Chapitre 2 : Architecture Interne du Microprocesseur Intel 8086

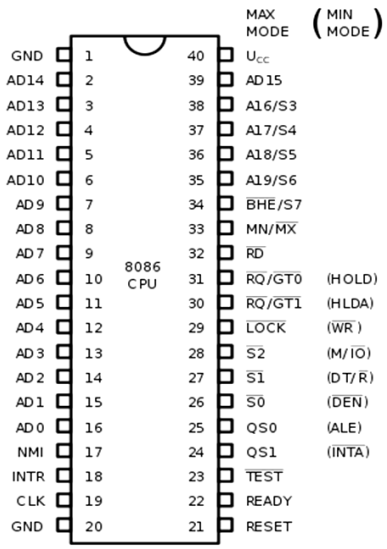
2.1 Introduction

L'Intel 8086 est un microprocesseur **CISC** 16 bits fabriqué par Intel. Il fut lancé en mai 1978. C'est le premier processeur de la famille x86, qui est devenue l'architecture de processeur la plus répandue dans le monde des ordinateurs personnels, stations de travail et serveurs informatiques.

Le processeur 8086 d'Intel est à la base des processeurs Pentium. Les processeurs successifs (de PC) se sont en effet construits petit à petit en ajoutant à chaque processeurs des instructions et des fonctionnalités supplémentaires, mais en conservant à chaque fois les spécificités du processeur précédent. C'est cette façon d'adapter les processeurs à chaque étape qui permet à un ancien programme écrit pour un 8086 de fonctionner sur un nouvel ordinateur équipé d'un Pentium 4.

2.2 Aspect externe du microprocesseur 8086

Il se présente sous la forme d'un boîtier DIP (Dual In-line Package) à 40 broches (figure 2.1).



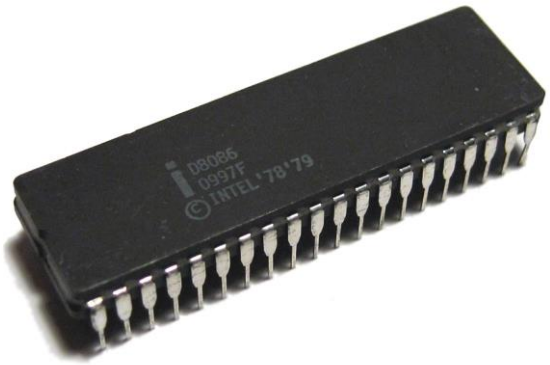


Figure 2.1. Brochage du microprocesseur 8086 intel 8086 en boitier DIP-40 *

***DIP** : Dual In-line Package et 40 : pour designer 40 broches.

2.3 Caractéristiques du 8086

Il est basé sur des registres 16 bits, et dispose d'un bus de données de 16 bits et d'un bus d'adresse de 20 bits, qui lui permet d'adresser 1 Mo. Il contient 29 000 transistors gravés en 3 µm. Sa puissance de calcul varie de 0,33 MIPS (lorsqu'il est cadencé à 4,77 MHz comme dans l'IBM PC) jusqu'à 0,75 MIPS pour la version 10 MHz. Le microprocesseur i8086 se caractérise par :

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

Définition : Un **registre** est une petite partie de mémoire intégrée au microprocesseur, dans le but de recevoir des informations, notamment des adresses et des données stockées durant l'exécution d'un programme.

2.4 Architecture interne du 8086

Le microprocesseur 8086 a été partitionné en deux unités fonctionnelles séparées : l'unité d'interface de bus **(BIU)** et l'unité d'exécution **(EU)**.

- La principale raison de cette séparation est d'assurer un fonctionnment simultané des deux unités, d'où une augmentation de la vitesse de traitement du processeur (fonctionnement selon le principe du pipeline).
- La **BIU** doit interagir avec la mémoire et les périphériques d'entrée/sortie pour récupérer les instructions et les données requises par l'**UE**.
- L'**UE** est responsable de l'exécution des instructions des programmes et du traitement requis.

