Informe final para la entrega del procesador monociclo.

Asignatura: Arquitectura de computadores

Docente: Juan Manuel Velásquez Isaza

Presentado por: Juan Esteban García Pulgarín y Juan Pablo Sánchez Zapata

➔ Se implementa un procesador monociclo basado en la arquitectura RISC-V de 32 bits para instrucciones de Assembler generadas por un compilador para lenguaje C++.

1. Diseño circuital en la aplicación *"Digital"*

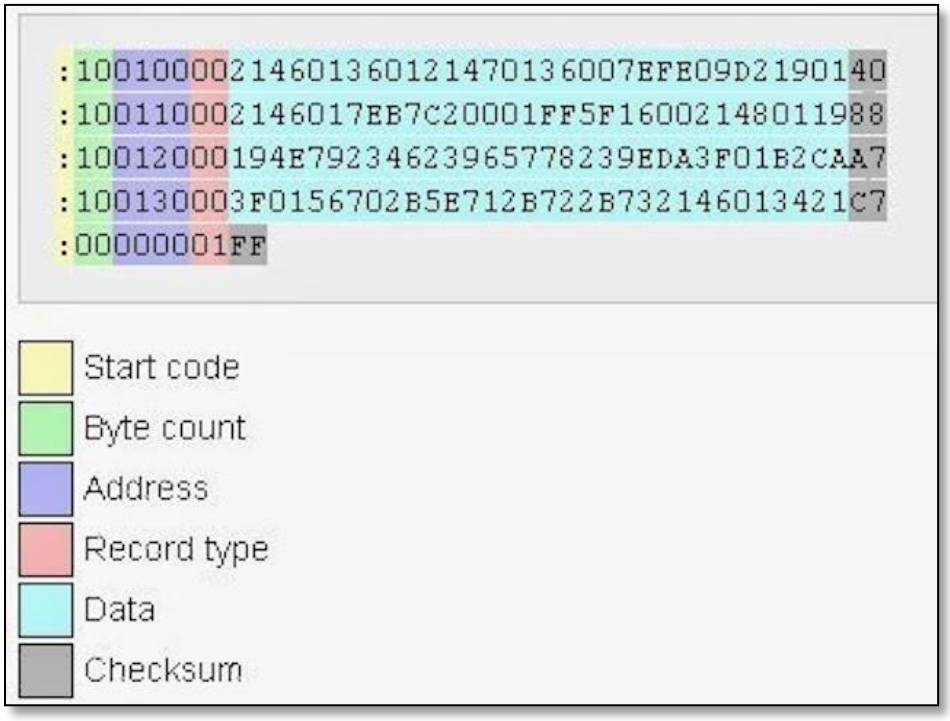
En efecto, el funcionamiento del procesador en la simulación sigue la lógica planteada en el camino de datos presentado en clase, sin embargo, para la implementación de un procesador capaz de procesar el código en C++ para calcular el factorial de un número usando ciclos.

1.1.RETOS ENFRENTADOS

- **Manejo de las instrucciones:** En Digital sólo existe un componente para la memoria de datos que es la RAM de 32 bits. El parámetro de direccionamiento que sigue esta memoria es de 24 bits a lo mucho, sin embargo, esto se pudo solucionar generando un componente personalizado con los componentes internos de la RAM original que viene por defecto con Digital. En este proceso de manejo de instrucciones, dado que la RAM solo recibe los datos a escribir de manera manual, es mejor usar *la memoria del programa* que tiene Digital.

- **Memoria del programa:** La memoria del programa se traduce a la memoria de instrucciones, y si se le carga la memoria con un archivo de extensión '. hex' el software lo reconoce. Por lo tanto, para la memoria de instrucciones no es necesario utilizar una RAM de 32 bits, sino que se puede precargar la memoria. En la memoria de datos si es necesario usar la RAM de 32 bits ya que esta permite lectura y escritura.

- **Formato de las instrucciones de carga a la memoria del programa:** Estas instrucciones no se pueden cargar solo en hexadecimal, sino que es necesario usar el formato Intel hex.



Claro en la imagen, se evidencia que la instrucción en realidad solo es la parte azul del formato. Lo demás es necesario para digital, tal como los ':', la cantidad de bytes en que se va a almacenar la instrucción, la dirección, el tipo de dato y la instrucción. Finalmente se calcula el checksum para asegurar que la línea no se modificó inconscientemente o que no tenga errores. La línea final, siempre es necesaria para indicar el final del archivo '. hex'.

