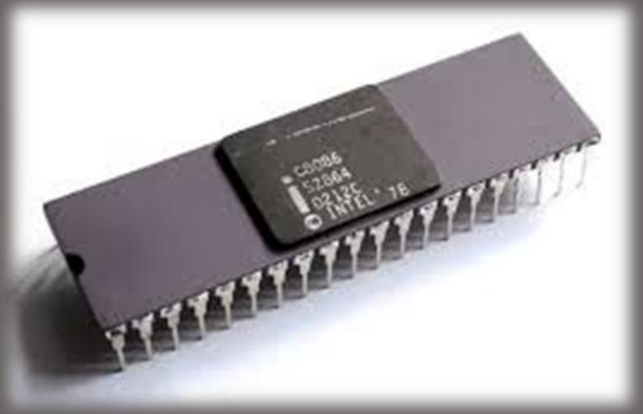
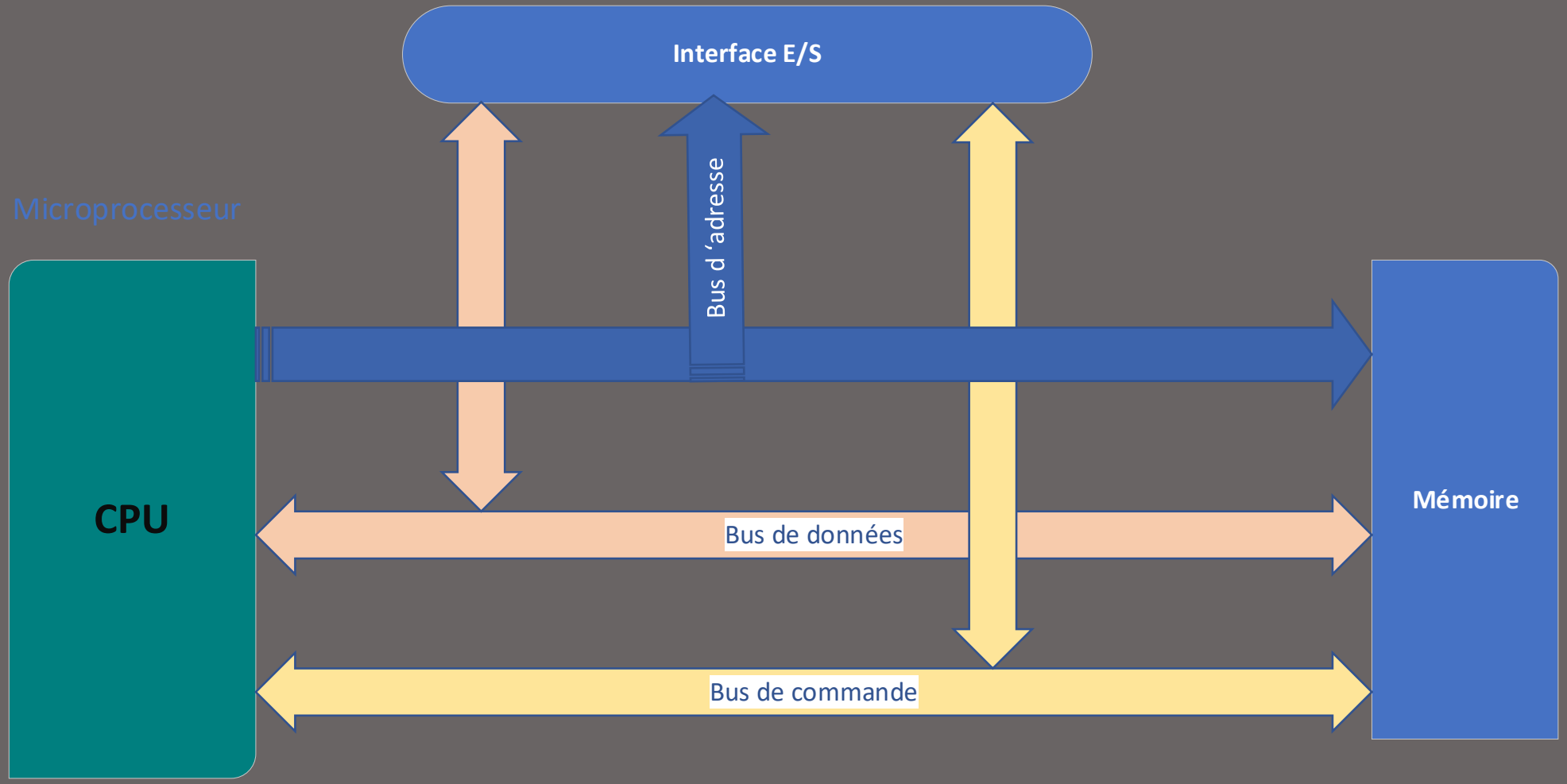
Microprocesseur et Api



ENSEIGNANT: BOUKHAMKHAM Hassen

Architecture de processeur



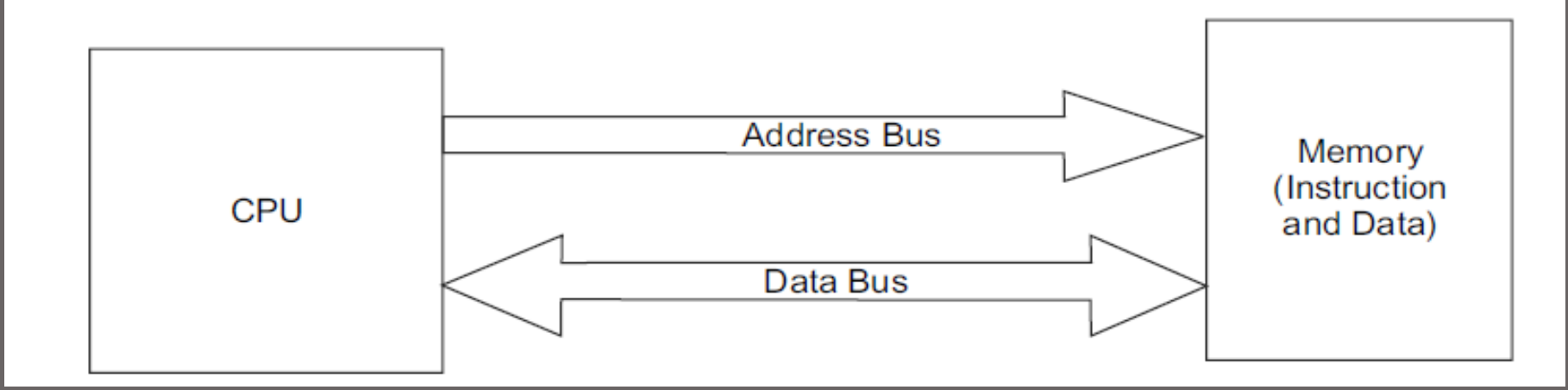
Selon le nombre de bus de données et de mémoire, il existe trois types d'architecture de processeur tel que

- Architecture de Von Neumann
- Architecture de Harvard
- Architecture de Super Harvard

Von Neumann architecture :

La **figure** 1 montre l'architecture des processeurs Von Neumann et cette architecture est la plus couramment utilisée.

dans les processeurs. Dans cette architecture, une puce mémoire est utilisée pour stocker à la fois les instructions et les données. Le processeur interagit avec la mémoire via des bus d'adresses et de données pour récupérer des instructions ainsi que des données.



Tous les ordinateurs sont conçus sur la même architecture, dite architecture de Von Neumann, qui a été proposée par le mathématicien John Von Neumann en 1948.

Architecture de Harvard

La figure 02 montre l'architecture Harvard d'un processeur. Dans cette architecture de processeur, deux mémoires distinctes

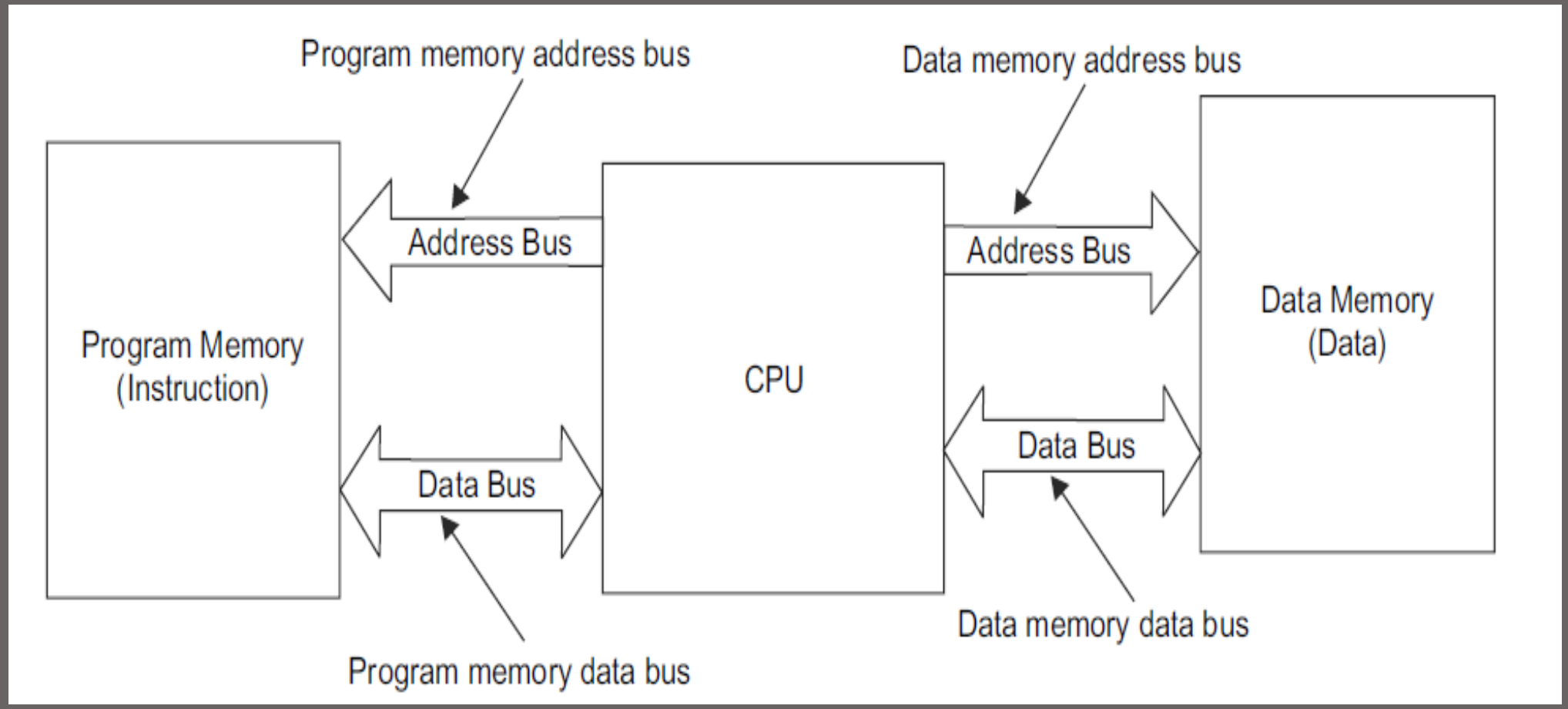
des blocs, à savoir la mémoire de programme et la mémoire de données, sont utilisés. La mémoire du programme est utilisée pour stocker uniquement

Les instructions et la mémoire de données sont utilisées pour stocker les données. Le bus d'adresses de la mémoire du programme est utilisé pour localiser le

Mémoire de programme et via le bus de données de mémoire de programme, le processeur peut écrire/lire des instructions vers/depuis

