Architecture et Système

Stefan Schwoon

Cours L3, 2022/2023, ENS Cachan

Fonctions arithmétiques: demi-additionneur

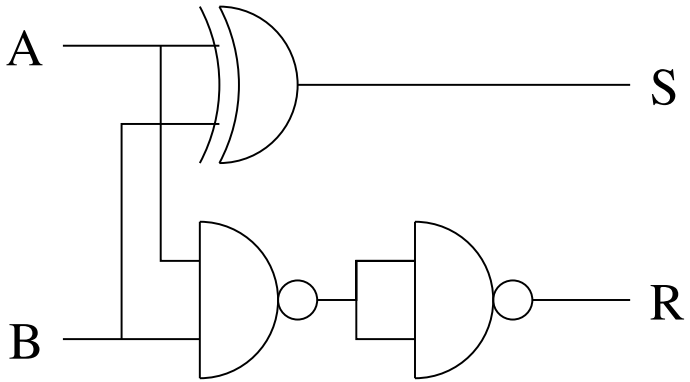
Fonction réalisée par un demi-additionneur:

deux bits en entrée, A et B;

deux bits en sortie, R (la *retenue*) et S (la *somme*);

résultat souhaité : $(R.S)_2 = A + B$ (ou . dénote la concaténation)

Réalisation potentielle :

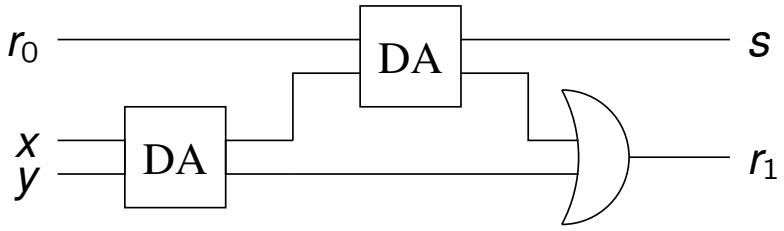


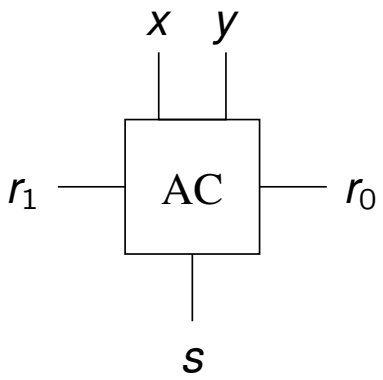
Additionneur complet

Un additionneur complet réalise l'addition de trois bits x, y et r_0, ou r_0 représente la retenue d'une autre addition.

Résultat souhaité : $(r_1.s)_2 = x + y + r_0$

Réalisation à l'aide des DA :



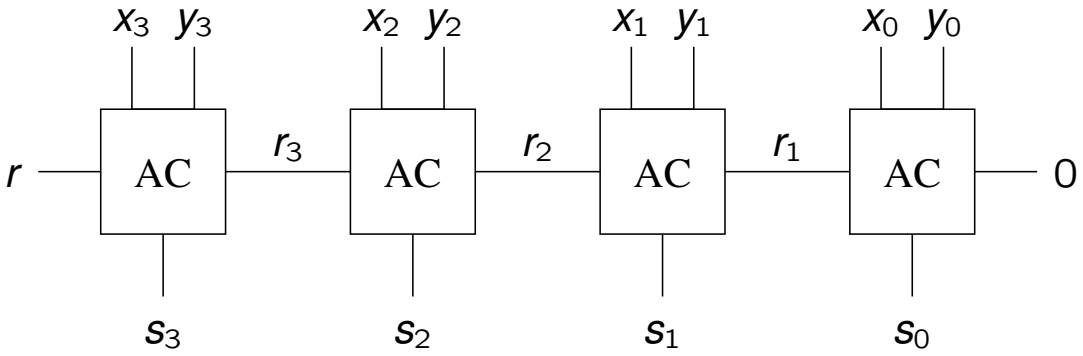


Addition de deux entiers

Supposons que nous avons deux entiers (non-négatifs) à deux bits :
$x = (x_3.x_2.x_1.x_0)_2$ and $y = (y_3.y_2.y_1.y_0)_2$.

On souhaite calculer $s = x + y$ sous la forme $s = (r.s_3.s_2.s_1.s_0)_2$.

