FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES
UNIVERSIDAD NACIONAL DE CÓRDOBA


# Arquitectura de Computadoras

## Trabajo Práctico Nº4 - MIPS

Aagaard Martin - Navarro Matias

Universidad Nacional de Córdoba
27 de noviembre de 2019

# Índice

# 1. Introducción

Para el Trabajo Final de la materia, se realizó la implementación en Verilog del pipeline de cinco etapas del procesador **MIPS**. El mismo se trata de un procesador de 32 bits, con 32 registros, una memoria de instrucciones de 2048 posiciones y una memoria de datos de 1024 posiciones. El mismo fue desarrollado mediante lenguaje de especificación de hardware Verilog y haciendo uso del entorno de desarrollo Vivado.

# 2. Objetivos

Las etapas se definieron según muestra la figura 1:

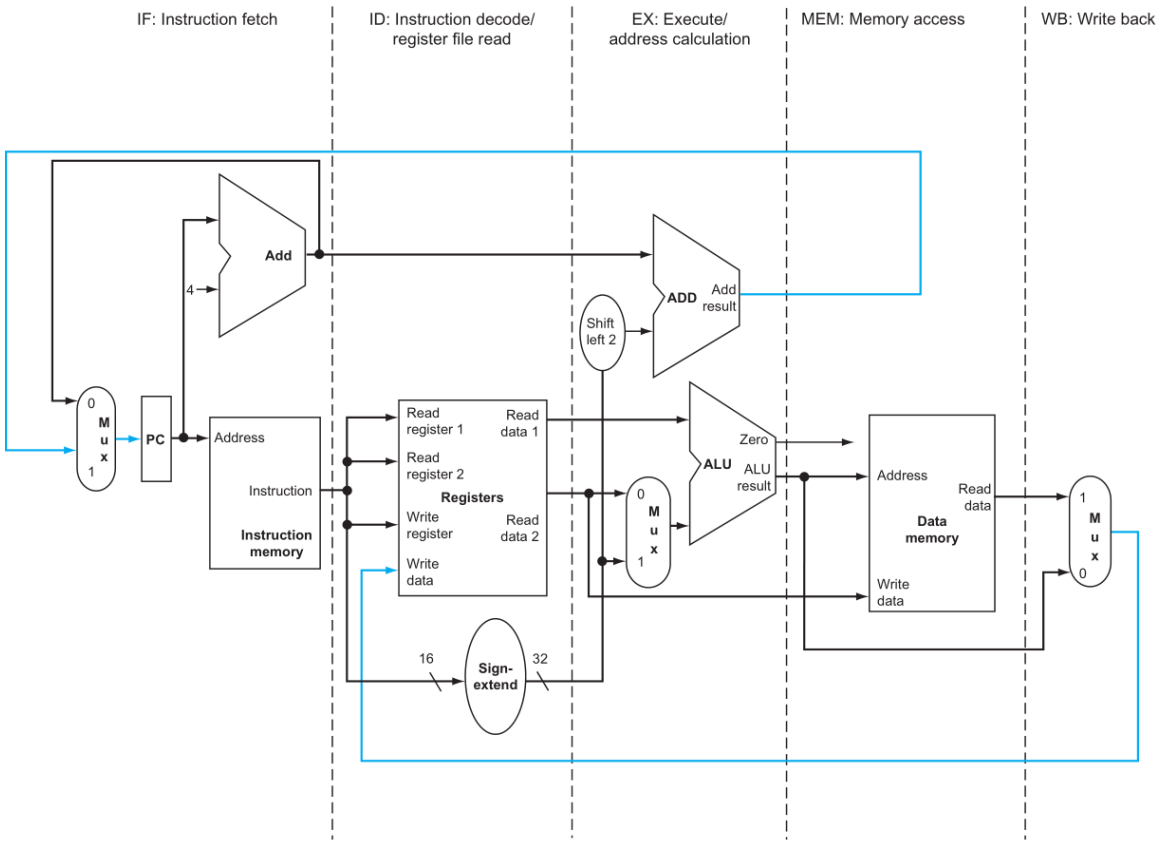


Figura 1: División del MIPS.

