

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/305681298>

Travaux pratique en programmation assembleur des systèmes à base d'un processeur 8086

Book · May 2016

CITATIONS

0

READS

27,349

2 authors:



Belkacem Benadda

Abou Bakr Belkaid University of Tlemcen

39 PUBLICATIONS 76 CITATIONS

[SEE PROFILE](#)



Beldjilali Bilal

Agence Spatiale Algérienne

20 PUBLICATIONS 66 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



New Software Defined Radio SDR Receiver development for New Wireless Telecommunications Generations Systems [View project](#)



contributions of augmentation Systems in Algeria [View project](#)

République Algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Abou Bekr Belkaid Tlemcen
Faculté de Technologie – Département de Télécommunications

Travaux pratique en programmation assembleur des systèmes à base d'un processeur 8086



Avec rappels de cours, corrigés et programmes types

Rédigé par
Belkacem BENADDA
Bilal BELDJILALI

Année 2016

*Ce travail est dédié à mes parents, ma chère épouse,
mes frère et sœurs, toute ma famille, sans oublier mes amis,*

*Une attention particulière est dédiée au personnel gérant
des laboratoires d'électronique.*

Belkacem BENADDA

*Ce travail est dédié à mes parents,
mes frère et sœurs, toute ma famille, sans oublier mes amis.*

Bilal BELDJILALI

Préface

La compréhension du langage assembleur est une étape qui permet de développer des applications avancées en relation avec le matériel. Un aspect souvent rencontré en télécommunications avec des systèmes qui assurent la transmission :

- en faisant clignoter une LED, cas d'une télécommande de télévision par exemple ou une émission sur une fibre optique,
- générer des impulsions via un convertisseur analogique numérique,
- contrôler un module GSM,
- jouer des fréquences sur un simple hautparleur, cas d'une numérotation téléphonique par fréquences vocales,

pour ne citer que quelques exemples. Des exemples que nous évoquons et que nous essayerons d'approcher les principes sur les travaux pratiques abordés. Bien que les privilèges de la programmation assembleur soient mis en évidence dans ce manuel, nous voulons également faire apprendre aux étudiants les limites de cette dernière et expliquer les raisons pour lesquelles la programmation de haut niveau prime. Les notions abordées sont traitées graduellement avec des degrés de difficultés croissants. Afin de pouvoir programmer convenablement, les étudiants sont également appelés à lire, analyser et comprendre les schématiques proposés avec les différents énoncés.

Les outils qui sont utilisés se présentent sous la forme d'un système embarqué à base d'un CPU Intel 8086, Micro Assembleur MASM comme assembleur et debugger. Nous ouvrons une parenthèse sur l'origine de cette appellation qui est née au temps des premiers ordinateurs volumineux ayant comme casse-tête de dysfonctionnement des petits insectes (bugs en anglais) qui perturbaient les connexions électriques, la nature des dysfonctionnements des systèmes actuels, au niveau binaire, cause un casse-tête bien plus sérieux, d'où l'appellation debugger.

Ce manuel est organisé comme suit. Un premier chapitre aborde une présentation générale du système utilisé, une description des outils de base et les principes généraux pour mener convenablement les différents travaux pratiques. Les éléments de la programmation assembleur et la structure des programmes écrit en assembleur sont décrits à ce niveau. Toutefois, nous

sommes conscients de la richesse des instructions de bas niveau dédiées à la programmation assembleur, notre expérience durant les séances des travaux pratiques a montré, qu'il est quasiment impossible de retenir avec précision les mnémoniques des instructions utilisées, la raison qui nous a poussé à énumérer avec chaque énoncé les instructions de base à utiliser. Bien évidemment les étudiants sont libres d'utiliser des instructions proposées à partir d'un travail personnel. Le premier chapitre se termine avec un énoncé d'un travail pratique d'initiation où l'analogie avec l'outil primitif de débogage Windows *Debug* est expliquée. Le second chapitre va aborder principalement des manipulations sur les interfaces parallèles. Allumer un afficheur sept segments, contrôler des LED, intégration d'un compteur et d'une minuterie, une partie est consacrée à des manipulations sur l'écran à Base de LEDs. Bien évidemment, les manipulations proposées à chaque niveau font appels aux notions graduellement acquises, nous avons procédé de la même manière sur les différents énoncés des Travaux Pratiques.

Nous souhaitons que ce travail trouve un enthousiasme exaltant à l'image de celui, que nous avons vécu, exprimé par les étudiants qui ont travaillé avec ces travaux pratiques dès l'année universitaire 2010-2011. Nous tenons à exprimer nos chaleureux remerciements aux personnels des laboratoires d'électronique pour l'écoute l'aide et le soutien qu'ils ont accordé pour faire réussir dans des meilleurs conditions ces travaux pratiques. Nous également à exprimer notre meilleur gratitude à Mr BECHAR Hacène et Mr NEMMICHE Ahmed pour les corrections recommandations et améliorations proposées pour ce polycopiés.

Chapitre 1

Présentation

générale du système

I. Introduction

Le 8086 à aujourd'hui presque 40 ans d'existence, commercialisé officiellement le 8 juin 1978, par l'entreprise Intel au-alentour de 40 000 Dinars Algériens de l'époque [1-5]. Il est toujours commercialisé de nos jours et ne coute pas plus de 500 Dinars Algérien [6]. Avec 133 instructions le 8086 d'Intel implémente une architecture CISC (Complexe Instruction Set Computer) un espace adressable d'un méga octet et des manipulations de données codées sur 16 bits [2-4, 7, 8]. Ce processeur a défini l'architecture de base utilisée jusqu'à nos jours dans les processeurs modernes de la firme Intel [8]. En effet, les processeurs Intel peuvent fonctionner en mode dit réel en activant un environnement de programmation identique à celui du 8086 [8]. Dans ce chapitre nous allons décrire le système de développement que nous avons utilisé pour assurer les travaux pratiques dédiés aux systèmes embarqués, un système dont le cœur est un processeur Intel 8086.

II. Description générale de la carte de développement

Le processeur 8086 d'Intel a été utilisé pour la fabrication des premiers PC commercialisés par IBM. Le système que nous utilisons est un ordinateur avec une architecture similaire aux premiers PC, sauf qu'il est interfacé avec des périphériques dédiés à une initiation au développement des systèmes embarqués, la *figure I-1* est une photographie prise de la carte de développement utilisée et la *figure I-2*[9] est une description synoptique de la composition de cette dernière. Cette carte de développement possède :

- plusieurs interfaces : parallèles et séries,
- un convertisseur analogique numérique,
- un convertisseur numérique analogique,
- des périphériques d'entrée sortie : écran LCD 2 X 16 caractères et un clavier à 25 touches,
- des pins à usage générales avec lesquelles il est possible d'interfacer différents genres de périphériques.



figure-I-1 : Carte de développement 8086 utilisée.

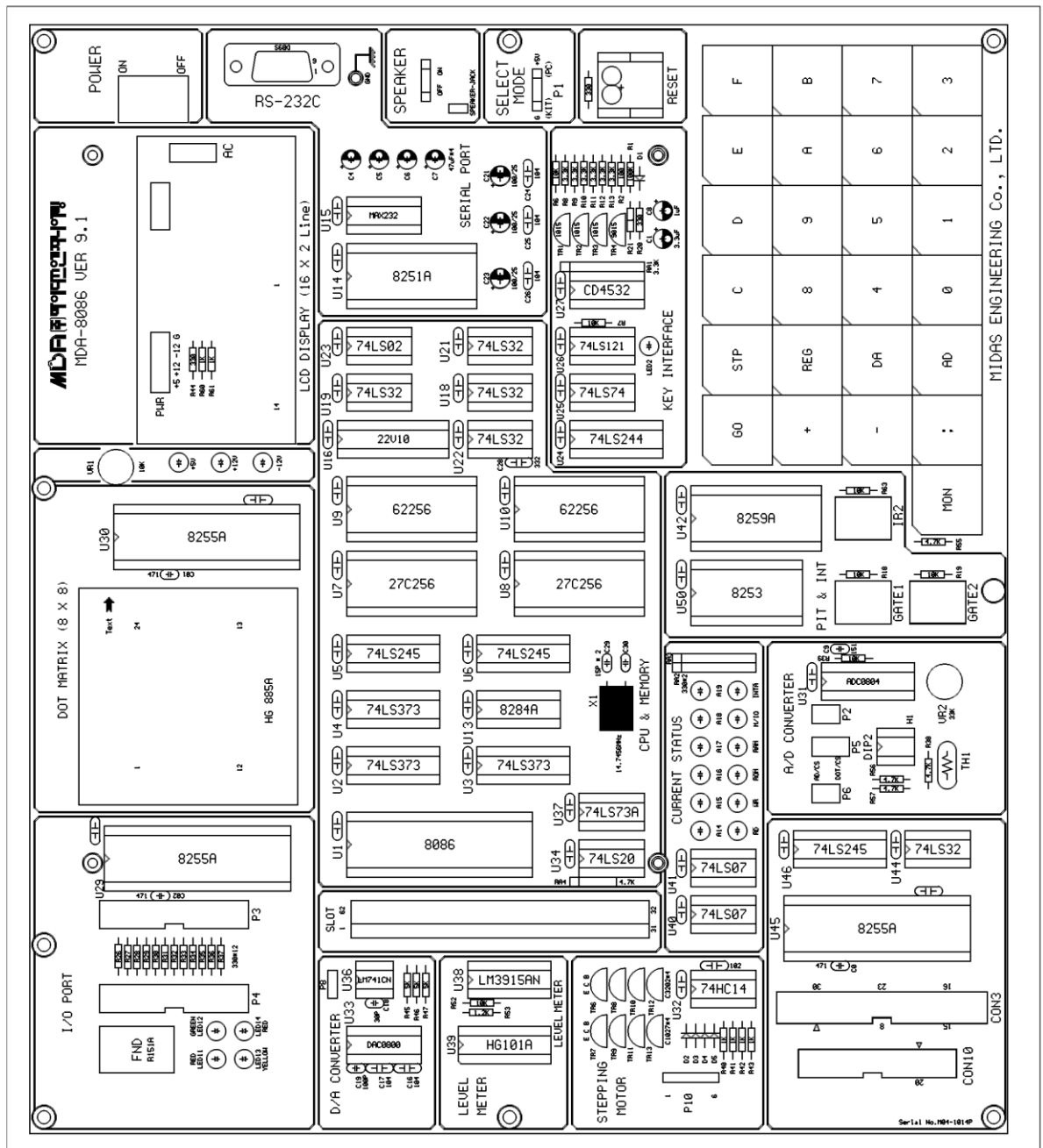


figure-I-2 : Schéma synoptique
de la configuration de la carte de développement 8086 utilisée [9].

