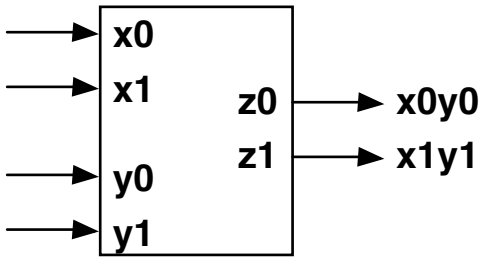
Diseño Modular de Circuitos

El número de filas de la tabla de verdad de un circuito combinacional aumenta exponencialmente con el número de entradas (mientras que el número de columnas aumenta linealmente con las entradas y salidas). Esto significa que la metodología que vimos no sirve para circuitos que tienen muchas entradas.

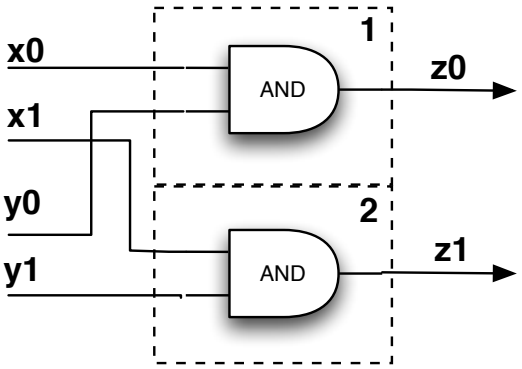
Los circuitos complejos (con muchas entradas) se diseñan acoplando circuitos más simples. Esto último se denomina **diseño modular**.

Estrategias de Diseño Modular

Primero: Diseño en Paralelo: Ejemplo a la derecha. Las salidas de 1 no dependen de ninguno de los cálculos que se hacen en 2, y viceversa ⇒ 1 es completamente independiente de 2.



Segundo: Diseño en Cascada: Por ejemplo el sumador de números de n bits que usa el circuito sumador de 3 bits en cascada. Una de las entradas de cada sumador de 3 bits (el carry) depende de una salida de otro sumador de 3 bits.

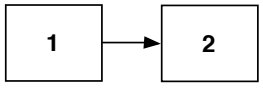


Tercero: Diseño Serial o Secuencial. El cálculo se descompone en **n** etapas o cálculos intermedios. Cada uno de estos cálculos se ejecuta en un ciclo del reloj. La idea es utilizar el mismo módulo para realizar los **n** cálculos intermedios en **n** ciclos. Ejemplo: un sumador serial.

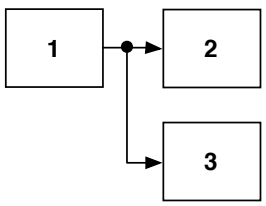
Cuarto: Diseño en Pipeline: Esto se verá al final del curso.

