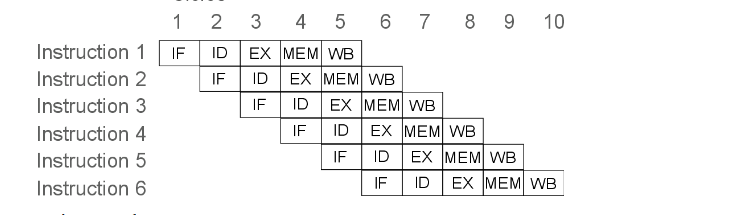
**TRABAJO PRÁCTICO FINAL ARQUITECTURA DE COMPUTADORAS**

**Objetivo**

Implementar el pipeline del proceso MIPS en nuestra placa adquirida Basys 3

**PIPELINE MIPS**

Implementación de etapas para ejecutar instrucciones, donde una instrucción empieza a ejecutarse antes de que hayan terminado las anteriores y, por lo tanto, que haya varias instrucciones procesando simultáneamente.

****

**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 propiamente dicha.
* **MEM (Memory Access):** Lectura o escritura desde/hacia la memoria de datos.
* **WB (Write back):** Escritura de resultados en los registros.

**Instrucciones permitidas:**

* Instrucciones **tipo R:**

**SLL, SRL, SRA, SLLV, SRLV, SRAV, ADDU, SUBU, AND, OR, XOR, NOR, SLT**

* Instrucciones **tipo I:**

**LB, LH, LW, LWU, LBU, LHU, SB, SH, SW, ADDI, ANDI, ORI, XORI, LUI, SLTI, BEQ, BNE, J, JAL**

* Instrucciones **tipo J:**

**JR, JALR**

**Riesgos:**

Nuestro procesador cuenta con la implementación para atender los siguientes tipos de riesgo:

* **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 disponible. Mantenimiento del orden estricto de lecturas y escrituras.
* **De control.** Se intenta tomar una decisión sobre una condición todavía no evaluada.

**Requerimientos de operación:**

El programa a ejecutar debe ser cargado en la memoria de programa mediante un archivo ensamblado. El procesador estará a la espera para recibir un programa mediante la **Debug Unit** (es decir mediante la interfaz de UART). 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.

**ETAPA IF**

La etapa IF contiene una instancia del módulo PC para manejar el contador de programa y calcular la siguiente address de instrucción. También permite almacenar instrucciones dentro de la memoria. Ya que las instrucciones son cargadas por la Unidad de de Debug.

Mediante un Multiplexor se obtiene el PC correspondiente y en el módulo de PC se incrementa el PC para la siguiente instrucción

**Módulos de etapa IF:**

* **IF.v:** Inicializa la etapa.
* **pc.v:** Se obtiene el PC, PC+4 y PC+8 en caso de JAL
* **mux\_pc.v:** Se setea un incremento en el PC si hubo un branch o un jump a partir de la instrucción que tiene.
* **memory\_instruc.v:** Recibe una address de la instrucción de que debe recuperar de los registros de la memoria de instrucciones. También escribe las instrucciones que son enviadas desde la pc por la **Debug Unit.**
* **IFID.v:** Es el latch que recibe la insgtruccion de la memoria de instrucciones y el los nuevos pc para la siguiente instrucción. Este latch envia estos datos a la etapa siguiente de decodificación.

**ETAPA ID**

Como se mencionó la instrucción se ejecuta dentro de un pipeline y luego de la primera etapa anterior continúa la instrucción en la etapa de decodificación.

En esta etapa se decodifica la instrucción separando sus diferentes campos. La decodificación sirve para identificar qué tipo de instrucción se trata, y hemos visto que pueden ser de distinto tipo y contener distintas partes que nos permitirán ejecutar esa instrucción.

## 

￼El código de operación de la instrucción indica qué tipo de instrucción es, y por tanto, qué tipo de operación se debe realizar. Si es necesario, se leen 1 o 2 operandos de los registros del banco de registro como también se calcula la extensión de signo.

**Módulos etapa ID:**

* **BankRegisters.v**: Los registros necesarios para almacenar valores y que se encuentran dentro de este banco de registro. Este banco recibe la dirección rs, rt y lee los registros correspondientes para presentar los datos correspondientes a la salida. También se encarga de recibir la dirección rd del registro donde se va a guardar el dato a escribir.
* **sign\_extensor.v:** extiende el signo de los operandos inmediatos, que son de 16 bits, a 32 bits. Esto va dar una salida que puede ser:
* Extender a 32 con los 16 más significativos respetando el signo.
* Extender a 32 con los 16 más significativos usando un cero.
* Colocar los 16 bits como más significativos y agregar ceros en los 16 menos significativos
* **PC\_Jump.v**: se utiliza para calcular nuevas address para el PC en los casos que se de saltos condicionales en una instrucción dentro del pipelines del MIPS. Suma la dirección de salto con PC+4. De esta manera se obtiene la dirección al cual teng que hacer el jump.
* **UnitRisk.v:** este módulo se encarga de detectar cuando hay riesgos y agregar burbujas cuando como los casos de una lectura después de una escritura para una instrucción de Load o para el caso de Branch.
* **UnitControl.v** Este módulo se encarga de recibir el Opcode de la instrucción y de esta manera generar las señales necesarias para viajar junto con la instrucción a través del resto del pipeline para activar o desactivar en base a la instrucción decodificada.
* **mux\_unit\_risk.v :** Solo hará un switch del valor de las señales de riesgo.Si hay una condición de riesgo se ponen a cero todas las señales de la Unit Control.
* **IDEX.v**: Módulo que recibe los datos de la etapa decode y los envía a la etapa execute, cuando hay un posedge del clock. De esta forma no pierdo datos entre las etapas, los retengo en este módulo. También envía las señales de control generadas por la Unit Control.

**Unidad de Debug**

Este módulo es uno de los más importantes ya que nos permite enviar datos desde la pc a nuestro mips. Desde la PC voy a poder enviar mediante un código las instrucciones que se cargan inicialmente en la memoria de instrucciones.

Es por eso que la comunicación se realiza mediante UART como la aprendida en el trabajo práctico anterior. Mediante este módulo y el módulo UART establecemos una comunicacion bidireccional que consta de diferentes etapas ya que se desarrolla siguiendo el concepto de máquinas de estados.

**IDLE:** Es el estado por default e inicial. Aquí la Unidad de Debug queda a la espera de recibir un dato que le indique a qué estado debe avanzar, dependiendo de ello es cómo va actuar el mips.

Para la primera etapa de carga de instrucciones, este estado espera un carácter como binario que indique la carga de instrucciones. De suceder esta acción, se asigna un nuevo estado denominado *ENABLE LOAD INSTR* que como su nombre indica habilita la carga de instrucciones proveniente del código ejecutando en nuestra terminal de usuario.

**ENABLE\_LOAD\_INSTR:** Aquí voy a recibir instrucciones y voy a ir dato completo para enviar a la memoria de instrucciones. Se irán concatenando los datos de 8 bits recibidos por la uart. Este estado se complementa con los siguientes: ***PREPARE\_INSTRUCT*** que se encarga de ir chequeando si ya se tienen los 32 bits de la instrucción para poder enviar o esperar datos desde la pc aun.

Si ya se recibieron los 4 bytes de la instrucción que estoy intentado cargar desde el código el estado ***WAIT\_DATA\_INSTR*** verifica si la instrucción corresponde a un *halt* siendo la última instrucción de carga y en este caso volvemos a un estado IDLE a la espera de un modo de ejecución o se incrementa la posición a la cual vamos a colocar la instrucción en la memoria y se regresa a este estado original descripto.