Réalisation avec enchaînement de quatre AC :



La géneralisation du principe d'enchaînement à des vecteurs de n bits donnerait un circuit avec taille et profondeur $\mathcal{O}(n)$.

La profondeur est mauvaise car un ordinateur deviendrait deux fois plus lent si on augmente la taille des registres, p.ex. en passant de 32 à 64.

On va étudier une solution avec profondeur *logarithmique*.

Additioneur basé sur propagation/génération

Soit $0 \leq i \leq j < n$. On considère les bits aux positions i à j dans les vecteurs x et y en entrée :

> le bloc $i..j$ génère une retenue si $r_{j+1} = 1$ *indépendemment* des bits sur d'autres positions (on note $g_{i,j} = 1$);
>
> le bloc $i..j$ propage la retenue si $r_i = 1$ implique $r_{j+1} = 1$ (on note $p_{i,j} = 1$).

Du coup, $r_{j+1} = g_{i,j} \vee (r_i \wedge p_{i,j})$.

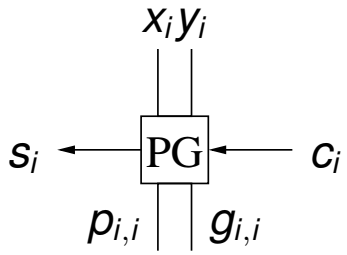
On peut calculer ces deux effets rien qu'avec $(x_i, y_i) \cdots (x_j, y_j)$!

Le cas $i = j$:

$g_{i,i} = 1$ ssi $x_i = y_i = 1$

$p_{i,i} = 1$ ssi $x_i + y_i \geq 1$.

Symbole pour un AC avec calcul des bits p/g en plus :



Le cas $i < j$:

Supposons que le bloc $i..j$ est découpé en deux blocs plus petits, $i..k$ et $k+1..j$, pour $i \leq k < j$.

$$g_{i,j} = g_{k+1,j} \vee (g_{i,k} \wedge p_{k+1,j})$$

$$p_{i,j} = p_{i,k} \wedge p_{k+1,j}$$

Symbole pour calculer la combinaison des bits p/g :

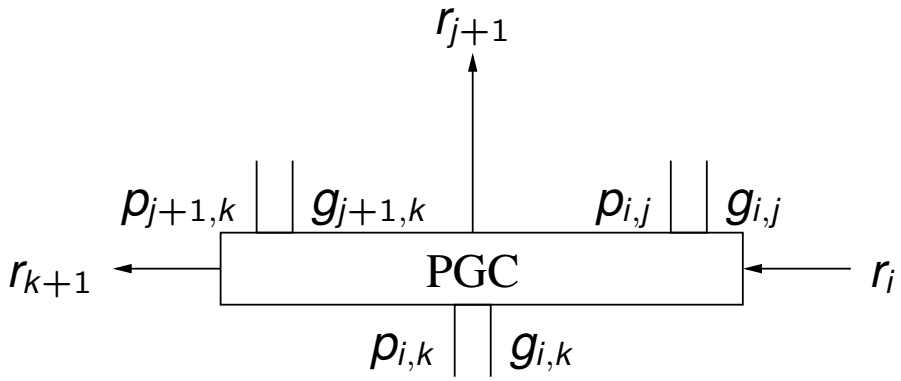
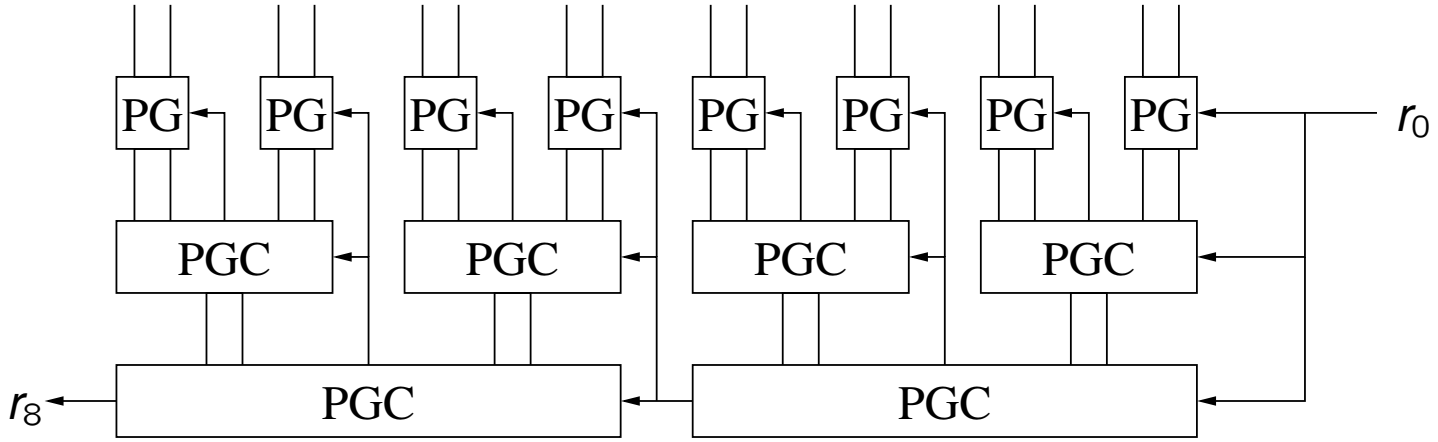


