Pipeline o segmentación encauzada: recordatorio

- Pipeline:
  - Cálculos **solapados**
  - Hardware **segmentado** (cauce o pipeline)
    - ✓ Dividido en **etapas** o segmentos
    - ✓ ***Latches* o registros separan** las **etapas** y **evitan sobreescrituras**
- **Latencia** de una tarea: ligeramente mayor que en un hardware no encauzado (retardos *latches*, tiempo de desviación [*skew*], esperas de las etapas más rápidas a las más lentas)
- **Productividad aumenta (ventaja fundamental)**
  - **Ganancia ideal = Nº. de etapas**. Para **aproximarnos** a ello:
    - ✓ Retardos *latches* y *skew* despreciables, minimizar las esperas

Clasificaciones de los cauces

- Aritméticos / de instrucciones
  - El cauce de instrucciones es una CPU encauzada y puede contener uno o varios cauces aritméticos
- Monofuncionales /multifuncionales
  - Los cauces de instrucciones son multifuncionales
  - Los aritméticos pueden ser de ambos tipos
- Síncronos / asíncronos
  - En los síncronos una misma señal de reloj activa todos los *latches* a la vez

Tareas típicas en una CPU encauzada

- Traída (fetch) de la instrucción y cálculo de siguiente valor del PC
- Descodificación
- Cálculo de dirección efectiva
- Acceso a memoria de datos
- Ejecución (ALU)
- Escribir resultados

- Diversos modos de organizar las tareas en los procesadores
  - A veces, una etapa tiene capacidad para varias de estas tareas
  - Otras, una tarea precisa varias etapas para completarse

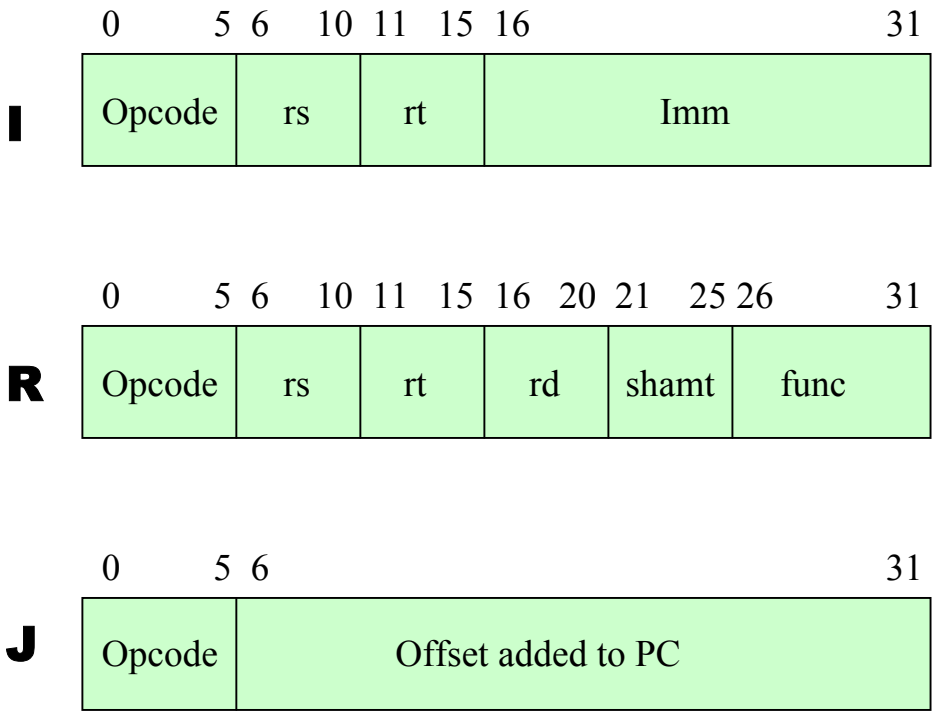
RISC: un modo eficiente de organizar una CPU encauzada..

