Prediccion de saltos

Problema a resolver

- ¿Cual es la proxima instruccion a hacer fetch?
- Cuando hacemos fetch la instr anterior esta en la etapa de **DECODE**
- Por ende no se codifico => no sabemos si es un salto o no
- **Para fetchear la instruccion correcta debemos saber:**
  1. Si la instruccion anterior es un salto
  2. Si es un salto condicional, si es salto se toma o no
  3. En caso que se tome, cual es la direccion de salto.
- Esto lleva a una **Penaldiad**
  - Supongamos que por cada instruccion mal predicha tenemos una penalidad de 10 ciclos
  - Suponiendo que el fetch toma una sola instruccion por ciclo
  - TotalCiclosPrecision = CantInstr + (CantInstr - InstrPredicted) * Penalidad
  - Siendo asi si la precision fuera del 100% tendriamos la misma cantidad de cilos para la misma cantidad de instrucciones

Solucion -> Prediccion de saltos

- Se intenta predecir la siguiente instruccion a ejecutar
- No solo si es necesario sino a que direccion saltar

¿ Es posible evitar el salto ?

- Miremos este ejemplo

```
if (a == 5)
     b = 4;
else
     b = 3;
```

-

```
        cmp   x0, 5
        b.ne  L2
        mov   x1, 4
        b     L3
L2:
        mov   x1, 3
L3:
```

- ARM tiene un tipo de instrucciones para evitar estos saltos (B)

Conditional Instructions		
CCMN	rn, #i_5, #f_4, cc	if(cc) rn + i; else N:Z:C:V = f
CCMN	rn, rm, #f_4, cc	if(cc) rn + rm; else N:Z:C:V = f
CCMP	rn, #i_5, #f_4, cc	if(cc) rn − i; else N:Z:C:V = f
CCMP	rn, rm, #f_4, cc	if(cc) rn − rm; else N:Z:C:V = f
CINC	rd, rn, cc	if(cc) rd = rn + 1; else rd = rn
CINV	rd, rn, cc	if(cc) rd = ∼rn; else rd = rn
CNEG	rd, rn, cc	if(cc) rd = −rn; else rd = rn
CSEL	rd, rn, rm, cc	if(cc) rd = rn; else rd = rm
CSET	rd, cc	if(cc) rd = 1; else rd = 0
CSETM	rd, cc	if(cc) rd = ∼0; else rd = 0
CSINC	rd, rn, rm, cc	if(cc) rd = rn; else rd = rm + 1
CSINV	rd, rn, rm, cc	if(cc) rd = rn; else rd = ∼rm
CSNEG	rd, rn, rm, cc	if(cc) rd = rn; else rd = −rm

- `CSEL rd, rn, rm, cc // if(cc) rd = rn; else rd = rm` (Seleccion Condicional)
- Utiliza las flags del procesador para evaluar una condicion y asignar un valor u otro al registro

```
cmpi  x0, 5
mov   x0, 4
mov   x1, 3
csel  x1, x0, x1, eq
```

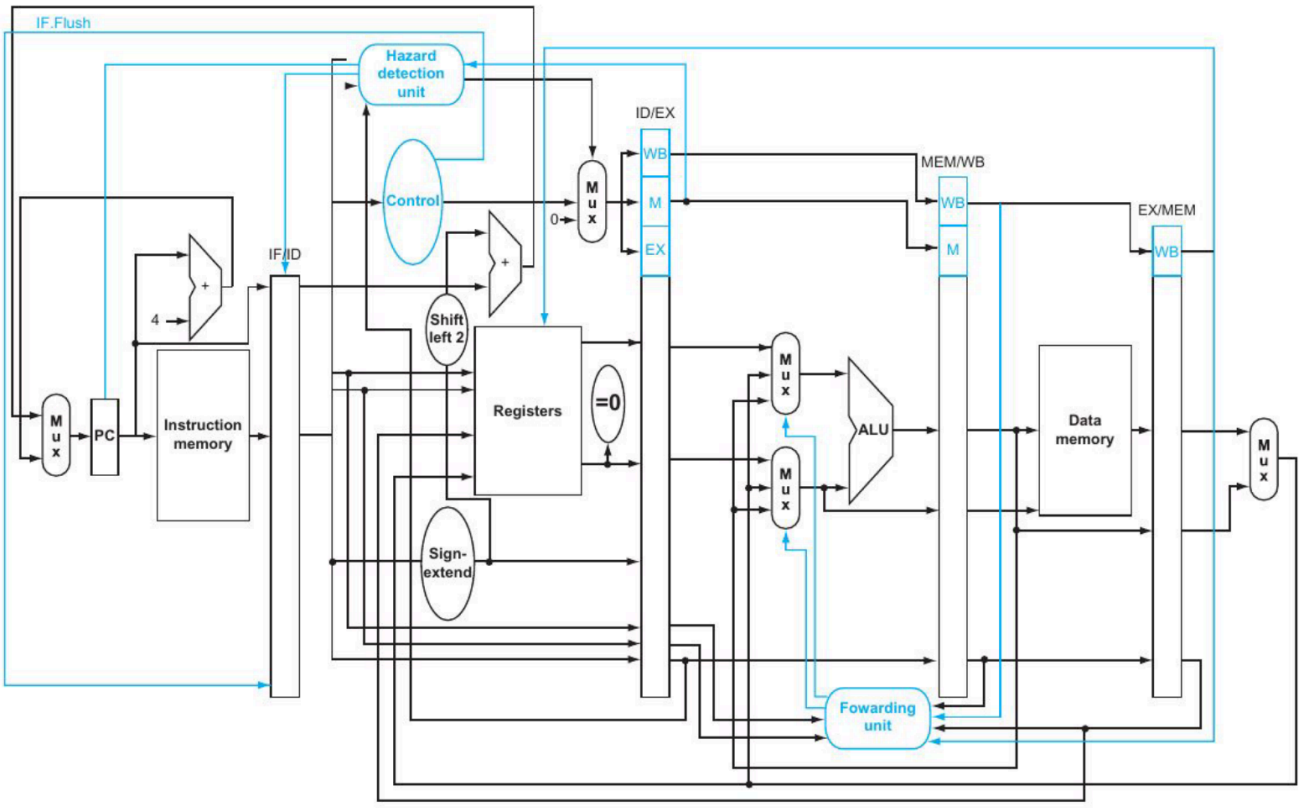
1. Compara x0 con 5 (setea la flag)
2. X0 = 4, X1 = 3
3. Dependiendo de la flag del cmpi si es == en x1 pongo x0 sino dejo x1