- **IF (Instruction Fetch)**: búsqueda de la instrucción en la memoria de programa.
- **ID (Instruction Decode)**: decodificación de la instrucción y lectura/escritura de registros.
- **EX (Execute)**: ejecución de la instrucción propiamente dicha.
- **MEM (Memory Access)**: lectura o escritura desde/hacia la memoria de datos.
- **WB (Write back)**: escritura de resultados en los registros.

Las instrucciones a implementar del set de instrucciones del MIPS son las siguientes:

- **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.

Su funcionamiento y descripción puede encontrarse en el manual de instrucciones siguiente: `http://math-atlas.sourceforge.net/devel/assembly/mips-iv.pdf`

El procesador MIPS debe poseer soporte para los siguientes tipos de riesgos:

- **Estructurales**: se genera cuando dos instrucciones, en el mismo ciclo, intentan utilizar el mismo recurso.
- **De datos**: se genera cuando se intenta utilizar un dato antes de que este preparado, se debe mantener el orden estricto de lecturas y escrituras.

- **De control**: se genera cuando intenta tomar una decisión sobre una condición no evaluada todavía.

Dos soluciones para los riesgos de datos:

- Unidad de cortocircuitos.
- Unidad de detección de riesgos.

Otros requerimientos:

- Se debe implementar un programa ensamblador.
- El programa a ejecutar debe ser cargado en la memoria de programa mediante un archivo ensamblado.
- Se debe enviar a la PC:
  - Contenido de los registros usados.
  - Contenido de los latches intermedios.
  - PC (Program Counter).
  - Contenido de la memoria usada.
- Se debe determinar la frecuencia máxima en la que el MIPS puede ser ejecutado.
- Se debe implementar un modo "paso a paso"que avance el pipeline utilizando una señal de entrada.

# 3. Desarrollo

El procesador fue desarrollado basándose en el diseño presentado en el libro *"Computer Organization and Design"* de David Patterson y John Hennessy en el capítulo 4 subíndice 5 "An Overview of Pipelining"; el cual

constituyó una excelente guía para comprender el funcionamiento de un procesador segmentado, las ventajas e inconvenientes, y los detalles de diseño a tener en cuenta.

## 3.1. MIPS

El procesador MIPS es un procesador RISC (reduced instruction set computer), la particularidad de estos es que su set de instrucciones es reducido, todas sus instrucciones tienen el mismo largo y solo las instrucciones de carga y almacenamiento acceden a la memoria de datos. El procesador consta de cinco etapas, esto significa que durante un ciclo de reloj se estarán ejecutando hasta 5 instrucciones. Para ello, se necesita colocar registros entre las etapas de segmentación. Al conjunto de registros que se encuentran entre una etapa y otra se los denomina latch o registro de segmentación. Cada uno de los latch lleva el nombre de las dos etapas que separan. Durante cada ciclo de reloj, todas las instrucciones avanzan de un latch al siguiente.

