation énergétique et la robustesse face aux perturbations électriques.

Les chapitres introductifs ont rappelé les notions fondamentales nécessaires à la manipulation correcte de l'information numérique (représentations, codages), puis ont permis de comprendre l'organisation générale d'un calculateur : circulation des données, rôle des bus, des mémoires et des registres. Cette base architecturale est indispensable pour passer d'une vision "utilisateur" à une vision "système", où l'on maîtrise le fonctionnement interne et les échanges entre les blocs matériels.

La programmation en assembleur et l'étude des interruptions ont ensuite montré l'importance de la structuration du logiciel bas niveau : organisation du programme, gestion des événements asynchrones, accès aux ports d'entrées/sorties et respect des contraintes temporelles. Ces mécanismes restent au cœur des systèmes embarqués, notamment lorsque l'on cherche à minimiser les ressources, à garantir des temps de réponse ou à diagnostiquer des anomalies.

L'introduction des microcontrôleurs a mis en évidence l'avantage de l'intégration matérielle : la présence sur une même puce du CPU, des mémoires et de nombreux périphériques (timers, PWM, ADC, interfaces série) facilite le développement d'applications de contrôle-commande et d'acquisition. L'exemple du microcontrôleur PIC illustre la démarche complète : configuration des ports, génération de bases de temps, acquisition analogique, communication et organisation efficace du programme (initialisation, boucle principale, interruptions).

Enfin, les applications abordées (interface LCD, clavier, génération de signaux, commande de convertisseurs et de moteurs, mesure de fréquence, acquisition de données) montrent clairement comment relier les concepts théoriques à des réalisations concrètes. Elles

constituent une étape essentielle pour développer l'autonomie de l'étudiant : analyser un besoin, proposer une architecture matérielle, développer un driver logiciel, puis valider expérimentalement.

Il est fortement recommandé, pour consolider les acquis, d'accompagner l'étude de ce cours par des travaux pratiques : observation des signaux à l'oscilloscope, tests de stabilité PWM, validation des temporisations, débogage des interruptions et vérification des chaînes de mesure (conditionnement + ADC). Cette approche expérimentale est souvent le meilleur moyen de maîtriser les contraintes réelles d'un système embarqué.

Pour aller plus loin, plusieurs perspectives naturelles peuvent être envisagées : l'étude de microcontrôleurs plus avancés (ARM Cortex-M), l'introduction d'un système temps réel (RTOS) lorsque la complexité logicielle augmente, l'utilisation de bus industriels (CAN, Modbus, Ethernet) pour l'intégration dans des systèmes de supervision, ainsi que l'approche sûreté de fonctionnement (watchdog, brown-out reset, diagnostics, stratégies de repli). Ces prolongements permettent d'élargir les compétences vers des applications industrielles complètes et modernes.

Références bibliographiques

[1]. M. Tischer et B. Jennrich, *La bible PC – Programmation système*, Micro Application, Paris, 1997.

[2]. R. Tourki, *L'ordinateur PC – Architecture et programmation – Cours et exercices*, Centre de Publication Universitaire, Tunis, 2002.

[3]. H. Schakel, *Programmer en assembleur sur PC*, Micro Application, Paris, 1995.

[4]. E. Pissaloux, *Pratique de l'assembleur I80x86 – Cours et exercices*, Hermès, Paris, 1994.

[5]. R. Zaks et A. Wolfe, *Du composant au système – Introduction aux microprocesseurs*, Sybex, Paris, 1988.

[6]. Intel Corporation, *8086 Family User's Manual* (architecture, modes d'adressage, interruptions).

[7]. Intel Corporation, *8086/8088 Programmer's Reference Manual* (jeu d'instructions et programmation).

[8]. Microchip Technology Inc., *PIC16F87XA Data Sheet* (ex. PIC16F877A) : registres SFR, timers, ADC, USART, SPI/I²C, interruptions.

[9]. M. A. Mazidi, R. D. McKinlay, D. Causey, *PIC Microcontroller and Embedded Systems*, Pearson.

[10]. J. B. Peatman, *Design with PIC Microcontrollers*, Prentice Hall.

[11]. R. H. Barnett, S. Cox, L. O'Cull, *Embedded Systems: Architecture, Programming and Design*, Wiley.

[12]. Hitachi, *HD44780U LCD Controller/Driver Datasheet* (commandes, timings, interface 4/8 bits).

ANNEXE

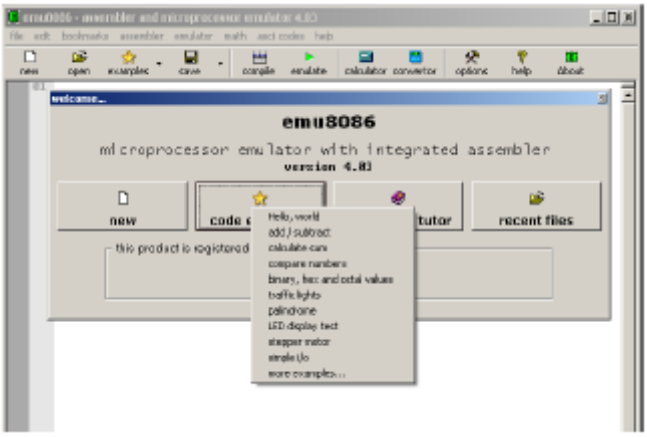
Fiches des TP

TP N° 1 : Initiation à l'outil EMU8086

Le but de ce TP est de familiariser avec l'outil EMU8086, c'est un logiciel plus pratique pour réaliser des programmes en langage assembleur.

Les étapes à suivre pour écrire un programme en assembleur

Lorsque nous démarrons l'outil EMU 8086, l'application nous propose une interface suivant : (voir fig. 1) Si nous appuyons sur le bouton « code example » nous avons accès à une liste d'exemple.



Un exemple de programme simple qui permet de faire une addition entre les contenus de deux registres AX et BX. Le programme est donné comme suit :