- **Direccionamiento:** Si en la memoria la dirección se cuenta de a 4 bits, en realidad en digital el sumador del PC debe incrementarse de 1 en 1 dado que la memoria de instrucciones es una ROM para este diseño. Esto tiene un problema y es que se pueden generar números impares en la dirección y esto es complicado para cuando haya saltos, ya que las instrucciones de salto saltan de a 4 generalmente, y es imposible traducir por ejemplo beq x2, x3, 31, porque 31 no es un múltiplo de 4. Para solucionar este problema simplemente se hace una regla de tres simple, necesitamos entonces que el PC cuente de a 4, lo cual quiere decir que si con las instrucciones almacenando en las direcciones de a 4 bits, el sumador debe ser de a 1, entonces si se cuenta de 4 en el sumador, las instrucciones se deben ir almacenando en direccionas de a 16 bits, es decir: 2 bytes. Esto soluciona el problema y permite hacer los saltos con tranquilidad.
- **Checksum y generación del archivo '.** Hex'**:** El cálculo del checksum es algo tedioso a mano, ya que se debe separar cada línea de cada instrucción en el formato Intel hex en partes de 2 recuentos, por ejemplo se tiene: ":04000000ff56de32" Se

debe separar de a dos recuentos y sumarlos: 04+00+00+00+ff+56+de+32. El resultado en hexadecimal de esto se le toman los 8 bits menos significativos y el complemento A2 de este número es el nuevo checksum.

También, para códigos que generan muchas instrucciones, se hace largo escribir línea por línea, así que, para solucionar este problema de ineficiencia, se creó un código en Python que genera el archivo '. hex' solo ingresándole todas las instrucciones en hexadecimal. Más adelante se dará un manual de uso.

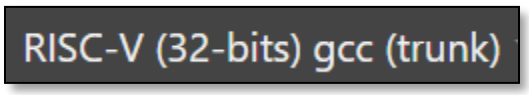
- **Cuidado con las instrucciones de salto incondicional:** Para estas, sólo fue necesario modificar en el control unit, que cuando el registro de destino para una de estas instrucciones sea x0, entonces deshabilita la escritura en el register unit.

1.2. USO

a. Primero hay que generar un código en lenguaje C++. Para este caso, el código principal usado fue el código para calcular el factorial de un número:

```cpp
int factorial(){
    int n = 10;
    if(n <= 1){
        return 1;
    }

    int result = 1;

    while (n > 1){
        result *= n;
        n--;
    }

    return result;
}
```

b. Llevar este código al compilador para C++: https://godbolt.org. En el compilador debe seleccionar el compilador:



c. Tomar todas las instrucciones en ensamblador que allí se generan y llevarlas a: https://riscvasm.lucasteske.dev (Las etiquetas no deben llevar paréntesis)

d. Luego, esta herramienta genera las instrucciones solo en hexadecimal que se deben llevar al código en Python 'formatAuto.py' y en el main, pegar estas instrucciones en el espacio asignado.

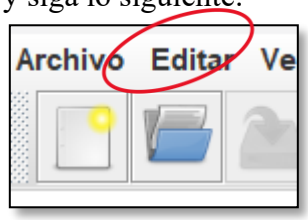
e. El archivo tiene una variable que es la dirección de la carpeta donde desea guardar el archivo '. Hex'.

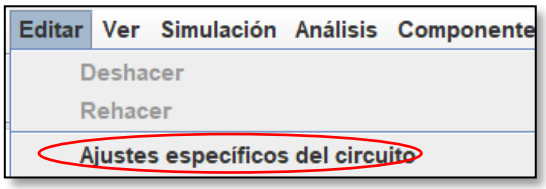
```
# Ruta de la carpeta donde se guardará el archivo
folder_path = r"C:\\Users\\usuario\\Documents\\UTP-Juan\\SEMESTRE VI\\AC\\Monocycle\\Componentes\\procesador"

# Generar y guardar el archivo
generator = IntelHexGenerator()
generator.save_to_file(instructions_block, folder_path)
```

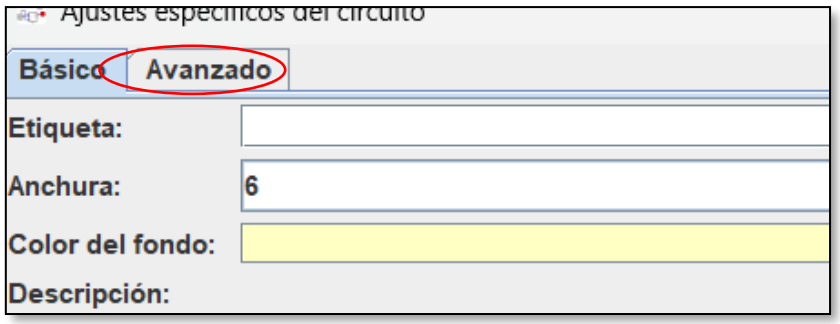
Nota: Copie la ruta de la carpeta donde desee guardar el archivo, si no funciona como la copia, entonces coloque cada \ de manera doble y ejecute:

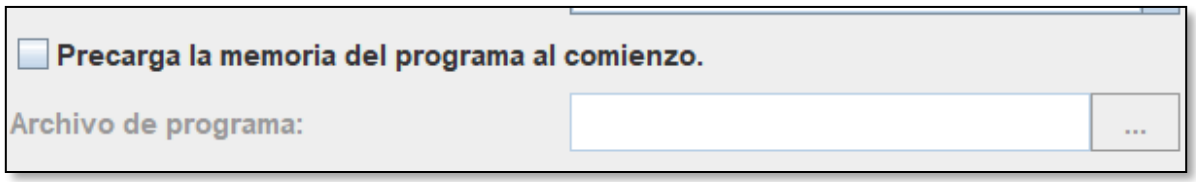
'python formatAuto.py'

f. Con las instrucciones generadas, vaya a digital y abra el archivo Monocycle-Style.dig y siga lo siguiente:









Seleccione la casilla de precarga de la memoria del programa y escoja la dirección donde esta el archivo generado por el programa formatAuto.py. Recuerde seleccionar la opción de big endian.

Ya que, con esta opción se asegura de que el software lea las instrucciones de izquierda a derecha.

1.3. VISUALIZACIÓN

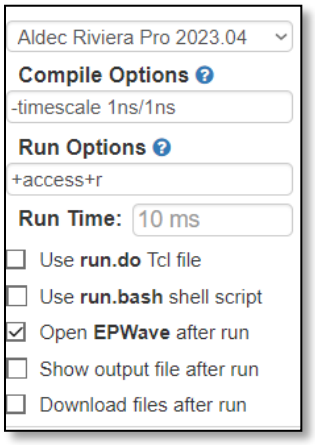
Puede ver la salida de la ALU, para saber que se está realizando en cada instrucción, sin embargo, finalmente, hay otro display que muestra lo que hay en el registro 10, que es el registro donde se guarda el resultado final. El procesador ¡funciona!

2. Descripción de hardware con *"Verilog"*

Con el diseño de los componentes no hubo mucho problema, solo en la carga de instrucciones y la forma en que se leían. Hubo inconvenientes con la forma en que se guardaban los registros sin embargo solo eran imprecisiones que eran de atención nada más. Uno de los principales retos fue la visualización de la salida. El IDE usado para Verilog fue EDA Playground: https://www.edaplayground.com. El Wave que usa este IDE es poco intuitivo, sin embargo genera un print con los resultados de varios aspectos como la instrucción que se está ejecutando y qué operación se está efectuando.

2.1. USO

Monte en su IDE de preferencia, (si es posible en EDA) cada componente de Verilog para el procesador. El archivo 'design' es el archivo principal que conecta todo. En la parte izquierda del EDA coloque el testbench, y simplemente active la opción de abrir el Wave y asegúrese de escoger el simulador.



Finalmente el wave se abre y muestra las señales, ejecutará el factorial de 5, que debe ser 78 en hexadecimal. ¡Verilog funciona!

Salvedades: Abstenerse de ejecutar alguno de estos pasos hará que no funcione correctamente como se espera. Las salidas en Digital se muestran en hexadecimal para el factorial de 10, por lo que si desea saber si funciona correctamente debe convertir el valor en hexadecimal que allí se visualiza y compararlo a su gusto, para darse cuenta de que funciona correctamente.

Nota: Descargue el repositorio en github: https://github.com/jpsz2004/ProcesadorMono-RISCV.git Este se descarga en ZIP, esta será la carpeta que usted abra en Digital, pero recuerde verificar (solo si va a crear un archivo '.hex' nuevo) que el folder path tenga la ruta actualizada.

Anexos: El archivo con extensión '.xlsx' es el archivo usado para construir la unidad de control, allí se tiene de manera específica para cada instrucción, el valor de cada señal.

Bibliografía

- Arquitectura de Computadoras con RISC-V. José Alfredo Jaramillo Villegas, Ph.D. Hernán Mauricio Zuluaga Bucheli, Camilo Sepúlveda Caviedes
- RISCV-Card
- Arquitectura de Computadores Clase: Camino de Datos Monociclo José Alfredo Jaramillo Villegas, Ph.D.