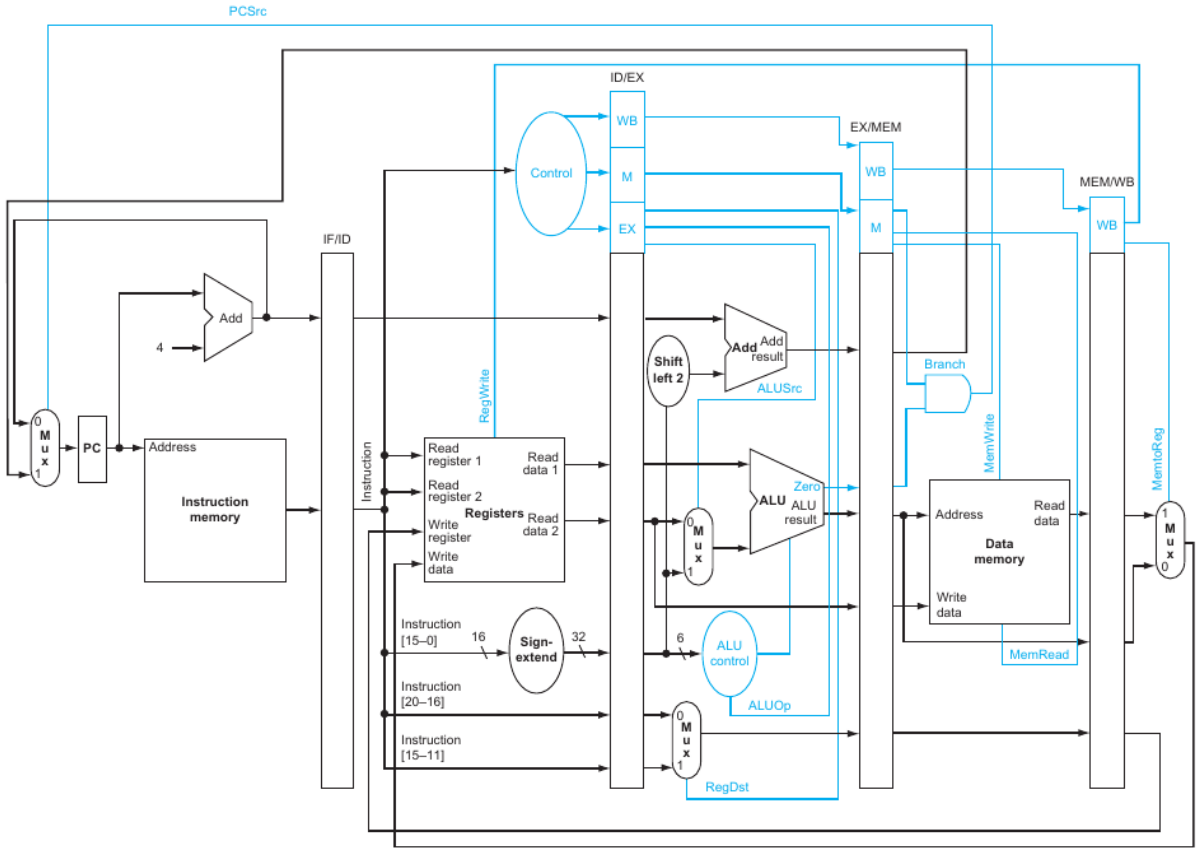


Figura 2: MIPS segmentado.

### 3.1.1. Instruction Fetch (IF)

Enviar el PC a memoria y buscar la instrucción a ejecutar. Actualizar el PC sumando 4 (cada instrucción es de 4 bytes).

En este módulo, la memoria de programa trabaja con el flanco ascendente del clock. Por otra parte, el contador de programa y los registros de salida del top utilizan el otro flanco de clock(descendente). A su vez, es importante mencionar que para la memoria de programa sintetizada como Block RAM por la herramienta Vivado, se utilizó el template Xilinx Single Port Byte-Write Read First RAM.

#### Instruction Memory

Memoria dónde son cargadas las instrucciones del programa. En cada ciclo, con el PC busca la instrucción que va a ejecutar. Antes de que comienza la ejecución del MIPS existe una etapa de carga, donde desde fuera del MIPS se envían instrucciones a ser escritas a posiciones de memoria.

### 3.1.2. Instruction Decode (ID)

Decodificar la instrucción y acceder a los registros implicados. Analizar valores de los registros por posibles saltos (branches). Extender el signo del valor de offset si es necesario. Calcular posible dirección de salto considerando el offset que se debería incrementar al PC.