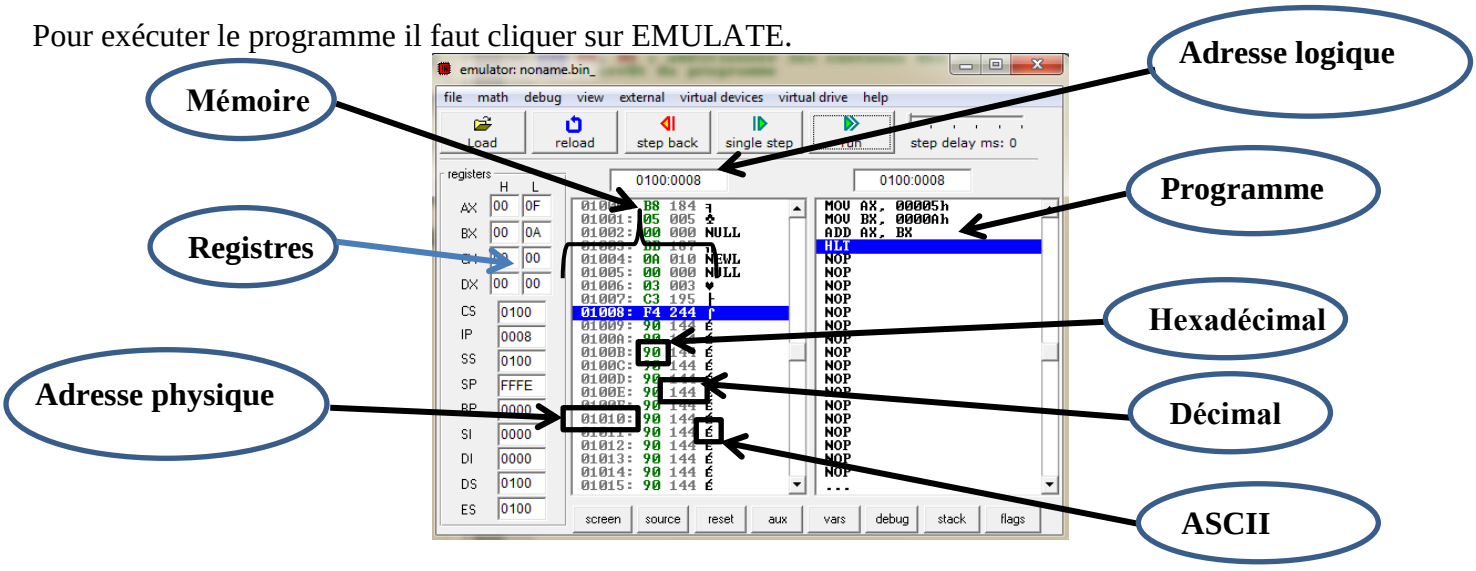
MOV AX, 5 ; transférer une donnée (05H) au registre AX.

MOV BX, 10 ; transférer une donnée (10H) au registre BX.

ADD AX, BX ; additionner les contenus des deux registres

HLT ; arrêt du programme

Pour exécuter le programme il faut cliquer sur EMULATE.



Types d'adressage 8086:

• **Adressage par registre (ou implicite)**
Add AX, BX
• **Adressage immédiat**
Add AX, valeur (1 accès mémoire pour lire la valeur)
• **Adressage direct**
Add AX, [adresse] (2 accès mémoire : adresse puis valeur [adresse])

Instruction de base :

- ✓ **MOV** : transfert d'information

L'instruction la plus utilisée est l'instruction mov, qui copie la valeur d'un opérande ***source*** dans un opérande ***destination***

Format : MOV destination, source

Cinq formes sont possibles :

1) **MOV** reg, reg reg :registre
2) **MOV** reg, mem mem : adresse mémoire
3) **MOV** mem, reg
4) **MOV** reg,immed immed :valeur immediate(binaire, décimale, hexa)
5) **MOV** mem, immed

Notez qu'il n'existe pas de transfert de mémoire à mémoire.

Exemple

```
mov ax, 54h
mov bx, ax
mov cx,bx
```

Opérations arithmétiques:

- ✓ **Add :** addition **; add** destination, source

Exemple

```
Mov al, 12h
Mov bl, 35h
Add al, bl
mov cl,al
mov [bx],10h
add al,[bx]          al= al  +  [DS:BX]
hlt
```

- ✓ **SUB:** soustraction ; **sub** destination, source

Exemple:

```
Mov al, 12h
Mov bl, 35h
sub al, bl
mov cl,al
mov [bx],10h
sub al,[bx]
sub bl,5h
hlt
```

- ✓ **MUL:** multiplication non signée ; **MUL** opérande
  1) Sur 1 octet **: MUL** opérande →AX=AL*opérande
  2) Sur 2octets :**MUL** opérande →DX,AX=AX* opérande

Exemple :

```
Mov al,5h
mov bl,3h
mul bl
mov cx,2f2ch
mov ax,0202h
mul cx
hlt
```

- ✓ **DIV:** division non signée ; DIV, opérande
  1) Sur 1 octet : DIV opérande → AX/opérande→AL=quotient, AH=Reste
  2) Sur 2 octets : DIV opérande → DX,AX/opérande→AX=quotient, DX=Reste

Exemple :

```
MOV AX, 1345H → AX=1345H
MOV BL, 85H   → BL= 85H
DIV BL        → AX/BL=1345H/85H → AL=25H, AH=0CH
MOV DX, 02B6H
MOV CX, 9256H
DIV CX        → DX,AX/CX → 02B60C25H/9256H → AX=04BE BX=1851
```

Le travaille à faire

1. Retracez, par l'exécution pas à pas, le programme suivant, en commentant chaque ligne. Quelle conclusion retirons-nous de ce fragment?

```
ORG100h
Mov ax,243 ; Charger le registre AX par le nombre décimal 243
mov bx,243h  ;…….
add ax,bx ; …….
mov [bx],10h ; ……
sub bx,[bx] ; ……
hlt
```

Compléter le tableau suivant :

Opération	Programme	Résultat (par l'exécution pas à pas)
45h+79h		
12h*04h **45E1* 4F2A**		
45BCh/45h		
(45*8)h/10h		

TP N° 2 : Initiation au micro-ordinateur MTS-86C

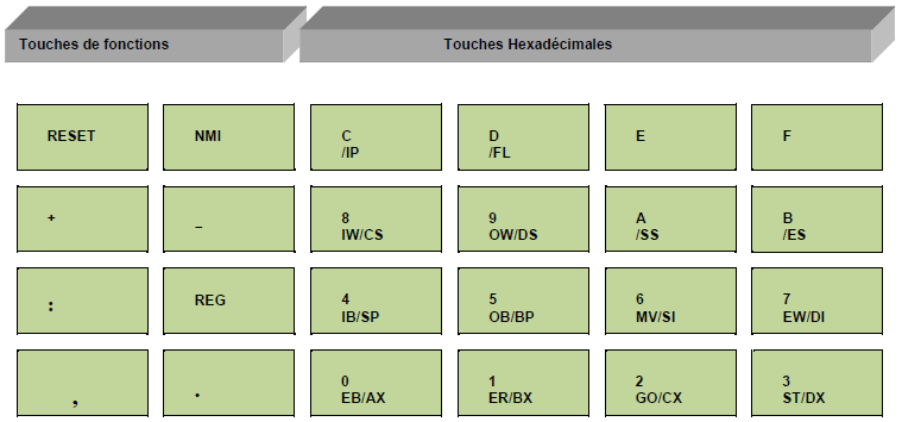
L'objectif de cette manipulation est :

- La familiarisation avec le fonctionnement de base du MTS-86C
- Apprendre à exécuter les principales commandes.