Estrategias de comunicación entre módulos

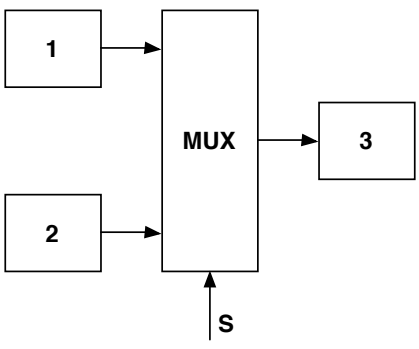
Primero: Punto a Punto



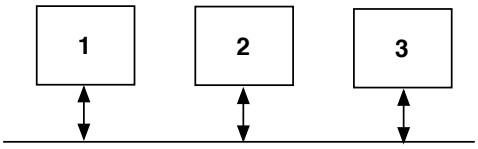
Segundo: **1** a **n**



Tercero: **n** a **1**



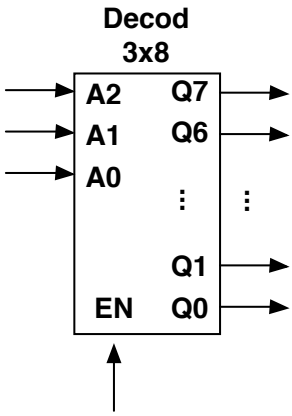
Cuarto: Bus



Elementos modulares para el diseño de circuitos

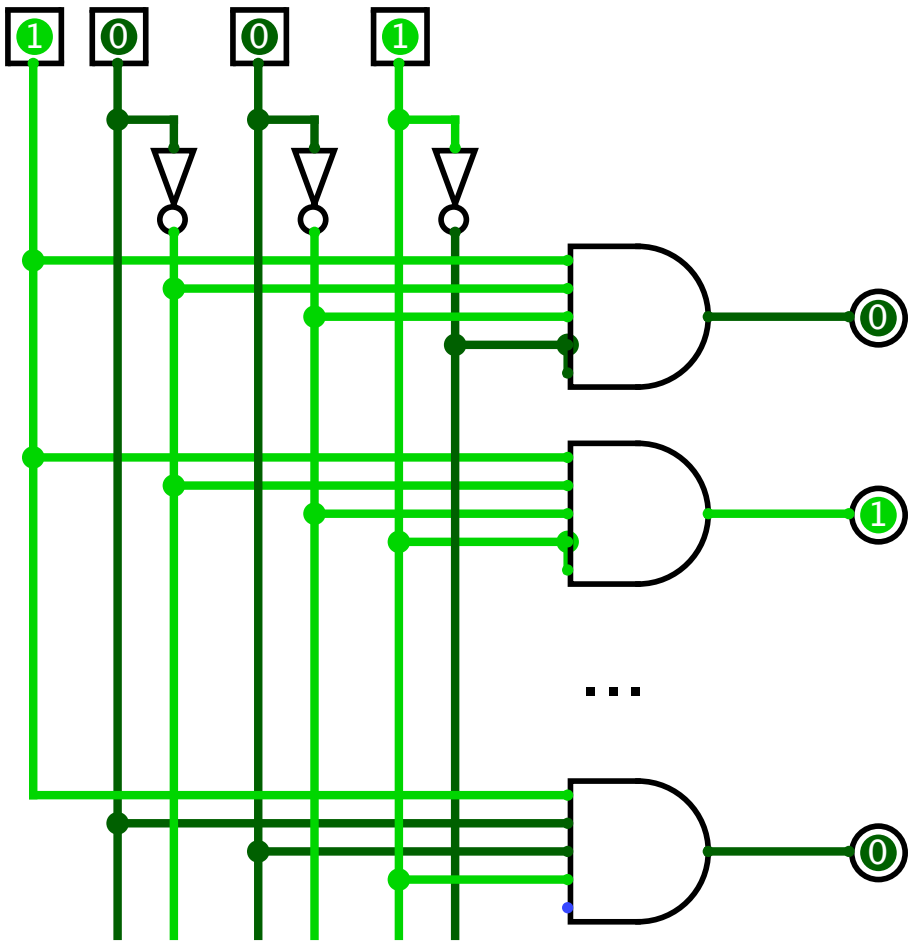
Decodificador

Descripción: si **EN = 0** todos los Q_j están en **0**. si **EN = 1** y el valor de **A2-A0** es **i**, entonces Q_i **= 1** y el resto **= 0**.



