Chapitre 5
CIRCUITS USUELS

L'objet de ce chapitre est la description des compteurs, des registres, des mémoires volatiles et non volatiles et des circuits arithmétiques. Ces circuits intégrés en MSI, LSI ou VLSI sont de grande utilité pratique à côté des circuits combinatoires décrits au chapitre 3 (décodeurs, encodeurs, multiplexeurs, démultiplexeurs, détecteurs de parité, comparateurs). Ils facilitent la réalisation des systèmes logiques complexes, réduisent considérablement leur dimension et augmentent leur fiabilité. Un ordinateur est essentiellement constitué de ces éléments.

5-1 COMPTEURS

Un compteur reçoit à son entrée u une suite d'impulsions et délivre à sa sortie y = (y_{n-1}, …, y_1, y_0) le nombre des impulsions reçues exprimé en binaire (fig. 5-1). Quand la sortie arrive à sa valeur maximum $2^n - 1$, le dénombrement se répète à partir de 0 et on dit que le compteur est modulo 2^n. Souvent le compteur est suivi d'un décodeur (7 segments) qui transforme le nombre binaire y en un nombre décimal d(y) lisible sur un afficheur :

$$d(y) = y_{n-1}2^{n-1} + \cdots + y_1 2^1 + y_0 2^0 .$$

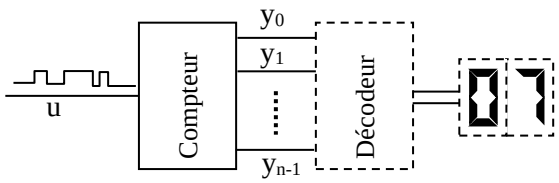


Fig. 5-1 Schéma de principe d'un compteur

On distingue entre compteurs synchrones et compteurs asynchrones selon qu'à la réception d'une nouvelle impulsion les sorties y_{n-1}, …, y_1, y_0 se modifient simultanément ou avec ondulation.

5-1-1 Compteurs asynchrones

Le tableau 5-1 montre la variation des 3 sorties y_2, y_1, y_0 d'un compteur modulo $2^3 = 8$ en fonction du nombre p des impulsions reçues.

p	y_2	y_1	y_0	p	y_2	y_1	y_0
0	0	0	0	5	1	0	1
1	0	0	1	6	1	1	0
2	0	1	0	7	1	1	1
3	0	1	1	8	0	0	0
4	1	0	0	9	0	0	1

Tableau 5-1 Variation des sorties d'un compteur mod. 8

On constate d'après ce tableau que y_0 bascule (change de valeur) après chaque nouvelle impulsion c'est-à-dire à la descente de u de 1 à 0, y_1 bascule à la descente de y_0 et y_2 à la descente de y_1. Les sorties y_0, y_1 et y_2 sont donc les états de 3 bascules T dont l'entrée de la première est connectée à l'entrée u du compteur, l'entrée de la deuxième à y_0 et l'entrée de la troisième à y_1 (nous supposons ici et dans la suite que les bascules sont à front d'activation descendant). La figure 5-2 montre le circuit de ce compteur où les bascules JK sont transformées en bascules T en connectant J et K à la tension d'alimentation V (voir section 4-2-2).

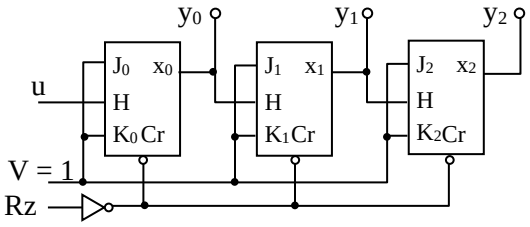


Fig. 5-2 Compteur asynchrone modulo 3

Pour commencer le dénombrement, on applique une impulsion sur Rz pour annuler, durant Rz = 1, les sorties à travers les entrées Cr de remise à 0 des

bascules. À remarquer d'après le tableau 5-1 que le compteur repart de 0 après 8 impulsions d'où le dénombrement est de modulo $2^3 = 8$. Le compteur modulo 2^n a la même structure à part qu'il comporte n bascules au lieu de 3.

Compteur modulo M ≠ 2^n. Supposons qu'on désire construire un compteur modulo M = 10. Comme l'expression binaire de 10 est (1 0 1 0), il suffit d'employer un compteur modulo 2^4 et de connecter à l'entrée Rz de remise à 0 la sortie d'une porte AND d'entrées x_1 et x_3. Cependant cette solution n'est valable que si les temps de propagation des bascules sont assez voisins. En effet, si le temps de propagation entre Cr et la sortie x_1 est sensiblement plus petit que le temps de propagation entre Cr et la sortie x_3, $Rz = x_1x_3$ s'annulera avant x_3 et ce dernier conservera alors sa valeur 1 et ne s'annulera pas. Pour surmonter cette difficulté, on ajoute un loquet qui mémorise la sortie de la porte AND durant le temps où u est nul (fig. 5-3). Si ce temps est supérieur au temps de propagation des bascules (ce qui est presque toujours le cas), les états de ces dernières auront le temps de s'annuler avant que Rz s'annule au front montant du signal u.

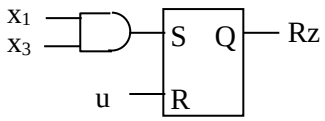


Fig. 5-3 Circuit de remise à zéro du compteur modulo 10

Décompteur.

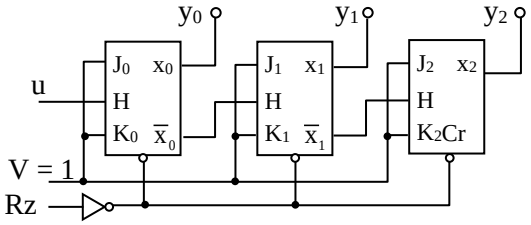


Fig. 5-4 Compteur asynchrone modulo 8 à rebours

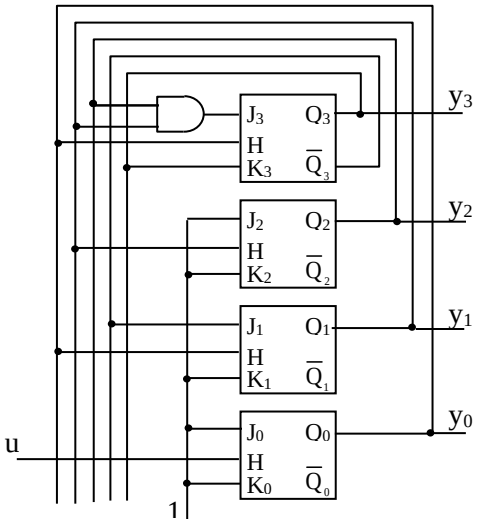
En parcourant le tableau 5-1 de bas en haut (de 9 à 0), on remarque que y_0 change de valeur à chaque ligne tandis que la sortie y_1 bascule à la montée (de 0 à 1) de y_0 et y_2 bascule à la montée de y_1. En posant comme ci-dessus $y_i = x_i$, on obtient un compteur à rebours (décompteur) si l'état x_0 change de valeur à chaque descente de u tandis que l'état x_i des autres bascules change de valeur à chaque montée de x_{i-1} c'est-à-dire à chaque descente de $\bar{x}_{i-1}$. La figure 5-4 montre comment modifier le circuit de la figure 5-2 pour transformer le compteur ordinaire (up counter) en un décompteur (down counter). Ceci se généralise directement à un compteur modulo 2^n.

Désavantage des compteurs asynchrones. À la réception d'une nouvelle impulsion, les sorties d'un compteur asynchrone ne changent pas simultanément mais sous forme d'une onde qui se propage de y_0 à y_{n-1}. Par exemple, la transition du nombre 3 = (0 1 1) au nombre 4 = (1 0 0) se déroule en passant transitoirement par 2 = (0 1 0) et par 0 = (0 0 0) avant d'arriver à (1 0 0). Pour cette raison le compteur asynchrone est aussi appelé *compteur à ondulation* (ripple counter). Le temps de transition d'un nombre stable m au nombre stable suivant est compris, selon m, entre t_b et nt_b où t_b est le temps de propagation à travers une bascule et n le nombre des bits de la sortie. Le compteur asynchrone ne fonctionne donc correctement que si la durée nt_b est inférieure au temps qui sépare deux impulsions de l'entrée c.à.d. que si la fréquence du signal d'entrée est suffisamment faible.

EXERCICE 5-1

Représenter le circuit de remise à zéro d'un compteur modulo 25.

EXERCICE 5-2



Vérifier que le circuit intégré 74LS90 ci-dessus est un compteur asynchrone modulo 10 qui ne comporte pas un circuit de remise à zéro.

5-1-2 Compteurs synchrones

Un compteur est synchrone lorsque les composantes y_{n-1}, …, y_0 de sa sortie changent simultanément à la descente de l'entrée u de 1 à 0. La simultanéité est assurée en connectant l'entrée u aux entrées d'activation H de toutes les bascules et c'est aux entrées J et K de réaliser la succession des valeurs de la sortie données dans le tableau 5-1. On remarque d'après ce tableau qu'à la fin d'une nouvelle impulsion la composante y_i change de valeur si et seulement si les composantes y_{i-1}, …, y_0 sont toutes égales à 1 avant cet instant (dans la ligne précédente). Ceci sera satisfait en prenant

$$\begin{aligned} &J_0 = K_0 = 1, \\ &J_i = K_i = y_{i-1}y_{i-2}\cdots y_0 = y_{i-1}J_{i-1} = x_{i-1}J_{i-1}. \end{aligned} \qquad (5\text{-}1)$$

La figure 5-5 montre le circuit d'un compteur synchrone modulo $2^3 = 8$ où la sortie s prend la valeur 1 à l'instant où toutes les composantes y_i deviennent égales à 1. La structure de ce circuit se généralise directement à un compteur modulo 2^n quelconque. En connectant s aux entrées J et K de la première bascule d'un autre compteur modulo 2^k, on obtient un compteur synchrone modulo 2^{n+k}.

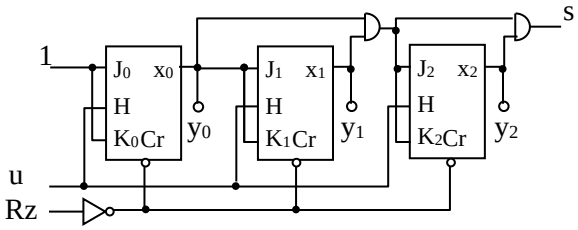


Fig. 5-5 Compteur synchrone modulo 8

Compteur synchrone bidirectionnel. En parcourant le tableau 5-1 de bas en haut on remarque que y_i change de valeur lorsque dans la ligne précédente les composantes y_{i-1}, …, y_0 sont toutes nulles. Pour compter à rebours, il suffit donc que les relations suivantes soient satisfaites :

$$\begin{aligned} &J_0 = K_0 = 1, \\ &J_i = K_i = \overline{y}_{i-1}\overline{y}_{i-2}\cdots\overline{y}_0 = \overline{x}_{i-1}K_{i-1}. \end{aligned} \qquad (5\text{-}2)$$

D'après (5-1) et (5-2), on obtient un compteur bidirectionnel (up-down counter) en connectant aux entrées J et K de chaque bascule (à part la première) un sélecteur dont le circuit est représenté par la figure 5-6. Pour d = 1, le dénombrement sera direct et pour d = 0, il sera à rebours.

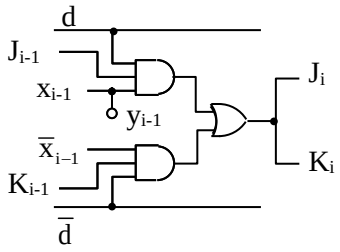


Fig. 5-6 Sélecteur de direction d'un compteur

Compteur synchrone modulo M ≠ 2^n. Ce compteur est généralement conçu en tant que machine de Moore en codant, comme ci-dessus, l'état x d'une étape par le même vecteur que la réponse y correspondante (x = y). Le tableau 5-2 définit en valeur décimale l'état suivant x' en fonction de l'état actuel x d'un compteur modulo 10. Par exemple l'état qui suit x = (0 1 0 1) de valeur décimale 5 est x' = (0 1 1 0) de valeur décimale 6 et l'état qui suit x = (1 0 0 1) de valeur décimale 9 est l'état x' = (0 0 0 0) de valeur décimale 0. Les états qui ont une valeur décimale supérieure à 9 n'apparaissent pas et ils sont par conséquent indifférents.

Tableau 5-2 États suivants du compteur mod 10

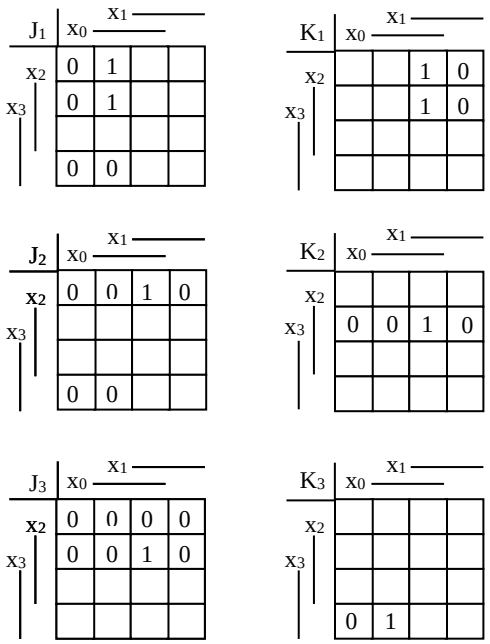
x_3 \ x_2 x_1 x_0				
	1	2	4	3
	5	6	8	7
	9	0		

Les tableaux de Karnaugh qui définissent les fonctions J_i et K_i des 4 bascules du compteur se déduisent du tableau 5-2 tenant compte des valeurs que doivent avoir J_i et K_i pour que l'état de la bascule i passe de x_i à x_i'. Nous rappelons que ces valeurs sont données par le tableau suivant où le trait signifie que la valeur est indifférente :

$x_i \to x_i'$	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$
J K	0 –	1 –	– 1	– 0

J_0 et K_0 sont toujours égaux à 1 puisque x_0 doit basculer à chaque descente de l'entrée u. Il est donc

inutile de dresser les tableaux de Karnaugh de ces fonctions. Les tableaux 5-3 sont ceux des entrées J et K des 3 autres bascules. Par exemple, le passage de 5 = (0 1 0 1) à 6 = (0 1 1 0), s'effectue par (J_1K_1) = (1–), (J_2K_2) = (–0) et (J_3K_3) = (0–).



