# Une introduction à l'architecture des ordinateurs

Béatrice Bérard, Jean Goubault-Larrecq
LSV/UMR 8643, CNRS, ENS Cachan & INRIA Futurs projet SECSI
61 avenue du président-Wilson, F-94235 Cachan Cedex
goubault@lsv.ens-cachan.fr
Phone: +33-1 47 40 75 68 Fax: +33-1 47 40 75 21

October 6, 2005

Ce document est la retranscription de notes de cours par Béatrice Bérard sur l'architecture des ordinateurs, datant de 1993. Ces notes s'inspiraient du livre d'A. Tannenbaum, "architecture des micro-ordinateurs". Ce cours présente les choses de façon suffisamment simple pour qu'on puisse se faire une idée raisonnablement claire de la façon dont fonctionnent les ordinateurs, depuis les circuits électroniques jusqu'à la micro-programmation.

Les processeurs aujourd'hui sont infiniment plus compliqués, les mémoires vives ne fonctionnent plus à base de flip-flops, soyez prévenus. Mais ceci reste une excellente introduction. Je mettrai quelques commentaires en notes en bas de page au sujet de certains points que je souhaute préciser, précédées de "[JGL]".

Je remercie Béatrice de m'avoir fourni ses notes de cours. Toute faute ou omission est bien sûr de ma responsabilité.

– Jean Goubault-Larrecq, Cachan, 30 septembre 2005.

# 1 Couche physique: circuits logiques

## 1.1 Portes logiques et algèbre de Boole

La forme la plus élémentaire de circuit est la porte logique. Son comportement est dit binaire car il est caractérisé par deux états: l'état 0, qui représente la valeur logique faux et l'état 1, qui représente la valeur logique vrai.

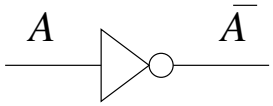
Les constructeurs utilisent dans certains cas une logique dite positive: l'état 1 correspond à une tension comprise entre 2 et 5 volts (niveau haut) et l'état 0 à une tension comprise entre 0 et 1 volt (niveau bas). Dans d'autres cas, ils utilisent une logique dite négative où l'état 1 correspond au niveau bas, tandis que l'état 0 correspond au niveau haut[1].

[1] [JGL] Plus un circuit contient de transistors, plus il est cadencé à une fréquence élevée, et plus sa tension

La transmission n'est pas instantanée: le délai de traversée d'une porte correspond au temps de propagation des signaux de l'entrée vers la sortie (de l'ordre de quelques nanosecondes pour la porte la plus simple[2]).

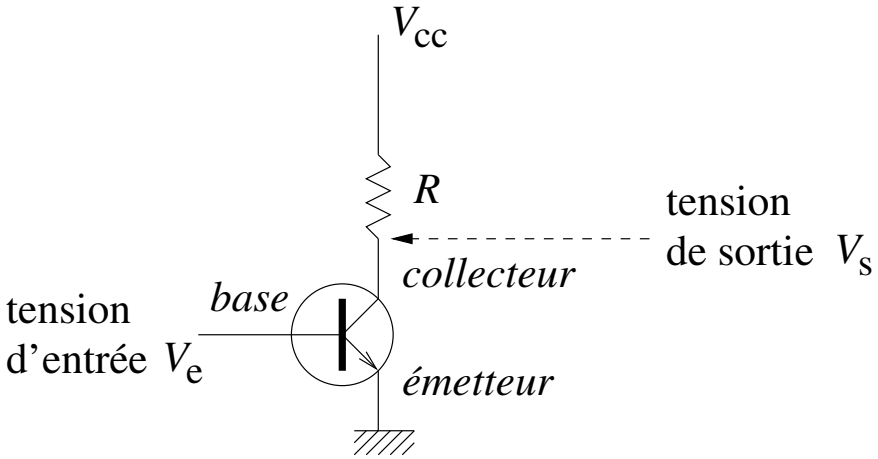
### 1.1.1 Porte NON

La porte logique la plus simple est celle qui réalise une inversion logique. Elle est symbolisée par:




Elle est construite à l'aide d'un transistor, selon deux techniques possibles: le transistor à jonctions (technologie dite bipolaire) ou le transistor à effet de champ (technologie dite unipolaire). Dans la première famille on trouve par exemple les circuits **RTL** (**R**esistor **T**ransistor **L**ogic), **TTL** (**T**ransistor **T**ransistor **L**ogic), **DTL** (**D**iode **T**ransistor **L**ogic) ou encore **ECL** (**E**mitter **C**oupled **L**ogic). La deuxième famille est celle des circuits **MOS** (**M**etal **O**xyde **S**emiconductor) qui sont en général moins rapides, sauf les plus récents comme **HMOS** ou **XMOS**[3].

Un transistor fonctionne comme un interrupteur[4]:




- Si la tension d'entrée $V_e$ est inférieure à une valeur critique, le transistor se bloque et n'est pas conducteur; l'interrupteur est ouvert et la tension de sortie $V_s$ est proche de $V_{cc}$, donc à un niveau haut.

d'alimentation est élevée, plus il consomme. Les processeurs modernes contiennent un nombre gigantesque de transistors: le Pentium IV Prescott, à la date d'aujourd'hui, contient 125 millions de transistors. Il tourne de plus à 3,2GHz. Pour diminuer la consommation électrique, donc la chaleur engendrée (il faut en premier lieu éviter que le processeur fonde! Un Pentium IV Prescott consomme entre 100 et 250 watts; poussé à 3,57GHz, il monte de 45 degrés à 94 degrés, et brûle, voir l'article), sa tension d'alimentation est basse: de 1,3, voire de 1,0 volt selon les modèles.

[2][JGL] Beaucoup moins aujourd'hui. Noter qu'un processeur cadencé à 3,2 GHz reçoit un tic d'horloge tous les $1/3,2.10^9 \approx 0,3ns$. Noter aussi qu'un processeur à cette fréquence doit être petit: la lumière ne parcourt que 10 cm pendant ces $0,3ns$!

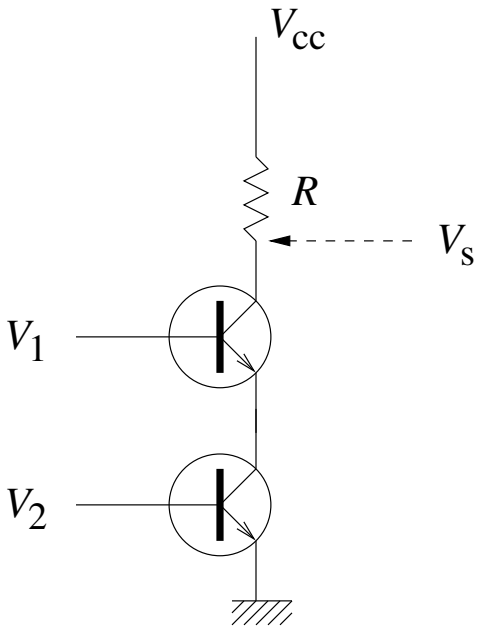
[3][JGL] Aujourd'hui, tous les processeurs sont en technologie MOSFET, ou des améliorations.

[4][JGL] Ici, un transistor à jonction NPN. Il en existe aussi des PNP, qui auraient la flèche de l'émetteur orientée dans l'autre sens.

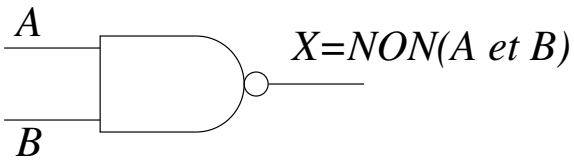
- Si la tension d'entrée $V_e$ est supérieure à la valeur critique, le transistor bascule, ce qui le rend conducteur; l'interrupteur est fermé et la tension de sortie est au niveau bas, proche de la masse. La résistance $R$ est conçue pour limiter le courant à travers le transistor, dans ce dernier cas.

### 1.1.2 Porte NON-ET

La porte NON-ET est réalisée en reliant deux transistors en série, selon le schéma suivant:



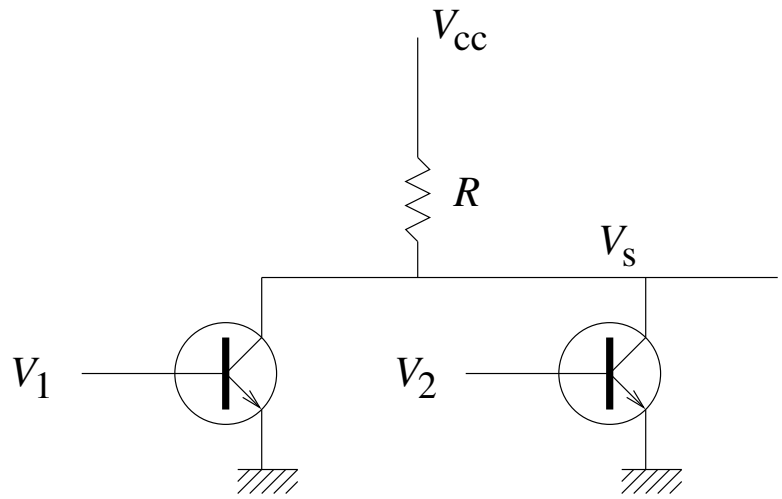

Symbole logique de la porte NON-ET:




On vérifie facilement que la tension de sortie $V_s$ est au niveau bas si et seulement si $V_1$ et $V_2$ sont au niveau haut.

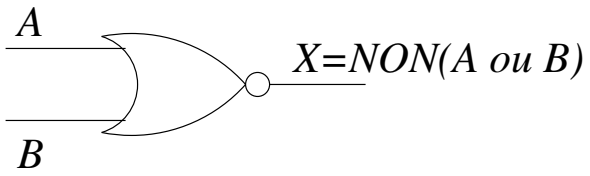
### 1.1.3 Porte NON-OU

Sur le même principe, le montage de deux transistors en parallèle permet d'obtenir une porte NON-OU:




Sur le schéma précédent, on vérifie que $V_s$ est au niveau bas si et seulement si l'une des deux tensions $V_1$ ou $V_2$ est au niveau haut.

Le symbole logique de cette porte est:




Ainsi, pour réaliser un OU (respectivement un ET), il faut relier un NON-OU avec un inverseur (respectivement un NON-ET avec un inverseur).

Ces portes sont utilisées pour construire des fonctions booléennes à variables booléennes, qui sont décrites soit par une table de vérité, soit en utilisant une notation fonctionnelle où le symbole $+$ remplace OU, le point ou aucun symbole représente ET et une variable comme $A$ surmontée d'une barre représente NON $A$.