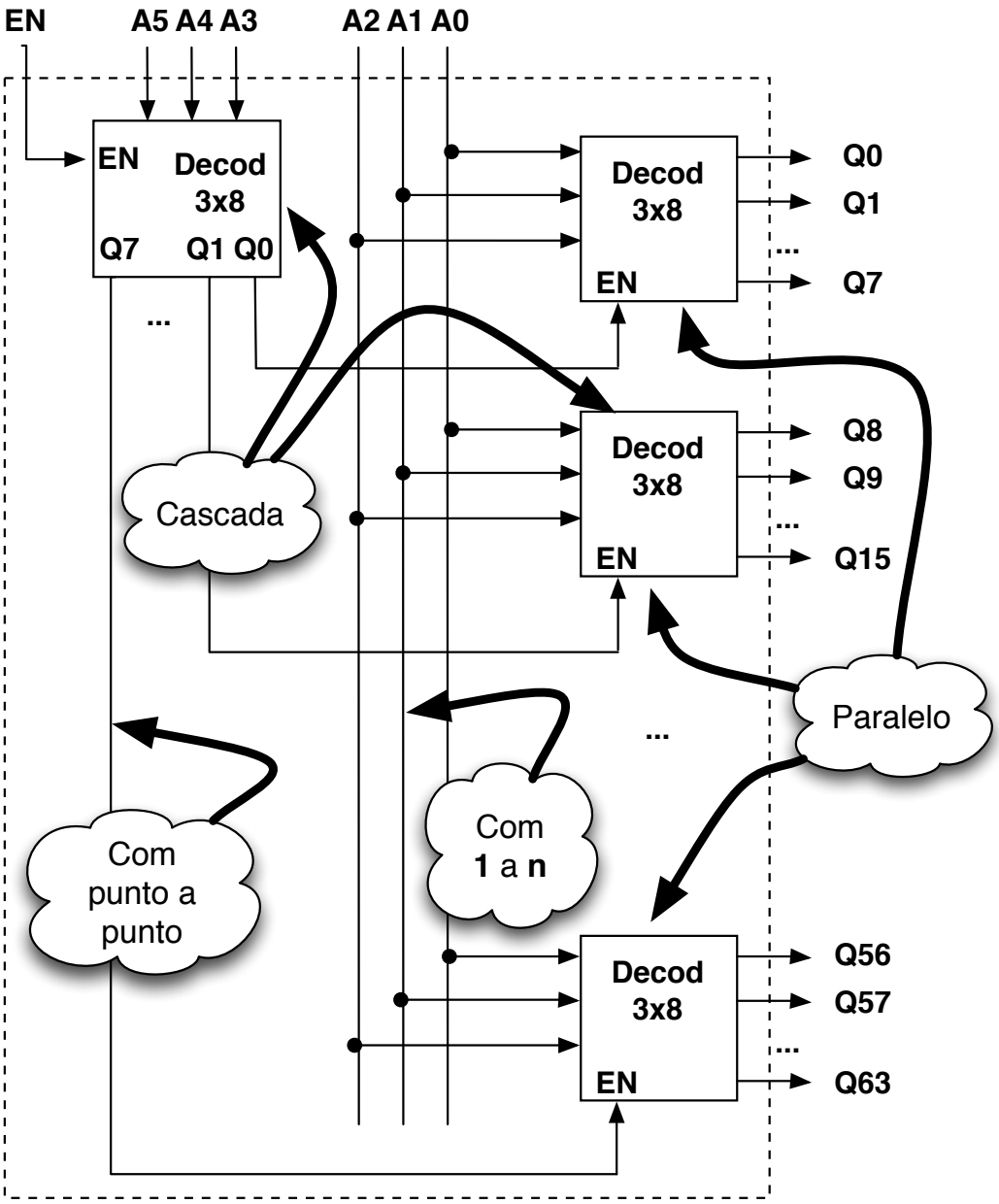
$$Qj = \begin{cases} 0 \text{ si } EN = 0 \\ 0 \text{ si } i \neq j,\ EN = 1 \\ 1 \text{ si } i = j,\ EN = 1 \end{cases}$$

Implementación:

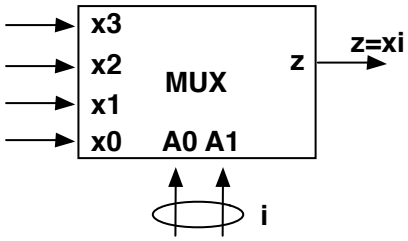


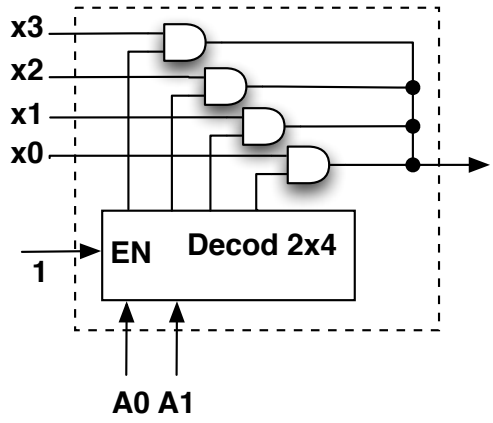
Descodificador de 6x64

Se usan 9 decodificadores 3x8 puertas en cascada y/o paralelo.