Tableaux 5-3 Fonctions J et K des bascules 3, 2 et 1 d'un compteur synchrone mod 10.

De ces tableaux on déduit que

$$J_1 = \overline{x}_3 x_0, \quad J_2 = x_1 x_0, \quad J_3 = x_2 x_1 x_0,$$
$$K_1 = x_0, \quad K_2 = x_1 x_0, \quad K_3 = x_0.$$

D'où le circuit de la figure 5-7.

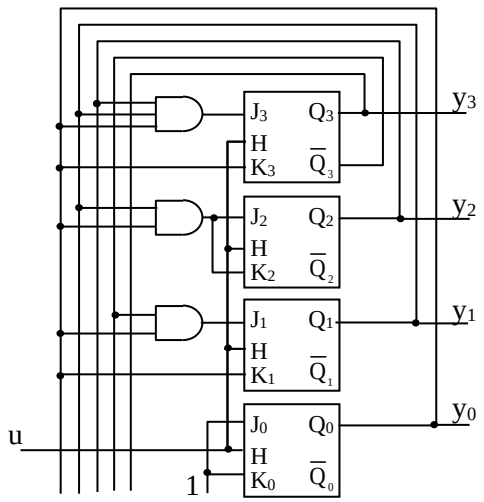


Fig. 5-7 Compteur synchrone modulo 10

EXERCICE 5-3
Construire un compteur synchrone modulo 12.

5-2 REGISTRES

Un registre sert à mémoriser un mot de n bits. Ces bits peuvent être introduits ou lus l'un après l'autre (en série) ou simultanément (en parallèle). La figure 5-8 représente le circuit d'un registre qui peut être chargé en série ou en parallèle et lu en série ou en parallèle. Chaque bascule JK joue le rôle d'une bascule à retard (bascule D) qui, à la descente de H, transmet son entrée J à sa sortie x.

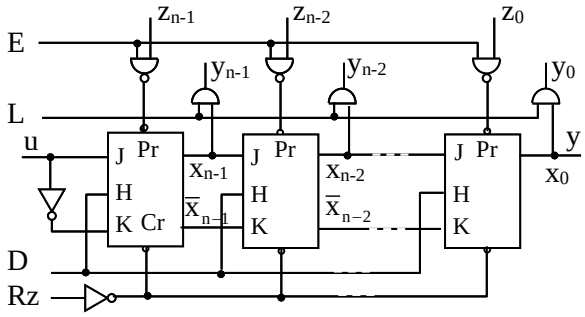


Fig. 5-8 Registre

Ce circuit admet 4 modes opératoires.

Écriture série. Les composantes du mot binaire (a_{n-1}, …, a_0) sont introduites l'une après l'autre de a_0 à a_{n-1} par l'entrée u. On connecte les entrées E et Rz à la masse et on suit la procédure suivante.

- On place sur u la valeur de a_0 puis on applique une impulsion sur l'entrée de décalage D. La sortie de la première bascule devient $x_{n-1} = a_0$.
- En répétant l'opération précédente une deuxième fois avec $u = a_1$, les sorties des deux premières bascules deviennent $x_{n-1} = a_1$, $x_{n-2} = a_0$.
- Après n opérations de la sorte, les sorties des bascules deviennent $x_{n-1} = a_{n-1}$, …, $x_0 = a_0$. Ceci complète l'écriture du mot dans le registre.

Lecture série. Les composantes du mot binaire (a_{n-1}, …, a_0) stocké dans le registre sont lues l'une après l'autre de a_0 à a_{n-1} sur la sortie y. Les entrées E et Rz étant connectées à la masse, chaque nouvelle impulsion sur D déplace le mot d'une position vers la

droite. La composante à l'extrême droite disparait et on lit sur y la valeur de la composante suivante à partir de la droite.

Écriture parallèle. Les composantes du mot binaire (a_{n-1}, a_{n-2} …, a_0) sont introduites à travers les entrées z_{n-1}, z_{n-2}, …, z_0. On connecte les entrées D et Rz à la masse et on suit la procédure suivante.

- On place sur les z_i les valeurs des a_i.
- Une impulsion sur l'entrée d'écriture E transmet simultanément toutes ces valeurs aux sorties des bascules : $x_0 = a_0$, …, $x_{n-1} = a_{n-1}$. Ceci complète l'écriture du mot dans le registre.

Lecture parallèle. Quand l'entrée de lecture L est égale à 1, les composantes du mot binaire (a_{n-1}, a_{n-2}, …, a_0) stocké dans le registre sont lues sur les sorties y_{n-1}, y_{n-2} …, y_0.

Ce qui précède permet de distinguer entre 4 modes de fonctionnement : écriture série/lecture série (serial-in, serial-out ou SISO), écriture série/lecture parallèle (SIPO), écriture parallèle/lecture série (PISO), écriture parallèle/lecture parallèle (PIPO).

Registre universel. En plus des 4 modes ci-dessus, le registre universel possède 2 directions de décalage l'une vers la droite et l'autre vers la gauche. La figure 5-9 montre le circuit de commande et celui de l'étage i d'un registre universel. Pour un registre à n bit, x_0 désigne l'entrée de lecture série vers la gauche et x_{n+1} l'entrée de lecture série vers la droite.

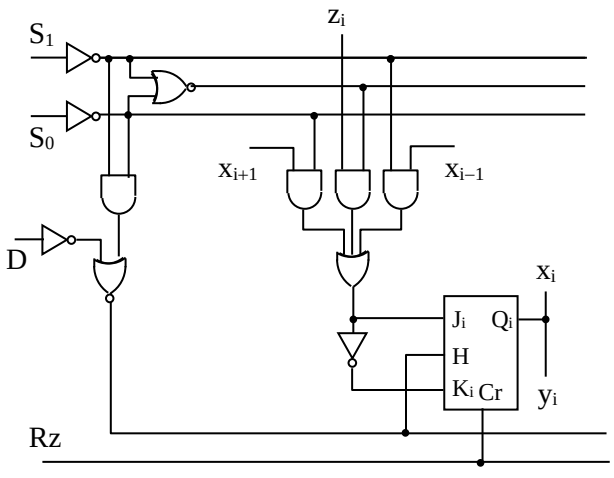


Fig. 5-9 Étage d'un registre universel.

La figure 5-10 représente la capsule d'un registre universel à 4 bits fabriqué par « Signetics Corporation : SC ». Son circuit interne est celui de la figure 5-9 où $x_i = Q_i$.

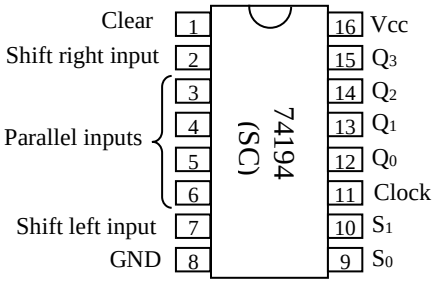


Fig. 5-10 Capsule d'un registre universel

- Pour $S_0 = 0$ et $S_1 = 1$, chaque coup d'horloge (sur l'entrée de décalage D) déplace les sorties d'une position vers Q_0 et transmet à Q_3 la donnée se trouvant sur la broche 2.
- Pour $S_0 = 1$ et $S_1 = 0$, chaque coup d'horloge déplace les sorties d'une position vers Q_3 et transmet à Q_0 la donnée placée sur la broche 7.
- Pour $S_0 = 1$ et $S_1 = 1$, chaque coup d'horloge transmet simultanément les bits placés sur les broches 3, 4, 5 et 6 (entrées z_i) respectivement aux sorties Q_3, Q_2, Q_1 et Q_0.
- Pour $S_0 = 0$ et $S_1 = 0$, le registre n'est affecté par aucune entrée (car H = 0) et retient le dernier mot qui lui est introduit.

EXERCICE 5-4

On connecte la sortie x_0 d'un registre à 5 bits à son entrée u et on applique sur l'entrée de décalage D un signal d'horloge de période T. Représenter durant 10 périodes les signaux aux sorties y_4, …, y_0 quand l'état initial est

a) $x_4 = x_3 = x_2 = x_1 = 0$ et $x_0 = 1$.
b) $(x_4, x_3, x_2, x_1, x_0) = (0, 1, 1, 0, 1)$.

EXERCICE 5-5

On connecte la sortie $\overline{x}_n$ d'un registre à n bits à son entrée u et on applique sur l'entrée de décalage D un signal d'horloge de période T.

a) Représenter les signaux aux sorties y_4, …, y_0 en supposant qu'elles sont initialement nulles. Déduire que ce circuit peut être employé en diviseur de fréquence.
b) Montrer que ce circuit peut être utilisé comme un compteur modulo n en lui ajoutant un encodeur.

5-3 MÉMOIRE ROM

Une mémoire à lecture seulement (read only memory, ROM) a p entrées constituant l'*adresse* (a_{p-1}, …, a_0) et q sorties constituant un mot binaire (d_1, …, d_q) appelé *donnée* (fig. 5-11). À chaque adresse correspond une donnée qui se conserve après la coupure du courant d'alimentation et on dit que cette mémoire est *non volatile*. Une mémoire ROM n'est autre qu'un circuit combinatoire réalisant une fonction de $\mathcal{B}^p$ dans $\mathcal{B}^q$ essentiellement composé d'un décodeur p/r, r = 2^p, et d'un encodeur à r entrées et q sorties.

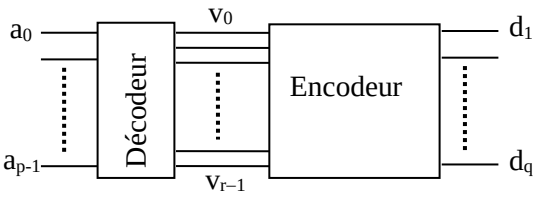


Fig. 5-11 Les deux parties d'une mémoire ROM

Le décodeur active la sortie v_i dont l'indice i est égale à la valeur décimale de l'adresse (a_{p-1}, …, a_0) (voir section 3-3-3). L'encodeur associe à l'entrée activée v_i une donnée $(d_1, …, d_q)_i$ définie par la fonction que la mémoire réalise. La figure 5-12 montre le circuit interne d'une petite mémoire ROM à diodes dont le rôle est de convertir le code binaire de $\mathcal{B}^2$ en code de Gray. Le circuit de ce convertisseur est déduit de son tableau de vérité suivant montrant la correspondance entre les deux codes :

a_1	a_0	d_1	d_2
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

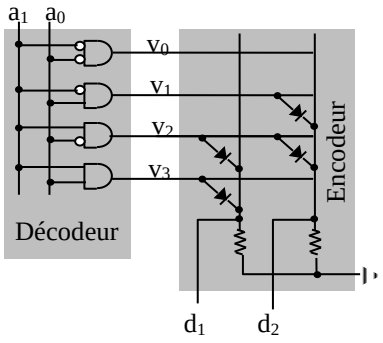


Fig. 5-12 Convertisseur ROM du code binaire au code de Gray.

ROM à transistors. Au lieu des diodes, on peut aussi employer un transistor à émetteur multiple pour chaque ligne v_i. Ceci réduit le courant traversant les portes AND du décodeur. La figure 5-13 montre les connexions du transistor de la ligne v_i à laquelle correspond la donnée (1 0 1 1).

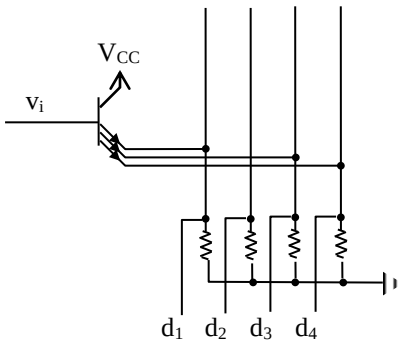


Fig. 5-13 Ligne v_i d'un encodeur à transistors.

La capacité d'une ROM se définit par le produit du nombre de mots par le nombre de bits par mot, soit 2^pxq bits. Pour réduire leur échauffement, les ROM de grande capacité (1 kilobits ou plus) sont généralement construites en transistors MOS. La figure 5-14 montre le circuit de l'encodeur d'une ROM en transistors MOS qui réalise le tableau de vérité suivant :

a_1	a_0	d_1	d_2	d_3
0	0	0	1	0
0	1	1	0	0
1	0	1	1	0
1	1	1	0	1

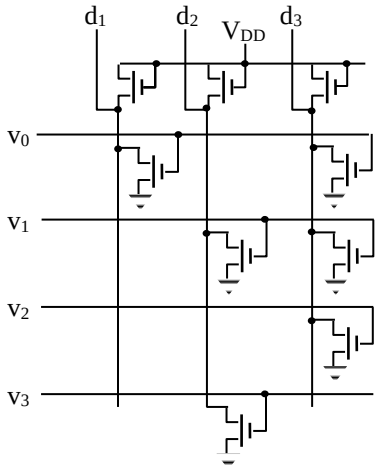


Fig. 5-14 Encodeur MOS

ROM à adresse bidimensionnelle. Quand le nombre p de bits de l'adresse est grand, un décodeur unidimensionnel tel que celui représenté à la figure 5-12 aura un grand nombre de portes AND. Par exemple, ce nombre sera égal à $2^9 = 512$ portes AND pour une adresse à p = 9 bits. Ceci provient du fait qu'une ligne v_i du décodeur s'adresse à une seule donnée. Une économe considérable de portes résulte de l'architecture représentée à la figure 5-15 où l'encodeur comporte $n = 2^r$ lignes, chacune reliée à 2^s données de q bits, s = p – r. La première partie (a_{r-1}, …., a_0) de l'adresse détermine la ligne v_i de l'encodeur et la partie (a_{p-1}, …, a_r) sélectionne de chaque colonne de l'encodeur le bit ayant cette adresse (sélecteur = multiplexeur). Les q bits d'une donnée sont situés à la même position dans les q colonnes de l'encodeur.