La répartition de l'espace mémoire implémentée dans cette carte de développement est représentée par le tableau suivant :

Tableau-I-1 répartition de l'espace mémoire [9] :

Plage d'adresse en notation hexadécimale	Affectation
00000H 0FFFFH	64 ko RAM Statique
10000H EFFFFH	Espace extension
F0000H FFFFFH	64 ko Firmware, Programme moniteur

Le firmware installé dans la ROM, est responsable du transfert des données et programmes à partir d'un PC vers l'espace mémoire réservé dans la RAM statique, ce transfert est assuré via une connexion série RS232. Le firmware assure également la gestion des entrées sorties de cette carte de développement. La contrainte est la limite de 64 Ko imposée pour les programmes et données utilisateur. Une contrainte qui pousse à écrire des programmes optimisés, une des spécificités du développement dans le cadre des systèmes embarqués.

Les processeurs de la famille 80x86 font la distinction entre l'espace réservé à la mémoire centrale et l'espace réservé aux périphériques. En effet, des signaux générés par les pins ($\overline{S_0}\overline{S_1}\overline{S_2}$) du processeur permettent de spécifier si l'adresse est destinée à un transfert vers la mémoire ou une lecture/écriture sur un périphérique [5,8]. Pour ce dernier cas l'adresse est dénommée port d'entrées sorties. L'affectation des adresses aux différents périphériques de la carte de développement, est énumérée dans le tableau suivant :

Tableau-I-2 Affectation des adresses port entrées sorties [9] :

Plage d'adresses en notation hexadécimale	Désignation du port d'entrée / Sotie	Description
00H 07H	Ecran LCD et Clavier	Ecran LCD 00H : Registre instruction. 02H : Registre d'état. 04H : Registre de données. Clavier 01H : Registre en lecture. 01H : Drapeaux (flags).
08H 0FH	8251 USART communication série et 8253 compteurs et Timer	8251 USART 08H : Registre données. 0AH : Instructions, état. 8253 Compteur et Timer 09H : Registre Timer0. 0BH : Registre Timer1. 0DH : Registre Timer2. 0FH : Registre de control.
10H 17H	8259 contrôleur d'interruption Et Hautparleur	8259 contrôleur d'interruption 10H : Registre de control. 12H : Registre de données. Hautparleur 17H : Registre Données
18H 1FH	Interface parallèle 8255A	8255A Circuit CS1 18H : PortA 1AH : PortB 1CH : PortC 1EH : Registre Control 8255A circuit CS2 19H : PortA 1BH : PortB 1DH : PortC 1FH : Registre de control
20H 2FH	Connecteur Utilisateur	Extensions
30H FFH	Non Définies	Adresses libres

III. Les Registres du 8086 :

La programmation en langage machine nécessite la connaissance au préalable des registres internes du processeur utilisé. En effet, en langage assembleur chaque instruction fait référence à une action élémentaire effectuée par le processeur. Il est mentionné pour chaque action qu'une des données manipulées doit obligatoirement se trouver au sein d'un registre du processeur. Nous présentons les différents registres du 8086 chacun dans sa catégorie. Les différents registres sont affichés comme mémorandum sur la carte de développement (*figure-I-3*)[3-9].

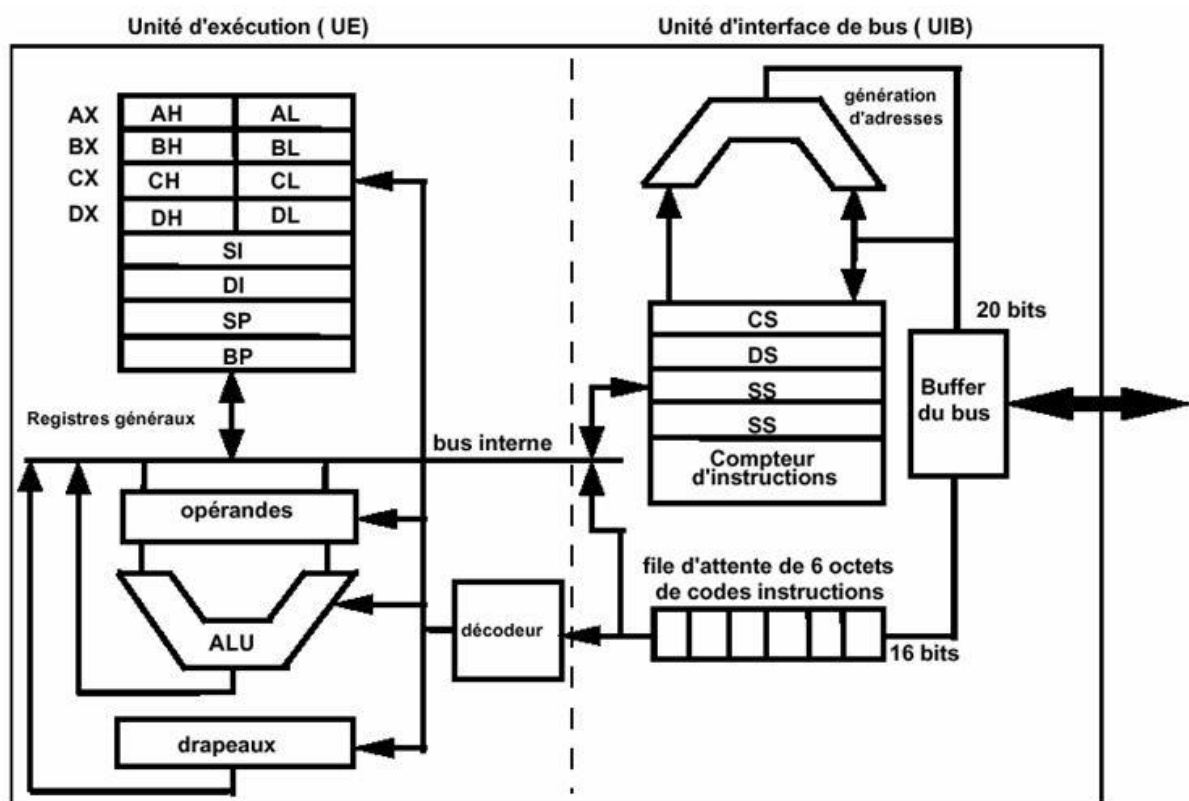


figure-I-3 : Registres du processeur 8086 [3-9].

III.1. Les registres segments

Ces registres 16 bits sont combinés avec un offset pour former les adresses physiques sur 20bits (bus d'adresse du 8086) [3-5]. Chaque case mémoire dans le programme peut être identifiée sous la forme *Segment : Offset*. Les registres segments vont identifier une zone de 64 Ko, l'offset est la position relative par rapport à la position du segment [7,8].

CS : *Code Segment*, segment de code, définit le début de la zone mémoire programme. Les adresses des différentes instructions sont représentées par un offset relatif au registre CS.

DS : *Data Segment*, Segment de données, définit le début de la zone mémoire réservée aux données traitées par le programme.

SS : *Stack Segment*, Segment de pile (l'utilité de ce dernier est expliquée lors des Travaux Pratiques).

La carte de développement utilisée possède un espace mémoire Ram Statique d'une capacité de 64 ko (voir *tableau I-1*), la raison qui nous force à combiner les différents segments Code Données et pile en un seul segment. Un avantage pédagogique puisque dans les machines contemporaines dotées des derniers processeurs Intel, ayants des registres 32 bits et 64 bits, où les différents segments sont combinés et s'apparentent à un seul segment (flat memory model)[8].

III.2. Les registres à usage général

Les registres à usage général participent aux opérations arithmétiques et logiques ainsi qu'à l'adressage. Ces registres sont divisés en deux, les demi-registres sont accessible comme registres de 8 bits.

AX : Accumulateur registre préféré du processeur (AH:AL),

- usage général,
- exigé pour les opérations de multiplication et de division,
- ne peut pas être utilisé pour l'adressage.

BX : Base (BH:BL),

- usage général,
- peut être utilisé pour l'adressage, si son contenu est un offset, il est relatif au segment DS.

CX : Comptage et calcul (CH:CL),

- usage général,
- utilisé par certaines instructions comme compteur,
- ne peut pas être utilisé pour l'adressage.

DX : Data (DH:DL),

- usage général,
- exigé pour les opérations de multiplication et de division comme extension au registre AX,
- ne peut pas être utilisé pour l'adressage.

III.3. Les registres d'adressage offset

Ces registres 16 bits permettent de calculer un offset. Sont utilisés pour la manipulation des tableaux par exemple.

SP : Stack Pointer, Pointeur de Pile,

- utilisé pour accéder à la pile,
- par défaut il est relatif au registre SS.

BP : Base Pointer, Pointeur de Base,

- utilisé comme Base,
- par défaut il est relatif au registre SS,
- usage général.

SI : Source Index,

- utilisé comme indexe,
- par défaut il est relatif au registre DS
- usage général.

DI : Destination Index,

- utilisé comme indexe,
- par défaut il est relatif au registre DS
- usage général.

IV. Travail Pratique 01 Initiation (Durée une séance de 3 heures)

IV.1. Questions

Au sein du processeur 8086 :

1. Quel est le rôle du registre IP ?
2. Quel sont les composants du registre d'état ?
3. CF est égal à 1, expliquez ?
4. ZF est égal à 1, expliquez ?

IV.2. Travail demandé

En utilisant l'utilitaire **Debug** du kit de développement.

1. Donner la valeur des adresses du tableau suivant puis modifier leurs contenues ? de quel segment s'agit-il ?

Adresse	Valeur Hexa	Caractère ASCII	Nouvelle valeur
154E :100			A
154E :103			B
17A0 :105			A
1730 :250			25

2. Soit les instructions suivantes :

<i>Instruction</i>	<i>Définition</i>
<i>mov destination, valeur</i>	initialisation de la destination par la valeur.
<i>add destination, valeur</i>	destination = destination + valeur.
<i>Int 3</i>	fin du programme.

- a. Écrire sous Debug, un programme assembleur qui permet d'additionner les nombres 150 et 360.
- b. Préciser le format complet de l'instruction en HEXA.
- c. Relever les adresses du programme.
- d. Introduire les formats dans la carte de développement en utilisant le clavier hexadécimal.
- e. Autres Applications : $3 + 5$; $255 + 8$.

3. Soit les instructions suivantes :

<i>Instruction</i>	<i>Définition</i>
<i>mov destination, [offset]</i>	initialisation de la destination par le contenu de la case mémoire <i>DS : offset</i> .
<i>add destination, [offset]</i>	destination = destination + contenu de la case mémoire <i>DS : offset</i> .
<i>Int 3</i>	fin du programme.

- Écrire un programme assembleur qui permet de permuter les données enregistrées respectivement aux adresses 130h et 132h.
- Préciser le format complet de l'instruction en HEXA (COP ...) ?
- Relever les adresses du programme ?
- Application : 127 + 80 ; 200 + 88 dans des adresses de votre choix ?

NB : les résultats doivent être formulés sous la forme.