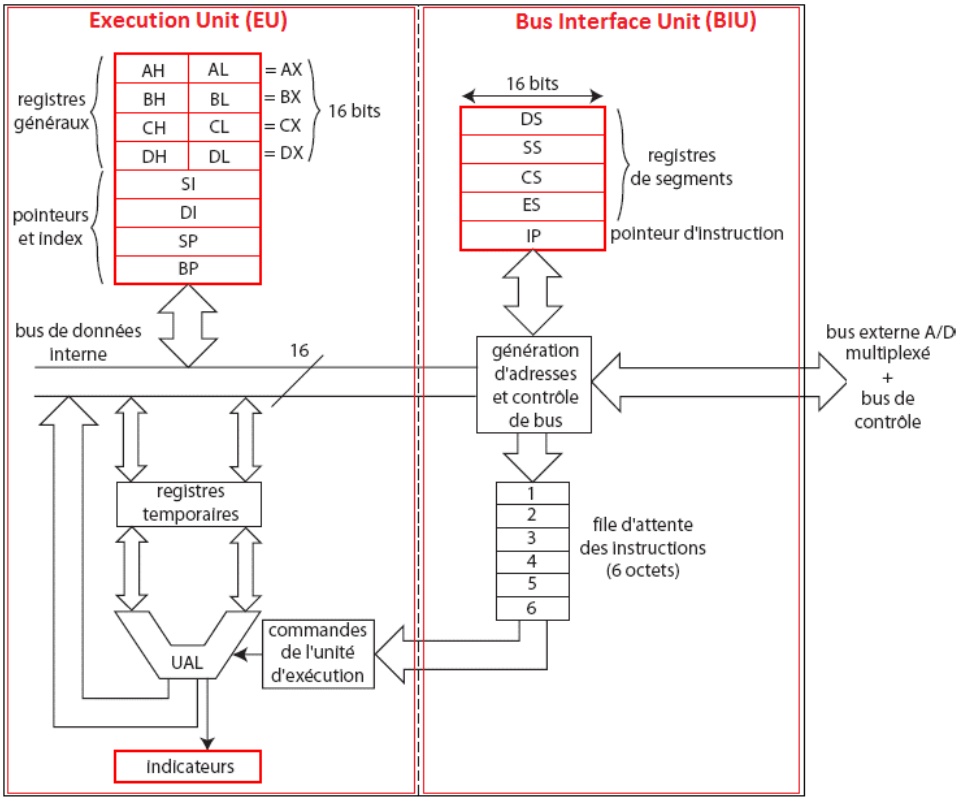


Figure 2.2. Architecture interne du 8086.

2.5 Description de la BIU (Bus Interface Unite)

La **BIU** comporte une file d'attente d'instructions gérée en **FIFO** (**F**irst **I**n **F**irst **O**ut), les registres de segments (**CS, DS, SS, ES**) et le pointeur d'instruction (**IP : I**nstruction **P**ointer). Le rôle de la BIU est de :

- Chercher les instructions à exécuter dans la mémoire et les stocke dans la file d'attente FIFO.
- Calculer les adresses physiques sur 20 bits.
- Réaliser le transfert des données avec la mémoire.

2.5.1 Les registres de segment

Le 8086 a quatre registres de segment de **16 bits** chacun : **CS** (code segment), **DS** (Data segment), **ES** (Extra segment) et **SS** (Stack segment), ces registres sont chargés de sélectionner les différents segments de la mémoire en pointant sur le début de chacun d'entre eux. Chaque segment de mémoire ne peut excéder les

65536 = 2^{16} octets (**64 Ko**).

a) Le registre CS (Code Segment) : Il pointe sur le segment qui contient les codes des instructions du programme en cours.

b) Le registre DS (Data Segment) : Le registre segment de données pointe sur le segment des variables globales du programme, bien évidemment la taille ne peut excéder 64 K octets.

c) Le registre ES (Extra Segment) : Le registre de données supplémentaires **ES** est utilisé par le microprocesseur lorsque l'accès aux autres registres est devenu difficile ou impossible pour modifier des données, de même ce segment est utilisé pour le stockage des chaînes de caractères.

d) Le segment SS (Stack Segment) : Le registre **SS** pointe sur la pile : la pile est une zone mémoire ou on peut sauvegarder les registres ou les adresses ou les données pour les récupérer après l'exécution d'un sous-programme ou l'exécution d'un programme d'interruption.