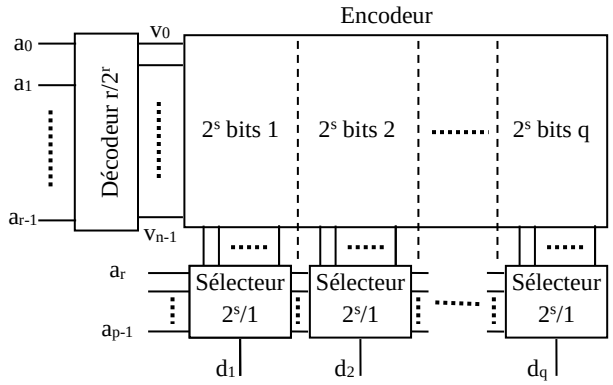


Fig. 5-15 ROM à adresse bidimensionnelle

Chaque multiplexeur comporte 2^s portes AND et une porte OR (voir fig. 3-27), d'où le décodeur et les sélecteurs comportent ensemble $2^r + q(2^s + 1)$ portes. Pour p = 9, r = 6, s = 3 et q = 8, ce nombre est égal à 136, bien plus petit que les 512 portes que nécessite l'adressage unidimensionnel.

ROM programmable. Une ROM dédiée à effectuer une fonction donnée se construit par un fabricant de circuits intégrés. Souvent la ROM de la fonction désirée n'est pas disponible sur le marché et doit être commandée spécialement, ce qui augmente son prix ainsi que son temps de livraison. Pour éviter cet inconvénient, on emploie une ROM qui est programmable par l'utilisateur lui-même et il en existe 3 types : PROM, EPROM et EEPROM.

a) PROM (programmable ROM). C'est une ROM où chaque ligne v_i sortant du décodeur est connectée à chaque ligne d_k de la donnée à travers une sorte de fusible (en série avec une diode ou sur l'émetteur d'un transistor). Pour réaliser sa fonction, l'utilisateur « brule » certains de ces fusibles en y envoyant un courant élevé à l'aide d'un instrument spécial appelé programmeur PROM. Une PROM ne peut être programmée qu'une seule fois puisque les fusibles une fois brulés ne seront plus réparables.

b) EPROM (erasable PROM). Elle a la même structure que le circuit de la figure 5-14 à part que toutes les connexions entre les lignes v_i et d_k sont présentes et que les transistors MOS sont à deux grilles G et F en poly silicone (fig. 5-16). La grille interne F, dite flottante, est complètement entourée par l'isolant SiO_2.

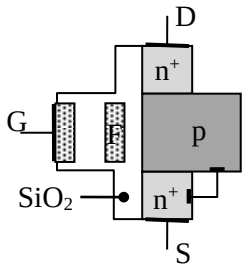


Fig. 5-16 Transistor MOS à 2 grilles

Une tension élevée entre la grille G et le drain D ($V_{GD} = V_G - V_D \approx 25V$) provoque une avalanche d'électrons qui traversent d'abord la jonction np entre le drain et le substrat puis l'isolant SiO_2 et s'enferment dans la grille F. Par leur attraction des charges positives du substrat et leur répulsion des charges négatives, ces électrons dans F empêchent la formation d'un canal n entre le drain D et la source S. Le transistor reste donc bloqué et n'annule pas D quand une tension normale (5V) est appliquée entre G et S. En neutralisant par ce moyen certains transistors l'utilisateur réalise la fonction désirée. Contrairement à la PROM, ce programme peut être effacé en exposant l'EPROM aux rayons ultraviolets. En effet, sous l'effet de ces rayons, l'isolant SiO_2 devient légèrement conducteur ce qui permet aux électrons de quitter la grille F et de retourner à leur position initiale dans D.

c) EEPROM (electrically erasable PROM). Le temps nécessaire pour effacer par rayons ultraviolets un programme de l'EPROM est assez long. La mémoire EEPROM dont le programme s'efface électriquement

n'a pas cet inconvénient. Elle a la même constitution que l'EPROM à part que la couche séparant la grille flottante F du substrat est très mince de l'ordre de 10 nanomètres ce qui permet de charger F sous la tension V_{GD} = 10V et de la décharger sous la tension inverse V_{GD} = –10V.

EXERCICE 5-6
Les mémoires ROM (programmable ou non) sont généralement munies d'une entrée d'activation CS telle que pour CS = 0 la mémoire fonctionne normalement et pour CS = 1 toutes ses sorties deviennent flottantes. Utilisant 4 mémoires 256x4 et un décodeur construire une mémoire 1024x4.

Étalage logique programmable (PLA). C'est une variante de la mémoire ROM qui sert à réaliser par programmation plusieurs fonctions logiques à plusieurs variables. La figure 5-17 est une représentation schématique d'un petit PLA (programmable logic array) pouvant réaliser 2 fonctions y_1 et y_2 de 3 variables u_1, u_2 et u_3, chaque fonction étant la somme d'un maximum de 4 produits.

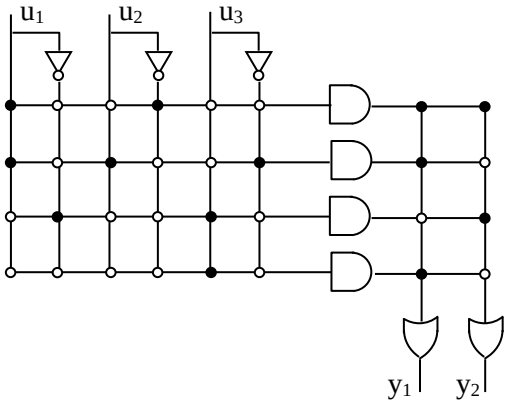


Fig. 5-17 PLA 3x4x2 (3 variables, 4 produits, 2 fonctions)

Pour simplifier le dessin, les 6 entrées d'une porte AND sont représentées par une seule ligne horizontale et les 4 entrées d'une porte OR sont représentées par une seule ligne verticale. Les ronds à l'intersection des lignes représentent des connexions par diodes ou par transistors. Un rond blanc signifie que la connexion est supprimée en brulant un fusible ou en désactivant un MOS à deux grilles. Le programme schématisé par la figure 5-17 réalise les fonctions suivantes :

$$y_1 = u_1\bar{u}_2 + u_1u_2\bar{u} + u_3,$$
$$y_2 = u_1\bar{u}_2 + \bar{u}_1u_3.$$

Si les connexions entre les portes AND et les portes OR sont fixes (non programmables) on aura un circuit appelé PAL (programmable array logic). Les PAL et les PLA rencontrés sur le marché sont généralement bien plus larges que le circuit de la figure 5-17. Par exemple, le 82S100 de Signetics comporte 16 entrées et 8 sorties chacune étant la somme d'un maximum de 48 produits (48 lignes horizontales).

5-4 MÉMOIE RAM

Une RAM (random access memory) est constituée d'un grand nombre de cellules chacune étant une mémoire à un seul bit pouvant être accédée à tout moment pour la lire ou la modifier. Contrairement à la ROM, une mémoire RAM est *volatile* dans le sens que ses bits s'effacent dès la coupure de son alimentation. Dans la plupart des RAM de faible capacité ($\leq$ 16 kilobits = 16 x 2^{10} bits) les bits sont mémorisés par des bascules (D flip-flop) tandis que les bits des grandes RAM sont mémorisés en chargeant ou déchargeant des condensateurs. Les RAM à bascules sont appelées *statiques* (SRAM) et celles à capacités sont appelées *dynamiques* (DRAM).

Adressage et accès. Les cellules d'une RAM à 2^{2p} bits sont disposées en r = 2^p lignes et r = 2^p colonnes adressées à travers deux décodeurs chacun à p entrées (fig. 5-18).

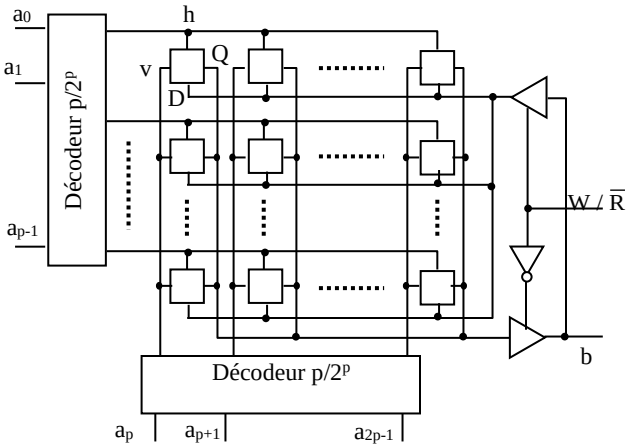


Fig. 5-18 Architecture d'une mémoire RAM

Chaque cellule a en général 3 entrées h, v et D et une sortie Q. La cellule est sélectionnée quand ses entrées h et v sont toutes les deux égales à 1. L'écriture d'un bit dans la cellule sélectionnée est envoyé de la ligne de donnée b et reçu par l'entrée D. Sur cette même ligne b se lit à partir de Q le bit mémorisé dans la cellule. Quand l'entrée $W/\overline{R}$ du RAM est égale à 1, il se met en mode d'écriture et quand elle est égale à 0, il se met en mode de lecture. Grâce aux deux portes 3 états commandées par $W/\overline{R}$, les deux modes ne peuvent pas s'effectuet en même temps.

Cellule d'une RAM statique. C'est une bascule D dont la sortie Q ne peut se modifier et ne peut être lue que lorsque la cellule est sélectionnée par les décodeurs des lignes et des colonnes (fig. 5-19).

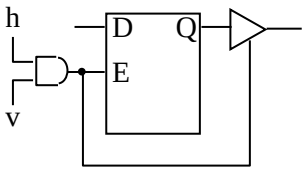


Fig. 5-19 Cellule d'un RAM statique

Pour augmenter la concentration des cellules par unité de surface sans dégager trop de chaleur, les bascules d'une SRAM sont le plus souvent construites en MOS dont les circuits ont de petites dimensions et dissipent peu d'énergie.

RAM dynamique. La concentration des cellules augmente beaucoup plus si le bit est mémorisé par un condensateur de très petite taille au lieu d'une bascule qui comporte plusieurs transistors. Le bit d'une cellule sera 1 si sa capacité est chargée et sera 0 si sa capacité est vide. Cependant, vu sa petite taille, cette capacité se décharge d'elle-même (sans la décharger intentionnellement) à cause des courants de fuite à travers elle et à travers les autres éléments du circuit. Il est donc nécessaire de la recharger périodiquement (la rafraichir) et c'est pour cette raison qu'une RAM à capacités est appelée dynamique (DRAM).

La figure 1-20 montre les deux premières colonnes d'une RAM dynamique connectées à leurs amplificateurs d'écriture, de lecture et de rafraichissement. Bien que ces amplificateurs (aussi en MOS) compliquent la circuiterie, une RAM dynamique reste globalement beaucoup plus dense en cellules qu'une RAM statique.

Pour écrire un bit dans une cellule d'adresse (h, v), on effectue les opérations suivantes :

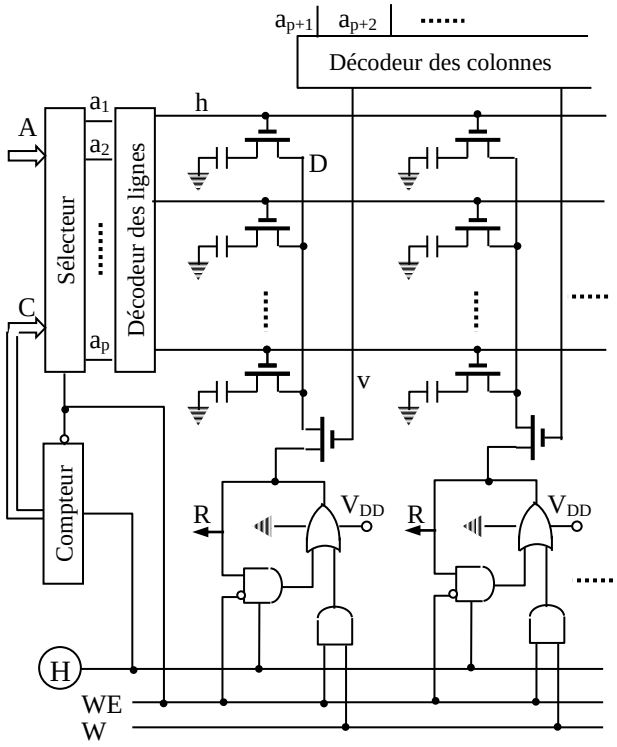


Fig. 5-20 Circuit d'un RAM dynamique

- On fait WE = 1 ce qui arrête le compteur, sélectionne l'adresse de la ligne h indiquée par les entrées A du sélecteur et annule la sortie de la porte AND 3 états connectée à l'horloge H par son entrée d'activation.
- On indique l'adresse v sur l'entrée du décodeur des colonnes.
- On place sur W le bit à introduire dans la cellule. Si W = 1, la capacité de la cellule se charge à partir de V_{DD} et à travers les transistors de la ligne et de la colonne de cette cellule. Si W = 0, le même chemin se connecte à la masse pour décharger la capacité.

Entre deux écritures, les capacités des cellules sont rafraichies périodiquement de la manière suivante :

- En donnant à WE la valeur 0, toutes les sorties du décodeur des colonnes deviennent 1 (circuit non représenté), le compteur se remet en marche et sa

sortie C est choisie par le sélecteur. À chaque coût d'horloge toutes les cellules d'une nouvelle ligne sont sélectionnées et la valeur de chaque cellule est lue (sur R) et envoyée à la porte OR quand H redevient 1. Si la valeur de R est 1, la cellule se recharge jusqu'à V_{DD} et si elle est 0, la cellule se décharge complètement. Typiquement, la durée d'une impulsion d'horloge est inférieure à 1 μs et toutes les lignes sont balayées en moins que 2 ms.

