Documentation sur
la machine Pédagogique

SOMMAIRE

Introduction

Tout a commencé en 1945 lorsque l'architecture de Von Neumann a été conçue. Elle utilise une structure de stockage unique pour conserver à la fois les instructions et les données demandées ou produites par le calcul.

Nous nous sommes inspirées d'un processeur ayant cette architecture qui est le INTEL 80X86 ainsi de la machine MIASM pour concevoir une machine pédagogique dont nous avons choisi 'Passe' comme nom.

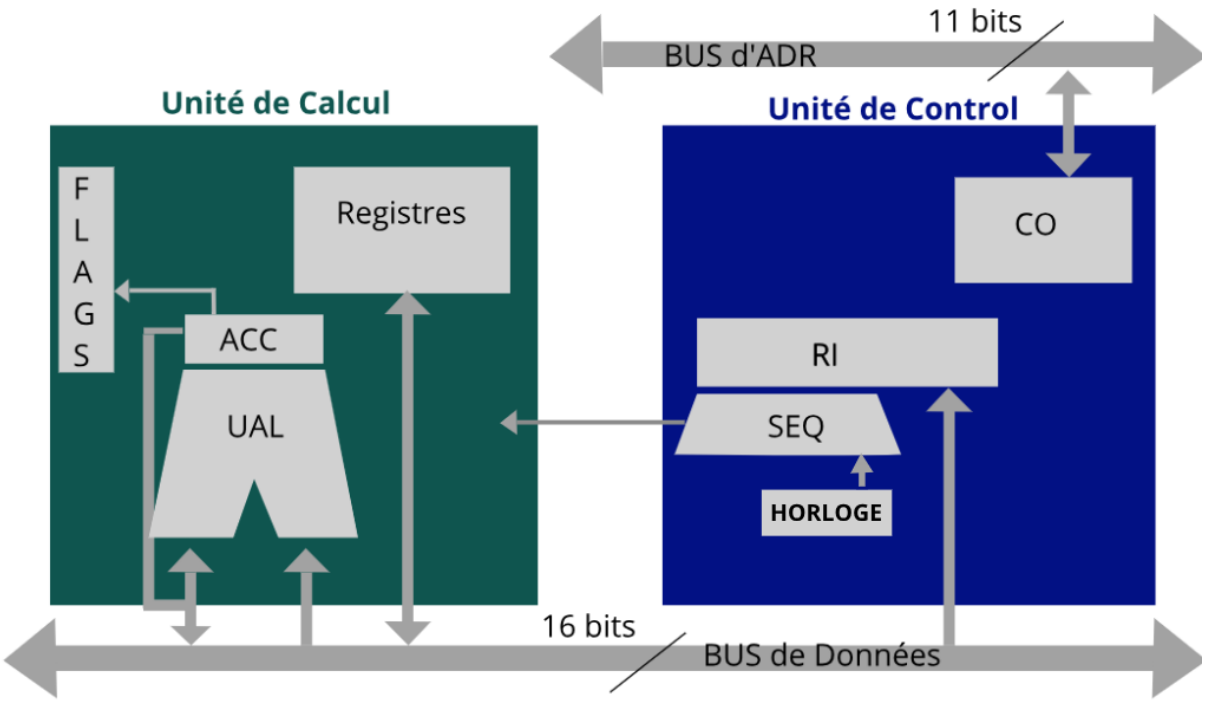
Passe est une machine d'une simple conception dédiée spécialement aux étudiants de la première année en Informatique. Elle leur permettra d'appréhender les concepts fondamentaux de l'architecture de l'ordinateur à savoir: les bus de données et d'adresses, la mémoire centrale, l'unité arithmétique et le compteur ordinal.

Afin de réaliser ce dessein nous avons concrétisé cette architecture en développant un simulateur dans le cadre d'une application web, permettant de suivre le déroulement d'un ensemble d'instructions, chose que nous espérons mènera à davantage compréhension de cet émulateur.

Architecture de la Machine

Architecture Générale Du Calculateur

Les différents constituants du processeur peuvent être regroupés en deux unités principales l'unité de Control et l'unité de Calcul (Exécution). Ci- dessous un schéma fonctionnel du processeur .



