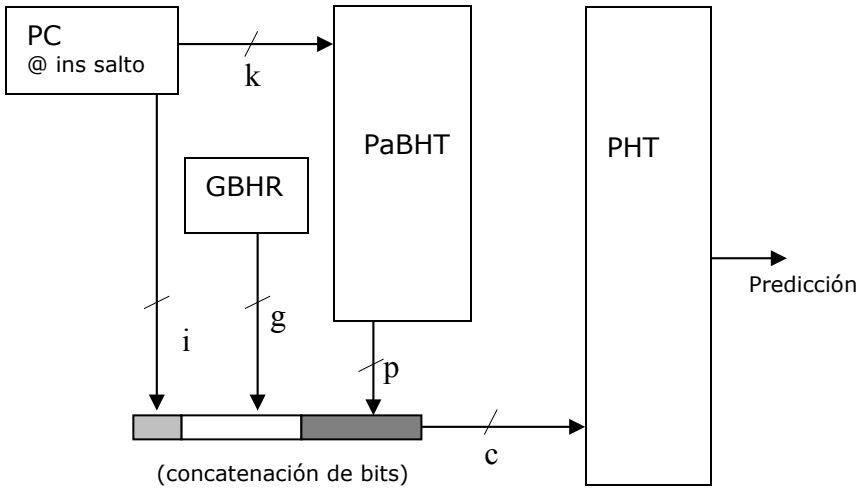
Simulación procesador Superescalar:

Predictores de Salto

2 fase:

Implementación del predictor de saltos *Alloyed*



El predictor mantiene la información de saltos ejecutados en tres estructuras.

1. *Global Branch History Register* (GBHR) guarda la historia de los últimos 'g' saltos que se han ejecutado en el procesador. Guarda un 1 si ha sido *Taken* y un 0 si ha sido *NotTaken*.
2. *Per address Branch History Table* (PaBHT) es una tabla de dimensión 2^k entradas y cada una de ellas con 'p' bits. El objetivo de esta tabla es guardar la historia de los últimos 'p' saltos de cada instrucción por separado. Se utiliza la dirección en memoria de las instrucciones de salto como identificador de ellas mismas. Como la tabla tiene una número limitado de entradas, se utilizan únicamente los 'k' bits de menos para seleccionar una entrada para una instrucción de salto. Como pega, sucede que aquellas instrucciones de salto que tengan los mismos 'k' últimos bits compartirán la misma entrada y almacenarán una historia mezclada.

3. *Pattern History Table* (PHT) es una tabla donde se almacenan dos bits para implementar un contador saturado del comportamiento de los saltos. De manera que cada vez que el salto es efectivo Taken, se incrementa y si el salto no es efectivo NotTalen se decrementa. Así se obtienen cuatro estados donde dos de ellos (los que tienen el bit de mayor peso a '1') predicen que el salto va a ser efectivo y los otros dos (los que tienen el bit de menor peso a '0') predicen que la instrucción de salto no va a saltar.

Cada vez que se ejecuta un salto estas estructuras se acceden dos veces. La primera para obtener una predicción y actuar en consecuencia en el pipeline y la segunda para actualizar el comportamiento del salto ejecutado y afinar subsiguientes predicciones.

Los parámetros a definir en este predictor són:

1. Número de bits del GBHR → define 'g'
2. Número de entradas de PaBHT y bits de cada entrada → define 'k' y 'p'
3. Número de entradas de PHT → define 'c'
4. A partir de estos valores se deduce 'i' como 'c'-'g'-'p'. Que no puede ser inferior a 1.

Los valores iniciales de estas estructuras son bits a '1'. De este modo la predicción de cualquier salto inicial será *Taken*.

Para modificar el simplescalar y añadir un nuevo predictor se puede uno fijar en la implementación de uno ya existente y duplicar añadir lo necesario. En este caso el predictor *2lev* es suficientemente parecido y será el modelo a seguir.

