

MICROPROCESSEUR - 2005

J. L. MAYORQUIM



Cergy, 4 Avril 2005

EISTI

REMERCIEMENTS

Je tiens à remercier **Mme M.J. LAMERRE, ISAMMOE,**
P. GAGNAT, qui m'ont apportés une aide précieuse pendant la phase de correc-
tion.

MICROPROCESSEUR 2005

1.1 Introduction

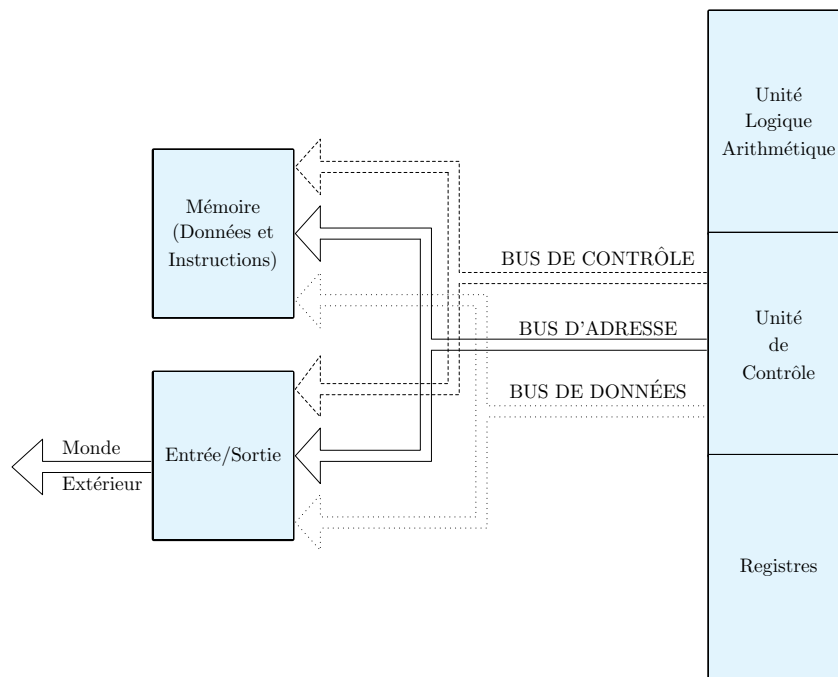
Pendant 30 ans, l'ordinateur l'**ABC**, construit par **John Vincent ATANASOFF**, **C. BERRY** et **CLIFFORD**, de 1939 et 1941, a été éclipsé par celui construit en 1945 par **John W. MAUCHLY** et **John Presper ECKERT** jusqu'à ce qu'un juge fédéral américain reconnaisse que le second n'était qu'une copie du premier.

La proposition d'**ATANASOFF** était composée de trois blocs fondamentaux qui sont illustrés par la Figure 1.1.1. Les blocs sont connectés entre eux par des lignes nommées **BUS**.

Central Processing Unit - CPU s'adresse à la mémoire à travers d'un **bus d'adresse** qui obtiendra les instructions ou les données par les **bus de données**. Le traitement de l'information est réalisé par l'unité de contrôle avec le soutien des registres de travail.

L'unité de contrôle fait le décodage des instructions et envoie plusieurs signaux de contrôle à tous les éléments qui interviennent sur l'opération à partir du **bus de contrôle**. De plus, l'unité de contrôle charge le **bus d'adresse** avec la valeur de la position mémoire ou des Registres d'**Entrée/Sortie** qui permettent de lire ou écrire les informations. Il faut noter que l'unité d'**Entrée/Sortie** fait l'adaptation des informations qui rentrent ou sortent du système vers les périphériques extérieurs.

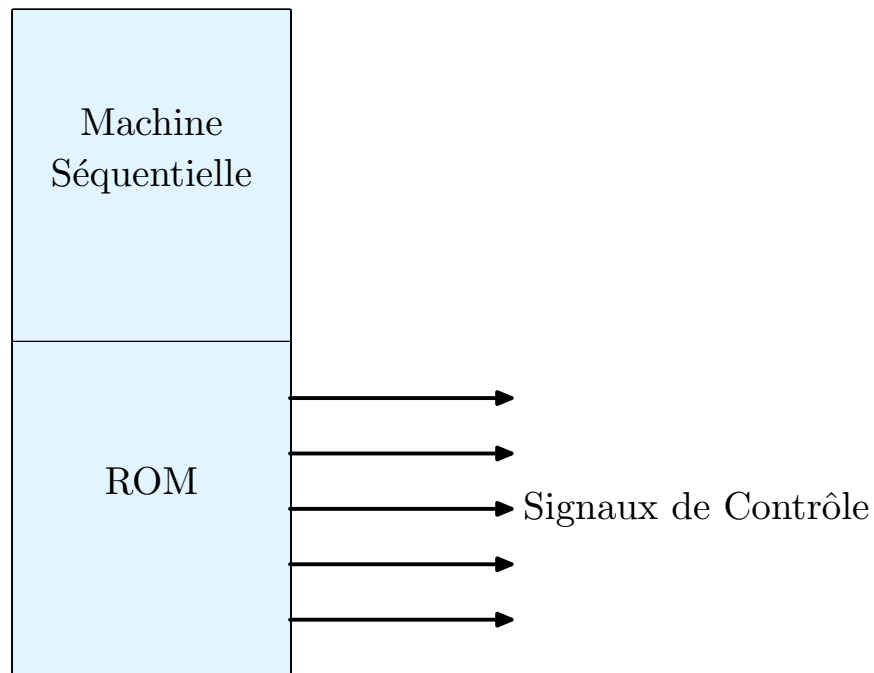
Figure 1.1.1 Structure de base de l'ordinateur.



1.2 Fonctionnement du microprocesseur

Les signaux qui contrôlent le fonctionnement des circuits intégrés de la **CPU** permettent d'exécuter l'instruction en cours. Mais les instructions sont complexes, il faut donc les décomposer en pas élémentaires, qui seront exécutés en fonction du cycle d'horloge de la machine séquentielle. Les pas élémentaires d'une instruction sont nommés des **micro-instructions**. Les instructions de machines qui admettent une **CPU** et qui sont toujours les mêmes, seront aussi appelées micro-instructions. De plus, les groupes de **bits** qui décrivent les micro-instructions sont enregistrés dans une mémoire permanente de type **Read Only Memory - ROM** qui est nommée **Mémoire de Contrôle**.

