ALGORITMOS Y PROCESADORES ARITMÉTICOS

1.1 - INTRODUCCIÓN

- Procesador aritmético
- Características
- Niveles de descripción funcional
    - Nivel abstracto
    - Nivel de algoritmo aritmético
    - Nivel de implementación

1.2 - SISTEMA DE REPRESENTACIÓN DE NÚMEROS

- Conjunto de valores de los dígitos
- Regla de interpretación
- Rango
- Clasificación
    - No redundantes
    - Redundantes
    - Ponderados
        - Vector de pesos
        - Vector de base
        - Base mixta
        - Base fija
    -No ponderados
- Sistema de numeración convencional en base $\Re$

1.3 - REPRESENTACIÓN Y SUMA DE NATURALES

- Introducción

1.3.2 - Algoritmo aritmético

1.3.3 - Algoritmo de suma a nivel de representación

- Full adder
- Half adder

1.3.4 - Tipos de sumadores

- Secuencial
- Paralelo
- 2 niveles de puertas

1.3.5 - Aceleración de la suma (CLA)

- CLA: Carry Look Ahead

1.4 - REPRESENTACIÓN, SUMA Y RESTA DE ENTEROS

1.4.1 - Introducción

- Representaciones

1.4.2 - Teoría de los sistemas complementados

- Representación gráfica
- Complemento a la base
- Complemento a la base disminuida
- Mapeo directo

1.4.3 - Suma en sistemas complementados

- Suma en C'2
- Overflow en C'2
- Suma en C'1
- Overflow en C'1

1.4.4 - Cambio de signos en sistemas complementados

- Cambio de signo en C'1
- Cambio de signo en C'2
- Overflow en cambio de signo

1.4.5 - Resta de enteros en sistemas complementados

- Cambio de signo y suma en C'1

- Cambio de signo y suma en C'2
- Overflow
- Implementación de un sumador/restador

1.4.6 - Extensión de rango

- N
- Z

1.4.7 - Signo y magnitud

- Mapeo directo
- Algoritmo de suma en S. y M.
  - Implementación

1.4.8 - Representación de enteros en exceso (o polarizados)

- Suma en exceso 2^{n-1} -1
- Cambio de signo
- Resta en exceso
- Sumador/Restador en exceso 2^{n-1} -1

1.5 - DESPLAZAMIENTOS ARITMÉTICOS

- N
- Z Sistemas complementados, base 2.
  - C '2
  - C '1

1.6 - ALGORITMOS ARITMÉTICOS PARA LA MULTIPLICACIÓN DE N

1.6.1 - Multiplicación Secuencial de naturales

- Algoritmo aritmético
- Recurrencias
- Implementación secuencial del algoritmo
  - Base 2
  - General

1.6.2 - Aceleración de la multiplicación secuencial para naturales

- Carry Save Adder (CSA)
- Carry Save Multiplication
- Exploración simultánea de varios dígitos sin solapamiento

1.6.3 - Algoritmos paralelos para multiplicar naturales

- Replicar el CPA en el espacio
- Replicar el CSA en el espacio
- Multiplicación con árboles de CPA'S.

1.7 - ALGORITMOS ARITMÉTICOS PARA MULTIPLICAR ENTEROS

1.7.1 - Multiplicación secuencial de enteros en C'2

- Implementación secuencial con CPA
- Implementación secuencial con CSA

1.7.2 - Algoritmos paralelos para multiplicar enteros (arrays) C'2

1.8 - REPRESENTACIÓN, SUMA, RESTA: MULTIPLICACIÓN DE REALES

- Punto fijo
- Punto flotante
- Mantisa normalizada con bit escondido

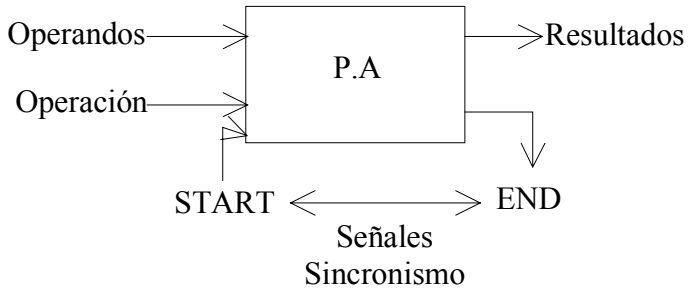
1.8.1 - Representación punto flotante IEEE 754-1985

- Características
- Representación en precisión simple
- Rango, precisión, error, overflow
- Suma y Resta en punto flotante

TEMA 1 - ALGORITMOS Y PROCESADORES ARITMÉTICOS

1.1 - INTRODUCCIÓN

- Procesadores Aritméticos



- Las señales de sincronismo (Start, End) indican al PA cuando las operaciones y el código de las operaciones son válidas en la entrada y a otras partes del sistema cuando los resultados están preparados.

Características:

- Representación de los operandos
- Rango sobre el que puede trabajar el P.A. (finito)
  (Aparece debido a la representación digital)
- Operaciones que puede hacer (1 o más de una)
- Velocidad
- Coste
- Área del Chip

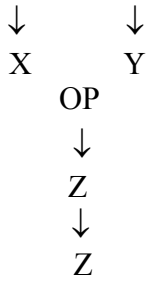
Niveles de descripción funcional

1 - Nivel abstracto: funciones matemáticas, propiedades --> NÚMEROS

2 - Nivel de algoritmo aritmético: los números son representados por VECTORES DE DÍGITOS y las operaciones son descritas por algoritmos que operan con este vector de dígitos.

$$x \in \mathrm{Z} \rightarrow X \in \mathrm{Z}^n \qquad \left(x_{n-1} x_{n-2}, \ldots x_1, x_0\right)$$

x y



f(X) = x

Z = x OP y

3 - Nivel de implementación: El vector de dígitos se codifica en un VECTOR DE BITS ya que la máquina trabaja en binario. Este nivel es el hardware que implementa el nivel anterior.

$$Z^n \rightarrow B^k$$

1.2 - SISTEMAS DE REPRESENTACIÓN DE NÚMEROS

- Los elementos que forman un sistema de representación son:

  - Conjunto de valores de los dígitos

    llamamos D_i al conjunto de valores de X_i

    Ejem: $D_i = \{0,1,2,...,9\}$ para el sistema decimal convencional

  - Regla de interpelación

    Busca una función que a partir del vector de dígitos retorne un valor.

    f(X) = x Ejem: Base 10 $x = \sum_{i=0}^{n-1} X_i 10^i$

- Rango: Conjunto de valores que puede tomar un número en nuestra representación. El rango se obtiene del conjunto de valores y de la regla. El número máximo de vectores de dígitos viene dado por:

  $K = \prod_{i=0}^{n-1} |D_i|$ Siendo: $|D_i|$: nº valores posibles

  n: nº de dígitos

- Clasificación de los sistemas:

  - No Redundantes: Todo vector de dígitos representa un número diferente.

$$X \neq Y \Rightarrow x \neq y$$

- Redundantes: Hay números que son representados por más de un vector de dígitos.

  ventajas --> incrementan la velocidad de ejecución.
  desventajas ---> disminuye el rango.

- Ponderados: regla de interpretación: $x = \sum_{i=0}^{n-1} X_i W_i$

  Donde: $W = (W_{n-1}, W_{n-2}, \ldots, W_1, W_0) \rightarrow$ VECTOR DE PESOS

  El vector de pesos va asociado con el VECTOR DE BASE:

  $$R = (R_{n-1}, \ldots, R_1, R_0)$$

  Existen 2 tipos de ponderados:

  1 - <u>Base mixta</u>

  $$W_0 = 1 \qquad W_i = W_{i-1} . R_{i-1} \qquad (1 \leq i \leq n\text{-}1)$$

  Ejem: sistema horario: representación del tiempo en términos de horas, minutos y segundos en un periodo de 24 horas:

  R = (24,60,60) --> vector de base
  W = (3600,60,1) --> vector de pesos

  2 - <u>Base fija:</u> Todos los elementos del vector de base tienen el mismo valor, r.

  R = (r,r,...,r)

  $$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\} x = \sum_{i=0}^{n-1} X_i . r^i$$

  $$W = (r^{n-1}, r^{n-2}, \ldots, r, 1)$$

- No ponderados: Sistema de numeración romano.

1.3 - REPRESENTACIÓN Y SUMA DE NATURALES

- $N = \{0,1,\ldots\}$

El sistema que utilizaremos será el sistema convencional en base r.

$X = X_{n-1},\ldots,X_1,X_0$

$$x = \sum_{i=0}^{n-1} X_i . r^i$$

$x_1 \in \{0,\ldots,r-1\}$

El rango es: $0 \le x \le r^n - 1$

SUMA: S = X + Y

$$X \equiv X_{n-1}X_{n-2}\ldots.X_1X_0 \qquad 0 \le x \le r^n - 1$$
$$Y \equiv Y_{n-1}Y_{n-2}\ldots..Y_1Y_0 \qquad 0 \le y \le r^n - 1$$
$$S \equiv S_nS_{n-1}S_{n-2}\ldots.S_1S_0 \qquad 0 \le s \le 2.r^n - 2$$

$\downarrow$ necesitamos n+1 dígitos

$0 \le s \le r^{n+1} - 1$

A nivel de dígitos:

$$s = x + y = \sum_{i=0}^{n-1} x_i . r^i + \sum_{y=0}^{n-1} y_i . r^i \qquad \sum_{i=0}^{n} S_i r^i = \sum_{i=0}^{n-1} (X_i + Y_i) r^i$$

Pero nosotros sólo queremos utilizar n dígitos. Podríamos considerar:

$S_i = X_i + Y_i$, i = 0,....,n-1

Sn = 0

} Pero no es válido, aparecen dígitos fuera de rango.

Vamos a ver como podemos hacer la suma sólo para *n* dígitos pero detectando el dígito Sn:

$$x = X_{n-1}.r^{n-1} + \ldots\ldots + X_{i+1}.r^{i+1} + X_i r^i + \ldots\ldots + X_0$$

$$y = Y_{n-1}.r^{n-1} + \ldots\ldots + Y_{i+1}.r^{i+1} + Y_i.r^i + \ldots\ldots + Y_0$$

$$S = \ldots\ldots\ldots\ldots + (X_{i+1} + Y_{i+1})r^{i+1} + (X_i + Y_i)r^i + \ldots + (X_0 + Y_0)$$

$$\underbrace{+1}_{X_{i+1} + Y_{i+1} + 1} \qquad \underbrace{-r}_{X_i + Y_i - r} \qquad \text{para } X_i + Y_i \geq r$$

Esta operación no altera el resultado ya que quitar la base a un dígito y sumar un carry al siguiente es no hacer nada

1.3.2 - Algoritmo Aritmético

$C_0 = 0$

DO i = 0, n-1

If $X_i + Y_i + C_i \geq r$

$S_i = X_i + Y_i + C_i - r$ $\qquad$ $C_n = \left| \frac{x + y}{r^n} \right|$

$C_{i+1} = 1$ $\qquad$ Solo el cociente

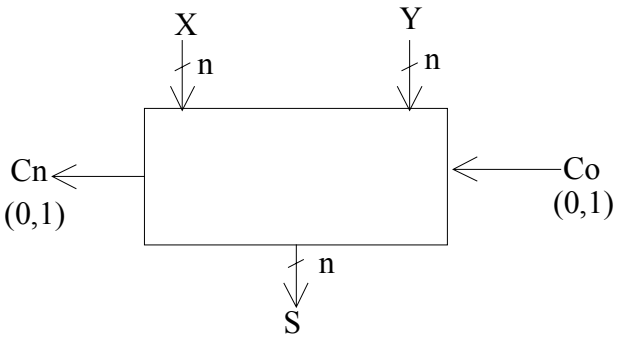
Else $\qquad$ Esto comprueba si la suma se

$S_i = X_i + Y_i + C_i$ $\qquad$ puede representar con n dígitos

$C_{i+1} = 0$ $\qquad$ o no

END DO

Sn = Cn



Cn siempre será 0 o 1 ---> Sn también

Cn = 0 --> resultado correcto representado con n dígitos.

