Simulador de Técnicas de Predicción Dinámicas

Vicente Arnau y Rubén Avendaño

Resumen—**Hemos desarrollado un simulador gráfico de técnicas de predicción dinámicas sobre el lenguaje de programación en ensamblador DLX. Permite cargar un fichero de texto con el programa y seleccionar una de las 4 técnicas de predicción dinámica realizadas. Ejecuta de diversas formas el programa y muestra en todo momento el estado de las tablas utilizadas para cada predictor. Ofrece la posibilidad de salvar a un fichero de texto un completo informe con los resultados de cada predicción realizada. Está accesible en la página web de los autores.**
Palabras clave—**Predicción dinámica, programación en ensamblador.**

I. INTRODUCCIÓN

LA predicción de saltos pretende reducir la penalización producida por las instrucciones de salto en la ejecución de los programas. Los procesadores con predicción dinámica de saltos, realizan la prebúsqueda de instrucciones de después del salto y comienza su ejecución antes de saber con exactitud si el salto será tomado o no. Además de predecir el comportamiento de salto, los procesadores necesitan conocer cual será la dirección efectiva en caso de que el salto sea tomado. Las técnicas de predicción se clasifican en dos tipos: estática y dinámicas [1][8]. Las predicciones estáticas tienen menos precisión y pueden llegar a incrementar significativamente el código final, pero requieren menos hardware específico dentro del procesador. Las dinámicas pueden llegar a obtener unos rendimientos cercanos al 98% de aciertos.

En el presente trabajo describimos un sencillo simulador de técnicas de predicción dinámicas realizado como proyecto final de carrera en el Departamento de Informática de la Universidad de Valencia. Este simulador está pensado para ser utilizado en la asignatura de 9 créditos llamada Arquitectura e Ingeniería de Computadores, de cuarto curso de la Ingeniería Informática. En esta asignatura troncal y anual de cuarto curso se ven técnicas avanzadas para procesadores segmentados, durante el primer cuatrimestre, y en el segundo cuatrimestre estudiamos los sistemas multiprocesadores y su rendimiento. En quinto curso, la Ingeniería Informática posee una asignatura obligatoria de 4,5 créditos donde se estudian los multicomputadores y los procesadores vectoriales.

El simulador de técnicas de predicción dinámicas, incorpora a su vez un simulador de ejecución de programas en el ensamblador DLX. La ejecución es en modo monociclo para poder concentrarnos solo en las técnicas de predicción realizadas y sus porcentajes de acierto. Tiene muchas limitaciones en cuanto a su uso en lo referente al tamaño de las tablas a crear para cada uno de los predictores, pero por contra nos proporciona una herramienta muy clara y visual de la utilización y el estado de cada uno de los predictores en cualquier instante de ejecución. Ejecuta las instrucciones una a una y cada vez que llega a una instrucción de salto nos muestra la predicción realizada sobre la misma. En todo momento utilizamos predictores de 2 bits, en concreto contadores saturados. En la figura 1 se muestra el conocido esquema de funcionamiento de los mismos.

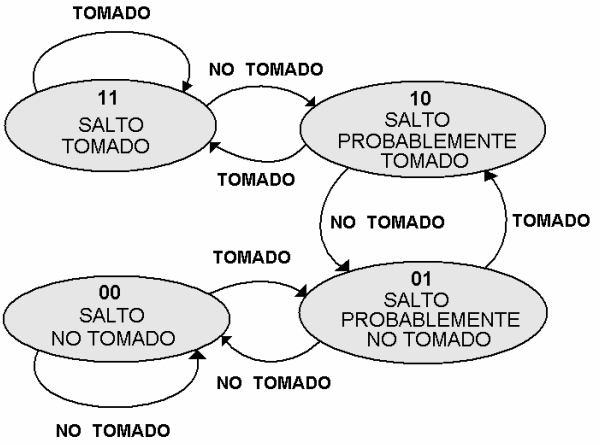


Fig. 1. Contadores saturados de 2 bits. El bit de mayor peso es el que nos indicará en todo momento sobre la previsión de que salto sea predicho como tomado o como no tomado.

El simulador permite generar un fichero de texto con los resultados para cada ejecución, que nos informa del estado final de las tablas y nos dará información sobre el porcentaje final de aciertos y el número de colisiones producidas [4].