[MICROPROCESSEURS DE LA FAMILLE 8086 par A. Oumnad]

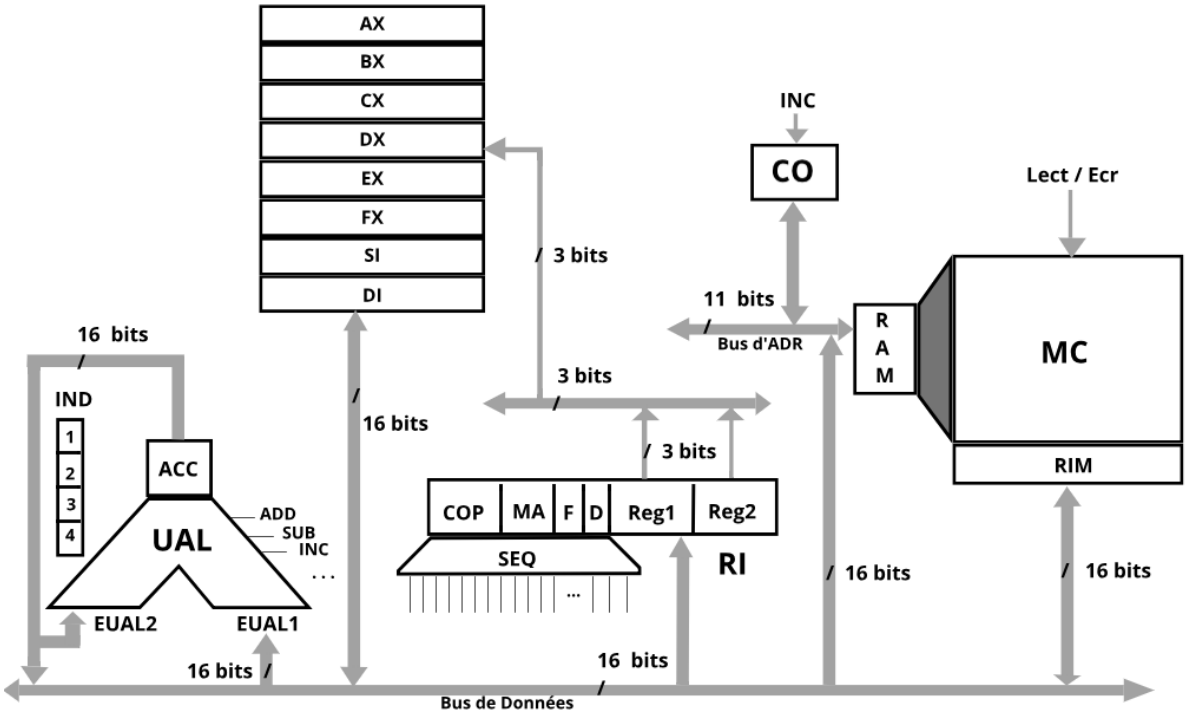
L'unité de Control

Elle est responsable de la gestion et de la coordination du fonctionnement du processeur, et supervise le déroulement de toutes les opérations au sein du processeur. Elle reçoit les instructions à partir de la mémoire centrale, les décode et les transmet à l'unité de calcul pour leur exécution.Elle est constituée principalement du CO(compteur ordinal), RI(registre d'instructions), le Séquenceur et l'horloge.

L'unité de Calcul (Exécution)

Elle est constituée de l'UAL (unité arithmétique et logique) qui effectue les opérations sur les données, et d'un ensemble de registres.

Architecture Détaillée De La Machine



UAL : unité arithmétique et logique de la machine . Elle s'occupe de l'exécution de toutes les opérations logiques et arithmétiques décrites dans le jeu d'instructions .

IND : les registres flags (sur 16 bits) qui mémorisent certaines informations après l'exécution des opérations.

Bit n°0: s'il est à 1, le résultat de l'opération effectuée est *zéro*. S'il est à 0, le résultat est différent de zéro.

Bit n°1 : pour le *signe* du résultat . Si le résultat est positif il est à 0 et s'il est négatif le bit est à 1.

Bit n°2 : pour la *retenue* . s'il est à 1 ,il ya eu une retenue et s'il est à 0 , pas de retenue.