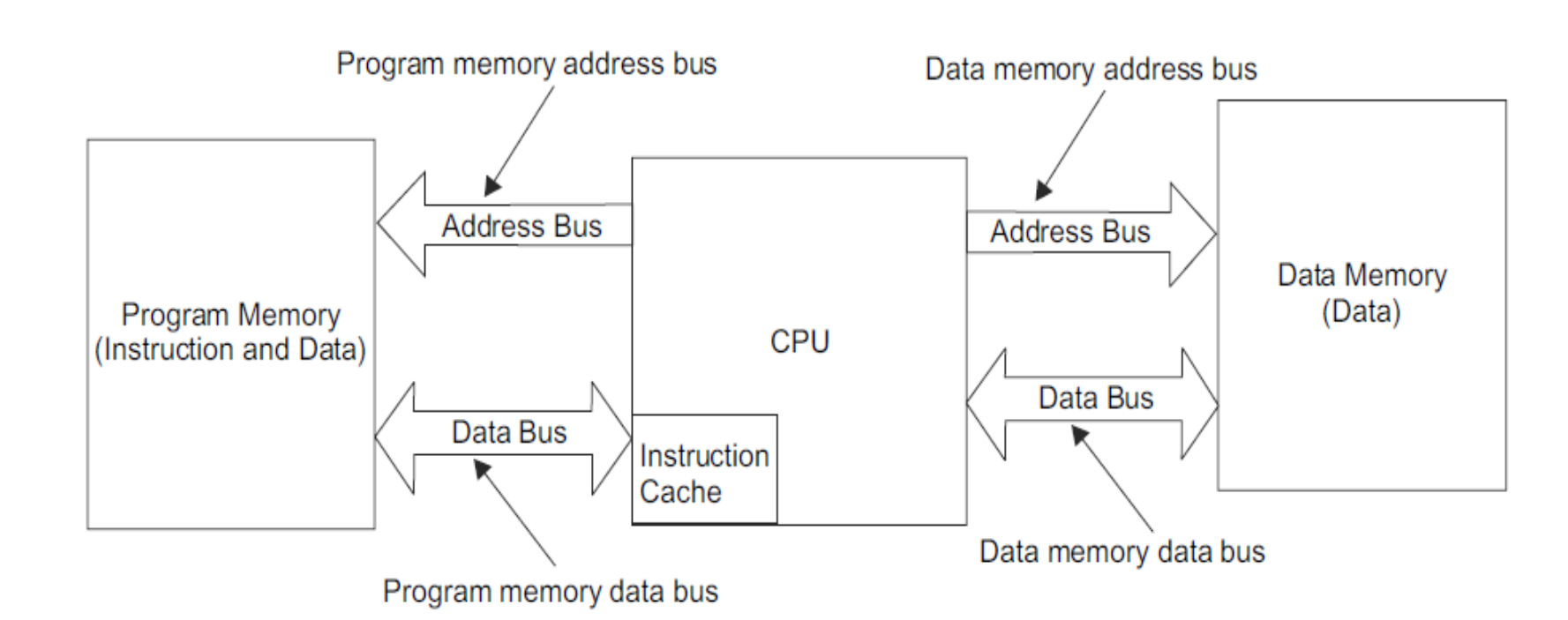
Mémoire. De même, le bus d'adresses de mémoire de données est utilisé pour localiser la mémoire de données et les données de la mémoire de données

Le bus peut être utilisé pour accéder à la mémoire de données. Par conséquent, cette architecture est plus efficace que Von Neumann architecture car les instructions et les données seront accessibles très rapidement.


Program memory address bus
Data memory address bus
Program Memory
(Instruction)
Address Bus
CPU
Address Bus
Data Memory
(Data)
Data Bus
Data Bus
Program memory data bus
Data memory data bus

Architecture de Super Harvard

La figure 03 montre l'architecture super Harvard qui est l'architecture Harvard modifiée. la mémoire des données est consultée plus fréquemment que la mémoire du programme dans l'architecture Harvard. Dans le super Architecture Harvard, la mémoire du programme peut stocker des données secondaires pour équilibrer la charge sur les deux programmes mémoire et mémoire de données. Le cache d'instructions est intégré au processeur. Cette architecture est la plus couramment utilisé dans le traitement du signal numérique (DSP).



2-Mémoire

2-1 Rappel

La capacité (taille) d'une mémoire est le nombre (quantité) d'informations qu'on peut enregistrer (mémoriser) dans cette mémoire.

•La capacité peut s'exprimer en :

– Bit : un bit est l'élément de base pour la représentation de l'information .

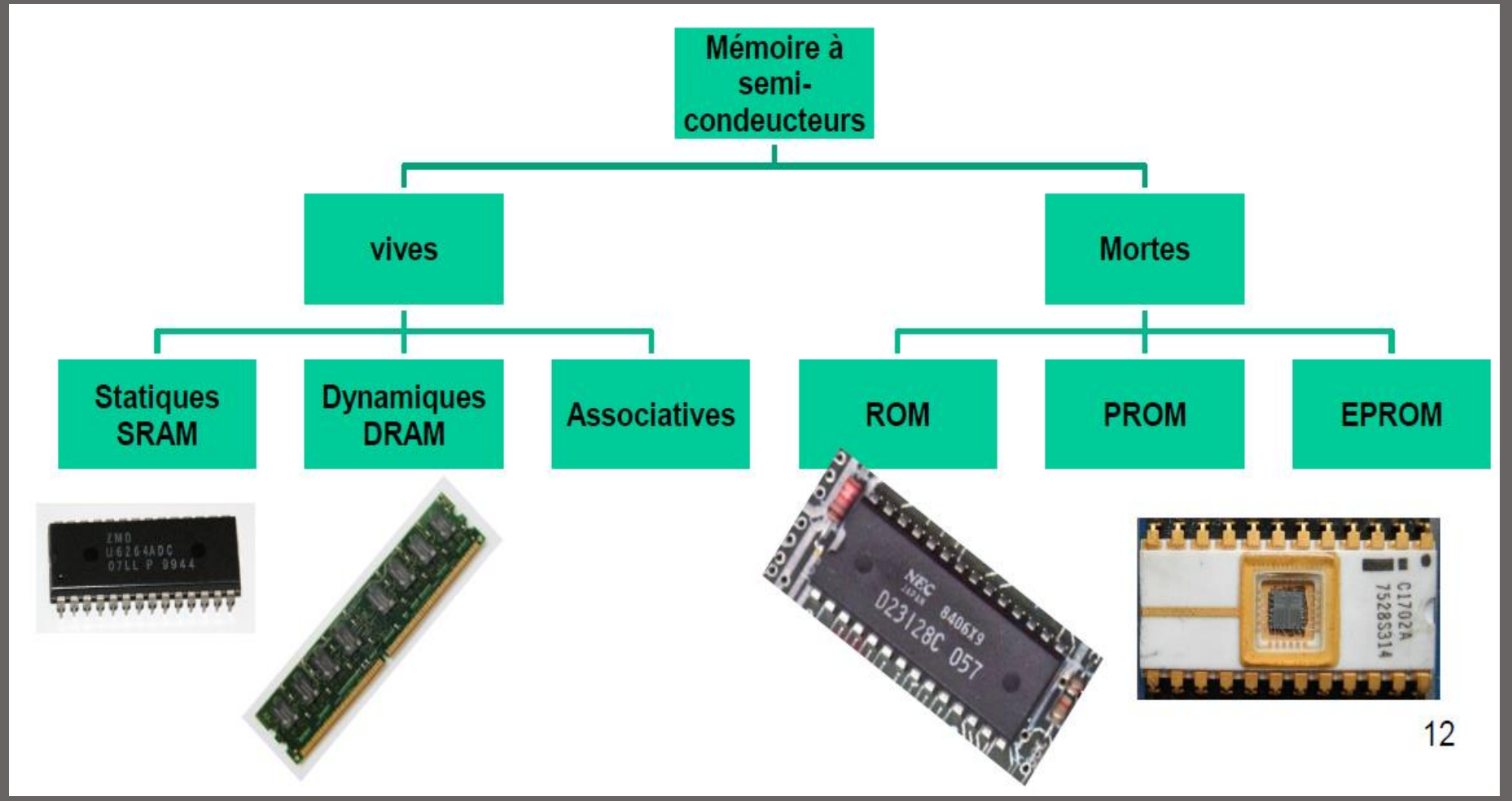
–Octet : 1 Octet = 8 bits

–kilo-octet (KO) : 1 kilo-octet (KO)= 1024 octets = 210 octets

–Méga-octet (MO) : 1 Méga-octet (MO)= 1024 KO = 220 octets

–Géga-octet (GO) :Géga-octet (GO)=1024 MO = 230 octets

–Téra-octet (To) : 1 téra-octet (To)= 1024 Go =240 octets


Mémoire à semi-condeucteurs
vives
Mortes
Statiques SRAM
Dynamiques DRAM
Associatives
ROM
PROM
EPROM
ZMD U6264ADC 07LL P 9944
NEC JAPAN 8406X9 D23128C 057
C1702A 7528S314

Si une mémoires perd sont contenu (les informations) lorsque la sources d'alimentation est coupée alors la mémoire est dite volatile.

Si une mémoire ne perd pas (conserve) sont contenu lorsque la sources d'alimentation est coupée alors la mémoire est dite non volatile (mémoire permanente ou stable).

Sur une mémoire on peut effectuer l'opération de :

• lecture : récupérer / restituer une information à partir de la mémoire.

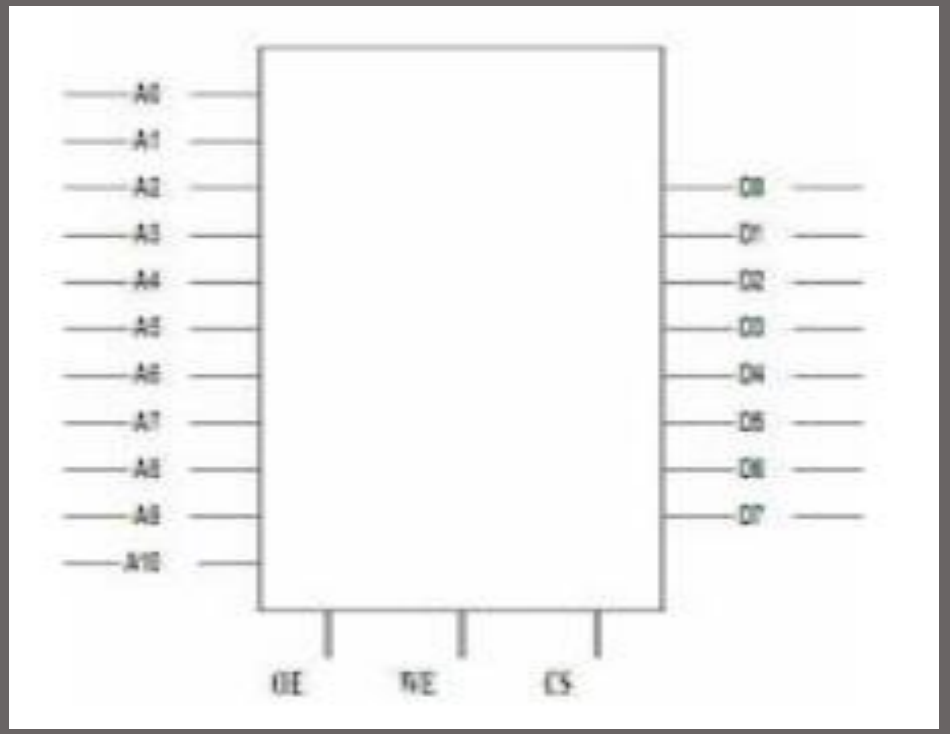
•écriture : enregistrer une nouvelle information ou modifier une information déjà existante dans la mémoire .

•Il existe des mémoires qui offrent les deux modes lecteur/écriture , ces mémoire s'appelles **mémoires vives**.

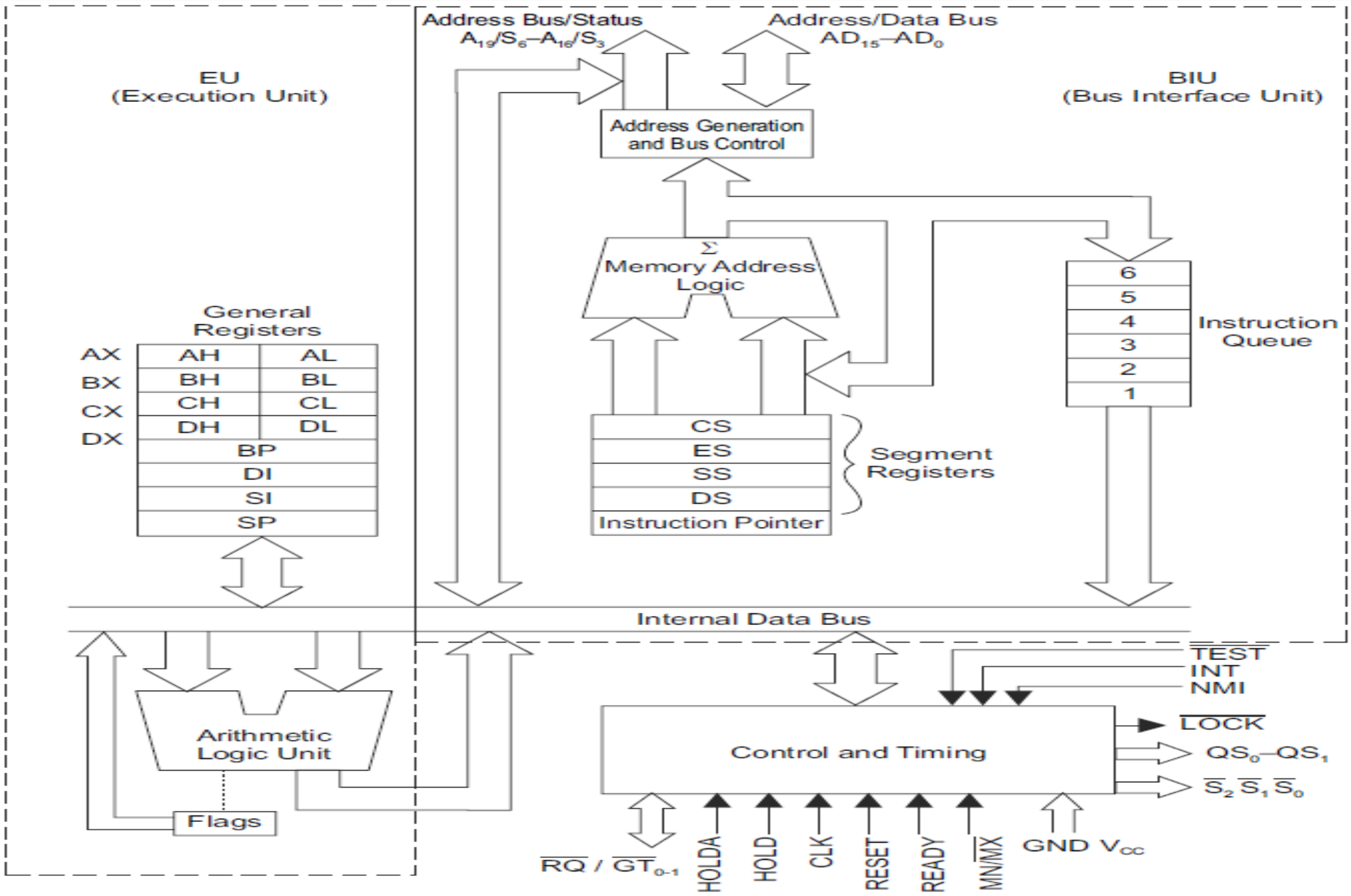
•Il existent des mémoires qui offrent uniquement la possibilité de la lecture (c'est pas possible de modifier le contenu). Ces mémoires s'appelles **mémoires mortes.**

