# 1 Introducción

## 1.1 IFQ

La IFQ es una FIFO que aloja las instrucciones leídas de la cache de instrucciones (I-Cache) y de donde la unidad de despacho (Dispatch Unit) las toma para que sean decodificadas, la unidad de despacho toma una instrucción a la vez. La FIFO tiene 16 localidades por lo tanto puede alojar un máximo de 16 instrucciones. En esta implementación, cuando leemos instrucciones de la I-Cache traemos una línea de cache que consta de 4 instrucciones (128 bits). Entonces en este diseño, desde el punto de vista de las escrituras la FIFO es vista como un arreglo de 4 x 128 y desde el punto de vista de las lecturas como un arreglo de 16 x 32.

## 1.2 I-Cache

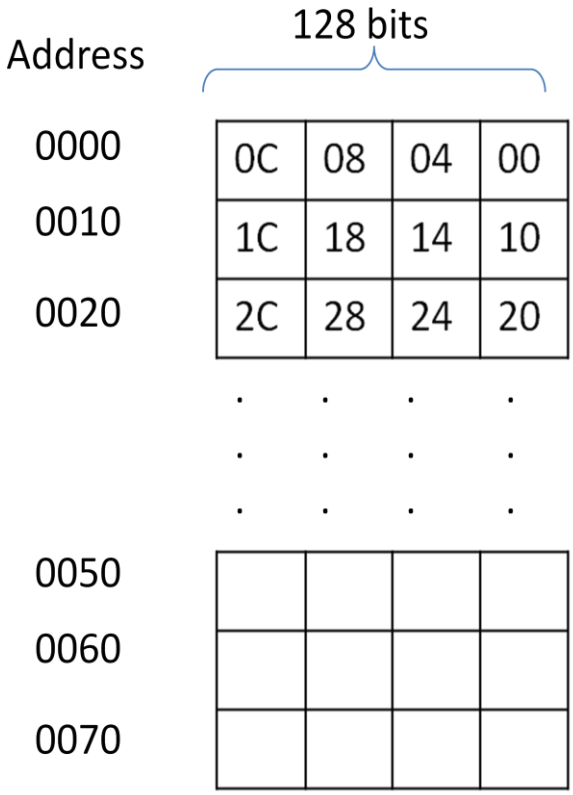
En nuestro diseño la I-Cache la vamos a implementar utilizando una BRAM de 32 x 128, entonces podemos almacenar 128 instrucciones. La BRAM en nuestro diseño más bien se comporta como una ROM ya que el contenido es pre-cargado durante la síntesis o al inicio de la simulación.

# 2 Especificaciones de Diseño.

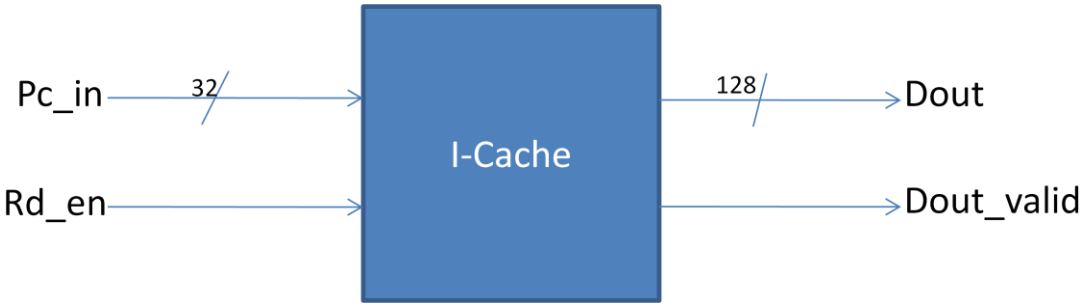
## 2.1 I-Cache

En la I-Cache siempre vamos a leer una línea de cache completa (128 bits) sin importar que la dirección no esté alineada a 128 bits o 16 bytes, por lo tanto a la dirección de entrada los 4 bits menos significativos serán atados a “0000”. Puede suceder que después de un BRANCH en vez de entregar 4 instrucciones “útiles” únicamente entregaremos 1, 2 o 3. La IFQ se encarga de elegir las instrucciones “útiles”.