- Características de un RISC que hacen eficiente su CPU encauzada
  - **Cachés separadas** para datos e instrucciones
    - ✓ Con el uso de caché se intenta evitar que la memoria sea el cuello de botella
    - ✓ Tener cachés separadas evita colisiones entre la etapa de traída y la de acceso a memoria de datos
  - Cálculo del siguiente valor del PC sencillo porque todas las instrucciones son de igual longitud
  - **Descodificación eficiente** debido a formatos sencillos

RISC: un modo eficiente de organizar una CPU encauzada

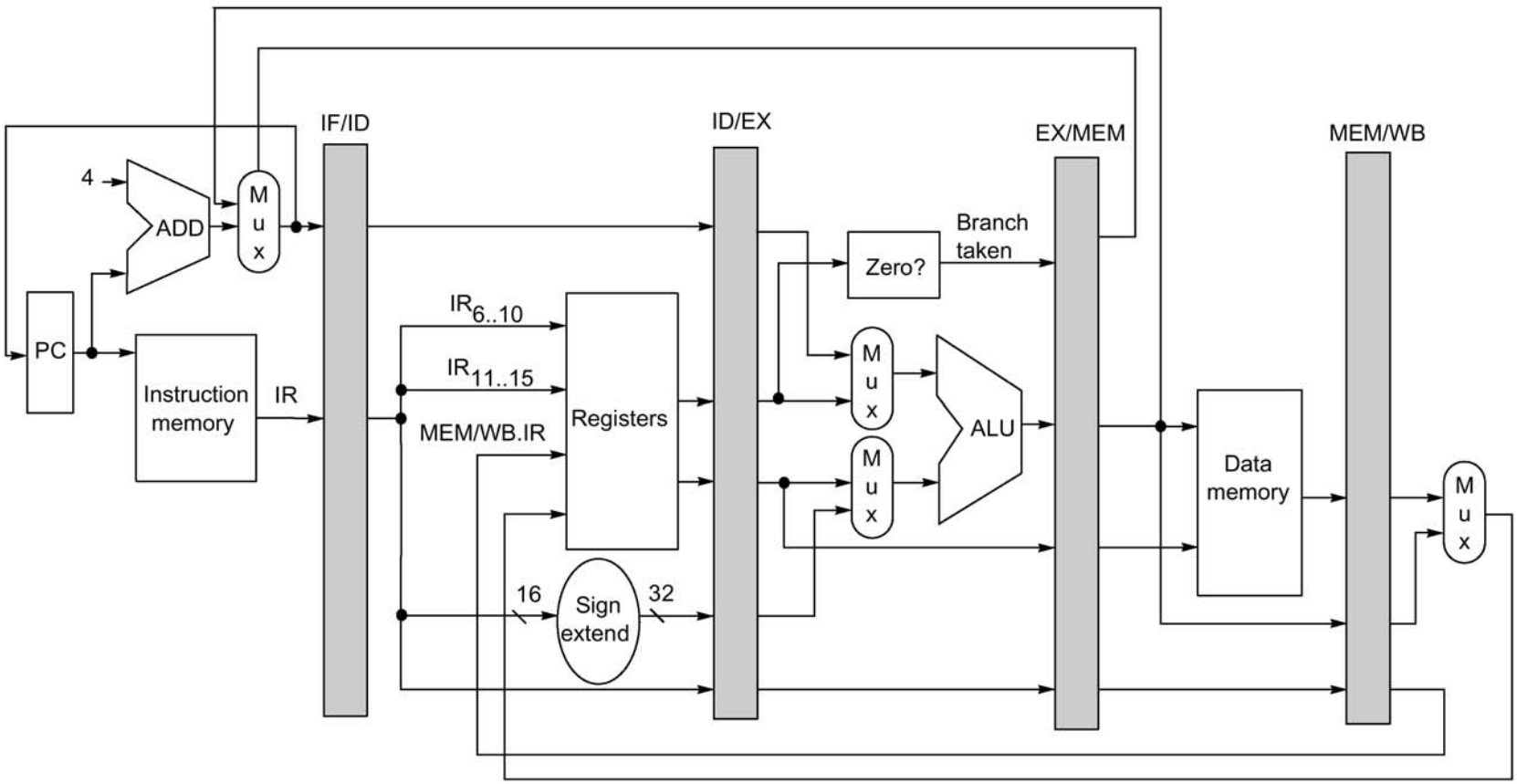
- El **cálculo de la dirección efectiva y** la **ejecución** los realiza **una misma etapa**
  - ✓ Si la instrucción es **aritmético-lógica no** interviene **dirección efectiva** alguna porque un RISC es una máquina de tipo registro-registro
  - ✓ Si es de **acceso a memoria o de salto o bifurcación** la **ALU** se utiliza **para calcular** la **dirección efectiva** (sumando un valor inmediato con el contenido de un registro)