Nous connaissons le concept de machine séquentielle, i.e. c'est un mécanisme qui prend en charge la séquence des micro-instructions de la **Mémoire de Contrôle** et qui correspond à l'instruction en cours d'exécution, comme l'illustre la Figure 1.2.1.

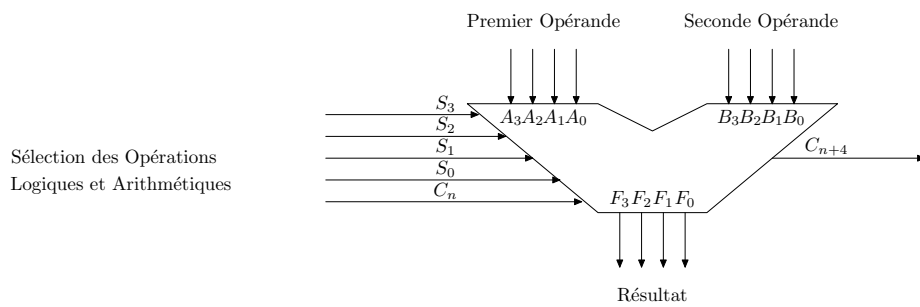
Figure 1.2.1 Machine séquentielle et ROM.

Les signaux de contrôle qui sont générés par la machine séquentielle réalisent un ensemble d'actions, entre autres, sur l'unité d'opération, i.e. sur l'**Unité Arithmétique & Logique - ALU**. Les instructions admissibles par le **CPU** sont fonction des opérations que l'**ALU** peut réaliser.

Dans ce projet, l'**ALU SN74181** a été choisi. Ce composant manipule les opérations de 4 bits, que nous avons définies comme étant la taille du mot de travail de la **CPU** et qui définira la dimension du **bus de données**.

Le schéma de connexion de l'**ALU** est montré dans la Figure 1.2.2. Il a besoin de 4 signaux ($S_0 - S_3$) pour sélectionner l'opération qui sera réalisée. De plus, le signal M permet de sélectionner 16 opérations logiques ou 16 opérations arithmétiques que cet **ALU** peut exécuter. Nous considérons $M = 1$ pour les opérations logiques et $M = 0$ pour les opérations arithmétiques.

Figure 1.2.2 Unité logique et arithmétique.



Le Tableau 1.1 illustre le groupe des 32 opérations de l'ALU. De plus, le circuit du microprocesseur a 28 circuits intégrés qui font partie de l'architecture du CPU et sont de type standard. Dans ce projet 6 circuits intégrés sont des portes simples et les autres sont d'usage spécifique, i.e. ALU, compteurs, buffers, bascules, ROM.

Table 1.1: Les 32 opérations de l'unité logique et arithmétique.

Logiques	Arithmétiques
$F = \bar{A}$	$F = A \text{ plus } C \text{ (} C \Rightarrow \text{Retenue)}$
$F = A + B$	$F = (A + B) \text{ plus } C$
$F = A \bullet B$	$F = (A + \bar{B}) \text{ plus } C$
$F = 0000$	$F = 1111 \text{ plus } C$
$F = \bar{A} \bullet \bar{B}$	$F = A + A \bullet \bar{B} \text{ plus } C$
$F = \bar{B}$	$F = (A + B) \text{ plus } A \bullet \bar{B} \text{ plus } C$
$F = A \oplus B$	$F = A \text{ moins } B \text{ plus } C$
$F = A \bullet \bar{B}$	$F = A \bullet \bar{B} \text{ moins } 1 \text{ plus } C$
$F = \bar{A} + B$	$F = A \text{ plus } A \bullet B \text{ plus } C$
$F = \bar{A} \oplus \bar{B}$	$F = A \text{ plus } B \text{ plus } C$
$F = B$	$F = (A + \bar{B}) \text{ moins } A \bullet B \text{ plus } C$
$F = A \bullet B$	$F = A \bullet B \text{ moins } 1 \text{ plus } C$
$F = 1111$	$F = A \text{ plus } A \text{ plus } C$
$F = A + \bar{B}$	$F = (A + B) \text{ plus } A \text{ plus } C$
$F = A + B$	$F = (A + \bar{B}) \text{ plus } A \text{ plus } C$
$F = A$	$F = A \text{ moins } 1 \text{ plus } C$

Le projet du CPU est composé de trois grands blocs qui sont les suivants :

- Bloc arithmétique et logique ;
- Programme Counter - Compteur de Programme - PC ;

— Machine séquentielle.

1.3 Bloc arithmétique et logique

Cette partie de la **CPU** est responsable de la réalisation des opérations du traitement des données de 4 bits. Il est basé sur l'**ALU** SN74181. Dans la Figure 1.3.1¹, nous illustrons le schéma correspondant au bloc arithmétique et logique. L'**ALU** a deux entrées de données de 4 bits : une où plusieurs données proviennent directement du **bus interne de données**, pendant que l'autre reçoit des bascules (SN74175) de 4 bits. Le composant SN74175 est formé par 4 bascules du type **D**. Ce registre est nommé **Registre A** et l'information qu'il stocke provient de l'**ALU** et/ou du **bus interne de données** du **CPU**. Les bascules **D** du composant SN74175 sont chargées au moment de la transition, i.e. le front montant du signal d'horloge (**CLOCK**). De plus, le signal **EA** généré par la machine séquentielle est synchronisé par l'horloge.