Bit n°3 : pour le *débordement* (overflow) . si ce bit est à 1 : il y a débordement et si c'est 0 : pas de débordement.

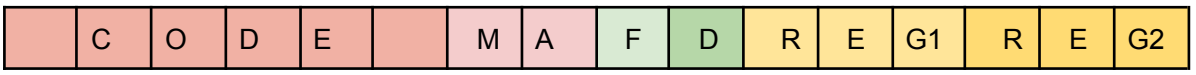
MC : c'est la Mémoire Centrale , l'organe qui s'occupe du stockage de toutes les données et les instructions manipulées par le processeur . Elle est de capacité 2048 bits (2 kilos octets) .

RAM : Registre Adresse Mémoire , c'est le registre qui stocke l'adresse de mot mémoire à adresser .Il est à 11 bits (la même taille que le bus d'adresse).

RIM : Registre Information Mémoire . c'est le registre qui stocke les informations lues à partir de la MC (données / instructions) ,sur 16 bits.

CO : Compteur Ordinal ,un registre de 11 bits qui stocke l'adresse de la prochaine instruction à exécuter, et est initialisé avec l'adresse de la première instruction d'un programme .

RI : Registre Instructions, sur 16 bits qui contient l'instruction au cours d'exécution. Les instructions ont un format spécifique, comme illustré ci-dessous, elles sont décodées par le décodeur (Séquenceur) .



Ce format sera détaillé un peu plus , dans les chapitres qui suivent .

Autres Registres

ACC : contient le résultat de l'opération exécutée dans l'UAL.
AX, BX, CX, DX, EX, FX : utilisées comme des registres normaux à usage général

Note : BX est utilisé dans l'adressage .

SI, DI : registres d'index . utilisées dans les opérations d'indexation (les tableaux) .

Ci dessous un tableau détaillant l'usage de chaque registre.

Registre	Usage
AX	Usage général **Ne peut pas servir pour l'adressage**
BX	Usage général Adressage
CX	Usage général Compteur de répétition **Ne peut pas servir pour l'adressage**

DX	Usage général **Ne peut pas servir pour l'adressage**
EX	
FX	
SI	Usage général Adressage comme registre d'index
DI	

[PROCESSEUR INTEL 80X86- AROUSSI 2013 - 2014]

Les Bus

BUS de données : sur **16 bits**
BUS d'adresses : sur **11 bits**
BUS d'adressage registres : sur **3 bits**. A travers ce bus et en utilisant les numéros de registres, présent dans un champ spécifique dans le RI, se fait la sélection de l'un des registres à utiliser .

Mode d'Adressage

Le terme modes d'adressage fait référence à la manière dont l'opérande d'une instruction est spécifié.

On distingue 4 modes d'adressage :

Adressage Direct
L'un des deux opérandes nous fournit l'emplacement direct de la donnée.
On accède directement soit à l'adresse mémoire spécifiée ou bien à un registre (adressage registre) pour récupérer directement ces données.

Adressage indirect
L'un des deux opérandes se trouve en mémoire, l'adresse de la mémoire n'est pas fourni directement à l'instruction. Au lieu de cela l'instruction contient un registre qui pointe vers une autre adresse mémoire. Ce registre est parmi les registres d'adressage suivants: BX, SI, DI.

Adressage Basé/Indexé :
Dans ce mode l'un des deux opérandes représente une adresse calculée en sommant le contenu d'un registre d'adressage avec un déplacement écrit en décimal. Si le registre d'adressage est BX alors on dira que le mode est Basé et dans le cas où le registre est DI ou SI on dira que le mode d'adressage est indexé.

Adressage Direct Indexé
Dans ce mode l'un des deux opérandes est représenté par une étiquette suivie d'une adresse calculée en sommant le contenu d'un registre d'adressage (SI ou DI) avec un déplacement écrit en décimal. L'adresse de la donnée souhaitée est accédée en ajoutant l'adresse où l'étiquette est sauvegardée dans la mémoire au résultat de l'adresse calculée auparavant. Ce mode est très util pour l'usage des tableaux.