Adresse	Instruction en mnémonique	Code opératoire

V. Solution Travail Pratique 01 Initiation

V.1. Questions

1. Le registre IP Instruction Pointer est un registre du processeur qui désigne la prochaine instruction à exécuter. Ce n'est qu'un simple compteur d'instructions la raison pour laquelle il est dit *Compteur Ordinal*.
2. Le registre d'état est constitué des bits suivants (*figure-I-4*):

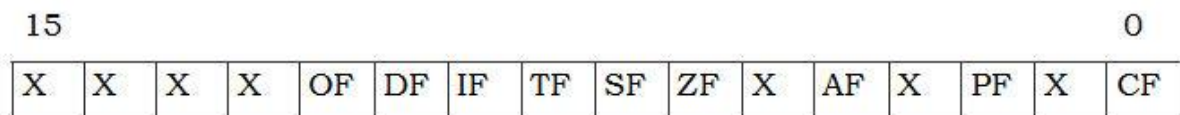


figure-I-4 : Composants du registre des états du processeur 8086 [3-9].

3. CF= 1 car l'instruction précédente a généré une retenue.
4. ZF=1 car l'instruction précédente a généré un résultat nul.

V.2. Travail demandé

1. Sous Debug la commande D permet de visualiser le contenu de la mémoire en l'assimilant à des données, le contenu varie en fonction de l'état de chaque Machine. La commande E Enter permet de modifier le contenu de la RAM exemple (*figure-I-5*).

```

-E 154E:100 'BENADDA Belkacem et BELDJILALI Bilal'
-D 154E:100
154E:0100  42 45 4E 41 44 44 41 20-42 65 6C 6B 61 63 65 6D  BENADDA Belkacem
154E:0110  20 65 74 20 42 45 4C 44-4A 49 4C 41 4C 49 20 42  et BELDJILALI B
154E:0120  69 6C 61 6C 00 00 00 00-00 00 00 00 00 00 00 00  ilal.....
154E:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
154E:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
154E:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
154E:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
154E:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
-E 17A0:100 'Micro programmation'
-d 17A0:105
17A0:0100  20 70 72-6F 67 72 61 6D 6D 61 74  ..... programmat
17A0:0110  69 6F 6E 00 00 00 00-00 00 00 00 00 00 00 00  ion.....
17A0:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
17A0:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
17A0:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
17A0:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
17A0:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
17A0:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
17A0:0180  00 00 00 00 00  .....

```

figure-I-5 : Fonctionnement de la commande Dump Memory D et Enter E.

Sur la *figure-I-5* trois zones sont visibles. Une zone des adresses *segment : offset*, une zone qui contient des adresses en représentation hexadécimale et une dernière zone montrant le contenu des cases mémoires en représentation ASCII. Il s'agit du segment des données.

2. Sous Debug la commande A assemble permet d'introduire directement les instructions machine dans la mémoire. Alors que la commande U Unassemble permet de décoder le contenu de la mémoire en le considérant comme étant des instructions et la commande G Go permet l'exécution d'un programme. Un exemple d'exécutions est montré sur la (*figure-I-6*).

```

-a
0B91:0100 MOV AX, 150
0B91:0103 ADD AX, 360
0B91:0106 INT 3
0B91:0107
-u 100,106
0B91:0100 B85001      MOV     AX,0150
0B91:0103 056003      ADD     AX,0360
0B91:0106 CC          INT     3
-a
0B91:0107 mov al, 3
0B91:0109 add al, 5
0B91:010B int 3
0B91:010C
-u 107,10b
0B91:0107 B003      MOV     AL,03
0B91:0109 0405      ADD     AL,05
0B91:010B CC          INT     3
-g=100

AX=04B0  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B91  ES=0B91  SS=0B91  CS=0B91  IP=0106  NU UP EI PL NZ NA PO NC
0B91:0106 CC          INT     3

```

figure-I-6 : Insertion, décodage et exécution des programmes directement en mémoire vive avec adressage immédiat.

Sur la *figure-I-6* pour la commande U trois zones sont visibles. Une zone des adresses *segment : offset*, une zone qui contient le code binaire des instructions en représentation hexadécimale compréhensible par le CPU et une dernière zone montrant les mnémoniques lisibles par un humain. La partie du bas montre le résultat d'exécution du premier programme (résultat dans AX), il faut remarquer l'état des flags.

L'adressage immédiat fait référence aux données contenues dans les instructions.

3. Sous Debug écrire de programmes accédant à des opérandes enregistrés dans la mémoire (*figure-I-7*). Dans ce cas il s'agit d'un adressage direct où les adresses sont clairement révélées dans les instructions.

```

-d 130,132
0B91:0130 03 75 3F                                     .u?
-a
0B91:0100 mov al, [130]
0B91:0103 add al, [132]
0B91:0107 mov [131], al
0B91:010A int 3
0B91:010B
-u 100,10a
0B91:0100 A03001      MOV     AL,[0130]
0B91:0103 02063201    ADD     AL,[0132]
0B91:0107 A23101      MOV     [0131],AL
0B91:010A CC          INT     3
-g=100

AX=0042 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B91 ES=0B91 SS=0B91 CS=0B91 IP=010A  NU UP EI PL NZ AC PE NC
0B91:010A CC          INT     3
-d 130,132
0B91:0130 03 42 3F                                     .B?
-

```

figure-I-7 : Insertion, décodage et exécution des programmes directement en mémoire vive avec adressage direct.

VI. Conclusion

La connaissance de l'architecture interne du CPU est primordiale pour sa programmation. Il est également démontré que l'écriture des instructions directement sur la mémoire centrale de la machine en binaire ou en mnémonique est une tâche ardue. Il est également évident que la résolution d'un algorithme ou traitement mathématique en langage machine s'avère relativement délicat. Les solutions proposées sur les figures sont une simple illustration d'échantillons d'une multitude de possibilités offertes aux étudiants. Des solutions types qui aideront sans aucun doute à proposer d'autres solutions.

Chapitre 2

Manipulations avec l'interface parallèle

I. Introduction

L'enjeu primordial de la programmation assembleur est de pouvoir manipuler directement des périphériques en assimilant la façon avec laquelle ils opèrent. Les périphériques les plus simples sont ceux qui échangent avec le processeur des données en parallèles. En effet ces derniers opèrent à l'image d'une simple case mémoire. Cependant, écrire des programmes avancés en assembleur requiert certaines règles imposées par les outils de programmation et de développement. Dans ce chapitre nous allons présenter les structures des programmes assembleurs. Des structures qui seront exploitées pour la manipulation des interfaces parallèles.

II. Format des instructions et programmes

II.1. Les instructions

En langage machine chaque instruction définit une opération élémentaire exécutée par le processeur, chaque instruction est codée par un nombre entier d'octets. Cependant, l'espace que va occuper une instruction en mémoire dépend également des opérandes. En effet en se référant aux *figures-I-6 et I-7* il est facile de remarquer que les instructions en représentation binaires sont divisées en deux parties comme indiqué sur le *Tableau-II-1* :

- ✓ le code opératoire qui caractérise l'instruction, montré avec la couleur rouge dans le tableau,
- ✓ l'opérande avec une mémorisation *little indian*, montrée avec la couleur bleu.

Il devient alors évident que les instructions codées en binaire sont sous la forme de la *figure-II-1*.

Pour la programmation en langage machine il est recommandé d'utiliser l'écriture en mnémonique (*figure-II-2*). En effet, l'introduction des instructions en binaires dans le système, comme réalisée dans le *Travail pratique N°1*, est très ardue et fait l'objet d'innombrables erreurs (saisies, emplacement mémoire, ...).

Tableau-II-1 Exemple représentation des instructions en mémoire

Format binaire		Instruction en notation Mnémonique
Code opératoire	Opérande	
B0	03	MOV AL, 03
04	05	ADD AL, 05
CC		INT 3
B8	5001	MOV AX, 0150
05	6003	ADD AX, 0360
A0	3001	MOV AL, [0130]
0206	3201	ADD AL,[0132]
A2	3101	MOV[0131], AL

Code opératoire	Opérande
-----------------	----------

figure-II-1 : Format binaire de l'instruction.

{étiquette:}	mnémonique	{opérande destination	{, opérande source}}	{ ; commentaire}
--------------	------------	-----------------------	----------------------	------------------

figure-II-2 : Ecriture de l'instruction en mnémonique.

Les parties entre accolade (en couleur rouge),
ne sont pas présentes dans certaines instructions.

Dans le cas d'une représentation en mnémonique, l'une des opérandes source ou destination doit faire référence à un registre du processeur 8086. Il faut également noté que les opérandes fixent le mode d'adressage utilisé. Dans la *figure-I-6* le mode immédiat est utilisé puisque l'opérande fait référence à la donnée elle-même. Etant donné que le programme de la *figure-I-7* fait usage de l'adresse en mémoire de la donnée il s'agit du mode direct.

II.2. Le format des programmes rédigés en assembleur

L'écriture des programmes est réalisée avec un outil fourni avec les cartes de développement *figure-II-3*. L'outil utilise MASM de Microsoft comme assembleur et Debugger. Les programmes assemblés par MASM sous la forme de fichiers objet (représentation binaire des instructions), sont convertis en fichiers *ABS* et transférés sur la mémoire vive de la carte de développement via une interface série RS232.

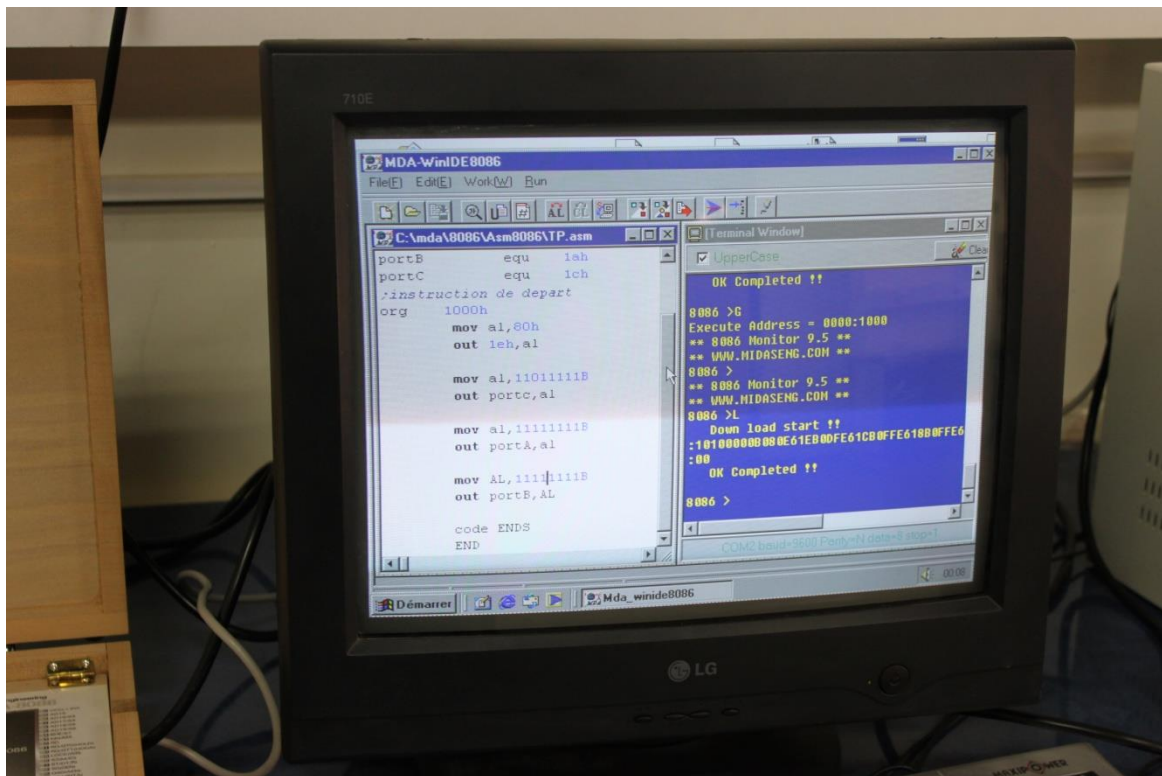


figure-II-3 : environnement de développement utilisé.