Diagramme simplifié pour $n = 8$, seule la propagation des retenue est détaillée :



Pour comprendre la profondeur du circuit, notons que

- le calcul des p/g est indépendant des valeurs de retenues ;
- la retenue d'un bloc PGC devient stable dès qu'on connaît les p/g et la retenue en entrée.

Du coup, le plus long chemin a une longueur d'environ $2 \log_2 n$.

Circuit pour multiplication

Soient $x = (x_{n-1} \cdots x_0)_2$ et $y = (y_{n-1} \cdots y_0)_2$ deux entiers naturels.

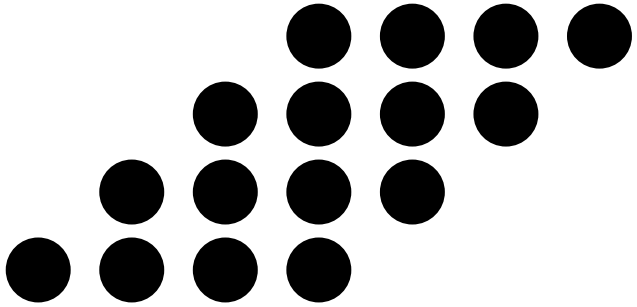
Construisons un circuit efficace pour $z = (z_{2n-1} \cdots z_0)_2$ tel que $z = x \cdot y$.

Technique de multiplication classique :

$$z = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (x_i \cdot y_j \cdot 2^{i+j})$$

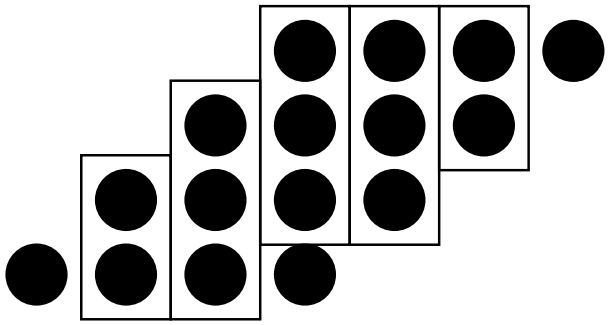
Notons que $x_i \cdot y_j \in \{0, 1\}$, du coup on obtient n^2 bits avec des “poids” différents.

Visualisation (pour $n = 4$), chaque boule représent un produit $x_i y_j$:

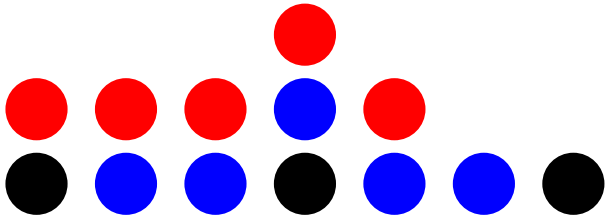


Ayant obtenu les n^2 bits, il faut donc faire l'addition dans les $2n$ colonnes (tout en propagant les retenues). Le nombre maximal de "boules" dans une colonne est de n.