Jeux d'instructions

Les types d'instructions traité :

- Instructions de transfert (MOV,LOAD,STORE,...)
- Instructions Arithmétique (ADD,SUB,INC,DEC,…)
- Instructions logiques (NOT,AND,OR,...)
- Instructions de décalage (SHR,SHL,ROL,ROR)
- Instructions de branchement (JMP,...)
- Instructions d'entrées/sorties(IN,OUT)
- Instructions d'arrêt(STOP)

1. Instructions de transfert

MOV Od,Os : copier le contenu de l'opérande source (Os) dans l'opérande destination (Od) [Od ← Os] . (Od/Os : reg,adr)

MOV reg1,reg2	reg1←reg2
MOV reg,[adr]	reg←[adr]
MOV [adr],reg	[adr]←reg
MOV reg1,[reg2]	reg1←[reg2]
MOV [reg1],reg2	[reg1]←reg2
MOV etiq[Ri+dépl],reg	etiq[Ri+dépl]← reg
MOV [RX+dépl],reg	[RX+dépl]←reg
MOV reg,etiq[Ri+dépl]	reg←etiq[Ri+dépl]
MOV etiq[Ri+dépl],reg	etiq[Ri+dépl]←reg

X MOV [adr],[adr]

X MOV reg,reg

Exemple :

MOV AX,BX // AX ←BX

MOVI Od,Os : copier le contenu de l'opérande source (Os) dans l'opérande destination (Od) [Od ← Os] (Os est une valeur immédiate) .

MOVI reg,val	reg←val
MOVI [adr],val	[adr]←val
MOVI [reg],val	[reg]←val
MOVI [RX+dépl], val	[RX+dépl]←val

Exemple :

MOVI EX,7H // EX←7H

LOAD Op : Charge l'accumulateur par Op (Op : reg,[adr]) .

LOAD reg	acc←reg
LOAD [reg]	acc←[reg]
LOAD [adr]	acc←[adr]
LOAD tab[XI]	acc←tab[XI] /XI : SI,DI

Exemple :
LOAD CX // ACC←CX

LOADI Op : Charge l'accumulateur par une valeur immédiate.

Exemple :
LOAD 5H // ACC←5H

STORE [adr] : Stocker le contenu de l'accumulateur dans une adresse mémoire .

Exemple :

STORE [0005H]

2. Instructions Arithmétiques

L’addition :

ADD Od,Os :additionne l'opérande source et l'opérande destination et met le résultat dans l'opérande destination (Od/Os : reg,adr) .

ADD reg1,reg2	reg1←reg1+reg2
ADD reg,[adr]	reg←reg+[adr]
ADD [adr],reg	[adr]←[adr]+reg
ADD reg1,[reg2]	reg1←reg1+[reg2]
ADD [reg1],reg2	[reg1]←[reg1]+reg2
ADD etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]+ reg
ADD [RX+dépl],reg	[RX+dépl]←[RX+dépl]+reg
ADD reg,etiq[Ri+dépl]	reg←reg+etiq[Ri+dépl]
ADD etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]+reg

X ADD [adr],[adr]

Exemple :

```
ADD AX,BX    // AX ←AX+BX
```

ADDI Od,Os : additionne l'opérande source et l'opérande destination et met le résultat dans l'opérande destination (Od : reg,adr) (Os: Valeur immédiate) .

ADDI reg,val	reg←reg+val
ADDI [adr],val	[adr]←[adr]+val
ADDI [reg],val	[reg]←[reg]+val

ADDI [RX+dépl],val	[RX+dépl]←[RX+dépl]+val

Exemple :
ADDI AX,5H // AX ←AX+5H

ADA Op : additionne l'opérande Op et le contenue de l'accumulateur et met le résultat dans Op .

ADA reg	acc←reg+acc
ADA [reg]	acc←[reg]+acc

Exemple :
ADA FX //acc←acc+FX

ADAI Op : additionne Op (une valeur immédiate) et le contenue de l'accumulateur .

ADAI Op	acc←acc+Op

Exemple :
ADA 8H //acc←acc+8H

INC Op : incrément l'opérande Op[Op←Op+1] (Op est un registre) .

Exemple :
INC SI //SI←SI+1

La soustraction :

SUB Od,Os :soustrait l'opérande source de l'opérande destination et met le résultat dans l'opérande destination [Od←Od-Os]. (Od/Os : reg,adr) .