Une RAM peut être intérieurement organisée ou plusieurs RAM peuvent être connectées ensemble pour lire et écrire des mots au lieu des bits. Par exemple, le TMS4464 de Texas Instruments est subdivisé en 4 régions chacune de 64 kilobits pour constituer une mémoire de 2^{16} mots de 4 bits. On dit que la capacité de cette mémoire est 64K x 4 bits. D'autre part, Texas Instruments fournit des modules contenant 4, 8 ou 9 RAM de 256 kilobits chacune (le TMS4256) connectées de sorte à pouvoir mémoriser 2^{18} mots de 4, 8 ou 9 bits.

EXERCICE 5-7
Comment connecter 4 RAM chacune de 256 kb pour constituer une RAM de 256K x 4 bits ?

5-5 ADDITIONNEURS

L'addition de nombres exprimés en chiffres binaires 0 ou 1 est l'opération de base dans les machines de calcul numérique. La soustraction n'est que l'addition d'un nombre positif avec un nombre négatif, la multiplication et la division sont des suites d'additions et de soustractions. Avant de décrire les circuits qui exécutent ces opérations, il convient d'abord de rappeler comment les nombres peuvent être représentés.

Systèmes de numération. Un système de numération est constitué d'un ensemble de symboles appelés *chiffres* ou *digits* et de règles permettant d'évaluer un nombre selon la disposition de ses chiffres. Par exemple dans le système de numération romain, les symboles I, V, X, L, C, D et M représentent les nombres 1, 5, 10, 50, 100, 500 et 1000. Le nombre romain LMXXVII vaut 977 car, d'après les règles de ce système (un chiffre s'ajoute au suivant s'il lui est plus grand ou égal, se retranche s'il lui est plus petit), c'est la somme de LM = 950, XX = 20 et VII = 7. Un tel système convient difficilement aux calculs arithmétiques et on préfère actuellement employer des numérations dites à *positions pondérées*. La pondération d'un chiffre à la position k est b^k, b étant un entier appelé *base*.

Un système de numération de base b est constitué d'un ensemble de b chiffres, S = {0, 1, …, b – 1}, représentant les nombres entiers de 0 à b – 1. Tout nombre positif x, entier ou fractionnaire, s'écrit sous la forme $(x_{n-1}x_{n-2}\ldots x_0 \bullet x_{-1}x_{-2}\ldots x_{-k})_b$ avec $x_i \in S$ et il est évalué par la formule :

$$x = x_{n-1}b^{n-1} + x_{n-2}b^{n-2} + \cdots + x_0 b^0 + x_{-1}b^{-1} + x_{-2}b^{-2} + \cdots + x_{-k}b^{-k} \qquad (5\text{-}3)$$

Les chiffres à gauche du point forment la partie entière de x et ceux situés à sa droite forment sa partie fractionnaire. Les chiffres non représentés à gauche de x_{n-1} ou à droite de x_{-k} sont considérés nuls.

La numération décimale (b = 10) est la plus connue et on sait, par exemple, que

$(263\bullet59)_{10} = 2.10^2 + 6.10^1 + 3.10^0 + 5.10^{-1} + 9.10^{-2}$.

Aux circuits logiques à deux niveaux (tension haute ou tension basse) s'adapte naturellement la numération *binaire* (b = 2) dont les chiffres (0 et 1) sont appelés *bits*. Par exemple,

$(1011\bullet011)_2 = 1.2^3 + 1.2^1 + 1.2^0 + 1.2^{-2} + 1.2^{-3}$,
$= (11\bullet375)_{10}$.

Inversement, $(11.375)_{10} = (11)_{10} + (0.375)_{10}$. Or

$$\left.\begin{array}{l} 11 = 2x5 + 1 \\ 5 = 2x2 + 1 \\ 2 = 2x1 + 0 \\ 1 = 2x0 + 1 \end{array}\right| \qquad \left.\begin{array}{l} 0.375x2 = 0.75 + 0 \\ 0.75x2 = 0.5 + 1 \\ 0.5x2 = 0 + 1 \end{array}\right| .$$

La partie entière de l'écriture binaire est la suite 1011(à partir de la droite) des restes obtenus par des divisions successives par 2 et sa partie fractionnaire est la suite 011 (à partir de la gauche) des parties entières obtenues par des multiplications successives

par 2. Cette règle se généralise à toute base b (divisions ou multiplications par b).

EXERCICE 5-8
a) Est-ce que $(0.6)_{10}$ peut avoir une représentation binaire exacte ?
b) Convertir $(714.1875)_{10}$ en octal (b = 8) et déduire ses écritures binaire et hexadécimale (b = 16).

Les numérations de base $b = 2^r$, $r \geq 2$, servent à simplifier l'écriture de la numération binaire. Par exemple, écrivons en octal ($b = 2^3$) le nombre binaire $x = (11010111001.10101)_2$. On a :

$$x = 2^{10} + 2^9 + 2^7 + 2^5 + 2^4 + 2^3 + 2^0 + 2^{-1} + 2^{-3} + 2^{-5}$$
$$= (2+1)(2^3)^3 + (2)(2^3)^2 + (2^2 + 2 + 1)(2^3) + (1)(2^3)^0 + (2^2 + 1)(2^3)^{-1} + (2)(2^3)^{-2}$$

On déduit que son écriture en numération *octale* est $(3271.52)_8$. Cette conversion du binaire à l'octal (ou l'inverse) s'effectue rapidement en remplaçant chaque 3 chiffres binaires à partir du point par leur valeur octale (ou l'inverse) comme le montre la figure suivante.

3 2 7 1 5 2
011010111001•101010 .

La conversion entre le binaire et l'*hexadécimal* ($b = 2^4 = 16$) s'effectue de la même manière à part qu'un chiffre hexadécimal est l'équivalent de 4 chiffres binaires au lieu de 3 et que les nombres de 10 à 15 se représentent par les symboles A, B, C, D, E et F :

6 B 9 A 8
011010111001•10101000 .

D'une façon similaire, la numération *décimale codée binaire* (binary coded decimal, BCD) consiste à remplacer chaque chiffre décimal par son code binaire en 4 digits (pour pouvoir coder en binaire les chiffres 8 et 9). Par exemple,

$$(69.25)_{10} = (0110\,1001.0010\,0101)_{BCD}\,.$$

Additionneur de 2 entiers positifs. Le tableau suivant donne la somme s et la retenue r de deux bits a et b.

a	b	r	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

On déduit directement de ce tableau que

$$s = a \oplus b \quad \text{et} \quad r = ab\,. \qquad (5\text{-}4)$$

Le circuit qui réalise ces fonctions est appelé *semi-addeur* et il est représenté avec son symbole à la figure 5-21.

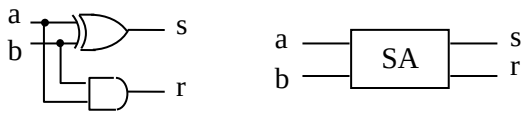


Fig. 5-21 Semi-addeur

Un *addeur complet* ajoute aux deux bits à additionner la retenue r_p provenant de la position précédente. La somme s sera 1 si le nombre des 1 dans le vecteur d'entrée (r_p, a, b) est impair et la retenue r sera 1 si ce nombre est 2 ou 3. En d'autres termes,

$$s = r_p \oplus a \oplus b,$$
$$r = r_p (a \oplus b) + ab. \qquad (5\text{-}5)$$

Ces relations se réalisent par deux semi-addeurs et une porte OR comme le montre la figure 5-22.

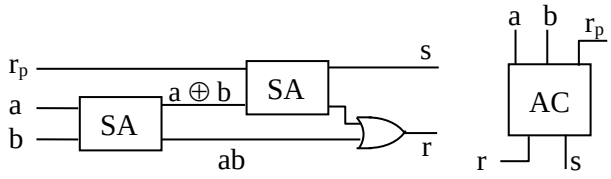


Fig. 5-22 Addeur complet

L'additionneur de deux nombres binaires positifs à n bits s'obtient alors en connectant en série n addeurs complets comme le montre la figure 5-23.

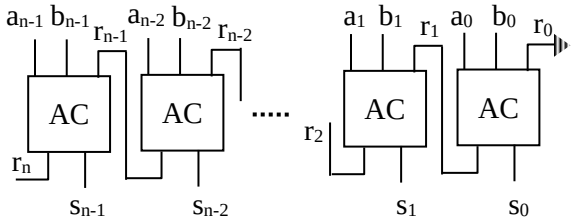


Fig. 5-23 Additionneur de deux nombres positifs

Le circuit de la figure 5-21 montre que le temps de propagation à travers un semi-addeur est égal au

temps de propagation t_p d'une porte. D'où, d'après la figure 5-22, le temps de propagation d'un addeur complet est $3t_p$ entre une entrée a ou b et la sortie r et il est égal à $2t_p$ entre r_p et r. Par conséquent, la retenue r_n de l'additionneur de la figure 5-23 ne sera pas obtenue avant $2nt_p$. Cet additionneur appelé *à ondulation* (rippel adder) est relativement lent et nous verrons plus loin comment construire des additionneurs plus rapides.

Représentation des nombres signés. Sachant qu'en circuits logiques on ne peut employer que les symboles 0 et 1, on convient de représenter le signe + par 0 et le signe – par 1. Ce bit de signe occupera la première position à gauche des autres bits qui constituent la magnitude du nombre.

L'opposé d'un nombre non nul positif à n – 1 chiffres de magnitude, $a = 0a_{n-2}\ldots a_1a_0$, est le nombre négatif $b = 1b_{n-2}\ldots b_1b_0$ tel que l'addition de a et b en tant que deux nombres binaires positifs à n chiffres (en supposant que les bits de signe sont des chiffres binaires) donne des 0 sur toutes les n sorties s_{n-1}, s_{n-2}, …, s_1, s_0 de l'additionneur de la figure 5-23. De cette manière, en ignorant la retenue r_n qui sera nécessairement égale à 1, on aura a + b = 0. En d'autres termes, les deux nombres relatifs a et b sont considérés opposés si la somme des deux nombres binaires positifs à n bits $(0a_{n-2}\ldots a_1a_0)_2$ et $(1b_{n-2}\ldots b_1b_0)_2$ est égale à $(10\ \ldots\ 0)_2 = 2^n$, c.à.d.

$$(1b_{n-2}\cdots b_1b_0)_2 = 2^n - (0a_{n-2}\cdots a_1a_0)_2$$

Ainsi, les chiffres de l'opposé b d'un nombre a à n bits (y compris le bit du signe) sont ceux de l'écriture binaire de $2^n - (a)_2$. On dit que b est le *complément-2* de a. Par exemple +6 s'écrit 0110 et –6 s'écrit 1010 car c'est la représentation binaire de $2^4 - 6 = 10$. D'autre part, en remplaçant dans l'équation précédente 2^n par $(11\ldots1)_2 + 1$, elle devient :

$$(1b_{n-2}\cdots b_1b_0)_2 = (11\cdots11)_2 - (0a_{n-2}\cdots a_1a_0)_2 + 1$$

$$\Rightarrow \begin{cases}(1b_{n-2}\cdots b_1b_0)_2 = (\overline{0}\,\overline{a}_{n-2}\cdots\overline{a}_1\overline{a}_0 + 1)_2\\ (0a_{n-2}\cdots a_1a_0)_2 = (\overline{1}\,\overline{b}_{n-2}\cdots\overline{b}_1\overline{b}_0 + 1)_2\end{cases}.$$

Le *complément-1* d'un nombre x est le nombre obtenu en complémentant tous les bits de x (y compris le bit de signe). Le complément-2 s'obtient donc en ajoutant 1 au complément-1. Par exemple, +6 = 0110 et son complément-1 est 1001 d'où –6 s'écrit 1001+1 = 1010 comme nous l'avons déduit plus haut d'une autre manière.

Une règle simple pour obtenir le complément-2 d'un nombre consiste à complémenter les bits situés à gauche du premier 1 rencontré à partir de la droite. Par exemple, –6 = 1010 ⇔ +6 = 0110.

EXERCICE 5-9
Sachant que le complément-2 est égale au complément-1 + 1, vérifier la règle précédente en déterminant le complément-2 de 01011000 = +88.

Additionneur de 2 nombres signés. À noter qu'en complément-2,

$$(00a_{n-2}\cdots a_0)_{c2} = (0a_{n-2}\cdots a_0)_{c2}$$

et

$$\begin{aligned}(11a_{n-2}\cdots a_0)_{c2} &= -(00\overline{a}_{n-2}\cdots\overline{a}_0 + 1)_2\\ &= -(0\overline{a}_{n-2}\cdots\overline{a}_0 + 1)_2\\ &= (1a_{n-2}\cdots a_0)_{c2}\end{aligned}$$

c.à.d. plusieurs bits égaux à gauche d'un nombre en complément-2 peuvent être remplacés par un seul de ces bits.

Si n est le nombre des bits réservés à l'écriture d'un nombre signé, ce nombre doit être compris entre $(10\ldots0) = (110\ldots0) = -2^{n-1}$ et $2^{n-1}-1 = (011\ldots1)$. Quand le résultat d'une opération sort de cet intervalle, on dit qu'on a un *débordement* (overflow).

Considérons deux nombres a et b exprimés en complément-2 :

$$a = (a_{n-1}a_{n-2}\ldots a_1a_0)_{c2} \quad \text{et} \quad b = (b_{n-1}b_{n-2}\ldots b_1b_0)_{c2}$$

où a_{n-1} et b_{n-1} sont les bits de signe. Désignons par r_k la retenue sortante de l'addeur k et considérons les 3 cas suivants.

a > 0, b > 0
Comme $a_{n-1} = b_{n-1} = 0$, la dernière retenue $r_n = 0$. Si la retenue $r_{n-1} = 1$, le résultat de l'additionneur 5-23 sera $01s_{n-2}\ldots s_0 \geq 2^{n-1}$ et on a un débordement car ce

résultat (y compris le bit 0 du signe) nécessite n + 1 bits au lieu de n. Si seulement n bits sont réservés pour l'écriture des nombres, le résultat n'est valable que si $r_n = r_{n-1} = 0$.