Formatos de instrucciones de un RISC típico: MIPS



- Formato tipo I
  - Loads y Stores
  - ALU reg-inm
  - Branches y Jump (and link) register
- Formato tipo R
  - ALU reg-reg y otras
- Formato tipo J
  - Jump (and link)
  - Trap y RFE (return from exception)

MIPS: ruta de datos encauzada



Acciones que se realizan en cada etapa..

- **IF** (*Instruction Fetch*)
  - Traída de la instrucción

    IF/ID.IR ← Mem[PC]
  - Actualización del PC e IF/ID.NPC

    If [(EX/MEM.opcode == branch) & EX/MEM.cond] **then** PC, IF/ID.NPC ← EX/MEM.ALUOutput **else** PC, IF/ID.NPC ← PC+4

Acciones que se realizan en cada etapa..

- **ID** (*Instruction Decode*)
  - **Descodificar** la instrucción (generar señales de control que, según el tipo de instrucción, han de activarse en otras etapas)
  - **Leer registros**

    ID/EX.A ← Regs [IF/ID.IR[rs]];

    ID/EX.B ← Regs [IF/ID.IR[rt]]
  - Pasar información al siguiente latch (ID/EX)

    ID/EX.Imm ← $(IF/ID.IR_{16})^{16}$##IF/ID.IR[Imm];

    ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR

Acciones que se realizan en cada etapa..

- **EX** (*Execution/effective address*)
  - Si la instrucción es de **bifurcación**

    EX/MEM.ALUOutput ← ID/EX.NPC+ (ID/EX.Imm << 2); /* "X<<2" = X desplazado 2 bits a izda */

    EX/MEM.cond ← (ID/EX.A == 0)
  - Si la instrucción es de **carga o almacenamiento**

    EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm;

    EX/MEM.IR ← ID/EX.IR; EX/MEM.B ← ID/EX.B;

Acciones que se realizan en cada etapa..

- En la etapa EX, si la instrucción es **aritmético-lógica de tipo R**

  EX/MEM.ALUOutput ← ID/EX.A ***func*** ID/EX.B

  EX/MEM.IR ← ID/EX.IR

- Si es **aritmético-lógica de tipo I**

  EX/MEM.ALUOutput ← ID/EX.A ***op*** ID/EX.Imm;

  EX/MEM.IR ← ID/EX.IR

Acciones que se realizan en cada etapa..

- **MEM** (*Memory access*)
  - Si la instrucción es de **carga**

    MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]

    MEM/WB.IR ← EX/MEM.IR;
  - Si la instrucción es de **almacenamiento**

    Mem[EX/MEM.ALUOutput] ← EX/MEM.B

    MEM/WB.IR ← EX/MEM.IR;
  - Si es **aritmético-lógica**, pasar datos

    MEM/WB.IR ← EX/MEM.IR;