On procèdera à l'introduction de la procédure d'utilisation du Kit MTS-86C en utilisant seulement les touches du clavier pour saisir des programmes simples.

1) Le clavier de la carte MTS-86C

Le clavier de la carte MTS-86C comporte 24 touches et il apparaît comme suit :



Les touches de fonctions

Touches	Description
RESET	La touche RESET vous permet d'initialiser la carte MTS-86C. Quand la touche est pressée le 8086 envoie un message sur le LCD et le monitor est près.
NMI	La touche NMI génère une interruption non masquable de type 2.
+	La touche + vous permet d'additionner 2 valeurs hexadécimale.
-	La touche – vous permet de substituer une valeur hexadécimale d'un autre.
:	Cette touche vous permet de séparer une adresse entrée sur deux parties ; la valeur de segment ou de l'offset.
REG	Cette touche vous permet d'utiliser les contenus des registres du 8086 comme une adresse ou une donnée en entrée.
,	Cette touche sépare les entrées saisies à partir des touches du clavier et incrémente l'adresse mémoire à chaque entrée.
.	Lorsque cette touche est appuyée la commande courante est exécutée. Notez que lorsque la touche GO est appuyée, le 8086 lance l'exécution à partir de l'adresse spécifiée quand la touche est pressée.

Les touches hexadécimales

Touches Hexadicimales	Commandes		Registres	
	Acronyme	Nom	Acronyme	Nom
0 EB/AX	EB	Examine par Octet (Byt)	AX	Accumulateur
1 ER/BX	ER	Examine registre	BX	Registre de Base
2 GO/CX	GO	Go	CX	Registre Compteur
3 ST/DX	ST	Un pas (STEP)	DX	Registre de Données
4 IB/SP	IB	Entrée Octet	SP	Stacker Pointer
5 OB/BP	OB	Sortie Octet	BP	Base Pointer
6 MV/SI	MV	Move	SI	Source Index
7 EW/DI	EW	Examine par Word	DI	Destination Index
8 IW/CS	IW	Entrée Word	CS	Registre Segment de Code (CodeSegment)
9 OW/DS	OW	Sortie Word	DS	Registre Segment de données (Data Segment)
A /SS	non	N/A	SS	Registre Segment de pile (Stack Segment)
B /ES	non	N/A	ES	Registre Segment extra (Extra Segment)
C /IP	non	N/A	IP	Pointeur d'instruction
D /FL	non	N/A	FL	Flag
E	non	N/A	non	N/A
F	non	N/A	non	N/A

2) **Initiation à la programmation en langage machine**

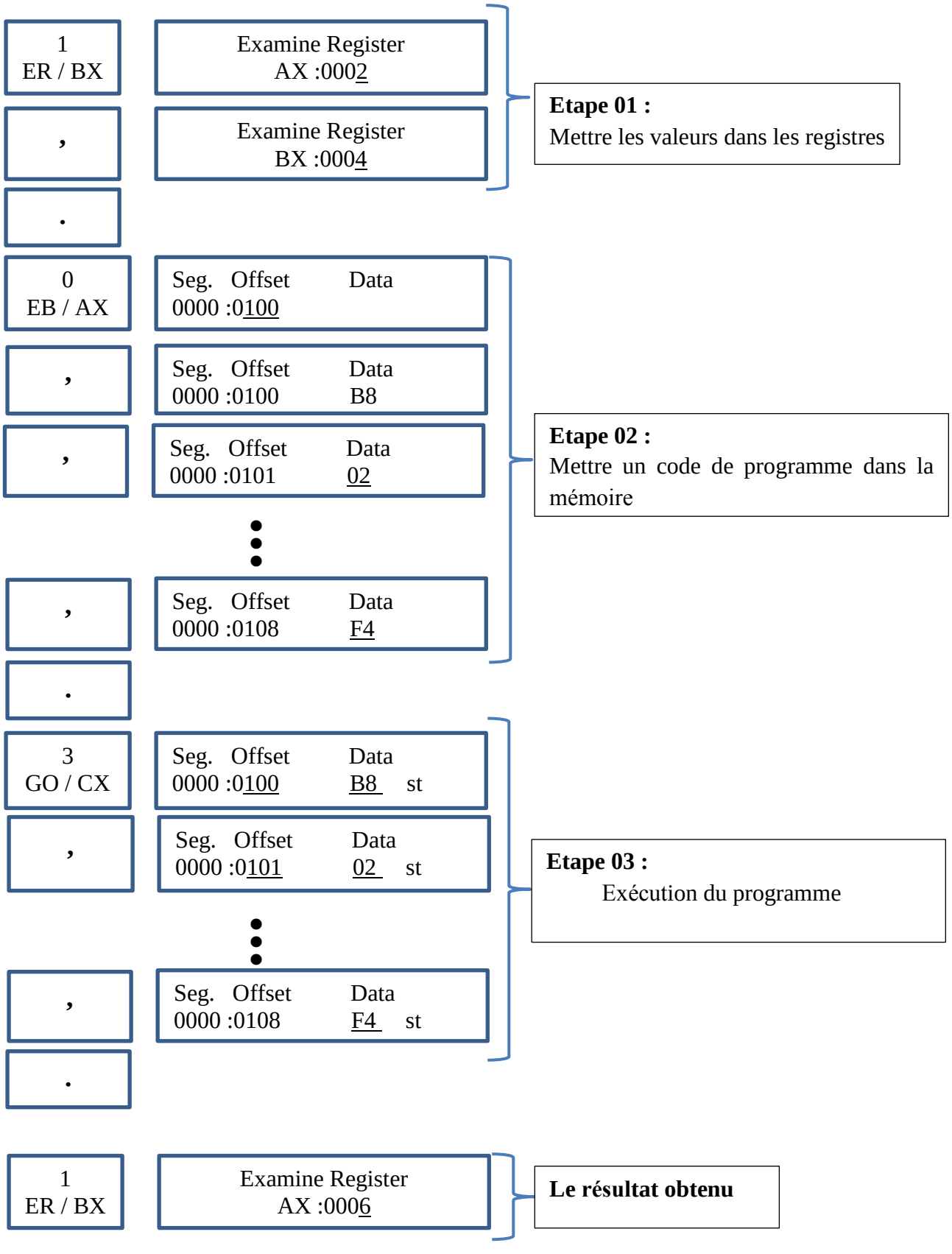
Soit le programme en CODE SEGMENT

mov ax,2h	**B80200**
mov bx,4h	**BB0400**
add ax,bx	**03C3**
hlt	**F4**

L'exécution du n'importe quel programme sur MTS-86C passe en étapes :

a) Mettre les valeurs dans les registres
b) Mettre un code de programme dans la mémoire
c) Exécuter le programme

Manipulation :



Le travail à faire

1. Comparer l'utilisation du langage machine et du langage assembleur.
2. Ecrit un programme qui permettre de faire l''opération suivante et complété le tableau

$$(5F2C - 37B0)h$$

Opération	Langage assembleur	Langage machine
$(5F2C - 37B0)h$	**Mov ax,5F2C**	

3. Taper les étapes à suivie pour exécuter ce programme sur le kit MTS-86C

TP N° 3 : Initiation à la programmation en assembleur (Les instructions logiques)

But du TP

Le but de cette manipulation est :

• Utilisation des instructions logiques du µp 80x86 en utilisant le MTS-86C.