SUB reg1,reg2	reg1←reg1-reg2
SUB reg,[adr]	reg←reg-[adr]
SUB [adr],reg	[adr]←[adr]-reg
SUB reg1,[reg2]	reg1←reg1-[reg2]

SUB [reg1],reg2	[reg1]←[reg1]-reg2
SUB etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]- reg
SUB [RX+dépl],reg	[RX+dépl]←[RX+dépl]-reg
SUB reg,etiq[Ri+dépl]	reg←reg-etiq[Ri+dépl]
SUB etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]-reg

X SUB [adr],[adr]

Exemple :
SUB DX,[0010H] // DX ←DX-[0010H]

SUBI Od,Os : soustrait l'opérande source de l'opérande destination et met le résultat dans l'opérande destination [Od←Od-Os]. (Od : reg,adr) (Os: Valeur immédiate) .

SUBI reg,val	reg←reg-val
SUBI [adr],val	[adr]←[adr]-val
SUBI [reg],val	[reg]←[reg]-val
SUBI [RX+dépl],val	[RX+dépl]←[RX+dépl]-val

Exemple :
SUBI FX,2H // FX ←FX-2H

SBA Op : soustrait l'opérande Op du contenu de l'accumulateur et met le résultat dans Op.

SBA reg	acc ← acc-reg
SBA [reg]	acc ← acc-[reg]

Exemple :
SBA CX //acc ←acc-CX

SBAI Op : soustrait Op (une valeur immédiate) du contenu de l'accumulateur .

SBAI Op	acc←acc-Op

Exemple :
SBAI 9H //acc←acc-9H
DEC Op : décrémente l'opérande Op[Op←Op-1] (Op est un registre) .

Exemple :
DEC DI // DI ←DI-1

Comparaison:

CMP Od,Os: compare (par une soustraction) les opérandes Od et Os et positionne les flags (registre d'état) en fonction du résultat. **L'opérande Od n'est pas modifiée.**(Od/Os : reg)

CMPI Od,Os: compare (par une soustraction) les opérandes Od et Os et positionne les flags (registre d'état) en fonction du résultat**. L'opérande Od n'est pas modifiée.** (Od : reg) (Os: Valeur immédiate) .

	Opérandes non Signés				Opérandes Signés			
	Z	S	R	O	Z	S	R	O
Od>Os	0	0	0	-	0	0	-	0/1
Od=Os	1	0	0	-	1	0	-	0
Od<Os	0	1	1	-	0	1	-	0/1

Exemple :

MOV AX,7H
CMPI AX,9H // on est dans 4eme cas de tableau au dessous Z=0 S=1 R=1

3. Instructions Logiques

Négations :

NOT Op: transforme la valeur de l'opérande (registre) en son complément à 1 [OP←CA1(OP)].

Exemple :

```
MOV AX,60H
NOT AX   // AX= FF9F
```

Et logique :

AND Op,Os : effectue un ET logique entre Od et Os.[Od←Od **et** Os](Od/Os:reg,adr)

AND reg1,reg2	reg1←reg1^reg2
AND [adr],reg	[adr]←[adr]^reg
AND reg,[adr]	reg←[adr]^reg
AND reg1,[reg2]	reg1←reg1^[reg2]
AND [reg1],reg2	[reg1]←[reg1]^reg2
AND etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]^reg
AND [RX+dépl],reg	[RX+dépl]←[RX+dépl]^reg
AND reg,etiq[Ri+dépl]	reg←reg^etiq[Ri+dépl]
AND etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]^reg

X AND [adr],[adr]

Exemple :

```
MOV  AX,503H
MOV BX,0201H
AND AX,BX
// AX=0001H
```

AX

0000	0101	0000	0011

BX

0000	0010	0000	0001

AX

0000	0000	0000	0001

ANDI Op,Os : effectue un ET logique entre Od et Os.[Od←Od **et** Os](Od:reg/adr)(Os:valeur immédiate)

ANDI reg,val	reg←reg^val
ANDI [adr],val	[adr]←[adr]^val
ANDI [reg],val	[reg]←[reg]^val
ANDI [RX+dépl], val	[RX+dépl]←[RX+dépl]^val

