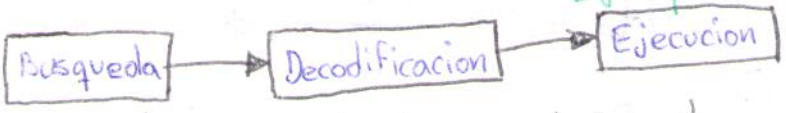
Unidad 4.- Segmentacion de la ruta de datos de un Procesador Risc.

En computacion, la segmentacion es un procesamiento de un conjunto de datos conectado en serie; Esto significa que la salida de un elemento es la ~~salida~~ entrada de otro.

Los elementos de un arreglo pipeline son ejecutados en paralelo o en tiempos definidos.

Ejemplo Basico de segmentacion



Es así, como a la segmentacion se le puede usar para:

→ Segmentacion de Instruccion: Muy usado en la arquitectura RISC, usado en uno o varios microprocesadores para una superposicion de multiples instrucciones a ejecutar con la misma circuiteria.

La circuiteria esta dividida en varias etapas y cada una de estas, procesa una parte de la instruccion en cierto tiempo; pasando el resultado de esta etapa a la siguiente etapa.

Ejemplos: Decodificacion de una instruccion, operacion aritmetica-logica y busqueda del registro

→ Segmentacion de graficos: Usado en procesadores graficos (GPUs), donde las operaciones tienen multiples unidades aritmeticas, o un microprocesador completo.

→ Segmentacion de Software: Consiste en la secuencia de procesos de computo como comandos, programas, ejecutador, tareas, conexiones y procedimientos. Estos procesos pueden ser ejecutados en paralelo con una salida de flujo de datos.

→ Segmentacion HTTP: Una tecnica de emision de multiples peticiones HTTP con una misma conexion ~~TCP~~ TCP, sin esperar a una peticion previa para generar una nueva peticion.

4.1 Segmentacion.

La segmentacion es la base fundamental de la parelizacion y este Concepto es ampliamente usado en arquitecturas vectoriales y superescalares.

4.1.1. Etapas de segmentacion.

Para entender las etapas de la segmentacion, es necesario usar un ejemplo de segmentacion basica con instrucciones, que permiten implementar el paralelismo a nivel de instruccion en un solo procesador.

Ejemplo: Estas son las etapas de ~~ejecucion~~ Procesamiento de una instruccion,

- Lectura de instruccion: LI
- Decodificacion de instruccion: DI
- Ejecucion: Ex
- Acceso a memoria: MEM
- Escritura y regreso al registro: ES

Así es como se empieza a segmentar la instruccion.

Inst \ Ciclo de R.	1	2	3	4	5	6	7
1	LI	DI	Ex	MEM	ES		
2		LI	DI	Ex	MEM	ES	
3			LI	DI	Ex	MEM	ES
4				LI	DI	Ex	MEM
5					LI	DI	Ex

En la columna verde se puede observar esta etapa MEM de la primera instruccion, mientras que la instruccion 5 aun no esta en la linea de ejecucion.

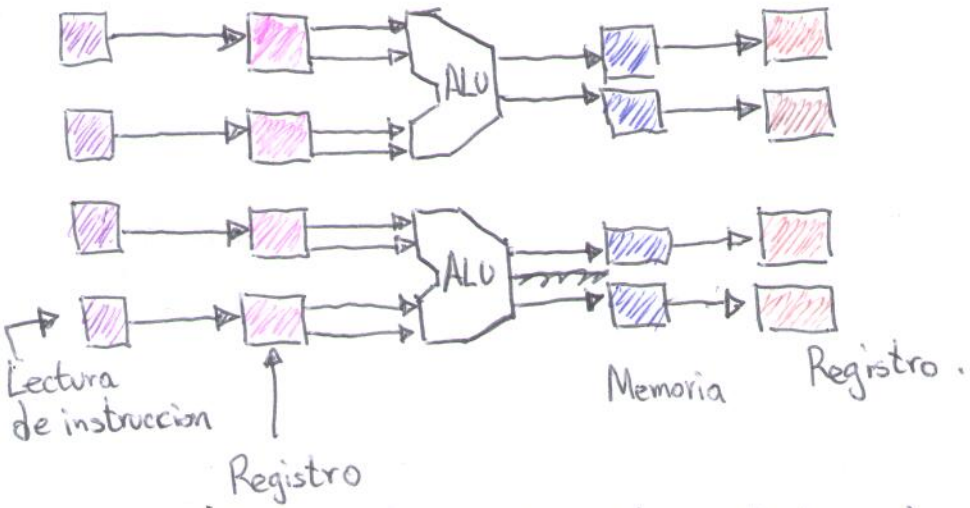
4.1.2 Registros inter etapa.