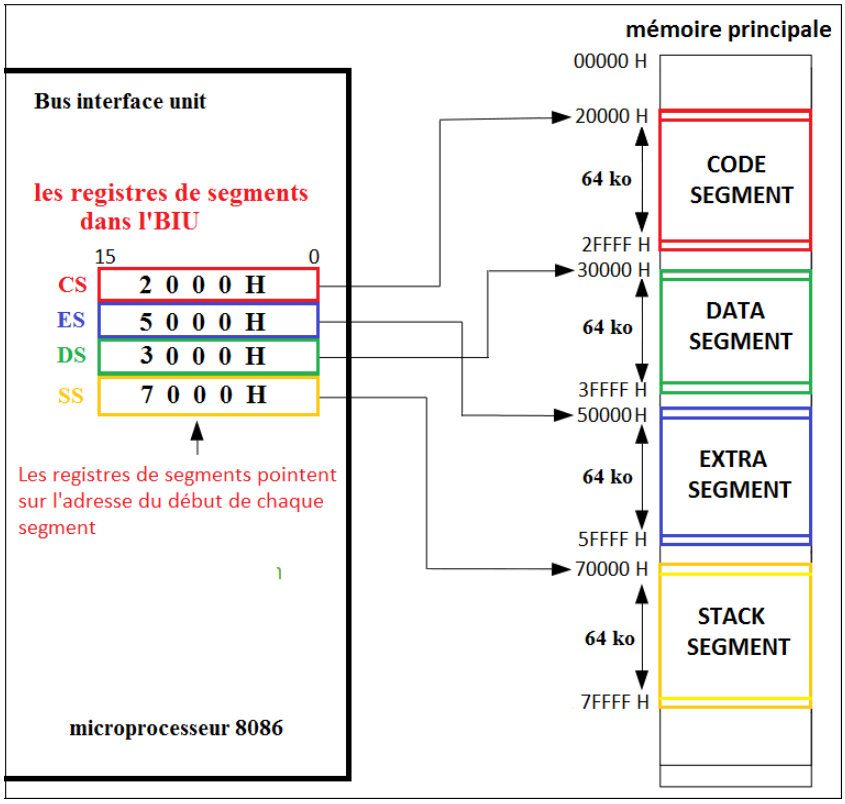


Figure 2.3. Segmentation de la mémoire

2.5.2 Le registre IP : (Instruction pointer)

Appelé aussi **compteur de programme**, il contient l'adresse de l'emplacement mémoire où se situe la prochaine instruction à exécuter. Le registre **IP** est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.

2.6 Description de l'EU (Exécution Unité)

L'**EU** comporte l'**ALU (A**rithmetic and **L**ogic **U**nit**)**, les registres généraux (**AX, BX, CX, DX**), les registres d'adressage (**SP, BP, SI, DI**), le registre d'état (**Flags :** drapeaux en français) et le décodeur d'instructions. Le rôle de la EU est de :

- extraire les codes des instructions à partir de la file d'attente située dans la BIU et les exécute.
- fournir les adresses des opérandes à l'UIB en nommant le segment concerné et en fournissant le déplacement dans ce segment.

2.6.1 l'Unité Arithmétique et Logique (UAL) : est un circuit complexe qui assure les fonctions logiques (ET, OU, Comparaison, Décalage, etc…) ou arithmétique (Addition, soustraction).

2.6.2 Les registres généraux

Les registres généraux peuvent être utilisés dans toutes les opérations arithmétiques et logiques que le programmeur insère dans le code assembleur. Un registre complet présente une grandeur de **16 bits**. Comme le montre la figure 2.4, chaque registre est en réalité divisé en deux registres distincts de **8 bits**. De cette façon, nous pouvons utiliser une partie du registre si nous désirons y stocker une valeur n'excédant pas 8 bits.
Le programmeur dispose de 8 registres internes de 16 bits qu'on peut diviser en deux groupes comme le montre la figure ci-dessous.

Figure 2.4. Registres généraux du 8086.

a. groupe de données :

Ce groupe est formé par 4 registres de 16 bits (**AX, BX, CX,** et **DX**) chaque registre peut être divisé en deux registres de 8 bits (**AH, AL, BH, BL ,CH, CL, DH** et **DL**).

Le registre AX (Accumulateur) : Toutes les opérations de transfert de données avec les entrées-sorties ainsi que le traitement des chaînes de caractères se font dans ce registre, de même les opérations arithmétiques et logiques. Les conversions en BCD du résultat d'une opération arithmétique (addition, soustraction, multiplication et division) se font dans ce registre.

