ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

INGENIERIA DE SOFTWARE

ARQUITECTURA DE COMPUTADORES
ICCD332

INTEGRANTES:

Cofre Santo Juan Fernando

Huilca Villagómez Fernando Eliceo

Jiménez González Jeremy Anderson

Parra Sarasti Jhordy Oliver

Proyecto II Bimestre

Tema:

Funcionamiento de una computadora a nivel de hardware

Código del programa desarrollado para el proyecto: https://github.com/FernandoHuilca/Emulador_Arquitectura_Computacional/tree/EnConsola

Índice:

Contenido

Índice de Figuras

Objetivos:

- Implementar al modelo de Von Neumann una memoria intermedia a través de software. Esta corresponderá a la memoria caché, que facilite el manejo de los datos que son extraídos de la memoria principal y permita el aumento de la eficiencia en el procesamiento de operaciones. (Escrito por: Juan Cofre)
- El objetivo es emular el comportamiento y la interacción entre los componentes de la CPU, la caché y la RAM, aplicando mapeo directo, políticas de lectura y escritura, operaciones booleanas, y proporcionando ejemplos prácticos de aplicación. (Escrito por: Fernando Huilca)
- Desarrollar una simulación en Java que represente la interacción entre la CPU, la memoria caché y la memoria principal (RAM), permitiendo analizar el impacto de diferentes políticas de caché (mapeo directo, asociativo, y asociativo por conjuntos) y estrategias de reemplazo (LRU, FIFO) en el rendimiento del sistema. (Escrito por Jeremy Jimenez)
- Desarrollar una simulación que integre operaciones avanzadas como multiplicación y división en la interacción entre la CPU, la memoria caché y la memoria principal, evaluando el impacto de estas operaciones en el rendimiento general del sistema, con especial énfasis en la gestión de condiciones de desbordamiento y el manejo eficiente de datos negativos. (Escrito por: Jhordy Parra)

Marco teórico:

Modelo de Von Neumann

El modelo de Von Neumann, también conocido como arquitectura de Von Neumann, es una estructura conceptual para computadoras digitales propuesta por John von Neumann en la década de 1940. Esta arquitectura es fundamental para las computadoras modernas y se caracteriza por su simplicidad y eficiencia en la ejecución de programas.

Componentes Principales:

1. **Unidad Central de Procesamiento (CPU):** Ejecuta instrucciones del programa y se divide en:
    - **Unidad de Control (CU):** Decodifica instrucciones y controla el flujo de datos.
    - **Unidad Aritmético-Lógica (ALU):** Realiza operaciones aritméticas y lógicas.
2. **Memoria:** Almacena datos e instrucciones, sin distinción entre ambos tipos de información.
3. **Dispositivos de Entrada y Salida (I/O):** Permiten la comunicación entre la computadora y el entorno externo.

4. **Bus:** Sistema de comunicación que transfiere datos entre la CPU, memoria y dispositivos I/O.

Ciclo de Instrucción:

1. **Fetch (Buscar):** La CPU recupera una instrucción de la memoria.
2. **Decode (Decodificar):** La CU interpreta la instrucción.
3. **Execute (Ejecutar):** La ALU realiza la operación indicada.
4. **Store (Almacenar):** Los resultados se almacenan en la memoria o en registros.

Características Claves:

- **Memoria Unificada:** La misma memoria almacena datos e instrucciones.
- **Secuencialidad:** Las instrucciones se ejecutan secuencialmente a menos que se indique lo contrario.
- **Flexibilidad:** Los programas pueden modificar sus propias instrucciones.

Ventajas:

- Simplicidad en el diseño y la implementación.
- Flexibilidad en la programación y ejecución.
- Uso eficiente de la memoria.

Desventajas:

- **Cuello de Botella de Von Neumann:** La velocidad de transferencia de datos entre la CPU y la memoria puede limitar el rendimiento.
- **Vulnerabilidad a Errores:** Tratar instrucciones como datos puede resultar en la ejecución de instrucciones incorrectas debido a errores en los datos.

Caché

Una memoria caché es una capa de almacenamiento de datos de alta velocidad que almacena un subconjunto de datos, con esto se logra que futuras solicitudes se atienden con mayor rapidez sin que se acceda a los datos que la memoria principal almacena.

Elementos de la caché

Función de correspondencia:

Como existen menos líneas de caché que bloques de memoria principal, se necesita de un algoritmo para corresponder los bloques de memoria principal con las líneas de caché.

Cuando se elige alguna función de correspondencia se puede organizar la memoria caché. Se utilizan técnicas como discreta, asociativa y asociativa por conjuntos.

Algoritmos de sustitución o reemplazo:

Para que estos algoritmos puedan trabajar de manera eficiente y rápida, es necesario que se implementen en el hardware.

- FIFO: Se basa en la lógica que siguen estas estructuras, por lo que se sustituye aquel bloque que ha estado más tiempo en caché (Independientemente de sus referencias). Se puede implementar con una cola.
- LRU (Least-Recently Used): Se busca sustituir el bloque que ha estado más tiempo en caché sin ser referenciado. Se lo aplica mediante bit de forma cíclica.
- LFU (Least-Frecuently Used): Se sustituye el bloque que ha sido el menos referenciado, y se lo implementa asociando un contador a cada línea de caché.

Política de escritura:

Las políticas de escritura en memoria caché (cache writing policies) son estrategias que determinan cómo se manejan las operaciones de escritura en una memoria caché. Existen varias políticas, y la elección de una u otra puede tener un impacto significativo en el rendimiento del sistema.

- Escritura inmediata: Toda escritura se maneja en la propia memoria caché tal como lo realiza la memoria principal (Mp). Se asegura que el contenido de la memoria principal siempre será válido, pero esto genera mucho tráfico con la memoria y da problemas de cuello de botella.

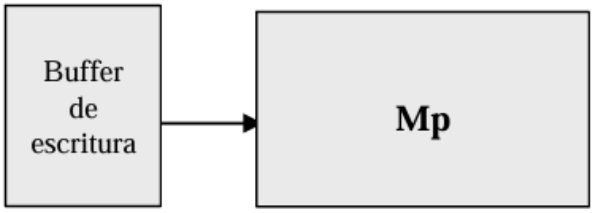

Figura 1 Escritura inmediata de caché.

- Postescritura: Busca minimizar las escrituras en memoria. A través de un bit que indica una instrucción llamada “Actualizar”, se realiza una actualización en la memoria caché. Cuando esto ocurre el bloque asociado a la línea de caché se sustituye o se escribe en memoria principal.
- Asignación en escritura: Cuando ocurre una falla de escritura el bloque que se ubica en memoria caché y luego se opera con alguna política de escritura (Inmediata o postescritura).
- No asignación en escritura: El bloque a utilizar se modifica en memoria principal sin cargarse en memoria caché.

Mapeo directo en caché:

La caché almacena copias de partes de la memoria principal. Cuando el procesador necesita leer una palabra de memoria, primero verifica si está en la caché. Si es así, la palabra se entrega al procesador. Si no, se guarda en la caché un bloque de la memoria principal, que contiene un número fijo de palabras, y luego se entrega la palabra al procesador. Debido a la localidad de

referencia, es probable que futuras referencias a la memoria apunten a la misma ubicación o a otras dentro del mismo bloque.