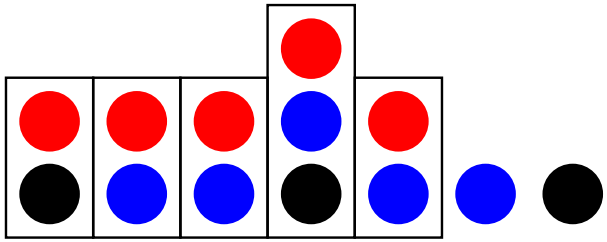
Principe : Successivement réduire les colonnes. On trouve des groupes de deux ou trois “boules” et on fait passer tous les groupes (en parallèle) par des DA ou AC.



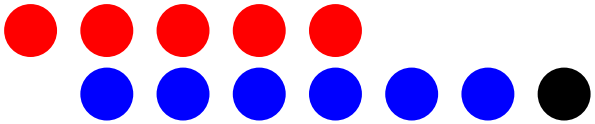
Ensuite on distribue les deux bits résultants (somme et retenue) aux bonnes colonnes :



Itération : identifier des groupes de deux ou trois . . .



. . . ce qui donne :



Ayant réduit le résultat tout les colonnes à deux, on fait l'addition comme avant.

Analyse du multiplicateur

Il faut n^2 portes ET pour obtenir les produits de départ.

Ensuite, on réduit la hauteur des colonnes par un facteur d'environ $2/3$ par itération.

> Une analyse précise montre qu'après $\log_{3/2} n$ itérations on obtient une hauteur d'au plus 4.
>
> Une fois la hauteur réduite à 4, trois itérations supplémentaires sont suffisantes.
>
> Du coup, la réduction se fait en profondeur $\mathcal{O}(\log n)$.

Le circuit pour l'addition finale est de taille $\mathcal{O}(n)$ et de profondeur $\mathcal{O}(\log n)$

Du coup, la multiplication entière se fait en taille $\mathcal{O}(n^2)$ et profondeur $\mathcal{O}(\log n)$.

Fonctions logiques sur les mots

Les ordinateurs permettent typiquement d'appliquer un opérateur logique sur tous les bits de deux mots en parallèle :

$$x \circ y = (x_{n-1} \circ y_{n-1}, \cdots, x_0 \circ y_0)_2$$

Exemples :

cas $\circ = \wedge$: souvent utilisé avec des constants, p.ex. $x \wedge 16$ sert à isoler le 4ème bit de x.

cas $\circ = \vee$: fait l'union des bits qui sont 1, p.ex. $x := x \vee 16$ met le quatrième bit de x à 1.

Multiplexeur

Données :

vecteurs $x_0, \ldots, x_{k-1} \in \{0,1\}^m$, pour $k = 2^n$

indice de sélection s entre 0 et $k-1$, codé par n bits

Sortie: x_s

Applications:

Lire un mot dans la mémoire en spécifiant son adresse.

Choisir entre plusieurs sources de données (p.ex. périphériques).

Obtenir une valeur dans un tableau pré-calculé.

Multiplexeur : Exemple

Soit $m = n = 1$, du coup on possède deux bits x_0, x_1 et un bit de sélection s_0.

Solution: on réalise la fonction $(x_0 \wedge \neg s_0) \vee (x_1 \wedge s_0)$.

Soit $m = 1, n = 2$, du coup on posséde $x_0, \cdots, x_3$ et $s_1 s_0$:

Dans un premier temps, on évalue s_0 pour sélectionner (en parallèle) entre x_0, x_1 et x_2, x_3.

Dans un deuxième temps, on évalue s_1 pour sélectionner parmi les deux bits choisis avant.

Le problème se géneralise pour $n > 2$ avec un circuit de profondeur $\mathcal{O}(n)$.

Pour $m > 1$: On utilise m circuits en parallèle, un par bit.

Décodeur

Données:

valeur s codé par n bits en entrée

2^n lignes de sortie $y_0, \ldots, y_{2^n-1}$

Spécification : $y_s = 1$ et $y_{s'} = 0$ pour tout $s' \neq s$.

Solution (facile) : Pour tout $0 \leq s' < 2^n$, on construit la conjonction qui évalue si $s' = s$.

Applications :

Sélectionner une case mémoire pour y écrire.

Sélectionner un périphérique parmi plusieurs pour échanger des données.