La partie à gauche est destinée à l'écriture des programmes, et la partie à droite est l'outil Debug associé avec la carte de développement.

Nous avons expliqué préalablement (*Chapitre 1*) que l'espace mémoire est de 64 ko, la limite imposée pour un segment. Ceci nous pousse à combiner sur la taille d'un seul segment les trois types : Code, Données et pile, l'ensemble ne peut dépasser la capacité de 64ko, ce qui oblige la définition d'un seul segment au niveau de l'assembleur (*figure-II-4*).

```

,*****
;
;   TP N°-- du --/--/----
;   Titres du Travail Pratique, Exercice N°--
;   Matière systemes embarques
,*****
; Début de définition du segment dit BK
BK                               SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK
    ;
    ORG 1000H
    ;
    ; Instructions
    ;
BK  ENDS ; Fin du segment BK
    END

```

figure-II-4 : Structure d'un programme assembleur utilisé.

Les programmes doivent respecter la structure représentée dans *figure-II-4* :

- ✓ ASSUME CS:BK,DS:BK,SS:BK: informe l'assembleur que les registres CS, DS et SS pointent à l'adresse du segment créé derrière l'identificateur BK.
- ✓ ORG 1000H : Une directive destinée à l'assembleur qui indique l'adresse mémoire de départ pour la séquence de code qui suit. Dans notre cas la première instruction exécutée. Cette valeur de l'adresse est imposée par le Firmware de la carte de développement.

III. Présentation de l'interface parallèle

L'interface parallèle intervient dans le cadre d'un échange simultané de plusieurs bits, un octet en général. Ce type d'interface présente l'avantage de la rapidité des transferts et facilité de la programmation. Chaque interface parallèle avant son utilisation doit être configurée en entrée, en sortie, bidirectionnelle, avec ou sans signaux de contrôle. Dans la carte de développement que nous utilisons, l'interface parallèle est matérialisée par le circuit 8255A (*figure-II-5*). Ce dernier possède trois ports de un octet chacun :

PortA, PortB et PortC. Le mode de fonctionnement de l'interface parallèle est assuré via un registre de commande (figure-II-6).

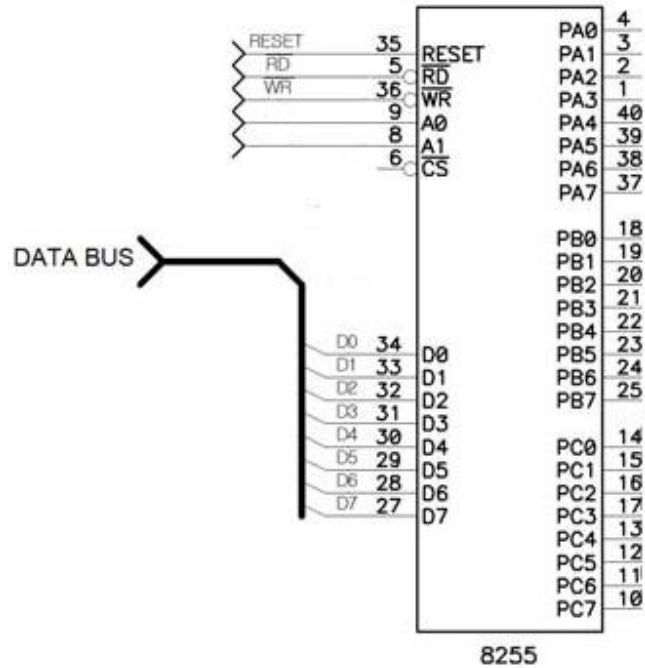


figure-II-5 : Circuit de l'interface parallèle 8255.

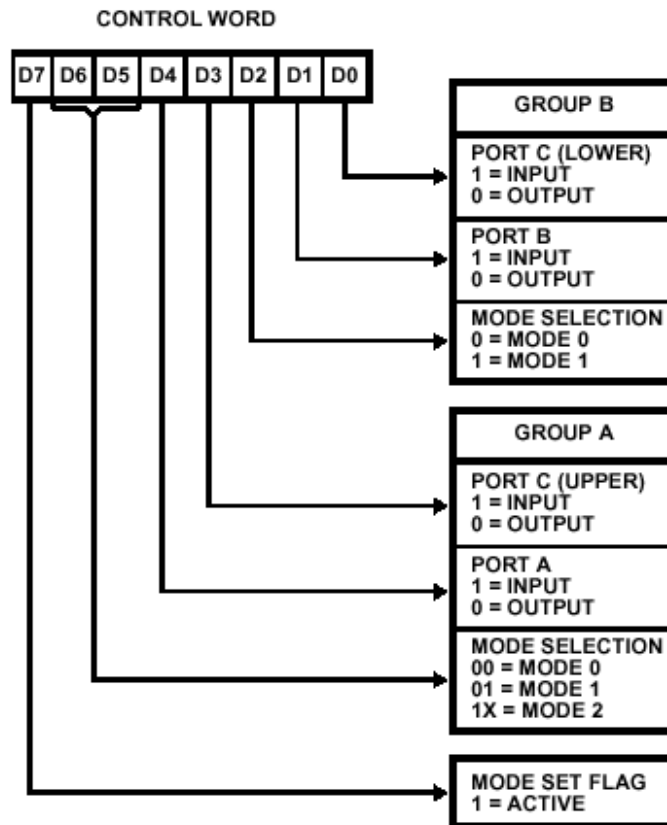


figure-II-6 : Configuration du 8255.

IV. Travail Pratique 02 (Durée 6 heures)

Manipulations sur l'afficheur 7 segments et LEDs

IV.1. Description du dispositif :

Soit l'interface parallèle représentée avec le circuit ci-dessous (figure-II-7).

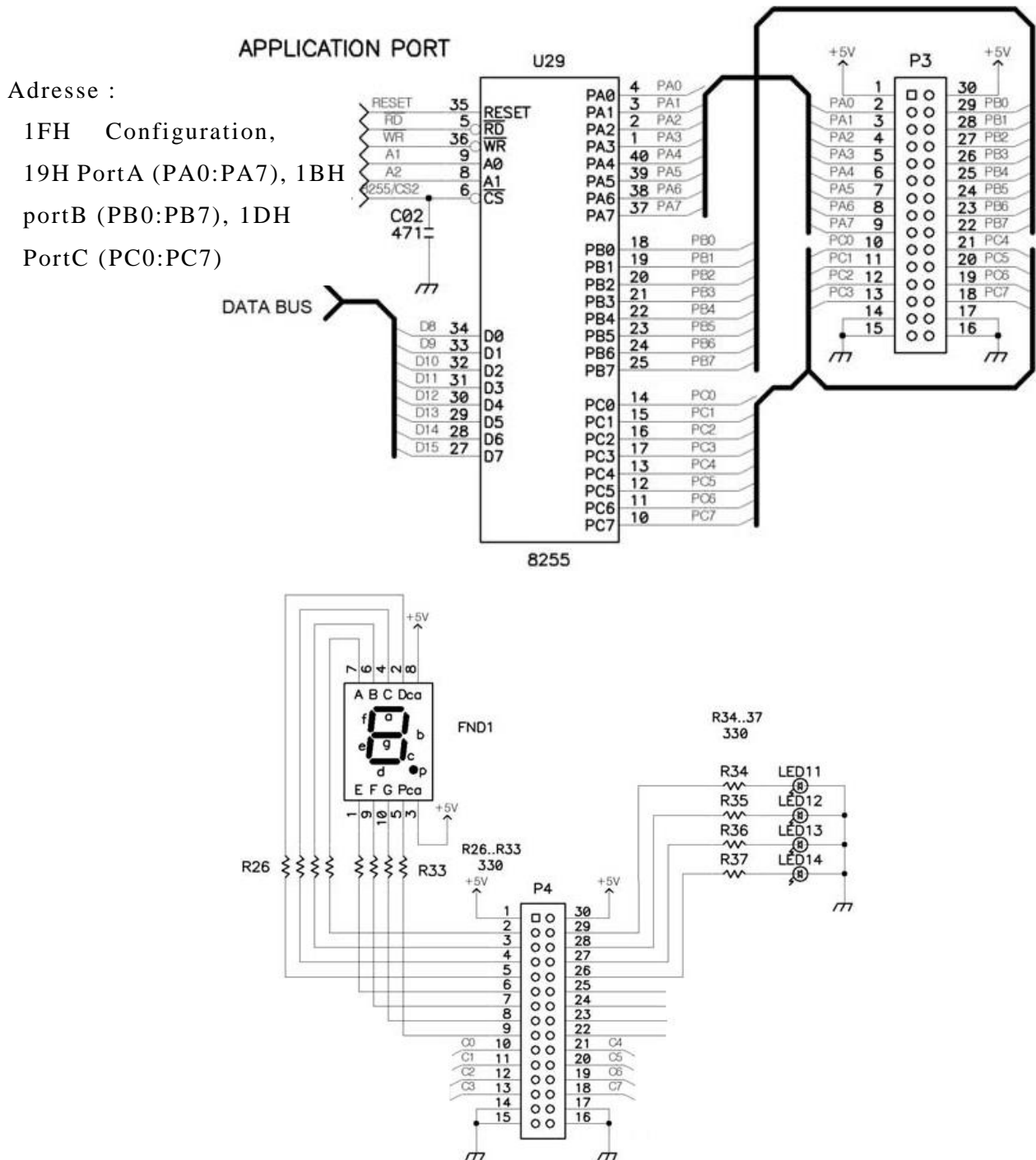
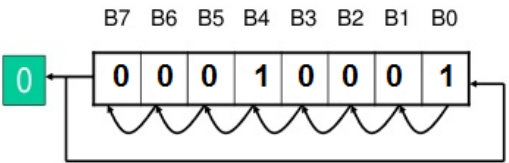


figure-II-7 : Schématique Afficheur 7 segment et 4 LEDs
interfacés avec le 8255A-CS2. P4=P3.