Exemple :

```
MOV  AX,0503H
ANDI AX,0201H
// AX=0001H
```

AX

0000	0101	0000	0011

0201H

0000	0010	0000	0001

AX

0000	0000	0000	0001

OU logique :

OR Od, Os : effectue un OU logique entre Od et Os.[Od←Od□Os](Od/Os:reg,adr)

OR reg1,reg2	reg1←reg1□reg2
OR [adr],reg	[adr]←[adr]□reg
OR reg,[adr]	reg←[adr]□reg

OR reg1,[reg2]	reg1←reg1□[reg2]
OR [reg1],reg2	[reg1]←[reg1]□reg2
OR etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]□reg
OR [RX+dépl],reg	[RX+dépl]←[RX+dépl]□reg
OR reg,etiq[Ri+dépl]	reg←reg□etiq[Ri+dépl]
OR etiq[Ri+dépl],reg	etiq[Ri+dépl]←etiq[Ri+dépl]□reg

X OR [adr],[adr]

Exemple :

```
MOV  AX,0503H
MOV  BX,0201H
ORI    AX,BX
// AX=0703H
```

AX

0000	0101	0000	0011

BX

0000	0010	0000	0001

AX

0000	0111	0000	0011

ORI Op,Os : effectue un OU logique entre Od et Os.[Od←Od **et** Os](Od:reg/adr)(Os:valeur immédiate)

ORI reg,val	reg←reg□val
ORI [adr],val	[adr]←[adr]□val
ORI [reg],val	[reg]←[reg]□val
ORI [RX+dépl], val	[RX+dépl]←[RX+dépl]□val

Exemple :

```
MOV  AX,503H
ORI   AX,0201H
// AX=0703H
```

AX

0000	0101	0000	0011

0201H

0000	0010	0000	0001

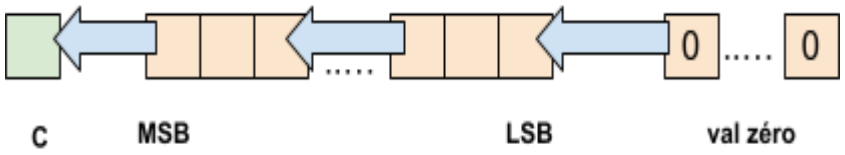
AX

0000	0111	0000	0011

4. Instructions De Décalage

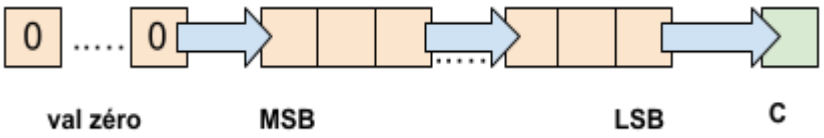
SHR reg,val	Décalage Logique Gauche (Shift Logical Left)
SHL reg,val	Décalage Logique Droit (Shift Logical Right)
ROL reg,val	Décalage Circulaire Gauche (Rotate Left)
ROR reg,val	Décalage Circulaire Droit (Rotate Right)

SHR reg,val



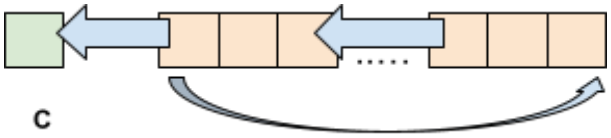
LSB : bit de poid faible
MSB : bit de poid fort

SHL reg,val

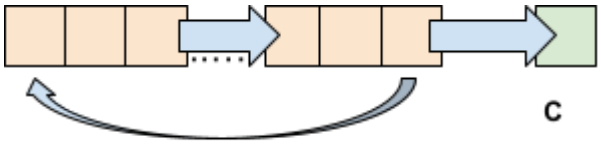


LSB : bit de poid faible
MSB : bit de poid fort

ROL reg,val



ROR reg,val



5. Instructions De Branchement

On distingue 2 types de branchement :

- Branchement conditionnelle .
- Branchement inconditionnel .