Los registros inter etapa son muy usados en arquitecturas avanzadas, como la vectorial y la super escalar. En estas arquitecturas se potencializan el paralelismo de la ejecución de las instrucciones.

Debido a este nivel de paralelismo requiere algun metodo anti-translape, es asi que se tiene un procesamiento super-escalar.

Las instrucciones tienden a ser grandes y el procesamiento de las instrucciones pueden ejecutarse en un mismo ciclo de reloj (maximo 4 instrucciones sin un metodo complejo de anti translape).

Ejemplo:



Si alguna de las etapas toma (o puede tomar) mucho mas tiempo de procesamiento que otras, esta no puede ser acelerada, pero las otras si pueden ser ralentizadas. Este concepto de paralelización requiere de registros que actuen como buffers, para poder ejecutar las instrucciones de forma paralela.

Idealmente, la ejecucion de las instrucciones de forma paralela, son ejecutadas al mismo tiempo, por lo que el procesamiento de los elementos esta sincronizada.

Pero la ejecucion de las instrucciones es irregular, por lo que los registros inter etapa ayudan a regularizar el tiempo de ejecucion.

4.2 Riesgos (Hazards)

El modelo de ejecucion secuencial "asume" que cada instruccion se completa antes de que comienze la siguiente:

Lamentablemente, esto no es posible en un procesador segmentado. Esta situacion representa un peligro, porque 2 instrucciones actuan sobre los registros del procesador.

~~Solucion:~~

4.21 Riesgo estructural

Este es el primer riesgo que se tiene que evitar, normalmente esta relacionado con el hardware, porque este ultimo no puede soportar una combinacion de instrucciones que se tienen que ejecutar en el mismo pulso de reloj.

El riesgo estructural seria como un cuarto de lavado y se quiere usar la lavadora y secadora al mismo tiempo.

Las instrucciones risc y Mips. fueron diseñadas para la segmentacion, marcando facilmente los potenciales riesgos estructurales.

Para entender esto, se usara nuevamente la tabla de ejecucion de instrucciones.

Ciclos / Inst	1	2	3	4	5	6	7
1	LI	DI	EX	MEM	WR		
2		LI	DI	EX	MEM	WR	
3			LI	DI	EX	MEM	WR
4				LI	DI	EX	MEM
5					LI	DI	EX

← Esta instruccion esta en la etapa de acceso a memoria.

← Esta instruccion Esta en la etapa de lectura, por lo que No es posible realizar la lectura

4.2.2 Riesgo de Datos.

Este tipo de riesgos ocurre cuando la segmentacion esta ~~d~~bloqueada Porque un paso debe esperar a otro para completarse la instruccion.

Esto puede entenderse mejor con la siguiente analogia; Supongase que una persona debe ir a una conferencia, y antes de salir descubre que solo tiene un zapato...

Para solucionar el problema, se tiene que buscar el zapato en toda la casa, y por lo consiguiente se detiene la conferencia.

En la segmentacion de procesos computacionales, los riesgos de datos crecen cuando la dependencia de una instruccion esta vinculada a la instruccion previa, y que posiblemente aun no termina la ejecucion de esta.