Módulo combinacional que decodifica la instrucción que proviene de la etapa Instruction Fetch. De este módulo salen las direcciones de registros rs, rt, rd, reg A y reg B. A su vez, el valor inmediato de la instrucción como también detectar los saltos incondicionales y su respectiva dirección.

#### Control

Es un módulo que genera las señales de control para las etapas EX, MEM y WB para la instrucción correspondiente.

#### Registers

Es el banco de 32 registros de 32 bits cada uno. En el flanco ascendente del clock, se escriben con los valores y las señales de control que provienen de la etapa Write Back. Por otra parte, en el flanco descendente se colocan en la salida los valores que contienen los registros apuntados por las direcciones reg A y reg B.

#### Unidad de detección de riesgos (Hazard Detection Unit)

Consiste en la detección de riesgos y parada del procesador en un ciclo. Esto involucra incorporar una unidad de detección de riesgos que actúa en ID impidiendo la carga y lectura de una nueva instrucción.

### 3.1.3. Execute (EX)

La ALU usa los operandos, realizando una de 3 funciones según el tipo de instrucción.

- **Memory reference**: suma el registro base y el offset para formar la dirección efectiva.
- **Register-Register**: opera según el opcode entre dos registros.
- **Register-Immediate**: opera según el opcode entre un registro y un valor inmediato.

#### ALU Control

Genera los bits de control para diferentes tipos de operaciones de la ALU.

#### ALU

Es un módulo combinacional que efectúa la operación solicitada entre los dos operandos de sus entradas en función de la señal de control ALU Control.

### 3.1.4. Memory Access (MEM)

Con la dirección efectiva, se realiza el acceso a memoria, tanto sea un "load" o un "store".

#### Data Memory

Memoria donde son almacenados los datos del programa.

### 3.1.5. Write Back (WB)

Para instrucciones R o Load, se escribe el resultado en el registro correspondiente.

### 3.1.6. Unidad de Cortocircuito (Forwarding Unit)

Toma operandos fuentes (**rs** y **rt**) de la instrucción que se está por ejecutar y los destino (**rd**) de las instrucciones anteriores y los compara. Si son iguales, adelanta el resultado para no esperar WB. Se hace cortocircuito si las anteriores quieren escribir el registro que estoy por usar.