Instruction	Nom	Condition
JZ	**J**ump if **Z**ero	Saut si Z=1
JNZ	**J**ump if **N**ot **Z**ero	Saut si Z=0
JC	**J**ump if **C**arry	Saut si C=1
JNC	**J**ump if **N**ot **C**arry	Saut si C=0
JS	**J**ump if **S**ign	Saut si S=1
JNS	**J**ump if **N**ot **S**ign	Saut Si S=0
JO	**J**ump if **O**verflow	Saut Si O=1
JNO	**J**ump if **N**ot **O**verflow	Saut Si O=0
JE	**J**ump if **E**qual	Saut si Z=0
JNE	**J**ump if **N**ot **E**qual	Saut si Z=1
JMP	**J**ump	Saut sans condition

6. Les instructions d’entrées/Sorties

IN : Lire une valeur à partir du clavier .
OUT : Afficher une valeur dans l'écran .

7. Instruction de début

START : indique que c’est le début du programme.

8. Instruction d'arrêt

STOP : indique que c’est la fin du programme.

Format d’instructions machine

Les instructions ont le format suivant :

CODE	MA	Format	D	REG1	REG2

- CODE : sur 6 bits

Instruction	COP
MOV	000000
MOVI	100000
ADD	000001
ADDI	100001
ADA	000010
ADAI	100010
SUB	000011
SUBI	100011
SBA	000100
SBAI	100100
CMP	000101
CMPI	100101
OR	000110
ORI	100110
AND	000111
ANDI	100111
SHR	001000
SHL	001001
ROL	001010
ROR	001011
JZ	001100
JNZ	001101
JC	001110
JNC	001111
JS	010000
JNS	010001
JO	010010

JNO	010011
JE	010100
JNE	010101
LOAD	010110
LOADI	110110
STORE	010111
INC	011000
DEC	011001
NOT	011010
JMP	011011
IN	011100
OUT	011101
STOP	011110
START	011111

- Mode d'adressage (MA) : sur 2 bits

Mode	**Code**
00	Direct
01	Indirect
10	Basé/Indexé
11	Direct Indexé

- Format : sur 1 bit

  F=0 → Format court l'instruction est sur un seul mot

  F=1 → Format long l'instruction est sur 2 ou 3 mots

- Destination : sur 1 bit

  D=0 → REG1 est source

  D=1 → REG1 est destinataire

- Les registres (REG1/REG2) : sur 3 bits

CODE	**REGISTRE**
000	AX
001	BX
010	CX
011	DX
100	EX
101	FX
110	SI
111	DI

Exemples de programmations

Commentaire	**Étiquette**	**Instruction**	**Param1**	**Param2**
l'adresse début du programme		**ORG**	**100H**	
réserver et initialiser A avec la valeur		**SET**	**A**	**45H**

45H				
		SET	**B**	**26H**
		START		
copier la valeur de A vers le registre AX		**MOV**	**AX**	**A**
additionner le contenu de AX avec la valeur contenue dans B		**ADD**	**AX**	**B**
charger l'accumulateur avec le contenue du registre AX		**LOAD**	**AX**	
sauvegarder la valeur de l'accumulateur dans l'adresse B		**STORE**	**B**	
la fin de l'exécution		**STOP**		

Commentaire	**Étiquette**	**Instruction**	**Param1**	**Param2**
l'adresse début du programme		**ORG**	**100H**	
réserver et initialiser A avec la valeur 45H		**SET**	**A**	**0H**
		SET	**B**	**0H**
		START		
la saisie la valeur de A par l'utilisateur		**IN**	**A**	
la saisie de la valeur de B par l'utilisateur		**IN**	**B**	
copier la valeur de A dans le registre AX		**MOV**	**AX**	**A**
copier la valeur de B dans le registre BX		**MOV**	**BX**	**B**
comparer le contenu de Ax et BX, faire la substitution, sauvegarder le résultat dans l'accumulateur et placer les flags		**CMP**	**AX**	**BX**
sauter à l'étiquette FIN si les contenus sont differents(jump if not equal)		**JE**	**FIN**	

sauvegarder le résultat de la comparaison dans le registre CX		**STORE**	**CX**	
la fin de l'exécution	**FIN**	**STOP**		

Ressources documentaires

- *MICROPROCESSEURS DE LA FAMILLE 8086 par A. Oumnad*
- *PROCESSEUR INTEL 80X86- AROUSSI 2013 - 2014*