Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Ingeniería en Computación

Arquitectura de Computadoras

Procesador MIPS

Alumnos:

- Tolay, Antonio Facundo - DNI 37634834
- Vaira, Franco Gabriel - DNI 38730172

Docentes:

- Ing. Martin Pereyra
- Ing. Santiago Rodriguez

Introducción

En este informe se presenta el proceso de desarrollo e implementación del trabajo integrador de la materia de Arquitectura de Computadoras denominado *Pipeline en un procesador MIPS simplificado*.

El trabajo cumple con los requerimientos definidos en la consigna:

- Implementar el procesador MIPS segmentado con las etapas *IF, ID, EX, MEM y WB*.
- Implementar las instrucciones de tipo *R, J* e *I*.
- Debe soportar el manejo de los riesgos estructurales, de datos y de control.

Desarrollo

La arquitectura de la segmentación (pipeline) permite dividir el procesamiento de una instrucción en múltiples etapas, de forma que cada etapa se puede enfocar en realizar un trabajo específico para distintas instrucciones. De esta manera cuando una etapa termina de procesar una instrucción, inmediatamente, comienza a trabajar sobre la siguiente y así maximizar los ciclos de reloj del sistema.

Las etapas de la segmentación se conectan por medio de registros, lo suficientemente grandes para almacenar todos los datos que corresponden con las líneas que pasan a través de ellos, que almacenan y transfieren datos de una etapa a otra y sincronizados por el clock del procesador.

Pipeline

Como ya se mencionó anteriormente, la segmentación contiene cinco etapas:

1. Instruction Fetch (IF): buscar una instrucción en memoria
2. Instruction Decode (ID): leer los registros mientras se decodifica la instrucción
3. Execute (EX): ejecutar una operación o calcular una dirección de memoria
4. Memory Access (MEM): acceder a un operando en la memoria de datos
5. Write Back (WB): escribir el resultado en un registro

Cada una de estas etapas cuenta con su módulo propio dentro del proyecto para tener una mayor claridad sobre la tarea que realiza cada una de estas etapas y facilitar el desarrollo, el seguimiento y el análisis de las mismas.

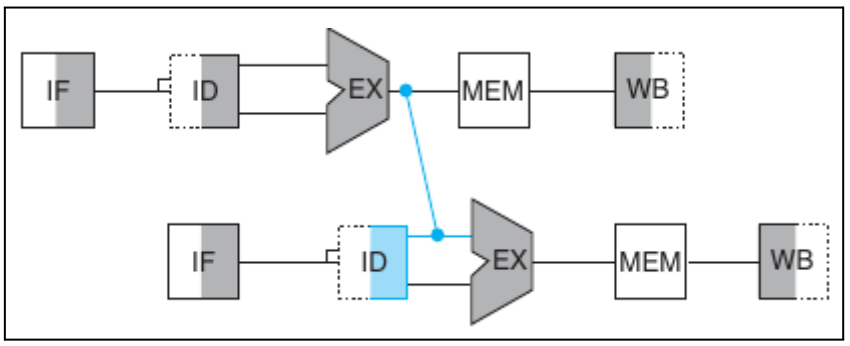
Riesgos

La implementación de la segmentación trae consigo riesgos que se pueden presentar durante la ejecución de las instrucciones, estos son:

- Riesgo estructural (structural hazard): ocurre cuando el hardware no admite una cierta combinación de instrucciones durante el mismo ciclo de clock.
- Riesgo de datos (data hazard): sucede cuando el pipeline se debe bloquear debido a que un paso de ejecución debe esperar a que otro sea completado.
- Riesgo de control (control hazard): se presenta cuando es necesario tomar una decisión basada en los resultados de una instrucción mientras las demás se están ejecutando.

