

Arquitectura de computadoras 2020

Trabajo Práctico N° 4 - PIPELINE PROCESADOR MIPS SIMPLIFICADO

Docentes:

Pereyra Martin (martinmiguelpereyra@gmail.com)

Pinzani Paulo (ppinzani89@gmail.com)

Rodriguez Santiago (srodriguez@unc.edu.ar)

Alumnos:

Bosack Facundo Matias (facubosack@mi.unc.edu.ar)

Pichetti Augusto (augusto.pichetti@mi.unc.edu.ar)

Diciembre, 2021

Introducción

En el presente informe se detalla la implementación del pipeline de 5 etapas del procesador MIPS simplificado, que corresponde al trabajo final de la materia de Arquitectura de Computadoras.

Consigna

Implementar el pipeline de 5 etapas del procesador MIPS

Requerimientos

Implementar el Procesador MIPS Segmentado en las siguientes Etapas:

- IF (Instruction Fetch): Búsqueda de la instrucción en la memoria de programa.
- ID (Instruction Decode): Decodificación de la instrucción y lectura de registros.
- EX (Execute): Ejecución de la instrucción.
- MEM (Memory Access): Lectura o escritura desde/hacia la memoria de datos.
- WB (Write back): Escritura de resultados en los registros.

Instrucciones a implementar

- R-type:
 - SLL, SRL, SRA, SLLV, SRLV, SRAV, ADDU, SUBU, AND, OR, XOR, NOR, SLT
- I-Type:
 - LB, LH, LW, LWU, LBU, LHU, SB, SH, SW, ADDI, ANDI, ORI, XORI, LUI, SLTI, BEQ, BNE, J, JAL
- J-Type:
 - JR, JALR

Riesgos

El procesador debe tener soporte para los siguientes tipos:

- **Estructurales.** Se producen cuando dos instrucciones tratan de utilizar el mismo recurso en el mismo ciclo.
- **De datos.** Se intenta utilizar un dato antes de que esté preparado. Mantenimiento del orden estricto de lecturas y escrituras.

- **De control.** Intentar tomar una decisión sobre una condición todavía no evaluada.

Para dar soporte a los riesgos nombrados se debe implementar las dos unidades riesgos:

- Unidad de Cortocircuitos
- Unidad de Detección de Riesgos

Otros Requerimientos

- El programa a ejecutar debe ser cargado en la memoria del programa mediante un archivo ensamblado.
 - Debe implementarse un programa ensamblador que convierte código assembler de MIPS a código de instrucción.
 - Debe transmitirse ese programa mediante interfaz UART antes de comenzar a ejecutar
- Se debe simular una unidad de Debug que envíe información hacia y desde el procesador mediante UART.

Debug Unit

Se debe enviar a través de la UART:

- Contenido de los 32 registros
- PC
- Contenido de la memoria de datos usada
- Cantidad de ciclos de clock desde el inicio

Modos de operación

Antes de estar disponible para ejecutar, el procesador está a la espera para recibir un programa mediante la Debug Unit.

Una vez cargado el programa, debe permitir dos modos de operación:

- **Continuo:** se envía un comando a la FPGA por la UART y esta inicia la ejecución del programa hasta llegar al final del mismo (Instrucción HALT). Llegado ese punto se muestran todos los valores indicados en pantalla.

- **Paso a paso:** Enviando un comando por la UART se ejecuta un ciclo de Clock. Se debe mostrar a cada paso los valores indicados.

Implementación

La implementación del procesador se realizó modularizando cada etapa del pipeline. Cada una tiene su propio módulo principal compuesto por registros y submódulos que lo conforman. A continuación se muestra un diagrama general del procesador detallando cada etapa separada.

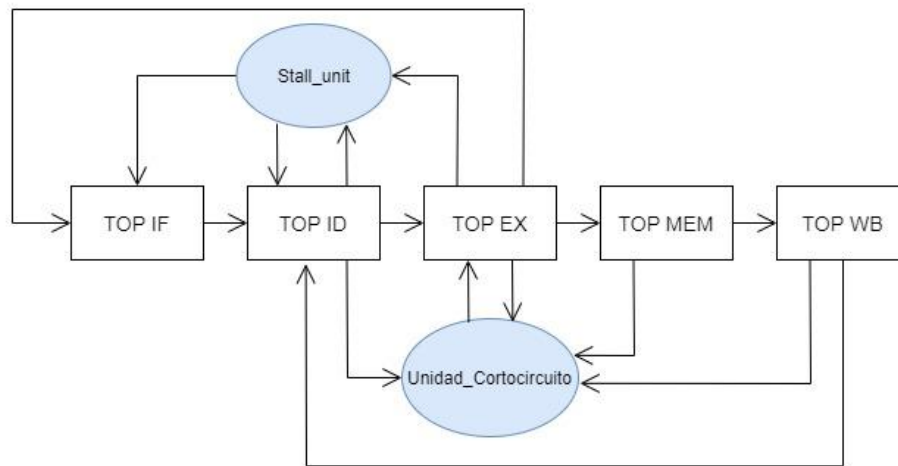


Figura 1: Diagrama de bloques MIPS

Cada módulo tuvo su propio desarrollo, por lo tanto a continuación se detalla la implementación de cada uno por separado:

Módulo TOP_IF

Este módulo corresponde a la primera de las 5 etapas del pipeline. Y hace referencia a la Búsqueda de la instrucción en la memoria de programa (Instruction Fetch). Va a estar compuesto por 4 submódulos:

1. **Multiplexor:** Se encarga de multiplexar las distintas posibles entradas del PC
2. **Program Counter (PC):** Contador de programa para la obtención de instrucciones. Va a ser la entrada de la memoria de instrucciones y va a direccionar la misma.
3. **Sumador:** Utilizado para actualizar el valor del contador de programa y obtener la dirección de la siguiente instrucción.
4. **Memoria Instrucciones:** Utilizada para el almacenamiento y lectura de las instrucciones de 32 bits del programa.