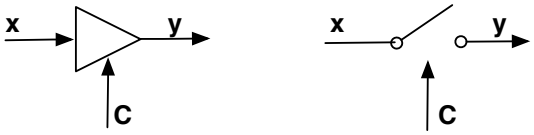
Multiplexor:

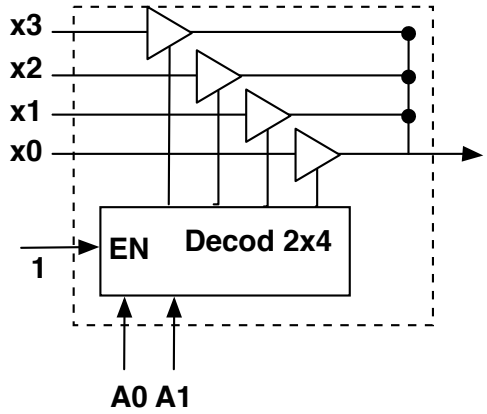




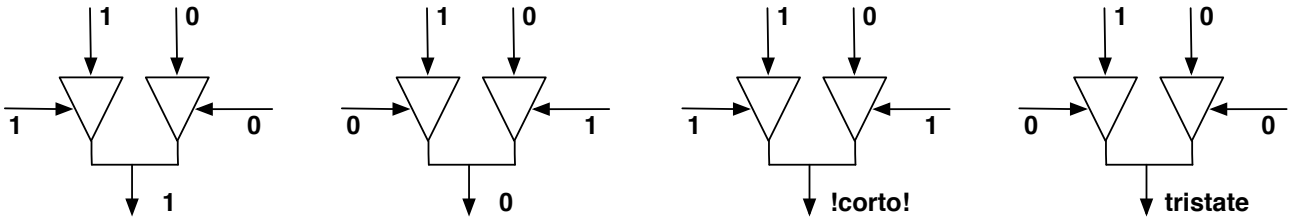
Implementación usando compuertas tristate:

y = x si **c = 1**, **y = tristate** si **c= 0**



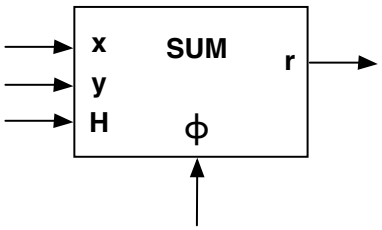


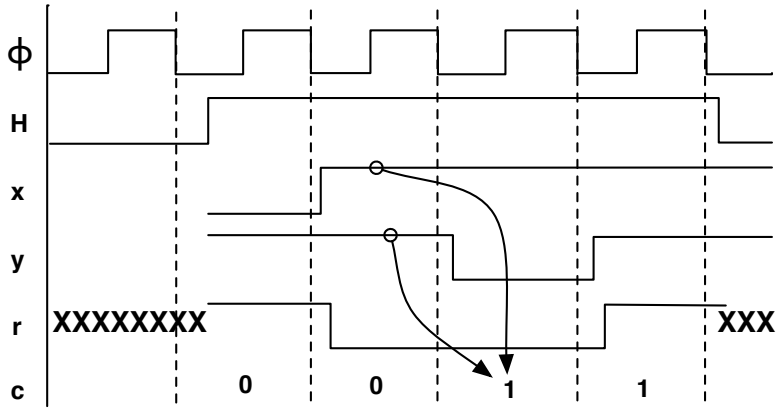
Una salida en estado tristate está desconectada y por lo tanto no hay corto circuito si otra fuente aplica un potencial a la misma línea.