Estos riesgos mencionados fueron mitigados mediante la implementación de las unidades para cada tipo tipo de riesgo:

- Unidad de riesgo (hazard unit): detecta si se está dando una instrucción de tipo Load y como consecuencia generar un bloqueo, es decir, la introducción de una operación vacía entre las operaciones, para luego resolver algunas dependencias de datos.
- Unidad de cortocircuito (forwarding unit): busca establecer un cortocircuito entre los conectores latcheados de las etapas con las entradas de la ALU para disponer de los datos antes que sean escritos en memoria.



- Unidad de control (control unit): se encarga de generar señales de control que guían el comportamiento del procesador a lo largo de cada etapa de la segmentación.

Señal	Acción al estar activa	Acción al estar desactivada
RegDst	El identificador del registro destino para la escritura viene del campo *rd*	El identificador del registro destino para la escritura a registro viene del campo *rt*
RegWrite	El registro se escribe con el valor de escritura	Ninguno
ALUSrc	El segundo operando de la ALU son los 16 bits de menor peso de la instrucción con el signo extendido	El segundo operando de la ALU proviene del segundo registro leído del banco de registro
PCSrc	El PC es reemplazado por la salida del sumador que calcula la dirección destino del salto	El PC es reemplazado por su valor anterior más cuatro (PC+4)
MemRead	El valor de la posición de memoria designada por la dirección se coloca en la salida de lectura	Ninguno
MemWrite	El valor de la posición de memoria designada por la dirección se reemplazas por el valor de la entrada de datos	Ninguno
MemToReg	El valor de entrada del banco de registros proviene de la memoria	El valor de entrada del banco de registros proviene de la ALU

Síntesis de hardware

Utilización

Una vez finalizado el diseño se pueden observar las tablas de utilización de las celdas de la FPGA. En la imagen debajo se detalla la cantidad de celdas en uso por cada bloque principal del top level.

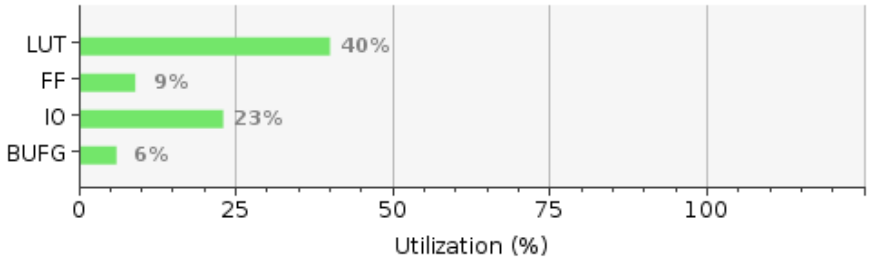
Hierarchy

Name	Slice LUTs ...	Slice Registers ...	F7 Muxes...	F8 Muxes...	Slice (81...	LUT as Logic ...	LUT Flip Flop Pairs...	Bonded IOB (106)	BUFG CTRL...
top	8324	3813	1194	157	2742	8324	9295	24	2
u_clock_divider ...	1	1	0	0	1	1	1	0	0
u_debug_unit (d...	70	86	1	0	43	70	114	0	0
u_mips (mips)	8163	3628	1193	157	2687	8163	9117	0	0
u_uart (uart_32b)	90	98	0	0	39	90	112	0	0

Por otra parte, en la imagen debajo podemos observar un resumen del tipo de celdas que se sintetizaron, como ser Flip-Flops, LUTs y puertos IO. Es muy importante notar que no se han utilizado latches en el diseño, los mismos no son recomendables.

Summary

Resource	Utilization	Available	Utilization %
LUT	8324	20800	40.02
FF	3813	41600	9.17
IO	24	106	22.64
BUFG	2	32	6.25



Timing

La herramienta provee un análisis de timing para diferentes casos al definir el clock con la directiva ***create_clock*** en el archivo de constraints. Se puede ver que el diseño cumple con las constraints de timing y el diseño es apto para funcionar sobre hardware.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8,429 ns	Worst Hold Slack (WHS):	0,460 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1	Total Number of Endpoints:	1	Total Number of Endpoints:	2

All user specified timing constraints are met.