En la página web http://www/~varnau/ se puede encontrar el ejecutable del simulador descrito en este trabajo, el cual se podrá descargar sin problemas y descomprimir en un sistema bajo Microsoft Windows®.

En el siguiente apartado describiremos brevemente los distintos predictores realizados y sus características, en el apartado siguiente mostraremos el funcionamiento del simulador desarrollado mostrando sus menús, para finalmente dar algunas conclusiones sobre su uso y mostraremos la bibliografía empleada.

II. PREDICTORES REALIZADOS

El simulador presenta la posibilidad de elegir entre cuatro predictores distintos: BTB (Brach Target Buffer), dos niveles de historia, siguiente línea de cache y predictor basado en el camino recorrido [2][3][5]. Además permite la posibilidad de elegir entre dos realizaciones distintas para la siguiente línea de cache,

Departamento de de Informática, Universidad de Valencia, Campus de Burjassot. C/Doctor Moliner 50. 46100 Burjassot (Valencia). Correo electrónico: Vicente.Arnau@uv.es.

con predicción incluida en la propia línea o con tabla de predicción asociada.

A. *Branch Target Buffer.*

El predictor BTB está basado en la utilización de una pequeña memoria asociativa que guarda información sobre la predicción de cada instrucción de salto y la dirección de salto de las mismas. Cada vez que llegamos a una instrucción de salto, los *n* bits menos significativos y distintos de cero de la dirección de esta instrucción nos dará la dirección de entrada a la tabla de predicción. El valor que toma *n* determina el tamaño de la tabla de predicción, que poseerá 2^n entradas. En el simulador este valor en determinado por el usuario entre un rango de valores dados. Cada salto es introducido en la tabla cuando es tomado por primera vez.

En la figura 2 puede verse el esquema de este primer predictor.

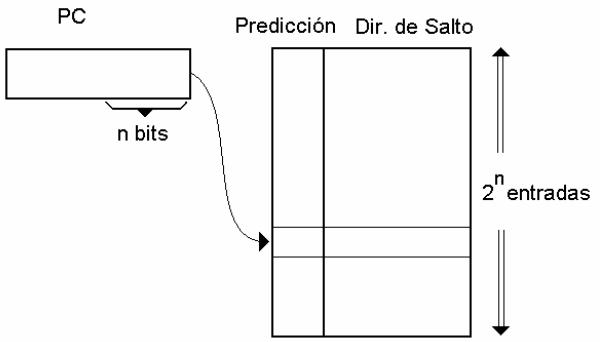


Fig. 2. Los *n* bits menos significativos del valor de la dirección de la instrucción de salto sirven para determinar la entrada a la tabla de predicción.

Tanto en este predictor como en los demás, utilizaremos siempre contadores saturados de 2 bits para almacenar la predicción. Donde el bits de mayor peso será el utilizado para determinar si el salto es predicho como tomado (11 o 10) o como no tomado (10 o 00).

El número de bits necesarios para realizar este predictor será de

$$\text{Nº de bits} = [2^n * (2 + 32)] \text{ bits} \quad (1)$$

B. *Predictor de 2 niveles de historia.*

El siguiente predictor realizado es el basado en dos niveles de historia. Este predictor surgió para mejorar los resultados que ofrecía el BTB. Es un predictor que utiliza una mayor cantidad de recursos dentro del procesador, pero que ofrece unos porcentajes de aciertos más altos.

Basa su funcionamiento en la utilización de un registro de desplazamiento de *m* bits, que almacenar la historia de las últimas *m* veces que fue ejecutado un salto. Se almacenará un *1* si el salto fue tomado o un *0* si el salto no fue tomado. Cada vez que se accede a una instrucción de salto, el patrón de bits almacenado en este registro de desplazamiento determinará la entrada a una segunda tabla de 2^m contadores saturados que determinarán si el salto será predicho como tomado o no.

En la figura 3 se muestra un esquema de este predictor tal como lo hemos realizado nosotros.

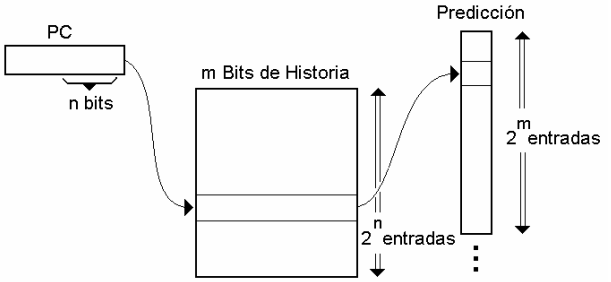


Fig. 3. Los *n* bits menos del PC de la dirección de salto, indexan una tabla de n registros de desplazamiento de m bits cada uno de ellos. Cada registro de desplazamiento apunta a una segunda tabla de predicción distinta.