<u>a < 0, b < 0</u>
Comme $a_{n-1} = b_{n-1} = 1$, la dernière retenue $r_n = 1$. Si la retenue $r_{n-1} = 0$, le résultat de l'additionneur 5-23 sera $10s_{n-2}\dots s_0 = 110s_{n-2}\dots s_0 < -2^{n-1}$ et on a un débordement. Par contre, si $r_{n-1} = 1$, le résultat sera valable car

$$(11s_{n-2}\cdots s_0)_{c2} = (1s_{n-2}\cdots s_0)_{c2} = -(0\bar{s}_{n-2}\cdots\bar{s}_0 + 1)_2 \geq -2^{n-1}.$$

<u>a ≥ 0, b ≤ 0</u>
Comme $a_{n-1} = 0$ et $b_{n-1} = 1$, on a : $r_n = 1 \Leftrightarrow r_{n-1} = 1$.

Par conséquent, en ignorant r_n, s = a + b sera correct de n bits sans débordement si et seulement si $r_n = r_{n-1}$ et un débordement aura lieu si et seulement si $r_n \neq r_{n-1} \Leftrightarrow r_n \oplus r_{n-1} = 1$.

D'autre part, comme la soustraction a – b se ramène à la somme a + (–b), l'additionneur de la figure 5-23 se transforme en soustracteur si b est remplacé par son complément-2 :

$$-b = (\bar{b}_{n-1}\bar{b}_{n-2}\cdots\bar{b}_0 + 1)_2 .$$

Sachant que $b_k = b_k \oplus 0$ et $\bar{b}_k = b_k \oplus 1$, la figure 5-24 représente un additionneur/soustracteur (l'opération est sélectionnée par $\bar{A}/S$) muni d'une sortie db qui prend la valeur 1 quand une opération déborde.

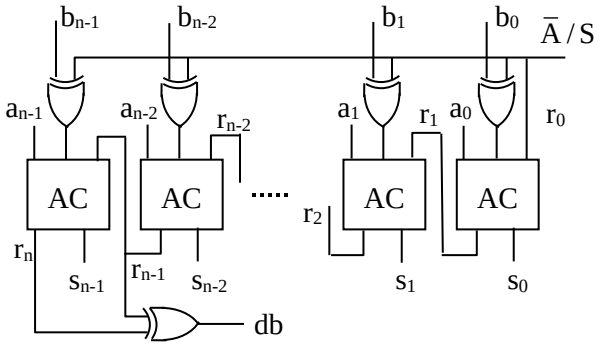


Fig. 5-24 Additionneur/Soustracteur de 2 nombres signés

Additionneur à retenue anticipée. Nous avons vu que l'additionneur à ondulation de la figure 5-23 (ou 5-24) est relativement lent car le $k^{ème}$ addeur complet (AC) attend à son entrée le calcul de la retenue r_k pour produire sa sortie s_k et livrer sa propre retenue r_{k+1} au AC suivant. D'après (5-4) et (5-5),

$$s_k = r_k \oplus p_k , \qquad (5\text{-}6)$$

$$r_{k+1} = r_k p_k + g_k , \quad k = 0,\cdots,n-1, \qquad (5\text{-}7)$$

où $p_k = a_k \oplus b_k$ et $g_k = a_k b_k$ sont les sorties d'un semi-addeur appelées respectivement terme propagateur et terme générateur de la retenue. Au lieu de calculer les retenues r_k d'une manière récurrente, elles peuvent être simultanément obtenues à la sortie d'un circuit réalisant une fonction R d'entrées r_0, p_0, …, p_{n-1}, g_0, …, g_{n-1} et de sorties r_1, …, r_n. Cette fonction se déduit facilement de (5-7) en écrivant :

$$\begin{aligned} r_{k+1} &= (r_{k-1}p_{k-1} + g_{k-1})p_k + g_k \\ &= (r_{k-2}p_{k-2} + g_{k-2})p_{k-1}p_k + g_{k-1}p_k + g_k \\ &= r_{k-2}p_{k-2}p_{k-1}p_k + g_{k-2}p_{k-1}p_k + g_{k-1}p_k + g_k \\ &\vdots \end{aligned}$$

$$\Rightarrow \quad r_{k+1} = r_0 \prod_0^k p_i + \sum_{i=0}^{k-1}\left(g_i \prod_{i+1}^{k} p_j \right) + g_k . \qquad (5\text{-}8)$$

Chaque sortie de la fonction R est donc une somme de produits qui se réalise en 2 niveaux, le premier pour effectuer les produits et le second pour effectuer leur somme.

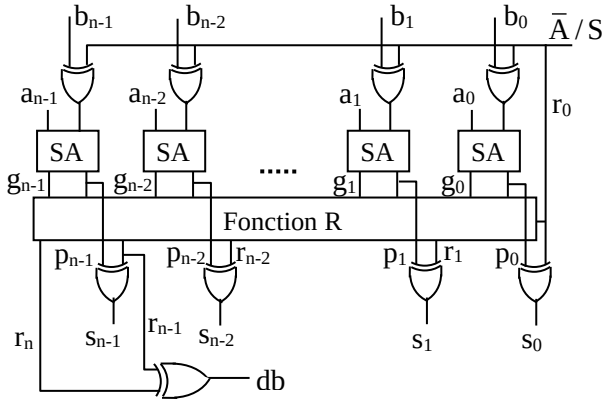


Fig. 5-25 Additionneur/Soustracteur CLA

Tenant compte de (5-6), le circuit de la figure 5-25 effectue les mêmes opérations que le circuit de la

figure 5-24 mais en un temps plus court ($5t_p$ au lieu de $(2n + 3)t_p$). Il est appelé *additionneur à retenue anticipée* (carry lookahead adder, CLA).

Remarque. Pour que le circuit de la fonction R de l'additionneur CLA soit seulement à 2 niveaux, il est nécessaire, d'après (5-8), que des portes AND et OR à n entrées (de fan-in = n) soient disponibles. Pour cette raison, les additionneurs CLA ont généralement un nombre réduit de bits. Cependant, en combinant dans une architecture série ou hiérarchique plusieurs additionneurs CLA, on peut construire des additionneurs plus larges et assez rapides.

EXERCICE 5-10
À l'aide de 4 additionneurs CLA à 4 bits chacun, construire un additionneur 16 bits en connectant un additionneur au précédent par la retenue finale de ce dernier. Comparer les durées de l'addition par le circuit ainsi obtenu et par l'additionneur à ondulation (fig. 5-23).

Additionneur CLA à deux niveaux. Un additionneur CLA (fig.5-25) peut avoir, avec la retenue finale r, deux autres sorties

$$P = \prod_0^{n-1} p_i \quad \text{et} \quad G = \sum_0^{n-2}\left(g_i \prod_{i+1}^{n-1} p_j \right) + g_{n-1}.$$

Il est clair, d'après (5-8), que la retenue finale r est liée à P, G et la retenue d'entrée r_0 par l'équation :

$$r = r_0P + G. \qquad (5\text{-}9)$$

À l'aide de m additionneurs de ce type munis des sorties P_k et G_k, k = 0, …, m – 1, on obtient un additionneur à mn bits en connectant la retenue finale d'un additionneur à l'entrée r_{kn} de l'additionneur k suivant. Pour accélérer le calcul, les retenues r_{kn} sont fournies simultanément par un circuit réalisant une fonction R_1 d'entrées r_0, P_0, P_1, …, P_{m-1}, G_0, G_1, …, G_{m-1} (indépendantes des r_{kn}) et de sorties r_n, r_{2n}, …, r_{mn}. Cette fonction se déduit facilement de (5-9) en écrivant :

$$\begin{aligned} r_{(k+1)n} &= r_{kn}P_k + G_k \\ &= r_{(k-1)n}P_{k-1}P_k + G_{k-1}P_k + G_k = \cdots \end{aligned}$$

$$\Rightarrow \qquad r_{(k+1)n} = r_0 \prod_0^k P_i + \sum_{i=0}^{k-1}\left(G_i \prod_{i+1}^k P_j \right) + G_k .$$

Le circuit de la fonction R_1 a donc la même constitution que celui de la fonction R du CLA à simple niveau (fig. 5-25) et la figure 5-26 représente un additionneur CLA à deux niveaux et à 4x4 bits. À remarquer que ce circuit comporte aussi des sorties P et G permettant de construire des CLA à plusieurs niveaux.

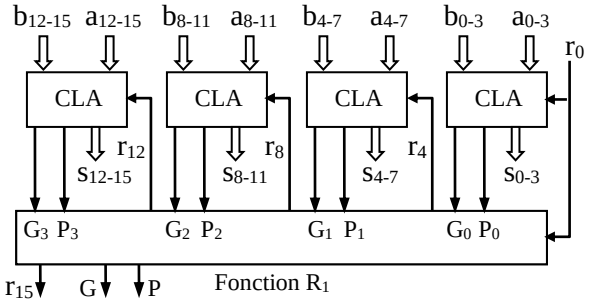


Fig. 5-26 Additionneur CLA à 2 niveaux

EXERCICE 5-11
Comparer la durée de l'addition par le circuit de la figure 5-26 avec celles des circuits de l'exercice 5-10.

Additionneur à sélection de la retenue. La figure 5-27 représente un additionneur 8 bits constitué de 3 additionneurs 4 bits A, B et C (à ondulation ou CLA) et d'un sélecteur constitué de 5 multiplexeurs 2\1. L'additionneur A effectue la somme normale des 4 bits inférieurs et, en même temps, chacun des additionneurs B et C effectue la somme des 4 bits supérieurs mais le premier avec une retenue d'entrée 0 et l'autre avec une retenue d'entrée 1. Si la retenue finale r_4 de A est 0 le sélecteur choisit le résultat de B et si elle est 1, il choisit le résultat de C.

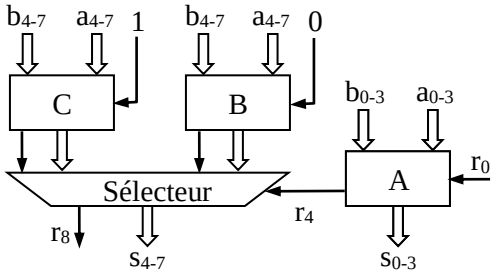


Fig. 5-27 Additionneur à sélection de la retenue

On élargie l'additionneur précédent en le connectant par r_8 à une série de plusieurs autres étages (B, C, sélecteur).

EXERCICE 5-12
Soit t_p le temps de propagation d'une porte logique et admettons que le temps de propagation à travers le sélecteur est $2t_p$. Considérons un additionneur à sélection de la retenue à 3 étages A, (B_0, C_0, sélecteur) et (B_1, C_1, sélecteur). Sachant que les additionneurs sont itératifs, déterminer la durée de l'addition si
a) A, B_0, C_0, B_1, C_1 sont tous à 4 bits,
b) A, B_0, C_0 sont à 4 bits mais B_1 et C_1 sont à 5 bits.

Additionneur BCD. Soit $(s_4s_3s_2s_1s_0)_{BCD}$ l'écriture décimale codée binaire de la somme s de deux chiffres décimaux. On a :

$$s = (s_4s_3s_2s_1s_0)_{BCD} = (s_3s_2s_1s_0)_2 + 10s_4$$
$$= (s_4s_3s_2s_1s_0)_2 - 2^4 s_4 + 10s_4$$
$$\Rightarrow \quad (s_4s_3s_2s_1s_0)_2 = s + 6s_4 \qquad (5\text{-}10)$$

où $s_4 = 1$ si et seulement si $s \geq 10$. Désignons par $(s'_4s'_3s'_2s'_1s'_0)_2$ l'écriture binaire de s et soit f la fonction définie par $f = 1 \Leftrightarrow s \geq 10$, Le tableau de Karnaugh de f est le suivant, les cases indifférentes étant celles des combinaisons qui n'apparaissent pas (où $s > 18$).

s'_4 \ s'_3	s'_0	s'_1	s'_2					
	0	0	0	0	0	0	0	0
	0	0	1	1	1	1	1	1
	1	1		1				

Tableau 5-3 Fonction f qui détecte $s \geq 10$

On déduit de ce tableau que

$$f = s'_4 + s'_3s'_2 + s'_3s'_1 .$$

et d'après (5-10) :

$$(s_4s_3s_2s_1s_0)_2 = \begin{cases} (s'_4s'_3s'_2s'_1s'_0)_2 + (00000)_2 & \text{si } f = 0 \\ (s'_4s'_3s'_2s'_1s'_0)_2 + (00110)_2 & \text{si } f = 1 \end{cases}$$

D'où, d'après (5-4),

$$s_0 = s'_0$$
$$s_1 = s'_1 \oplus f$$
$$s_2 = s'_2 \oplus f \oplus r_2$$
$$s_3 = s'_3 \oplus 0 \oplus r_3 = s'_3 \oplus r_3$$

et on sait que $s_4 = 1 \Leftrightarrow f = 1$.

On déduit que l'additionneur BCD de deux chiffres décimaux codés en binaire a le circuit représenté par la figure 5-28.

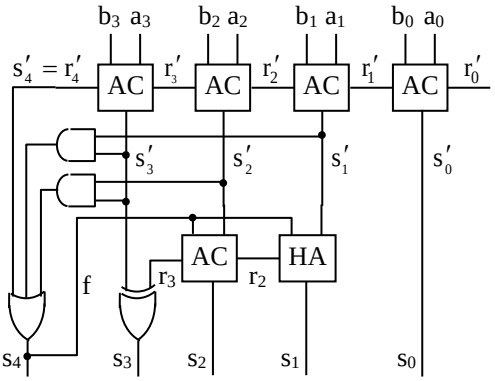


Fig. 5-28 Addeur BCD de 2 chiffres

Un additionneur à n chiffres décimaux codés en binaire est constitué de n addeurs BCD, l'entrée r'_0 d'un addeur étant connectée à la sortie s_4 du précédent. En effet, on ajoute la retenue 1 à la somme des deux chiffres de la position suivante si et seulement si $s \geq 10 \Leftrightarrow s_4 = 1$.