    MEM/WB.ALUOutput ← EX/MEM.ALUOutput

Acciones que se realizan en cada etapa

- **WB** (*Write Back*)
  - Si la instrucción es de **carga**

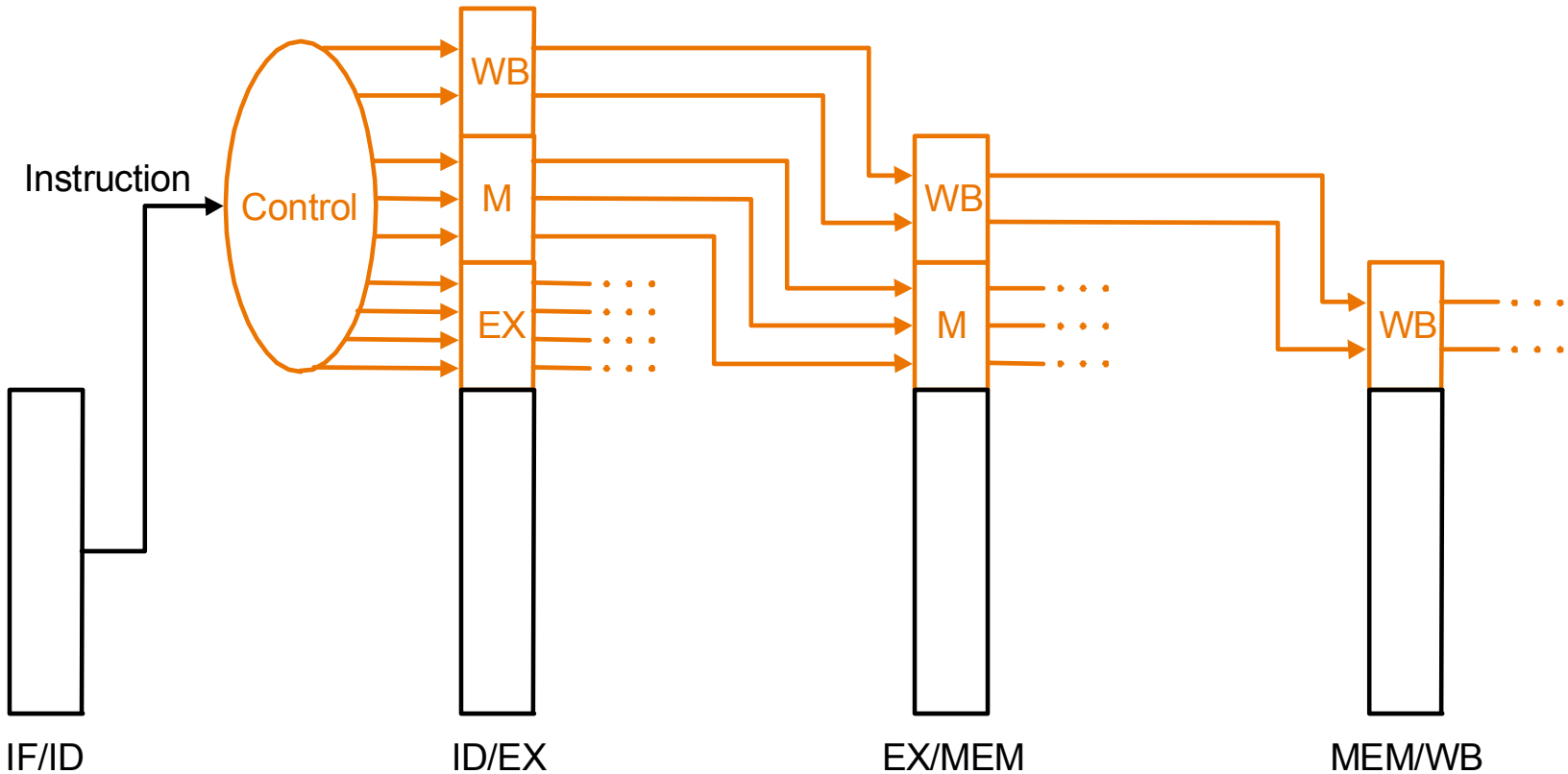
    Regs[MEM/WB.IR[rt]] ← MEM/WB.LMD
  - Si es **aritmético-lógica de tipo R**