Memoria principal y caché:

La memoria principal consiste en: 2^n ubicaciones de memoria, cada ubicación puede tener una cantidad variable de bits para almacenar. Estas ubicaciones están registradas en una posición especifica que corresponde a un índice el cual debe tener el numero necesario de bits para representar las 2^n ubicaciones que posee la memoria.

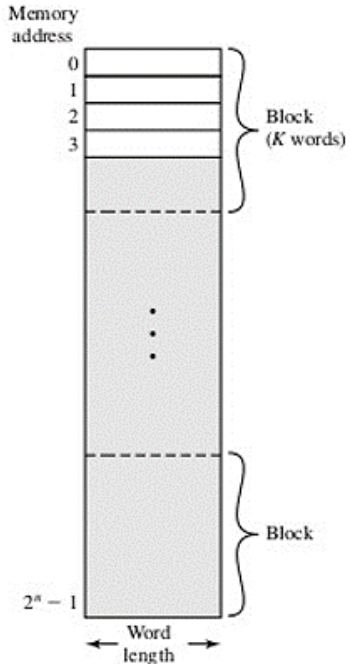

Figura 2 Diagrama de la memoria principal.

Además de las ubicaciones de la memoria, se forman bloques que tienen una cierta cantidad de ubicaciones, están normalmente están dividas de tal forma que haya un número total de bloques y que estos tengan la misma cantidad de ubicaciones. Para determinar el número de bloques disponibles en la memoria se utiliza:

$$M = \frac{2^n}{K}$$

$M = número\ de\ bloques.$

$2^n = número\ de\ ubicaciones\ de\ memoria.$

$K = número\ de\ ubicaciones\ o\ palabras\ que\ tendrá\ cada\ bloque$

La memoria caché está conformada de m bloques llamados “líneas”. Cada línea contiene una cantidad de bits que permiten hacer referencia a los bloques y a la línea especifica donde se guarde el dato que se desee almacenar. Esta

división de la caché se conforma del “Tag” (Etiqueta) y el número del bloque que tendrá cada línea de caché. Hay que considerar que el número de líneas de caché es considerablemente menor al número de bloques de la memoria principal.

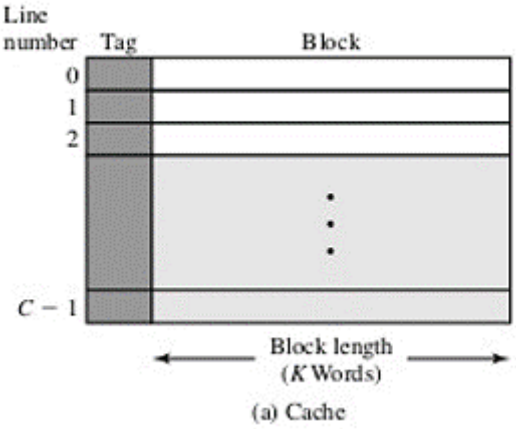

Figura 3 Esquema de la memoria caché.

Una forma sencilla de explicar el como se van colocando los bloques de la memoria principal en cada línea de caché es considerar que, con el mapeo directo, ya está predefinido que bloque va a ocupar cada línea. Si se lee la ubicación de un bloque, dicho bloque se transfiere a una de las líneas de la caché. Como cada línea no almacena un bloque de manera exclusiva, el tag o la etiqueta se utiliza para identificar que bloque necesita ser cargado en una línea determinada. La etiqueta es normalmente una parte de la dirección que el CPU necesita de un dato en específico que este alojado en la memoria principal.

Matemáticamente se mapea cada bloque de memoria con la línea de caché correspondiente a través de la operación:

$$i = j \ módulo \ m$$

$i = número\ de\ línea.$

$j = número\ del\ bloque\ en\ memoria.$

$m = número\ de\ líneas\ de\ caché.$

Como la cantidad de bloques de memoria M es mayor a la cantidad de línea m, cada línea debe ser mapeado a una línea única de caché. Por ejemplo, si la caché tiene 10 líneas y existen 100 bloques en la memoria, al seguir la operación los bloques 1,11,21, 31, ..., 91 pueden ser mapeados en la línea 1, los bloques 2, 12, 22, 32, …, 92 serían mapeados en la línea 2.

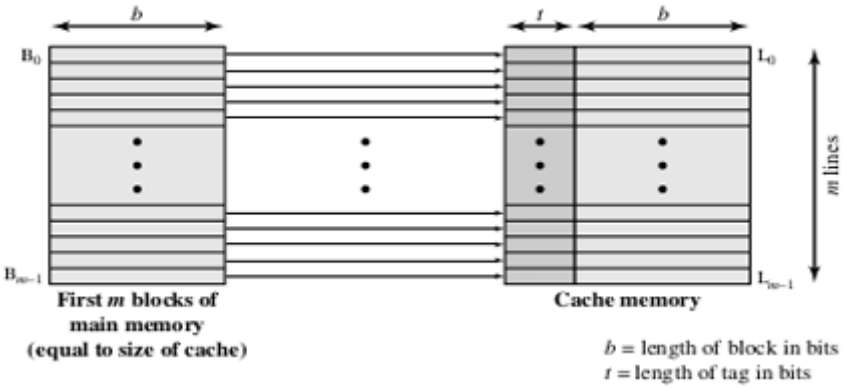

Figura 4 Mapeo directo.

Cuando el CPU requiere de un dato de memoria, se envía una dirección de memoria. Esta se divide en los elementos pertinentes (Tag, bloque y offset) para que la caché pueda procesarla y determinar si esta almacenada ahí o necesita buscar el dato requerido en la memoria principal, en este proceso es cuando se determina los estados de *inválido* o *acierto*. *Inválido* corresponde a cuando se busca un dato en la memoria caché, pero este no se encuentra ahí, por consiguiente, el estado de *acierto* indique que el dato requerido se encuentra cargado en la memoria caché y por ende no es necesario volver a buscarlo en la memoria principal.

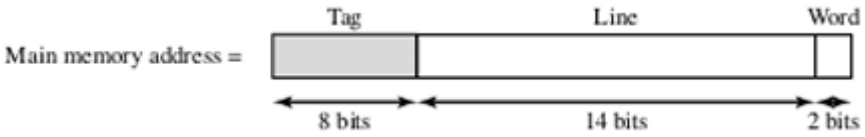

Figura 5 Dirección de memoria.

En síntesis, la memoria caché en mapeo directo tiene el funcionamiento descrito. Además, el mapeo directo es simple y barato de implementar. Su principal desventaja es que existe una ubicación fija en la caché para cada bloque, por lo que un programa que referencia datos de forma consecutiva ocasionará un alto intercambio de bloques lo que disminuye el porcentaje de aciertos.

Desarrollo de la práctica:

Inicio del programa y configuración inicial de la memoria:

Al iniciar el programa muestra una interfaz gráfica de usuario (GUI) diseñada para configurar parámetros relacionados con la memoria y la caché de un sistema informático. Se detallan los componentes principales observados:

1. **Dirección de Memoria en Bits:**
    - Se presentan cuatro opciones para la longitud de la dirección de memoria: 4 bits, 6 bits, 8 bits y 12 bits.