Le registre BX (registre de base) : Il est utilisé pour l'adressage de données, en général il contient une adresse de décalage par rapport à une adresse de référence (segment de données DS). De plus il peut servir pour la conversion d'un code à un autre.

Le registre CX (Le compteur) : Lors de l'exécution d'une boucle on a souvent recours à un compteur de boucles pour compter le nombre d'itérations, le registre CX a été fait pour servir comme compteur lors des instructions de bouclage.

Le registre DX : il est utilisé pour les opérations de multiplication et de division mais surtout pour contenir le numéro d'un port d'entrée/sortie pour adresser les interfaces d'E/S.

b. groupe de pointeurs et index :

Ces registres sont plus spécialement adaptés au traitement des éléments dans la mémoire. Ils sont en général munis de propriétés d'incrémentation et de décrémentation.

L'index SI (source index) : Il permet de pointer la mémoire il forme en général un décalage (un offset) par rapport à une base fixe (le registre **DS**), il sert aussi pour les instructions de chaîne de caractères, en effet il pointe sur le caractère source.

L'index DI (Destination index) : Il permet aussi de pointer la mémoire, il présente un décalage par rapport à une base fixe (**DS** ou **ES**), il sert aussi pour les instructions de chaîne de caractères, il pointe alors sur la destination.

Les pointeurs SP et BP (Stack pointer et base pointer) :

Ils pointent sur la zone pile (une zone mémoire gérée en LIFO), ils présentent un décalage par rapport à la base (le registre **SS**). Pour le registre **BP**, il a un rôle proche de celui de **BX**, mais il est généralement utilisé avec le segment de pile.
Le pointeur de pile **SP** (**Stack Pointer**) permet de pointer la pile pour stocker des données ou des adresses selon le principe du **LIFO** (Last In First Out).

2.6.3 Le registre d'état (indicateur)

Le registre d'état sert à contenir l'état de certaines opérations effectuées par le processeur. Par exemple, le drapeau de signe **SF** (sign flag) est positionné à **1** si la dernière opération a générée un résultat négatif, à **0** s'il est positif ou nul.

Remarque : Les drapeaux sont des indicateurs qui annoncent une condition particulière suite à une opération arithmétique ou logique.

Le registre d'état du 8086 est formé par les bits suivants :

Figure 2.5. Le registre d'état (Program Status Word).

× : pour désigner un bit non utilisé.

CF (Carry Flag : drapeau de Retenue) : cet indicateur est positionné à **1** lorsqu'il y a une retenue du résultat 8 ou 16 bits. Il intervient dans les opérations d'additions (retenue) et de soustractions (emprunt) sur des entiers naturels.
CF = 1 s'il y a une retenue après l'addition ou la soustraction du bit de poids fort des opérandes.

PF (Parity Flag : drapeau de Parité) : si le résultat de l'opération contient un nombre pair de 1, cet indicateur est positionné à **1**, sinon **0**.

AF (Auxiliary Carry : Demie retenue**) :** ce bit est égal à **1** si on a une retenue du quarter (4 bits) de poids faible dans le quarter de poids plus fort.

ZF (Zero Flag : drapeau de Zéro) : Cet indicateur est positionné à **1** quand le résultat d'une opération est égal à zéro. Lorsque l'on vient d'effectuer une soustraction (ou une comparaison), ZF **= 1** indique que les deux opérandes étaient égaux. Sinon, **ZF** est mis à **0**.

SF (Sign Flag : drapeau de Signe) : Cet indicateur est positionné à 1 si la dernière opération a générée un résultat négatif, à 0 s'il est positif ou nul. **SF** est utile lorsque l'on manipule des entiers signés, car le bit de poids fort donne alors le signe du résultat.

OF (Overflow Flag : drapeau de Débordement) : si on a un débordement arithmétique ce bit est positionné à **1**, c'est à dire si le résultat d'une opération excède la capacité de l'opérande (registre ou case mémoire), sinon il est à **0**.

DF (Direction Flag : Auto Incrémentation/Décrémentation) : utilisé par les instructions de traitement des chaînes de caractères pour auto incrémenter ou auto décrémenter le **SI** et le **DI**.