• Déterminer les différents indicateurs impliqués pour chaque opération logique

Rappel

Les instructions logiques ce sont des instructions qui permettent de manipuler des données au niveau des bits. Les opérations logiques de base sont : ET(AND), OU (OR), OU exclusif (XOR), complément à 1 (NOT), complément à 2 (NEG), décalages et rotations.

1. Instruction AND: AND destination, source

MOV BL, 35H
AND BL , 0FH ; BL AND 0Fh → BL (05h).

MOV AL, 45H
MOV BL, 9CH
AND AL, BL AL AND BL→ AL(04H)

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

L'instruction AND est utilisée pour:

✓ Masquer (mise à 0) certains ou tous les bits (AND AL, 00h)

2. Instruction OR: OR destination, source

MOV AX, 857H

MOV BX, 54CH

OR AX, BX **AX→D5FH**

MOV AL, 7BH

OR AL, 0DAH **AL→FBH**

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	0

L'instruction OR est utilisée pour:

– Mettre à 1 certains ou tous les bits (OR BL, FFh).
– Tester si un opérande est "0"

3. Instruction XOR: XOR destination, source

MOV DH, 54H
XOR DH , 78H ; **DH XOR 78h → DH (2Ch).**

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

L'instruction XOR est utilisée pour:

- ✓ Mettre à 0 le contenu d'un registre (XOR AL, AL), AL → 00h
- ✓ Comparer deux registres si leur contenus sont égaux (XOR AX, CX).

Si c'est les cas ZF → 1 et le résultat est sauvegardé dans la destination (0000 →AX)

4. **Instruction CMP** (Compare) CMP destination, source

```
MOV BX, 7888H
MOV CX, 9FFFH
CMP BX, CX ;        BX < CX →   CF=1
Hlt
```

Comparer opérandes	ZF	CF
Dest. > Source	0	0
Dest. = Source	1	0
Dest. < Source	0	1

***Note*:** les contenus de (BX, et CX) ne changent pas après CMP

5. **instruction: NOT (Complément à 1)** NOT opérande

```
MOV AL, 36H
NOT AL
HLT

MOV CX,10111001B
NOT CX
HLT
```

Manipulation

1. Soit le programme suivant écrit en assembleur

Mnémoniques	Registre AL resultat	Résultat théoriquement
MOV AL,00000001B		
NOT AL		
AND AL,00001111B		
MOV AL,10101010B		
XOR AL,11001100B		
HLT		

Exécuter le programme étape par étape et écrire les résultats trouvez en AL et comparer les résultats théoriques obtenus avec ceux obtenus par l'exécution du programme.

2. Soit une valeur dans le registre AL

- ✓ Donner l'instruction qui permet de mettre à 1 les bits 0,2,4 et 6 de la valeur contenue dans AL sans modifier les autres bits.
- ✓ Donner l'instruction qui permet de mettre à 0 les bits 0,2,4 et 6 de la valeur contenue dans AL sans modifier les autres bits.
- ✓ Donner l'instruction qui permet d'inverser les bits 0,2,4 et 6 de la valeur contenue dans AL sans modifier les autres bits.

TP N° 2 : Initiation au micro-ordinateur MTS-86C

Exercice 01 :

```
MOV  AX,2105H        ?
MOV   BX,0123H       ?
MOV   CX,AX          ?
MOV  AX,BX           ?
MOV  BX,AX           ?
HLT                  F4
```

- ✓ Quelle fonction ce programme réalise-t-il?
- ✓ Déterminer le code machine et établir les étapes nécessaires à l'exécution du programme.
- ✓ Quelles seront les valeurs dans les registres AX, BX, après exécution de ce programme ?

Exercice 02 :

A= (11001011)b , B= (10011001)b ,

- ✓ Ecrire un programme qui permet de faire les opérations logiques suivantes :

  A AND B **A OR B** **complément à 1 (A)**
- ✓ Déterminer le code machine et établir les étapes nécessaires à l'exécution du programme.
- ✓ écrire le programme qui permet de trouve le plus grand nombres et commenté les résultats trouves.

TP N° 5: Introduction à l'Arduino

Introduction

ARDUINO est une **plateforme matérielle et logicielle de développement** d'applications embarquée.

Côté matériel, elle se compose d'une carte électronique basée autour d'un **microcontrôleur (**ATMEL AVR) comportant un certain nombre d'entrées et de sorties (les ports) permettant la connexion de capteurs, ou d'actionneurs.

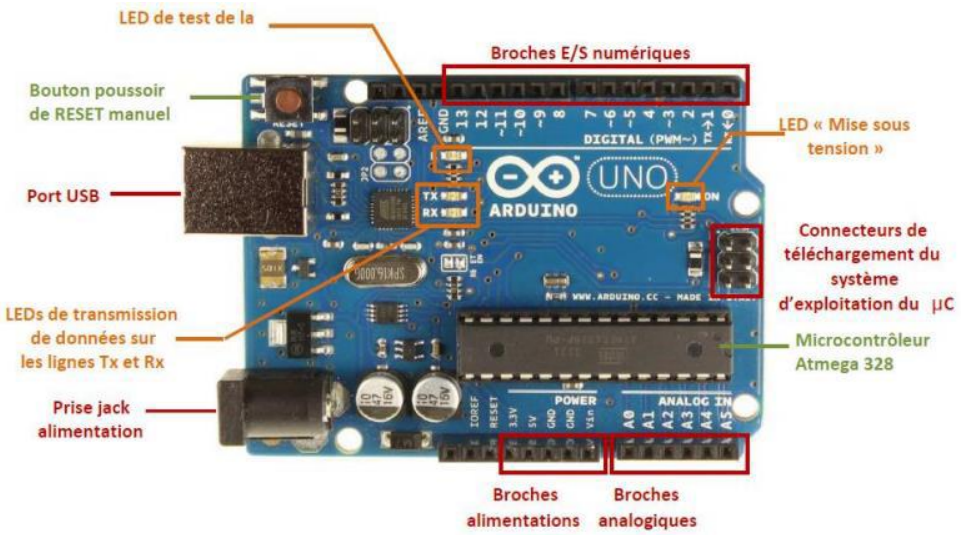
Le logiciel de programmation des modules ARDUINO est une **application Java** servant d'éditeur de code qui peut transférer le programme a travers de la liaison USB. Le langage de programmation utilisé est un **mélange de C et de C++,** restreint et adapté aux possibilités de la carte.