En este predictor, para cada registro de desplazamiento de m bits tendremos una tabla de de 2^m contadores que serán los que realmente realizarán la predicción.

Además, cada registro de desplazamiento tiene asociado una dirección de salto, para cuando la predicción sea que el salto será tomado.

El número total de bits a utilizar para realizar este predictor viene determinado por la siguiente expresión.

$$\text{Nº de bits} = [2^n * (m + 32) + 2^n * 2^m * 2] \text{ bits} \quad (2)$$

Como podemos observa, el coste en bits es mucho mayor que el BTB. Cuando estamos ejecutando programas con bucles que se repiten muchas veces, este predictor, si posee suficientes bits en el registro de historia, es capaz de almacenar todos los patrones de comportamiento de los saltos y obtener unos porcentajes de acierto elevados. Incluso en aplicaciones reales [6].

C. *Siguiente línea de caché.*

Estos predictores son llamados también NLS (*Next Cache Line and Set*). Podemos encontrar en la bibliografía diversas realizaciones de este predictor. Nosotros vamos a realizar dos versiones. La primera de ellas, que llamaremos Método I, añade a cada línea de cache unos bits adicionales que nos informarán, en caso de que la línea de cache contenga alguna instrucción de salto, de cual es la predicción sobre dicho salto, y en caso de que se prediga que será tomado, cual es la siguiente línea de cache a cargar y en que posición está la instrucción apuntada por la etiqueta del salto dentro de esta nueva línea de cache.

Un detalle importante de esta realización es que cuando en una línea de cache nos encontramos dos instrucciones de salto condicional, el simulador añade antes de la segunda instrucciones de salto tantas instrucciones *nop* como hagan falta para que no coincidan en una línea de cache dos instrucciones de salto. En la figura 4 se puede observa el detalle de la realización del Metodo I del NLS.

Las líneas de cache están formadas por un número de instrucciones que va desde 2 hasta 8 instrucciones por línea de cache. Tenemos 2 bits para la predicción, más *n* bits para indicar cual es la siguiente línea de cache y *p*

bits más para indicar la posición de la instrucción a la que se salta en la nueva línea de cache.

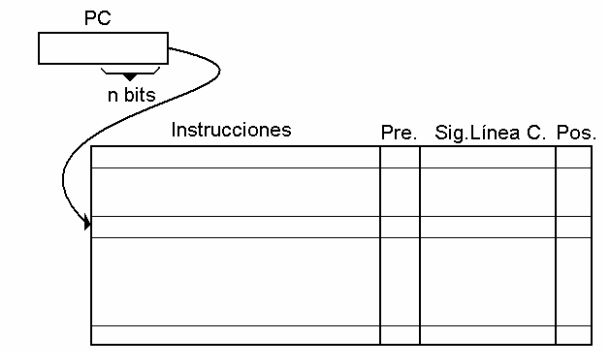


Fig. 4. NLS Método I. Se puede apreciar como cada línea de cache contiene los 2 bits de predicción, cual es la siguiente línea de cache y que posición ocupa instrucción de la etiqueta del salto en la línea de cache a la cual se predice que se va a saltar si el salto es efectivo.

En la segunda realización, que llamamos Metodo II, las líneas de cache no contienen la predicción en la propia línea, si no que utilizan una tabla auxiliar, similar al BTB, que contendrá la predicción asociada a cada línea de cache. Esta tabla posee un tamaño mucho menor que el número de líneas de cache existentes, y está mapeada con los *n* bits menos significativos de la cada línea de cache.

El la figura 4 se puede observa la realización que hemos implementado de este predictor.