Les instructions à utiliser pour ce travail sont énumérées sur le *Tableau-II-2*.

Tableau-II-2 instructions proposées pour le travail demandé.

<u>Mnémonique</u>	<u>Signification</u>	<u>Nombre de cycles</u>
OUT Adresse, AL	Ecrire le contenu du registre AL vers l'interface spécifiée par l'Adresse.	14
IN AL, Adresse	Lire dans le registre AL le contenu de l'interface spécifiée par l'Adresse.	14
MOV Registre, Valeur	Copier la Valeur dans le Registre (adressage immédiat)	04
MOV Registre1, Registre2	Copier le Registre2 dans le Registre 1	03
ROL Registre		02
NOP	No Opération	03
CMP Registre, Valeur	Effectue (Registre – Valeur), le résultat est perdu et actualise les flags : AF CF OF PF SF ZF	04
XOR registre1, registre2	XOR : registre1= registre1 \oplus registre2	04
DEC Registre	Registre = Registre -1	03
JNZ étiquette	IP= étiquette si ZF = 0	04
JNE étiquette		16 cas du jump
JLE étiquette	IP = étiquette si CF=1 ou ZF =1	04
JBE étiquette		16 cas du jump
JMP étiquette	IP = étiquette.	11
{id_Adr} DB valeur	Une directive qui définit un octet dans la mémoire est l'initialise avec valeur. L'adresse de cette case est accessible via <i>id_Adr</i> .	/
Identificateur EQU valeur	Déclaration de constante, une directive pour l'assembleur.	/

IV.2. Travail demandé:

1. Ecrire un programme qui permet d'allumer La LED verte et affiche 4 sur l'afficher 7 segments.
2. Ecrire un programme qui permet d'allumer Les LEDs rouges et affiche 9 sur l'afficher 7 segments.
3. Ecrire un programme qui affiche un compteur modulo 10 sur l'afficheur 7 segment, la transition est effectuée chaque seconde. Une des quatre LEDs doit s'allumer après chaque séquence complète du compteur.

V. Solution Travail Pratique 02 :

Manipulations sur l'afficheur 7 segments et LEDs

V.1. Analyse du problème

L'afficheur 7 segment utilisé, représenté par la figure ci dessous (*figure-II-8*) est un afficheur à anode commune [10]. La commande des différentes LEDs est assurée par la cathode de chacune via le PortA: un 0 allume un 1 éteint.

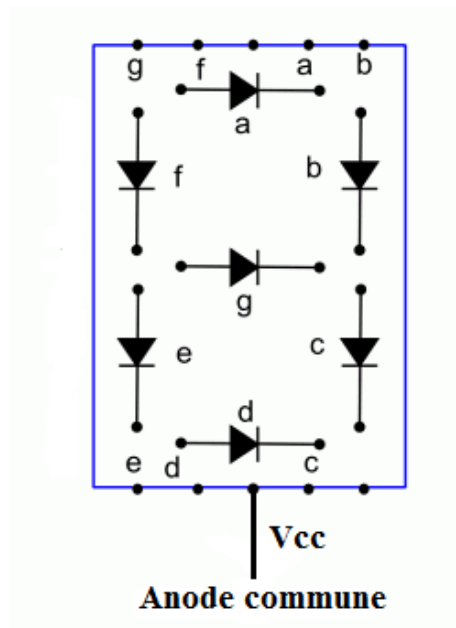


figure-II-8 : Afficheur 7 segment à anode commune.

Les LEDs sont placées sur la carte :

- ✓ LED11 : Rouge, anode connectée avec PB0,
- ✓ LED12 : Verte, anode connectée avec PB1,
- ✓ LED13 : Jaune, anode connectée avec PB2,
- ✓ LED14 : Rouge, anode connectée avec PB3.

Pour finir les PortA et PortB sont utilisés en sortie, ce qui impose leurs configuration en sortie en écrivant la valeur 80H sur le registre de commande du 8255A-CS2, une valeur calculée à partir de la *figure-II-6*.

V.2. Solutions

1. Allumer la LED verte et afficher 4.

Allumer la LED Verte revient à écrire le motif binaire 00000010 sur le PortB. Afficher 4 revient à écrire le motif binaire 10011001 sur le PortA. La solution est fournie ci-dessous (*figure-II-9*). Le résultat de l'exécution est représenté par la *figure-II-10*.

```

,*****
;
;   TP N°02 du 22 /11/2015
;   Affichage 7 segments, Exercice N°01
;   Matière systemes embarques
,*****
; Début de définition du segment dit BK
BK                                SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK
    ;
    ORG 1000H
    ; Configuration PortA, PortB et PortC Mode 0 En sortie
    MOV AL, 80H
    OUT 1FH, AL
    ; Allumer la LED Verte
    MOV AL, 00000010B
    OUT 1BH, AL
    ; Afficher 4
    MOV AL, 10011001B
    OUT 19H, AL
    INT 3
BK    ENDS ; Fin du segment BK
    END

```

figure-II-9 : Programme type pour allumer la LED verte et afficher 4 sur le 7 segments

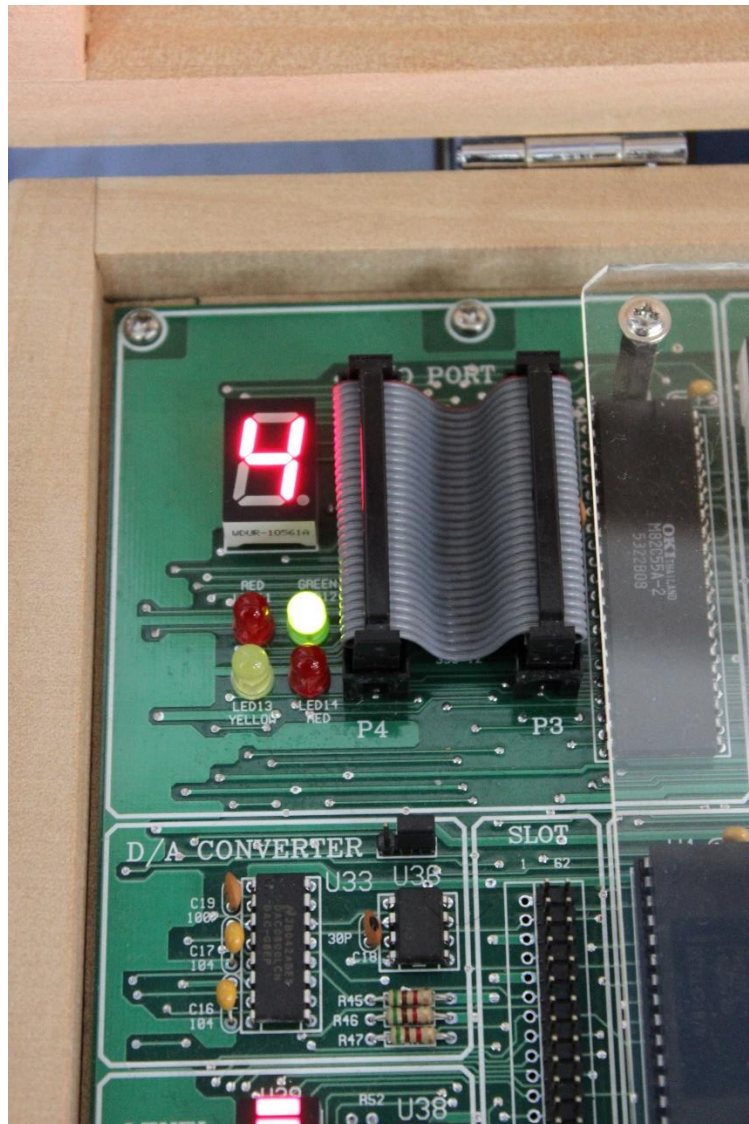


figure-II-10 : Résultat d'exécution du programme (Figure-II-9).

2. Allumer les LEDs rouges et afficher 9.

Allumer les LEDs rouges revient à écrire le motif binaire *0000 1001* sur le PortB. Afficher 9 revient à écrire le motif binaire *1010 0000* sur le PortA. La solution est fournie ci-dessous (*figure-II-11*).

Dans cette solution, la directive EQU est utilisée pour remplacer les adresses physiques des ports par des noms symboliques, ceci va faciliter la lecture et la compréhension du programme.

```

,*****
;
;   TP N°02 du 22 /11/2015
;   Affichage 7 segments, Exercice N°02
;   Matière systemes embarques
,*****
; Début de définition du segment dit BEN
BEN                                SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK

    Config    EQU    1FH
    PortA     EQU    19H
    PortB     EQU    1BH

    ORG 1000H
    ; Configuration PortA, PortB et PortC Mode 0 En sortie
    MOV AL, 80H
    OUT Config, AL
    ; Allumer les LEDs rouges
    MOV AL, 00000010B
    OUT PortB, AL
    ; Afficher 9
    MOV AL, 10100100B
    OUT PortA, AL
    INT 3
BEN ENDS ; Fin du segment BEN
END

```

figure-II-11 : Programme type pour allumer les LEDsrouges et afficher 9 sur le 7 segments.