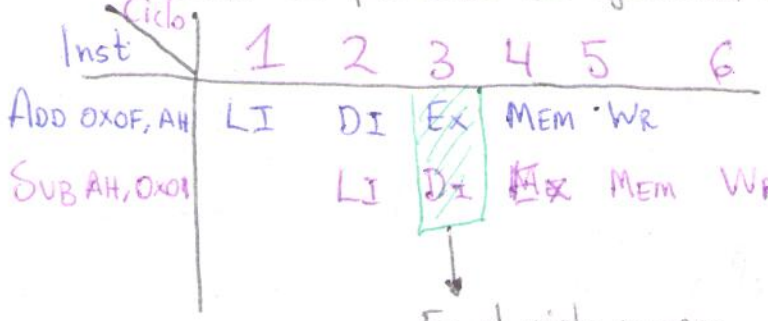
Ejemplo: Se tiene las siguientes instrucciones...

ADD 0x0F, AH ---> ~~Resta~~ Suma 0x0F a AH y el resultado se guarda en AH

SUB AH, 0x01 ---> Resta a AH el valor 0x01h y el resultado se guarda en AH

Sin una prevencion de riesgos de datos, el segmento puede quedarse detenido con consecuencias a todo el sistema.

Graficando el proceso de ejecucion de instrucciones, esta quedaria así

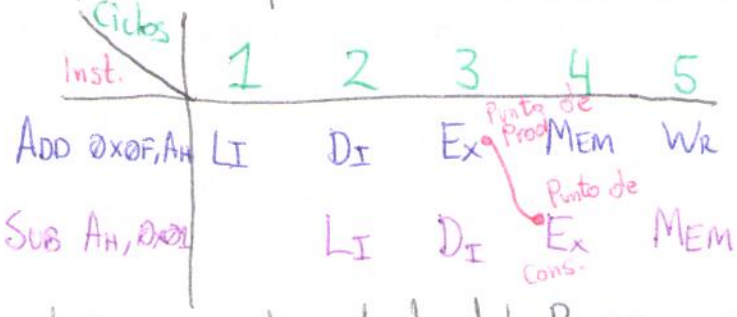


En el ciclo numero 3 es donde ocurre el Riesgo.

Para solucionarlo, se tiene que recurrir a diversos metodos.

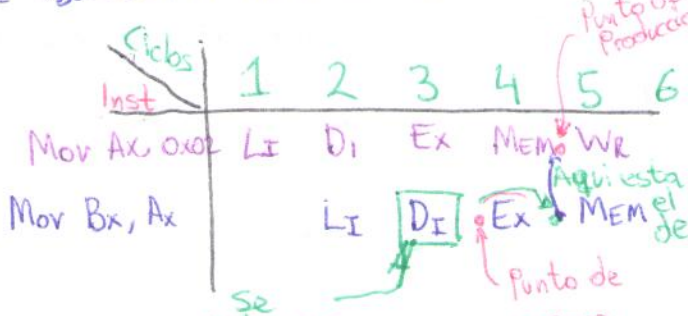
4.2.3 Metodo de bypassing.

Este metodo esta enfocado a los riesgos de datos, para entenderlo mejor es necesario aplicarlo al problema mostrado en la seccion de riesgos de datos.



La solucion por el metodo del Bypass es vincular la necesidad o requerimiento del dato, con la generacion del dato, a estos puntos se les conoce como punto de produccion (donde se crea el dato) y el punto de consumo (donde se requiere el ~~dato de consumo~~ generado).

En el ejemplo anterior las instrucciones que requieren el metodo, pero tambien se puede aplicar con cualquiera de los pasos mencionados, para la ejecucion de ~~instruccion~~ instruccion.



- En este ejemplo se tiene el Punto de produccion despues del Punto de consumo.
- Como el punto de produccion esta antes de la escritura de registro, el punto de consumo no puede usarlo.