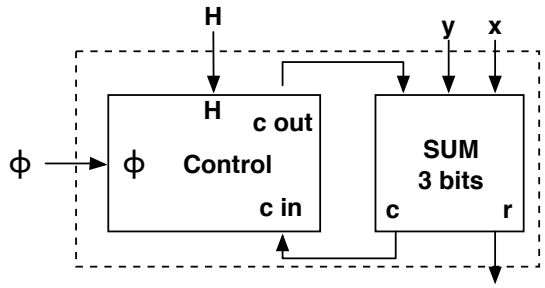


Sumador Serial

El sumador recibe serialmente dos números binarios de menor a mayor significancia. La suma comienza cuando **H** pasa de **0** a **1** y termina cuando **H** pasa de **1** a **0**.

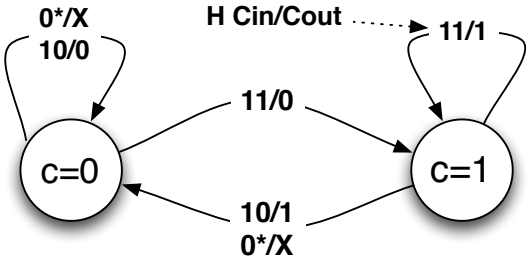




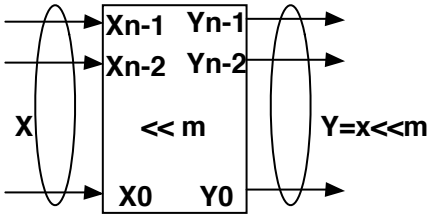


Diseño modular en serie: se reutiliza el sumador de 3 bits ya visto y se introduce un circuito sequencial que se acuerda del acarreo **c**.

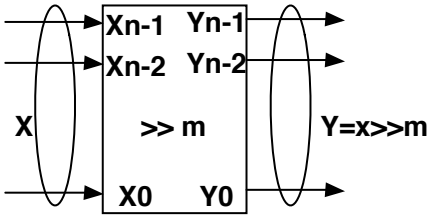
Diagrama de estados para el circuito "Control" (estados = carry generado en el ciclo anterior).



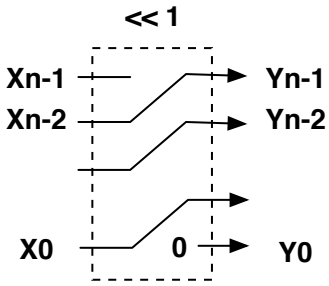
Left Shifter / Right Shifter



$$Yj = \begin{cases} 0 \text{ si } j < m \\ Xj-m \text{ si } j >= m \end{cases}$$



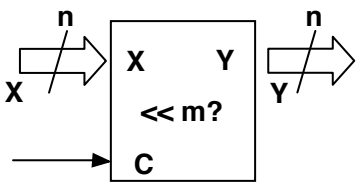
$$Yj = \begin{cases} 0 \text{ si } j >= n-m \\ Xj-m \text{ si } j < n-m \end{cases}$$



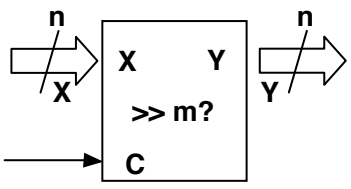
Solo mostramos la implementación de **<<1**

Nótense que **m** es una constante fija para el circuito. Más tarde usaremos estos circuitos para implementar un circuito que reciba **m** como argumento.