**Exemple 1** *Construire la fonction $f$ définie par $f(A, B, C) = M$, où $M = 1$ si et seulement si au moins deux des variables valent* 1.

**Remarque 2** *Avec une porte de type NON-OU (ou NON-ET), on peut réaliser n'importe quelle fonction booléenne.*

On cherche souvent à réduire le nombre de portes, en utilisant les propriétés algébriques des opérations logiques, par exemple la distributivité de ET sur OU: $A(B + C) = AB + AC$.

En fait, on dispose de circuits qui réalisent directement des fonctions logiques complexes.

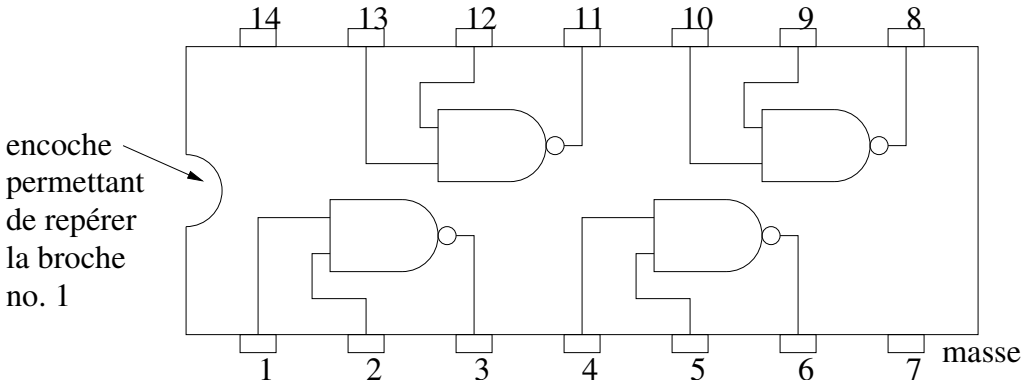
## 1.2 Principe des circuits logiques

Un circuit intégré est une plaquette de silicium sur laquelle sont intégrées les portes du circuit. La plaquette est encapsulée dans un boîtier avec sur les côtés des broches permettant les connexions électriques.

Ces circuits sont classés suivant la densité d'intégration, c'est-à-dire le nombre de portes ou transistors par circuits (ou par $mm^2$):

| | | |
|---|---|---|
| **SSI** | **S**mall **S**cale **I**ntegration | 1 à 10 portes par circuit |
| **MSI** | **M**edium **S**cale **I**ntegration | 10 à 100 |
| **LSI** | **L**arge **S**cale **I**ntegration | 100 à 100 000 |
| **VLSI** | **V**ery **L**arge **S**cale **I**ntegration | plus de 100 000 et jusqu'à 1 million[5] |

**Exemple 3** *Un circuit SSI dans un boîtier à 14 broches: circuit TTL 7400 de Texas Instruments.*




Bien entendu, plus l'échelle d'intégration augmente, plus il y a de portes par rapport au nombre de broches.

- MSI: de 5 à 10 portes par broche;
- LSI: de 25 à 100;
- VLSI: plusieurs milliers.

On distingue plusieurs catégories de circuits MSI qui vont être décrits dans les paragraphes suivants.

## 1.3 Les circuits combinatoires

Dans ces circuits, la sortie est une expression logique des valeurs d'entrée.

### 1.3.1 Multiplexeur

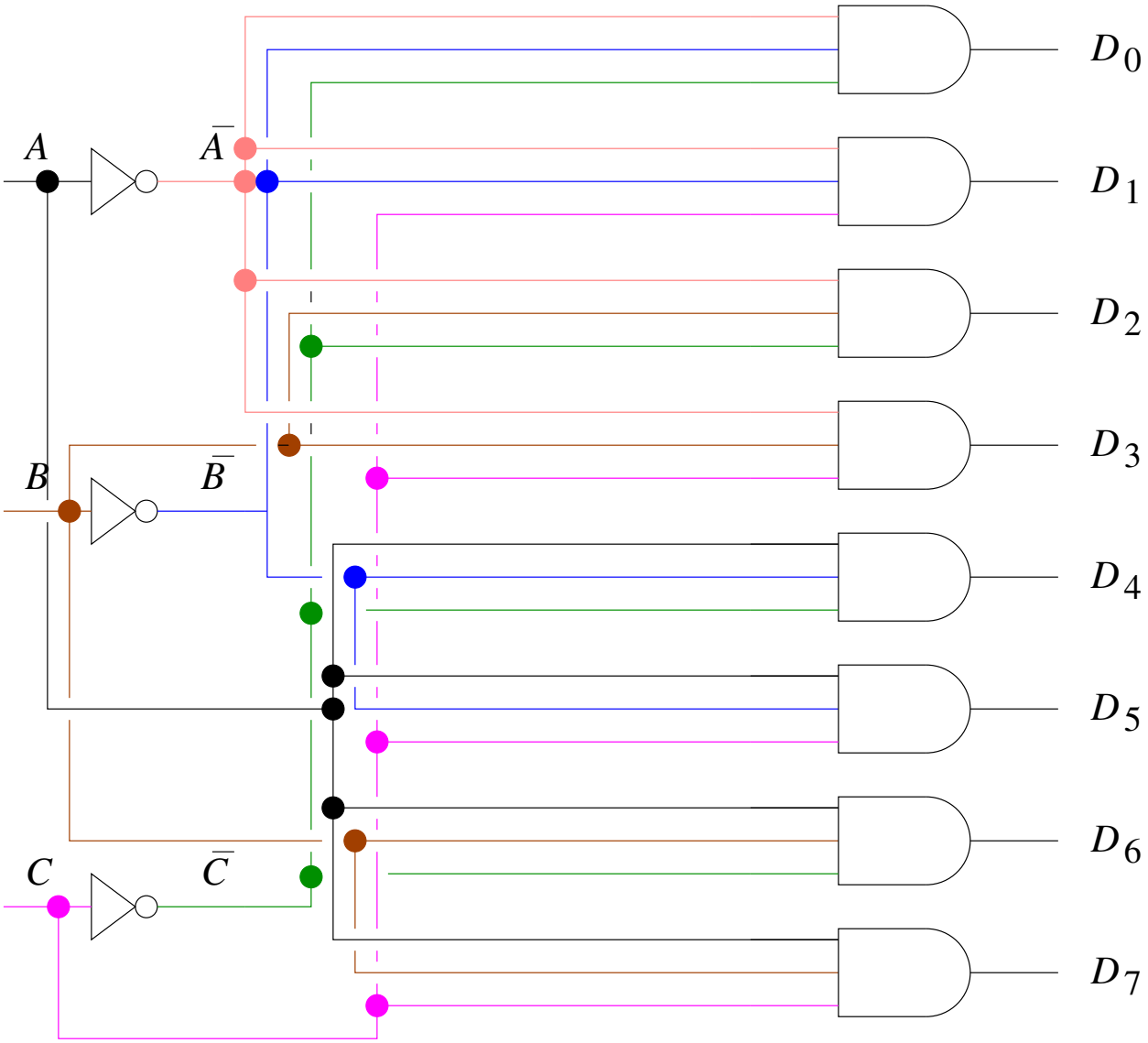
Un multiplexeur comporte $2^n$ entrées, $1$ sortie et $n$ lignes de sélection. La configuration des $n$ lignes de sélection fournit une valeur parmi les $2^n$ entrées et connecte cette entrée à la sortie.

Par exemple, une conversion parallèle-série utilise un multiplexeur de la façon suivante: une information de $8$ bits arrive sur les $8$ lignes d'entrée appelées $D_0$, $D_1$, ..., $D_7$. Sur les trois lignes de sélection $A$, $B$, $C$, on diffuse séquentiellement les valeurs binaires dans l'ordre $000 = 0$ à $111 = 7$, de sorte que les $8$ bits du mot d'entrée sont transmis en série sur la ligne de sortie.

### 1.3.2 Décodeur

Un décodeur comprend $n$ entrées et $2^n$ sorties, la sortie activée correspondant à la configuration binaire du mot formé par les $n$ entrées. Un tel circuit sert à sélectionner des adresses de la mémoire.

Un décodeur 3 vers 8 ($n = 3$) est réalisé par:




Un tel décodeur est utilisé pour une mémoire de 8 mots. Pour une mémoire composée de 8 circuits de 8K chacun[6], le circuit 0 contient les octets d'adresses 0 à $8 \times 1\,024 - 1 = 8\,192 - 1 = 8\,191$, le numéro 1 de 8 192 à 16 383, etc. On sélectionne suivant les 3 bits de poids fort pour avoir le numéro de circuit.

[6] 1K, ou plutôt 1Ko, signifie 1 Kilo-octet, soit $1024 = 2^{10}$ octets.

### 1.3.3 Comparateur

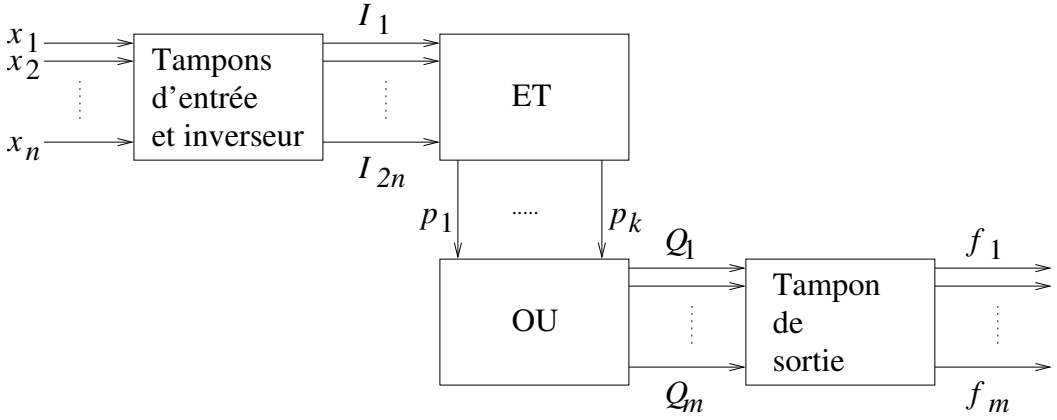
Un comparateur à $2n$ entrées et $1$ sortie, les $2n$ entrées formant deux mots de $n$ bits: $A$ et $B$. La sortie vaut $1$ si le mot $A = B$, sinon[7].

### 1.3.4 Circuits FPLA

Ce sont des réseaux logiques programmables (**F**ield **P**rogrammable **L**ogic **A**rray) qui permettent de réaliser des fonctions lorsqu'elles sont sous la forme d'une somme de produits.