Cn = 1 --> resultado incorrecto representado con n dígitos.

1.3.3 - Algoritmo de suma a nivel de representación

$D_i \in \{0,1\}$

$\left.\begin{array}{l}\\\\\end{array}\right\}$ Los dígitos son representados ahora como bits.

r = 2 La suma la encontraremos con operaciones lógicas.

Particularización del algoritmo:

$C_0 = 0$

DO i = 0, n-1

If $X_i + Y_i + C_i > r$

$S_i = 1$

$C_{i+1} = 1$

Else If $X_i + Y_i + C_i = r$

$S_i = 0$

$C_{i+1} = 1$

Else

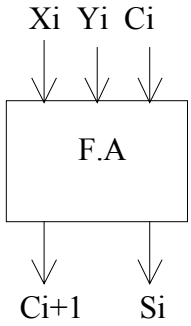
$S_i = 1$

$C_{i+1} = 0$

END DO

Xi	Yi	Ci	Si	Ci+1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder:



Otra particularización del algoritmo:

$C_0 = 0$

DO i = 0, n-1

If $X_i + Y_i + C_i \geq r$

$S_i = X_i + Y_i + C_i - r$

$C_{i+1} = 1$

Else

Si = $X_i + Y_i + C_i$

$C_{i+1} = 0$

END DO

Sn = Cn

Vamos a implementarlo con el mínimo número de puertas:

Ci \ XiYi	00	01	11	10
0	0	1	0	1
1	1	0	1	0

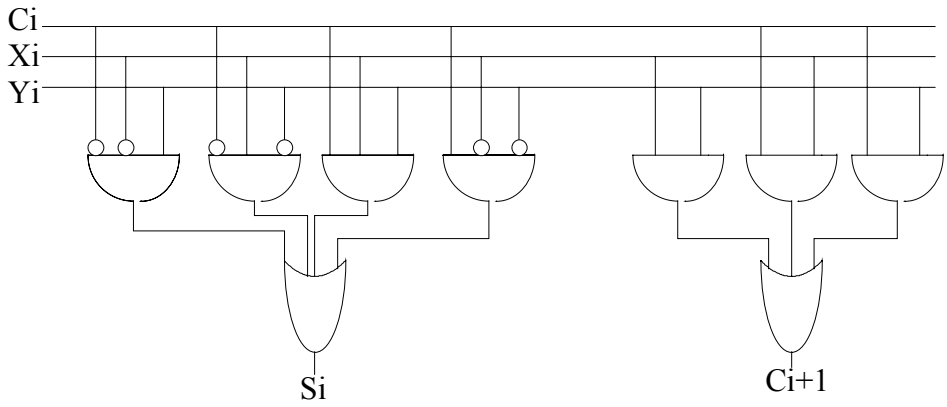
Si

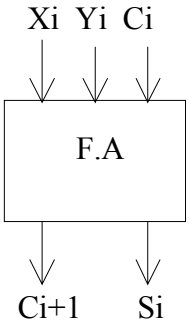
Ci \ XiYi	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Ci+1

Si = $\overline{C}i.\overline{X}i.Yi + \overline{C}i.Xi.\overline{Y}i + Ci.\overline{X}i.\overline{Y}i + Ci.Xi.Yi$

Ci+1 = XiYi + CiXi + CiYi





Tiempo de respuesta = 2 niveles de puertas
(no tenemos en cta
los inversores).

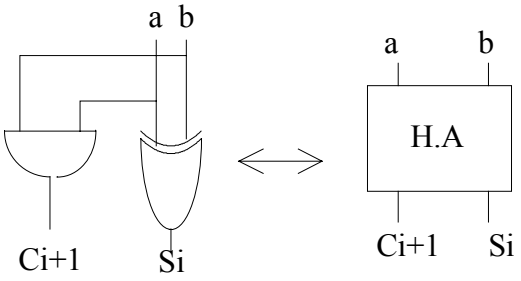
T = 2 Z

Vamos a realizar el mismo circuito pero utilizando puertas HALF ADDER:

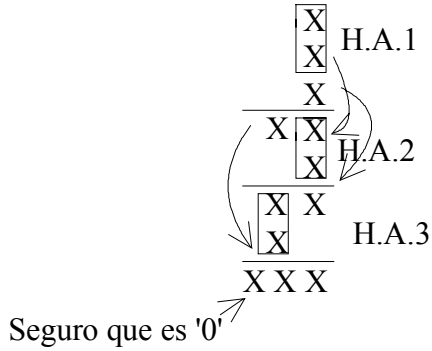
a	b	Ci+1	Si
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$Si = a \oplus b$

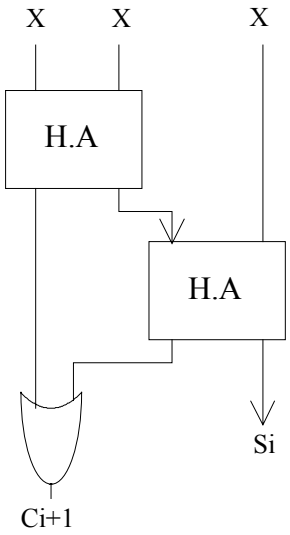
$Ci+1 = a \cdot b$

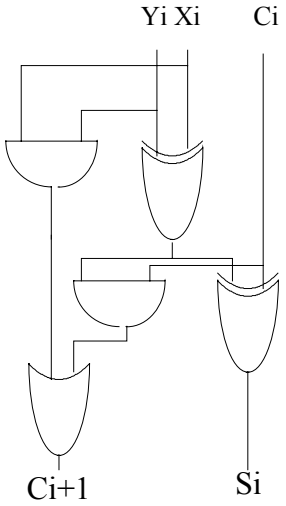


Si hacemos la suma a partir de H.A:



Nunca podrá ser 1+1 --> podemos poner una OR.





T = 3Z

$Si = Xi \oplus Yi \oplus Ci$

$Ci+1 = Xi.Yi + Ci\ (Xi \oplus Yi)$

Conclusiones:

- Con F.A más puertas que con H.A

---> Compromiso entre vel. y coste.

- Con F.A más rápido que con H.A

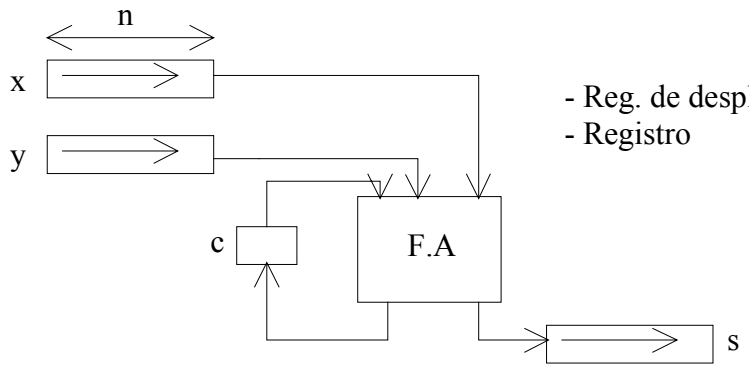
1.3.4 - Tipos de Sumadores

1- Secuenciales

2- Paralelo
- CPA
- CLA
- CSA

3- 2 niveles de puertas: es el más rápido pero con muchas puertas y cada puerta con muchas entradas.

1 - Secuencial



- Reg. de desplazamiento
- Registro

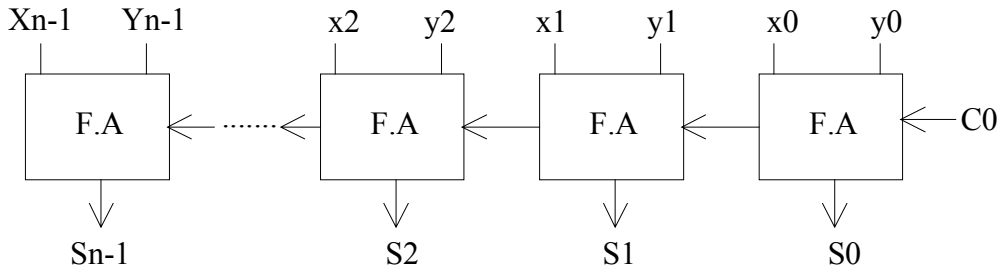
$$T = n\left(T_{FA} + T_{BI}\right)$$

T_{FA} = Depende del diseño del F.A

T_{BI} = Tiempo del biestable

2 - Paralelo

CPA (Carry Propagation Adder)



T = n 2Z Mejor que el anterior

1.3.5 - Aceleración de la suma

El problema es el carry. Vamos a analizarlo:

$$Ci+1 = Xi.Yi + Ci.Xi + Ci.Yi = Xi.Yi + Ci(Xi \oplus Yi)$$

------ ---------

Gi Pi

↓ ↓

Generador de Carry

Propagador de Carry

Por tanto, tendremos un carry en la salida si:

Gi = 1 independientemente de Pi ---> genera un carry

Ci = 1 y Pi =1 ---> se propaga el carry anterior

Ejem:

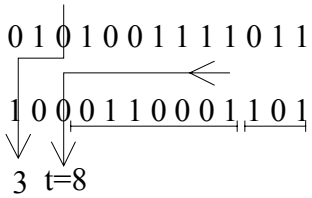
0 1 1 0 1 1 0 1 0 1 1 1 0 0

0 1 0 0 1 0 1 1 0 0 1 0 1 0

Una primera idea:

Podríamos empezar a sumar por la combinación 1/1 que sabemos que seguro genera acarreo o por la 0/0 que el acarreo siempre es cero --> PARALELISMO.

El tiempo de suma viene dado por la cadena más larga:



Este sistema no acaba de ser bueno ya que tendríamos que detectar las combinaciones 0/0 y 1/1 --> añadir puertas --> más lento.

CLA (Carry Look Ahead) Anticipación de Carry

Pretendemos acelerar los carry. Vamos a analizar la expresión del carry:

$$Ci+1 = Gi + Ci.Pi$$

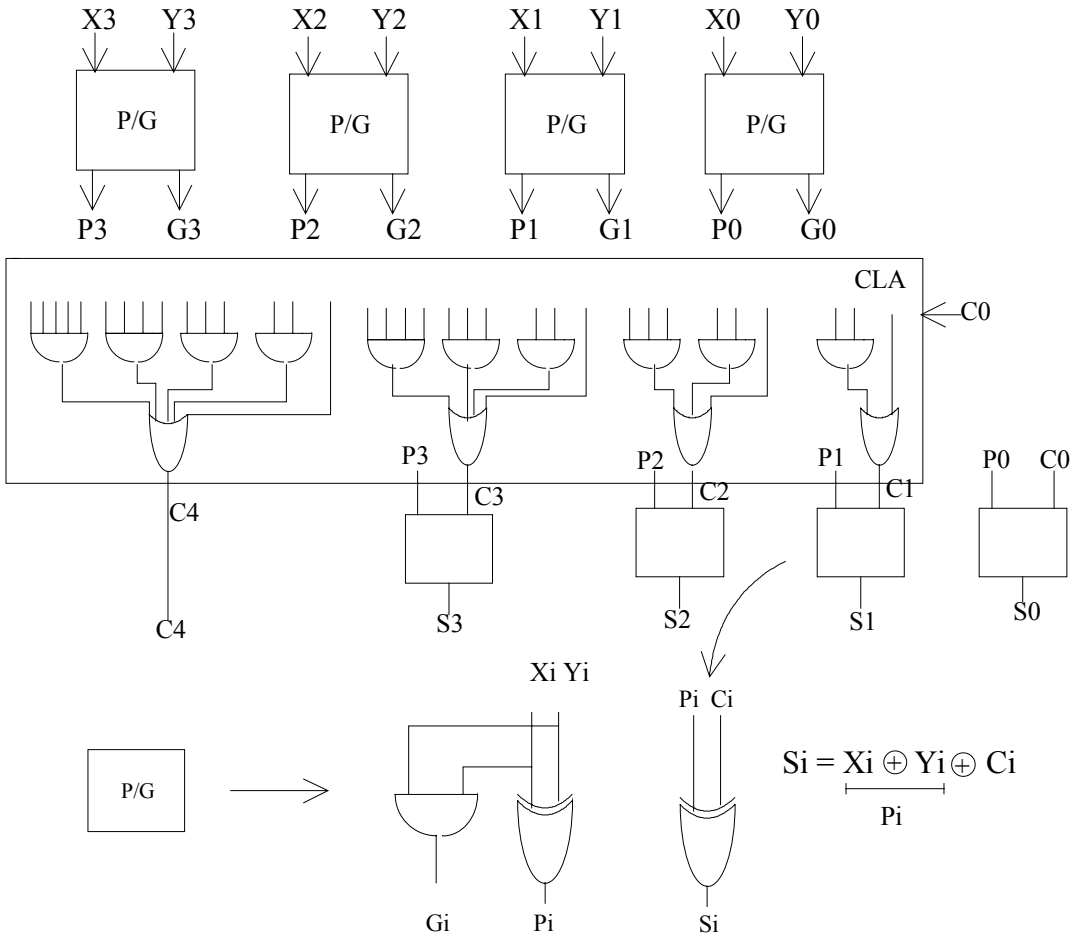
$$C_1 = G_0 + C_0 P_0$$

$$C_2 = G_1 + C_1 P_1 = G_1 + G_0 P_1 + C_0 P_0 P_1$$

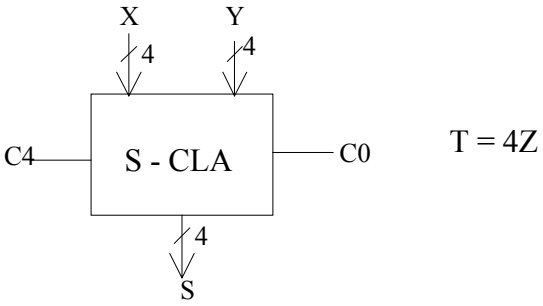
$$C_3 = G_2 + C_2 P_2 = G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2$$

Observar que los carry no dependen del acarreo de la etapa anterior sino sólo de Pi, Gi, C_0.

Si lo implementamos, por ejemplo para 4 bits:

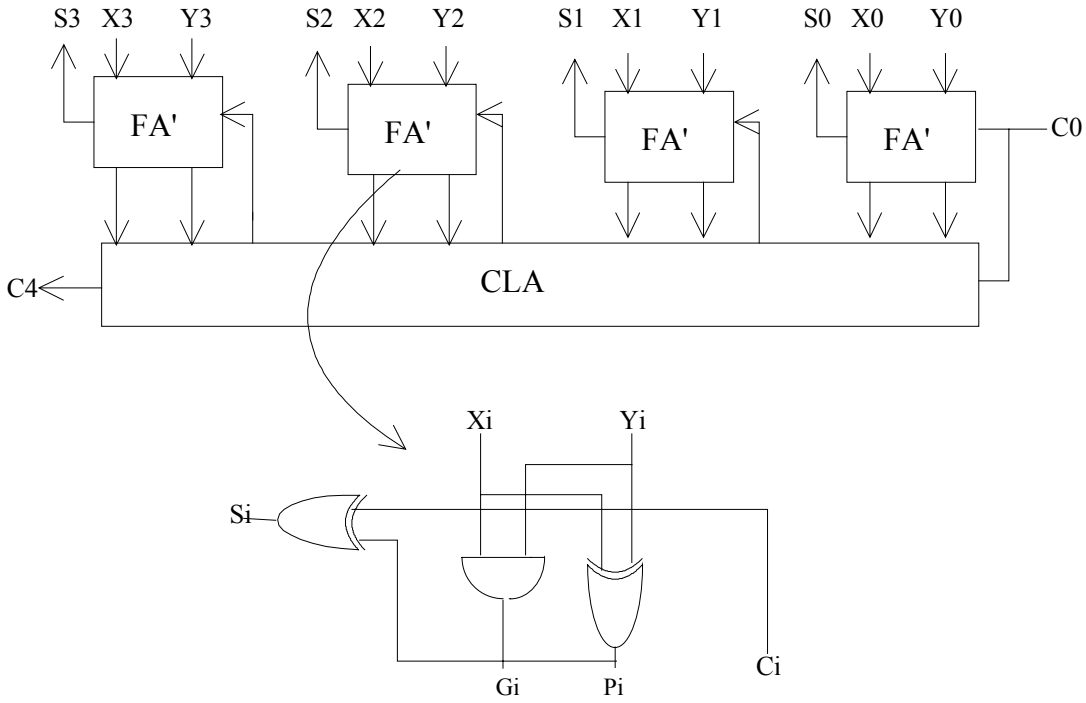


Tiempo de respuesta: $T = 4Z$ Independiente de n (aconsejable $n \leq 4$)



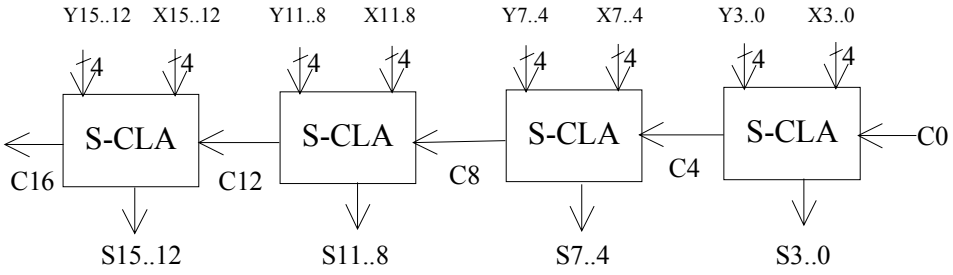
$T = 4Z$

Normalmente se dibuja:



Ejemplo: Realizar un sumador de palabras de 16 bits con CLA.

- Una opción:



Volvemos a tener problemas con la propagación del carry. Este sistema no se utiliza.

- Otra opción: Utilizar un CLA para los carry´s C4, C8, C12, C16
Si analizamos C4:

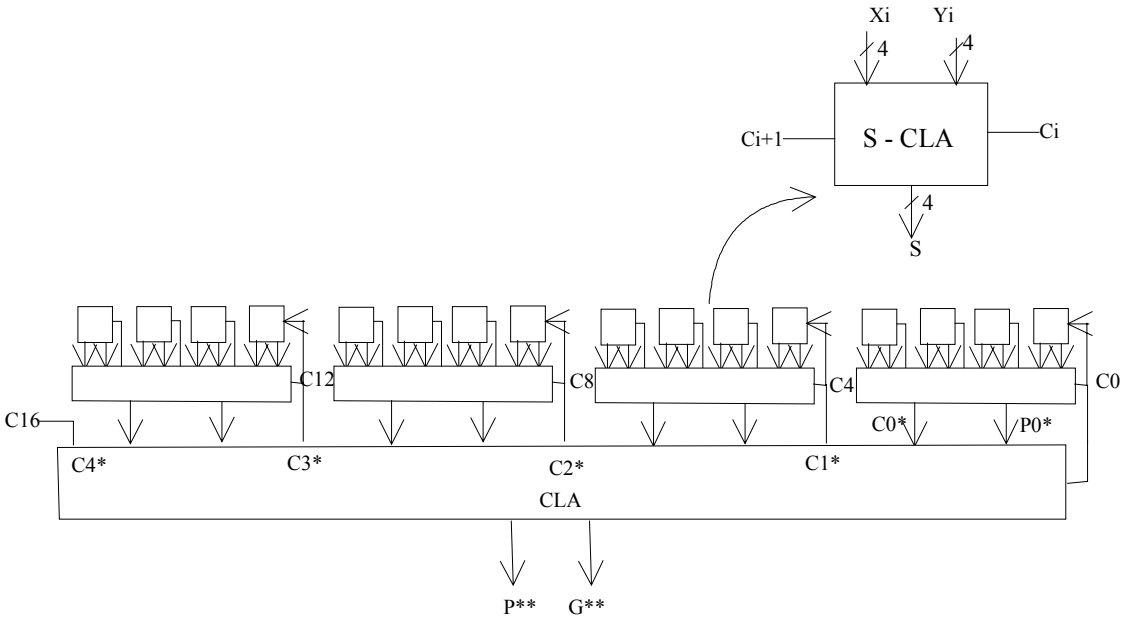
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

G_0^* P_0^*

generador de grupo propagador de grupo

$$C_4 = G_0^{'} + P_0^{'}C_0$$

Así quedará:



En general: El número de niveles de CLA es: $\log_r n$

r: número de entradas del sumador (normalmente 4)
n: número de bits que se quieren sumar

El tiempo de respuesta será: $T_n = 4Z\left[\log_4 n\right]$ Nota: $\log_a N = \dfrac{\log_b N}{\log_b a}$

1.4 - REPRESENTACIÓN, SUMA Y RESTA DE ENTEROS

1.4.1 - Introducción

$$\left.\begin{array}{l} Z \rightarrow N^{n} \\ x \rightarrow X(vector\_de\_digitos) \end{array}\right\} \text{ mapeo directo}$$

Representamos un valor entero por un vector de dígitos de números naturales.

Otra forma:

$$\left.\begin{array}{l} Z \xrightarrow{f'} N \xrightarrow{f^{2}} N \\ x \longrightarrow x_{e} \longrightarrow X \end{array}\right\} \text{ mapeo directo}$$

x: valor implícito
x_e: valor explícito

$$x_e = \sum_{i=0}^{n-1} X_i r^i$$

El número entero con signo, 'x', se representa con un valor entero positivo, x_e, el cual se representará por un vector de dígitos, X.

REPRESENTACIONES (de un número entero como un natural)

- S y M (Signo y magnitud)
- Sistemas complementados
- Exceso 2^m (polarizado)
- r < 0 (base negativa)

1.4.2 - Teoría de los sistemas complementados

En los sistemas complementados no hay separación entre la representación del signo y de la magnitud, lo que hacemos es representar el número entero con un número natural x_e.

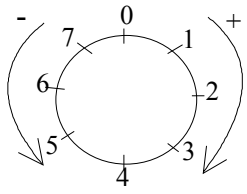
La función que nos da x_e

x_e = **x mod c** siendo: c: constante de complementación

La función <u>mod</u> viene definida como:

$$x_e \begin{cases} x \rightarrow x \geq 0 \\ c - |x| \rightarrow \leq 0 \end{cases}$$

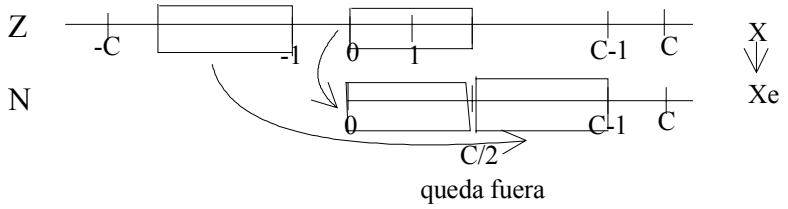
Ejem: c = 8 , x = 2 , x = -2



x_e = 2 mod 8 = 2
x_e = -2 mod 8 = 6

Es necesario que no se produzcan solapamientos, por tanto: $|x|max < \frac{c}{2}$

<u>REPRESENTACIÓN GRÁFICA</u>



La función inversa será:

$$x = \begin{cases} x_e , & \text{si } x_e < c/2 \\ x_e - c & \text{si } x_e > c/2 \end{cases}$$

Consideración:

Si ' c ' es par consideraremos ' c / 2 ' en los negativos:

- c par $\rightarrow \frac{c}{2} \in \mathrm{N} \rightarrow rango: \left[-\frac{c}{2} ..... \frac{c}{2} - 1 \right]$

- c impar $\rightarrow \frac{c}{2} \notin \mathrm{N} \rightarrow rango: \left[\frac{-(c-1)}{2} ..... \frac{c-1}{2} \right]$

De esta forma conseguimos que haya el mismo número de números negativos que positivos.