Esta BRAM no tiene una señal de *enable* para el registro de la dirección de entrada, por lo tanto es importante que la IFQ mantenga la dirección de entrada estable hasta que la línea de cache sea regresada.




*Distribución de la I-Cache*

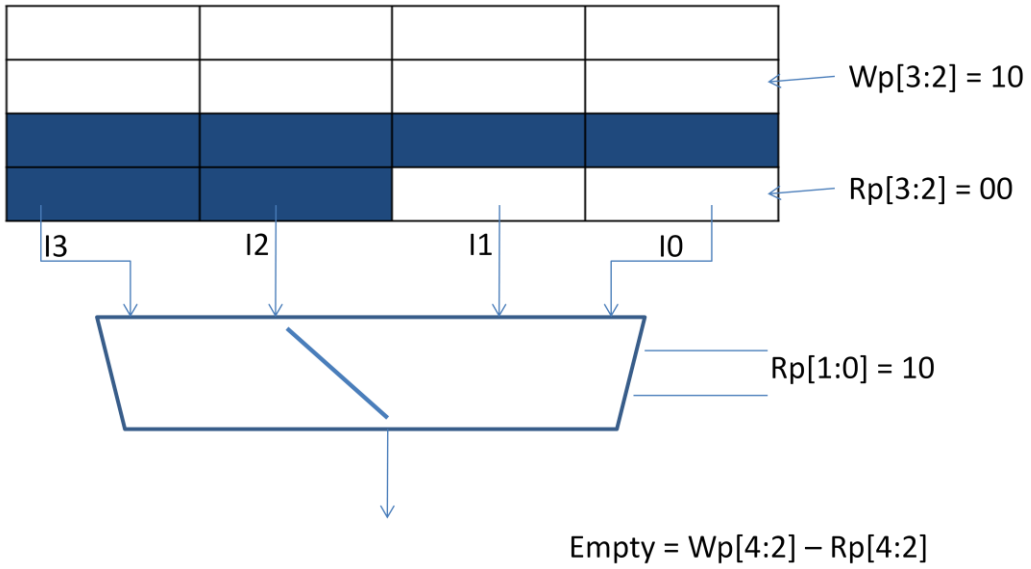



*I-Cache Interface.*

**Pc_in:** la dirección de entrada de 32 bits.
**Rd_en:** señal de control de lectura activa en alto.
**Dout:** la línea de cache de 128 bits
**Dout_valid:** señal que indica que los datos en Dout son validos.

## 2.2 IFQ

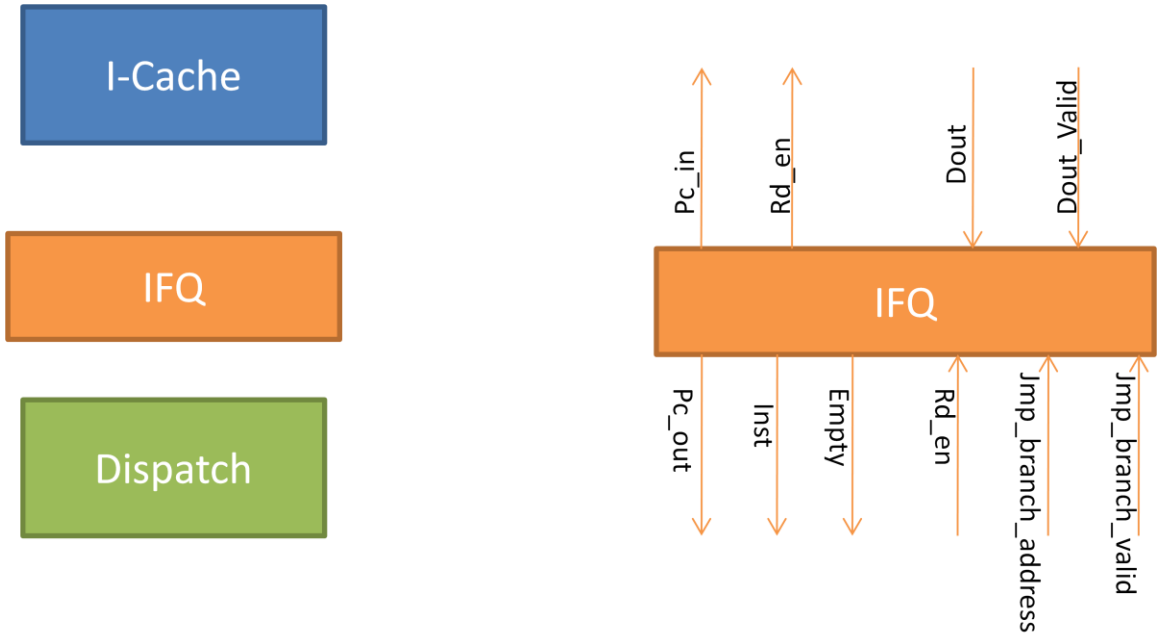
Para lograr mostrar una vista de 4 x 128 desde el punto de vista de la escritura y de 16 x 32 desde el vista de vista de la lectura la estructura de IFQ puede ser diseñada como se muestra en la siguiente figura.