Instrucciones soportadas

Se detalla una tabla con todas las instrucciones soportadas.

Tipo		Instrucciones
Tipo R	**Aritméticas y lógicas**	ADDU - SUBU - AND - OR - XOR - NOR - SLT
	Desplazamiento	SLL - SRL - SRA - SLLV - SRLV - SRAV
Tipo I	**Aritméticas y lógicas con valor inmediato**	ADDI - ANDI - ORI - XORI - SLTI
	R/W en memoria	LB - LH - LW - LBU - LHU - LWU - SB - SH - SW - LUI
	Saltos	BEQ - BNE - J - JAL
Tipo J		JALR - JR

Verificación y testing

En primer lugar, la verificación a nivel de bloques individuales fue llevada a cabo mediante testbenches y simulaciones hechas dentro del entorno de Vivado.

Por otro lado, para verificar la funcionalidad a nivel de sistema se implementaron una serie de tests que prueben todas las instrucciones y simulan algunas situaciones de riesgos de datos y de control. Los test implementados destinados a probar el procesador se basan en programas escritos en Assembler que, al finalizar, dejan a los registros y la memoria de datos en un estado determinado y conocido.

Además de ello, se implementó un mecanismo de regresiones de modo que se ejecutan todos los test consecutivamente, de modo que si alguno de todos falla, la regresión reporta una falla al no coincidir la salida del programa con el resultado esperado.

En la siguiente tabla podemos ver la relación entre cada test y que instrucciones pone a prueba.

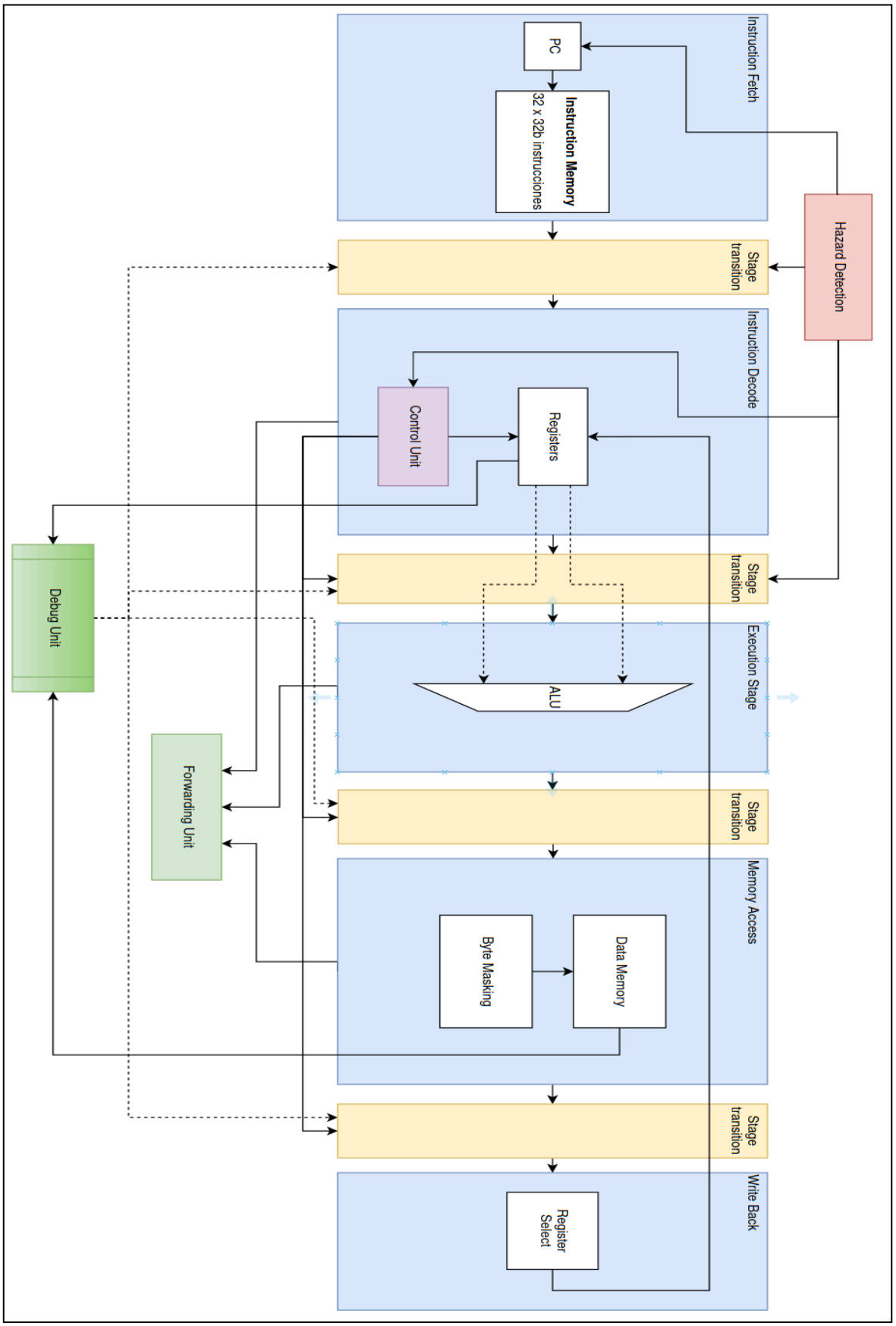
Testcase	Instrucciones que prueba
aaa.asm	ADDI - ADDU - SUBU - OR - NOR - AND - XOR - ANDI - ORI - XORI
bbb.asm	ADDI - SB - SH - SW - LB - LH - LW - ADDU - SUBU
ccc.asm	ADDI - SLL - SRL - SRA - SLLV - SRLV - SRAV - NOP - ADDU - SB - LB - BEQ
ddd.asm	ADDI - LUI - SB - SH - SW - SLL - SRL - SRA - LBU - LHU - LWU - AND - OR - XOR
eee.asm	ADDI - SRL - BNE - BEQ - J
fff.asm	ADDI - SLT - BEQ - J - ADDU - JAL - JR
ggg.asm	ADDI - SLT - BEQ - J - ADDU - JAL - JALR - J - JR
hhh.asm	ADDI - SLT - BEQ - ADDU - SLTI - J