La memoria de instrucciones puede estar habilitada para escritura o para lectura y según corresponda, el multiplexor va a seleccionar como entrada del PC a la salida del sumador que va a ser el PC + 1, el PC más un offset si es que llego algun salto, el índice de la instrucción y por último el PC_register que va a corresponder a si la instrucción anterior fue un salto directo.

Los inputs y outputs de este módulo se pueden observar en la figura 2 y esta etapa concluye enviando por la salida del TOP_IF la instrucción de memoria de 32 bits correspondiente.

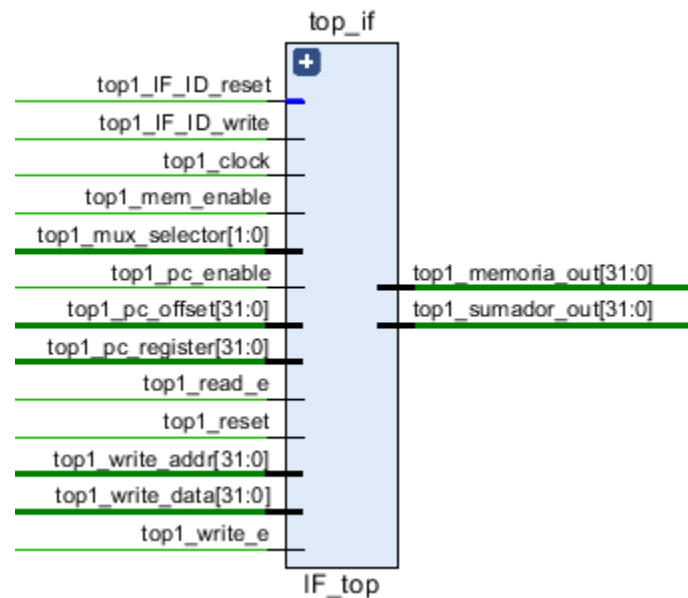


Figura 2: Inputs/Outputs de TOP_IF

Módulo TOP_ID

Este módulo corresponde a la segunda etapa del pipeline y se encarga de la decodificación de la instrucción y lectura de registros. Hay que tener en cuenta en la decodificación los 3 tipos de instrucciones que se van a implementar, donde cada uno de los 32 bits representa lo visto en la Figura 3:

1. **R-type:** SLL, SRL, SRA, SLLV, SRLV, SRAV, ADDU, SUBU, AND, OR, XOR, NOR, SLT
2. **I-Type:** LB, LH, LW, LWU, LBU, LHU, SB, SH, SW, ADDI, ANDI, ORI, XORI, LUI, SLTI, BEQ, BNE, J, JAL
3. **J-Type:** JR, JALR

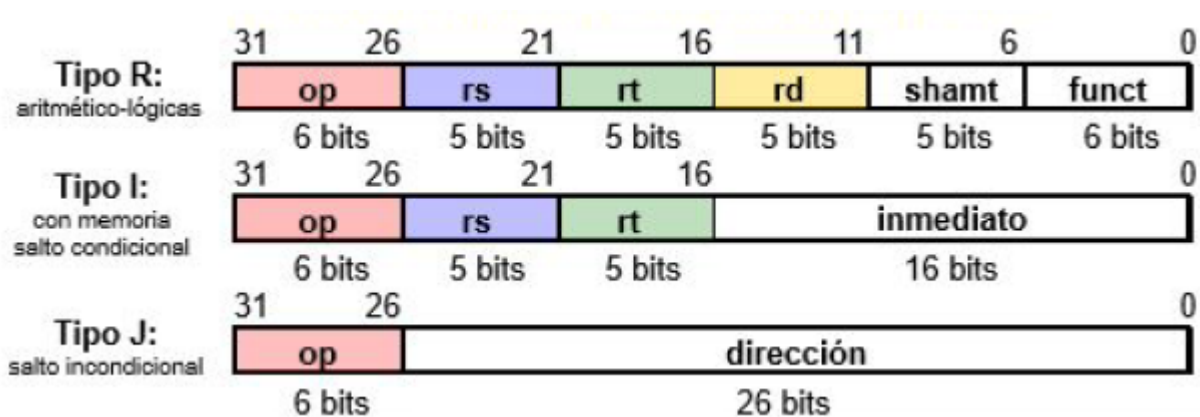


Figura 3: 3 tipos de instrucciones

Una vez que la instrucción se decodifica, las señales de control se encargan de decirle al procesador que se comporte de una u otra forma según cual sea la operación que se necesite realizar. Además esta etapa cuenta con un banco de 32 registros que se utiliza tanto para leer como para escribir, entonces se podrá acceder a los mismos antes de ejecutar la instrucción.

