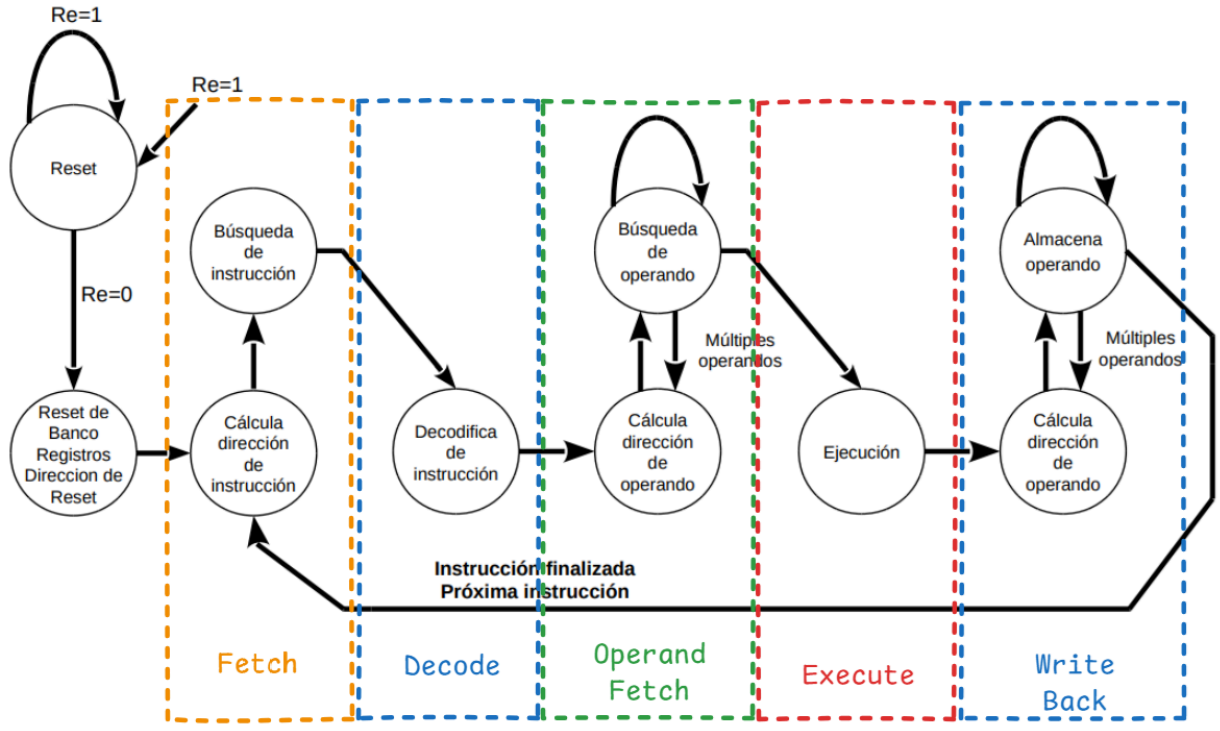
Unidad 3 - Ciclo de Instrucción e Implementación Monociclo

1. Ciclo de Instrucción

Es el proceso por el cual la computadora recupera una instrucción del programa de su memoria, determina las acciones que describe y las ejecuta. En una CPU simple, este ciclo se ejecuta secuencialmente, pero en las CPU modernas los ciclos se ejecutan de forma **concurrente**.
Para lograr la ejecución concurrente, el ciclo de instrucción se **divide** en cinco operaciones básicas (el número de divisiones y qué se hace en cada una puede variar dependiendo la arquitectura):

- **Fetch**: lectura de la instrucción, se ve la dirección de la memoria donde se encuentra la instrucción.
- **Decode**: decodificación de la instrucción, se identifica qué instrucción ejecutar.
- **Operand Fetch**: lectura de los operandos, se ve la dirección en memoria o registro donde se encuentran los operandos.
- **Execute**: ejecución de la instrucción, se realiza la operación indicada por la instrucción (lógica, aritmética, control de flujo, lectura/escritua en memoria).
- **Write Back**: escritura del resultado, se ve la dirección en dónde se debe escribir el resultado de la instrucción.



Ciclo de Instrucción

2. ISA RISC-V

El RISC-V define un ISA **base** (RV32I) de operaciones enteras que debe estar presente en cualquier implementación de RISC-V, sumado a otras extensiones opcionales. Lo más importante de este ISA es que proporciona un "esqueleto" de herramientas alrededor del cual se puede construir un ISA personalizado.