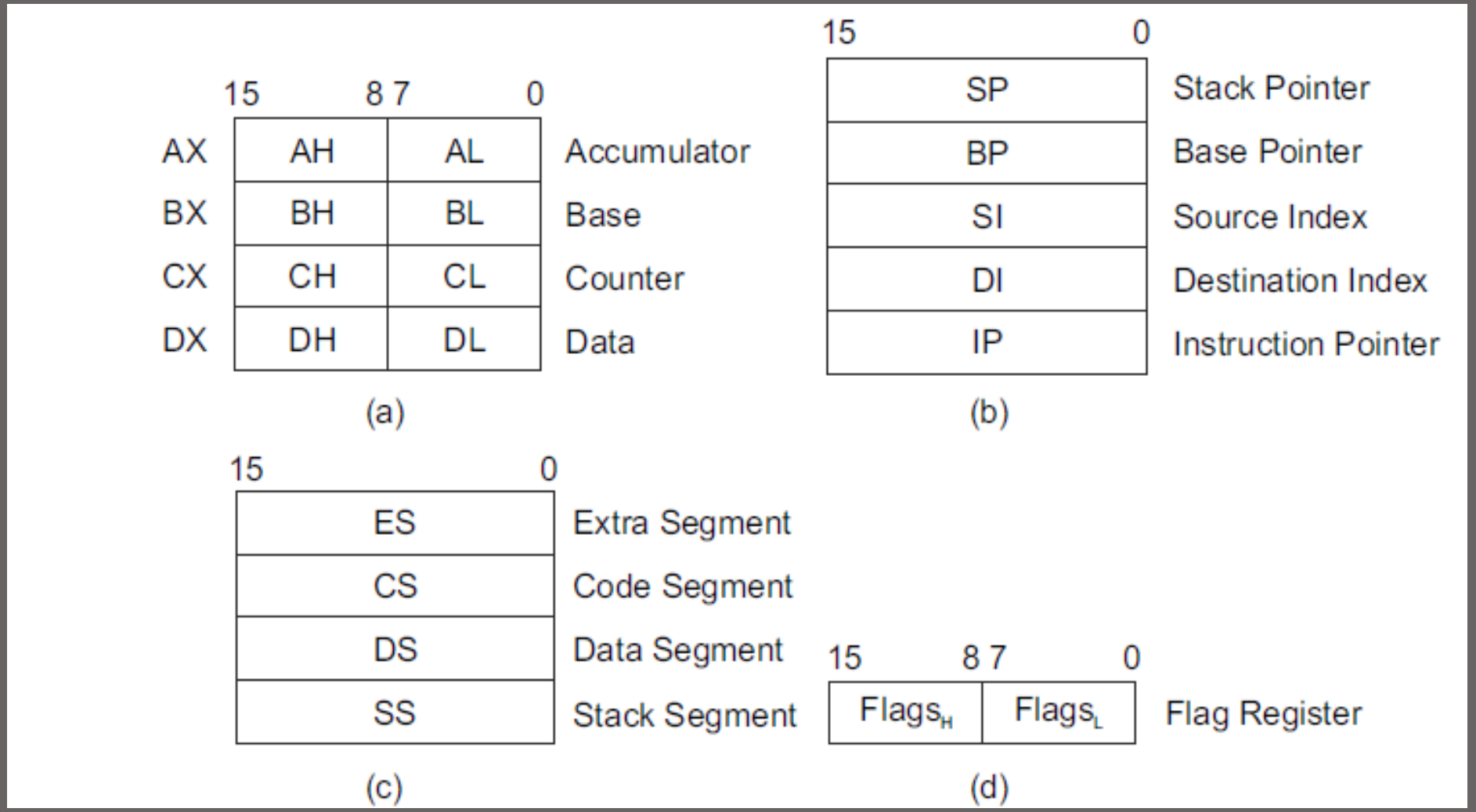
Exercice 01
On veut réaliser une mémoire de 1Ko (la taille d'un mot est de 16 bits) en utilisant des boîtiers de taille 1Ko mots de 4 bits) ?
Exercice 02

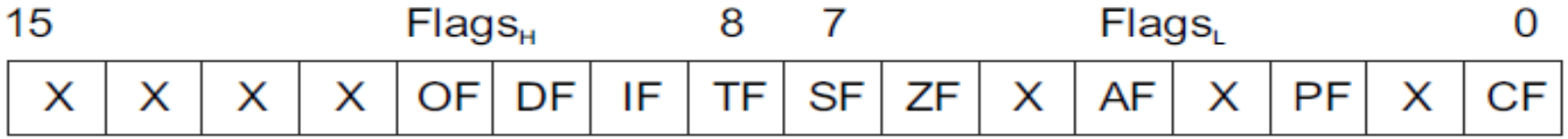
Réaliser une mémoire de 1Ko (la taille d'un mot est de 8 bits) en utilisant des boîtiers de taille 256 mots de 8 bits
Exercice 03
On veut réaliser une mémoire de 1KO (la taille d'un mot est de 8 bits) en utilisant des boîtiers de taille 256 mots de 4 bits) ?
Exercice 04
Soit la mémoire suivant :



Déterminer :
La taille de mémoire en mot
La taille de mémoire en bits


EU
(Execution Unit)
BIU
(Bus Interface Unit)
Address Bus/Status
A_{19}/S_6–A_{16}/S_3
Address/Data Bus
AD_{15}–AD_0
Address Generation and Bus Control
Σ
Memory Address Logic
6
5
4
3
2
1
Instruction Queue
General Registers
AX
BX
CX
DX
AH
AL
BH
BL
CH
CL
DH
DL
BP
DI
SI
SP
CS
ES
SS
DS
Instruction Pointer
Segment Registers
Internal Data Bus
Arithmetic Logic Unit
Flags
Control and Timing
TEST
INT
NMI
LOCK
QS_0–QS_1
$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$
$\overline{RQ}$ / $\overline{GT}_{0\text{-}1}$
HOLDA
HOLD
CLK
RESET
READY
MN/$\overline{MX}$
GND V_{CC}


15
8 7
0
AX
AH
AL
Accumulator
BX
BH
BL
Base
CX
CH
CL
Counter
DX
DH
DL
Data
(a)
15
0
SP
Stack Pointer
BP
Base Pointer
SI
Source Index
DI
Destination Index
IP
Instruction Pointer
(b)
15
0
ES
Extra Segment
CS
Code Segment
DS
Data Segment
SS
Stack Segment
(c)
15
8 7
0
Flags$_H$
Flags$_L$
Flag Register
(d)



CF – Carry flag
PF – Parity flag
AF – Auxilary cary flag
ZF – Zero flag
X – Underfined

SF – Sign flag
TF – Trap flag
IF – Interrupt flag
DF – Directional flag
OF – Overflow flag

Fig. 5.6 *Flag registers of 8086*

15				Flags$_H$				8	7			Flags$_L$			0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

CF – Carry flag
PF – Parity flag
AF – Auxilary cary flag
ZF – Zero flag
X – Underfined

SF – Sign flag
TF – Trap flag
IF – Interrupt flag
DF – Directional flag
OF – Overflow flag

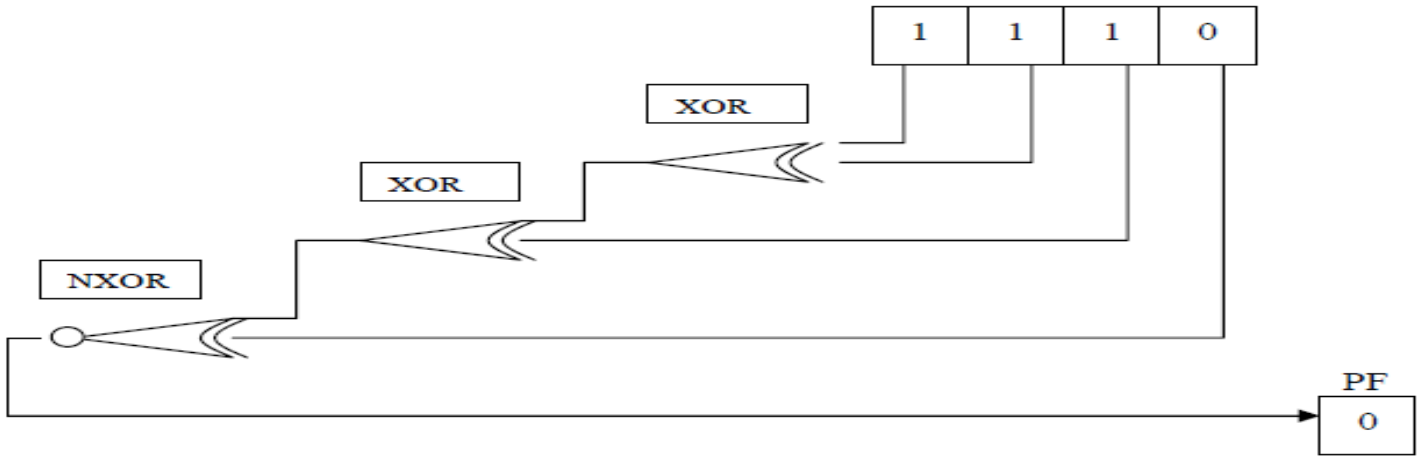
Comme vous pouvez le remarquer, c'est un registre 16 bits, chaque bit s'appelle un flag (ou un indicateur) et peut prendre la valeur **1** ou **0**.

CARRY flag : ce flag positionne à « 1 » lorsque il y a un débordement.

	1	0	0	1
+				
	1	0	0	0
=	0	0	0	0

CF
1

PARITY flag : la parité est celle du nombre de bits égaux à « 1 » s'il est pair PF est mis à « 1 ».