Este módulo va a estar compuesto por 3 submódulos:

1. **Banco de registros:** Es un banco que va a contener 32 registros de 32 bits cada uno. Se va a poder realizar lectura y escritura de cada registro y en el caso que se requiera escribir (de la etapa de WB) y leer (ID) en el mismo ciclo y registro, se realiza la lectura del dato a escribir (para evitar dependencias con la instrucción en la etapa de Wb).
2. **Extensión de signo:** Es un módulo que cambia el tamaño de la entrada a 32 bits.
3. **Unidad de control:** Módulo encargado de generar las señales de control necesarias para las etapas EX, MEM y WB, para la ejecución de la instrucción correspondiente. Si las instrucciones son saltos, la unidad de control incluye la lógica. Además esta unidad de control también posee la lógica para la detección de instrucciones de halt o desconocidas la cual generará señales de control nulas, impidiendo la modificación de los registros del procesador. Las señales de Control son las siguientes:
 - a. **o_pc_src:** Selecciona el siguiente valor del contador del programa según corresponda.
 - b. **o_reg_dst:** Selecciona la dirección de escritura para registrar la memoria.
 - c. **o_Alu_src:** Selecciona el segundo operando de alu.
 - d. **o_reg_wr:** Selecciona la habilitación de escritura de la memoria de registros.
 - e. **o_mem_to_reg:** Selecciona los datos de escritura para registrar la memoria.
 - f. **o_mem_rd:** Selecciona la habilitación de lectura de la memoria de datos.
 - g. **o_mem_wr:** selecciona la habilitación de escritura de la memoria de datos.

- h. **o_branch:** va a una gate y con un indicador cero de alu para seleccionar el siguiente valor de PC.
- i. **o_alu_op[2:0]:** Selecciona los códigos de tipo de instrucción que van al módulo alu_control que luego se encarga de seleccionar la operación alu correspondiente al módulo alu.

En la Figura 4 se pueden observar los inputs y outputs de este módulo:

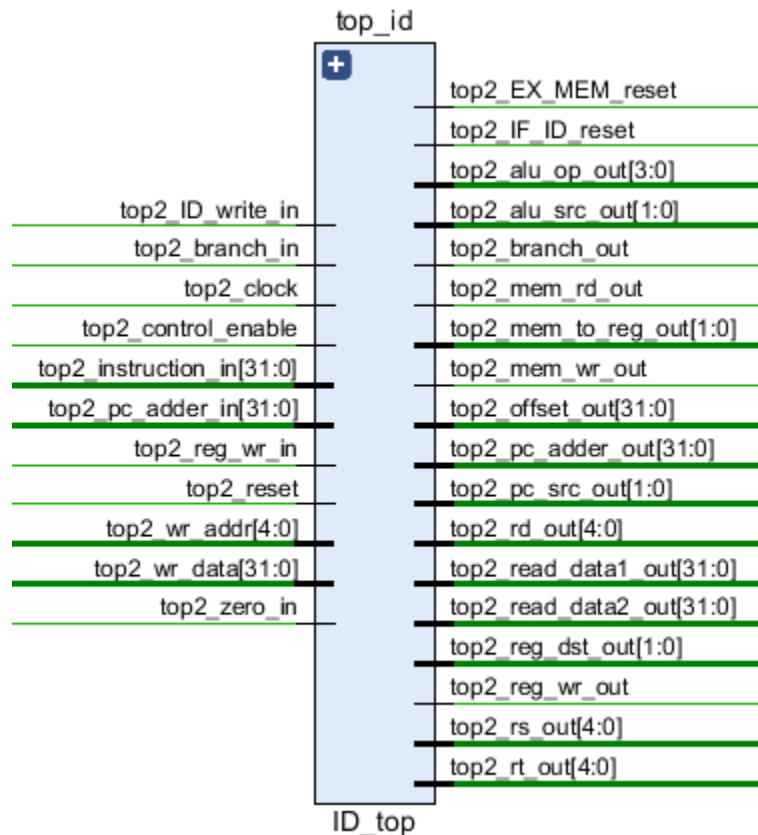


Figura 4: Inputs/Outputs de TOP_ID

Módulo TOP_EX

Este módulo corresponde a la tercera etapa y es el encargado de la ejecución de la instrucción. Una vez que se decodifica la instrucción en la etapa anterior y se obtienen los operandos, el código de la operación y los datos antes mencionados, este módulo se encarga de realizar las operaciones correspondientes. Va a estar compuesto por 8 submódulos:

1. **ALU:** Unidad Aritmetica Logica combinacional que efectúa la operación solicitada entre los operandos en función de la señal de operación que corresponda.

2. **ALU Control:** Módulo encargado de generar la señal de control para especificar la operación a realizar en la ALU, utiliza como referencia la señal que se decodifica en la etapa anterior.
3. **Multiplexores 1, 2, 3, 4 y 5:** Estos multiplexores van a generar la lógica adecuada para los operandos de la entrada de la ALU, seleccionando según sea necesario el cortocircuito para las dependencias, además de la selección de los distintos segmentos de instrucción (rt,rd) para establecer las direcciones de escritura en la etapa de WB.
4. **Sumador:** Utilizado para sumar el pc con el offset ,para las instrucciones de salto condicional o branch.