Dado n dígitos el rango válido es: $0 \le x_e \le r^n - 1$

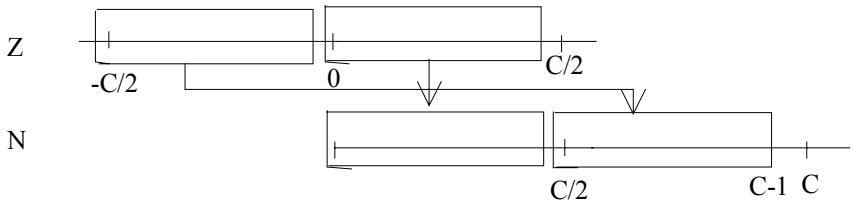
Según sea 'c' tenemos:

- c = $r^n \rightarrow$ complemento a la base. Si r = 2 $\rightarrow$ c ' 2
- c = $r^n - 1 \rightarrow$ complemento a la base disminuida. Si r = 2 $\rightarrow$ c ' 1

<u>COMPLEMENTO A LA BASE</u>

rango: $0 \le x_e \le r^n - 1$ c = $r^n \rightarrow$ par

rango: $-\frac{c}{2} \le x \le \frac{c}{2} - 1 \rightarrow$ Desequilibrio



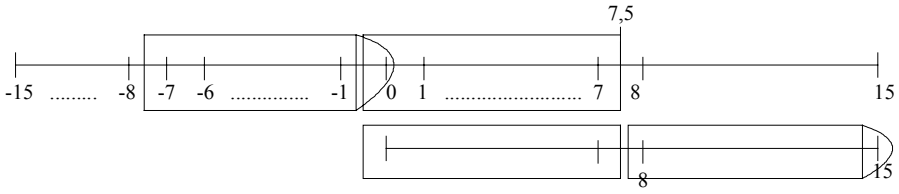
<u>COMPLEMENTO A LA BASE DISMINUIDA</u>

c = $r^n - 1$ $\rightarrow$ Impar

$\rangle$ Si r = 2 $\rightarrow$ Complemento a 1

rango: $0 \le x_e \le r^n - 1$

* Ejemplo: r = 2 , n = 4 $\rightarrow$ c = $2^4 - 1 = 15 \rightarrow \frac{c}{2} = 7{,}5$



El 15 se mapea sobre el '0'

rango: $\left[-\left(\frac{r^n}{2}-1\right)....\frac{r^n}{2}-1\right]$

rango (n = 2) $\left[-\left(2^{n-1}-1\right)-\left(2^{n-1}-1\right)\right]$

MAPEO DIRECTO

(Solo es válido cuando la base es 2).

Permite encontrar un camino directo sin tener que pasar por x_e

C '2

$$x = \begin{cases} x_e & \text{si} \quad x_e \le 2^{n-1} \quad \left(x_{n-1}=0\right) \\ x_e - 2^n & \text{si} \quad x_e \ge 2^{n-1} \quad \left(x_{n-1}=1\right) \end{cases} \rightarrow$$

$$x = \begin{cases} \sum_{i=0}^{n-1} X_i 2^i = \sum_{i=0}^{n-2} X_i 2^i \\ \sum_{i=0}^{n-1} X_i 2^i - 2^n = 2^{n-1} + \left(\sum_{i=0}^{n-2} X_i 2^i\right) - 2^n = -2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i \end{cases}$$

generalizando:

$$x = -2^{n-1} X_{n-1} + \sum_{i=0}^{n-2} X_i 2^i$$

C '1

$$x = \begin{cases} x_e, \ldots . si \ldots . . x_e < 2^{n-1} \\ x_e - (2^n - 1), \ldots . si \ldots . . , x_e \geq 2^{n-1} \end{cases} \rightarrow$$

$$x = \begin{cases} \sum_{i=0}^{n-1} X_i 2^i = \sum_{i=0}^{n-2} X_i 2^i \\ \sum_{i=0}^{n-1} X_i 2^i - (2^n - 1) = -2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i + X_{n-1} \end{cases}$$

$$x = -2^{n-1} X_{n-1} + \sum_{i=0}^{n-2} X_i 2^i + X_{n-1}$$

1.4.3 - Suma en sistemas complementados

$S = x + y$ $\qquad$ $Z \rightarrow N \rightarrow N^n$

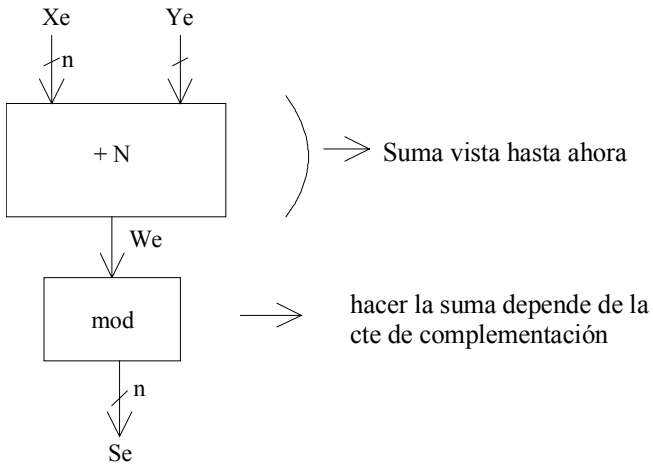
$x \qquad x_e \quad X$

Interesa que la suma quede en valores explícitos

$S_e = (x_e + y_e) \bmod c$

$S_e = f(x_e, y_e)$

En un principio hacer la suma representa:



SUMA EN C'2

$S_e = W_e \bmod c$ Para garantizar que la suma sea correcta (∄ problemas de overflow) tenemos que añadir un bit.

$W_e = X_e + Y_e$

↓

(Wn , Wn-1, Wn-2 .....W_0) → n+1 bits

$$W_e \bmod 2^n \begin{cases} W_e .....si..... W_e < 2^n \\ W_e - 2^n .....si..... W_e \geq 2^n \end{cases} \rightarrow$$

$W_e \rightarrow$ 1 Wn-1 Wn-2 Wn-3.....W1 W_0 para $W_e \geq 2^n$

\- 1 0 0 0 0 0

0 Wn-1 Wn-2 Wn-3.....W1 W_0

↳ Es equivalente a despreciar el bit de mayor peso

El carry no es el bit de mayor peso ya que lo despreciamos.

Ejercicios: 21,9,6

Ahora necesitamos detectar cuando se produce overflow:

OVERFLOW EN C'2

rango: $\left[-2^{n-1} ....2^{n-1} - 1\right]$ con n bits

X, Y ,S $\in \left[-2^{n-1} ....2^{n-1} - 1\right]$ → tendremos problemas en la suma

$$0 \leq X_e \leq 2^n - 1$$

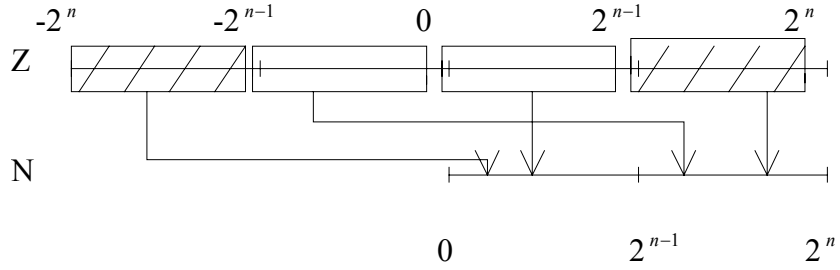
CASOS:

1 - Suma de X , Y con signos diferentes → S es representable

2 - Suma de X , Y con mismo signo → S no siempre representable

$$-2^n \leq X + Y \leq 2(2^{n-1} - 1)$$

$$S_e = S \bmod 2^n = \begin{cases} X+Y \ldots si \ldots X+Y > 0 \\ 2^n + X + Y \ldots si \ldots X+Y < 0 \end{cases}$$

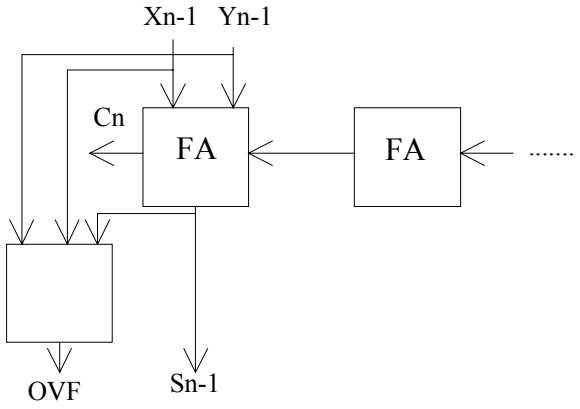


la expresión del overflow:

OVF = Xn-1.Yn-1. $\overline{S}$ **n-1 +** $\overline{X}$ **n-1.** $\overline{Y}$ **n-1.Sn-1**

El ovf existe cuando los números son '+' $\rightarrow$ Xn-1 = Yn-1 = 0 y el bit de signo de la suma es Sn-1 = 0 o los números son negativos $\rightarrow$ Xn-1 = Yn-1 = 1 y el bit de signo de la suma es Sn-1 = 0.

La implementación:



Para el caso concreto de complemento a 2:

OVF = Cn $\oplus$ **Cn-1** Existe OVF si los carry más significativos son diferentes.

SUMA EN C'1

Se = ($X_e + Y_e$) mod c Lo único que varia respecto al c'2 es

W_e

C = $2^n - 1$ la operación mod.

$$W_e \bmod (2^n - 1) \begin{cases} 1) W_e \ldots\ldots si \ldots\ldots\ldots\ldots\ldots 0 \le W_e \le 2^n - 1 \\ 2) 0 \ldots\ldots\ldots si \ldots\ldots\ldots\ldots\ldots W_e = 2^n - 1 \\ 3) W_e - (2^n - 1) \ldots si \ldots . 2^{n-1} < W_e < 2(2^n - 1) \\ 4) 0 \ldots\ldots\ldots si \ldots\ldots\ldots\ldots\ldots W_e = 2(2^n - 1) \end{cases}$$

rango: $0 \le W_e \le 2(2^n - 1)$

W_e = (Wn, Wn-1, ...W_1 W_0) → n+1 dígitos

1) (0,X,X....,X) ≠ (0,1,1....1)
} Wn = 0
2) (0,1,1,.....1)

3) (1,X,X,....X) ≠ (1,1,...,1,0)
} Wn = 1
4) (1,1.....,1,0)

Casos:

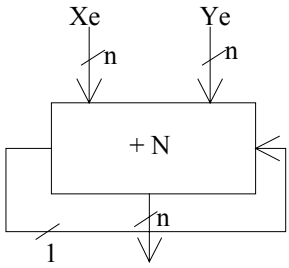
1 , 2 → (Wn-1 , Wn-2 ,.....,W_1 ,W_0) no podemos con los n-1 bits de menor peso

3 → $W_e - 2^n + 1$

1XXX....XX
10 0 0.....0 0 Despreciamos el bit de mayor
XXX....XX peso y sumamos 1
+1

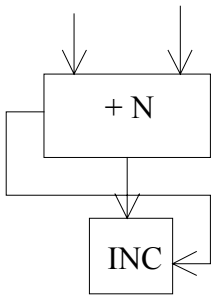
4 → 11.....10
1 Despreciamos el bit de mayor peso y sumamos 1
11.....11

El circuito que lo implementa:



Las combinaciones peligrosas son las que tienen bits que todas propagan el carry

Un sumador que funciona:



Es mejor que el anterior pero requiere más area.
El tiempo es el mismo.

Un resultado de suma no será nunca todo ‘0’ a no ser que sea forzado (sumando ‘0’ + ‘0’)

OVERFLOW EN C’1

- Igual que en C’2.