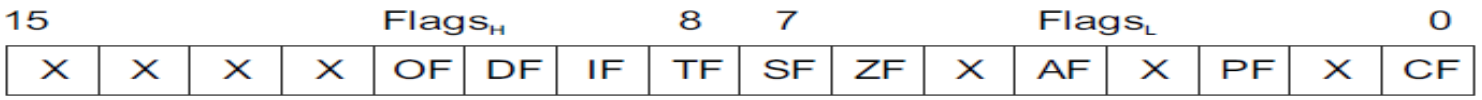
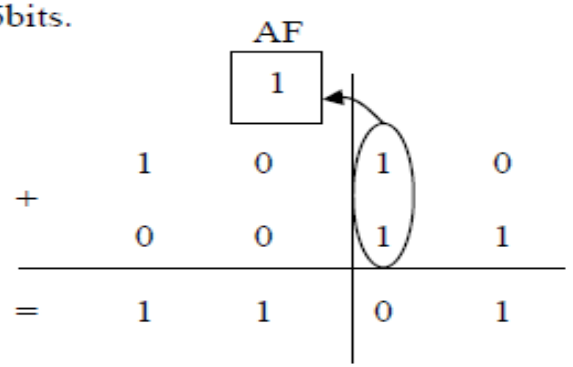
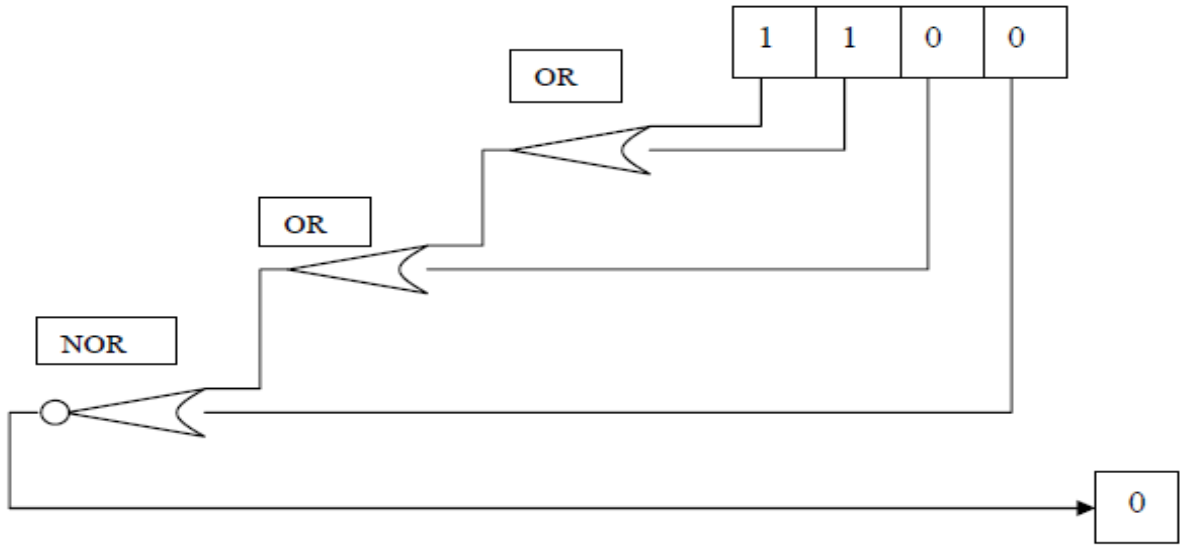


Fig. 5.6 *Flag registers of 8086*

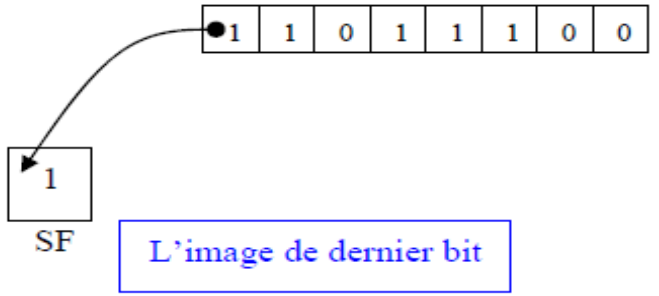
AUXILLIARY flag : ce flag est à « **1** »lorsqu'un **débordement** a lieu de demi-mot de 8bits ou 16bits.



ZERO flag : ce flag est à « 1 » lorsque tous les bits d'un mot sont à « 0 ».



SIGN flag : ce flag est à **1** lorsque le résultat est **négatif**. Lorsque le résultat est **positif**, le flag est à **0**. Ce flag prend la valeur du bit le plus fort (MSB).



OVERFLOW flag : ce flag est à « **1** » lorsqu'un **débordement signé.**

	0	1	1	1	(+7)
+					
	0	1	0	1	(+5)
=	1	1	0	0	(Résultat non signé = 12) (Résultat signé = - 4).

OF
1

TRACE flag : ce bit est à « 1 » signifie que l'exécution est en mode pas à pas (STEP BY STEP).

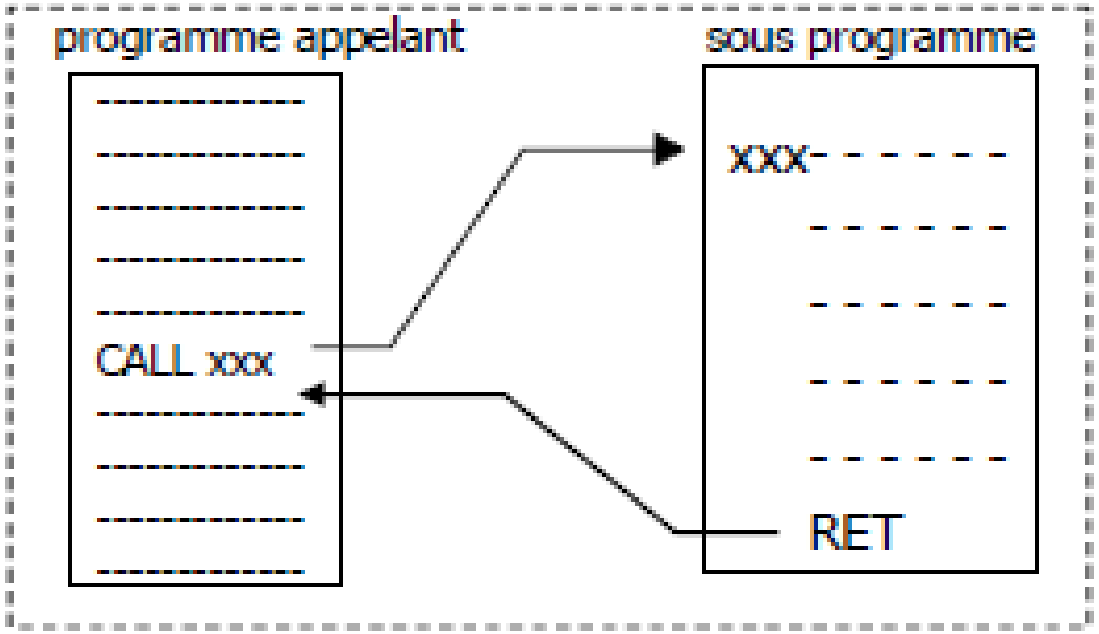
INTERUPT flag : si ce bits à « 1 » le µp prend en conte les interruptions vienne de la pin INTR (INTERRUPT REQUESTE) « les interruption hardware ».

DIRECTION flag : ce flag est utilisé par quelques instructions pour traiter les chaînes de données, Lorsque ce drapeau est placé à **0**, la chaîne est traitée octet par octet en **incrémentant**, lorsque ce drapeau est placé à **1**, la chaîne est traitée octet par octet en **décrémentant**.

Branchements inconditionnels

JMP xyz Provoque un saut sans condition à la ligne portant l'étiquette xyz.
CALL xyz Appel d'une procédure (sous programme) qui commence à la ligne xyz. La position de l'instruction suivant le CALL est empilée pour assurer une poursuite correcte après l'exécution du sous programme.

RET Retour de sous programme. L'exécution du programme continue à la position récupérée dans la pile.
INT n appel à l'interruption logicielle n° n



Branchements conditionnels

Les branchements conditionnels sont conditionnés par l'état des indicateurs (drapeaux) qui sont eux même positionnés par les instructions précédentes.

Dans la suite nous allons utiliser la terminologie :

- supérieur ou inférieur pour les nombres non signés

- plus petit ou plus grand pour les nombres signés

- + pour l'opérateur logique OU

JE/JZ xyz (Jump if Equal or Zero) Aller à la ligne xyz si résultat nul ou si égalité. C'est-à-dire si Z=1

JNE/JNZ xyz (Jump if Not Equal or Not Zero) Aller à la ligne xyz si résultat non nul ou si différent. C'est-à-dire si Z=0

JA xyz (Jump if Above) aller à la ligne xyz si supérieur (non signé). C'est-à-dire si C + Z = 0

JAE xyz (Jump if Above or Equal) aller à la ligne xyz si supérieur ou égal (non signé). C'est-à-dire si C = 0

JB xyz (Jump if Bellow) Branche si inférieur (non signé). C'est-à-dire si C = 1

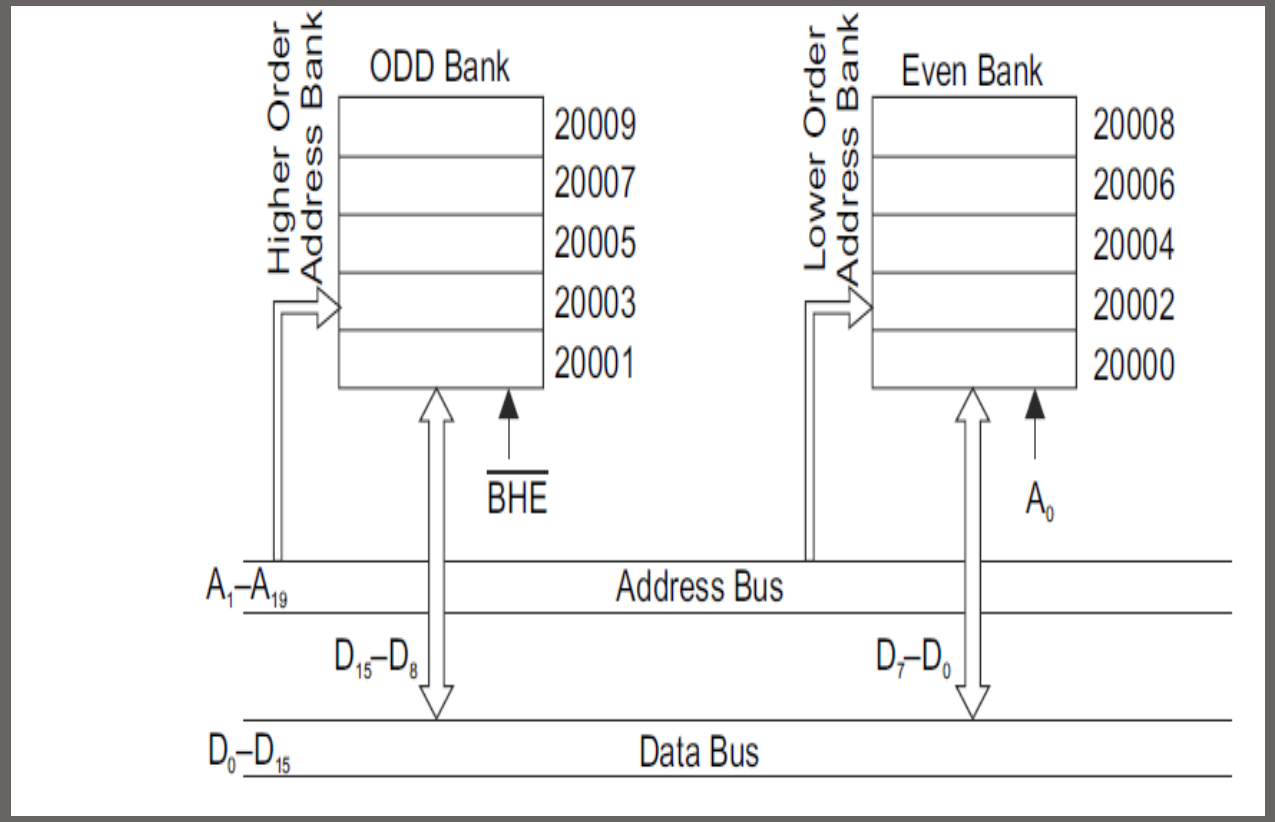
JBE xyz (Jump if Bellow or Equal) aller à la ligne xyz si inférieur ou égal (non signé). C'est-à-dire si C + Z = 1

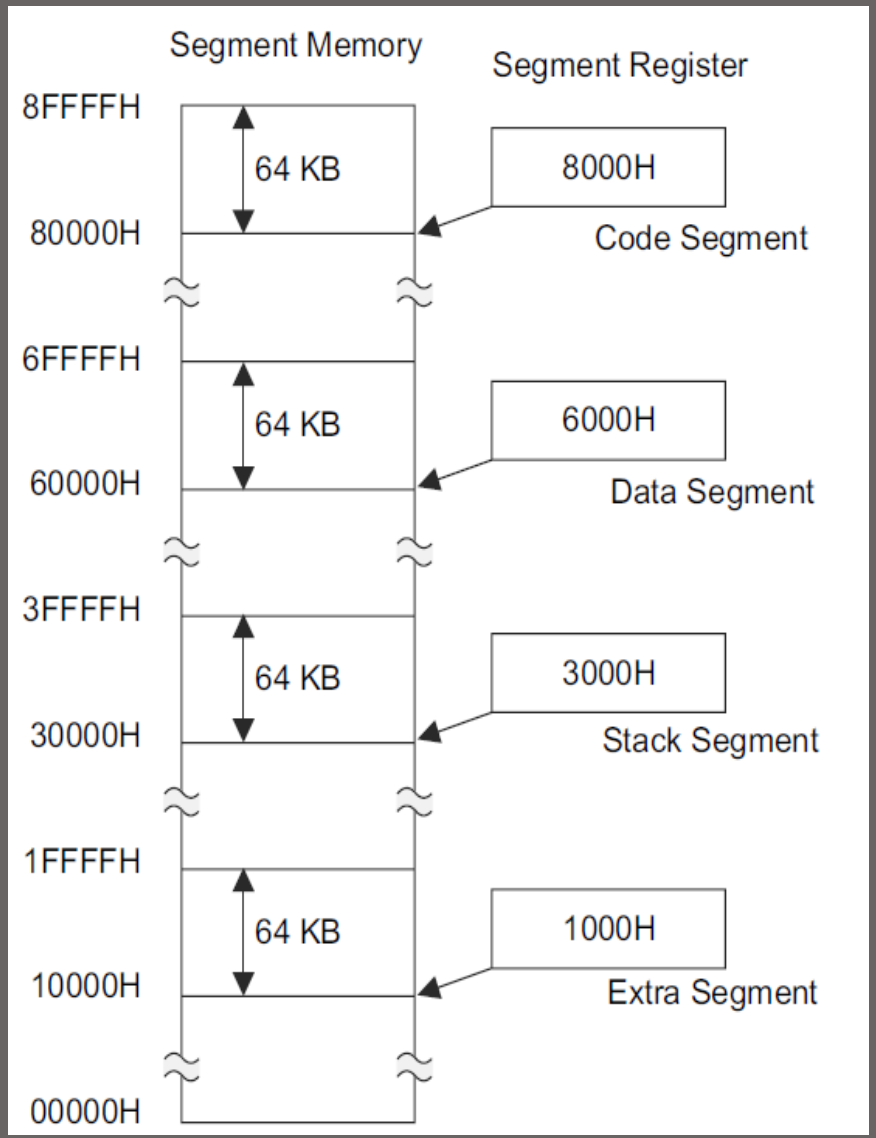
JC xyz (Jump if CArry) aller à la ligne xyz s'il y a retenu. C'est-à-dire si C = 1