Luego de la decodificación de la instrucción, las salida del banco de registros va a arrojar 2 datos distintos, los mismos van a pasar por la lógica de los multiplexores y según sea la instrucción, dependiendo de la unidad de cortocircuitos, se va a llegar a tener dos operandos en la ALU, por lo tanto, según sea el código de operación que le pase el control de la ALU a la ALU, la misma va a realizar la operación de manera combinacional y va a arrojar su resultado por la salida. Este resultado, luego va a ser utilizado en las etapas posteriores. En caso que la instrucción sea un salto condicional, el sumador va a actuar sumando al PC con un offset que sale de la instrucción para que esa sea la próxima dirección de la memoria de instrucciones, es decir, para que sea el próximo valor del PC que es el registro que direcciona la misma. Los inputs y outputs de este módulo se pueden ver en la Figura 5:

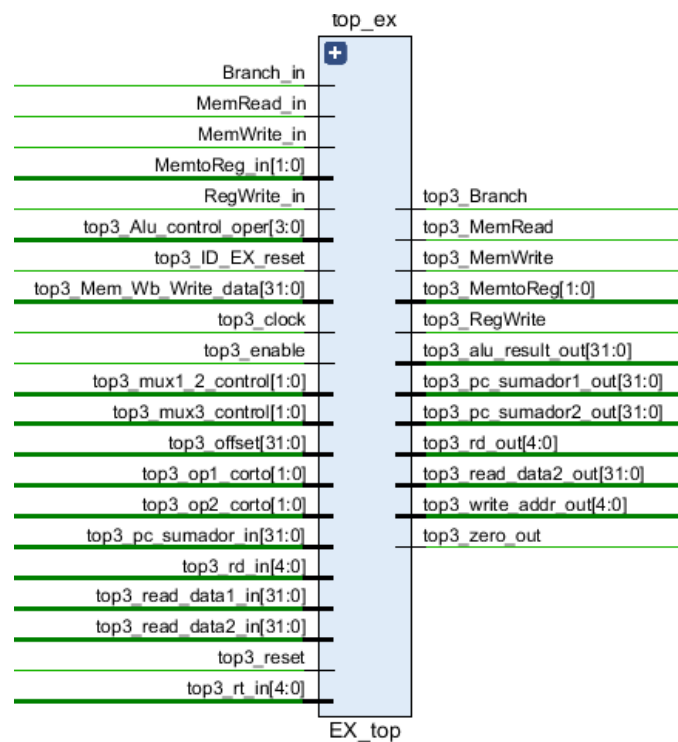


Figura 5: Inputs/Outputs de TOP_EX

Módulo TOP_MEM

Este módulo corresponde a la cuarta etapa del MIPS. En esta se accede a la memoria si la instrucción actual lo requiere. Puede ser para lectura o para escritura. Va a tener una memoria de datos y el módulo se va a encargar de la lectura o escritura dentro de esta.

Este módulo va a estar compuesto por un submódulo que va a ser la **Memoria de datos**, este va a ser el único módulo utilizado en la etapa para la lectura y almacenamiento de datos en memoria como alternativa a la memoria de registros. Los inputs y outputs del módulo se pueden observar en la Figura 6.

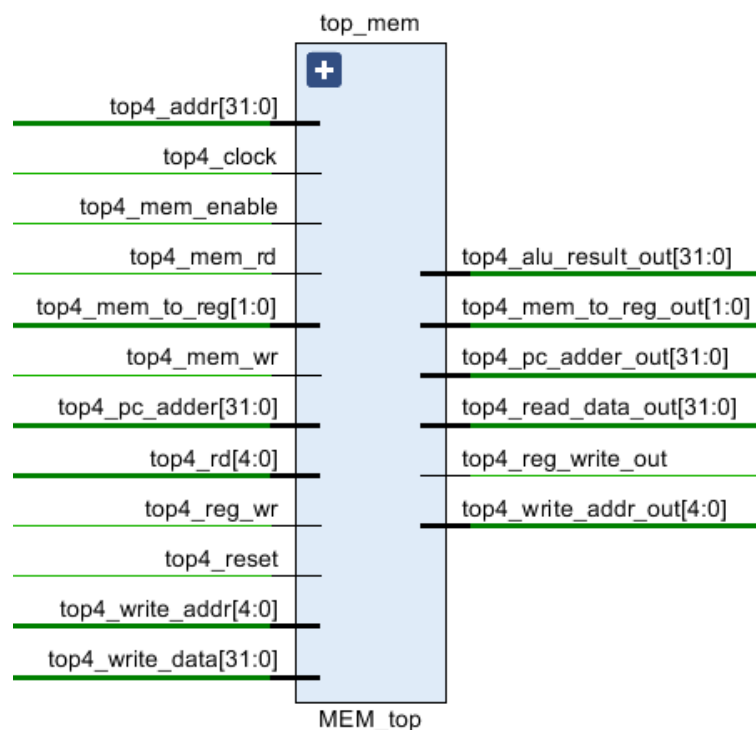


Figura 6: Inputs/Outputs de TOP_MEM

Módulo TOP_WB

Este módulo corresponde a la última etapa del pipeline. En esta etapa se actualizan los registros. Para las instrucciones que tienen un resultado (un registro de destino), Write Back escribe este resultado en el archivo de registro. Va a estar formado por un submódulo que va a ser un **multiplexor**. El módulo va a tener 4 entradas y una salida vistas en la Figura 7. Tres de esas entradas van a corresponder a si el dato es leído en memoria de datos (cuando es un store), al resultado de la ALU (luego de una instrucción aritmética) o a la segunda dirección siguiente de la instrucción JALR.

Luego, la salida va a depender de la 4 señal Mem_to_Reg que va a ser de control, según el valor de esa señal, el multiplexor va a seleccionar alguna de las 3 entradas antes mencionadas y se va a realizar el Write Back.