- **Modos de Operación**: RV32I soporta tres modos de operación: usuario, supervisor y máquina. El modo máquina tiene los privilegios más altos y es obligatorio. Los modos usuario, hipersupervisor y supervisor se destinan a la

aplicación y uso convencional del sistema operativo.

- **Tipos de datos**: soporta bit, nibble, byte, palabra (word) y doble palabra en formato entero.
- Operandos: proporciona una arquitectura con 32 registros de 32 bits de propósito general.
- **Conjunto de Instrucciones**: utiliza instrucciones de tres operandos registro-registro y un esquema Load y Store para mover datos entre memoria y registros. El contador de programa es un registro de uso específico que no puede ser accedido por el programador.
- **Organización de la Memoria**: el **área reservada** almacena registros de estado y configuración para los modos de operación y gestión de eventos. Los periféricos se mapean en la **memoria del procesador** y se gestionan a través de instrucciones load/store. La pila se maneja por un puntero de frame, que apunta al primer registro guardado, y un puntero de pila, que apunta al "top" de la pila. Los registros de **control y estado** identifican características del procesador y ayudan a medir su rendimiento.
- **Gestión de Eventos**: RISC-V clasifica los eventos en **Excepciones** e **Interrupciones**. Las excepciones ocurren durante la ejecución de una instrucción, y pueden ser Fallas de Acceso, Dirección Desalineada, Breakpoint (ebreak), Llamadas al Entorno (ecall) o Instrucción Ilegal (opcode invalido). Las interrupciones son asíncronas al flujo de instrucciones y pueden deberse a Interrupciones de Software (suele ser disparada por otro hilo), Interrupciones de Temporizador o Interrupciones Externas (disparadas por un controlador de interrupciones a nivel plataforma).
- **Modos de Direccionamiento**: puede emular modos de direccionamiento del x86-32, como el registro-indirecto, dejando el valor inmediato en cero. También obtiene los beneficios de las instrucciones push y pop utilizando simplemente un registro como stack pointer. Dispone de los siguientes modos de direccionamiento:
  - Directo a registro.
  - Indirecto a registro con desplazamiento.
  - Relativo a PC con desplazamiento.
  - Inmediato.
  - Pseudodirecto.
  - Implícito.

Acá dice que, a diferencia de ARM-32 y MIPS-32, no se exige que los datos estén alineados en memoria, los loads y stores pueden acceder a memoria desalineada. Pero antes menciona que hay excepciones por dirección desalineada. Ni idea

2.1. Codificación

Las instrucciones se codifican utilizando el código de operación (opcode), las fuentes de datos (rs1, rs2, imm), el destino (rd) y la función a realizar (funct3 y funct7). Se codifican en seis tipos, de acuerdo con los operandos que utiliza y el tipo de operación que realiza.

3. Implementación de RV32I

La implementación es la forma en que los procesadores implementan (duh) la **arquitectura** a través de circuitos digitales. En el caso de la arquitectura RISC-V, los diseñadores de microprocesadores tienen la libertad crear cualquier diseño dentro de las definiciones de la arquitectura. Los ***requisitos* que deben cumplir las implementaciones son:

- Implementar todas las instrucciones del ISA RV32I.
- Modo máquina obligatorio, mientras que los modos supervisor y usuario son opcionales.

La metodología de gestión de interrupciones es definida por el diseñador, y la implementación de las extensiones es opcional.

Para esta materia, se definirán las siguientes **restricciones** a la implementación:

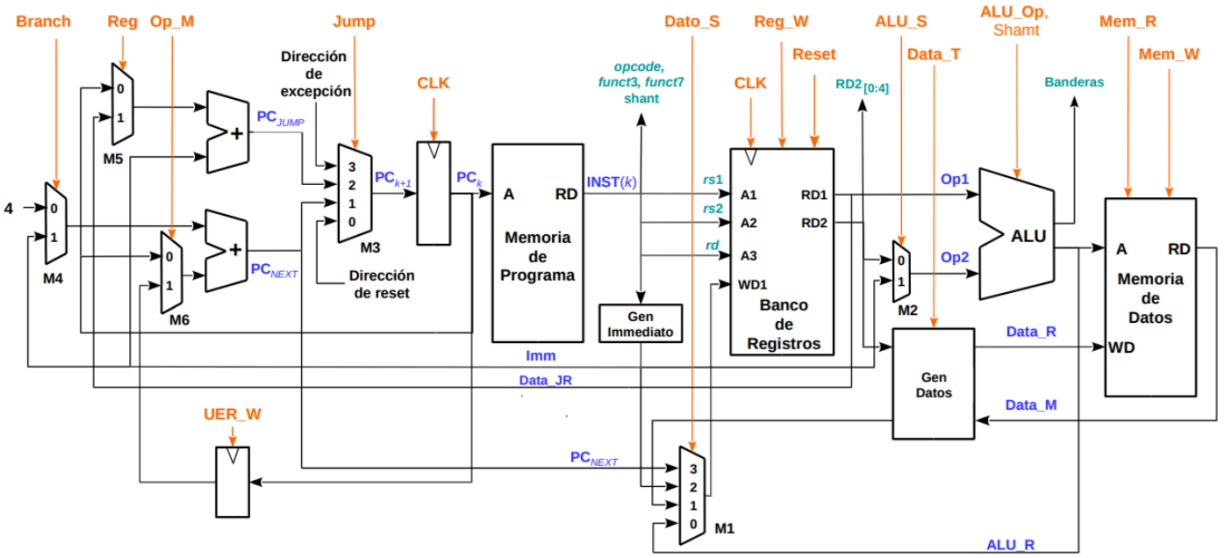
- El único modo de operación soportado por el procesador será el modo máquina.
- Se omitirán las instrucciones de operación de los registros de control y estados
- El procesador utilizará solo 32 registros de propósitos generales con arquitectura Load/Store y los modos de direccionamiento soportados por RV32I.
- Los eventos internos se gestionarán con excepciones definidas en el RV32I sin las excepciones de Llamada al entorno y Dirección Desalineada. Las direcciones de las rutinas de atención serán predeterminadas.
- Se utilizará una microarquitectura monociclo, empleando una configuración Harvard.

3.1. Componentes

Se utilizarán los siguientes componentes para la implementación:

- **Contador del programa**: registro de 32 bits que guarda la dirección de la instrucción que se está ejecutando.
  - Entradas: dirección de la próxima instrucción a ejecutar (PC') y el reloj del sistema (CLK) que indica cuándo actualizar la salida.
  - Salidas: dirección de la instrucción que se ejecuta.
- **Memoria de programa**: región de la memoria donde se aloja el programa que se está ejecutando. Supondremos que todas las señales son de 32 bits de ancho.
  - Entradas: dirección de memoria accedida (A)
  - Salidas: información hallada en la dirección A (RD).
- **Sumador**: circuito combinacional que calcula la suma de los dos operandos de 32 bits de entrada.
- **Unidad de Generación de Inmediato**: genera el campo de dato inmediato, de 12 o 20 bits, para instrucciones aritméticas, lógicas, carga, almacenamiento y saltos. Reorganiza la información del campo inmediato de acuerdo al código de operación, reordenando bits, completando con 0 y desplazando el dato resultante.
- **Unidad de Generación de Datos**: reorganiza los datos recibidos desde los registros (almacenamiento) y memoria (carga) para obtener el formato (8, 16 o 32 bits) y la posición (lower o upper) correspondiente a la instrucción, completando el resto de posiciones con 0's.
- **Banco de Registros**: contiene todos los registros, con dos puertos de lectura y uno de escritura.
  - Entradas: las entradas de lectura (A1 y A2) indican los registros a leer en las salidas (RD1 y RD2). La señal de control de escritura (WE) indica explícitamente la escritura del dato de entrada (WD) en la dirección de escritura (A3) al momento de un flaco de reloj del sistema (CLK).
  - Salidas: las salidas de lectura (RD1 y RD2) de 32 bits llevan los datos indicados por las direcciones de lectura (A1 y A2).
- **Memoria de Datos**: región de la memoria donde se alojan los datos utilizados por el programa (pila, dinámicos y estáticos).
  - Entradas: la dirección de memoria accedida (A) de 32 bits, los datos a escribir (WD) de 32 bits, la señal de escritura (WE) y el reloj del sistema (CLK).
  - Salidas: la información (RD) de 32 bits que se halla en la dirección de memoria accedida (A).
- **Unidad Aritmético Lógica** (ALU): calcula operaciones aritméticas (suma, resta, multiplicación, etc.), lógicas (and, or, not, xor) y comparaciones entre los operandos (para saltos condicionales).
  - Entradas: los operandos (OP1 y OP2) de 32 bits y la operación a realizar (ALU_Op) de 3 bits.
  - Salidas: el resultado (ALU_R) de 32 bits y las banderas de condición (F) de 3 bits.