Considérons par exemple $20$ broches dont $12$ correspondent à des entrées et $6$ à des sorties. Les $12$ entrées sont inversées, ce qui fournit $24$ variables internes. Ces $24$ variables sont toutes connectées à $50$ portes ET, ce qui donne $1\,200$ fusibles, au départ intacts.

Programmer le circuit consiste alors à détruire certains fusibles pour obtenir les produits de la fonction à programmer. Les $6$ sorties proviennent de $6$ portes OU qui reçoivent chacune les sorties des $50$ portes ET. On obtient ainsi $300$ fusibles dont certains sont détruits pour obtenir la somme des termes de la fonction.




## 1.4 Les circuits de calcul

### 1.4.1 Décaleur

Un décaleur est formé de $(n + 1)$ entrées $D_1, \ldots, D_n$, $C$ et de $n$ sorties $S_1, \ldots, S_n$ et opère un décalage de $1$ bit sur les entrées $D_1, \ldots, D_n$. Si $C = 1$, il s'agit d'un décalage à droite et si $C = 0$, d'un décalage à gauche[8].

### 1.4.2 Additionneur

Pour réaliser des additions de $2$ mots de $16$ bits, on utilise $16$ additionneurs à $1$ bit, reliés pour gérer la propagation éventuelle de retenues. Ces additionneurs sont eux-mêmes formées à l'aide de demi-additionneurs.

[7] Exercice: réalisez-le avec $n$ portes "ou exclusif" et $n - 1$ portes "et"

[8] Exercice: réalisez-le en circuit.

Ce demi-additionneur est donc constitué de deux portes, un OU exclusif pour le résultat et un ET pour la retenue:

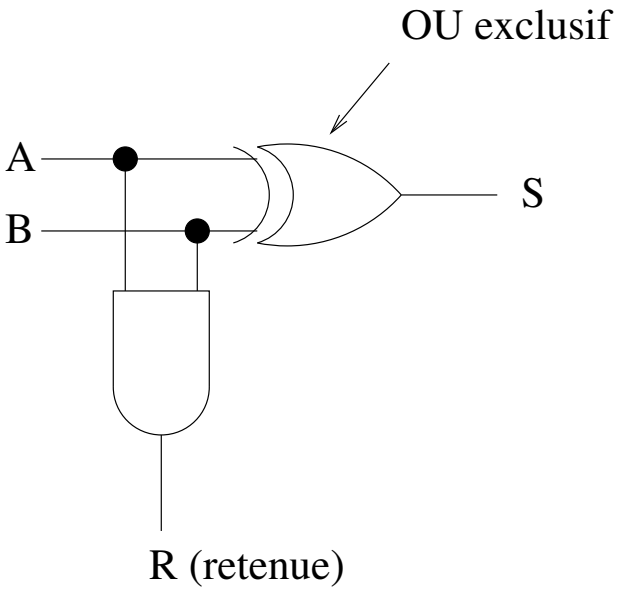


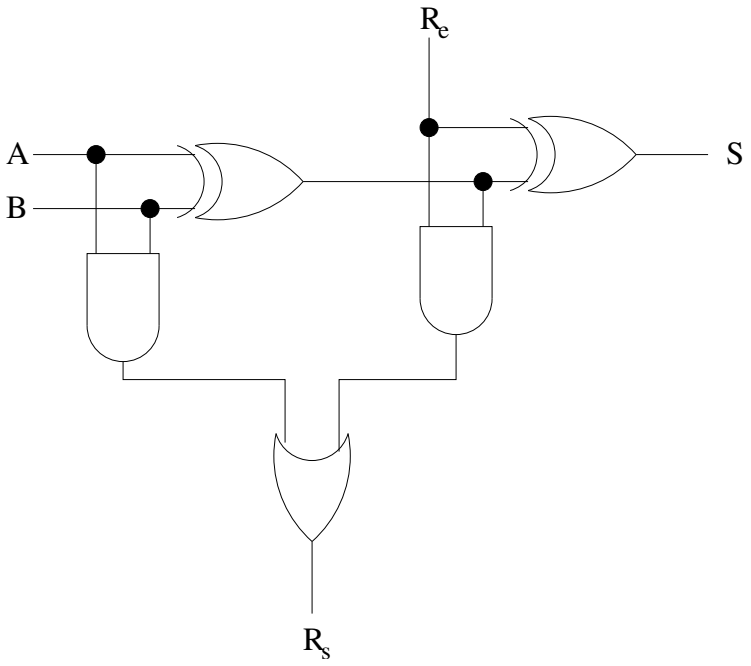

Table de vérité:

| $A$ | $B$ | $S$ | $R$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

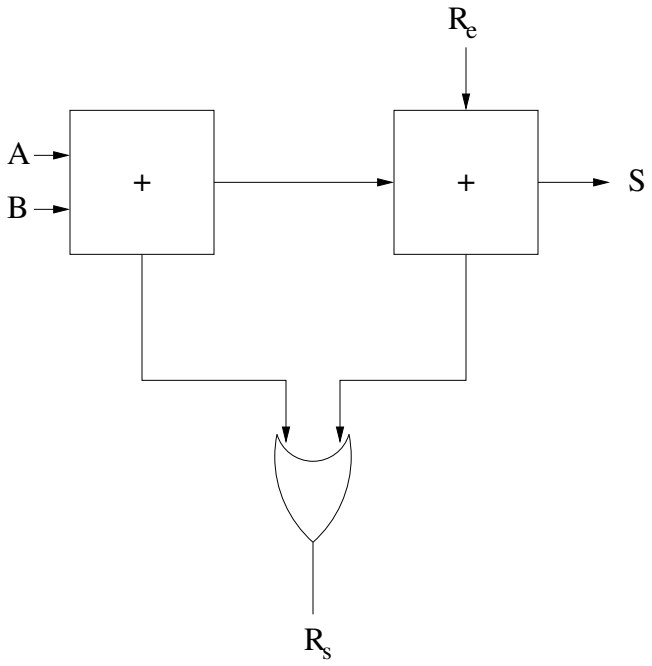
Afin de permettre une liaison de plusieurs additionneurs en série, un additionneur doit avoir une retenue en entrée $R_e$ en plus de la retenue en sortie $R_s$:

| $A$ | $B$ | $R_e$ | $S$ | $R_s$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

On peut vérifier que le circuit suivant réalise correctement l'additionneur.

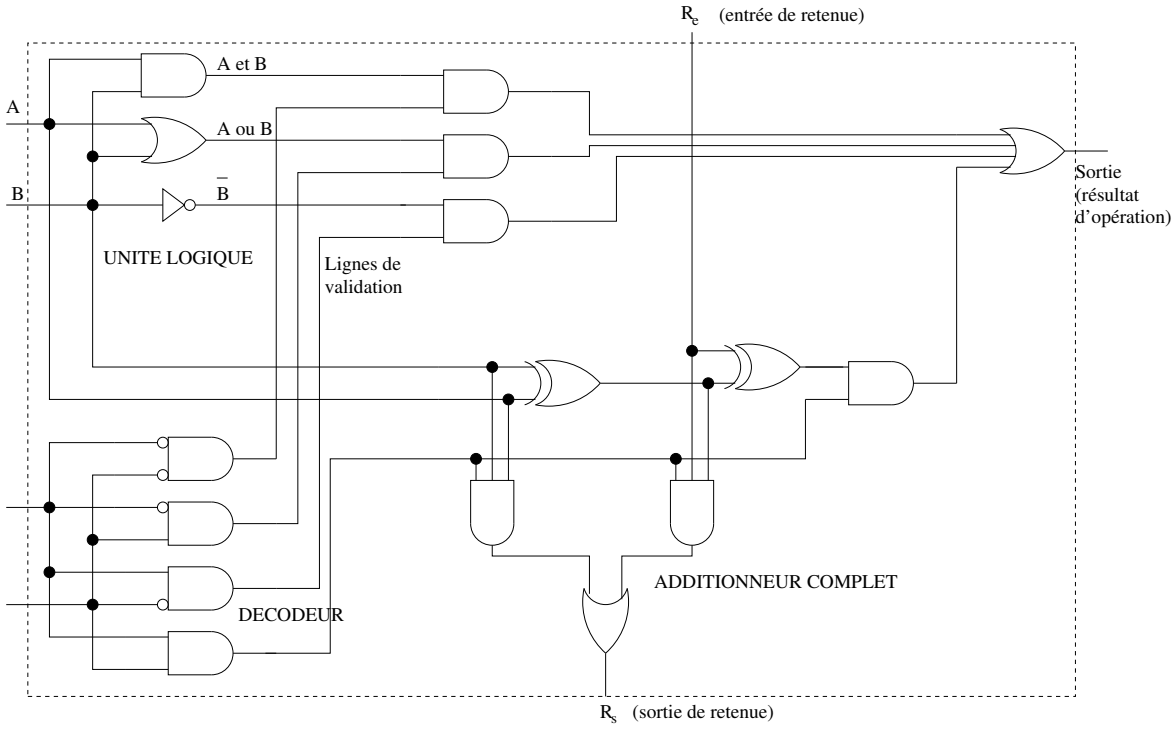



En représentant les deux demi-additionneurs par des carrés contenant le signe $+$, on obtient une représentation simplifiée de l'additionneur $1$ bit:



### 1.4.3 Unité arithmétique et logique

Exemple d'une UAL (**U**nité **A**rithmétique et **L**ogique) à un bit qui réalise ET, OU, NON, SOMME[9].




## 1.5 Circuits logiques à mémoire: les bascules

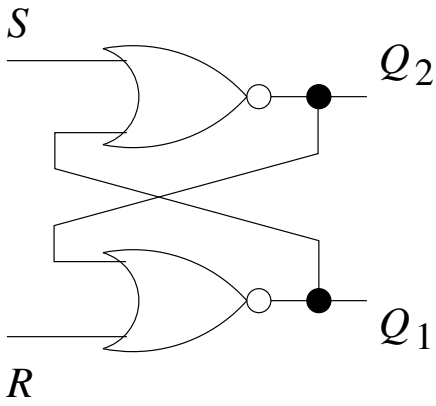
Appelés également circuits séquentiels ou flip-flops, les bascules mémorisent l'état antérieur des variables de sortie et permettent ainsi de mémoriser un bit.

### 1.5.1 Bascules RS (avec portes NON OU)

Ces circuits ont deux entrées: $S$ (set) qui correspond à la mise à $1$ et $R$ (reset) qui correspond à la remise à $0$. Ils ont également deux sorties: $Q_1$ et $Q_2$.

Une bascule doit en principe mémoriser celle des deux entrées $R$ et $S$ qui a été mise à $1$ le plus récemment.