3. Compteur et horloge.

La solution est donnée par la *figure-II-12*. Les différentes combinaisons à utiliser avec l'afficheur sept segments sont énumérées à la fin du programme. L'adresse de la première combinaison (le zéro) est signalée par l'étiquette *chiffre* utilisée comme base permettant l'accès aux suivantes, l'adresse de la combinaison suivante à afficher est calculée en utilisant le registre *BX* pour mémoriser le déplacement par rapport à l'adresse de base *chiffre*. Le processeur opère avec une vitesse très

rapide, une temporisation est insérée pour ralentir l'affichage à une période calculée via les cycles horloge des instructions.

```

;*****
;
; TP N°02 du 22 /11/2015
; Affichage 7 segments, Exercice N°03
; Matière systemes embarques
;*****
COMPT                                SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK
        Config    EQU        1FH
        PortA     EQU        19H
        PortB     EQU        1BH
    ORG 1000H
; Configuration PortA, PortB et PortC Mode 0 En sortie
        MOV AL, 80H
        OUT Config, AL
; Allumer la première LED
        MOV AH, 10001000B
Debut :   ROL AH, 1
        MOV AL, AH
        OUT PortB, AL
; Afficher 2
        XOR BX, BX
Comp:    MOV AL, [Chiffre+BX]
        OUT PortA, AL
        INC BX
        XOR CX, CX
Tempo:   NOP
        NOP
        NOP
        DEC CX
        JNZ Tempo
        CMP BX, 09H
        JLE  Comp
        JMP Debut
        INT 3

```


Chiffre	DB	11000000B
	DB	11111001B
	DB	10100100B
	DB	10110000B
	DB	10011001B
	DB	10010010B
	DB	10000010B
	DB	11111000B
	DB	10000000B
	DB	10010000B
COMPT	ENDS	; Fin du segment COMPT
	END	

figure-II-12 : Compteur et minuterie

VI. Travail Pratique N°3 (durée 6 heures)

Panneau d'affichage LED

VL1. Description du dispositif

Le dispositif d'affichage à base de LEDs que nous voulons contrôler est représenté par la *figure-II-13* :

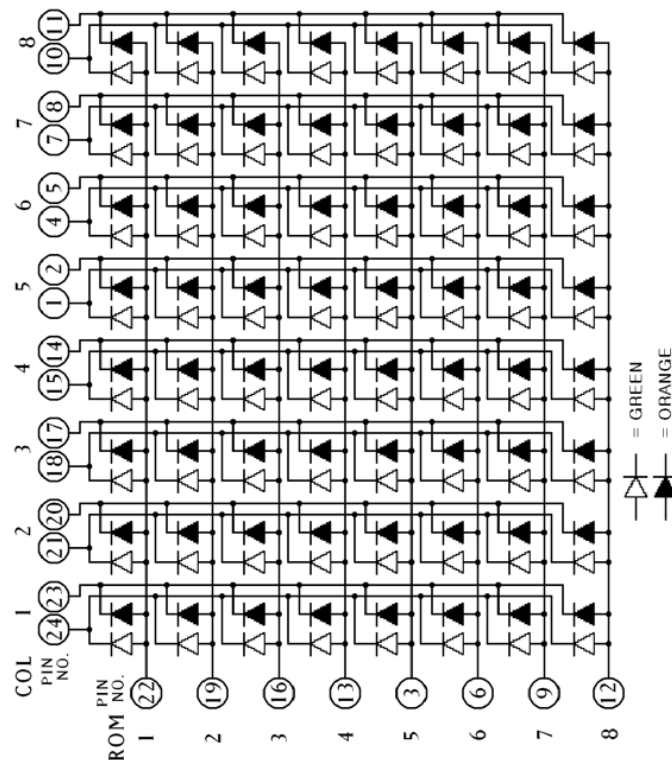
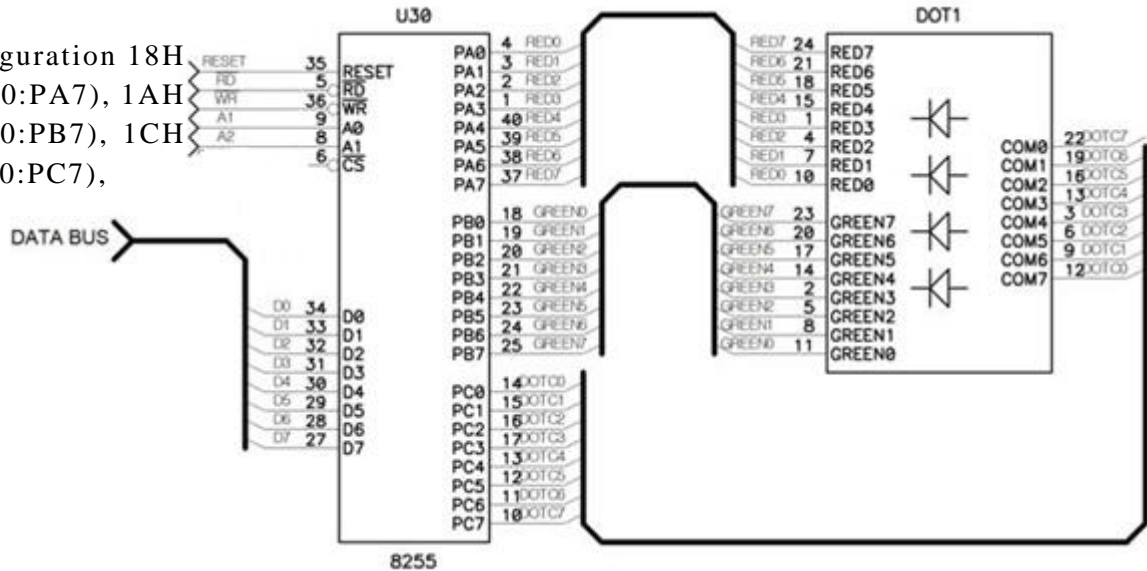
Adresses :

1EH configuration 18H

PortA (PA0:PA7), 1AH

portB (PB0:PB7), 1CH

PortC (PC0:PC7),



*figure-II-13 : Schématique panneau d'affichage 8x8 LEDs
interfacés avec le 8255A-CS1.*

Les nouvelles instructions à utiliser pour ce travail sont énumérées sur le *Tableau-II-3*.

Tableau-II-3 instructions proposées pour le travail demandé.

<u>Mnémonique</u>	<u>Signification</u>	<u>Nombre de cycles</u>
LOOP offset	C'est une instruction de boucle utilisant CX comme compteur décrémenté à chaque exécution de Loop, si ZF=1 alors IP= offset.	04 08 dernière itération
CALL offset	Le registres IP est empilé dans la pile, IP = offset	19
RET	IP est dépilé de la pile	16

VI.2. Travail demandé

1. écrire un programme assembleur qui affiche toute la matrice en vert.
2. écrire un programme assembleur qui affiche la deuxième ligne en vert et la cinquième ligne de la matrice en rouge.
3. Ecrire un programme assembleur qui fait défiler les colonnes une à une en orange.
4. Ecrire un programme qui affiche la lettre A.
5. Ecrire un programme qui dessine un triangle.
6. Ecrire un programme qui fait défiler de gauche vers la droite le Texte RST. Chaque défilement sera effectué en une couleur différente : Vert, Rouge ensuite Orange.

VII. Solution Travail Pratique N°3 Panneau d'affichage LED1. Afficher la matrice en vert (*figure-II-14*)

```

*****
;
;   TP N°03 du 22 /11/2015
;   Affichage Matrice de LEDs, Exercice N°01
;   Matière systemes embarques
*****
; Début de définition du segment dit BK
BK                                     SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK
        Config    EQU    1EH
        PortA     EQU    18H
        PortB     EQU    1AH
        PortC     EQU    1CH

    ORG 1000H
; Configuration PortA, PortB et PortC Mode 0 En sortie
    MOV AL, 80H
    OUT Config, AL
; Activer toutes les colonnes
    MOV AL, 11111111B
    OUT PortC, AL
; Activer toutes les lignes vertes
    MOV AL, 00000000B
    OUT PortA, AL
; Désactiver toutes les lignes rouges
    MOV AL, FFH
    OUT PortB, AL

    INT 3
BK    ENDS ; Fin du segment BK
    END

```

figure-II-14 : Programme type pour allumer le panneau en vert.

2. Programme type pour allumer la deuxième ligne en vert et la cinquième colonne en rouge (*figure-II-15*) le résultat de l'exécution est montré par la (*figure-II-16*).

```

,*****
;
;   TP N°03 du 22 /11/2015
;   Affichage Matrice de LEDs, Exercice N°02
;   Matière systemes embarques
,*****
; Début de définition du segment dit BK
BK                                SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK
        Config    EQU    1EH
        PortA     EQU    18H
        PortB     EQU    1AH
        PortC     EQU    1CH

    ORG 1000H
    ; Configuration PortA, PortB et PortC Mode 0 En sortie
    MOV AL, 80H
    OUT Config, AL
    ; Activer toutes les colonnes
    MOV AL, 11111111B
    OUT PortC, AL
    ; Activer la deuxième ligne verte
    MOV AL, 11111101B
    OUT PortA, AL
    ; Activer la cinquième ligne rouge
    MOV AL, 11101111B
    OUT PortB, AL

    INT 3
BK  ENDS ; Fin du segment BK
    END

```

figure-II-15 : Programme type pour allumer la deuxième ligne en vert et la cinquième ligne en rouge.

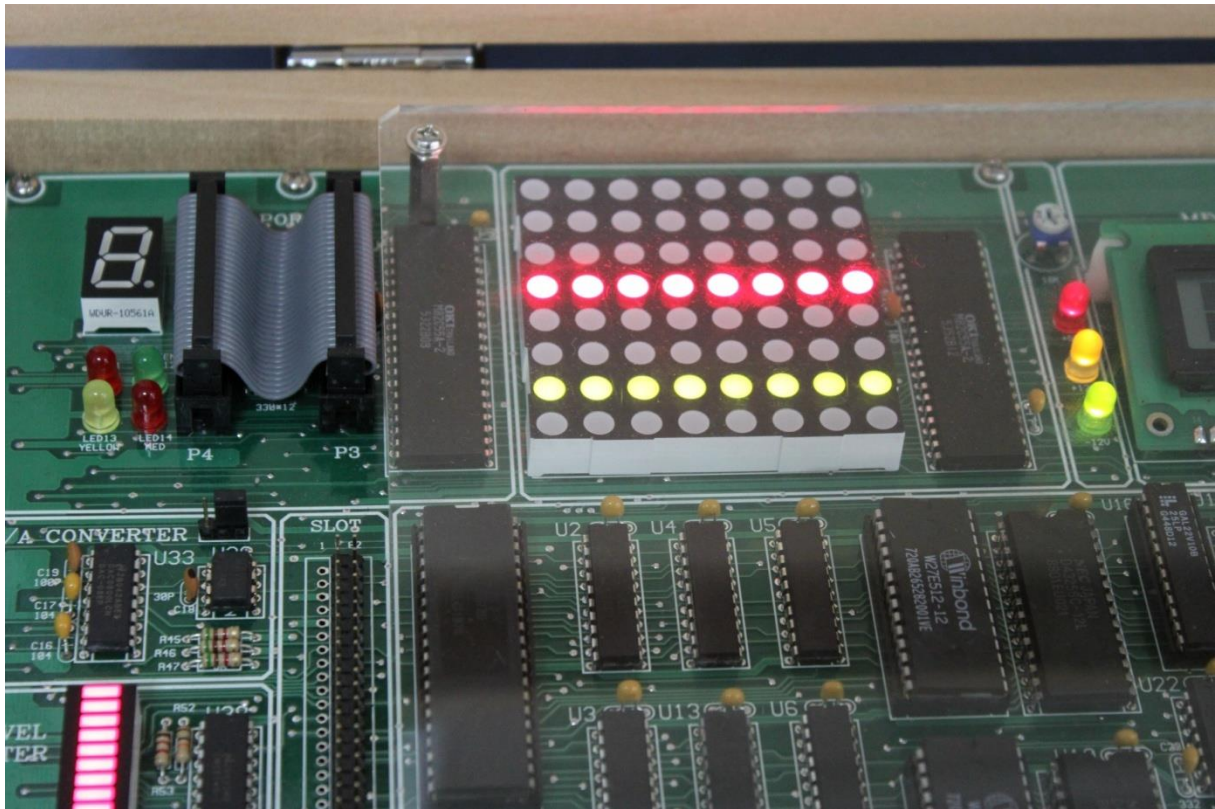


figure-II-16 : Exécutions Programme pour allumer la deuxième ligne en vert et la cinquième ligne en rouge.