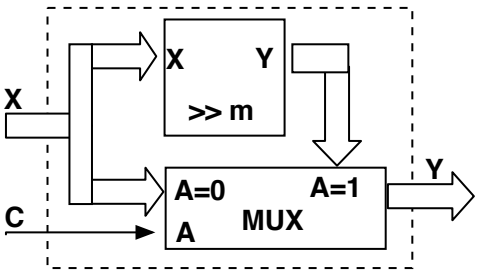
Conditional Left Shifter / Right Shifter



$$Y = \begin{cases} X \text{ si } c = 0 \\ X << m \text{ si } c = 1 \end{cases}$$



$$Y = \begin{cases} X \text{ si } c = 0 \\ X >> m \text{ si } c = 1 \end{cases}$$



Solo mostramos la implementación de **>>m?**

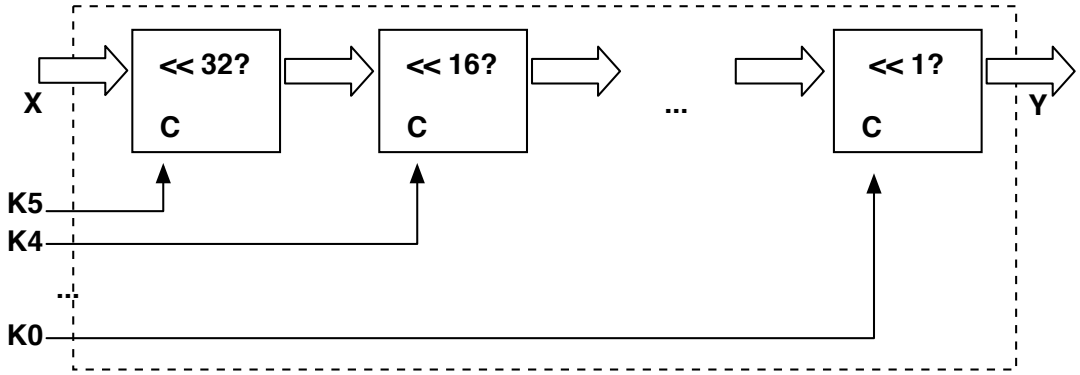
Barrel Shifter

Implementación: usando las siguientes propiedades:

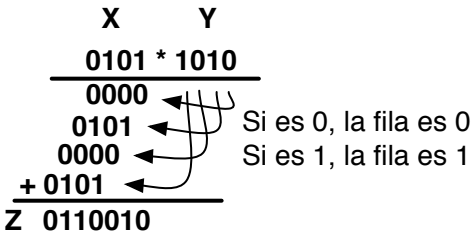
$\mathbf{m = K_0.2^0 + K_1.2^1 + K_2.2^2 + K_3.2^3 + K_4.2^4 + K_5.2^5}$ y $\mathbf{x<<(n+l) = (x << n) << l}$

$\mathbf{\Rightarrow x << m = (((((x << K_5.2^5) << K_4.2^4) << K_3.2^3) << K_2.2^2) << K_1.2^1) << K_0.2^0}$

Usando esta última fórmula haremos una implementación en cascada.



La Multiplicación en Cascada



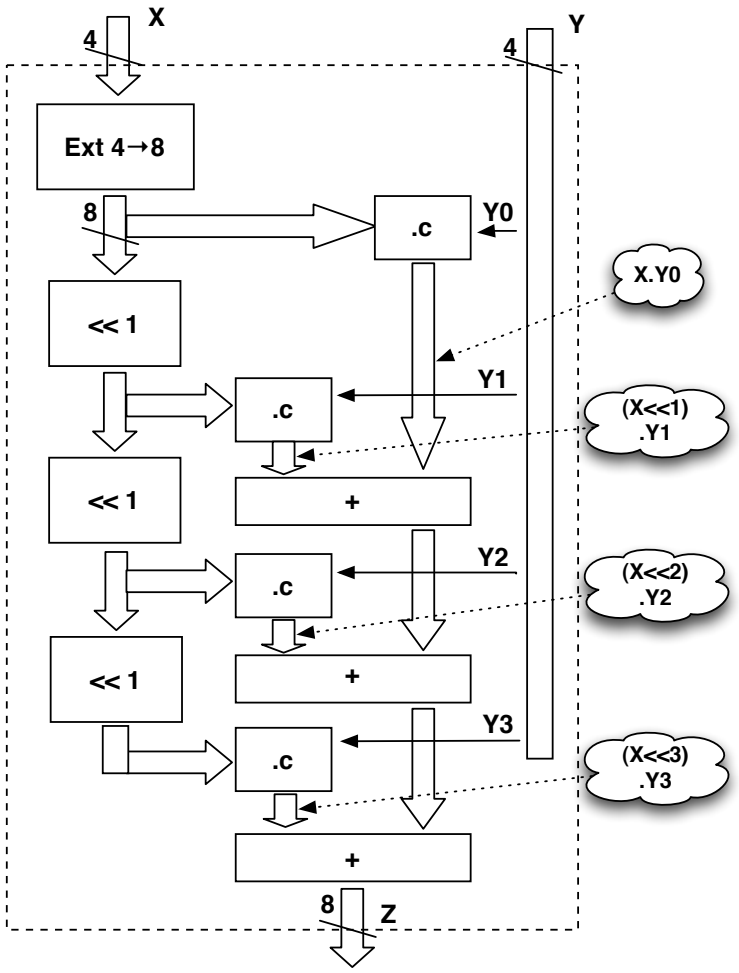
La multiplicación en binario es más simple que la misma en decimal.
El resultado del producto de **2** números de **n** bits tiene **2n** bits como máximo.