Los apuntadores de escritura y lectura, Wp y Rp respectivamente, son de 5 bits para poder direccionar las 16 localidades de la FIFO y también para encargarse de generar las banderas de *vacía* y *llena* fácilmente.

La IFQ debe de ser vaciada (flush) cada vez que una instrucción del tipo branch o jump sea resuelta. Los bits [4:2] tanto del apuntador de lectura como de escritura deben tomar el valor de “000”, los bits [1:0] del apuntador de lectura debe de tomar el valor de los bits [3:2] de la dirección de salto. Los bits [1:0] del apuntador de escritura pueden ser igualados a los del apuntador de lectura o bien siempre al valor “00”.

El IFQ interactúa tanto con la I-Cache como con la unidad de despacho. La siguiente figura muestra la interfaz que la IFQ comparte con estos dos bloques.




*IFQ: interfaz con I-Cache y Dispatch.*

**Pc_out:** 32bits del *program counter* (PC) usado para calculo de direcciones de salto
**Inst:** 32 bits del *opcode* de la instrucción a ser decodificada.
**Empty:** Señal que indica que la IFQ esta vacía

**Rd_en:** Señal de lectura de una instrucción de la IFQ
**Jmp_branch_address:** Nuevo valor que debe tomar el PC, 32 bits
**Jmp_branch_valid:** señal que indica que el valor de jmp_branch_address es válido.

Como se puede apreciar, existen dos señales de PC, pc_in y pc_out. Lo que significa que tenemos que llevar registro de dos *Program Counters*, uno con respecto al apuntador de escritura y otro con respecto al apuntador de lectura. El PC con respecto a la escritura (pc_in) es incrementado en saltos de 16, esto porque de la I-Cache leemos una línea de 4 instrucciones. El PC con respecto a la lectura (pc_out) es incrementado en saltos de 4, esto porque pc+4 es utilizado en el cálculo de la dirección de salto tanto para las instrucciones del tipo *branch* o del tipo *jump.*

### 2.3 Path Crítico.

En el diseño de todo bloque funcional es sumamente importante no perder ciclos de reloj en los casos de frontera, en este proyecto vamos a optimizar para perder el menor número de ciclo de reloj en estas situaciones de frontera.

En ejecución “idónea” la IFQ cada ciclo de reloj le pide a la I-Cache 4 nuevas instrucciones y la unidad de despacho le pide a la IFQ una nueva instrucción cada ciclo de reloj, por lo tanto la IFQ debe de ser capaz de realizar estas operaciones en un ciclo de reloj.

A continuación se enlistan las diferentes situación que pueden ocasionar que la IFQ no le pida a la I-Cache nuevas instrucciones cada ciclo de reloj.