[9] Les petits ronds en entrée des portes ET du décodeur sont des négations logiques. Cette UAL calcule $A$ et $B$, $A$ ou $B$, non $B$, ou bien $A + B$ selon que les bits sur les deux lignes en bas à gauche valent $00$, $01$, $10$, ou $11$.



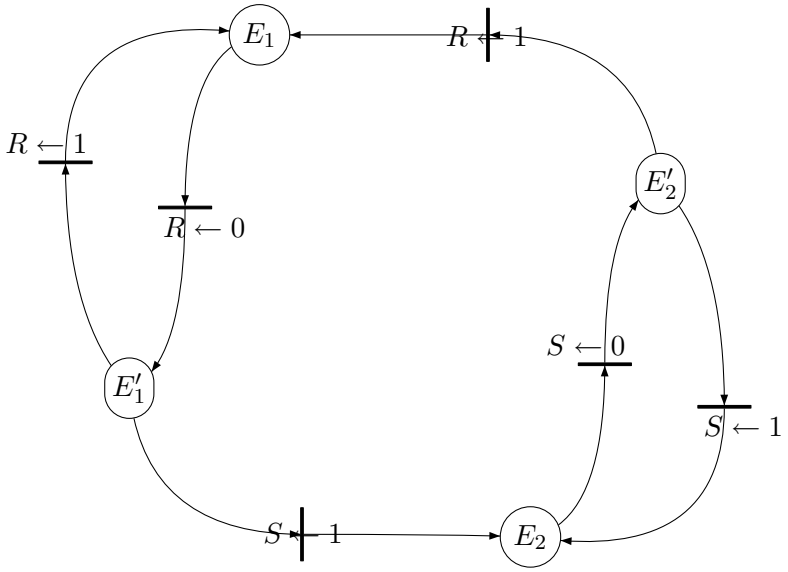

| $S$ | $R$ | $Q_1$ | $Q_2$ | |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | $E_1$ |
| 0 | 0 | 0 | 1 | $E'_1$ |
| 0 | 0 | 1 | 0 | $E'_2$ |
| 1 | 0 | 1 | 0 | $E_2$ |
| 1 | 1 | 0 | 0 | inutilisé |

Si $S = 0$ et $R = 1$, on vérifie qu'on passe dans l'état stable $E_1$ où $Q_1 = 0$ et $Q_2 = 1$. À partir de l'état $E_1$, si $R$ passe à 0, on obtient $S = 0$, $R = 0$, $Q_1 = 0$, $Q_2 = 1$, donc pas de changement[10].

À partir de ce même état, si $S$ passe à 1, on arrive à un état stable $E_2$: $S = 1$, $R = 0$, $Q_1 = 1$, $Q_2 = 0$. Si $S$ repasse à 0, cet état est inchangé.

On remarque que si $S = R = 0$, il est possible d'avoir soit $Q_1 = 1$, $Q_2 = 0$, soit l'inverse, ce qui conduit à n'utiliser que les états $E_1$, $E_2$, $E'_1$, $E'_2$, et pas $R = S = 1$.

Le fonctionnement et les états de la bascule sont représentés par un automate:



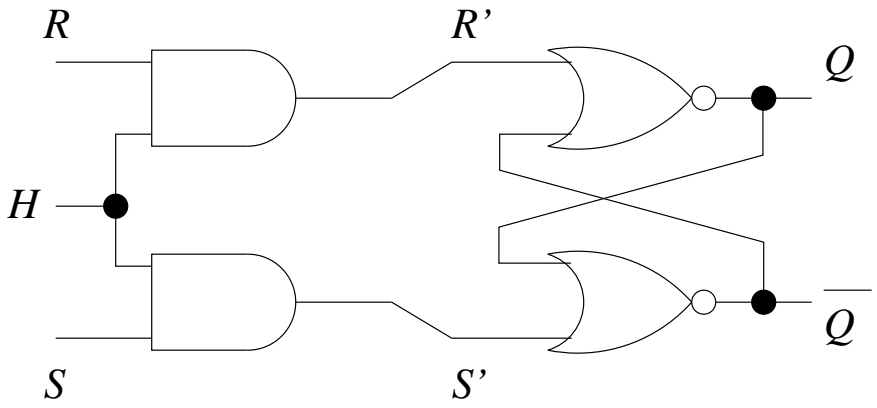

[10] sur les sorties $Q_1$ et $Q_2$.

En général, on représente $Q_1$ et $Q_2$ comme $Q$ et non $Q$, car dans les états *utiles*, $Q_2$ est la négation de $Q_1$. Il suffit alors de se donner une sortie $Q$ et la table de vérité d'une bascule décrit la valeur de l'état suivant en fonction de la valeur de l'état courant.

| $S$ | $R$ | $Q_{n+1}$ | (état suivant) |
|---|---|---|---|
| 0 | 1 | 0 | |
| 1 | 0 | 1 | |
| 0 | 0 | $Q_n$ | (état suivant=état précédent) |
| 1 | 1 | indéfini | |

### 1.5.2 Bascules RS (à horloges)

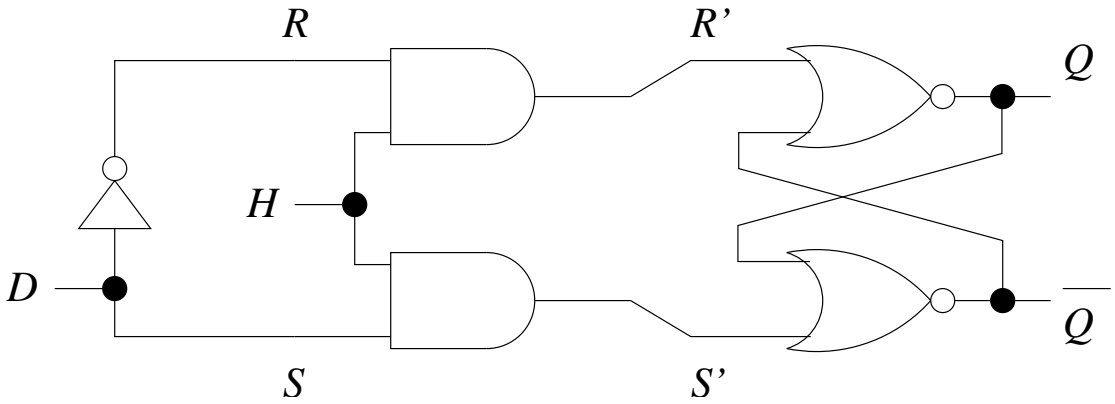
Quand $H = 0$, $R' = S' = 0$, il n'y a aucun changement d'état possible, quelles que soient les valeurs des entrées $R$ et $S$. En revanche, quand $H = 1$, $R' = R$ et $S' = S$.




Le nouvel état $Q_{n+1}$ est obtenu uniquement à l'issue d'un signal ($H = 1$) d'horloge.

### 1.5.3 Bascules D

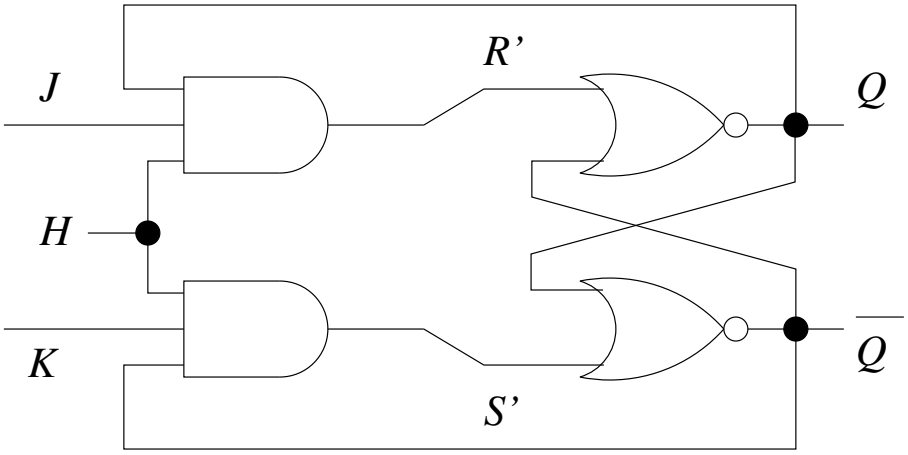
La bascule D est obtenue à partir de ce qui précède en utilisant une seule entrée $D$, avec $S = D$, $R =$ non $D$.



Cette bascule fournit en $Q$ la valeur de $D$ jusqu'à la prochaine impulsion de l'horloge. Ce mécanisme lève l'ambiguïté puisqu'on n'a jamais $R = S = 1$. Bien entendu, il y a encore d'autres bascules.

### 1.5.4 Bascules JK

Les bascules JK règlent aussi le problème de l'état indéterminé.




On a la table de transitions suivantes. L'état $Q$ ne change que lorsque l'entrée d'horloge $H$ est mise à 1.

| $J$ | $K$ | $Q_{n+1}$ |
|---|---|---|
| 0 | 0 | $Q_n$ |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | non$Q_n$ |