Para la implementación necesitamos un circuito que extiende un número de **n** bits a uno de **m** bits, agregando **0** a la izquierda, y un multiplicador por un valor de un bit (es decir, calcula **x.1** o **x.0**). La implementación de ambos es trivial (Primero: inputs siempre 0. Segundo: usando compuertas AND).

Implementación: pagina siguiente

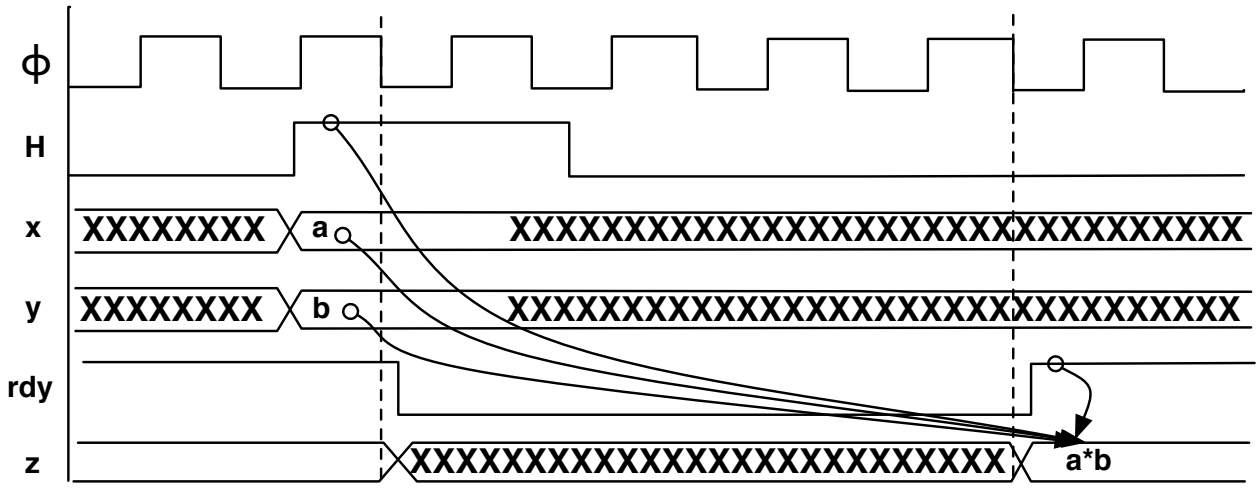
El problema de este circuito es que es my costoso. Para multiplicar cantidades de 32 bits se requieren 31 sumadores ! Por esta razón, los microprocesadores más económicos realizan una multiplicación en serie.

Pseudo-code: pagina siguiente.

