Bitácora de Trabajo Proyecto 2 – Fundamentos de Arquitectura de Computadores

Kevin Lobo Juárez	Javier Tenorio Cervantes	Randall Bolaños López
2020087823	2020065308	2019043784

Bitácora de Trabajo - 8/11/2024

Actividades Realizadas:

1. **Inicio del Proyecto:**
   - Se estableció la estructura inicial del simulador, incluyendo los módulos principales:
     - `main.py`: Coordinador principal.
     - `pipeline.py`: Lógica del pipeline.
     - `visualization.py`: Representación gráfica con Pygame.
   - Se diseñó el flujo general del pipeline (`IF`, `ID`, `EX`, `MEM`, `WB`).
2. **Desarrollo Básico:**
   - Se implementaron las primeras funciones en `pipeline.py`:
     - Inicialización del pipeline (`initialize_pipeline`).
     - Simulación de un ciclo (`execute_cycle`) para las instrucciones básicas.
   - En `visualization.py`, se creó la base de la visualización, mostrando el pipeline, los registros y la memoria en una ventana.
3. **Pruebas Iniciales:**
   - Se cargó un programa con una instrucción `ADD` básica.
   - Se verificó la actualización de los registros y la memoria tras la ejecución.

Resultados:

- Se estableció un pipeline funcional básico con visualización gráfica.
- Se logró simular el flujo de una instrucción `ADD` en todas las etapas del pipeline.

Siguientes Pasos:

- Ajustar la lógica para manejar mejor la sincronización entre etapas.
- Introducir colores dinámicos y más instrucciones en el pipeline.

9/11/2024

Actividades Realizadas:

1. **Corrección de Errores:**
    - Se solucionó el problema donde IF no mostraba correctamente las instrucciones, introduciendo un retardo en la transferencia a ID.
    - Se corrigieron errores de sintaxis en los mensajes de depuración.
2. **Mejoras en Visualización:**
    - Se añadieron colores dinámicos para etapas activas y vacías.
    - Se reorganizó la visualización de los registros y la memoria en cuadrículas claras.
    - Se agregó un encabezado mostrando ciclo, tiempo y valor del PC.
3. **Reestructuración del Flujo:**
    - Se mejoró `main.py` para soportar simulaciones en consola o visualizaciones en tiempo real.
4. **Sincronización de Pipeline:**
    - Se corrigió el problema donde dos etapas del pipeline se mostraban como activas simultáneamente. Ahora solo una etapa está activa a la vez.
    - Se agregó lógica para limpiar cada etapa del pipeline después de mover una instrucción, asegurando una transición clara y precisa entre las etapas.

Resultados:

- Pipeline sincronizado y funcional, mostrando correctamente el estado en cada etapa.
- Interfaz gráfica clara y en sincronía con la simulación textual.

Siguientes Pasos:

- Implementar más instrucciones (SUB, LOAD, STORE) y manejar dependencias de datos.

- Documentar y optimizar el código.

Bitácora de Trabajo - 10/11/2024

Actividades Realizadas:

1. **Implementación de Nuevas Instrucciones:**
    - Se añadieron las instrucciones **SUB** (resta) y **MUL** (multiplicación) al pipeline.
    - Ajustes realizados en las siguientes áreas:
        - **pipeline.py**: Extensión de la lógica de `execute_cycle` para manejar las nuevas operaciones.
        - **registers.py**: Inicialización de valores específicos en los registros para probar SUB y MUL.
        - **main.py**: Inclusión de botones interactivos para las nuevas instrucciones en la interfaz gráfica.
2. **Manejo de Segmentación:**

    - Se detectó un problema inherente a la segmentación del pipeline que causaba conflictos al sobrescribir registros intermedios.
    - Ajustes realizados para garantizar que las instrucciones procesen los valores correctos en cada etapa.
3. **Pruebas y Validaciones:**
    - Verificación de la ejecución correcta de SUB y MUL en todas las etapas del pipeline.
    - Validación de resultados en los registros tras la ejecución:
        - SUB: R4 = R5 - R6.
        - MUL: R7 = R0 * R1.
    - Confirmación de que los resultados eran correctos en un entorno segmentado.
4. **Actualización de Visualización:**
    - Sincronización mejorada entre la ejecución y la representación gráfica.
    - Inclusión de colores dinámicos para las nuevas instrucciones SUB y MUL.