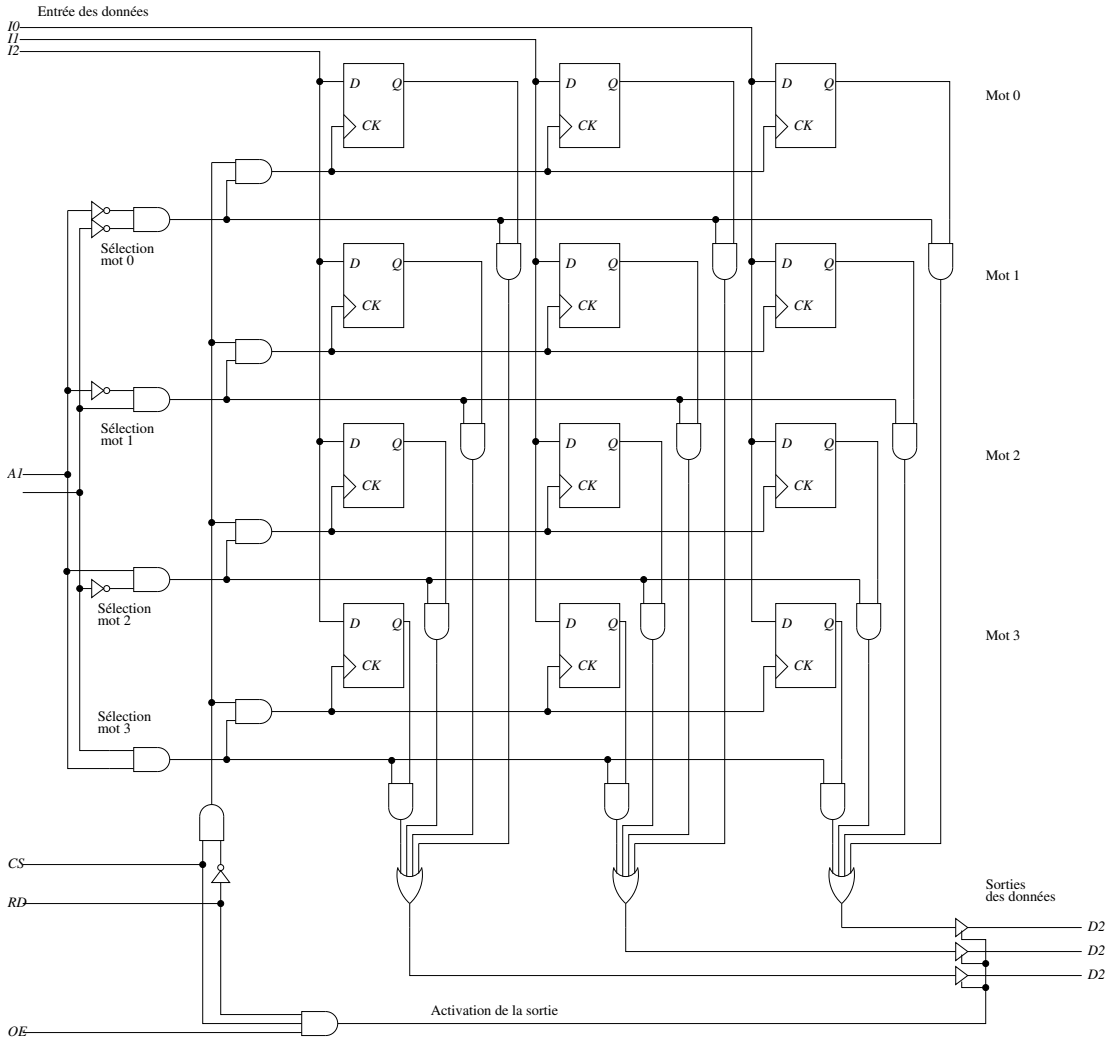
## 1.6 Structure d'une mémoire

En réunissant plusieurs bascules sur un même signal d'horloge, on peut fabriquer un circuit qui constitue un registre, d'où la possibilité de construire des mémoires[11]. Le problème est de minimiser le nombre de broches.

**Exemple 4** *Une mémoire de quatre mots de* 3 *bits chacun,* $M_0, \ldots, M_3$ *dans un boîtier à* 14 *broches, chaque mot de* 3 *bits étant formé à partir de* 3 *bascules*[12].

[11] Les mémoires actuelles ne sont plus fabriquées à l'aide de bascules, et stockent plusieurs bits d'information par transistor. Néanmoins, l'explication à base de bascules reste simple et compréhensible.

[12] Les bascules utilisées sont des bascules D. L'entrée d'horloge est ici nommée CK.




*8 entrées:*

- *3 entrées de données* $I0$, $I1$, $I2$*;*
- *2 entrées d'adresses* $A0$, $A1$ *(*$4$ *mots);*
- *3 entrées de commande:*
  - $CS$*: sélection du boîtier ("chip select") si* $CS = 1$*;*
  - $RD$*: lecture/écriture (*$1/0$*);*
  - $OE$*: activation des sorties ("output enable").*

*3 sorties de données* $D0$, $D1$, $D2$.

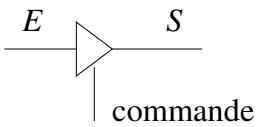
*Les lectures et les écritures sont réalisées de la manière suivante:*

- *si* $RD = 1$*, le mot lu à l'adresse indiquée par* $A0$ $A1$ *est positionné en sortie;*

- *si $RD = 0$, le mot d'entrée est chargé dans celui sélectionné par $A0$ $A1$.*

*Remarquons que $A0$ $A1$ et les entrées des quatre mots forment un décodeur.*

*Sur les lignes de sortie, on utilise des circuits qui jouent le rôle d'interrupteurs*[13] *et comportent* 3 *états:* 0, 1, *et un état de haute impédance qui déconnecte le circuit lorsque la commande vaut* 0. *Circuit* 3 *états:*




*Si les* 3 *signaux $RD$, $CS$, $OE$ sont à* 1, *les sorties sont activées, sinon elles sont déconnectées par les circuits trois états.*

Ce schéma se généralise avec un nombre de mots égal à $2^n$ en utilisant $n$ lignes d'adresses $A_0, \ldots, A_{n-1}$. La taille d'un mot correspond au nombre de bascules, au nombre d'entrées et au nombre de sorties.

La structure décrite ci-dessus concerne les mémoires statiques. Pour les mémoires dynamiques, la structure interne n'utilise pas des bascules mais des condensateurs[14]. La capacité est plus grande, mais la charge électrique baisse avec le temps, d'où la nécessité de rafraîchir. Il existe différentes variantes intermédiaires entre les mémoires dynamiques[15] et les mémoires à lecture seulement (ROM, "read only memory"), programmables une seule fois à la fabrication[16].

## 1.7 Microprocesseur

Il est relié aux mémoires et aux circuits d'entrée-sortie par des bus, c'est-à-dire des lignes groupées: au moins $n$ lignes d'adresses dans le bus d'adresses pour $2^n$ mots mémoire et $p$ lignes de données dans le bus de données si les mots ont chacun $p$ bits.

**Exemple 5** *Le microprocesseur Z80*[17]*:* 16 *lignes d'adresses $A_0, \ldots, A_{15}$ permettant d'adresser* 64 *Koctets*[18]*, des mots de* 8 *bits correspondant à* 8 *lignes de données $D_0, \ldots, D_7$,* 13 *lignes de commande.*

[13] On les appelle des "latch".

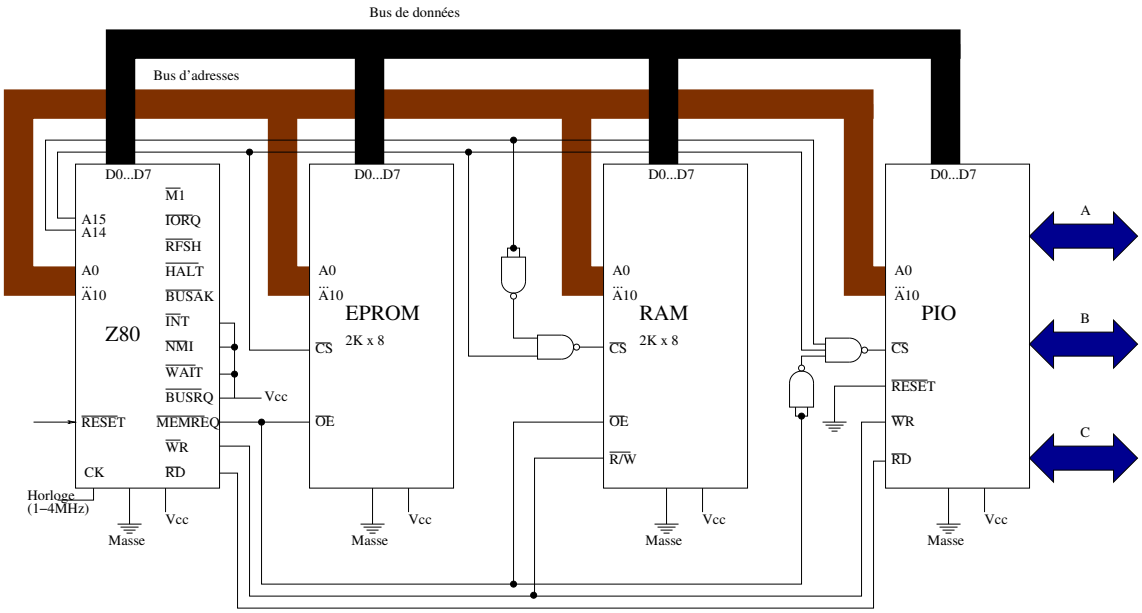
[14] En fait des transistors à effet de champ, dans lesquels on piège des électrons (ou des trous, c'est-à-dire des absences d'électrons), ce qui revient effectivement à les voir comme des condensateurs de très faible capacité.

[15] et aujourd'hui les mémoires statiques-dynamiques (SDRAM), les innombrables variantes de mémoire vidéo, etc.

[16] mais il existe aussi les PROM ("programmable ROM"), qui sont programmables une fois après fabrication, par destruction de fusibles, les EPROM ("erasable PROM"), que l'on peut effacer en totalité typiquement par une exposition prolongée à des rayons ultraviolets, puis reprogrammer, les EEPROM ("electrically erasable PROM"), que l'on peut effacer en totalité par une impulsion électrique, puis reprogrammer, les mémoires FLASH, qui sont intermédiaires entre les EEPROM et les mémoires vives (RAM, "random access memory") que nous avons décrites dans cette section. On peut lire et écrire dans une FLASH à volonté, mais le temps d'accès est plus long que pour les RAM. Ceci permet d'utiliser les mémoires FLASH comme des substituts rapides à des disques durs.

[17] Un microprocesseur très populaire à la fin des années 1970 et jusque vers 1985, de la défunte firme Zilog—laquelle avait été fondée par des transfuges de chez Intel.

[18] Rappel: 1 Koctet égale $2^{10} = 1\ 024$ octets, donc 64 Koctets égalent $2^{16} = 65\ 536$ octets. Un octet est un mot de $p = 8$ bits.




*Les bus de d'adresses et de données sont actifs au niveau haut* (1)*, tandis que les signaux de commande sont actifs au niveau bas* (0)*. Commandes:*

- $\overline{\text{MEMREQ}}$*: demande d'accès mémoire;*
- $\overline{\text{IORQ}}$*: demande d'accès entrée-sortie;*
- $\overline{\text{RD}}$*: lecture;*
- $\overline{\text{WR}}$*: écriture;*
- $\overline{\text{WAIT}}$*: attente de l'activation d'un circuit;*
- $\overline{\text{BUSRQ}}$*: demande du bus par un circuit d'entrée-sortie;*
- $\overline{\text{BUSAK}}$*: attribution du bus à ce circuit;*
- $\overline{\text{INT}}$*: interruption masquable;*
- $\overline{\text{NMI}}$*: interruption non masquable;*
- $\overline{\text{RESET}}$*: mise à* 0*;*
- $\overline{\text{RFSH}}$*: rafraîchissement;*
- $\overline{\text{HALT}}$*: arrêt (pour l'extérieur);*
- $\overline{\text{M1}}$*: décodage du code opération d'une instruction.*

# 2 Couche microprogrammée

Dans ce chapitre, on considère une mémoire centrale de $2^{12}$ mots de $16$ bits, comportant une pile d'exécution, reliée à un bus d'adresses à $12$ lignes et à un bus de données à $16$ lignes. On suppose qu'un programme en langage machine est chargé dans cette mémoire centrale.