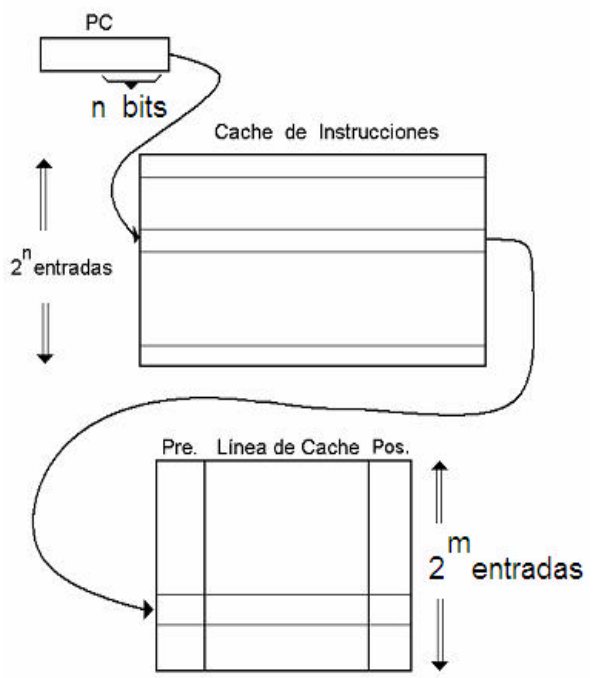


Fig. 5. NLS Método II. Usamos una Tabla de Predicción que se accede a partir de los n bits menos significativos de la línea de cache.

Si tenemos 2^k líneas de cache, y la predicción está contenida en la propia línea de cache, el número total de bits a utilizar para realizar el predictor Metodo I será:

$$\text{Nº de bits} = [2^m * (2+n+k)] \text{ bits} \quad (3)$$

Si por el contrario, utilizamos una pequeña memoria asociativa de 2^n entradas, ahora necesitaremos bastantes menos bits para la realización, pues el valor n es ahora un número bastante menor que el número total de líneas de cache que teníamos antes:

$$\text{Nº de bits} = [2^n * (2+k+p)] \text{ bits} \quad (4)$$

D. Predictor basado en el camino recorrido.

Este predictor presenta una filosofía de funcionamiento bastante diferente al resto de predictores vistos hasta ahora [9]. Este predictor no guarda la historia de los últimos saltos ejecutados y su comportamiento, si no que guarda información a cerca del flujo del programa y estable la predicción en base a los bloques de código ejecutados con anterioridad.

Los anteriores predictores ofrecen un funcionamiento muy bueno cuando ejecutamos bucles que se repiten muchas veces, pero fallan bastante cuando tenemos instrucciones del tipo *if-then-else* en las cuales el que se tome el camino del *if* o el del *else* puede depender de por donde viene el flujo del programa.

Para estos predictores lo que guardamos es la información del camino recorrido, antes de llegar a una determinada instrucción de salto. Para su realización, utilizaremos un número *n* de bits para etiquetar cada uno de los bloques básicos por lo que puede transitar el programa. En nuestra realización particular, hemos utilizaremos *n* bits para etiquetar cada una de las instrucciones de salto del programa, así sabiendo que instrucciones de salto se han ejecutado con anterioridad podemos saber por que bloques de código ha pasado el programa en su ejecución. Y usaremos un registro de historia, que será un registro de desplazamiento de *k*n* bits, don el valor *k* nos informará del número de bloques que queremos recordar y *n* el número de bits que usaremos para codificarlos.

Para realizar la predicción, usaremos una tabla de 2^{k*n} contadores saturados de 2 bits. Por lo tanto la cantidad de memoria necesaria es de:

$$\text{Nº de bits} = [2^{k*n} * (2 + 32)] \text{ bits} \quad (5)$$

III. EL SIMULADOR

El simulador realizado ofrece la posibilidad de cargar y ejecutar un fichero escrito en ensamblador de DLX. Pudiendo elegir entre 4 predictores dinámicos distintos.

Posee 4 menús desplegables. El Primero de ellos depende de la opción *Archivo* y nos permite cargar un programa, restaurar la ejecución del programa o reiniciar completamente el simulador. La figura 6 (a) muestra este submenu.

El siguiente menú desplegable que se muestra en la figura 6 (b) nos ofrece la posibilidad de ejecutar un programa paso a paso, hasta una instrucción determinada, todo el programa o ejecutar grupos de *n* instrucciones. Además nos ofrece la posibilidad de visualizar una ventana con un informe sobre el resultado de la ejecución del programa con un determinado

simulador y opcionalmente salvar dicha salida en un fichero de texto.