1.4.4 - Cambio de signo en sistemas complementados

$Z = -X \rightarrow Z_e = f(X_e)$ Donde Z y X son números enteros

$Z_e = Z \bmod c = (-X) \bmod C = C - (X \bmod C) = C - X_e$

Demo:

$$-X \bmod C = \begin{cases} -X, X < 0 \\ C - |X|, X \geq 0 \end{cases}$$

$\rightarrow$ Equivalente

$$C - X \bmod C = \begin{cases} C - X, X \geq 0 \\ C - (C - |X|), X < 0) \end{cases}$$

CAMBIO DE SIGNO EN C'1

$Z_e = C - X_e$

$C = r_{n-1}(2^n - 1)$

$$Z_e = \sum_{i=0}^{n-1} Z_i r^i$$

$$X_e = \sum_{i=0}^{n-1} X_i r^i$$

$X_i, Z_i \in [0..r-1]$

$$Z_e = (r^n - 1) - \sum_{i=0}^{n-1} X_i r^i = \sum_{i=0}^{n-1} (r-1) r^i - \sum_{i=0}^{n-1} X_i r^i = \sum_{i=0}^{n-1} (r - 1 - X_i) r^i$$

En concreto si $C = 2^n - 1$

$$Z_i = \left.\begin{cases} X_i = 0 \rightarrow Z_i = 1 \\ X_i = 1 \rightarrow Z_i = 0 \end{cases}\right\} \rightarrow Z_i = \overline{X_i} \rightarrow Z_e = \overline{X_e}$$

Complementando bit a bit

CAMBIO DE SIGNO EN C'2

$C = 2^n$

$Z_e = C - X_e = 2^n - 1 + 1 - X_e = \overline{X_e} + 1$

OVERFLOW EN CAMBIO DE SIGNO

- C'1:

Rango simétrico: $\left[-(2^{n-1}-1).....2^{n-1}-1\right] \rightarrow \nexists$ overflow

- C'2:

Rango asimétrico: $\left[-2^{n-1}.....2^{n-1}-1\right] \rightarrow$ Existe overflow para $100...0 \ (-2^{n-1})$

Ejem:

$n = 3 \quad x = -4 \quad \rightarrow \quad X: 1\ 0\ 0$

$\downarrow \qquad\qquad \overline{X}: 0\ 1\ 1$

$X_e = 4 \qquad\qquad +1$

$Z_e = 4 \rightarrow \quad 1\ 0\ 0$

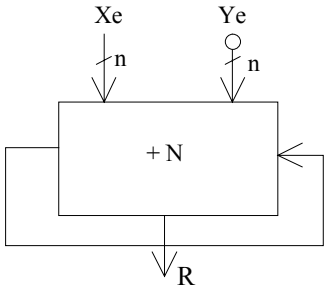
1.4.5 - Resta de enteros en sistemas complementados

- 2 opciones:

1.- Algoritmo directo

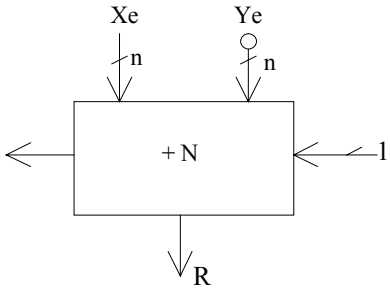
2.- Cambio de signo y suma $\rightarrow$ R = X - Y = X + (-Y)

CAMBIO DE SIGNO EN C'1



$\text{ADD}_{\text{N}}(X_e, \overline{Y}_e, Cout)$

CAMBIO DE SIGNO Y SUMA EN C'2



$\text{ADD}_{\text{N}}(X_e, \overline{Y}_e, 1)$

OVERFLOW

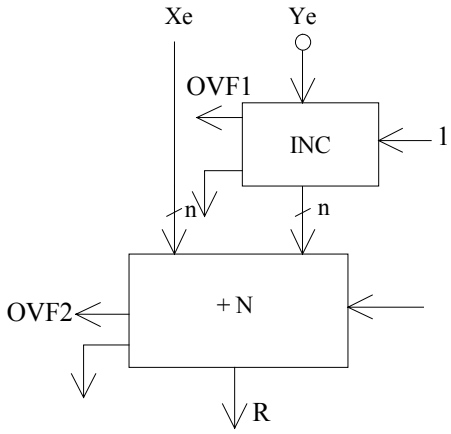
C'1: Igual que en la suma

C'2: Existen 2 posibles overflow:

- OVF1 $\rightarrow$ cambio de signo

- OVF2 $\rightarrow$ en la suma

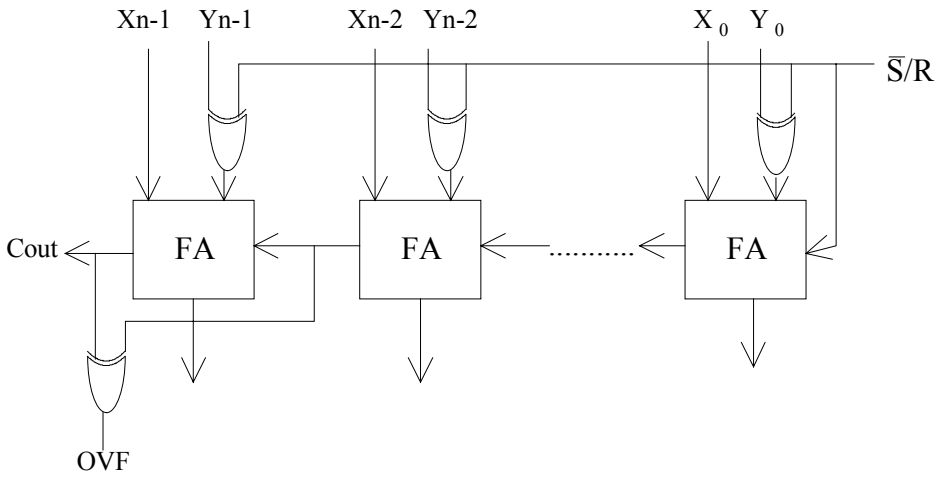
También podríamos hacer:



Esta suma es más eficiente que la anterior.

IMPLEMENTACIÓN DE UN SUMADOR/RESTADOR

C'2:



C'1: Pareado.

1.4.6 - Extensión de Rango

- Es necesario para hacer multiplicaciones.

$X \in \left[-2^{n-1} \ldots 2^{n-1} - 1\right]$ Queremos representar el valor de X de n bits con un vector de dígitos de m bits tal que:

$Y \in \left[-2^{m-1} \ldots 2^{m-1} - 1\right]$ $X = Y$ m > n

Es decir , queremos la representación de Y cuyo valor es igual al de X pero tiene 'm' bits (m > n).

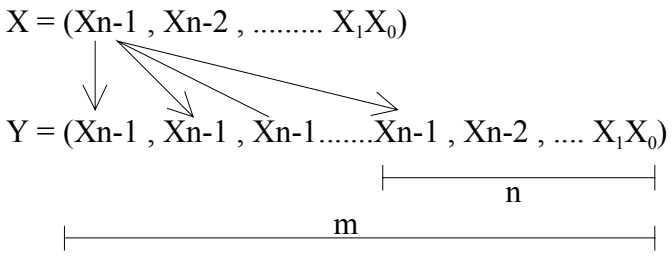
a) Para naturales N :

$$Y_i = \begin{cases} X_i, i = 0 \ldots . n-1 \\ 0, i = n \ldots . . m-1 \end{cases} \Leftrightarrow \text{añadimos '0' a la izquierda.}$$

b) Para enteros Z :

$$Y_i = \begin{cases} X_i, i = 0 \ldots . . n-1 \\ \begin{cases} 0, i = n \ldots . m-1 \rightarrow X \geq 0 \\ (r-1), i = n \ldots . m-1 \rightarrow X < 0 \end{cases} \end{cases}$$

Para el caso concreto de r = 2 lo que hacemos es replicar el signo:



Demostración para C'2:

$$X = \begin{cases} X_e, X_e < \dfrac{r^n}{2} \\ X_e - r^n, \dfrac{r^n}{2} \leq X_e < r^n \end{cases}$$ obtención de X a partir de X_e

$$Y = \begin{cases} Y_e, Y_e < \dfrac{r^m}{2} \\ Y_e - r^m, \dfrac{r^m}{2} \le Y_e < r^m \end{cases}$$

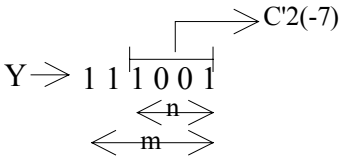
Si $X_e < \dfrac{r^n}{2} \rightarrow Y = X \equiv \text{N}$

Si $X_e \ge \dfrac{r^n}{2} \rightarrow Y_e - r^m = X_e - r^n \rightarrow Y_e = X_e + r^m - r^n$

Ejemplo:

r = 2 , n = 4 , m = 6 , X = -7 $\rightarrow X_e = 9 > \dfrac{2^4}{2}$

$Y_e = 9 + 2^6 - 2^4 = 9 + 64 - 16 = 57$



* Demostrarlo para C ‘1

1.4.7 - Signo y Magnitud

$$X = (X_s, X_m)$$

$$\mathrm{Z} \to \mathrm{N}^2 \to \mathrm{N}^n$$

$$X \to (X_s, X_m) \to \mathrm{X} = (\underbrace{X_{n-1}}_{X_s}, \underbrace{X_{n-2} \ldots X_1 X_0}_{X_m})$$

$|x| < r^{n-1} \to$ rango simétrico pero salen 2 '0'

$$\mathrm{Z} \to \mathrm{N} \to \mathrm{N}^n$$

$$X \to X_e \to \mathrm{X}$$

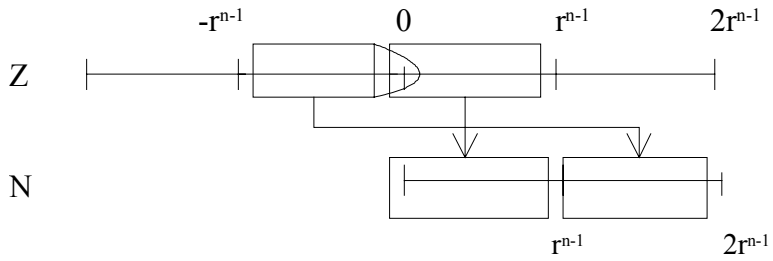
El valor explícito:

$$X_e = \begin{cases} X, X \geq 0 \\ r^{n-1} - X, X < 0 \end{cases}$$

Inversa:

$$X = \begin{cases} X_e, 0 \leq X_e < r^{n-1}, (X_{n-1} = 0) \\ r^{n-1} - X_e, r^{n-1} \leq X_e < 2r^{n-1}, (X_{n-1} = 1) \end{cases}$$

De forma gráfica:



MAPEO DIRECTO (Para r = 2)

$$X = \begin{cases} \sum_{i=0}^{n-2} \mathrm{X}_i 2^i \\ 2^{n-1} - \sum_{i=0}^{n-1} \mathrm{X}_i 2^i = 2^{n-1} - 2^{n-1} - \sum_{i=0}^{n-2} \mathrm{X}_i 2^i \end{cases} \Rightarrow$$

$$X = (1 - 2X_{n-1}) \sum_{i=0}^{n-2} \mathrm{X}_i 2^i$$

ALGORITMO DE SUMA EN S Y M

$S = (S_s, S_m)$

$X = (X_s, X_m)$

$Y = (Y_s, Y_m)$

```
if (X_s = Y_m) then
    S_m = (X_m + Y_m) mod 2^(n-1)
    OVF = Cn
else if (X_m ≥ Y_m) then
        S_m = X_m − Y_m
        S_s = X_s
        else
            S_m = Y_m − X_m
            S_s = Y_s
        end if
        OVF = 0
end if
```

Pero para implementarlo es necesario un hardware muy complejo.

Debemos optimizar el algoritmo. Intentaremos que la complejidad sea parecida a la de la suma en C'1 o C'2.

MEJORA:

if $(X_s = Y_s)$ then

$S_s = X_s$

$S_m = (X_m + Y_m) \bmod 2^{n-1}$

OVF = Cn

else

$S'_m = X_m - Y_m$

if $(S'_m \geq 0)$ then

$S_m = S'_m$

$S_s = X_s$

else

$S_m = -S'_m$

$S_s = Y_s$

end if

OVF = 0

end if

Nos ahorramos girar los operandos, haciendo un cambio de signo.

¿Que convenio de complementación utilizaremos C'1 o C'2?

- Tenemos que hacer una resta y un cambio de signo $\rightarrow$ es más sencillo en C'1.