- En este caso, la opción seleccionada es de **6 bits**.
- Además, se indica que el **número de datos en RAM es 64**.

2. **Tamaño del Bloque en Bytes:**
   - Se ofrecen tres opciones para el tamaño del bloque en bytes: 4 bytes, 8 bytes y 12 bytes.
   - La opción seleccionada en la imagen es de **8 bytes**.
   - Se especifica que el **número de bloques es 8**.
3. **Tamaño de la Caché en Bytes:**
   - Se listan cuatro opciones para el tamaño de la caché en bytes: 8 bytes, 12 bytes, 16 bytes y 32 bytes.
   - En este caso, se ha seleccionado la opción de **32 bytes**.
   - También se indica que el **número de líneas de caché es 4**.

Figura 6 Inicio del programa.

Elementos del programa:

Después de configurar, se muestra un emulador de una Unidad Central de Procesamiento (CPU) interactuando con su unidad de control (CU), la unidad aritmético-lógica (ALU), registros, memoria caché y memoria principal. A continuación, se describe cada uno de los componentes visualizados:

1. **CPU (Unidad Central de Procesamiento):**
   - La CPU contiene una ALU (Unidad Aritmético-Lógica) que recibe dos entradas A y B, ambas de 8 bits, y realiza operaciones aritméticas o lógicas para producir una salida también de 8 bits.
   - Se observan indicadores para Cero, Negativo y Over Flow que señalan estados específicos tras la operación realizada por la ALU.

- Además, se encuentran los Registros (PC, RI, R0, R1), donde se almacenan valores que pueden ser utilizados por la ALU o la Unidad de Control.

2. **CU (Unidad de Control):**
   - La CU se encarga de dirigir y coordinar las operaciones entre la ALU, los registros y la memoria caché. Intercambia información con la ALU y gestiona las operaciones a realizar.
3. **Caché:**
   - La memoria caché contiene varios bloques, aunque en la imagen actual todos están marcados con "x" indicando que están vacíos o no han sido asignados.
   - Se muestran parámetros como el Tag, Index, Offset, y un Bit Sucio (Dirty Bit) que indica si los datos en caché han sido modificados y necesitan ser escritos en la memoria principal.
4. **Memoria Principal:**
   - Se observa un conjunto de direcciones de memoria en la columna izquierda, y los datos asociados en la columna derecha. Esta memoria contiene los valores almacenados en cada ubicación de la memoria principal.
5. **Opciones de Operación:**
   - En la parte inferior izquierda, se puede ver un conjunto de opciones de instrucciones como LOAD, STORE, MOVE, SUB, ADD, MUL y BUBBLE que indican las operaciones que pueden ser realizadas por la CPU.
   - También se proporciona una casilla para seleccionar un Registro y realizar una operación específica.
6. **Botones Adicionales:**
   - Existe un botón para Volver a configurar la RAM, lo que sugiere que se puede modificar o resetear la configuración de la memoria RAM en el emulador.

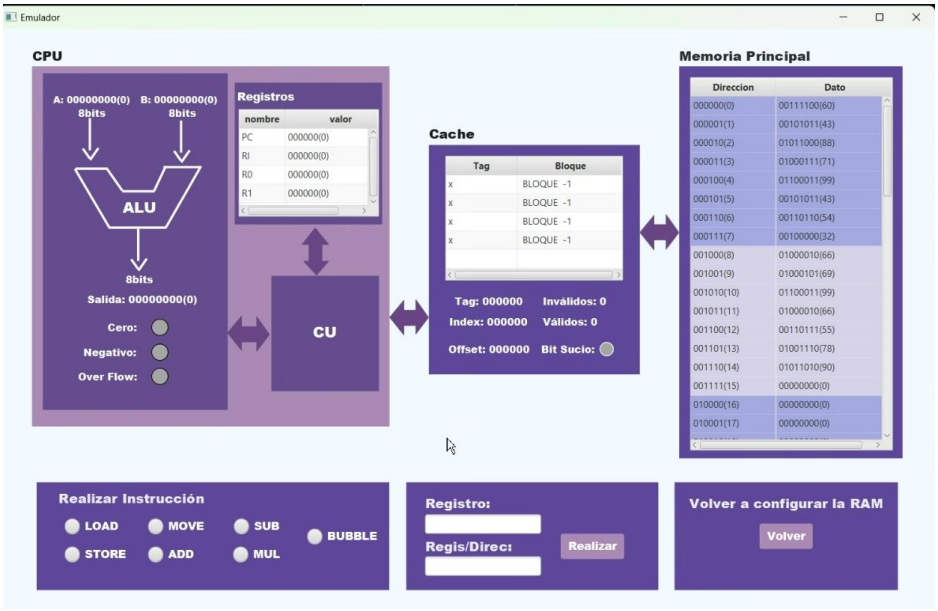

Figura 7 Ventana principal del programa

Antes de ejecutar la instrucción de carga (LOAD) se va a detallar cada componente:

1. **ALU (Unidad Aritmética Lógica):**
   - Las entradas A y B están en 00000000(0) y la salida también es 00000000(0).
   - Los indicadores de Cero, Negativo, y Over Flow están apagados.
2. **Registros:**
   - PC (Contador de Programa): 000000(0)
   - RI (Registro de Instrucción): 000000(0)
   - R0: 000000(0)
   - R1: 000000(0)
   - Los valores de los registros están en cero, indicando que no hay datos almacenados en ellos antes de ejecutar la instrucción.
3. **Caché:**
   - Todos los bloques de la memoria caché están marcados como inválidos, con el valor del bloque en -1 y el tag sin definir (x).
   - Los indicadores de la caché (Inválidos, Válidos, Offset, y Bit Sucio) están en 0, lo que indica que la caché está vacía y no hay datos válidos presentes.
4. **Memoria Principal:**
   - La memoria principal muestra varias direcciones y sus datos asociados, donde cada dirección tiene un valor específico en binario.
   - Por ejemplo, la dirección 000110(6) contiene el valor binario 00110110(54).
5. **Instrucción a Realizar:**
   - No se ha seleccionado ninguna instrucción en este punto, por lo que los registros están vacíos y la caché no contiene datos válidos.