Aqui no se puede hacer un bypass como en el caso anterior, Porque No se puede regresar los ciclos.

La solucion es detener el bloque o proceso anterior para dar tiempo a la escritura del registro a realizarse, y asi se puede hacer el bypass.

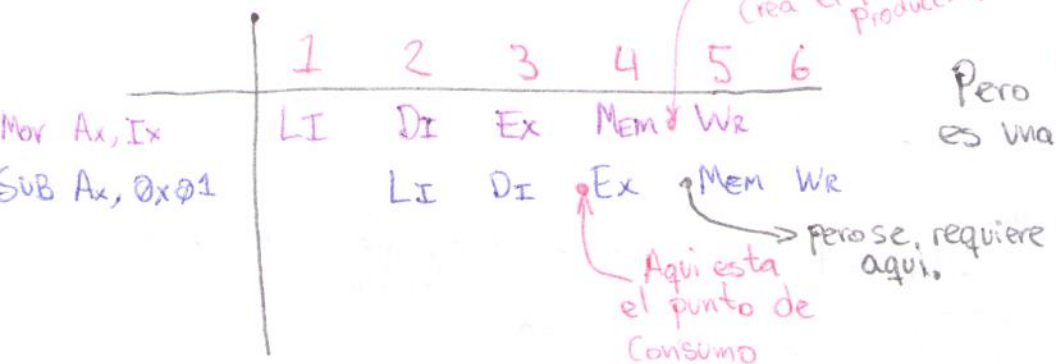
4.2.4 Metodo de forwarding.

Algunos autores manejan el metodo como forwarding como el de bypassing. En cierta medida se parecen, pero van a tener una variacion dependiendo el caso.

Para los casos anteriormente mostrados en el metodo de bypassing. Ambos metodos son iguales y se puede considerar el mismo.

En donde se puede considerar diferente en el tercer caso Cuando ocurre la detencion de la segmentacion, y es porque los metodos no son 100% confiable.

Supongase que la primera instruccion un valor en lugar de hacer una Suma,



Pero la otra instruccion es una resta

En este caso, toda la segmentacion se considera detenida.

Para solucionarlo, se requiere si o si, detener la instruccion mas adelantada para realizar un redireccionamiento del Punto de producción al punto de consumo.

Estos son los metodos mas usados, para los casos mas comunes. Pero existen otros casos donde los riesgos son diferentes.

La mencion del forwarding viene de la idea que el resultado debe ser re-dirigido desde una instruccion temprana a una instruccion tardia. El bypassing viene de pasar el resultado por el archivo de registros a la unidad deseada.

4.3 Riesgo de control.

En las anteriores explicaciones de riesgos, se observo que una instruccion podia detener el correcto procesamiento de las otras; es aqui cuando uno se pregunta si la ejecucion de una instruccion puede afectar otras en ejecucion.

Este tipo de riesgo se le conoce como riesgo de control.

Para entender este riesgo se debe usar la siguiente analogia:

Supongase que en una planta de ensamblado, los productos de salida deben tener la suficiente calidad para que el producto pueda pasar las normas de calidad; se tiene que examinar el tipo de tornillos, el torque con el que fueron apretados, el exterior que tenga la dureza y calidad. Pero una segunda etapa debe probar el funcionamiento del producto, si se requiere realizar algun ajuste en la linea de ensamblado o no. ¿Que se tiene que hacer?

~~Hay 3 posibles soluciones...~~

~~4.3.1 Predicciones de Salto~~

Para entender la prediccion de salto, se tiene que regresar al ejemplo anterior; La Planta opera secuencialmente hasta que el primer lote esta ensamblado, y se repite hasta que la calidad del producto sea la deseada.

Esto funciona, pero es bastante lento.

En un sistema de computo, el equivalente de esta tarea es una instruccion de Salto. Esto ultimo ocurre cuando la busqueda y lectura de la instruccion, es sucedida por un salto muy cerca del siguiente cicdo de reloj.