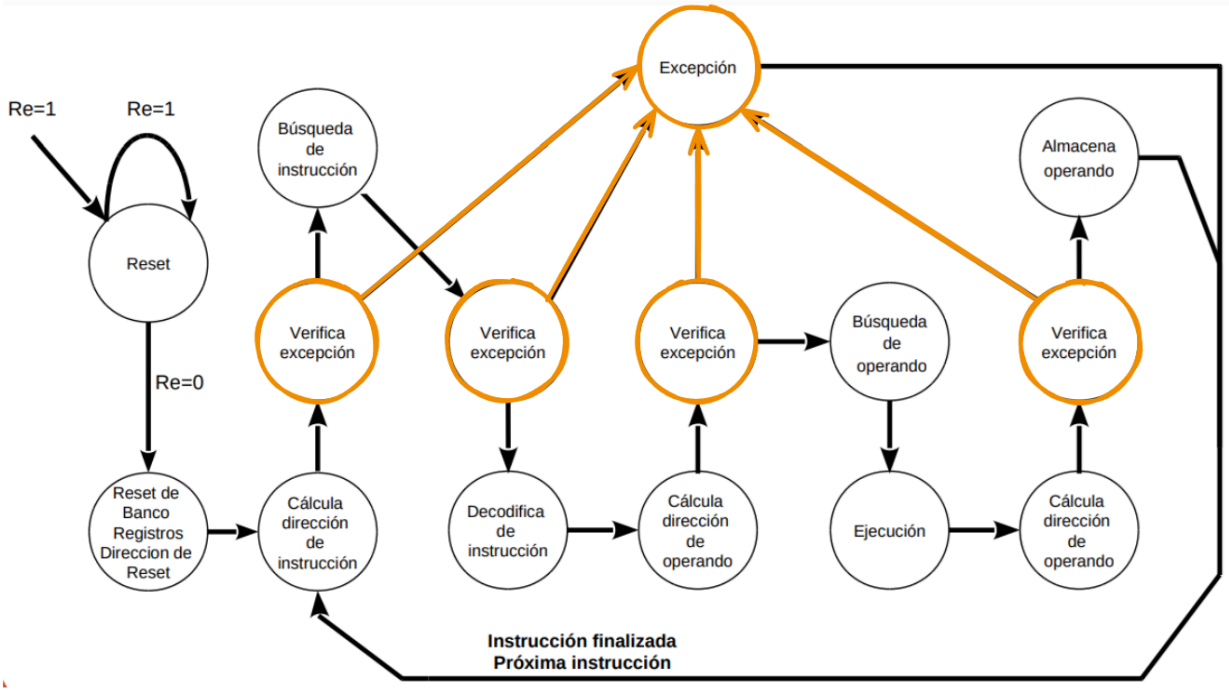
- **Unidad de Control**: circuito que controla el flujo de datos en el procesador, generando señales para operar el camino de datos. Es una máquina de estados finitos cuyas tareas son leer, decodificar, ejecutar la instrucción y almacenar los resultados. Los subcomponentes de esta son:
  - Control Principal: genera las señales de los bloques que gestionan el flujo de información a través del datapath.
  - Decodificador de Salto: genera las señales de los bloques que implementan el fetch (saltos, interrupciones, excepciones y reset).
  - Decodificador de Carga: genera las señales de los bloques que implementan la carga y almacenamiento en memoria.
  - Decodificador de Operaciones: genera las señales para operar la ALU.
- **Multiplexores**: enrutan las fuentes de datos de entrada al banco de registros y del operando de la ALU.
  - Entradas: las fuentes de datos (pueden ser el generador de datos, la salida de la ALU, el generador de inmediato u otro) además de la señal de control que indica la fuente a enrutar.
  - Salidas: como ya sabemos, la salida del multiplexor será la fuente indicada por la señal de control.



Excepciones en el ciclo de instrucción

3.2. Excepciones

Como ya vimos, una excepción es un evento no planificado dentro del procesador que altera la ejecución del programa. Pueden producirse por un overflow, una instrucción desconocida, entre otras. En ese momento, el control es transferido a un programa encargado de resolver el problema (gestor de excepciones) y luego de resuelto el control es devuelto al programa y continúa como si no hubiera sucedido. Para esto es importante que el gestor de excepciones pueda mantener el estado del programa al momento del evento.
El ciclo de ejecución es monitoreado para generar las excepciones correspondientes.



Excepciones en el ciclo de instrucción