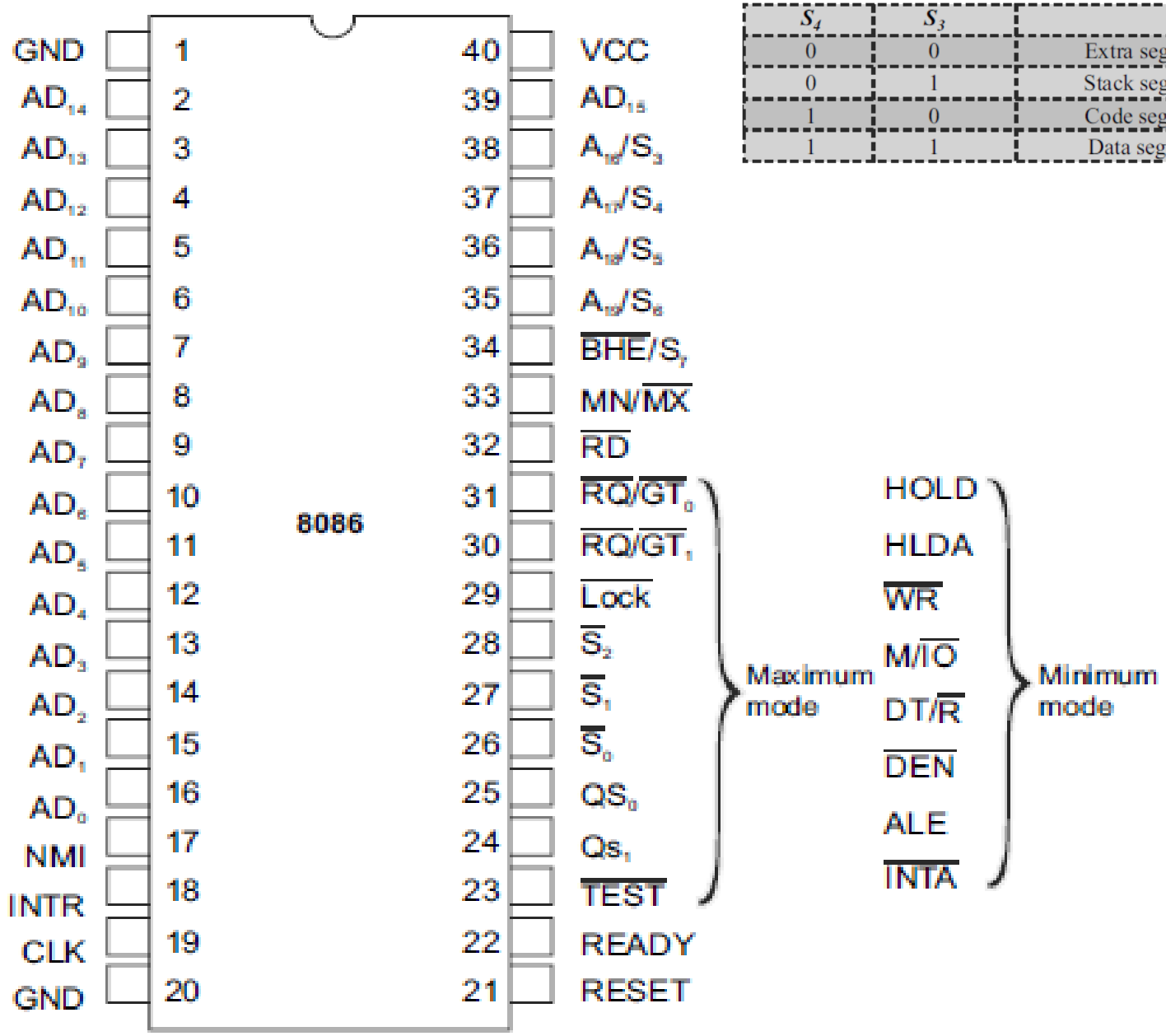
JNC xyz (Jump if No CArry) aller à la ligne xyz s'il n'y a pas de retenu. C'est-à-dire si C = 0
JG xyz (Jump if Grater) aller à la ligne xyz si plus grand (signé). C'est-à-dire si (S ⊕ O) + Z = 1
JGE xyz (Jump if Grater or Equal) aller à la ligne xyz si plus grand ou égal (signé). C'est-à-dire si S ⊕ O = 0
JL xyz (Jump if Less) aller à la ligne xyz si plus petit (signé). C'est-à-dire si S ⊕ O = 1
JLE xyz (Jump if Less or Equal) aller à la ligne xyz si plus petit ou égal (signé). C'est-à-dire si (S ⊕ O) + Z = 1
JO xyz (Jump if Overflow) aller à la ligne xyz si dépassement. C'est-à-dire si O = 1
JNO xyz (Jump if No Overflow) aller à la ligne xyz s'il n'y a pas de dépassement O = 0
JP/JPE xyz (Jump if Parity or Parity Even) aller à la ligne xyz si parité paire. C'est-àdire si P = 1
JNP/JPO xyz (Jump if No Parity or if Parity Odd) aller à la ligne xyz si parité impaire. C'est-à-dire si P = 0
JS xyz (Jump if Sign) aller à la ligne xyz si signe négatif. C'est-à-dire si S = 1
JNS xyz (Jump if No Sign) aller à la ligne xyz si signe positif. C'est-à-dire si S = 0

Table 6.1 *Addressing modes of 8086/8088 microprocessors*

Addressing mode	*Mnemonic*	*Symbolic Operation*	*Destination of Operand*	*Source of Operand*	*Functions*
Immediate addressing mode	MOV AX, 2000H	AH←20H; AL←00	AX register	Data 2000	Source of data is within instruction
Register addressing mode	MOV AX, BX	AX←BX	AX register	BX register	Source and destination of data are registers of microprocessors
Direct addressing mode	MOV AH, [0400]	AH←[0400H]	AH register	0400H = Displacement	Memory address is available within the instruction.
Register indirect addressing mode	MOV AX, [SI]	AL←[SI]; AH←[SI + 1]	AX register	SI + DS × 10 = memory location	Memory address is supplied in any index or pointer registers.
Indexed addressing mode	MOV AX , [SI + 06]	AL←[SI + 6]; AH←[SI + 7]	AX register	[SI + 06] + DS × 10 = memory location	Memory address is the sum of the indexed register and a displacement within the instruction.
Based addressing mode	MOV AX, [BP]	AL← [BP] ; AH←[BP + 1].	AX register	BP + DS × 10 = memory location	Memory address is the content of BX or BP register within instruction.
Based and indexed addressing mode	MOV [BX + SI], AX.	[BX + SI]←AL; [BX + SI + 1] ←AH.	BX + SI + DS × 10 = memory location	AX register	Memory address is the sum of an index register and a base register.
Based and indexed with displacement addressing mode	MOV AX, [BX + SI + 10]	AL←[BX + SI + 10]: AH←[BX + SI + 11]	AX register	[BX + SI + 10] + DS × 10 = memory location	Memory address is the sum of an index register, a base register and a displacement within instruction.


Higher Order Address Bank
ODD Bank
20009
20007
20005
20003
20001
BHE
Lower Order Address Bank
Even Bank
20008
20006
20004
20002
20000
A0
A1–A19
Address Bus
D15–D8
D7–D0
D0–D15
Data Bus


Segment Memory
Segment Register
8FFFFH
64 KB
8000H
80000H
Code Segment
6FFFFH
64 KB
6000H
60000H
Data Segment
3FFFFH
64 KB
3000H
30000H
Stack Segment
1FFFFH
64 KB
1000H
10000H
Extra Segment
00000H



S_4	S_3	*Function*
0	0	Extra segment memory access
0	1	Stack segment memory access
1	0	Code segment memory access
1	1	Data segment memory access

Fig. 5.15 *Pin diagram of 8086*

II. Description et utilisation des signaux du 8086 :

CLK : entrée du signal d'horloge qui cadence le fonctionnement du microprocesseur. Ce

signal provient d'un **géné**rateur d'**Horloge** : le 8284.

RESET : entrée de remise à zéro du microprocesseur. Lorsque cette entrée est mise à l'état haut pendant au moins 4 périodes d'horloge, le microprocesseur est réinitialisé : il va exécuter l'instruction se trouvant à l'adresse FFFF0H (adresse de bootstrap). Le signal de RESET est fourni par le générateur d'horloge.

READY : entrée de synchronisation avec la mémoire. Ce signal provient également du générateur d'horloge.

TEST : entrée de mise en attente du microprocesseur d'un événement extérieur.

MN/MX : entrée de choix du mode de fonctionnement du microprocesseur :

- Mode minimum (MN/MX = 1) : le 8086 fonctionne de manière autonome, il génère lui-même le bus de commande (RD, WR, ...) ;
- Mode maximum (MN/MX=0) : Ces signaux de commande sont produits par un **contrôleur de bus**, le 8288. Ce mode permet de réaliser des systèmes multiprocesseurs

.

NMI et **INTR** : entrées de demande d'interruption. INTR : interruption normale, NMI (Non Maskable Interrupt) : interruption prioritaire.

INTA : Interrupt Acknowledge, indique que le microprocesseur accepte l'interruption.

HOLD et **HLDA** : signaux de demande d'accord d'accès direct à la mémoire (DMA).

S0 à **S7** : signaux d'état indiquant le type d'opération en cours sur le bus.

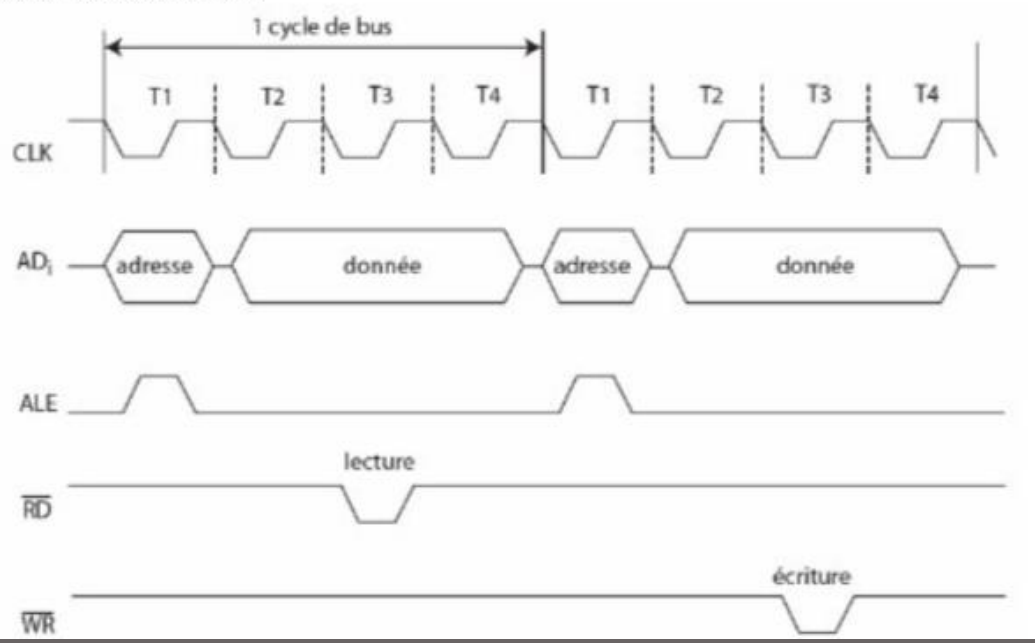
A16/S3 à **A19/S6** : 4 bits de poids fort du bus d'adresses, **multiplexés** avec 4 bits d'état.