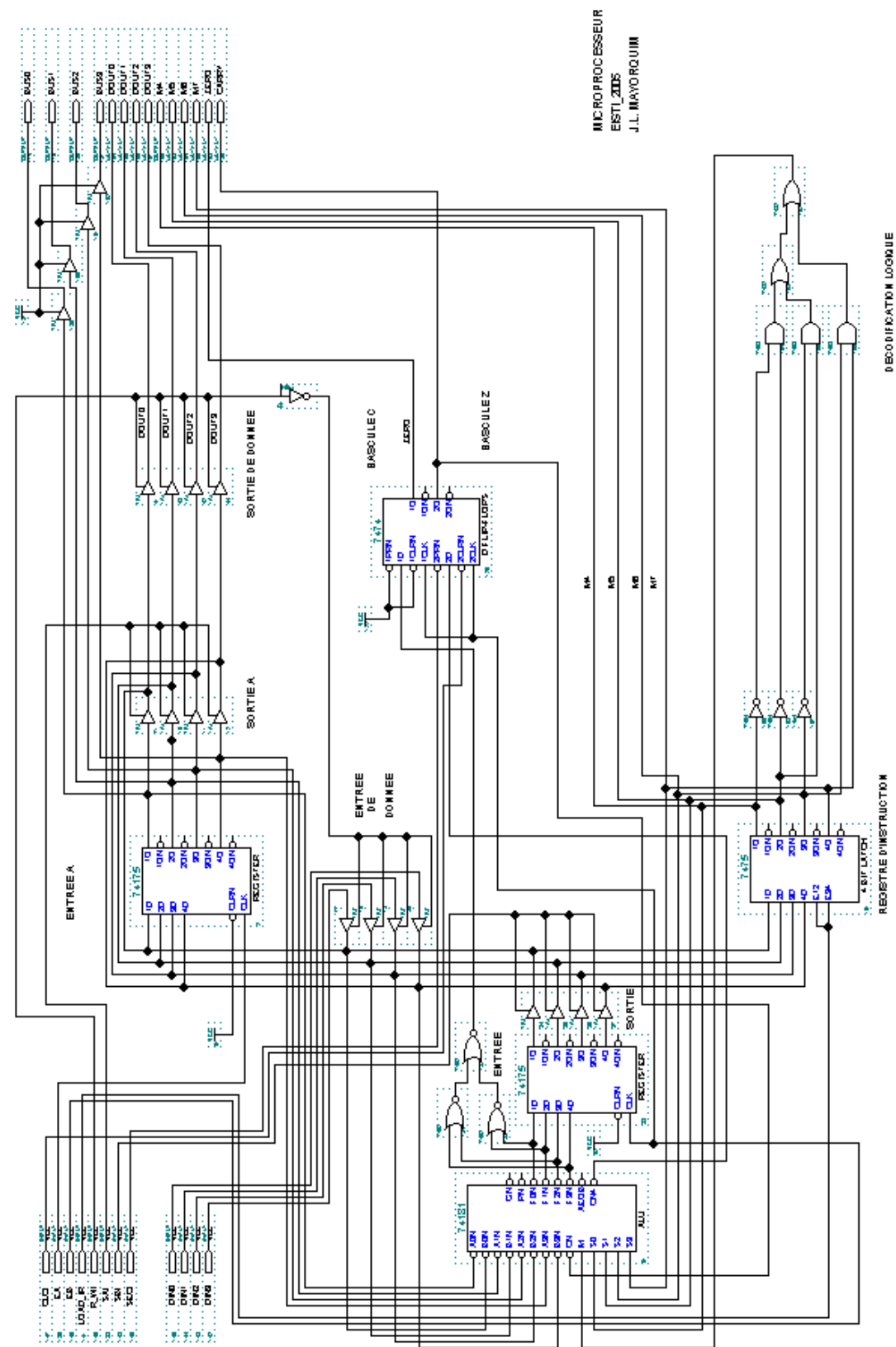
L'objectif est d'envoyer au **bus interne de données** l'information contenue dans le **Registre A**. La sortie du SN74175 (nommée aussi **Entrée A**) envoie les données vers l'**ALU** et le tri-state (SN74125), i.e. nous faisons le transfert de l'information du **Registre A** vers le **bus interne de données**. La machine séquentielle active le signal niveau bas **SA (Sortie A)**, pendant que le SN74175 retient l'information en permanence. De plus, le SN74125 transmet le **bus interne de données** qui reçoit le signal de contrôle **SA**.

Dans un chemin semblable, la sortie de l'**ALU** reste dans un autre registre SN74175 qui est nommé **Registre B**. Un autre composant tri-state SN74125 (**Sortie B**) prend en charge le transfert de l'information du **Registre B** vers le **bus interne de données**, quand la machine séquentielle active le signe **SB**.

Quand l'**ALU** effectue une opération, elle génère une sortie de retenue (C_{n+4}), l'information d'état de la retenue **C** sera stockée par une bascule. Elle utilisera son contenu antérieur à l'entrée de la retenue (C_n), avant la réalisation de l'opération. La bascule de retenue **C** reçoit deux signaux de la machine séquentielle, un signal pour mettre à 1 (**SEC**) et l'autre pour mettre à 0 (**CLC**). Le signal d'horloge **CLOCK** de la bascule **C** est activé en même temps que le signal **EB (Entrée B)**, qui charge le **Registre B** avec le résultat de sortie de l'**ALU**.

¹ Schémas :/corriges/jma/VHDL_Micro_2005.

Figure 1.3.1 Schéma de l'ALU.



Un groupe de 3 portes logiques examine la sortie de l'**ALU** et il contrôle une bascule du type **D**, qui est nommé état zéro (**Z**), qui aura la valeur 1 quand le résultat d'une opération sera zéro. Les états, **C** et **Z**, sont intégrés dans le même circuit intégré *SN7474*.

L'**ALU** a aussi besoin de recevoir 4 signaux qui sélectionnent l'opération, et le signal (**M**) qui détermine si l'opération est de type logique ou arithmétique. Le signal qui sélectionne l'opération provient du code d'opération (**Operation Code - OP**) de l'instruction qui entre par le **bus externe de données**. Dans ce cas, nous disposons de deux circuits intégrés *SN74125* et *SN74126* qui contiennent 4 **buffers** de **3-états** (tri-state) et qui contrôlent les entrées et les sorties de données. Le signal R/\overline{W} de la machine séquentielle définit s'il y a entrée ou sortie de données.

Quand le *SN74125* reçoit un **OP**, celui-ci est chargé dans le circuit intégré *SN7475* utilisé comme un **Registre d'Instructions**. Il est appliqué alors aux signaux de sélection de l'**ALU** (S_0, S_1, S_2 et S_3). En utilisant un circuit décodeur à partir de portes logiques, les signaux M_4, M_5, M_6 et M_7 obtiendront la valeur de M qui définit si l'**ALU** réalisera une opération logique ou arithmétique.