Sin embargo, en la segmentacion No es posible saber cual es la siguiente instruccion, ya que solo se recibe la instruccion de salto desde la memoria.

Asumiendo que se agregue suficiente hardware, y asi se puedan revisar los registros para calcular la direccion de salto, solo se hace mas grande la circuiteria pero no se soluciona el Problema.

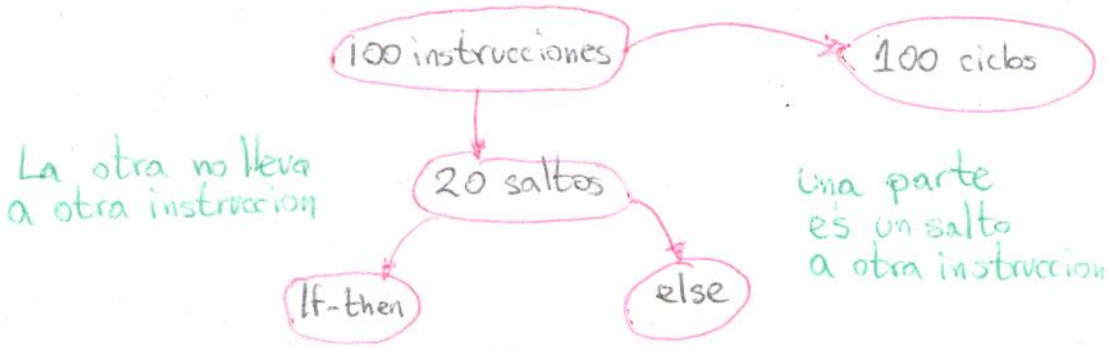
Para entender los riesgos de control, se presenta el siguiente ejemplo:

Supongase que se tiene 100 instrucciones a ejecutar, en condiciones ideales esas 100 instrucciones se tienen que ejecutar en 100 ciclos...

Sin embargo, de esas 100 instrucciones, 20 son saltos ademas de la posibilidad que en esos saltos hay condicionales if-then-else.
De esos casos if-then-else, 12 corresponden al then y 8 al else.

De la forma que esta escrito el programa; el if-then viene justo despues del salto, y si el salto indica otra instruccion (registrado en el contador de Programa. Se puede argumentar que en la ramificacion de saltos hay una parte que no lleva a ninguna instruccion, y otra que si lleva a otra instruccion.

Ramificacion.



Si una instruccion de salto tienen varias instrucciones despues.

Bra else

inst { instruccion 1, instruccion 2, ..., instruccion n

else

inst { instruccion 1, instruccion 2, ..., instruccion n

Hay muchas estrategias y tecnicas para lidear con estas ramificaciones:

- Prediccion de salto
- Prediccion estatica
- Prediccion dinamica.

} estos son las mas usadas

Por lo tanto el programa puede terminar en 120 ciclos.
O mas.

4.3.1 Predicciones de ~~salto~~ estatica. (salto.)

Considerando que no se puede resolver la ramificacion en un segundo plano, como es comunmente usado en casos de segmentacion largas.

Entonces, al hacer una detencion en las ramificacion incrementa el costo computacional a niveles que hacen inutil el sistema de computo.

Solucion:

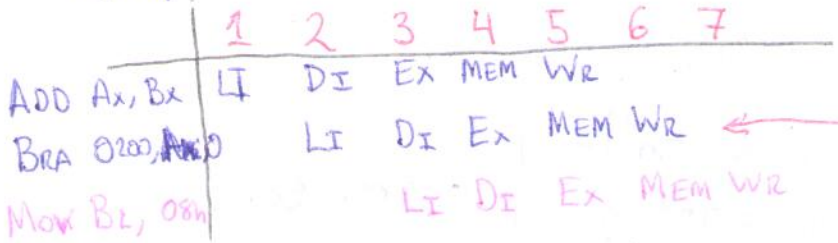
Predecir: Si se esta seguro de la ejecucion en el sistema de produccion en la planta de ensamblado (ejemplo mencionado anteriormente), entonces, solo queda predecir que el primer lote esta bien y empezar con la produccion del siguiente lote, mientras el primer lote esta siendo inspeccionado.