AD0 à **AD15** : 16 bits de poids faible du bus d'adresses, **multiplexés** avec 16 bits de données.

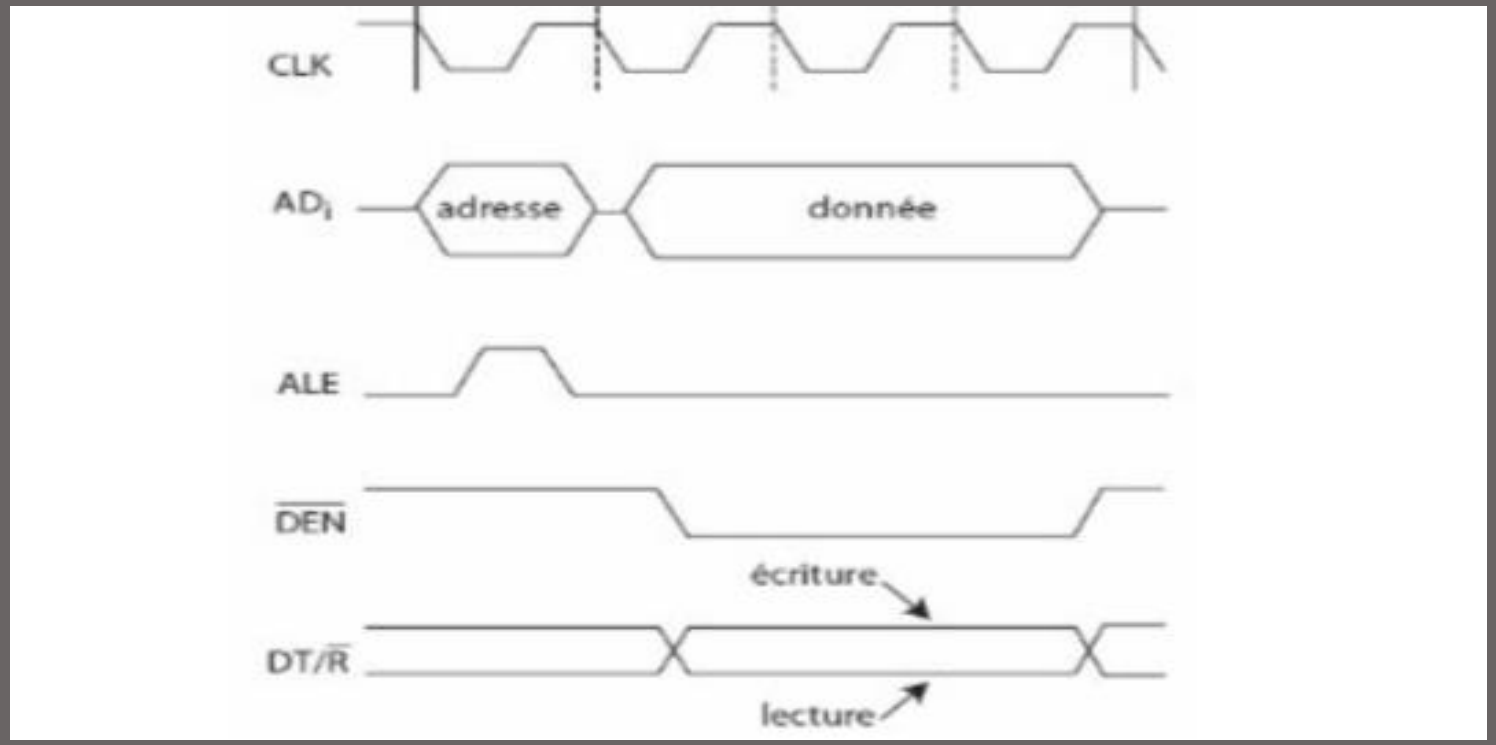
Le bus A/D est multiplexé (multiplexage temporel) d'où la nécessité d'un **démultiplexage** pour obtenir séparément les bus d'adresses et de données :

- 16 bits de données (microprocesseur 16 bits) ;

Chronogramme du bus A/D :



- **si ALE = 1, le verrou est transparent (Q = D) ;**
- **si ALE = 0, mémorisation de la dernière valeur de D sur les sorties Q; ⍰ les signaux de lecture (RD) ou d'écriture (WR) ne sont générés par le microprocesseur que lorsque les données sont présentes sur le bus A/D.**



RD : Read, signal de lecture d’une donnée.
WR : Write, signal d’écriture d’une donnée.
M/IO : Memory/Input-Output, indique si le 8086 adresse la mémoire (M/IO = 1) ou les entrées/sorties (M/IO = 0).
DEN : Data Enable, indique que des données sont en train de circuler sur le bus A/D (équivalent de ALE pour les données).
DT/R : Data Transmit/Receive, indique le sens de transfert des données :
DT/R = 1 : données émises par le microprocesseur (écriture) ;
DT/R = 0 : données reçues par le microprocesseur (lecture).

HE : Bus High Enable, signal de lecture de l'octet de poids fort du bus de données. Le 8086 possède un bus d'adresses sur 20 bits, d'où la capacité d'adressage de 1 Mo ou 512 Kmots de 16 bits (bus de données sur 16 bits). Le méga-octet adressable est divisé en deux banques de 512 Ko chacune : la banque inférieure (ou paire) et la banque supérieure (ou impaire). Ces deux banques sont sélectionnées par :

- **A 0 pour la banque paire qui contient les octets de poids faible ;**
- **BHE pour la banque impaire qui contient les octets de poids fort.**

Table 5.4 *Selection of proper byte from even-addressed and odd-addressed memory banks for processing*

$\overline{BHE}$	A_0	*Processing*
0	0	Both banks active. 16-bit data transfer, 16-bit word transfer on AD_{15}–AD_0
0	1	Only high bank active, one byte transfer on AD_{15}–AD_8
1	0	Only low bank active, one byte transfer on AD_7–AD_0
1	1	No bank active

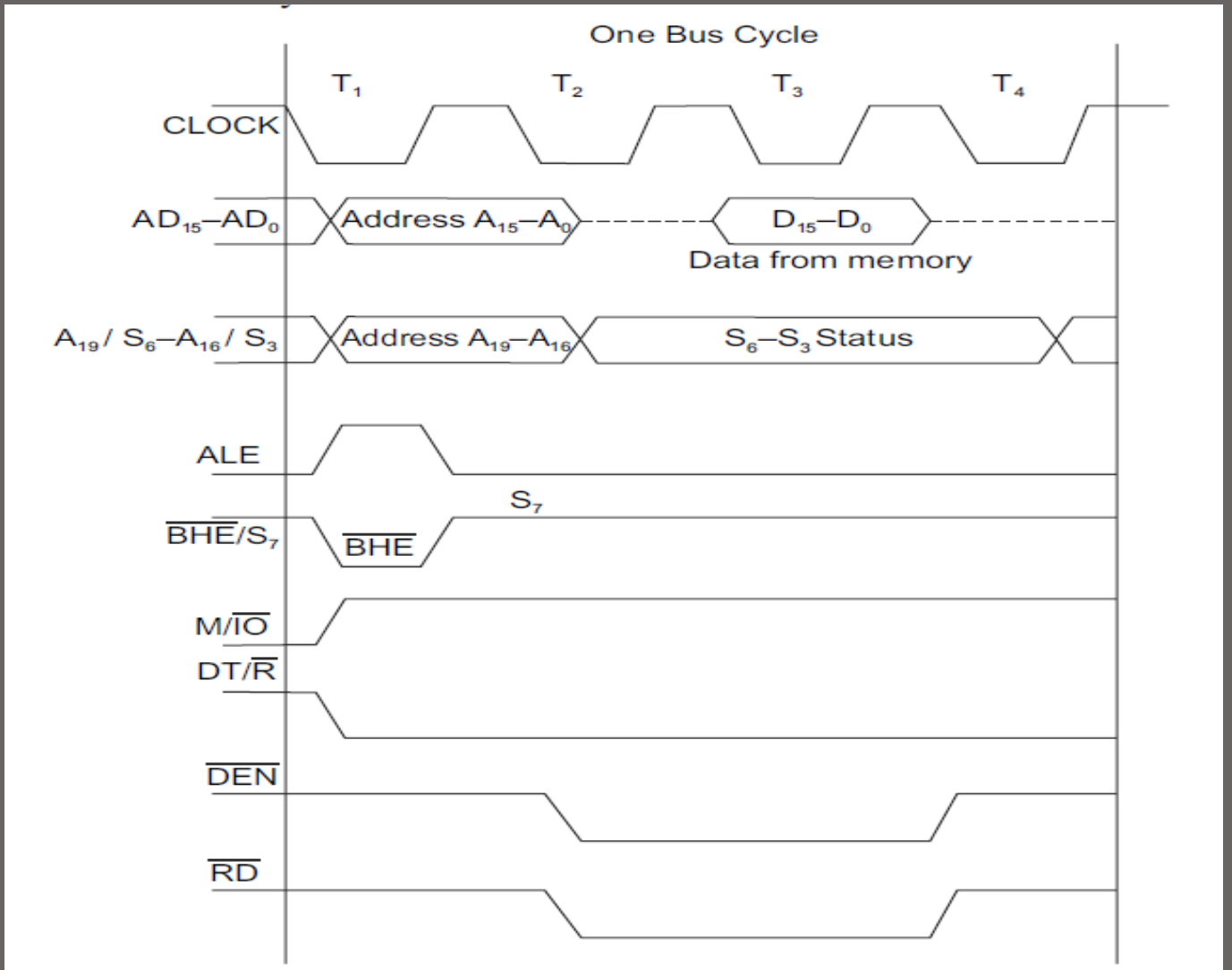


Fig. 5.19 *Memory read bus cycle of 8086 for minimum mode*

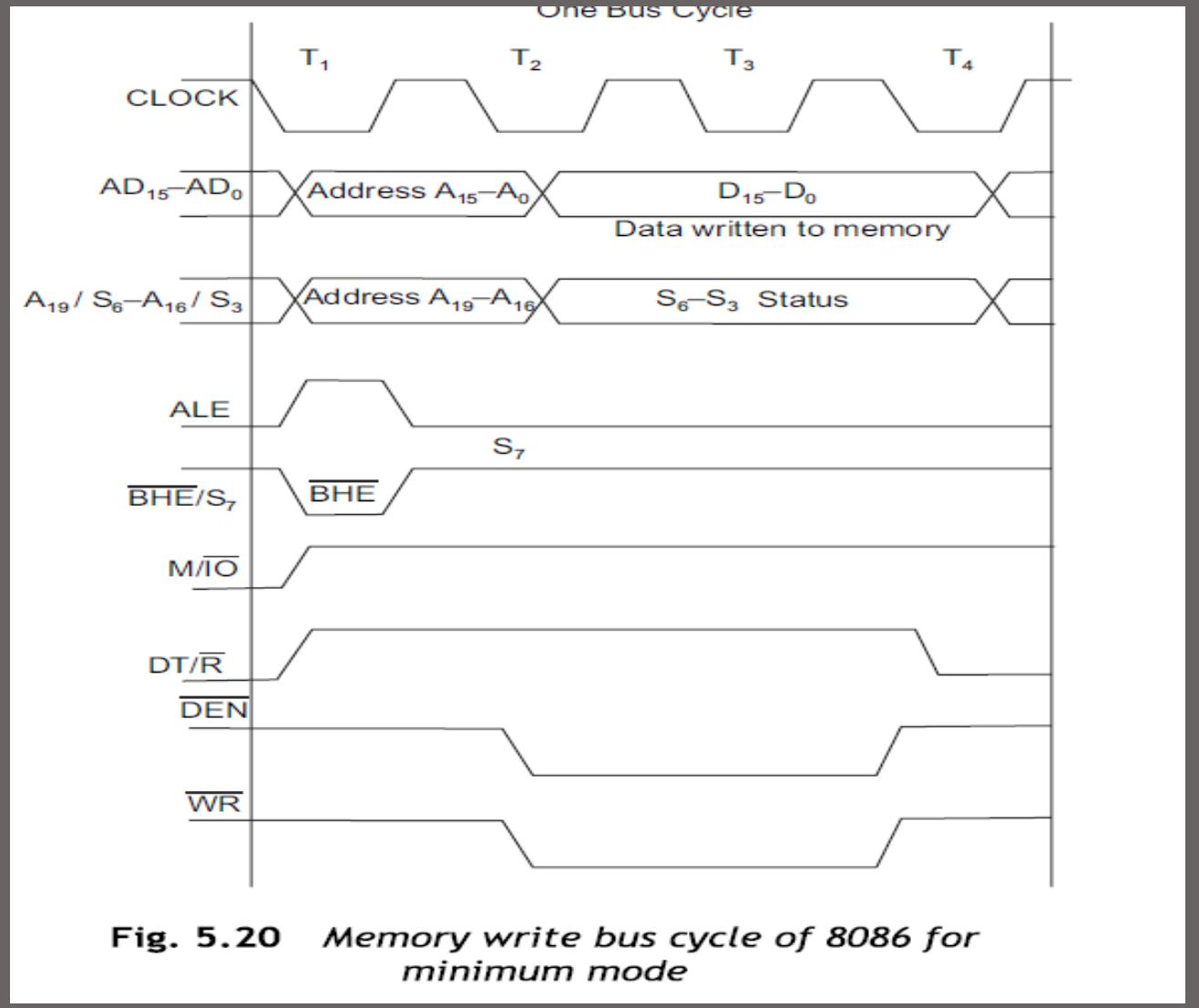
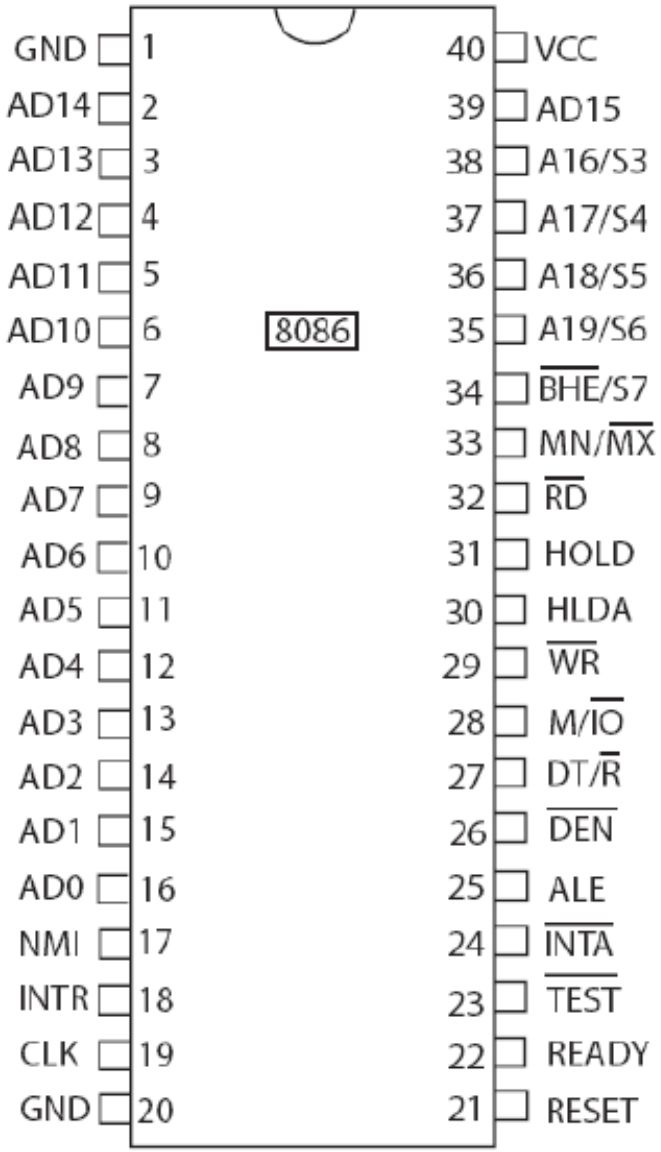