    Regs[MEM/WB.IR[rd]] ← MEM/WB.ALUOutput
  - Si es **aritmético-lógica de tipo I**

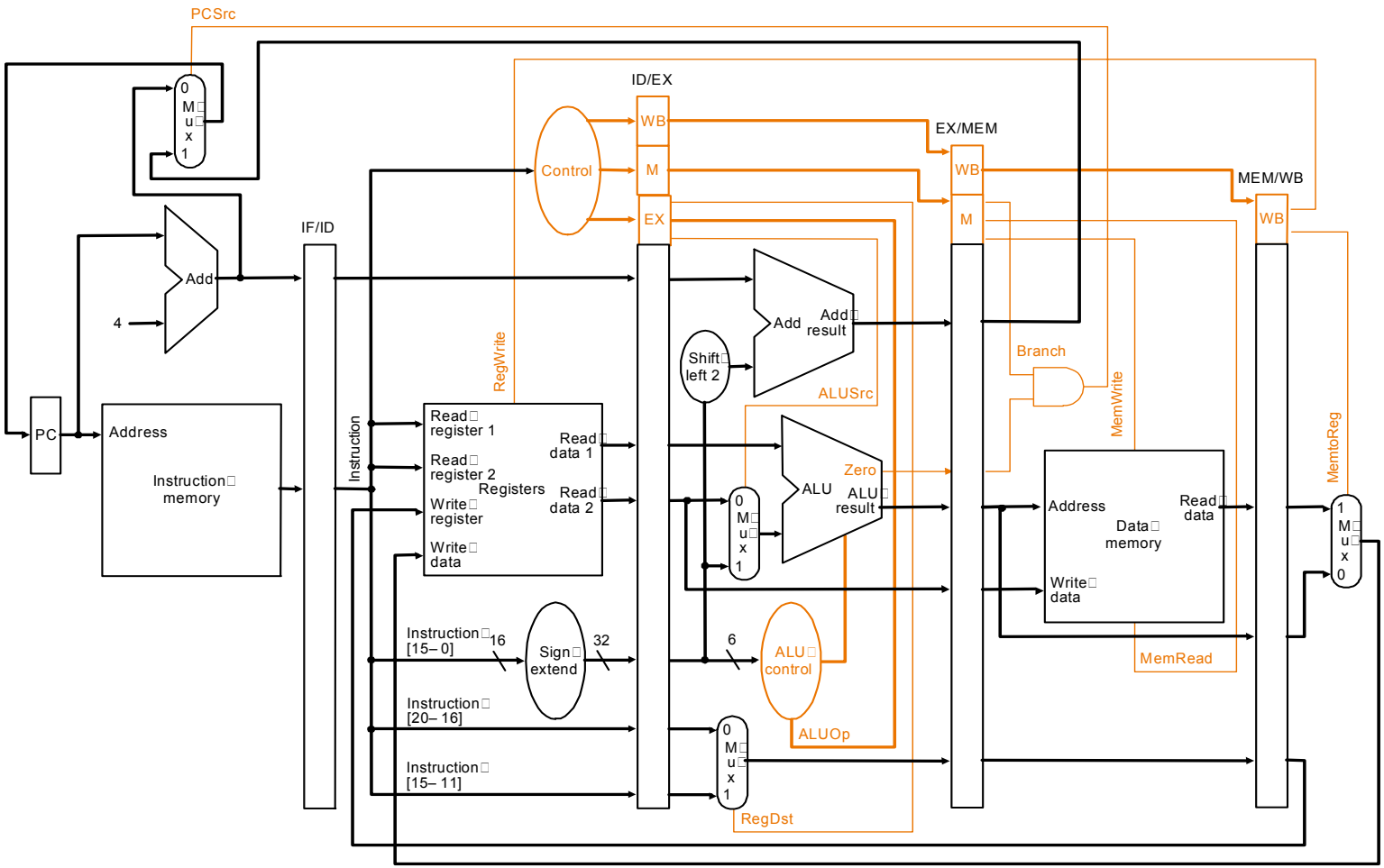
    Regs[MEM/WB.IR[rt]] ← MEM/WB. ALUOutput