3. défiler les colonnes une à une en orange (figure-II-17).

```

*****
;
;   TP N°03 du 22 /11/2015
;   Affichage Matrice de LEDs, Exercice N°03
;   Matière systemes embarques
*****
; Début de définition du segment dit BK
BK                               SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK
        Config    EQU    1EH
        PortA     EQU    18H
        PortB     EQU    1AH
        PortC     EQU    1CH

    ORG 1000H
    
```

```

; Configuration PortA, PortB et PortC Mode 0 En sortie
MOV AL, 80H
OUT Config, AL
; Activer les lignes vertes
XOR AL, AL
OUT PortA, AL
; Activer les lignes rouges
MOV AL, 00H
OUT PortB, AL
; Activer les colonnes une à une
MOV AL, 00000001B
Rep : OUT PortC, AL
      CALL TEMPO
      ROL AL, 1
      JMP Rep

      INT 3

TEMPO: MOV CX,0FFFFH
T1:    NOP
      NOP
      NOP
      NOP
      LOOPT1
      RET
BK    ENDS ; Fin du segment BK
      END

```

figure-II-17 : Programme type pour allumer la deuxième ligne en vert et la cinquième colonne en rouge.

4. Afficher la lettre A (*figure-II-18*) le résultat de l'exécution est montrée par la (*figure-II-19*).

```

,*****
;
;   TP N°03 du 22 /11/2015
;   Affichage Matrice de LEDs, Exercice N°04
;   Matière systemes embarques
,*****
; Début de définition du segment dit BK
BK                                     SEGMENT
    ASSUME    CS:BK,DS:BK,SS:BK
        Config    EQU    1EH
        PortA     EQU    18H
        PortB     EQU    1AH
        PortC     EQU    1CH
    ORG 1000H
        ; Configuration PortA, PortB et PortC Mode 0 En sortie
        MOV AL, 80H
        OUT Config, AL

        ; Désactiver les lignes vertes
        MOV AL, 11111111B
        OUT PortA, AL
        ; Dessiner sur les lignes rouges
Debut :   XOR BX, BX
          MOV AH, 00000001B
ET2:     MOV AL, [A+BX]
          OUT PortB, AL

          MOV AL, AH
          OUT PortC, AL
          CALL TEMPO
          INC BX
          ROL AH, 1
          JNC ET2
          JMP Debut
          INT 3

```



```
TEMPO:  MOV  CX, 300H
T1:      NOP
        NOP
        NOP
        NOP
        LOOPT1
        RET

A:       DB   11111111B
        DB   11000000B
        DB   10110111B
        DB   01110111B
        DB   01110111B
        DB   10110111B
        DB   11000000B
        DB   11111111B

BK  ENDS ; Fin du segment BK
END
```

figure-II-18 : Programme type pour afficher la lettre A.

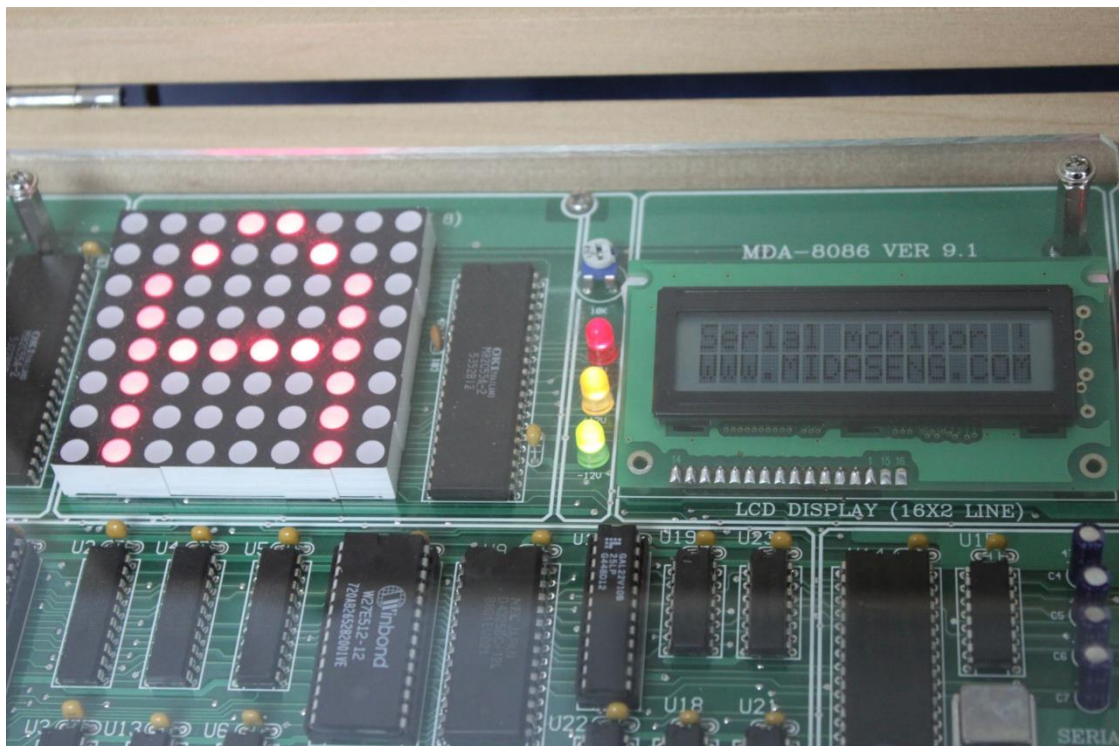


figure-II-19 : Exécution programme type pour afficher la lettre A.

VIII. Travail Pratique N°4 (durée 3 heures)

Manipulation avec le clavier

VIII.1. Description du dispositif

Le dispositif à clavier que nous voulons manipuler est représenté sur la figure-II-20.

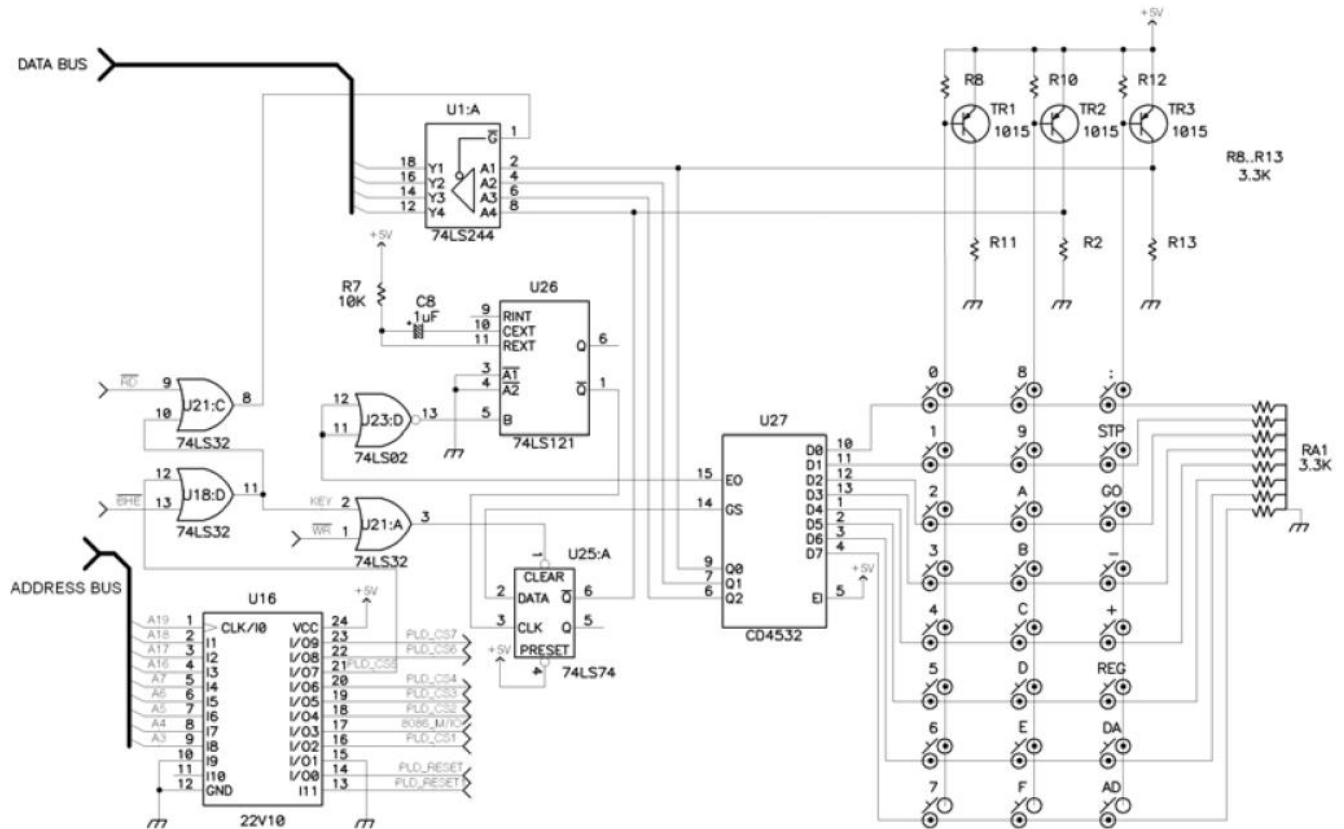


figure-II-20 : Schématique du clavier.

Les touches du clavier possèdent les codes si dessous :

Key	0	1	2	3	4	5	6	7
Code	00	01	02	03	04	05	06	07
Key	8	9	A	B	C	D	E	F
Code	08	09	0A	0B	0C	0D	0E	0F
Key	:	STP	GO	REG	-	+	DA	AD
Code	10	11	12	13	14	15	16	17

L'écran LCD est un module standard de deux lignes seize caractères par ligne, qui répond aux instructions du tableau suivant :

Tableau-II-4 instructions de l'écran LCD.