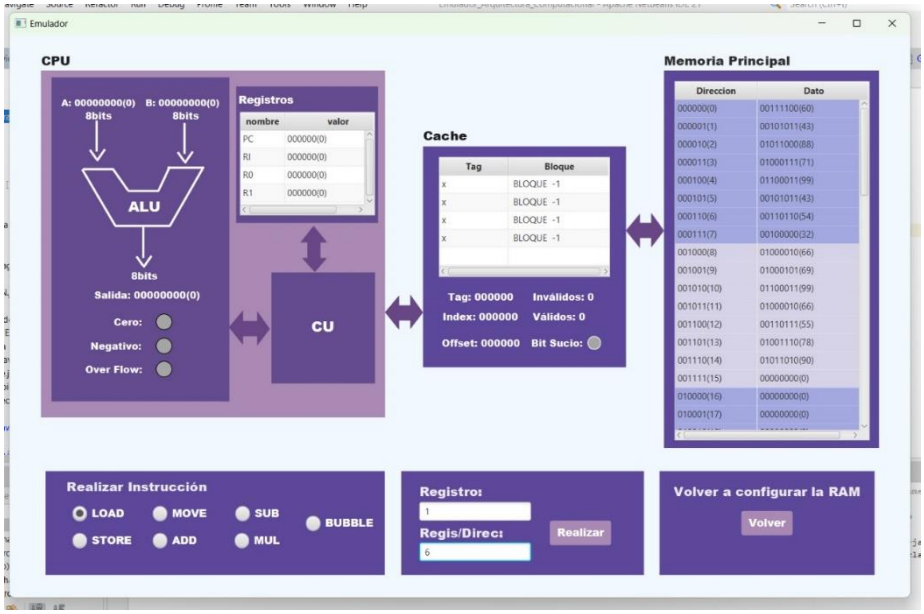

Figura 8 Programa antes de la ejecución de la operación Load

Operaciones del programa:

Después de ejecutar la instrucción de carga (LOAD), se observa cómo los datos se han movido desde la memoria principal hacia un registro, y la caché también ha sido parcialmente actualizada.

1. **ALU (Unidad Aritmética Lógica):**
   - Las entradas A y B permanecen en 00000000(0) y la salida también sigue siendo 00000000(0).
   - Los indicadores de Cero, Negativo, y Over Flow siguen apagados, indicando que no se ha realizado ninguna operación aritmética o lógica en este paso.
2. **Registros:**
   - PC (Contador de Programa): 000000(0)
   - RI (Registro de Instrucción): 000001(1)
   - R0: 000000(0)
   - R1: 110110(54)
   - El registro R1 ahora contiene el valor 110110(54), que fue cargado desde la memoria principal.
3. **Caché:**
   - La caché ahora tiene un bloque 0, con un tag de 0, indicando que se ha cargado una línea de la memoria principal.
   - El índice es 00, y el offset es 110, reflejando que se ha accedido a una dirección específica en la memoria.
   - El indicador de inválidos es 1, lo que sugiere que se ha reemplazado un bloque anterior (que era inválido), pero no hay datos válidos adicionales en la caché.
4. **Memoria Principal:**

- La memoria principal sigue mostrando las mismas direcciones y datos que antes, pero ahora sabemos que el valor 110110(54) de la dirección 000110(6) ha sido cargado en el registro R1.
- No ha habido cambios en la memoria principal, ya que solo se ha leído un valor, no se ha modificado.

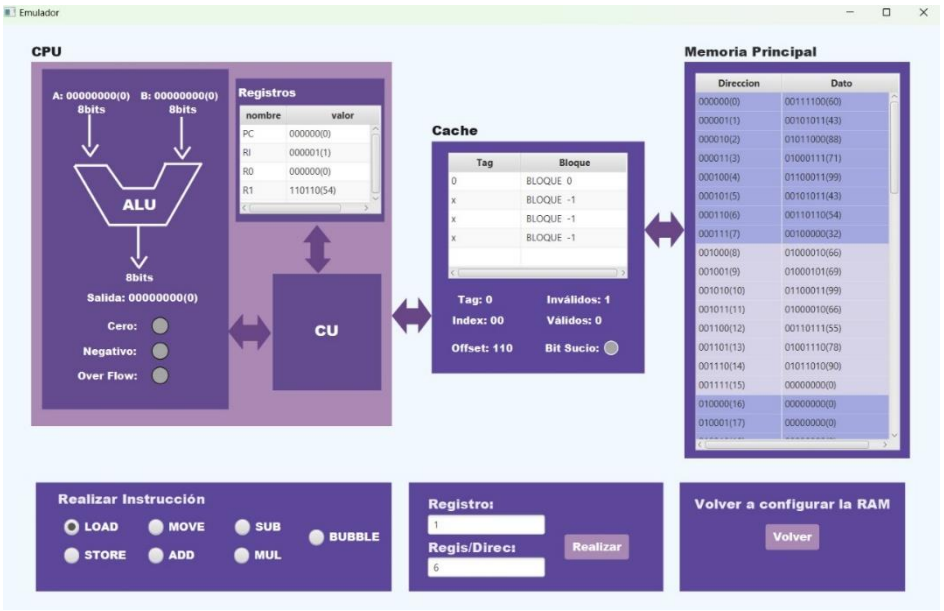

Figura 9 Programa después de la ejecución de la operación Load

A continuación, se mostrará el funcionamiento de la instrucción Guardar (Store). Pero primero se indicará los datos antes de su ejecución.

1. **Configuración inicial:**
   - La ALU (Unidad Aritmético-Lógica) muestra los valores de los registros A y B, ambos con valores 00000000(0), lo que significa que están vacíos o no se han utilizado en operaciones recientes.
   - La salida de la ALU también es 00000000(0), indicando que no se ha realizado ninguna operación aritmética o lógica.
2. **Registros:**
   - Los registros muestran el estado actual de los valores:
     - PC: 000001(1)
     - RI: 000001(1)
     - R0: 000000(0)
     - R1: 110110(54)
   - Estos registros son utilizados para guardar direcciones, instrucciones, o valores temporales en las operaciones.
3. **Cache:**

   - En la memoria caché, solo el bloque 0 está asignado (BLOQUE 0), con las demás entradas mostrando BLOQUE -1, lo que indica que no están asignadas.

- Los campos Inválidos, Válidos y Bit Sucio están configurados según la información actual, mostrando Inválidos: 1, Válidos: 0, y Bit Sucio desactivado.

4. **Memoria Principal:**
   - La memoria principal muestra una lista de direcciones y datos correspondientes. Por ejemplo, en la dirección 000000(0) se encuentra el dato 00111100(60).
   - Estas direcciones y datos están listados en formato binario y decimal.