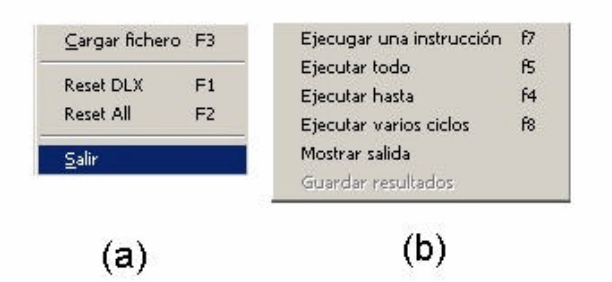


Fig. 6. Menús desplegables que ofrece el simulador.

El tercer menú desplegable nos ofrece la posibilidad de elegir entre cualquiera de los 4 predictores realizados. En caso de elegir el predictor por siguiente línea de cache, ofrece las dos realizaciones comentada con anterioridad, a las cuales llamamos Método I y Método II. Este menú se puede apreciar en la figura 7.

Fig. 7. El simulador ofrece la posibilidad de elegir entre 4 predictores dinámicos. En caso de elegir el predictor por siguiente línea de cache, habrá que especificar cual de las dos realizaciones vamos a utilizar.

El cuarto menú desplegable solo sirve para informa de quien ha programado el simulador y quien le ha dirigido el proyecto.

Una vez elegido el tipo de predictor a utilizar, al comenzar la ejecución del programa, pedirá el tamaño de las tablas que se van a utilizar en cada uno de los predictores. El tamaño de estas tablas es bastante pequeño comparado con las realizaciones que suelen tener los procesadores comerciales [7]. Pero será suficiente para mostrar el funcionamiento de los predictores sobre programas escritos por nosotros para resolver problemas típicos de una sesión de laboratorio.

La tabla I ofrece los tamaños de los dos primeros predictores.

TABLA I

LIMITACIONES DEL TAMAÑO DE LOS PREDICTORE S BTB (TABLA DE PREDICCIÓN) Y BNH (DOS NIVELES DE HISTORIA).

BTB	DNH
Nº entradas: 4, 8, 16, 32, 64	Nº entradas: 4, 8, 16, 32, 64
	Nº de bits de Historia: 2 a 6

La tabla II nos ofrece los tamaños de las dos posibles realizaciones del predictor NLS, el Método I en el que la predicción está incorporada a cada línea de cache y el Método II donde la predicción se realiza con una tabla de predicción auxiliar. Además se ofrece el tamaño del predictor basado en el camino recorrido (PCR). Este último predictor solo ofrece la posibilidad de un registro de desplazamiento capaz de recordar las últimas 2 o 3 instrucciones de salto ejecutadas. Con un total de 4 u 8 posible instrucciones de salto etiquetadas, según se elija 2 o 3 bits por registro.

TABLA II

LIMITACIONES DEL TAMAÑO DELAS DOS REALIZACIONES DEL PREDICTOR LNS Y DEL PCR.

NLS I	NLS II	PCR
Nº de líneas de cache: 8, 16, 32, 64	Nº de líneas de cache: 8, 16, 32, 64	Nº de registros 2, 3
Nº de instrucciones por línea: 2, 4, 8	Nº de instrucciones por línea: 2, 4, 8	Nº de bits por registro 2, 3
	Nº de líneas de la tabla de predicción: 2, 4,8	

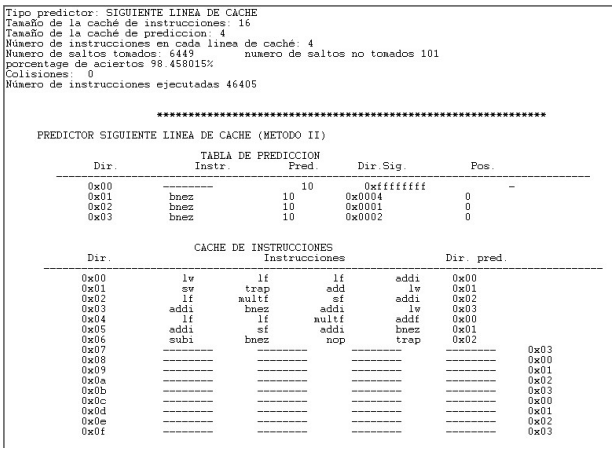
IV. RESULTADOS

Este simulador lo hemos utilizado en una de las sesiones de prácticas de la asignatura Arquitectura e Ingeniería de Computadores de 4º curso de la Ingeniería Informática. Ha sido una herramienta muy útil para la corrección de los simuladores que previamente los alumnos realizaban a mano. Hemos pensado que utilizar solo el simulador podría conllevar el perder la noción de cómo estaba funcionando realmente la técnica de predicción paso a paso y por ese motivo se ha utilizado más como una herramienta de apoyo.