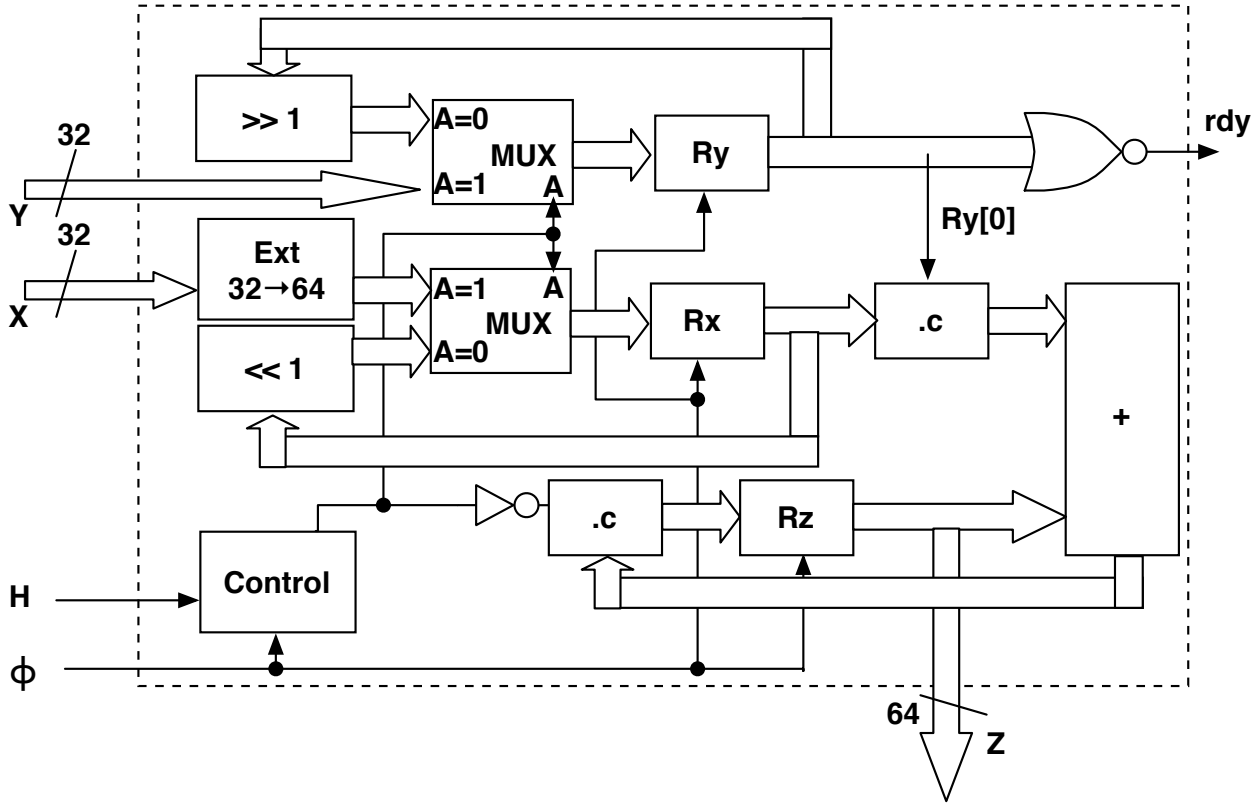



```
Mult (x,y de 32 bits){

sea Rx de 64 bits =
          Ext 32→64 (x);
sea Ry de 32 bits = y;
sea Rz de 64 bits = 0;

while (Ry != 0) {
   if(Ry[0] != 0)
         Rz += Rx;
   Rx := Rx << 1;
   Ry := Ry >> 1;
}

return Rz;
}
```

La multiplicación en serie se implementará en un circuito con el siguiente diagrama de tiempo.



La transición de **0** a **1** en **H** indica que en **x** e **y** hay valores que deben multiplicarse. El multiplicador calcula de a un bit por cada ciclo.


>> 1
A=0
MUX
A=1
A
Ry
rdy
32
Y
32
X
Ext
32→64
<< 1
A=1
A
MUX
A=0
Rx
.c
Ry[0]
+
.c
Rz
Control
H
ϕ
64
Z