IF (Interrupt Flag : Masque d'interruption) : pour masquer les interruptions venant de l'extérieur ce bit est mis à **0**, dans le cas contraire (IF = 1) le microprocesseur reconnaît l'interruption de l'extérieur.

TF (Trap Flag : Piège) : pour que le microprocesseur exécute le programme pas à pas.

Remarques :
Chaque drapeau est manipulé individuellement par des instructions spécifiques.

Les bits **IF**, **DF** et **TF** sont des indicateurs de contrôle qui permettent de modifier le comportement du microprocesseur. Ils sont positionnés par le programmeur.

2.7 Définition et principe de fonctionnement de la pile (stack segment) :

Définition : Une pile et un ensemble d'emplacement mémoire réservé à un empilement de données ça caractéristiques essentielles et que le premier élément introduit dans la pile est toujours au fond et que le dernier élément introduit est toujours au sommet (**l**ast **I**n **F**irst **O**ut). La pile est **organisée par mot de 16 bits**.

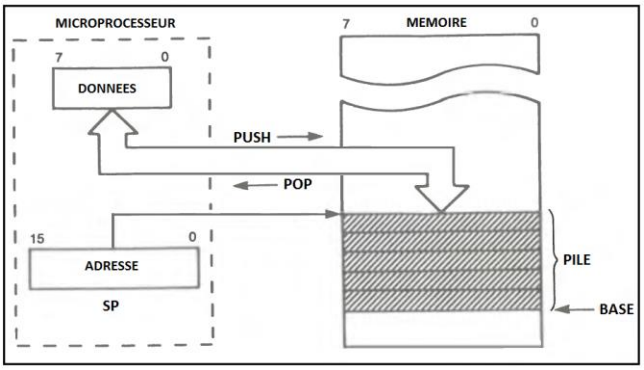


Figure 2.6. Schéma des deux instructions de manipulation de la pile : PUSH et POP.

Fonctionnement : En utilisation normal on peut accéder à la pile par deux instructions, ces deux instructions sont **PUSH** (empiler) et **POP** (dépiler) représentés par la figure ci-dessus.

SP (stack pointer) est le registre qui pointe vers le sommet de la pile, c'est-à-dire sur le dernier mot occupé de la pile.

L'instruction PUSH sauvegarde (déposer) un élément (16 bits) au sommet de la pile, elle fonctionne comme suit **:**

1) **SP ← SP – 2**
2) **SS:[SP] ← operand** (16 bits)

SS:[SP] : adresse du sommet de la pile

Instruction	Opérande (16 bits)	Description de l'instruction	Indicateurs
PUSH	**Reg** **SReg** **Mem**	**Store 16 bits operand in the stack Segment.** Cette instruction permet d'empiler (stocker) un opérande 16 bits dans la pile.	**N'affecte pas les drapeau**

SReg : pour désigner un registre de segment c'est-à-dire CS, DS, ES et SS

L'instruction POP fournie (transfère) le mot mémoire du sommet de la pile à l'opérande de destination, elle fonctionne comme suit **:**

1) **Operand** (16 bits) **← SS:[SP]**
2) **SP ← SP + 2**

Instruction	Opérande (16 bits)	Description de l'instruction	Indicateurs
POP	**Reg** **SReg** **Mem**	**Get 16 bits operand from the stack Segment.** Cette instruction fournie une valeur 16 bits depuis la pile et la sauvegarder dans l'opérande.	**N'affecte pas les drapeau**

Exemple :

PUSH AX ; empilage du registre AX ...
PUSH BX ; empilage du registre BX ...
PUSH [1100H] ; empilage et de la case mémoire 1100H-1101H
POP [1100H] ; dépilage dans l'ordre inverse de I 'empilage
POP BX
POP AX

Remarque : la valeur de SP doit être initialisée par le programme principal avant de pouvoir utiliser la pile.