(Téngase en cuenta que hay instrucciones no contempladas en las transparencias 8 a 13)

Control en procesadores encauzdos



Ejemplo de cauce de instrucciones y su control



Riesgos (*hazards*)

- Situaciones que pueden obligar a **detener** alguna instrucción para **evitar** un **incorrecto funcionamiento** del programa. Pueden ser
  - **Estructurales** (colisiones en el acceso a un recurso)
  - **Dependencias de datos**
    - ✓**RAW** (leer después de escribir)
    - ✓**WAR** (escribir después de leer)
    - ✓**WAW** (escribir después de escribir)
  - **Dependencias de control**: producidas por saltos y bifurcaciones.

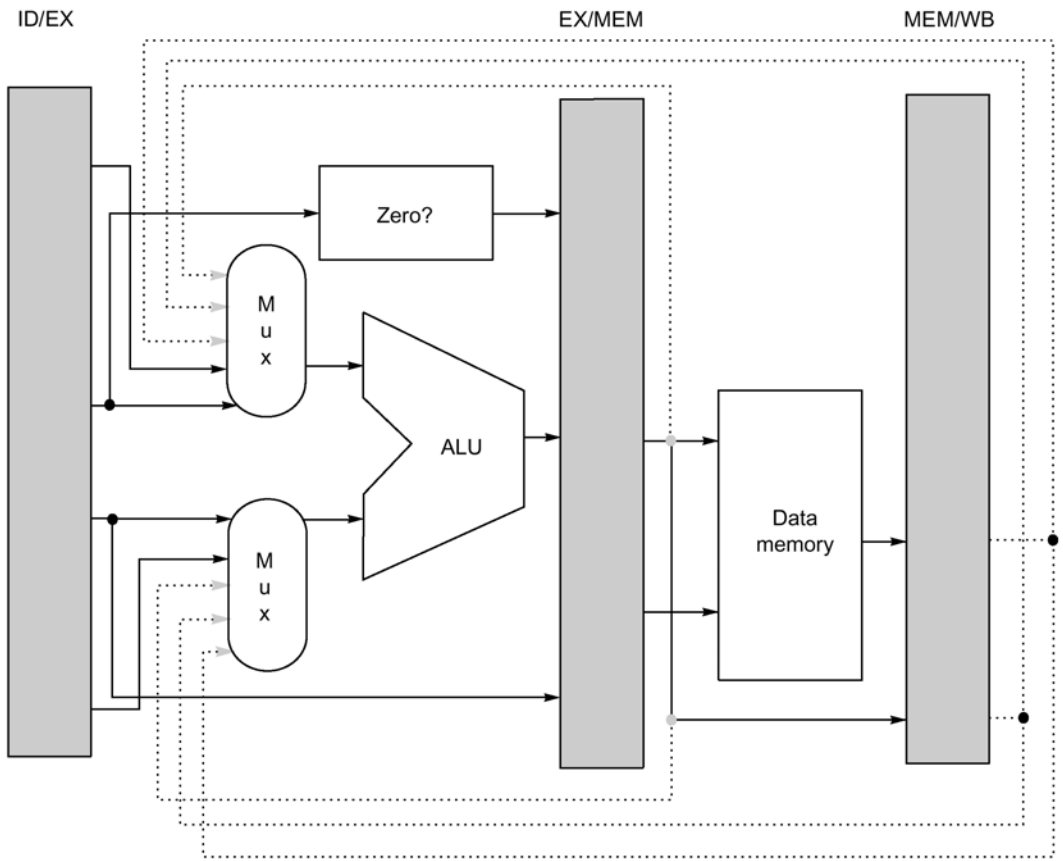
Detección y resolución de riesgos

- Con **planificación estática: orden** de ejecución de las instrucciones **determinado** antes de ejecutarse el programa
  - Puede realizarla:
    - ✓ El **hardware**: **deteniendo el cauce** (es decir una instrucción y las que le siguen) cuando un riesgo lo haga necesario
      - Las paradas pueden **localizarse**:
        - Siempre en la misma etapa (la de **descodificación**)
        - En diversas etapas (más eficiencia y coste)
    - ✓ El **compilador**: reordenando instrucciones y colocando instrucciones NOP donde sea conveniente
- Mediante **planificación dinámica:** el **orden** de ejecución de las instrucciones se determina en tiempo de ejecución
  - Se basa pues en **hardware** pero **el compilador puede ayudar** a mejorarla

Algunas técnicas básicas para tratar riesgos

- **Colisiones**
  - En **banco de registros** (en MIPS entre ID y WB)
    - ✓ Solución en MIPS: WB escribe en la 1ª. mitad del ciclo e ID lee en la 2ª.
  - En general, se concede el recurso a la instrucción más avanzada en el cauce
- **Dependencias RAW** (las más frecuentes)
  - La técnica de **anticipación** (*forwarding)* o **cortocircuito** (*short-circuiting*) reduce el número de ciclos perdidos
  - Cuando la 2ª. instrucción necesita un valor calculado por la 1ª., un **camino de anticipación** le permite tomarlo del *latch* donde se encuentre