1.4 Program Counter

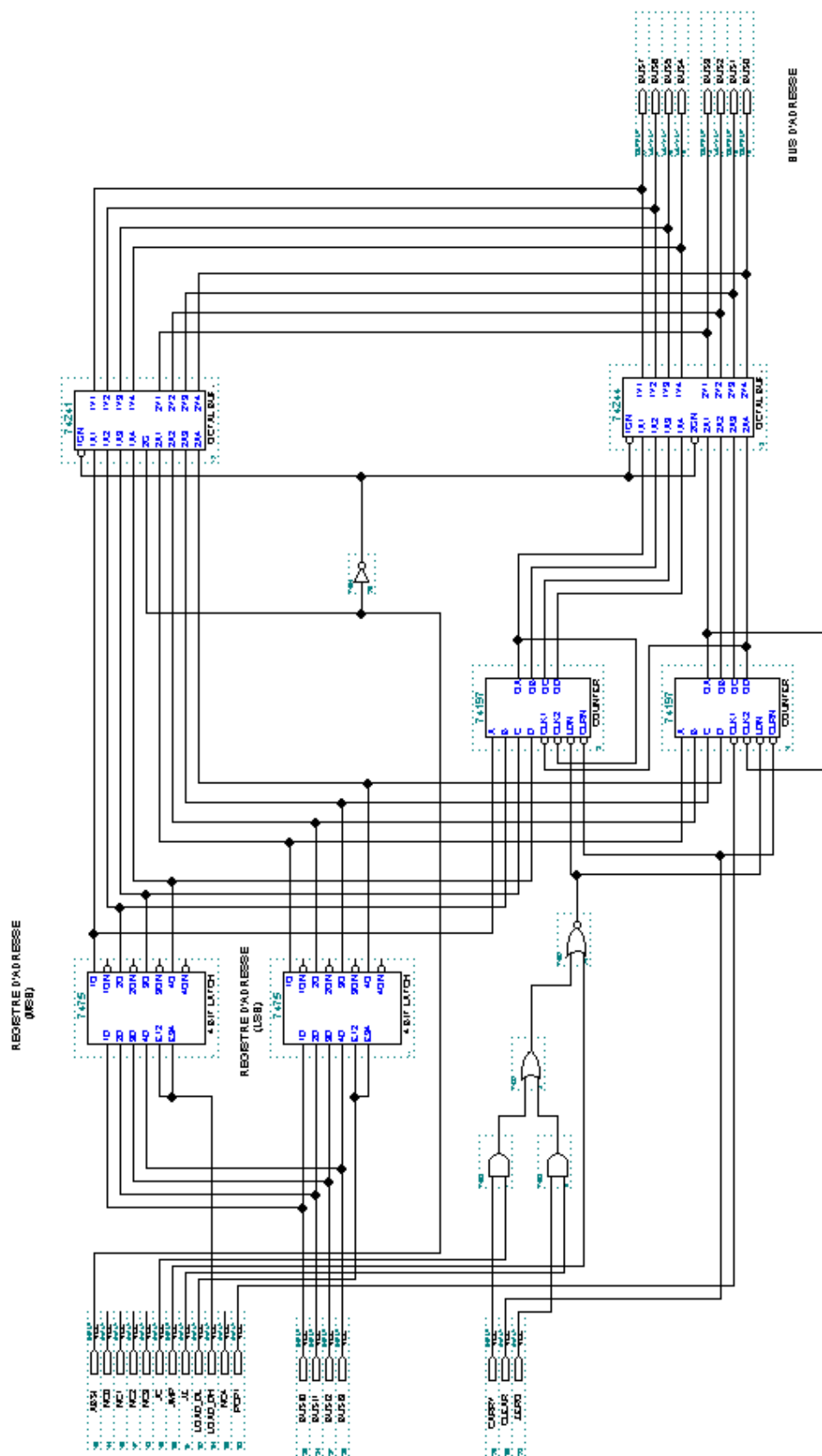
Dans le **CPU**, le composant fondamental est le **Program Counter - PC** (Compteur de Programme). Il fournit l'adresse de la mémoire où se trouve la prochaine instruction qui sera exécutée. Le schéma de la Figure 1.4.1 illustre le **PC** qui peut fonctionner de trois manières différentes.

1.4.1 Forme habituelle

En général, le **PC** s'incrémente chaque fois qu'il exécute une instruction, i.e. la machine séquentielle incrémente le **PC** par le signal **PCP1**.

Le **PC** est formé par des compteurs binaires, *SN74197*, en série, et l'adresse est de 8 *bits*. Les compteurs *SN74197* sont de 4 *bits*, et sont mis à zéro par le signal **CLEAR**. La sortie du **PC** est l'entrée d'un **buffer** de sortie **tri-state** (*SN74244*), qui est composé par un ensemble de 4 signaux parallèles. La sortie du **buffer** est contrôlée par le signal **G**, qui actualisera le **bus d'adresse**.

Figure 1.4.1 Schéma du compteur de programme.



1.4.2 Mode d'adressage absolu

Dans ce cas, le **bus d'adresse** envoie la position de la mémoire qui lira ou stockera la donnée. Une fois l'accès à la mémoire réalisé, le **PC** est incrémenté et adresse la prochaine instruction qui sera exécutée. Le mode d'opération, dans ce cas, est **absolu**, i.e. l'instruction a un accès direct où se trouve l'opérande qui doit être lu ou écrit.

Le procédé est initialisé par le chargement du **Registre d'Instruction**. Il est formé de deux bascules *SN7475* qui gardent les valeurs *MSB (DH)* et *LSB (DL)*. Les bascules sont activées par les signaux *DH* et *DH* provenant de la machine séquentielle.

Une fois les bascules chargées, les contenus passent par un **buffer** *SN74244* contrôlé par le signal \overline{ABS} de la machine séquentielle.

1.4.3 Sauts

Quand un programme réalise un saut, il passe à l'adresse qui correspond au saut. Le **PC** sera augmenté à partir de cette adresse. Cette opération est en opposition avec le mode d'adressage absolu.

Comme le **CPU** est simplifié, il n'a pas été préparé à attendre les interruptions, i.e. il faut noter dans ce cas qu'il est nécessaire de garder le contenu du **PC** pour le retourner plus tard.

Quand le **CPU** fait un saut, il reçoit de la machine séquentielle un des trois signaux suivant : **JMP**, **JC** ou **JZ**. À partir d'un groupe de portes logiques, le signal \overline{LC} du compteur *SN74197* est activé. Il permet de charger le **Registre d'Adresse** (*SN7475* × 2). Il continue alors le comptage de l'adresse du **PC**.