5-6 UNITÉ ARITHMÉTIQUE ET LOGIQUE (ALU)

Dans le processeur central d'un ordinateur (CPU), les calculs arithmétiques et logiques sont exécutés par l'unité arithmétique et logique (ALU). On peut dire que cet élément est le cœur du processeur et les autres parties (registres, compteur d'instructions, décodeur d'instructions, accumulateur, circuit de contrôle) sont à son service. Elles lui fournissent les données, lui indiquent l'opération à exécuter sur ces données et reçoivent le résultat.

Nous commençons par construire autour d'un additionneur une unité arithmétique à 4 bits pouvant exécuter les opérations fondamentales. Les ALU plus larges se construisent d'une manière similaire.

Nous savons que la sortie s = (s_3, s_2, s_1, s_0) d'un additionneur est liée aux données a = (a_3, a_2, a_1, a_0) et b = (b_3, b_2, b_1, b_0) et à la retenue initiale r_0 par la relation

$$s = a + b + r_0 .$$

À partir de cette relation, on peut réaliser plusieurs opérations arithmétiques en modifiant b et r_0. Le tableau suivant donne pour 4 transformations β du vecteur b les opérations qu'effectue l'additionneur pour $r_0 = 0$ et pour $r_0 = 1$. Chaque opération est codée en première colonne par un vecteur (c_2, c_1, c_0) où c_2 et c_1 sont relatifs à β et c_0 est la valeur de r_0.

c_2 c_1 c_0	β	s= a + β + c_0	Opération
0 0 0	0000	a	Transfert
0 0 1	0000	a + 1	Incrémentation
0 1 0	b	a + b	Addition
0 1 1	b	a + b + 1	Addition + 1
1 0 0	$\overline{b}$	a + $\overline{b}$	Soustraction – 1
1 0 1	$\overline{b}$	a + $\overline{b}$ + 1	Soustraction
1 1 0	1111	a – 1	Décrémentation
1 1 1	1111	a	Transfert

Les deux premières colonnes du tableau précédent conduisent au tableau de Karnaugh de β_i en fonction de c_2, c_1 et b_i :

b_i \ c_1 c_2	0 0	0 1	1 1	1 0
0	0	1	1	0

duquel on déduit que

$$\beta_i = c_1 b_i + c_2 \overline{b}_i + c_1 c_2 .$$

Le circuit de l'unité arithmétique est donc celui de la figure 5-29.

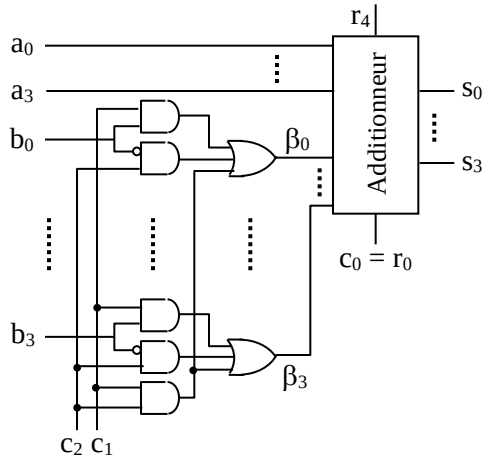


Fig. 5-29 Unité arithmétique

La figure 5-30 représente le circuit d'une unité logique qui effectue l'une des 4 opérations NOT, AND, OR et XOR codées par c_0 et c_1. La même opération est choisie par les multiplexeurs 4/1 pour toutes les composantes des données a et b.

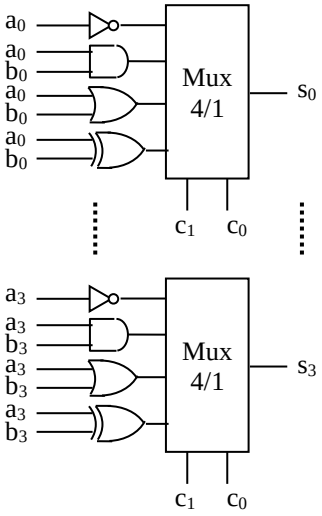


Fig. 5-30 Unité logique

Enfin la figure 5-31 représente le circuit complet de cet ALU. Il regroupe l'unité arithmétique, l'unité logique et un sélecteur, constitué de 4 multiplexeurs 2/1, qui fournit à sa sortie le résultat arithmétique ou logique selon la valeur de l'entrée de sélection c_3.

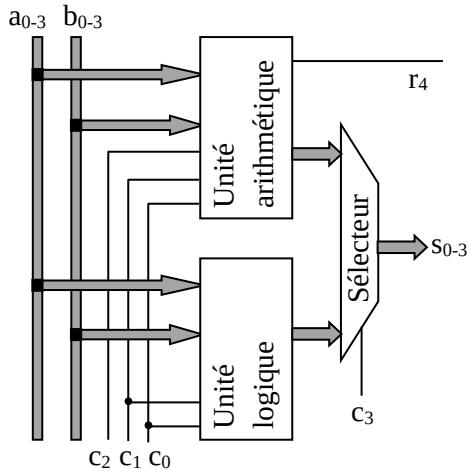


Fig. 5-31 ALU à 4 bits

5-7 MULTIPLICATION ET DIVISION

Multiplicateur combinatoire. La multiplication binaire s'effectue selon le même processus que la multiplication décimale mais en plus simple puisque la multiplication d'un nombre x par 0 ou 1 est égale à 0 ou x. Voici, comme exemple, la multiplication en binaire de 13x11 (= 143).

				1	1	0	1	Multiplicande
				1	0	1	1	Multiplieur
				1	1	0	1	
			1	1	0	1		Produits partiels
		0	0	0	0			
	1	1	0	1				
1	0	0	0	1	1	1	1	Produit

Sachant que le produit arithmétique $a_i x b_i$ de deux bits est égal à leur produit logique $a_i.b_i$, les produits partiels de deux nombres $a = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)_2$ et $b = (b_{n-1}, b_{n-2}, \ldots, b_1, b_0)_2$ sont les nombres binaires $p_i = (a_{n-1}.b_i, a_{n-2}.b_i, \ldots, a_1.b_i, a_0.b_i)_2$, $i = 0, \ldots, n-1$, chacun pouvant être obtenu au moyen de n portes AND. Le produit final $p = (p_{2n-1}, p_{2n-2}, \ldots, p_1, p_0)$ est la somme de ces produits partiels l'un étant décalé par rapport au précédent de 1 bit vers la gauche. Ceci conduit au multiplieur combinatoire à 4 bits représenté par la figure 5-32.

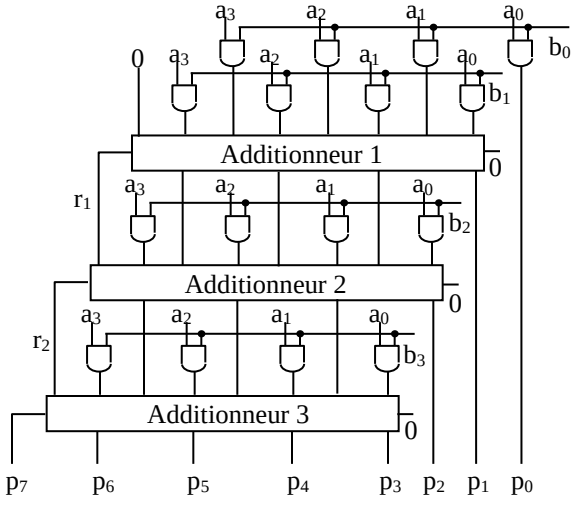


Fig. 5-32 Multiplieur combinatoire

Le nombre de portes d'un multiplicateur combinatoire augmente rapidement avec la longueur n des opérandes puisqu'il est proportionnel à n^2. D'autre part, comme le prouve l'exemple suivant, ce multiplicateur donne un résultat erroné si l'un des opérandes est négatif et écrit en complément-2.

		0	1	1	$= (+3)_{10}$
		1	1	0	$= (-2)_{10}$ $\quad (+3)\ (-2) = (-6)_{10}$
	0	1	1	0	
0	1	1			
1	0	0	1	0	$= -(01110) \quad = (-14)_{10}$

Pour obtenir un résultat correct, on multiplie d'abord les valeurs absolues du multiplicande et du multiplieur puis, si leurs signes diffèrent, on remplace le résultat par son complément-2. Cependant, la conversion en valeurs absolues des opérandes et la comparaison de leurs signes puis l'écriture éventuelle du résultat en complément-2 compliquent davantage le circuit du multiplicateur.

Pour cette raison, dans les machines de calcul numérique, on préfère effectuer la multiplication d'une manière séquentielle à l'aide d'un programme qui gère les opérations des registres et de l'unité arithmétique et logique (ALU) du processeur.

Multiplicateur séquentiel. Ce multiplicateur est basé sur les remarques suivantes :

- Un bit du multiplieur n'intervient dans le processus de la multiplication qu'une seule fois. Il peut être supprimé à l'étape suivante.
- Si le bit le plus à droite des bits restants du multiplieur est 0, le produit partiel est nul et s'il est 1, le produit partiel est égal au multiplicande.
- Décaler le produit partiel à gauche pour l'ajouter à la somme partielle revient à décaler cette somme vers la droite en fixant la position des produits partiels c.à.d. du multiplicande.
- Les bits d'une somme partielle situés à droite du multiplicande restent les mêmes dans les sommes partielles suivantes et ne seront plus affectés. Il est donc inutile d'en tenir compte dans les additions futures.

Effectuons de nouveau le produit 13x11 mais en organisant le calcul tenant compte des remarques précédentes. À chaque étape i, on ajoute le produit partiel (le multiplicande ou 0 selon la valeur du bit de droite du multiplieur restant) à la partie de la somme partielle sp_i située sous le multiplicande (MD) en

plaçant la retenue finale dans une colonne r. On décale ensuite vers la droite (D) l'ensemble somme partielle et multiplieur (MR).

MD→		1	1	0	1					
	r	0	0	0	0	1	0	1	1	←MR
+	0	1	1	0	1	1	0	1	1	
D		0	1	1	0	1	1	0	1	←sp_1
+	1	0	0	1	1	1	1	0	1	
D		1	0	0	1	1	1	1	0	←sp_2
	0	1	0	0	1	1	1	1	0	
D		0	1	0	0	1	1	1	1	←sp_3
+	1	0	0	0	1	1	1	1	1	
D		1	0	0	0	1	1	1	1	143

Il est évident que ce processus n'est pas avantageux pour le calcul manuel mais, moyennant seulement 2 registres, une bascule et un ALU, il peut se traduire en un algorithme efficace. Comme le montre la figure 5-33, le premier registre A est de n bits et contient le multiplicande. Le registre P est de 2n bits et contient initialement le multiplieur dans sa moitié droite C et des 0 dans sa moitié gauche B. La bascule F d'état initialement nul est connectée en série à P et lui transmet son état lors d'un décalage. Le circuit de contrôle décode successivement les instructions de l'algorithme et gère leur exécution. p_0 est le bit de P à l'extrême droite et r est la retenue sortant de l'ALU et mémorisée dans F.

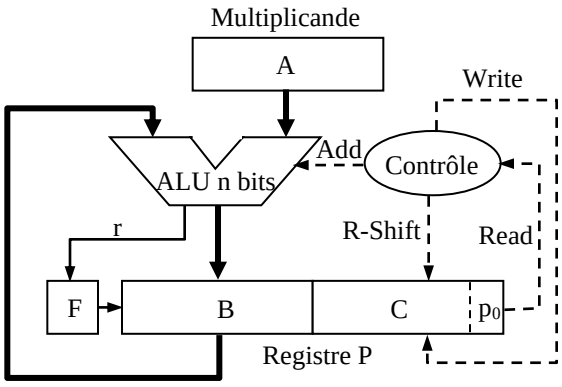


Fig. 5-33 Multiplicateur séquentiel

L'algorithme est le suivant :

```
for i = 1 to n,
    if p0 = 1, B ← B + A, F ← r
    else B ← B, F ← 0
    end
    Décaler F et P de 1 bit vers la droite
end
```

Ici aussi l'algorithme n'est valable que pour des opérandes positifs. Un opérande négatif doit être remplacé par sa valeur absolue (son complément-2) et si les signes des opérandes diffèrent, le résultat sera remplacé par son complément-2. L'algorithme de Booth que nous allons maintenant détailler, est valable pour des opérandes de signes quelconques. Il évite les complémentations et minimise en même temps le nombre d'additions à effectuer.

Algorithme de Booth. La valeur d'un nombre b exprimé en complément-2 par $(b_{n-1}b_{n-2}\ \dots\ b_1b_0)_{c2}$, est

$$b = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \ . \qquad (5\text{-}11)$$

Cette relation est évidemment vraie pour $b \geq 0$ puisque, dans ce cas, $b = (0b_{n-2}\ \dots\ b_1b_0)_2$. D'autre part, pour $b < 0$, $b_{n-1} = 1$ et on a :

$$(b_{n-1}b_{n-2}\cdots b_1b_0)_2 = 2^n - |b| = b_{n-1}2^n - |b|$$

$$\Rightarrow \quad b = -|b| = -b_{n-1}2^n + b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

$$= -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Sachant que $b_i2^i = b_i2^{i+1} - b_i2^i$, l'équation 5-11 peut aussi s'écrire sous la forme

$$b = (-b_{n-1} + b_{n-2})2^{n-1} + \sum_{i=0}^{n-2} (-b_i + b_{i-1})2^i$$

$$= -\sum_{i=0}^{n-1} \delta_i 2^i \quad \text{avec} \quad \delta_i = (b_i - b_{i-1}) \quad \text{et} \quad b_{-1} = 0.$$

Le produit d'un multiplicande a par un multiplieur b est donc donné par

$$axb = -\sum_{i=0}^{n-1} 2^i.(\delta_i a), \quad \begin{cases} \delta_i a & i^{ème} \text{ ligne} \\ 2^i & i^{ème} \text{ décalage} \end{cases}.$$

Par conséquent, à la $i^{ème}$ itération,

- Si $\delta_i = 0$ ($b_i = b_{i-1}$), aucun calcul ne sera effectué mais seulement un décalage.
- Si $\delta_i = +1$ ($b_i = 1$ et $b_{i-1} = 0$), a sera soustrait de la somme partielle puis un décalage.
- Si $\delta_i = -1$ ($b_i = 0$ et $b_{i-1} = 1$), a sera ajouté à la somme partielle puis un décalage.