Instruction	CODE										Description	Execution time(max) fosc is 250 KHz	
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display	1.64 μ s	
Return Home	0	0	0	0	0	0	0	0	0	1	* Returns display being shifted to original position	1.64 μ s	
Entry Mode set	0	0	0	0	0	0	0	0	1	I/D	S Sets cursor move direction and specifies shift of display	40 μ s	
Display ON/OFF Control	0	0	0	0	0	0	0	1	D	C	B D : Display ON/OFF C : Cursor ON/OFF B : Cursor Blink/Not	40 μ s	
Cursor or Display Shift	0	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor and Shifts display	40 μ s
Function Set	0	0	0	0	0	1	DL	N	F	*	*	Refer to Remark	40 μ s
Set CGRAM	0	0	0	0	1	ACG					Sets CG RAM Addr.	40 μ s	
Set DD RAM Addr.	0	0	0	1	ADD					Sets DD RAM Address	40 μ s		
Read Busy Flag & Addr	0	0	1	BF	AC					BF : Busy flag Reads AC contents.	40 μ s		
Write Data CG or DD	1	0	Write data					Writes data into DD RAM or CG RAM				40 μ s	
Read Data from CG or DD RAM	1	1	Read data					Reads data from DD RAM or CG RAM				40 μ s	
Remark	I/D= 1: Increment 0: Decrement										DD RAM : Display data RAM		
	S= 1: Accompanies display shift										CG RAM : Character generator RAM		
	S/C=1:Display shift. 0:cursor move										ACG : CG RAM address		
	R/L=1:Shift right. 0: Shift left.										ADD : DD RAM address		
	DL= 1 : 8bits 0 : 4 bits										Corresponds to cursor address		
	N = 1 : 2 lines 0 : 1 lines										AC : Address counter used for both DD and CG RAM address		
	F = 1 : 5×10dots 0 : 5×7dots												
BF = 1: Internally operating 0: Can accept instruction													
* NO EFFECT													

VIII.2. Travail demandé

Ecrire un programme qui affiche la touche appuyée sur l'écran LCD.

IX. Solution type Travail Pratique N°4 Clavier et LCD

```

;*****
;
;   TP N°04 du 25 /05/2016
;   Affichage LCD,
;   Matière systemes embarques
;*****
;*****CODE   SEGMENT
;
;       ASSUMECS:CODE,DS:CODE,ES:CODE,SS:CODE
STACK   EQU     0540H
LCDC    EQU     00H
LCDC_S  EQU     02H
LCDD    EQU     04H

;
;       ORG     1000H
;       XOR     AX,AX
;       MOV     SS,AX
;       MOV     SP,STACK

L2:      CALLALLCLR ; Appel procédure utilisée pour effacer l'écran
;
;       MOVSI,OFFSET CUSOR1
;       CALL    CLAVIER
;
;       CALL     LN21
;       MOVSI,OFFSET CUSOR2
;       CALL     STRING
;
;       MOV     DL,16
L1:      CALL     TIMER
;       CALL     SHIFT
;       CALL     TIMER
;       DEC     DL
;       JNZ     L1
;       JMP     L2
;
CUSOR1   DB 'TP 04 RST 2015/2016',00H
CUSOR2   DB'Systèmes embarqués',00H
;

```

```

; instructions LCD
ALLCLR:  MOVAH,01H ; Procédure utilisée pour effacer l'écran
        JMP      LNXX
        ;
SHIFT:   MOV  AH,00011100B
        JMP      LNXX
        ;
LN21:    MOV   AH,0C0H
        ;
LNXX:    CALL   BUSY
        MOV    AL,AH
        OUT    LCDC,AL
        RET
BUSY:    IN     AL,LCDC_S
        AND    AL,10000000B
        JNZ    BUSY
        RET
        ;
        ; 1 char. LCD OUT
        ; AH = out data
CHAROUT:
        CALL   BUSY
        ;
        MOV    AL,AH
        OUT    LCDD,AL
        RET
        ;
Clavier: MOVAH,BYTE PTR CS:[SI]
        CMP    AH,00H
        JE     STRING1
        ; out
        CALL   BUSY
        CALL   CHAROUT
        INC    SI
        JMP    STRING
STRING1:
        RET

```

```
;  
TIMER:  MOV    CX,2  
TIMER2: PUSH    CX  
        MOV    CX,0  
TIMER1: NOP  
        NOP  
        NOP  
        NOP  
        LOOP   TIMER1  
        POP    CX  
        LOOP   TIMER2  
        RET  
;  
CODE    ENDS  
        END
```

X. Conclusion

Les programmes présentés paraissent simple, sauf que leur écriture requière une analyse détaillée du fonctionnement des circuits mis en œuvre. En effet, dans certaines configurations un 1 utilisé dans le programme permet d'activer l'interface alors que dans d'autres c'est le 0 qui le fait. Lors des différentes séances des travaux pratiques souvent les étudiants ont consommé du temps pour trouver la bonne combinaison ont opéré par tâtonnement, une approche qui s'adapte à trouver des solutions dans le cas de quelques LEDs. Cependant, avec un nombre important il est primordiale d'effectuer une analyse détaillée du schématique, déclarer convenablement les données dans la mémoire, une partie du travail qui prend presque la moitié du temps de la séance, enfin écrire les instructions dans ordre correcte.

Sur les dernières questions les étudiants commencent à comprendre la difficulté de la programmation assembleur et l'avantage d'une programmation évoluée.

Conclusion

Un de mes étudiants m'a posé la question suivante : Monsieur à quoi bon de travailler avec ces cartes de développements, à quoi bon d'étudier le langage machine, moi je suis un étudiant en télécommunications je ne pense pas que sa puisse me servir un jour.

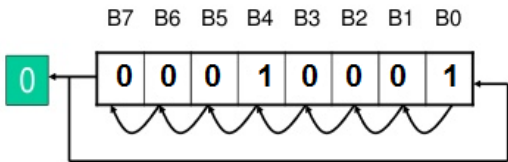
Je peux lui répondre en disant que contrôler des LEDs revient à gérer le flux routier, à informer des gens, beaucoup plus transmettre de l'information, et pourquoi pas gérer un allumage d'ambiance lors d'un moment de romance. J'ai appris par la suite que l'étudiant est un fan de la nouvelle philosophie du développement en système embarqué ADUINO. Des systèmes destinés au prototypage rapide des produits finaux, et créées en premier lieu pour des artistes. J'attire l'attention sur le point que ces systèmes sont dotés d'un Firmware qui facilite leur utilisation et leur confère le succès enregistré. Toutefois, pour certaines applications, particulièrement celles dédiées aux télécommunications, services nouveaux et protocoles actualisés, il faut toujours agir sur le Firmware, une tâche qui ne peut être réalisée que par une personne qui maîtrise la programmation bas niveau.

Bibliographie

- [1] John Sheesley, "Intel's 8086 passes the big 3-0", TechRepublic, June 16th, 2008.
- [2] Stephen P. Morse, Bruce W Ravenel, Stanley Mazor, William B. Pohlman, "Intel Microprocessors: 8008 to 8086", Computer Structures, pages 615-646, Siewiorek/Bell/Newell, 1982.
- [3] Emmanuel Viennet, Architecture des Ordinateur, GTR 1999-2000, IUT de Villetaneuse.
- [4] A. Oumnad, "Microprocesseurs de la famille 8086", Ecole des Ingénieurs Mohamadia, 2007.
- [5] Intel, "The 8086 family user's manual", Intel Corporation 1979.
- [6] <http://www.jameco.com/1/1/1111-8086-2-microprocessor-16-bit-32-i-o-8mhz-dip-40.html>
- [7] Cyril CAUCHOIS, "L'Assembleur Intel", Support de cours, Institut Universitaire de Technologie d'Amiens, 2000.
- [8] Intel, "Intel 64 and IA-32 architectures software developer's manual", Intel Corporation 2015.
- [9] MDA-WIN8086 Manual, An Integrated Development Environment kit, User's Manual, Midas Engineering Co., Ltd.
- [10] R. P. JAIN, M. M. S. ANAND, "Digital Electronics practice using integrated circuits", McGraw-Hill, 1984.

Annexe

Annexes des Instructions utilisées

<u>Mnémonique</u>	<u>Signification</u>	<u>Nombre de cycles</u>
OUT Adresse, AL	Ecrire le contenu du registre AL vers l'interface spécifiée par l'Adresse.	14
IN AL, Adresse	Lire dans le registre AL le contenu de l'interface spécifiée par l'Adresse.	14
MOV Registre, Valeur	Copier la Valeur dans le Registre (adressage immédiat)	04
MOV Registre1, Registre2	Copier le Registre2 dans le Registre 1	03
ROL Registre		02
NOP	No Opération	03
CMP Registre, Valeur	Effectue (Registre – Valeur), le résultat est perdu et actualise les flags : AF CF OF PF SF ZF	04
XOR registre1, registre2	XOR : registre1 = registre1 \oplus registre2	04
DEC Registre	Registre = Registre - 1	03
JNZ étiquette	IP = étiquette si ZF = 0	04
JNE étiquette		16 cas du jump
JLE étiquette	IP = étiquette si CF=1 ou ZF =1	04
JBE étiquette		16 cas du jump
JMP étiquette	IP = étiquette.	11
{id_Adr} DB valeur	Une directive qui définit un octet dans la mémoire est l'initialise avec valeur. L'adresse de cette case est accessible via id_Adr.	/
Identificateur EQU valeur	Déclaration de constante, une directive pour l'assembleur.	/
LOOP offset	C'est une instruction de boucle utilisant CX comme compteur décrémente à	04 08 dernière itération

Annexe

	chaque exécution de Loop, si ZF=1 alors IP= offset.	
CALL offset	Le registres IP est empilé dans la pile, IP = offset	19
RET	IP est dépilé de la pile	16

Préface	5
Chapitre 1 Présentation générale du système.....	9
<i>I. Introduction</i>	11
<i>II. Description générale de la carte de développement</i>	11
<i>III. Les Registres du 8086 :</i>	16
III.1. Les registres segments.....	16
III.2. Les registres à usage général.....	17
III.3. Les registres d'adressage offset	18
<i>IV. Travail Pratique 01 Initiation (Durée une séance de 3 heures)</i>	19
IV.1. Questions.....	19
IV.2. Travail demandé.....	19
<i>V. Solution Travail Pratique 01 Initiation</i>	21
V.1. Questions.....	21
V.2. Travail demandé.....	21
<i>VI. Conclusion</i>	23
Chapitre 2 Manipulations avec l'interface parallèle.....	25
<i>I. Introduction</i>	27
<i>II. Format des instructions et programmes</i>	27
II.1. Les instructions	27
II.2. Le format des programmes rédigés en assembleur	29
<i>III. Présentation de l'interface parallèle</i>	30
<i>IV. Travail Pratique 02 (Durée 6 heures) Manipulations sur l'afficheur 7 segments et LEDs</i>	32
IV.1. Description du dispositif :.....	32
IV.2. Travail demandé:.....	34
<i>V. Solution Travail Pratique 02 : Manipulations sur l'afficheur 7 segments et LEDs</i>	35
V.1. Analyse du problème.....	35
V.2. Solutions.....	36
<i>VI. Travail Pratique N°3 (durée 6 heures) Panneau d'affichage LED</i>	41
VI.1. Description du dispositif.....	41
VI.2. Travail demandé.....	42
<i>VII. Solution Travail Pratique N°3 Panneau d'affichage LED</i>	43

VIII. Travail Pratique N°4 (durée 3 heures) Manipulation avec le clavier	49
VIII.1. Description du dispositif.....	49
VIII.2. Travail demandé.....	50
IX. Solution type Travail Pratique N°4 Clavier et LCD	51
X. Conclusion	53
Conclusion	55
Bibliographie	59
Annexe	63