Figure 1.4.2 Machine séquentielle.

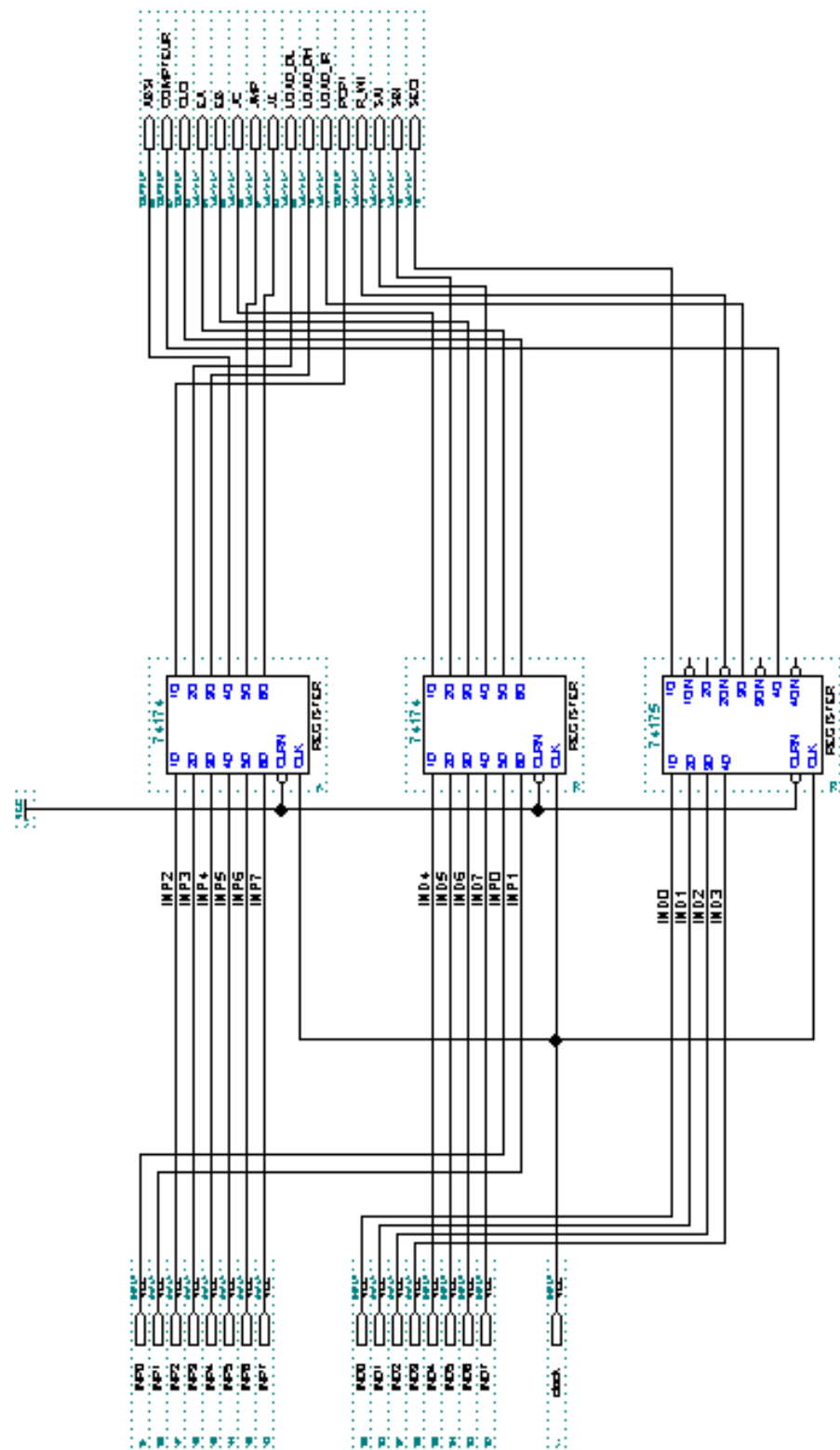
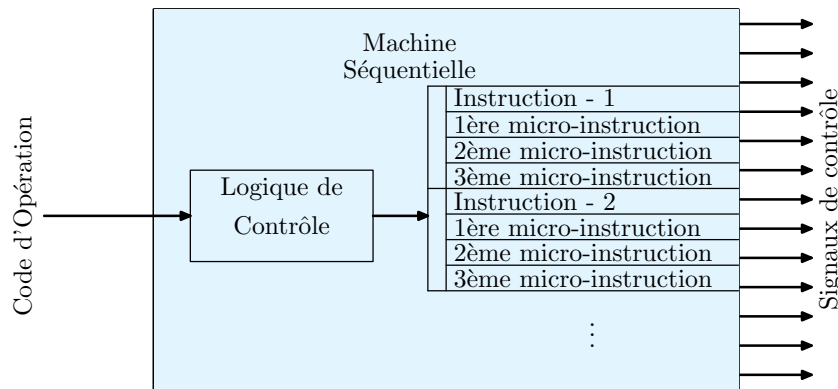


Figure 1.4.4 Machine séquentielle.



Dans ce projet, toutes les micro-instructions qui composent le jeu d'instruction de la **CPU** sont enregistrées dans deux mémoires **ROM**. Ces mémoires ont une capacité de $2\text{ Kbytes} \times 8\text{ bits}$. Dans cette application, l'utilisation de la mémoire est inférieure à 2 Kbytes , i.e. seulement 8 lignes d'adresse sont utilisées pour la **ROM** et trois lignes d'adresse sont mises à la masse.

Les deux mémoires **ROM** sont adressées en parallèle par 2 bytes d'information (un byte pour chaque composant). Les 16 bits sont obtenus pour faire l'adressage du couple de **ROM**. Les signes de sortie de la machine séquentielle permettent de contrôler le système. De plus, les sorties des mémoires passent par 3 registres (*SN74174* et *SN74175*) pour former le **bus du contrôle**.