- Me ahorre 2 saltos que vimos en el codigo del inicio
- **De esto se encarga el compilador**

- Si el codigo fuera muy extenso (mas que una simple asignacion) en el if/else pues no me serviria de mucho (lab2 se hara un analisis como este)

Mejora al procesador (si no se puede evitar el salto)

(Micro mejorado Lindo pero COSTOSO)



- En la etapa de memory (que terminaba en memory) tiene la logica para determinar el salto y la direccion
- **Lo podemos mover a DECODE** esa logica
- Donde en la imagen vemos que se agrega ese dectector
  - Que si es =0 podemos hacer el CBZ(Compare and Branch if zero)
- Mejoramos la penalidad
- **Problema**
  - Alargamos el tiempo de la etapa debido a la nueva logica
  - Seguimos teniendo la penalidad si bad predict
  - Me sirve solamente para el CBZ
  - Decode va a ser un cuello de botella (> tiempo de ciclo para esta etapa)

Mejora al procesador v2(si no se puede evitar el salto)

- Volvemos a la forma original del micro
- Esta vez vamos a intentar predecir todo en la etapa de fetch

- Tenemos ns para hacer esa prediccion
- FETCH ->
  - Saber si es un salto
  - Si se va a tomar
  - La direccion
- ¿COMO? -> Como toda mejora de rendimiento siempre hay que poner memoria

Branch Target Buffer (BTB)

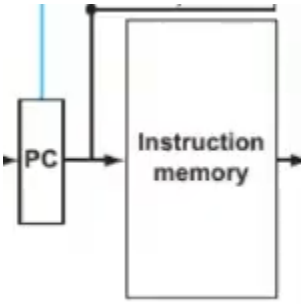
PC	Target

- Es una pequeña memoria que almacena la direccion donde se encuentra un salto y la direccion a donde debe saltar
- Forma 1: Es como una cahce asociativa donde el PC hace de tag (se hace ese chequeo en paralelo)
  - Es seguro
- Forma2 : Poner una memoria que sea de por ejemplo 8K (No uso todo el PC y uso los bits menos significativos para direccionar)
  - Desventaja podria haber 2 saltos
  - Que casualmente los bits menos significativos sean iguales y los mas significativos sean distintos
  - Me equivoco en la direccion que salto

Proceso

1. Se fetchea un salto
2. Se guarda su PC y su direccion a saltar
3. Si se vuelve a usar el mismo salto ya se tiene guardado en esta memoria el dato (cache)