Le procédé de calcul est semblable à celui de l'algorithme précédent à part que δ_i dépend non seulement du bit courant b_i mais aussi du bit précédent b_{i-1} et il faut donc réserver une place à ce dernier. Le processeur exécute ce calcul moyennant les mêmes éléments que dans la figure 5-31 plus une bascule F_p dont l'état initial est 0 et l'état courant est égal au bit le moins significatif du multiplieur précédent. Ce bit sera désigné par p_{-1} et, comme ci-dessus, on désignera par p_0 le bit le moins significatif du registre P.

Les calculs sont exécutés en complément-2 et pour obtenir après le décalage un résultat toujours correct il est nécessaire d'introduire dans F le bit de signe s de B + A ou de B – A. Ce bit se déduit en ajoutant implicitement à gauche de chaque opérande un bit égal à son bit de signe avant d'exécuter l'opération (+ ou –). À titre d'exemple, effectuons en complément-2 les opérations 13 + 8 = 21 et (–13) + (– 8) = –21. Sans le calcul du signe s, on obtient des faux résultats :

$$\begin{array}{r} 0\,1\,1\,0\,1 \\ +\ 0\,1\,0\,0\,0 \\ \hline 1\,0\,1\,0\,1 \end{array} = -11 \qquad \begin{array}{r} 1\,0\,0\,1\,1 \\ +\ 1\,1\,0\,0\,0 \\ \hline 0\,1\,0\,1\,1 \end{array} = +11$$

En calculant le signe s les résultats seront corrects :

$$\begin{array}{r} 0\,0\,1\,1\,0\,1 \\ +\ 0\,0\,1\,0\,0\,0 \\ \hline 0\,1\,0\,1\,0\,1 \end{array} = +21 \qquad \begin{array}{r} 1\,1\,0\,0\,1\,1 \\ +\ 1\,1\,1\,0\,0\,0 \\ \hline 1\,0\,1\,0\,1\,1 \end{array} = -21$$

Ceci montre que $s = a_{n-1} \oplus p_{2n-1} \oplus r_n$ qui s'obtient à travers une porte XOR à trois entrées.

L'algorithme de Booth est donc le suivant :

```
for i = 1 to n,
    if p0p–1 = 00, B ← B. F ← p2n–1
    elseif p0p–1 = 01, B ← B + A, F ← s
    elseif p0p–1 = 10, B ← B – A, F ← s
    elseif p0p–1 = 11, B ← B, F ← p2n–1
    end
    Décaler F, P et Fp de 1 bit vers la droite
end
```

Effectuons selon cet algorithme le produit 11x(–13) = –143.

		0	0	1	0	1	1	←MD					F_p↓	
	s	0	0	0	0	0	1	0	0	1	1	0	MR
–	1	1	0	1	0	1	1	0	0	1	1	0	
D		1	1	0	1	0	1	1	0	0	1	1	sp_1
	1	1	1	0	1	0	1	1	0	0	1	1	
D		1	1	1	0	1	0	1	1	0	0	1	sp_2
+	0	0	1	0	0	0	0	1	1	0	0	1	
D		0	0	1	0	0	0	0	1	1	0	0	sp_3
	0	0	0	1	0	0	0	0	1	1	0	0	
D		0	0	0	1	0	0	0	0	1	1	0	sp_4
–	1	1	0	1	1	1	0	0	0	1	1	0	
D		1	1	0	1	1	1	0	0	0	1		–143

EXERCICE 5-13
Calculer le produit 22x30 par multiplication séquentielle et le produit (□22)x30 par l'algorithme de Booth.

Division. En binaire la division est plus simple qu'en décimal puisque chaque digit du quotient est recherché parmi deux chiffres (0 et 1) au lieu de 10 (0 à 9). Effectuons par exemple la division 71÷13 :

Dividende	Diviseur
1 0 0 0 1 1 1 (= 71)	1 1 0 1 (= 13)
1 1 0 1	0 1 0 1 (= 5) Quotient
0 1 0 0 1 1	
1 1 0 1	
0 0 1 1 0 (= 6) Reste	

Pour justifier ce calcul, considérons d'abord le cas où le dividende a et le diviseur b sont des entiers strictement positifs tels que b ≤ a. Le quotient q et le reste r sont alors des entiers non négatifs vérifiant :

$$a = bq + r \quad \text{avec} \quad r < b. \qquad (5\text{-}12)$$

Si n est le nombre de bits de a, m le nombre de bits de b (m ≤ n) et q = (q_{p-1}, …, q_0), la relation précédente peut aussi s'écrire sous la forme :

$$a = b\sum_{0}^{p-1} q_i 2^i + r = b\sum_{p-k}^{p-1} q_i 2^i + r^k \quad \text{où} \quad p = n - m + 1.$$

où $\quad r^k = b\sum_{0}^{p-k-1} q_i 2^i + r$, k = 0, …, p – 1

Il est évident que $r^k \geq 0\ \forall k$ et que :

$$r^k = r^{k-1} - bq_{p-k}2^{p-k} \quad \text{avec} \quad r^0 = a \quad \text{et} \quad r^p = r\,.$$

r^k est le reste à l'itération k et soit $\Delta^k = r^{k-1} - b2^{p-k}$. Comme $r^k \geq 0\ \forall k$, on déduit de la relation précédente que

$$\begin{cases} q_{p-k} = 1 \text{ et } r^k = \Delta^k & \text{si} \quad \Delta^k \geq 0, \\ q_{p-k} = 0 \text{ et } r^k = r^{k-1} & \text{si} \quad \Delta^k < 0. \end{cases} \qquad (5\text{-}13)$$

D'après ces relations, on détermine récursivement les bits q_i du quotient et les restes r^k à partir du signe de la différence Δ^k entre le reste précédent r^{k-1} et le diviseur b décalé de p – k bits vers la gauche. Pour exécuter efficacement ce calcul dans un processeur on se base sur les remarques suivantes.

- Au lieu de décaler le diviseur b d'un bit vers la droite après chaque itération, on le maintient fixe et on décale le reste r^k d'un bit vers la gauche. Ceci laisse une place à droite du registre des restes pour y introduire le nouveau bit du quotient.
- D'après (5-13), si $\Delta^k \geq 0$, $r^k = \Delta^k$, d'où

$$\Delta^{k+1} = \Delta^k - b2^{p-k-1}\,.$$

Il suffit donc, pour obtenir Δ^{k+1}, de décaler Δ^k d'un bit vers la gauche et d'en soustraire le diviseur b. Par contre, si $\Delta^k<0$, Δ^{k+1} s'obtient en ajoutant $b2^{p-k}$ à Δ^k pour restituer r^{k-1} (= r^k) avant de calculer $\Delta^{k+1} = r^{k-1} - b2^{p-k-1}$. Mais, on peut éviter cette restitution en remarquant que

$$\Delta^{k+1} = \underbrace{\Delta^k + b2^{p-k}}_{r^k = r^{k-1}} - b2^{b-k-1} = \Delta^k + b2^{b-k-1}.$$

Ainsi, quand $\Delta^k < 0$, il suffit de décaler Δ^k d'un bit vers la gauche et de lui ajouter le diviseur b (au lieu de le soustraire). Cependant, à la dernière itération où k = p (k + 1 n'existe plus), si $\Delta^p < 0$, q_0 = 0 mais $r = r^p = r^{p-1} = \Delta^p + b2^{p-p} = \Delta^p + b$, d'où le reste final s'obtient en ajoutant à Δ^p le diviseur b (sans décalage). Par contre, si $\Delta^p \geq 0$, la division se termine normalement avec $q_0 = 1$ et $r = r^p = \Delta^p$.

Le processeur exécute ce calcul moyennant les mêmes éléments que dans la figure 5-31 à part que le décalage est vers la gauche. Le registre A de m bits contient le diviseur avec des 0 à gauche et le registre P de n = 2m – 1 bits contient initialement le dividende avec des zéros à gauche. La bascule F reçoit le bit de signe de Δ^k, $s = \text{reste}(\Delta^k) \oplus P_n$, et le nouveau bit du quotient est égal à $\bar{s}$. Ce dernier est introduit à la position la moins significative de P. En désignant par B les m bits supérieurs de P, l'algorithme est le suivant :

```
for k = 1 to p – 1,
    décaler F et P à gauche
    if s = 0, B = B – A, F ← s, p0 = 1
    else B = B + A, F ← s, p0 = 0
    end
end
if s = 1 (à la dernière itération p)
   B = B + A, F ← s
end
```

Le tableau suivant montre la modification du continu de la bascule F et du registre P des Δ^k (et du quotient partiel) au cours de la division 71÷11.

	0	0	1	0	1	1	←Dvr			$\bar{s}$	
	s	0	0	1	0	0	0	1	1	1	←Dvd
D	0	0	1	0	0	0	1	1	1		
–	1	1	1	1	0	1	1	1	1	0	
D	1	1	1	0	1	1	1	1	0		
+	0	0	0	1	1	0	1	1	0	1	
D	0	0	1	1	0	1	1	0	1		
–	0	0	0	0	1	0	1	0	1	1	
D	0	0	0	1	0	1	0	1	1		
–	1	1	1	0	1	0	0	1	1	0	
+	0	0	0	1	0	1	0	1	1	0	

Comme nous l'avons signalé plus haut, cet algorithme n'est valable que pour des opérandes a et b positifs. Dans le cas général, on calcule le quotient q' et le reste r' de la division |a|÷|b| et on déduit le quotient q et le reste r de la division a÷b des relations suivantes :

$$q = \begin{cases} q' & \text{si } \text{sgn}(a) = \text{sgn}(b) \\ -q' & \text{si } \text{sgn}(a) \neq \text{sgn}(b) \end{cases} \qquad r = \begin{cases} r' & \text{si } a > 0 \\ -r' & \text{si } a < 0 \end{cases}$$

Ces relations proviennent du fait que bq et r doivent être de même signe que a.

EXERCICE 5-14
Construire le tableau de la division 83÷14 par l'algorithme ci-dessus.

5-8 ARITHMÉTIQUE EN VIRGULE FLOTTANTE

Jusqu'ici nous n'avons considéré que des opérations arithmétiques appliquées à des entiers positifs ou négatifs. Quand il s'agit de nombres fractionnaires, l'exécution de ces opérations par un processeur dépend de la façon dont ces nombres sont représentés. La représentation presque universellement adoptée actuellement est la virgule flottante (VF) selon le standard IEEE 754. Cette représentation des nombres fractionnaires a l'avantage d'accélérer le calcul et de minimiser les erreurs d'arrondi.

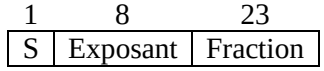
Le format IEEE 754 de la virgule flottante. Tout nombre réel non nul x peut s'écrire sous la forme dite *normalisée* :

$$x = \pm\left(1 + \sum_{i=1}^{\infty} x_{-i}\, 2^{-i}\right) 2^{k}, \quad x_{-i} \in \{0,1\},\ k \in \mathbb{Z}. \quad \text{(5-14)}$$

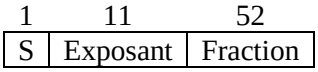
Par exemple,

$$(22.3)_{10} = +(10110.010011001100....)_2$$
$$= +(1.0110010011001100...)_2\, 2^4$$
$$-(0.8125)_{10} = -(0.1101)_2 = -(1.101)_2\, 2^{-1}$$

Dans (5-14), le nombre entre parenthèses $1.x_{-1}x_{-2}x_{-3}\ldots$ est appelé *mantisse* M et la puissance k de 2 est appelée *exposant effectif*. La partie fractionnaire $0.x_{-1}x_{-2}x_{-3}\ldots$ de la mantisse est appelée fraction F. Il est évident que dans un processeur on ne peut réserver qu'un nombre limité de bits pour représenter un nombre. Ceci implique qu'une partie de l'ensemble des réels ne peut être représentée qu'approximativement. Le format IEEE 754 réserve 32 bits pour la représentation des nombres en *simple précision* et 64 bits pour la représentation des nombres en *double précision*. Ces bits sont subdivisés en 3 parties : un bit S pour le signe, p bits pour l'exposant E et m bits pour la fraction F. À simple précision, p = 8 et m = 23 et à double précision, p = 11 et m = 52.



Simple précision



Double précision

- Le bit de signe S est égal à 0 si le nombre représenté est positif et il est égal à 1 si ce nombre est négatif.
- L'exposant E est un nombre binaire de p bits compris entre 1 = (0…01) et $(2^p - 2) = (1\ldots10)$. Les exposants E = 0 = (00 … 0) et $E = (2^p - 1) =$ (11 … 1) sont réservés à des nombres dits *non normalisés* que nous définirons plus loin. À noter que E n'est pas la puissance effective k de 2 dans (5-14) mais c'est une valeur *biaisée* de cette puissance égale à $k + (2^{p-1} - 1)$. D'où,

$$k = E - 127 \quad \text{en simple précision,}$$
$$k = E - 1023 \quad \text{en double précision.}$$

Comme $1 \le E \le 2^p - 2$, la puissance effective k des nombres normalisés est un entier relatif tel que $-(2^{p-1} - 2) \le k \le +(2^{p-1} - 1)$ c.à.d.

$$-126 \le k \le 127 \quad \text{en simple précision,}$$
$$-1022 \le k \le 1023 \quad \text{en double précision.}$$

- Les bits $x_{-1}x_{-2}x_{-3}\ldots,x_{-m}$ de la partie fractionnaire de la mantisse sont placés successivement à partir de la gauche dans la région de la fraction F. La partie entière de la mantisse M de tout nombre normalisé étant égale à 1, elle est sous entendue et on a :

$$M = 1 + F.$$

Ainsi, en désignant par $B = 2^{p-1} - 1$ le biais de E, la valeur du nombre représenté par (S, E, F) est

$$x = (-1)^S M 2^k = (-1)^S (1 + F) 2^{E-B}. \qquad \text{(5-15)}$$

Par exemple, la valeur du nombre représenté en simple précision par

$$\underset{s}{1}\ \underbrace{1000\ 0010}_{E}\ \underbrace{1010\ 1100\ 0000\ 0000\ 0000\ 000}_{F}$$

est

$$x = (-1)^1(1+2^{-1}+2^{-3}+2^{-5}+2^{-6})2^{130-127} = -13.375.$$

Les arrondis. Quand la fraction d'un nombre comporte plus que m bits, on ne peut que représenter une approximation de ce nombre en arrondissant sa fraction à m bits. Le IEEE 754 permet 4 modes d'arrondi.

a) Arrondi par défaut (vers $-\infty$)
C'est le plus grand nombre machine inférieur ou égal au nombre exact x. Sa mantisse sera désignée par M_{df}.
b) Arrondi par excès (vers $+\infty$)
C'est le plus petit nombre machine supérieur ou égal au nombre exact x. Sa mantisse sera désignée par M_{ex}.
c) Arrondi vers 0
C'est l'arrondi par défaut si $x > 0$ et l'arrondi par excès si $x < 0$. Sa mantisse sera désignée par M_{zr}.
d) Arrondi au plus près
C'est le nombre machine le plus proche de x. Sa mantisse sera désignée par M_{pp}. Si x est juste au milieu de 2 nombres machines consécutifs, on choisit pour M_{pp} la mantisse de celui dont la fraction se termine par 0 (arrondi pair).