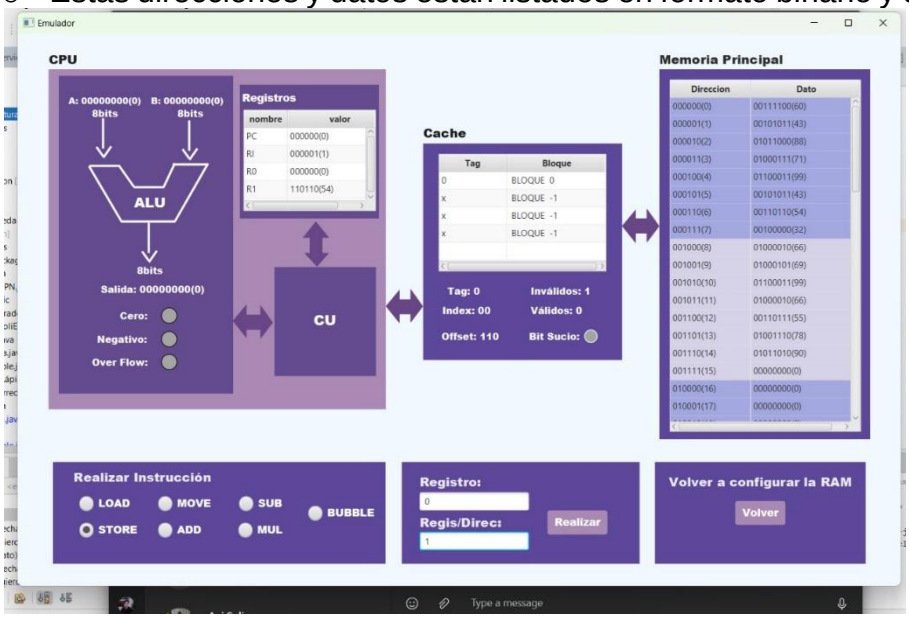

Figura 10 Programa antes de la ejecución de la operación Store

1. **Estado tras la ejecución:**
   - Después de ejecutar la instrucción STORE, la configuración de la ALU y la CPU no ha cambiado en términos de los valores mostrados en A, B y la salida.
   - Sin embargo, el estado de la memoria caché y la memoria principal ha cambiado.
2. **Cache:**
   - El bloque 0 sigue asignado (BLOQUE 0), pero ahora tiene un Bit Sucio activado, indicando que el bloque en caché ha sido modificado, pero no aún escrito en la memoria principal.
   - El campo Válidos ha cambiado a 1, lo que indica que la caché ahora contiene datos válidos.
   - Offset muestra 001, lo que sugiere que se ha realizado un acceso específico a una posición dentro del bloque de caché.
3. **Memoria Principal:**
   - La memoria principal muestra que se ha modificado el dato en la dirección 000001(1), el cual antes era 00000000(0) y ahora es 110110(54), coincidiendo con el valor del registro R1.
   - Esto confirma que la operación STORE ha almacenado el valor de R1 en la dirección 1 de la memoria.

4. **Realizar Instrucción:**
   - En esta etapa, la instrucción STORE se ha completado, y el sistema está preparado para la próxima operación.