Fig. 5.20 *Memory write bus cycle of 8086 for minimum mode*



Boîtier DIP du microprocesseur 8086

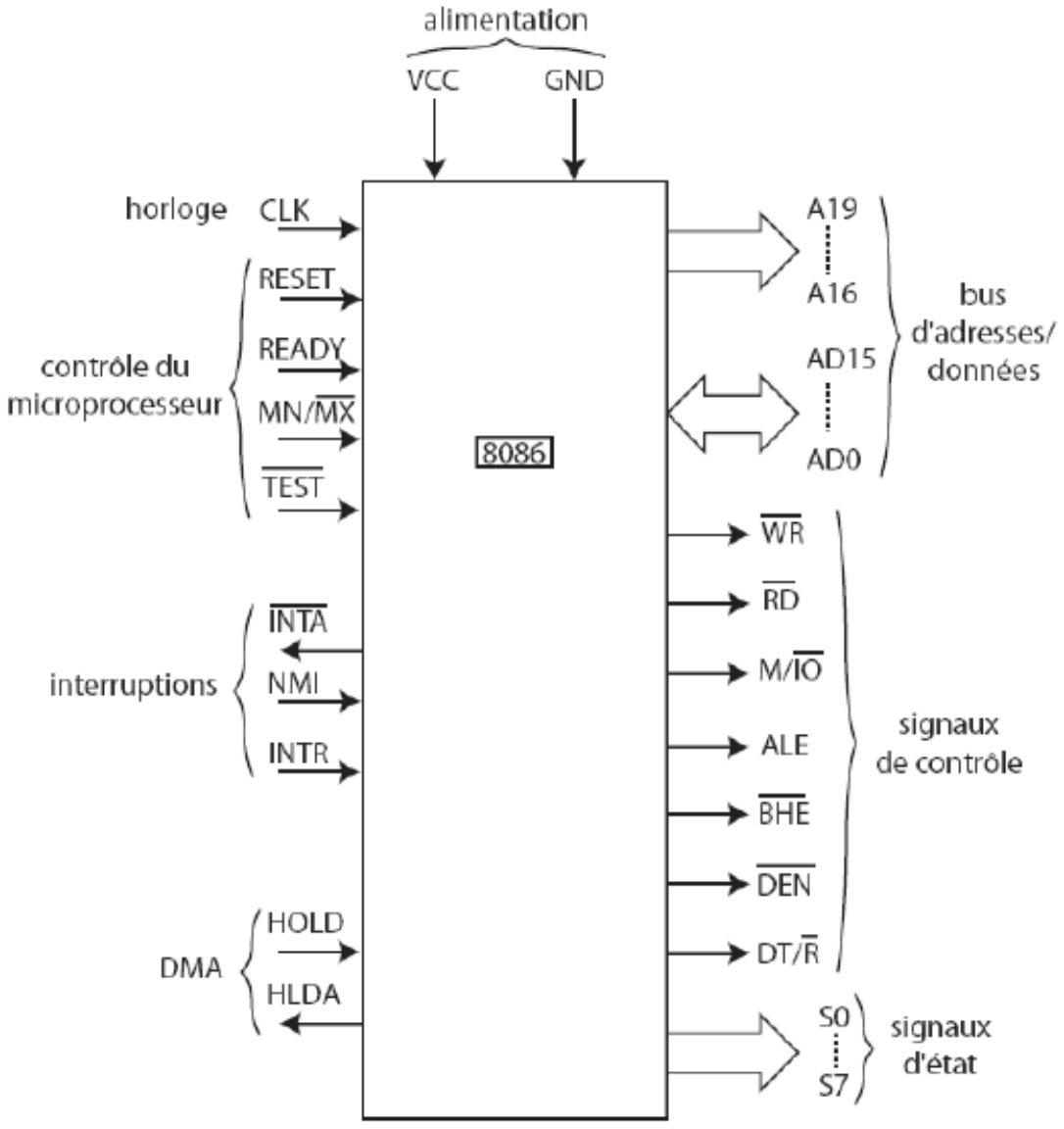


Schéma fonctionnel du 8086

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

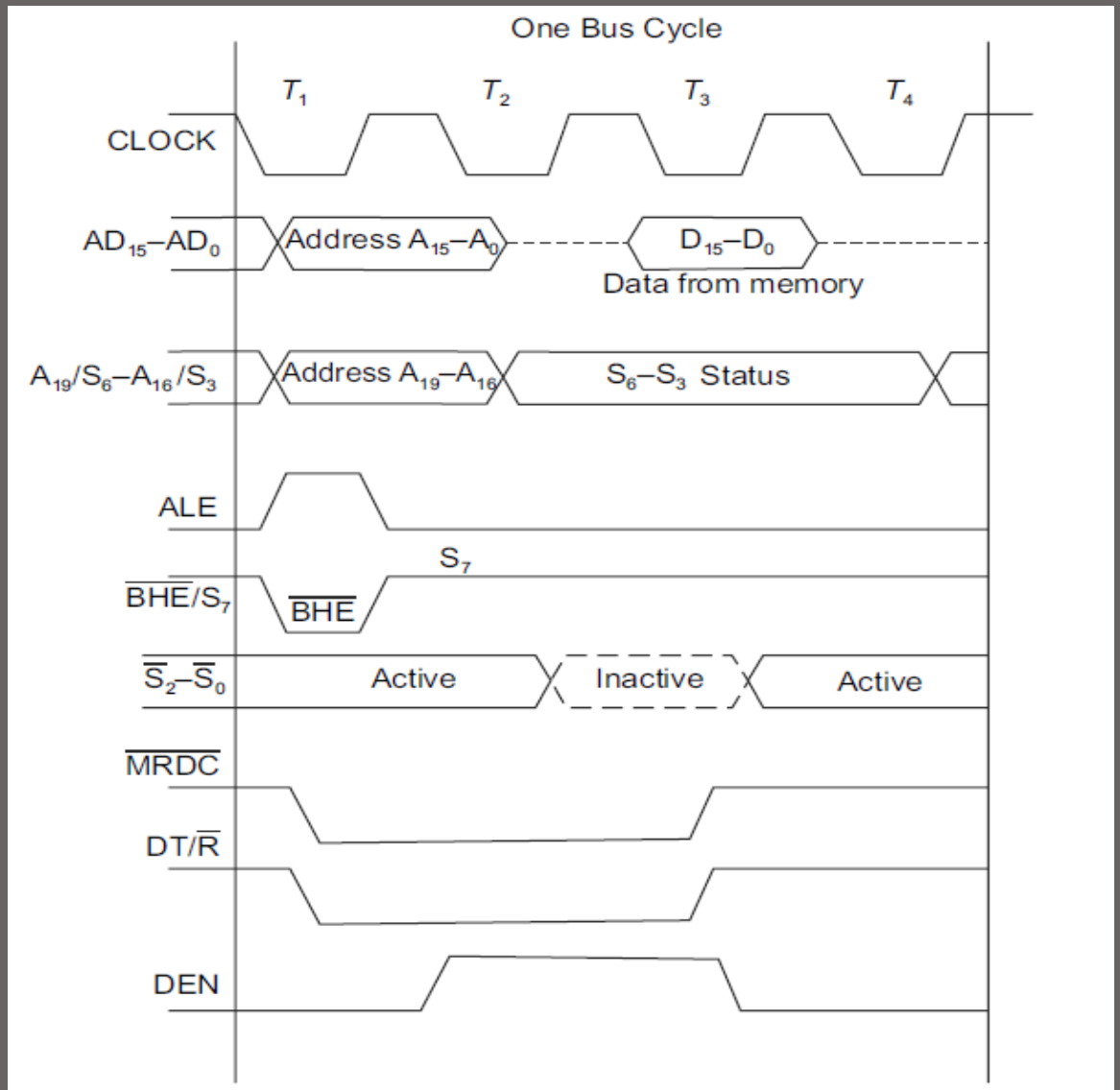


Fig. 5.22 *Memory read bus cycle of 8086 for maximum mode*

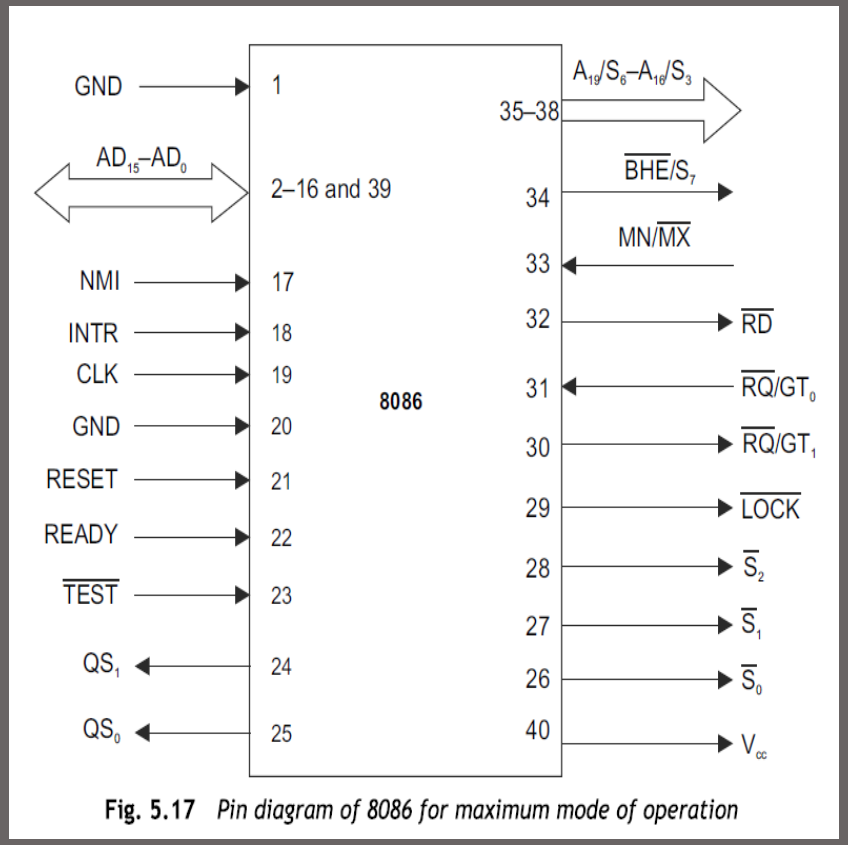


Fig. 5.17 *Pin diagram of 8086 for maximum mode of operation*

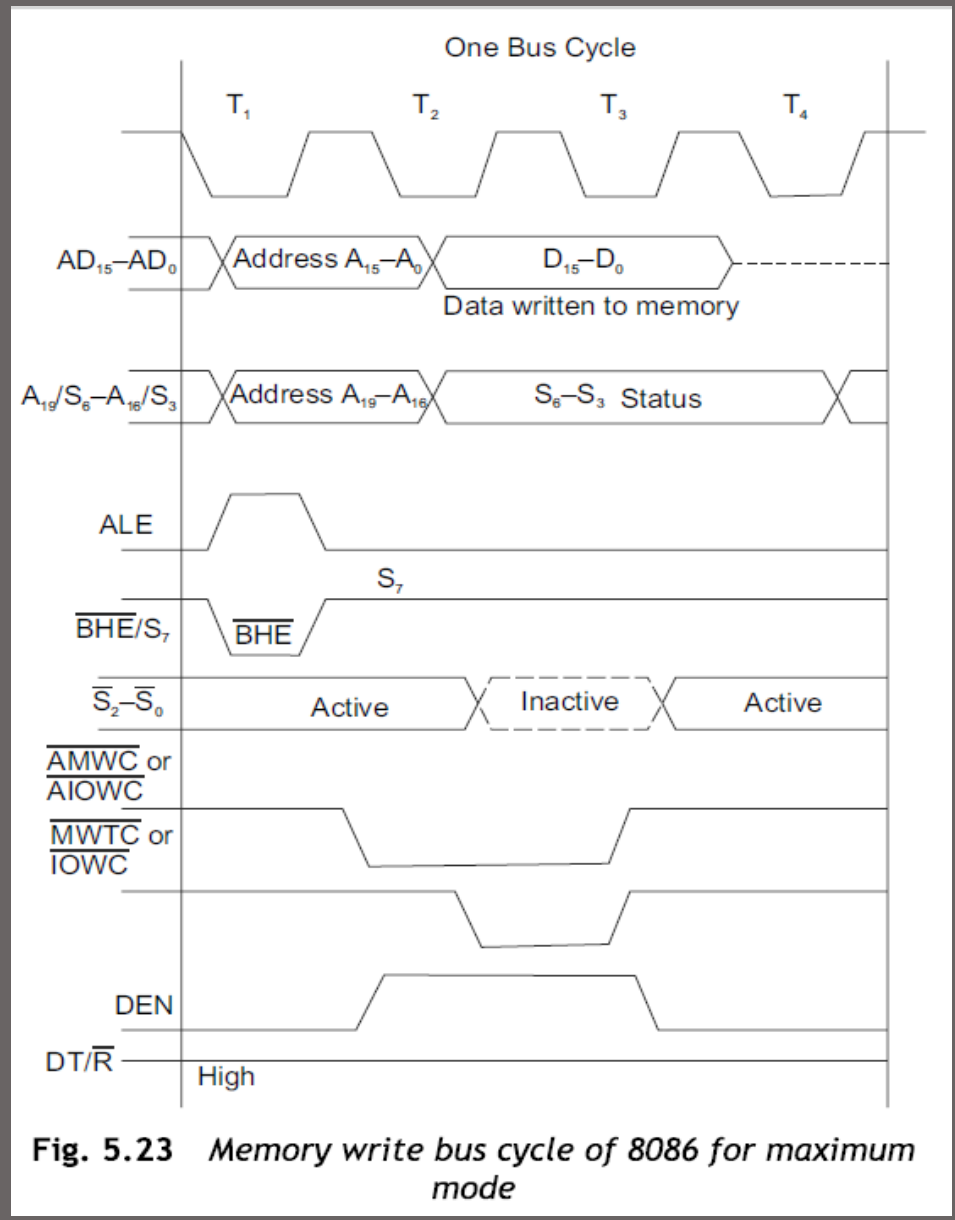


Fig. 5.23 *Memory write bus cycle of 8086 for maximum mode*

Table 5.9 *Logic for status signals $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$*

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	*Function*
0	0	0	Interrupt acknowledge
0	0	1	Read data from I/O port
0	1	0	Write data into I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

QS_1,	QS_0	Indication
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

Table 1.4

IV.6.1. Décalage logique vers la droite (Shift Right) :
SHR opérande,n
Cette instruction décale l'opérande de n positions vers la droite.
Exemple 01 :
mov al,11001011B
shr al,1

entrée d'un 0 à la place du bit de poids fort ; le bit sortant passe à travers l'indicateur de retenue CF.
Remarque : si le nombre de bits à décaler est supérieur à 1, ce nombre doit être placé dans le registre CL ou CX.
Exemple 02 : décalage de AL de trois positions vers la droite :
mov cl,3
shr al,cl

IV.6.2. Décalage logique vers la gauche (Shift Left) :
SHL opérande,n
Cette instruction décale l'opérande de n positions vers la droite.
Exemple :
mov al,11001011B
shl al,1

entrée d'un 0 à la place du bit de poids faible ; le bit sortant passe à travers l'indicateur de retenue CF.
Même remarque que précédemment si le nombre de positions à décaler est supérieur à 1.

IV.6.3. Décalage arithmétique vers la droite :
SAR opérande,n
Ce décalage conserve le bit de signe bien que celui-ci soit décalé.
Exemple :
mov al,11001011B
sar al,1

le bit de signe est **réinjecté**.

IV.6.4. Décalage arithmétique vers la gauche :
SAR opérande,n
Identique au décalage logique vers la gauche.

IV.6.5. Applications des instructions de décalage : ➢ **Cadrage à droite d'un groupe de bits.**
Exemple : on veut avoir la valeur du quartet de poids fort du registre AL :

```
mov al,11001011B
mov cl,4
shr al,cl
```

→ AL = 0000**1100**B ➢ **Test de l'état d'un bit dans un mot.**
Exemple : on veut déterminer l'état du bit 5 de AL :

```
mov cl,6
shr al,cl
ou
mov cl,3
shl al,cl
```

IV.6.6. Rotation à droite (Rotate Right) :
ROR opérande,n
Cette instruction décale l'opérande de n positions vers la droite et réinjecte par la gauche les
bits sortant.
Exemple :

```
mov al,11001011B
ror al,1
```

réinjection du bit sortant qui est copié dans l'indicateur de retenue CF.

IV.6.7. Rotation à gauche (Rotate Left) :
ROL opérande,n
Cette instruction décale l'opérande de n positions vers la gauche et réinjecte par la droite les
bits sortant.

Exemple :
mov al,11001011B
rol al,1

IV.6.8. Rotation à droite avec passage par l'indicateur de retenue (Rotate Right through Carry)
RCR opérande,n
Cette instruction décale l'opérande de n positions vers la droite en passant par l'indicateur de retenue CF.
Exemple :
mov al,11001011B
rcr al,1

le bit sortant par la droite est copié dans l'indicateur de retenue CF et la valeur précédente de CF est réinjectée par la gauche.

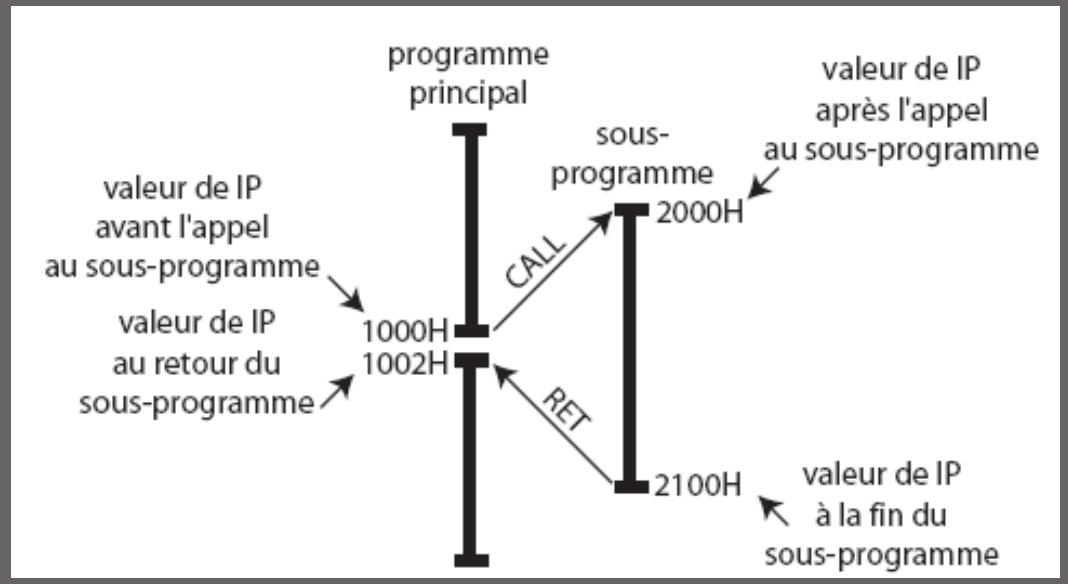
IV.6.9. Rotation à gauche avec passage par l'indicateur de retenue (Rotate Left through Carry)
:
RCL opérande,n