Las partes de simplescalar a modificar son:

- en bpred.h:
  - Añadir el nuevo predictor en la lista de *enum bpred_class*.
  - En la definición de tipos de los predictores añadir en la estructura *struct bpred_dir_t* dentro de la subestructura *two* las variables y apuntadores extras. Tened en cuenta que *level-1* podría ser PaBHT y *level-2* podría ser PHT.
- en sim-outorder.c:
  - Registrar en *sim_reg_options* los parámetros de configuración del predictor: con *opt_reg_int_list(opt,"-bpred:alloy",*..... Será necesario definir *predictor_config* y *predictor_nelt*.
  - Añadir en *sim_check_options* la lectura de los parámetros del predictor. Llamar a la función de creación del predictor *bpred_create*(,,,,,,,). Fijarse en el *2lev.*

- En bpred.c:
    - Añadir en *bpred_create* y *bpred_dir_create* un nuevo caso (*case*) de inicialización de predictores donde a partir de los parámetros definidos del predictor, se reserva la memoria necesaria y se inicializan las variables y tablas a los valores necesarios.
    - Añadir en *bpred_config* y *bpred_dir_config* un nuevo caso (*case*) para mostrar por la salida del simulador la configuración del predictor.
    - Añadir en *bpred_reg_stats* un nuevo caso (case) para poder ver el resultado de la simulación.
    - Añadir en bpred_lookup y bpred_dir_lookup un nuevo caso (case) para obtener la dirección de la siguiente instrucción, atendiendo a la predicción que realiza a partir de la instrucción de salto que se ha hecho el *fetch*. (siempre que se encuentre en el BTB). Aclaración: *bpred_dir_lookup* devuelve la dirección de la entrada de la tabla PHT (L2 Table) y *bpred_lookup* la guarda en el campo *pdir1* de la propia instrucción para que en la siguiente llamada a *bpred_update* ya tenga calculada la posición en PHT (es una optimización)
    - Añadir en bpred_update el caso para actualizar el GBHR, el PaBHT (L1 Table) y el PHT (L2 Table). Hay que tener en cuenta esta ultima tabla se puede actualizar a partir de la dirección guardada anteriormente (la optimización)

En general la simulación en simplescalar empieza en *main.c*. Para la funciones que hemos mencionado se llama a *sim_reg_options*(), después a *sim_check_options*() y después a *sim_main*(). Las tres están implementadas dentro del código principal del simulador sim-outorder y a partir de este instante empieza la simulación.

La función *sim_main* implementa el pipeline del superescalar. Llama a diferentes funciones para simular el comportamiento del procesador y de entre ellas nos interesa la función *ruu_fetch*().

ruu_fetch realiza la lectura de instrucciones de memoria, simula la cache, el TLB y el predictor de saltos. Al leer la instrucción la decodifica parcialmente y si se da cuenta de que es una instrucción de salto llama a *bpred_lookup* para que le devuelva la dirección de la siguiente instrucción que según su predicción que debe seguir buscando en memoria.

La función *bpred_update()* actualiza la información del predictor a partir de la ejecución real de la instrucción de salto. Se llama generalmente desde el *ruu_commit*() pero también se puede llamar antes desde el *ruu_writeback*() o incluso desde *ruu_dispatch*().

Una vez implementado el predictor se añadirá el comportamiento del mismo a las gráficas anteriores.

Los parámetros a tener en cuenta del nuevo predictor:

- **`Alloy`**`: opción -bpred alloy`
  - Tamaño del PaBHT <l1-size> y del PHT<l2-size> respectivamente: (8-8), (16-32), (32-128), (64-512), (128-2048), (64-4096) son (Y-X)
  - Anchos de GBHR <g> y PaBHT <p> que concatenados con 'i' forman 'c' serán: (1-1), (2-2), (3-2), (3-3), (4-4) y (4-4)
  - Así la configuración de c=i+g+p quedaría: (1+1+1->3), (1+2+2->5), (2+3+2->7), (3+3+3->9), (3+4+4->11), (4+4+4->12)
  - `-bpred:alloy  <Y>  <X>  <p> <g> 0`