Le microprogramme est un programme qui réside dans une mémoire locale du (micro) processeur: c'est un interpréteur qui décode et exécute une par une les instructions en langage machine[19], en répétant le schéma général suivant:

- transfert d'une instruction de la mémoire centrale vers un registre spécifique du processeur ($RI$, comme Registre d'Instruction);
- modification du compteur ordinal[20], qui est positionné à l'adresse suivante de la mémoire centrale;
- décodage de l'instruction: extraction du code opération et éventuellement des adresses des données en mémoire centrale;
- transfert éventuel des données de la mémoire centrale vers des registres spécifiques du processeur;
- exécution proprement dite de l'instruction, qui peut utiliser l'UAL;
- transfert éventuel du résultat vers la mémoire centrale.

Avant de revoir le détail de ces opérations, nous décrivons la structure d'une micromachine et les micro-instructions.

## 2.1 Structure de la micromachine

La micromachine contient deux parties: le chemin des données et le bloc de contrôle. Voir la figure 1.

### 2.1.1 Le chemin des données

Il comprend (voir la partie gauche de la figure 1):

- $16$ registres généraux à $16$ bits. Chacun de ces registres peut soit être chargé par une donnée en provenance du bus $C$, soit transférer son contenu sur l'un ou l'autre des deux bus $A$ et $B$. On compte les registres suivants:

[19] Tous les microprocesseurs n'ont pas une couche microprogrammée: c'était par exemple le cas des processeurs RISC (Reduced Instruction Set Computer) des années 1980. Les processeurs modernes sont microprogrammés, c'est le cas des Pentium. C'est aussi le cas de petits processeurs appelés PIC, dont on peut reprogrammer le micro-code. Il se trouve qu'on peut en fait reprogrammer en partie le micro-code des Pentium.

[20] Le registre `%eip` dans le Pentium, par exemple.

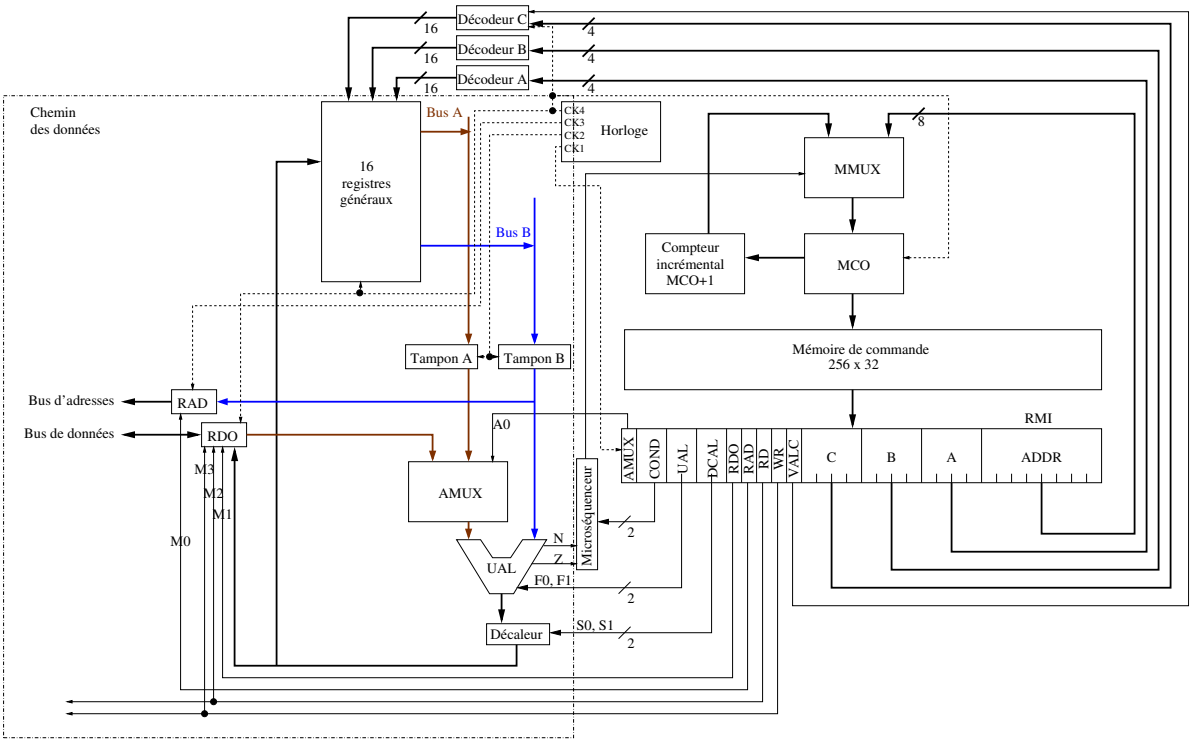



Figure 1: Une machine microprogrammée

- Les registres **A**, **B**, **C**, **D**, **E**, **F** sont des registres (vraiment) généraux.
- Le registre **CO** contient la valeur du **C**ompteur **O**rdinal, c'est-à-dire l'adresse (en mémoire centrale) de l'instruction suivante du programme en langage machine.
- Le registre **AC** est un registre **AC**cumulateur, utilisé pour conserver provisoirement la valeur d'une donnée.
- Le registre **PP** contient le **P**ointeur de **P**ile, c'est-à-dire l'adresse du sommet de la pile d'exécution, située en mémoire centrale (la pile étant en général inversée, elle est d'autant plus grande que cette adresse est basse).
- Le registre **RI** (**R**egistre d'**I**nstruction) contient une copie de l'instruction en cours de décodage et d'exécution.
- Le registre **RIT** contient une copie plus ou moins exacte de RI[21].
- Les registres **0**, **+1**, et **-1** contiennent les valeurs constantes correspondantes[22].
- Le registre **AMASQ** est un masque d'adresse, il contient la constante `0x0fff`, c'est-à-dire $0fff$ en hexadécimal, tandis que le registre **PMASQ** est un masque de pile, qui

[21] C'est-à-dire que c'est un endroit où le microprogramme va sauvegarder RI, ou bien utiliser comme registre auxiliaire pour calculer RI. Le "T" à la fin de "RIT" est pour **T**emporaire.

[22] Ce sont donc en particulier des registres non modifiables.

contient la constante `0x00ff`[23].

Les registres généraux sont numérotés:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CO** | **AC** | **PP** | **RI** | **RIT** | **0** | **+1** | **-1** | **AMASQ** | **PMASQ** | **A** | **B** | **C** | **D** | **E** | **F** |

- Un multiplexeur AMUX, qui sélectionne une entrée parmi deux, avec la commande A0. Les deux entrées possibles sont le contenu du tampon A, en sortie du bus $A$, et le registre **RDO**, registre de données qui assure le transfert de données entre le microprocesseur et la mémoire centrale.

- Une UAL avec $4$ opérations possibles (sur des mots de $16$ bits). Une opération sur les entrées A (en sortie de AMUX) et B (provenant du tampon B) est sélectionnée par la configuration des deux lignes F0 et F1:

  0 $A + B$
  1 $A$ ET $B$
  2 $A$
  3 NON $A$

  Outre le résultat de l'opération, l'UAL comporte deux sorties Z et N qui valent 1 respectivement lorsque le résultat est nul ou lorsqu'il est négatif (c'est-à-dire lorsque le bit de poids fort de ce résultat vaut $1$).

  Les opérandes de l'UAL sont maintenus stables dans les deux tampons d'entrée A et B, dont le chargement (par les valeurs présentes sur les bus $A$ et $B$) est commandé par les signaux L0 et L1.

- Un décaleur avec deux commandes S0 et S1 dont la configuration indique le sens du décalage:
  0 pas de décalage
  1 décalage à droite
  2 décalage à gauche
  3 inutilisé

- Deux registres qui réalisent la communication avec la mémoire centrale:

  – Le registre d'adresses **RAD** à $12$ bits, qui prend en entrée les $12$ bits de poids faible de la donnée en provenance du tampon B, selon la commande M0, et dont le contenu peut être placé en sortie sur le bus d'adresses vers la mémoire centrale.

  – Le registre de données **RDO** à $16$ bits, qui peut être chargé par la donnée du bus $C$ en sortie du décaleur (commande M1). L'écriture en mémoire centrale est réalisée par le positionnement du contenu de **RDO** sur le bus de données (commande M2 ou WR)

[23] Ce microprocesseur-jouet ne peut accéder qu'à des adresses sur $12$ bits: pour toute adresse $a$, l'adresse effective envoyée sur le bus d'adresses sera le ET bit à bit de $a$ avec **AMASQ**. La pile est elle stockée sur les $256$ premiers octets, et est donc aux adresses de la forme $a$ ET **PMASQ**.

tandis que la lecture en mémoire correspond au chargement de **RDO** par le bus de données (commande M3 ou RD).

Remarquons que **RDO** est toujours positionné en sortie vers l'unité centrale, puisque la commande A0 du multiplexeur AMUX permet de choisir entre le contenu de **RDO** et celui du tampon A pour l'entrée A de l'UAL.

0 tampon A
1 **RDO**

### 2.1.2 Le bloc de contrôle

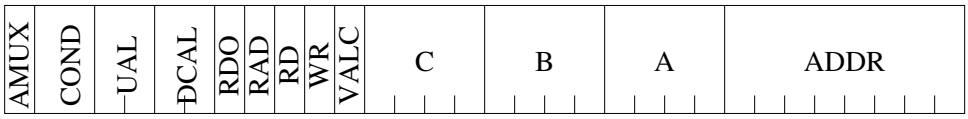
Il comprend (voir la partie droite de la figure 1):

- Trois décodeurs $4$ vers $16$, sélectionnant un des $16$ registres généraux pour les transferts avec les bus $A$, $B$, $C$.
- Une horloge qui assure la synchronisation. Cette horloge comporte trois retards successifs, de façon à diviser un cycle en $4$ sous-cycles, H1, H2, H3, H4.
- Une mémoire de commande de $256$ mots de $32$ bits chacun, qui contient le microprogramme, c'est-à-dire une suite de micro-instructions (pas plus de $256$) de $32$ bits chacune.
- Un registre **MCO**, micro-compteur ordinal, qui contient l'adresse de la micro-instruction suivante[24].
- Un compteur incrémental, circuit qui ajoute $1$ au contenu du registre **MCO**.
- Un multiplexeur MMUX, qui choisit une entrée parmi:
  - le résultat provenant du compteur incrémental, c'est-à-dire MCO+1 (adresse de la prochaine micro-instruction) et
  - une autre adresse correspondant généralement à un saut dans le micro-programme. La commande de ce multiplexeur est constituée par la sortie d'un micro-séquenceur.
- Un registre de micro-instruction, qui contient la micro-instruction en cours d'exécution. Ce registre comporte donc $32$ bits et nous l'examinons maintenant en détail.