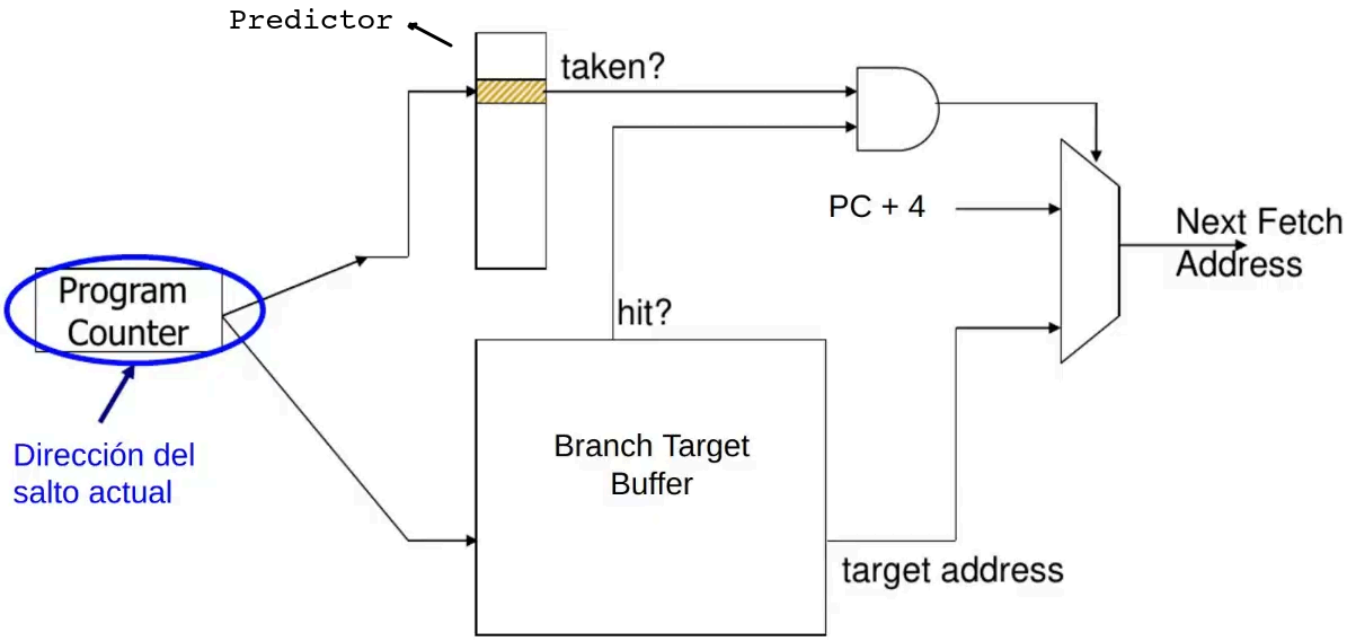
Ejemplo



- !Entro salto! guardamos y predecimos (Taken o NotTaken)
- 0x0040 CBZ X0, #-10
- Si X0 es 0 saltamos 10 instrucciones para atras (0x0040 - 10)

PC	Target
0x0040	0x0030

Prediccion de direccion de salto



---

1. Viene el PC
    1. Predictor
        - Revisamos si es Taken o No Taken
    2. BTB
        - Voy al BTB
        - ¿HIT?
        - Saco la direccion a la que tendria que saltar
2. Si hay HIT y es Taken paso la direccion de salto de la cache
3. Si no PC + 4

- Otra forma de implementarlo es en el predictor tener el BTB adentro

Tipos de salto

Tipo de salto	¿Sabemos la dirección al hacer fetch?	Número de posibles direcciones de salto	En qué etapa se resuelve la dirección del salto
Condicional (CBZ, CBNZ, B.cond)	No (PC + offset o PC + 4)	2	Mem o Dec (ver mod)
Incondicional - B	Si (PC + offset)	1	Dec (PC + offset)
Salto con registro - BR	No (PC = Reg)	Muchos	Dec

- Conditional Branch
    - Dos opciones taken o no taken
- Branch
    - Una sola posible direccion
- BR - El mas problematico

- No saltas al mismo lugar, depende del valor que hay en el registro
- Puede pasar que la direccion que dio el BTB, sea incorrecta debido al valor del registro que se modifico

Prediccion del saltos

Con el BTB ya resolvimos a que direccion tenemos que saltar
Ahora hay que tratar de predecir el salto con buena precision

Estaticos (Aquellos en donde siempre tomo la misma decision para cualquier tipo de salto)

- **Not Taken**
- **Taken**
  - Ventajas
    - Implementacion mas sencilla (nuestro micro)
    - No necesita predecir
    - Si acierta no tiene penalidad
  - Desventajas
    - Baja preicision (30-40% para saltos condicionales)
    - Penalidad de 3 ciclos
- **BTFNT** (Backward Taken, forward not taken) (No lo veremos mucho)
  -
    - probable como taken - probable como not taken

Dinamicos (Para un mismo salto se pueden predecir cosas distintas) + Memoria

- A medida que una instruccion de saltos es ejecutada se almacena info sobre el resultado
- Esta informacion se utiliza luego para intentar predecir el resultado de ejecuciones posteriores

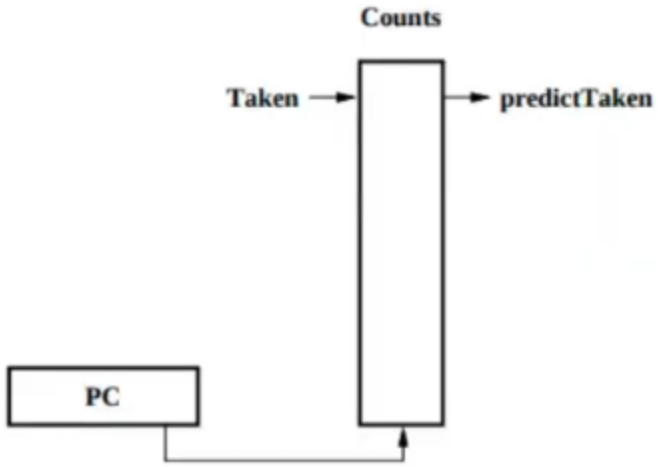
Tipos de predictores dinamicos:

Predictores locales - Basan su prediccion en ejecuciones anteriores del mismo salto

- Chequeamos el mismo salto
- **Diseño**
  - Ponemos una memoria con un ancho de n bits (para lo que queremos 2) (BHT)
  - Aca guardamos la prediccion de salto
  - Indexamos esa memoria con los n bits menos significativos del PC
  - Vemos que prediccion hay y la actualizamos dependiendo si el salto local es un Taken o Not taken

Predictor de 2 bits (local)

- **Usar 2 bits**
- PC del salto anterior bits menos significativos -> Indexan esa memoria (Hash Table) Branch History Taken (BHT)
  - Que esta memoria sea muy grande grande no garantiza mejor precision



-
- y 2 bits de ancho para representar los 4 estados
- Puede pasar que un salto comparta con otro salto los bits menos significativos
- Entonces tenemos 4 estados (4 valores posibles)