Resultados:

- El simulador ahora soporta tres instrucciones aritméticas: **ADD**, **SUB**, y **MUL**.
- Visualización gráfica sincronizada y representativa del estado del pipeline en tiempo real.
- Resolución de problemas de segmentación, garantizando resultados precisos en un entorno segmentado.

Bitácora de Trabajo - 11/11/2024

Actividades Realizadas:

Implementación de Nuevas Instrucciones:

- Se añadieron las instrucciones **LOAD** (carfa) y **STORE** (multiplicación) al pipeline.
- Ajustes realizados en las siguientes áreas:
    - memory_instructions.py: Nuevo archivo donde se definen las funciones LOAD y STORE.
    - **pipeline.py**: Extensión de la lógica de `execute_cycle` para el manejo de las instrucciones nuevas, además de limpiar o borrar los registros según se la instrucción correspondiente. Además, se agregó lógica de manejo de dependencias.
    - **main.py**: Inclusión de botones interactivos para las nuevas instrucciones en la interfaz gráfica.

Manejo de Segmentación:

- Se detectó un problema inherente a la segmentación del pipeline que provoca un índice inexistente tanto en registros como en direcciones de la memoria.
- Se realizaron ajustes para el correcto manejo y lógica secuencial de cada una de las instrucciones, además de dependencias adicionales que provocan saltos de procesos innecesarios para este tipo de instrucción.

Pruebas y Validaciones:

- Verificación de la ejecución correcta de **LOAD** y **STORE** en todas las etapas del pipeline.
- Validación de resultados en los registros tras la ejecución:
  - LOAD R7, MEM [3]
  - STORE MEM [3], R4

Optimización de Recursos:

En caso de la instrucción LOAD, el valor almacenado en memoria se limpia después de cargarse al registro de destino. De forma similar para la instrucción STORE, el registro que almacena el valor a destino para la memoria es limpiado después de ejecutar la instrucción.

Resultados:

- El simulador ahora soporta dos instrucciones de memoria: **LOAD**, **STORE**
- Visualización gráfica sincronizada y representativa del estado del pipeline en tiempo real.
- Resolución de problemas de manejo de índices, optimización y desplazamiento entre memoria y registros.

Siguientes Pasos:

- Continuar con el manejo de dependencias de datos y riesgos de pipeline para el resto de instrucciones.
- Implementar la lógica de procesamiento de instrucción según lo requerido (ciclos de tiempo, paso a paso, ciclo completo).
- Continuar documentando y optimizando el código base para futuras extensiones.

Bitácora de Trabajo - 12/11/2024

Actividades Realizadas:

1. **Implementación de Nuevas Instrucciones:**
    - Se añadieron las instrucciones **DIV** (división) y **MOD** (módulo) al pipeline.
    - **Ajustes realizados en las siguientes áreas:**
        - **pipeline.py:** Extensión de la lógica en `execute_cycle` para manejar las nuevas operaciones, asegurando el manejo correcto de las instrucciones DIV y MOD, incluso con divisores cero.
        - **registers.py:** Se inicializaron valores específicos en los registros para probar DIV y MOD, con énfasis en los casos especiales como divisor igual a cero.
        - **main.py:** Inclusión de botones interactivos para las nuevas instrucciones en la interfaz gráfica.
2. **Manejo de Segmentación:**
    - **Dependencias entre registros:** Se verificó que el resultado de operaciones previas no afectara incorrectamente a las instrucciones DIV y MOD.
    - **Sincronización del pipeline:** Se garantizaron transiciones limpias entre las etapas del pipeline para evitar conflictos en registros compartidos.
3. **Pruebas y Validaciones:**
    - **Pruebas funcionales:**
        - **DIV:** R4 = R5 ÷ R6.
        - **MOD:** R7 = R5 % R6.
    - **Validación de escenarios especiales:**
        - Caso de división por cero en **DIV**: Se aseguró que el resultado en estos casos fuera 0 sin provocar errores en la ejecución.
        - Caso de módulo con divisor cero en **MOD**: Similar a DIV, el resultado es 0 en estos casos.
    - **Verificación de los resultados:** Confirmación de valores precisos en los registros tras la ejecución de ambas instrucciones en un entorno segmentado.
4. **Actualización de Visualización:**
    - **Nuevas instrucciones:** Se añadió representación gráfica para DIV y MOD en las etapas del pipeline (IF, ID, EX, MEM, WB).
    - **Botones:** Se organizaron los botones en la interfaz gráfica para acomodar las nuevas instrucciones.
    - Se aseguraron los colores dinámicos y la representación visual correcta de las nuevas operaciones.

Resultados:

- El simulador ahora soporta dos nuevas instrucciones aritméticas: **DIV** y **MOD**.
- Visualización gráfica actualizada para incluir las nuevas instrucciones y mantener sincronía con el estado real del pipeline.
- Resolución de casos especiales como divisores cero para DIV y MOD, garantizando estabilidad y funcionalidad.

Siguientes Pasos:

- Continuar con la implementación de instrucciones de control de flujo (BNE, BEQ).
- Optimizar la representación gráfica para mejorar la claridad del uso de hardware durante la ejecución de cada instrucción.
- Integrar el manejo completo de dependencias de datos y riesgos de pipeline para un procesamiento más robusto.
- Finalizar la documentación del sistema completo.

Bitácora de Trabajo - 13/11/2024

Actividades Realizadas:

1. **Implementación de Nuevas Instrucciones:**
    - **Instrucciones de salto y control de flujo:**
        - Se añadieron las instrucciones **BNE** (Branch if Not Equal), **BEQ** (Branch if Equal), y **SWAP** al pipeline.
    - **Ajustes realizados en las siguientes áreas:**
        - **pipeline.py:**
            - Extensión de la lógica en `execute_cycle` para manejar las nuevas instrucciones:
                - **BNE**: Salto condicional si los valores de los registros especificados son distintos.
                - **BEQ**: Salto condicional si los valores de los registros especificados son iguales.
                - **SWAP**: Intercambio directo entre dos registros.
            - Actualización del **Program Counter (PC)** para reflejar correctamente los saltos condicionales durante las instrucciones **BNE** y **BEQ**.
        - **registers.py:**
            - Se reservó el registro `R8` para visualizar el estado del PC.
        - **main.py:**
            - Inclusión de botones interactivos para las nuevas instrucciones en la interfaz gráfica.

        - Ajustes para soportar múltiples filas de botones en la interfaz.
2. **Manejo de Segmentación y Control:**
    - **Dependencias entre registros:** Se aseguraron transiciones limpias entre las etapas del pipeline para evitar conflictos al ejecutar instrucciones que dependen de resultados previos.
    - **Sincronización del PC:**
        - Se garantizó que el registro `R8` refleje el estado actual del PC en todo momento, incluidas las instrucciones de salto condicional.
        - Validación de comportamientos en escenarios donde el salto no es tomado.
3. **Pruebas y Validaciones:**
    - **BNE y BEQ:**
        - Pruebas funcionales para validar los saltos condicionales:
            - **BNE:** Salto tomado cuando `src1` ≠ `src2`.
            - **BEQ:** Salto tomado cuando `src1` = `src2`.
        - Validación de comportamiento cuando los saltos no son tomados.
    - **SWAP:**
        - Validación del intercambio correcto de valores entre dos registros.
    - **Representación gráfica:**
        - Confirmación de que las nuevas instrucciones se visualizan correctamente en cada etapa del pipeline (IF, ID, EX, MEM, WB).
    - **Resultados en consola:**
        - Confirmación del estado del PC a través de `R8` en cada ciclo.
4. **Actualización de Visualización:**
    - **Encabezado:** Se añadió el valor de `R8` (PC) en tiempo real para proporcionar un contexto completo durante la ejecución.
    - **Botones:** Organización en múltiples filas para acomodar las nuevas instrucciones.
    - **Colores dinámicos:** Representación visual mejorada de las instrucciones en cada etapa.