## 2.2 Les micro-instructions

### 2.2.1 Structure d'une micro-instruction

Une micro-instruction est donc un mot de $32$ bits comportant $13$ champs:




[24] L'adresse à l'intérieur de la mémoire de commande, s'entend, pas de la mémoire centrale.

- Le bit AMUX forme la commande A0 du multiplexeur de même nom.
- Le champ COND comporte deux bits dont la valeur binaire permet de contrôler des sauts éventuels:
  0 pas de saut
  1 saut si N $= 1$
  2 saut si Z $= 1$
  3 saut sans condition
- Les $2$ bits du champ UAL correspondent aux commandes F0 et F1 de l'unité arithmétique et logique et permettent le choix de l'opération à effectuer.
- Les $2$ bits du champ DCAL sont reliés aux commandes S0 et S1 du décaleur.
- Les $5$ champs suivants contiennent chacun un bit et correspondent respectivement aux commandes suivantes:
  - RDO: M1 $= 1$ autorise le chargement de **RDO** par la donnée du bus $C$.
  - RAD: M0 $= 1$ autorise le chargement de **RAD** par la donnée du tampon B.
  - RD: M3 $= 1$ autorise le chargement de **RDO** par le bus de données.
  - WR: M2 $= 1$ autorise le transfert de **RDO** sur le bus de données.
  - VALC: à $1$, autorise le chargement d'un des $16$ registres généraux par la donnée du bus $C$.
- Les trois champs C, B, A comportent $4$ bits chacun:
  - Les $4$ bits de C forment les entrées du décodeur C et sélectionnent un des $16$ registres généraux où décharger la donnée du bus $C$.
  - Les $4$ bits de B (resp. de A) sont les entrées du décodeur B (resp. A) et sélectionnent un des $16$ registres généraux dont le contenu est transféré sur B (resp. A).
- Le champ ADDR contient une adresse de saut éventuel sur $8$ bits.

La figure 2 montre quelques exemples de micro-instructions. La colonne de gauche indique de façon fonctionnelle (dans un langage de type Pascal) les opérations à effectuer et les colonnes suivantes contiennent les valeurs des bits de la micro-instruction correspondante.

Dans la dernière ligne du tableau, par exemple, les opérations sont

```
rit:=decalg(ri+ri); if n goto 69;
```

La micro-instruction associée comporte donc:

- `ri+ri`: les valeurs $3$ dans les champs A et B commandent le transfert du contenu du registre **RI** (de numéro $3$) sur les bus $A$ et $B$, de façon à positionner ces valeurs en entrée de l'UAL.

| | AMUX | COND | UAL | DCAL | RDO | RAD | RD | WR | VALC | C | B | A | ADDR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `rad:=co; rd;` | 0 | 0 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 00 |
| `rd;` | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 00 |
| `ri:=rdo;` | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 0 | 0 | 00 |
| `co:=co+1;` | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 6 | 0 | 00 |
| `rad:=ri; rdo:=ac; wr;` | 0 | 0 | 2 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 3 | 1 | 00 |
| `ual:=rit; if n goto 15;` | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 15 |
| `ac:=inv(rdo);` | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 00 |
| `rit:=decalg(rit); if n goto 25;` | 0 | 1 | 2 | 2 | 0 | 0 | 0 | 0 | 1 | 4 | 0 | 4 | 25 |
| `ual:=ac; if z goto 25;` | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 22 |
| `ac:=bet(ri,amasq); goto 0;` | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 8 | 3 | 00 |
| `pp:=pp-1; rd;` | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 2 | 7 | 00 |
| `rit:=decalg(ri+ri); if n goto 69;` | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 4 | 3 | 3 | 69 |

Figure 2: Quelques exemples de micro-instructions

- En général, la valeur contenue par défaut dans un champ est $0$, mais ici, la valeur $0$ dans le champ UAL signifier l'exécution d'une addition, qui permet dans ce cas d'effectuer un premier décalage à gauche sur le contenu du registre **RI**.
- `decalg`: la valeur $2$ dans le champ DCAL permet d'obtenir le décalage à gauche.
- `rit:=`: le bit $1$ dans le champ VALC valide le déchargement du résultat (en sortie du décaleur) situé sur le bus $C$ dans un des registres généraux (ici **RIT**.
- La valeur $4$ dans le champ C indique l'adresse du registre **RIT** (rappelons que les numéros de registre vont de $0$ à $15$) où déposer la donnée du bus $C$.
- `if n`:la valeur $1$ dans le champ COND commande un saut lorsque N est égal à $1$. Si les bits sont numérotés de $0$ à $15$, le bit $15$ étant celui de poids le plus fort, ceci se produit lorsque le bit numéro $14$, qui est le bit de poids fort du résultat de l'addition `ri+ri`, vaut $1$.
- `goto 69`: la valeur $69$ dans le champ ADDR contient l'adresse du saut (rappelons qu'il s'agit d'une adresse du micro-programme, donc de la mémoire de commande).

### 2.2.2 Exécution d'une micro-instruction

L'exécution est divisée en quatre étapes, correspondant chacune à un des quatre sous-cycles de l'horloge.

- Pendant le premier sous-cycle, la micro-instruction est transférée de la mémoire de commande vers le registre de micro-instruction (**RMI**). Ainsi, pour la mémoire de commande,

le registre **RMI** joue le rôle d'un registre de donnée, tandis que le registre **MCO** (micro-compteur ordinal) joue le rôle d'un registre d'adresse.

- Pendant le deuxième sous-cycle, le compteur incrémental ajoute $1$ au contenu de **MCO** et les valeurs des champs A et B sélectionnent, par l'intermédiaire des décodeurs, les registres généraux dont les contenus sont transférés sur les bus, puis sur les tampons A et B.

- Pendant le troisième sous-cycle, les opérations sont exécutées par l'UAL et le décaleur, avec comme opérandes d'une part le tampon B et d'autre part la sortie du multiplexeur qui est soit le tampon A soit le contenu du registre de donnée **RDO**. Lorsque le champ RAD est à $1$, la donnée du tampon B est transférée dans le registre d'adresse **RAD**.

- Pendant le quatrième sous-cycle, la donnée du bus $C$ en sortie du décaleur peut être déchargée dans un des registres généraux (désigné par le champ C), lorsque VALC vaut $1$, et dans **RDO**, si le champ RDO vaut $1$. le décodeur C est donc relié à la fois aux champs C et VALC de la micro-instruction et au quatrième signal d'horloge CK4.

## 2.3 L'interprétation des instructions en langage machine

### 2.3.1 Les instructions du langage machine

Un ensemble d'instructions est représenté dans le tableau de la figure 3.

La première colonne de ce tableau contient le code binaire de l'instruction. Remarquons que les instructions sont rangées par ordre croissant suivant leur code. Ce code est en général divisé en deux parties, la première représentant l'opération proprement dite et la seconde une adresse de la mémoire centrale ou une constante. Dans la deuxième colonne figure le symbole correspondant, tandis que la troisième colonne contient le nom de l'instruction et la quatrième une description fonctionnelle de cette instruction dans un langage du type Pascal. Dans cette colonne, le mot de la mémoire centrale d'adresse $x$ est noté $m[x]$.

Considérons par exemple l'instruction de symbole JUMP. Le code binaire de cette opération commence par $0110$, suivi de l'adresse en mémoire centrale (sur $12$ bits) où le branchement doit avoir lieu. La description fonctionnelle consiste simplement en une instruction: $co := x$, qui signifie que cette adresse de la mémoire centrale doit être copiée dans le compteur ordinal.

Certaines opérations ont un code opération sur $16$ bits, car elles ne nécessitent pas d'information supplémentaire: par exemple l'instruction PUSH ajoute le contenu du registre **AC** au sommet de la pile d'exécution. La pile considérée dans ce chapitre est située à des adresses décroissantes de la mémoire centrale, il faut donc décrémenter de $1$ l'adresse du sommet de pile pour y ajouter une donnée: $pp := pp - 1; m[pp] := ac$.

On peut observer dans ce tableau trois modes d'adressage:

- L'adressage *direct*: le code opération est suivu d'une adresse mémoire sur $12$ bits. L'instruction JUMP vue précédemment se situe dans cette catégorie, ainsi par exemple que le chargement en mode direct LODD: $ac := m[x]$, qui charge le mot à l'adresse $x$ de la mémoire centrale dans le registre **AC**.

| Binaire | Symbole | Instruction | Opération |
|---|---|---|---|
| 0000 $xxxx\ xxxx\ xxxx$ | LODD | Chargement mode direct | $ac := m[x]$ |
| 0001 $xxxx\ xxxx\ xxxx$ | STOD | Rangement mode direct | $m[x] := ac$ |
| 0010 $xxxx\ xxxx\ xxxx$ | ADDD | Addition mode direct | $ac := ac + m[x]$ |
| 0011 $xxxx\ xxxx\ xxxx$ | SUBD | Soustraction mode direct | $ac := ac - m[x]$ |
| 0100 $xxxx\ xxxx\ xxxx$ | JPOS | Saut si $> 0$ | $if(ac > 0)\ co := x$ |
| 0101 $xxxx\ xxxx\ xxxx$ | JZER | Saut si $= 0$ | $if(ac = 0)\ co := x$ |
| 0110 $xxxx\ xxxx\ xxxx$ | JUMP | Saut | $co := x$ |
| 0111 $xxxx\ xxxx\ xxxx$ | LOCO | Chargement constante | $ac := x\ (0 \leq x \leq 4\ 095)$ |
| 1000 $xxxx\ xxxx\ xxxx$ | LODL | Chargement mode local | $ac := m[pp + x]$ |
| 1001 $xxxx\ xxxx\ xxxx$ | STOL | Rangement mode local | $m[pp + x] := ac$ |
| 1010 $xxxx\ xxxx\ xxxx$ | ADDL | Addition mode local | $ac := ac + m[pp + x]$ |
| 1011 $xxxx\ xxxx\ xxxx$ | SUBL | Addition mode local | $ac := ac - m[pp + x]$ |
| 1100 $xxxx\ xxxx\ xxxx$ | JNEG | Saut si $< 0$ | $if(ac < 0)\ co := x$ |
| 1101 $xxxx\ xxxx\ xxxx$ | JNZE | Saut si $\neq 0$ | $if(ac \neq 0)\ co := x$ |
| 1110 $xxxx\ xxxx\ xxxx$ | CALL | Appel procédure | $pp := pp - 1; m[pp] := co; co := x$ |
| 1111 0000 0000 0000 | PSHI | Empilement indirect | $pp := pp - 1; m[pp] := m[ac]$ |
| 1111 0010 0000 0000 | POPI | Dépilement indirect | $m[ac] := m[pp]; pp := pp + 1$ |
| 1111 0100 0000 0000 | PUSH | Empilement | $pp := pp - 1; m[pp] := ac$ |
| 1111 0110 0000 0000 | POP | Dépilement indirect | $ac := m[pp]; pp := pp + 1$ |
| 1111 1000 0000 0000 | RETN | Retour | $co := m[pp]; pp := pp + 1$ |
| 1111 1010 0000 0000 | SWAP | Échange **AC** et pile | $tmp := ac; ac := pp; pp := tmp$ |
| 1111 1100 $yyyy\ yyyy$ | INSP | Incrémentation de **PP** | $pp := pp + y\ (0 \leq y \leq 255)$ |
| 1111 1110 $yyyy\ yyyy$ | DESP | Décrémentation de **PP** | $pp := pp - y\ (0 \leq y \leq 255)$ |

Figure 3: Instructions du langage machine

- L'adressage *indirect*: l'adresse en mémoire centrale est d'abord chargée dans **AC**. Par exemple POPI est une instruction qui dépile en mode indirect: $m[ac] := m[pp]; pp := pp + 1$.