Verrous et bascules

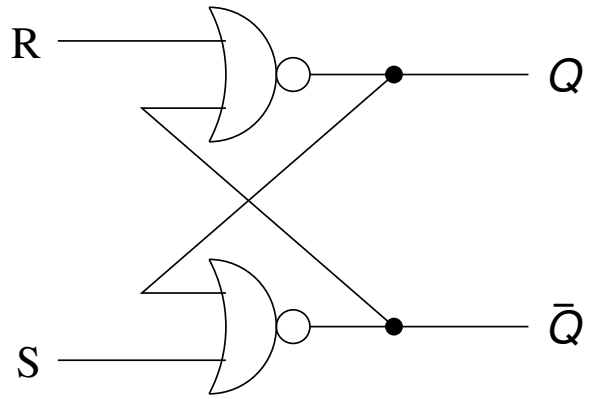
Ce sont des circuits utilisés pour stocker des bits de mémoire. La littérature distingue parfois deux types de circuits (mais sans y être strict) :

- les circuits non-temporisés sont appelés verrous (*latch* en anglais)
- les circuits temporisés sont appelés bascules (*flip-flop* en anglais)

Il y a une grand variété de verrous et bascules, selon des besoins spécifiques. Quelques éléments typiques :

- signaux de contrôle : déterminent si la valeur stockée doit changer
- signal d'horloge (bascules seulement)
- deux sorties Q et $\bar{Q}$, la valeur du bit et son complément

Verrou RS



controls R (reset) and S (set):

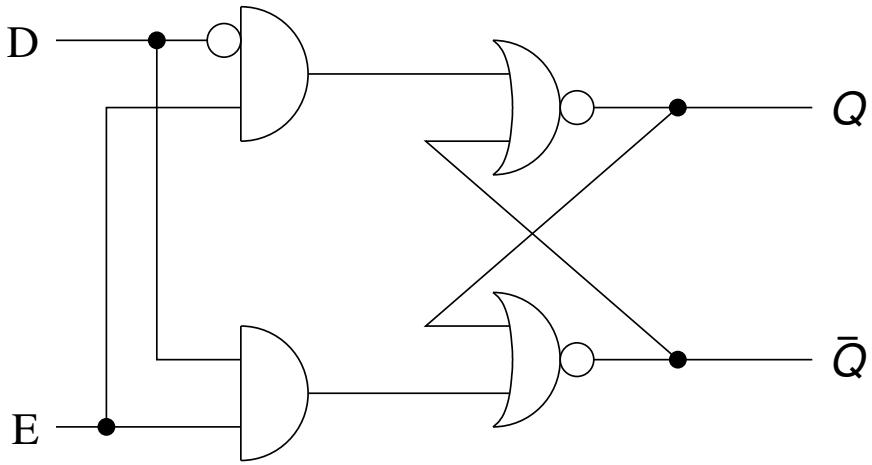
$R = 0$, $S = 1$: après peu de temps, $Q = 1$ et $\bar{Q} = 0$

$R = 1$, $S = 0$: après peu de temps, $Q = 0$ et $\bar{Q} = 1$

$R = S = 0$: les deux sorties restent inchangées

$R = S = 1$: on obtient $Q = \bar{Q} = 0$, devient instable lorsqu'on on revient à $R = S = 0$

Verrou D

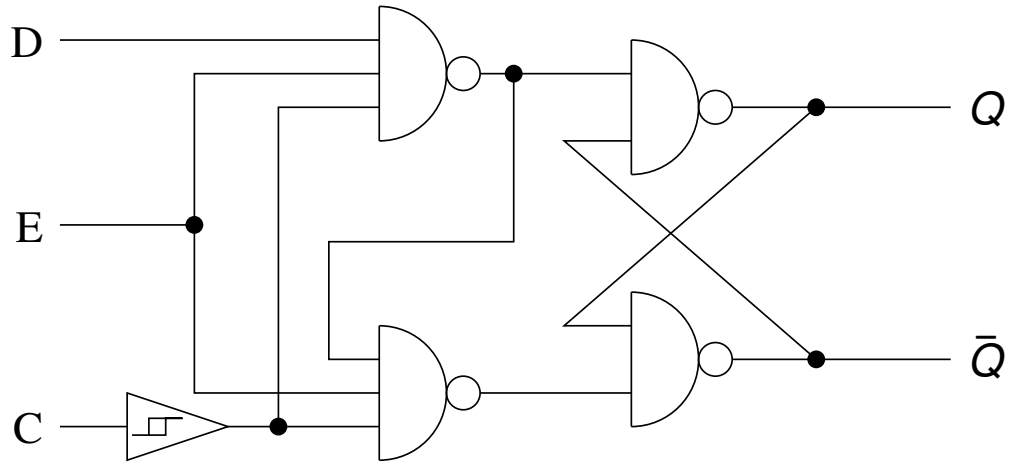


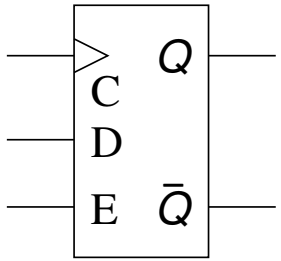
contrôles D (data) et E (enable):