Los resultados obtenidos en cuanto a porcentajes de predicción han sido bastante similares a los porcentajes que alcanzan los procesadores comerciales según sus propios fabricantes.

El fichero de salida que se genera ofrece información detallada de cómo se ha ejecutado un fichero y de cómo han quedado las tablas de memoria o las líneas de cache después de la ejecución del programa. La figura 8 ofrece el contenido de uno de estos ficheros de salida.

En la figura 9 que se encuentra al final de este trabajo se ofrece una imagen de la apariencia del simulador desarrollado. Podemos observar 4 ventanas en el simulador. Una primera ventana con el estado de los 32 registro enteros y 32 en coma flotante que posee el procesador de DLX. Una segunda ventana con el código que se está ejecutando, en la cual aparecerán todas las instrucciones en minúscula salvo la que se está ejecutando que aparecerá en mayúscula, si se ejecuta paso a paso. En la ventana de la derecha se muestra el estado de las tablas que usa cada uno de los predictores. Y en la parte inferior se muestra una ventana con información auxiliar sobre el comportamiento del simulador de técnicas de predicción dinámicas que se está ejecutando, mostrando los porcentajes de acierto alcanzados en las predicciones.



```
Tipo predictor: SIGUIENTE LINEA DE CACHE
Tamaño de la caché de instrucciones: 16
Tamaño de la caché de prediccion: 4
Número de instrucciones en cada linea de caché: 4
Numero de saltos tomados: 6449       numero de saltos no tomados 101
porcentage de aciertos 98.458015%
Colisiones:  0
Número de instrucciones ejecutadas 46405

************************************************************

PREDICTOR SIGUIENTE LINEA DE CACHE (METODO II)

                    TABLA DE PREDICCION
      Dir.          Instr.          Pred.       Dir.Sig.          Pos.
-------------------------------------------------------------------------
      0x00          --------          10        0xffffffff          -
      0x01          bnez              10        0x0004              0
      0x02          bnez              10        0x0001              0
      0x03          bnez              10        0x0002              0


                    CACHE DE INSTRUCCIONES
      Dir.                    Instrucciones                  Dir. pred.
-------------------------------------------------------------------------
      0x00          lw        lf        lf        addi      0x00
      0x01          sw        trap      add       lw        0x01
      0x02          lf        multf     sf        addi      0x02
      0x03          addi      bnez      addi      lw        0x03
      0x04          lf        lf        multf     addf      0x00
      0x05          addi      sf        addi      bnez      0x01
      0x06          subi      bnez      nop       trap      0x02
      0x07          --------  --------  --------  --------  --------  0x03
      0x08          --------  --------  --------  --------  --------  0x00
      0x09          --------  --------  --------  --------  --------  0x01
      0x0a          --------  --------  --------  --------  --------  0x02
      0x0b          --------  --------  --------  --------  --------  0x03
      0x0c          --------  --------  --------  --------  --------  0x00
      0x0d          --------  --------  --------  --------  --------  0x01
      0x0e          --------  --------  --------  --------  --------  0x02
      0x0f          --------  --------  --------  --------  --------  0x03
```

Fig. 8. Fichero resultado que ofrece el simulador.

V. AGRADECIMIENTOS

El presente trabajo ha sido motivado por la lectura del report interno de la Universidad Politécnica de Barcelona UPC-CEPBA-1996-11, escrito por José Gonzáles y Antonio González [8]. Este trabajo ha sido financiado por la MCYT (grant no. TIC2003-08154-C06-04).

VI. BIBLIOGRAFÍA

[1] J.L. Hennessy, D. A. Patterson. "Computer Achitecture. A Quantitative Approach". 3ª edición. Morgan Kaufmann Publishers. 2003.

[2] J.E. Smith. "A Study of Branch Prediction Strategies". Proc. Computer Architecture, pp. 135-148. 1981.

[3] J. Lee, A.J. Smith. "Brach Prediction Strategies and Brach Target Buffer Design". Computer, vol. 17, N. 1, pp 6-22. 1984.

[4] S. Sechrest, C.C. Lee, T.N. Mudge. "Correlation and Aliasing in Dynamic Branch Predictors". Proceeding of 29th Annual International Symposium on Computer Architectue, pp. 22-32. 1996.