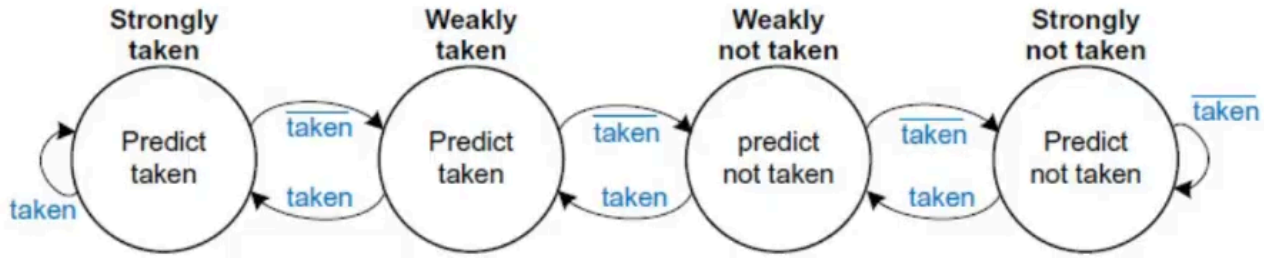
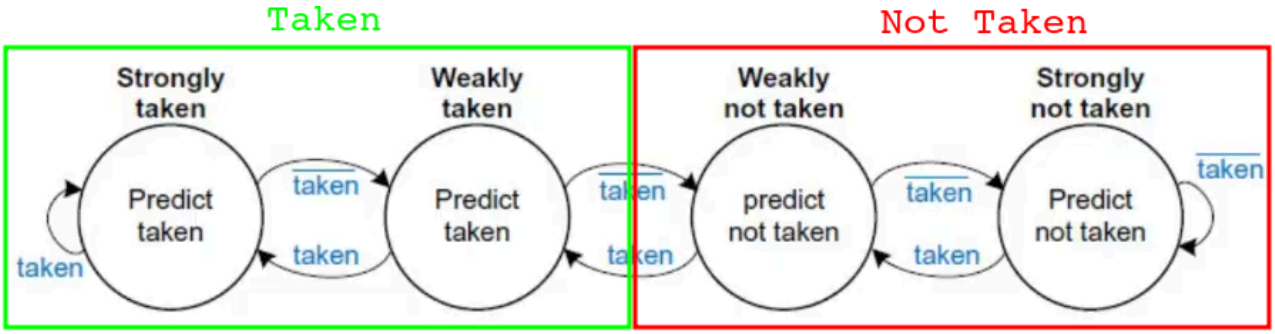


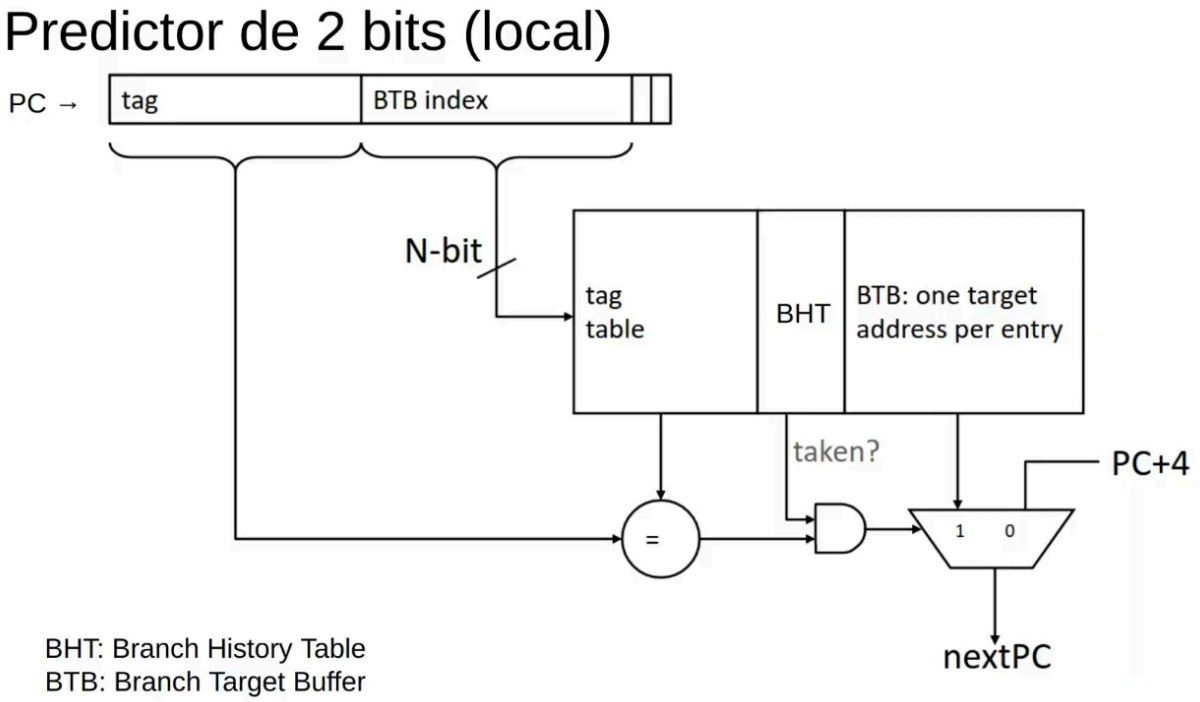
Figure 7.62 Two-bit branch predictor state transition diagram

-


- **Ejemplo** - Empezando en el estado Not Taken
  - Predicted NT pero T (Fallo) => Seteo Weakly Not taken
  - Predicted NT pero NT (Acierto) => Seteo Strongly Not Taken
  - Predicted NT pero T (Fallo) => Seteo Weakly Not Taken