La carte ARDUINO UNO

Le modèle UNO de la société ARDUINO est une carte électronique dont le coeur est un microcontrôleur ATMEL de référence ATMega328. Le microcontrôleur ATMega328 est un microcontrôleur 8bits de la famille.



Fonctions de base

Pour manipuler l'Arduino, il y a un ensemble de fonctions fréquemment utilisées presque dans chaque programme. Dans cette section, on essaie de mentionner quelques-unes et les expliquer selon le référentiel officiel de l'Arduino (https://www.arduino.cc/reference/en/)

Nous commençons par la première fonction qui permet de configurer les ports de l'Arduino, soit en entrée (ou input) soit en sortie (ou output). C'est la fonction **pinMode(pin, mode)** qui prend en charge de deux paramètres, i.e. le numéro du port (pin) et le mode (INPUT, OUTPUT ou INPUT_PULLUP). Le mode OUTPUT configure le port en étant sortie, ce qui veut dire un signal sort du port de la carte vers le dispositif. Les deux modes

INPUT et INPUT_PULLUP configurent le port en étant entrée, ce qui veut dire un signal sort du dispositif vers le port de la carte. Cependant, la différence entre les deux réside dans l'état du bouton quand il est appuyé, où dans le mode INPUT_PULLUP on lit l'état HIGH quand le bouton est en repos et l'état LOW quand il est appuyé.

Après la configuration du port, on effectue soit une lecture soit une écriture, et pour faire cela on utilise des fonctions dépendant du type du port (numérique ou analogique). Pour envoyer un signal via un port numérique, on utilise la fonction **digitalWrite(pin, value)** qui prend en charge un numéro de port et la valeur à écrire (soit HIGH/1 ou soit LOW/0). Bien évidemment le contraire de la fonction ***digitalWrite*** est **digitalRead(pin)** qui prend en charge d'un seul paramètre (le numéro du port) et renvoie une valeur booléenne (soit HIGH/1 ou soit LOW/0).

Les fonctions similaires utilisées avec les ports analogiques sont **analogWrite(pin, value)** et **analogRead(pin)**, où le deuxième paramètre de la première fonction est une valeur numérique entre 0 et 255. La fonction analogRead renvoie une valeur numérique comprise entre 0 et 1023 ce qui correspond à 0-5v, donc on applique la règle des trois pour trouver le voltage d'entrée. Par exemple, lorsqu'on lit la valeur 980, elle correspond à 4.7v (962 * 5 / 1023).

Pour faire un délai de retard entre les instructions du programme, on utilise la fonction **delay(milliSeconds)** dans laquelle on passe le temps en milli secondes. Cette instruction met la carte ou le microcontrôleur en repos pendant un moment tout en gardant l'état des ports tels qu'ils sont avant cette instruction. Donc, elle est très utile dans les protocoles de communication pour régler la fréquence du signal inter-échangé, ainsi pour faire des animations (e.g. clignotement d'une LED).

Il y a une autre alternative de la fonction ***delay*** et qui travaille sur l'ordre de micro secondes, c'est la fonction **delayMicroseconds(microSeconds)** qui reçoit la valeur en micro secondes comme paramètre. Dans la même famille des fonctions, on trouve la fonction **millis()** qui renvoie le nombre de milli secondes passés depuis le démarrage de la carte Arduino (branchement de l'alimentation).

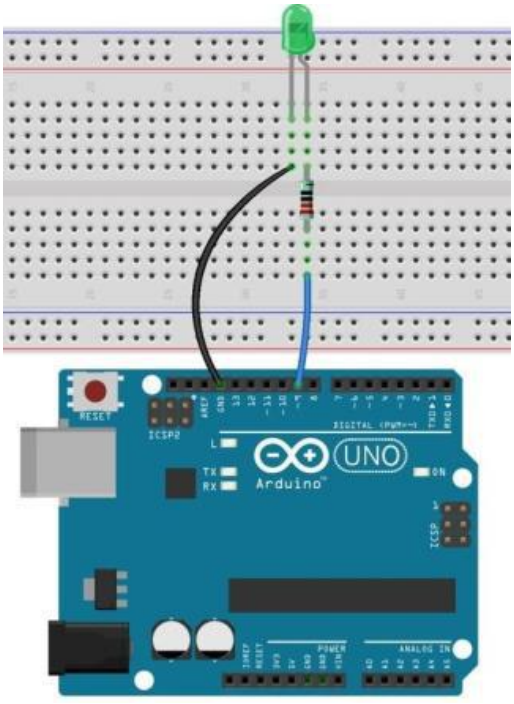
Enfin, si on veut échanger des données entre la carte Arduino et l'ordinateur en temps réel via le câble USB en utilisant le bus UART, on a trois fonctions dédiées. La première fonction est la fonction ***begin*** (de l'objet ***Serial***) qui initie la communication avec l'ordinateur, elle reçoit en paramètre la vitesse de communication en baud (ou bits par seconde). Donc, on fait appel à ***begin*** dans la fonction principale **setup()**, où on précise la vitesse telle que 4800, 9600, 19200, etc.

Pour écrire une valeur (numérique ou autre) de l'Arduino vers l'ordinateur, ou autrement on affiche une valeur acquise depuis l'Arduino, on fait appel à la fonction **print(value)** ou **println(value)** appartenant au même objet ***Serial***. Donc, on écrit **Serial.print(value)** ou **Serial.println(value)**. On peut même spécifier le format d'affichage tel que BIN (binaire), OCT (octal), DEC (décimal) et HEX (hexadécimal) en mettant un de ces formats comme deuxième paramètre de la fonction, par exemple **Serial.println(value, HEX)**. On note que la différence entre les deux fonctions d'affichage ***print*** et ***println*** réside dans le retour à la ligne (***line feed*** en anglais) ajouté par la deuxième fonction (**println**).

Pour faire l'inverse et écrire une valeur depuis l'ordinateur vers l'Arduino, ou acquérir une valeur par l'Arduino depuis l'ordinateur, il y a une multitude de fonctions selon l'usage et l'application. Par exemple, la fonction **Serial.read()** lit les données octet par octet et la fonction **Serial.readBytes(buffer, size)** lit les données en vrac et les sauvegarde dans un buffer avec une taille passée comme paramètre. Il y a également deux fonctions qui convertissent les données d'entrée en valeur numérique entière ou réelle, ces fonctions sont **Serial.parseInt()** et **Serial.parseFloat()**.