[5] T.Y. Yeh, Y.N. Patt. "A Comparison of Dynamic Branch Predictors that use Two levels of Brach History". Pro. 20th Annual International Symposium on Computer Architectue, pp. 257-266. 1993.

[6] R.B. Hilgendorf, G.J. IEM, W. Rosentiel. "Evaluation of Branch-Prediction Methods on Traces from Commercial Applications". IBM J. Res. Develop. Vol. 43, no. 4. Julio 1999.

[7] K. Yeager. "The MIPS R10000 Superscalar Microprocessor". IEEE Micro. Vol. 16. No. 2. Pp 28-40. 1996.

[8] J. Gonzáles, A. González. "Recopilación de Técnicas de predicción de Saltos". Report Interno UPC-CEPBA-1996-11. 1996.

[9] C. Young, M.D. Smith. "Improving the accuracy of Static Branch Prediction Using Branch Correlation". Pro. of the 6th Annual International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 232-241. 1994.

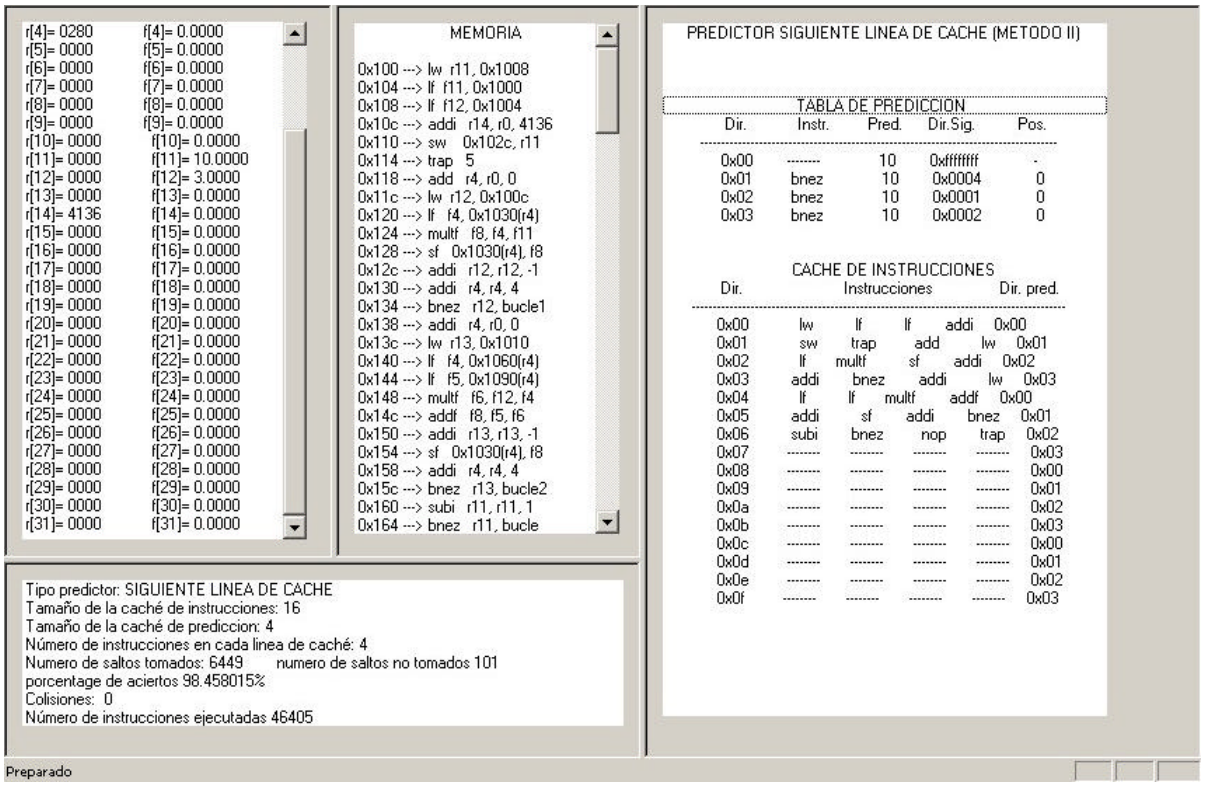


Fig. 9. Imagen que ofrece el simulador una vez ejecutado un programa con un predictor de saltos determinado.