- Predicted NT pero T (Fallo) => Seteo Weakly Taken
- Predicted T pero NT (Fallo) => Seteo Weakly Not Taken
- O sea -> (NT -> NT -> NT -> NT -> T)

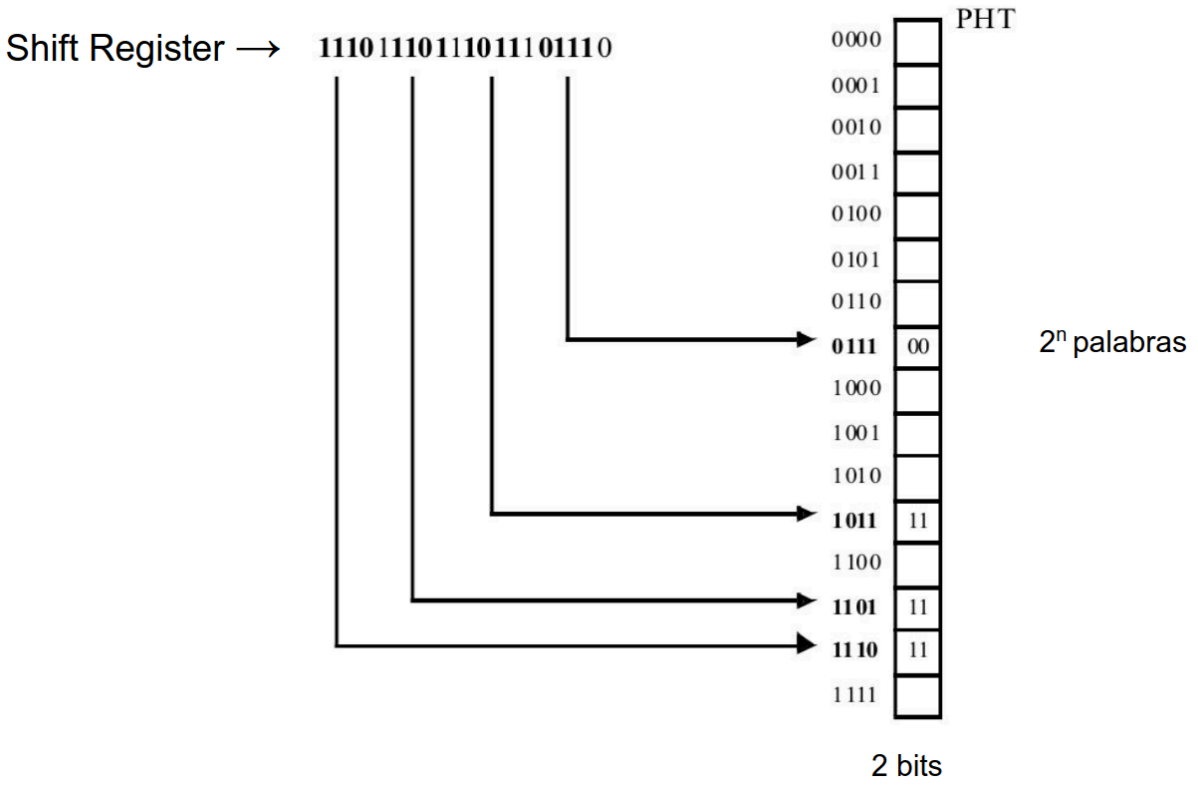
- **Implementacion**



- PC -> BMASS como tag y BMENOSS como index para la BTB
  1. Entro con el PC del salto
  2. Si el index de la BTB me da que el Tag = Tag original del PC
  3. y si en la BHT (4 estados) da que es taken
  4. => Damos la direccion de la BTB para saltar
  5. Otherwise PC + 4

Predictores globales - Basan su prediccion saltos anteriores del codigo (no solo el mismo)

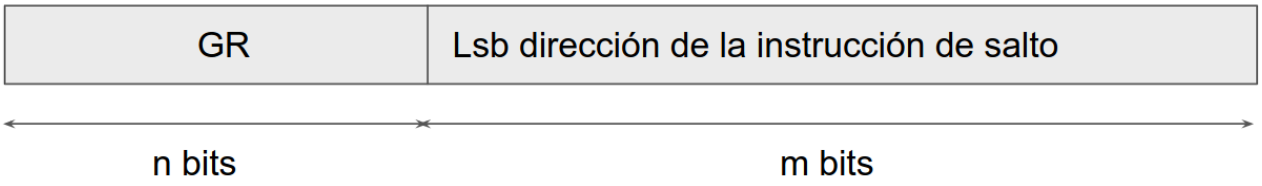
- Necesito alguna forma de almacenar el patron de los saltos anteriores
- **Diseño**
  - Se crea un **shift register** o **Global register** de **n bits** donde se almacenan los resultados de los ultimos **n saltos**
  - **Donde cada bit del shift register representa un salto** (1 TAKEN | 0 NOT TAKEN)
    - Cada salto entra por la derecha del **GR**
  - La prediccion de los ultimos n saltos se guarda en PHT - pattern History Table de 2^n palabras de dos bits (si tengo 10 bits de **GR** => 2^{10} palabras en PHT)