Caminos de anticipación a las entradas de la ALU

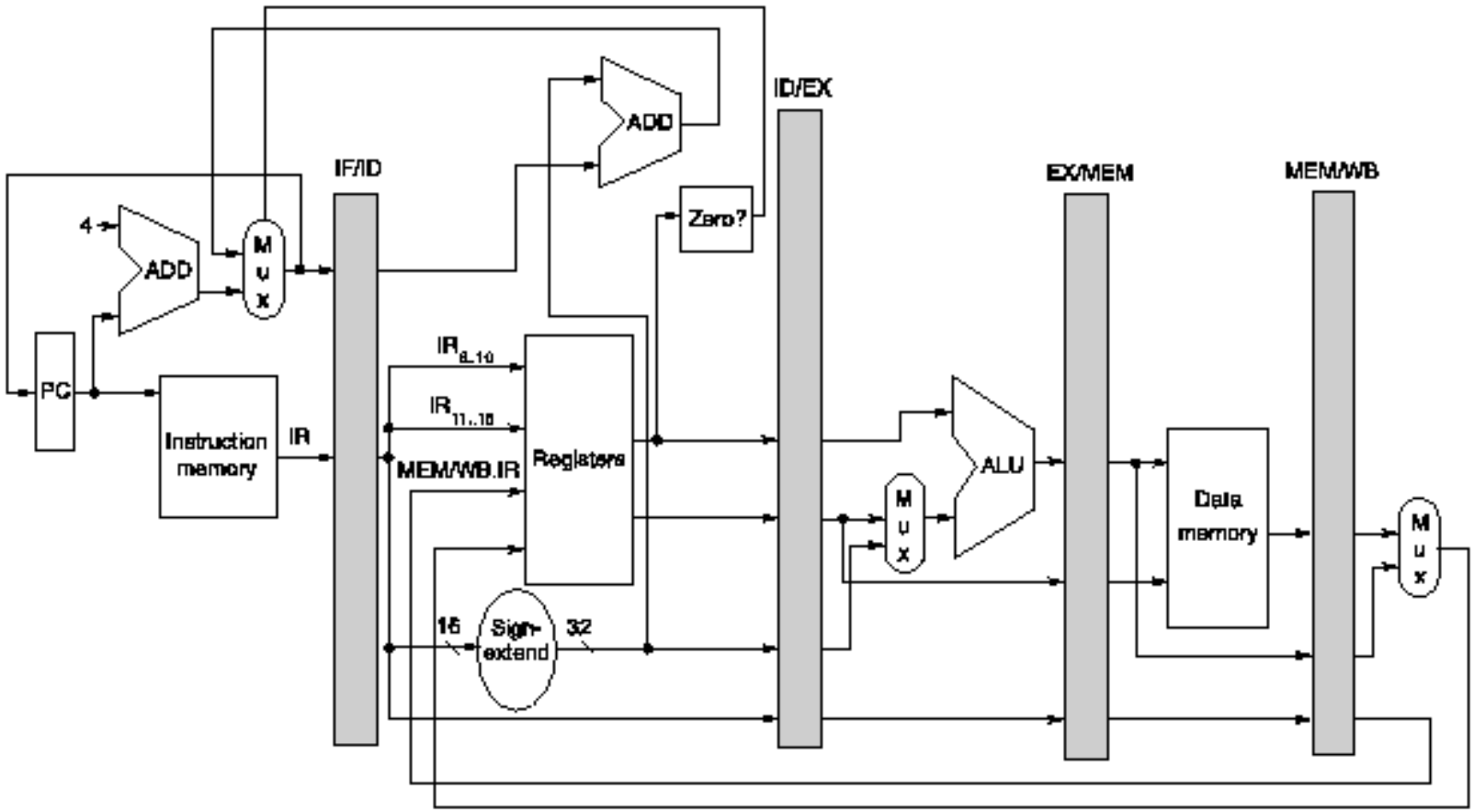


- Además de los caminos de anticipación, se requieren ampliar los multiplexores para que tengan más entradas
- Pero el aumento de complejidad es mayor en el control

Tratamiento de los riesgos de control

- Problema: las **instrucciones** que están en el cauce **tras un salto o bifurcación** (y las prebuscadas) hay que **cancelarlas** si se salta =>
  - Mejor **conocer cuanto antes la condición y la dirección de salto**
  - Conviene **evitar** que las instrucciones que entran después del salto o bifurcación **modifiquen el estado** del procesador
    - ✓Si lo modifican, hay que proveer medios para que esas modificaciones sean reversibles

MIPS: cauce mejorado para resolver antes los saltos y bifurcaciones



Acciones en la etapa IF del cauce MIPS mejorado anterior

- **Traída** de la instrucción: IF/ID.IR ← Mem[PC]
- **Actualización del PC** e IF/ID.NPC
  - **If** (IF/ID.IR[opcode] == branch) and (Regs[IF/ID.IR[rs]] *op* 0)

    then PC, IF/ID.NPC ←

    IF/ID.NPC + $(IF/ID.IR_{16})^{16}$## IF/ID.IR[Imm]##00

    else PC, IF/ID.NPC ← PC+4

Acciones en las demás etapas del cauce MIPS mejorado anterior

- ID: el único cambio es que no hay que pasar NPC al *latch* siguiente
- EX:
  - Si la instrucción es de bifurcación
    - ✓Nada que hacer: la bifurcación terminó en ID =>
      - Sólo **un hueco de retardo** en saltos y bifurcaciones
  - No hay más cambios en EX
- Ningún cambio en MEM o WB

(No se olvide que hay instrucciones no contempladas en este estudio)

Soluciones estáticas de riesgos de control

- **Predecir** siempre
  - **Salto no efectivo**
  - **Salto efectivo**
    - ✓ Sólo posible si se conoce la dirección de salto antes que la condición de la bifurcación
  - Si predicción errónea, recuperar el flujo correcto
- **Saltos y bifurcaciones retardados**
  - **La instrucción** o instrucciones **en** el/los ***hueco(s) de retardo*** **se ejecuta**n independientemente de que haya o no que saltar
  - El **compilador** coloca instrucciones del programa en los huecos de retardo. Donde las dependencias se lo impiden coloca instrucciones NOP