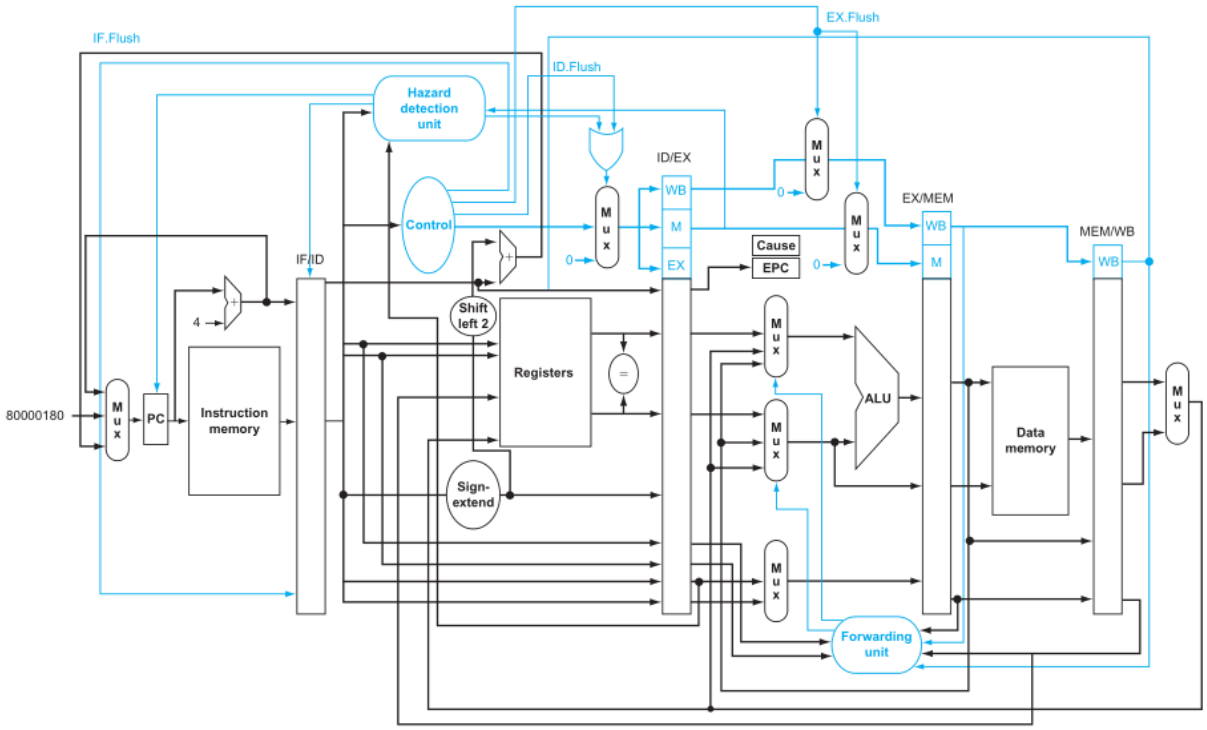


Figura 3: MIPS completo con control de riesgos.

# 4. Ensamblador

Se desarrolló en Python un script que permite la generación de código binario a partir del Set de instrucciones MIPS. El programa a ejecutar es escrito en un archivo de texto que es utilizado para inicializar la memoria de instrucciones.

**Código assembler**:

```
1  ADDI   1,2,10
2  ADDI   2,2,5
3  SW     1,0(3)
4  SW     2,4(3)
5  LWU    3,0(3)
6  LWU    4,4(4)
7  ADDU   5,3,4
8  SW     5,8(7)
9  HALT
```

Figura 4: Código Assembler a convertir.

**Salida del ensamblador**:

```
1  00100000010000010000000000001010
2  00100000010000100000000000000101
3  10101100011000010000000000000000
4  10101100011000100000000000000100
5  10011100011000110000000000000000
6  10011100100001000000000000000100
7  00000000011001000010100000100001
8  10101100111001010000000000001000
9  11111111111111111111111111111111
```

Figura 5: Código binario generado a partir del script.

# 5. Módulo Clock Wizard

La implementación del proyecto dio como resultado una frecuencia de clock máximo de 32 MHz. A esta frecuencia se cumplieron todas las restricciones de tiempo especificadas como se puede observar en la figura 6. Este módulo provee la frecuencia elegida a todos los demás bloques del proyecto.

**Design Timing Summary**

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 1.085 ns | Worst Hold Slack (WHS): | 0.049 ns | Worst Pulse Width Slack (WPWS): | 3.000 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: | 16052 | Total Number of Endpoints: | 16052 | Total Number of Endpoints: | 2258 |

**All user specified timing constraints are met.**

Figura 6: Resumen de reporte de tiempo a 32MHz.

# 6. Clock Enable

Se modificó el comportamiento del clock en el procesador MIPS. Este es habilitado mediante una señal de enable para poder realizar una ejecución paso a paso. Dicha señal de enable dura exactamente 1 ciclo de reloj del MIPS cada vez que es pulsado el botón correspondiente para la ejecución.

# 7. Test Bench

Para el diseño del procesador se confeccionó una hoja de donde se encuentra el estado de todas las señales de control frente a cada instrucción. El testing se realizó primeramente mediante test benches, que permitieron comprobar que la simulación realizara lo esperado. Este proceso se expandió etapa a etapa, hasta completar la simulación del procesador en su totalidad.

Se comprobó que cada una de las instrucciones funcionaran correctamente en diferentes casos planteados.