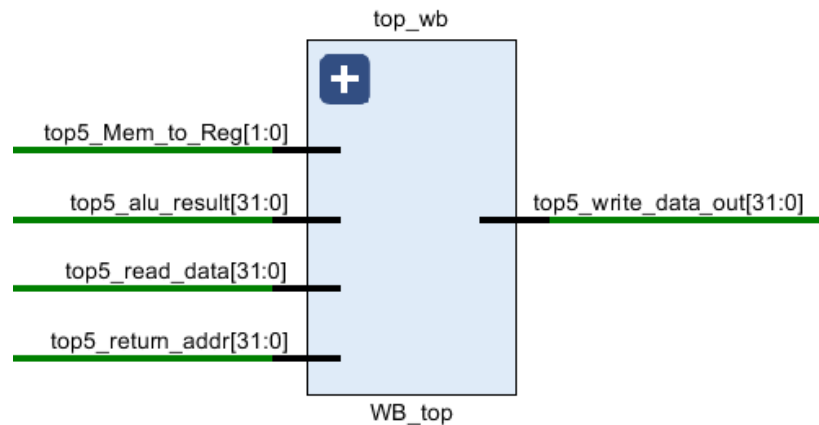


Figura 7: Inputs/Outputs de TOP_WB

Módulo Stall_Unit

Este módulo se implementó como una unidad para la detección de riesgos. Específicamente para un caso especial de dependencia de datos, la cual además de la necesidad de un cortocircuito mediante la unidad de cortocircuito se requiere introducir un retardo de un ciclo de reloj.

Este caso se da cuando una instrucción seguida de un load requiere la lectura del mismo registro que el load debe escribir en los registros. La necesidad del retardo se da debido a que, cuando la instrucción dependiente intenta acceder al registro, este valor todavía no se encuentra disponible, ni en la memoria de registros, ni en la salida de la memoria de datos (ya que la instrucción load todavía no lo ha leído).

Una vez que se introduce el retardo, el valor leído de la memoria de datos es multiplexado hacia el operando de la ALU para que la instrucción que está esperando ese dato pueda realizar la operación.

En la Figura 8 se observan los inputs y outputs del módulo

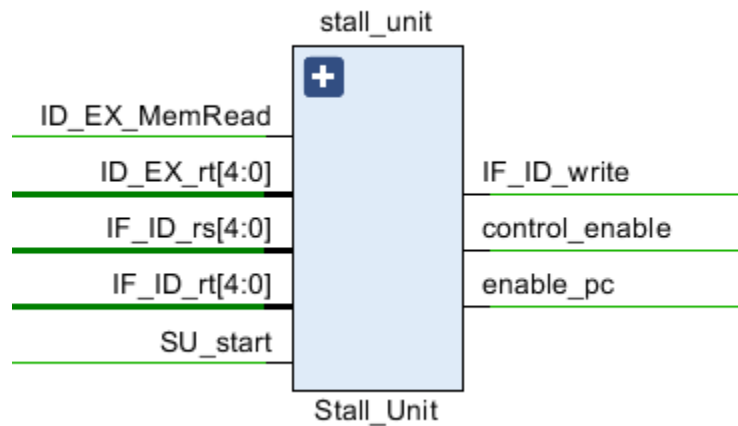


Figura 8: Inputs/Outputs de Stall_unit

Luego, en la Figura 9 se va a ver la condición de dependencia de los registros, cuando esas condiciones se cumplan, se va establecer la salida del enable del PC, del control y del cable que escritura entre IF e ID en 0 debido a que allí estaría ocurriendo un Stall.

```
always@(*) begin
    if(SU_start)
    begin
        enable_pc <= 1;
        control_enable <= 1;
        IF_ID_write <= 1;
    end

    else
    begin
        enable_pc <= 0;
        control_enable <= 0;
        IF_ID_write <= 0;
    end

    //Condicion de dependencia de los registros
    if(ID_EX_MemRead && ((ID_EX_rt == IF_ID_rs) || (ID_EX_rt == IF_ID_rt)))
    begin
        enable_pc <= 0;
        control_enable <= 0;
        IF_ID_write <= 0;
    end
end
```

Figura 9: Combinacional de la lógica de Stall Unit

Módulo Unidad Cortocircuito

El módulo de la unidad de cortocircuito se va a encargar de verificar los registros que se intersectan y si llega a encontrar alguno, va a tener multiplexores que se encargan de cortocircuitar esos datos. Esta unidad va a proveer al módulo de EX, las señales de control necesarias, calculadas uno o dos ciclos antes. Las entradas y salidas se ven en la Figura