Cette instruction décale l'opérande de n positions vers la gauche en passant par l'indicateur de retenue CF.
Exemple :
mov al,11001011B
rcl al,1

le bit sortant par la gauche est copié dans l'indicateur de retenue CF et la valeur précédente de CF est réinjectée par la droite.

VI.2. Appel d'un sous-programme par le programme principal : CALL procédure

Lors de l'exécution de l'instruction CALL, le pointeur d'instruction IP est chargé avec l'adresse de la première instruction du sous-programme. Lors du retour au programme appelant, l'instruction suivant le CALL doit être exécutée, c'est-à-dire que IP doit être rechargé avec l'adresse de cette instruction.

```
      ⋮        ← instructions précédant l'appel au sous-programme
call nom_sp    ← appel au sous-programme
      ⋮        ← instructions exécutées après le retour au programme principal
```



La déclaration d'une procédure

Etant donné qu'une procédure est une suite d'instructions, il s'agit de regrouper les instructions composant la procédure entre des mots clés. L'ensemble de cette manipulation est appelée *déclaration de procédure*.
Ces mots clés permettant la déclaration de la procédure sont *une étiquette* (qui représente le nom de la fonction) précédant le mot clé *PROC* marquant le début de la procédure, suivi de *near* (qui signale que la procédure est située dans le même segment que le programme appelant) et *RET* désignant la dernière instruction, et enfin le mot clé *ENDP* qui annonce la fin de la procédure. Ainsi une déclaration de procédure ressemble à ceci :

```
LABEL PROC NEAR
INSTRUCTION1
INSTRUCTION2
RET
LABEL ENDP
```

Appel d'une procédure

C'est l'instruction *CALL* qui permet l'appel d'une procédure. Elle est suivie soit d'une adresse 16 bits, désignant la position du début de la procédure, ou bien du nom de la procédure (celui de l'étiquette qui précède le mot clé *PROC*).

Comment l'appel et la fin de la procédure fonctionnent ?

Lorsqu'on appelle une procédure, la première adresse de la procédure est stockée dans le registre IP (pointeur d'instruction). Le processeur traite ensuite toutes les lignes d'instructions précédant le mot clé *RET*.

Finalement, l'adresse stockée avant l'appel de *PROC* est remise dans le registre IP.

Cela paraît simple mais le problème provient du fait que les procédures peuvent être imbriquées, c'est-à-dire que de saut en saut, le processeur doit être capable de revenir successivement aux adresses de retour. En fait, à chaque appel de fonction de l'instruction *CALL*, le processeur empile l'adresse contenue dans le registre IP (il pointe alors sur l'instruction suivant l'instruction *CALL*) avant de la modifier. A l'appel de l'instruction *RET* (qui ne prend pas d'arguments) le contenu de la pile est dépilé puis stocké dans le registre IP.

Le passage de paramètres

Une procédure effectue généralement des actions sur des données qu'on lui fournit, toutefois dans la déclaration de la procédure il n'y a pas de paramètres (dans des langages évolués on place généralement les noms des variables paramètres entre des parenthèses, séparés par des virgules). Il existe toutefois deux façons de passer des paramètres à une procédure :

- **Le passage des paramètres par registre** : on stocke les valeurs dans les registres utilisés dans la procédure
- **Le passage des paramètres par pile** : on stocke les valeurs dans la pile avant d'appeler la procédure, puis on lit le contenu de la pile dans la procédure

Le passage de paramètres par registre