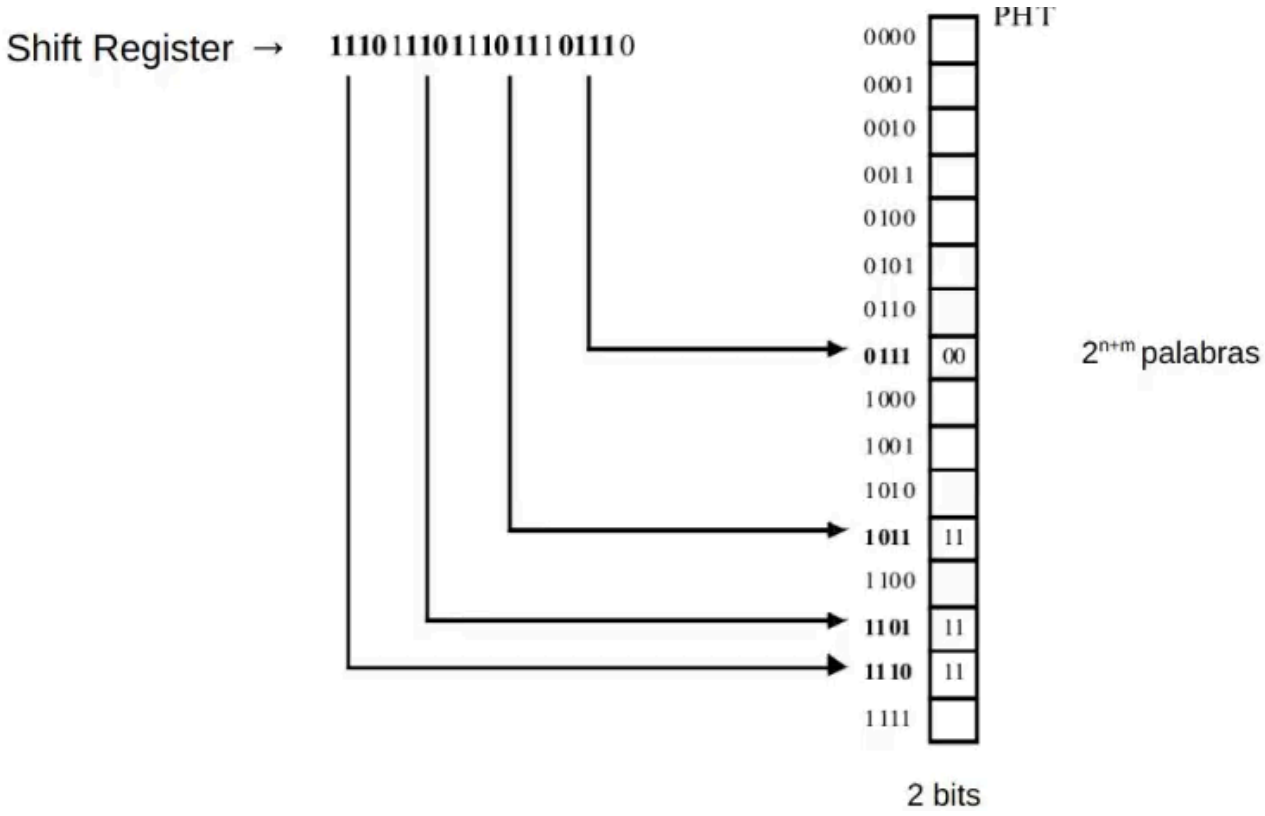
- 4 bits de shift register (se muestran los bits empujados tambien)
- Se utilizan 4 bits para indexar a la PHT (ultimos 4 saltos) porque la PHT es de 2^4 palabras de 2 bits
- Ejemplo 4 BMS **1110** con eso indexamos el PHT para acceder al resultado guardado (que veremos como se setea)

Predictor de 2 niveles (Local- utilizando el diseño del global)

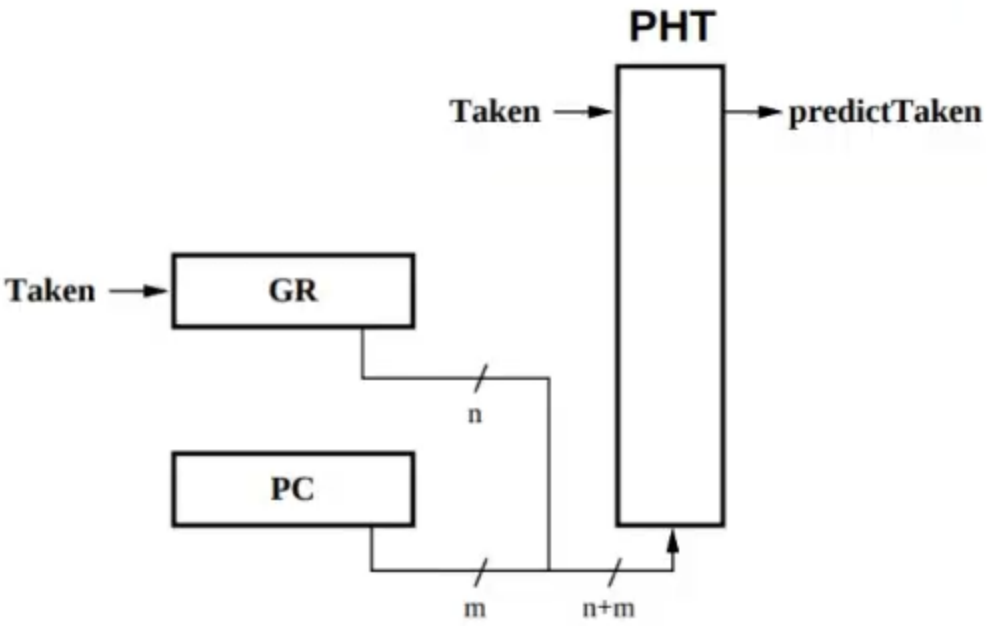
- Con el diseño del predictor global lo podemos convertir en Local utilizando el PC
- Si concatenamos el registro GR con los bits menos significativos del PC se obtiene un predictor de dos niveles local
- **De esta forma asocio un determinado patron a un salto en particular**