Unidad de Debug

La unidad de Debug es uno de los bloques principales del diseño dado que se encarga de la comunicación bidireccional entre la PC con puerto serie y el procesador. Las tareas principales de la DU son grabar el programa, obtener el valor de los 32 registros y de la memoria de datos y ejecutar paso a paso un programa.

Dado que se trata de un procesador de 32 bits, la mayoría de las transacciones son a ese nivel, por lo que fue necesario realizar algunas modificaciones al módulo UART para que pueda transmitir palabras enteras de 32 bits, es decir 4 bytes.

Arquitectura final



Conclusión

Al desarrollar el presente proyecto hemos adquirido vastos conocimientos sobre diseño digital y sobre las mejores prácticas para lograr que el diseño sea funcional al ser sintetizado y sin latches.

Se alcanzaron los objetivos de llevar a cabo un procesador de arquitectura MIPS con un pipeline de 5 etapas que interpreta y ejecuta instrucciones contemplando riesgos de datos y de control.

Además, se le dio una solución a la comunicación bidireccional por parte del procesador, esto mediante la implementación de una Unidad de Debug que envía y recibe información y comandos, respectivamente.

Al mismo tiempo se ha tenido un fuerte enfoque en la verificación y la integración de nuevos tests a medida que avanzó el diseño.

Finalmente podemos concluir que el proyecto fue realizado con éxito y que el proyecto puede ser extendido para convertirse en un microcontrolador con periféricos.

Referencias

- MIPS IV Instruction Set
- Computer Organization and Design 3rd Edition. Chapter 6. Hennessy-Patterson
- Computer Architecture: A Quantitative Approach. Hennessy- Patterson