Les 8 bits qui forment l'adresse servent simultanément les deux mémoires, ils sont distribués par les 4 bits de l'**OP** de l'instruction qui proviennent de **CMI**. Les 4 **Most Significant Bits - MSB** de l'adresse (A_4, A_5, A_6 et A_7) correspondent avec les 4 bits de l'**OP** de l'instruction qui provient du **Registre d'Instruction** (*SN7475*). Les 4 **Lost Significant Bits - LSB** d'adresse proviennent du compteur *SN74197* qui est nommé **Compteur de Micro-Instructions - CMI**. L'**CMI** augmente d'une unité à chaque impulsion d'horloge externe. De cette manière, l'**OP** fixe la valeur des 4 *MSB* du couple d'adresse des mémoires (permettant d'avoir un maximum de $2^4 = 16$ instructions). En sélectionnant avec les 4 *MSB*, il démarre à la position de l'instruction à exécuter, le **CMI** est incrémenté à chaque impulsion de l'horloge et il fait un balayage des 16 emplacements mémoires. De cette façon, il fait varier la valeur des 4 *LSB* d'adresse. Et chaque position de la mémoire a une micro-instruction. De plus, une instruction peut être composée d'un maximum de 16 micro-instructions.

En résumé, l'**OP** plus le **CMI** augmente à partir des impulsions de l'horloge. Il adressera les deux mémoires en parallèle. Dans chaque cycle d'horloge, une position de chaque mémoire est adressée et une micro-instruction de 16 bits est obtenue. Les signaux de sorties contrôlent différents composants.

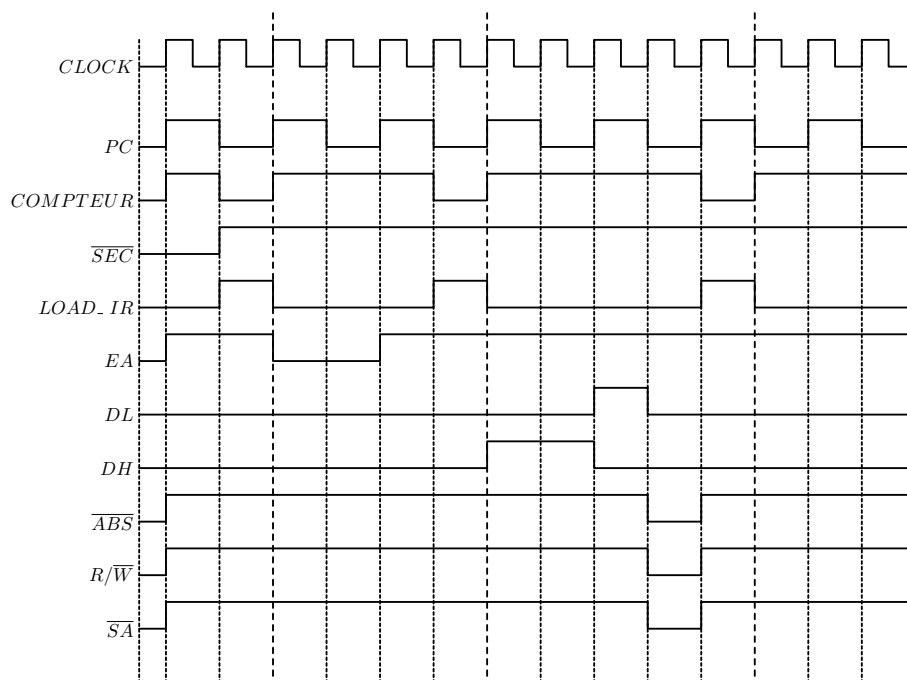
Le **CMI** est incrémenté par les fronts descendants d'horloge, pendant que les bascules de sortie des signaux d'instructions reconnaissent l'information aux fronts montants pour retenir l'information après le temps d'accès des mémoires.

1.5 Diagramme de fonctionnement

Pour l'opération de la **CPU**, nous disposons d'un signal de **RESET** qui met à 0 le **PC**. Ensuite, il reconnaît l'**OP** de la première position de la mémoire principale. Il met aussi à 0 le **CMI**, pour former l'adresse de l'instruction. Après un temps en millisecondes, il adresse la première instruction. Il commence à travailler au niveau bas d'horloge. Durant le premier flanc montant, les bascules sont chargées par les signaux de contrôle, et durant le front descendant le **CMI** est augmenté, et adressera la micro-instruction. Quand nous arrivons à la dernière micro-instruction, la propre machine séquentielle prend la charge de mettre elle-même à 0 le **CMI** et chargera le **Registre d'Instruction** avec l'**OP** suivante.

Sur le Figure 1.5.1 le diagramme correspond au signaux de contrôle qui sont présentés pour 3 instructions :

- *SEC* : Mettre à 1 la bascule de retenue **C** ;
- *LDA* : Charger le **Registre A** ;
- *STA* : Stocker le contenu de la mémoire dans le **Registre A**.

Figure 1.5.1 Diagramme temporelle pour les instructions *SEC*, *LAD* et *STA*.

1.5.1 Modes d'adressage et jeux d'instructions

Plusieurs composants de la **CPU** ont été inclus pour être capable de rendre effectives des instructions ou des micro-instructions. Pour cette raison, l'architecture d'un processeur est très liée au système logique, même pour les opérations élémentaires. Il y a plusieurs modes pour faire l'adressage du projet en fonction du **PC** travail. Nous utiliserons les suivantes :

- Adressage absolu ;
- Adressage relatif du *PC*, i.e. sauter sans retour. ;
- Adressage immédiat.

Le jeu d'instructions est composé par 16 instructions, puisque l'**OP** a 4 *bits*, i.e. nous pouvons seulement atteindre 16 combinaisons différentes. De plus, chaque instruction peut consister d'un maximum de 16 micro-instructions. Une fois définies, les instructions avec les 4 bits de l'**OP** seront utilisées comme adresse de 4 *MSB* du couple de **ROM**. Pour compléter l'adresse le **CMI** fournira les 4 *LSB* d'adressage. Les instructions sont rendues effectives par les micro-instructions qui sont enregistrées dans le couple de mémoires de type **ROM**.