- O sea tenemos los bits del {**Global register**, los bits menos significativos del PC}



-
- Asi teniendo un PHT de 2^(n+m) palabras

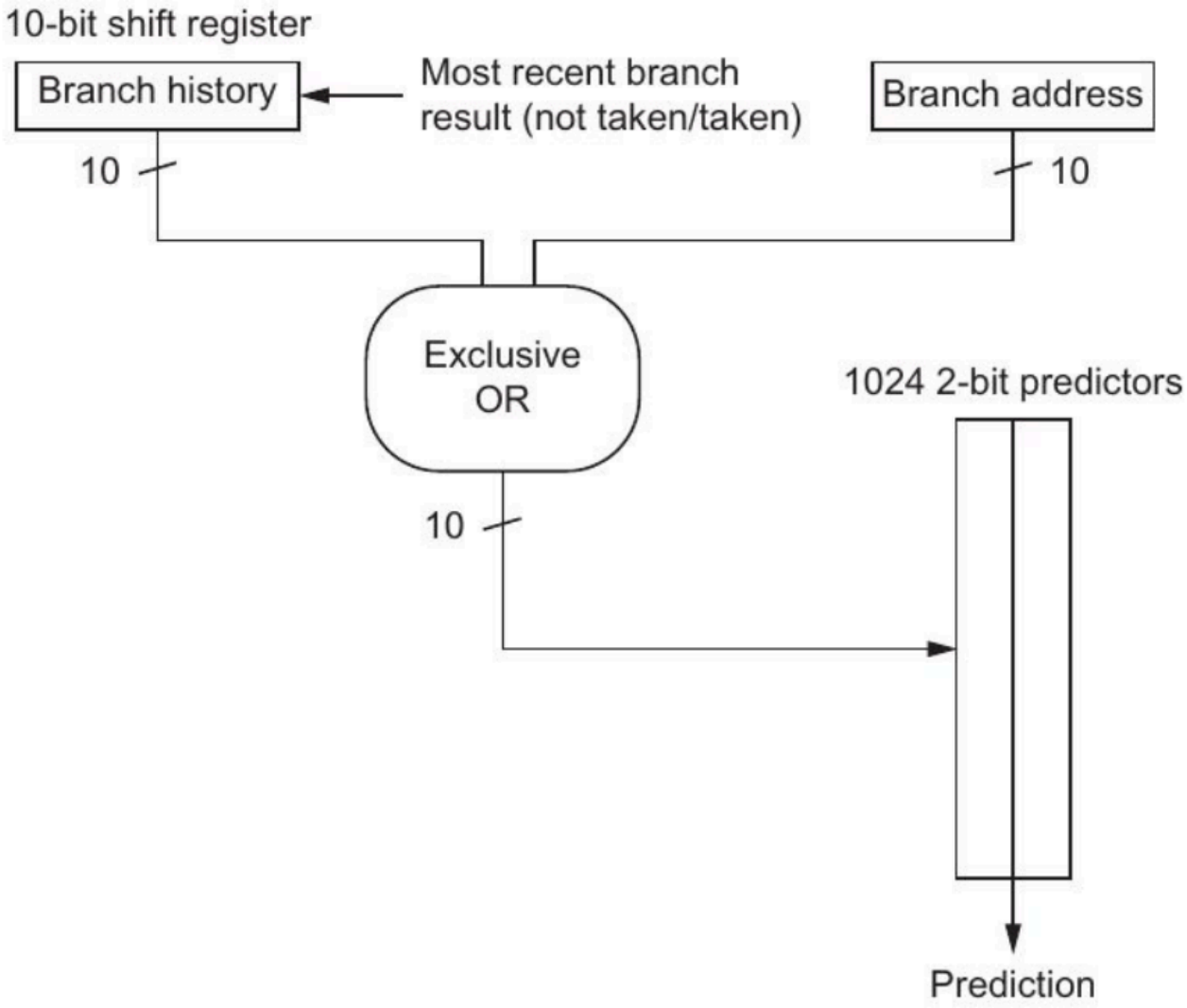


-
- Entonces indexamos la PHT con el GR (La info de los saltos anteriores) y el bits menos significativos del PC n+m
- Tenias un salto 0x0040 y se da que el global register fue 1111
  => PHT[0x111140]

Predictor gshared (Mejora del de 2 niveles local)