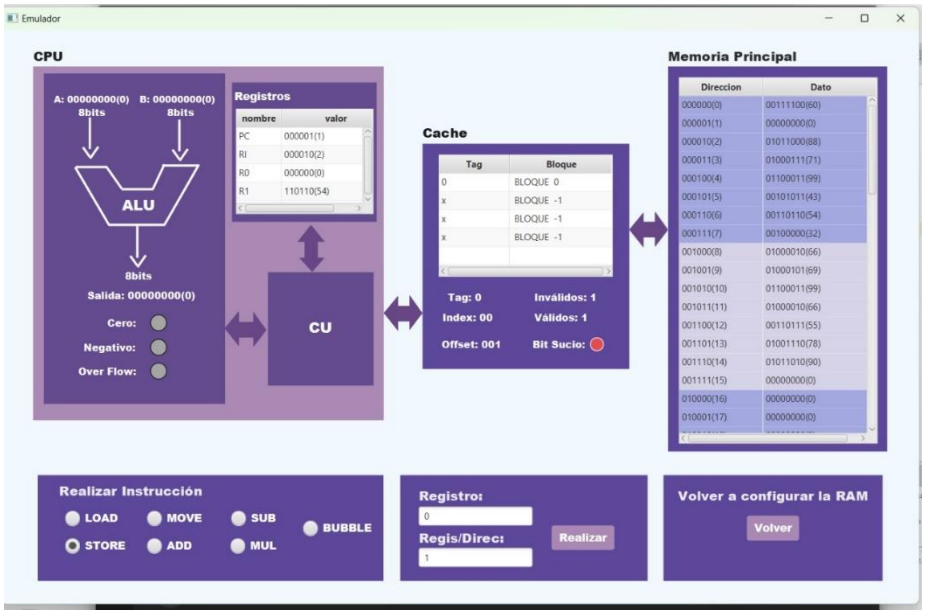

Figura 11 Programa después de la ejecución de la operación Store

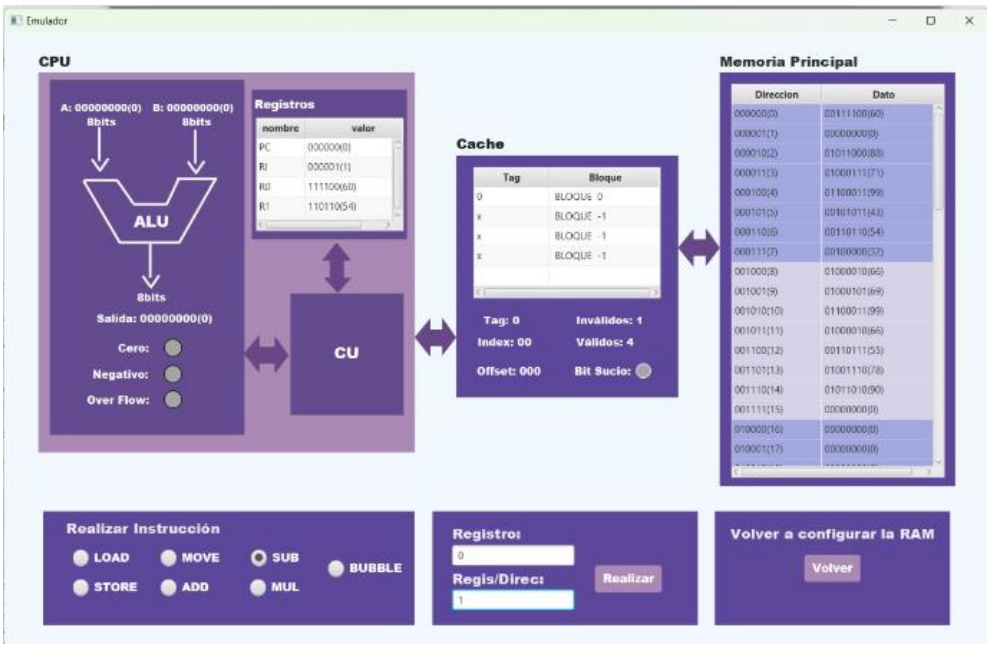

Figura 12 Operación Resta

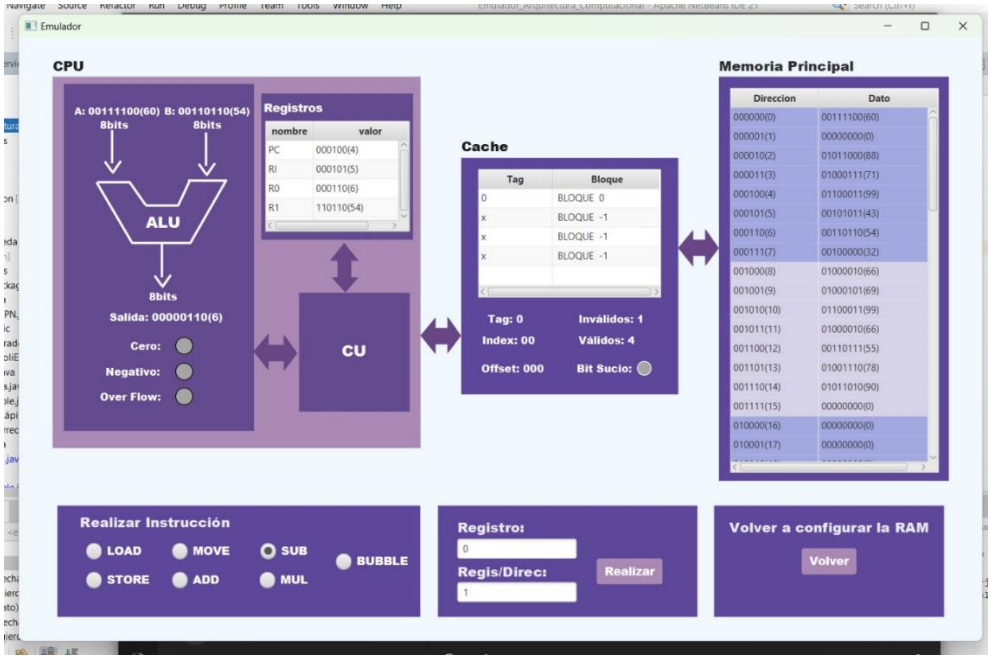

Figura 13 Resultado Resta

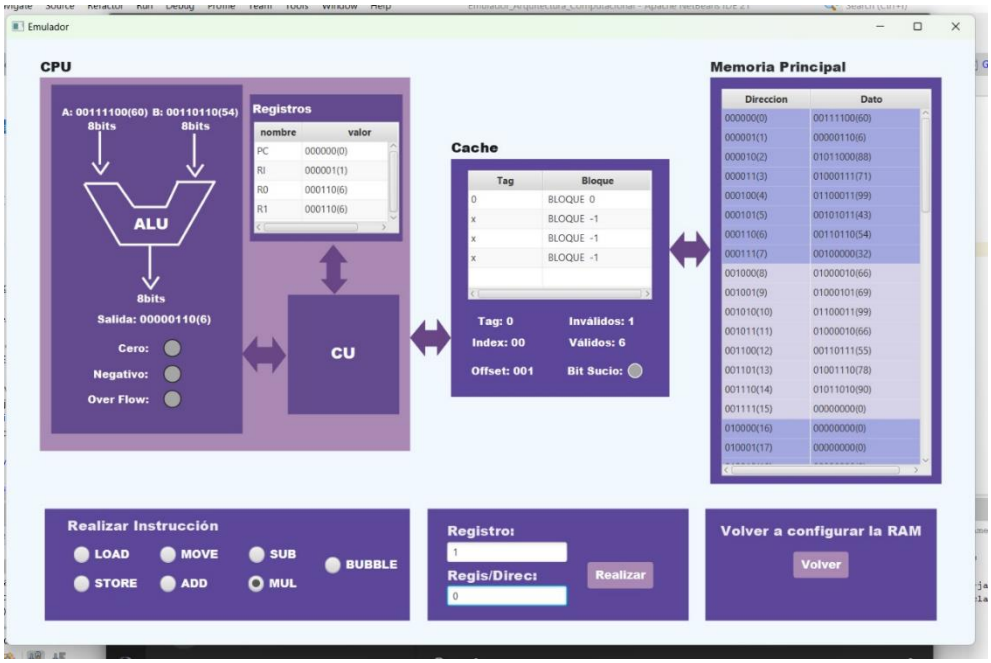

Figura 14 Operación multiplicación

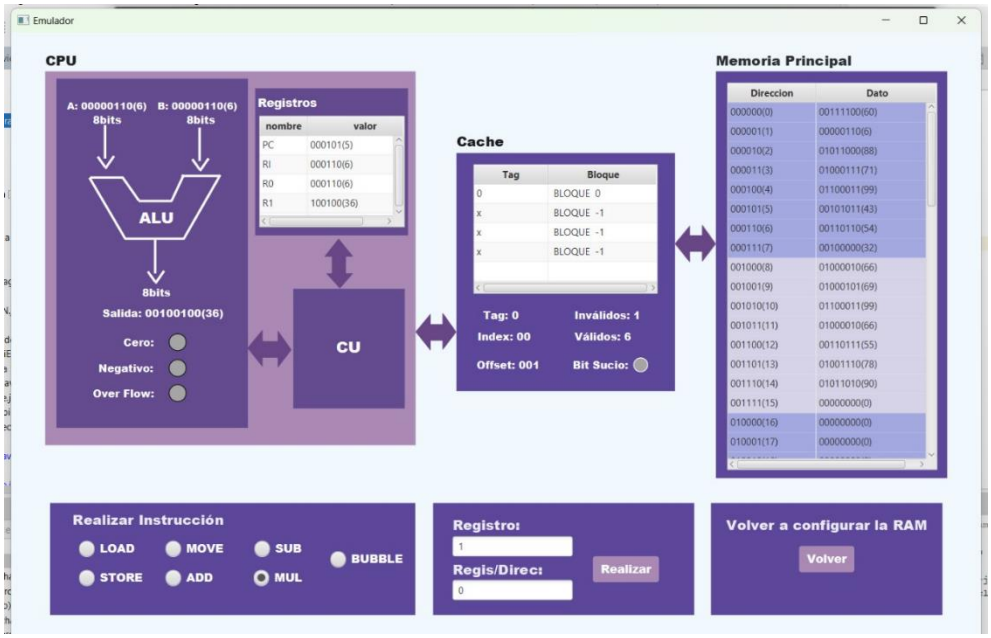

Figura 15 Resultado multiplicación

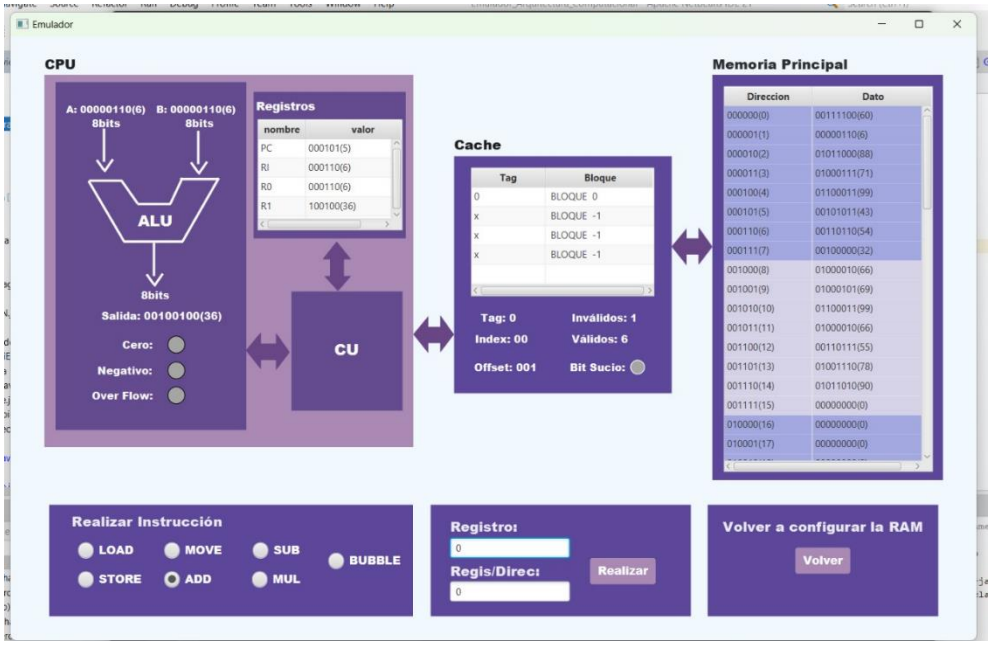

Figura 16 Operación suma

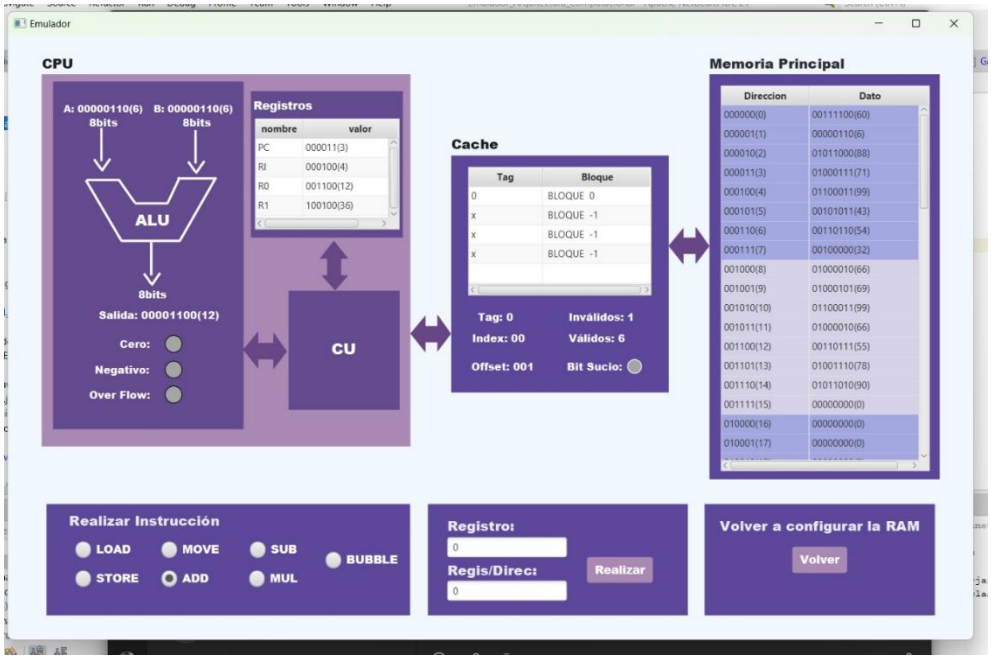

Figura 17 Resultado suma

El programa realizara las operaciones de suma, resta y multiplicación ejecutando las operaciones se realizarán gracias a las operaciones de carga las operaciones de restantes sirven para almacenar los datos en la memoria principal o mover los datos que copia los datos de un registro a otro. Las operaciones secundarias se ejecutarán de acuerdo con los datos que sean solicitados por el usuario, en este caso, se deberán cargar primero de la memoria principal hacia (RAM) y luego el bloque que contiene esa dirección del dato se copia