- $E = 0$: stable
- $E = 1$: Q devient D

Bascule D

On reprend le verrou D, on ajoutant un signal oscillant C (*clock* = horloge). Dans ce cas, D n'est pris en compte que si C est dans sa phase haute et que $E = 1$.





Dans les diagrammes, le signal d'horloge est typiquement indiqué par un triangle.

Calcul cyclique d'un ordinateur

Les bascules définissent l'*état actuel* de l'ordinateur :

mémoire, registres, signaux de périphériques, . . .

Le signal fourni par l'horloge *C* est composé d'une longue phase de 0 alternant avec une courte phase de 1.

Pendant la phase 0, le processeur calcule l'état suivant :

Quel registre doit changer ? Quelle case de mémoire ?

Quelles sont les nouvelles valeurs ?

Placer les 'enable' des bascules concernées.

Lorsque l'horloge passe en phase 1, les nouvelles valeurs sont acceptés dans les bascules. L'ordinateur passe ainsi dans un nouvel état, et le cycle recommence.

Signal de horloge

L'horloge *C* est donc partagé parmi toutes les bascules.

La pause entre les "1" de *C* doit être suffisamment longue que tous les calculs d'un cycle puissent réussir.

Les "1" doivent être suffisamment courts pour bien séparer les phases d'un calcul.

Mémoire vive

Les bascules/verrous sont un exemple d'une *mémoire vive*, avec les propritétés suivantes :

mode lecture ou écriture

volatile (contenu perdu sans alimentation électrique)

utilisée pour faire des calculs, stocker des données

en anglais: RAM (random-access memory, on accède à toute adresse en temps constant)

Mémoire morte

Par contre, une *mémoire morte* est dite d'avoir les propriétés suivantes :

mode lecture seulement (pas de moyen pour l'ordinateur ou le programmeur d'en modifier)

durable (même sans alimentation électrique)

utilisée pour assurer le fonctionnement de l'ordinateur (microprogrammation, tables mathématiques pré-calculées)

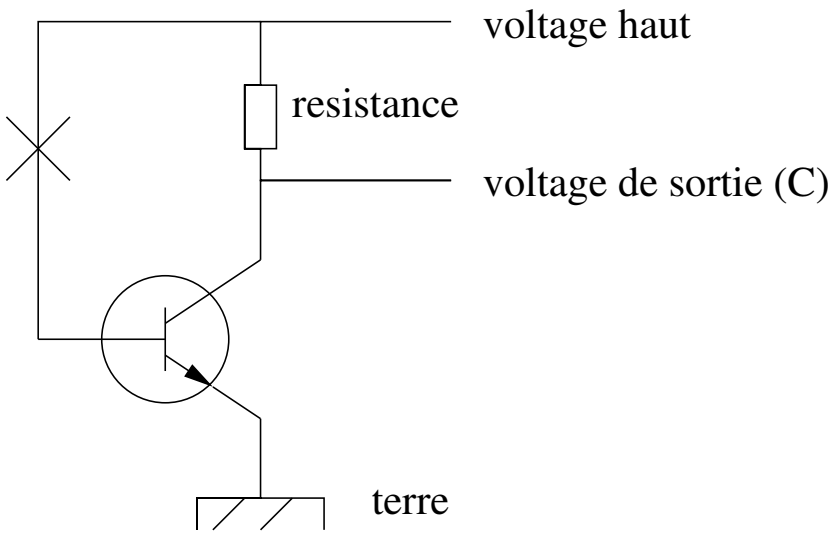
en anglais: ROM (read-only memory)

Exemple d'une mémoire morte

La façon la plus simple de produire un ROM :

prendre une série de transistors, chacun représentera un bit

physiquement interrompre le fil à la position X pour donner 1, sinon 0.



D'autres types de mémoire morte : PROM, EPROM, EEPROM