Implementación

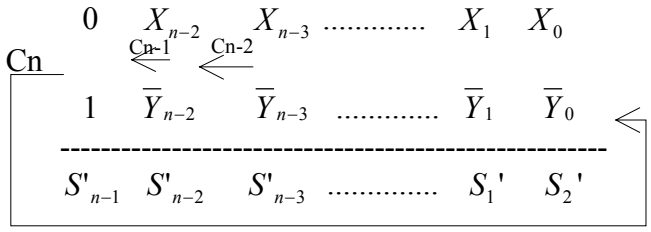
* Codificar X_m, Y_m en C'1

$0X_{n-2}, X_{n-3} \ldots . X_1, X_0 \rightarrow X_m$ en C`1

$0Y_{n-2}, Y_{n-3} \ldots . Y_1, Y_0 \rightarrow Y_m$ en C`1

* Operar con + , - en C'1 (n bits)

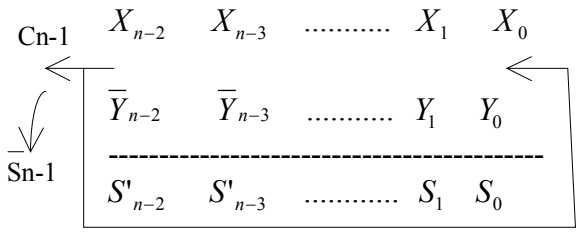
i) $(X_s \neq Y_s) \rightarrow X_m - Y_m$



Observar que: $C_n = C_{n-1}$ y $S'_{n-1} = \overline{C_{n-1}}$

Por tanto:

-

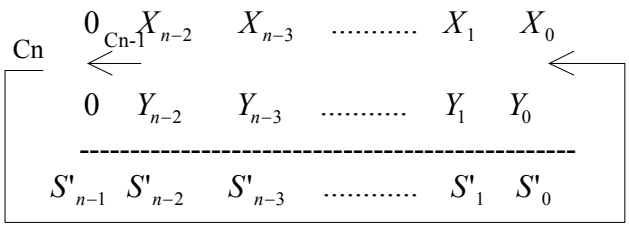


En este caso:

- No existe overflow

- Si $S'_{n-1} = 1 \rightarrow$ Magnitud negativa $\rightarrow$ hay que complementar ya que el resultado estará en C'1 y lo quieres en signo y módulo.

ii) $(X_s = Y_s) \rightarrow X_m + Y_m$



-¿Necesitamos el último FA?

Observar que: $C_n = 0$ y $S'_{n-1} = C_{n-1}$

Casos posibles:

- $C_{n-1} = 0$

- $C_{n-1} = 1 \rightarrow S'_{n-1} = 1 \rightarrow$ OVERFLOW

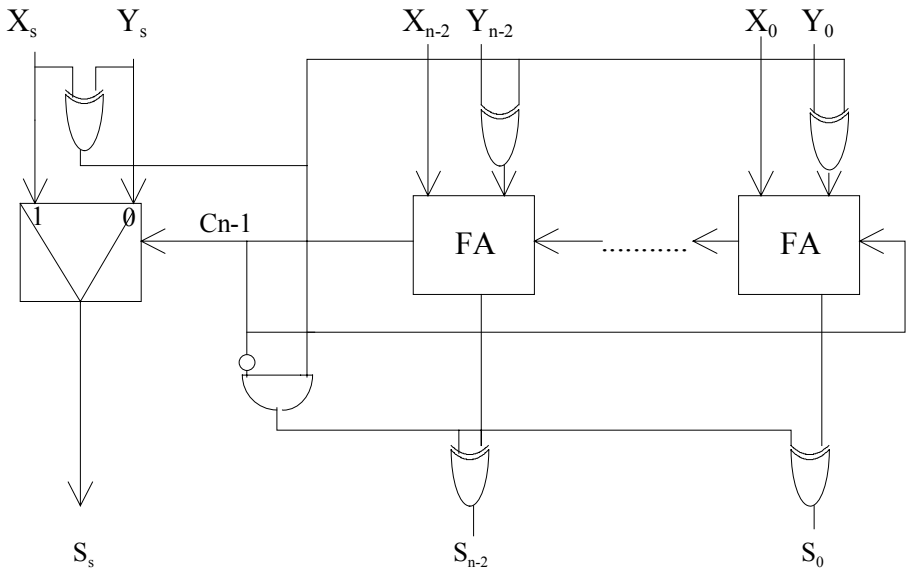
Estamos sumando 2 magnitudes $\oplus \rightarrow$ no puede dar un resultado negativo.

Por tanto:

$$\begin{array}{llllll} X_{n-2} & X_{n-3} & \ldots\ldots\ldots & X_1 & X_0 \\ Y_{n-2} & Y_{n-3} & \ldots\ldots\ldots & Y_1 & Y_0 \\ \hline S_{n-2} & S_{n-3} & \ldots\ldots\ldots & S'_1 & S'_0 \end{array}$$

OVF = $X_s Y_s C_{n-1} + \overline{X}_s \overline{Y}_s C_{n-1}$

El circuito será:



Faltaría el circuito de detección de overflow.

1.4.8 - Representación De Enteros En Exceso (o Polarizado)

$$\left.\begin{array}{l} Z \rightarrow N \rightarrow N^n \\ X \rightarrow X_e \rightarrow X \end{array}\right\} X_e = X + C$$

En función de 'C' tendremos diferentes excesos.

$$C = \begin{cases} 2^{n-1} - 1 \\ 2^{n-1} \end{cases}$$

Para $C = 2^{n-1} - 1$

$$X_e = X + 2^{n-1} - 1$$

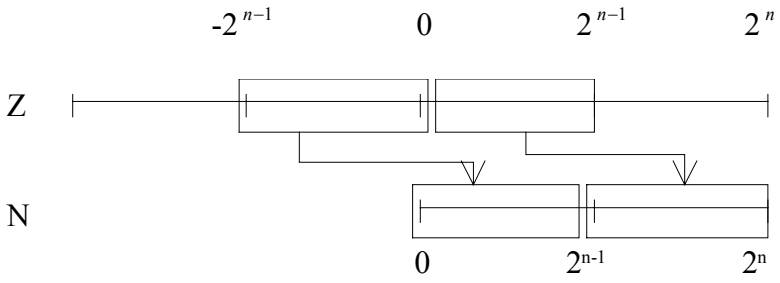
Como X_e es un número natural $\rightarrow$ $0 \le X_e \le 2^n - 1$

Por tanto:

$$X_{MIN} \rightarrow X_e = 0 \rightarrow X = -2^{n-1} + 1$$

$$X_{MAX} \rightarrow X_e = 2^n - 1 \rightarrow X = 2^n - 1 - 2^{n-1} + 1 = 2^{n-1}$$

El rango será: $X \in \left[-2^{n-1} + 1 \ldots . 2^{n-1}\right] \rightarrow$ es asimétrico



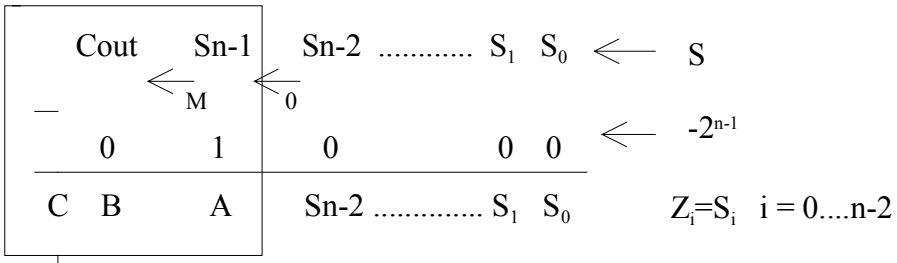
Este rango los mantiene ordenados $\rightarrow$ bueno para comparaciones rápidas

$$X_{n-1} = 0 \rightarrow X \le 0$$
$$X_{n-1} = 1 \rightarrow X > 0$$

SUMA EN EXCESO $2^{n-1} - 1$

$$Z = X + Y$$

$$Ze = Z + 2^{n-1} - 1 = X + Y + 2^{n-1} = X + Y_e = X + 2^{n-1} - 1 + Y_e - 2^{n-1} + 1 =$$
$$X_e + Y_e - 2^{n-1} + 1$$



Cout	Sn-1	M	A	C	B	(B=Cout - M)
0	0	1	1	1	1	underflow
0	1	0	0	0	0	
1	0	1	1	0	0	
1	1	0	0	0	1	overflow

$Z_i = S_i$ i = 0 ..... n-2

A = $Z_{n-1} = \overline{S_{n-1}}$

$\left[Z_e = Z + 2^{n-1} - 1\right];$
$Z = Z_e - 2^{n-1} + 1 \geq -2^{n-1} + 1$

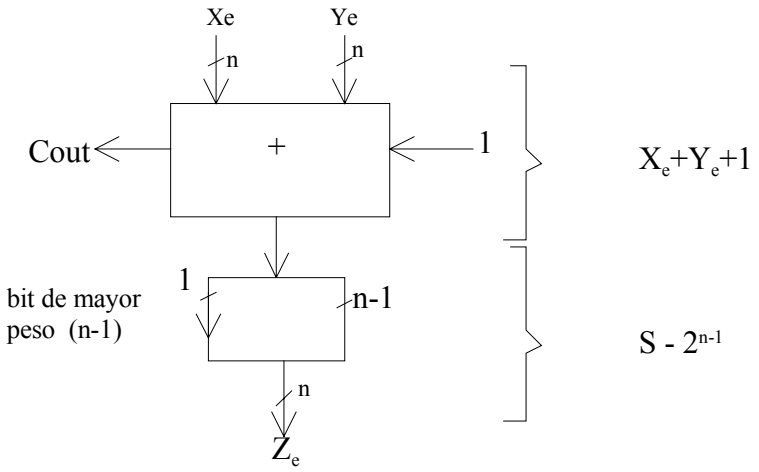
	C	B	
$Z_e \geq 0$	1	1	← *Underflow* (ovf. por debajo)
	0	1	← *Overflow* (por arriba)

OVERF = $Cout \oplus A$ $A = Z_{n-1} = \overline{S}_{n-1}$

↑

overflow + underflow

Representación:



CAMBIO DE SIGNO EN EXCESO

$$Z = -X; \qquad X_e = X + 2^{n-1} - 1 \rightarrow X = X_e - 2^{n-1} + 1$$

$$Z_e = Z + 2^{n-1} - 1 = -X + 2^{n-1} - 1 = 2^{n-1} - 1 + 2^{n-1} - 1 - X_e =$$

$$2^n - 1 - X_e - 1 = \begin{cases} \left(\left(2^n - 1\right) - X_e\right) - 1 = \overline{X_e} - 1 \\ \left(2^n - 1\right) - \left(X_e + 1\right) = \overline{X_e + 1}(mejor) \end{cases}$$

RESTA EN EXCESO

$$Z = X - Y = X + (-Y)$$

Otra opción es realizar un algoritmo directo manipulando expresiones.

$$\left.\begin{array}{l} Z_e = Z + 2^{n-1} - 1 \\ Y_e = Y + 2^{n-1} - 1 \\ X_e = X + 2^{n-1} - 1 \end{array}\right\} Z_e = X - Y + 2^{n-1} = X_e - Y = X_e + 2^{n-1} - 1 - Y_e =$$

$$= X_e + 2^{n-1} - 1 - Y_e - 2^{n-1} + 2^{n-1} =$$

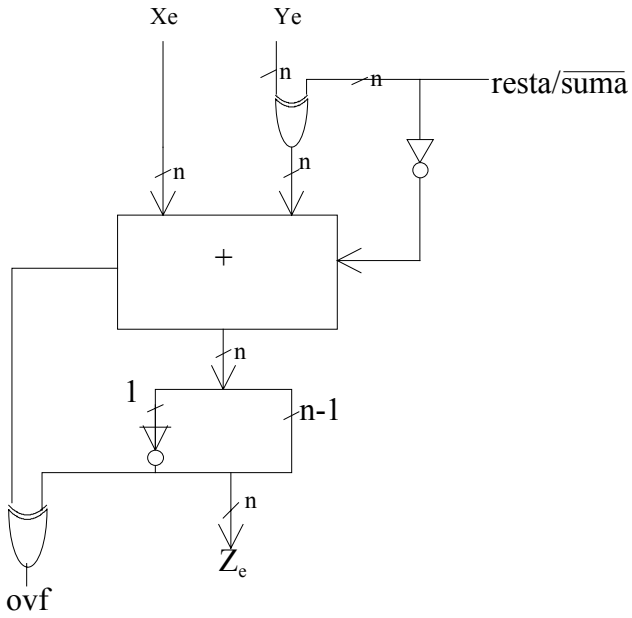
$$= \underbrace{X_e + 2^n - 1 - Y_e - 2^{n-1}}_{\overline{Y_e}} = \underbrace{X_e + \overline{Y_e} - 2^{n-1}}_{S}$$

Cout	Sn-1	Sn-2	S_1	S_0
0	1	0	0	0

→ Los bits que faltan se obtienen igual que en la suma.