1.5.1.1 CPL

Cette instruction fait le complément du contenu du **Registre A** dans l'**ALU** et le résultat est stocké dans le **Registre B**. Elle consiste en deux micro-instructions qui sont les suivantes :

- La première micro-instruction est l'**OP** 0000₂ qui fait la sélection de l'opération de l'**ALU**. La machine séquentielle active le signal qui correspond à l'augmentation du prochain **PC**, i.e. $PC + 1$ et active le **Registre A**.
- La deuxième micro-instruction augmente le **PC** et habilite l'entrée de donnée du **Registre B** par le signal **EB**. Celui-ci recevra le résultat du complément du **Registre A** qui est le résultat de l'**ALU**. Le signal **LOAD_IR** habilite le **Registre d'Instruction** pour être capable de ramasser le code du prochain **OP** et le signal **COMPTEUR** passe à 0, ce qui permet au **CMI** de commencer l'exploration des micro-instructions de l'instruction.

Avec cette instruction, les registres d'états **Z** et **C** peuvent être affectés :

- Si **C** = **1** est indépendamment du résultat de l'opération et **Z** sera affecté par le résultat.
- Si **C** = **0**, cet état se comporte inversement à l'état **Z**.

1.5.1.2 SEC

Cette instruction dont l'**OP** est 1₁₆ met à 1 la retenue **C**. Elle est constituée de deux micro-instructions. La première, prépare l'augmentation du **PC** quand le signal \overline{SEC} est actif, la deuxième efface le **CMI** et augmente le **PC**.

1.5.1.3 CLC

Elle met la retenue **C** à 0. L'**OP** est 2₁₆. La première micro-instruction, la machine séquentielle prépare l'augmentation du **PC** quand le signal \overline{CLC} est actif. La deuxième efface le **CMI** et augmente le **PC**.

1.5.1.4 NOP

Elle n'opère pas. Il occupe 4 pulsations d'horloge sans affecter aucun élément ni réaliser d'opération. L'**OP** est 3₁₆.

Durant les trois premières micro-instructions, elle prépare l'augmentation du **PC** et dans la quatrième, elle actualise cette augmentation. Ensuite, elle efface le **CMI** et se prépare pour faire un cycle de **FETCH**, i.e. il est capable de charger une nouvel **OP**.

1.5.1.5 JC

C'est une instruction de saut conditionnel, quand la retenue **C** est égale à 1, le **PC** saute à une nouvelle position. Cette position est spécifiée dans le deuxième et le troisième opérandes qui continuent l'**OP** de l'instruction. L'**OP** est 4_{16} .

Dans la première micro-instruction, elle se prépare pour augmenter le **PC**. La deuxième se prépare pour charger le **Registre d'Adresse** avec 4 *MSB*. La troisième micro-instruction charge le **Registre d'Adresse** avec les 4 *MSB* et se prépare pour faire l'augmentation du **PC**. Dans la quatrième micro-instruction, le **PC** est augmenté et elle se prépare pour charger les 4 *LSB* du **Registre d'Adresse**. Dans la cinquième, elle se prépare encore pour augmenter le **PC** et elle active le signal **JC**. Si **C** = 1, le **PC** est chargé avec le contenu du **Registre d'Adresse** et garde la position où le saut aura lieu. Le sixième cycle prépare l'augmentation du **PC** et donne le temps à la mémoire pour accéder l'**OP**. Dans la septième micro-instruction le **PC** est augmenté, le **CMI** est effacé et il se prépare à stocker l'instruction suivante. Si **C** = 0, le saut n'aura pas lieu et le programme suivra la séquence normale.

1.5.1.6 JZ

C'est un saut conditionnel à la position spécifiée par le second et le troisième opérandes de l'**OP**, si **Z** = 1. Si **Z** = 0, le programme suit la séquence normale. L'**OP** est le 5_{16} . Il est aussi constitué de 7 micro-instructions comme **JC** à la seule différence qu'à la cinquième micro-instruction, les signaux **JC**, **JZ** sont activés.

1.5.1.7 SBC

Cette instruction fait la soustraction du contenu du **Registre A** et du contenu adressé par la position (adressage absolue) et la retenue. Le résultat de l'opération est entreposé dans le **Registre B**. De plus, cette instruction peut affecter **C** et **Z**. L'**OP** est le 6_{16} . Elle est constituée de 6 micro-instructions. Dans la première, elle prépare l'augmentation du **PC**. Pour la deuxième, le **PC** est augmenté et le **Registre d'Adresse** est chargé par le *MSB*. À la troisième micro-instruction, elle prépare le **PC** et elle est continue le chargement du **Registre d'Adresse**. Pour la quatrième, le **Registre d'Adresse** est chargée par le *LSB*. Pour la cinquième, elle prépare l'augmentation du **PC** et le signal \overline{ABS} est activé, avec le contenu adressé par le **bus d'adresse**. La dernière micro-instruction, le **PC** s'incrémente, le contenu du **bus de données** passe par l'**ALU** où est effectuée la soustraction et le signal **EB** est activé. Enfin, le résultat est entreposé dans le **Registre B**. Le **CMI** et le **Registre d'Instruction** sont préparés pour recevoir la prochaine instruction.

1.5.1.8 JMP

C'est un saut inconditionnel à une position de mémoire indiquée par le second et troisième opérandes qui suivent l'**OP** (7_{16}). Il consiste en 7 micro-instructions ou cycles machine et suit la même séquence que l'instruction **JC**, à l'exception de la cinquième micro-instruction qui active le signal **JC** à la place de **JMP**.

1.5.1.9 LDA

Elle charge le **Registre A** avec le contenu d'une position de mémoire adressée par le second et le troisième opérandes qui suivent l'**OP**, i.e. l'adressage absolu. Le code est 8_{16} et elle est constituée de 6 micro-instructions. En premier, il prépare le **PC**. La seconde augmente la valeur du **PC** et le **Registre d'Adresse** est chargé avec la valeur *MSB*. La troisième, le **PC** est incrémenté et il est continue le chargement du *MSB*. La quatrième, le **Registre d'Adresse** est chargé avec la valeur *LSB*. La cinquième, elle se prépare à l'augmentation du **PC**, le signal \overline{ABS} est activé. Le signal *EA* est aussi préparé, et passera à 0. La sixième micro-instruction, le **PC** est augmenté et le contenu du **bus de données** est transféré vers le **Registre A**. Le **PC** est augmenté et prépare le **CMI** et le **Registre d'Instruction** stocke l'instruction suivante.