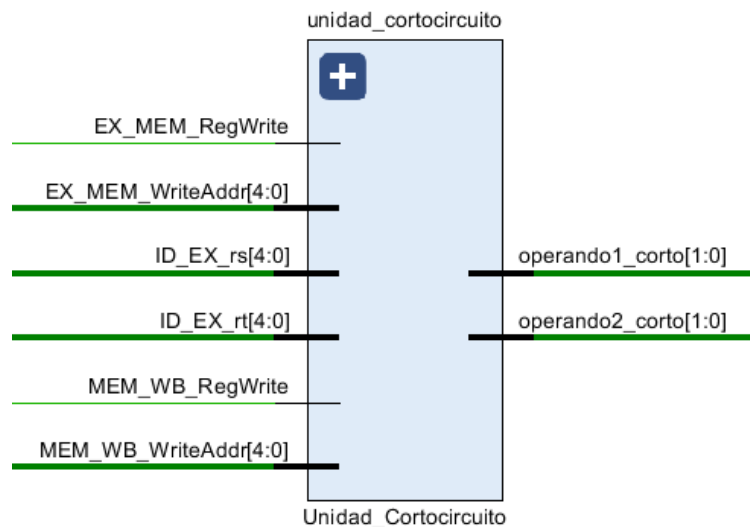


Figura 10: Inputs/Outputs de Unidad_cortocircuito

y dependiendo de las condiciones que se ven en la figura 11, es de donde serán multiplexadas.

```

//Condiciones del cortocircuito
always@(*) begin
    if((EX_MEM_RegWrite == 0) && (MEM_WB_RegWrite == 0))
    begin
        operandol_corto = 'b0;
        operando2_corto = 'b0;
    end

    if (EX_MEM_WriteAddr == ID_EX_rs && EX_MEM_RegWrite)
    begin
        operandol_corto = 'b1;
    end
    else if (MEM_WB_WriteAddr == ID_EX_rs && MEM_WB_RegWrite)
    begin
        operandol_corto = 'b10;
    end
    else
    begin
        operandol_corto = 'b0;
    end

    if (EX_MEM_WriteAddr == ID_EX_rt && EX_MEM_RegWrite)
    begin
        operando2_corto = 'b1;
    end
    else if (MEM_WB_WriteAddr == ID_EX_rt && MEM_WB_RegWrite)
    begin
        operando2_corto = 'b10;
    end
    else
    begin
        operando2_corto = 'b0;
    end
end
end

```

Figura 11: Condiciones de cortocircuito

Módulo UART

Este módulo va a actuar como interfaz de comunicación hacia el MIPS. Se van a cargar las instrucciones del programa y a su vez se va a utilizar la interfaz para enviar al procesador el modo en que se debe ejecutar, ya sea continuo o paso a paso.

Va a estar compuesto por 4 submódulos:

1. **Baud Rate Generator:** Encargado de generar la señal necesaria para que la tasa de baudios de la comunicación sea la correcta.

2. **Tx:** Máquina de estados para la transmisión de datos.
3. **Rx:** Máquina de estados para la recepción de datos.
4. **Unidad de debug:** Máquina de estados que envía información hacia y desde el procesador mediante UART y se encarga de gestionar el cargado de instrucciones y el control del modo de operación del programa que se ejecuta. Estos modos son:
 - a. **Modo Continuo:** se envía un comando a la FPGA por la UART y esta inicia la ejecución del programa hasta llegar al final del mismo (Instrucción HALT). Llegado ese punto se muestran todos los valores indicados en pantalla.
 - b. **Modo Paso a paso:** Enviando un comando por la UART se ejecuta un ciclo de Clock. Se debe mostrar a cada paso los valores indicados.

En la Figura 12 se observa las inputs y outputs de este módulo:

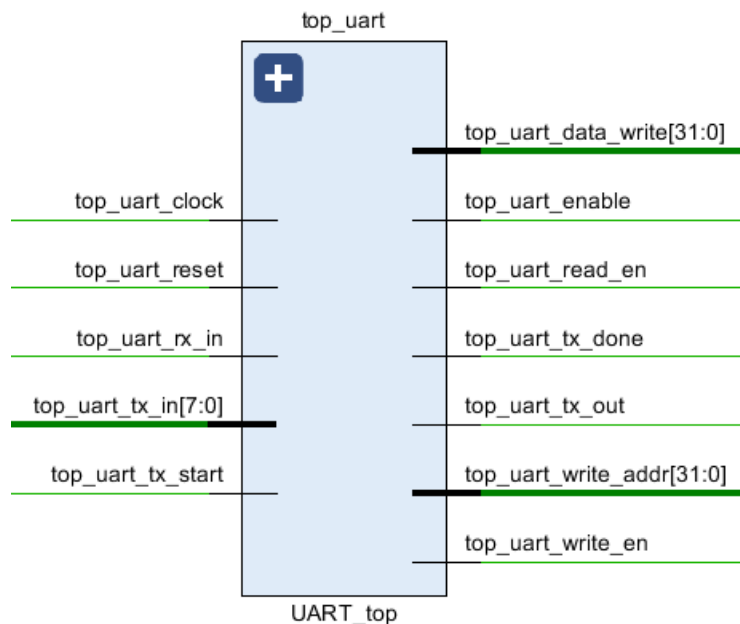


Figura 12: Inputs/Outputs de Módulo UART

Módulo TOP

Este módulo encapsula todas las unidades de las etapas de pipeline anteriormente descritas y las conexiones entre las mismas.

El mismo posee las siguientes entradas y salidas:

- Como entrada la señal para el reloj, un reset para los módulos secuenciales del procesador, una señal de start para controlar cuándo debe correr el procesador, y otras señales para el cargado en memoria de instrucciones

Cuando el PC es igual a 1 se pone en el programa la primera instrucción que corresponde a un ADDI que se suma el contenido del registro 2 con un entero que se pasa en la instrucción, en este caso un 5. Así, a medida que pasan los ciclos de reloj, el PC va aumentando en 1 debido que todavía no hay ninguna instrucción de salto y va direccionando cada instrucción que sigue de la memoria del programa. Hasta llegar a lo que se observa en la Figura 15 cuando entra en juego la instrucción de salto que es la del código 10850003 donde el PC salta a otro valor que no es el siguiente.



Figura 15: Modo continuo II - Vista de PC en salto

Por último, en la Figura 16 se muestra como se van modificando los registros del banco de registros. El primero en modificarse es el registro 11 que se carga con un 5 ya que es el resultado de sumar el contenido del registro 2 con un 5 entero. Luego el que le sigue es el registro 0 que se carga con un 1 correspondiente a la segunda instrucción de la memoria.