Las operaciones en exceso son igual que hasta ahora sólo que hay que restar 2^{n-1} al final. Este produce overflow y underflow.

SUMADOR/RESTADOR EN EXCESO $2^{n-1} - 1$



* Ejercicios 20 , 15 , 25

1.5 - DESPLAZAMIENTOS ARITMÉTICOS

$\rightarrow$ Multiplicar o dividir por la base

* Desplazamiento izquierda:

$z = x \cdot r$

* Desplazamiento derecha:

x ¡ r
d z

$z \cdot r + d = x \leftrightarrow z + d/r = x/r$

Vamos a estudiar el desplazamiento para naturales y enteros.

A) N

- Desplazamiento izquierda:

$$x = X_{n-1}r^{n-1} + X_{n-2}.r^{n-2} + \ldots\ldots + X_1 r + X_0 \rightarrow X = \left(X_{n-1}, X_{n-2}, \ldots, X_1, X_0\right)$$

$$z = r.x = X_{n-1}r^{n} + X_{n-2}r^{n-1} + \ldots\ldots + X_1 r^2 + X_0 r + 0 \rightarrow$$

$$Z = \left(X_{n-1}, \underbrace{X_{n-2}, \ldots, X_1, X_0, 0}_{n} \right)$$

↓

Si es distinto de '0' $\rightarrow$ overflow

- Desplazamiento derecha:

$$\frac{d}{r} + z = \frac{x}{r} = \frac{X_{n-1}r^{n-1} + X_{n-2}.r^{n-2} + \ldots\ldots + X_1 r + X_0}{r} =$$

$$X_{n-1}r^{n-2} + X_{n-2}.r^{n-3} + \ldots\ldots + X_2 r + X_1 + \frac{X_0}{r} \rightarrow$$

$$Z = \left(0, X_{n-1}, X_{n-2}, \ldots, X_2, X_1\right)$$

B) Z sistema complementado, base = 2

- C '2

- Desplazamiento izquierda:

$$z = r.x = 2.x = x + x$$

$$\begin{array}{cccccc} & X_{n-1} & X_{n-2} & ..... & X_1 & X_0 \\ & X_{n-1} & X_{n-2} & ..... & X_1 & X_0 \\ \hline X_{n-1} & X_{n-2} & X_{n-3} & ..... & X_0 & 0 \end{array}$$

$\downarrow \quad \equiv$ N

$\exists$ overflow cuando $X_{n-1} \neq X_{n-2}$

- Desplazamiento derecha:

$$\frac{d}{r} + z = \frac{x}{r}$$

$$z = \begin{cases} Z_e, si, (Z_{n-1} = 0) \\ Z_e - 2^n, si, Z_{n-1} = 1) \end{cases}$$

Vamos a verlo para el caso < 0:

$$x = X_e - 2^n \rightarrow \frac{x}{2} = \frac{X_e}{2} - 2^{n-1}$$

$$\frac{d}{r} + Z_e - 2^n = \frac{X_e}{2} - 2^{n-1} \rightarrow \frac{d}{r} + Z_e = \frac{X_e}{2} - 2^{n-1} + 2^n = \frac{X_e}{2} + 2^{n-1}$$

N $\quad (0, 0\ ,\ X_{n-1}, ... X_1)$

$\rightarrow$ replicamos el bit de signo

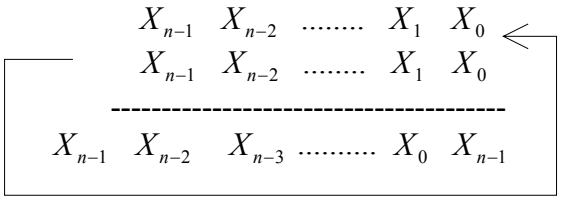
<0 $\quad (1, 1\ ,\ X_{n-1}, ... X_1)$

En general: $\quad (X_{n-1} \quad X_{n-1} \quad X_{n-2}$ **.....** $X_1)$

- C'1

- Desplazamiento izquierda:

$$z = r.x = 2.x = x + x$$



$$C_{i+1} = X_i \qquad C_0 = C_n = X_{n-1}$$
$$S_i = C_i$$

El bit de mayor peso pasa delante.

- Desplazamiento derecha

- Igual que en C'2.

$$(X_{n-1} \quad X_{n-1} \quad X_{n-2} \ \ldots\ldots\ldots \ X_1)$$

1.6 - ALGORITMOS ARITMÉTICOS PARA LA MULTIPLICACIÓN DE 'N'

$x, y \in \left[0, \ldots r^{n-1}\right]$

$z = x.y$

Vamos a ver cuantos bits necesitamos para dar el resultado.

$Z_{MIN} = 0$

$Z_{MAX} = \left(r^n - 1\right)^2 = r^{2n} - 2r^n + 1 \quad \rightarrow$ 2.n dígitos

$$N \rightarrow N^{2n}$$

$$z \rightarrow Z$$

1.6.1 - Multiplicación secuencial de naturales

$$z = x.y = \sum_{i=0}^{n-1} X_i r^i \sum_{j=0}^{n-1} Y_j r^i = \ldots\ldots = \sum_{k=0}^{2n-1} Z_k r^k$$

Para hacer el algoritmo utilizaremos otra opción:

Algoritmo Aritmético

$z = x.y = X.\sum_{i=0}^{n-1} Y_i r^i \sum_{i=0}^{n-1} X.Y_i r^i$ Desplazamos a la izquierda y multiplicamos un nº par por un dígito.

X_{n-1} X_{n-2} ..... X_1 X_0
Y_{n-1} Y_{n-2} ..... Y_1 Y_0
---
x x ..... x x
x x x x
x x x x

$Z_{n-1} Z_{n-2}$ ..... Z_1 Z_0

Lo que haremos es rescribir la expresión

$$P = \sum_{i=0}^{n-1} X.Y_i r^i \quad \text{como recurrencias}$$

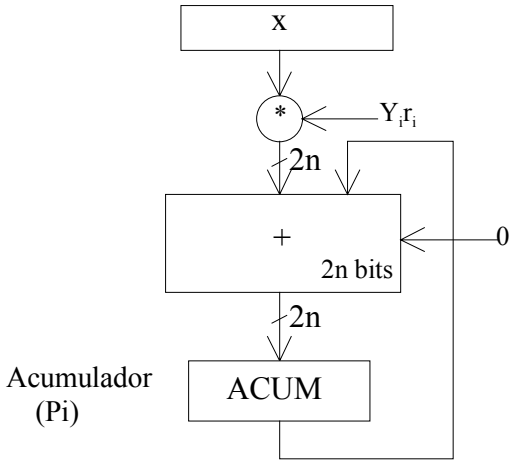
RECURRENCIAS:

i)

$$\begin{cases} P_0 = 0 \\ P_{i+1} = P_i + X.Y_i.r^i, (i = 0...n-1) \\ P = P_n \end{cases}$$

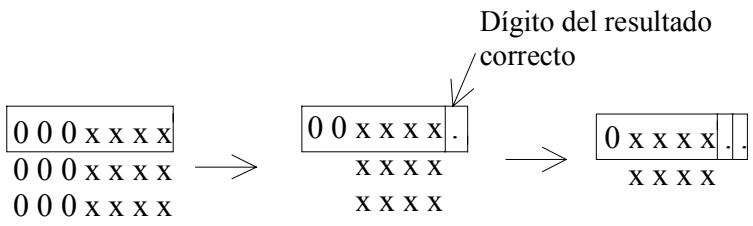
Se hace un producto parcial y se acumula en P.

Implementación:



ii)

$$\begin{cases} P_0 = 0 \\ P_{i+1} = r^{-1}(P_i + X.Y_i r^n) \\ P = P_n \end{cases}$$



Con un sumador de 'n' dígitos es suficiente

Ejem: n = 4

$$P_0 = 0$$

$$P_1 = r^{-1} X.Y_0 r^4$$

$$P_2 = r^{-1}\left(r^{-1} XY_0 r^4 + XY_1 r^4\right)$$

$$P_3 = r^{-1}\left(r^{-1}(r^{-1} XY_0 r^4 + XY_1 r^4\right) + XY_2 r^4)$$

$$P_4 = r^{-1}\left(r^{-1}(r^{-1}(r^{-1} XY_0 r^4 + XY_1 r^4\right) + XY_2 r^4) + XY_3 r^4)$$

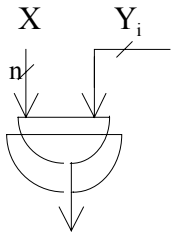
IMPLEMENTACIÓN SECUENCIAL DEL ALGORITMO

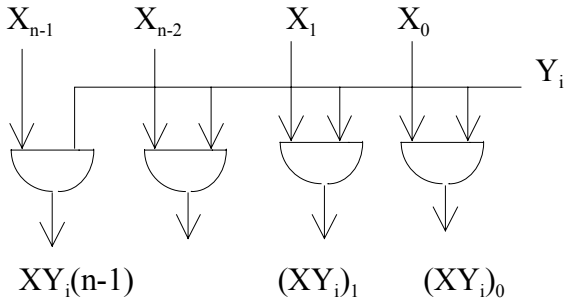
1) BASE 2

$$X.Y = \begin{cases} X, si, Y_i = 1 \\ 0, si, Y_i = 0 \end{cases}$$

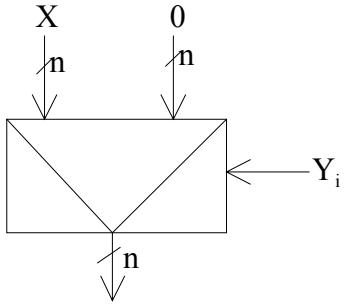
En este caso el producto siempre ocupa el mismo número de dígitos

Sismología:

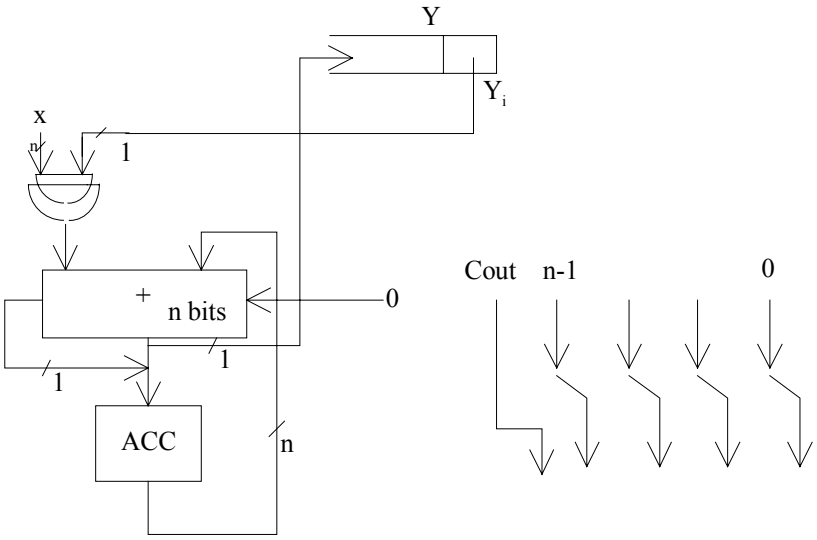




También se puede hacer com mux:



El multiplicador quedará:



El resultado será:

ACC	Y

El sumador es uno cualquiera de N

- Tiempo de multiplicación:

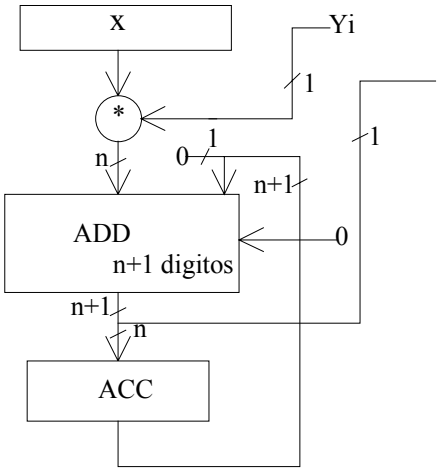
CPA $\rightarrow 0(n^2)$

CLA $\rightarrow$ logarítmico

B) GENERAL

$X.Y_i \rightarrow n+1$ dígitos

$\left(X.Y_i\right)_n \leq r-2 \leftarrow$ necesitamos un sumador de n+1 dígitos pero no sale nunca overflow

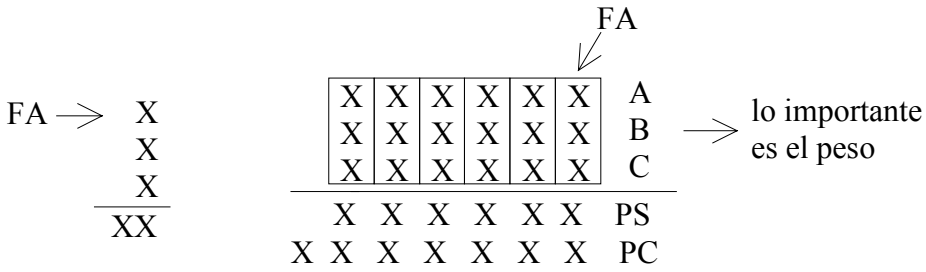


Sumamos con un número de n+1 bits pero el dígito de más peso es '0'.

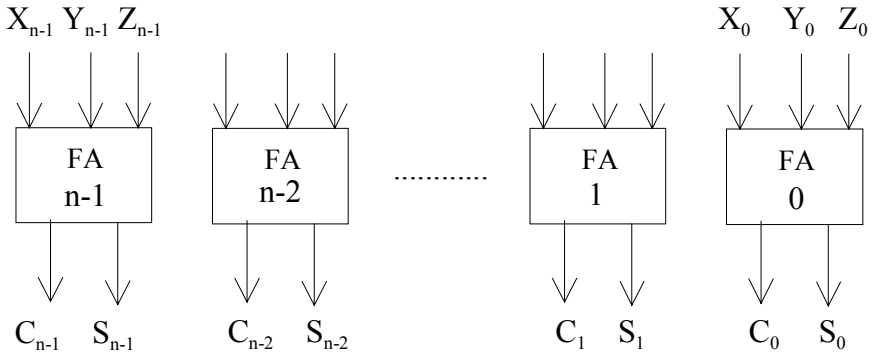
1.6.2 - Aceleración de la Multiplicación Secuencial de Naturales

- A: Acelerar los pasos
- B: Reducir los pasos

A) <u>CARRY SAVE ADDERS</u> (CSA) → CARRY SAVE MULTIPLICATION

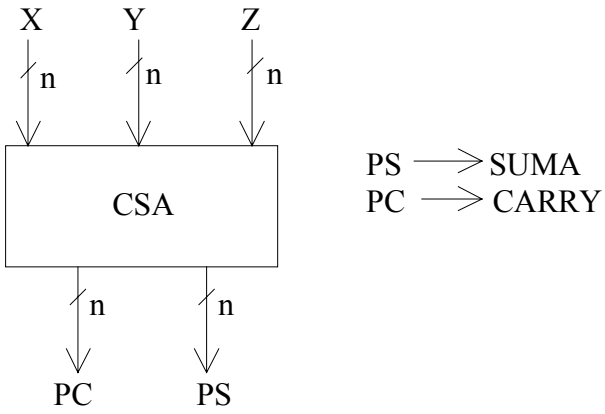


<u>CSA</u> :



- No existe conexión entre FA.

Normalmente se dibuja reorganizando entradas:



CPA

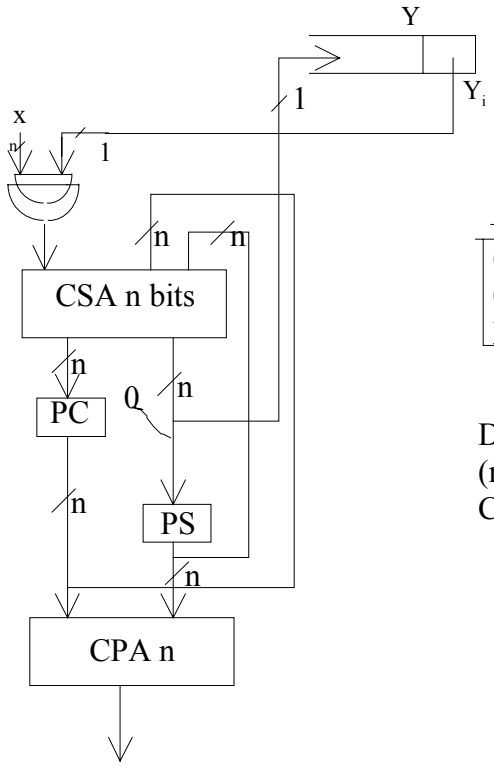
```
    X X X X   A
  X X X X     B
X X X X X     A+B
    X X X X   C
X X X X X X   A+B+C
    X X X X   D
X X X X X X X A+B+C+D
    X X X X   E
X X X X X X X X A+B+C+D+E
```

CSA

```
    X X X X   A
    X X X X   B
    X X X X   C
    X X X X   PS
  X X X X     PC
    X X X X   D
  X X X X X   PS1
X X X X X X   PC1
    X X X X   E
  X X X X X X PS2
X X X X X X X PC2
X X X X X X X X A+B+C+D+E
```

Cuantos más números sumemos y más grande sea 'n' → más grande es la diferencia.

CARRY SAVE MULTIPLICATION

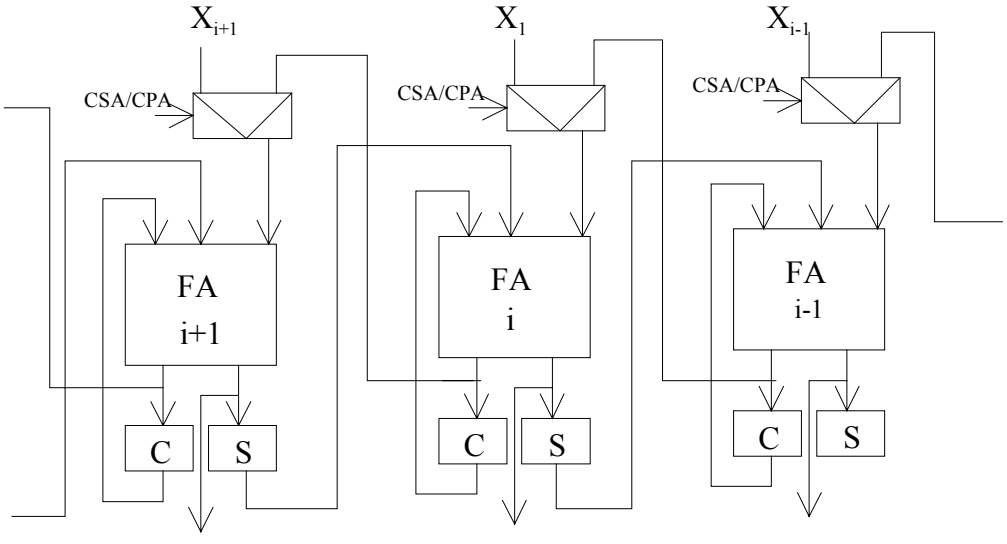


X X X X		$X.Y_0.2^0$
0 X X X	X	PS
0 0 0	0	PC
X X X	X	$X.Y_1 2^1$

Después de los n ciclos (n bits de Y) se hace el CPA n.

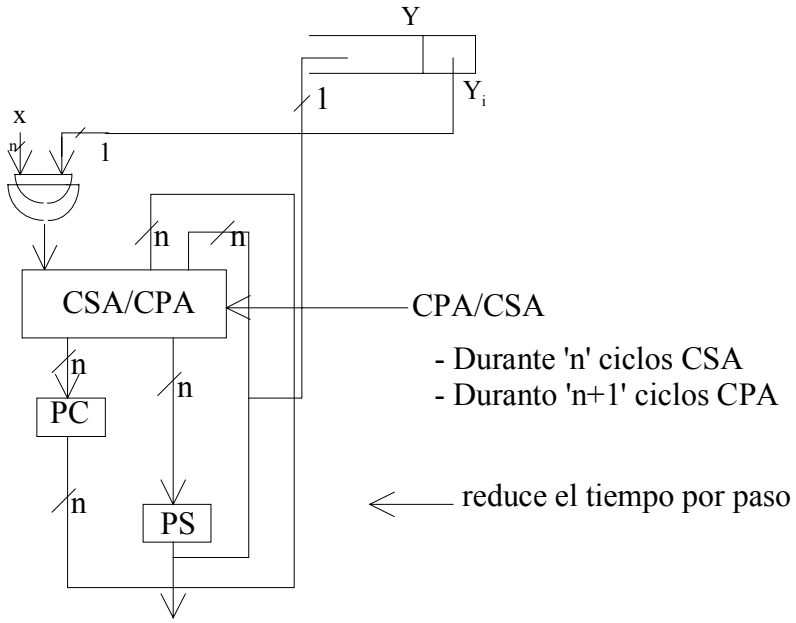
n+1 pasos

Con el mismo hardware pero conectado diferente, añadiendo un CPA conseguimos que sea mucho más rápido → Realizamos un circuito que puede trabajar como CSA o como CPA (CSA/CPA):



* Modificación sobre un CSA para que actue como CPA

El multiplicador quedará:



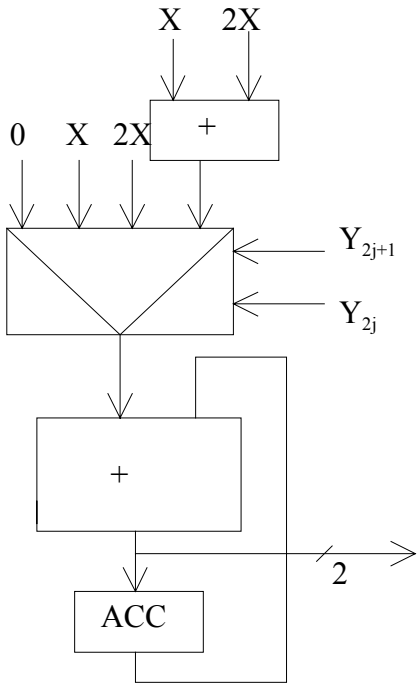
B) <u>EXPLORACION SIMULTANEA DE VARIOS DIGITOS SIN SOLAPAMIENTO</u>

- Reduce el número de pasos
- Se llama RECODIFICACIÓN $\rightarrow$ BASE 2^K
  - Ej: K = 2 $\rightarrow$ r = 4

Y_5 Y_4	Y_3 Y_2	Y_1 Y_0
Y_2^R	Y_1^R	Y_0^R
Y^2	Y^1	Y^0

$$Y_j^R = 2.Y_{2j+1} + Y_{2j}$$

Y_{2j+1}	Y_{2j}	Y_j^R
0	0	0
0	1	1
1	0	2
1	1	3



IDEA: Conceptual (falta dimensionar) tiene sentido para bases grandes.

- Ciclos $\left\lceil \frac{n}{k} \right\rceil$

- Desplazar k bits:

$$\sum_{i=0}^{\left\lceil \frac{n-1}{2} \right\rceil} Y_i^R (2^2)^i$$