> Esta opcion no detiene la segmentacion (linea de produccion) mientras este correcta, y cuando existan errores el primer lote entra a la linea de produccion nuevamente mientras se adivina (predice) la decision del segundo lote.

Las computadoras pueden usar la "prediccion", para manejar la ramificacion producida por los saltos. Una simple aproximacion es usada para predecir que ramamificacion NO SERA USADA.

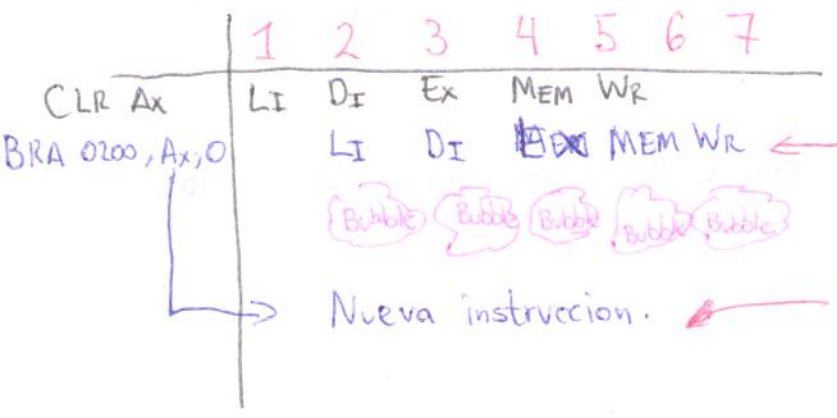
> Si la prediccion es correcta, la segmentacion funciona a toda velocidad, y unicamente cuando los saltos son efectuados, la segmentacion se detiene.

Ejemplo:



Si la prediccion es correcta.

Por la instruccion, esta misma se omite debido a los valores usados.



Si la prediccion no es correcta

La instruccion hace un salto

Solo toma un ciclo mas para ejecutarse.

4.3.2 Prediccion ~~Estatica~~ de Salto (desicion retardada)

Es la prediccion mas simple y tambien considerada la mas riesgosa; tambien conocida como una decision retardada.

Ahora usando la analogia de una lavanderia, se toma un pedido de un equipo de futbol para lavar y secar. Omitiendo la revision de los uniformes (uniformes) despues del lavado, y listos para secar...

Hay una peticion de lavado para ropa que no son los uniformes, si no hay una carga considerable de ropa para lavar, esta opcion de no revisar los uniformes, y lavar la ropa que no son los uniformes esta bien.

Volviendo a un ejemplo de computacion...

Supongase que se tienen una instruccion de salto, en el set de instrucciones de la arquitectura:

BRA [Condicion] ~~then else~~ — Si la condicion no se cumple, salta a las instrucciones donde esta el ~~s~~else

instrucciones ← Estas instrucciones se cumplen si la condicion se cumple

Else

instrucciones ← Estas instrucciones se cumplen si la condicion no se cumple.

En el metodo anterior solo despues del tercer ciclo, cuando la instruccion de salto ha sido ejecutada, es cuando comienza a ejecutarse la siguiente instruccion.

En este metodo depende del compilador introducir un retardo.

Ejemplo:

Antes del salto

~~ADD~~ AL, 00h

if AL = 00h then — La instruccion salta aqui

[Espacio de R.]

Como la condicion de salto depende de una instruccion previa...

Durante el Salto

if AL = 00h then

[ADD AL, 00h]

En este espacio se ejecuta la instruccion anterior a la condicion de salto.

Desde la condicional.

ADD AL, 00h

if BL = 00h then → SUB Cx, Ax — La instruccion salta aqui

[Esp. de Retardo]

Aqui no depende de la condicion previa

Durante el salto

ADD AL, 00h

if BL = 00h then

[SUB Cx, Ax]

Ahora se ejecuta la instruccion objetivo.