Resultados:

- El simulador ahora soporta tres nuevas instrucciones: **BNE**, **BEQ**, y **SWAP**.
- **Visualización gráfica:**
    - Sincronización completa con el estado real del pipeline y del PC (a través de `R8`).
    - Representación clara del estado de hardware durante cada ciclo.
- **Validación funcional:** Se confirmó el comportamiento correcto de los saltos condicionales y del intercambio de registros.

Siguientes Pasos:

- **Optimización del manejo de dependencias:** Continuar afinando las transiciones entre etapas del pipeline.
- **Representación gráfica:** Mostrar visualmente los registros y memoria utilizados en cada instrucción.
- **Implementación adicional:**
    - Continuar con más instrucciones de control de flujo si es necesario.
    - Finalizar pruebas exhaustivas para garantizar la estabilidad del sistema.
- **Documentación:** Actualizar la documentación completa del sistema y del conjunto de instrucciones soportadas.

Bitácora de Trabajo - 16/11/2024

Actividades Realizadas:

1. **Implementación de la Ejecución completa en el Pipeline**
2. **Implementación incompleta del StepByStep de la ejecución Pipeline**

Siguientes Pasos:

- **Optimización del manejo de dependencias:** Continuar afinando la ejecución entre las transiciones entre etapas del pipeline.
- **implementación de la Ejecución de ciclos por una unidad de tiempo.**
- **Representación gráfica:** Mostrar visualmente los registros y memoria utilizados en cada instrucción, además de modificarla para que funcione con el StepByStep.

Bitácora de Trabajo 23/11/24

Actividades realizadas:

1. **Creación del mockup de la ventana de simulación (etapas, información) se utilizó draw.io.**
2. **Creación del mockup del historial de ejecuciones se utilizó draw.io.**
3. **Arreglo de la ejecución step by step.**
4. **Trabajo en la documentación.**

Siguientes pasos:

- **Implementación de la ejecución de ciclos por una unidad de tiempo**
- **Optimización de los datos que ven en la ventana**

Historial de Ejecuciones

N°	Configuración	Tiempo	CPI	IPC	Frecuencia	Configuración	Tiempo	CPI	IPC	Frecuencia
1	Item 1	Item 1	Item 1	Item 1	Item 1	Item 1	Item 1	Item 1	Item 1	Item 1
2	Item 2	Item 2	Item 2	Item 2	Item 2	Item 2	Item 2	Item 2	Item 2	Item 2
3	Item 3	Item 3	Item 3	Item 3	Item 3	Item 3	Item 3	Item 3	Item 3	Item 3
4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
5	Item 5	Item 4	Item 4	Item 4	Item 4	Item 5	Item 4	Item 4	Item 4	Item 4
6	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
7	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
8	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
9	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
10	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
11	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
12	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4	Item 4
.										
.										
.										

Figura 1. Mockup de la ventana de historial de ejecuciones creada con draw.io.