Le tableau ci-dessous définit les instructions de sauvegarde et de restitution des registres dans le segment de pile (stack segment)

Instruction (sans opérande)	Description de l'instruction	Indicateurs
PUSHF	**Store flags register in the stack Segment.** Cette instruction permet d'empiler (stocker) le registre d'état (drapeau des indicateurs) dans la pile.	**[SP] ← Registre d'état** **SP ← SP - 2**
POPF	**Get flags register from the stack.** Cette instruction permet de désempiler de la pile le registre 16 bits de drapeaux contenant les indicateurs d'état.	**Registre d'état ← [SP]** **SP ← SP +2**
PUSHA	**Push all general purpose registers AX, CX, DX, BX, SP, BP, SI, DI in the stack.** Cette instruction empile le contenu des registres AX, CX, DX, BX, SP, BP, SI et DI avant l'appel de cette instruction dans la pile.	**PUSH AX** **PUSH CX** **PUSH DX** **PUSH BX** **PUSH SP** **PUSH BP** **PUSH SI** **PUSH DI**
POPA	**Pop all general purpose registers DI, SI, BP, SP, BX, DX, CX, AX from the stack.** permet de désempiler de la pile respectivement les registres DI, SI, BP, SP, BX, DX, CX et AX.	**POP DI** **POP SI** **POP BP** **POP xx (valeur de SP ignoré)** **POP BX** **POP DX** **POP CX** **POP AX**

2.8 Cycle général d'exécution d'une instruction

Le traitement d'une instruction peut être décomposé en quatre phases.

- **Phase 1 :** Recherche de l'instruction à traiter
- **Phase 2 :** Décodage de l'instruction et recherche de l'opérande
- **Phase 3 :** Exécution de l'instruction
- **Phase 4 :** Sauvegarde du résultat

2.8.1 Recherche d'une instruction :

Le contenu du registre **IP** (pointeur d'instruction) est déposé sur le bus d'adresses et envoyer vers la mémoire. Le bus de commande produit ensuite un **signal de lecture** en mémoire. Quand la mémoire reçoit le signal de lecture, les données qu'elle contient à l'adresse spécifiée sont déposées sur le bus de données. Le microprocesseur lit alors l'information sur le bus de données et la dépose dans un registre interne appelé registre d'instruction (**IR** : **I**nstruction **R**egister). L'information lue dans le microprocesseur est une instruction. On peut dire alors que l'instruction a été recherchée en mémoire. La figure 2.7 montre un schéma de ce processus.

Un **incrementateur** est relié au pointeur d'instruction (**IP)**, il est utilisé pour indiquer l'adresse de la prochaine instruction à exécuter. Les instructions sont cherchées séquentiellement lors de l'exécution d'un programme.

2.8.2 Décodage

Une fois que l'instruction se trouve dans le registre **IR**, l'**BIU** la décode et produit la séquence appropriée de signaux interne et externe permettant son exécution. Il faut un certain délai, en principe un cycle d'horloge, pour que le microprocesseur décode une instruction et décide de l'action à correspondre.

2.8.3 Exécution

Dès que la CPU à décoder l'instruction, elle effectue une série de tâches dictée par l'instruction. Certaines

instructions comportent peu de tâches par rapport à d'autres, leur temps d'exécution est donc court. On exprime généralement la **vitesse d'exécution** d'une instruction par le nombre total de cycle horloge requis par son exécution. Le nombre de cycles dépend de **la complexité** de l'instruction et aussi du **mode d'adressage**.