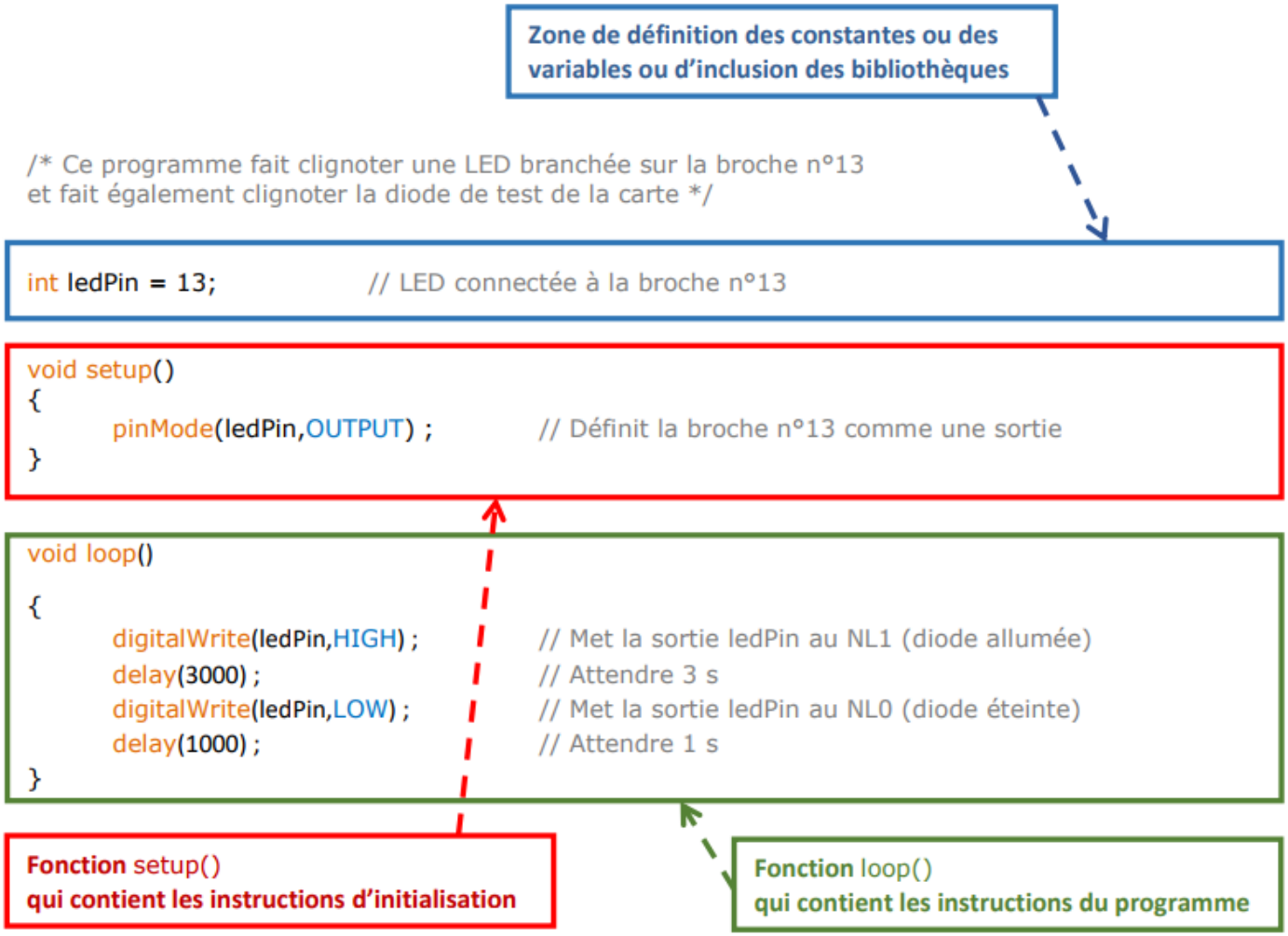
Application

1. **Contrôle d'une LED avec un Arduino**



Donc, on relit le pole positif (anode) de la LED à n'importe quel port numérique de l'Arduino (par exemple le port 13), et le pole négatif (cathode) à une extrémité de la résistance et l'autre extrémité de celle-ci au GND de l'Arduino

Après avoir fait le branchement, on passe à la programmation de la carte Arduino en utilisant le logiciel Arduino IDE. Donc, une fois le logiciel est démarré, on crée un nouveau projet et on le nomme « controle_LED ». Bien évidemment le port sur lequel la LED est branchée doit être configuré en étant sortie, car on a besoin d'un signal qui sort de l'Arduino vers la LED. Plus précisément, soit il y a un courant sortant qui allume la LED, soit il n'y a pas de courant et la LED s'éteint.



La première partie permet de **définir les constantes et les variables** en déclarant leur type. Elle permet également l'inclusion des bibliothèques utilisées dans le programme au moyen de #include.

La fonction setup() contient **les instructions d'initialisation ou de configuration des ressources** de la carte comme par exemple, la configuration en entrée ou sorties des broches d'E/S numériques, la définition de la vitesse de communication de l'interface série, etc.. Cette fonction **n'est exécutée qu'une seule fois** juste après le lancement du programme.

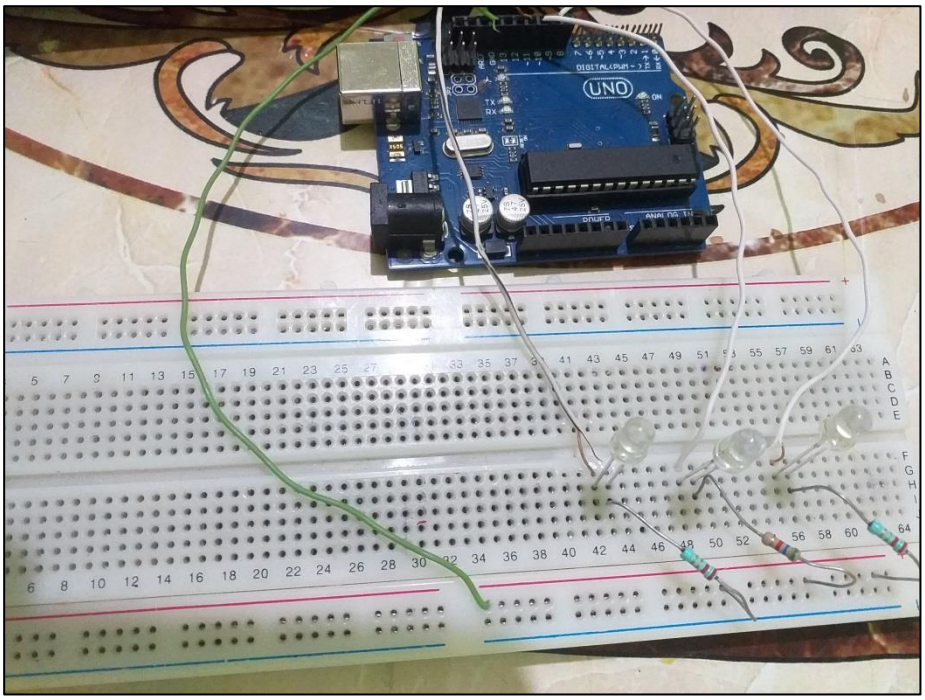
La fonction loop() contient **les instructions du programme** à proprement parlé. Cette fonction sera **répétée indéfiniment** tant que la carte ARDUINO restera sous tension.