hacia la memoria cache, la memoria cache pasa la dirección del dato que necesita para la ejecución, ya sea suma, resta o multiplicación. El proceso vuelve a repetirse para el o los datos que se necesiten para cada operación a realizar.

Cuando la operación se ha completado, el resultado de momento se guardará en el primer registro del número que se tomó para realizar la operación solicitada por el usuario, la opción de almacenar es una instrucción independiente a las operaciones de suma, resta y multiplicación, por ende, el resultado no será copiado y almacenado en la memoria principal.

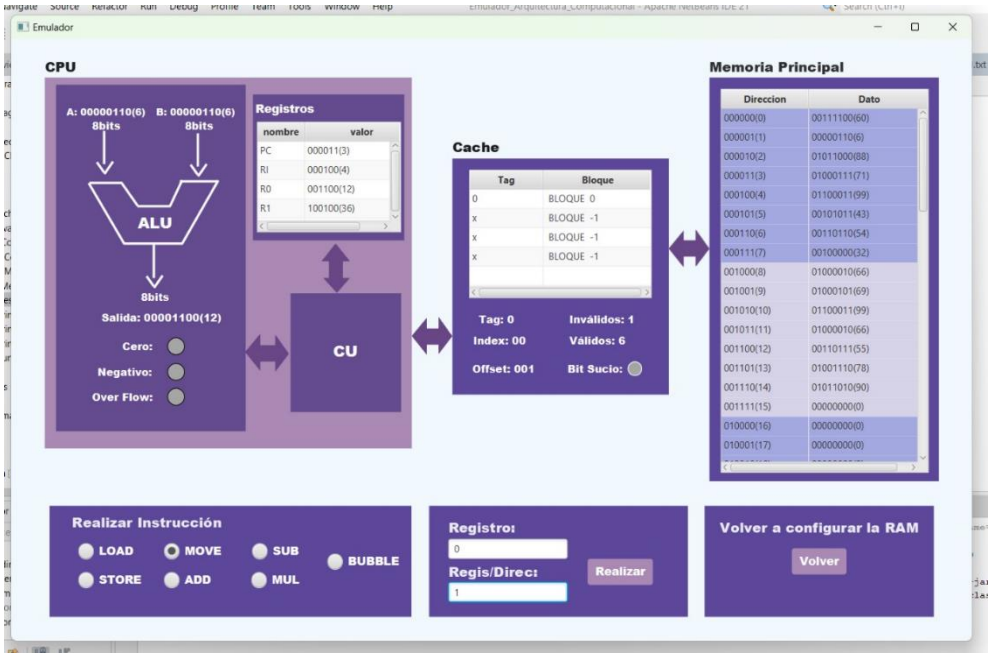

Figura 18 Operación mover

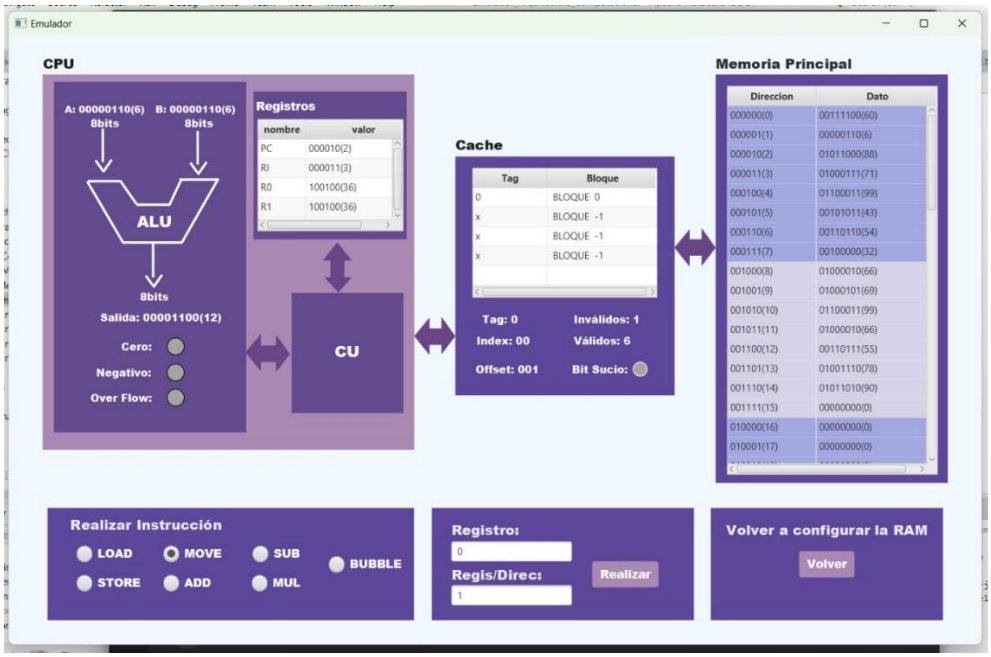

Figura 19 Resultado mover

El proceso es diferente para la ejecución de mover, inicialmente en uno de los dos registros ya debe existir un dato, pero de no ser cierto, se deberá cargar de la memoria cache, o caso contrario la cache solicitara la dirección del dato a la memoria principal, la cual enviara el bloque que contiene la dirección del dato a la cache.