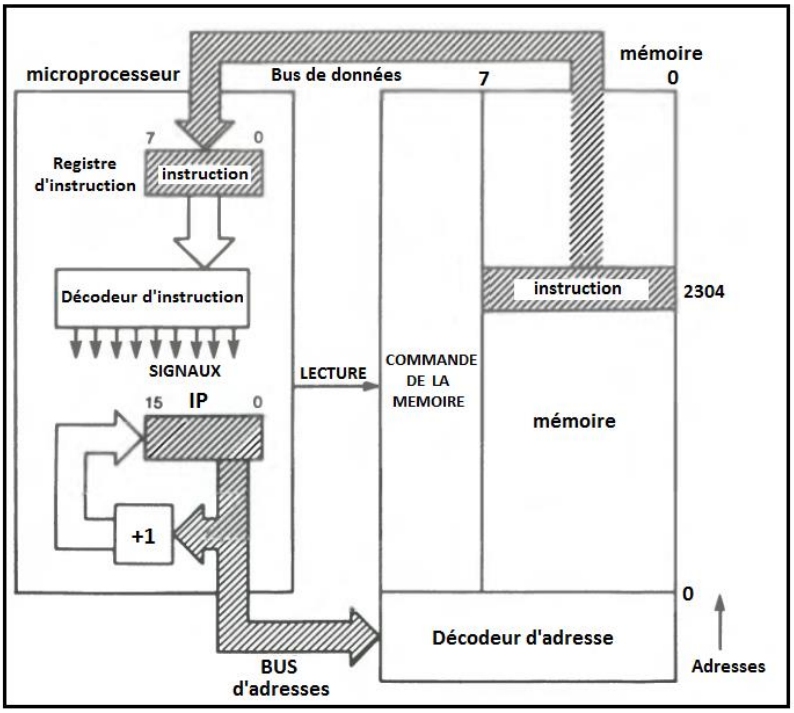


Figure 2.7. Schéma de la recherche d'une instruction en mémoire.

Exemple :

ADD	**ADD destination, source** **Addition**		**Drapeaux** O D I T S Z A P C X X X X X X
Opérandes	**Cycles horloge**	**Octets**	**Exemple de programme**
register, register	3	2	ADD CX, DX
register, memory	9 + EA	2-4	ADD DI, [BX].ALPHA
memory, register	16 + EA	2-4	ADD TEMP, CL
register, immediate	4	3-4	ADD CL, 2
memory, immediate	17 + EA	3-6	ADD ALPHA, 2
accumulator, immediate	4	2-3	ADD AX, 200

Tableau 2.1. Description de l'instruction ADD

MUL	**MUL source** **Multiplication non signée**		**Drapeaux** O D I T S Z A P C X U U U U X
Opérandes	**Cycles horloge**	**Octets**	**Exemple de programme**
reg8	70-77	2	MUL BL
reg16	118-133	2	MUL CX
mem8	(76-83) + EA	2-4	MUL MONTH [SI]
mem16	(124-139) + EA	2-4	MUL BAUD_RATE

Tableau 2.2. Description de l'instruction MUL

EA : pour désigner l'adresse effective (**E**ffective **A**ddress) de la donnée dans la mémoire.

Comme vous pouvez le constatez le temps d'exécution met par le microprocesseur pour exécuter l'instruction **MUL** (70-77 cycles horloge) est beaucoup plus important (plus long) que celui de l'instruction **ADD** (3 cycles horloge). (Voir les deux tableaux ci-dessus).

2.9 Codage d'instructions

Les instructions et leurs opérandes (paramètres) sont stockées en mémoire principale. La taille totale d'une instruction dépend du type d'instruction et aussi du type d'opérande. Une instruction est composée de deux champs :

- **le code opération (Opcode)** indique au processeur quelle instruction a réaliser.
- **le champ opérande** contient la donnée, ou la référence à une donnée en mémoire (son adresse).

Instruction	Code opération	Champ opérande
MOV AX, BX	**8B H (1 octet)**	**C3 H (1 octet)**

Chaque instruction commence par l'**opcode** qui détermine la nature de l'instruction, suivi du champ de l'opérande indiquant le type des opérandes, leur localisation, ou une valeur immédiate (littérale).

Exemple :

Instructions	Opcode	Code opérande	Taille (octets)	Code machine
MOV AX, BX	8B H	C3 H	2	1000 1011 1100 0011
MOV DS, AX	8E H	D8 H	2	1000 1110 1101 1000
MOV AX, 5H	8B H	0500 H	3	1000 1011 0000 0101 0000 0000
MOV AL, temper	A0 H	0E01 H	3	1010 0000 0000 1110 0000 0001

Le **langage machine** ou (**code machine)** est la suite de bits qui est interprétée par le microprocesseur d'un ordinateur exécutant un programme informatique. C'est le seul langage que le processeur puisse traiter.

2.10 Améliorations de l'architecture de base

L'ensemble des **améliorations** des microprocesseurs **visent à diminuer le temps d'exécution** du programme. Une autre possibilité d'augmenter la puissance de traitement d'un microprocesseur est de diminuer le nombre moyen de cycles d'horloge nécessaire à l'exécution d'une instruction. Une autre solution est d'utiliser une architecture de microprocesseur qui réduise le nombre de cycles par instruction.