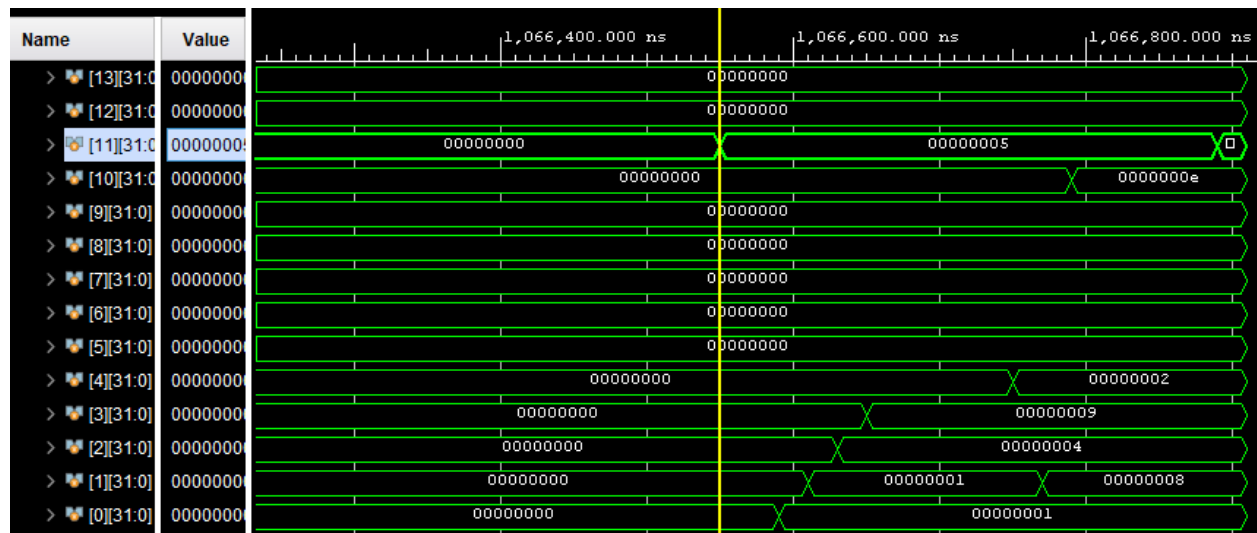


Figura 16: Modo continuo III - Actualización de registros

Page 10 of 10

En este modo se envían comandos por UART para que el procesador avance de instrucción en instrucción.

Esto se realiza haciendo uso de señales habilitadoras para las distintas etapas del pipeline, de forma tal que si las mismas están ausentes el procesador esté en un estado de “espera”. De esta manera cada vez que se envíe este comando al procesador, el mismo ejecutara un ciclo.

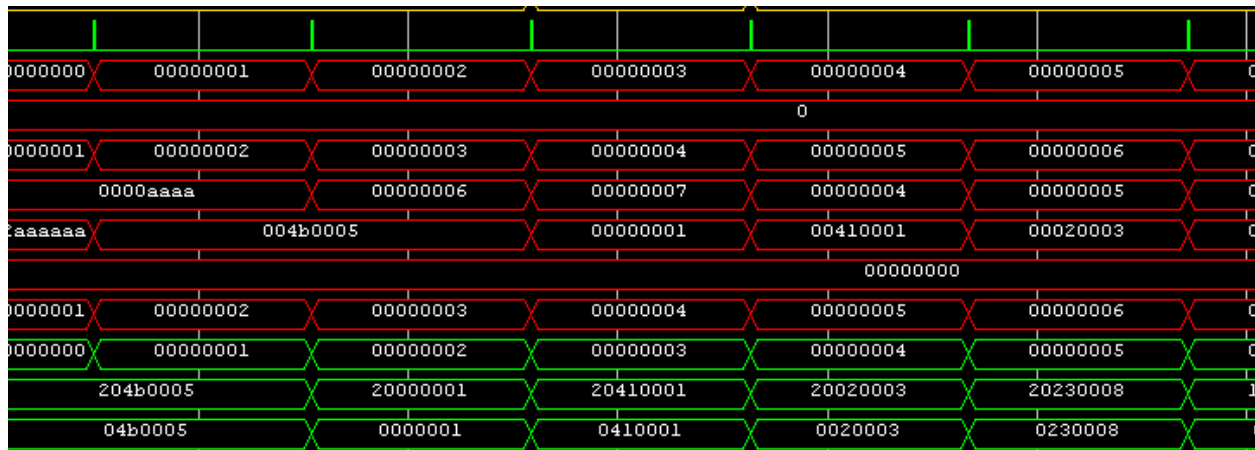


Figura 17: Modo paso a paso - Señal enable

En la imagen anterior puede observarse cómo al llegar la señal de habilitación, el PC aumenta y se van ejecutando las instrucciones, siendo la primera señal roja el PC y los pulsos verdes de arriba la señal de habilitación.

Análisis de Timing

Una vez realizada la implementación del trabajo, se procede a realizar un análisis temporal. Para esto se incluye al módulo de clock wizard. Este módulo permite al circuito contar con una señal de reloj más estable. Para incluirlo se siguieron los siguientes pasos:

1. Acceder a panel IP Catalog
2. Buscar Clocking Wizard
3. Configurar los parámetros deseados
4. Instanciar el módulo creado en el top

En primer lugar, se corre una simulación del programa y se van a seleccionar dos puntos importantes, el punto inicial va a ser cuando el PC = 0 y el final cuando el PC llega a su valor máximo al recorrer todas las instrucciones cargadas en memoria y llegar a la instrucción de halt.