Ces définitions sont équivalentes aux formules suivantes où M désigne la mantisse de x :

Si $x > 0$,
- $M_{df} = 1.x_{-1}x_{-2} \ldots x_{-m}$
- $M_{ex} = M_{df} + 2^{-m}$
- $M_{zr} = M_{df}$.
- Soit $M_{moy} = (M_{df} + M_{ex})/2 = M_{df} + 2^{-(m+1)}$.
  Si $(M < M_{moy})$ ou $(M = M_{moy}$ et $x_{-m} = 0)$,
  $M_{pp} = M_{df}$.
  Si $(M > M_{moy})$ ou $(M = M_{moy}$ et $x_{-m} = 1)$,
  $M_{pp} = M_{ex}$.

Si $x < 0$
- $M_{df} = -(-M)_{ex}$.
- $M_{ex} = -(-M)_{df}$.
- $M_{zr} = M_{ex}$.
- $M_{pp} = -(-M)_{pp}$.

Le mode M_{pp} est le plus précis mais son calcul est le plus long.

EXERCICE 5-15
Représenter en simple précision les 4 modes d'arrondi de $+(25.4)_{10}$ et de $-(25.4)_{10}$.

Les nombres non normalisés. Comme, pour les nombres normalisés, $-(2^{p-1} - 2) \le k \le +(2^{p-1} - 1)$ et $1 \le M \le 1+\Sigma_{i=1}^{m} 2^{-i} = 2 - 2^{-m}$, les valeurs absolues $|x|$ de ces nombres sont telles que

$2^{-126} \le |x| \le (2 - 2^{-23})2^{127}$ en simple précision,
$2^{-1022} \le |x| \le (2 - 2^{-52})2^{1023}$ en double précision.

Les nombres dont la valeur absolue est à l'extérieur de ces intervalles sont représentés en IEEE 754, selon les conventions suivantes :

a) ± 0
$S = \pm 1$, $E = 0 = (0 \ldots 0)$, $F = 0 = (0 \ldots 0)$.
b) $\pm \infty$
$S = \pm 1$, $E = 2^p - 1 = (1 \ldots 1)$, $F = 0 = (0 \ldots 0)$.
c) NaN (Not a Number)
$S = \pm 1$, $E = 2^p - 1 = (1 \ldots 1)$, $F \neq 0$.
C'est le résultat d'une opération invalide comme 0/0 ou ∞/∞.
d) Nombres dénormalisés
La mantisse est de la forme $0.x_{-1}x_{-2} \ldots x_{-m}$ et
$S = \pm 1$, $E = 0 = (0 \ldots 0)$, $F \neq 0$.
Ce sont des nombres très voisins de 0 servant à remplir le gap séparant 0 des nombres normalisés les plus voisins.

Addition/soustraction en VF. Les opérations d'addition/soustraction de deux nombres en VF s'effectuent comme suit :

- Tester l'existence d'un zéro. Si l'un des opérandes est 0, le résultat est l'autre opérande
  $(a + 0 = a,\ a - 0 = a,\ 0 - b = -b)$.
- Égaliser les exposants. Si l'exposant E_a d'un opérande a est inférieur à l'exposant E_b de l'autre opérande b, on décale la mantisse de a vers la droite de $E_b - E_a$ bits (les exposants seront alors égaux).
- Addition/soustraction des mantisses. Cette opération est exécutée par l'ALU comme pour les nombres entiers. Si la mantisse obtenue est 0, l'exposant E du résultat sera aussi ramené à 0 pour que le résultat soit ± 0.
- Normaliser le résultat. Décaler la mantisse du résultat à droite ou à gauche de sorte que sa forme

soit normalisée et modifier en conséquence l'exposant afin de réajuster le résultat. Si l'exposant E s'annule indiquer une sous-capacité (résultat dénormalisé) et s'il dépasse $(2^p - 2)$ indiquer un débordement (infini ou NaN).

- Arrondir le résultat quand le nombre de bits de la fraction F dépasse m (le choix du mode d'arrondi est facultatif dépendant de l'importance qu'on accorde à la rapidité du calcul ou à sa précision).

En suivant ce procédé, effectuons la différence entre les deux nombres

$46.75 = (1.0111\ 0110)2^5$ et
$9.3125 = (1.0010\ 1010)2^3$

en supposant que p = m = 8.

1) Aucun des opérandes n'est nul.
2) Égalisation des exposants :

$9.3125 = (0.0100\ 1010\ 10)2^5$.

3) Soustraction des mantisses (addition de 46.75 avec le complément-2 de 9.3125) :

$01.01110110 + 11.1011010110 = 01.0010\ 1011\ 10.$

4) La mantisse du résultat est positive et elle est déjà normalisée d'où E = 5 +127 = 132.
5) Comme la fraction du résultat a plus que m = 8 bits, arrondissons au plus près :

$M = M_{df} + 2^{-9}$, $M_{df} = 01.0010\ 1011.$
$M_{ex} = M_{df} + 2^{-8} \Rightarrow M_{moy} = (M_{df} + M_{ex})2^{-1} = M.$
$\Rightarrow M_{pp} = M_{ex}$ (arrondi pair)
$\Rightarrow M_{pp} = 01.0010\ 1011 + 2^{-8}$
$= 01.0010\ 1100$

La représentation de la différence arrondie est donc

$$\underset{S}{0}\ \underbrace{1000\ 0100}_{E}\ \underbrace{0010\ 1100}_{F}$$

dont la valeur est $(1+ 2^{-3} + 2^{-5} + 2^{-6})2^5 = 37.5$ tandis que la valeur exact est 46.75 – 9.3125 = 37.4375. L'erreur absolue est donc 0.0625 et l'erreur relative est $0.0625/37.4375 \approx 1.6(10^{-3})$. L'erreur relative de l'arrondi est inférieure à 2^{-m} qui vaut environ 10^{-7} pour m = 23 et $2(10^{-16})$ pour m = 52. Il est évident que dans un calcul à plusieurs opérations les erreurs d'arrondi peuvent s'accumuler.

Multiplication en VF. Le procédé est essentiellement constituée des étapes suivantes :

- Existence d'un zéro. Si l'un des deux nombres est nul le résultat est nul (E = F = 0).
- Addition des exposants. C'est l'addition $E_a + E_b$ de deux nombres entiers positifs.
- Multiplication des mantisses. Cela revient à multiplier les entiers $(1F_a)_2$ et $(1F_b)_2$ et à placer la virgule après le bit 2m à partir de la droite.
- Normalisation du résultat. Décaler, si nécessaire, la mantisse du résultat à droite de sorte que sa forme soit normalisée et augmenter l'exposant afin de réajuster le résultat. Si l'exposant dépasse $(2^p - 2)$ indiquer un débordement.
- Arrondi du résultat.
- Signe. S = 0 ou 1 selon les signes des deux opérandes.

Division en VF. Le procédé est essentiellement constituée des étapes suivantes :

- Existence d'un zéro. Si seul le dividende a est nul, le résultat est nul. Si seul le diviseur b est nul, le résultat est infini. Si a et b sont nuls, le résultat est NaN.
- Soustraction des exposants. C'est la soustraction entière de l'exposant de b de l'exposant de a.
- Division des mantisses. Cela revient à diviser l'entier $(1F_a)_2$ par l'entier $(1F_b)_2$ en continuant la division après la virgule jusqu'à ce que la somme des bits de la partie entière et de la partie fractionnaire du quotient dépasse m + 2.
- Normalisation du résultat. Décaler la mantisse du résultat à droite ou à gauche de sorte que sa forme soit normalisée et modifier l'exposant afin de réajuster le résultat. Si l'exposant E est négatif ou nul indiquer une sous-capacité (résultat dénormalisé) et s'il dépasse $(2^p - 2)$ indiquer un débordement (infini ou NaN).
- Arrondi du résultat.
- Signe. S = 0 ou 1 selon les signes des deux opérandes.

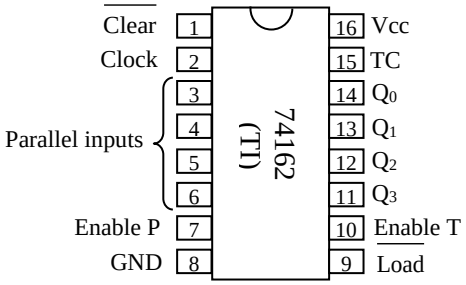
AUTRES EXERCICES ET COMPLÉMENTS

5-16 Après chaque impulsion de l'horloge, un compteur synchrone passe à la valeur suivante de la séquence 0, 2, 4, 6, 1, 3, 5, 7, 0, 2, 4, … . Pour réaliser ce compteur avec des bascules JK,

a) Dresser le tableau de l'état suivant en fonction de l'état actuel.
b) Dresser les tableaux des J_i et K_i des bascules et déduire ces fonctions.
c) Vérifier les résultats précédents en examinant la suite des mots binaires que doit produire le compteur
d) Représenter le circuit.

5-17 Construire un décompteur synchrone modulo 10 (9→8→…→0→9…) qui, par une impulsion sur une entrée d'initialisation I, ramène l'état de décompteur à 9 à partir de tout autre état.

5-18 La figure représente la puce d'un compteur BCD (modulo 10) complètement synchrone à front descendant (negative-going edge).



Les broches ont les rôles suivants :

- Un 0 sur 1 annule les sorties Q_0, Q_1, Q_2, Q_3 au front montant de l'horloge.
- $(Q_3Q_2Q_1Q_0)_2$ est le nombre modulo 16 des impulsions de l'horloge qui arrivent à 2.
- Un 0 sur la broche 9 transmet, au front montant, le mot binaire placé sur les broches 3, 4, 5 et 6 aux sorties Q_0, Q_1, Q_2 et Q_3 dans cet ordre.
- Le compteur n'avance que si les broches 7 et 10 sont tous les deux à 1.
- La sortie 15 (terminal compte TC) n'est égale à 1 que lorsque $(Q_3Q_2Q_1Q_0)_2$ est à sa valeur maximum 9.

Utilisant deux puces 74162, l'un pour les unités et l'autre pour les dizaines, et une porte NAND,

a) Construire pour les secondes ou les minutes d'une montre digitale un compteur de 00 à 59.
b) Construire un compteur de 01 à 12 pour les heures de cette montre.
c) L'horloge de cadence étant de 1 Hz, montrer les connexions entre les 3 compteurs : des secondes, des minutes et des heures.

5-19 Un additionneur série, synchronisé par une horloge, a deux entrées a et b et une sortie s. Chaque entrée reçoit successivement les bits d'un nombre et la sortie délivre successivement les bits de la somme des deux nombres. Réaliser cet additionneur en tant que machine de Moore à bascules D.

5-20 Utilisant un registre à 4 bits et une porte XOR,

a) construire un circuit synchrone dont l'état suivant x' est lié à l'état actuel x par les relations :
$$x_3' = x_1 \oplus x_0, \quad x_2' = x_3, \quad x_1' = x_2, \quad x_0' = x_1 . \ (1)$$
b) La suite des valeurs signées de $(x_3x_2x_1x_0)_{c2}$ constitue ce qu'on appelle un *signal pseudo-aléatoire*. Écrire et représenter cette suite partant de la valeur +3.
c) Soit x' = Tx la forme matricielle des relations (1) où T est la transformation qui associe à un vecteur x appartenant à l'espace $\mathcal{B}^4$ muni des opérations logiques • et ⊕ le vecteur x'. Calculer T^2, T^3, T^5 et T^{15}. Pour x = (0 0 1 1), calculer T^2x, T^3x, T^5x et $T^{15}x$ et vérifier ces calculs en se référant à b).

5-21 Construire une mémoire RAM de 128Mx32 bits utilisant 4 RAM de 32Mx8 bits, 2 RAM de 32Mx16 bits et 4 RAM de 16Mx32 bits et un circuit de décodage. Chaque RAM comporte les 2 broches $\overline{CS}$ (chip select), $R/\overline{W}$ (read/write) et OE (output enable).

5-22 Une lampe L s'allume ou reste allumée si aux 3 derniers fronts d'activation de l'horloge les valeurs d'un signal u sont alternées (010 ou 101). Construire pour ce système une machine de Mealy et une machine de Moore en n'utilisant qu'un PROM et un registre. Comparer avec les circuits de l'exemple 4-6.

5-23 Montrer que

$$(a_8 \dots a_0)_{BCD} = (a_8 \dots a_0)_2 - 6(a_8 \dots a_4)_2$$

et construire un convertisseur BCD → binaire à 8 bits utilisant un soustracteur et deux registres 8 bits.