La ejecución del proceso mover consiste en mover o copiar el dato que existe en un registro hacia el otro registro, aunque el almacenamiento de este dato deberá ser definido por el usuario, o en el caso que se necesite realizar otra operación, el usuario será el encargo de solicitarlo.

Conclusiones:

- Se realizó la implementación de una memoria intermedia en el modelo de Von Neumann, que ya se tenía de anteriormente, esto actúa como la memoria caché. Aunque a nivel de software no se puede apreciar la significancia de tener una memoria intermedia, el hecho de que toda computadora posea memoria caché muestra la importancia de esta. Así, el uso de una memoria caché no solo acelera la ejecución de los programas, sino que también mejora la eficiencia del sistema en general, por lo que se convierte en una parte esencial en la arquitectura de computadores. (Escrito por: Juan Cofre)
- Uno de los aspectos más valiosos que he podido extraer de este proyecto es que el esfuerzo por simular o emular el comportamiento de una arquitectura computacional no solo refuerza el conocimiento adquirido en clase, sino que también nos desafía a tomar decisiones fundamentadas frente a los problemas que surgen. Un ejemplo de esto es determinar qué ocurre cuando la ALU arroja condiciones como overflow o negative. Para garantizar que estas decisiones sean las más adecuadas, es fundamental que estén respaldadas por un conocimiento profundo y un dominio sólido del tema. (Escrito por: Fernando Huilca).
- La simulación de este sistema nos permite comprender exactamente cómo se gestionan los datos en un entorno informático real, desde el acceso rápido a la caché hasta las operaciones más lentas en la memoria principal. Simular estos procesos nos brinda la oportunidad de ilustrar el impacto del diseño de la caché y las estrategias de gestión de la memoria en el rendimiento global del sistema. Mediante una simulación de este tipo, podemos observar cómo interactúa la CPU con los distintos niveles de caché, cómo se producen los “hitts” y “miss” de caché y cómo actúa la memoria principal como respaldo cuando los datos no están en la caché. (Escrito por Jeremy Jiménez)
- La implementación de simulaciones dinámicas con la capacidad de modificar parámetros en tiempo real permite una evaluación más precisa y adaptativa del rendimiento del sistema. Esta metodología revela cómo cambios en la latencia de la memoria caché y la frecuencia de la CPU afectan el rendimiento en aplicaciones críticas, destacando la importancia de la flexibilidad en la configuración de sistemas embebidos para optimizar su funcionamiento en escenarios variables. (Escrito por Jhordy Parra)

Recomendaciones:

- La ampliación de las capacidades de un programa debe ser fundamental. Se puede mejorar en el manejo de las operaciones que se realizan, tomando en cuanta mayor cantidad de números. De forma especial, los números negativos serían un nivel más por encima de las operaciones básicas y que tanto la unidad central de procesamiento (CPU) como la unidad aritmético-lógica (ALU) puedan procesar sin que exista un error de por medio. (Escrito por: Juan Cofre)
- Es fundamental prestar mucha atención a la implementación y configuración de la memoria caché y la RAM. Idealmente, la caché debería contar con suficiente espacio para almacenar los bloques más críticos y con un mayor número de líneas en caché que en RAM. Esto permitirá apreciar mejor la importancia de la caché y su impacto positivo en la mejora del tiempo de ejecución. (Escrito por Fernando Huilca)
- Se consideró la posibilidad de añadir una interfaz gráfica o, al menos, un sistema de registro que pueda mostrar el flujo de datos entre la CPU, la memoria caché y la memoria principal en tiempo real. Esto puede ayudar a comprender de forma más intuitiva el comportamiento del sistema y cómo se ve afectado el rendimiento por los accesos a la memoria. (Escrito por Jeremy Jimenez)
- Se recomienda desarrollar herramientas de simulación con interfaces amigables, que faciliten la familiarización del usuario con el programa de manera intuitiva. Además, estas herramientas deben permitir la modificación en tiempo real de parámetros clave del sistema, para ofrecer una mayor flexibilidad y un análisis más detallado. La inclusión de características interactivas y visuales puede mejorar significativamente la experiencia del usuario y la eficacia en el uso del software. (Escrito por Jhordy Parra)

Bibliografía:

[1] “Arquitectura de Von Neumann,” Wikipedia, la enciclopedia libre. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Arquitectura_de_Von_Neumann. [Accedido: 17-jun-2024].
[2] F. Huilca, "Arquitectura de Von Neumann," GitHub. [En línea]. Disponible en: https://github.com/FernandoHuilca/Arquitectura_de_Von_Neumann. [Accedido: 10-agos-2024].
[3] AWS, «Información general sobre el almacenamiento en caché», Amazon Web Services, Inc. [En línea]. Disponible en: https://aws.amazon.com/es/caching/#:~:text=En%20inform%C3%A1tica%2C%

20una%20memoria%20cach%C3%A9,la%20ubicaci%C3%B3n%20de%20almacenamiento%20principal.

[4] R. Hernández, «Elementos de diseño de memoria caché». Accedido: 25 de julio de 2024. [En línea]. Disponible en: https://repository.uaeh.edu.mx/bitstream/bitstream/handle/123456789/14432/elementos_cache.pdf?sequence=1&isAllowed=y

[5] «Elementos de diseño de caché». Accedido: 25 de julio de 2024. [En línea]. Disponible en: https://lc.fie.umich.mx/~rochoa/Materias/PROGRAMACION/ORGANIZACION/TEMA_5_2.pdf

[6] «Tema 6. Memoria Caché». Accedido: 25 de julio de 2024. [En línea]. Disponible en: https://www.fdi.ucm.es/profesor/jjruz/WEB2/Temas/EC6.pdf

[7] D. Maldonado, «Arquitectura de Computadores - Sistemas de memoria». Accedido: 4 de agosto de 2024. [En línea]. Disponible en: https://aulasvirtuales.epn.edu.ec/mod/resource/view.php?id=9517605

[8] UMSNH, «Capitulo 5 - Memoria caché». Accedido: 15 de agosto de 2024. [En línea]. Disponible en: https://lc.fie.umich.mx/~rochoa/Materias/PROGRAMACION/ORGANIZACION/Notas_cap_5.pdf