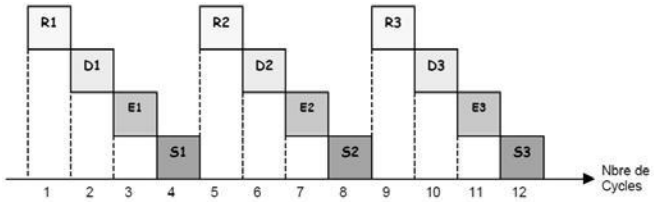
2.10.1 Architecture pipeline

L'exécution d'une instruction est décomposée en une succession d'étapes et chaque étape correspond à l'utilisation d'une des fonctions du microprocesseur (recherche du code, décodage, exécution). Lorsqu'une instruction se trouve dans l'une des étapes, les composants associés aux autres étapes ne sont pas utilisés. Le fonctionnement d'un microprocesseur simple n'est donc pas efficace.

L'architecture pipeline permet d'améliorer l'efficacité du microprocesseur. En effet, lorsque la première étape de l'exécution d'une instruction est achevée, l'instruction entre dans la seconde étape de son exécution et la première phase de l'exécution de l'instruction suivante débute. Une machine pipeline se caractérise par le nombre d'étapes utilisées pour l'exécution d'une instruction ou nombre d'étages du pipeline.

Exemple de l'exécution en 4 phases d'une instruction :

Modèle classique :



Modèle pipeline :

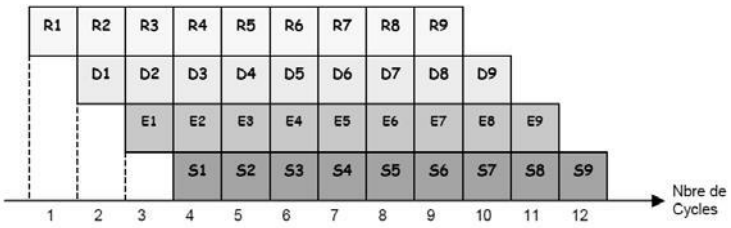


Figure 2.8. Comparaison entre le modèle d'exécution classique et le modèle d'exécution pipeline

2.10.2 Notion de cache mémoire

L'écart de performance entre le microprocesseur et la mémoire ne cesse de s'accroître. En effet, les composants mémoire bénéficient des mêmes progrès technologique que les microprocesseurs mais le décodage des adresses et la lecture/écriture d'une données sont des étapes difficiles à accélérer. La mémoire n'est plus en mesure de délivrer des informations aussi rapidement que le processeur. Il existe donc une latence d'accès entre ces deux organes.

Une des solutions utilisées pour masquer cette latence est de disposer une mémoire très rapide entre le microprocesseur et la mémoire. Elle est appelée "**cache mémoire"**. On compense ainsi la faible vitesse relative de la mémoire en permettant au microprocesseur d'acquérir les données à sa vitesse propre. Au départ cette mémoire était intégrée en dehors du microprocesseur mais elle fait maintenant partie intégrante du microprocesseur et se décline même sur plusieurs niveaux.

Le principe de cache est très simple : le microprocesseur n'a pas conscience de sa présence et lui envoie toutes ses requêtes comme s'il agissait de la mémoire principale :
Soit la donnée ou l'instruction requise est présente dans le cache et elle est alors envoyée directement au microprocesseur. On parle de succès de cache.

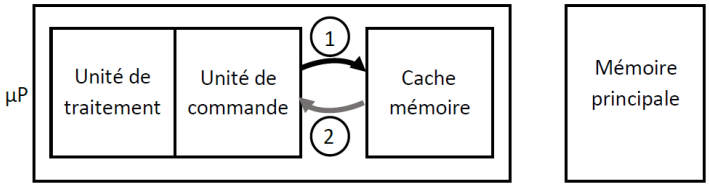


Figure 2.9. Succès de cache.

soit la donnée ou l'instruction n'est pas dans le cache, et le contrôleur de cache envoie alors une requête à la mémoire principale. Une fois l'information récupérée, il la renvoie au microprocesseur tout en la stockant dans le cache. On parle de défaut de cache.

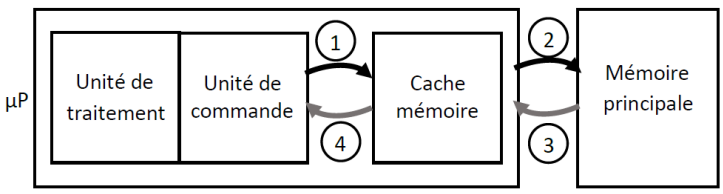


Figure 2.10. Défaut de cache.