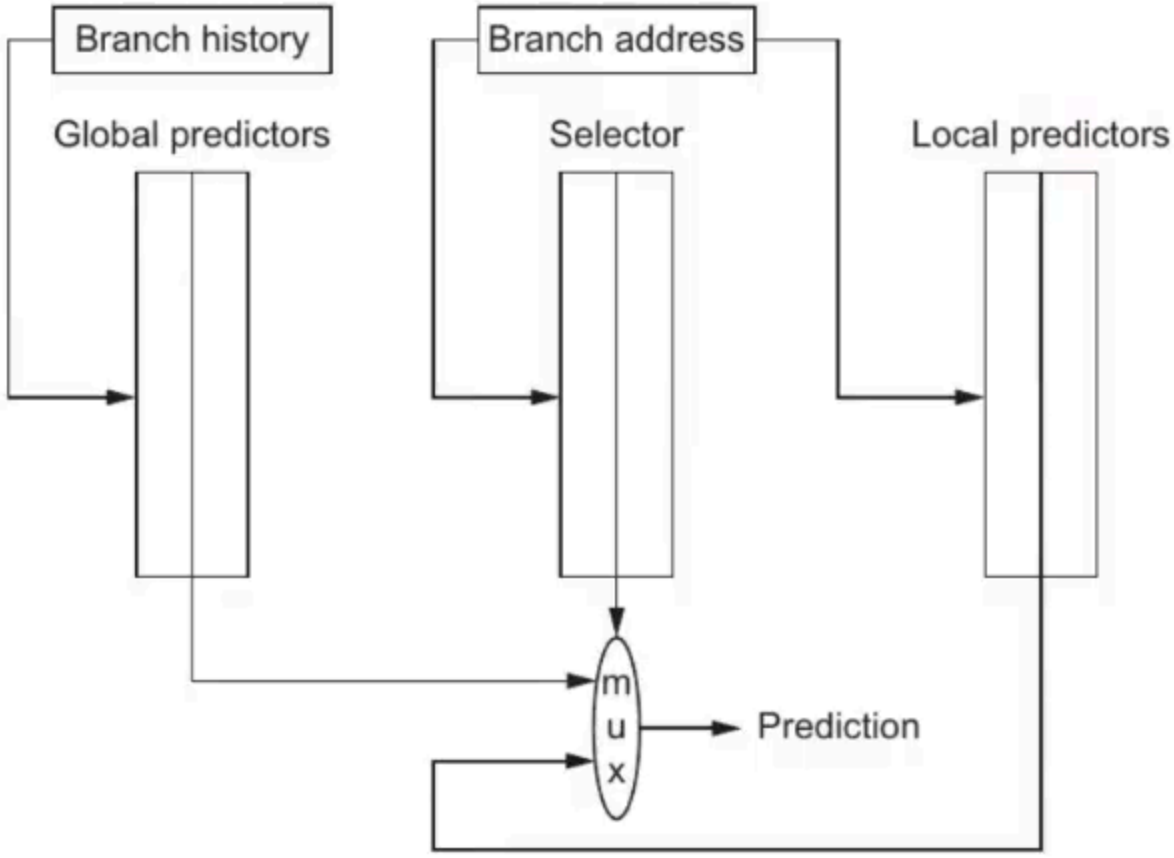
- En lugar de hacer una concatenacion GR ++ BmenosSPC

- Hacemos una XOR (hash)
  - Entre los ultimos 10 bits de la direccion de instruccion de salto PC
  - y el GR/SR (Global register)
- Para indexar a la PHT



Predictores por torneo

- Se combina predictores locales y globales
  - **Predictor de 2 niveles**
- La ventaja es que elige el tipo de predictor que mejor funciona para cada salto
- Se tiene un selector de predicciones que es una memoria
  - Se registra cual de los predictores funciona mejor para cada uno de los saltos



-
  - El global predictor se accede con el **GR**
  - Al local solamente con el **PC**
  - Al selector tambien con el **PC**
    - 1 bit que decide el mux si usar el local o el global

Resumen

BTB -> Branch Target Buffer

- Por cada fetch de un salto se guarda en una memoria su direccion a saltar
- BTB[PC] = Target

Predictores Locales

- Basan su decision chequeando resultados anteriores del mismo salto
- y utilizan una memoria (BHT) para indexar la info del mismo salto con los Bits menos significativos del PC de ese salto
- Tambien puede utilizarse el diseño del global para locales

Predictores Globales

- Basan su decision chequeando resultados de n saltos anteriores
- Se utiliza un **Shift/Global Register** de n bits para representar los n ultimos saltos
- 0 -> Not Taken y 1 -> Taken
- Utiliza una memoria **(PHT -> Pattern History Table)**
- que indexa con los n bits del **Global Register** para ver la prediccion de esos ultimos n saltos
  - (que lo mas bien puede pasar que sean n ejecuciones del mismo salto)
- PHT[GR] = Prediccion

Predictor de 2 bits (Local)

- Se una una memoria (**BHT -> Branch History Taken**) que sirve para obtener la prediccion del salto
- Indexada por los bits menos significativos del PC
- BHT[BmenosSPC] = Prediction
- Tenemos 2 bits para representar los estados ST - WT - WNT - SNT (T - T - NT - NT)
- **Proceso**
  1. O sea entra el PC del salto actual
  2. Se fija si esta en el BTB
  3. Se fija en el predictor en la BHT si es taken
  4. En caso de hit en la BTB y de taken => Pasamos el target como siguiente instruccion
  5. Sino PC + 4

Predictor de dos niveles (Local)

- Con el diseño de los predictores globales podemos convertirlo en local
- En vez de indexar con solo los n bits del **GR**
- Le concatenamos los m bits menos significativos del PC del salto en cuestion
- PHT[n+m] = prediction (00 01 10 11) (SNT WNT WT ST)
- Siendo asi local
- **Proceso**
  1. Entra la instruccion de salto
  2. Vemos si el predict fue correcto en la PHT
  3. Actualizamos el **GR** y el PHT[GR+PC] para usarla en el siguiente salto que venga

Predictor gshared (Local)

- Es lo mismo que el anterior pero en vez de concatenar
- Usamos una xor entre **GR** y los 10 bits menos significativos del PC
- Que es otra forma de hashear para el mismo salto

- PHT[XOR(GR,PC)] = Prediction