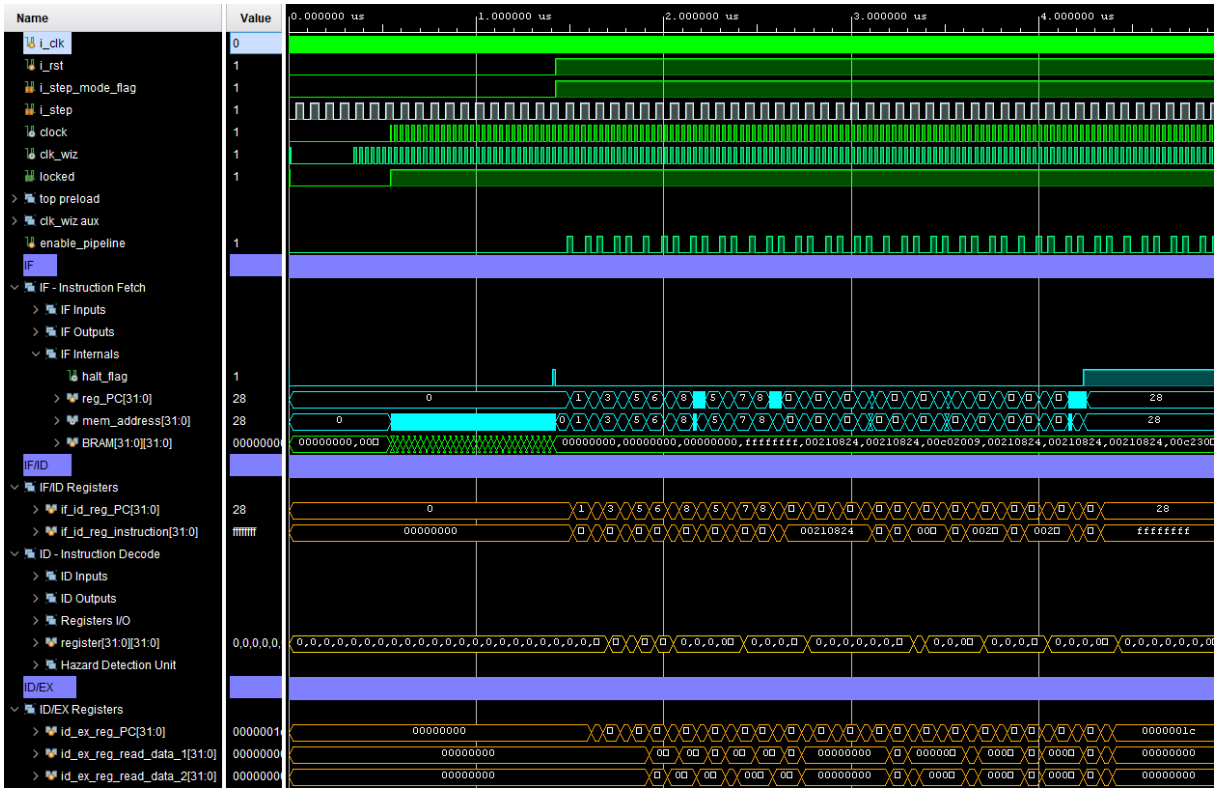


Figura 7: Wavefile de la ejecución del test 8.

# 8. Conclusión

Una vez terminado el práctico, a través de la herramienta Vivado se obtuvieron las especificaciones en la utilización de los recursos de la FPGA y en la potencia que consume dicho circuito instanciado.

Se logró implementar y probar todas las instrucciones, verificando que no haya conflictos entre todas las posibles acciones que se pueden tomar.

A demás, la ejecución del MIPS a una frecuencia máxima de 32 MHz. Para aumentar la velocidad de funcionamiento, se debería rediseñar la arquitectura del mismo, debido a los problemas de lógica en cada etapa; un ejemplo puntual de esto, se manifiesta en las escrituras a registros entre flancos consecutivos cuando existe demasiada lógica combinacional entre eventos.