2. **Contrôle de plusieurs LEDs avec un Arduino**

Réaliser le schéma et le programme suivant

```
Fichier Edition Croquis Outils Aide
Button
void setup() {
pinMode(9,OUTPUT);
pinMode(10,OUTPUT);
pinMode(11,OUTPUT);
}

void loop() {
digitalWrite(9,HIGH);
digitalWrite(10,LOW);
digitalWrite(11,LOW);
delay(100);
digitalWrite(9,LOW);
digitalWrite(10,HIGH);
digitalWrite(11,LOW);
delay(100);
digitalWrite(9,LOW);
digitalWrite(10,LOW);
digitalWrite(11,HIGH);
delay(100);
}
```



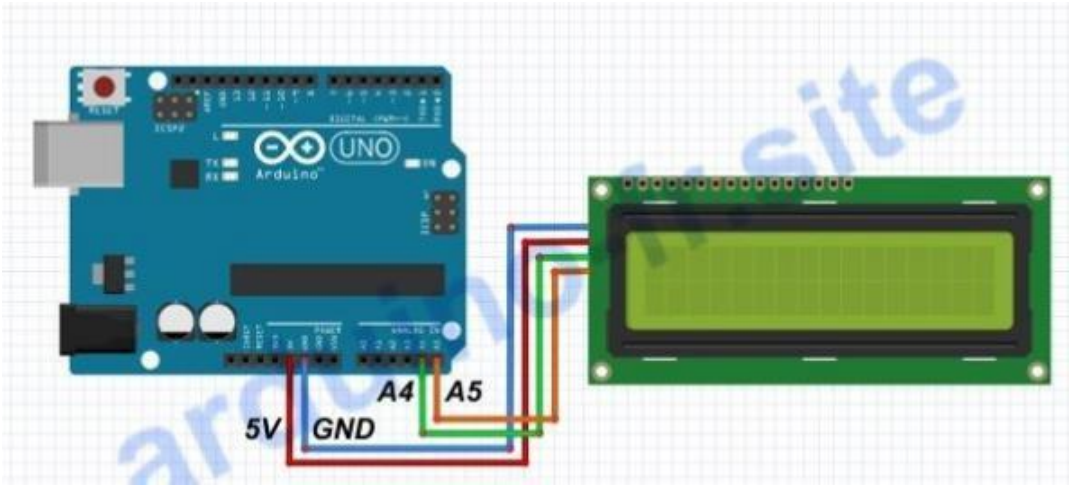
Le travail à faire

1. Modifier le programme précédant pour faire les commande suivante
   - ✓ Allumer la 1er led puis la 2eme led sans étendre la 1er puis allume la 3éme led.
   - ✓ Etendre 1er led puis la 2eme led puis la 3éme led.
2. Faire un programme qui permet de clignote les 3 lEDs au même temps avec un délai de 100ms.
3. **Traffic signal :** faire **un programme** de feux de circulation selon les conditions suivante
   1. Allumer la 1er LED (red) pondant 3s
   2. Faire clignoter la 2eme lED (oange) 3fois
   3. Allumer la 3eme LED (red) pendant 3s

TP N° 6:Affichage sur écran LCD et mesure de température avec le capteur LM35 avec Arduino

Application 1 : afficheur LCD

Soit le montage suivant :



Le montage a pour objective d'afficher des textes sur le LCD.

- **Câblage**

* GND → GND
* VCC → 5V
* SDA → A4
* SCL → A5

Pour cela on réalise le programme suivant :

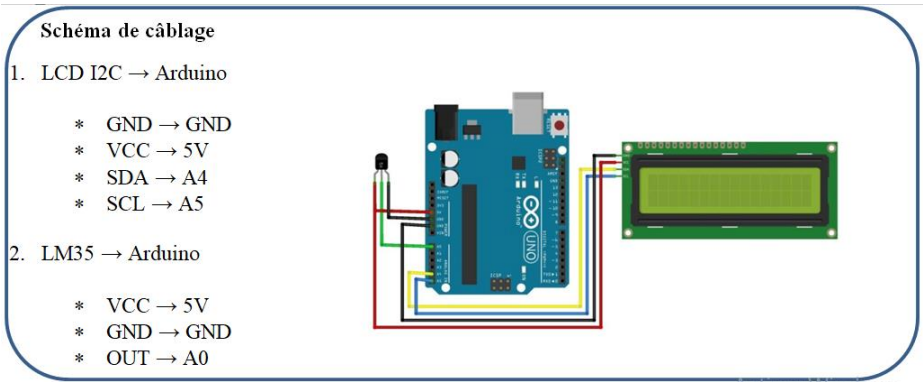
```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup()
{
 // initialize the LCD
 lcd.init();
// Turn on the blacklight and print a message.
 lcd.backlight();
 lcd.setCursor(0,0);
 lcd.print("electrotechnique");
}
void loop()
{

}
```

Le travail à faire

- Réaliser le montage et exécutée le programme
- Faites bouger l'écriture avec l'instruction **lcd.scrollDisplayLeft();** avec une délai de 0.5s de façon répétitive.

Application 2: Mesure et affichage de la température sur écran LCD à l'aide d'un capteur LM35
Soit le montage suivant :




```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const int lm35Pin = A0;
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  lcd.init();         // Initialisation du LCD
  lcd.backlight();    // Activer le rétroéclairage
}

void loop() {
  int lecture = analogRead(lm35Pin);          // Lire la valeur du LM35
  float tension = lecture * (5.0 / 1023.0);   // Conversion en tension
  float temperature = tension * 100.0;        // Conversion en °C

  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(temperature, 1);  // Affiche avec 1 décimale
  lcd.print((char)223);       // Symbole °C
  lcd.print("C");

  delay(1000);                // Mise à jour toutes les 1s
}
```

Le travail à faire

- Réaliser le montage et exécutée le programme ; visualiser les résultats
- On souhaite ajouter deux LEDs à notre montage avec le capteur LM35 :
  - ✓ **LED verte** → s'allume si la température est normale (inférieure ou égale à 35 °C)
  - ✓ **LED rouge** → s'allume si la température dépasse 35 °C
- Écrivez un extrait de code Arduino qui allume la LED verte quand la température est normale et la LED rouge si la température dépasse 35 °C.

Rappel: La structure **`if ... else`** permet de **prendre des décisions** dans le code selon une condition.