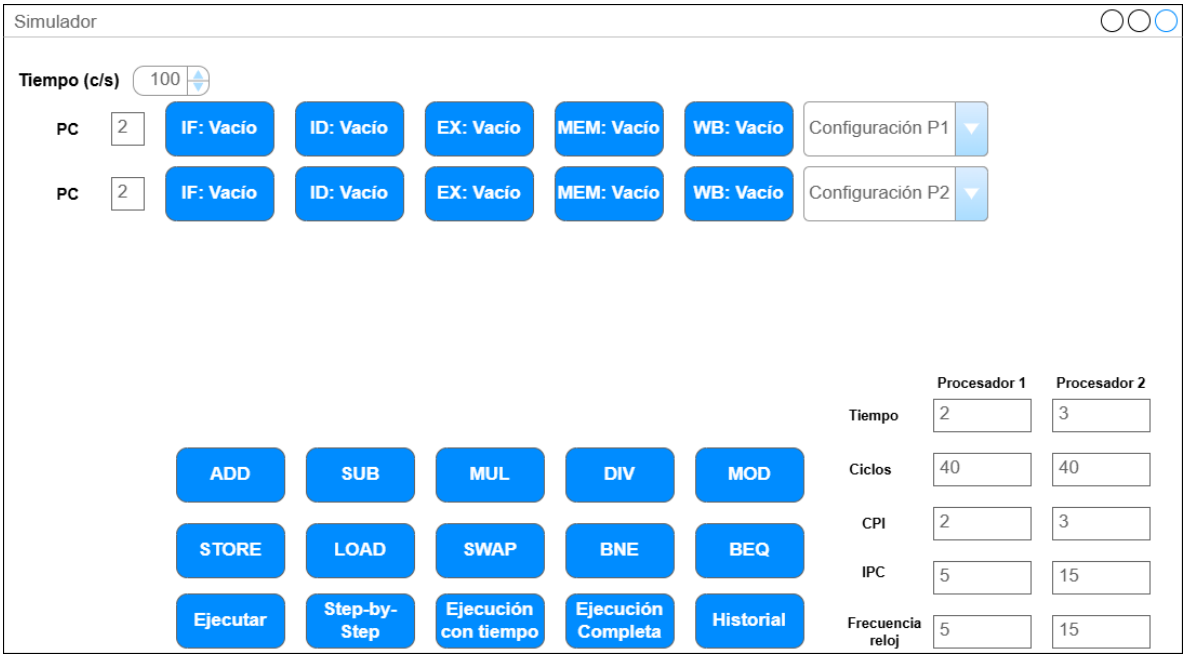

Figura 2. Mockup de la ventana del simulador de ejecuciones creada con draw.io.

Bitácora de Trabajo 24/11/24

Actividades realizadas:

5. **Continuación del trabajo para arreglar el modo de ejecución ciclos step by step.**
6. **Implementación de la ejecución de ciclo por una unidad de tiempo.**
7. **Implementación de la ejecución completa**

8. **Añadir el Unit Hazard**
9. **Diagrama general.**
10. **Trabajo en la documentación.**

Siguientes pasos:

- **Recolección de datos para verificar un correcto funcionamiento de los modos del procesador.**
- **Realizar arreglos en la ejecución step by step, ejecutar, y ejecución completa.**
- **Completar el Unit Hazard**
- **Terminar documentación y diagramas o mockups pendientes**

Bitácora de Trabajo 25/11/24

Actividades realizadas:

11. **Continuación del trabajo para arreglar el modo de ejecución ciclos step by step.**
12. **Implementación de la ejecución de ciclo por una unidad de tiempo.**
13. **Implementación de la ejecución completa**
14. **Añadir el Unit Hazard**
15. **Diagrama general.**
16. **Trabajo en la documentación.**

Siguientes pasos:

- **Recolección de datos para verificar un correcto funcionamiento de los modos del procesador.**

Bitácora de Trabajo 26/11/24

Actividades realizadas:

1. **Comprobación de ciertos casos, donde el programa funciona correcto como lo es modo básico, modo de predicción y modo completo, mientras que en el forwarding tiene problemas con BEQ BNE.**
2. **Se añade una ventana para el historial de ejecuciones.**
3. **Se distribuye la información más organizada para que se pueda leer sin problemas.**

Consulte el avance desde Github: https://github.com/Randall-BL/RBolanos-compu-archi-found-2G1-2023.git