- L'adressage *local*: l'adresse est relative au pointeur de pile. Par exemple STOL est une instruction de rangement en mode local, qui copie le contenu de **AC** en mémoire centrale à l'adresse $pp + x$ (où $pp$ est la valeur du registre **PP**), c'est-à-dire dans la pile, à la distance $x$ du sommet: $m[pp + x] := ac$.

Indépendamment du mode d'adressage, les instructions se divisent en plusieurs catégories:

- Les transferts de données entre le processeur (registre **AC**) et la mémoire centrale (éventuellement la pile), parfois assortis d'opérations arithmétiques.

- Les branchements, avec ou sans condition.

- L'appel et le retour de procédure: le paramètre de l'appel est l'adresse de la procédure appelée, qui est placée dans le compteur ordinal **CO**. La valeur précédente est sauvegardée au sommet de la pile d'exécution et sera à nouveau copiée dans le registre **CO** lors du retour.

### 2.3.2 Interprétation d'une instruction du langage machine

Les figures 4, 5, et 6 montrent un exemple de micro-programme. Un micro-programme est donc une suite de micro-instructions, qui sont ici décrites de manière fonctionnelle pour des raisons de lisibilité, mais qu'il faudrait en fait traduire par les $32$ bits correspondants, comme cela a été fait dans la figure 2. Le micro-programme est formé d'une boucle, comprenant trois étapes:

- Recherche de l'instruction en langage machine en mémoire centrale et copie de cette instruction dans le registre **RI**.

- Analyse des bits de poids forts de l'instruction, permettant d'obtenir le code de l'opération.

- Traitement de l'instruction.

La première étape est réalisée par les micro-instructions $0$, $1$, et $2$ du micro-programme.

- $0$: le contenu du compteur ordinal, représentant l'adresse de la prochaine instruction en langage machine, est placé dans le registre d'adresse et le signal RD est diffusé puisqu'une lecture de la mémoire centrale va être réalisée.

- $1$: le compteur ordinal est incrémenté.

- $2$: l'instruction en provenance de la mémoire centrale se trouve maintenant dans le registre de données **RDO** et elle est copiée dans le registre d'instruction **RI**. Numérotons ses bits de $15$ à $0$, le bit numéro $15$ étant celui de poids le plus fort. Remarquons que la copie ne peut être effectuée qu'en faisant transiter la donnée par l'UAL, puis sur le bus $C$. Il est donc possible de tester son bit de poids fort, qui est contenu dans la sortie N de l'UAL. Ainsi, la deuxième étape commence dès cette micro-instruction. Si le bit de poids fort vaut $1$, il s'agit

d'une des 15 dernières instructions du tableau de la figure 3 et le micro-programme contient donc un branchement vers la première micro-instruction qui les analyse (numéro 28). Dans le cas contraire, il s'agit d'une des 8 premières instructions.

Les deux étapes suivantes sont réalisées par tous les groupes de micro-instructions qui précèdent le retour au début de la boucle, représenté par un branchement sur la première micro-instruction: `goto 0;`.

Fin de la deuxième étape: supposons par exemple que les micro-instructions exécutées soient celles de numéros 3, 4, et 5.

- 3: une micro-instruction de ce type a été examinée lors de l'étude de la structure d'une micro-instruction. L'instruction en langage machine, initialement dans **RI**, est recopiée dans **RIT** après avoir été privée de ses deux bits de poids fort. Au passage, le bit numéro 14 est testé et l'on suppose ici qu'il vaut 0.

- 4: la partie de l'instruction présente dans le registre **RIT** est à nouveau privée de son bit de poids fort par le décalage à gauche. Ce bit, qui est le numéro 13 pour l'instruction complète, est testé et l'on suppose encore qu'il vaut 0.

- 5: la variable `ual` est utilisée pour indiquer que la donnée contenue dans **RIT** ne fait que transiter par l'UAL sans être modifiée, pour qu'il soit possible de tester le bit numéro 12. Si ce bit vaut 0, l'instruction en langage machine commence par 0000, c'est donc un LODD, traité par les micro-instructions 6, 7, et 8. Si au contraire ce bit vaut 1, il s'agit d'un STOD, traité par les instructions 9 et 10.

Troisième étape: supposons que l'instruction STOD ait été détectée. Il faut transférer la donnée contenue dans le registre **AC** en mémoire centrale à l'adresse spécifiée par les 12 bits restants de l'instruction.

- 9: ces 12 bits sont copiés dans le registre d'adresse **RAD** par `rad:=ri;`. On peut supposer que, puisque ce registre comporte 12 bits, ce sont les bits de poids faible qui y sont placés. De plus, le contenu de **AC** est copié dans le registre de données **RDO** et le signal `wr;` est diffusé, pour signaler une écriture dans la mémoire centrale.

- 10: le signal `wr;` continue à être diffusé car l'écriture n'est pas instantanée et le retour en début de boucle est exécuté.

**Exercice** Citer et expliquer la suite des micro-instructions exécutées lors d'un appel de procédure et retour.

```
 0: rad:=co; rd;                          programme principal
 1: co:=co+1; rd;                         incrémentation de CO
 2: ri:=rdo; if n goto 28;                analyse et sauvegarde de RDO
 3: rit:=decalg(ri+ri); if n goto 19;     00xx ou 01xx?
 4: rit:=decalg(rit); if n goto 11;       000x ou 001x?
 5: ual:=rit; if n goto 9;                0000 ou 0001?
 6: rad:=ri; rd;                          0000 = LODD
 7: rd;
 8: ac:=rdo; goto 0;
 9: rad:=ri; rdo:=ac; wr;                 0001 = STOD
10: wr; goto 0;
11: ual:=rit; if n goto 15;               0010 ou 0011?
12: rad:=ri; rd;                          0010 = ADDD
13: rd;
14: ac:=rdo+ac; goto 0;
15: rad:=ri; rd;                          0011 = SUBD
16: ac:=ac+1; rd;
17: a:=inv(rdo);
18: ac:=ac+a; goto 0;
19: rit:=decalg(rit); if n goto 25;       010x ou 011x?
20: ual:=rit; if n goto 23;               0100 ou 0101?
21: ual:=ac; if n goto 0;                 0100 = JPOS
22: co:=bet(ri,amasq); goto 0;
23: ual:=ac; if z goto 0;                 0101 = JZER
24: goto 0;
25: ual:=rit; if n goto 27;               0110 ou 0111?
26: co:=bet(ri,amasq); goto 0;            0110 = JUMP
27: ac:=bet(ri,amasq); goto 0;            0111 = LOCO
28: rit:=decalg(ri+ri); if n goto 40;     10xx ou 11xx?
29: rit:=decalg(ri); if n goto 35;        100x ou 101x?
30: ual:=rit; if n goto 33;               1000 ou 1001?
31: a:=ri+pp;                             1000 = LODL
32: rad:=a; rd; goto 7;
33: a:=ri+pp;                             1001 = STOL
34: rad:=a; rdo:=ac; wr; goto 10;
```

Figure 4: Un exemple de micro-programme

```
35: ual:=rit; if n goto 38;          1010 ou 1011?
36: a:=ri+pp;                        1010 = ADDL
37: rad:=a; rd; goto 13;
38: a:=ri+pp;                        1011 = SUBL
39: rad:=a; rd; goto 16;
40: rit:=decalg(rit); if n goto 46;  110x ou 111x?
41: ual:=rit; if n goto 44;          1100 ou 1101?
42: ual:=ac; if n goto 22;           1100 = JNEG
43: goto 0;
44: ual:=ac; if z goto 0;            1101 = JNZE
45: co:=bet(ri,amasq); goto 0;
46: rit:=decalg(rit); if n goto 50;
47: pp:=pp-1;                        1110 = CALL
48: rad:=pp; rdo:=co; wr;
49: co:=bet(ri,amasq); wr; goto 0;
50: rit:=decalg(rit); if n goto 65;  1111, adresse?
51: rit:=decalg(rit); if n goto 59;
52: ual:=rit; if n goto 56;
53: rad:=ac; rd;                     1111 000 = PSHI
54: pp:=pp-1; rd;
55: rad:=pp; wr; goto 10;
56: rad:=pp; pp:=pp+1; rd;           1111 001 = POPI
57: rd;
58: rad:=ac: wr: goto 10;
59: ual:=rit; if n goto 62;
60: pp:=pp-1;                        1111 010 = PUSH
61: rad:=pp; rdo:=ac; wr; goto 10;
62: rad:=pp; pp:=pp+1; rd;           1111 011 = POP
63: rd;
64: ac:=rdo; goto 0;
65: rit:=decalg(rit); if n goto 73;
66: ual:=rit; if n goto 70;
67: rad:=pp; pp:=pp+1; rd;           1111 100 = RETN
68: rd;
69: co:=rdo; goto 0;
```

Figure 5: Un exemple de micro-programme (suite)

```
70:  a:=ac;                          1111 101 = SWAP
71:  ac:=pp;
72:  pp:=a; goto 0;
73:  ual:=rit; if n goto 76;
74:  a:=bet(ri,smasq);               1111 110 = INSP
75:  pp:=pp+a; goto 0;
76:  a:=bet(ri,smasq);               1111 111 = DESP
77:  a:=inv(a);
78:  a:=a+1; goto 75;
```

Figure 6: Un exemple de micro-programme (fin)