1.5.1.10 ADC

Additionner le contenu du **Registre A** avec le contenu de l'adresse de position de l'instruction et la retenue. Le résultat est entreposé dans le **Registre B**. De plus, il peut affecter l'état de **C** et **Z**. L'**OP** est 9_{16} et elle consiste en 6 micro-instructions. La première prépare le **PC** pour l'augmenter. La deuxième, le **PC** est augmenté et le **Registre d'Adresse** est chargé avec *MSB*. La troisième prépare le **PC** pour faire l'augmentation et continuer le chargement le *MSB*. La quatrième, le **Registre d'Adresse** est chargé. La cinquième, la machine séquentielle active le signal \overline{ABS} . Dans ce cas, l'adresse sort du **Registre d'Adresse** vers la mémoire. Elle prépare aussi le signal \overline{EB} . À la dernière micro-instruction, le contenu du **bus de données** passe par l'**ALU** qui effectue l'opération d'addition et active le signal \overline{EB} . Le résultat est entreposé dans le **Registre B**. Le **PC** est augmenté et il prépare le **CMI** et le **Registre d'Instruction** pour recevoir un nouveau **OP**.

1.5.1.11 STA

Elle transfère le contenu du **Registre A** à une position de mémoire adressé par les seconde et troisième opérandes de l'**OP** A_{16} . Elle consiste en 6 micro-instructions. Les quatre premières sont semblables à ce qui a été commenté pour l'instruction **LDA**. La cinquième, prépare le **PC**, en activant le signal \overline{ABS} et le signal R/\overline{W} pour écriture. Dans le dernier cycle, le **PC** est augmenté et elle prépare le **CMI** et le **Registre d'Instruction** pour recevoir une nouvelle instruction.

1.5.1.12 ET

Cette instruction effectue l'opération logique **ET** parmi les bits du **Registre A** et le contenu d'une position de mémoire adressé par le second et le troisième opérandes de l'**OP** (B_{16}). Elle consiste en 6 cycles machine semblables au reste des opérations logiques ou arithmétiques qui sont réalisées par l'**ALU**.

1.5.1.13 STB

Le contenu du **Registre B** est transféré à une position de mémoire adressée par le second et le troisième opérandes qui continuent l'**OP** (C_{16}). Elle est constituée de 6 micro-instructions semblables à ce qui a été commenté pour l'instruction **STA**. La seule différence est la cinquième qui activera les signaux \overline{SA} et \overline{SB} .

1.5.1.14 LDA

Cette instruction charge le **Registre A**. Le deuxième opérande de l'**OP** (D_{16}) est chargé dans le **Registre A**, égal au mode d'adressage immédiat. Elle consiste en 4 cycles machine. Le premier prépare le **PC**. Le second, le **PC** augmente et prépare le signal \overline{EA} . La troisième micro-instruction prépare le **PC** et active \overline{EA} . La dernière micro-instruction, augmente le **PC** et prépare le **CMI** et le **Registre d'Instruction**.

1.5.1.15 OU

Cette instruction est une opération logique **OU** entre le **Registre A** et le contenu d'une position de mémoire adressé par le second et le troisième opérandes. Le résultat est entreposé dans le **Registre B** et l'**OP** est E_{16} . Elle consiste en 6 micro-instructions semblables à l'instruction **ET**.

1.5.1.16 MOV A, B

Cette instruction transfère le contenu du **Registre A** vers le **Registre B** en passant par l'**ALU**. Les états **Z** et **C** sont affectés. L'**OP** est le F_{16} . Elle consiste en deux micro-instructions. La première prépare le **PC** et le signal \overline{EB} . La dernière augmente le **PC**, et active \overline{EB} et le **Registre d'Instruction**. De plus, le **CMI** est initialisé pour recevoir un nouvel **OP**.

Les Tableaux 1.2, 1.3, 1.4 et 1.5 représentent les micro-programmes des signaux de contrôle qui sortent de la machine séquentielle pour le jeu d'instructions.

Table 1.2: Signaux de contrôle et instructions.

Instructions & Fonctions		Bus d'Adresse								Bus de Contrôle														
	Code (OP)				CMI				ROM I						ROM II									
	3	2	1	0	3	2	1	0	PCP1	DL	DH	ABS	JMP	JZ	JC	SB	EB	SA	EA	CLC	SEC	RW	LD_JR	COMPTEUR
<i>CPL</i> ($A \rightarrow B$)	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	1	1	1	1	0	0
$\$ 0 (A \rightarrow B)$	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	1	1	1	1	1	0	1
<i>SEC</i> ($1 \rightarrow C$)	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	0	1	0	0
$\$ 1 (1 \rightarrow C)$	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0
<i>CLC</i> ($0 \rightarrow C$)	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	1	1	0	0
$\$ 2 (0 \rightarrow C)$	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1
<i>NOP</i> (NON OPERATION)	0	0	1	1	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1
$\$ 3$ (NON OPERATION)	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1
0	0	0	1	1	1	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	0	0	1	1	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0
<i>JC</i> ($C = 1$)	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	0
$\$ 4 (C = 1)$	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1	0	0	1													

Table 1.3: Signaux de contrôle et instructions (suite).

Instructions & Fonctions		Bus d'Adresse								Bus de Contrôle																
		Code (OP)		CMI				ROM I				ROM II														
		3	2	1	0	3	2	1	0	PCPI	DL	DH	ABS	JMP	JZ	JC	SB	EB	SA	EA	CLC	SEC	RW	LD_IR	COMPTEUR	
<i>SBC (A ← M ← C → B)</i> <i>\$ 6 (A ← M ← C → B)</i>		0	1	1	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1
		0	1	1	0	0	0	0	0	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	0	1
		0	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	0	1
		0	1	1	0	0	0	1	1	0	1	0	1	0	0	0	0	1	1	1	1	1	1	1	0	1
		0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1
		0	1	1	0	0	1	0	1	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1
<i>JMP</i> <i>\$ 7</i>		0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1
		0	1	1	1	0	0	0	0	1	0	1	1	0	0	0	1	1	1	1	1	1	1	0	1	
		0	1	1	1	0	0	0	1	1	0	1	1	0	0	0	1	1	1	1	1	1	1	0	1	
		0	1	1	1	0	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	
		0	1	1	1	1	0	1	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	
		0	1	1	1	1	0	1	1	0	0	0	1	0	0	0	0	1	1	1	1	1	1	1	0	1
<i>LDA (M → A)</i> <i>\$ 8 (M → A)</i>		1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1
		1	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	0	1
		1	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	0	1
		1	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	0	1
		1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1
		1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1