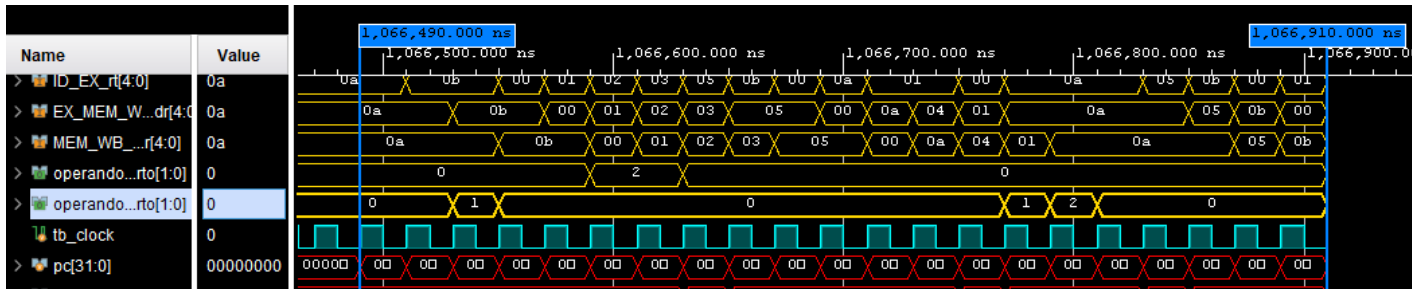


Figura 18: Análisis temporal

Por lo tanto, el tiempo que se demora en realizar estas instrucciones es de 420 ns ($1.066.910 \text{ ns} - 1.066.490 \text{ ns} = 420 \text{ ns}$). Si se ve en la figura 19, el tiempo de duración de cada ciclo de reloj es de 20 ns. Por lo tanto, se tardan 21 ciclos de reloj en correr todas las instrucciones cargadas en la memoria.



Figura 19: Análisis temporal: tiempo de clock

Esta prueba fue realizada con un clock con un período de 20 ns, es decir que en la simulación, este diseño está corriendo a una frecuencia de 50 MHz y tardaría un total de 420 ns en correr todo el diseño.

Ahora bien, se quiere analizar cuál es la frecuencia máxima a la que pueda operar el MIPS implementado para ver si se puede llegar a los 50 MHz que fue lo simulado. Para esto se va a tomar como referencia el concepto de Worst Negative Slack, que es la diferencia entre el delay del path combinacional más largo con el periodo de clock. Si se tiene un valor negativo, significa que el delay del path combinacional más largo es más grande que el periodo de clock lo que haría que el clock no alcance a cubrir ese path combinacional.

Para comprobar el Worst Negative Slack a una frecuencia de 50 MHz o un período de 20 ns, se corrió la implementación y síntesis del diseño con un clock wizard de 50 MHz y los resultados

fueron los observados en la figura 20. En este caso, el Worst Negative Slack es negativo por lo que se llega la conclusión de que con hay casi 72 ns del path combinacional más largo que no alcanzan a cubrir con un período de 20 ns, por lo tanto, a priori para conocer la frecuencia máxima a la que puede operar el MIPS debemos hacer más grande el período para que el slack sea positivo y el período de clock alcance a cubrir al path combinacional más largo.

Design Timing Summary

Setup

Worst Negative Slack (WNS): -71,669 ns

Total Negative Slack (TNS): -927,603 ns

Number of Failing Endpoints: 27

Total Number of Endpoints: 3827

Hold

Worst Hold Slack (WHS): 0,056 ns

Total Hold Slack (THS): 0,000 ns

Number of Failing Endpoints: 0

Total Number of Endpoints: 3827


Timing constraints are not met.

Figura 20: Worst Negative Slack

Por lo tanto, se corrió el diseño con frecuencias más chicas, es decir, períodos más grandes de clock y se registraron los resultados en la siguiente tabla:

Frecuencia de output de clock [MHz]	Periodo de clock [ns]	Worst Negative Slack [ns]	Tiempo de ejecución [ns]
50	20	-71.669	420
15	66.667	-25.125	1400
13	76.923	-14.871	1615
11	90.909	-0.895	1909
10	100	8.130	2100
10.7	93.458	1.589	1963
10.8	92.592	0.775	1944

Observando los resultados, vemos que la frecuencia máxima a la que se puede correr el diseño varía entre 10,7 y 10,8 MHz debido a que si se corre a 11 MHz, el slack es negativo por lo que el período de clock no alcanza a cubrir el path combinacional más largo. Es importante destacar que mientras menos frecuencia de clock se tenga, se va a tardar más tiempo en correr las instrucciones, por este motivo, es importante este análisis, ya que se tiene que correr el diseño a



la máxima frecuencia posible para lograr el mejor tiempo posible de ejecución siempre respetando el tiempo del path combinacional más largo.

Conclusiones

A modo de conclusión, consideramos que con la realización de este trabajo final pudimos terminar de afianzar todos los conocimientos adquiridos a lo largo de la materia. Nos sirvió mucho al principio el consejo de hacer un diagrama de bloques con las entradas y salidas de cada módulo para luego en la implementación poder ir guiándonos. Tuvimos dificultades con la herramienta de vivado porque muchas veces se nos cerraba el software sin razón o no simulaba correctamente pero son cosas que fuimos aprendiendo y modificando.

También con la realización de este trabajo pudimos afianzar conceptos que vimos en la parte teórica de la materia. Creemos que llevar a la práctica algo teórico siempre es buena manera de aprender.

Referencias

- MIPS IV Instruction Set. Revision 3.2. By Charles Price (September, 1995)
- [Diagrama de bloques del procesador](#)
- [Repositorio MIPS](#)