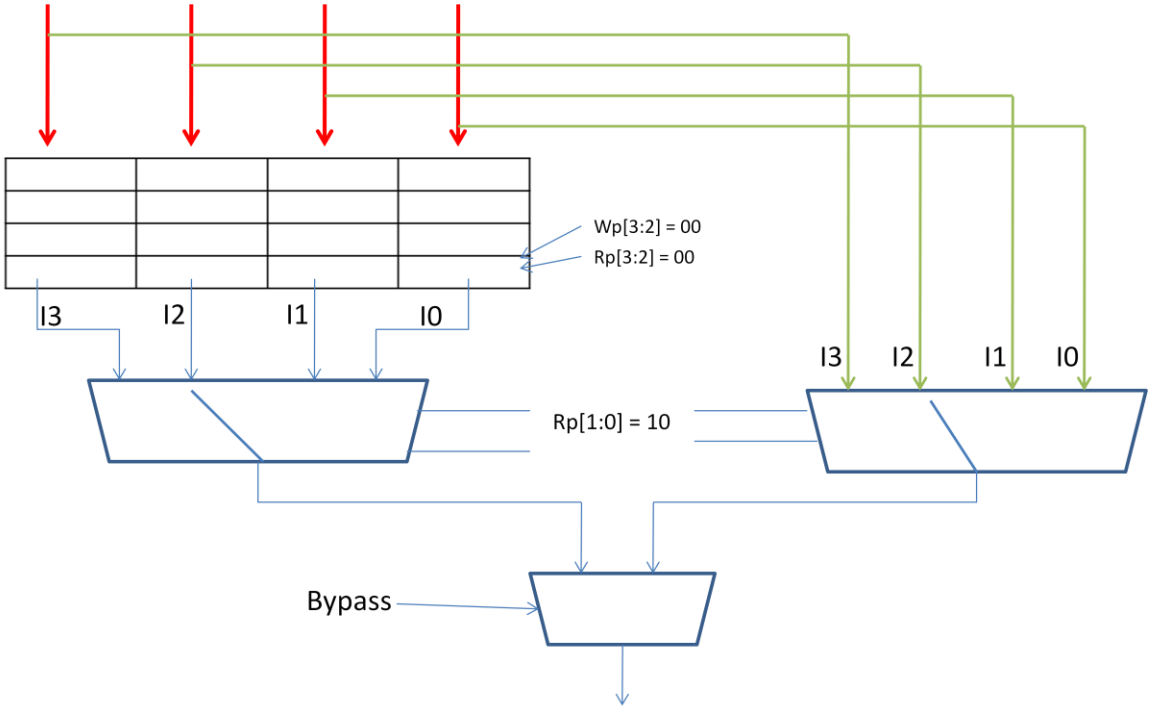
1) la IFQ no tenga lugar para guardar las 4 nuevas instrucciones: recuerden que la instrucciones salen de la IFQ una a una y entran de 4 en 4 lo cual fácilmente puede producir este escenario. En este caso tenemos que esperar hasta que la IFQ tenga lugar para almacenar 4 nuevas instrucciones para poder pedírselas a la I-Cache.

2) La IFQ recibe una dirección de salto: la IFQ tiene que ser vaciada y tiene que empezar a pedir instrucciones a partir de la dirección de salto recibida. En este escenario la IFQ debe de ser capaz de transmitir la dirección de salto a la I-Cache en el mismo ciclo de reloj en el que la recibe.

El caso critico para que la IFQ no pueda entregarle una nueva instrucción a la unidad de despacho es cuando la FIFO sale del estado *vacía.* Dos escenarios obvios donde la FIFO se encuentra *vacía* son los siguientes; después de reset y cada vez que la FIFO tiene que ser vaciada por una condición de salto.

Ya vimos que debemos de transmitir la dirección de salto lo antes posible a la I-Cache, ahora analizaremos como transmitir la primer instrucción a la unidad de despacho sin necesidad de guardarla primero en la FIFO, lo cual implicaría perder un ciclo de reloj mas.

La siguiente figura muestra un mecanismo de bypass para poder transmitir la primer instrucción a la unidad de despacho perdiendo el menor número de ciclos de reloj cuando la FIFO se encuentra *vacía*.


Wp[3:2] = 00
Rp[3:2] = 00
I3
I2
I1
I0
I3
I2
I1
I0
Rp[1:0] = 10
Bypass