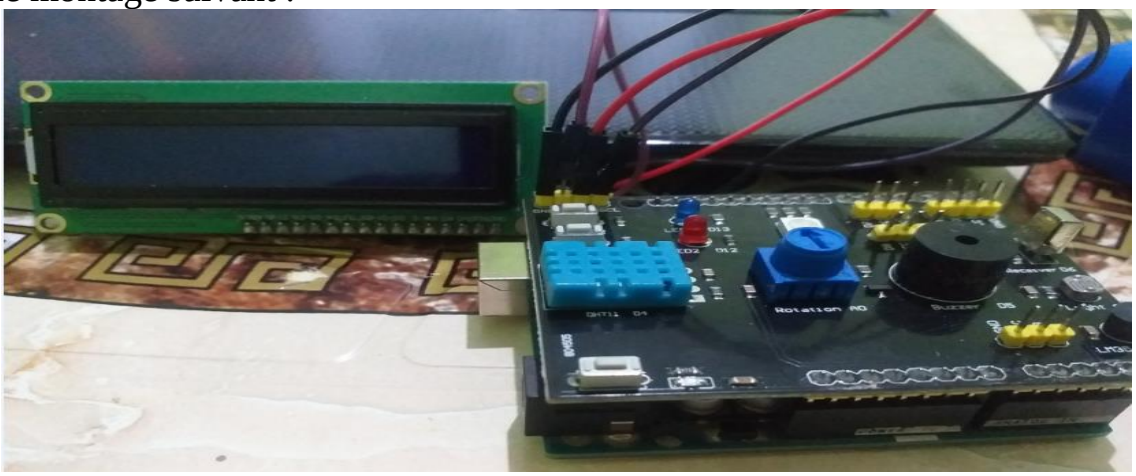
Syntaxe générale

```
if (condition) {
   // Instructions à exécuter si la condition est vraie
} else {
   // Instructions à exécuter si la condition est fausse
}
```

TP N° 8: Mesure de la température et l'humidité à l'aide d'un montage Arduino

Application 1 : afficheur LCD

Soit le montage suivant :



Le montage a pour objective d'afficher la température et l'humidité sur le LCD.

Pour cela on réalise le programme suivant :

```
#include <DHT.h>
#include<LiquidCrystal_I2C.h>
DHT dht(4, DHT11);
int temp;
int humedity;
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup() {
  dht.begin();
  lcd.init();
  lcd.backlight();
}
void loop() {
  delay(2000);
  temp= dht.readTemperature();
  humedity= dht.readHumidity();
  lcd.setCursor(0,0);
  lcd.print("Temperature: ");
  lcd.print(Temperature);
  lcd.print("C");
   lcd.setCursor(0,1);
  lcd.print("humidity: ");
  lcd.print(humidity);
  lcd.print("%");
}
```

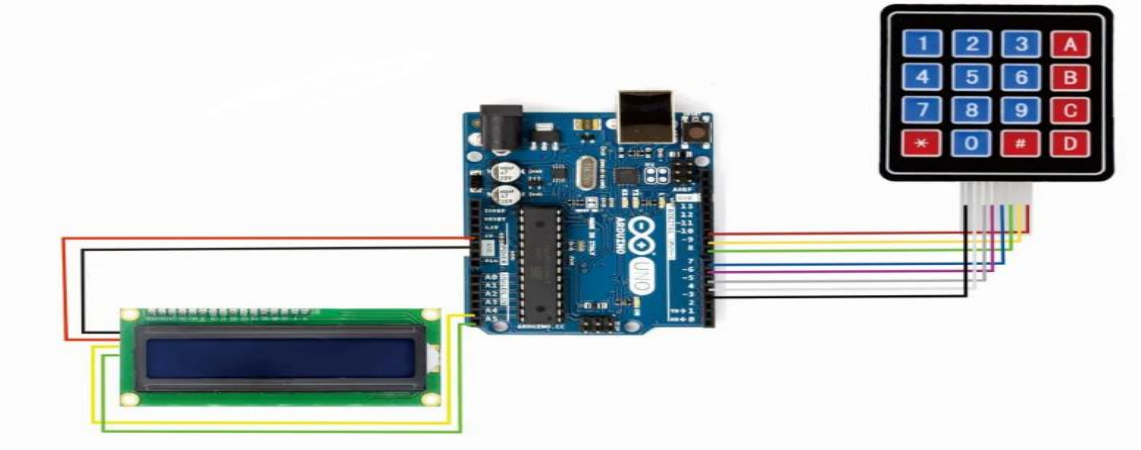
Le travail à faire

- Réaliser le montage et exécutée le programme
- Faites bouger l'écriture avec l'instruction **lcd.scrollDisplayLeft();** avec une délai de 0.5s de façon répétitive.

TP N° 9: Utiliser un afficheur LCD et un clavier Hexa avec Arduino

Application : Making a password lock

Soit le montage suivant :



L'objectif de ce tp est de savoir créer un mot de passe d'accès qui pourra utiliser dans de des systèmes de sécurité.Pour cela on réalise le programme suivant

```
#include <Keypad.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
char a; char b; char c; char d;
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char hexaKeys[ROWS][COLS] = {
 {'1','2','3','A'},
 {'4','5','6','B'},
 {'7','8','9','C'},
 {'*','0','D','#'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {5, 4, 3, 2}; //connect to the column pinouts of the keypad
Keypad myKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);
void setup()
{
 lcd.init();lcd.backlight(); lcd.setCursor(0,0);}
```

```
 void loop(){
lcd.cursor();
  a=myKeypad.getKey();
  while(a == NO_KEY)a=myKeypad.getKey();
  lcd.setCursor(0,0); lcd.println(a); delay(500);  lcd.setCursor(0,0);  lcd.println("*");
  b=myKeypad.getKey();
  while(b == NO_KEY)b=myKeypad.getKey();
  lcd.setCursor(1,0);lcd.println(b);delay(500);lcd.setCursor(1,0); lcd.println("*");
  c=myKeypad.getKey();
  while(c == NO_KEY)c=myKeypad.getKey();
lcd.setCursor(2,0); lcd.println(c);  delay(500); lcd.setCursor(2,0); lcd.println("*");
  d=myKeypad.getKey();
  while(d == NO_KEY)d= myKeypad.getKey();
  lcd.setCursor(3,0); lcd.println(d);  delay(500);  lcd.setCursor(3,0);  lcd.println("*");  lcd.noCursor();
if(a=='1' && b=='2' && c=='3' && d=='4')
{ lcd.clear();
  lcd.setCursor(1,0);
  lcd.println("*** Verified ***");
     delay(3000);
   lcd.clear();
}
else if (a!='1' || b!='2' || c!='3' || d!='4')
  {  lcd.setCursor(1,0);
  lcd.println("Invalid Key!");
  delay(1000);lcd.clear();
  }}
```

Le travail à faire

- Réaliser le montage et exécutée le programme ; visualiser les résultats
- Ajouter 2 LEDs (red and green) aux montages tels que la 1er LED (green) s'allume si le code est correct, sinon la 2eme LED (red) se clignote.