Esperando que Cx No sea un valor que se use despues de la condicion de salto

4.3.3 Prediccion Dinamica.

Una forma simple de prediccion de salto, es asumir que una eleccion no es tomada. Para este caso las elecciones que no son ~~tomadas~~ tomadas, estas son descartadas de la segmentacion (mas si la prediccion fue incorrecta).

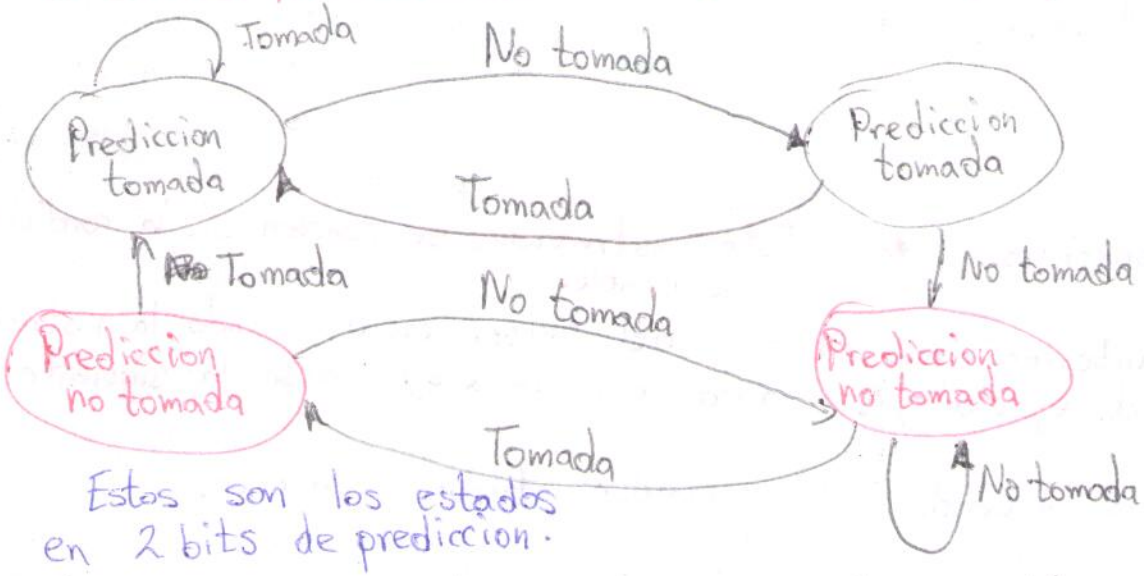
Una aproximacion ~~atentadora~~, es mirar la direccion de la instruccion para ver si ese salto fue tomado la ultima vez que la instruccion de salto fue tomada, de ser afirmativa esta suposicion, la lectura inicia con las nuevas instrucciones (las que no fueron usadas en el ciclo anterior), desde el mismo lugar que el salto fue usado la ultima vez

A esta tecnica se le llama prediccion dinamica.

Ejemplo:

Imagine un salto en un ciclo cerrado (loop), que se ejecuta 9 veces en una sentencia de instruccion, despues no se vuelve a tomar nuevamente.

¿Que exactitud tendra la prediccion de este salto, asumiendo que el bit de prediccion se mantiene en el buffer de prediccion?



Estos son los estados en 2 bits de prediccion.

Usando 2 bits en vez de 1, el salto que favorese definivamente a una eleccion (tomada o no tomada) despues de muchos saltos. se equivocara solo una vez.

Pero los 2 bits son usados para codificar un sistema de 4 estados.

El esquema de 2 bits es un ejemplo general de un contador basado en un predictor, el cual se incrementa cuando la prediccion es acertada y se decrementa cuando esta no lo es. Además, se usa un punto muerto de este rango como una division de lo tomado y no-tomado.